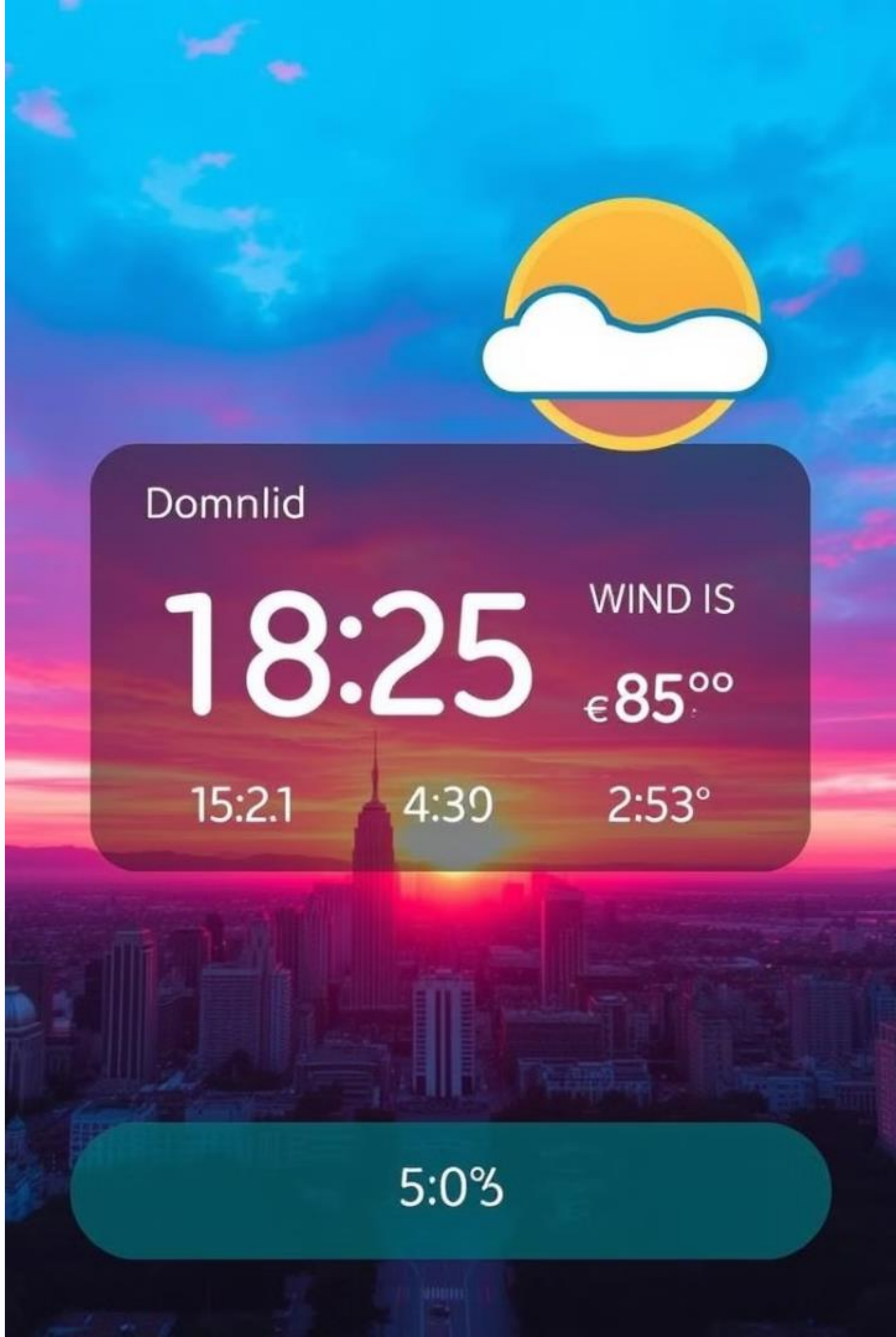
# Real-Time Weather Analysis Using Python

