AQI_Data_Cleaning

October 19, 2024

#Importing Necessary Libraries and Loading Data

[1]:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
```

[2]:

```
# Load the Data
file_path = '/content/air_pollution_data.csv'
df = pd.read_csv(file_path)
```

#Data Exploration

[3]:

```
# Display first few rows
print("Initial Data Snapshot:")
print(df.head())
```

```
Initial Data Snapshot:
        city        date  aqi       co     no    no2      o3    so2   pm2_5  \
0  Ahmedabad  30-11-2020    5   520.71   2.38  16.28  130.18  47.68   65.96
1  Ahmedabad  01-12-2020    5  1682.28   7.71  54.84    0.73  21.70  120.95
2  Ahmedabad  02-12-2020    5  1815.80  16.54  49.35    0.17  23.84  133.47
3  Ahmedabad  03-12-2020    5  2296.45  41.57  40.10    0.00  35.76  150.37
4  Ahmedabad  04-12-2020    5  2189.64  23.92  58.95    0.02  28.13  160.79

     pm10    nh3
0   72.13   8.36
1  154.53  27.36
2  172.63  28.12
3  202.15  36.48
4  205.80  40.53
```

[5]:

```
# Initial Data Exploration
print("\nData Information:")
print(df.info())
```

```
Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23504 entries, 0 to 23503
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   city    23504 non-null  object
 1   date    23504 non-null  object
 2   aqi     23504 non-null  int64
 3   co      23504 non-null  float64
 4   no      23504 non-null  float64
 5   no2     23504 non-null  float64
 6   o3      23504 non-null  float64
 7   so2     23504 non-null  float64
 8   pm2_5   23504 non-null  float64
 9   pm10    23504 non-null  float64
 10  nh3     23504 non-null  float64
dtypes: float64(8), int64(1), object(2)
memory usage: 2.0+ MB
None
```

[6]:

```
print("\nStatistical Summary:")
print(df.describe())
```

```
Statistical Summary:
                aqi            co           no           no2            o3  \
count  23504.000000  23504.000000  23504.00000  23504.000000  23504.000000
mean       3.920354   1113.224543      6.00554     25.044104     35.059777
std        1.415490   1401.770372     24.50272     25.839242     31.901760
min        1.000000    173.570000      0.00000      0.310000      0.000000
25%        3.000000    447.270000      0.00000      8.740000      7.870000
50%        5.000000    700.950000      0.00000     16.450000     28.250000
75%        5.000000   1188.280000      0.27000     32.220000     54.360000
max        5.000000  23071.290000    457.76000    331.760000    406.270000

                so2         pm2_5          pm10           nh3
count  23504.000000  23504.000000  23504.000000  23504.000000
mean      15.971449     98.598310    121.848091     12.060212
std       23.943464    135.572391    160.429589     17.544759
min        0.190000      0.500000      0.580000      0.000000
25%        4.470000     24.677500     32.277500      2.340000
50%        7.990000     58.860000     75.775000      6.520000
75%       16.450000    117.605000    147.642500     15.830000
max      442.510000   2203.550000   2429.130000    352.620000
```

#Data Cleaning

#1) Handling Missing Values Using KNNImputer

[7]:
```python
# List of pollutant columns where we want to treat 0 as missing
pollutant_cols = ['co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3']
```

[9]:
```python
# Print the number of 0s in each column before cleaning
print("\nNumber of 0s in each column before cleaning:")
for col in pollutant_cols:
    num_zero = (df[col] == 0).sum()
    print(f"{col}: {num_zero}")
```

```
Number of 0s in each column before cleaning:
co: 0
no: 12740
no2: 0
o3: 1551
so2: 0
pm2_5: 0
pm10: 0
nh3: 223
```

[10]:
```python
# Initialize KNNImputer (0 values will be treated as missing automatically by setting missing_values=0)
imputer = KNNImputer(n_neighbors=5, missing_values=0)
```

[11]:
```python
# Apply imputer on the pollutant columns
df_imputed = imputer.fit_transform(df[pollutant_cols])

# Replace the original columns with the imputed values
df[pollutant_cols] = df_imputed
```

[12]:
```python
# Print the number of 0s in each column after cleaning
print("\nNumber of 0s in each column after cleaning:")
for col in pollutant_cols:
    num_zero = (df[col] == 0).sum()
    print(f"{col}: {num_zero}")
```

```
Number of 0s in each column after cleaning:
co: 0
no: 0
no2: 0
o3: 0
so2: 0
pm2_5: 0
pm10: 0
nh3: 0
```

#Data Transformation for Dashboard

[13]:
```
# Convert 'date' column to datetime format and extract 'year' and 'month'
df['date'] = pd.to_datetime(df['date'], format='%d-%m-%Y')
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
```

[14]:
```
# Check for duplicate rows
duplicates = df.duplicated().sum()
if duplicates:
    print(f"Found {duplicates} duplicate rows. Dropping them.")
    df = df.drop_duplicates()
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

#Visualization with Matplotlib and Seaborn

[16]:
```
# City-wise Data Count
city_counts = df['city'].value_counts()

# Convert city counts to a DataFrame for a cleaner table display
city_counts_df = city_counts.reset_index()
city_counts_df.columns = ['City', 'Data Count']

# Print the city-wise data count as a table
print("\nCity-wise Data Count:")
print(city_counts_df)
```

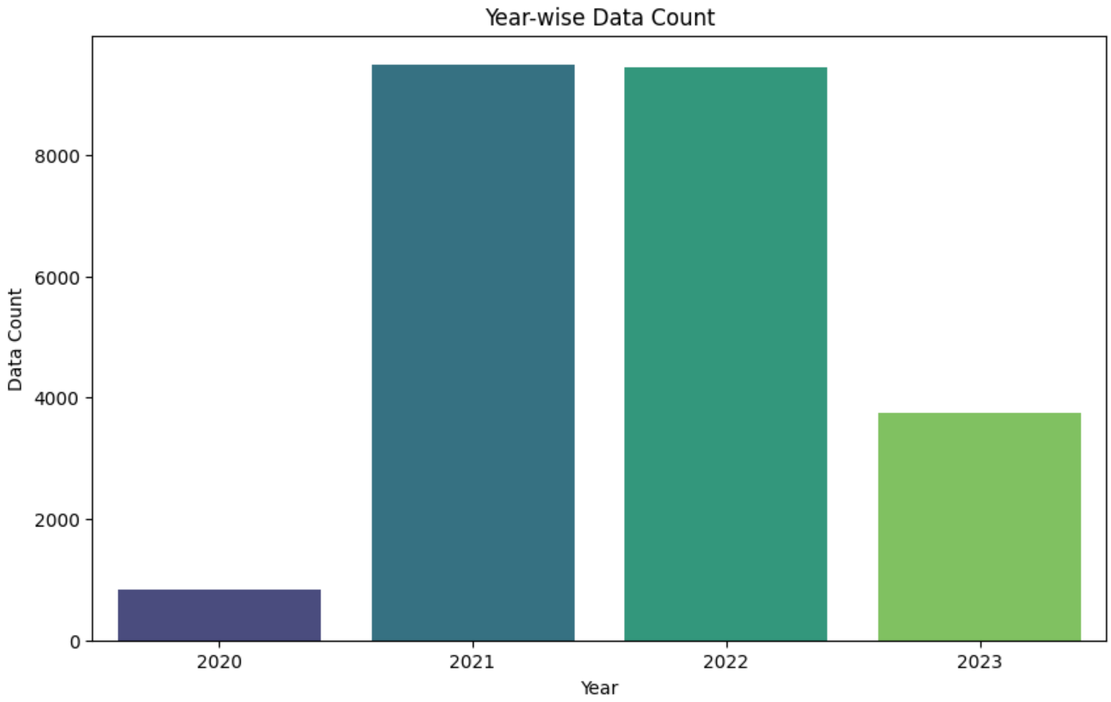
```

City-wise Data Count:
                  City  Data Count
0            Ahmedabad         904
1               Aizawl         904
2   Thiruvananthapuram         904
3              Talcher         904
4             Shillong         904
5                Patna         904
6               Mumbai         904
7              Lucknow         904
8              Kolkata         904
9                Kochi         904
10          Jorapokhar         904
11              Jaipur         904
12           Hyderabad         904
13            Guwahati         904
14            Gurugram         904
15           Ernakulam         904
```

```
16             Delhi         904
17        Coimbatore         904
18           Chennai         904
19        Chandigarh         904
20       Brajrajnagar        904
21            Bhopal         904
22         Bengaluru         904
23          Amritsar         904
24         Amaravati         904
25      Visakhapatnam        904
```

[17]:

```python
# Year-wise Data Count
year_counts = df['year'].value_counts().sort_index()

# Plot Year-wise Data Count
plt.figure(figsize=(10,6))
sns.barplot(x=year_counts.index, y=year_counts.values, palette='viridis')
plt.title('Year-wise Data Count')
plt.xlabel('Year')
plt.ylabel('Data Count')
plt.show()
```

```
<ipython-input-17-4982ac4a5279>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=year_counts.index, y=year_counts.values, palette='viridis')
```



[19]:
```
# Detect and Visualize Outliers Using Boxplots
def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers
```

