Start coding or generate with AI.

Start coding or generate with AI.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler

# Read the CSV file from GitHub
url = 'https://raw.githubusercontent.com/kssandraeshwar/Data-Visualisation_course/refs/heads/main/AirQuality.csv'
df = pd.read_csv(url)

# Initial Data Exploration
print("Original Dataset Information:")
print(df.info())

# Identify numeric and categorical columns
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
categorical_columns = df.select_dtypes(include=['object']).columns

# Handle Negative Values
def handle_negative_values(df, columns):
    for col in columns:
        # Check if column has negative values
        if (df[col] < 0).any():
            print(f"Negative values found in {col}")

            # Replace negative values with absolute values
            df[col] = np.abs(df[col])
    return df

# Apply negative value handling
df = handle_negative_values(df, numeric_columns)

# Detailed Missing Value Analysis
print("\nMissing Values:")
missing_values = df.isnull().sum()
missing_percentages = 100 * df.isnull().sum() / len(df)
missing_table = pd.concat([missing_values, missing_percentages], axis=1, keys=['Missing Values', 'Percentage'])
print(missing_table)

# Comprehensive Missing Value Handling
# For numeric columns
for col in numeric_columns:
    # Handle extreme outliers with the median
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Replace extreme values with median
    df.loc[(df[col] < lower_bound) | (df[col] > upper_bound), col] = df[col].median()

    # Fill remaining missing values with median
    df[col].fillna(df[col].median(), inplace=True)

# For categorical columns
for col in categorical_columns:
    # Fill missing values with mode
    df[col].fillna(df[col].mode()[0], inplace=True)

# Remove Duplicate Rows
df.drop_duplicates(inplace=True)

# Normalization
# Create a copy of numeric columns
numeric_df = df[numeric_columns]

# Apply Min-Max Scaling to ensure all values are between 0 and 1
min_max_scaler = MinMaxScaler()
df_normalized = pd.DataFrame(
```

```
df_normalized = pd.DataFrame(
    min_max_scaler.fit_transform(numeric_df),
    columns=numeric_columns,
    index=numeric_df.index
)

# Combine Normalized Numeric and Original Categorical Columns
df_final = pd.concat([df_normalized, df[categorical_columns]], axis=1)

# Final Dataset Information
print("\nFinal Preprocessed Dataset Information:")
print(df_final.info())

# Descriptive Statistics
print("\nDescriptive Statistics:")
print(df_final.describe())

# Save Preprocessed Data
df_final.to_csv('preprocessed_air_quality.csv', index=False)
print("Preprocessing Complete. Preprocessed file saved as 'preprocessed_air_quality.csv'")

# Dataset dimensions
numeric_columns_count = len(numeric_columns)
numeric_rows = df.shape[0]
print('Number of numeric columns:', numeric_columns_count)
print('Number of rows:', numeric_rows)

# Dataset data types
data_types = df.dtypes
print('Data types:', data_types)

# Check null values
null_values = df.isnull().sum()
print('Null values present:', null_values)

#Boxplot to identify the pollutant Levels, by printing the numeric values of each


pollutant_columns = [
    'CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)',
    'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)'
]

# Ensuring pollutants are absolute
df[pollutant_columns] = df[pollutant_columns].abs()

# Plotting the boxplot
plt.figure(figsize=(14, 7))
sns.boxplot(data=df[pollutant_columns], palette="pastel")

# Adding numeric values above each box
for col_index, column in enumerate(pollutant_columns):
    col_data = df[column].dropna()  # Exclude nulls for correct stats
    median_value = col_data.median()
    plt.text(x=col_index, y=median_value + (0.05 * median_value),
             s=f"{median_value:.2f}", ha='center', color='black', fontsize=10)

plt.title("Boxplot of Pollutant Levels", fontsize=16)
plt.ylabel("Pollutant Levels", fontsize=12)
plt.xlabel("Pollutants", fontsize=12)
plt.xticks(ticks=range(len(pollutant_columns)), labels=pollutant_columns, rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
# Contribution of pollutants pie chart
pollutants = ['CO(GT)','PT08.S1(CO)','NMHC(GT)','C6H6(GT)','PT08.S2(NMHC)','NOx(GT)','PT08.S3(NOx)','NO2(GT)','PT08.S4(NO2)','PT08.S5(O3)']
existing_pollutants = [pollutant for pollutant in pollutants if pollutant in df.columns]

if existing_pollutants:
    pollutant_contributions = df[existing_pollutants].sum()
    plt.figure(figsize=(8, 8))
    pollutant_contributions.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors=sns.color_palette('pastel'))
    plt.title('Contribution of Pollutants')
    plt.ylabel('')  # Remove default ylabel for better aesthetics
    plt.show()
else:
    print("None of the specified pollutants are found in the dataset.")
```

```
# Scatter Plot for NOx vs Temperature
if 'NOx(GT)' in df.columns:
    # Plotting the histogram
    plt.figure(figsize=(10, 6))
    plt.hist(df['NOx(GT)'].dropna(), bins=30, color='skyblue', edgecolor='black', alpha=0.7)
    plt.title('Distribution of NOx')
    plt.xlabel('NOx Concentration')
    plt.ylabel('Frequency')
    plt.grid(axis='y', alpha=0.75)
    plt.show()
else:
    print("NOx column not found in the dataset.")
if 'NOx(GT)' in df.columns and 'T' in df.columns:
    # Plotting the scatter plot
    plt.figure(figsize=(10, 6))
    plt.scatter(df['T'], df['NOx(GT)'], color='blue', alpha=0.6)
    plt.title('Scatter Plot of NOx vs Temperature')
    plt.xlabel('Temperature (°C)')
    plt.ylabel('NOx Concentration')
    plt.grid(True)
    plt.show()
else:
    print("One or both of the columns 'NOx' and 'Temperature' are not found in the dataset.")

# Line Chart for the trend of C6H6 over Time
# Ensure 'Start Time' and 'C6H6' columns exist
if 'Time' in df.columns and 'C6H6(GT)' in df.columns:
    # Convert 'Start Time' to datetime format
    df['Time'] = pd.to_datetime(df['Time'], errors='coerce')

    # Drop rows where 'C6H6' or 'Start Time' is NaT or NaN
    df = df.dropna(subset=['Time', 'C6H6(GT)'])

    # Plotting the line chart
    plt.figure(figsize=(12, 6))
    plt.plot(df['Time'], df['C6H6(GT)'], color='green', marker='o', linestyle='-', alpha=0.7)
    plt.title('Trend of C6H6 (Benzene) Over Time')
    plt.xlabel('Time')
    plt.ylabel('C6H6 Concentration (ppm)')
    plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
    plt.grid()
    plt.tight_layout()  # Adjust layout to prevent clipping of labels
    plt.show()
else:

    print("One or both of the columns 'Start Time' and 'C6H6' are not found in the dataset.")

#Stacked Bar Chart for Monthly Pollutant Levels
df['Time'] = pd.to_datetime(df['Time'], errors='coerce')

# Extract month and year from 'Start Time'
df['Month'] = df['Time'].dt.month
df['Year'] = df['Time'].dt.year

# Group by Year and Month and calculate the sum of each pollutant (assuming columns are named accordingly)
monthly_pollutants = df.groupby(['Year', 'Month'])[['NOx(GT)', 'C6H6(GT)', 'T']].sum().reset_index()
monthly_pollutants_pivot = monthly_pollutants.pivot(index='Month', columns='Year', values=['NOx(GT)', 'C6H6(GT)', 'T'])

# Plotting the stacked bar chart
monthly_pollutants_pivot.plot(kind='bar', stacked=True, figsize=(12, 7))

plt.title('Monthly Pollutant Levels')
plt.xlabel('Month')
plt.ylabel('Concentration')
plt.xticks(rotation=0)
plt.legend(title='Year', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y')
plt.tight_layout()  # Adjust layout
plt.show()

#Draw the Bar Chart for hourly average NOx
# Ensure 'Start Time' is in datetime format
df['Time'] = pd.to_datetime(df['Time'], errors='coerce')

# Extract hour from 'Start Time'
df['Hour'] = df['Time'].dt.hour

# Calculate hourly average of NOx (assuming NOx is in a column named 'NOx')
```

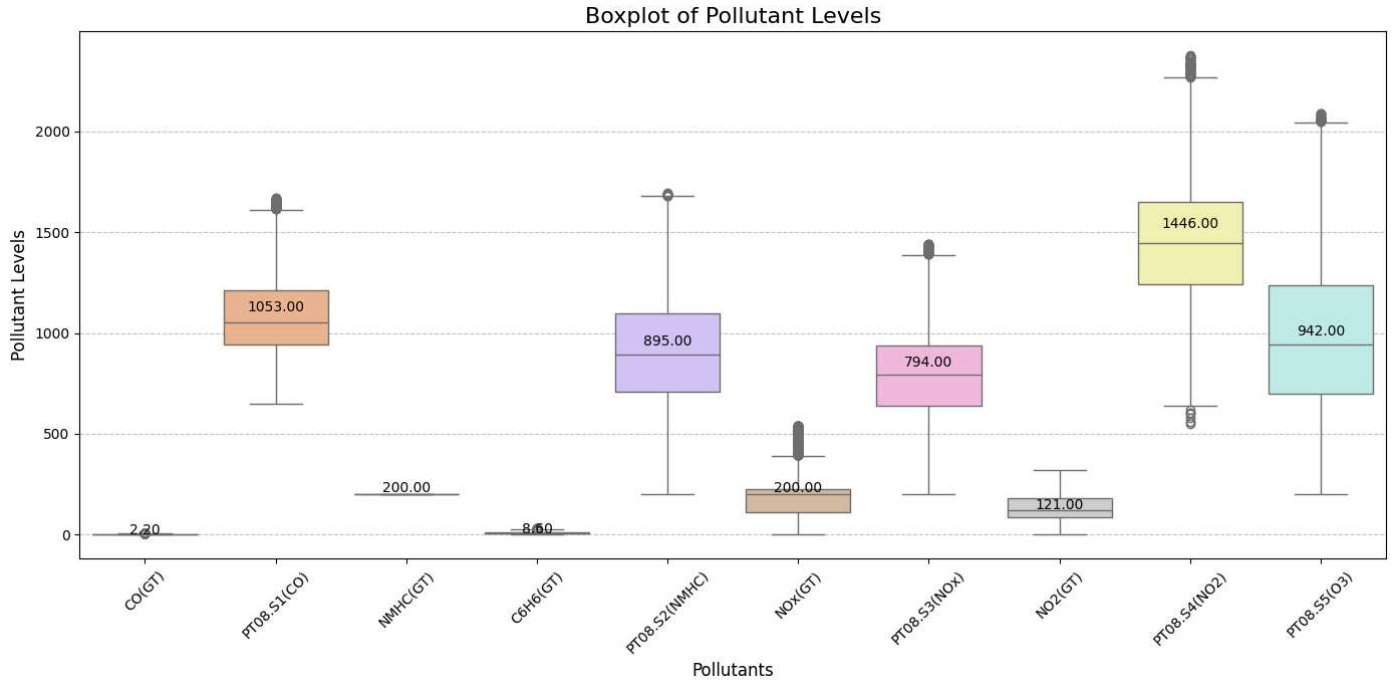
```
# Calculate hourly average of NOx (assuming NOx is in a column named 'NOx')
hourly_avg_nox = df.groupby('Hour')['NOx(GT)'].mean()

# Plotting the bar chart
plt.figure(figsize=(10, 6))
hourly_avg_nox.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Hourly Average NOx Levels')
plt.xlabel('Hour of Day')
plt.ylabel('Average NOx Concentration (ppm)')
plt.xticks(rotation=0)  # Rotate x-axis labels for better readability
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

