Weather Data Analytics Project

GitHub Repository:

`https://github.com/adildeokar/Weather-Data-Analytics-Project`

Table of Contents

- Project Overview
- Objectives
- Workflow Architecture
- Environment Configuration
- Database Setup (`database_setup.py`)
- Live Weather Data API Client (`weather_api.py`)
- Data Collection (`data_collector.py`)
- Data Visualization (`weather_visualization.py`)
- Machine Learning Forecasting (`weather_ml_prediction.py`)
- Application Orchestration (`main.py`)
- requirements.txt
- Sample `.env.example`
- Key Screenshots & Outputs

Project Overview

This project is a **comprehensive weather analytics system** involving automated data collection from a live public weather API, storage into PostgreSQL, robust visualizations for the last 30 days, and ML-based weather forecasting. Code uses modular, industry-standard Python and is ready for deployment and extension.

Objectives

1. **Fetch live weather data from an open public API and store in PostgreSQL**
2. **Design and implement the PostgreSQL database and schema**
3. **Visualize the last 30 days of weather data for a city**
4. **Implement basic ML models for predictive weather forecasting**

Workflow Architecture

- **Data is fetched** from OpenWeatherMap using REST API calls.
- **Data is parsed and inserted** into a PostgreSQL database (schema provided).
- **Automated scripts** enable scheduling of data collection.
- **Visualization** modules analyze and plot historical trends.
- **ML models** are trained/validated for temperature forecasting.

Environment Configuration

requirements.txt

```
requests==2.31.0
psycopg2-binary==2.9.7
pandas==2.0.3
numpy==1.24.3
matplotlib==3.7.2
seaborn==0.12.2
scikit-learn==1.3.0
plotly==5.15.0
python-dotenv==1.0.0
schedule==1.2.0
```

.env.example

```
OPENWEATHERMAP_API_KEY=your_api_key_here
DB_HOST=localhost
DB_PORT=5432
DB_NAME=weather_db
DB_USER=your_username
DB_PASSWORD=your_password
DEFAULT_CITY=Mumbai
DEFAULT_COUNTRY_CODE=IN
```

