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

def readArray():
  ## Read the files and combine them into a single array for processing

  arr=[]
  animals=pd.read_csv('animals',sep=' ',header=None)
  countries=pd.read_csv('countries',sep=' ',header=None)
  fruits=pd.read_csv('fruits',sep=' ',header=None)
  veggies=pd.read_csv('veggies',sep=' ',header=None)
  arr1=animals.to_numpy()
  arr2=countries.to_numpy()
  arr3=fruits.to_numpy()
  arr4=veggies.to_numpy()
  arr = np.concatenate((arr1, arr2, arr3, arr4))
  originall=arr.copy()
  new=np.delete(arr,0,axis=1)

  return new,originall

def chooseRandomCentroid(origin,n):
  ## Choose first set of random centroid from the dataset based on the value of k

  random.seed(5)
  centroid=random.choices(origin, k=n)
  centroid=np.array(centroid)

  return centroid

def normalise(arr):
  ## Normalise the input array

  data=[[0]*300]*327
  #print(arr)
  for i in range(327):
    x=arr[i]
    n=np.linalg.norm(x)
    data[i]=x/n
  data=np.array(data)

  return data

def findEuclidianDist(arr1,arr2):
  ## Find Euclidian Distance for the given set of points

  sq = np.sum(np.square(arr1 - arr2))
  distance = (np.sqrt(sq))

  return distance

def findManhattanDist(arr1,arr2):
  ## Find Manhattan Distance for the given set of points

  distance = sum(abs(val1-val2) for val1, val2 in zip(arr1,arr2))

  return distance
```

```
def findCluster(n,centroid,origina,flag):
  ## Find the distance between each data point and centroid
  ## Create a new array which denotes the cluster each point belong to

  mini=[[100.0]*327]*9
  mini=np.array(mini)
  cluster=[['none']*327]*9
  cluster=np.array(cluster)
  sam=[[0.0]*327]*9
  for j in range(n):
    string='c'+str(j)
    c=0
    for x in range(327):
      if flag == 1:
        dist=findEuclidianDist(origina[x],centroid[j])
        #dist = np.linalg.norm(origina[x]-centroid[j],axis=0)
      elif flag == 2:
        dist=findManhattanDist(origina[x],centroid[j])
      sam[n-1][x]=dist

      if mini[n-1][x] > dist :
        mini[n-1][x]= dist
        cluster[n-1][x]=string
        c= c+1
    #print(sam)
    #print('cluster count ',c)
    #print(cluster.shape)    ##   (9, 327)
    #print(mini.shape)    ##   (9, 327)

  return cluster

def categorise(n,cluster,oriiginal):
  ## Categorise each points into diferent arrays based on clusters to calculate
mean/median

  c_arr=[[[0]*300]*327]*n
  fin=[]
  categ=[]
  for i in range(n):
    c=0
    string='c'+str(i)
    for j in range(327):
      if cluster[j]==string:
        c_arr[i][c]=oriiginal[j]
        c=c+1
      if j ==326:
        for k in range(327):
          if not(np.count_nonzero(c_arr[i][k]) == 0):
            fin.append(c_arr[i][k])
    categ.append(c)
    c_arr=[[[0]*300]*327]*n

  return fin,categ

def findMean(carr,cat_list):
  ## Find mean based on the array created as per the generated cluster

  x=0
  ar_fin=[[0]*300]*327
  arr_centroid=[]
  for i in range(len(cat_list)):
    for j in range(cat_list[i]):
      ar_fin[j]=carr[x]
      x=x+1
    arr_centroid.append(np.mean(ar_fin,axis=0))
    ar_fin=[[0]*300]*327

  return arr_centroid

def findMedian(carr,cat_list):
  ## Find median based on the array created as per the generated cluster

  x=0
  ar_fin=[[0]*300]*327
  arr_centroid=[]
  for i in range(len(cat_list)):
    for j in range(cat_list[i]):
      ar_fin[j]=carr[x]
      x=x+1
    arr_centroid.append(np.median(ar_fin,axis=0))
    ar_fin=[[0]*300]*327

  return arr_centroid

def display(cluster,n,original):
  ## Display the clusters each point belongs to

  dis=[]
  for i in range(n):
    arr_dis=[]
    string='c'+str(i)
    for j in range(327):
      if cluster[j]==string:
        arr_dis.append(original[j][0])
    dis.append(arr_dis)

  return dis

def findNum(cluster,n,):
  ## Find Number of animals, fruits, vegetables, countries in each cluster


  animal=[0]*n
  country=[0]*n
  fruit=[0]*n
  veg=[0]*n
  for i in range(n):
    string='c'+str(i)
    for j in range(327):
      if cluster[j] == string:
        if j<50:
          animal[i]=animal[i]+1
        elif j< 211:
          country[i]=country[i]+1
        elif j< 269:
          fruit[i]=fruit[i]+1
        else:
          veg[i]=veg[i]+1

  return animal,country,fruit,veg

def BCube(al,cy,ft,vg,ctr_list,n):
  ## Find the B-Cubed precision, recall, fscore

