Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Name :Ayush Jain

SAP-ID : 60004200132

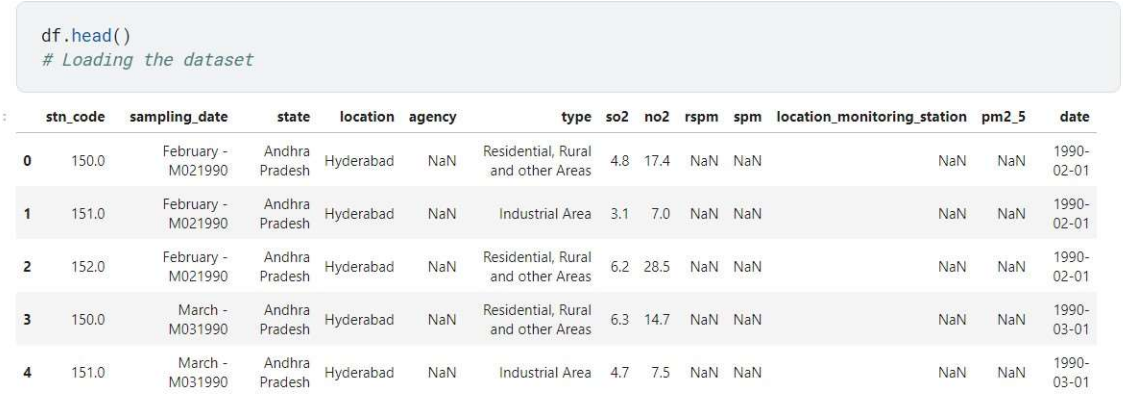
Branch : Computer Engineering

Machine Learning Minors – Mini Project Task 2

Problem Statement : Air Quality Prediction for Indian Cities

Data Understanding

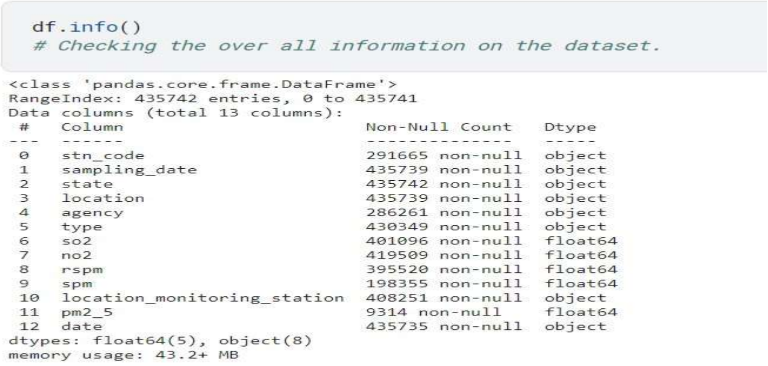
Firstly, we check the data :



```
df.head()
# Loading the dataset
```

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date
0	150.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	NaN	1990-02-01
1	151.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	3.1	7.0	NaN	NaN	NaN	NaN	1990-02-01
2	152.0	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	NaN	1990-02-01
3	150.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	NaN	1990-03-01
4	151.0	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.7	7.5	NaN	NaN	NaN	NaN	1990-03-01

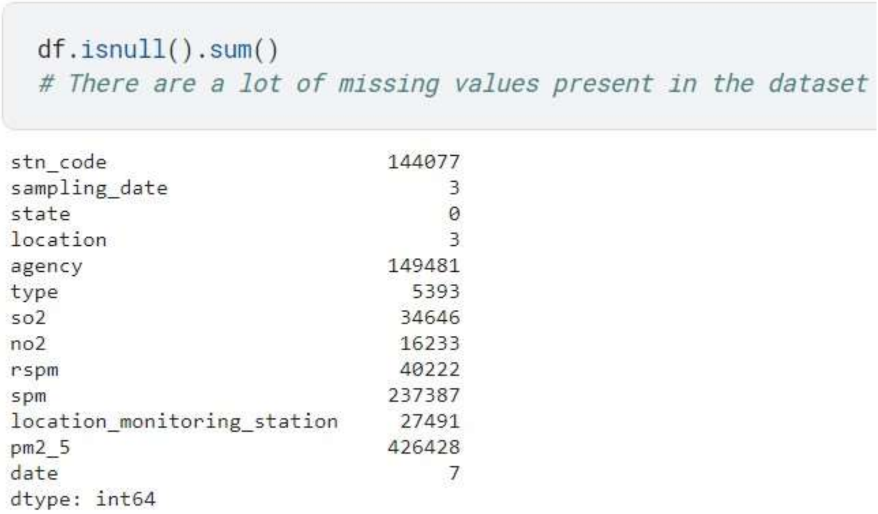
Next we check the info of all attributes,



```
df.info()
# Checking the over all information on the dataset.
```

```
<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
```

