Air Quality Analysis and Prediction Using Random Forest

Objective

This project explores air pollution data collected from a sensor network in an Italian city to:

- Understand seasonal variations in pollutant levels (CO, NOx, Benzene, etc.)
- Build a regression model (Random Forest) to predict pollutant concentrations
- Evaluate how adding "season" as a feature impacts model performance

Dataset Overview

- Source: kaggle (Air Quality Dataset)
- Period: March 2004 – February 2005 (Hourly Data)
- Sensors: 5 metal oxide chemical sensors
- Pollutants: CO, NMHC, Benzene, NOx, NO2
- Features: Temperature, Humidity, Absolute Humidity

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

In [4]:
```
df = pd.read_csv("AirQuality.csv.xls",sep=';')
df.head()
```

Out[4]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08
0	10/03/2004	18.00.00	2,6	1360.0	150.0	11,9	
1	10/03/2004	19.00.00	2	1292.0	112.0	9,4	
2	10/03/2004	20.00.00	2,2	1402.0	88.0	9,0	
3	10/03/2004	21.00.00	2,2	1376.0	80.0	9,2	
4	10/03/2004	22.00.00	1,6	1272.0	51.0	6,5	

In [5]:
```
# to get the breif idea about cleaning and preparation
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           9357 non-null   object 
 1   Time           9357 non-null   object 
 2   CO(GT)         9357 non-null   object 
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 non-null   float64
 5   C6H6(GT)       9357 non-null   object 
 6   PT08.S2(NMHC)  9357 non-null   float64
 7   NOx(GT)        9357 non-null   float64
 8   PT08.S3(NOx)   9357 non-null   float64
 9   NO2(GT)        9357 non-null   float64
 10  PT08.S4(NO2)   9357 non-null   float64
 11  PT08.S5(O3)    9357 non-null   float64
 12  T              9357 non-null   object 
 13  RH             9357 non-null   object 
 14  AH             9357 non-null   object 
 15  Unnamed: 15    0 non-null      float64
 16  Unnamed: 16    0 non-null      float64
dtypes: float64(10), object(7)
memory usage: 1.2+ MB
```

Data Cleaning & Preprocessing

Steps Taken:

1. **Drop empty columns/rows** – Removed columns and rows with all values missing.
2. **Convert 'object' to numeric** – Converted all numerical columns from `object` type to `float`.
3. **Format 'Date' and 'Time'** – Combined into a single datetime column and set as index.
4. **Drop highly-missing columns** – Dropped `NMHC(GT)` which had >90% missing data.
5. **Visualize missing data** – Used a heatmap to identify patterns.
6. **Interpolate missing values** – Applied interpolation to `CO(GT)` which had ~19% missing.
7. **Outlier Detection** – Used boxplots to visualize sensor data and remove extreme outliers.
8. **Create cleaned dataframe** – Filtered outliers based on IQR and formed a final dataset.

In [6]:
```
# Drop completely empty columns
df = df.drop(['Unnamed: 15', 'Unnamed: 16'], axis=1)
```

In [7]:
```
print(df.columns.tolist())
```

```
['Date', 'Time', 'CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'P
T08.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)',
'PT08.S5(O3)', 'T', 'RH', 'AH']
```

In [8]:
```
# converting 'object' to 'float'

df.replace(to_replace=',',value='.',regex=True,inplace=True)

for i in ['CO(GT)','C6H6(GT)','T','RH' ,'AH']:
    df[i] = pd.to_numeric(df[i],errors='coerce')
```

In [9]:
```
df.head()
```

Out[9]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08
0	10/03/2004	18.00.00	2.6	1360.0	150.0	11.9	
1	10/03/2004	19.00.00	2.0	1292.0	112.0	9.4	
2	10/03/2004	20.00.00	2.2	1402.0	88.0	9.0	
3	10/03/2004	21.00.00	2.2	1376.0	80.0	9.2	
4	10/03/2004	22.00.00	1.6	1272.0	51.0	6.5	

In [10]:
```
#formating date and time

df['Date'] = pd.to_datetime(df['Date'],dayfirst=True)
df['Time'] = pd.to_datetime(df['Time'],format= '%H.%M.%S' ).dt.time

df.head()
```

Out[10]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(
0	2004-03-10	18:00:00	2.6	1360.0	150.0	11.9	
1	2004-03-10	19:00:00	2.0	1292.0	112.0	9.4	
2	2004-03-10	20:00:00	2.2	1402.0	88.0	9.0	
3	2004-03-10	21:00:00	2.2	1376.0	80.0	9.2	
4	2004-03-10	22:00:00	1.6	1272.0	51.0	6.5	

In [11]:
```
df[df['CO(GT)'] < 0]
```

Out[11]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08
10	2004-03-11	04:00:00	-200.0	1011.0	14.0	1.3	
34	2004-03-12	04:00:00	-200.0	831.0	10.0	1.1	
39	2004-03-12	09:00:00	-200.0	1545.0	-200.0	22.1	
58	2004-03-13	04:00:00	-200.0	1147.0	56.0	6.2	
82	2004-03-14	04:00:00	-200.0	1130.0	56.0	5.2	
...	...	...	...	...	...	...	
9058	2005-03-23	04:00:00	-200.0	993.0	-200.0	2.3	
9130	2005-03-26	04:00:00	-200.0	1122.0	-200.0	6.0	
9202	2005-03-29	04:00:00	-200.0	883.0	-200.0	1.3	
9274	2005-04-01	04:00:00	-200.0	818.0	-200.0	0.8	
9346	2005-04-04	04:00:00	-200.0	864.0	-200.0	0.8	

1683 rows × 15 columns

In [12]:
```
# Replace negative values with NaN

df['CO(GT)'] = df['CO(GT)'].apply(lambda x: x if x >= 0 else np.nan
df['NMHC(GT)'] = df['NMHC(GT)'].apply(lambda x: x if x >= 0 else np
```

In [13]:
```
df[df['CO(GT)'].isna()]
```

Out[13]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08
10	2004-03-11	04:00:00	NaN	1011.0	14.0	1.3	
34	2004-03-12	04:00:00	NaN	831.0	10.0	1.1	
39	2004-03-12	09:00:00	NaN	1545.0	NaN	22.1	
58	2004-03-13	04:00:00	NaN	1147.0	56.0	6.2	
82	2004-03-14	04:00:00	NaN	1130.0	56.0	5.2	
...	...	...	...	...	...	...	
9466	NaT	NaT	NaN	NaN	NaN	NaN	
9467	NaT	NaT	NaN	NaN	NaN	NaN	
9468	NaT	NaT	NaN	NaN	NaN	NaN	
9469	NaT	NaT	NaN	NaN	NaN	NaN	
9470	NaT	NaT	NaN	NaN	NaN	NaN	

1797 rows × 15 columns

In [14]:
```
df.info()
```

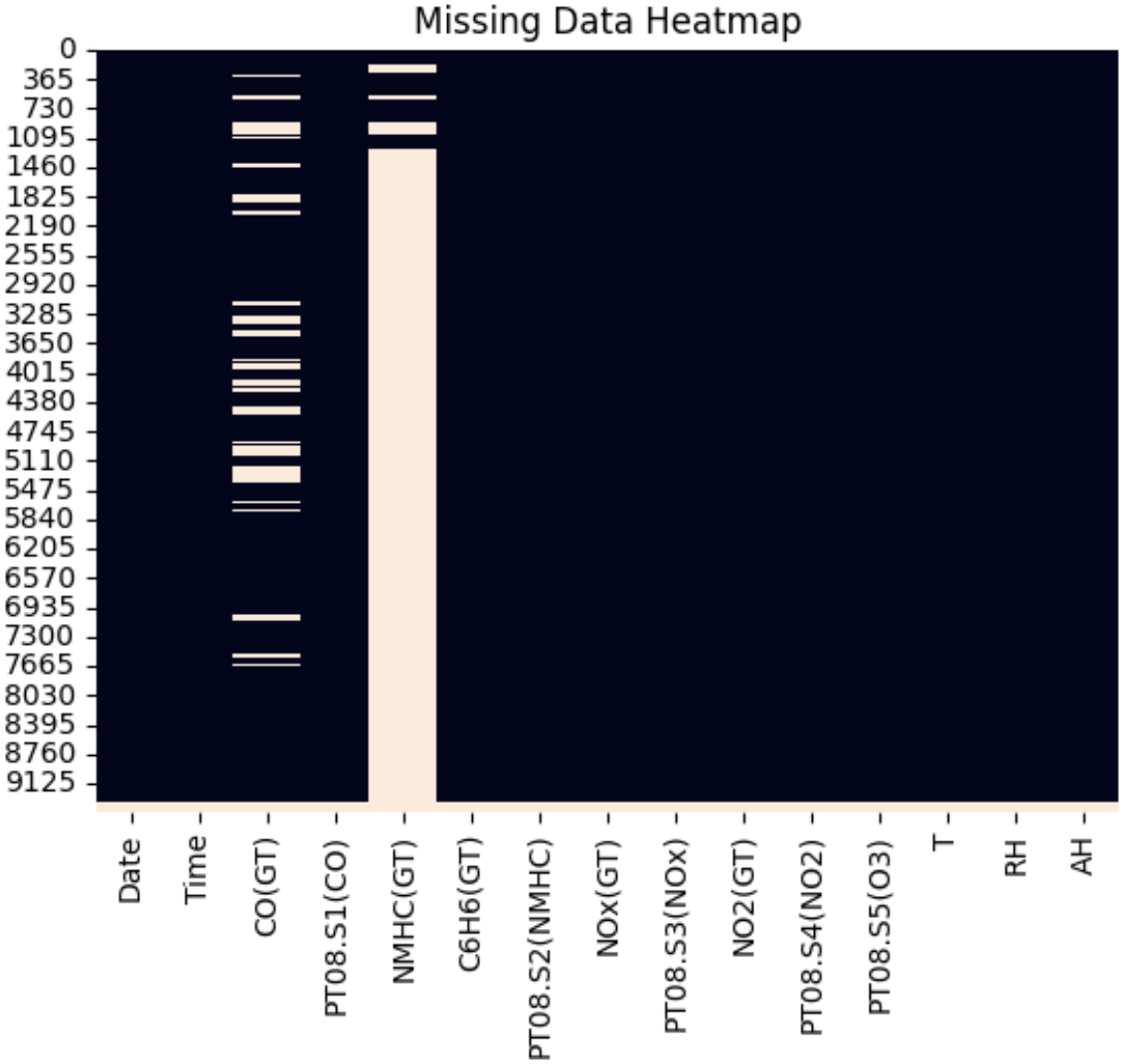
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   Date           9357 non-null   datetime64[ns]
 1   Time           9357 non-null   object        
 2   CO(GT)         7674 non-null   float64       
 3   PT08.S1(CO)    9357 non-null   float64       
 4   NMHC(GT)       914 non-null    float64       
 5   C6H6(GT)       9357 non-null   float64       
 6   PT08.S2(NMHC)  9357 non-null   float64       
 7   NOx(GT)        9357 non-null   float64       
 8   PT08.S3(NOx)   9357 non-null   float64       
 9   NO2(GT)        9357 non-null   float64       
 10  PT08.S4(NO2)   9357 non-null   float64       
 11  PT08.S5(O3)    9357 non-null   float64       
 12  T              9357 non-null   float64       
 13  RH             9357 non-null   float64       
 14  AH             9357 non-null   float64       
dtypes: datetime64[ns](1), float64(13), object(1)
memory usage: 1.1+ MB
```

In [15]:
```
# calculation the number of missing value
CO_ratio = df['CO(GT)'].isna().sum() / len(df['CO(GT)'])
```

