SHADOWFOX

TASK2

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = '/kaggle/input/delhiaqi/delhiaqi.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(df.head())

# Step 1: Handle missing values
print(df.isnull().sum())
numeric_columns= df.select_dtypes(include=['number']).columns
df[numeric_columns] =
df[numeric_columns].fillna(df[numeric_columns].mean())

if 'date' in df.columns:
    df['date'] = pd.to_datetime(df['date'])
    print(df.columns.tolist())

df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day

pollutants = [ 'no2', 'so2', 'co', 'o3']
scaler = StandardScaler()
df[pollutants] = scaler.fit_transform(df[pollutants])

# Display the first few rows after preprocessing
print(df.head())
```

```
                  date       co     no    no2    o3    so2   pm2_5
pm10  \
0  2023-01-01 00:00:00  1655.58   1.66  39.41  5.90  17.88  169.29
194.64
1  2023-01-01 01:00:00  1869.20   6.82  42.16  1.99  22.17  182.84
211.08
2  2023-01-01 02:00:00  2510.07  27.72  43.87  0.02  30.04  220.25
260.68
3  2023-01-01 03:00:00  3150.94  55.43  44.55  0.85  35.76  252.90
304.12
```

```
4  2023-01-01 04:00:00  3471.37  68.84  45.24  5.45  39.10  266.36
322.80

     nh3
0   5.83
1   7.66
2  11.40
3  13.55
4  14.19
date     0
co       0
no       0
no2      0
o3       0
so2      0
pm2_5    0
pm10     0
nh3      0
dtype: int64
['date', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3']
                 date        co     no       no2        o3       so2
pm2_5  \
0 2023-01-01 00:00:00 -0.669597   1.66 -0.845569 -0.606902 -0.766585
169.29
1 2023-01-01 01:00:00 -0.603356   6.82 -0.780765 -0.704790 -0.696278
182.84
2 2023-01-01 02:00:00 -0.404628  27.72 -0.740469 -0.754109 -0.567301
220.25
3 2023-01-01 03:00:00 -0.205901  55.43 -0.724445 -0.733330 -0.473559
252.90
4 2023-01-01 04:00:00 -0.106538  68.84 -0.708185 -0.618168 -0.418822
266.36

     pm10    nh3  year  month  day
0  194.64   5.83  2023      1    1
1  211.08   7.66  2023      1    1
2  260.68  11.40  2023      1    1
3  304.12  13.55  2023      1    1
4  322.80  14.19  2023      1    1
```

```
df.columns
```

```
Index(['date', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3',
'year',
       'month', 'day'],
      dtype='object')
```

```
# Dimensions of the dataset
print(f"Dataset dimensions: {df.shape}")
```

```
Dataset dimensions: (561, 12)
```

```python
# Column names and data types
print(f"Column names and data types:\n{df.dtypes}")
```

```
Column names and data types:
date     datetime64[ns]
co              float64
no              float64
no2             float64
o3              float64
so2             float64
pm2_5           float64
pm10            float64
nh3             float64
year              int32
month             int32
day               int32
dtype: object
```

```python
# Summary of the dataset
print(df.describe())
```

```
                      date            co          no           no2  \
count                  561  5.610000e+02  561.000000  5.610000e+02
mean   2023-01-12 16:00:00  2.533129e-17   51.181979  2.279816e-16
min    2023-01-01 00:00:00 -9.801094e-01    0.000000 -1.459200e+00
25%    2023-01-06 20:00:00 -6.530385e-01    3.380000 -7.244452e-01
50%    2023-01-12 16:00:00 -3.797869e-01   13.300000 -2.719983e-01
75%    2023-01-18 12:00:00  1.915545e-01   59.010000  5.193125e-01
max    2023-01-24 08:00:00  4.050176e+00  425.580000  4.428265e+00
std                    NaN  1.000892e+00   83.904476  1.000892e+00

                 o3         so2        pm2_5         pm10         nh3
year  \
count  5.610000e+02  561.000000   561.000000   561.000000  561.000000
561.0
mean  -5.066258e-17    0.000000   358.256364   420.988414   26.425062
2023.0
min   -7.546096e-01   -0.973571    60.100000    69.080000    0.630000
2023.0
25%   -7.528571e-01   -0.598603   204.450000   240.900000    8.230000
2023.0
50%   -4.591942e-01   -0.285912   301.170000   340.900000   14.820000
2023.0
75%    4.273022e-01    0.206397   416.650000   482.570000   26.350000
2023.0
max    3.363931e+00    7.317669  1310.200000  1499.270000  267.510000
2023.0
std    1.000892e+00    1.000892   227.359117   271.287026   36.563094
```

```
0.0

       month        day
count  561.0  561.000000
mean     1.0   12.192513
min      1.0    1.000000
25%      1.0    6.000000
50%      1.0   12.000000
75%      1.0   18.000000
max      1.0   24.000000
std      0.0    6.756374
```

```
# Unique values in each column
print(df.nunique())
```

```
date     561
co       224
no       346
no2      198
o3       283
so2      231
pm2_5    557
pm10     556
nh3      269
year       1
month      1
day       24
dtype: int64
```

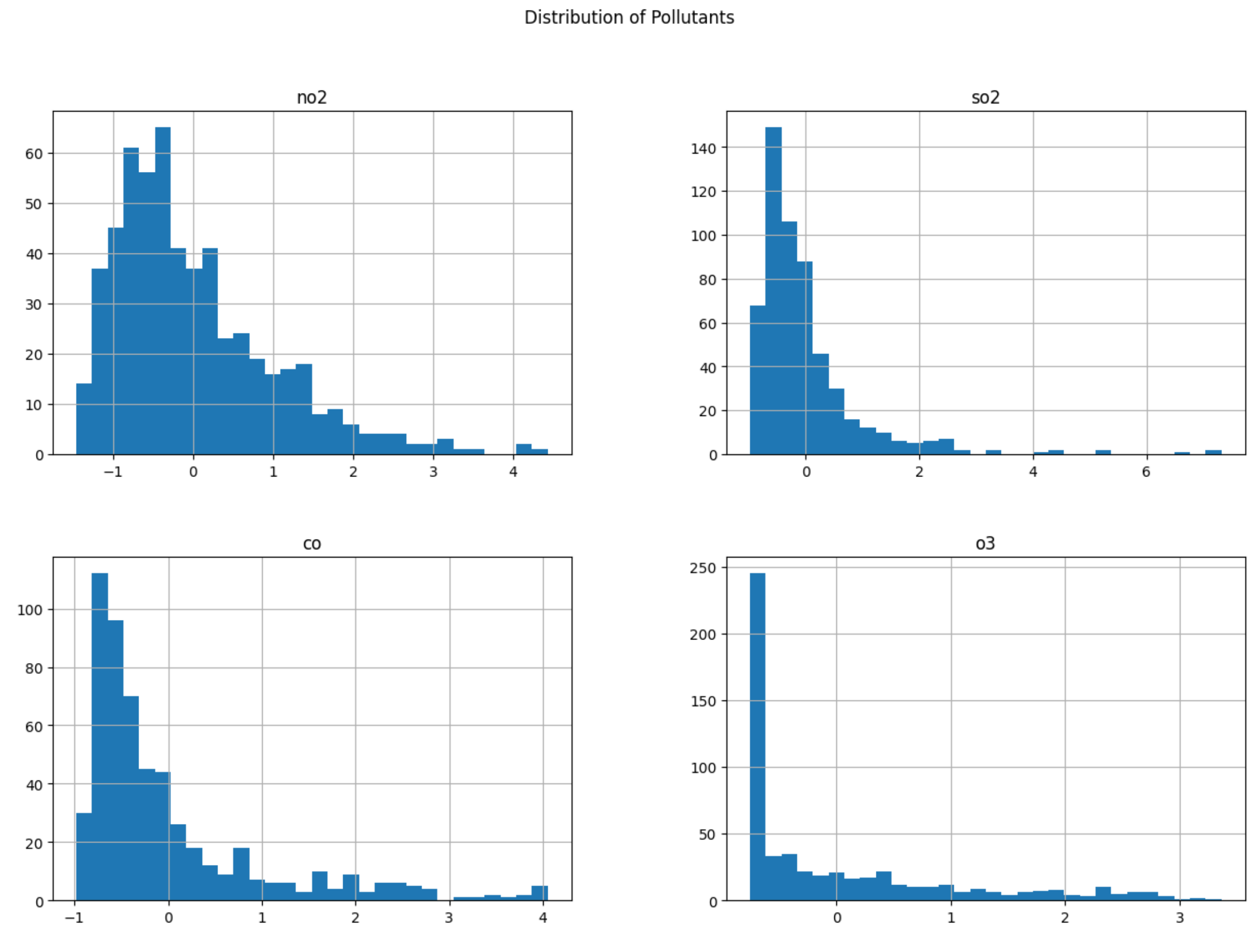
```
#  Columns with missing values
print(df.isnull().sum())
```

