AQI-Predictor Project Report

Zuhair Farhan - 10Pearls Shine Intern

November 9, 2025

Abstract

This report provides an overview of the AQI-Predictor project, made for the city of Karachi, completed during an internship in 10Pearls Shine Program. The project is designed to forecast the Air Quality Index (AQI) using Python, using automated flows via Github Actions. The project also makes use of **hopsworks**, specifically its feature store and model registry. Additionally, **Azure** Blob Storage and VMs were used for storing hourly ingested data and running the frontend dashboard. The project repo is hosted on github, and can be viewed here.

1 Introduction

The project uses hourly fetched data via the OpenWeatherAPI. For model training purposes, it uses backfill data from two sources:

- Open-Meteo API
- A Kaggle dataset.

For storing the data, a feature store from hopsworks was used. The hourly fetched data was stored as JSONs in an Azure blob storage, then used by a script to tabularize it, engineer new features, and then store it in the feature store. All of this is done automatically every hour via Github Actions.

EDA was performed on the data, including finding correlations with the label value (the AQI index), finding the feature importances via a random forest model, and using SHAP explanations. Some preliminary model training on about 20 models, using these results was performed. After the preliminary training, the top 5 models were chosen to be included in the actual training script, which is run daily. Whatever model records the lowest RMSE is uploaded to a model registry on hopsworks. The frontend dashboard goes through all the models uploaded to the model registry, and selects the overall best one to use for its predictions.

2 Overall Repository Structure

The repository is broken down into several directories, each serving a specific purpose. The explanation of the directories is as follows:

- `.github/workflows/`: This directory contains the configuration files for GitHub Actions, which are used to automate various tasks in the development lifecycle. There are three automated workflows within this directory, which are explained in the next section
- `app/`: The `app` directory holds the source code for the frontend of the application, which was built using the popular framework Streamlit.
- `eda/`: This folder is dedicated to Exploratory Data Analysis. The Jupyter notebook (`air_quality.ipynb`) is where the this analysis was performed, and the resulting figures and csvs were saved in the same directory.
- `features/`: This directory handles feature engineering. Scripts such as `derived_features.py` and `time_based_features.py` are where new features are created from the raw data to improve model performance. The script `hopsworks_fs.py` is where all the features are created, then uploaded to the hopsworks feature store.

- `fetch_data/`: This directory is responsible for data acquisition. It includes scripts to fetch raw data (`fetch_raw_data.py`) via the OpenWeather API and to backfill historical data via the two source files within the same directory.
- `models/`: The `models` folder is central to the machine learning pipeline. It contains scripts for training models (`preliminary_training_models.py`, `training_models.py`, and `automated_training.py`).

There are also a few additional files in the root of the project, used for CI and pre-commit hooks.

3 Automated Workflows

The repository utilizes GitHub Actions for automation, as indicated by the `.github/workflows` directory. There are three automated workflows within this directory, explained below:

- `hourly_ingest.yml`: This file defines an automated data ingestion pipeline.
  - **Trigger**: It is scheduled to run automatically every hour at minute 0 (using `cron: "0 * * * *"`) and can also be triggered manually (`workflow_dispatch`).
  - **Secrets**: It requires three secrets: `OPENWEATHER_API_KEY`, `HOPSWORKS_API_KEY`, and `AZURE_STORAGE_CONNECTION_STRING`.
  - **Jobs**: It runs a single job called `ingest` on an Ubuntu runner.
  - **Steps**: The job checks out the repository, sets up Python 3.10, installs dependencies from `requirements.txt`, and verifies that all necessary secrets are available. It then executes two Python scripts:
    1. `fetch_data/fetch_raw_data.py`: This script (as noted in the workflow) fetches raw data and uploads it to Azure Blob Storage.
    2. `features/hopsworks_fs.py`: This script processes the data and uploads it to a Hopsworks Feature Store.
- `daily_model_training.yml`: This file automates the machine learning model training process.
  - **Trigger**: It runs once every day at midnight UTC (using `cron: "0 0 * * *"`) and can also be run manually.
  - **Secrets**: It uses `HOPSWORKS_API_KEY` and `OPENWEATHER_API_KEY`.
  - **Jobs**: It runs a `train` job on an Ubuntu runner.
  - **Steps**: The job checks out the code, sets up Python 3.10, and installs all dependencies. It then runs the `models/automated_training.py` script to execute the entire training pipeline.
- `ci.yml`: This file defines a Continuous Integration (CI) pipeline to ensure code quality.
  - **Trigger**: It runs automatically on every `push` and `pull_request` that targets the `main` branch.
  - **Jobs**: It runs a single `build` job on an Ubuntu runner.
  - **Steps**: After checking out the code and setting up Python, it installs the main dependencies (`requirements.txt`) as well as specific CI tools: `black`, `ruff`, `pytest`, and `pre-commit`.
  - **Checks**: It performs two quality checks:
    1. `black --check .`: Verifies that all code is formatted according to the Black code style.
    2. `ruff check .`: Lints the code using Ruff to find errors, bugs, and style issues.

