```
import tensorflow as tf
from tensorflow.keras.optimizers import Adam

def solution_C2():
    mnist = tf.keras.datasets.mnist
    (train_x,train_y), (test_x, test_y) = mnist.load_data()

    train_x = train_x/255
    test_x  = test_x/255

    model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape = [28,28,1]),
        tf.keras.layers.Dense(128, activation = 'relu'),
        tf.keras.layers.Dense(64, activation = 'relu'),
        tf.keras.layers.Dense(10, activation = 'softmax')
    ])

    model.compile(loss= 'sparse_categorical_crossentropy', optimizer = Adam(learning_rate = 0.001), metrics = ['accuracy'])

    model.fit(train_x,train_y, steps_per_epoch = 8, epochs = 50, validation_data = (test_x,test_y), validation_steps = 8)

    return model



if __name__ == '__main__':
    model = solution_C2()
    model.save("model_C2.h5")
```

```
Epoch 22/50
8/8 [==============================] - 1s 119ms/step - loss: 0.1385 - accuracy: 0.9608 - val_loss: 0.1442 - val_accuracy: 0.9583
Epoch 23/50
8/8 [==============================] - 1s 118ms/step - loss: 0.1330 - accuracy: 0.9622 - val_loss: 0.1401 - val_accuracy: 0.9595
Epoch 24/50
8/8 [==============================] - 1s 125ms/step - loss: 0.1287 - accuracy: 0.9635 - val_loss: 0.1368 - val_accuracy: 0.9604
Epoch 25/50
8/8 [==============================] - 1s 125ms/step - loss: 0.1237 - accuracy: 0.9650 - val_loss: 0.1328 - val_accuracy: 0.9612
Epoch 26/50
8/8 [==============================] - 1s 111ms/step - loss: 0.1191 - accuracy: 0.9666 - val_loss: 0.1305 - val_accuracy: 0.9615
Epoch 27/50
8/8 [==============================] - 1s 112ms/step - loss: 0.1148 - accuracy: 0.9678 - val_loss: 0.1268 - val_accuracy: 0.9615
Epoch 28/50
```

Colab paid products - Cancel contracts here

✓ 1m 25s completed at 4:21 PM

```
import tensorflow as tf
import urllib.request
import zipfile
import tensorflow as tf
import os
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator


def solution_C3():
    data_url = 'https://github.com/dicodingacademy/assets/raw/main/Simulation/machine_learning/cats_and_dogs.zip'
    urllib.request.urlretrieve(data_url, 'cats_and_dogs.zip')
    local_file = 'cats_and_dogs.zip'
    zip_ref = zipfile.ZipFile(local_file, 'r')
    zip_ref.extractall('data/')
    zip_ref.close()

    BASE_DIR = 'data/cats_and_dogs_filtered'
    train_dir = os.path.join(BASE_DIR, 'train')
    validation_dir = os.path.join(BASE_DIR, 'validation')

    train_datagen =  ImageDataGenerator(
        rescale = 1/255,
        rotation_range = 40,
        width_shift_range = 0.2,
        height_shift_range = 0.2,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True,
        fill_mode= 'nearest'
    )
    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150,150),
        batch_size=128,
        class_mode='binary'
    )
    validation_datagen = ImageDataGenerator(
        rescale = 1/255
    )
    validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=(150,150),
        batch_size=32,
        class_mode='binary'
    )


    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32,(3,3), activation = 'relu', input_shape = [150,150,3]),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation = 'relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])
    model.compile(loss = 'binary_crossentropy', optimizer= RMSprop(learning_rate = 0.001), metrics = ['acc'])
    model.fit(
        train_generator,
        epochs=50,
        validation_data=validation_generator,
        verbose=2
    )

    return model



if __name__ == '__main__':
    model = solution_C3()
    model.save("model_c3.h5")
```

17m 28s completed at 4:42 PM

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import SGD

def solution_C1():
    X = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0], dtype=float)
    Y = np.array([0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5], dtype=float)

    model = tf.keras.Sequential([
        tf.keras.layers.Dense(1, input_shape = [1,])
    ])
    model.compile(loss = 'mean_squared_error', optimizer = SGD(), metrics = ['mse'])
    model.fit(X,Y, epochs =100, batch_size = 1)

    print(model.predict([-2.0, 10.0]))
    return model


if __name__ == '__main__':
    model = solution_C1()
    model.save("model_C1.h5")
```

