Sairam Konar **D15C** **27** **2024-2025**

Experiment No 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:

Clustering Algorithms for Unsupervised Classification

Clustering is an unsupervised machine learning technique used to group similar data points based on certain features. Below are three widely used clustering algorithms:

1. K-Means Clustering

K-Means is a centroid-based clustering algorithm that partitions data into k clusters.

Steps of K-Means Algorithm:

1. Choose the number of clusters k.
2. Initialize k cluster centroids randomly.
3. Assign each data point to the nearest centroid based on Euclidean distance.
4. Compute the new centroids as the mean of all points in each cluster.
5. Repeat steps 3 and 4 until centroids no longer change or a stopping criterion is met.

Mathematical Steps:

- Compute the distance between a point x_i and centroid C_j:

$$d(x_i, C_j) = \sqrt{\sum_{d=1}^{n}(x_{id} - C_{jd})^2}$$

- Update centroid:

$$C_j = \frac{1}{|S_j|}\sum_{x_i \in S_j} x_i$$

where S_j is the set of points assigned to cluster

2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based clustering algorithm that groups points that are closely packed together while marking outliers as noise.

Steps of DBSCAN Algorithm:

1. Select a random point P and check if it has at least MinPts neighbors within radius ε.
2. If yes, create a new cluster and expand it by adding density-reachable points.
3. If no, mark P as noise.
4. Repeat until all points are processed.

Mathematical Concepts:

- A point P is a **core point** if it has at least MinPts neighbors within ε.
- A point Q is **density-reachable** from P if $d(P,Q) \leq \varepsilon$.
- A point is **noise** if it does not belong to any cluster.

3. Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters using either **Agglomerative (bottom-up)** or **Divisive (top-down)** approaches.

Steps of Agglomerative Clustering (Bottom-Up Approach):

1. Treat each data point as its own cluster.
2. Compute the distance between all pairs of clusters.
3. Merge the two closest clusters.
4. Repeat steps 2-3 until one cluster remains.

Mathematical Concepts:

- **Single linkage:**

$$d(A,B) = \min_{a \in A, b \in B} d(a,b)$$

- **Complete linkage:**

$$d(A,B) = \max_{a \in A, b \in B} d(a,b)$$

- **Average linkage:**

$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a,b)$$

Steps :

1) **Load and Explore Data**

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
[ ] # Load dataset
    file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Clean_Dataset_Categorized.csv"
    df = pd.read_csv(file_path)
```

```
[ ] df.info()
    df.head()
```

```
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        300153 non-null  int64  
 1   airline           300153 non-null  object 
 2   flight            300153 non-null  object 
 3   source_city       300153 non-null  object 
 4   departure_time    300153 non-null  object 
 5   stops             300153 non-null  object 
 6   arrival_time      300153 non-null  object 
 7   destination_city  300153 non-null  object 
 8   class             300153 non-null  object 
 9   duration          300153 non-null  float64
 10  days_left         300153 non-null  int64  
 11  price             300153 non-null  int64  
 12  price_category    300153 non-null  object 
dtypes: float64(1), int64(3), object(9)
memory usage: 29.8+ MB
```

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	price_category
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953	Cheap
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953	Cheap
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956	Cheap
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955	Cheap
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955	Cheap

In this step, the cleaned flight fare dataset containing 300,153 entries was loaded and explored. Each entry includes details like airline, source and destination cities, departure/arrival times, flight duration, and price.

2) **Printing Missing Values**

```
df.isnull().sum()
```

	0
Unnamed: 0	0
airline	0
flight	0
source_city	0
departure_time	0
stops	0
arrival_time	0
destination_city	0
class	0
duration	0
days_left	0
price	0
price_category	0

dtype: int64

No missing values were found across the 13 columns.

