From flask import Flask, render\_template, request, jsonify

Import pandas as pd

Import numpy as np

From sklearn.linear\_model import LinearRegression

Import re

App = Flask(\_\_name\_\_)

# Global months list

MONTHS = [“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”]

# Global dictionary to track state

Chat\_state = {

‘update\_mode’: False,

‘current\_expense\_index’: 0,

‘new\_expenses’: {cat: 0 for cat in [‘Rent’, ‘Utilities’, ‘Groceries’, ‘Transportation’, ‘Entertainment’, ‘Healthcare’, ‘Misc’]},

‘new\_income’: None

}

# Expense categories in order

EXPENSE\_CATEGORIES = [‘Rent’, ‘Utilities’, ‘Groceries’, ‘Transportation’, ‘Entertainment’, ‘Healthcare’, ‘Misc’]

# Load dataset and process it

Def load\_salary\_data(file\_path):

Try:

Df = pd.read\_excel(file\_path)

Except Exception as e:

Return f”Error reading file: {str€}”

Df = calculate\_total\_usage(df)

Return df

# Function to calculate ‘Total\_Usage’ and ‘Savings\_Percentage’

Def calculate\_total\_usage(df):

Expense\_columns = [‘Rent’, ‘Utilities’, ‘Groceries’, ‘Transportation’, ‘Entertainment’, ‘Healthcare’, ‘Misc’]

If ‘Total\_Usage’ not in df.columns:

Df[‘Total\_Usage’] = df[expense\_columns].sum(axis=1)

# Calculate ‘Savings\_Percentage’ if not present

If ‘Savings\_Percentage’ not in df.columns and ‘Savings’ in df.columns:

Df[‘Savings\_Percentage’] = (df[‘Savings’] / df[‘Total\_Usage’]) \* 100

Return df

# Predict usage and suggest savings

Def predict\_usage(df):

Df[‘Previous\_Month\_Usage’] = df[‘Total\_Usage’].shift(1)

Df[‘2\_Months\_Ago\_Usage’] = df[‘Total\_Usage’].shift(2)

# Fill missing values with zero to avoid dropping useful rows

Df.fillna(0, inplace=True)

X = df[[‘Previous\_Month\_Usage’, ‘2\_Months\_Ago\_Usage’]]

Y = df[‘Total\_Usage’]

Model = LinearRegression()

Model.fit(X, y)

Recent\_usage = df.tail(1)[[‘Previous\_Month\_Usage’, ‘2\_Months\_Ago\_Usage’]]

Predicted\_usage = model.predict(recent\_usage)

Return predicted\_usage[0]

# Convert pandas DataFrame to JSON serializable format

Def dataframe\_to\_json(df):

Return df.to\_dict(orient=’records’)

# Generate detailed suggestions for savings and expenses or check spending validity

Def generate\_suggestions(df, predicted\_usage, user\_query):

Global chat\_state

# Handle expense update mode

If chat\_state[‘update\_mode’]:

If chat\_state[‘current\_expense\_index’] >= len(EXPENSE\_CATEGORIES):

# Add new data in the next row without altering previous data

New\_row = df.iloc[-1].copy()

For category, amount in chat\_state[‘new\_expenses’].items():

New\_row[category] = amount

New\_row[‘Total\_Income’] = df[‘Total\_Income’].iloc[-1] # Retain previous month’s income

Df = df.append(new\_row, ignore\_index=True)

Df = calculate\_total\_usage(df)

# Save updated dataset

Try:

Df.to\_excel(‘Dataset.xlsx’, index=False)

Except Exception as e:

Return f”Error saving file: {str€}”

Chat\_state[‘update\_mode’] = False

Chat\_state[‘current\_expense\_index’] = 0

Return “All expenses have been updated. You can now ask for suggestions or predictions.”

Current\_expense = EXPENSE\_CATEGORIES[chat\_state[‘current\_expense\_index’]]

If “ok” in user\_query.lower():

Return f”Please enter the amount for {current\_expense}.”

