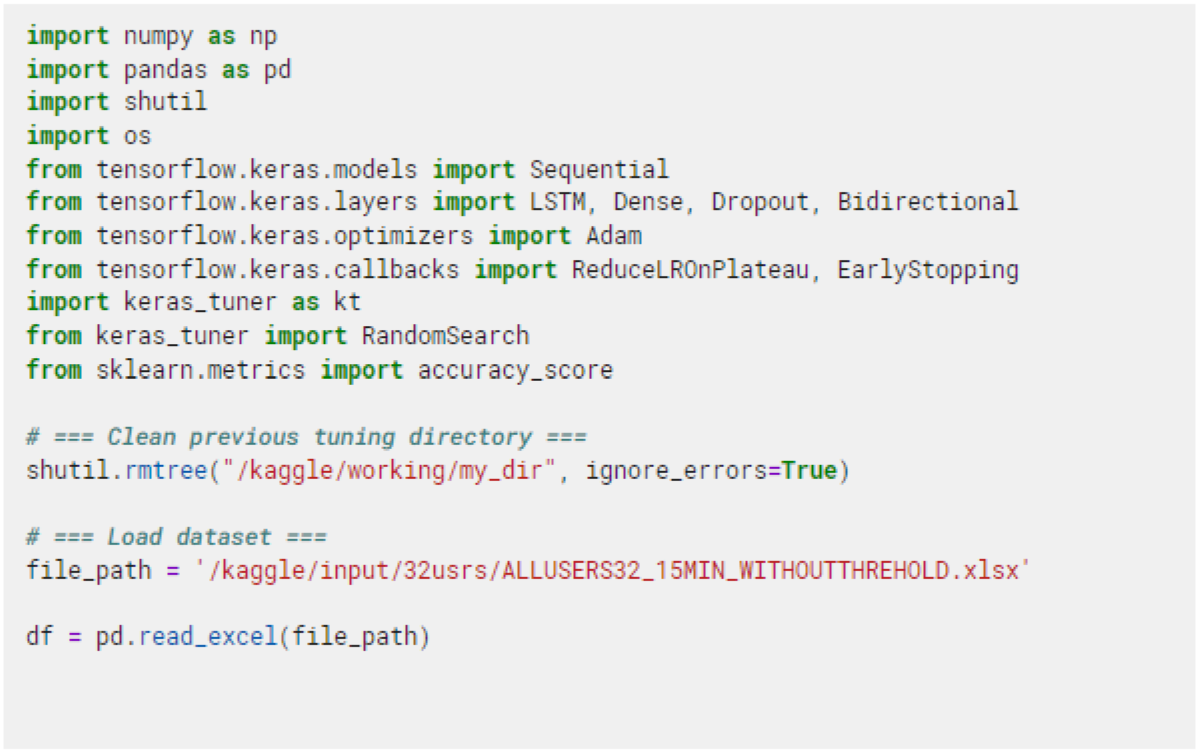
MODIFICATIONS AND INSTRUCTIONS: STEP DATA CODE

The modified code had automated the sections mentioned in the meeting.

