```
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt




def kmean(data, num_clusters, norm,num_iters=100):  #function to compute kmeans
    if(norm=="True"):
        for i in range(0,len(data)):
            data[i]=data[i]/(np.linalg.norm(data[i],axis=0))
    n_samples = data.shape[0]
    n_features = data.shape[1]
    classifications = np.zeros(n_samples, dtype = np.int64)    #all the data points
are classified as 0 initially
    I = np.random.choice(n_samples, num_clusters)
    centroids = data[I, :]
    loss = 0
    for m in range(0, num_iters):                               #Based on the number
of iterations the clusters are computed
        for i in range(0, n_samples):
            distances = np.zeros(num_clusters)                  #For each iteration
and sample the distances are intialized as zero
            for j in range(0, num_clusters):
                distances[j] = np.sqrt(np.sum(np.power(data[i, :] - centroids[j],
2)))
            classifications[i] = np.argmin(distances)
    new_centroids = np.zeros((num_clusters, n_features))
    new_loss = 0
    for j in range(0, num_clusters):
        J = np.where(classifications == j)
        X_C = data[J]
        new_centroids[j] = X_C.mean(axis = 0)
    for i in range(0, X_C.shape[0]):
        new_loss += np.sum(np.power(X_C[i, :] - centroids[j], 2))
    cluster_assignments=classifications
    cluster_centroids = new_centroids
    loss = new_loss
    return loss, cluster_assignments, cluster_centroids


def kmedian(data, num_clusters,norm, num_iters=100):
    if(norm=="True"):
        for i in range(0,len(data)):
            data[i]=data[i]/(np.linalg.norm(data[i],axis=0))
    n_samples = data.shape[0]
    n_features = data.shape[1]
    classifications = np.zeros(n_samples, dtype = np.int64)
    I = np.random.choice(n_samples, num_clusters)
    centroids = data[I, :]
    loss = 0
    for m in range(0, num_iters):
        for i in range(0, n_samples):
            distances = np.zeros(num_clusters)
            for j in range(0, num_clusters):
                distances[j] = np.sum(abs(data[i, :] - centroids[j]))
            classifications[i] = np.argmin(distances)
```

```
    new_centroids = np.zeros((num_clusters, n_features))
    new_loss = 0
    for j in range(0, num_clusters):
        J = np.where(classifications == j)
        X_C = data[J]
        new_centroids[j] = np.median(X_C,axis = 0)
    for i in range(0, X_C.shape[0]):
        new_loss += np.sum(np.power(X_C[i, :] - centroids[j], 2))
    cluster_assignments=classifications
    cluster_centroids = new_centroids
    loss = new_loss
    return loss, cluster_assignments, cluster_centroids




def extract_data(filename):                                          #The data
of each file is processed
    instance=open(filename,'rt')
    instance=csv.reader(instance,delimiter=' ')                      #The file
is opened and read as a csv
    instance=list(instance)
    instance=np.array(instance)
    label=np.array([filename])
    instance_l=np.tile(label[np.newaxis,:],(instance.shape[0],1))    #An extra
column is added at the end for labels
    total_instances=np.concatenate((instance,instance_l),axis=1)
    return total_instances

def prepare_data(total_data):                                       #The
extracted data points are combined to form the concatenated dataset.
