👋 Hi, I'm Ashish Mishra

📊 Internship in Data Science

🏢 at ShadowFox

Python Visualization Libraries: Matplotlib and Seaborn

🎨 Matplotlib

> **Matplotlib** is one of the most popular and powerful Python libraries for data visualization. It allows for creating a wide variety of static, animated, and interactive plots.

🔑 **Key Features**
✅ Highly customizable plots with detailed control
📁 Supports multiple file formats (PNG, PDF, SVG)
🛠️ Compatible with GUI toolkits (Tkinter, Qt, etc.)
📊 Ideal for both 2D and limited 3D plotting

📚 Typical Use Cases

- 📖 **Academic Research** – Create publication-quality plots.
- 📈 **Data Exploration** – Visualize trends and data distributions.
- 🧩 **Custom Visualizations** – Tailored plots for specific needs.

🌈 Seaborn

> **Seaborn** is a high-level API built on top of Matplotlib that simplifies the process of creating visually attractive and statistically meaningful graphics.

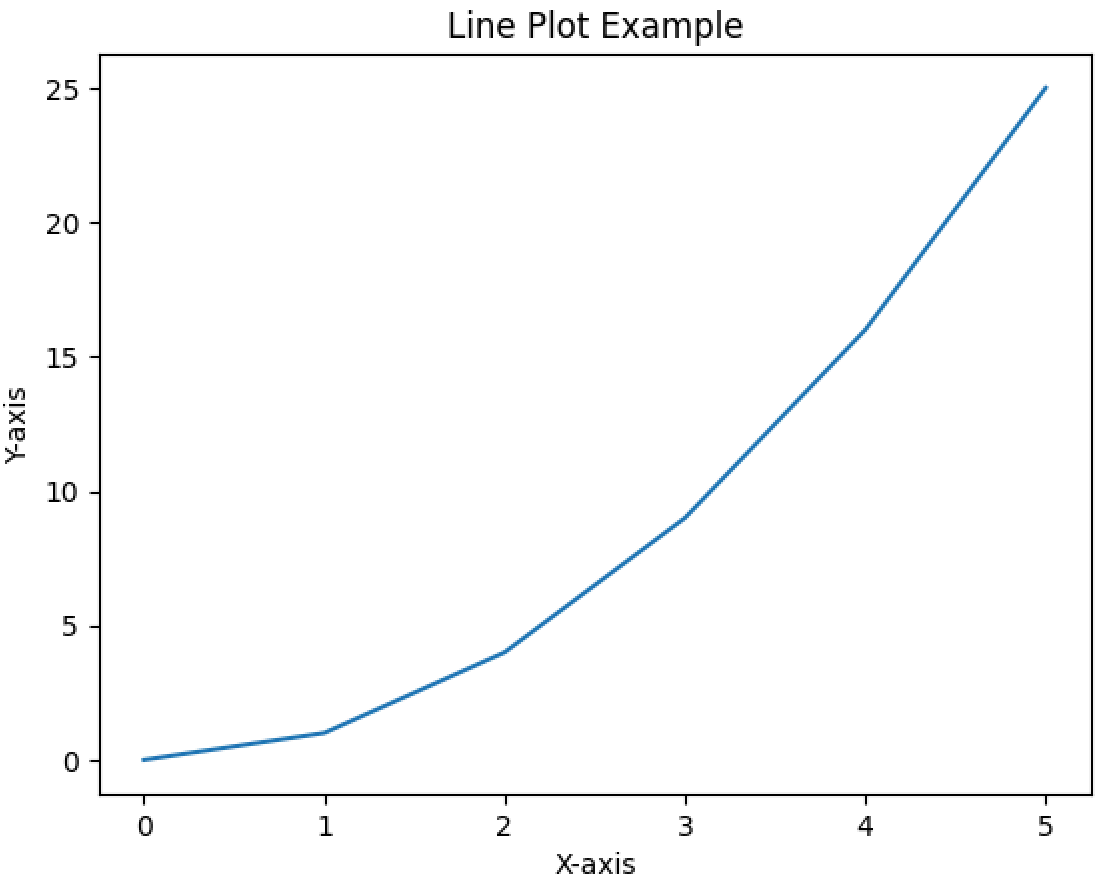
🌟 **Key Features**
🧠 Simplified syntax for quick and efficient plotting
🎨 Beautiful default themes and color palettes
📊 Built-in support for boxplots, violin plots, heatmaps
📉 Seamless integration with Pandas for EDA

📚 Typical Use Cases

- 🔍 **Exploratory Data Analysis (EDA)** – Understand your data quickly.
- 📉 **Statistical Visualization** – Relationships and distributions.
- 🖼️ **Beautiful Plots** – Create insightful and presentable charts fast.

In [1]:

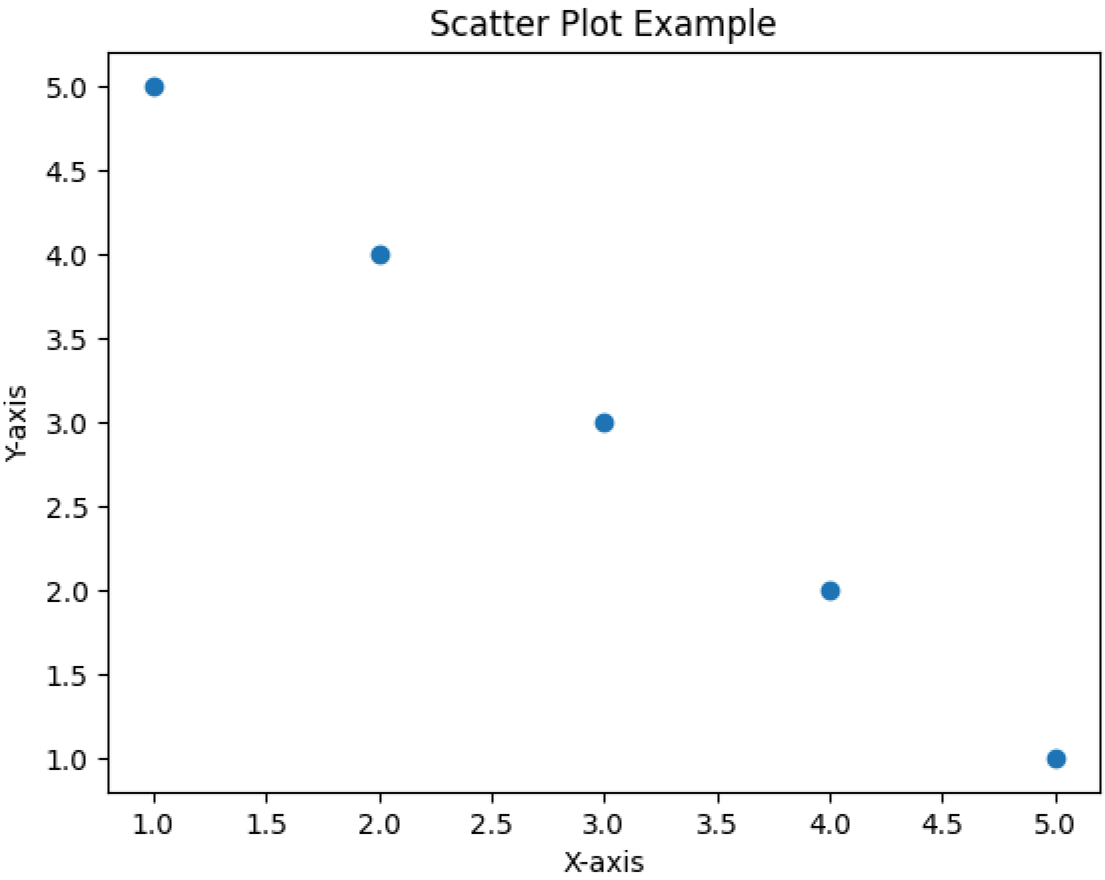
```
import matplotlib.pyplot as plt

# Sample data
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

# Create line plot
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example')
plt.show()
```



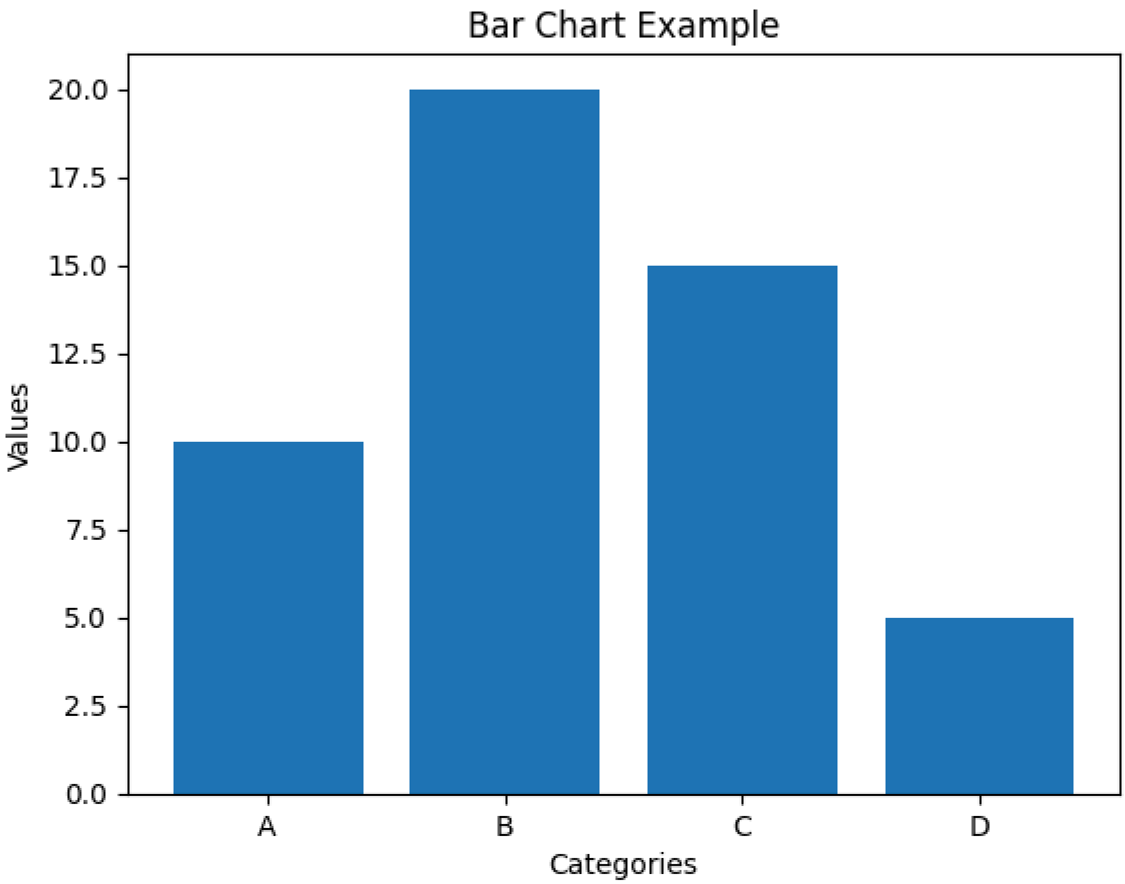
In [2]:

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [5, 4, 3, 2, 1]

# Create scatter plot
```

```
plt.scatter(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')
plt.show()
```

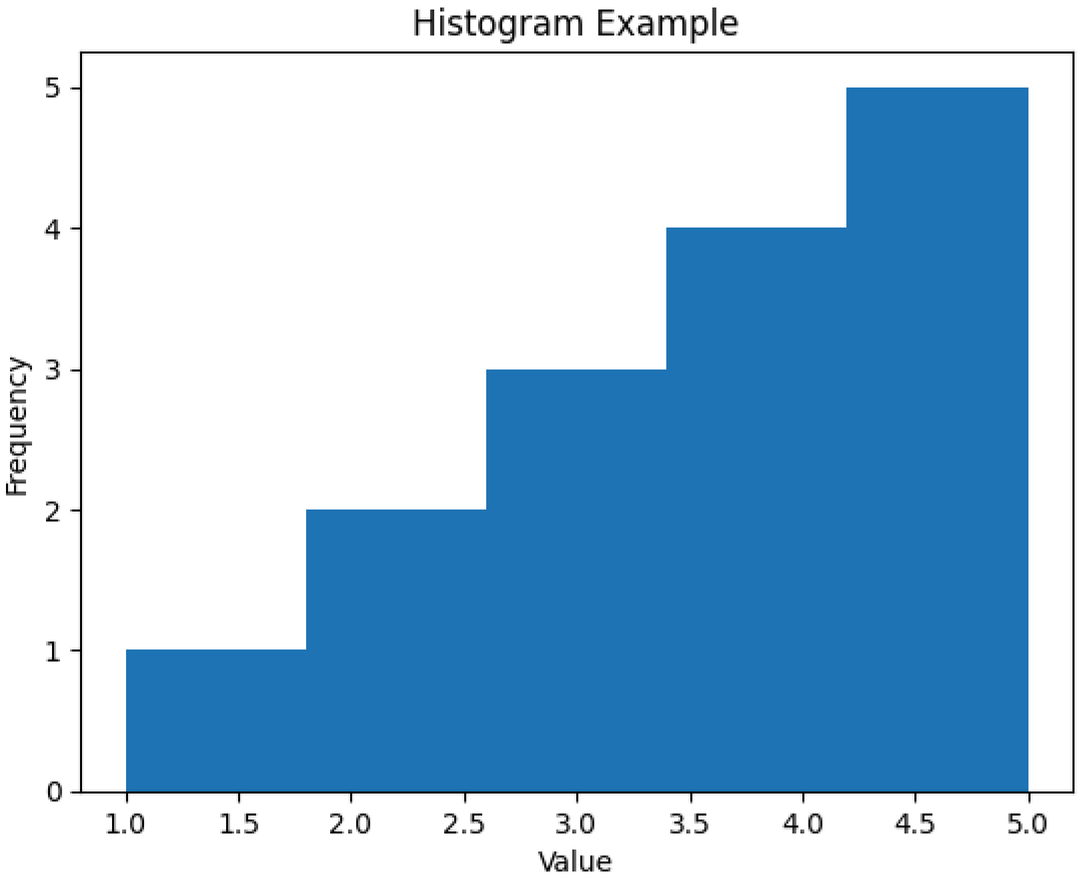


In [3]:
```
import matplotlib.pyplot as plt

# Sample data
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 5]

# Create bar chart
plt.bar(categories, values)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart Example')
plt.show()
```

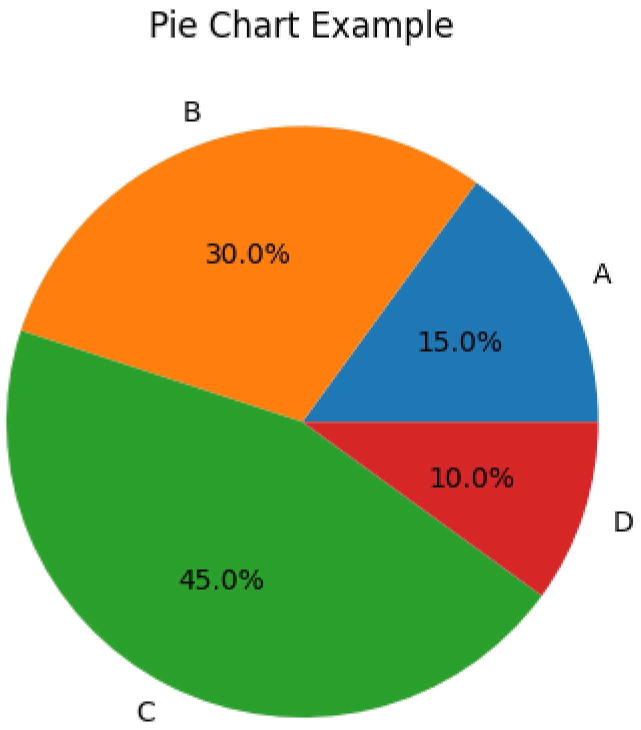


In [4]:

```
import matplotlib.pyplot as plt

# Sample data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

# Create histogram
plt.hist(data, bins=5)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()
```

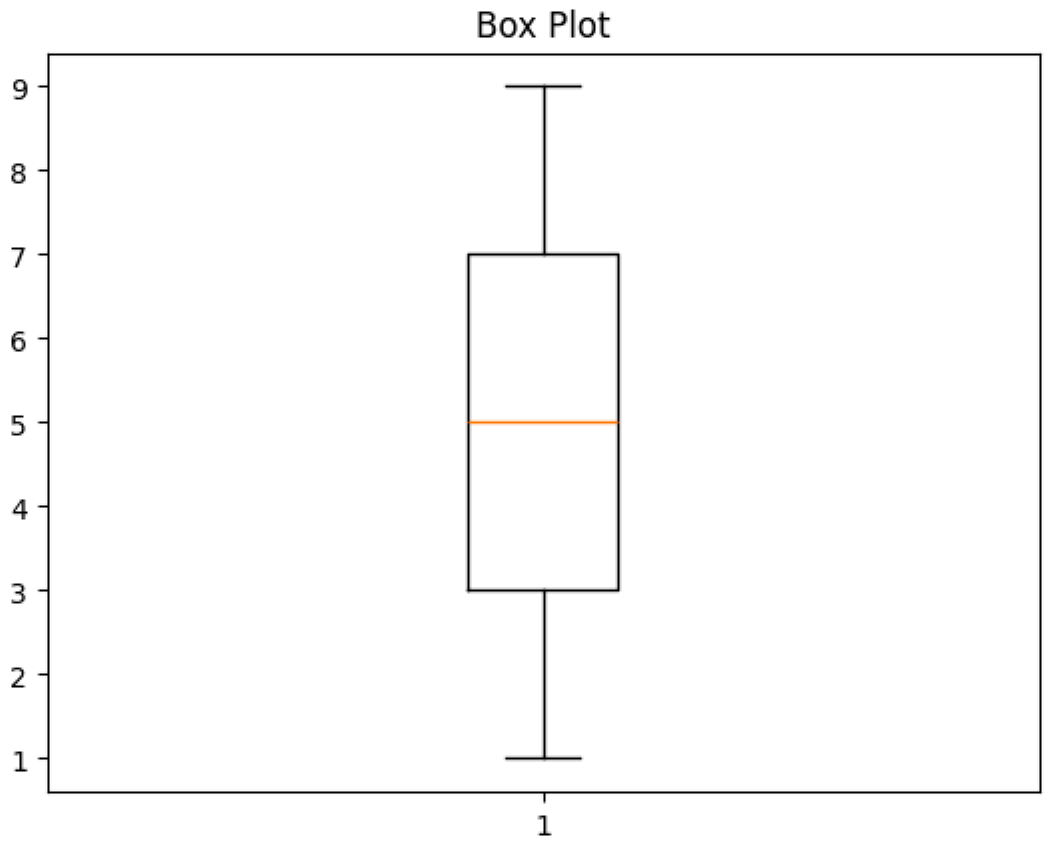


In [5]:
```
import matplotlib.pyplot as plt

