Air Quality (PM2.5) Forecasting

(3 horizons: +24h, +48h, +72h)

1. Objective

The goal of this project was to build an **end-to-end hourly pipeline** that:

- Pulls the latest air-quality and weather data for **Karachi**,
- Features (lags, rolling stats, calendar info),
- Trains and versions models for **24h, 48h, 72h ahead PM2.5**,
- Exposes the forecasts in a **Streamlit UI**
- Pushes the engineered features to **Hopsworks Feature Store** from GitHub Actions.

2. Data Ingestion (src/ingest.py)

Data sources used:

- **Open-Meteo Geocoding API,** to get latitude, longitude and timezone for the city (Karachi).
- **Open-Meteo Weather API,** to get hourly weather for the date range.
- **OpenAQ,** to get **hourly PM2.5** near Karachi. I searched for nearby locations/sensors within 25 km and aggregated hourly values.
- **Fallback:** if OpenAQ gives nothing, I call **Open-Meteo Air Quality (pm2_5)** so the pipeline still works.

What the script does:

- Pick a lookback window (in GitHub Action I used 90 days).
- Download weather hourly.
- Download PM2.5 hourly.
- Merge the two on timestamp and save to parquet files.

3. Feature Engineering (src/features.py)

Steps:

- **Time features**
    - Hour, day-of-week, day-of-month, month
    - Reason: AQI often follows human/traffic and daily patterns.
- **Lag / rolling features on PM2.5**
    - pm25_lag1 (1 hour ago)
    - pm25_lag24 (same hour previous day)
    - pm25_ma6 (6-hour moving average)
    - pm25_ma24 (24-hour moving average)
    - pm25_chg1 (current – previous hour)
    - Reason: AQI is very auto correlated; recent history is the strongest signal.
- **Future targets**
    - pm25_tplus_24
    - pm25_tplus_48
    - pm25_tplus_72
- **Save final training file to:** data/features/features.parquet

4. Model Training (src/train.py)

- Predict PM2.5 +24h
- Predict PM2.5 +48h
- Predict PM2.5 +72h

Models compared per horizon:

- **Ridge Regression,** fast, good baseline
- **RandomForestRegressor,** non-linear, handles interactions
- **GradientBoostingRegressor,** added to improve expressiveness

For each horizon:

- Loaded data/features/features.parquet
- Did a train/test split
- Trained the 3 models
- Evaluated with **RMSE, MAE, R^2**
- Picked the model with the **lowest RMSE**

5. Automation (GitHub Actions)

- **features-hourly.yml**
  - Every hour: checkout → install → run ingest.py → run features.py → commit updated features.
  - This keeps features.parquet fresh.
- **train-daily.yml**
  - Daily: checkout → install → run train.py
  - Copy the latest trained models into models/latest/
  - Hopsworks secrets exist → switch to Python 3.10 → install only HSFS → run src/push_features_hopsworks.py
  - Upload models as artifact

6. Streamlit UI (app/streamlit_app.py)

- **Loads the latest trained models** from models/latest/
- **Gets the next hours' weather** from Open-Meteo (live)
- **Rebuilds the same feature row** as training (lags from your local pm25 parquet + weather) and runs prediction for 24/48/72h.

