Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 7

Aim: To implement different clustering algorithms.

Problem Statement:
a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN), Hierarchical clustering)
b) Plot the cluster data and show mathematical steps.

Performance:

- Prerequisite: Import essential libraries: pandas for data manipulation, numpy for numerical computations, matplotlib.pyplot and seaborn for data visualization, and sklearn for clustering using K-Means and DBSCAN. Also, use scipy for hierarchical clustering and PCA for dimensionality reduction. Load the Electric Vehicle Population Dataset into a Pandas DataFrame using pd.read_csv(). Finally, explore the dataset by displaying the first few rows with df.head() and checking column names, data types, and missing values using df.info():

Command:
```
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)
print("First 5 rows of the dataset:")
print(df.head())
print("\nDataset Information:")
print(df.info())
```

```
First 5 rows of the dataset:
   VIN (1-10)    County      City State  Postal Code  Model Year    Make  \
0  2T3YL4DV0E      King  Bellevue    WA      98005.0        2014  TOYOTA
1  5YJ3E1EB6K      King   Bothell    WA      98011.0        2019   TESLA
2  5UX43EU02S  Thurston   Olympia    WA      98502.0        2025     BMW
3  JTMAB3FV5R  Thurston   Olympia    WA      98513.0        2024  TOYOTA
4  5YJYGDEE8M    Yakima     Selah    WA      98942.0        2021   TESLA

        Model                   Electric Vehicle Type  \
0        RAV4          Battery Electric Vehicle (BEV)
1     MODEL 3          Battery Electric Vehicle (BEV)
2          X5  Plug-in Hybrid Electric Vehicle (PHEV)
3  RAV4 PRIME  Plug-in Hybrid Electric Vehicle (PHEV)
4     MODEL Y          Battery Electric Vehicle (BEV)

   Clean Alternative Fuel Vehicle (CAFV) Eligibility  Electric Range  \
0            Clean Alternative Fuel Vehicle Eligible           103.0
1            Clean Alternative Fuel Vehicle Eligible           220.0
2            Clean Alternative Fuel Vehicle Eligible            40.0
3            Clean Alternative Fuel Vehicle Eligible            42.0
4  Eligibility unknown as battery range has not b...             0.0

   Base MSRP  Legislative District  DOL Vehicle ID  \
0        0.0                  41.0       186450183
1        0.0                   1.0       478093654
2        0.0                  35.0       274800718
3        0.0                   2.0       260758165
4        0.0                  15.0       236581355
```

```
              Vehicle Location                              Electric Utility  \
0   POINT (-122.1621 47.64441)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
1  POINT (-122.20563 47.76144)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
2  POINT (-122.92333 47.03779)                         PUGET SOUND ENERGY INC
3  POINT (-122.81754 46.98876)                         PUGET SOUND ENERGY INC
4  POINT (-120.53145 46.65405)                                     PACIFICORP

  2020 Census Tract
0      5.303302e+10
1      5.303302e+10
2      5.306701e+10
3      5.306701e+10
4      5.307700e+10
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232230 entries, 0 to 232229
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         232230 non-null  object
 1   County                                             232226 non-null  object
 2   City                                               232226 non-null  object
 3   State                                              232230 non-null  object
 4   Postal Code                                        232226 non-null  float64
 5   Model Year                                         232230 non-null  int64
 6   Make                                               232230 non-null  object
 7   Model                                              232230 non-null  object
 8   Electric Vehicle Type                              232230 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  232230 non-null  object
 10  Electric Range                                     232203 non-null  float64
 11  Base MSRP                                          232203 non-null  float64
 12  Legislative District                               231749 non-null  float64
 13  DOL Vehicle ID                                     232230 non-null  int64
 14  Vehicle Location                                   232219 non-null  object
 15  Electric Utility                                   232226 non-null  object
 16  2020 Census Tract                                  232226 non-null  float64
dtypes: float64(5), int64(2), object(10)
memory usage: 30.1+ MB
None
```

Step 1: Feature Selection and Preprocessing:-
Command: 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())