```
date     0
co       0
no       0
no2      0
o3       0
so2      0
pm2_5    0
pm10     0
nh3      0
year     0
month    0
day      0
dtype: int64
```

```
# Percentage of missing values in each column
print(df.isnull().mean() * 100)
```

```
date     0.0
co       0.0
```

```
no         0.0
no2        0.0
o3         0.0
so2        0.0
pm2_5      0.0
pm10       0.0
nh3        0.0
year       0.0
month      0.0
day        0.0
dtype: float64

# Distribution of each pollutant
df[pollutants].hist(bins=30, figsize=(15, 10))
plt.suptitle('Distribution of Pollutants')
plt.show()
```

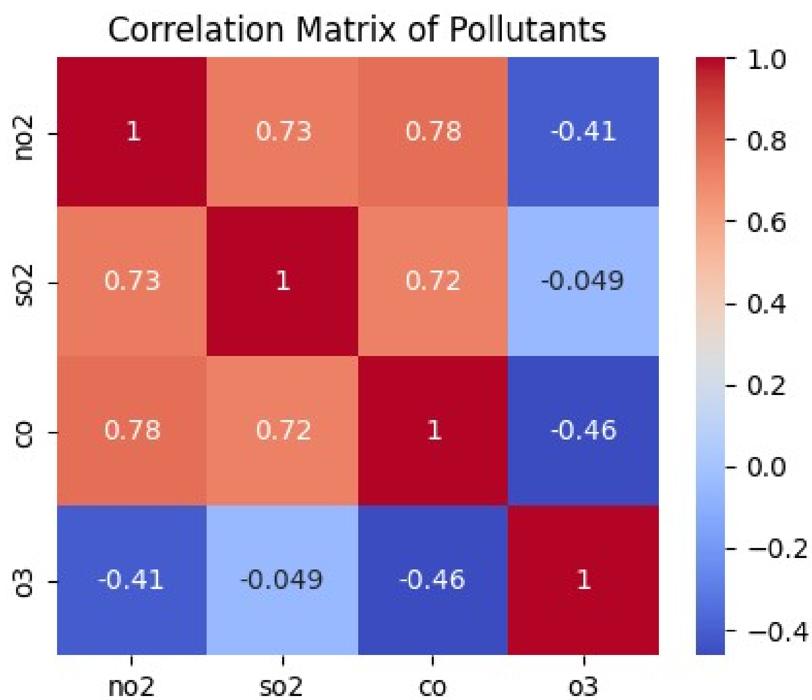



```
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation matrix
```

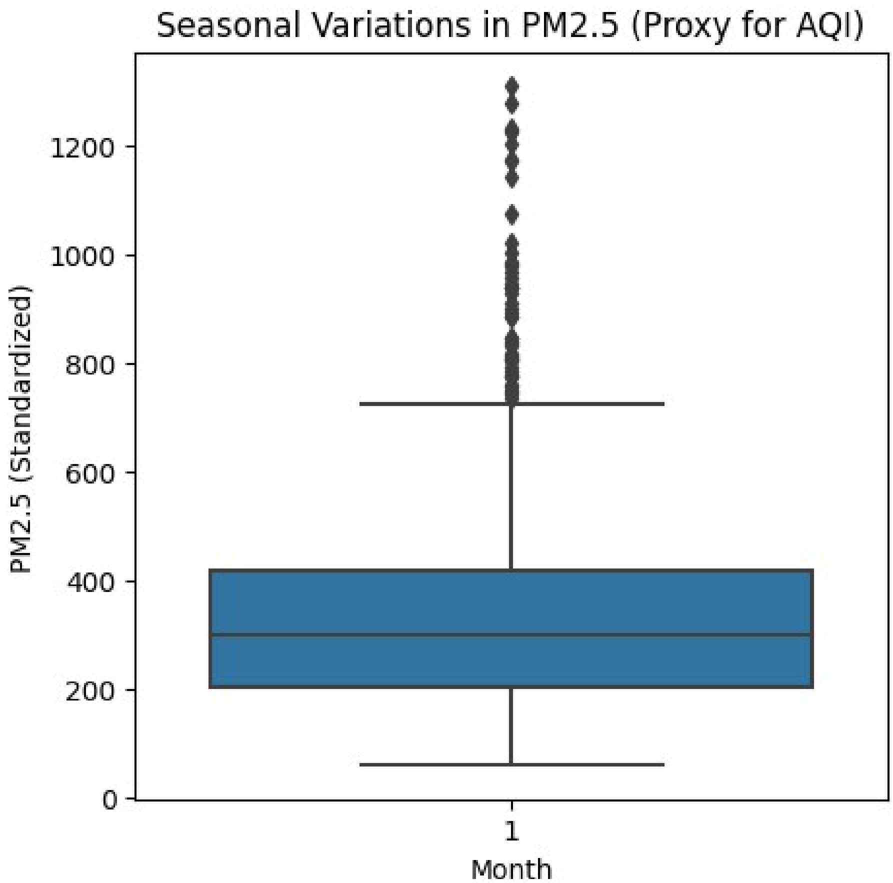
```
corr_matrix = df[pollutants].corr()
print(corr_matrix)
```

```
          no2       so2        co        o3
no2  1.000000  0.734961  0.776402 -0.407177
so2  0.734961  1.000000  0.716831 -0.049158
co   0.776402  0.716831  1.000000 -0.463082
o3  -0.407177 -0.049158 -0.463082  1.000000
```

```
# Heatmap of the correlation matrix
plt.figure(figsize=(5,4))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Pollutants')
plt.show()
```

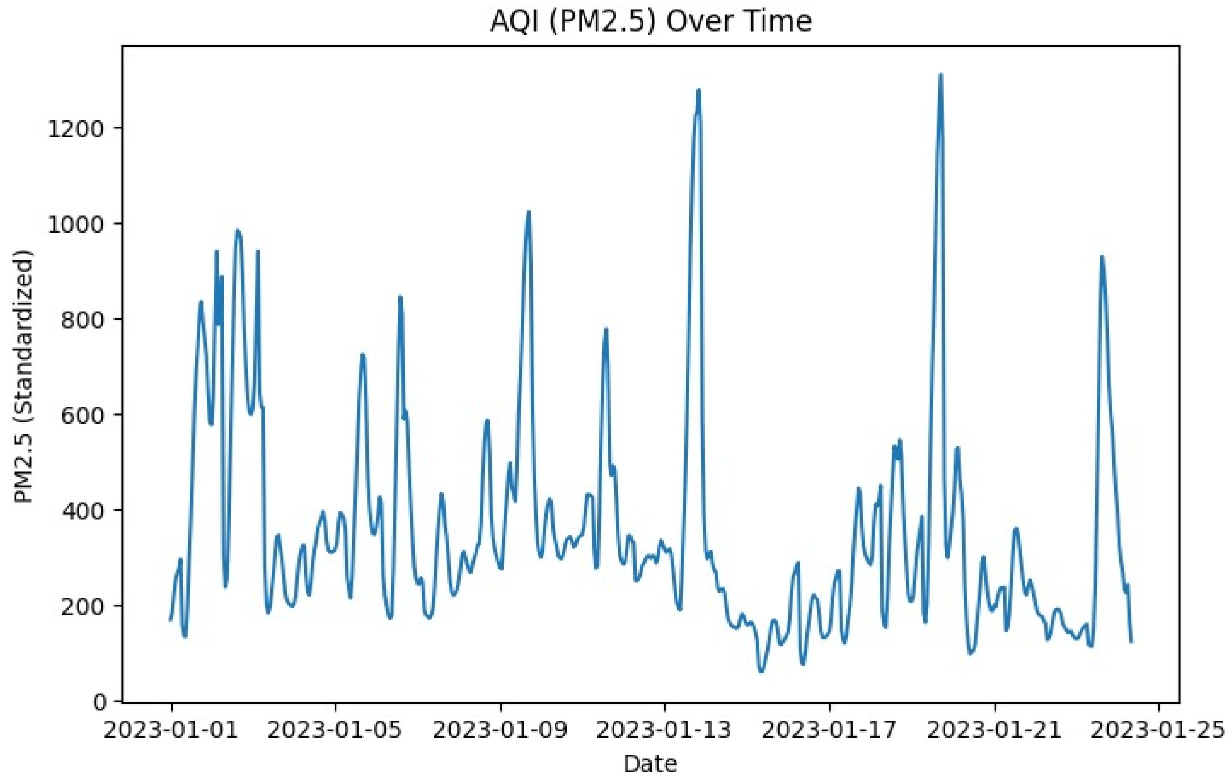



