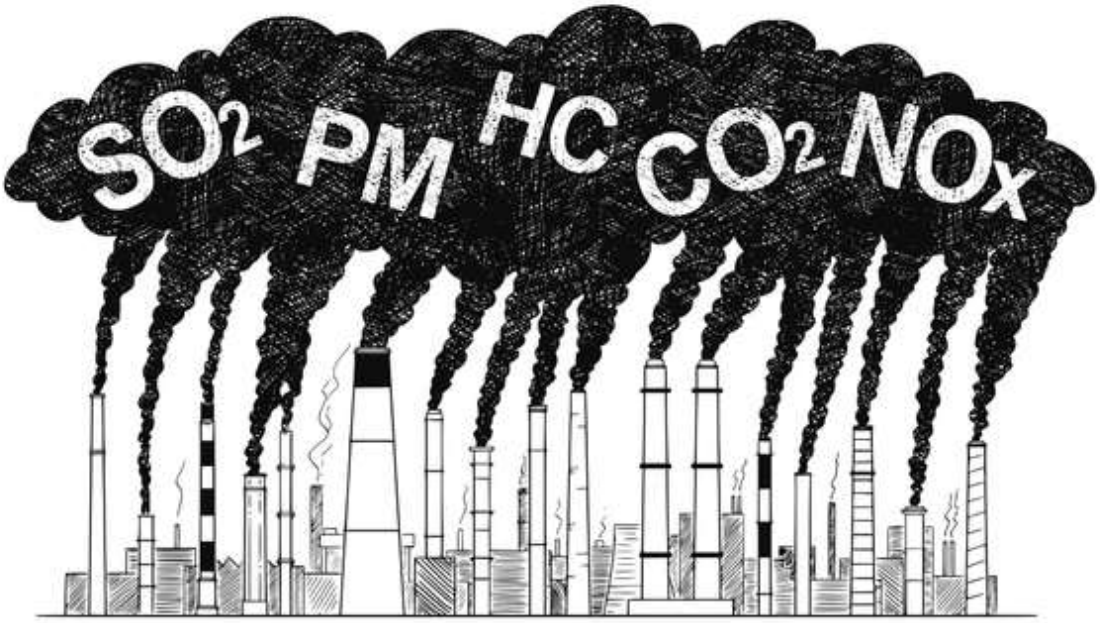


INDIA AIR QUALITY DATA ANALYSIS, VISUALISATION AND PREDICTION

Project by: Drishti & Abhishek Shukla

Aim

To do data analysis on India Air Quality data and predict tha value of Air Quality Index based on given features of concentration of sulphur dioxide,nitrogen dioxide, respirable suspended particualte matter, suspended particulate matter and classify the Air Quality as good, moderate, poor, unhealthy, healthy.

Description of the Dataset is as follows:

- stn_code : Station code. A code is given to each station that recorded the data.
- sampling_date: The date when the data was recorded.
- state: It represents the states whose air quality data is measured.
- location: It represents the city whose air quality data is measured.
- agency: Name of the agency that measured the data.
- type: The type of area where the measurement was made.
- so2: The amount of Sulphur Dioxide measured.
- no2: The amount of Nitrogen Dioxide measured.
- rspm: Respirable Suspended Particulate Matter measured.
- spm: Suspended Particulate Matter measured.
- location_monitoring_station: It indicates the location of the monitoring area.

- pm2_5: It represents the value of particulate matter measured.
- date: It represents the date of recording.

