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
```

2. 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])
```

3. 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
```

4. 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)
```

5. 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()
```

6. 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.