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

In [2]:
```
!pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\abhir\anaconda3\lib\site-packages
(0.0.post5)
```

In [3]:
```
df=pd.read_csv('city_day.csv',parse_dates = ["Date"])
df
```

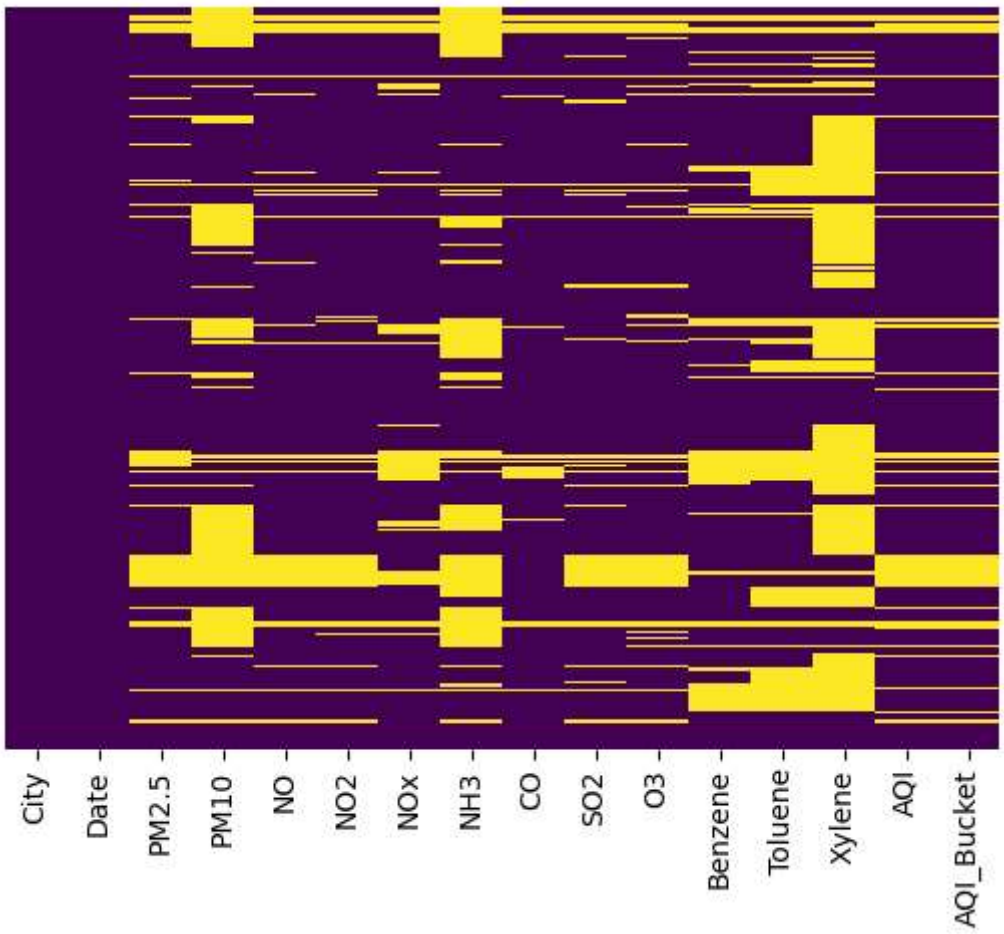
Out[3]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene
0	Ahmedabad	2015-01-01	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.00
1	Ahmedabad	2015-01-02	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.68
2	Ahmedabad	2015-01-03	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.80
3	Ahmedabad	2015-01-04	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.43
4	Ahmedabad	2015-01-05	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.01
...	...	...	...	...	...	...	...	...	...	...	...	...
29526	Visakhapatnam	2020-06-27	15.02	50.94	7.68	25.06	19.54	12.47	0.47	8.55	23.30	2.24
29527	Visakhapatnam	2020-06-28	24.38	74.09	3.42	26.06	16.53	11.99	0.52	12.72	30.14	0.74
29528	Visakhapatnam	2020-06-29	22.91	65.73	3.45	29.53	18.33	10.71	0.48	8.42	30.96	0.01
29529	Visakhapatnam	2020-06-30	16.64	49.97	4.05	29.26	18.80	10.03	0.52	9.84	28.30	0.00
29530	Visakhapatnam	2020-07-01	15.00	66.00	0.40	26.85	14.05	5.20	0.59	2.10	17.05	NaN

29531 rows × 16 columns

In [4]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[4]:
```
<Axes: >
```



In [5]:
```
print(df.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 [6]:
```
(df.isnull().sum()/df.shape[0]*100).sort_values(ascending=False)
```

Out[6]:
```
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 [7]:
```
df.describe()
```

Out[7]:

	PM2.5	PM10	NO	NO2	NOx	NH3	
count	24933.000000	18391.000000	25949.000000	25946.000000	25346.000000	19203.000000	27472.000
mean	67.450578	118.127103	17.574730	28.560659	32.309123	23.483476	2.248
std	64.661449	90.605110	22.785846	24.474746	31.646011	25.684275	6.962
min	0.040000	0.010000	0.020000	0.010000	0.000000	0.010000	0.000
25%	28.820000	56.255000	5.630000	11.750000	12.820000	8.580000	0.510
50%	48.570000	95.680000	9.890000	21.690000	23.520000	15.850000	0.890
75%	80.590000	149.745000	19.950000	37.620000	40.127500	30.020000	1.450
max	949.990000	1000.000000	390.680000	362.210000	467.630000	352.890000	175.810

In [8]:
```
#converting dtype of date column to datetime
df['Date']=df['Date'].apply(pd.to_datetime)
#setting date column as index
df.set_index('Date',inplace=True)
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['City', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3',
       'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
      dtype='object')
```