4 Exploratory Data Analysis (EDA)

4.1 Python Notebook Overview (`air_quality.ipynb`)

The notebook executes the following key steps:

1. **Correlation Graphs:** Several graphs were generated comparing the features correlation with the target variable, `ow_aqi_value`, including a heatmap graph and csv listing all the correlation values.

2. **Feature Importance:** The notebook employs two distinct methods to understand the important features:
    - **Random Forest Feature Importance:** It calculates and plots the Gini-based feature importance directly from the trained `RandomForestRegressor`. This gives a high-level overview of which features the model "looked at" most often.
    - **SHAP (SHapley Additive exPlanations):** It runs a comprehensive SHAP analysis using the `shap_analysis` module. This is a more advanced technique that provides a detailed, instance-level breakdown of each feature's contribution to each individual prediction.

4.2 Exploratory Data Analysis (EDA) Results

4.2.1 Feature Correlations (`feature_correlations.csv`)

This file shows the Pearson correlation coefficient between each feature and the target variable, `ow_aqi_index`.

- **Strong Positive Correlation:** The features most strongly and positively correlated with AQI are all related to particulate matter and their recent averages. This is expected, as PM is a primary component of AQI.
    - `pm10_rolling_avg_24h` (0.61)
    - `pm10_rolling_avg_7d` (0.59)
    - `pm10` (0.56)
    - `pm2_5_rolling_avg_24h` (0.53)
- **Weak/Negative Correlation:** Meteorological features like temperature (`temp`, -0.03) and weather based features show almost no linear relationship with the AQI.

4.2.2 Model-Based Feature Importance (`feature_importance.png`)

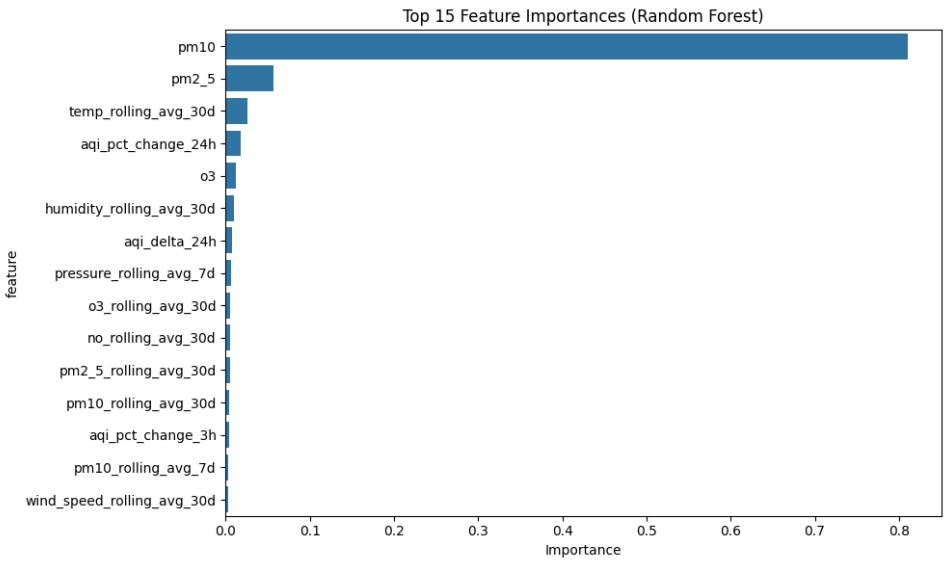

Figure 1: Feature importance from the Random Forest model.

