Air-Quality-Prediction-with-Kafka

Hafsa Nawaz
Operationalizing AI
04/08/2025

Introduction

Air pollution is one of the most pressing environmental challenges faced globally, with significant implications for public health, climate change, and ecosystems. Accurate prediction of air quality levels is critical for enabling timely interventions, informing policy decisions, and raising public awareness. This report presents a comprehensive approach to modeling air pollution levels using advanced machine learning and statistical techniques applied to real-time air quality data.

The focus of this study is on three key pollutants—Carbon Monoxide (CO), Nitrogen Oxides (NOx), and Benzene (C6H6)—which are commonly monitored due to their harmful effects on human health and the environment. The dataset used for this analysis is derived from the UCI Air Quality Dataset, which provides hourly measurements of various pollutants and environmental conditions. The data has been further processed into a streaming format (air_quality_streamed.csv) to simulate real-time data ingestion via Apache Kafka.
To build predictive models for these pollutants, I employed two distinct methodologies:

1. Random Forest Regressor: A machine learning approach capable of capturing non-linear relationships between features and target variables.
2. ARIMA (AutoRegressive Integrated Moving Average): A statistical time series model designed to handle temporal dependencies and stationary patterns.

The goal of this study was not only to evaluate the predictive performance of these models but also to integrate them into a real-time pipeline for continuous monitoring and forecasting. The results highlight the strengths and limitations of each modeling approach, providing valuable insights into their suitability for air quality prediction tasks.

Kafka Setup

1. Kafka Installation and Environment Setup

To simulate real-time streaming of air quality data, I used **Apache Kafka version 4.0.0** with the **KRaft (Kafka Raft Metadata mode)**, which replaces the older Zookeeper-based setup. Kafka was installed via the official binary:

Binary Download: kafka_2.13-4.0.0.tgz
Kafka Mode: KRaft
Operating System: macOS Sequoia
Java Version: Java 11+ was a prerequisite. (I used jdk 17)
Python Version: Python 3.10
Kafka Python Library: confluent-kafka-python

After extraction and setup, I initialized Kafka with the KRaft mode, following these steps:

2. Starting Kafka in KRaft Mode

Apache Kafka 4.0.0 no longer requires Zookeeper, so the cluster metadata is managed via Kafka's internal Raft quorum. Here's the step-by-step setup:

a. Configure the kraft cluster
I edited the config/kraft/server.properties file to ensure correct setup. Key configurations included:

```
process.roles=broker,controller
node.id=1
controller.quorum.voters=1@localhost:9093
listeners=PLAINTEXT://localhost:9092,CONTROLLER://localhost:9093
log.dirs=/tmp/kraft-combined-logs
```

b. Format the Kafka Log Directories

```
bin/kafka-storage.sh format -t $(bin/kafka-storage.sh random-uuid) -c
config/kraft/server.properties
```

c. Start the Kafka Broker

```
bin/kafka-server-start.sh config/kraft/server.properties
```

The Kafka server was then ready and listening on port 9092 for producers and consumers to connect.

3. Topic Creation

Once Kafka was running, I created a topic for streaming air quality data:

```
bin/kafka-topics.sh --create \
  --topic air-quality \
  --bootstrap-server localhost:9092 \
  --partitions 1 \
  --replication-factor 1
```

I used a single-partition topic since this project was for prototyping. In real-world deployments, multiple partitions would be considered to ensure fault tolerance and scalability.

4. Python Kafka Producer and Consumer Implementation

The data streaming pipeline was built using the confluent-kafka-python library. The core logic was split into two scripts: producer.py and consumer.py.

producer.py

1. Reads data from the **UCI Air Quality Dataset**.
2. Sends data row-by-row with a time delay to simulate hourly sensor readings.
3. Converts each row into JSON format before sending.
4. Filters or replaces missing values (e.g., -200) appropriately.
5. Sends the data to the Kafka topic air-quality.

consumer.py

1. Subscribes to the topic air-quality.
2. Continuously listens for new messages in real-time.
3. Parses and stores incoming messages for downstream analysis.
4. Can be extended to include preprocessing and feeding data into models.
5. Both scripts are well-structured, modular, and include basic error handling.