**Analyzing and Visualizing Weather Data Using Python Tools. A Simple Approach to Understanding Weather Trends.**

*Submitted to-*
*Neeraj Kumar Sharma*
*Assistant Professor*
*IIT Guwahati*

*Submitted by-*
*Vansh Kalra*
*244161009*
*M. Tech Data Science*
*IIT Guwahati*



# Motivation: Why is real-time weather analysis important?

1 Informed Decision-Making

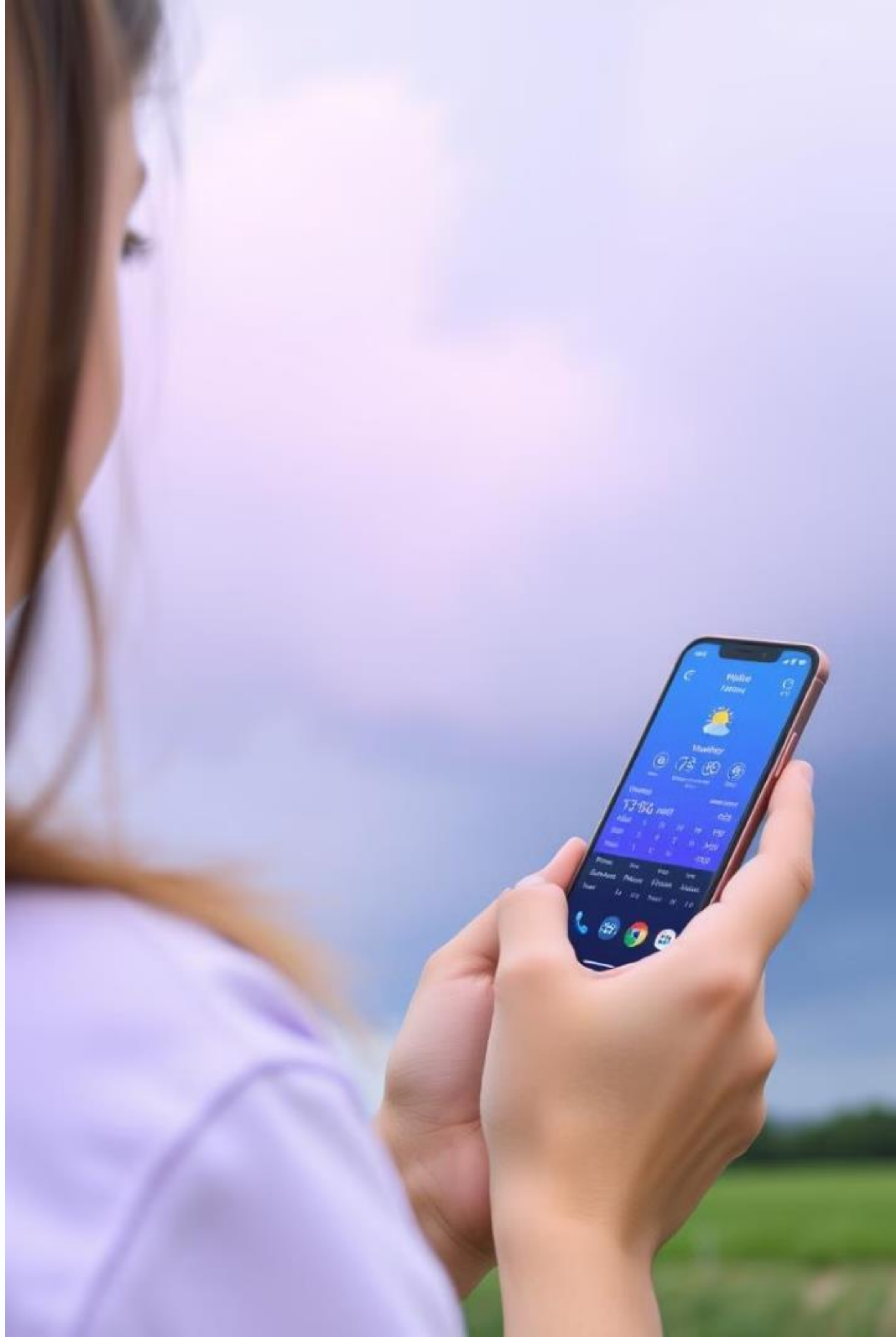
Real-time weather information empowers individuals and organizations to make informed decisions based on current conditions.

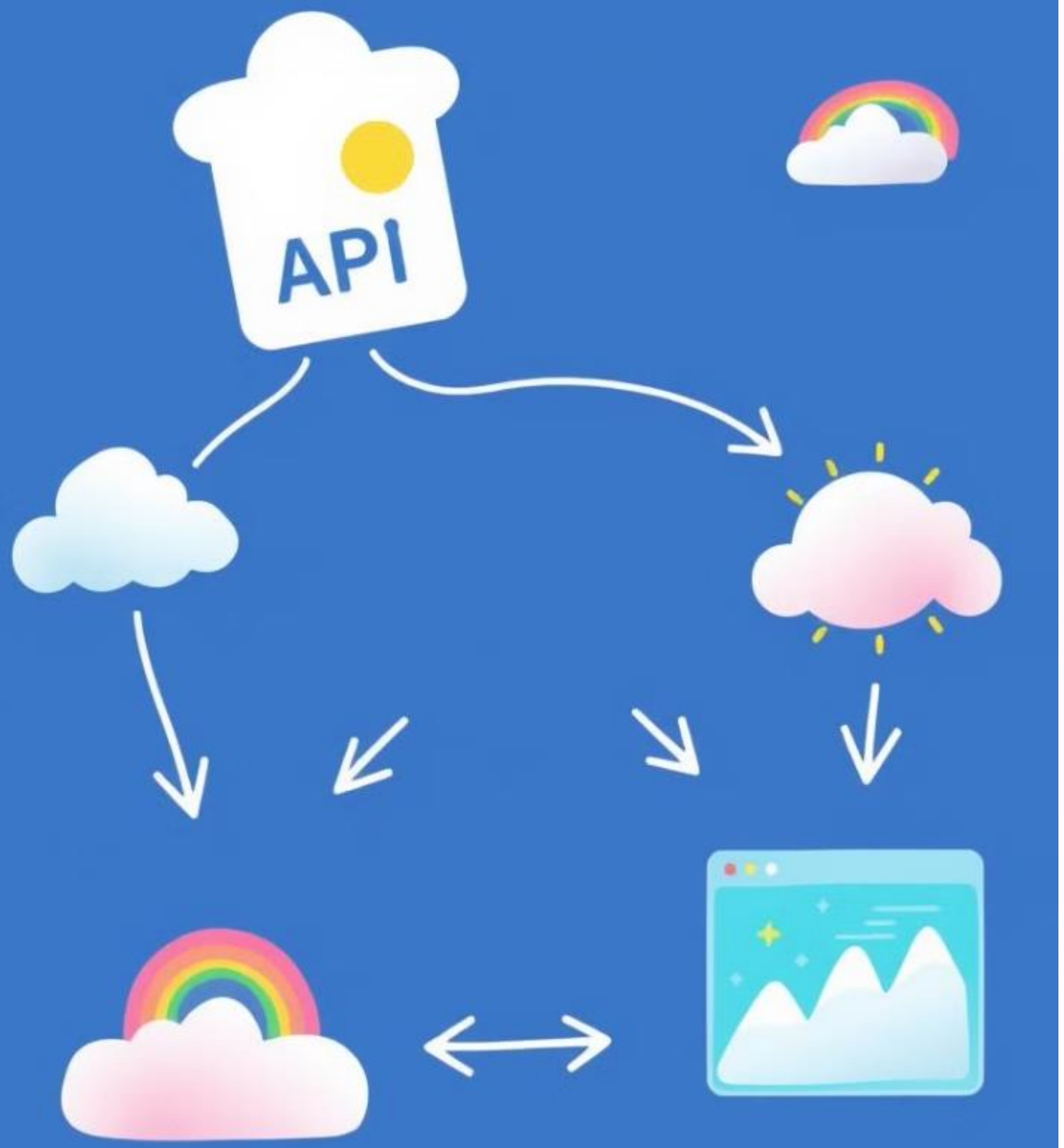
2 Safety and Preparedness

Accurate weather forecasting is essential for ensuring safety in various scenarios, such as transportation, agriculture, and disaster management.

3 Business Optimization

Industries like transportation, energy, and tourism can optimize their operations based on real-time weather data.





# Approach: High-level block diagram of the solution

1 Data Acquisition

Fetch real-time weather data from a weather API using Python libraries.

2 Data Preprocessing

Clean and transform the raw data into a format suitable for analysis.

3 Data Analysis

Perform statistical analysis and generate insights from the processed data.

4 Data Visualization

Create interactive plots and charts to present the results in a visually appealing manner.

# Pseudo Code-

```
# Function to call Openweather API and fetch weather data
response =
requests.get(f"http://api.openweathermap.org/data/2.5/forecast?q={city}&appid={api_key}&units=metric")
data = response.json()
weather_data = {
    "date": [entry["dt_txt"] for entry in data["list"]],
    "temperature": [entry["main"]["temp"] for entry in data["list"]],
    "humidity": [entry["main"]["humidity"] for entry in data["list"]],
    "weather": [entry["weather"][0]["description"] for entry in data[“list"]],
}
```

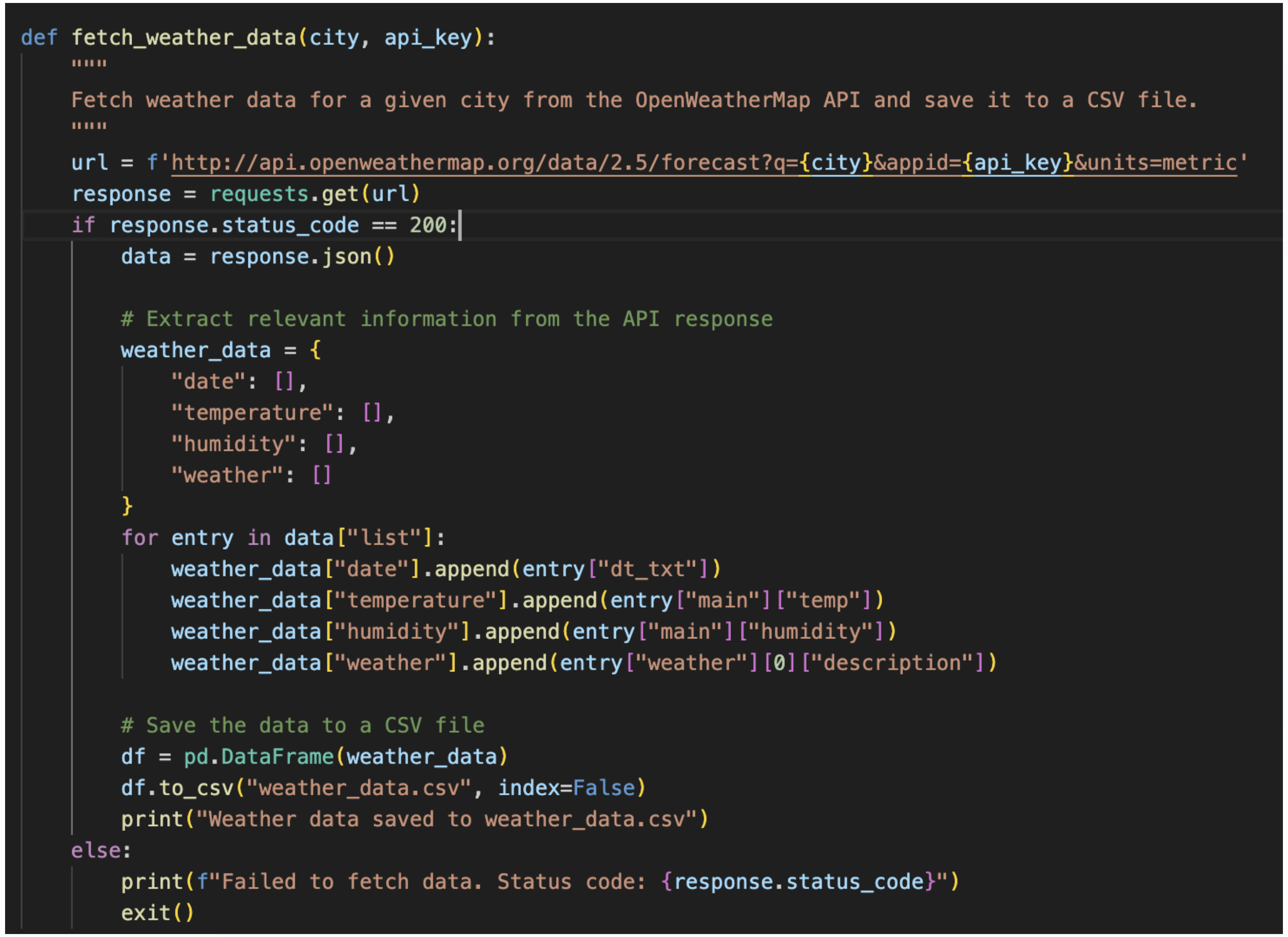
```
# Function to calculate rolling statistics and outlier detection
df['rolling_mean'] = df['temperature'].rolling(window=3).mean()
df['rolling_std'] = df['temperature'].rolling(window=3).std()
df['is_outlier'] = np.abs(df['temperature'] - df['rolling_mean']) > 2 * df['rolling_std']
```

```
# Function to visualise the data
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x="date", y="temperature", marker="o", label="Temperature (°C)")
plt.xticks(rotation=45)
plt.title("Temperature Trend Over Time")
plt.xlabel("Date")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.tight_layout()
plt.show()
```

Similarly more visualisations of different trends is done using different visuals…

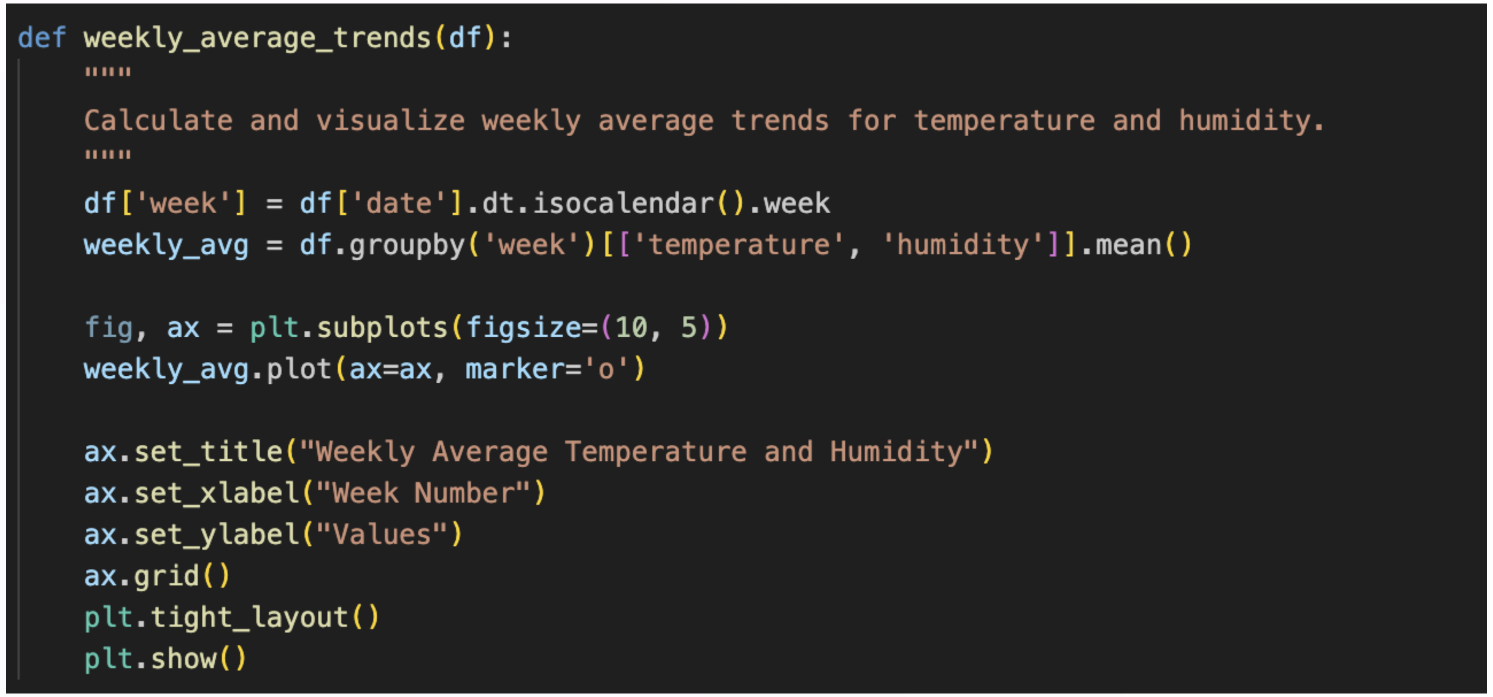
# Snapshots of python features used

Making csv file of the extracted data at real time



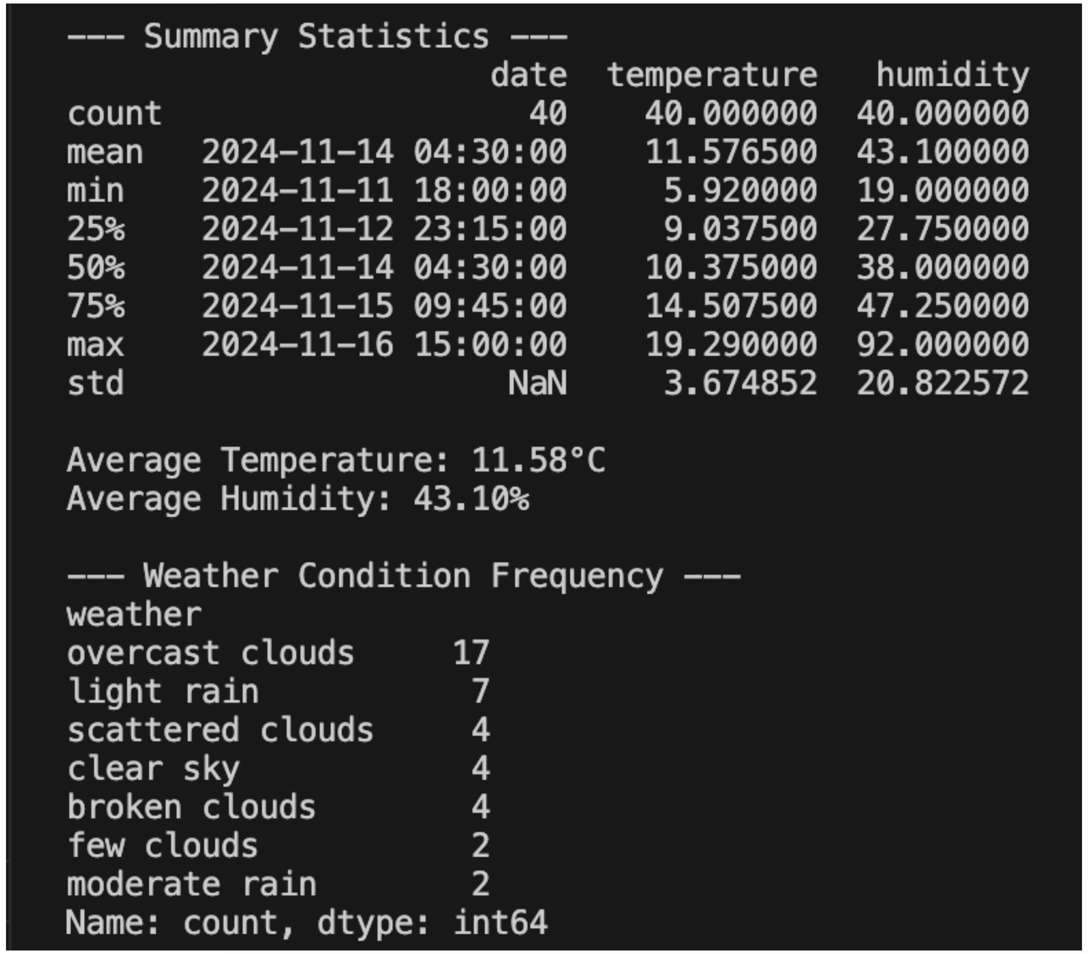
```python
def fetch_weather_data(city, api_key):
    """
    Fetch weather data for a given city from the OpenWeatherMap API and save it to a CSV file.
    """
    url = f'http://api.openweathermap.org/data/2.5/forecast?q={city}&appid={api_key}&units=metric'
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()

        # Extract relevant information from the API response
        weather_data = {
            "date": [],
            "temperature": [],
            "humidity": [],
            "weather": []
        }
        for entry in data["list"]:
            weather_data["date"].append(entry["dt_txt"])
            weather_data["temperature"].append(entry["main"]["temp"])
            weather_data["humidity"].append(entry["main"]["humidity"])
            weather_data["weather"].append(entry["weather"][0]["description"])

        # Save the data to a CSV file
        df = pd.DataFrame(weather_data)
        df.to_csv("weather_data.csv", index=False)
        print("Weather data saved to weather_data.csv")
    else:
        print(f"Failed to fetch data. Status code: {response.status_code}")
        exit()
```

Accessing csv file and plotting visuals using python libraries.

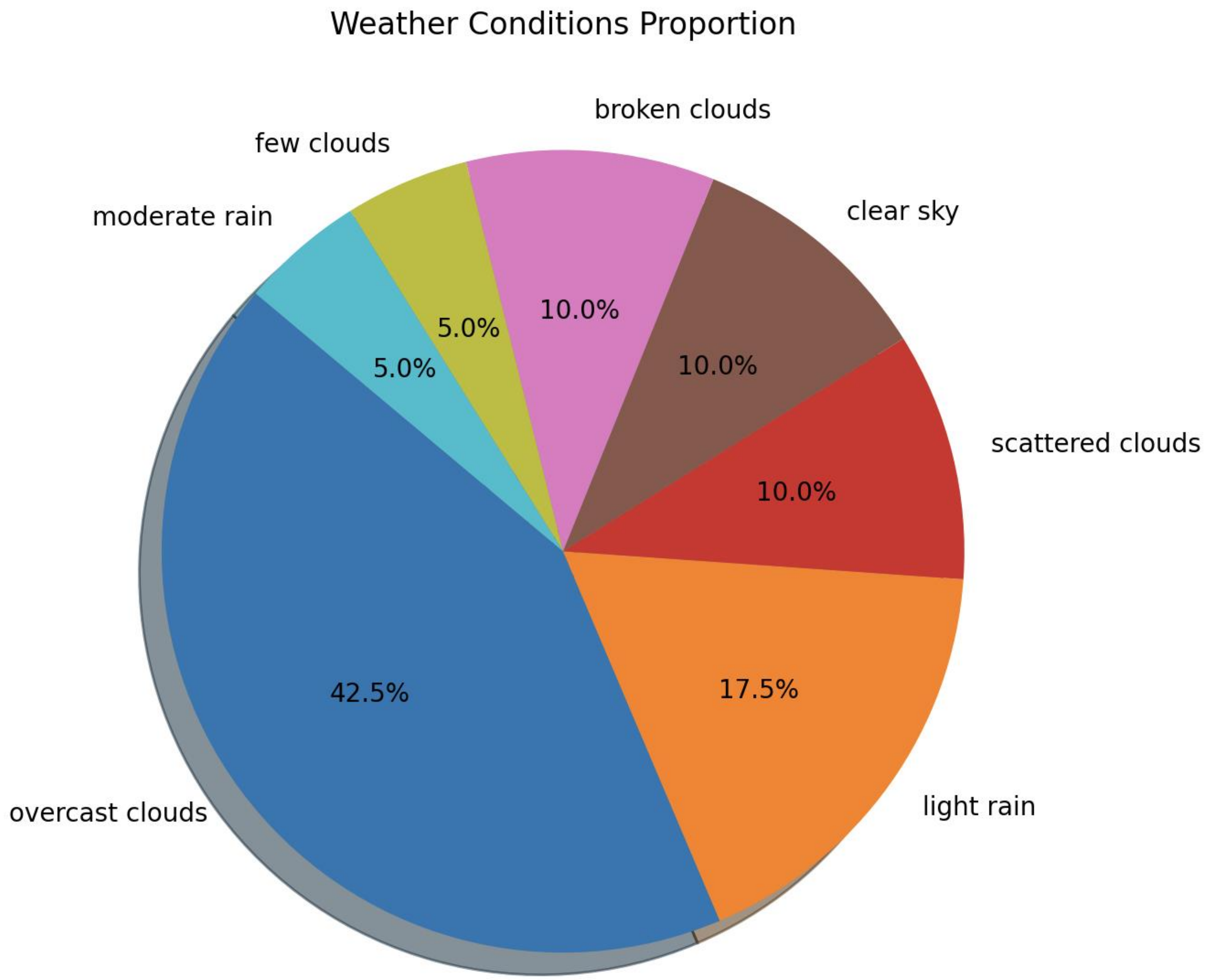


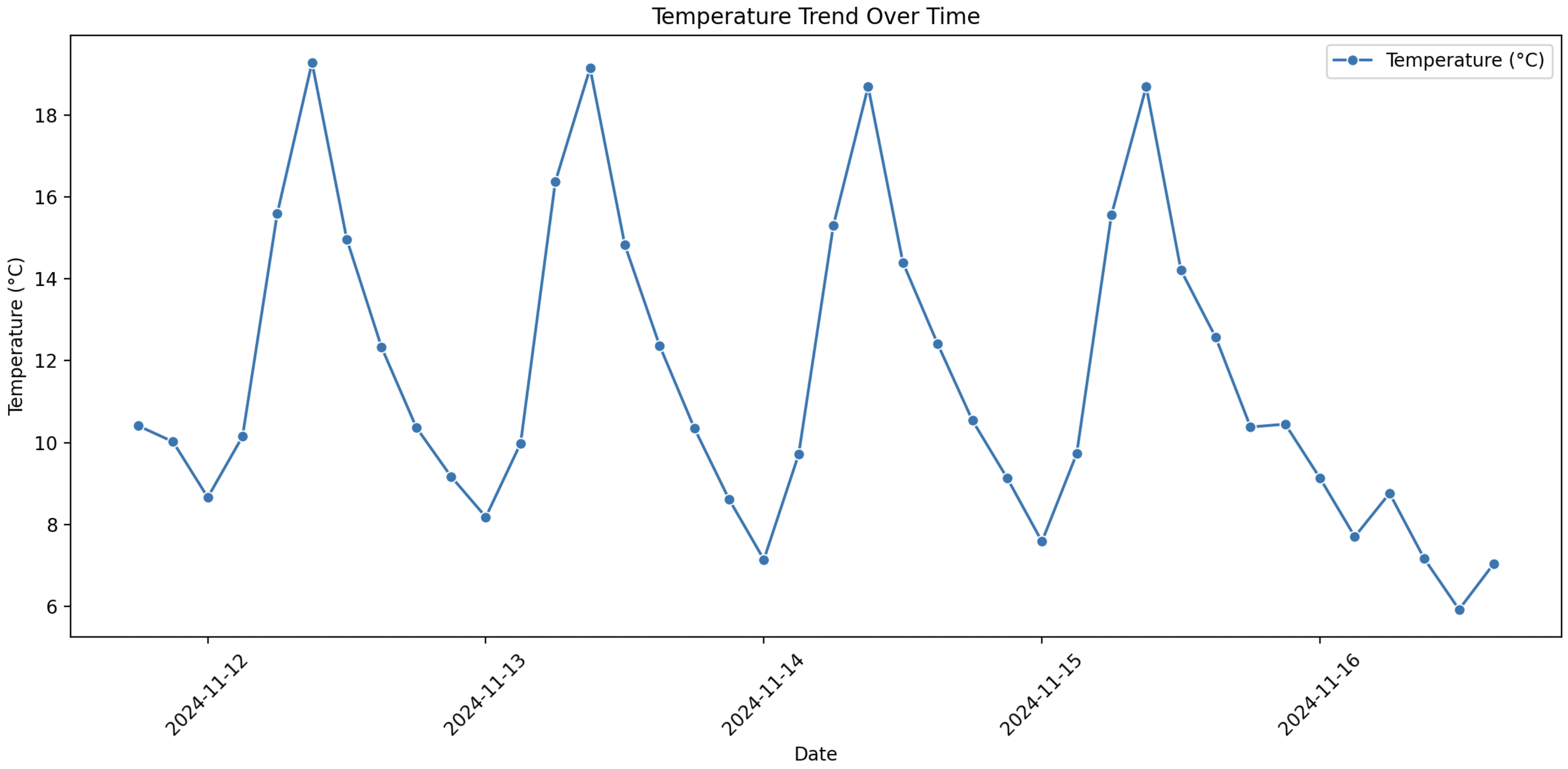
```python
def weekly_average_trends(df):
    """
    Calculate and visualize weekly average trends for temperature and humidity.
    """
    df['week'] = df['date'].dt.isocalendar().week
    weekly_avg = df.groupby('week')[['temperature', 'humidity']].mean()

    fig, ax = plt.subplots(figsize=(10, 5))
    weekly_avg.plot(ax=ax, marker='o')

    ax.set_title("Weekly Average Temperature and Humidity")
    ax.set_xlabel("Week Number")
    ax.set_ylabel("Values")
    ax.grid()
    plt.tight_layout()
    plt.show()
```