```
df.describe()
```

	Unnamed: 0	duration	days_left	price
count	300153.000000	300153.000000	300153.000000	300153.000000
mean	150076.000000	12.221021	26.004751	20889.660523
std	86646.852011	7.191997	13.561004	22697.767366
min	0.000000	0.830000	1.000000	1105.000000
25%	75038.000000	6.830000	15.000000	4783.000000
50%	150076.000000	11.250000	26.000000	7425.000000
75%	225114.000000	16.170000	38.000000	42521.000000
max	300152.000000	49.830000	49.000000	123071.000000

3) Data Cleaning and Preprocessing

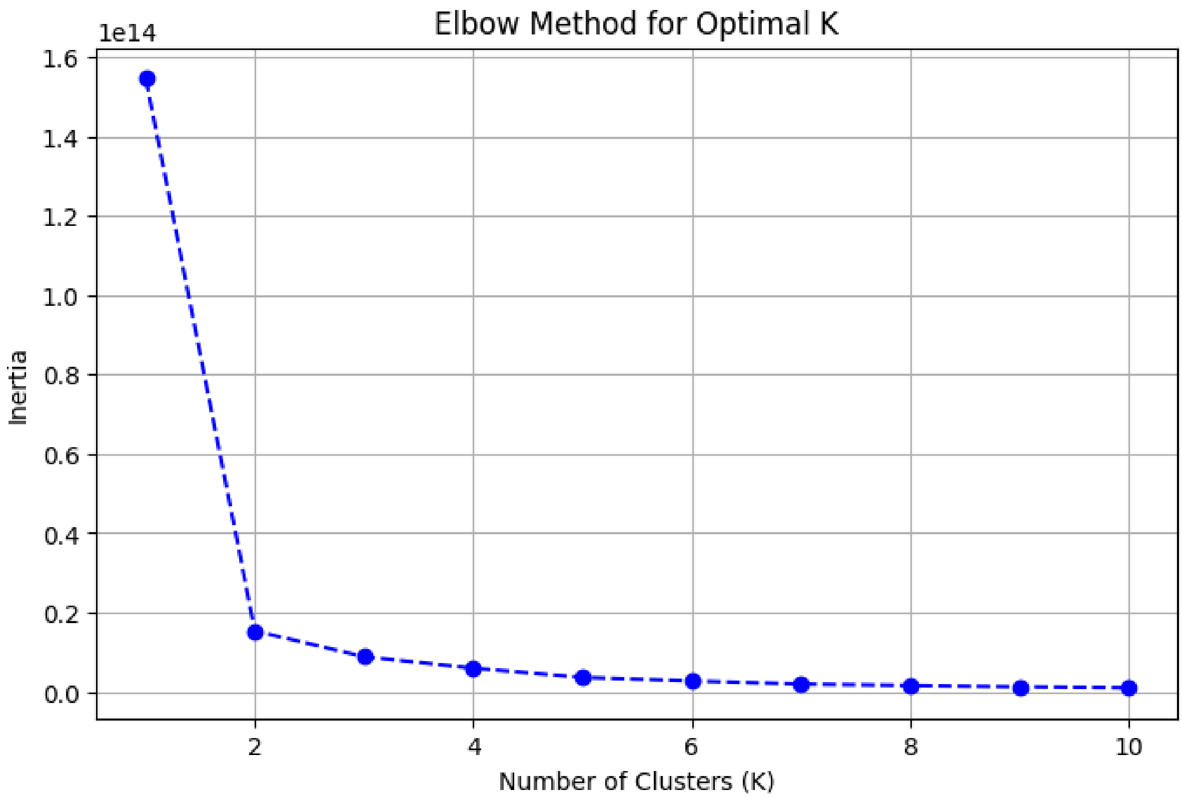
Drop Unnecessary and Convert categorical data

```
# Drop unnecessary columns
df_cleaned = df.drop(columns=['Unnamed: 0', 'flight'])

# Encode categorical columns
categorical_cols = ['airline', 'source_city', 'departure_time', 'stops',
                    'arrival_time', 'destination_city', 'class', 'price_category']

