AIR QUALITY INDEX PREDICTION USING MACHINE LEARNING MODEL(RANDOM FOREST ALGORITHM)

EXPLORATORY DATA ANALYSIS

DATA PREPROCESSING

In [1]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# reading the air_quality dataset
air_quality=pd.read_csv("air_quality.csv")
air_quality
```

Out[2]:

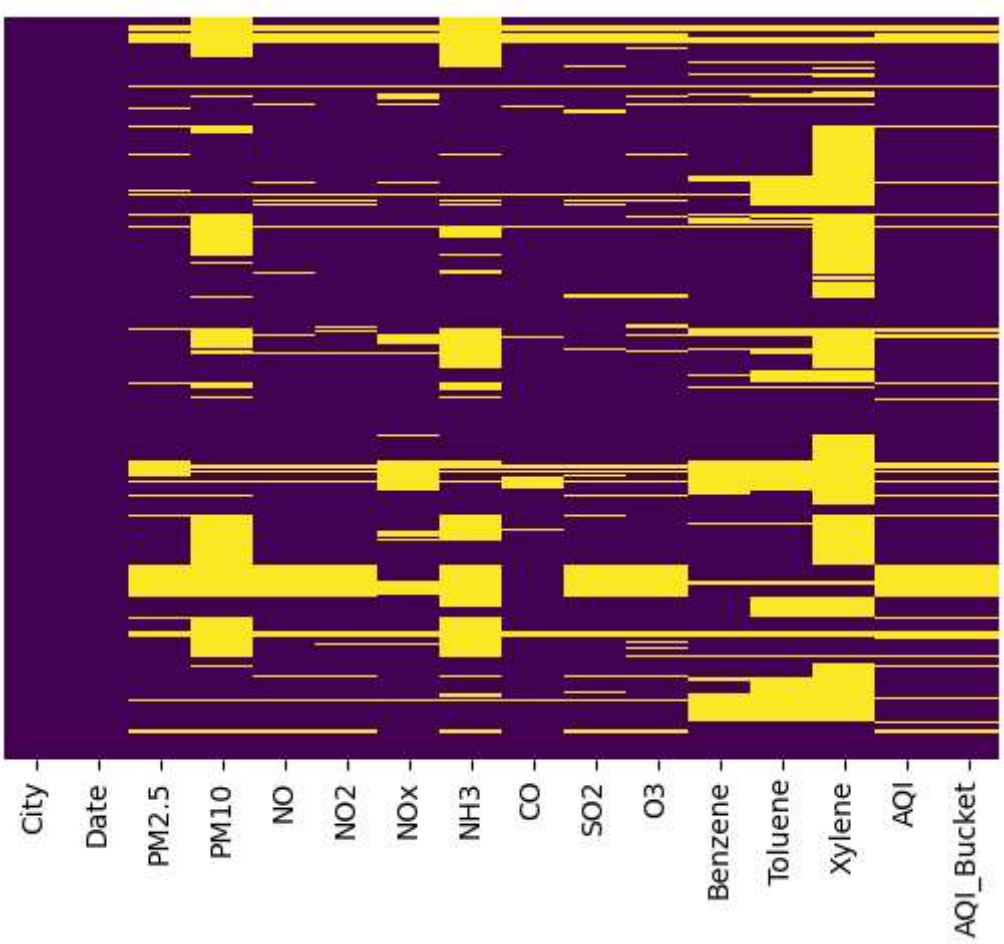
	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benz
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	
...	...	...	...	...	...	...	...	...	...	...	...	...
29526	Visakhapatnam	2020-06-27	15.02	50.94	7.68	25.06	19.54	12.47	0.47	8.55	23.30	
29527	Visakhapatnam	2020-06-28	24.38	74.09	3.42	26.06	16.53	11.99	0.52	12.72	30.14	
29528	Visakhapatnam	2020-06-29	22.91	65.73	3.45	29.53	18.33	10.71	0.48	8.42	30.96	
29529	Visakhapatnam	2020-06-30	16.64	49.97	4.05	29.26	18.80	10.03	0.52	9.84	28.30	
29530	Visakhapatnam	2020-07-01	15.00	66.00	0.40	26.85	14.05	5.20	0.59	2.10	17.05	

29531 rows × 16 columns

Visualizing the features to see the null values by using heatmap

In [3]:
```
1 sns.heatmap(air_quality.isnull(),yticklabels=False,cbar=False,cmap='viridi
```

Out[3]: <Axes: >



In [4]:
```
1 print(air_quality.isnull().sum())
```

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

In [5]:
```
(air_quality.isnull().sum()/air_quality.shape[0]*100).sort_values(ascendin
```

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

In [6]:
```
air_quality.describe()
```

Out[6]:

	PM2.5	PM10	NO	NO2	NOx	NH3	
count	24933.000000	18391.000000	25949.000000	25946.000000	25346.000000	19203.000000	274
mean	67.450578	118.127103	17.574730	28.560659	32.309123	23.483476	
std	64.661449	90.605110	22.785846	24.474746	31.646011	25.684275	
min	0.040000	0.010000	0.020000	0.010000	0.000000	0.010000	
25%	28.820000	56.255000	5.630000	11.750000	12.820000	8.580000	
50%	48.570000	95.680000	9.890000	21.690000	23.520000	15.850000	
75%	80.590000	149.745000	19.950000	37.620000	40.127500	30.020000	
max	949.990000	1000.000000	390.680000	362.210000	467.630000	352.890000	1

In [7]:
```
#converting dtype of date column to datetime
air_quality['Date']=air_quality['Date'].apply(pd.to_datetime)
#setting date column as index
air_quality.set_index('Date',inplace=True)
```

In [8]:
```
air_quality.columns
```

Out[8]:
```
Index(['City', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3',
       'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
      dtype='object')
```

In [9]:
```
1 air_quality.iloc[:, 1:13] = air_quality.groupby("City").transform(lambda x
```

