MECE4520_FinalReport_Group7

December 16, 2023

1 Introduction

This report is the culmination of the MECE 4520 final project, with the purpose of employing quantitative methods discussed in the course. Our team analyzed the air-quality data posted by the UCI ML Repository, specifically monitoring the pollutants of an Italian city. The dataset encompasses the concentration of five pollutants, including CO, Non-Metallic Hydrocarbons, Benzene, NOx and NO2. These were generated by five metal oxide sensors embedded in an Air Quality Chemical Multi-sensor Device and are located on the field in a significantly polluted area, at road level. The data spans from March 2004 to February 2005.

2 Objectives

1. Monitor the pollutants concentration in real-time and determine the pollution level.
2. Serve as a predictive learning model and simulation for future pollution forecasting.

Our project aims to contribute to environmental research, providing a rich dataset for studying air pollution trends and their impact on urban environments.

3 Data Processing

To enhance our analysis and provide clearer insights, a comprehensive data preprocessing strategy should be implemented, focusing initially on data cleaning. This includes the identification and rectification of missing values, which were denoted by a placeholder value of -200, which is designed for the purpose of distinguishing them clearly from the actual data distribution. These values were earmarked for substitution with interpolated data during the training phase of our Neural Network. Originally, this dataset was designed for calibrating air pollutant sensors, leading to the inclusion of some extraneous data. To refine our feature set, we extracted and expanded the original date and time column into separate features: “Hour,” “Day of Week,” and “Month.” This transformation enabled us to enrich our dataset with more relevant features, facilitating a more nuanced analysis. Our aim was to discern patterns in air pollutant distribution over time, leveraging these newly derived temporal features.

[ ]:
```
import pandas as pd
def load_and_preprocess_data(file_path):
    """
    Load the Air Quality dataset and perform preprocessing steps including:
    - Identifying missing values represented as -200
```

```
    - Feature engineering (combining date and time, extracting useful time␣
↪components)
        """
    air_quality_data = pd.read_excel(file_path)
    # Identify -200 values; these could represent sensor malfunctions or truly␣
↪missing data
    # Combine Date and Time into a single datetime column
    # Ensure Time is converted to string format before concatenation
    air_quality_data['DateTime'] = pd.to_datetime(air_quality_data['Date'].
↪astype(str) + ' ' + air_quality_data['Time'].astype(str))

    # Extract hour, day of the week, and month from the DateTime column
    air_quality_data['Hour'] = air_quality_data['DateTime'].dt.hour
    air_quality_data['DayOfWeek'] = air_quality_data['DateTime'].dt.dayofweek
    air_quality_data['Month'] = air_quality_data['DateTime'].dt.month

    air_quality_data.drop(['Date', 'Time'], axis=1, inplace=True)

    return air_quality_data

def save_to_csv(df, output_path):
        df.to_csv(output_path, index=False)
    print(f"Data saved to {output_path}")

input_file_path = 'AirQualityUCI.xlsx'
output_file_path = 'processed_air_quality_data.csv'
processed_data = load_and_preprocess_data(input_file_path)
save_to_csv(processed_data, output_file_path)
```

```
Data saved to processed_air_quality_data.csv
```

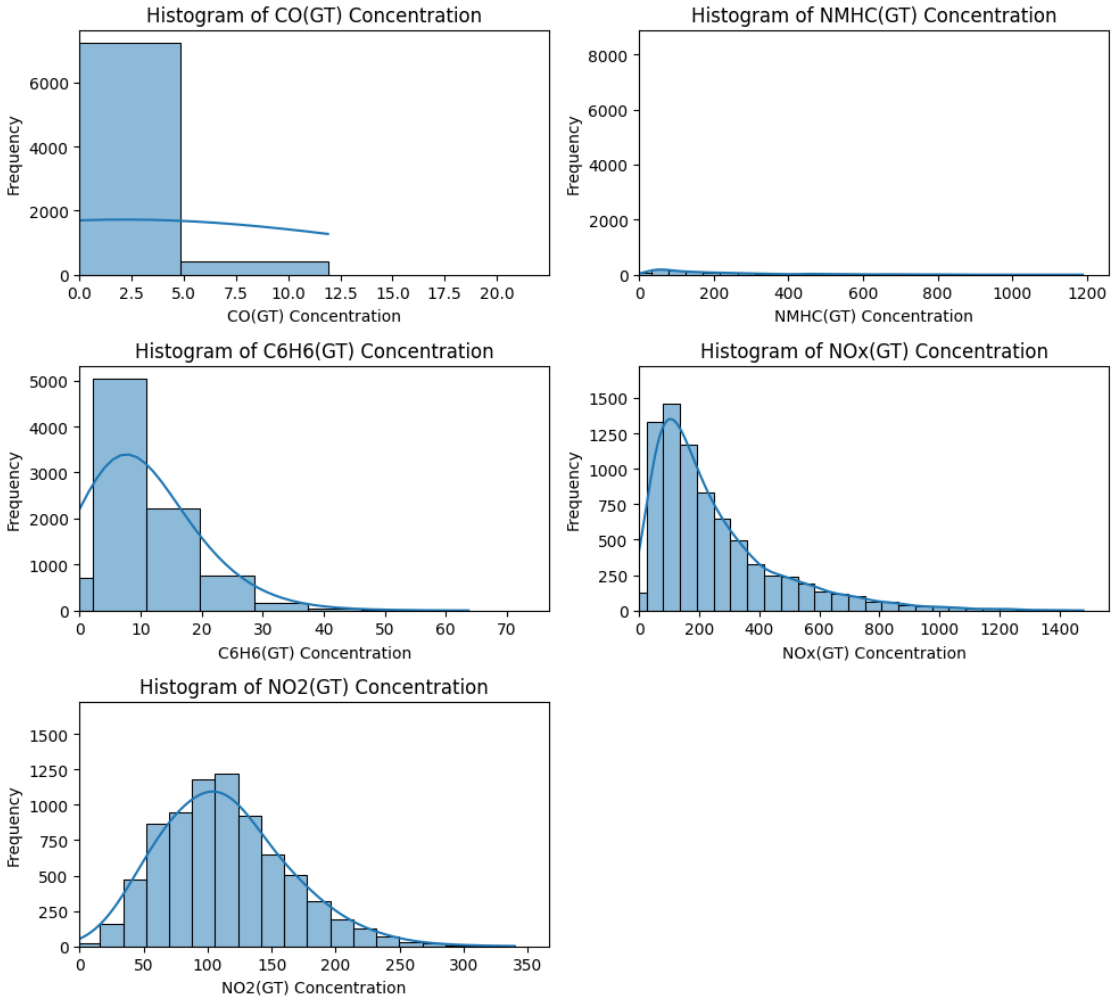
4 Exploratory Data Analysis

Exploratory Data Analysis is an essential phase in analyzing a dataset from a comprehensive view. This process encompasses vital features, including understanding data structure, effective data visualization, and hypothesis formulation. In this section, our team prioritized data visualization to grasp a fundamental understanding of the overall dataset. Common visualized methods that served as EDA procedures are histograms and boxplots, and their detailed explanations are generalized as follows:

4.1 Histogram

The histogram is a bar chart displaying the data's frequency distribution. It offers a detailed examination of the frequency distribution of pollutant concentration, which provides valuable insight into the data's overall distribution pattern. The procedural steps via the Colab platform are outlined below, we initially import all necessary libraries and load the dataset. Subsequently, we identify the five key pollutants and generate a histogram for each dataset.

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
file_path = 'processed_air_quality_data.csv'
df = pd.read_csv(file_path)
pollutants= ['CO(GT)', 'NMHC(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']
num_rows = int(np.ceil(len(pollutants) / 2))
fig, axes = plt.subplots(num_rows, 2, figsize=(10, num_rows * 3))
for idx, pollutant in enumerate(pollutants):

    row_idx = idx // 2
    col_idx = idx % 2
    ax = axes[row_idx, col_idx]
    sns.histplot(df[pollutant], bins=30, kde=True, ax=ax)
    ax.set_title(f'Histogram of {pollutant} Concentration')
    ax.set_xlabel(f'{pollutant} Concentration')
    ax.set_ylabel('Frequency')
    ax.set_xlim(left=0)
if len(pollutants) % 2 != 0:
    axes[-1, -1].axis('off')
plt.tight_layout()
plt.show()

for pollutant in pollutants:
  filtered_df = df[df[pollutant] != -200]
  mean_value = np.mean(filtered_df[pollutant])
  variance_value = np.var(filtered_df[pollutant])
  print(f"Mean of {pollutant}: {mean_value:.2f}")
  print(f"Variance of {pollutant}: {variance_value:.2f}")
```




```
Mean of CO(GT): 2.15
Variance of CO(GT): 2.11
Mean of NMHC(GT): 218.81
Variance of NMHC(GT): 41758.12
Mean of C6H6(GT): 10.08
Variance of C6H6(GT): 55.49
Mean of NOx(GT): 246.88
Variance of NOx(GT): 45350.87
Mean of NO2(GT): 113.08
Variance of NO2(GT): 2338.31
```

Figures above show the results derived from the previous codes. With all these data visualized, we indicate that the monitored location was experiencing high levels of NOx and NO2 pollution. Elevated concentrations of nitrogen oxide have concerns resulting in serious respiratory diseases and increasing the risk of heart disease. Therefore, the utilization of histograms proves to be an effective method for detecting the air quality safety level. Regarding our primary objective mentioned in section 1.1, we aim to conduct a thorough and professional dataset analysis. Carbon

monoxide (CO) and Non-Methane Hydrocarbons (NMHC) had relatively low concentrations during the monitoring period. However, NMHC exhibited a distinct lognormal distribution with a mean value of 218.81 and a variance of 41758.12. Similarly, NOx also demonstrated a clear log-normal distribution pattern, characterized by a mean of 246.88 and a variance of 45350.87. The other two pollutants, Benzene (C6H6) and Nitrogen Dioxide (NO2), both appeared to follow a symmetric normal distribution pattern.

4.2 Box plot

Unlike the Histogram, which primarily shows the distribution, the box plot also excels in identifying the outliners. Besides, it is also effective in analyzing the dataset across different time frames, including hours, days and weeks. In this section, we mainly focus on the distributions hourly and daily. However, the codes below can also serve as a foundation for examining other time periods.

To assess the hourly and daily pollutant levels, we first turned to one-sample t-tests—a statistical tool that tests the mean of a single group against a known average. Each t-test was carried out under a null hypothesis which is no significant difference between the mean pollutant level at a given time such as a specific hour or day and the dataset's overall mean. The resulting p-values were then used to validate or reject the hypothesis. The significance level was set at $= 0.05$, which refers to the allowance for a 5% probability of rejecting a true null hypothesis (Type I error).

[3]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns

file_path = 'processed_air_quality_data.csv'
air_quality_data = pd.read_csv(file_path)

pollutants = ['CO(GT)', 'NMHC(GT)', 'C6H6(GT)', 'NOx(GT)', 'NO2(GT)']

# Replace -200 with NaN to identify missing values
air_quality_data.replace(-200, np.nan, inplace=True)
# Interpolate missing values (NaN) with the mean of the previous and next values
for pollutant in pollutants:
    air_quality_data[pollutant] = air_quality_data[pollutant].
 ↪interpolate(method='linear')
```

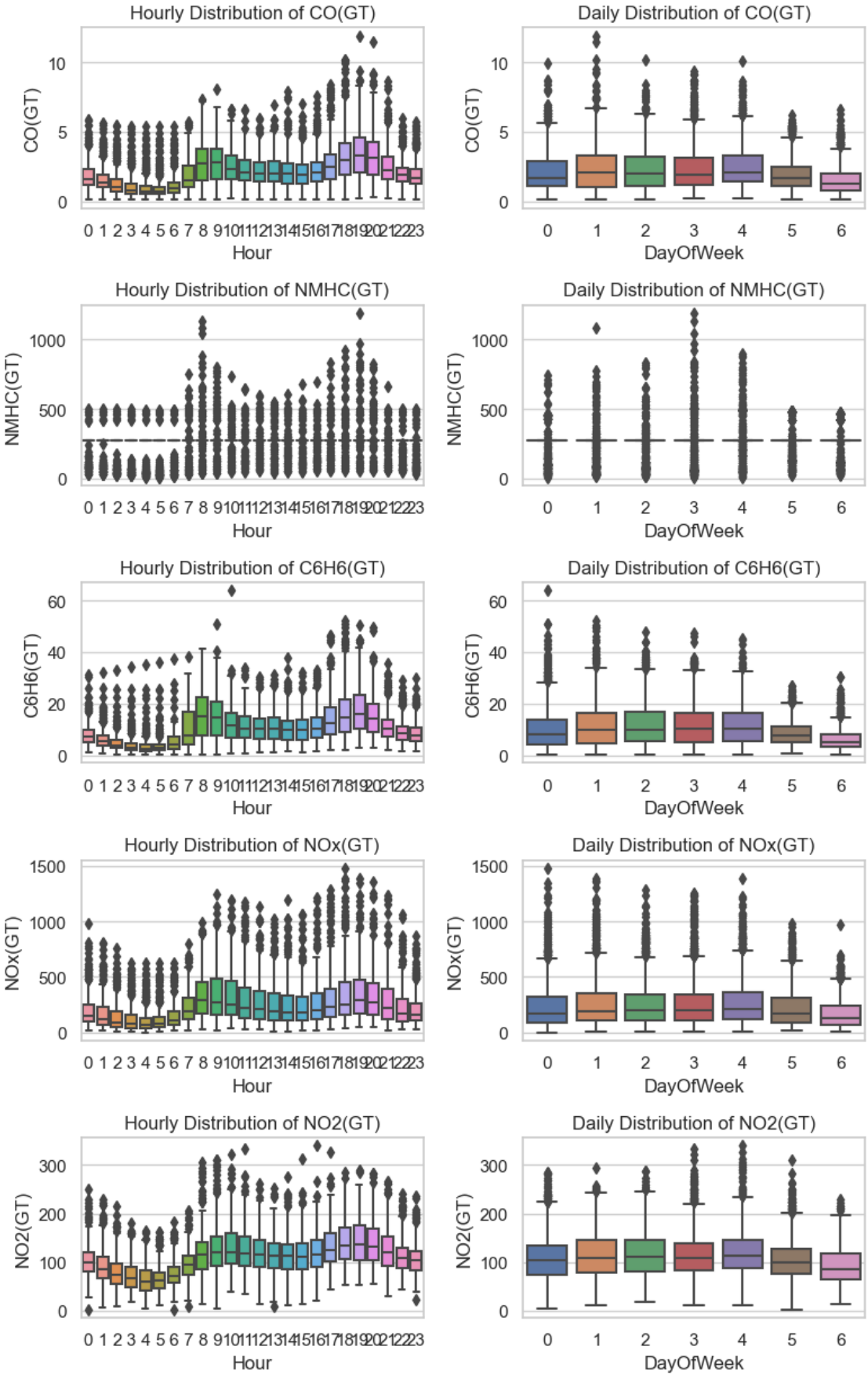
To start off, we interpolated missing values in the DataFrame to maintain continuity. This can be achieved with the 'interpolate' function above.

[4]:
```
p_values_hourly = {pollutant: {} for pollutant in pollutants}
p_values_daily = {pollutant: {} for pollutant in pollutants}

for pollutant in pollutants:
    hourly_means = air_quality_data.groupby('Hour')[pollutant].mean()
    for hour in sorted(air_quality_data['Hour'].unique()):
```

```
        sample = air_quality_data[air_quality_data['Hour'] == hour][pollutant].
 ↪dropna()
        overall_mean = hourly_means.mean()
        t_stat, p_val = stats.ttest_1samp(sample, overall_mean)
        p_values_hourly[pollutant][hour] = p_val

for pollutant in pollutants:
    daily_means = air_quality_data.groupby('DayOfWeek')[pollutant].mean()
    for day in sorted(air_quality_data['DayOfWeek'].unique()):
        sample = air_quality_data[air_quality_data['DayOfWeek'] ==␣
 ↪day][pollutant].dropna()
        overall_mean = daily_means.mean()
        t_stat, p_val = stats.ttest_1samp(sample, overall_mean)
        p_values_daily[pollutant][day] = p_val
```

These two loops calculates the mean pollutant levels for each hour and day and perform a one-sample t-test comparing each hour's and day's mean to the overall average. The resulting p-values are stored in p_values_hourly. Thus to test the hypothesis that each hourly mean is not significantly different from the overall daily mean.

[ ]:
```
sns.set(style="whitegrid")
fig, axes = plt.subplots(len(pollutants), 2, figsize=(8, len(pollutants) * 2.5))

for i, pollutant in enumerate(pollutants):
    # Hourly box plots
    sns.boxplot(x="Hour", y=pollutant, data=air_quality_data, ax=axes[i, 0])
    axes[i, 0].set_title(f'Hourly Distribution of {pollutant}')

    # Daily box plots
    sns.boxplot(x="DayOfWeek", y=pollutant, data=air_quality_data, ax=axes[i,␣
 ↪1])
    axes[i, 1].set_title(f'Daily Distribution of {pollutant}')

plt.tight_layout()
plt.show()
```


Hourly Distribution of CO(GT)
CO(GT)
Hour
Daily Distribution of CO(GT)
CO(GT)
DayOfWeek
Hourly Distribution of NMHC(GT)
NMHC(GT)
Hour
Daily Distribution of NMHC(GT)
NMHC(GT)
DayOfWeek
Hourly Distribution of C6H6(GT)
C6H6(GT)
Hour
Daily Distribution of C6H6(GT)
C6H6(GT)
DayOfWeek
Hourly Distribution of NOx(GT)
NOx(GT)
Hour
Daily Distribution of NOx(GT)
NOx(GT)
DayOfWeek
Hourly Distribution of NO2(GT)
NO2(GT)
Hour
Daily Distribution of NO2(GT)
NO2(GT)
DayOfWeek

The results, illustrated through intuitive box plots, effectively highlighted the key temporal patterns, which help to visualize central tendencies and dispersion in the data, including median, quartiles, and outliers. One-sample t-tests accompanied by box plot visualizations, provide a robust statistical framework for comparing individual time points against a population mean, ideal for identifying specific periods with significant deviation in pollutant levels.

With the p-values of the hourly and daily t-test output at the bottom of the box, we are able to indicate whether there's a statistically significant difference between the pollutant levels at a given time and the overall average. A low p-value (typically less than 0.05) suggests that the difference is significant. The t-tests indicated significant variations in pollutant levels at different times, as suggested by the low p-values.

Generally, we observe higher median pollution levels during peak hours, which are likely correlating with increased traffic and industrial activities, and a pronounced elevation in pollutant levels on weekdays compared to weekends. These observations underscore the advantages of using boxplots over histograms, as they provide more detailed and comprehensive views, without sacrificing the distribution clarity.

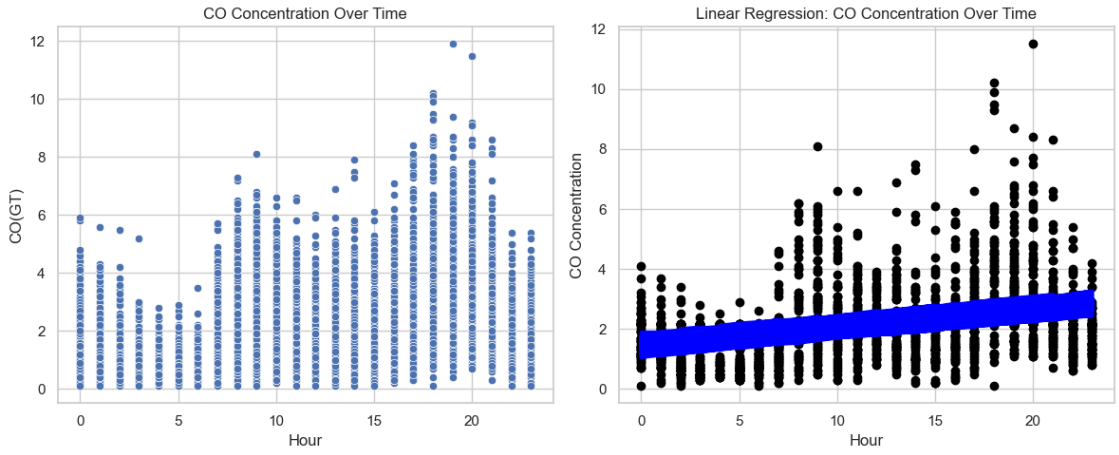
4.3 Linear regression

Linear regression is a statistical method used to model the relationship between a dependent variable and independent variables, which aims to find the best-fit straight line through the data. This procedure allows us to identify whether the linear relationship is an appropriate prediction for future evaluation, which aligns with our second stated objective.

The following code serves as an example of this method with only plotting the data in relation to the hours. However, depending on the specific requirements of our analysis, this approach can be adapted to include varying time frames, such as days and months. This code initially identifies the Time columns and the pollutants columns in order to graphically represent the concentration over "hours, incorporating a best-fit linear line. To evaluate the goodness of fit, the corresponding R-values are computed to indicate the appropriateness of the linear assumption.

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('processed_air_quality_data.csv')
df['DateTime'] = pd.to_datetime(df['DateTime'])
df['Month'] = df['DateTime'].dt.month
df['Day'] = df['DateTime'].dt.dayofweek
df['Hour'] = df['DateTime'].dt.hour

df['CO(GT)'] = df['CO(GT)'].apply(lambda x: (df['CO(GT)'][df['CO(GT)'] >= 0]).
 ↪mean() if x < 0 else x)
df = df.dropna()
```

```
X = df[['Month', 'Day', 'Hour']]
y = df['CO(GT)']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.scatterplot(x='Hour', y='CO(GT)', data=df, ax=axes[0])
axes[0].set_title('CO Concentration Over Time')

axes[1].scatter(X_test['Hour'], y_test, color='black')
axes[1].plot(X_test['Hour'], model.predict(X_test), color='blue', linewidth=3)
axes[1].set_title('Linear Regression: CO Concentration Over Time')
axes[1].set_xlabel('Hour')
axes[1].set_ylabel('CO Concentration')

plt.tight_layout()
plt.show()

print(f'R-squared value: {model.score(X_test, y_test)}')
```




```
R-squared value: 0.15006210077294735
```

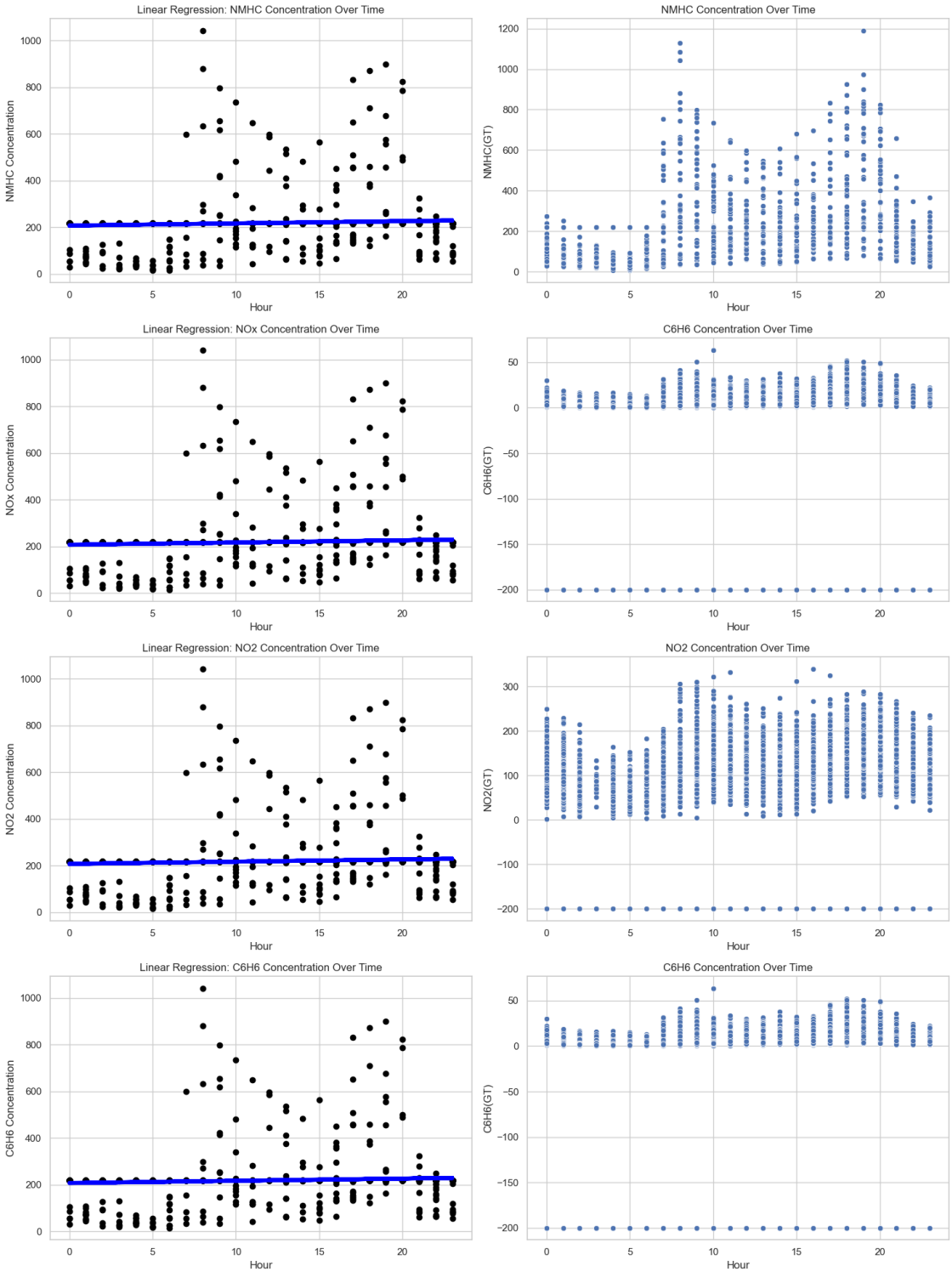
With the example codes, we can also plot the remaining four pollutants for their linear regression below. The output figures, 5.1 to 5.4, present a series of black dots, depicting the concentration data over hours, and prominent blue linear lines indicating the best-fit straight line. Upon the analysis, it is observed that a majority of the data is situated away from the linear line, leading to the conclusion that linear regression is not an appropriate method to analyze our data or for

future predictive modelling. This conclusion is further substantiated by the corresponding R-values, detailed in output box below.

[ ]:
```
X = df[['Month', 'Day', 'Hour']]
y = df['NMHC(GT)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

print(f'R-squared value: {model.score(X_test, y_test)}')

fig, axes = plt.subplots(4, 2, figsize=(15, 20))

model.fit(X_train, y_train)
axes[0, 0].scatter(X_test['Hour'], y_test, color='black')
axes[0, 0].plot(X_test['Hour'], model.predict(X_test), color='blue',␣
 ↪linewidth=3)
axes[0, 0].set_title('Linear Regression: NMHC Concentration Over Time')
axes[0, 0].set_xlabel('Hour')
axes[0, 0].set_ylabel('NMHC Concentration')

sns.scatterplot(x='Hour', y='NMHC(GT)', data=df, ax=axes[0, 1])
axes[0, 1].set_title('NMHC Concentration Over Time')

model.fit(X_train, y_train)
axes[1, 0].scatter(X_test['Hour'], y_test, color='black')
axes[1, 0].plot(X_test['Hour'], model.predict(X_test), color='blue',␣
 ↪linewidth=3)
axes[1, 0].set_title('Linear Regression: NOx Concentration Over Time')
axes[1, 0].set_xlabel('Hour')
axes[1, 0].set_ylabel('NOx Concentration')

sns.scatterplot(x='Hour', y='C6H6(GT)', data=df, ax=axes[1, 1])
axes[1, 1].set_title('C6H6 Concentration Over Time')

model.fit(X_train, y_train)
axes[2, 0].scatter(X_test['Hour'], y_test, color='black')
axes[2, 0].plot(X_test['Hour'], model.predict(X_test), color='blue',␣
 ↪linewidth=3)
axes[2, 0].set_title('Linear Regression: NO2 Concentration Over Time')
axes[2, 0].set_xlabel('Hour')
axes[2, 0].set_ylabel('NO2 Concentration')

sns.scatterplot(x='Hour', y='NO2(GT)', data=df, ax=axes[2, 1])
```

```
axes[2, 1].set_title('NO2 Concentration Over Time')

model.fit(X_train, y_train)
axes[3, 0].scatter(X_test['Hour'], y_test, color='black')
axes[3, 0].plot(X_test['Hour'], model.predict(X_test), color='blue',␣
 ↪linewidth=3)
axes[3, 0].set_title('Linear Regression: C6H6 Concentration Over Time')
axes[3, 0].set_xlabel('Hour')
axes[3, 0].set_ylabel('C6H6 Concentration')

sns.scatterplot(x='Hour', y='C6H6(GT)', data=df, ax=axes[3, 1])
axes[3, 1].set_title('C6H6 Concentration Over Time')

plt.tight_layout()
plt.show()
```

```
R-squared value: 0.008846779845325337
```


Linear Regression: NMHC Concentration Over Time
NMHC Concentration
Hour
NMHC Concentration Over Time
NMHC(GT)
Hour
Linear Regression: NOx Concentration Over Time
NOx Concentration
Hour
C6H6 Concentration Over Time
C6H6(GT)
Hour
Linear Regression: NO2 Concentration Over Time
NO2 Concentration
Hour
NO2 Concentration Over Time
NO2(GT)
Hour
Linear Regression: C6H6 Concentration Over Time
C6H6 Concentration
Hour
C6H6 Concentration Over Time
C6H6(GT)
Hour

5 Advanced Analysis

Compared to EDA, Advanced analysis delves deeper, not only achieving a basic understanding of the data but aiming to construct predictive models, undergoing complex relationships. In order to have a thorough overview of the dataset, it is essential to integrate both analysis methods, each serving distinct objectives. Advanced Analysis allows for predicting future pollution levels and plays a crucial role in identifying effective measurements or controls in future environmental control. This dual approach ensures a balanced exploration of data, enhancing the fundamental EDA with absolute predictive power.

5.1 Principal Component Analysis (PCA)

PCA is a sophisticated technique that greatly simplifies the complexity of high-dimensional data. It is instrumental in revealing trends and enhancing data interpretation. PCA works by creating new dimensions through linear combinations of existing ones, focusing on those that preserve the maximum amount of information.

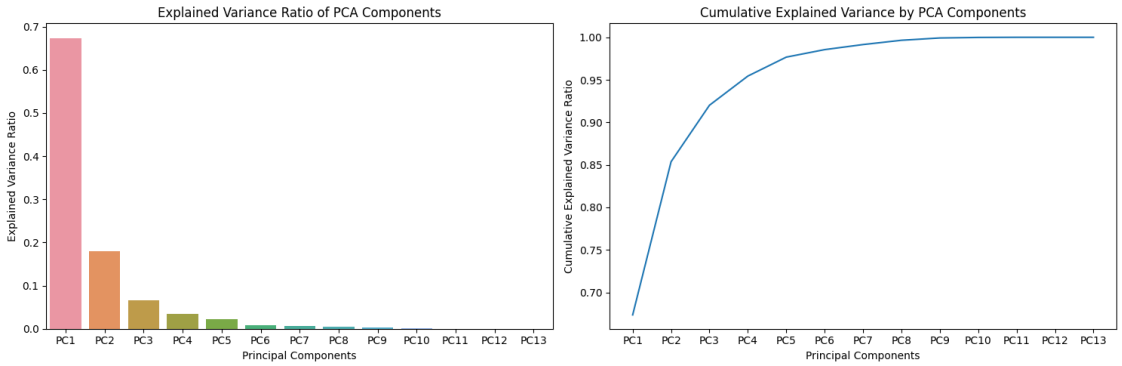
In our analysis, all non-date and non-categorical columns were treated as numerical features, with gas levels (GT) as the target variables. These features were standardized using the StandardScaler, a preprocessing tool that adjusts them to have unit variance. Our PCA results reveal that the first few principal components accounted for a significant portion of the variance in the data, with the first component alone explaining approximately 67%.

Additionally, Single Value Decomposition (SVD) was performed on the scaled feature matrix from the dataset. This technique decomposes the matrix into three matrices: U (left singular vectors), S (singular values), and Vh (right singular vectors). We calculated the reconstruction error to approximate the scaled dataset using different singular vectors from the SVD. The 'all_errors' matrix offers insights into the effectiveness of dimensionality reduction; lower values indicate better approximations of the original data.

Lastly, we visualized the dataset to showcase the relationship between two specific variables: x1 (an array of 50 evenly spaced numbers between 10 and 100) and x2 (representing a second dimension of the data). This visualization further elucidates the underlying patterns and relationships in the dataset.

[7]:
```
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
file_path = 'processed_air_quality_data.csv'
air = pd.read_csv(file_path)
numerical_features = air.select_dtypes(include=['float64', 'int64']).columns.
  ↪tolist()
numerical_features.remove('Hour')
numerical_features.remove('DayOfWeek')
numerical_features.remove('Month')

target_values = ['CO(GT)', 'NMHC(GT)', 'NOx(GT)', 'NO2(GT)']
```

```
X = air[numerical_features]
y = air[target_values]

pca = PCA(n_components=len(numerical_features))
X_pca = pca.fit_transform(X)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.barplot(x=[f'PC{i+1}' for i in range(len(numerical_features))], y=pca.
 ↪explained_variance_ratio_, ax=axes[0])
axes[0].set_title('Explained Variance Ratio of PCA Components')
axes[0].set_ylabel('Explained Variance Ratio')
axes[0].set_xlabel('Principal Components')

# Calculating Cumulative Explained Variance
cumulative_variance = pca.explained_variance_ratio_.cumsum()

sns.lineplot(x=[f'PC{i+1}' for i in range(len(numerical_features))],␣
 ↪y=cumulative_variance, ax=axes[1])
axes[1].set_title('Cumulative Explained Variance by PCA Components')
axes[1].set_ylabel('Cumulative Explained Variance Ratio')
axes[1].set_xlabel('Principal Components')

plt.tight_layout()
plt.show()
```



5.2 Hyperparameter Tuning

In this phase, our goal was to fine-tune the alpha ( ) parameter of a ridge regression model, employing the Root Mean Squared Error (RMSE) as the key metric for evaluating performance.

We initiated the process by training and evaluating a multiple ridge regression model. In line with our prior approach, we began by identifying and selecting the relevant features. Subsequently, we divided the data into a training set and a validation set, adhering to a 4:1 ratio. This split ensures

that the model is trained on a comprehensive subset of data while retaining a separate portion for validation, crucial for assessing the model's performance.

Once the data was partitioned, we initialized multiple models corresponding to different alpha values. Each model underwent training on the training set. Following this, we evaluated them on the validation set, focusing on their RMSE values. This process is essential as it quantifies the average magnitude of the errors between the predicted and actual values, providing a clear measure of model performance.

The results of these evaluations were then visualized to facilitate the selection of the optimal alpha value. This visualization made it evident that the model achieved its best performance at an alpha of 0.001. Selecting this alpha value is a critical step in ridge regression, as it balances the trade-off between fitting the model's complexity and controlling for overfitting, thereby enhancing the model's prediction accuracy on new data.

5.3 Cross Validation

Selecting the appropriate tuning parameter is a crucial step in model development, as it effectively controls the complexity of the model. This tuning parameter helps in balancing the bias-variance trade-off which is a key aspect in preventing overfitting or underfitting. To capture more complex relationships in the data, we employed a custom function named hstack_higher_order. This function accepts an array and an integer specifying the highest polynomial degree. It is designed to transform the input features into a higher-dimensional space by fitting a polynomial model. This transformation is particularly useful for capturing higher-order interactions between variables, which might be missed in a simple linear regression approach.

We conducted an experiment focusing on two variables: "AH" and "C6H6(GT)". The goal was to determine the polynomial order that best models the relationship between these variables while minimizing error, and avoiding overfitting or underfitting.

In this experiment, different polynomial degrees were applied to these variables, and for each degree, a linear regression model was implemented. To ensure the robustness of our assessment, each trial involved using a different split of the data. This approach allows us to evaluate the model's performance across various subsets of the data, thereby providing a more comprehensive understanding of its generalized evaluation To compare the performance of models with varying complexities, we created plots that displayed how each model performed on both the training and validation sets. These plots were instrumental in visualizing the trade-off between model complexity and performance. They helped in identifying the optimal polynomial degree that captures the underlying pattern of the data effectively without falling prey to overfitting or underfitting.

[8]:
```
from sklearn.decomposition import PCA
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
import numpy as np
```

```
numerical_features = ['PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)',␣
 ↪'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T',␣
 ↪'RH', 'AH']

target_values = ["CO(GT)","NMHC(GT)",
    "NOx(GT)",
    "NO2(GT)",
    "C6H6(GT)"]

X_train, X_validation, y_train, y_validation = train_test_split(
    air[numerical_features],
    air[target_values],
    test_size=0.2,
    random_state=42,
)

all_models = []
rmse = pd.DataFrame(columns=["alpha", "train", "validation"])

for alpha in np.logspace(start=-3, stop=.2, num=50):
    model = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=alpha *␣
 ↪len(X_train)))
    model.fit(X=X_train, y=y_train)
    all_models.append(model)

    y_pred_train = model.predict(X_train)
    y_pred_validation = model.predict(X_validation)
    row = pd.Series({
        "alpha": alpha,
        "train": np.sqrt(mean_squared_error(y_true=y_train,␣
 ↪y_pred=y_pred_train)),
        "validation": np.sqrt(mean_squared_error(y_true=y_validation,␣
 ↪y_pred=y_pred_validation)),
    })
    rmse = pd.concat((rmse, row.to_frame().T), ignore_index=True)
rmse.tail()
```

[8]:

	alpha	train	validation
45	0.868511	61.281735	61.122227
46	1.009443	65.094936	64.995842
47	1.173242	69.095316	69.046132
48	1.363622	73.252499	73.241672
49	1.584893	77.531419	77.546818

Taking into account the insights gained from Advanced analysis, PCA has reinforced our initial observations from the EDA phase. It futher uncoveredthat a substantial portion of the dataset's

variance could be effectively captured by the first few principal components. This finding underscores the efficacy of dimensionality reduction for this particular dataset, confirming that PCA is a suitable technique for simplifying the data without significant loss of information. The process of hyperparameter tuning provided valuable insights into the model's performance. It was determined that an alpha value of 0.001 was optimal for our ridge regression model. This specific alpha value led to the lowest Root Mean Squared Error (RMSE), indicating a strong balance between the model's complexity and its ability to generalize.

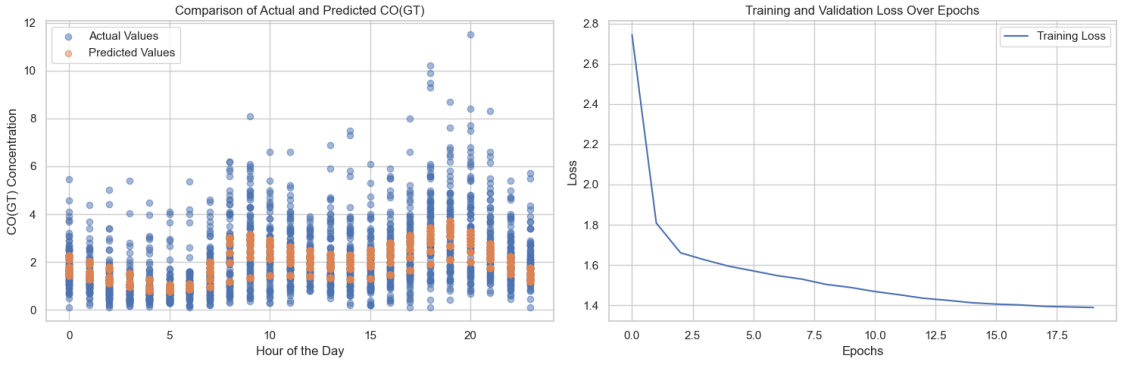
6 Neural Network

Neural Network is one of the most frequently used machine learning models that is designed to recognize patterns and allow for future prediction for a large and complex dataset. The scatter plots below demonstrate the inadequacy of a simple linear relationship between pollutants's concentrations and time, which aligns with the finding in the linear regression session 3.2. Given these complexities, the neural network approach is particularly suitable for this dataset, in accordance with the objective listed in section 1.2.

Presented below are codes that aim to train the CO's temporal distribution, which also serves as a representative case for the methodology applied.

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

data = pd.read_csv('processed_air_quality_data1.csv')
input_features = ['Hour', 'DayOfWeek']
targets = ['CO(GT)']

X = data[input_features].values
y = data[targets].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = Sequential([
    Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(y_train.shape[1], activation='linear')
])
```

```
model.compile(optimizer='adam', loss='mean_squared_error')
history = model.fit(X_train_scaled, y_train, epochs=20, batch_size=32,␣
 ↪verbose=0)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].scatter(X_test[:, 0], y_test, label='Actual Values', alpha=0.5)  # Plot␣
 ↪actual values
axes[0].scatter(X_test[:, 0], y_pred, label='Predicted Values', alpha=0.5)  #␣
 ↪Plot predicted values
axes[0].set_title('Comparison of Actual and Predicted CO(GT)')
axes[0].set_xlabel('Hour of the Day')
axes[0].set_ylabel('CO(GT) Concentration')
axes[0].legend()


axes[1].plot(history.history['loss'], label='Training Loss')
if 'val_loss' in history.history:
    axes[1].plot(history.history['val_loss'], label='Validation Loss')
axes[1].set_title('Training and Validation Loss Over Epochs')
axes[1].set_xlabel('Epochs')
axes[1].set_ylabel('Loss')
axes[1].legend()

plt.tight_layout()
plt.show()
```



The training outputs of the neural network indicate notably lower loss values, which signifies strengthened model accuracy. The consistency between the predicted and actual values of the pollutant level distribution at different times of the day proves the effectiveness of the neural network. Align with the boxplots in the above session, the data show higher levels of air pollutants during the daytime, peaking in the morning and evening. This somehow emphasizes the critical period for pollution control measures.

The prediction model we developed using the neural network approach can estimate pollutant levels based on input parameters such as time of day. However, the discrepancies between predicted and actual values could be caused by the absence of certain variables. This observation motivates us to consider incorporating a wider range of variables in future iterations of the model. By expanding the set of variables, we aim to improve the accuracy of the model and make it more precise in predicting air pollution levels. We aspire not just to refine the reliability of the model, but also to provide a valuable reference for the development of more accurate and effective pollution management strategies.

7 Summary and future works

This study reveals a significant correlation between pollutant levels and time-specific factors, especially on weekdays and rush hours. These insights have important implications for decision-making in areas such as transportation planning and pollution control. The majority of our methods implemented in this study effectively fulfill objective 1, which is to monitor the pollutant concentration in real time and determine the pollution level. However, when evaluating their long-term applicability, certain methods do not perform as satisfactorily as anticipated.

Implementing a variety of methods, we identified a set of optimal methodologies for our data. First of all, under the Explotary Data Analysis, boxplots emerged as a preferred choice. They not only encapsulate the overall distribution insights from the Histogram but also offer adaptability to different time periods. In contrast, the Linear regression methods demonstrate poor fit under the assumption of a simple linear relationship. Therefore, advanced analysis methods are required to address the shortcomings and enhance the model prediction. PCA, combined with hyperparameter tuning, proves to be a crucial tool, distilling the essence of the data into fewer, and more informative components makes it an indispensable tool for a complex database. Cross-validation plays a key role in mitigating overfitting risks and provides a fair ground for comparing the performances along with other methods, enhancing the interpretability and manageability of the dataset. Lastly, the neural network stands out as an exceptionally effective predictive model for any future data inputs, showcasing the remarkable ability to navigate non-linear complex relationships.

In general, it is unrealistic to anticipate that one single method can encompass the entirety of defining, processing, evaluating and accurately predicting within a dataset. For a comprehensive data analysis, it is essential that various methods complement and support each other. Although at the moment what we have performed is a simple univariate study, control of pollutant levels is determined by many other complex factors, and their influence on the error and fitting effect cannot be ignored, such as seasonal variations, policy implementation, population size, changes in transportation use, and so on. Therefore, we plan to incorporate a wider range of variables to improve the accuracy of the predictions. Another direction of improvement lies in optimizing the architecture of the neural network, especially the layer settings and the learning rate to produce more refined results. Finally, it is clear that we need to expand our dataset to ensure that our model has broad applicability and robustness across different scenarios. By addressing these issues, we aim to significantly advance our understanding and prediction of air pollution patterns, ultimately contributing to the development of more effective environmental management strategies.