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

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
data = pd.read_csv('Pollution.csv') #import data
```

In [4]:
```
data.head(10) #print first 10 rows
```

Out[4]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	lo
0	150.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	
1	151.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	3.1	7.0	NaN	NaN	
2	152.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	
3	150.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	
4	151.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.7	7.5	NaN	NaN	
5	152.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.4	25.7	NaN	NaN	
6	150.0	April - M041990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	5.4	17.1	NaN	NaN	
7	151.0	April - M041990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.7	8.7	NaN	NaN	
8	152.0	April - M041990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	4.2	23.0	NaN	NaN	
9	151.0	May - M051990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.0	8.9	NaN	NaN	

In [5]:
```
data.tail(10) #printing last 10 rows
```

Out[5]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm
435732	SAMP	9/12/2015	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	22.0	50.0	145.0
435733	SAMP	12/12/2015	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	34.0	61.0	161.0
435734	SAMP	15-12-15	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	20.0	44.0	148.0
435735	SAMP	18-12-15	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	17.0	44.0	131.0
435736	SAMP	21-12-15	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	18.0	45.0	140.0
435737	SAMP	24-12-15	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	22.0	50.0	143.0
435738	SAMP	29-12-15	West Bengal	ULUBERIA	West Bengal State Pollution Control Board	RIRUO	20.0	46.0	171.0
435739	NaN	NaN	andaman-and-nicobar-islands	NaN	NaN	NaN	NaN	NaN	NaN
435740	NaN	NaN	Lakshadweep	NaN	NaN	NaN	NaN	NaN	NaN
435741	NaN	NaN	Tripura	NaN	NaN	NaN	NaN	NaN	NaN

In [6]:
```
data.columns #print the columns/features of the data
```

Out[6]:
```
Index(['stn_code', 'sampling_date', 'state', 'location', 'agency', 'type',
       'so2', 'no2', 'rspm', 'spm', 'location_monitoring_station', 'pm2_5',
       'date'],
      dtype='object')
```

In [7]:
```
data.describe() #basic info of the dataset
```

Out[7]:

	so2	no2	rspm	spm	pm2_5
count	401096.000000	419509.000000	395520.000000	198355.000000	9314.000000
mean	10.829414	25.809623	108.832784	220.783480	40.791467
std	11.177187	18.503086	74.872430	151.395457	30.832525
min	0.000000	0.000000	0.000000	0.000000	3.000000
25%	5.000000	14.000000	56.000000	111.000000	24.000000
50%	8.000000	22.000000	90.000000	187.000000	32.000000
75%	13.700000	32.200000	142.000000	296.000000	46.000000
max	909.000000	876.000000	6307.033333	3380.000000	504.000000

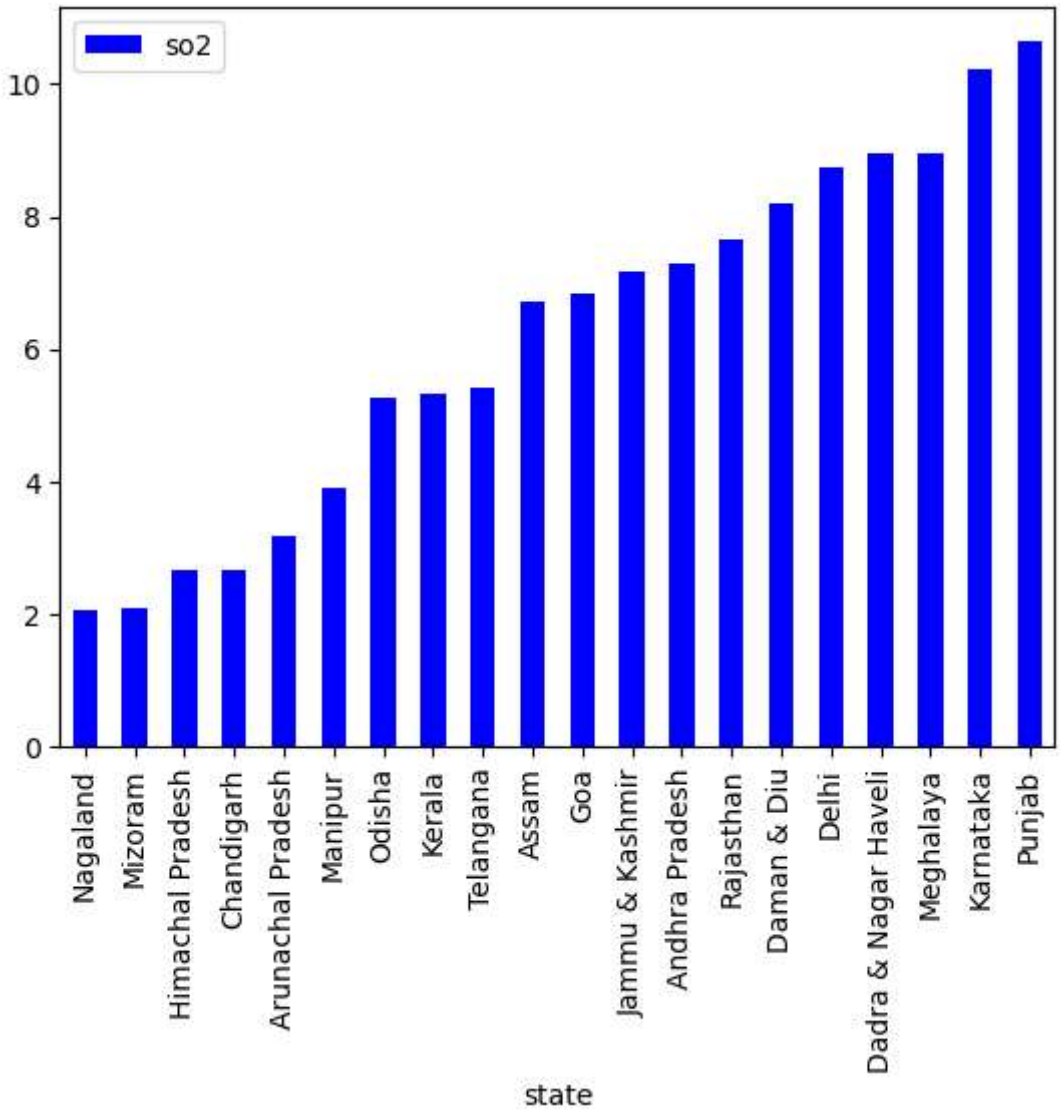
DATA SHAPE (DIMENSION)

In [8]:
```
data.shape #dimensions of the data
```

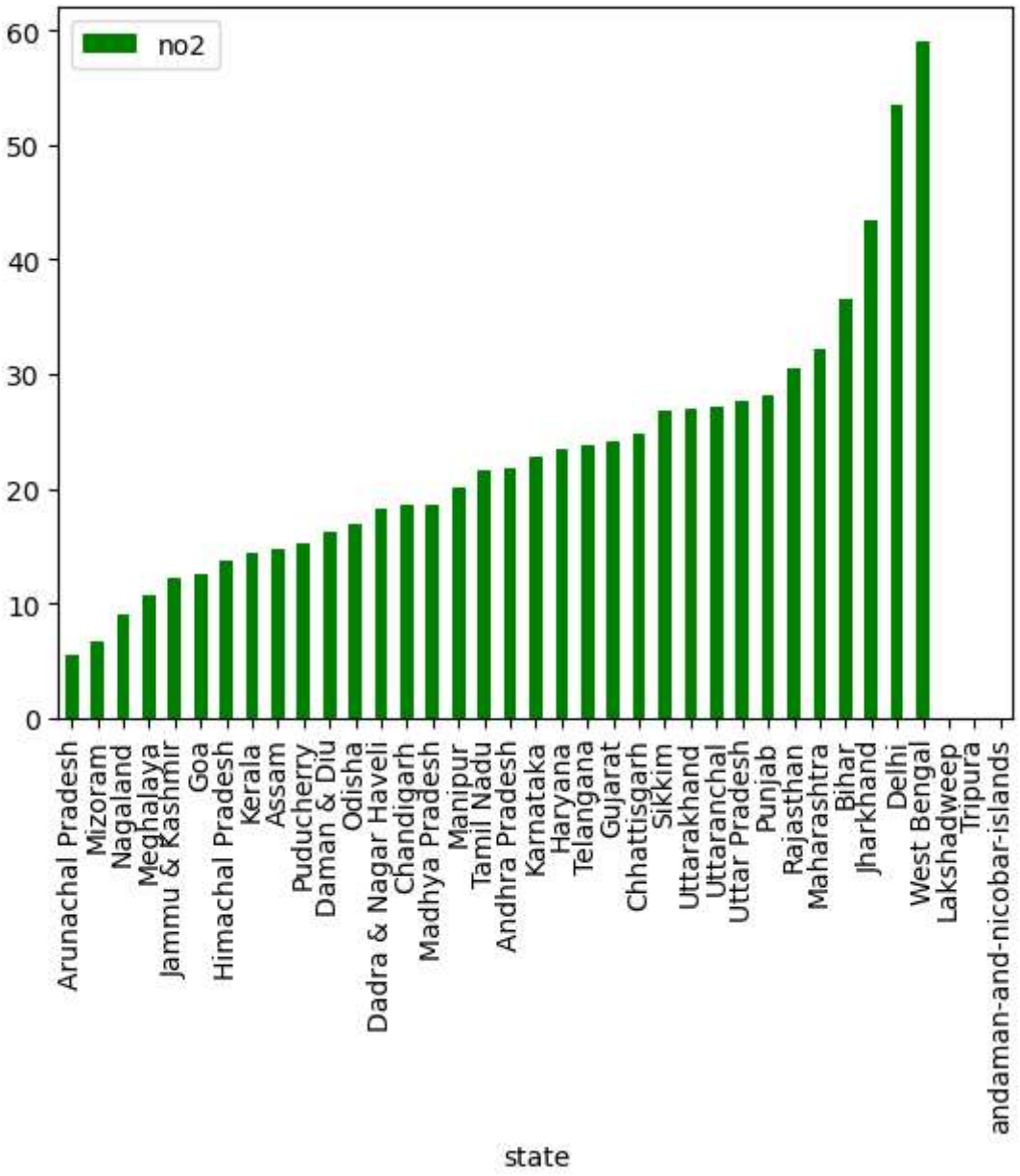
Out[8]:
```
(435742, 13)
```

Visualization for states with highest pollutants

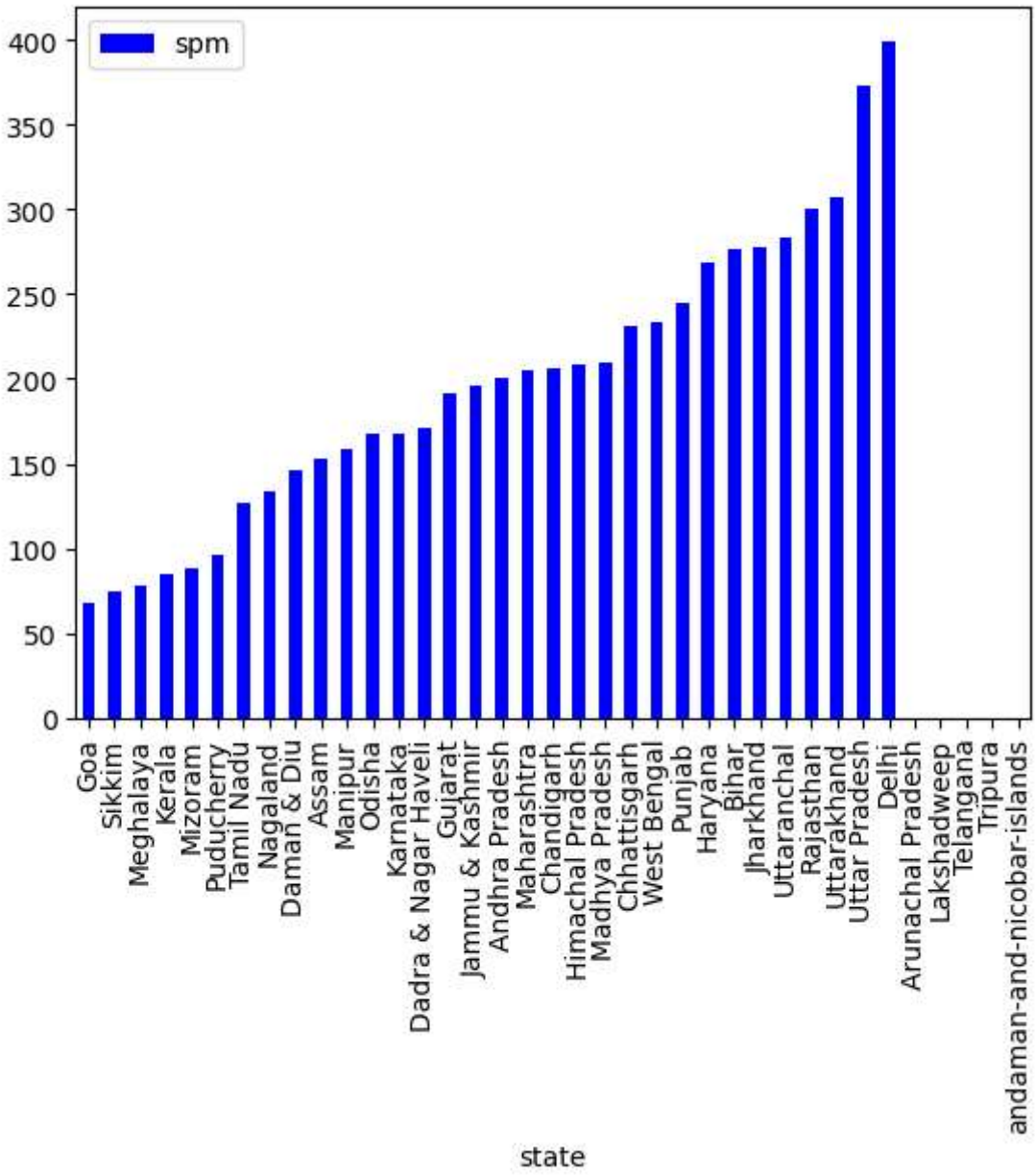
In [9]:
```
data[['so2','state']].groupby(["state"]).mean().sort_values(by='so2').head(20)
plt.show()
```



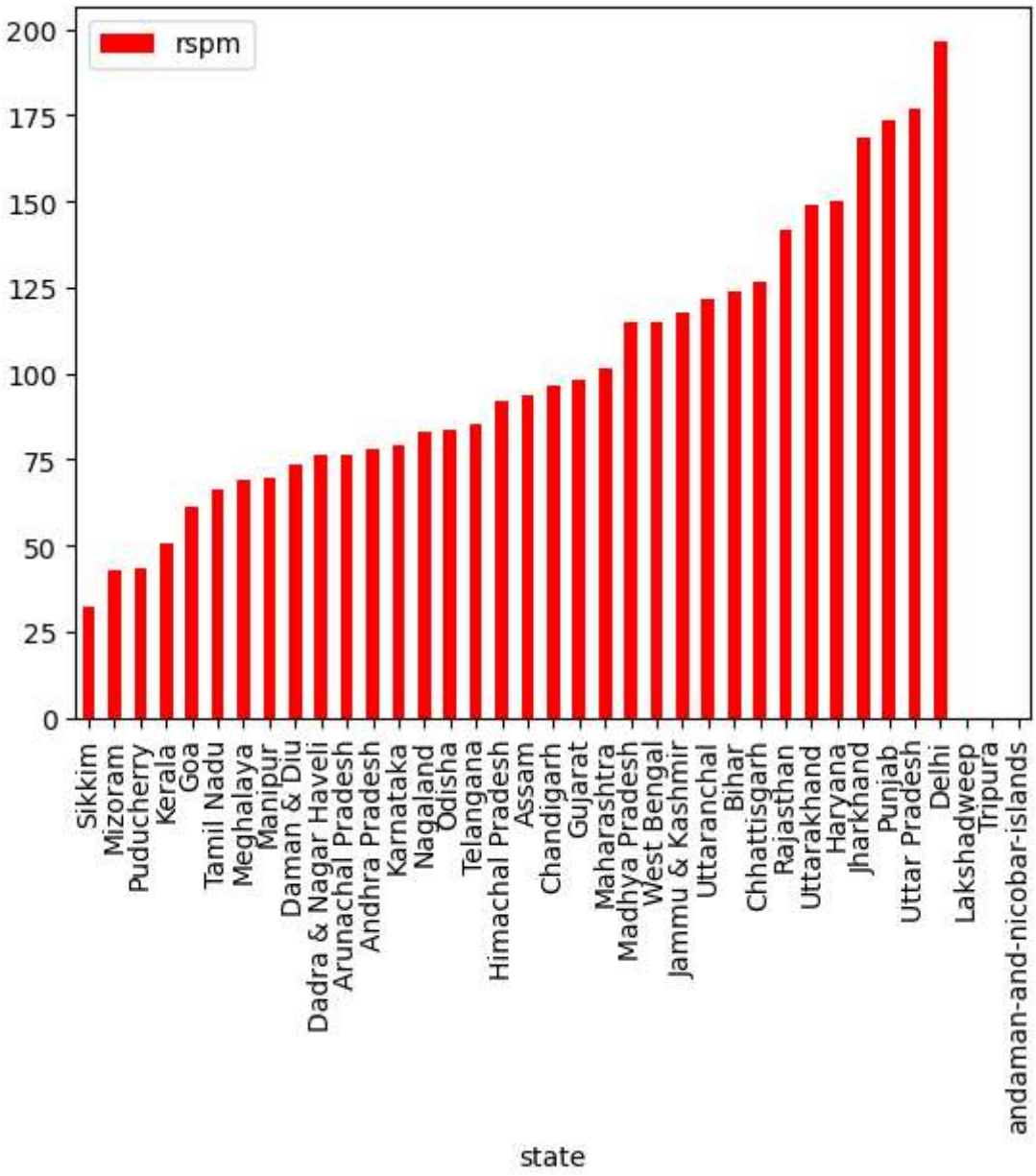
In [10]:
```
data[['no2','state']].groupby(["state"]).mean().sort_values(by='no2').plot.bar
plt.show()
```



In [11]:
```
data[['spm','state']].groupby(["state"]).mean().sort_values(by='spm').plot.bar
plt.show()
```



In [12]:
```
data[['rspm','state']].groupby(["state"]).mean().sort_values(by='rspm').plot.b
plt.show()
```



NULL VALUES COUNT

In [13]:
```
data.isna().sum() #print the sum of null values for each columns
```

Out[13]:
```
stn_code                       144077
sampling_date                       3
state                               0
location                            3
agency                         149481
type                             5393
so2                             34646
no2                             16233
rspm                            40222
spm                            237387
location_monitoring_station     27491
pm2_5                          426428
date                                7
dtype: int64
```

DROP UNNECESSARY COLUMN

In [14]:
```
data.drop(['stn_code','agency','sampling_date','location_monitoring_station'],
```

In [15]:
```
data.head(10)
```

Out[15]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	2/1/1990
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	NaN	NaN	NaN	2/1/1990
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	2/1/1990
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	3/1/1990
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	NaN	NaN	NaN	3/1/1990
5	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.4	25.7	NaN	NaN	NaN	3/1/1990
6	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	5.4	17.1	NaN	NaN	NaN	4/1/1990
7	Andhra Pradesh	Hyderabad	Industrial Area	4.7	8.7	NaN	NaN	NaN	4/1/1990
8	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.2	23.0	NaN	NaN	NaN	4/1/1990
9	Andhra Pradesh	Hyderabad	Industrial Area	4.0	8.9	NaN	NaN	NaN	5/1/1990

CALCULATE TOTAL MISSING VALUES AND THEIR PERCENTAGE

In [16]:
```
total = data.isnull().sum().sort_values(ascending=False)
```

In [17]:
```
total.head()