```
C:\Users\vanaja\AppData\Local\Temp\ipykernel_8352\2326754928.py:1: FutureWarn
ing: Dropping invalid columns in DataFrameGroupBy.transform is deprecated. In
a future version, a TypeError will be raised. Before calling .transform, sele
ct only columns which should be valid for the function.
  air_quality.iloc[:, 1:13] = air_quality.groupby("City").transform(lambda x:
x.fillna(x.mean()))
```

In [10]:
```
1 air_quality
```

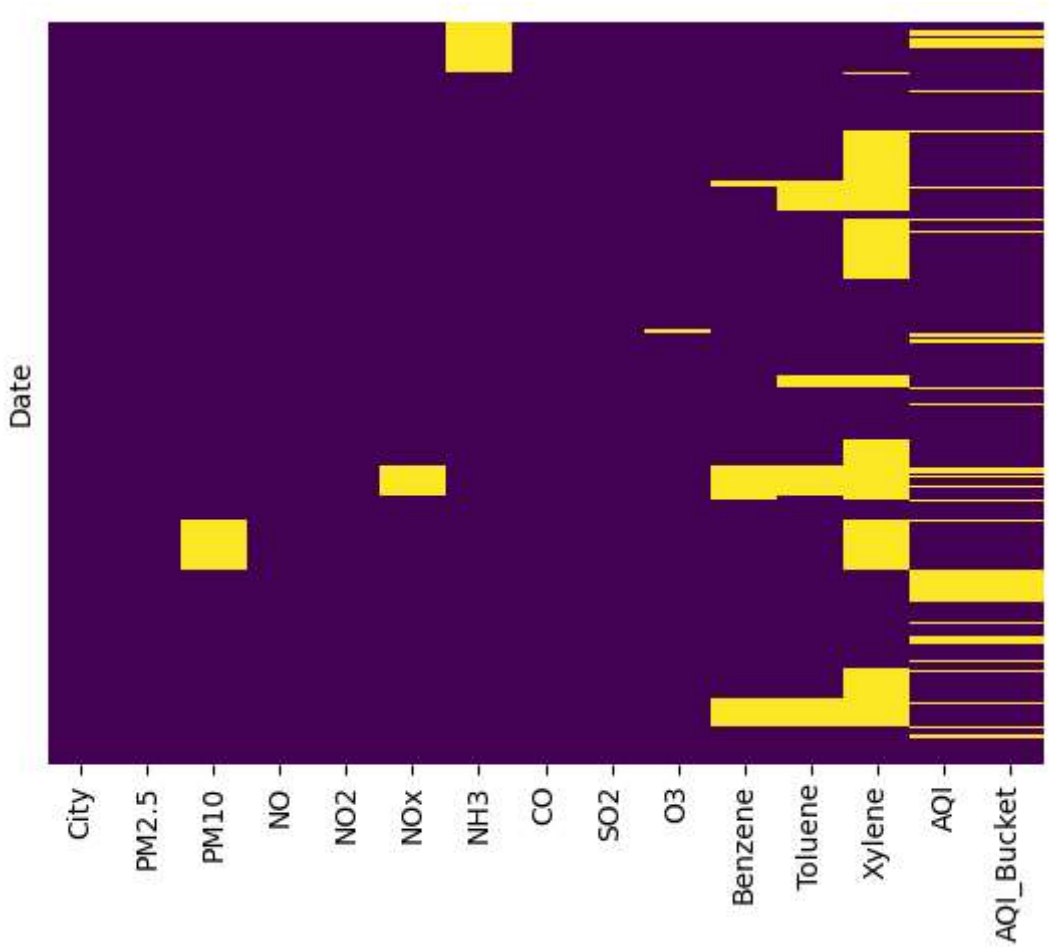
Out[10]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Be
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.0
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.6
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.8
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.4
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.0
...	...	...	...	...	...	...	...	...	...	...	...
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.47	0.47	8.55	23.30	2.2
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.99	0.52	12.72	30.14	0.7
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.71	0.48	8.42	30.96	0.0
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.03	0.52	9.84	28.30	0.0
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.20	0.59	2.10	17.05	3.8

29531 rows × 15 columns

In [11]:
```
1 sns.heatmap(air_quality.isnull(),yticklabels=False,cbar=False,cmap='viridi
```

Out[11]: <Axes: ylabel='Date'>



In [12]:
```
air_quality.iloc[:, 1:13] = air_quality.iloc[:, 1:13].fillna(air_quality.i
air_quality
```

Out[12]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	23.024137	0.92	27.64	133.36
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	23.024137	0.97	24.55	34.06
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	23.024137	17.40	29.07	30.70
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	23.024137	1.70	18.59	36.08
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	23.024137	22.10	39.33	39.31
...	...	...	...	...	...	...	...	...	...	...
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05

29531 rows × 15 columns

In [13]:
```
sns.heatmap(air_quality.isnull(),yticklabels=False,cbar=False,cmap='viridi
```

Out[13]:
```
<Axes: ylabel='Date'>
```

In [14]:

```
# PM10 Sub-Index calculation
def get_PM10_subindex(x):
    if x <= 50:
        return x
    elif x > 50 and x <= 100:
        return x
    elif x > 100 and x <= 250:
        return 100 + (x - 100) * 100 / 150
    elif x > 250 and x <= 350:
        return 200 + (x - 250)
    elif x > 350 and x <= 430:
        return 300 + (x - 350) * 100 / 80
    elif x > 430:
        return 400 + (x - 430) * 100 / 80
    else:
        return 0

air_quality["PM10_SubIndex"] = air_quality["PM10"].astype(int).apply(lambd

# PM2.5 Sub-Index calculation
def get_PM25_subindex(x):
    if x <= 30:
        return x * 50 / 30
    elif x > 30 and x <= 60:
        return 50 + (x - 30) * 50 / 30
    elif x > 60 and x <= 90:
        return 100 + (x - 60) * 100 / 30
    elif x > 90 and x <= 120:
        return 200 + (x - 90) * 100 / 30
    elif x > 120 and x <= 250:
        return 300 + (x - 120) * 100 / 130
    elif x > 250:
        return 400 + (x - 250) * 100 / 130
    else:
        return 0

air_quality["PM2.5_SubIndex"] = air_quality["PM2.5"].astype(int).apply(lam


# SO2 Sub-Index calculation
def get_SO2_subindex(x):
    if x <= 40:
        return x * 50 / 40
    elif x > 40 and x <= 80:
        return 50 + (x - 40) * 50 / 40
    elif x > 80 and x <= 380:
        return 100 + (x - 80) * 100 / 300
    elif x > 380 and x <= 800:
        return 200 + (x - 380) * 100 / 420
    elif x > 800 and x <= 1600:
        return 300 + (x - 800) * 100 / 800
    elif x > 1600:
        return 400 + (x - 1600) * 100 / 800
    else:
        return 0

air_quality["SO2_SubIndex"] = air_quality["SO2"].astype(int).apply(lambda
```

```
58
59 # NOx Sub-Index calculation
60 def get_NOx_subindex(x):
61     if x <= 40:
62         return x * 50 / 40
63     elif x > 40 and x <= 80:
64         return 50 + (x - 40) * 50 / 40
65     elif x > 80 and x <= 180:
66         return 100 + (x - 80) * 100 / 100
67     elif x > 180 and x <= 280:
68         return 200 + (x - 180) * 100 / 100
69     elif x > 280 and x <= 400:
70         return 300 + (x - 280) * 100 / 120
71     elif x > 400:
72         return 400 + (x - 400) * 100 / 120
73     else:
74         return 0
75
76 air_quality["NOx_SubIndex"] = air_quality["NOx"].astype(int).apply(lambda
77
78 # NH3 Sub-Index calculation
79 def get_NH3_subindex(x):
80     if x <= 200:
81         return x * 50 / 200
82     elif x > 200 and x <= 400:
83         return 50 + (x - 200) * 50 / 200
84     elif x > 400 and x <= 800:
85         return 100 + (x - 400) * 100 / 400
86     elif x > 800 and x <= 1200:
87         return 200 + (x - 800) * 100 / 400
88     elif x > 1200 and x <= 1800:
89         return 300 + (x - 1200) * 100 / 600
90     elif x > 1800:
91         return 400 + (x - 1800) * 100 / 600
92     else:
93         return 0
94
95 air_quality["NH3_SubIndex"] = air_quality["NH3"].astype(int).apply(lambda
96
97 # CO Sub-Index calculation
98 def get_CO_subindex(x):
99     if x <= 1:
100         return x * 50 / 1
101     elif x > 1 and x <= 2:
102         return 50 + (x - 1) * 50 / 1
103     elif x > 2 and x <= 10:
104         return 100 + (x - 2) * 100 / 8
105     elif x > 10 and x <= 17:
106         return 200 + (x - 10) * 100 / 7
107     elif x > 17 and x <= 34:
108         return 300 + (x - 17) * 100 / 17
109     elif x > 34:
110         return 400 + (x - 34) * 100 / 17
111     else:
112         return 0
113
114 air_quality["CO_SubIndex"] = air_quality["CO"].astype(int).apply(lambda x:
```

```
115
116 # O3 Sub-Index calculation
117 def get_O3_subindex(x):
118     if x <= 50:
119         return x * 50 / 50
120     elif x > 50 and x <= 100:
121         return 50 + (x - 50) * 50 / 50
122     elif x > 100 and x <= 168:
123         return 100 + (x - 100) * 100 / 68
124     elif x > 168 and x <= 208:
125         return 200 + (x - 168) * 100 / 40
126     elif x > 208 and x <= 748:
127         return 300 + (x - 208) * 100 / 539
128     elif x > 748:
129         return 400 + (x - 400) * 100 / 539
130     else:
131         return 0
132
133 air_quality["O3_SubIndex"] = air_quality["O3"].astype(int).apply(lambda x:
134
135
```

In [15]:

```
1 air_quality["AQI"] = air_quality["AQI"].fillna(round(air_quality[["PM2.5_S
```

In [16]:
```
1 air_quality
```

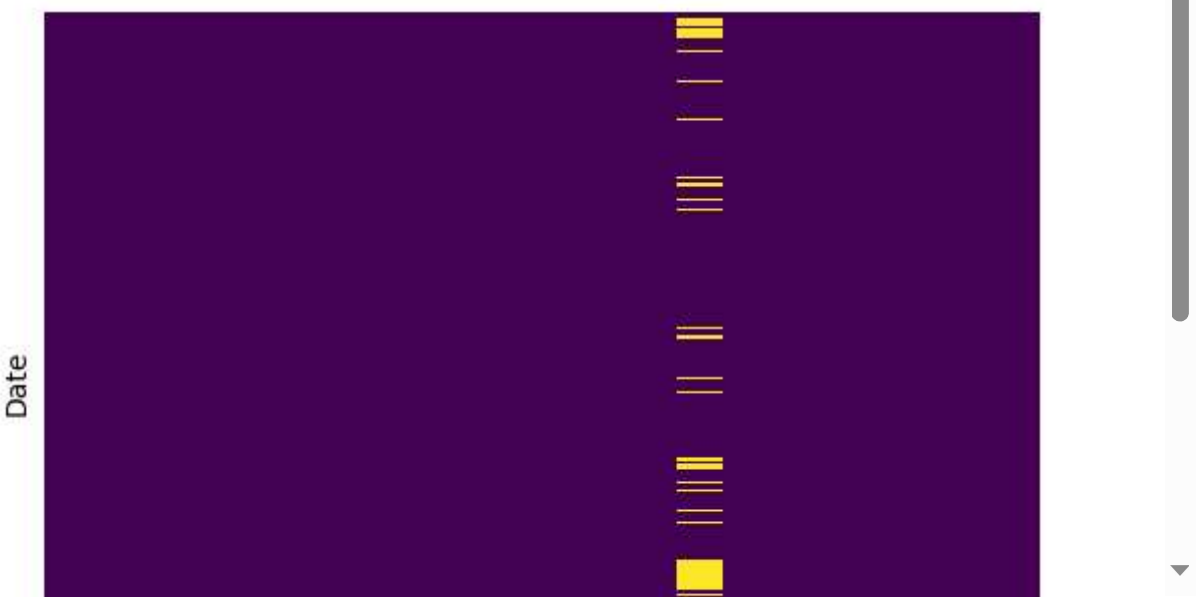
Out[16]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	23.024137	0.92	27.64	133.36
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	23.024137	0.97	24.55	34.06
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	23.024137	17.40	29.07	30.70
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	23.024137	1.70	18.59	36.08
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	23.024137	22.10	39.33	39.31
...	...	...	...	...	...	...	...	...	...	...
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05

29531 rows × 22 columns

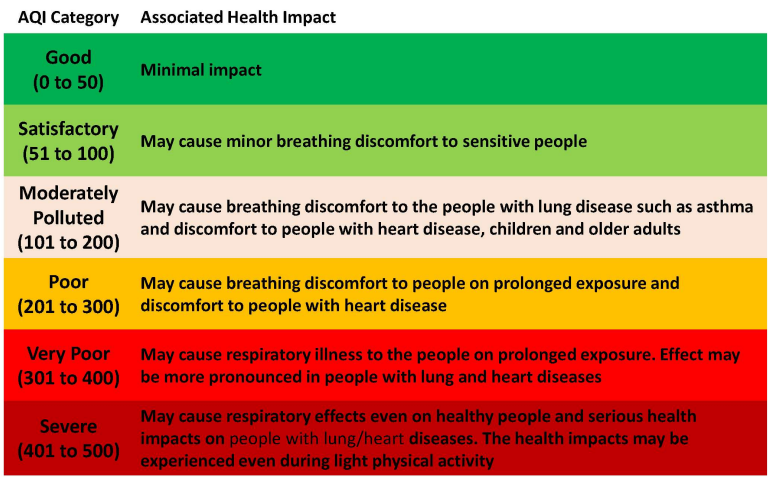
In [17]:
```
1 sns.heatmap(air_quality.isnull(),yticklabels=False,cbar=False,cmap='viridi
```

Out[17]:
```
<Axes: ylabel='Date'>
```



In [18]:
```
from IPython import display
display.Image("aqi_image.png",width = 400, height = 200)
```

Out[18]:



In [19]:
```
## AQI bucketing
def get_AQI_bucket(x):
    if x <= 50:
        return "Good"
    elif x > 50 and x <= 100:
        return "Satisfactory"
    elif x > 100 and x <= 200:
        return "Moderate"
    elif x > 200 and x <= 300:
        return "Poor"
    elif x > 300 and x <= 400:
        return "Very Poor"
    elif x > 400:
        return "Severe"
    else:
        return '0'