In [10]:
```
df.iloc[:, 1:13] = df.groupby("City").transform(lambda x: x.fillna(x.mean()))
```

```
C:\Users\abhir\AppData\Local\Temp\ipykernel_16136\3116926243.py:1: FutureWarning: Dro
pping invalid columns in DataFrameGroupBy.transform is deprecated. In a future versio
n, a TypeError will be raised. Before calling .transform, select only columns which s
hould be valid for the function.
  df.iloc[:, 1:13] = df.groupby("City").transform(lambda x: x.fillna(x.mean()))
```

In [11]:
```
df
```

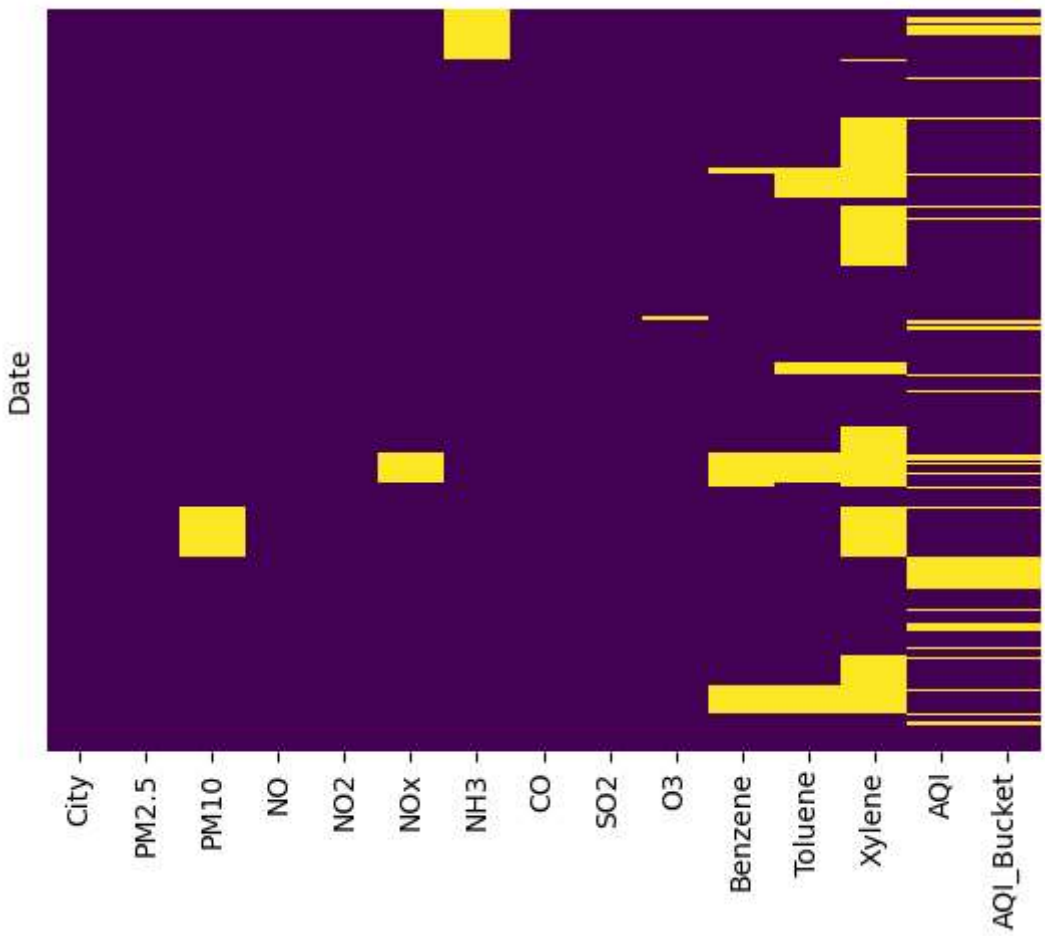
Out[11]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.000000
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.680000
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.800000
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.430000
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.010000
...	...	...	...	...	...	...	...	...	...	...	...
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.47	0.47	8.55	23.30	2.240000
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.99	0.52	12.72	30.14	0.740000
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.71	0.48	8.42	30.96	0.010000
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.03	0.52	9.84	28.30	0.000000
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.20	0.59	2.10	17.05	3.891348

29531 rows × 15 columns

In [12]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[12]:
```
<Axes: ylabel='Date'>
```



In [13]:
```
df.iloc[:, 0:13]=df.fillna(df.mean())
df
```

```
C:\Users\abhir\AppData\Local\Temp\ipykernel_16136\2813135996.py:1: FutureWarning: The
default value of numeric_only in DataFrame.mean is deprecated. In a future version, i
t will default to False. In addition, specifying 'numeric_only=None' is deprecated. S
elect only valid columns or specify the value of numeric_only to silence this warnin
g.
  df.iloc[:, 0:13]=df.fillna(df.mean())
```

