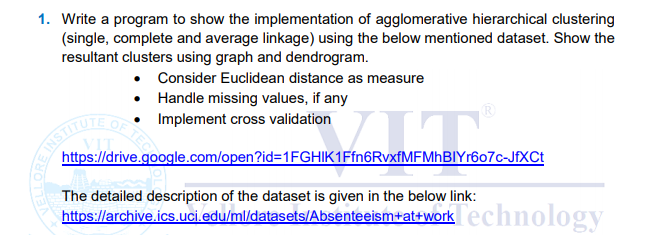
Name: Shaunak Sensarma Registration Number: 18BCE2527

Web Mining Lab (CSE-3024)

Web Content Mining, Web Usage Mining – Assessment – 4

Faculty: Dr. Shashank Mouli Satapathy.

**Question No.1:**



**Code:**

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

@author: Shaunak\_Sensarma

"""

import pandas as pd

from sklearn.model\_selection import cross\_val\_score

from sklearn.tree import DecisionTreeClassifier

print("Dataset\n")

absent\_data = pd.read\_excel("absent.xls", names=[i for i in range(21)])

print(absent\_data.head())

print()

print()

#**Agglomerative hierarchical clustering...**

from matplotlib import pyplot as plt

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.cluster import KMeans

import numpy as np

data\_part = absent\_data[[0, 1]]

kmeans = KMeans(n\_clusters=2, init="k-means++", max\_iter=300, n\_init=10, random\_state=0)

pred\_y = kmeans.fit(data\_part)

cross\_val\_score(kmeans, data\_part, cv=5)

plt.scatter(data\_part[0], data\_part[1])

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=50, c="red")

plt.show()

print("Dictionary format..\n")

mydict = {i: np.where(kmeans.labels\_ == i)[0] for i in range(kmeans.n\_clusters)}

print(mydict)

**#clustering..**

from scipy.cluster import hierarchy

Z = hierarchy.linkage(data\_part, "single")

plt.figure()

dn = hierarchy.dendrogram(Z)

hierarchy.set\_link\_color\_palette(["m", "c", "y", "k"])

fig, axes = plt.subplots(1, 2, figsize=(8, 3))

dn1 = hierarchy.dendrogram(Z, ax=axes[0], above\_threshold\_color="y", orientation="top")

dn2 = hierarchy.dendrogram(

Z, ax=axes[1], above\_threshold\_color="#bcbddc", orientation="right"

)

hierarchy.set\_link\_color\_palette(None) # reset to default after use

plt.show()

Z = hierarchy.linkage(data\_part, "complete")

plt.figure()

dn = hierarchy.dendrogram(Z)

hierarchy.set\_link\_color\_palette(["m", "c", "y", "k"])

fig, axes = plt.subplots(1, 2, figsize=(8, 3))

dn1 = hierarchy.dendrogram(Z, ax=axes[0], above\_threshold\_color="y", orientation="top")

dn2 = hierarchy.dendrogram(

Z, ax=axes[1], above\_threshold\_color="#bcbddc", orientation="right"

)

hierarchy.set\_link\_color\_palette(None) # reset to default after use

plt.show()

Z = hierarchy.linkage(data\_part, "average")

plt.figure()

dn = hierarchy.dendrogram(Z)

hierarchy.set\_link\_color\_palette(["m", "c", "y", "k"])

fig, axes = plt.subplots(1, 2, figsize=(8, 3))

dn1 = hierarchy.dendrogram(Z, ax=axes[0], above\_threshold\_color="y", orientation="top")

dn2 = hierarchy.dendrogram(

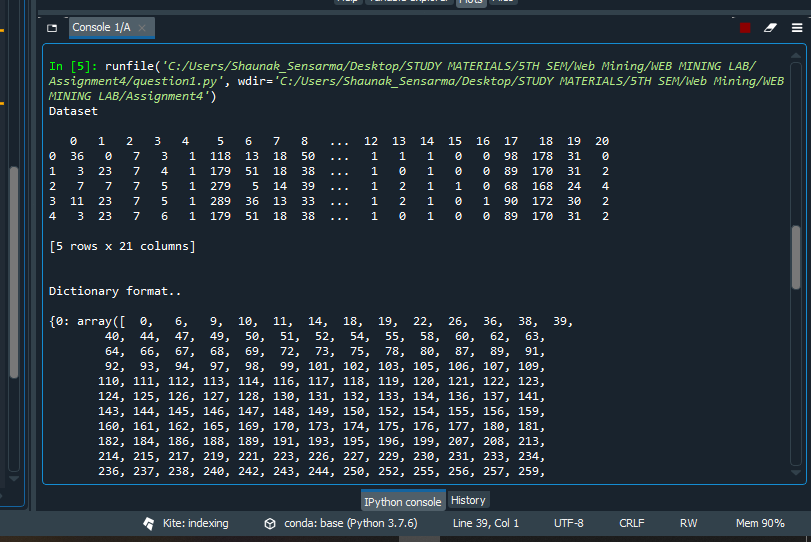
Z, ax=axes[1], above\_threshold\_color="#bcbddc", orientation="right"

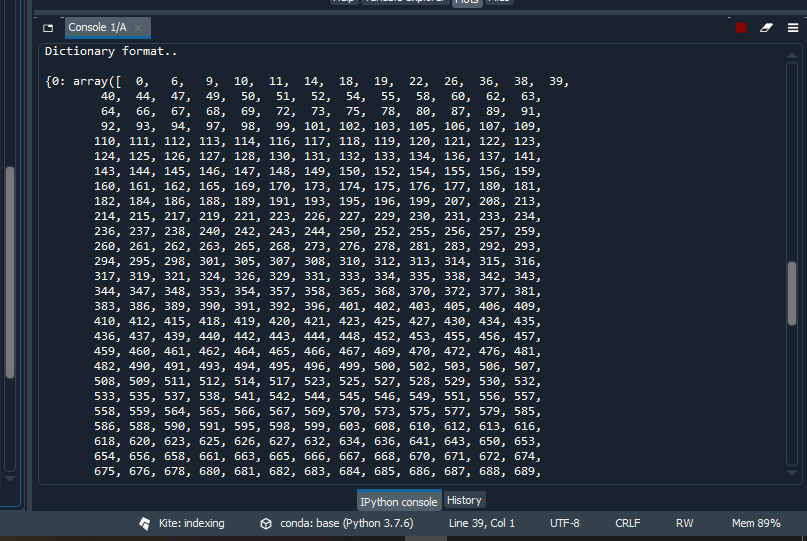
)

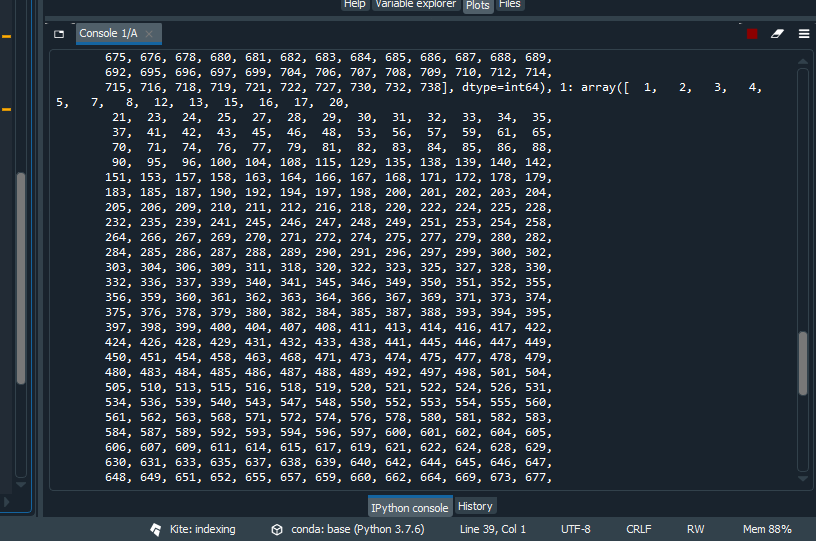
hierarchy.set\_link\_color\_palette(None) # reset to default after use

plt.show()

**OUTPUT:**

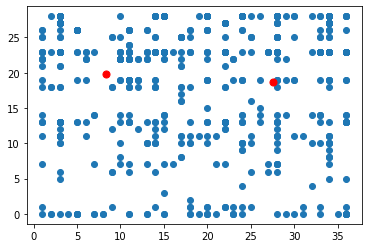




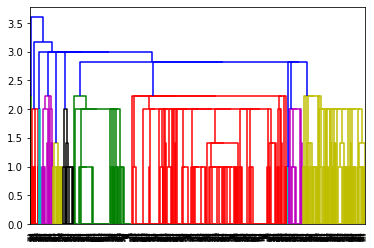


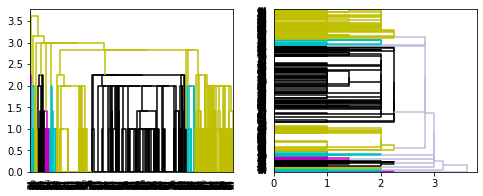
**PLOTS:**

K-Means Clustering….

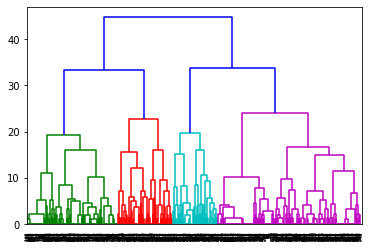


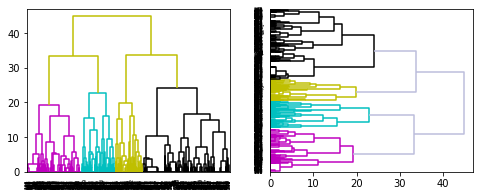
Single Clustering….



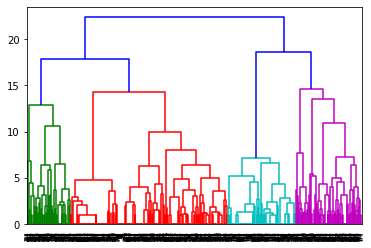


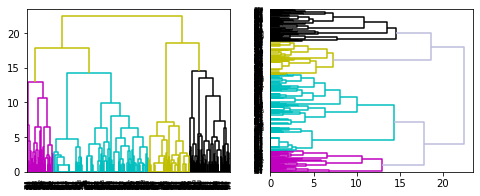
Complete Clustering..



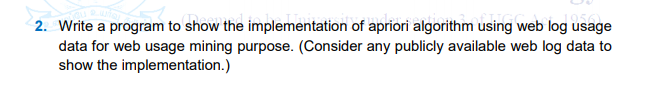


Average Clustering..





**Question No.2:**



**Code:**

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

"""

Created on Wed Oct 14 19:40:17 2020

@author: Shaunak\_Sensarma

"""

import pandas as pd

df = pd.read\_csv("weblog.csv")

print(df.head(3))

print()

print(df.columns)

print()

new\_df = df[["IP", "URL"]]

print(new\_df.tail(3))

def extract\_name(x):

p1 = x.find("/")

p2 = x.find(".php")

return x[p1 + 1 : p2]

print()

print(extract\_name("GET /showcode.php?id=309&nm=ham05 HTTP/1.1"))

print()

new\_df["URL"] = new\_df["URL"].apply(extract\_name)

print("\nLength of URL....")

print(len(new\_df["URL"].unique()))

print("\n\nLength of IP...")

len(new\_df["IP"].unique())

print()

print(new\_df.head(3))

ips\_final = list(new\_df["IP"].unique())[:5]

print("\n\nFinal IP's..\n")

print(ips\_final)

print()

print(list(set(new\_df[new\_df["IP"] == ips\_final[0]]["URL"].values))[:10])

dict\_visited = {x: [] for x in ips\_final}

print()

print(dict\_visited)

print("\n\n\n")

for a in range(len(ips\_final)):

dict\_visited[ips\_final[a]] = list(

set(new\_df[new\_df["IP"] == ips\_final[a]]["URL"].values)

)

print(dict\_visited["10.128.2.1"][:4])

print("\n\n")

print("Visited length..")

for a in dict\_visited:

print(len(dict\_visited[a]))

size = 56

for a in dict\_visited:

temp = dict\_visited[a]

temp.extend(["0" for a in range(size - len(temp))])

dict\_visited[a] = temp

for a in dict\_visited:

print(len(dict\_visited[a]))

df\_visited = pd.DataFrame(dict\_visited)

print("\n\n")

print(df\_visited.tail(5))

print("\n\n")

print("Visited Shape..")

print(df\_visited.shape)

dataset = []

for x in range(df\_visited.shape[1]):

dataset.append(df\_visited[df\_visited.columns[x]].values)

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()

te\_ary = te.fit(dataset).transform(dataset)

df\_ml = pd.DataFrame(te\_ary, columns=te.columns\_)

print("\n\n")

print(df\_ml.head(3))

df\_ml = df\_ml.replace(True, 1)

print("\n\n")

print(df\_ml.head(3))

lis\_cls = df\_ml.columns

print()

print(lis\_cls[0])

from itertools import combinations

combs = list(combinations(lis\_cls, 2))

print("\n\n\nCombinations..")

print(combs[100])

print("\n\n\n Shape..")

df\_ml.shape

print("\n\n\n Value Counts....")

print(df\_ml["/home"].value\_counts()[1])

temp\_df = df\_ml[["/home", "fonts/fontawesome-webfont.woff2 HTTP/1."]]

print(temp\_df.head(1))

len(

temp\_df[

(temp\_df["/home"] == 1.0)

& (temp\_df["fonts/fontawesome-webfont.woff2 HTTP/1."] == 1.0)

]

)

def support(tup, tup1):

support = 0.80

x1, x2 = tup,tup1

# print(x1, x2)

supportx1 = df\_ml[x1].value\_counts()[1]/5

# print(supportx1)

if supportx1<support:

return 0

else:

# supportx2 = df\_ml[x2].value\_counts()[1]

temp\_df = df\_ml[[x1, x2]]

supportBoth = len(temp\_df[(temp\_df[x1] == 1.0) & (temp\_df[x2] == 1.0)])

conf = supportBoth/supportx1

return conf

combs[1][1]

print("\n\n\nSupport Combination...")

print(support(combs[1][0],combs[1][1]))

threshold = 0.9

print("\n\n\nLength of combination....")

print(len(combs))

list\_deci = []

for a in range(len(combs)):

sup\_store = support(combs[a][0],combs[a][1])

# print(sup\_store)

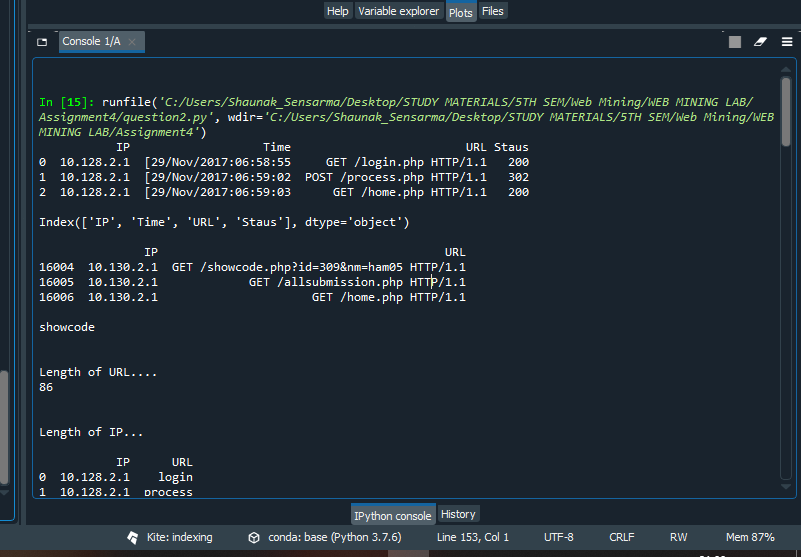
if sup\_store>threshold:

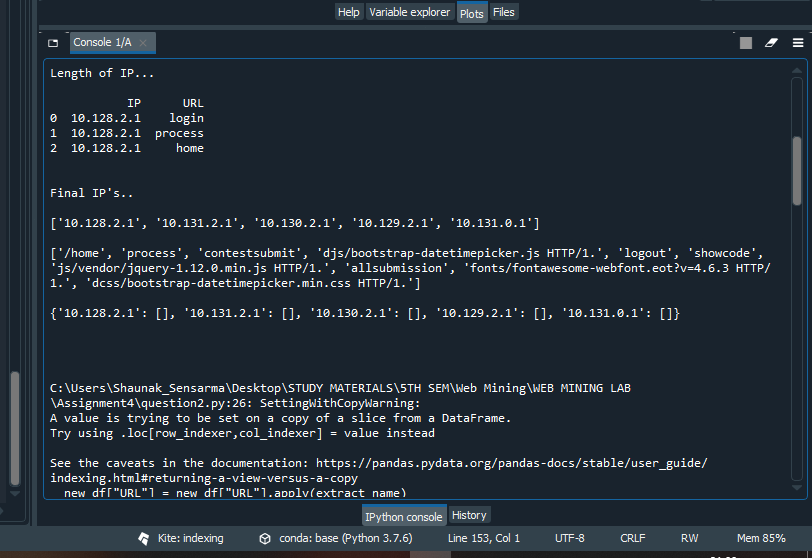
list\_deci.append(combs[a])