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_cleaned[col] = le.fit_transform(df_cleaned[col])
    label_encoders[col] = le
```

Unnecessary columns like 'Unnamed: 0' and 'flight' were dropped to simplify the dataset. Then, the categorical columns are encoded using Label Encoding to convert them into numerical format suitable for machine learning models. This step ensured that the dataset was clean, compact, and fully numerical, preparing it for clustering and further analysis.

4) K-Means Clustering

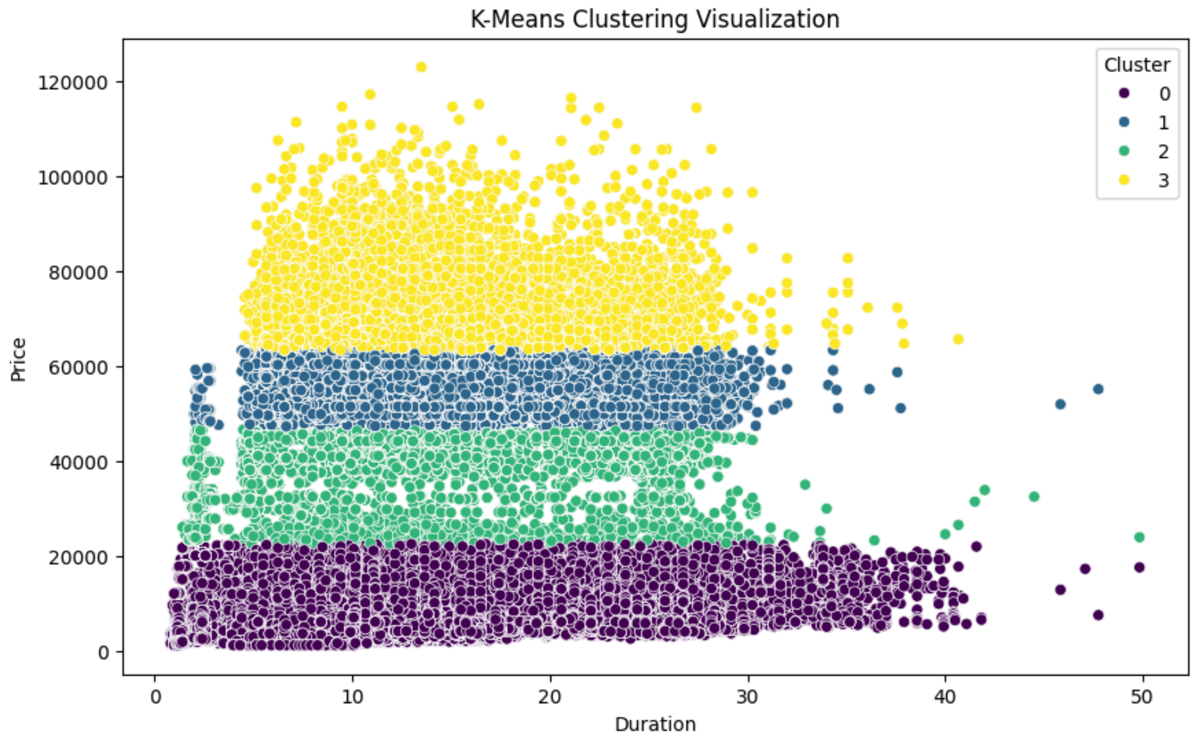
```
X = df_cleaned[['duration', 'days_left', 'price']]
inertia = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o', linestyle='--', color='b')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```



K-Means clustering was applied using the numerical features 'duration', 'days_left', and 'price'. To determine the optimal number of clusters (K), the Elbow Method was used by plotting inertia values for K ranging from 1 to 10. The elbow point in the curve appears around **K=4**, indicating that 4 clusters provide a good balance between model complexity and performance. This step is crucial for identifying distinct flight pricing patterns.