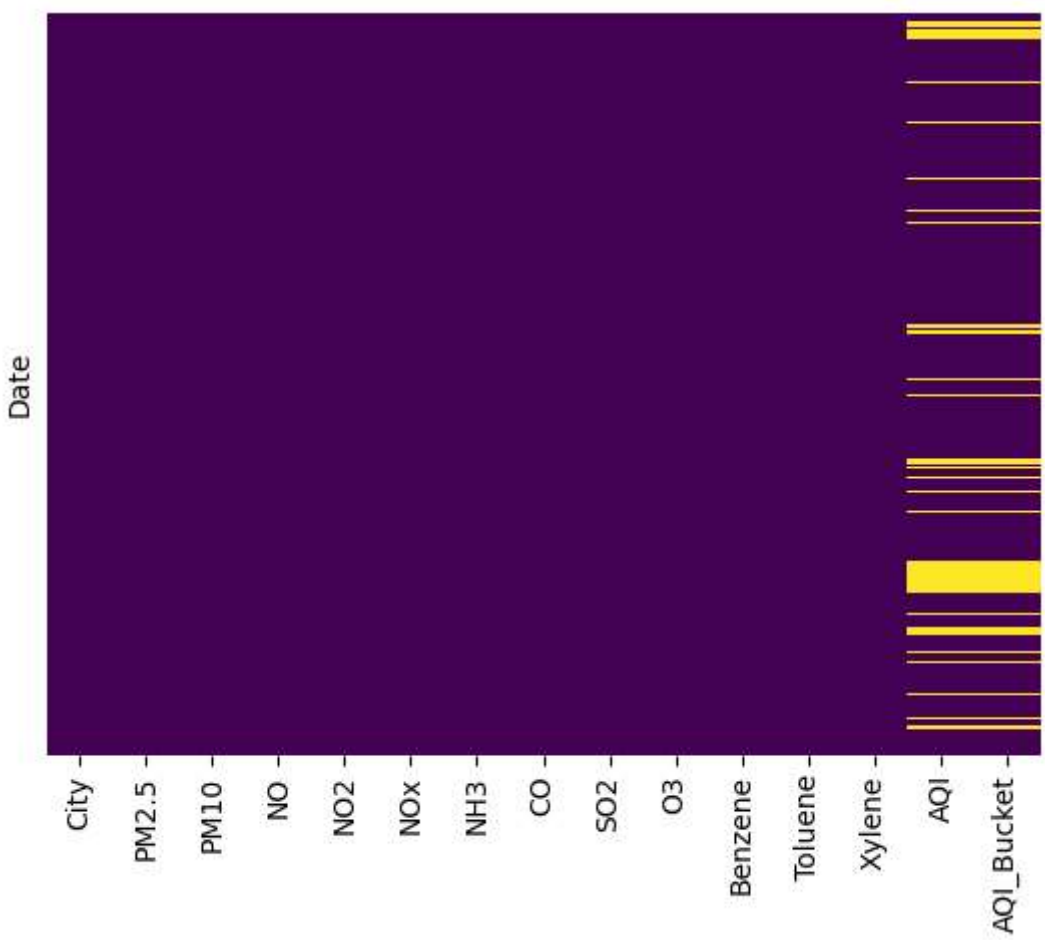
Out[13]:

Date	City	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Ben
2015-01-01	Ahmedabad	67.854497	114.584029	0.92	18.22	17.15	23.024137	0.92	27.64	133.36	0.00
2015-01-02	Ahmedabad	67.854497	114.584029	0.97	15.69	16.46	23.024137	0.97	24.55	34.06	3.68
2015-01-03	Ahmedabad	67.854497	114.584029	17.40	19.30	29.70	23.024137	17.40	29.07	30.70	6.80
2015-01-04	Ahmedabad	67.854497	114.584029	1.70	18.48	17.97	23.024137	1.70	18.59	36.08	4.43
2015-01-05	Ahmedabad	67.854497	114.584029	22.10	21.42	37.76	23.024137	22.10	39.33	39.31	7.01
...	...	...	...	...	...	...	...	...	...	...	
2020-06-27	Visakhapatnam	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30	2.24
2020-06-28	Visakhapatnam	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14	0.74
2020-06-29	Visakhapatnam	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96	0.01
2020-06-30	Visakhapatnam	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30	0.00
2020-07-01	Visakhapatnam	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05	3.89

29531 rows × 15 columns

In [14]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[14]:
```
<Axes: ylabel='Date'>
```



In [15]:
```python
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

df["PM10_SubIndex"] = df["PM10"].astype(int).apply(lambda x: get_PM10_subindex(x))

# 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

df["PM2.5_SubIndex"] = df["PM2.5"].astype(int).apply(lambda x: get_PM25_subindex(x))


# 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

df["SO2_SubIndex"] = df["SO2"].astype(int).apply(lambda x: get_SO2_subindex(x))

# NOx Sub-Index calculation
def get_NOx_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 <= 180:
        return 100 + (x - 80) * 100 / 100
    elif x > 180 and x <= 280:
        return 200 + (x - 180) * 100 / 100
    elif x > 280 and x <= 400:
        return 300 + (x - 280) * 100 / 120
    elif x > 400:
        return 400 + (x - 400) * 100 / 120
    else:
        return 0

df["NOx_SubIndex"] = df["NOx"].astype(int).apply(lambda x: get_NOx_subindex(x))

# NH3 Sub-Index calculation
def get_NH3_subindex(x):
    if x <= 200:
        return x * 50 / 200
    elif x > 200 and x <= 400:
        return 50 + (x - 200) * 50 / 200
    elif x > 400 and x <= 800:
        return 100 + (x - 400) * 100 / 400
    elif x > 800 and x <= 1200:
        return 200 + (x - 800) * 100 / 400
    elif x > 1200 and x <= 1800:
        return 300 + (x - 1200) * 100 / 600
    elif x > 1800:
```

```
        return 400 + (x - 1800) * 100 / 600
    else:
        return 0
df["NH3_SubIndex"] = df["NH3"].astype(int).apply(lambda x: get_NH3_subindex(x))

# CO Sub-Index calculation
def get_CO_subindex(x):
    if x <= 1:
        return x * 50 / 1
    elif x > 1 and x <= 2:
        return 50 + (x - 1) * 50 / 1
    elif x > 2 and x <= 10:
        return 100 + (x - 2) * 100 / 8
    elif x > 10 and x <= 17:
        return 200 + (x - 10) * 100 / 7
    elif x > 17 and x <= 34:
        return 300 + (x - 17) * 100 / 17
    elif x > 34:
        return 400 + (x - 34) * 100 / 17
    else:
        return 0

df["CO_SubIndex"] = df["CO"].astype(int).apply(lambda x: get_CO_subindex(x))

# O3 Sub-Index calculation
def get_O3_subindex(x):
    if x <= 50:
        return x * 50 / 50
    elif x > 50 and x <= 100:
        return 50 + (x - 50) * 50 / 50
    elif x > 100 and x <= 168:
        return 100 + (x - 100) * 100 / 68
    elif x > 168 and x <= 208:
        return 200 + (x - 168) * 100 / 40
    elif x > 208 and x <= 748:
        return 300 + (x - 208) * 100 / 539
    elif x > 748:
        return 400 + (x - 400) * 100 / 539
    else:
        return 0

df["O3_SubIndex"] = df["O3"].astype(int).apply(lambda x: get_O3_subindex(x))
```

