Air Quality Index (AQI) Forecasting System

End-to-End Machine Learning Project

Submitted by

Muhamamd Talha

Organization

Pearls Organization

Date

17 February 2026

Contents

1 Introduction

This report explains a small but complete Air Quality Index (AQI) forecasting system for Islamabad, Pakistan. The system collects air and weather data, trains machine learning models, makes 3-day AQI forecasts, and shows results in a web dashboard.

The main ideas of the project are:

- use real hourly data from a public API,
- build a clear pipeline from data to prediction,
- keep everything simple so another student can understand and run it.

2 Project Summary

2.1 High-level description

The project predicts AQI for the next 72 hours (3 days). It uses:

- OpenMeteo for air quality and weather data,
- MongoDB Atlas as a cloud database,
- Random Forest, XGBoost and LightGBM as models,
- Streamlit for the user interface,
- GitHub Actions to keep the system updated automatically.

2.2 System overview

Layer	Main tools / files	Purpose
Data collection	`data_collector.py`, `collect_and_store_features.py`	Get raw and live data from Open-Meteo
Feature engineering	`calculate_aqi.py`, `feature_engineering.py`	Turn raw data into AQI and features
Modeling	`train_and_register_model.py`	Train three models and pick the best
Prediction	`predict_next_3_days.py`	Make 72-hour AQI forecast
Storage	`mongo_db.py`, MongoDB Atlas	Save features, models and predictions
Dashboard	`app.py` (Streamlit)	Show current AQI, forecast and metrics

Table 1: System overview

3 Data Collection

3.1 Data source

The project uses the OpenMeteo APIs:

- Air Quality API: PM2.5, PM10, CO, NO_2, SO_2, O_3,
- Weather API: temperature, humidity, wind speed, wind direction, precipitation, cloud cover.

The city is Islamabad. The latitude and longitude are stored in environment variables but default to the known coordinates.

3.2 Historical data (one-time)

The script `src/data/data_collector.py`:

- reads the city name and coordinates,
- downloads around one year of hourly data from both APIs,
- merges air and weather data on the timestamp,
- drops rows with missing key values,
- saves the result as a CSV file under `data/raw/`.

This CSV is later converted into AQI and features.

3.3 Live hourly data

The pipeline `src/pipeline/collect_and_store_features.py` is used after the historical load. It:

- requests the latest hour of data from OpenMeteo,
- calculates AQI for that hour,
- adds time and lag features,
- writes one document into the `aqi_features` collection in MongoDB.

The script can be called locally or by GitHub Actions every hour.

4 Feature Engineering

4.1 AQI calculation

The script `src/features/calculate_aqi.py`:

- uses official EPA breakpoints for each pollutant,
- applies the AQI formula for PM2.5, PM10, O_3, NO_2, SO_2 and CO,
- takes the maximum AQI across all pollutants as the final AQI,
- adds a simple category such as Good, Moderate or Unhealthy.

The output is saved as `data/processed/aqi_data.csv`.

4.2 Extra features for modeling

The script `src/features/feature_engineering.py` builds:

- **Time features**: hour of day, day of week, month, sine and cosine of hour,
- **Lag features**: AQI one hour ago (`aqi_lag_1`) and 24 hours ago (`aqi_lag_24`),
- **Weather interactions**: temperature $\times$ humidity, wind x and y components.

The final cleaned table is stored as `data/processed/processed_aqi.csv` and then uploaded to MongoDB.

5 Model Training

5.1 Training data

The script `src/pipeline/train_and_register_model.py` uses the `aqi_features` collection from MongoDB. It:

- reads all rows and drops the MongoDB `_id` column,
- splits data into 80% train and 20% test (no shuffle to keep time order).

5.2 Models and metrics

Three regression models are trained:

- Random Forest Regressor,
- XGBoost Regressor,

- LightGBM Regressor.

Each model is evaluated on the test set using:

- coefficient of determination R^2,
- root mean squared error (RMSE),
- mean absolute error (MAE).

Example scores from one experiment are shown in Table 2.

Model	R^2	**RMSE**	**MAE**
Random Forest	0.70	45.10	15.24
XGBoost	0.70	45.26	15.69
LightGBM	0.82	34.86	14.68

Table 2: Example test scores for the three models

LightGBM has the highest R^2 and is selected as the best model.

5.3 Model registry

After training, the best model is saved as `models/best_model_lightgbm.pkl`. A document is also written into the `model_registry` collection in MongoDB. This registry stores:

- metrics for all three models,
- the name of the best model,
- the path to the best model file,
- the training time and flags such as "latest" and "baseline".

This makes it easy for the prediction step and the dashboard to know which model to use.

6 Prediction Pipeline

The script `src/pipeline/predict_next_3_days.py` performs the 72-hour forecast. It:

1. Loads the best model from the model registry.
2. Reads the most recent 24 hours of data from `aqi_features`.
3. Builds features for the next 72 hours by:
    - moving the timestamp forward one hour at a time,
    - using average values from the recent data for pollutants and weather,
    - recomputing time and lag features.
4. Runs the model to get predicted AQI for each future hour.
5. Saves the 72 predictions in the `predictions` collection with:
    - forecast timestamp,
    - predicted AQI,
    - model name,
    - prediction date.

These predictions are used by the Streamlit dashboard.

7 Dashboard (Streamlit)

The Streamlit app is defined in `app.py`. It acts as a simple front-end on top of MongoDB.

The dashboard shows:

- **Current AQI:** estimated from the first prediction point. If predictions are missing, it falls back to the latest measured AQI from `aqi_features`.
- **3-day forecast chart:** line chart of the 72 predicted AQI values, with colored bands representing AQI zones and a small daily summary table.
- **Model snapshot:** name of the best model (usually LightGBM) and its latest R^2 score from the registry.

The layout is kept clean and simple. Text like "powered by Muhammad Talha" is included to clearly show who customized and deployed the project.

When the app is run locally, extra buttons in the sidebar allow the user to collect a fresh hour of data and regenerate the 3-day forecast manually. On Streamlit Cloud these buttons are hidden, because GitHub Actions already refresh the data.

8 Automation with GitHub Actions

Three GitHub Actions workflows in the `.github/workflows` folder keep the system up to date.

8.1 Workflow summary

Each workflow:

- checks out the GitHub repository,
- sets up Python 3.12,
- installs dependencies from `requirements.txt`,
- reads MongoDB settings from GitHub secrets,
- runs the correct pipeline script.

This way, the system keeps running without manual work.

9 Deployment Details

9.1 MongoDB Atlas

The project uses a free MongoDB Atlas cluster. The database (for example named `aqi_predictor`) contains the following collections:

- `aqi_features` – hourly feature rows,
- `model_registry` – model metadata and best model path,
- `predictions` – 72-hour forecast rows.

Connection details are stored in:

- a local `.env` file for development,
- `st.secrets` on Streamlit Cloud,
- GitHub Actions secrets (`MONGODB_URI`, `MONGODB_DATABASE`).

9.2 Streamlit Cloud

The application is deployed on Streamlit Cloud using the GitHub repository.

- Main file: `app.py`,
- Secrets: `MONGODB_URI` and `MONGODB_DATABASE`,
- Live URL at the time of writing:
  https://pearlairpredictor-iv38jgdt9fsg2lctbjm7q8.streamlit.app/.

10 Conclusion

This project demonstrates a complete but understandable AQI forecasting pipeline. It connects data collection, feature engineering, model training, prediction and a web dashboard into one flow.

The design uses simple and common tools in data science:

- Python and pandas for data processing,
- tree-based models (Random Forest, XGBoost, LightGBM) for prediction,
- MongoDB Atlas for storing features, models and predictions,
- Streamlit for the dashboard,
- GitHub Actions for automation.

The code is written in a straightforward style so that other students can read it, learn from it and extend it. Possible future work includes supporting more cities, adding alerts for very bad AQI days, explaining predictions with SHAP, and trying deeper models if needed.