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.

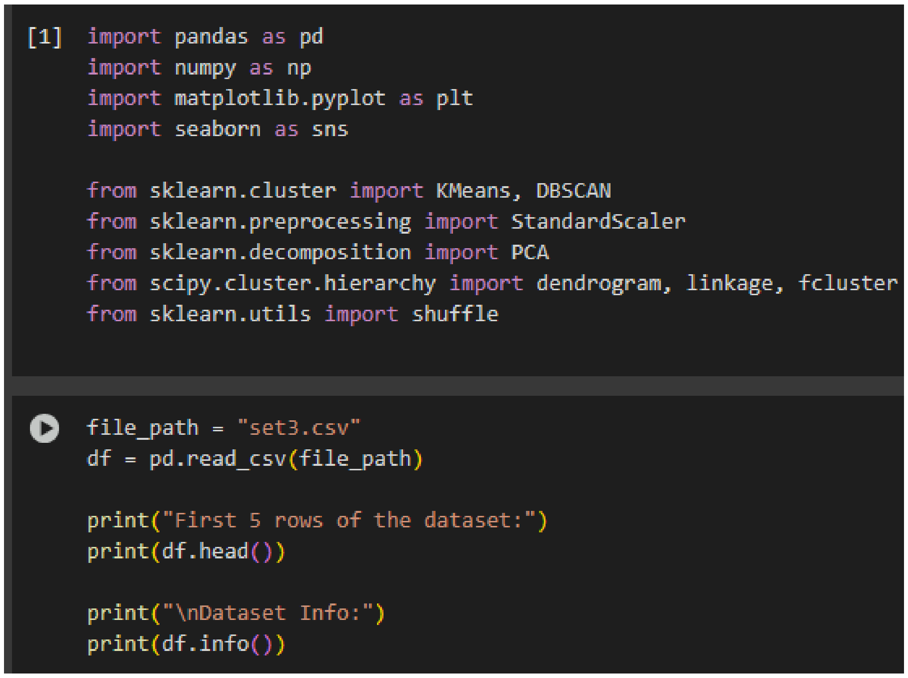
Theory:

- **K-Means Clustering:** K-Means is a centroid-based algorithm that splits data into *k* clusters by minimizing variance within each cluster. It randomly places *k* centroids, assigns points to the nearest one, and updates centroids based on the mean of assigned points. This repeats until convergence. It's fast and effective for well-separated, spherical clusters, but it doesn't handle outliers well.

- **DBSCAN Clustering:** DBSCAN groups densely packed points and labels sparse regions as noise. It uses two parameters: eps (neighborhood radius) and min_samples (minimum points to form a dense area). It can find clusters of any shape and doesn't need *k*, but it's sensitive to parameter tuning and struggles with variable densities.