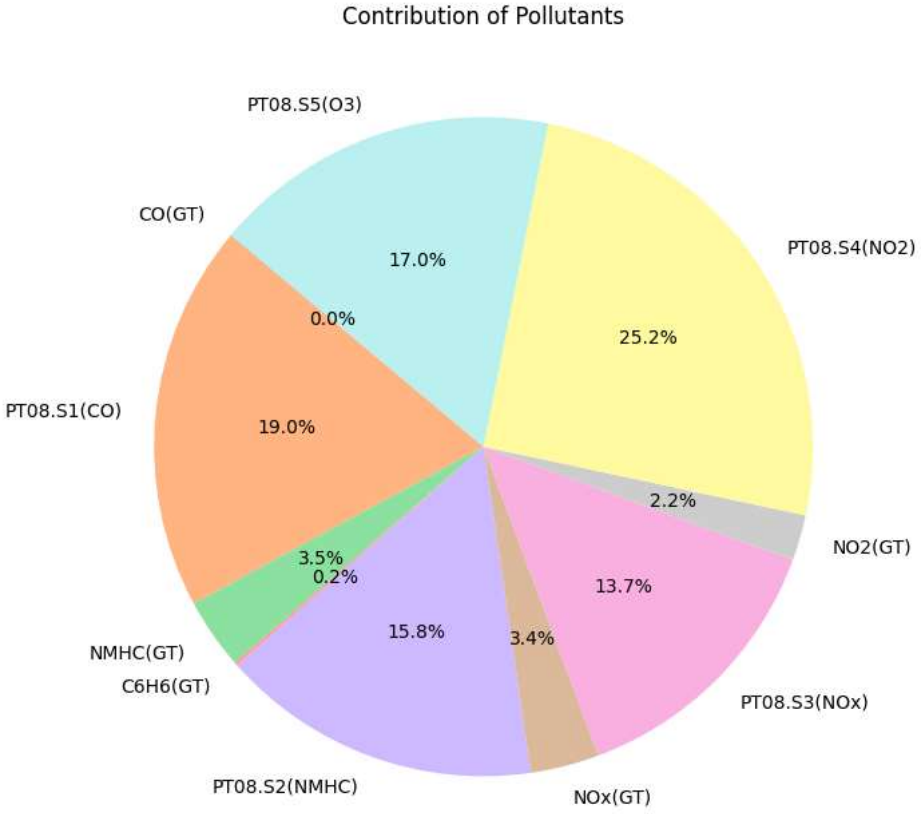
```
Original Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Date           9357 non-null   object 
 1   Time           9357 non-null   object 
 2   CO(GT)         9357 non-null   float64
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 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
 15  Unnamed: 15    0 non-null      float64
 16  Unnamed: 16    0 non-null      float64
dtypes: float64(15), object(2)
memory usage: 1.2+ MB
None
Negative values found in CO(GT)
Negative values found in PT08.S1(CO)
Negative values found in NMHC(GT)
Negative values found in C6H6(GT)
Negative values found in PT08.S2(NMHC)
Negative values found in NOx(GT)
Negative values found in PT08.S3(NOx)
Negative values found in NO2(GT)
Negative values found in PT08.S4(NO2)
Negative values found in PT08.S5(O3)
Negative values found in T
Negative values found in RH
Negative values found in AH

Missing Values:
               Missing Values  Percentage
Date                      114    1.203674
Time                      114    1.203674
CO(GT)                    114    1.203674
PT08.S1(CO)               114    1.203674
NMHC(GT)                  114    1.203674
C6H6(GT)                  114    1.203674
PT08.S2(NMHC)             114    1.203674
NOx(GT)                   114    1.203674
PT08.S3(NOx)              114    1.203674
NO2(GT)                   114    1.203674
PT08.S4(NO2)              114    1.203674
PT08.S5(O3)               114    1.203674
T                         114    1.203674
RH                        114    1.203674
AH                        114    1.203674
Unnamed: 15              9471  100.000000
Unnamed: 16              9471  100.000000
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me
```

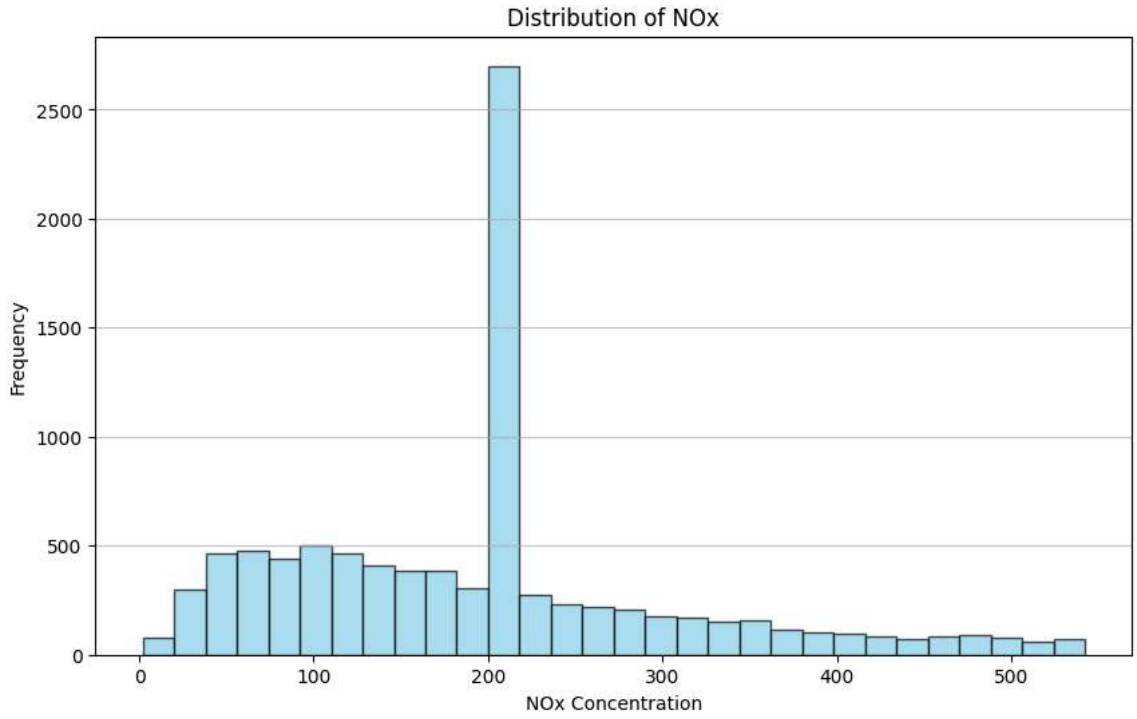
```
  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:55: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  df[col].fillna(df[col].median(), inplace=True)
<ipython-input-41-049c444d7bb6>:60: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assign
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me
```

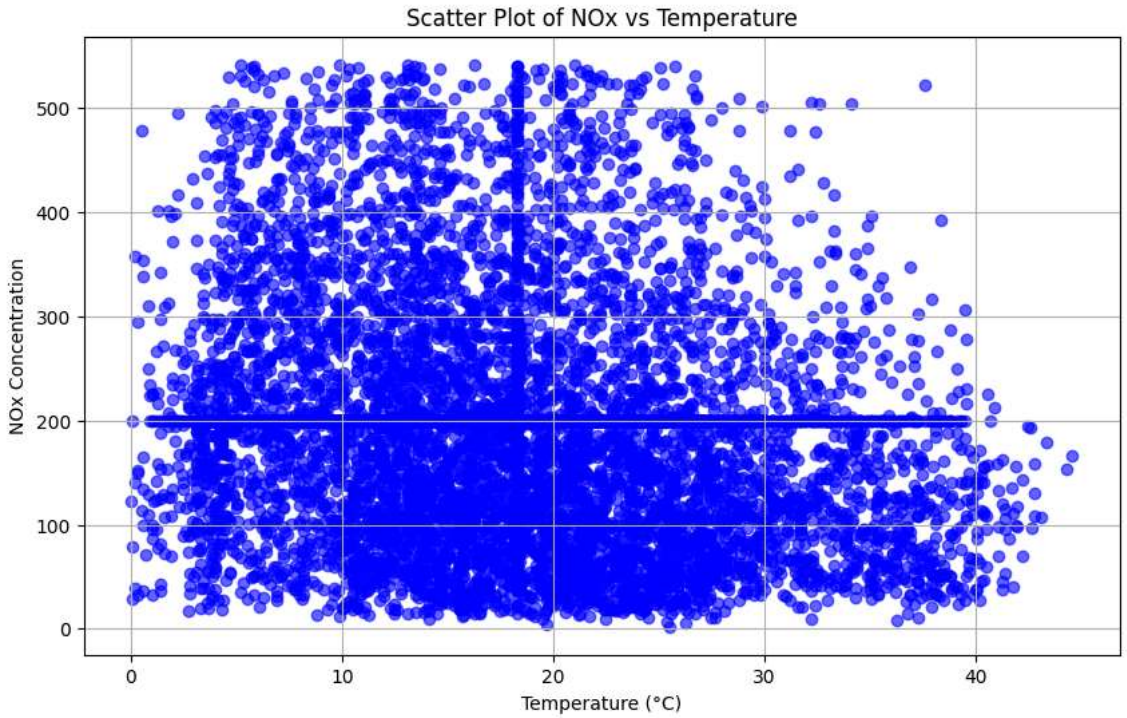
```
  df[col].fillna(df[col].mode()[0], inplace=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:695: RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmin(X, axis=axis))
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py:712: RuntimeWarning: All-NaN slice encountered
  return xp.asarray(numpy.nanmax(X, axis=axis))

Final Preprocessed Dataset Information:
<class 'pandas.core.frame.DataFrame'>
Index: 9358 entries, 0 to 9357
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   CO(GT)         9358 non-null   float64
 1   PT08.S1(CO)    9358 non-null   float64
 2   NMHC(GT)       9358 non-null   float64
 3   C6H6(GT)       9358 non-null   float64
 4   PT08.S2(NMHC)  9358 non-null   float64
 5   NOx(GT)        9358 non-null   float64
 6   PT08.S3(NOx)   9358 non-null   float64
 7   NO2(GT)        9358 non-null   float64
 8   PT08.S4(NO2)   9358 non-null   float64
 9   PT08.S5(O3)    9358 non-null   float64
 10  T              9358 non-null   float64
 11  RH             9358 non-null   float64
 12  AH             9358 non-null   float64
 13  Unnamed: 15    0 non-null      float64
 14  Unnamed: 16    0 non-null      float64
 15  Date           9358 non-null   object
 16  Time           9358 non-null   object
dtypes: float64(15), object(2)
memory usage: 1.3+ MB
None

Descriptive Statistics:
            CO(GT)  PT08.S1(CO)  NMHC(GT)     C6H6(GT)  PT08.S2(NMHC)  \
count  9358.000000  9358.000000    9358.0  9358.000000    9358.000000
mean      0.238525     0.432381       0.0     0.313533       0.470845
std       0.149512     0.194302       0.0     0.210488       0.193352
min       0.000000     0.000000       0.0     0.000000       0.000000
25%       0.127907     0.287671       0.0     0.150000       0.341578
50%       0.244186     0.397260       0.0     0.283333       0.464572
75%       0.290698     0.550881       0.0     0.430000       0.600267
max       1.000000     1.000000       0.0     1.000000       1.000000

           NOx(GT)  PT08.S3(NOx)      NO2(GT)  PT08.S4(NO2)  PT08.S5(O3)  \
count  9358.000000   9358.000000  9358.000000   9358.000000  9358.000000
mean      0.354906      0.474145     0.394605      0.490604     0.412301
std       0.204401      0.195358     0.171399      0.177637     0.213645
min       0.000000      0.000000     0.000000      0.000000     0.000000
25%       0.203704      0.351852     0.262500      0.378630     0.264971
50%       0.366667      0.478261     0.371875      0.490411     0.393217
75%       0.411111      0.594002     0.559375      0.603288     0.550079
max       1.000000      1.000000     1.000000      1.000000     1.000000

                T           RH           AH  Unnamed: 15  Unnamed: 16
count  9358.000000  9358.000000  9358.000000          0.0          0.0
mean      0.410749     0.504199     0.410709          NaN          NaN
std       0.194000     0.213530     0.193432          NaN          NaN
min       0.000000     0.000000     0.000000          NaN          NaN
25%       0.269058     0.344654     0.274361          NaN          NaN
50%       0.410314     0.519497     0.405952          NaN          NaN
75%       0.540359     0.662893     0.543127          NaN          NaN
max       1.000000     1.000000     1.000000          NaN          NaN
Preprocessing Complete. Preprocessed file saved as 'preprocessed_air_quality.csv'
Number of numeric columns: 15
Number of rows: 9358
Data types: Date              object
Time              object
CO(GT)           float64
PT08.S1(CO)      float64
NMHC(GT)         float64
C6H6(GT)         float64
PT08.S2(NMHC)    float64
NOx(GT)          float64
PT08.S3(NOx)     float64
NO2(GT)          float64
PT08.S4(NO2)     float64
PT08.S5(O3)      float64
T                float64
RH               float64
AH               float64
Unnamed: 15      float64
Unnamed: 16      float64
dtype: object
```

```
Null values present: Date               0
Time                0
CO(GT)              0
PT08.S1(CO)         0
NMHC(GT)            0
C6H6(GT)            0
PT08.S2(NMHC)       0
NOx(GT)             0
PT08.S3(NOx)        0
NO2(GT)             0
PT08.S4(NO2)        0
PT08.S5(O3)         0
T                   0
RH                  0
AH                  0
Unnamed: 15      9358
Unnamed: 16      9358
dtype: int64
```

