AIR QUALITY INDEX (AQI) PREDICTION SYSTEM

Faisalabad, Pakistan

Table of Contents

1 INTRODUCTION

This project develops a real-time Air Quality Index prediction system for Faisalabad, Pakistan. The system fetches live weather and pollution data, uses machine learning to predict air quality for the next 3 days, and displays the results on an interactive web dashboard.

The primary objective was to create an end-to-end MLOps pipeline that demonstrates the complete lifecycle of a machine learning project, from data collection to deployment.

2 PROJECT OVERVIEW

The system consists of several integrated components working together:

2.1 Data Collection

The system collects weather and pollution data every hour from the OpenWeather API. This includes temperature, humidity, wind speed, atmospheric pressure, and pollutant concentrations such as PM2.5, PM10, nitrogen dioxide, ozone, carbon monoxide, and sulfur dioxide.

2.2 Database

All collected data is stored in MongoDB Atlas, a cloud-based database that provides reliable and scalable storage.

2.3 Feature Engineering

Raw data is transformed into 34 engineered features that help the machine learning model identify patterns. These include time-based features, lag features (past values), rolling averages, and change features.

2.4 Machine Learning Model

A Random Forest model is trained to predict PM2.5 concentrations for 24, 48, and 72 hours ahead simultaneously. This Multi-Output approach allows a single model to make predictions for all three time horizons at once.

2.5 Dashboard

A Streamlit web application displays current air quality conditions and the 3-day forecast with color-coded health recommendations.

2.6 Automation

GitHub Actions pipelines run automatically to collect data every hour and retrain the model daily.

3 METHODOLOGY

3.1 Data Collection Process

The OpenWeather API provides both current weather conditions and air pollution data for any location. The system makes API calls every hour using the latitude and longitude coordinates of Faisalabad (31.4504°N, 73.1350°E).

Each API call returns temperature in Celsius, humidity percentage, wind speed, atmospheric pressure, and concentrations of six pollutants: PM2.5, PM10, NO2, O3, CO, and SO2.

3.2 Feature Engineering

Feature engineering is the process of creating new variables from raw data to help the model learn better. The following types of features were created:

3.2.1 Time Features

Hour of day, day of week, month, and weekend indicator. These help the model understand when pollution levels typically rise or fall.

3.2.2 Lag Features

Values from 1, 3, 6, 12, and 24 hours ago. If pollution was high yesterday at this time, it's likely high today too.

3.2.3 Rolling Averages

Average values over the last 3, 6, 12, and 24 hours. These smooth out random fluctuations and reveal underlying trends.

3.2.4 Change Features

How much values changed in the last few hours. This shows whether pollution is increasing or decreasing.

3.3 Target Variable Selection

A significant challenge was that the OpenWeather API returns air quality as discrete categories (only 5 levels: 1 through 5). Due to consistently high pollution in Faisalabad, 97 percent of the data had the same AQI category, making it impossible for the model to learn meaningful patterns.

The solution was to use PM2.5 concentration (measured in micrograms per cubic meter) as the prediction target instead of the categorical AQI. PM2.5 provides continuous values ranging from about 8 to over 700, giving the model sufficient variation to learn patterns.

For the dashboard display, predicted PM2.5 values are converted back to AQI using the standard EPA breakpoint formulas, so users see the familiar 0-500 AQI scale.

3.4 Machine Learning Approach

Three models were evaluated: Ridge Regression, Random Forest, and XGBoost.

3.4.1 Ridge Regression

Ridge Regression is a linear model with regularization. It's fast and interpretable but may not capture complex patterns.

3.4.2 Random Forest

Random Forest is an ensemble method that builds many decision trees and averages their predictions. It handles non-linear relationships well and natively supports predicting multiple outputs at once.

3.4.3 XGBoost

XGBoost is a gradient boosting algorithm known for winning machine learning competitions. For multi-output prediction, it requires a wrapper because it doesn't natively support multiple targets.

The multi-output approach trains a single model that predicts PM2.5 for 24, 48, and 72 hours ahead simultaneously. This is better than training three separate models because the predictions for different horizons are naturally correlated, and the model can learn from these relationships.

Random Forest performed best in testing and was selected as the production model.

3.5 Cross-Validation

For time series data, random train-test splits are inappropriate because they mix future and past data, causing data leakage. Instead, TimeSeriesSplit validation was used, which always trains on earlier data and tests on later data, simulating how the model would be used in production.

4 CHALLENGES AND SOLUTIONS

4.1 Challenge: Identical Target Values

4.1.1 Problem

97 percent of AQI values were the same (category 5), preventing the model from learning.

