10 PEARLS AQI PREDICTOR PROJECT REPORT

August 2025

Deployed on:

https://aqi-predictor-self.vercel.app/

Github Link:

https://github.com/Qar-Raz/AQI-Predictor.git

By Qamar Raza, Institute Of Business Administration (IBA), Karachi

PROJECT OVERVIEW

This project details the creation of a system to predict the Air Quality Index (AQI) for the next three days in Karachi. It is built using serverless architecture, so it runs on the cloud without needing a dedicated server. The process begins by automatically collecting weather and pollution data from external sources.

This raw data is then processed to create inputs, known as features, that an AI model can use. Then EDA is carried out to extract relevant information and highlight important trends. The features are then stored and used to train various machine learning models to find the most accurate one for forecasting air quality. This entire pipeline is automated, with data being regularly collected and the model frequently retrained to ensure the predictions remain current with a CI/CD pipeline.

A web application was also created that uses the trained model and the latest data to generate predictions. These forecasts are displayed on a webpage, allowing users to easily view both real-time and future AQI.

ARCHITECTURE

1. Tech Stack Used

Data Collection	Open-Meteo API
Feature Store	CSV Files
CI/CD Pipeline	CI/CD Pipeline
Machine Learning Frameworks	Scikit-learn, TensorFlow, Xgboost, Pandas
Backend	FastAPI, Hugging Face
Frontend	Next.js
Website Hosting	Vercel

2. Architecture Overview

The architecture follows a modern data science pipeline. The process begins with data collection, where historical and real-time weather and air pollution data are fetched from the Open-Meteo API. This data is then processed and engineered into features suitable for machine learning, which are subsequently stored in CSV files that act as a lightweight feature store. CSV files were selected over a traditional database for their simplicity and efficiency within the project's scope. This choice offered the convenience of hosting the data directly within the GitHub repository, which streamlined version control and access. Given the small size of the dataset the use of a database would have introduced unnecessary architectural complexity and the overhead of additional API calls.

These features are used to train and evaluate a variety of machine learning models developed using Scikit-learn and TensorFlow. The best-performing model is saved using python joblib and served via a backend API built with FastAPI. FastAPI was chosen for the backend because it's a Python framework, which made it simple to serve the machine learning models that were also written in Python.This backend is responsible for running the python code which handles requests, loads the model, and generates predictions.

The front end was made using Next.js. Next.js was used for the frontend because of my prior experience with it, and it allows for a straightforward hosting process using Vercel. The front end communicates with the FastAPI backend to retrieve and display the AQI predictions on the webpage. The entire web application is deployed and hosted on Vercel. Throughout the process, a CI/CD pipeline using github actions is used to fetch daily data and retrain the model on the newer data.

DATA COLLECTION

1. Selecting the Weather API

This project required a reliable weather forecasting API to source data for Karachi. The selection process was guided by several key criteria: the API had to

be free to use, provide detailed and relevant air quality features, and offer access to historical data for model training.

Several APIs were evaluated to meet these requirements. While OpenWeather was considered, it did not provide the necessary historical data for this project's needs. Another option, IQAir, offered comprehensive data but was a paid service.

After experimentation, the Open-Meteo weather API emerged as the most suitable choice. It successfully met all the essential criteria, providing free, detailed, and historical weather data for Karachi. Therefore, Open-Meteo was selected as the primary data source for this project.

One issue I faced even with the Open-Meteo API was that the data was only reported in hours. My projects timeframe is for days, so to fix this I just ran an aggregation script which aggregated 24 hours each day. The aggregation aqi was done using max for each day (max used as it represents the worst AQI for each day), but for the rest of the numerical features average was used. Using the API I was able to get historical data for 2022- 2025, so about 3 years of daily aqi data.

2. Data Overview

Feature	Purpose
timestamp	The exact date and time the data was recorded
pm10	The concentration of coarse airborne particles smaller than 10 micrometers.
pm25	The concentration of fine airborne particles smaller than 2.5 micrometers.
carbon_monoxide	The amount of carbon monoxide gas in the air.
nitrogen_dioxide	The concentration of nitrogen dioxide gas, often from vehicle emissions.
temperature	The air temperature at the time of measurement.
humidity	The amount of water vapor in the air, expressed as a percentage.
wind_speed	The speed of the wind.
Aqi (Target Var)	The Air Quality Index, a value used to represent the overall air pollution level.

EXPLORATORY DATA ANALYSIS

1. Data Preperation and Descriptive Statistics

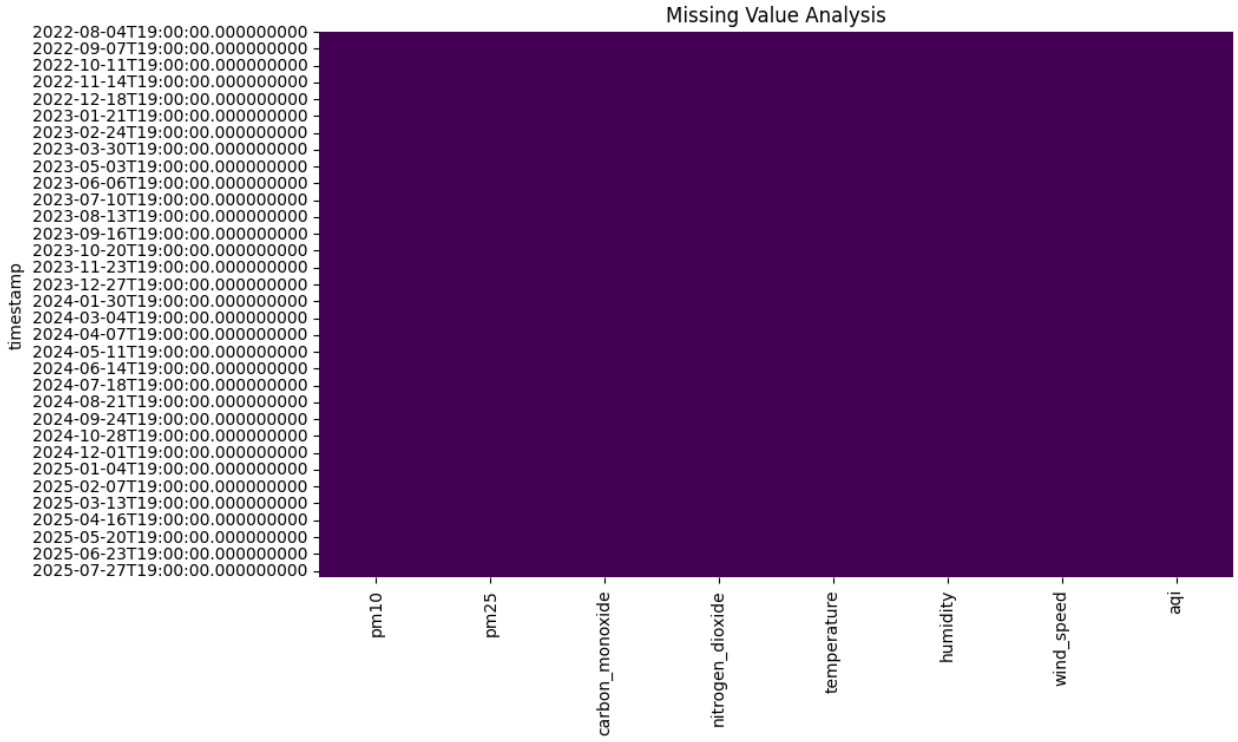
The dataset was loaded, and data types were made consistent. The timestamp column was converted to datetime object and set as the index. Then, df.describe() was called to provide a high-level summary of all features. The results are shown on the table below.

