Name : Advik Hegde **Div : D15C** **Roll No. : 15**

AIDS 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 :

Clustering Overview
Clustering is a form of unsupervised machine learning, where the model learns patterns in data that lack predefined labels. Rather than using target outcomes to guide learning, clustering identifies underlying structures and inherent groupings within datasets. The goal is to categorize data points into collections (clusters) such that items in the same group are more alike to each other than to those in other groups. For example, a visual plot might clearly reveal three distinct clusters where data points are closely packed, indicating they share certain attributes.

Practical Applications of Clustering
Clustering has wide-ranging use cases across different industries:

1. **Marketing** – Helps segment consumers based on purchasing behavior or preferences.

2. **Biological Research** – Aids in classifying species of flora and fauna.

3. **Library Management** – Books can be grouped by themes, topics, or genres.

4. **Insurance Sector** – Useful for profiling clients, detecting policy anomalies, or spotting fraud.

5. **Urban Development** – Assists in zoning properties based on value or location.

6. **Seismology** – Helps isolate earthquake-prone regions using historical tremor data.

Types of Clustering Techniques

When picking a clustering method, it's essential to consider the data volume and structure. Some algorithms involve pairwise similarity checks, resulting in a time complexity of $O(n^2)$, which may not scale well with large datasets. Various clustering methods are classified as follows:

1. **Density-Oriented Techniques**
   These techniques identify groups as areas of higher point density separated by sparser regions. They offer flexibility and can merge neighboring clusters effectively. Examples include DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and OPTICS.

2. **Hierarchical Approaches**
   Clusters are constructed based on hierarchical relationships, forming a tree-like structure. This is split into:

   - Agglomerative: Starts with individual points and merges them upward.

   - Divisive: Begins with the full dataset and splits it recursively.
     Tools like CURE and BIRCH fall under this category.

3. **Partition-Based Techniques**
   These methods divide the data into a pre-set number of clusters. They focus on optimizing a distance metric to ensure tight clusters. K-Means and CLARANS are popular algorithms in this group.

4. **Grid-Oriented Methods**
   Here, the data space is divided into a grid structure, with operations applied on these cells. This approach is efficient and scales independently of data size. Notable algorithms include STING, CLIQUE, and WaveCluster.

Dataset Description :

The dataset used in ‘cleaned_ciy_hour.csv’,contains data about various pollutants that are used to determine the value of AQI for major Indian cities. It has about 20000 entries and contains data for a large number of cities like Mumbai,Delhi,Chennai, Jaipur etc.

In this case , we are using K - means clustering :

1. K-Means Clustering

Definition:
K-Means is a popular **unsupervised learning algorithm** used for grouping data into **k distinct, non-overlapping clusters**. The main idea is to minimize the distance between the data points and their respective cluster centroids (means).

How It Works:

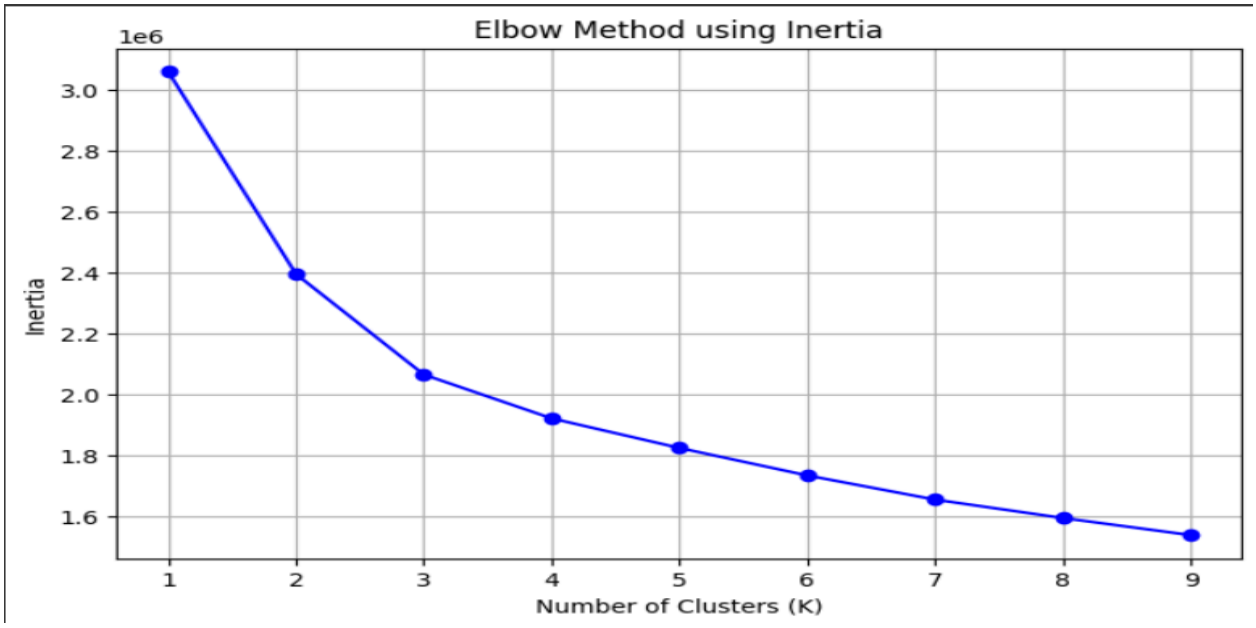
1. **Choose the number of clusters (k).**
2. **Initialize k centroids** randomly in the feature space.
3. **Assign each data point** to the nearest centroid (using distance measures like Euclidean distance).
4. **Recompute the centroid** of each cluster by calculating the mean of all points in that cluster.
5. **Repeat steps 3–4** until the centroids stop changing significantly (i.e., convergence is achieved).