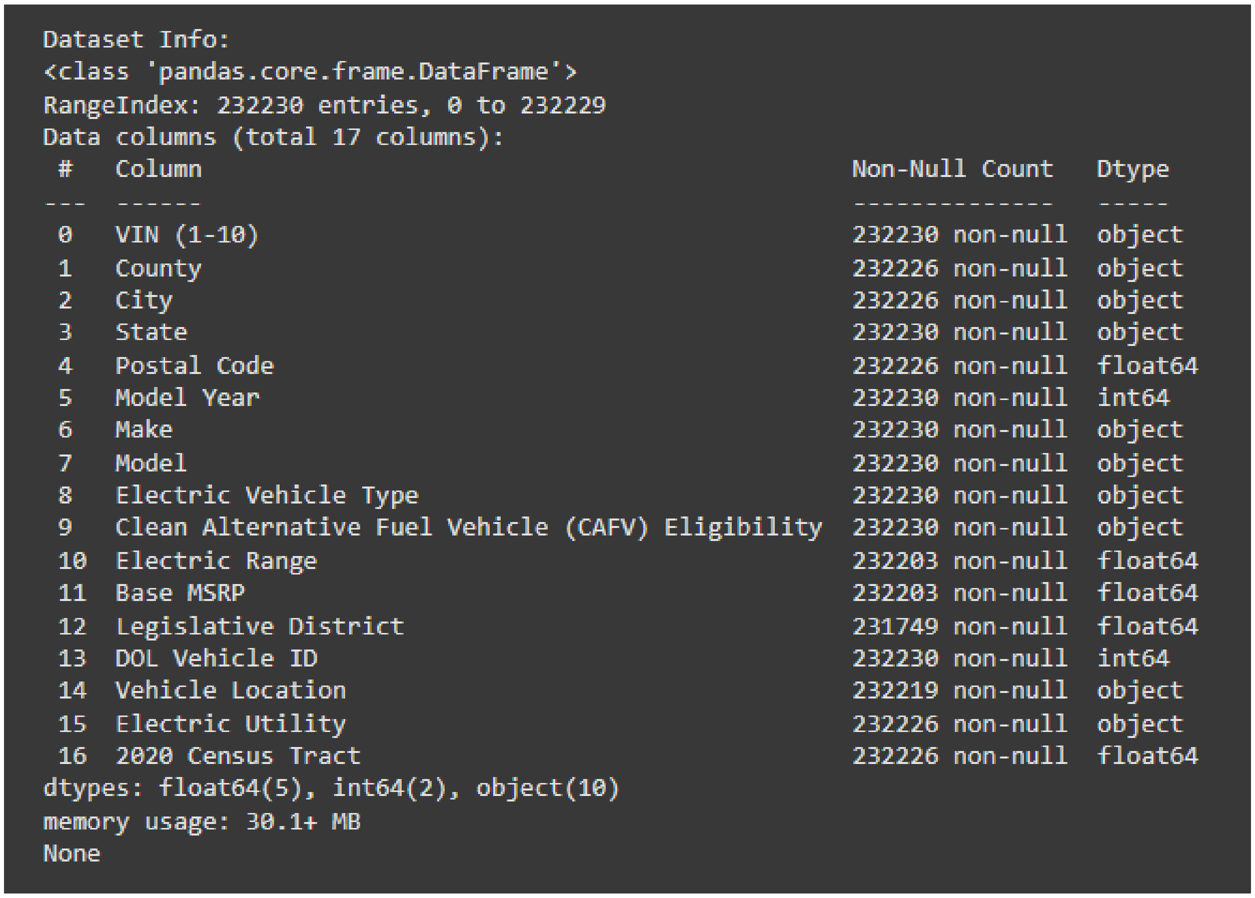
- **Hierarchical clustering**: This method builds a hierarchy of clusters using either bottom-up (agglomerative) or top-down (divisive) approaches. It doesn't need the number of clusters in advance and visualizes relationships using a dendrogram. It's good for small datasets and interpretability, but it gets slow and unstable with noise or large datasets.

Performance:



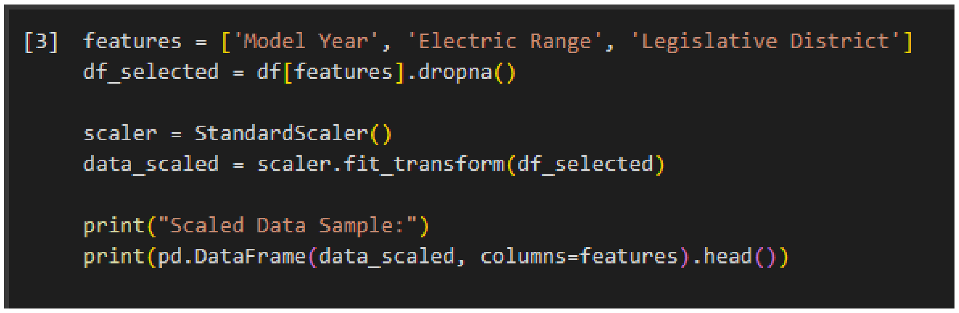
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.utils import shuffle
```

```python
file_path = "set3.csv"
df = pd.read_csv(file_path)

print("First 5 rows of the dataset:")
print(df.head())

print("\nDataset Info:")
print(df.info())
```

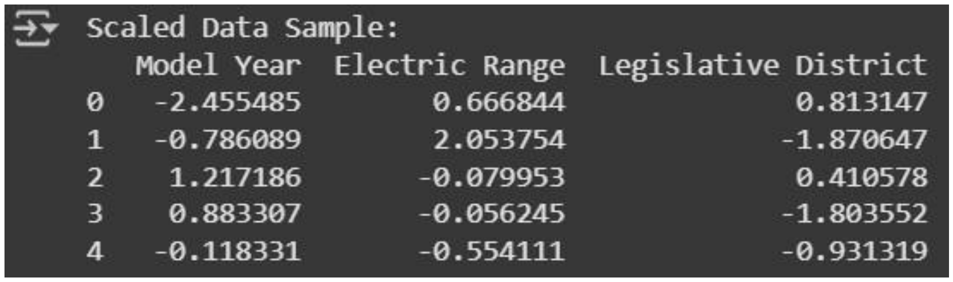


