In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [2]:
```
df=pd.read_csv('C:\\Users\\HP\\Downloads\\data.csv.zip',encoding='unicode_escape')
```

In [3]:
```
df.head()
```

Out[3]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_n
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	

In [4]:
```
df.shape
```

Out[4]:
```
(435742, 13)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435742 entries, 0 to 435741
Data columns (total 13 columns):
```

```
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   stn_code                     291665 non-null  object
 1   sampling_date                435739 non-null  object
 2   state                        435742 non-null  object
 3   location                     435739 non-null  object
 4   agency                       286261 non-null  object
 5   type                         430349 non-null  object
 6   so2                          401096 non-null  float64
 7   no2                          419509 non-null  float64
 8   rspm                         395520 non-null  float64
 9   spm                          198355 non-null  float64
 10  location_monitoring_station  408251 non-null  object
 11  pm2_5                        9314 non-null    float64
 12  date                         435735 non-null  object
dtypes: float64(5), object(8)
memory usage: 43.2+ MB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

In [7]:
```
df.describe()
```

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

In [8]:
```
df.nunique()
```

Out[8]:
```
stn_code                     803
sampling_date               5485
```

```
state                          37
location                      304
agency                         64
type                           10
so2                          4197
no2                          6864
rspm                         6065
spm                          6668
location_monitoring_station   991
pm2_5                         433
date                         5067
dtype: int64
```

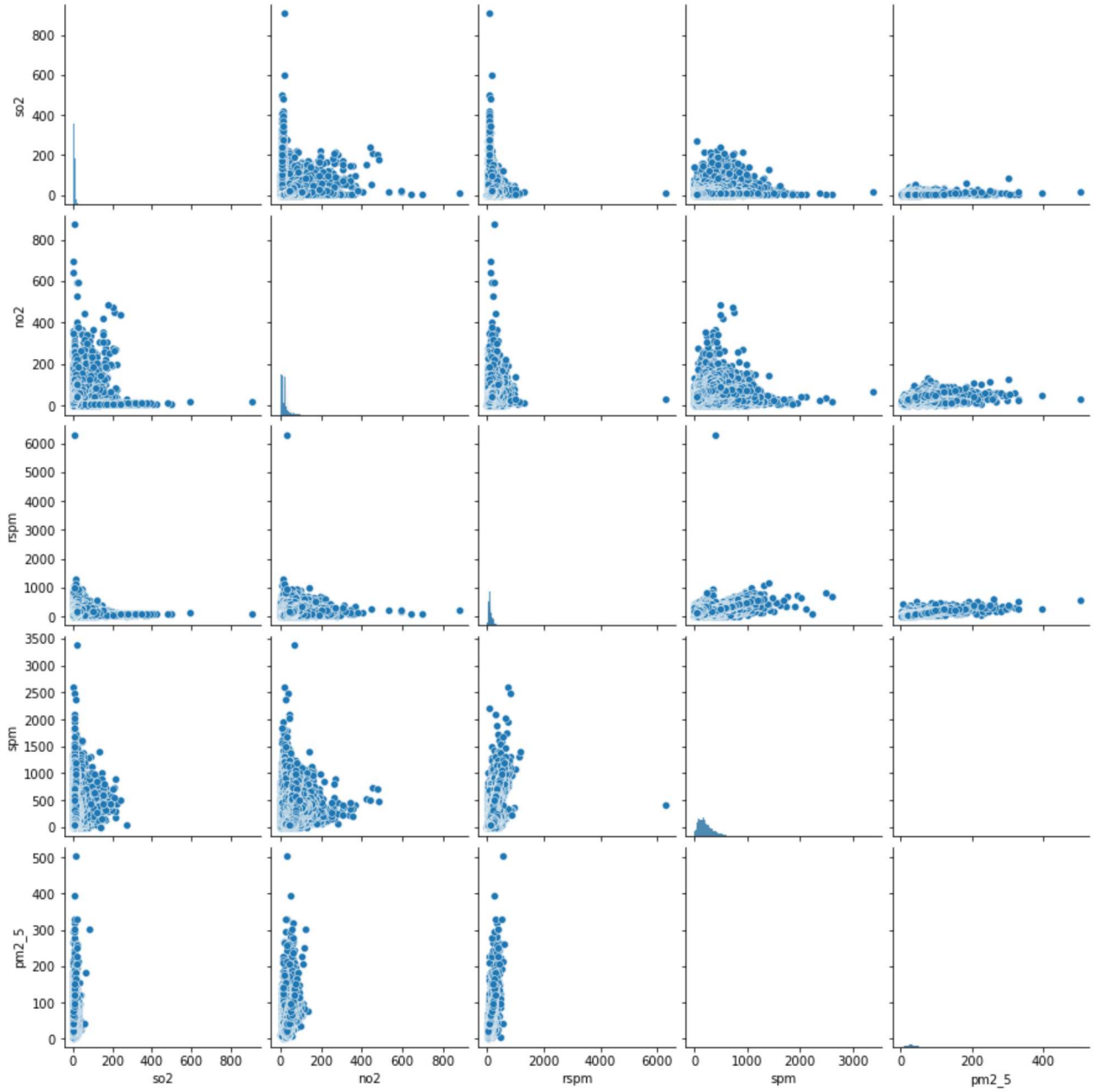
In [9]:
```
df.columns
```

Out[9]:
```
Index(['stn_code', 'sampling_date', 'state', 'location', 'agency', 'type',
       'so2', 'no2', 'rspm', 'spm', 'location_monitoring_station', 'pm2_5',
       'date'],
      dtype='object')
```

Data Visualization

In [10]:
```
sns.pairplot(data=df)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x216e60096d0>
```



In [11]:
```
df['state'].value_counts()
```

Out[11]:

Maharashtra	60384
Uttar Pradesh	42816
Andhra Pradesh	26368
Punjab	25634
Rajasthan	25589
Kerala	24728
Himachal Pradesh	22896
West Bengal	22463
Gujarat	21279
Tamil Nadu	20597
Madhya Pradesh	19920
Assam	19361
Odisha	19279
Karnataka	17119
Delhi	8551
Chandigarh	8520
Chhattisgarh	7831

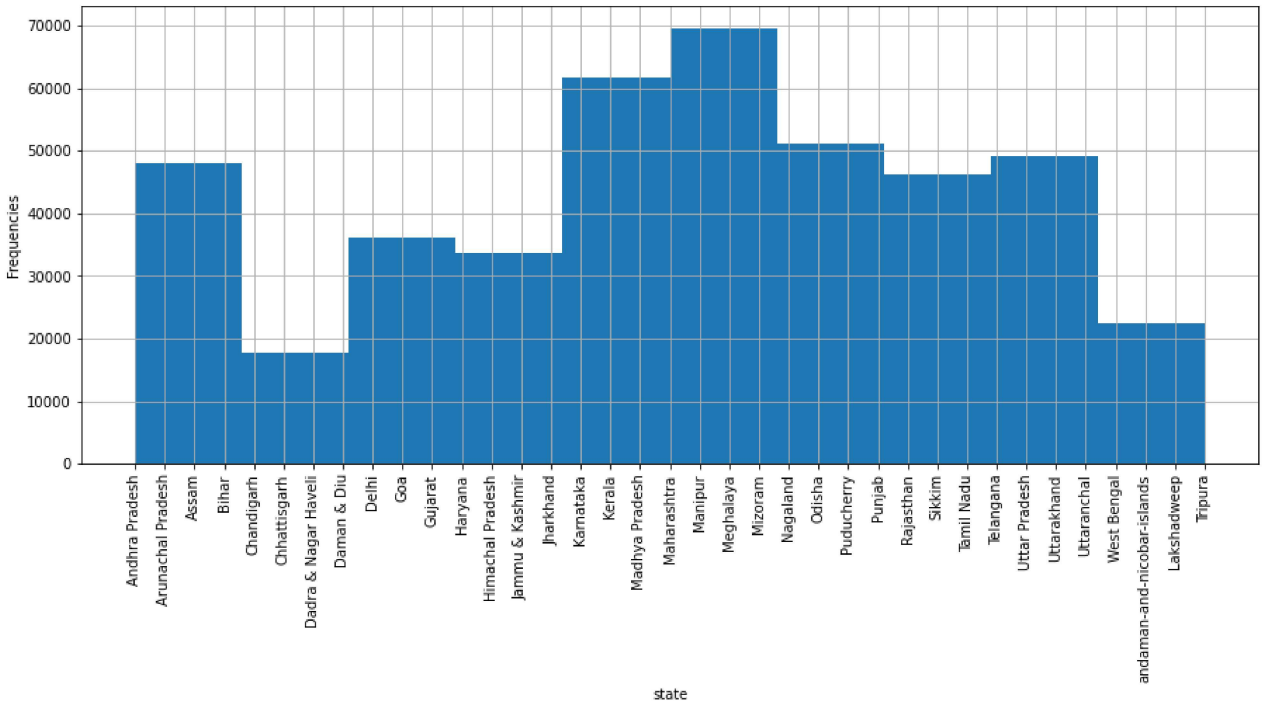
```
Goa                              6206
Jharkhand                        5968
Mizoram                          5338
Telangana                        3978
Meghalaya                        3853
Puducherry                       3785
Haryana                          3420
Nagaland                         2463
Bihar                            2275
Uttarakhand                      1961
Jammu & Kashmir                  1289
Daman & Diu                       782
Dadra & Nagar Haveli              634
Uttaranchal                       285
Arunachal Pradesh                  90
Manipur                            76
Sikkim                              1
andaman-and-nicobar-islands         1
Lakshadweep                         1
Tripura                             1
Name: state, dtype: int64
```

In [12]:

```
plt.figure(figsize=(15,6))
plt.xticks(rotation=90)
df.state.hist()
plt.xlabel('state')
plt.ylabel('Frequencies')
plt.plot()
```

Out[12]:

```
[]
```



In [13]:

```
df['type'].value_counts()
```

Out[13]:

```
Residential, Rural and other Areas    179014
Industrial Area                        96091
```

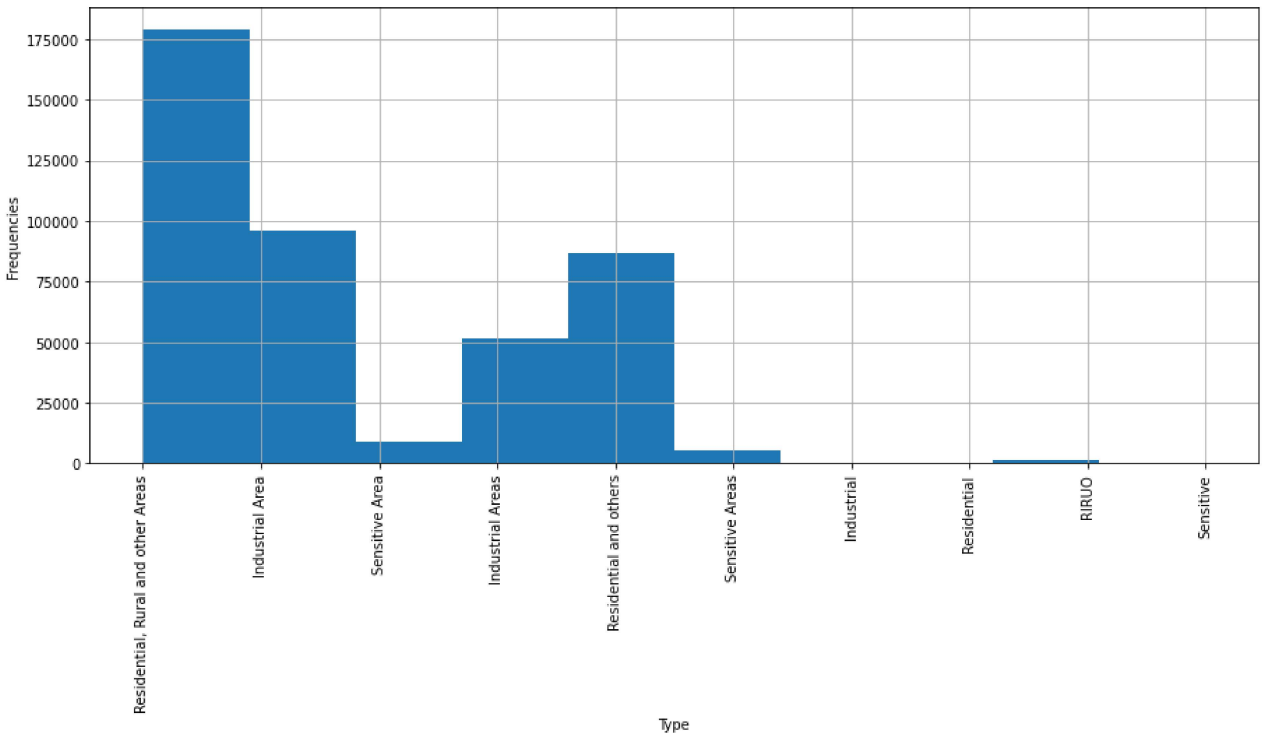
```
Residential and others          86791
Industrial Areas                51747
Sensitive Area                   8980
Sensitive Areas                  5536
RIRUO                            1304
Sensitive                         495
Industrial                        233
Residential                       158
Name: type, dtype: int64
```

In [14]:

```
plt.figure(figsize=(15,6))
plt.xticks(rotation=90)
df.type.hist()
plt.xlabel('Type')
plt.ylabel('Frequencies')
plt.plot()
```

Out[14]:

```
[]
```



In [15]:

```
df['agency'].value_counts()
```

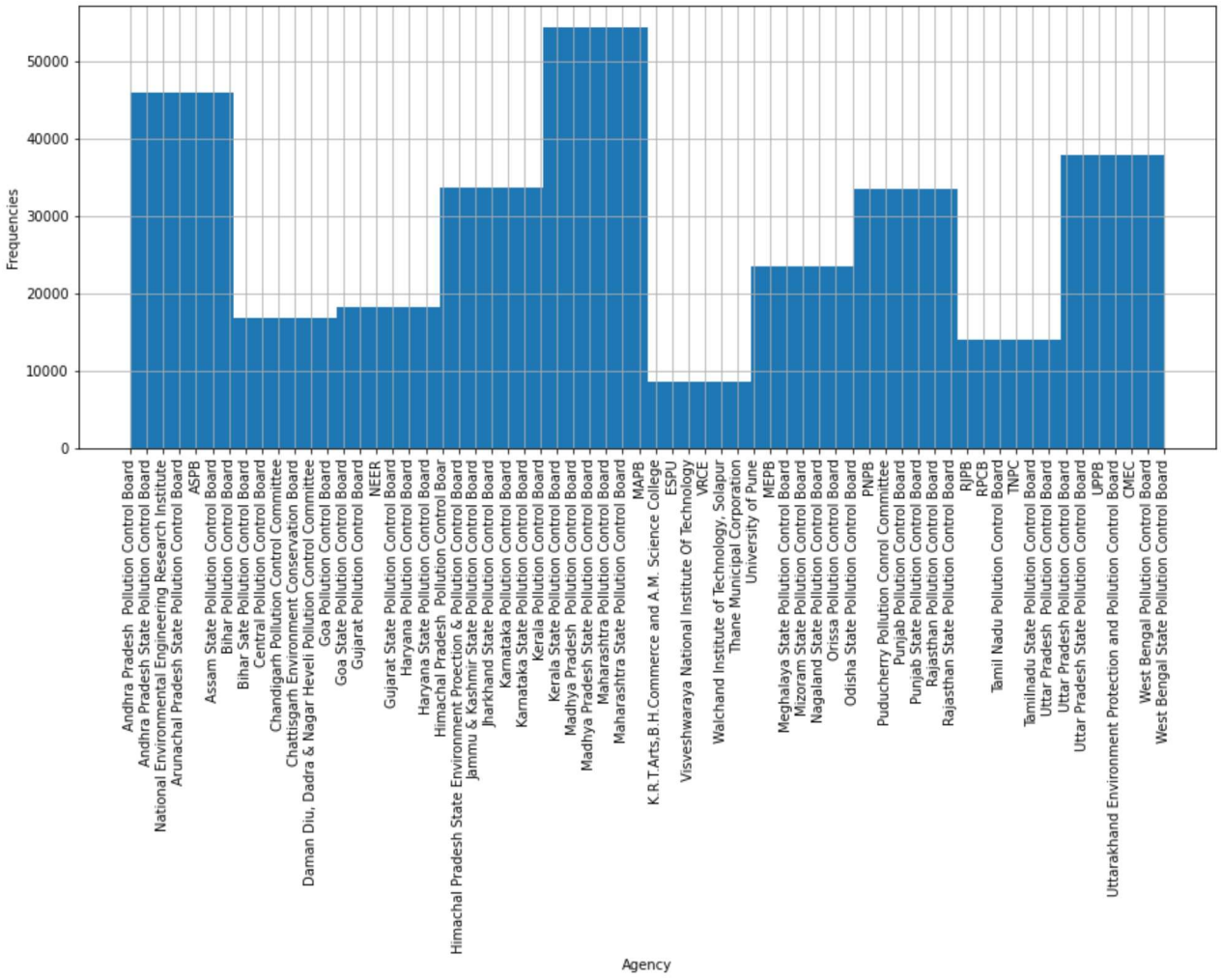
Out[15]:

```
Maharashtra State Pollution Control Board                          27857
Uttar Pradesh State Pollution Control Board                        22686
Andhra Pradesh State Pollution Control Board                       19139
Himachal Pradesh State Environment Proection & Pollution Control Board    15287
Punjab State Pollution Control Board                               15232
                                                                     ...
Arunachal Pradesh State Pollution Control Board                       90
TNPC                                                                  82
RPCB                                                                  63
VRCE                                                                  61
RJPB                                                                  53
Name: agency, Length: 64, dtype: int64
```

In [16]:

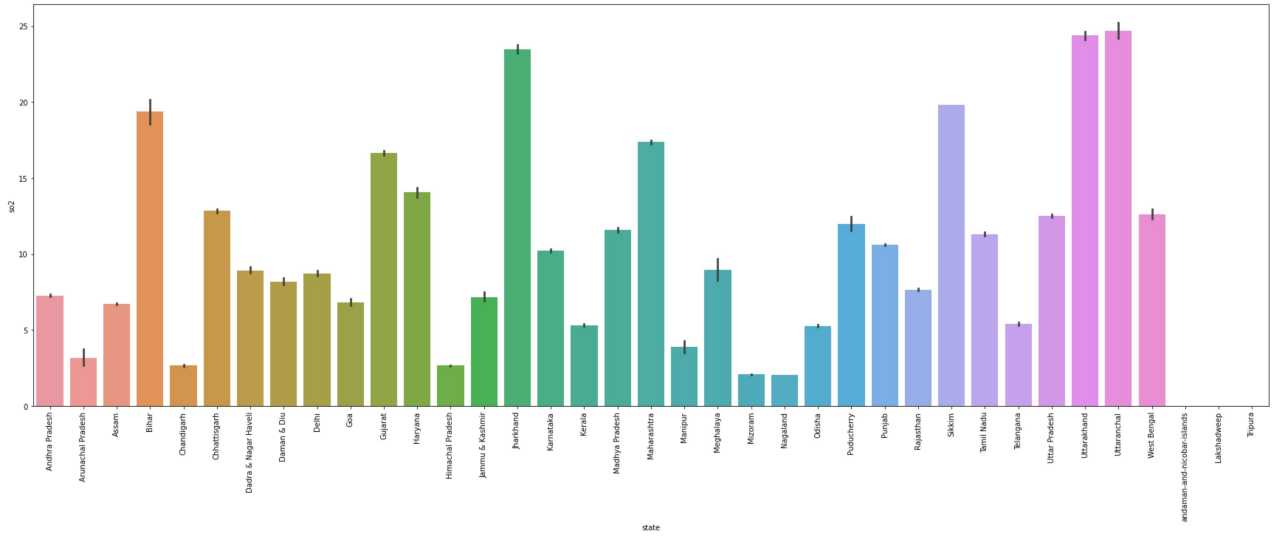
```
plt.figure(figsize=(15,6))
plt.xticks(rotation=90)
df.agency.hist()
plt.xlabel('Agency')
plt.ylabel('Frequencies')