```
7/7 [==============================] - 0s 2ms/step - loss: 1.1535e-05 - mse: 1.1535e-05
Epoch 74/100
7/7 [==============================] - 0s 2ms/step - loss: 9.8801e-06 - mse: 9.8801e-06
Epoch 75/100
7/7 [==============================] - 0s 2ms/step - loss: 8.7366e-06 - mse: 8.7366e-06
Epoch 76/100
7/7 [==============================] - 0s 2ms/step - loss: 7.7131e-06 - mse: 7.7131e-06
Epoch 77/100
7/7 [==============================] - 0s 2ms/step - loss: 6.7360e-06 - mse: 6.7360e-06
Epoch 78/100
7/7 [==============================] - 0s 2ms/step - loss: 5.8174e-06 - mse: 5.8174e-06
Epoch 79/100
7/7 [==============================] - 0s 2ms/step - loss: 5.1598e-06 - mse: 5.1598e-06
Epoch 80/100
7/7 [==============================] - 0s 2ms/step - loss: 4.4886e-06 - mse: 4.4886e-06
Epoch 81/100
7/7 [==============================] - 0s 2ms/step - loss: 3.9308e-06 - mse: 3.9308e-06
Epoch 82/100
7/7 [==============================] - 0s 2ms/step - loss: 3.4089e-06 - mse: 3.4089e-06
Epoch 83/100
7/7 [==============================] - 0s 2ms/step - loss: 2.9975e-06 - mse: 2.9975e-06
Epoch 84/100
7/7 [==============================] - 0s 2ms/step - loss: 2.6163e-06 - mse: 2.6163e-06
Epoch 85/100
7/7 [==============================] - 0s 2ms/step - loss: 2.3104e-06 - mse: 2.3104e-06
Epoch 86/100
7/7 [==============================] - 0s 2ms/step - loss: 1.9986e-06 - mse: 1.9986e-06
Epoch 87/100
7/7 [==============================] - 0s 2ms/step - loss: 1.7436e-06 - mse: 1.7436e-06
Epoch 88/100
7/7 [==============================] - 0s 2ms/step - loss: 1.5281e-06 - mse: 1.5281e-06
Epoch 89/100
7/7 [==============================] - 0s 2ms/step - loss: 1.3110e-06 - mse: 1.3110e-06
Epoch 90/100
7/7 [==============================] - 0s 2ms/step - loss: 1.1734e-06 - mse: 1.1734e-06
Epoch 91/100
7/7 [==============================] - 0s 2ms/step - loss: 1.0169e-06 - mse: 1.0169e-06
Epoch 92/100
7/7 [==============================] - 0s 2ms/step - loss: 8.8779e-07 - mse: 8.8779e-07
Epoch 93/100
7/7 [==============================] - 0s 2ms/step - loss: 7.8106e-07 - mse: 7.8106e-07
Epoch 94/100
7/7 [==============================] - 0s 2ms/step - loss: 6.7860e-07 - mse: 6.7860e-07
Epoch 95/100
7/7 [==============================] - 0s 2ms/step - loss: 6.0169e-07 - mse: 6.0169e-07
Epoch 96/100
7/7 [==============================] - 0s 2ms/step - loss: 5.2255e-07 - mse: 5.2255e-07
Epoch 97/100
7/7 [==============================] - 0s 2ms/step - loss: 4.5968e-07 - mse: 4.5968e-07
Epoch 98/100
7/7 [==============================] - 0s 2ms/step - loss: 3.9354e-07 - mse: 3.9354e-07
Epoch 99/100
7/7 [==============================] - 0s 2ms/step - loss: 3.4958e-07 - mse: 3.4958e-07
Epoch 100/100
7/7 [==============================] - 0s 2ms/step - loss: 3.0565e-07 - mse: 3.0565e-07
1/1 [==============================] - 0s 93ms/step
[[-1.0936856e-03]
 [ 6.0010428e+00]]
```

✓ 6s completed at 4:18 PM