Code to Apply K-Means

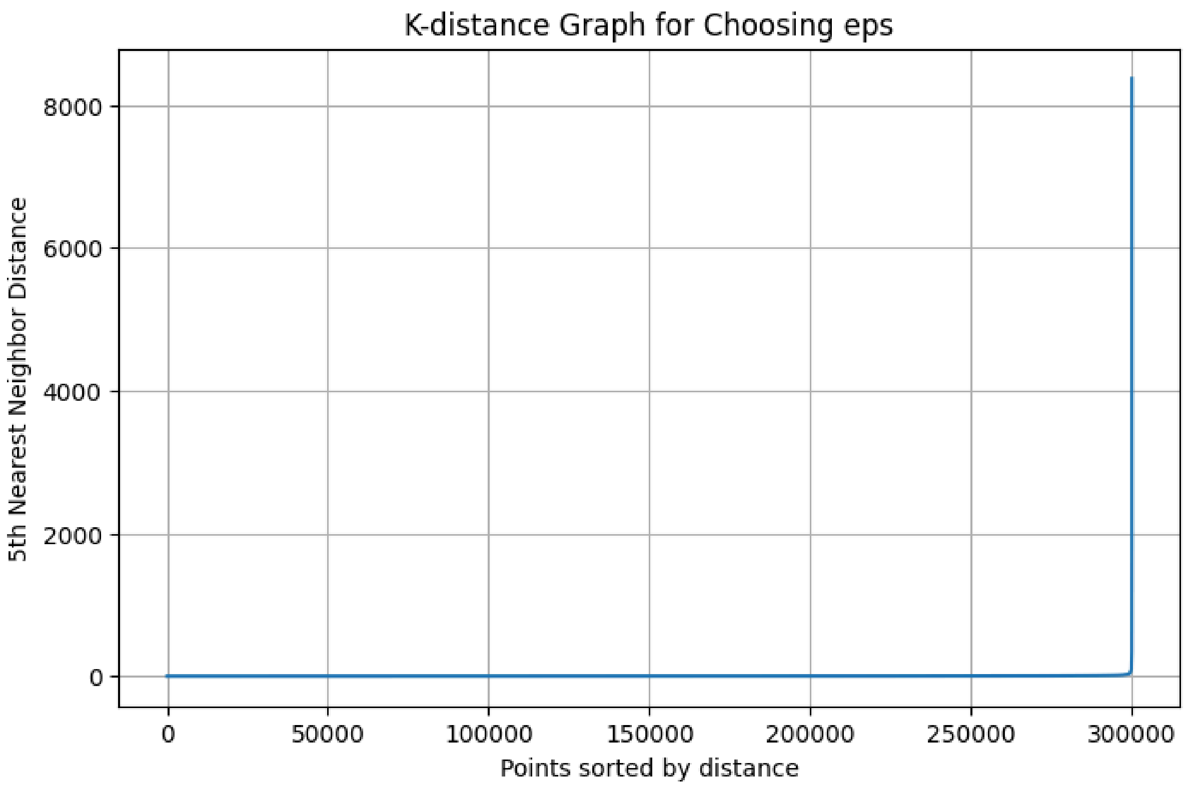
```
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_cleaned['Cluster'] = kmeans.fit_predict(X)

df_cleaned.head()

import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['duration'], y=df_cleaned['price'], hue=df_cleaned['Cluster'], palette='viridis')
plt.xlabel('Duration')
plt.ylabel('Price')
plt.title('K-Means Clustering Visualization')
plt.show()
```