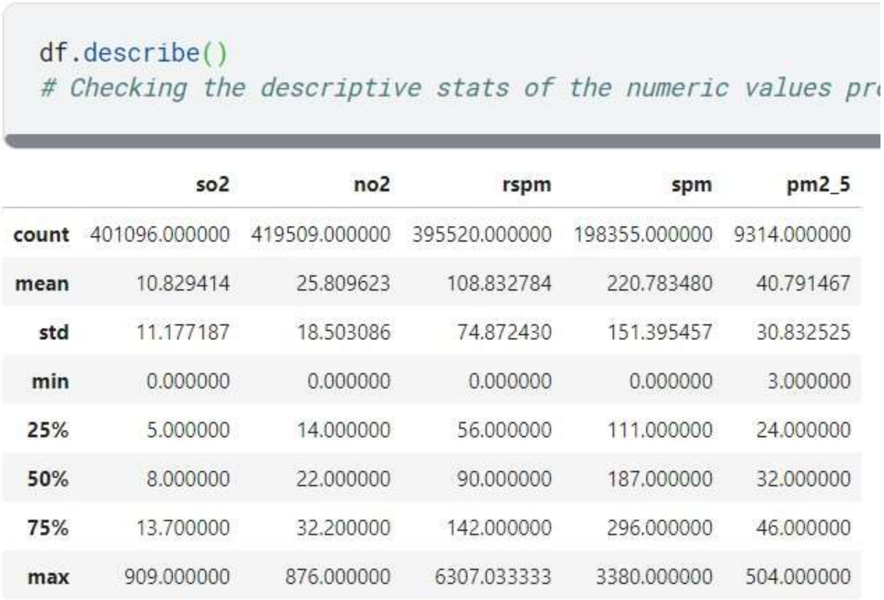
The shape of data is (435742, 13).
Now we check for null values,



```
df.isnull().sum()
# There are a lot of missing values present in the dataset
```

```
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
```

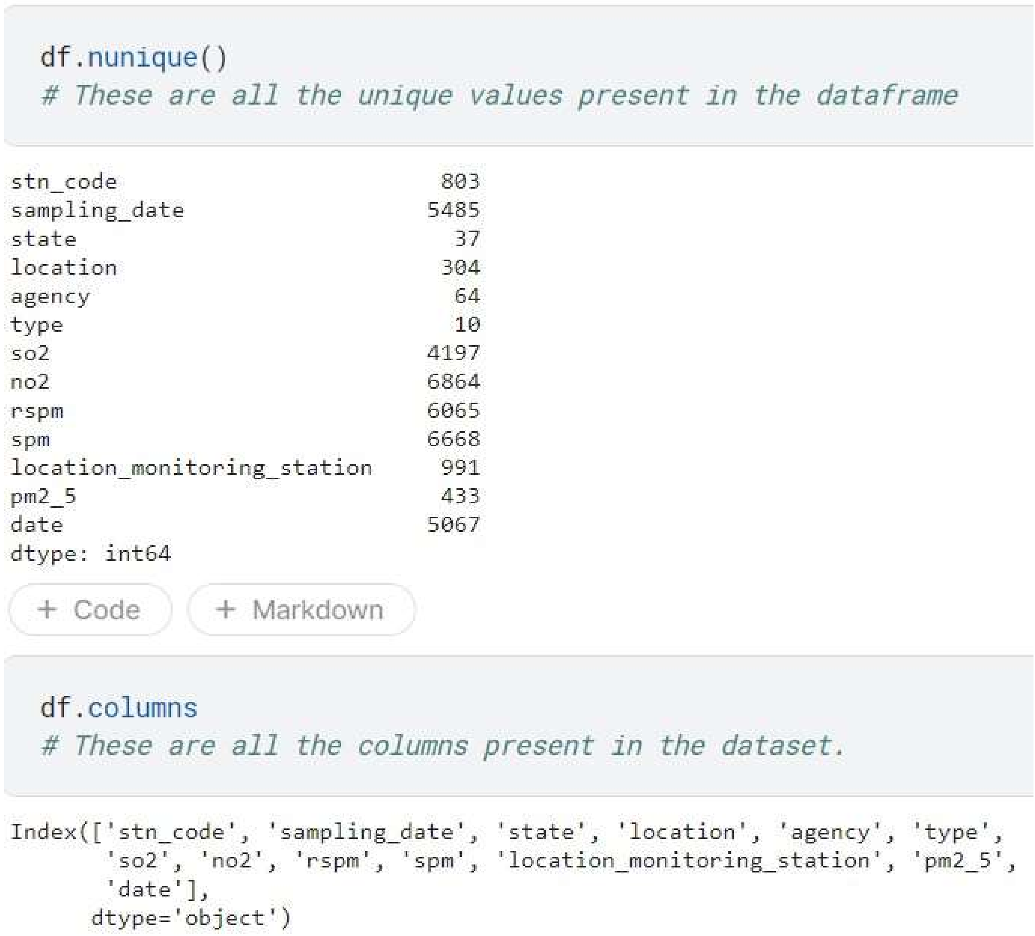
Next, Checking the descriptive stats of the numeric values present in the data like mean, standard deviation, min values and max value present in the data,



```
df.describe()
# Checking the descriptive stats of the numeric values pr
```