# Sample data
labels = ['A', 'B', 'C', 'D']
sizes = [15, 30, 45, 10]

# Create pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title('Pie Chart Example')
plt.show()
```

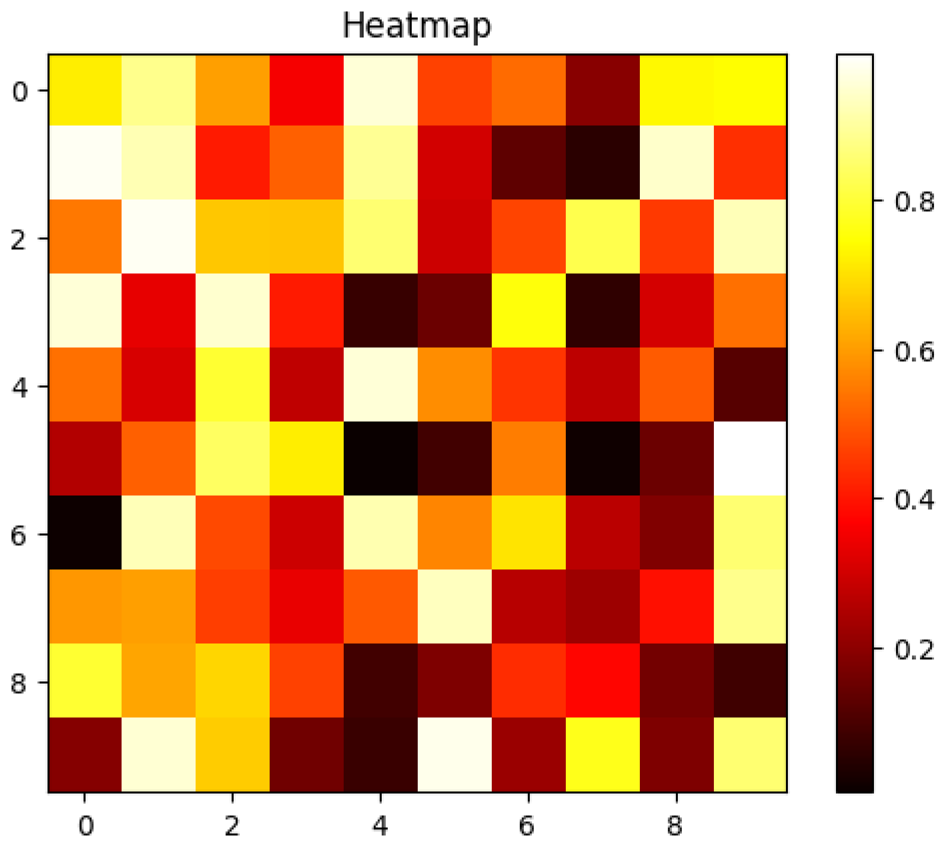


In [6]:

```
import matplotlib.pyplot as plt

data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

plt.boxplot(data)
plt.title("Box Plot")
plt.show()
```

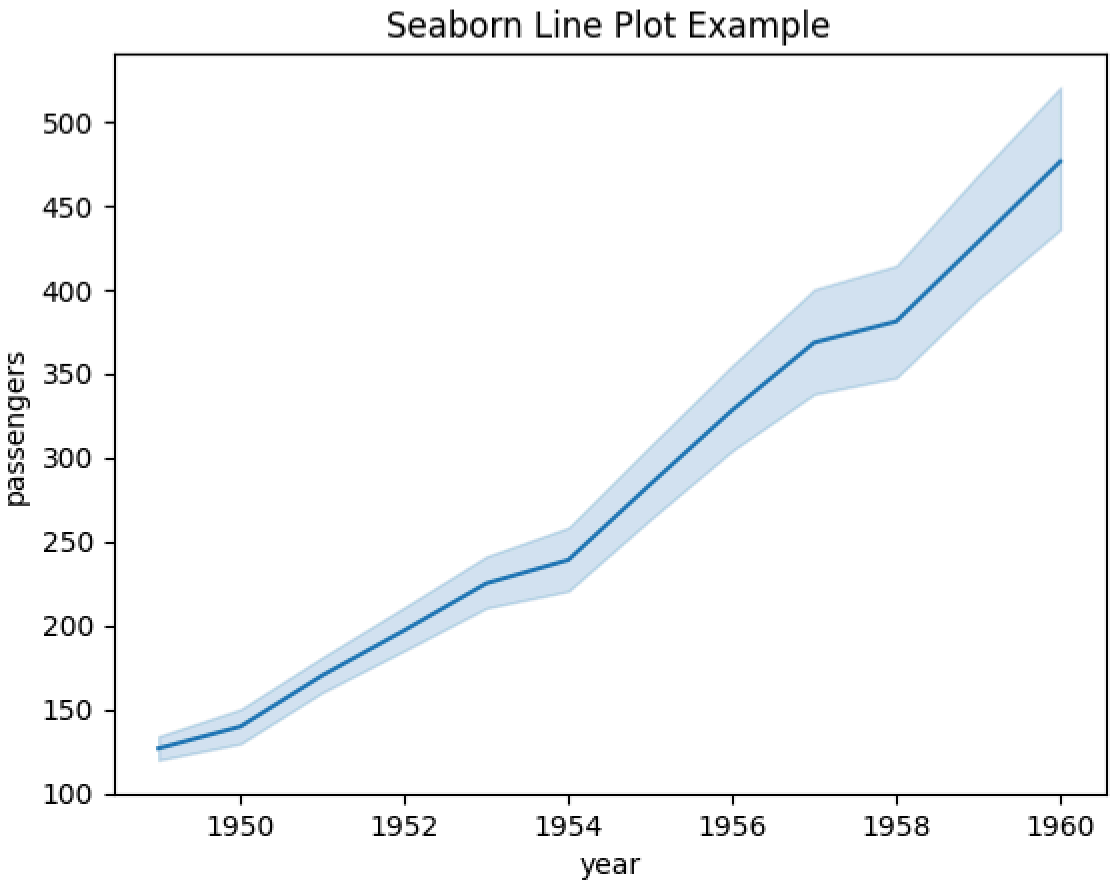


In [7]:
```
import matplotlib.pyplot as plt
import numpy as np

data = np.random.rand(10, 10)

plt.imshow(data, cmap='hot', interpolation='nearest')
plt.title("Heatmap")
plt.colorbar()
plt.show()
```



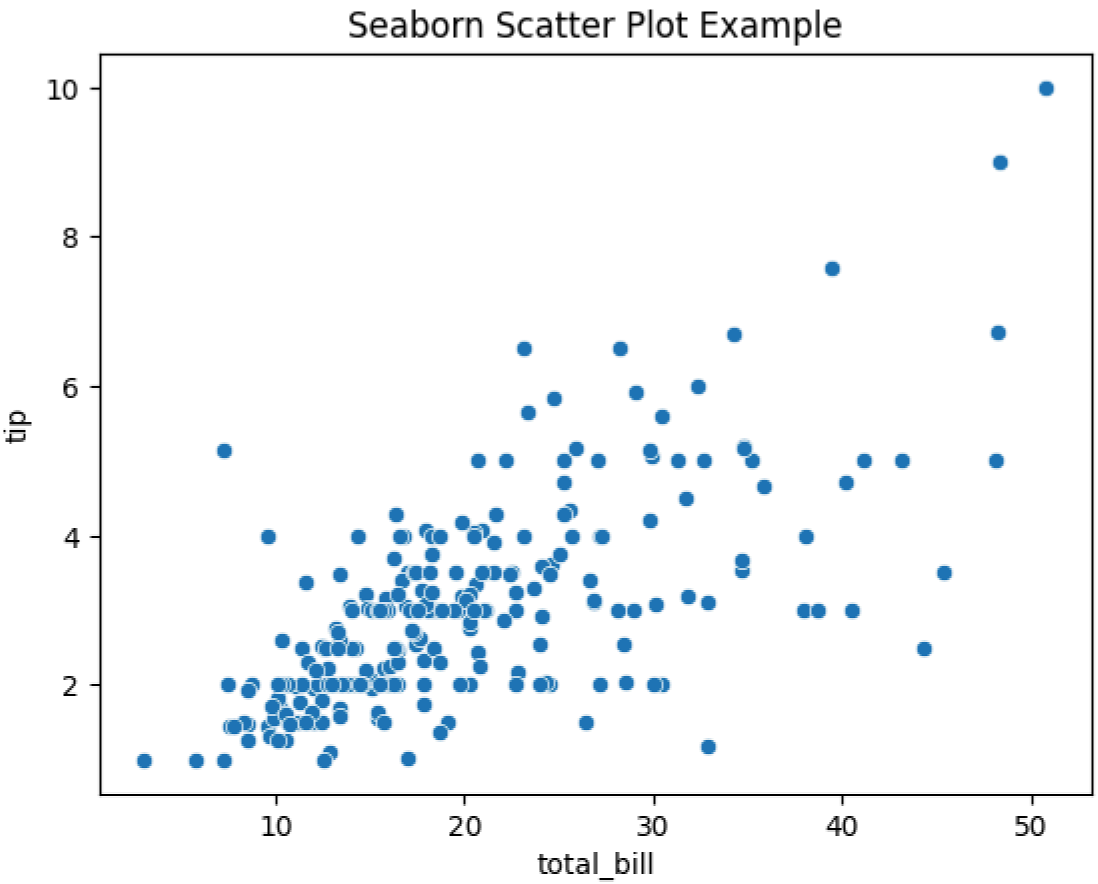
In [8]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("flights")

# Create line plot
sns.lineplot(x="year", y="passengers", data=data)
plt.title('Seaborn Line Plot Example')
plt.show()
```

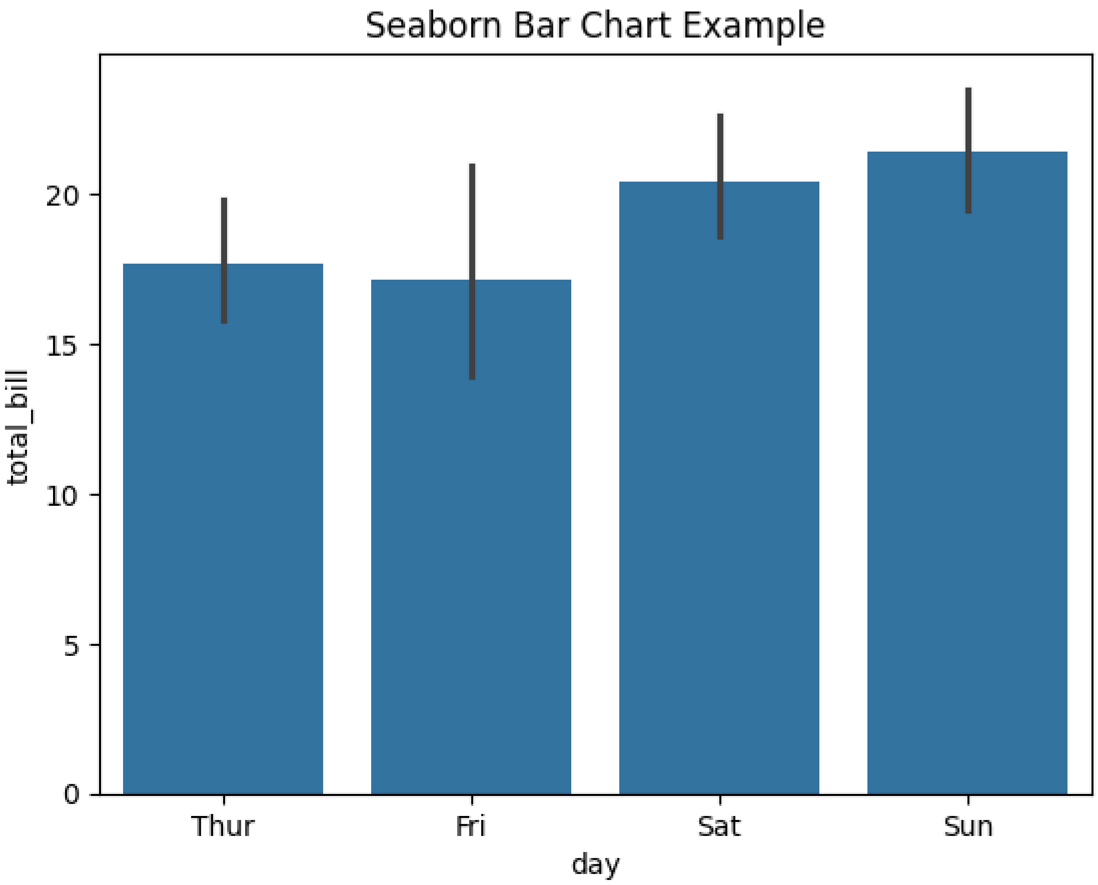


In [9]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("tips")

# Create scatter plot
sns.scatterplot(x="total_bill", y="tip", data=data)
plt.title('Seaborn Scatter Plot Example')
plt.show()
```

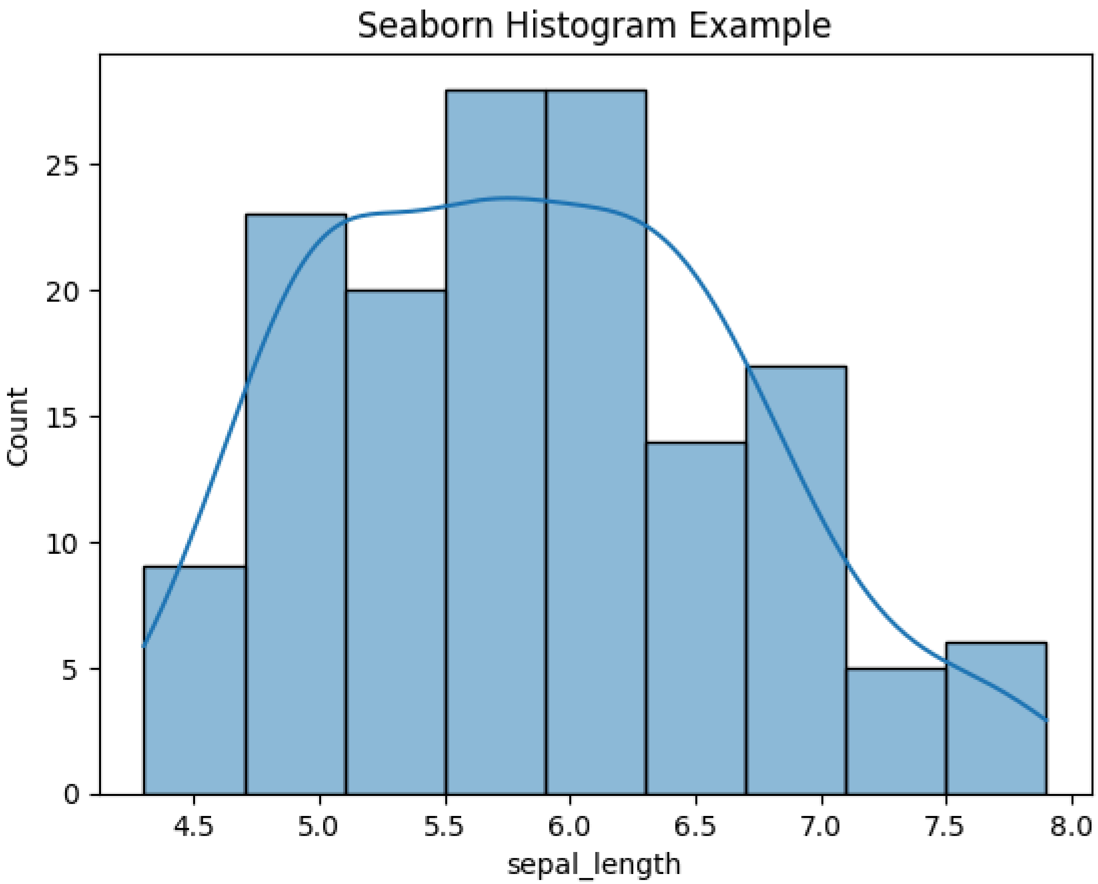


In [10]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("tips")

# Create bar chart
sns.barplot(x="day", y="total_bill", data=data)
plt.title('Seaborn Bar Chart Example')
plt.show()
```



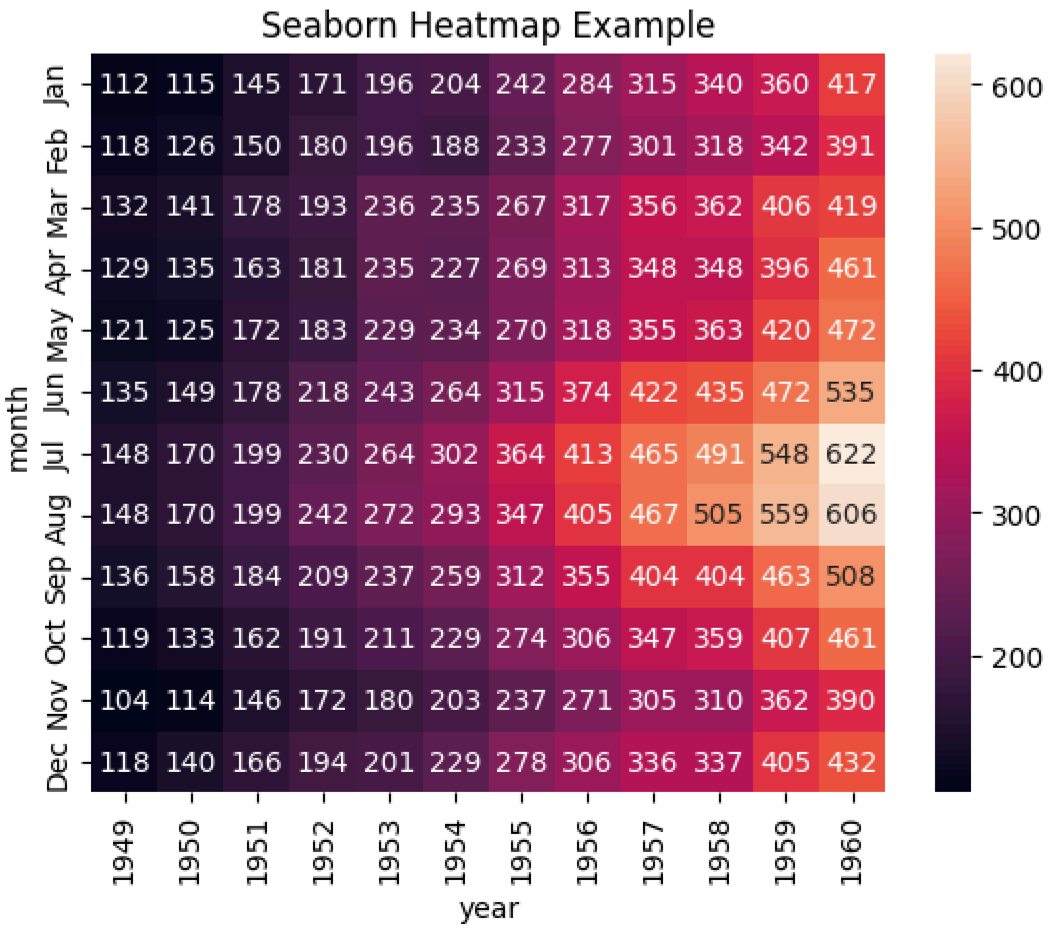
In [11]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("iris")