Database Setup (database_setup.py)

```python
import psycopg2
import os
from dotenv import load_dotenv

load_dotenv()

class DatabaseManager:
    def __init__(self):
        self.connection = None
        self.cursor = None

    def connect(self):
        try:
            self.connection = psycopg2.connect(
                host=os.getenv('DB_HOST'),
                port=os.getenv('DB_PORT'),
                database=os.getenv('DB_NAME'),
                user=os.getenv('DB_USER'),
                password=os.getenv('DB_PASSWORD')
            )
            self.cursor = self.connection.cursor()
            print("Connected to PostgreSQL database successfully!")
            return True
        except Exception as e:
            print(f"Error connecting to database: {e}")
            return False

    def create_tables(self):
        drop_tables_query = '''
        DROP TABLE IF EXISTS weather_data CASCADE;
        DROP TABLE IF EXISTS cities CASCADE;
        '''
        create_cities_table = '''
        CREATE TABLE cities (
            id SERIAL PRIMARY KEY,
            city_name VARCHAR(100) NOT NULL,
            country_code VARCHAR(5) NOT NULL,
            latitude DECIMAL(10, 8),
            longitude DECIMAL(11, 8),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            UNIQUE(city_name, country_code)
        );
        '''
        create_weather_table = '''
        CREATE TABLE weather_data (
            id SERIAL PRIMARY KEY,
            city_id INTEGER REFERENCES cities(id),
            temperature DECIMAL(5, 2),
            feels_like DECIMAL(5, 2),
            humidity INTEGER,
            pressure INTEGER,
            weather_main VARCHAR(50),
            weather_description VARCHAR(100),
            wind_speed DECIMAL(5, 2),
            wind_direction INTEGER,
            cloud_coverage INTEGER,
            visibility INTEGER,
            uv_index DECIMAL(4, 2),
            recorded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            date_only DATE GENERATED ALWAYS AS (recorded_at::date) STORED
        );
        '''
        create_indexes = '''
        CREATE INDEX idx_weather_city_date ON weather_data(city_id, date_only);
        CREATE INDEX idx_weather_recorded_at ON weather_data(recorded_at);
        CREATE INDEX idx_cities_name_country ON cities(city_name, country_code);
        '''
        try:
            self.cursor.execute(drop_tables_query)
            self.cursor.execute(create_cities_table)
            self.cursor.execute(create_weather_table)
            self.cursor.execute(create_indexes)
            self.connection.commit()
            print("Tables created successfully!")
        except Exception as e:
            print(f"Error creating tables: {e}")
            self.connection.rollback()

    def insert_city(self, city_name, country_code, latitude=None, longitude=None):
        try:
            query = '''
            INSERT INTO cities (city_name, country_code, latitude, longitude)
            VALUES (%s, %s, %s, %s)
            ON CONFLICT (city_name, country_code) DO UPDATE SET
                latitude = EXCLUDED.latitude,
                longitude = EXCLUDED.longitude
            RETURNING id;
            '''
            self.cursor.execute(query, (city_name, country_code, latitude, longitude))
            city_id = self.cursor.fetchone()[0]
            self.connection.commit()
            return city_id
        except Exception as e:
            print(f"Error inserting city: {e}")
            self.connection.rollback()
            return None

    def insert_weather_data(self, city_id, weather_data):
        try:
            query = '''
            INSERT INTO weather_data (
                city_id, temperature, feels_like, humidity, pressure,
                weather_main, weather_description, wind_speed, wind_direction,
                cloud_coverage, visibility, uv_index
            ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
            '''
            self.cursor.execute(query, (
                city_id,
                weather_data.get('temperature'),
                weather_data.get('feels_like'),
                weather_data.get('humidity'),
                weather_data.get('pressure'),
                weather_data.get('weather_main'),
                weather_data.get('weather_description'),
                weather_data.get('wind_speed'),
                weather_data.get('wind_direction'),
                weather_data.get('cloud_coverage'),
                weather_data.get('visibility'),
                weather_data.get('uv_index')
            ))
            self.connection.commit()
            return True
        except Exception as e:
            print(f"Error inserting weather data: {e}")
            self.connection.rollback()
            return False

    def get_weather_data(self, city_name, days=30):
        try:
            query = '''
            SELECT wd.*, c.city_name, c.country_code
            FROM weather_data wd
            JOIN cities c ON wd.city_id = c.id
            WHERE c.city_name = %s
            AND wd.recorded_at >= NOW() - INTERVAL '%s days'
            ORDER BY wd.recorded_at DESC;
            '''
            self.cursor.execute(query, (city_name, days))
            return self.cursor.fetchall()
        except Exception as e:
            print(f"Error retrieving weather data: {e}")
            return []
    def close(self):
        if self.cursor:
            self.cursor.close()
        if self.connection:
            self.connection.close()
        print("Database connection closed")

if __name__ == "__main__":
    db = DatabaseManager()
    if db.connect():
        db.create_tables()
        # Optional: Insert initial city
        city_id = db.insert_city("Mumbai", "IN", 19.0760, 72.8777)
        if city_id:
            print(f"Mumbai city inserted with ID: {city_id}")
        db.close()
```

Live Weather Data API Client (weather_api.py)

```
import requests
import os
from dotenv import load_dotenv

load_dotenv()

class WeatherAPIClient:
    def __init__(self):
        self.api_key = os.getenv('OPENWEATHERMAP_API_KEY')
        self.base_url = "https://api.openweathermap.org/data/2.5"
        if not self.api_key:
            raise ValueError("OpenWeatherMap API key not set.")

    def get_current_weather(self, city_name, country_code=None):
        try:
            location = city_name
            if country_code:
                location += f",{country_code}"
            url = f"{self.base_url}/weather"
            params = {'q': location, 'appid': self.api_key, 'units': 'metric'}
            response = requests.get(url, params=params)
            response.raise_for_status()
            data = response.json()
            weather_data = {
                'temperature': data['main']['temp'],
                'feels_like': data['main']['feels_like'],
                'humidity': data['main']['humidity'],
                'pressure': data['main']['pressure'],
                'weather_main': data['weather'][0]['main'],
                'weather_description': data['weather'][0]['description'],
                'wind_speed': data.get('wind', {}).get('speed', 0),
                'wind_direction': data.get('wind', {}).get('deg', 0),
                'cloud_coverage': data.get('clouds', {}).get('all', 0),
                'visibility': data.get('visibility', 10000),
                'uv_index': None,
                'city_info': {
                    'name': data['name'],
                    'country': data['sys']['country'],
                    'latitude': data['coord']['lat'],
                    'longitude': data['coord']['lon']
                }
            }
            return weather_data
        except Exception as e:
            print(f"Error fetching weather data: {e}")
            return None

    def get_uv_index(self, latitude, longitude):
        # Endpoint for UV index is paid, typically skip or simulate in free plans
        return 0

    def get_weather_with_uv(self, city_name, country_code=None):
        weather_data = self.get_current_weather(city_name, country_code)
        if weather_data:
            uv_index = self.get_uv_index(weather_data['city_info']['latitude'],
weather_data['city_info']['longitude'])
            weather_data['uv_index'] = uv_index
        return weather_data

    def test_api_connection(self):
        try:
            test_data = self.get_current_weather("London", "GB")
            return bool(test_data)
        except:
            return False

if __name__ == "__main__":
    api_client = WeatherAPIClient()
    if api_client.test_api_connection():
        print("API key works!")
    else:
        print("API test failed.")
```

Automated Data Collection (data_collector.py)

```
import schedule
import time
from datetime import datetime
from weather_api import WeatherAPIClient
from database_setup import DatabaseManager

class WeatherDataCollector:
    def __init__(self):
        self.api_client = WeatherAPIClient()
        self.db_manager = DatabaseManager()

    def collect_weather_data(self, city_name="Mumbai", country_code="IN"):
        if not self.db_manager.connect():
            return False
        try:
            weather_data = self.api_client.get_weather_with_uv(city_name, country_code)
            if weather_data:
                city_id = self.db_manager.insert_city(
                    weather_data['city_info']['name'],
                    weather_data['city_info']['country'],
                    weather_data['city_info']['latitude'],
                    weather_data['city_info']['longitude']
                )
                if city_id:
                    success = self.db_manager.insert_weather_data(city_id, weather_data)
                    if success:
                        print(f"Weather data collected for {city_name} at 
{datetime.now()}")
                        return True
        except Exception as e:
            print(f"Error collecting weather data: {e}")
        finally:
            self.db_manager.close()
        return False

    def start_scheduled_collection(self):
        schedule.every().hour.do(self.collect_weather_data)
        print("Started scheduled weather data collection...")
        while True:
            schedule.run_pending()
            time.sleep(60)

if __name__ == "__main__":
    collector = WeatherDataCollector()
    collector.start_scheduled_collection()
```

Data Visualization (`weather_visualization.py`)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from database_setup import DatabaseManager

class WeatherVisualizer:
    def __init__(self):
        self.db_manager = DatabaseManager()
    def get_data_for_visualization(self, city_name="Mumbai", days=30):
        if not self.db_manager.connect():
            return None
        try:
            data = self.db_manager.get_weather_data(city_name, days)
            cols = ['id', 'city_id', 'temperature', 'feels_like', 'humidity',
                    'pressure', 'weather_main', 'weather_description',
                    'wind_speed', 'wind_direction', 'cloud_coverage',
                    'visibility', 'uv_index', 'recorded_at', 'date_only',
                    'city_name', 'country_code']
            df = pd.DataFrame(data, columns=cols)
            df['recorded_at'] = pd.to_datetime(df['recorded_at'])
            return df
        finally:
            self.db_manager.close()
    def create_temperature_trend(self, df):
        plt.plot(df['recorded_at'], df['temperature'], label="Temperature (C)")
        plt.plot(df['recorded_at'], df['feels_like'], label="Feels Like (C)", alpha=0.7)
        plt.title("Temperature Trend (Last 30 Days)")
        plt.xlabel("Date")
        plt.ylabel("Temperature (Celsius)")
        plt.grid(alpha=0.3)
        plt.legend()
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.savefig("temperature_trend.png")
        plt.show()
    def create_interactive_plot(self, df):
        fig = go.Figure()
        fig.add_trace(go.Scatter(x=df['recorded_at'], y=df['temperature'],
                                 mode='lines+markers', name='Temperature'))
        fig.add_trace(go.Scatter(x=df['recorded_at'], y=df['humidity'],
                                 mode='lines+markers', name='Humidity (%)', yaxis="y2"))
        fig.update_layout(
            title="Temperature and Humidity (Last 30 Days)",
            xaxis_title="Date",
            yaxis=dict(title='Temperature (C)', side='left'),
            yaxis2=dict(title='Humidity (%)', side='right', overlaying='y'),
            hovermode='x unified'
        )
        fig.write_html("interactive_weather_plot.html")
        fig.show()

if __name__ == "__main__":
    viz = WeatherVisualizer()
    df = viz.get_data_for_visualization()
    if df is not None and not df.empty:
        viz.create_temperature_trend(df)
        viz.create_interactive_plot(df)
```

ML Model Training & Prediction (weather_ml_prediction.py)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from database_setup import DatabaseManager

class WeatherPredictor:
    def __init__(self):
        self.db_manager = DatabaseManager()
    def prepare_features(self, df):
        df['hour'] = df['recorded_at'].dt.hour
        df['day'] = df['recorded_at'].dt.day
        df['month'] = df['recorded_at'].dt.month
        df['temp_lag_1'] = df['temperature'].shift(1)
        df['humidity_lag_1'] = df['humidity'].shift(1)
        df = df.dropna()
        return df
    def get_training_data(self, city_name="Mumbai", days=30):
        if not self.db_manager.connect():
            return None, None
        try:
            data = self.db_manager.get_weather_data(city_name, days)
            cols = ['id', 'city_id', 'temperature', 'feels_like', 'humidity',
                    'pressure', 'weather_main', 'weather_description',
                    'wind_speed', 'wind_direction', 'cloud_coverage',
                    'visibility', 'uv_index', 'recorded_at', 'date_only',
                    'city_name', 'country_code']
            df = pd.DataFrame(data, columns=cols)
            df['recorded_at'] = pd.to_datetime(df['recorded_at'])
            df = self.prepare_features(df)
            features =
['humidity','pressure','wind_speed','cloud_coverage','hour','day','month','temp_lag_1','
humidity_lag_1']
            X = df[features]
            y = df['temperature']
            return X, y
        finally:
            self.db_manager.close()
    def train_models(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
        models = {
            'Linear Regression': LinearRegression(),
            'Random Forest': RandomForestRegressor(),
            'SVR': SVR()
        }
        for name, model in models.items():
            model.fit(X_train_scaled, y_train)
            y_pred = model.predict(X_test_scaled)
            rmse = np.sqrt(mean_squared_error(y_test, y_pred))
            mae = mean_absolute_error(y_test, y_pred)
            r2 = r2_score(y_test, y_pred)
            print(f"Model: {name}  RMSE: {rmse:.2f}  MAE: {mae:.2f}  R2: {r2:.2f}")
        return models

if __name__ == "__main__":
    predictor = WeatherPredictor()
    X, y = predictor.get_training_data()
    if X is not None and y is not None:
        predictor.train_models(X, y)
```

Application Orchestration (`main.py`)

```
import time
from database_setup import DatabaseManager
from weather_api import WeatherAPIClient
from data_collector import WeatherDataCollector
from weather_visualization import WeatherVisualizer
from weather_ml_prediction import WeatherPredictor

def main():
    db_manager = DatabaseManager()
    if db_manager.connect():
        db_manager.create_tables()
        db_manager.close()
    api = WeatherAPIClient()
    if api.test_api_connection():
        print("API ok, continuing.")
    collector = WeatherDataCollector()
    collector.collect_weather_data("Mumbai", "IN")
    viz = WeatherVisualizer()
    df = viz.get_data_for_visualization("Mumbai", 30)
    if df is not None and not df.empty:
        viz.create_temperature_trend(df)
        viz.create_interactive_plot(df)
    predictor = WeatherPredictor()
    X, y = predictor.get_training_data("Mumbai", 30)
    if X is not None:
        predictor.train_models(X, y)

if __name__ == "__main__":
    main()
```

Key Outputs

- PNG/HTML weather trend graphs.
- Interactive visualizations.
- ML model evaluation with scores.