SOFT COMPUTING(CS3123)

Assignment – 1

1🡪 (A) REMOVAL OF DECISION ATTRIBUTE AND,

(B) DATASET IN MATRIX FORM

CODE :

import pandas as pd

with open('iris.data','r') as file:

  csv\_reader=pd.read\_csv(file)

df=pd.read\_csv('/content/iris.data')

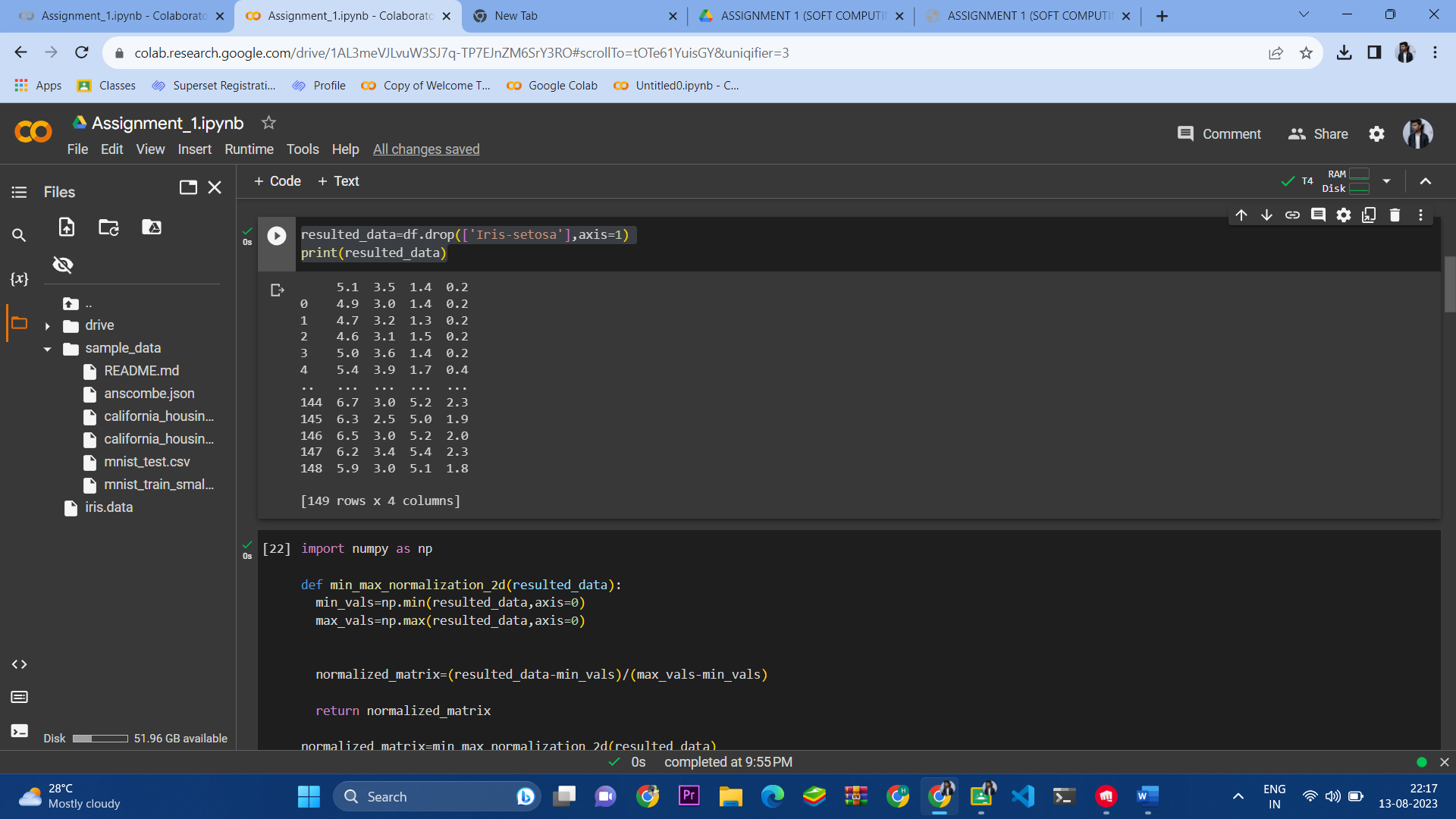
from google.colab import drive

drive.mount('/content/drive')

resulted\_data=df.drop(['Iris-setosa'],axis=1)

print(resulted\_data)

OUTPUT :



1🡪 (C) MIN-MAX NORMALIZATION

CODE :

import numpy as np

def min\_max\_normalization\_2d(resulted\_data):

  min\_vals=np.min(resulted\_data,axis=0)

  max\_vals=np.max(resulted\_data,axis=0)