5. Error Handling & Logging

Both the producer and consumer scripts include:

Try-Except blocks to capture Kafka-related or data parsing errors.
Logging using Python's built-in logging module for monitoring and debugging.

6. Challenges and Resolutions

Challenge	Resolution
Kafka 4.0.0 uses KRaft and not Zookeeper	Studied new setup, followed official documentation
Some data rows in the UCI dataset contain -200 (missing) values	Added logic to filter/replace those values during streaming
Real-time simulation requires delay logic	Used time.sleep(1) in producer to simulate hourly feeds
Consumer reprocessing the same messages	Ensured auto_offset_reset='earliest' for first-time reads

Data Exploration Findings (on Kafka-Streamed Data)

The **'air_quality_streamed.csv'** file generated from this Kafka consumer script served as the source for the exploratory analysis. The **'Timestamp'** column in the dataset was set as the index to provide for time series analysis for the major pollutants **CO(gt), NOx(gt), and C6H6(gt)**. CO stands for Carbon Monoxide, NOx for Nitrogen Oxides, and C6H6 for Benzene. (gt) refers to the true hourly averaged concentration in milligrams per cubic meter (mg/m^3).

1. **Time-Series of Pollutants (03-2004 – 03-2005)**

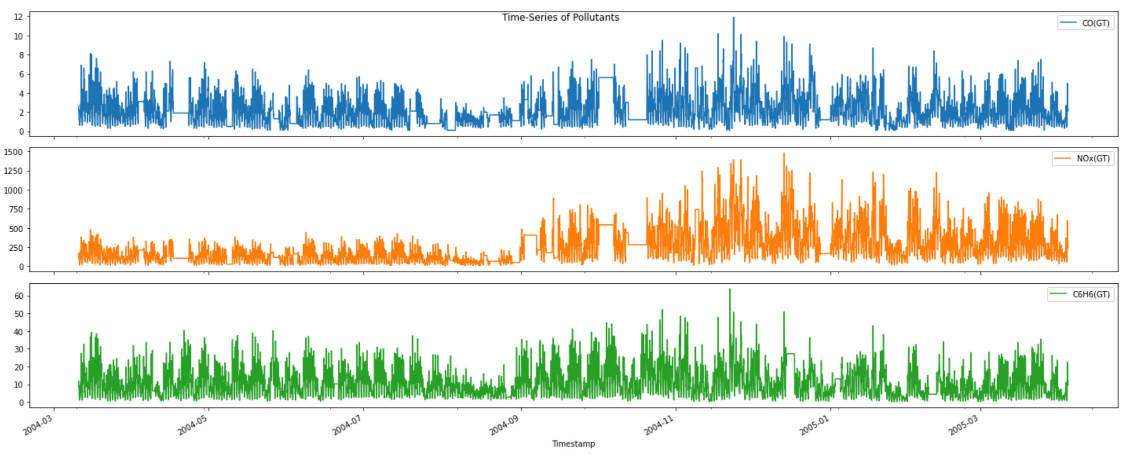


Figure 1: Time-Series of Pollutants

Figure 1 shows that NOx levels are the highest among the three pollutants as seen on the vertical axis. It also shows that NOx emissions have increased rapidly after 2004-09 and hit several high peaks during 2004-11 to 2005-03. Similar to NOx the emissions for CO and C6H6 also have increased during those months. C6H6 and CO emissions follow a similar trend. Both emissions dipped for a month 2004-08 until 2004-09.