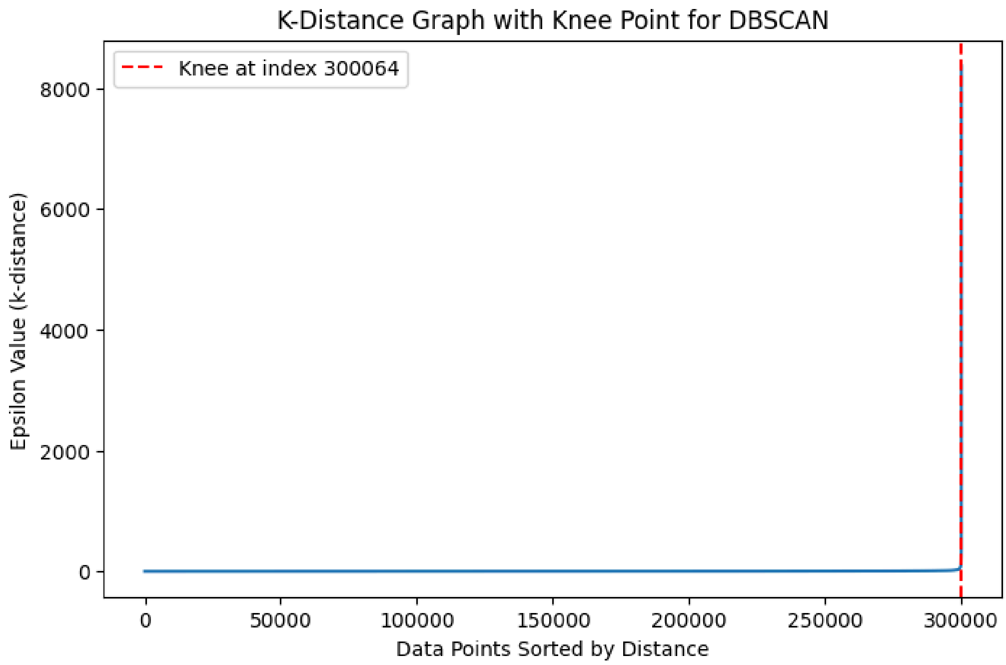
K-Means was applied with K=4 (from the elbow method) to segment flights based on duration, price, and days left. Each flight was assigned a cluster label, and the results were visualized using a scatter plot. The plot reveals four distinct price-based groupings, showing clear patterns in how flight duration and price correlate. This clustering can help identify trends in fare segmentation and assist in price prediction or customer targeting.

5) DBSCAN Clustering

```
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

X = df_cleaned[['duration', 'days_left', 'price']]

nearest_neighbors = NearestNeighbors(n_neighbors=5)
neighbors = nearest_neighbors.fit(X)
distances, indices = neighbors.kneighbors(X)

distances = np.sort(distances[:, 4], axis=0)

plt.figure(figsize=(8,5))
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("5th Nearest Neighbor Distance")
plt.title("K-distance Graph for Choosing eps")
plt.grid(True)
plt.show()
```