	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

Checking unique values,



```
df.nunique()
# These are all the unique values present in the dataframe
```

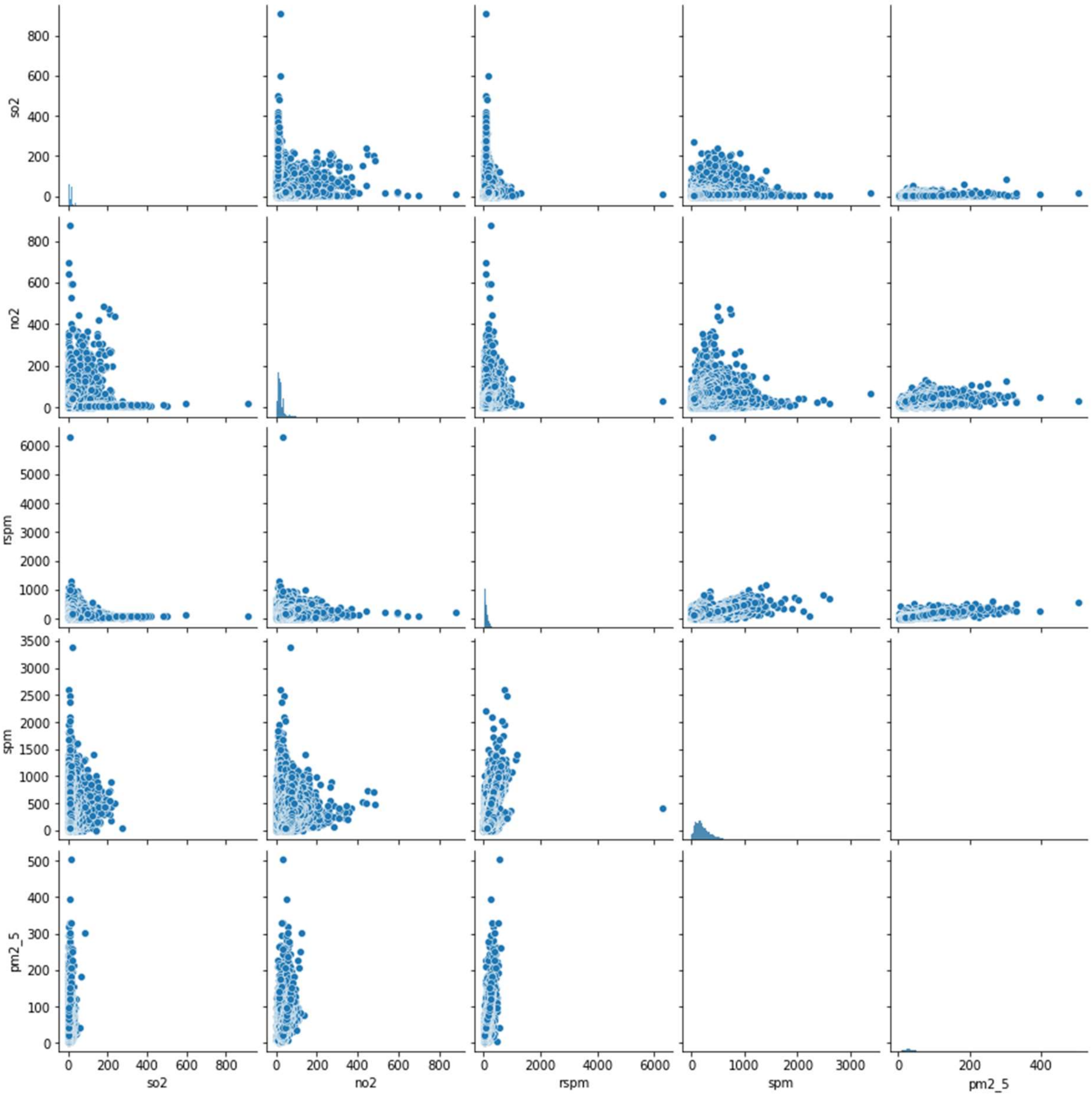
```
stn_code                        803
sampling_date                  5485
state                            37
location                        304
agency                           64
type                             10
so2                            4197
no2                            6864
rspm                           6065
spm                            6668
location_monitoring_station     991
pm2_5                           433
date                           5067
dtype: int64
```

```
df.columns
# These are all the columns present in the dataset.
```

