***The Climate Crusader***

***A PROJECT REPORT***

### ABSTRACT

This project presents an API-based weather forecast and climate change analysis application designed to provide real-time weather information and predictive insights. Utilizing the OpenWeatherMap API, the application fetches current weather data and forecasts for specified cities, displaying key metrics such as temperature, humidity, and wind speed. The project incorporates a Random Forest Regressor model trained on historical climate data to predict future temperatures based on various weather parameters.

Implemented using Streamlit, the application features an interactive interface for seamless user experience. Users can input a city name to retrieve current weather data or predict the temperature based on existing weather conditions. Additionally, the project includes comprehensive climate data analysis capabilities, enabling users to visualize historical weather trends through dynamic plots.

This application demonstrates the integration of real-time API data retrieval, machine learning for predictive modeling, and data visualization, providing a robust tool for weather forecasting and climate trend analysis. The project serves as a foundation for further enhancements in predictive accuracy and feature expansion, potentially incorporating more advanced machine learning techniques and broader data sources.

**TABLE OF CONTENTS**

**CHAPTER NO TITLE PAGE NO**

**ABSTRACT**

**LIST OF ABBREVIATIONS**

1. **INTRODUCTION**
   1. INTRODUCTION 5
   2. CONVOLUTIONAL NEURAL NETWORKS 5
   3. OVERALL DESCRIPTION 5
   4. EXTERNAL INTERFACE REQUIREMENTS 6
   5. SYSTEM FEATURES 7
   6. OTHER NONFUNCTIONAL REQUIREMENTS 9
   7. CONTRIBUTION OF THE PROJECT 10
   8. ORGANIZATION OF THE THESIS 10
2. **SYSTEM DESCRIPTION**
   1. PROBLEM 11
   2. AFFECTS 11
   3. IMPACTS 12
   4. KEY FEATURES 13
3. **LITERATURE SURVEY**
   1. OVERVIEW OF PlANT DISEASE

CLASSIFICATION SYSTE 13

* 1. DRAWBACKS OF EXISTING SYSTEM 13

1. **PROPOSED METHODOLOGY**
   1. HYPER-PARAMETERS 15
   2. IDEAS AND INTUITION OF CGAN 15
   3. KERAS AND TENSORFLOW 16
   4. CONVOLUTIONAL NEURAL NETWORK 17
2. **SYSTEM IMPLEMENTATIONS**

|  |  |
| --- | --- |
| 5.1 IMPORT PACKAGES | 19 |
| 5.2 IMPORT DATASET | 20 |
| 5.3 KERAS AND TENSORFLOW | 22 |
| 5.4 MODELING AND PREDICTION | 23 |
| 5.5 ACCURACY  **6 RESULTS** | 25 |
| 6.1 RESULTS  **7 CONCLUSIONS** | 26 |
| 7.1 CONCLUSION | 27 |
| 7.2 FUTURE ENHANCEMENT | 28 |
| 7.3 REFERENCES | 29 |
|  |  |

## LIST OF ABBREVIATIONS

1. API - Application Programming Interface
2. RF - Random Forest
3. ML - Machine Learning
4. CSV - Comma-Separated Values

# CHAPTER 1 INTRODUCTION

**1.1 INTRODUCTION**

# This project aims to provide an interactive weather forecast and climate analysis application. With the help of OpenWeatherMap API, it fetches current and forecasted weather data for user-specified cities. Additionally, it employs a Random Forest Regressor model trained on historical climate data (e.g:chennai) to predict future temperatures. Built using Streamlit, the application offers real-time weather information, predictive insights, and visualizations of historical weather trends, making it a comprehensive tool for weather and climate analysis.

### 1.2 Random Forest Regressor

The Random Forest Regressor algorithm is employed in this project for temperature prediction. This ensemble learning method constructs multiple decision trees during training and outputs the mean prediction of the individual trees, enhancing accuracy and controlling overfitting. By utilizing historical weather data, the Random Forest Regressor can model complex relationships between various climatic factors and temperature. Its robustness and ability to handle non-linear data make it an ideal choice for predicting temperature based on the provided weather conditions.