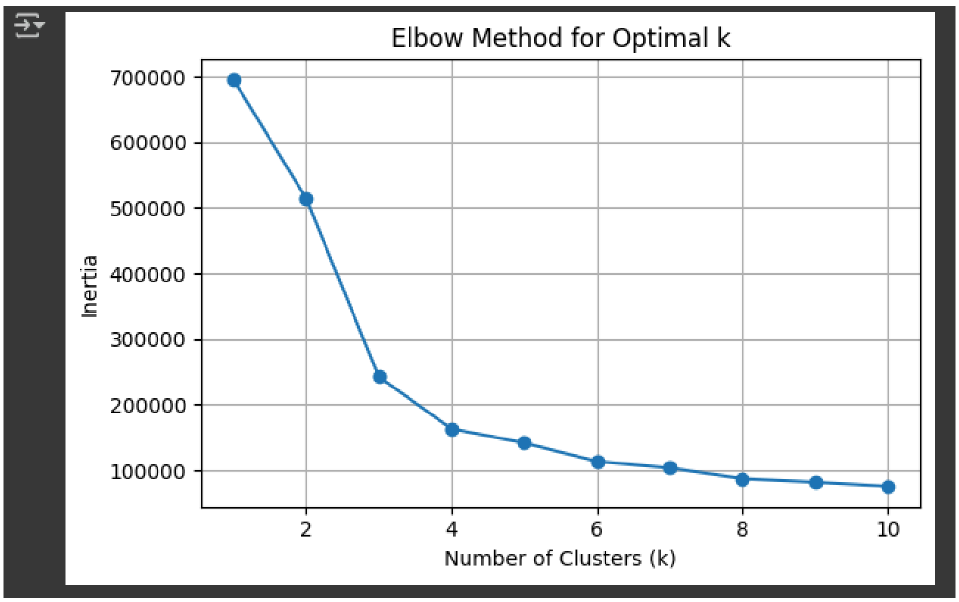
```
Scaled Data Sample:
   Model Year  Electric Range  Legislative District
0   -2.455485        0.666844              0.813147
1   -0.786089        2.053754             -1.870647
2    1.217186       -0.079953              0.410578
3    0.883307       -0.056245             -1.803552
4   -0.118331       -0.554111             -0.931319
```

The above code selects three relevant numerical features—'Model Year', 'Electric Range', and 'Legislative District'—from the electric vehicle dataset for clustering analysis. It removes any rows containing missing values to ensure clean data. The selected features are then standardized using StandardScaler, which scales the values to have a mean of 0 and standard deviation of 1. This normalization is important because it ensures that all features contribute equally to the clustering algorithms, preventing features with larger numeric ranges from dominating the results.

Step 2: Apply PCA for Dimensionality Reduction (for visualization):-
Command: 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())

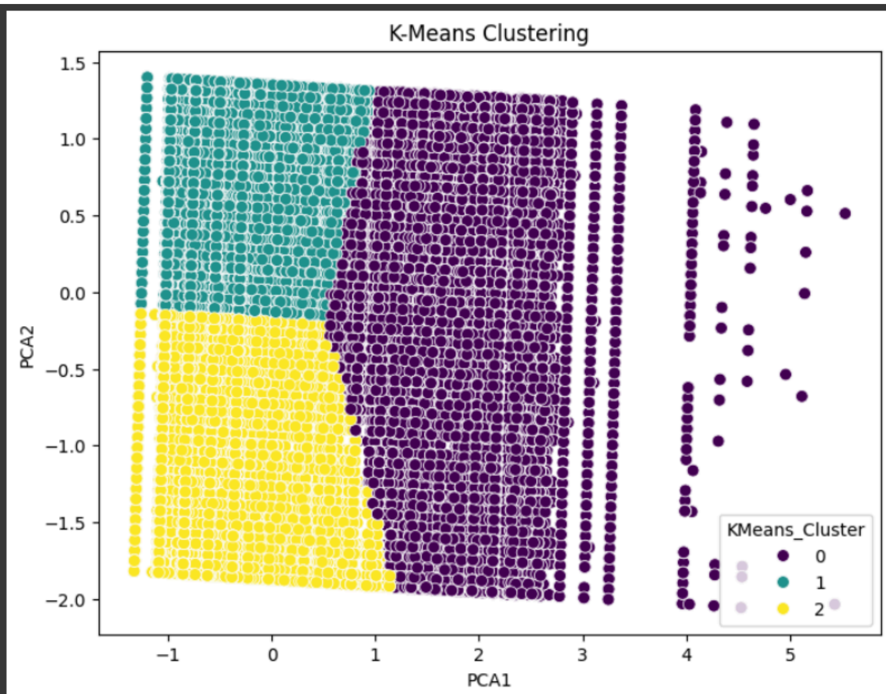
```
PCA-Transformed Data Sample:
       PCA1      PCA2
0  2.238305  0.721200
1  1.932234 -1.946371
2 -0.900138  0.448977
3 -0.735446 -1.774065
4 -0.344920 -0.919696
```

