**Question 1: What is the difference between K-Means and Hierarchical Clustering? Provide a use case for each.**
**Answer:**

## Difference between K-Means and Hierarchical Clustering

- **Clustering approach**
    - K-Means: Partition-based clustering method
    - Hierarchical: Tree-based clustering method
- **Number of clusters**
    - K-Means: Must be specified in advance (K)
    - Hierarchical: Not required beforehand
- **Working method**
    - K-Means: Iteratively assigns points to the nearest centroid
    - Hierarchical: Builds clusters by merging or splitting data points
- **Output**
    - K-Means: Flat set of clusters
    - Hierarchical: Dendrogram (hierarchical tree)
- **Scalability**
    - K-Means: Fast and suitable for large datasets
    - Hierarchical: Slower and computationally expensive for large datasets
- **Flexibility**
    - K-Means: Less flexible once K is fixed
    - Hierarchical: More flexible due to multiple clustering levels

## Use Cases

- **K-Means Use Case**
    - Customer segmentation when the number of groups is known
- **Hierarchical Clustering Use Case**
    - Gene expression analysis to explore data relationships using dendrograms

**Question 2: Explain the purpose of the Silhouette Score in evaluating clustering algorithms.**
**Answer:**
The Silhouette Score is used to evaluate the quality of clustering results.

It measures how similar a data point is to its own cluster compared to other clusters.

The score helps determine how well clusters are separated and cohesive.

Silhouette Score values range from −1 to +1.

A value close to +1 indicates well-separated and compact clusters.

A value around 0 indicates overlapping clusters.

A negative value indicates that data points may be assigned to the wrong cluster.

It is commonly used to compare different clustering algorithms or to choose the optimal number of clusters (K).

**Question 3: What are the core parameters of DBSCAN, and how do they influence the clustering process?**
**Answer:**

## Core Parameters of DBSCAN and Their Influence

- **eps (Epsilon)**
    - Defines the maximum distance between two points to be considered neighbors.
    - A small eps results in many small clusters and more noise points.

    - A large eps may merge nearby clusters into one.

- **`min_samples`**

    - Specifies the minimum number of points required to form a dense region (cluster).

    - A higher value makes clusters more strict and increases noise detection.

    - A lower value allows formation of smaller clusters.

- **Influence on Clustering Process**

    - Together, `eps` and `min_samples` determine core points, border points, and noise points.

    - Proper tuning helps DBSCAN identify arbitrarily shaped clusters and outliers effectively.

**Question 4: Why is feature scaling important when applying clustering algorithms like K-Means and DBSCAN?**
**Answer:**
Feature scaling is crucial when applying clustering algorithms like K-Means and DBSCAN because:

**K-Means:**

- Distance-based: K-Means relies on Euclidean distance between points.
- Scale sensitivity: Features with large ranges dominate distance calculations.
- Inconsistent clustering: Without scaling, clusters may be skewed towards features with large scales.

**DBSCAN:**

- Density-based: DBSCAN uses epsilon (ε) and min_samples to define density.
- Scale-dependent: ε is sensitive to feature scales; scaling ensures consistent density calculations.
- Inconsistent clustering: Without scaling, clusters may not be identified correctly.

**Why scaling helps:**

- Equal weightage: Features contribute equally to distance calculations.
- Improved clustering: Algorithms identify meaningful patterns and structures.
- Robustness: Scaling reduces impact of outliers and noise.

**Common scaling techniques**:

- Standardization (Z-score): (x - μ) / σ
- Normalization (Min-Max Scaling): (x - min) / (max - min)

