```
#!/usr/bin/python3

import pandas as pd
import random
import time
import numpy as np
import matplotlib.pyplot as plt
import sys

def hexColor():
    return ''.join([random.choice('0123456789ABCDEF') for x in range(6)])

class METRICS(object):
    """
    Input: Takes in two matrices of size (1 x n)
           Reshape the input matrices for this class to work.
           trueLabels = input1.reshape(1,input1.size)
    Usage: metrics=METRICS(trueLables = input1.reshape(1,input1.size,generatedLabels=input2.re
shape(1, input2.size))
    """
    def __init__(self, trueLabels, generatedLabels):
        self.tL = trueLabels
        self.gL = generatedLabels

    def randIndex(self):
        self.rand=(self.tL == self.tL.T).astype(int) == (self.gL == self.gL.T).astype(int)
        return (self.rand.sum()/self.rand.size)

    def jaccardIndex(self):
        jaccard=((self.tL == self.tL.T).astype(int) & (self.gL == self.gL.T).astype(int)).sum(
)/((self.tL == self.tL.T).astype(int) | (self.gL == self.gL.T).astype(int)).sum()
        return jaccard

class KMEANS(object):
    import numpy as np
    def __init__(self, k, metricOrder):
        self.k=k
        self.ord=metricOrder
        self.maxIterations=1000

    def setMaxIter(self,maxIterations):
        self.maxIterations=maxIterations

    def getDB(self, DB):
        self.DB=DB
        self.labelCentroids={}
        self.iterations=0
        self.oldCentroids=np.empty(shape=(self.k,self.DB.shape[1]))
        self.labelData=np.concatenate((np.ndarray((self.DB.shape[0],1)),self.DB), axis=1) #App
end cluster id 0 to all data

    def fit(self, DB, init):
        self.getDB(DB)
        self.initCentroids(init) #Initialize centroids randomly from available dataset
        while(not self.shouldStop()):
            self.oldCentroids=np.copy(self.centroids)
#             print(self.oldCentroids)
            self.iterations +=1
            self.labels=self.getLabels()
#             print(self.oldCentroids)
            self.getCentroids()
#             print(self.centroids)
#             print(self.oldCentroids)
```

```
    def initCentroids(self, init):
        if(len(init)==k):
            self.centroids=self.DB[init,:]
        elif(self.iterations==0):
            print("Incorrect initialization. Using random centroids.")
            perm=np.random.permutation(self.DB.shape[0])
            self.centroids=self.DB[perm[0:self.k]]

    def getCentroids(self):
        for i in np.unique(self.labelData[:,0]):
            self.labelCentroids[int(i)]=self.centroids[int(i)]=self.labelData[self.labelData[:
,0] == int(i)].mean(0)[1:]

    def getLabels(self):
        for i in range(len(self.DB)):
            dist = np.linalg.norm(self.DB[i] - self.centroids, ord=self.ord, axis=1)
            self.labelData[i][0] = np.argmin(dist)
        return self.labelData[:,0]

    def shouldStop(self):
        return ((np.linalg.norm(km.oldCentroids-km.centroids)==0)|(self.iterations>self.maxIte
rations))

file=sys.argv[1]
k=int(sys.argv[2])
metricOrder=int(sys.argv[3])

data=np.genfromtxt(file, delimiter="\t")[:,2:]
X=(data - data.mean(0))
true_labels=np.array(list(pd.read_csv(file, sep='\t', lineterminator='\n', header=None).iloc[:
,1]))
###############################################################################
#  Compute the covariance matrix
S=(1/(X.shape[0]))*X.T.dot(X)
###############################################################################
#  Compute and extract the eigen vectors from the covariance matrix
eigen_vectors=np.linalg.eig(S)[1]
###############################################################################
#  Select the first two columns from the eigen vector table as the principal components
#  recompute samples based on principal components
pca_plotData=data.dot(eigen_vectors[:,0:2])

km=KMEANS(k=k, metricOrder=metricOrder)

km.centroids=np.load("centroids.npy")

km.getDB(X)

km.labels=km.getLabels()

metrics=METRICS(km.labels.reshape(1,len(km.labels)),true_labels.reshape(1,len(true_labels)))

df_pca = pd.DataFrame(dict(x=list(pca_plotData[:,0]),y=list(pca_plotData[:,1]), labels=km.labe
ls))
lb=list(set(km.labels.astype(int)))

print("Jaccard Index: "+str(metrics.jaccardIndex()))
print("Rand Index: "+str(metrics.randIndex()))
print(df_pca['labels'].groupby(df_pca['labels']).describe()['count'])

###############################################################################
#  Plotting the dataframe with labels
```

```
fig = plt.figure()
ax1 = fig.add_subplot(111)
for i in range(len(lb)):
    ax1.scatter(df_pca[df_pca['labels']==lb[i]]['x'], df_pca[df_pca['labels']==lb[i]]['y'], co
lor=("#"+hexColor()), label=lb[i])
plt.legend(loc='upper left')
plt.title("KMEANS-MR plot on "+file)
plt.savefig('KMEANS_MR_plot_'+file[:-4]+'.pdf')
plt.show()
```