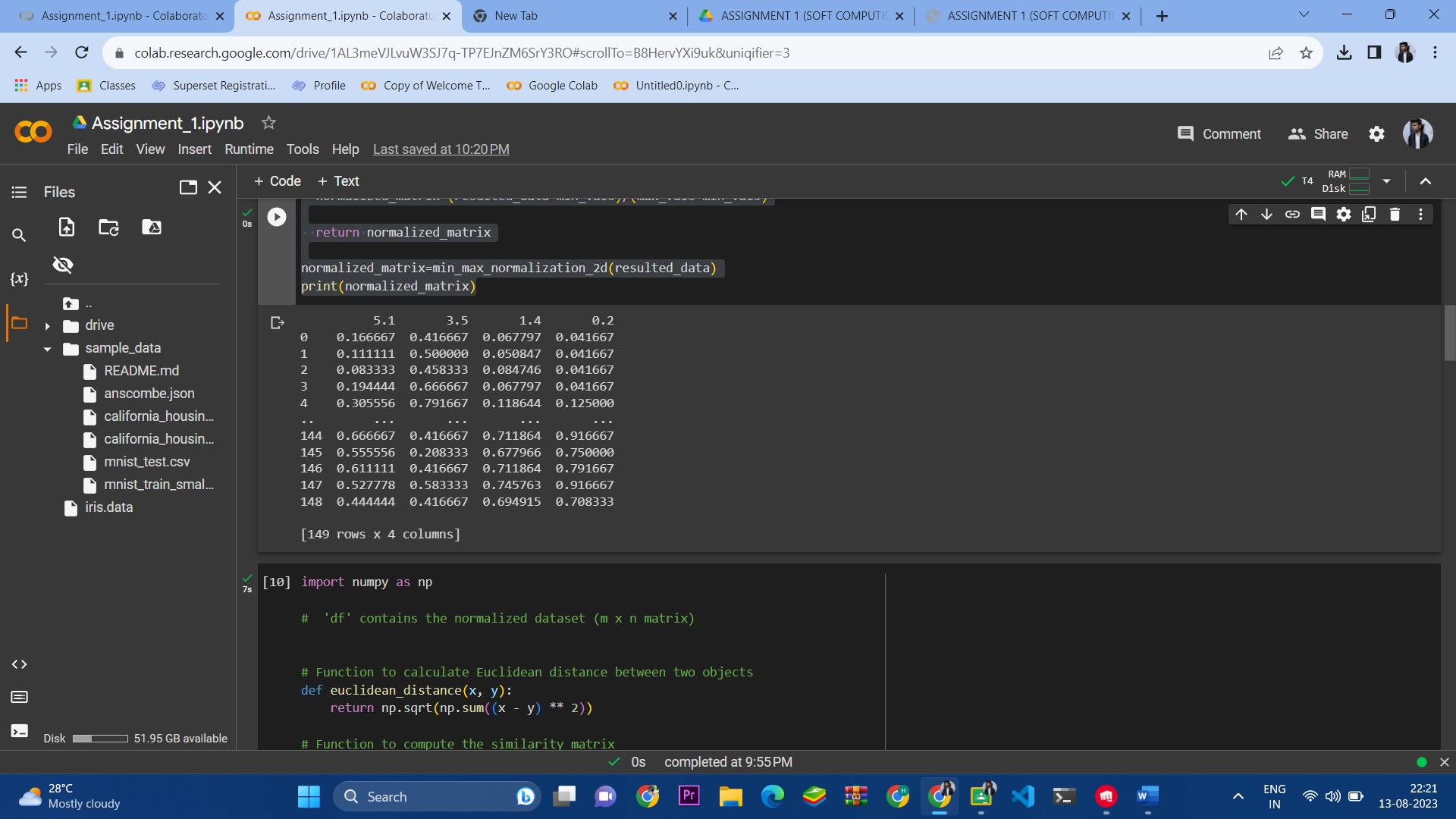
  normalized\_matrix=(resulted\_data-min\_vals)/(max\_vals-min\_vals)

  return normalized\_matrix

normalized\_matrix=min\_max\_normalization\_2d(resulted\_data)

print(normalized\_matrix)

OUTPUT :



2🡪 (A)SIMILARITY MATRIX

CODE :

import numpy as np

#  'df' contains the normalized dataset (m x n matrix)

# Function to calculate Euclidean distance between two objects

def euclidean\_distance(x, y):

    return np.sqrt(np.sum((x - y) \*\* 2))