Steps :

First, missing or null values are handled by either removing the affected records or replacing them with statistical estimates like mean or median. Then, irrelevant or redundant attributes are eliminated to reduce noise and focus on important features. The data is also checked for inconsistent formats or incorrect entries, which are corrected or standardized. After that, normalization or scaling is applied to ensure all numerical features are within the same range, which helps algorithms treat each feature equally.

After this, we move forward with k-means clustering for which the first step is to make use of the elbow method to identify the ideal number of clusters that the dataset should be divided into.

```
# Finding optimal number of clusters using the Elbow Method
inertia_values = []
K_range = range(1, 10)  # Trying K from 1 to 9

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia_values.append(kmeans.inertia_)  # Store inertia (sum of squared distances)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia_values, marker='o', linestyle='-', color="b")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method using Inertia")
plt.grid()
plt.show()
```

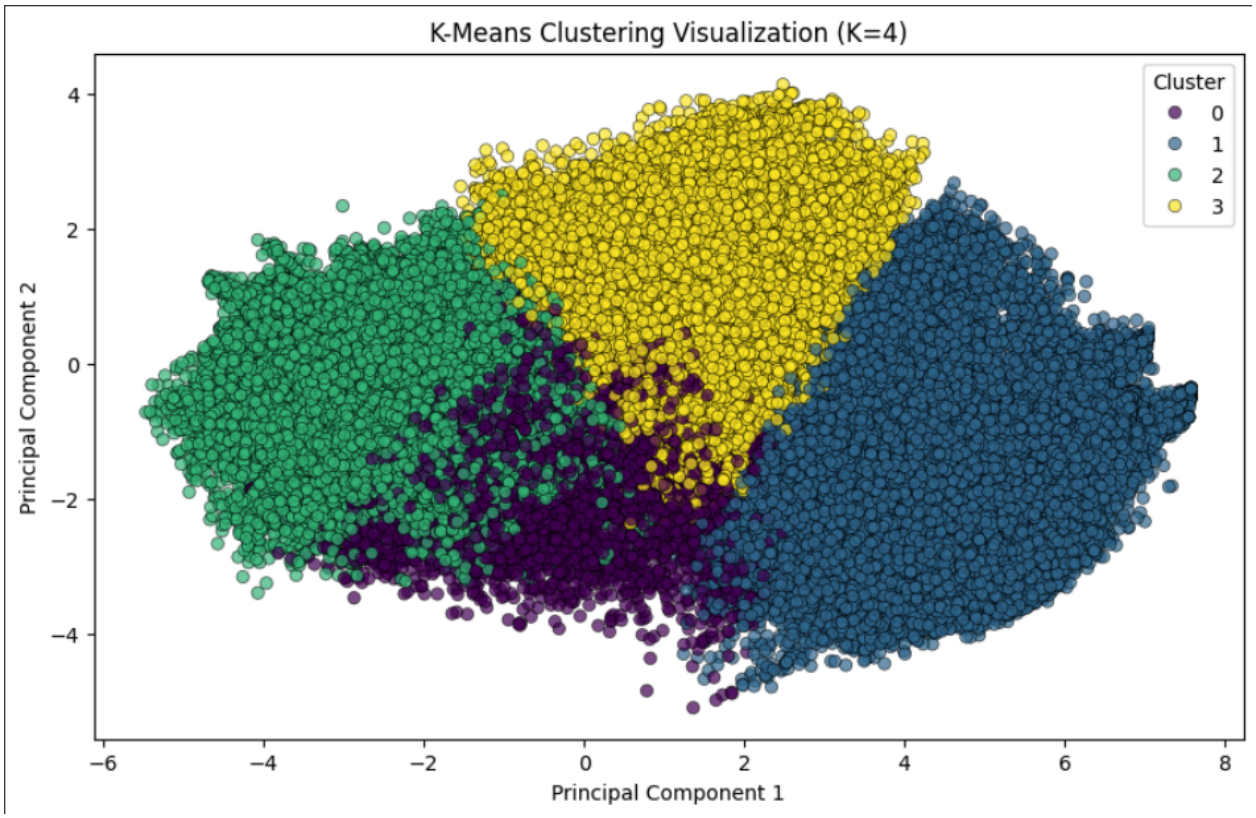


From this plot , we can identify that the ideal number of clusters for our dataset is 4 as the graph becomes smooth beyond that point and no longer bends that sharply .

So we move ahead and divide the dataset into 4 clusters :

The results of clustering are plotted below after applying PCA (Principal Component Analysis) which helps in numerosity and dimension reduction to visualize the results of clustering more clearly.

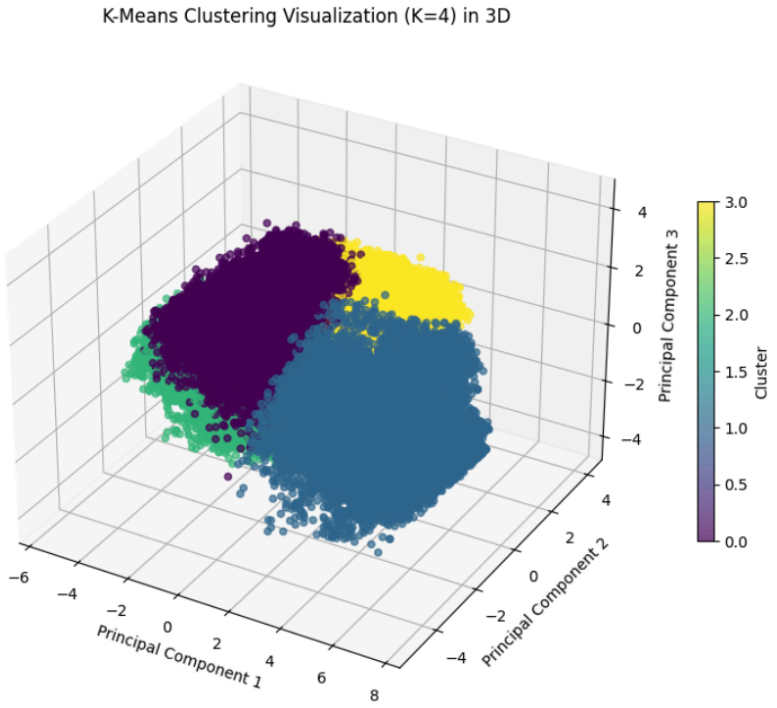
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensions to 2D
pca = PCA(n_components=2)
pca_features = pca.fit_transform(df_scaled.drop(columns=["Cluster_K4"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2"])
df_pca["Cluster"] = df_scaled["Cluster_K4"]