```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232230 entries, 0 to 232229
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         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
```

~~Feature~~ Selection and Preprocessing:-



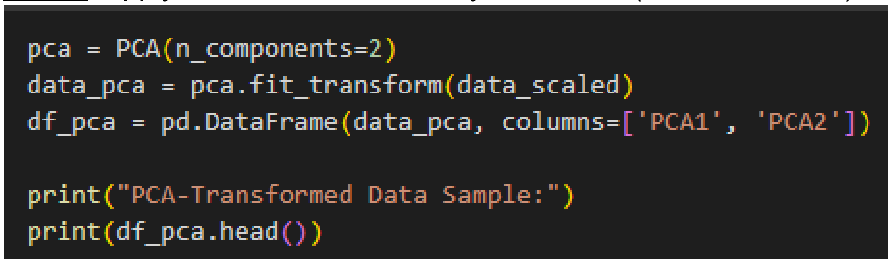
```
[3]  features = ['Model Year', 'Electric Range', 'Legislative District']
     df_selected = df[features].dropna()

     scaler = StandardScaler()
     data_scaled = scaler.fit_transform(df_selected)

     print("Scaled Data Sample:")
     print(pd.DataFrame(data_scaled, columns=features).head())
```



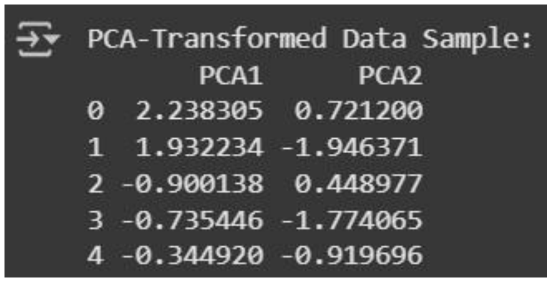
```
Scaled Data Sample:
   Model Year  Electric Range  Legislative District
0   -2.455485        0.666844              0.813147
1   -0.786089        2.053754             -1.870647
2    1.217186       -0.079953              0.410578
3    0.883307       -0.056245             -1.803552
4   -0.118331   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

The above code selects three relevant numerical features.

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



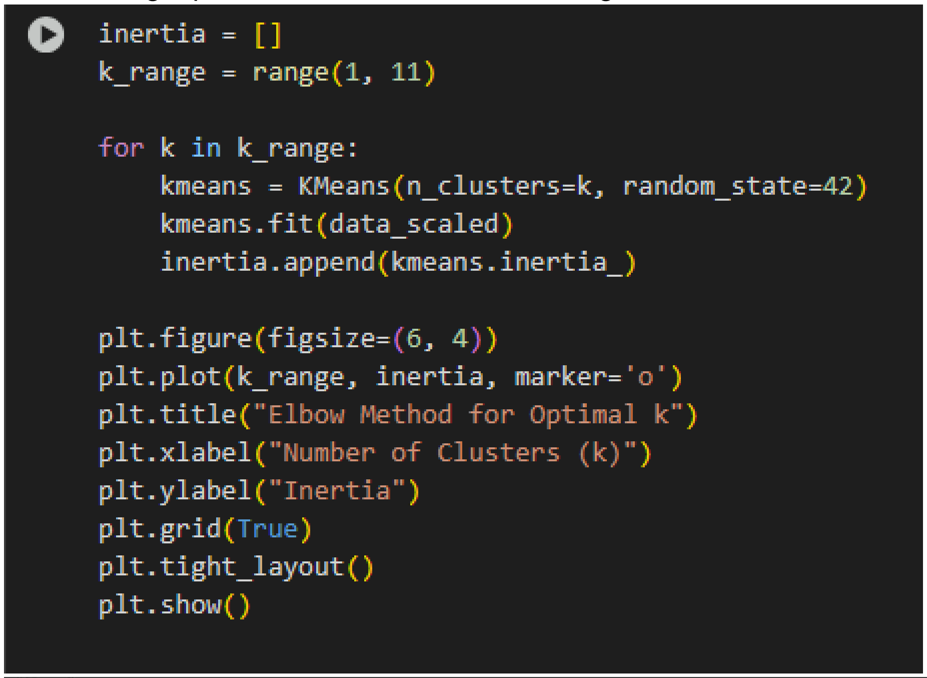
```
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
df_pca = pd.DataFrame(data_pca, columns=['PCA1', 'PCA2'])

print("PCA-Transformed Data Sample:")
print(df_pca.head())
```