4.1.2 Solution

Switched to PM2.5 concentration as the target, which has continuous values with sufficient variation. The model's ability to learn improved significantly after this change.

4.2 Challenge: GitHub Actions Not Running

4.2.1 Problem

Scheduled pipelines were defined but not executing automatically.

4.2.2 Solution

GitHub Actions requires at least one successful push-triggered run before schedules begin working. Added a push trigger to the workflow files, and after the first successful run, the scheduled jobs started working correctly.

4.3 Challenge: Data Leakage

4.3.1 Problem

Initial random train-test splits allowed future data to leak into training.

4.3.2 Solution

Implemented TimeSeriesSplit validation, which respects temporal order and provides more realistic performance estimates.

4.4 Challenge: Estimation Instead of Prediction

4.4.1 Problem

The initial approach trained only a 24-hour model and estimated 48 and 72 hour predictions by adding arbitrary percentages.

4.4.2 Solution

Implemented a true Multi-Output model that trains on actual 48 and 72 hour targets from historical data.

5 RESULTS

The Random Forest Multi-Output model was selected as the best performer. The model predicts PM2.5 concentration for 24, 48, and 72 hours ahead, which is then converted to AQI for display.

With the current limited data (approximately 700 records spanning several weeks), the model shows modest but improving performance. As more data is collected over time, prediction accuracy is expected to improve.

The dashboard successfully displays:

- Current air quality with color coding based on severity
- 3-day forecast with predicted AQI values
- Health recommendations appropriate for the predicted air quality level
- Current weather conditions
- Breakdown of individual pollutant concentrations

6 TECHNOLOGIES USED

- Programming Language: Python 3.10
- Data Processing: Pandas for data manipulation, NumPy for numerical operations
- Machine Learning: Scikit-learn for model training and evaluation, XGBoost for gradient boosting
- Web Dashboard: Streamlit for the interactive web interface, Plotly for charts
- Database: MongoDB Atlas for cloud-based data storage
- APIs: OpenWeather API for weather and pollution data
- DevOps: GitHub for version control, GitHub Actions for automation
- Python Packages: pandas, numpy, scikit-learn, xgboost, streamlit, plotly, pymongo, requests

7 FILE STRUCTURE

- The source code folder contains the main Python modules: data fetcher for API calls, database module for MongoDB operations, feature engineering for creating features, and train model for the machine learning pipeline.
- The GitHub workflows folder contains two automation files: one that runs hourly to collect data and engineer features, and one that runs daily to retrain the model.
- The model's folder stores the trained model file.
- The notebooks folder contains exploratory analysis charts and SHAP analysis charts.
- The main application file creates the Streamlit dashboard.

8 HOW TO RUN

To run the project locally:

First, clone the repository from GitHub. Then install the required packages listed in the requirements file. Create an environment file with the MongoDB connection string and OpenWeather API key. Finally, run the Streamlit application.

In production, the pipelines run automatically via GitHub Actions. The feature pipeline collects data every hour, and the training pipeline retrains the model daily.

9 LESSONS LEARNED

- Data quality is more important than algorithm selection. The 97 percent identical AQI problem showed that no algorithm can find patterns that don't exist in the data.
- Time series validation is different from regular cross-validation. Random splits cause data leakage and give misleadingly optimistic results.
- Understanding the API before building is crucial. The OpenWeather API returns simplified AQI categories, not the full 0-500 scale.
- Multi-output models are often better than multiple separate models for related predictions.
- GitHub Actions scheduled workflows need activation through at least one push-triggered run.
- Being honest about model limitations is important. With limited data, performance will be modest, and it's better to acknowledge this than hide it.

10 FUTURE IMPROVEMENTS

10.1 Short term

Short term improvements include collecting more data to improve accuracy, adding more cities, and implementing monitoring to detect model performance degradation.

10.2 Medium term

Medium term improvements include experimenting with deep learning models, adding seasonal features, and incorporating additional data sources.

10.3 Long term

Long term goals include deploying to a cloud platform, developing a mobile application, and creating SMS alerts for hazardous air quality.

11 CONCLUSION

This project demonstrates a complete MLOps pipeline for air quality prediction. It includes automated data collection, feature engineering, machine learning model training with proper validation, and deployment through an interactive dashboard.

The key technical challenge, the 97 percent identical AQI values, was solved by using PM2.5 concentration as the prediction target. This allowed the model to learn meaningful patterns from continuous values rather than being stuck predicting a constant category.

The project successfully implements continuous data collection and model retraining, making it a production-ready system that improves automatically as more data becomes available.