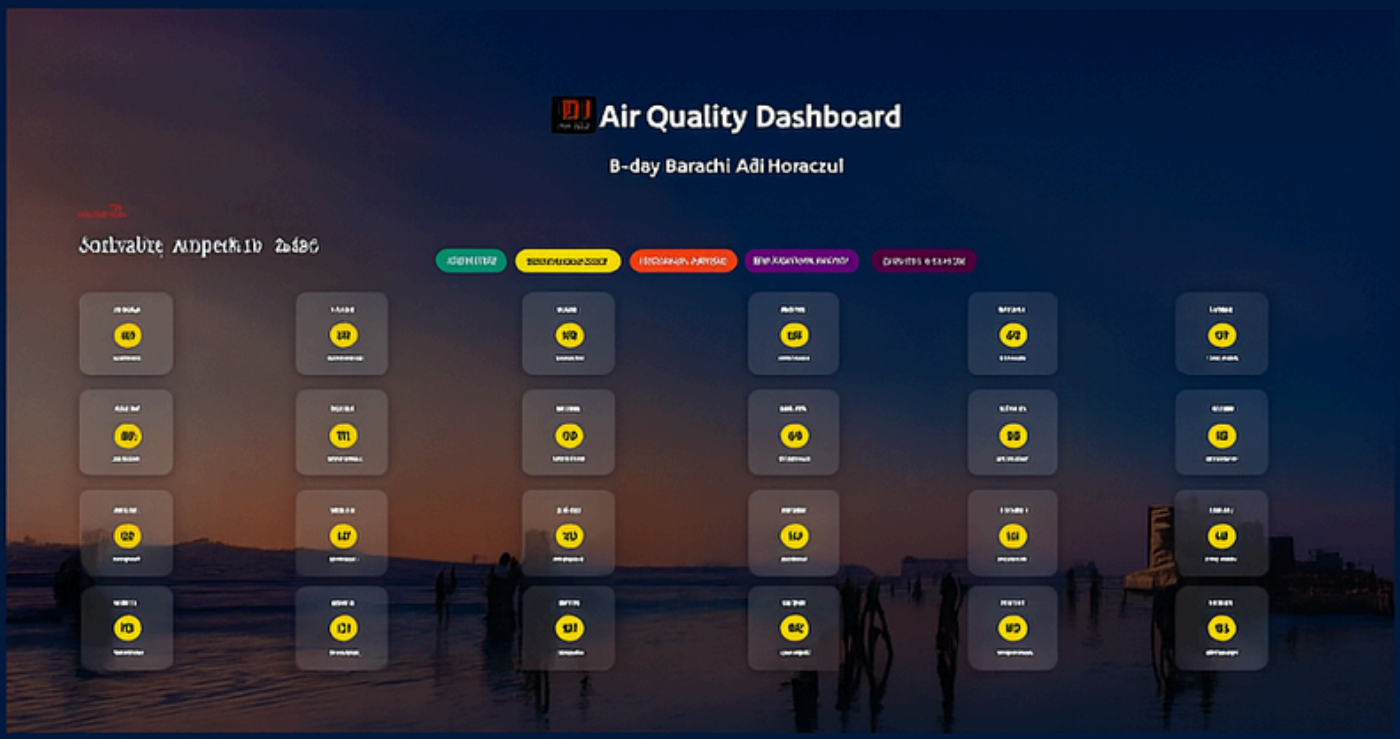
KARACHI AQI PREDICTION REPORT



Alina Siddiqui

10 PEARLS AQI PREDICTION REPORT

Name: Alina Siddiqui
Institute: IBA
Dept: Data Science
Submission Date: 15/8/25

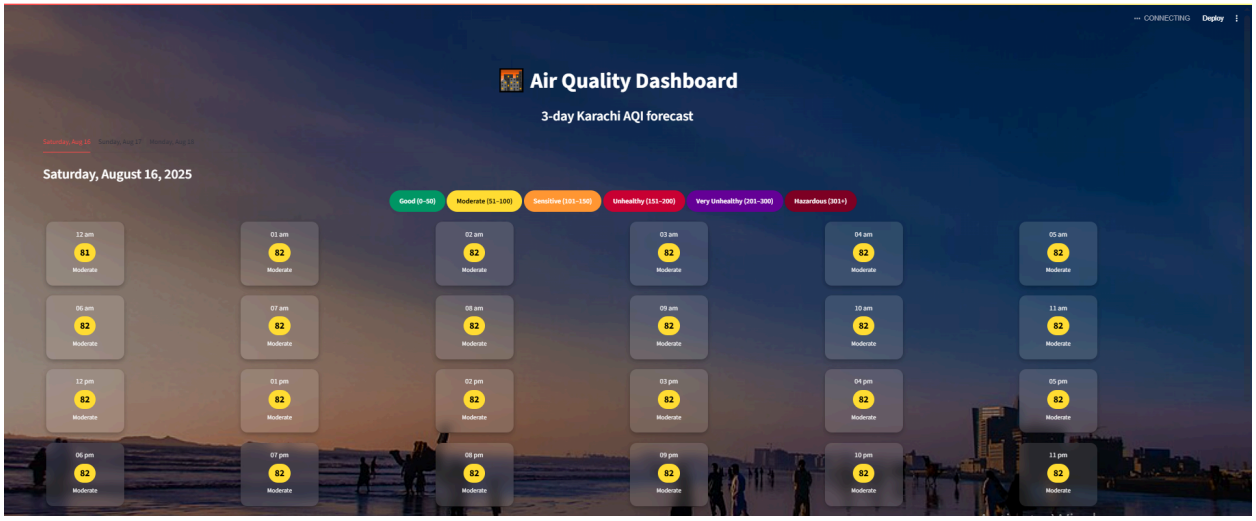


TABLE OF CONTENTS

1. Introduction **5**

1.1 Overview of Air Quality Index (AQI) 5

1.2 How AQI is Calculated 5

1.3 US AQI (USAQI) Standards 6

1.4 Importance of Forecasting Air Quality in Karachi 7

1.5 Project Context — 10Pearls AQI Forecasting System 8

2. Problem Statement **9**

3. Objectives **10**

Specific Objectives: 10

4. Project Scope **12**

4.1 In Scope 12

4.2 Out of Scope 13

5. Data Collection & Sources **14**

5.1 Overview 14

5.2 Data Providers & Endpoints 14

5.3 Historical Backfill (one-time) 15

5.4 Daily Collection (recurring) 15

5.5 Feature Store Integration (Hopsworks) 15

5.6 CI/CD Automation (GitHub Actions) 16

6.7 Data Dictionary (post-merge hourly table) 17

6.8 Data Quality & Integrity Practices 17

6. Exploratory Data Analysis (EDA) - Before Data Preprocessing and Feature Engineering **19**

6.1 Data Loading and Initial Inspection 19

6.2 Time-Series Structure and Missing Values 19

6.3 Statistical Summary 19

6.4 Univariate Analysis 20

6.5 Temporal Trend Analysis 20

6.6 Bivariate and Correlation Analysis 20

6.7 Outlier Detection 20

6.8 Key Insights from EDA 21

7. Data Preprocessing & Feature Engineering **22**

7.1 Inputs and Outputs 22

7.2 Time Indexing & Basic Derivations 22

7.3 Target-Aware Temporal Features (Overall USAQI) 22

7.4 Exogenous / Interaction Features 23
7.5 Missingness & Row Filtering 23
7.6 Feature Scaling 23
7.7 Incremental Processing & Idempotency 24
7.8 Persistence & Formatting 24
7.9 Feature Store Publishing (Hopsworks) 25
7.10 CI/CD (Hourly Preprocessing Automation) 26
7.11 Quality & Governance Considerations 26
7.12 Feature Set Summary (post-processing) 26
8. Exploratory Data Analysis (after Feature Engineering) and Feature Selection 27
8.1 Dataset Overview after Feature Engineering 27
8.2 Correlation and Redundancy Analysis 27
8.3 Feature Importance Ranking (Initial Selection in EDA) 27
8.4 Automation of Selected Features 28
8.5 Continuous Update via GitHub Actions 28
8.6 Final Outcome 28
9. Feature Selection - Approach 2 29
9.1 Objective 29
9.2 Inputs 29
9.3 Method 29
9.4 Why this improved over Approach 1 30
9.5 Outputs 31
9.6 Automation Note 31
10. Model Training 32
LightGBM Multi-Output Model - Detailed Analysis 32
10.1 Objective 32
10.2 Data & features 32
10.3 Preprocessing & sequence building 32
10.4 Model 33
10.5 Training 33
10.6 Evaluation (validation 80/20) 33
10.7 Explainability (SHAP) 34
10.8 Artifacts saved 35
LSTM (Multi-Output, 72h) 35
10.1 Objective 35

10.2 Data & features 35
10.3 Preprocessing & sequence building 36
10.4 Model 36
10.5 Training 37
10.6 Evaluation (validation 80/20) 37
10.7 Explainability 38
10.8 Artifacts saved 38
LSTM (Multi-Output, 72h) - Optuna-tuned 38
10.1 Objective 38
10.2 Data & features 38
10.3 Preprocessing & sequence building 38
10.4 Model 39
10.5 Optuna search 39
10.6 Training 40
10.7 Evaluation (validation 80/20) 40
10.8 Explainability 40
10.9 Artifacts saved 40
TCN (Multi-Output, 72h) 41
10.1 Objective 41
10.2 Data & features 41
10.3 Preprocessing & sequence building 41
10.4 Model 41
10.6 Evaluation (validation 80/20) 43
10.7 Explainability 43
10.8 Artifacts saved 43
Random Forest (Multi-Output, 72h) 44
10.1 Objective 44
10.2 Data & features 44
10.3 Preprocessing & sequence building 44
10.4 Model 44
10.5 Training 45
10.6 Evaluation (validation 80/20) 45
10.7 Explainability 45
10.8 Artifacts saved 45
SARIMAX (Single-Output per Horizon, 72h) - Best Model **45**
10.1 Objective 45

10.2 Data & features 45
10.3 Preprocessing 46
10.4 Model 46
10.5 Forecasting strategy 46
10.6 Evaluation (validation 80/20) 46
10.7 Artifacts saved 47
10.8 Automation 47
11. Frontend Dashboard 51
12. Containerization with Docker 52
13. Key Learnings from the Project 53

1. Introduction

1.1 Overview of Air Quality Index (AQI)

The **Air Quality Index (AQI)** is a standardized scale used to communicate how clean or polluted the air is, along with the potential health effects that poor air quality can cause. It translates complex air pollution data into a single, easy-to-understand number, enabling governments, researchers, and the public to take informed action.

