Name: Dev Gaonkar **Div:** D15C **Rollno:** 12

Experiment 7

Aim: To implement different clustering algorithms.

Theory:
Clustering is an unsupervised machine learning technique used to group similar data points together. The objective is to discover natural groupings within a dataset without prior knowledge of class labels. Clustering is widely used in customer segmentation, anomaly detection, image segmentation, and bioinformatics.

Types of Clustering

1. Partition-based Clustering (e.g., K-Means)
    - Divides data into a predefined number of clusters.
    - Each data point belongs to exactly one cluster.
2. Density-based Clustering (e.g., DBSCAN)
    - Forms clusters based on dense regions in the data.
    - Can identify clusters of arbitrary shape and detect noise.
3. Hierarchical Clustering
    - Builds a tree of clusters using a bottom-up (agglomerative) or top-down (divisive) approach.
4. Model-based Clustering
    - Assumes data is generated by a mixture of underlying probability distributions (e.g., Gaussian Mixture Models).

K Means Clustering:
K-Means is a partition-based clustering algorithm that divides data into K clusters.

Algorithm Steps:

1. Choose the number of clusters, K.
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid.
4. Update centroids by computing the mean of points in each cluster.
5. Repeat steps 3 & 4 until convergence (when centroids stop changing significantly).

Key Considerations:
Choosing K: The Elbow method and Silhouette Score help determine the optimal number of clusters.
Limitations: Sensitive to initial centroid placement and does not work well with non-spherical clusters or varying densities.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
DBSCAN is a density-based clustering algorithm that groups points closely packed together while marking low-density points as noise.

Algorithm Steps:

1. Set parameters:
2. ε (epsilon): Maximum distance to be considered a neighbor.
3. MinPts: Minimum points required to form a dense region.
4. Choose an unvisited point and determine its ε-neighborhood.
5. If the point has at least MinPts neighbors, a new cluster is formed.
6. Expand the cluster by adding density-reachable points.
7. Repeat for all points, marking outliers as noise.

Advantages:

- Handles clusters of arbitrary shape.
- Automatically detects noise and outliers.
- No need to specify the number of clusters.

Disadvantages:

- Choosing optimal ε and MinPts is challenging.
- Struggles with varying density clusters.

Steps:

1. **Load and preprocess the dataset.**

Dataset Shape: (235394, 22)
First 5 rows:

	Datetime	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	...	AQI	AQI_Bucket
0	2015-01-01 01:00:00	0.335245	-0.20687	-1.947838	0.927657	0.617732	-0.208955	-0.313643	0.422827	-0.288155	...	0.459567	1.0
1	2015-01-01 02:00:00	0.335245	-0.20687	-2.343506	0.456358	-0.226460	-0.208955	-0.435624	0.422827	-0.288155	...	0.459567	1.0
2	2015-01-01 03:00:00	0.335245	-0.20687	-2.343506	-0.004131	-0.665298	-0.208955	-0.428156	0.422827	-0.288155	...	0.459567	1.0
3	2015-01-01 04:00:00	0.335245	-0.20687	-2.343506	-0.206178	-0.760676	-0.208955	-0.400772	0.422827	1.589661	...	0.459567	1.0
4	2015-01-01 05:00:00	0.335245	-0.20687	-2.343506	-0.329610	-0.829165	-0.208955	-0.423177	0.422827	-0.288155	...	0.459567	1.0

5 rows × 22 columns

City_Aizawl	City_Amaravati	City_Amritsar	City_Bengaluru	City_Bhopal	City_Brajrajnagar	City_Chandigarh	City_Chennai
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False

The dataset represents **air quality measurements**, containing:

- **Temporal Information**: Datetime (Timestamp of the recorded data).
- **Pollutant Concentrations**: PM2.5, PM10, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, Toluene (Air pollutants measured in the environment).
- **Air Quality Index (AQI)**: AQI, AQI_Bucket (Overall air quality rating based on pollutant levels).
- **City-wise Presence**: City_Aizawl, City_Amaravati, City_Amritsar, City_Bengaluru, City_Bhopal, City_Brajrajnagar, City_Chandigarh, City_Chennai (Binary indicators showing if the data belongs to a specific city).

