Phase 1: Enhanced Real-Time Air Quality Monitoring System

94-879: Fundamentals of Operationalizing AI

Amy Kang | Helom Berhane | Shaurya Gulati | Trevor Foster

Introduction

This interim report presents the process, methodology, and results of a machine learning project focused on real-time air quality prediction via data streaming. The assignment centers around leveraging Apache Kafka to simulate and process hourly air quality readings in real-time. The primary goal was to build a system capable of analyzing environmental data streams and forecasting pollutant concentrations to support public health and environmental decision-making.

As a group, we have completed the following objectives in phase 1:

1. Real-time data streaming: Use Kafka producers and consumers to simulate real-time data streaming of hourly air quality metrics.
2. Exploratory Data Analysis (EDA): Analyze temporal trends, seasonal patterns, and interrelationships between various pollutant concentrations through visuals such as heatmaps and line graphs.
3. Predictive modeling: Develop and evaluate machine learning models to forecast pollutant levels—particularly the Carbon Monoxide concentration (CO(GT))—based on real-time streamed data. Model performance was assessed using Mean Absolute Error (MSE), Root Mean Squared Error (RMSE), and R-Squared.
4. ML experiment tracking: Integrate MLFlow into our data pipeline to track experiments, log metrics, record model parameters, and save artifacts for deployment.

This report details the system architecture for real-time data streaming, the machine learning models implemented, and the experimental results. It also outlines the project timeline and the next steps planned to enhance model performance and real-time deployment capabilities.

Dataset Overview

The dataset used in this project originates from the UCI Machine Learning Repository. It contains 9,358 instances of hourly averaged responses from five metal

oxide chemical sensors. The sensors were embedded in a chemical multi-sensor device deployed at road level in a highly polluted area within an unnamed Italian city. Data collection spanned from March 2004 to February 2005, providing a comprehensive view of pollutant variations over an entire year.

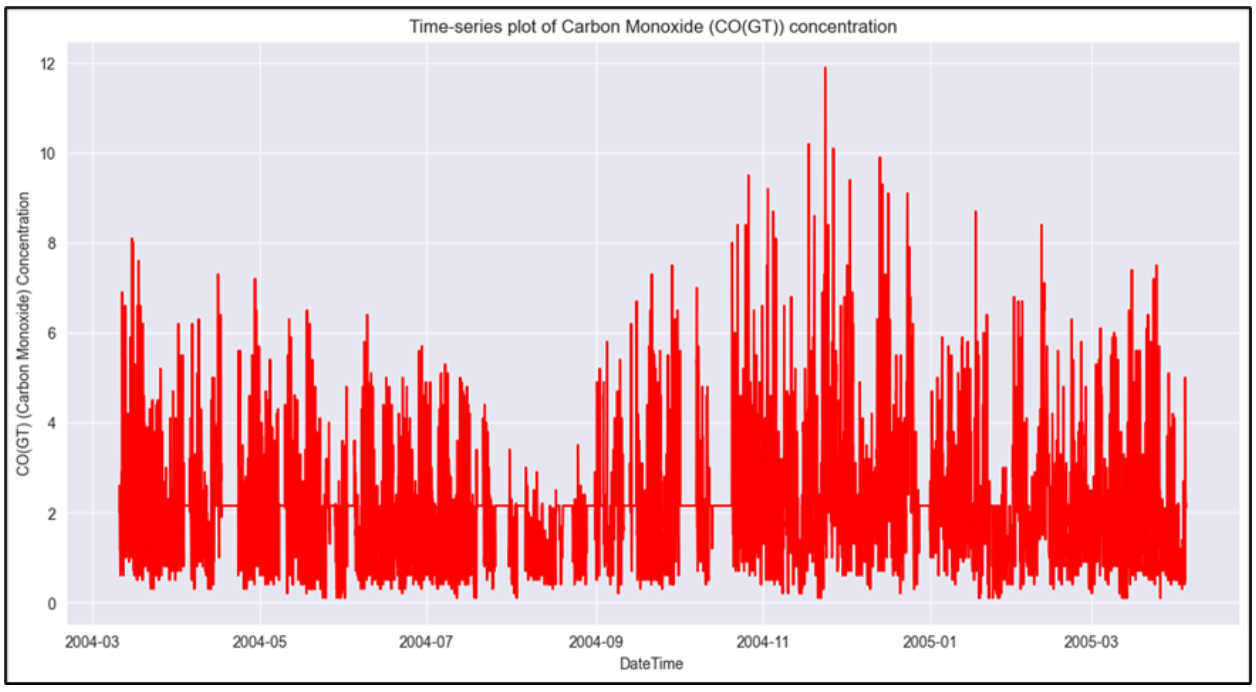
This dataset includes 15 features:

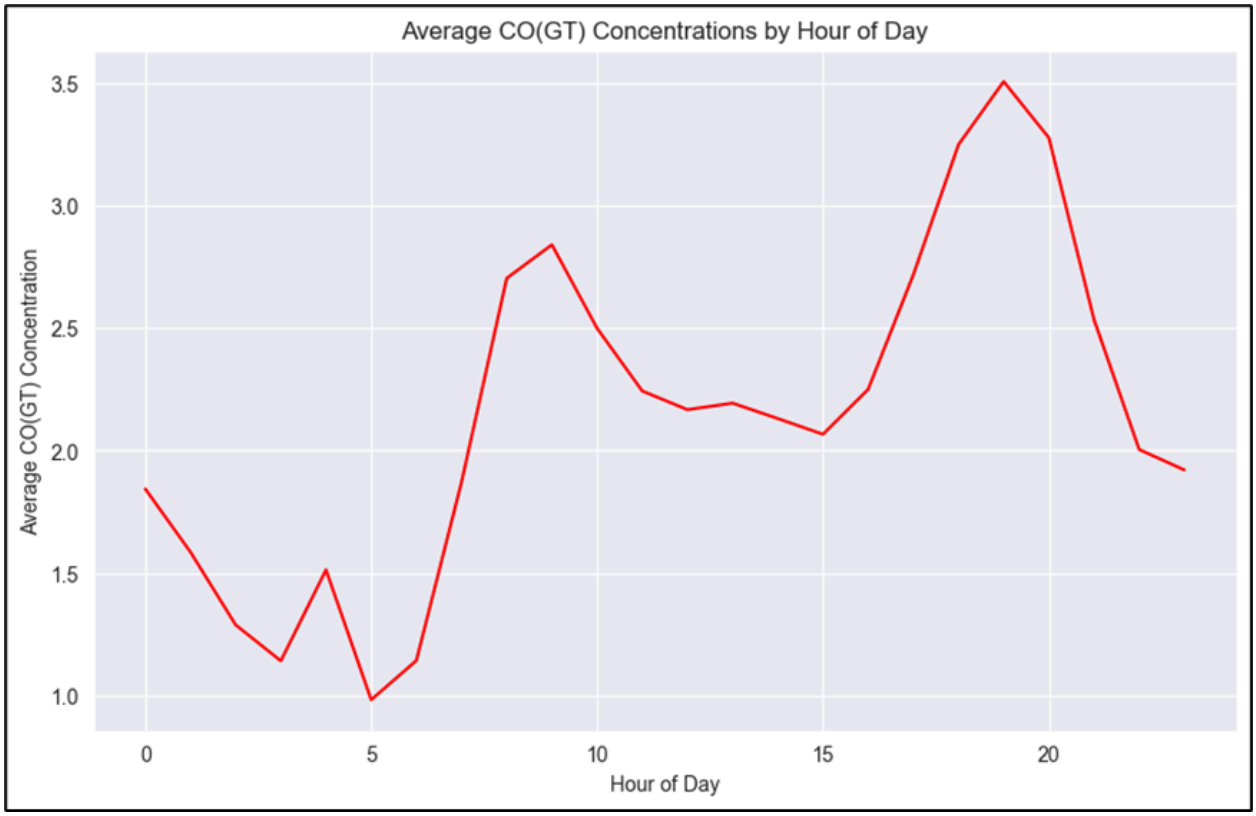
Feature	***Description***
Data	Date of record
Time	Time of record
CO(GT)	Ground truth hourly averaged concentration of Carbon Monoxide in mg/m³ **(*target variable*)**
PT08.S1 (CO)	Sensor 1 (Tin Oxide) – related to CO concentration
NMHC(GT)	Non-Methane Hydrocarbons in microg/m³
C6H6(GT)	Benzene concentration in microg/m³
NOx(GT)	Nitrogen Oxides concentration in ppb
NO2(GT)	Nitrogen Dioxide concentration in microg/m³
PT08.S2 (NMHC)	Sensor 2 (Manganese Oxide) – related to NMHC
PT08.S3 (NOx)	Sensor 3 (Tungsten Oxide) – related to NOx
PT08.S4 (NO2)	Sensor 4 (Indium Oxide) – related to NO2
PT08.S5 (O3)	Sensor 5 (Indium Oxide) – related to Ozone
T	Temperature measured in Celsius
RH	Relative humidity
AH	Absolute Humidity