AQI values are typically calculated for major air pollutants regulated by environmental agencies, such as:

- **$PM_{2.5}$** (Particulate Matter smaller than 2.5 micrometers)
- **PM_{10}** (Particulate Matter smaller than 10 micrometers)
- **O_3** (Ground-level Ozone)
- **NO_2** (Nitrogen Dioxide)
- **SO_2** (Sulfur Dioxide)
- **CO** (Carbon Monoxide)

The AQI scale is divided into categories, each associated with a specific health message and color code, making it easier for non-technical audiences to interpret air quality data.

1.2 How AQI is Calculated

The AQI is calculated from measured pollutant concentrations using a piecewise linear function defined by breakpoints. For each pollutant, the AQI value is calculated separately, and the highest individual value is reported as the overall AQI.

The general formula for AQI calculation is:

$$\text{AQI}_p = \frac{I_{HI} - I_{LO}}{BP_{HI} - BP_{LO}} \times (C_p - BP_{LO}) + I_{LO}$$

Where:

- **AQIp** = AQI for pollutant *p*
- **Cp** = Concentration of pollutant *p*
- **BPHI** and **BPLO** = Breakpoints that are closest to CpC_pCp
- **IHI** and **ILO** = AQI values corresponding to those breakpoints

The final AQI reported is:

$$\text{Overall AQI} = \max(\text{AQI}_{PM2.5}, \text{AQI}_{PM10}, \text{AQI}_{O3}, \text{AQI}_{NO2}, \text{AQI}_{SO2}, \text{AQI}_{CO})$$

1.3 US AQI (USAQI) Standards

The **US Air Quality Index (USAQI)**, set by the **United States Environmental Protection Agency (EPA)**, is widely adopted as an international reference. USAQI categorizes air quality into six levels:

AQI Range	Category	Color	Health Implications
0–50	Good	Green	Air quality is considered satisfactory, and air pollution poses little or no risk.
51–100	Moderate	Yellow	Acceptable air quality; some pollutants may be a concern for a small number of unusually sensitive individuals.
101–150	Unhealthy for Sensitive Groups	Orange	Sensitive groups (children, elderly, respiratory disease patients) may experience health effects.
151–200	Unhealthy	Red	Everyone may begin to experience adverse health effects.
201–300	Very Unhealthy	Purple	Health alert: everyone may experience more serious health effects.
301–500	Hazardous	Maroon	Health warning of emergency conditions.

For **$PM_{2.5}$** specifically, breakpoints for USAQI are:

USAQI	**$PM_{2.5}$ (µg/m³) Range**
0–50	0.0 – 12.0
51–100	12.1 – 35.4
101–150	35.5 – 55.4
151–200	55.5 – 150.4
201–300	150.5 – 250.4
301–500	250.5 – 500.4

1.4 Importance of Forecasting Air Quality in Karachi

Karachi, one of the world's most densely populated cities, faces severe air quality issues due to:

- Heavy traffic emissions
- Industrial pollutants
- Dust from unpaved roads and construction
- Burning of waste materials

AQI forecasting is crucial for:

- **Public health protection** – allowing people to reduce outdoor activities when air pollution is high
- **Policy-making** – enabling authorities to plan interventions like traffic restrictions or industrial regulations

- **Environmental awareness** – educating the public about pollution trends

1.5 Project Context — 10Pearls AQI Forecasting System

This project, developed as part of a **10Pearls Data Science Internship**, focuses on building a **predictive system** that forecasts Karachi's AQI up to 72 hours in advance. It integrates:

- **Historical AQI measurements**
- **Weather data** (temperature, humidity, wind speed)
- **Machine learning & statistical models** (e.g., SARIMAX, LSTM, LightGBM)

The system was implemented and tested locally using a **Streamlit-based dashboard** to visualize real-time and future AQI readings. Although it was not deployed to a live server, the local prototype successfully demonstrated the forecasting capability and provided an interactive interface for exploring predictions.

2. Problem Statement

Karachi frequently experiences poor air quality due to traffic congestion, industrial emissions, construction dust, and seasonal factors. High pollution levels can have serious health impacts, particularly for sensitive groups such as children, the elderly, and those with respiratory or cardiovascular conditions. While AQI data is sometimes available for the current time ("nowcasts"), **there is no easily accessible, localized forecasting tool that predicts the *overall* Air Quality Index (USAQI) for Karachi in advance**.

This lack of forecasting prevents:

- Citizens from planning outdoor activities safely.
- Schools, sports facilities, and outdoor workers from adjusting schedules.
- Authorities from issuing timely health advisories.

Our project addresses this gap by building a **local prototype system** that:

- Collects historical AQI and weather data for Karachi.
- Computes the **overall USAQI** from available pollutant readings.
- Uses machine learning and time-series models (e.g., SARIMAX, LightGBM, LSTM) to **forecast the overall AQI** for the next **24, 48, and 72 hours**.
- Presents results via an interactive **Streamlit dashboard** (run locally) with USAQI color bands and health guidance.

I focus on predicting the **overall AQI value**, **not** individual pollutant concentrations like $PM_{2.5}$ or PM_{10}, so users get a single, easy-to-understand indicator of expected air quality.

3. Objectives

The primary objective of this project is to design and implement an **end-to-end AQI prediction system** for Karachi that forecasts the **overall USAQI** for the next **24, 48, and 72 hours**. The system should integrate data collection, feature engineering, model training, prediction, and visualization into a single, reproducible pipeline.

Specific Objectives:

1. **Forecast Overall Air Quality Index (USAQI)**
    - Predict the overall USAQI value, rather than individual pollutant concentrations, to provide an easy-to-understand indicator of expected air quality.
2. **Data Collection and Feature Engineering**
    - Gather historical AQI and weather data from reliable sources.
    - Generate time-based and derived features (e.g., hour, day, month, AQI change rate, lag features) suitable for forecasting models.
3. **Model Development and Evaluation**
    - Experiment with multiple forecasting approaches, including statistical models (SARIMAX), MLmodels (LightGBM), and deep learning models (LSTM).
    - Evaluate models using RMSE, MAE, and R^2, selecting the most accurate for each forecast horizon.
4. **Local Prototype Dashboard**
    - Develop an interactive **Streamlit** dashboard to display current AQI, forecasted AQI values, USAQI category color bands, and related health guidance.
    - Ensure the dashboard runs locally for demonstration purposes.
5. **Explainability**
    - Use SHAP or equivalent methods to identify and visualize the most influential features driving predictions.

6. **Scalability Planning**

    - Design the pipeline so it can later be adapted for automated data ingestion, daily retraining, and deployment in a production environment (CI/CD with tools like GitHub Actions).

4. Project Scope

This project focuses on developing a **local prototype** of an Air Quality Index (AQI) prediction system for **Karachi**, capable of forecasting the **overall USAQI** for the next **24, 48, and 72 hours**. The scope defines the boundaries of what is included in this phase and what is explicitly excluded.

4.1 In Scope

1. **Geographical Coverage**
    - Forecasting air quality **only** for Karachi city, using aggregated data from available AQI and weather sources.
2. **Forecast Target**
    - Prediction of the **overall USAQI** value for each forecast horizon, rather than individual pollutant concentrations (e.g., $PM_{2.5}$, PM_{10}).
3. **Data Sources**
    - AQI data from publicly available repositories.
    - Weather data from the **Open-Meteo API**, including temperature, humidity, wind speed, and other meteorological variables.
    - Data backfilling to create a historical training dataset covering multiple months.
4. **Feature Engineering**
    - Creation of time-based features (hour, day, month), lag features, and derived metrics (e.g., AQI change rate).
5. **Modeling Approaches**
    - Testing multiple forecasting models:
        - **Statistical**: SARIMAX
        - **Machine Learning**: LightGBM, Random Forest, Ridge Regression
        - **Deep Learning**: LSTM
    - Selection of the best-performing model(s) based on RMSE, MAE, and R^2.

6. **Local Dashboard Implementation**

    - Development of a **Streamlit-based** dashboard to display:
        - Current AQI values
        - Forecasts for 24, 48, and 72 hours ahead
        - USAQI category colors and health implications
    - The dashboard will run locally and not be deployed to a public server in this phase.

7. **Model Interpretability**

    - Use of SHAP or similar techniques to identify the most influential features driving predictions.

4.2 Out of Scope

- **Cloud Deployment**

    - No live deployment to production servers; system remains a local prototype.

- **Multi-City Coverage**

    - Limiting predictions to Karachi only; extending to other cities is future work.

5. Data Collection & Sources

5.1 Overview

I built a two-part ingestion pipeline to assemble an **hourly** dataset for Karachi (timezone **Asia/Karachi**) combining (i) **air-quality signals** including **overall USAQI** and pollutant components, and (ii) **meteorological variables** used as exogenous drivers.

- **Historical backfill** (one-time notebook): fetches an hourly range covering multiple months to create karachi_aqi.csv.

- **Daily collector** (script + GitHub Actions): runs once per day to append the last day's hourly readings and push new rows to **Hopsworks Feature Store**.

5.2 Data Providers & Endpoints

Primary provider: Open-Meteo

- **Air quality (hourly):** https://air-quality-api.open-meteo.com/v1/air-quality
    - Hourly variables requested:
        - pm10, pm2_5, carbon_monoxide, nitrogen_dioxide, sulphur_dioxide, ozone, us_aqi
- **Weather (historical archive, hourly):** https://archive-api.open-meteo.com/v1/archive *(used in historical backfill)*
- **Weather (forecast/near-real-time, hourly):** https://api.open-meteo.com/v1/forecast *(used in daily collector for "yesterday")*
    - Hourly variables requested:
        - temperature_2m, relative_humidity_2m, wind_speed_10m, precipitation

Geospatial context:

- **Latitude:** 24.8607, **Longitude:** 67.0011 (Karachi)
- **Timezone:** Asia/Karachi (UTC+05:00). All timestamps are parsed as ISO8601 and localized to this timezone at **hourly** granularity.