The code applies Principal Component Analysis (PCA) to reduce the standardized dataset to two principal components—PCA1 and PCA2. This helps simplify the dataset while retaining most of its important information (variance), making it suitable for 2D visualization of clustering results. The transformed data is stored in a new DataFrame df_pca, and the first few rows are printed to preview the reduced dataset.

Step 3: Determining Optimal Number of Clusters Using the Elbow Method:-
Command: inertia = []
```
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(6, 4))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.tight_layout()
plt.show()
```


The code implements the Elbow Method to identify the optimal number of clusters (k) for K-Means clustering by plotting inertia values (sum of squared distances from points to their closest cluster center) against various k values from 1 to 10. The resulting plot shows a sharp drop in inertia from k=1 to k=3, after which the curve starts to flatten, forming an "elbow" around k=3 or k=4. This suggests that choosing 3 or 4 clusters would provide a good balance between model accuracy and simplicity—beyond this point, adding more clusters offers diminishing returns in reducing inertia. Thus, based on the curve, k=3 appears to be a strong candidate for optimal clustering.

Step 3: Apply K-Means Clustering:-
Command: kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_pca['KMeans_Cluster'] = kmeans.fit_predict(data_scaled)
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_)

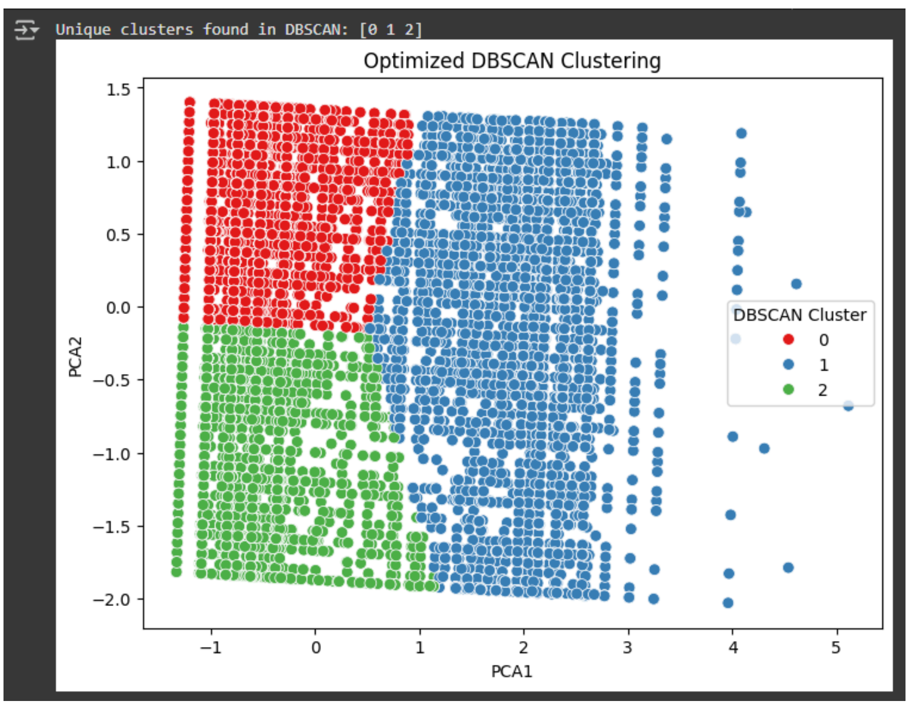



```
K-Means Centroids (in original scaled feature space):
 [[-1.4056407   1.42704481  0.05121492]
 [ 0.47824574 -0.48900124  0.74625981]
 [ 0.46254972 -0.46477085 -1.07719359]]
K-Means Inertia (Sum of Squared Distances): 242539.03233889575
```

