PEARLS AQI PREDICTOR

Let’s predict the Air Quality Index (AQI) in your city in the next 3 days, using a 100% serverless stack.

BY: BARIRAH BAKHTIAR
EMAIL: barirahbakhtiar315@gmail.com

1. Table of Content

1. INTRODUCTION & PROBLEM

Introduction:

This project is a complete End-to-End Machine Learning Pipeline for Air Quality Forecasting.
It uses real-time data ingestion, feature engineering, model training, prediction, storage in a feature store, and deployment via Docker.
It integrates with GitHub repositories for code/data versioning and uses Hopsworks Feature Store to maintain clean, versioned datasets — ensuring a production-level data pipeline.
The solution focuses on predicting key air pollutants like PM2.5 and PM10 using multiple machine learning models.

Problem Statement:

Air pollution forecasting is critical for human health and policy-making.
However, real-time, accurate prediction faces challenges:

- Noisy, missing, or delayed data.
- Changing pollution patterns.
- Requirement for fast, reliable predictions.

Thus, the problem is to design an automated, scalable system that:

- Fetches and preprocesses live air quality data.
- Stores clean features for consistent model training.
- Trains strong predictive models.
- Generates reliable short-term forecasts.
- Runs consistently in production environments.

2. PARADIGMS

This project uses several advanced paradigms:

a. Machine Learning Paradigm

- Supervised learning using historical AQI data.
- Models trained on input features like PM2.5, PM10, SO2, CO, NO2, O3, etc.
- Regression models predict continuous AQI values.

b. Automation & Pipeline Paradigm

- Data Fetch → Store → Feature Engineering → Model Train → Predict → Evaluate → Deploy.
- Scripts are automated to run sequentially or through GitHub Actions.

c. Containerization Paradigm (Docker)

- Docker ensures that environment, libraries, and dependencies are consistent across local, cloud, and production deployments.

d. Feature Store Paradigm (Hopsworks)

- Hopsworks Feature Store is used for:
    - Storing historical and real-time features.
    - Version-controlling datasets.
    - Ensuring feature consistency between training and prediction.
    - Making model retraining easy as new data flows in.

e. MLOps Paradigm

- Elements of MLOps are visible:
    - Feature versioning.
    - Automated model training.
    - Data lineage tracking through Hopsworks.
    - Reproducible pipelines.

3. ALGORITHM & EXPLANATION

The overall solution can be broken into 7 Stages:

Stage 1: Data Fetching

- **File:** fetch_and_store_currentdata.py, fetch_and_store_history.py
- **Function:**
    - Fetches air pollution data from APIs (or GitHub repositories that auto-update live AQI).
    - Saves data temporarily into CSV files.
- **Important Details:**
    - Data includes fields like datetime, PM2.5, PM10, CO, NO2, SO2, and others.
    - Handles missing values, normalizes timestamps.

Stage 2: Feature Engineering and Storage (Hopsworks Feature Store)

- **Task:**
    - Clean fetched data.
    - Transform raw data into ML-ready features.
    - Upload feature datasets into the Hopsworks Feature Store:
        - Primary Key: timestamp, location
        - Features: PM2.5, PM10, etc.
- **Benefits:**
    - Historical features are versioned.
    - Future model retraining uses consistent data.
    - Faster access compared to local CSVs.
    - Enables online/offline feature serving.
- **Tools Used:**
    - Hopsworks featurestore API via Python.

Stage 3: Model Training

- File: train.py
- Models Used:

1. Random Forest Regressor

- **Definition:**
    - Random Forest is an ensemble learning algorithm.
    - It builds many decision trees during training.
    - The final prediction is the average of predictions from all the trees.

- **Strengths:**
  - Handles non-linear relationships very well.
  - Robust against overfitting (by averaging many trees).
  - Can work with missing data and noisy features.
  - Feature importance insights can be extracted easily.
- **Why it's best for this project:**
  - Environmental data like AQI is non-linear and noisy.
  - Random Forest generalizes well even on messy real-world data.
  - Very good baseline model for tabular data like pollution readings.

2. Ridge Regression

- **Definition:**
  - Ridge Regression is a linear regression model that uses L2 regularization.
  - It penalizes large coefficients to avoid overfitting.
- **Strengths:**
  - Works very well when features are correlated.
  - Prevents overfitting by shrinking coefficients.
  - Simple, fast to train, and interpretable.
- **Why it's best for this project:**
  - Some air quality parameters (like PM2.5, PM10, CO) might be correlated.
  - Ridge can handle this multicollinearity efficiently.
  - Provides a simple linear baseline to compare against more complex models.

3. XGBoost Regressor

- **Definition:**
  - XGBoost stands for Extreme Gradient Boosting.
  - It builds models sequentially, where each new model focuses on correcting errors made by previous models.
  - It uses gradient descent optimization to minimize errors.
- **Strengths:**
  - Extremely high predictive power.
  - Handles missing values internally.
  - Supports regularization (both L1 and L2), helping to prevent overfitting.
  - Efficient and fast, even with large datasets.
  - Works well on tabular data.
- **Why it's best for this project:**
  - XGBoost often outperforms other models on structured/tabular data (like AQI features).
  - Can model complex non-linear interactions between features.
  - Good choice when we want a high-performance, production-grade model.

