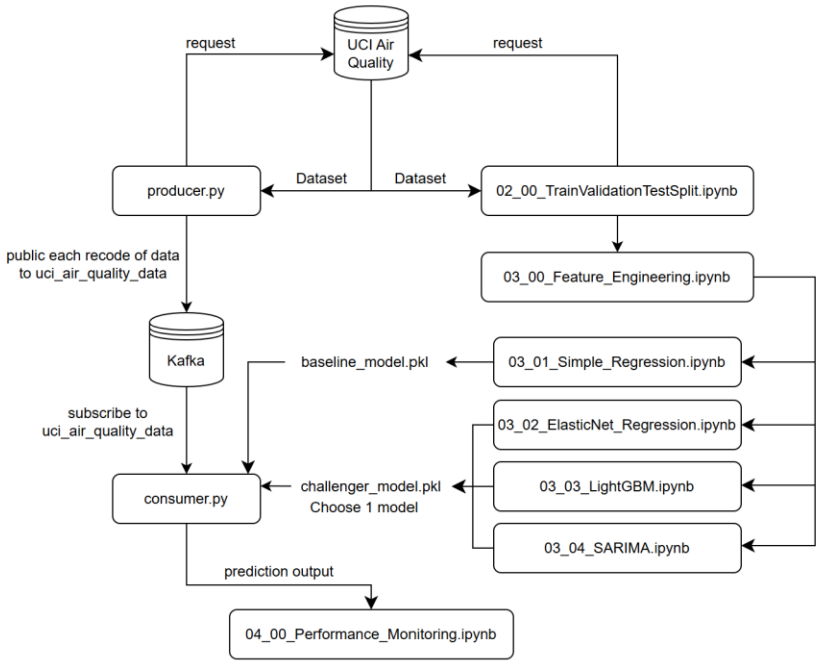
Name: Pongsakorn Supakpanichkul
ID: psupakpa

Introduction

This project presents an end-to-end data pipeline that ingests, processes, and analyzes hourly air quality data from the UCI Air Quality dataset using Apache Kafka as the core streaming infrastructure. The pipeline is composed of multiple phases, each designed to address key challenges in real-time data handling: ingestion, transformation, modeling, and evaluation.



From the picture, the system begins with Kafka producer that reads historical air quality data and simulates real-time streaming by sending messages at fixed intervals. The Kafka consumer listens for these messages, applies data preprocessing and feature engineering, including handling missing values, generating rolling statistics, and creating lagged and dummy variables, and then feeds the processed data into based-line model and challenger model to compare the prediction of pollutant concentrations. The results are logged and exported for monitoring and analysis.

To simulate a real-world scenario, we use chronological train-validation-test split to ensure robust model evaluation and avoid data leakage in this temporal context. The dataset was divided as follows:

- Training Dataset: From March 11, 2004 to October 31, 2004 — used to train predictive models.
- Validation Dataset: From November 1, 2004 to December 31, 2004 — used to tune hyperparameters and compare model performance.
- Testing Dataset: From January 1, 2005 to April 4, 2005 — used to evaluate the final model performance on unseen data.

This report will outline the Kafka implementation, key insights from exploratory data analysis (EDA), model results and limitation of the analysis. By demonstrating how Kafka and machine learning can be integrated for real-time environment, this project will serve as a foundation for future deployment in public health and environmental monitoring systems.

Name: Pongsakorn Supakpanichkul
ID: psupakpa

Kafka Setup Description

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. It plays a central role in this project by facilitating the continuous flow of air quality data from a simulated source (producer) to the data processing and prediction engine (consumer). This section outlines the end-to-end setup of Kafka for the real-time air quality monitoring pipeline.

