Air Quality Index (AQI) Statistical Analysis - Pune

This notebook analyzes the Air Quality Index (AQI) dataset for Pune to apply various statistical techniques as required for the mini-project.

Installing dependencies

First, let's install the required packages for our analysis.

In [ ]:
```
%pip install -r requirements.txt
```

1. Loading and Exploring the Dataset

First, let's load the dataset and explore its structure. We're using a Pune Air Quality dataset with over 14,000 records, which meets our minimum requirement.

Dataset Columns:

- Date: The date of the recorded observation (non-null, string).
- SO2: Sulfur dioxide concentration (one missing value, string).
- NOx: Nitrogen oxides concentration (non-null, string).
- RSPM: Respirable suspended particulate matter (35 missing values, float).
- AQI: Air Quality Index (non-null, float).
- Area: Name of the area within Pune where the observation was recorded (non-null, string).

In [3]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import zscore, ttest_ind, spearmanr
import plotly.express as px
import statsmodels.api as sm
import kagglehub

# Set up visualization style - Fix for the deprecated seaborn-whitegrid style
plt.style.use('seaborn-v0_8-whitegrid')  # Updated style name
sns.set_palette('viridis')
%matplotlib inline
plt.rcParams['figure.figsize'] = (12, 8)
```

In [4]:
```
# Download the new dataset using kagglehub (with more records)
try:
    path = kagglehub.dataset_download("tejasnarkhede03/pune-air-quality-index")
    print(f"Dataset downloaded to: {path}")
except Exception as e:
    print(f"Error downloading dataset: {e}")
    print("If the automatic download fails, please download the dataset manually from Kaggle a
```

```
Dataset downloaded to: /home/codespace/.cache/kagglehub/datasets/tejasnarkhede03/pune-air-qual
ity-index/versions/1
```

In [5]:
```
# Load the dataset
# Note: Adjust the file path if needed after downloading
import os
```

```python
# Check what files are in the downloaded directory
if os.path.exists(path):
    print("Files in the dataset directory:")
    for file in os.listdir(path):
        print(f" - {file}")

# Load the CSV file (adjust filename if needed)
try:
    # Check for CSV files and load the first one
    csv_files = [f for f in os.listdir(path) if f.endswith('.csv')]
    if csv_files:
        file = csv_files[0]  # Choose the first CSV file
        df = pd.read_csv(os.path.join(path, file))
        print(f"Loaded dataset: {file}")
        print(f"Number of records: {len(df)}")
    else:
        print("No CSV files found in the dataset directory.")
except Exception as e:
    print(f"Error loading dataset: {e}")
```

```
Files in the dataset directory:
 - Pune Air Quality Index Dataset.csv
Loaded dataset: Pune Air Quality Index Dataset.csv
Number of records: 14873
```

In [6]:

```python
# Explore the dataset
print(f"Dataset shape: {df.shape}")
print(f"\nDataset info:")
df.info()
print(f"\nFirst 5 rows:")
df.head()
```

```
Dataset shape: (14873, 6)

Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14873 entries, 0 to 14872
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    14873 non-null  object 
 1   SO2     14872 non-null  object 
 2   NOx     14873 non-null  object 
 3   RSPM    14838 non-null  float64
 4   AQI     14873 non-null  float64
 5   Area    14873 non-null  object 
dtypes: float64(2), object(4)
memory usage: 697.3+ KB

First 5 rows:
```

Out[6]:

	Date	SO2	NOx	RSPM	AQI	Area
0	29-08-2005	20	38	84.0	84.0	Pimpri-Chinchwad
1	30-08-2005	21	43	94.0	94.0	Pimpri-Chinchwad
2	01-09-2005	17	35	74.0	74.0	Pimpri-Chinchwad
3	02-09-2005	15	28	68.0	68.0	Pimpri-Chinchwad
4	04-09-2005	17	31	72.0	72.0	Pimpri-Chinchwad

In [7]:

```python
# Check for and handle missing values
print("Missing values by column:")
print(df.isnull().sum())
```

```python
# Convert string columns to numeric if they contain numeric data
# This dataset might have numeric values stored as strings
for col in df.columns:
    if df[col].dtype == 'object' and col != 'Date' and col != 'Area':
        try:
            # Try to convert to numeric, coercing errors to NaN
            df[col] = pd.to_numeric(df[col], errors='coerce')
            print(f"Converted {col} to numeric type")
        except Exception as e:
            print(f"Could not convert {col}: {e}")

# Handle missing values
df = df.dropna()
print(f"\nDataset shape after handling missing values: {df.shape}")
```

```
Missing values by column:
Date     0
SO2      1
NOx      0
RSPM    35
AQI      0
Area     0
dtype: int64
Converted SO2 to numeric type
Converted NOx to numeric type

Dataset shape after handling missing values: (14547, 6)
```

In [8]:

```python
# Fix Area names as requested
print("Unique areas before correction:")
print(df['Area'].unique())

# Replace 'Karvegar' with 'Karvenagar' and 'l Stop' with 'Other'
area_replacements = {
    'Karvegar': 'Karvenagar',
    'l Stop': 'Other'
}

df['Area'] = df['Area'].replace(area_replacements)

print("\nUnique areas after correction:")
print(df['Area'].unique())

# Count records by Area to verify changes
area_counts = df['Area'].value_counts()
print("\nRecords by Area:")
print(area_counts)
```

```
Unique areas before correction:
['Pimpri-Chinchwad' 'Karvegar' 'l Stop' 'Bhosari' 'Swargate']

Unique areas after correction:
['Pimpri-Chinchwad' 'Karvenagar' 'Other' 'Bhosari' 'Swargate']

Records by Area:
Area
Karvenagar          5043
Pimpri-Chinchwad    4666
Other               1687
Bhosari             1603
Swargate            1548
Name: count, dtype: int64
```

In [9]:

```python
# Data summary statistics
df.describe()
```

Out[9]:

	SO2	NOx	RSPM	AQI
count	14547.000000	14547.000000	14547.000000	14547.000000
mean	21.594006	50.940194	99.030865	98.011892
std	13.251183	29.370449	57.242143	46.078003
min	4.000000	9.000000	4.000000	11.000000
25%	13.000000	32.000000	54.000000	63.000000
50%	20.000000	46.000000	91.000000	96.000000
75%	27.000000	64.000000	135.000000	127.000000
max	525.000000	896.000000	801.000000	864.000000

In [ ]:

```python
# Add a date index for time-based analysis
try:
    # Convert the Date column to datetime
    df['Date'] = pd.to_datetime(df['Date'])

    # Extract year, month for additional analysis
    df['Year'] = df['Date'].dt.year
    df['Month'] = df['Date'].dt.month

    print("Date column successfully converted to datetime format")
    print(f"Date range: {df['Date'].min()} to {df['Date'].max()}")
except Exception as e:
    print(f"Error converting date: {e}")
```

2. Central Limit Theorem Demonstration

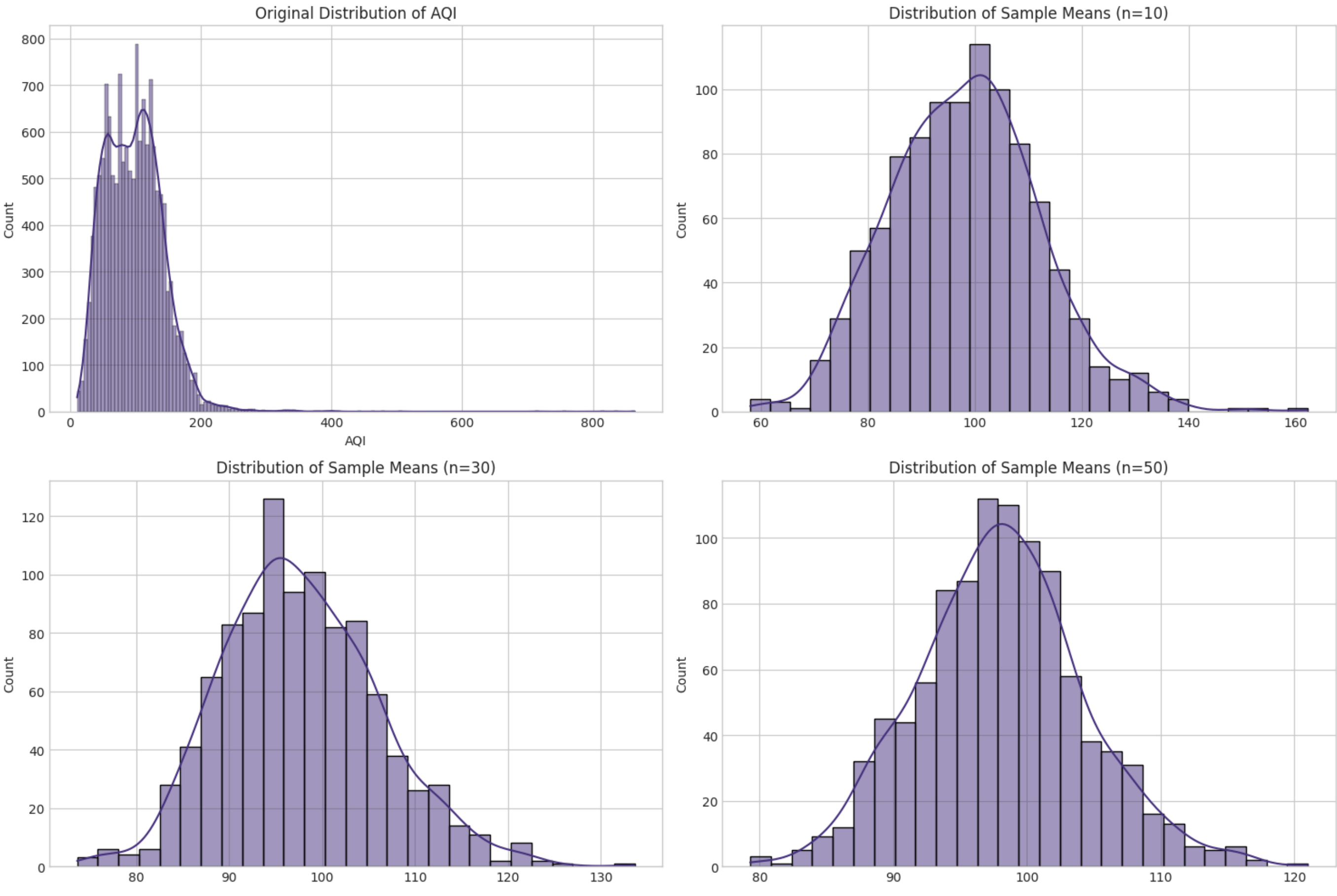
The Central Limit Theorem (CLT) states that the sampling distribution of the mean approaches a normal distribution as the sample size gets larger, regardless of the population's distribution.

In [11]:

```python
# Select a numeric column for demonstration (AQI is a good candidate)
numeric_column = 'AQI'  # Using AQI column
print(f"Demonstrating CLT using column: {numeric_column}")

# Original distribution
plt.figure(figsize=(15, 10))
plt.subplot(2, 2, 1)
sns.histplot(df[numeric_column], kde=True)
plt.title(f'Original Distribution of {numeric_column}')

# Sample means for different sample sizes
sample_sizes = [10, 30, 50]
num_samples = 1000

for i, size in enumerate(sample_sizes):
    sample_means = [df[numeric_column].sample(size).mean() for _ in range(num_samples)]

    plt.subplot(2, 2, i+2)
    sns.histplot(sample_means, kde=True)
    plt.title(f'Distribution of Sample Means (n={size})')

    # Calculate and display statistics
    sample_mean = np.mean(sample_means)
    sample_std = np.std(sample_means)
    expected_std = df[numeric_column].std() / np.sqrt(size)  # Expected standard error

    print(f"Sample Size {size}:")
```

```
    print(f"Mean of sample means: {sample_mean:.2f}")
    print(f"Standard deviation of sample means: {sample_std:.2f}")
    print(f"Expected standard error: {expected_std:.2f}")
    print()

plt.tight_layout()
plt.show()
```

```
Demonstrating CLT using column: AQI
Sample Size 10:
Mean of sample means: 98.27
Standard deviation of sample means: 14.04
Expected standard error: 14.57

Sample Size 30:
Mean of sample means: 97.70
Standard deviation of sample means: 8.44
Expected standard error: 8.41