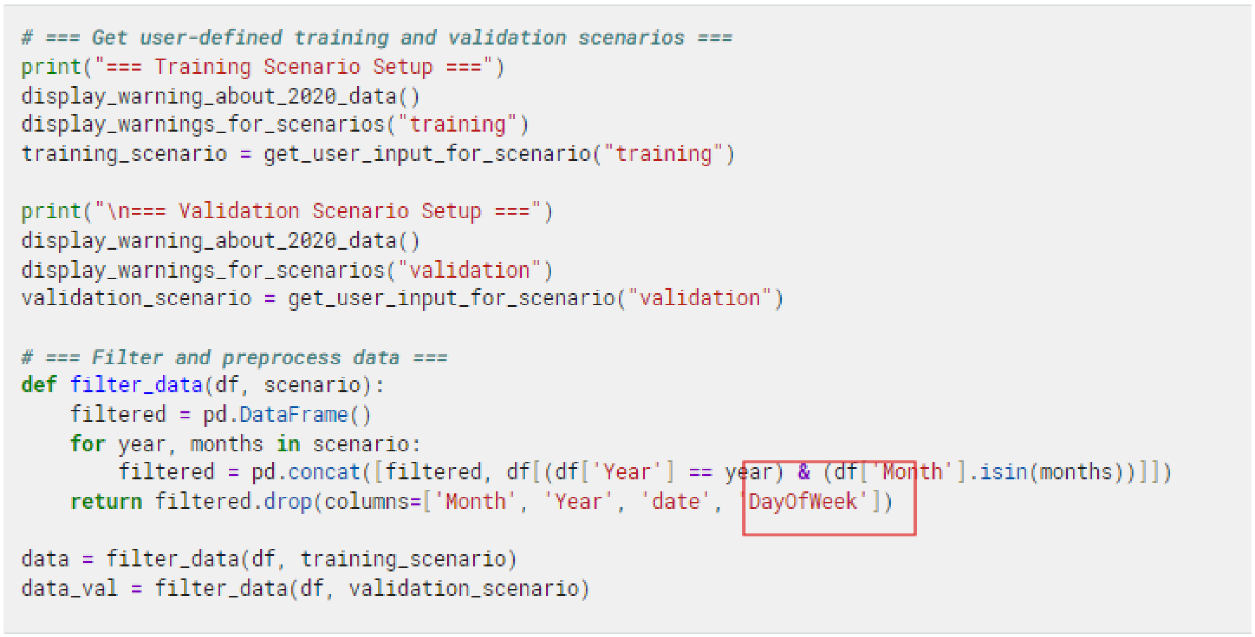
FEW INSTRUCTIONS FOR RUNNING THE CODE

1. **We have 4 datasets, the differences are considered to be known. For selecting the datasets, change the 'file path'. See the screenshot.**



```python
import numpy as np
import pandas as pd
import shutil
import os
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
import keras_tuner as kt
from keras_tuner import RandomSearch
from sklearn.metrics import accuracy_score

# === Clean previous tuning directory ===
shutil.rmtree("/kaggle/working/my_dir", ignore_errors=True)

# === Load dataset ===
file_path = '/kaggle/input/32usrs/ALLUSERS32_15MIN_WITHOUTTHREHOLD.xlsx'

df = pd.read_excel(file_path)
```

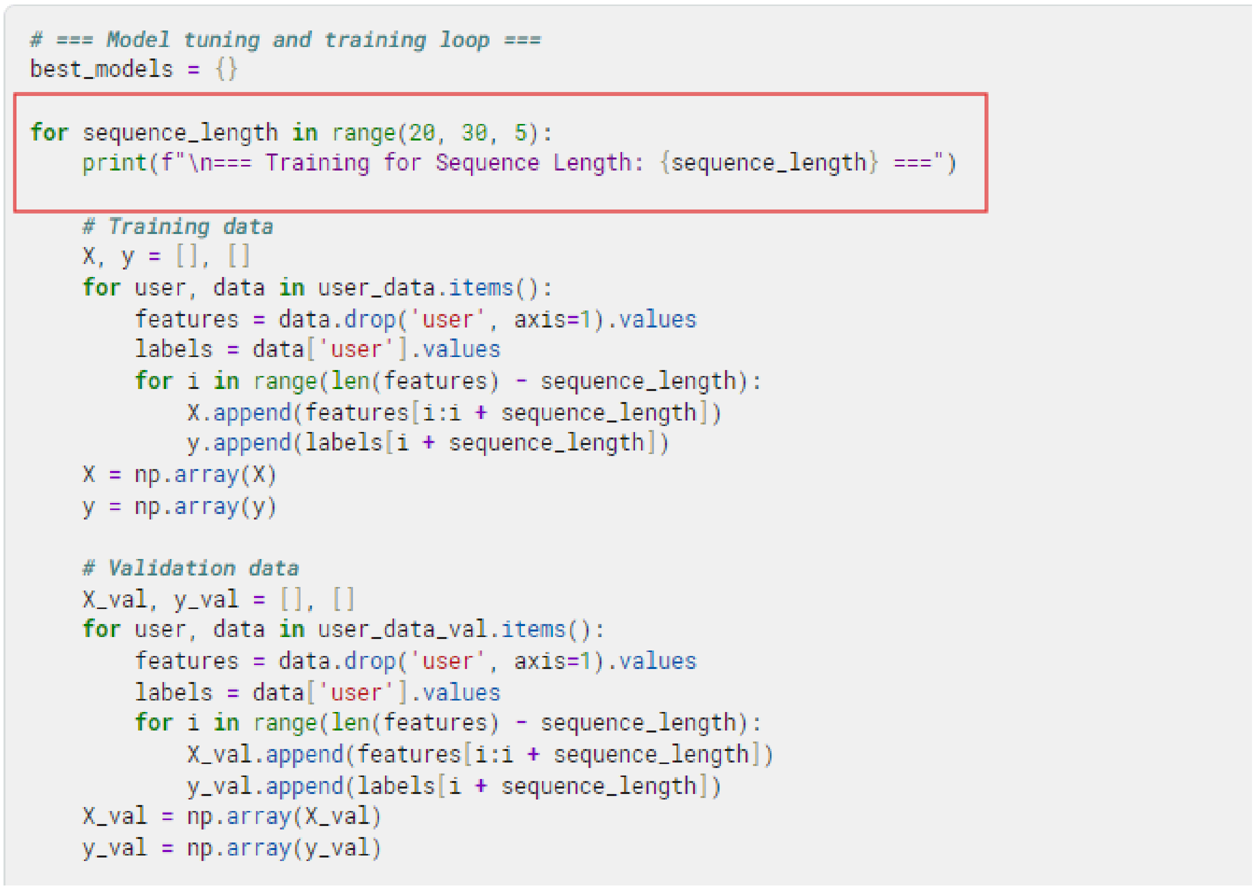
Special Instruction: While running only this file 'ALLUSERS32_15MIN_WITHOUTTHREHOLD.xlsx', we need to remove an additional column 'DayofWeek'. This column is not present in other datasets, so kindly remove this *column name from the drop function when running other files.*



```
# === Get user-defined training and validation scenarios ===
print("=== Training Scenario Setup ===")
display_warning_about_2020_data()
display_warnings_for_scenarios("training")
training_scenario = get_user_input_for_scenario("training")

print("\n=== Validation Scenario Setup ===")
display_warning_about_2020_data()
display_warnings_for_scenarios("validation")
validation_scenario = get_user_input_for_scenario("validation")

# === Filter and preprocess data ===
def filter_data(df, scenario):
    filtered = pd.DataFrame()
    for year, months in scenario:
        filtered = pd.concat([filtered, df[(df['Year'] == year) & (df['Month'].isin(months))]])
    return filtered.drop(columns=['Month', 'Year', 'date', 'DayOfWeek'])

data = filter_data(df, training_scenario)
data_val = filter_data(df, validation_scenario)
```