- **Training Process:**
  - Read data from Hopsworks Feature Store.
  - Train multiple models.
  - Save the models into .pkl files (serialized).
- **Model Evaluation Metrics:**
  - R^2 Score (Coefficient of Determination)
  - Mean Squared Error (MSE)
  - Root Mean Squared Error (RMSE)
- **Output:**
  - Saved trained models.
  - model_results.csv with model scores for easy comparison.

Stage 4: Prediction (Forecasting)

- File: predict_forcaste.py
- Task:
  - Load trained models.
  - Fetch the latest real-time data from Hopsworks.
  - Predict PM2.5, PM10 values for the next few hours/days.
  - Store predictions locally (forecast.csv) and optionally send back to Hopsworks for monitoring.

Stage 5: Containerize with Docker

- **File:** Dockerfile
- **Container Steps:**
  - Base Image: Python (official image).
  - Install required libraries (requirements.txt).
  - Copy code files.
  - Set environment variables.
  - Run training/prediction scripts automatically.
- **Advantages:**
  - Same environment everywhere: no "works on my machine" problems.
  - Easy scaling to cloud servers.

Stage 6: Automation via GitHub

- **GitHub Repository Management:**
  - Code is version-controlled on GitHub.
  - Optional: GitHub Actions could be used to trigger data fetches, retraining, or evaluations periodically (e.g., every hour).
- **Advantages:**
  - Full pipeline automation.
  - Ensures continuous improvement as new data comes in.

4. CODE

fetch_and_store_currentdata.py

```
import requests
import pandas as pd
from datetime import datetime
import hopsworks
def fetch_current_air_pollution(lat, lon, api_key):
    url = f"http://api.openweathermap.org/data/2.5/air_pollution?lat={lat}&lon={lon}&appid={api_key}"
    response = requests.get(url)
    return response.json()
def process_current_data(response_json):
    record = response_json.get("list", [])[0]
    dt_obj = datetime.utcfromtimestamp(record["dt"])
    data = {
        "datetime": dt_obj.isoformat(),
        "aqi": record["main"]["aqi"],
        **record["components"]
    }
    df = pd.DataFrame([data])
    # Fixing schema issues for Hopsworks compatibility
```

```
    df["nh3"] = df["nh3"].astype('int64')  # Ensuring nh3 is bigint
    pollutant_cols = ["co", "no", "no2", "o3", "pm2_5", "pm10", "so2"]
    for col in pollutant_cols:
        df[col] = df[col].astype('float64')  # Ensuring pollutants are float
    return df
def save_to_hopsworks(df):
    if df.empty:
        print("❌ DataFrame is empty. Nothing to insert.")
        return
    df["datetime"] = df["datetime"].astype(str)
    project = hopsworks.login(
        project="barirahb",
        api_key_value="fiEMd5rPImQfpLA8.8qWqBnYd7YjnDY0oNriJquZFzcSHVAsBIkdiDGL8bAkRgBIWa8pye4cUjfVFHU
Ae"
    )
    fs = project.get_feature_store()

    fg = fs.get_feature_group(
        name="historical_air_quality",
        version=1
    )
    fg.insert(df, write_options={"wait_for_job": True})
    print("✅ Current data inserted into feature store.")
if __name__ == '__main__':
    api_key = '38138b1c1a295cef06f2d6918a10e562'
    latitude = 24.8607
    longitude = 67.0011
    current_data = fetch_current_air_pollution(latitude, longitude, api_key)
    df_current = process_current_data(current_data)
    save_to_hopsworks(df_current)
```

fetch_and_store_currentdara.py

```
import requests
import pandas as pd
from datetime import datetime, timedelta
import hopsworks
def fetch_historical_air_pollution(lat, lon, start, end, api_key):
    url =
f"http://api.openweathermap.org/data/2.5/air_pollution/history?lat={lat}&lon={lon}&start={start}&end={end}&appid={api_key
}"
    response = requests.get(url)
    return response.json()
def process_historical_data(response_json):
    records = []
    for entry in response_json.get("list", []):
        dt_obj = datetime.utcfromtimestamp(entry["dt"])
        data = {
            "datetime": dt_obj.isoformat(),
            "aqi": entry["main"]["aqi"],
            **entry["components"]
```

```
            }
            records.append(data)
        return pd.DataFrame(records)
    def save_to_hopsworks(df):
        if df.empty:
            print("❌ DataFrame is empty. Nothing to insert.")
            return
        df["datetime"] = df["datetime"].astype(str)
        project = hopsworks.login(
            project="barirahb",
            api_key_value="fiEMd5rPImQfpLA8.8qWqBnYd7YjnDY0oNriJquZFzcSHVAsBIkdiDGL8bAkRgBIWa8pye4cUjfVFHU
    Ae"
        )
        fs = project.get_feature_store()
        fg = fs.get_or_create_feature_group(
            name="historical_air_quality",
            version=1,
            primary_key=["datetime"],
            description="Historical air pollution data from OpenWeather",
            online_enabled=False
        )
        fg.insert(df)
        print("✅ Historical data inserted into feature store.")
    if __name__ == '__main__':
        api_key = '38138b1c1a295cef06f2d6918a10e562'
        latitude = 24.8607
        longitude = 67.0011
        end_date = datetime.utcnow()
        start_date = end_date - timedelta(days=3)
        start_unix = int(start_date.timestamp())
        end_unix = int(end_date.timestamp())
        historical_data = fetch_historical_air_pollution(latitude, longitude, start_unix, end_unix, api_key)
        df_historical = process_historical_data(historical_data)
        save_to_hopsworks(df_historical)
```