Sample Size 50:
Mean of sample means: 98.05
Standard deviation of sample means: 6.21
Expected standard error: 6.52
```

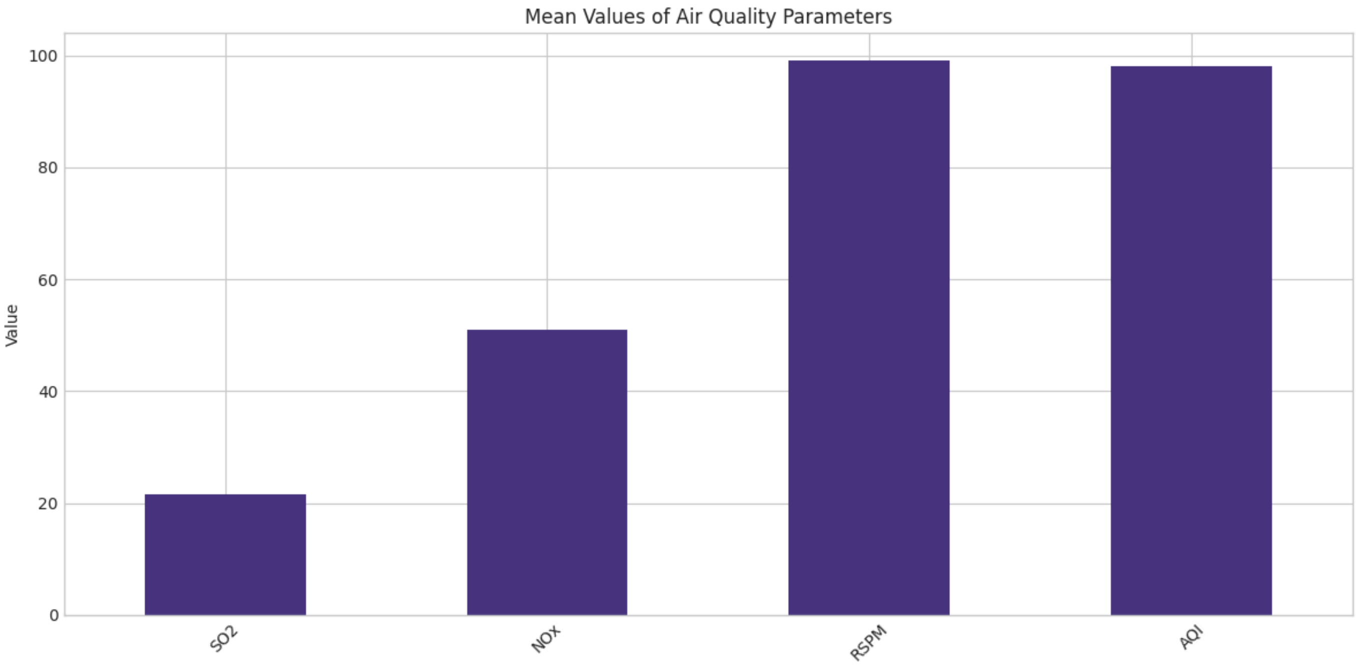


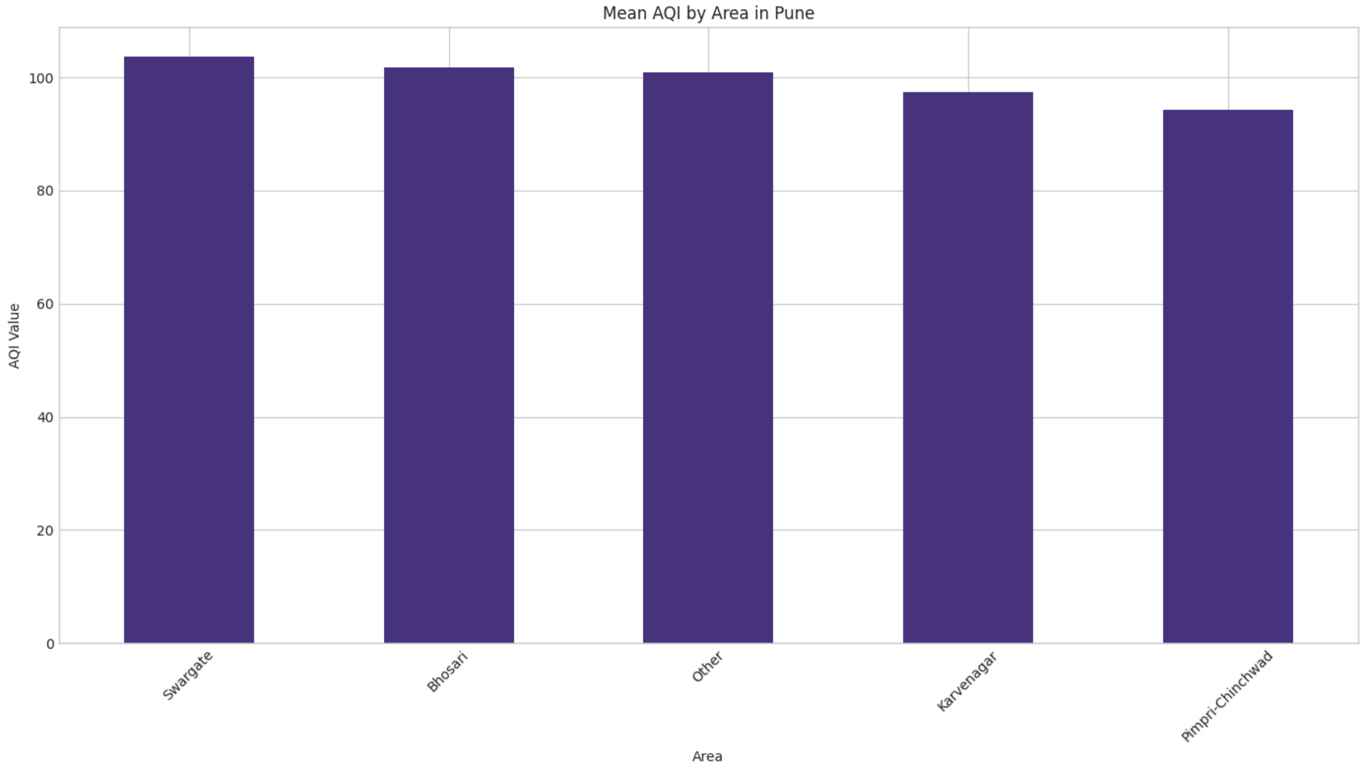
3. Mean Analysis

In [12]:
```
# Select relevant numeric columns for analysis
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
# Remove Year and Month from numeric analysis if they exist
if 'Year' in numeric_cols:
    numeric_cols.remove('Year')
if 'Month' in numeric_cols:
    numeric_cols.remove('Month')

print(f"Analyzing numeric columns: {numeric_cols}")

# Mean values
```

```python
means = df[numeric_cols].mean()
print("Mean values for each parameter:")
print(means)

# Visualize means
plt.figure(figsize=(12, 6))
means.plot(kind='bar')
plt.title('Mean Values of Air Quality Parameters')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Mean AQI by area if 'Area' column exists
if 'Area' in df.columns:
    area_means = df.groupby('Area')['AQI'].mean().sort_values(ascending=False)
    plt.figure(figsize=(14, 8))
    area_means.plot(kind='bar')
    plt.title('Mean AQI by Area in Pune')
    plt.ylabel('AQI Value')
    plt.xlabel('Area')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
Analyzing numeric columns: ['SO2', 'NOx', 'RSPM', 'AQI']
Mean values for each parameter:
SO2     21.594006
NOx     50.940194
RSPM    99.030865
AQI     98.011892
dtype: float64
```



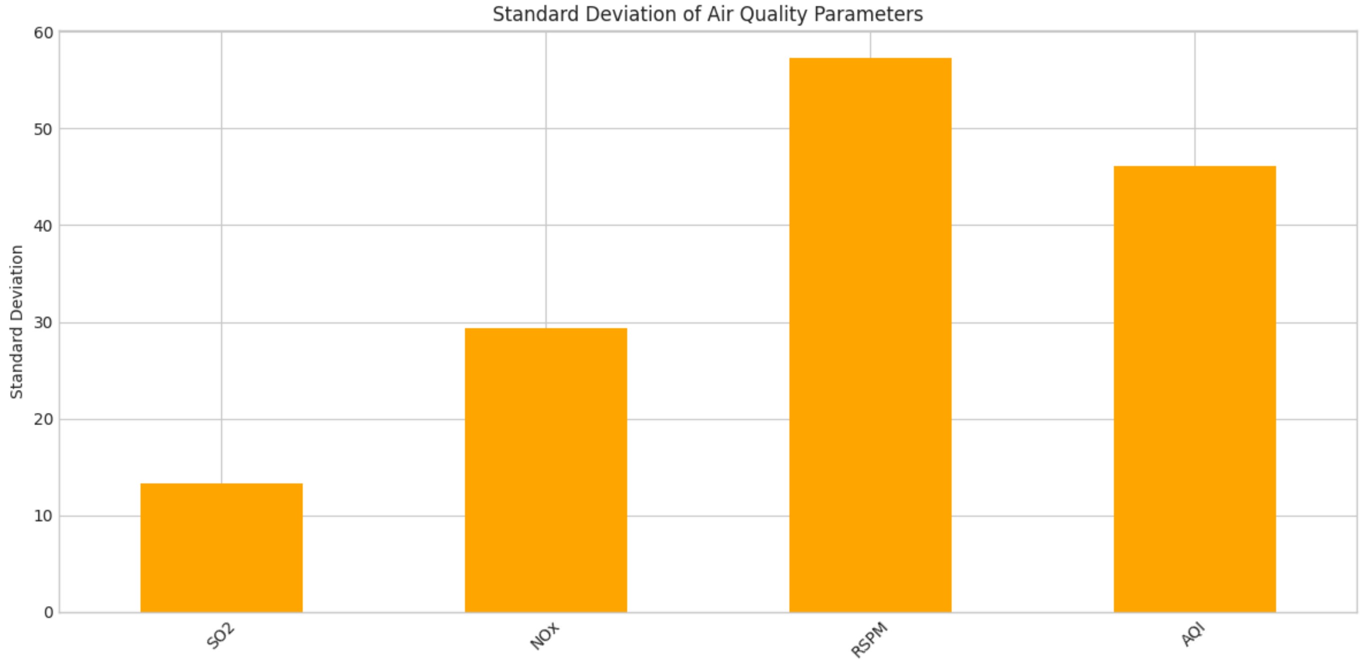


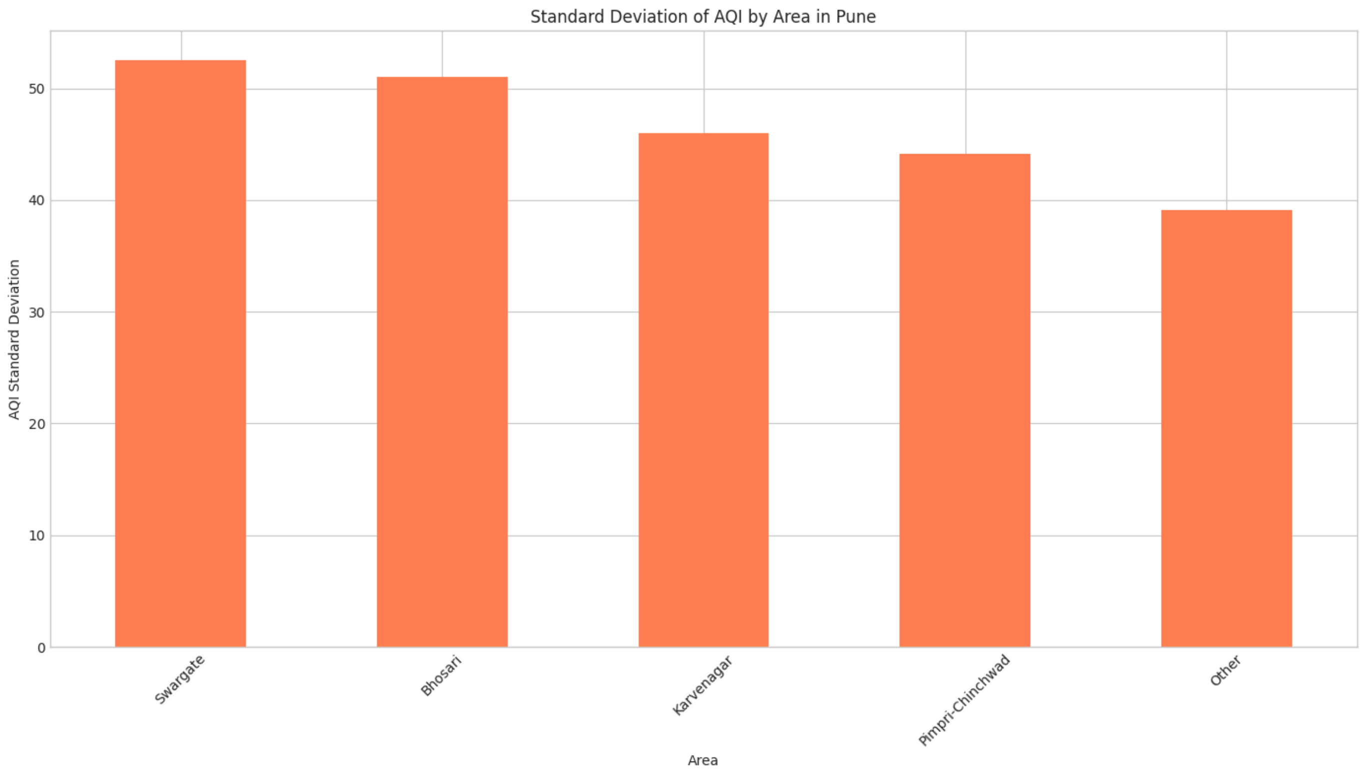
4. Variance and Standard Deviation

In [13]:

```
# Calculate variance and standard deviation for each numeric column
variance = df[numeric_cols].var()
std_dev = df[numeric_cols].std()

# Display results
print("Variance for each parameter:")
print(variance)
print("\nStandard Deviation for each parameter:")
print(std_dev)

# Visualize standard deviation
plt.figure(figsize=(12, 6))
std_dev.plot(kind='bar', color='orange')
plt.title('Standard Deviation of Air Quality Parameters')
plt.ylabel('Standard Deviation')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Standard deviation of AQI by area if 'Area' column exists
if 'Area' in df.columns:
    area_std = df.groupby('Area')['AQI'].std().sort_values(ascending=False)
    plt.figure(figsize=(14, 8))
    area_std.plot(kind='bar', color='coral')
    plt.title('Standard Deviation of AQI by Area in Pune')
    plt.ylabel('AQI Standard Deviation')
    plt.xlabel('Area')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
Variance for each parameter:
SO2      175.593854
NOx      862.623262
RSPM    3276.662941
AQI     2123.182383
dtype: float64