    animal=extract_data(total_data[0])
    countries=extract_data(total_data[1])
    fruits=extract_data(total_data[2])
    veggies=extract_data(total_data[3])
    total_data=np.concatenate((animal,countries,fruits,veggies),axis=0)
    return total_data

def prepare_for_b_cubed(cluster_assignments,data,k):
    len_animals=0
    len_countries=0
    len_fruits=0
    len_veggies=0
    store_clusters=dict()
    for i in range(0,k):
        store_clusters[i]=[]
    for i in range(0,len(data)):

        if(data[i][-1]=="animals"):
            print("x")
            len_animals=len_animals+1
        if(data[i][-1]=="countries"):
            len_countries=len_countries+1
        if(data[i][-1]=="fruits"):
            len_fruits=len_fruits+1
        if(data[i][-1]=="veggies"):
            len_veggies=len_veggies+1
```

```
    for i in range(0,len(cluster_assignments)):

        if(i<(len_animals-1)):
            print(cluster_assignments[i])
            store_clusters[cluster_assignments[i]].append("animals")
        if(i>(len_animals-1) and i<(len_animals+len_countries-1)):
            store_clusters[cluster_assignments[i]].append("countries")
        if(i>(len_animals+len_countries-1) and
i<(len_animals+len_countries+len_fruits-1)):
            store_clusters[cluster_assignments[i]].append("fruits")
        if(i>(len_animals+len_countries+len_fruits-1) and
i<(len_animals+len_countries+len_fruits+len_veggies-1)):
            store_clusters[cluster_assignments[i]].append("veggies")
    list_precision=[]
    list_recall=[]
    print(store_clusters)
    for i in range(0,len(store_clusters)):
        num_animals=0
        num_countries=0
        num_fruits=0
        num_veggies=0
        for j in range(0,len(store_clusters[i])):
            if(store_clusters[i][j]=="animals"):
                num_animals=num_animals+1

            if(store_clusters[i][j]=="countries"):
                num_countries=num_countries+1

            if(store_clusters[i][j]=="fruits"):
                num_fruits=num_fruits+1

            if(store_clusters[i][j]=="veggies"):
                num_veggies=num_veggies+1
        print(num_animals)
        print(len(store_clusters[i]))
        precision=num_animals/len(store_clusters[i])
        list_precision.append(precision)
        recall=num_animals/len_animals
        list_recall.append(recall)

        precision=num_countries/len(store_clusters[i])
        list_precision.append(precision)
        recall=num_countries/len_countries
        list_recall.append(recall)

        precision=num_fruits/len(store_clusters[i])
        list_precision.append(precision)
        recall=num_fruits/len_fruits
        list_recall.append(recall)

        precision=num_veggies/len(store_clusters[i])
        list_precision.append(precision)
        recall=num_veggies/len_veggies
        list_recall.append(recall)
    avg_precision=sum(list_precision)/len(list_precision)
    avg_recall=sum(list_recall)/len(list_recall)
    avg_f1_score=2*((avg_precision*avg_recall)/(avg_precision+avg_recall))
    return avg_precision,avg_recall,avg_f1_score
    #print(data[0][-1])
```

```
def bcubed(cluster_assignments,data,k):
#BCubed prescision, recall and f1 score is calculated
    len_animals=0
    len_countries=0
    len_fruits=0
    len_veggies=0
    store_clusters=dict()
    animals_clusters=[]
    countries_clusters=[]
    fruits_clusters=[]
    veggies_clusters=[]
    for i in range(0,len(data)):
#The number of datapoints in each class is calculated and stored.

        if(data[i][-1]=="animals"):
            len_animals=len_animals+1
        if(data[i][-1]=="countries"):
            len_countries=len_countries+1
        if(data[i][-1]=="fruits"):
            len_fruits=len_fruits+1
        if(data[i][-1]=="veggies"):
            len_veggies=len_veggies+1
    animal_clusters=cluster_assignments[0:len_animals]
#After applying k means or k medians the cluster assignments for each class of data
is retrieved and stored.
