How accurately can machine learning models predict mental health outcomes based on social media usage patterns and demographic factors?

```
# prompt: upload excl file from gogle colan and print

from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

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 mental_health_and_technology_usage_2022.xlsx to mental_health_and_technology_usage_2022.xlsx
User uploaded file "mental_health_and_technology_usage_2022.xlsx" with length 965160 bytes
```

```
import pandas as pd

# Replace the file path with the actual path of your Excel file
file_path = 'mental_health_and_technology_usage_2022.xlsx'

# Load the Excel file into a Pandas DataFrame
df = pd.read_excel(file_path)

# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
     Timestap     User_ID  Age  Birth Year Generation  Technology_Usage_Hours  \
0 2022-04-01  USER-00001   23        1999      Gen Z                    6.57
1 2022-04-01  USER-00002   21        2001      Gen Z                    3.01
2 2022-04-01  USER-00003   51        1971      Gen X                    3.04
3 2022-04-01  USER-00004   25        1997      Gen Z                    3.84
4 2022-04-01  USER-00005   53        1969      Gen X                    1.20

   Social_Media_Usage_Hours  Gaming_Hours  Screen_Time_Hours  \
0                      6.00          0.68              12.36
1                      2.57          3.74               7.61
2                      6.14          1.26               3.16
3                      4.48          2.59              13.08
4                      0.56          0.29              12.63

  Mental_Health_Status Stress_Level  Sleep_Hours  Physical_Activity_Hours  \
0                 Good          Low         8.01                     6.71
1                 Poor         High         7.28                     5.88
2                 Fair         High         8.04                     9.81
3            Excellent       Medium         5.62                     5.28
4                 Good          Low         5.55                     4.00

  Support_Systems_Access Work_Environment_Impact Online_Support_Usage
0                     No                Negative                  Yes
1                    Yes                Positive                   No
2                     No                Negative                   No
3                    Yes                Negative                  Yes
4                     No                Positive                  Yes
```

```
df.columns
```

```
Index(['level_0', 'index', 'Timestap', 'User_ID', 'Age', 'Birth Year',
       'Generation', 'Technology_Usage_Hours', 'Social_Media_Usage_Hours',
       'Gaming_Hours', 'Screen_Time_Hours', 'Mental_Health_Status',
       'Stress_Level', 'Sleep_Hours', 'Physical_Activity_Hours',
       'Support_Systems_Access', 'Work_Environment_Impact',
       'Online_Support_Usage'],
      dtype='object')
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np



# Encode categorical columns (e.g., Generation, Mental Health Status)
label_encoder = LabelEncoder()

df['Mental_Health_Status'] = label_encoder.fit_transform(df['Mental_Health_Status'])
df['Generation'] = label_encoder.fit_transform(df['Generation'])

# Select features and target
features = ['Technology_Usage_Hours', 'Social_Media_Usage_Hours', 'Gaming_Hours',
            'Screen_Time_Hours', 'Sleep_Hours', 'Physical_Activity_Hours', 'Age', 'Generation']
target = 'Mental_Health_Status'

X = df[features].values
y = df[target].values

# Scale the features for LSTM
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Reshape input for LSTM [samples, time steps, features]
X_reshaped = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.2, random_state=42)
```

```
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam

# Define the LSTM model
model = Sequential()

# LSTM layer
model.add(LSTM(50, return_sequences=False, input_shape=(X_train.shape[1], X_train.shape[2])))

# Dropout to prevent overfitting
model.add(Dropout(0.2))

# Fully connected layer with one output (for binary classification)
model.add(Dense(1, activation='sigmoid'))  # Use 'softmax' for multiclass classification

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# For multiclass classification, replace 'binary_crossentropy' with 'categorical_crossentropy'
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