# Extract category and amount

Amount\_match = re.search(r’\$?(\d+\.?\d\*)’, user\_query)

If amount\_match:

Amount\_found = float(amount\_match.group(1))

Chat\_state[‘new\_expenses’][current\_expense] = amount\_found

Chat\_state[‘current\_expense\_index’] += 1

If chat\_state[‘current\_expense\_index’] < len(EXPENSE\_CATEGORIES):

Next\_expense = EXPENSE\_CATEGORIES[chat\_state[‘current\_expense\_index’]]

Return f”Amount for {current\_expense} saved. Now, please enter the amount for {next\_expense}.”

Else:

Return “All expenses collected. Your data is being updated.”

Return f”Could not understand the amount. Please specify a valid amount for {current\_expense}.”

# Handle income update

If chat\_state[‘new\_income’]:

Amount\_match = re.search(r’\$?(\d+\.?\d\*)’, user\_query)

If amount\_match:

New\_income = float(amount\_match.group(1))

Df.at[df.index[-1], ‘Total\_Income’] = new\_income

Try:

Df.to\_excel(‘Dataset.xlsx’, index=False)

Except Exception as e:

Return f”Error saving file: {str€}”

Chat\_state[‘new\_income’] = None

Return f”Your income has been updated to ${new\_income}.”

Return “Could not understand the amount. Please specify a valid income amount.”

# Not in update mode, handle suggestions and predictions

If “update income” in user\_query.lower():

Chat\_state[‘new\_income’] = True

Return “Enter the amount to be updated for the Total\_Income.”

If “give me all the predicted expenses” in user\_query.lower():

Predicted\_expenses = {cat: df[cat].mean() for cat in EXPENSE\_CATEGORIES} # Placeholder logic for predictions

Return f”The predicted expenses for each category are: {‘, ‘.join([f’{cat}: ${amt:.2f}’ for cat, amt in predicted\_expenses.items()])}.”

If any(keyword in user\_query.lower() for keyword in [“add expense”, “update expenses for the next month”]):

Chat\_state[‘update\_mode’] = True

Chat\_state[‘current\_expense\_index’] = 0

Return f”Let’s update your next month expenses one by one. Please type ‘ok’ to proceed.”

If any(keyword in user\_query.lower() for keyword in [“predicted”, “predicted expenses”, “predictions”, “spend for next month”]):

Return f”The predicted total usage for the next month is ${predicted\_usage:.2f}.”

# Check specific category spending suitability

For category in EXPENSE\_CATEGORIES:

If category.lower() in user\_query.lower():

Try:

Amount = float([word for word in user\_query.split() if word.replace(‘.’, ‘’, 1).isdigit()][0])

Except IndexError:

Return “Could not understand the category or amount. Please specify both.”

Average\_spending = np.mean(df[category])

Latest\_savings\_percentage = df[‘Savings\_Percentage’].tail(1).values[0]

If amount <= average\_spending:

Return f”Actually, it is good to spend ${amount} on {category} next month and it will increase your savings percentage up to {latest\_savings\_percentage:.2f}%.”

Else:

Return f”Sorry, it is not suggested for you to spend ${amount} on {category} next month because typically it will reduce your savings percentage to {latest\_savings\_percentage:.2f}%.”

Return “Could not understand the query. Please ask about spending suitability, predicted usage, income update, or expense details.”

@app.route(‘/’)

Def index():

Return render\_template(‘index.html’)

@app.route(‘/get\_chart\_data’)

Def get\_chart\_data():

Df = load\_salary\_data(‘Dataset.xlsx’)

# Ensure the dataset has a ‘Month’ column

If isinstance(df, str): # Error occurred while reading the file

Return jsonify({‘error’: df})

If ‘Month’ not in df.columns:

Return jsonify({‘error’: “Dataset must contain a ‘Month’ column”})

# Sort dataframe by the ‘Month’ column to ensure correct order

Df[‘Month’] = pd.Categorical(df[‘Month’], categories=MONTHS, ordered=True)

Df = df.sort\_values(‘Month’).reset\_index(drop=True)

Predicted\_usage = predict\_usage(df)

# Extract months and usage for the chart

Months = df[‘Month’].tolist()

Total\_usage = df[‘Total\_Usage’].tolist()

Chart\_data = {

‘months’: months,

‘usage’: total\_usage

}

Return jsonify(chart\_data)

@app.route(‘/chat’, methods=[‘POST’])

Def chat():

User\_message = request.json.get(‘message’)

Df = load\_salary\_data(‘Dataset.xlsx’)