```
import json
import tensorflow as tf
import numpy as np
import urllib
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import RMSprop

def solution_C4():
    data_url = 'https://github.com/dicodingacademy/assets/raw/main/Simulation/machine_learning/sarcasm.json'
    urllib.request.urlretrieve(data_url, 'sarcasm.json')
    vocab_size = 1000
    embedding_dim = 16
    max_length = 120
    trunc_type = 'post'
    padding_type = 'post'
    oov_tok = "<OOV>"
    training_size = 20000

    sentences = []
    labels = []
    urls = []
    with open('sarcasm.json','r') as f:
        datastore = json.load(f)
    for item in datastore:
        sentences.append(item['headline'])
        labels.append(item['is_sarcastic'])
        urls.append(item['article_link'])
    train_sentences = sentences[:training_size]
    train_labels = np.array(labels[:training_size])
    test_sentences = sentences[training_size:]
    test_labels = np.array(labels[training_size:])
    tokenizer = Tokenizer(num_words=vocab_size,
                          oov_token=oov_tok)
    tokenizer.fit_on_texts(train_sentences)

    training_seq = tokenizer.texts_to_sequences(train_sentences)
    train_sentences_pad = pad_sequences(training_seq,
                                    maxlen=max_length,
                                    truncating=trunc_type,
                                    padding=padding_type)

    testing_seq = tokenizer.texts_to_sequences(test_sentences)
    test_sentences_pad = pad_sequences(testing_seq,
                                   maxlen=max_length,
                                   truncating=trunc_type,
                                   padding=padding_type)

    model = tf.keras.Sequential([
        tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
        tf.keras.layers.GlobalAveragePooling1D(),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=RMSprop(),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    class myCallback(tf.keras.callbacks.Callback):
        def on_epoch_end(self, epoch, logs={}):
            if (logs.get('accuracy') > 0.8 and logs.get('val_accuracy') > 0.8):
                print("\n\n\t====================================")
                print("\t|| accuracy and val_accuracy > 80% ||")
                print("\t====================================\n")
                self.model.stop_training = True
    callbacks = myCallback()

    model.fit(train_sentences_pad,
                train_labels,
                epochs=1000,
                validation_data=(test_sentences_pad, test_labels),
                callbacks=[callbacks])
    return model


if __name__ == '__main__':
    model = solution_C4()
    model.save("model_c4.h5")
```

```
 Epoch 1/1000
 625/625 [==============================] - 3s 3ms/step - loss: 0.6853 - accuracy: 0.5602 - val_loss: 0.6827 - val_accuracy: 0.5633
```

```
Epoch 2/1000
625/625 [==============================] - 2s 3ms/step - loss: 0.6787 - accuracy: 0.5618 - val_loss: 0.6678 - val_accuracy: 0.5733
Epoch 3/1000
625/625 [==============================] - 3s 4ms/step - loss: 0.6402 - accuracy: 0.6215 - val_loss: 0.6020 - val_accuracy: 0.6701
Epoch 4/1000
625/625 [==============================] - 2s 3ms/step - loss: 0.5570 - accuracy: 0.7172 - val_loss: 0.5252 - val_accuracy: 0.7554
Epoch 5/1000
625/625 [==============================] - 2s 3ms/step - loss: 0.4855 - accuracy: 0.7693 - val_loss: 0.4784 - val_accuracy: 0.7703
Epoch 6/1000
625/625 [==============================] - 2s 3ms/step - loss: 0.4471 - accuracy: 0.7943 - val_loss: 0.4569 - val_accuracy: 0.7809
Epoch 7/1000
625/625 [==============================] - 2s 3ms/step - loss: 0.4244 - accuracy: 0.8055 - val_loss: 0.4347 - val_accuracy: 0.7976
Epoch 8/1000
615/625 [============================>.] - ETA: 0s - loss: 0.4082 - accuracy: 0.8118

        ====================================
        || accuracy and val_accuracy > 80% ||
        ====================================

625/625 [==============================] - 2s 3ms/step - loss: 0.4086 - accuracy: 0.8117 - val_loss: 0.4212 - val_accuracy: 0.8074
```