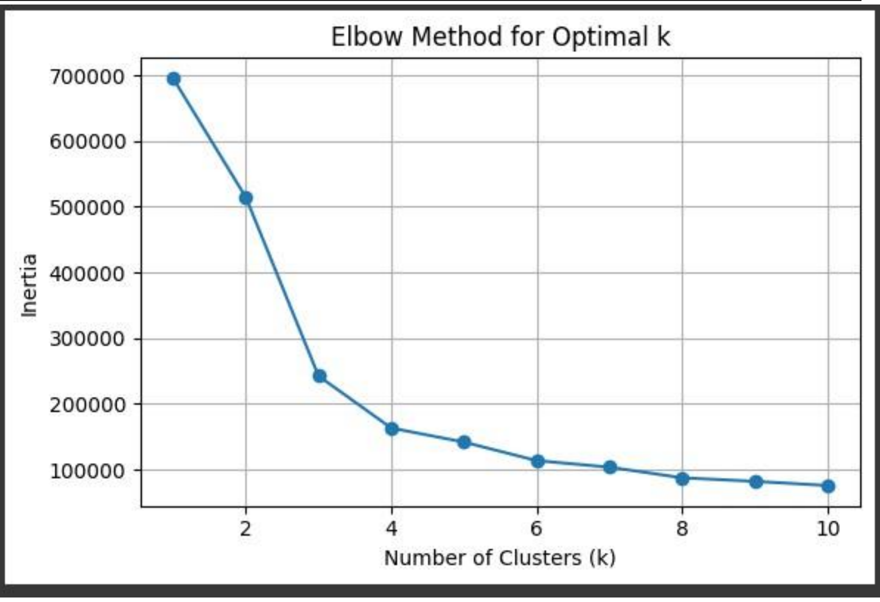
```
PCA-Transformed Data Sample:
       PCA1      PCA2
0  2.238305  0.721200
1  1.932234 -1.946371
2 -0.900138  0.448977
3 -0.735446 -1.774065
4 -0.344920 -0.919696
```

The code applies Principal Component Analysis (PCA) to reduce the standardized dataset to two principal components PCA1 and PCA2.

Determining Optimal Number of Clusters Using the Elbow Method:-



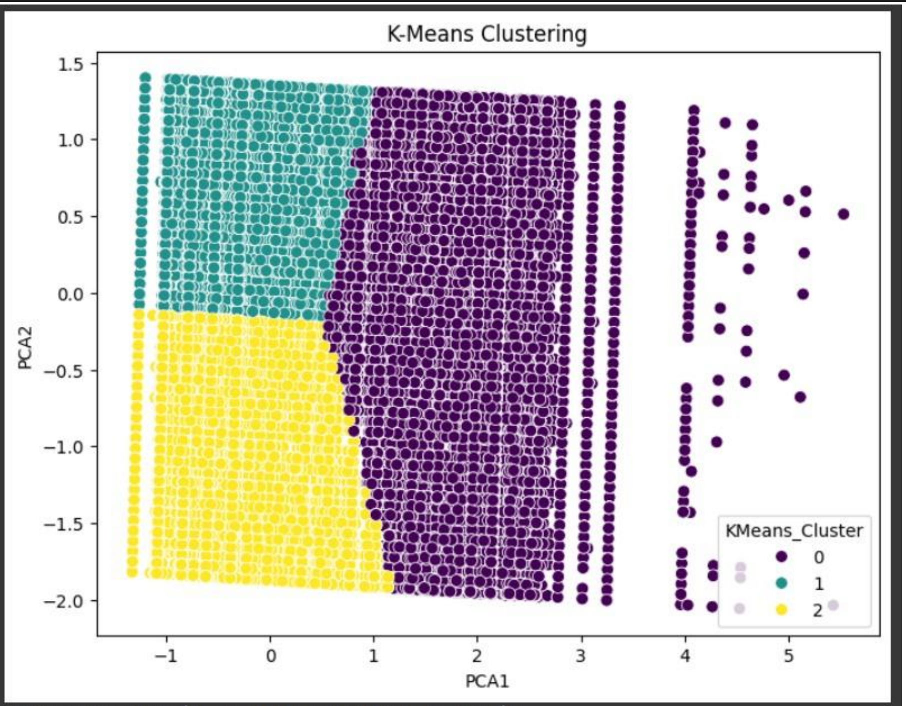
```python
inertia = []
k_range = range(1, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_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.

Apply K-Means Clustering:-