```
Epoch 1/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.2480 - loss: 0.4430 - val_accuracy: 0.2450 - val_loss: -0.7672
Epoch 2/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2523 - loss: -1.2684 - val_accuracy: 0.2450 - val_loss: -3.1345
Epoch 3/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2442 - loss: -3.6863 - val_accuracy: 0.2450 - val_loss: -6.2010
Epoch 4/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2452 - loss: -6.9134 - val_accuracy: 0.2450 - val_loss: -9.8138
Epoch 5/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2524 - loss: -10.4796 - val_accuracy: 0.2450 - val_loss: -13.9429
Epoch 6/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2465 - loss: -14.8679 - val_accuracy: 0.2450 - val_loss: -18.8514
Epoch 7/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2547 - loss: -20.0572 - val_accuracy: 0.2450 - val_loss: -24.4613
Epoch 8/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2587 - loss: -25.9312 - val_accuracy: 0.2450 - val_loss: -30.7899
Epoch 9/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2509 - loss: -31.3494 - val_accuracy: 0.2450 - val_loss: -37.2122
Epoch 10/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2501 - loss: -38.0416 - val_accuracy: 0.2450 - val_loss: -43.1484
Epoch 11/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2516 - loss: -41.6221 - val_accuracy: 0.2450 - val_loss: -48.5851
Epoch 12/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2517 - loss: -46.9211 - val_accuracy: 0.2450 - val_loss: -53.7259
Epoch 13/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2527 - loss: -50.6975 - val_accuracy: 0.2450 - val_loss: -58.5311
Epoch 14/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2453 - loss: -56.6811 - val_accuracy: 0.2450 - val_loss: -63.2108
Epoch 15/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2393 - loss: -65.9260 - val_accuracy: 0.2450 - val_loss: -67.7141
Epoch 16/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2491 - loss: -63.8484 - val_accuracy: 0.2450 - val_loss: -72.1178
Epoch 17/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2494 - loss: -67.1086 - val_accuracy: 0.2450 - val_loss: -76.4234
Epoch 18/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2470 - loss: -73.5372 - val_accuracy: 0.2450 - val_loss: -80.6813
Epoch 19/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2457 - loss: -78.4938 - val_accuracy: 0.2450 - val_loss: -84.8455
Epoch 20/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2531 - loss: -82.2724 - val_accuracy: 0.2450 - val_loss: -88.9263
```

```
# Evaluate the model on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2424 - loss: -92.2240
Test Accuracy: 0.24899999797344208
```

```
# Predict on new data
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_class = (y_pred > 0.5).astype(int)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

Start coding or generate with AI.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.optimizers import Adam
from sklearn.utils import class_weight

# Assuming 'df' is your DataFrame

# Step 2: Data Preprocessing
label_encoder = LabelEncoder()

df['Mental_Health_Status'] = label_encoder.fit_transform(df['Mental_Health_Status'])
df['Generation'] = label_encoder.fit_transform(df['Generation'])

features = ['Technology_Usage_Hours', 'Social_Media_Usage_Hours', 'Gaming_Hours',
            'Screen_Time_Hours', 'Sleep_Hours', 'Physical_Activity_Hours', 'Age', 'Generation']
target = 'Mental_Health_Status'

X = df[features].values
y = df[target].values

# Step 3: Feature Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Reshape the input to [samples, timesteps, features]
X_reshaped = X_scaled.reshape((X_scaled.shape[0], 1, X_scaled.shape[1]))

# Step 4: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_reshaped, y, test_size=0.2, random_state=42)

# Step 5: Handle Class Imbalance (if needed)
# Compute class weights to handle imbalanced datasets
class_weights = class_weight.compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)

# Convert class_weights to a dictionary
class_weight_dict = dict(enumerate(class_weights))
# This maps class indices (0, 1, 2, etc.) to their corresponding weights.