index	pm10	pm25	carbon_monoxide	nitrogen_dioxide	temperature	humidity	wind_speed	aqi
count	1101.0	1101.0	1101.0	1101.0	1101.0	1101.0	1101.0	1101.0
mean	66.12	30.82	475.44	20.97	26.48	66.9	15.5	104.98
std	32.32	16.59	321.67	13.72	3.89	15.97	5.87	43.22
min	0.92	0.65	59.92	0.0	14.8	13.83	5.25	16.0
25%	44.53	20.04	223.62	10.46	23.56	57.42	11.02	73.0
50%	58.11	25.77	362.21	16.7	27.59	73.25	14.35	92.0
75%	79.16	36.32	655.46	28.09	29.58	78.17	19.5	125.0
max	294.48	119.77	2022.96	97.48	33.27	90.25	36.92	297.0

2. Missing Value Analysis

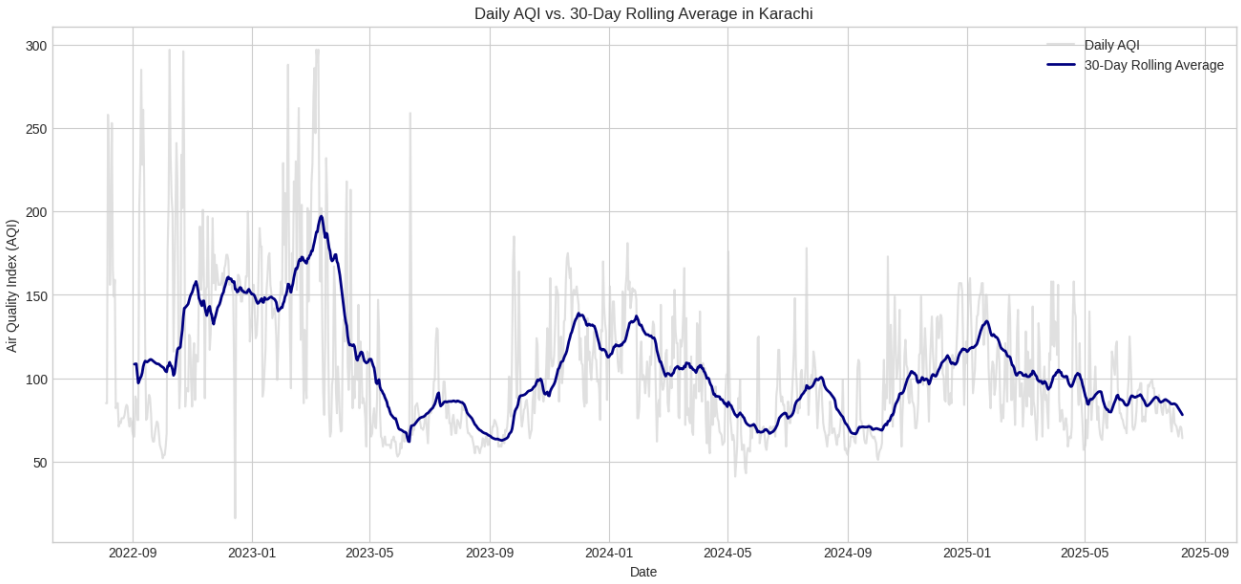
The following table shows the missingness of the data from my Open-Meteo API. It shows that there are no missing values present. Therefore,

no imputation is needed for the dataset.



3. Univariate Analysis

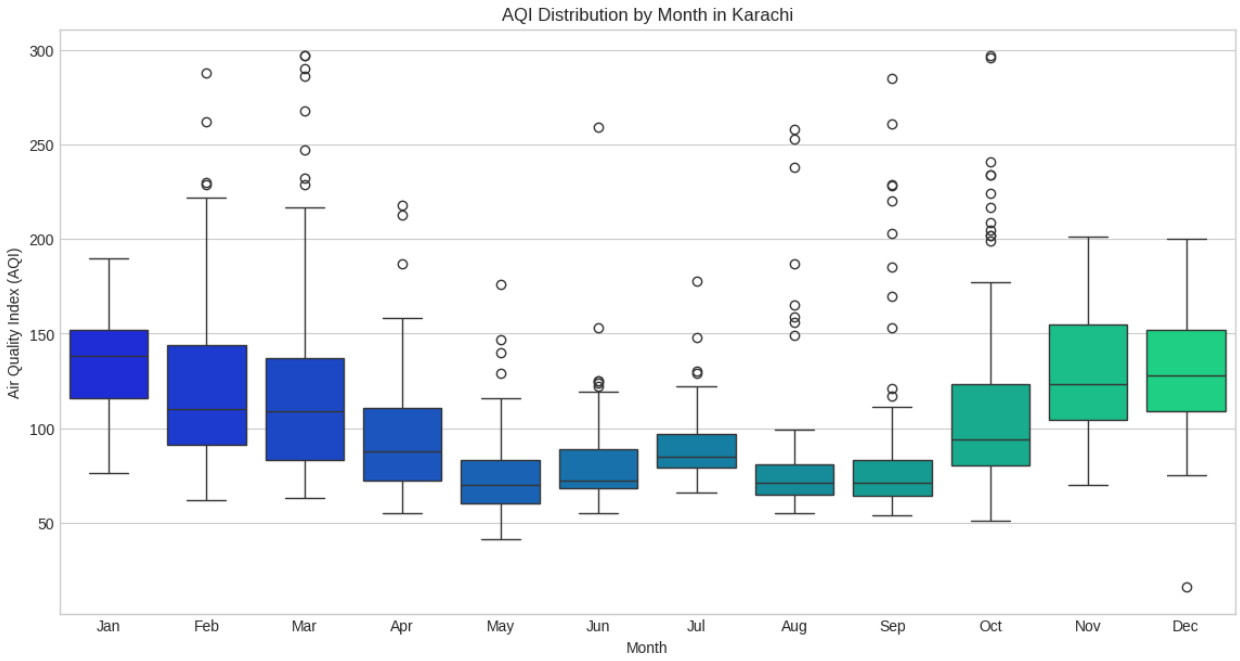
3.1 Analyzing Seasonal Trends



The line graph of the daily AQI trend reveals a clear seasonal pattern. AQI levels consistently spike during the winter months, particularly around the start of the new year, which suggests that air quality is generally worse in winter than in summer.