```
# Boxplot of AQI by month
plt.figure(figsize=(5,5))
sns.boxplot(x='month', y='pm2_5', data=df)  # Using pm2.5 as a proxy for AQI
plt.title('Seasonal Variations in PM2.5 (Proxy for AQI)')
plt.xlabel('Month')
plt.ylabel('PM2.5 (Standardized)')
plt.show()
```



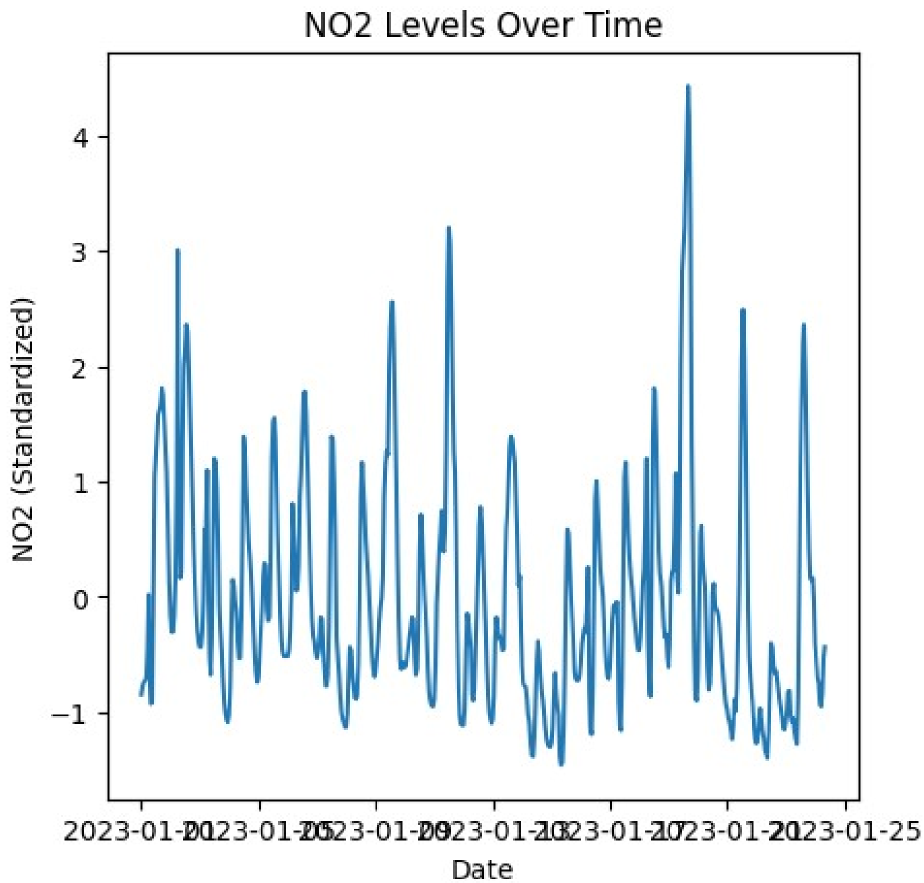

```
# Distribution of AQI over time
plt.figure(figsize=(8,5))
sns.lineplot(x='date', y='pm2_5', data=df)
plt.title('AQI (PM2.5) Over Time')
plt.xlabel('Date')
plt.ylabel('PM2.5 (Standardized)')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

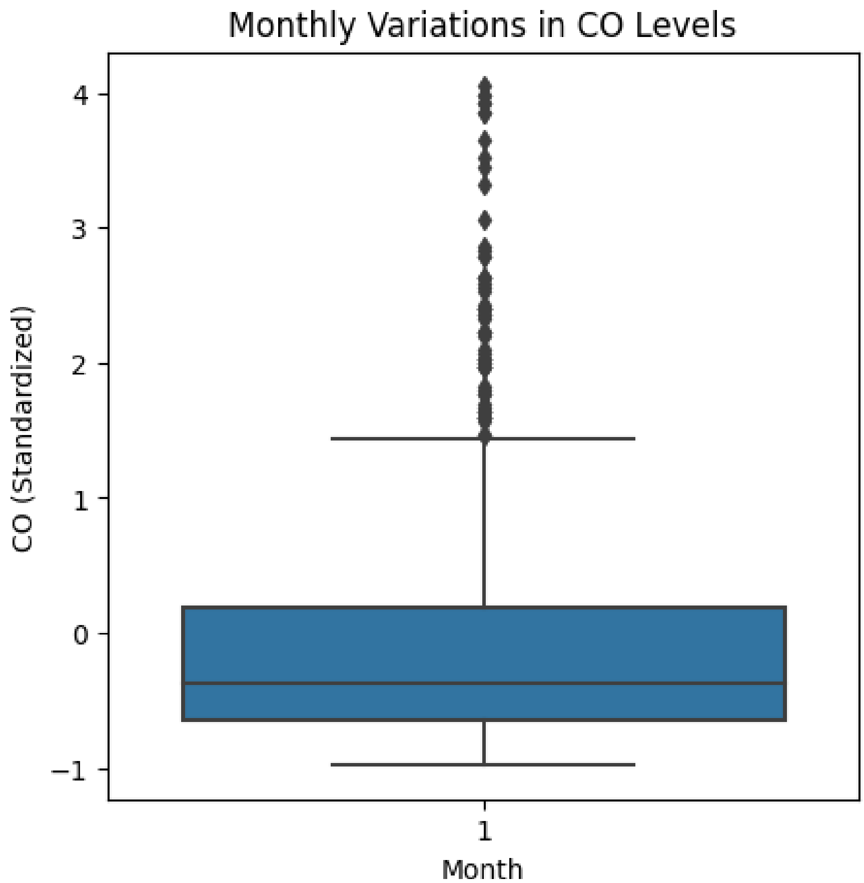



```
# NO2 levels over time
plt.figure(figsize=(5,5))
sns.lineplot(x='date', y='no2', data=df)
plt.title('NO2 Levels Over Time')
plt.xlabel('Date')
plt.ylabel('NO2 (Standardized)')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
# Distribution of CO levels by month
plt.figure(figsize=(5,5))
sns.boxplot(x='month', y='co', data=df)
plt.title('Monthly Variations in CO Levels')
plt.xlabel('Month')
plt.ylabel('CO (Standardized)')
plt.show()
```