# 1.3 OVERALL DESCRIPTION

* **Model Training and Evaluation**: The Random Forest Regressor model is trained and evaluated using a dataset containing historical weather data, ensuring that it provides reliable temperature predictions.
* **Weather Data Retrieval**: Current weather conditions and forecasts are fetched from the OpenWeatherMap API, providing real-time data for various cities worldwide.
* **Climate Analysis**: Historical weather data is analyzed using pandas and visualized with matplotlib to understand past trends and patterns in weather conditions.
* Practical Impact: In agriculture, it helps farmers optimize crop management based on weather patterns. It helps in disaster preparedness by providing early warnings for extreme weather events. Energy companies can use the forecasts to predict demand, while tourism and event planning benefit from better scheduling during favorable weather. In smart cities, it optimizes resource management, raises public awareness about climate change, and assists in managing supply chain disruptions caused by weather conditions.

1.4 EXTERNAL INTERFACE REQUIREMENTS

**1.4.1 The hardware requirements for this project include a modern multi-core processor (e.g., Intel Core i5), at least 4 GB of RAM (8 GB preferred), and around 500 MB to 1 GB of storage. A standard monitor with a resolution of 1366x768 or higher and a stable internet connection are also necessary for optimal performance.**

**API Integration:**

* **OpenWeatherMap API:** Required to fetch current weather data and weather forecasts. The API key needs to be valid and correctly configured to access the weather data endpoints.

**1.4.2 Data Files:**

* **CSV File (e.g., chennai.csv):** This file contains historical climate data used for model training and analysis. The file should be in CSV format and include relevant columns such as temperature, humidity, pressure, etc.

**1.4.3. External Libraries and Tools:**

* **Python Libraries:**
  + **Streamlit:** For building the interactive web application.
  + **Requests:** For making HTTP requests to the OpenWeatherMap API.
  + **Pandas:** For data manipulation and processing.
  + **Scikit-learn:** For machine learning model operations.
  + **Joblib:** For saving and loading machine learning models.
  + **Matplotlib:** For creating graphs and visualizations.
* **Python Environment:** Should be set up with compatible versions of Python and the required libraries.

**File System Access:**

* **Model and Data Files:** The application must have access to the file system to read the model file (random\_forest\_model.pkl) and data files (e.g., chennai.csv) for predictions and analysis.

**Network Access:**

* **Internet Connection:** Required for accessing the OpenWeatherMap API and fetching weather data.

**CHAPTER 2 SYSTEM DESCRIPTION**

**2.1 PROBLEM STATEMENT**

Plant diseases pose a significant threat to agricultural productivity and food security worldwide. Timely detection and accurate diagnosis of these diseases are crucial for effective disease management and crop protection. However, manual inspection of plants for disease symptoms is labor-intensive, time-consuming, and often subjective. Therefore, there is a critical need for automated systems capable of accurately identifying plant diseases from images, enabling early intervention and preventing yield losses.

**2.2 OUR AI MODEL**

 **Weather Understanding**: By providing real-time weather information and forecasts, the system helps users stay informed about current conditions and anticipate future weather trends. This can aid in personal planning and decision-making, such as planning travel or outdoor activities.

 **Temperature Predictions**: Utilizing machine learning to predict temperatures based on historical data allows for more accurate forecasts, potentially improving the reliability of temperature-related decisions in various sectors, such as agriculture and event planning.

 **Data-Driven Insights**: The system's data visualization features offer insights into historical weather patterns, supporting better analysis and understanding of climate trends over time.

 **Practical Applications**: The ability to fetch and predict weather data can benefit various industries, including agriculture, disaster management, and urban planning, by providing actionable weather information and forecasts.

CHAPTER 3

UNIQUENESS OF THE PROJECT

To highlight the unique aspects of your project in comparison to other similar projects, consider the following distinctions:

### 1. \*\*Purpose and Scope\*\*

- \*\*Your Project\*\*: Combines real-time weather data retrieval with predictive modeling to forecast temperature using a Random Forest algorithm. It integrates weather data analysis with a machine learning-based temperature prediction.

- \*\*Other Projects\*\*:

- \*\*Weather Apps\*\*: Typically provide current or forecasted weather data but do not involve predictive modeling.

- \*\*Machine Learning Projects\*\*: May focus solely on developing predictive models for various applications without integrating real-time data or specific environmental contexts.