```
NMHC_ratio = df['NMHC(GT)'].isna().sum() / len(df['NMHC(GT)'])

print('The CO(GT) sensor has {:.2f}% of missing data.'.format(CO_ra
print('The NMHC(GT) sensor has {:.2f}% of missing data.'.format(NMH
```

```
The CO(GT) sensor has 18.97% of missing data.
The NMHC(GT) sensor has 90.35% of missing data.
```

In [16]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(df.isnull(), cbar=False)
plt.title("Missing Data Heatmap")
plt.show()
```



In [17]:
```
df.drop(columns=['NMHC(GT)'], inplace=True)
```

In [18]:
```
df['CO(GT)'] = df['CO(GT)'].interpolate(method='linear')
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           9357 non-null   datetime64[ns]
 1   Time           9357 non-null   object
 2   CO(GT)         9471 non-null   float64
 3   PT08.S1(CO)    9357 non-null   float64
 4   C6H6(GT)       9357 non-null   float64
 5   PT08.S2(NMHC)  9357 non-null   float64
 6   NOx(GT)        9357 non-null   float64
 7   PT08.S3(NOx)   9357 non-null   float64
 8   NO2(GT)        9357 non-null   float64
 9   PT08.S4(NO2)   9357 non-null   float64
 10  PT08.S5(O3)    9357 non-null   float64
 11  T              9357 non-null   float64
 12  RH             9357 non-null   float64
 13  AH             9357 non-null   float64
dtypes: datetime64[ns](1), float64(12), object(1)
memory usage: 1.0+ MB
```

In [20]:
```
#Find the rows with missing values

df[df.isnull().any(axis=1)].head()
```

Out[20]:

	Date	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(NMHC)	NOx
9357	NaT	NaT	2.2	NaN	NaN	NaN	
9358	NaT	NaT	2.2	NaN	NaN	NaN	
9359	NaT	NaT	2.2	NaN	NaN	NaN	
9360	NaT	NaT	2.2	NaN	NaN	NaN	
9361	NaT	NaT	2.2	NaN	NaN	NaN	

In [21]:
```
df = df[df.isnull().sum(axis=1) <= 4]
df.info()
```

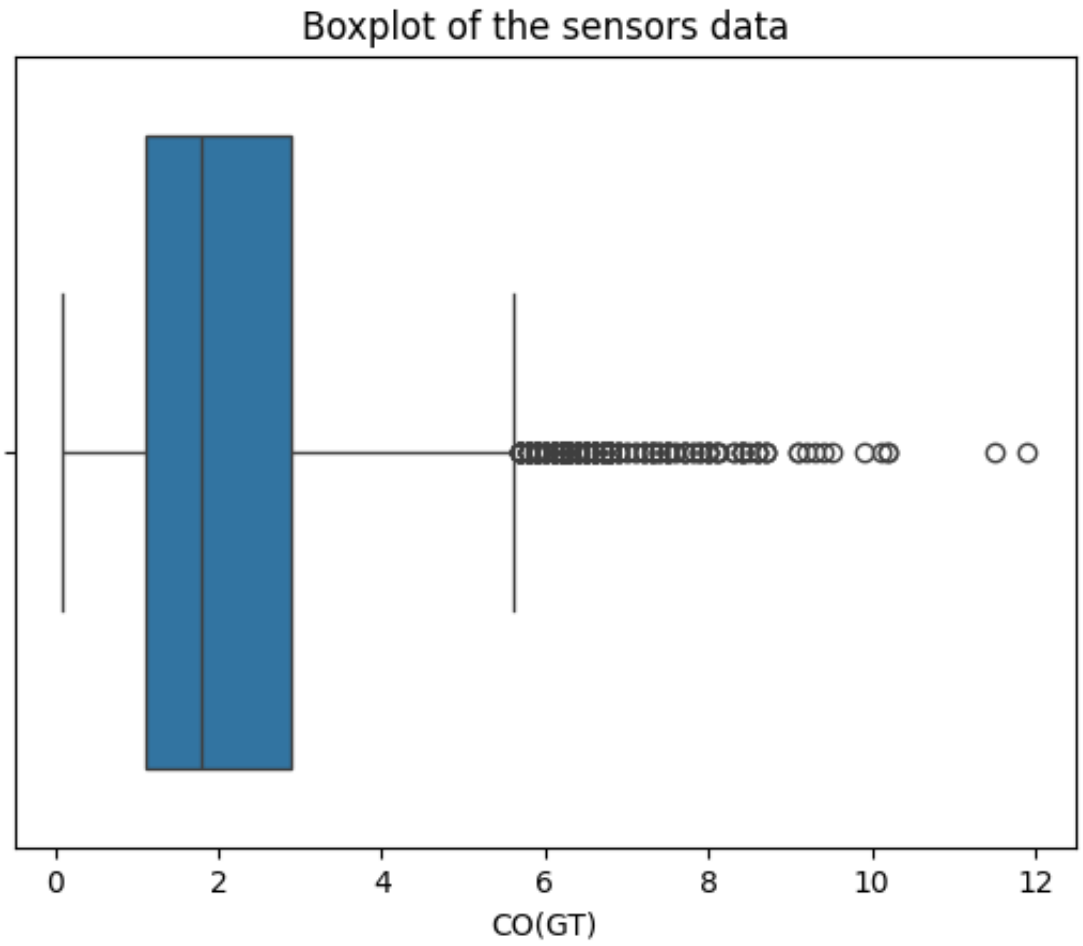
```
<class 'pandas.core.frame.DataFrame'>
Index: 9357 entries, 0 to 9356
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   Date           9357 non-null   datetime64[ns]
 1   Time           9357 non-null   object        
 2   CO(GT)         9357 non-null   float64       
 3   PT08.S1(CO)    9357 non-null   float64       
 4   C6H6(GT)       9357 non-null   float64       
 5   PT08.S2(NMHC)  9357 non-null   float64       
 6   NOx(GT)        9357 non-null   float64       
 7   PT08.S3(NOx)   9357 non-null   float64       
 8   NO2(GT)        9357 non-null   float64       
 9   PT08.S4(NO2)   9357 non-null   float64       
 10  PT08.S5(O3)    9357 non-null   float64       
 11  T              9357 non-null   float64       
 12  RH             9357 non-null   float64       
 13  AH             9357 non-null   float64       
dtypes: datetime64[ns](1), float64(12), object(1)
memory usage: 1.1+ MB
```

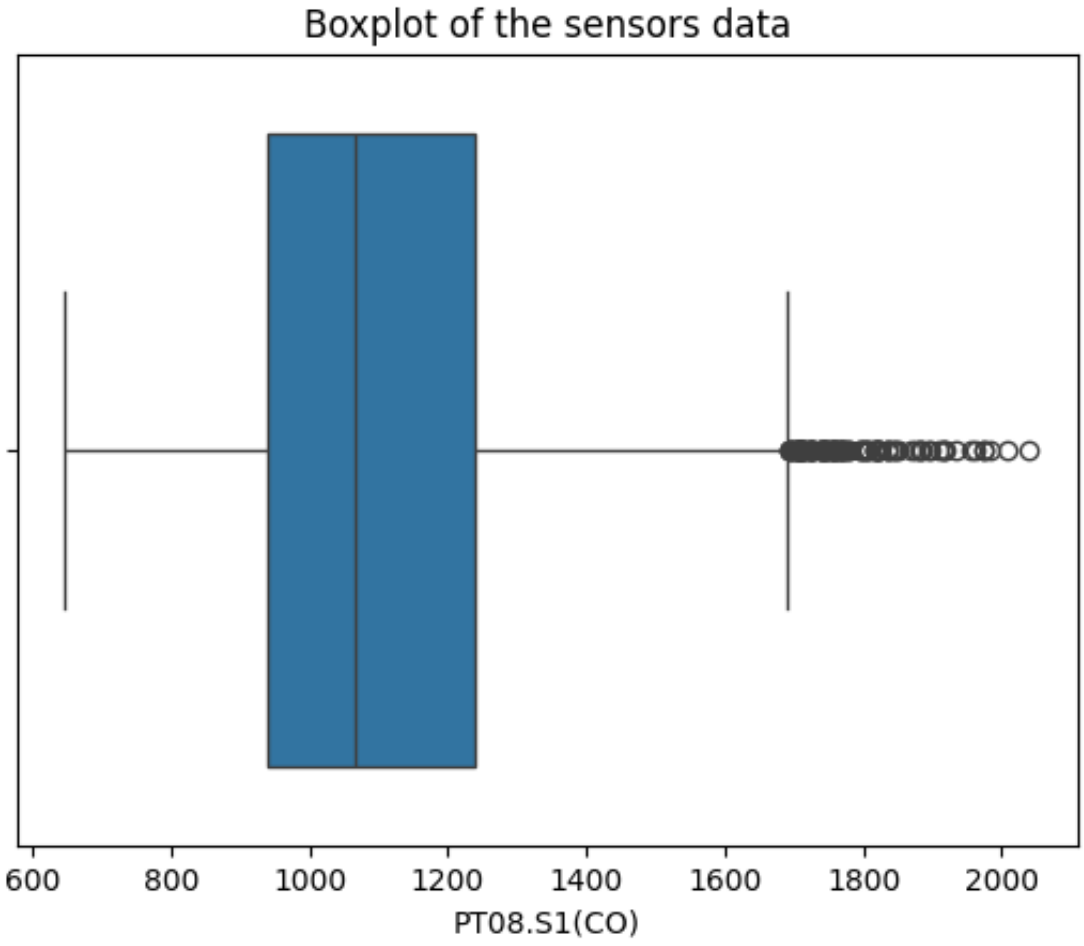
In [22]:
```
#checking duplication

duplicates = df.duplicated().sum()
if duplicates == 0:
    print("The data has no duplication.")
else:
    print(f"The data has {duplicates} duplicate rows.")
```