2. **When running the code, you'll be prompted to enter the training and validation periods. A warning will display a previously tested time period; you can either:**
   i.Copy and paste the same period to use the predefined setup, which is the already tested setup.
   ii. Enter a custom time period following the prompt's instruction based on choice.

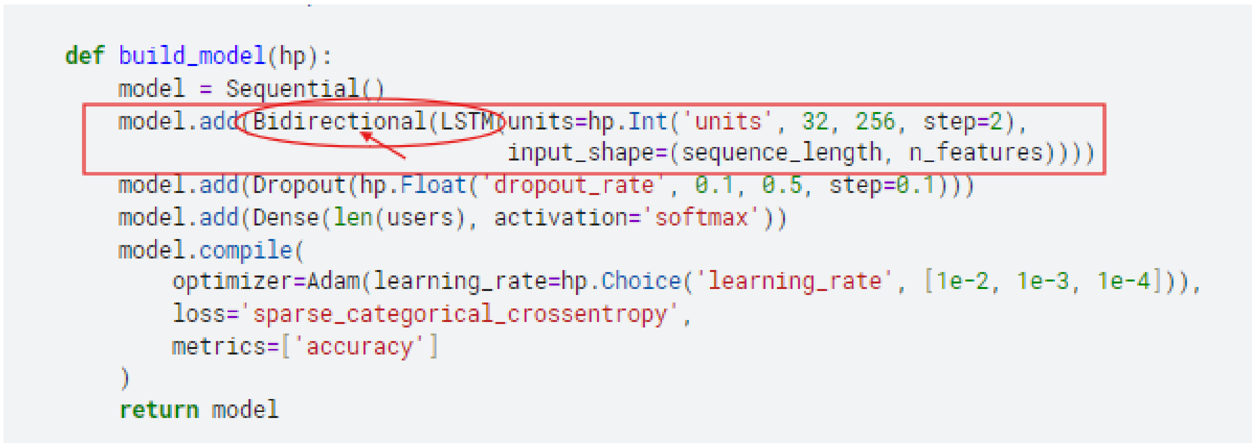
3. **The sequence length can be changed in the for loop depending on your choice. See the screenshot below.**



```
# === Model tuning and training loop ===
best_models = {}

for sequence_length in range(20, 30, 5):
    print(f"\n=== Training for Sequence Length: {sequence_length} ===")

    # Training data
    X, y = [], []
    for user, data in user_data.items():
        features = data.drop('user', axis=1).values
        labels = data['user'].values
        for i in range(len(features) - sequence_length):
            X.append(features[i:i + sequence_length])
            y.append(labels[i + sequence_length])
    X = np.array(X)
    y = np.array(y)

    # Validation data
    X_val, y_val = [], []
    for user, data in user_data_val.items():
        features = data.drop('user', axis=1).values
        labels = data['user'].values
        for i in range(len(features) - sequence_length):
            X_val.append(features[i:i + sequence_length])
            y_val.append(labels[i + sequence_length])
    X_val = np.array(X_val)
    y_val = np.array(y_val)
```

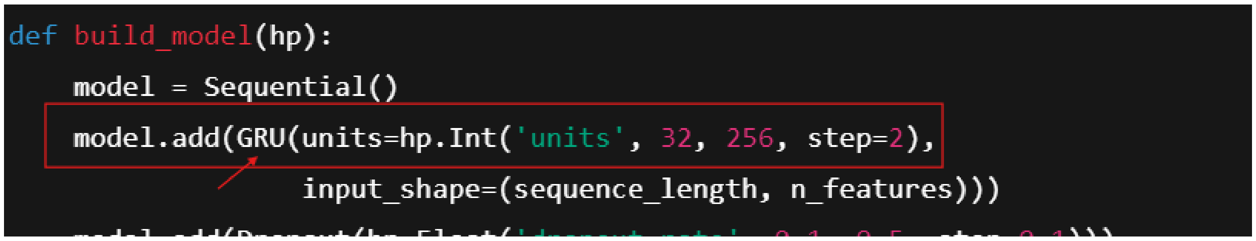
for sequence_length in range(20, 30, 5):
Definition of range:- (Start,Stop,Step) follow this to choose your choice.