air_quality["AQI_Bucket"] = air_quality["AQI_Bucket"].fillna(air_quality["

```

In [20]:
```
1 air_quality
```

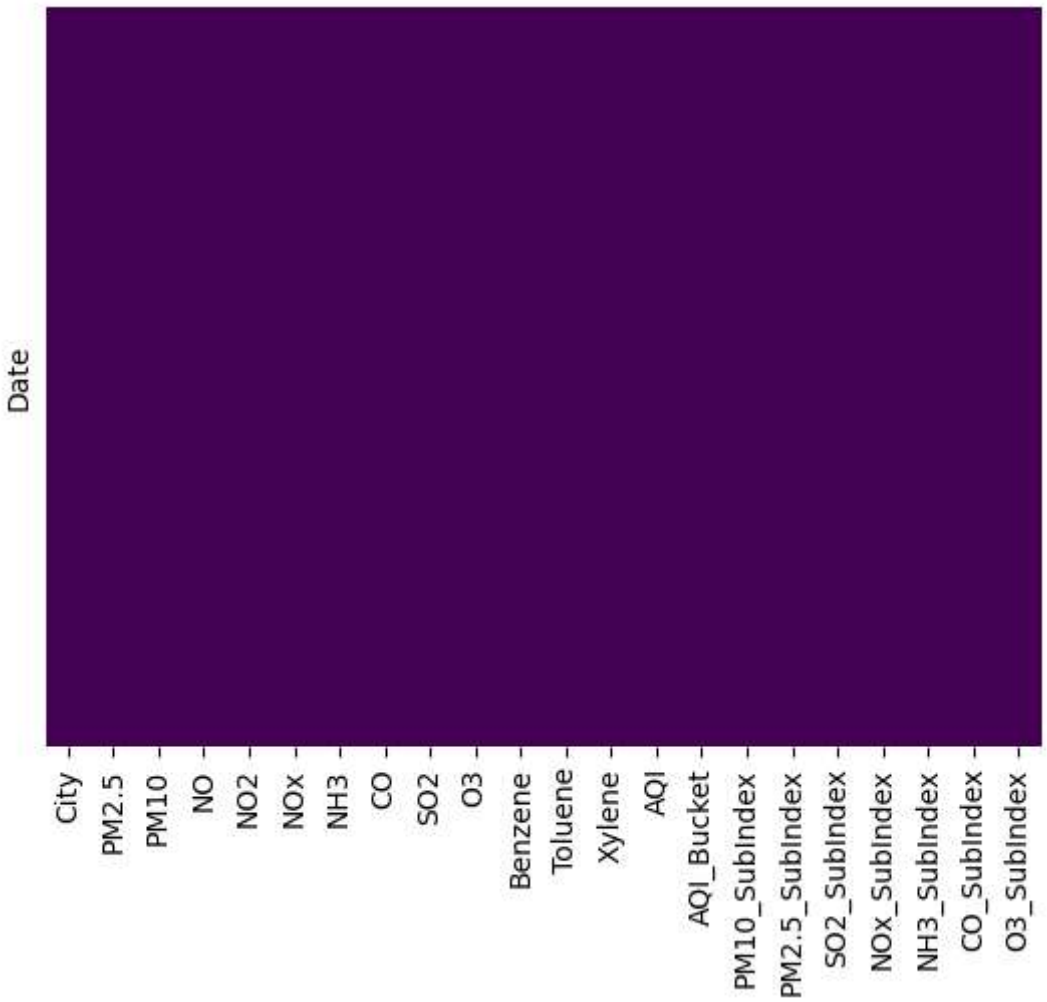
Out[20]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	23.024137	0.92	27.64	133.36
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	23.024137	0.97	24.55	34.06
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	23.024137	17.40	29.07	30.70
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	23.024137	1.70	18.59	36.08
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	23.024137	22.10	39.33	39.31
...	...	...	...	...	...	...	...	...	...	...
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05

29531 rows × 22 columns

In [21]:
```
1 sns.heatmap(air_quality.isnull(),yticklabels=False,cbar=False,cmap='viridi
```

Out[21]: <Axes: ylabel='Date'>



In [22]:
```
1 air_quality.columns
```