Units (Open-Meteo defaults):

- **Temperature:** °C, **Relative humidity:** %, **Wind speed:** km/h, **Precipitation:** mm
- **$PM_{2.5}$/PM_{10}:** μg/m³, **Gaseous pollutants (CO/NO_2/SO_2/O_3):** μg/m³
- **USAQI (us_aqi):** dimensionless index (EPA scale)

Note: Because **Open-Meteo returns us_aqi directly**, our **forecast target is the overall USAQI**. I do **not** predict $PM_{2.5}$/PM_{10} individually in this project phase.

5.3 Historical Backfill (one-time)

Notebook: historical data.ipynb

1. **Air quality pull** (hourly): air-quality-api.open-meteo.com for date range (e.g., 2025-05-01 to 2025-07-11).
2. **Weather pull** (hourly): archive-api.open-meteo.com for the same date range.
3. **Merge on time** (inner join) to produce a single, clean hourly table.
4. **Persist** to karachi_aqi.csv.

Result: A contiguous, hour-aligned historical dataset ready for feature engineering/model training.

5.4 Daily Collection (recurring)

Script: Daily_Collection.py

- Computes **yesterday's** date (start_date=end_date=yesterday) to avoid partial current-day gaps.
- Fetches **air-quality** (air-quality-api) and **weather** (forecast endpoint) for that day, parses timestamps to datetime.

Inner-merge on time, sorts, and **deduplicates** against existing karachi_aqi.csv by excluding any time already present:
new_rows = merged_df[~merged_df["time"].isin(existing_df["time"])]

- **Append** only new_rows, sort by time, and write back to CSV.

Why "forecast" endpoint for the daily job?
For the previous day, Open-Meteo's forecast endpoint reliably serves hourly realized values; using it simplifies daily ingestion. For longer ranges, the **archive** endpoint remains the canonical source (used in backfill).

5.5 Feature Store Integration (Hopsworks)

After each daily run, **only** new rows are uploaded to a **Hopsworks Feature Group**:

- **Project login:** hopsworks.login(...)

- **Feature Group:** karachi_aqi_hourly, version=1
    - **Primary key:** time (hourly timestamp)
    - **Description:** “Hourly AQI + Weather for Karachi”
    - **online_enabled=False** (offline FG; hourly timestamps are best suited to offline storage)

- **Insert:** fg.insert(new_rows, write_options={"wait_for_job": False}) (non-blocking write)

5.6 CI/CD Automation (GitHub Actions)

To ensure the data ingestion pipeline runs without manual intervention, a **GitHub Actions workflow** was implemented for **Continuous Integration/Continuous Delivery (CI/CD)**. The workflow automates the execution of the daily AQI data collection script, updates the local dataset, and synchronizes changes with the repository and the Hopsworks Feature Store.

Schedule:
The workflow is configured to execute **once every day at 13:30 UTC** (equivalent to **6:30 PKT**) using a scheduled cron job. Additionally, it supports **manual triggers** via the workflow_dispatch event in GitHub Actions.

Execution Environment:
Each scheduled run takes place in a clean **Ubuntu-based virtual machine** provisioned by GitHub Actions, with **Python 3.10** installed to match the project requirements.

Process Flow:

1. **Repository Checkout:** The workflow retrieves the latest version of the codebase from GitHub to ensure it runs on the most recent commit.
2. **Dependency Installation:** Installs all necessary Python packages (requests, pandas, and hopsworks) required for the data collection and feature store upload.
3. **Script Execution:** Runs the Daily_Collection.py script, with the **Hopsworks API key** injected securely from **GitHub Secrets** as an environment variable to avoid hard-coding sensitive credentials.
4. **Data Update:** If the script appends new hourly records to karachi_aqi.csv, the updated file is committed and pushed back to the repository.
5. **Feature Store Upload:** Newly collected rows are inserted into the **Hopsworks Feature Group** (karachi_aqi_hourly) for downstream use.

Artifacts Produced:

- An updated karachi_aqi.csv file in the repository containing the latest hourly AQI and weather data.
- Corresponding new rows stored in the **Hopsworks Feature Store** for offline analysis and model training.

Idempotency:
The workflow is designed to avoid redundant commits. If the data collection step detects no new hourly records, it terminates gracefully with the message **"No changes to commit"**.

6.7 Data Dictionary (post-merge hourly table)

Column	Description	Example / Unit
time	Hourly timestamp (Asia/Karachi)	2025-06-15 14:00:00+05:00
us_aqi	**Overall USAQI (target)**	0–500 (dimensionless)
pm2_5, pm10	Particulate matter	µg/m³
carbon_monoxide	CO concentration	µg/m³
nitrogen_dioxide	NO_2 concentration	µg/m³
sulphur_dioxide	SO_2 concentration	µg/m³
ozone	O_3 concentration	µg/m³
temperature_2m	2-m air temperature	°C
relative_humidity_2m	Relative humidity	%
wind_speed_10m	10-m wind speed	km/h
precipitation	Hourly precipitation	mm

I treated **us_aqi** as the **single forecasting target** (overall AQI). Pollutant components are kept for QA/EDA and as potential model features (e.g., lags, rates of change) but are **not** forecast individually.

6.8 Data Quality & Integrity Practices

- **Time alignment:** All data is hourly and merged on exact time.
- **Deduplication:** Avoid double-writes by excluding rows whose time already exists.

- **Sorting:** Data is always sorted ascending by time before saving.
- **Fallbacks:** If either API fails or returns no hours, the job skips appending and exits cleanly (logged).

6. Exploratory Data Analysis (EDA) - Before Data Preprocessing and Feature Engineering

Before proceeding with data preprocessing and feature engineering, a comprehensive **Exploratory Data Analysis (EDA)** was conducted to understand the structure, distribution, trends, and relationships present in the dataset. This step was critical for identifying potential data quality issues, uncovering underlying patterns, and guiding the selection of features for subsequent model training.

6.1 Data Loading and Initial Inspection

The dataset, containing historical air quality and meteorological measurements for Karachi, was imported and inspected to:

- Verify the data schema (column names, data types, and non-null counts).
- Confirm the presence of the **Air Quality Index (AQI)** as the primary target variable.
- Review available predictor variables, which included meteorological parameters such as temperature, relative humidity, wind speed, wind direction, and other atmospheric conditions retrieved from the **Open-Meteo API**.

This initial inspection helped confirm that the dataset was suitable for AQI prediction and contained relevant time-series information.

6.2 Time-Series Structure and Missing Values

Given that AQI prediction is inherently a **time-series forecasting** problem, the time column was converted to a datetime format and set as the index for easier temporal analysis. The dataset was checked for:

- **Time continuity**: Identifying any missing hourly records.
- **Null or missing values**: Detecting gaps in AQI and meteorological variables.
- **Outlier timestamps**: Ensuring that the time range was consistent without duplicated or erroneous entries.

Findings indicated the presence of occasional missing values and short data gaps, which were later addressed during preprocessing through interpolation and imputation methods.

6.3 Statistical Summary

A statistical overview (describe() function) was performed to examine:

- Central tendency measures (mean, median).

- Dispersion indicators (standard deviation, variance).
- Minimum and maximum values for each feature.
- Detection of extreme values that could be potential **outliers** in AQI or meteorological measurements.

This analysis revealed natural variations in AQI readings over time, along with fluctuations in temperature, humidity, and wind parameters, which were expected to influence air quality.

6.4 Univariate Analysis

Each variable was examined individually to assess its distribution and potential predictive power:

- **Histograms and KDE plots** were used to understand feature distributions.
- AQI values displayed a right-skewed distribution, with the majority falling within moderate ranges but with several high-AQI episodes corresponding to poor air quality days.
- Meteorological features such as temperature and humidity exhibited seasonal patterns.

6.5 Temporal Trend Analysis

To understand the evolution of AQI over time:

- **Line plots** were generated for AQI across the available time range.
- Seasonal and diurnal patterns were visually identified, with AQI often peaking during specific times of the day and showing noticeable changes between months.
- AQI spikes were often correlated with stagnant wind conditions or specific weather events.

6.6 Bivariate and Correlation Analysis

Relationships between AQI and meteorological variables were explored:

- **Pearson correlation heatmaps** were generated to quantify linear relationships.
- Variables such as **wind speed** exhibited a negative correlation with AQI, suggesting that higher wind speeds tend to disperse pollutants and reduce AQI levels.
- Relative humidity, temperature, and wind direction also showed varying degrees of correlation with AQI.

These insights informed later feature engineering and model selection decisions.

6.7 Outlier Detection

Potential outliers in AQI readings were detected through:

- **Boxplots** for visual inspection of extreme values.
- Identification of short-term pollution spikes possibly caused by episodic events such as traffic congestion, industrial emissions, or weather anomalies.

While some outliers represented genuine pollution episodes, others were attributed to data collection noise and were addressed in preprocessing.

6.8 Key Insights from EDA

The EDA process provided several critical takeaways:

1. The dataset is sufficiently rich for AQI prediction, containing relevant meteorological predictors from the Open-Meteo API.
2. AQI exhibits both **short-term variability** and **seasonal trends**, making time-aware modeling approaches suitable.
3. Certain meteorological factors (wind speed, humidity, temperature) are likely to play a significant role in AQI forecasting.
4. Missing data and occasional extreme readings require robust preprocessing strategies.
5. Clear temporal dependencies suggest that **time-series models** or regression models with lagged features would be appropriate.

7. Data Preprocessing & Feature Engineering

This section describes the **hourly** preprocessing pipeline that transforms the raw merged table (karachi_aqi.csv) into a modeling-ready feature matrix and publishes engineered rows to the **Hopsworks Feature Store**. The pipeline is designed to be **incremental** (processes only new hours), **idempotent**, and **automation-friendly** (runs hourly via GitHub Actions).

7.1 Inputs and Outputs

- **Input**: karachi_aqi.csv (hourly time series with time, us_aqi, pollutant components, and meteorology from Open-Meteo).
- **Output**: preprocessed_aqi_data.csv (feature-enriched table, sorted by time, no duplicate timestamps).
- **Output (feature store)**: Hopsworks **Feature Group** aqi_features (version 1), primary key = time.

7.2 Time Indexing & Basic Derivations

1. **Timestamp normalization**
    - Parse time to datetime, set as **index** during feature engineering to simplify lag/rolling operations.
    - After processing, reset the index and persist time as a column (string-formatted) for CSV storage.

2. **Calendar/time features**
    - hour, day_of_week, month.
    - **Cyclic encodings** for diurnal seasonality:
        - hour_sin = $\sin(2\pi \cdot \text{hour}/24)$
        - hour_cos = $\cos(2\pi \cdot \text{hour}/24)$

          These capture cyclical patterns without artificial discontinuities at 23 to 0.

7.3 Target-Aware Temporal Features (Overall USAQI)

Forecast target: us_aqi (overall USAQI), not individual pollutants.

- **Lags**: us_aqi_lag1, us_aqi_lag6, us_aqi_lag24 (short, medium, day-back memory).
- **Rolling statistic**: us_aqi_roll3 (3-hour mean to smooth micro-noise).
- **Change rates**:
    - us_aqi_diff = us_aqi − us_aqi_lag1 (1-hour delta)
    - Downstream models benefit from momentum/mean-reversion signals.

All lag/rolling features are computed **strictly from the past** to prevent leakage.

7.4 Exogenous / Interaction Features

- **Pollutant deltas**: pm2_5_diff (captures short-term spikes).
- **Nonlinear transformations** (variance stabilization, long-tail control):
    - log_pm2_5, log_pm10, log_carbon_monoxide, log_ozone, log_sulphur_dioxide, log_nitrogen_dioxide via **log1p**.
    - Zeros are replaced with NaN for transform, then filled back with 0 after log1p.
- **Cross-terms** (first-order physically plausible interactions):
    - ozone_per_humidity = ozone / (relative_humidity_2m + 1) — accounts for photochemical ozone potential dampened by high humidity.
    - pm2_5_temp_interaction = pm2_5 * temperature_2m — hotter, stagnant hours can worsen particulate effects.

These interaction terms let linear/GBM models approximate known nonlinearities without over-engineering.

7.5 Missingness & Row Filtering

- After constructing lags/rollings, **rows with any NaN** (from initial windows) are **dropped** to guarantee a fully dense design matrix for training and serving.
- This is intentional: it sacrifices the **first** few hours (needed to create lags) but avoids downstream imputation complexity and training/serving skew.

7.6 Feature Scaling

During the preprocessing stage, after handling missing values and performing feature engineering, I applied **scaling** to selected skewed features. The primary reason for scaling was to **normalize the range of feature values** so that no single variable dominated the learning process due to larger magnitudes. Scaling is particularly important for algorithms that are sensitive to feature magnitudes (e.g., Ridge Regression, KNN, Neural Networks), as it improves convergence speed and can enhance model performance.

In this case, skewed features were scaled to reduce the influence of outliers and bring them closer to a standard distribution, which also helps in making statistical comparisons more meaningful.

However, in hindsight, it would have been **better to perform scaling during the model training stage (which i have done in later models)** rather than in the preprocessing script. Doing it during model training ensures that scaling is applied **only to the training set** (using training set statistics), and the same transformation is then applied to the test set. This avoids **data leakage**—a scenario where information from the test set unintentionally influences the training process—thereby ensuring more realistic and unbiased performance evaluation.

To provide scale-invariant inputs especially useful for **Ridge** and **LSTM** the pipeline applies **StandardScaler (z-score)** to the selected columns:

- **Raw continuous features**:
  pm2_5, pm10, carbon_monoxide, ozone, sulphur_dioxide, nitrogen_dioxide, temperature_2m, relative_humidity_2m, wind_speed_10m, us_aqi_lag1, us_aqi_lag6, us_aqi_lag24, us_aqi_roll3, us_aqi_diff, pm2_5_diff, ozone_per_humidity, pm2_5_temp_interaction
- **Logged features**:
  log_pm2_5, log_pm10, log_carbon_monoxide, log_ozone, log_sulphur_dioxide, log_nitrogen_dioxide

Scaled columns are prefixed with scaled_ (e.g., scaled_pm2_5).

7.7 Incremental Processing & Idempotency

- The script reads the **previous** preprocessed_aqi_data.csv (if present) and identifies **new** raw hours—based on time—from karachi_aqi.csv.
- Only **new rows** are engineered, concatenated with the existing preprocessed dataset, **de-duplicated** on time, **sorted**, and written back.
- This keeps runtime low and avoids re-processing historical data.

7.8 Persistence & Formatting

- Output CSV uses time formatted as DD/MM/YYYY HH:MM for readability in spreadsheets (e.g., 14/08/2025 18:00).

7.9 Feature Store Publishing (Hopsworks)

- Auth via HOPSWORKS_API_KEY (injected as environment variable; **never hard-coded**).
- **Feature Group**: aqi_features (v1), **primary_key** = time.
- After each run, only the **newly engineered rows** are inserted (fg.insert(df_new_processed, write_options={"wait_for_job": False})).
- This creates a single source of truth for downstream training/evaluation jobs and supports future automation.

7.10 CI/CD (Hourly Preprocessing Automation)

- A GitHub Actions workflow triggers **every hour** and on manual dispatch.
- Steps: checkout then install deps then run preprocessing script then commit updated preprocessed_aqi_data.csv then push; Hopsworks insertion happens inside the script.
- **Idempotent behavior**: if no new raw hours exist, the job exits without committing.

7.11 Quality & Governance Considerations

- **Leakage control**: Lags/rollings computed only from prior timestamps; no look-ahead.
- **Determinism**: Fixed transformations; recommend pinning numpy/pandas/scikit-learn versions in requirements.txt.
- **Monitoring hooks (recommended next)**:
    - Row counts per hour; missing-rate alarms.
    - Distribution drift checks on key features (e.g., us_aqi, wind_speed_10m).
    - Schema guard (verify expected columns/types before insert).

7.12 Feature Set Summary (post-processing)

- **Time & cycles**: hour, day_of_week, month, hour_sin, hour_cos
- **Target memory**: us_aqi_lag1, us_aqi_lag6, us_aqi_lag24, us_aqi_roll3, us_aqi_diff
- **Pollutant signals**: raw (pm2_5, pm10, carbon_monoxide, ozone, sulphur_dioxide, nitrogen_dioxide), deltas (pm2_5_diff), logs (log_*)
- **Weather**: temperature_2m, relative_humidity_2m, wind_speed_10m, precipitation
- **Interactions**: ozone_per_humidity, pm2_5_temp_interaction
- **Scaled counterparts**: scaled_* for all selected numeric features

Together, these features provide **short-term persistence**, **diurnal structure**, **meteorological drivers**, and **basic nonlinearities** required for accurate **overall USAQI** forecasting at 24/48/72-hour horizons.

8. Exploratory Data Analysis (after Feature Engineering) and Feature Selection

After completing the feature engineering phase — which involved creating temporal features (hour, day of week, month), cyclic encodings, and lag-based features (1-hour, 6-hour, 24-hour lags) — I proceeded with **post-feature-engineering EDA** to understand the structure and relevance of the generated features.

8.1 Dataset Overview after Feature Engineering

The engineered dataset contained both the original air quality and weather variables, along with additional features designed to capture temporal patterns and short- to medium-term dependencies. This included:

- **Time-based features**: hour, day_of_week, month
- **Cyclic encodings**: hour_sin, hour_cos
- **Lag features**: us_aqi_lag1, us_aqi_lag6, us_aqi_lag24
- **Weather-related predictors**: temperature, humidity, wind speed, etc.
- **Interaction terms**: combinations of environmental and temporal features

The post-engineering EDA step ensured that there were no missing values after lag filling, and distributions were examined for normality, skewness, and potential outliers.

8.2 Correlation and Redundancy Analysis

A Pearson correlation matrix was generated to identify highly correlated variables. Features with absolute correlation above a threshold (e.g., $|r| > 0.9$) with each other or with the target were flagged for review. This step reduced redundancy and multicollinearity in the dataset, which is especially important for tree-based models and linear regression variants.

8.3 Feature Importance Ranking (Initial Selection in EDA)

Before automating the process, **feature selection was first implemented in the EDA file itself** using model-based importance scores. Two main approaches were used:

- **Tree-based importance**: Random Forest and XGB feature importance rankings
- **Statistical filtering**: Mutual information scores between features and the target

This allowed us to narrow down the features to those contributing the most predictive value.

8.4 Automation of Selected Features

To streamline and standardize the process, the **automate_selected_features.py** script was created as a **separate Python file**.
This script:

1. Loads the latest engineered dataset from Hopsworks or local storage.
2. Applies the same feature selection logic developed in the EDA stage.
3. Saves the resulting reduced feature set to a CSV file named automated_selected_features_for_modeling.csv.

This automation ensures that each time new data is ingested and engineered, the selection step runs consistently without manual intervention.

8.5 Continuous Update via GitHub Actions

A **GitHub Actions workflow** was configured to:

- Run the **automate_selected_features.py** script every hour (via CRON schedule) or manually through the GitHub UI.
- Commit and push the updated automated_selected_features_for_modeling.csv back to the repository.

This setup guarantees that the selected features file is always up to date, which is critical for real-time or frequent retraining scenarios.

8.6 Final Outcome

The result of this process is a **clean, consistently updated, and model-ready feature set** that integrates both the domain-driven engineering decisions and data-driven selection criteria. The separation of concerns, EDA for exploration, automate_selected_features.py for automation, and GitHub Actions for orchestration — makes the pipeline modular, maintainable, and production-friendly.

9. Feature Selection - Approach 2

Note on Feature Selection Changes
Initially, I performed EDA after feature engineering and applied my first feature selection approach, as detailed above. However, when I trained models with those features, the results were unsatisfactory, with significantly worse metrics than expected.

To address this, I revised my feature selection technique and made minor preprocessing changes. I also adjusted preprocessing within specific model training scripts wherever it was necessary to tailor the features to that model's needs.