4.This build_model function which selects the RNN algorithm



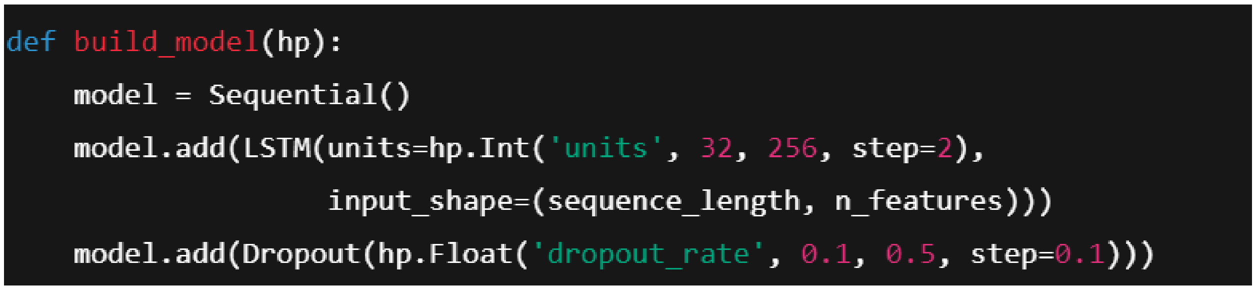
```
def build_model(hp):
    model = Sequential()
    model.add(Bidirectional(LSTM(units=hp.Int('units', 32, 256, step=2),
                                 input_shape=(sequence_length, n_features))))
    model.add(Dropout(hp.Float('dropout_rate', 0.1, 0.5, step=0.1)))
    model.add(Dense(len(users), activation='softmax'))
    model.compile(
        optimizer=Adam(learning_rate=hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model
```

Replace the circle marking in the above screenshot with ‘GRU’ or ‘LSTM’as in the screenshot shown below.
GRU



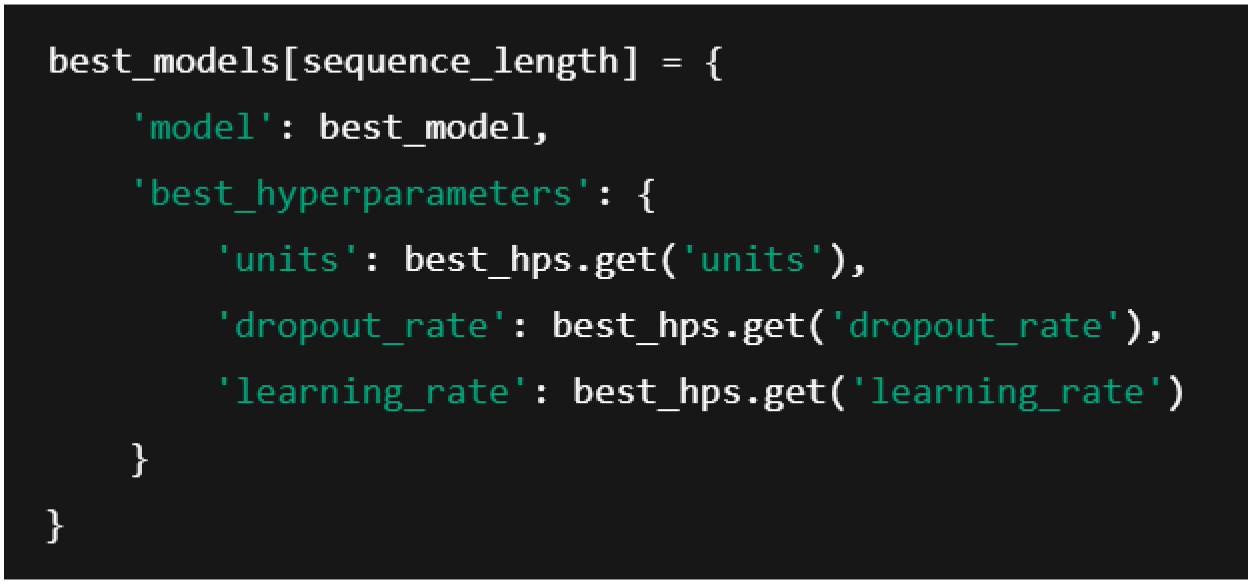
```
def build_model(hp):
    model = Sequential()
    model.add(GRU(units=hp.Int('units', 32, 256, step=2),
                  input_shape=(sequence_length, n_features)))
    model.add(Dropout(hp.Float('dropout_rate', 0.1, 0.5, step=0.1)))
```