```python
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='PCA1', y='PCA2', hue='KMeans_Cluster', data=df_pca,
                palette='viridis', s=50)
plt.title('K-Means Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()

print("Centroids (scaled space):", kmeans.cluster_centers_)
print("K-Means Inertia:", kmeans.inertia_)
```

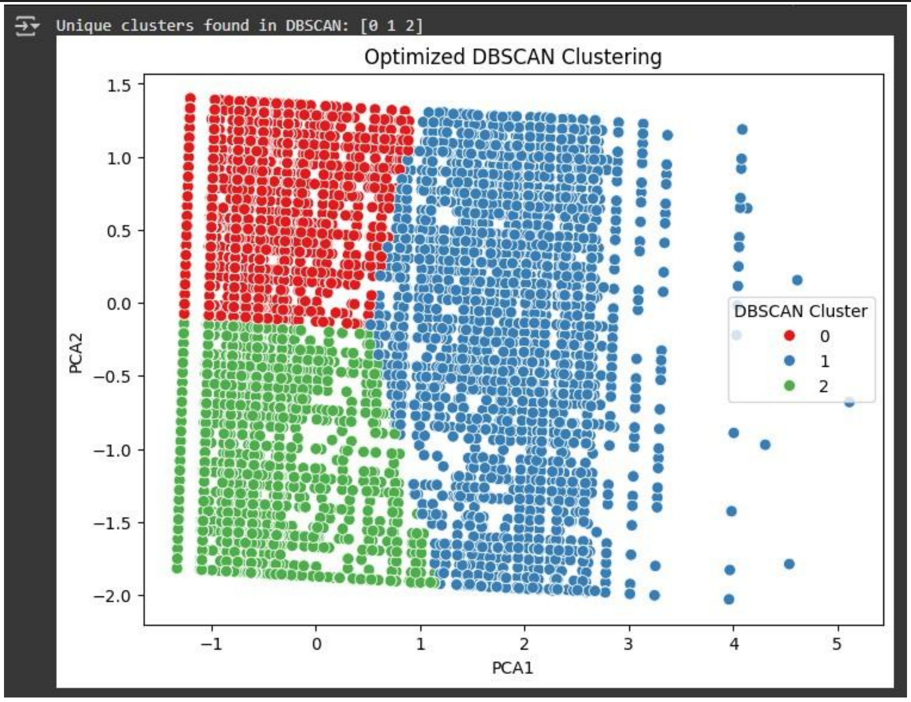