Standard Deviation for each parameter:
SO2     13.251183
NOx     29.370449
RSPM    57.242143
AQI     46.078003
dtype: float64
```



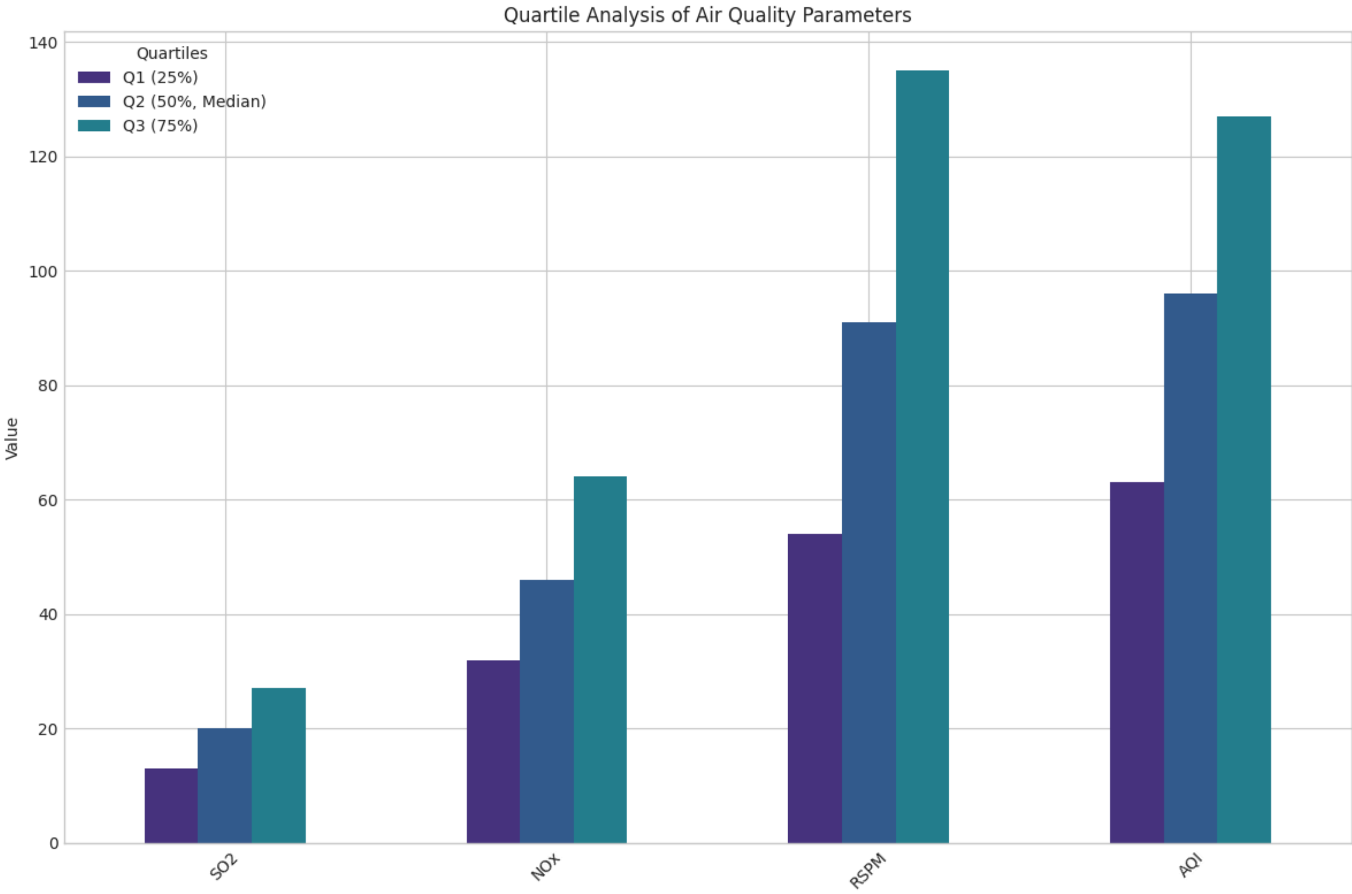


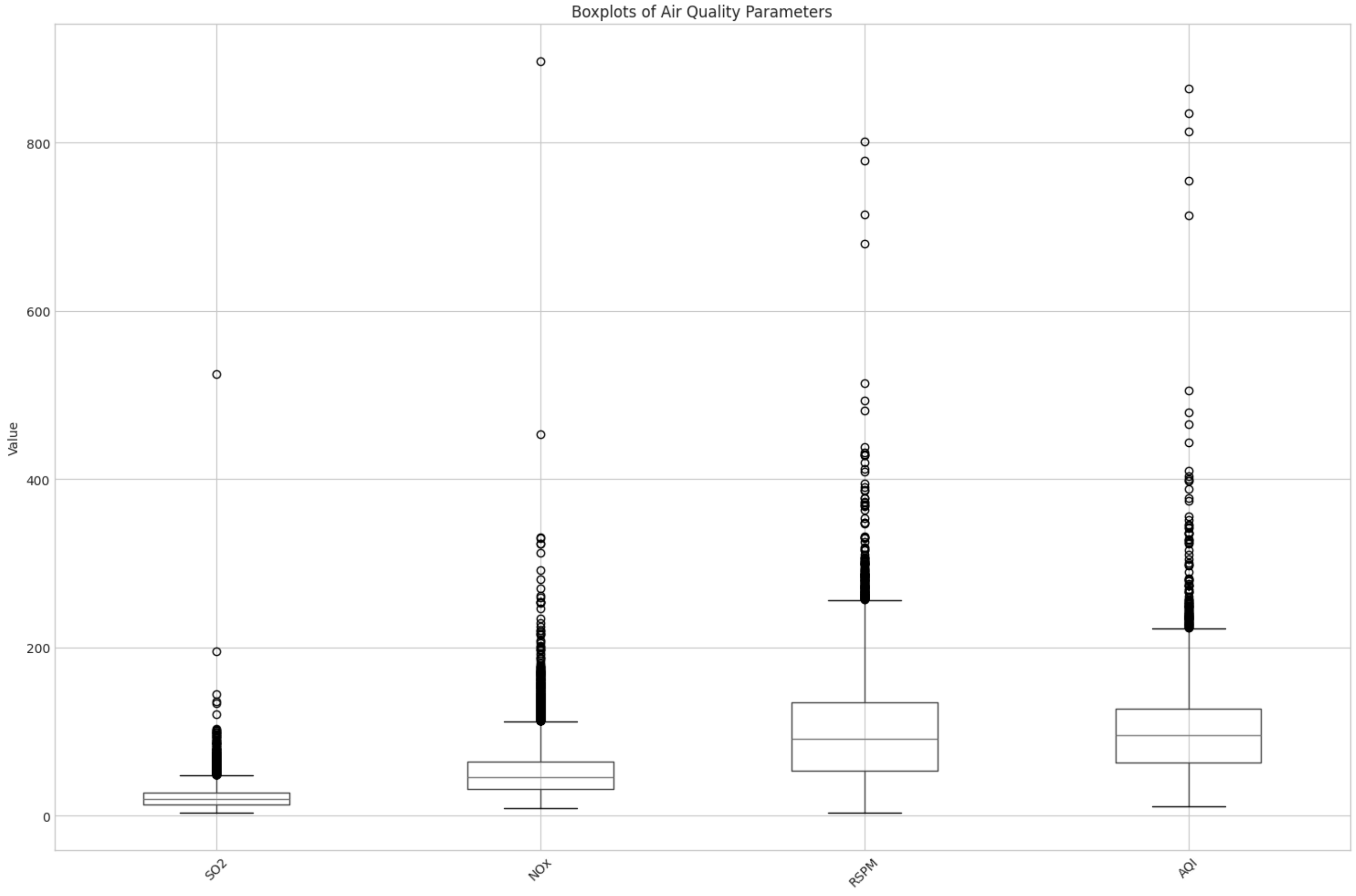
5. Quartile Analysis (Q1, Q2, Q3)

In [14]:
```
# Calculate quartiles
q1 = df[numeric_cols].quantile(0.25)
q2 = df[numeric_cols].quantile(0.50)  # Median
q3 = df[numeric_cols].quantile(0.75)
```

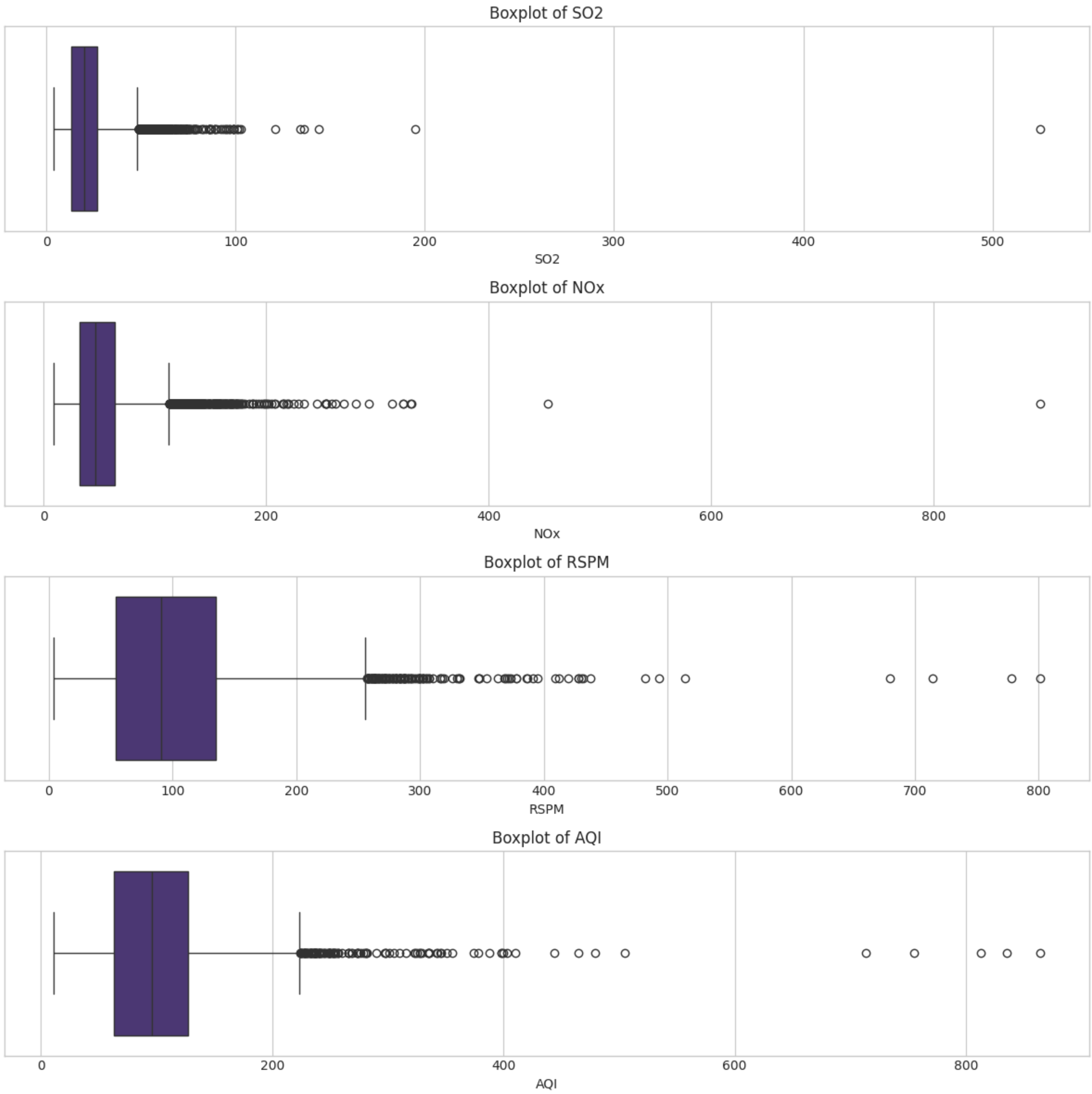
```python
# Display results
print("First Quartile (Q1, 25%):")
print(q1)
print("\nSecond Quartile (Q2, 50%, Median):")
print(q2)
print("\nThird Quartile (Q3, 75%):")
print(q3)

# Create a DataFrame for quartile visualization
quartiles_df = pd.DataFrame({
    'Q1 (25%)': q1,
    'Q2 (50%, Median)': q2,
    'Q3 (75%)': q3
})

# Plot quartiles for each parameter
plt.figure(figsize=(14, 8))
quartiles_df.plot(kind='bar')
plt.title('Quartile Analysis of Air Quality Parameters')
plt.ylabel('Value')
plt.legend(title='Quartiles')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Create boxplots for each numeric parameter
plt.figure(figsize=(15, 10))
df[numeric_cols].boxplot()
plt.title('Boxplots of Air Quality Parameters')
plt.ylabel('Value')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Create individual boxplots for better visualization
fig, axes = plt.subplots(nrows=len(numeric_cols), figsize=(12, 3*len(numeric_cols)))
for i, col in enumerate(numeric_cols):
    sns.boxplot(x=df[col], ax=axes[i])
    axes[i].set_title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()

# Boxplot of AQI by area if 'Area' column exists
if 'Area' in df.columns:
    plt.figure(figsize=(16, 10))
    sns.boxplot(x='Area', y='AQI', data=df)
    plt.title('AQI Distribution by Area')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

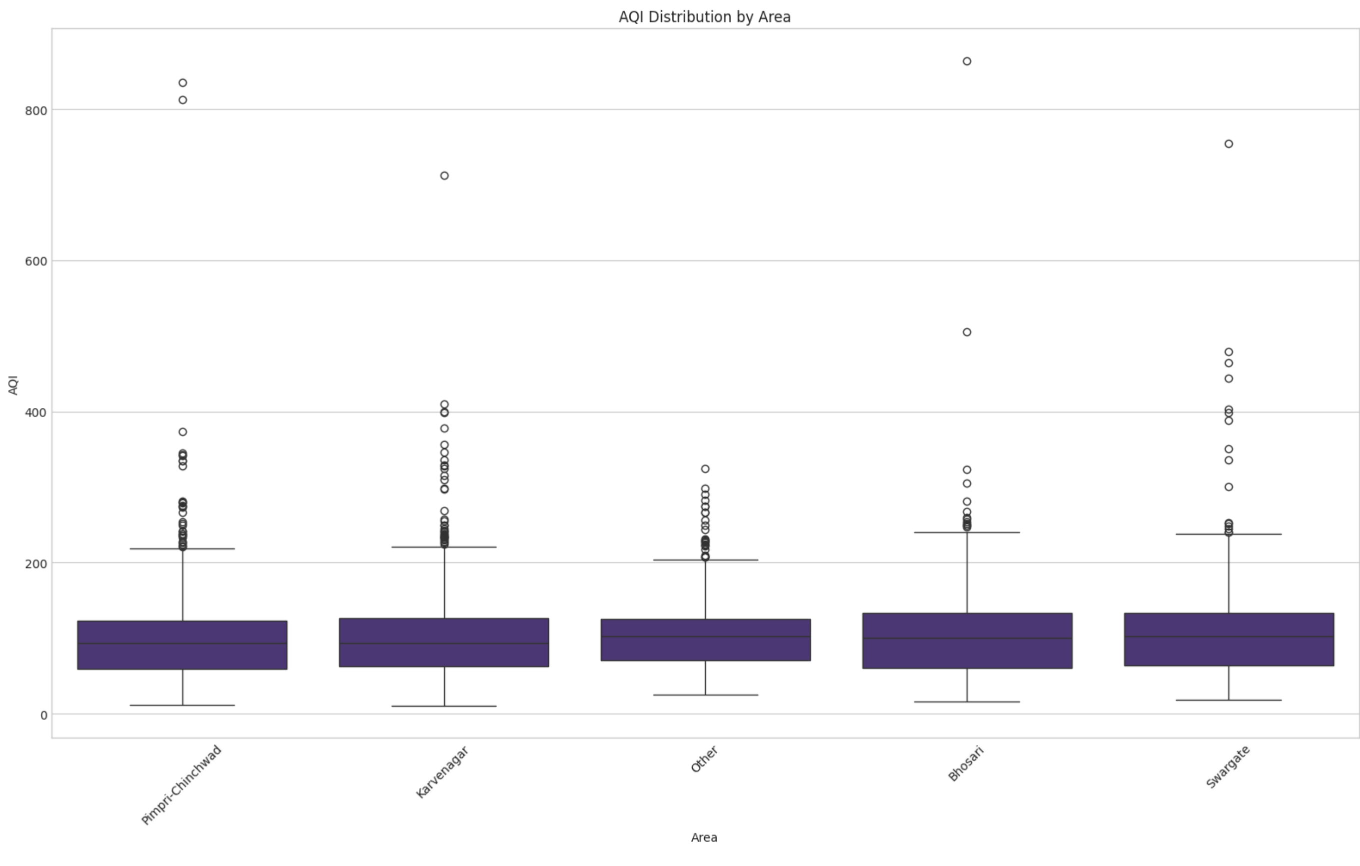
```
First Quartile (Q1, 25%):
SO2      13.0
NOx      32.0
RSPM     54.0
AQI      63.0
Name: 0.25, dtype: float64

Second Quartile (Q2, 50%, Median):
SO2      20.0
NOx      46.0
RSPM     91.0
AQI      96.0
Name: 0.5, dtype: float64

Third Quartile (Q3, 75%):
SO2       27.0
NOx       64.0
RSPM     135.0
AQI      127.0
Name: 0.75, dtype: float64
<Figure size 1400x800 with 0 Axes>
```