plt.plot()
```

Out[16]: []



In [17]:
```
plt.figure(figsize=(30,10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='so2',data=df)
```

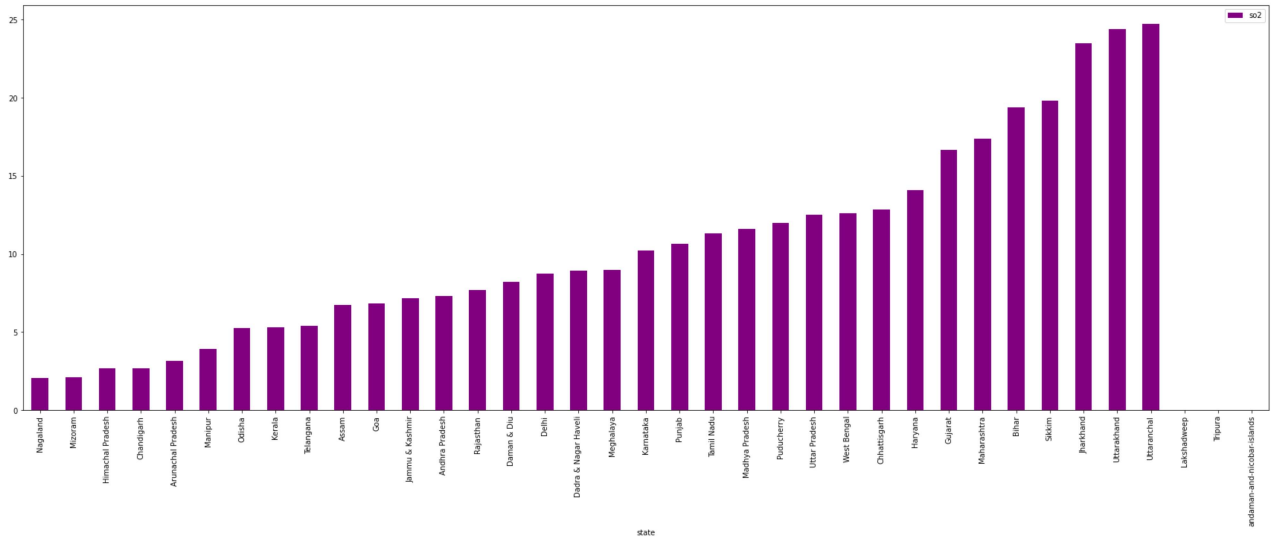
Out[17]: <AxesSubplot:xlabel='state', ylabel='so2'>



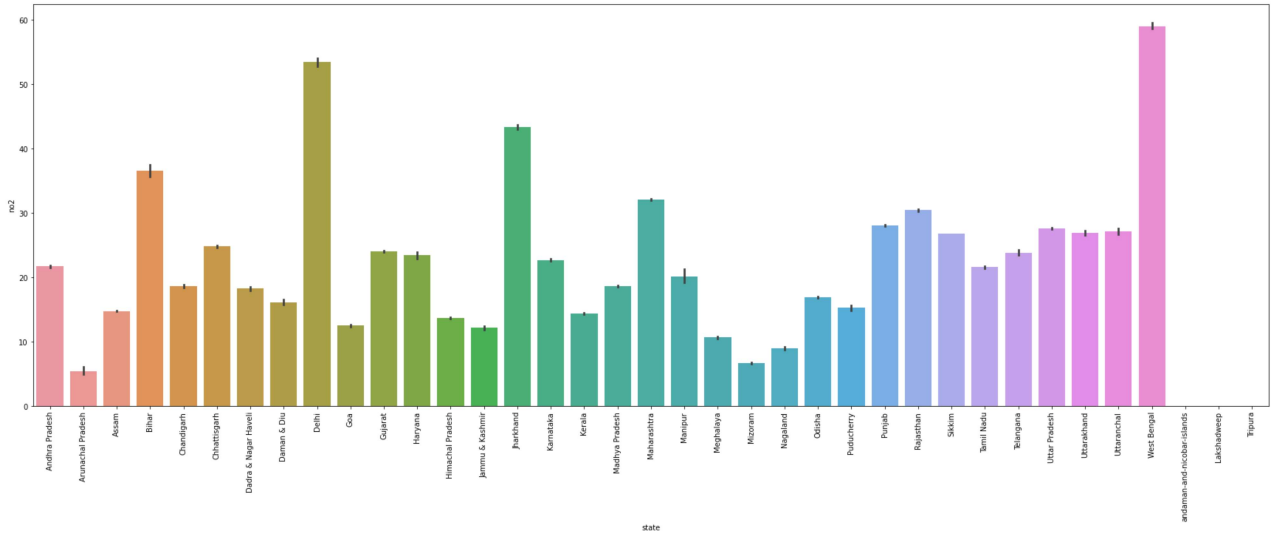
In [18]:
```
plt.rcParams['figure.figsize']=(30,10)
```

In [19]:
```
df[['so2','state']].groupby(["state"]).mean().sort_values(by='so2').plot.bar(color='pur
```

Out[19]: <AxesSubplot:xlabel='state'>

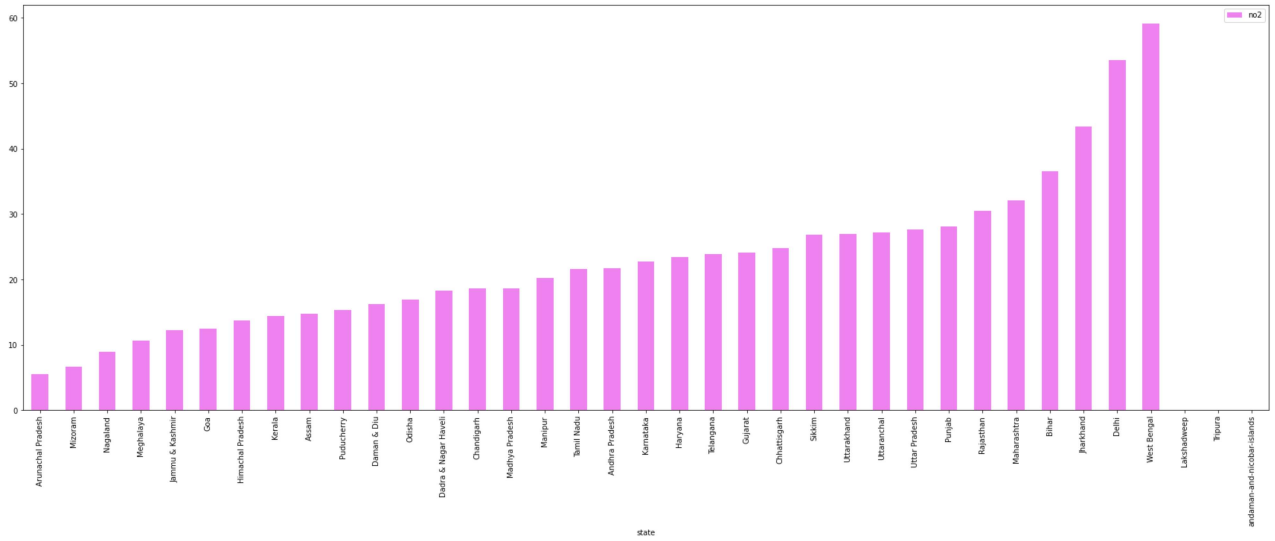


In [20]:
```
plt.figure(figsize=(30,10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='no2',data=df);
```

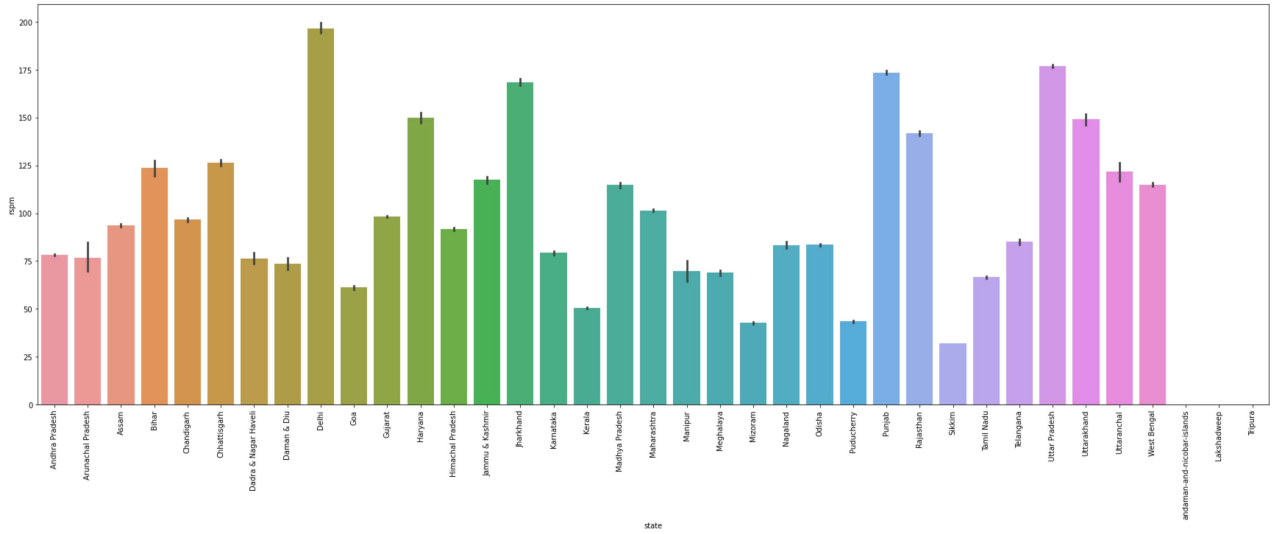


In [21]:
```
df[['no2','state']].groupby(["state"]).mean().sort_values(by='no2').plot.bar(color='vio
```

Out[21]: <AxesSubplot:xlabel='state'>

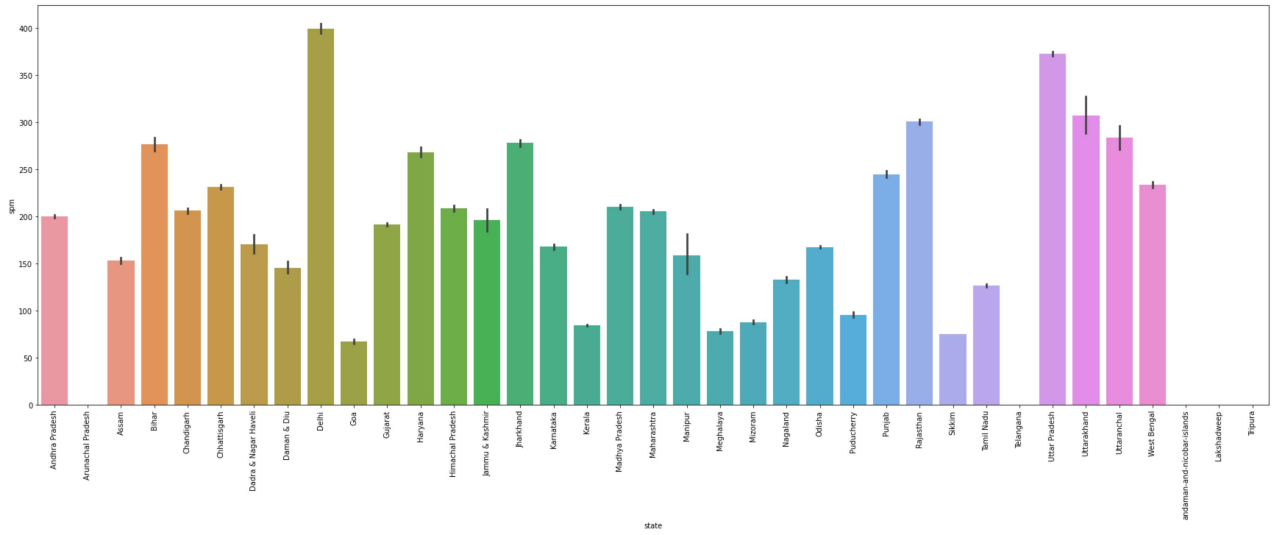


In [22]:
```
plt.figure(figsize=(30,10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='rspm',data=df);
```



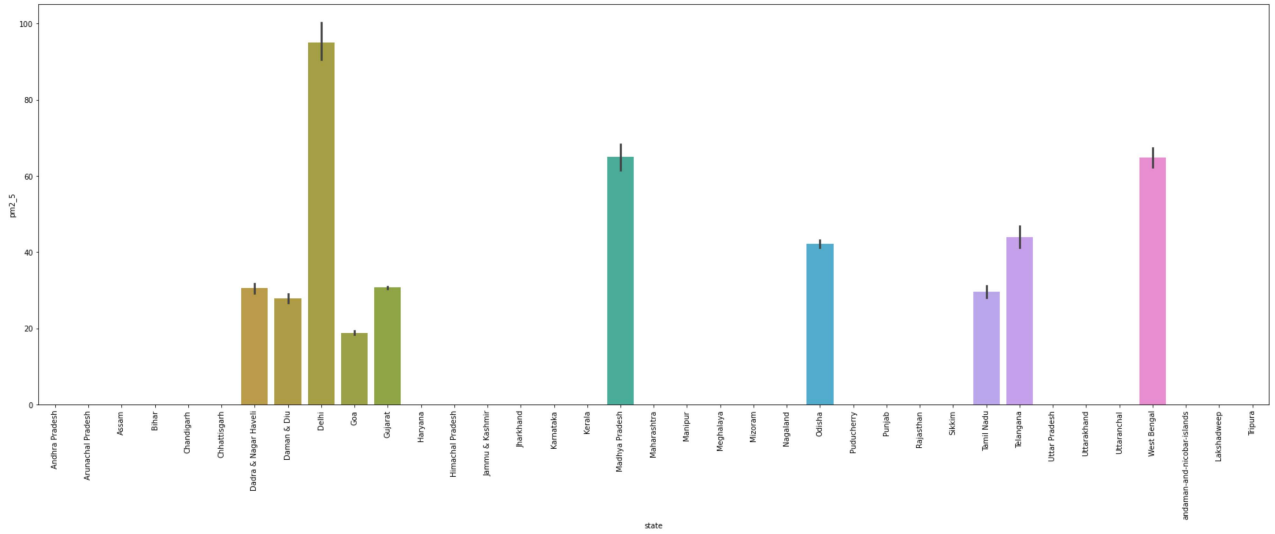
In [23]:

```
plt.figure(figsize=(30,10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='spm',data=df);
```



In [24]:

```
plt.figure(figsize=(30,10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='pm2_5',data=df);
```



In [25]:
```
nullvalues = df.isnull().sum().sort_values(ascending=False)
```

In [26]:
```
nullvalues
```

Out[26]:
```
pm2_5                          426428
spm                            237387
agency                         149481
stn_code                       144077
rspm                            40222
so2                             34646
location_monitoring_station     27491
no2                             16233
type                             5393
date                                7
sampling_date                       3
location                            3
state                               0
dtype: int64
```

In [27]:
```
null_values_percentage = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascend
```

In [28]:
```
missing_data_with_percentage = pd.concat([nullvalues,null_values_percentage], axis=1,ke
```

In [29]:
```
missing_data_with_percentage
```

Out[29]:

	total	Percent
pm2_5	426428	97.862497
spm	237387	54.478797
agency	149481	34.304933
stn_code	144077	33.064749
rspm	40222	9.230692
so2	34646	7.951035

	total	Percent
location_monitoring_station	27491	6.309009
no2	16233	3.725370
type	5393	1.237659
date	7	0.001606
sampling_date	3	0.000688
location	3	0.000688
state	0	0.000000

In [30]:
```
df.drop(['agency'],axis=1,inplace=True)
df.drop(['stn_code'],axis=1,inplace=True)
df.drop(['date'],axis=1,inplace=True)
df.drop(['sampling_date'],axis=1,inplace=True)
df.drop(['location_monitoring_station'],axis=1,inplace=True)
```

In [31]:
```
df.isnull().sum()
```

Out[31]:
```
state            0
location         3
type          5393
so2          34646
no2          16233
rspm         40222
spm         237387
pm2_5       426428
dtype: int64
```

In [32]:
```
df
```

Out[32]:

	state	location	type	so2	no2	rspm	spm	pm2_5
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	NaN	NaN	NaN
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
435737	West Bengal	ULUBERIA	RIRUO	22.0	50.0	143.0	NaN	NaN
435738	West Bengal	ULUBERIA	RIRUO	20.0	46.0	171.0	NaN	NaN
435739	andaman-and-nicobar-islands	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	state	location	type	so2	no2	rspm	spm	pm2_5
435740	Lakshadweep	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435741	Tripura	NaN	NaN	NaN	NaN	NaN	NaN	NaN

435742 rows × 8 columns

In [33]:
```
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])
```

In [34]:
```
df.fillna(0,inplace=True)
```

In [35]:
```
df.isnull().sum()
```

Out[35]:
```
state       0
location    0
type        0
so2         0
no2         0
rspm        0
spm         0
pm2_5       0
dtype: int64
```

In [36]:
```
df
```

Out[36]:

	state	location	type	so2	no2	rspm	spm	pm2_5
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0.0
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	0.0	0.0	0.0
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0.0
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	0.0	0.0	0.0
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
435737	West Bengal	ULUBERIA	RIRUO	22.0	50.0	143.0	0.0	0.0
435738	West Bengal	ULUBERIA	RIRUO	20.0	46.0	171.0	0.0	0.0
435739	andaman-and-nicobar-islands	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0
435740	Lakshadweep	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0
435741	Tripura	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0

435742 rows × 8 columns

CALCULATE AIR QUALITY INDEX

In [37]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Function to calculate s02 individual pollutant index(si)

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
df['SOi']=df['so2'].apply(cal_SOi)
data= df[['so2','SOi']]
data.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

Function to calculate no2 individual pollutant index(ni)

In [41]:
```
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
df['Noi']=df['no2'].apply(cal_Noi)
data = df[['no2','Noi']]
data.head()
```

Out[41]:

	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

Function to calculate rspm individual pollutant index(rpi)

In [42]:

```
def cal_RSPMI(rspm):
    rpi=0
    if(rpi<=30):
        rpi=rpi*50/30
    elif (rpi>30 and rpi<=60):
        rpi=50+(rpi-30)*50/30
    elif (rpi>60 and rpi<=90):
        rpi=100+(rpi-60)*100/30
    elif (rpi>90 and rpi<=120):
        rpi=200+(rpi-90)*100/30
    elif (rpi>120 and rpi<=250):
        rpi=300+(rpi-120)*(100/130)
    else:
        rpi=400+(rpi-250)*(100/130)
    return rpi
df['Rpi']=df['rspm'].apply(cal_RSPMI)
data= df[['rspm','Rpi']]
data.head()
```

Out[42]:

	rspm	Rpi
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

In [43]:

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

Out[43]:

	spm	SPMi
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

Function to calculate the air quality index (AQI) of every data value

In [44]:

```
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
df['AQI']=df.apply(lambda x:cal_aqi(x['SOi'],x['Noi'],x['Rpi'],x['SPMi']),axis=1)
data= df[['state','SOi','Noi','Rpi','SPMi','AQI']]
data.head()
```

Out[44]:

	state	SOi	Noi	Rpi	SPMi	AQI
0	Andhra Pradesh	6.000	21.750	0.0	0.0	21.750
1	Andhra Pradesh	3.875	8.750	0.0	0.0	8.750
2	Andhra Pradesh	7.750	35.625	0.0	0.0	35.625
3	Andhra Pradesh	7.875	18.375	0.0	0.0	18.375
4	Andhra Pradesh	5.875	9.375	0.0	0.0	9.375

In [45]:

```
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"

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

Out[45]:

	state	location	type	so2	no2	rspm	spm	pm2_5	SOi	Noi	Rpi	SPMi	AQI	A
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0.0	6.000	21.750	0.0	0.0	21.750	
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	0.0	0.0	0.0	3.875	8.750	0.0	0.0	8.750	
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0.0	7.750	35.625	0.0	0.0	35.625	
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	0.0	0.0	0.0	7.875	18.375	0.0	0.0	18.375	
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	0.0	0.0	0.0	5.875	9.375	0.0	0.0	9.375	

In [46]:

```
df['AQI_Range'].value_counts()
```

Out[46]:

```
Good              219643
Poor               93272
Moderate           56571
Unhealthy          31733
Hazardous          18700
Very unhealthy     15823
Name: AQI_Range, dtype: int64
```

Splitting the dataset into Dependent and Independent columns

In [48]:

```
X=df[['SOi','Noi','Rpi','SPMi']]
Y=df['AQI']
```

```
X.head()
```

Out[48]:

	SOi	Noi	Rpi	SPMi
0	6.000	21.750	0.0	0.0
1	3.875	8.750	0.0	0.0
2	7.750	35.625	0.0	0.0
3	7.875	18.375	0.0	0.0
4	5.875	9.375	0.0	0.0

