```
#This code is written and submitted by Balkrishna Bhatt (201673048)

#Import the library
import numpy as np
import random
import matplotlib.pyplot as plt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

#Reading the data from datasets
data = pd.read_csv("dataset", delim_whitespace=True, index_col=0)

#Define the Euclidean distance function
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y)**2))

#K-Means clustring Implementation
def k_means_clustering(data, k, max_iterations=100):
    # Initialize the centroids randomly
    n = data.shape[0]
    centroids = data.sample(n=k).values

    # Initialize the cluster assignments and distances
    cluster_assignments = np.zeros(n)
    distances = np.zeros((n, k))

    # Iterate until convergence or until max_iterations is reached
    for i in range(max_iterations):
        # Assign each data point to the nearest centroid
        for j in range(n):
            for l in range(k):
                distances[j, l] = euclidean_distance(data.values[j], centroids[l])
            cluster_assignments[j] = np.argmin(distances[j])

        # Update the centroids to be the means of the data points assigned to them
        for l in range(k):
            centroids[l] = np.mean(data.values[cluster_assignments == l], axis=0)

    silhouette_coefficient = s_coefficient(data,n, cluster_assignments)
    return cluster_assignments, centroids, silhouette_coefficient

#K-Means++ clustring Implementation
def kmeanspp(data, k, max_iter=100):
    # Initialize the centroids array with the first randomly selected point
    n = data.shape[0]
    centroids = np.zeros((k, data.shape[1]))
    centroids[0] = data.sample(n=1).to_numpy()

    # Loop over the remaining centroids
    for i in range(1, k):
        # Calculate the distance from each point to the nearest centroid
        distances = np.array([min([np.linalg.norm(x - c) for c in centroids[:i]])
for x in data.to_numpy()])
        # Normalize the distances to create a probability distribution
        prob = distances / distances.sum()
        # Choose the next centroid randomly according to the probabilities
        centroids[i] = data.sample(n=1, weights=prob).to_numpy()
```

```
    #Assign each point to the nearest centroid
    labels = np.zeros(data.shape[0])
    for i, x in enumerate(data.to_numpy()):
        distances = np.array([np.linalg.norm(x - c) for c in centroids])
        labels[i] = np.argmin(distances)

    # Iterate until convergence
    for i in range(max_iter):
        # Update the centroids to the mean of the points in each cluster
        for j in range(k):
            points = data.to_numpy()[labels == j]
            if len(points) > 0:
                centroids[j] = np.mean(points, axis=0)

        # Assign each point to the nearest centroid
        new_labels = np.zeros(data.shape[0])
        for j, x in enumerate(data.to_numpy()):
            distances = np.array([np.linalg.norm(x - c) for c in centroids])
            new_labels[j] = np.argmin(distances)

        # Check for convergence
        if np.array_equal(labels, new_labels):
            break

        labels = new_labels

    silhouette_coefficient = s_coefficient(data,n, labels)

    return labels, centroids, silhouette_coefficient

#Compute the Silhouette coefficient
def s_coefficient(data, n, cluster_assignments):

    a = np.zeros(n)
    b = np.zeros(n)
    for j in range(n):
        cluster = int(cluster_assignments[j])
        members = np.nonzero(cluster_assignments == cluster)[0]
        if len(members) == 1:
            a[j] = 0
        else:
            a[j] = np.mean([euclidean_distance(data.values[j], data.values[m]) for
m in members if m != j])
        other_clusters = list(set(range(k)) - set([cluster]))
        if len(other_clusters) == 0:
            b[j] = 0
        else:
            b[j] = np.min([np.mean([euclidean_distance(data.values[j],
data.values[m]) for m in np.nonzero(cluster_assignments == c)[0]]) for c in
other_clusters])
    s = (b - a) / np.maximum(a, b)
    silhouette_coefficient = np.mean(s)
    return silhouette_coefficient

#Vary k from 1 to 9 and compute the Silhouette coefficient for each set of clusters
k_values = range(2, 10)
silhouette_coefficients = []
for k in k_values:
    _, _, silhouette_coefficient = k_means_clustering(data, k)
```

```
    silhouette_coefficients.append(silhouette_coefficient)