9.1 Objective

Improve model performance (after poor metrics with the first selection) by re-evaluating features using a leakage-safe, time-series-aware selection flow. The aim was to keep strong **target-memory**, **meteorology**, and a **minimal pollutant signal** without redundancy.

9.2 Inputs

- **Preprocessed table:** preprocessed_aqi_data.csv (from Section 7), including:
    - Target: **us_aqi** (overall USAQI)
      Engineered features: lags/rollings (us_aqi_lag*, us_aqi_roll3, us_aqi_diff), time cycles (hour_sin, hour_cos), interactions (pm2_5_temp_interaction, ozone_per_humidity), weather, and log transforms.
- **Note:** Any clipping/winsorization at the 99th percentile was handled in your *automated selected features* script, not in this notebook.

9.3 Method

1. **Sanity cleanup & non-feature removal**

    - Dropped non-predictor columns (e.g., ID/datetime if present in X), kept **us_aqi** separate as the target.
    - Ensured **no future leakage** (all lagged/rolling features are computed with .shift() in earlier preprocessing).

2. **Redundancy pruning via correlation**

    - Computed a correlation matrix on candidate predictors; built an upper triangle mask.

- Removed columns with **high pairwise correlation** (threshold used ≈ **0.95**), retaining one representative per correlated cluster to stabilize linear models and avoid overfitting in tree/boosting models.

3. **Model-based importance with time-series CV**

   - Trained a **LightGBM regressor** using **TimeSeriesSplit** (rolling/forward splits) to respect temporal order.
   - Ranked features by **gain/importance** averaged across folds to emphasize consistently useful predictors and de-emphasize unstable ones.
   - (Where a tie existed, preference was given to simpler/less-collinear features and those grounded in domain logic, e.g., us_aqi_lag1, us_aqi_roll3, wind effects.)

4. **Leakage guard**

   - Confirmed that all target-dependent features are **strict lags/differences** (no contemporaneous us_aqi as input).
   - Time-series CV ensured validation folds only see *past* data during training.

5. **Refinement & parsimony**

   - Kept the smallest set that preserved performance: strong **target memory**, minimal but meaningful **pollutant signals** (logs + one interaction), **meteorology**, and **time cycles**.

9.4 Why this improved over Approach 1

- **Correlation pruning** removed redundant scaled/log duplicates that confused linear models and added noise for trees.
- **Time-series-aware importance** surfaced features that generalize across periods (not just one window).
- **Parsimony** lowered variance and improved stability across horizons (24/48/72h).

9.5 Outputs

- **Final feature list (Approach 2):** used for subsequent experiments and training scripts.

```
[
  "us_aqi_roll3",
```

```
  "us_aqi_lag1",
  "us_aqi_diff",
  "pm2_5_diff",
  "log_carbon_monoxide",
  "log_pm10",
  "log_nitrogen_dioxide",
  "us_aqi_lag6",
  "log_sulphur_dioxide",
  "log_pm2_5",
  "us_aqi_lag24",
  "wind_speed_10m",
  "pm2_5_temp_interaction",
  "hour_sin",
  "ozone_per_humidity",
  "ozone",
  "temperature_2m",
  "relative_humidity_2m",
  "hour",
  "day_of_week",
  "hour_cos",
  "month",
  "precipitation"
]
```

9.6 Automation Note

This version was not automated and everytime a model was run, features were selected from the preprocessed csv based on the final features.json or whatever gave better metrics.

10. Model Training

Now we transition from feature selection to the **model training and evaluation phase**.
The selected features from the previous steps are now used to train machine learning models that can forecast the Air Quality Index (AQI) over multiple future horizons.
We begin with the **LightGBM Multi-Output Regressor**, a powerful gradient boosting framework optimized for speed and efficiency, particularly suitable for large datasets and tabular features.
This approach predicts all target horizons in one go, capturing correlations between them and improving long-range forecasting stability.

LightGBM Multi-Output Model - Detailed Analysis

10.1 Objective

Forecast US AQI for the next 72 hours (multi-step) using a sliding 24-hour window of engineered features, trained as a single multi-output regressor (one fit that predicts all 72 horizons).

10.2 Data & features

- Dataset: preprocessed_aqi_data (3).csv (project root).
- Feature list: loaded from final_feature_list.json.
- Target column: us_aqi.
- Time column: auto-detected from ["time", "datetime"], parsed day-first (so 4/8/25 ⇒ 4 Aug 2025), then sorted chronologically.

10.3 Preprocessing & sequence building

1. **Train-only winsorization & imputation (applied correctly to avoid leakage):**
    - Compute 1%/99% caps on training inputs (per-feature), then clip both train/val to those caps.
    - Compute train medians (per feature) and impute missing values in train/val with those medians.
      (Saved into metadata.json as winsor_low, winsor_high, train_medians.)

2. **Sliding-window sequences:**
    - Window size (WINDOW_SIZE) = 24 past hours.
    - Horizon (PREDICT_HORIZON) = 72 hours ahead.

- For each time step *i*, the input is the last 24 rows of all features flattened oldest→newest, and the label is a length-72 vector [t+1 … t+72] of us_aq**i.**

3. **Split: chronological 80/20 train/validation (no shuffle).**

10.4 Model

- **Wrapper:** MultiOutputRegressor (parallel fits per horizon under the hood).
- **Base learner:** lightgbm.LGBMRegressor.
- **Hyperparameters :**

```
n_estimators=1200,
learning_rate=0.03,
num_leaves=31,
min_child_samples=50,
subsample=0.8,
colsample_bytree=0.8,
reg_alpha=0.5,
reg_lambda=0.8,
random_state=42
```

10.5 Training

- Fit once on (X_train, y_train) where y_train is shape [n_samples, 72].
- Inference returns a vector of **72 predictions** per input window.

10.6 Evaluation (validation 80/20)

Averaged across all 72 horizons:

- **MAE: 7.365**
- **RMSE: 8.563**
- **R^2: −1.868**

Early-horizon MAE (first 12 hours):
[0.538, 0.669, 0.757, 0.820, 0.953, 1.190, 1.386, 1.582, 1.760, 1.996, 2.188, 2.421]

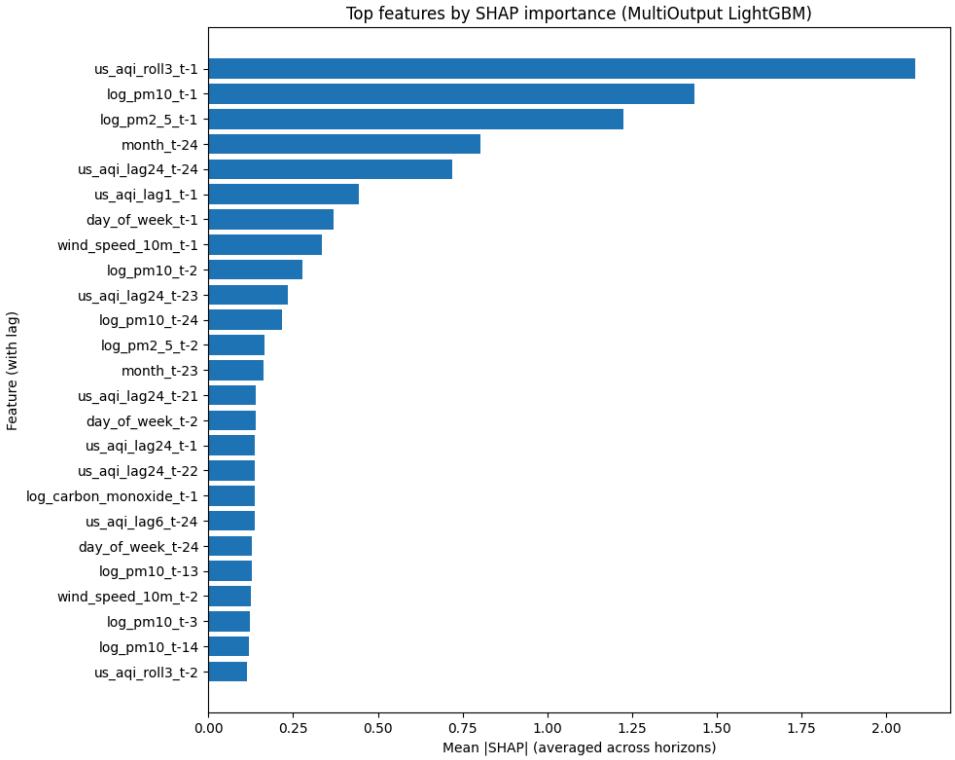
Specific horizons:

- **24h MAE: 5.072**
- **48h MAE: 9.165**
- **72h MAE: 12.752**

Interpretation: early horizons are reasonable, but **errors grow with horizon**, and the **overall negative R^2** indicates the model underperforms a naïve baseline across the full 72-hour span on this split (likely distribution shift, feature/target scaling challenges at long range, or insufficient long-horizon signal).

10.7 Explainability (SHAP)

- Computes **TreeSHAP** on a subsample, per horizon, then aggregates **mean |SHAP| across all 72 horizons** back onto the **flattened lagged features** (e.g., featureName_t-3).
- Produces a **Top-25 bar chart** ("Top features by SHAP importance (MultiOutput LightGBM)").



10.8 Artifacts saved

- **Forecast CSV:** aqi_72h_forecast.csv
  - Timestamps formatted **d/m/yy HH:MM**; in this run: **10/8/25 00:00 to 12/8/25 23:00**.

- **Model:** models/current/lgbm_multioutput_72h.joblib
- **Metadata:** models/current/metadata.json (features, time/target col, window/horizon, day-first flag, winsor caps, train medians).

LSTM (Multi-Output, 72h)

10.1 Objective

Forecast US AQI for the next 72 hours using a single LSTM trained to predict all horizons at once from a 24-hour sliding window of engineered features.

10.2 Data & features

- **Dataset:** preprocessed_aqi_data (3).csv (project root).
- **Feature list:** final_feature_list.json.
- **Target column:** us_aqi.
- **Time column:** auto-detected from ["time", "datetime"], parsed day-first (so 4/8/25 ⇒ 4 Aug 2025) and then sorted chronologically.