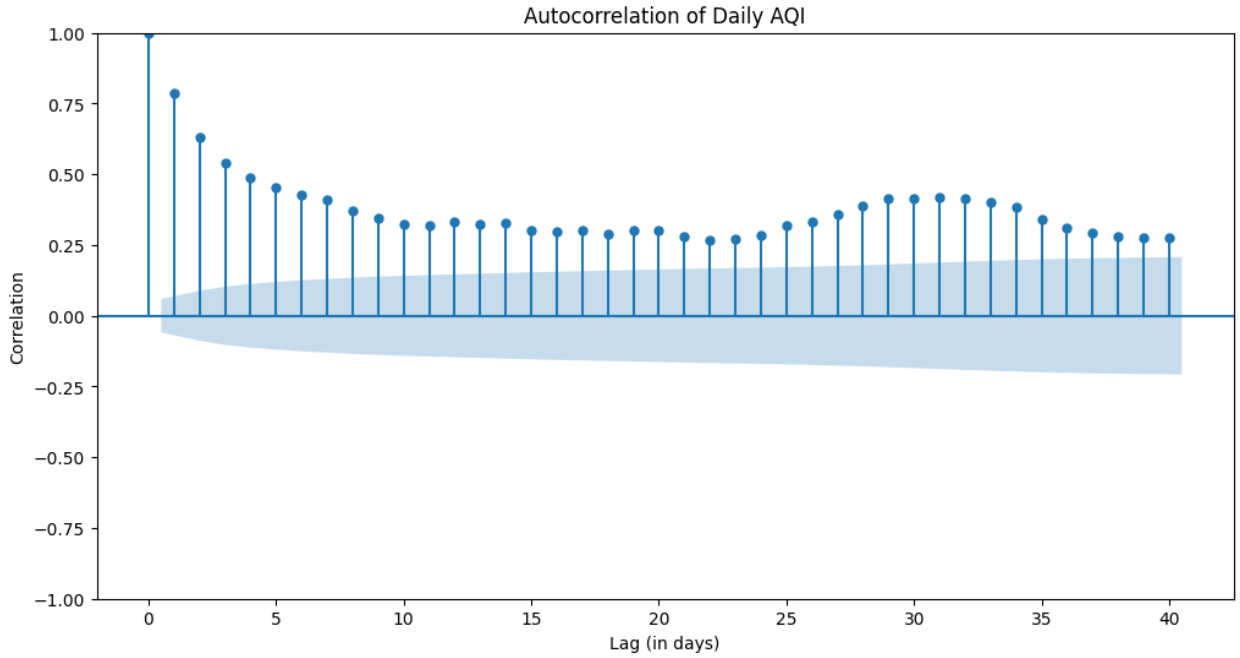
To statistically validate this observation, an Independent Samples T-test was conducted. A T-test determines whether the difference between the average values of two groups is statistically meaningful or simply due to random chance. The null hypothesis was that there is no significant difference in the mean AQI between winter and summer.

The test resulted in a p-value of less than 0.05. This leads us to reject the null hypothesis and conclude that there is a statistically significant difference in AQI between the winter and summer months.



A box plot was also made to further highlight the seasonal variation of the AQI.

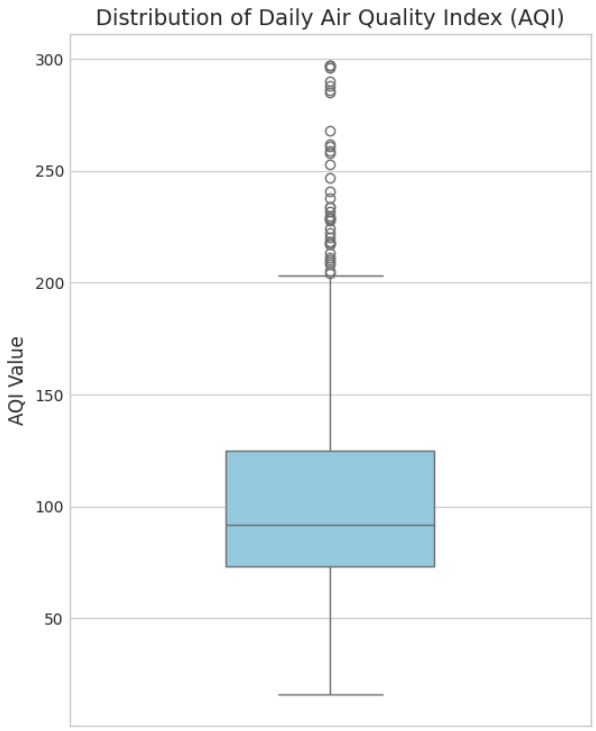
3.2 Analyzing Autocorrelation of AQI



To understand how past AQI values influence future values, an autocorrelation analysis was performed. Autocorrelation measures the relationship between a variable and its past self, helping to determine if the data has "memory."

The resulting Autocorrelation Function (ACF) plot reveals a very strong, positive correlation at the first lag (Lag 1), indicating that yesterday's AQI is an excellent predictor of today's AQI. This correlation remains statistically significant and decreases slowly over many days, which confirms that air quality conditions are persistent; polluted days tend to be followed by other polluted days. This strong autocorrelation is a key finding, as it validates that the time-series data is highly predictable and well-suited for forecasting models.

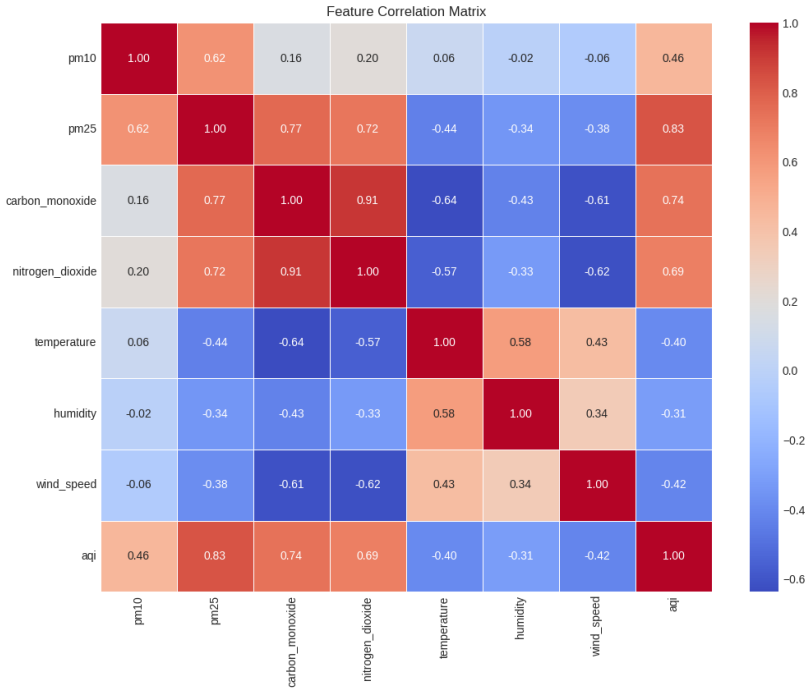
3.3 Analyzing the AQI distribution



The chart above, box plot of the daily AQI, reveals a median value of approximately 92, with 50% of the days falling within a typical range of 73 to 125. The data points located outside this range represent statistically significant outliers, corresponding to days with severe air pollution where the AQI reached nearly 300. These are not data errors but are considered critical, real-world events that are essential to retain for training a model capable of accurately forecasting hazardous air quality conditions.

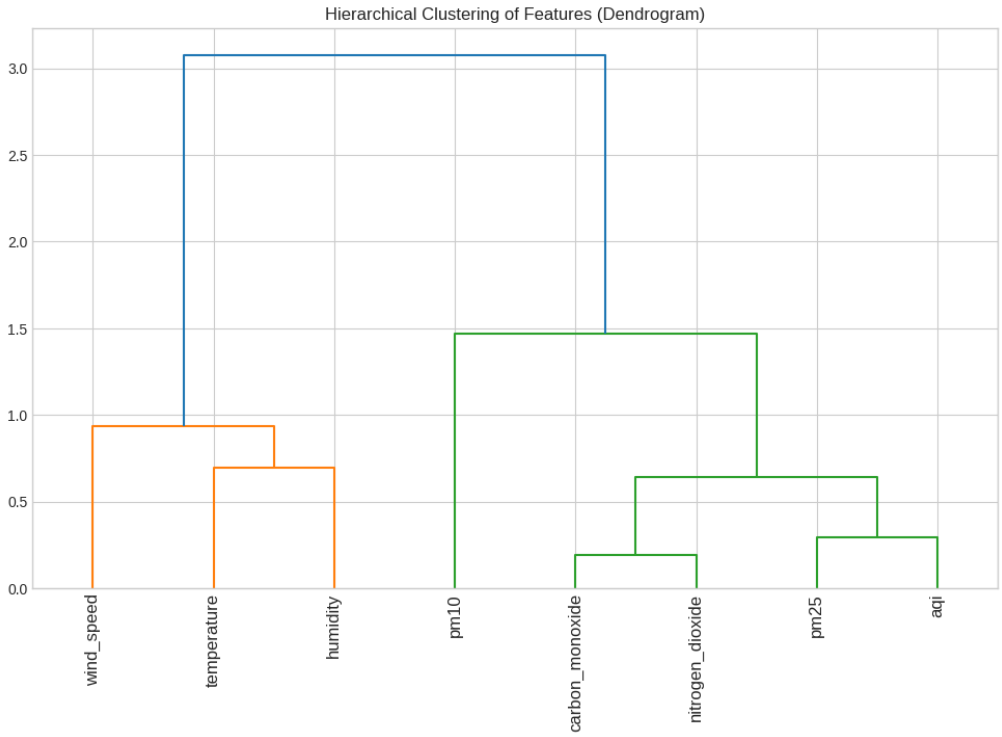
4. Bivariate Analysis

4.1 Correlation Heatmap



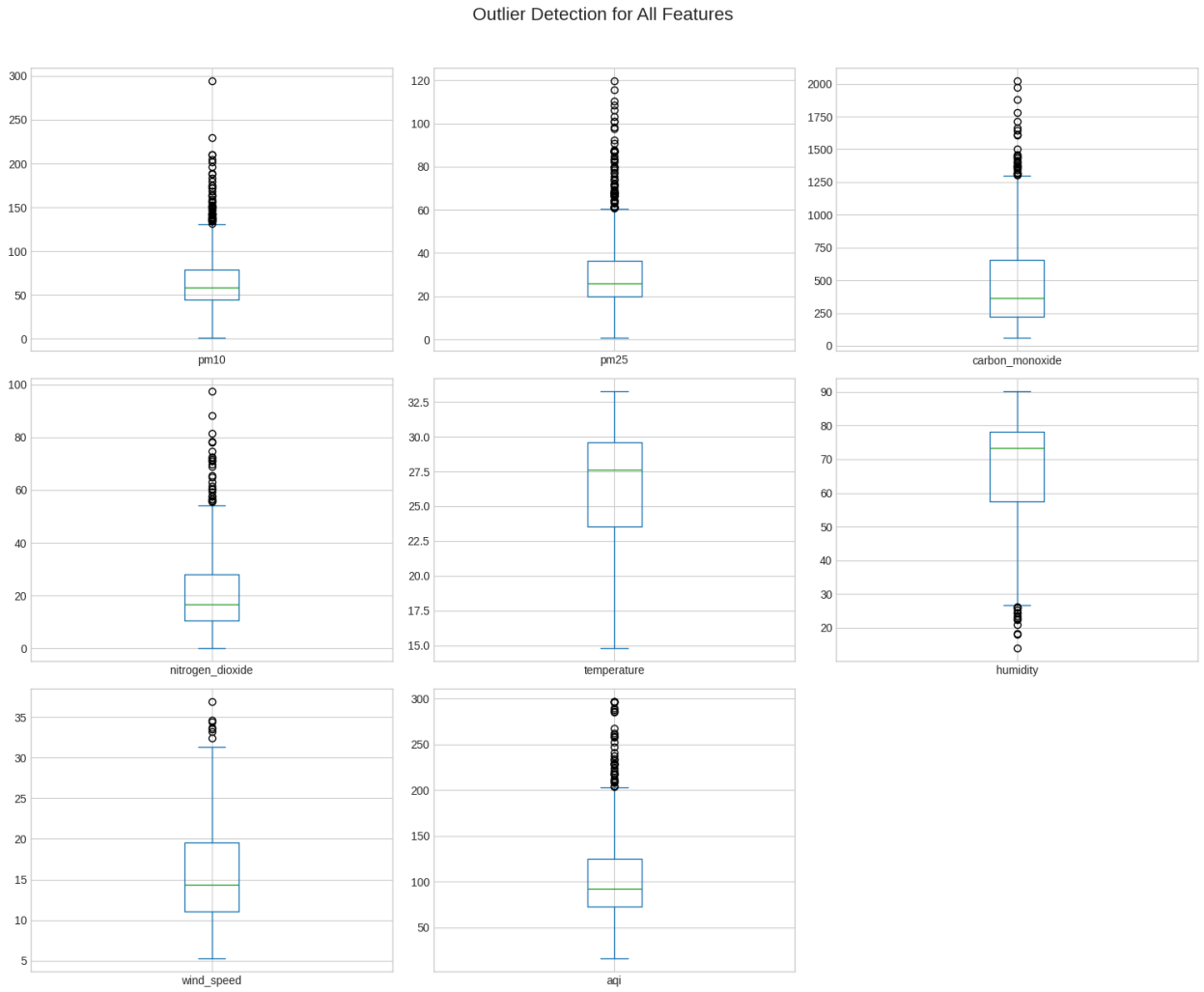
To understand the relationships between the different weather and pollution features, a correlation matrix was generated. This heatmap visualizes the strength and direction of the linear relationship between every pair of variables. The analysis clearly shows that the Air Quality Index (AQI) has a strong positive correlation with pollutant concentrations, especially with PM2.5 (0.83), carbon monoxide (0.74), and nitrogen dioxide (0.69). This confirms that these pollutants are the primary drivers of poor air quality. Inversely, weather variables like wind speed (-0.42) and temperature (-0.40) show a notable negative correlation with AQI. This matches with real life as higher wind speed and temperature contribute to lower AQI.

4.2 Hierarchical Clustering Dendogram



To further visualize these relationships, a hierarchical clustering dendrogram was created. This chart groups features based on the similarity of their correlation patterns, with closely related features appearing on the same branch. The dendrogram reveals two distinct primary clusters: one for weather variables (wind speed, temperature, humidity) and another for all the pollutants and the AQI itself. This visual grouping confirms that the AQI's is tied to the pollutant features, while the weather variables act as a separate group of influencing factors.

5. Outlier Analysis



An outlier analysis was conducted using box plots to identify any extreme data points within the dataset. The investigation revealed that several features have values statistically classified as outliers. However, these points are not considered data errors but are interpreted as valid recordings of real-world, severe pollution events. Removing them would prevent the machine learning model from learning the patterns associated with hazardous air quality, which is a primary objective of this forecasting project.

MODEL TRAINING

1. Data Preprocessing And Feature Engineering

1.1 Data Preprocessing

To prepare the raw data for our models, we first convert the timestamp column into a proper datetime format, and set it as the index. This is essential for time-based analysis. We then sorted all the records chronologically to ensure they were in the correct historical order. There were no missing values in the data, hence no imputation was needed.

1.2 Feature Engineering

1.2.1 Simple Feature Engineering

I created two main types of new features. Lag Features were created by taking the AQI from the previous 7 days and adding each one as a new column. This is important as the EDA showed us that the previous days AQI was a good predictor for current days AQI. I then extracted Time-Based Features directly from the timestamp, creating columns for the month, day_of_week, and day_of_year. These features allow the model to recognize and learn any weekly or seasonal cycles in the air quality data.

1.2.2 Advanced Feature Engineering

I implemented a second layer of more advanced feature engineering to provide the model with a deeper, more contextual understanding of the data. This involved three key techniques. First, I created Rolling Window Features by calculating the mean and standard deviation of key variables (like aqi and pm25) over the last 3 and 7 days. These rolling stats are crucial because they capture the recent trend (is the pollution getting better or worse?) and volatility (are conditions stable or fluctuating wildly. This is needed as generally AQI moves as a trend as opposed to moving in extreme fluctuations. Then I added engineered Interaction Features to model real-world physical relationships, such as dividing PM2.5 by wind speed to create a feature that directly represents how well pollutants are being dispersed. Finally, I applied a Cyclical Transformation using sine and cosine functions to the month and day_of_week features. This standard technique converts time into

a circular format, allowing the model to properly understand the continuous nature of seasons and weeks (for example December is right next to January). Overall, the advanced feature selection yield much better results on the same models than the basic feature selection.

2. Model Setup

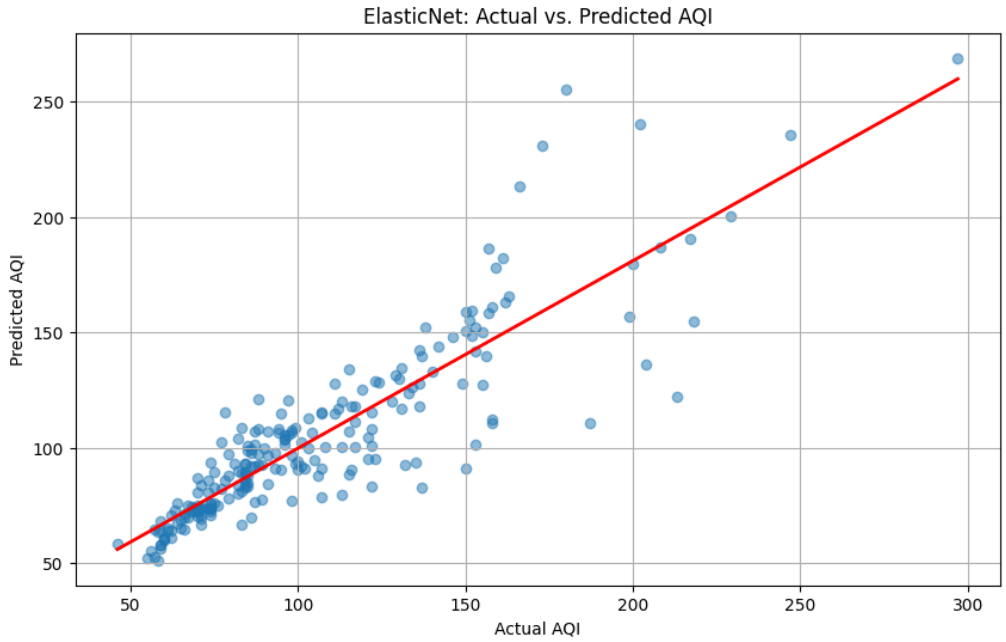
2.1 Data Splitting Strategy

Instead of a simple random split, I used a more robust method called Stratified Splitting. A regular random split can sometimes create an unbalanced test set by pure chance, for example, by putting very few high-pollution days into it. This would give us a misleadingly optimistic score. Stratified splitting avoids this problem by maintaining the original data's proportions. Since our AQI target is a number, I first grouped the AQI values into categories (Good, Moderate, Unhealthy). The stratified method then splits the data while ensuring that the percentage of days in each of these categories was the same in both the training and the test sets. This is a better approach because it guarantees that our model is tested on a fair and representative sample of all air quality conditions. Our model needs to be reliable even for extreme AQI predictions, and stratified splitting allows us to better judge model performance.

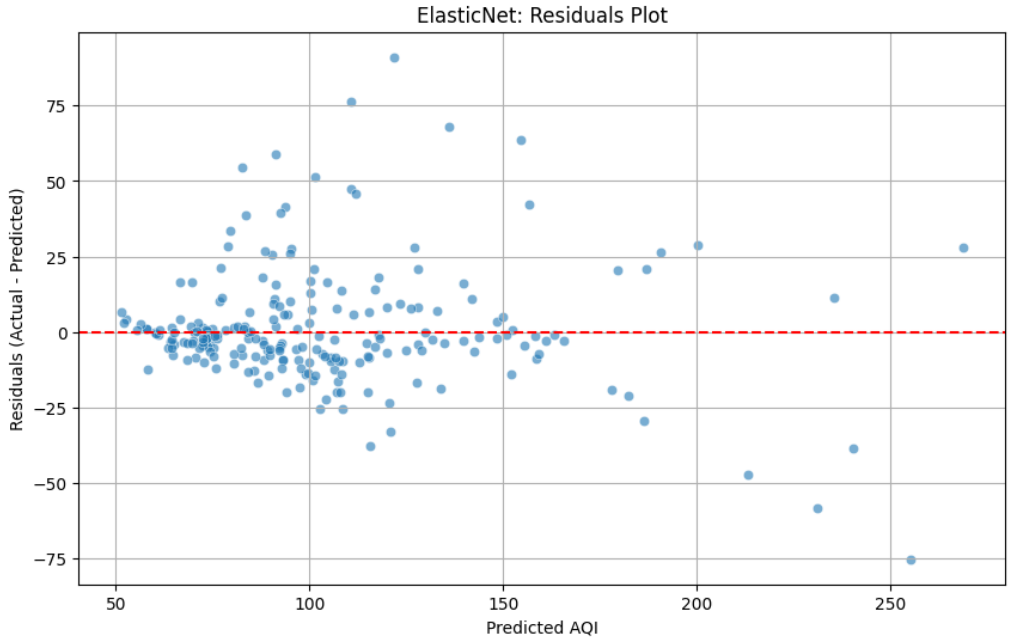
2.2 Evaluation Metrics

I primarily relied on four key metrics to judge accuracy. R-squared (R^2) tells us what percentage of the change in AQI our model can explain, where a higher value is better. Mean Absolute Error (MAE) gives us the average error in AQI points, providing a straightforward measure of how far off our predictions are on average. Root Mean Squared Error (RMSE) also measures the average error but gives a much heavier penalty to large mistakes, helping us see if the model is making any significant blunders. Mean Absolute Percentage Error (MAPE) expresses the average error as a percentage, which helps in understanding the error relative to the AQI value. For all error metrics (MAE, RMSE, MAPE), a lower score is better.

To visually confirm these numbers, we used an Actual vs. Predicted plot to see how closely our model's predictions align with the true values. Example plot shown below.



A Residuals Plot to ensure the model's errors were random and not following a hidden pattern. Together, these plots provide a overview of how well our model is performing.



3. Model Types

3.1 Linear Models

The first models I trained were series of linear models. This group included standard Linear Regression along with regularized versions like Ridge, Lasso, and ElasticNet. These models are fast to train and work by finding simple, straight-line relationships between the input features and the AQI. Overall, these models performed well, and successfully captured the main trends in the data, but they were outperformed by more complex, non-linear models. I concluded that the linear models were underfitting the data, and moved on to more complex models.

3.2 Tree Based Models

Tree based models make predictions by learning a set of splitting rules from the data (similar to a flowchart). I started with a single Decision Tree, which provided a decent baseline but was outperformed by more advanced methods. I then tested Random Forest, which combines hundreds of individual decision trees. By averaging the predictions of all the trees, it produces a much more accurate and stable result. After hyperparameter tuning, the Random Forest emerged as the top-performing model in our entire study, achieving the highest R-squared score of 0.832. This result shows that its ability to capture complex patterns makes it extremely well-suited for predicting air quality.

3.3 Boosting Models

The final category of algorithms we tested was boosting models. Like Random Forest, these models also use multiple trees, but they build them one after another in a sequence. Each new tree in the sequence is specifically trained to correct the errors made by the one before it, allowing the model to get progressively better with each step. I evaluated a powerful suite of these models, including Gradient Boosting, XGBoost, LightGBM, and CatBoost. After tuning, both the CatBoost and XGBoost models delivered excellent results, with R-squared scores over 0.81, placing them right behind our top Random Forest model and proving they are also top-tier choices for this prediction task. However, these models were prone to overfitting.

3.4 Ensemble Models

After I got some good base models I explored ensemble techniques, which create a better model by combining the predictions of several strong individual models. I tested two primary strategies: a Weighted Averaging Ensemble and a more advanced method called Stacking. Stacking involves a two-stage process: first, the base models (like RandomForest and SVR) make their own predictions. Then, a final meta-model is trained to combine those predictions, learning the strengths and weaknesses of each base model to produce a final, improved forecast. After testing both architectures, the simpler Weighted Averaging Ensemble was better, achieving the highest R-squared score of my entire project at 0.858. I also tried stacking the ensemble models themselves (double stacking), however this resulted in high overfitting. I combined all my best individual models with the weighted averaging ensemble to achieve my best model.

3.5 Advanced Models

I tested deep learning models to see if they could offer a performance advantage. I focused on two architectures which specialize in sequence data: Convolutional Neural Networks (CNNs), which are designed to automatically detect important patterns within a time window, and Long Short-Term Memory (LSTM) networks, which use an internal memory to understand the order and long-range dependencies in a sequence. After preparing the data by scaling it and reshaping it into 7-day windows, I trained both a standalone CNN and a heavily regularized LSTM. However, both models performed poorly, yielding negative R-squared scores (means predictions were worse than simply guessing the average AQI value). I concluded the cause of this failure was extremely overfitting due to the small dataset size. With fewer than 1,000 training samples, these complex, high-capacity neural networks simply memorized the noise in the training data instead of learning the true underlying patterns, making them unable to generalize to unseen test data. Due to the small data size, advanced models which are generally used for AQI prediction were ineffective in predicting the AQI.

4. Hyperparameter Tunning

After finding the best models for this project, I tried to improve the performance, using hyperparameter tuning. This involves manipulating the base model parameters to achieve better performance. Hyperparameters are the high-level config settings of an algorithm, such as the number of trees in a Random Forest or the learning rate in XGBoost. Instead of manually testing combinations, I used an automated and efficient technique called RandomizedSearchCV. This method is much faster than an exhaustive Grid Search because it intelligently samples a fixed number of random parameter combinations from a grid of possible values that I defined for each model. To ensure the results were reliable, the process used 5-fold cross-validation, meaning each combination was trained and tested five times on different subsets of the data. This automated search identified the optimal settings for each algorithm, which led to a significant improvement in predictive accuracy over the baseline unoptimized versions.

5. Model Comparision and Final Selection

After training and optimizing a diverse range of models, the final step was to compare them directly to select a single champion for the prediction pipeline. To do this, I compiled the results into a performance leaderboard, ranking each model based on the evaluation metrics on the unseen test data. The results clearly identified a top performer: the Weighted Averaging Ensemble. This model, which ensembles the strong base models of RandomForest, CatBoost, and XGBoost models, achieved the highest R-squared score of the entire project at 0.858. I used the advanced feature engineering techniques and uses weights to average the models which themselves were optimized using RandomizedCV. It also had one of the lowest Root Mean Squared Errors (RMSE). Although, it was not the best in all evaluation metrics, I chose it due to it having the highest R-square model. While the base optimized RandomForest was a very close runner-up, the ensemble's slight edge in performance made it the definitive choice. Below is the top 10 leaderboard for my models. More models were trained but here the 10 best are displayed.

Rank	Experiment Name	R2 Score	RMSE	MAE	MAPE	Feature Set
1	Ensemble Weighted	0.858	16.18	10.73	9.68%	Advanced

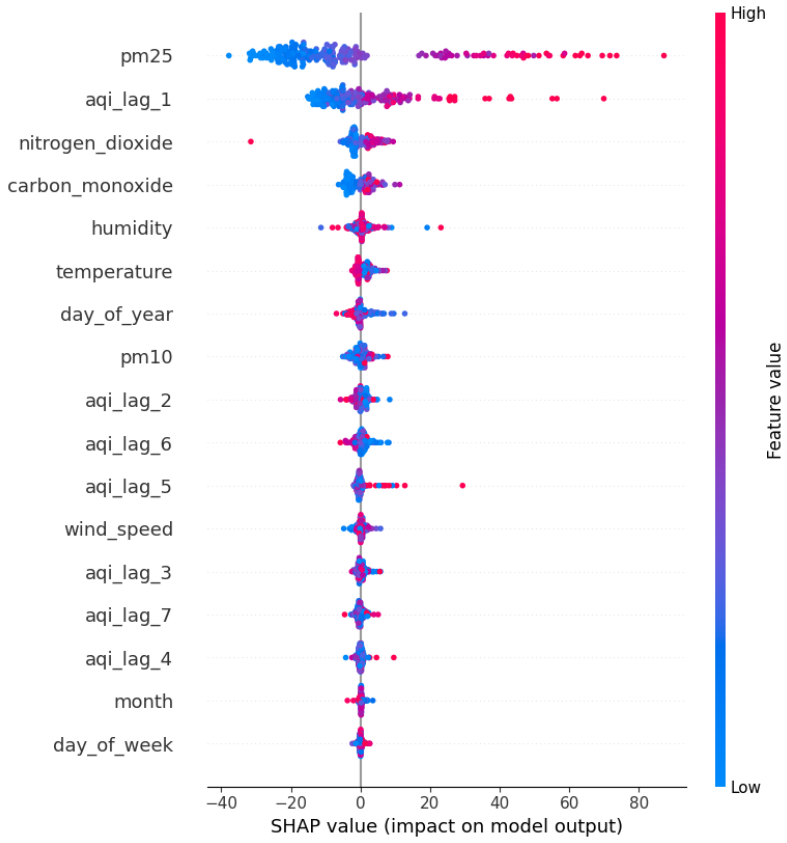
	Average (Advanced Features)					
2	Ensemble Weighted Average	0.836	16.61	10.55	9.46%	Advanced
3	Optimized RandomForest (Base Features)	0.832	16.83	10.99	9.94%	Advanced
4	Optimized CatBoost (Base Features)	0.831	16.89	10.69	9.57%	Base
5	Base CatBoost (Base Features)	0.827	17.09	10.97	9.71%	Base
6	Ensemble Stacking (Base Features)	0.818	17.54	10.92	9.63%	Base
7	Optimized XGBoost (Base Features)	0.816	17.63	10.70	9.46%	Advanced
8	Optimized GradientBoosting (Base Features)	0.809	17.93	11.12	9.79%	Base
9	Optimized LightGBM (Base Features)	0.801	18.32	11.74	10.47%	Base
10	Base RandomForest (Base Features)	0.800	18.37	11.58	10.21%	Base

6. Champion Model Interpretation (XAI)

After selecting the Weighted Averaging Ensemble as the champion model, the final step was to understand how it makes its predictions. Using a model as a black box is risky, so it is advisable to use Explainable AI (XAI) techniques like SHAP and LIME, I could verify that the model was learning sensible patterns from the data and identify the key features and their effects on air pollution in Karachi.

6.1 Global Feature Importance using SHAP

To understand which features were most important to the model overall, I used a SHAP Summary Plot. This plot visualizes the impact of every feature for every prediction in the test set. The results confirmed that the model learned logical relationships: pm25 and aqi_lag_1 were identified as the two most influential features. The plot clearly showed that high values of these features consistently pushed the AQI prediction higher, which is expected and follows real life. Meteorological factors like humidity and temperature were also shown to have a moderate, also has high impact. Below is the SHAP plot for my Champion Model.



6.2 Local Analysis using LIME

To understand why the model made a specific prediction on a given day, I used LIME. This technique builds a simple, interpretable model around a single prediction to explain its reasoning. For example, when analyzing a day with a very high predicted AQI, LIME showed that the prediction was driven primarily by the high values of pm25, carbon_monoxide, and the previous day's AQI. This ability to break down individual forecasts provides explainability, making it possible to understand the model's reasoning for any prediction. Below is a LIME plot for a specific prediction.

For this chart, when I analyzed a day where the model correctly predicted a very high AQI of 296.85, the LIME plot clearly showed that the decision was driven by three key factors: the extremely high PM2.5 level (87.06), the already high AQI from the previous day (290), and a high Carbon Monoxide reading.

CI/CD PIPELINES

To keep the project running automatically (serverless stack), I used GitHub Actions. This tool lets me schedule scripts to run directly from my GitHub repository without needing a dedicated server. I created three separate automated workflows to handle all the project's updates.

1. Fetch Daily AQI Data

The first workflow is the Daily Data Fetcher. Its job is to keep my dataset current. It runs four times a day and automatically fetches the latest weather and air quality data from online APIs. It then processes this new data, calculates daily averages, and appends the new records to my main historical CSV file. Finally, it commits the updated data file back to the repository. This keeps adding more data, making the model better as time passes on.