# Function to compute the similarity matrix

def create\_similarity\_matrix(data):

    m, n = data.shape

    similarity\_matrix = np.zeros((m, m))

    for i in range(m):

        for j in range(m):

            similarity\_matrix[i, j] = euclidean\_distance(data.iloc[i], data.iloc[j])

    return similarity\_matrix

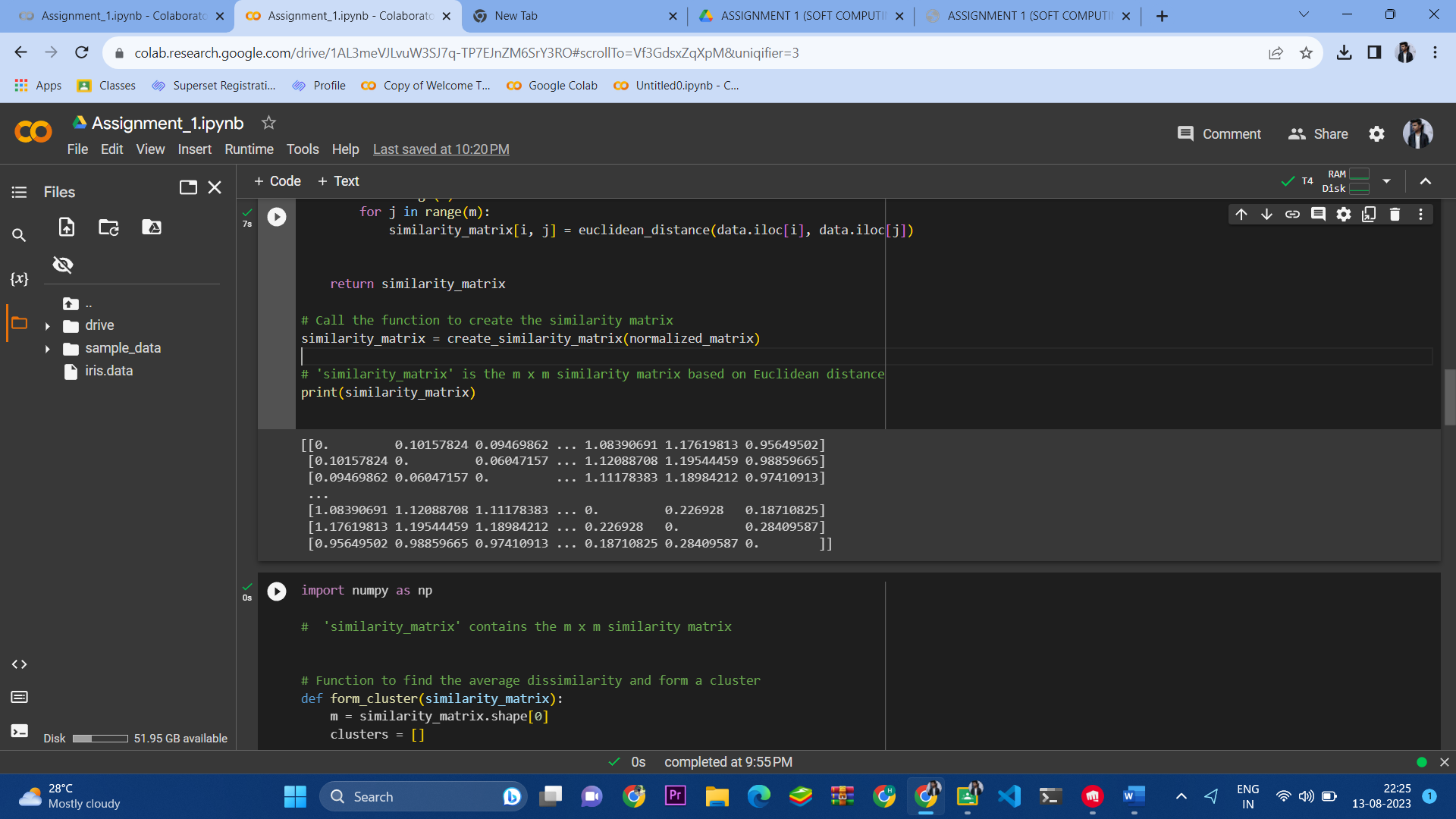
# Call the function to create the similarity matrix

similarity\_matrix = create\_similarity\_matrix(normalized\_matrix)

# 'similarity\_matrix' is the m x m similarity matrix based on Euclidean distance

print(similarity\_matrix)

OUTPUT :



2-> (B) AVG DISSIMILARITY AND CLUSTER FORMATION

CODE :

import numpy as np

#  'similarity\_matrix' contains the m x m similarity matrix

# Function to find the average dissimilarity and form a cluster

def form\_cluster(similarity\_matrix):

    m = similarity\_matrix.shape[0]

    clusters = []

    for i in range(m):

        # Calculate average dissimilarity of i-th object with others

        avg\_dissimilarity = np.sum(similarity\_matrix[i]) / (m - 1)

        # Form a cluster Ci with i-th object and objects having dissimilarity less than average

        cluster = [j for j in range(m) if j != i and similarity\_matrix[i, j] < avg\_dissimilarity]

        cluster.append(i)  # Include the i-th object in the cluster

        clusters.append(cluster)

    return clusters

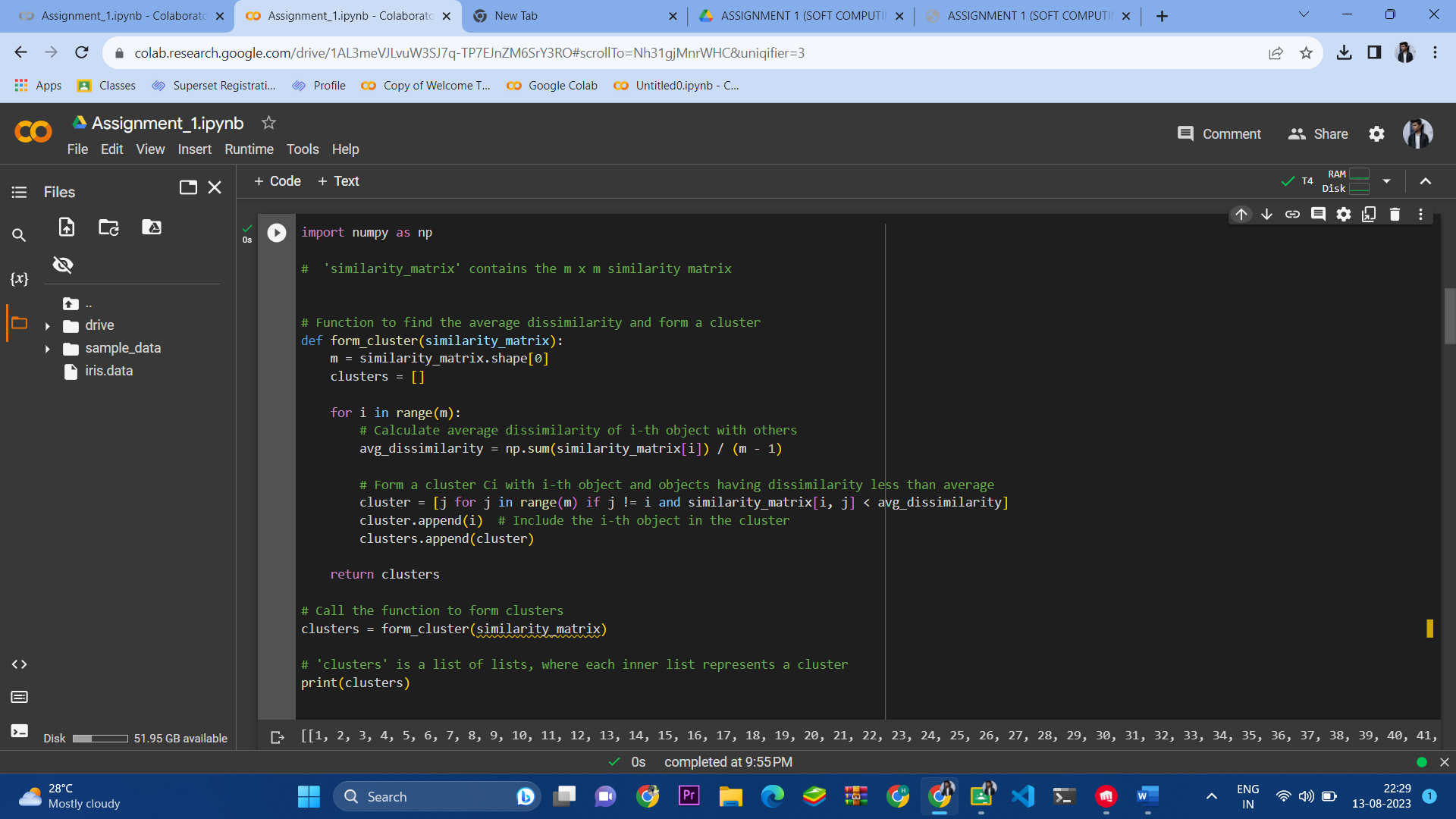
# Call the function to form clusters

clusters = form\_cluster(similarity\_matrix)

# 'clusters' is a list of lists, where each inner list represents a cluster

print(clusters)

OUTPUT :



Ques - 3->

import numpy as np

def min\_max\_normalization\_2d(resulted\_data):

  min\_vals=np.min(resulted\_data,axis=0)

  max\_vals=np.max(resulted\_data,axis=0)

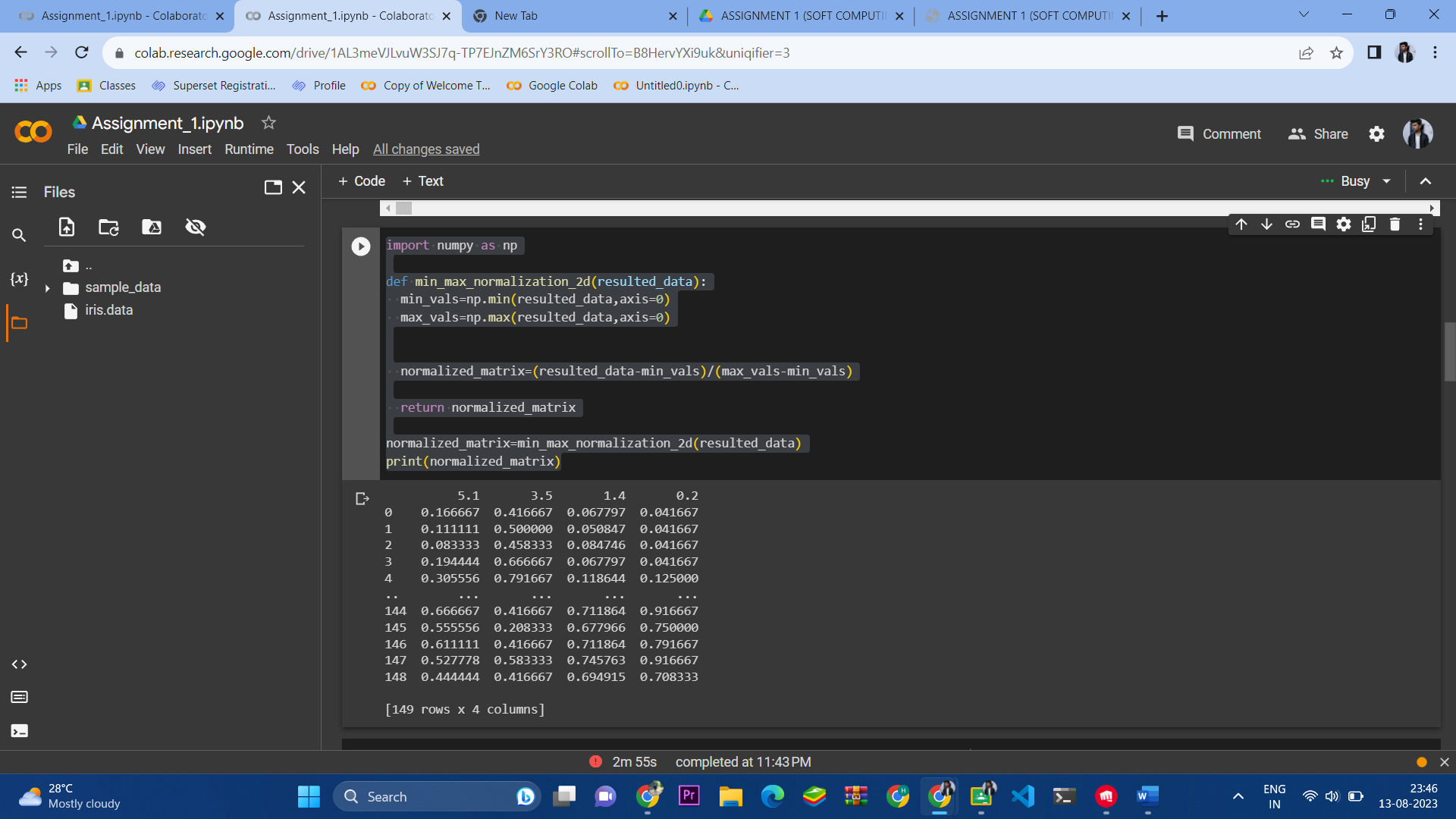
  normalized\_matrix=(resulted\_data-min\_vals)/(max\_vals-min\_vals)

  return normalized\_matrix

normalized\_matrix=min\_max\_normalization\_2d(resulted\_data)

print(normalized\_matrix)

OUTPUT :



import numpy as np

#  'df' contains the normalized dataset (m x n matrix)

# Function to calculate Euclidean distance between two objects

def euclidean\_distance(x, y):

    return np.sqrt(np.sum((x - y) \*\* 2))

# Function to compute the similarity matrix

def create\_similarity\_matrix(data):

    m, n = data.shape

    similarity\_matrix = np.zeros((m, m))

    for i in range(m):

        for j in range(m):

            similarity\_matrix[i, j] = euclidean\_distance(data.iloc[i], data.iloc[j])

    return similarity\_matrix

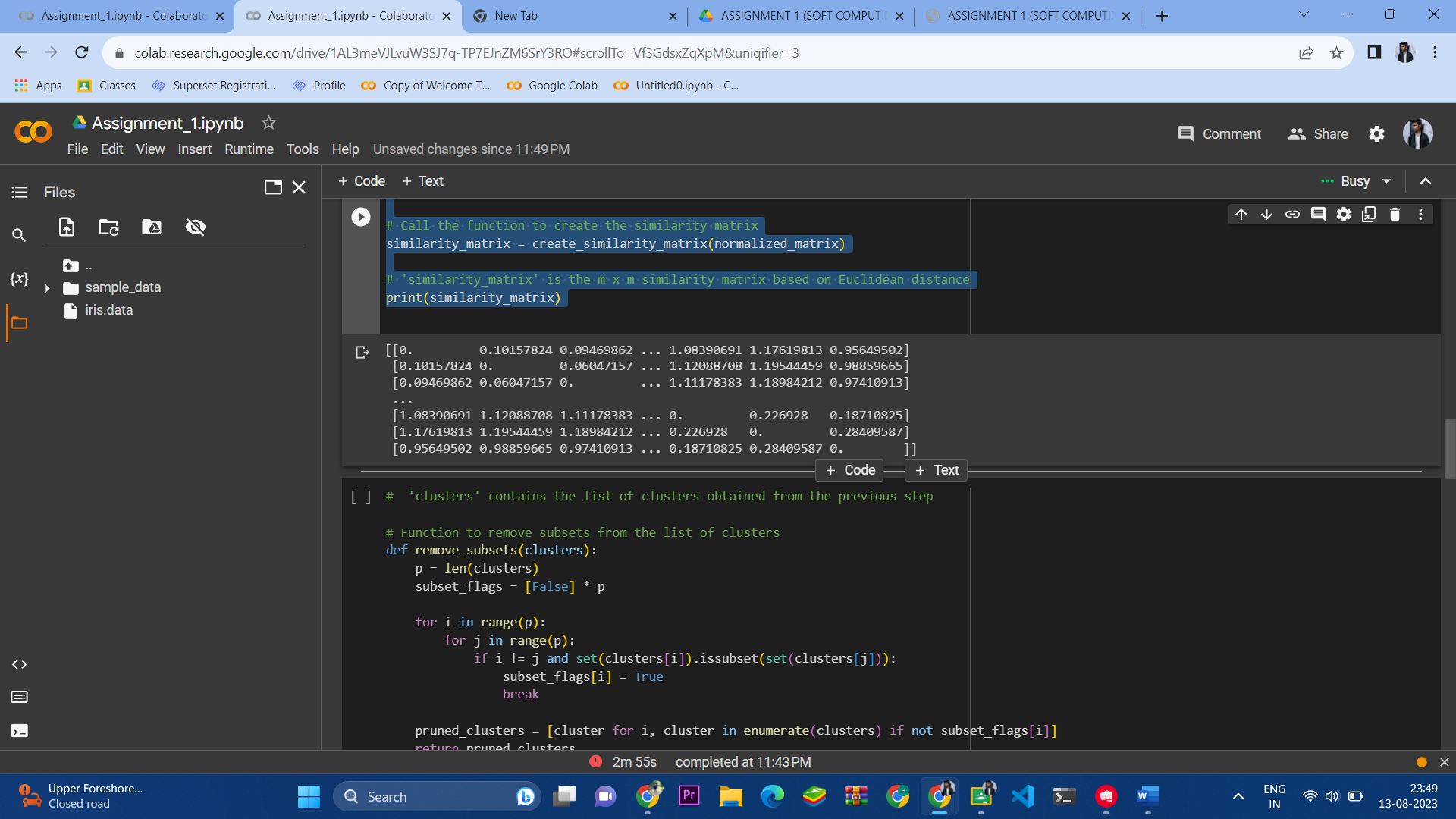
# Call the function to create the similarity matrix

similarity\_matrix = create\_similarity\_matrix(normalized\_matrix)

# 'similarity\_matrix' is the m x m similarity matrix based on Euclidean distance

print(similarity\_matrix)

OUTPUT :



3-> (A) SUBSET FREE CLUSTERS / SUBSET CLUSTERS REMOVAL

CODE :

#  'clusters' contains the list of clusters obtained from the previous step

# Function to remove subsets from the list of clusters

def remove\_subsets(clusters):

    p = len(clusters)

    subset\_flags = [False] \* p

    for i in range(p):

        for j in range(p):

            if i != j and set(clusters[i]).issubset(set(clusters[j])):

                subset\_flags[i] = True

                break

    pruned\_clusters = [cluster for i, cluster in enumerate(clusters) if not subset\_flags[i]]

    return pruned\_clusters

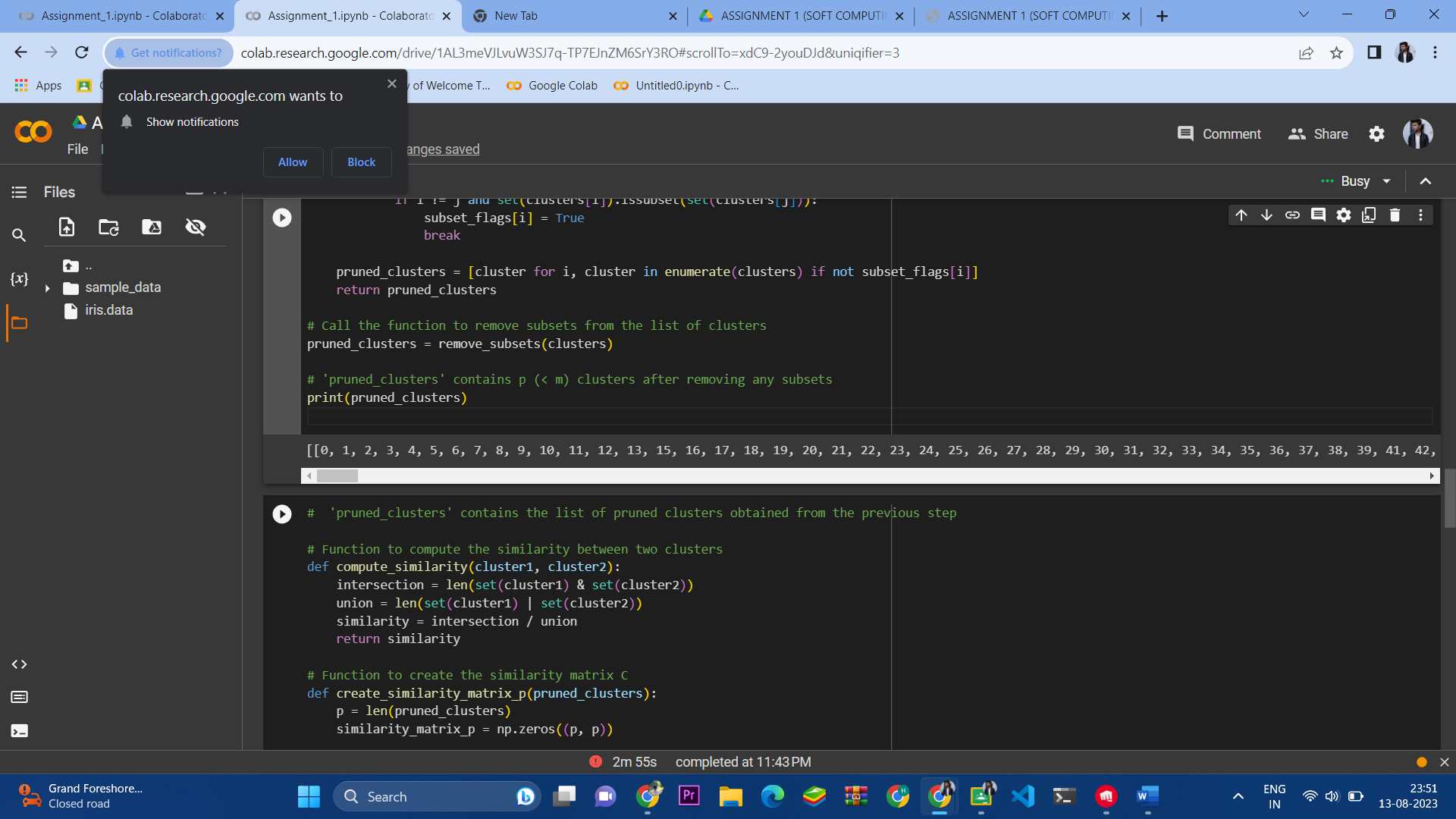
# Call the function to remove subsets from the list of clusters

pruned\_clusters = remove\_subsets(clusters)

# 'pruned\_clusters' contains p (< m) clusters after removing any subsets

print(pruned\_clusters)

OUTPUT :



3-> (B) SIMILARITY MATRIX OF SUBSET FREE CLUSTERS

CODE :

#  'pruned\_clusters' contains the list of pruned clusters obtained from the previous step

# Function to compute the similarity between two clusters

def compute\_similarity(cluster1, cluster2):

    intersection = len(set(cluster1) & set(cluster2))

    union = len(set(cluster1) | set(cluster2))

    similarity = intersection / union

    return similarity

# Function to create the similarity matrix C

def create\_similarity\_matrix\_p(pruned\_clusters):

    p = len(pruned\_clusters)

    similarity\_matrix\_p = np.zeros((p, p))

    for i in range(p):

        for j in range(p):

            similarity\_matrix\_p[i, j] = compute\_similarity(pruned\_clusters[i], pruned\_clusters[j])

    return similarity\_matrix\_p

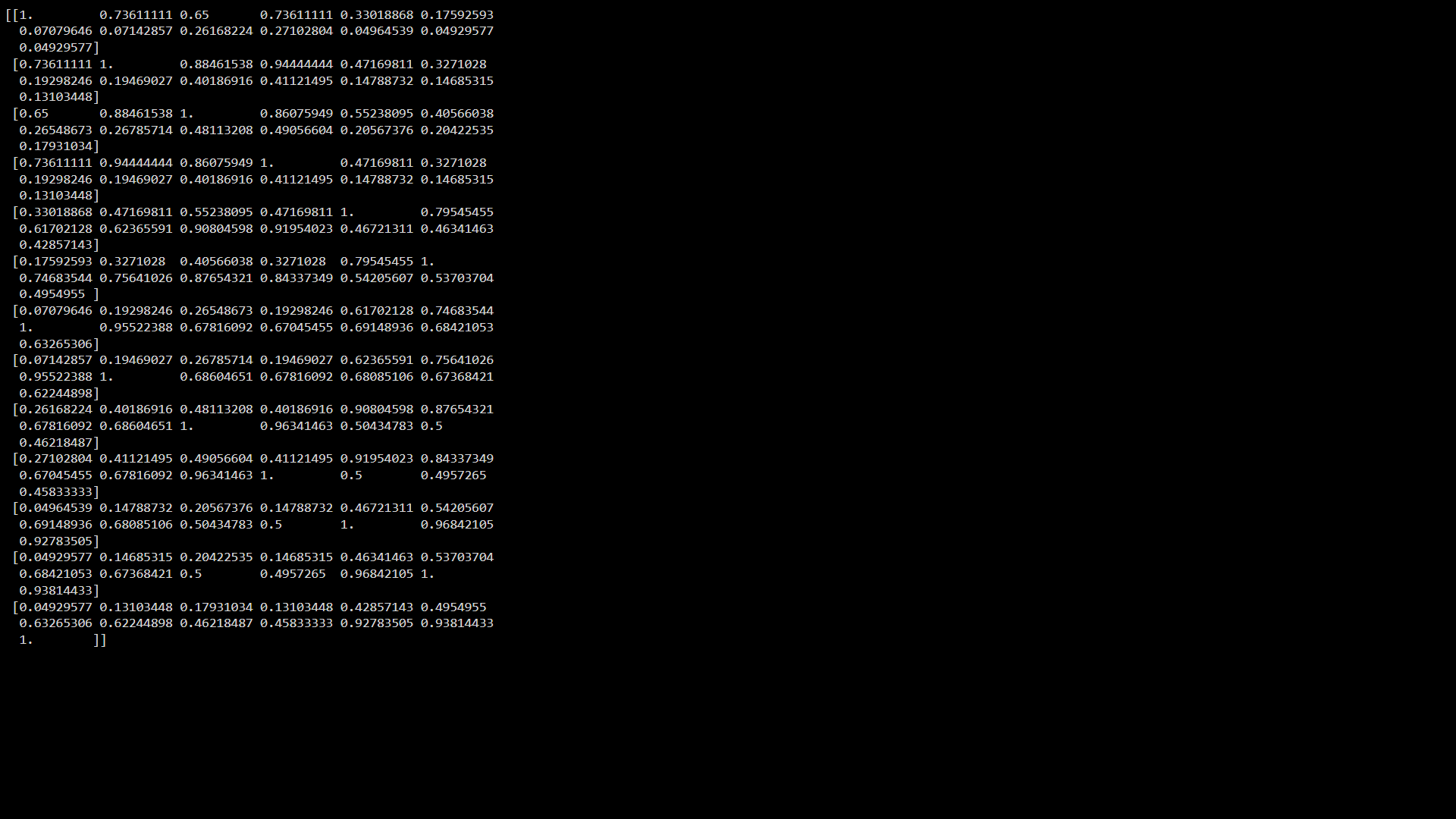
# Call the function to create the similarity matrix C

similarity\_matrix\_p = create\_similarity\_matrix\_p(pruned\_clusters)

# 'similarity\_matrix\_p' is the p x p similarity matrix between clusters

print(similarity\_matrix\_p)

OUTPUT :



3-> (C) MERGING TWO CLUSTERS TO FORM NEW CLUSTER AND ADD IT TO INITIAL CLUSTERS i.e No. of Clusters are Reduced

CODE :

# Function to find the most similar clusters and merge them

def merge\_most\_similar\_clusters(similarity\_matrix\_p, pruned\_clusters):

    p = len(pruned\_clusters)

    max\_similarity = -1.0

    most\_similar\_clusters = (None, None)

    # Find the most similar clusters Ck and Cl

    for k in range(p):

        for l in range(k + 1, p):  # To avoid checking pairs twice (symmetric matrix)

            similarity = similarity\_matrix\_p[k, l]

            if similarity > max\_similarity:

                max\_similarity = similarity

                most\_similar\_clusters = (k, l)

    # Get the indices of the most similar clusters

    k, l = most\_similar\_clusters

    # Merge the most similar clusters Ck and Cl to get a new cluster Ckl

    Ck = set(pruned\_clusters[k])

    Cl = set(pruned\_clusters[l])

    Ckl = list(Ck.union(Cl))

    # Remove the individual clusters Ck and Cl from the list and add the new cluster Ckl

    pruned\_clusters.pop(max(k, l))

    pruned\_clusters.pop(min(k, l))

    pruned\_clusters.append(Ckl)

    return pruned\_clusters

# Call the function to find the most similar clusters and merge them

merged\_clusters = merge\_most\_similar\_clusters(similarity\_matrix\_p, pruned\_clusters)

# 'merged\_clusters' contains the updated list of clusters after merging the most similar clusters

print(merged\_clusters)

OUTPUT :