# Step 6: Define the LSTM Model
model = Sequential()

# Add LSTM layers
```

```
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(50))

# Add Dropout to prevent overfitting
model.add(Dropout(0.2))

# Output layer for binary classification (change to softmax for multi-class classification)
model.add(Dense(1, activation='sigmoid'))  # Use softmax if predicting multiple classes

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Step 7: Model Training
# Fit the model, using class weights if there's imbalance
history = model.fit(X_train, y_train, epochs=20, batch_size=32, class_weight=class_weight_dict, validation_split=0.2)
# Use class_weight_dict here

# Step 8: Model Evaluation
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc}")

# Step 9: Making Predictions
# Predict on the test set
y_pred = model.predict(X_test)

# Convert predicted probabilities to binary class labels
y_pred_class = (y_pred > 0.5).astype(int)

# Print the first 5 predictions and their corresponding actual values
print("Predicted Classes: ", y_pred_class[:5].ravel())
print("Actual Classes: ", y_test[:5])
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.2495 - loss: 0.6730 - val_accuracy: 0.2450 - val_loss: 0.6012
Epoch 2/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.2386 - loss: 0.5613 - val_accuracy: 0.2450 - val_loss: 0.3869
Epoch 3/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2397 - loss: 0.2825 - val_accuracy: 0.2450 - val_loss: -0.1848
Epoch 4/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2501 - loss: -0.3909 - val_accuracy: 0.2450 - val_loss: -1.2023
Epoch 5/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2408 - loss: -1.4840 - val_accuracy: 0.2450 - val_loss: -2.4711
Epoch 6/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2372 - loss: -2.9070 - val_accuracy: 0.2450 - val_loss: -3.9024
Epoch 7/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2453 - loss: -4.3107 - val_accuracy: 0.2450 - val_loss: -5.2760
Epoch 8/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2464 - loss: -5.2151 - val_accuracy: 0.2450 - val_loss: -6.4570
Epoch 9/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2459 - loss: -6.6576 - val_accuracy: 0.2450 - val_loss: -7.4560
Epoch 10/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2602 - loss: -7.2293 - val_accuracy: 0.2450 - val_loss: -8.2727
Epoch 11/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.2496 - loss: -8.0249 - val_accuracy: 0.2450 - val_loss: -8.9925
Epoch 12/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.2480 - loss: -8.5994 - val_accuracy: 0.2450 - val_loss: -9.6185
Epoch 13/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2499 - loss: -9.8081 - val_accuracy: 0.2450 - val_loss: -10.1848
Epoch 14/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2531 - loss: -9.3492 - val_accuracy: 0.2450 - val_loss: -10.7133
Epoch 15/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2544 - loss: -11.2548 - val_accuracy: 0.2450 - val_loss: -11.2018
Epoch 16/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2499 - loss: -10.6837 - val_accuracy: 0.2450 - val_loss: -11.6629
Epoch 17/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2500 - loss: -11.5096 - val_accuracy: 0.2450 - val_loss: -12.1099
Epoch 18/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2396 - loss: -12.0591 - val_accuracy: 0.2450 - val_loss: -12.5388
Epoch 19/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2567 - loss: -12.0162 - val_accuracy: 0.2450 - val_loss: -12.9579
Epoch 20/20
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.2502 - loss: -11.6326 - val_accuracy: 0.2450 - val_loss: -13.3705
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2424 - loss: -13.8698
Test Accuracy: 0.24899999797344208
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
Predicted Classes:  [1 1 1 1 1]
Actual Classes:  [3 0 0 0 0]
```

```
# Predict on the test set
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_class = np.argmax(y_pred, axis=1)

# Print the predicted values (class labels)
print("Predicted Class Labels: ", y_pred_class)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted Class Labels:  [1 1 2 ... 1 0 0]
```

```
# Print the predicted probabilities
print("Predicted Probabilities: ", y_pred)
```

```
Predicted Probabilities:  [[0.13675968 0.31579185 0.28575101 0.2616974 ]
 [0.25943047 0.3071715  0.24181698 0.19158106]
 [0.2483383  0.19383757 0.3116312  0.24619292]
 ...
 [0.25587097 0.2753247  0.23754545 0.23125885]
 [0.26063883 0.23154579 0.25174993 0.25606543]
 [0.2891149  0.18150495 0.2672633  0.26211685]]
```