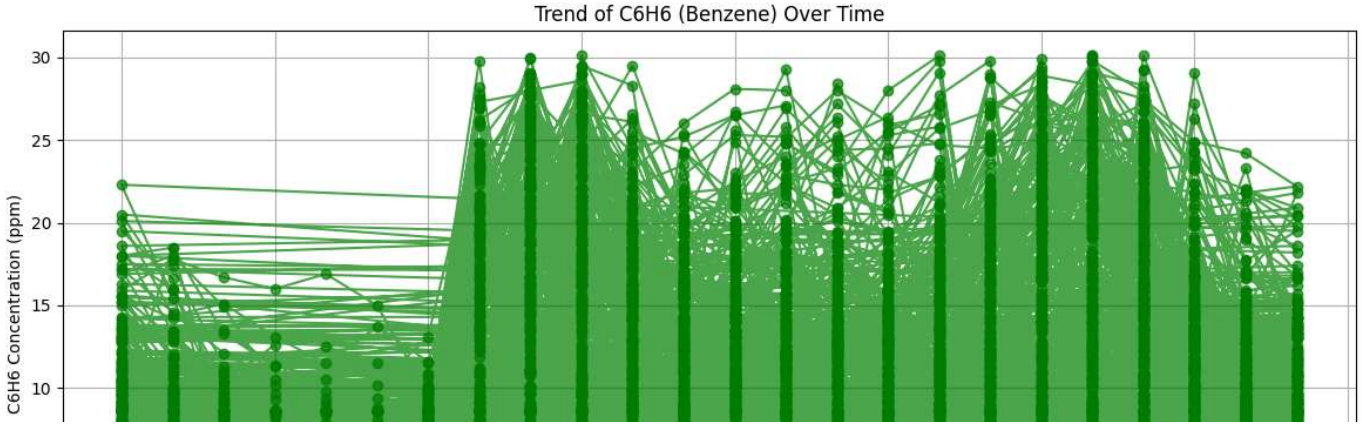


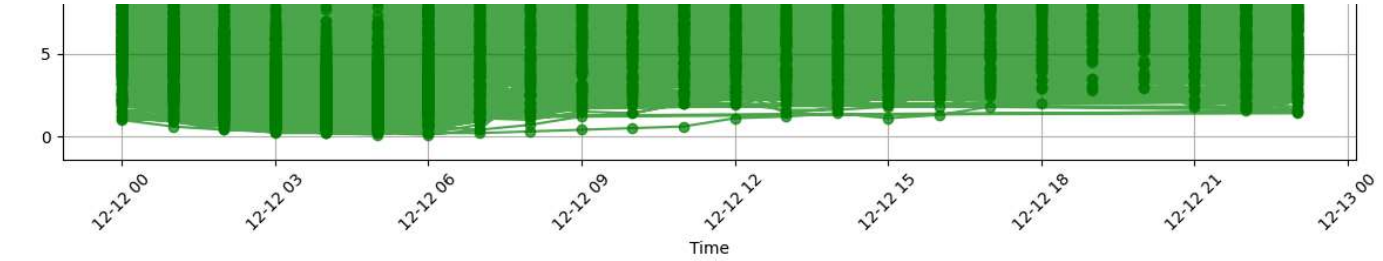


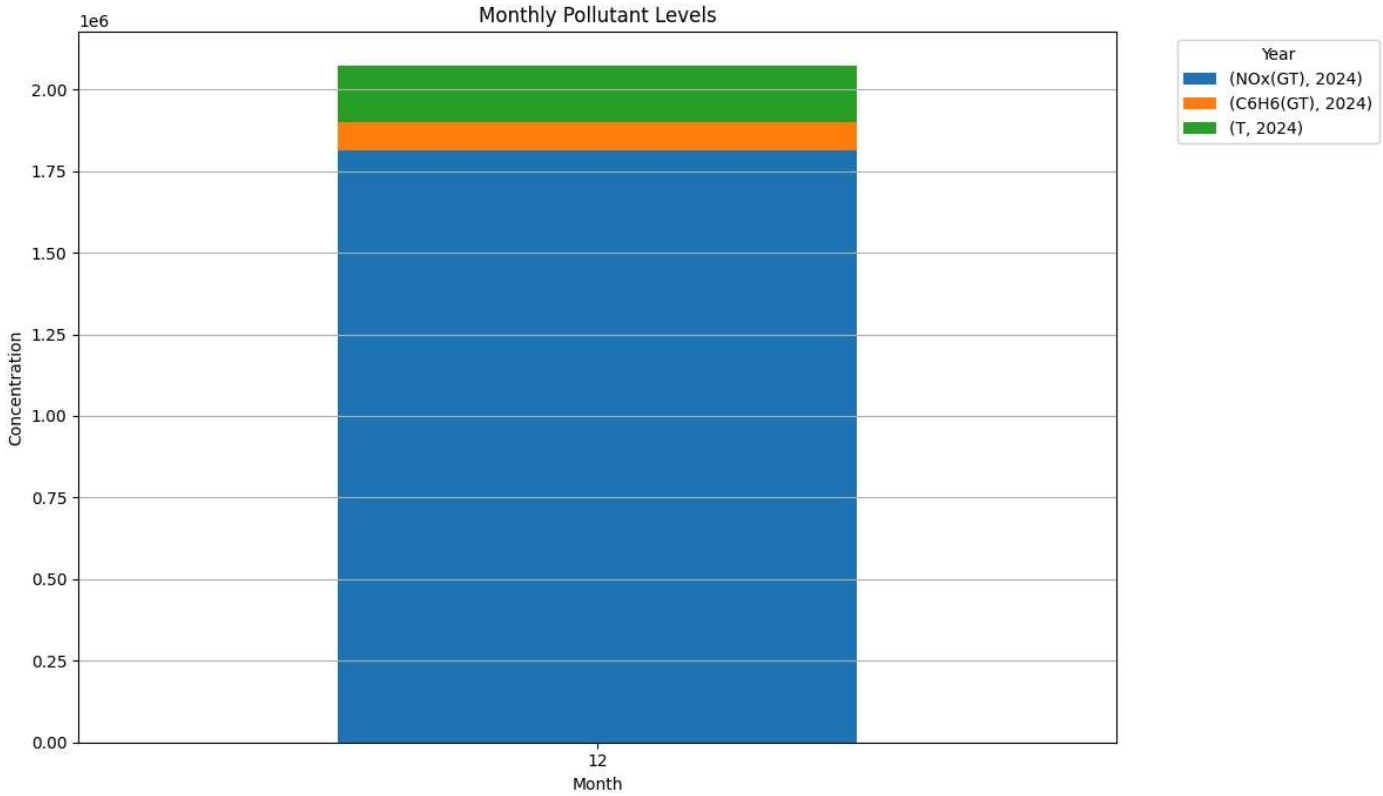


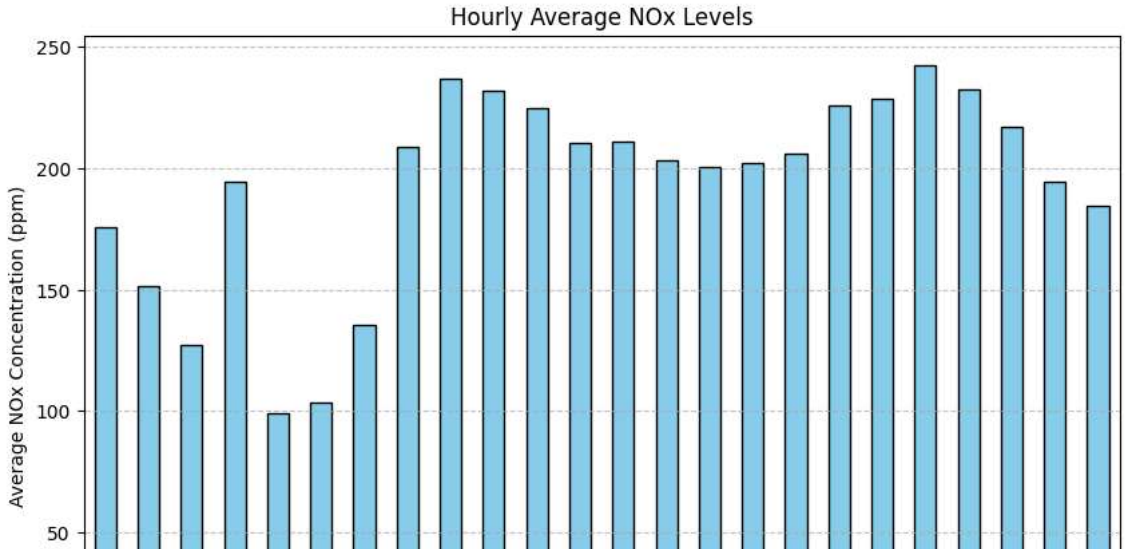



```
<ipython-input-41-049c444d7bb6>:177: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `
  df['Time'] = pd.to_datetime(df['Time'], errors='coerce')
```




5
0
12-12 00
12-12 03
12-12 06
12-12 09
12-12 12
12-12 15
12-12 18
12-12 21
12-13 00
Time


Monthly Pollutant Levels
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Concentration
12
Month
Year
(NOx(GT), 2024)
(C6H6(GT), 2024)
(T, 2024)


Hourly Average NOx Levels
250
200
150
100
50
Average NOx Concentration (ppm)