In [49]:
```
Y.head()
```

Out[49]:
```
0    21.750
1     8.750
2    35.625
3    18.375
4     9.375
Name: AQI, dtype: float64
```

In [51]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=70)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

```
(348593, 4) (87149, 4) (348593,) (87149,)
```

Linear Regression

In [52]:
```
model=LinearRegression()
model.fit(X_train,Y_train)
```

Out[52]:
```
LinearRegression()
```

In [53]:
```
train_pred=model.predict(X_train)
test_pred=model.predict(X_test)
```

In [54]:
```
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_pred)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_pred)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',model.score(X_train,Y_train))
print('RSquared value on test:',model.score(X_test,Y_test))
```

```
RMSE TrainingData =  13.583424938613533
RMSE TestData =  13.672937344789007
--------------------------------------------------
RSquared value on train: 0.9849533579250526
RSquared value on test: 0.9847286394495923
```

Decision Tree Regressor

In [56]:
```
DT=DecisionTreeRegressor()
DT.fit(X_train,Y_train)
```

Out[56]:
```
DecisionTreeRegressor()
```

In [58]:
```
train_preds=DT.predict(X_train)
test_preds=DT.predict(X_test)
```

In [59]:
```
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',DT.score(X_train,Y_train))
print('RSquared value on test:',DT.score(X_test,Y_test))
```

```
RMSE TrainingData =  2.2360361352585847e-13
RMSE TestData =  1.2989914612260256
--------------------------------------------------
RSquared value on train: 1.0
RSquared value on test: 0.9998621627258399
```

Random Forest Regressor

In [60]:
```
RF=RandomForestRegressor().fit(X_train,Y_train)
```

In [61]:
```
train_preds1=RF.predict(X_train)
test_preds1=RF.predict(X_test)
```

In [62]:
```
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_preds1)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_preds1)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
print('RSquared value on train:',RF.score(X_train,Y_train))
print('RSquared value on test:',RF.score(X_test,Y_test))
```

```
RMSE TrainingData =  0.4262778965571081
RMSE TestData =  1.160631302015957
--------------------------------------------------
RSquared value on train: 0.9999851814181251
RSquared value on test: 0.99988996201175
```

Classification Algorithms

In [63]:
```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [64]:
```
X2 = df[['SOi','Noi','Rpi','SPMi']]
Y2 = df['AQI_Range']
```

In [65]:
```
X_train2,X_test2,Y_train2,Y_test2=train_test_split(X2,Y2,test_size=0.33,random_state=70
```

Logistic Regression

In [69]:
```
log_reg = LogisticRegression().fit(X_train2, Y_train2)
train_preds2=log_reg.predict(X_train2)
print("Model accuracy on train is: ",accuracy_score(Y_train2,train_preds2))
test_preds2=log_reg.predict(X_test2)
print("Model accuracy on test is: ",accuracy_score(Y_test2,test_preds2))
print('-'*50)
print('KappaScore is: ',metrics.cohen_kappa_score(Y_test2,test_preds2))
```

```
Model accuracy on train is:  0.7276012426913104
Model accuracy on test is:  0.7271254216071491
--------------------------------------------------
KappaScore is:  0.584377382981496
```

In [70]:
```
log_reg.predict([[727,327.55,78.2,100]])
```

Out[70]:
```
array(['Good'], dtype=object)
```

In [71]:
```
log_reg.predict([[2.7,45,35.16,23]])
```

Out[71]:
```
array(['Poor'], dtype=object)
```

In [72]:
```
log_reg.predict([[10,2.8,82,20]])
```

Out[72]:
```
array(['Good'], dtype=object)
```

In [73]:
```
log_reg.predict([[2,45.8,37,32]])
```

Out[73]:
```
array(['Poor'], dtype=object)
```

Decision Tree Classifier

In [77]:
```
DT2=DecisionTreeClassifier().fit(X_train2,Y_train2)
train_preds3 = DT2.predict(X_train2)
print("Model accuracy on train is: ",accuracy_score(Y_train2,train_preds3))
test_preds3=DT2.predict(X_test2)
```

```
print("Model accuracy on test is: ",accuracy_score(Y_test2,test_preds3))
print('-'*50)
print('KappaScore is: ',metrics.cohen_kappa_score(Y_test2,test_preds3))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  0.9998052783476477
--------------------------------------------------
KappaScore is:  0.9997111966529943
```

Random Forest Classifier

In [78]:
```
RF=RandomForestClassifier().fit(X_train2,Y_train2)
train_preds4=RF.predict(X_train2)
print("Model accuracy on train is: ",accuracy_score(Y_train2,train_preds4))
test_preds4=RF.predict(X_test2)
print("Model accuracy on test is: ",accuracy_score(Y_test2,test_preds4))
print('-'*50)
print('KappaScore is: ',metrics.cohen_kappa_score(Y_test2,test_preds4))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  0.9998191870371014
--------------------------------------------------
KappaScore is:  0.999731826010858
```

K-Nearest Neighbours

In [79]:
```
KNN=KNeighborsClassifier().fit(X_train2,Y_train2)
train_preds5=KNN.predict(X_train2)
print("Model accuracy on train is: ",accuracy_score(Y_train2,train_preds5))
test_preds5=KNN.predict(X_test2)
print("Model accuracy on test is: ",accuracy_score(Y_test2,test_preds5))
print('-'*50)
print('KappaScore is: ',metrics.cohen_kappa_score(Y_test2,test_preds5))
```

```
Model accuracy on train is:  0.998153774486465
Model accuracy on test is:  0.9967105949441913
--------------------------------------------------
KappaScore is:  0.9951205476925056
```

In [80]:
```
KNN.predict([[7.4,47.7,78.182,100]])
```

Out[80]:
```
array(['Poor'], dtype=object)
```

In [81]:
```
KNN.predict([[1,1.2,3.12,0]])
```

Out[81]:
```
array(['Good'], dtype=object)
```

In [82]:
```
KNN.predict([[325.7,345,798.182,203]])
```

Out[82]:
```
array(['Unhealthy'], dtype=object)
```

In [ ]: