Assignment No. 4

Problem Statement: Perform the following operations using Python on the Air quality data sets a. Data cleaning b. Data transformation

Objective:

This analysis aims to enhance the quality and consistency of a 16,219-row air quality dataset by performing data cleaning and preprocessing. Key steps include handling missing values, removing duplicates, addressing outliers, and applying normalization and encoding as needed. By standardizing the data, we can ensure reliable analysis and meaningful insights into air quality trends. Visualizations will be created to reveal key patterns, helping to interpret the data effectively.

Prerequisite :

1. Basic understanding of Python programming.
2. Understanding of Data cleaning and Data Preprocessing.
3. Understanding of libraries like Pandas, NumPy, Matplotlib, and Seaborn.
4. Knowledge of libraries such as NumPy and Matplotlib for data generation and visualization

Theory :

Data Preprocessing:

Data preprocessing, also known as data preparation, involves the process of identifying and correcting errors or inconsistencies in a dataset. It prepares raw data for analysis by ensuring that it is clean, organized, and suitable for machine learning models.
Data preprocessing includes four main categories:

1. Data Cleaning:

Real-world data is often incomplete, noisy, and inconsistent, with some values potentially irrelevant or missing. Data cleaning addresses these issues by filling in missing values, smoothing out noise, detecting outliers, and resolving discrepancies. Unprocessed data can lead to confusion and inaccuracies in analysis and modeling. Therefore, applying data cleaning techniques is a crucial part of data preprocessing to ensure quality and reliability.

a) **Handling Missing Data**
Datasets often contain missing values, which can arise during data collection or due to validation rules. It's essential to address these gaps in various ways:

- **Dropping Rows/Columns**: If a row or column contains only NaN values or has over 65% missing data, it may be prudent to remove it.
- **Checking for Duplicates**: Duplicate rows or columns should be eliminated, retaining only the first instance to avoid bias during machine learning processes.
- **Estimating Missing Values**: For datasets with a small percentage of missing values, interpolation methods can fill in the gaps. A common approach is to replace missing values with the mean, median, or mode of the feature.

b) **Addressing Noisy Data**
Noisy data, which lacks meaningful information, can result from poor data collection or input errors. Several techniques can be employed to manage noisy data:

- **Binning Method**: This technique smooths sorted data by dividing it into equal-sized segments and replacing values within each segment with the mean or boundary values.
- **Clustering**: This approach groups related data points into clusters, helping to identify or isolate outliers.
- **Regression**: Smoothing can be achieved by fitting the data to a regression model, which can be linear (one independent variable) or multiple (multiple independent variables).

2. Data Integration:

Data integration is the process of combining data from multiple sources into a unified data store, ensuring that the information is coherent and usable for analysis. This process often involves addressing challenges related to schema integration and entity identification.

a) **Purpose**: To unify data from various sources (databases, data cubes, flat files) into a single coherent dataset.
b) **Schema Integration**: Involves aligning different data structures and formats to ensure compatibility.
c) **Entity Identification**: Addresses the challenge of matching real-world entities across sources (if customer_id and cust_number refer to the same entity).

d) **Role of Metadata**: Utilizes metadata (data about data) to help avoid errors during the integration process.

e) **Redundancy Issues**: Can arise when attributes are derived from other tables, leading to duplicate information.

f) **Inconsistencies**: Naming conventions for attributes or dimensions may differ across sources, causing further redundancies in the integrated dataset.

3. Data Transformation:

Data transformation is the process of converting data into a format suitable for mining. This can be achieved through several methods:

1. **Normalization**: Adjusts data values to fit within a specified range, such as -1.0 to 1.0 or 0.0 to 1.0.
2. **Concept Hierarchy Generation**: Involves replacing low-level or raw data with higher-level concepts, such as generalizing categorical data (e.g., street to city) or transforming numeric values into categories (e.g., age into youth, middle-aged, or elderly).
3. **Smoothing**: Aims to eliminate noise in the data using techniques like binning, clustering, and regression.
4. **Aggregation**: Combines data to create summary statistics. For instance, daily sales can be aggregated to compute monthly or yearly totals. Feature aggregation reduces dimensionality by merging highly correlated features, such as calculating the area from height and width, thus decreasing multicollinearity.

4. Data Reduction:

Data reduction refers to techniques used to simplify and reduce the amount of data that needs to be processed during analysis. This is essential for improving storage efficiency and minimizing the costs associated with data storage and analysis, especially when dealing with large datasets.

Techniques of Data Reduction:

a) Dimensionality Reduction:

- Aims to reduce the number of features in a dataset.
- Focuses on minimizing dimensions without merely selecting a subset of features.
- Techniques include methods like Principal Component Analysis (PCA) that transform data into a lower-dimensional space.

b) Numerosity Reduction:

- Involves replacing or estimating data using smaller representations.
- Uses parametric models (e.g., regression, log-linear models) that store only essential parameters instead of full datasets.
- Employs non-parametric methods (e.g., clustering, sampling) to summarize data effectively.

c) Data Cube Aggregation:

- Involves applying aggregation operations during the construction of a data cube.
- Facilitates multi-dimensional analysis by summarizing data points.

d) Data Compression:

- Utilizes encoding techniques to reduce dataset sizes.
- Common methods include Wavelet Transform and PCA, which help preserve essential information while minimizing space.

e) Discretization and Concept Hierarchy Generation:

- Replaces raw data values with ranges or higher-level concepts to simplify analysis.
- Allows for mining data at various abstraction levels, enhancing interpretability and usability in data mining tasks.

Code & Output:

```python
import pandas as pd
# Load the dataset
df = pd.read_csv('C:/Users/dnyan/FODS Assignments/Datasets/Air_Quality.csv')
data = df.iloc[:, 4]
print(data.head())
```

```
0    ppb
1    ppb
2    ppb
3    ppb
4    ppb
Name: Measure Info, dtype: object
```

```
#Step 1: Data Cleaning
# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
Unique ID           0
Indicator ID        0
Name                0
Measure             0
Measure Info        0
Geo Type Name       0
Geo Join ID         0
Geo Place Name      0
Time Period         0
Start_Date          0
Data Value          0
Message         16218
dtype: int64
```

```
dataset_null = df.isnull()
print(dataset_null)
```

```
      Unique ID  Indicator ID   Name  Measure  Measure Info  Geo Type Name  \
0         False         False  False    False         False          False
1         False         False  False    False         False          False
2         False         False  False    False         False          False
3         False         False  False    False         False          False
4         False         False  False    False         False          False
...         ...           ...    ...      ...           ...            ...
16213     False         False  False    False         False          False
16214     False         False  False    False         False          False
16215     False         False  False    False         False          False
16216     False         False  False    False         False          False
16217     False         False  False    False         False          False
```

```
       Message
0         True
1         True
2         True
3         True
4         True
...        ...
16213     True
16214     True
16215     True
16216     True
16217     True
```

```python
#Message column have almost 100% missing vlaue so we can drop it
df = df.drop('Message', axis=1)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16218 entries, 0 to 16217
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Unique ID       16218 non-null  int64
 1   Indicator ID    16218 non-null  int64
 2   Name            16218 non-null  object
 3   Measure         16218 non-null  object
 4   Measure Info    16218 non-null  object
 5   Geo Type Name   16218 non-null  object
 6   Geo Join ID     16218 non-null  int64
 7   Geo Place Name  16218 non-null  object
 8   Time Period     16218 non-null  object
 9   Start_Date      16218 non-null  object
 10  Data Value      16218 non-null  float64
dtypes: float64(1), int64(3), object(7)
memory usage: 1.4+ MB
```

```python
#converting Data into Numerical Format
# Convert 'Start_Date' to datetime format
df['Start_Date'] = pd.to_datetime(df['Start_Date'], errors='coerce')

# For categorical columns, apply one-hot encoding
categorical_columns = ['Name', 'Measure', 'Measure Info', 'Geo Type Name', 'Geo Place Name', 'Time Period']

# One-Hot Encoding
df_encoded = pd.get_dummies(df, columns=categorical_columns, drop_first=True)

# Check the first few rows of the transformed DataFrame
print(df_encoded.head())

# Optionally, check the data types after conversion
print(df_encoded.dtypes)
```

```
   Unique ID  Indicator ID  Geo Join ID Start_Date  Data Value  \
0     172653           375          203 2010-12-01       25.30
1     172585           375          203 2008-12-01       26.93
2     336637           375          204 2015-01-01       19.09
3     336622           375          103 2015-01-01       19.76
4     172582           375          104 2008-12-01       22.83
```

```
[5 rows x 202 columns]
Unique ID                                    int64
Indicator ID                                 int64
Geo Join ID                                  int64
Start_Date                          datetime64[ns]
Data Value                                 float64
                                         ...
Time Period_Winter 2016-17                    bool
Time Period_Winter 2017-18                    bool
Time Period_Winter 2018-19                    bool
Time Period_Winter 2019-20                    bool
Time Period_Winter 2020-21                    bool
Length: 202, dtype: object
```

```python
# Check for duplicates again
duplicates = df_encoded.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")
```

```
Number of duplicate rows: 0
```

```python
# Check for missing values again
missing_values = df_encoded.isnull().sum()
print("Missing values in each column:")
print(missing_values[missing_values > 0])
```

```
Missing values in each column:
Series([], dtype: int64)
```

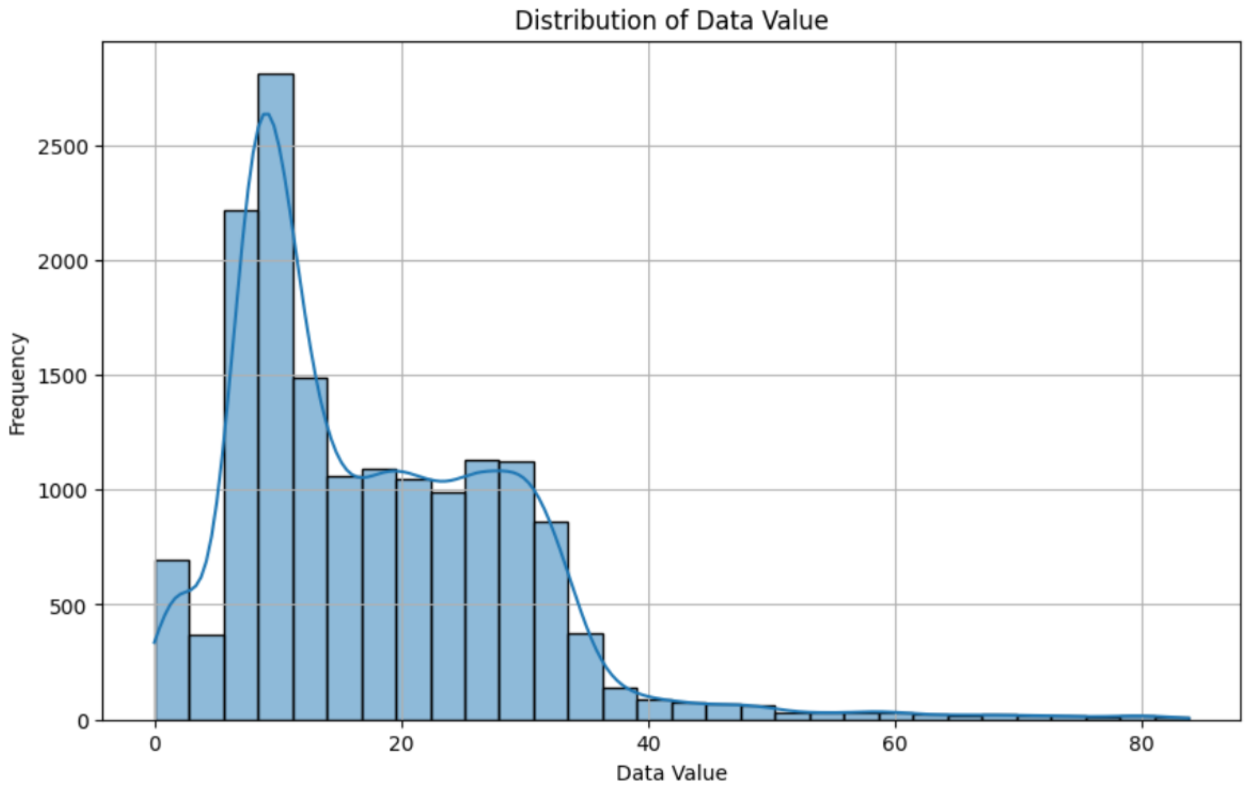
```python
# Outlier detection (example using Z-score)
from scipy import stats

# You can choose to check for outliers in 'Data Value'
z_scores = stats.zscore(df_encoded['Data Value'])
abs_z_scores = abs(z_scores)
outliers = (abs_z_scores > 3).sum()  # threshold can be adjusted

print(f"Number of outliers in 'Data Value': {outliers}")

# Optionally, you can remove outliers
df_encoded = df_encoded[(abs_z_scores <= 3)]
print(f"New shape after removing outliers: {df_encoded.shape}")
```

```
Number of outliers in 'Data Value': 274
New shape after removing outliers: (15944, 202)
```

```python
#Re-verify after handling the missing values
# Check for missing values
missing_values = df_encoded.isnull().sum()
print("Missing values in each column:")
print(missing_values[missing_values > 0])

# Summary statistics
print("Summary statistics of the DataFrame:")
print(df_encoded.describe())
```

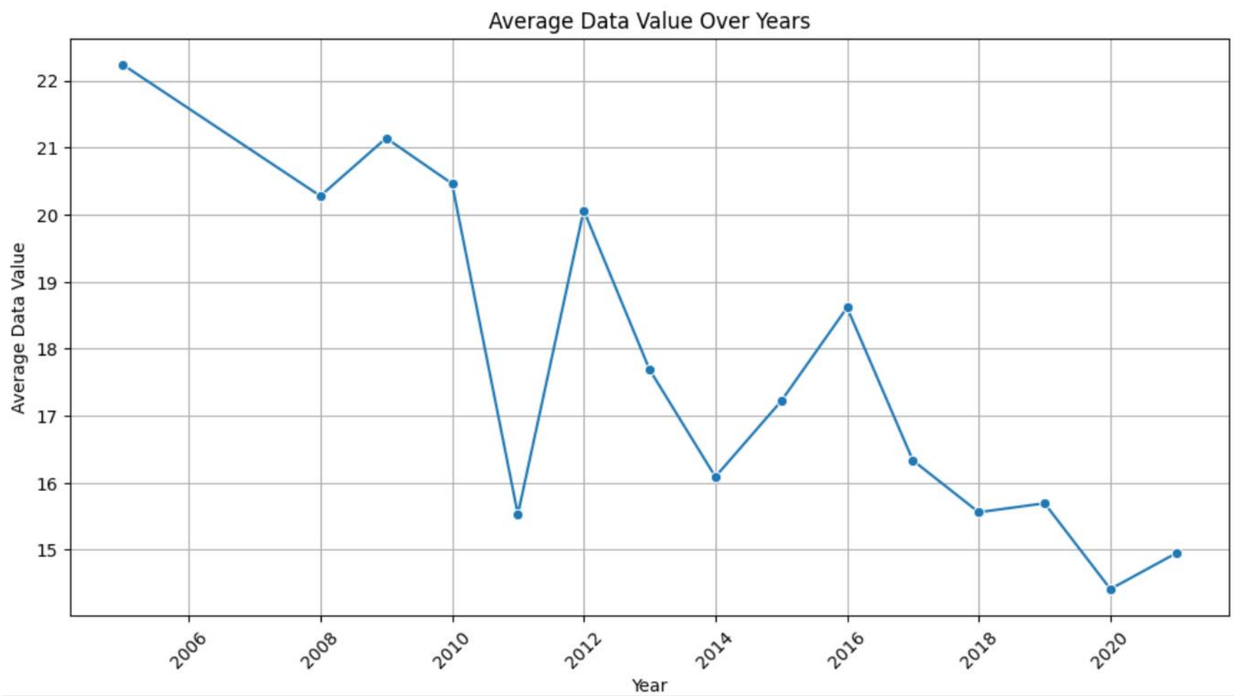
```
Missing values in each column:
Series([], dtype: int64)
Summary statistics of the DataFrame:
           Unique ID  Indicator ID   Geo Join ID  \
count   15944.000000  15944.000000  1.594400e+04
mean   373678.183705    423.956222  6.201849e+05
min    121644.000000    365.000000  1.000000e+00
25%    173633.750000    365.000000  2.020000e+02
50%    325285.500000    375.000000  3.030000e+02
75%    605287.250000    386.000000  4.040000e+02
max    799868.000000    661.000000  1.051061e+08
std    215387.957752    107.881226  7.960520e+06

                          Start_Date    Data Value
count                          15944  15944.000000
mean   2014-04-07 10:34:49.914701568     17.826834
min              2005-01-01 00:00:00      0.000000
25%              2011-01-01 00:00:00      8.980000
50%              2014-06-01 00:00:00     14.820000
75%              2017-06-01 00:00:00     25.570000
max              2021-06-01 00:00:00     83.800000
std                              NaN     11.623999
```

```python
#now data is in good shape and clean so we can do data transformation
#Distribution Plot of Data Value
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.histplot(df_encoded['Data Value'], bins=30, kde=True)
plt.title('Distribution of Data Value')
plt.xlabel('Data Value')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

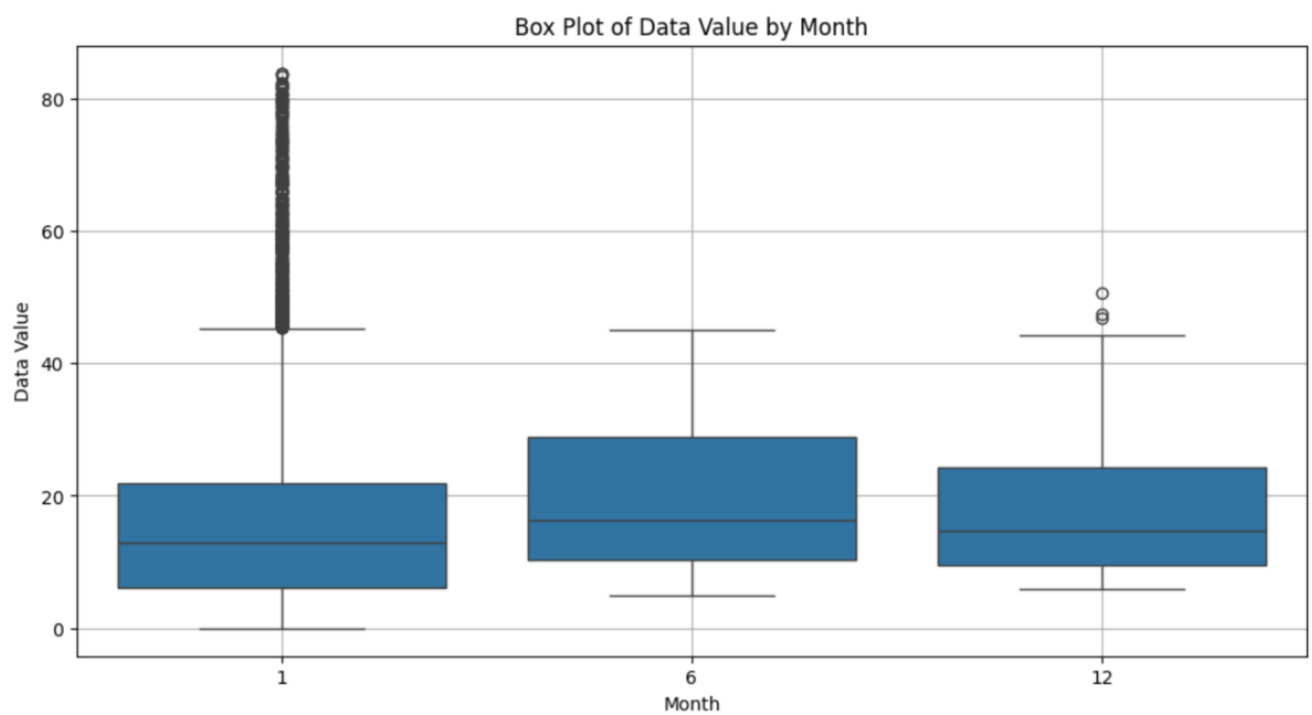



```
#feature selection
# Drop irrelevant columns
# Assuming Unique ID, Indicator ID, and Geo Join ID are not directly relevant for your analysis
df_transformed = df_encoded.drop(['Unique ID', 'Indicator ID', 'Geo Join ID'], axis=1)

# Extracting year and month from Start_Date
df_transformed['Year'] = df_transformed['Start_Date'].dt.year
df_transformed['Month'] = df_transformed['Start_Date'].dt.month

# we can Drop Start_Date if it's not needed anymore
df_transformed = df_transformed.drop('Start_Date', axis=1)
```

```
# Grouping by Year to find the average Data Value
average_data_value_per_year = df_transformed.groupby('Year')['Data Value'].mean().reset_index()

plt.figure(figsize=(12, 6))
sns.lineplot(data=average_data_value_per_year, x='Year', y='Data Value', marker='o')
plt.title('Average Data Value Over Years')
plt.xlabel('Year')
plt.ylabel('Average Data Value')
plt.xticks(rotation=45)
plt.grid()
plt.show()
```




```python
plt.figure(figsize=(12, 6))
sns.boxplot(x='Month', y='Data Value', data=df_transformed)
plt.title('Box Plot of Data Value by Month')
plt.xlabel('Month')
plt.ylabel('Data Value')
plt.grid()
plt.show()
```



Another case study for cleaning handling missing values:

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('C:/Users/ML/Desktop/123B2F144 DS/Datasets/Air qua state wise/city_day.csv')

# Assuming 'pm2.5' is the 5th column, assign it to a new variable
data = df.iloc[:, 4]

# Display a few rows of PM2.5 data
print(data.head())
```

```
0     0.92
1     0.97
2    17.40
3     1.70
4    22.10
Name: NO, dtype: float64
```

```python
dataset_null = df.isnull()
print(dataset_null)
```

```
        City   Date  PM2.5   PM10     NO    NO2    NOx    NH3     CO    SO2  \
0      False  False   True   True  False  False  False   True  False  False
1      False  False   True   True  False  False  False   True  False  False
2      False  False   True   True  False  False  False   True  False  False
3      False  False   True   True  False  False  False   True  False  False
4      False  False   True   True  False  False  False   True  False  False
...      ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
29526  False  False  False  False  False  False  False  False  False  False
29527  False  False  False  False  False  False  False  False  False  False
29528  False  False  False  False  False  False  False  False  False  False
29529  False  False  False  False  False  False  False  False  False  False
29530  False  False  False  False  False  False  False  False  False  False

          O3  Benzene  Toluene  Xylene    AQI  AQI_Bucket
0      False    False    False   False   True        True
1      False    False    False   False   True        True
2      False    False    False   False   True        True
3      False    False    False   False   True        True
4      False    False    False   False   True        True
```

```python
print(df.isnull().sum())
```

```
City              0
Date              0
PM2.5          4598
PM10          11140
NO             3582
NO2            3585
NOx            4185
NH3           10328
CO             2059
SO2            3854
O3             4022
Benzene        5623
Toluene        8041
Xylene        18109
AQI            4681
AQI_Bucket     4681
dtype: int64
```

```python
percent_missing_dataset = df.isnull().mean()*100
print(percent_missing_dataset)
```

```
City           0.000000
Date           0.000000
PM2.5         15.570079
PM10          37.723071
NO            12.129626
NO2           12.139785
NOx           14.171549
NH3           34.973418
CO             6.972334
SO2           13.050692
O3            13.619586
Benzene       19.041008
Toluene       27.229014
Xylene        61.322001
AQI           15.851139
AQI_Bucket    15.851139
dtype: float64
```

```python
#  function to fill in missing values using median
def data_imputation(data, column_grouping, column_selected):
    # Parameter meaning
    # data => The name of the dataframe to be processed
    # column_grouping => The column used to group values and take the median
    # column_selected => The column in which we will fill its NaN values

    # Get unique category groups
    group = data[column_grouping].unique()

    # Loop through each value in the group category
    for value in group:
        # get median
        median = data.loc[(data[column_grouping]==value) & ~(data[column_selected].isna()), column_selected].median()

        # change missing value
        data.loc[(data[column_grouping]==value) & (data[column_selected].isna()), column_selected] = median

    # Return the dataframe after filling the missing values
    return data
```

```python
# apply the function to 'PM2.5' column
df = data_imputation(data=df, column_grouping='City', column_selected='PM2.5')
```

```python
# apply the function to 'Xylene' column
df = data_imputation(data=df, column_grouping='City', column_selected='Xylene')
```

```python
# apply the function to 'PM2.5' column
df = data_imputation(data=df, column_grouping='City', column_selected='PM10')
```

```python
# apply the function to 'NO' column
df = data_imputation(data=df, column_grouping='City', column_selected='NO')
```

```python
# apply the function to 'NO2' column
df = data_imputation(data=df, column_grouping='City', column_selected='NO2')
```

