**Assignment No.: ML 6**

**Title :** Implement K-Means clustering/ hierarchical clustering on sales\_data\_sample.csv dataset. Determine the number of clusters using the elbow method.

**Objective:**

The objective of this task is to implement K-Means clustering and hierarchical clustering on the sales\_data\_sample.csv dataset. The primary goal is to identify distinct clusters (groupings) in the data, which might represent different customer segments or sales categories. To determine the optimal number of clusters for K-Means, the elbow method will be used.

### Theory:

### Clustering is an unsupervised learning technique used to group similar data points together based on their features. The main goal is to identify inherent structures in the data without prior knowledge of labels.

### Classification, on the other hand, is a supervised learning technique where the model is trained using labeled data. The goal is to assign predefined labels to new data points based on learned patterns.

K-Means is a widely used unsupervised learning algorithm for partitioning data into **K distinct, non-overlapping clusters**. It minimizes intra-cluster variance and maximizes inter-cluster variance by iteratively adjusting cluster centroids and reassigning data points.

#### Steps in K-Means:

1. **Initialization**: Randomly select K initial centroids.
2. **Assignment**: Assign each data point to the nearest centroid.
3. **Update**: Recalculate centroids by averaging the points in each cluster.
4. **Repeat**: Repeat the assignment and update steps until convergence (i.e., when assignments no longer change).

**Elbow Method**: The elbow method is a technique to determine the optimal number of clusters. It involves plotting the within-cluster sum of squares (WCSS) against the number of clusters. The "elbow" point, where the rate of decrease sharply slows, is considered optimal.

### Hierarchical Clustering:

Hierarchical clustering builds a hierarchy of clusters. Unlike K-Means, which requires the number of clusters to be specified beforehand, hierarchical clustering provides a dendrogram representing nested clusters that can be cut at various levels to create the desired number of clusters.

#### Types of Hierarchical Clustering:

* **Agglomerative (Bottom-up)**: Starts with each data point in its own cluster and merges the closest clusters iteratively.
* **Divisive (Top-down)**: Starts with all data points in a single cluster and recursively splits them.

**Linkage Criteria**:

* **Single Linkage**: The distance between the closest points of two clusters.
* **Complete Linkage**: The distance between the farthest points of two clusters.
* **Average Linkage**: The average distance between all points in the two clusters.

**Key differences between K-Means and hierarchical clustering:**

* **Approach**: K-Means is a partitioning method that requires the number of clusters to be specified in advance, while hierarchical clustering does not require this and builds a tree of clusters.
* **Structure**: K-Means creates clusters that are non-overlapping and have a centroid, whereas hierarchical clustering produces a nested structure of clusters (dendrogram).
* **Scalability**: K-Means is generally more scalable and efficient for large datasets than hierarchical clustering, which can be computationally expensive as it examines all pairs of data points.
* **Sensitivity**: K-Means is sensitive to the initial placement of centroids and outliers, while hierarchical clustering can be more robust but may also produce misleading results if the data contains noise.

**Python Code:**

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.cluster import AgglomerativeClustering

import numpy as np

import seaborn as sns

# Load the dataset

data = pd.read\_csv('D:/Sem-I 2024-25/LP-III/Sales.csv', encoding='ISO-8859-1')

# Preprocess the dataset (choose relevant features)

# Replace 'SALES', 'QUANTITYORDERED', 'PRICEEACH' with actual column names in your dataset

features = data[['SALES', 'QUANTITYORDERED', 'PRICEEACH']]

# Standardize the features

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(features)

# Elbow method to determine the optimal number of clusters for K-Means

wcss = []  # within-cluster sum of squares

for i in range(1, 11):

    kmeans = KMeans(n\_clusters=i, random\_state=42)

    kmeans.fit(scaled\_features)

    wcss.append(kmeans.inertia\_)

# Plot the elbow curve

plt.figure(figsize=(8, 5))

plt.plot(range(1, 11), wcss, marker='o')

plt.title('Elbow Method for Optimal K (K-Means)')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS (Within-Cluster Sum of Squares)')

plt.show()

# Automatically determine the optimal number of clusters from the elbow

# This function calculates the "elbow" as the point with the maximum curvature

def optimal\_k\_elbow(wcss):

    x1, y1 = 1, wcss[0]

    x2, y2 = 10, wcss[-1]

    distances = []

    for i in range(10):

        x0 = i + 1

        y0 = wcss[i]

        numerator = abs((y2 - y1) \* x0 - (x2 - x1) \* y0 + x2 \* y1 - y2 \* x1)

        denominator = np.sqrt((y2 - y1) \*\* 2 + (x2 - x1) \*\* 2)

        distances.append(numerator / denominator)

    return distances.index(max(distances)) + 1

optimal\_k = optimal\_k\_elbow(wcss)

# Apply K-Means clustering with the optimal number of clusters

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

kmeans\_labels = kmeans.fit\_predict(scaled\_features)

# Print the number of clusters for K-Means

print(f'Number of clusters determined by K-Means (Elbow Method): {optimal\_k}')

# Plot K-Means clusters with colored datapoints

plt.figure(figsize=(8, 6))

sns.scatterplot(x=scaled\_features[:, 0], y=scaled\_features[:, 1], hue=kmeans\_labels, palette='Set1', s=100)

plt.title(f'K-Means Clustering with {optimal\_k} Clusters')

plt.xlabel('Feature 1 (e.g., SALES)')

plt.ylabel('Feature 2 (e.g., QUANTITYORDERED)')

plt.show()

# Perform hierarchical clustering using 'ward' linkage method

linked = linkage(scaled\_features, method='ward')

# Plot the dendrogram showing the first 4 levels (truncate\_mode='level', p=4)

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=True, truncate\_mode='level', p=4)

plt.title('Dendrogram for Hierarchical Clustering (Truncated to 4 Levels)')

plt.xlabel('Sample Index')

plt.ylabel('Euclidean Distance')

plt.show()

# Automatically determine the number of clusters from the dendrogram

# You can determine this by visually inspecting where the largest gap in the dendrogram appears

def get\_optimal\_clusters\_from\_dendrogram(linked, threshold=0.7):

    dendrogram\_data = dendrogram(linked, no\_plot=True)

    distances = np.diff(dendrogram\_data['dcoord'], axis=1).ravel()

    threshold\_distance = np.percentile(distances, threshold \* 100)

    num\_clusters = np.sum(distances > threshold\_distance) + 1

    return num\_clusters

optimal\_hc = get\_optimal\_clusters\_from\_dendrogram(linked)

# Apply hierarchical clustering with the chosen number of clusters

hc = AgglomerativeClustering(n\_clusters=optimal\_hc, affinity='euclidean', linkage='ward')

hierarchical\_labels = hc.fit\_predict(scaled\_features)

# Print the number of clusters for hierarchical clustering

print(f'Number of clusters determined by Hierarchical Clustering: {optimal\_hc}')

**Conclusion:**

 **K-Means** clustering is applied using the elbow method to determine the optimal number of clusters. The elbow point in the WCSS plot helps decide the number of clusters.

 **Hierarchical Clustering** is performed and visualized using a dendrogram. The dendrogram helps determine where to "cut" to form clusters.

 The cluster labels from both methods are added to the dataset, which is saved for further analysis.