Journey of 10pearls Pakistan — AQI Prediction Project

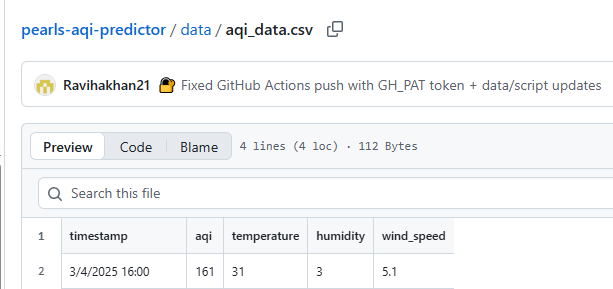
# 🌟 Internship Journey

My data science internship with 10Pearls Pakistan started with weekly sync meetings where we were introduced to our main project: PEARLS AQI Predictor. Before this internship, I had never worked with CI/CD pipelines, GitHub Actions, workflows, or deployment. I only had a little exposure to machine learning from my AI course lab. This project gave me the chance to go much deeper — building data pipelines, training models, automating predictions, and finally deploying a live dashboard.  
  
At first, I struggled with mistakes, broken pipelines, and merge conflicts, but through trial and error I learned how to fix problems step by step. By the end, I not only achieved the project goal but also gained confidence in applying machine learning in the real world.

# 🔎 Step 1: Data Collection — First Attempt (OpenWeather API)

I started with the OpenWeather API, which gives AQI levels (1–5) instead of numeric AQI values. I treated it as a classification problem, mapping:  
- 1 → Good (0–50)  
- 2 → Fair (51–100)  
- 3 → Moderate (101–150)  
- 4 → Poor (151–200)  
- 5 → Very Poor (201+)  
  
I thought this would work, but in a sync meeting I learned that our project is actually a regression problem because the target column is AQI, not categories. This meant I had to switch APIs and change my approach.

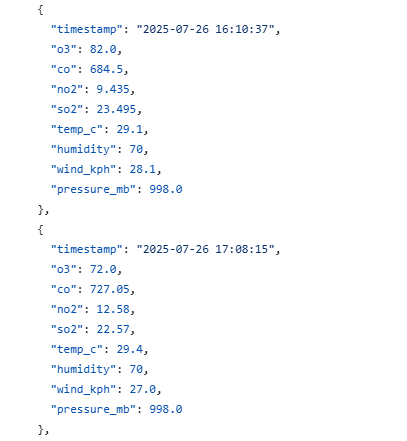
Screenshot of OpenWeather Api



# 🔎 Step 2: Data Collection — Switching to WeatherAPI

Next, I used WeatherAPI to collect pollutants (CO, NO₂, O₃, SO₂) on an hourly basis. At first, I rushed and tried fetching every 10 minutes to gather more data quickly. Later I realized this created duplicate values with only different timestamps, which would make the model overfit.  
  
I fixed the GitHub workflow to fetch data hourly instead. However, I mistakenly kept both OpenWeather and WeatherAPI scripts in my repo, which broke the CI/CD pipeline. After resolving merge conflicts, the pipeline resumed and I began collecting proper hourly data starting from 26 July 2025.

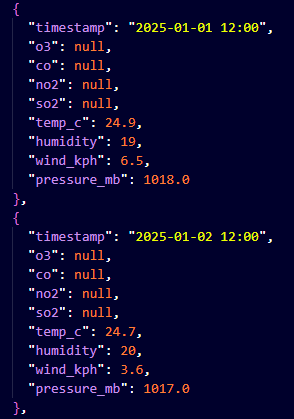
Screenshot of hourly data from 26 july 2025 using WeatherApi



# 🔎 Step 3: Historic Data Collection

Because I started late, I didn’t have enough history to train a good model. At first, I tried to fetch historic data directly from **WeatherAPI**, but I quickly noticed that while the weather columns (temperature, humidity, wind, pressure) were filled, all pollutant values (CO, NO₂, O₃, SO₂) came back as **null**. The timestamps were valid but repeated daily around 12:00, for example:

Screenshot of hourly fetch data using Weather api ( fetch\_historic\_data.py)

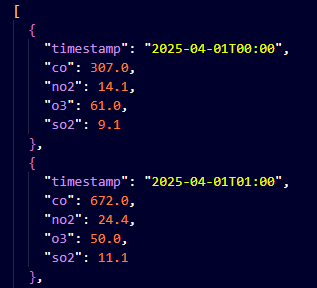


This meant that WeatherAPI’s historic endpoint was **not reliable for pollutant data**.

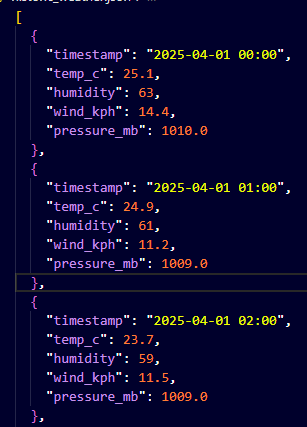
To solve this, I decided to fetch pollutants from **OpenMeteo API** (which provided CO, NO₂, O₃, SO₂ historically) and use WeatherAPI only for weather variables (temperature, humidity, wind, pressure).

This combined approach gave me a proper dataset from **1 April to 21 July 2025** (since the free API only allowed 3 months).

**Screenshot of OpenMeteo Api API (which provided CO, NO₂, O₃, SO₂ historically) using script fetch\_historic\_openmeteo.py and save in historic\_openmeteo\_pollutants.json.**



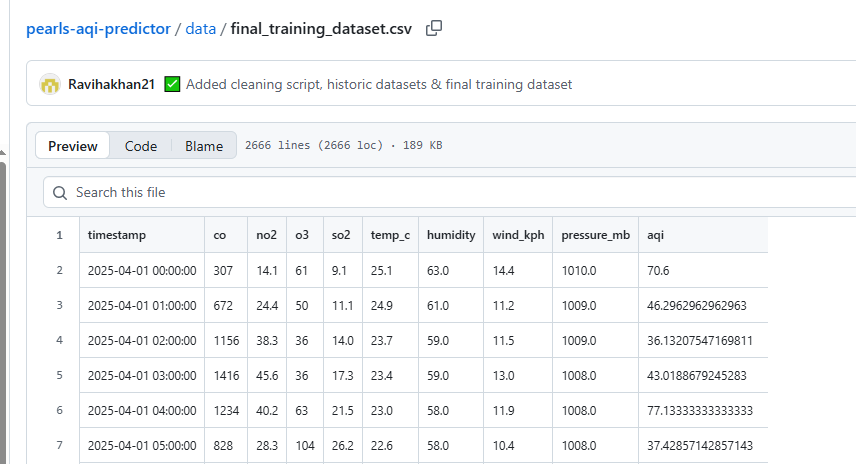
**Screenshot of WeatherApi (which provided CO, NO₂, O₃, SO₂ historically) using script** fetch\_historic\_data.py **and save in** historic\_weather.json.

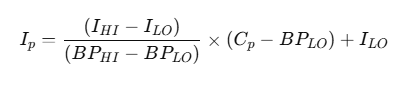


However, I noticed a **gap between 21–26 July**, where no data was available. My dataset only became complete again once my CI/CD pipeline resumed live collection on **26 July**.

# 🔎 Step 4: Data Cleaning & AQI Calculation

I created a script clean\_merge\_data.py to merge pollutant, weather, and AQI data into a single file (final\_training\_dataset.csv).

  
  
Cleaning Steps:  
- Unified timestamps.  
- Removed duplicates and null entries.  
- Restricted dataset to April–July 2025.  
  
**AQI Calculation (EPA Standard Method)**  
Since APIs only provided pollutant concentrations (CO, NO₂, O₃, SO₂) and not hourly AQI, we had to calculate AQI ourselves. This was important because **AQI is our target column for regression**. We used the **US EPA’s piecewise linear formula**:



Where:

* IpI\_pIp​ = AQI sub-index for pollutant ppp
* CpC\_pCp​ = observed pollutant concentration
* BPHI,BPLOBP\_{HI}, BP\_{LO}BPHI​,BPLO​ = breakpoint concentrations that bound CpC\_pCp​
* IHI,ILOI\_{HI}, I\_{LO}IHI​,ILO​ = AQI values corresponding to those breakpoints

For each pollutant, we identified its breakpoint range, calculated the sub-index, and then took the **maximum sub-index** as the final AQI for that timestamp

1. **Merging Measured & Calculated AQI**

* If an AQI value was available (rare cases), we used it.
* If missing (most cases), we replaced it with the calculated AQI.
* This ensured a **continuous, complete AQI column** in our dataset.

1. **Final Dataset**  
   The merged dataset contained:

* Pollutant levels (CO, NO₂, O₃, SO₂)
* Weather variables (temperature, humidity, wind speed, pressure)
* AQI values (calculated target column)

This dataset was then **ready for feature engineering and model training**.

Keypoint:

**" AQI was calculated using the EPA’s standard formula, which converts pollutant concentrations into sub-indices based on defined breakpoints, and the highest sub-index across CO, NO₂, O₃, and SO₂ was taken as the final AQI.”**

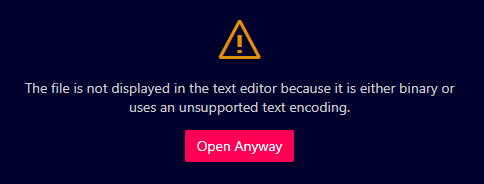
# 🔎 Step 5: Feature Engineering & Model Training

### 1) Local training in VS Code (first attempt)

* I used final\_training\_dataset.csv to start **feature engineering** and **Random Forest training** in VS Code (script-based).
* I dropped unnecessary columns, engineered features, trained the model, calculated **RMSE**, and saved:
  + aqi\_model.pkl (trained model, **binary** Pickle/Joblib file)
  + feature\_importance.csv (human-readable)
* I learned that **.pkl is a binary file** and **cannot be opened** in a text editor like VS Code. Because I couldn’t see rich outputs, I wasn’t sure whether the training was correct.
* I still **pushed the files to GitHub** to keep progress/version history, but decided to **switch to Google Colab** for transparent, step-by-step checks (metrics, plots, importance, etc.).





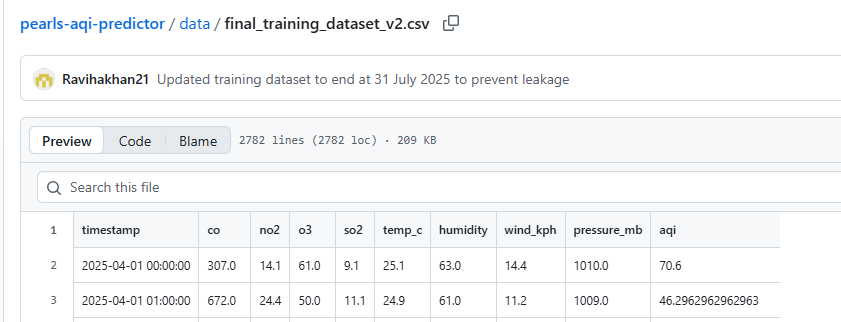


Then first time I got to known that .pkl (Pickle / Joblib file) is a **binary file** – it’s not meant to be opened in a text editor like VS Code. So I decide I will trained my model on collab

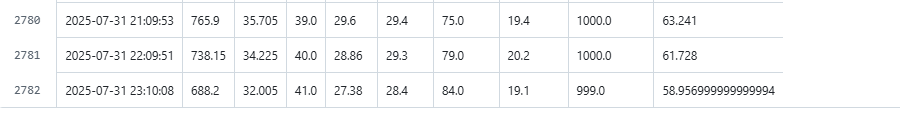
### 2) Preparing final\_training\_dataset\_v2.csv

* As hourly API collection (CI/CD) progressed, I now had **historic data + recent live data**.
* To build a reliable training set, I **merged**:
  + Historic data (1 Apr → 21 Jul 2025)
  + Most recent CI/CD JSON records (26 → 31 Jul 2025)
* Cleaning steps:
  + Filtered desired time range, **removed rows with missing pollutant/weather** values, sorted timestamps.
  + **AQI is our target**; APIs only gave pollutants hourly, so I **calculated AQI (EPA method)** from **CO, NO₂, O₃, SO₂** (PM2.5/PM10 were not allowed).
  + Selected relevant variables, handled gaps by **interpolation + forward/backward fill**.
* Exported the clean training file as **final\_training\_dataset\_v2.csv**.

Screenshot of **final\_training\_dataset\_v2.csv**

**Starting**

Ending



### 3) Moving to Google Colab (feature engineering + training + evaluation)

* In Colab, I rebuilt feature engineering and retrained models with clear outputs.
* **Feature engineering** (reusable pipeline):
  + Time features (hour/day/month, optional cyclical encoding)
  + Lags (e.g., AQI t−1, t−3, t−6), rolling means (3h, 12h), change rate
* **Models tested**: Random Forest, XGBoost, Decision Tree
  + Metrics: **RMSE, MAE, R²**
  + Feature importance inspected
* **Short-term prediction files** prepared because v2 ended at **31 July**:
  + pollutants\_1\_3\_aug.csv → Predict AQI for 1–3 Aug
  + pollutants\_4\_6\_aug.csv → Predict AQI for 4–6 Aug
  + For comparison/ground truth, I also computed **real\_aqi\_1\_3\_aug.csv** and **real\_aqi\_4\_6\_aug.csv** from pollutants (EPA formula).

### 4) Why August predictions were weak with v2

* Predictions for **1–3 Aug** (and forecasts for **4–6 Aug**) were **not good**, because the model **had not seen any August trend** during training (v2 ended at 31 Jul).
* Lesson: time-series models need **recent trend exposure**; training must include the **same distribution** as the prediction window.

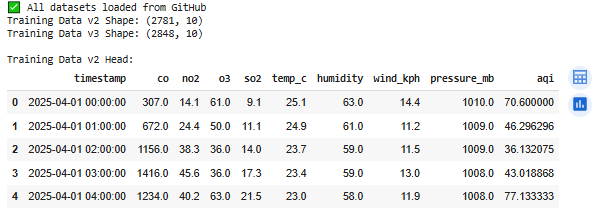
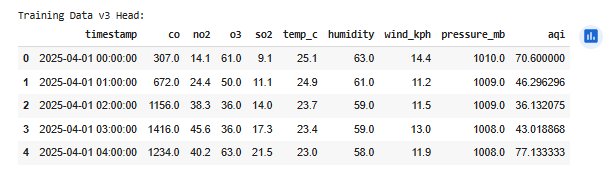
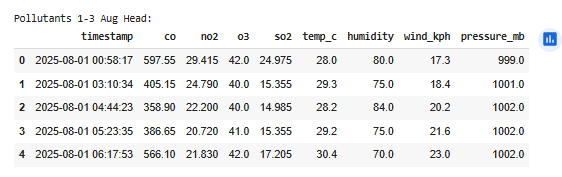
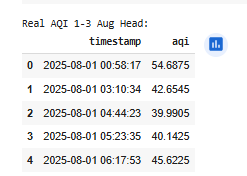
### 5) Upgrading to final\_training\_dataset\_v3.csv

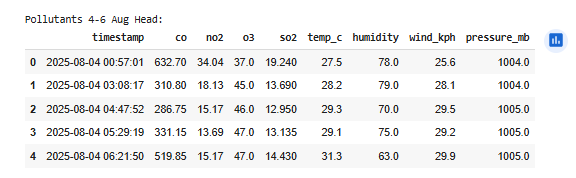
* I created **final\_training\_dataset\_v3.csv** by combining:
  + Everything from **v2**, **plus** the newly available **1–3 Aug** data.
* I used a reusable **feature engineering pipeline** on v3 and **retrained** the models.
* With v3, predictions for **4–6 Aug** improved (better RMSE/MAE/R²) because the model now **saw early August patterns**.
* Script used to assemble v3: **update\_dataset\_v3.py** (produces data up to **3 Aug**).

