Experiment 7

Aim:To implement different clustering algorithms.

Theory:

Clustering is an unsupervised machine learning technique used to group similar data points into clusters without predefined labels. In this experiment, we implemented the K-Means Clustering Algorithm as well as DBSCAN and Hierarchical Clustering Algorithm on a real-world dataset (loan_default_r.csv) using numerical features.

1)**K-Means Clustering:**K-Means is a centroid-based algorithm that partitions the dataset into K distinct non-overlapping subsets (clusters).The goal is to minimize intra-cluster variance (distance between points and their cluster centroid).

Algorithm Steps:

1. Select the number of clusters (k).
2. Initialize centroids randomly.
3. Assign each data point to the closest centroid.
4. Recalculate centroids as the mean of points in each cluster.
5. Repeat steps 3–4 until centroids do not change significantly or a maximum number of iterations is reached.

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 is an **unsupervised machine learning algorithm** used for **clustering** based on **data density**. It groups closely packed points into clusters and marks sparse regions as noise.

Key Concepts:

1. **Epsilon (ε):** Radius of neighborhood around a point.
2. **MinPts:** Minimum number of points required to form a dense region (core point).
3. **Core Point:** Has at least MinPts within ε radius.
4. **Border Point:** Has fewer than MinPts within ε but lies within the neighborhood of a core point.
5. **Noise (Outlier):** Not a core point and not within ε of any core point.

Steps in DBSCAN:

1. **Select a point randomly** from the dataset.
2. **Check the number of points** within the radius ε.
   - If **≥ MinPts** → it's a **core point**, a new cluster is formed.
   - If **< MinPts** → mark it as **noise** for now (might become part of a cluster later).
3. **Expand the cluster**:
   - For each point within ε of the core point:
     - If it's also a core point, add all its neighbors to the cluster (recursively).
     - If it's a border point, add it to the current cluster (but don't expand further).
4. **Repeat** until all points are assigned to a cluster or labeled as noise.

3)Hierarchical Clustering:

Hierarchical clustering is an unsupervised machine learning algorithm used to group data into a tree of nested clusters — forming a structure called a dendrogram. It does not require you to pre-specify the number of clusters.

Types of Hierarchical Clustering:

<u>Agglomerative (Bottom-Up) – Most common</u>

- Each data point starts as its own cluster.
- Pairs of clusters are merged as you move up the hierarchy.
- Divisive (Top-Down) – Less common
- All points start in one cluster.
- Clusters are split recursively as you go down.
- We'll focus on Agglomerative since it's widely used.

Steps in Agglomerative Hierarchical Clustering:

1. Start with each data point as its own cluster (n clusters for n points).
2. Compute the distance (e.g., Euclidean) between all clusters.
3. Merge the two closest clusters (based on a linkage criterion).
4. Update the distance matrix after merging.
5. Repeat steps 2–4 until all points are merged into a single cluster or until a desired number of clusters is reached.

Linkage Criteria (How distances between clusters are computed):

1. Single Linkage: Minimum distance between two points in different clusters.
2. Complete Linkage: Maximum distance between two points in different clusters.
3. Average Linkage: Average distance between all points in the two clusters.
4. Ward's Method: Minimizes the total within-cluster variance.

📊 Dendrogram:

- A tree-like diagram that shows how clusters are merged/split.
- The height of the branches represents the distance (or dissimilarity).
- You can cut the dendrogram at a certain height to select the number of clusters.

Dataset Used:

Dataset: loan_default_r.csv

Selected Features: Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, and DTIRatio.

Mathematical Insight:

The Elbow Method was used to determine the optimal number of clusters by plotting the inertia (within-cluster sum of squares) against various values of k.

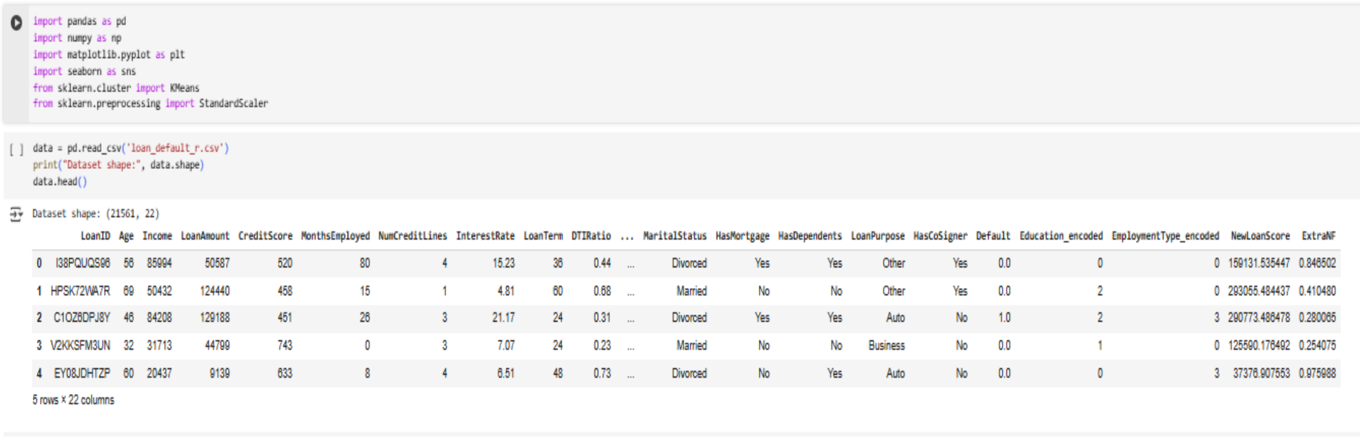
From the elbow plot, the optimal value of k was selected, and K-Means was applied accordingly.

Plot Information:

Elbow Plot: Visualizes how inertia decreases with increasing number of clusters and helps select the optimal k.

Cluster Visualization (Not shown fully here): Often performed using PCA or t-SNE for reducing dimensions to 2D, followed by plotting colored clusters.

Implementation:
1)Load and Explore Data

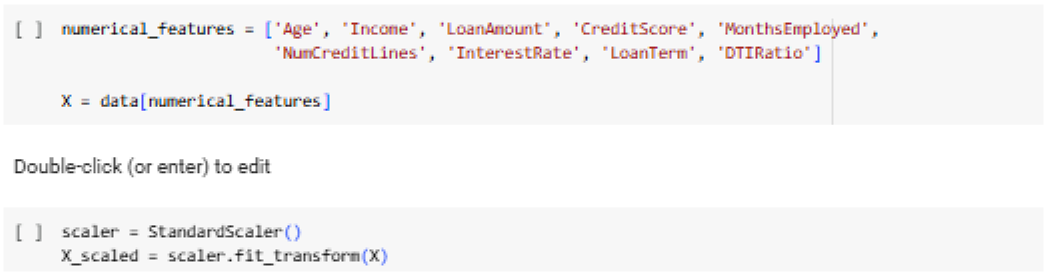


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
data = pd.read_csv('loan_default_r.csv')
print("Dataset shape:", data.shape)
data.head()
```

```
Dataset shape: (21561, 22)
       LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  NumCreditLines  InterestRate  LoanTerm  DTIRatio  ...  MaritalStatus  HasMortgage  HasDependents  LoanPurpose  HasCoSigner  Default  Education_encoded  EmploymentType_encoded   NewLoanScore   ExtraNF
0  I38PQUQS96   56   85994       50587          520              80               4         15.23        36      0.44  ...       Divorced          Yes            Yes        Other          Yes      0.0                  0                       0  159131.535447  0.846502
1  HPSK72WA7R   69   50432      124440          458              15               1          4.81        60      0.68  ...        Married           No             No        Other          Yes      0.0                  2                       0  293055.484437  0.410480
2  C1OZ6DPJ8Y   46   84208      129188          451              26               3         21.17        24      0.31  ...       Divorced          Yes            Yes         Auto           No      1.0                  2                       3  290773.486478  0.280065
3  V2KKSFM3UN   32   31713       44799          743               0               3          7.07        24      0.23  ...        Married           No             No     Business           No      0.0                  1                       0  125590.176492  0.254075
4  EY08JDHTZP   60   20437        9139          633               8               4          6.51        48      0.73  ...       Divorced           No            Yes         Auto           No      0.0                  0                       3   37376.907553  0.975988
5 rows × 22 columns
```

2)Scales the numerical features to normalize the data using StandardScaler.

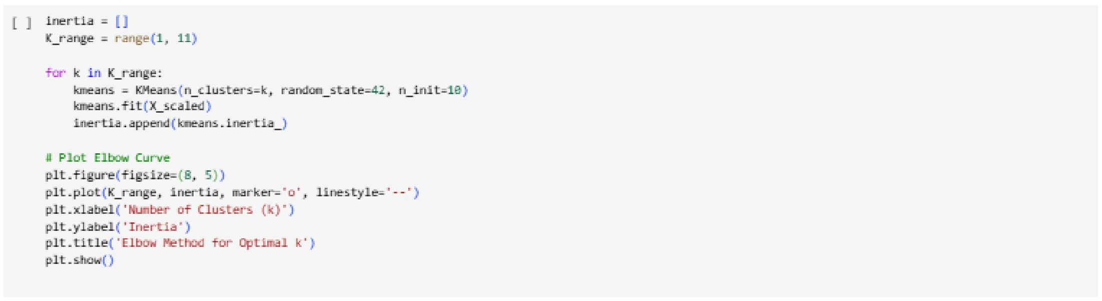


```
numerical_features = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
                      'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio']

X = data[numerical_features]
```

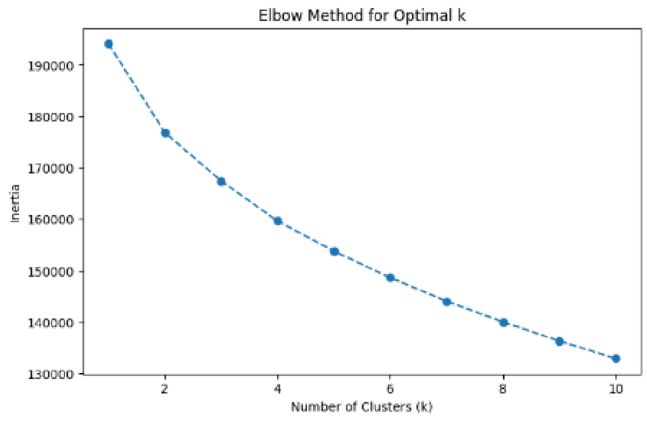
Double-click (or enter) to edit

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

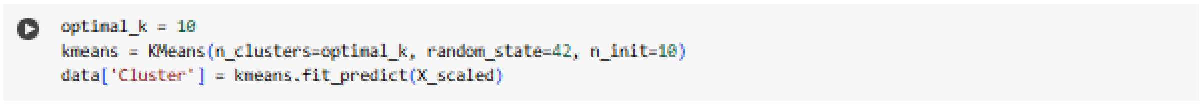
3)Initializes a loop to calculate inertia for different values of K to use the Elbow Method to find the optimal number of clusters.