```
import pandas as pd
import numpy as np

# Predict on the test set
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_class = np.argmax(y_pred, axis=1)

# Assuming 'Age' is in the features, extract the 'Age' column from the test set
age_test = X_test[:, features.index('Age')]

# Create age groups by binning the age values into categories
age_bins = [0, 18, 30, 40, 50, 60, 100]  # Define the age ranges
age_labels = ['<18', '18-30', '31-40', '41-50', '51-60', '60+']  # Labels for each bin
age_groups = pd.cut(age_test, bins=age_bins, labels=age_labels)

# Combine Age groups and predicted class labels into a DataFrame for easier viewing
predictions_with_age_groups = pd.DataFrame({'Age_Group': age_groups, 'Predicted_Stress_Class': y_pred_class})

# Group the results by age group and calculate the count of each prediction in each group
grouped_predictions = predictions_with_age_groups.groupby('Age_Group')['Predicted_Stress_Class'].value_counts().unstack()

# Print the grouped results
print(grouped_predictions)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Predicted_Stress_Class    0    1    2    3
Age_Group
<18                     310  367  102  190
18-30                     0    0    0    0
31-40                     0    0    0    0
41-50                     0    0    0    0
51-60                     0    0    0    0
60+                       0    0    0    0
<ipython-input-28-ddc6cfce3c5f>:22: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  grouped_predictions = predictions_with_age_groups.groupby('Age_Group')['Predicted_Stress_Class'].value_counts().unstack()
```

```
import pandas as pd
import numpy as np

# Predict on the test set
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_class = np.argmax(y_pred, axis=1)

# Assuming 'Social_Media_Usage_Hours' is in the features, extract it from the test set
social_media_test = X_test[:, features.index('Social_Media_Usage_Hours')]

# Create social media usage groups by binning the usage values into categories
social_media_bins = [0, 1, 3, 5, 8, 12]  # Define the social media usage ranges (in hours)
social_media_labels = ['0-1 hrs', '1-3 hrs', '3-5 hrs', '5-8 hrs', '8+ hrs']  # Labels for each bin
social_media_groups = pd.cut(social_media_test, bins=social_media_bins, labels=social_media_labels)

# Combine Social Media usage groups and predicted class labels into a DataFrame for easier viewing
predictions_with_social_media_groups = pd.DataFrame({'Social_Media_Usage_Group': social_media_groups, 'Predicted_Stress_Class': y_pred_class})

# Group the results by social media usage group and calculate the count of each prediction in each group
grouped_predictions = predictions_with_social_media_groups.groupby('Social_Media_Usage_Group')['Predicted_Stress_Class'].value_counts().unstack()

# Print the grouped results
print(grouped_predictions)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Predicted_Stress_Class      0    1   2    3
Social_Media_Usage_Group
0-1 hrs                   168  167  75  144
1-3 hrs                   122  104  78  126
3-5 hrs                     0    0   0    0
5-8 hrs                     0    0   0    0
8+ hrs                      0    0   0    0
<ipython-input-29-237185ceca91>:22: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  grouped_predictions = predictions_with_social_media_groups.groupby('Social_Media_Usage_Group')['Predicted_Stress_Class'].value_counts().unstack()
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Assuming df is your original DataFrame and the target is 'Mental_Health_Status'
features = ['Age', 'Generation', 'Technology_Usage_Hours', 'Social_Media_Usage_Hours',
            'Gaming_Hours', 'Screen_Time_Hours', 'Stress_Level', 'Sleep_Hours',
            'Physical_Activity_Hours']  # Ensure these match during training and testing