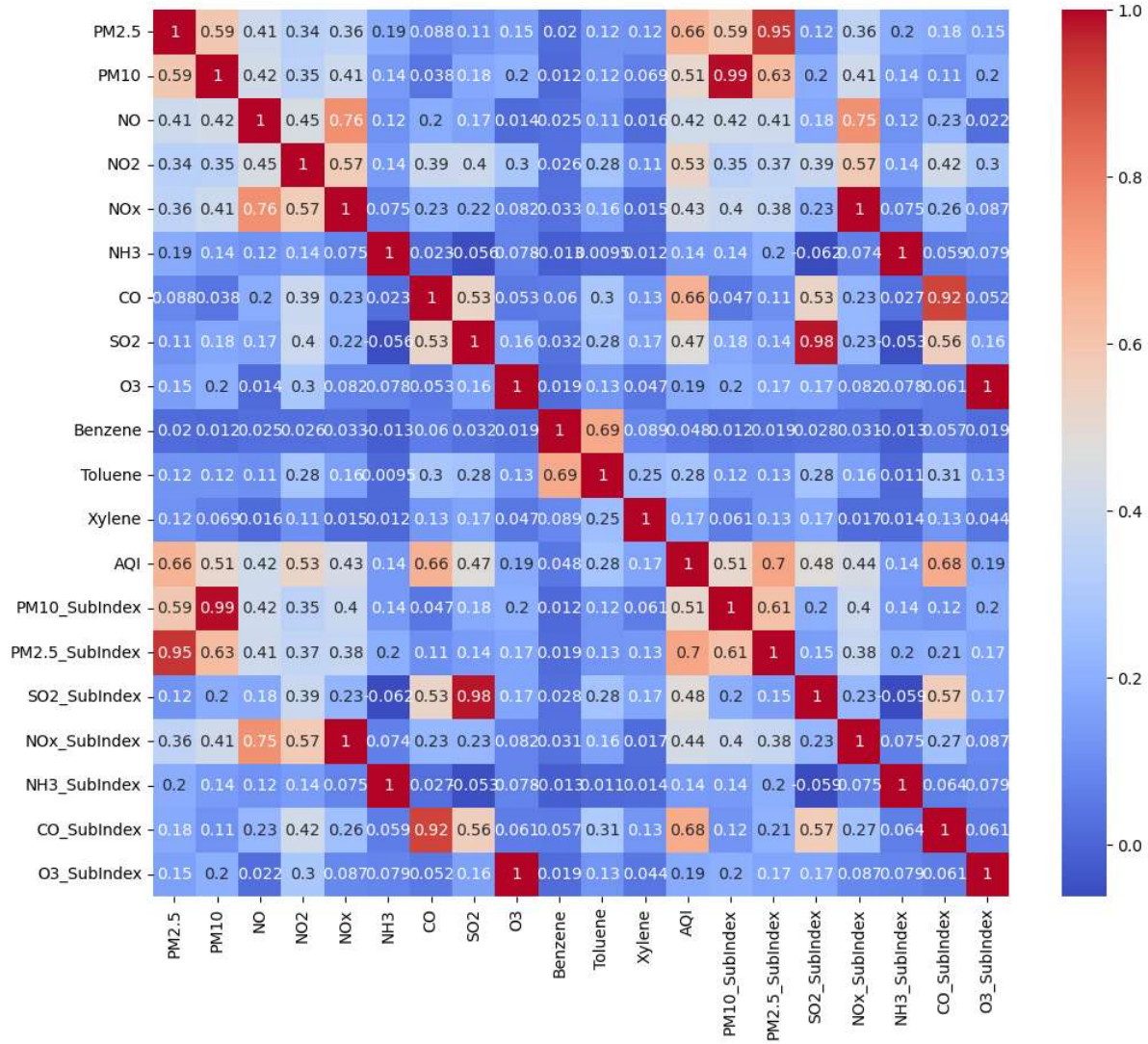
Out[22]:
```
Index(['City', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3',
       'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket', 'PM10_SubIndex',
       'PM2.5_SubIndex', 'SO2_SubIndex', 'NOx_SubIndex', 'NH3_SubIndex',
       'CO_SubIndex', 'O3_SubIndex'],
      dtype='object')
```

In [23]:
```
1 air_quality_city_day = air_quality.copy()
2 air_quality_city_day.columns
```

Out[23]:
```
Index(['City', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3',
       'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket', 'PM10_SubIndex',
       'PM2.5_SubIndex', 'SO2_SubIndex', 'NOx_SubIndex', 'NH3_SubIndex',
       'CO_SubIndex', 'O3_SubIndex'],
      dtype='object')
```

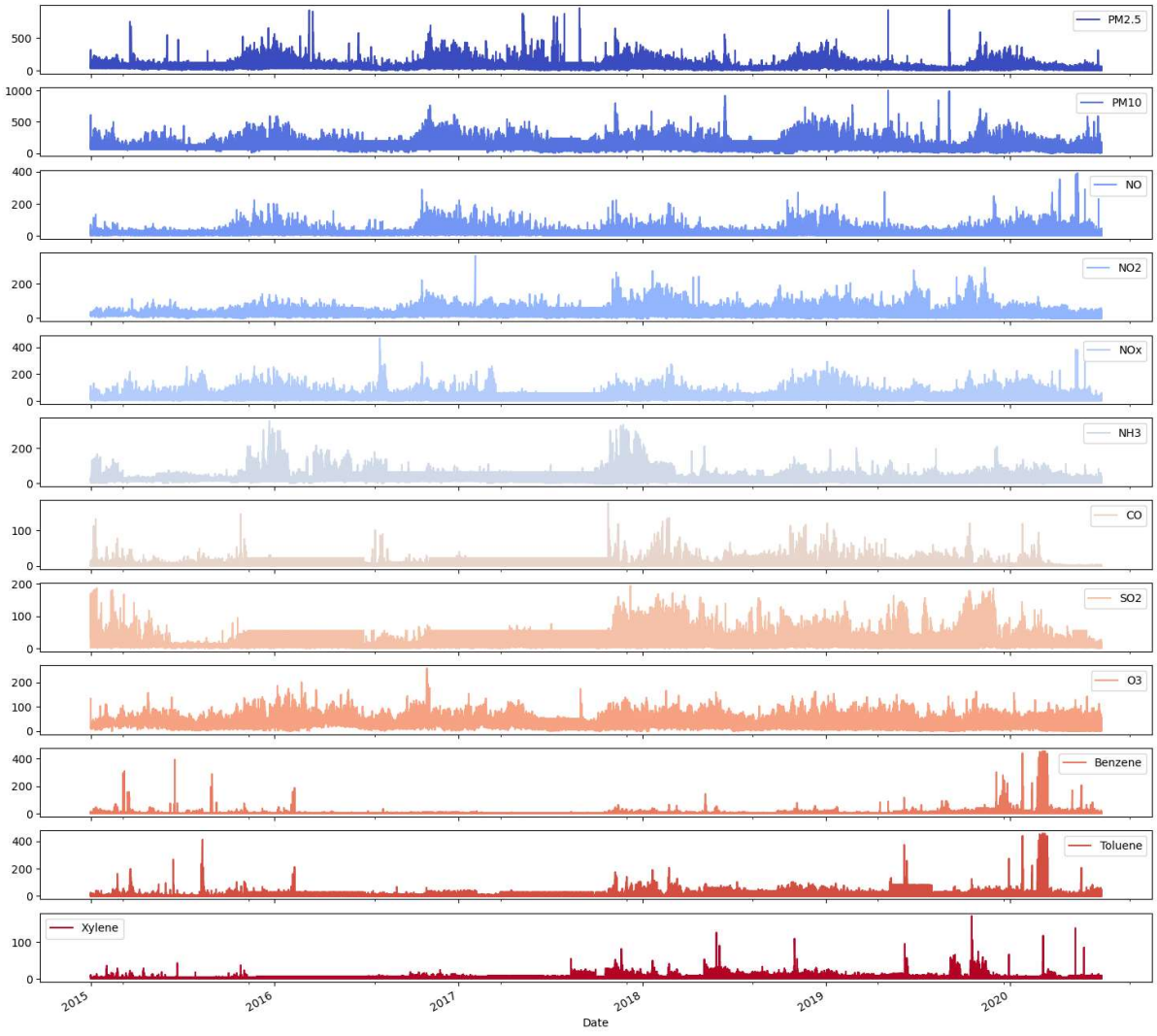
In [24]:
```
plt.figure(figsize=(12,10))
sns.heatmap(air_quality.corr(),cmap='coolwarm',annot=True);
```