```
import tensorflow as tf
import numpy as np
#import matplotlib.pyplot as plt
import csv
import urllib
from tensorflow.keras.losses import Huber
from tensorflow.keras.optimizers import SGD


def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch(1)


def solution_C5():
    data_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv'
    urllib.request.urlretrieve(data_url, 'daily-min-temperatures.csv')

    time_step = []
    temps = []

    with open('daily-min-temperatures.csv') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        next(reader)
        step = 0
        for row in reader:
            temps.append(float(row[1]))
            time_step.append(row[0])
            step=step + 1

    series=  np.array(temps)


    min=np.min(series)
    max=np.max(series)
    series -= min
    series /= max
    time=np.array(time_step)


    split_time=2500

    time_train=  time[:split_time]
    x_train=  series[:split_time]
    time_valid=  time[split_time:]
    x_valid=  series[split_time:]


    window_size=64
    batch_size=256
    shuffle_buffer_size=1000

    train_set=windowed_dataset(
        x_train, window_size, batch_size, shuffle_buffer_size)
    print(train_set)
    print(x_train.shape)

    model=tf.keras.models.Sequential([
        tf.keras.layers.Dense(30, input_shape = [None, 1], activation = 'relu'),
        tf.keras.layers.Dense(10,activation = 'relu'),
        tf.keras.layers.Dense(1)
    ])
    model.compile(loss= Huber(),
                  optimizer = SGD(momentum = 0.9),
                  metrics=['mae'])
    model.fit(train_set,epochs = 100)


    return model


if __name__ == '__main__':
    model = solution_C5()
    model.save("model_c5.h5")
```

```
10/10 [==============================] - 1s 36ms/step - loss: 0.0050 - mae: 0.0777
Epoch 73/100
10/10 [==============================] - 1s 37ms/step - loss: 0.0050 - mae: 0.0776
Epoch 74/100
10/10 [==============================] - 1s 35ms/step - loss: 0.0050 - mae: 0.0775
Epoch 75/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0050 - mae: 0.0774
Epoch 76/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0050 - mae: 0.0773
Epoch 77/100
10/10 [==============================] - 1s 34ms/step - loss: 0.0050 - mae: 0.0772
Epoch 78/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0049 - mae: 0.0772
Epoch 79/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0049 - mae: 0.0772
Epoch 80/100
10/10 [==============================] - 1s 34ms/step - loss: 0.0049 - mae: 0.0771
Epoch 81/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0049 - mae: 0.0771
Epoch 82/100
10/10 [==============================] - 1s 37ms/step - loss: 0.0049 - mae: 0.0771
Epoch 83/100
10/10 [==============================] - 1s 34ms/step - loss: 0.0049 - mae: 0.0771
Epoch 84/100
10/10 [==============================] - 1s 68ms/step - loss: 0.0049 - mae: 0.0771
Epoch 85/100
10/10 [==============================] - 1s 72ms/step - loss: 0.0049 - mae: 0.0770
Epoch 86/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0049 - mae: 0.0770
Epoch 87/100
10/10 [==============================] - 1s 35ms/step - loss: 0.0049 - mae: 0.0770
Epoch 88/100
10/10 [==============================] - 1s 35ms/step - loss: 0.0049 - mae: 0.0770
Epoch 89/100
10/10 [==============================] - 1s 40ms/step - loss: 0.0049 - mae: 0.0770
Epoch 90/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0049 - mae: 0.0770
Epoch 91/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0049 - mae: 0.0770
Epoch 92/100
10/10 [==============================] - 1s 35ms/step - loss: 0.0049 - mae: 0.0770
Epoch 93/100
10/10 [==============================] - 1s 37ms/step - loss: 0.0049 - mae: 0.0770
Epoch 94/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0049 - mae: 0.0770
Epoch 95/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0049 - mae: 0.0770
Epoch 96/100
10/10 [==============================] - 1s 37ms/step - loss: 0.0049 - mae: 0.0770
Epoch 97/100
10/10 [==============================] - 1s 35ms/step - loss: 0.0049 - mae: 0.0770
Epoch 98/100
10/10 [==============================] - 1s 39ms/step - loss: 0.0049 - mae: 0.0770
Epoch 99/100
10/10 [==============================] - 1s 37ms/step - loss: 0.0049 - mae: 0.0770
Epoch 100/100
10/10 [==============================] - 1s 36ms/step - loss: 0.0049 - mae: 0.0770
```

✓ 1m 26s completed at 4:48 PM

