**Homework 4**

**Problem 1**

1. Use hierarchical clustering with average linkage and Euclidean distance to cluster this data set.

First I imported my dataset as well as the linkage function from scipy as well as the pdist function. Using the pdist function I found the Euclidean distance of the dataset and stored it into a variable Euc\_dist. I then used the linkage function using the Euc\_dist variable to find the linkage. See code below:

# -\*- coding: utf-8 -\*-

"""

Created on Fri Jun 18 10:33:16 2021

@author: 16319

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv("student.csv")

from scipy.cluster.hierarchy import linkage

from scipy.spatial.distance import pdist

Euc\_dist = pdist(data, 'euclidean')

hc\_linkage = linkage(Euc\_dist, "average")

print(hc\_linkage)

Results:

**[[0.00000000e+00 1.00000000e+00 1.00000000e+00 2.00000000e+00]**

**[3.50000000e+01 3.60000000e+01 1.00000000e+00 2.00000000e+00]**

**[6.10000000e+01 6.20000000e+01 1.41421356e+00 2.00000000e+00]**

**...**

**[1.23100000e+03 1.23400000e+03 1.24733851e+02 2.45000000e+02]**

**[1.23300000e+03 1.23500000e+03 1.88717043e+02 3.75000000e+02]**

**[1.23600000e+03 1.23700000e+03 3.12249594e+02 6.20000000e+02]]**

1. Cut the dendrogram at a height that results in three distinct clusters. Which observations belong to which clusters?

I first imported the dendrogram function from scipy. Then using matplotlib I created my figure to plot my dendrogram. Using the dendrogram function I plotted the clusters. See code below:

from scipy.cluster.hierarchy import dendrogram

plt.figure(figsize=(25, 10))

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('sample index')

plt.ylabel('distance')

dendrogram(

hc\_linkage,

leaf\_rotation=90., # rotates the x axis labels

leaf\_font\_size=8., # font size for the x axis labels

)

plt.show()

from scipy.cluster.hierarchy import cut\_tree

print(cut\_tree(hc\_linkage, n\_clusters = 3).T)

plt.figure(figsize=(25, 10))

plt.title('Hierarchical Clustering Dendrogram')

plt.xlabel('sample index')

plt.ylabel('distance')

dendrogram(

hc\_linkage,

leaf\_rotation=90.,

leaf\_font\_size=8.,

)

plt.show()

print(cut\_tree(hc\_linkage, n\_clusters = 3).T)

Results:

[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

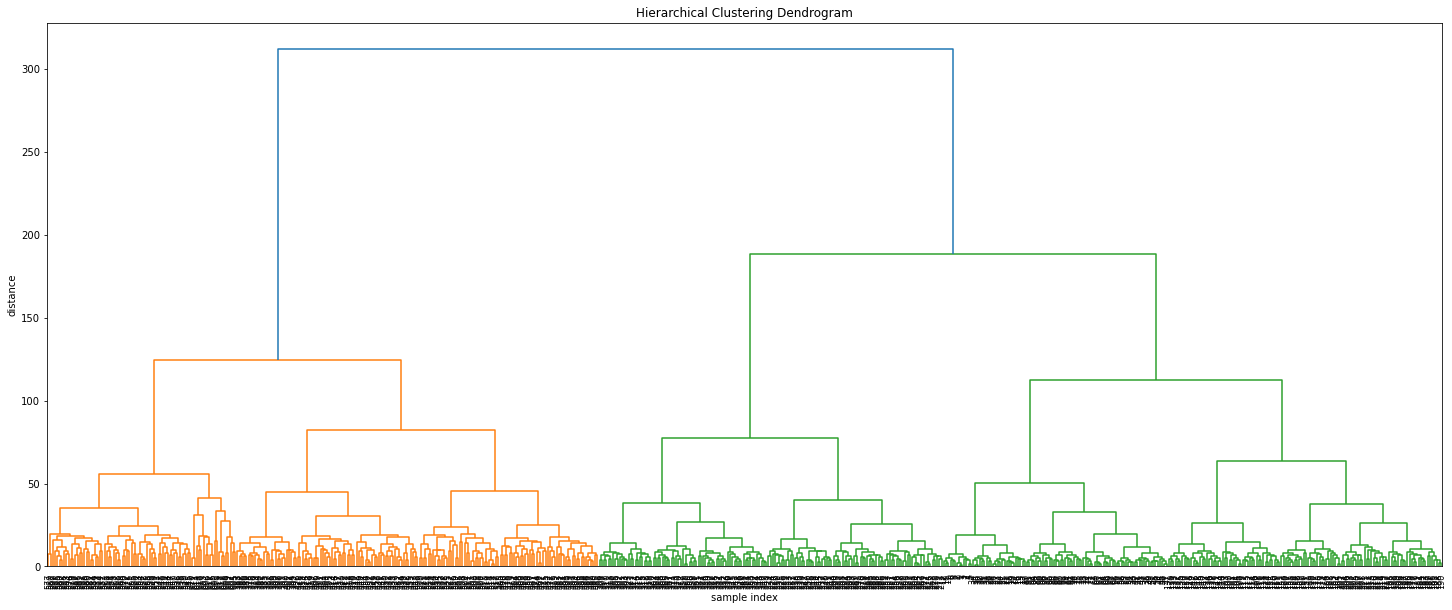
…………

…………

…………

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2]]



1. Use the elbow method to find the number,Kof clusters needed toperformK−means clustering.

I first imported the KMeans function from sklearn.cluster. I then set my cluster range from 2-10 and created an empty list called inertia. Then I used a for loop to calculate the inertia at each # of clusters in the cluster range. I then plotted these values onto a scree plot. By looking at the figure you can see the elbow occurs at 3 clusters. See code below:

from sklearn.cluster import KMeans

cluster\_range = np.arange(2,11, 1)

inertia = []

for i in cluster\_range:

kmeans = KMeans(init = 'k-means++', n\_clusters = i, n\_init = 20, random\_state =0).fit(data)

inertia.append(kmeans.inertia\_)

plt.figure()

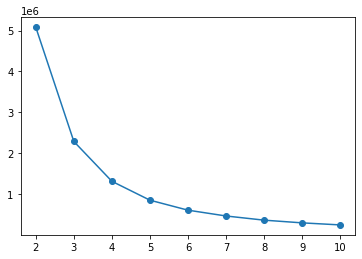
plt.plot(cluster\_range, inertia, marker = 'o')

plt.show()

print(inertia)

Results:

[5080837.580645157, 2290727.2095975825, 1317504.1537357997, 856013.9484766668, 611956.4622386323, 470081.4078070477, 368243.76291537425, 302112.92063578096, 251916.29204242842]



1. Perform K−means clustering of the observations with K= 3. Compare your results with the hierarchical clustering results for (2).

I first set a variable kmeans equal to the KMeans function using 3 clusters fitted to our student dataset. Using the labels\_ attribute to obtain the clusters I plot the clusters onto a scatterplot. Comparing the two graphs I noticed some differences so I wrote a for loop to compare the assigned classes for each observation. See code below:

kmeans = KMeans(n\_clusters =3 , random\_state = 123).fit(data)

print(kmeans.labels\_)

data = pd.DataFrame(data)

plt.figure(figsize=(6,5))

plt.scatter(data.iloc[:,0], data.iloc[:,1], c = kmeans.labels\_, cmap = plt.cm.bwr)

plt.scatter(kmeans.cluster\_centers\_[:, 0],

kmeans.cluster\_centers\_[:, 1],

marker = '\*',

s = 150,

color = 'cyan',

label = 'Centers')

plt.legend(loc = 'best')

plt.xlabel('X0')

plt.ylabel('X1')

compare1 = cut\_tree(hc\_linkage, n\_clusters = 3)

compare2 = kmeans.labels\_

result = np.zeros(len(compare1))

for i in range (len(compare1)):

if (compare1[i] == compare2[i]):

result[i] = 1

else:

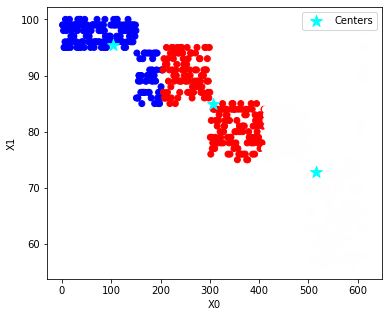
result[i] = 0

comparison = np.mean(result)

print("Percent of equivalent clusters:", comparison\*100)

Results:

Percent of equivalent clusters: 38.54838709677419



1. Perform PCA on the observations and plot the first two principal com-ponent score vectors.

First I imported scale and PCA from sklearn. I then stored my scaled data into a data frame called X. I then used the PCA() function to obtain the loading vectors and stored them into a data frame called pca\_loadings. I then verified the loading vectors by taking the dot product between them obtaining a result of 1. I then used the pca function to obtain the score vectors. Once I had obtained the score vectors I plotted them. See code below:

from sklearn.decomposition import PCA

from sklearn.preprocessing import scale

X = pd.DataFrame(scale(data), index=data.index, columns = data.columns)

pca\_loadings = pd.DataFrame(PCA().fit(X).components\_.T, index=data.columns, columns =['V1','V2','V3','V4'])

pca\_loadings

#dot\_prod = about 1 to check loading vectors

dot\_prod = pca\_loadings["V1"].dot(pca\_loadings["V1"])

pca = PCA()

data\_plot = pd.DataFrame(pca.fit\_transform(X), columns = ['PC1', 'PC2', 'PC3', 'PC4'], index=X.index)

fig, ax1 = plt.subplots(figsize=(9,7))

ax1.set\_xlim(-3.5,3.5)

ax1.set\_ylim(-3.5,3.5)

#Plotting PC's #1 #2

for i in data\_plot.index:

ax1.annotate(i, (data\_plot.PC1.loc[i], -data\_plot.PC2.loc[i]), ha='center')

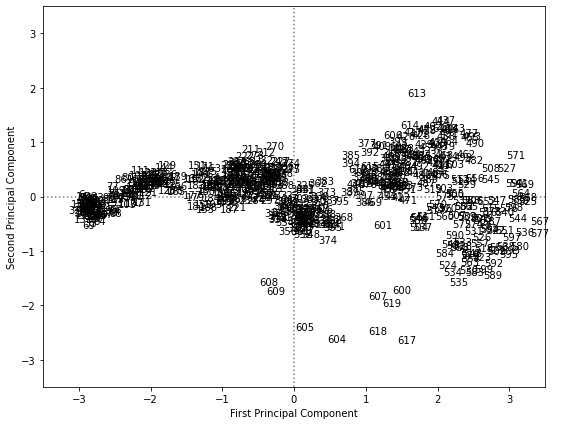
#Reference lines

ax1.hlines(0,-3.5,3.5, linestyles='dotted', colors = 'grey')

ax1.vlines(0,-3.5,3.5, linestyles='dotted', colors='grey')

ax1.set\_xlabel('First Principal Component')

ax1.set\_ylabel('Second Principal Component')



1. Now perform K−means clustering with K= 3 on the erst two principal component score vectors, rather than on the raw data. Compare your results with the K−means clustering results of (4).