This plot confirms the EDA findings above. The model learned that **`pm10`, and `pm2_5` are by far the most important features for making an accurate prediction.**

4.2.3 SHAP Analysis (`shap_bar_plot.png` & `shap_summary_plot.jpg`)

SHAP provides a more nuanced view of the model's internal logic.

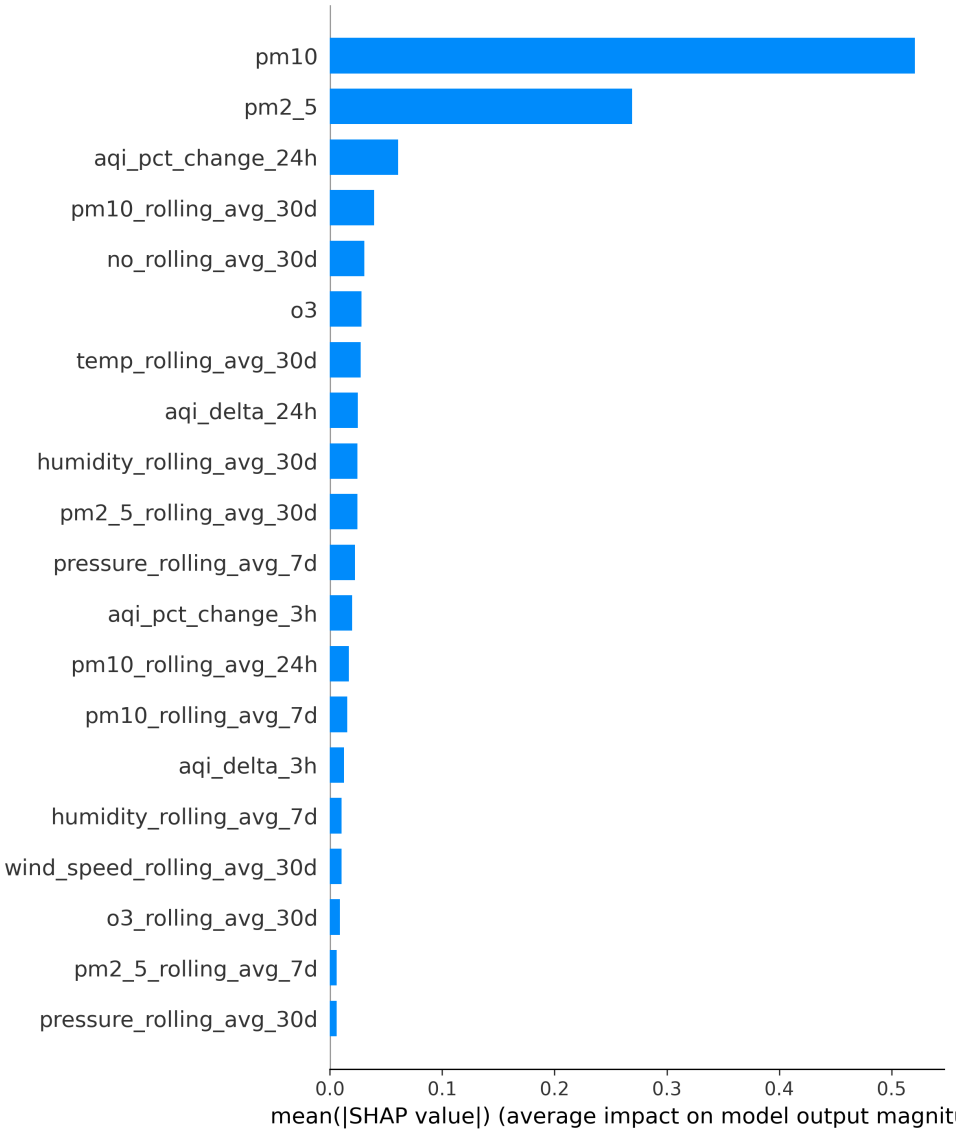

Figure 2: Mean absolute SHAP value (feature impact).