In [16]:

```
# 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
```

```
df["PM2.5_SubIndex"] = df["PM2.5"].astype(int).apply(lambda x: get_PM25_subindex(x))
```

In [17]:
```
df["AQI"] = df["AQI"].fillna(round(df[["PM2.5_SubIndex", "PM10_SubIndex", "SO2_SubInde
```

In [18]:
```
df
```

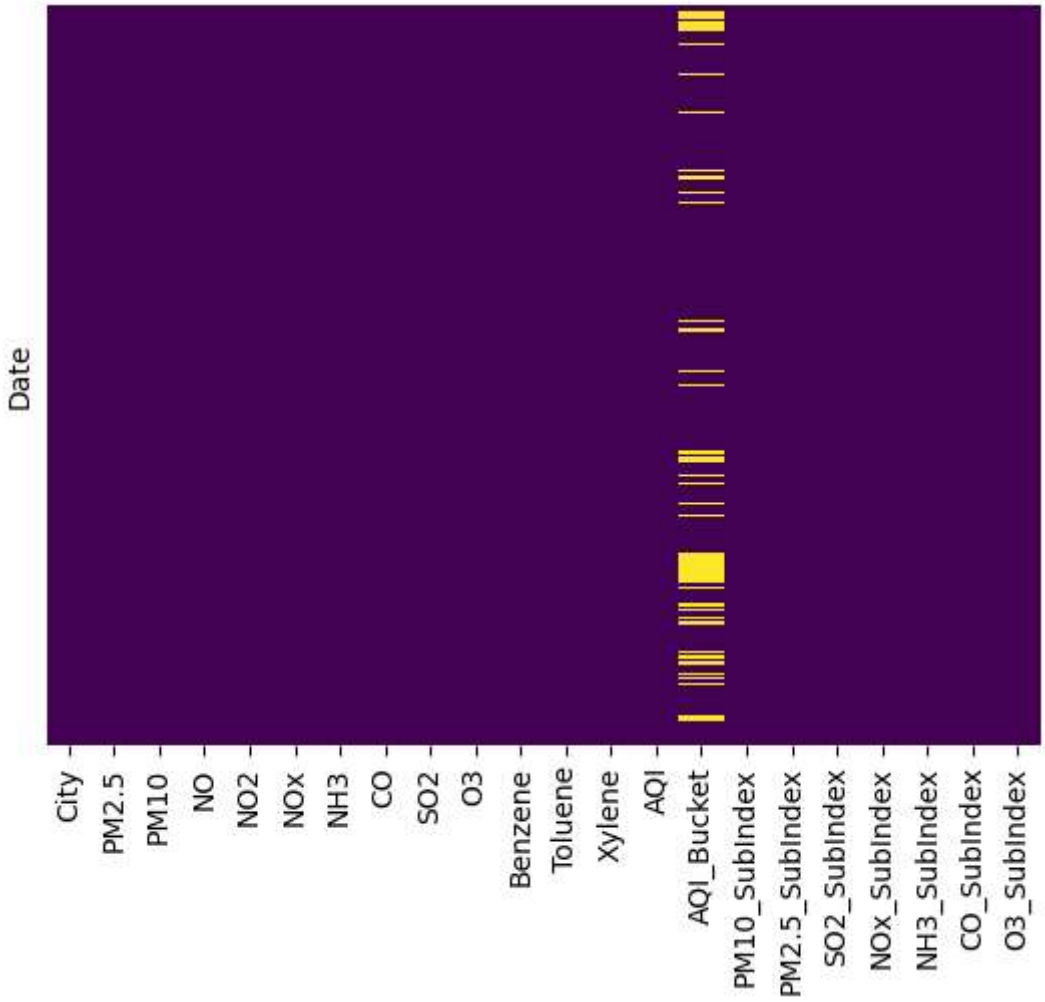
Out[18]:

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 [19]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[19]:
```
<Axes: ylabel='Date'>
```



