Project Report - AQI Forecasting for NED University Region (Karachi)

Internship Domain: Data Science
Author: Kashif Mahmood

Abstract

This project implements an end-to-end system to forecast the Air Quality Index (AQI) for the NED University region of Karachi for the next 72 hours. Hourly AQI observations are obtained from the World Air Quality Index (WAQI) API and appended to a CSV file via an hourly GitHub Actions workflow. The pipeline performs preprocessing, exploratory data analysis (EDA), outlier detection and removal, and feature engineering (lags, cyclical encodings, rolling statistics). Engineered features are stored in the Hopsworks Feature Store. Multiple regression models included XGBoost, SVM, Random Forest and Gradient Boosting are trained and fine-tuned with GridSearchCV. The best base model is saved and wrapped with MultiOutputRegressor for multi-step forecasting (72-hour horizon). Both model binaries are saved locally and registered in Hopsworks Model Registry. All stages from feature engineering through model registration are automated using GitHub Actions, and a Streamlit application provides a user interface for visualization and prediction.

Data collection:

Hourly AQI data for the NED University region is collected from the World Air Quality Index API (https://waqi.info/). A dedicated script, fetch_aqi_data.py, calls the WAQI API, extracts relevant fields, and appends each new hourly observation to a csv file named hourly_aqi_data.csv. This ingestion step is fully automated through GitHub Actions workflow that runs every hour and append the data to the csv file. The CSV contains historical observations that are subsequently read by preprocessing and feature-engineering routines.

Preprocessing:

Preprocessing starts with basic dataset checks to understand the data that how it looks like, what is the size of data, what are the datatypes, are there any null values or duplicate values. These checks guide decisions on imputation, removal of redundant columns and handling of erroneous measurements. Data types are converted where necessary like converting timestamp to a datetime object is necessary. Correlation analysis helps identify redundant or strongly correlated columns, and any column found to be uninformative (e.g., a column with a single value across all rows) is flagged for removal. This staged preprocessing ensures the dataset is consistent, clean, and ready for exploratory analysis and feature construction.

Exploratory Data Analysis (EDA):

EDA is split into two parts:

Summary statistics:

- Max, min, mean values for numerical variables (temperature, humidity, pm1, pm10, aqi) are computed to understand ranges and central tendencies.

Distributional and relational analysis:

- Distributional and relational analysis
- Pie chart is utilized for categorical variables to inspect class balance.
- Histograms and kde plots are plotted for numerical features to inspect skewness of data. In this dataset the distributions were not heavily skewed, so no mathematical transformations were required.
- Boxplots revealed outliers in one or more columns, an IQR-based method was chosen for detection and removal of outliers
- Pairplot is used to inspect linear and non-linear relationships between all numerical columns.
- Heatmap visualizes correlations among numeric features and the target column.

Outliers detection and removal:

After ploting the boxplots, we have found the outliers which removed using the Interquartile Range (IQR) method on selected numeric columns (['aqi', 'temperature', 'humidity', 'pm1', 'pm10']). For each column, Q1 and Q3 are computed and values outside the range [Q1−1.5×IQR, Q3+1.5×IQR] are removed.

Feature engineering:

Timestamp decomposition: Convert timestamp to datetime, sort by time, and extract hour, day, month, day_of_week, and is_weekend. These fields capture calendar and context.

Lag features: Create lagged versions of AQI for 1, 3, 6, 12, 24, 48 and 72 hours (aqi_lag_1h, aqi_lag_3h, ...). which provide explicit previous hours values so the model can leverage recent and longer-term past values. Rows with insufficient history (first 72 rows when 72-hour lag exists) are removed.

Cyclical encoding: Apply sine and cosine transforms to hour and day_of_week to preserve circularity (e.g., make hour 23 and hour 0 adjacent in feature space): sin_hour, cos_hour, sin_dayofweek, cos_dayofweek.

Rolling statistics: Compute rolling means for AQI with windows 3h, 6h, 12h, 24h, 48h and 72h to compute an aggregate over a sliding window of past observations to avoid leakage and not to effect by the outliers and noisy data.

Feature pruning: Based on EDA, the uninformative or redundant columns such as city, dominentpol are dropped and the pm25 column is dropped as well due to same information with AQI.

Scaling & imputation pipeline: Build a ColumnTransformer that applies SimpleImputer(strategy='mean') followed by StandardScaler() to numeric features. Fit the pipeline on the cleaned dataset and convert the resulting scaled array back to a DataFrame with the same column labels.