```

Out[17]:
```
pm2_5    426428
spm      237387
rspm      40222
so2       34646
no2       16233
dtype: int64
```

Calculate the percent of null values for each columns (sum of null values / total non-null value) *100

In [18]:
```
percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascendin
```

In [19]:
```
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
```

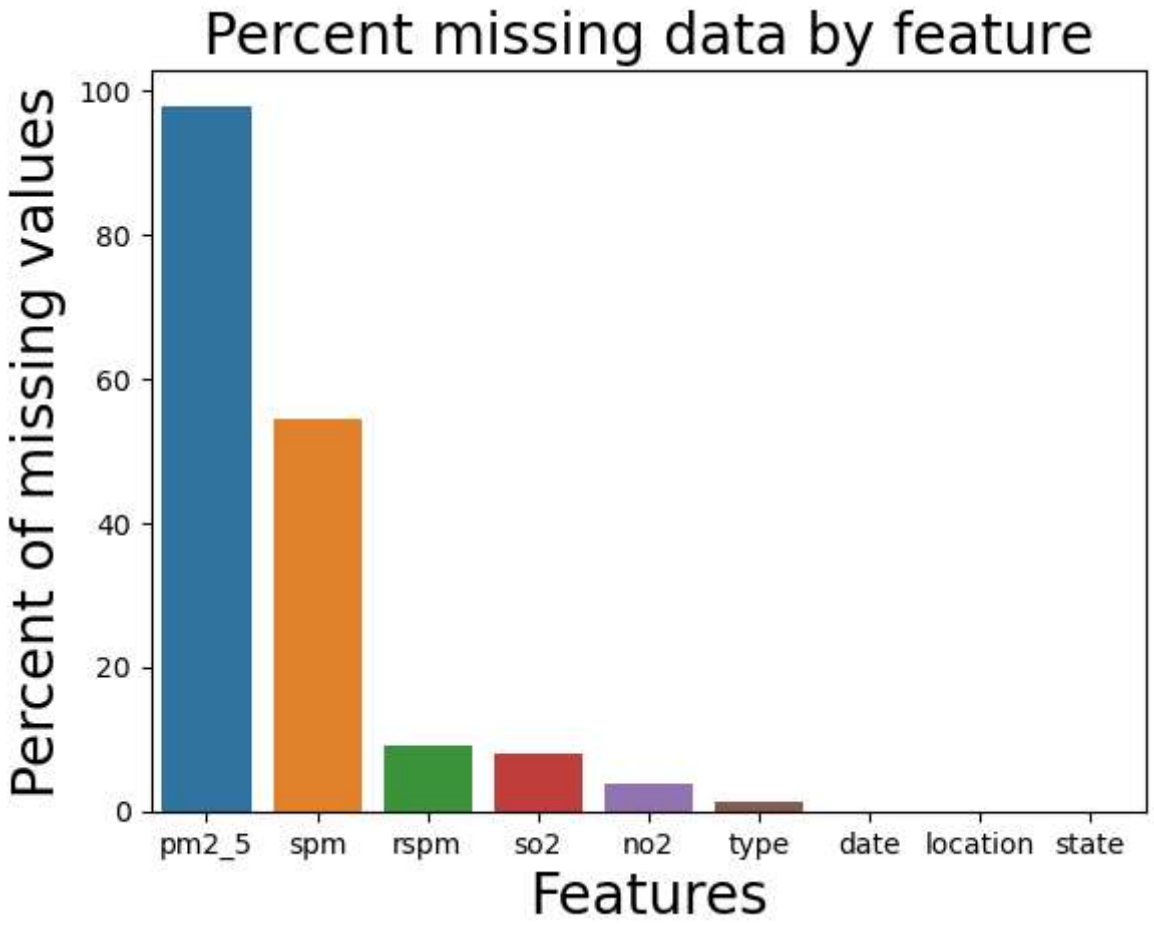
In [20]:
```
missing_data.head()
```

Out[20]:

	Total	Percent
pm2_5	426428	97.862497
spm	237387	54.478797
rspm	40222	9.230692
so2	34646	7.951035
no2	16233	3.725370

PERCENT OF MISSING VALUE (BAR PLOT)

In [21]:
```
sns.barplot(x=missing_data.index, y=missing_data['Percent'])
plt.xlabel('Features', fontsize=20)
plt.ylabel('Percent of missing values', fontsize=20)
plt.title('Percent missing data by feature', fontsize=20)
plt.show()
```



MEAN DISTRIBUTION BY STATE

In [22]:
```
data.groupby('state')[['spm','pm2_5','rspm','so2','no2']].mean()
```

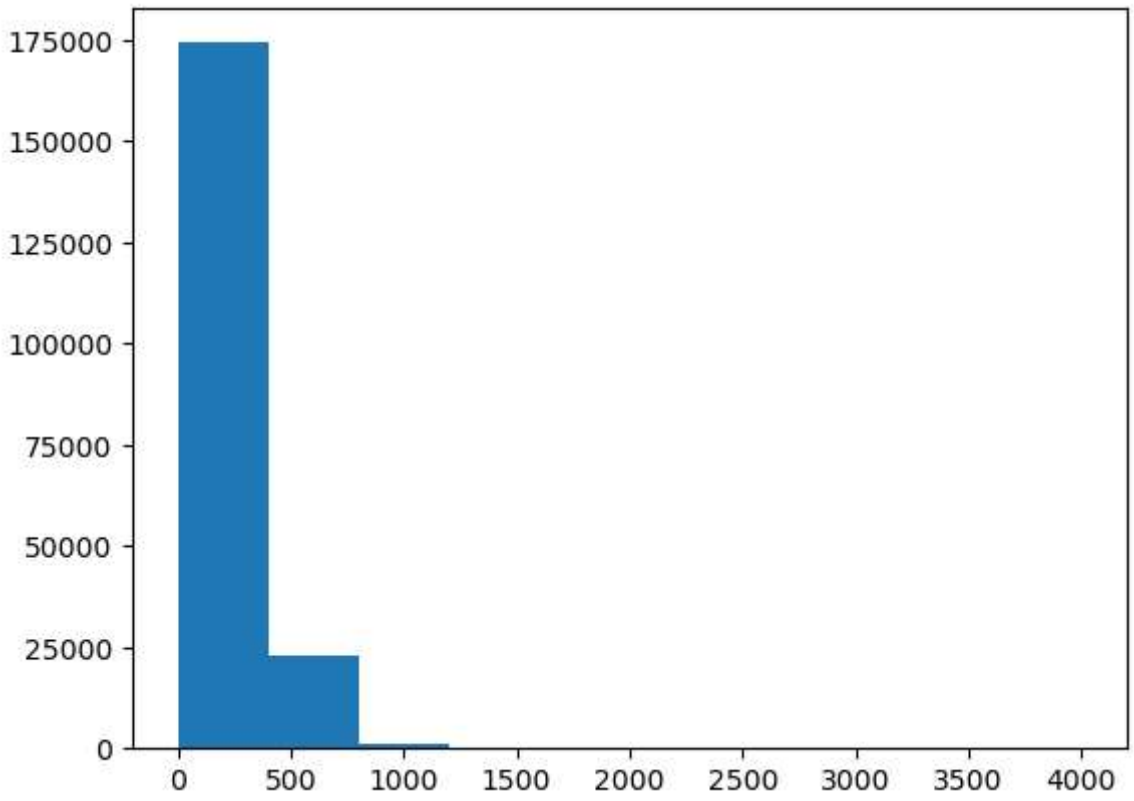
Out[22]:

state	spm	pm2_5	rspm	so2	no2
Andhra Pradesh	200.260378	NaN	78.182824	7.284845	21.704451
Arunachal Pradesh	NaN	NaN	76.629213	3.179104	5.469697
Assam	153.355386	NaN	93.724912	6.723263	14.793691
Bihar	276.917416	NaN	123.705176	19.381476	36.575525
Chandigarh	206.056150	NaN	96.587079	2.676986	18.619404
Chhattisgarh	231.290969	NaN	126.472399	12.846609	24.815961
Dadra & Nagar Haveli	170.545024	30.511628	76.536530	8.939587	18.293959
Daman & Diu	145.681416	27.886364	73.749431	8.192958	16.168926
Delhi	399.402088	95.113208	196.639771	8.737273	53.489147
Goa	67.254193	18.855612	61.212766	6.827913	12.506337
Gujarat	191.567930	30.729696	98.244510	16.656343	24.065631
Haryana	268.264804	NaN	149.860537	14.064957	23.428311
Himachal Pradesh	208.575630	NaN	91.870202	2.667013	13.658688
Jammu & Kashmir	196.221053	NaN	117.449483	7.180521	12.213181
Jharkhand	277.940746	NaN	168.517763	23.485794	43.366341
Karnataka	168.001743	NaN	79.371801	10.223099	22.702837
Kerala	84.419791	NaN	50.636064	5.322350	14.421889
Lakshadweep	NaN	NaN	NaN	NaN	NaN
Madhya Pradesh	210.067545	65.064565	114.717967	11.587410	18.639596
Maharashtra	205.255823	NaN	101.479608	17.366863	32.115370
Manipur	158.657895	NaN	69.815789	3.900000	20.173684
Meghalaya	78.002445	NaN	68.988442	8.955908	10.659706
Mizoram	87.833333	NaN	42.716466	2.085009	6.682171
Nagaland	133.311449	NaN	83.357027	2.059736	8.947265
Odisha	167.609844	42.204089	83.619824	5.275874	16.899568
Puducherry	95.598188	NaN	43.418217	11.970639	15.279496
Punjab	244.918926	NaN	173.493711	10.628598	28.085846
Rajasthan	300.735397	NaN	142.016832	7.665725	30.441008
Sikkim	75.000000	NaN	32.000000	19.800000	26.800000
Tamil Nadu	126.729064	29.550441	66.585638	11.315134	21.601202
Telangana	NaN	43.968927	85.043008	5.418609	23.864005
Tripura	NaN	NaN	NaN	NaN	NaN
Uttar Pradesh	372.663688	NaN	176.952308	12.528500	27.610095
Uttarakhand	306.758923	NaN	148.978906	24.372957	26.938090
Uttaranchal	283.335714	NaN	121.694340	24.697736	27.163019

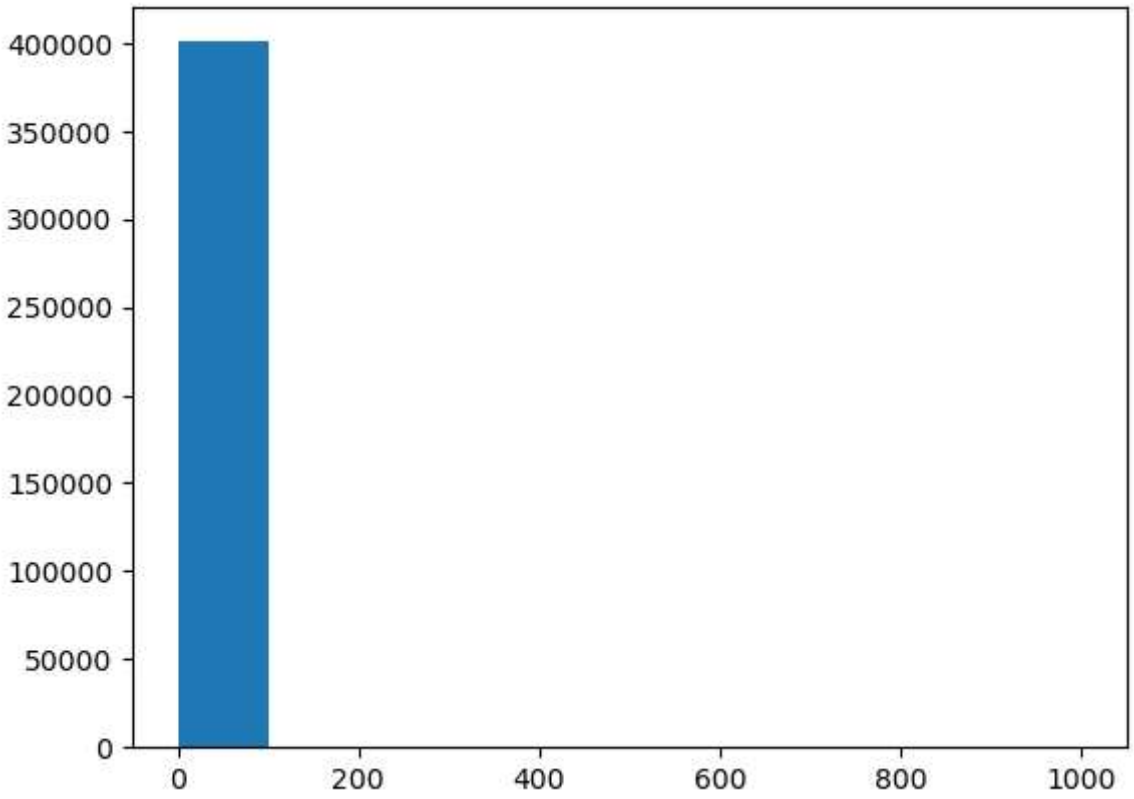
state	spm	pm2_5	rspm	so2	no2
West Bengal	233.506524	64.890625	115.039909	12.608766	59.075731
andaman-and-nicobar-islands	NaN	NaN	NaN	NaN	NaN

CHECKING DATA DISTRIBUTION

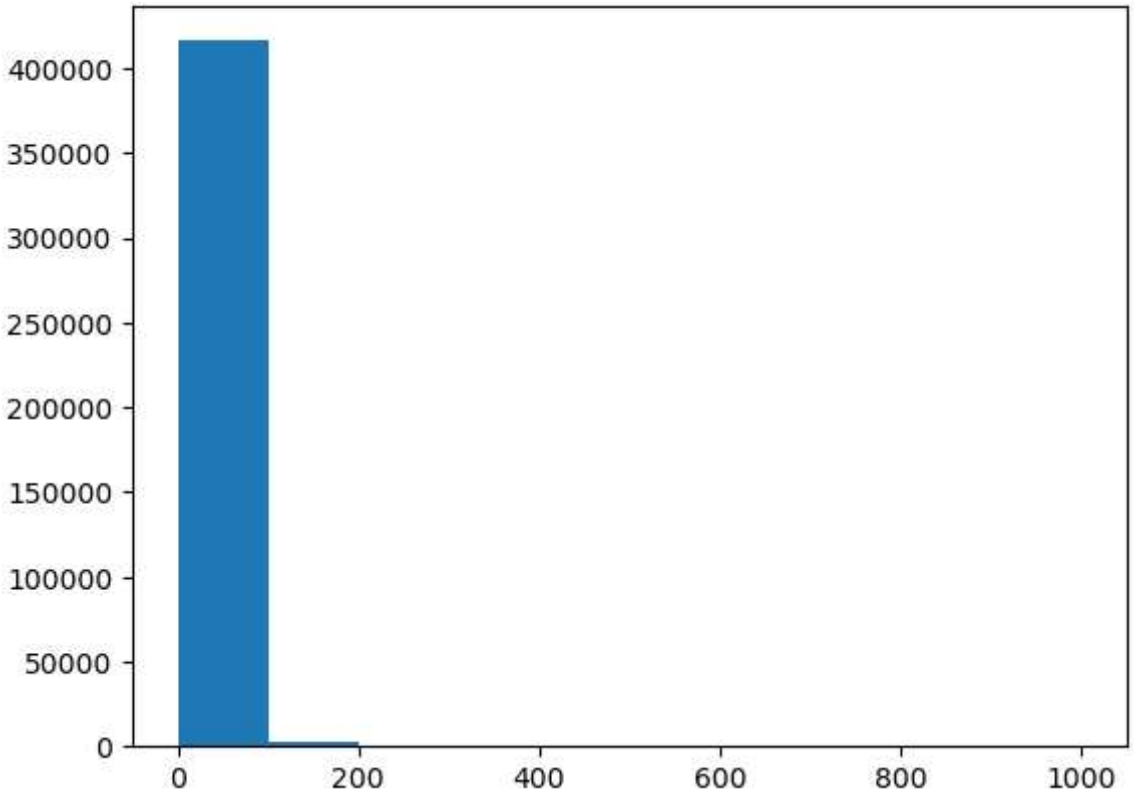
In [23]:
```
plt.hist(data.spm,range=(0.0,4000)) #spm
plt.show()
```



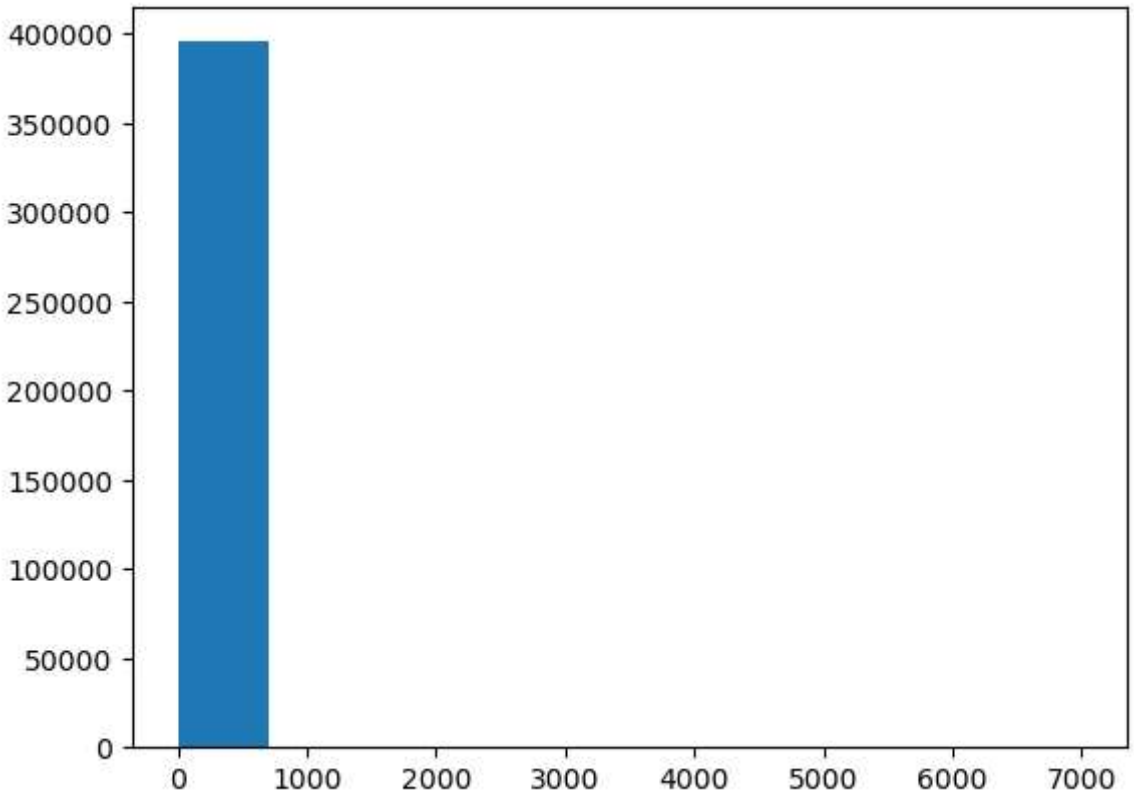
In [24]:
```
plt.hist(data.so2,range=(0,1000)) #so2
plt.show()
```



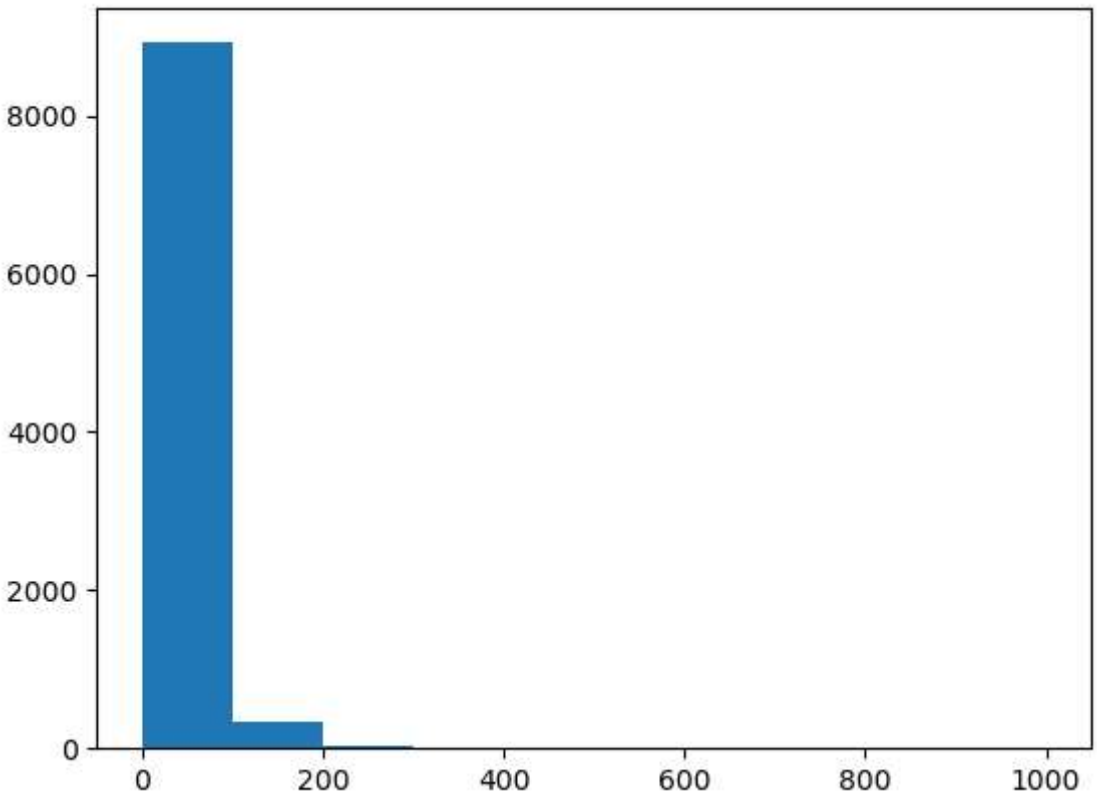
In [25]:
```
plt.hist(data.no2,range=(0,1000)) #no2
plt.show()
```



In [26]:
```
plt.hist(data.rspm,range=(0,7000)) #rspm
plt.show()
```



In [27]:
```
plt.hist(data.pm2_5,range=(0,1000)) #pm2_5
plt.show()
```



CONCLUSION: NO POTENTIAL OUTLIERS

FILL MISSING VALUES BY MEAN (GROUP BY STATE)

In [28]:
```
grp_state = data.groupby('state')
```

In [29]:
```
def mean(series):
    return series.fillna(series.mean())
```

In [30]:
```
data['rspm']=grp_state['rspm'].transform(mean)  #fill value with mean value gr
data['so2']=grp_state['so2'].transform(mean)
data['no2']=grp_state['no2'].transform(mean)
data['spm']=grp_state['spm'].transform(mean)
data['pm2_5']=grp_state['pm2_5'].transform(mean)
```

In [31]:
```
data.describe()
```

Out[31]:

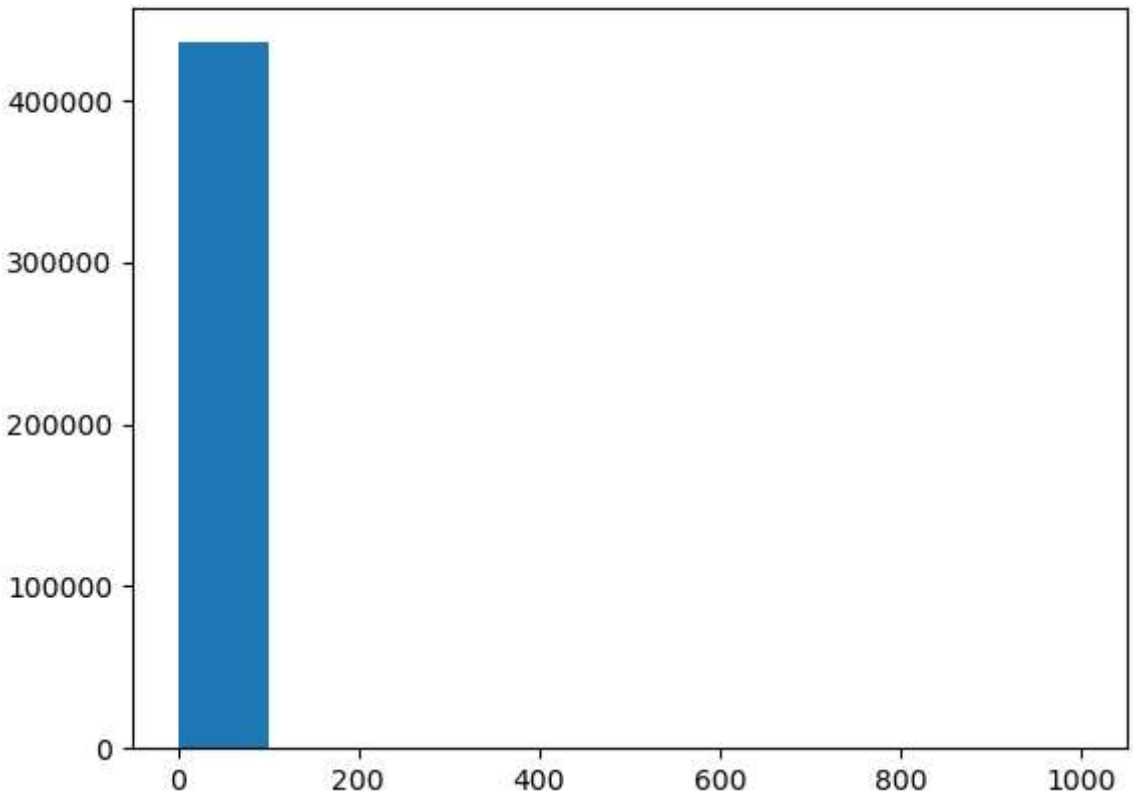
	so2	no2	rspm	spm	pm2_5
count	435739.000000	435739.000000	435739.000000	431671.000000	123689.000000
mean	10.589141	25.663170	109.680907	215.198748	48.317274
std	10.863679	18.271145	72.519399	117.470157	21.455206
min	0.000000	0.000000	0.000000	0.000000	3.000000
25%	4.800000	14.000000	59.000000	142.000000	30.729696
50%	8.000000	21.800000	93.000000	205.255823	42.204089
75%	13.000000	32.000000	142.016832	248.000000	64.890625
max	909.000000	876.000000	6307.033333	3380.000000	504.000000

In [32]:
```
data.isna().sum() #some null value remains since some state have one value(i.e
```

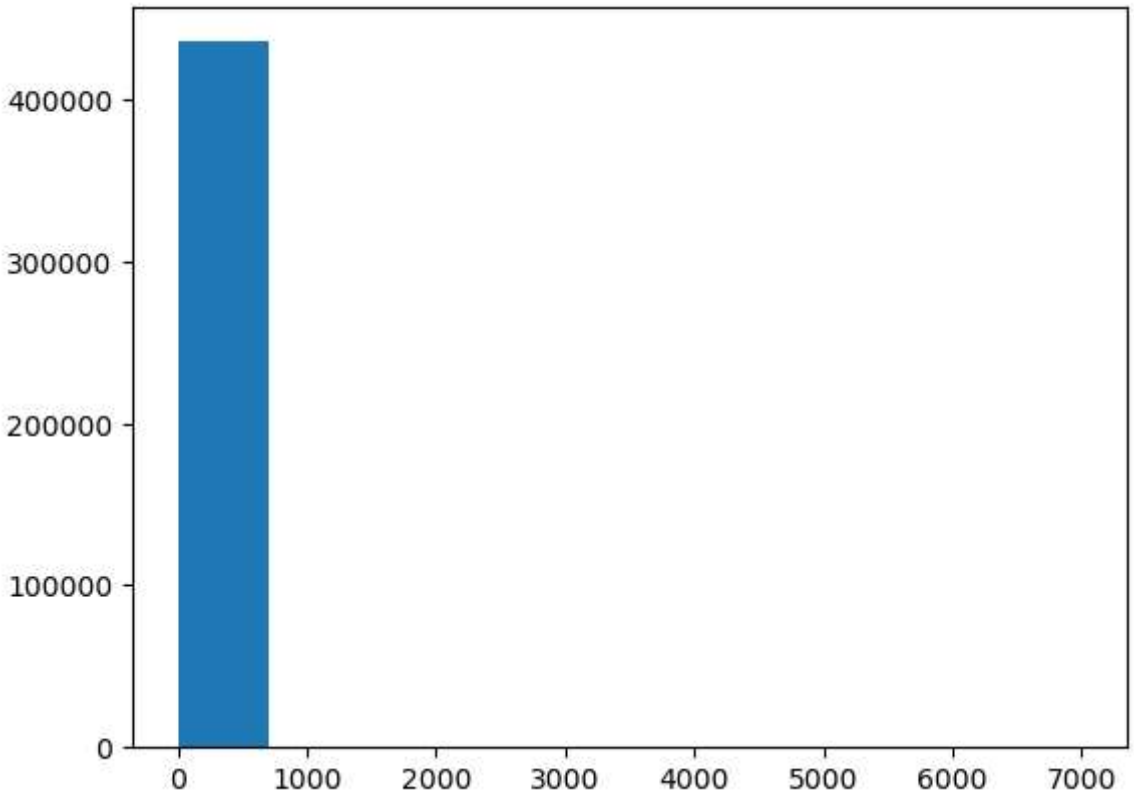
Out[32]:
```
state            0
location         3
type          5393
so2              3
no2              3
rspm             3
spm           4071
pm2_5       312053
date             7
dtype: int64
```

Data Distribution after Replacing Null value

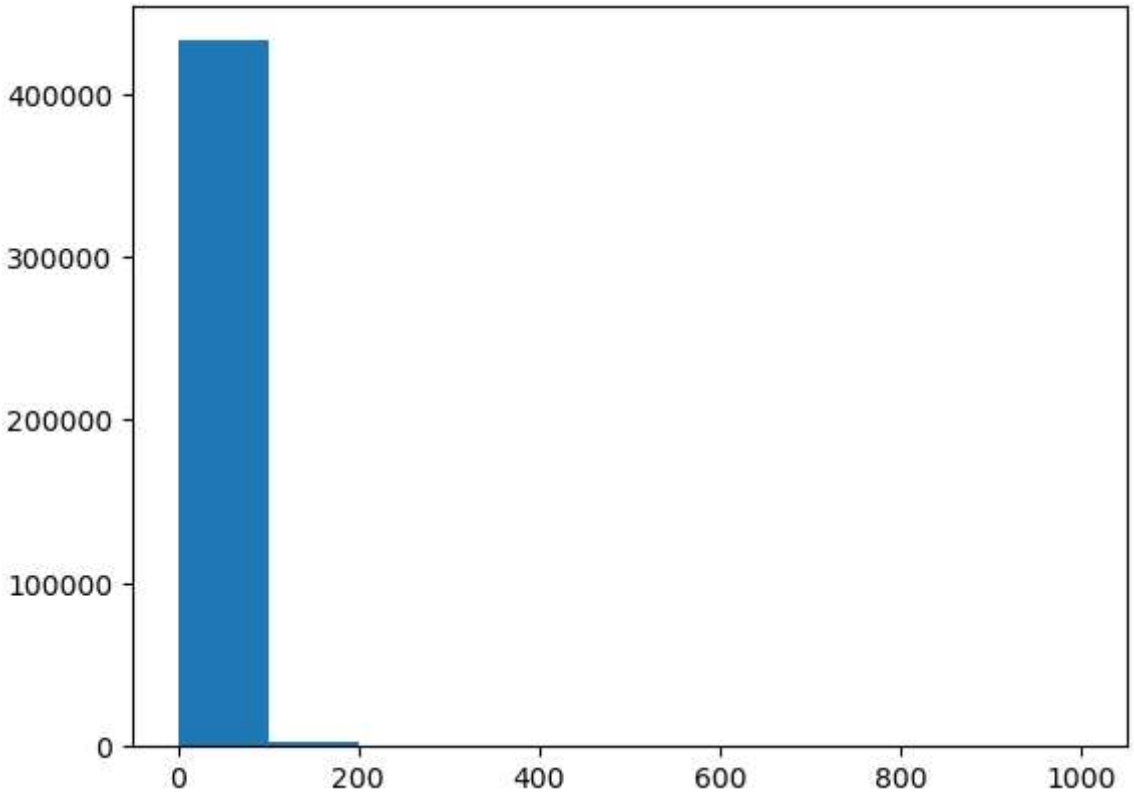
In [33]:
```
plt.hist(data.so2,range=(0,1000))
plt.show()
```



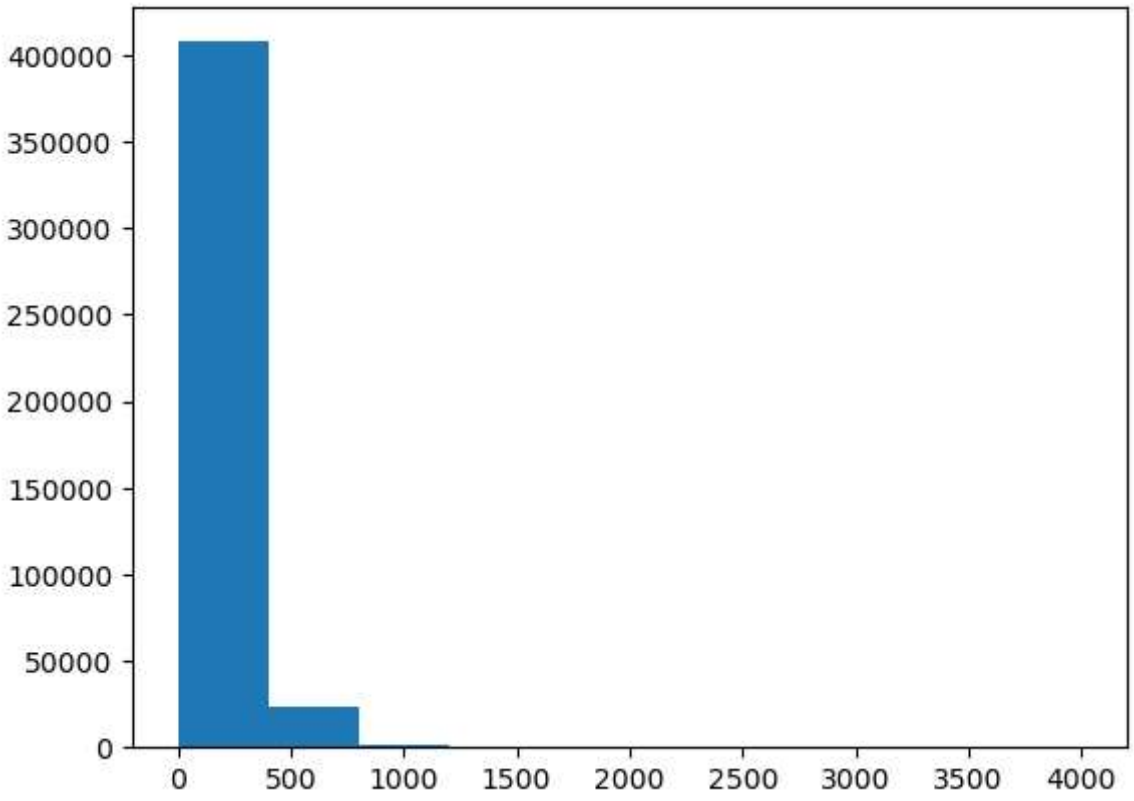
