Renuka Role ADT23SOCB0847
TY AIEC 2 Roll no: 14
24-08-2025 Data Engineering

Case Study

Capstone Project: Data Lifecycle for Weather — Live Weather Data Collection, Storage, Analysis & Forecasting

Problem Statement: Weather plays a vital role in daily life, agriculture, transportation, and disaster management. Accurate and timely forecasting helps in decision-making and preparedness. The aim of this project is to collect live weather data from a public API, store it in PostgreSQL, analyze the last 30 days for a selected city, visualize results, and apply basic machine learning models for weather forecasting. This project demonstrates the complete data lifecycle from acquisition to analysis and prediction.

Project Objectives:

1. Fetch live weather data from a public API (OpenWeatherMap).
2. Design PostgreSQL databases and implement efficient storage structures.
3. Build an ETL pipeline to clean and transform raw API data.
4. Perform 30-day weather trend analysis for a specific city.
5. Create visualizations (temperature, humidity trends).
6. Apply ML models (Linear Regression, Random Forest, ARIMA) to forecast short-term weather.
7. Document the entire process and host the working code on GitHub.

Project Architecture (High-Level):

1. Data Source: Public Weather API (OpenWeatherMap) 2. Data Ingestion: Python script scheduled at intervals (cron/Task Scheduler). 3. Data Storage: PostgreSQL database with raw and cleaned tables. 4. Data Transformation: ETL pipeline using Pandas. 5. Data Visualization:

Matplotlib/Seaborn plots. 6. Forecasting: Machine learning models trained on historical data. 7. Delivery: Report in PDF with GitHub repo for reproducibility.

Database Schema:

```
CREATE DATABASE weather_capstone;
\c weather_capstone;

CREATE TABLE raw_weather (   id SERIAL
PRIMARY KEY,   city VARCHAR(100),
api_timestamp TIMESTAMP WITH TIME ZONE,
raw_json JSONB,   fetched_at TIMESTAMP
DEFAULT now() );

CREATE TABLE cleaned_weather (   id
SERIAL PRIMARY KEY,   city
VARCHAR(100),   obs_time TIMESTAMP,
temp_c REAL,   humidity INTEGER,
pressure INTEGER,   wind_speed REAL,
weather_main VARCHAR(50),
weather_desc VARCHAR(100),
created_at TIMESTAMP DEFAULT now()
);

CREATE INDEX idx_cleaned_city_time ON cleaned_weather(city, obs_time);
```

Data Ingestion Script (fetch_weather.py):

```
# fetch_weather.py import os,
requests, json from datetime
import datetime
from sqlalchemy import create_engine, text

API_KEY = os.getenv('OWM_API_KEY')
CITY = 'London'
API_URL = f"https://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}&units=metric"
DB_URL = os.getenv('DATABASE_URL') engine = create_engine(DB_URL)

resp = requests.get(API_URL)
data = resp.json()

with engine.connect() as conn:
    insert_sql = text("INSERT INTO raw_weather (city, api_timestamp, raw_json) VALUES (:city, :api_t
conn.execute(insert_sql, {'city': CITY, 'api_timestamp': datetime.utcfromtimestamp(data['dt']), '
conn.commit() print('Inserted raw weather')
```

ETL Script (etl_load.py):

```
# etl_load.py import os, json,
pandas as pd from sqlalchemy import
create_engine

DB_URL = os.getenv('DATABASE_URL')
engine = create_engine(DB_URL)

query = "SELECT id, city, api_timestamp, raw_json FROM raw_weather WHERE api_timestamp >= now() - int
raw = pd.read_sql(query, engine)

rows = [] for _, r in
raw.iterrows():
    j = json.loads(r['raw_json'])     obs_time =
pd.to_datetime(r['api_timestamp'])     main =
j.get('main', {})     wind = j.get('wind', {})
weather = (j.get('weather') or [{}])[0]
rows.append({         'city': r['city'],
        'obs_time': obs_time,
        'temp_c': main.get('temp'),
        'humidity': main.get('humidity'),
        'pressure': main.get('pressure'),
        'wind_speed': wind.get('speed'),
        'weather_main': weather.get('main'),
        'weather_desc': weather.get('description')
})

clean_df = pd.DataFrame(rows)
clean_df.drop_duplicates(subset=['city','obs_time'], inplace=True)
clean_df.to_sql('cleaned_weather', engine, if_exists='append', index=False)
print('ETL completed, inserted', len(clean_df), 'rows')
```

Visualization of Last 30 Days (visualize_last30.py):

```
# visualize_last30.py import os, pandas as pd,
matplotlib.pyplot as plt from sqlalchemy import
create_engine

DB_URL = os.getenv('DATABASE_URL')
engine = create_engine(DB_URL)
CITY='London'

query = "SELECT obs_time, temp_c, humidity FROM cleaned_weather WHERE city = :city AND obs_time >= nc
df = pd.read_sql(query, engine, params={'city': CITY})

df['obs_time'] = pd.to_datetime(df['obs_time'])
df.set_index('obs_time', inplace=True) daily =
df.resample('D').mean().ffill()

plt.plot(daily.index, daily['temp_c'])
plt.title(f'Temperature Trend - Last 30 Days ({CITY})')
plt.show()

plt.plot(daily.index, daily['humidity'])
plt.title(f'Humidity Trend - Last 30 Days ({CITY})')
plt.show()
```

Machine Learning Models (ml_models.py):

```
# ml_models.py import os, pandas as pd from
sqlalchemy import create_engine from
sklearn.ensemble import RandomForestRegressor from
sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_absolute_error

engine = create_engine(os.getenv('DATABASE_URL'))
CITY='London'

df = pd.read_sql("SELECT obs_time, temp_c FROM cleaned_weather WHERE city = :city ORDER BY obs_time",
df['obs_time'] = pd.to_datetime(df['obs_time']) df.set_index('obs_time', inplace=True) daily =
df.resample('H').mean().ffill()

for lag in [1,2,3,6,24]:
    daily[f'lag_{lag}'] = daily['temp_c'].shift(lag)
daily['roll_24'] = daily['temp_c'].rolling(24).mean().shift(1)
daily.dropna(inplace=True)

train = daily.iloc[:-24*7] test = daily.iloc[-24*7:] X_train,
y_train = train.drop(columns=['temp_c']), train['temp_c']
X_test, y_test = test.drop(columns=['temp_c']), test['temp_c']

# Linear Regression lr =
LinearRegression().fit(X_train, y_train) pred_lr =
lr.predict(X_test) print("LR MAE:",
mean_absolute_error(y_test, pred_lr))

# Random Forest rf =
RandomForestRegressor().fit(X_train, y_train) pred_rf
= rf.predict(X_test) print("RF MAE:",
mean_absolute_error(y_test, pred_rf))
```

Conclusion:

This capstone project demonstrates a full data pipeline for live weather data, from acquisition using public APIs, storage in PostgreSQL, ETL transformation, visualization of trends, and short-term forecasting using machine learning. The project highlights the importance of structured data management, reproducibility via GitHub-hosted code, and predictive modeling for decision-making. Future enhancements may include deep learning models (LSTM), integration with cloud data warehouses, and real-time dashboards using Plotly or Streamlit.