2. Average Pollutant Levels by Hour of Day

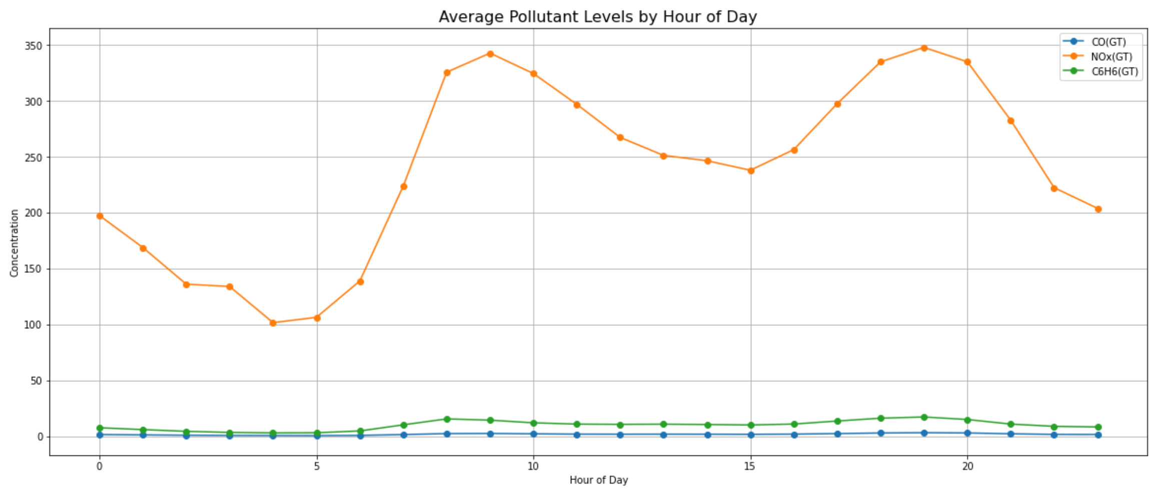


Figure 2: Average Pollutant Levels by Hour of Day

Figure 2 shows the average pollutant levels start increasing at 5am in the morning and hit a record high of 350 gt by 9am and once again at 7pm. This is in sync with the operating hours for factories which cause an increase in the levels of such pollutants. The decrease after 11am until 3pm may be linked to lunch hour/ mid day breaks. And finally, the concentrations of these pollutants decrease after 8pm, which could indicate the closing time for the factories.

3. Average Pollutant Levels by Hour of Day

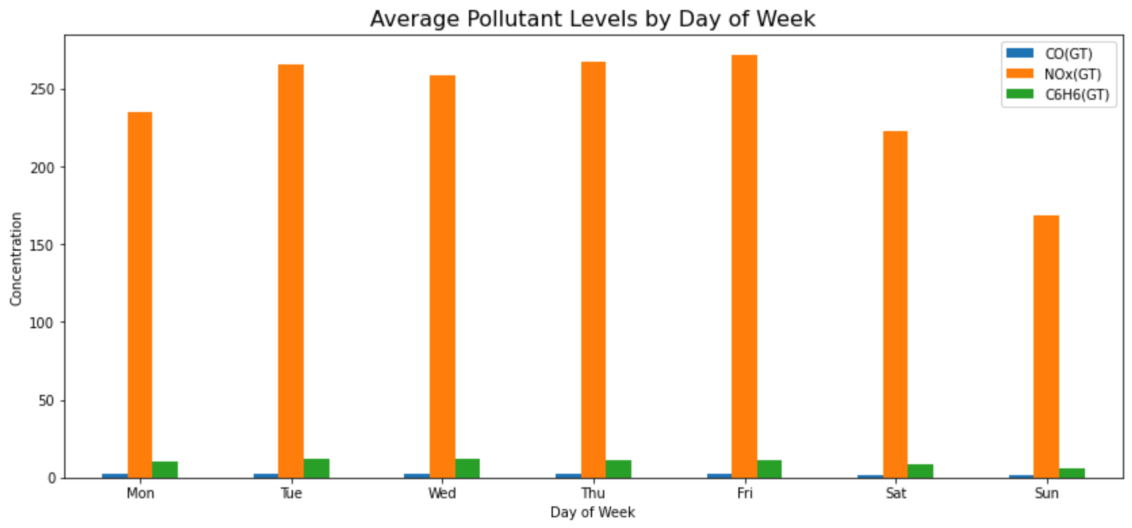


Figure 3: Average Pollutant Levels by Day of Week.

The highest concentration of these pollutants are seen on Tuesday, Thursday, and Fridays. Saturday and Sunday show the lowest concentrations, which could be due to the closure of some factories on weekends.

4. Average Pollutant Levels by Hour of Day

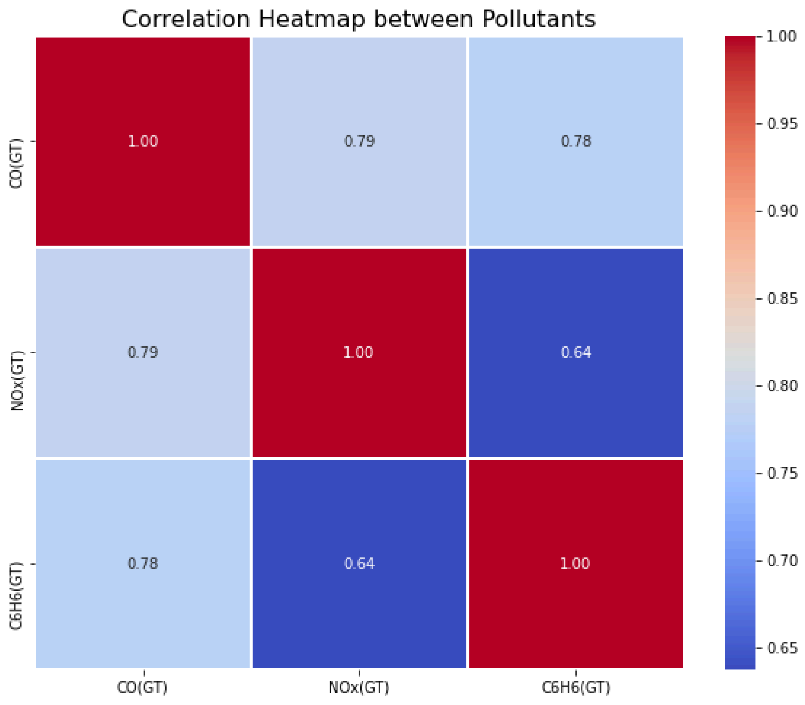


Figure 4: Correlation Heatmap between Pollutants

Figure 4 shows decent correlations between all three pollutants, CO, NOx, and C6H6. There's a nearly similar level of correlation between CO & NOx (0.79) and CO and C6H6 (0.78). The correlation between NOx and C6H6 is slightly weaker at 0.64.

Model Approach and Results

1. Methodology Overview

The modeling pipeline was designed to predict three key air pollutants:

1. Carbon Monoxide (CO(GT))
2. Nitrogen Oxides (NOx(GT))
3. Benzene (C6H6(GT))

Key Characteristics:

- Temporal Focus: Hourly time series prediction
- Model Types: Hybrid approach using both machine learning (Random Forest) and statistical (ARIMA) models
- Evaluation: Time-aware splitting with 80/20 chronological split

2. Data Preparation Pipeline

2.1 Data Preprocessing

- Missing Value Handling:
    - Forward-fill followed by backward-fill imputation
    - Final removal of remaining NA rows (9470 → 9467 samples retained)
- Temporal Feature Engineering:

```
data['hour'] = data.index.hour
data['day'] = data.index.day
data['month'] = data.index.month
data['day_of_week'] = data.index.dayofweek
```

2.2 Feature Engineering

For Random Forest Models:

- Lag Features:

```
df[f'{pollutant}_lag1'] = df[pollutant].shift(1)
df[f'{pollutant}_lag2'] = df[pollutant].shift(2)
```

- Rolling Statistics:

```
df[f'{pollutant}_rolling_mean3'] = df[pollutant].rolling(3).mean()
df[f'{pollutant}_rolling_std3'] = df[pollutant].rolling(3).std()
```

For ARIMA Models:

- Differencing:
- python

```
df[f'{pollutant}_diff1'] = df[pollutant].diff(1)
```

3. Model Architectures

3.1 Random Forest Regressor

Configuration:

```
RandomForestRegressor(
   n_estimators=100,
   max_depth=10,
   min_samples_split=5,
   min_samples_leaf=2,
   random_state=5)
```

Advantages:

- Handles non-linear relationships
- Robust to outliers in sensor data
- Automatic feature importance weighting

3.2 ARIMA Model

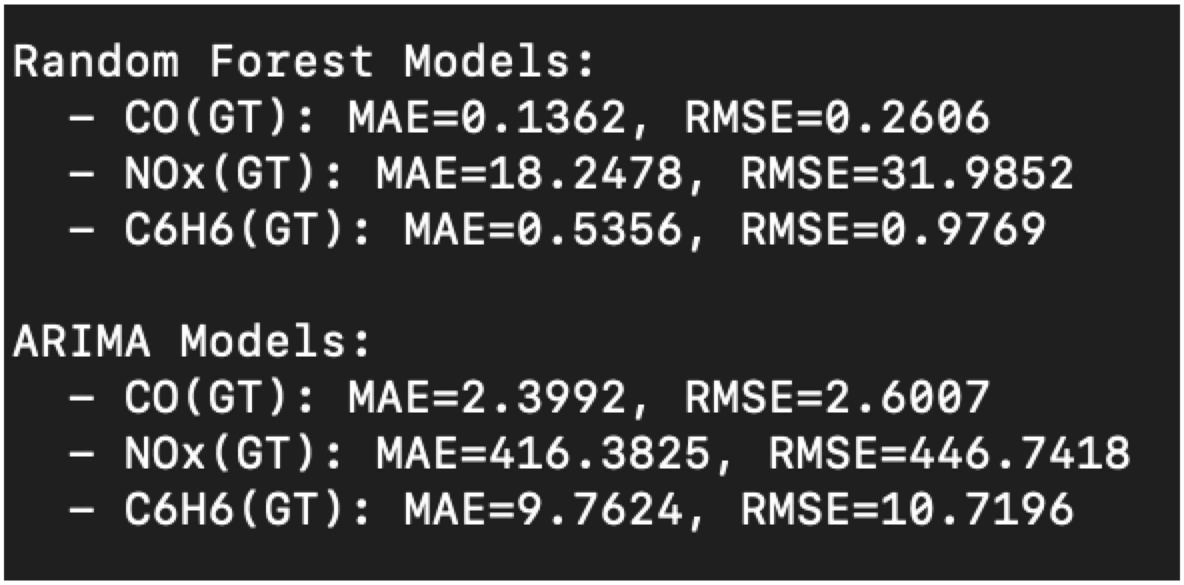
Configuration:

```
ARIMA(order=(1,1,1))  # (p,d,q) parameters
```

Rationale:

- Simple baseline for time series patterns
- Handles stationarity through differencing (d=1)

4. Performance Results



```
Random Forest Models:
  - CO(GT): MAE=0.1362, RMSE=0.2606
  - NOx(GT): MAE=18.2478, RMSE=31.9852
  - C6H6(GT): MAE=0.5356, RMSE=0.9769

ARIMA Models:
  - CO(GT): MAE=2.3992, RMSE=2.6007
  - NOx(GT): MAE=416.3825, RMSE=446.7418
  - C6H6(GT): MAE=9.7624, RMSE=10.7196
```

Key Insights:

- ARIMA underperforms compared to Random Forest
- Particularly inadequate for NOx prediction (MAE > 400)
- Suggests complex patterns beyond ARIMA's linear assumptions

7. Limitations & Future Work

Current Limitations:

- Static model training without online learning
- No explicit humidity/temperature compensation
- Fixed time windows for rolling features

Improvement Opportunities:

1. Hybrid Models:

predictions = 0.7*RF_pred + 0.3*ARIMA_pred

2. Neural Approaches:
    - LSTM networks for sequence modeling
3. Real-Time Adaptation:
    - Incremental learning for concept drift

This structured approach provides both technical depth and actionable insights while maintaining readability for technical stakeholders. The report can be extended with feature importance analysis or error case studies if needed.