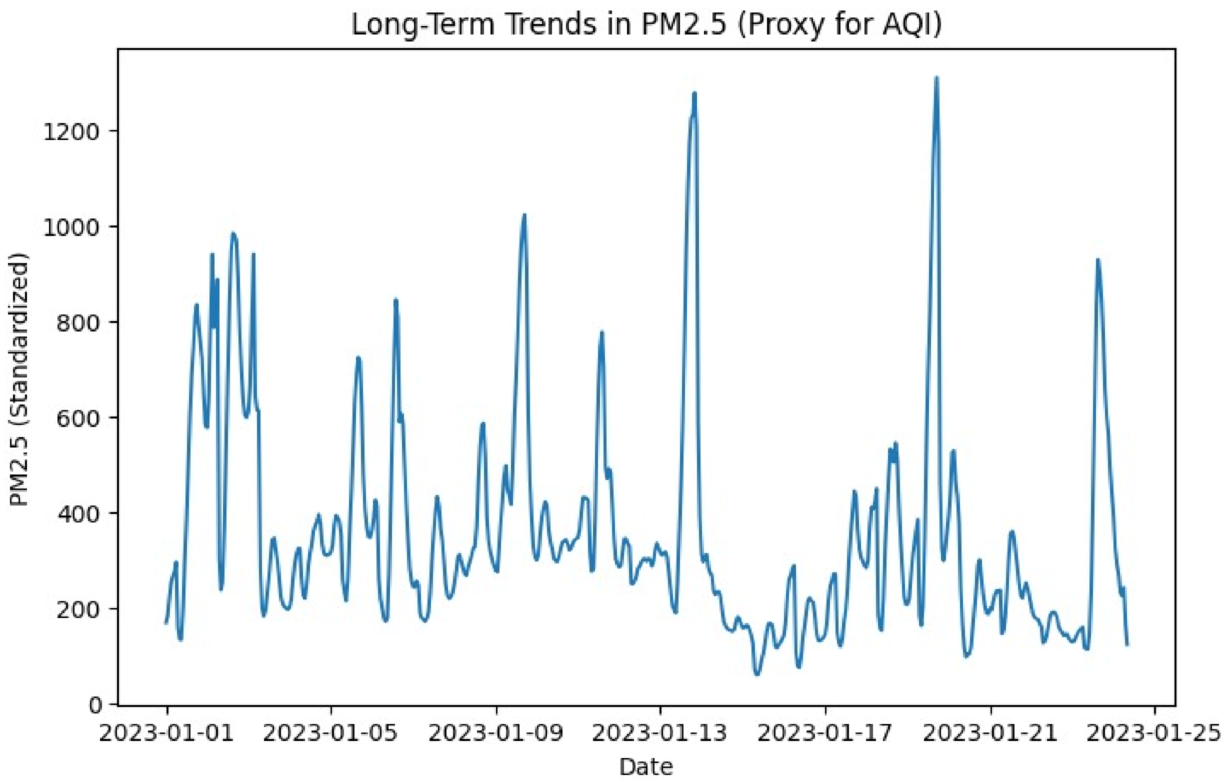

```
# Highest recorded SO2 levels and when they occurred
max_so2 = df.loc[df['so2'].idxmax()]
print(f"Highest SO2 level: {max_so2['so2']} on {max_so2['date']}")
```

```
Highest SO2 level: 7.317668558293588 on 2023-01-19 17:00:00
```

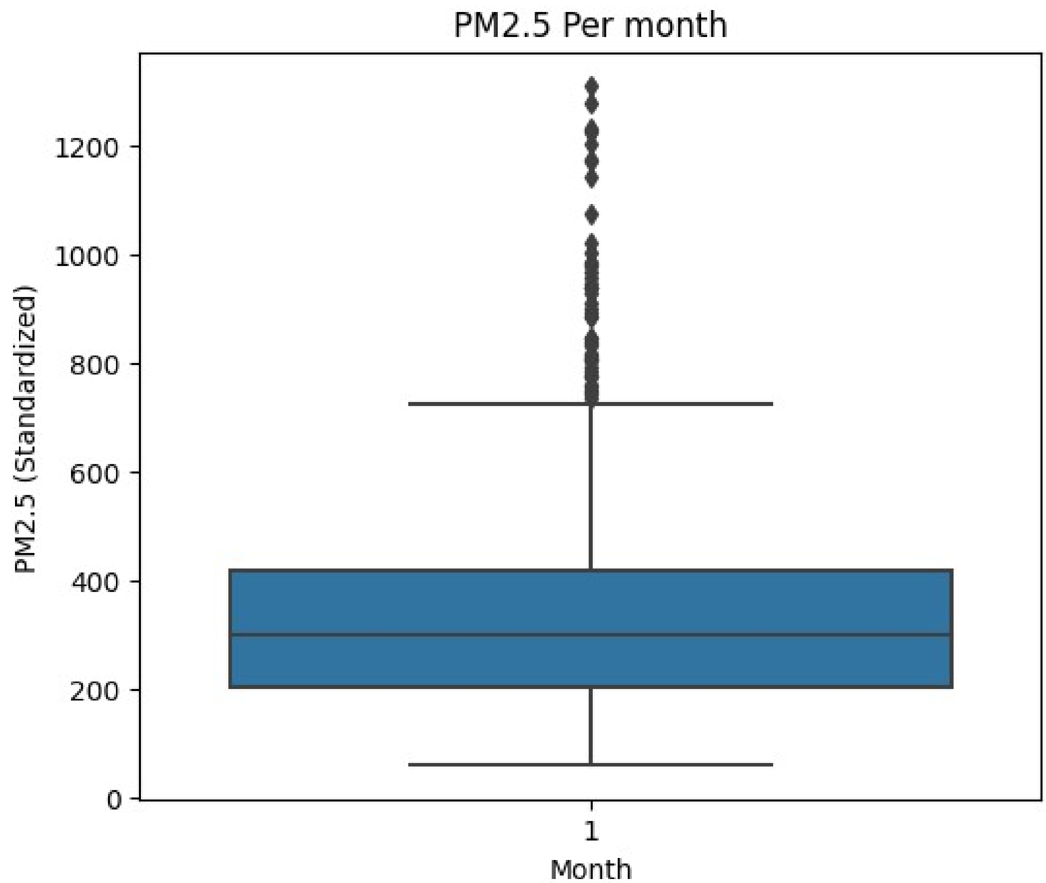
```
# Long-term trends in AQI
plt.figure(figsize=(8,5))
sns.lineplot(x='date', y='pm2_5', data=df)
plt.title('Long-Term Trends in PM2.5 (Proxy for AQI)')
plt.xlabel('Date')
plt.ylabel('PM2.5 (Standardized)')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
```

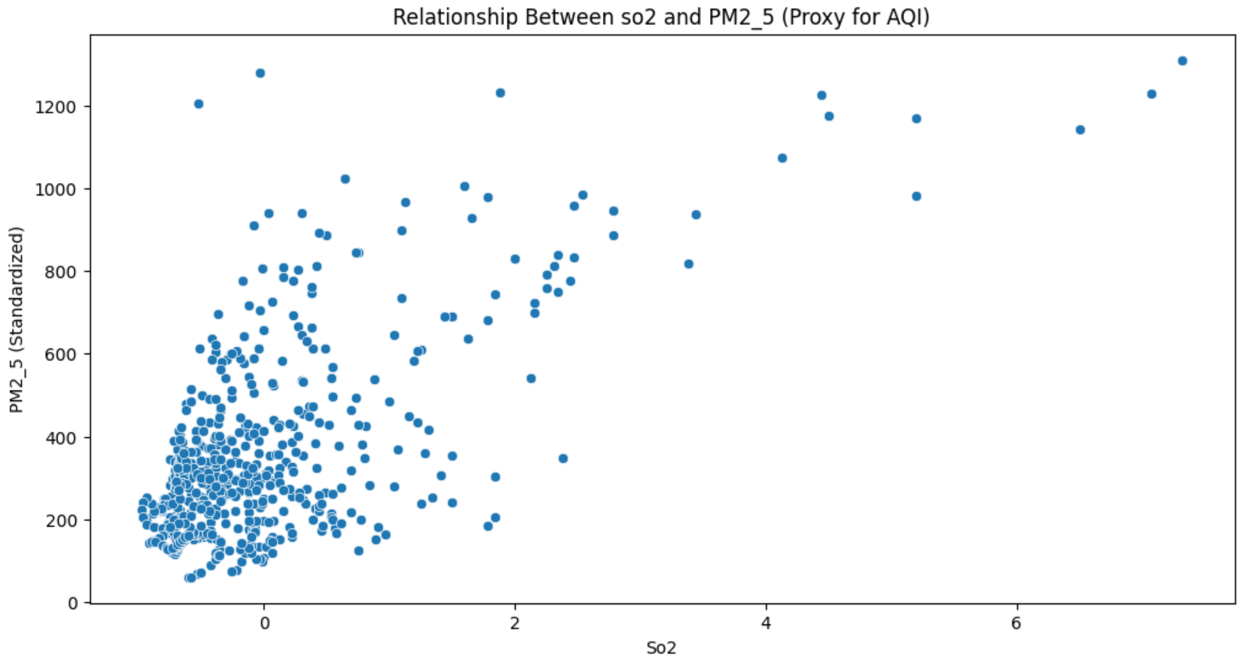
```
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
plt.figure(figsize=(6,5))
sns.boxplot(x='month', y='pm2_5', data=df)
plt.title('PM2.5 Per month')
plt.xlabel('Month')
plt.ylabel('PM2.5 (Standardized)')
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='so2', y='pm2_5', data=df)
plt.title('Relationship Between so2 and PM2_5 (Proxy for AQI)')
plt.xlabel('So2')
plt.ylabel('PM2_5 (Standardized)')
plt.show()
```




```
# Days exceeding hazardous AQI level (assuming PM2.5 level of 300 as
hazardous)
hazardous_days = df[df['pm2_5'] > 300].shape[0]
print(f"Number of hazardous days: {hazardous_days}")
```

```
Number of hazardous days: 282
```

```
# Average AQI level per year
average_aqi_per_year = df.groupby('year')['pm2_5'].mean()
print(average_aqi_per_year)
```

```
year
2023    358.256364
Name: pm2_5, dtype: float64
```

```
# Top 10 days with the worst AQI levels
top_10_worst_days = df.nlargest(10, 'pm2_5')
print(top_10_worst_days[['date', 'pm2_5']])
```

```
                   date    pm2_5
449 2023-01-19 17:00:00  1310.20
308 2023-01-13 20:00:00  1278.35
307 2023-01-13 19:00:00  1232.62
448 2023-01-19 16:00:00  1228.04
306 2023-01-13 18:00:00  1225.39
309 2023-01-13 21:00:00  1204.33
305 2023-01-13 17:00:00  1174.70
450 2023-01-19 18:00:00  1170.46
447 2023-01-19 15:00:00  1142.61
304 2023-01-13 16:00:00  1074.91
```

```
df.columns
```

```
Index(['date', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3',
'year',
       'month', 'day'],
      dtype='object')
```

```
# Average daily AQI for each month
daily_avg_aqi = df.groupby(['year', 'month', 'day'])
['pm2_5'].mean().reset_index()
print(daily_avg_aqi.head())
```