```
C:\Users\vanaja\AppData\Local\Temp\ipykernel_8352\3041913524.py:2: FutureWarn
ing: The default value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid columns or specif
y the value of numeric_only to silence this warning.
  sns.heatmap(air_quality.corr(),cmap='coolwarm',annot=True);
```

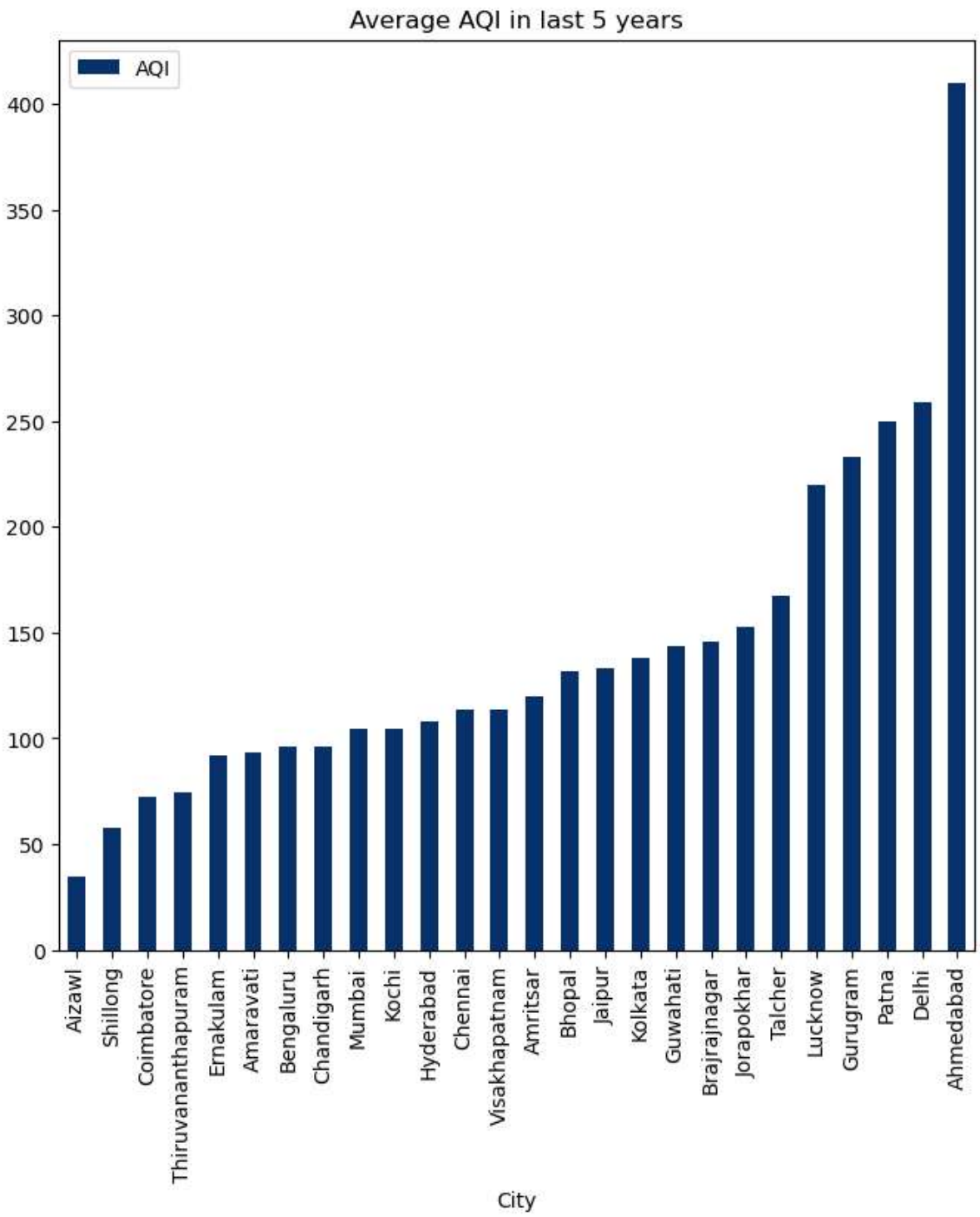


In [25]:
```
pollutants = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2','O3'
air_quality_city_day = air_quality_city_day[pollutants]

