Real-Time Air Quality Prediction with Kafka

Devan Rajendran

Introduction:

This project utilizes Apache Kafka and Python to create and build predictive models that predict the pollutants concentration in the air with real-time data streaming and environmental time series analysis. The UCI Air Quality dataset has been utilized, which consists of hourly pollutant and sensor readings from an Italian city between 2004 and 2005, providing the foundation for training and evaluating predictive models.

The steps to realize this project involves setting up Apache Kafka server and consumer that reads a dataset and records it into a Kafka topic. Exploratory Data Analysis was performed on the collected data set. This involved analyzing time-based patterns, exploring relationships between different pollutants and identifying seasonality, trends, and anomalies. Based on these features derived from the time-series data, predictive models to forecast CO concentrations were developed.

With rising concerns over urban pollution and its health impacts, timely and accurate forecasting of pollutant concentrations has become crucial. This project highlights the integration of big data tools and predictive analytics for smart city applications.

Kafka Set up:

Kafka Installation

Apache Kafka version 3.9.0 was installed with JDK 11 using homebrew. The commands carried out for the installation includes –

```
brew install openjdk@11
```

```
brew install kafka
```

The later versions of kafka and jdk ?(such as jdk17) had issues related to zookeeper not being automatically installed that it was necessary to install the above versions.

Once the environment was installed, the following commands were used to start the kafka server and zookeeper -

```
kafka-server-start /opt/homebrew/etc/kafka/server.properties
zookeeper-server-start /opt/homebrew/etc/kafka/zookeeper.properties
```

The kafka topic was established with the following set of commands –

```
kafka-topics --create  --topic air_quality_data --bootstrap-server
localhost:9092  - partitions 1 --replication-factor 1
```

And it was confirmed by the following command –

```
kafka-topics --list --bootstrap-server localhost:9092
```

The following commands were run to resolve dependencies with kafka-python libraries –

```
pip install kafka-python pandas
```

Producer Implementation:

A python script named producer.py was developed to read records from the UCI dataset and stream them to the air-quality topic as JSON messages, simulating real-time by adding a delay of 1 second per record using time.sleep(). Each message represents one hourly reading.

Consumer implementation:

A python script named producer.py was developed to subscribe to the same topic, receive incoming messages, and log the data. It also loads a pre-trained model to predict CO concentrations based on engineered features from the streamed data.

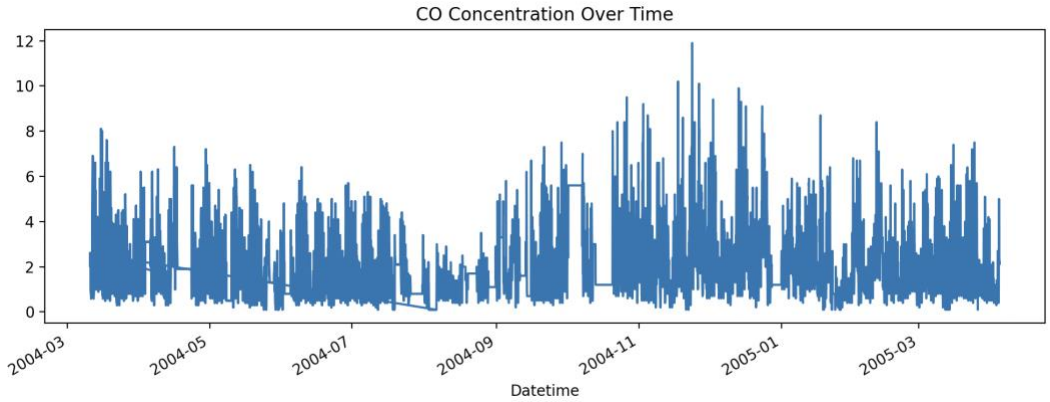
The strategy to pre process the data was then recorded in the preprocessing.md file which suggests a selected approach to data preprocessing.

