**MACHINE LEARNING CODE EXPLANATION:**

**!pip install mplfinance** // installing the mplfinance module.

**import numpy as np**  // importing the numpy module.

**import pandas as pd** // importing panda module.

**import matplotlib.pyplot as plt** // importing matplotlib

**from matplotlib.dates import DateFormatter** //importing date formatter from matplotlib.

**pd.pandas.set\_option('display.max\_rows',None)** // setting the option to display the maximum number of rows.

**import seaborn as sns** // importing the seaborn module.

**from sklearn import preprocessing** // importing preprocessing from sklearn module.

**import mplfinance as mpf** // importing the mplfinance module.

**data = pd.read\_csv('dataset\_new.csv')** // connecting the dataset

**data.head()**  // showing the top 5 data of the datset.

**data['TIME'] = data['DATE'] + ' ' + data['TIME']** // dividing the time column into date & time format

data.drop(['DESCRIPTION','DATE'],axis=1,inplace=True) // dropping date & description column.

**import re**

**def clean\_time(text):**

**# 1. Make a space**

**review = re.sub('[^a-zA-Z0-9-.:]',' ', text)** // creating a lambda function.

**return (review)**

**data['TIME'] = data['TIME'].apply(lambda x: clean\_time(x))** // creating a space in time column by lambda function.

**data\_copy = data.copy()** // copying the data.

**data['TIME'] = pd.to\_datetime(data['TIME'])** // converting the time

**data['MINUTE'] = data['TIME'].dt.minute** // Dividing time into minute.

**data['HOUR'] = data['TIME'].dt.hour** // Dividing time into hour.

**data['SECOND'] = data['TIME'].dt.second** // Dividing time into second.

**data.shape** // showing the shape of the dataset.

**data['SPO2'] = data['SPO2'].astype('float64')** // converting the SPo2 datatype into float.

**data.info()**  // showing the all other columns information like datatype null data etc.

**plt.plot\_date(data['TIME'], data['BPM'], linestyle='solid')** //plotting graph

**plt.title('Heart Beat comparison w.r.t Time')**

**plt.xlabel('<----------------Time---------------->')** // along x axis time.

**plt.ylabel('<----------------Heart Beat---------------->')** // along y axis heart beat

**plt.plot\_date(data['TIME'], data['SPO2'], linestyle='solid')** // plotting the graph

**plt.title('SPO2 comparison w.r.t Time')**

**plt.xlabel('<----------------Time---------------->')**  // along x axis time

**plt.ylabel('<----------------SPO2---------------->')**  // along y axis

SPo2

**i=0 //**The code starts by creating a list called time.

**time = []**

**while(i<data\_length): //**The code then iterates through the data\_copy dictionary, which is a copy of the original data dictionary.

**a=data['MINUTE'][i]**

**j=i**

**count = 0**

**s = 0**

**mean = 0**

**while(j<data\_length):**

**if(a == data['MINUTE'][j]):**

**count = count + 1**

**j=j+1**

**else:**

**break**

**time.append(data\_copy['TIME'][j-1])**

**i=i+couut //**It starts at index 0 and goes to index length-1.

**i=0 //**It starts at index 0 and goes to index length-1.The code starts by declaring the variables hour and i.

**hour = []**

**while(i<data\_length):**

**a=data['MINUTE'][i]**

**j=i**

**count = 0**

**s = 0**

**mean = 0**

**while(j<data\_length):**

**if(a == data['MINUTE'][j]): //**The code then goes on to loop through each of the data items in the list, which are stored in a variable called data.

**count = count + 1**

**j=j+1**

**else:**

**break**

**hour.append(data['HOUR'][j-1])**

**i=i+count**

**i=0**

**minute = []**

**while(i<data\_length): //**The code starts by creating an empty list called minute.

**a=data['MINUTE'][i]**

**j=i**

**count = 0 //**The code then loops through the data, and for each iteration of the loop, it adds a new element to the list minute.

**s = 0**

**mean = 0**

**while(j<data\_length):**

**if(a == data['MINUTE'][j]):**

**count = count + 1 //**The first line creates an empty list called minute**.**

**j=j+1**

**else:**

**break**

**minute.append(data['MINUTE'][j-1])**

**i=i+count**

**i=0 //**The code starts by declaring a variable called data.

second = []

**while(i<data\_length):**

**a=data['MINUTE'][i] //**It is initialized with an empty list, [] The code then iterates through the list of minutes and creates a new list for each minute in the loop.