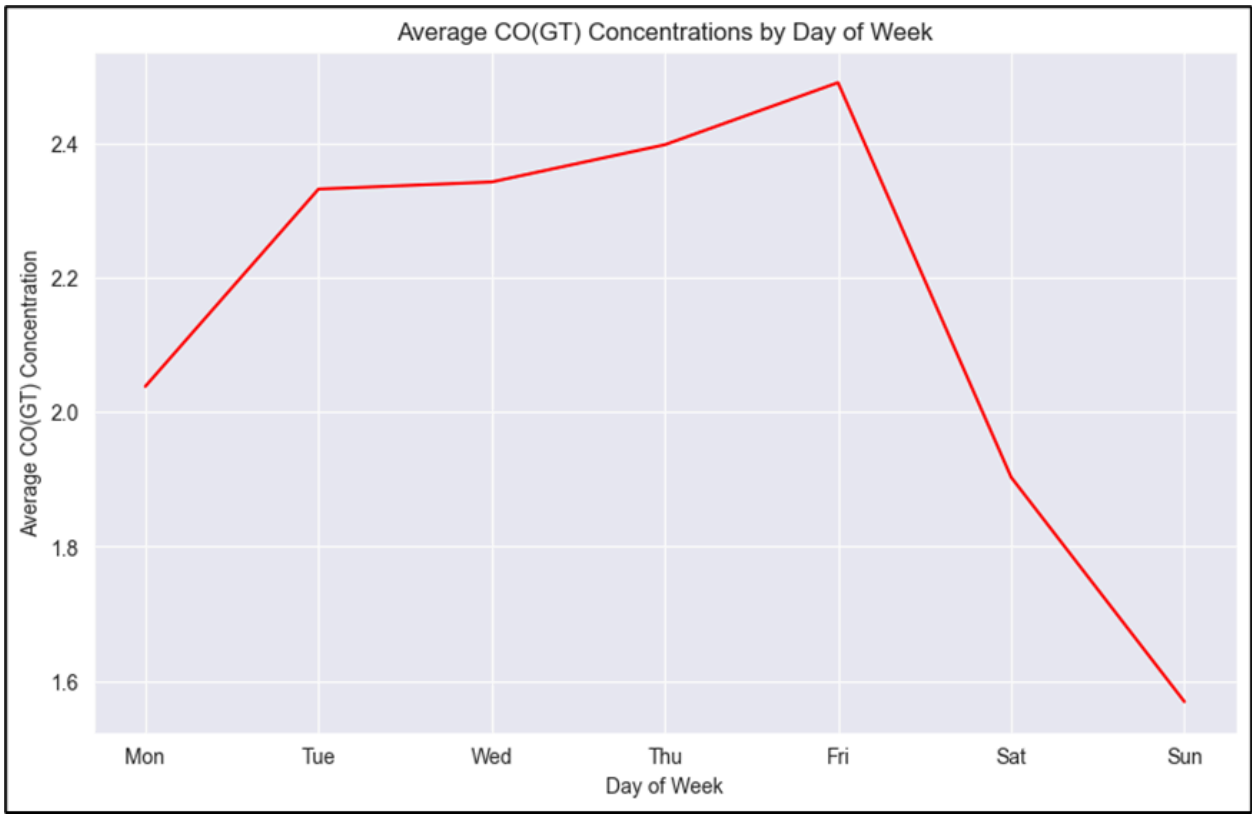
The raw dataset contained missing values across multiple sensor measurement columns, requiring a comprehensive data cleaning process prior to model development. Placeholder values such as -200, used in the original dataset to denote missing observations, were systematically converted to np.nan. To restore continuity in time series measurements, missing values were imputed using linear interpolation. Additionally, the Date and Time columns were also merged into a single datetime object for efficient indexing and time-based feature extraction. This preprocessing stage ensured that the dataset was more robust for temporal modeling.

Exploratory Data Analysis (EDA)

Our EDA of CO(GT) revealed spiked concentration levels during the winter months and lower levels during the summer months. When zooming in by hour, there were two clear peaks of CO(GT) levels around 8AM and 7PM, corresponding to typical rush hours and indicating traffic as a clear contributor to pollution levels. Further, daily patterns showcase higher concentrations from Tuesdays to Fridays, with a significant dip on Saturdays and Sundays – reinforcing the traffic observation. This step indicated that time-based features, such as hour, day, and month, will be necessary to predict future CO(GT) concentrations.




Average CO(GT) Concentrations by Hour of Day
Average CO(GT) Concentration
3.5
3.0
2.5
2.0
1.5
1.0
0
5
10
15
20
Hour of Day


Average CO(GT) Concentrations by Day of Week
Average CO(GT) Concentration
2.4
2.2
2.0
1.8
1.6
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Day of Week

Feature Engineering

To optimize the effectiveness of our model, we made a number of decisions while engineering the features of our dataset. First, we converted the “Date” column into the more interpretable integer-based features: “month”, “day_of_week”, and “hour”. Then, to better weigh the effect of temporal patterns, we included two lagging columns for the main pollutants: CO, NOx, and Benzene. These two lagging columns contained the values for the values recorded 3 hours and 6 hours prior for the corresponding pollutant. These lagging columns help us capture time-dependent patterns and non-linear relationships that would otherwise be difficult to identify without adding them to the observed row. We felt the 3 and 6-hour windows were appropriate for our use case since we were mainly concerned with increasing the influence of very recent samples.

Moreover, to increase our ability to capture temporal patterns, we added rolling mean and standard deviation columns to our three main pollutants using a 3-hour window. These additional columns helped increase the robustness of our model in observing the variations of these pollutants over time. The rolling mean columns help smooth out the very recent fluctuations in our pattern identification and further helps reduce noise. The standard deviation columns help us measure the variability of the observation and help with appropriately weighing the volatility of the pollutants.

In [11]:
```
#Rolling statistics (mean and std over the last 3 hours)
data['CO_rolling_mean'] = data['CO(GT)'].rolling(window=3).mean()
data['CO_rolling_std'] = data['CO(GT)'].rolling(window=3).std()
data['NOx_rolling_mean'] = data['NOx(GT)'].rolling(window=3).mean()
data['NOx_rolling_std'] = data['NOx(GT)'].rolling(window=3).std()
data['C6H6_rolling_mean'] = data['C6H6(GT)'].rolling(window=3).mean()
data['C6H6_rolling_std'] = data['C6H6(GT)'].rolling(window=3).std()
```

