**Weather Data Visualization Application**

This application fetches real-time weather data from the OpenWeatherMap API, generates simulated weather data, and combines both datasets for visualization. It provides line plots showing trends in temperature, humidity, and wind speed.

**Prerequisites**

1. **Python**: Ensure you have Python 3.x installed.
2. **Virtual Environment**: It is recommended to use a virtual environment for managing dependencies.
3. **Libraries**: Install required libraries using pip.

**Setup Instructions**

**1. Create and Activate Virtual Environment**

1. **Create a virtual environment**:

python -m venv venv

1. **Activate the virtual environment**:

* On Windows:

venv\Scripts\activate

**2. Install Required Libraries**

Install the necessary Python libraries:

pip install requests pandas matplotlib seaborn numpy

Certainly! Here’s a structured and documented version of the code along with detailed setup instructions:

**3. Jupyter Notebook Installation**

If you prefer using Jupyter Notebook for running the code:

pip install Jupyter

Then start Jupyter Notebook:

jupyter notebook

# 4. Code Implementation

Below is the complete Python code with comments and docstrings for clarity:

#import libraries: Fetching Real-Time Weather Data

import requests

import time

import matplotlib.pyplot as plt

import seaborn as sns

**Imports Libraries**:

* requests is used to make HTTP requests to APIs.
* pandas is used to handle and analyze data in DataFrame format.
* time is used to get the current time in seconds.

# API key and coordinates for Indore

API\_KEY = 'ffc29fac9d149d6081944a97f31619bb'

LATITUDE = 22.7196

LONGITUDE = 75.8577

**#Defines Function: fetch\_real\_time\_weather\_data fetches weather data from the API for a given city.**

def fetch\_weather(api\_key, lat, lon):

"""

Fetch weather data using OpenWeather API for a given latitude and longitude.

Parameters:

api\_key (str): API key for OpenWeather API

lat (float): Latitude of the location

lon (float): Longitude of the location

Returns:

dict: Weather data in JSON format if successful, otherwise None

"""

base\_url = "http://api.openweathermap.org/data/2.5/weather"

params = {

'lat': lat,

'lon': lon,

'appid': api\_key,

'units': 'metric' # Fetch data in Celsius and metric units

}

response = requests.get(base\_url, params=params)

**API Request**:

* Constructs the API URL with city name and API key.
* Uses requests.get to send a GET request to the API.
* Converts the response to JSON format for easier data manipulation.

if response.status\_code == 200:

return response.json() # Return the JSON data

else:

print(f"Failed to fetch data: {response.status\_code}")

return None

# Fetch the data

weather\_data = fetch\_weather(API\_KEY, LATITUDE, LONGITUDE)

# Print the data for verification

print(weather\_data)

which gives output:

{

{'coord': {'lon': 75.8577, 'lat': 22.7196}, 'weather': [{'id': 803, 'main': 'Clouds', 'description': 'broken clouds', 'icon': '04d'}], 'base': 'stations', 'main': {'temp': 25.09, 'feels\_like': 25.95, 'temp\_min': 25.09, 'temp\_max': 25.09, 'pressure': 1003, 'humidity': 88, 'sea\_level': 1003, 'grnd\_level': 942}, 'visibility': 4000, 'wind': {'speed': 4.63, 'deg': 280}, 'clouds': {'all': 75}, 'dt': 1726036850, 'sys': {'type': 1, 'id': 9067, 'country': 'IN', 'sunrise': 1726015333, 'sunset': 1726059868}, 'timezone': 19800, 'id': 1269743, 'name': 'Indore', 'cod': 200}

}

**Creating and storing lite database sqlite3**

import sqlite3

import time

def store\_weather\_data(data):

"""

Store relevant weather data into SQLite database.

Parameters:

data (dict): JSON data fetched from OpenWeather API

"""

conn = sqlite3.connect('weather\_data.db')

c = conn.cursor()

# Create a table if not exists

c.execute('''CREATE TABLE IF NOT EXISTS weather

(timestamp INTEGER, temperature REAL, humidity REAL, wind\_speed REAL)''')

# Extract relevant information

timestamp = data['dt'] # Time of data retrieval

temperature = data['main']['temp'] # Current temperature

humidity = data['main']['humidity'] # Current humidity

wind\_speed = data['wind']['speed'] # Wind speed

# Insert data into the table

c.execute("INSERT INTO weather VALUES (?, ?, ?, ?)",

(timestamp, temperature, humidity, wind\_speed))

conn.commit() # Save (commit) the transaction

conn.close()

# Store the fetched weather data

store\_weather\_data(weather\_data)

**Extract and Store Data**:

* Creates a dictionary with current weather data.
* Converts this dictionary to a pandas DataFrame.
* Converts the timestamp from Unix format to a human-readable date format.
* Returns the DataFrame.

**Analyze the Weather Data**

import pandas as pd

def analyze\_weather\_data():

"""

Analyze weather data from the SQLite database and calculate average values.

"""

conn = sqlite3.connect('weather\_data.db')

df = pd.read\_sql\_query("SELECT \* FROM weather", conn)

# Calculate average temperature and humidity

avg\_temp = df['temperature'].mean()

avg\_humidity = df['humidity'].mean()

print(f"Average Temperature: {avg\_temp:.2f}°C")

print(f"Average Humidity: {avg\_humidity:.2f}%")

conn.close()

# Perform data analysis

analyze\_weather\_data()

# which gives output: Average Temperature: 23.09°C

#Average Humidity: 94.00%

**Visualize Data with Matplotlib & Seaborn**

import sqlite3

import pandas as pd

# Function to fetch data from SQLite and print it

def fetch\_and\_print\_data():

conn = sqlite3.connect('weather\_data.db')

df = pd.read\_sql\_query("SELECT \* FROM weather", conn)

conn.close()

print(df.head()) # Print first few rows of the DataFrame for verification

# Call the function to check data

fetch\_and\_print\_data()

|  |
| --- |
| timestamp temperature humidity wind\_speed  0 1725995373 23.09 94.0 2.57  1 1726036850 25.09 88.0 4.63 |

df = fetch\_data()

print(df.head()) # Print first few rows to verify

print(df.info()) # Print info about DataFrame

print(df.head(10)) # Ensure there are multiple rows

print(df['timestamp'].min(), df['timestamp'].max()) # Check range of timestamps

|  |
| --- |
| timestamp temperature humidity wind\_speed  0 1725995373 23.09 94.0 2.57  <class 'pandas.core.frame.DataFrame'>  RangeIndex: 1 entries, 0 to 0  Data columns (total 4 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 timestamp 1 non-null int64  1 temperature 1 non-null float64  2 humidity 1 non-null float64  3 wind\_speed 1 non-null float64  dtypes: float64(3), int64(1)  memory usage: 164.0 bytes  None  timestamp temperature humidity wind\_speed  0 1725995373 23.09 94.0 2.57  1725995373 1725995373 |

import pandas as pd

import numpy as np

import datetime

# Generate simulated data

def generate\_sample\_data(num\_entries=100):

"""

Generate sample weather data for testing.