The SHAP bar plot ranks features by their average impact on the model's output. It aligns with the feature importances from the random forest model, where the top drivers are **`pm10`, and `pm2_5`**.

The SHAP summary plot can also be seen below:

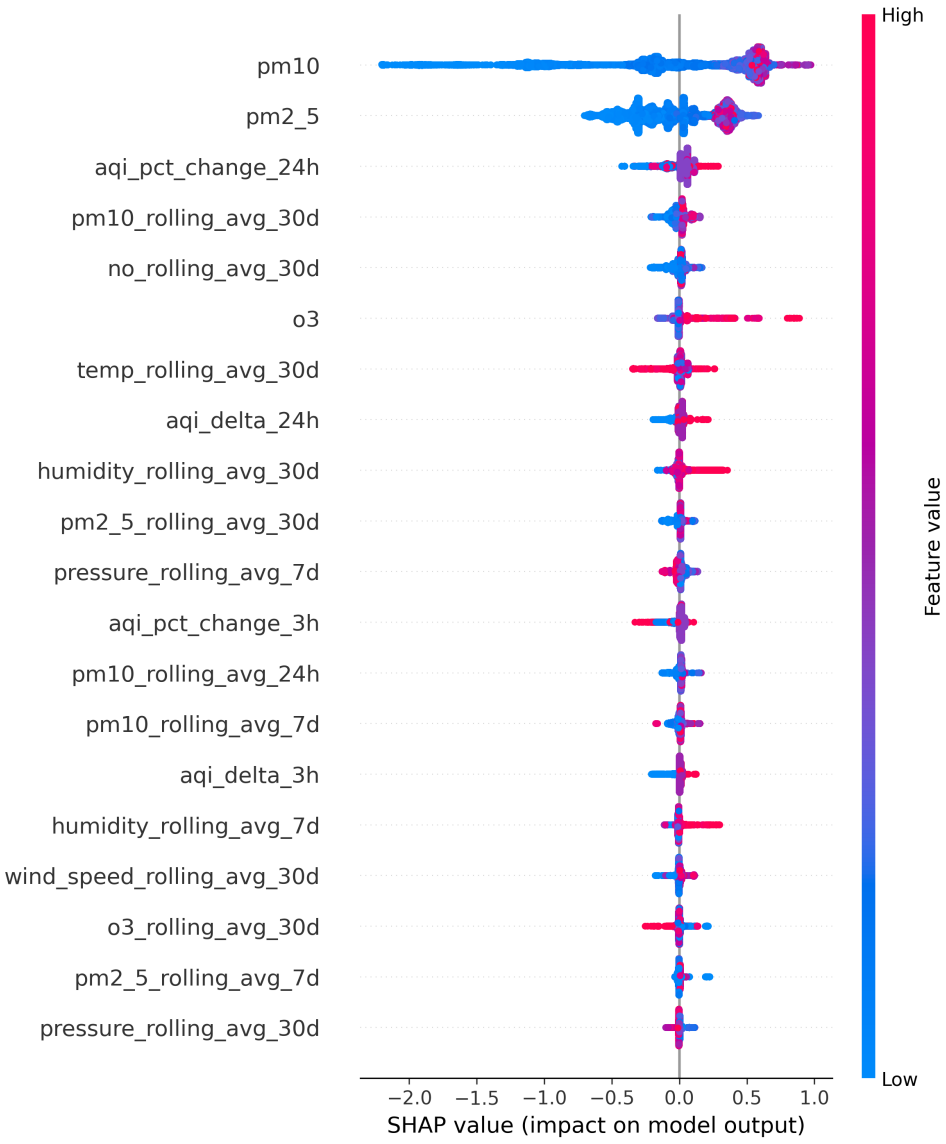

Figure 3: SHAP summary plot (beeswarm).

4.3 Summary of the EDA

The entire analysis, from feature correlations to model explainability, is highly consistent. Both the initial data correlation and the final model's internal logic (via SHAP) confirm that **particulate matter (especially `pm10` and `pm2_5`) are the dominant and logical drivers of AQI predictions.**

5 Training the Models

5.1 `preliminary_training_models.py`: Initial Model Experimentation

This script serves as a comprehensive "sandbox" for identifying the best model and feature set. Its primary goal is to "battle-test" a wide array of regression models and explore different feature selection strategies.