Boxplots of Air Quality Parameters
Value
800
600
400
200
0
SO2
NOx
RSPM
AQI


Boxplot of SO2
SO2
0
100
200
300
400
500
Boxplot of NOx
NOx
0
200
400
600
800
Boxplot of RSPM
RSPM
0
100
200
300
400
500
600
700
800
Boxplot of AQI
AQI
0
200
400
600
800


AQI Distribution by Area
AQI
0
200
400
600
800
Area
Pimpri-Chinchwad
Karvenagar
Other
Bhosari
Swargate

6. Weighted Mean

For weighted mean, we need to determine appropriate weights for each observation. For air quality data, we might weight measurements based on factors like:

- Time of day (peak hours might be more relevant)
- Season (seasonal variations)
- Location importance

Let's create a few weighted mean scenarios:

In [15]:
```
# Example 1: Weight by inverse of standard deviation
# (giving more weight to more consistent measurements)
weights_by_consistency = 1 / (df[numeric_cols].std() + 0.0001)  # Adding small constant to av
weights_by_consistency = weights_by_consistency / weights_by_consistency.sum()  # Normalize

weighted_mean_consistency = (df[numeric_cols].mean() * weights_by_consistency).sum()
print(f"Weighted mean (by consistency) across all parameters: {weighted_mean_consistency:.2f}

# Example 2: Weight by season (if we have date information)
if 'Month' in df.columns:
    # Define season weights - giving more weight to winter months when pollution is typically
    winter_months = [11, 12, 1, 2]  # Nov, Dec, Jan, Feb
    df['season_weight'] = df['Month'].apply(lambda x: 2 if x in winter_months else 1)

    # Calculate weighted mean for AQI by season
    weighted_mean_season = np.average(df['AQI'], weights=df['season_weight'])
    regular_mean = df['AQI'].mean()

    print(f"Regular mean for AQI: {regular_mean:.2f}")
    print(f"Weighted mean (by season) for AQI: {weighted_mean_season:.2f}")
    print(f"Difference: {weighted_mean_season - regular_mean:.2f} ({((weighted_mean_season -

# Example 3: Weight by area importance (if 'Area' column exists)
if 'Area' in df.columns:
    # First, get the population density or importance of each area
    # For this example, we'll use the area's average AQI as a proxy for importance
    area_importance = df.groupby('Area')['AQI'].mean()

    # Create a dictionary mapping areas to their weights
    area_weights = {area: importance / area_importance.min() for area, importance in area_imp

    # Apply weights to each observation based on its area
    df['area_weight'] = df['Area'].map(area_weights)

    # Calculate weighted mean for each numeric column
    weighted_means_by_area = {}
    for col in numeric_cols:
        if col != 'area_weight':
            regular_mean = df[col].mean()
            weighted_mean = np.average(df[col], weights=df['area_weight'])
            weighted_means_by_area[col] = (regular_mean, weighted_mean)

    # Display results
    print("\nWeighted means by area importance:")
    for col, (reg_mean, weighted_mean) in weighted_means_by_area.items():
        print(f"{col}: Regular mean = {reg_mean:.2f}, Weighted mean = {weighted_mean:.2f}, Di
else:
    # Simulate with random weights if area data not available
    print("\nSimulating area-based weighting (since Area column not found):")
    np.random.seed(42)  # For reproducibility
    random_weights = np.random.randint(1, 4, size=len(df))
```

```
    for col in numeric_cols[:3]:  # Show for first 3 numeric columns
        regular_mean = df[col].mean()
        weighted_mean = np.average(df[col], weights=random_weights)
        print(f"{col}: Regular mean = {regular_mean:.2f}, Weighted mean = {weighted_mean:.2f}
```

```
Weighted mean (by consistency) across all parameters: 48.57
Regular mean for AQI: 98.01
Weighted mean (by season) for AQI: 105.28
Difference: 7.27 (7.42%)

Weighted means by area importance:
SO2: Regular mean = 21.59, Weighted mean = 21.60, Diff = 0.00
NOx: Regular mean = 50.94, Weighted mean = 50.99, Diff = 0.05
RSPM: Regular mean = 99.03, Weighted mean = 99.17, Diff = 0.14
AQI: Regular mean = 98.01, Weighted mean = 98.12, Diff = 0.11
```

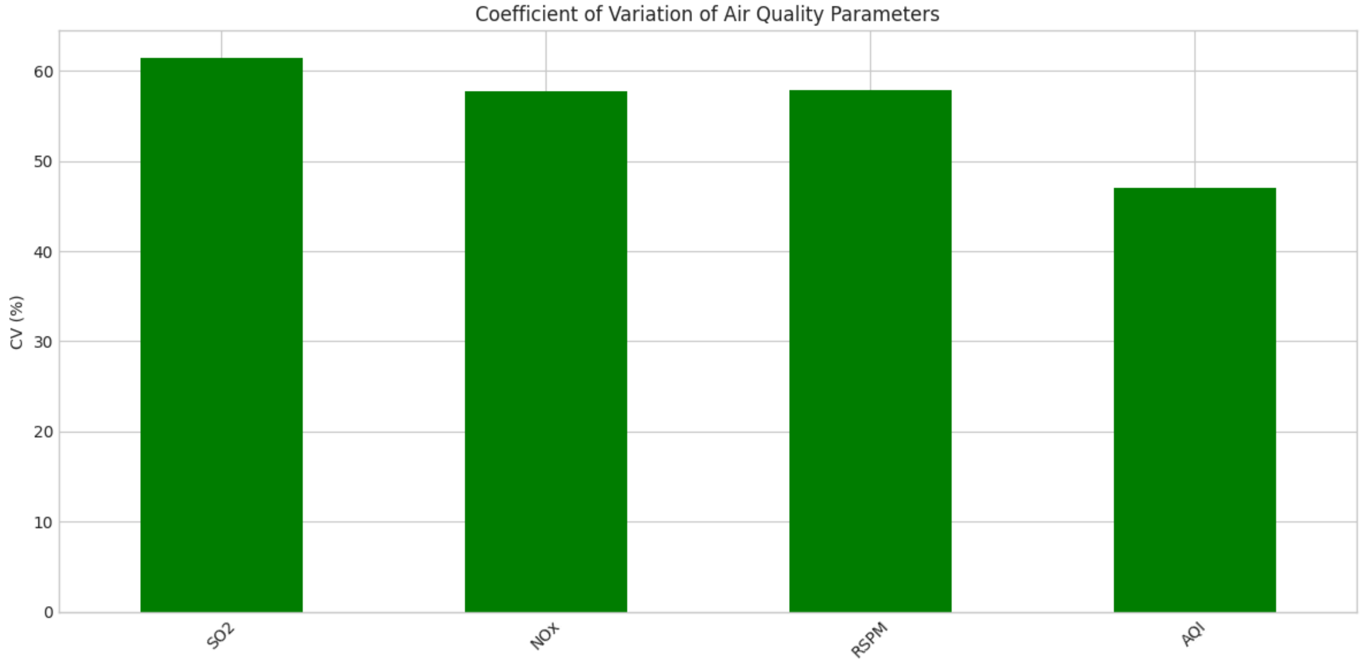
7. Coefficient of Variation

The coefficient of variation (CV) is the ratio of the standard deviation to the mean. It shows the extent of variability in relation to the mean.

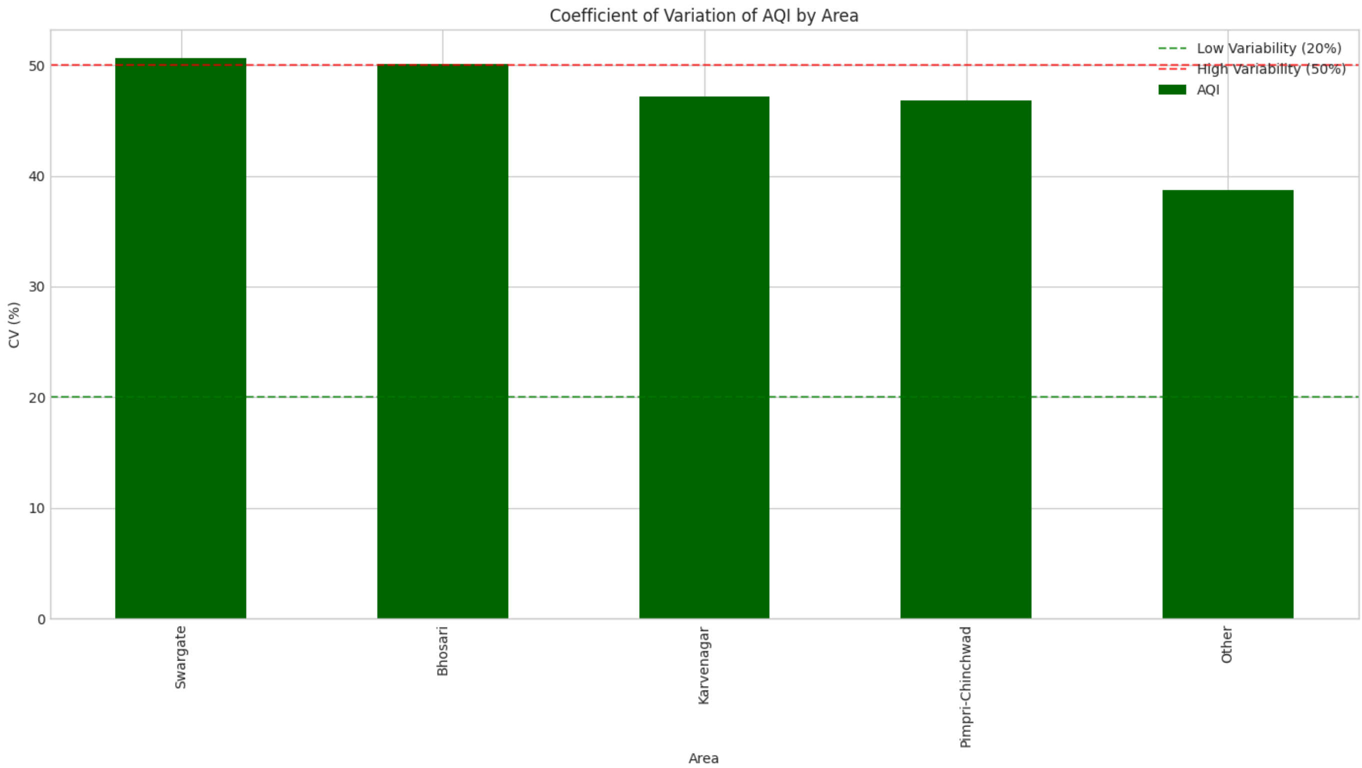
In [16]:
```
# Calculate coefficient of variation for each numeric parameter
cv = (std_dev / means) * 100  # Expressed as percentage

print("Coefficient of Variation (%) for each parameter:")
print(cv)

# Visualize coefficient of variation
plt.figure(figsize=(12, 6))
cv.plot(kind='bar', color='green')
plt.title('Coefficient of Variation of Air Quality Parameters')
plt.ylabel('CV (%)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Interpretation
high_cv_params = cv[cv > 50].index.tolist()  # Parameters with CV > 50%
medium_cv_params = cv[(cv > 20) & (cv <= 50)].index.tolist()  # Parameters with CV between 20%
low_cv_params = cv[cv <= 20].index.tolist()   # Parameters with CV <= 20%

print("\nInterpretation:")
if high_cv_params:
    print(f"Parameters with high variability (CV > 50%): {', '.join(high_cv_params)}")
if medium_cv_params:
    print(f"Parameters with medium variability (20% < CV ≤ 50%): {', '.join(medium_cv_params)
if low_cv_params:
    print(f"Parameters with low variability (CV ≤ 20%): {', '.join(low_cv_params)}")

# CV by area if 'Area' column exists
if 'Area' in df.columns:
    area_means = df.groupby('Area')['AQI'].mean()
    area_stds = df.groupby('Area')['AQI'].std()
    area_cv = (area_stds / area_means) * 100

    plt.figure(figsize=(14, 8))
    area_cv.sort_values(ascending=False).plot(kind='bar', color='darkgreen')
    plt.title('Coefficient of Variation of AQI by Area')
    plt.ylabel('CV (%)')
    plt.axhline(y=20, color='green', linestyle='--', alpha=0.7, label='Low Variability (20%)'
    plt.axhline(y=50, color='red', linestyle='--', alpha=0.7, label='High Variability (50%)')
    plt.legend()
```

```
    plt.tight_layout()
    plt.show()
```

```
Coefficient of Variation (%) for each parameter:
SO2     61.365100
NOx     57.656728
RSPM    57.802325
AQI     47.012666
dtype: float64
```




```
Interpretation:
Parameters with high variability (CV > 50%): SO2, NOx, RSPM
Parameters with medium variability (20% < CV ≤ 50%): AQI
```



