Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 7

AIM: To implement different clustering algorithms.

THEORY:

1] Clustering:

Clustering is an unsupervised learning technique used to group similar data points into clusters based on their features. It helps to discover patterns, structures, or groupings in data without predefined labels.

2] K-Means Clustering

- **Type**: Partition-based clustering
- **Concept**: Divides data into **K** clusters where each point belongs to the nearest **centroid** (center).
- **Steps**:
    1. Choose number of clusters **K**
    2. Initialize **K centroids** randomly
    3. Assign each point to the nearest centroid
    4. Update centroids as the **mean** of points in the cluster
    5. Repeat until centroids don't change
- **Pros**: Simple, fast
- **Cons**: Sensitive to outliers, needs predefined **K**, assumes spherical clusters

3] DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- **Type**: Density-based clustering
- **Concept**: Groups data points that are closely packed together, and marks points in low-density regions as **outliers/noise**.
- **Parameters**:
    - **ε (epsilon)**: radius to search nearby points
    - **MinPts**: minimum number of points to form a dense region
- **Pros**: Can find clusters of **arbitrary shape**, handles **noise** well
- **Cons**: Difficult to choose optimal ε and MinPts

3] Hierarchical Clustering

- **Type**: Tree-based (Hierarchical) clustering
- **Concept**: Builds a **dendrogram** (tree) by either:
    - **Agglomerative**: start with individual points and merge clusters

  - **Divisive**: start with all points in one cluster and split
- **Linkage methods**: Single, Complete, Average
- **Pros**: No need to choose **K**, good for visualizing structure
- **Cons**: Slow for large datasets, sensitive to noise

DATASET:

Source: https://catalog.data.gov/dataset/electric-vehicle-population-data

The **Electric Vehicle Population Data** dataset provides detailed information about electric vehicles (EVs) registered in the state of Washington. It includes attributes such as vehicle make, model, year, electric vehicle type (e.g., battery electric or plug-in hybrid), electric range, and the city and ZIP code where the vehicle is registered. This dataset is useful for analyzing EV adoption trends, geographic distribution, and the types of electric vehicles in use across different regions.

STEPS:

Step 1: Import Library

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.decomposition import PCA
file_path = "Electric_Vehicle_Population_Data.csv"
df = pd.read_csv(file_path)
# Display first few rows
print("First 5 rows of the dataset:")
print(df.head())
# Display dataset information
print("\nDataset Information:")
print(df.info())
```

Output:

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208184 entries, 0 to 208183
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         208184 non-null  object
 1   County                                             208184 non-null  object
 2   City                                               208184 non-null  object
 3   State                                              208184 non-null  object
 4   Postal Code                                        208184 non-null  int64
 5   Model Year                                         208184 non-null  int64
 6   Make                                               208184 non-null  object
 7   Model                                              208184 non-null  object
 8   Electric Vehicle Type                              208184 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  208184 non-null  object
 10  Electric Range                                     208157 non-null  float64
 11  Base MSRP                                          208157 non-null  float64
 12  Legislative District                               208030 non-null  float64
 13  DOL Vehicle ID                                     208184 non-null  int64
 14  Vehicle Location                                   208178 non-null  object
 15  Electric Utility                                   208184 non-null  object
 16  2020 Census Tract                                  208184 non-null  int64
dtypes: float64(3), int64(4), object(10)
memory usage: 27.0+ MB
None
```

This code imports essential Python libraries for data analysis, visualization, and clustering, then loads the Electric Vehicle Population dataset from a CSV file using pandas. It displays the first five rows of the dataset to give a quick overview of the data and uses .info() to show the structure of the dataset, including column names, data types, and non-null counts, helping to understand the dataset's composition before applying any clustering techniques like K-Means, DBSCAN, or Hierarchical clustering.

Step 2:

Code:

```
features = ['Model Year', 'Electric Range', 'Legislative District']
df_selected = df[features].dropna()  # Remove rows with missing values
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df_selected)
print("Scaled Data Sample:")
print(pd.DataFrame(data_scaled, columns=features).head())
```

Output:

```
Scaled Data Sample:
   Model Year  Electric Range  Legislative District
0   -2.437415        0.658622              0.790910
1   -0.774051        2.041202             -1.888065
2    1.221985       -0.085845              0.389064
3    0.889313       -0.062211             -1.821091
4   -0.108706       -0.558522             -0.950424
```

This code selects three relevant features—**Model Year**, **Electric Range**, and **Legislative District**—from the dataset for clustering. It removes any rows with missing values to ensure clean input data. The selected features are then standardized using **StandardScaler**, which scales them to have zero mean and unit variance, making the data suitable for clustering algorithms. Finally, it prints the first few rows of the scaled data to verify the preprocessing step.