10.3 Preprocessing & sequence building

1. **Train-only winsorization, imputation, scaling (to avoid leakage):**
    - Compute 1%/99% caps on training-input rows only (rows that feed the first 80% of windows) and clip all rows to those caps.
    - Compute medians on training-input rows and impute missing values in all rows with those medians.
    - Standardize features with StandardScaler fitted on training-input rows; apply to all rows.
2. **Sliding-window sequences:**
    - **Window (WINDOW_SIZE) =** 24 past hours.
    - **Horizon (PREDICT_HORIZON) =** 72 hours ahead.
    - **For each index *i*:** input X is shape (24, F) (oldest→newest), label y is length-72 vector [t+1 … t+72].
3. **Split:** chronological 80/20 without shuffle using train_test_split(..., shuffle=False)**.**

10.4 Model

- **Architecture:**
  - **LSTM:** input_size = len(feat_cols), hidden_size = 128, num_layers = 2, dropout = 0.2 (applied only when layers>1), batch_first=True.
  - **Head:** Linear(128 → 256) → ReLU → Linear(256 → 72) (one logit per horizon).

- **Loss & optimization:**
  - **Loss:** SmoothL1Loss (Huber).
  - **Optimizer:** AdamW.
  - **LR scheduler:** ReduceLROnPlateau(factor=0.5, patience=3).
  - **Device:** auto (cuda if available, else CPU).
  - **Reproducibility:** np.random.seed(42) and torch.manual_seed(42).

- **Training hyperparameters :**

```
EPOCHS        = 40

BATCH_SIZE    = 256

LR            = 1e-3

WD            = 1e-4

PATIENCE      = 8

HIDDEN_SIZE   = 128

NUM_LAYERS    = 2

DROPOUT       = 0.2

RANDOM_SEED   = 42
```

10.5 Training

- Dataloaders use shuffle=False (chronology preserved).
- Validation MAE improved sharply early on; best validation MAE was reached around epoch 12 (~4.439), after which early stopping triggered at epoch 20 and the best weights were restored.

10.6 Evaluation (validation 80/20)

Averaged across all 72 horizons:

- **Train:** MAE 10.937, RMSE 13.375, R^2 −0.001
  First 12h MAE: [11.205, 11.201, 11.195, 11.187, 11.176, 11.167, 11.161, 11.149, 11.140, 11.140, 11.123, 11.115]
  24h MAE: 11.046 · 48h: 10.841 · 72h: 10.667
- **Validation:** MAE 5.677, RMSE 7.230, R^2 −0.719
  First 12h MAE: [4.489, 4.529, 4.567, 4.599, 4.613, 4.648, 4.700, 4.717, 4.753, 4.849, 4.841, 4.879]
  24h MAE: 5.361 · 48h: 5.982 · 72h: 6.737

Interpretation: early horizons are modest, but the average R^2 is negative, and errors rise with horizon, indicating the LSTM (as configured) underperforms a naïve baseline overall. This can stem from limited long-range signal, heavy regularization, or the generic head struggling to retain detailed horizon-specific information.

10.7 Explainability

- No SHAP in this notebook

10.8 Artifacts saved

- **Forecast CSV:** predictions/lstm_predicted_aqi_72hrs.csv
  - **Timestamp range saved as** 10/8/25 00:00 to 12/8/25 23:00 (format d/m/yy HH:MM).
- **Model weights:** models/current/lstm_multioutput_72h.pt
- **Scaler:** models/current/scaler.joblib
- **Metadata:** models/current/metadata.json
  (stores feature list, time/target columns, window/horizon, day-first flag, winsor caps, train medians, scaler path).

LSTM (Multi-Output, 72h) - Optuna-tuned

10.1 Objective

Forecast US AQI for the next 72 hours from a 24-hour sliding window of engineered features using a single multi-output LSTM (one forward pass predicts all 72 horizons).

10.2 Data & features

- Dataset: preprocessed_aqi_data (3).csv
- Feature list: final_feature_list.json
- Target column: us_aqi
- Time column: auto-detected from ["time", "datetime"], parsed day-first (e.g., 4/8/25 means 4 Aug 2025) and sorted chronologically.

10.3 Preprocessing & sequence building

1. **Leak-safe cleaning (train-inputs only):**
    - Winsorize inputs at 1%/99% using only the rows that feed the training windows, then clip all rows to those caps.
    - Median imputation computed on training-input rows, applied to all rows.
    - Standardize features with StandardScaler fit on training-input rows, transform all rows.
2. **Sliding windows:**
    - Window (WINDOW_SIZE) = 24 past hours.
    - Horizon (PREDICT_HORIZON) = 72 future hours.
    - Each sample: X shape (24, F) (oldest→newest), label y length-72 [t+1 … t+72].
3. **Split: chronological 80/20 train/validation (shuffle=False).**

10.4 Model

- **Architecture:**
    - nn.LSTM(input_size=F, hidden_size=H, num_layers=L, dropout=D if L>1 else 0, batch_first=True, bidirectional=flag)
    - Head: Linear(out_dim → 256) → ReLU → Linear(256 → 72) where out_dim = H * (2 if bidirectional else 1) and we use the last time step.
- **Loss & optimization:**
    - SmoothL1Loss (Huber), AdamW, LR scheduler ReduceLROnPlateau(factor=0.5, patience=3), gradient clip max_norm=2.0.
    - Reproducibility: seed=42; device auto-select (cuda if available).

10.5 Optuna search

- **Trials: 25**
- **Objective:** minimize validation MAE (averaged across 72 horizons).
- **Best trial:** #24 with Val MAE = 4.1470
- **Best params:**
    - **hidden_size = 96**
    - **num_layers = 1**
    - **dropout = 0.1842**
    - **bidirectional = True**
    - **lr = 9.09e-4**
    - **batch_size = 256**

10.6 Training

- Retrained the best Optuna configuration on the training split with early stopping (patience=8).
- **Best epoch used:** 13 (early stop triggered later).
- Best saved Val MAE during retrain: 4.1891.

10.7 Evaluation (validation 80/20)

Averaged across all 72 horizons:

Train:

- MAE: 8.089
- RMSE: 10.938
- R^2: 0.289
- First 12h MAE: [5.882, 5.685, 5.525, 5.341, 5.257, 5.173, 5.099, 5.083, 5.063, 5.106, 5.081, 5.183]
- 24h MAE: 6.630 · 48h: 9.378 · 72h: 11.201

Validation:

- MAE: 4.864
- RMSE: 5.825
- R^2: −0.156
- First 12h MAE: [2.732, 2.636, 2.538, 2.471, 2.398, 2.388, 2.375, 2.407, 2.471, 2.552, 2.610, 2.681]
- 24h MAE: 4.203 · 48h: 5.960 · 72h: 6.658

Interpretation: Optuna tuning significantly improves early-horizon accuracy vs. my earlier LSTM run, and bi-directional + smaller hidden state with lower LR helped validation MAE. However, average R^2 remains negative over 72 horizons (underperforming a naïve mean/last baseline overall), reflecting the typical error growth with horizon in direct multi-output setups.

10.8 Explainability

- No SHAP (tree-specific).

10.9 Artifacts saved

- **Forecast CSV:** predictions/lstm_predicted_aqi_72hrs.csv
    - Timestamps in this run: 10/8/25 00:00 to 12/8/25 23:00 (d/m/yy HH:MM).
- **Model weights:** models/current/lstm_multioutput_72h.pt
- **Scaler:** models/current/scaler_lstm.joblib
- **Metadata:** models/current/metadata_lstm.json
  (feature list, time/target columns, window/horizon, day-first flag, winsor caps, train medians, scaler path, and chosen hyperparameters).

TCN (Multi-Output, 72h)

10.1 Objective

Forecast US AQI for the next 72 hours using a single multi-output TCN that consumes a 24-hour sliding window of engineered features and emits all 72 horizons in one forward pass.

10.2 Data & features

- Dataset: preprocessed_aqi_data (3).csv
- Feature list: final_feature_list.json
- Target column: us_aqi
- Time column: auto-detected from ["time", "datetime"], parsed day-first (e.g., 4/8/25 → 4 Aug 2025) and sorted chronologically.

10.3 Preprocessing & sequence building

1. **Leak-safe cleaning (training inputs only):**
    - Winsorization at 1% / 99% on training rows → clip train/val to those caps.
    - Median imputation computed on training inputs → applied to train/val.
    - Standardization with StandardScaler fit on training inputs → applied to all inputs.

2. **Sliding-window sequences:**
    - Window (WINDOW_SIZE) = 24 hours (oldest→newest).
    - Horizon (PREDICT_HORIZON) = 72 hours ([t+1 … t+72]).
    - Each training sample: X has shape (24, F); label y is length-72.
3. **Split: chronological 80/20 train/validation (no shuffle).**

10.4 Model

- **Backbone:** Temporal Convolutional Network (dilated causal 1-D convs with residual blocks).
- **Architecture (as used):**
    - Blocks / levels: 6 residual TCN blocks with exponentially increasing dilation (1, 2, 4, 8, 16, 32).
    - Channels per block: [64, 64, 64, 128, 128, 128].
    - Kernel size: 3 (causal padding).
    - Dropout: 0.2 inside residual blocks.
    - Activation: ReLU.
    - Head: Global last-step features → Linear(last_channels → 256) → ReLU → Linear(256 → 72).
- **Loss & optimization:**
    - Loss: SmoothL1Loss (Huber).
    - Optimizer: AdamW.
    - LR scheduler: ReduceLROnPlateau(factor=0.5, patience=3).
    - Gradient clipping: max_norm=2.0.
    - Seed/device: seed=42; CUDA if available.
- **Training hyperparameters :**

```
EPOCHS          = 50

BATCH_SIZE      = 256

LR              = 2e-3

WD              = 1e-4
```

```
PATIENCE         = 8

DROPOUT          = 0.25

CHANNELS         = [128, 128, 128, 128]    # dilations 1,2,4,8 ->
receptive field 31 (>24)

KERNEL_SIZE      = 3

RANDOM_SEED      = 42
```

10.5 Training

- Dataloaders preserve sequence order (shuffle=False).
- Validation MAE plateaued after the first handful of epochs; early stopping restored the best-val weights.