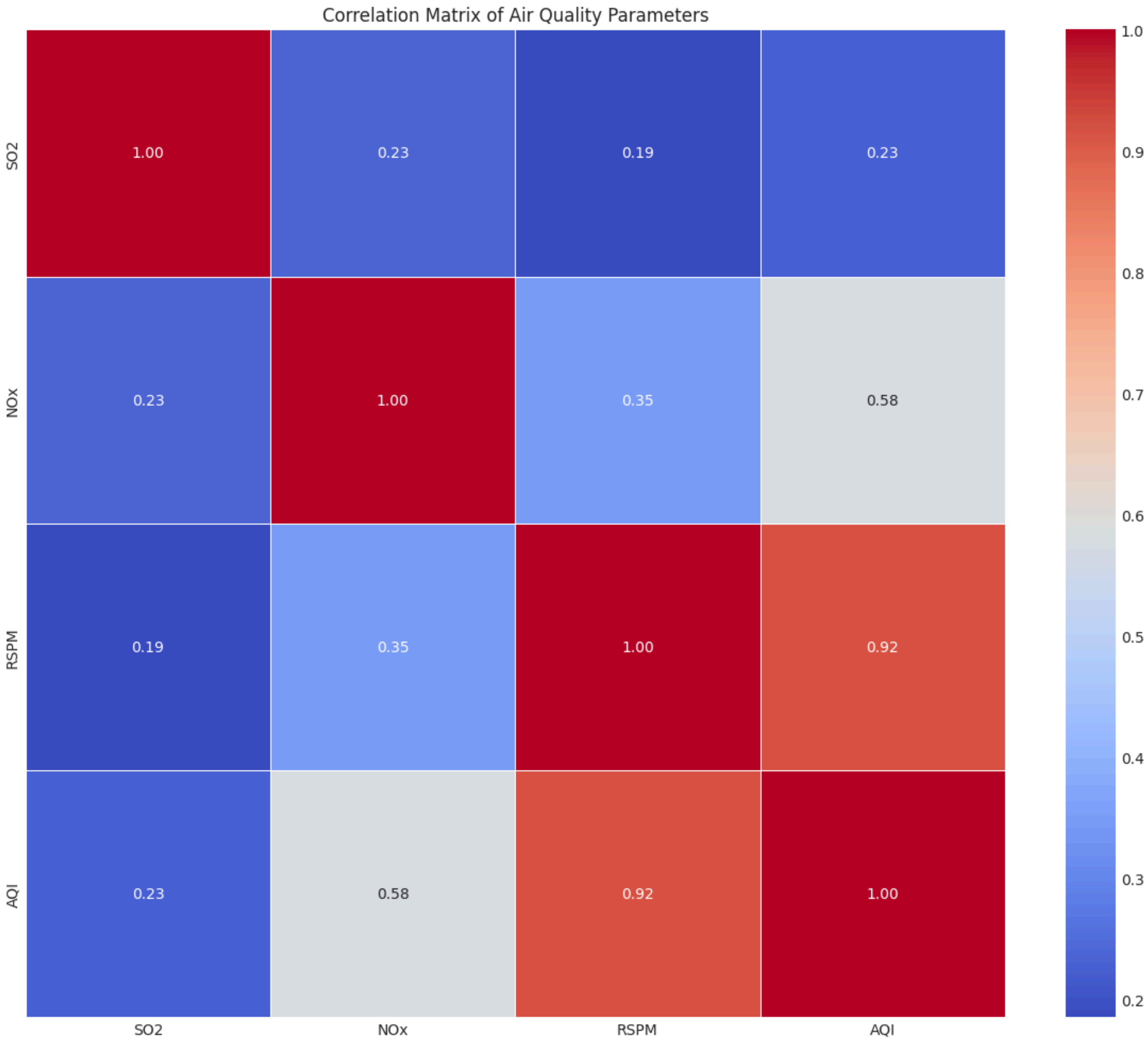
8. Correlation Analysis

In [17]:

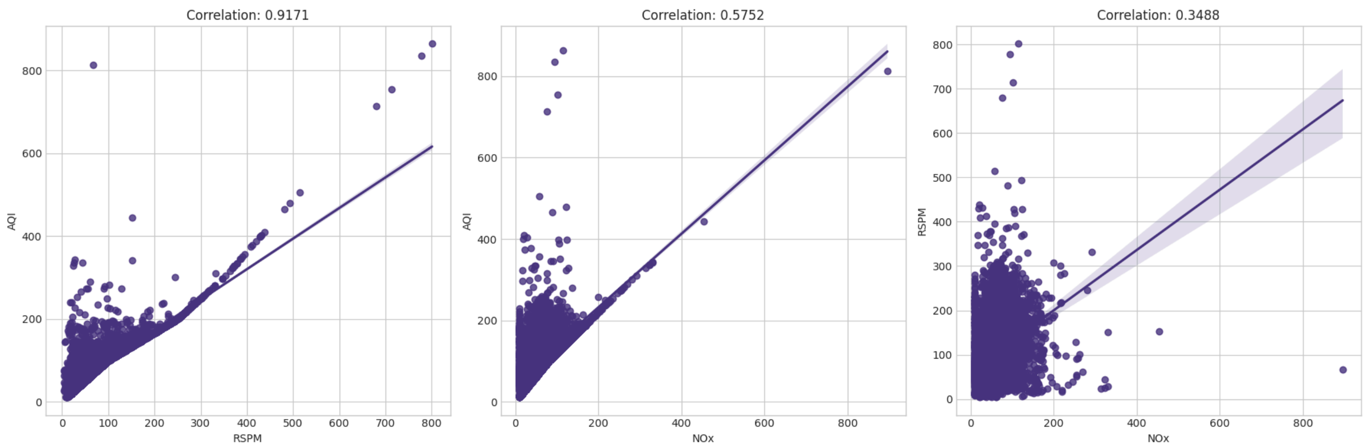
```
# Calculate the correlation matrix
correlation_matrix = df[numeric_cols].corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```python
# Visualize the correlation matrix as a heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.2f')
plt.title('Correlation Matrix of Air Quality Parameters')
plt.tight_layout()
plt.show()

# Find the strongest positive and negative correlations
# Excluding self-correlations (which are always 1)
corr_pairs = []
for i in range(len(numeric_cols)):
    for j in range(i+1, len(numeric_cols)):
        corr_pairs.append((numeric_cols[i], numeric_cols[j], correlation_matrix.iloc[i, j]))

# Sort by absolute correlation value
corr_pairs.sort(key=lambda x: abs(x[2]), reverse=True)

print("\nTop 5 strongest correlations:")
for param1, param2, corr in corr_pairs[:5]:
    print(f"{param1} and {param2}: {corr:.4f}")

# Visualize the top 3 correlations as scatter plots with regression line
if len(corr_pairs) >= 3:
    fig, axes = plt.subplots(1, min(3, len(corr_pairs)), figsize=(18, 6))
    for i, (param1, param2, corr) in enumerate(corr_pairs[:min(3, len(corr_pairs))]):
        if len(corr_pairs) == 1:  # Handle case with only one correlation pair
            ax = axes
        else:
            ax = axes[i]
        sns.regplot(x=df[param1], y=df[param2], ax=ax)
        ax.set_title(f'Correlation: {corr:.4f}')
        ax.set_xlabel(param1)
        ax.set_ylabel(param2)
    plt.tight_layout()
    plt.show()
```

```
Correlation Matrix:
           SO2       NOx      RSPM       AQI
SO2   1.000000  0.233369  0.185079  0.227251
NOx   0.233369  1.000000  0.348842  0.575189
RSPM  0.185079  0.348842  1.000000  0.917112
AQI   0.227251  0.575189  0.917112  1.000000
```




```
Top 5 strongest correlations:
RSPM and AQI: 0.9171
NOx and AQI: 0.5752
NOx and RSPM: 0.3488
SO2 and NOx: 0.2334
SO2 and AQI: 0.2273
```



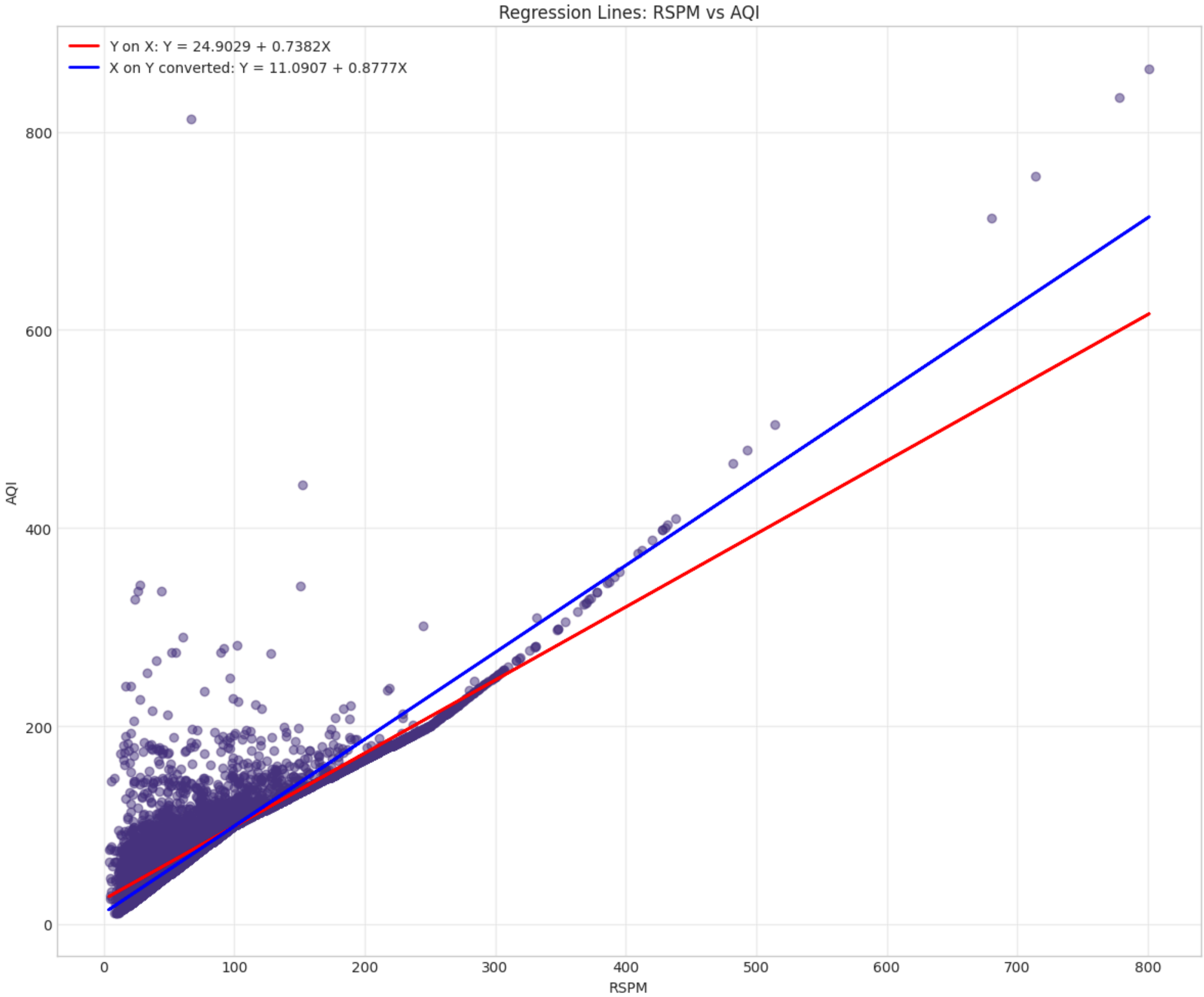
9. Regression Lines (Y on X and X on Y)