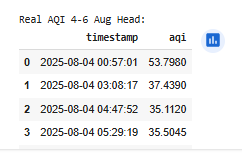
### 6) Model choice and current best setup

* I compared **Random Forest, XGBoost, Decision Tree** on the same engineered features.
* **Random Forest** consistently gave the **best RMSE/MAE** and stable performance, so I selected it as the **production model**.
* A comprehensive explanation of each step and the overall working of the pipeline in google colab.
* 1. Load Datasets:
* The pipeline begins by loading all necessary datasets (`final\_training\_dataset\_v2.csv`, `final\_training\_dataset\_v3.csv`, `pollutants\_1\_3\_aug.csv`, `real\_aqi\_1\_3\_aug.csv`, `pollutants\_4\_6\_aug.csv`, and `real\_aqi\_4\_6\_aug.csv`) directly from a specified GitHub repository using their raw file URLs. This ensures the data is readily accessible for the subsequent steps. The `timestamp` columns are parsed as datetime objects during loading. We load both v2 and v3 training datasets as we will first build a pipeline with v2 and then demonstrate the improvement with v3 for the August 4-6 forecast.

Screenshot

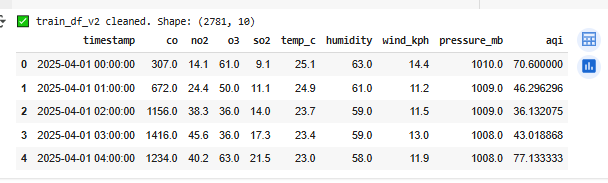
   





* 2. Clean Training Data (v2):
* The `train\_df\_v2` dataset undergoes cleaning to ensure data quality before feature engineering and modeling. This involves sorting the data by timestamp and removing any duplicate entries based on the timestamp. Sentinel values (998 and 999) in the `pressure\_mb` column are replaced with `NaN` (Not a Number). Missing values are then handled by first forward-filling and then backward-filling. Any remaining `NaN` values in numerical columns are filled with the median of their respective columns.

Screenshot:



* 3. Feature Engineering (v2):
* A crucial step is creating relevant features for the models. A reusable `feature\_engineering\_pipeline` function is defined to apply several transformations:
* Time-based features: Extracting hour, day of the week, and month from the timestamp, and also creating sine and cosine transformations of the hour to capture cyclical patterns.
* Lag features: Creating lagged values of the 'aqi' column (1, 3, and 6 hours prior) to incorporate the time series nature of the data.
* Rolling features: Calculating rolling mean of the 'aqi' column over specified window sizes (3 and 12 hours) to capture recent trends.
* Change rate: Calculating the difference in AQI from the previous hour to capture the rate of change.
* The pipeline is applied to the cleaned `train\_df\_v2`. Rows with NaN values introduced by the lag/rolling features at the beginning of the time series are dropped.

Screenshot:



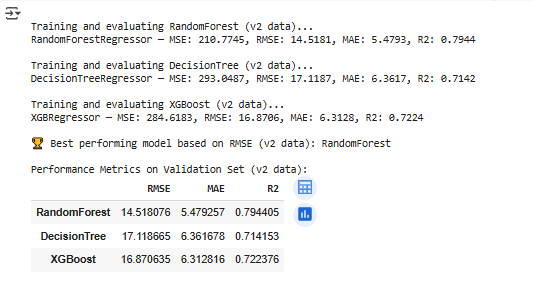
* 4. Prepare Data for Modeling (v2):
* The feature-engineered training data v2 (`df\_fe\_v2`) is prepared for model training and validation. The target variable (`aqi`) is separated from the features (`X\_v2`). A `TimeSeriesSplit` is used to split the data into training and validation sets, respecting the temporal order of the data, which is crucial for time series analysis. The features are then scaled using a `StandardScaler` fitted on the training portion of the split (`X\_train\_v2`) to ensure all features have a similar scale, which can improve model performance. The scaler is fitted \*only\* on the training data to prevent data leakage from the validation set.

Screenshot:



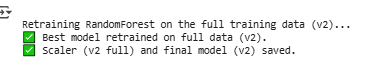
* 5. Train & Evaluate Models (v2):
* Three regression models (Random Forest, Decision Tree, and XGBoost) are instantiated. Each model is trained on the scaled training data (`X\_train\_s\_v2`) and evaluated on the scaled validation data (`X\_val\_s\_v2`) using the `evaluate\_model` function. This function calculates and prints the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R²) metrics. The best-performing model based on the lowest RMSE on the validation set is identified (`best\_model\_name\_v2`).

Screenshot



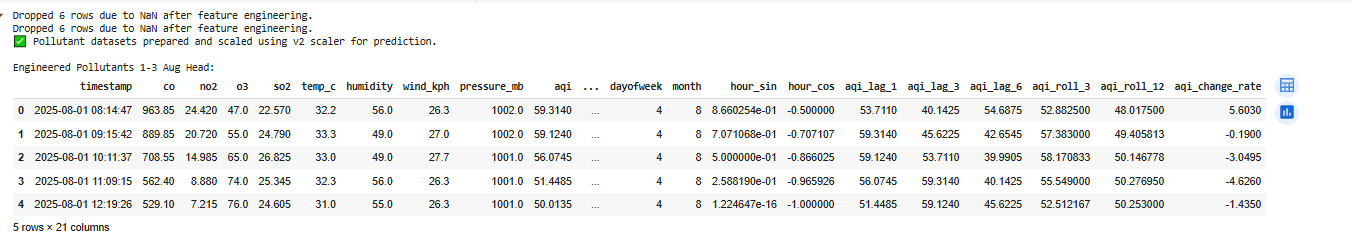
* 6. Retrain Best Model on Full Training Data (v2):
* The best-performing model type identified in step 5 (e.g., RandomForest) is retrained on the \*entire\* feature-engineered training dataset v2 (`df\_fe\_v2`). This uses all available v2 training data to build the final model for prediction. A new `StandardScaler` (`scaler\_full\_v2`) is also fitted on the \*full\* feature set of `df\_fe\_v2`. The retrained model (`best\_model\_v2\_final`) and the full data scaler are saved using `joblib` so they can be loaded and used later for making predictions on new, unseen data.

Screenshot



* 7. Prepare Pollutant Data for Prediction (using v2 scaler):
* The pollutant datasets for August 1-3 (`pollutants\_1\_3`) and August 4-6 (`pollutants\_4\_6`) are prepared for making predictions using the model trained on v2 data. This involves merging the pollutant data with the corresponding real AQI data (`real\_aqi\_1\_3` and `real\_aqi\_4\_6`) based on timestamp. This merge is necessary because the feature engineering pipeline requires the `aqi` column to create lag and rolling features. The same `feature\_engineering\_pipeline` used for training data is applied to these merged dataframes (`df\_1\_3\_fe\_v2` and `df\_4\_6\_fe\_v2`). Finally, the features of these prepared datasets (`X\_1\_3\_s\_v2` and `X\_4\_6\_s\_v2`) are scaled using the `StandardScaler` (`scaler\_full\_v2`) that was fitted on the full v2 training data in step 6. Using the same fitted scaler is crucial for consistent scaling between training and prediction.

SCREENSHOT





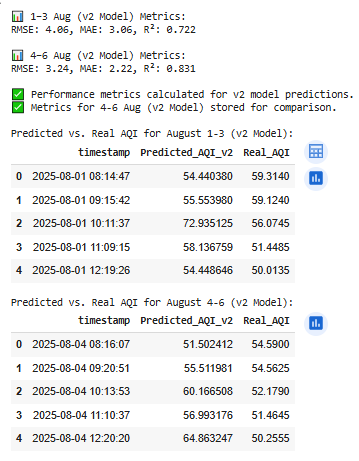
* 8. Predict AQI (using v2 model):
* The retrained best model trained on v2 data (`best\_model\_v2\_final`), loaded from the saved file, is used to generate AQI predictions for the prepared and scaled pollutant datasets (`X\_1\_3\_s\_v2` and `X\_4\_6\_s\_v2`).

Screenshot



* 9. Evaluate Predictions (v2 model):
* The generated predictions for August 1-3 (`preds\_1\_3\_v2`) and August 4-6 (`preds\_4\_6\_v2`) from the v2 model are compared against their corresponding real AQI values. DataFrames (`predictions\_aug1\_3\_v2` and `predictions\_aug4\_6\_v2`) are created to hold the predicted and real values along with timestamps. Performance metrics (RMSE, MAE, R²) are calculated and printed for both periods. Crucially, the performance metrics for the August 4-6 predictions from the v2 model (`rmse\_4\_6\_v2`, `mae\_4\_6\_v2`, `r2\_4\_6\_v2`) are stored in a dictionary (`v2\_4\_6\_metrics`) for direct comparison with the v3 model's performance on the same period later. The head of the prediction dataframes is also displayed in a tabular format.

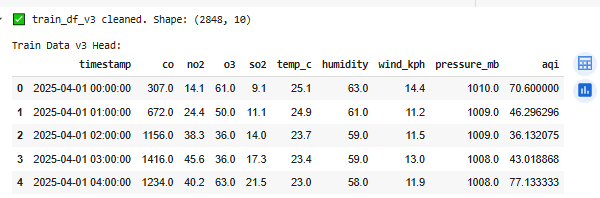
Screenshot

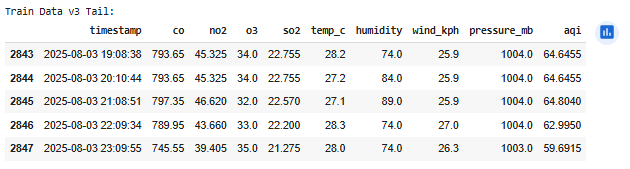


10. Clean Training Data (v3):

* The `train\_df\_v3` dataset undergoes the same cleaning process as the v2 data: sorting by timestamp, dropping duplicates, replacing sentinel values with NaN, and filling missing values using ffill/bfill and median imputation. Displaying the head and tail of the cleaned dataframe confirms its structure and the time range of the data.

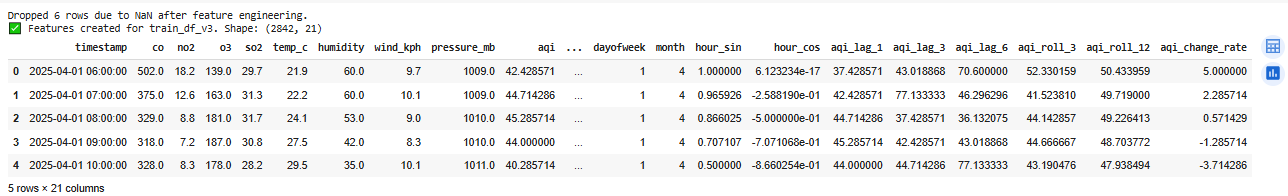
Screenshot





* 11. Feature Engineering (v3):
* The reusable `feature\_engineering\_pipeline` is applied to the cleaned `train\_df\_v3` dataset. This creates the same set of time-based, lag, rolling, and change rate features as were created for the v2 data. The resulting feature-engineered dataframe is stored as `df\_fe\_v3`.

Screenshot



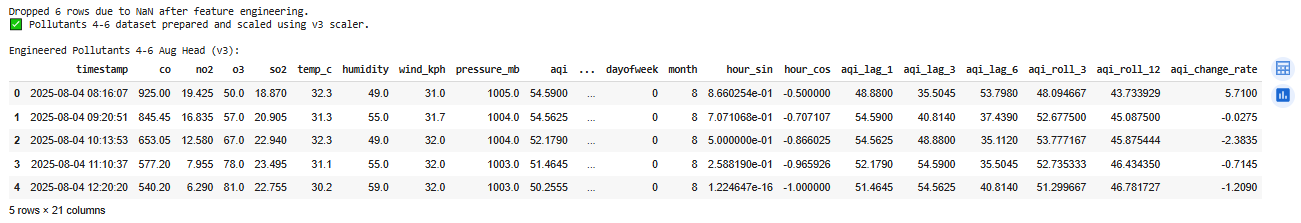
* 12. Retrain Best Model on Full Training Data (v3):
* The same best-performing model type identified from the v2 evaluation (`best\_model\_name\_v2`, which was RandomForest) is now retrained on the \*entire\* feature-engineered training dataset v3 (`df\_fe\_v3`). A new `StandardScaler` (`scaler\_full\_v3`) is fitted on the \*full\* feature set of `df\_fe\_v3`. This retrained v3 model (`best\_model\_v3\_final`) and the new v3 scaler are saved using `joblib`. This step prepares the final model that leverages the extended training data for improved forecasting.

Screenshot



* 13. Prepare Pollutant Data for Prediction (using v3 scaler):
* The `pollutants\_4\_6` dataset (for August 4-6) is prepared for prediction using the model trained on v3 data. This involves merging it with `real\_aqi\_4\_6`, applying the `feature\_engineering\_pipeline`, and importantly, scaling the features using the `StandardScaler` (`scaler\_full\_v3`) that was fitted on the \*full v3 training data\* in the previous step. This ensures the new data is scaled consistently with the v3 training data. The resulting engineered dataframe is `df\_4\_6\_fe\_v3`, and the scaled feature array is `X\_4\_6\_s\_v3`.

Screenshot



14. Predict AQI (using v3 model):

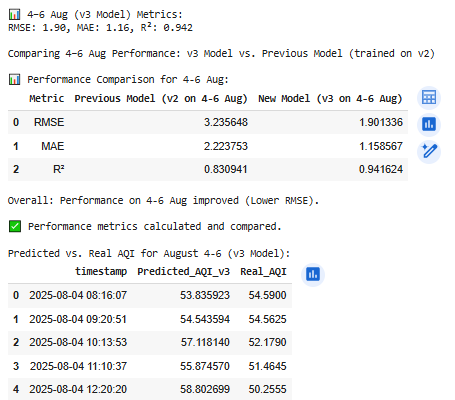
* The retrained best model trained on v3 data (`best\_model\_v3\_final`), loaded from the saved file, is used to generate AQI predictions (`preds\_4\_6\_v3`) for the prepared and scaled August 4-6 pollutant dataset (`X\_4\_6\_s\_v3`).

Screenshot



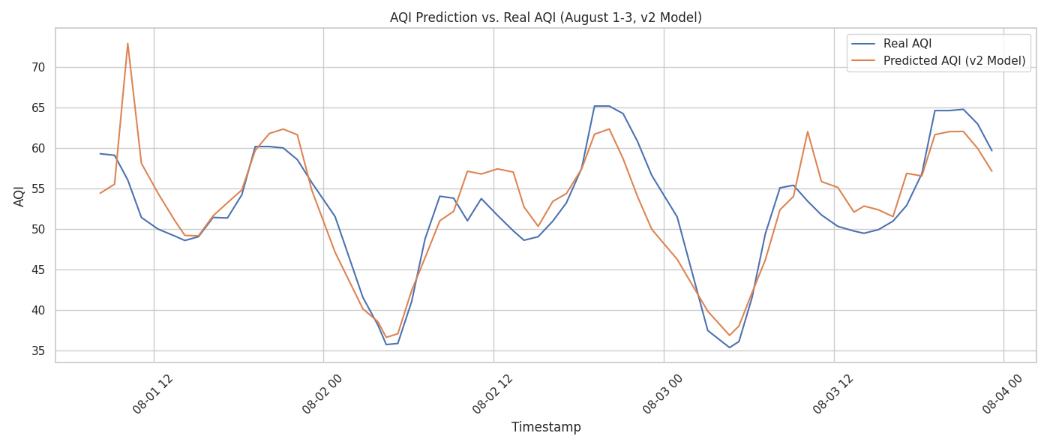
* 15. Evaluate and Compare Predictions (v3 model on 4-6 Aug vs. v2 model on 4-6 Aug):
* The predictions from the v3 model on the August 4-6 data (`preds\_4\_6\_v3`) are evaluated by comparing them to the real AQI values. A DataFrame (`predictions\_aug4\_6\_v3`) is created. Performance metrics (RMSE, MAE, R²) are calculated and printed for these predictions. This step then performs a direct comparison of these metrics (from the v3 model on 4-6 Aug) with the stored metrics from the v2 model's predictions on the \*same\* August 4-6 period (`v2\_4\_6\_metrics`). This comparison clearly quantifies the improvement achieved by using the v3 training data. A comparison table is displayed, and a conclusion is drawn about the performance improvement.