print('Distribution of different pollutants in last 5 years')
air_quality_city_day.plot(kind='line',figsize=(18,18),cmap='coolwarm',subp


```

```
Distribution of different pollutants in last 5 years
```



In [26]:
```
air_quality[['City','AQI']].groupby('City').mean().sort_values('AQI').plot
plt.title('Average AQI in last 5 years');
```



In [27]:
```
final_air_quality= air_quality[['AQI', 'AQI_Bucket']].copy()
final_air_quality
```

Out[27]:

Date	AQI	AQI_Bucket
2015-01-01	149.0	Moderate
2015-01-02	123.0	Moderate
2015-01-03	300.0	Poor
2015-01-04	123.0	Moderate
2015-01-05	329.0	Very Poor
...	...	...
2020-06-27	41.0	Good
2020-06-28	70.0	Satisfactory
2020-06-29	68.0	Satisfactory
2020-06-30	54.0	Satisfactory
2020-07-01	50.0	Good

29531 rows × 2 columns

In [28]:
```
final_air_quality['AQI_Bucket'].unique()
```

Out[28]:
```
array(['Moderate', 'Poor', 'Very Poor', 'Severe', 'Satisfactory', 'Good'],
      dtype=object)
```

In [29]:
```
final_air_quality['AQI_Bucket']=final_air_quality['AQI_Bucket'].map({'Good
final_air_quality.head()
```

Out[29]:

Date	AQI	AQI_Bucket
2015-01-01	149.0	2
2015-01-02	123.0	2
2015-01-03	300.0	3
2015-01-04	123.0	2
2015-01-05	329.0	4

In [30]:
```
X = final_air_quality[['AQI']]
y = final_air_quality[['AQI_Bucket']]
```

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import  train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0

clf = RandomForestClassifier(random_state = 0).fit(X_train, y_train)

y_pred = clf.predict(X_test)

```

```
C:\Users\vanaja\AppData\Local\Temp\ipykernel_8352\68117190.py:6: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples,), for example using ravel().
  clf = RandomForestClassifier(random_state = 0).fit(X_train, y_train)
```

In [32]:
```
print("Enter the value of AQI:")
AQI = float(input("AQI : "))
output = clf.predict([[AQI]])
output
#0-->Good
#1-->Satisfactory
#2-->moderate
#3-->poor
#4-->Very poor
#5-->Severe
```

```
Enter the value of AQI:
AQI : 567

C:\Users\vanaja\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning:
X does not have valid feature names, but RandomForestClassifier was fitted wi
th feature names
  warnings.warn(
```

Out[32]:
```
array([5])
```

In [34]:
```
if AQI <= 50:
    print("Good")
    print("Associated Health Impact:\nHealth impact is minimum")
elif AQI > 50 and AQI <= 100:
    print("Satisfactory")
    print("Associated Health Impact:\nMay cause minor breathing discomfort
elif AQI > 100 and AQI <= 200:
    print("Moderately polluted")
    print("Associated Health Impact:\nMay cause breathing discomfort to th
elif AQI > 200 and AQI <= 300:
    print("Poor")
    print("Associated Health Impact:\nMay cause breathing discomfort to pe
elif AQI > 300 and AQI <= 400:
    print("Very Poor")
    print("Associated Health Impact:\nMay cause respiratory illness to the
elif AQI > 400:
    print("Severe")
    print("Associated Health Impact:\nMay cause respiratory effects even o
else:
    print("Please enter a valid AQI Range!!")

```

```
Severe
Associated Health Impact:
May cause respiratory effects even on healthy people and serious health Impac
ts on people with lang/heart diseases. The health impacts may be experienced
even during Night physical activity
```

In [35]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       388
           1       1.00      1.00      1.00      2397
           2       1.00      1.00      1.00      2608
           3       1.00      1.00      1.00       805
           4       1.00      1.00      1.00       861
           5       1.00      1.00      1.00       324

    accuracy                           1.00      7383
   macro avg       1.00      1.00      1.00      7383
weighted avg       1.00      1.00      1.00      7383

[[ 388    0    0    0    0    0]
 [   0 2397    0    0    0    0]
 [   0    0 2608    0    0    0]
 [   0    0    0  805    0    0]
 [   0    0    0    0  861    0]
 [   0    0    0    0    0  324]]
```

In [37]:
```
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
DT = DecisionTreeClassifier(random_state = 0).fit(X_train, y_train)
predictions = clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 1.0
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
LR = LogisticRegression(random_state = 0).fit(X_train, y_train)
predictions = clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")
```

```
C:\Users\vanaja\anaconda3\lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
C:\Users\vanaja\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(

Accuracy: 1.0
```

In [ ]:
```

```