Feature store upload: Insert the scaled feature DataFrame into a Hopsworks Feature Group named aqi_scaled_features_10pearls (version 1) with timestamp as the event time. This ensures reproducible, centralized feature management.

Feature Fetching: After inserting the features into Hopsworks, the scaled features are fetched again to train and fine tune the Machine Learning Models.

Model training & hyperparameter tuning:

Data split: The data was split into 20/80 train/test format to introduce the generalization and it's been a recommended practice.

Candidate models: Evaluate XGBRegressor (XGBoost), SVR (Support Vector Regressor), RandomForestRegressor, and GradientBoostingRegressor. These provide a mix of tree-based kernel-based and ensemble learning approches.

Grid search tuning: For each algorithm, run GridSearchCV with a carefully selected grid of hyperparameters and 5-fold cross-validation and use r2 as the primary scoring metric.

Evaluation metrics: Compute R^2, Mean Squared Error (MSE) and Mean Absolute Error (MAE) on the held-out test set for every tuned model. Compare results to select the best base model (highest R^2 and competitive MSE/MAE).

Model Storage: Save the best single-step model locally as models/best_model.pkl using pickle. This model will be the base estimator for multi-step forecasting.

Multi-step forecasting (72 hours):

To produce a 72-hour forecast, the saved base model is wrapped in MultiOutputRegressor, A target matrix Y is constructed with columns y_plus_1 through y_plus_72 by shifting the aqi column backward for each horizon. Only rows that contain the full horizon of targets are kept for training. The multi-output regressor is trained on the same set of engineered features and then saved as models/best_forecast_model.pkl. This approach trains a single estimator to predict the full 72-hour sequence in one call, leveraging the base model's predictive power across horizons.

Model registry (Hopsworks):

Both the base model and the multi-output forecast model are uploaded and registered in the Hopsworks Model Registry under the names:

- aqi_base_model -- single-step predictor

- aqi_forecast_model -- multi-output 72-hour forecaster

Frontend (Streamlit):

The app loads models from local storage (or, when configured, from Hopsworks), reads the latest features, and runs predictions through the base and multi-output models. Outputs are presented as tables and interactive plots showing predicted AQI values over the 72-hour horizon.

CI/CD Pipeline:

Two GitHub Actions workflows automate the project lifecycle. The first workflow is scheduled hourly and runs fetch_aqi_data.py to append new WAQI observations to csv/hourly_aqi_data.csv. The second workflow runs daily and automates the pipeline from feature engineering through model registration: it pulls the latest CSV, runs preprocessing and feature engineering, uploads scaled features to Hopsworks, fetches features for training, performs model tuning and selection, saves model artifacts, and finally registers the models in Hopsworks. These automated workflows ensure timely ingestion and periodic model refresh without manual intervention.

Project Structure:

The overall project directory is organized as follows:

csv/

This directory contains the primary dataset file:

- hourly_aqi_data.csv – Stores the hourly AQI readings collected through the API, serving as the base dataset for preprocessing and model training.

models/

This folder holds all trained models and configuration files related to feature order:

- best_forecast_model.pkl – The final multi-step forecasting model predicting AQI for the next 72 hours.
- best_model.pkl – The base AQI prediction model trained on historical data.
- feature_order.json – Maintains the order of input features to ensure consistency during model inference.

notebooks/

This directory includes Jupyter notebooks for experimentation, feature engineering, and analysis:

- Preprocessing & EDA.ipynb – Covers data cleaning, transformation, and exploratory data analysis.
- Air Quality Index.ipynb – Includes the full pipeline from feature engineering to model training, evaluation, and registration on Hopsworks.

src/

This directory contains the core Python scripts responsible for model training, automation, and the frontend interface:

- app.py – Implements the Streamlit frontend for visualizing AQI forecasts interactively.
- fetch_aqi_data.py – Periodically fetches new AQI data from the external API and appends it to the dataset (executed hourly).
- train_model.py – Performs daily model retraining, feature engineering, evaluation, and model storage on Hopsworks.

venv/

Contains the isolated Python 3.11 virtual environment, which ensures reproducibility and consistent dependency management across systems.

.github/workflows/

This directory contains automation workflows defined for continuous integration and deployment using GitHub Actions:

- fetch_aqi.yml – Schedules hourly execution of fetch_aqi_data.py to update the dataset.
- train_model.yml – Automates daily retraining by running train_model.py.

Supporting Files

- README.md – Provides an overview of the project, setup instructions, and usage guidance.
- requirements.txt – Lists all Python dependencies required for the project.
- .gitignore – Specifies files and folders to exclude from version control (e.g., venv, cached files, model artifacts).