############################################### reading image #########################################################

image\_path = "//Users//pratheek//Desktop//photos//pratheek.jpeg"

from PIL import Image

im = Image.open(image\_path)

r, g, b = list(im.getdata(0)), list(im.getdata(1)), list(im.getdata(2))

x=[]

for i in range(len(r)):

x.append([])

for i in range (len(r)):

x[i].append(r[i])

x[i].append(g[i])

x[i].append(b[i])

########################################################################################################################

############################################# k mean clusture code #####################################################

import random

import matplotlib as pl

def distan(x,y): #this function calcuate the distace between two points

l=(((x[0]-y[0])\*\*2)+((x[1]-y[1])\*\*2)+((x[2]-y[2])\*\*2))\*\*0.5

return l

def listminimumindex(l): #this function gives the index of the minimum element in list

min = 0

for i in range (len(l)):

if l[min]>l[i] :

min = i

return min

#print(listminimumindex(l))

def distancelist(l,x): #this function finds the length of distance between a point and the centroid and return the centroid wich is least distance from the point

output\_list=[]

for i in range(len(l)):

output\_list.append(distan(l[i],x))

return listminimumindex(output\_list)

#print(distancelist([[1,1],[1,2],[1,3],[1,4]],[1,2.5]))

def similar\_check(l): #this function finds similar elements in list if it finds similar elements in list return 0

for i in range (len(l)):

for j in range(i+1,len(l)):

if l[i]==l[j]:

return 0

def rand\_dif\_list(l,k):

listrandom = random.sample(l, k)

if similar\_check(listrandom)==0:

return rand\_dif\_list(l,k)

else:

return listrandom

def k\_mean\_cluster\_r(l,k):#this function takes random point as centroid and assign all the points to the respective centre which is near

nvariable=[] #if k=3 then n variable is becoming [ [], [], [] ]

for i in range (k):

nvariable.append([])

len\_of\_list = len(l)

listrandom = rand\_dif\_list(l,k)

for i in range (len(l)):

nvariable[distancelist(listrandom,l[i])].append(l[i])

return [nvariable,listrandom]

#print(k\_mean\_cluster\_r(X,8))

def centroidl(l): # finding the centroid of set of points

cent=[0,0,0]

for i in range (len(l)):

cent[0] += l[i][0]

cent[1] += l[i][1]

cent[2] += l[i][2]

cent[0]=cent[0]/len(l)

cent[1]=cent[1]/len(l)

cent[2]=cent[2]/len(l)

return cent

#print(centroidl([[1,1],[1,2],[1,3],[1,4]]))

def k\_mean\_clusture\_alpha(l,centre):

sample\_centre=[] #this list is to find the centroid of list l(l is of type list in a list in a list

for i in range(len(l)):

sample\_centre.append(centroidl(l[i]))

if sample\_centre == centre :

return centre

else:

x=[]

for i in range(len(l)):

x.append([])

for i in range(len(l)):

for j in range (len(l[i])):

x[distancelist(sample\_centre, l[i][j])].append(l[i][j])

return k\_mean\_clusture\_alpha(x,sample\_centre)

#print( k\_mean\_clusture\_alpha((k\_mean\_cluster\_r(X,2))[0],(k\_mean\_cluster\_r(X,2))[1]))

def k\_mean\_clusture(l,k):

return k\_mean\_clusture\_alpha((k\_mean\_cluster\_r(l,k))[0],(k\_mean\_cluster\_r(l,k))[1])

print(k\_mean\_clusture(x,4))

########################################################################################################################

################################################### elbow curve ########################################################

def variance(l,k):

x=0

for i in range(len(l)):

x=x+ (distan(l[i],k))\*\*2

return x

#print(variance([[1,2],[1,3],[1,4],[1,5]],[1,3.5]))