```
import os
import random
import zipfile
import shutil
from shutil import copyfile
!pip install kaggle
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelBinarizer, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.13)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.27.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.65.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.26.15)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (2.0.12
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
```

```
from google.colab import files
files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session.
Please rerun this cell to enable.
Saving kaggle.json to kaggle (1).json
{'kaggle.json': b'{"username":"bestinvanfortuna","key":"858f25653ce72c96edcff809928fa347"}'}
```

```
os.environ['KAGGLE_CONFIG_DIR'] = '/content'
```

```
!kaggle datasets download -d catherinerasgaitis/mxmh-survey-results
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /content/kaggle.json'
mxmh-survey-results.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
data_dir = '/content/mxmh-survey-results.zip'
zip_ref  = zipfile.ZipFile(data_dir,'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
df = pd.read_csv('mxmh_survey_results.csv')
```

```
df.shape
```

```
(736, 33)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 736 entries, 0 to 735
Data columns (total 33 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Timestamp                     736 non-null    object 
 1   Age                           735 non-null    float64
 2   Primary streaming service     735 non-null    object 
 3   Hours per day                 736 non-null    float64
 4   While working                 733 non-null    object 
 5   Instrumentalist               732 non-null    object 
 6   Composer                      735 non-null    object 
 7   Fav genre                     736 non-null    object 
 8   Exploratory                   736 non-null    object 
 9   Foreign languages             732 non-null    object 
 10  BPM                           629 non-null    float64
 11  Frequency [Classical]         736 non-null    object 
 12  Frequency [Country]           736 non-null    object 
 13  Frequency [EDM]               736 non-null    object 
 14  Frequency [Folk]              736 non-null    object 
 15  Frequency [Gospel]            736 non-null    object 
 16  Frequency [Hip hop]           736 non-null    object 
 17  Frequency [Jazz]              736 non-null    object 
 18  Frequency [K pop]             736 non-null    object
```

```
 19  Frequency [Latin]             736 non-null    object
 20  Frequency [Lofi]              736 non-null    object
 21  Frequency [Metal]             736 non-null    object
 22  Frequency [Pop]               736 non-null    object
 23  Frequency [R&B]               736 non-null    object
 24  Frequency [Rap]               736 non-null    object
 25  Frequency [Rock]              736 non-null    object
 26  Frequency [Video game music]  736 non-null    object
 27  Anxiety                       736 non-null    float64
 28  Depression                    736 non-null    float64
 29  Insomnia                      736 non-null    float64
 30  OCD                           736 non-null    float64
 31  Music effects                 728 non-null    object
 32  Permissions                   736 non-null    object
dtypes: float64(7), object(26)
memory usage: 189.9+ KB
```

```
pd.set_option('display.max_columns', None )
df.head(10)
```

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Explo
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	
1	8/27/2022 19:57:31	63.0	Pandora	1.5	Yes	No	No	Rock	
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	
3	8/27/2022 21:40:40	61.0	YouTube Music	2.5	Yes	No	Yes	Jazz	
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	
5	8/27/2022 21:56:50	18.0	Spotify	5.0	Yes	Yes	Yes	Jazz	
6	8/27/2022 22:00:29	18.0	YouTube Music	3.0	Yes	Yes	No	Video game music	
7	8/27/2022 22:18:59	21.0	Spotify	1.0	Yes	No	No	K pop	
8	8/27/2022 22:33:05	19.0	Spotify	6.0	Yes	No	No	Rock	
9	8/27/2022 22:44:03	18.0	I do not use a streaming service.	1.0	Yes	No	No	R&B	