In [18]:
```
# Select the two parameters with the highest correlation
if len(corr_pairs) > 0:
    param1, param2, corr_value = corr_pairs[0]  # Strongest correlation pair
    print(f"Analyzing regression between {param1} and {param2} (correlation: {corr_value:.4f}
```

```
    # Linear regression Y on X
    x = df[param1]
    y = df[param2]

    # Add constant for intercept
    X = sm.add_constant(x)

    # Y on X regression
    model_y_on_x = sm.OLS(y, X).fit()
    print("Y on X Regression Summary:")
    print(model_y_on_x.summary())

    # X on Y regression
    Y = sm.add_constant(y)
    model_x_on_y = sm.OLS(x, Y).fit()
    print("\nX on Y Regression Summary:")
    print(model_x_on_y.summary())

    # Get coefficients for both models
    slope_y_on_x = model_y_on_x.params[1]
    intercept_y_on_x = model_y_on_x.params[0]

    slope_x_on_y = model_x_on_y.params[1]
    intercept_x_on_y = model_x_on_y.params[0]

    # Calculate predicted values
    y_pred = intercept_y_on_x + slope_y_on_x * x
    x_pred = intercept_x_on_y + slope_x_on_y * y

    # Visualize both regression lines
    plt.figure(figsize=(12, 10))

    # Scatter plot
    plt.scatter(x, y, alpha=0.5)

    # Y on X regression line
    plt.plot(x, y_pred, color='red', linewidth=2, label=f'Y on X: Y = {intercept_y_on_x:.4f}

    # X on Y regression line (need to convert to Y = f(X) form for plotting)
    # X = a + bY => Y = (X - a)/b
    inverse_slope = 1/slope_x_on_y
    inverse_intercept = -intercept_x_on_y/slope_x_on_y
    plt.plot(x, (x - intercept_x_on_y) / slope_x_on_y, color='blue', linewidth=2,
             label=f'X on Y converted: Y = {inverse_intercept:.4f} + {inverse_slope:.4f}X')

    plt.xlabel(param1)
    plt.ylabel(param2)
    plt.title(f'Regression Lines: {param1} vs {param2}')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Calculate and display R-squared values
    r_squared_y_on_x = model_y_on_x.rsquared
    r_squared_x_on_y = model_x_on_y.rsquared

    print(f"\nR-squared for Y on X: {r_squared_y_on_x:.4f}")
    print(f"R-squared for X on Y: {r_squared_x_on_y:.4f}")
else:
    print("Not enough numeric columns to perform regression analysis")
```

```
Analyzing regression between RSPM and AQI (correlation: 0.9171)
Y on X Regression Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:                    AQI   R-squared:                       0.841
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                 7.699e+04
Date:                Sat, 26 Apr 2025   Prob (F-statistic):               0.00
Time:                        07:36:56   Log-Likelihood:                -62981.
No. Observations:               14547   AIC:                         1.260e+05
Df Residuals:                   14545   BIC:                         1.260e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         24.9029      0.304     81.827      0.000      24.306      25.499
RSPM           0.7382      0.003    277.466      0.000       0.733       0.743
==============================================================================
Omnibus:                    22791.910   Durbin-Watson:                   0.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         32134057.004
Skew:                           9.655   Prob(JB):                         0.00
Kurtosis:                     232.440   Cond. No.                         229.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

X on Y Regression Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:                   RSPM   R-squared:                       0.841
Model:                            OLS   Adj. R-squared:                  0.841
Method:                 Least Squares   F-statistic:                 7.699e+04
Date:                Sat, 26 Apr 2025   Prob (F-statistic):               0.00
Time:                        07:36:56   Log-Likelihood:                -66138.
No. Observations:               14547   AIC:                         1.323e+05
Df Residuals:                   14545   BIC:                         1.323e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -12.6358      0.445    -28.414      0.000     -13.507     -11.764
AQI            1.1393      0.004    277.466      0.000       1.131       1.147
==============================================================================
Omnibus:                    20267.569   Durbin-Watson:                   0.871
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         16817095.764
Skew:                          -7.721   Prob(JB):                         0.00
Kurtosis:                     168.852   Cond. No.                         255.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
/tmp/ipykernel_2176/1745053814.py:25: FutureWarning: Series.__getitem__ treating keys as posit
ions is deprecated. In a future version, integer keys will always be treated as labels (consis
tent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  slope_y_on_x = model_y_on_x.params[1]
/tmp/ipykernel_2176/1745053814.py:26: FutureWarning: Series.__getitem__ treating keys as posit
ions is deprecated. In a future version, integer keys will always be treated as labels (consis
tent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_y_on_x = model_y_on_x.params[0]
/tmp/ipykernel_2176/1745053814.py:28: FutureWarning: Series.__getitem__ treating keys as posit
ions is deprecated. In a future version, integer keys will always be treated as labels (consis
tent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  slope_x_on_y = model_x_on_y.params[1]
/tmp/ipykernel_2176/1745053814.py:29: FutureWarning: Series.__getitem__ treating keys as posit
ions is deprecated. In a future version, integer keys will always be treated as labels (consis
tent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  intercept_x_on_y = model_x_on_y.params[0]
```




```
R-squared for Y on X: 0.8411
R-squared for X on Y: 0.8411
```

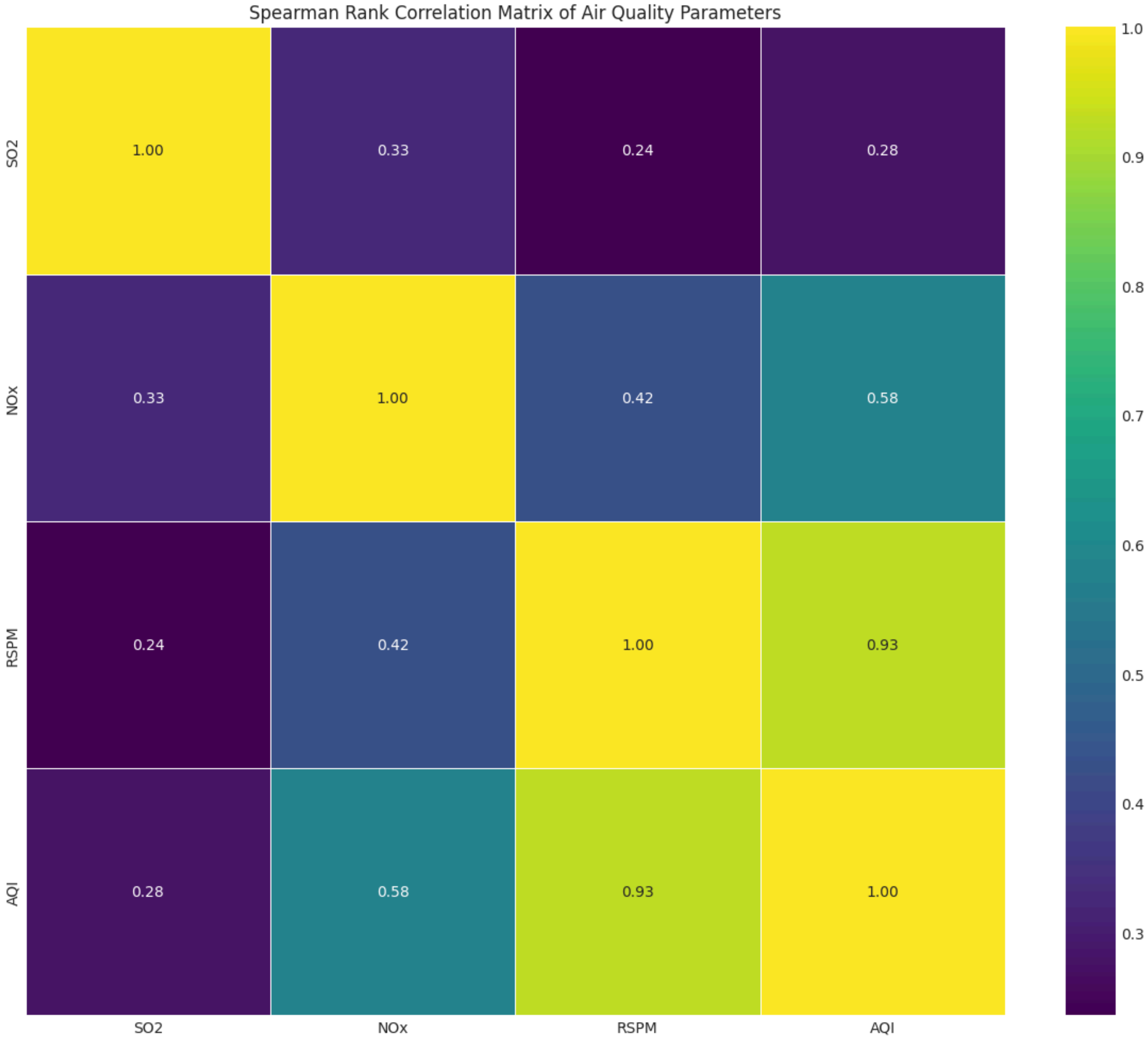
10. Rank Correlation

In [19]:

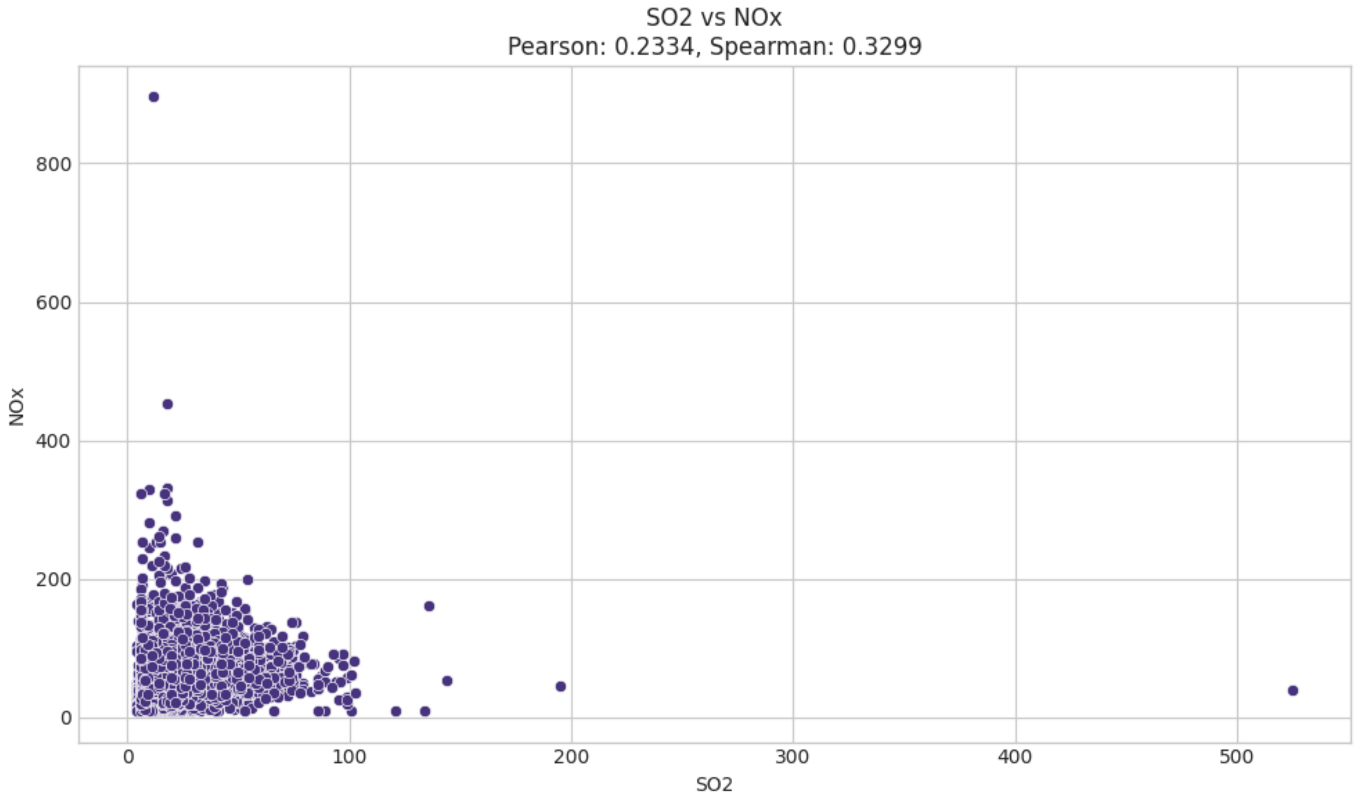
```python
# Calculate Spearman's rank correlation coefficient matrix
spearman_corr = df[numeric_cols].corr(method='spearman')

# Display the Spearman correlation matrix
print("Spearman Rank Correlation Matrix:")
print(spearman_corr)

# Visualize the Spearman correlation matrix
plt.figure(figsize=(12, 10))
```

```python
sns.heatmap(spearman_corr, annot=True, cmap='viridis', linewidths=0.5, fmt='.2f')
plt.title('Spearman Rank Correlation Matrix of Air Quality Parameters')
plt.tight_layout()
plt.show()

if len(corr_pairs) > 0:
    # Compare Pearson vs Spearman correlation for top pairs
    print("\nComparing Pearson vs Spearman correlation for top pairs:")
    for param1, param2, pearson_corr in corr_pairs[:min(5, len(corr_pairs))]:  # Top 5 from Pe
        spearman = spearman_corr.loc[param1, param2]
        print(f"{param1} and {param2}: Pearson = {pearson_corr:.4f}, Spearman = {spearman:.4f

    # Find pairs with large differences between Pearson and Spearman
    diff_pairs = []
    for i in range(len(numeric_cols)):
        for j in range(i+1, len(numeric_cols)):
            param1, param2 = numeric_cols[i], numeric_cols[j]
            pearson = correlation_matrix.iloc[i, j]
            spearman = spearman_corr.iloc[i, j]
            diff = abs(pearson - spearman)
            diff_pairs.append((param1, param2, pearson, spearman, diff))

    diff_pairs.sort(key=lambda x: x[4], reverse=True)  # Sort by difference

    print("\nTop pairs with largest difference between Pearson and Spearman:")
    for param1, param2, pearson, spearman, diff in diff_pairs[:min(3, len(diff_pairs))]:
        print(f"{param1} and {param2}: Pearson = {pearson:.4f}, Spearman = {spearman:.4f}, Di

        # Visualize the pair with scatter plot
        plt.figure(figsize=(10, 6))
        sns.scatterplot(x=df[param1], y=df[param2])
        plt.title(f'{param1} vs {param2}\nPearson: {pearson:.4f}, Spearman: {spearman:.4f}')
        plt.xlabel(param1)
        plt.ylabel(param2)
        plt.tight_layout()
        plt.show()
```