"""

timestamps = [datetime.datetime.now() - datetime.timedelta(days=i) for i in range(num\_entries)]

temperatures = np.random.uniform(low=15, high=35, size=num\_entries)

humidities = np.random.uniform(low=30, high=100, size=num\_entries)

wind\_speeds = np.random.uniform(low=0, high=10, size=num\_entries)

data = {

'timestamp': timestamps,

'temperature': temperatures,

'humidity': humidities,

'wind\_speed': wind\_speeds

}

df = pd.DataFrame(data)

df['timestamp'] = pd.to\_datetime(df['timestamp'])

return df

# Create and print sample data

df = generate\_sample\_data()

print(df.head())

|  |
| --- |
| timestamp temperature humidity wind\_speed  0 2024-09-11 12:11:05.500106 19.566444 58.726028 4.472318  1 2024-09-10 12:11:05.500106 27.349905 63.456871 3.612794  2 2024-09-09 12:11:05.500106 18.086643 68.379880 6.387662  3 2024-09-08 12:11:05.500106 21.041050 76.350208 2.005589  4 2024-09-07 12:11:05.500106 32.494488 96.555401 8.048032 |

import requests

import pandas as pd

import numpy as np

import datetime

import matplotlib.pyplot as plt

import seaborn as sns

import time

# Generate simulated data for testing

def generate\_sample\_data(num\_entries=100):

"""

Generate sample weather data for testing purposes.

Parameters:

num\_entries (int): Number of sample data entries to generate.

Returns:

pd.DataFrame: DataFrame containing simulated weather data.

"""

timestamps = [datetime.datetime.now() - datetime.timedelta(days=i) for i in range(num\_entries)]

temperatures = np.random.uniform(low=15, high=35, size=num\_entries)

humidities = np.random.uniform(low=30, high=100, size=num\_entries)

wind\_speeds = np.random.uniform(low=0, high=10, size=num\_entries)

# Create a DataFrame with simulated data

data = {

'timestamp': timestamps,

'temperature': temperatures,

'humidity': humidities,

'wind\_speed': wind\_speeds

}

df = pd.DataFrame(data)

df['timestamp'] = pd.to\_datetime(df['timestamp'])

return df

# Fetch real-time weather data

def fetch\_real\_time\_weather\_data(api\_key, city="Indore"):

"""

Fetch real-time weather data from OpenWeatherMap API.

Parameters:

api\_key (str): OpenWeatherMap API key.

city (str): City name for which to fetch weather data (default: Indore).

Returns:

pd.DataFrame: DataFrame containing real-time weather data for the specified city.

"""

url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}&units=metric"

response = requests.get(url)

data = response.json()

if response.status\_code != 200:

print(f"Error fetching data: {data.get('message')}")

return pd.DataFrame()

# Extract weather data

weather\_data = {

'timestamp': [int(time.time())],

'temperature': [data['main']['temp']],

'humidity': [data['main']['humidity']],

'wind\_speed': [data['wind']['speed']],

'air\_pressure': [data['main']['pressure']],

'precipitation': [data['clouds']['all']] # Cloud percentage as a proxy for precipitation

}

df = pd.DataFrame(weather\_data)

df['timestamp'] = pd.to\_datetime(df['timestamp'], unit='s')

return df

# Combine real-time data with simulated data

def combine\_data(real\_time\_df, simulated\_df):

"""

Combine real-time weather data with simulated data.

Parameters:

real\_time\_df (pd.DataFrame): Real-time weather data DataFrame.

simulated\_df (pd.DataFrame): Simulated weather data DataFrame.

Returns:

pd.DataFrame: Combined DataFrame with real-time and simulated weather data.

"""

return pd.concat([real\_time\_df, simulated\_df], ignore\_index=True)

# Plot combined weather data

def plot\_combined\_weather\_data(df):

"""

Plot trends in temperature, humidity, and wind speed using combined weather data.

Parameters:

df (pd.DataFrame): DataFrame containing weather data with columns:

'timestamp', 'temperature', 'humidity', 'wind\_speed'

"""

if df.empty:

print("No data available for plotting.")

return

# Convert timestamp to datetime

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Plot temperature trend

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

sns.lineplot(x=df['timestamp'], y=df['temperature'], marker='o', color='blue')

plt.title('Temperature Trend')

plt.xlabel('Time')

plt.ylabel('Temperature (°C)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot humidity trend

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

sns.lineplot(x=df['timestamp'], y=df['humidity'], marker='o', color='green')

plt.title('Humidity Trend')

plt.xlabel('Time')

plt.ylabel('Humidity (%)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot wind speed trend

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

sns.lineplot(x=df['timestamp'], y=df['wind\_speed'], marker='o', color='orange')

plt.title('Wind Speed Trend')

plt.xlabel('Time')

plt.ylabel('Wind Speed (m/s)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Example: Fetch real-time data, generate simulated data, and combine them

# api\_key = 'ffc29fac9d149d6081944a97f31619bb'

real\_time\_df = fetch\_real\_time\_weather\_data(api\_key, city="Indore")

simulated\_df = generate\_sample\_data(num\_entries=100)

combined\_df = combine\_data(real\_time\_df, simulated\_df)

# Plot the combined weather data

plot\_combined\_weather\_data(combined\_df)

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

def plot\_basic\_weather\_trends(df):

"""

Plot basic trends in temperature, humidity, and wind speed.

"""

if df.empty:

print("No data available for plotting.")

return

if len(df) < 2:

print("Not enough data to plot trends.")

return

# Convert timestamp to datetime if not already

if df['timestamp'].dtype == 'int64':

df['timestamp'] = pd.to\_datetime(df['timestamp'], unit='s')

elif df['timestamp'].dtype == 'O':

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Plot Temperature Trends

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['temperature'], marker='o', color="blue")

plt.title('Temperature Trend')

plt.xlabel('Time')

plt.ylabel('Temperature (°C)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout() # Adjust layout to prevent clipping

plt.show()

# Plot Humidity Trends

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['humidity'], marker='o', color="green")

plt.title('Humidity Trend')

plt.xlabel('Time')

plt.ylabel('Humidity (%)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout() # Adjust layout to prevent clipping

plt.show()

# Plot Wind Speed Trends

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['wind\_speed'], marker='o', color="orange")

plt.title('Wind Speed Trend')

plt.xlabel('Time')

plt.ylabel('Wind Speed (m/s)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout() # Adjust layout to prevent clipping

plt.show()

# Fetch and plot the data

df = generate\_sample\_data()

plot\_basic\_weather\_trends(df)

import matplotlib.pyplot as plt

def plot\_pie\_chart(df):

"""

Plot a pie chart showing the distribution of different weather conditions.