In [34]:
```
plt.hist(data.rspm,range=(0,7000))
plt.show()
```



In [35]:
```
plt.hist(data.no2,range=(0.0,1000))
plt.show()
```

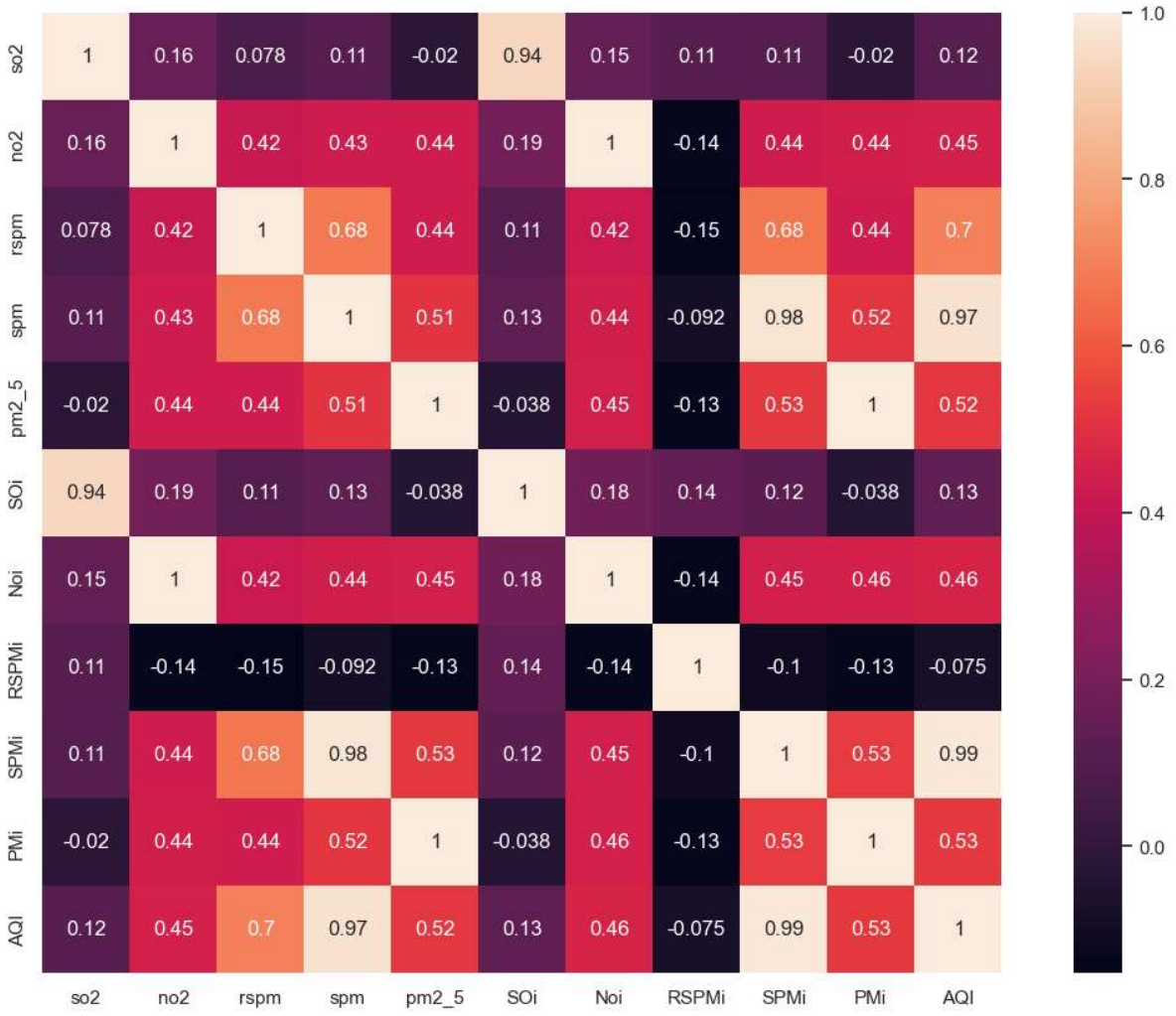


In [36]:
```
plt.hist(data.spm,range=(0.0,4000)) #spm
plt.show()
```



In [109]:
```
#Scatter plots of all columns
sns.set()
cols = ['so2', 'no2', 'rspm', 'spm', 'pm2_5']
sns.pairplot(data[cols], size = 2.5)
plt.show()
```

In [111]:
```
#Correlation matrix
corrmat = data.corr()
f, ax = plt.subplots(figsize = (15, 10))
sns.heatmap(corrmat, vmax = 1, square = True, annot = True)
plt.show()
```



In [37]:
```
data.tail(10)
```

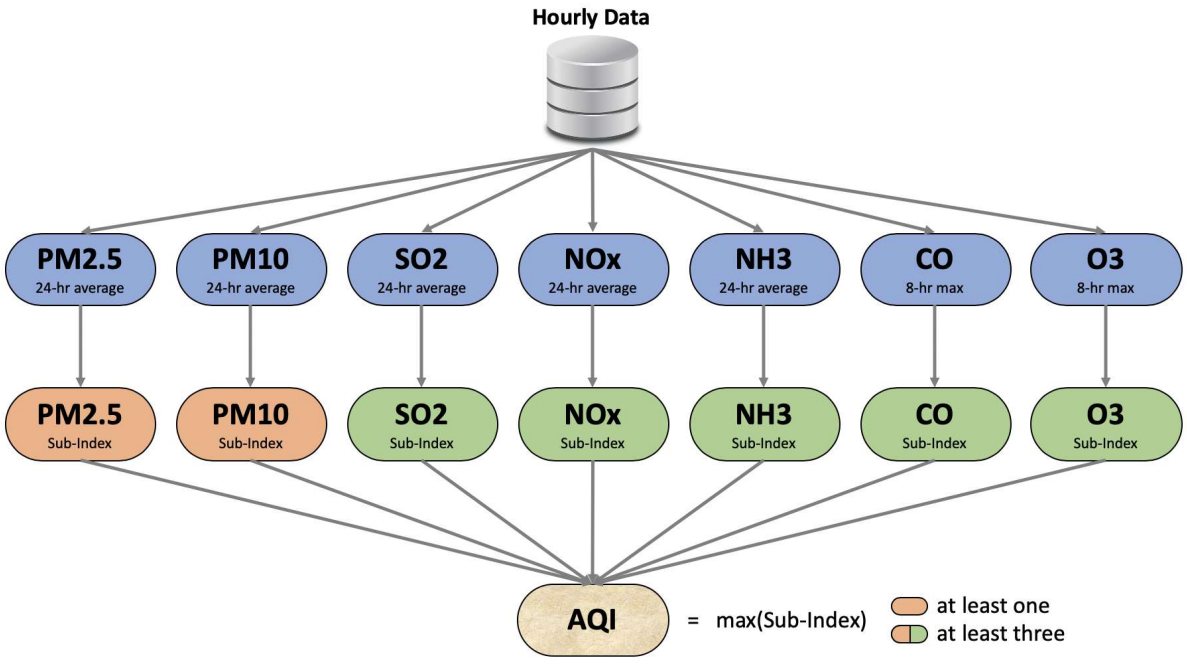
Out[37]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date
435732	West Bengal	ULUBERIA	RIRUO	22.0	50.0	145.0	233.506524	64.890625	12/9/2015
435733	West Bengal	ULUBERIA	RIRUO	34.0	61.0	161.0	233.506524	64.890625	12/12/2015
435734	West Bengal	ULUBERIA	RIRUO	20.0	44.0	148.0	233.506524	64.890625	12/15/2015
435735	West Bengal	ULUBERIA	RIRUO	17.0	44.0	131.0	233.506524	64.890625	12/18/2015
435736	West Bengal	ULUBERIA	RIRUO	18.0	45.0	140.0	233.506524	64.890625	12/21/2015
435737	West Bengal	ULUBERIA	RIRUO	22.0	50.0	143.0	233.506524	64.890625	12/24/2015
435738	West Bengal	ULUBERIA	RIRUO	20.0	46.0	171.0	233.506524	64.890625	12/29/2015
435739	andaman-and-nicobar-islands	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435740	Lakshadweep	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435741	Tripura	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

CALCULATE AIR QUALITY INDEX FOR SO2 BASED ON FORMULA

The air quality index is a piecewise linear function of the pollutant concentration. At the boundary between AQI categories, there is a discontinuous jump of one AQI unit. To convert from concentration to AQI this equation is used

$$I = I_{low} + \frac{I_{high} - I_{low}}{C_{high} - C_{low}}(C - C_{low})$$



What is sulfur dioxide?

Sulfur dioxide is a gas. It is invisible and has a nasty, sharp smell. It reacts easily with other substances to form harmful compounds, such as sulfuric acid, sulfurous acid and sulfate particles.

About 99% of the sulfur dioxide in air comes from human sources. The main source of sulfur dioxide in the air is industrial activity that processes materials that contain sulfur, eg the generation of electricity from coal, oil or gas that contains sulfur. Some mineral ores also contain sulfur, and sulfur dioxide is released when they are processed. In addition, industrial activities that burn fossil fuels containing sulfur can be important sources of sulfur dioxide.

Sulfur dioxide is also present in motor vehicle emissions, as the result of fuel combustion. In the past, motor vehicle exhaust was an important, but not the main, source of sulfur dioxide in air. However, this is no longer the case.

In [38]:
```
def cal_SOi(so2):
    si=0
    if (so2<=40):
     si= so2*(50/40)
    elif (so2>40 and so2<=80):
     si= 50+(so2-40)*(50/40)
    elif (so2>80 and so2<=380):
     si= 100+(so2-80)*(100/300)
    elif (so2>380 and so2<=800):
     si= 200+(so2-380)*(100/420)
    elif (so2>800 and so2<=1600):
     si= 300+(so2-800)*(100/800)
    elif (so2>1600):
     si= 400+(so2-1600)*(100/800)
    return si
data['SOi']=data['so2'].apply(cal_SOi)
df= data[['so2','SOi']]
df.head()
```

Out[38]:

	so2	SOi
0	4.8	6.000
1	3.1	3.875
2	6.2	7.750
3	6.3	7.875
4	4.7	5.875

CALCULATE AIR QUALITY INDEX FOR NO2 BASED ON FORMULA

In [39]:
```
def cal_Noi(no2):
    ni=0
    if(no2<=40):
     ni= no2*50/40
    elif(no2>40 and no2<=80):
     ni= 50+(no2-40)*(50/40)
    elif(no2>80 and no2<=180):
     ni= 100+(no2-80)*(100/100)
    elif(no2>180 and no2<=280):
     ni= 200+(no2-180)*(100/100)
    elif(no2>280 and no2<=400):
     ni= 300+(no2-280)*(100/120)
    else:
     ni= 400+(no2-400)*(100/120)
    return ni
data['Noi']=data['no2'].apply(cal_Noi)
df= data[['no2','Noi']]
df.head()
```

Out[39]:

	no2	Noi
0	17.4	21.750
1	7.0	8.750
2	28.5	35.625
3	14.7	18.375
4	7.5	9.375

CALCULATE AIR QUALITY INDEX FOR RSPM BASED ON FORMULA

In [40]:
```
def cal_RSPMi(rspm):
    rpi=0
    if(rspm<=100):
     rpi = rspm
    elif(rspm>=101 and rspm<=150):
     rpi= 101+(rspm-101)*((200-101)/(150-101))
    elif(rspm>=151 and rspm<=350):
     ni= 201+(rspm-151)*((300-201)/(350-151))
    elif(rspm>=351 and rspm<=420):
     ni= 301+(rspm-351)*((400-301)/(420-351))
    elif(rspm>420):
     ni= 401+(rspm-420)*((500-401)/(420-351))
    return rpi
data['RSPMi']=data['rspm'].apply(cal_RSPMi)
df= data[['rspm','RSPMi']]
df.head()
```

Out[40]:

	rspm	RSPMi
0	78.182824	78.182824
1	78.182824	78.182824
2	78.182824	78.182824
3	78.182824	78.182824
4	78.182824	78.182824

In [41]:
```
df.tail()
```

Out[41]:

	rspm	RSPMi
435737	143.0	185.857143
435738	171.0	0.000000
435739	NaN	0.000000
435740	NaN	0.000000
435741	NaN	0.000000

CALCULATE AIR QUALITY INDEX FOR SPM BASED ON FORMULA

In [42]:
```
def cal_SPMi(spm):
    spi=0
    if(spm<=50):
     spi=spm*50/50
    elif(spm>50 and spm<=100):
     spi=50+(spm-50)*(50/50)
    elif(spm>100 and spm<=250):
     spi= 100+(spm-100)*(100/150)
    elif(spm>250 and spm<=350):
     spi=200+(spm-250)*(100/100)
    elif(spm>350 and spm<=430):
     spi=300+(spm-350)*(100/80)
    else:
     spi=400+(spm-430)*(100/430)
    return spi

data['SPMi']=data['spm'].apply(cal_SPMi)
df= data[['spm','SPMi']]
df.head()
```

Out[42]:

	spm	SPMi
0	200.260378	166.840252
1	200.260378	166.840252
2	200.260378	166.840252
3	200.260378	166.840252
4	200.260378	166.840252

CALCULATE AIR QUALITY INDEX FOR PM 2.5 BASED ON FORMULA

In [43]:
```
def cal_pmi(pm2_5):
    pmi=0
    if(pm2_5<=50):
     pmi=pm2_5*(50/50)
    elif(pm2_5>50 and pm2_5<=100):
     pmi=50+(pm2_5-50)*(50/50)
    elif(pm2_5>100 and pm2_5<=250):
     pmi= 100+(pm2_5-100)*(100/150)
    elif(pm2_5>250 and pm2_5<=350):
     pmi=200+(pm2_5-250)*(100/100)
    elif(pm2_5>350 and pm2_5<=450):
     pmi=300+(pm2_5-350)*(100/100)
    else:
     pmi=400+(pm2_5-430)*(100/80)
    return pmi
data['PMi']=data['pm2_5'].apply(cal_pmi)
df= data[['pm2_5','PMi']]
df.head()
```

Out[43]:

	pm2_5	PMi
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

In [44]:
```
type(data['PMi'])
```

Out[44]:
```
pandas.core.series.Series
```

Based on the measured ambient concentrations, corresponding standards and likely health impact, a sub-index is calculated for each of these pollutants. The worst sub-index reflects overall AQI.If multiple pollutants are measured at a monitoring site, then the largest or "dominant" AQI value is reported for the location

In [45]:
```
def cal_aqi(si,ni,rspmi,spmi):
    aqi=0
    if(si>ni and si>rspmi and si>spmi):
     aqi=si
    if(ni>si and ni>rspmi and ni>spmi ):
     aqi=ni
    if(rspmi>si and rspmi>ni and rspmi>spmi ):
     aqi=rspmi
    if(spmi>si and spmi>ni and spmi>rspmi):
     aqi=spmi
    return aqi