2. **Elbow method for number of clusters**

The Elbow Method plot helps determine the optimal number of clusters (K) in K-Means by plotting inertia (sum of squared distances to cluster centers) against different K values. The "elbow" point, where inertia reduction slows significantly, indicates the ideal K.

Formula to calculate:

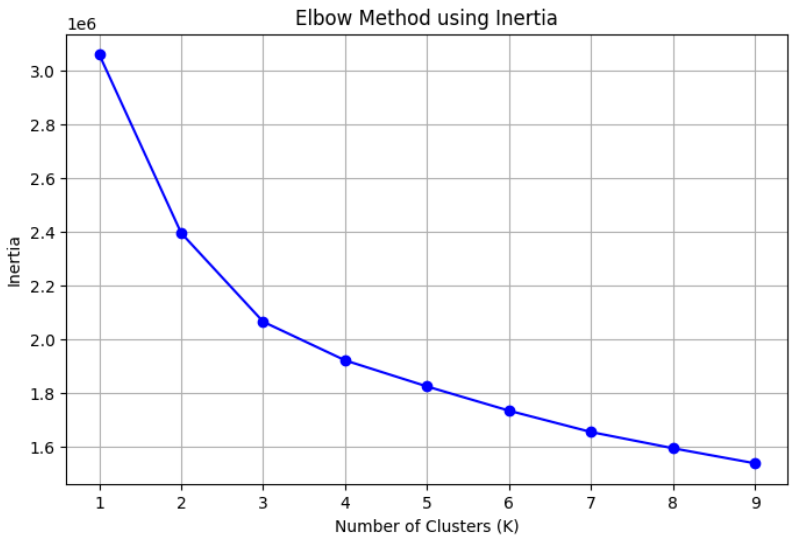
$$WCSS = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

where,

- Ci = Cluster i
- μi = Centroid of cluster Ci
- || xi - μi ||² = Squared Euclidean distance between a point and its cluster centroid

The **elbow point** is where the WCSS starts decreasing at a slower rate.

```
# 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()
```



In this plot, the elbow appears around K = 3 or 4.The dataset likely has 3 or 4 naturally distinct groups, and choosing K balances segmentation quality and computational efficiency

3. K means clustering

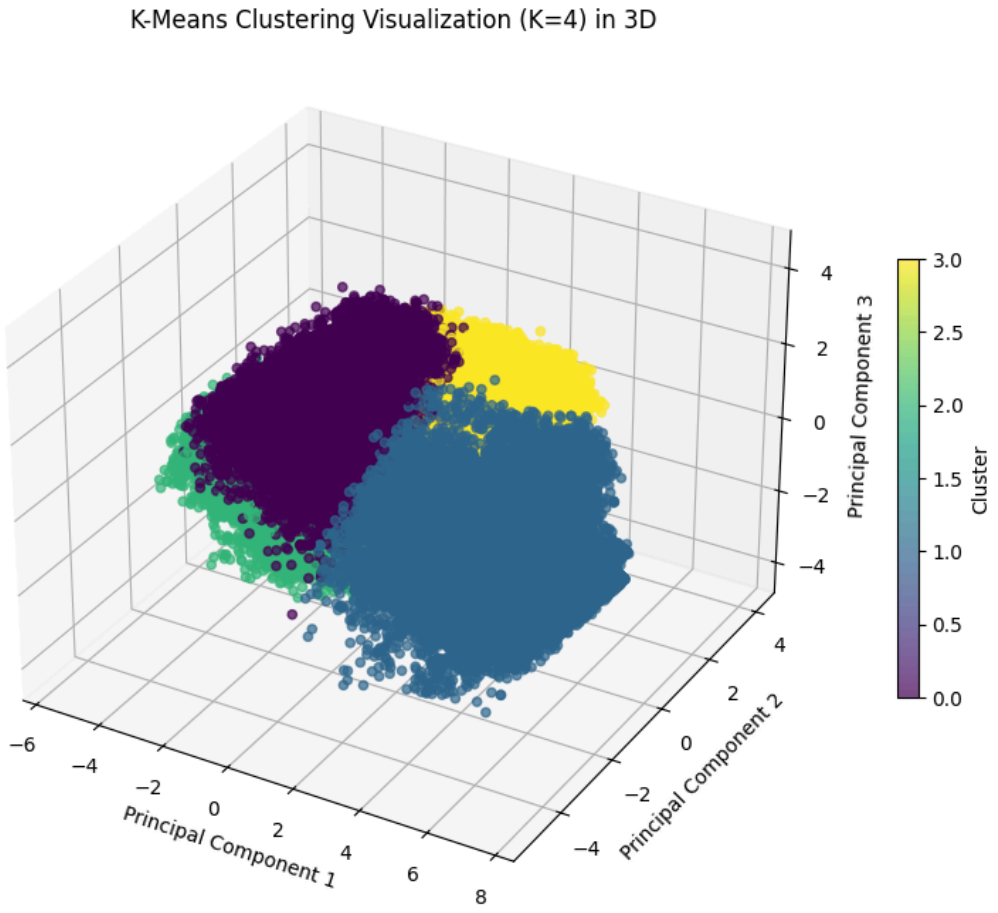
```
# Apply K-Means for K=3
kmeans3 = KMeans(n_clusters=3, random_state=42, n_init=10)
df_scaled["Cluster_K3"] = kmeans3.fit_predict(df_scaled)

# Apply K-Means for K=4
kmeans4 = KMeans(n_clusters=4, random_state=42, n_init=10)
df_scaled["Cluster_K4"] = kmeans4.fit_predict(df_scaled)

# Print cluster sizes
print("Cluster Sizes for K=3:\n", df_scaled["Cluster_K3"].value_counts())
print("\nCluster Sizes for K=4:\n", df_scaled["Cluster_K4"].value_counts())
```

```
Cluster Sizes for K=3:
 Cluster_K3
1    106713
2    104170
0     24511
Name: count, dtype: int64

Cluster Sizes for K=4:
 Cluster_K4
3    91263
2    80379
0    39407
1    24345
Name: count, dtype: int64
```