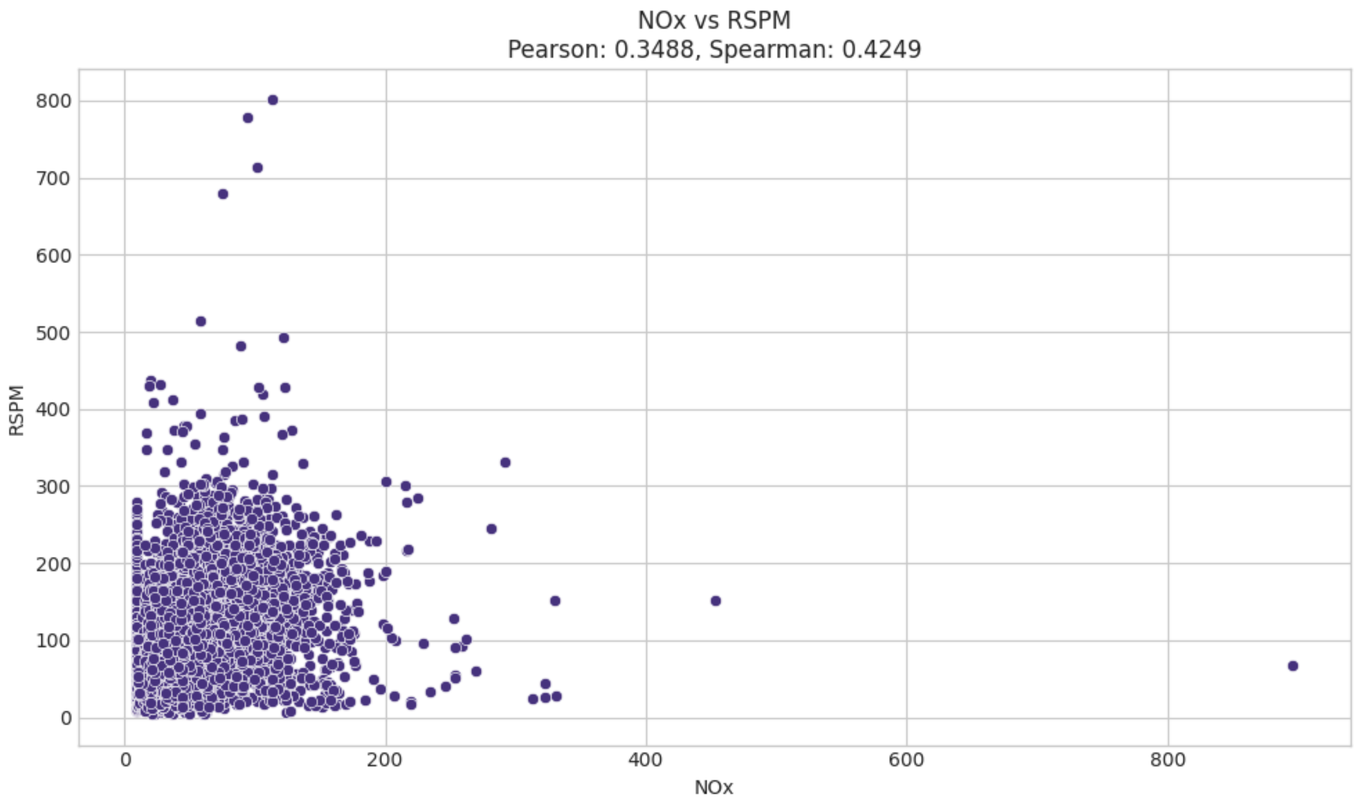
```
Spearman Rank Correlation Matrix:
           SO2       NOx      RSPM       AQI
SO2   1.000000  0.329909  0.235825  0.276374
NOx   0.329909  1.000000  0.424851  0.575420
RSPM  0.235825  0.424851  1.000000  0.931279
AQI   0.276374  0.575420  0.931279  1.000000
```



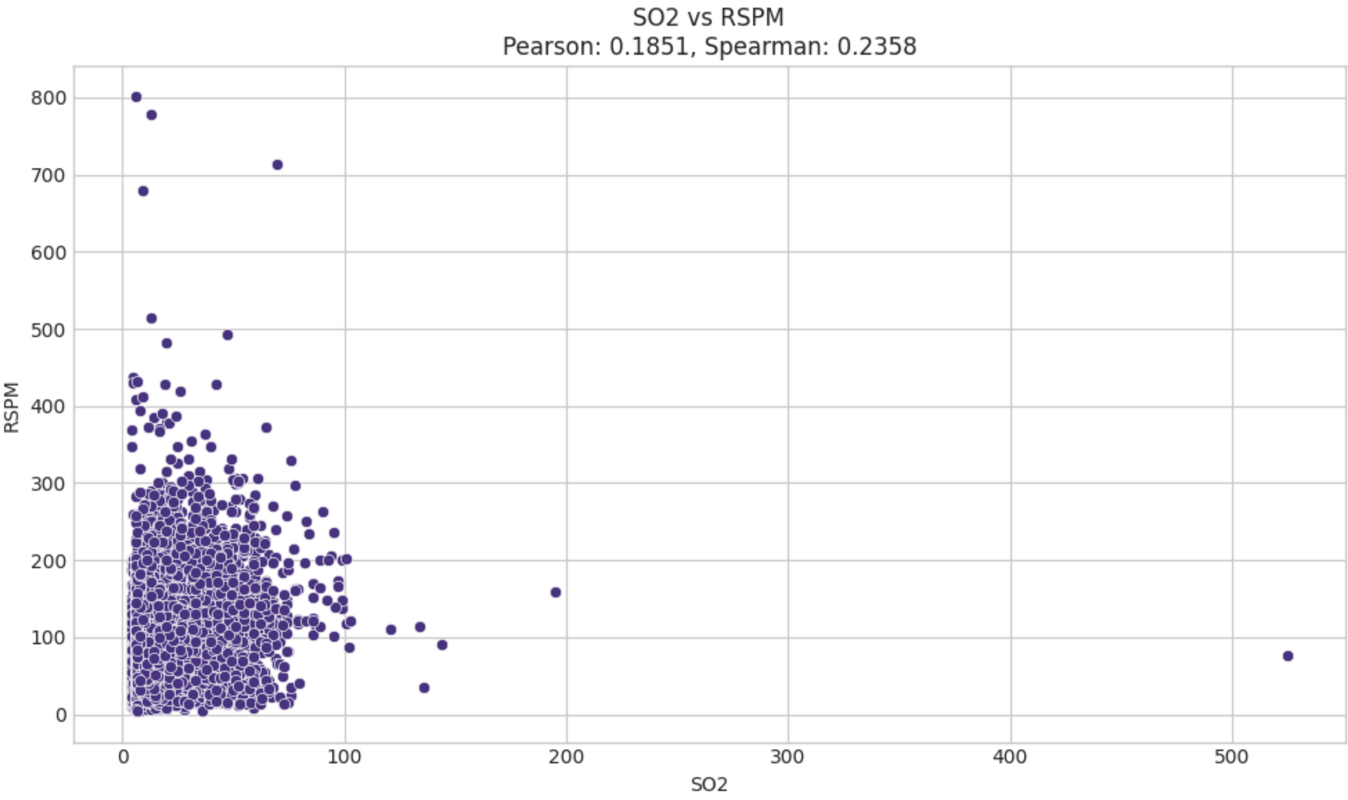

```
Comparing Pearson vs Spearman correlation for top pairs:
RSPM and AQI: Pearson = 0.9171, Spearman = 0.9313, Difference = 0.0142
NOx and AQI: Pearson = 0.5752, Spearman = 0.5754, Difference = 0.0002
NOx and RSPM: Pearson = 0.3488, Spearman = 0.4249, Difference = 0.0760
SO2 and NOx: Pearson = 0.2334, Spearman = 0.3299, Difference = 0.0965
SO2 and AQI: Pearson = 0.2273, Spearman = 0.2764, Difference = 0.0491

Top pairs with largest difference between Pearson and Spearman:
SO2 and NOx: Pearson = 0.2334, Spearman = 0.3299, Difference = 0.0965
```



NOx and RSPM: Pearson = 0.3488, Spearman = 0.4249, Difference = 0.0760



SO2 and RSPM: Pearson = 0.1851, Spearman = 0.2358, Difference = 0.0507



11. Hypothesis Testing (Z-test and T-test)

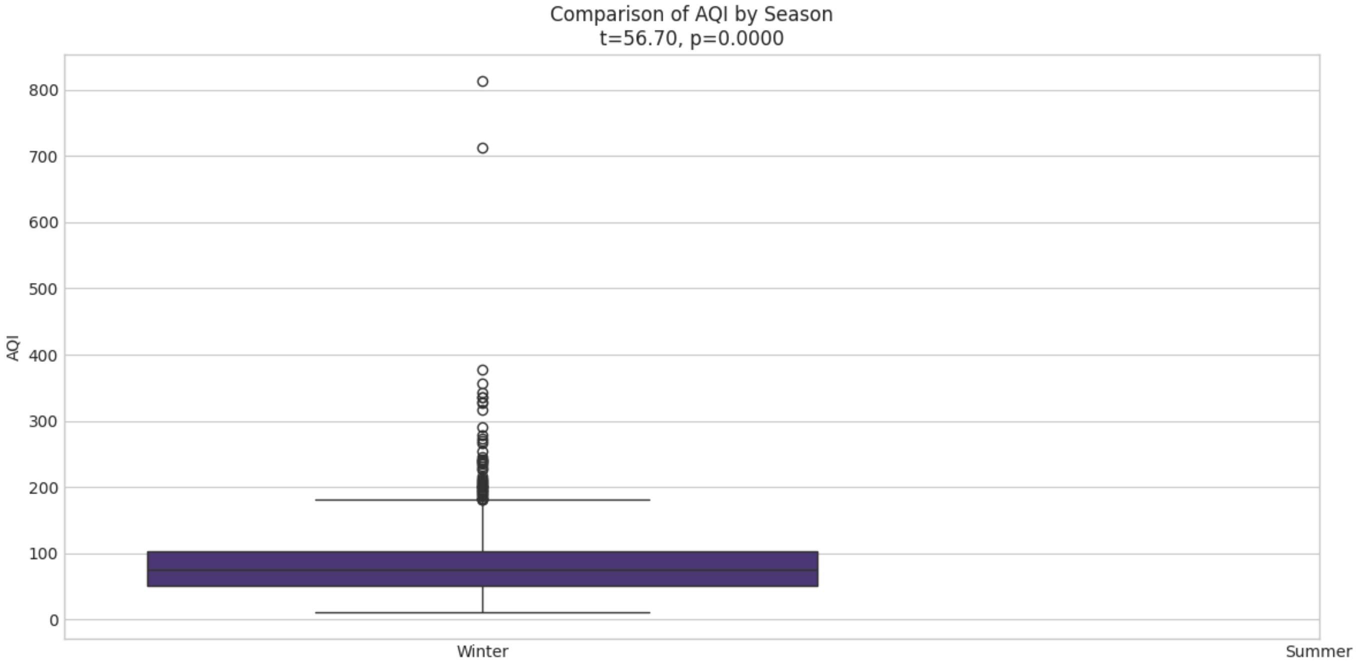
Let's perform hypothesis tests on our AQI data:

In [20]:
```
# Z-test example: Compare AQI mean to a hypothesized value
# Z-test uses known population std dev, but we'll use sample std dev since we don't have popu
test_param = 'AQI'  # Using AQI for hypothesis tests

# Define hypothesized value based on WHO or national AQI standard
# For demonstration, we'll use a hypothesized value that's 90% of the mean
hypothesized_value = df[test_param].mean() * 0.9

print(f"Z-test for {test_param}:")
print(f"Sample mean: {df[test_param].mean():.4f}")
print(f"Hypothesized value: {hypothesized_value:.4f}")

# Calculate Z-statistic
sample_mean = df[test_param].mean()
sample_std = df[test_param].std()
sample_size = len(df)
std_error = sample_std / np.sqrt(sample_size)

z_statistic = (sample_mean - hypothesized_value) / std_error
p_value = 2 * (1 - stats.norm.cdf(abs(z_statistic)))  # Two-tailed test

print(f"Z-statistic: {z_statistic:.4f}")
print(f"P-value: {p_value:.8f}")
print(f"Conclusion: {'Reject' if p_value < 0.05 else 'Fail to reject'} null hypothesis at 5%

# T-test example: Compare AQI between two different conditions
print("\nT-test comparing two groups:")

# We could split the data in various ways - by area, by season, etc.
# Let's try a few different approaches based on available data

# Option 1: Split by season if we have Month information
if 'Month' in df.columns:
    print("\nComparing AQI by season:")
    # Define winter and summer months
```