# Create histogram
sns.histplot(data["sepal_length"], kde=True)
plt.title('Seaborn Histogram Example')
plt.show()
```

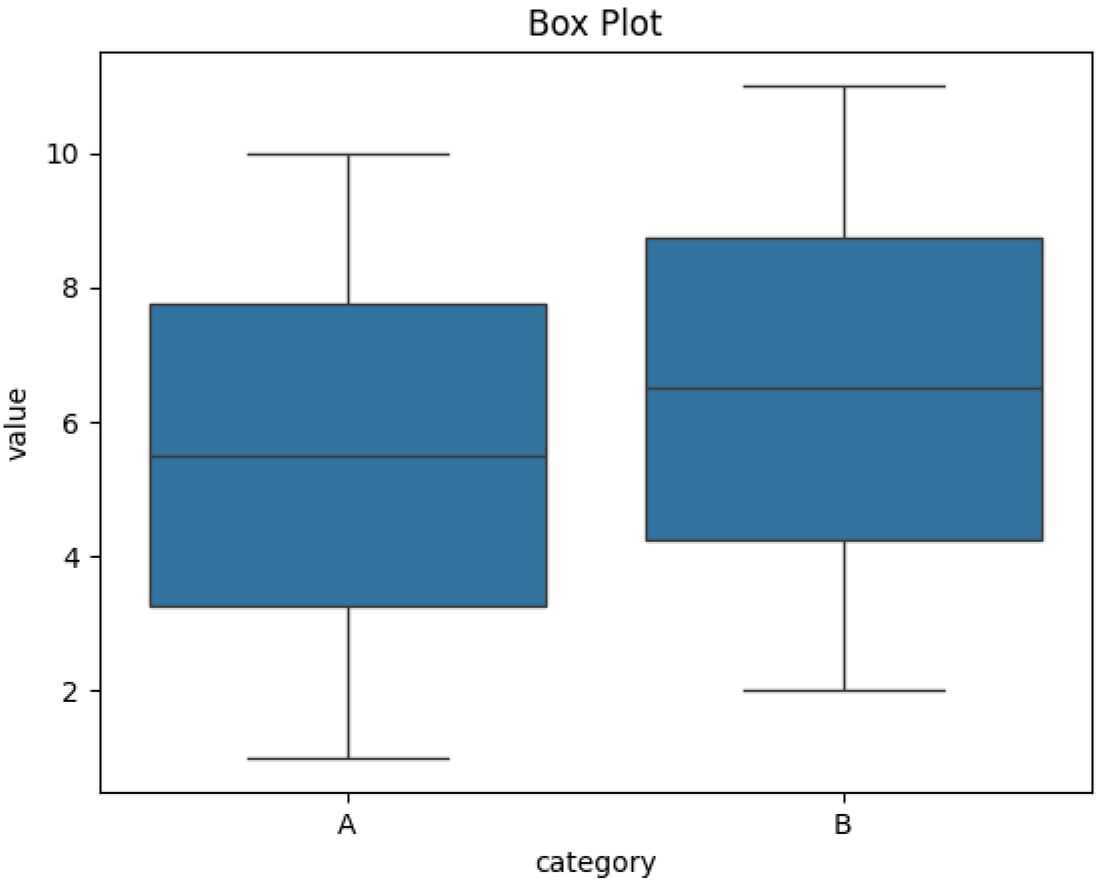


In [12]:

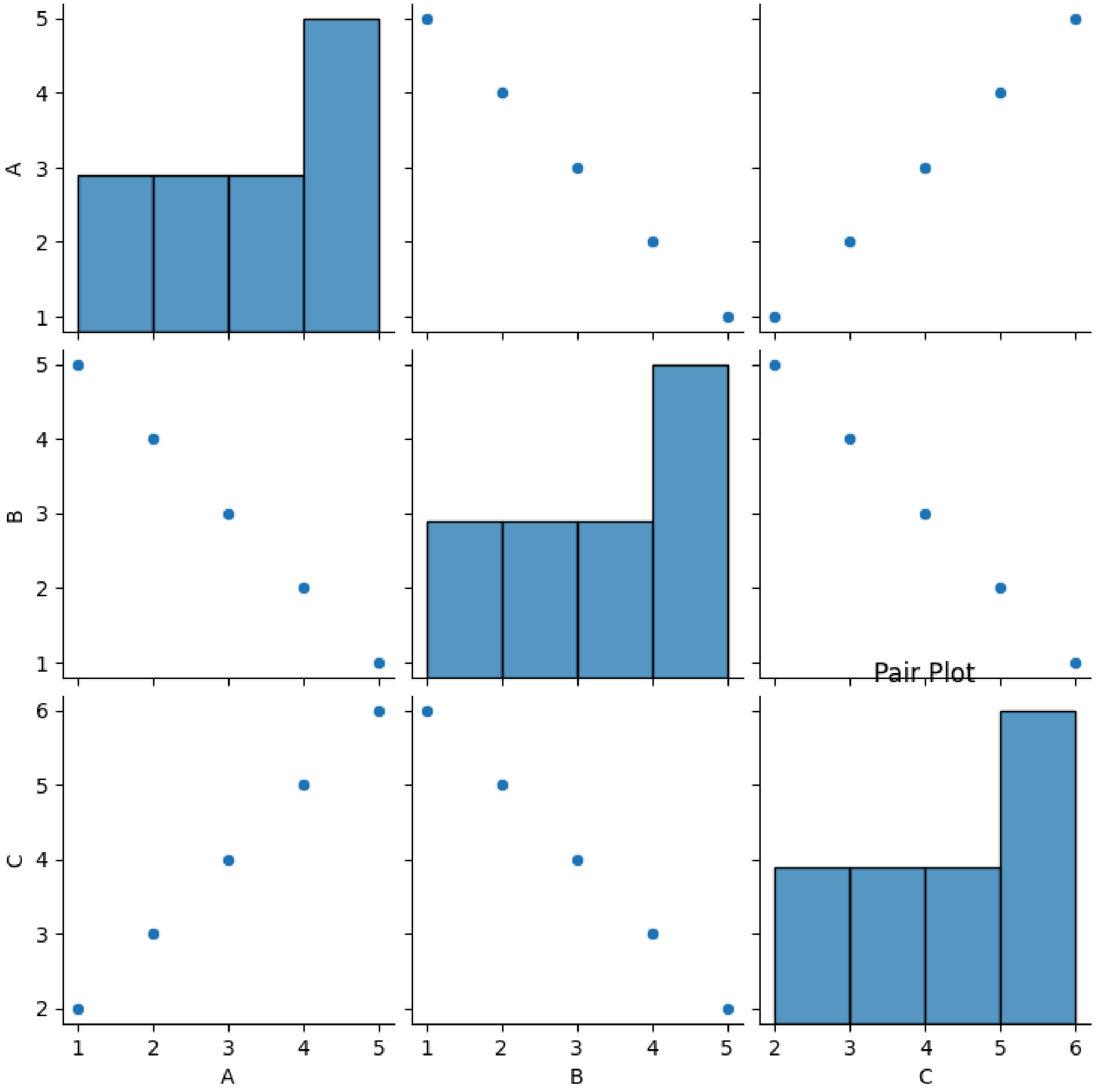
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = sns.load_dataset("flights")
pivot_data = data.pivot(index="month", columns="year", values="passengers")

# Create heatmap
sns.heatmap(pivot_data, annot=True, fmt="d")
plt.title('Seaborn Heatmap Example')
plt.show()
```



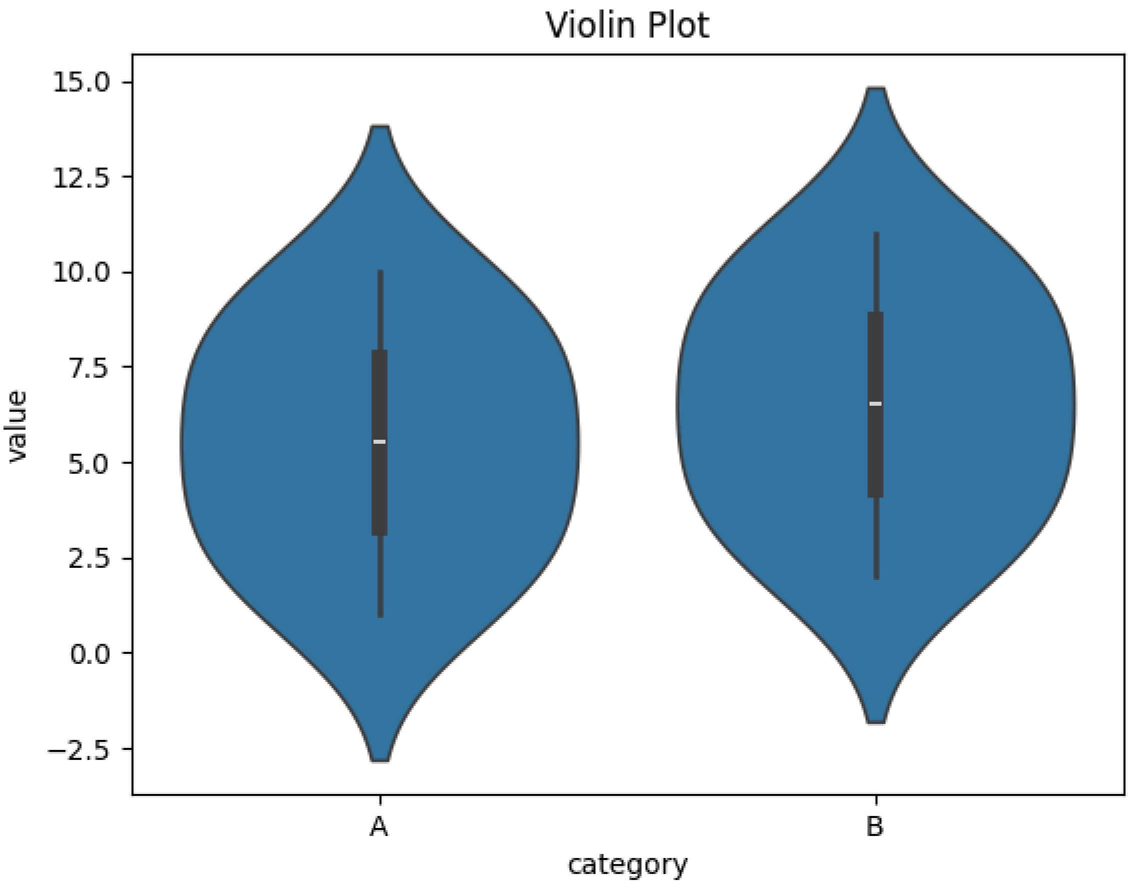
```
In [13]: import seaborn as sns
         import pandas as pd

         data = pd.DataFrame({
             "category": ['A']*10 + ['B']*10,
             "value": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
         })

         sns.boxplot(data=data, x="category", y="value")
         plt.title("Box Plot")
         plt.show()
```



In [14]:
```
import seaborn as sns
import pandas as pd

data = pd.DataFrame({
    "A": [1, 2, 3, 4, 5],
    "B": [5, 4, 3, 2, 1],
    "C": [2, 3, 4, 5, 6]
})

sns.pairplot(data)
plt.title("Pair Plot")
plt.show()
```

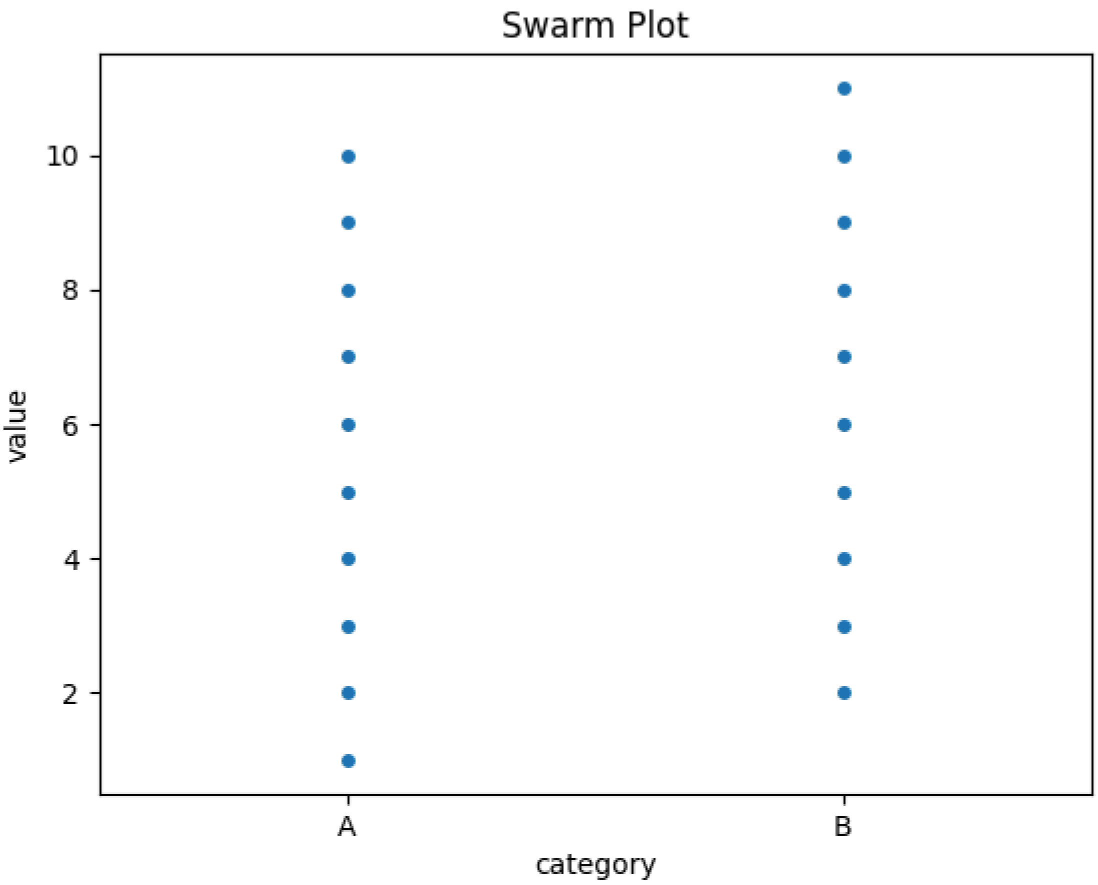


In [15]:
```
import seaborn as sns
import pandas as pd

data = pd.DataFrame({
    "category": ['A']*10 + ['B']*10,
    "value": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
})

sns.violinplot(data=data, x="category", y="value")
plt.title("Violin Plot")
plt.show()
```



In [16]:

```
import seaborn as sns
import pandas as pd

data = pd.DataFrame({
    "category": ['A']*10 + ['B']*10,
    "value": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
})

sns.swarmplot(data=data, x="category", y="value")
plt.title("Swarm Plot")
plt.show()
```



🧑‍💻 Strengths and Weaknesses of Matplotlib and Seaborn

🎨 Matplotlib

Strengths

🔑 Strength	📊 Details
Flexibility and Control	🔧 Matplotlib offers complete control over every aspect of a plot, allowing highly customized visualizations.
Versatility	🔄 Supports a wide variety of plots, including line, scatter, bar, histograms, 3D plots, and more. Also supports animations and interactive plots.
Integration	🔗 Works seamlessly with libraries like NumPy, Pandas, SciPy, and GUI toolkits such as Tkinter, wxPython, and Qt.
Large Community & Documentation	📚 Extensive resources, tutorials, and support from a large user community.

Weaknesses

❌ Weakness	💬 Details
Complexity	🧠 Requires more code for simple plots and can become verbose due to its extensive customization options.
Default Styles	💻 The default aesthetic of plots can appear outdated and less attractive compared to more modern libraries like Seaborn.
Interactivity	🖱️ Basic interactivity is supported, but it lacks the advanced interactive capabilities of libraries like Plotly or Bokeh.
Performance with Large Datasets	📉 Can be slow with very large datasets, particularly for interactive visualizations.

🌈 Seaborn

Strengths

🔑 Strength	📊 Details
Ease of Use	✨ Simplifies the creation of complex statistical plots with minimal code, ideal for quick data visualization and exploration.
Aesthetics	🌟 Beautiful default styles and color palettes enhance visual appeal without needing customization.
Integration with Pandas	🐼 Works seamlessly with Pandas DataFrames, making it easy to create visualizations directly from structured data.
Statistical Visualization	📉 Specialized in statistical plots like violin plots, box plots, and heatmaps.

Weaknesses

❌ Weakness	💬 Details
Limited Customization	⚙️ Provides fewer customization options compared to Matplotlib, limiting detailed adjustments for complex visualizations.
Dependency on Matplotlib	🔗 Built on top of Matplotlib, so advanced customization requires understanding Matplotlib's features.
Performance with Large Datasets	📉 Struggles with very large datasets due to high-level statistical computations that can be resource-intensive.
Interactivity	🖱️ Primarily focused on static plots, requiring additional tools for interactivity.

⚖️ Comparison

🏷️ Aspect	🎨 Matplotlib	🌈 Seaborn
Ease of Use	🧑‍💻 Highly customizable but requires more code for basic plots.	🧑‍🏫 Simplifies the process of creating beautiful, informative plots with minimal code.
Customization Options	⚙️ Extensive customization options for detailed visual adjustments.	🎨 Provides fewer customization options, but with beautiful defaults.
Interactivity	🖱️ Basic interactivity; needs additional libraries like Plotly or Bokeh for advanced features.	🔒 Primarily static plots, but can integrate with Matplotlib's interactive capabilities.
Performance with Large Datasets	📊 Handles large datasets well but can slow down with very large data.	📉 Struggles with very large datasets due to its high-level nature and statistical features.

📝 Summary

- **Matplotlib** is ideal for users who need **full control** over their plots and want to create **publication-quality** figures. It is best suited for complex, highly customized visualizations, and integration with other Python libraries.
- **Seaborn** is best for users who need **quick, beautiful visualizations** and **statistical analysis** with **minimal code**. It excels in **exploratory data analysis (EDA)** and works seamlessly with Pandas.

👋 Hi, I'm Ashish Mishra

📊 Internship in Data Science

🏢 at ShadowFox

AIR QUALITY ANALYSIS (DELHI)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
```

In [2]:
```
df = pd.read_csv(r'C:\Users\Ashish Mishra\OneDrive\Desktop\ShadowFox\Shadowfox_DS\I
```

In [3]:
```
df.head()
```

Out[3]:

	date	co	no	no2	o3	so2	pm2_5	pm10	nh3
0	2023-01-01 00:00:00	1655.58	1.66	39.41	5.90	17.88	169.29	194.64	5.83
1	2023-01-01 01:00:00	1869.20	6.82	42.16	1.99	22.17	182.84	211.08	7.66
2	2023-01-01 02:00:00	2510.07	27.72	43.87	0.02	30.04	220.25	260.68	11.40
3	2023-01-01 03:00:00	3150.94	55.43	44.55	0.85	35.76	252.90	304.12	13.55
4	2023-01-01 04:00:00	3471.37	68.84	45.24	5.45	39.10	266.36	322.80	14.19

In [4]:
```
df.shape
```

Out[4]:
```
(561, 9)
```

In [5]:
```
df.dtypes
```

Out[5]:
```
date      object
co       float64
no       float64
no2      float64
o3       float64
so2      float64
pm2_5    float64
pm10     float64
nh3      float64
dtype: object
```

In [6]:
```
df.columns
```

```
Out[6]: Index(['date', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3'], dtype='obj
        ect')
```

```
In [7]: df.size
```

```
Out[7]: 5049
```

```
In [8]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 561 entries, 0 to 560
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   date    561 non-null    object 
 1   co      561 non-null    float64
 2   no      561 non-null    float64
 3   no2     561 non-null    float64
 4   o3      561 non-null    float64
 5   so2     561 non-null    float64
 6   pm2_5   561 non-null    float64
 7   pm10    561 non-null    float64
 8   nh3     561 non-null    float64
dtypes: float64(8), object(1)
memory usage: 39.6+ KB
```

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

```
Out[9]: date                 co       no     no2     o3     so2     pm2_5   pm10    nh3  
        2023-01-01 00:00:00  1655.58  1.66   39.41   5.90   17.88   169.29  194.64  5.83 
        1
        2023-01-16 17:00:00  2857.21  8.72   80.20   1.31   41.48   211.19  274.45  31.92
        1
        2023-01-16 11:00:00  1949.31  11.51  74.03   67.23  58.65   135.85  172.38  22.80
        1
        2023-01-16 12:00:00  2670.29  15.65  111.04  18.24  59.13   163.88  211.14  29.13
        1
        2023-01-16 13:00:00  3257.75  28.16  117.90  0.11   60.08   194.19  251.70  36.98
        1
        
        ..
        2023-01-08 13:00:00  4005.43  32.19  124.75  0.44   39.10   370.36  425.01  16.47
        1
        2023-01-08 12:00:00  2990.72  3.74   112.41  28.97  39.10   327.78  365.97  14.31
        1
        2023-01-08 11:00:00  2590.18  5.59   76.77   86.55  46.73   325.19  357.19  15.07
        1
        2023-01-08 10:00:00  2136.23  4.92   50.04   131.61 57.22   308.40  332.44  12.92
        1
        2023-01-24 08:00:00  1134.87  8.61   56.89   80.11  110.63  123.76  140.26  5.51 
        1
        Name: count, Length: 561, dtype: int64
```