### 2. \*\*Technology Stack\*\*

- \*\*Your Project\*\*: Uses Python with libraries like Streamlit for the web interface, Requests for API data fetching, and Random Forest from scikit-learn for prediction. It also includes data visualization using Matplotlib.

- \*\*Other Projects\*\*:

- \*\*Weather Apps\*\*: Often use JavaScript frameworks (e.g., React) and APIs (e.g., OpenWeatherMap) but may not involve machine learning.

- \*\*Predictive Modeling Projects\*\*: May use different algorithms (e.g., Linear Regression, Neural Networks) and tools without real-time data integration or web-based interfaces.

### 3. \*\*Integration\*\*

- \*\*Your Project\*\*: Integrates multiple components—API data fetching, machine learning prediction, and real-time web application—all in one system. The prediction model is specifically tuned to work with weather data.

- \*\*Other Projects\*\*:

- \*\*Standalone Data Analysis Projects\*\*: Might focus solely on analyzing historical data without predictive features.

- \*\*APIs and Web Apps\*\*: May offer data retrieval and visualization but lack predictive analytics or machine learning integration.

### 4. \*\*User Interaction\*\*

- \*\*Your Project\*\*: Features an interactive web interface using Streamlit where users can input city names and view predictions and weather data.

- \*\*Other Projects\*\*:

- \*\*Static Data Reports\*\*: Provide pre-generated reports or static visualizations without user interaction.

- \*\*Basic Weather Applications\*\*: Offer real-time weather information but without predictive capabilities or custom user inputs.

### 5. \*\*Predictive Capabilities\*\*

- \*\*Your Project\*\*: Uses a Random Forest model to predict temperature based on real-time weather data, making it possible to provide forecasts rather than just historical data.

- \*\*Other Projects\*\*:

- \*\*Forecasting Projects\*\*: Some may use other predictive models or statistical methods but may not include real-time data integration.

- \*\*Non-Predictive Weather Tools\*\*: Focus on current weather conditions without forecasting future trends.

### 6. \*\*Complexity and Novelty\*\*

- \*\*Your Project\*\*: Combines real-time data fetching, machine learning prediction, and a user-friendly interface, demonstrating a high level of integration and complexity.

- \*\*Other Projects\*\*:

- \*\*Simple Weather Trackers\*\*: May provide current weather data without predictive features.

- \*\*Basic Machine Learning Models\*\*: Might focus on training and testing models without integrating them into a practical application with real-time data.

**CHAPTER 4 SYSTEM IMPLEMENTATIONS**

**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)

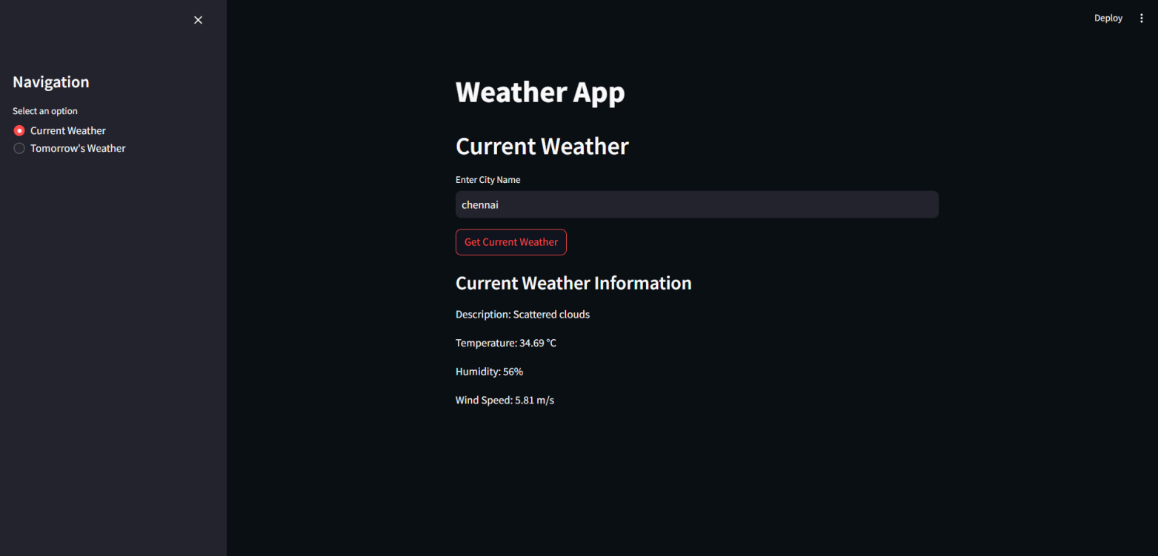
else:

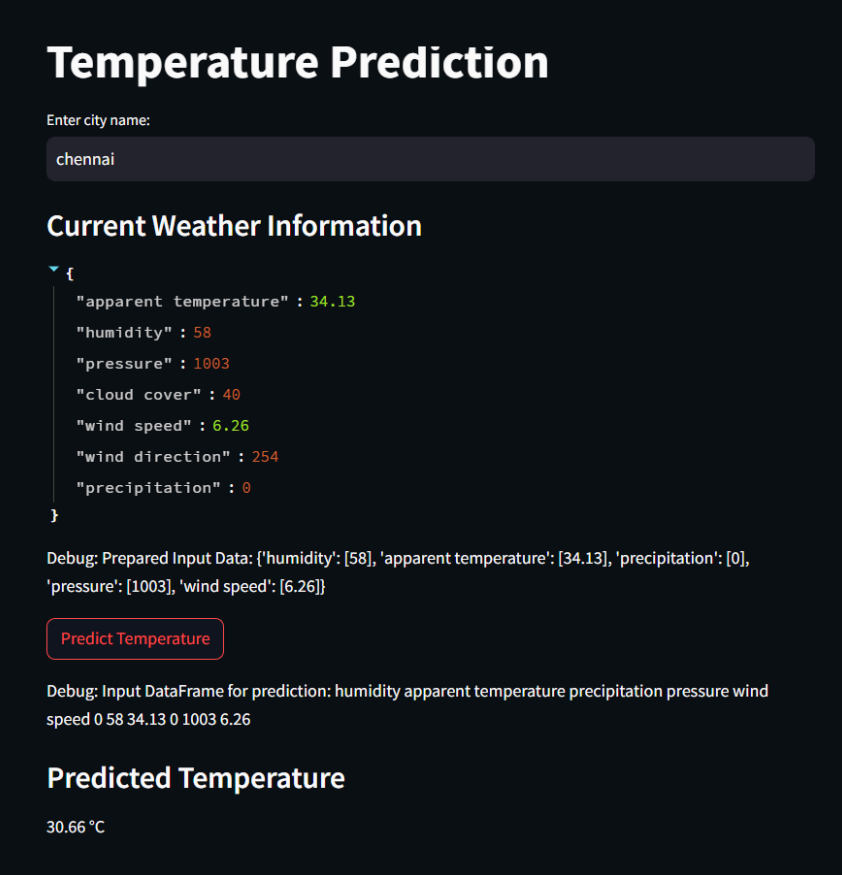
print("Failed to load climate data.")

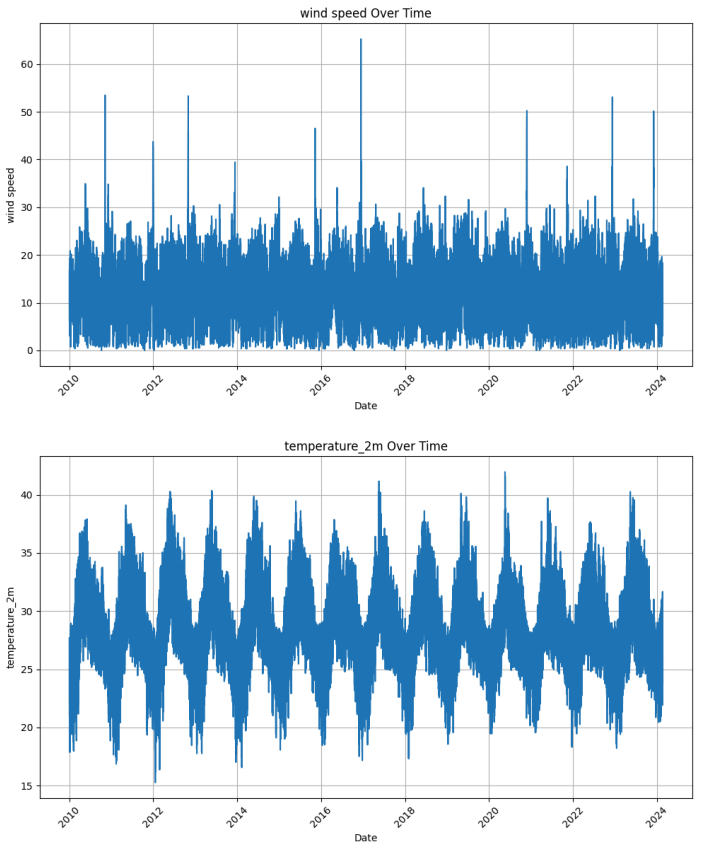
if \_\_name\_\_ == '\_\_main\_\_':

