Northeastern University
Khoury College of
Computer Sciences

Analyzing Air Quality: Insights into the Impact of Weather and Traffic in Big Cities

Author(s):
Atreyo Das, Northeastern University
Omer Seyfeddin Koc, Northeastern University
Shruti Suhas Kute, Northeastern University

Northeastern University

DS 5500 Capstone: Application in Data Science

Spring 2025

Date: 22 April 2025

Abstract

Air pollution in urban environments remains a pressing concern due to its profound impact on public health and environmental quality. This study examines how meteorological variables and traffic density influence air quality in three major US cities (New York, Chicago, and Los Angeles) by modeling concentrations of key pollutants, such as PM2.5, PM10, and NO_2. Drawing on five years of data (2020–2024) from various trusted sources, we implemented and compared multiple machine learning and time series models, including Random Forest, XGBoost, ARIMA and LSTM.

The project focused on understanding the effect of weather and traffic variables on pollutant levels using regression models, and forecasting future pollution trends using sequential time series models. Extensive pre-processing, including interpolation, scaling, and feature engineering, was applied to enhance the data quality and model performance. XGBoost emerged as the superior regression model, outperforming Random Forest, with an average RMSE score of 4.35. LSTM was the best performing time series model, with an average RMSE score of 4.42, significantly outperforming the more traditional ARIMA model. To enhance accessibility, we also developed an interactive dashboard to visualize model outputs and pollution trends across the cities studied.

The study's findings validate the effectiveness of machine learning models in forecasting pollution and underscore the value of integrating diverse data sources. This research lays the groundwork for scalable, real-time air quality monitoring systems and provides actionable insights to inform environmental policy and urban planning.

Contents

Chapter 1

Introduction

1.1 Background

Air pollution is one of the most critical environmental challenges affecting urban populations today. In densely populated cities, pollutants such as PM2.5, PM10, and nitrogen dioxide (NO_2) are primarily driven by vehicle emissions and influenced by changing weather conditions. According to the World Health Organization (WHO), air pollution has strong links to strokes, ischemic heart disease, chronic obstructive pulmonary disease, lung cancer and pneumonia[1].WHO also estimates that air pollution is associated with 8 million premature deaths annually[2]. This makes it vital to use tools like machine learning to study and predict pollutant levels. While prior research, like that of Castelli et al. [3], has utilized either weather or pollution data independently, few studies have integrated both meteorological and traffic data for comprehensive analysis, particularly across multiple cities. Leveraging such integrated data through advanced modeling techniques presents an opportunity to improve urban air quality forecasting and support informed policymaking.

1.2 Problem Statement

Predicting urban air pollution is a complex task due to the dynamic interplay between meteorological conditions, traffic patterns, and pollutant emissions. Although Abhilash et al. [4] received great results using ARIMA, traditional statistical methods like them often fall short in modeling non-linear data typically seen in urban pollution datasets. Furthermore, many existing studies focus on a single city or data source, limiting the generalizability and applicability of their findings. This project addresses these limitations by combining traffic and weather data with pollutant measurements to build robust, predictive models that can be applied across different urban settings. The goal is to provide actionable insights and tools for real-time air quality monitoring and intervention.

1.3 Objectives

The primary objective of this project is to analyze the relationship between air quality, weather conditions, and traffic patterns in major US cities and develop accurate predictive models for pollutant levels. We aim to identify key factors influencing air quality and build robust models capable of predicting pollutant concentrations in urban environments.

Our experimentation is guided by a set of well-defined hypotheses that align with the project objectives. The first hypothesis suggests that meteorological variables, such as temperature, wind speed, etc., significantly influence pollutant levels, and these relationships can be effectively captured using machine learning regression models. The second hypothesis proposes that traffic density leads to increased levels of pollutants, making it an important feature in above mentioned prediction models. The third hypothesis posits that time series models such as LSTM and ARIMA will provide more accurate predictions for sequential air quality data compared to regression models by capturing long-term temporal dependencies.

1.4 Scope

This project is limited to analyzing historical data collected between 2020 and 2024. The study focuses on three pollutants (PM2.5, PM10, and NO_2) across three cities: New York City (limited to Manhattan), Chicago (Cook County), and Los Angeles. The weather data was obtained from Open-Meteo, while pollution data was sourced from the U.S. Environmental Protection Agency (EPA). Traffic data, due to availability, was restricted to New York City and gathered from the MTA's bridge and tunnel crossing records. Upon reviewing the collected data, we also identified that the EPA dataset lacked PM10 and NO_2 concentration levels for New York County, limiting our analysis in that region to only PM2.5 levels.

The analysis centers on the month of January across the study period to reduce seasonal variability. The scope does not include real-time data ingestion, causal inference modeling, or analysis of industrial emissions, population density, or socioeconomic indicators. While the project presents an interactive dashboard for model results, deployment for public or operational use is outside the current scope. Future work may address these limitations and expand the framework to additional cities and data dimensions.

Chapter 2

Technical Approaches and Code UML

2.1 Development Environment

The results of our project were obtained using Python 3.12, running on a CPU-based environment without the need for dedicated GPU support. Development was primarily conducted using Jupyter Notebook and Google Colab, which provided flexible environments for experimentation and model tuning. Tableau Public was also utilized to create high-level visualizations of model outputs and pollutant trends, while GitHub served as our platform for version control and collaborative development.

A variety of Python libraries were employed throughout the project to support data processing, modeling, and visualization tasks:

- **Pandas, Numpy:** Used extensively for data cleaning, transformation, and manipulation.
- **Scikit-learn:** Provided machine learning utilities and models, including the Random Forest Regressor, as well as tools for feature scaling and validation.
- **Xgboost:** Utilized for its powerful and efficient gradient boosting regressor, which was the top-performing model in our regression tasks.
- **Tensorflow:** Enabled the development and training of LSTM models for time-series forecasting.
- **Statsmodels:** Used to implement the ARIMA model for statistical time-series forecasting.
- **Matplotlib, Seaborn:** Employed for data visualization and exploratory data analysis to better understand trends and model outputs.

2.2 Data Collection and Preparation

2.2.1 Dataset Sources

We collected information from several trusted sources to examine the relationship between traffic, weather, and air quality in large cities. Each dataset provides insightful details on various facets of the city's environment. Due to the diverse objectives of our project, the data collected for it came from three main sources.

Air Pollutant Information

The United States Environmental Protection Agency (EPA)[5] monitors pollutant levels across several counties in the country. Their official website provides a comprehensive pre-generated dataset, detailing various pollutants dating back to 1980. These datasets are categorized into average yearly summaries, daily levels, and hourly levels, further separated by pollutant type and the year in which they were collected. For this

project, we used fifteen different datasets focused on three pollutants: PM2.5 (non-FRM / FEM Mass), PM10 and NO_2, collected hourly from 2020 to 2024.

The datasets were downloaded in compact CSV format for easier access. Each dataset consisted of about 1-4 million rows and 24 columns. The columns included important details such as the county name, the coordinates of the monitoring sites, the parameter collected along with its unit of measurement, and the time of data collection.

Meteorological Data

The weather data was collected from Open-Meteo[6], an open-sourced weather API that compiles an comprehensive record historical weather data by compiling observations from weather stations, aircraft, buoys, radars and satellites. The platform features a user-friendly interface that allows users to select a location by name or coordinates, specify a desired time range, and choose from a variety of hourly weather variables such as temperature, air pressure, humidity and wind speed. Using this website, we generated three weather datasets for Chicago, New York, and Los Angeles, covering the years 2020 to 2024. We selected four key parameters for analysis: temperature, relative humidity, precipitation levels (including rain and ice), and wind speed (measured 100 meters above ground level). The resulting dataset contained over 43,000 rows and five columns, covering these parameters along with the corresponding timestamps.

Traffic Count

While traffic datasets are readily available for many U.S. cities, obtaining uniformly collected hourly traffic data across all three target cities proved to be a significant challenge. As a result, we narrowed our traffic analysis to New York City. The data was sourced from the New York State's official website, specifically the Metropolitan Transportation Authority (MTA) Bridges and Tunnels Hourly Crossings dataset[7]. This dataset provides detailed hourly records of bridge and tunnel crossings, categorized by facility, direction, vehicle class, and payment method (e.g., EZ-Pass or tolls). It is available in several formats, including CSV, RDF, XML, and JSON.

At the time of collection, the dataset included approximately 11 million rows and contained 11 columns with detailed information such as date, time, facility name, vehicle type, and traffic count. The dataset is updated regularly and offers a comprehensive view of vehicular movement across key points in New York City.

2.2.2 Data Accumulation

A total of 19 datasets were collected for this project. To facilitate efficient analysis, the required information from each dataset was extracted and merged into a single unified dataset. The pollutant data was structured such that each row corresponded to pollutant measurements recorded hourly for each specific region, New York County (primarily representing Manhattan), Cook County (part of the Chicago Metropolitan Area), and Los Angeles County (representing the greater Los Angeles area). Additionally, the analysis was limited to data from the month of January for each year between 2020 and 2024 to minimize seasonal variability.

Upon reviewing the dataset, it was observed that the EPA records for New York County lacked data for the PM10 and NO_2 levels, restricting the analysis for that region to only the PM2.5 concentration.

The weather data was merged with pollutant data based on shared fields: date, hour and the county name. For New York's traffic data, we filtered the dataset to retain only entries collected within or directly connected to New York County. The hourly traffic counts from the relevant bridge and tunnel facilities were aggregated to compute an average traffic count, which was then merged into the dataset for the New York entries.

The final aggregated dataset contained 11,160 rows and 14 columns, combining all the relevant information on air pollutant levels, meteorological conditions, and traffic volume necessary for our task.

2.2.3 Data Transformation

A series of pre-processing techniques were employed to enhance the quality and consistency of the dataset, ensuring its readiness for analysis and modeling:

- **Linear Interpolation:** Small gaps in the data were addressed using linear interpolation, maintaining temporal continuity without introducing abrupt changes.
- **Feature Scaling:** Standard scaling was applied to normalize the data, ensuring that all features followed a consistent distribution.
- **Feature Engineering:** To better capture the underlying patterns, new features were derived from the original data. These included rolling averages of pollutant concentrations and cyclic encoding of temporal variables, using sine and cosine transformations.

2.3 Implementation Details

The implementation involved developing, training, and evaluating multiple machine learning and time-series forecasting models to predict air pollution levels. Each stage of the process was carefully designed to ensure robustness and accuracy. The overall architecture of the data processing and modeling pipeline is illustrated in figure 2.1.

2.3.1 Regression Model

For the regression task, we implemented Random Forest and XGBoost using their respective libraries. These models were trained to predict hourly pollutant levels (PM2.5, PM10, NO_2) based on input features such as temperature, humidity, wind speed, precipitation, and traffic volume.

- **Random Forest:** Configured with a hundred decision trees and limited maximum depth to reduce overfitting. Hyperparameters were tuned using Randomized Search with five-fold cross-validation.
- **XGBoost:** Optimized using Grid Search to tune parameters such as learning rate, tree depth, and number of estimators. Regularization (L1 and L2) was also applied to improve generalizability.

2.3.2 Time-Series Forecasting

To capture temporal dependencies in pollution trends, we implemented two time series models, ARIMA and LSTM.

- **ARIMA:** Parameters (p, d, q) were selected via a manual grid search algorithm based on RMSE scores.
- **LSTM:** The model architecture included stacked LSTM layers with dropout (rate = 0.2) to mitigate overfitting. The model was trained using the Adam optimizer (learning rate = 0.001) for 20 epochs with early stopping based on validation loss.

2.3.3 Evaluation and Visualization

All models were trained using an 80/20 train-test split, using sequential data splitting in time series models, to maintain temporal integrity. The performance of these models were then evaluated using two key metrics: Coefficient of Determination (R^2) and Root Mean Squared Error (RMSE).

Model outputs and pollution trends were visualized using libraries, like matplotlib and seaborn. In addition, an dashboard was developed using Tableau Public to display model outputs and pollutant trends.

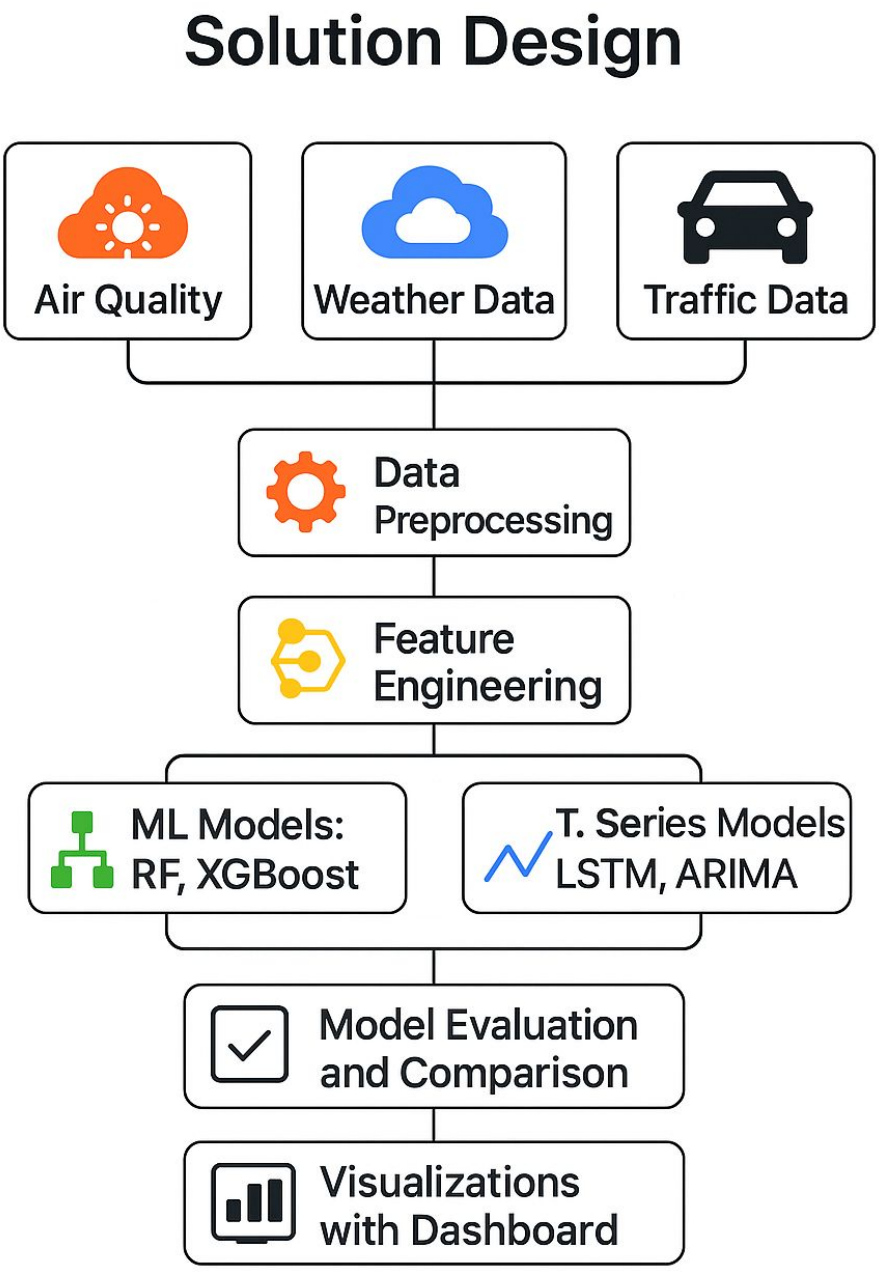

Figure 2.1: UML diagram showing the overall workflow

Chapter 3

Model Selection and Rationale

3.1 Model Selection Overview

This project aimed to forecast urban air quality by leveraging both structured data (weather, traffic) and sequential time-series data (pollution levels over time). Accordingly, we evaluated models that excel in either regression-based prediction or temporal pattern recognition. The final selection was made after comparative evaluation based on accuracy, interpretability, and suitability for the problem space.

3.2 Selected Models

XGBoost (Extreme Gradient Boosting)

XGBoost was selected for regression-based tasks due to its superior performance in structured data environments.

- **Reason for Selection:** Demonstrated the highest R^2 scores and lowest error rates across pollutants.
- **Key Advantages:**
  - High accuracy and speed due to optimized gradient boosting algorithm.
  - Robust to outliers and missing values.
  - Effective handling of large-scale datasets.
  - Built-in feature importance visualization.

LSTM (Long Short-Term Memory Networks)

LSTM models were selected for time-series forecasting due to their capacity to learn from sequential dependencies in data.

- **Reason for Selection:** Outperformed traditional models like ARIMA in capturing long-term pollutant trends.
- **Key Advantages:**
  - Excellent for modeling non-linear relationships in temporal data.
  - Adaptable to dynamic changes in pollutant levels.
  - Captures seasonality and lag effects effectively.

3.3 Other Models Considered

- **Random Forest:** Provided strong baseline performance in regression but lacked the capability to model sequential dependencies.
- **ARIMA:** Useful for short-term forecasting with stationary data but underperformed with complex urban pollution trends.

3.4 Final Justification

After conducting multiple rounds of evaluation using R^2, MAE, and RMSE as metrics, we selected:

- **XGBoost** as the best model for structured regression analysis involving weather and traffic inputs.
- **LSTM** as the optimal choice for time-series forecasting of pollutant trends across major cities.

These selections were validated using cross-validation strategies and benchmarked against other models to ensure reliability and robustness.

Chapter 4

Project Demonstration

4.1 Screenshots and Code Snippets

4.1.1 XGBoost

In [5]:

```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("/Users/shrutikute/Downloads/updated_air_quality_dataset.csv")

#  Map county names to city names (using "County Name" column)
county_to_city = {
    "Cook": "Chicago",
    "New York": "New York",
    "Los Angeles": "Los Angeles"
}

df["City"] = df["County Name"].map(county_to_city)  # Create a 'City' column

# Define features and target variable
features = ['temperature_2m (°C)', 'relative_humidity_2m (%)',
            'precipitation (mm)', 'wind_speed_100m (km/h)', 'Day', 'Hour']
target = 'PM2.5'

# Drop rows with missing target values
df = df.dropna(subset=[target])

#  Loop through each city and train a separate model
for city in ["New York", "Chicago", "Los Angeles"]:
    print(f"\n Training Model for {city} ")

    # Filter data for the specific city
    city_df = df[df["City"] == city]

    # Check if there's enough data
    if city_df.shape[0] < 50:  # Adjust threshold as needed
        print(f" Not enough data for {city}. Skipping...")
        continue

    # Extract features and target
    X = city_df[features]
    y = city_df[target]

    # Standardizing features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

# Splits Data into Training (80%) and Testing (20%)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Train XGBoost model with optimized hyperparameters
    xgb_regressor = xgb.XGBRegressor(
        objective='reg:squarederror',
        n_estimators=500,
```

Figure 4.1: Initial XGBoost model training and evaluation pipeline across cities.

```
        max_depth=8,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )

    xgb_regressor.fit(X_train, y_train)

    # Make predictions
    y_pred = xgb_regressor.predict(X_test)

    # Evaluate model performance
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f" {city} Model Performance:")
    print(f"   • Mean Absolute Error (MAE): {mae:.4f}")
    print(f"   • Mean Squared Error (MSE): {mse:.4f}")
    print(f"   • R-squared Score (R²): {r2:.4f}")
```

```
Training Model for New York
New York Model Performance:
  • Mean Absolute Error (MAE): 2.5263
  • Mean Squared Error (MSE): 15.3329
  • R-squared Score (R²): 0.5273

Training Model for Chicago
Chicago Model Performance:
  • Mean Absolute Error (MAE): 2.5957
  • Mean Squared Error (MSE): 15.1180
  • R-squared Score (R²): 0.5433

Training Model for Los Angeles
Los Angeles Model Performance:
  • Mean Absolute Error (MAE): 6.4970
  • Mean Squared Error (MSE): 128.3980
  • R-squared Score (R²): 0.6084
```

In [6]:

```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import numpy as np
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# Load dataset
df = pd.read_csv("/Users/shrutikute/Downloads/updated_air_quality_dataset.csv")
```

Figure 4.2: XGBoost model performance results for each city before hyperparameter tuning.

```python
#  Map County Names to City Names
county_to_city = {
    "Cook": "Chicago",
    "New York": "New York",
    "Los Angeles": "Los Angeles"
}
df["City"] = df["County_Name"].map(county_to_city)

#  Feature Engineering – Convert 'Day' to Cyclic Encoding
df['Day_sin'] = np.sin(2 * np.pi * df['Day'] / 7)
df['Day_cos'] = np.cos(2 * np.pi * df['Day'] / 7)

#  Feature Engineering – Rolling Average for PM2.5 (3-hour trend)
df["PM2.5_Rolling_Avg"] = df["PM2.5"].rolling(window=3, min_periods=1).mean()

# Define updated features
features = [
    'temperature_2m_(°C)', 'relative_humidity_2m_(%)',
    'precipitation_(mm)', 'wind_speed_100m_(km/h)',
    'Day_sin', 'Day_cos', 'PM2.5_Rolling_Avg', 'Hour'
]
target = 'PM2.5'

# Drop rows with missing values in features
df = df.dropna(subset=features)

#  Train a model for each city
for city in ["New York", "Chicago", "Los Angeles"]:
    print(f"\n Training Model for {city} ")

    # Filter data for the specific city
    city_df = df[df["City"] == city]

    # Check if there's enough data
    if city_df.shape[0] < 50:
        print(f" Not enough data for {city}. Skipping...")
        continue

    # Extract features and target
    X = city_df[features]
    y = city_df[target]

    # Standardizing features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Splits Data into Training (80%) and Testing (20%)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

    # Define parameter grid for Grid Search
    param_grid = {
        'n_estimators': [300, 500],
        'learning_rate': [0.01, 0.05, 0.1],
        'max_depth': [6, 8],
```

Figure 4.3: Feature engineering and grid search parameter tuning setup for XGBoost.

```python
        'subsample': [0.8, 1.0],
        'colsample_bytree': [0.8, 1.0],
        'reg_alpha': [0.01, 0.1],  # L1 Regularization
        'reg_lambda': [1, 10, 50]  # L2 Regularization
    }

    # Initialize XGBoost model
    xgb_regressor = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

    # Perform Grid Search CV
    tuner = GridSearchCV(
        xgb_regressor, param_grid, scoring='r2', cv=3, verbose=0, n_jobs=-1
    )

    tuner.fit(X_train, y_train)

    # Best parameters
    best_params = tuner.best_params_
    print(f" Best Hyperparameters for {city}: {best_params}")

    # Train model with best parameters found through grid search
    xgb_best = xgb.XGBRegressor(**best_params, objective='reg:squarederror', random_state=42)
    xgb_best.fit(X_train, y_train)

    # Make predictions
    y_pred = xgb_best.predict(X_test)

    # Evaluate model performance
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f" {city} Model Performance:")
    print(f"   • Mean Absolute Error (MAE): {mae:.4f}")
    print(f"   • Mean Squared Error (MSE): {mse:.4f}")
    print(f"   • R-squared Score (R²): {r2:.4f}")
```

Figure 4.4: GridSearchCV hyperparameter setup and best parameter output.

```
 Training Model for New York
 Best Hyperparameters for New York: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 6, 'n_estimator
s': 300, 'reg_alpha': 0.1, 'reg_lambda': 50, 'subsample': 0.8}
 New York Model Performance:
   • Mean Absolute Error (MAE): 1.5079
   • Mean Squared Error (MSE): 7.5891
   • R-squared Score (R²): 0.7660

 Training Model for Chicago
 Best Hyperparameters for Chicago: {'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimator
s': 500, 'reg_alpha': 0.01, 'reg_lambda': 10, 'subsample': 0.8}
 Chicago Model Performance:
   • Mean Absolute Error (MAE): 1.6409
   • Mean Squared Error (MSE): 7.0835
   • R-squared Score (R²): 0.7860

 Training Model for Los Angeles
 Best Hyperparameters for Los Angeles: {'colsample_bytree': 0.8, 'learning_rate': 0.05, 'max_depth': 6, 'n_estimat
ors': 500, 'reg_alpha': 0.1, 'reg_lambda': 50, 'subsample': 1.0}
 Los Angeles Model Performance:
   • Mean Absolute Error (MAE): 2.6565
   • Mean Squared Error (MSE): 53.2468
   • R-squared Score (R²): 0.8376
```

Figure 4.5: Final XGBoost model performance using best-tuned hyperparameters.

4.1.2 Random Forest

The following code demonstrates the use of Random Forest Regressor with feature engineering techniques like rolling averages and cyclic encoding. Hyperparameters were optimized using GridSearchCV for each city and pollutant type.

In [ ]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
final_df = pd.read_csv("updated_air_quality_dataset.csv")

# Convert the combined Date to a datetime column and create day_of_week
final_df['datetime'] = pd.to_datetime(final_df['Date'])
final_df['day_of_week'] = final_df['datetime'].dt.dayofweek

# Drop the original Date and datetime columns
final_df = final_df.drop(columns=['Date', 'datetime'])

# ------------------------------------------------
# Feature Engineering: Rolling Average
# ------------------------------------------------
# Sort by county and time components to ensure proper ordering for rolling computation
final_df = final_df.sort_values(['County Name', 'Year', 'Month', 'Day', 'Hour'])
# Compute a rolling average for temperature over a window of 3 (adjust window as needed)
final_df['temp_roll_avg'] = final_df.groupby('County Name')['temperature_2m (°C)'] \
                                    .rolling(window=3, min_periods=1).mean() \
                                    .reset_index(level=0, drop=True)

# ------------------------------------------------
# Feature Engineering: Cyclic Encoding for 'Day'
# ------------------------------------------------
# Convert the 'Day' column into cyclic features (assuming a maximum of 31 days)
final_df['Day_sin'] = np.sin(2 * np.pi * final_df['Day'] / 31)
final_df['Day_cos'] = np.cos(2 * np.pi * final_df['Day'] / 31)
# Drop the original 'Day' column as its information is now captured in the cyclic features
final_df = final_df.drop(columns=['Day'])

# Update the features list to include the new cyclic encoding and rolling average columns,
# and remove the dropped 'Day' column.
features = ['Year', 'Month', 'Hour', 'day_of_week', 'Day_sin', 'Day_cos',
            'temperature_2m (°C)', 'relative_humidity_2m (%)',
            'precipitation (mm)', 'wind_speed_100m (km/h)', 'temp_roll_avg']

target_pm25 = 'PM2.5'
target_pm10 = 'PM10'
target_no2 = 'NO2'
```

Figure 4.6: Initial setup for feature engineering and target selection for Random Forest. Includes rolling average and cyclic encoding.

```
    # Define the parameter grid for hyperparameter tuning
    param_grid = {
         'n_estimators': [50, 100, 200],
         'max_depth': [None, 10, 20, 30],
         'min_samples_split': [2, 5, 10],
         'min_samples_leaf': [1, 2, 4],
         'bootstrap': [True, False]
    }

    # Initialize the RandomForestRegressor
    rf = RandomForestRegressor(random_state=42)

    # Use GridSearchCV for hyperparameter tuning with 3-fold cross-validation
    grid_search = GridSearchCV(estimator=rf,
                               param_grid=param_grid,
                               cv=3,
                               n_jobs=-1,
                               scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    # Retrieve the best estimator
    best_rf = grid_search.best_estimator_

    # Predict on the test set and calculate performance metrics
    y_pred = best_rf.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"{target_name} Model - MSE: {mse:.3f}, R2: {r2:.3f}")
    return best_rf

##########################################################################################
# New York (Using PM2.5 and Average Traffic Density)
ny_df = final_df[final_df['County Name'] == 'New York']
# Include traffic density for New York along with our features
ny_features = features + ['Average Traffic Density']
X_ny = ny_df[ny_features]
y_ny_pm25 = ny_df[target_pm25]

print("-------------------------------------------------")
# Split the New York dataset into training and testing sets
X_ny_train, X_ny_test, y_ny_train, y_ny_test = train_test_split(X_ny, y_ny_pm25, test_size=0.2, random_state=42)
# Train and evaluate the PM2.5 model for New York with hyperparameter tuning
rf_pm25_ny = train_and_evaluate_rf(X_ny_train, X_ny_test, y_ny_train, y_ny_test, "New York PM2.5")

##########################################################################################
# Chicago (Using PM10, NO2, and PM2.5)
chicago_df = final_df[final_df['County Name'] == 'Cook']
X_chicago = chicago_df[features]
y_chicago_pm10 = chicago_df[target_pm10]
y_chicago_no2 = chicago_df[target_no2]
y_chicago_pm25 = chicago_df[target_pm25]
```

Figure 4.7: GridSearchCV hyperparameter tuning, training, and evaluation logic for Random Forest Regressor.

```
# Split the Chicago dataset into training and testing sets (for three target variables)
X_chicago_train, X_chicago_test, \
y_chicago_pm10_train, y_chicago_pm10_test, \
y_chicago_no2_train, y_chicago_no2_test, \
y_chicago_pm25_train, y_chicago_pm25_test = train_test_split(
    X_chicago, y_chicago_pm10, y_chicago_no2, y_chicago_pm25, test_size=0.2, random_state=42)

print("-------------------------------------------------")
# Train and evaluate the PM10 model for Chicago
rf_pm10_chicago = train_and_evaluate_rf(X_chicago_train, X_chicago_test, y_chicago_pm10_train, y_chicago_pm10_tes
# Train and evaluate the NO2 model for Chicago
rf_no2_chicago = train_and_evaluate_rf(X_chicago_train, X_chicago_test, y_chicago_no2_train, y_chicago_no2_test,
# Train and evaluate the PM2.5 model for Chicago
rf_pm25_chicago = train_and_evaluate_rf(X_chicago_train, X_chicago_test, y_chicago_pm25_train, y_chicago_pm25_tes

##########################################################################################
# Los Angeles (Using PM10, NO2, and PM2.5)
los_angeles_df = final_df[final_df['County Name'] == 'Los Angeles']
X_la = los_angeles_df[features]
y_la_pm10 = los_angeles_df[target_pm10]
y_la_no2 = los_angeles_df[target_no2]
y_la_pm25 = los_angeles_df[target_pm25]

# Split the Los Angeles dataset into training and testing sets (for three target variables)
X_la_train, X_la_test, \
y_la_pm10_train, y_la_pm10_test, \
y_la_no2_train, y_la_no2_test, \
y_la_pm25_train, y_la_pm25_test = train_test_split(
    X_la, y_la_pm10, y_la_no2, y_la_pm25, test_size=0.2, random_state=42)

print("-------------------------------------------------")
# Train and evaluate the PM10 model for Los Angeles
rf_pm10_la = train_and_evaluate_rf(X_la_train, X_la_test, y_la_pm10_train, y_la_pm10_test, "Los Angeles PM10")
# Train and evaluate the NO2 model for Los Angeles
rf_no2_la = train_and_evaluate_rf(X_la_train, X_la_test, y_la_no2_train, y_la_no2_test, "Los Angeles NO2")
# Train and evaluate the PM2.5 model for Los Angeles
rf_pm25_la = train_and_evaluate_rf(X_la_train, X_la_test, y_la_pm25_train, y_la_pm25_test, "Los Angeles PM2.5")
```

```
-------------------------------------------------
New York PM2.5 Model - MSE: 14.173, R2: 0.563
-------------------------------------------------
Chicago PM10 Model - MSE: 50.844, R2: 0.770
Chicago NO2 Model - MSE: 13.118, R2: 0.787
Chicago PM2.5 Model - MSE: 12.501, R2: 0.622
-------------------------------------------------
Los Angeles PM10 Model - MSE: 116.538, R2: 0.357
Los Angeles NO2 Model - MSE: 24.609, R2: 0.813
```

Figure 4.8: City-specific model training and final model performance summary for Random Forest on PM2.5, PM10, and NO2.

4.1.3 ARIMA

The ARIMA model was implemented for time-series forecasting of pollutants using annual data split into training and testing sets. A manual grid search was performed to find optimal (p, d, q) parameters, followed by RMSE-based evaluation and visual comparison of actual vs predicted pollutant levels.

In [2]:
```python
def grid_search_arima(train, p_values, d_values, q_values):
    """Perform a grid search to find the best ARIMA hyperparameters."""
    best_score, best_cfg= float("inf"), None
    for p, d, q in itertools.product(p_values, d_values, q_values):
        try:
            model= ARIMA(train, order=(p, d, q)).fit()
            predictions= model.predict(start=len(train), end=len(train) + len(test) - 1, dynamic=False)
            error= mean_squared_error(test, predictions)
            if error<best_score:
                best_score, best_cfg= error, (p, d, q)
        except:
            continue
    return best_cfg
```

In [4]:
```python
df= pd.read_csv("updated_air_quality_dataset.csv")

df['Date']= pd.to_datetime(df['Date'] + ' ' + df['Hour'].astype(str) + ':00:00')
df.set_index('Date', inplace=True)

counties= ["Cook", "Los Angeles", "New York"]
target_vars= ["PM2.5", "PM10", "NO2"]

p_values= range(0, 3)
d_values= range(0, 3)
q_values= range(0, 3)
```

Figure 4.9: ARIMA code setup including parameter grid creation, and model initialization.

In [9]:
```python
results= {}

for county in counties:
    county_df= df[df["County Name"] == county]
    years= county_df["Year"].unique()
    print("-"*100)
    print(f"Training ARIMA for {county if county!="Cook" else "Chicago"}")

    for year in years:
        yearly_df= county_df[county_df["Year"] == year]
        yearly_df= yearly_df.sort_values(["Month", "Day", "Hour"])  # Ensuring time order

        for var in target_vars:
            if yearly_df[var].isna().sum() > 0.2 * len(yearly_df): #Ensuring there is enough data
                continue

            data = yearly_df[var]
            train_size= int(len(data) * 0.8)
            train, test= data[:train_size], data[train_size:]

            best_params= grid_search_arima(train, p_values, d_values, q_values) #Applying Grid Search
            print(f"\n{var}-{year}")
            if best_params:
                print(f"Best ARIMA hyperparameters: {best_params}")
                model= ARIMA(train, order=best_params).fit()
                predictions= model.predict(start=len(train), end=len(train) + len(test) - 1, dynamic=False)

                rmse= np.sqrt(mean_squared_error(test, predictions))
                print(f"Test RMSE: {rmse:.3f}")
                results[(county if county!="Cook" else "Chicago", year, var)]= (best_params, rmse)

                #Generating the graph to visualize the results
                plt.figure(figsize=(12,6))
                plt.plot(train.index, train, label="Actual (Train)", color="blue", linewidth=2)
                plt.plot(test.index, test, label="Actual (Test)", color="green", linewidth=2)
                plt.plot(test.index, predictions, label="Predicted (Test)", color="red", linewidth=2)
                plt.xlabel("Time")
                plt.ylabel("Pollutant Level")
                plt.title("Actual vs Predicted Values (Train & Test)")
                plt.legend()
                plt.show()
            else:
                print(f"No suitable ARIMA model found")
```

Figure 4.10: ARIMA model evaluation across cities and years using grid search and RMSE scoring.

```
Training ARIMA for Chicago

PM2.5-2020
Best ARIMA hyperparameters: (1, 2, 1)
Test RMSE: 4.715
```

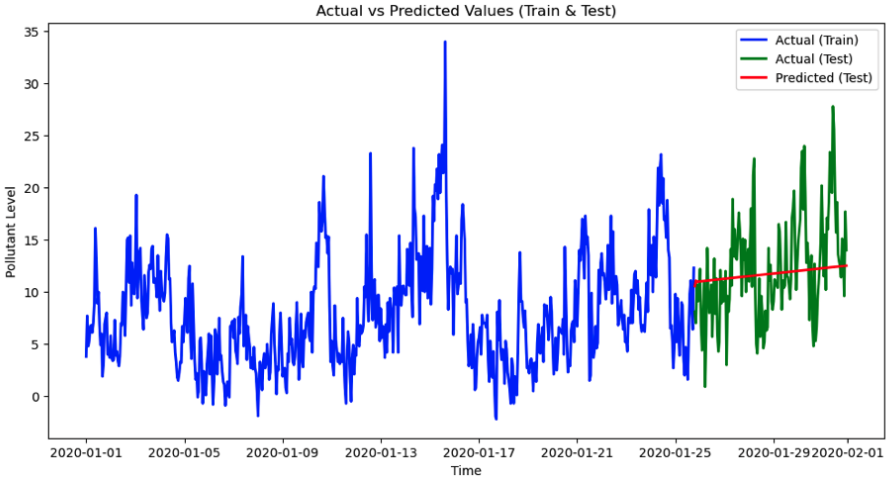

Figure 4.11: Visualization of actual vs predicted pollutant levels using the ARIMA model for PM2.5 in Chicago (2020).

4.1.4 LSTM

The LSTM model was used for time-series forecasting with hourly pollutant data, leveraging sequence-based learning. Preprocessing included normalization, missing value handling, and reshaping for sequence input. The final model output was plotted against ground truth.

```python
# Load the dataset
data = pd.read_csv('updated_air_quality_dataset.csv')

# Function to preprocess data, train LSTM model for a given city and target variable, and plot results
def train_lstm_for_city(data, city_name, target_column):
    # Filter data for the city
    city_data = data[data['County Name'] == city_name]

    # Select relevant columns
    if city_name == 'New York':
        city_data = city_data[['Date', 'Hour', target_column, 'temperature_2m (°C)', 'relative_humidity_2m (%)',
                               'precipitation (mm)', 'wind_speed_100m (km/h)', 'Average Traffic Density']]
    else:
        city_data = city_data[['Date', 'Hour', target_column, 'temperature_2m (°C)', 'relative_humidity_2m (%)',
                               'precipitation (mm)', 'wind_speed_100m (km/h)']]

    # Convert Date to datetime
    city_data['Date'] = pd.to_datetime(city_data['Date'])

    # Sort by Date and Hour
    city_data = city_data.sort_values(by=['Date', 'Hour'])

    # Handle missing values if any
    city_data = city_data.fillna(method='ffill')

    # Normalize the data
    scaler = MinMaxScaler()
    if city_name == 'New York':
        cols_to_scale = [target_column, 'temperature_2m (°C)', 'relative_humidity_2m (%)',
                         'precipitation (mm)', 'wind_speed_100m (km/h)', 'Average Traffic Density']
    else:
        cols_to_scale = [target_column, 'temperature_2m (°C)', 'relative_humidity_2m (%)',
                         'precipitation (mm)', 'wind_speed_100m (km/h)']
    city_data_scaled = scaler.fit_transform(city_data[cols_to_scale])

    # Create sequences for LSTM
    def create_sequences(data, seq_length):
        xs, ys = [], []
        for i in range(len(data) - seq_length):
            x = data[i:(i + seq_length)]
            y = data[i + seq_length, 0]  # Target column is the first one
            xs.append(x)
            ys.append(y)
        return np.array(xs), np.array(ys)
```

Figure 4.12: LSTM model preprocessing pipeline: selecting, scaling, and sequencing input features.

```
seq_length = 24  # 24 hours
X, y = create_sequences(city_data_scaled, seq_length)

# Split the data into training and testing sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Build the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=20, verbose=0)

# Make predictions
predictions = model.predict(X_test)

# Inverse transform the predictions to original scale
dummy_array = np.zeros((len(predictions), city_data_scaled.shape[1]))
dummy_array[:, 0] = predictions.flatten()  # Fill the target column with predictions
predictions_original = scaler.inverse_transform(dummy_array)[:, 0]

# Inverse transform the actual test values
dummy_array[:, 0] = y_test.flatten()  # Fill the target column with actual values
y_test_original = scaler.inverse_transform(dummy_array)[:, 0]

# Calculate MSE, MAE, and R2
mse = mean_squared_error(y_test_original, predictions_original)
mae = mean_absolute_error(y_test_original, predictions_original)
r2 = r2_score(y_test_original, predictions_original)

# Print the metrics
print(f'\n{city_name} - {target_column} Metrics:')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Mean Absolute Error (MAE): {mae}')
print(f'R-squared (R²): {r2}')

# Print some predictions
for i in range(5):
    print(f'Predicted: {predictions_original[i]:.2f}, Actual: {y_test_original[i]:.2f}')

# Inverse transform the training target values for plotting
dummy_array_train = np.zeros((len(y_train), city_data_scaled.shape[1]))
dummy_array_train[:, 0] = y_train.flatten()
y_train_original = scaler.inverse_transform(dummy_array_train)[:, 0]
```

Figure 4.13: Model building, training, prediction, and performance evaluation in LSTM.

```
    # Print some predictions
    for i in range(5):
        print(f'Predicted: {predictions_original[i]:.2f}, Actual: {y_test_original[i]:.2f}')

    # Inverse transform the training target values for plotting
    dummy_array_train = np.zeros((len(y_train), city_data_scaled.shape[1]))
    dummy_array_train[:, 0] = y_train.flatten()
    y_train_original = scaler.inverse_transform(dummy_array_train)[:, 0]

    # Plot the results:
    # X-axis: zaman adımları; ilk kısımda eğitim verileri, son %20'de test (gerçek ve tahmin) verileri
    plt.figure(figsize=(12, 6))
    # Plot training target values
    plt.plot(range(len(y_train_original)), y_train_original, label='Train Target')
    # Plot test actual values
    test_index = range(len(y_train_original), len(y_train_original) + len(y_test_original))
    plt.plot(test_index, y_test_original, label='Test Actual')
    # Plot test predicted values
    plt.plot(test_index, predictions_original, label='Test Predicted')
    plt.xlabel('Time Step')
    plt.ylabel(target_column)
    plt.title(f'{city_name} - {target_column} Prediction')
    plt.legend()
    plt.show()

# List of cities and target columns
cities = ['Cook', 'Los Angeles', 'New York']
target_columns = {
    'Cook': ['PM2.5', 'PM10', 'NO2'],
    'Los Angeles': ['PM2.5', 'PM10', 'NO2'],
    'New York': ['PM2.5']
}

# Train models for each city and target column and plot graphs
for city in cities:
    for target in target_columns[city]:
        train_lstm_for_city(data, city, target)
```

```
24/24 [==============================] - 0s 3ms/step

Cook - PM2.5 Metrics:
Mean Squared Error (MSE): 8.450241752806534
Mean Absolute Error (MAE): 2.3595059298552776
R-squared (R²): 0.7530920850331789
Predicted: 8.70, Actual: 12.90
Predicted: 11.15, Actual: 7.30
Predicted: 9.75, Actual: 9.80
Predicted: 10.31, Actual: 7.60
Predicted: 9.36, Actual: 6.70
```

Figure 4.14: Final model loop for multi-city training and generation of prediction metrics.

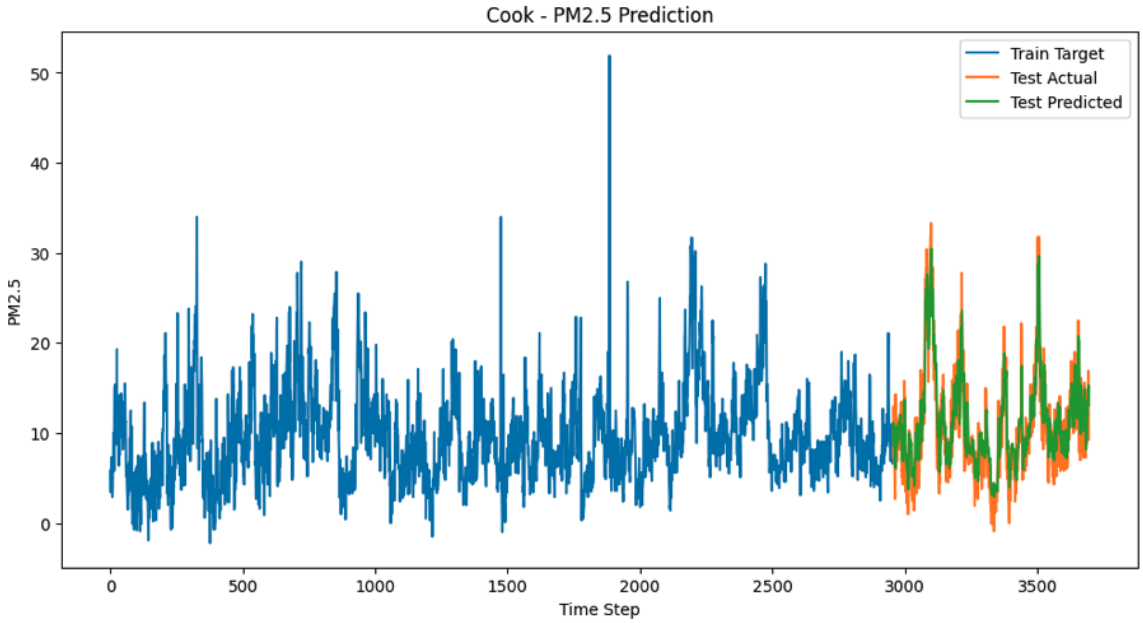

Figure 4.15: Prediction plot for PM2.5 in Cook County using LSTM — showing test actual vs predicted.

4.2 Results

This project employed a hybrid approach combining regression and time-series forecasting models to predict pollutant concentrations (PM2.5, PM10, NO_2) in three major U.S. cities—*New York City, Chicago, and Los Angeles*—based on weather and traffic data.

Regression Models

We utilized Random Forest and XGBoost regressors to model the relationship between pollutants and features such as temperature, humidity, wind speed, precipitation, and traffic volumes (available for NYC only).

- **Random Forest Regressor** performed reasonably well but struggled with capturing temporal dependencies in the data.
- **XGBoost Regressor** consistently outperformed Random Forest with better accuracy and generalization:
  - $R^2 = 0.9019$ for NO_2 in Chicago
  - $R^2 = 0.8557$ for PM10 in Los Angeles
  - RMSE as low as 4.35 across several pollutant-city combinations

Table 4.1: Performance Comparison: Random Forest vs XGBoost

Model	**MSE**	R^2	**RMSE**
Random Forest Regressor	38.630	0.652	5.635
XGBoost Regressor	**22.378**	**0.835**	**4.352**

Time-Series Forecasting

We implemented both LSTM and ARIMA models to forecast hourly pollutant levels.

- **LSTM (Long Short-Term Memory)** significantly outperformed ARIMA in capturing non-linear, temporal dependencies:
  - LSTM RMSE: 4.43 (PM2.5)
  - ARIMA RMSE: 9.09 (PM2.5)
- LSTM was especially effective in Chicago, where forecasted PM2.5 values closely matched actual observations.

These results validated our model selection strategy, with XGBoost excelling at regression tasks and LSTM proving most suitable for sequential prediction. Full implementation details and source code are accessible via our GitHub repository[1]

4.3 Insights (Interactive Dashboard)

To improve the accessibility and interpretability of our findings, we developed an interactive Tableau dashboard that visualizes pollutant trends, model predictions, and feature correlations.

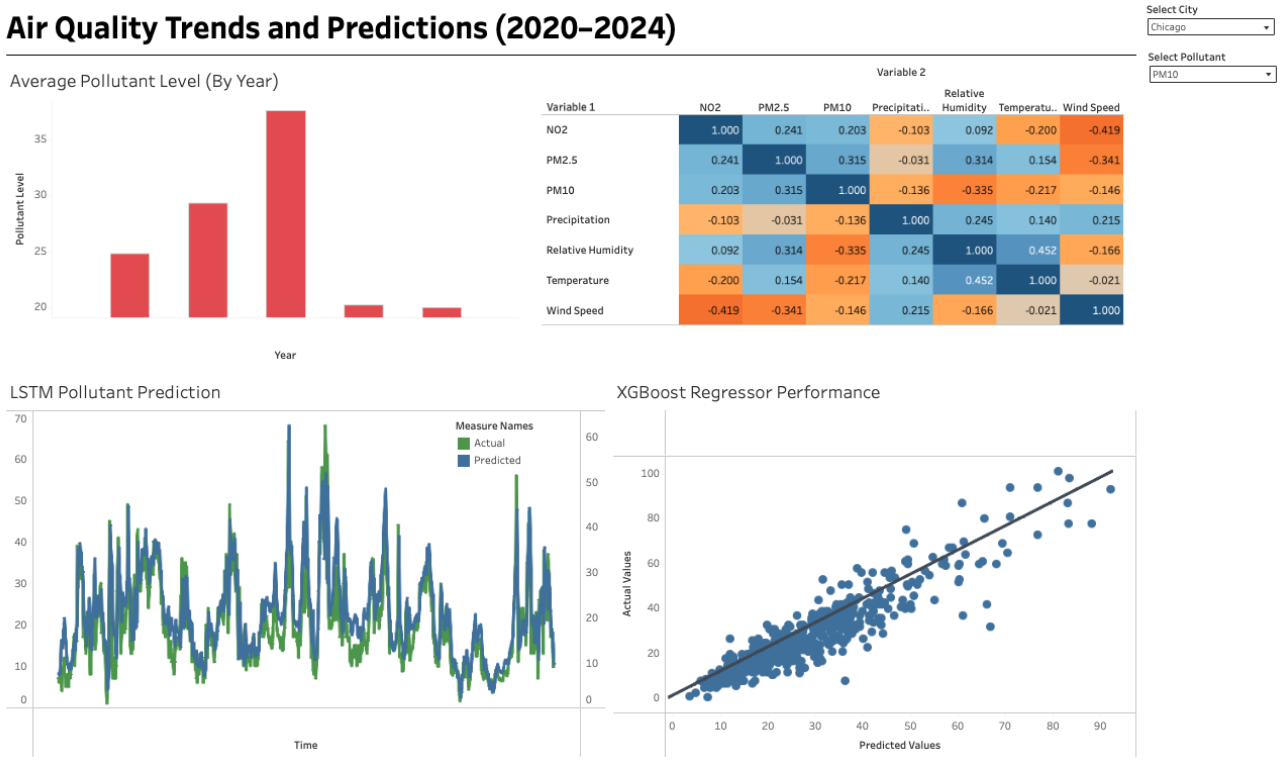

Figure 4.16: Interactive Tableau Dashboard visualizing pollutant levels (PM10, PM2.5, NO2) across major cities (Chicago, NYC, LA). Users can select specific pollutants and cities using the top-right filters.

[1] https://github.com/Shrutikute09/Capstone_2025

Dashboard Overview

The dashboard features dynamic filtering based on:

- **City:** New York City, Chicago, Los Angeles
- **Pollutant:** PM2.5, PM10, NO_2

Each component updates based on the selected parameters, enabling targeted exploration of temporal and spatial trends.

Key Components

- **Average Pollutant Levels (2020–2024):** Line charts display annual averages by pollutant and city, highlighting long-term patterns.
- **LSTM Forecasting Visualization:** Actual vs. predicted pollutant values are plotted for a visual performance evaluation. Chicago's PM2.5 forecasts showed strong alignment with ground truth.
- **XGBoost Regression Output:** Predicted vs. observed pollutant levels are visualized to assess regression model accuracy. NYC PM2.5 predictions were especially precise.
- **Correlation Matrix:** Pearson correlations between environmental variables and pollutants are shown via heatmap.
  - PM2.5 and PM10 exhibit moderate positive correlation.
  - Temperature and wind speed generally show negative correlation with pollutants.
  - Precipitation has weak to moderate correlation depending on location and pollutant.

Users can access the dashboard online to explore these insights interactively[2]

Insights Extracted

- **New York City:** PM2.5 levels strongly correlate with hourly traffic volumes. Model performance improved significantly with traffic data integration.
- **Chicago:** NO_2 is influenced by meteorological variables, particularly wind speed and temperature.
- **Los Angeles:** PM10 levels are highly variable and responsive to dry, hot weather conditions. XGBoost effectively captured this variation.

Overall, the dashboard offers a comprehensive, interactive summary of our analysis. It enables users to explore urban air quality patterns, model predictions, and environmental dependencies through an intuitive, visual interface. This tool supports evidence-based insights for researchers, city planners, and environmental policymakers.

[2]The dashboard is available at `https://public.tableau.com/app/profile/atreyo.das/viz/Capstone_Air_Quality_17440009911560/Dashboard1`.

Chapter 5

Discussion and Future Work

5.1 Discussion

This study aimed to explore the relationship between urban air pollution and external variables such as meteorological conditions and traffic volume using a comprehensive machine learning approach. Our multi-model framework, including both regression and time-series algorithms, enabled us to evaluate the predictive power of various environmental factors and assess temporal dynamics in pollutant behavior.

The results showed that the XGBoost regressor consistently outperformed Random Forest in terms of both R^2 scores and error metrics (MSE, RMSE). Its superior performance can be attributed to its robustness to overfitting, ability to handle missing data effectively, and its built-in feature importance evaluation. These properties made it especially valuable in handling the diverse and partially incomplete datasets encountered in this project. Moreover, XGBoost provided interpretability through its feature importance scores, which revealed that variables such as wind speed, temperature, and time-of-day were particularly influential in determining pollution levels, especially in cities like Los Angeles and Chicago.

For time-series forecasting, LSTM networks demonstrated significant improvements over ARIMA models. The LSTM architecture captured long-term temporal dependencies more effectively, producing predictions that closely followed actual pollutant patterns. For example, in New York City, LSTM achieved a notable RMSE of 4.43 compared to ARIMA's 9.09 for $PM_{2.5}$, highlighting its capacity to model non-linear, sequential data. However, the deep learning model's performance was highly dependent on appropriate data preprocessing and hyperparameter tuning, as evidenced by lower R^2 scores in locations with higher data noise or limited training data, such as PM_{10} in Los Angeles.

Despite these successes, the project faced several **challenges and limitations**:

- **Limited traffic data**: Due to data availability, traffic patterns were only incorporated for New York City. As a result, our models for Chicago and Los Angeles relied solely on meteorological data. This constrained our ability to fully understand the interaction between vehicular density and pollution across multiple cities.
- **Incomplete pollutant coverage**: Some pollutant datasets, especially for PM_{10} and NO_2 in New York, were either missing or incomplete, which limited the generality of our models and forced us to reduce the scope of analysis to $PM_{2.5}$ for that city.
- **Model limitations and variance**: While XGBoost and LSTM performed well overall, their predictive power varied across different locations and pollutants. The LSTM model, in particular, required substantial training time and was prone to overfitting if hyperparameters were not carefully optimized.

Another important insight was the value of feature engineering in boosting model performance. Time-related features (e.g., hour of the day, day of the week) encoded using cyclical transformations, along with rolling averages of pollutants, contributed significantly to improving the models' predictive accuracy.

Additionally, the use of visualization tools such as Tableau enhanced the interpretability of our results, allowing both technical and non-technical audiences to interact with the data. This was essential for effectively communicating key trends and model outputs.

Overall, this project demonstrated that while machine learning and deep learning models hold great promise for environmental forecasting, their effectiveness is heavily dependent on data quality, preprocessing, and thoughtful feature engineering. The insights gained from this study can inform future urban planning and public health decisions, but must be further validated and expanded for broader application.

5.2 Future Work

While the findings of this study offer valuable insights into the relationship between environmental conditions and urban air quality, there are several directions in which the project can be extended or improved in future iterations.

1. **Expanding Traffic Data Integration Across Cities**
   Due to data availability, traffic information was only incorporated for New York City. In future studies, acquiring or generating hourly traffic datasets for Chicago and Los Angeles would provide a more holistic view of vehicular contributions to urban pollution. Techniques such as traffic simulations or crowdsourced GPS data (e.g., Waze, Google Maps API) could be explored to fill existing gaps, enabling more equitable model comparisons across urban areas.

2. **Enhancing Spatial Resolution Using Geospatial Data**
   Our current approach relied on aggregated city-level data. Future efforts can improve spatial granularity by integrating Geographic Information System (GIS) layers, satellite-derived pollution estimates (e.g., from NASA's MODIS/AIRS), and census-based demographic information. This would allow for neighborhood-level pollution modeling, which is essential for understanding environmental justice implications and targeting high-risk zones more effectively.

3. **Deploying Real-Time Forecasting Pipelines**
   Building upon the trained models, future work can involve developing real-time prediction systems that pull updated weather and traffic data via APIs, perform automatic preprocessing, and update predictions on an hourly or daily basis. Integration with interactive dashboards could allow city officials and citizens to view near real-time pollution forecasts and take precautionary actions, such as issuing health alerts or adjusting commuting routes.

4. **Exploring More Advanced and Interpretable Models**
   While XGBoost and LSTM yielded strong results, emerging time-series models like Temporal Fusion Transformers (TFT) and attention-based encoder-decoder networks may offer improvements in both performance and interpretability. Additionally, model-agnostic interpretation frameworks such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) can be integrated to better understand model decisions and support data-driven policy recommendations.

5. **Evaluating Impact on Health Outcomes**
   A valuable extension of this work would be to correlate predicted pollution levels with hospital admission rates or public health indicators, thereby quantifying the direct human impact of air quality fluctuations. Accessing anonymized health records or published epidemiological data could support this line of research, creating a stronger foundation for health-informed urban policy.

6. **Simulating Policy Interventions**
   Scenario modeling can be introduced to simulate the effects of hypothetical interventions such as traffic restrictions, green corridor expansion, or industrial regulation changes. By adjusting input features (e.g., traffic volume or wind blockage) and observing output changes in pollutant levels, city planners could assess the potential impact of new policies before implementation.

7. **Improving Model Robustness and Generalization**
   Further research should focus on making models more robust to noise, missing values, and city-specific variance. Incorporating Bayesian modeling or ensemble learning across diverse model families may help address overfitting and improve generalization across different urban environments.

In conclusion, the current project lays a strong foundation for data-driven environmental forecasting and urban policy support. However, extending the temporal, spatial, and methodological dimensions will be crucial in building more powerful, equitable, and actionable air quality intelligence systems in the future.

Chapter 6

Conclusion

This project set out to analyze the impact of weather conditions and traffic density on air pollution levels in three major U.S. cities—New York, Chicago, and Los Angeles—using advanced machine learning and time-series forecasting models. By focusing on three major pollutants (PM2.5, PM10, and NO_2), we aimed to build accurate and generalizable models that could predict pollutant concentrations and provide meaningful insights for public health and urban planning.

To achieve this, we employed a combination of regression models (XGBoost and Random Forest) and time-series models (LSTM and ARIMA). Extensive data preprocessing, including feature engineering (rolling averages, cyclic encoding), normalization, and rigorous hyperparameter tuning, ensured high-quality inputs for model training and validation.

Key Findings

- **XGBoost** consistently outperformed other models in regression tasks, achieving the highest R^2 scores and lowest error rates across all three cities.
- **LSTM** proved to be the most effective time-series forecasting model, capturing long-term dependencies in pollutant trends.
- **Traffic volume** emerged as a significant predictor of pollution, especially in New York City, confirming the hypothesis about urban mobility's impact on air quality.
- **Meteorological conditions** such as temperature, wind speed, and precipitation played a vital role in pollution dispersion and concentration.
- **ARIMA** was effective for short-term forecasting but less suitable for complex or long-term pollution trends.

Overall Contribution

This study highlights how machine learning and deep learning methods can offer robust predictions for air quality in diverse urban environments. The project not only provided accurate models but also reinforced the importance of integrating weather and traffic data for predictive environmental analysis.

Future Work

Future iterations of this project may include:

- Expanding the dataset to include more cities and additional pollutant types.
- Integrating real-time air quality and traffic data for dynamic forecasting.

- Exploring hybrid models that combine statistical and neural network-based approaches.
- Deploying interactive dashboards to allow real-time monitoring and visualization for policy makers and the public.

In summary, this project lays the groundwork for scalable, interpretable, and actionable air quality prediction tools. These models have the potential to support smarter environmental policies and improve public health outcomes in metropolitan areas.

Bibliography

[1] World Health Organization. 2025. Air quality, energy and health: Health impacts. Retrieved: April 16, 2025.

[2] World Health Organization. 2025. Air pollution. Retrieved: April 16, 2025.

[3] Mauro Castelli, Fabiana Martins Clemente, Aleš Popovič, Sara Silva, and Leonardo Vanneschi. 2020. A Machine Learning Approach to Predict Air Quality in California. Complexity, 2020, 8049504, 23 pages, 2020.

[4] M. S. K. Abhilash, Amrita Thakur, Deepa Gupta, and B. Sree-vidya. 2018. Time Series Analysis of Air Pollution in Bengaluru Using ARIMA Model. In Ambient Communications and Computer Systems, pages 413–426, Singapore, Springer Singapore.

[5] US Environmental Protection Agency. 2025. Air Quality System Data Mart [internet database]. Accessed: January 28, 2025.

[6] Patrick Zippenfenig. 2023. Open-Meteo.com Weather API.

[7] Metropolitan Transportation Authority. 2025. MTA Bridges and Tunnels Hourly Crossings: Beginning 2019. Accessed: February 10, 2025.