[21]:
```
columns_to_check = ['aqi', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10',␣
 ↪'nh3']
```

[22]:
```
# Detect outliers for each column and print the count
for col in columns_to_check:
    outliers = detect_outliers_iqr(df, col)
    print(f"Number of outliers in '{col}': {len(outliers)}")
```

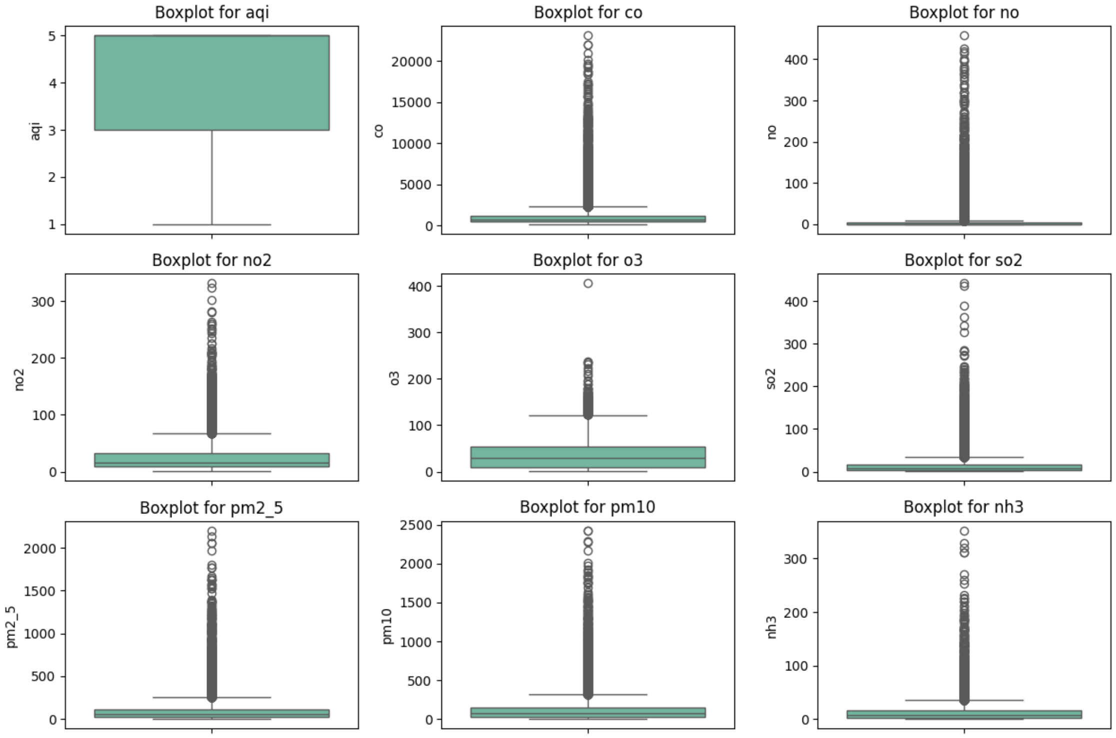
```
Number of outliers in 'aqi': 0
Number of outliers in 'co': 2257
Number of outliers in 'no': 3558
Number of outliers in 'no2': 1515
Number of outliers in 'o3': 283
Number of outliers in 'so2': 2694
Number of outliers in 'pm2_5': 1850
Number of outliers in 'pm10': 1746
Number of outliers in 'nh3': 1394
```

[23]:

```python
# Plot Boxplots for Outliers Detection
plt.figure(figsize=(12,8))
for i, col in enumerate(columns_to_check, 1):
    plt.subplot(3, 3, i)
    sns.boxplot(y=df[col], palette="Set2")
    plt.title(f'Boxplot for {col}')
plt.tight_layout()
plt.show()
```

```
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
```

```
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
<ipython-input-23-209e5b9b6d71>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(y=df[col], palette="Set2")
```



#Why Handling Outliers May Not Be Necessary for Power BI Dashboard?

Handling outliers isn't always necessary for Power BI dashboards since their purpose is to visualize data trends, and outliers can provide valuable insights, especially in environmental data like air pollution. Removing them could result in losing important information about extreme events. Power BI can display outliers clearly, allowing users to analyze their impact without altering the dataset. Thus, it's often better to retain outliers for transparency.

[24]:

```python
# Save the cleaned data to a CSV file
cleaned_file_path = '/content/air_pollution_cleaned_data.csv'
df.to_csv(cleaned_file_path, index=False)
print(f"\nCleaned data saved to {cleaned_file_path}")
```

```
Cleaned data saved to /content/air_pollution_cleaned_data.csv
```