```
In [10]: display(df.describe().T)
```

	count	mean	std	min	25%	50%	75%	max
co	561.0	3814.942210	3227.744681	654.22	1708.98	2590.18	4432.68	16876.22
no	561.0	51.181979	83.904476	0.00	3.38	13.30	59.01	425.58
no2	561.0	75.292496	42.473791	13.37	44.55	63.75	97.33	263.21
o3	561.0	30.141943	39.979405	0.00	0.07	11.80	47.21	164.51
so2	561.0	64.655936	61.073080	5.25	28.13	47.21	77.25	511.17
pm2_5	561.0	358.256364	227.359117	60.10	204.45	301.17	416.65	1310.20
pm10	561.0	420.988414	271.287026	69.08	240.90	340.90	482.57	1499.27
nh3	561.0	26.425062	36.563094	0.63	8.23	14.82	26.35	267.51

In [11]:
```
df.isna()
```

Out[11]:

	date	co	no	no2	o3	so2	pm2_5	pm10	nh3
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
556	False	False	False	False	False	False	False	False	False
557	False	False	False	False	False	False	False	False	False
558	False	False	False	False	False	False	False	False	False
559	False	False	False	False	False	False	False	False	False
560	False	False	False	False	False	False	False	False	False

561 rows × 9 columns

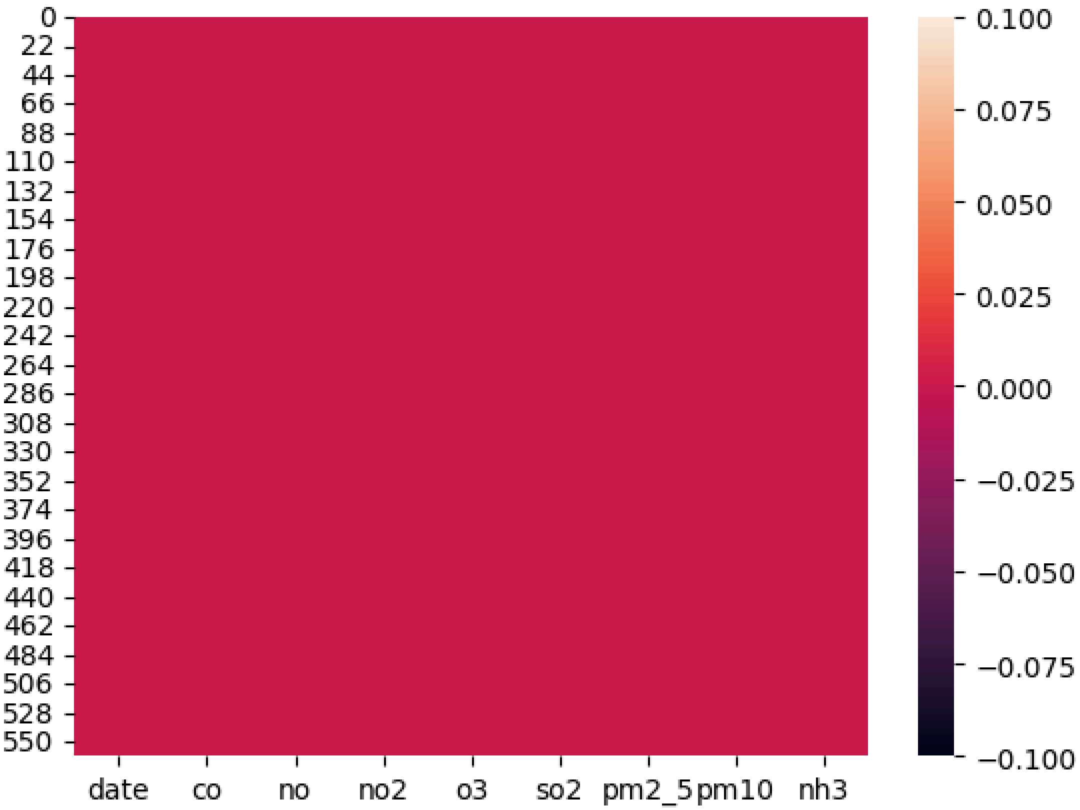
In [12]:
```
df.dropna(inplace=True)
```

In [13]:
```
df.isna().any()
```

Out[13]:
```
date     False
co       False
no       False
no2      False
o3       False
so2      False
pm2_5    False
pm10     False
nh3      False
dtype: bool
```

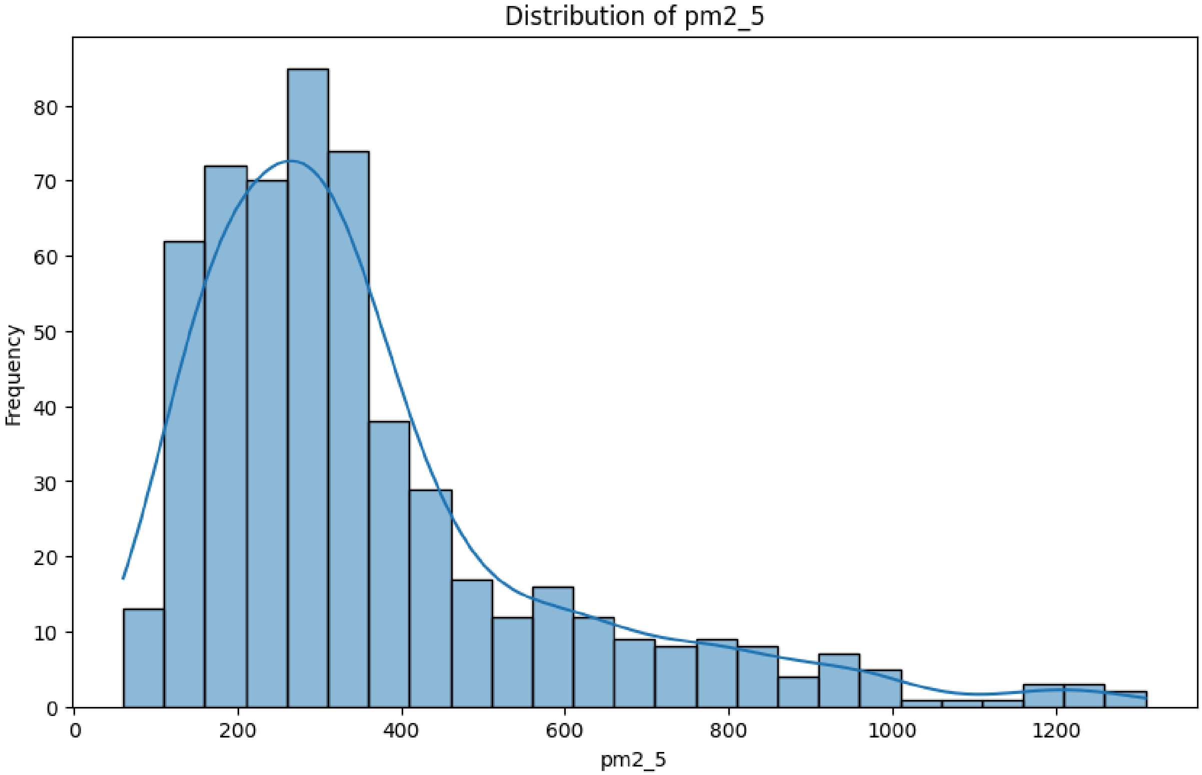
In [14]:
```
df.fillna(method='ffill',inplace=True)
sns.heatmap(df.isna(),cbar=True)
```

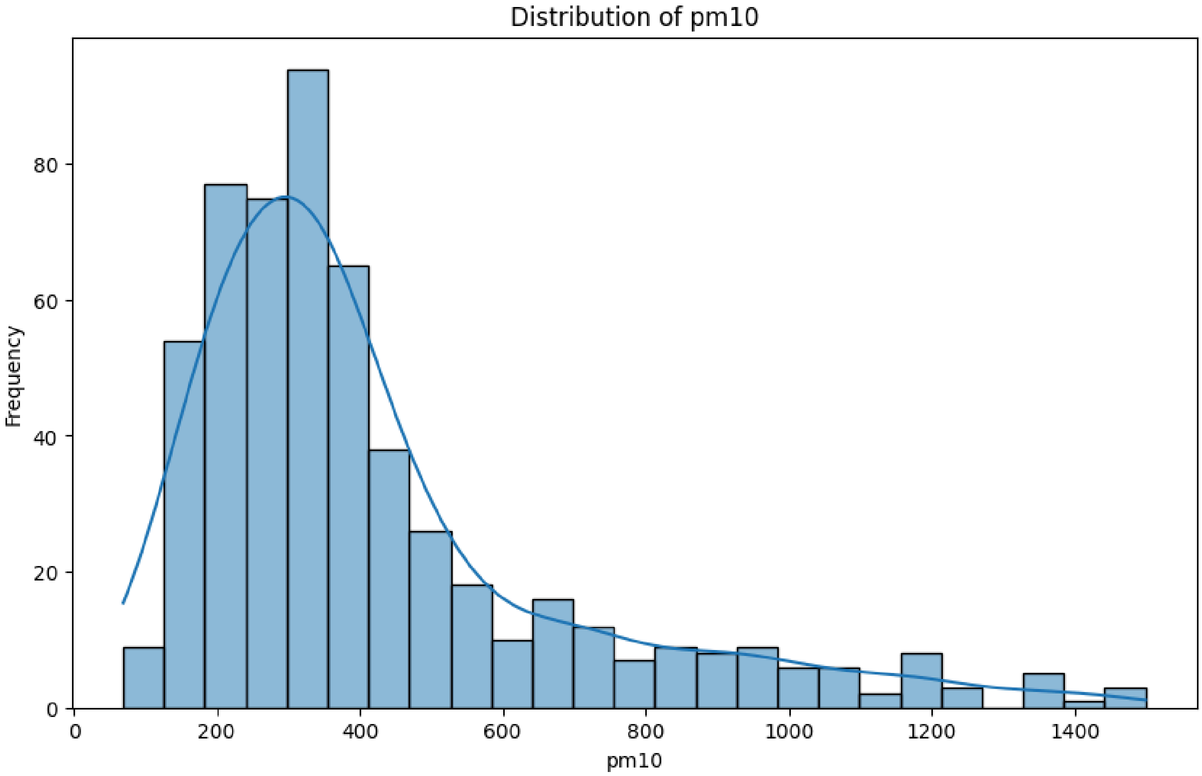
Out[14]:
```
<Axes: >
```

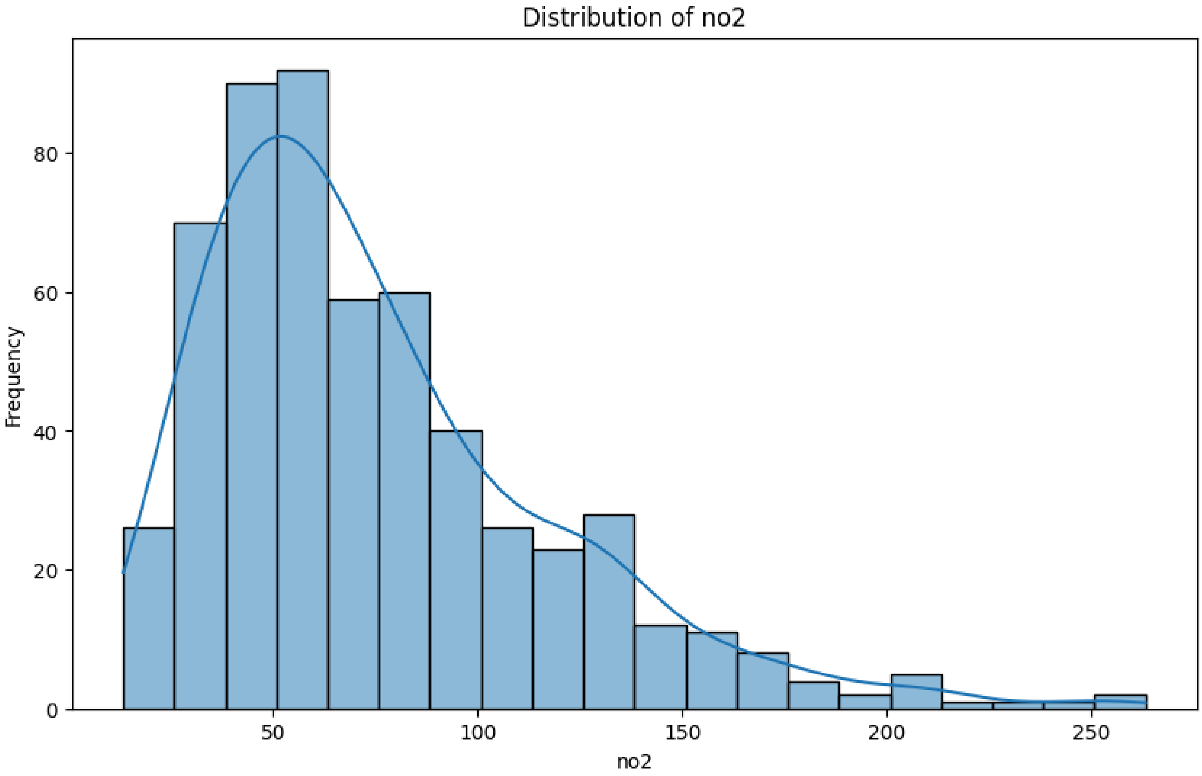


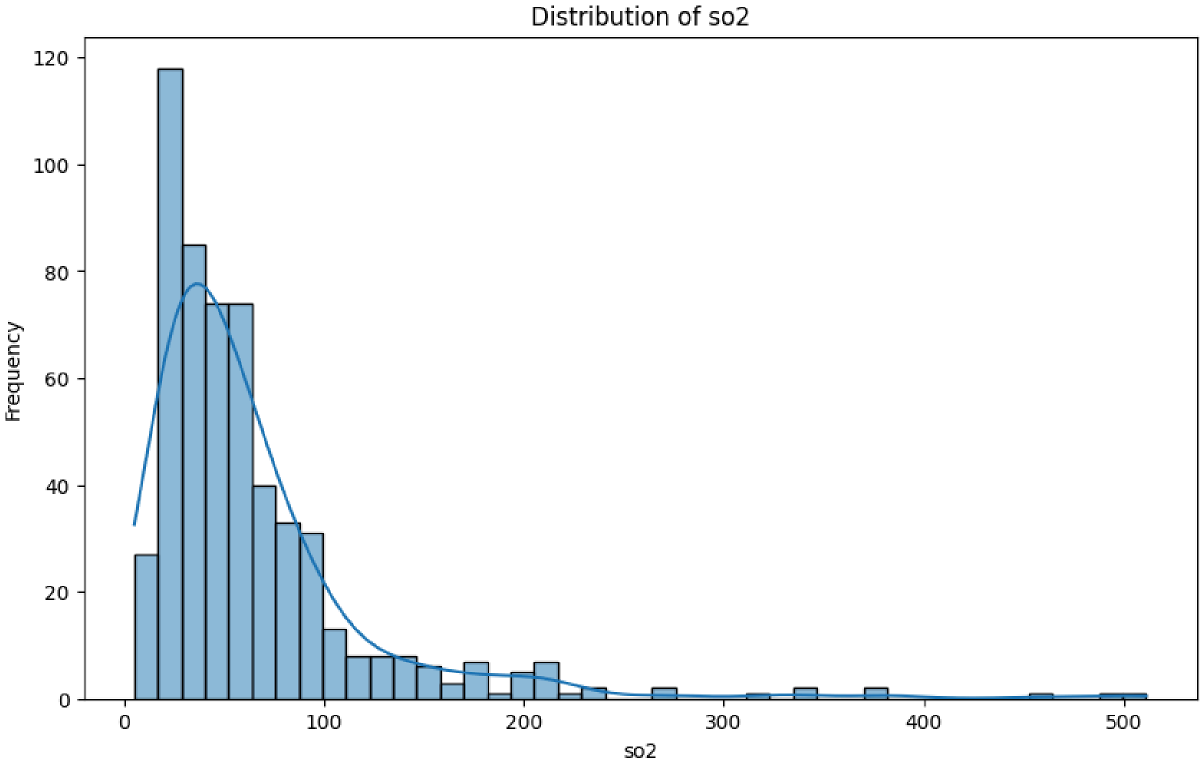
In [15]:
```
df['date'] = pd.to_datetime(df['date'])

def calculate_aqi(row):

    return max(row['pm2_5'], row['pm10'], row['no2'], row['o3'], row['co'], row['so

df['AQI'] = df.apply(calculate_aqi, axis=1)
```

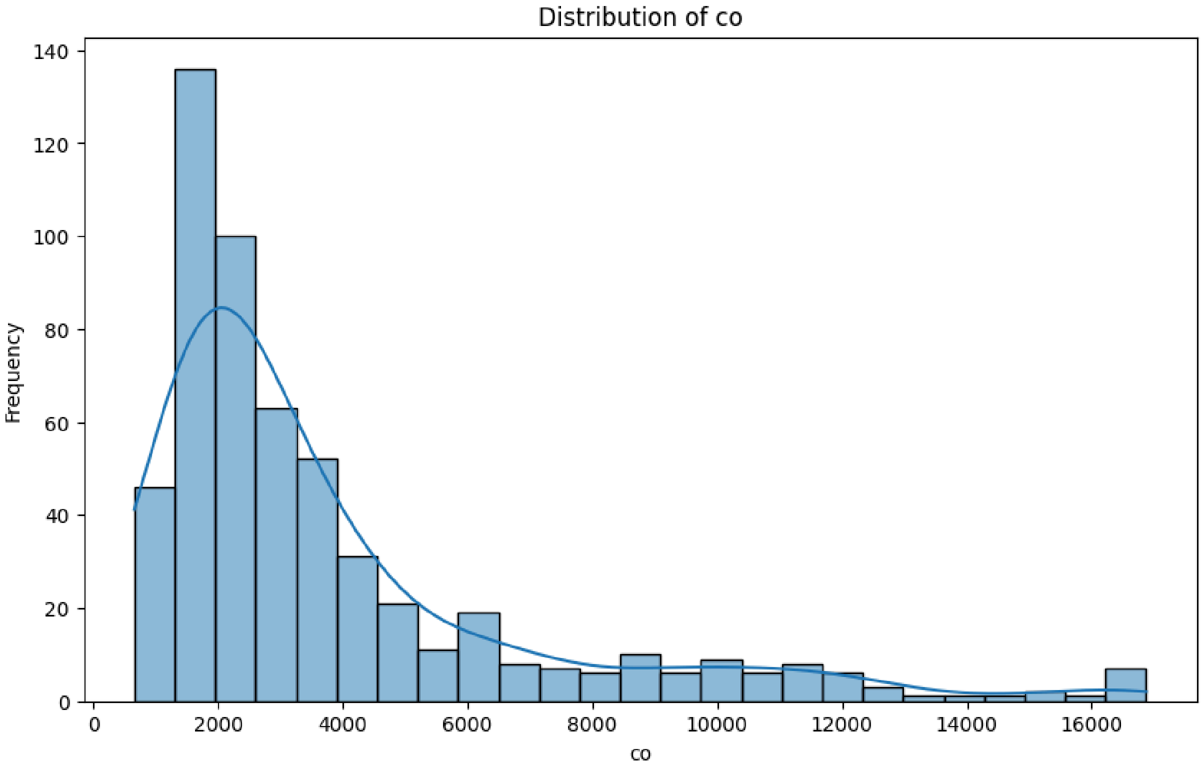
In [16]:
```
pollutants = ['pm2_5', 'pm10', 'no2', 'so2', 'co', 'o3', 'nh3', 'no', 'AQI']