Data Exploration

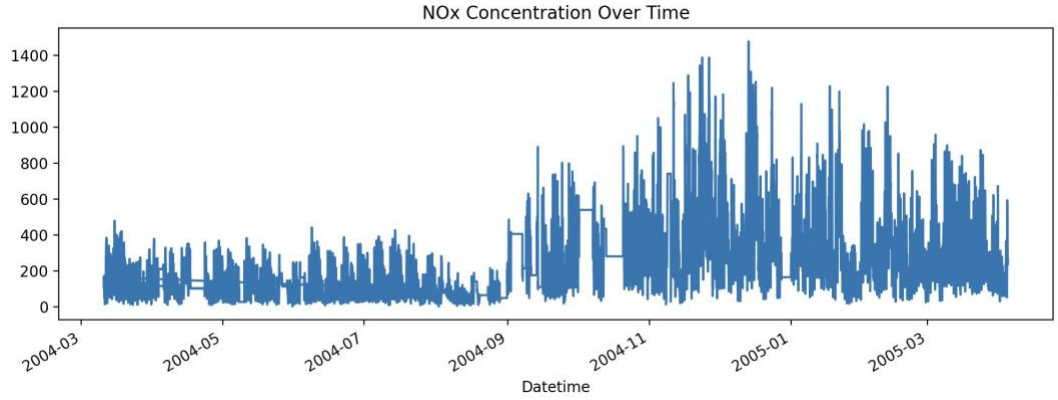
Basic Visualizations:

1. Time series plots (

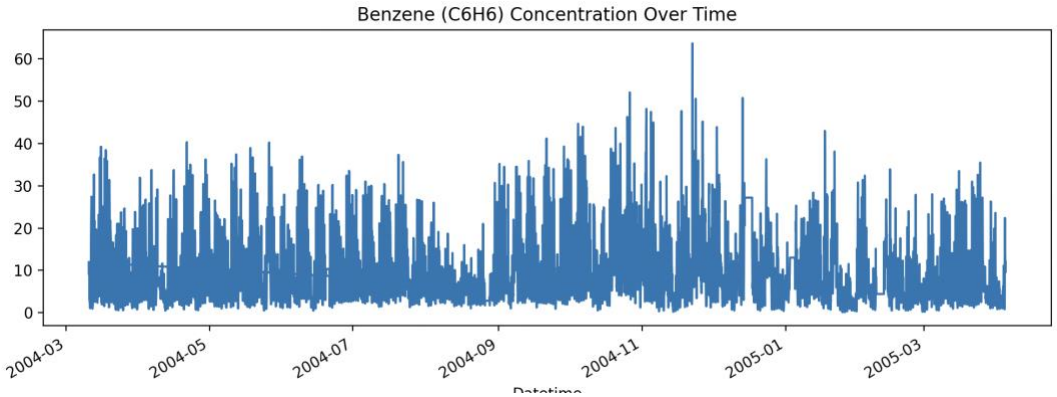
CO



Nox

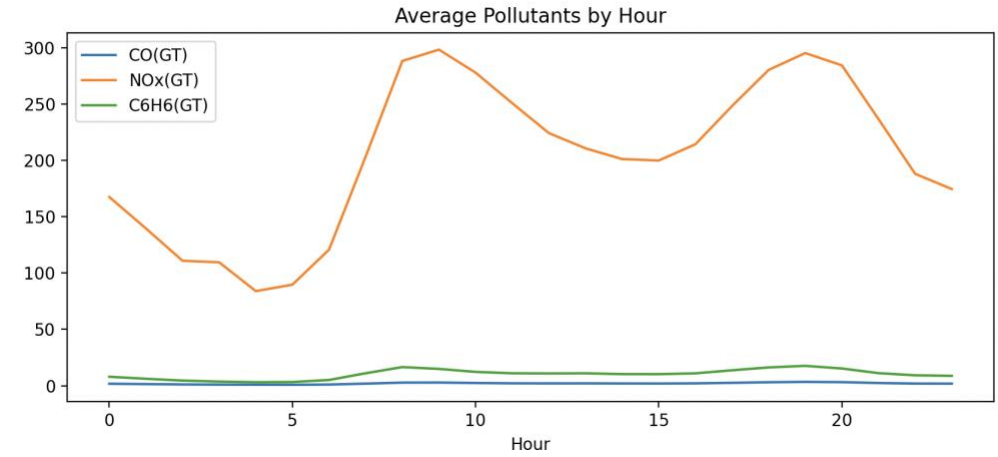


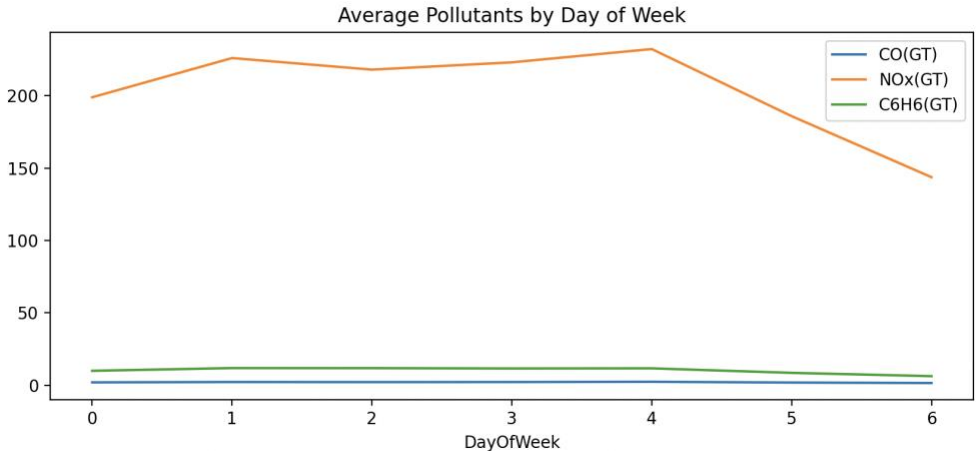
Benzene



These graphs shows the CO, NOx and Benzene concentration levels in the Air over the year. The peaks show the higher emission levels and traffic levels, and it is seen that the emission levels are higher in the winter months compared to the summer and fall. This indicates a seasonal influence on pollutant levels except for Benzene.

2. Daily/Weekly Pattern

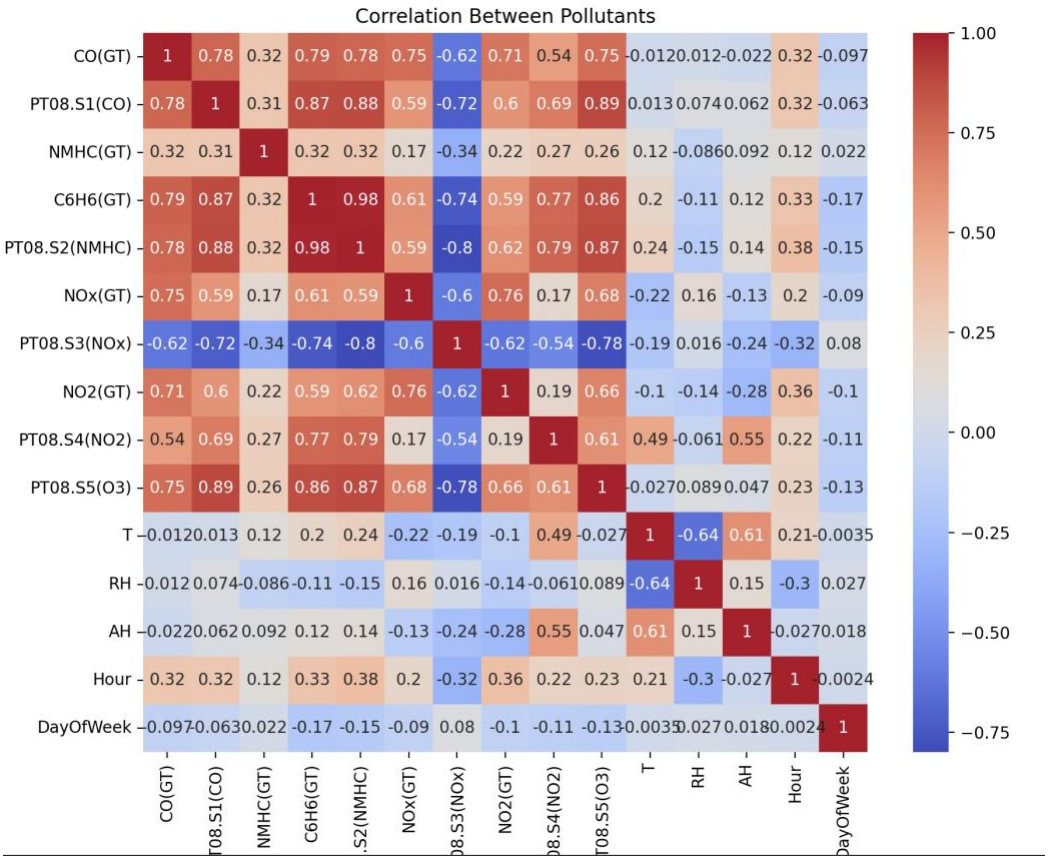




The peaks seen at 8AM and 8PM suggests the effect of peak traffic on the pollutants emission on the road, with NOx being the most emitted pollutant in terms of quantity. CO and C6H6 have visible morning and evening peaks as well. The higher average over the first few days of the week suggests increased traffic and industrial activities during the weekdays compared to the weekends.

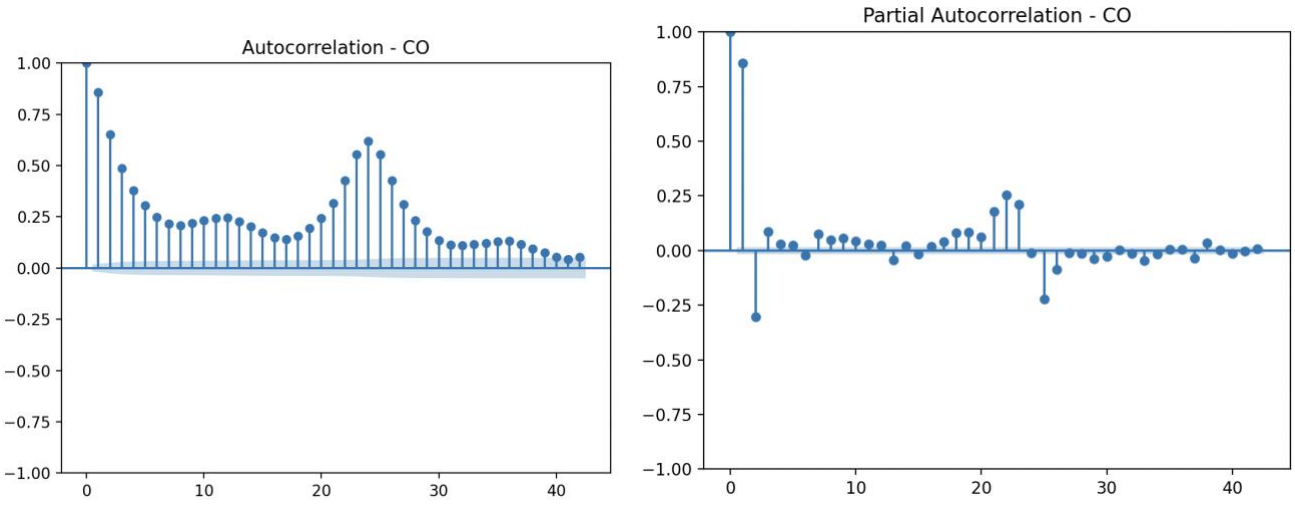
3. Correlation matrix

The graph shows strong correlations between CO, NOx, and Benzene which indicates shared emission sources. There is a mild correlation between time-based features like Hour and pollutant levels.



Advanced Visualizations

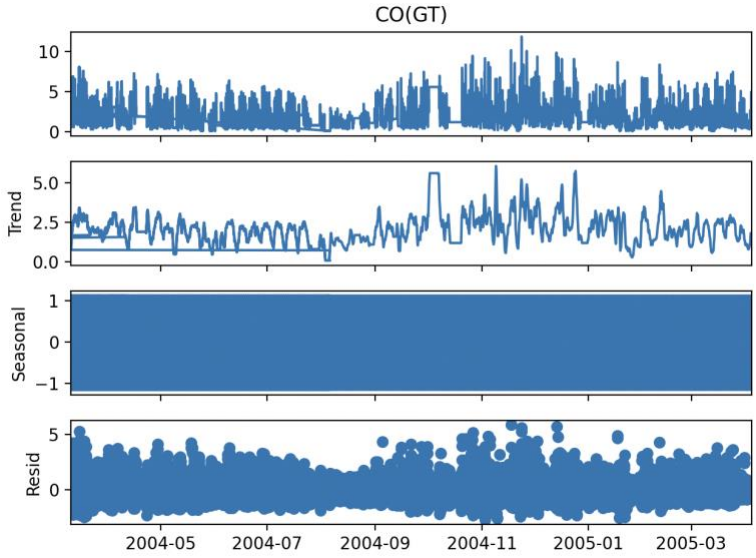
1. Autocorrelation and Partial Correlation for CO



2. Partial correlation CO

High autocorrelation at lag 1 and every 24 hour cycle suggest the influence of peak traffic and industrial times on the quantity of CO in the air. After controlling for intermediate lags, the partial correlation graphs show the directly predictive lags. The lags around 3 and 24 suggest the influence of traffic cycles. This is useful for a time based modelling.

2. Seasonal Trend



The decomposition shows a clear long-term trend and some repeating seasonality. The residuals show variability not explained by the trend or seasonality.

<u>Potential factors influencing air quality variations:</u>

1. Traffic Patterns: There is a visible correlation between the peak traffic hours and amounts of pollutants emitted in the air, indicating vehicular emissions to be a main source of pollution, especially for NOx
2. Weather and Seasons: Its seen that there is a pattern exhibited in the amount of pollutants emitted that varies with season. There are significantly more pollutants emitted during the winter when compared the summer and fall.
3. Industrial Activity: More pollutants emitted during the weekdays than the weekends show that industrial emissions during the work week also contributes to the NOx , CO and Benzene emissions.
4. Time of the day: Human activities (traffic and work hours) play a role in the emissions as well.

Modelling approach

Based on these observed patterns and factors, time-aware models like ARIMA, SARIMA, or ML models with lagged/time features such as Random Forests, XGBoost with time features looks to be more suitable for this dataset.

Some of the reasons for choosing such models are that variables like 'Hour' and 'DayOfWeek' should be included as categorical or cyclical features and that long-term trend and seasonality justify decomposition-based models or seasonal regressors

Feature Engineering

To prepare the data for model training, several features were created that could help capture temporal and trend-related information into a Python file called model_training.py. This included basic time components like hour, day, and month, which are known to influence pollutant behavior. Lagged values (CO_lag1, CO_lag2) were added to provide the model with memory of past observations. We also included rolling mean and standard deviation over a 3-hour window to account for short-term trends and fluctuations in CO levels.

```
df['Hour'] = df.index.hour
df['Day'] = df.index.day
df['Month'] = df.index.month
df['CO_lag1'] = df['CO(GT)'].shift(1)
df['CO_lag2'] = df['CO(GT)'].shift(2)
df['CO_roll3'] = df['CO(GT)'].rolling(3).mean()
df['CO_std3'] = df['CO(GT)'].rolling(3).std()
df.dropna(inplace=True)
```