LSTM



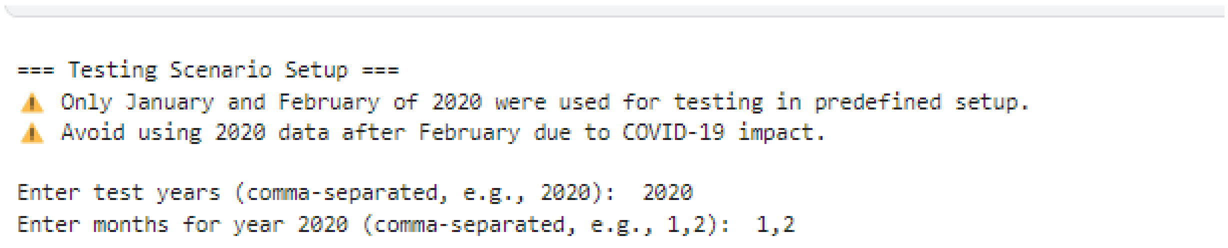
```
def build_model(hp):
    model = Sequential()
    model.add(LSTM(units=hp.Int('units', 32, 256, step=2),
                   input_shape=(sequence_length, n_features)))
    model.add(Dropout(hp.Float('dropout_rate', 0.1, 0.5, step=0.1)))
```

5. At the end of each iteration sequence length along with other best parameters are stored as a dictionary.The structure of dictionary looks like in the screenshot.



```
best_models[sequence_length] = {
    'model': best_model,
    'best_hyperparameters': {
        'units': best_hps.get('units'),
        'dropout_rate': best_hps.get('dropout_rate'),
        'learning_rate': best_hps.get('learning_rate')
    }
}
```

6. When the for loop ends, you will be prompted to enter the test data period.The already tested scenario is shown as a warning.

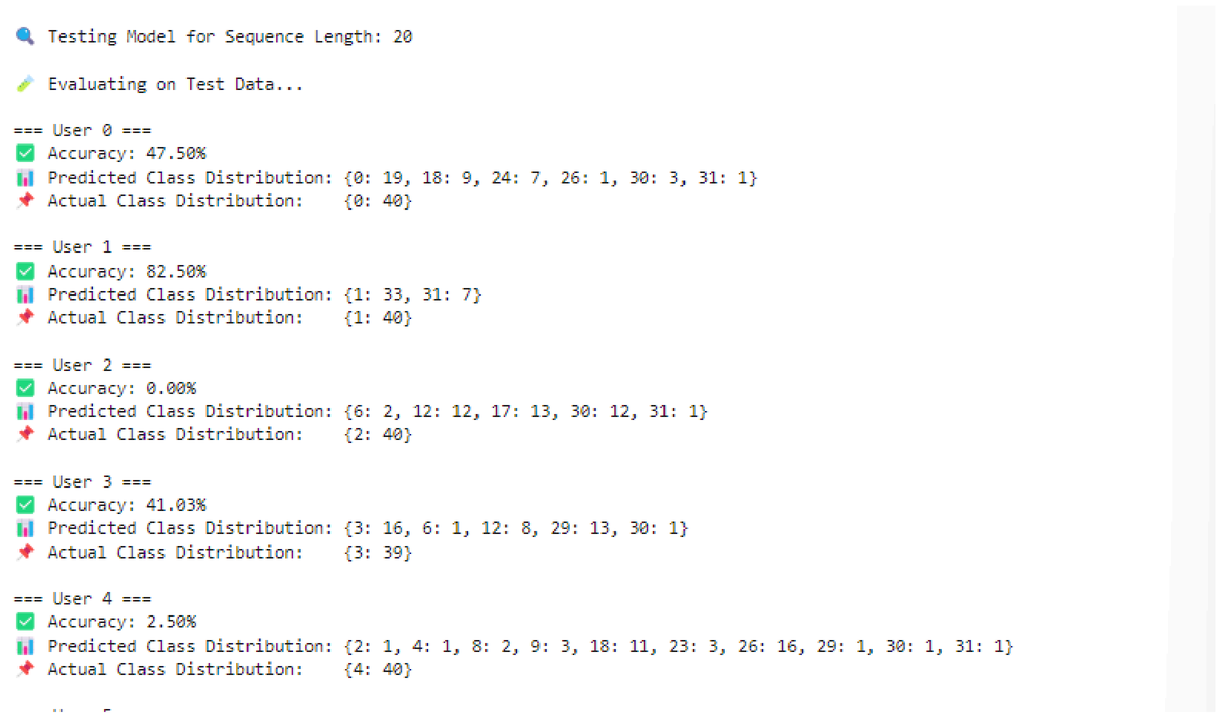


```
=== Testing Scenario Setup ===
⚠️ Only January and February of 2020 were used for testing in predefined setup.
⚠️ Avoid using 2020 data after February due to COVID-19 impact.

Enter test years (comma-separated, e.g., 2020):  2020
Enter months for year 2020 (comma-separated, e.g., 1,2):  1,2
```