train.py

```
    import hopsworks
    import pandas as pd
    import numpy as np
    import pickle
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.linear_model import Ridge
    from xgboost import XGBRegressor
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
    # Connect to Hopsworks
    project = hopsworks.login(
        project="barirahb",
        api_key_value="fiEMd5rPImQfpLA8.8qWqBnYd7YjnDY0oNriJquZFzcSHVAsBIkdiDGL8bAkRgBIWa8pye4cUjfVFHUAe
    "
```

```
)
fs = project.get_feature_store()
fg = fs.get_feature_group("historical_air_quality", version=1)
df = fg.read()
# Preprocessing
df['datetime'] = pd.to_datetime(df['datetime'])
df = df.sort_values('datetime')
# Feature Engineering
df['pm10_lag1'] = df['pm10'].shift(1)
df['pm10_lag2'] = df['pm10'].shift(2)
df['pm10_avg3'] = df['pm10'].rolling(window=3).mean()
df = df.dropna()
# Features and Target
features = ['pm10', 'pm2_5', 'pm10_lag1', 'pm10_lag2', 'pm10_avg3']
target = 'aqi'
X = df[features]
y = df[target]
# Train models
models = {
    'RandomForest': RandomForestRegressor(),
    'XGBoost': XGBRegressor(),
    'Ridge': Ridge()
}
model_results = {}
for name, model in models.items():
    model.fit(X, y)
    y_pred = model.predict(X)
    mse = mean_squared_error(y, y_pred)
    mae = mean_absolute_error(y, y_pred)
    r2 = r2_score(y, y_pred)
    model_results[name] = {'model': model, 'MSE': mse, 'MAE': mae, 'R2': r2}
# Save models
for name, result in model_results.items():
    with open(f'{name}_model.pkl', 'wb') as f:
        pickle.dump(result['model'], f)
# Save model metrics
results_df = pd.DataFrame.from_dict(model_results, orient='index').drop(columns='model')
results_df.reset_index(inplace=True)
results_df.rename(columns={'index': 'Model'}, inplace=True)
results_df.to_csv('model_results.csv', index=False)
```

predict_forcaste.py

```
import hopsworks
import pandas as pd
import numpy as np
import pickle
from datetime import timedelta
# Connect to Hopsworks
project = hopsworks.login(
    project="barirahb",
```

```
    api_key_value="fiEMd5rPImQfpLA8.8qWqBnYd7YjnDY0oNriJquZFzcSHVAsBIkdiDGL8bAkRgBIWa8pye4cUjfVFHUAe
"
)
fs = project.get_feature_store()
fg = fs.get_feature_group("historical_air_quality", version=1)
df = fg.read()
# Preprocessing
df['datetime'] = pd.to_datetime(df['datetime'])
df = df.sort_values('datetime')
# Feature engineering
df['pm10_lag1'] = df['pm10'].shift(1)
df['pm10_lag2'] = df['pm10'].shift(2)
df['pm10_avg3'] = df['pm10'].rolling(window=3).mean()
df = df.dropna()
features = ['pm10', 'pm2_5', 'pm10_lag1', 'pm10_lag2', 'pm10_avg3']
pred_input = df[features].iloc[-1]
# Load model
with open('RandomForest_model.pkl', 'rb') as f:
    model = pickle.load(f)
# Predict next 3 days
forecast = []
last_date = df['datetime'].max()
for i in range(1, 4):
    next_date = last_date + timedelta(days=i)
    pred_input['pm10'] += np.random.uniform(-5, 5)
    pred_input['pm2_5'] += np.random.uniform(-2, 2)
    pred_input['pm10_lag2'] = pred_input['pm10_lag1']
    pred_input['pm10_lag1'] = pred_input['pm10']
    pred_input['pm10_avg3'] = (pred_input['pm10'] + pred_input['pm10_lag1'] + pred_input['pm10_lag2']) / 3
    input_array = pred_input.to_numpy().reshape(1, -1)
    pred_aqi = model.predict(input_array)[0]
    forecast.append((next_date.strftime('%Y-%m-%d'), round(pred_aqi)))
forecast_df = pd.DataFrame(forecast, columns=['Date', 'Predicted AQI'])
forecast_df.to_csv('forecast.csv', index=False)
```