for pollutant in pollutants:
    plt.figure(figsize=(10, 6))
    sns.histplot(df[pollutant].dropna(), kde=True)
    plt.title(f'Distribution of {pollutant}')
    plt.xlabel(pollutant)
    plt.ylabel('Frequency')
    plt.show()
```

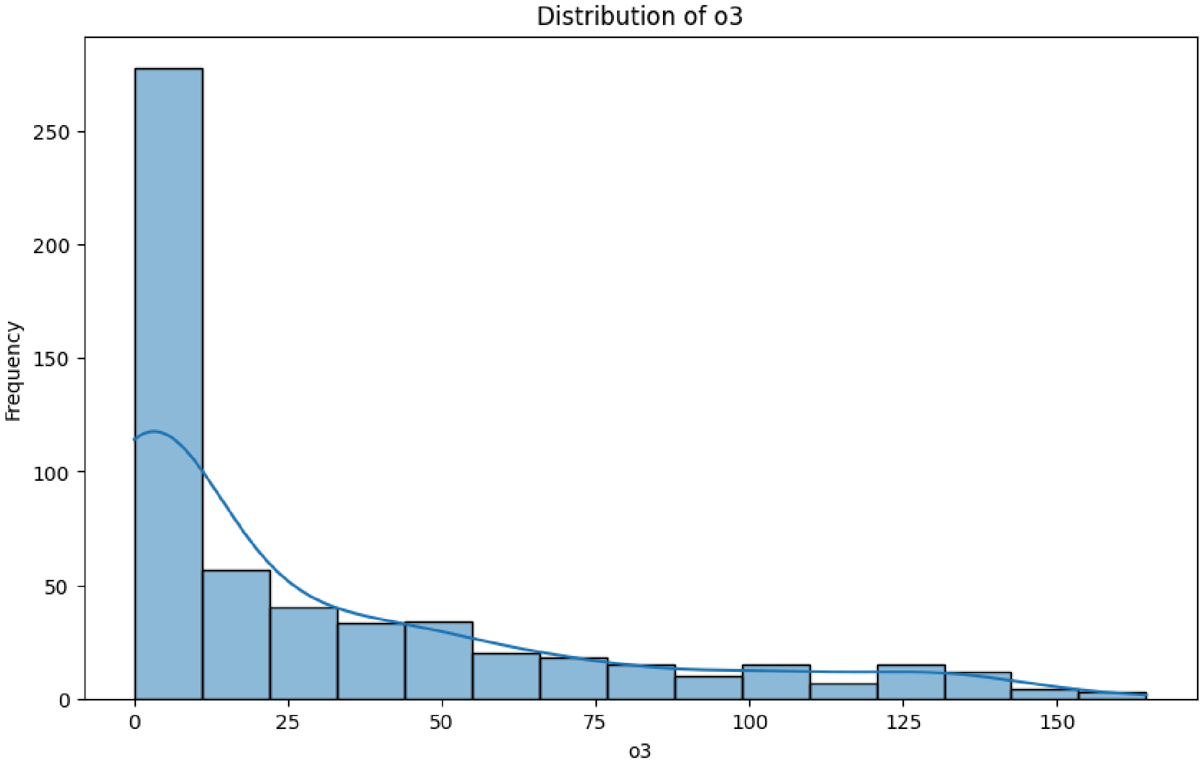

Distribution of pm2_5

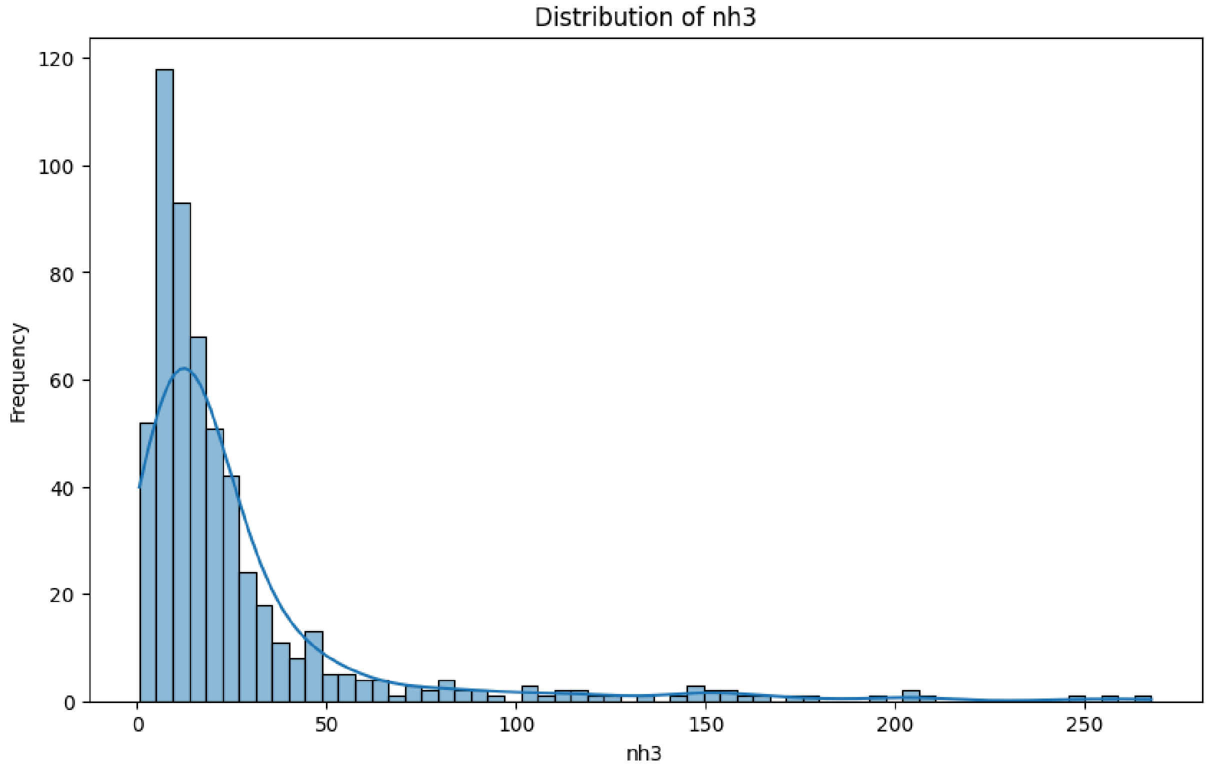

Distribution of pm10

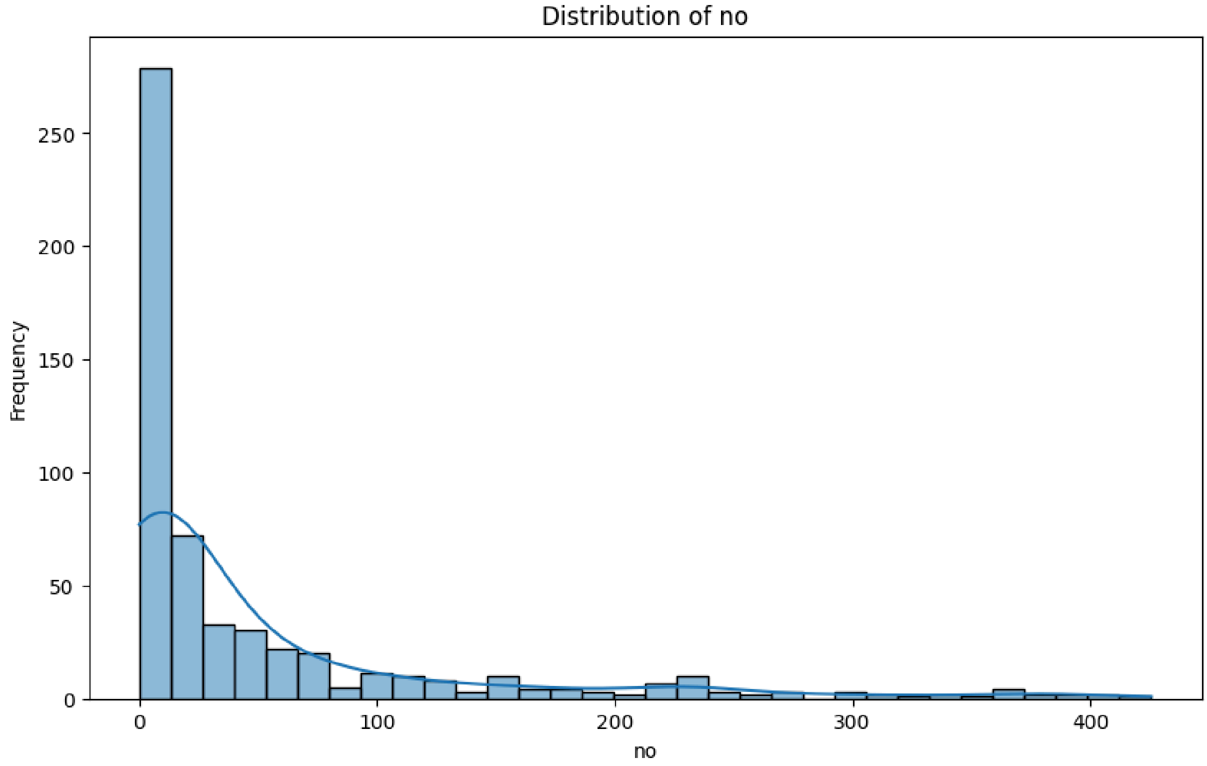

Distribution of no2
Frequency
0
20
40
60
80
50
100
150
200
250
no2

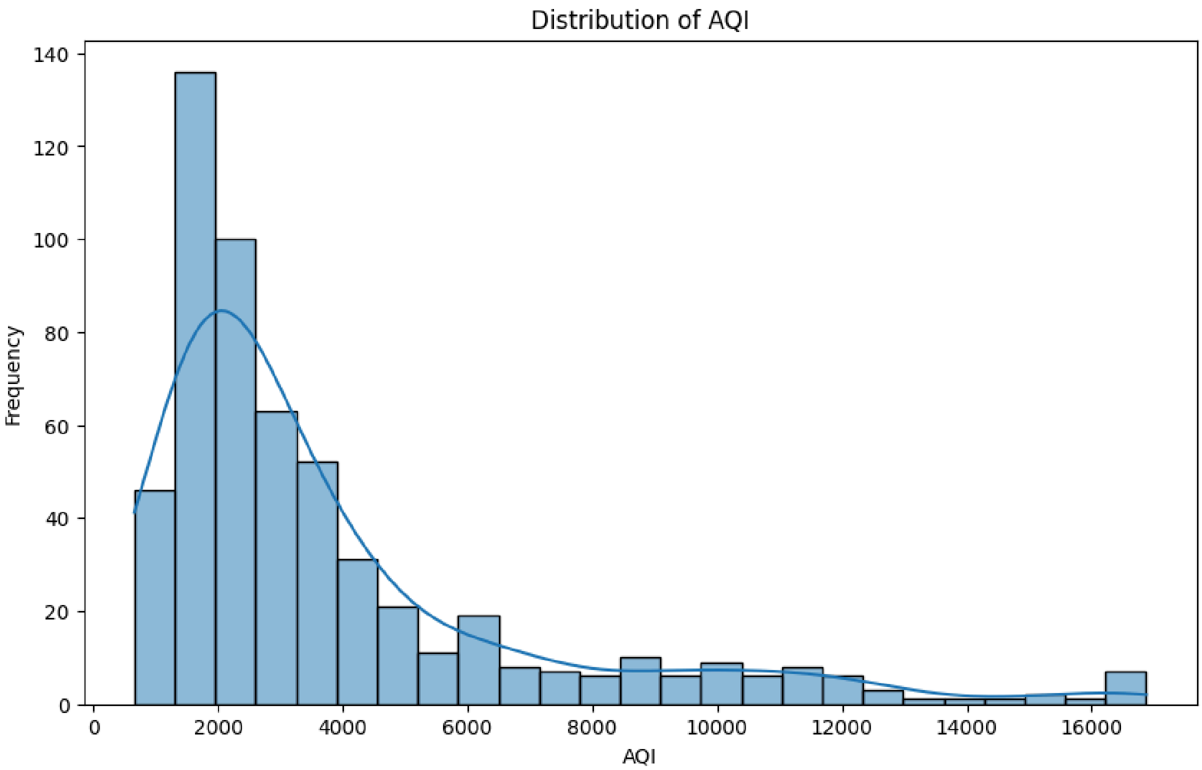

Distribution of so2
Frequency
0
20
40
60
80
100
120
0
100
200
300
400
500
so2


Distribution of co
Frequency
0
20
40
60
80
100
120
140
0
2000
4000
6000
8000
10000
12000
14000
16000
co


Distribution of o3
Frequency
0
50
100
150
200
250
0
25
50
75
100
125
150
o3


Distribution of nh3
Frequency
0
20
40
60
80
100
120
0
50
100
150
200
250
nh3


Distribution of no
Frequency
0
50
100
150
200
250
0
100
200
300
400
no



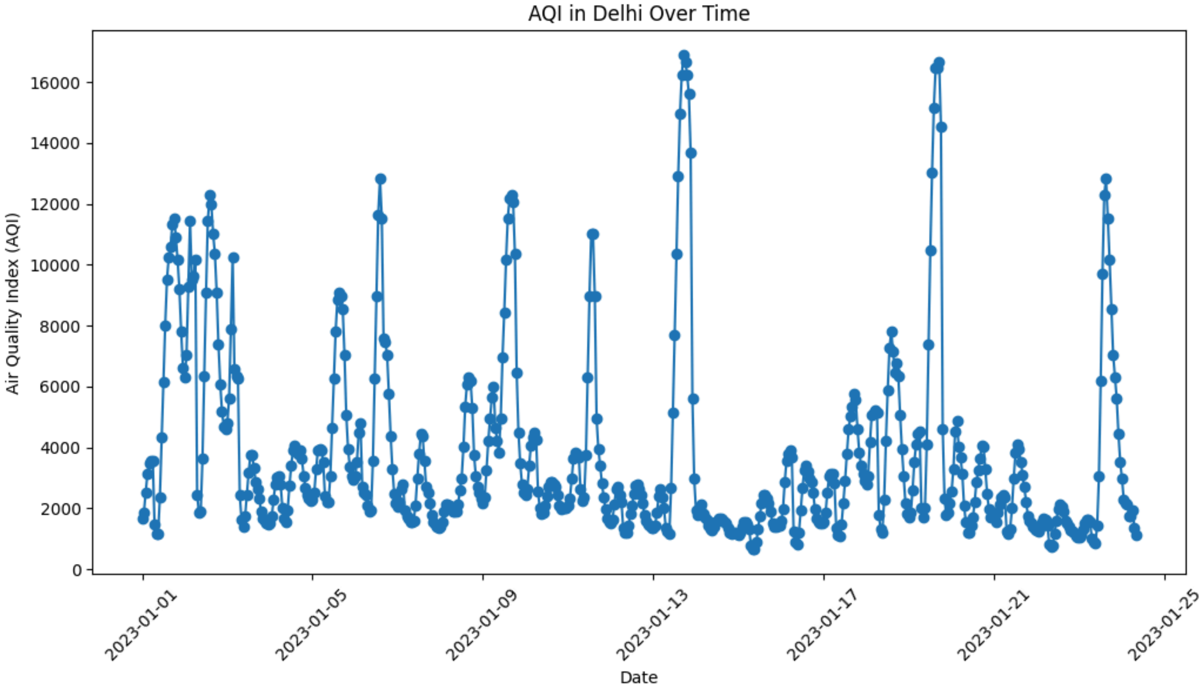
In [17]:
```
corr = df[['pm2_5', 'pm10','no2', 'o3', 'co', 'so2', 'no','nh3','co','AQI']].corr()
print(corr)
```

```
          pm2_5      pm10       no2        o3        co       so2        no  \
pm2_5  1.000000  0.994088  0.698696 -0.450458  0.953083  0.648996  0.888810   
pm10   0.994088  1.000000  0.720050 -0.468477  0.966801  0.658325  0.903339   
no2    0.698696  0.720050  1.000000 -0.407177  0.776402  0.734961  0.702201   
o3    -0.450458 -0.468477 -0.407177  1.000000 -0.463082 -0.049158 -0.377813   
co     0.953083  0.966801  0.776402 -0.463082  1.000000  0.716831  0.969740   
so2    0.648996  0.658325  0.734961 -0.049158  0.716831  1.000000  0.734503   
no     0.888810  0.903339  0.702201 -0.377813  0.969740  0.734503  1.000000   
nh3    0.720303  0.754468  0.700254 -0.299663  0.826299  0.843635  0.823638   
co     0.953083  0.966801  0.776402 -0.463082  1.000000  0.716831  0.969740   
AQI    0.953083  0.966801  0.776402 -0.463082  1.000000  0.716831  0.969740   

            nh3        co       AQI  
pm2_5  0.720303  0.953083  0.953083  
pm10   0.754468  0.966801  0.966801  
no2    0.700254  0.776402  0.776402  
o3    -0.299663 -0.463082 -0.463082  
co     0.826299  1.000000  1.000000  
so2    0.843635  0.716831  0.716831  
no     0.823638  0.969740  0.969740  
nh3    1.000000  0.826299  0.826299  
co     0.826299  1.000000  1.000000  
AQI    0.826299  1.000000  1.000000  
```

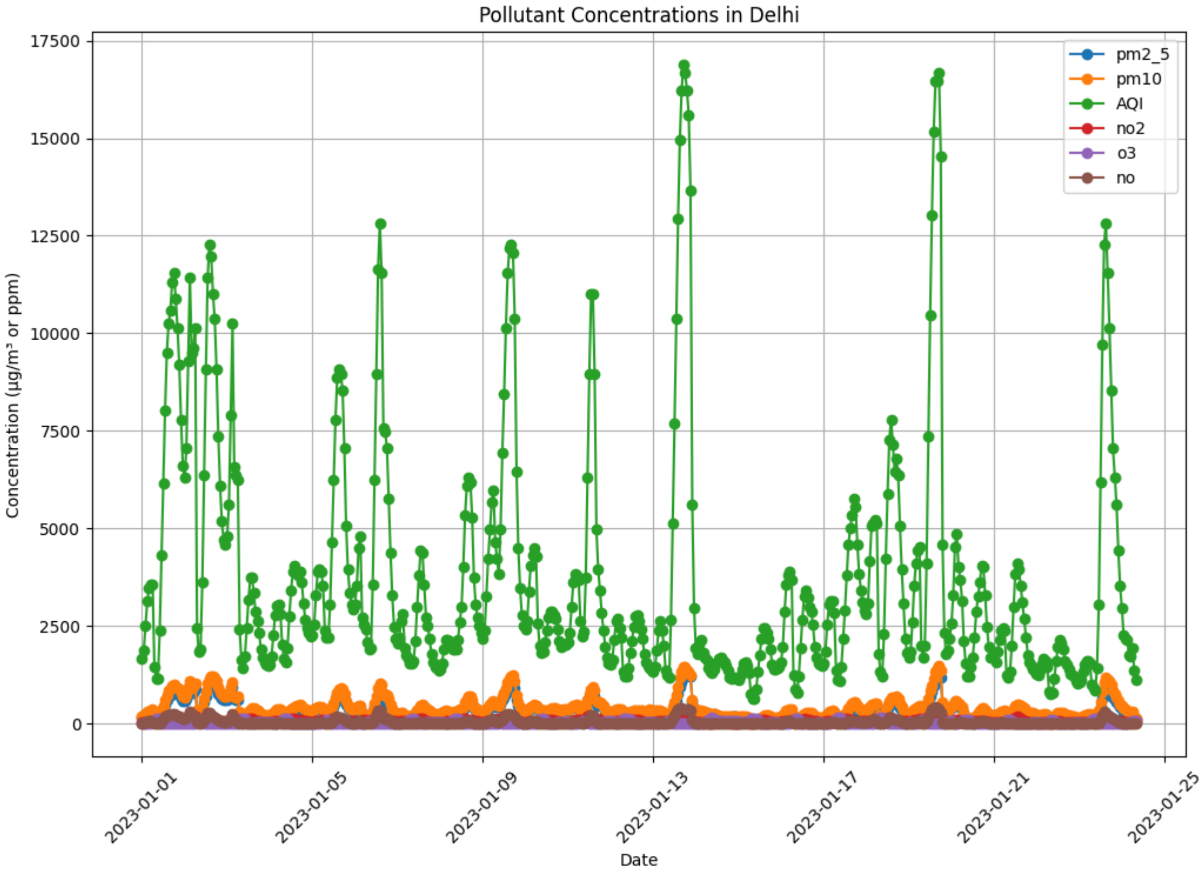
In [18]:
```
seasonal_data = df.groupby('date').agg({'AQI': 'mean', 'pm2_5': 'mean', 'pm10': 'me
print("Seasonal AQI and Pollutant Averages:")
print(seasonal_data.head())
```

```
Seasonal AQI and Pollutant Averages:
                         AQI   pm2_5    pm10
date
2023-01-01 00:00:00  1655.58  169.29  194.64
2023-01-01 01:00:00  1869.20  182.84  211.08
2023-01-01 02:00:00  2510.07  220.25  260.68
2023-01-01 03:00:00  3150.94  252.90  304.12
2023-01-01 04:00:00  3471.37  266.36  322.80
```

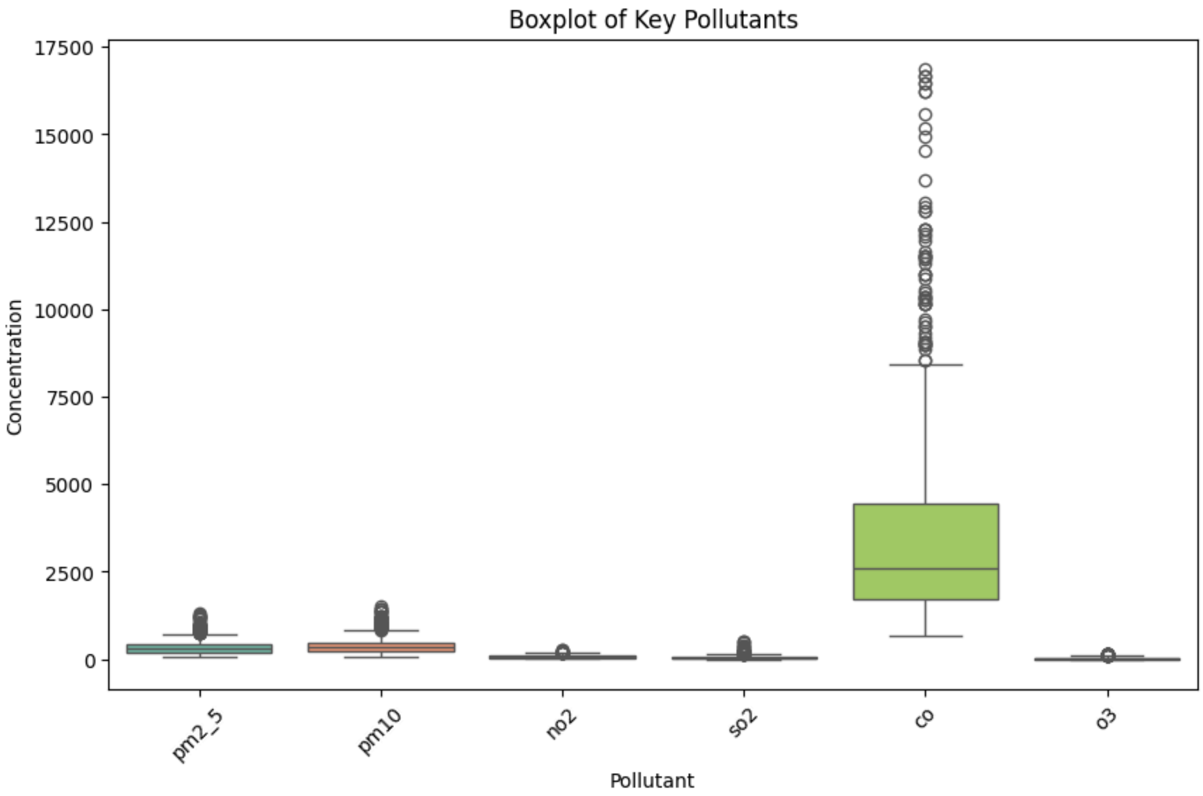
In [19]:
```
plt.figure(figsize=(12, 6))
plt.plot(df['date'], df['AQI'], marker='o', linestyle='-')
plt.xticks(rotation=45)
plt.xlabel('Date')
plt.ylabel('Air Quality Index (AQI)')
plt.title('AQI in Delhi Over Time')
plt.show()
```



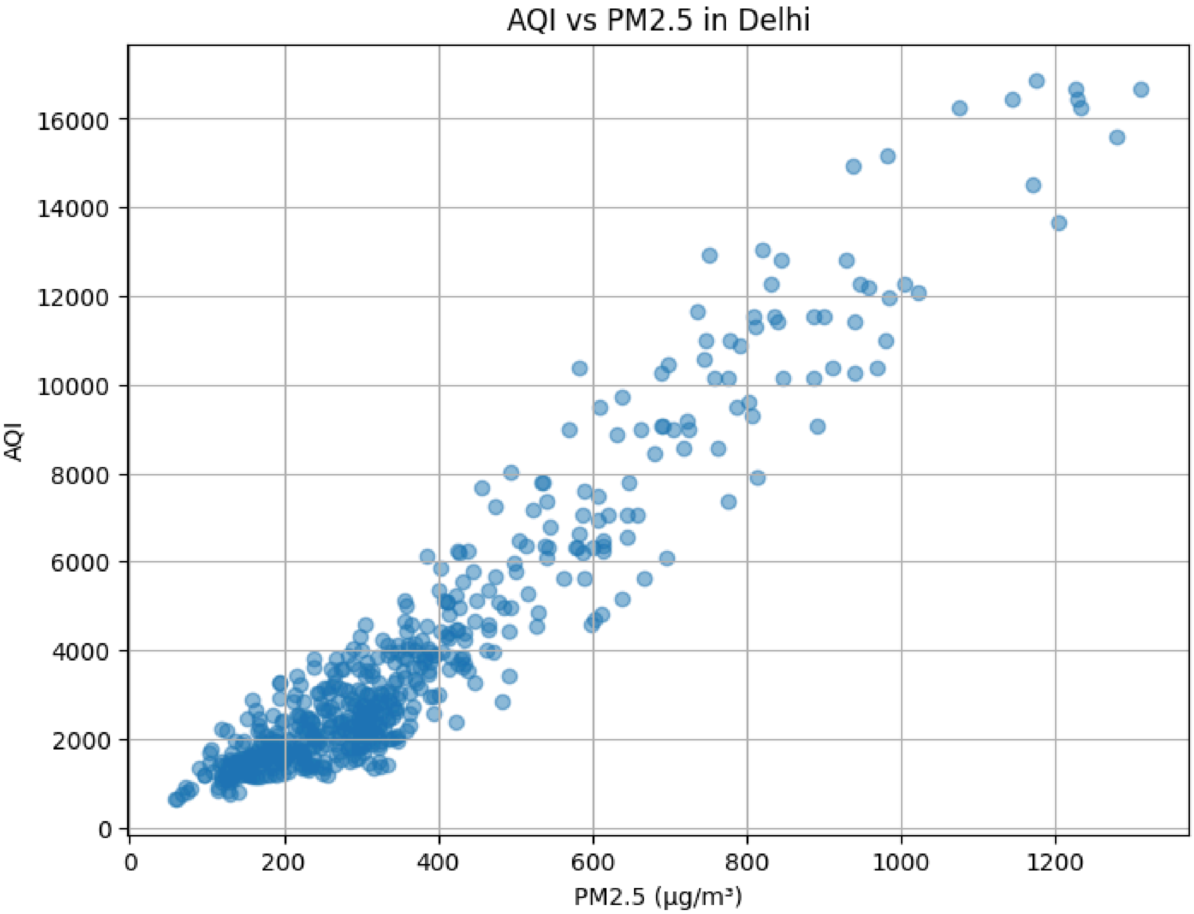
In [20]:
```
plt.figure(figsize=(12, 8))
pollutants = ['pm2_5', 'pm10', 'AQI', 'no2', 'o3', 'no']  # List of pollutants to p
for pollutant in pollutants:
    plt.plot(df['date'], df[pollutant], label=pollutant, marker='o', linestyle='-')
plt.title('Pollutant Concentrations in Delhi')
plt.xlabel('Date')
plt.ylabel('Concentration (µg/m³ or ppm)')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.show()
```



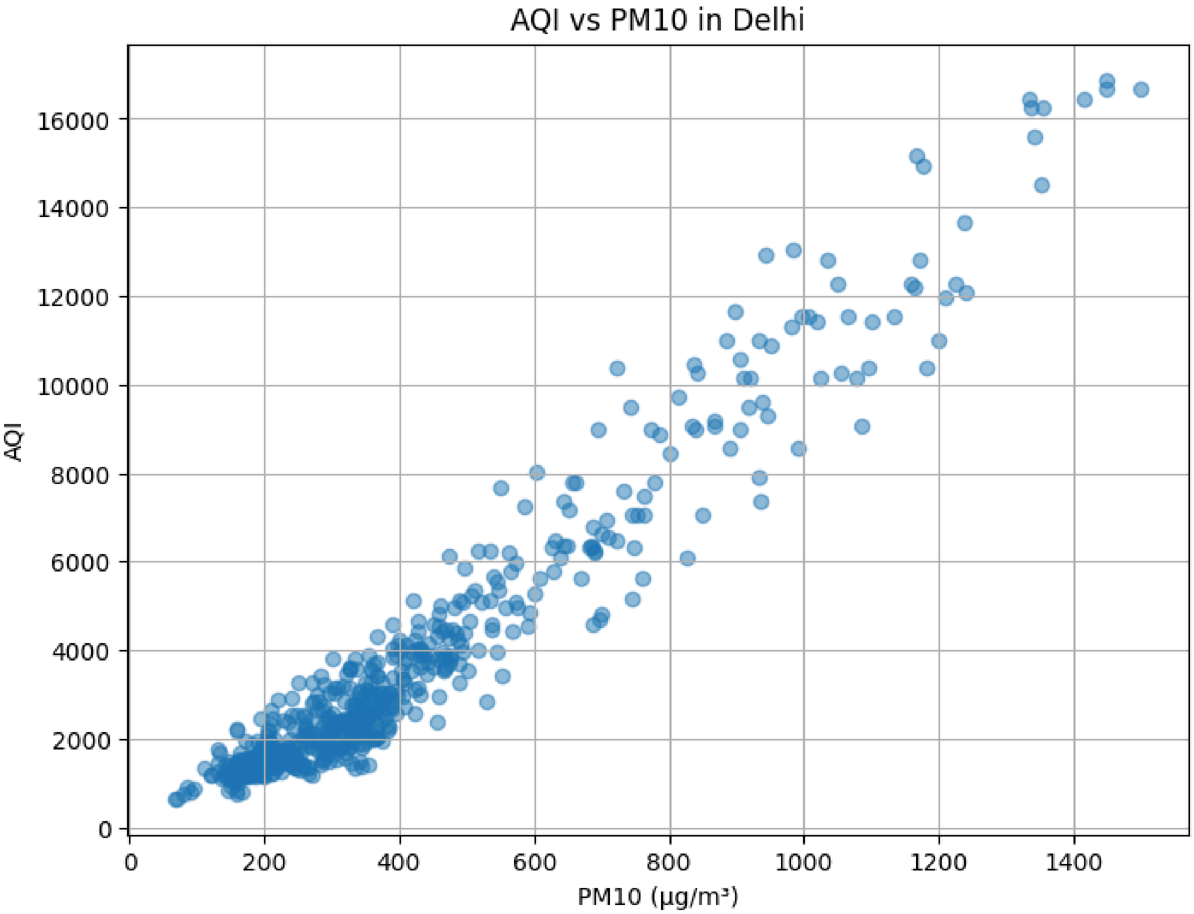
In [21]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[['pm2_5', 'pm10', 'no2', 'so2', 'co', 'o3']], palette='Set2')
plt.title('Boxplot of Key Pollutants')
plt.xlabel('Pollutant')
plt.ylabel('Concentration')
plt.xticks(rotation=45)
plt.show()
```



In [22]:
```
plt.figure(figsize=(8, 6))
plt.scatter(df['pm2_5'], df['AQI'], alpha=0.5)
plt.title('AQI vs PM2.5 in Delhi')
plt.xlabel('PM2.5 (µg/m³)')
plt.ylabel('AQI')
plt.grid(True)
plt.show()
```

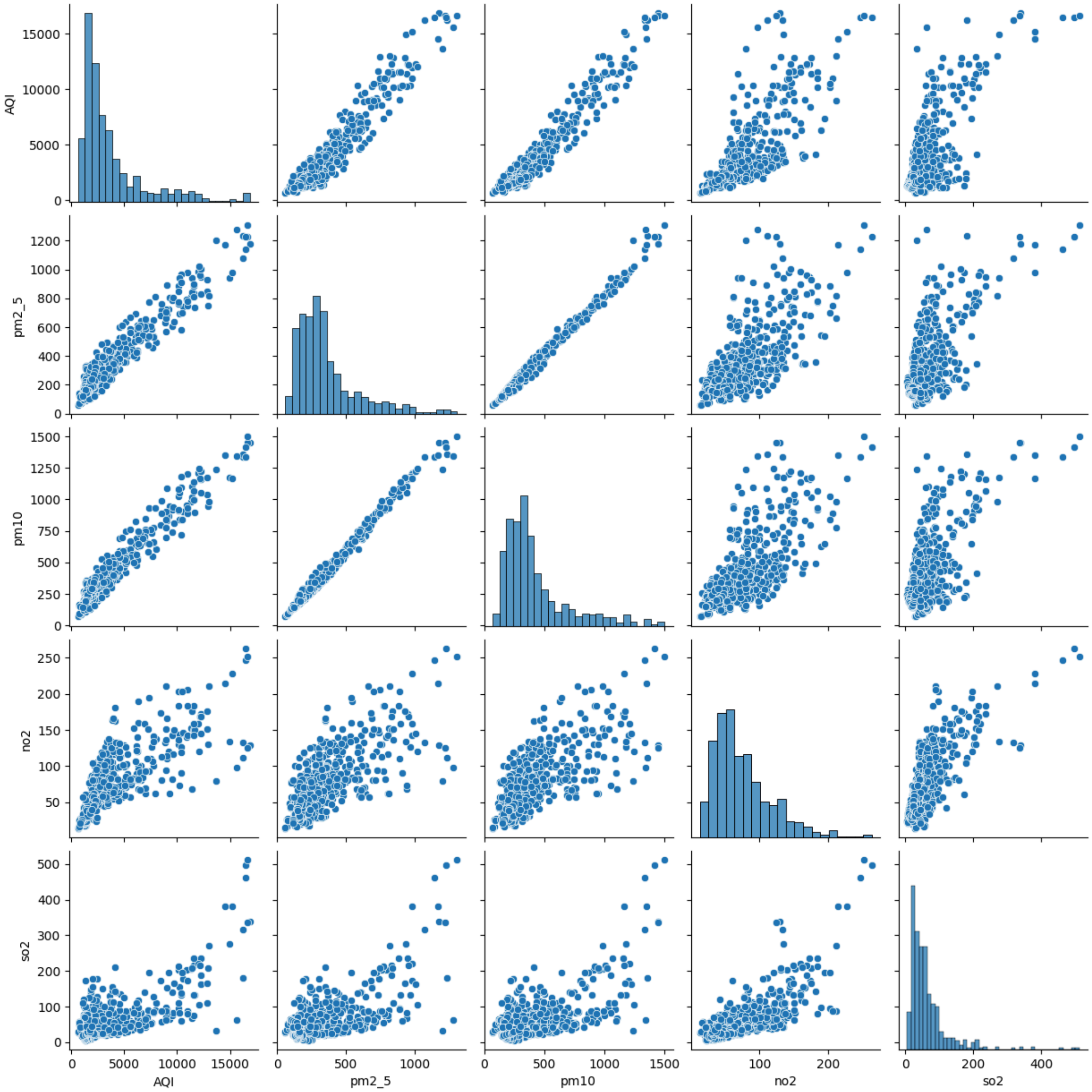


```
In [23]: plt.figure(figsize=(8, 6))
         plt.scatter(df['pm10'], df['AQI'], alpha=0.5)
         plt.title('AQI vs PM10 in Delhi')
         plt.xlabel('PM10 (µg/m³)')
         plt.ylabel('AQI')
         plt.grid(True)
         plt.show()
```



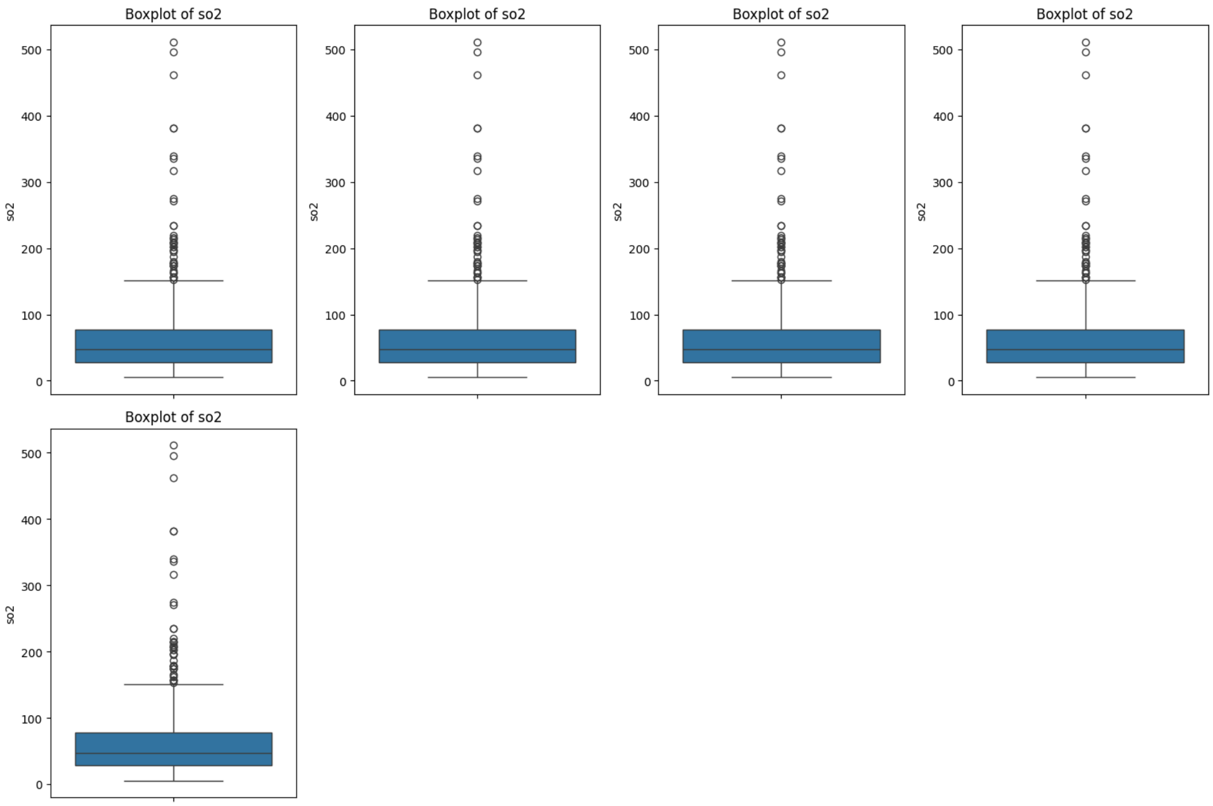
```
In [ ]: df['AQI'] = pd.to_numeric(df['AQI'], errors='coerce')
        pollutants = ['pm2_5', 'pm10', 'no2', 'so2']
        for pollutant in pollutants:
            df[pollutant] = pd.to_numeric(df[pollutant], errors='coerce')
```

```
In [25]: sns.pairplot(df[['AQI'] + pollutants])
         plt.show()
```

