**Real-Time Air Quality Prediction with Kafka**

*94-879: Fundamentals of Operationalizing AI*

Amy Kang

*Executive Summary*

As one of the most pressing environmental challenges globally, real-time data streaming and prediction of air quality can be critical to mitigating its impact and informing timely public health interventions. This project presents a real-time air quality forecasting pipeline using Apache Kafka and machine learning, based on the [UCI Air Quality dataset](https://archive.ics.uci.edu/dataset/360/air+quality). The goal is to simulate a continuous streaming environment where air quality data is ingested, processed, and used for immediate predictions.

The UCI dataset includes hourly measurements of key air pollutants such as CO, NOx, and Benzene, along with data on temperature and humidity. A comprehensive data preprocessing phase was taken to prepare the dataset for modeling. This included cleaning missing values, removing unreliable segments, and creating meaningful time-based and statistical features such as hourly, daily, and weekly lags and rolling means. Exploratory Data Analysis (EDA) revealed distinct temporal patterns: pollutant levels spiked during morning and evening rush hours and were higher on weekdays than weekends, suggesting traffic as a major source. Seasonality was also evident, with winter months showing elevated pollution levels.

To capture these patterns, a Random Forest model was trained using engineered features. This model was selected for its ability to handle non-linear relationships and multivariate dependencies, which were prominent in the dataset. Once trained, the model was integrated into a Kafka consumer script that processes incoming data in real-time, applies feature transformations, and outputs predictions on air quality levels.

The system architecture includes a Kafka producer that streams cleaned air quality records and a consumer that logs predictions and stores them for further analysis. This setup mimics real-world pipelines and demonstrates how streaming frameworks can be paired with machine learning to achieve dynamic environmental monitoring. Despite initial challenges in configuring Kafka on Windows, the final implementation supports message flow and real-time inference, laying the foundation for scalable air quality systems.

*Kafka Setup & Integration*

Setting up Apache Kafka for real-time air quality prediction presented several technical issues, especially on a Windows-based development environment. Kafka provides its startup scripts in .bat format, designed primarily for command-line execution. However, these scripts are not always compatible with Windows PowerShell, which introduced a range of challenges during configuration and testing.

Initially, I attempted to start Kafka’s core services (ZooKeeper and the Kafka broker) via PowerShell using the standard startup scripts (zookeeper-server-start.bat and kafka-server-start.bat). PowerShell, however, did not recognize the .bat scripts unless they were explicitly prefixed with a backward slash (.\). This minor but crucial syntax issue delayed progress early on. Furthermore, Kafka’s deep installation path within my system caused additional problems. For example, the Windows command-line interface has a limit on the length of a command line, and the deeply nested Kafka path triggered “input line is too long” errors when executing scripts. I relocated my entire Kafka installation to a simplified directory: C:\kafka. This adjustment resolved some path-related issues and allowed for smoother startup of ZooKeeper and Kafka.

With the environment corrected, I ran ZooKeeper and the Kafka broker as separate processes in individual PowerShell windows. This separation allowed me to monitor logs in real time and independently control the services. Once both services were running in the background, I created a Kafka topic (**air-qual-test**) with three partitions for message streaming using the built-in kafka-topics.bat utility.

With Kafka running and the topic configured, I moved on to building and connecting the data pipeline through custom Python scripts for a producer (airqual-producer.py) and consumer (airqual-consumer.py). The producer script reads from a cleaned CSV file containing historical air quality records and sends the last 20% as JSON messages to the Kafka topic (the first 80% of the dataset is used to train models; see Modeling Approach & Results). Each row is converted into a dictionary and streamed into Kafka using a loop. I initially included a time.sleep() command to simulate hourly streaming but commented it out during testing for faster throughput. The script is equipped with logging statements that record each successful message dispatch and any exceptions encountered.

On the consumer side, the setup is more complex. The airqual-consumer.py script initializes a Kafka consumer that listens for messages from the “air-qual-test” topic. Upon receiving a message, it logs the raw input, processes the data into a DataFrame, generates relevant time-based features (Hour, Day, Month, Year), and reorders the columns based on a predefined feature set. These transformations align the incoming data with the format expected by the machine learning model. See Modeling Approach & Results for more on model integration in the consumer script.

*Data Exploration Findings*

A graph showing a graph

AI-generated content may be incorrect.Figure 1

A graph showing a number of blue lines

AI-generated content may be incorrect.A graph showing a number of blue lines

AI-generated content may be incorrect.To understand the behavior of air pollutants collected through the Kafka streaming pipeline, I performed EDA on temporal patterns and relationships between different pollutants. The time-series plots in figure 1 indicate spiked concentration levels during the winter months (around October to February) and lower levels during the summer months (around July to August). NOx levels are also more than half less than CO and Benzene concentrations from March to August 2004.

When I zoom in by hour, figure 2 on hourly patterns reveal two clear peaks across all pollutant types – around 8AM and 6PM – which could correspond to typical city rush hours, indicating traffic is a major contributor to pollution levels. Further, figure 2 on daily patterns showcase higher concentrations from Tuesdays to Fridays across all pollutant types, with a significant dip on Saturdays and Sundays – reinforcing the traffic observation.

Figure 2

A graph with lines and numbers

AI-generated content may be incorrect.A graph with lines and numbers

AI-generated content may be incorrect.When examining the correlation between pollutants, figure 3 highlights strong positive relationships among all three. CO and Benzene have a particularly high correlation at 0.93; NOx is also positively correlated with CO and Benzene, though slightly less at 0.8 and 0.73, respectively. These relationships suggest that the pollutants are not independent.