  ## Precision calculation
  pre=0
  for i in range(n):
    if ctr_list[i] != 0:
      pre=pre+(al[i]*al[i]/ctr_list[i])+(cy[i]*cy[i]/ctr_list[i])+(ft[i]*ft[i]/
ctr_list[i])+(vg[i]*vg[i]/ctr_list[i])
  pre=pre/327

  ## Recall calculation
  rcal=0
  for i in range(n):
    rcal=rcal+(al[i]*al[i]/50)+(cy[i]*cy[i]/161)+(ft[i]*ft[i]/58)+(vg[i]*vg[i]/58)
  rcal=rcal/327

  ## F-Score calculation
  fsc=0
  for i in range(n):
    a=0
    b=0
    c=0
    d=0
    if ctr_list[i] != 0:
      if ((al[i]*al[i]/ctr_list[i])+(al[i]*al[i]/50)) != 0:
        a =
((2*(al[i]*al[i]/ctr_list[i])*(al[i]*al[i]/50))/((al[i]*al[i]/ctr_list[i])+
(al[i]*al[i]/50)))
      if ((cy[i]*cy[i]/ctr_list[i])+(cy[i]*cy[i]/161)) != 0:
        b =
((2*(cy[i]*cy[i]/ctr_list[i])*(cy[i]*cy[i]/161))/((cy[i]*cy[i]/ctr_list[i])+
(cy[i]*cy[i]/161)))
      if ((ft[i]*ft[i]/ctr_list[i])+(ft[i]*ft[i]/58)) != 0:
        c =
((2*(ft[i]*ft[i]/ctr_list[i])*(ft[i]*ft[i]/58))/((ft[i]*ft[i]/ctr_list[i])+
(ft[i]*ft[i]/58)))
      if ((vg[i]*vg[i]/ctr_list[i])+(vg[i]*vg[i]/58)) != 0:
        d =
((2*(vg[i]*vg[i]/ctr_list[i])*(vg[i]*vg[i]/58))/((vg[i]*vg[i]/ctr_list[i])+
(vg[i]*vg[i]/58)))
    fsc=fsc+ a + b + c + d
  fsc=fsc/327

  return pre, rcal, fsc

def plotGraph(pre,rec,fsc):
  ## Plot graph based on the precison, recall and fscore

  k=[1,2,3,4,5,6,7,8,9]
  plt.plot(k,pre,color="blue",label='Precision')
  plt.plot(k,rec,color="red",label='Recall')
  plt.plot(k,fsc,color="green",label='F-Score')
  plt.xlabel('K Value')
  plt.ylabel('B-Cubed Values')
  plt.legend(loc="upper right")
  plt.show()

  return

############################
##  Implementing K-Means  ##
############################

p = [0]*9
r = [0]*9
f = [0]*9
for i in range(1,10):
#for i in range(4,5):
  print("\t K Means with k = ",i)
  print("\t ********************")
  arr_new,original=readArray()
  cnt=0
  old_centroid=[[0.0]*300]*i
  new_centroid=chooseRandomCentroid(arr_new,i)
  while not(np.array_equal(old_centroid,new_centroid)):
    #print('Iteration ',cnt)
    clust=findCluster(i,new_centroid,arr_new,1)
    cat_arr,c_list=categorise(i,clust[i-1],arr_new)
    old_centroid=new_centroid
    new_centroid=findMean(cat_arr,c_list)
    new_centroid=np.array(new_centroid)
    cnt= cnt+1

  #a_dis=display(clust[i-1],i,original)
  #for n in range (i):
  #  print(a_dis[n],'\n')
  #print("\n")

  print('Total number of Iteration taken for k=',i, " is ",cnt)

  ani,cou,fru,vegg=findNum(clust[i-1],i)
  p[i-1],r[i-1],f[i-1] = BCube(ani,cou,fru,vegg,c_list,i)