model.py

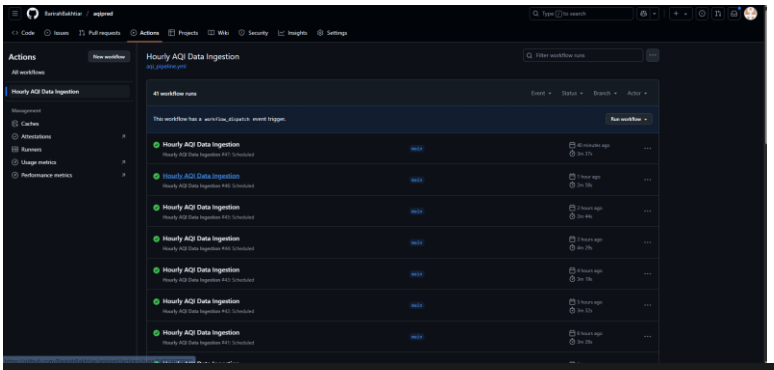
```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt
import hopsworks
import pickle
import numpy as np
from datetime import timedelta

st.set_page_config(page_title="AQI Prediction Dashboard", layout="wide")
st.title("Air Quality Forecast Dashboard")
# Connect to Hopsworks
project = hopsworks.login(
```

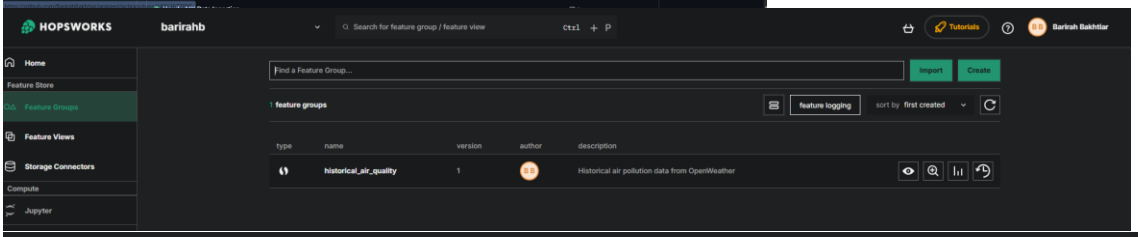
```
    project="barirahb",
    api_key_value="fiEMd5rPImQfpLA8.8qWqBnYd7YjnDY0oNriJquZFzcSHVAsBIkdiDGL8bAkRgBIWa8pye4cUjfVFHUAe
"
)
fs = project.get_feature_store()
fg = fs.get_feature_group("historical_air_quality", version=1)
df = fg.read()
# Preprocessing
df['datetime'] = pd.to_datetime(df['datetime'])
df = df.sort_values('datetime')
# Load model results
model_results = pd.read_csv('model_results.csv')
model_results['MSE'] = pd.to_numeric(model_results['MSE'], errors='coerce')
model_results['MAE'] = pd.to_numeric(model_results['MAE'], errors='coerce')
model_results['R2'] = pd.to_numeric(model_results['R2'], errors='coerce')
model_results.set_index('Model', inplace=True)
# Layout: 1 Single Page Dashboard
st.sidebar.title("Dashboard Sections")
section = st.sidebar.radio("Jump to", ["Overview", "Model Performance", "Forecasting"])
# --- Section 1: Overview ---
if section == "Overview":
    st.header("Latest Air Quality Data")
    st.dataframe(df.tail(10))
    st.header("AQI Trends")
    col1, col2 = st.columns([3, 1])
    with col1:
        fig, ax = plt.subplots(figsize=(6, 3))
        ax.plot(df['datetime'], df['aqi'], label='AQI', color='blue')
        ax.set_xlabel("Date")
        ax.set_ylabel("AQI")
        ax.set_title("Air Quality Index Over Time")
        ax.legend()
        plt.xticks(rotation=45)
        st.pyplot(fig)
        st.info("This graph shows the AQI trend over time.")
    st.header("AQI Distribution")
    col1, col2 = st.columns([3, 1])
    with col1:
        fig2, ax2 = plt.subplots(figsize=(6, 3))
        ax2.hist(df['aqi'], bins=30, color='skyblue', edgecolor='black')
        ax2.set_title('Distribution of AQI Values')
        ax2.set_xlabel('AQI')
        ax2.set_ylabel('Frequency')
        st.pyplot(fig2)

    st.header("PM10 vs PM2.5 Levels")
    col1, col2 = st.columns([3, 1])
    with col1:
        fig3, ax3 = plt.subplots(figsize=(6, 3))
        ax3.scatter(df['pm10'], df['pm2_5'], alpha=0.7)
```