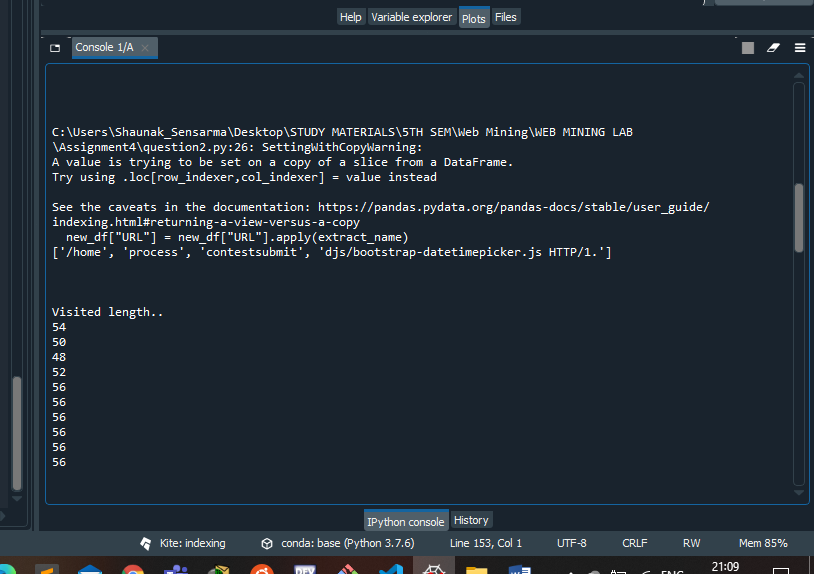
print("\n\n\nLength of list of set..")

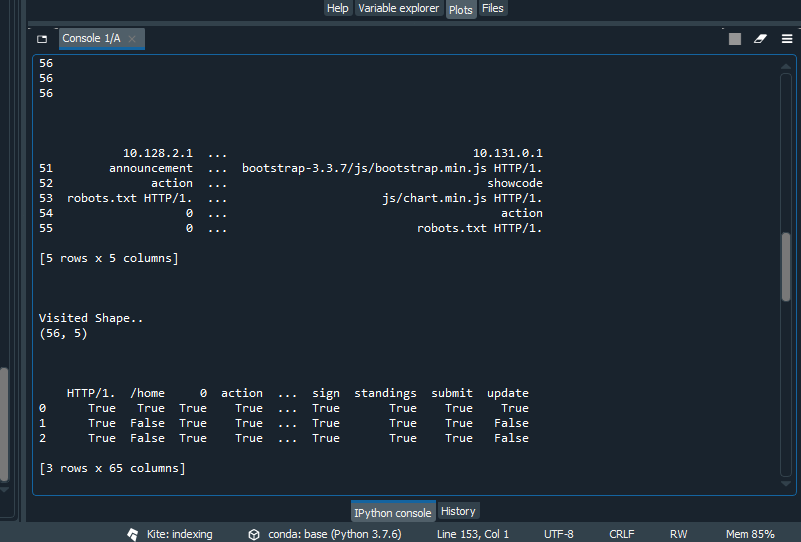
print(len(list(set(list\_deci))))

**OUTPUT:**

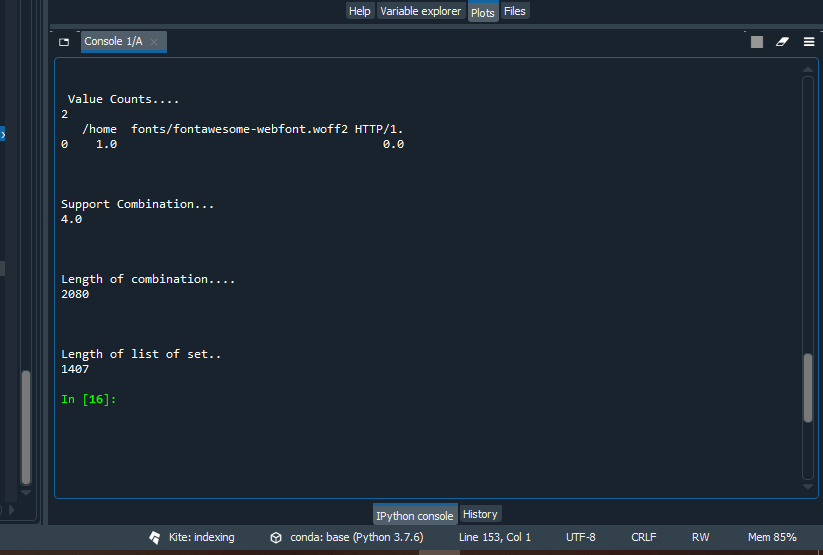




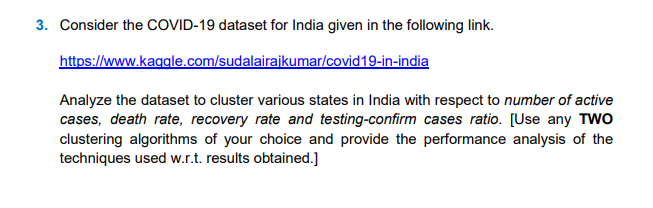








**Question-3:**



**Code:**