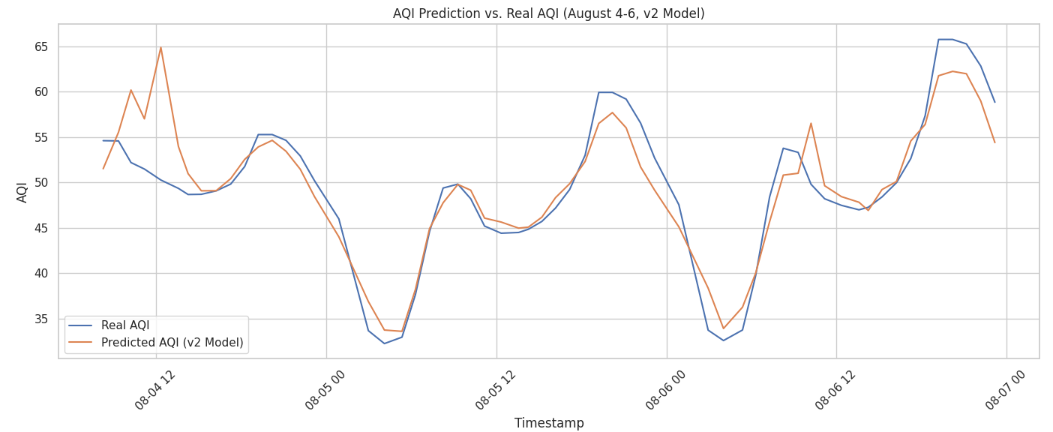
Screenshot

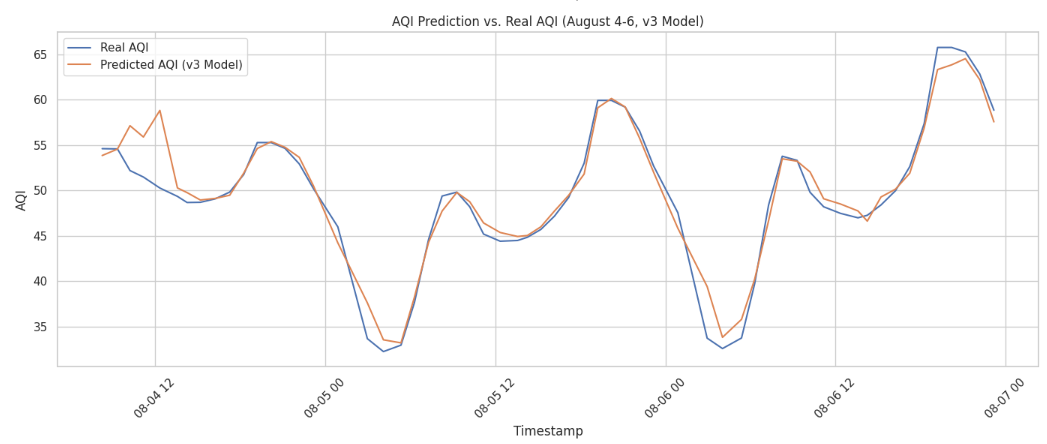


* 16. Visualize Predictions:
* Line plots are generated to visually compare the predicted AQI values with the real AQI values. Plots are created for:
* \* August 1-3 predictions using the v2 model.
* \* August 4-6 predictions using the v2 model.
* \* August 4-6 predictions using the v3 model.
* These visualizations allow for an intuitive understanding of how well each model tracks the real AQI and visually demonstrate the improved fit of the v3 model predictions for August 4-6.

Screenshot



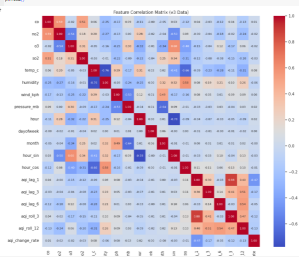




17. Correlation Matrix (v3):

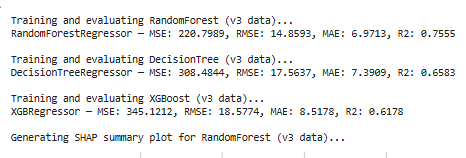
* A heatmap is generated to visualize the correlation matrix of the features in the feature-engineered v3 training data (`df\_fe\_v3[feature\_cols]`). This helps in understanding the relationships between different features and can provide insights into which features might be highly correlated with the target AQI.

Screenshot



* 18. SHAP Feature Importance (v3):
* SHAP values are calculated for the best-performing model trained on the v3 data (RandomForest). A SHAP summary plot (specifically a bar plot showing mean absolute SHAP values) is generated to show the average impact of each feature on the model's output magnitude. This provides a quantitative ranking of feature importance based on the model trained with the most recent data.

Screenshot





* 19. Save Outputs:
* The dataframes containing the predicted and real AQI values are saved as CSV files:
* \* `Predictions\_Aug1\_3\_v2\_model.csv` (Predictions from v2 model on 1-3 Aug)
* \* `Predictions\_Aug4\_6\_v2\_model.csv` (Predictions from v2 model on 4-6 Aug)
* \* `Predictions\_Aug4\_6\_v3\_model.csv` (Predictions from v3 model on 4-6 Aug)
* These files provide a record of the predictions for reporting or further analysis.

Screenshot



* 20. Save Feature Engineered Data:
* The feature-engineered dataframes created throughout the pipeline are saved as CSV files:
* \* `df\_fe\_v2.csv` (Feature-engineered v2 training data)
* \* `df\_fe\_v3.csv` (Feature-engineered v3 training data)
* \* `df\_1\_3\_fe\_v2.csv` (Feature-engineered 1-3 Aug pollutant data, processed with v2 pipeline)
* \* `df\_4\_6\_fe\_v3.csv` (Feature-engineered 4-6 Aug pollutant data, processed with v3 pipeline)
* Saving these allows you to download and inspect the transformed data used at various stages of the pipeline.

Screenshot

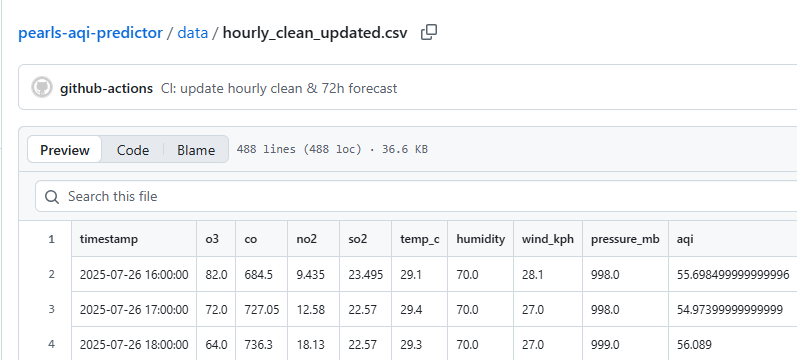


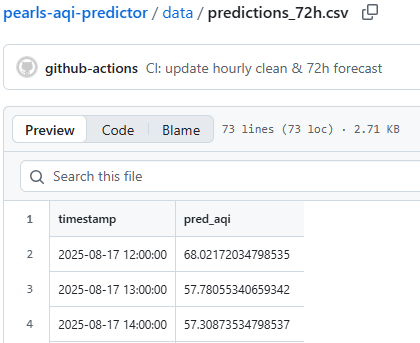
# 🔎 Step 6: CI/CD Automation (GitHub Actions)

I created predict\_live.yml to automate the pipeline. Every hour:  
1. Fetch pollutants → save to hourly\_clean\_updated.csv  
2. Run predict\_live.py → forecast 72 hours  
3. Save predictions in predictions\_72h.csv  
4. Push updates back to GitHub repo  
  
Earlier, my pipeline broke due to merge conflicts, but I fixed it and resumed from 26 July. I also made another yml file but its not in used so I named it as .disabled.yml so it will not disturb my environment.

Screenshot

# 



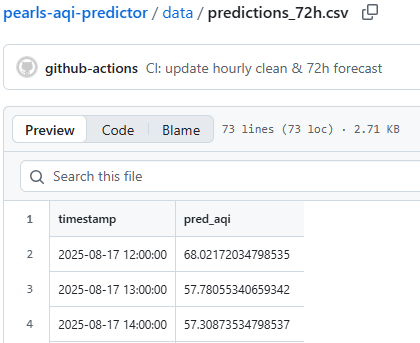


# 🔎

# Step 8: Autoregressive Forecasting (72h)

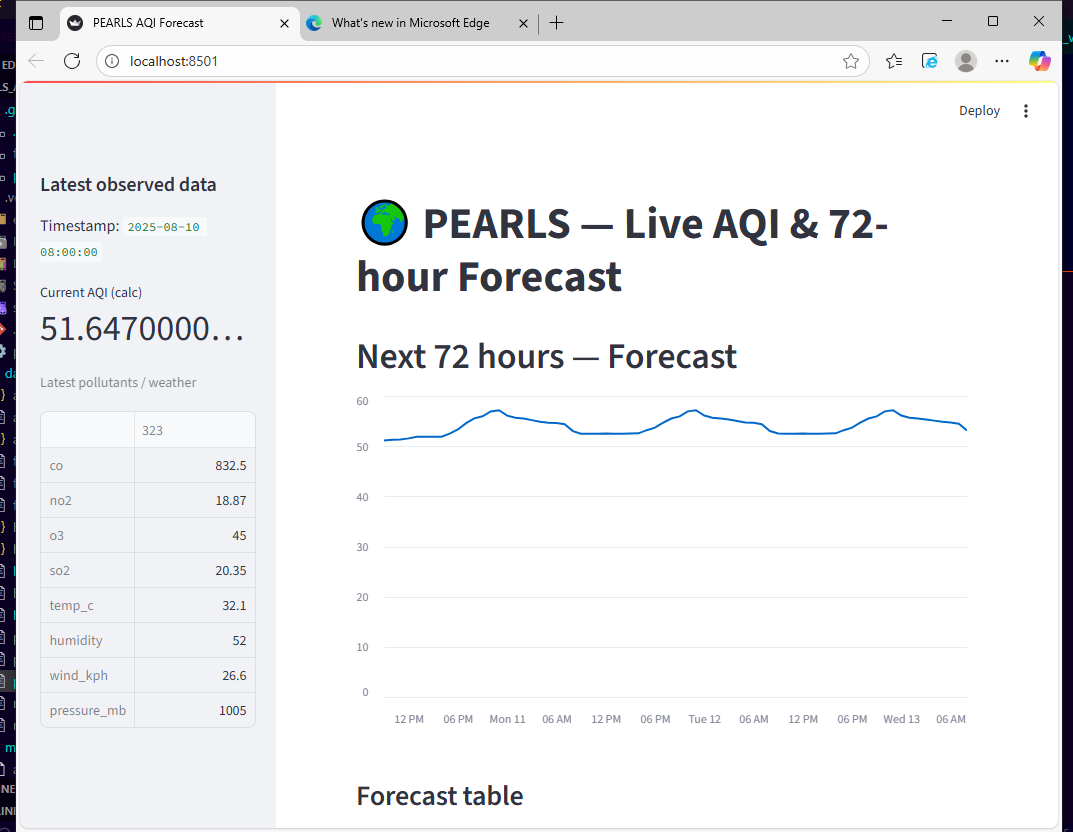
The forecast works in an autoregressive loop:  
- Predict AQI for the next hour  
- Feed that prediction back into the dataset  
- Use it to predict the following hour  
- Repeat until 72 hours are covered  
  
This creates a realistic 3-day AQI forecast.

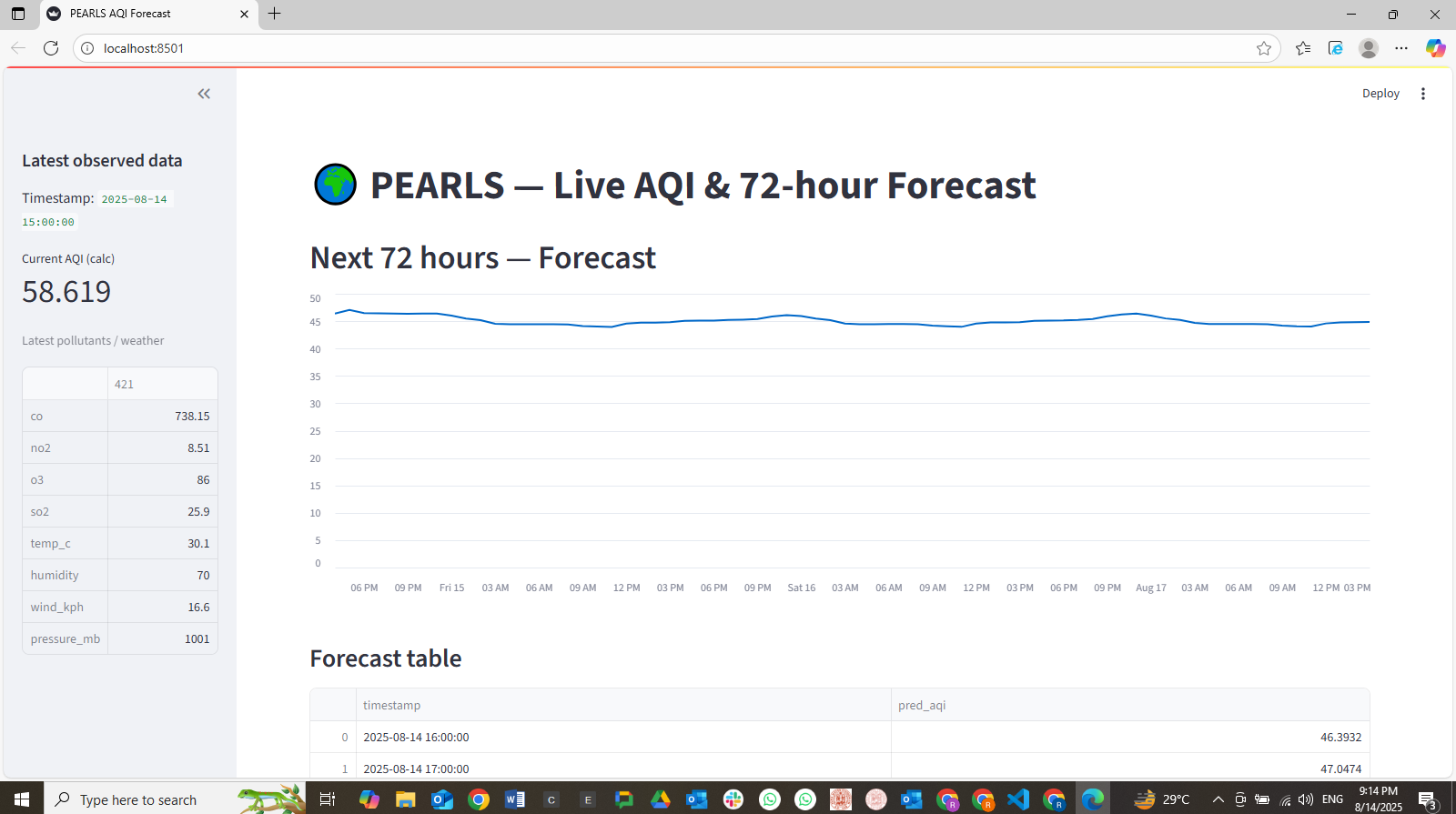
Screenshot:

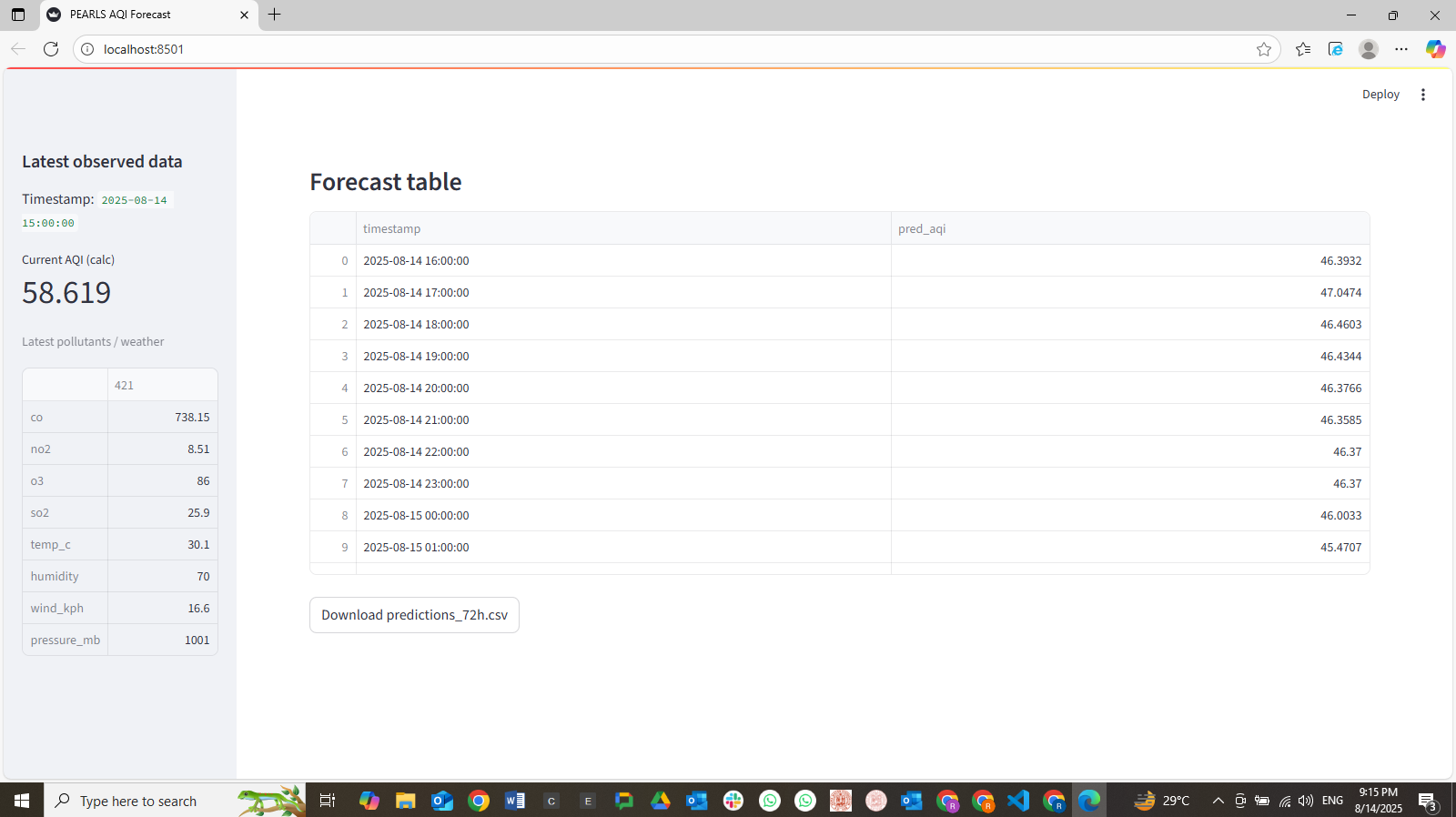


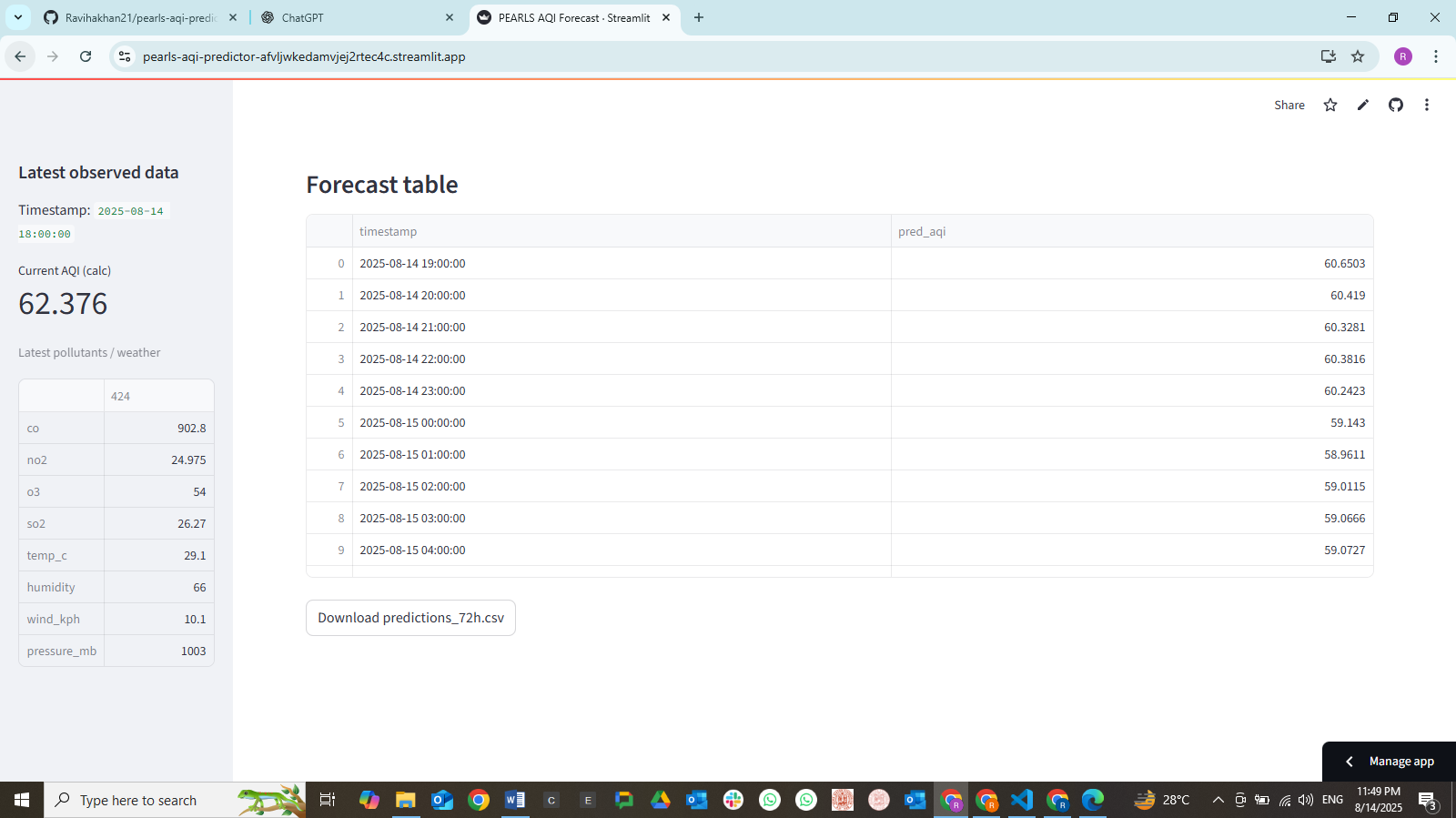
# 🔎 Step 9: Streamlit Dashboard

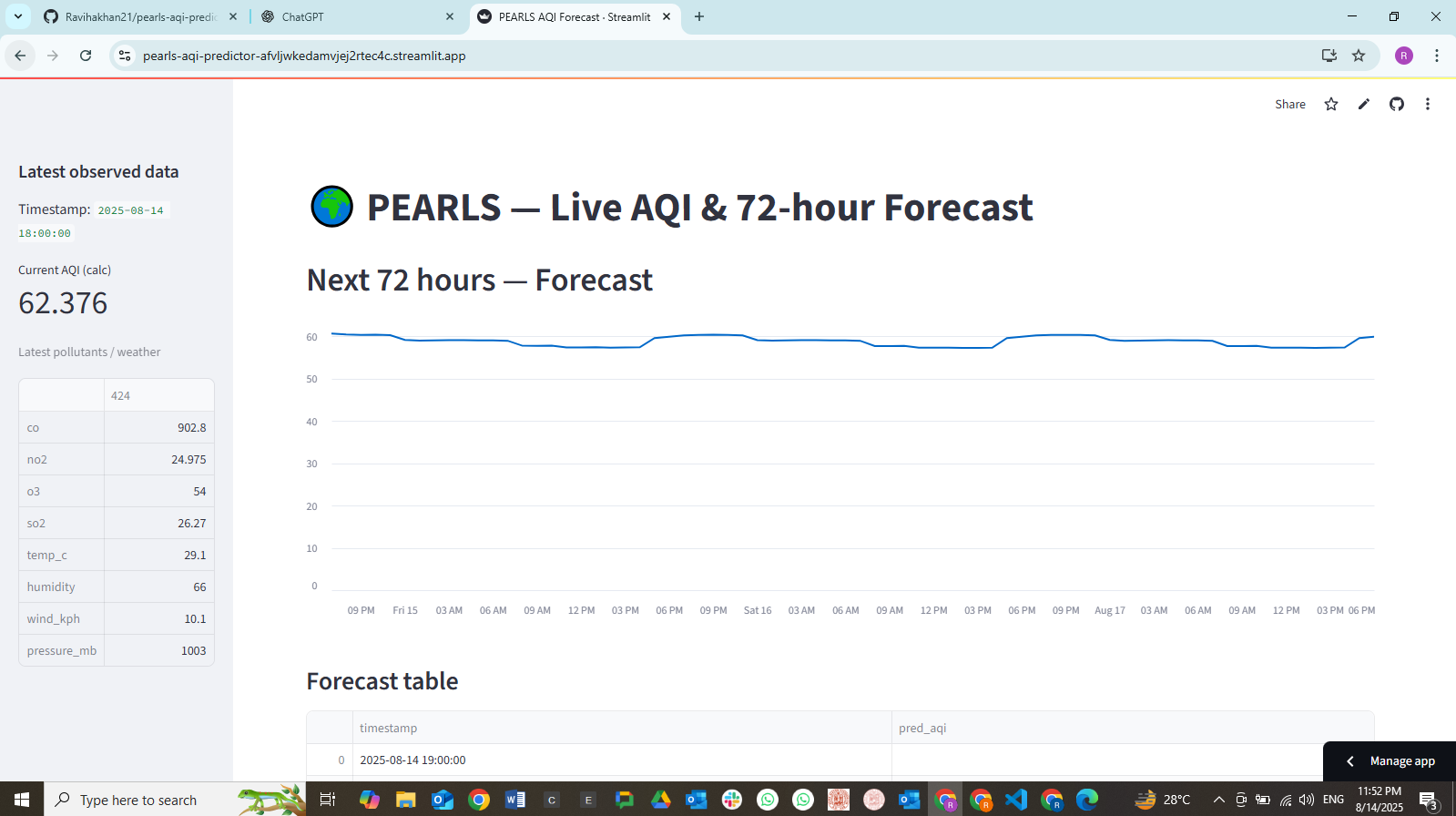
I built app.py in Streamlit to display:  
- Sidebar: latest AQI snapshot + pollutant details  
- Main: 72-hour forecast line chart + table  
- Option to download predictions as CSV  
  
This dashboard is the final user-facing product.