    countries_clusters=cluster_assignments[len_animals:(len_animals+len_countries)]
    fruits_clusters=cluster_assignments[(len_animals+len_countries):
(len_animals+len_countries+len_fruits)]
    veggies_clusters=cluster_assignments[(len_animals+len_countries+len_fruits):
(len_animals+len_countries+len_fruits+len_veggies)]
    TP=0
    FP=0
    FN=0
    TN=0
#The process of bcubed precision is started
    for i in range(0,len(animal_clusters)):
        for j in range(0,len(animal_clusters)):
            if(j>i and i!=j):
                if(animal_clusters[i]==animal_clusters[j]):
                    TP=TP+1
                else:
                    FN=FN+1
        for j in range(0,len(countries_clusters)):
            if(animal_clusters[i]==countries_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1
        for j in range(0,len(fruits_clusters)):
            if(animal_clusters[i]==fruits_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1
        for j in range(0,len(veggies_clusters)):
            if(animal_clusters[i]==veggies_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1

    for i in range(0,len(countries_clusters)):
```

```
        for j in range(0,len(countries_clusters)):
            if(j>i and i!=j):
                if(countries_clusters[i]==countries_clusters[j]):
                    TP=TP+1
                else:
                    FN=FN+1
        for j in range(0,len(fruits_clusters)):
            if(countries_clusters[i]==fruits_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1
        for j in range(0,len(veggies_clusters)):
            if(countries_clusters[i]==veggies_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1

    for i in range(0,len(fruits_clusters)):

        for j in range(0,len(fruits_clusters)):
            if(j>i and i!=j):
                if(fruits_clusters[i]==fruits_clusters[j]):
                    TP=TP+1
                else:
                    FN=FN+1
        for j in range(0,len(veggies_clusters)):
            if(fruits_clusters[i]==veggies_clusters[j]):
                FP=FP+1
            else:
                TN=TN+1
    for i in range(0,len(veggies_clusters)):

        for j in range(0,len(veggies_clusters)):
            if(j>i and i!=j):
                if(veggies_clusters[i]==veggies_clusters[j]):
                    TP=TP+1
                else:
                    FN=FN+1
    precision=TP/(TP+FP)
    recall=TP/(TP+FN)
    f1_score=2*((precision*recall)/(precision+recall))
    return precision,recall,f1_score
def run_plot_results(data_mod,data,algo,norm):
#based on the parameters the function runs the clustering algorithms from k=1 to 9
and generated the plots.
    list_prec=[]
    list_recall=[]
    list_f1=[]
    list_k=[]
    if(algo=="kmeans"):
        for i in range(1,10):
            print("i")
            print(i)
            loss, cluster_assignments, cluster_centroids=kmean(data_mod,i,norm)

#precision,recall,f1_score=prepare_for_b_cubed(cluster_assignments,data,i)
            precision,recall,f1_score=bcubed(cluster_assignments,data,i)
            print("Precision: "+str(precision))
```

```
            print("Recall: "+str(recall))
            print("F1-score: "+str(f1_score))
            list_prec.append(precision)
            list_recall.append(recall)
            list_f1.append(f1_score)
            list_k.append(i)

        plt.plot(list_k,list_prec , label = "Precision",linewidth=6)
        plt.plot(list_k,list_recall, label = "Recall",linewidth=6)
        plt.plot(list_k,list_f1, label = "F1 Score",linewidth=6)
        plt.xlabel('No. of clusters',fontsize=20)
        plt.xticks(fontsize=20)
        plt.ylabel('BCUBED metrics',fontsize=20)
        plt.yticks(fontsize=20)
        if(norm=="False"):
            plt.title('Clustering using K-means algorithm',fontsize=20)
        else:
            plt.title('Clustering using K-means algorithm and L2 Norm',fontsize=20)
        plt.legend(fontsize=20)
        plt.show()
    elif(algo=="kmedian"):                                                  #The
kmeans algorithm is utilized for clustering the data for k=1 to 9 and then generate
the plots
        for i in range(1,10):
            loss, cluster_assignments, cluster_centroids=kmedian(data_mod,i,norm)

#precision,recall,f1_score=prepare_for_b_cubed(cluster_assignments,data,i)
            precision,recall,f1_score=bcubed(cluster_assignments,data,i)
            print("Precision: "+str(precision))
            print("Recall: "+str(recall))
            print("F1-score: "+str(f1_score))
            list_prec.append(precision)
            list_recall.append(recall)
            list_f1.append(f1_score)
            list_k.append(i)
        plt.plot(list_k,list_prec , label = "Precision",linewidth=6)
        plt.plot(list_k,list_recall, label = "Recall",linewidth=6)
        plt.plot(list_k,list_f1, label = "F1 Score",linewidth=6)
        plt.xlabel('No. of clusters',fontsize=20)
        plt.xticks(fontsize=20)
        plt.ylabel('BCUBED metrics',fontsize=20)
        plt.yticks(fontsize=20)
        if(norm=="False"):
            plt.title('Clustering using K-median algorithm',fontsize=20)
        else:
            plt.title('Clustering using K-medians algorithm and L2
Norm',fontsize=20)
        plt.legend(fontsize=20)
        plt.show()


if __name__ == "__main__":
    file_names=["animals","countries","fruits","veggies"]
    data=prepare_data(file_names)

    data=np.delete(data,0,axis=1)                                        #The names of
the objects present in the first word of each line is removed.