**Example (Python):**

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import numpy as np

# Sample data
X = np.array([[1, 100], [2, 200], [3, 300]])

# Scale data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply K-Means
kmeans = KMeans(n_clusters=2)
kmeans.fit(X_scaled)
```

**Question 5: What is the Elbow Method in K-Means clustering and how does it help determine the optimal number of clusters?**
**Answer:**
The Elbow Method is a popular technique used to determine the optimal number of clusters (K) in K-Means clustering. Here's how it works:

**What is the Elbow Method?**

- The Elbow Method involves plotting the Within-Cluster Sum of Squares (WCSS) or distortion score against different values of K.
- WCSS measures the sum of squared distances between each point and its assigned centroid.

**How does it help?**

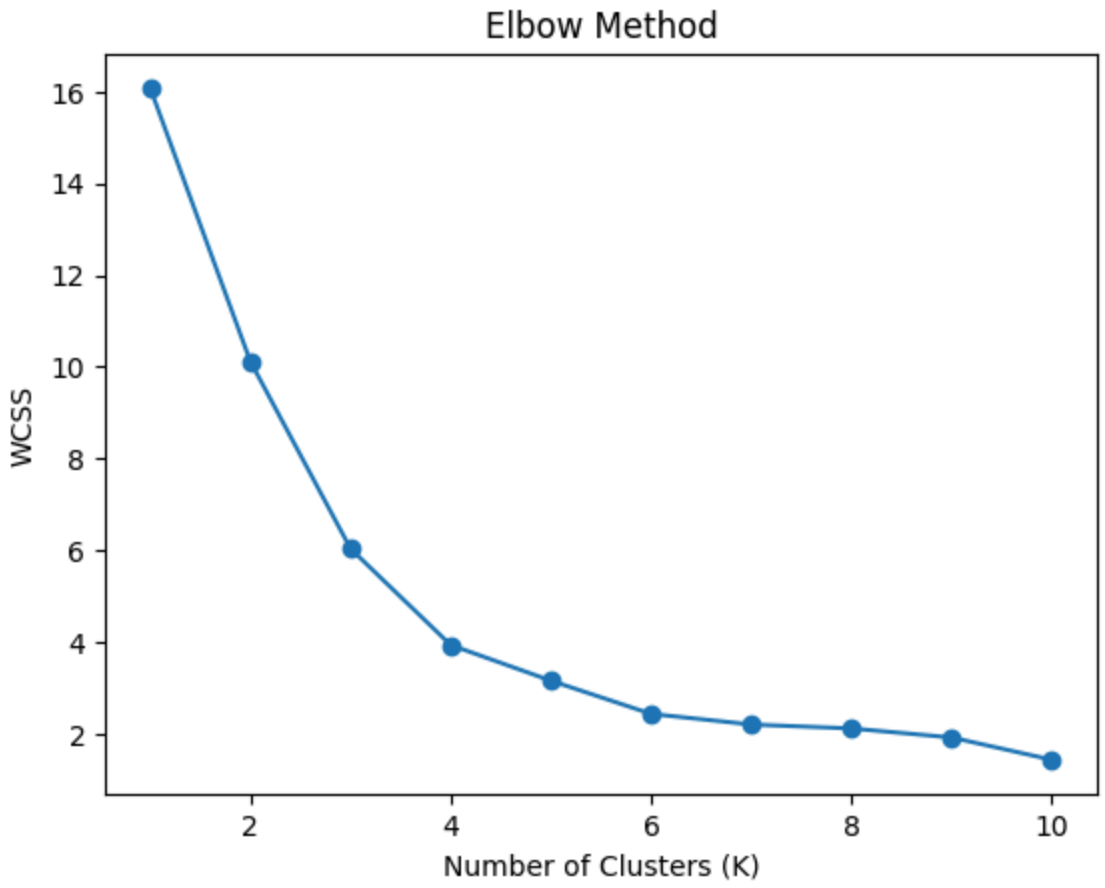
- As K increases, WCSS decreases, indicating that points are closer to their centroids.
- The plot typically shows a sharp decrease in WCSS (steep slope) followed by a gradual decrease (flat slope).
- The point where the slope changes abruptly is called the "elbow point", indicating the optimal K.

**Why is it useful?**

- The Elbow Method provides a visual cue to identify the optimal K.
- It balances model complexity (more clusters) and fit (lower WCSS).

**Here's a Python example using matplotlib:**

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np

# Generate sample data
np.random.seed(0)
X = np.random.rand(100, 2)

# Calculate WCSS for different K values
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot WCSS vs K
plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS")
plt.title("Elbow Method")
plt.show()
```

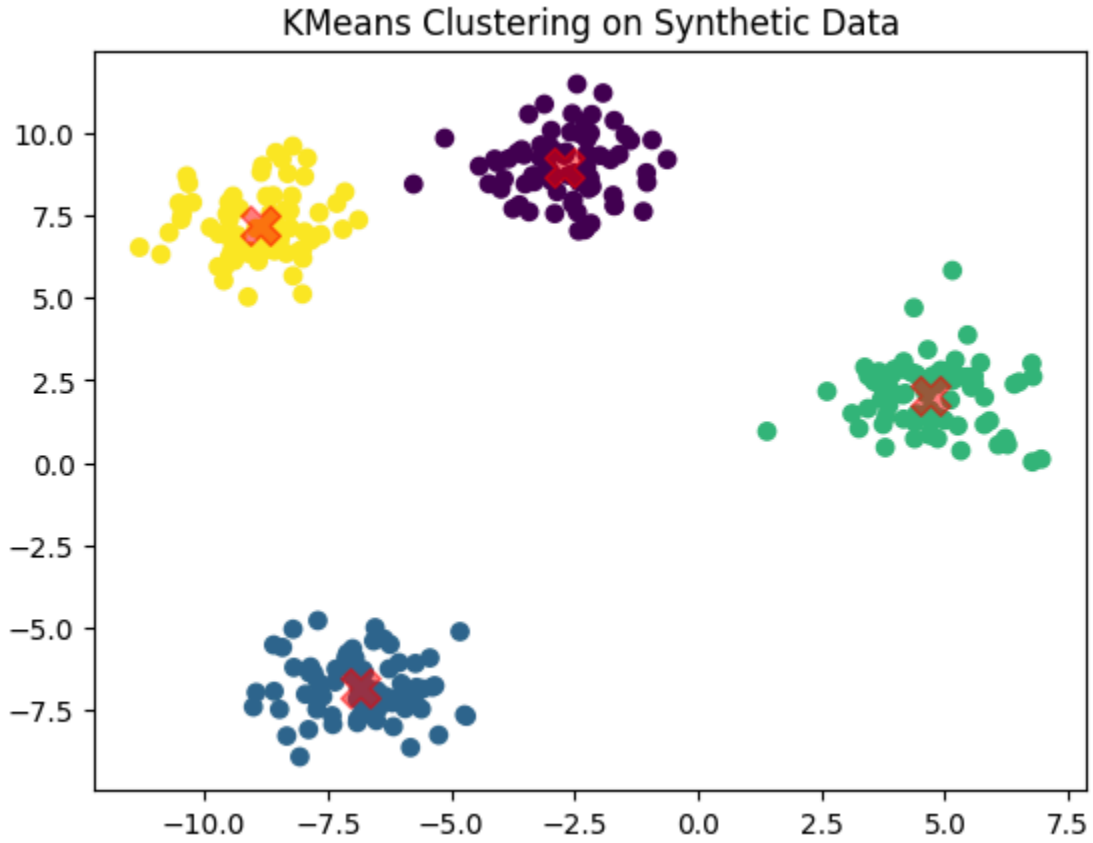



**Question 6: Generate synthetic data using make_blobs(n_samples=300, centers=4), apply KMeans clustering, and visualize the results with cluster centers. (Include your Python code and output in the code box below.)**
**Answer:**
**Here's how you can generate synthetic data using make_blobs, apply KMeans clustering, and visualize the results:**

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import numpy as np

# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=4, random_state=42)
```

```
kmeans.fit(X)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Visualize results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.5, marker='X')
plt.title("KMeans Clustering on Synthetic Data")
plt.show()
```

**Output:**




**Question 7: Load the Wine dataset, apply StandardScaler , and then train a DBSCAN model. Print the number of clusters found (excluding noise). (Include your Python code and output in the code box below.)**
**Answer:**
**Here's how you can load the Wine dataset, apply StandardScaler, and train a DBSCAN model:**

```
from sklearn.datasets import load_wine
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

# Load Wine dataset
wine = load_wine()
X = wine.data

# Apply StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X_scaled)

# Get cluster labels
labels = dbscan.labels_

# Print number of clusters (excluding noise)
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print("Number of clusters:", n_clusters)
```

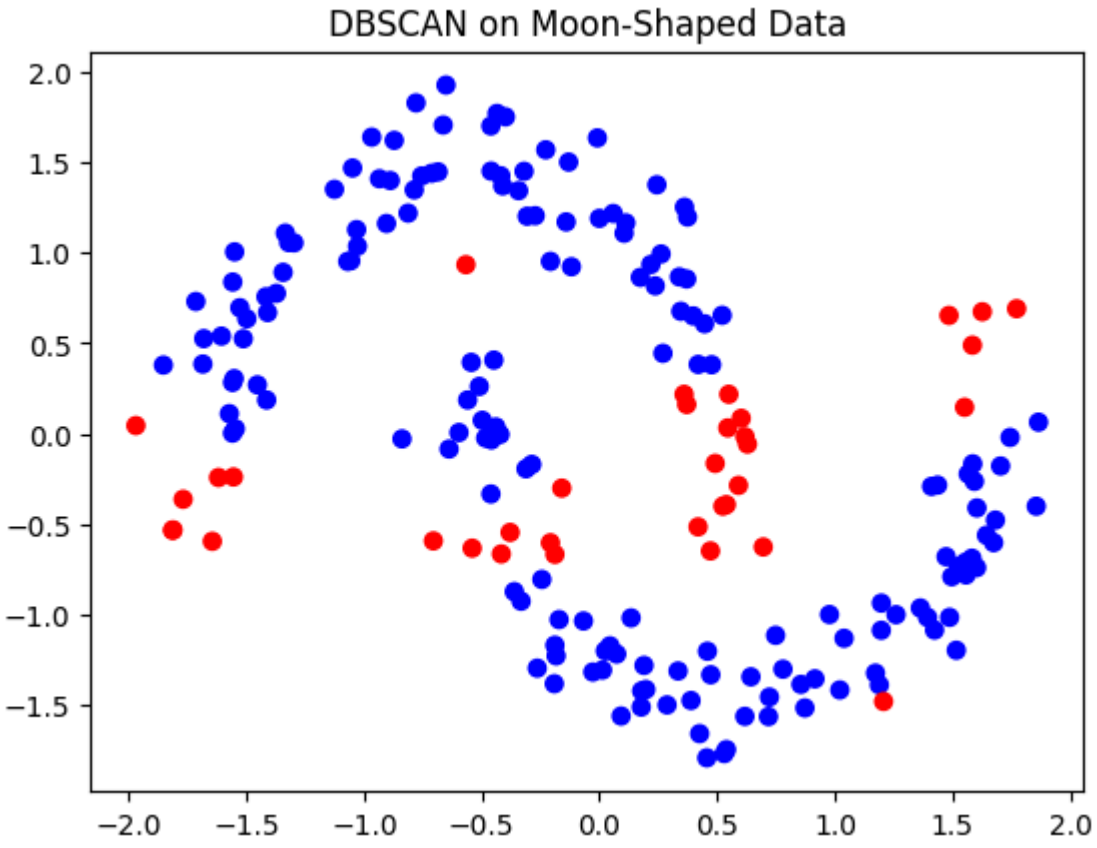
**Output:**
Number of clusters: 0

**Question 8: Generate moon-shaped synthetic data using make_moons(n_samples=200, noise=0.1), apply DBSCAN, and highlight the outliers in the plot. (Include your Python code and output in the code box below.)**
**Answer:**
Here's how you can generate moon-shaped synthetic data, apply DBSCAN, and highlight outliers:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

# Generate moon-shaped data
X, y = make_moons(n_samples=200, noise=0.1, random_state=42)

# Apply StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=10)
dbscan.fit(X_scaled)
labels = dbscan.labels_

# Plot results
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=['red' if label == -1 else 'blue' for label in labels])
plt.title("DBSCAN on Moon-Shaped Data")
plt.show()
```

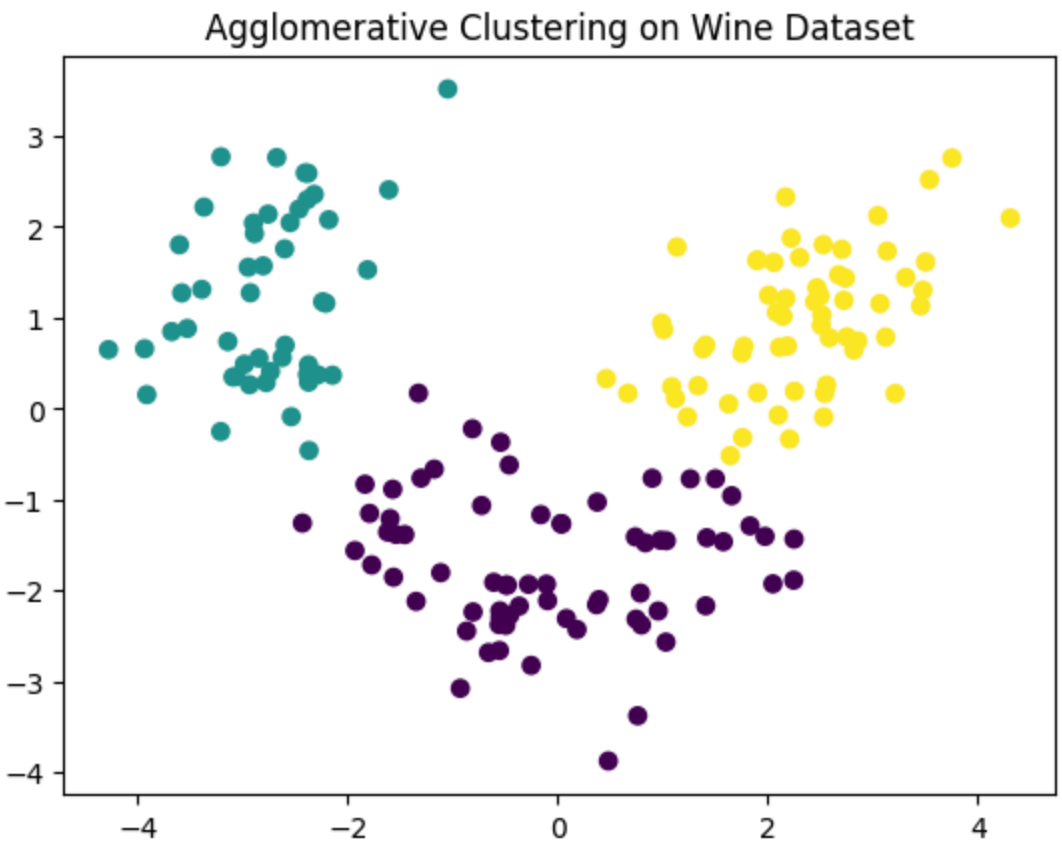
**Output:**




**Question 9: Load the Wine dataset, reduce it to 2D using PCA, then apply Agglomerative Clustering and visualize the result in 2D with a scatter plot. (Include your Python code and output in the code box below.)**
**Answer:**

Here's how you can load the Wine dataset, reduce it to 2D using PCA, apply Agglomerative Clustering, and visualize the result:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering

# Load Wine dataset
wine = load_wine()
X = wine.data

# Apply StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Reduce to 2D using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Apply Agglomerative Clustering
agglo_cluster = AgglomerativeClustering(n_clusters=3)
labels = agglo_cluster.fit_predict(X_pca)

# Visualize result
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis')
plt.title("Agglomerative Clustering on Wine Dataset")
plt.show()
```


Agglomerative Clustering on Wine Dataset
3
2
1
0
−1
−2
−3
−4
−4
−2
0
2
4

**Question 10: You are working as a data analyst at an e-commerce company. The marketing team wants to segment customers based on their purchasing behavior to run targeted promotions. The dataset contains customer demographics and their product purchase history across categories. Describe your real-world data science workflow using clustering: ● Which clustering algorithm(s) would you use and why? ● How would you preprocess the data (missing values, scaling)? ● How would you determine the number of clusters? ● How would the marketing team benefit from your clustering analysis? (Include your Python code and output in the code box below.)**
**Answer:**
Here's a real-world data science workflow using clustering for customer segmentation:

**Clustering Algorithm:**

- I'd use K-Means or Hierarchical Clustering (AgglomerativeClustering) depending on data characteristics.
- K-Means is efficient for large datasets and provides clear cluster assignments.
- Hierarchical Clustering helps visualize cluster structure and doesn't require pre-specifying K.

**Data Preprocessing:**

- Handle missing values: impute with median (numerical) or most frequent (categorical).
- Scale numerical features using StandardScaler.
- Encode categorical variables (e.g., one-hot encoding).

**Determining Number of Clusters:**

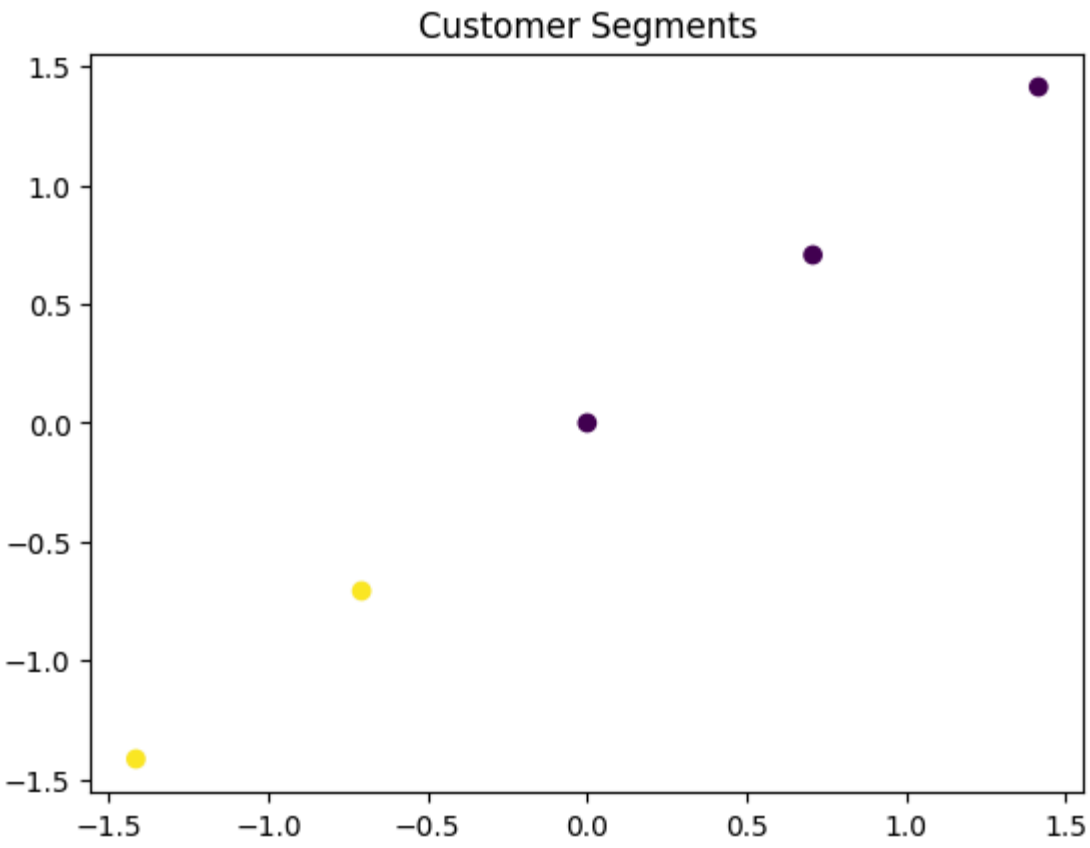
- Use Elbow Method (WCSS vs. K) for K-Means.
- Analyze dendrogram for Hierarchical Clustering.
- Evaluate silhouette score or Calinski-Harabasz index.

**Benefits for Marketing Team:**

- Targeted promotions based on customer segments.
- Personalized marketing strategies.
- Improved customer retention and conversion.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Load dataset (e.g., customer_data.csv)
```

```
# df = pd.read_csv('customer_data.csv')

# Preprocess data
df = pd.DataFrame({
    'Age': [25, 30, 40, 35, 20],
    'Income': [50000, 60000, 80000, 70000, 40000],
    'Spending': [1000, 1500, 2000, 1200, 800]
})
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Apply K-Means
kmeans = KMeans(n_clusters=2)
labels = kmeans.fit_predict(X_scaled)

# Evaluate silhouette score
score = silhouette_score(X_scaled, labels)
print("Silhouette Score:", score)

# Visualize clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
plt.title("Customer Segments")
plt.show()
```


Customer Segments
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5