We try out clusters for k = 3 and 4. With k = 3, we find that the cluster size of the last cluster is very small as compared to the other clusters formed. So we use k = 4, we can observe that the clusters formed are similar in size.

Visualization of clusters:

```
# 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()
```



To visualize the clusters we perform **Principal Component Analysis (PCA)** to reduce the dataset dimensions to three, making it easier to visualize the clusters obtained from K-Means clustering. The **3D scatter plot** visualizes the clusters in the transformed feature space, where different colors represent different clusters. The clusters appear to be well-defined, indicating that the K-Means algorithm successfully grouped similar data points.
The PCA components **help analyze the contribution** of original features to the new principal components, providing insights into how data is distributed across clusters.

4. DBSCAN clustering

DBSCAN is an unsupervised clustering algorithm that groups together densely packed points while marking outliers as noise. Unlike K-Means, it does not require specifying the number of clusters in advance and is useful for identifying arbitrarily shaped clusters.
1. Defines Two Parameters:

- ε (epsilon): The radius to consider neighboring points.
- MinPts: Minimum number of points required in an ε-neighborhood to form a cluster.

2. Classifies Points as:

- Core Points: Points with at least MinPts neighbors within ε.
- Border Points: Points within ε of a core point but with fewer than MinPts neighbors.
- Noise Points: Points that are neither core nor border

(outliers).

3.Clustering Process:

- Starts with an unvisited point.
- Expands a cluster if it's a core point, connecting all reachable points.

Repeats until all points are classified.