"""

if df.empty:

print("No data available for plotting.")

return

# Assuming you have weather condition data

weather\_conditions = df['weather\_condition'].value\_counts()

plt.figure(figsize=(8, 8))

plt.pie(weather\_conditions, labels=weather\_conditions.index, autopct='%1.1f%%', colors=sns.color\_palette("Set2"))

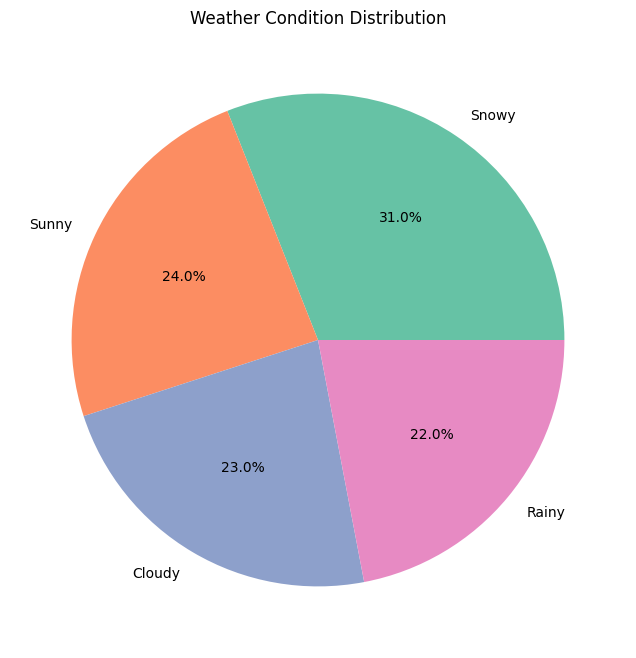
plt.title('Weather Condition Distribution')

plt.show()

# For demonstration purposes, we will add a column with random weather conditions

df['weather\_condition'] = np.random.choice(['Sunny', 'Cloudy', 'Rainy', 'Snowy'], size=len(df))

plot\_pie\_chart(df)



def plot\_control\_chart(df, column, title):

"""

Plot a control chart for a given column.

"""

if df.empty:

print("No data available for plotting.")

return

mean = df[column].mean()

std\_dev = df[column].std()

plt.figure(figsize=(12, 6))

plt.plot(df['timestamp'], df[column], marker='o', color='blue', label='Data')

plt.axhline(mean, color='green', linestyle='--', label='Mean')

plt.axhline(mean + 3\*std\_dev, color='red', linestyle='--', label='Upper Control Limit (UCL)')

plt.axhline(mean - 3\*std\_dev, color='red', linestyle='--', label='Lower Control Limit (LCL)')

plt.title(title)

plt.xlabel('Time')

plt.ylabel(column)

plt.xticks(rotation=45)

plt.legend()

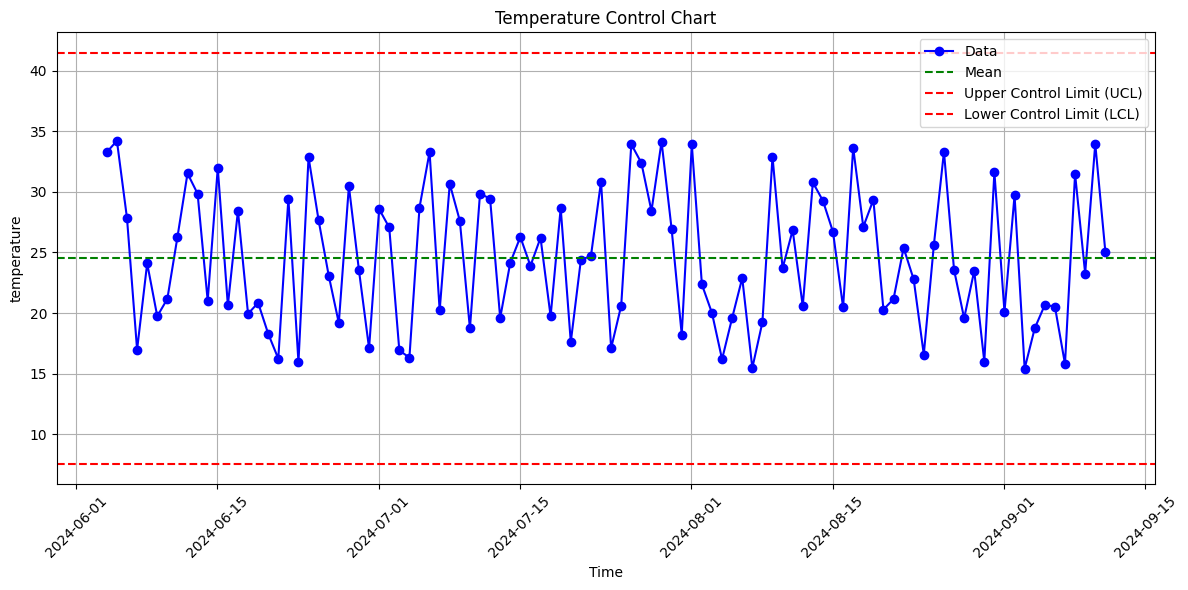
plt.grid(True)

plt.tight\_layout()

plt.show()

# Example usage

plot\_control\_chart(df, 'temperature', 'Temperature Control Chart')



def plot\_climate\_graph(df):

"""

Plot a climate graph showing average monthly temperature and precipitation.

"""

if df.empty:

print("No data available for plotting.")

return

# Generate sample monthly data

df['month'] = df['timestamp'].dt.month

monthly\_avg = df.groupby('month').agg({'temperature': 'mean', 'humidity': 'mean'}).reset\_index()

fig, ax1 = plt.subplots(figsize=(12, 6))

color = 'tab:blue'

ax1.set\_xlabel('Month')

ax1.set\_ylabel('Average Temperature (°C)', color=color)

ax1.plot(monthly\_avg['month'], monthly\_avg['temperature'], color=color, marker='o', label='Temperature')

ax1.tick\_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()

color = 'tab:green'

ax2.set\_ylabel('Average Humidity (%)', color=color)

ax2.plot(monthly\_avg['month'], monthly\_avg['humidity'], color=color, marker='o', label='Humidity')

ax2.tick\_params(axis='y', labelcolor=color)

