PROJECT SUBMITTED BY: ASHNA IQBAL SHAIKH

10 PEARLS SHINE INTERNSHIP

Air Quality Forecasting System

Project Report

PROJECT LIVE ON :Air Quality Prediction App

1. Introduction

In this project, I developed an end-to-end air quality forecasting system that predicts the AQI (Air Quality Index) for the next 3 days. The objective was to design a complete ML pipeline starting from raw data ingestion to frontend visualization.

Instead of building only a model, I focused on building a production-style ML system including data ingestion, feature store integration, model registry, automation, and deployment.

2. Data Collection

I fetched historical weather data using the Open-Meteo API. The API provided structured weather features including:

- Temperature
- Humidity
- Wind speed
- Pressure
- Rain
- Pm25,pm10
- Nitrogen dioxide (no2)
- Oxygen(o2)
- Ozone(o3)
- SulphurDioxid
- aqi

To ensure proper model training, I implemented a backfill mechanism to collect sufficient historical data for training and testing.

3. Data Storage

After fetching the data, I:

- Cleaned and transformed it using Pandas
- Created engineered features
- Stored the processed data in MongoDB
- Integrated the data into Hopsworks Feature Store

4. Feature Engineering

Feature engineering was a major focus of this project.

I created:

- Lag features
- Rolling mean features
- Weather interaction features
- Multi-output AQI targets

At first, I trained the model using a single output target. However, the model evaluation scores were mostly negative, indicating poor performance and instability.

To solve this, I shifted from:

Single target → Multi-target prediction (3 future AQI outputs)

This significantly stabilized the training and improved model consistency.

5. Exploratory Data Analysis (EDA)

Before moving to model training, I performed extensive exploratory data analysis.

My objective during EDA was to:

- Understand feature distributions
- Identify correlations
- Detect anomalies
- Validate assumptions about AQI behavior

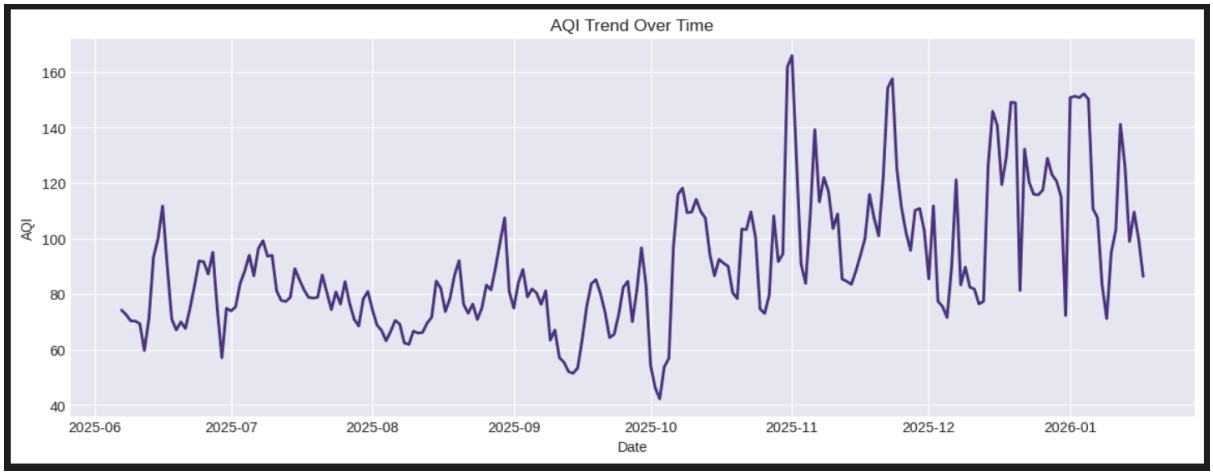
I plotted:

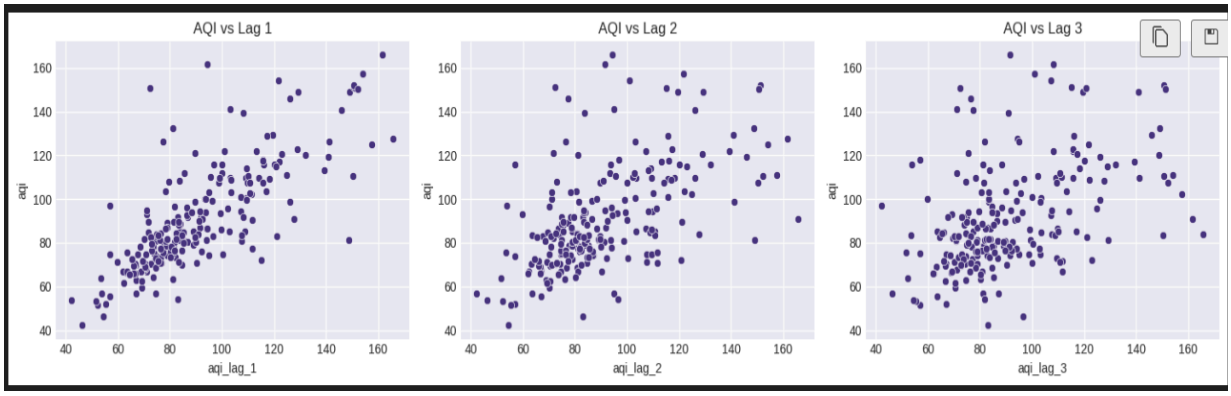
- Feature histograms
- Correlation heatmaps
- Time-series visualizations
- Feature vs AQI scatter plots

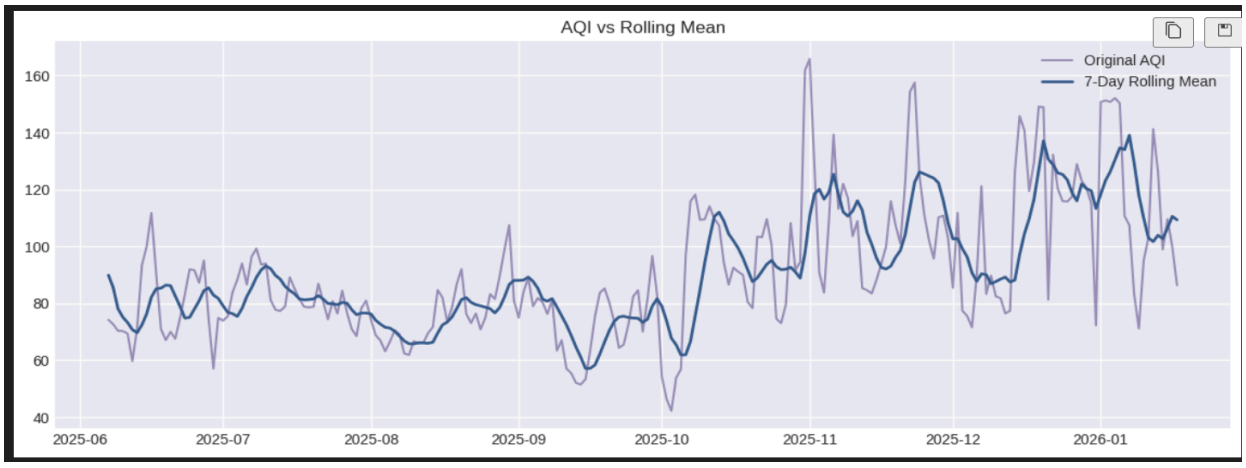
Through EDA, I discovered that:

- AQI had strong temporal dependency.
- Lag features significantly improved stability.
- Wind speed had an inverse relationship with AQI.
- Rainfall had an immediate but short-lived reduction effect.
- Some features had skewed distributions that required scaling.

Some snapshot from my EDA:







EDA directly influenced my feature engineering strategy.

6. Model Training & Registry

I trained regression models using Scikit-learn.

After training:

- I stored trained models in a model registry
- Versioned them properly
- Ensured reproducibility

This made the system production-oriented rather than experimental.

7. Model Explainability using SHAP

After training the model, I wanted to ensure that predictions were interpretable and reliable. Instead of treating the model as a black box, I applied SHAP (SHapley Additive Explanations).

SHAP allowed me to:

- Measure feature importance at global level
- Analyze per-prediction contribution
- Validate whether engineered features were meaningful

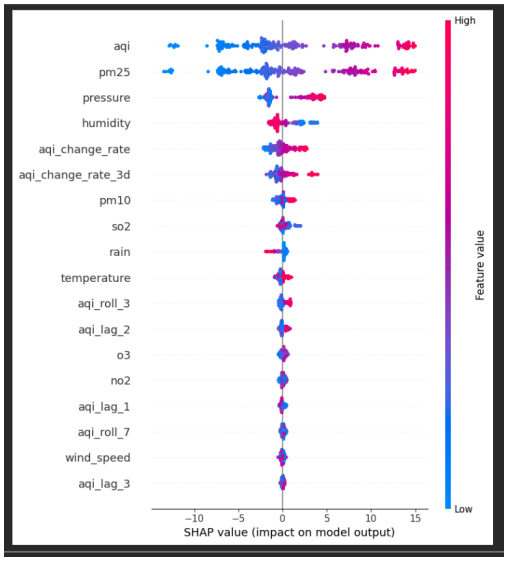
From SHAP analysis, I observed:

- Lag-based AQI features were among the most influential predictors.
- Pressure had strong positive SHAP values in certain ranges.
- Wind speed often contributed negatively toward AQI prediction.
- Humidity had a moderate but consistent influence.

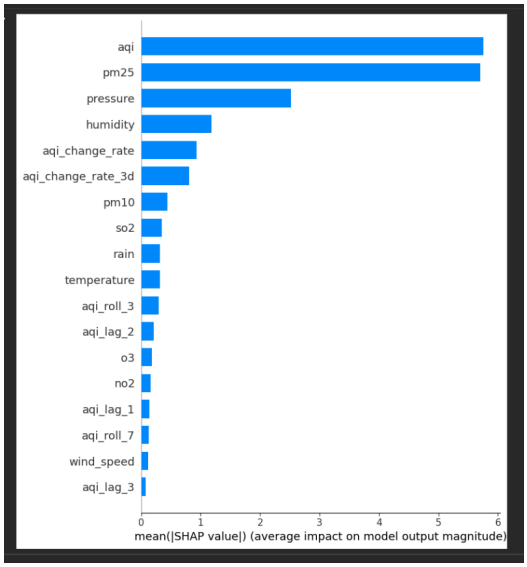
- Some engineered features had lower contribution than expected, which helped refine my feature selection.

This analysis increased my confidence in the model's behavior and improved transparency.

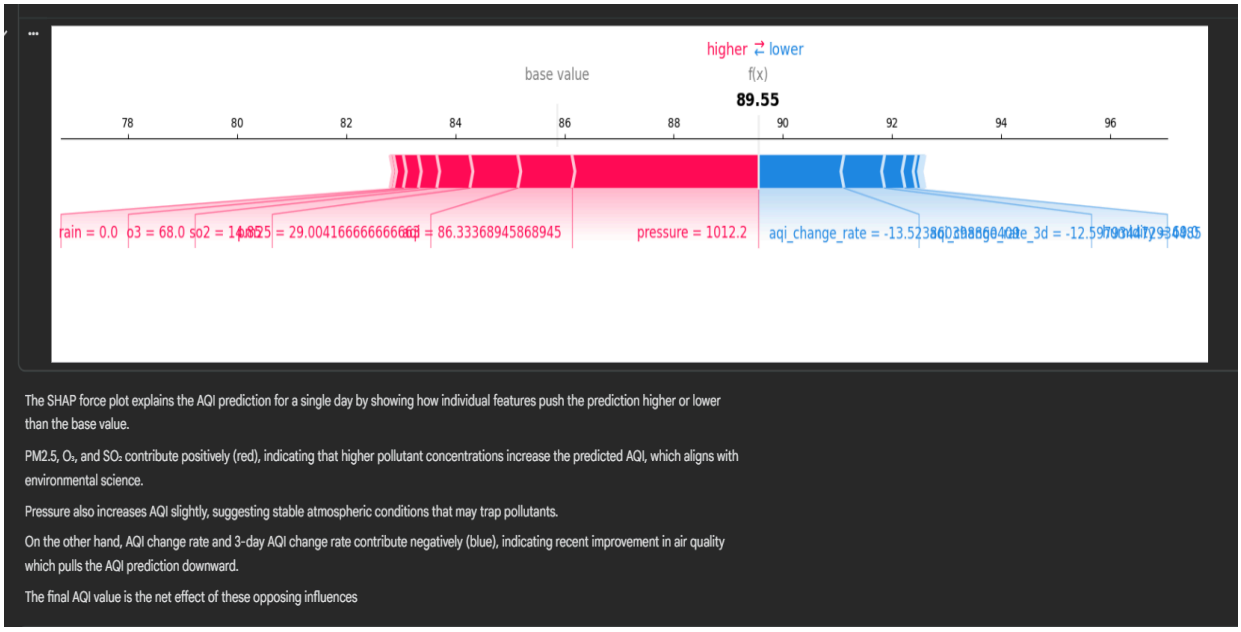
SHAP EXPLAINER:



FEATURE IMPORTANCE:



SHAP FORCE PLOT:



8. Major Challenges & How I Solved Them

1️⃣ Hopsworks Feature Store Issue

One major issue I faced was that Hopsworks was uploading only the schema of the feature group, but the actual feature values were not loading correctly.

I debugged this by:

- Verifying dataframe structure before upload
- Checking feature group primary keys
- Ensuring schema alignment
- Re-uploading feature groups with correct configurations

Eventually, I identified the mismatch between dataframe structure and expected schema.

2️⃣ Negative Model Scores

Initially, my model training scores were mostly negative.

This happened because:

- Target definition was weak
- Feature relationships were not stable
- Single-output prediction was too noisy

To fix this:

- I redesigned the target variable
- Shifted to 3-output prediction
- Reworked feature engineering

This improved stability and performance.

3️⃣ Pipeline Automation Failure

During inference automation, my pipeline kept failing.

The reason:

- Target columns were missing at inference time
- Feature order was inconsistent

This was a critical production-level issue.

I solved it by:

- Separating training and inference feature sets
- Removing target columns during inference
- Enforcing strict feature ordering before prediction

This ensured that the model received features in the exact same order as during training.

4️⃣ Feature Order Mismatch

Another issue was incorrect feature ordering during prediction.

Since Scikit-learn models depend strictly on column order, even small ordering mismatches caused wrong predictions.

I fixed this by:

- Saving feature column order during training
- Reusing the exact same order during inference

5️⃣ Frontend Transition

Initially, I built the frontend using Streamlit.

However, I wanted to challenge myself and learn React, so I rebuilt the frontend using React (Vite).

This was my first time using React in a full ML project.

Challenges included:

- Component structuring
- State management
- API integration
- Styling from scratch
- Responsive layout issues

But it gave me much better control over UI design and deployment flexibility.

9. Final System Features

The final system includes:

- Automated data ingestion
- Feature store integration
- Model registry
- Multi-output regression
- Automated inference pipeline

- Backend API (FastAPI)
- React frontend dashboard
- 3-day AQI forecast
- Interactive trend chart
- Weather highlights

10. Deployment

I deployed the AQI Forecasting System as a full-stack production application.

I developed the backend using **FastAPI** and deployed it on **Railway**, where it serves the trained machine learning model through REST API endpoints (such as `/predict`). The backend is connected to **MongoDB Atlas**, which I used as a feature store to store engineered features and historical data.

Backend URL: Backend URL

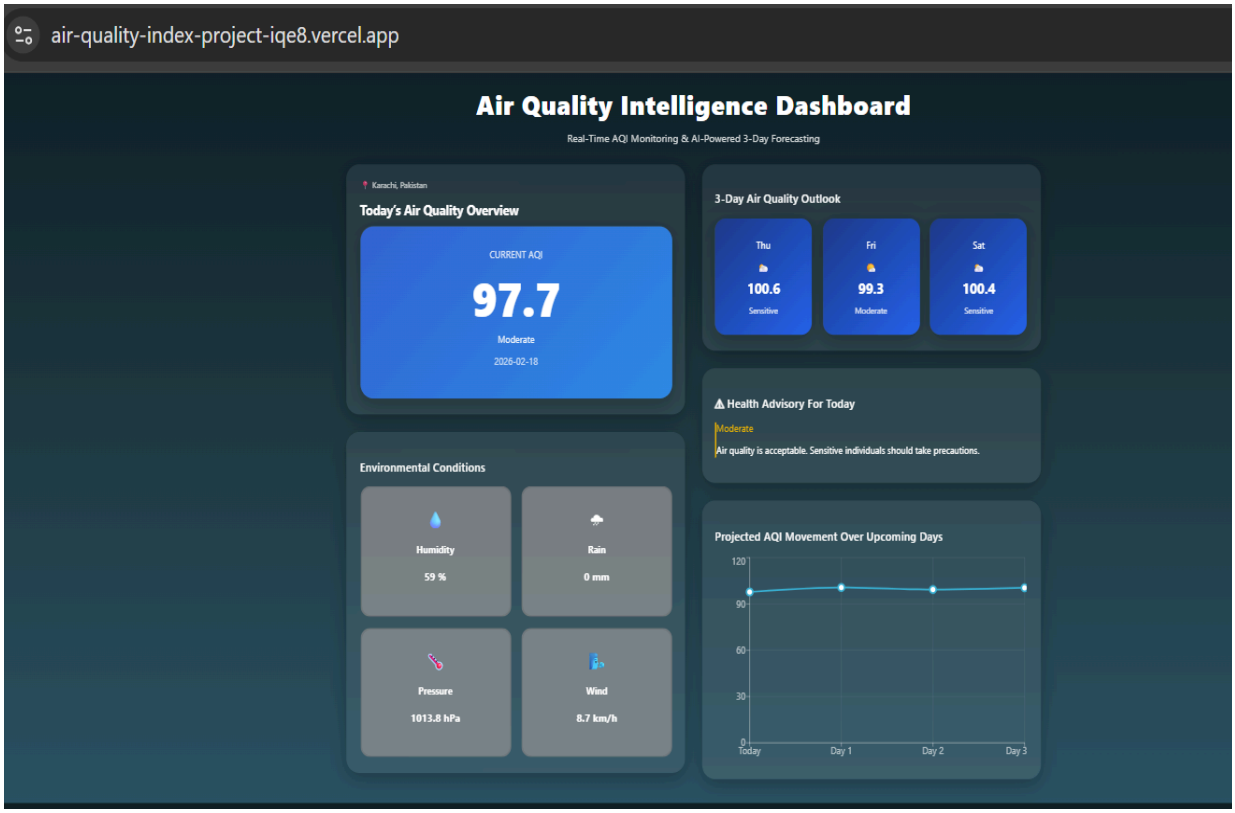
For the frontend, I built the user interface using **React (Vite)** and deployed it on **Vercel**. The frontend communicates with the Railway-hosted backend through secure API requests to fetch AQI forecasts and display them in an interactive interface.

App URL: Frontend URL

I configured environment variables in both Railway and Vercel to securely manage database credentials and API URLs. After deployment, I verified the system functionality using browser developer tools to ensure successful API communication (HTTP 200 responses).

Through this deployment, I ensured that the system is publicly accessible, scalable, and production-ready.

A little sneak-peak to my app:



11. Conclusion

This project evolved from a simple AQI predictor into a complete ML system with automation, registry, and frontend visualization.

The biggest improvement was shifting from experimentation to production-style design.

This experience significantly strengthened my understanding of machine learning engineering and system design.