# Plot the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x="PC1", y="PC2", hue=df_pca["Cluster"], palette="viridis", data=df_pca, alpha=0.7, edgecolor="k")
plt.title("K-Means Clustering Visualization (K=4)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.show()
```



After visualizing clustering results in 2D, plotting them in 3D provides a deeper understanding of how data points are grouped across three features. This involves selecting three relevant numerical attributes from the dataset and using a 3D scatter plot, where each axis represents one attribute. The data points are then plotted in 3D space, with colors or shapes indicating their cluster assignments. This helps in visually

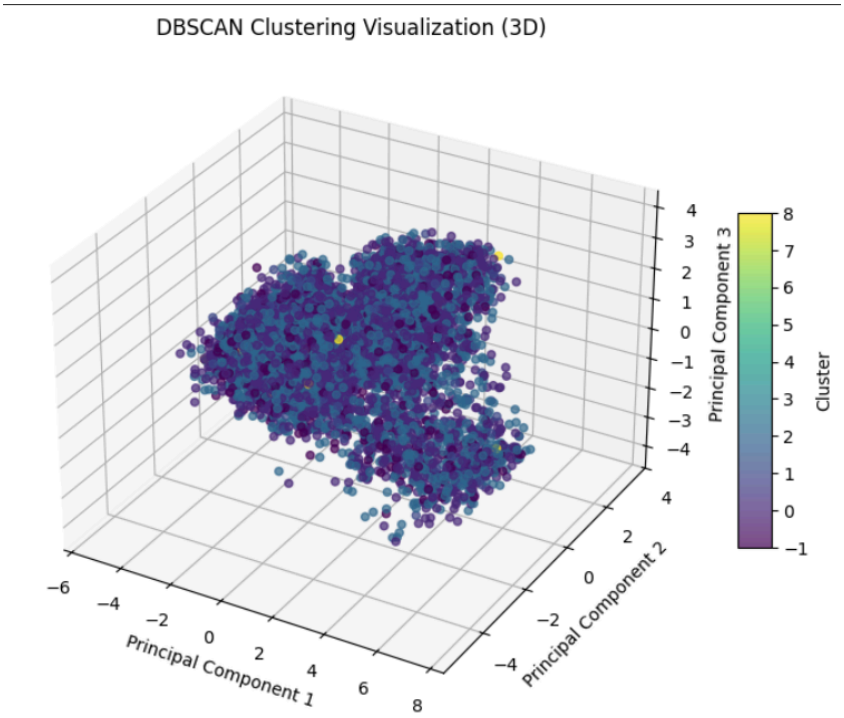
assessing cluster separation, compactness, and overlap that may not be fully visible in 2D plots.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Apply PCA to reduce dimensions to 3D for K=4 clusters
pca = PCA(n_components=3)
pca_features = pca.fit_transform(df_scaled.drop(columns=["Cluster_K4"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2", "PC3"])
df_pca["Cluster"] = df_scaled["Cluster_K4"]  # Use K=4 clusters

# Plot the clusters in 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot without outlines (edgecolors=None)
scatter = ax.scatter(df_pca["PC1"], df_pca["PC2"], df_pca["PC3"], c=df_pca["Cluster"], cmap="viridis", alpha=0.7)

# Labels and title
ax.set_title("K-Means Clustering Visualization (K=4) in 3D")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")

# Colorbar for clusters
cbar = plt.colorbar(scatter, ax=ax, shrink=0.5)
cbar.set_label("Cluster")

plt.show()
```



As the next step after performing K-Means clustering, DBSCAN clustering is applied to the same dataset using three selected numerical features to explore density-based grouping. Unlike K-Means, which assumes spherical clusters and a fixed number of clusters, DBSCAN identifies clusters based on the density of data points, allowing it to detect irregularly shaped clusters and outliers. The three attributes used are plotted in a 3D scatter plot, with each point colored according to its DBSCAN-assigned cluster or marked as noise. This provides a more flexible view of the data structure, helping to compare how DBSCAN captures natural groupings differently than K-Means in a 3-dimensional space.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Apply PCA to reduce dimensions to 3D
pca = PCA(n_components=3)
pca_features = pca.fit_transform(df_sample.drop(columns=["Cluster"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2", "PC3"])
df_pca["Cluster"] = df_sample["Cluster"]

# Plot the clusters in 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")

# Scatter plot for clusters
scatter = ax.scatter(df_pca["PC1"], df_pca["PC2"], df_pca["PC3"],
                     c=df_pca["Cluster"], cmap="viridis", alpha=0.7)

# Labels and title
ax.set_title("DBSCAN Clustering Visualization (3D)")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")

# Add color bar
legend1 = fig.colorbar(scatter, ax=ax, shrink=0.5, aspect=10)
legend1.set_label("Cluster")

plt.show()
```



Conclusion:

In this experiment, we implemented and compared two popular unsupervised clustering algorithms: **K-Means** and **DBSCAN**, using the adult.csv dataset.

- **K-Means** clustering required us to select the number of clusters (k) using the **Elbow Method**, which helped in identifying the optimal k by observing the point where WCSS started to level off.
- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) automatically detected clusters based on data density, without requiring k. It also identified outliers (labeled as -1), which K-Means cannot do.

This comparison highlighted the strengths of both algorithms — **K-Means** works well for well-separated spherical clusters, while **DBSCAN** is more robust in handling **noise and arbitrary-shaped clusters**, making it suitable for more complex data distributions.