Step 3:

Code:

```
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
df_pca = pd.DataFrame(data_pca, columns=['PCA1', 'PCA2'])
print("PCA-Transformed Data Sample:")
print(df_pca.head())
```

Output:

```
PCA-Transformed Data Sample:
       PCA1      PCA2
0  2.219979  0.695349
1  1.910952 -1.965692
2 -0.907835  0.429744
3 -0.747534 -1.789679
4 -0.357140 -0.938073
```

This code applies **Principal Component Analysis (PCA)** to reduce the dimensionality of the scaled data from three features down to **two principal components**—PCA1 and PCA2. This transformation helps simplify the dataset while preserving most of its variance, making it easier to visualize and interpret during clustering. The resulting PCA-transformed data is stored in a new DataFrame and the first few rows are printed to preview the output.

Step 4:

Code:

```
# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_pca['KMeans_Cluster'] = kmeans.fit_predict(data_scaled)
```

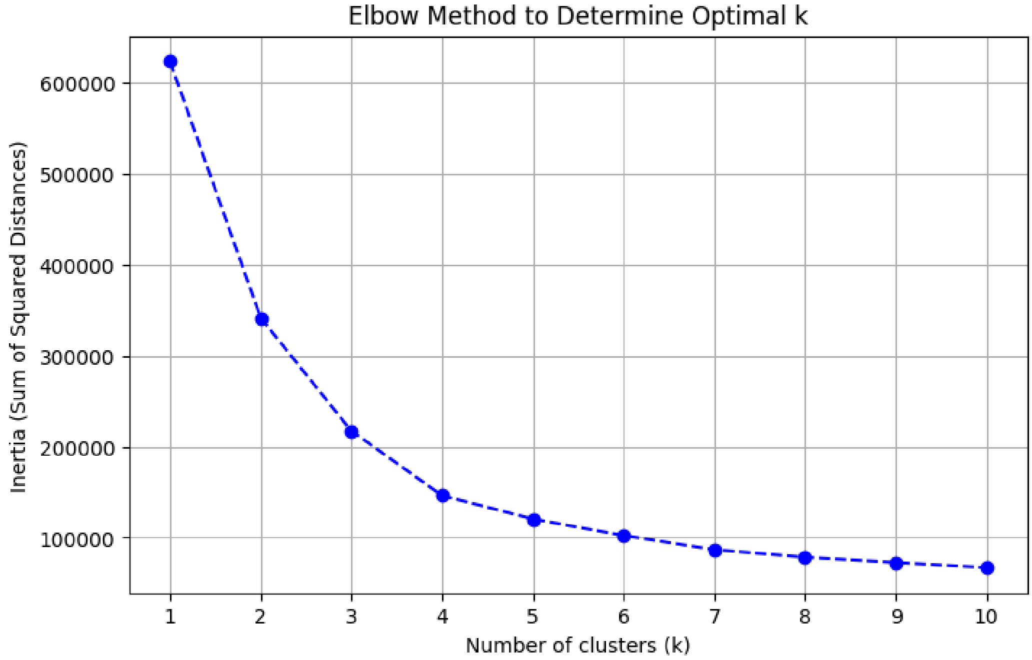
This code applies the **K-Means clustering algorithm** to the **scaled data** using 3 clusters. It initializes the KMeans model with n_clusters=3, sets a random_state for reproducibility, and runs the algorithm with n_init=10 (i.e., 10 different centroid initializations to ensure a good solution). The resulting cluster labels are stored in a new column KMeans_Cluster in the PCA-transformed DataFrame df_pca.