data['AQI']=data.apply(lambda x:cal_aqi(x['SOi'],x['Noi'],x['RSPMi'],x['SPMi']
df= data[['state','SOi','Noi','RSPMi','SPMi','AQI']]
df.head()
```

Out[45]:

	state	SOi	Noi	RSPMi	SPMi	AQI
0	Andhra Pradesh	6.000	21.750	78.182824	166.840252	166.840252
1	Andhra Pradesh	3.875	8.750	78.182824	166.840252	166.840252
2	Andhra Pradesh	7.750	35.625	78.182824	166.840252	166.840252
3	Andhra Pradesh	7.875	18.375	78.182824	166.840252	166.840252
4	Andhra Pradesh	5.875	9.375	78.182824	166.840252	166.840252

In [46]:
```
data.head()
```

Out[46]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date	SOi	
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	78.182824	200.260378	NaN	2/1/1990	6.000	2
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	78.182824	200.260378	NaN	2/1/1990	3.875	
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	78.182824	200.260378	NaN	2/1/1990	7.750	3
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	78.182824	200.260378	NaN	3/1/1990	7.875	1
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	78.182824	200.260378	NaN	3/1/1990	5.875	

In [ ]:

AQI RANGE for corresponding AQI value

In [47]:
```
def AQI_Range(x):
    if x<=50:
        return "Good"
    elif x>50 and x<=100:
        return "Moderate"
    elif x>100 and x<=200:
        return "Poor"
    elif x>200 and x<=300:
        return "Unhealthy"
    elif x>300 and x<=400:
        return "Very unhealthy"
    elif x>400:
        return "Hazardous"