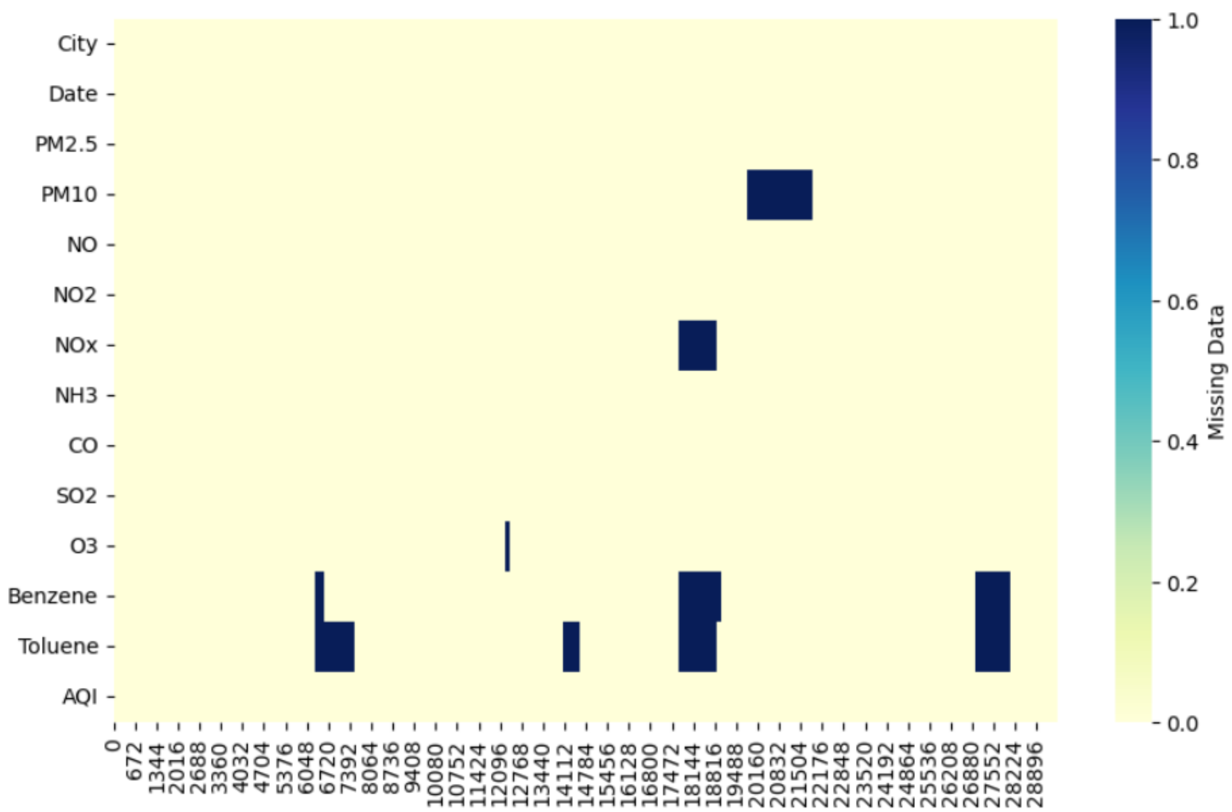
```python
# apply the function to 'NO2' column
df = data_imputation(data=df, column_grouping='City', column_selected='O3')
```

```python
n = df.isna()
missing_counts = n.sum()
missing_per = missing_counts / len(df)

print(missing_per)
```

```
City        0.000000
Date        0.000000
PM2.5       0.000000
PM10        0.068030
NO          0.000000
NO2         0.000000
NOx         0.039586
NH3         0.000000
CO          0.000000
SO2         0.000000
O3          0.005486
Benzene     0.092513
Toluene     0.135790
Xylene      0.441807
AQI         0.000000
dtype: float64
```

```python
# Visualizing Missing Data using a seaborn heatmap.
plt.figure(figsize=(10,6))
sns.heatmap(df.isna().transpose(),
            cmap="YlGnBu",
            cbar_kws={'label': 'Missing Data'})
```

```
<Axes: >
```



References :

https://medium.easyread.co/basics-of-data-preprocessing-71c314bc7188
https://medium.com/almabetter/data-preprocessing-techniques-6ea145684812
https://medium.com/@yogeshojha/data-preprocessing-75485c7188c4
https://medium.com/womenintechnology/data-preprocessing-steps-for-machine-learning-in-phyton-part-1-18009c6f1153
Github- https://github.com/dnyaneshwardhere/FODS

Conclusion :

The dataset has been cleaned by removing outliers and ensuring no missing values. Categorical features were converted to numerical formats, and temporal features were extracted. The average Data Value was analyzed over the years, making the data ready for further analysis.