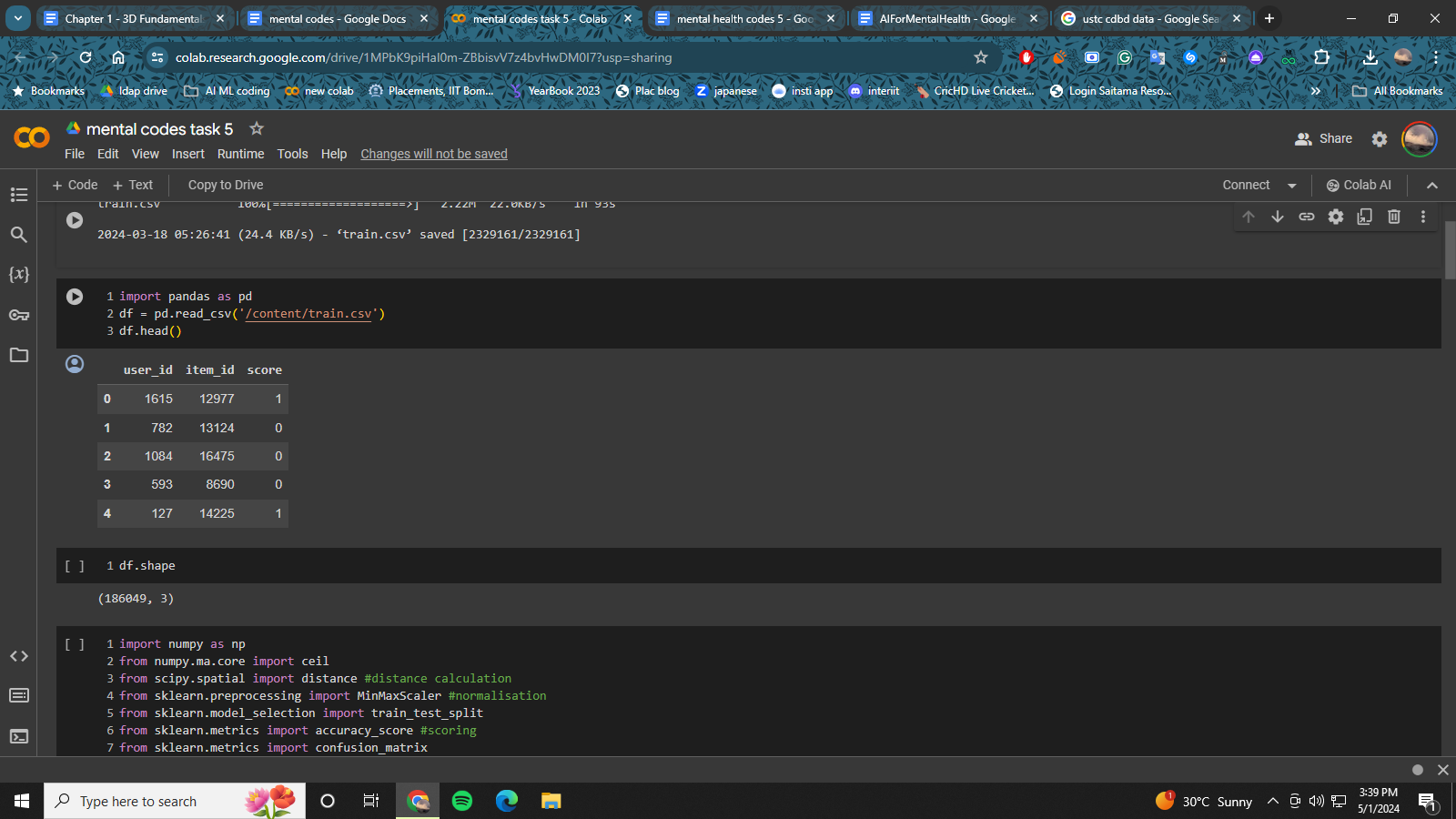
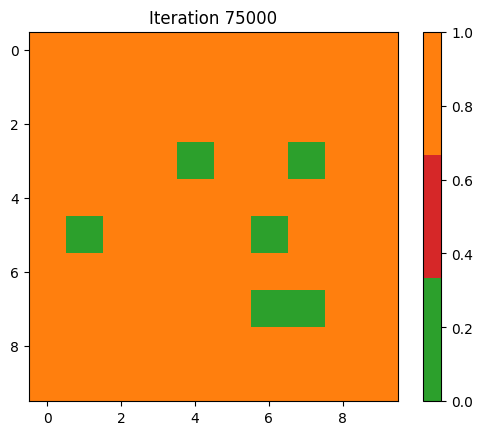
# Comprehensive Psychiatric Differential Diagnosis Test

In this part we take a psychiatric diagnosis dataset from the University of Science and Technology of China- [cdbd](https://base.ustc.edu.cn/data/cdbd/a0910/train.csv). This dataset consists of scores from 0 to 1 for all the study participants without disclosing their identity. It has more than 185k datapoints



First we begin by loading the dataset and splitting it into train and test dataset. We apply minmax normalisation on our dataset to aid the training. For the predictions and training we use the self-organising maps Though it is an unsupervised technique but like most of the artificial neural networks, it operates in two modes training and mapping. It produces a low dimensional mapping of a higher dimensional data. The algorithm provides a score between 0 to 1 for every datapoint.



## CODE:

## Installation

Pip install pandas sklearn matplotlib numpy

## Imports

Import pandas as pd

import numpy as np

from numpy.ma.core import ceil

from scipy.spatial import distance #distance calculation

from sklearn.preprocessing import MinMaxScaler #normalisation

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score #scoring

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

from matplotlib import animation, colors

## Data Loading

df = pd.read\_csv('train.csv')

## Data processing

train\_x, test\_x, train\_y, test\_y = train\_test\_split(df.iloc[:,:2].values, df.iloc[:,2].values, test\_size=0.2, random\_state=42)

def minmax\_scaler(data):

scaler = MinMaxScaler()

scaled = scaler.fit\_transform(data)

return scaled

train\_x\_norm = minmax\_scaler(train\_x) # normalisation

## Hyperparameters

num\_rows = 10

num\_cols = 10

max\_m\_dsitance = 4

max\_learning\_rate = 0.5

max\_steps = int(7.5\*10e3)

## SOM Building

# Euclidean distance

def e\_distance(x,y):

return distance.euclidean(x,y)

# Manhattan distance

def m\_distance(x,y):

return distance.cityblock(x,y)

# Best Matching Unit search

def winning\_neuron(data, t, som, num\_rows, num\_cols):

winner = [0,0]

shortest\_distance = np.sqrt(data.shape[1]) # initialise with max distance

input\_data = data[t]

for row in range(num\_rows):

for col in range(num\_cols):

distance = e\_distance(som[row][col], data[t])

if distance < shortest\_distance:

shortest\_distance = distance

winner = [row,col]

return winner

# Learning rate and neighbourhood range calculation

def decay(step, max\_steps,max\_learning\_rate,max\_m\_dsitance):

coefficient = 1.0 - (np.float64(step)/max\_steps)

learning\_rate = coefficient\*max\_learning\_rate

neighbourhood\_range = ceil(coefficient \* max\_m\_dsitance)

return learning\_rate, neighbourhood\_range

# initialising self-organising map

num\_dims = train\_x\_norm.shape[1] # numnber of dimensions in the input data

np.random.seed(40)

som = np.random.random\_sample(size=(num\_rows, num\_cols, num\_dims)) # map construction

# start training iterations

for step in range(max\_steps):

if (step+1) % 1000 == 0:

print("Iteration: ", step+1) # print out the current iteration for every 1k

learning\_rate, neighbourhood\_range = decay(step, max\_steps,max\_learning\_rate,max\_m\_dsitance)

t = np.random.randint(0,high=train\_x\_norm.shape[0]) # random index of traing data

winner = winning\_neuron(train\_x\_norm, t, som, num\_rows, num\_cols)

for row in range(num\_rows):

for col in range(num\_cols):

if m\_distance([row,col],winner) <= neighbourhood\_range:

som[row][col] += learning\_rate\*(train\_x\_norm[t]-som[row][col]) #update neighbour's weight

print("SOM training completed")

## Model Prediction

label\_data = train\_y

map = np.empty(shape=(num\_rows, num\_cols), dtype=object)

for row in range(num\_rows):

for col in range(num\_cols):

map[row][col] = [] # empty list to store the label

for t in range(train\_x\_norm.shape[0]):

if (t+1) % 1000 == 0:

print("sample data: ", t+1)

winner = winning\_neuron(train\_x\_norm, t, som, num\_rows, num\_cols)

map[winner[0]][winner[1]].append(label\_data[t]) # label of winning neuron

## Plotting Results

label\_map = np.zeros(shape=(num\_rows, num\_cols),dtype=np.int64)

for row in range(num\_rows):

for col in range(num\_cols):

label\_list = map[row][col]

if len(label\_list)==0:

label = 2

else:

label = max(label\_list, key=label\_list.count)

label\_map[row][col] = label

title = ('Iteration ' + str(max\_steps))

cmap = colors.ListedColormap(['tab:green', 'tab:red', 'tab:orange'])

plt.imshow(label\_map, cmap=cmap)

plt.colorbar()

plt.title(title)

plt.show()