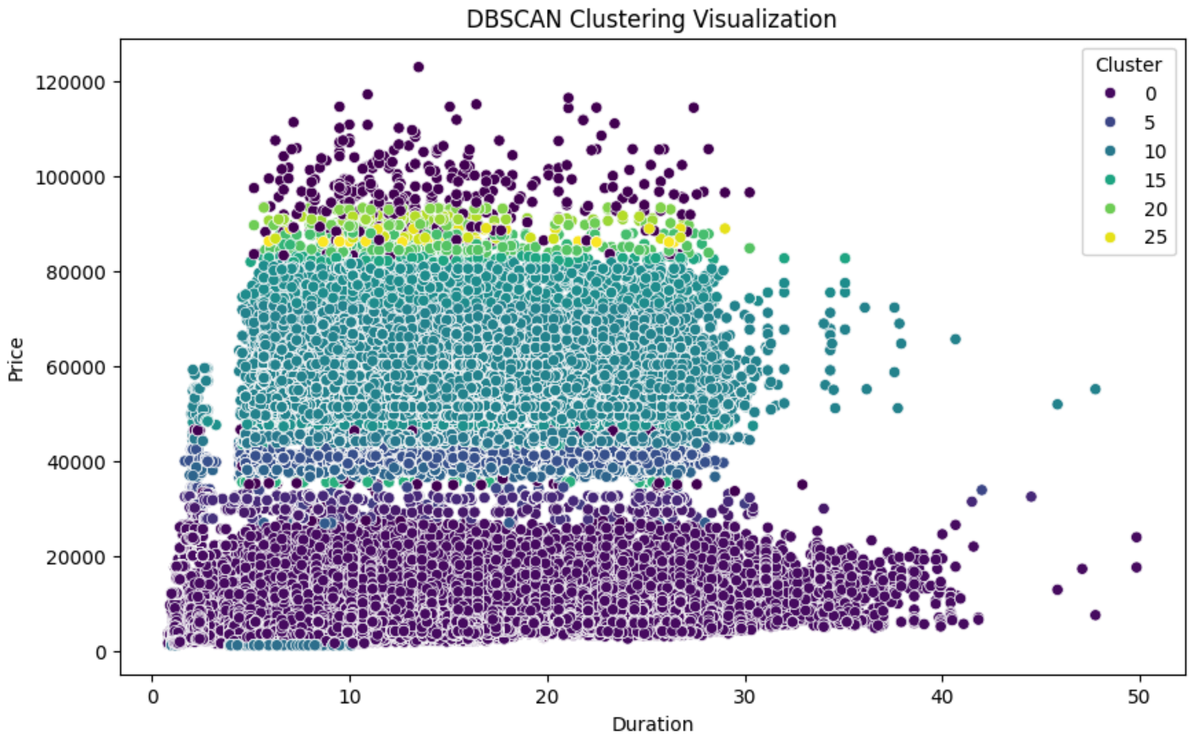

```
!pip install kneed

from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
from kneed import KneeLocator

# Compute nearest neighbors
neigh = NearestNeighbors(n_neighbors=5)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)

# Sort distances
distances = np.sort(distances[:, 4])  # 4th NN distance

# Find knee point
knee = KneeLocator(range(len(distances)), distances, curve="convex", direction="increasing")

# Plot
plt.figure(figsize=(8, 5))
plt.plot(distances)
plt.axvline(x=knee.knee, color='r', linestyle='--', label=f"Knee at index {knee.knee}")
plt.xlabel("Data Points Sorted by Distance")
plt.ylabel("Epsilon Value (k-distance)")
plt.title("K-Distance Graph with Knee Point for DBSCAN")
plt.legend()
plt.show()

# Suggested epsilon
print(f"Suggested Epsilon (ε): {distances[knee.knee]}")
```



Suggested Epsilon (ε): 168.02811937291924

```
eps_value = 168.02811937291924
min_samples_value = 26

dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
df_cleaned['DBSCAN_Cluster'] = dbscan.fit_predict(X)

print(df_cleaned['DBSCAN_Cluster'].value_counts())
print(df_cleaned['DBSCAN_Cluster'].unique())

plt.figure(figsize=(10,6))
sns.scatterplot(x=df_cleaned['duration'], y=df_cleaned['price'], hue=df_cleaned['DBSCAN_Cluster'], palette='viridis')
plt.xlabel('Duration')
plt.ylabel('Price')
plt.title('DBSCAN Clustering Visualization')
plt.legend(title="Cluster")
plt.show()
```