In [20]:
```
from IPython import display
display.Image("download.png",width = 400, height = 200)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\IPython\core\display.py:1045, in Image._data_and_m
etadata(self, always_both)
   1044 try:
-> 1045     b64_data = b2a_base64(self.data).decode('ascii')
   1046 except TypeError as e:

TypeError: a bytes-like object is required, not 'str'

The above exception was the direct cause of the following exception:

FileNotFoundError                         Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\IPython\core\formatters.py:972, in MimeBundleForma
tter.__call__(self, obj, include, exclude)
    969     method = get_real_method(obj, self.print_method)
    971     if method is not None:
--> 972         return method(include=include, exclude=exclude)
    973     return None
    974 else:

File ~\anaconda3\lib\site-packages\IPython\core\display.py:1035, in Image._repr_mimeb
undle_(self, include, exclude)
   1033 if self.embed:
   1034     mimetype = self._mimetype
-> 1035     data, metadata = self._data_and_metadata(always_both=True)
   1036     if metadata:
   1037         metadata = {mimetype: metadata}

File ~\anaconda3\lib\site-packages\IPython\core\display.py:1047, in Image._data_and_m
etadata(self, always_both)
   1045     b64_data = b2a_base64(self.data).decode('ascii')
   1046 except TypeError as e:
-> 1047     raise FileNotFoundError(
   1048         "No such file or directory: '%s'" % (self.data)) from e
   1049 md = {}
   1050 if self.metadata:

FileNotFoundError: No such file or directory: 'download.png'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\IPython\core\display.py:1045, in Image._data_and_m
etadata(self, always_both)
   1044 try:
-> 1045     b64_data = b2a_base64(self.data).decode('ascii')
   1046 except TypeError as e:

TypeError: a bytes-like object is required, not 'str'

The above exception was the direct cause of the following exception:

FileNotFoundError                         Traceback (most recent call last)
File ~\anaconda3\lib\site-packages\IPython\core\formatters.py:342, in BaseFormatter._
__call__(self, obj)
    340     method = get_real_method(obj, self.print_method)
    341     if method is not None:
--> 342         return method()
    343     return None
    344 else:

File ~\anaconda3\lib\site-packages\IPython\core\display.py:1067, in Image._repr_png_
(self)
   1065 def _repr_png_(self):
   1066     if self.embed and self.format == self._FMT_PNG:
-> 1067         return self._data_and_metadata()

File ~\anaconda3\lib\site-packages\IPython\core\display.py:1047, in Image._data_and_m
etadata(self, always_both)
   1045     b64_data = b2a_base64(self.data).decode('ascii')
   1046 except TypeError as e:
-> 1047     raise FileNotFoundError(
   1048         "No such file or directory: '%s'" % (self.data)) from e
   1049 md = {}
   1050 if self.metadata:

FileNotFoundError: No such file or directory: 'download.png'
```

Out[20]: <IPython.core.display.Image object>

In [21]:
```
## 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'

df["AQI_Bucket"] = df["AQI_Bucket"].fillna(df["AQI"].apply(lambda x: get_AQI_bucket(x)
```

In [22]:
```
df
```

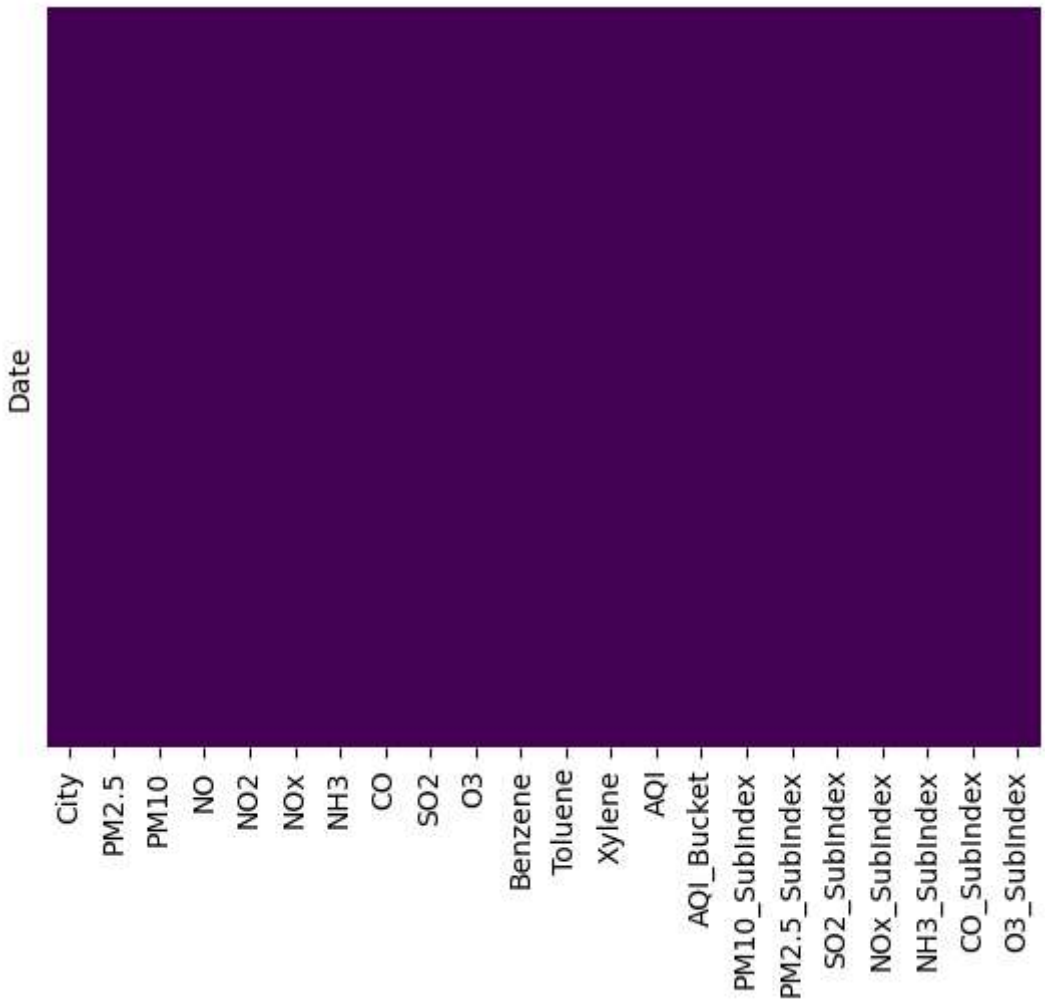
Out[22]:

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 [23]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[23]:
```
<Axes: ylabel='Date'>
```



In [24]:
```
df.columns
```

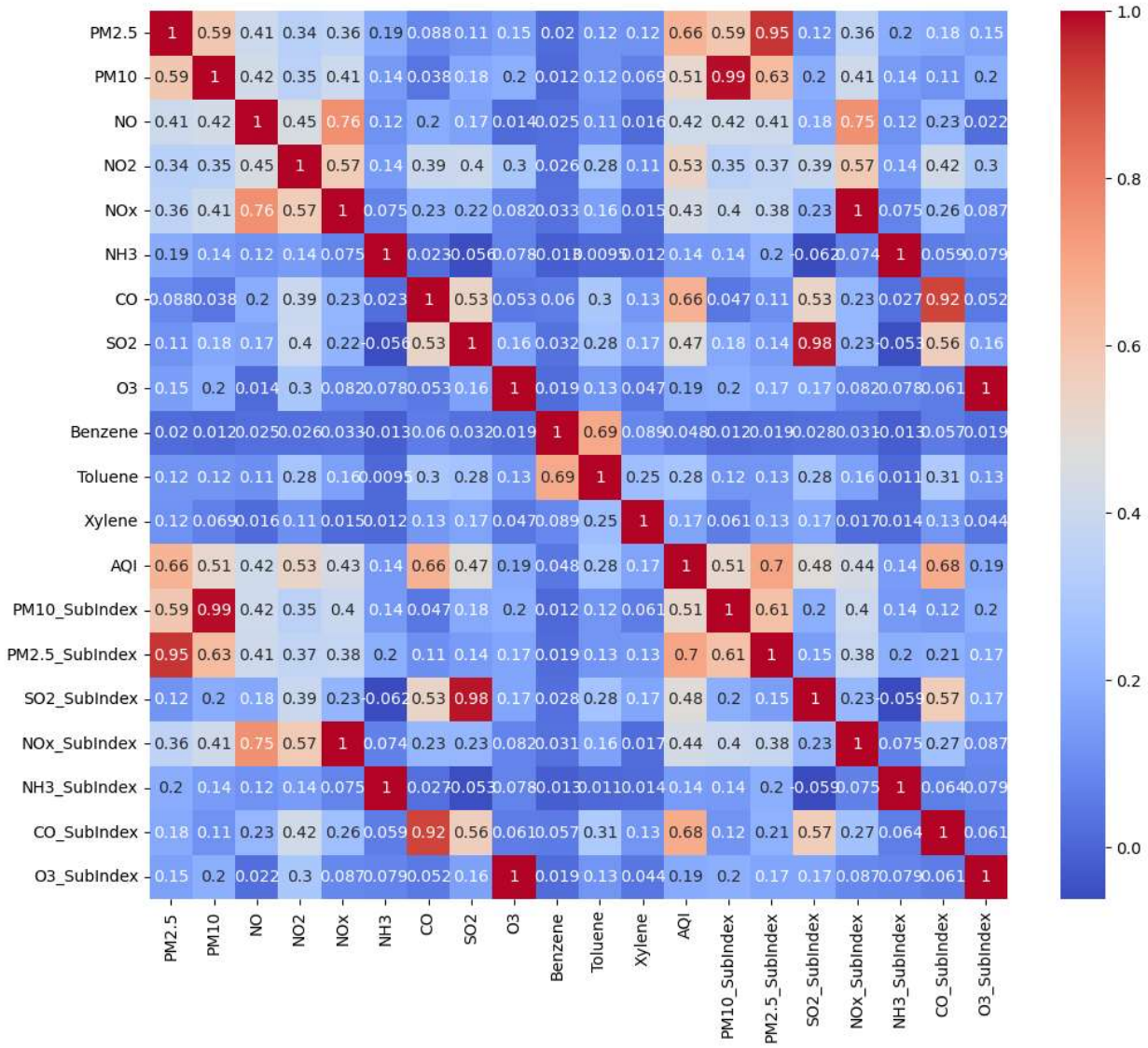
Out[24]:
```
Index(['City', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3',
       'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket', 'PM10_SubIndex',
       'PM2.5_SubIndex', 'SO2_SubIndex', 'NOx_SubIndex', 'NH3_SubIndex',
       'CO_SubIndex', 'O3_SubIndex'],
      dtype='object')
```

In [25]:
```
df_city_day = df.copy()
df_city_day.columns
```

Out[25]:
```
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 [26]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True);
```