data['AQI_Range'] = data['AQI'] .apply(AQI_Range)
data.head()
```

Out[47]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date	SOi	
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	78.182824	200.260378	NaN	2/1/1990	6.000	2
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	78.182824	200.260378	NaN	2/1/1990	3.875	8
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	78.182824	200.260378	NaN	2/1/1990	7.750	35
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	78.182824	200.260378	NaN	3/1/1990	7.875	18
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	78.182824	200.260378	NaN	3/1/1990	5.875	9

Type *Markdown* and LaTeX: α^2

In [48]:
```
d=data #saving data in new value
d.head()
```

Out[48]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date	SOi	
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	78.182824	200.260378	NaN	2/1/1990	6.000	2
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	78.182824	200.260378	NaN	2/1/1990	3.875	8
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	78.182824	200.260378	NaN	2/1/1990	7.750	35
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	78.182824	200.260378	NaN	3/1/1990	7.875	18
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	78.182824	200.260378	NaN	3/1/1990	5.875	9

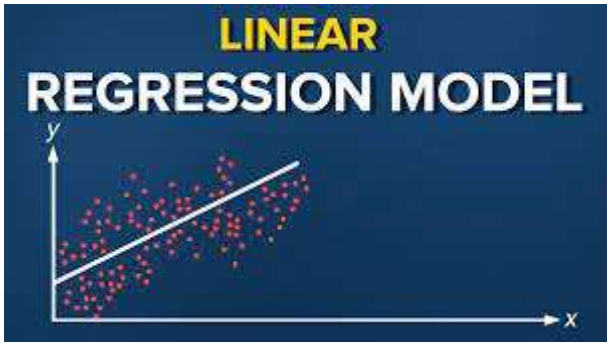
Remove the rows with null values

In [49]:
```
data=data.dropna(subset=['spm']) #spm
```

In [50]:
```
data=data.dropna(subset=['pm2_5']) #spm
```

In [51]:
```
data.isna().sum() #all null values removed
```

Out[51]:
```
state           0
location        0
type         1925
so2             0
no2             0
rspm            0
spm             0
pm2_5           0
date            1
SOi             0
Noi             0
RSPMi           0
SPMi            0
PMi             0
AQI             0
AQI_Range       0
dtype: int64
```



Linear Regression prediction

1. Using SOi, NOi, RSPMi, SPMi TO PREDICT AQI

In [52]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [53]:
```
X = data[['SOi','Noi','RSPMi','SPMi']]
y = data['AQI']
y.head()
```

Out[53]:
```
64445    76.53653
64446    76.53653
64447    76.53653
64448    76.53653
64449    76.53653
Name: AQI, dtype: float64
```

In [54]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_
```

In [55]:
```
X_train.head()
```

Out[55]:

	SOi	Noi	RSPMi	SPMi
360028	3.750	40.000	51.000000	117.819376
434684	2.500	46.250	87.000000	189.004349
64493	13.250	23.375	115.142857	210.000000
433906	5.000	55.000	115.142857	189.004349
183595	34.375	26.000	91.000000	106.000000

In [56]:
```
LR = LinearRegression()
LR.fit(X_train, y_train)
```

Out[56]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [57]:
```
print('Intercept',LR.intercept_)
```

```
Intercept 4.168841608263278
```

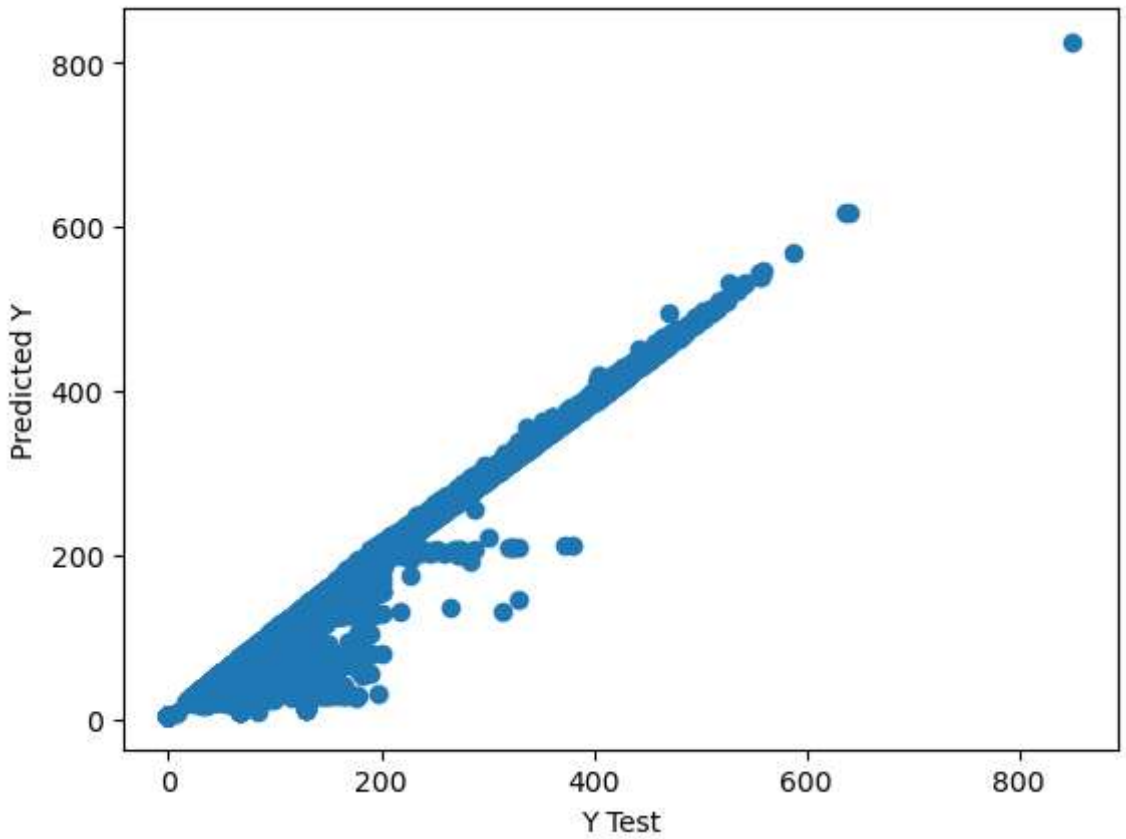
In [58]:
```
print('Coefficients',LR.coef_)
```

```
Coefficients [0.02669704 0.06811693 0.05053633 0.9544088 ]
```