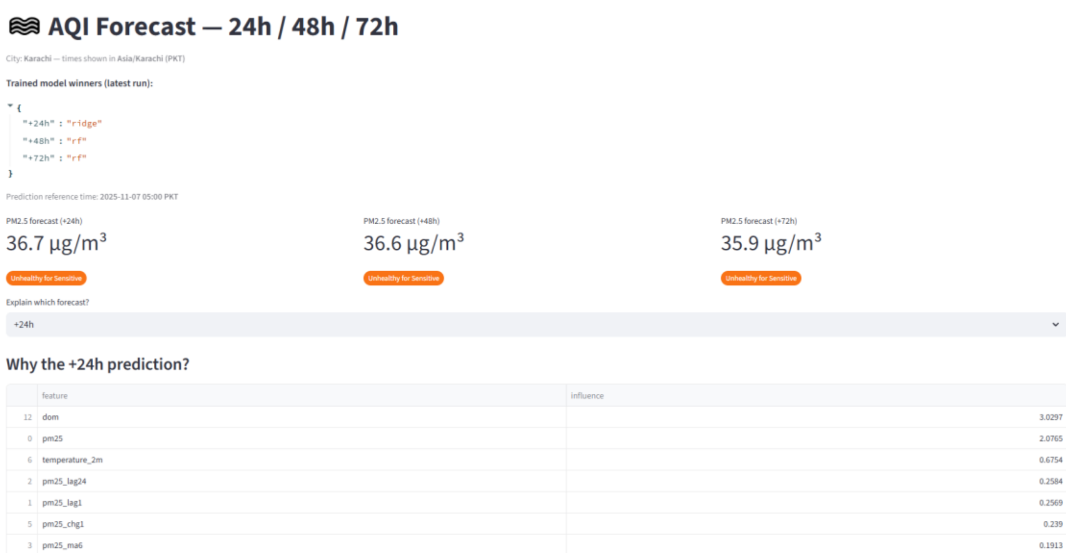
It displays:

- 3 big metrics: “PM2.5 forecast (+24h)” etc.
- color badge for AQI category (Good / Moderate / …)
- a small SHAP-style explanation if available
- debug expanders for recent PM2.5 and training report

7. Hopsworks Integration

We **successfully pushed** the engineered features to Hopsworks Feature Store by installing only **hsfs[python] on Python 3.10 in CI**.

UI:



Training report (JSON)


```
{
  "version" : "2025-11-07_1943"
  "feature_names" : [
    0 : "pm25"
    1 : "pm25_lag1"
    2 : "pm25_lag24"
    3 : "pm25_ma6"
    4 : "pm25_ma24"
    5 : "pm25_chg1"
    6 : "temperature_2m"
    7 : "relative_humidity_2m"
    8 : "wind_speed_10m"
    9 : "surface_pressure"
    10 : "hour"
    11 : "dow"
    12 : "dom"
    13 : "month"
  ]
  "horizons" : {
    "h24" : {
      "best_model" : "ridge"
      "metrics" : {
        "ridge" : {
          "rmse" : 14.4856241253357
          "mae" : 10.187771451380758
          "r2" : 0.31632355556542713
        }
        "rf" : {
          "rmse" : 17.717570910160017
          "mae" : 11.785370577281185
          "r2" : -0.022785476110592784
```


FEATURES:

```python
def add_time_features(df: pd.DataFrame):

    df = df.copy()
    df[TIME_COL] = pd.to_datetime(df[TIME_COL], utc=True)

    df["hour"] = df[TIME_COL].dt.hour
    df["dow"] = df[TIME_COL].dt.dayofweek
    df["dom"] = df[TIME_COL].dt.day
    df["month"] = df[TIME_COL].dt.month

    print("time featuring\n", df.head())
    return df


def add_lag_rolling_change(df: pd.DataFrame):
    """
    Add PM2.5 history columns:
      - pm25_lag1   (1 hour ago)
      - pm25_lag24  (24 hours ago)
      - pm25_ma6    (6-hour moving avg)
      - pm25_ma24   (24-hour moving avg)
      - pm25_chg1   (current - 1 hour ago)
    """
    df = df.copy()

    df[f"{TARGET_COL}_lag1"] = df[TARGET_COL].shift(1)
    df[f"{TARGET_COL}_lag24"] = df[TARGET_COL].shift(24)
    df[f"{TARGET_COL}_ma6"] = df[TARGET_COL].rolling(6).mean()
    df[f"{TARGET_COL}_ma24"] = df[TARGET_COL].rolling(24).mean()
    df[f"{TARGET_COL}_chg1"] = df[TARGET_COL] - df[f"{TARGET_COL}_lag1"]

    print("Lag , Rolling\n", df.head())
    return df


def add_future_targets(df: pd.DataFrame):

    df = df.copy()

    for h in [24, 48, 72]:
        col_name = f"{TARGET_COL}_tplus_{h}"
        df[col_name] = df[TARGET_COL].shift(-h)

    print("Future Targets\n", df.head())
    return df
```