# Results

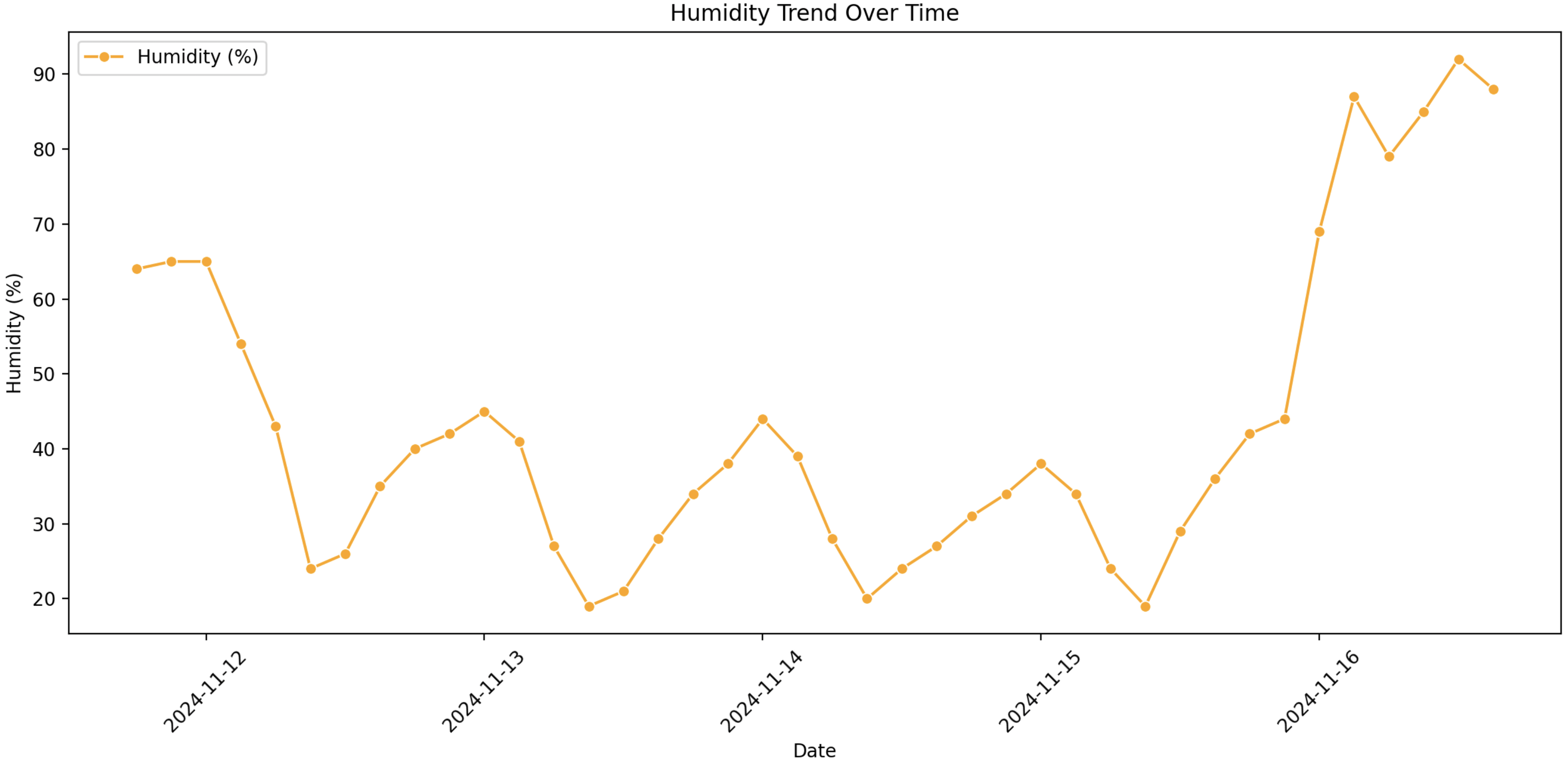


```
--- Summary Statistics ---
                      date  temperature   humidity
count                   40    40.000000  40.000000
mean   2024-11-14 04:30:00    11.576500  43.100000
min    2024-11-11 18:00:00     5.920000  19.000000
25%    2024-11-12 23:15:00     9.037500  27.750000
50%    2024-11-14 04:30:00    10.375000  38.000000
75%    2024-11-15 09:45:00    14.507500  47.250000
max    2024-11-16 15:00:00    19.290000  92.000000
std                    NaN     3.674852  20.822572

Average Temperature: 11.58°C
Average Humidity: 43.10%

--- Weather Condition Frequency ---
weather
overcast clouds     17
light rain           7
scattered clouds     4
clear sky            4
broken clouds        4
few clouds           2
moderate rain        2
Name: count, dtype: int64
```





Temperature Trend Over Time



Humidity Trend Over Time

# Challenges Faced and Solutions Implemented

### Data Consistency

Ensure data accuracy and handle missing values or inconsistent data formats.

### API Rate Limits

Implement strategies to avoid exceeding API request limits and manage resource consumption.

### Data Visualization Complexity

Select appropriate visualization techniques to effectively convey complex weather data.

# Key Takeaways

1. Using **pandas** for efficient data processing.

2. Creating interactive and informative visualizations with **matplotlib** and ***seaborn***.

3. How to fetch and parse live weather data from an API.

4. Successfully analyzed and visualized weather data for a selected city.

5. Identified outliers and key trends using Python.

6. Visualizations provide actionable insights into weather patterns.

# Future Enhancements and Potential Improvements

1

## Machine Learning

Implement machine learning algorithms to predict future weather patterns with higher accuracy.

2

## Advanced Visualization

Explore interactive and immersive visualization techniques for enhanced data exploration.

3

## Data Integration

Integrate weather data with other relevant data sources, such as traffic and air quality information.

# Conclusion

Real-time weather analysis using Python empowers us to gain valuable insights from weather data, leading to informed decisions and improved outcomes in various domains.

THANK YOU !