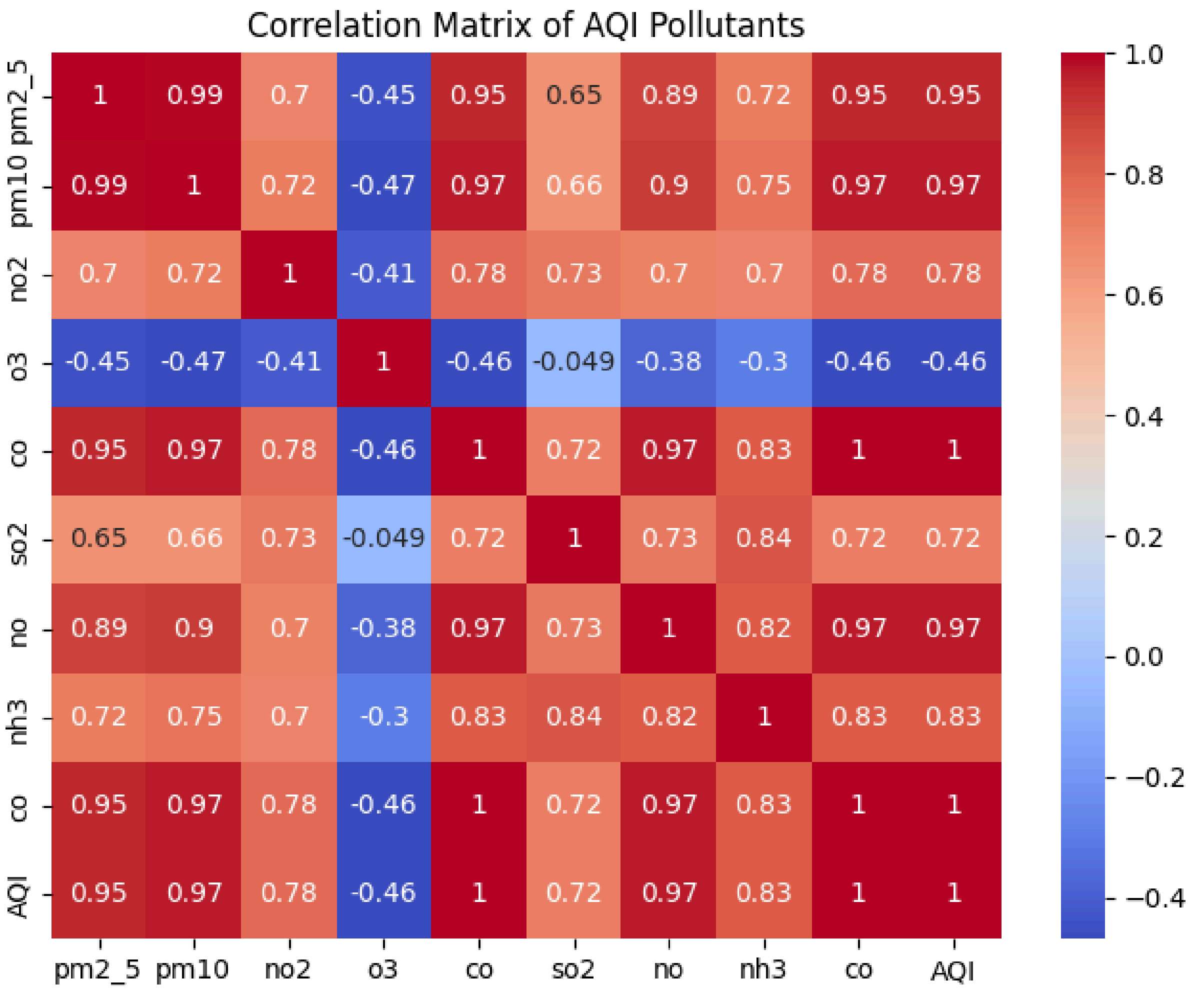


In [26]:

```
plt.figure(figsize=(15, 10))
for i, pollutants in enumerate(['AQI'] + pollutants, 1):
    plt.subplot(2, 4, i)
    sns.boxplot(y=df[pollutant])
    plt.title(f'Boxplot of {pollutant}')
plt.tight_layout()
plt.show()
```



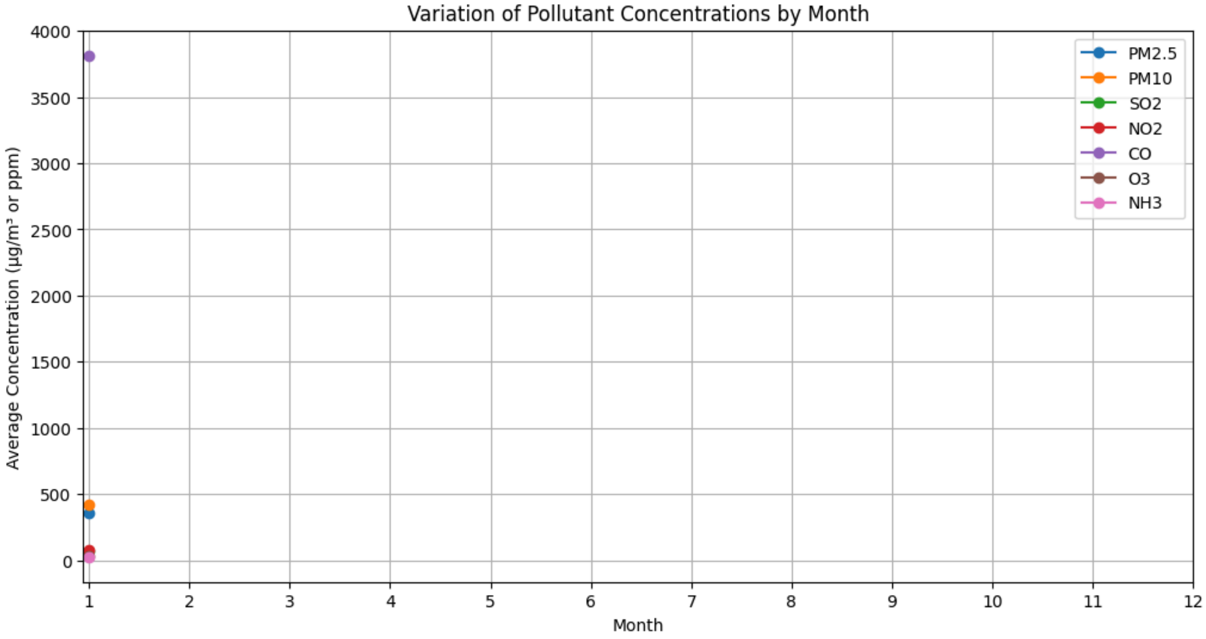
In [27]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of AQI Pollutants')
plt.show()
```



In [28]:
```
avg_concentrations = df[['pm2_5', 'pm10', 'so2', 'no2', 'co', 'o3']].mean()

highest_pollutant = avg_concentrations.idxmax()
highest_concentration = avg_concentrations.max()

print(f"The pollutant with the highest average concentration in Delhi is {highest_p
```

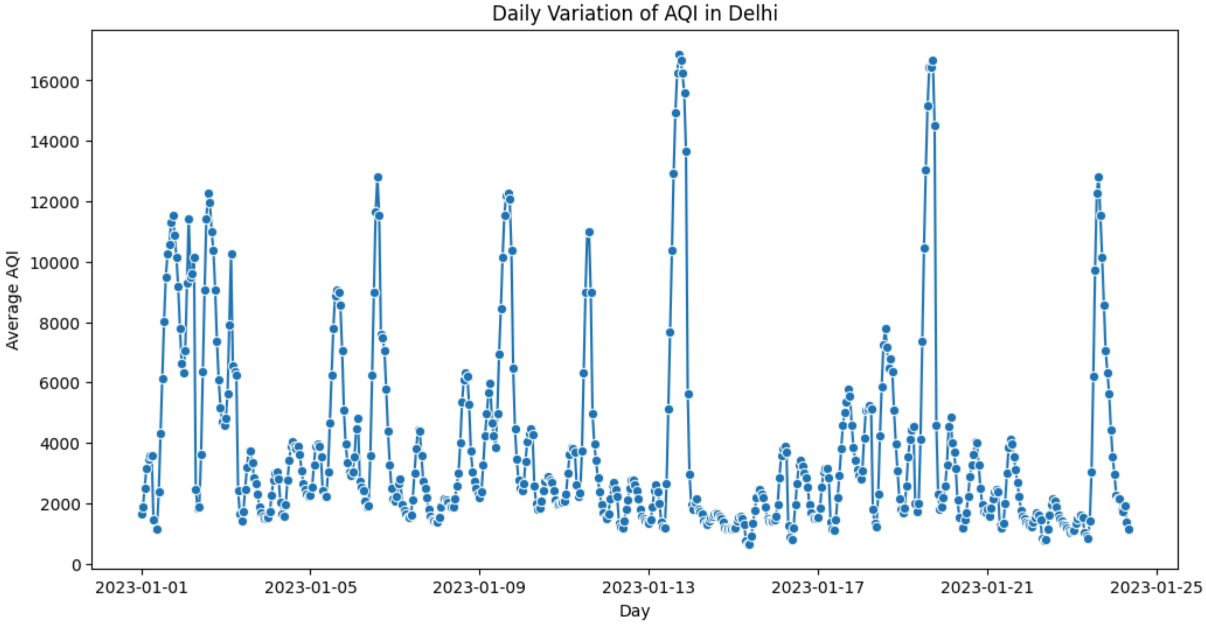
```
The pollutant with the highest average concentration in Delhi is co with an average
concentration of 3814.94 µg/m³ or ppm.
```

In [29]:
```
df['Month'] = pd.to_datetime(df['date']).dt.month

monthly_avg_concentrations = df.groupby('Month').mean()

plt.figure(figsize=(12, 6))
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['pm2_5'], lab
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['pm10'], labe
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['so2'], label
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['no2'], label
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['co'], label=
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['o3'], label=
plt.plot(monthly_avg_concentrations.index, monthly_avg_concentrations['nh3'], label
plt.title('Variation of Pollutant Concentrations by Month')
plt.xlabel('Month')
plt.ylabel('Average Concentration (µg/m³ or ppm)')
plt.xticks(range(1, 13))
plt.grid(True)
plt.legend()
plt.show()
```



In [30]:

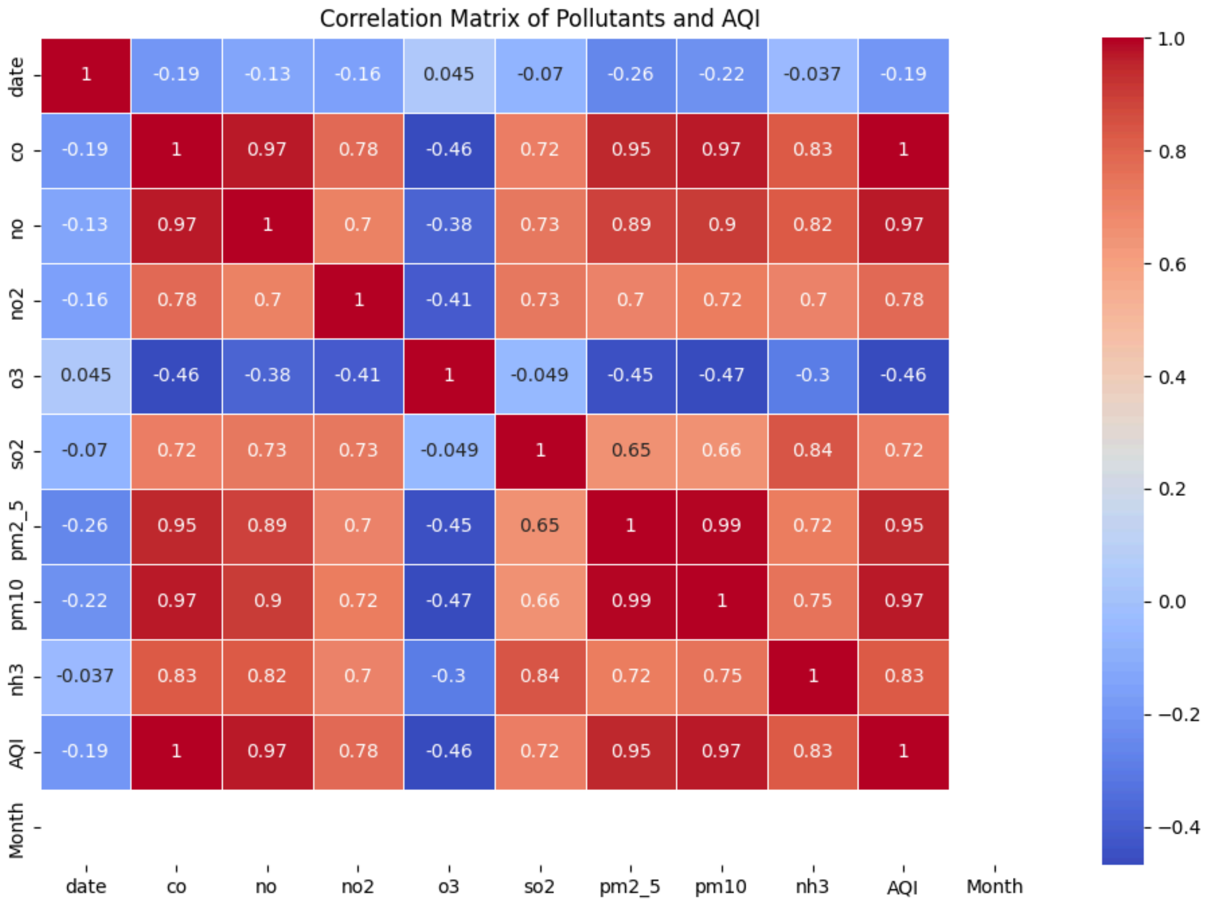
```
Day_aqi = df.groupby('date')['AQI'].mean().reset_index()
plt.figure(figsize=(12, 6))
sns.lineplot(x='date', y='AQI', data=Day_aqi, marker='o')
plt.title('Daily Variation of AQI in Delhi')
plt.xlabel('Day')
plt.ylabel('Average AQI')
plt.show()
```



In [31]:

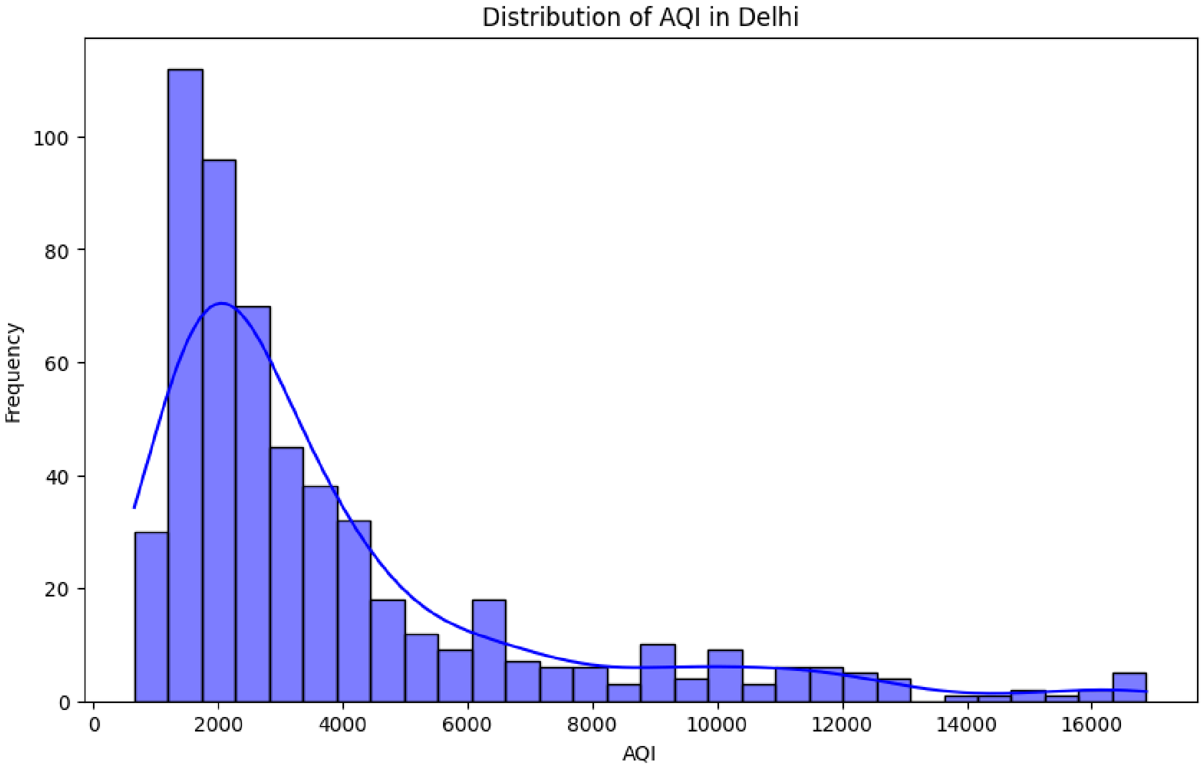
```
correlation_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
```

```
plt.title('Correlation Matrix of Pollutants and AQI')
plt.show()
```



In [32]:

```
plt.figure(figsize=(10, 6))
sns.histplot(df['AQI'], bins=30, kde=True, color='blue')
plt.title('Distribution of AQI in Delhi')
plt.xlabel('AQI')
plt.ylabel('Frequency')
plt.show()
```


Distribution of AQI in Delhi
Frequency
0
20
40
60
80
100
0
2000
4000
6000
8000
10000
12000
14000
16000
AQI

👋 Hi, I'm Ashish Mishra

📊 Internship in Data Science

🏢 at ShadowFox

Cricket Fielding Analysis

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.read_excel(r'C:\Users\Ashish Mishra\OneDrive\Desktop\ShadowFox\Shadowfox_DS
```

In [3]:
```
df.head()
```

Out[3]:

	Pick	Y->	Clean Pick	N->	Fumble	C->	Catch	DC->	Dropped Catch	S->	Stu
0	Throw	Y->	Good Throw	N->	Bad throw	DH->	Dirct Hit	RO->	Run Out	MR->	
1	Runs	"+" stands for runs saved "-" stands for runs ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	Match No.	Innings	Teams	Player Name	BallCount	Position	Pick	Throw	Runs	Ov
4	NaN	IPL2367	1	Delhi Capitals	Rilee russouw	0.1	Short mid wicket	n	NaN	1	

In [4]:
```
# Reload the dataset with a specific skiprows parameter to better clean and structu
data_refined = pd.read_excel(r"C:\Users\Ashish Mishra\OneDrive\Desktop\ShadowFox\Sh

# Rename columns based on their actual meaning and drop any unnecessary columns
data_refined.columns = [
    'Pick',
    'Match_No',
    'Innings',
    'Team',
    'Player_Name',
    'BallCount',
    'Position',
    'Pick_Type',
    'Throw_Type',
    'Runs',
    'Overcount',
    'Venue',
    'Stadium'
]

# Convert numeric fields (e.g., Runs, BallCount) to appropriate data types
data_refined['Runs'] = pd.to_numeric(data_refined['Runs'], errors='coerce').fillna(
data_refined['BallCount'] = pd.to_numeric(data_refined['BallCount'], errors='coerce

# Display the cleaned and structured dataset
data_refined.head()
```

Out[4]:

	Pick	Match_No	Innings	Team	Player_Name	BallCount	Position	Pick_Type	Throw_
0	NaN	Match No.	Innings	Teams	Player Name	NaN	Position	Pick	-
1	NaN	IPL2367	1	Delhi Capitals	Rilee russouw	0.1	Short mid wicket	n	
2	NaN	IPL2367	1	Delhi Capitals	Phil Salt	0.2	wicket keeper	Y	
3	NaN	IPL2367	1	Delhi Capitals	Yash Dhull	0.3	covers	Y	
4	NaN	IPL2367	1	Delhi Capitals	Axer Patel	0.4	point	Y	

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74 entries, 0 to 73
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Pick            2 non-null      object
 1   Y->             31 non-null     object
 2   Clean Pick      54 non-null     object
 3   N->             22 non-null     object
 4   Fumble          19 non-null     object
 5   C->             22 non-null     object
 6   Catch           19 non-null     object
 7   DC->            19 non-null     object
 8   Dropped Catch   16 non-null     object
 9   S->             11 non-null     object
 10  Stumping        22 non-null     object
 11  Unnamed: 11     21 non-null     object
 12  Unnamed: 12     13 non-null     object
dtypes: object(13)
memory usage: 7.6+ KB
```

In [6]:

```
# Drop rows where essential columns like 'Player_Name' or 'Position' are NaN
data_refined.dropna(subset=['Player_Name', 'Position'], inplace=True)

# Convert numeric fields (e.g., Runs, BallCount) to appropriate data types
data_refined['Runs'] = pd.to_numeric(data_refined['Runs'], errors='coerce').fillna(
data_refined['BallCount'] = pd.to_numeric(data_refined['BallCount'], errors='coerce
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74 entries, 0 to 73
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Pick            2 non-null      object
 1   Y->             31 non-null     object
 2   Clean Pick      54 non-null     object
 3   N->             22 non-null     object
 4   Fumble          19 non-null     object
 5   C->             22 non-null     object
 6   Catch           19 non-null     object
 7   DC->            19 non-null     object
 8   Dropped Catch   16 non-null     object
 9   S->             11 non-null     object
 10  Stumping        22 non-null     object
 11  Unnamed: 11     21 non-null     object
 12  Unnamed: 12     13 non-null     object
dtypes: object(13)
memory usage: 7.6+ KB
```

In [8]:

```
weights = {
    'CP': 1.5,  # Clean Picks
    'GT': 1.2,  # Good Throws
    'C': 2.0,   # Catches
    'DC': -1.0, # Dropped Catches
    'ST': 2.5,  # Stumpings
    'RO': 3.0,  # Run Outs
    'MRO': -0.5, # Missed Run Outs
    'DH': 2.0,  # Direct Hits
    'RS': 1.0   # Runs Saved
}
```

In [9]:

```
# Initialize a performance score column
data_refined['Performance Score'] = 0

# Calculate the performance score for each player
data_refined['Performance Score'] = (
    data_refined['Pick_Type'].apply(lambda x: weights['CP'] if x == 'Y' else 0) +
    data_refined['Throw_Type'].apply(lambda x: weights['GT'] if x == 'Y' else 0) +
    data_refined['Runs'] * weights['RS']
)

# Group by player and calculate total performance scores
top_players = data_refined[['Player_Name', 'Performance Score', 'Match_No', 'Inning
top_players_grouped = top_players.groupby('Player_Name').agg({
    'Performance Score': 'sum',
    'Match_No': 'first',
    'Innings': 'first',
    'Team': 'first',
    'Position': 'first',
    'Venue': 'first'
}).sort_values(by='Performance Score', ascending=False)

# Display top performers
top_3_players = top_players_grouped.head(3)
print("Top 3 Performers with Details:\n", top_3_players)
```

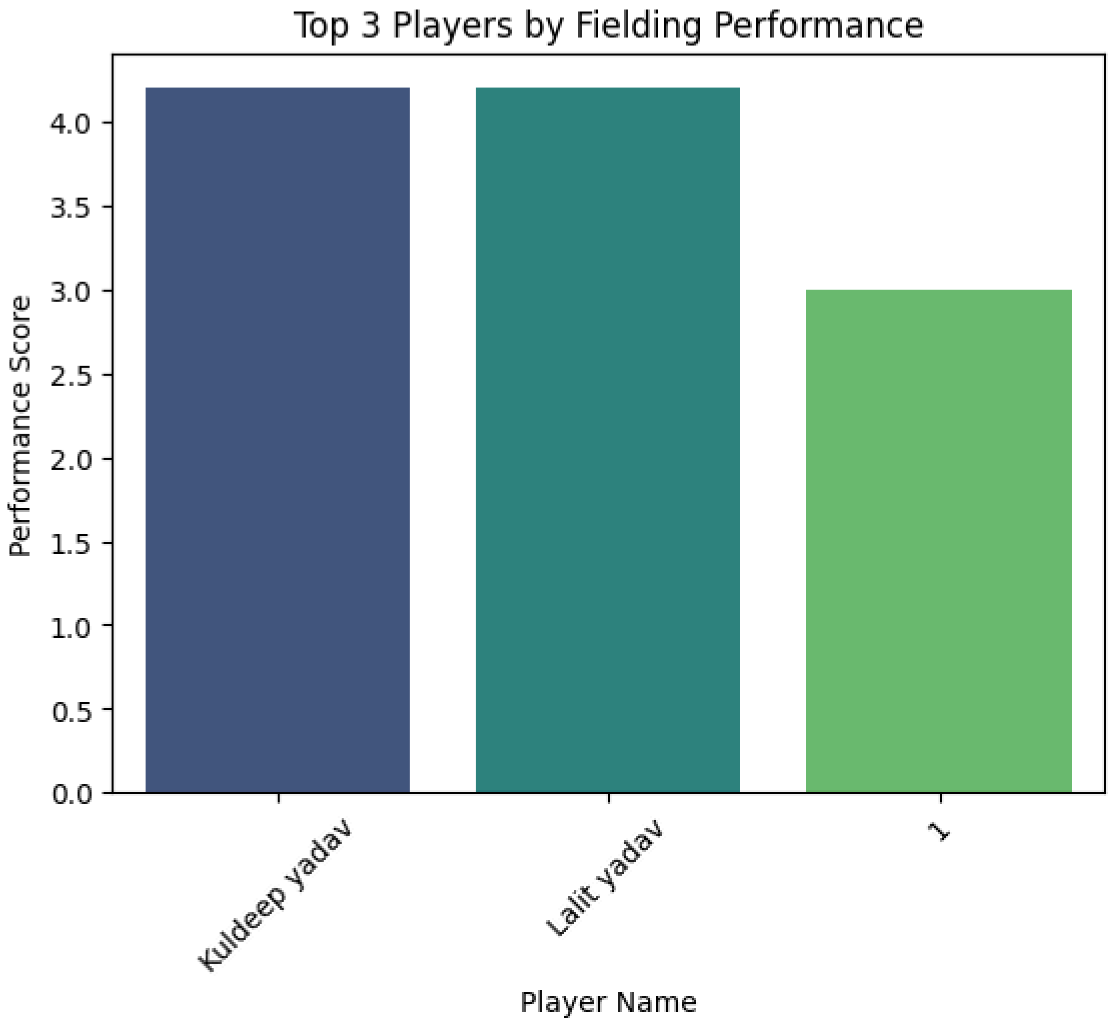
```
Top 3 Performers with Details:
                 Performance Score       Match_No Innings            Team  \
Player_Name
Kuldeep yadav                4.2        IPL2367       1  Delhi Capitals
Lalit yadav                  4.2        IPL2367       1  Delhi Capitals
1                            3.0  Rilee russouw       2               1

                       Position  Venue
Player_Name
Kuldeep yadav  Short mid wicket  Delhi
Lalit yadav         cover point  Delhi
1                             0     10
```

In [10]:

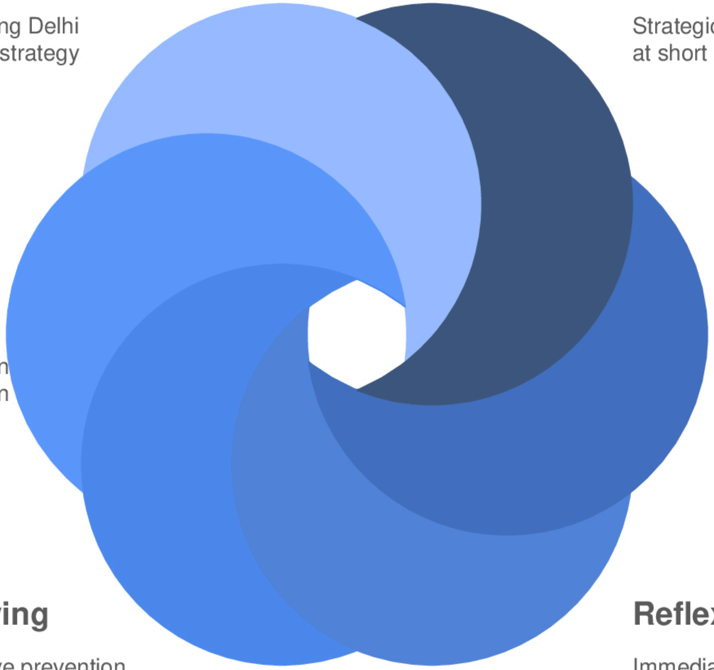
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [11]:

```
sns.barplot(x=top_3_players.index, y=top_3_players['Performance Score'], hue=top_3_
plt.title('Top 3 Players by Fielding Performance')
plt.xlabel('Player Name')
plt.ylabel('Performance Score')
plt.xticks(rotation=45)
plt.show()
```


Top 3 Players by Fielding Performance
Performance Score
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Kuldeep yadav
Lalit yadav
1
Player Name

Kuldeep Yadav's Fielding Excellence



Team Contribution

Supporting Delhi Capitals' strategy

Positioning

Strategic placement at short mid-wicket

Pressure Maintenance

Keeping pressure on the opposition

Awareness

Quick recognition of plays

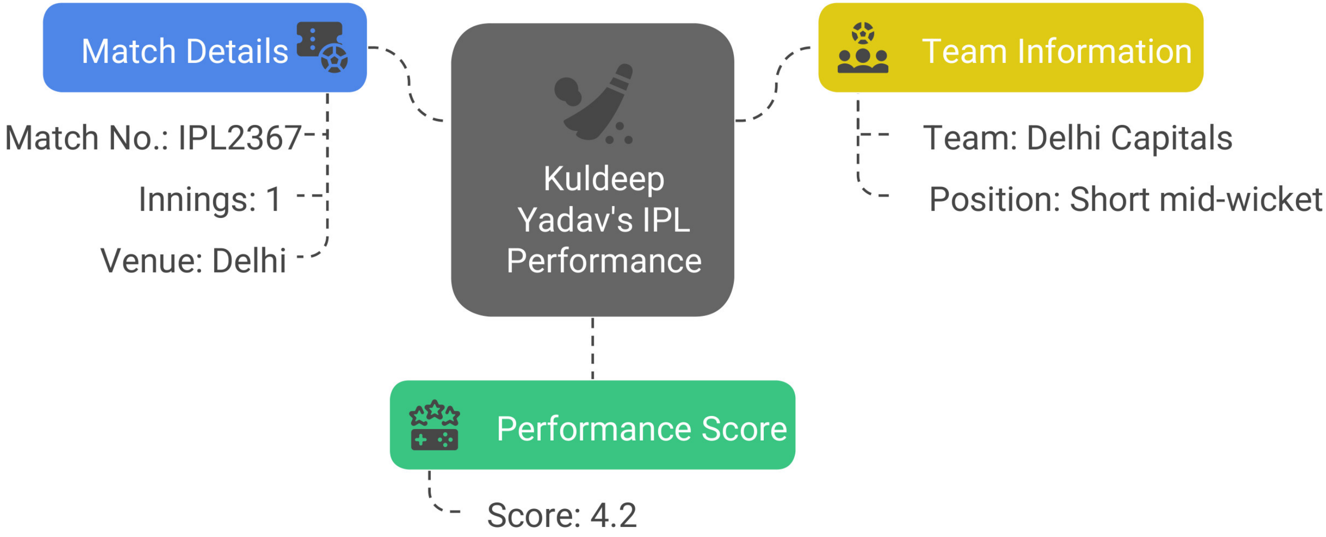
Run Saving

Effective prevention of runs

Reflexes

Immediate response to actions

Kuldeep Yadav's IPL Performance Overview



Kuldeep Yadav's IPL Performance

Match Details

Match No.: IPL2367

Innings: 1

Venue: Delhi

Team Information

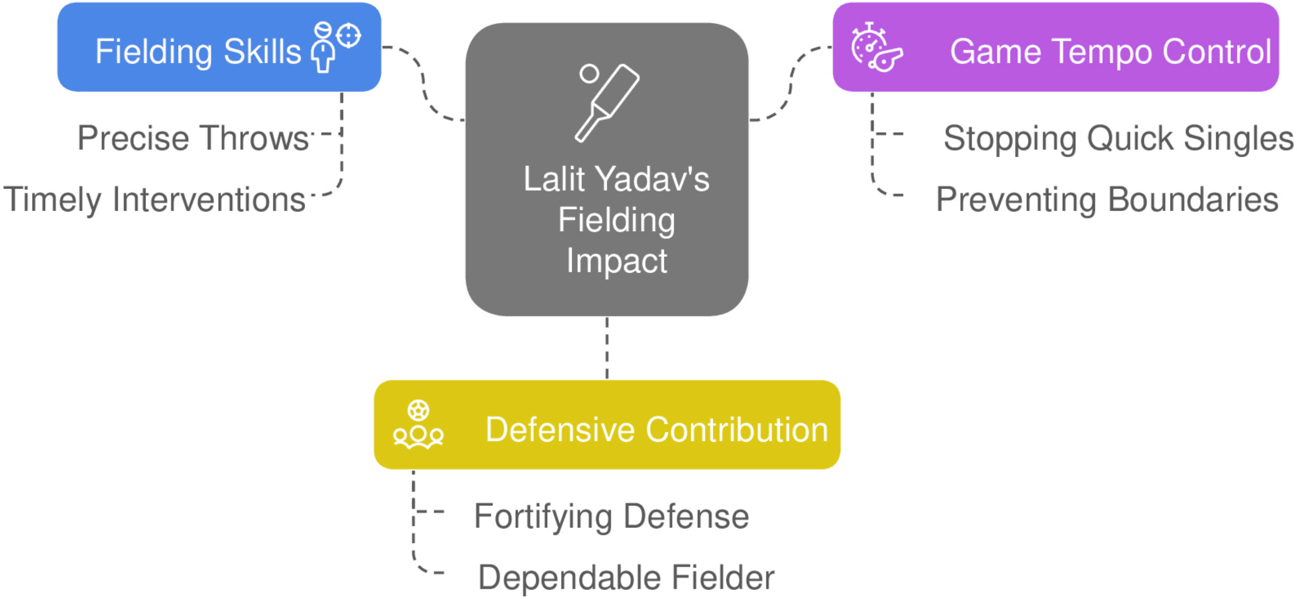
Team: Delhi Capitals

Position: Short mid-wicket

Performance Score

Score: 4.2

Lalit Yadav's Impact on Delhi Capitals' Defense



Lalit Yadav's Fielding Impact

Fielding Skills

Precise Throws

Timely Interventions

Game Tempo Control

Stopping Quick Singles

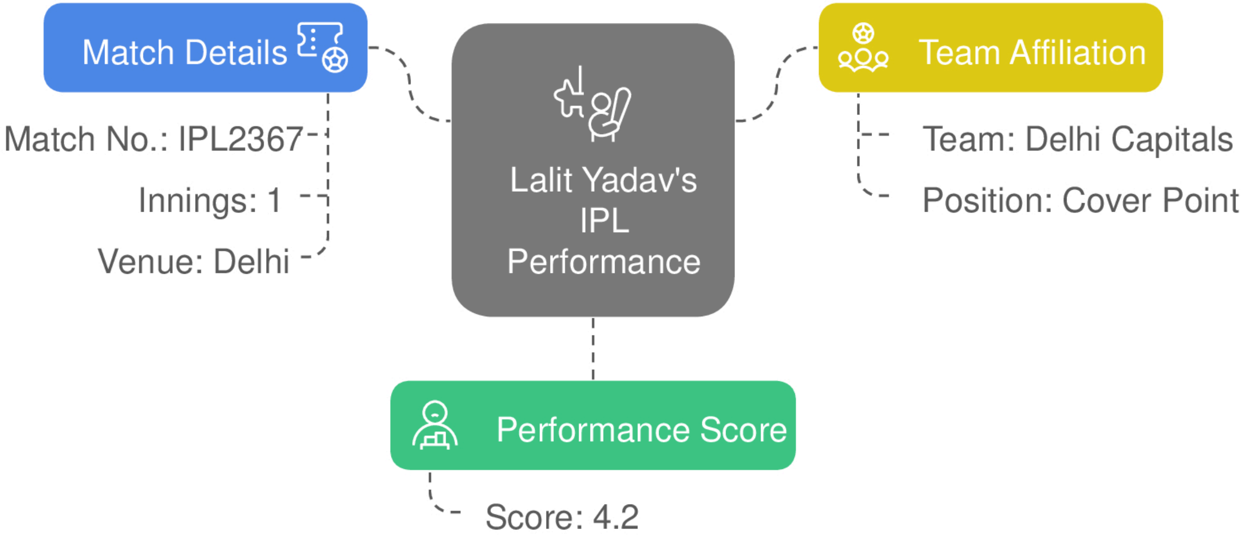
Preventing Boundaries

Defensive Contribution

Fortifying Defense

Dependable Fielder

Lalit Yadav's IPL Performance Overview



Lalit Yadav's IPL Performance

Match Details

Match No.: IPL2367

Innings: 1

Venue: Delhi

Team Affiliation

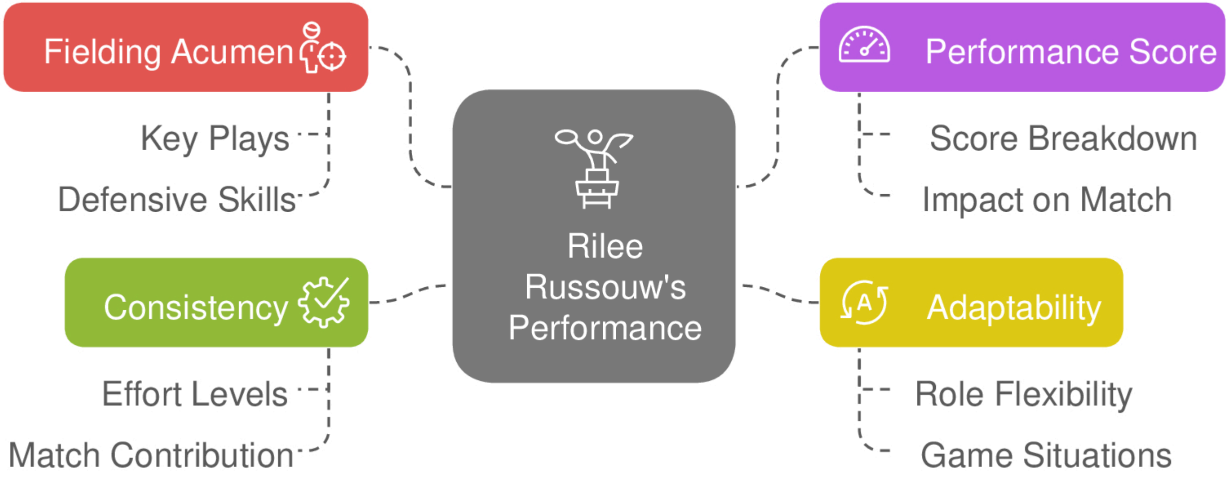
Team: Delhi Capitals

Position: Cover Point

Performance Score

Score: 4.2

Rilee Russouw's Impactful Performance



Rilee Russouw's Performance

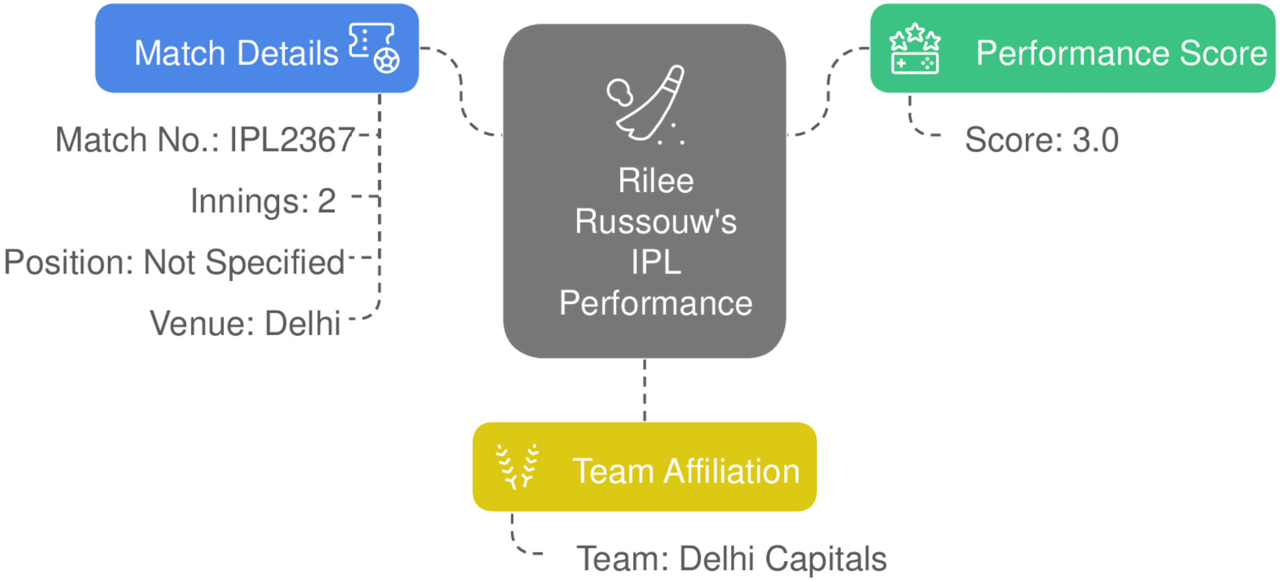
Fielding Acumen
- Key Plays
- Defensive Skills

Consistency
- Effort Levels
- Match Contribution

Performance Score
- Score Breakdown
- Impact on Match

Adaptability
- Role Flexibility
- Game Situations

Rilee Russouw's IPL Performance Overview



Rilee Russouw's IPL Performance

Match Details

- Match No.: IPL2367
- Innings: 2
- Position: Not Specified
- Venue: Delhi

Performance Score

- Score: 3.0

Team Affiliation

- Team: Delhi Capitals