**K-means and K-medoids Clustering Algorithms**

Objective:

The objective of this lab manual is to help you understand and implement the K-means and K-medoids clustering algorithm using Python. You will apply the K-means algorithm to a dataset and visualize the results.

Introduction:

K-means is an unsupervised learning algorithm that partitions a dataset into K clusters. Each cluster is represented by its centroid, which is the mean of the points in the cluster. The algorithm iteratively updates the centroids by minimizing the sum of squared distances between each data point and its corresponding centroid.

Lab Outline:

1. Import the required libraries:

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.datasets import make\_blobs  from sklearn.preprocessing import StandardScaler |

1. Load and preprocess the dataset

|  |
| --- |
| # Generate a synthetic dataset with 300 samples and 2 features  data = make\_blobs(n\_samples=300, centers=4, n\_features=2, random\_state=42)  # Standardize the dataset  scaler = StandardScaler()  data\_scaled = scaler.fit\_transform(data[0]) |

1. Implement the K-means algorithm

|  |
| --- |
| def initialize\_centroids(data, k):  """Randomly initialize the centroids from the data points."""  # Your implementation here  def compute\_distances(data, centroids):  """Compute the distances between each data point and centroids."""  # Your implementation here  def assign\_clusters(distances):  """Assign each data point to the closest centroid."""  # Your implementation here  def update\_centroids(data, clusters, k):  """Update the centroids by computing the mean of the points in each cluster."""  # Your implementation here  def k\_means(data, k, max\_iterations=100):  """Implement the K-means clustering algorithm."""  # Your implementation here |

1. Evaluate the results

|  |
| --- |
| # Choose the number of clusters, K  k = 4  # Run the K-means algorithm  centroids, clusters = k\_means(data\_scaled, k)  # Compute the total within-cluster sum of squares  wcss = np.sum([np.sum(np.square(data\_scaled[clusters == i] - centroids[i])) for i in range(k)])  print("Total within-cluster sum of squares: ", wcss) |

1. Visualize the clusters

|  |
| --- |
| # Plot the dataset with the assigned clusters and centroids  plt.scatter(data\_scaled[:, 0], data\_scaled[:, 1], c=clusters, cmap='viridis')  plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x')  plt.xlabel('Feature 1')  plt.ylabel('Feature 2')  plt.title('K-means Clustering Results')  plt.show() |

1. For K-Medoids

|  |
| --- |
| def initialize\_medoids(data, k):  """Randomly initialize the medoids from the data points."""  # Your implementation here  def compute\_dissimilarities(data, medoids):  """Compute the dissimilarities between each data point and medoids."""  # Your implementation here  def assign\_clusters(dissimilarities):  """Assign each data point to the closest medoid."""  # Your implementation here  def update\_medoids(data, clusters, k):  """Update the medoids by selecting the data point with the minimum sum of dissimilarities in each cluster."""  # Your implementation here  def k\_medoids(data, k, max\_iterations=100):  """Implement the K-medoids clustering algorithm."""  # Your implementation here |

**Make sure to complete both K mean and K medoids with their visualization and testing and submit a single notebook**.