```
K-Means Centroids (in original scaled feature space):
 [[-1.4056407   1.42704481  0.05121492]
 [ 0.47824574 -0.48900124  0.74625981]
 [ 0.46254972 -0.46477085 -1.07719359]]
K-Means Inertia (Sum of Squared Distances): 242539.03233889575
```

This code performs K-Means clustering on the standardized dataset with 3 clusters, using a fixed random seed for reproducibility.

Apply DBSCAN Clustering:

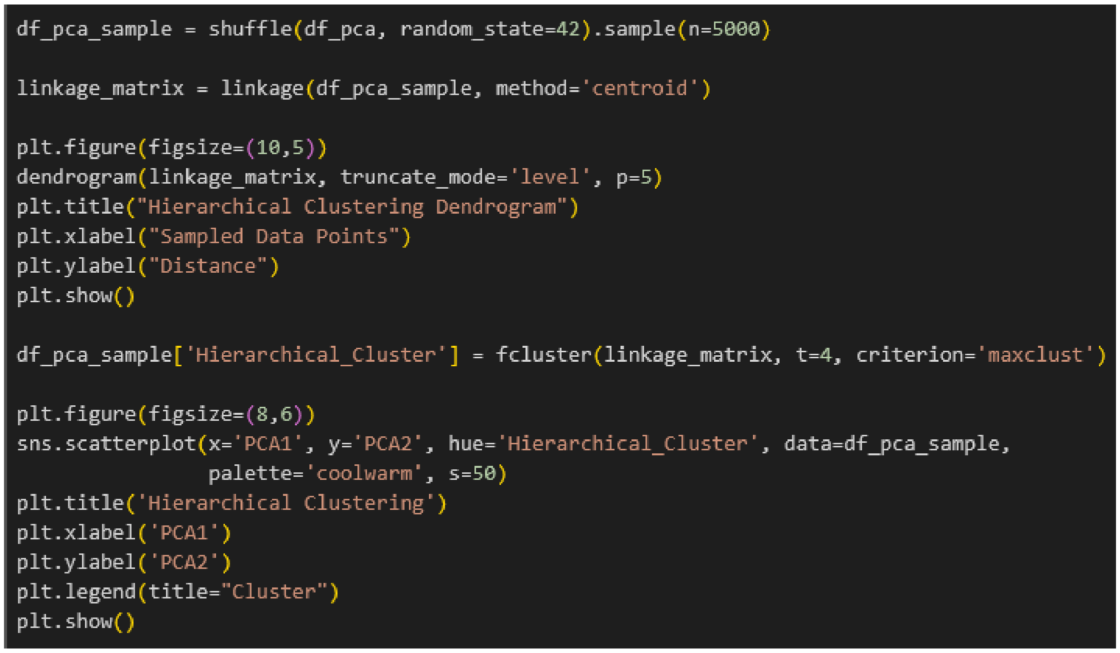
```python
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=20000)

dbscan = DBSCAN(eps=1.0, min_samples=10, n_jobs=1)
df_pca_sample['DBSCAN_Cluster'] = dbscan.fit_predict(df_pca_sample)

unique_clusters = np.unique(df_pca_sample['DBSCAN_Cluster'])
print("DBSCAN Clusters:", unique_clusters)

plt.figure(figsize=(8,6))
sns.scatterplot(x='PCA1', y='PCA2', hue='DBSCAN_Cluster', data=df_pca_sample,
                palette='Set1', s=50)

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

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



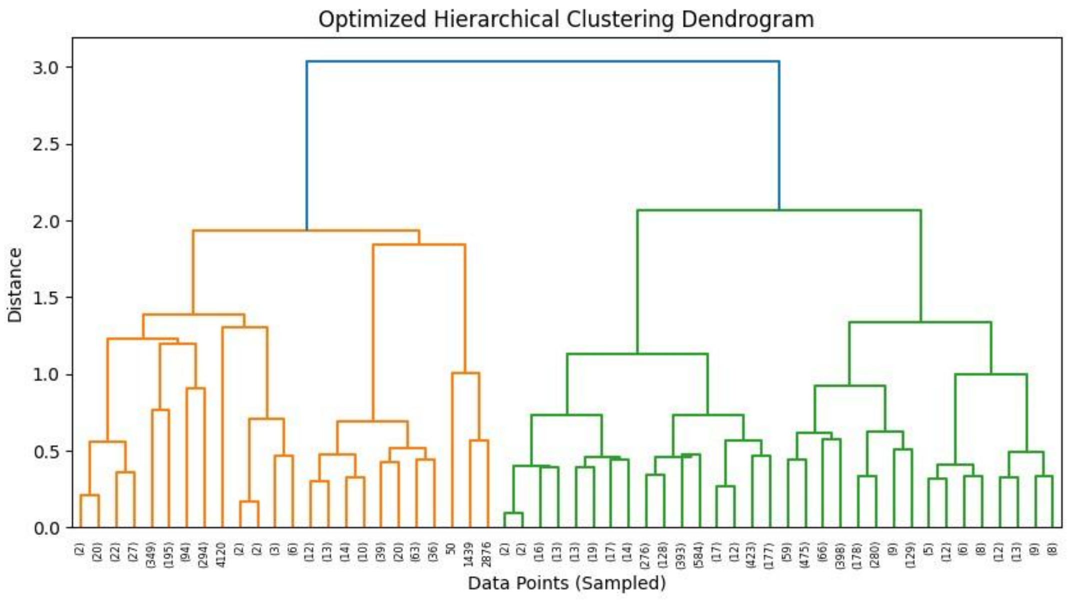
```
Unique clusters found in DBSCAN: [0 1 2]
```