# Check if there’s any issue loading the data

If isinstance(df, str): # Error string returned

Return jsonify({‘response’: df})

Predicted\_usage = predict\_usage(df)

Response = generate\_suggestions(df, predicted\_usage, user\_message)

Return jsonify({‘response’: response})

If \_\_name\_\_ == ‘\_\_main\_\_’:

App.run(debug=True)

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<title>Financial Dashboard & Chatbot</title>

<style>

Body {

Font-family: Arial, sans-serif;

Background: linear-gradient(to bottom right, #a8e6ff, #e6ffff);

Margin: 0;

Padding: 0;

}

.container {

Display: flex;

Flex-wrap: wrap;

Justify-content: space-around;

Padding: 20px;

}

.chart-container, .chat-container {

Background-color: #fff;

Border-radius: 10px;

Padding: 20px;

Box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);

Margin: 10px;

}

.chart-container {

Width: 60%;

Min-width: 300px;

}

.chat-container {

Width: 30%;

Min-width: 300px;

Display: flex;

Flex-direction: column;

Justify-content: space-between;

}

#chart {

Height: 400px;

Width: 100%;

}

.chat-messages {

Height: 350px;

Overflow-y: scroll;

Border: 1px solid #ccc;

Padding: 10px;

Margin-bottom: 10px;

Border-radius: 5px;

}

.chat-input {

Display: flex;

}

Input[type=”text”] {

Width: 100%;

Padding: 10px;

Border: 2px solid #00bcd4;

Border-radius: 5px;

Font-size: 16px;

}

Button {

Background-color: #00bcd4;

Color: #fff;

Border: none;

Padding: 10px 20px;

Margin-left: 10px;

Border-radius: 5px;

Cursor: pointer;

Font-size: 16px;

}

Button:hover {

Background-color: #0097a7;

}

</style>

</head>

<body>

<div class=”container”>

<div class=”chart-container”>

<canvas id=”chart”></canvas>

</div>

<div class=”chat-container”>

<div class=”chat-messages” id=”chat-messages”></div>

<div class=”chat-input”>

<input type=”text” id=”user-input” placeholder=”Ask your financial question…”>

<button id=”send-btn”>Send</button>

</div>

</div>

</div>

<script src=<https://cdn.jsdelivr.net/npm/chart.js>></script>

<script>

Const ctx = document.getElementById(‘chart’).getContext(‘2d’);

Let chart;

Function fetchChartData() {

Fetch(‘/get\_chart\_data’)

.then(response => response.json())

.then(data => {

If (data.error) {

Console.error(data.error);

Return;

}

Const months = data.months;

Const usage = data.usage;

If (chart) {

Chart.destroy();

}

Chart = new Chart(ctx, {

Type: ‘bar’,

Data: {

Labels: months,

Datasets: [{

Label: ‘Total Usage’,

Data: usage,

backgroundColor: ‘rgba(0, 188, 212, 0.5)’,

borderColor: ‘rgba(0, 188, 212, 1)’,

borderWidth: 1

}]

},

Options: {

Responsive: true,

Scales: {

Y: {

beginAtZero: true

}

}

}

});

})

.catch(error => {

Console.error(‘Error fetching chart data:’, error);

});

}

Document.getElementById(‘send-btn’).addEventListener(‘click’, () => {

Const userInput = document.getElementById(‘user-input’).value;

If (userInput.trim() !== ‘’) {

addMessageToChat(‘You’, userInput);

sendMessage(userInput);

document.getElementById(‘user-input’).value = ‘’;

}

});

Function sendMessage(message) {

Fetch(‘/chat’, {

Method: ‘POST’,

Headers: {

‘Content-Type’: ‘application/json’

},

Body: JSON.stringify({ message })

})

.then(response => response.json())

.then(data => {

addMessageToChat(‘Bot’, data.response);

})

.catch(error => {

Console.error(‘Error sending message:’, error);

});

}

Function addMessageToChat(sender, message) {

Const chatMessages = document.getElementById(‘chat-messages’);

Const messageElement = document.createElement(‘div’);

messageElement.textContent = `${sender}: ${message}`;

chatMessages.appendChild(messageElement);

chatMessages.scrollTop = chatMessages.scrollHeight;

}

Window.onload = fetchChartData;

</script>

</body>

</html>