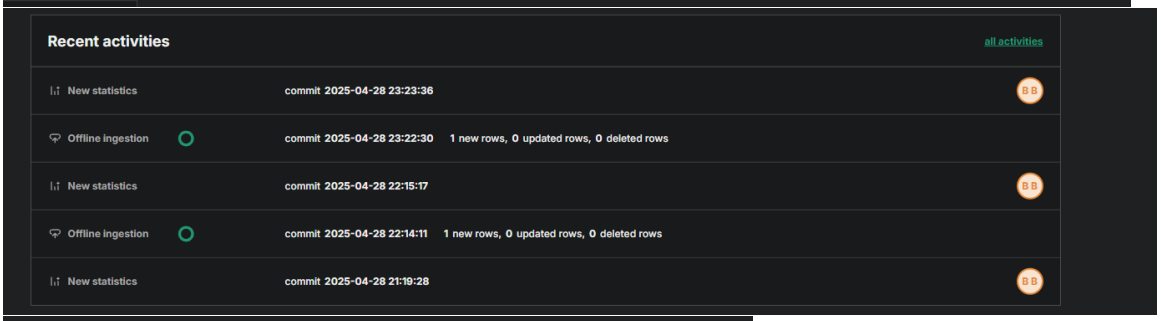
```
            ax3.set_xlabel("PM10")
            ax3.set_ylabel("PM2.5")
            ax3.set_title("PM10 vs PM2.5 Scatter Plot")
            st.pyplot(fig3)
        st.header("Pollutants Correlation Heatmap")
        col1, col2 = st.columns([3, 1])
        with col1:
            fig4, ax4 = plt.subplots(figsize=(6, 3))
            corr = df[['aqi', 'pm10', 'pm2_5', 'co', 'no', 'no2', 'o3', 'so2']].corr()
            sns.heatmap(corr, annot=True, cmap='coolwarm', ax=ax4)
            ax4.set_title("Correlation Heatmap")
            st.pyplot(fig4)
    # --- Section 2: Model Performance ---
    elif section == "Model Performance":
        st.header("Model Performance Metrics")
        st.dataframe(model_results)
        st.subheader("Detailed Metrics")
        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric(label="Best R² Model", value=model_results['R2'].idxmax(), delta=f"{model_results['R2'].max():.2f}")
        with col2:
            st.metric(label="Lowest MSE Model", value=model_results['MSE'].idxmin(), delta=f"{model_results['MSE'].min():.4f}")
        with col3:
            st.metric(label="Lowest MAE Model", value=model_results['MAE'].idxmin(), delta=f"{model_results['MAE'].min():.4f}")
    # --- Section 3: Forecast ---
    elif section == "Forecasting":
        st.header("3-Day AQI Forecast from All Models")
        latest = df.copy()
        latest['datetime'] = pd.to_datetime(latest['datetime'])
        latest = latest.sort_values('datetime')
        latest['pm10_lag1'] = latest['pm10'].shift(1)
        latest['pm10_lag2'] = latest['pm10'].shift(2)
        latest['pm10_avg3'] = latest['pm10'].rolling(window=3).mean()
        latest = latest.dropna()
        features = ['pm10', 'pm2_5', 'pm10_lag1', 'pm10_lag2', 'pm10_avg3']
        pred_input = latest[features].iloc[-1]
        forecast_results = []
        hazardous_detected = False
        models_to_forecast = ['RandomForest_model.pkl', 'XGBoost_model.pkl', 'Ridge_model.pkl']

        model_forecasts = {}
        for model_file in models_to_forecast:
            model_name = model_file.split('_')[0]
            with open(model_file, 'rb') as f:
                model = pickle.load(f)
            predictions = []
            last_date = latest['datetime'].max()
            temp_input = pred_input.copy()
            for i in range(1, 4):
                next_date = last_date + timedelta(days=i)
```

```
            temp_input['pm10'] += np.random.uniform(-8, 8)
            temp_input['pm2_5'] += np.random.uniform(-4, 4)
            temp_input['pm10_lag2'] = temp_input['pm10_lag1']
            temp_input['pm10_lag1'] = temp_input['pm10']
            temp_input['pm10_avg3'] = (temp_input['pm10'] + temp_input['pm10_lag1'] + temp_input['pm10_lag2']) / 3
            pred_input_arr = temp_input.to_numpy().reshape(1, -1)
            pred_aqi = model.predict(pred_input_arr)[0]
            pred_aqi_rounded = round(pred_aqi)
            if pred_aqi_rounded > 300:
                hazardous_detected = True
            predictions.append((next_date.strftime('%Y-%m-%d'), model_name, pred_aqi_rounded))
        forecast_results.extend(predictions)
        model_forecasts[model_name] = predictions
    forecast_df = pd.DataFrame(forecast_results, columns=['Date', 'Model', 'Predicted AQI'])
    st.dataframe(forecast_df)
    st.subheader("Forecast Comparison Across Models")
    forecast_chart = alt.Chart(forecast_df).mark_line(point=True).encode(
        x='Date:T',
        y='Predicted AQI:Q',
        color='Model:N'
    ).properties(
        width=800,
        height=400
    )
    st.altair_chart(forecast_chart)
    st.subheader("Individual Model Forecasts")
    for model_name, preds in model_forecasts.items():
        st.markdown(f"### {model_name} Forecast")
        model_df = pd.DataFrame(preds, columns=['Date', 'Model', 'Predicted AQI'])
        col1, col2 = st.columns([3, 1])
        with col1:
            fig_model, ax_model = plt.subplots(figsize=(6, 3))
            ax_model.plot(model_df['Date'], model_df['Predicted AQI'], marker='o')
            ax_model.set_xlabel('Date')
            ax_model.set_ylabel('Predicted AQI')
            ax_model.set_title(f'{model_name} Forecasted AQI')
            plt.xticks(rotation=45)
            st.pyplot(fig_model)
            st.info(f"Forecast generated using {model_name}.")
    if hazardous_detected:
        st.error("Hazardous AQI Level forecasted! Please take precautions.")
    else:
        st.success("No hazardous AQI levels predicted in the next 3 days.")
```