In [59]:
```
predictions = LR.predict(X_test)
```

In [60]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[60]:
```
Text(0, 0.5, 'Predicted Y')
```



In [61]:
```
LR.score(X_test,y_test)
```

Out[61]:
```
0.9791014082309275
```

In [62]:
```
LR.predict([[4.8,21.75,78.18,100]])
```

Out[62]:
```
array([105.17034145])
```

In [63]:
```
LR.predict([[5.2,7.625,76.53,75.0]])
```

Out[63]:
```
array([80.27526359])
```

In [65]:
```
print('R^2_Square:%.2f '% r2_score(y_test, predictions))
print('MSE:%.2f '% np.sqrt(mean_squared_error(y_test, predictions)))
```

```
R^2_Square:0.98
MSE:12.58
```

Linear Regression Model 2

Using so2, no2, rspm, spm

In [66]:
```
X1= data[['so2','no2','rspm','spm']]
y1 = data['AQI']
y.tail()
```

Out[66]:
```
435734    195.959184
435735    189.004349
435736    189.004349
435737    189.004349
435738    189.004349
Name: AQI, dtype: float64
```

In [67]:
```
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1,y1, test_size=0.2,r
```

In [68]:
```
X_train1.head()
```

Out[68]:

	so2	no2	rspm	spm
360028	3.0	32.0	51.0	126.729064
434684	2.0	37.0	87.0	233.506524
64493	10.6	18.7	108.0	260.000000
433906	4.0	44.0	108.0	233.506524
183595	27.5	20.8	91.0	109.000000

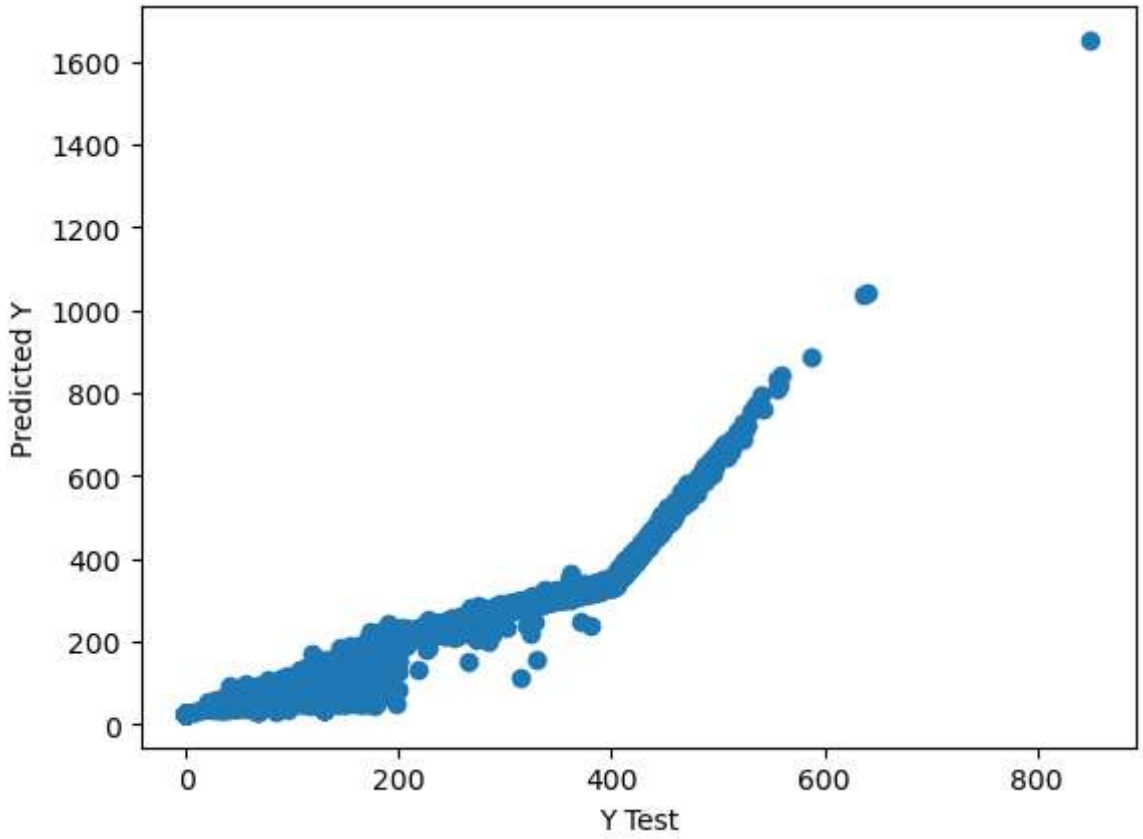
In [69]:
```
LR1 = LinearRegression()
LR1.fit(X_train1, y_train1)
```

Out[69]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [70]:
```
prediction1 = LR1.predict(X_test1)
```

In [71]:
```
plt.scatter(y_test1,prediction1) #scatter plot for actual and predicted values
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[71]:
```
Text(0, 0.5, 'Predicted Y')
```



In [72]:
```
LR1.predict([[9.1,16.3,67,179]])
```

Out[72]:
```
array([154.79655317])
```

In [74]:
```
y_test1_np= np.array(y_test1)
prediction1_np = np.array(prediction1)
```

In [75]:
```
LR1.score(X_test1,y_test1) #accuracy score 76.82%
```

Out[75]:
```
0.9433954748037254
```

Mean Squared error, R^2 sqaured

In [76]:
```
print('R^2_Square:%.2f '% r2_score(y_test1, prediction1))
print('MSE:%.2f '% np.sqrt(mean_squared_error(y_test1, prediction1)))
```

```
R^2_Square:0.94
MSE:20.71
```

Classification of AQI

Logistic Regression

1.Using SOi, Noi, RSPMi, SPMi

In [77]:
```
from sklearn.linear_model import LogisticRegression
```

In [78]:
```
X2 = data[['SOi','Noi','RSPMi','SPMi']]
y2 = data['AQI_Range']
```

In [79]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.33
```

In [80]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train2,y_train2)
```

Out[80]:
```
LogisticRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [81]:
```
predictions = logmodel.predict(X_test)
```

In [82]:
```
logmodel.score(X_test2,y_test2) #accuracy score 89.25 %
```

Out[82]:
```
0.7013289457030756
```

Creating new csv file to store AQI range values inorder to cross verify predicted value

In [83]:
```
new = pd.DataFrame(d)
file1 = 'new1.csv'
new.to_csv(file1,index=True)
```

In [84]:
```
d.tail()
```

Out[84]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date
435737	West Bengal	ULUBERIA	RIRUO	22.0	50.0	143.0	233.506524	64.890625	12/24/2015
435738	West Bengal	ULUBERIA	RIRUO	20.0	46.0	171.0	233.506524	64.890625	12/29/2015
435739	andaman-and-nicobar-islands	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435740	Lakshadweep	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435741	Tripura	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [85]:
```
logmodel.predict([[77.4,147.7,78.182,100]]) #correct
```

Out[85]:
```
array(['Moderate'], dtype=object)
```

In [86]:
```
logmodel.predict([[32.7,35,78.182,203]]) #correct
```

Out[86]:
```
array(['Poor'], dtype=object)
```

In [87]:
```
logmodel.predict([[100,182.2,78.182,400]]) #correct
```

Out[87]:
```
array(['Poor'], dtype=object)
```

Logistic regression model 2

2.Using so2,no2,rspm,spm

In [88]:
```
X3 = data[['so2','no2','rspm','spm']]
y3 = data['AQI_Range']
```

In [89]:
```
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.33
```

In [90]:
```
logmodel2 = LogisticRegression()
logmodel2.fit(X_train3,y_train3)
```

Out[90]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [91]:
```
logmodel2.score(X_test3,y_test3) #very low accuracy score
```

Out[91]:
```
0.6829008986204278
```

In [92]:
```
logmodel2.predict([[4.8,17.4,78.48,200]]) #correct
```

Out[92]:
```
array(['Poor'], dtype=object)
```

In [93]:
```
logmodel2.predict([[67.4,127.7,78.48,215]]) #correct
```

Out[93]:
```
array(['Poor'], dtype=object)
```

In [94]:
```
logmodel2.predict([[2.059,8.94,102,256]])  #wrong
```

Out[94]:
```
array(['Hazardous'], dtype=object)
```

Using Random forest classifier

In [95]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [96]:
```
model = RandomForestClassifier(n_estimators=10)
model.fit(X_train3,y_train3)
```

Out[96]:
```
RandomForestClassifier(n_estimators=10)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [97]:
```
model.score(X_test3,y_test3) #high accuracy score of 99.97 %
```

Out[97]:
```
0.9997721807366157
```

In [98]:
```
X_train3.head()
```

Out[98]:

	so2	no2	rspm	spm
360294	4.000000	24.0	23.000000	126.729064
278096	5.800000	11.1	129.000000	262.000000
277941	2.900000	26.3	72.000000	140.000000
269637	18.300000	28.6	83.619824	155.000000
272765	5.275874	12.7	80.000000	176.000000

In [99]:
```
model.predict([[2.059,8.94,102,256]]) #correct
```

Out[99]:
```
array(['Unhealthy'], dtype=object)
```

Using Decision Tree Classifier

In [100]:
```
from sklearn import tree
```

In [101]:
```
model2 = tree.DecisionTreeClassifier()
```

In [102]:
```
model2.fit(X_train3,y_train3)
```

Out[102]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [103]:
```
model2.score(X_test3,y_test3) #high accuracy score of 99.98%
```

Out[103]:
```
0.9997215542336413
```

Some predictions

In [104]:
```
model2.predict([[9,31,51,205.25]]) # correct
```

Out[104]:
```
array(['Poor'], dtype=object)
```

In [105]:
```
model2.predict([[2,5.8,17,36]]) # correct
```

Out[105]:
```
array(['Good'], dtype=object)
```

In [106]:
```
model2.predict([[18.6,48.3,142,285]]) # correct
```

Out[106]:
```
array(['Unhealthy'], dtype=object)
```

In [107]:
```
model2.predict([[6,11,109,84.41]]) # correct
```

Out[107]:
```
array(['Poor'], dtype=object)
```

In [108]:
```
model2.predict([[10,16,156,372.66]]) # correct
```

Out[108]:
```
array(['Very unhealthy'], dtype=object)
```

Generating Profile Report

In [112]:
```
#import the another library
from ydata_profiling import ProfileReport
```

In [113]:
```
prof = ProfileReport(data)
prof.to_file(output_file = 'output.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Conclusion

- AQI is highly correlated with all the independent variables(so2, no2, rspm, spm)
- AQI has been increasing over the years.

Best models for AQI range classification :

1. Random Forest Classifier
2. Decision Tree Classifier

THE END

In [ ]:
```
```