```
df = df.drop(columns = ['Timestamp','Age','Fav genre','Foreign languages','Permissions','Primary streaming service'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 736 entries, 0 to 735
Data columns (total 27 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Hours per day                 736 non-null    float64
 1   While working                 733 non-null    object
 2   Instrumentalist               732 non-null    object
 3   Composer                      735 non-null    object
 4   Exploratory                   736 non-null    object
 5   BPM                           629 non-null    float64
 6   Frequency [Classical]         736 non-null    object
 7   Frequency [Country]           736 non-null    object
 8   Frequency [EDM]               736 non-null    object
 9   Frequency [Folk]              736 non-null    object
 10  Frequency [Gospel]            736 non-null    object
 11  Frequency [Hip hop]           736 non-null    object
 12  Frequency [Jazz]              736 non-null    object
 13  Frequency [K pop]             736 non-null    object
 14  Frequency [Latin]             736 non-null    object
 15  Frequency [Lofi]              736 non-null    object
 16  Frequency [Metal]             736 non-null    object
 17  Frequency [Pop]               736 non-null    object
 18  Frequency [R&B]               736 non-null    object
```

```
 19  Frequency [Rap]               736 non-null    object
 20  Frequency [Rock]              736 non-null    object
 21  Frequency [Video game music]  736 non-null    object
 22  Anxiety                       736 non-null    float64
 23  Depression                    736 non-null    float64
 24  Insomnia                      736 non-null    float64
 25  OCD                           736 non-null    float64
 26  Music effects                 728 non-null    object
dtypes: float64(6), object(21)
memory usage: 155.4+ KB
```

```
df.isnull().sum()
```

```
Hours per day                     0
While working                     3
Instrumentalist                   4
Composer                          1
Exploratory                       0
BPM                             107
Frequency [Classical]             0
Frequency [Country]               0
Frequency [EDM]                   0
Frequency [Folk]                  0
Frequency [Gospel]                0
Frequency [Hip hop]               0
Frequency [Jazz]                  0
Frequency [K pop]                 0
Frequency [Latin]                 0
Frequency [Lofi]                  0
Frequency [Metal]                 0
Frequency [Pop]                   0
Frequency [R&B]                   0
Frequency [Rap]                   0
Frequency [Rock]                  0
Frequency [Video game music]      0
Anxiety                           0
Depression                        0
Insomnia                          0
OCD                               0
Music effects                     8
dtype: int64
```

```
df = df.dropna()
```

```
df.isna().sum()
```

```
Hours per day                   0
While working                   0
Instrumentalist                 0
Composer                        0
Exploratory                     0
BPM                             0
Frequency [Classical]           0
Frequency [Country]             0
Frequency [EDM]                 0
Frequency [Folk]                0
Frequency [Gospel]              0
Frequency [Hip hop]             0
Frequency [Jazz]                0
Frequency [K pop]               0
Frequency [Latin]               0
Frequency [Lofi]                0
Frequency [Metal]               0
Frequency [Pop]                 0
Frequency [R&B]                 0
Frequency [Rap]                 0
Frequency [Rock]                0
Frequency [Video game music]    0
Anxiety                         0
Depression                      0
Insomnia                        0
OCD                             0
Music effects                   0
dtype: int64
```

```
df.shape
```

```
(620, 27)
```

```
df = (
    df
    .rename(columns={
    'Frequency [Classical]': 'Classical',
    'Frequency [Country]': 'Country',
    'Frequency [EDM]': 'EDM',
```

```
    'Frequency [Folk]': 'Folk',
    'Frequency [Gospel]': 'Gospel',
    'Frequency [Hip hop]': 'Hip hop',
    'Frequency [Jazz]': 'Jazz',
    'Frequency [K pop]': 'K pop',
    'Frequency [Latin]': 'Latin',
    'Frequency [Lofi]': 'Lofi',
    'Frequency [Metal]': 'Metal',
    'Frequency [Pop]': 'Pop',
    'Frequency [R&B]': 'R&B',
    'Frequency [Rap]': 'Rap',
    'Frequency [Rock]': 'Rock',
    'Frequency [Video game music]': 'Games music'
}))
df
```

	Hours per day	While working	Instrumentalist	Composer	Exploratory	BPM	Classical	Country	EDM	Folk	
2	4.0	No	No	No	No	132.0	Never	Never	Very frequently	Never	
3	2.5	Yes	No	Yes	Yes	84.0	Sometimes	Never	Never	Rarely	Sc
4	4.0	Yes	No	No	Yes	107.0	Never	Never	Rarely	Never	
5	5.0	Yes	Yes	Yes	Yes	86.0	Rarely	Sometimes	Never	Never	
6	3.0	Yes	Yes	No	Yes	66.0	Sometimes	Never	Rarely	Sometimes	
...	...	...	...	...	...	...	...	...	...	...	
731	2.0	Yes	Yes	No	Yes	120.0	Very frequently	Rarely	Never	Sometimes	
732	1.0	Yes	Yes	No	Yes	160.0	Rarely	Rarely	Never	Never	
733	6.0	Yes	No	Yes	Yes	120.0	Rarely	Sometimes	Sometimes	Rarely	
734	5.0	Yes	Yes	No	No	170.0	Very frequently	Never	Never	Never	
735	2.0	Yes	No	No	Yes	98.0	Sometimes	Rarely	Very frequently	Sometimes	

620 rows × 27 columns