A red squares with white text

AI-generated content may be incorrect.Figure 3

Through my EDA, some of the variations in figures 1 and 2 can be explained by city traffic patterns and workweek cycles, seasonality, and potentially temperature and humidity. These combined patterns suggest the dynamics between human activity and environmental conditions in shaping air quality. The importance of time and seasonality trends in this analysis indicates that time-based features, in addition to lagged features and rolling statistics, will be necessary to predict pollution concentrations. Features such as hour of day, day of week, and month can help capture cyclical behavior linked to traffic and human activity. Lagged variables will allow the model to account for short-term persistence in pollutant levels, while rolling means and standard deviations can help smooth out fluctuations and detect longer-term trends or anomalies.

Given the strong correlations observed between pollutants (e.g., CO and Benzene), multivariate models that leverage the relationships between variables are likely to perform better than treating each pollutant independently. Additionally, the presence of seasonal patterns suggests that time-series-aware models, such as ARIMA or SARIMA, may be more appropriate for capturing dynamic dependencies over time.

*Modeling Approach & Results*

To prepare the air quality dataset for time series modeling, I engineered features that capture both temporal patterns and historical trends. I extracted time-based features including Hour, Day, Month, and Year to help models learn daily and seasonal cycles. To incorporate temporal dependencies, I created lagged features for CO, NOx, and Benzene concentrations at 1-hour, 24-hour (daily), and 168-hour (weekly) intervals. Additionally, I calculated rolling statistics — mean and standard deviation over 3-hour, 24-hour, and 168-hour windows — to represent recent trends and variability. I combined all features into a “feature\_cols” list, providing temporal context and pollutant history to enhance model performance in forecasting tasks.

To train various models, I used the train-test split approach, with 80% of the cleaned dataset for training and 20% for testing. I set *x* as my feature columns from before and *y* as the target variables, CO, NOx, and Benzene. Given their strong correlations found from EDA, I decided to create multivariate models, if applicable. See below for various model results:

* *Baseline model*: I ran a simple multivariate linear regression model with time-based, lagged features. RMSE: 0.57; MAE: 0.40.
* *Random Forest model*: I trained a multivariate Random Forest model using MultiOutputRegressor (n\_estimators = 100), delivering the best performance. RMSE: 0.36; MAE: 0.22. This model was saved and integrated with Kafka.
* *SARIMA model (co\_gt only)*: I applied a univariate SARIMA model to forecast CO levels, performing worse than other models. RMSE: 1.23; MAE: 0.91.

To evaluate model performance, I used RMSE and MAE. RMSE penalizes larger errors more heavily, making it useful for identifying models that consistently stay close to true values, while MAE gives a straightforward average of prediction errors, offering interpretability in the original units. The baseline linear regression model achieved an RMSE of 0.57 and MAE of 0.40, setting a benchmark for comparison. The Random Forest model significantly outperformed it, with an RMSE of 0.36 and MAE of 0.22, indicating better accuracy and tighter error margins across all predicted pollutants. In contrast, the SARIMA model, which could only be applied to the CO series due to its univariate limitation, performed poorly with an RMSE of 1.23 and MAE of 0.91, likely due to its inability to leverage multivariate relationships or complex temporal patterns. Given its above-baseline performance, support for multivariate outputs, and robustness in handling feature interactions, Random Forest was the choice for integration with the Kafka consumer in the real-time prediction pipeline – rejecting my initial hypothesis that SARIMA may be a better performing model.

For real-time prediction, I integrated the trained Random Forest model into the Kafka-based data pipeline. The setup begins with the Kafka consumer script, where processed data is passed to the Random Forest model for prediction. After each prediction is generated, the result is combined with the original input data and stored in a list of records. These records are continuously written to a CSV file (air\_quality\_predictions.csv) using pandas. This happens inside the message loop, so the file is updated in real time as new messages are consumed from Kafka. Each row in the CSV includes the original air quality data along with the predicted values, allowing for easy tracking, evaluation, and future analysis. This Kafka- machine learning model integration allows for scalable air quality forecasting in a live data environment.

*Implications & Limitations*

This project demonstrates how real-time air quality prediction can be operationalized through a data streaming pipeline using Apache Kafka. In real-life deployment, air quality sensors could stream data directly into a Kafka topic, where the consumer script would process and predict pollution levels in near real time. My infrastructure supports continuous monitoring and could be extended to trigger alerts, inform dashboards, or assist policy decisions. This approach allows for scalable, automated, and responsive environmental monitoring, making it well-suited for smart city applications or industrial pollution tracking.

Despite the functional prototype, several limitations exist. First, Kafka's setup and the file paths in the current code are tightly coupled to a local Windows environment, which may cause compatibility issues when deploying to other systems without modification. Second, while the current setup handles a fixed volume of data smoothly, Kafka’s performance may degrade with significantly larger message loads, especially without proper partitioning, buffering, or fault tolerance mechanisms. Lastly, the consumer script is designed for a short test window and may not be robust enough for long-term, high-volume streaming, lacking error recovery, scalability features, and optimization for memory or performance under sustained load.

*Conclusion*

This project demonstrated a real-time air quality prediction system by integrating a Random Forest model with Apache Kafka. The pipeline processes streaming data, generates predictions, and logs results efficiently. While the system performs well in a test environment, future work should focus on improving portability, scalability, and robustness for real-world deployment.