# Plot k on the horizontal axis and the Silhouette coefficient on the vertical axis

k_values = range(2, 10)
silhouette_coefficients = []
for k in k_values:
    _, _, silhouette_coefficient = k_means_clustering(data, k)
    silhouette_coefficients.append(silhouette_coefficient)
# Plot k on the horizontal axis and the Silhouette coefficient on the vertical axis

fig, axs = plt.subplots(3)
fig.suptitle('Diffrent Algos Of K-means with Silhouette coefficient implementaion')
fig.subplots_adjust(hspace=.5)
axs[0].plot(k_values, silhouette_coefficients, color='orange', marker='o',
linewidth=2, markersize=12)
axs[0].set_title("K-means Algo")

k_values = range(2, 10)
silhouette_coefficients = []
for k in k_values:
    _, _, silhouette_coefficient = kmeanspp(data, k)
    silhouette_coefficients.append(silhouette_coefficient)
# Plot k on the horizontal axis and the Silhouette coefficient on the vertical axis

axs[1].plot(k_values, silhouette_coefficients, color='blue', marker='X',
linewidth=2, markersize=12)
axs[1].set_title("K-means++ Algo")

plt.subplots_adjust(bottom=0.1)

for ax in axs.flat:
    ax.set(xlabel='K', ylabel='Silhouette coefficient')

#============================================================Bisecting K-
means============================================================================#

def load_word_embeddings(file_path):
    df = pd.read_csv(file_path, header=None, sep=' ', index_col=0)
    words = df.index.tolist()
    embeddings = df.values
    return words, embeddings

def get_closest_center(point, centers):
    distances = [euclidean_distance(point, center) for center in centers]
    return np.argmin(distances)

def kmeans(embeddings, k, max_iter=100):

    centers = embeddings[random.sample(range(embeddings.shape[0]), k)]

    for _ in range(max_iter):
        labels = np.array([get_closest_center(embedding, centers) for embedding in
embeddings])
        new_centers = np.array([embeddings[labels == i].mean(axis=0) for i in
range(k)])

        if np.allclose(centers, new_centers, rtol=1e-4):
            break
```

```
        else:
            centers = new_centers

    return labels, centers

#Bisecting K-Means Implementation
def bisecting_kmeans(embeddings, num_clusters):
    if num_clusters == 1:
        return [np.arange(embeddings.shape[0])]

    current_clusters = [embeddings]

    while len(current_clusters) < num_clusters:
        largest_cluster_idx = np.argmax([c.shape[0] for c in current_clusters])
        largest_cluster = current_clusters[largest_cluster_idx]

        labels, _ = kmeans(largest_cluster, 2)

        split_cluster_1 = largest_cluster[labels == 0]
        split_cluster_2 = largest_cluster[labels == 1]

        current_clusters[largest_cluster_idx] = split_cluster_1
        current_clusters.append(split_cluster_2)

    return current_clusters

#Compute the Silhouette coefficient
def silhouette_score(embeddings, labels):
    n_samples = len(embeddings)
    unique_labels = np.unique(labels)

    a = np.zeros(n_samples)
    b = np.full(n_samples, np.inf)
    epsilon = 1e-10

    for label in unique_labels:
        same_cluster_idxs = labels == label
        other_cluster_idxs = labels != label
        same_cluster = embeddings[same_cluster_idxs]
        other_cluster = embeddings[other_cluster_idxs]

        a[same_cluster_idxs] = np.array([np.mean([euclidean_distance(x, y) for y in
same_cluster if x is not y]) if len(same_cluster) > 1 else 0 for x in
same_cluster])

        for other_label in unique_labels:
            if other_label != label:
                temp_b = np.array([np.mean([euclidean_distance(x, y) for y in
other_cluster]) for x in same_cluster])
                b[same_cluster_idxs] = np.minimum(b[same_cluster_idxs], temp_b)

    s = (b - a) / (np.maximum(a, b) + epsilon)
    return np.mean(s)

def main():
    # Load word embeddings
    file_path = 'dataset'
    words, embeddings = load_word_embeddings(file_path)
```

```
    silhouette_scores = []

    for s in range(2, 10):
        clusters = bisecting_kmeans(embeddings, s)
        labels = np.array([label for label, cluster in enumerate(clusters) for _ in
range(cluster.shape[0])])
        score = silhouette_score(embeddings, labels)
        silhouette_scores.append(score)

    axs[2].plot(range(2, 10), silhouette_scores, color='green', marker='v',
linewidth=2, markersize=12)
    axs[2].set_title("Bisecting K-means Algo")
    plt.show()

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