The scripts tests a large range of models, including:

- **Linear Models:** `LinearRegression`, `Ridge`, and `Lasso`.
- **Distance-Based:** `KNeighborsRegressor`.
- **Tree-Based:** `DecisionTreeRegressor`.

- **Ensemble Models:** `RandomForestRegressor`, `ExtraTreesRegressor`, `XGBRegressor`, and `LGBMRegressor`.

The script was used to test multiple feature engineering hypotheses based on the EDA performed. This included:

1. **Full Feature Set:** An initial run which used all of the features.
2. **Low-Correlation Pruning:** An iteration where features with a correlation score close to 0 with the target label (e.g., `temp_feels_like`, `co_no2_ratio`, etc.) were dropped.
3. **Top-5 Feature Selection:** The top 5 features identified from a Random Forest feature importance analysis in the EDA, which are (`pm10`, `pm2_5`, `temp_rolling_avg_30d`, `aqi_pct_change_24h`, and `o3`).

The results are saved in subdirectories:

- `all_features_kept___no_standardization/`
- `all_features_kept___standardization/`
- `removing_low_correlation_features___no_standardization/`
- `removing_low_correlation_features___standardization/`
- `top5_rf_feature_imp___no_standardization/`
- `top5_rf_feature_imp___standardization/`

The results of this preliminary training found that:

1. Standardization and no standardization did not seem to make any major impact. In fact, no standardization proved to be slightly better in some of the runs.
2. Although only containing the top 5 features from the random forest feature importances explanation did provide better results for a couple of models, in general, just removing the low correlation features provided better overall results.
3. From all the models, the models which provided the best and consistent results were the Random Forest, XGBoost, and LGBoost models - hence, there are retained in the daily automated model training, explainen below.

5.2 `training_models.py`: Production Training Pipeline

This script is the refined, production-ready version that emerged from the experiments in the preliminary file. It is optimized for performance and integration with the MLOps platform (Hopsworks). The script focuses only on the consistently best-performing ensemble models: `RandomForestRegressor`, `XGBRegressor`, and `LGBMRegressor`.

Instead of saving to a local folder, this script:

- Uses a `tempfile.mkdtemp()` to store artifacts temporarily.
- Connects to the Hopsworks Model Registry (`mr = project.get_model_registry()`).
- Creates a new model version in the registry, uploading the model artifact, metrics (RMSE, MAE, R2), and metadata.
- The temporary directory is automatically deleted, leaving no local files.

5.3 `automated_training.py`: Daily Execution Script

This script is the high-level entry point that automates the entire production pipeline. It is executed daily via a GitHub Actions workflow.

Its logic is very straightforward:

1. **Import:** It imports the `train_and_evaluate_models` function directly from `training_models.py`.
2. **Fetch Data:** It connects to the Hopsworks project and reads the latest data from the `air_quality_data` feature group.
3. **Execute:** It passes the new DataFrame to the imported training function, which runs the entire process of training, evaluation, and uploading the new best model to the Hopsworks Model Registry.

This ensures that a new model, trained on the most recent data, is registered and available for use every day.

6 The Frontend

The frontend is built using Streamlit. A custom CSS block was used to override some of the default styles of the framework, applying a custom dark-mode theme with a gradient background and specially styled "metric cards" for displaying data.

The core predictions are organized into a four-column layout using, which displays the "Current AQI" and forecasts for +24, +48, and +72 hours. These prediction cards are color-coded using a helper function, `get_aqi_category()`, which maps the model's numerical output (1-5) to human-readable categories and colors (e.g., "Moderate", "Unhealthy").

For visualization, the dashboard uses `st.plotly_chart()` to render two interactive Plotly graphs, both styled to match the dark theme:

- **7-Day AQI Trend:** Shows the historical AQI for the past week.
- **Rolling AQI Averages:** Plots the 24-hour and 72-hour rolling averages against the actual AQI.

Finally, the UI includes a static, two-column informational section (`st.columns(2)`) that explains different international AQI scales. The `st.sidebar` is used to display a simple AQI legend and the backend connection status.