**j=i**

**count = 0**

**s = 0**

**mean = 0**

**while(j<data\_length):**

**if(a == data['MINUTE'][j]): //**The first element in this new list is assigned to be MINUTE[0], which will contain the time 0:00.

**count = count + 1**

**j=j+1**

**else:**

**break**

**second.append(data['SECOND'][j-1])**

**i=i+count**

**print(len(time))**

**print(len(hour)) //**The code prints the length of each time component in seconds.

**print(len(minute))**

**print(len(second))**

**def mean(data,feature): //**The code is a function that takes in data and feature.

**i=0**

**spo2\_mean=[]**

**while(i<data\_length):**

**a=data['MINUTE'][i] //**The code starts by setting the variable i to 0, then loops through each of the values in data.

**j=i**

**count = 0**

**s = 0**

**mean = 0 //**It sets s to 0, then loops through each of the values in feature.

**while(j<data\_length):**

**if(a == data['MINUTE'][j]):**

**s = data[feature][j] + s**

**count = count + 1**

**#spo2\_mean = (s/count).astype(int)**

**#spo2\_mean.append(s/count)**

**j=j+1**

**else:**

**break**

**mean = round(s/count,2) //**Then it calculates mean using round(s/count), which is added into spo2\_mean as an element at index j=i+1.

**spo2\_mean.append(mean) //**The code will take the mean of all the features in a minute and put it into a list.

**i=i+count**

**return(spo2\_mean)**

**data\_new = pd.DataFrame()** //This is a DataFrame, which is an object that can hold multiple rows of data.

The code then creates the first row in this DataFrame with Time as the column name and hour as the value for each row.

Next, minute becomes Minute, second becomes Second.

Next, we create another variable called time and assign it to be equal to 0:00:00 (midnight).

Then we create another variable called hour and assign it to be equal to 12:00am (noon).

**data\_new =pd.DataFrame({'Time' : time,**

**'Hour' : hour,**

**'Minute' : minute,**

**'Second' : second })**

**data\_new.head()**

**for feature in ['BPM','SPO2']: //**The code starts by creating a variable called xy.

//This is the mean of data, which is a DataFrame object.

**xy = mean(data,feature)**

**mn = pd.DataFrame(xy)**

**data\_new = pd.concat([data\_new,mn],axis=1) //**The code then creates another variable called mn and assigns it to be the concatenation of data\_new and mn, which are both DataFrames objects.

**data\_new.columns = ['Time','Hour','Minute','Second','BPM\_mean','SPO2\_mean']**

**data\_new.head() //**The code then goes on to create new columns in data\_new that correspond with those in data\_old: Time, Hour, Minute, Second, BPM\_mean (the mean for BPM), SPO2\_mean (the mean for SPO2).

**data\_new.loc[(data\_new['BPM\_mean'] == 0.00), 'Condition'] = 'Dead'**

**data\_new.loc[((data\_new['BPM\_mean'] < 60)|(data\_new['BPM\_mean'] > 100)) |**

**((data\_new['SPO2\_mean'] < 94)|(data\_new['SPO2\_mean'] > 100)), 'Condition'] = 'Critical'**

**data\_new.loc[((data\_new['BPM\_mean'] >= 60)&(data\_new['BPM\_mean'] <= 100)) &**

**((data\_new['SPO2\_mean'] >= 94)&(data\_new['SPO2\_mean'] <= 99)), 'Condition'] = 'Normal'**

**data\_new.head()** //The code then creates two columns in the data frame, BPM and SPO2.

Next, it sets up a for loop that will iterate through each row of the original dataset (data) and create a new column with the condition of either 'Dead' or 'Critical'.

Then it uses this new column to set up another for loop that will iterate through each row again but this time setting the condition to be either 'Normal' or 'Dead'.

Finally, it prints out all rows from both loops.

**data\_new = pd.read\_csv('C:/Users/SOUVIK/Downloads/PPG\_work.csv')**

**data\_new.drop('Unnamed: 0',axis=1,inplace=True)**

**data\_new['Time'] = pd.to\_datetime(data\_new['Time'])**

**sns.countplot(x = data\_new['Condition']) //The code displays a countplot of the number of times each condition was used in the data set.**

**plt.title('Count of number of Target Feature')**

**data\_new['Condition'].value\_counts().plot(kind='pie',autopct='%.2f')**

**plt.title('Percentge share of Target Feature') //** The code first sets up a variable called data\_new, which is an empty list.Then it iterates through all the conditions and adds them to data\_new as key-value pairs where key is 'Condition' and value is either True or False.

**sns.scatterplot(x = data\_new['Time'], y = data\_new['BPM\_mean'],hue = data\_new['Condition'])** // The code creates a scatterplot of the data in the variable 'Time' and the average BPM for each condition.

**sns.scatterplot(x = data\_new['Time'], y = data\_new['SPO2\_mean'],hue = data\_new['Condition']) //** The code creates a scatterplot of the data set, with the x-axis representing time and the y-axis representing SPO2.

**le=preprocessing.LabelEncoder()**

**le.fit(['Critical','Normal']) //** This is an object that will be used to encode the labels of the dataframe into integers.

**y=pd.DataFrame()**

**data\_new['Condition'] = le.transform(data\_new['Condition'])** // The fit method then takes in two strings, "Critical" and "Normal", and encodes them as integer values 0 and 1 respectively.

**data\_new.head()**

**corr = data\_new.corr()**

**plt.figure(figsize=(10,7)) //** The code does the following: Creates a new dataframe with the same columns as data\_new but with a new column called corr.

**sns.heatmap(corr, annot=True, cmap='coolwarm')**

**data\_new.info()** // Shows the information and datatypes of the dataframe.

**from sklearn.preprocessing import LabelEncoder**

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

**from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report** // importing the necessary model for processing the code.

**x = data\_new[['Hour','Minute','Second','BPM\_mean','SPO2\_mean']]**

// The code will take the data from the file, and create a list of tuples.

**y = data\_new['Condition']**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.20, random\_state = 42)**

**print('Tota train rows {}'.format(len(x\_train))) //** The variable x is a list of data points that are being split into train and test sets.

**print('Tota test rows {}'.format(len(x\_test))) //** The variable y is the target value for each point in the dataset.

**def training\_model():**

**from sklearn.tree import DecisionTreeClassifier //** The code starts by importing the DecisionTreeClassifier class from sklearn.tree, which is a Python module that implements decision tree algorithms.

**classifier = DecisionTreeClassifier(random\_state=42)**

**classifier.fit(x\_train, y\_train) //** The code then calls the fit method on the classifier object, passing in x\_train and y\_train as input arguments.

**return classifier**

**model = training\_model() //** The code creates a new model object and then initializes it with the training\_model**.**

**y\_pred\_train = model.predict(x\_train)** //Visualizing the performace of the model

**print(classification\_report(y\_train, y\_pred\_train,**

**target\_names = ['Critical','Normal']))**

**//** printing the classification report**.**

**cm = confusion\_matrix(y\_train , y\_pred\_train)**

**print(cm)** // printing the confusion matrix.

**acc\_score = accuracy\_score(y\_pred\_train , y\_train)**

**acc\_score = acc\_score\*100**

**print('\nAccuracy of train set {}%'.format(acc\_score))** // printing the accuracy score.

**y\_pred\_test = model.predict(x\_test)**

// Visualizing the performace of the model

**print(classification\_report(y\_test, y\_pred\_test,**

**target\_names = ['Critical','Normal']))**

**//** printing the classification report**.**

**cm = confusion\_matrix(y\_test , y\_pred\_test)**

**print(cm)** // printing the confusion matrix.

**acc\_score = accuracy\_score(y\_pred\_test , y\_test)**

**acc\_score = acc\_score\*100**

**print('\nAccuracy of test set {}%'.format(acc\_score))** // printing the accuracy score.

**def check\_data():**

**hour = int(input('Enter the current hour : '))**

**minute = int(input('Enter the current minute : '))**

**second = int(input('Enter the current second : '))**

**bpm\_mean = float(input('What is your current BPM : '))**

**spo2\_mean = float(input('What is your current SPO2 : '))**

**d = [[hour,minute,second,bpm\_mean,spo2\_mean]]** // The code is trying to predict the health condition of a patient based on their current BPM, SPO2, and hour.

**pd.DataFrame(d)**

**prediction = model.predict(d)**

**normal = 'Your health condition is normal'** // The code first creates an empty list called d with three elements: hour, minute, second.

    critical = 'Your health condition is critical'

**if prediction == 0:**

**return critical** // Finally it prints out whether or not your health is normal or critical depending on what prediction returned by model.predict()

**elif prediction == 1:**

**return normal**

**check\_data() //** to check the data provided by thepatient and predict their health condition.