5. INTERFACES

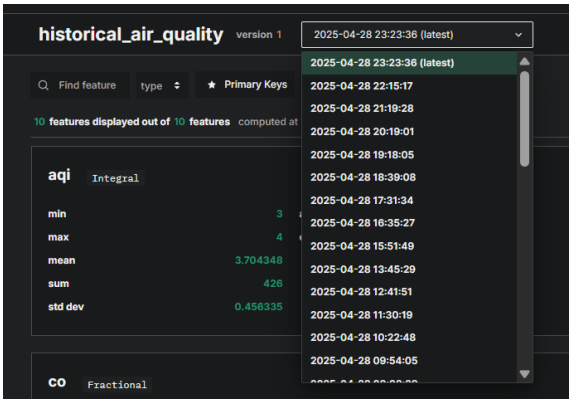


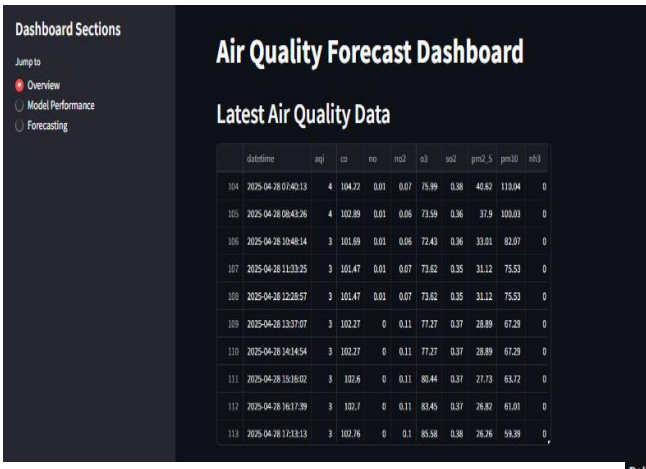


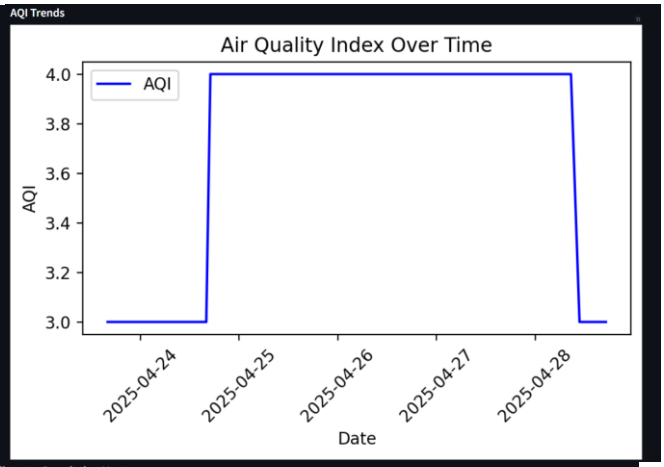


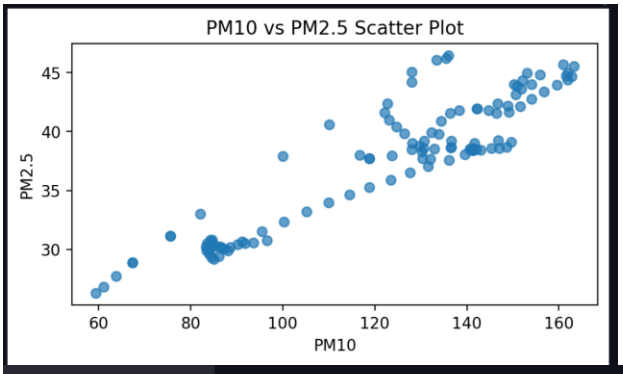
10 out of 10 features displayed - last 50 rows

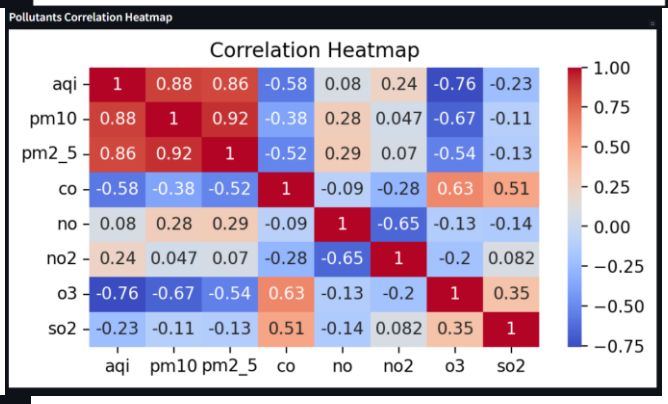
	aqi	co	datetime	nh3	no	no2	o3	pm10	pm2_5	so2
1	4	113.35	2025-04-25T06:00:00	0	0.01	0.07	61.82	140.73	38.53	0.37
2	4	113.71	2025-04-26T09:00:00	0	0.01	0.05	72.03	161.93	44.39	0.45
3	3	128.15	2025-04-23T16:00:00	0	0.0	0.09	92.04	96.54	30.73	0.41
4	4	115.12	2025-04-26T14:00:00	0	0.0	0.12	73.4	160.9	45.7	0.54
5	3	133.01	2025-04-24T03:00:00	0	0.0	0.09	100.31	83.99	30.23	0.43
6	4	113.02	2025-04-25T07:00:00	0	0.01	0.06	60.4	141.54	38.57	0.36
7	4	116.82	2025-04-24T19:00:00	0	0.0	0.11	70.28	109.98	33.95	0.43
8	4	117.81	2025-04-24T18:00:00	0	0.0	0.11	72.65	105.19	33.19	0.44
9	4	115.39	2025-04-24T23:00:00	0	0.0	0.1	65.23	127.69	36.5	0.4
10	4	115.65	2025-04-24T22:00:00	0	0.0	0.1	66.01	123.5	35.91	0.4
11	3	132.51	2025-04-23T20:00:00	0	0.0	0.08	93.9	88.56	30.17	0.45
12	3	133.37	2025-04-24T02:00:00	0	0.0	0.08	100.77	83.76	30.05	0.44
13	4	114.41	2025-04-26T13:00:00	0	0.01	0.09	71.56	163.25	45.55	0.52
14	4	112.03	2025-04-25T10:00:00	0	0.01	0.06	56.28	145.3	38.56	0.35
15	4	111.97	2025-04-25T11:00:00	0	0.01	0.06	55.78	147.09	38.57	0.36
16	3	132.55	2025-04-24T04:00:00	0	0.01	0.07	98.29	84.48	30.55	0.42
17	4	115.19	2025-04-26T15:00:00	0	0.0	0.13	72.98	153.11	44.96	0.51
18	4	116.18	2025-04-24T20:00:00	0	0.0	0.1	68.4	114.49	34.66	0.42

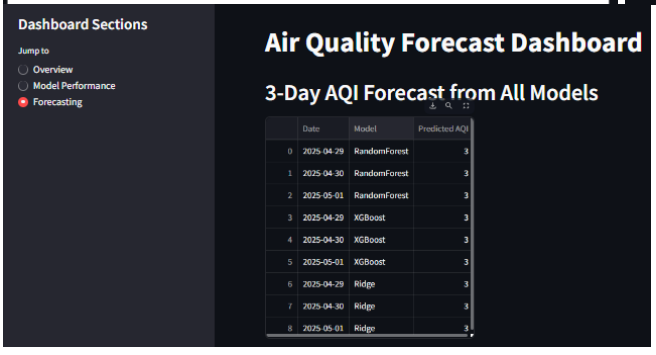

historical_air_quality
version 1
2025-04-28 23:23:36 (latest)
2025-04-28 23:23:36 (latest)
2025-04-28 22:15:17
2025-04-28 21:19:28
2025-04-28 20:19:01
2025-04-28 19:18:05
2025-04-28 18:39:08
2025-04-28 17:31:34
2025-04-28 16:35:27
2025-04-28 15:51:49
2025-04-28 13:45:29
2025-04-28 12:41:51
2025-04-28 11:30:19
2025-04-28 10:22:48
2025-04-28 09:54:05
Find feature
type
Primary Keys
10 features displayed out of 10 features computed at
aqi
Integral
min 3
max 4
mean 3.704348
sum 426
std dev 0.456335
co
Fractional


Dashboard Sections
Jump to
Overview
Model Performance
Forecasting
Air Quality Forecast Dashboard
Latest Air Quality Data

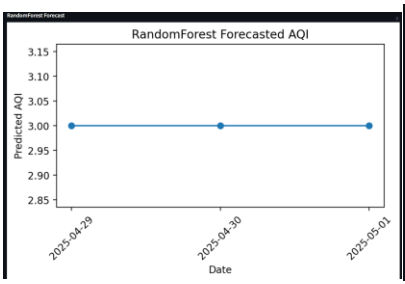

AQI Trends
Air Quality Index Over Time
AQI
4.0
3.8
3.6
3.4
3.2
3.0
2025-04-24
2025-04-25
2025-04-26
2025-04-27
2025-04-28
Date

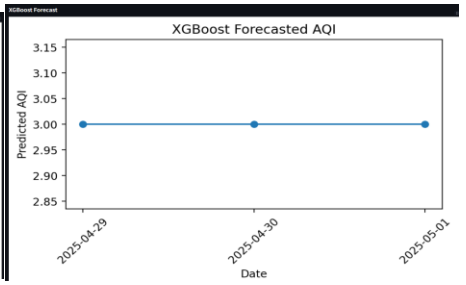

PM10 vs PM2.5 Scatter Plot
PM2.5
45
40
35
30
60
80
100
120
140
160
PM10

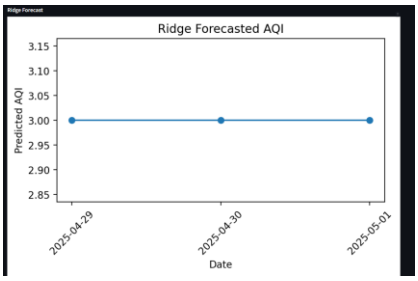

Pollutants Correlation Heatmap
Correlation Heatmap
aqi 1 0.88 0.86 -0.58 0.08 0.24 -0.76 -0.23
pm10 0.88 1 0.92 -0.38 0.28 0.047 -0.67 -0.11
pm2_5 0.86 0.92 1 -0.52 0.29 0.07 -0.54 -0.13
co -0.58 -0.38 -0.52 1 -0.09 -0.28 0.63 0.51
no 0.08 0.28 0.29 -0.09 1 -0.65 -0.13 -0.14
no2 0.24 0.047 0.07 -0.28 -0.65 1 -0.2 0.082
o3 -0.76 -0.67 -0.54 0.63 -0.13 -0.2 1 0.35
so2 -0.23 -0.11 -0.13 0.51 -0.14 0.082 0.35 1
aqi pm10 pm2_5 co no no2 o3 so2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

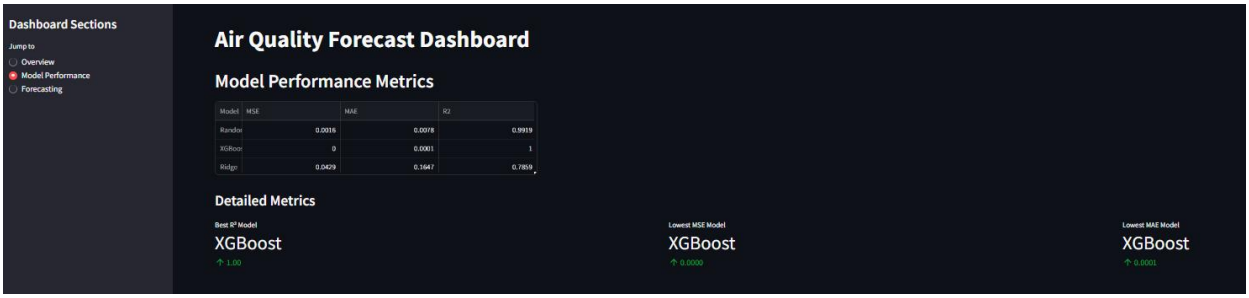

Dashboard Sections
Jump to
Overview
Model Performance
Forecasting
Air Quality Forecast Dashboard
3-Day AQI Forecast from All Models
Date
Model
Predicted AQI
0 2025-04-29 RandomForest 3
1 2025-04-30 RandomForest 3
2 2025-05-01 RandomForest 3
3 2025-04-29 XGBoost 3
4 2025-04-30 XGBoost 3
5 2025-05-01 XGBoost 3
6 2025-04-29 Ridge 3
7 2025-04-30 Ridge 3
8 2025-05-01 Ridge 3


Forecast Comparison Across Models
Predicted AQI
3.0
2.5
2.0
1.5
1.0
0.5
0.0
06 AM 09 AM 12 PM 03 PM 06 PM 09 PM Wed 30 03 AM 06 AM 09 AM 12 PM 03 PM 06 PM 09 PM May 03 AM
Date
Model
RandomForest
Ridge
XGBoost









6. CONCLUSION

The project demonstrates a professional-grade machine learning system for Air Quality Forecasting, integrating live data ingestion, feature engineering, model training, prediction, and deployment.

Strengths:

- Uses real-time and historical data.
- Integrates Hopsworks Feature Store for feature management and consistency.
- Trains multiple models and provides detailed evaluation metrics.
- Pipeline is containerized via Docker — ready for deployment anywhere.
- GitHub-based version control ensures reproducibility and team collaboration.

Best Practices Followed:

- Separation of data ingestion, feature creation, modeling, and prediction stages.
- Consistent feature tracking using a feature store.
- Clear model versioning and evaluation storage.
- Automation-friendly design.

Areas of Further Enhancement:

- Use Hyperparameter tuning (GridSearchCV / RandomSearchCV) for models.
- Add more sophisticated feature engineering (lag features, time-of-day, weather data).
- Implement model monitoring (drift detection, performance tracking).
- Explore deploying a streamlit dashboard for live AQI visualization.
- Implement a proper CI/CD pipeline with GitHub Actions for retraining.