# Reset the index of your DataFrame to ensure consistent indexing
df = df.reset_index(drop=True)

# Create a LabelEncoder object
encoder = LabelEncoder()

# Apply Label Encoding to categorical columns ('Generation' and 'Stress_Level')
for column in ['Generation', 'Stress_Level']:  # Add any other categorical features as needed
    df[column] = encoder.fit_transform(df[column])

# Split the data into training and testing sets
X = df[features]
y = df['Mental_Health_Status']

# Split data while keeping track of the indices
X_train, X_test, y_train, y_test, X_train_indices, X_test_indices = train_test_split(
    X, y, df.index, test_size=0.2, random_state=42
)

# Ensure the shape of X_train and X_test matches
print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")

# Ensure model is trained on X_train and y_train
# Example of model creation (adjust as needed for your specific model)
from keras.models import Sequential
from keras.layers import Dense, Dropout

# Define a basic model structure (adjust to your needs)
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))  # input_dim = number of features
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification for 'Mental_Health_Status'

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Predict on the test set
y_pred = model.predict(X_test)

# Convert predicted probabilities to class labels
y_pred_class = (y_pred > 0.5).astype(int)  # Assuming binary classification, adjust for multi-class

# Extract 'Social_Media_Usage_Hours' and 'Timestamp' from the original df using the test indices
```

```
social_media_test = df.loc[X_test_indices, 'Social_Media_Usage_Hours'].values
time_test = df.loc[X_test_indices, 'Timestap'].values  # Fixing typo from 'Timestap' to 'Timestamp' if needed

# Convert 'Timestap' to datetime if necessary
time_test = pd.to_datetime(time_test)

# Sort by time to analyze predictions over time
sorted_indices = np.argsort(time_test)
social_media_test_sorted = social_media_test[sorted_indices]
y_pred_class_sorted = y_pred_class[sorted_indices]
time_test_sorted = time_test[sorted_indices]
```

```
Shape of X_train: (8000, 9)
Shape of X_test: (2000, 9)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.2460 - loss: -501.8250
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.2410 - loss: -21115.8516
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.2498 - loss: -121038.8125
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.2519 - loss: -376366.0312
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2536 - loss: -837826.3750
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2422 - loss: -1523003.3750
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2429 - loss: -2556390.0000
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2513 - loss: -3600284.7500
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2511 - loss: -5192044.0000
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.2550 - loss: -6791245.0000
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
# Ensure all arrays are 1-dimensional
time_test_sorted = np.array(time_test_sorted).ravel()  # Ensures time is 1D
social_media_test_sorted = np.array(social_media_test_sorted).ravel()  # Ensures social media usage is 1D
y_pred_class_sorted = np.array(y_pred_class_sorted).ravel()  # Ensures predictions are 1D

# Check that all arrays have the same length before combining into a DataFrame
print(f"Time length: {len(time_test_sorted)}")
print(f"Social Media Usage length: {len(social_media_test_sorted)}")
print(f"Predicted Class length: {len(y_pred_class_sorted)}")

# Combine the time, social media usage, and predicted class labels into a DataFrame
predictions_with_time = pd.DataFrame({
    'Time': time_test_sorted,
    'Social_Media_Usage': social_media_test_sorted,
    'Predicted_Stress_Class': y_pred_class_sorted
})

# Group by time-based intervals (e.g., by month, week) to analyze trends
predictions_with_time['Month'] = predictions_with_time['Time'].dt.to_period('M')  # Group by month

# Aggregate the data to get the counts of each predicted stress class per time period
grouped_predictions_over_time = predictions_with_time.groupby('Month')['Predicted_Stress_Class'].value_counts().unstack()

# Print the grouped results
print(grouped_predictions_over_time)
```

```
Time length: 2000
Social Media Usage length: 2000
Predicted Class length: 2000
Predicted_Stress_Class    1
Month
2020-07                  65
2020-08                 105
2022-04                  39
2022-05                 202
2022-06                 271
2022-09                 485
2022-10                 299
2022-11                 246
2022-12                 288
```