**The Climate Crusader code**

**IMPORT PACKAGES**

**API BASED WEATHER FORECAST**

import streamlit as st

import requests

from datetime import datetime, timedelta, timezone

import streamlit.components.v1 as components

# Function to fetch current weather data

def fetch\_weather\_data(city\_name, api\_key):

url =

f"http://api.openweathermap.org/data/2.5/weather?q={

city\_name}&appid={api\_key}"

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

# Function to fetch weather forecast data

def fetch\_weather\_forecast(city\_name, api\_key):

url =

f"http://api.openweathermap.org/data/2.5/forecast?q={

city\_name}&appid={api\_key}&cnt=16"

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

def main():

st.title("Weather App")

# Sidebar for navigation

st.sidebar.title("Navigation")

options = st.sidebar.radio("Select an option",

("Current Weather", "Tomorrow's Weather"))

# API Key

api\_key = "6095197276f832b8beeeb0e40f7d8443" #

Replace with your OpenWeatherMap API key

if options == "Current Weather":

st.header("Current Weather")

city\_name = st.text\_input("Enter City Name",

"London")

if st.button("Get Current Weather"):

weather\_data = fetch\_weather\_data(city\_name,

api\_key)

if weather\_data:

st.subheader("Current Weather Information")

weather\_description =

weather\_data['weather'][0]['description'].capitalize()

st.write(f"Description: {weather\_description}")

temperature = weather\_data['main']['temp'] -

273.15 # Kelvin to Celsius

st.write(f"Temperature: {temperature:.2f} °C")

humidity = weather\_data['main']['humidity']

st.write(f"Humidity: {humidity}%")

wind\_speed = weather\_data['wind']['speed']

st.write(f"Wind Speed: {wind\_speed} m/s")

else:

st.write("Failed to fetch weather data. Please

check the city name.")

elif options == "Tomorrow's Weather":

st.header("Tomorrow's Weather")

city\_name = st.text\_input("Enter City Name",

"London")

if st.button("Get Tomorrow's Weather"):

forecast\_data =

fetch\_weather\_forecast(city\_name, api\_key)

if forecast\_data:

st.subheader("Tomorrow's Weather

Information")

# Find the forecast closest to 24 hours from

now

tomorrow\_date =

(datetime.now(timezone.utc) + timedelta(days=1)).date()

for item in forecast\_data['list']:

forecast\_date =

datetime.strptime(item['dt\_txt'], "%Y-%m-%d

%H:%M:%S").date()

if forecast\_date == tomorrow\_date:

weather\_description =

item['weather'][0]['description'].capitalize()

st.write(f"Description:

{weather\_description}")

temperature = item['main']['temp'] -

273.15 # Kelvin to Celsius

st.write(f"Temperature: {temperature:.2f}

°C")

humidity = item['main']['humidity']

st.write(f"Humidity: {humidity}%")

wind\_speed = item['wind']['speed']

st.write(f"Wind Speed: {wind\_speed}

m/s")

break

else:

st.write("Failed to fetch weather forecast data.

Please check the city name.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**#RANDOM FOREST**

import pandas as pd

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

from sklearn.ensemble import

RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

import joblib

# Load data from CSV file

df = pd.read\_csv('chennai.csv')

# Splitting data into features (X) and target (y)

X = df.drop(['temperature','date'], axis=1)

y = df['temperature']

# Save feature names

feature\_names = X.columns.tolist()

# Splitting data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.2, random\_state=42)

# Initialize the Random Forest model

rf\_model =

RandomForestRegressor(n\_estimators=100,

random\_state=42)

# Train the model

rf\_model.fit(X\_train, y\_train)

# Save the model and feature names

joblib.dump((rf\_model, feature\_names),

'random\_forest\_model.pkl')

# Evaluate the model

y\_pred = rf\_model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Random Forest Mean Squared Error: {mse}")

**#WEB STREAMLIT IMPLEMENTATION**

import streamlit as st

import pandas as pd

import joblib

import requests

import matplotlib.pyplot as plt

# Load the trained model and feature names

model, feature\_names =

joblib.load('random\_forest\_model.pkl')

# Load weather data

@st.cache\_data

def load\_data(file\_path):

return pd.read\_csv(file\_path)

# Define a function to make predictions

def predict\_temperature(input\_data):

input\_df = pd.DataFrame(input\_data)

st.write(f"Debug: Input DataFrame for

prediction:\n{input\_df}")

prediction = model.predict(input\_df)

return prediction[0]

# Define a function to fetch weather data from

OpenWeatherMap API

def fetch\_weather\_data(api\_key, city\_name):

url =

f"http://api.openweathermap.org/data/2.5/weather?q={

city\_name}&appid={api\_key}&units=metric"

response = requests.get(url)

if response.status\_code == 200:

data = response.json()

weather\_data = {

"apparent temperature": data["main"]["temp"],

"humidity": data["main"]["humidity"],

"pressure": data["main"]["pressure"],

"cloud cover": data["clouds"]["all"],

"wind speed": data["wind"]["speed"],

"wind direction": data["wind"]["deg"],

"precipitation": data.get("rain", {}).get("1h", 0) #

Get precipitation in the last hour, default to 0

}

return weather\_data

else:

st.error("Failed to fetch weather data. Please try

again later.")

return None

**# Create the Streamlit app**

def main():

st.title('Climate Change Analysis and Temperature

Prediction')

# User input fields for city name and API key

api\_key = 'fd3f156b7b58c6a2da5cf3cc64354316'

city\_name = st.text\_input('Enter city name:')

# Fetch weather data from OpenWeatherMap API

if city\_name:

weather\_data = fetch\_weather\_data(api\_key,

city\_name)

if weather\_data:

st.subheader('Current Weather Information')

st.write(weather\_data)

# Create input data matching the model's

expected feature names

input\_data = {}

for feature in feature\_names:

if feature in weather\_data:

input\_data[feature] =

[weather\_data[feature]]

else:

input\_data[feature] = [0] # Default value

for missing features

st.write(f"Debug: Prepared Input

Data:\n{input\_data}")

# Predict temperature based on fetched weather

data

if st.button('Predict Temperature'):

if input\_data: # Check if input\_data is not

empty

temperature\_prediction =

predict\_temperature(input\_data)

st.subheader('Predicted Temperature')

st.write(f'{temperature\_prediction:.2f} °C')

else:

st.error("No valid input data available for

prediction.")

else:

st.warning('Please enter a city name.')

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

main()

**#WEATHER ANALYSIS**

import pandas as pd

import matplotlib.pyplot as plt

# Function to load data

def load\_data(file\_path):

data = pd.read\_csv(file\_path)

data['date'] = pd.to\_datetime(data['date']) # Ensure

'date' is in datetime format

return data

# Function to plot climate data

def plot\_climate\_data(weather\_data):

# Plot each column

for column in weather\_data.columns:

if column != 'date': # Exclude the 'date' column

from plotting

plt.figure(figsize=(10, 6))

plt.plot(weather\_data['date'],

weather\_data[column])

plt.title(f'{column} Over Time')

plt.xlabel('Date')

plt.ylabel(column)

plt.xticks(rotation=45) # Rotate x-axis labels for

better readability

plt.grid(True)

plt.tight\_layout()

plt.savefig(f'{column}\_over\_time.png') # Save

each plot as an image

plt.show()

# Main function

def main():

weather\_file\_path = 'Chennai.csv' # Change this to

your weather data file path

weather\_data\_df = load\_data(weather\_file\_path)

if not weather\_data\_df.empty:

plot\_climate\_data(weather\_data\_df)

else:

print("Failed to load climate data.")

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

main()