```
lb = LabelBinarizer()
binary_column = ['While working','Instrumentalist','Composer','Exploratory','Music effects']
df1 = df[binary_column]
for column in df1.columns :
  df[column] = lb.fit_transform(df1[column])
#df['While working'] = lb.fit_transform(df['While working'])
#df['Instrumentalist'] = lb.fit_transform(df['Instrumentalist'])
#df['Composer'] = lb.fit_transform(df['Composer'])
#df['Exploratory'] = lb.fit_transform(df['Exploratory'])
#df['Music effects'] = lb.fit_transform(df['Music effects'])
```

```
df
```

	Hours per day	While working	Instrumentalist	Composer	Exploratory	BPM	Classical	Country	EDM	Folk	
2	4.0	0	0	0	0	132.0	Never	Never	Very frequently	Never	
3	2.5	1	0	1	1	84.0	Sometimes	Never	Never	Rarely	Sc
4	4.0	1	0	0	1	107.0	Never	Never	Rarely	Never	
5	5.0	1	1	1	1	86.0	Rarely	Sometimes	Never	Never	
6	3.0	1	1	0	1	66.0	Sometimes	Never	Rarely	Sometimes	
...	...	...	...	...	...	...	...	...	...	...	
731	2.0	1	1	0	1	120.0	Very frequently	Rarely	Never	Sometimes	
732	1.0	1	1	0	1	160.0	Rarely	Rarely	Never	Never	

```
df['Classical'].unique()
```

```
array(['Never', 'Sometimes', 'Rarely', 'Very frequently'], dtype=object)
```

```
replace_dir = {'Never': 0, 'Sometimes': 2, 'Rarely':1 , 'Very frequently' : 4}
change = ['Classical','Country','EDM','Folk','Gospel','Hip hop','Jazz','K pop','Latin','Lofi','Metal','Pop','R&B','Rap','Rock','Games mus
df[change] = df[change].replace(replace_dir)
```

```
df
```

	Hours per day	While working	Instrumentalist	Composer	Exploratory	BPM	Classical	Country	EDM	Folk	Gospel	Hip hop	Jaz
2	4.0	0	0	0	0	132.0	0	0	4	0	0	1	
3	2.5	1	0	1	1	84.0	2	0	0	1	2	0	
4	4.0	1	0	0	1	107.0	0	0	1	0	1	4	
5	5.0	1	1	1	1	86.0	1	2	0	0	0	2	
6	3.0	1	1	0	1	66.0	2	0	1	2	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
731	2.0	1	1	0	1	120.0	4	1	0	2	0	2	
732	1.0	1	1	0	1	160.0	1	1	0	0	0	0	
733	6.0	1	0	1	1	120.0	1	2	2	1	1	4	
734	5.0	1	1	0	0	170.0	4	0	0	0	0	0	
735	2.0	1	0	0	1	98.0	2	1	4	2	1	4	

620 rows × 27 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 620 entries, 2 to 735
Data columns (total 27 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Hours per day    620 non-null    float64
 1   While working    620 non-null    int64  
 2   Instrumentalist  620 non-null    int64  
 3   Composer         620 non-null    int64  
 4   Exploratory      620 non-null    int64  
 5   BPM              620 non-null    float64
 6   Classical        620 non-null    int64  
 7   Country          620 non-null    int64  
 8   EDM              620 non-null    int64  
 9   Folk             620 non-null    int64  
 10  Gospel           620 non-null    int64  
 11  Hip hop          620 non-null    int64  
 12  Jazz             620 non-null    int64  
 13  K pop            620 non-null    int64  
 14  Latin            620 non-null    int64
```

```
 15  Lofi             620 non-null    int64
 16  Metal            620 non-null    int64
 17  Pop              620 non-null    int64
 18  R&B              620 non-null    int64
 19  Rap              620 non-null    int64
 20  Rock             620 non-null    int64
 21  Games music      620 non-null    int64
 22  Anxiety          620 non-null    float64
 23  Depression       620 non-null    float64
 24  Insomnia         620 non-null    float64
 25  OCD              620 non-null    float64
 26  Music effects    620 non-null    int64
dtypes: float64(6), int64(21)
memory usage: 135.6 KB
```

```
label_column = ['Music effects']
feature_columns = df.columns.drop(label_column)
```

```
label = df['Music effects']
label
```

```
2      0
3      1
4      1
5      1
6      1
      ..
731    1
732    1
733    1
734    1
735    1
Name: Music effects, Length: 620, dtype: int64
```

```
feature = df.drop(columns = 'Music effects',axis = 0)
feature
```