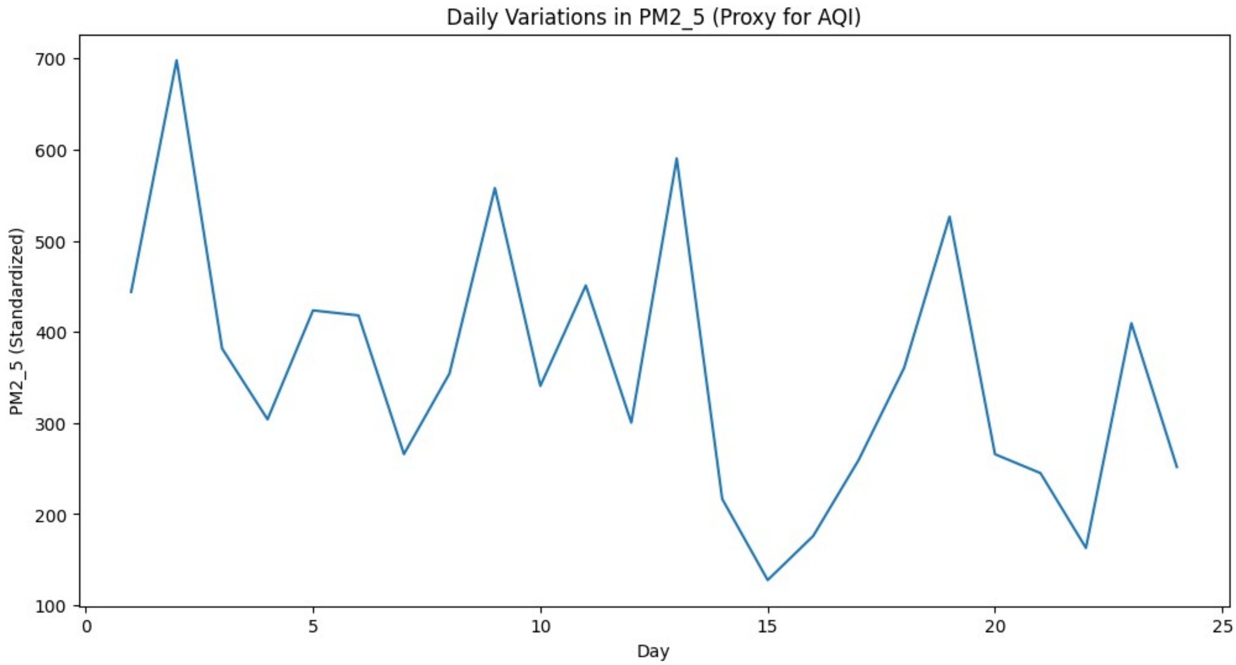
```
   year  month  day       pm2_5
0  2023      1    1  443.940000
1  2023      1    2  698.104167
2  2023      1    3  381.810417
3  2023      1    4  304.021667
4  2023      1    5  423.604583
```

```
# Most and least polluted months
monthly_avg_aqi = df.groupby('month')
['pm2_5'].mean().sort_values(ascending=False)
print(monthly_avg_aqi)
```

```
month
1    358.256364
Name: pm2_5, dtype: float64
```