#############################
##  Implementing K-Median  ##
#############################

pp = [0]*9
rr = [0]*9
ff = [0]*9
for i in range(1,10):
#for i in range(4,5):
  print("\t K Median with k = ",i)
  print("\t ********************")
  arr_new,original=readArray()
  cnt=0
  old_centroid=[[0.0]*300]*i
  new_centroid=chooseRandomCentroid(arr_new,i)
  while not(np.array_equal(old_centroid,new_centroid)):
    #print('Iteration ',cnt)
    clust=findCluster(i,new_centroid,arr_new,2)
    cat_arr,c_list=categorise(i,clust[i-1],arr_new)
    old_centroid=new_centroid
    new_centroid=findMedian(cat_arr,c_list)
    new_centroid=np.array(new_centroid)
    cnt= cnt+1
  #a_dis=display(clust[i-1],i,original)
  #for n in range (i):
  #  print(a_dis[n],'\n')
  #print("\n")

  print('Total number of Iteration taken for k=',i, " is ",cnt)
  #print(c_list)
  ani,cou,fru,vegg=findNum(clust[i-1],i)
  pp[i-1],rr[i-1],ff[i-1] = BCube(ani,cou,fru,vegg,c_list,i)



#########################
####  Plot the Graph ####
#########################
print("\n Plotting the graph\n")
print("K-Means")
print("Precision: ", p)
print("Recall: ", r)
print("F Score: ",f)
print('\n\n')

print("K_Median")
print("Precision: ", pp)
print("Recal: ", rr)
print("F Score: ",ff)
print('\n\n')

plt.title('K-Mean Comparison')
plotGraph(p,r,f)
plt.title('K-Median Comparison')
plotGraph(pp,rr,ff)

##############################################
##  Implementing K-Means with Normalisation ##
##############################################

p = [0]*9
r = [0]*9
f = [0]*9
for i in range(1,10):
#for i in range(4,5):
  print("\t K Means with normalisation k = ",i)
  print("\t ********************************")
  arrnew,original=readArray()
  arr_new = normalise(arrnew)
  cnt=0
  old_centroid=[[0.0]*300]*i
  new_centroid=chooseRandomCentroid(arr_new,i)
  while not(np.array_equal(old_centroid,new_centroid)):
    #print('Iteration ',cnt)
    clust=findCluster(i,new_centroid,arr_new,1)
    cat_arr,c_list=categorise(i,clust[i-1],arr_new)
    old_centroid=new_centroid
    new_centroid=findMean(cat_arr,c_list)
    new_centroid=np.array(new_centroid)
    cnt= cnt+1
  #a_dis=display(clust[i-1],i,original)
  #for n in range (i):
  #  print(a_dis[n],'\n')
  #print("\n")
  print('Total number of Iteration taken for k=',i, " is ",cnt)

  ani,cou,fru,vegg=findNum(clust[i-1],i)
  p[i-1],r[i-1],f[i-1] = BCube(ani,cou,fru,vegg,c_list,i)


###############################################
##  Implementing K-Median with normalisation ##
###############################################

pp = [0]*9
rr = [0]*9
ff = [0]*9
for i in range(1,10):
#for i in range(4,5):
  print("\t K Median with with normalisation k = ",i)
  print("\t ********************************")
  arrnew,original=readArray()
  arr_new = normalise(arrnew)
  cnt=0
  old_centroid=[[0.0]*300]*i
  new_centroid=chooseRandomCentroid(arr_new,i)
  while not(np.array_equal(old_centroid,new_centroid)):
    #print('Iteration ',cnt)
    clust=findCluster(i,new_centroid,arr_new,2)
    cat_arr,c_list=categorise(i,clust[i-1],arr_new)
    old_centroid=new_centroid
    new_centroid=findMedian(cat_arr,c_list)
    new_centroid=np.array(new_centroid)
    cnt= cnt+1
  #a_dis=display(clust[i-1],i,original)
  #for n in range (i):
  #  print(a_dis[n],'\n')
  #print("\n")

  print('Total number of Iteration taken for k=',i, " is ",cnt)

  ani,cou,fru,vegg=findNum(clust[i-1],i)
  pp[i-1],rr[i-1],ff[i-1] = BCube(ani,cou,fru,vegg,c_list,i)



####################################
####  Plot the Normalised Graph ####
####################################
print("\n Plotting the graph\n")
print("K-Means with Normalisation")
print("Precision: ", p)
print("Recall: ", r)
print("F Score: ",f)
print('\n\n')
print("K_Median with Normalisation")
print("Precision: ", pp)
print("Recal: ", rr)
print("F Score: ",ff)
print('\n\n')

plt.title('K-Mean with Normalisation')
plotGraph(p,r,f)
plt.title('K-Median with Normalisation')
plotGraph(pp,rr,ff)
```