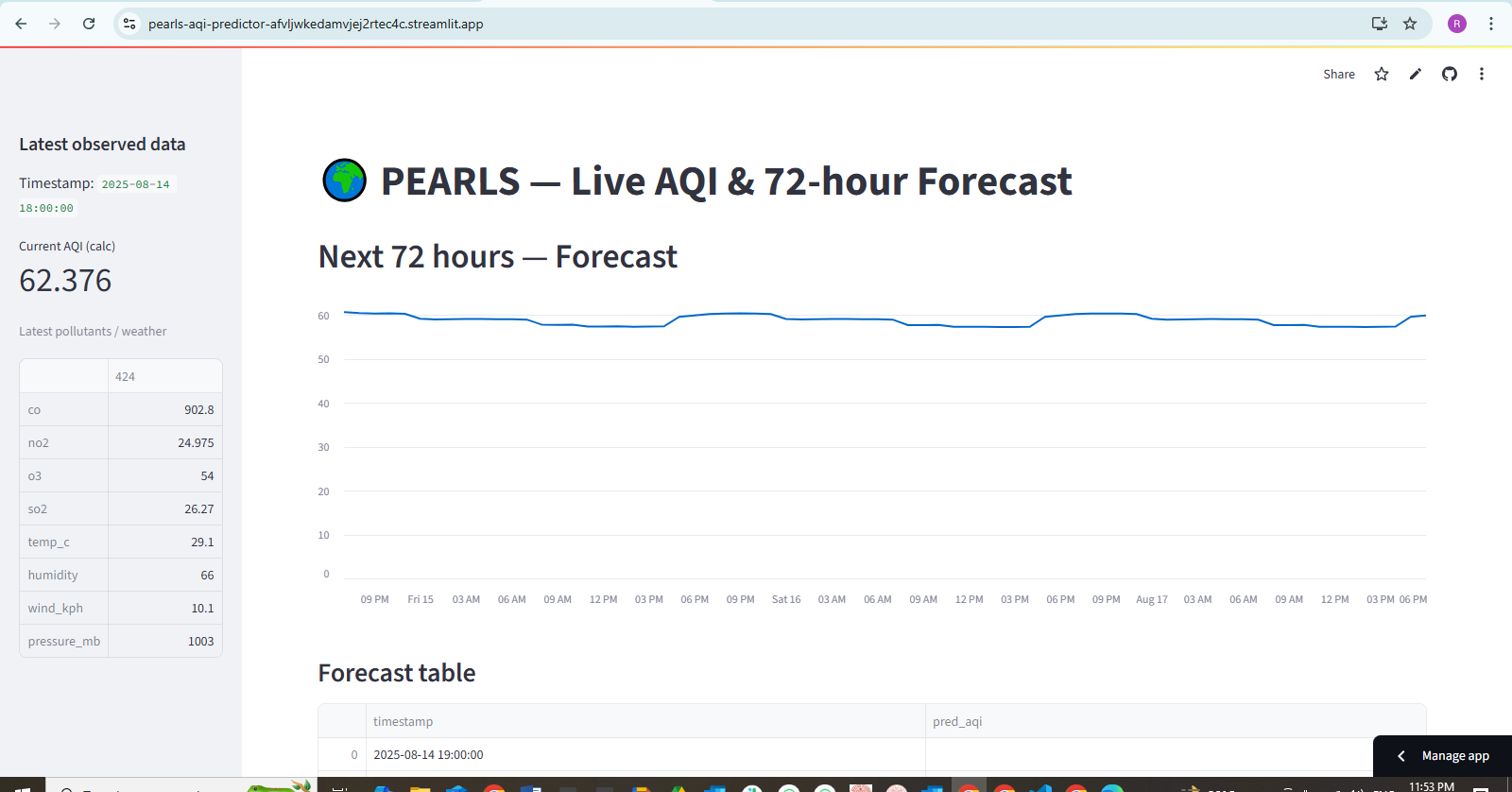


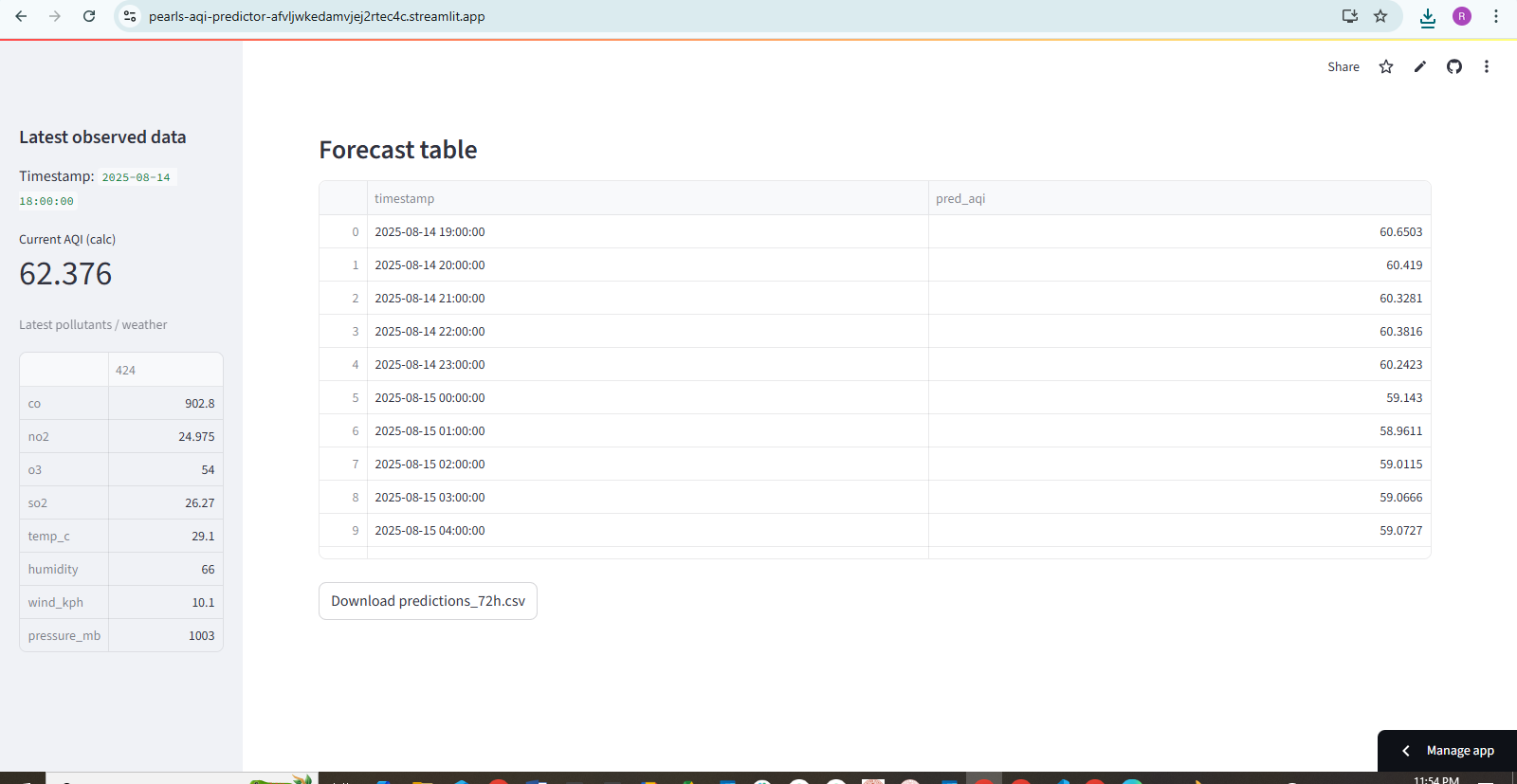


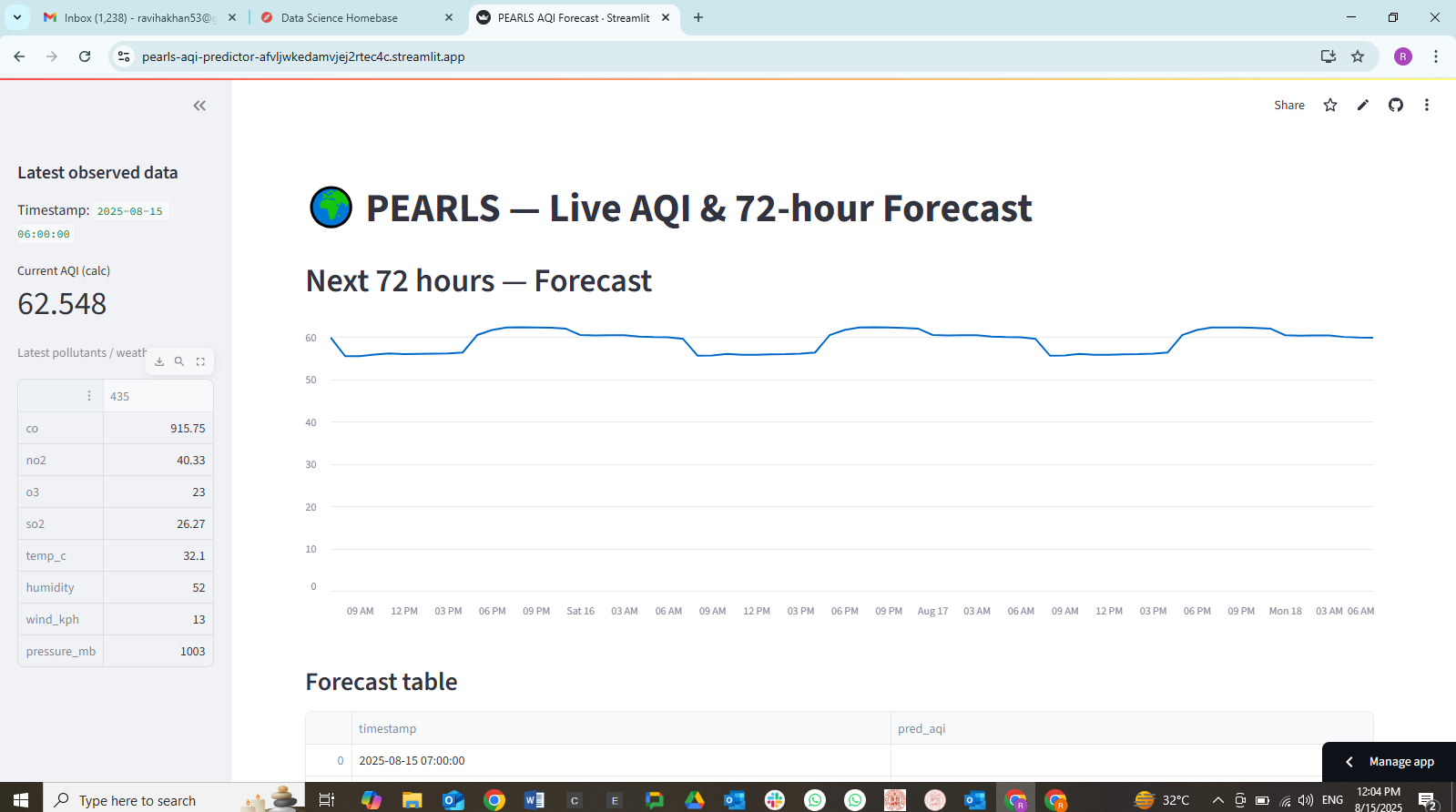


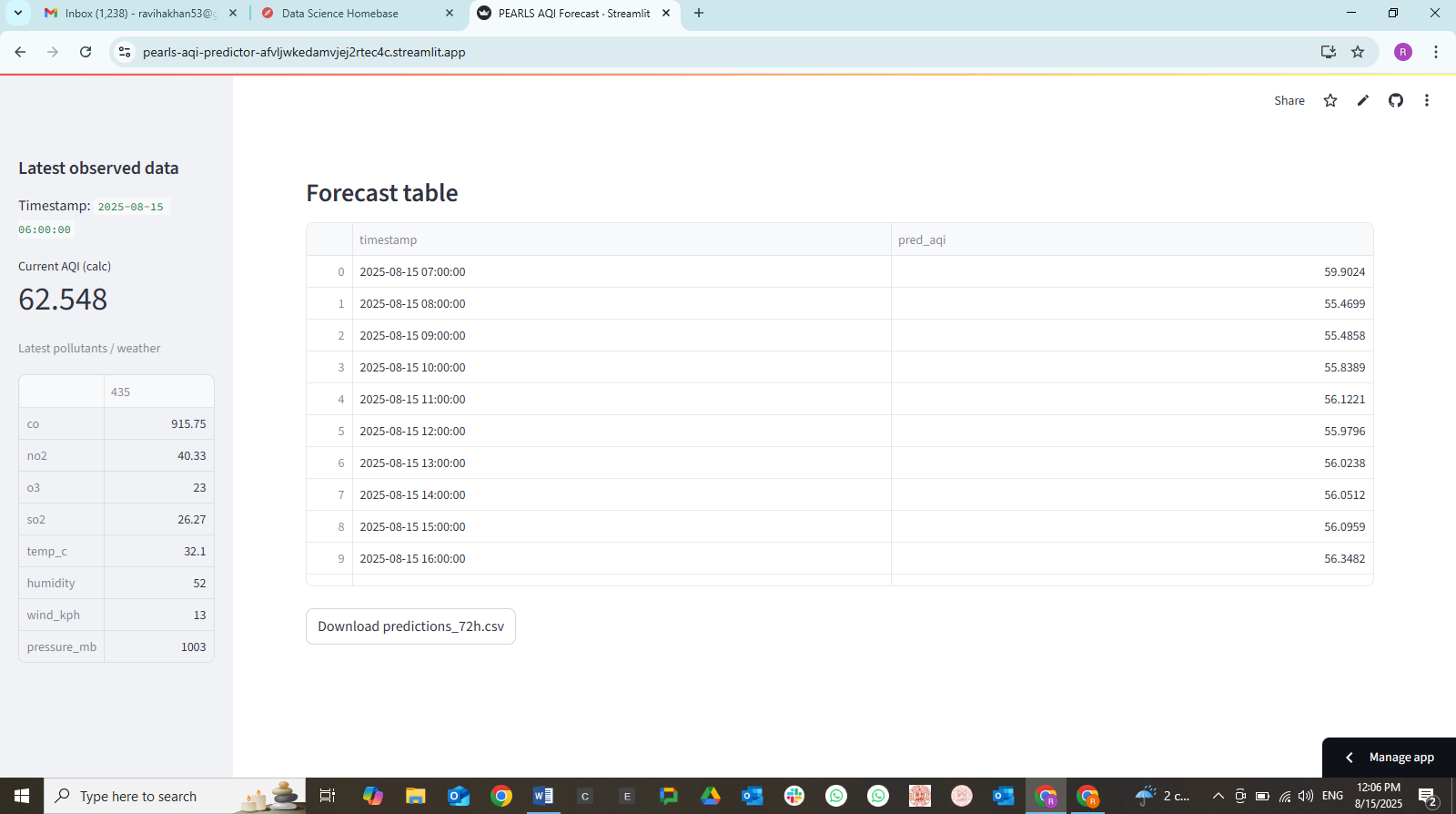


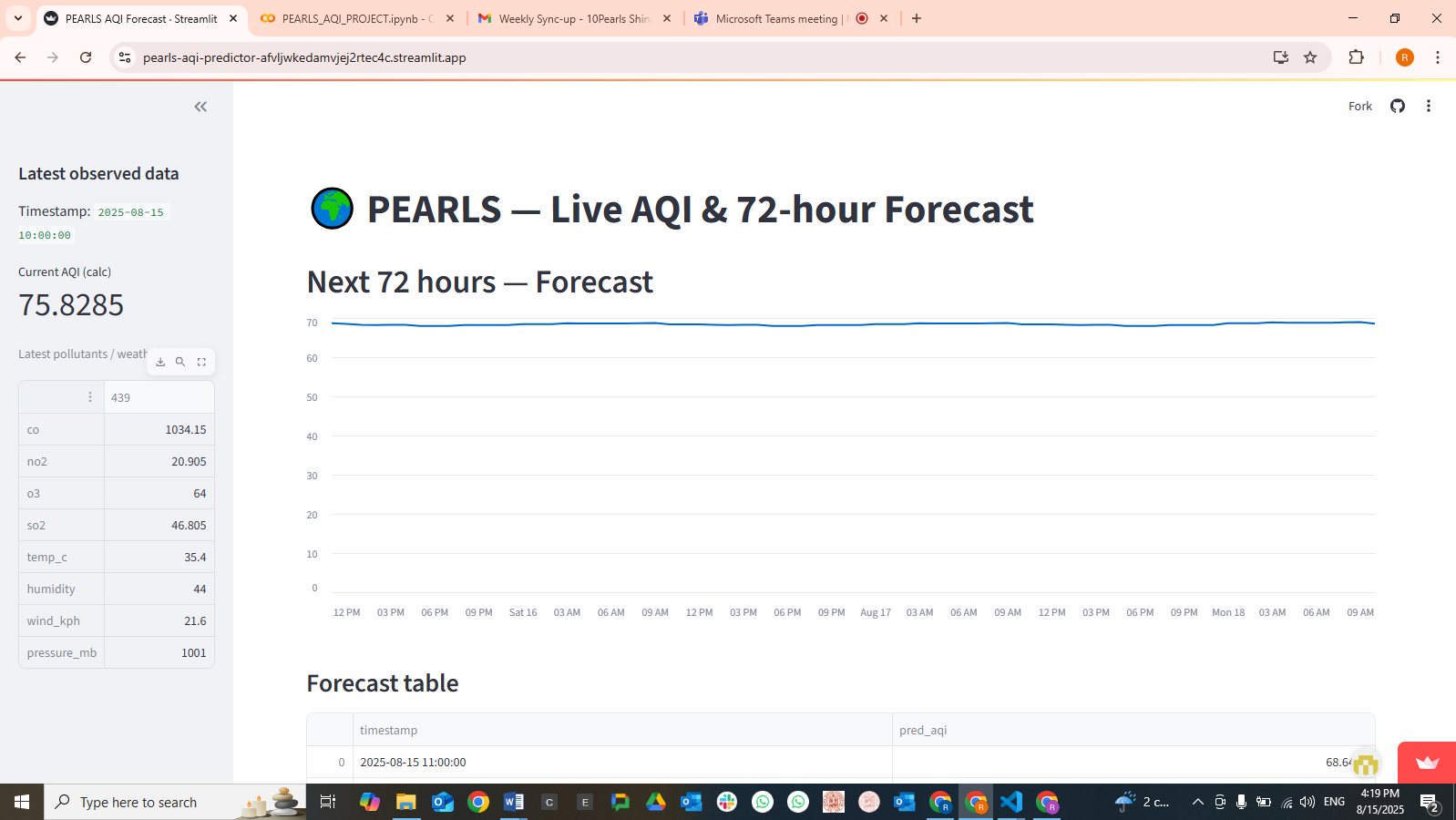


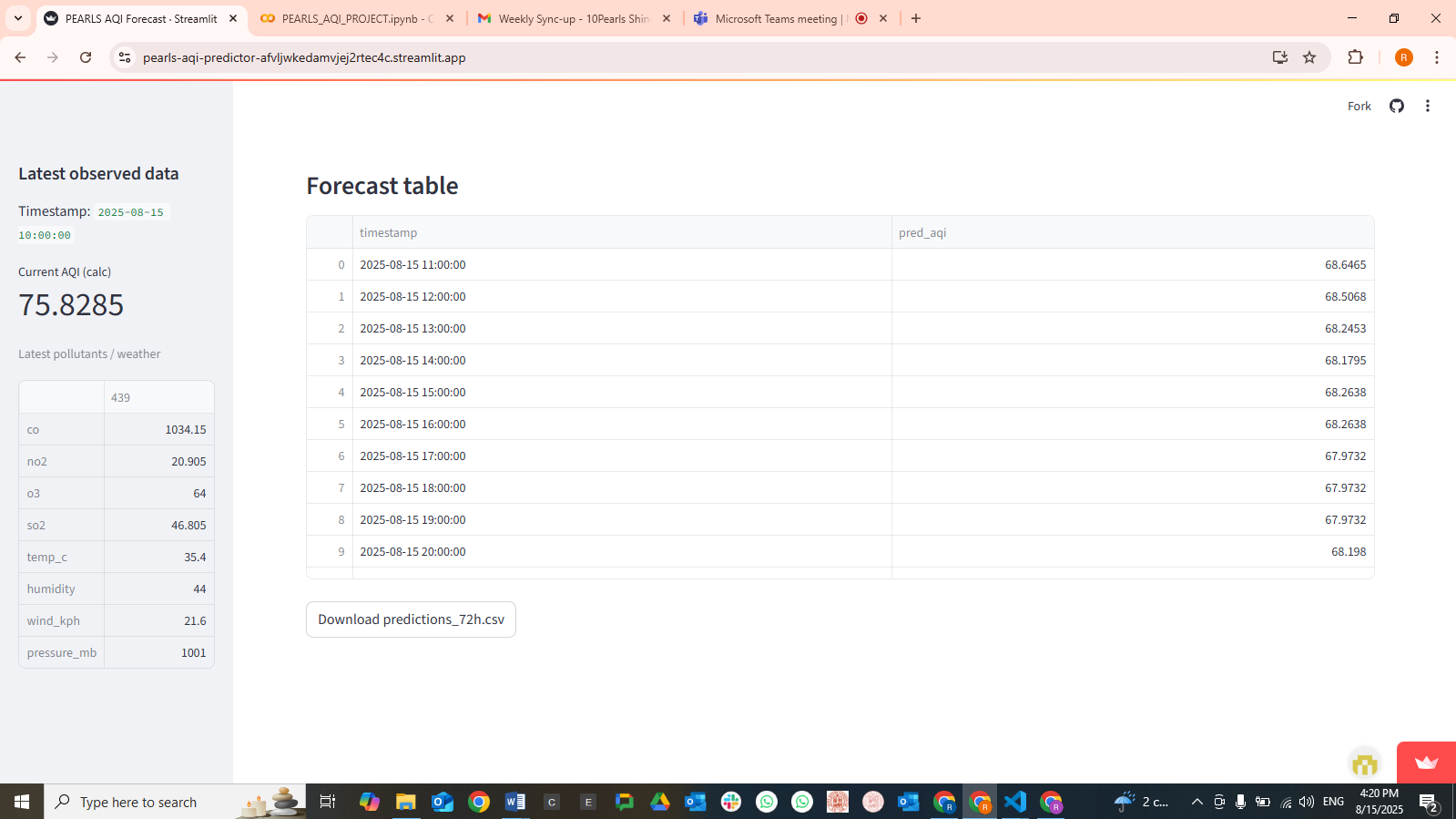




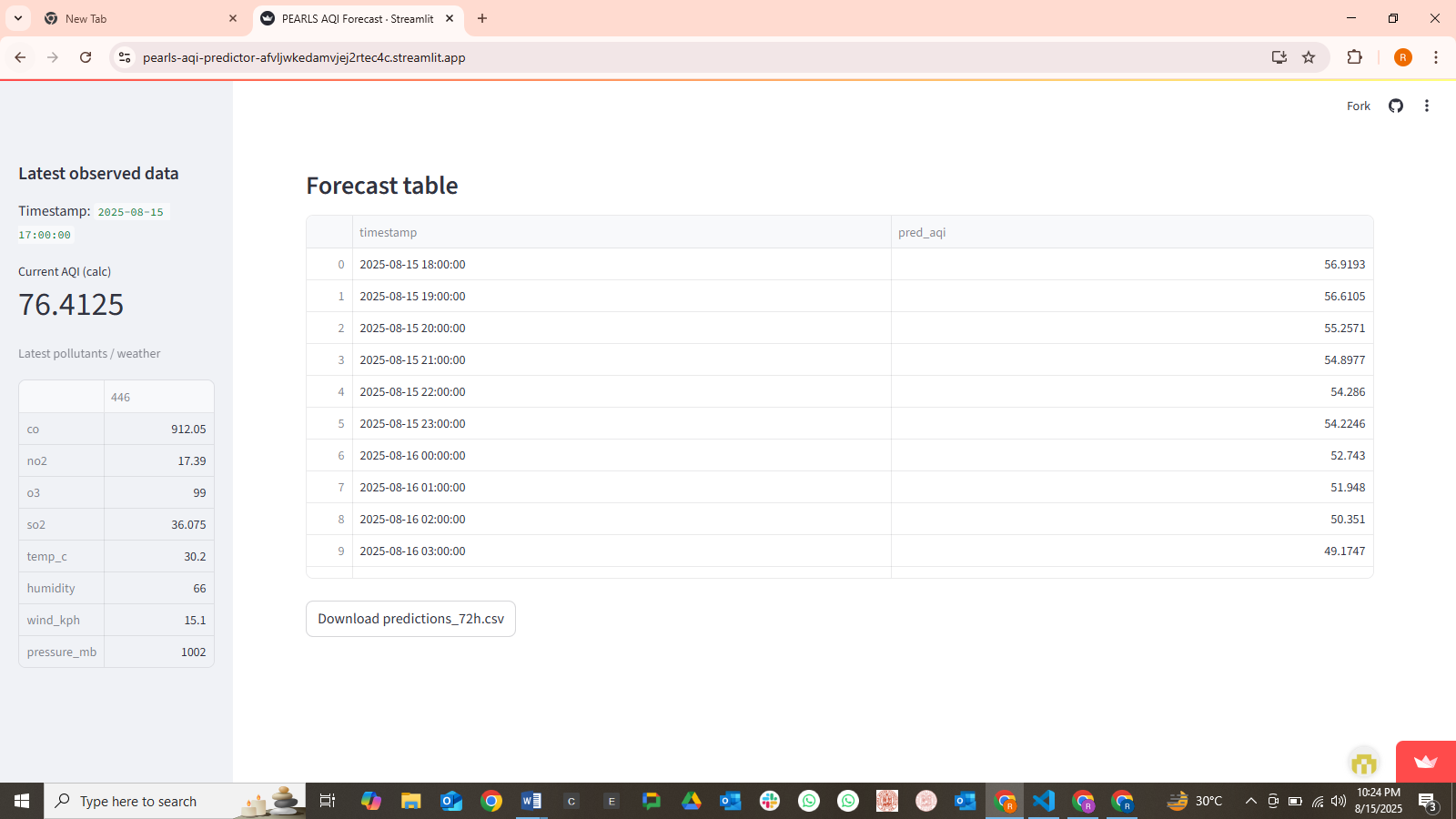


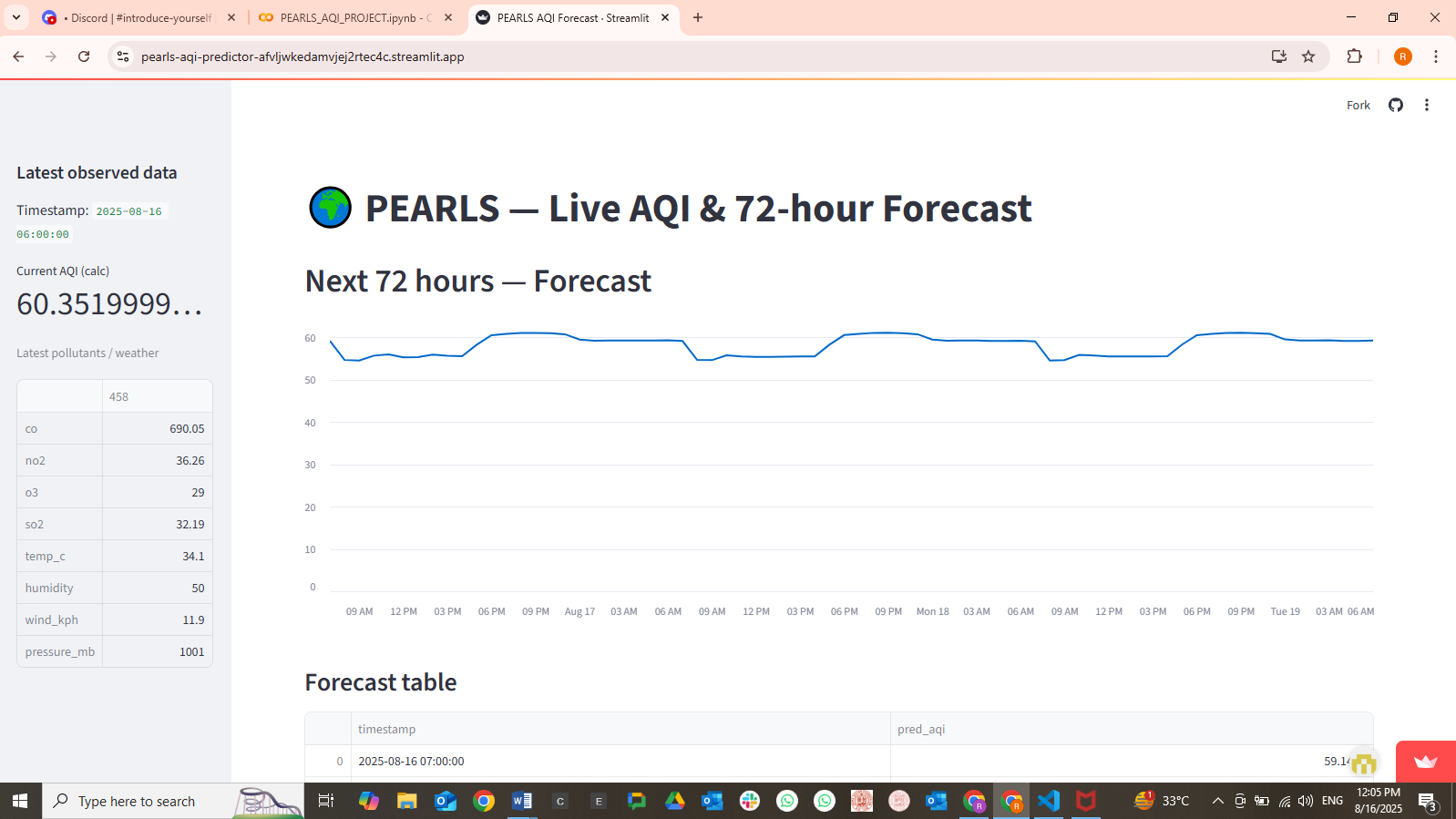


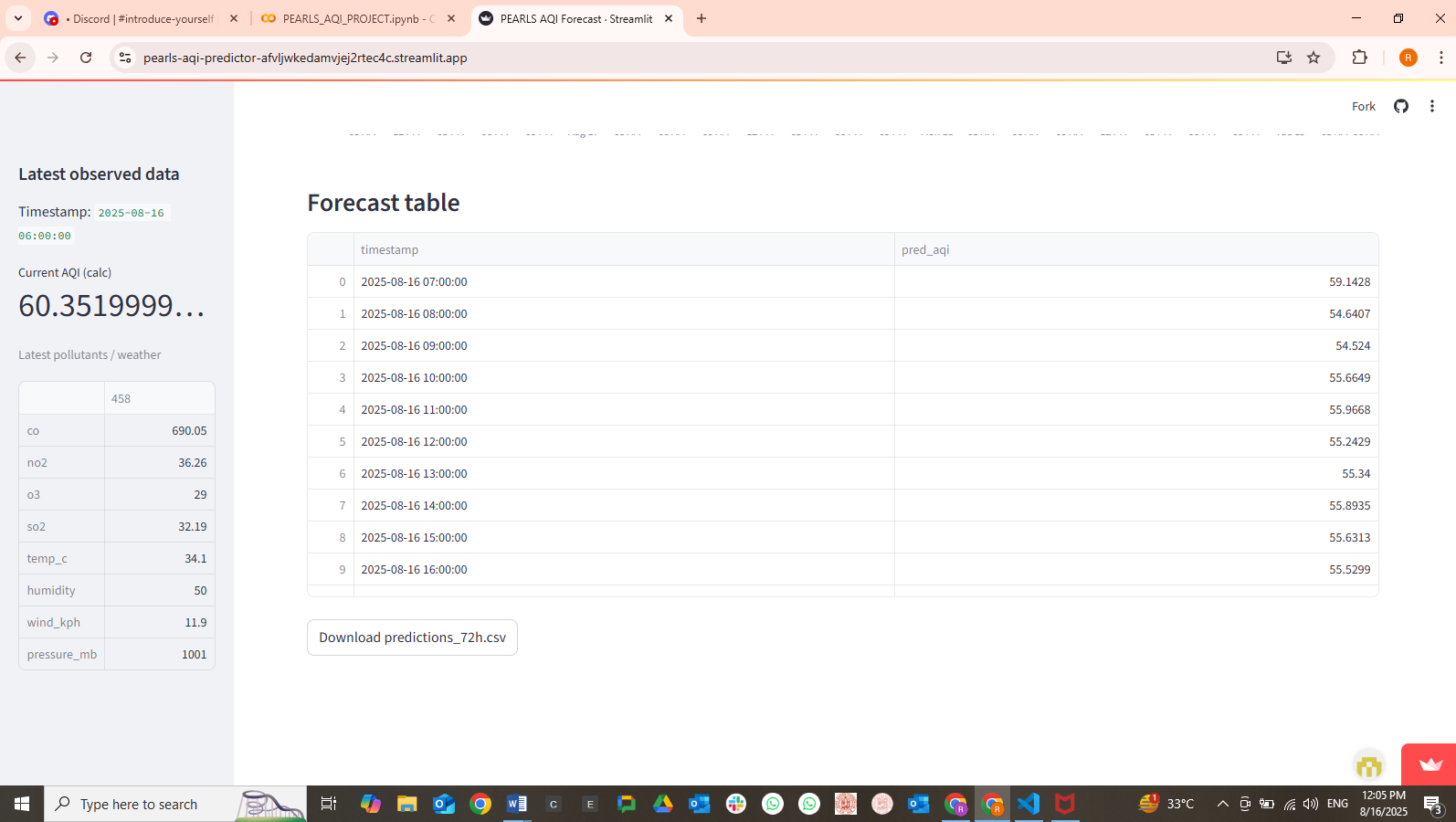


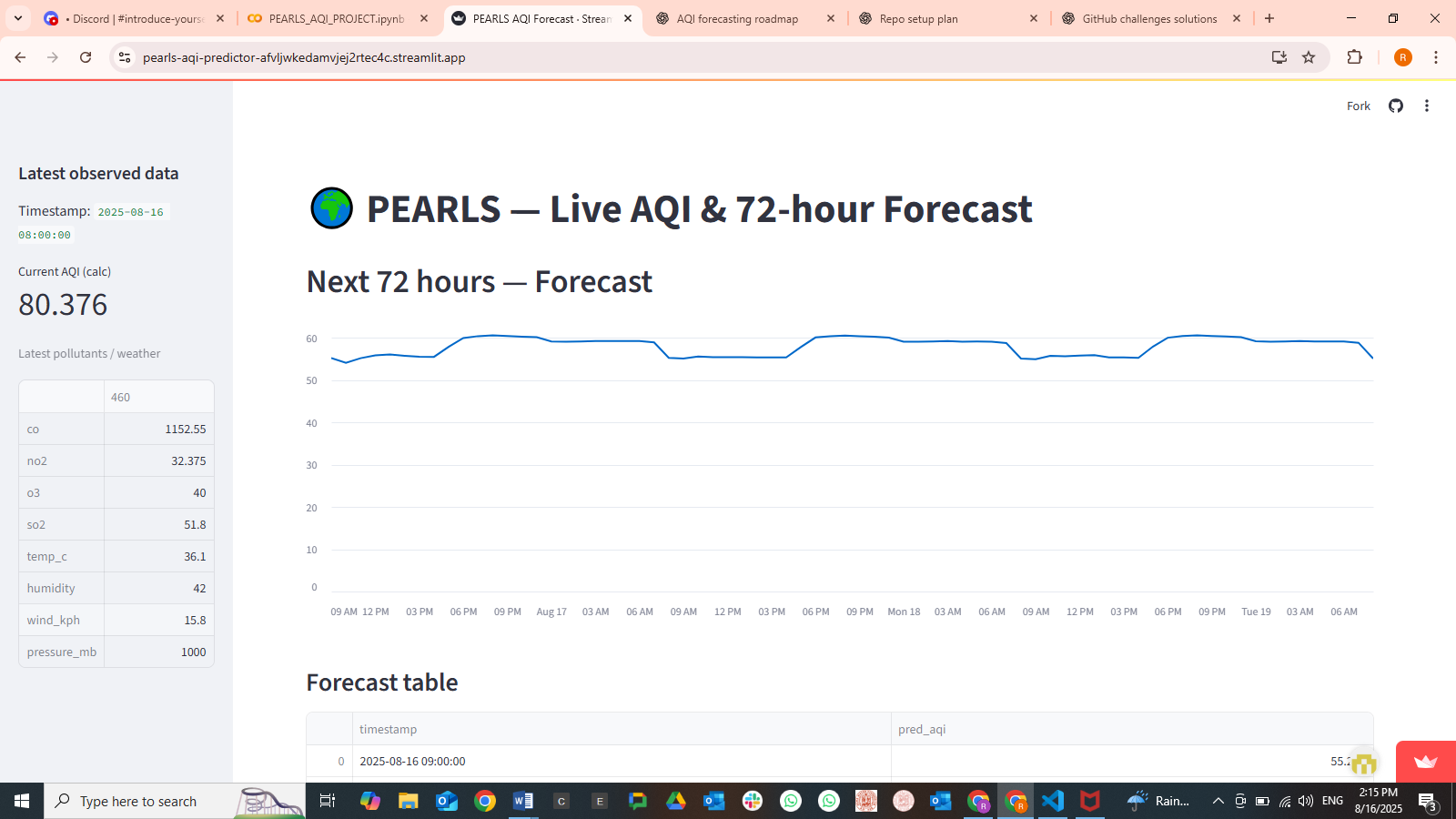


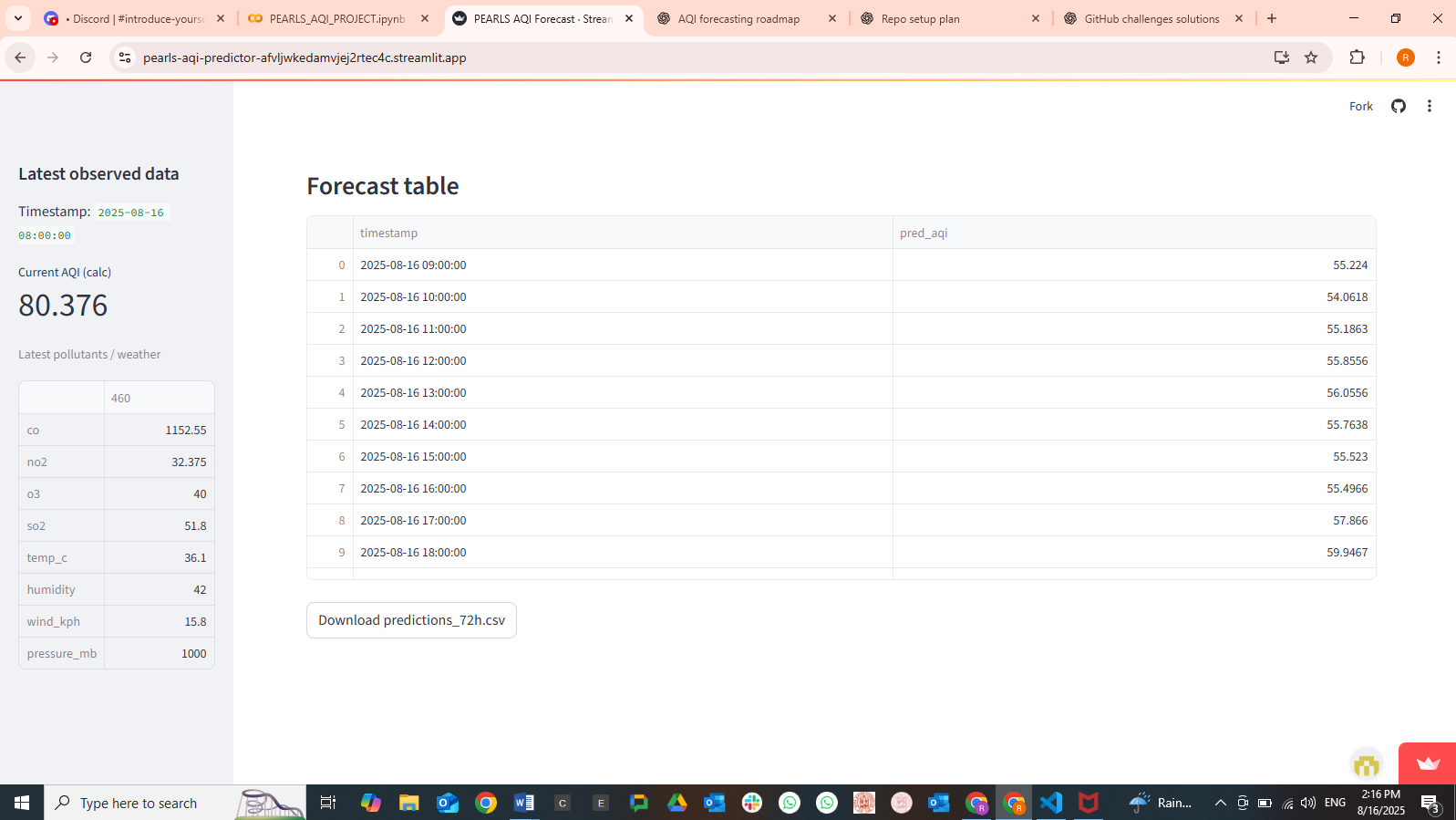


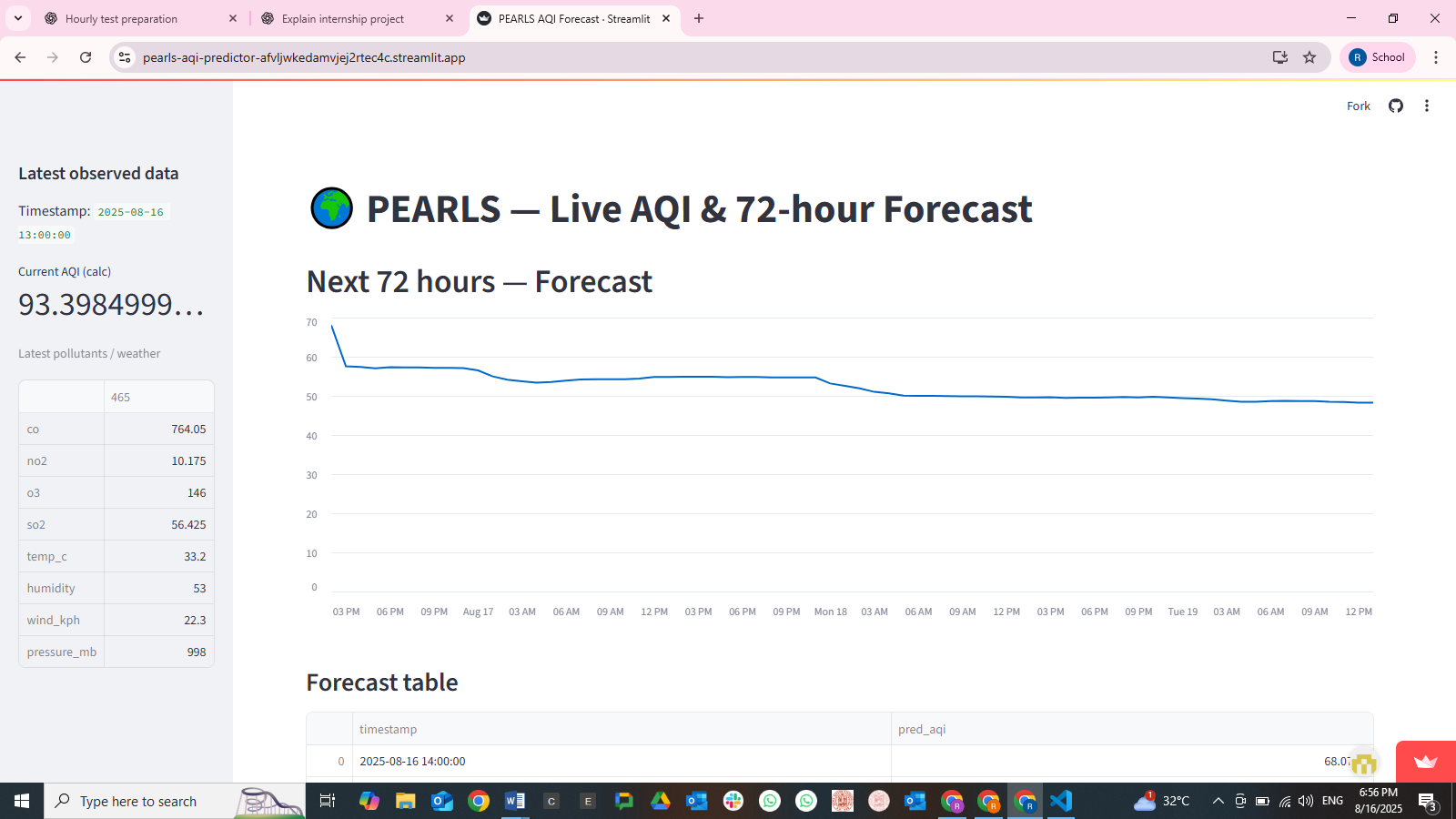


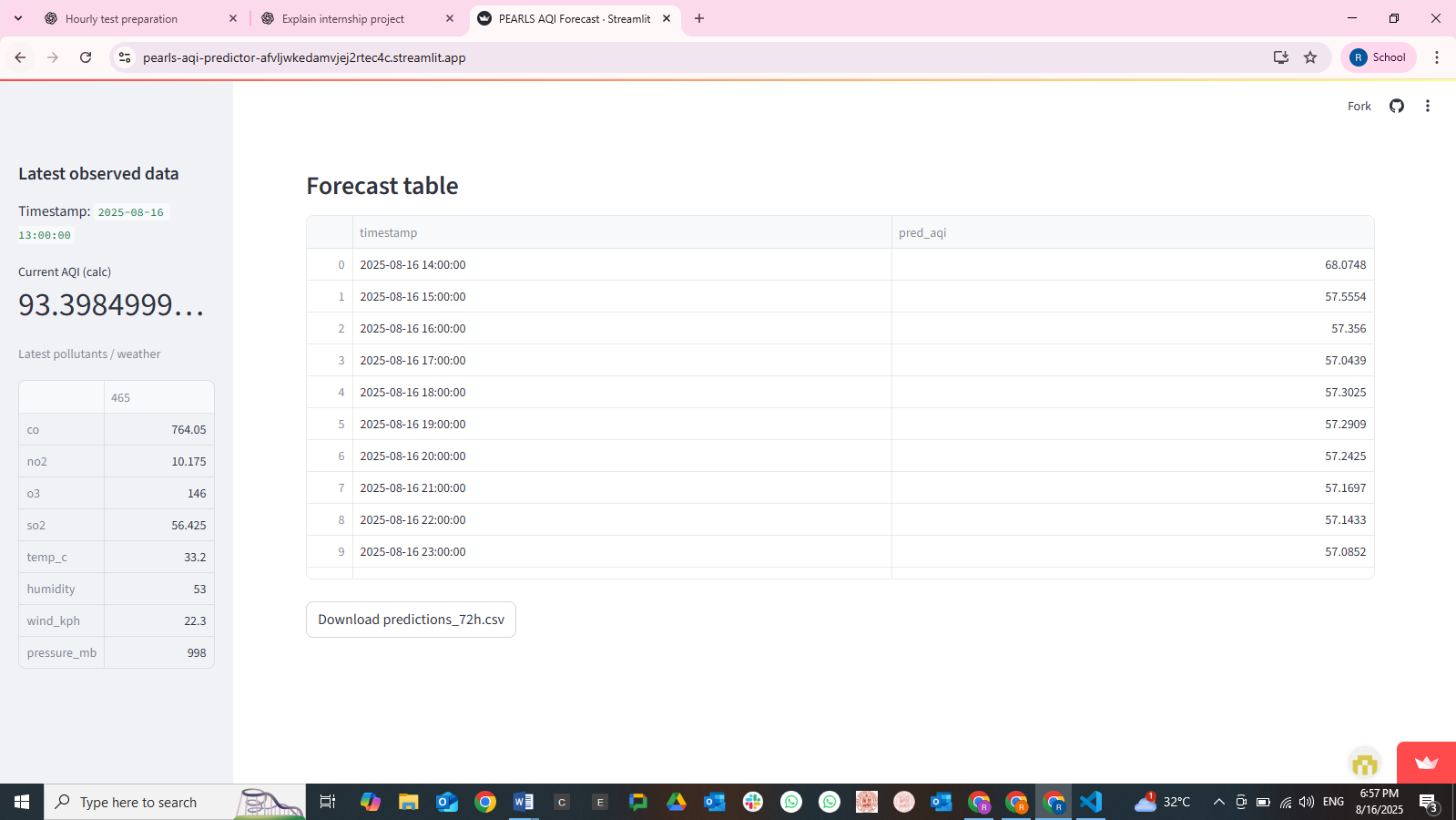


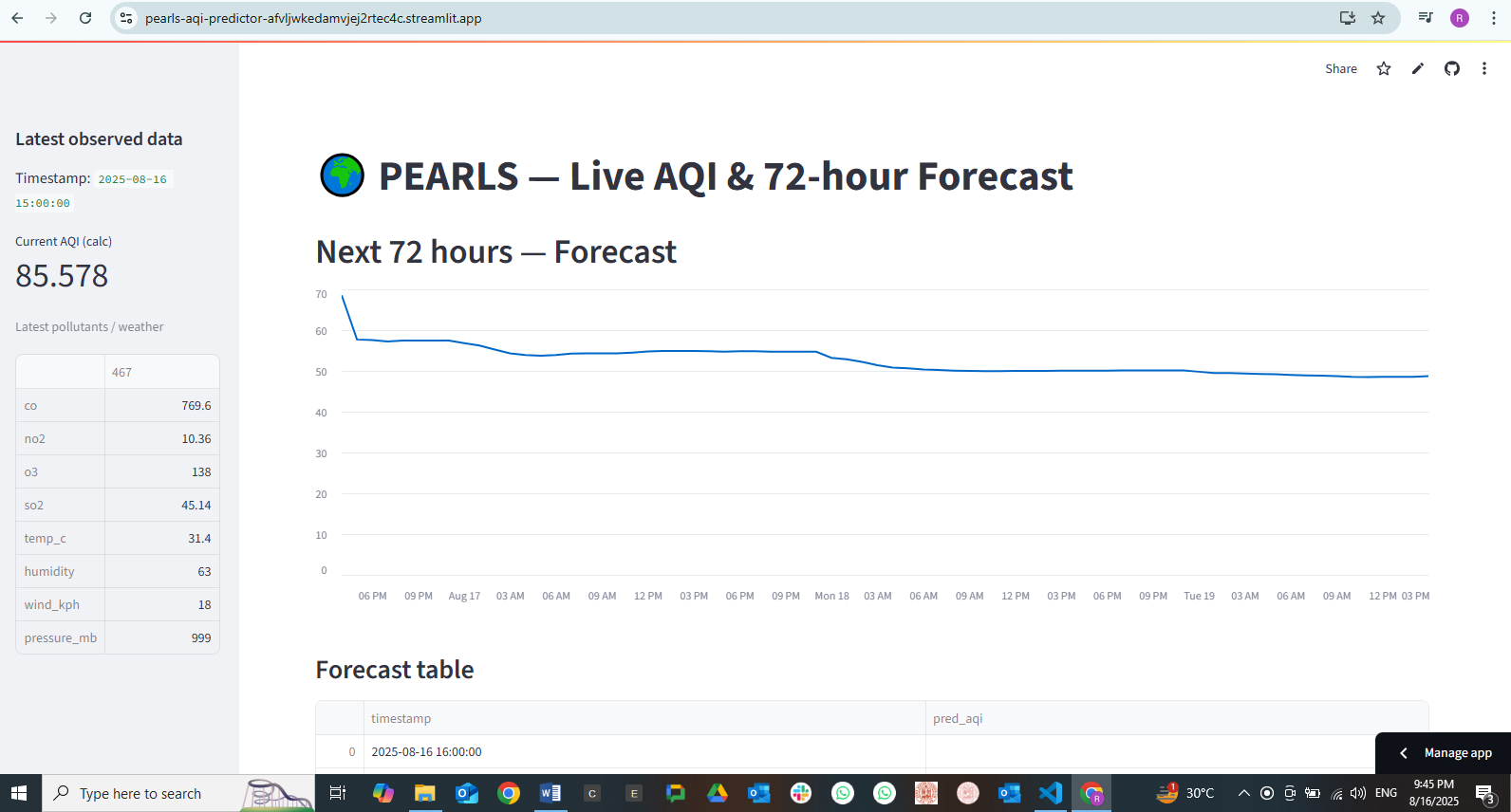