```
inertia = []
K_range = range(1, 11)

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

# Plot Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

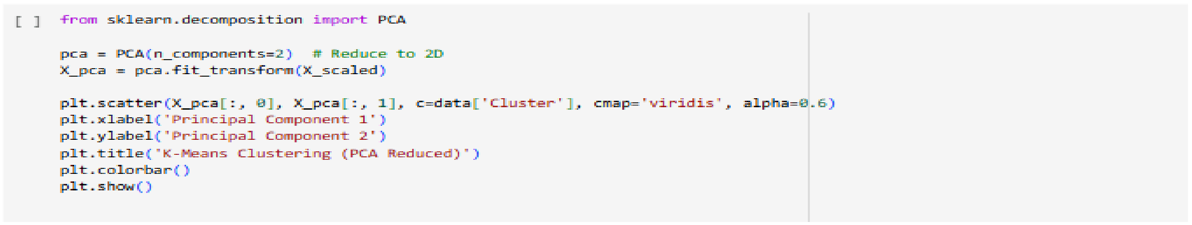


4)Applies K-Means clustering to the dataset using the chosen optimal number of clusters.

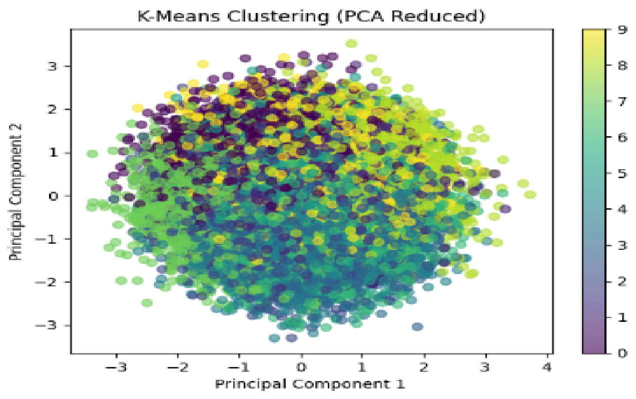


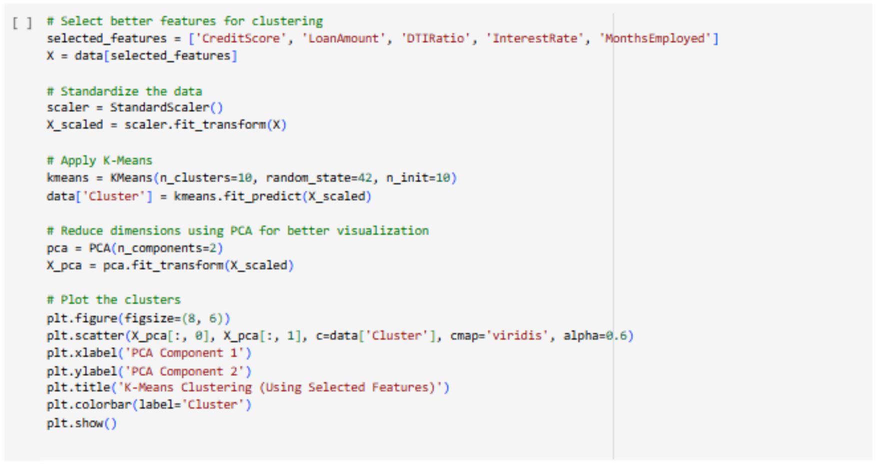
```
optimal_k = 10
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
data['Cluster'] = kmeans.fit_predict(X_scaled)
```

5) (Re)Imports libraries; possibly due to repeated or unorganized code cells and perform clustering related data-preprocessing and visualization.

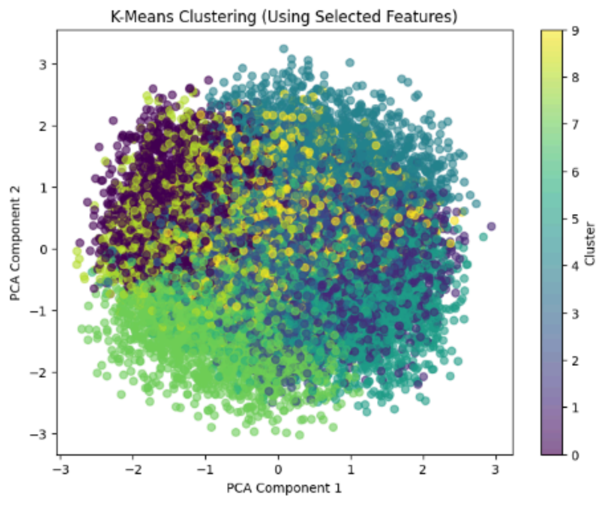


```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)  # Reduce to 2D
X_pca = pca.fit_transform(X_scaled)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['Cluster'], cmap='viridis', alpha=0.6)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('K-Means Clustering (PCA Reduced)')
plt.colorbar()
plt.show()
```