First I created two zero vectors with rows equal to the observations of our dataset. I then stored the first two PC scores into these vectors using a for loop. I then stored these vectors into a data frame called PC\_Scores\_df. I then used the KMeans function on this data frame to obtain the clusters. I then compared them. See code below:

#Now performing K-means clustering with K = 3 on score vectros

PC1\_Scores = np.zeros(shape = [len(data\_plot), 1])

PC2\_Scores = np.zeros(shape = [len(data\_plot), 1])

for i in range(len(data\_plot)):

PC1\_Scores[i] = data\_plot.PC1.loc[i]

PC2\_Scores[i] = data\_plot.PC2.loc[i]

PC\_Scores\_df = pd.DataFrame(data = PC1\_Scores, columns = ["PC1"])

PC\_Scores\_df["PC2"] = PC2\_Scores

kmeans1 = KMeans(n\_clusters =3 , random\_state = 123).fit(PC\_Scores\_df)

print(kmeans1.labels\_)

plt.figure(figsize=(6,5))

plt.scatter(data.iloc[:,0], data.iloc[:,1], c = kmeans1.labels\_, cmap = plt.cm.bwr)

plt.scatter(kmeans1.cluster\_centers\_[:, 0],

kmeans1.cluster\_centers\_[:, 1],

marker = '\*',

s = 150,

color = 'cyan',

label = 'Centers')

plt.legend(loc = 'best')

plt.xlabel('X0')

plt.ylabel('X1')

compare\_1 = kmeans.labels\_

compare\_2 = kmeans1.labels\_

result\_ = np.zeros(len(compare\_1))

for i in range (len(compare\_1)):

if (compare\_1[i] == compare\_2[i]):

result\_[i] = 1

else:

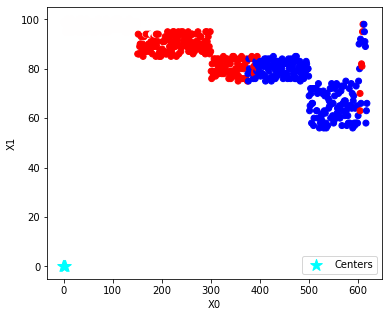
result\_[i] = 0

comparison\_ = np.mean(result)

print("Percent of equivalent clusters:", comparison\_\*100)

Results:

Percent of equivalent clusters: 38.54838709677419



**Problem 2**

1. Determine the proportion of (standardized) sample variance due to the first two sample principal components, the first three sample principal components.

I scaled and standardized the student data in the previous problem. First, I performed principal component analysis with only two components and stored the result in a data frame called data\_plot\_2. I then printed the explained variance ratio of this analysis. I then repeated the process for 3 components. See code below:

pca2 = PCA(n\_components = 2)

data\_plot\_2 = pd.DataFrame(pca2.fit\_transform(X), columns = ['PC1', 'PC2'])

print("Proportion of Variance of first 2 components:", pca2.explained\_variance\_ratio\_)

pca3 = PCA(n\_components = 3)

data\_plot\_3 = pd.DataFrame(pca3.fit\_transform(X), columns = ['PC1', 'PC2', 'PC3'])

print("Proportion of Varince of first 3 components:", pca3.explained\_variance\_ratio\_)

Results:

Proportion of Variance of first 2 components: [0.8508123 0.08054016]

Proportion of Varince of first 3 components: [0.8508123 0.08054016 0.04482171]

1. Decide on the number of sample principal components to retain using the proportion of variance explained.

The proportion of variance explained drops off significantly after the 2nd principal component. This can be seen visually as well in the graph. Due to this we should choose 2 sample principal components.

1. Decide on the number of sample principal components to retain using ‘scree’ plot.

I plotted the explained variance ratio against the number of principal components. See code below:

plt.figure(figsize=(7,5))

plt.plot([1,2,3,4], pca.explained\_variance\_ratio\_, '-o', label='Individual component')

plt.plot([1,2,3,4], np.cumsum(pca.explained\_variance\_ratio\_), '-s', label='Cumulative')

plt.ylabel('Proportion of Variance Explained')

plt.xlabel('Principal Component')

plt.xlim(0.75,4.25)

plt.ylim(0,1.05)

plt.xticks([1,2,3,4])

plt.legend(loc=2);