	Hours per day	While working	Instrumentalist	Composer	Exploratory	BPM	Classical	Country	EDM	Folk	Gospel	Hip hop	Jaz
2	4.0	0	0	0	0	132.0	0	0	4	0	0	1	
3	2.5	1	0	1	1	84.0	2	0	0	1	2	0	
4	4.0	1	0	0	1	107.0	0	0	1	0	1	4	
5	5.0	1	1	1	1	86.0	1	2	0	0	0	2	
6	3.0	1	1	0	1	66.0	2	0	1	2	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
731	2.0	1	1	0	1	120.0	4	1	0	2	0	2	
732	1.0	1	1	0	1	160.0	1	1	0	0	0	0	
733	6.0	1	0	1	1	120.0	1	2	2	1	1	4	
734	5.0	1	1	0	0	170.0	4	0	0	0	0	0	
735	2.0	1	0	0	1	98.0	2	1	4	2	1	4	

620 rows × 26 columns

```
x_training, x_test, y_training, y_test = train_test_split(feature,label,test_size = 0.3,random_state = 1234567)
```

```
for column in [x_training,x_test, y_training,y_test]:
  print(column.shape)
```

```
(434, 26)
(186, 26)
(434,)
(186,)
```

```
model = LogisticRegression(class_weight = 'balanced', max_iter = 500)
model.fit(x_training, y_training)
```

```
LogisticRegression(class_weight='balanced', max_iter=500)
```

```
model.score(x_training, y_training)
```

```
0.6682027649769585
```

```
model.score(x_test, y_test)
```

```
0.6505376344086021
```

```
X_test2 = x_test.copy()
X_test2['prediction'] = model.predict(X_test2[feature_columns])
X_test2[['probability_no','probability_yes']] = model.predict_proba(X_test2[feature_columns])
X_test2['prediction'] = X_test2['prediction'].replace({0: "Sakit jiwa", 1: "No mental health issue"})
X_test2
```

	Hours per day	While working	Instrumentalist	Composer	Exploratory	BPM	Classical	Country	EDM	Folk	Gospel	Hip hop	Jaz
692	4.0	1	0	0	1	105.0	1	1	4	1	1	1	
81	3.0	1	0	0	1	118.0	1	0	1	0	0	4	
80	5.0	0	1	1	1	90.0	1	0	0	1	0	2	
582	1.0	0	1	0	0	79.0	1	0	0	1	2	0	
455	0.0	0	1	0	0	115.0	1	1	0	2	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
268	1.0	0	0	1	1	113.0	4	0	0	2	0	1	
381	2.0	1	0	0	1	114.0	1	1	0	2	0	0	
308	4.0	1	0	0	1	140.0	2	2	1	1	0	2	
7	1.0	1	0	0	1	95.0	0	0	1	0	0	4	
231	6.0	1	0	0	1	104.0	2	0	0	1	0	1	

186 rows × 29 columns