1. **Prerequisites**
   - Download Java Development Kit (JDK)
     - Install JDK 8 or higher: Download from the Oracle website or use OpenJDK.
     - Set JAVA_HOME Environment Variable:
       - Right-click on This PC or My Computer and select Properties.
       - Click on Advanced system settings.
       - Click on Environment Variables.
       - Under System variables, click New:
         - Variable name: JAVA_HOME
         - Variable value: Path to your JDK installation (e.g., C:\ProgramFiles\Java\jdk-17.0.1)
       - Edit the Path variable:
         - Add %JAVA_HOME%\bin to the list.
   - Create kafka folder
     - Go to C:\
     - Create new folder and name it as "kafka"
2. **Download Apache Kafka**
   - Visit the Apache Kafka Downloads page. (https://kafka.apache.org/downloads)
   - Download the latest binary release (e.g., kafka_2.13-4.0.0.tgz).
   - Extract the downloaded .tgz file by using 7-Zip or a similar tool.
   - Double-click into the extracted folder (e.g., folder name: kafka_2.13-4.0.0)
   - Double-click into the inside single folder (e.g., single folder name: kafka_2.13-4.0.0)
   - Move all the inside folder to the kafka directory created in step 1 (e.g., C:\kafka)
3. **Generate Cluster ID**
   - Open Command Prompt (CMD) in administrator mode by
     - go to window
     - in search box type cmd
     - once cmd program appear, right click and click "Run as administrator"
   - Change current directory to C:\kafka by the following command:
     - cd C:\kafka
   - Generate Cluster ID by the following command:
     - ./bin/kafka-storage.sh random-uuid
   - Format Log Directories by the following command:
     - ./bin/kafka-storage.sh format -t <CLUSTER_ID> -c fig/kraft/server.properties
     - replace CLUSTER_ID by the cluster id from the previous step
     - e.g. ./bin/kafka-storage.sh format -t c9a5c1a2-d6f2-4c11-8f7f-db0015e8ab3 -c config/kraft/server.properties

4. **Configuration on broker.properties**
   - Go to C:\kafka\config
   - Open file server.properties with text editor (name of the file will only server)
   - Change the file according to the following:
     - node.id=0
     - process.roles=broker,controller
     - log.dirs=C:/tmp/kraft-combined-logs
     - listeners=PLAINTEXT://localhost:9092,CONTROLLER://localhost:9093
     - advertised.listeners=PLAINTEXT://localhost:9092
     - controller.listener.names=CONTROLLER
     - inter.broker.listener.name=PLAINTEXT
     - controller.quorum.voters=0@localhost:9093
     - auto.create.topics.enable=true
     - num.network.threads=3
     - num.io.threads=8
5. **Create tmp folder for kraft-combined-logs**
   - Go to C:\
   - Create folder name tmp
6. **Start Kafka in KRaft mode**
   - Open CMD in administrator mode
   - Open kafka in KRaft mode from the following command:
     - .\bin\windows\kafka-server-start.bat .\config\broker.properties
   - If you see "[KafkaServer id=0] started" on the CMD then it means the system fully running Kafka 4.0.0 in KRaft mode
7. **Create kafka Topic**
   - Open CMD in
   - Create kafka topic from the following command:
     - .\bin\windows\kafka-topics.bat --create --topic <TopicName> --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
     - e.g. .\bin\windows\kafka-topics.bat --create --topic uci_air_quality_data --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

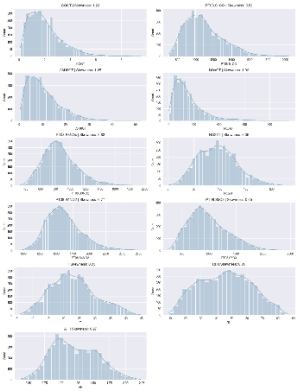
Name: Pongsakorn Supakpanichkul
ID: psupakpa

Data Exploration Findings

Before developing any analytical or predictive models, it is essential to thoroughly understand the dataset. This section presents key insights uncovered through exploratory data analysis (EDA) of the Air Quality dataset obtained from the UCI Machine Learning Repository.

In general, we first replace the default placeholder for missing values (−200) with actual null values (NaN) to more accurately reflect the true nature of the data. All EDA is then conducted exclusively on the training dataset to prevent data leakage and maintain the integrity of model evaluation.