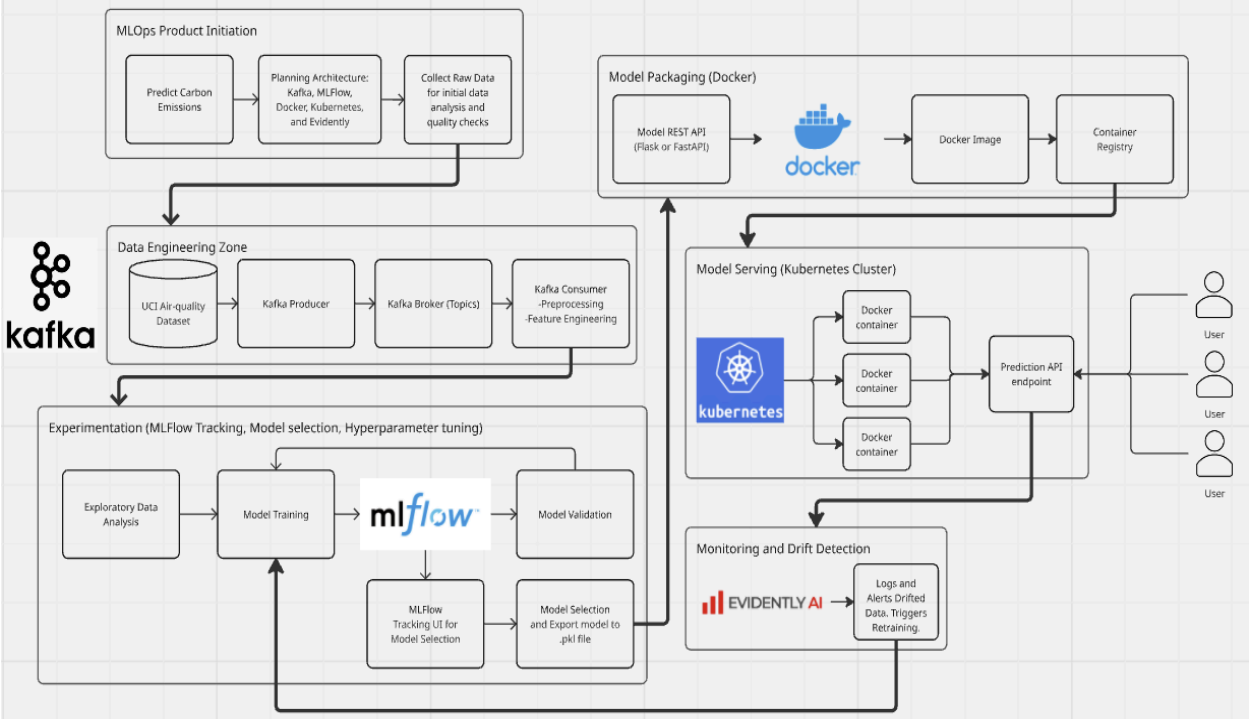
Engineered Features	***Description***
hour	Indexed hour of the day
day_of_week	Indexed day of the week
month	Indexed month of the year

CO_lag1	One hour lag of CO(GT) reading
CO_lag2	Two hour lag of CO(GT) reading
NOx_lag1	One hour lag of NOx(GT) reading
NOx_lag2	Two hour lag of NOx(GT) reading
CH6H_lag1	One hour lag of CH6H(GT) reading
CH6H_lag1	Two hour lag of CH6H(GT) reading
CO_rolling_mean	CO(GT) mean last 3 hours
CO_rolling_std	CO(GT) standard deviation last 3 hours
NOx_rolling_mean	NOx(GT) mean last 3 hours
NOx_rolling_std	NOx(GT) standard deviation last 3 hours
C6H6_rolling_mean	C6H6(GT) mean last 3 hours
C6H6_rolling_std	C6H6(GT) standard deviation last 3 hours

Overall, these additional adjustments are sufficient enough to help us develop our model with the most useful set of features. They provide a combination of adding the additional weight and measurability of time-based trends along with safeguards to help smooth out any fluctuations in the data.

System Architecture Design

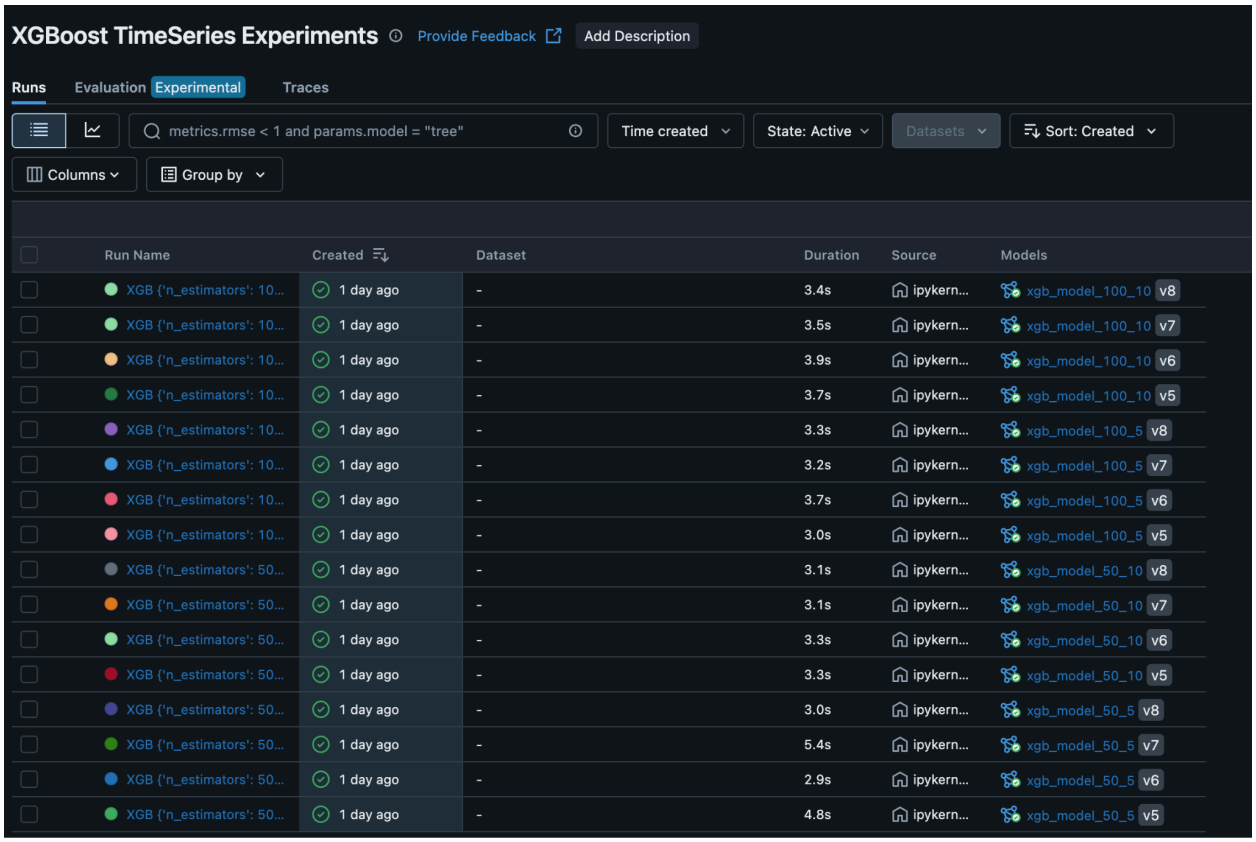
Current Progress: Data Prepared, Model Trained, and Best Model Selected

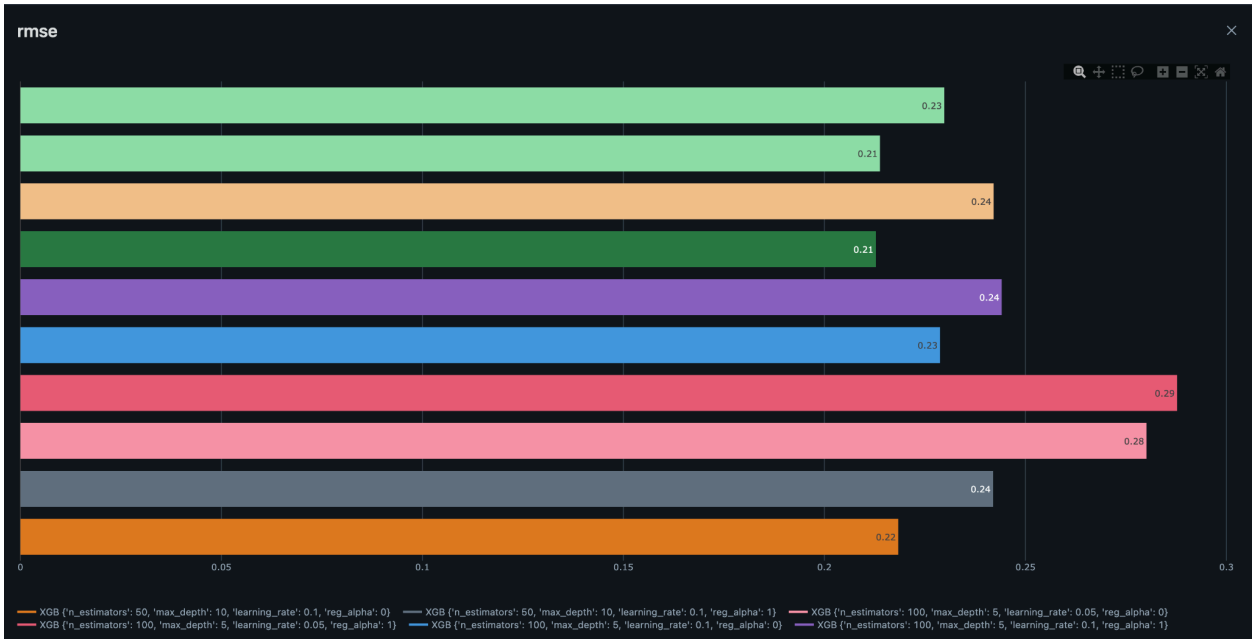


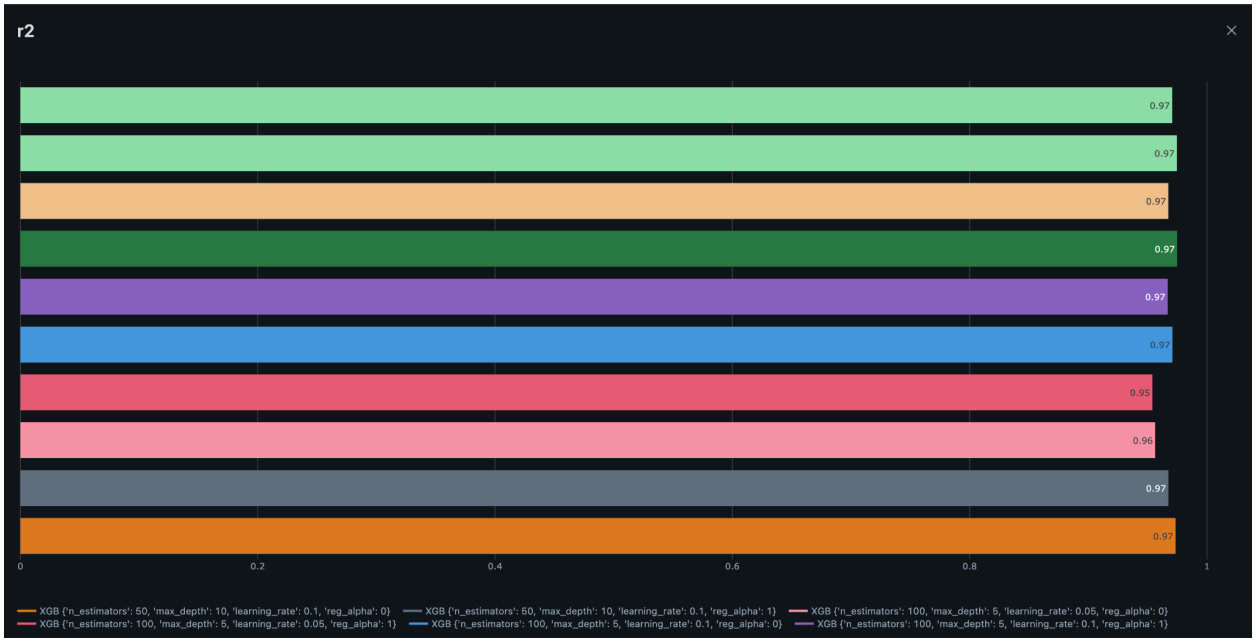
The initial phase of the pipeline focused on preparing and experimenting with data from the UCI Air-quality dataset. This phase involved exploratory data analysis (EDA), including identifying missing values, examining feature distributions, and understanding correlations between environmental variables and the target output, CO(GT). The feature engineering techniques mentioned above were applied to create meaningful inputs for the model like engineering time based features.

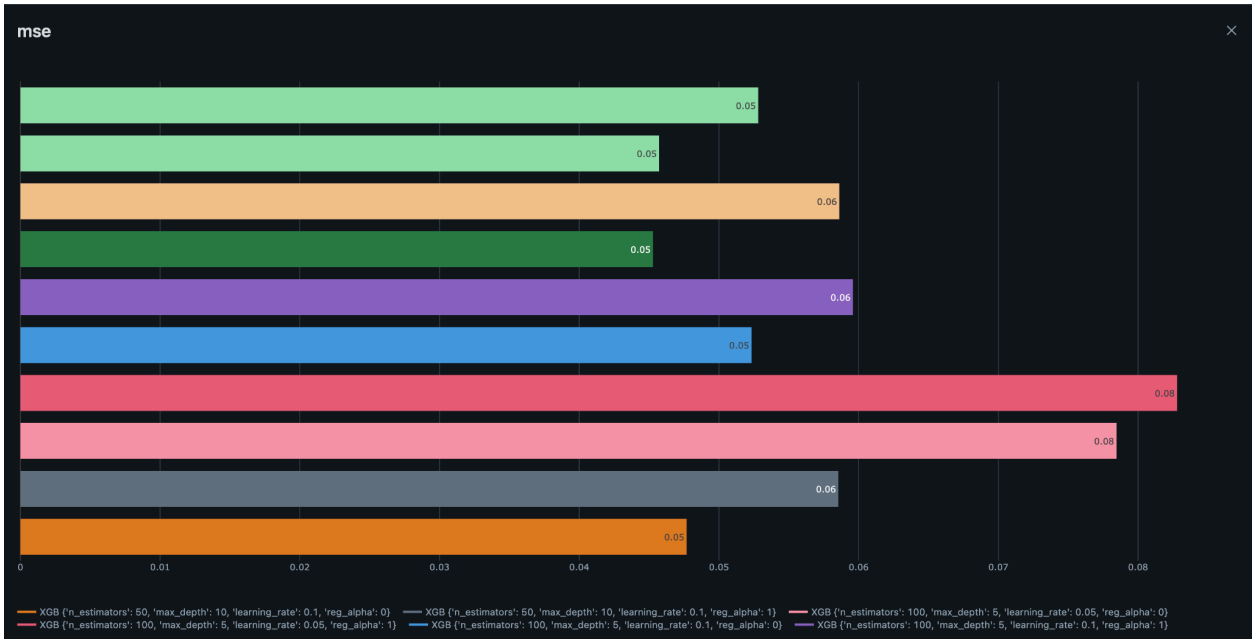
Following data preprocessing, multiple machine learning models were developed and evaluated using Python-based libraries such as scikit-learn and XGBoost. These models were assessed based on performance metrics like mean squared error (MSE), root mean squared error (RMSE), and R-squared score. Throughout this experimentation, MLFlow was integrated to log hyperparameters, training metrics, model artifacts, and results. Each run was tracked to facilitate comparison and

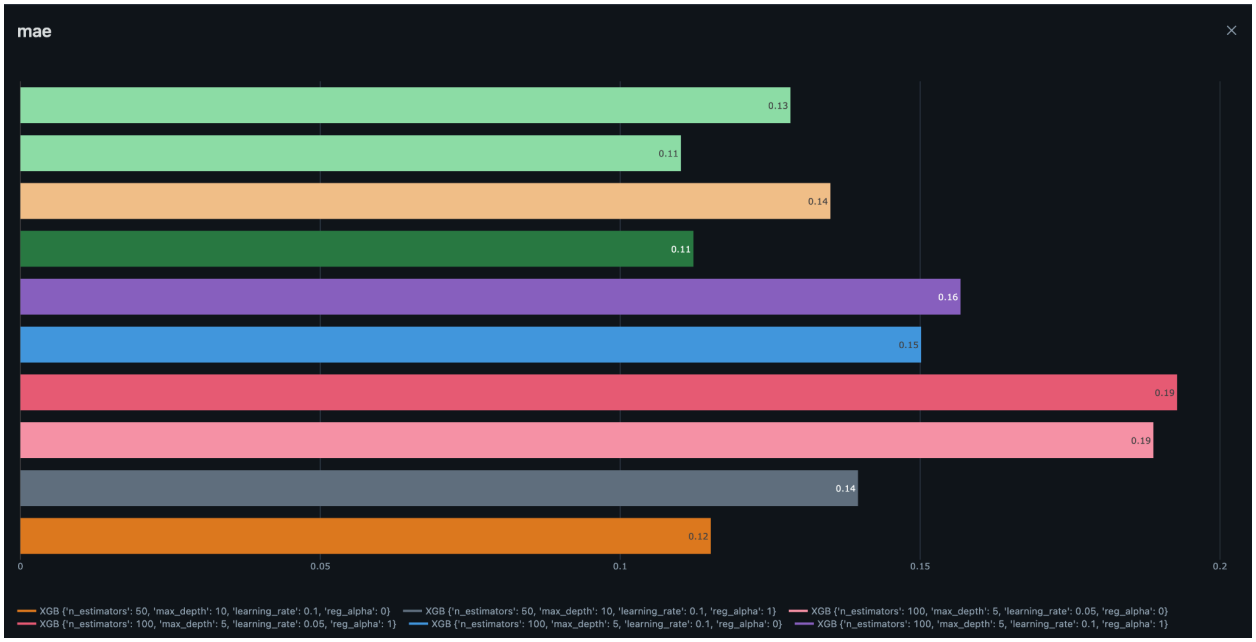
reproducibility, while the MLFlow Model Registry provided a versioned history of the trained models.






r2
0.97
0.97
0.97
0.97
0.97
0.97
0.95
0.96
0.97
0.97
0
0.2
0.4
0.6
0.8
1
XGB {'n_estimators': 50, 'max_depth': 10, 'learning_rate': 0.1, 'reg_alpha': 0}
XGB {'n_estimators': 50, 'max_depth': 10, 'learning_rate': 0.1, 'reg_alpha': 1}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.05, 'reg_alpha': 0}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.05, 'reg_alpha': 1}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1, 'reg_alpha': 0}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1, 'reg_alpha': 1}


mse
0.05
0.05
0.06
0.05
0.06
0.05
0.08
0.08
0.06
0.05
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
XGB {'n_estimators': 50, 'max_depth': 10, 'learning_rate': 0.1, 'reg_alpha': 0}
XGB {'n_estimators': 50, 'max_depth': 10, 'learning_rate': 0.1, 'reg_alpha': 1}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.05, 'reg_alpha': 0}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.05, 'reg_alpha': 1}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1, 'reg_alpha': 0}
XGB {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.1, 'reg_alpha': 1}



After selecting the best-performing model based on validation performance, the model was serialized into a .pkl file using the pickle library. This artifact represents a deployable version of the trained model and will be used as input for future stages of the pipeline. In parallel with model development, Apache Kafka was used to simulate data streaming of live data. A Kafka producer was configured to simulate the time-series air quality data into a dedicated Kafka topic. On the receiving end, a Kafka consumer was developed to serve as the preprocessing component of the pipeline. This consumer ingests the raw streamed data, performs feature engineering and prepares the data in the same format used during model training. With the .pkl file integrated into the consumer file, the model uses the preprocessed data to make CO(GT) predictions in the final output:

actual_CO(GT)	predicted_CO(GT)
1.1	1.099938
1.1	1.085428
1.2	1.211365
2.0	1.916879
2.2	2.127414

Now with the data engineering complete, experimentation and model packaging complete, and predictions made, the team is now ready to containerize the integrated model and prepare for deployment.

Remaining Work: Model Packaging, Serving, and Monitoring

The next step in deploying our model involves wrapping it inside a lightweight web service using a framework like Flask or FastAPI. This exposes an API endpoint for users and applications to interact with. This endpoint allows external applications to send data to the model and receive our model's predictions in response. To ensure that the deployment is scalable and consistent, the model and its web service are

containerized using Docker. Containerization of the model ensures that the environment remains the same across different machines so it is easier to manage and maintain.

Once the model is containerized, its deployment is orchestrated using Kubernetes. Kubernetes clusters automate scaling the application by replicating the Docker containers to handle varying user demand, balancing the load across multiple instances to ensure even distribution of requests, and managing fault recovery to maintain high availability. By scaling the number of the model's containers up or down based on demand, Kubernetes ensures efficient resource utilization. In the event a container fails, it can automatically restart the container or replicate a new instance to replace it, ensuring minimal disruption to the service. This makes the model resilient, scalable, and adaptable.

After deployment using Docker and Kubernetes, monitoring the model's performance is essential to maintain accuracy. A model monitoring tool like Evidently AI can detect data drift and generate alerts when the model's performance starts to decline. These alerts can prompt a return to the model training phase, using new or updated data to retrain and improve the model. This forms a continuous feedback loop that ensures that our model evolves alongside real-world changes.

Model Development and Experimentation

Developing our model required a number of assumptions in our experiment design. Before choosing the model, we needed to decide which pollutant we would attempt to predict and how best to split the dataset for our model to train on it. Given the relative sparseness of CO values in the original dataset, we thought it would be a unique challenge to attempt to predict the CO values for our experiment. We also figured that it would make the most sense to preserve the ordinality of the data and use an 80-20 train-test split, with the oldest 80% of rows belonging to the training dataset and our test dataset as the latest 20% of rows. This ensures we preserve any temporal or seasonal trends throughout the data that we would potentially lose with a traditional random sample.

We used MLflow to create our models and set up experiments to track our model performance. We established a baseline by modeling a linear regression without additional feature engineering. This allowed us to evaluate its performance and determine if the more complex methods would provide meaningful improvements in the accuracy of the predictions.

Baseline Linear Regression Metrics (Mean ± Std)

Mean Squared Error:	0.4010 ± 0.1156
Root Mean Squared Error:	0.6265 ± 0.0918
R-squared:	0.7282 ± 0.1271

After experimenting with a number of different models, we found that we had exceptional performance with XGBoost. In order to help fine-tune the performance of our model, we included a number of hyperparameters to circulate through in order to tune our performance in our own version of a grid search to find the best parameters.

In [23]:
```
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [5, 10],
    'learning_rate': [0.05, 0.1],
    'reg_alpha': [0, 1],
}

best_model = {'mse': float('inf'), 'params': None, 'model_uri': None}
```

To track performance across training runs, we kept track of several metrics to measure against, including MSE, RMSE, MAE, and R2**.** After testing our model across all different combinations of the parameters above, we found that we achieved the best performance with the hyperparameters of 100 n_estimators, a max_depth of 10, a learning_rate of 0.5, and reg_alpha of 0. With this combination of parameters, we achieved an impressive MSE of 0.0453, RMSE of 0.2128, MAE of 0.1122, and R2 of 0.9748. These metrics reflect the relatively strong performance of our model's ability to predict our CO values accurately.

Project Plan and Timeline

Week 1 (April 8 - April 14):

- ☑ Identify which ML model (including cleaned dataset) to experiment with and integrate into MLflow
- ☑ Align on feature engineering approach & begin experimentation, logging the results of each
- ☑ Integrate model and experiments into MLflow, highlighting evaluation metrics
- ☑ Integrate best-performing model into the Kafka consumer file using the MLflow model API; test predictions
- ☑ Draft phase 1 interim report

Week 2 (April 15 - April 21):

- ☐ Containerize best-performing model using Docker
- ☐ Integrate containerized model in Kubernetes for deployment
    - ☐ Deployment file: replicas
    - ☐ Deployment file: container image
    - ☐ Deployment file: ports
- ☐ Implement model monitoring using Evidently
- ☐ Draft phase 2 final report

Week 3 (April 22 - April 28):

- ☐ Develop presentation slides on our project approach, results & system architecture
- ☐ Practice presentation
- ☐ Record presentation (save as MP4 format; submit in Canvas)

*Team structure**

- Shaurya Gulati - data/software engineer; MLflow
- Helom Berhane - software engineer; Docker
- Trevor Foster - software engineer; Kubernetes
- Amy Kang - product manager; Evidently

* While we assigned the primary user for these technologies, every member will also learn the full stack at a high level to contribute during meetings and implementation, as well as provide additional support. Each member will also play a role in understanding and monitoring our full system architecture, ensuring the pipeline is in motion.

Challenges and Mitigation

One key challenge was attempting to export our models using MLflow in a cloud environment. MLflow's tracking and artifact storage functions were more suited to local environments for our current setup, which led us to centralize the infrastructure on a single team member's computer to ensure consistency.

Additionally, since each team member initially used slightly different preprocessing steps and model configurations, we faced discrepancies in results. To address this gap, we aligned on a unified preprocessing pipeline, feature set, and modeling approach to ensure comparability and reproducibility across experiments.

Conclusion and Next Steps

In this first phase, we successfully built a real-time data streaming and machine learning pipeline using Apache Kafka and MLflow to predict air quality based on hourly pollutant readings. Through data preprocessing, time-based feature engineering, and model experimentation, we developed a system capable of forecasting CO(GT) concentrations with improved accuracy. Our integration of MLflow enables consistent tracking of model experiments and metrics across development cycles.

Looking ahead, our next steps focus on operationalizing this pipeline for deployment. We plan to containerize the best-performing model using Docker and set up deployment with Kubernetes to enable scalability. To ensure model reliability in production, we will implement monitoring tools using Evidently to track prediction drift and data quality over time. Finally, we will prepare a presentation to communicate our technical approach, findings, and improvements to stakeholders.