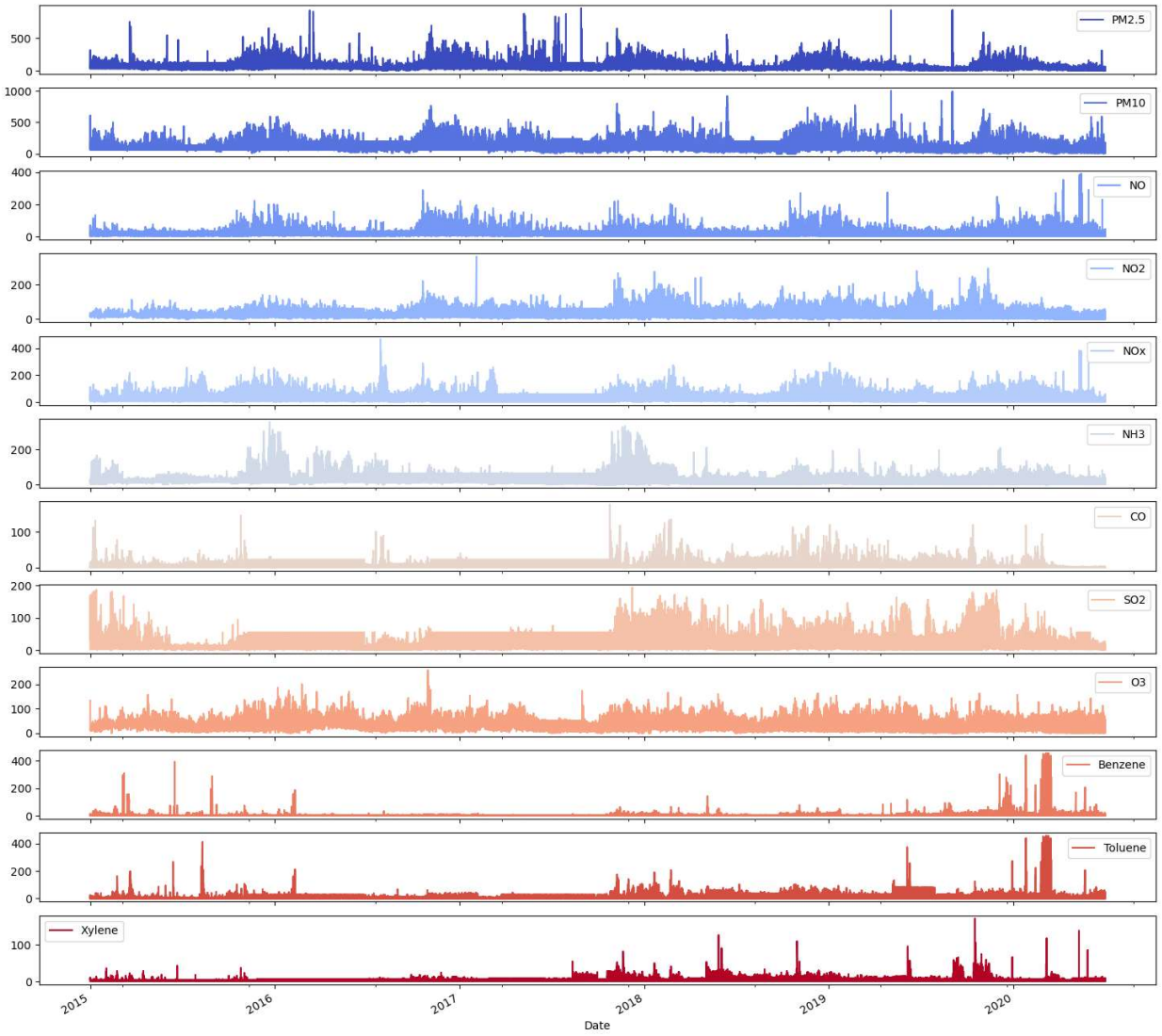
```
C:\Users\abhir\AppData\Local\Temp\ipykernel_16136\2154488151.py:2: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
t will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  sns.heatmap(df.corr(),cmap='coolwarm',annot=True);
```



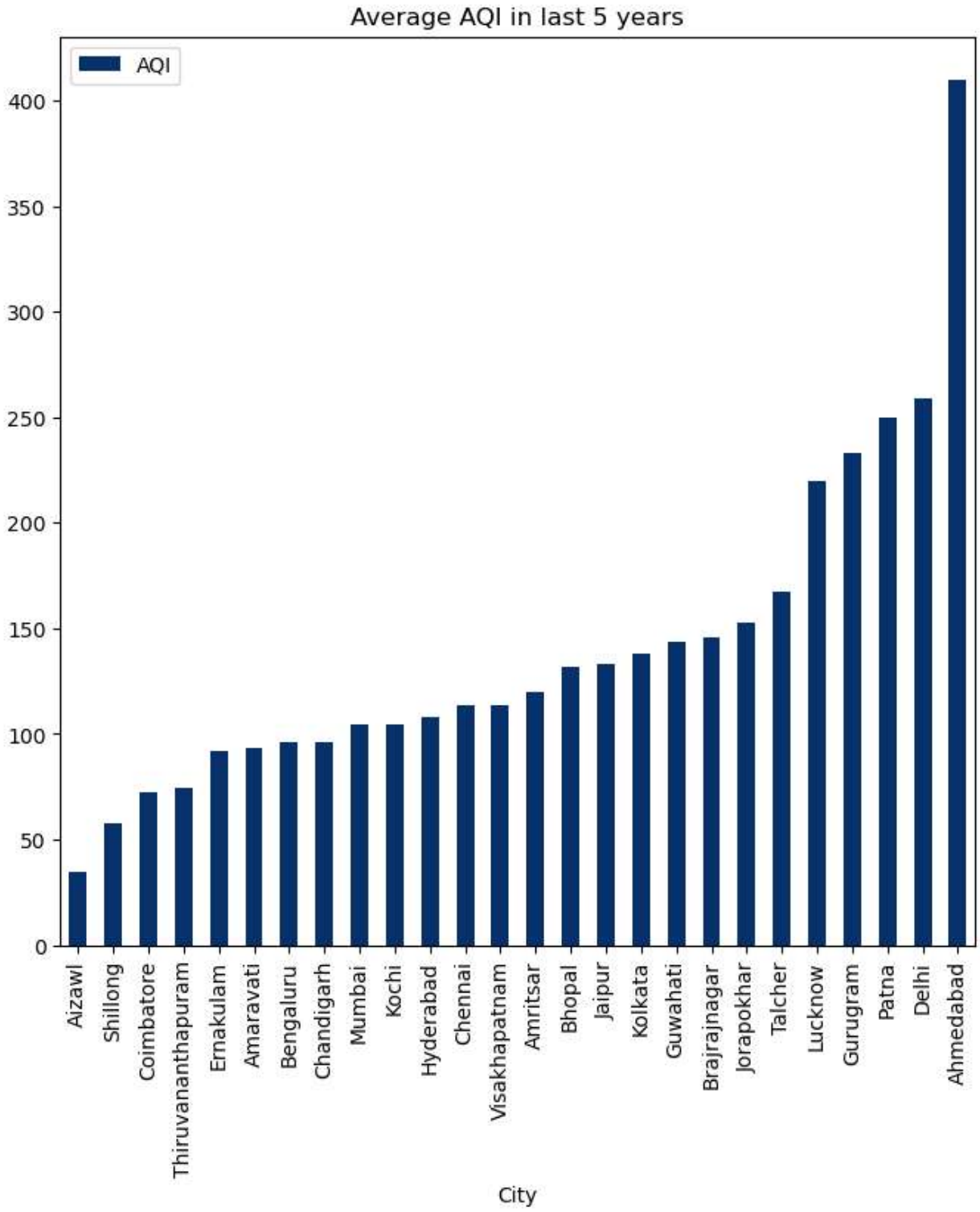
In [27]:
```
pollutants = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2','O3', 'Benzene',
df_city_day = df_city_day[pollutants]

print('Distribution of different pollutants in last 5 years')
df_city_day.plot(kind='line',figsize=(18,18),cmap='coolwarm',subplots=True,fontsize=10
```