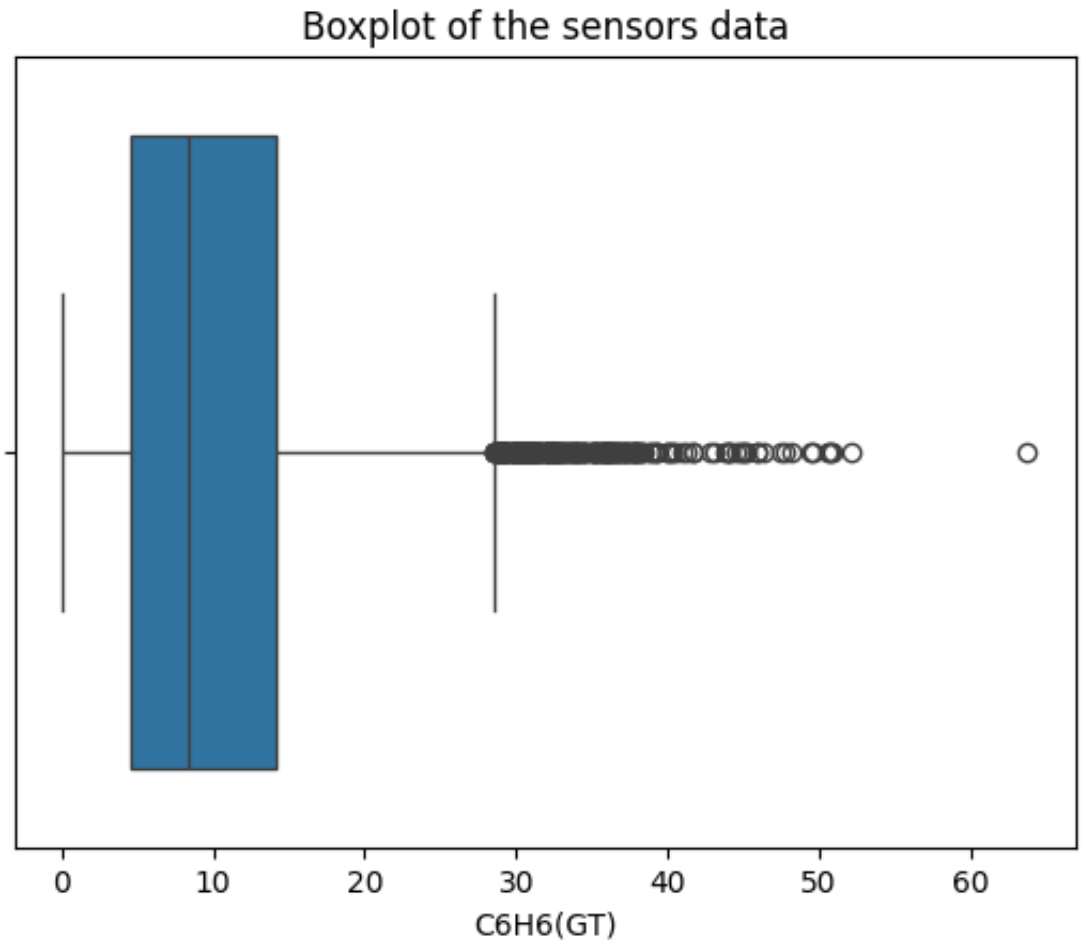
```
The data has no duplication.
```

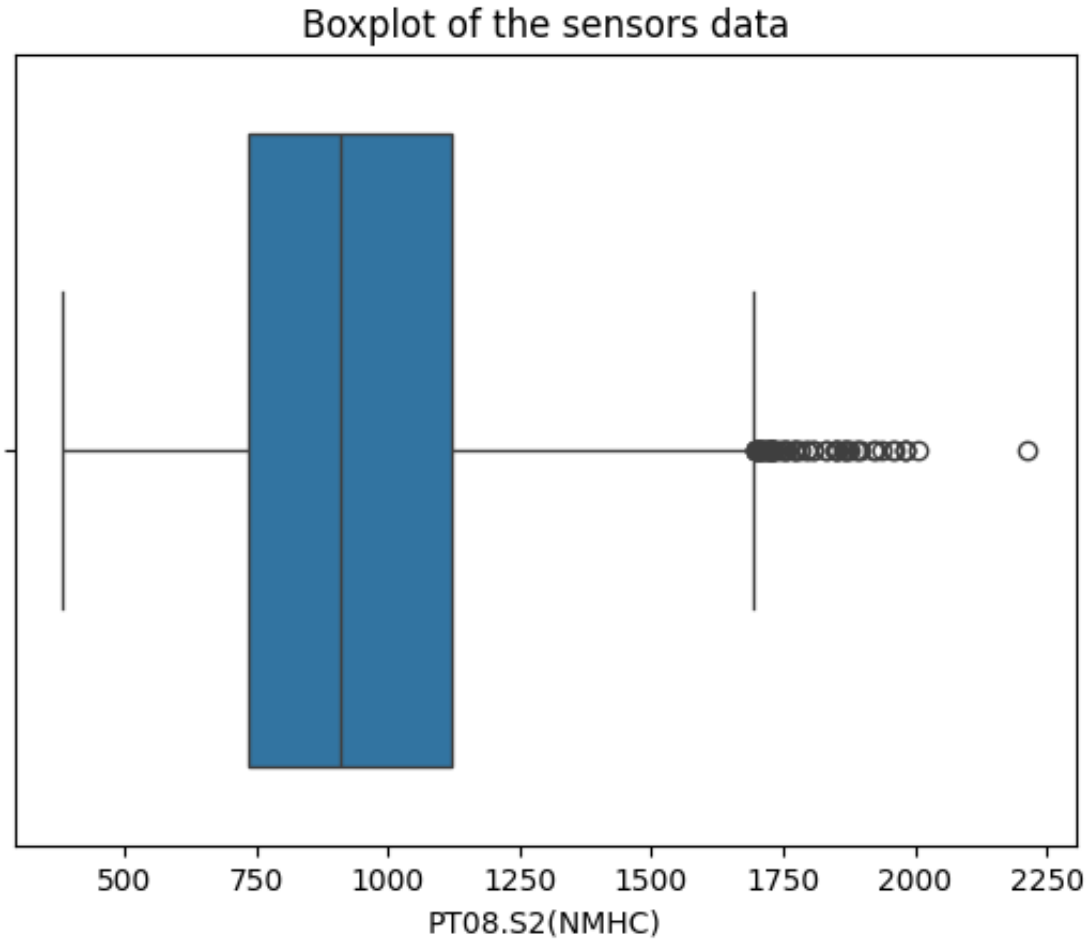
In [23]:
```
# Replace negatives with NaN and then impute (interpolation)

df.loc[df[i] < 0, i] = np.nan
df[i] = df[i].interpolate(method='linear')
```

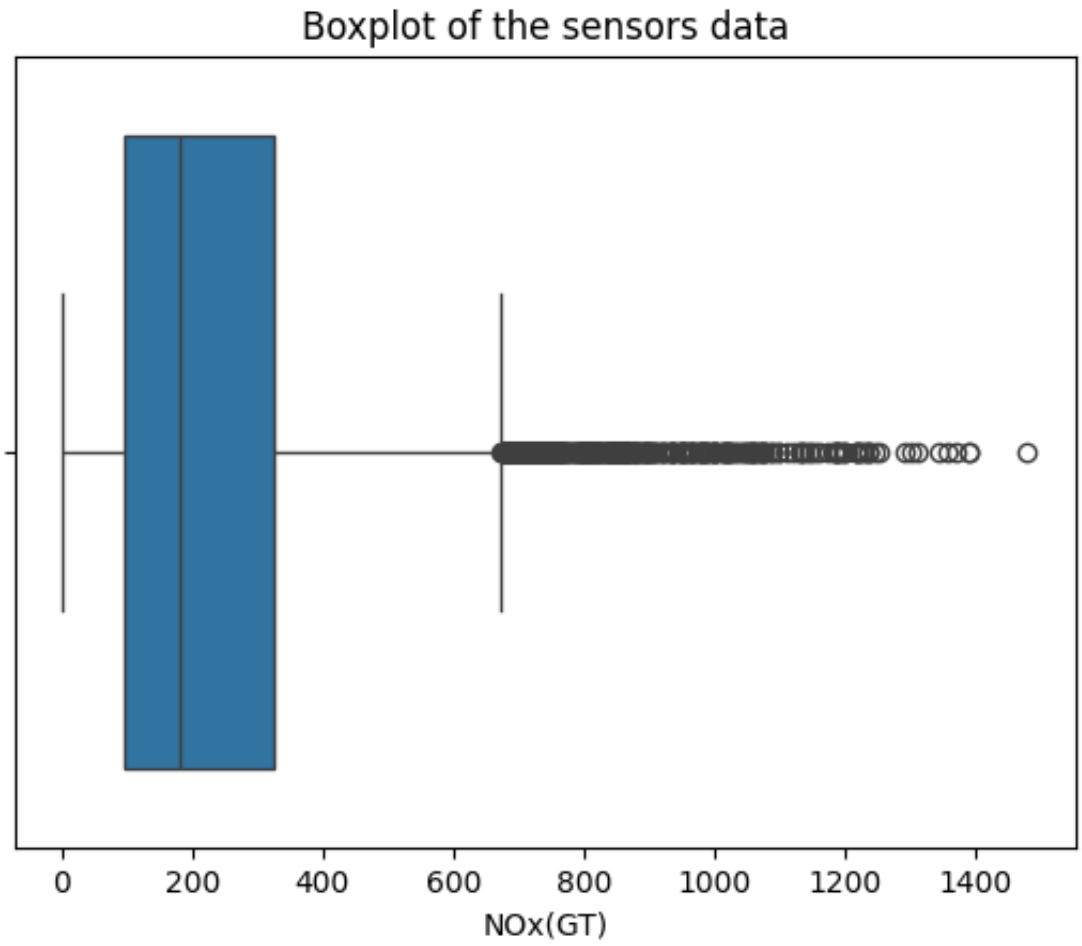
In [24]:
```
for i in df.columns[2:14]:
    df.loc[df[i] < 0, i] = np.nan
    df[i] = df[i].interpolate(method='linear')
    sns.boxplot(x=df[i])
    plt.title('Boxplot of the sensors data')
    plt.show()
```

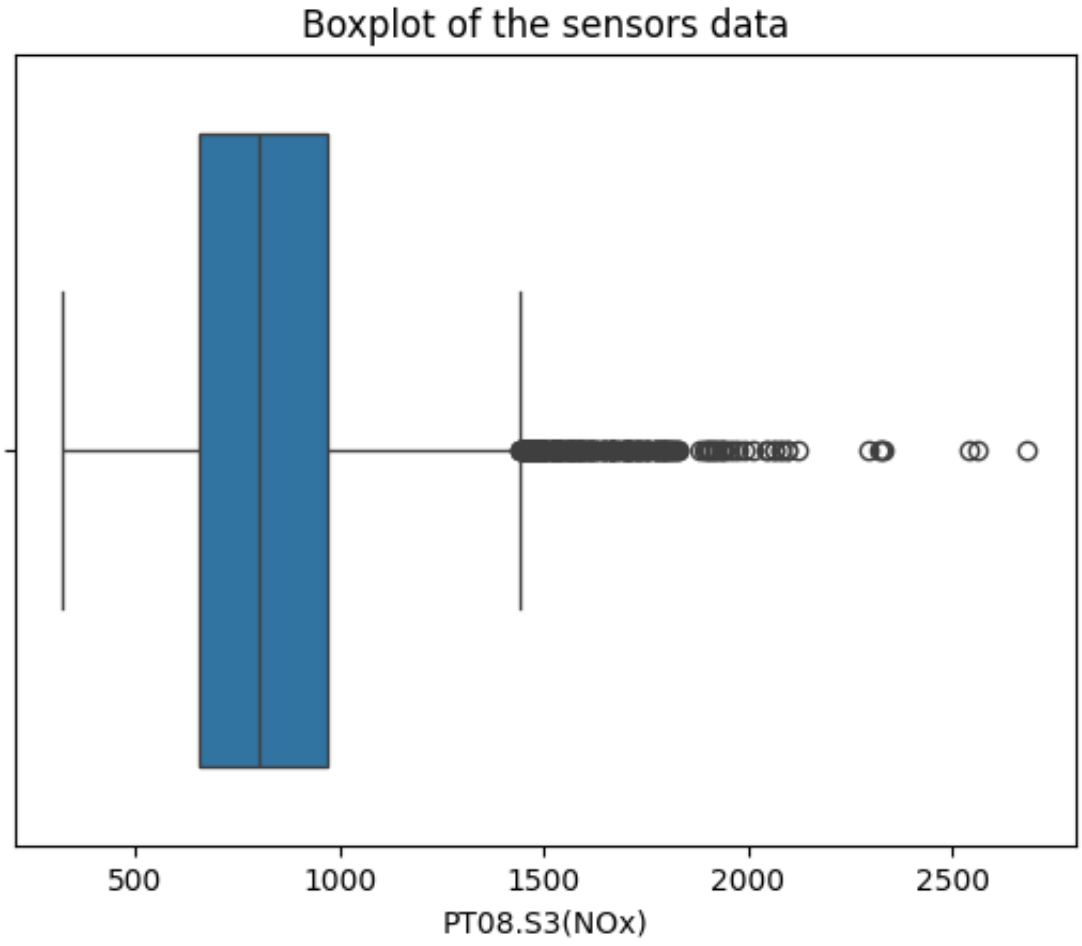

Boxplot of the sensors data
0
2
4
6
8
10
12
CO(GT)

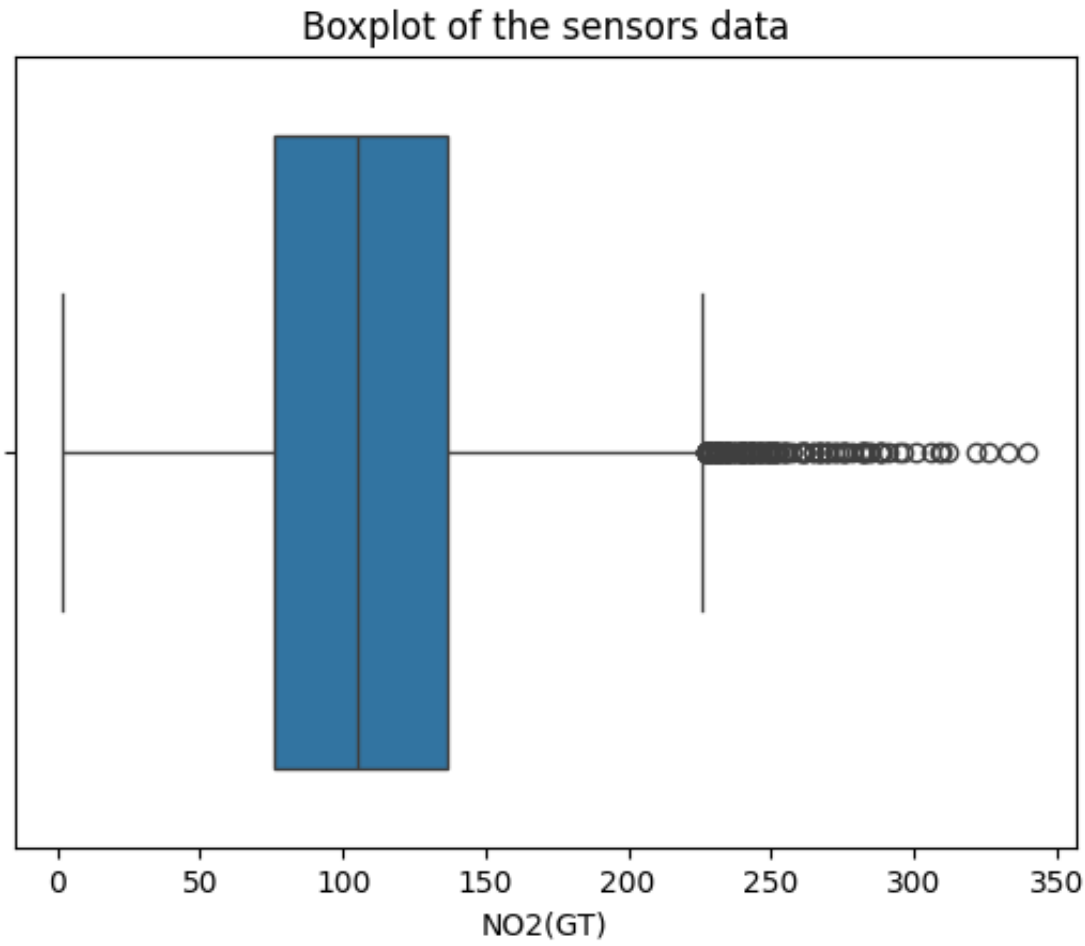

Boxplot of the sensors data
600
800
1000
1200
1400
1600
1800
2000
PT08.S1(CO)

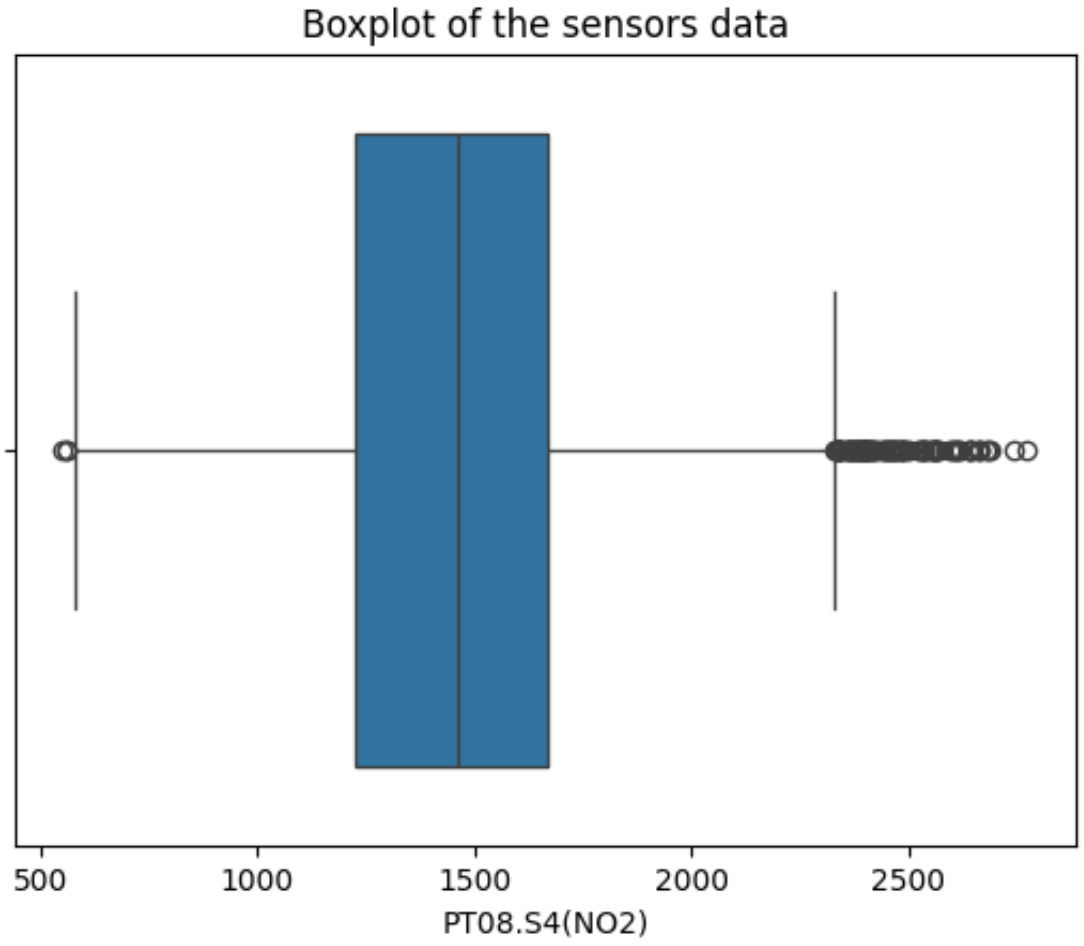

Boxplot of the sensors data
0
10
20
30
40
50
60
C6H6(GT)

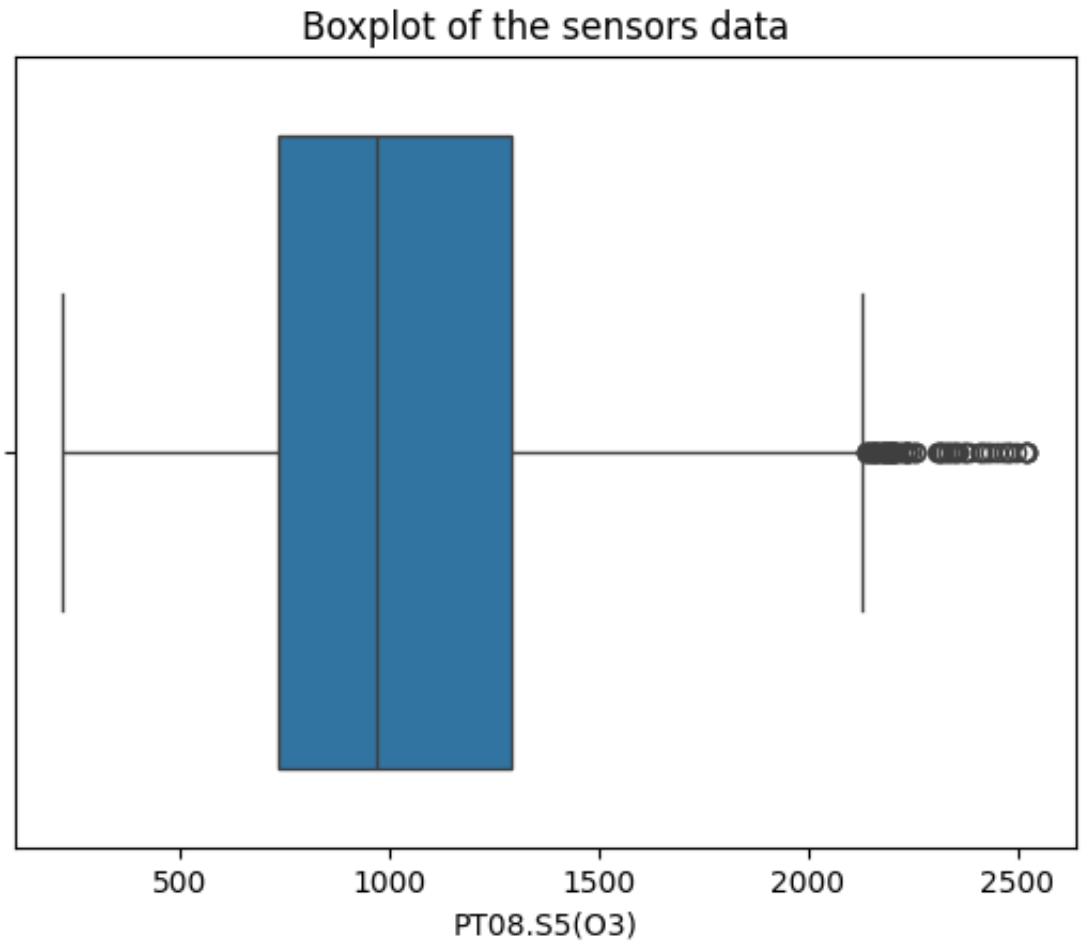

Boxplot of the sensors data
500
750
1000
1250
1500
1750
2000
2250
PT08.S2(NMHC)

