import pandas as pd

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

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

from sklearn.preprocessing import LabelEncoder

from xgboost import XGBClassifier

from sklearn.metrics import classification\_report

import requests

import json

from sqlalchemy import create\_engine

from twilio.rest import Client

WEATHER\_API\_KEY = '563b8d6c849242fb82c135303240111'

TWILIO\_ACCOUNT\_SID = 'AC01fcc98d3ed79edef20435d2a72d09b6'

TWILIO\_AUTH\_TOKEN = '25ed289e6ef394a998d6351ecc95d884'

TWILIO\_PHONE\_NUMBER = '+12407248945'

LOCATION = "Tiruvallur"

agro\_weather\_data = pd.read\_csv(r"C:\Agro-weather project\backend\agro\_weather\_data.csv")

agro\_weather\_data['winddirection'].fillna(agro\_weather\_data['winddirection'].mean())

agro\_weather\_data['windspeed'].fillna(agro\_weather\_data['windspeed'].mean())

label\_encoder = LabelEncoder()

agro\_weather\_data['rainfall'] = label\_encoder.fit\_transform(agro\_weather\_data['rainfall'])

X = agro\_weather\_data.drop(columns=['rainfall'])

y = agro\_weather\_data['rainfall']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

xgb\_model = XGBClassifier(

    objective='binary:logistic',

    learning\_rate=0.05,

    max\_depth=6,

    n\_estimators=1000,

    verbosity=0,

    early\_stopping\_rounds=50,

    use\_label\_encoder=False

)

xgb\_model.fit(X\_train, y\_train, eval\_set=[(X\_train, y\_train), (X\_test, y\_test)], verbose=False)

y\_pred\_proba = xgb\_model.predict\_proba(X\_test)

rainfall\_percentage = y\_pred\_proba[:, 1] \* 100

def fetch\_weather\_data(WEATHER\_API\_KEY, location):

    url = f"http://api.weatherapi.com/v1/forecast.json?key={WEATHER\_API\_KEY}&q={location}&days=1&aqi=yes&alerts=yes"

    try:

        response = requests.get(url)

        response.raise\_for\_status()

        data = response.json()

        return data

    except requests.exceptions.RequestException as e:

        print(f"Error fetching weather data: {e}")

        return None

def get\_advisory(forecast\_data, crop\_type, specific\_hour, rainfall\_prediction\_percentage):

    weather\_condition = forecast\_data['forecast']['forecastday'][0]['hour'][specific\_hour]['condition']['text']

    temperature = forecast\_data['forecast']['forecastday'][0]['hour'][specific\_hour]['temp\_c']

    humidity = forecast\_data['forecast']['forecastday'][0]['hour'][specific\_hour]['humidity']

    advisory\_message = (f"In the next {specific\_hour} hour, weather will be {weather\_condition} "

                        f"The Temperature will be {temperature}°C & Humidity will be {humidity}% "

                        f"predicted rainfall: {rainfall\_prediction\_percentage:.2f}% " )

    if crop\_type == "Rice":

        if temperature > 30:

            advisory\_message += "Irrigate the field to prevent water stress."

        elif humidity < 50:

            advisory\_message += "Consider irrigation to maintain soil moisture."

    elif crop\_type == "Tomatoes":

        if humidity > 70:

            advisory\_message += "Apply fungicides to prevent blight."

    elif crop\_type == "Mangoes":

        if temperature > 35:

            advisory\_message += "Ensure adequate watering to prevent fruit drop."

    elif crop\_type == "Sugarcane":

        if humidity < 60:

            advisory\_message += "Consider irrigation to support growth."

    elif crop\_type == "Okra":

        if temperature > 32:

            advisory\_message += "Water the plants to promote healthy growth."

    elif crop\_type == "Bananas":

        if humidity < 50:

            advisory\_message += "Increase watering to prevent stress."

    elif crop\_type == "Millets":

        if temperature > 30 and humidity < 40:

            advisory\_message += "Provide adequate irrigation."

    elif crop\_type == "Brinjal":

        if humidity > 60:

            advisory\_message += "Watch for pests and apply pesticides if necessary."

    elif crop\_type == "Vegetables":

        if temperature > 28:

            advisory\_message += "Ensure proper watering to keep the plants hydrated."

    elif crop\_type == "Finger Millet":

        if humidity < 50:

            advisory\_message += "Irrigate to maintain soil moisture."

    elif crop\_type == "Groundnut":

        if temperature > 30:

            advisory\_message += "Water the plants to avoid stress."

    elif crop\_type == "Pulses":

        if humidity > 70:

            advisory\_message += "Apply fungicides to prevent diseases."

    elif crop\_type == "Fruits":

        if temperature > 30 and humidity < 50:

            advisory\_message += "Ensure adequate watering to support fruit development."

    else:

        advisory\_message += "No specific advisory available."

    return advisory\_message

def send\_sms(mobile\_numbers, message):

    client = Client(TWILIO\_ACCOUNT\_SID, TWILIO\_AUTH\_TOKEN)

    for mobile\_number in mobile\_numbers:

        try:

            message\_response = client.messages.create(

                body=message,

                from\_=TWILIO\_PHONE\_NUMBER,

                to=mobile\_number

            )

            print(f"Message sent to {mobile\_number}: {message\_response.sid}")

        except Exception as e:

            print(f"Failed to send message to {mobile\_number}: {e}")

engine = create\_engine(r"mysql+pymysql://root:1234@localhost/farmers\_data")

def get\_farmers\_data():

    query = "SELECT mobile\_number, crops\_cultivated FROM farmers"

    farmers\_df = pd.read\_sql(query, engine)

    return farmers\_df

if \_\_name\_\_ == '\_\_main\_\_':

    weather\_data = fetch\_weather\_data(WEATHER\_API\_KEY, LOCATION)

    if weather\_data:

        farmers\_df = get\_farmers\_data()

        specific\_hour = 2

        mobile\_numbers = ['+919940435527', '+917200253877', '+919597194648']

        for i, row in farmers\_df.iterrows():

            mobile\_number = row['mobile\_number']

            rainfall\_prediction\_percentage = rainfall\_percentage.mean()

            advisory\_message = get\_advisory(weather\_data, 'generic crop',specific\_hour, rainfall\_prediction\_percentage)

            print(f"Sent Advisory for {mobile\_number} : {advisory\_message}")

            print(f"Sent Advisory for {mobile\_numbers} : {advisory\_message}")

            #send\_sms(mobile\_number, advisory\_message)

send\_sms(mobile\_numbers, advisory\_message)