



Pearls AQI Predictor

Submitted By:

Maya Khurshid Anwar

Software Engineering Student from Jinnah University for Women

1. Data Acquisition, EDA, and Feature Engineering:

1.1 Introduction

The primary objective of this project was to develop an **end-to-end, serverless Air Quality Index (AQI) prediction system for Karachi**.

The foundation of this system is a robust **data pipeline** designed to acquire, process, and store high-quality data using the **Hopsworks API** for feature storage and management.

This report covers the **initial phase** of the system, including:

- Project structure and workflow
- Data acquisition via external API
- Exploratory Data Analysis (EDA)
- Feature Engineering and Hopsworks integration

1.2 Data Acquisition

The raw data was sourced from the **OpenWeatherMap Air Pollution API**, which provides real-time and historical air quality metrics.

- The script `data_fetcher.py` automates this process.
- It retrieves Karachi's coordinates and fetches air quality data for a defined date range.
- The data includes attributes like **datetime, aqi, co, no, no₂, o₃, so₂, pm_{2.5}, pm₁₀, and nh₃**.

The fetched data is stored as **CSV files** in the `data/raw/` directory for persistence and traceability.

1.3 Exploratory Data Analysis (EDA)

EDA was performed in `notebooks/eda.ipynb` to uncover meaningful patterns:

- **Correlation Analysis:**
A heatmap revealed a strong positive correlation between **PM_{2.5}**, **PM₁₀**, and AQI. These pollutants were identified as key contributors to air degradation.
- **Time-Series Analysis:**
AQI variation was plotted over 24 hours, showing **spikes during traffic hours**. This guided the creation of **time-based features** for prediction.

1.4 Feature Engineering

Raw environmental data is rarely suitable for direct model consumption.

The `feature_engineer.py` script enriches the dataset with temporal and derived features.

Time-Based Features

Extracted from the `datetime` column:

- **hour** → Hour of the day (0–23)
- **dayofweek** → Day of the week (0=Monday, 6=Sunday)

Derived Features

- **aqi_change_rate** → Represents the change in AQI from the previous hour to capture recent trends.

The processed dataset is saved in `data/processed/`, ready for model training.

1.5 Feature Store Implementation (with Hopsworks API)

To enable **centralized, versioned, and reusable feature management**, the **Hopsworks Feature Store API** was integrated.

The `feature_store.py` script performs the following:

1. Connects securely to the **Hopsworks platform** using the **Hopsworks Python SDK** and **API key** stored in `.env`.
2. Creates or updates a **feature group** (`aqi_features_karachi`).

3. Inserts processed feature data (`pandas DataFrame`) into the Hopsworks Feature Store.

This allows other components (like model training and prediction services) to **access the same consistent feature data** directly from Hopsworks — ensuring data governance and MLOps scalability.

2. Model Training and Evaluation

2.1 Introduction

With high-quality features available in the Hopsworks Feature Store, the next step involved training and evaluating multiple machine learning models to predict **AQI for the next hour**.

The objective was to identify a model that balances **accuracy, interpretability, and scalability** for real-time use.

2.2 Model Selection

Three regression models were tested using `model_trainer.py`:

1. **Linear Regression:**

Baseline model, useful for understanding linear feature relationships.

2. **Random Forest Regressor:**

Ensemble-based model, reduces overfitting, handles nonlinearities well.

3. **XGBoost Regressor:**

A gradient boosting algorithm offering state-of-the-art performance and computational efficiency.

2.3 Training and Evaluation Process

- The dataset was retrieved directly from **Hopsworks Feature Store** using the **Hopsworks API**, ensuring consistent and up-to-date data.
- Data was split into **80% training** and **20% testing** sets using `train_test_split()`.
- Models were evaluated using key regression metrics:

Metric	Description
RMSE	Measures average magnitude of prediction errors.
MAE	Reflects the average absolute difference between predicted and true values.
R²	Indicates how well the model explains AQI variance.

2.4 Model Performance Comparison

Model	RMSE	MAE	R ²
Linear Regression	0.45	0.32	0.75
Random Forest	0.22	0.08	0.91
XGBoost Regressor	0.19	0.05	0.94

2.5 Final Model Selection

The **XGBoost Regressor** achieved the best overall results with:

- **Lowest RMSE and MAE**
- **Highest R² (0.94)** — explaining 94% of AQI variance

The trained model was exported as `best_aqi_predictor.pkl` for deployment in the Streamlit web application.

3. Automation and Deployment with CI/CD

3.1 Introduction

To maintain up-to-date and reliable predictions, the system incorporates a **CI/CD pipeline** powered by **GitHub Actions** and **Hopsworks API integration**.

This ensures fully automated retraining and deployment without manual effort.

3.2 CI/CD with GitHub Actions

The pipeline, defined in `workflows/github_action.yml`, automates every stage of the ML lifecycle:

- Scheduled hourly runs
- Secure integration with **Hopsworks API** using stored secrets
- Automatic feature updates and model retraining

3.3 Automated Workflow Steps

1. **Checkout Repository:**
Fetches the latest project code.
2. **Set Up Python Environment:**
Initializes Python 3.11 for reproducibility.

3. Install Dependencies:

Installs all libraries from `requirements.txt`.

4. Run Feature Pipeline:

Executes:

- o `data_fetcher.py` → Fetch new API data
- o `feature_engineer.py` → Generate time-based features
- o `feature_store.py` → Upload features via **Hopsworks API**

5. Run Training Pipeline:

`model_trainer.py` retrieves the latest features from Hopsworks and retrains the **XGBoost** model.

6. Model Storage:

The updated model is saved as `best_aqi_predictor.pkl`, ready for use in the prediction service.