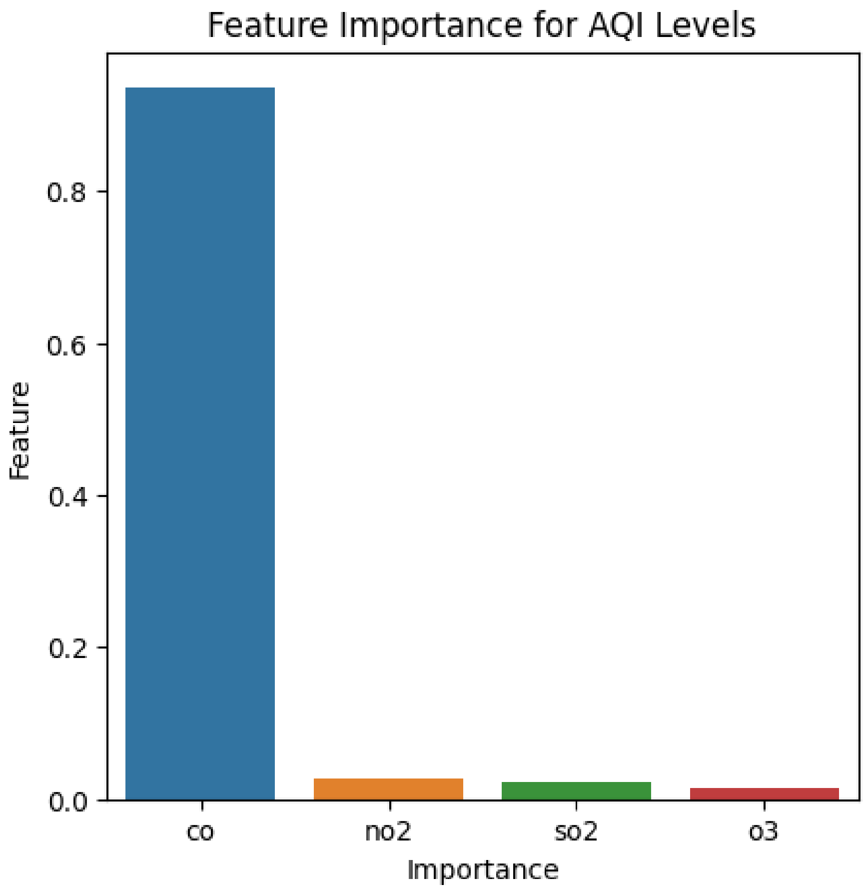
```
# AQI variation during different times of the day
hourly_avg_aqi = df.groupby('day')['pm2_5'].mean()
plt.figure(figsize=(12, 6))
sns.lineplot(x=hourly_avg_aqi.index, y=hourly_avg_aqi.values)
plt.title('Daily Variations in PM2_5 (Proxy for AQI)')
plt.xlabel('Day')
plt.ylabel('PM2_5 (Standardized)')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
# 35. Top contributing factors to high AQI levels (feature importance analysis)
from sklearn.ensemble import RandomForestRegressor

# Prepare the data
X = df[pollutants]
y = df['pm2_5']

# Fit a Random Forest model
model = RandomForestRegressor()
model.fit(X, y)

# Feature importance
feature_importance = pd.Series(model.feature_importances_,
index=pollutants).sort_values(ascending=False)
print(feature_importance)

co      0.935362
no2     0.027649
so2     0.022677
o3      0.014311
dtype: float64

# Plot feature importance
plt.figure(figsize=(5,5))
sns.barplot(y=feature_importance.values, x=feature_importance.index)
plt.title('Feature Importance for AQI Levels')
plt.xlabel('Importance')
```

```
plt.ylabel('Feature')
plt.show()
```




```
#Long-term trends in pollutant levels
plt.figure(figsize=(7, 6))
plt.figure(figsize=(14, 12))

# Create a subplot for PM2.5
plt.subplot(4, 1, 1)
sns.lineplot(x='date', y='pm2_5', data=df)
plt.title('Long-Term Trends in PM2.5 Levels')
plt.xlabel('Date')
plt.ylabel('PM2.5 (Standardized)')

# Create a subplot for PM10
plt.subplot(4, 1, 2)
sns.lineplot(x='date', y='pm10', data=df)
plt.title('Long-Term Trends in PM10 Levels')
plt.xlabel('Date')
plt.ylabel('PM10 (Standardized)')

# Create a subplot for NO2
```

```
plt.subplot(4, 1, 3)
sns.lineplot(x='date', y='no2', data=df)
plt.title('Long-Term Trends in NO2 Levels')
plt.xlabel('Date')
plt.ylabel('NO2 (Standardized)')

# Create a subplot for SO2
plt.subplot(4, 1, 4)
sns.lineplot(x='date', y='so2', data=df)
plt.title('Long-Term Trends in SO2 Levels')
plt.xlabel('Date')
plt.ylabel('SO2 (Standardized)')

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

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
```

```
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):

<Figure size 700x600 with 0 Axes>
```


```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Prepare data
X = df[pollutants]
y = df['pm2_5']
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Predict and evaluate
y_pred = lr.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred)}")
print(f"R-squared Score: {r2_score(y_test, y_pred)}")
```

```
Mean Squared Error: 4254.347236790641
R-squared Score: 0.8837485822443608
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

# Initialize models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(),
    'Support Vector Regression': SVR(),
    'K-Neighbors Regression': KNeighborsRegressor()
}

# Evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{name} - Mean Squared Error: {mse}, R-squared Score:
{r2}")
```

```
Linear Regression - Mean Squared Error: 4254.347236790641, R-squared
Score: 0.8837485822443608
Random Forest - Mean Squared Error: 3948.004758304696, R-squared
Score: 0.8921194898032916
Support Vector Regression - Mean Squared Error: 29188.86257929797, R-
squared Score: 0.20240486526960522
K-Neighbors Regression - Mean Squared Error: 4277.278643044247, R-
squared Score: 0.8831219741327664
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Binning AQI levels into categories (e.g., good, moderate, unhealthy)
df['aqi_category'] = pd.cut(df['pm2_5'], bins=[-np.inf, 50, 100, 150,
```

```python
200, 300, np.inf], labels=[0, 1, 2, 3, 4, 5])

# Prepare data
X = df[pollutants]
y = df['aqi_category']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train classifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)

# Predict and evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

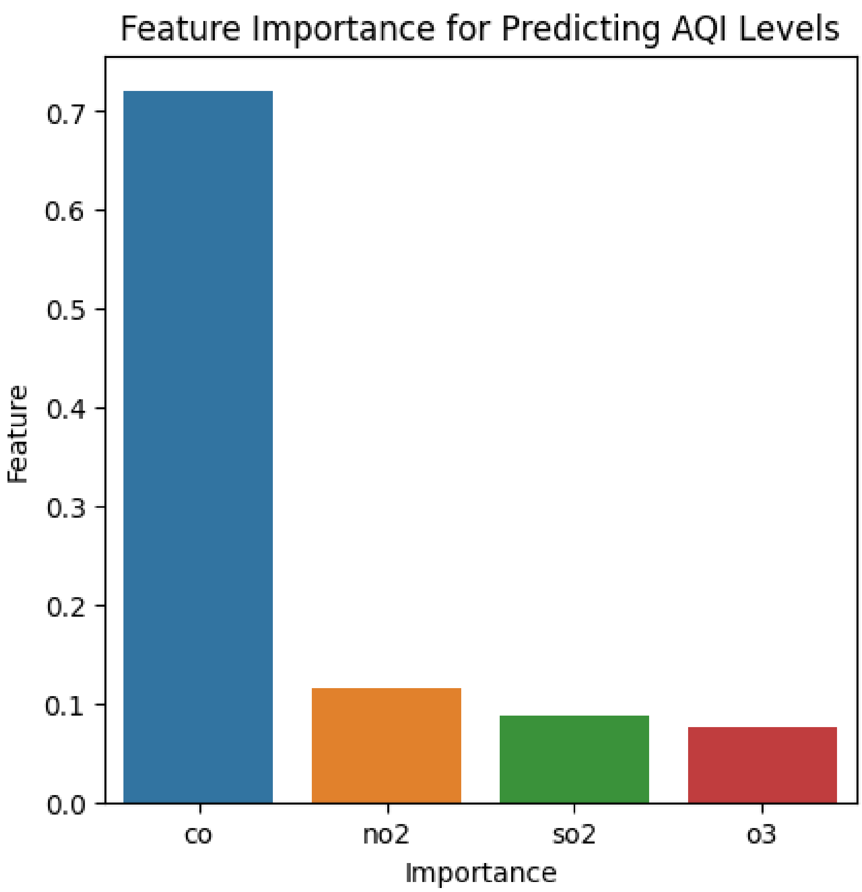
```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.58      0.78      0.67         9
           3       0.47      0.54      0.50        13
           4       0.69      0.53      0.60        38
           5       0.81      0.90      0.85        51

    accuracy                           0.71       113
   macro avg       0.51      0.55      0.52       113
weighted avg       0.70      0.71      0.70       113
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/
_classification.py:1344: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classificatio
n.py:1344: UndefinedMetricWarning: Precision and F-score are ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```python
# Feature importance from the RandomForestRegressor
model = RandomForestRegressor()
model.fit(X_train, y_train)
feature_importance = pd.Series(model.feature_importances_,
```

```
index=pollutants).sort_values(ascending=False)
print(feature_importance)

# Plot feature importance
plt.figure(figsize=(5,5))
sns.barplot(y=feature_importance.values, x=feature_importance.index)
plt.title('Feature Importance for Predicting AQI Levels')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

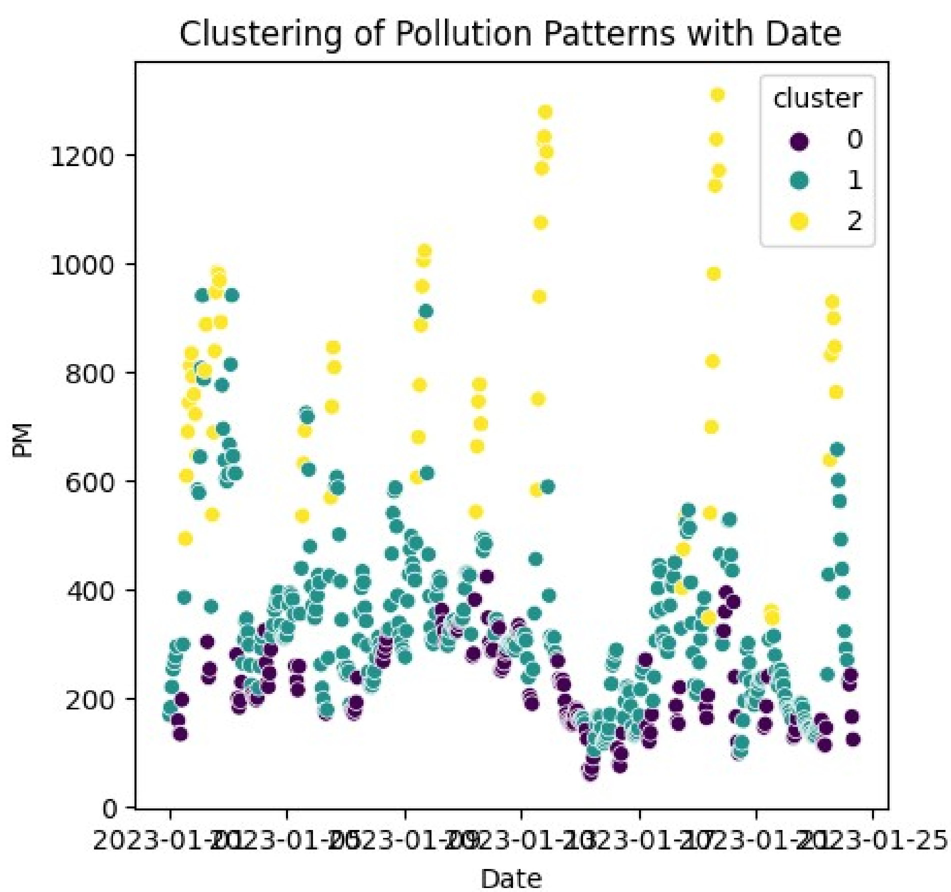
```
co      0.720518
no2     0.115221
so2     0.087636
o3      0.076626
dtype: float64
```




```
from sklearn.cluster import KMeans

# Fit KMeans clustering
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit_predict(X)
```

```
# Add cluster information to the dataset
df['cluster'] = clusters

# Plot clusters
plt.figure(figsize=(5,5))
sns.scatterplot(x='date', y='pm2_5', hue='cluster', palette='viridis',
data=df)
plt.title('Clustering of Pollution Patterns with Date')
plt.xlabel('Date')
plt.ylabel('PM')
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```