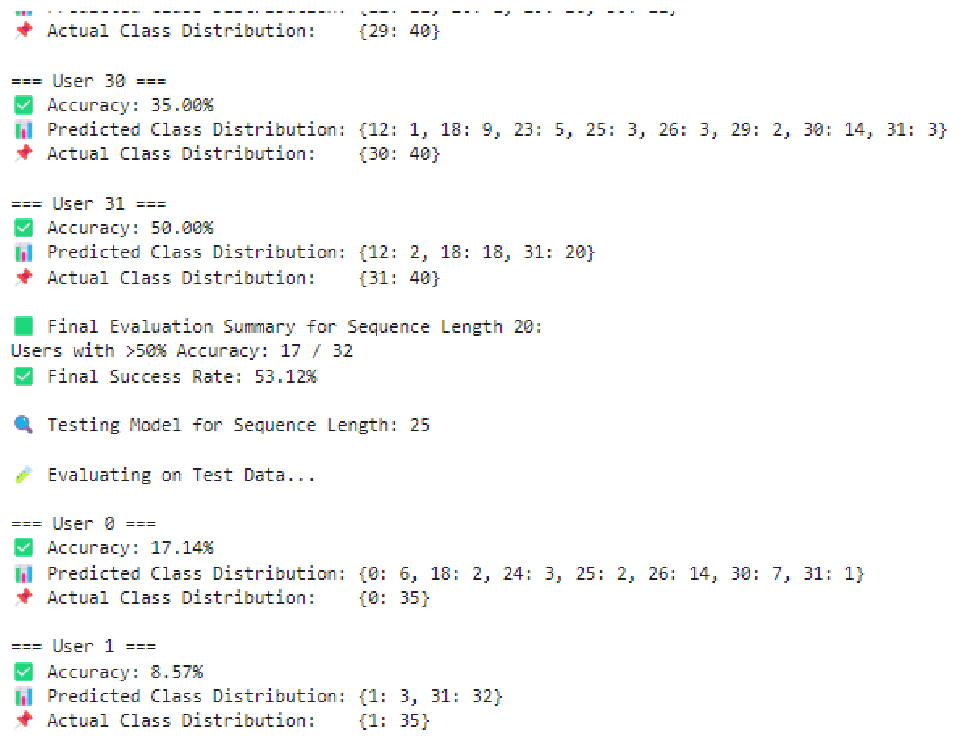
7. After the testing period is entered, a for loop runs which test the model with best parameters for each sequence length. The final result is printed in the terminal and also saved to an excel sheet.

The entire steps for calculating accuracy of each user and final accuracy is also autoamted.

A demo of the expected output is shown below.

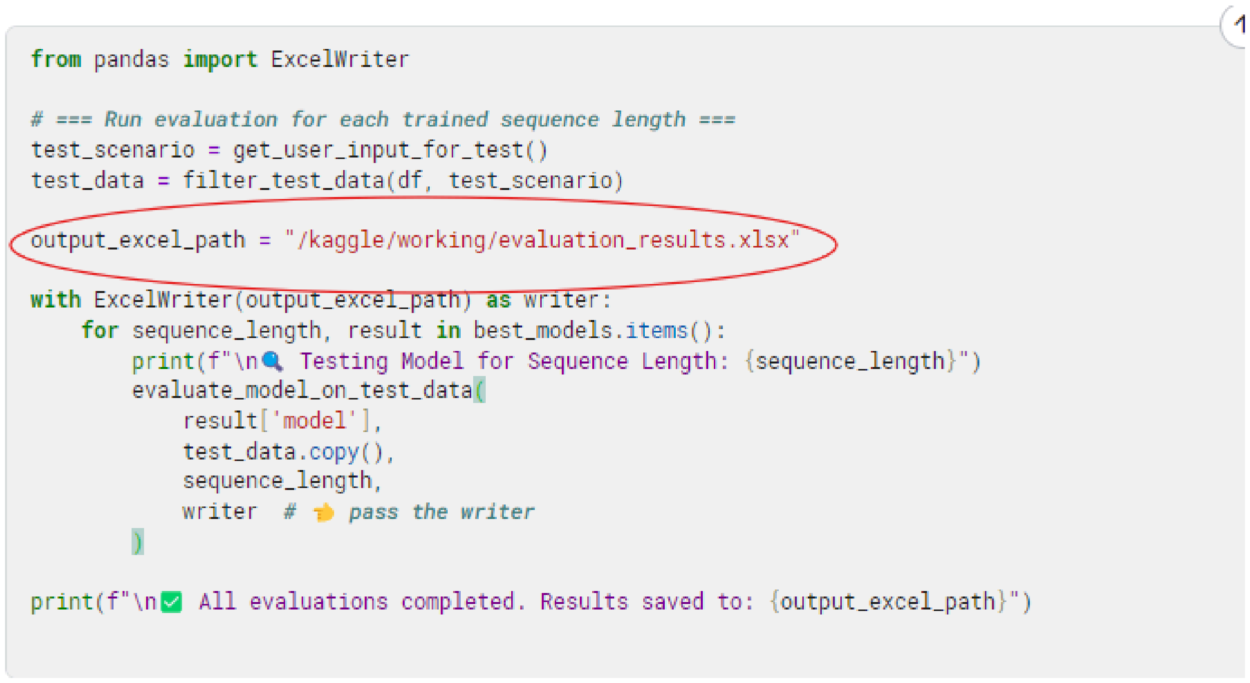


```
🔍 Testing Model for Sequence Length: 20

🧪 Evaluating on Test Data...

=== User 0 ===
✅ Accuracy: 47.50%
📊 Predicted Class Distribution: {0: 19, 18: 9, 24: 7, 26: 1, 30: 3, 31: 1}
📌 Actual Class Distribution:    {0: 40}

=== User 1 ===
✅ Accuracy: 82.50%
📊 Predicted Class Distribution: {1: 33, 31: 7}
📌 Actual Class Distribution:    {1: 40}

=== User 2 ===
✅ Accuracy: 0.00%
📊 Predicted Class Distribution: {6: 2, 12: 12, 17: 13, 30: 12, 31: 1}
📌 Actual Class Distribution:    {2: 40}

=== User 3 ===
✅ Accuracy: 41.03%
📊 Predicted Class Distribution: {3: 16, 6: 1, 12: 8, 29: 13, 30: 1}
📌 Actual Class Distribution:    {3: 39}

=== User 4 ===
✅ Accuracy: 2.50%
📊 Predicted Class Distribution: {2: 1, 4: 1, 8: 2, 9: 3, 18: 11, 23: 3, 26: 16, 29: 1, 30: 1, 31: 1}
📌 Actual Class Distribution:    {4: 40}
```