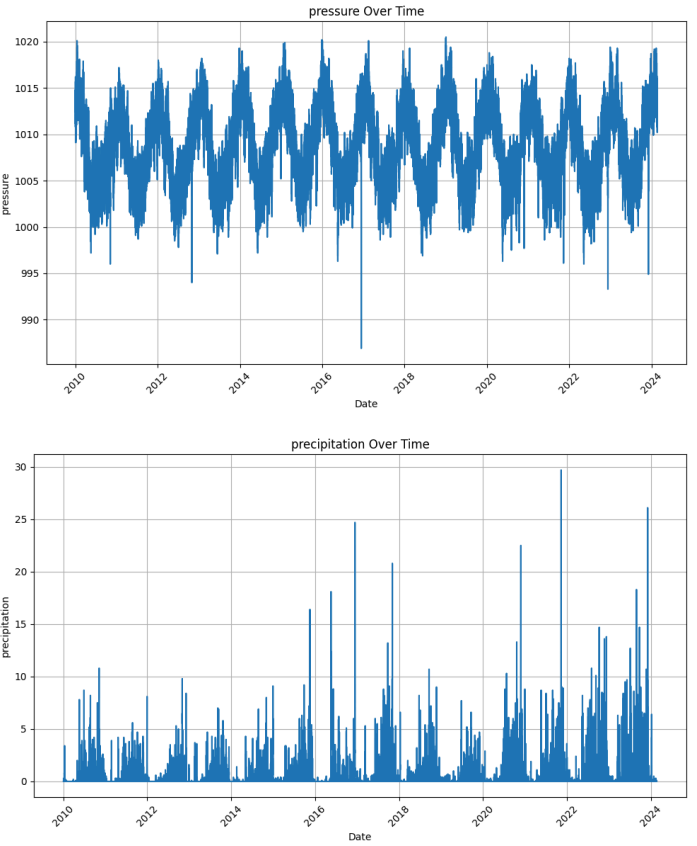
main()