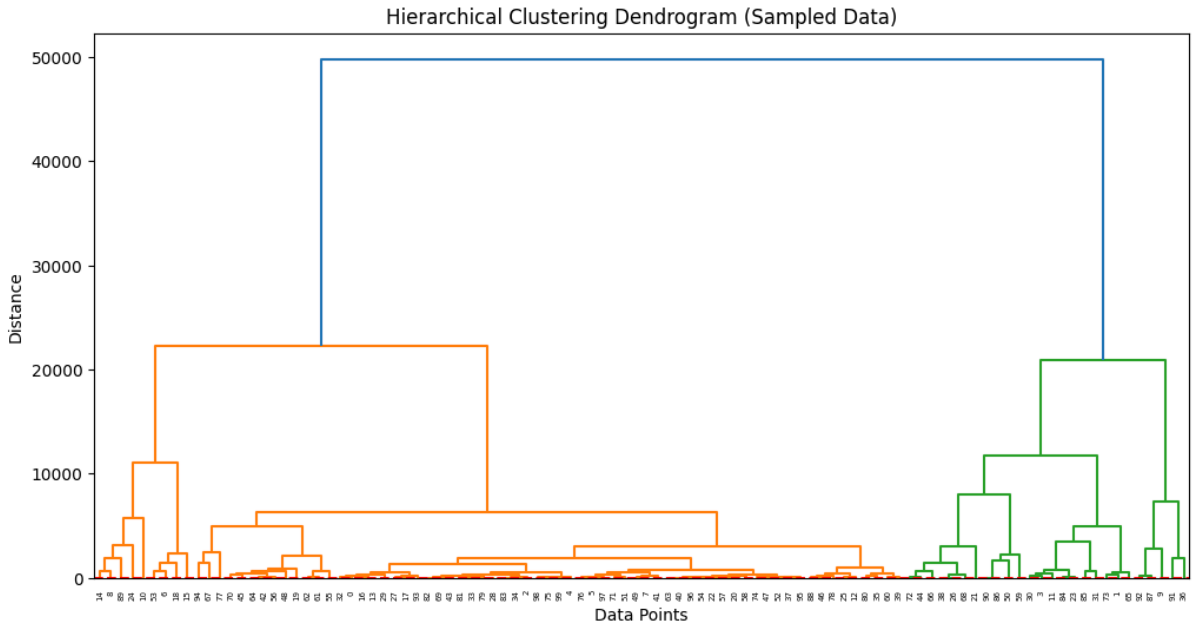
```
DBSCAN_Cluster
 0     210014
 11     59594
 10      7710
 6       6765
 8       3279
 12      3162
 2       2290
 13      1873
 1       1179
 3       1011
 5        502
-1        492
 16       428
 19       379
 9        307
 14       288
 15       275
 20       125
 18        76
 4         74
 22        73
 24        64
 17        51
 21        40
 23        32
 26        28
 25        27
 7         15
Name: count, dtype: int64
[ 0 -1  1  2  3  4  7  5  6  8  9 10 11 12 13 14 15 17 26 24 23 16 18 19
 20 21 22 25]
```



DBSCAN clustering was applied using an optimal epsilon value of **168.03**, determined from the k-distance graph. With min_samples set to 26, the algorithm detected **27 clusters**, including a few outliers labeled as **-1**. Most data points were grouped into dense clusters, clearly visible in the visualization using 'duration' and 'price'. The method effectively identified natural groupings in the data without needing the number of clusters in advance.

6) Hierarchical Clustering

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

df_sampled = df_cleaned.sample(n=100, random_state=42)
X_sampled = df_sampled[['duration', 'days_left', 'price']]

linked = linkage(X_sampled, method='average')

plt.figure(figsize=(12, 6))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=False)
plt.axhline(y=8, color='r', linestyle='--')
plt.title('Hierarchical Clustering Dendrogram (Sampled Data)')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```



Hierarchical clustering was applied to a random sample of 100 flight records using average linkage. The dendrogram shows how data points are progressively merged based on their similarity in duration, price, and days left. A horizontal cut at distance = 8 (red line) suggests the presence of **3–4 optimal clusters**, supporting the K-Means result. This technique provides a visual understanding of cluster hierarchy and the similarity between data points.

Conclusion:

In this project, we implemented and analyzed three clustering algorithms—K-Means, DBSCAN, and Hierarchical Clustering—for unsupervised classification, each applied to a cleaned dataset and visualized through scatter plots and dendrograms. K-Means efficiently grouped data based on centroid proximity but required predefined cluster numbers, while Hierarchical Clustering revealed nested structures through dendrograms, offering deeper insights into data hierarchy. DBSCAN excelled in identifying clusters of varying densities and detecting outliers without prior assumptions about cluster count. The comparison highlighted K-Means' simplicity and speed, Hierarchical Clustering's interpretability, and DBSCAN's robustness to noise, collectively demonstrating clustering's effectiveness in uncovering hidden patterns within unlabeled data.