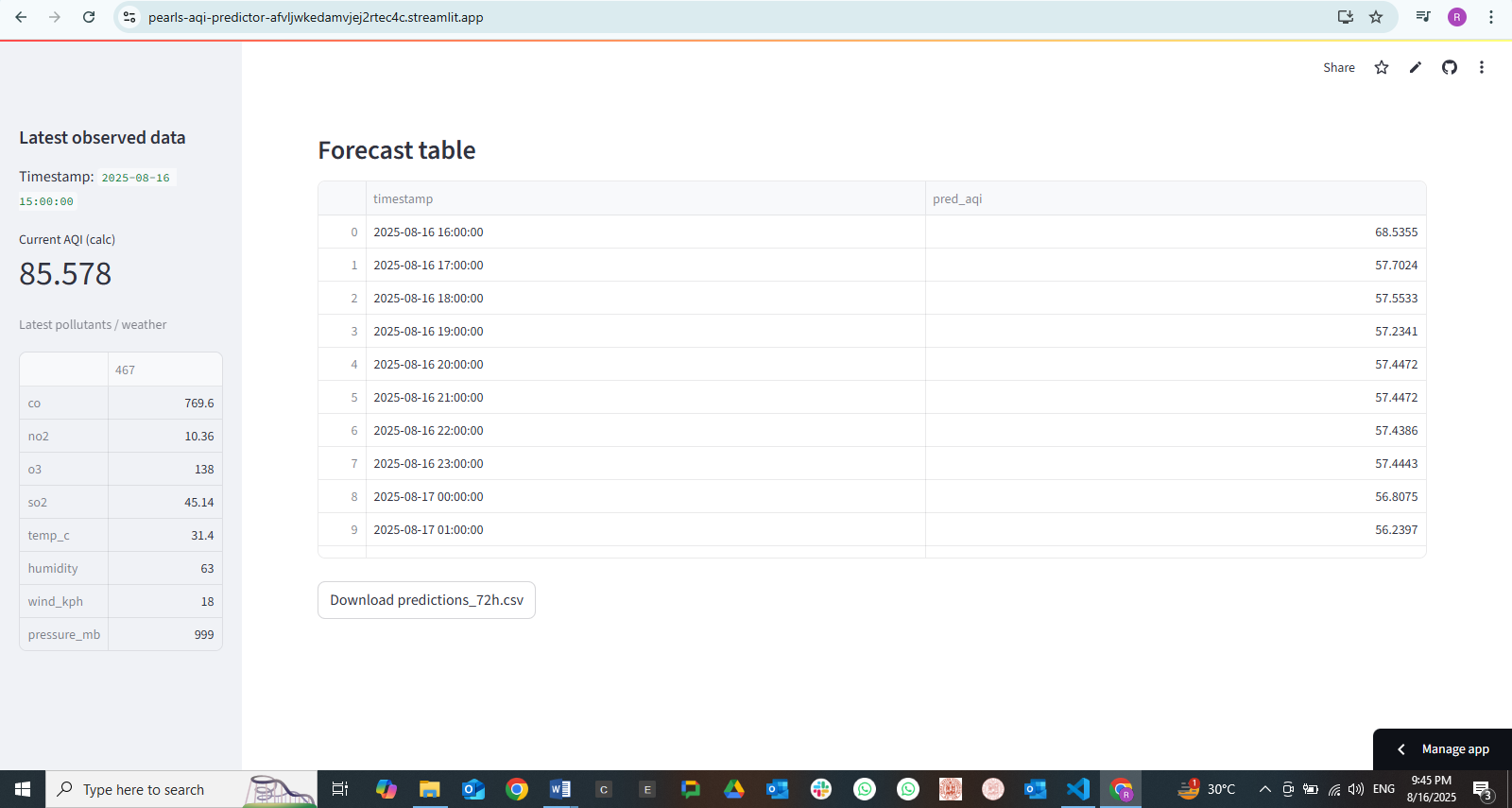




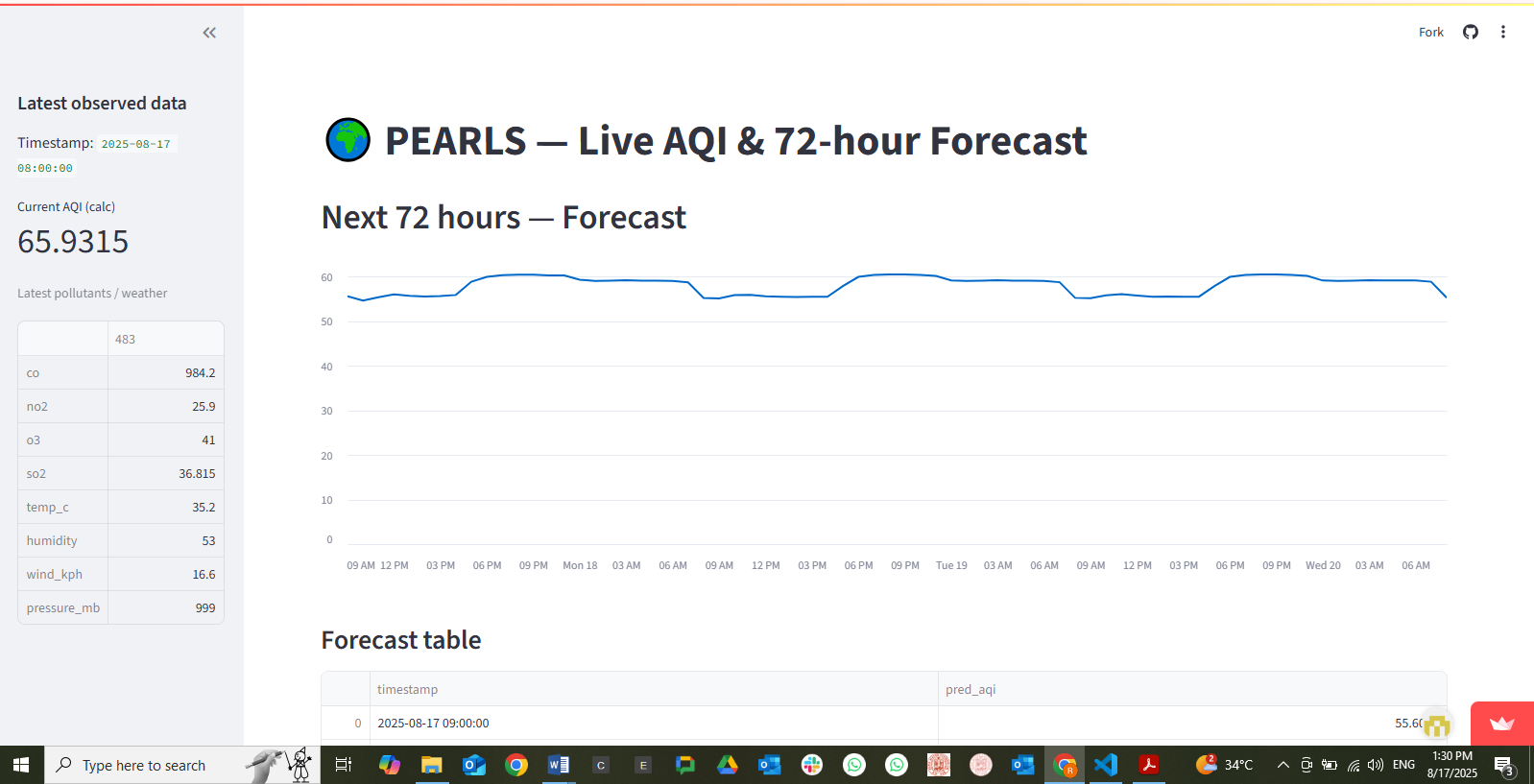


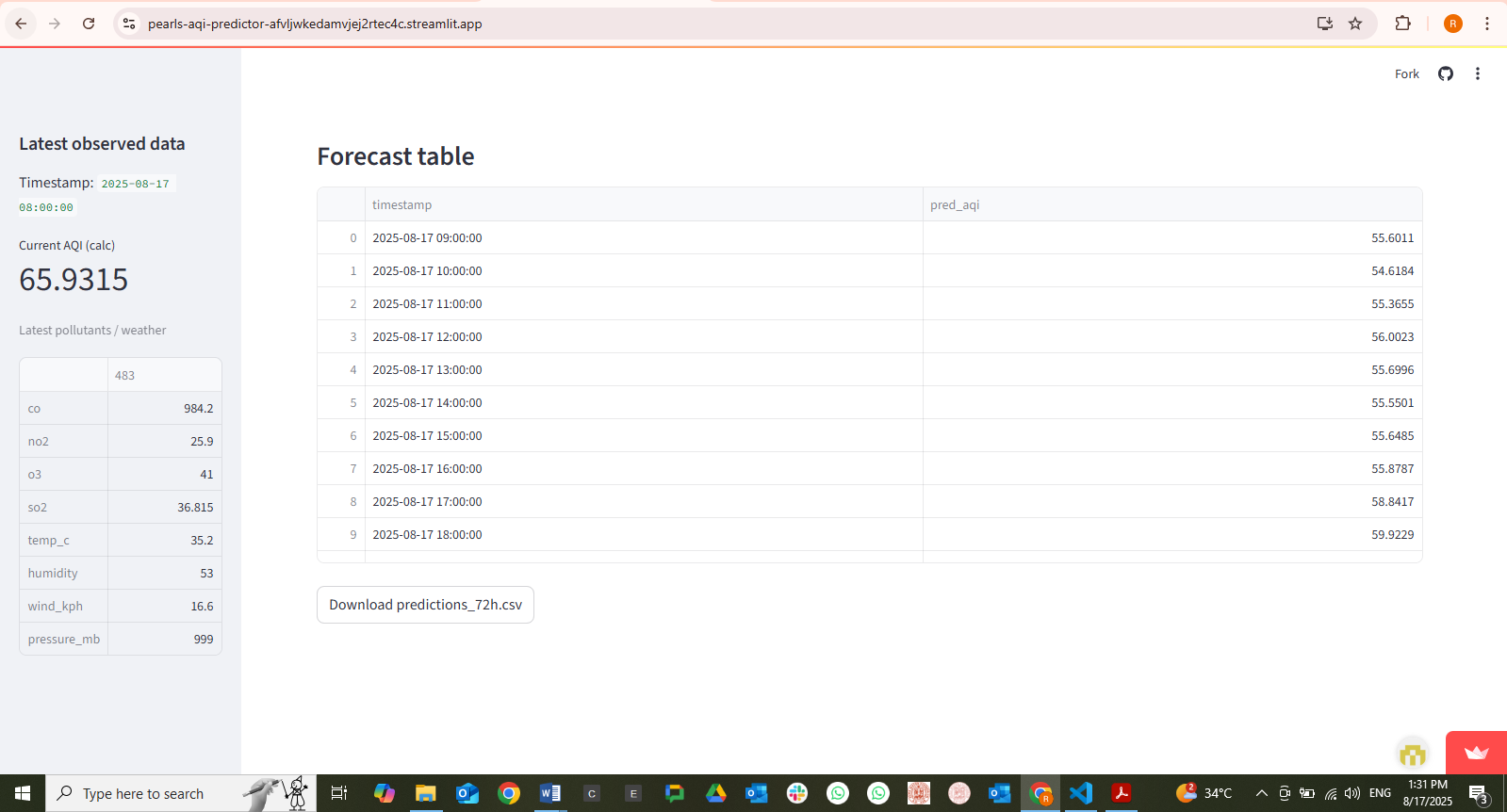


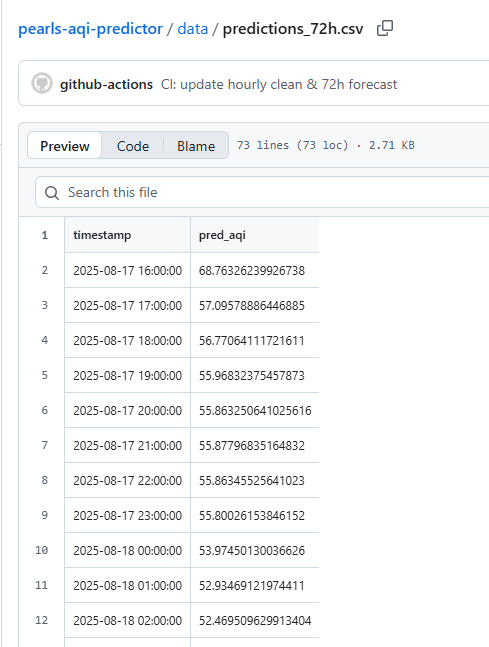




17 august 2025







# 📂 Scripts Used

• fetch\_aqi\_data.py — Hourly pollutant fetch  
• fetch\_historic\_data.py — Historic pollutant + weather fetch  
• clean\_merge\_data.py — Merging and AQI calculation  
• feature\_engineering.py — Feature pipeline  
• predict\_live.py — Autoregressive forecast  
• app.py — Streamlit dashboard  
• predict\_live.yml — GitHub Actions CI/CD  
  
👉 (Here you can paste key script code OR submit them separately as a zip.)

# Final Outcomes

• Collected and merged historic + live data  
• Handled missing values and gaps (21–26 July)  
• Implemented EPA AQI calculation  
• Built a feature engineering pipeline for time-series AQI prediction  
• Trained models → chose Random Forest  
• Automated serverless forecasting with GitHub Actions  
• Created a Streamlit dashboard for live + forecast AQI

This was my very first time working on a complete **data science project** from start to finish. At the beginning, I had very little experience beyond my AI lab course, but this project pushed me into **real-world workflows** like data collection, feature engineering, model training, and deployment. Along the way, I also learned how to use **GitHub CI/CD pipelines** and set up a **serverless automation system** for hourly AQI prediction. It was a journey full of trial and error, but also full of growth. In the end, I successfully built a **working AQI prediction system**, and this made the whole experience with **10Pearls Pakistan** not just a technical achievement, but also a very **memorable milestone** in my learning journey.

Dashboard link:

<https://pearls-aqi-predictor-afvljwkedamvjej2rtec4c.streamlit.app/>