**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)**

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.