Step 5:

Code:

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

inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(data_scaled)  # Use scaled data, not PCA
    inertia.append(kmeans.inertia_)

# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, 'bo--')
plt.title('Elbow Method to Determine Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(K_range)
plt.grid(True)
plt.show()
```

Output:



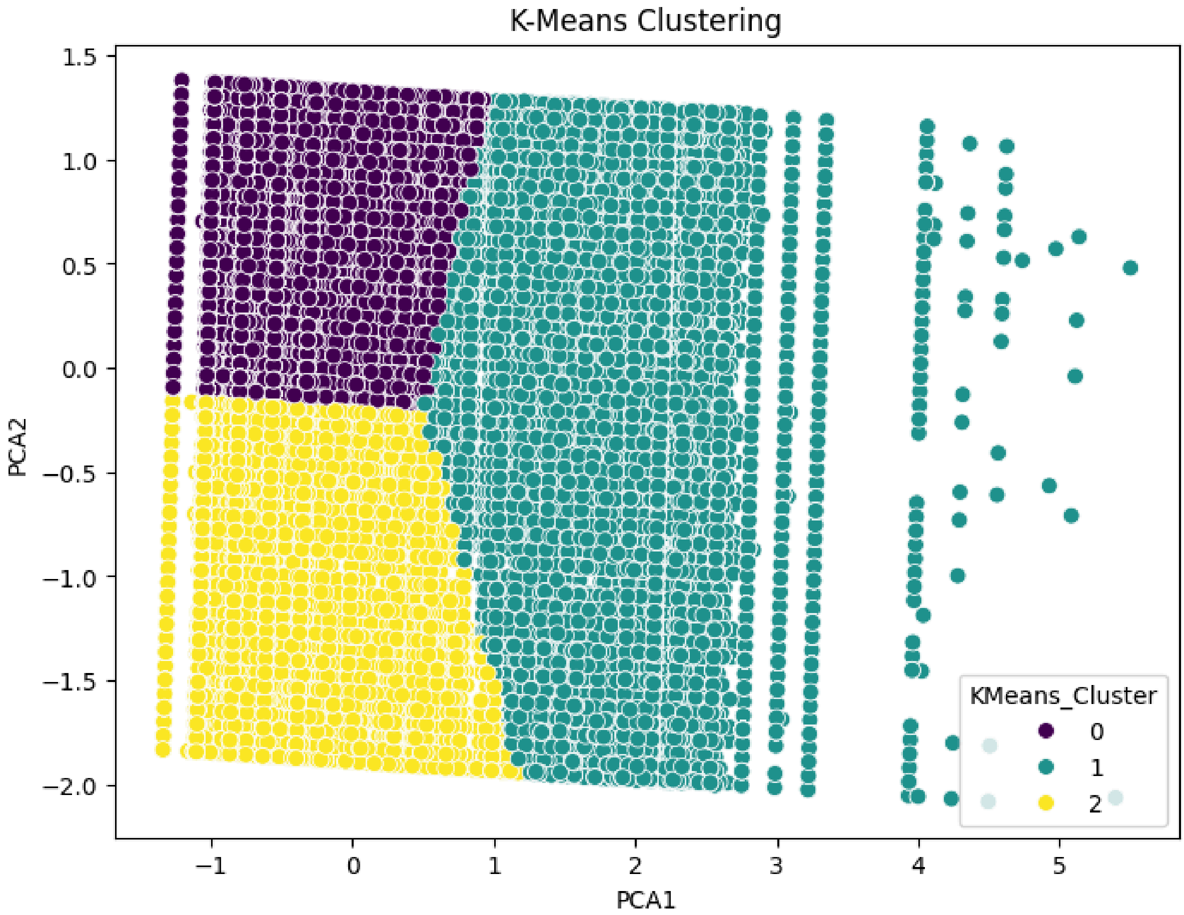
This code applies **K-Means clustering** to the standardized dataset with the number of clusters set to **3**. It initializes the KMeans algorithm with a fixed random_state for reproducibility and n_init=10 to run the algorithm multiple times with different centroid seeds for better results. The predicted cluster labels are then added as a new column called **'KMeans_Cluster'** to the PCA-transformed DataFrame for further analysis or visualization.

Step 6:

Code:

```
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca['PCA1'], y=df_pca['PCA2'], hue=df_pca['KMeans_Cluster'], palette='viridis', s=50)
plt.title('K-Means Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()

print("K-Means Centroids (in original scaled feature space):\n", kmeans.cluster_centers_)
print("K-Means Inertia (Sum of Squared Distances):", kmeans.inertia_)
```

Output:




```
K-Means Centroids (in original scaled feature space):
 [[ 0.4878545  -0.48239497 -0.53415955]
 [ 0.04846403 -0.48794389  0.83730684]
 [-0.97240482  2.18804824 -1.06501944]
 [-1.30655933  0.05551354  0.62651575]
 [-2.61009677  0.24585807  0.54012916]
 [ 0.70305006 -0.50249748  0.25972122]
 [-1.83434221  0.02669846 -1.09806921]
 [ 0.55258459 -0.47808297 -1.51845795]
 [-0.93226094  2.18340283  0.7419867 ]
 [ 0.74332277 -0.50687879  1.01404936]]
K-Means Inertia (Sum of Squared Distances): 66976.1058078572
```

This code visualizes the **K-Means clustering results** using a scatter plot of the two PCA components. Each point is colored according to its assigned cluster using the hue parameter and the **'viridis'** color palette. The plot helps interpret how well the data has been grouped into clusters. After the plot, the code prints the **cluster centroids** (in the original scaled feature space) and the **inertia**, which indicates how compact the clusters are—lower inertia generally means better clustering.

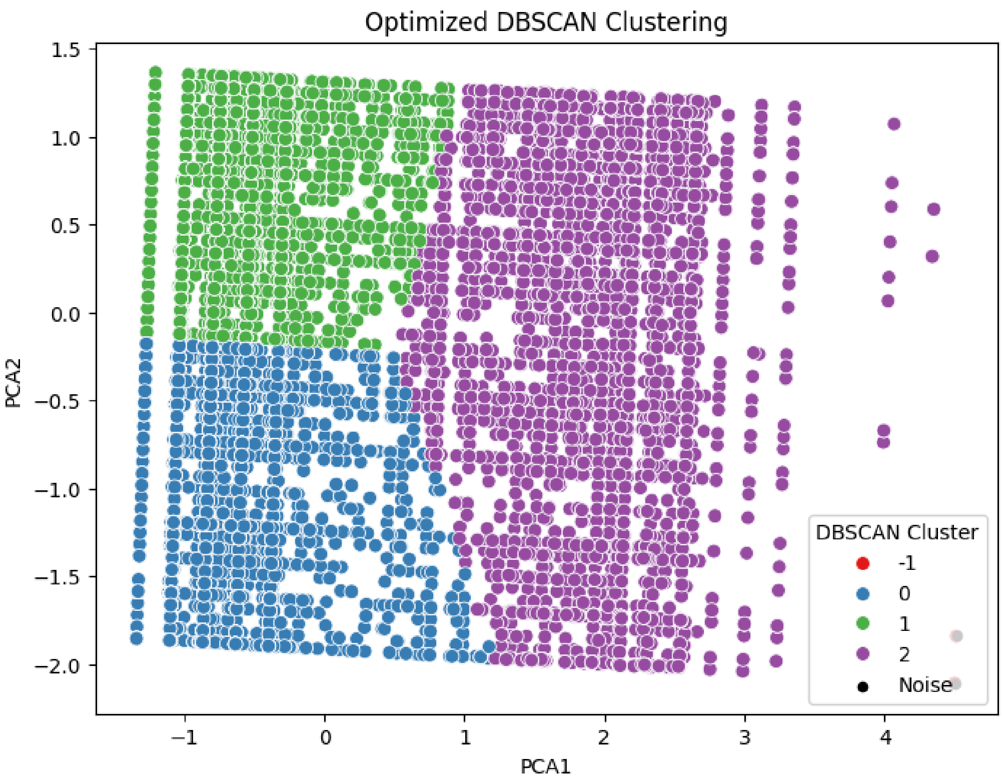
Step 6:

Code:

```
from sklearn.utils import shuffle
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=20000)  # Use a
smaller sample
dbscan = DBSCAN(eps=1.0, min_samples=10, n_jobs=1)  # Adjust `eps` and
`min_samples` to improve performance
df_pca_sample['DBSCAN_Cluster'] = dbscan.fit_predict(df_pca_sample)
unique_clusters = np.unique(df_pca_sample['DBSCAN_Cluster'])
print("Unique clusters found in DBSCAN:", unique_clusters)
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca_sample['PCA1'], y=df_pca_sample['PCA2'],
            hue=df_pca_sample['DBSCAN_Cluster'], palette='Set1', s=50)

if -1 in unique_clusters:
      plt.scatter(df_pca_sample.loc[df_pca_sample['DBSCAN_Cluster'] == -1,
'PCA1'],
                  df_pca_sample.loc[df_pca_sample['DBSCAN_Cluster'] == -1,
'PCA2'],
            color='black', label="Noise", s=20)

plt.title('Optimized DBSCAN Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title="DBSCAN Cluster")
plt.show()
```

Output:

```
Unique clusters found in DBSCAN: [-1  0  1  2]
```



This code applies the **DBSCAN clustering algorithm** on a **random sample of 20,000 rows** from the PCA-transformed dataset to improve performance. DBSCAN identifies clusters based on density, using eps=1.0 as the maximum distance between points and min_samples=10 as the minimum points to form a dense region. The resulting clusters, including noise points (labeled as -1), are visualized using a scatter plot with different colors for each cluster. Noise points are highlighted in **black**, helping to visualize outliers or sparse areas in the data.

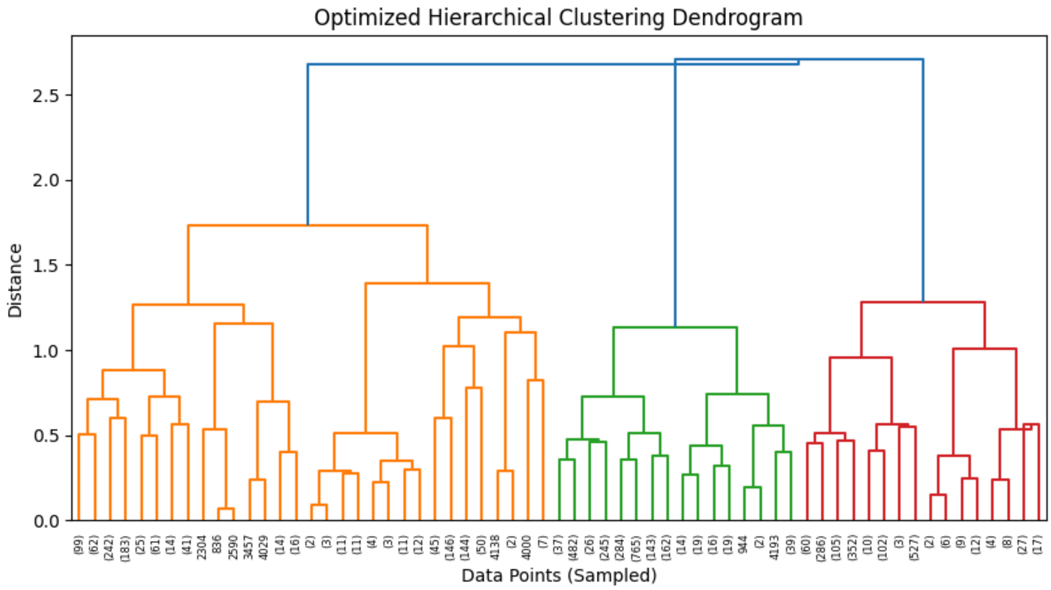
Step 7:

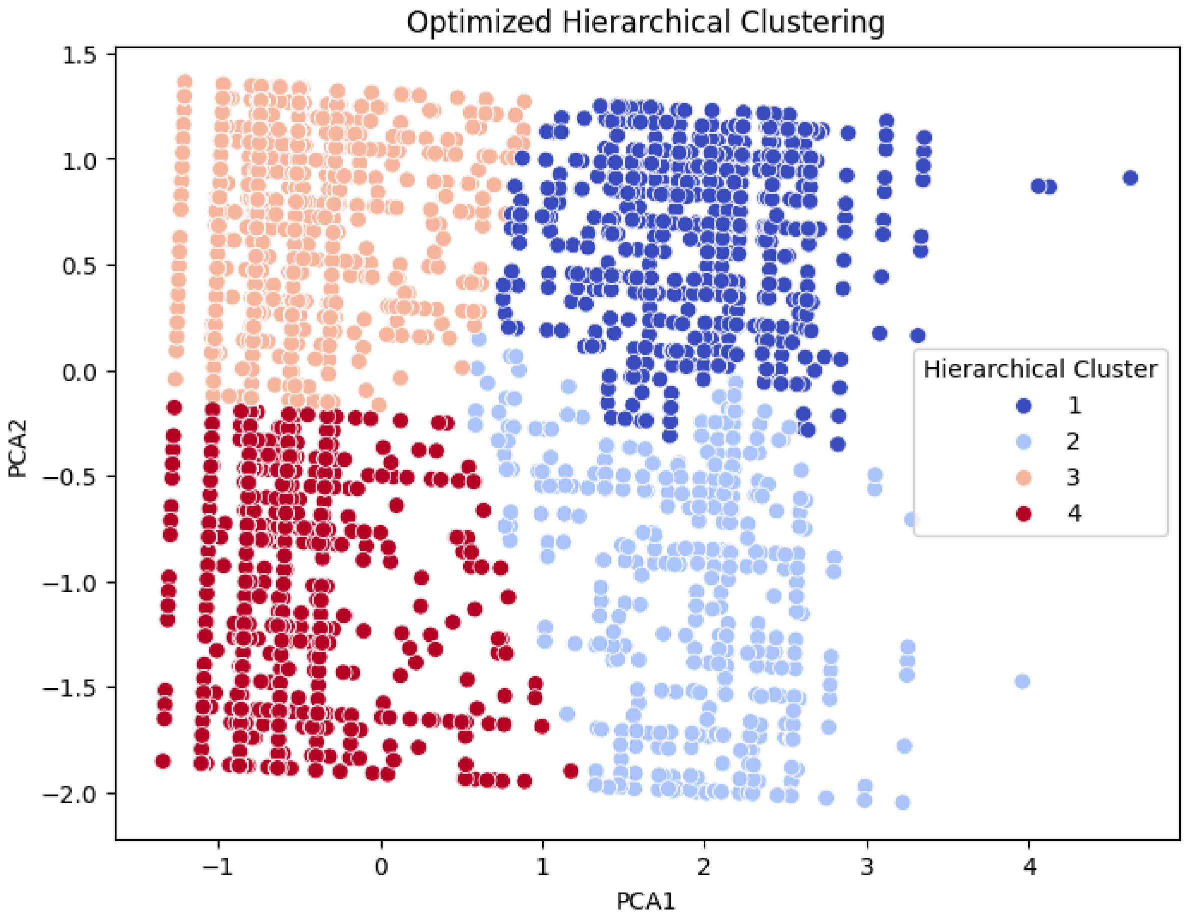
Code:

```
from sklearn.utils import shuffle

# Reduce dataset size to prevent RAM overuse
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=5000)  # Sample 5000 points

# Compute hierarchical clustering linkage matrix (Use 'centroid' for better performance)
linkage_matrix = linkage(df_pca_sample, method='centroid')

# Plot the Dendrogram (Limit displayed levels to avoid overloading memory)
```

```
plt.figure(figsize=(10,5))
dendrogram(linkage_matrix, truncate_mode='level', p=5)  # Only show top 5 levels
plt.title("Optimized Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points (Sampled)")
plt.ylabel("Distance")
plt.show()

# Extract clusters using an optimized threshold
df_pca_sample['Hierarchical_Cluster'] = fcluster(linkage_matrix, t=4, criterion='maxclust')

# Visualize Hierarchical Clusters
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca_sample['PCA1'], y=df_pca_sample['PCA2'],
                hue=df_pca_sample['Hierarchical_Cluster'], palette='coolwarm', s=50)

plt.title('Optimized Hierarchical Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title="Hierarchical Cluster")
plt.show()
```

Output:





This code performs **Hierarchical Clustering** on a randomly sampled subset of 5,000 PCA-transformed data points to reduce memory usage. It computes a **linkage matrix** using the centroid method and visualizes the hierarchical relationships between clusters using a **dendrogram**, truncated to show only the top 5 levels for clarity. Then, it extracts **4 clusters** from the hierarchy using a distance threshold (t=4) and visualizes the resulting clusters in a scatter plot, colored by their hierarchical cluster labels for easy interpretation.

CONCLUSION: In this experiment, clustering techniques—K-Means, DBSCAN, and Hierarchical Clustering—were applied to the Electric Vehicle Population dataset after preprocessing, feature selection, scaling, and PCA. K-Means efficiently grouped data into distinct clusters, with the Elbow Method helping determine the optimal number of clusters. DBSCAN identified density-based clusters and effectively detected outliers, while Hierarchical Clustering provided a tree-based structure of relationships among data points. Each algorithm offered unique insights: K-Means was suitable for compact clusters, DBSCAN handled irregular patterns and noise, and Hierarchical Clustering revealed multi-level groupings.