The dashboard can be seen in the following figures:

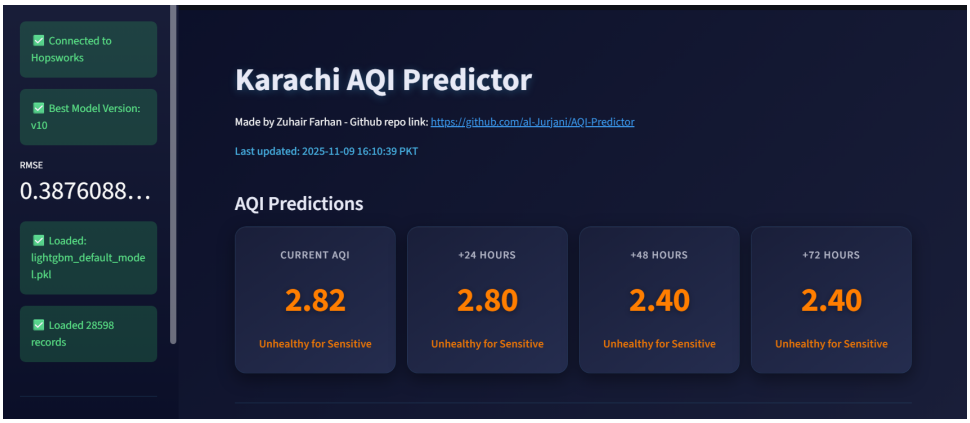

Figure 4: The Frontend - Predictions

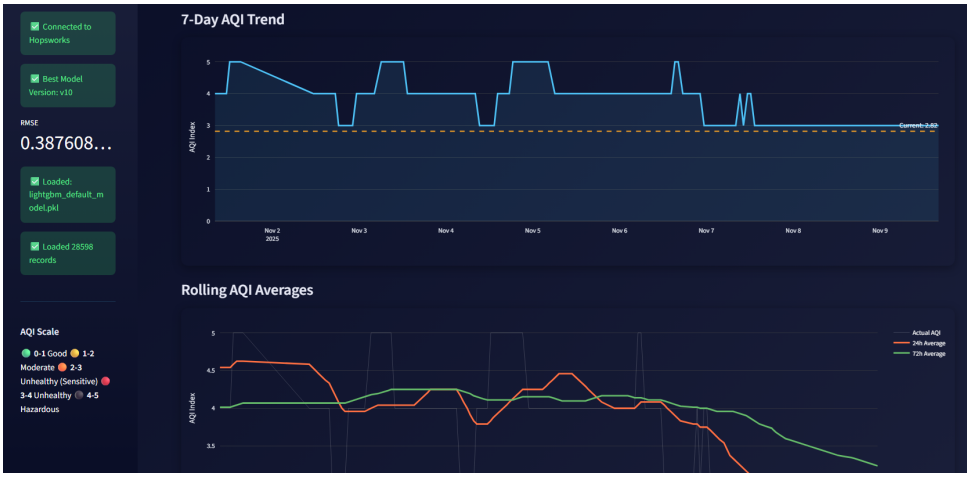

Figure 5: The Frontend - Statistical Plots

7 Additional Things Done

The project also incorporates several modern software and data engineering best practices:

- **Azure Blob Storage**: The hourly generated weather and air quality data, fetched from the OpenWeather API, is stored as JSON files in an online storage location, using Microsoft Azure's Blob Storage service. This provides a scalable and robust solution for data storage.
- **Pre-commit Hooks and CI**: The `.pre-commit-config.yaml` file and dependencies like `black`, `ruff`, and `flake8` are used to implement pre-commit hooks, which enforce code quality and consistency. This Continuous Integration (CI) practice helps to maintain a clean and readable codebase by automatically formatting and linting the code before it is committed.
- **Azure VM for Hosting**: The Streamlit frontend is hosted on an Azure Virtual Machine. So it can be accessed publicly as long as the VM itself is active.

8 Learning Experience

This internship has been a great opportunity for me to build a great project that I can add to my resume. I got to apply core data science skills such as data preprocessing, model hyperparameter tuning, and dashboarding. In addition, also got some MLOps experience by implementing CI/CD pipelines via github actions and using cloud storage and compute resources via Azure. I'm really grateful for this experience, and genuinely hope for more opportunities like this from 10Pearls!