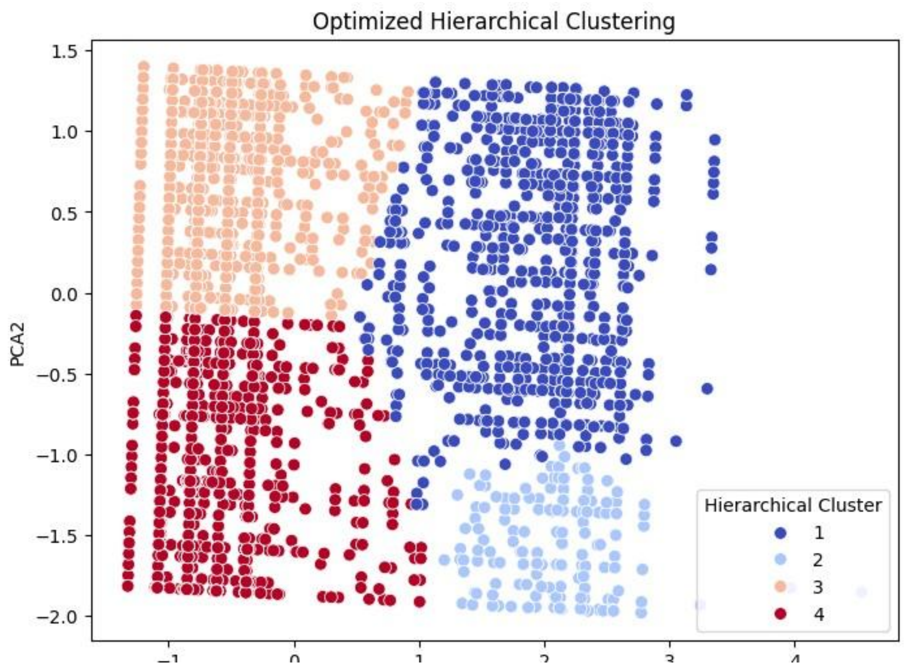
This code applies DBSCAN clustering to a reduced sample (20,000 rows) of the dataset. A scatter plot is generated to visualize the clusters in 2D using the PCA components, with noise points highlighted in black. This approach ensures DBSCAN runs efficiently while still providing meaningful clustering output.

Apply Hierarchical clustering:



```
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=5000)

linkage_matrix = linkage(df_pca_sample, method='centroid')

plt.figure(figsize=(10,5))
dendrogram(linkage_matrix, truncate_mode='level', p=5)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Sampled Data Points")
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='PCA1', y='PCA2', hue='Hierarchical_Cluster', data=df_pca_sample,
                palette='coolwarm', s=50)
plt.title('Hierarchical Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title="Cluster")
plt.show()
```





Hierarchical clustering is performed on a random sample of 5,000 PCA-reduced data points.

Conclusion:

In this experiment, we explored and applied various clustering algorithms K-Means, DBSCAN, and Hierarchical Clustering to analyze patterns in data. K-Means was able to identify three distinct clusters, but the high inertia value (242,539) suggested that it struggled to clearly separate dense regions, revealing its limitations with overlapping or irregularly shaped clusters. DBSCAN, on the other hand, handled noise well and uncovered three clusters without needing a predefined number, but its effectiveness heavily relied on fine-tuning parameters like eps and min_samples, often resulting in uneven cluster sizes. Hierarchical clustering stood out for its visual clarity through dendrograms, uncovering four distinct clusters and offering deeper insights into the relationships between points, although it deviated from the other methods in its final grouping. Overall, DBSCAN proved superior for handling non-uniform densities, hierarchical clustering delivered a more interpretable structure, and K-Means remained efficient for clean, well-separated data.