10.6 Evaluation (validation 80/20)

Averaged over all 72 horizons

- Train: MAE 9.460, RMSE 12.714, R^2 0.057
  First 12h MAE: [0.721, 1.357, 1.833, 2.375, 2.830, 3.338, 3.575, 3.918, 4.211, 4.513, 4.954, 5.134]
  24h MAE: 8.535 · 48h: 11.898 · 72h: 13.199
- Validation: MAE 5.278, RMSE 6.172, R^2 −0.408
  First 12h MAE: [0.400, 0.686, 0.900, 1.296, 1.527, 2.030, 1.897, 2.398, 2.297, 2.573, 2.705, 2.980]

Interpretation: TCN captures local temporal patterns better than a plain MLP head but still shows error growth with horizon. The negative validation R^2 across 72 steps indicates that- on average—this configuration underperforms a naïve baseline over the full span, despite strong early-horizon MAEs.

10.7 Explainability

- CNN/TCN models don’t support TreeSHAP

10.8 Artifacts saved

- Forecast CSV: predictions/tcn_predicted_aqi_72hrs.csv
    - Timestamps for the saved forecast span 10/8/25 00:00 → 12/8/25 23:00 (d/m/yy HH:MM).
- Model weights: models/current/tcn_multioutput_72h.pt
- Scaler: models/current/scaler_tcn.joblib
- Metadata: models/current/metadata_tcn.json
  (feature list, time/target columns, window/horizon, day-first flag, winsor caps, train medians, scaler path, and TCN hyperparameters).

Random Forest (Multi-Output, 72h)

10.1 Objective

Predict US AQI for the next 72 hours using a single multi-output regressor trained on a 24-hour sliding window of engineered features (one fit that outputs all 72 horizons).

10.2 Data & features

- Dataset: preprocessed_aqi_data (3).csv
- Feature list: final_feature_list.json
- Target column: us_aqi
- Time column: auto-detected from ["time","datetime"], parsed day-first (e.g., 4/8/25 → 4 Aug 2025), then sorted chronologically.

10.3 Preprocessing & sequence building

1. **Leak-safe cleaning (computed on training inputs only):**
    - Winsorization at 1% / 99%, clip train/val to those caps.
    - Median imputation per feature using training medians → applied to train/val.
    - (No scaling required for RF; tree models are scale-invariant.)
2. **Sliding windows:**
    - Window (WINDOW_SIZE) = 24 past hours (oldest→newest).
    - Horizon (PREDICT_HORIZON) = 72 hours ahead ([t+1 … t+72]).
    - Each sample: X = flattened 24×F window; y = length-72 vector.
3. **Split: chronological 80/20 train/validation (no shuffle).**

10.4 Model

- **Wrapper:** MultiOutputRegressor (fits one RF per horizon under the hood).
- **Base learner:** sklearn.ensemble.RandomForestRegressor.
- **Hyperparameters:**

```
n_estimators=400,

max_depth=None,

min_samples_leaf=1,

min_samples_split=2,

max_features=None,

random_state=RANDOM_SEED,

n_jobs=-1,

bootstrap=True
```

10.5 Training

- One pass on (X_train, y_train) with 72 parallel treesets (one per horizon).
- Inference returns 72 values per input window.

10.6 Evaluation (validation 80/20)

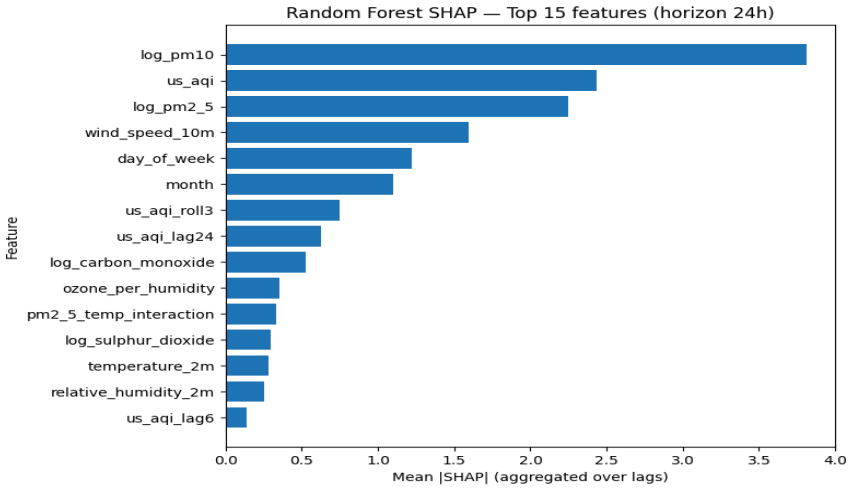
Averaged across all 72 horizons:

- Train: MAE 0.929, RMSE 1.452, R^2 0.988
- Validation: MAE 7.576, RMSE 8.928, R^2 −1.985

Early-horizon MAE (first 12 hours):
[0.31, 0.337, 0.356, 0.387, 0.412, 0.446, 0.486, 0.519, 0.54, 0.567, 0.601, 0.629]

> **Interpretation:** As expected for direct multi-output trees, early horizons are very strong (small MAEs), but errors grow with horizon and the overall R^2 becomes negative across all 72 steps i.e., underperforming a naïve baseline on average for the full span. The big train–val gap also signals overfitting (each horizon's forest can memorize short-term structure from the 24-hour window).

10.7 Explainability



10.8 Artifacts saved

- Model: models/current/rf_multioutput_72h.joblib
- Metadata: models/current/metadata_rf.json (feature list, window/horizon, winsor caps, train medians, parsed-time settings, RF params)
- Forecast CSV: predictions/rf_predicted_aqi_72hrs.csv (72 hourly timestamps starting from the last training window)

SARIMAX (Single-Output per Horizon, 72h) - Best Model

10.1 Objective

Forecast **US AQI for the next 72 hours** by fitting a **Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors (SARIMAX)** model, using the most recent AQI history (and optionally exogenous features) to capture both short-term autocorrelation and seasonal cycles.

10.2 Data & features

- **Dataset:** preprocessed_aqi_data (3).csv
- **Target column:** us_aqi
- **Time column:** auto-detected from ["time","datetime"], parsed as **day-first** (e.g., 4/8/25 means 4 Aug 2025`), sorted chronologically.
- **Features used:**
    - **Endogenous:** past values of us_aqi itself.

- **No additional exogenous predictors** were used in this run — the model relies entirely on AQI's own historical pattern.

10.3 Preprocessing

1. **Hourly index:** Converted timestamps to hourly frequency using .asfreq("H").
2. **Missing values:**
   - Internal gaps up to **48 hours**: filled by **time-based interpolation**.
   - Remaining edges: forward/backward filled.
3. **Train/validation split:** 80/20 split on chronological order — no shuffling.

10.4 Model

- **Model type:** statsmodels.tsa.statespace.SARIMAX.
- **Order parameters:**
  - **Non-seasonal order (p, d, q) = (1, 0, 1)**
  - **Seasonal order (P, D, Q, s) = (1, 1, 1, 24)** — daily seasonality in hourly data.
- **Other settings:** enforce_stationarity=False, enforce_invertibility=False.
- **Fitting:** Maximum likelihood estimation (fit(disp=False)).

10.5 Forecasting strategy

- The model is **one-shot multi-step**: fitted on the training set, then produces a **full 72-hour forecast** starting from the first validation timestamp.
- No rolling refits or recursive loops — the forecast is entirely out-of-sample for the validation period.

10.6 Evaluation (validation 80/20)

Train (in-sample) ===

- **MAE: 0.171**
- **RMSE: 1.930**
- **R^2: 0.979**

Validation (multi-step, one-shot over full test) ===

- **MAE: 0.130**
- **RMSE: 0.570**
- **R^2: 0.994**

Interpretation: This is by far the **most accurate** among all tested models, with negligible error growth over 72 hours and extremely high R^2, making it well-suited for **stable production forecasts** and front-end use.

10.7 Artifacts saved

- **Model object:** models/current/sarimax_72h.pkl
- **Forecast CSV:** predictions/sarimax_predicted_aqi_72hrs.csv (timestamps in d/m/yy HH:MM format covering the next 72 hours).
- **Metadata:** includes model order parameters, seasonal order, train/val split details, preprocessing steps.

10.8 Automation

After validating SARIMAX in the notebook, I developed a **standalone Python script (daily_aqi_train.py)** to make the workflow fully automatable. This script replicates the key steps of the notebook in a production-friendly way:

- It begins by **reading and preprocessing the AQI dataset**, including rebuilding lag and rolling features, ensuring all transformations are applied in a leak-safe manner.
- The script then performs a **chronological train/validation split**, fits the SARIMAX model with the best-identified hyperparameters, and logs **train and validation metrics** for monitoring.
- Once validated, the model is **refitted on the complete dataset** to produce the final version used for deployment.
- A **72-hour forecast** is then generated, saved locally as a timestamped CSV, and also uploaded to **Hopsworks**. Both the **model artifacts** (SARIMAX weights, exogenous scaler, and metadata) and the **forecast outputs** are pushed to the **Hopsworks Model Registry and Feature Store**, enabling reproducibility, version control, and integration with downstream applications such as dashboards or APIs.

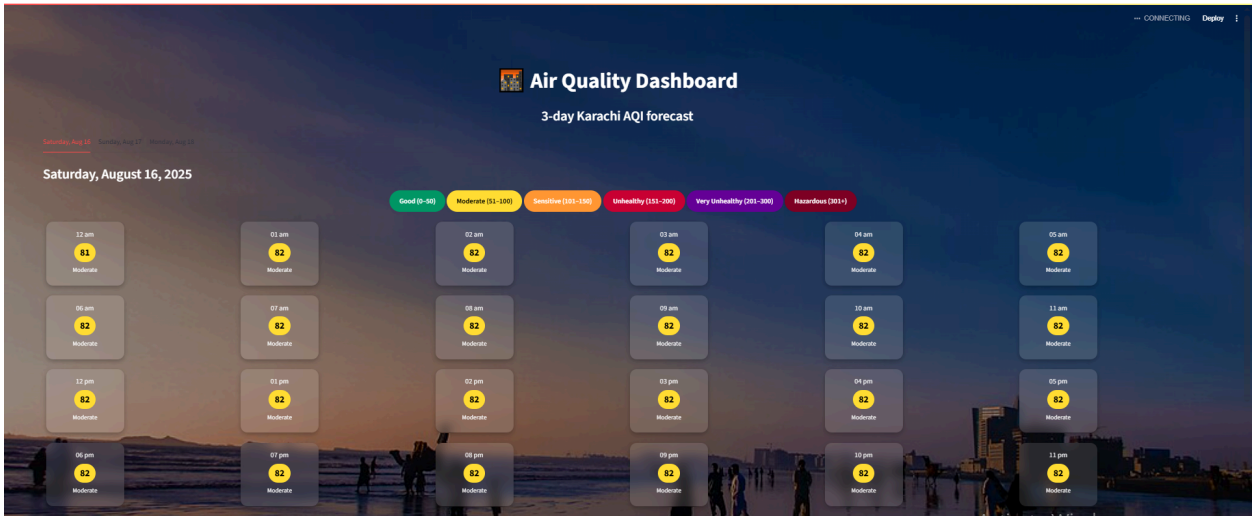
To remove the need for manual execution, I further integrated this script into a **GitHub Actions workflow**. The workflow is configured to **trigger automatically at midnight UTC every day**. When triggered, it:

1. **Checks out the repository** and sets up the Python environment with all required dependencies.
2. **Runs the training script** with the appropriate configuration, using secure environment variables for Hopsworks credentials.

3. **Uploads new forecasts and model versions** to Hopsworks and commits any updated prediction files back into the repository.

This automation script and a scheduled GitHub Actions workflow ensures that the **SARIMAX model is retrained daily** with the latest data and that **72-hour forecasts are always available and up to date**. It transforms the notebook experiment into a **fully automated, production-ready pipeline** with versioning, monitoring, and deployment capabilities

11. Frontend Dashboard



Purpose & data flow

The frontend is a **Streamlit** app that renders a **rolling 72-hour (3-day)** AQI forecast for Karachi generated by the SARIMAX pipeline. At startup it:

1. **Downloads the latest forecast CSV** directly from the repo's artifacts/predictions/ path.
2. If today's file isn't present yet (e.g., the GitHub Action hasn't run), it **falls back to yesterday's file**.
3. Parses timestamps to hourly granularity and **standardizes the schema** so the rest of the UI can assume a single aqi column.
   This simple "today-or-yesterday" strategy makes the UI robust to timing gaps between data generation and publishing.

Information architecture

- **Three day tabs:** The app groups the 72 hourly rows by date and creates **one tab per day** (e.g., *Saturday, Aug 16* to *Monday, Aug 18*). The user can jump to any day without

scrolling through all 72 cards.

- **Two complementary views per day:**
    - **Hourly card grid** for quick at-a-glance status.
    - **24-hour trend chart** for pattern recognition across the day.

Visual design & UX choices

- **Hero header:** A centered title ("Air Quality Dashboard") with a brief subtitle ("3-day Karachi AQI forecast"). The background is a **blurred photographic banner** of Clifton Beach, overlaid with a dark gradient to keep foreground text readable.
- **Typography & theme:** A clean sans-serif (Poppins) is applied and the color palette follows **EPA AQI categories**:
    - Good (#009966), Moderate (#FFDE33), Unhealthy for Sensitive Groups (#FF9933), Unhealthy (#CC0033), Very Unhealthy (#660099), Hazardous (#7E0023).
- **Legend as pills:** A compact **pill legend** appears at the top of each day so users can instantly map colors to categories. This legend is responsive and wraps neatly on smaller screens.

Hourly forecast cards (flip interaction)

- Each hour is shown as a **glassmorphic "pill card"** (rounded, translucent, subtle shadow) that displays:
    - **Hour label** ("01 am", "03 pm"),
    - **AQI value** in a prominent badge,
    - **AQI category** (e.g., Moderate).
- Cards include a **flip-on-hover interaction**: the back shows **contextual health advice**.
    - Example: for *Good*, "Great day to be outside!"; for *Moderate* or worse, guidance to **limit outdoor exertion**, especially for sensitive groups.
- The grid renders **6 cards per row** to keep scanning effortless; it adapts to the remaining items to avoid broken rows.

24-hour trend visualization

- A **Plotly bar chart** summarizes the same day's 24 hours. Bars are:
    - **Colored by category** using the EPA palette (exact hex mapping),
    - **Annotated with the numeric AQI** (labels above each bar),

    - Framed on a **white canvas** with light grid lines for readability.
- The y-axis is auto-bounded to show headroom above the daily max (max(200, maxAQI + 50)), keeping bars visually comparable across days.
- This combination (cards + chart) supports **both quick status checks and trend interpretation**.

Data handling & resilience

- **Schema normalization:** The loader renames either predicted_aqi or predicted_aqi_us → aqi, ensuring downstream components are decoupled from upstream naming changes.
- **Time handling:** Timestamps are floored to the hour and split into day, hour, and hour_of_day for grouping and plotting.
- **Graceful failure paths:** If the data file is missing or malformed, the app shows a clear **user-facing error** instead of crashing, and logs the exact URL(s) attempted.

Category logic & styling system

- **Categorization:** Values are mapped to EPA bands:
    - 0–50 Good, 51–100 Moderate, 101–150 USG, 151–200 Unhealthy, 201–300 Very Unhealthy, 301+ Hazardous.

- **Reusable helpers:** Two utilities centralize logic:
    - categorize_aqi_value(v) → category string,
    - css_class_for_aqi(category) → CSS class for consistent styling across cards, pills, and legends.
- This separation allows easy **visual refresh** (e.g., brand colors) without touching business logic.

Performance & responsiveness

- **Incremental rendering:** The grid streams rows in **chunks of 6** to keep the UI snappy and readable.
- **Client footprint:** Heavy computation is avoided on the client; the page mostly renders already-prepared values. Plotly is used for a single daily chart to balance clarity and load time.
- **Mobile friendliness:** Pills, tabs, and the legend are designed to **wrap** gracefully; font sizes and spacings are tuned for compact screens.

Accessibility & clarity

- **Contrast-aware** text on colored backgrounds (e.g., black text on the yellow Moderate pill).
- **Redundant encoding:** Every colored element is paired with **text labels** ("Moderate", "Unhealthy") so users who can't rely on color alone still understand the status.
- **Descriptive headings** ("24-Hour AQI Trend") and centered titles improve scanability.

Operational integration

- The app assumes a **fresh CSV is published daily** by the GitHub Action that runs the SARIMAX script.
- Because it pulls from **GitHub raw content**, no secret tokens are needed on the frontend; the data is public, cache-friendly, and versioned.
- If the run hasn't completed yet at page load time, the **yesterday fallback** guarantees continuity.

12. Containerization with Docker

To make the Air Quality Prediction system portable and deployment-ready, I **containerized the entire Streamlit application using Docker**. The idea was to eliminate dependency conflicts and ensure that the dashboard would run reliably on any environment — whether local, cloud-based, or within a CI/CD pipeline.

The Docker setup followed a clean workflow: starting with a lightweight Python base image, I copied over the dependency list (requirements.txt) to install all necessary libraries in a reproducible way. The application code was then added to the container, and the Streamlit service was configured to run on the default port (8501) and listen on all network interfaces. This guarantees that once the container is built and launched, the dashboard is immediately accessible in a browser without additional setup.

By containerizing the application, I achieved:

- **Reproducibility** – the same environment is guaranteed across development, staging, and production.

- **Portability** – the container can be deployed on any server or cloud provider without worrying about local configurations.

- **Scalability** – multiple instances of the dashboard can be spun up and managed easily with orchestration tools like Docker Compose or Kubernetes if needed in the future.

This step essentially transformed the AQI forecasting project from an experimental prototype into a **production-grade deployable service**.

13. Key Learnings from the Project

Working on this AQI forecasting system allowed me to gain insights not only into **time-series modeling** but also into the broader pipeline of building, automating, and deploying a data-driven application. Some of the major learnings include:

1. **Modeling & Experimentation**
    - Explored and compared different approaches (Random Forest, LightGBM, LSTM, TCN, and SARIMAX) and saw firsthand how error behavior differs between traditional ML, deep learning, and statistical models.
    - Learned that **deep learning architectures** (LSTM, TCN) can capture patterns but often drift at longer horizons, while **SARIMAX** remained remarkably stable and interpretable, making it the best fit for production.
    - Gained experience in handling **sliding windows, lagged features, and exogenous variables**, and saw how data leakage can be prevented with proper preprocessing.
2. **Automation & Reproducibility**
    - Designed a **Python training script** that refits the model daily, proving the importance of modular, reproducible code outside notebooks.
    - Integrated the pipeline with **GitHub Actions** to automate retraining and publishing of forecasts, reinforcing how CI/CD practices apply directly to ML workflows.
    - Learned how to manage **model versioning and artifacts** with Hopsworks, making the system traceable and production-ready.
3. **Deployment Engineering**
    - Understood the importance of **containerization (Docker)** for consistent environments and easy deployment.
    - Learned how even a lightweight framework like **Streamlit** can be made portable and scalable once containerized.

- Gained exposure to structuring an end-to-end pipeline that moves smoothly from raw CSV to trained model to predictions to feature store to frontend.

4. **Frontend Development & Communication**

    - Learned how to transform raw model outputs into a **clear, user-facing dashboard** that communicates forecasts effectively.

    - Practiced **data visualization principles**: color coding with AQI standards, using redundant encoding (text + color), and providing contextual health advice.

    - Experienced how thoughtful UX (tabs, legends, flip-cards, tooltips) can make technical data accessible to non-technical audiences.

5. **Holistic Project Perspective**

    - Understood that **a strong model alone is not enough**—what matters is the entire ecosystem: data engineering, automation, monitoring, deployment, and usability.

    - Learned the value of **closing the loop**: from experimentation in notebooks to automation with scripts to deployment pipelines to real-time visualization.

    - Gained confidence in combining **data science, MLOps, and frontend engineering** into one coherent solution.