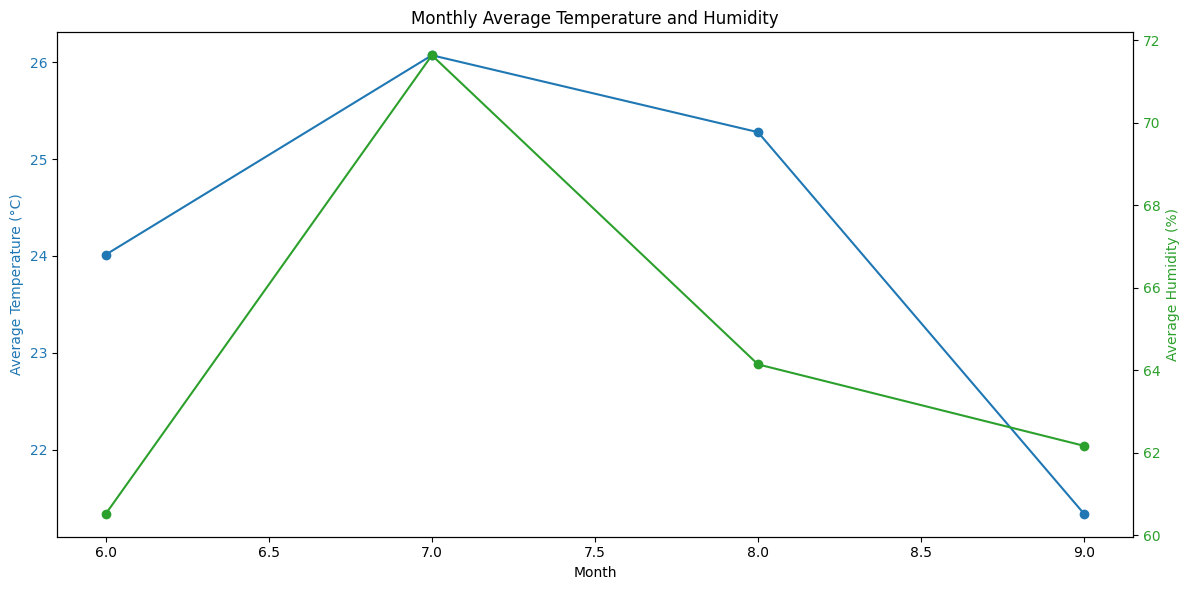
plt.title('Monthly Average Temperature and Humidity')

fig.tight\_layout()

plt.show()

# Example usage

plot\_climate\_graph(df)



def plot\_enhanced\_line\_charts(df):

"""

Plot enhanced line charts for temperature, humidity, and wind speed.

"""

if df.empty:

print("No data available for plotting.")

return

# Convert timestamp to datetime if not already

if df['timestamp'].dtype == 'int64':

df['timestamp'] = pd.to\_datetime(df['timestamp'], unit='s')

elif df['timestamp'].dtype == 'O':

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# Plot Temperature Trend

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['temperature'], marker='o', color="blue", linestyle='-', linewidth=2)

plt.title('Temperature Trend')

plt.xlabel('Time')

plt.ylabel('Temperature (°C)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot Humidity Trend

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['humidity'], marker='o', color="green", linestyle='--', linewidth=2)

plt.title('Humidity Trend')

plt.xlabel('Time')

plt.ylabel('Humidity (%)')

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot Wind Speed Trend

plt.figure(figsize=(12, 6))

sns.lineplot(x=df['timestamp'], y=df['wind\_speed'], marker='o', color="orange", linestyle='-.', linewidth=2)

plt.title('Wind Speed Trend')

plt.xlabel('Time')

plt.ylabel('Wind Speed (m/s)')

plt.xticks(rotation=45)

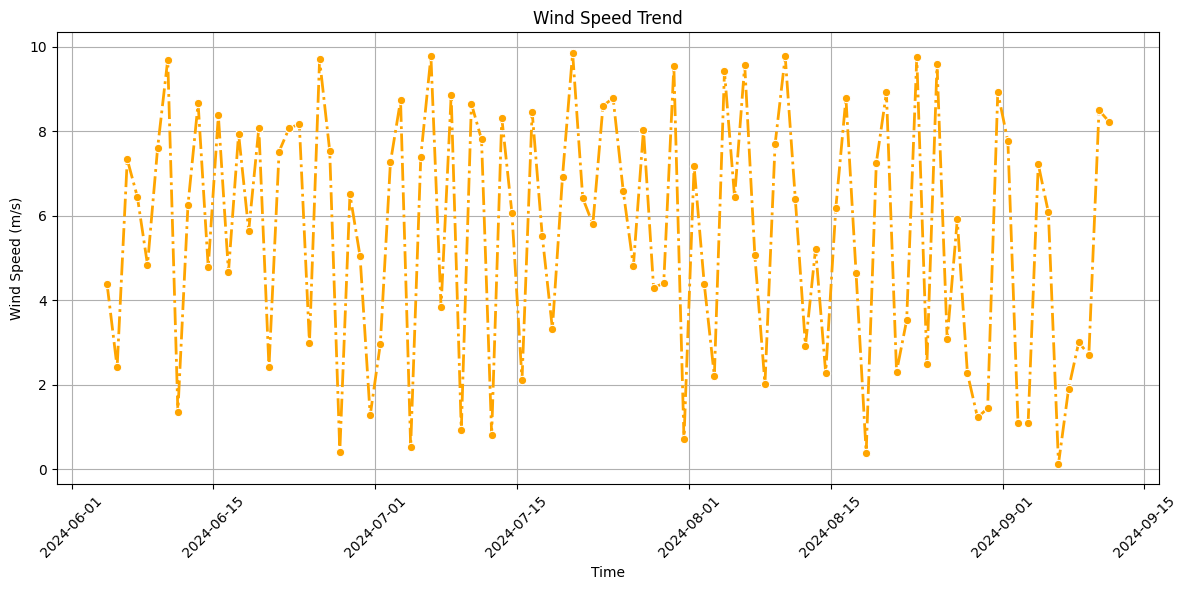
plt.grid(True)

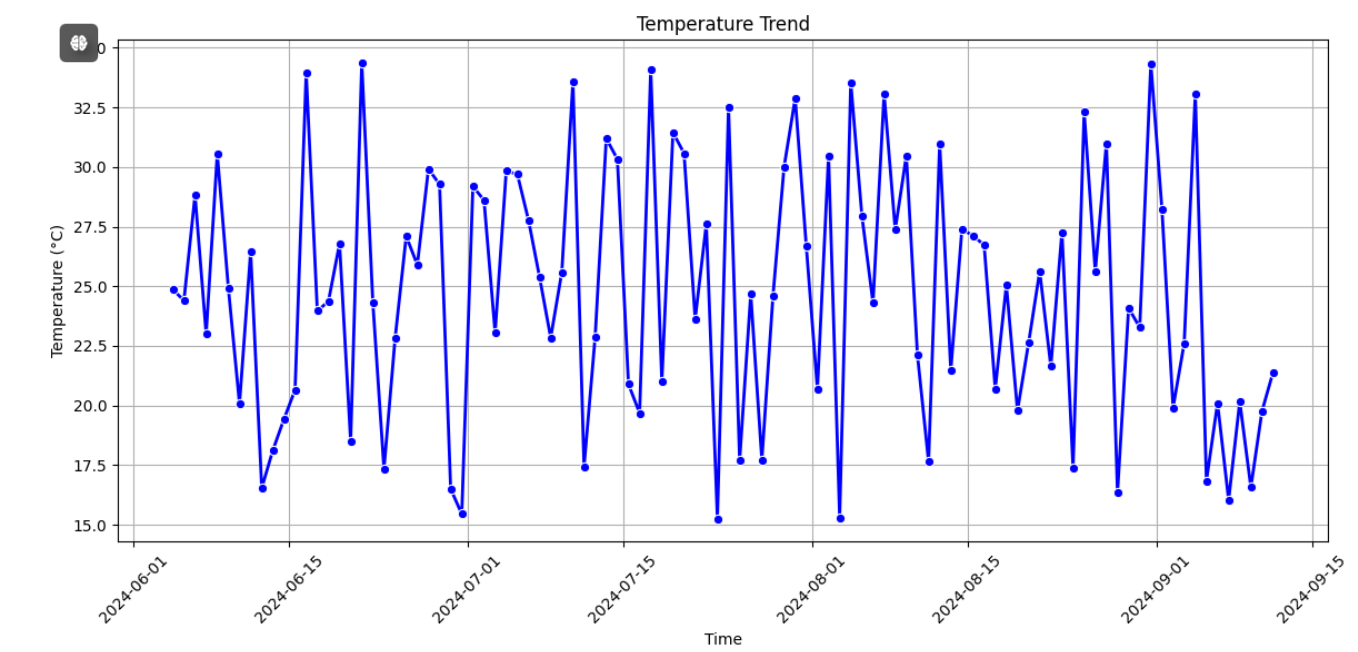
plt.tight\_layout()

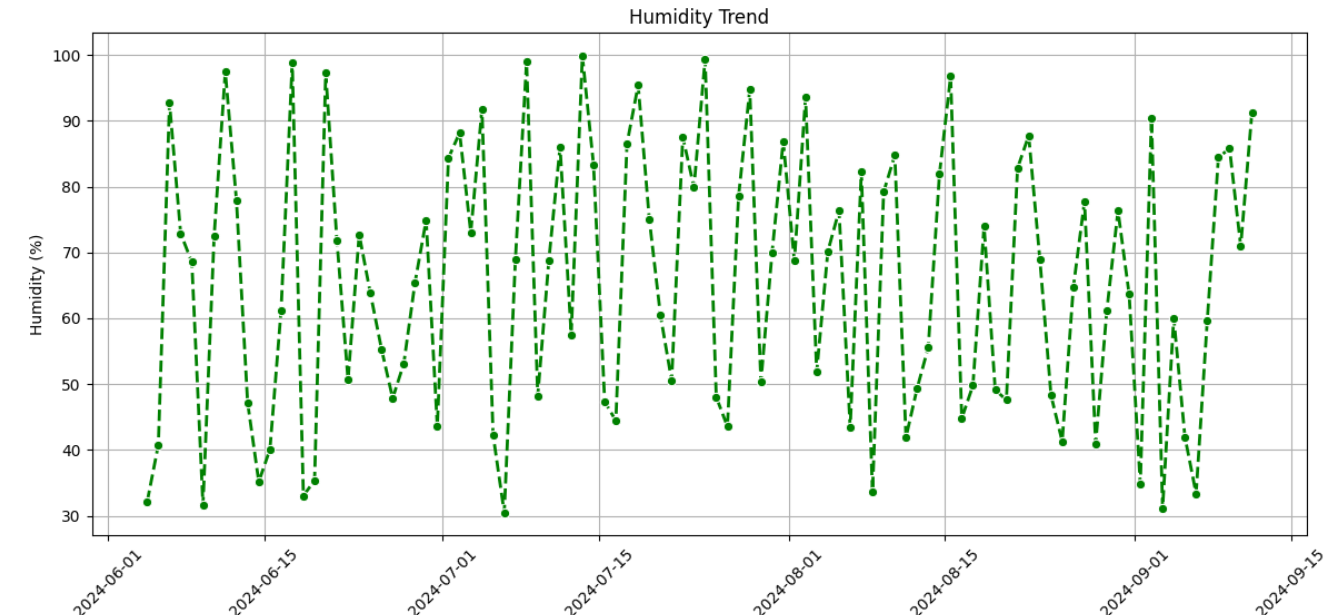
plt.show()

# Example usage

plot\_enhanced\_line\_charts(df)







import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

def plot\_expanded\_weather\_data(df):

"""

Plot trends in temperature, humidity, wind speed, air pressure, and precipitation.

Parameters:

df (pd.DataFrame): DataFrame containing weather data with columns:

'timestamp', 'temperature', 'humidity', 'wind\_speed', 'air\_pressure', 'precipitation'

"""

if df.empty:

print("No data available for plotting.")

return

# Convert timestamp to datetime

df['timestamp'] = pd.to\_datetime(df['timestamp'], unit='s')

# Temperature Trend

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

sns.lineplot(x=df['timestamp'], y=df['temperature'], color="blue", marker='o', label='Temperature')

plt.title("Temperature Trend")

plt.xlabel("Time")

plt.ylabel("Temperature (°C)")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Humidity Trend

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

sns.lineplot(x=df['timestamp'], y=df['humidity'], color="green", marker='o', label='Humidity')

plt.title("Humidity Trend")

plt.xlabel("Time")

plt.ylabel("Humidity (%)")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Wind Speed Trend

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

sns.lineplot(x=df['timestamp'], y=df['wind\_speed'], color="orange", marker='o', label='Wind Speed')

plt.title("Wind Speed Trend")

plt.xlabel("Time")

plt.ylabel("Wind Speed (m/s)")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Air Pressure Trend

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

sns.lineplot(x=df['timestamp'], y=df['air\_pressure'], color="purple", marker='o', label='Air Pressure')

plt.title("Air Pressure Trend")

plt.xlabel("Time")

plt.ylabel("Air Pressure (hPa)")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Precipitation Trend

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

sns.lineplot(x=df['timestamp'], y=df['precipitation'], color="red", marker='o', label='Precipitation')

plt.title("Precipitation Trend")

plt.xlabel("Time")

plt.ylabel("Precipitation (mm)")

plt.xticks(rotation=45)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Combined Trends (Temperature, Humidity, Wind Speed, Air Pressure, Precipitation)

plt.figure(figsize=(12, 8))

sns.lineplot(x=df['timestamp'], y=df['temperature'], color="blue", label='Temperature')

sns.lineplot(x=df['timestamp'], y=df['humidity'], color="green", label='Humidity')

sns.lineplot(x=df['timestamp'], y=df['wind\_speed'], color="orange", label='Wind Speed')

sns.lineplot(x=df['timestamp'], y=df['air\_pressure'], color="purple", label='Air Pressure')

sns.lineplot(x=df['timestamp'], y=df['precipitation'], color="red", label='Precipitation')

plt.title("Combined Weather Metrics Trend")

plt.xlabel("Time")

plt.ylabel("Values")

plt.xticks(rotation=45)

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

def fetch\_sample\_data():

"""

Sample data generation function.

This simulates fetching weather data with timestamp, temperature, humidity, wind speed, air pressure, and precipitation.