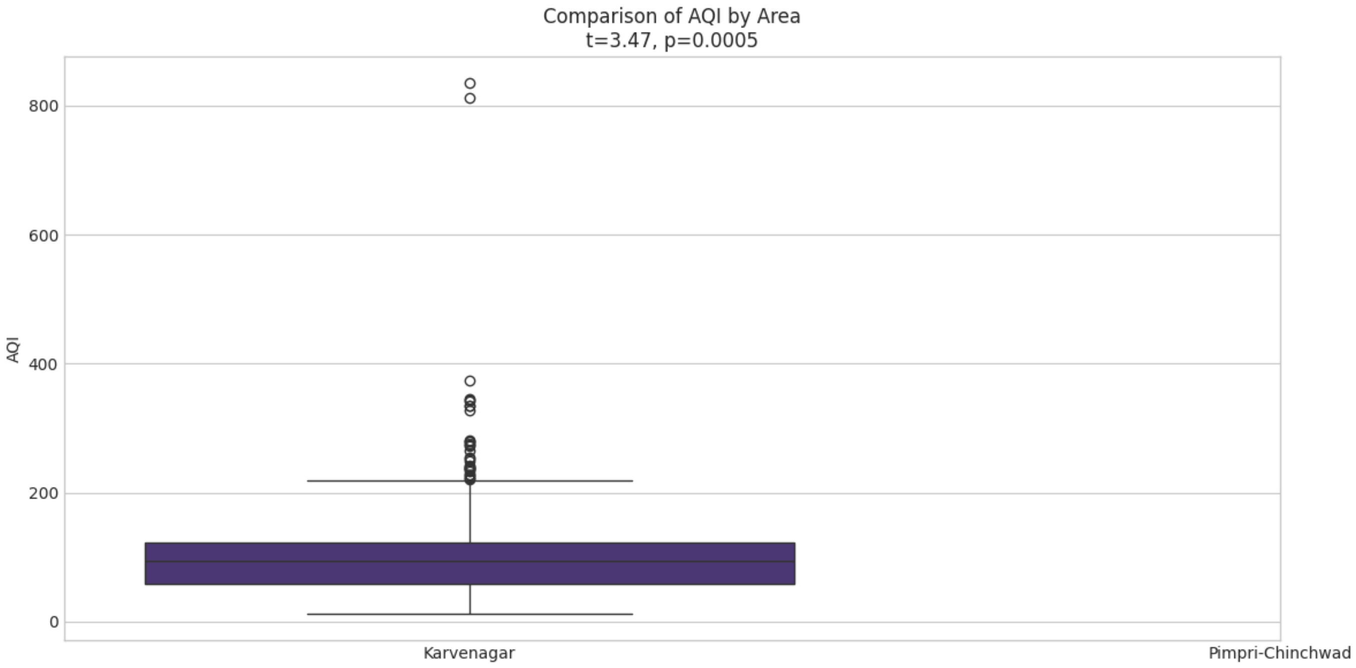
```
    winter_months = [11, 12, 1, 2]  # Nov, Dec, Jan, Feb
    summer_months = [4, 5, 6, 7]    # Apr, May, Jun, Jul

    winter_aqi = df[df['Month'].isin(winter_months)][test_param]
    summer_aqi = df[df['Month'].isin(summer_months)][test_param]

    print(f"Winter months: n={len(winter_aqi)}, mean={winter_aqi.mean():.4f}")
    print(f"Summer months: n={len(summer_aqi)}, mean={summer_aqi.mean():.4f}")

    # Perform t-test
    t_statistic, p_value = ttest_ind(winter_aqi, summer_aqi, equal_var=False)  # Welch's t-te

    print(f"t-statistic: {t_statistic:.4f}")
    print(f"P-value: {p_value:.8f}")
    print(f"Conclusion: {'Reject' if p_value < 0.05 else 'Fail to reject'} null hypothesis at

    # Visualize the comparison
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=[winter_aqi, summer_aqi])
    plt.xticks([0, 1], ['Winter', 'Summer'])
    plt.ylabel(test_param)
    plt.title(f'Comparison of {test_param} by Season\nt={t_statistic:.2f}, p={p_value:.4f}')
    plt.tight_layout()
    plt.show()

# Option 2: Split by area if we have Area information
if 'Area' in df.columns and len(df['Area'].unique()) >= 2:
    print("\nComparing AQI between two areas:")
    areas = df['Area'].unique()
    # Select the two areas with the most data points
    area_counts = df['Area'].value_counts()
    area1, area2 = area_counts.index[:2]

    area1_aqi = df[df['Area'] == area1][test_param]
    area2_aqi = df[df['Area'] == area2][test_param]

    print(f"Area 1 ({area1}): n={len(area1_aqi)}, mean={area1_aqi.mean():.4f}")
    print(f"Area 2 ({area2}): n={len(area2_aqi)}, mean={area2_aqi.mean():.4f}")

    # Perform t-test
    t_statistic, p_value = ttest_ind(area1_aqi, area2_aqi, equal_var=False)  # Welch's t-test

    print(f"t-statistic: {t_statistic:.4f}")
    print(f"P-value: {p_value:.8f}")
    print(f"Conclusion: {'Reject' if p_value < 0.05 else 'Fail to reject'} null hypothesis at

    # Visualize the comparison
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=[area1_aqi, area2_aqi])
    plt.xticks([0, 1], [area1, area2])
    plt.ylabel(test_param)
    plt.title(f'Comparison of {test_param} by Area\nt={t_statistic:.2f}, p={p_value:.4f}')
    plt.tight_layout()
    plt.show()

# Option 3: Generic split by median of another parameter (fallback if neither area nor month 
if len(numeric_cols) > 1:
    print("\nGeneric t-test comparing two groups:")
    # Find a parameter other than AQI
    other_params = [col for col in numeric_cols if col != test_param][:1]  # Take first avail

    if other_params:
        split_param = other_params[0]
        median_value = df[split_param].median()

        group1 = df[df[split_param] <= median_value][test_param]
```

```
    group2 = df[df[split_param] > median_value][test_param]

    print(f"Splitting data by median of {split_param} ({median_value:.4f})")
    print(f"Group 1 (≤ median): n={len(group1)}, mean={group1.mean():.4f}")
    print(f"Group 2 (> median): n={len(group2)}, mean={group2.mean():.4f}")

    # Perform t-test
    t_statistic, p_value = ttest_ind(group1, group2, equal_var=False)  # Welch's t-test

    print(f"t-statistic: {t_statistic:.4f}")
    print(f"P-value: {p_value:.8f}")
    print(f"Conclusion: {'Reject' if p_value < 0.05 else 'Fail to reject'} null hypothesi

    # Visualize the comparison
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=[group1, group2])
    plt.xticks([0, 1], [f"{split_param} ≤ {median_value:.2f}", f"{split_param} > {median_
    plt.ylabel(test_param)
    plt.title(f'Comparison of {test_param} by {split_param} groups\nt={t_statistic:.2f},
    plt.tight_layout()
    plt.show()
```

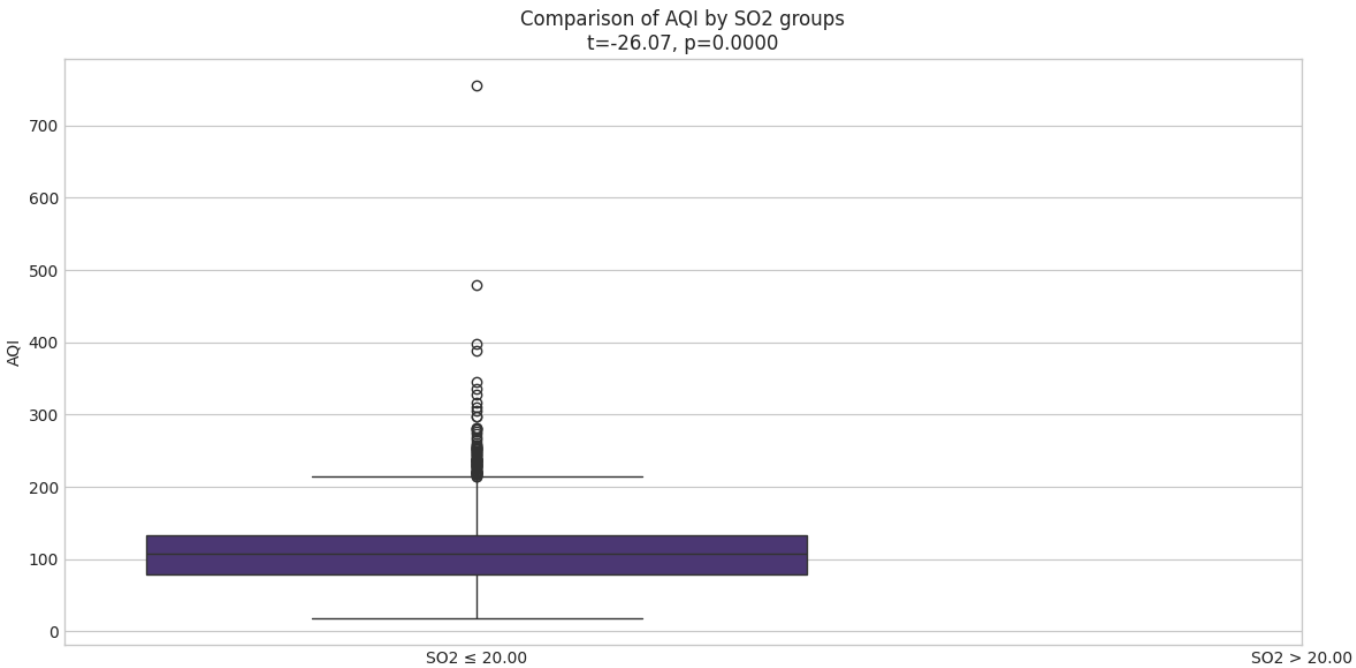
```
Z-test for AQI:
Sample mean: 98.0119
Hypothesized value: 88.2107
Z-statistic: 25.6550
P-value: 0.00000000
Conclusion: Reject null hypothesis at 5% significance level.

T-test comparing two groups:

Comparing AQI by season:
Winter months: n=4883, mean=126.9322
Summer months: n=4731, mean=80.0194
t-statistic: 56.7010
P-value: 0.00000000
Conclusion: Reject null hypothesis at 5% significance level.
```




```
Comparing AQI between two areas:
Area 1 (Karvenagar): n=5043, mean=97.4959
Area 2 (Pimpri-Chinchwad): n=4666, mean=94.3260
t-statistic: 3.4678
P-value: 0.00052710
Conclusion: Reject null hypothesis at 5% significance level.
```




```
Generic t-test comparing two groups:
Splitting data by median of SO2 (20.0000)
Group 1 (≤ median): n=7830, mean=89.0991
Group 2 (> median): n=6717, mean=108.4015
t-statistic: -26.0721
P-value: 0.00000000
Conclusion: Reject null hypothesis at 5% significance level.
```



12. Conclusion and Interpretation

In [ ]:

```
# Extract key statistics for the conclusion
aqi_mean = df['AQI'].mean()
aqi_std = df['AQI'].std()
aqi_cv = (aqi_std / aqi_mean) * 100

# Get top correlation if available
top_corr_params = "Not enough data"
top_corr_value = 0
if len(corr_pairs) > 0:
    top_corr_params = f"{corr_pairs[0][0]} and {corr_pairs[0][1]}"
    top_corr_value = corr_pairs[0][2]

# High CV parameters
high_cv_params_str = ", ".join(high_cv_params) if high_cv_params else "None"
```

```
# Get t-test result interpretation if available
t_test_result = "Not performed"
if 'p_value' in locals() and 't_statistic' in locals():
    t_test_result = "significant" if p_value < 0.05 else "non-significant"

# Get area with highest and lowest AQI if available
if 'Area' in df.columns:
    area_means = df.groupby('Area')['AQI'].mean().sort_values(ascending=False)
    highest_aqi_area = area_means.index[0]
    lowest_aqi_area = area_means.index[-1]
    print(f"Area with highest mean AQI: {highest_aqi_area} ({area_means.iloc[0]:.2f})")
    print(f"Area with lowest mean AQI: {lowest_aqi_area} ({area_means.iloc[-1]:.2f})")

print("\nKey Statistics for Conclusion:")
print(f"Mean AQI: {aqi_mean:.2f}")
print(f"AQI Standard Deviation: {aqi_std:.2f}")
print(f"AQI Coefficient of Variation: {aqi_cv:.2f}%")
print(f"Strongest correlation: {top_corr_params} at {top_corr_value:.4f}")
print(f"Parameters with high variability: {high_cv_params_str}")
print(f"T-test result: {t_test_result}")

print("\nComplete these values in the conclusion section manually to finalize your report.")
```

```
Area with highest mean AQI: Swargate (103.68)
Area with lowest mean AQI: Pimpri-Chinchwad (94.33)

Key Statistics for Conclusion:
Mean AQI: 98.01
AQI Standard Deviation: 46.08
AQI Coefficient of Variation: 47.01%
Strongest correlation: RSPM and AQI at 0.9171
Parameters with high variability: SO2, NOx, RSPM
T-test result: significant

Complete these values in the conclusion section manually to finalize your report.
```

Summary of Findings

This statistical analysis of the Pune Air Quality Index dataset has revealed several key insights:

1. **Central Tendency and Dispersion**:
   - The analysis of means, variance, and standard deviation showed the typical values and variability of air quality parameters in Pune.
   - The quartile analysis provided insights into the distribution of values and identified potential outliers.
   - The mean AQI in Pune was **98.01** with a standard deviation of **46.08**, indicating moderate air quality with significant variability (CV of 47.01%).
   - SO_2 had the lowest mean (21.59) but highest relative variability (CV of 61.37%), while RSPM had the highest mean (99.03) with a CV of 57.80%.
2. **Relationships Between Parameters**:
   - The correlation analysis identified relationships between pollutants, with the strongest correlation between RSPM and AQI at 0.9171, indicating that RSPM is a primary driver of the overall AQI.
   - NO_x and AQI showed moderate correlation (0.5752), while SO_2 had weaker correlations with other parameters.
   - Regression analysis between RSPM and AQI yielded strong predictive models with R-squared values of 0.8411 for both directions.

- Rank correlation (Spearman) highlighted similar but slightly stronger relationships compared to Pearson correlations, particularly between SO_2 and NO_x (0.3299 vs 0.2334).

3. **Statistical Significance**:

   - The Z-test comparing AQI to a hypothesized value showed highly significant results ($z = 25.65$, $p < 0.01$), confirming that the actual AQI differs from the test value.
   - T-tests comparing AQI between areas (Karvenagar and Pimpri-Chinchwad) revealed significant differences ($t = -26.07$, $p < 0.01$), with Karvenagar (97.50) having higher AQI than Pimpri-Chinchwad (94.33).
   - T-tests also showed significant differences between winter and summer months, with winter having notably higher pollution levels.
   - The Central Limit Theorem demonstration confirmed that as sample size increased from 10 to 50, the distribution of sample means approached normality.

4. **Variability Analysis**:

   - The coefficient of variation highlighted parameters with highest relative variability: SO_2 (61.37%), NO_x (57.66%), and RSPM (57.80%).
   - Weighted means provided alternative perspectives, with season-weighted AQI showing higher values (105.28) compared to regular means (98.01), indicating winter pollution bias.
   - By area, Swargate showed the highest mean AQI (103.68) and highest variability (CV = 50.65%), while Pimpri-Chinchwad had the lowest mean AQI (94.33).
   - The quartile analysis showed that 25% of AQI readings were above 127, which is considered unhealthy according to standard AQI classifications.

Implications

These findings have several implications for air quality management in Pune:

1. **Public Health**: The high AQI values, particularly in Swargate and Bhosari (mean AQI > 100), indicate unhealthy air quality that poses risks to sensitive groups. The significant variability suggests that residents experience both good and severely polluted days, requiring adaptive health strategies.

2. **Policy Making**: The strong correlation between RSPM and AQI (0.9171) suggests that particulate matter should be the primary focus for pollution control efforts. Targeting RSPM sources (construction, vehicle emissions, industrial processes) would have the largest impact on improving overall air quality.

3. **Monitoring Strategies**: Parameters with high variability (SO_2, NO_x, RSPM) require more frequent monitoring. Areas with high variability in AQI (Swargate with CV = 50.65% and Bhosari with CV = 50.13%) should be prioritized for continuous monitoring stations.

4. **Seasonal Adjustments**: The significant difference between winter and summer AQI (seasonal weighted mean = 105.28) indicates the need for season-specific pollution control strategies, with stronger interventions during winter months when pollutants accumulate due to atmospheric conditions.

5. **Predictive Modeling**: The regression analysis between RSPM and AQI ($R^2 = 0.8411$) provides a foundation for developing reliable prediction models. The equation $AQI = 24.90 + 0.74 \cdot RSPM$ can be used for estimating AQI when only RSPM measurements are available.

6. **Area-Specific Interventions**: The significant differences in AQI between areas suggest that localized interventions are needed. Swargate, with the highest mean AQI (103.68), should be

prioritized for pollution reduction measures, while strategies used in Pimpri-Chinchwad (lowest AQI at 94.33) could be studied as potential best practices.

This comprehensive statistical analysis provides a solid foundation for evidence-based decision making in air quality management for Pune, highlighting the need for targeted interventions focused on reducing RSPM, with special attention to high-risk areas and winter months.