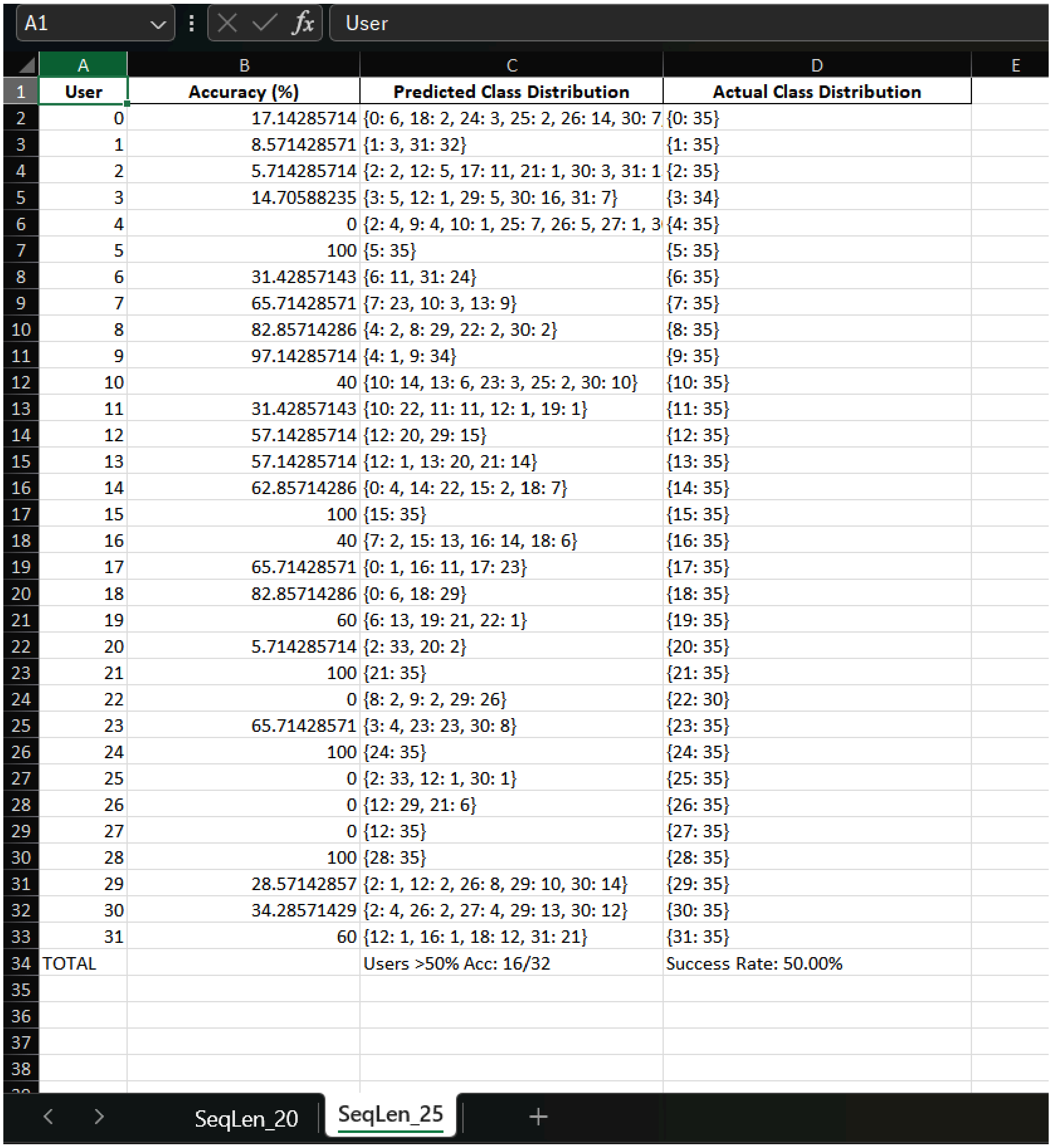
```
📌 Actual Class Distribution:    {29: 40}

=== User 30 ===
✅ Accuracy: 35.00%
📊 Predicted Class Distribution: {12: 1, 18: 9, 23: 5, 25: 3, 26: 3, 29: 2, 30: 14, 31: 3}
📌 Actual Class Distribution:    {30: 40}

=== User 31 ===
✅ Accuracy: 50.00%
📊 Predicted Class Distribution: {12: 2, 18: 18, 31: 20}
📌 Actual Class Distribution:    {31: 40}

🟩 Final Evaluation Summary for Sequence Length 20:
Users with >50% Accuracy: 17 / 32
✅ Final Success Rate: 53.12%

🔍 Testing Model for Sequence Length: 25

🧪 Evaluating on Test Data...

=== User 0 ===
✅ Accuracy: 17.14%
📊 Predicted Class Distribution: {0: 6, 18: 2, 24: 3, 25: 2, 26: 14, 30: 7, 31: 1}
📌 Actual Class Distribution:    {0: 35}

=== User 1 ===
✅ Accuracy: 8.57%
📊 Predicted Class Distribution: {1: 3, 31: 32}
📌 Actual Class Distribution:    {1: 35}
```

8. Select the directory where you want to save the final result.



```python
from pandas import ExcelWriter

# === Run evaluation for each trained sequence length ===
test_scenario = get_user_input_for_test()
test_data = filter_test_data(df, test_scenario)

output_excel_path = "/kaggle/working/evaluation_results.xlsx"

with ExcelWriter(output_excel_path) as writer:
    for sequence_length, result in best_models.items():
        print(f"\n🔍 Testing Model for Sequence Length: {sequence_length}")
        evaluate_model_on_test_data(
            result['model'],
            test_data.copy(),
            sequence_length,
            writer  # 👈 pass the writer
        )

print(f"\n✅ All evaluations completed. Results saved to: {output_excel_path}")
```

The excel output looks like the below screenshot.



User	Accuracy (%)	Predicted Class Distribution	Actual Class Distribution
0	17.14285714	{0: 6, 18: 2, 24: 3, 25: 2, 26: 14, 30: 7	{0: 35}
1	8.571428571	{1: 3, 31: 32}	{1: 35}
2	5.714285714	{2: 2, 12: 5, 17: 11, 21: 1, 30: 3, 31: 1	{2: 35}
3	14.70588235	{3: 5, 12: 1, 29: 5, 30: 16, 31: 7}	{3: 34}
4	0	{2: 4, 9: 4, 10: 1, 25: 7, 26: 5, 27: 1, 3	{4: 35}
5	100	{5: 35}	{5: 35}
6	31.42857143	{6: 11, 31: 24}	{6: 35}
7	65.71428571	{7: 23, 10: 3, 13: 9}	{7: 35}
8	82.85714286	{4: 2, 8: 29, 22: 2, 30: 2}	{8: 35}
9	97.14285714	{4: 1, 9: 34}	{9: 35}
10	40	{10: 14, 13: 6, 23: 3, 25: 2, 30: 10}	{10: 35}
11	31.42857143	{10: 22, 11: 11, 12: 1, 19: 1}	{11: 35}
12	57.14285714	{12: 20, 29: 15}	{12: 35}
13	57.14285714	{12: 1, 13: 20, 21: 14}	{13: 35}
14	62.85714286	{0: 4, 14: 22, 15: 2, 18: 7}	{14: 35}
15	100	{15: 35}	{15: 35}
16	40	{7: 2, 15: 13, 16: 14, 18: 6}	{16: 35}
17	65.71428571	{0: 1, 16: 11, 17: 23}	{17: 35}
18	82.85714286	{0: 6, 18: 29}	{18: 35}
19	60	{6: 13, 19: 21, 22: 1}	{19: 35}
20	5.714285714	{2: 33, 20: 2}	{20: 35}
21	100	{21: 35}	{21: 35}
22	0	{8: 2, 9: 2, 29: 26}	{22: 30}
23	65.71428571	{3: 4, 23: 23, 30: 8}	{23: 35}
24	100	{24: 35}	{24: 35}
25	0	{2: 33, 12: 1, 30: 1}	{25: 35}
26	0	{12: 29, 21: 6}	{26: 35}
27	0	{12: 35}	{27: 35}
28	100	{28: 35}	{28: 35}
29	28.57142857	{2: 1, 12: 2, 26: 8, 29: 10, 30: 14}	{29: 35}
30	34.28571429	{2: 4, 26: 2, 27: 4, 29: 13, 30: 12}	{30: 35}
31	60	{12: 1, 16: 1, 18: 12, 31: 21}	{31: 35}
TOTAL		Users >50% Acc: 16/32	Success Rate: 50.00%