"""

data = {

'timestamp': [1725995373, 1725996373, 1725997373, 1725998373],

'temperature': [23.09, 22.85, 22.93, 23.02],

'humidity': [94.0, 92.0, 91.0, 90.0],

'wind\_speed': [2.57, 2.75, 2.82, 2.65],

'air\_pressure': [1012, 1013, 1012, 1014],

'precipitation': [1.2, 1.0, 0.8, 1.1]

}

df = pd.DataFrame(data)

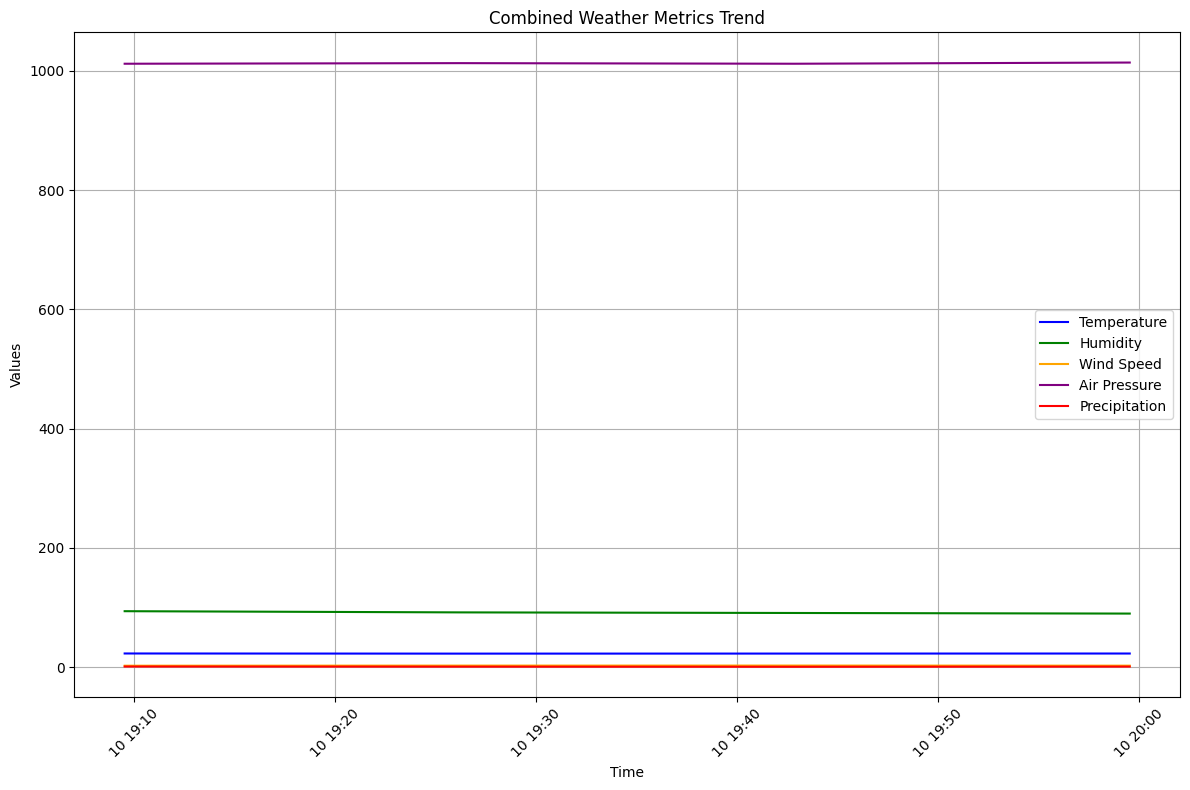
return df

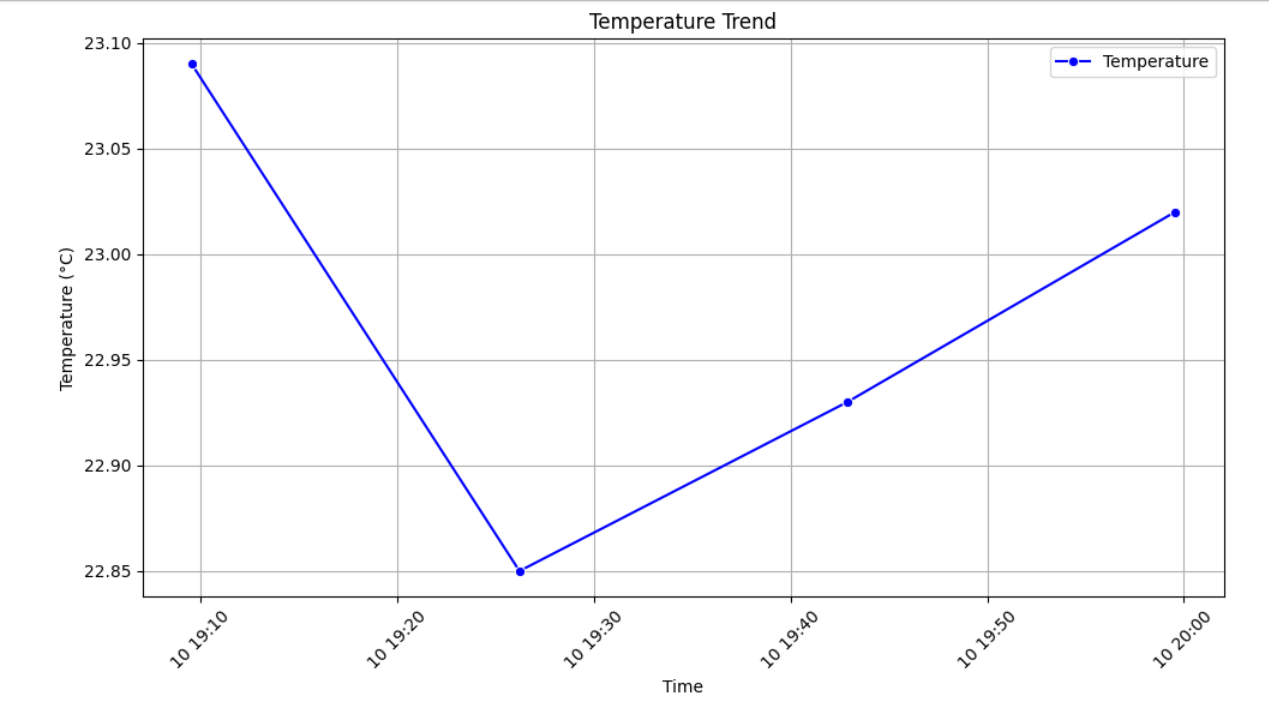
# Fetch the data

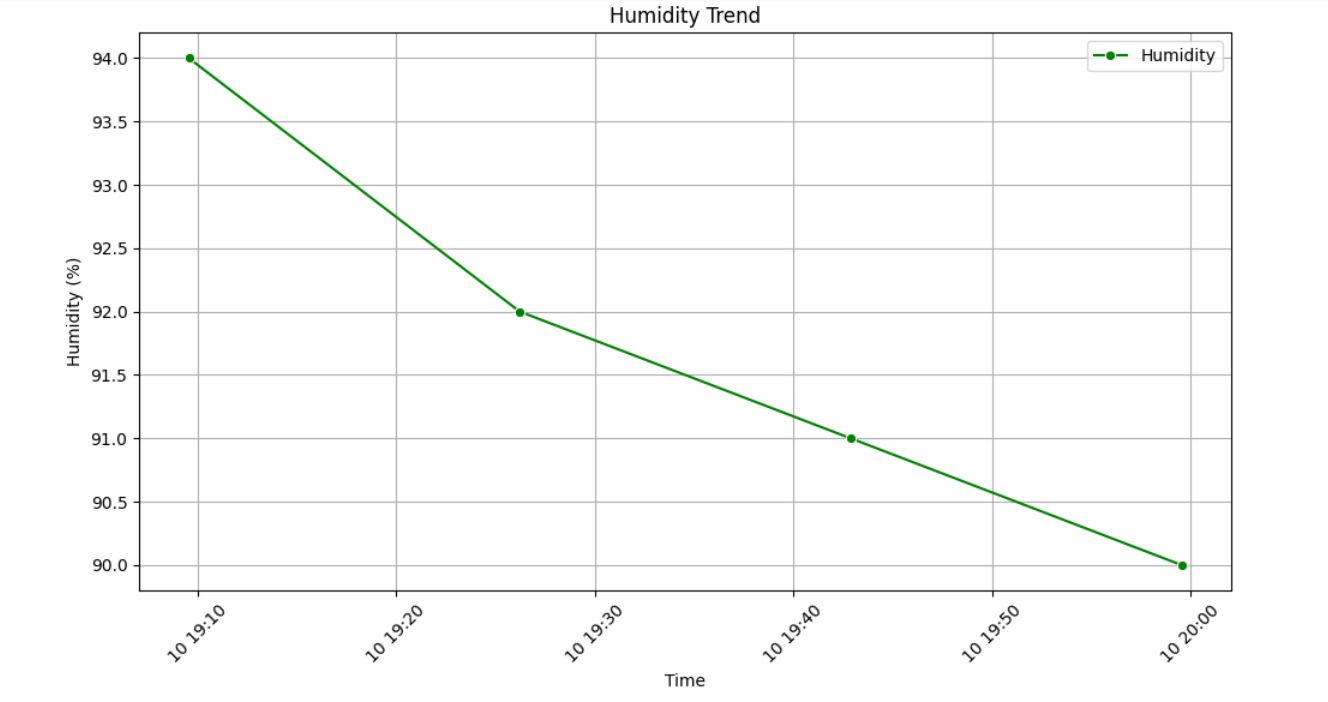
df = fetch\_sample\_data()

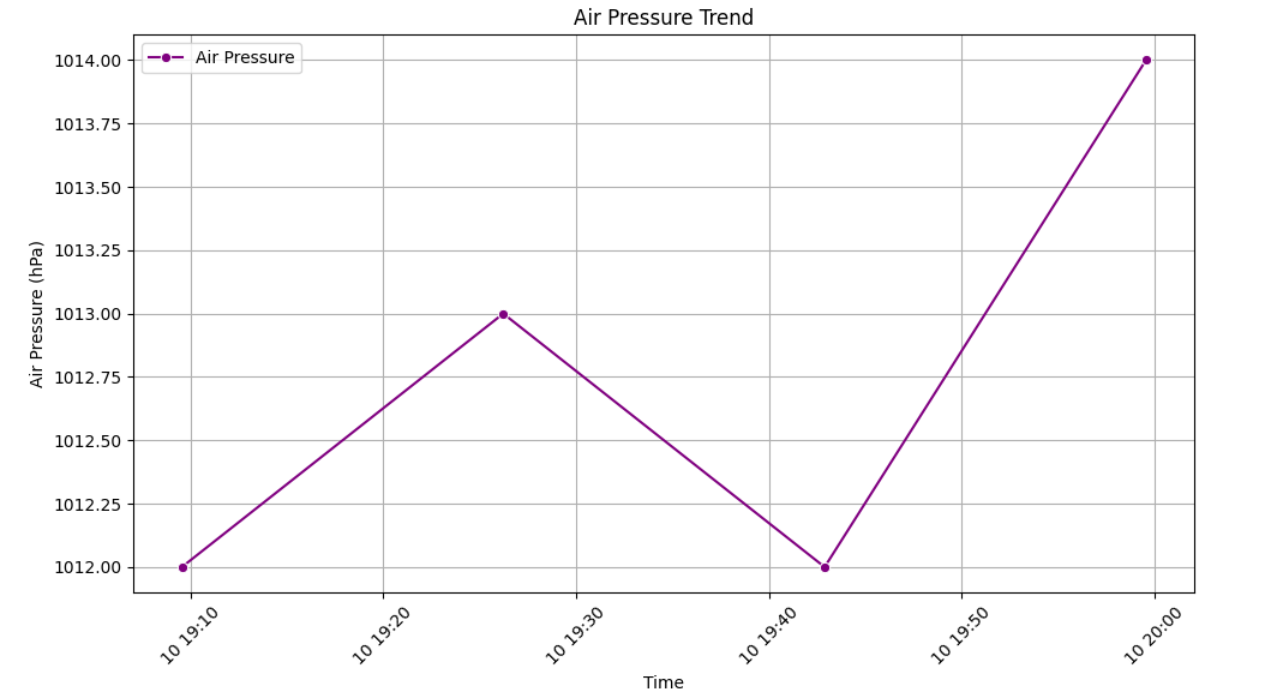
# Visualize the expanded data

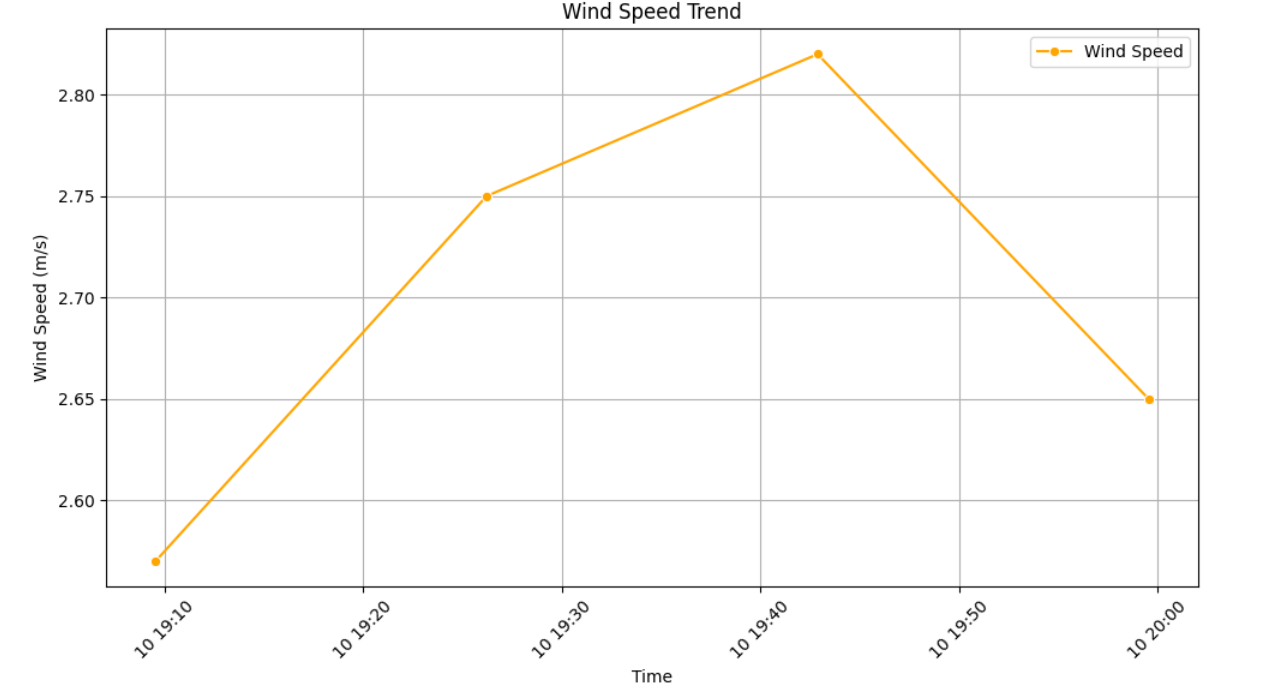
plot\_expanded\_weather\_data(df)