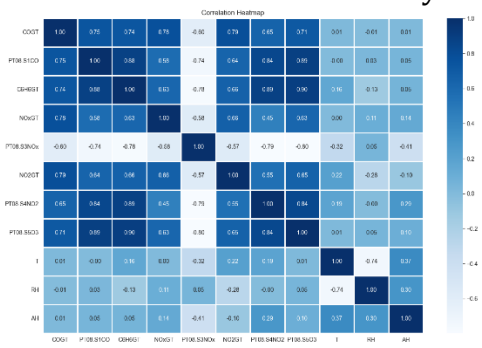
1. **Null Value**
   The dataset uses -200 as the default placeholder for missing values. We first replaced all instances of -200 with actual null values and conducted exploratory data analysis (EDA) to identify patterns in the missing data. Ultimately, we chose to forward-fill the missing values using the previous valid observation, as this approach preserves daily and weekly trends.
   Moreover, since NMHC(GT) contains high null values (83.90%), we decided to drop this pollutant from our analysis. And because the ground truth of NMHC contains a lot of null values, we cannot validate the sensor related to it, then we also drop PT08.S2NMHC from our analysis.
2. **Distribution**
   Based on the distribution plot, we can notice that most of the data follow Gaussian distribution except for CO(GT), C6H6(GT), NOx(GT). The criteria is that if skewness is more than 1, we will determine the data is not following Gaussian distribution. This suggests that for these skewed features, we may need to apply some transformation to make it follow Gaussian distribution before modeling.



   *Full size plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb
3. **Pair Plot**
   Based on the pair plot, we can observe that most of the data are linearly related to each other, especially among the pollutant concentrations. This suggests that models well-suited for capturing linear dependencies, such as linear regression, may be effective for this dataset. Alternatively, it also indicates that certain pollutants could serve as useful predictors for estimating the concentrations of others.

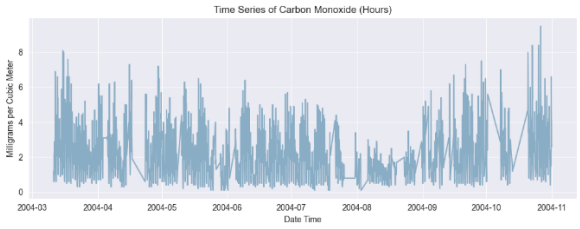
Name: Pongsakorn Supakpanichkul
ID: psupakpa

4. **Correlation heatmap between different pollutants**
   Based on the heat map of correlation, we can observe that there are high correlations between the pollutant concentrations and its sensor readings. This suggests that we need to handle multicollinearity when we build a regression model.



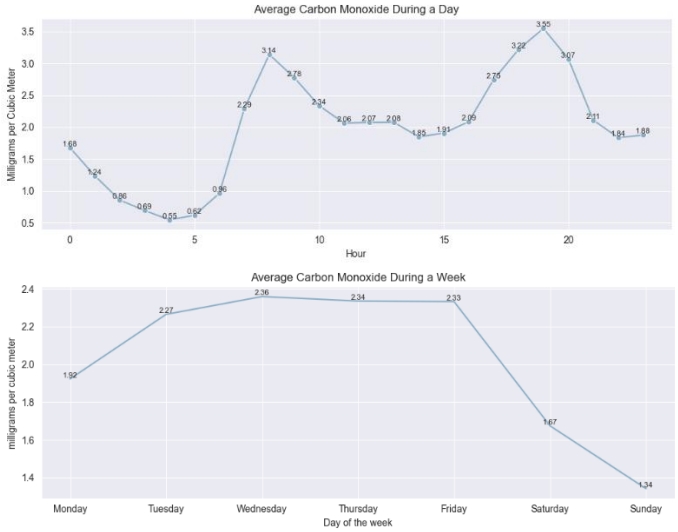
*Full size plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb

5. **Time-series plots of CO, NOx, and Benzene concentrations**
   Based on the time-series plots for each pollutant, we can observe fluctuations over time but no clear long-term trend. This suggests that when building a SARIMA model, the trend component can likely be omitted. Alternatively, the ADF test can be conducted under the assumption of a constant mean without a trend.



*Other plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb

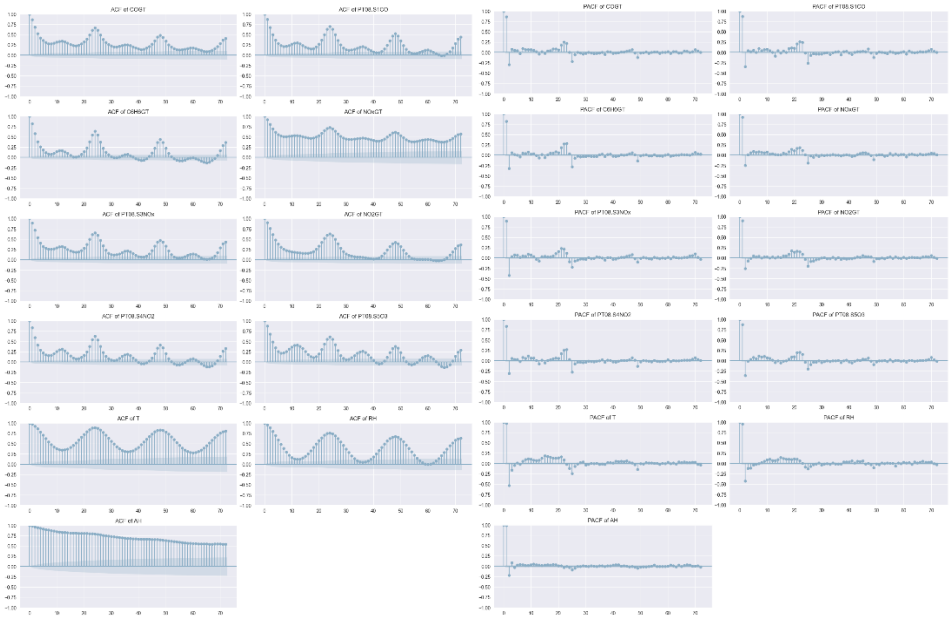
6. **Daily/weekly patterns (average by hour of day, day of week)**
   Based on the daily and weekly patterns, we observe significant variations throughout the day and across different days of the week. This suggests that missing values should be filled using the previous valid observation, as this method helps preserve both hourly and weekly trends in the data.



*Other plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb

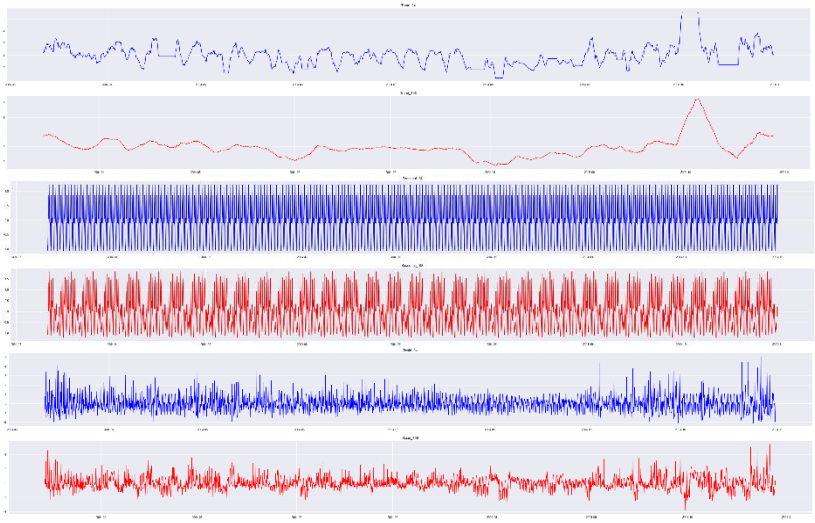
Name: Pongsakorn Supakpanichkul
ID: psupakpa

7. **Autocorrelation and partial autocorrelation plots**
   Based on the autocorrelation plots, we observe that past values have a strong relationship with current values, indicating temporal dependence in the data. The partial autocorrelation plots further confirm that a few recent lags contribute significantly to the current value. This suggests that incorporating lagged features into our model is important for capturing the underlying time-dependent structure of the data, and that time series models like SARIMA are well-suited for this task.



*Full size plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb

8. **Decomposition of time series into trend, seasonality, and residuals**
   Based on the decomposition of the time series into trend, seasonality, and residuals both 24 lags and 168 lags, we observe clear seasonal patterns and some level of noise, but no strong long-term trend. This suggests that modeling efforts should focus on capturing seasonal effects and minimizing residual variance, while the trend component can be considered omitted.



*Full size plots are in 02_01_00_DataCleaningAndEDA_for_Train.ipynb

Name: Pongsakorn Supakpanichkul
ID: psupakpa

Modeling Approach and Results

To ensure that our modeling approach can operate in a real-time environment, we need to design the pipeline to integrate with Kafka's streaming architecture. Therefore, feature engineering will be the most important aspect that we need to consider whether it can be implementable in real-time situations.

Feature Engineering Technique

1. **Time-based features**
   Based on the timestamp provided in the dataset, we extracted several time-based features to help the model learn patterns related to time. These include:
   - Hour of the day to capture daily cycles
   - Day of the week to capture weekly cycles

   However, we cannot perform the monthly features because it will not be implementable in the real-time environment. This happens because the training dataset only contains the data from March to October. If the real data happens during November to February, the model will not know the pattern during that period. This leads to the decision not to develop the monthly feature to capture the yearly cycle.
2. **Lagged features**
   Based on the features provided in the dataset, we extracted several lagged features to represent pollutant values from previous time steps and are essential for capturing temporal dependencies. Specifically, we generated lag features for each pollutant variable up to 24 previous periods (hours). This allows the model to learn from a full day historical data when making predictions.
3. **Rolling statistics features**
   Based on the features provided in the dataset, we extracted several rolling statistics features both mean and standard deviation to help smooth out short-term noise and highlight consistent patterns over recent time windows.
   - Rolling Mean: The average value over a specified window up to 24 previous periods. This helps the model recognize local trends in the data.
   - Rolling Standard Deviation: Measures the variability of pollutant levels within the window. This feature is useful for identifying periods of high volatility or instability in air quality.

   To ensure that data leakage is prevented during model training, rolling features are created by shifting the data forward in time. This means that, at any given timestamp, the model only has access to information from the past—never from the future. For example, the features to predict the data at 2004-03-11 02:00:00 will use the average from 2004-03-11 00:00:00 to 2004-03-11 01:00:00.

	date_time	COGT	rolling_COGT_mean_2	rolling_COGT_mean_3
0	2004-03-11 00:00:00	1.2	NaN	NaN
1	2004-03-11 01:00:00	1.0	NaN	NaN
2	2004-03-11 02:00:00	0.9	1.10	NaN
3	2004-03-11 03:00:00	0.6	0.95	1.033333
4	2004-03-11 04:00:00	0.6	0.75	0.833333

*Validation dataset will be used in consumer.py as the historical context for calculating lagged and rolling features.

4. **Scaling features**
   To ensure that all features are on the same scale and to improve the performance of models sensitive to feature magnitude, we applied StandardScaler to all features used in linear models.
   *Scaling model for each pollutant will be saved as pickle files so that the same scaling method can be used in consumer.py when we predict the test dataset.
5. **Column Ordering**
   To ensure that all features are stored according to the train dataset, we export list of column of the train dataset in json files. These files will be use when we predict the test dataset in consumer.py

Modeling Technique
Based on the insights gained during the EDA process, we scope down models that are well-suited for time series forecasting and capable of capturing the linear relationships between variables, temporal dependencies, and seasonal patterns in the data. In addition to selecting advanced models, we also introduced a baseline model and a challenger model to benchmark performance.

- The baseline model relies on a simple model, we will use Simple Linear Regression as our based-line model.
- The challenger models include:
  - Linear Regression with Elastic Net: capture linear relationship, temporal dependencies and prevent multicollinearity.
  - Light Gradient Boosting Machine (LightGBM): complex non-linear relationships and interactions between features.
  - SARIMA: capture seasonality, trend, and autocorrelation in time series data.

Model Tuning
To ensure optimal performance, each selected model was fine-tuned to minimize prediction error. Model tuning involves identifying the best combination of hyperparameters that improve accuracy and generalization.

- Simple Linear Regression: No specific setting
- Linear Regression with Elastic Net
  - Performed grid search to tune the balance between L1 (Lasso) and L2 (Ridge) regularization using the l1_ratio and alpha hyperparameters.
    - l1_ratio: [0.1, 0.5, 0.9]
    - alpha: [0.01, 0.1, 1.0, 10]
- Light Gradient Boosting Machine (LightGBM)
    - n_estimators=1000
    - learning_rate=0.01
- SARIMA
  - Perform ADF test to make sure the series is stationary.
  - Manually select p, d, q and P, D, Q, S from ACF and PACF plot. (refer to plot and each parameter in 03_04_SARIMA.ipynb)

Evaluation Methodology

Each model was evaluated using two standard regression metrics:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in predictions, providing an interpretable unit-level error.
- **Root Mean Squared Error (RMSE)**: Penalizes larger errors more heavily than MAE, making it useful for detecting models that perform poorly on extreme values.

However, if there are conflicts between these two metrics, we will select the best model based on RMSE because it penalizes larger errors more heavily.

Model Result (Validation Dataset)

Model	CO(GT)		C6H6(GT)		NOx(GT)		NO2(GT)	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Simple Linear Regression	0.3361	0.5178	0.7811	1.4859	51.3732	74.9846	10.3662	14.0801
Regression with Elastic Net	0.3031	0.4697	0.7129	1.4497	51.1363	74.9202	9.6164	13.1835
Light Gradient Boosting Machine	0.4346	0.7068	2.0502	3.1705	72.741	123.223	11.2456	17.9303
SARIMA	1.2389	1.7132	6.4742	8.7177	200.919	279.889	52.3785	64.6502

Based on the evaluation table in validation dataset, the best-performing model is Linear Regression with Elastic Net, which achieved the lowest MAE and RMSE among all challenger models. Therefore, this model was selected for deployment. It was exported as a pickle file, allowing the consumer.py script to load the trained model and perform real-time predictions on incoming test data from the Kafka stream.

Model Result (Test Dataset)

Model	CO(GT)		C6H6(GT)		NOx(GT)		NO2(GT)	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
Simple Linear Regression	0.2621	0.3764	0.5723	0.9233	35.1226	53.6169	9.7264	13.4843
Regression with Elastic Net	0.2543	0.3592	0.5768	0.9318	34.8479	54.0417	10.4906	14.3561

Based on the evaluation table in the test dataset, the best-performing model for each pollutant is different.

- For CO, the best-performing model is Regression with Elastic Net.
- For C6H6, the best-performing model is Simple Linear Regression.
- For NOx, the best-performing model is Simple Linear Regression since we use RMSE as the final decision to select the best model.
- For NO2, the best-performing model is Simple Linear Regression.

Conclusion and Limitations

Conclusion

In this project, we developed a real-time air quality prediction pipeline using Apache Kafka and machine learning models on the UCI Air Quality dataset. Through careful preprocessing, feature engineering, and exploratory data analysis, we identified temporal patterns and relationships between pollutants.

Among the models tested,

For Carbon Monoxide, the Linear Regression with Elastic Net performed best based on MAE and RMSE.

For Benzene, the simple Linear Regression performed best based on MAE and RMSE.

For Nitrogen Oxides, the simple Linear Regression performed best based on RMSE.

For Nitrogen Dioxide, the simple Linear Regression performed best based on MAE and RMSE.

The best-performing models for each pollutant were integrated into a Kafka-based streaming environment, enabling real-time predictions on incoming air quality sensor data.

Limitations

1. **Domain Knowledge**
   - We don't know the causal relationships between each pollutant. For example, certain pollutants may lead to the formation of others under specific environmental conditions. Understanding these dependencies could help in preventing data leakage, especially when designing relevant model features and lagged features
   - We don't know how long each pollutant typically persists in the atmosphere. This would allow us to more effectively engineer lag features and rolling windows. As a result, it will reduce the complexity of model and computing power.
2. **Data**
   - We should aim to collect more comprehensive and recent air quality data to enable training and validation across a full year. A larger and more diverse dataset would improve model generalization and robustness.
   - We should incorporate external datasets, such as traffic flow data, weather conditions, or industrial activity logs, which could significantly enhance model performance. These variables are theoretically known to influence pollutant levels and would provide additional context for prediction.
3. **Model**
   - We should explore alternative modeling approaches, including deep learning techniques like LSTM or Temporal Convolutional Networks (TCNs), which may yield better performance, especially for capturing complex temporal patterns.
   - For SARIMA, we should investigate using automated hyperparameter selection tools to streamline model tuning. Relying solely on visual interpretation of ACF and PACF plots can be subjective and may not always result in the most optimal configuration.