```
from sklearn.ensemble import GradientBoostingRegressor

# Train Gradient Boosting model
```

```
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)

# Predict and evaluate
y_pred = gbr.predict(X_test)
print(f"Gradient Boosting - Mean Squared Error:
{mean_squared_error(y_test, y_pred)}")
print(f"R-squared Score: {r2_score(y_test, y_pred)}")
```

```
Gradient Boosting - Mean Squared Error: 0.3680296685022231
R-squared Score: 0.6423069845406542
```

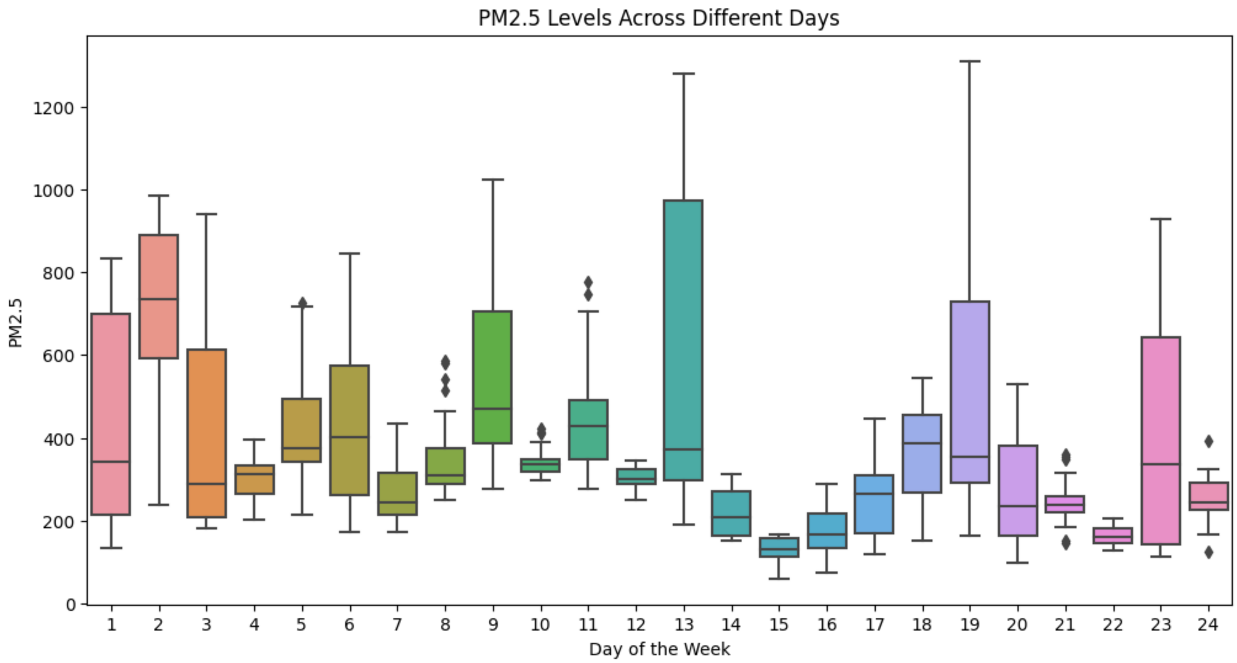
```
from sklearn.decomposition import PCA

# Apply PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)

# Train and evaluate model with PCA components
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)
model = RandomForestRegressor()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"Mean Squared Error with PCA: {mean_squared_error(y_test,
y_pred)}")
print(f"R-squared Score with PCA: {r2_score(y_test, y_pred)}")
```

```
Mean Squared Error with PCA: 0.4209964601769911
R-squared Score with PCA: 0.5908278428984626
```

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=RandomForestRegressor(),
param_grid=param_grid, cv=3, n_jobs=-1, scoring='r2')
grid_search.fit(X_train, y_train)

# Best hyperparameters
print(f"Best Hyperparameters: {grid_search.best_params_}")
```

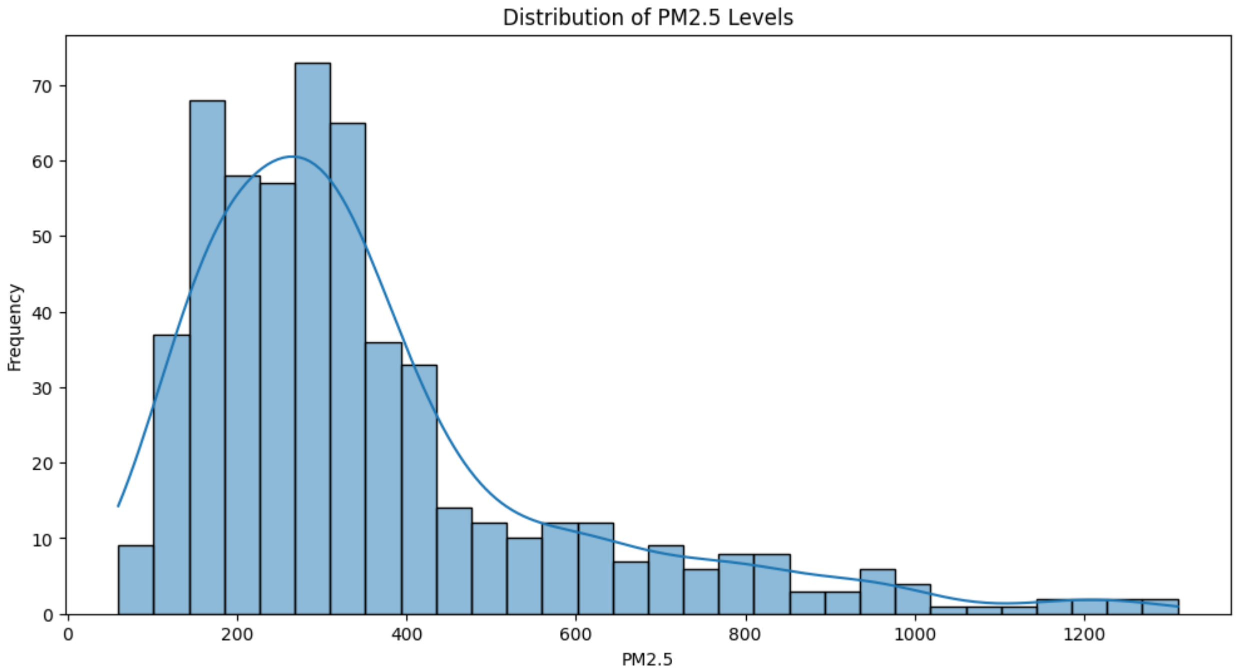
```
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 10,
'n_estimators': 100}
```

```
plt.figure(figsize=(12, 6))
sns.boxplot(x='day', y='pm2_5', data=df)
plt.title('PM2.5 Levels Across Different Days')
plt.xlabel('Day of the Week')
plt.ylabel('PM2.5')
plt.show()
```

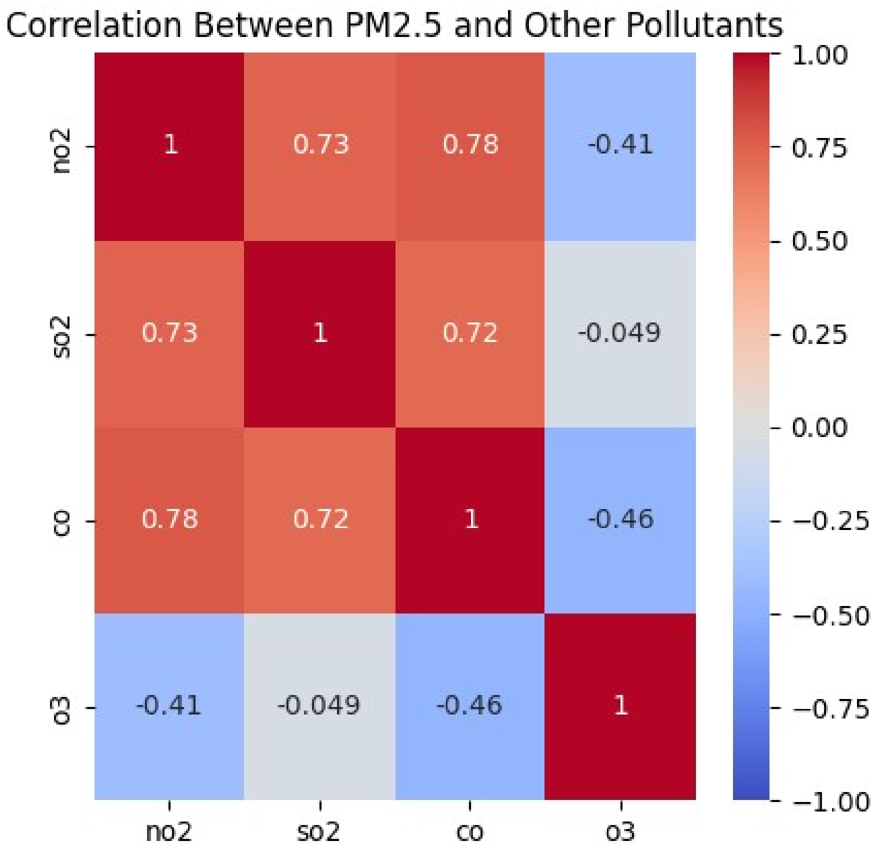



```
plt.figure(figsize=(12, 6))
sns.histplot(df['pm2_5'], bins=30, kde=True)
plt.title('Distribution of PM2.5 Levels')
plt.xlabel('PM2.5')
plt.ylabel('Frequency')
plt.show()
```

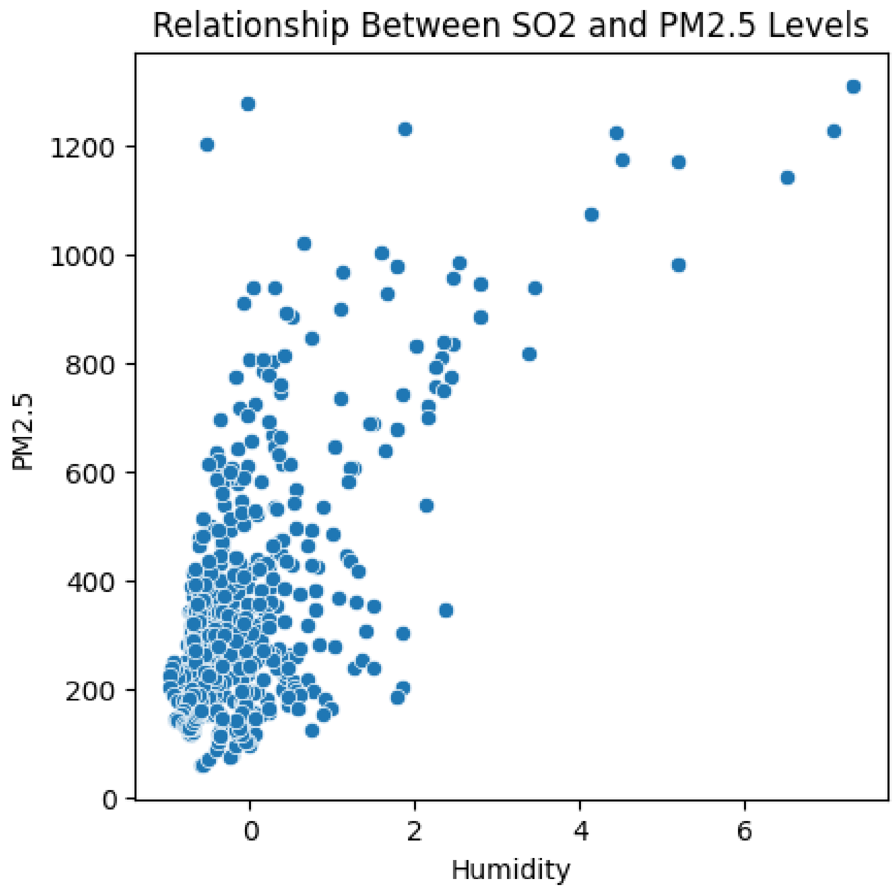
```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



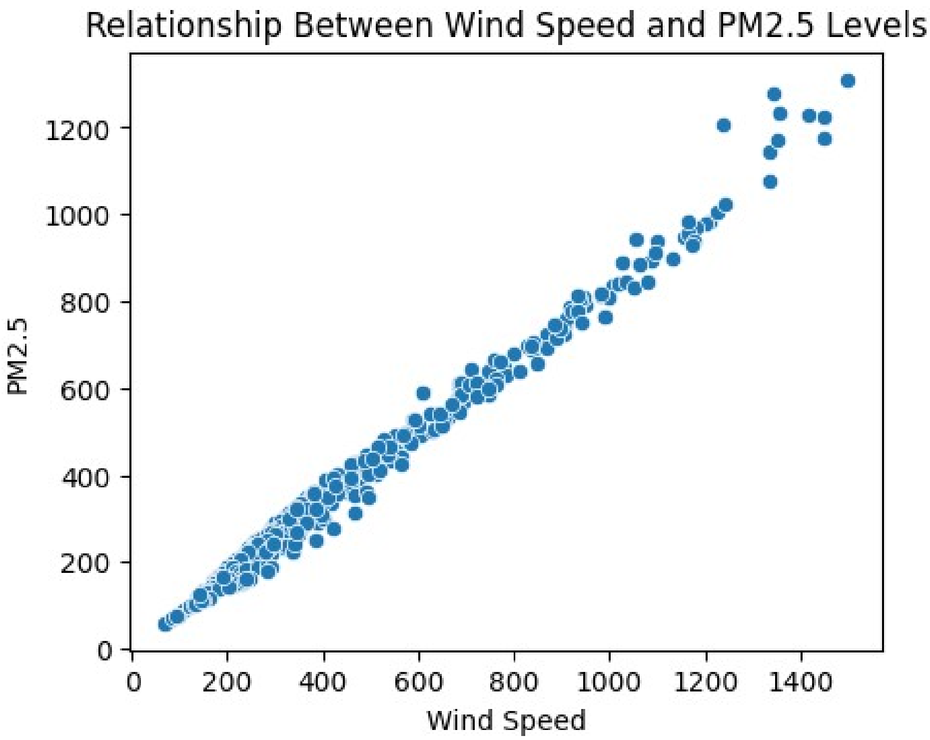

```
plt.figure(figsize=(5,5))
sns.heatmap(df[pollutants].corr(), annot=True, cmap='coolwarm', vmin=-
1, vmax=1)
plt.title('Correlation Between PM2.5 and Other Pollutants')
plt.show()
```




```
plt.figure(figsize=(5,5))
sns.scatterplot(x='so2', y='pm2_5', data=df)
plt.title('Relationship Between SO2 and PM2.5 Levels')
plt.xlabel('Humidity')
plt.ylabel('PM2.5')
plt.show()
```




```
plt.figure(figsize=(5,4))
sns.scatterplot(x='pm10', y='pm2_5', data=df)
plt.title('Relationship Between Wind Speed and PM2.5 Levels')
plt.xlabel('Wind Speed')
plt.ylabel('PM2.5')
plt.show()
```


Relationship Between Wind Speed and PM2.5 Levels
PM2.5
Wind Speed
0
200
400
600
800
1000
1200
1400