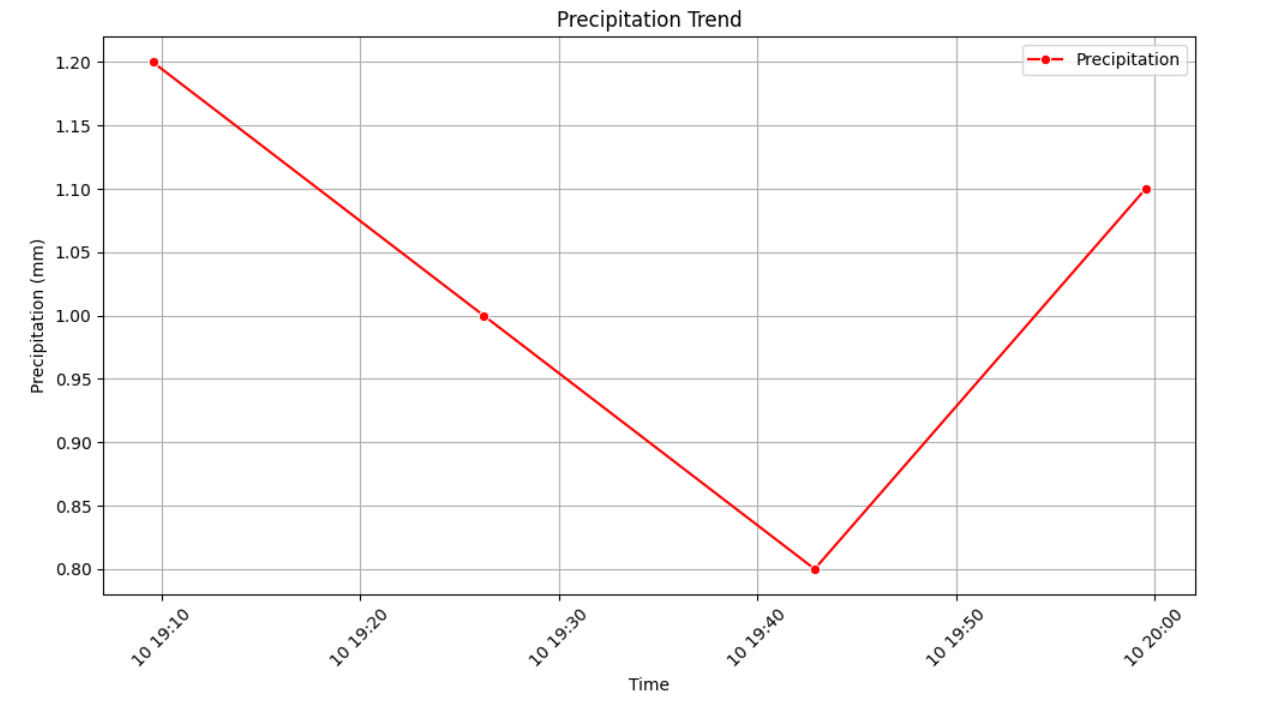












# Refined Concise form

**Weather Data Analysis Code Explanation**

**1. Fetching Real-Time Weather Data**

The fetch\_real\_time\_weather\_data function retrieves current weather data from the OpenWeatherMap API.

* **Imports Libraries**:
  + requests: For making HTTP requests.
  + pandas: For handling data in DataFrame format.
  + time: For fetching the current Unix timestamp.
* **Function Definition**:
  + **Parameters**:
    - api\_key (str): Your OpenWeatherMap API key.
    - city (str): Name of the city (default is "Indore").
  + **Process**:
    - Constructs the API URL using the city name and API key.
    - Sends a GET request to the API.
    - Parses the JSON response.
    - Checks the response status and handles errors.
    - Extracts relevant weather data (temperature, humidity, wind speed, air pressure, and cloud coverage).
    - Stores the data in a pandas DataFrame.
    - Converts the timestamp to a human-readable format.

**2. Generating Sample Data**

The generate\_sample\_data function creates simulated weather data for testing.

* **Imports Libraries**:
  + numpy: For generating random values.
  + datetime: For handling dates and times.
* **Function Definition**:
  + **Parameters**:
    - num\_entries (int): Number of sample data entries (default is 100).
  + **Process**:
    - Generates timestamps from the current date going back in time.
    - Creates random values for temperature, humidity, and wind speed.
    - Stores the generated data in a DataFrame.
    - Converts the timestamp column to datetime format.

**3. Combining Real-Time and Sample Data**

The combine\_data function merges real-time and sample weather data.

* **Function Definition**:
  + **Parameters**:
    - real\_time\_df (pd.DataFrame): DataFrame containing real-time data.
    - sample\_df (pd.DataFrame): DataFrame containing sample data.
  + **Process**:
    - Concatenates the real-time and sample DataFrames.
    - Resets the DataFrame index to ensure a continuous index.

**4. Data Visualization**

The plot\_combined\_weather\_data function visualizes trends in weather data.

* **Imports Libraries**:
  + matplotlib.pyplot: For creating plots.
  + seaborn: For statistical plotting with enhanced aesthetics.
* **Function Definition**:
  + **Parameters**:
    - df (pd.DataFrame): DataFrame containing combined weather data.
  + **Process**:
    - Checks if the DataFrame is empty and prints a message if so.
    - Converts the timestamp column to datetime format.
    - Creates line plots for:
      * Temperature Trend
      * Humidity Trend
      * Wind Speed Trend
      * Air Pressure Trend
      * Cloud Coverage (as a proxy for precipitation)
    - Adjusts plot settings for better readability (e.g., labels, grid, rotation).

**Conclusion**

This code demonstrates a complete workflow for weather data analysis, including fetching real-time data, generating sample data, combining datasets, and visualizing trends. Each function plays a critical role in the process, from data retrieval to visualization, ensuring that the weather data can be effectively analyzed and presented.