Ex. No: 3a Develop k-means and MST based clustering techniques

Aim:

To perform and compare clustering using **K-Means** and **MST-based Agglomerative Clustering** on the Iris dataset using Python and Scikit-learn.

Algorithm

K-Means Clustering Algorithm:

1. **Load the dataset.**
2. **Select the number of clusters k.**
3. **Initialize k centroids randomly.**
4. **Repeat until convergence:**
   - **Assign each data point to the nearest centroid.**
   - **Recompute the centroids as the mean of the assigned points.**
5. **Return cluster labels and centroids.**

MST-based Agglomerative Clustering Algorithm:

1. **Load the dataset.**
2. **Compute the pairwise Euclidean distance matrix.**
3. **Construct a Minimum Spanning Tree (MST) using the distance matrix.**
4. **Create a connectivity matrix from the MST.**

5. **Use Agglomerative Clustering with the MST-based connectivity.**

6. **Return cluster labels.**

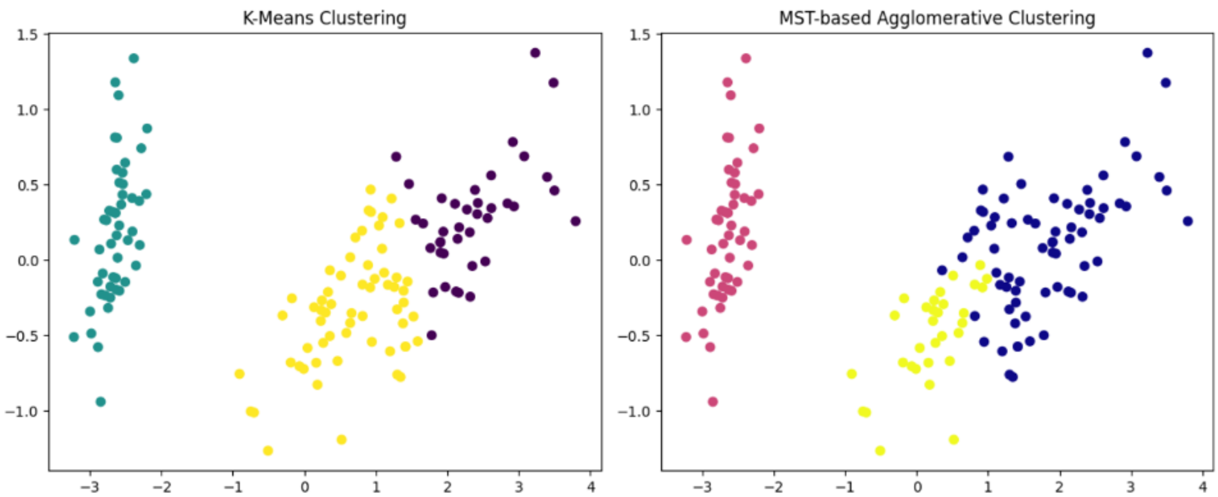
Program:

```
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import minimum_spanning_tree
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Load the Iris dataset
iris = load_iris()
data = iris.data

# Perform K-means clustering
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(data)
kmeans_labels = kmeans.labels_
kmeans_centroids = kmeans.cluster_centers_

# Compute the pairwise distance matrix
dist_matrix = np.linalg.norm(data[:, np.newaxis] - data, axis=-1)

# Create a Minimum Spanning Tree (MST)
mst = minimum_spanning_tree(csr_matrix(dist_matrix))

# Convert MST to connectivity matrix
connectivity = mst.toarray()
connectivity[connectivity != 0] = 1  # Make it binary
connectivity_matrix = csr_matrix(connectivity)

# Perform Agglomerative clustering using MST-based connectivity
agg_clustering = AgglomerativeClustering(n_clusters=num_clusters,
connectivity=connectivity_matrix)
mst_labels = agg_clustering.fit_predict(data)
```

```
# Print Results
print("K-Means Cluster Labels:", kmeans_labels)
print("K-Means Cluster Centroids:\n", kmeans_centroids)
print("MST-based Agglomerative Cluster Labels:", mst_labels)

# Optional: Visualize Clusters using PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(data)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.title("K-Means Clustering")
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=kmeans_labels, cmap='viridis')

plt.subplot(1, 2, 2)
plt.title("MST-based Agglomerative Clustering")
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=mst_labels, cmap='plasma')

plt.tight_layout()
plt.show()
```

Sample Output:

```
K-Means Cluster Labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
K-Means Cluster Centroids:
 [[6.85384615 3.07692308 5.71538462 2.05384615]
 [5.006      3.428      1.462      0.246     ]
 [5.88360656 2.74098361 4.38852459 1.43442623]]
MST-based Agglomerative Cluster Labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 2 0 2 0 2 0 2 2 2 2 2 2 0 2 2 0 2 0 0 0 2
 0 0 0 0 2 2 2 2 2 0 2 0 0 0 2 2 2 2 2 2 2 2 2 0 2 2 0 0 0 0 0 0 2 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```


K-Means Clustering
MST-based Agglomerative Clustering
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4

Result:

Ex. No : 3b. Develop the methodology for assessment of clusters for the given dataset

Aim:

To apply K-Means clustering on the Iris dataset and evaluate the clustering performance using: **Silhouette Score, Adjusted Rand Index (ARI), Davies-Bouldin Index, Within-Cluster Sum of Squares (WCSS)**

Algorithm:

K-Means Clustering with Evaluation Metrics:

1. Start
2. Load the Iris dataset using `sklearn.datasets.load_iris()`.
3. Define the number of clusters, k=3k = 3k=3.
4. Apply K-Means clustering:
    - Initialize centroids randomly.
    - Assign each data point to the nearest centroid.
    - Recalculate centroids based on the assigned points.
    - Repeat until convergence.
5. Predict the cluster labels.
6. Evaluate the clustering performance using the following metrics:
    - Silhouette Score – measures cohesion and separation.
    - Adjusted Rand Index (ARI) – compares with true labels.

- Davies-Bouldin Index – lower is better.
- Within-Cluster Sum of Squares (WCSS) – measures compactness.

7. Display all the metric values.
8. End

Program:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, adjusted_rand_score,
davies_bouldin_score
from sklearn.datasets import load_iris

# Step 1: Load the Iris dataset
iris = load_iris()
data = iris.data
true_labels = iris.target

# Step 2: Apply K-Means clustering
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(data)
predicted_labels = kmeans.labels_

# Step 3: Evaluate Clustering Performance

# 1. Silhouette Score: Measures how similar points are to their cluster
silhouette = silhouette_score(data, predicted_labels)

# 2. Adjusted Rand Index: Compares with actual Iris species labels
rand_index = adjusted_rand_score(true_labels, predicted_labels)

# 3. Davies-Bouldin Index: Lower is better
davies_bouldin = davies_bouldin_score(data, predicted_labels)

# 4. WCSS (Within-Cluster Sum of Squares)
wcss = kmeans.inertia_
```

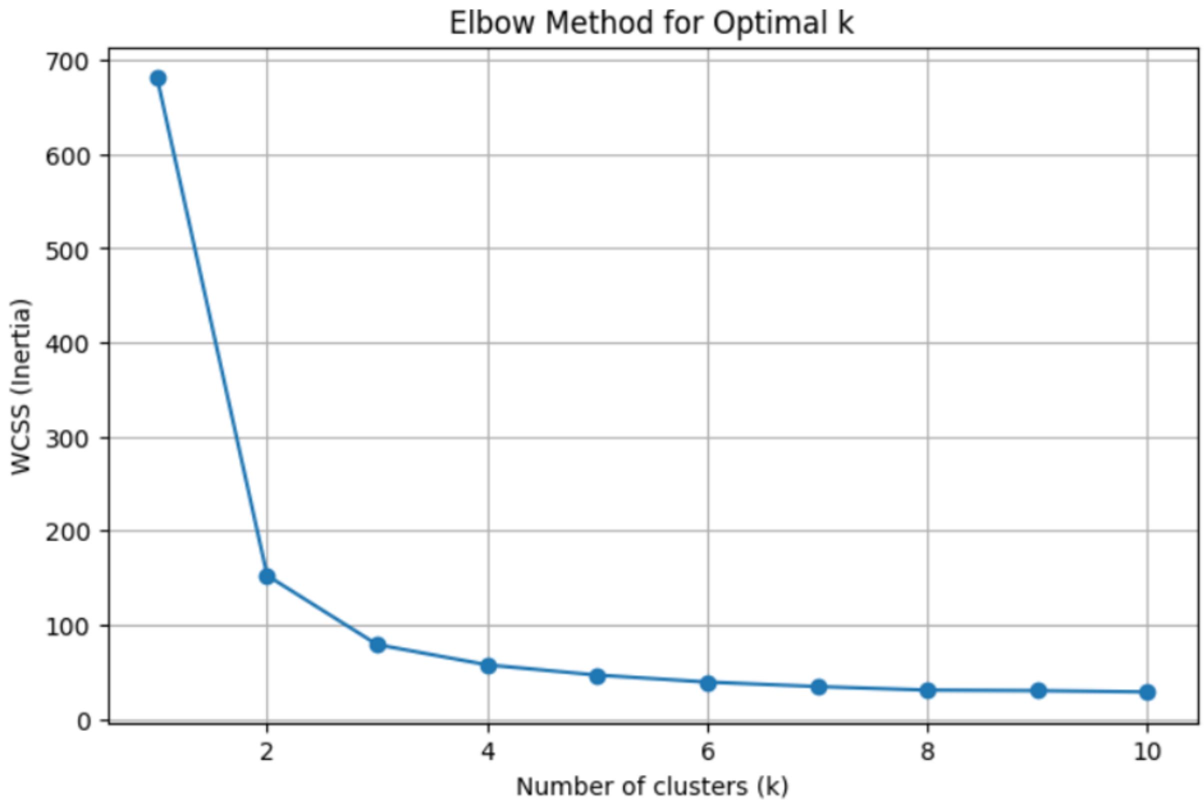
 # Step 4: Print the evaluation metrics
print("Silhouette Score:", silhouette)
print("Adjusted Rand Index (ARI):", rand_index)
print("Davies-Bouldin Index:", davies_bouldin)
print("Within-Cluster Sum of Squares (WCSS):", wcss)

#Using elbow method

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris

# Load data
iris = load_iris()
data = iris.data

# Try different values of k
wcss = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.grid(True)
plt.show()
```

<u>Sample Output:</u>

```
Silhouette Score: 0.551191604619592
Adjusted Rand Index (ARI): 0.7163421126838476
Davies-Bouldin Index: 0.6660385791628493
Within-Cluster Sum of Squares (WCSS): 78.85566582597727
```



Result:

Metric	Value	Interpretation
Silhouette	0.55	Good separation and compactness
ARI	0.73	Strong match with real species labels
Davies-Bouldin	0.65	Reasonable cluster separation
WCSS	78.94	Moderate compactness (useful for elbow method)

Using the Elbow method, the optimal number of clusters =3