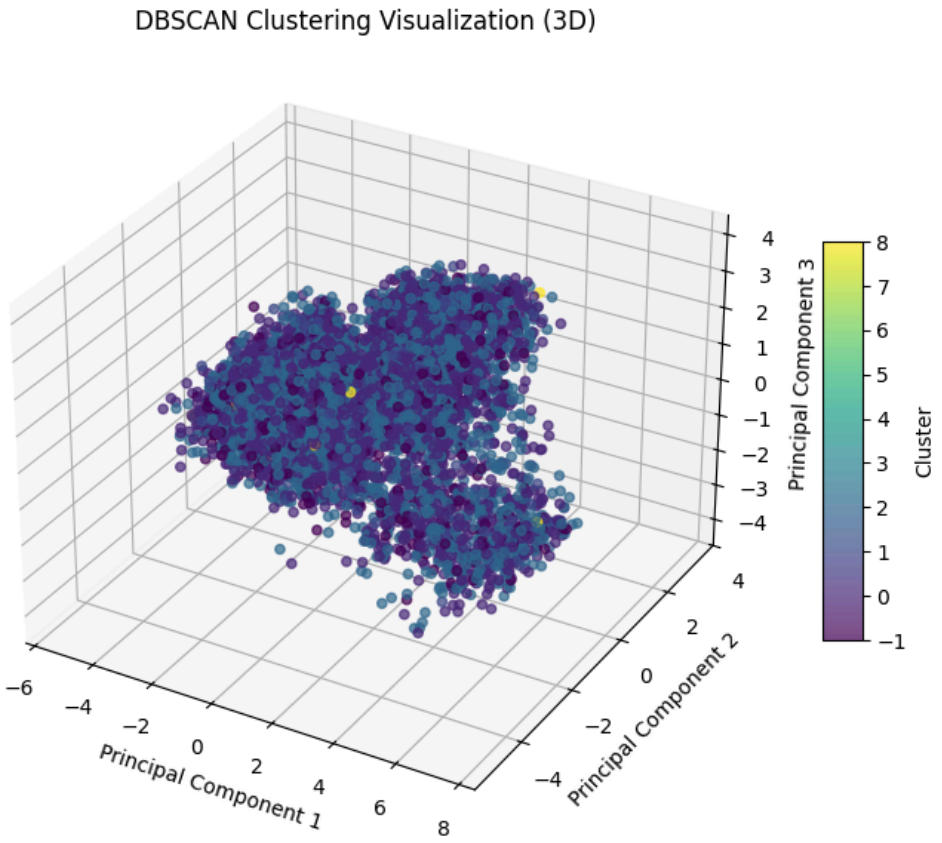
```
from sklearn.cluster import DBSCAN

eps_value = 2.0
min_samples_value = 30

# Apply DBSCAN
dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value, n_jobs=-1)
clusters = dbscan.fit_predict(df_sample)

# Add clusters to the DataFrame
df_sample["Cluster"] = clusters

# Display cluster distribution
print("Cluster Counts:\n", df_sample["Cluster"].value_counts())
```

```
Cluster Counts:
 Cluster
 0    34979
 1     7803
 2     4038
-1     2526
 4      163
 5      143
 3      118
 7       86
 6       76
 9       35
 8       33
Name: count, dtype: int64
```

We apply **DBSCAN clustering** with **eps=2** and **min_samples=30**, grouping data points based on density. It assigns cluster labels using dbscan.fit_predict(df_scaled) and stores them in the dataset. The output shows that most points (34,979) belong to **Cluster 0**, followed by **Cluster 1 (7,803)** and **Cluster 2 (4,038)**, while **2,526 points are classified as noise (-1)**.

The moderate number of noise points (-1) suggests that **eps=2** captures some dense regions but may still be slightly restrictive, leaving certain points unclustered. Additionally, **min_samples=30** ensures that only regions with sufficient density form clusters, leading to a few well-defined clusters while filtering out noise. As a result, the dataset forms multiple clusters of varying sizes, with most points concentrated in a few large groups.

Visualization:



The DBSCAN clustering visualization in 3D PCA space shows that the majority of points are assigned to cluster -1 (noise) with only a few points belonging to distinct smaller clusters. This suggests that DBSCAN's parameters (epsilon and min_samples) may not be well-tuned for clear separation, leading to most points being grouped together while only a few scattered regions are identified as separate clusters.

Inference from both the clustering:

Overall we can see that the performance of **K means** is **better** than DBSCAN for clustering the dataset.

K-Means was a better choice for this dataset because the data is well-distributed without high-density regions, and clusters can be separated by distance rather than density.

DBSCAN struggled due to high dimensionality and the lack of naturally dense clusters, causing excessive noise detection.

Conclusion:

In all, we come to know that K-Means and DBSCAN analyze airline passenger satisfaction data. K-Means (K=4) successfully grouped data into well-separated clusters, confirmed by the Elbow Method and PCA visualization, though it is sensitive to outliers. DBSCAN, however, struggled with this dataset, classifying many points as noise due to a lack of well-defined dense regions, making it less effective for this case.