2. Daily Model Retraining

The second workflow is the Daily Model Retraining pipeline. This process ensures the prediction model is always as smart as possible. It runs once every night at midnight Karachi time. The script takes the newly updated dataset, applies all the advanced feature engineering steps, and completely retrains the champion ensemble model from scratch.

3. Deploy Backend to Hugging Face

The third workflow serves as the dedicated Continuous Deployment (CD) pipeline for the projects FastAPI backend.I configured this workflow to run at 2 AM daily, but it could be made to run on every commit. Its primary responsibility is to deploy the latest version of the application from the main branch to its Hugging Face Space (more info about the deployment in the Deployment section). A problem I faced with Hugging face is that it rejected pushes containing a history of large model files. To solve this, the workflow performs an in-memory history rewrite: it checks out the repository, creates a new orphan branch with a single clean commit, and force-pushes only this branch to the hugging space face. This workflow is the most essential as it ensures that the changes performed by the other workflows are reflected on our backend deployment.

BACKEND ARCHITECTURE AND LOGIC

The backend uses FastAPI for speed and async support, Pandas/Numpy for data handling, Joblib for loading ML models, and Requests for fetching live weather and air quality data.

1. API Endpoints

The API uses two primary endpoints to serve the frontend application. The main endpoint, /api/forecast, is where the actual inference happens; it takes no input and returns a JSON object containing the latest available AQI value for the current day and a list of predicted AQI values for the next three days. A secondary utility endpoint, /api/status, provides crucial metadata about the backend . It returns a JSON object indicating that the service is online and, most importantly, includes a timestamp of when the machine learning model file was

last updated. This was done as a debugging procedure to check whether the backend deployment was being updated or not.

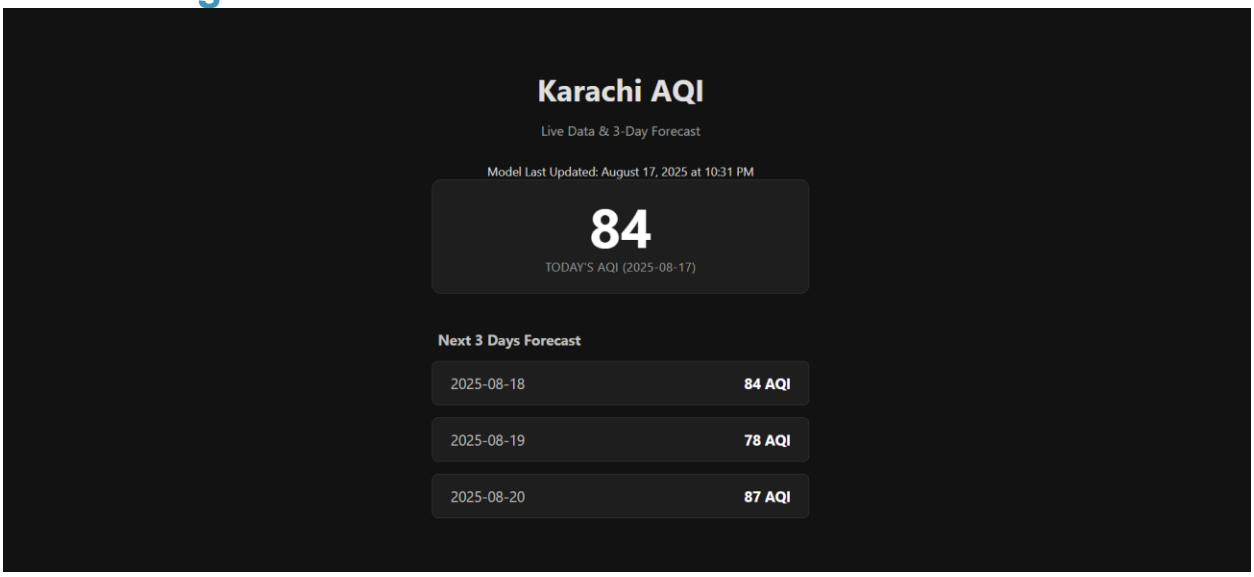
2. Prediction Pipeline Logic

When a request hits the /api/forecast endpoint, multi-step prediction pipeline is triggered. First, the application loads a historical dataset from a local CSV file and a pre-trained .joblib model file of our best model. It then identifies today's AQI by taking the most recent entry from this historical data.Then get_future_forecast_from_api function makes a fetch to the Open-Meteo weather API to get forecast data for the upcoming days. This future data is then processed to get the same feature set the model was trained on (same feature engineering). The create_features_for_single_day function generatesfeatures, including lagged AQI values, rolling averages, and cyclical date features. This feature set is fed into the loaded model to produce an AQI prediction, which is then added to the historical data to provide context for the next day's prediction, then that prediction is added and so on. This process of getting next day's prediction based on the previous day's prediction is called iterative predictions.

FRONTEND IMPLEMENTATION

The frontend of project was made using Next.js 15. The frontends role is to take the JSON objects from the backend API and present the information in a visually appealing format.

1. UI Design



I made a very basic UI for this project. It is structured into three main sections: a header that establishes the application's context, a display for Today's AQI, and a list with the 3-day forecast. To ensure the application always feels current, the date displayed for Today's AQI is generated client-side using the user's current system time.

DEPLOYMENT AND MLOPS

https://aqi-predictor-self.vercel.app/

The project is deployed using decoupled architecture that uses the specific strengths of specialized cloud platforms for each component of the stack. This separation of concerns ensures optimal performance for the user-facing application while accommodating the requirements of the machine learning backend. This is done as the backend due to its python libraries is too heavy and can’t be hosted on a singular hosting service like Vercel. The application is split as two distinct services: a lightweight Next.js frontend and a heavyweight Python

backend. The frontend is hosted on Vercel while the FastAPI backend is hosted on Hugging Face Spaces.

1. Frontend Deployment (Vercel)

The deployment is directly linked to the main GitHub repository, where Vercel's CI/CD automatically detects any push to the main branch. It then builds the Next.js application and deploys it on my URL.

It uses the vercel.json configuration file to path the frontend directory as the project's root. This is needed as if the root pointed to the root of my repository, Vercel would also pick up my FastAPI backend, which would be too large and would break deployment.

2. Backend Deployment (Hugging Face Spaces)

The FastAPI backend is deployed on Hugging Face Spaces, selected because its container-based environment easily handles the large library dependencies that exceed the size limits of typical serverless platforms. The deployment is managed via a Dockerfile, which creates a reproducible and isolated environment for the Python application. A key challenge was handling the large model file, which was solved by using Git LFS to manage the binary file as required by the Hugging Face platform. The Deploy Backend to Hugging Face pipeline triggers the refresh for the hugging face space, ensuring that the data and model updates are reflected on the backend deployment.