**Machine Learning Experiment 10**

**Title: Write a python program to implement K-Means clustering Algorithm.**

**Objective:** The objective of this lab assignment is to implement the K-Means clustering algorithm from scratch in Python and gain a deep understanding of how the algorithm works.

**Tasks:**

**Make dataset**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

X, y = make\_blobs(n\_samples=300, centers=4, random\_state=42, cluster\_std=1.0)

plt.scatter(X[:, 0], X[:, 1], s=50, c='gray')

plt.title('Generated Dataset')

plt.show()

**1. Implement the K-Means clustering algorithm from scratch in Python**

**Code:**

class KMeans:

def \_\_init\_\_(self, k, max\_iters=100):

self.k = k

self.max\_iters = max\_iters

self.centroids = None

def initialize\_centroids(self, X):

np.random.seed(42)

random\_indices = np.random.permutation(X.shape[0])

centroids = X[random\_indices[:self.k]]

return centroids

def assign\_clusters(self, X, centroids):

distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)

return np.argmin(distances, axis=1)

def update\_centroids(self, X, labels):

new\_centroids = np.array([X[labels == i].mean(axis=0) for i in range(self.k)])

return new\_centroids

def fit(self, X):

self.centroids = self.initialize\_centroids(X)

for i in range(self.max\_iters):

labels = self.assign\_clusters(X, self.centroids)

new\_centroids = self.update\_centroids(X, labels)

if np.all(self.centroids == new\_centroids):

break

self.centroids = new\_centroids

return self.centroids, labels

def inertia(self, X, labels):

return np.sum((X - self.centroids[labels]) \*\* 2)

**2. Visualize the dataset and plot the initial centroids.**

**Code:**

kmeans = KMeans(k=4)

initial\_centroids = kmeans.initialize\_centroids(X)

centroids, labels = kmeans.fit(X)

plt.scatter(X[:, 0], X[:, 1], s=50, c='gray', label='Data Points')

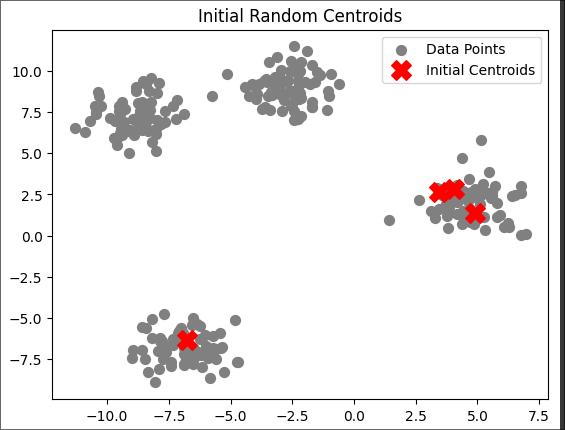
plt.scatter(initial\_centroids[:, 0], initial\_centroids[:, 1], s=200, c='red', marker='X', label='Initial Centroids')

plt.title('Initial Random Centroids')

plt.legend()

plt.show()

**Output:**

****

**3. Implement a method to calculate the sum of squared distances (inertia) for different values of K. Use this to determine an appropriate K for your dataset.**

**Code:**

plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis', label='Data Points')

plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='blue', marker='X', label='Final Centroids')

plt.title('K-Means Clustering with Final Centroids')

plt.legend()

plt.show()

inertia\_values = []

K\_values = range(1, 11)

for k in K\_values:

kmeans = KMeans(k=k)

centroids, labels = kmeans.fit(X)

inertia = kmeans.inertia(X, labels)

inertia\_values.append(inertia)

plt.plot(K\_values, inertia\_values, marker='o')

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

plt.xlabel('Number of clusters (K)')

plt.ylabel('Inertia (Sum of Squared Distances)')

plt.xticks(K\_values)

plt.grid()

plt.show()

for k, inertia in zip(K\_values, inertia\_values):

print(f"K = {k}, Inertia = {inertia:.2f}")

optimal\_k = K\_values[np.argmin(inertia\_values)]

print(f"The optimal K is: {optimal\_k}")

kmeans\_optimal = KMeans(k=optimal\_k)

centroids\_optimal, labels\_optimal = kmeans\_optimal.fit(X)

plt.scatter(X[:, 0], X[:, 1], c=labels\_optimal, s=50, cmap='viridis', label='Data Points')

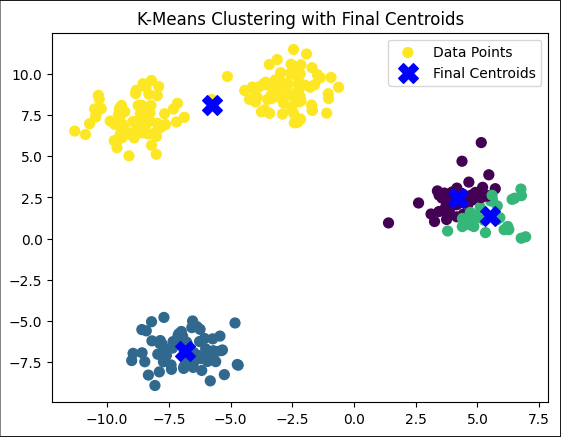
plt.scatter(centroids\_optimal[:, 0], centroids\_optimal[:, 1], s=200, c='blue', marker='X', label='Final Centroids')

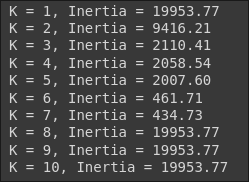
plt.title(f'K-Means Clustering with Optimal K = {optimal\_k}')

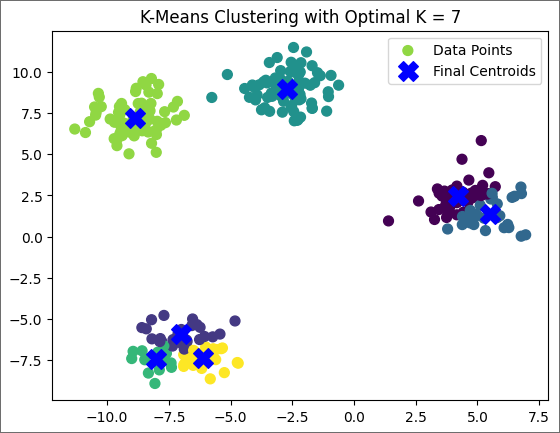
plt.legend()

plt.show()

**Output:**

**** *Clusters = 5*

* Sum of squared errors for different values of K*

**** *Least sum of squares for K = 7*