This code performs K-Means clustering on the standardized dataset with 3 clusters, using a fixed random seed for reproducibility and n_init=10 to ensure stable results. The resulting cluster labels are added to the df_pca DataFrame for visualization. A scatter plot is then generated using the two PCA components to visually display how the data points are grouped into clusters. Finally, it prints the cluster centroids (in the original scaled feature space) and the inertia, which is the total within-cluster sum of squared distances—a measure of how well the clusters fit the data.

Step 4: Apply DBSCAN Clustering:

```
Command: 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()
```

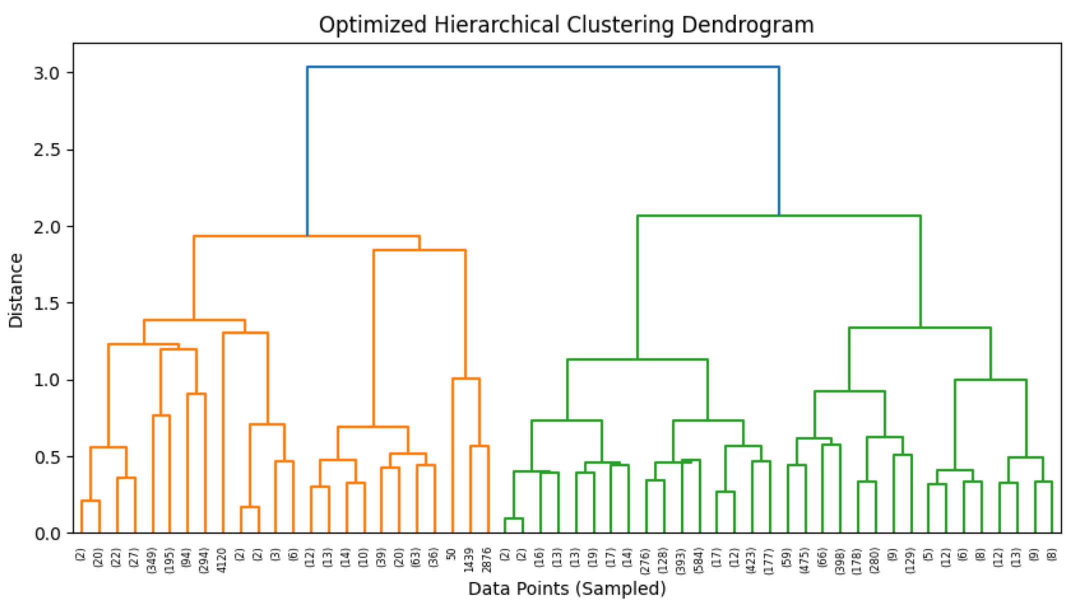


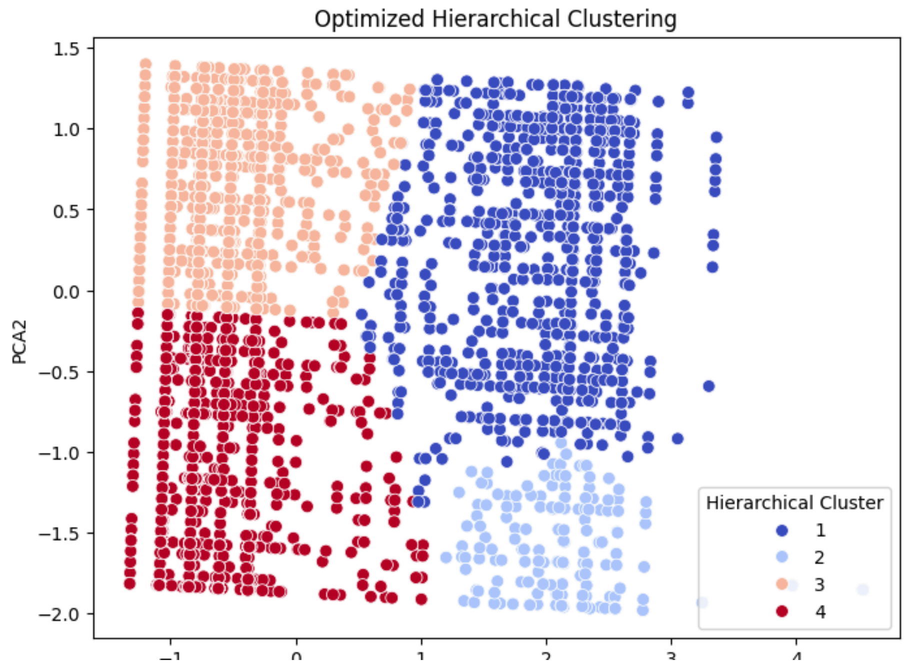
This code applies DBSCAN clustering to a reduced sample (20,000 rows) of the dataset to prevent Google Colab from crashing due to memory overload. It randomly shuffles the full PCA-transformed DataFrame (df_pca) and selects a sample to work with. DBSCAN is then applied with optimized parameters (eps=1.0, min_samples=10, n_jobs=1 to control CPU usage). After clustering, the unique cluster labels—including -1 for noise or outliers—are printed. A scatter plot is generated to visualize the clusters in 2D using the PCA components, with noise points highlighted in black. This approach ensures DBSCAN runs efficiently while still providing meaningful clustering output.

Step 5: Apply Hierarchical clustering:

Command: from sklearn.utils import shuffle

```
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=5000)  # Sample 5000 points
linkage_matrix = linkage(df_pca_sample, method='centroid')
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()
df_pca_sample['Hierarchical_Cluster'] = fcluster(linkage_matrix, t=4, criterion='maxclust')
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()
```





In this code, hierarchical clustering is performed on a random sample of 5,000 PCA-reduced data points to reduce memory usage and prevent system crashes. The 'centroid' linkage method is used, which clusters data points based on the centroid (mean) of clusters rather than individual distances, making it more efficient for larger datasets. A dendrogram is plotted with truncate_mode='level' and p=5 to visualize only the top 5 levels of the hierarchy, giving an overview without overloading the plot. Clusters are then extracted using fcluster with t=4, meaning the data is grouped into 4 final clusters. Finally, a scatter plot shows how these clusters are distributed in the reduced 2D PCA space. This method gives both a visual and analytical view of how the data groups together hierarchically.

Conclusion:

1. In this experiment, we learned how to perform different clustering algorithms.
2. K-Means clustering identified three distinct clusters but exhibited high inertia (242,539), indicating that it struggled with optimal separations, particularly in areas with dense data points.
3. DBSCAN successfully handled noise and discovered three clusters, but its clustering structure was highly dependent on parameter tuning, leading to an uneven distribution of cluster sizes.
4. Hierarchical clustering provided an interpretable dendrogram, effectively showcasing relationships between data points, but the final clustering outcome resulted in four distinct clusters, differing from the other methods.

5. Compared to K-Means, DBSCAN performed better in detecting non-uniform cluster densities, while hierarchical clustering provided more structured and detailed insights into data grouping.
6. K-Means is ideal for well-separated, structured clusters, DBSCAN excels with complex shapes and noise, and hierarchical clustering offers valuable visualization but becomes computationally expensive for large datasets.