Train/Test Split –

A chronological split method was used where the first 80% of the data was used for training and the remaining 20% for testing. This mimics a real-world scenario where we predict future values based on past data, avoiding data leakage.

```
split_index = int(len(df) * 0.8)
train = df.iloc[:split_index]
test = df.iloc[split_index:]
```

Models Trained –

We trained three types of models: a basic Linear Regression, which provides a good baseline; a Random Forest Regressor, which handles non-linearity and interactions well; and XGBoost, a gradient boosting technique that often performs well on structured data. Each model was trained on the same set of engineered features.

```
X_train = train.drop(columns=['CO(GT)'])
y_train = train['CO(GT)']
X_test = test.drop(columns=['CO(GT)'])
y_test = test['CO(GT)']

# Train models
lr = LinearRegression().fit(X_train, y_train)
rf = RandomForestRegressor(n_estimators=100).fit(X_train, y_train)
xgb = XGBRegressor().fit(X_train, y_train)
```

Evaluation Process –

To evaluate the models, there are two standard regression metrics used: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). These give a clear picture of average error and how much large errors are penalized. Implemented a function called evaluate_and_save_best_model() to automatically compare all trained models and pick the one with the lowest total error (MAE + RMSE). Models with a suspiciously perfect score (0.00 error) were ignored to prevent saving a broken or untrained model.

```
def evaluate(model, name):
    preds = model.predict(X_test)
    mae = mean_absolute_error(y_test, preds)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    print(f"{name} - MAE: {mae:.2f}, RMSE: {rmse:.2f}")

evaluate(lr, "Linear Regression")
evaluate(rf, "Random Forest")
evaluate(xgb, "XGBoost")
```

Results –

Out of all three, the Random Forest model showed the best performance on the test set. It produced the lowest MAE and RMSE values, meaning it was able to capture the pollutant trends most effectively. We also compared this with a simple baseline model that predicts the next CO value as the same as the previous one. The machine learning models, especially Random Forest, significantly outperformed this baseline.

```
models = {
    "Linear Regression": lr,
    "Random Forest": rf,
    "XGBoost": xgb
}

# Evaluate all and save the best one
best_model = evaluate_and_save_best_model(models, X_test, y_test)

# Save best model
joblib.dump(best_model, 'co_forecast_model.pkl')
print("Model saved as 'co_forecast_model.pkl'")
```

Kafka Integration –

Once the model was trained and saved using joblib, it was integrated into the Kafka consumer script. As the consumer received each real-time message, it extracted the same features used during training and fed them to the model. The predicted CO value was logged to the terminal, effectively creating a real-time inference system.

```
baseline_preds = X_test['CO_lag1']  # because you already engineered this feature
baseline_mae = mean_absolute_error(y_test, baseline_preds)
baseline_rmse = np.sqrt(mean_squared_error(y_test, baseline_preds))

print(f"Baseline Model (Previous Value) - MAE: {baseline_mae:.2f}, RMSE: {baseline_rmse:.2f}")
```

Conclusions and limitations

This project successfully implemented a real-time pipeline for air quality prediction using Kafka and machine learning. It combined time-series analysis, feature engineering, and model deployment to predict CO levels from streamed sensor data.

Limitations

- Only one pollutant (CO) was modeled.
- Rolling features assume fixed time intervals and may fail with missing data.
- Static model — no online learning or retraining on new data.

Future Work

- Extend to multi-pollutant prediction.
- Add visualization dashboards using Streamlit.
- Implement anomaly detection or alert systems.
- Explore LSTM or SARIMA models for better sequence learning.