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



In [28]:
```
df[['City','AQI']].groupby('City').mean().sort_values('AQI').plot(kind='bar',cmap='Blu
plt.title('Average AQI in last 5 years');
```



In [29]:
```
final_df= df[['AQI', 'AQI_Bucket']].copy()
final_df
```

Out[29]:

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 [30]:
```
final_df['AQI_Bucket'].unique()
```

Out[30]:
```
array(['Moderate', 'Poor', 'Very Poor', 'Severe', 'Satisfactory', 'Good'],
      dtype=object)
```

In [31]:
```
#final_df = pd.get_dummies(final_df)
final_df['AQI_Bucket'] = final_df['AQI_Bucket'].map({'Good' :0, 'Satisfactory' :1, 'Mo
final_df.head()
```

Out[31]:

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 [32]:
```
X = final_df[['AQI']]
y = final_df[['AQI_Bucket']]
```

In [33]:
```
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\abhir\AppData\Local\Temp\ipykernel_16136\2930373155.py:6: DataConversionWarn
ing: A column-vector y was passed when a 1d array was expected. Please change the sha
pe of y to (n_samples,), for example using ravel().
  clf = RandomForestClassifier(random_state = 0).fit(X_train, y_train)
```

In [34]:
```
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 : 3
C:\Users\abhir\anaconda3\lib\site-packages\sklearn\base.py:420: UserWarning: X does n
ot have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

Out[34]:
```
array([0])
```

In [35]:
```
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
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 [36]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [37]:
```
X = final_df.iloc[:,[0]].values
y = final_df.iloc[:,[1]].values
```

In [38]:
```
regressor = RandomForestRegressor(n_estimators = 100,random_state=0)
regressor.fit(X,y)
```

```
C:\Users\abhir\AppData\Local\Temp\ipykernel_16136\3226747133.py:2: DataConversionWarn
ing: A column-vector y was passed when a 1d array was expected. Please change the sha
pe of y to (n_samples,), for example using ravel().
  regressor.fit(X,y)
```

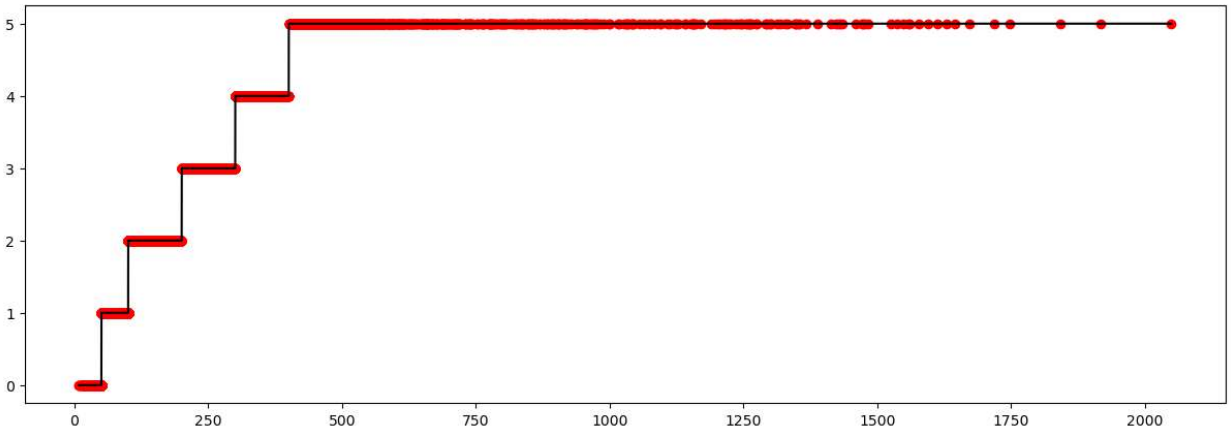
Out[38]:

```
RandomForestRegressor(random_state=0)
```

In [39]:
```
Y_pred = regressor.predict(X)
```

In [40]:
```
X_grid = np.arange(min(X),max(X),0.01)
X_grid=X_grid.reshape(len(X_grid),1)
```

In [41]:
```
plt.figure(figsize=(15,5))
plt.scatter(X,y,color="red")
plt.plot(X_grid,regressor.predict(X_grid),color="black")
plt.show()
```



In [ ]: