AQI Prediction Project – Complete Journey Report

1. Project Vision and Objective

The main goal of this project was to build a complete AQI forecasting system that predicts the next 3 days of Air Quality Index for Okara using historical environmental data. My objective was to design a full pipeline that includes data preparation, model training, experiment tracking, explanation, automation, and deployment.

I approached this project as if I were building a real production system instead of just a notebook experiment. Every step was structured carefully so that the system could run automatically and be deployed properly.

2. Data Preparation and Target Creation

I started by collecting historical AQI and environmental data. After collecting and cleaning the dataset, I designed a multi-step forecasting setup.

Since I wanted to predict the next 72 hours, I created a multi-output target where each row predicts:

- AQI at t+1
- AQI at t+2
- …
- AQI at t+72

This was done using shifting logic so that the model learns to predict multiple future steps at once.

For training and testing, I used time-based splitting instead of random splitting. The first 80% of the data was used for training and the remaining 20% was used for testing. This preserved the time order, which is very important in forecasting problems.

3. Feature Engineering

After preparing the target, I worked on feature engineering.

I initially created many features including time-based features and pollutant-based features. Later, I redesigned the feature set to make it more realistic and robust.

I created lag-based features so the model can learn patterns from historical AQI values. These lag features helped the model understand trends and temporal dependencies properly.

The final feature set mainly included:

- Time-based features
- Lag-based features
- Pollutant based features

This made the system more stable and suitable for real forecasting.

4. Model Experimentation

During experimentation, I trained multiple models:

- Random Forest
- XGBoost
- MLP (Neural Network)

Since the target had 72 outputs, I used multi-output handling techniques where required.

I evaluated models using MAE, RMSE, and MAPE metrics. After comparing results carefully, XGBoost performed the most consistently and gave stable predictions.

I tuned important parameters such as learning rate, max depth, and number of estimators to improve generalization.

5. Dagshub Integration

Once the model was stable, I integrated Dagshub for experiment tracking.

Using Dagshub model registry , I was able to:

- Log model parameters
- Log evaluation metrics
- Log trained models
- Log SHAP plots as artifacts
- Register models for version control

This allowed me to compare multiple runs and maintain a clean training history. Instead of manually tracking experiments, everything was automatically stored and versioned.

6. SHAP Explainability

After training the final model, I implemented SHAP for feature importance analysis.

SHAP helped me understand which features contribute most to AQI prediction. I generated SHAP summary plots and logged them as artifacts in MLflow.

Later, I integrated this into the Streamlit application so that the SHAP plot could be displayed directly in the user interface.

This added transparency to the system by showing how the model makes decisions.

7. Automation with GitHub Actions

After experimentation, I moved to automation.

I created a dedicated training pipeline script and configured GitHub Actions to:

- Install dependencies
- Train the model automatically
- Log metrics
- Push results to MLflow

Now whenever the pipeline runs, everything is executed automatically without manual intervention.

This transformed the project from a manual ML experiment into a production-like system.

8. Streamlit Deployment

Finally, I built a Streamlit web application.

The application:

- Loads the latest registered model from Dagshub
- Makes AQI predictions
- Provides an interactive interface

This completed the full workflow:

Data → Model → Tracking → Explanation → Automation → Web App

Challenges Faced and Their Solutions

1. Feature Selection and Data Leakage

The first major challenge happened during feature engineering.

Initially, I created many new features including both time-based and pollutant-based features. When I trained the model, the accuracy went almost close to 100%, which looked impressive but unrealistic.

After deeper analysis, I discovered the real problem: data leakage.

AQI is calculated using pollutant values. When I included pollutant-based features directly in training, the model was indirectly reverse engineering AQI because those pollutants are already used in AQI computation. This caused the model to almost perfectly predict AQI.

To solve this:

- I carefully reviewed the relationship between AQI and pollutants.
- I removed pollutant-based features that directly leak target information.
- I created lag-based features instead.
- I trained the model only on time-based and lag-based features.

After this correction, performance became realistic and stable.

This was an important lesson about leakage and how easily it can create fake high accuracy.

2. Slow Model Loading on DagsHub

Another issue I faced was very slow model loading on DagsHub.

Initially, the model was large and complex. This increased artifact size and slowed down loading and deployment.

To fix this:

- I reduced model complexity.
- Controlled max depth.
- Reduced unnecessary estimators.
- Optimized parameters.

After reducing model size, loading became smooth and deployment performance improved.

3. Poor Neural Network Performance

When I trained the MLP model, its performance was very poor compared to tree-based models.

After investigation, I realized that neural networks are very sensitive to feature scaling.

My data was not scaled properly.

To solve this:

- I applied proper feature scaling.
- Retrained the MLP model.

The performance improved significantly. MAPE reduced from around 40% to nearly 10%.

This helped me understand the importance of preprocessing when working with neural networks.

4. Automation Issues (CI/CD Problems)

During automation, GitHub Actions pipeline failed multiple times.

The main issue was related to:

- GitHub secrets configuration
- Password handling
- Authentication for MLflow / DagsHub

The pipeline was failing because credentials were not configured correctly.

To fix this:

- I properly configured GitHub Secrets.
- Verified environment variables.
- Debugged logs carefully.
- Ensured secure credential handling.

After fixing this, the automation pipeline ran successfully.

5. SHAP Analysis

Before this project, I had never used SHAP and did not fully understand how model explainability works.

When I first implemented:

explainer = shap.TreeExplainer(model)

I started getting errors, especially related to version incompatibility between XGBoost and SHAP.

GitHub Actions pipeline kept failing due to this issue.

After investigation, I discovered that newer versions of XGBoost had compatibility issues with SHAP.

To fix this:

- I checked installed versions.

- Identified incompatibility.
- Downgraded XGBoost to a stable compatible version.
- Regenerated SHAP plots.
- Logged them properly as PNG artifacts.

After that, SHAP worked correctly both locally and in CI/CD.

This challenge taught me:

- Library version compatibility is critical.
- Debugging production pipelines is very different from notebook debugging.
- Explainability tools require proper understanding of model internals.

Final Reflection

This project started as a simple AQI prediction idea, but it evolved into a complete machine learning system.

I learned that real ML work is not just about training models. It involves:

- Careful feature design
- Avoiding leakage
- Handling preprocessing
- Experiment tracking
- Debugging automation
- Managing dependencies
- Deploying reliable systems

This project significantly improved my understanding of production-level machine learning systems.