    data_mod=np.delete(data,-1,axis=1)                                  #The labels are
removed to prepare the dataset for clustering
```

```
    data_mod=np.array(data_mod).astype(np.float)                    #The data is
converted to float
    #k=KMeans()
    #x=k.fit(data_mod)


    #Q="1"   #Uncomment to run Question 1
    #Q="2"   #Uncomment to run Question 2
    #Q="3"   #Uncomment to run Question 3
    #Q="4"   #Uncomment to run Question 4
    #Q="5"   #Uncomment to run Question 5
    #Q="6"   #Uncomment to run Question 6
    Q="run_plots"   # Runs questions 3-6
    if(Q=="1"):
        print("Question 1 run started")
        norm="False"
        loss, cluster_assignments, cluster_centroids=kmean(data_mod,4,"False")
        precision,recall,f1_score=bcubed(cluster_assignments,data,4)
        print("Precision: "+str(precision))
        print("Recall: "+str(recall))
        print("F1-score: "+str(f1_score))
        print("Question 1 run completed")
    if(Q=="2"):
        print("Question 2 run started")
        norm="False"
        loss, cluster_assignments, cluster_centroids=kmedian(data_mod,4,"False")
        precision,recall,f1_score=bcubed(cluster_assignments,data,4)
        print("Precision: "+str(precision))
        print("Recall: "+str(recall))
        print("F1-score: "+str(f1_score))
        print("Question 2 run completed")
    if(Q=="3"):
        print("Question 3 run started")
        norm="False"
        run_plot_results(data_mod,data,"kmeans",norm)
        print("Question 3 run completed")
    if(Q=="4"):
        print("Question 4 run started")
        norm="True"
        run_plot_results(data_mod,data,"kmeans",norm)
        print("Question 4 run completed")
    if(Q=="5"):
        print("Question 5 run started")
        norm="False"
        run_plot_results(data_mod,data,"kmedian",norm)
        print("Question 5 run completed")
    if(Q=="6"):
        print("Question 6 run started")
        norm="True"
        run_plot_results(data_mod,data,"kmedian",norm)
        print("Question 6 run completed")
    if(Q=="run_plots"):
        print("Question 3 run started")
        norm="False"
        run_plot_results(data_mod,data,"kmeans",norm)
        print("Question 3 run completed")
        print("Question 4 run started")
        norm="True"
        run_plot_results(data_mod,data,"kmeans",norm)
```

```
    print("Question 4 run completed")
    print("Question 5 run started")
    norm="False"
    run_plot_results(data_mod,data,"kmedian",norm)
    print("Question 5 run completed")
    print("Question 6 run started")
    norm="True"
    run_plot_results(data_mod,data,"kmedian",norm)
    print("Question 6 run completed")




#extract_information()
```