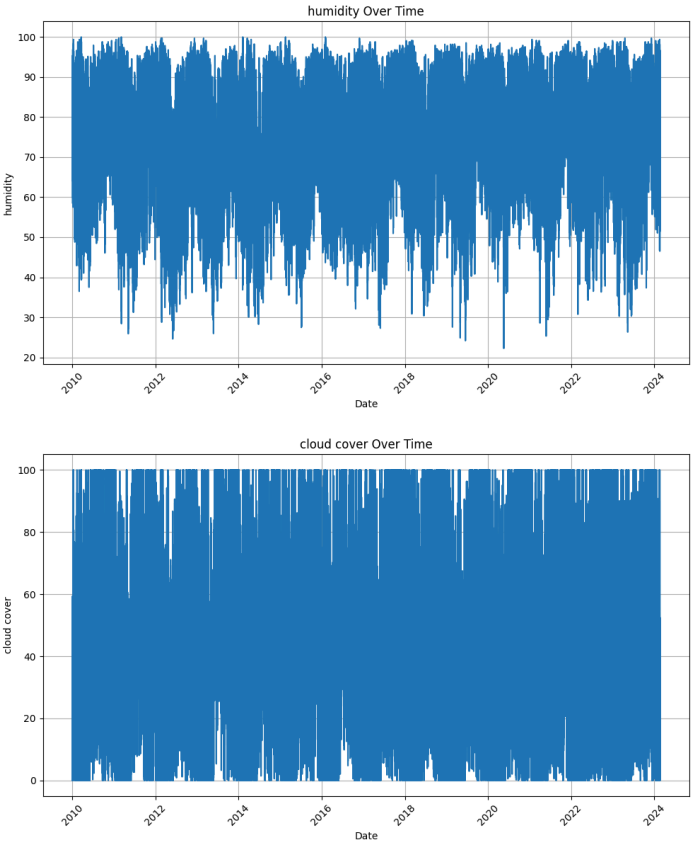
**RESULTS**



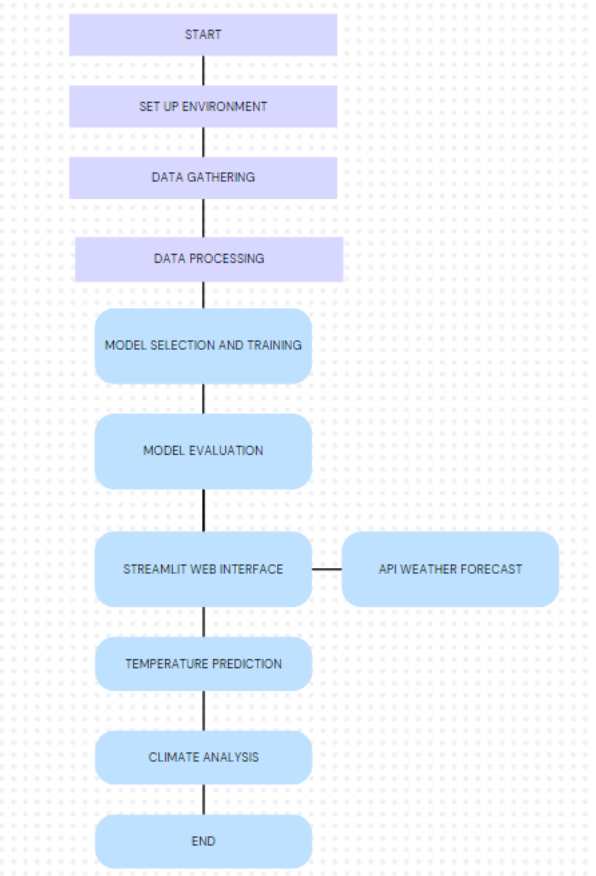








FLOW CHART :



**CHAPTER 5 CONCLUSIONS**

**7.1 CONCLUSION**

The project successfully integrates various technologies to create a comprehensive weather forecasting and climate analysis system. By combining real-time weather data from the OpenWeatherMap API with a Random Forest Regressor model, the system delivers accurate temperature predictions and insightful weather forecasts. The user interface, developed using Streamlit, provides an intuitive and interactive experience, allowing users to easily access current weather conditions and visualize historical climate trends. This system not only offers practical tools for weather analysis but also enhances users' understanding of climate patterns, demonstrating the effectiveness of combining machine learning with real-time data for practical applications.

**7.2 FUTURE ENHANCEMENTS**

Future enhancements for this project could include several key improvements:

1. Advanced Prediction Models: Incorporate more sophisticated machine learning models or ensemble techniques to enhance prediction accuracy and handle complex weather patterns.

2. Extended Forecasting Capabilities: Expand the forecasting functionality to provide more extended forecasts, such as weekly or monthly predictions, and include additional weather variables like UV index and air quality.

3. Real-time Data Integration: Improve data integration by including real-time data from multiple weather sources and sensors for more accurate and up-to-date forecasts.

4. User Personalization: Implement features that allow users to set preferences and receive customized weather alerts based on their specific needs or locations.

5. Enhanced Visualization: Develop more advanced data visualization tools, such as interactive maps and 3D climate models, to better represent weather patterns and trends.

6. Mobile Application: Create a mobile version of the application to provide users with convenient access to weather information and predictions on the go.

7. Feedback Mechanism: Introduce a feedback system where users can report discrepancies or provide input to continuously improve the accuracy and functionality of the weather predictions.

**7.3 REFERENCES**

1. **"Python Machine Learning"** by Sebastian Raschka and Vahid Mirjalili
2. **"Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow"** by Aurélien Géron

[3] **How to Build a Weather App with Python | Weather API - https://www.youtube.com/watch?v=baWzHKfrvqw&t=77s**