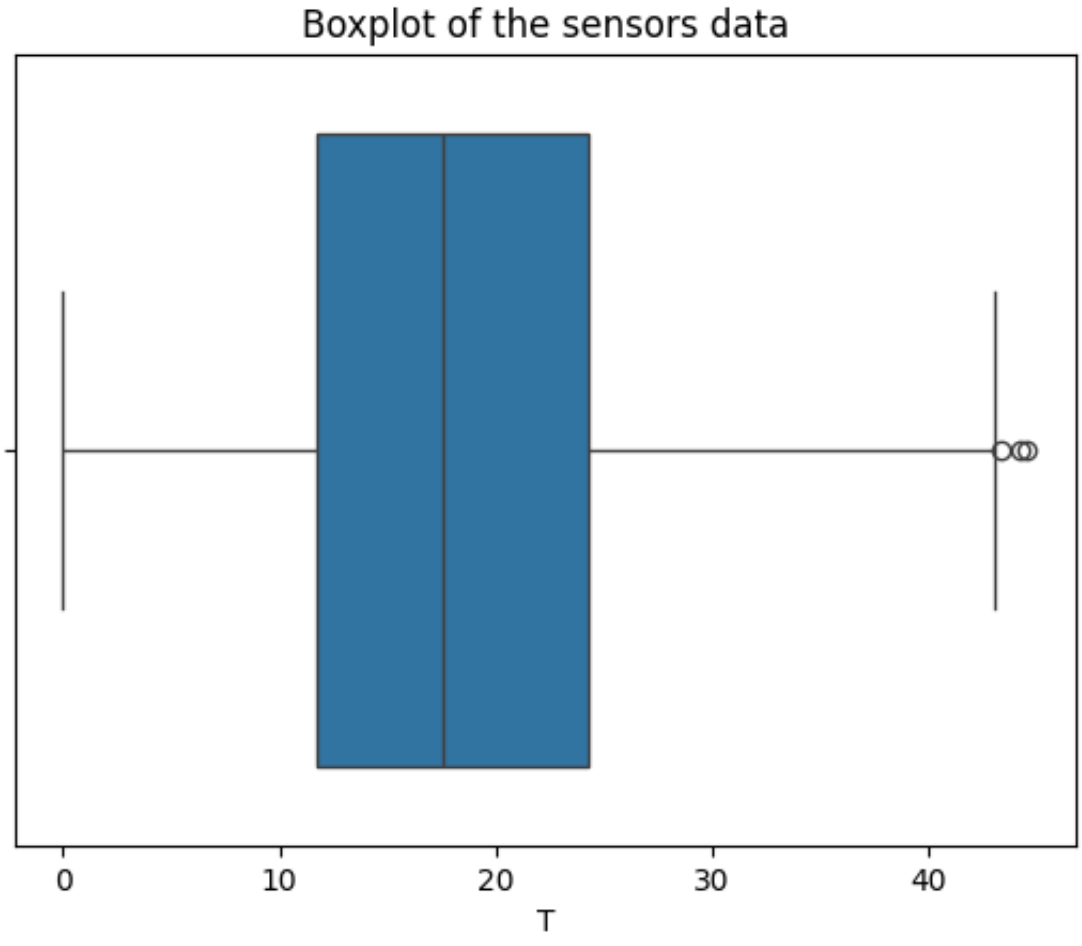

Boxplot of the sensors data
0
200
400
600
800
1000
1200
1400
NOx(GT)

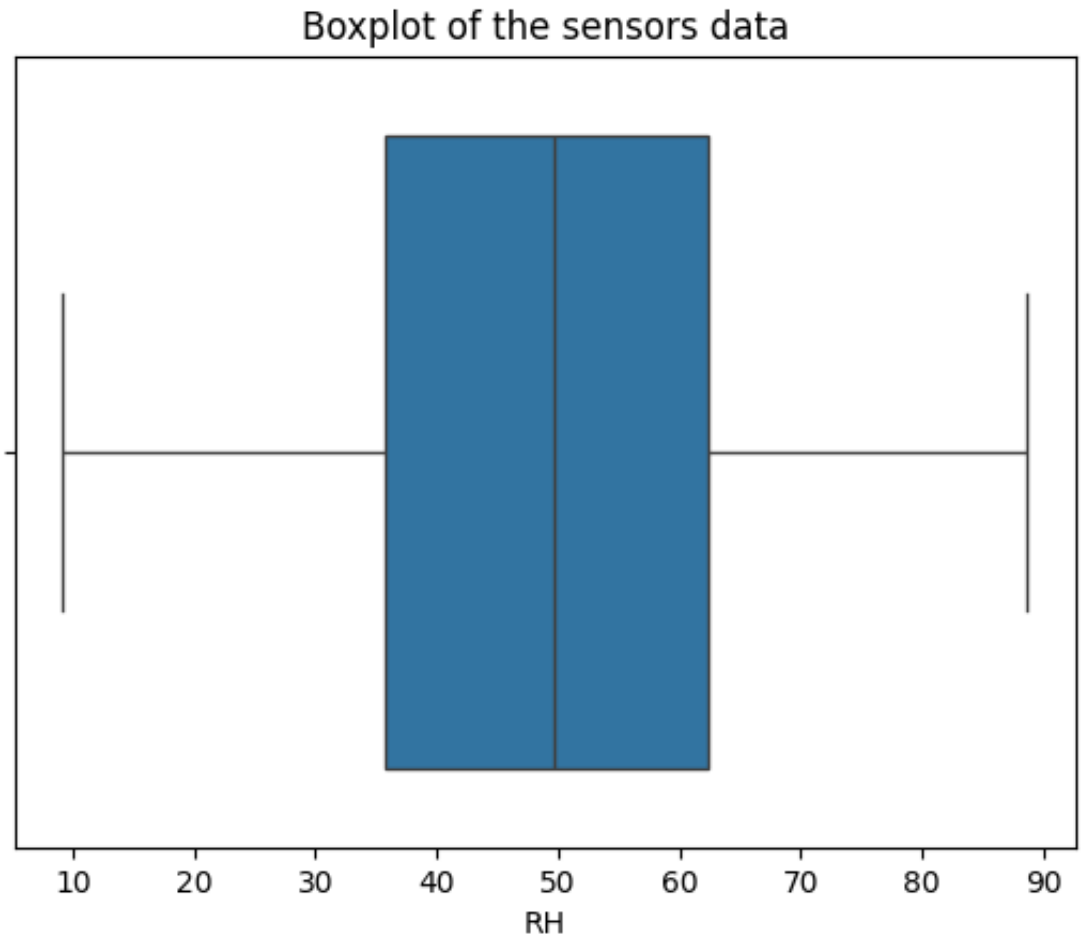

Boxplot of the sensors data
500
1000
1500
2000
2500
PT08.S3(NOx)

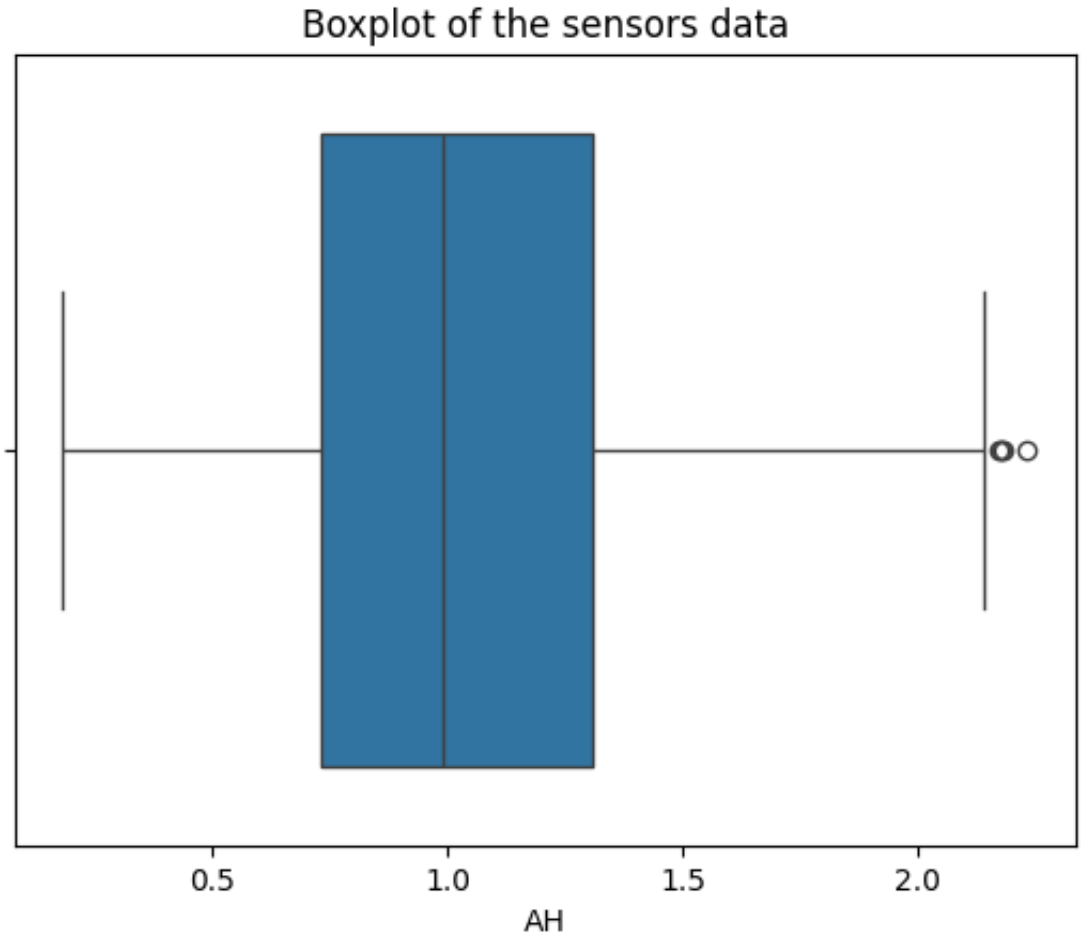

Boxplot of the sensors data
0
50
100
150
200
250
300
350
NO2(GT)


Boxplot of the sensors data
500
1000
1500
2000
2500
PT08.S4(NO2)


Boxplot of the sensors data
500
1000
1500
2000
2500
PT08.S5(O3)


Boxplot of the sensors data
0
10
20
30
40
T





In [25]:
```
#Removing Outliers with the Interquartile Range Method (IQR)
```

```
sensor_cols = df.columns[2:14]

Q1 = df[sensor_cols].quantile(0.25) #first 25% of the data
Q3 = df[sensor_cols].quantile(0.75) #first 75% of the data
IQR = Q3 - Q1 #IQR = InterQuartile Range

scale = 2 #For Normal Distributions, scale = 1.5
lower_lim = Q1 - scale*IQR
upper_lim = Q3 + scale*IQR

lower_outliers = (df[sensor_cols]< lower_lim)
upper_outliers = (df[sensor_cols] > upper_lim)
```

In [26]:
```
#Checking the resulting outliers calculated by the above method
df[sensor_cols][(lower_outliers | upper_outliers)].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9357 entries, 0 to 9356
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   CO(GT)         105 non-null    float64
 1   PT08.S1(CO)    22 non-null     float64
 2   C6H6(GT)       110 non-null    float64
 3   PT08.S2(NMHC)  13 non-null     float64
 4   NOx(GT)        242 non-null    float64
 5   PT08.S3(NOx)   111 non-null    float64
 6   NO2(GT)        49 non-null     float64
 7   PT08.S4(NO2)   28 non-null     float64
 8   PT08.S5(O3)    12 non-null     float64
 9   T              0 non-null      float64
 10  RH             0 non-null      float64
 11  AH             0 non-null      float64
dtypes: float64(12)
memory usage: 950.3 KB
```

In [27]:
```
#Create new DataFrame without the outliers

num_cols = list(sensor_cols)
df_IQR = df[~((df[num_cols] < (Q1 - 2 * IQR)) |(df[num_cols] > (Q3 
df_IQR.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8887 entries, 0 to 9356
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   Date           8887 non-null   datetime64[ns]
 1   Time           8887 non-null   object        
 2   CO(GT)         8887 non-null   float64       
 3   PT08.S1(CO)    8887 non-null   float64       
 4   C6H6(GT)       8887 non-null   float64       
 5   PT08.S2(NMHC)  8887 non-null   float64       
 6   NOx(GT)        8887 non-null   float64       
 7   PT08.S3(NOx)   8887 non-null   float64       
 8   NO2(GT)        8887 non-null   float64       
 9   PT08.S4(NO2)   8887 non-null   float64       
 10  PT08.S5(O3)    8887 non-null   float64       
 11  T              8887 non-null   float64       
 12  RH             8887 non-null   float64       
 13  AH             8887 non-null   float64       
dtypes: datetime64[ns](1), float64(12), object(1)
memory usage: 1.0+ MB
```

In [28]:
```
#Eliminating rows with NaN values

df_filt = df_IQR.dropna(how='any', axis=0)
df_filt.reset_index(drop=True,inplace=True)
```

In [29]:
```
#Adding a column with the week days

df_filt['Week Day'] = df_filt['Date'].dt.day_name()

#Rearranging columns

cols = df_filt.columns.tolist()
cols = cols[:1] + cols[-1:] + cols[1:11]
df_filt = df_filt[cols]
df_filt.head()
```

Out[29]:

	Date	Week Day	Time	CO(GT)	PT08.S1(CO)	C6H6(GT)	PT08.S2(
0	2004-03-10	Wednesday	18:00:00	2.6	1360.0	11.9	
1	2004-03-10	Wednesday	19:00:00	2.0	1292.0	9.4	
2	2004-03-10	Wednesday	20:00:00	2.2	1402.0	9.0	
3	2004-03-10	Wednesday	21:00:00	2.2	1376.0	9.2	
4	2004-03-10	Wednesday	22:00:00	1.6	1272.0	6.5	

Exploratory Data Analysis (EDA)

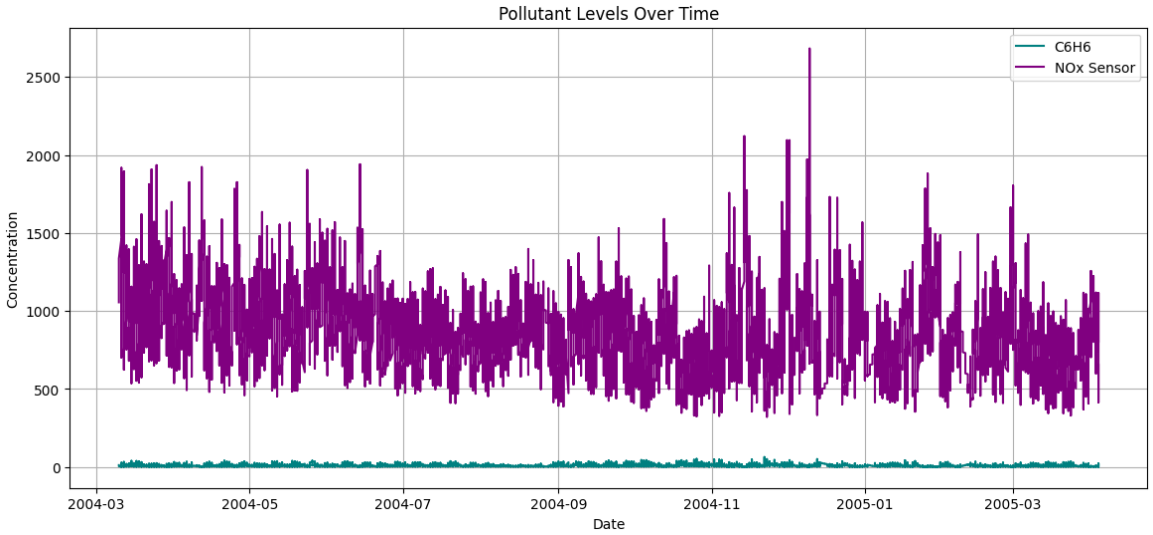
1 Normalized pollutant trends

- Normalize C6H6(GT), PT08.S3(NOx)
- Plot smoothed line chart

2 Monthly Boxplot for C6H6(GT)

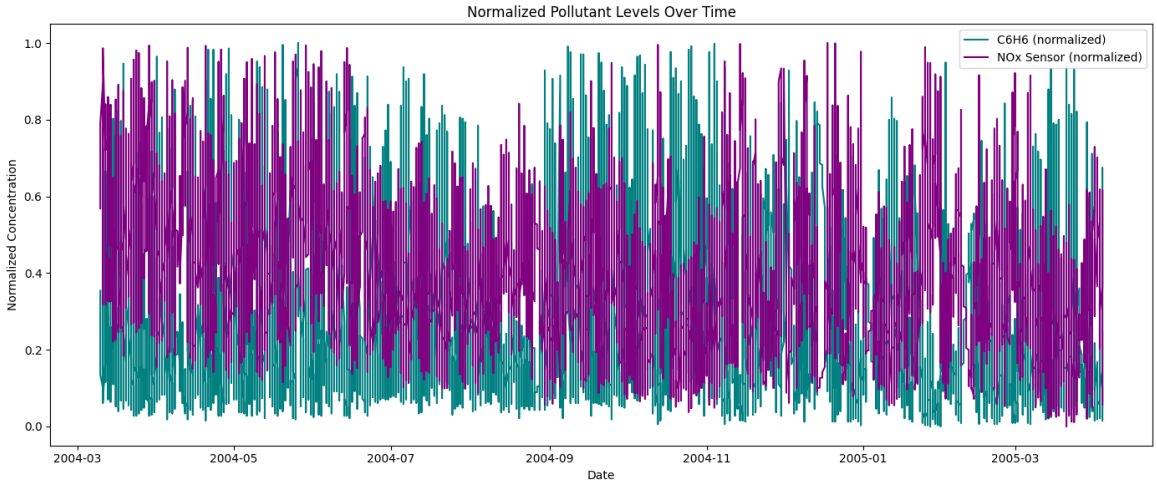
- Shows seasonal trend
- October highest, August lowest

In [30]:
```
plt.figure(figsize=(14, 6))
plt.plot(df['Date'], df['C6H6(GT)'], label='C6H6', color='teal')
plt.plot(df['Date'], df['PT08.S3(NOx)'], label='NOx Sensor', color=
plt.title('Pollutant Levels Over Time')
plt.xlabel('Date')
plt.ylabel('Concentration')
plt.legend()
plt.grid(True)
plt.show()
```

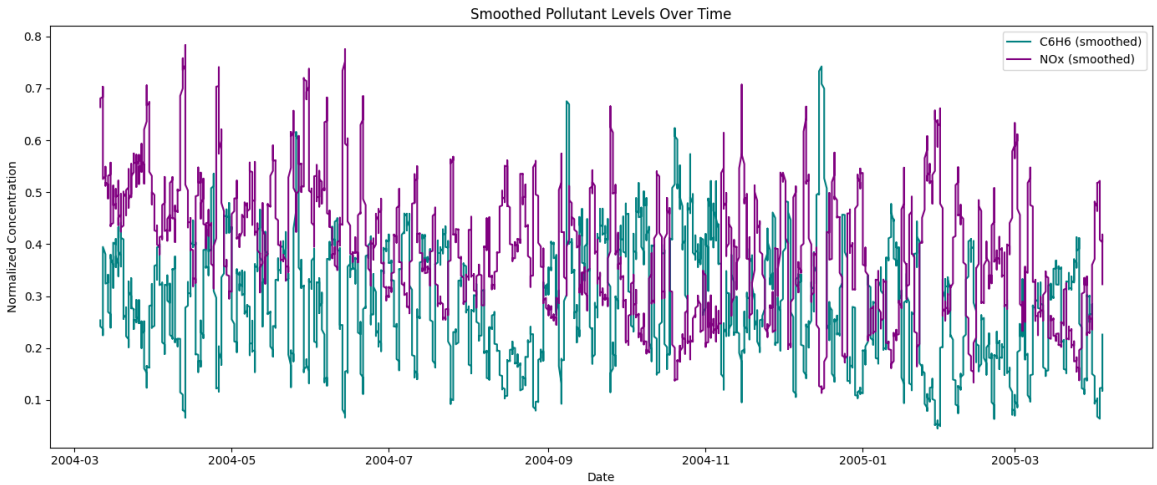


In [31]:
```
from sklearn.preprocessing import MinMaxScaler

# Select columns and normalize
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(df_filt[['C6H6(GT)', 'PT08.S

# Create a new DataFrame for plotting
df_normalized = df_filt[['Date']].copy()
df_normalized['C6H6_norm'] = normalized_data[:, 0]
df_normalized['NOx_norm'] = normalized_data[:, 1]

# Plot
plt.figure(figsize=(14, 6))
plt.plot(df_normalized['Date'], df_normalized['C6H6_norm'], label='
plt.plot(df_normalized['Date'], df_normalized['NOx_norm'], label='N
```

```
plt.title('Normalized Pollutant Levels Over Time')
plt.xlabel('Date')
plt.ylabel('Normalized Concentration')
plt.legend()
plt.tight_layout()
plt.show()
```

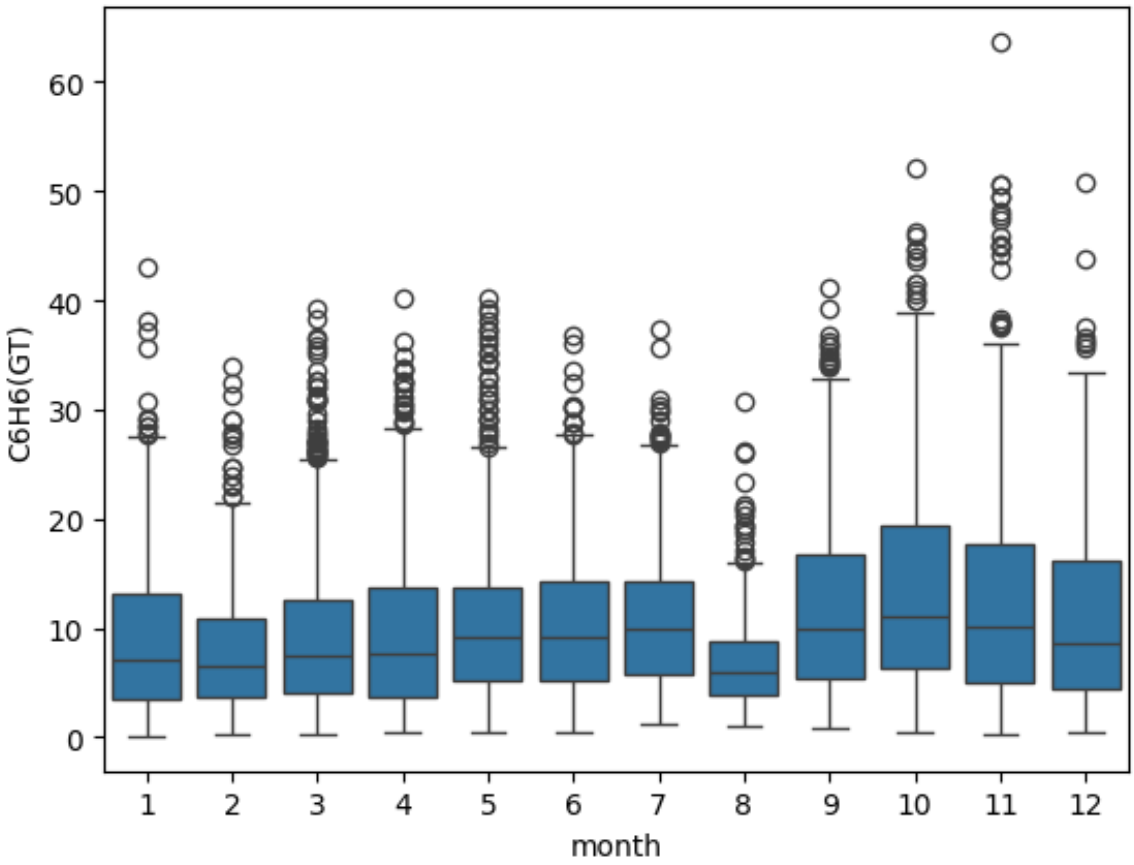


In [32]:
```
df_normalized['C6H6_smooth'] = df_normalized['C6H6_norm'].rolling(w
df_normalized['NOx_smooth'] = df_normalized['NOx_norm'].rolling(win

plt.figure(figsize=(14,6))
plt.plot(df_normalized['Date'], df_normalized['C6H6_smooth'], label
plt.plot(df_normalized['Date'], df_normalized['NOx_smooth'], label=
plt.title('Smoothed Pollutant Levels Over Time')
plt.xlabel('Date')
plt.ylabel('Normalized Concentration')
plt.legend()
plt.tight_layout()
plt.show()
```



In [33]:
```
df_normalized[df_normalized['Date'].between('2004-07', '2004-08')]
df['month'] = df['Date'].dt.month
sns.boxplot(x='month', y='C6H6(GT)', data=df)
```

Out[33]: <Axes: xlabel='month', ylabel='C6H6(GT)'>



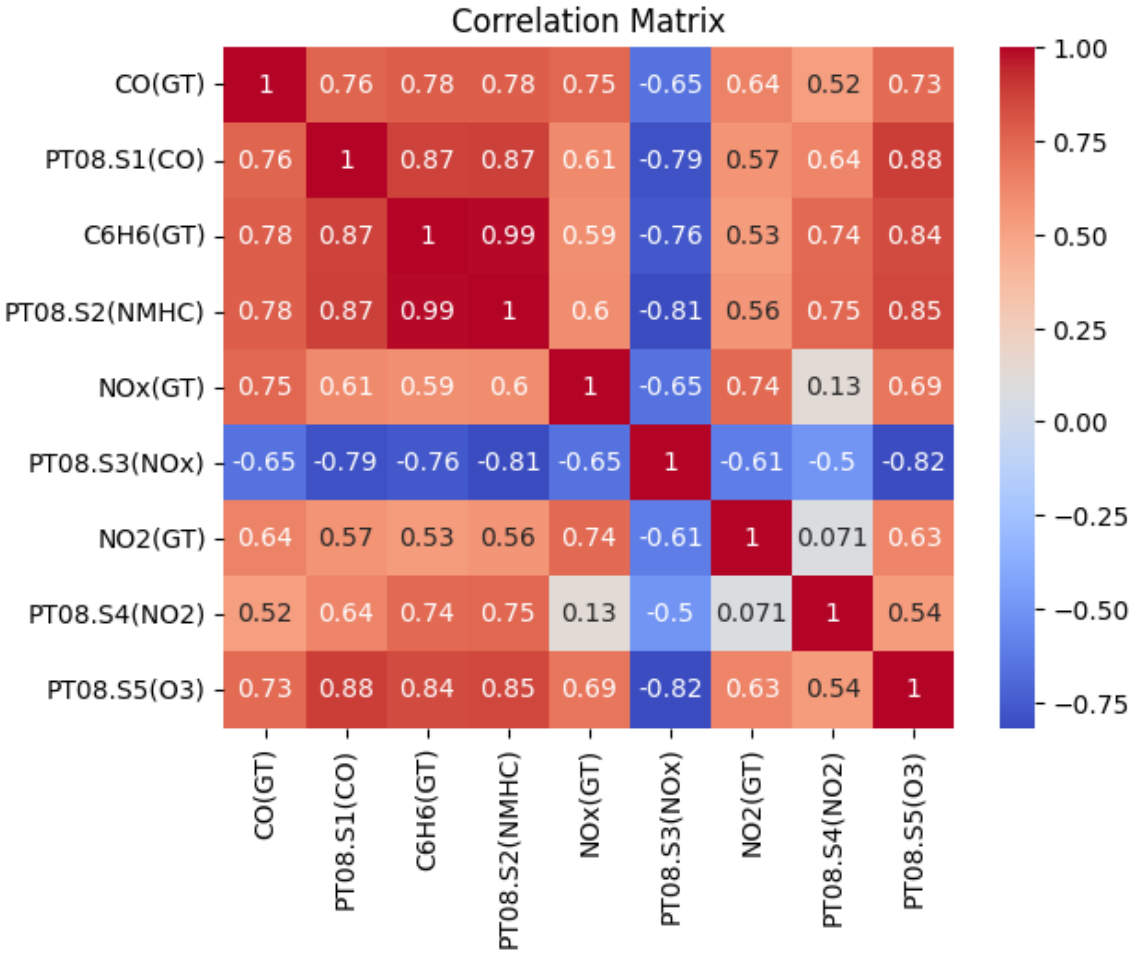
In [34]:
```
# Group by month and get median pollutant value

monthly_median = df.groupby('month')['C6H6(GT)'].median()
highest = monthly_median.idxmax()
lowest = monthly_median.idxmin()

print(f"Highest pollution month (median): {highest}")
print(f"Lowest pollution month (median): {lowest}")
```

```
Highest pollution month (median): 10
Lowest pollution month (median): 8
```

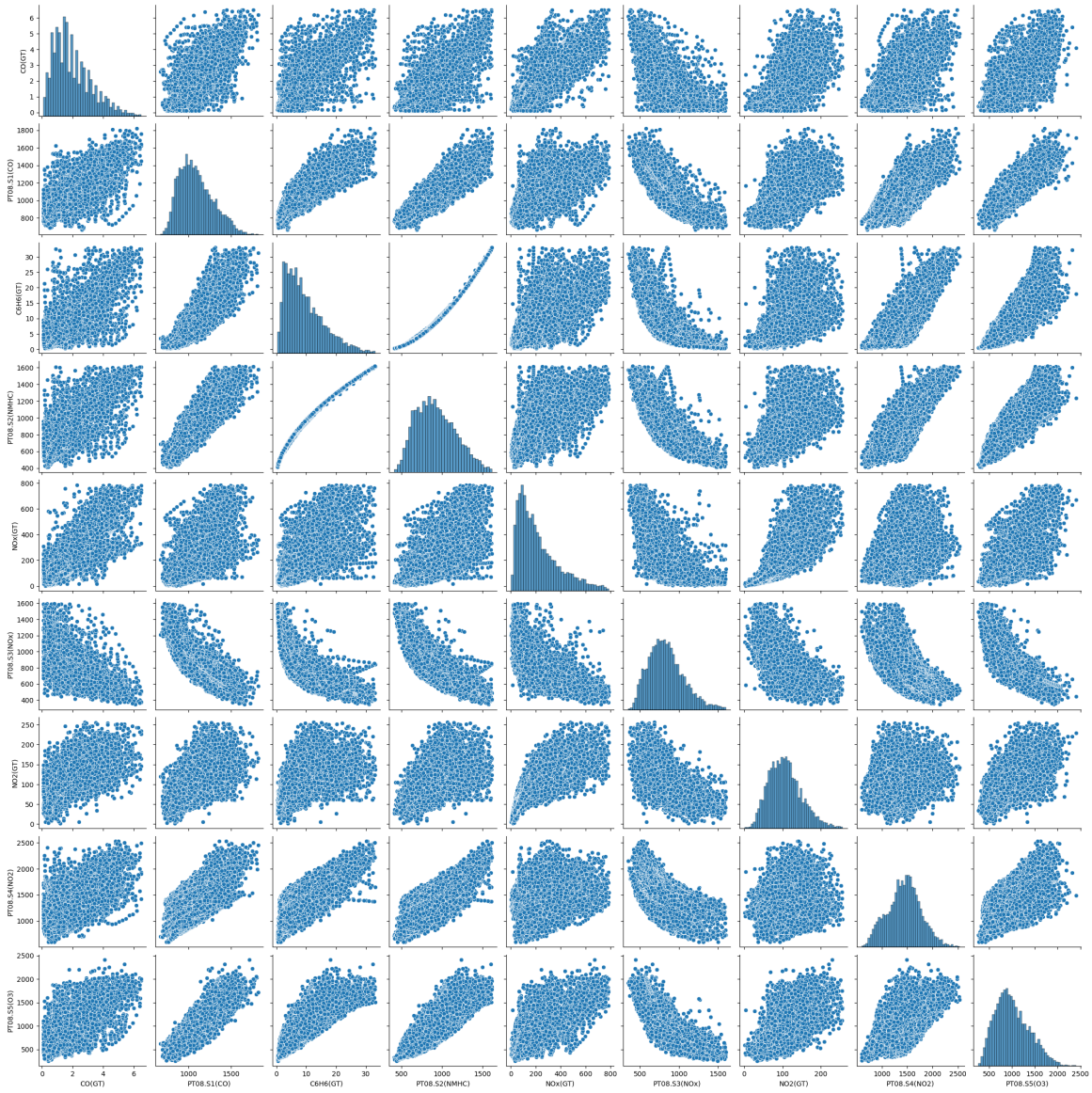
Correlation Analysis

- Heatmap + Pairplot
- Drop C6H6(GT) due to high correlation with NMHC(GT)

In [35]:
```
numeric_df = df_filt.select_dtypes(include='number')
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



In [36]:
```
sns.pairplot(df_filt)
plt.show()
```



In [37]:
```
#Eliminating the C6H6(GT) for being a redundant sensor (the C6HC mo
 #so the correlation between those two sensor is exact

df_filt.drop('C6H6(GT)', axis=1, inplace=True)
```

Regression Modeling (Random Forest)

- Target: PT08.S1(CO)
- Model: RandomForestRegressor
- Split: 80% train, 20% test

In [38]:
```
from sklearn.model_selection import train_test_split

Y = df_filt['PT08.S1(CO)'] #variable for prediction
X = df_filt.drop(['PT08.S1(CO)','Date', 'Time', 'Week Day'], axis=1

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
print(X_train.shape, X_test.shape)
```

```
(7109, 7) (1778, 7)
```

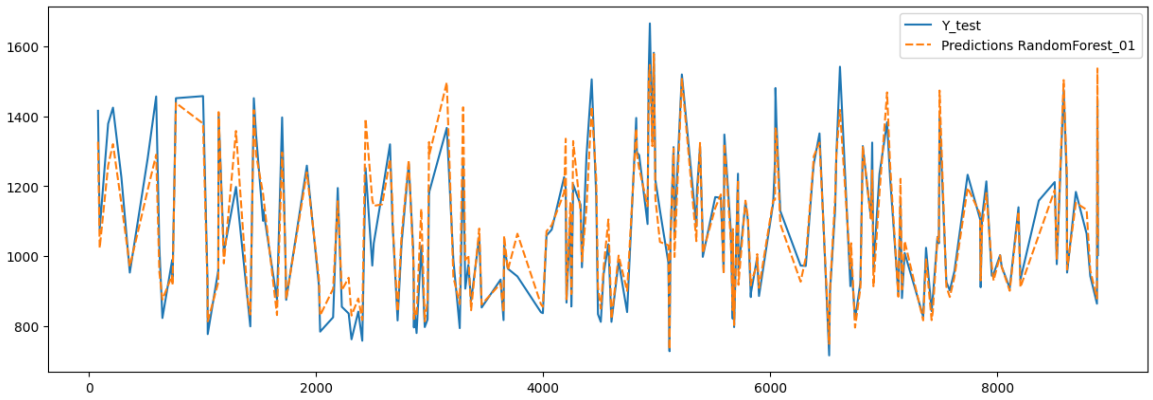
In [39]:
```
from sklearn.ensemble import RandomForestRegressor

modelo_randomforest = RandomForestRegressor(n_estimators=100)

modelo_randomforest.fit(X_train, Y_train)
```

Out[39]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [40]:
```
#Evaluating the results with the R² metric:
#Test data evaluation

from sklearn import metrics

pred_randomforest = modelo_randomforest.predict(X_test) #predicted

print('Random Forest Regression Model: R²={:.2f}'.format(metrics.r2
```

```
Random Forest Regression Model: R²=0.92
```

In [41]:
```
#Comparing model predictions with the Ground Truth Test Data

DF = pd.DataFrame()

DF['Y_test'] = Y_test
DF['Predictions RandomForest_01'] = pred_randomforest

plt.figure(figsize=(15,5))
sns.lineplot(data=DF.iloc[:200,:])
plt.show()
```



In [42]:
```
#Extracting season information from Date column:

def season(date):
    year = str(date.year)
    seasons = {'spring': pd.date_range(start='20/03/'+year, end='20
               'summer': pd.date_range(start='21/06/'+year, end='22
               'autumn': pd.date_range(start='22/09/'+year, end='20
    if date in seasons['spring']:
        return 'spring'
    if date in seasons['summer']:
        return 'summer'
```

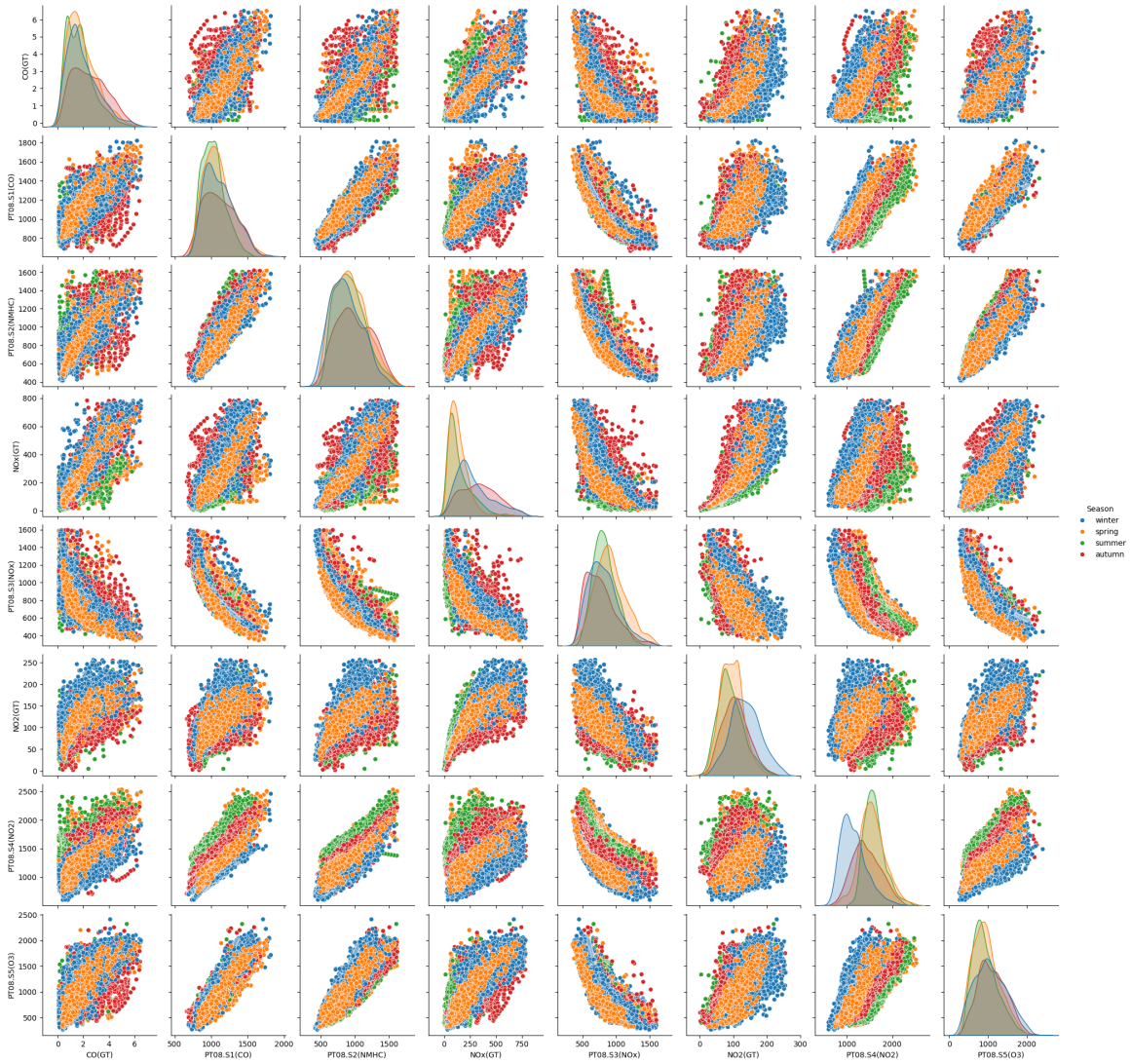
```
    if date in seasons['autumn']:
        return 'autumn'
    else:
        return 'winter'

df_filt['Season'] = df_filt['Date'].map(season)
df_filt.head()
```

Out[42]:

	Date	Week Day	Time	CO(GT)	PT08.S1(CO)	PT08.S2(NMHC)	NO
0	2004-03-10	Wednesday	18:00:00	2.6	1360.0	1046.0	
1	2004-03-10	Wednesday	19:00:00	2.0	1292.0	955.0	
2	2004-03-10	Wednesday	20:00:00	2.2	1402.0	939.0	
3	2004-03-10	Wednesday	21:00:00	2.2	1376.0	948.0	
4	2004-03-10	Wednesday	22:00:00	1.6	1272.0	836.0	

In [43]:

```
sns.pairplot(df_filt, hue='Season')
plt.show()
```



In [44]:
```
#Creating categorical features from Season column and splitting new

Y_2 = df_filt['PT08.S1(CO)']
X_2 = df_filt.drop(['PT08.S1(CO)','Date', 'Time', 'Week Day'], axis
X_2 = pd.get_dummies(data=X_2)

X_2_train, X_2_test, Y_2_train, Y_2_test = train_test_split(X_2, Y_
X_2.head()
```

Out[44]:

	CO(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(N
0	2.6	1046.0	166.0	1056.0	113.0	169
1	2.0	955.0	103.0	1174.0	92.0	155
2	2.2	939.0	131.0	1140.0	114.0	155
3	2.2	948.0	172.0	1092.0	122.0	158
4	1.6	836.0	131.0	1205.0	116.0	149

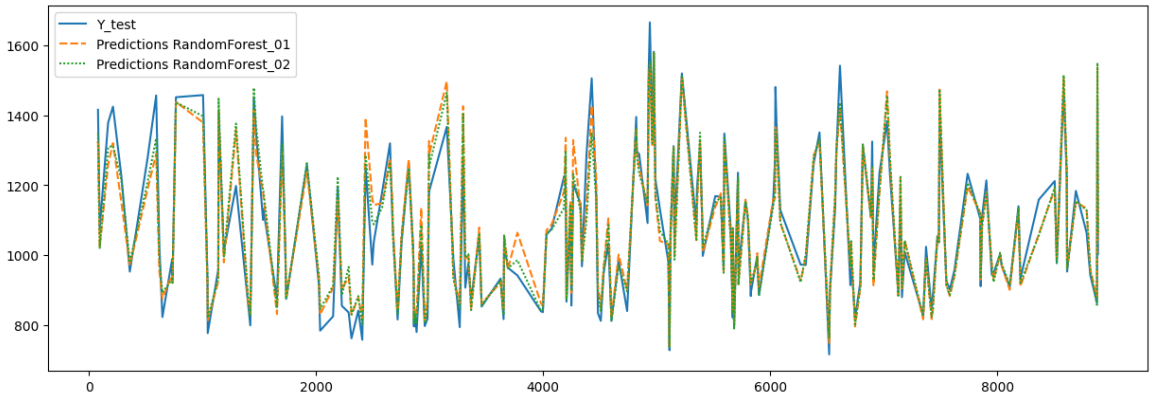
In [45]:
```
modelo_randomforest_2 = RandomForestRegressor()

modelo_randomforest_2.fit(X_2_train, Y_2_train)
```

Out[45]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [46]:
```
pred_randomforest_2 = modelo_randomforest_2.predict(X_2_test)

print('Regression Model without Seasons: R²={:.2f}'.format(metrics.
print('Regression Model with Seasons: R²={:.2f}'.format(metrics.r2_
```

```
Regression Model without Seasons: R²=0.92
Regression Model with Seasons: R²=0.93
```

In [47]:
```
DF['Predictions RandomForest_02'] = pred_randomforest_2

plt.figure(figsize=(15,5))
sns.lineplot(data=DF.iloc[:200,:])
plt.show()
```



Summary

- Pollution peaks in colder months (benzene)
- $R^2 \sim 0.93$ → Good model performance
- Seasonality improves prediction
- Remove highly correlated features