import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans

from sklearn import datasets

from sklearn import preprocessing

import pylab as pl

from sklearn import decomposition

import seaborn as sns

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn import metrics

from sklearn.cluster import AgglomerativeClustering

import scipy.cluster.hierarchy as hc

from sklearn.decomposition import PCA

from sklearn.cluster import DBSCAN

##Reference:

 ## https://scikit-learn.org/stable/modules/clustering.html

## https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

#bring in the data

#filename="Small\_RecordData.csv"

filename="Small\_kmeans\_dataset.csv"

smalldata=pd.read\_csv(filename)

print(type(smalldata))

print(smalldata)

#####

# KMEANS

# Use k-means clustering on the data.

# Create clusters

k = 4

## Sklearn required you to instantiate first

kmeans = KMeans(n\_clusters=k)

kmeans.fit(smalldata) ## run kmeans

labels = kmeans.labels\_

print(labels)

centroids = kmeans.cluster\_centers\_

print(centroids)

prediction = kmeans.predict(smalldata)

print(prediction)

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

##

## Try to predict new data vectors

##

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

NewData=[0, 3, 4, 0, 1, 2, 3, 0, 0, 1, 0]

#NewData=[3.6, 90, 169, 7, 91]

print(type(NewData))

NewData=np.asarray(NewData)

#print(NewData)

print(NewData.shape)

NewData=NewData.reshape(1,-1)

print(NewData.shape)

#NewData=np.transpose(NewData)

#print(NewData)

#print(kmeans.predict([[3.6, 90, 169, 7, 91]]))

#print(kmeans.predict([[2.8, 70, 139, 2, 61]]))

#print(kmeans.predict(NewData))

print(kmeans.predict(NewData))

print(kmeans.predict([[4, 3, 0, 0, 1, 0, 3, 7, 0, 0, 0]]))

print(kmeans.predict([[0, 0, 4, 6, 1, 0, 1, 0, 0, 0, 0]]))

print(kmeans.predict([[0, 0, 1, 6, 6, 7, 8, 0, 1, 0, 1]]))

print(kmeans.predict([[0, 0, 1, 1, 0, 1, 1, 0, 4, 4, 5]]))

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

##

## Look at best values for k

##

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

SS\_dist = []

values\_for\_k=range(2,7)

#print(values\_for\_k)

for k\_val in values\_for\_k:

 print(k\_val)

 k\_means = KMeans(n\_clusters=k\_val)

 model = k\_means.fit(smalldata)

 SS\_dist.append(k\_means.inertia\_)

print(SS\_dist)

print(values\_for\_k)

plt.plot(values\_for\_k, SS\_dist, 'bx-')

plt.xlabel('value')

plt.ylabel('Sum of squared distances')

plt.title('Elbow method for optimal k Choice')

plt.show()

####

# Look at Silhouette

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

Sih=[]

Cal=[]

k\_range=range(2,8)

for k in k\_range:

 k\_means\_n = KMeans(n\_clusters=k)

 model = k\_means\_n.fit(smalldata)

 Pred = k\_means\_n.predict(smalldata)

 labels\_n = k\_means\_n.labels\_

 R1=metrics.silhouette\_score(smalldata, labels\_n, metric = 'euclidean')

 R2=metrics.calinski\_harabasz\_score(smalldata, labels\_n)

 Sih.append(R1)

 Cal.append(R2)

print(Sih) ## higher is better

print(Cal) ## higher is better

fig1, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)

ax1.plot(k\_range,Sih)

ax1.set\_title("Silhouette")

ax1.set\_xlabel("")

ax2.plot(k\_range,Cal)

ax2.set\_title("Calinski\_Harabasz\_Score")

ax2.set\_xlabel("k values")

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

##

## Look at Clusters

##

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

# It is often best to normalize the data

## before applying the fit method

## There are many normalization options

## This is an example of using the z score

smalldata\_normalized=(smalldata - smalldata.mean()) / smalldata.std()

print(smalldata\_normalized)

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

## PCA -

## or principle component analysis

## can be used to identify the principle comp

## the vectors (columns) with the highest eigenvalue

## or most distinct/largest variation from the other components

## This is a method of dimensionality reduction

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

print(smalldata\_normalized.shape[0]) ## num rows

print(smalldata\_normalized.shape[1]) ## num cols

NumCols=smalldata\_normalized.shape[1]

## Instantiated my own copy of PCA

My\_pca = PCA(n\_components=2) ## I want the two prin columns

## Transpose it

smalldata\_normalized=np.transpose(smalldata\_normalized)

My\_pca.fit(smalldata\_normalized)

print(My\_pca)

print(My\_pca.components\_.T)

KnownLabels=["food", "food", "hike", "hike", "pet", "pet", "sports", "sports", "sports" ]

# Reformat and view results

Comps = pd.DataFrame(My\_pca.components\_.T,

 columns=['PC%s' % \_ for \_ in range(2)],

 index=smalldata\_normalized.columns

 )

print(Comps)

print(Comps.iloc[:,0])

#RowNames = list(Comps.index)

#print(RowNames)

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

## Look at 2D PCA clusters

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

plt.figure(figsize=(12,12))

plt.scatter(Comps.iloc[:,0], Comps.iloc[:,1], s=100, color="green")

plt.xlabel("PC 1")

plt.ylabel("PC 2")

plt.title("Scatter Plot Clusters PC 1 and 2",fontsize=15)

for i, label in enumerate(KnownLabels):

 #print(i)

 #print(label)

 plt.annotate(label, (Comps.iloc[i,0], Comps.iloc[i,1]))

plt.show()

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

##

## DBSCAN

##

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

MyDBSCAN = DBSCAN(eps=6, min\_samples=2)

## eps:

 ## The maximum distance between two samples for

 ##one to be considered as in the neighborhood of the other.

MyDBSCAN.fit\_predict(smalldata)

print(MyDBSCAN.labels\_)

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

##

## Hierarchical

##

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

MyHC = AgglomerativeClustering(n\_clusters=4, affinity='euclidean', linkage='ward')

FIT=MyHC.fit(smalldata)

HC\_labels = MyHC.labels\_

print(HC\_labels)

plt.figure(figsize =(12, 12))

plt.title('Hierarchical Clustering')

dendro = hc.dendrogram((hc.linkage(smalldata, method ='ward')))

## WARD

## Recursively merges the pair of clusters that

## minimally increases within-cluster variance.

from sklearn.metrics.pairwise import euclidean\_distances

EDist=euclidean\_distances(smalldata)

print(EDist)