Experiment - 7 K-means Clustering Algorithm

Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import random

def euclidean_distance(point1, point2):
    """
    Calculate the Euclidean distance between two points
    """
    return np.sqrt(np.sum((point1 - point2) ** 2))

def initialize_centroids(data, k):
    """
    Initialize k centroids by randomly selecting k data points
    """
    # Get the number of samples in the dataset
    n_samples = data.shape[0]

    # Randomly select k data points as initial centroids
    centroid_indices = random.sample(range(n_samples), k)
    centroids = data[centroid_indices]

    return centroids

def assign_to_clusters(data, centroids):
    """
    Assign each data point to the nearest centroid
    """
    # Get the number of samples and centroids
    n_samples = data.shape[0]
    k = centroids.shape[0]

    # Initialize cluster assignments
    cluster_assignments = np.zeros(n_samples, dtype=int)

    # Assign each point to the nearest centroid
    for i in range(n_samples):
        # Calculate distances to all centroids
        distances = [euclidean_distance(data[i], centroid) for centroid in
centroids]

        # Assign to the closest centroid
        cluster_assignments[i] = np.argmin(distances)

    return cluster_assignments
```

```python
def update_centroids(data, cluster_assignments, k):
    """
    Update centroids based on current cluster assignments
    """
    # Get the number of features in the dataset
    n_features = data.shape[1]

    # Initialize new centroids
    centroids = np.zeros((k, n_features))

    # Calculate new centroid for each cluster
    for cluster in range(k):
        # Get points assigned to this cluster
        cluster_points = data[cluster_assignments == cluster]

        # If no points are assigned to this cluster, keep the centroid
unchanged
        if len(cluster_points) > 0:
            centroids[cluster] = np.mean(cluster_points, axis=0)

    return centroids

def kmeans(data, k, max_iterations=100):
    """
    Implement K-means clustering algorithm

    Parameters:
    data (numpy array): The input data points
    k (int): The number of clusters
    max_iterations (int): Maximum number of iterations

    Returns:
    centroids (numpy array): Final cluster centroids
    cluster_assignments (numpy array): Cluster assignments for each data point
    """
    # Initialize centroids
    centroids = initialize_centroids(data, k)

    # Iterate until convergence or max iterations
    for i in range(max_iterations):
        # Store old centroids for convergence check
        old_centroids = centroids.copy()

        # Assign points to clusters
        cluster_assignments = assign_to_clusters(data, centroids)

        # Update centroids
        centroids = update_centroids(data, cluster_assignments, k)

        # Check for convergence
        if np.array_equal(old_centroids, centroids):
            print(f"Converged after {i+1} iterations")
            break
    return centroids, cluster_assignments

def plot_clusters(data, cluster_assignments, centroids, k):
    """
    Visualize the clusters and centroids
    """
    plt.figure(figsize=(10, 8))

    # Plot data points with their cluster assignments
    colors = ['r', 'g', 'b', 'y', 'c', 'm', 'orange', 'purple', 'brown',
'pink']
    for cluster in range(k):
        # Get points in this cluster
        cluster_points = data[cluster_assignments == cluster]

        # Plot points
        plt.scatter(
            cluster_points[:, 0],
            cluster_points[:, 1],
            s=50,
            c=colors[cluster % len(colors)],
            label=f'Cluster {cluster+1}'
        )

    # Plot centroids
    plt.scatter(
        centroids[:, 0],
        centroids[:, 1],
        s=200,
        c='black',
        marker='*',
        label='Centroids'
    )

    plt.title(f'K-means Clustering (k={k})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.savefig('kmeans_clusters.png')
    plt.show()

def main():
    # Generate synthetic data for clustering
    print("Generating synthetic data...")
    X, y = make_blobs(n_samples=300, n_features=2, centers=4, random_state=42)

    # Choose the number of clusters (k)
    k = 4

    print(f"Running K-means with k={k}...")
    # Run K-means algorithm
    centroids, cluster_assignments = kmeans(X, k)
```

```python
    # Print results
    print("\nFinal Centroids:")
    for i, centroid in enumerate(centroids):
        print(f"Centroid {i+1}: {centroid}")

    print("\nCluster Assignments:")
    unique, counts = np.unique(cluster_assignments, return_counts=True)
    for i, (cluster, count) in enumerate(zip(unique, counts)):
        print(f"Cluster {cluster+1}: {count} points")

    # Visualize the clusters
    plot_clusters(X, cluster_assignments, centroids, k)

    # Calculate inertia (sum of squared distances to centroids)
    inertia = 0
    for i, point in enumerate(X):
        centroid = centroids[cluster_assignments[i]]
        inertia += euclidean_distance(point, centroid) ** 2

    print(f"\nInertia (Sum of squared distances): {inertia:.2f}")

if __name__ == "__main__":
    main()
```

Output

```
PS V:\Deeptanshu Lal\PROJECTS\ML\exp-6> cd ..\exp-7; python exp-7.py
Running K-means with k=4...
Converged after 3 iterations

Final Centroids:
Centroid 1: [-6.83235205 -6.83045748]
Centroid 2: [-2.70981136  8.97143336]
Centroid 3: [4.7182049   2.04179676]
Centroid 4: [-8.87357218  7.17458342]

Cluster Assignments:
Cluster 1: 75 points
Cluster 3: 75 points
Cluster 4: 74 points

Inertia (Sum of squared distances): 564.91
```

K-means Clustering