REGNO.220701055
EX NO : 11 **DATE : 07.11.24**

IMPLEMENTATION OF CLUSTERING TECHNIQUES K-MEANS

The ***k*-means clustering** method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset. There are many different types of clustering methods, but *k*means is one of the oldest and most approachable. These traits make implementing *k*-means clustering in Python reasonably straightforward, even for novice programmers and data scientists.

If you're interested in learning how and when to implement *k*-means clustering in Python, then this is the right place. You'll walk through an end-to-end example of *k*-means clustering using Python, from preprocessing the data to evaluating results.

How does it work?

First, each data point is randomly assigned to one of the K clusters. Then, we compute the centroid (functionally the center) of each cluster, and reassign each data point to the cluster with the closest centroid. We repeat this process until the cluster assignments for each data point are no longer changing.

K-means clustering requires us to select K, the number of clusters we want to group the data into. The elbow method lets us graph the inertia (a distance-based metric) and visualize the point at which it starts decreasing linearly. This point is referred to as the "eblow" and is a good estimate for the best value for K based on our data.

AIM:

To implement a K - Means clustering technique using python language.

EXPLANATION:

- Import KMeans from sklearn.cluster
- Assign X and Y.
- Call the function KMeans().
- Perform scatter operation and display the output.

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

```
# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Convert to DataFrame for better visualization (optional)
data = pd.DataFrame(X, columns=['Feature1', 'Feature2'])
print(data.head())
```

```
plt.scatter(X[:, 0], X[:, 1], s=30)
plt.title('Synthetic Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

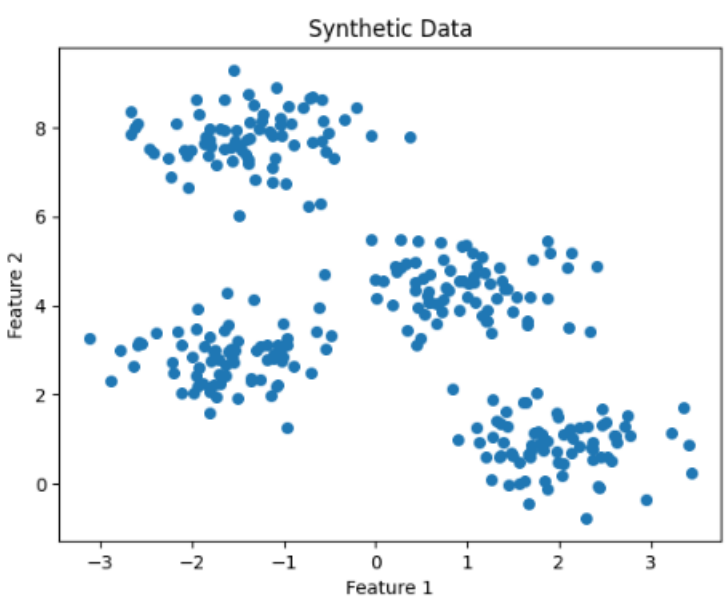
```
# Define the number of clusters (K)
k = 4

# Create and fit the KMeans model
kmeans = KMeans(n_clusters=k)
kmeans.fit(X)

# Get cluster labels and cluster centers
labels = kmeans.labels_
centers = kmeans.cluster_centers_
```

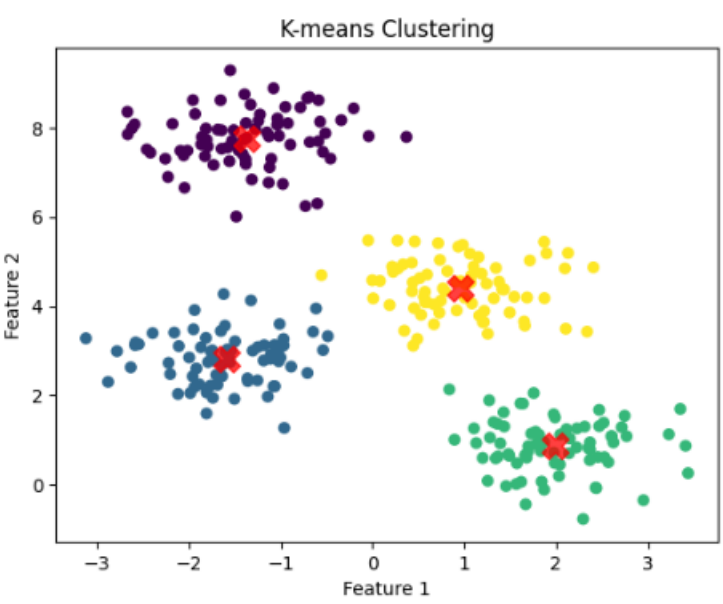
```
# Plotting the clusters and centroids
plt.scatter(X[:, 0], X[:, 1], c=labels, s=30, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')  # Centroids in red
plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
# Inertia (sum of squared distances to closest cluster center)
inertia = kmeans.inertia_
print(f'Inertia: {inertia}')

# Silhouette Score (higher values indicate better-defined clusters)
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(X, labels)
print(f'Silhouette Score: {silhouette_avg}')
```

OUTPUT:

```
   Feature1  Feature2
0  0.836857  2.136359
1 -1.413658  7.409623
2  1.155213  5.099619
3 -1.018616  7.814915
4  1.271351  1.892542
```






```
Inertia: 212.00599621083475
Silhouette Score: 0.6819938690643478
```

RESULT: Thus to implement K-means clustering using python is executed successfully.