AQI Prediction Project – Complete Journey Report

1. Project Idea and Goal

The goal of this project was to build a system that can predict Air Quality Index (AQI) using historical pollution and weather data. The main objective was not only to train a machine learning model, but to build a complete pipeline that includes training, evaluation, tracking, explanation, automation, and deployment.

I wanted this project to be practical, not just a notebook experiment. So I focused on building it in a structured way like a real-world ML system.

2. Data Understanding and Feature Selection

The first major challenge I faced was feature selection.

Initially, I used many available columns directly while creating the model. The model showed very high training accuracy, but when I evaluated it on test data, performance dropped. It was clearly overfitting.

After analyzing the issue carefully, I realized the problem was target leakage.

Some of the features I included were directly related to AQI calculation itself. Since AQI is calculated using pollutant values, using those values without careful handling caused the model to indirectly “see” the answer.

That is why training performance was unrealistically good.

To fix this:

- I reviewed each feature carefully and find out that components that come by data fetching are actually leaking target information.
- I removed columns that directly leak target information.
- I avoided using future-based values, columns like chemical components.
- I restructured the feature set properly created lag features and used only time based and lag features for training

After cleaning the features, the model performance became more realistic and stable. That was an important learning point in the project.

3. Model Experimentation Phase

In the experimentation phase, I tried different models.

First, I trained baseline models to understand how the data behaves. Then I moved to more powerful models like:

- XGBoost
- Random Forest
- MLP (Neural Network)

I compared their performance using proper train-test split.

At first, models were unstable because:

- Data scaling was not handled properly.
- Multi-output handling was tricky.
- Hyperparameters were not tuned.

I improved this step by step:

- Used proper preprocessing.
- Applied scaling where required.
- Wrapped models using MultiOutputRegressor where needed.
- Tuned parameters like learning rate, max depth, and number of estimators.

Eventually, XGBoost gave the most stable and strong performance.

4. Overfitting and Model Stability

Another challenge was overfitting again during experimentation.

When I increased model complexity, training accuracy increased but generalization dropped.

To fix this, I:

- Reduced max_depth
- Controlled learning rate
- Increased estimators carefully
- Checked validation metrics instead of only training metrics

This helped me build a more stable model.

5. MLflow Integration

After building a working model, I wanted proper experiment tracking.

So I integrated MLflow.

This allowed me to:

- Log model metrics
- Log parameters
- Log artifacts like SHAP plots
- Track multiple runs

At first, I had issues logging artifacts properly, especially images. I had to:

- Save SHAP plot as PNG
- Log it using `mlflow.log_artifact`
- Later retrieve it in Streamlit

This helped me structure the project like a real ML system instead of just scripts.

6. SHAP Integration

The most difficult technical issue in this project was SHAP with XGBoost.

When I tried:

```
explainer = shap.TreeExplainer(model)
```

I started getting errors.

The error was related to:

```
ValueError: could not convert string to float: '[3.859387E0]'
```

At first, I thought the issue was in my code.

But after checking versions, I found:

```
XGBoost version: 3.1.3
SHAP version: 0.49.1
```

The issue was version incompatibility.

New versions of XGBoost changed internal format of base_score. SHAP could not parse it properly.

I tried:

- Using shap.Explainer
- Using booster object
- Wrapping inside try-except

But GitHub Actions kept failing.

Finally, the correct solution was:

Downgrade XGBoost to:

```
xgboost==1.7.6
```

After downgrading, SHAP worked perfectly.

This was an important lesson:

Version compatibility matters a lot in ML pipelines.

7. CI/CD Automation Phase

After experimentation, I moved to automation.

I created a training pipeline script:

```
automation/training_pipeline.py
```

Then I configured GitHub Actions to:

- Install dependencies
- Run training automatically
- Log metrics
- Generate SHAP plots
- Fail if training crashes

During CI runs, many issues appeared:

- SHAP crashes
- Version mismatch
- Artifact saving problems
- Path issues

Each time I debugged using GitHub logs.

Eventually, I stabilized the pipeline.

Now:

- Model trains automatically
- SHAP plot is generated
- MLflow logs everything
- No manual intervention required

This made the project production-like.

8. Streamlit Integration

Next, I integrated the trained model into Streamlit.

The goal was to:

- Load latest MLflow run
- Download SHAP artifact
- Display SHAP summary plot
- Make predictions

I created a function to fetch the latest run using MlflowClient and then display the PNG using:

st.image(shap_path)

That completed the end-to-end pipeline:

Data → Model → Tracking → Explanation → UI

9. Key Learnings from the Project

This project taught me several important lessons:

1. Feature selection is critical. Even small leakage can destroy model reliability.
2. Overfitting is easy to create but hard to detect if you only look at training accuracy.
3. Version compatibility between libraries is extremely important.
4. Automation exposes hidden bugs.
5. CI/CD forces you to write cleaner and more stable code.
6. ML projects are more about debugging and fixing than just training models.

10. Final Architecture Overview

Final system includes:

- Cleaned feature pipeline
- XGBoost model (stable version)
- SHAP explanation
- MLflow experiment tracking
- Automated GitHub Actions pipeline
- Streamlit app for prediction and visualization

This is no longer just a machine learning model. It is a complete ML system.