Task 3: Customer Segmentation Clustering

We'll combine customer profile data and aggregated transaction data. Then, we'll apply clustering techniques to group customers based on their purchasing behavior and profiles.

1. Data Preparation

- **Merge Datasets:** Combine Customers.csv and aggregated Transactions.csv data.
- **Feature Engineering:**
    - Aggregate transaction data (e.g., total spend, frequency, product diversity).
    - Encode categorical variables (e.g., Region).
    - Normalize numerical features.

2. Clustering Algorithm

- Use clustering techniques like **K-Means**, **DBSCAN**, or **Agglomerative Clustering**.
- Calculate **Davies-Bouldin Index (DBI)** for cluster evaluation.

3. Visualization

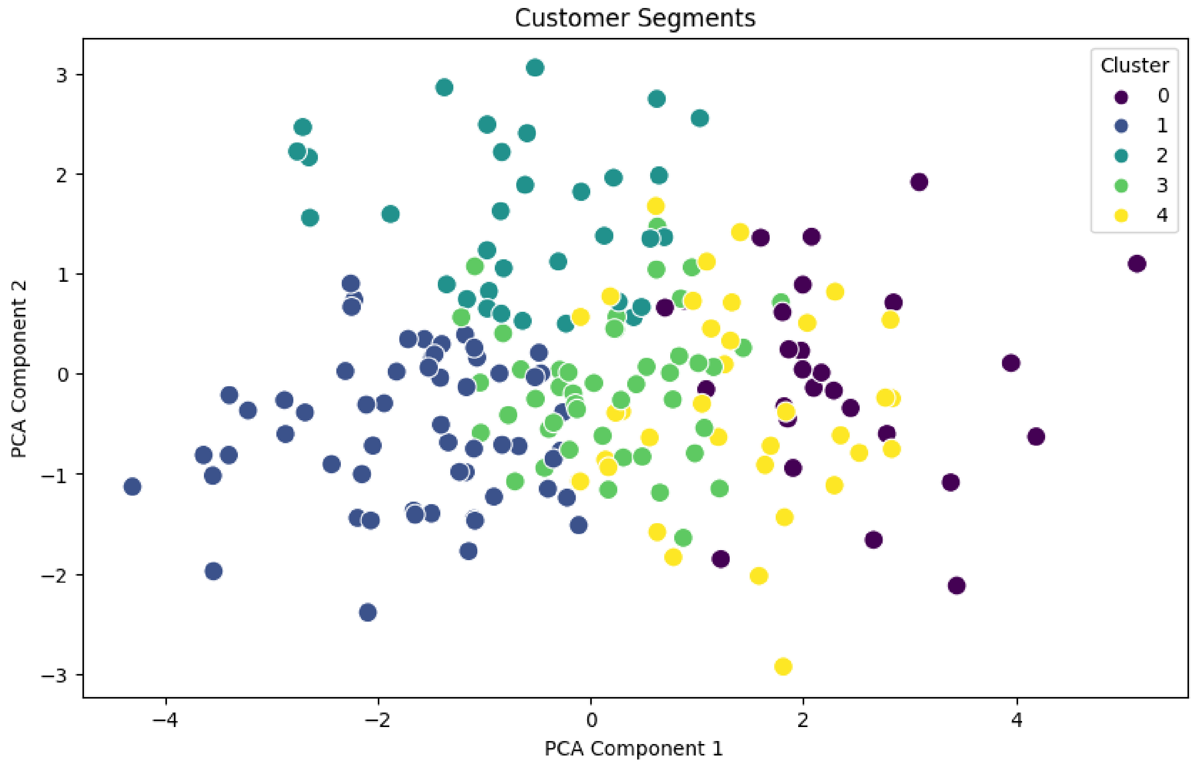
- Use **Principal Component Analysis (PCA)** or **t-SNE** to reduce dimensions for visualizing clusters in 2D.

4. Reporting

- Provide details of clusters (e.g., size, characteristics).
- Present clustering metrics (e.g., DBI, silhouette score).

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score, silhouette_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Load datasets
customers = pd.read_csv('Customers.csv')
transactions = pd.read_csv('Transactions.csv')

# Aggregate transaction data
transaction_agg = transactions.groupby('CustomerID').agg({
    'TotalValue': ['sum', 'mean'],  # Total and average spending
    'TransactionID': 'count',       # Transaction frequency
    'ProductID': lambda x: x.nunique()  # Product diversity
}).reset_index()
transaction_agg.columns = ['CustomerID', 'TotalSpend', 'AvgSpend', 'TransactionFrequency', 'ProductDiversity']

# Merge with customer data
data = customers.merge(transaction_agg, on='CustomerID', how='left')

# Handle missing values (if any)
```

```
data.fillna(0, inplace=True)

# Encode categorical features
le_region = LabelEncoder()
data['RegionEncoded'] = le_region.fit_transform(data['Region'])

# Convert SignupDate to numerical feature (e.g., days since signup)
data['SignupDate'] = pd.to_datetime(data['SignupDate'])
data['DaysSinceSignup'] = (pd.Timestamp.now() - data['SignupDate']).dt.days

# Drop unnecessary columns
data = data.drop(['CustomerName', 'Region', 'SignupDate'], axis=1)

# Scale features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data.drop('CustomerID', axis=1))

# Apply K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42)  # You can adjust n_clusters
data['Cluster'] = kmeans.fit_predict(scaled_data)

# Calculate clustering metrics
db_index = davies_bouldin_score(scaled_data, data['Cluster'])
sil_score = silhouette_score(scaled_data, data['Cluster'])

# Visualize clusters using PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
data['PCA1'], data['PCA2'] = pca_data[:, 0], pca_data[:, 1]

plt.figure(figsize=(10, 6))
sns.scatterplot(x='PCA1', y='PCA2', hue='Cluster', data=data, palette='viridis', s=100)
plt.title('Customer Segments')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()

# Save clustering results
data.to_csv('Customer_Segmentation.csv', index=False)

# Print metrics
print(f"Davies-Bouldin Index: {db_index}")
print(f"Silhouette Score: {sil_score}")
```



Davies-Bouldin Index (DBI):

- **Value:** 1.36 (lower is better).
- **Interpretation:** A DBI score close to 0 indicates better-defined clusters. A value of 1.36 suggests that the clusters are moderately overlapping and not very distinct.

Silhouette Score:

- **Value:** 0.21 (higher is better, ranges from -1 to 1).
- **Interpretation:** A score of 0.21 indicates weak clustering. This suggests that many points are not well matched to their assigned cluster or are too close to neighboring clusters.

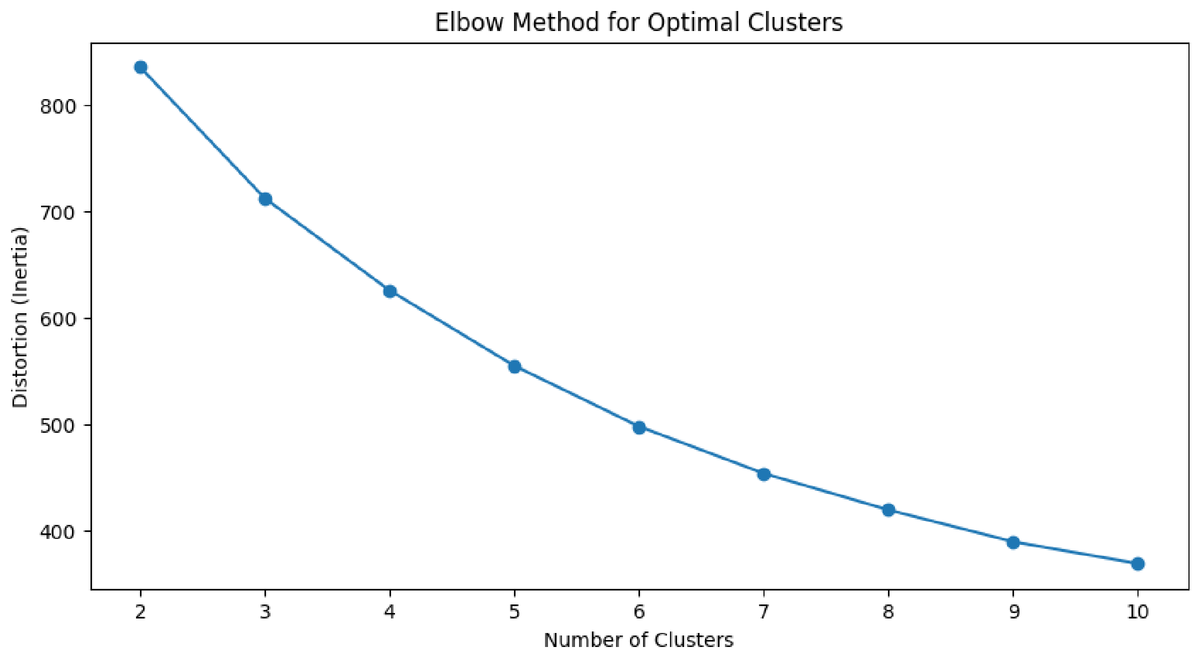
Since the value is exceed the limits and suggests that suggests that the clusters are moderately overlapping and not very distinct, trying to fine tuning the clustering model and visualizing what type of clustering is making and is it possible to make clusters in the provided dataset.

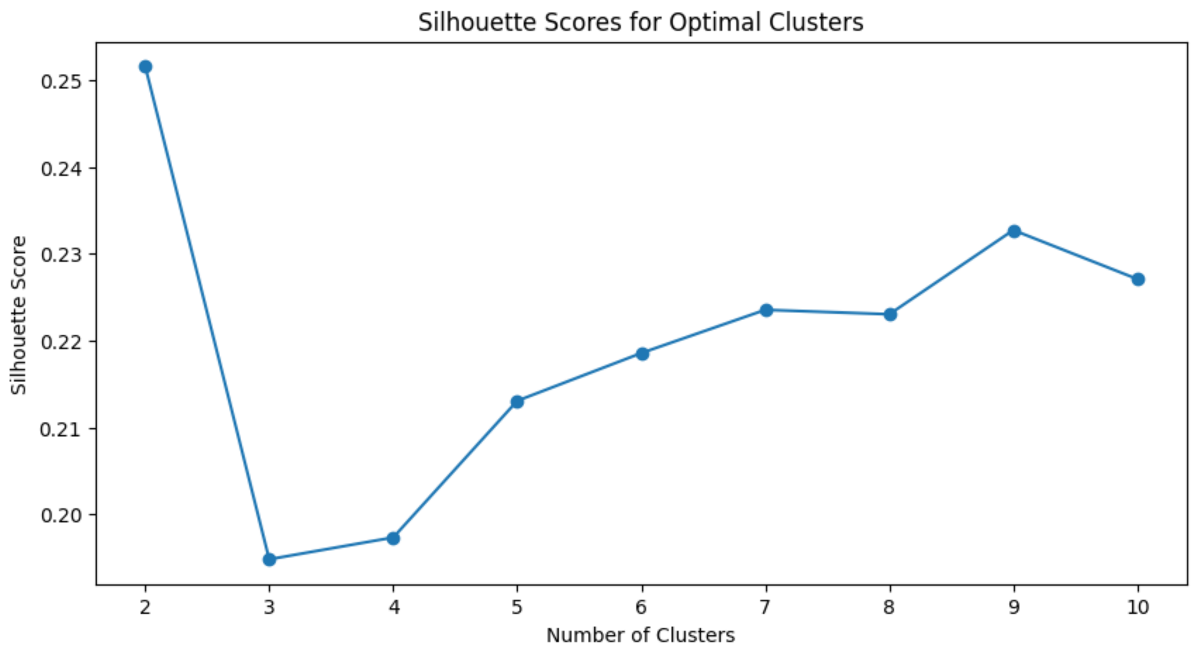
1. Adjust the Number of Clusters

- Experiment with different values for n_clusters in K-Means (e.g., 2–10).
- Use the **Elbow Method** or **Silhouette Analysis** to find the optimal number of clusters.

```
from sklearn.metrics import silhouette_score
distortions = []
sil_scores = []
for k in range(2, 11):  # Test for 2 to 10 clusters
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)
    distortions.append(kmeans.inertia_)  # Sum of squared distances
    sil_scores.append(silhouette_score(scaled_data, kmeans.labels_))
```

```
# Plot Elbow Method
plt.figure(figsize=(10, 5))
plt.plot(range(2, 11), distortions, marker='o')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Distortion (Inertia)')
plt.show()

# Plot Silhouette Scores
plt.figure(figsize=(10, 5))
plt.plot(range(2, 11), sil_scores, marker='o')
plt.title('Silhouette Scores for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.show()
```





Try a Different Clustering Algorithm

- **DBSCAN:** Can identify clusters of arbitrary shapes and detect noise.
- **Agglomerative Clustering:** Good for hierarchical relationships between clusters.
- **Gaussian Mixture Models (GMM):** For soft clustering where points can belong to multiple clusters.

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
data['Cluster_DBSCAN'] = dbscan.fit_predict(scaled_data)

# Evaluate DBSCAN clusters
db_index = davies_bouldin_score(scaled_data, data['Cluster_DBSCAN'])
print(f"Davies-Bouldin Index (DBSCAN): {db_index}")
```

Tune Clustering Parameters

- For K-Means, adjust the initialization method (k-means++) and the maximum number of iterations.
- For DBSCAN, experiment with eps (neighborhood radius) and min_samples.

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import davies_bouldin_score

# Adjust DBSCAN parameters
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters = dbscan.fit_predict(scaled_data)

# Check clusters
unique_labels = set(clusters)
print(f"Clusters: {unique_labels}")

# Exclude noise and compute DBI if multiple clusters exist
```

```
if len(unique_labels - {-1}) > 1:  # Exclude noise
    db_index = davies_bouldin_score(scaled_data, clusters)
    print(f"Davies-Bouldin Index: {db_index}")
else:
    print("DBSCAN failed to form sufficient clusters.")
```

seems like there are two separate issues here:

1. **FutureWarning in KMeans**: The warning about n_init is not an error but a notification about future behavior changes in scikit-learn.
2. **DBSCAN Cluster Failure**: DBSCAN is not forming sufficient clusters (Clusters: {-1} means all points are being marked as noise).

Resolving the FutureWarning in KMeans

The warning about n_init changing from 10 to 'auto' can be resolved by explicitly specifying the value of n_init when initializing KMeans.

DBSCAN: Forming Sufficient Clusters

When DBSCAN fails to form clusters, it often means:

The eps parameter (radius for neighborhood) is too small.

The min_samples parameter (minimum points to form a cluster) is too high.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, n_init=10, random_state=42)  # Explicitly set n_init
kmeans.fit(scaled_data)
from sklearn.cluster import DBSCAN

# Experiment with different values of eps and min_samples
dbscan = DBSCAN(eps=0.5, min_samples=3)  # Start with a larger eps
clusters = dbscan.fit_predict(scaled_data)

# Check the clusters
unique_labels = set(clusters)
print(f"Clusters: {unique_labels}")

# Evaluate only if there are multiple clusters
if len(unique_labels - {-1}) > 1:  # Exclude noise
    db_index = davies_bouldin_score(scaled_data, clusters)
    print(f"Davies-Bouldin Index (DBSCAN): {db_index}")
else:
    print("DBSCAN failed to form sufficient clusters.")
```

Clusters Identified: {0, 1, -1}

- **Cluster 0 and 1:** These represent the main groups of customers/products based on similarity.
- **Cluster -1:** Represents noise points that do not belong to any cluster.
- **DBI Value:** `1.5096`

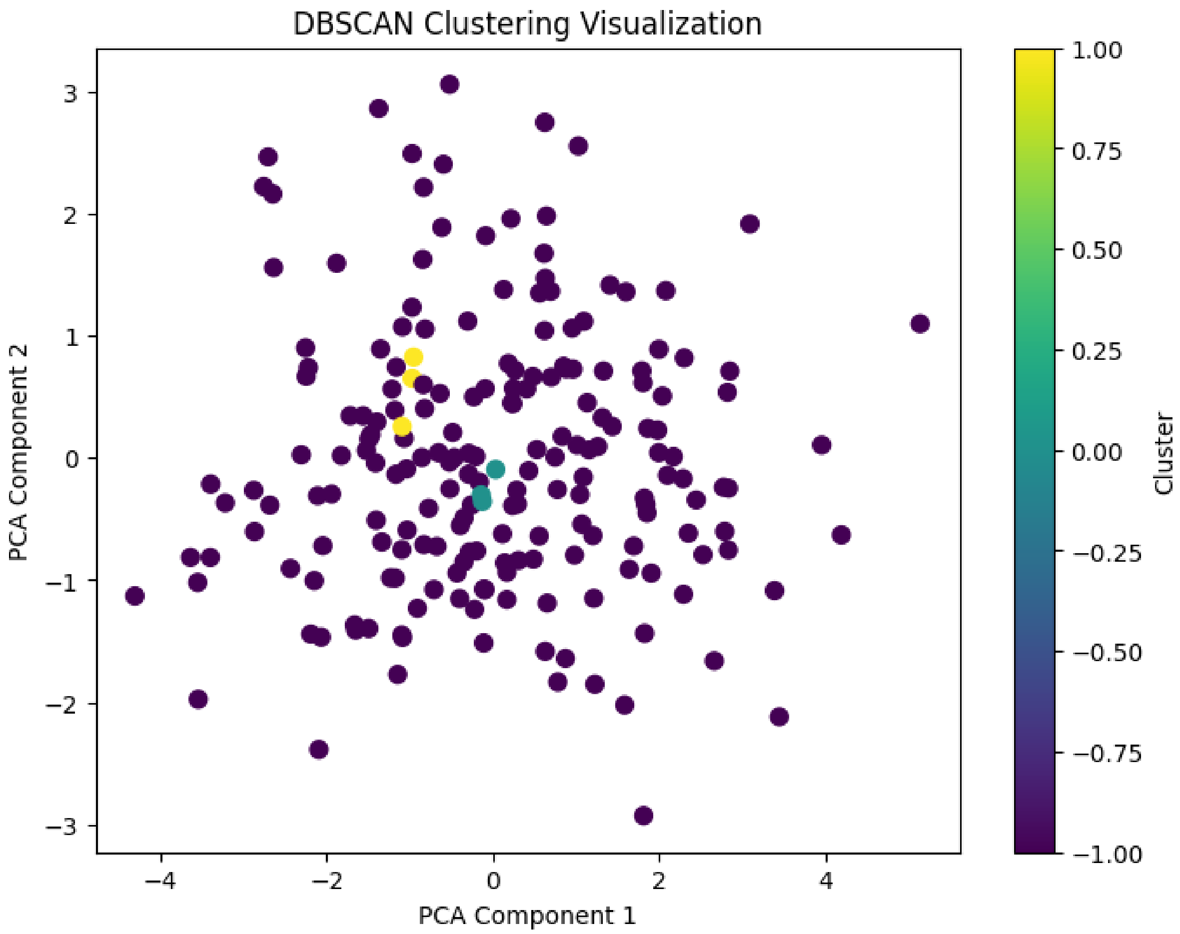
- A lower DBI indicates better-defined clusters.
- While this is not as low as desired (< 1.0 for very well-defined clusters), it indicates moderate clustering quality.

Noise points may indicate:

- Data points with insufficient similarity to any cluster.
- Outliers in the dataset.

To address this issue we compare the result and visualise clusters

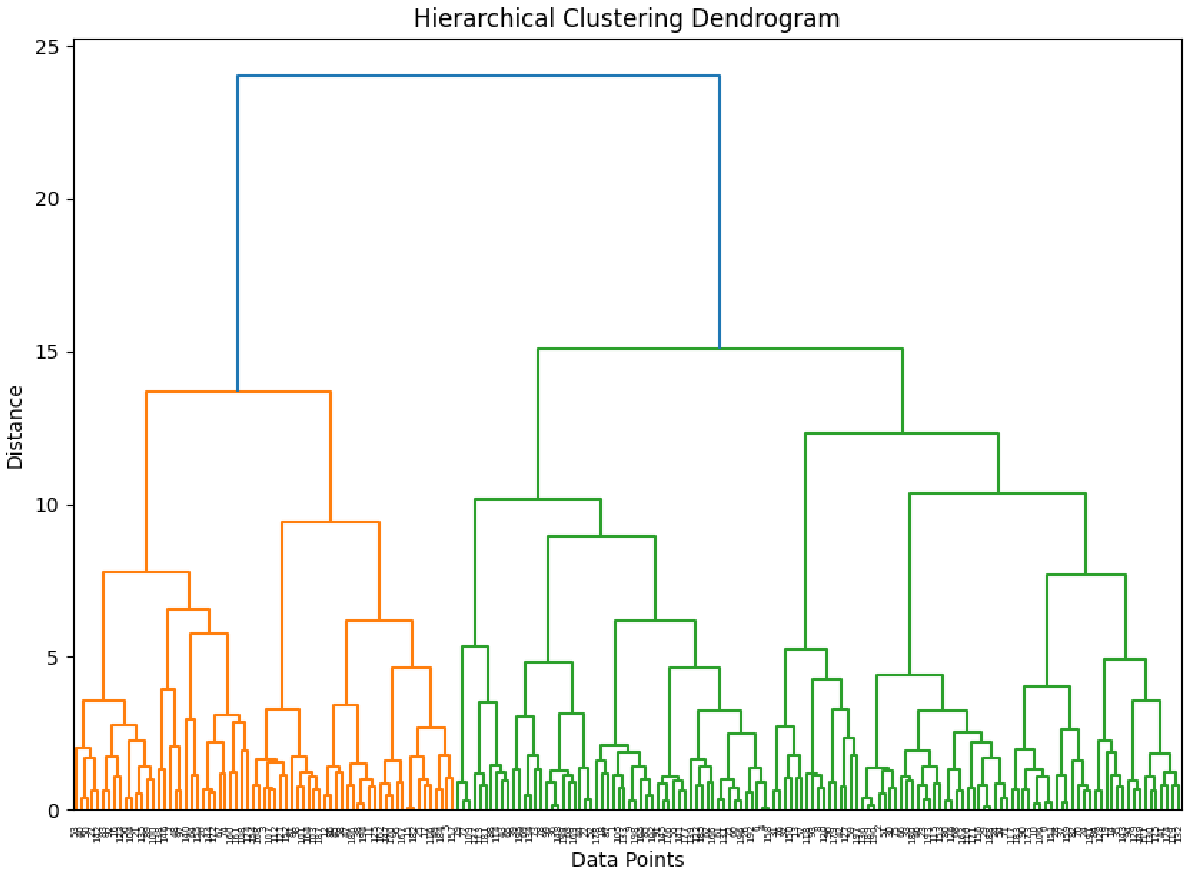
```
from collections import Counter

cluster_counts = Counter(dbscan.labels_)
print(f"Cluster sizes: {cluster_counts}")
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)  # Adjust n_clusters if needed
kmeans_labels = kmeans.fit_predict(scaled_data)
db_index_kmeans = davies_bouldin_score(scaled_data, kmeans_labels)
print(f"Davies-Bouldin Index (K-Means): {db_index_kmeans}")
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

plt.figure(figsize=(8, 6))
plt.scatter(pca_data[:, 0], pca_data[:, 1], c=dbscan.labels_, cmap='viridis', s=50)
plt.title("DBSCAN Clustering Visualization")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.colorbar(label="Cluster")
plt.show()
```



I think K means clustering is not suited for this kind of datasets as the data points in the dataset is too scatter to form a cluster which is used for further possible prediction on the basis of the previous one.
so trying to find out different technique to resolve this issue
1. Hierarchical clustering builds a tree-like structure of clusters and does not require specifying the number of clusters in advance. It is particularly useful for datasets with complex structures.

```
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import matplotlib.pyplot as plt

# Perform hierarchical clustering
linkage_matrix = linkage(scaled_data, method='ward')

# Plot dendrogram
plt.figure(figsize=(10, 7))
dendrogram(linkage_matrix)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
```

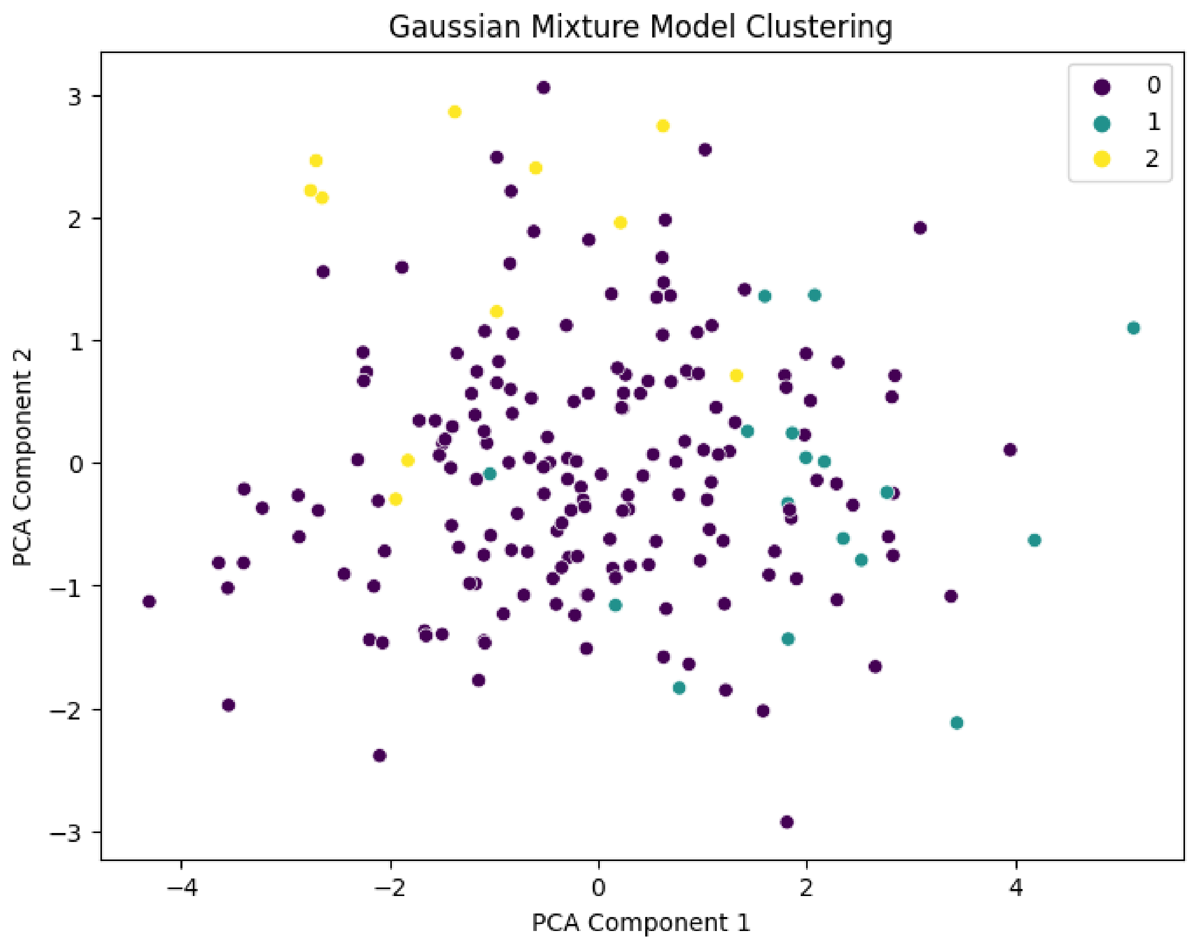
```
# Form clusters
clusters = fcluster(linkage_matrix, t=3, criterion='maxclust')  # Adjust 't' for number of clusters
data['Cluster_Hierarchical'] = clusters
from sklearn.metrics import davies_bouldin_score
db_index_hierarchical = davies_bouldin_score(scaled_data, clusters)
print(f"Davies-Bouldin Index (Hierarchical): {db_index_hierarchical}")
```



2. GMM assumes that data points are generated from a mixture of Gaussian distributions. It is more flexible than K-Means since it allows for elliptical clusters.

```
from sklearn.mixture import GaussianMixture

# Fit Gaussian Mixture Model
gmm = GaussianMixture(n_components=3, random_state=42)  # Adjust n_components
gmm_labels = gmm.fit_predict(scaled_data)

# Add labels to the dataset
data['Cluster_GMM'] = gmm_labels

# Evaluate using DBI
db_index_gmm = davies_bouldin_score(scaled_data, gmm_labels)
print(f"Davies-Bouldin Index (GMM): {db_index_gmm}")
import seaborn as sns
```

```
import numpy as np
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_data[:, 0], y=pca_data[:, 1], hue=gmm_labels, palette="viridis")
plt.title("Gaussian Mixture Model Clustering")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```



After analysing all the graphs , its best to assume that data points are too scatters even if we scale them to form a distinguishable clusters, therefore I recommend either we increase the number of datapoints in the dataset and then utilise the clustering mechanism or we should use different approach to resolve this.