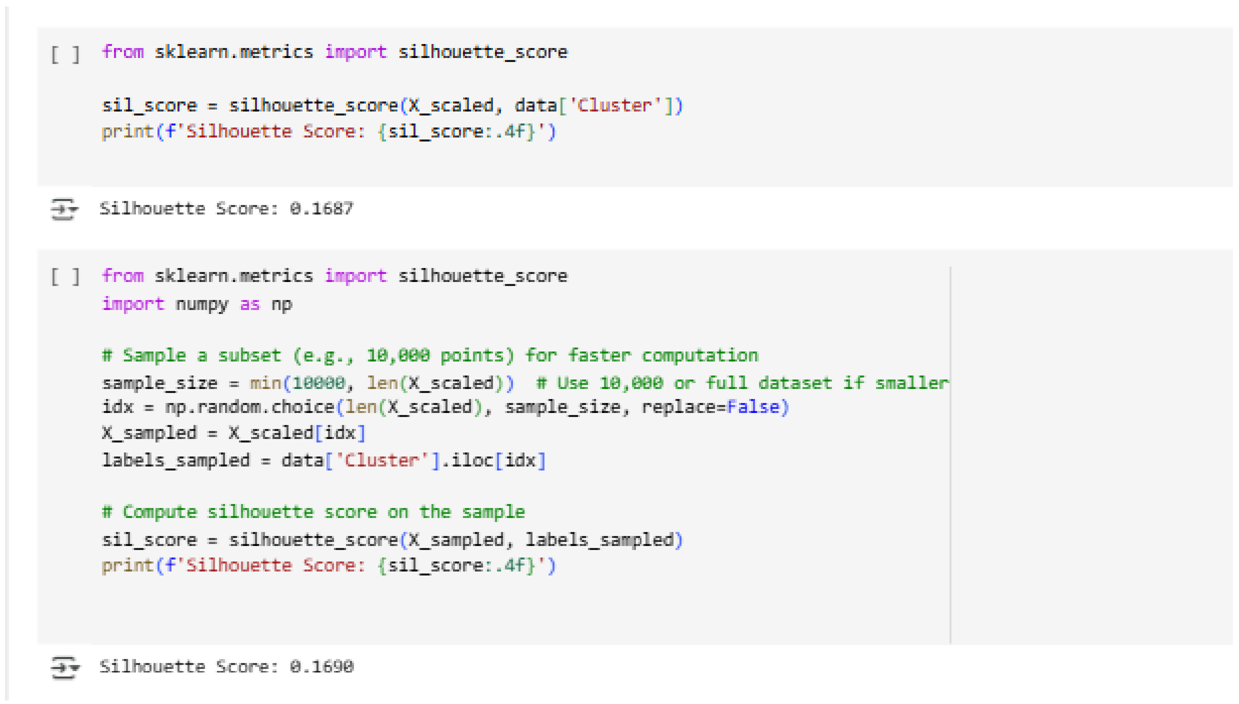
```
[ ] # Select better features for clustering
    selected_features = ['CreditScore', 'LoanAmount', 'DTIRatio', 'InterestRate', 'MonthsEmployed']
    X = data[selected_features]

    # Standardize the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply K-Means
    kmeans = KMeans(n_clusters=10, random_state=42, n_init=10)
    data['Cluster'] = kmeans.fit_predict(X_scaled)

    # Reduce dimensions using PCA for better visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)

    # Plot the clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['Cluster'], cmap='viridis', alpha=0.6)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.title('K-Means Clustering (Using Selected Features)')
    plt.colorbar(label='Cluster')
    plt.show()
```



6)

a)Calculates the Silhouette Score for the entire dataset using the features X_scaled and cluster labels stored in data['Cluster'].

b)Randomly selects a subset (max 10,000 points) from the dataset to speed up computation and computes the **Silhouette Score** only on this sample to reduce computational cost, especially useful for very large datasets.



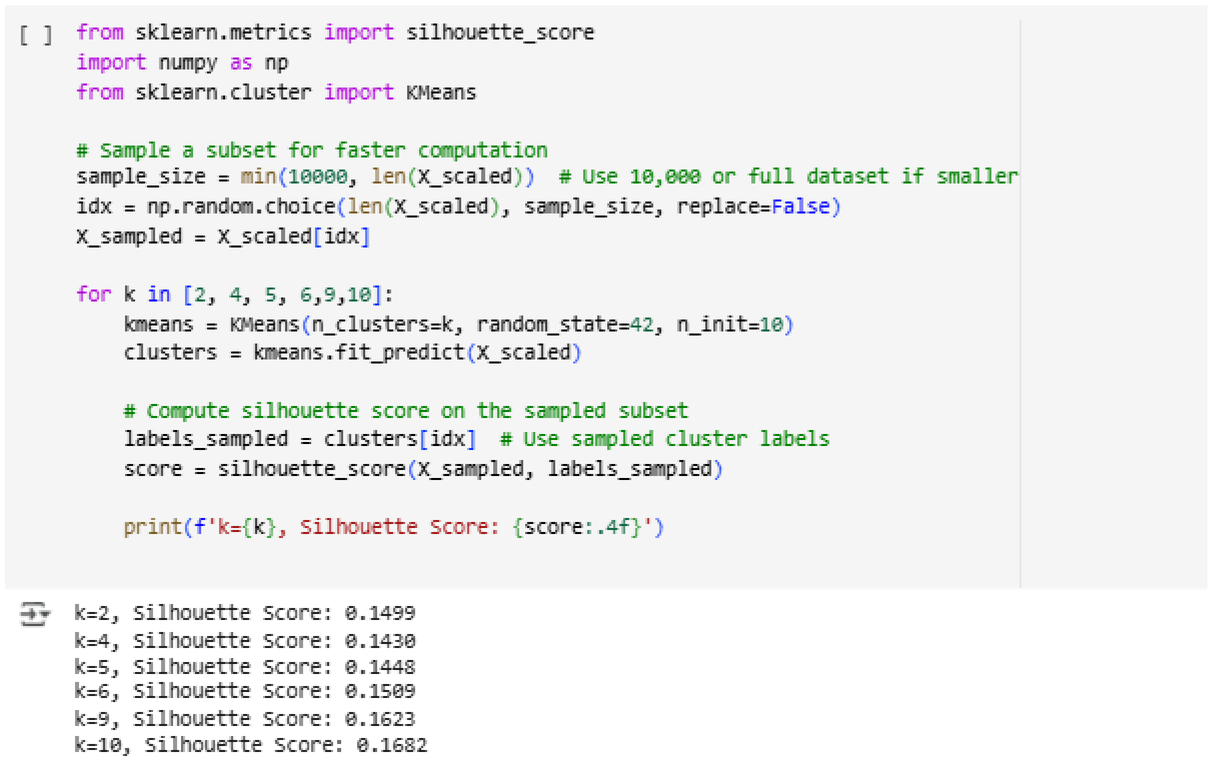
```
[ ] from sklearn.metrics import silhouette_score

    sil_score = silhouette_score(X_scaled, data['Cluster'])
    print(f'Silhouette Score: {sil_score:.4f}')
```

```
Silhouette Score: 0.1687
```

```
[ ] from sklearn.metrics import silhouette_score
    import numpy as np

    # Sample a subset (e.g., 10,000 points) for faster computation
    sample_size = min(10000, len(X_scaled))  # Use 10,000 or full dataset if smaller
    idx = np.random.choice(len(X_scaled), sample_size, replace=False)
    X_sampled = X_scaled[idx]
    labels_sampled = data['Cluster'].iloc[idx]

    # Compute silhouette score on the sample
    sil_score = silhouette_score(X_sampled, labels_sampled)
    print(f'Silhouette Score: {sil_score:.4f}')
```

```
Silhouette Score: 0.1690
```

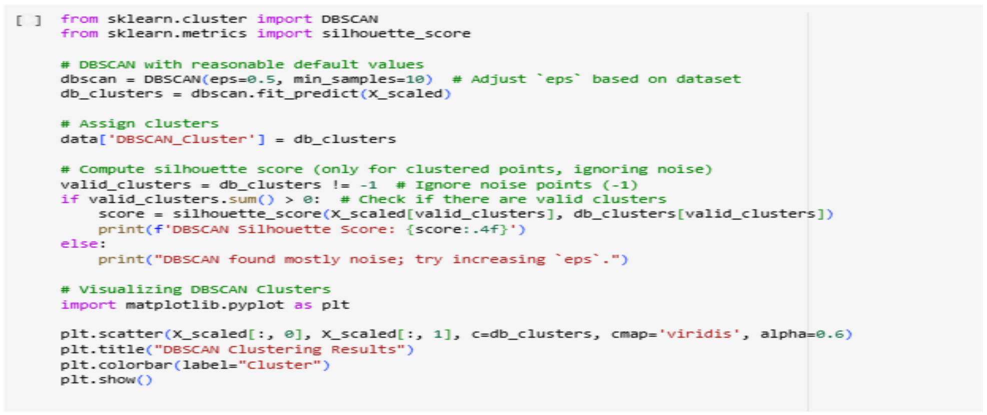
7)Evaluate the optimal number of clusters (k) for K-Means clustering using the Silhouette Score.
Turns out ,among the tested values, k = 10 yields the highest Silhouette Score (0.1682), suggesting it's the most suitable number of clusters based on this metric.
However, the scores are still quite low (all < 0.2), implying clusters are weakly separated.



```
[ ] from sklearn.metrics import silhouette_score
    import numpy as np
    from sklearn.cluster import KMeans

    # Sample a subset for faster computation
    sample_size = min(10000, len(X_scaled))  # Use 10,000 or full dataset if smaller
    idx = np.random.choice(len(X_scaled), sample_size, replace=False)
    X_sampled = X_scaled[idx]

    for k in [2, 4, 5, 6,9,10]:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        clusters = kmeans.fit_predict(X_scaled)

        # Compute silhouette score on the sampled subset
        labels_sampled = clusters[idx]  # Use sampled cluster labels
        score = silhouette_score(X_sampled, labels_sampled)

        print(f'k={k}, Silhouette Score: {score:.4f}')
```

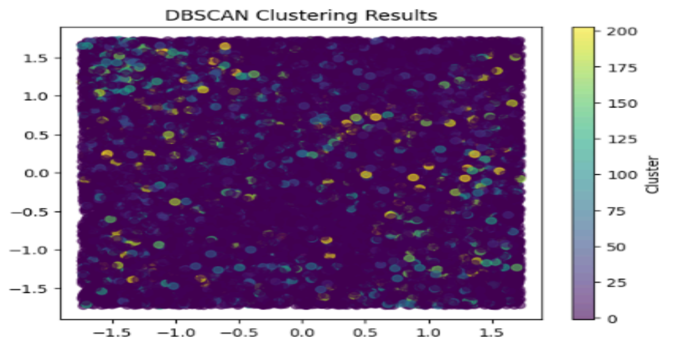
```
k=2, Silhouette Score: 0.1499
k=4, Silhouette Score: 0.1430
k=5, Silhouette Score: 0.1448
k=6, Silhouette Score: 0.1509
k=9, Silhouette Score: 0.1623
k=10, Silhouette Score: 0.1682
```

8)Applies DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm using the sklearn.cluster module.

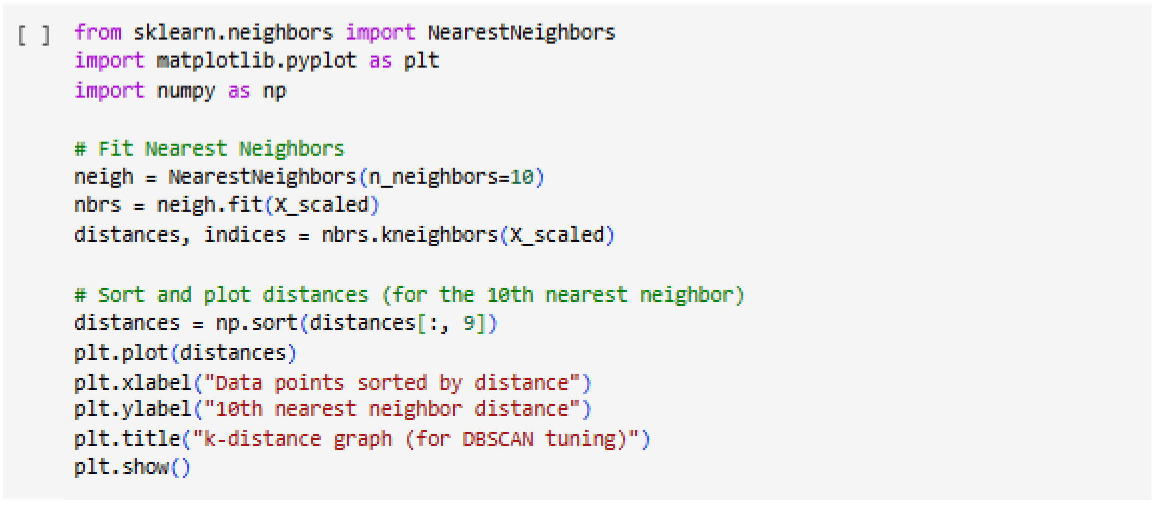


```
[ ] from sklearn.cluster import DBSCAN
    from sklearn.metrics import silhouette_score

    # DBSCAN with reasonable default values
    dbscan = DBSCAN(eps=0.5, min_samples=10)  # Adjust `eps` based on dataset
    db_clusters = dbscan.fit_predict(X_scaled)

    # Assign clusters
    data['DBSCAN_Cluster'] = db_clusters

    # Compute silhouette score (only for clustered points, ignoring noise)
    valid_clusters = db_clusters != -1  # Ignore noise points (-1)
    if valid_clusters.sum() > 0:  # Check if there are valid clusters
        score = silhouette_score(X_scaled[valid_clusters], db_clusters[valid_clusters])
        print(f'DBSCAN Silhouette Score: {score:.4f}')
    else:
        print("DBSCAN found mostly noise; try increasing `eps`.")

    # Visualizing DBSCAN Clusters
    import matplotlib.pyplot as plt

    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db_clusters, cmap='viridis', alpha=0.6)
    plt.title("DBSCAN Clustering Results")
    plt.colorbar(label="Cluster")
    plt.show()
```

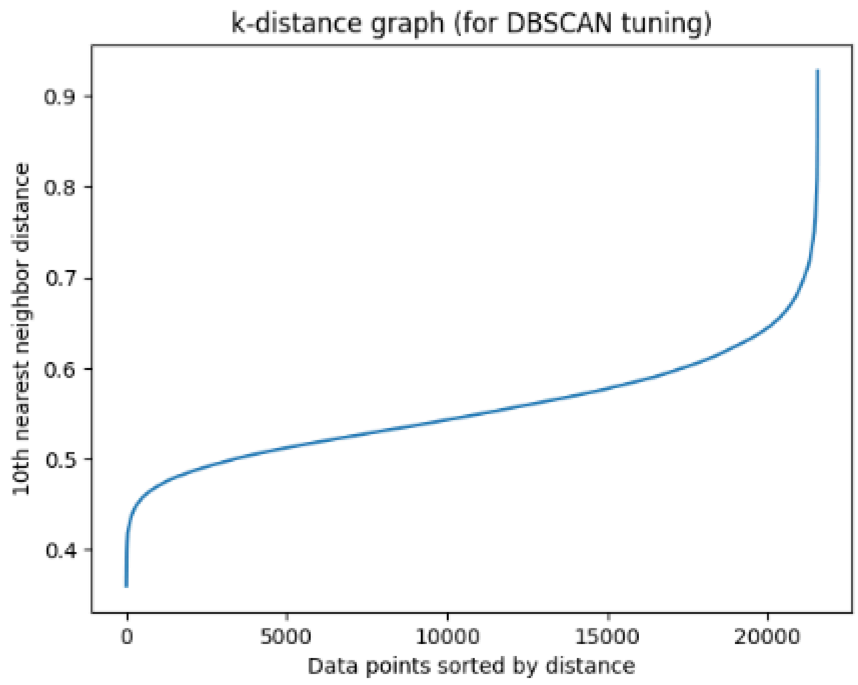
```
DBSCAN Silhouette Score: -0.4034
```



9) tune the eps parameter for DBSCAN by generating a k-distance graph.



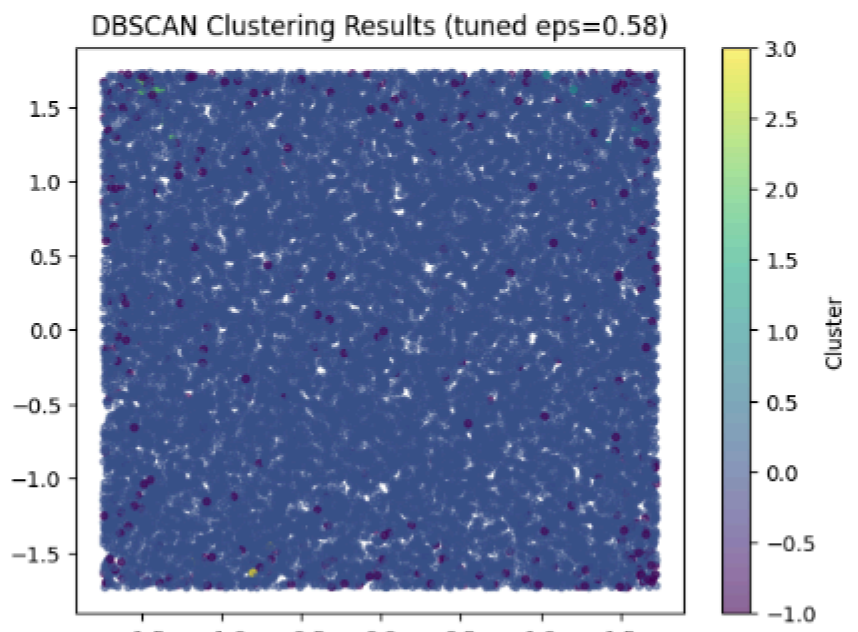
```
[ ] from sklearn.neighbors import NearestNeighbors
    import matplotlib.pyplot as plt
    import numpy as np

    # Fit Nearest Neighbors
    neigh = NearestNeighbors(n_neighbors=10)
    nbrs = neigh.fit(X_scaled)
    distances, indices = nbrs.kneighbors(X_scaled)

    # Sort and plot distances (for the 10th nearest neighbor)
    distances = np.sort(distances[:, 9])
    plt.plot(distances)
    plt.xlabel("Data points sorted by distance")
    plt.ylabel("10th nearest neighbor distance")
    plt.title("k-distance graph (for DBSCAN tuning)")
    plt.show()
```



10)DBSCAN Clustering (Tuned eps=0.58)

- DBSCAN is applied with eps=0.58, but most points are classified as noise (dark purple, -1).
- The silhouette score is -0.0809, indicating poor clustering. A higher eps may be needed.

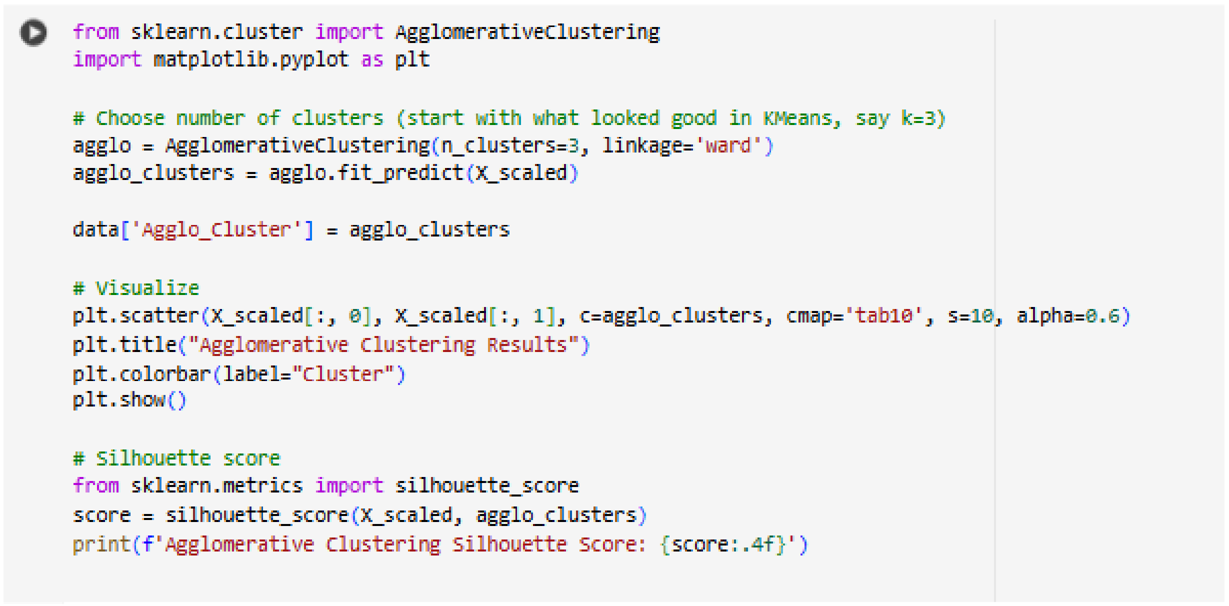
```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

dbscan = DBSCAN(eps=0.58, min_samples=10)
db_clusters = dbscan.fit_predict(X_scaled)

data['DBSCAN_Cluster'] = db_clusters

# Visualize
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db_clusters, cmap='viridis', s=10, alpha=0.6)
plt.title("DBSCAN Clustering Results (tuned eps=0.58)")
plt.colorbar(label="Cluster")
plt.show()

# Silhouette score (only for clustered points)
from sklearn.metrics import silhouette_score
valid_points = db_clusters != -1
if valid_points.sum() > 0:
    sil_score = silhouette_score(X_scaled[valid_points], db_clusters[valid_points])
    print(f"DBSCAN Silhouette Score (ignoring noise): {sil_score:.4f}")
else:
    print("All points classified as noise. Try increasing eps.")
```



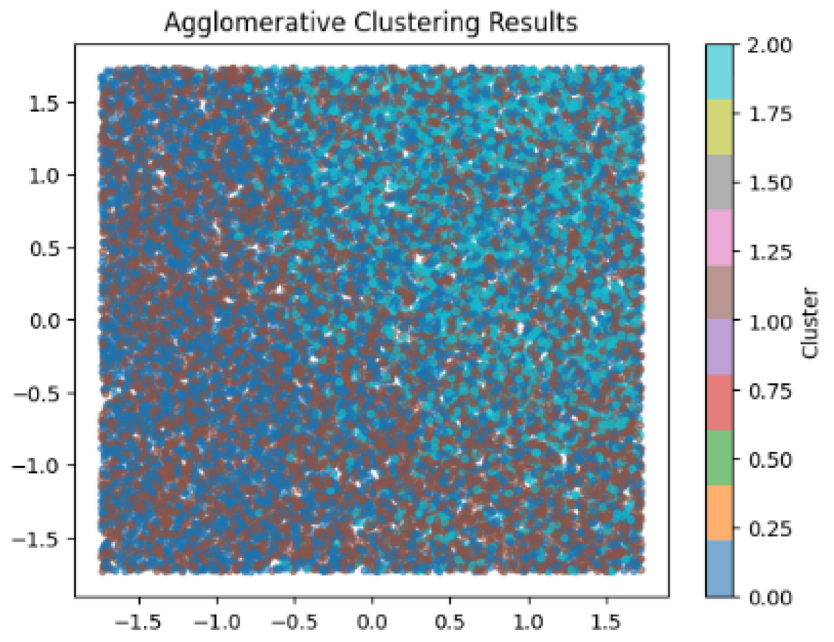
```
DBSCAN Silhouette Score (ignoring noise): -0.0809
```

11)Dendrogram for Hierarchical Clustering

- A dendrogram is created using hierarchical clustering (ward method) on a sample of data.
- Helps determine the optimal number of clusters by identifying where to cut the tree.



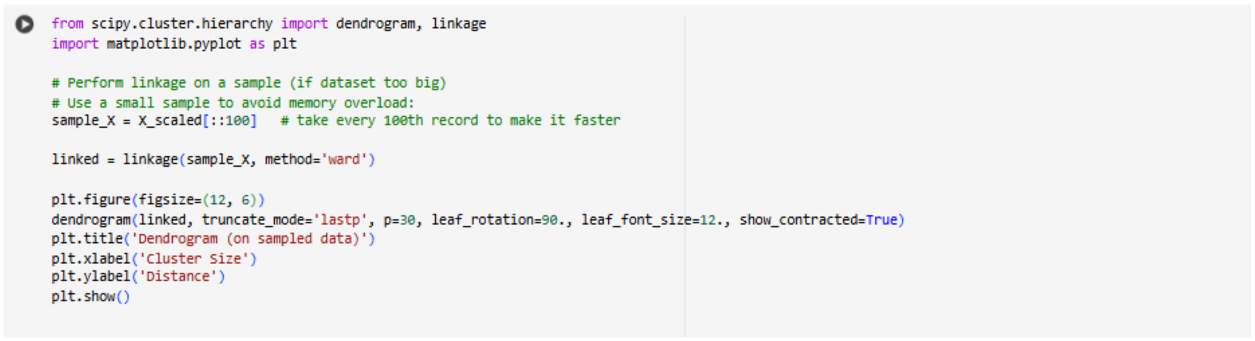
```
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

# Choose number of clusters (start with what looked good in KMeans, say k=3)
agglo = AgglomerativeClustering(n_clusters=3, linkage='ward')
agglo_clusters = agglo.fit_predict(X_scaled)

data['Agglo_Cluster'] = agglo_clusters

# Visualize
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=agglo_clusters, cmap='tab10', s=10, alpha=0.6)
plt.title("Agglomerative Clustering Results")
plt.colorbar(label="Cluster")
plt.show()

# Silhouette score
from sklearn.metrics import silhouette_score
score = silhouette_score(X_scaled, agglo_clusters)
print(f'Agglomerative Clustering Silhouette Score: {score:.4f}')
```



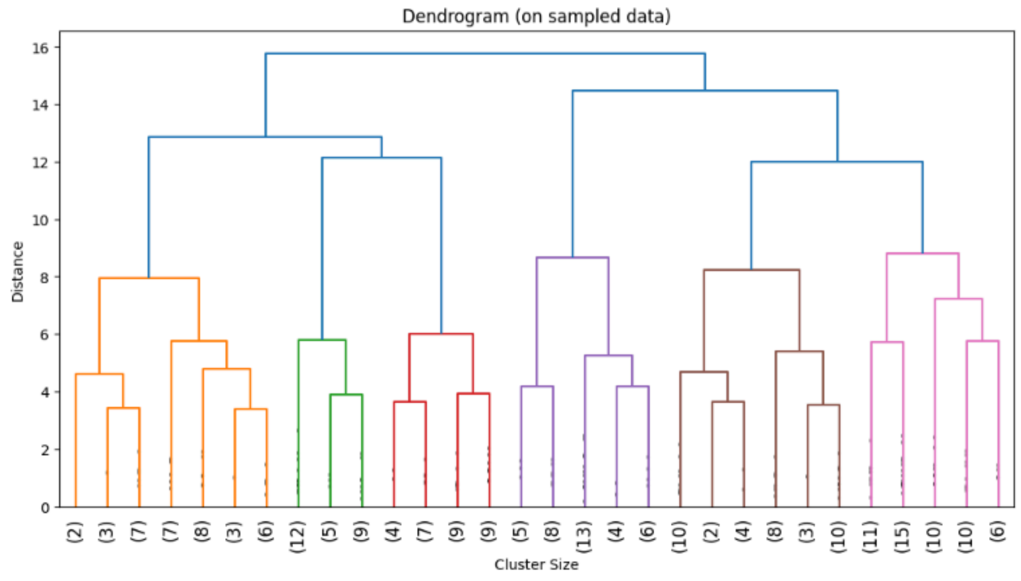
```
Agglomerative Clustering Silhouette Score: 0.0760
```

12)Agglomerative Clustering (k=3)

- Hierarchical clustering is applied with n_clusters=3 using the ward linkage method.
- The silhouette score is **0.0760**, indicating weak clustering but better than DBSCAN.



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

# Perform linkage on a sample (if dataset too big)
# Use a small sample to avoid memory overload:
sample_X = X_scaled[::100]   # take every 100th record to make it faster

linked = linkage(sample_X, method='ward')

plt.figure(figsize=(12, 6))
dendrogram(linked, truncate_mode='lastp', p=30, leaf_rotation=90., leaf_font_size=12., show_contracted=True)
plt.title('Dendrogram (on sampled data)')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.show()
```



Conclusion:

In this experiment, we successfully implemented and analyzed different clustering algorithms including K-Means, DBSCAN, and Hierarchical Clustering. Each method demonstrated its capability in identifying meaningful clusters in an unsupervised manner. K-Means proved effective for spherical-shaped clusters, DBSCAN for arbitrary shapes and noise detection, and Hierarchical Clustering for structure discovery and visualization through dendrograms. The clustering outcomes were visualized using scatter plots and dendrograms, providing a clear understanding of how the data is grouped.