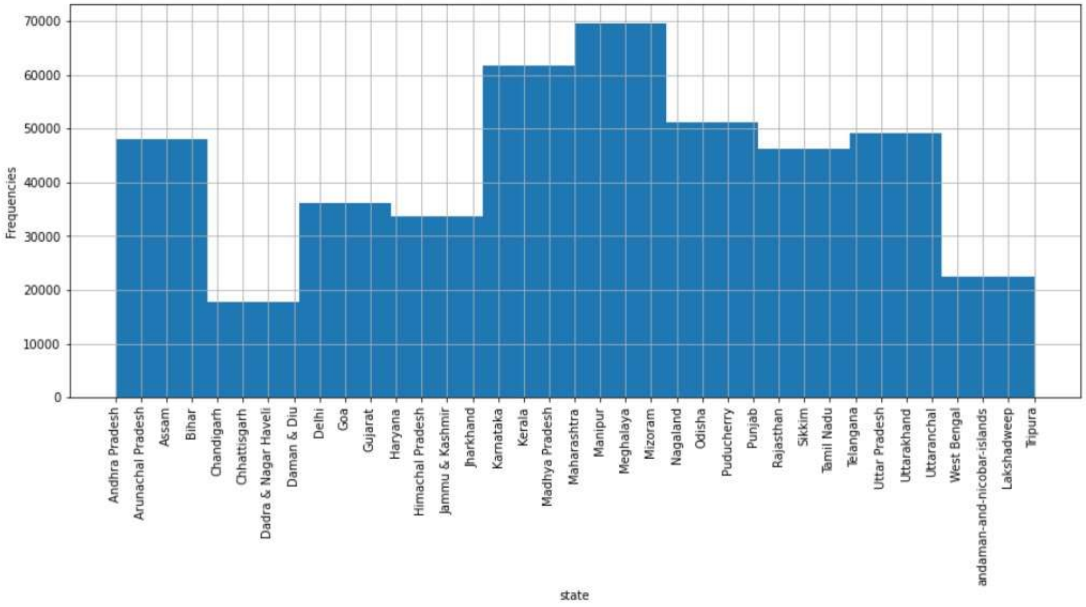
```
Index(['stn_code', 'sampling_date', 'state', 'location', 'agency', 'type',
       'so2', 'no2', 'rspm', 'spm', 'location_monitoring_station', 'pm2_5',
       'date'],
      dtype='object')
```

Data Visualization

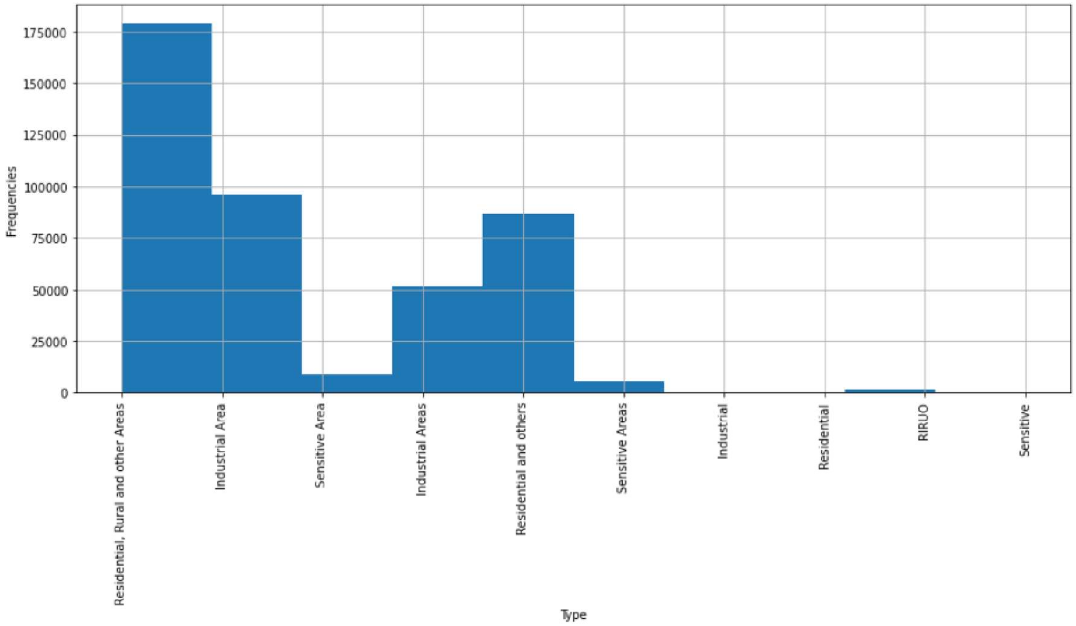
Displaying pair-plot between numeric attributes,



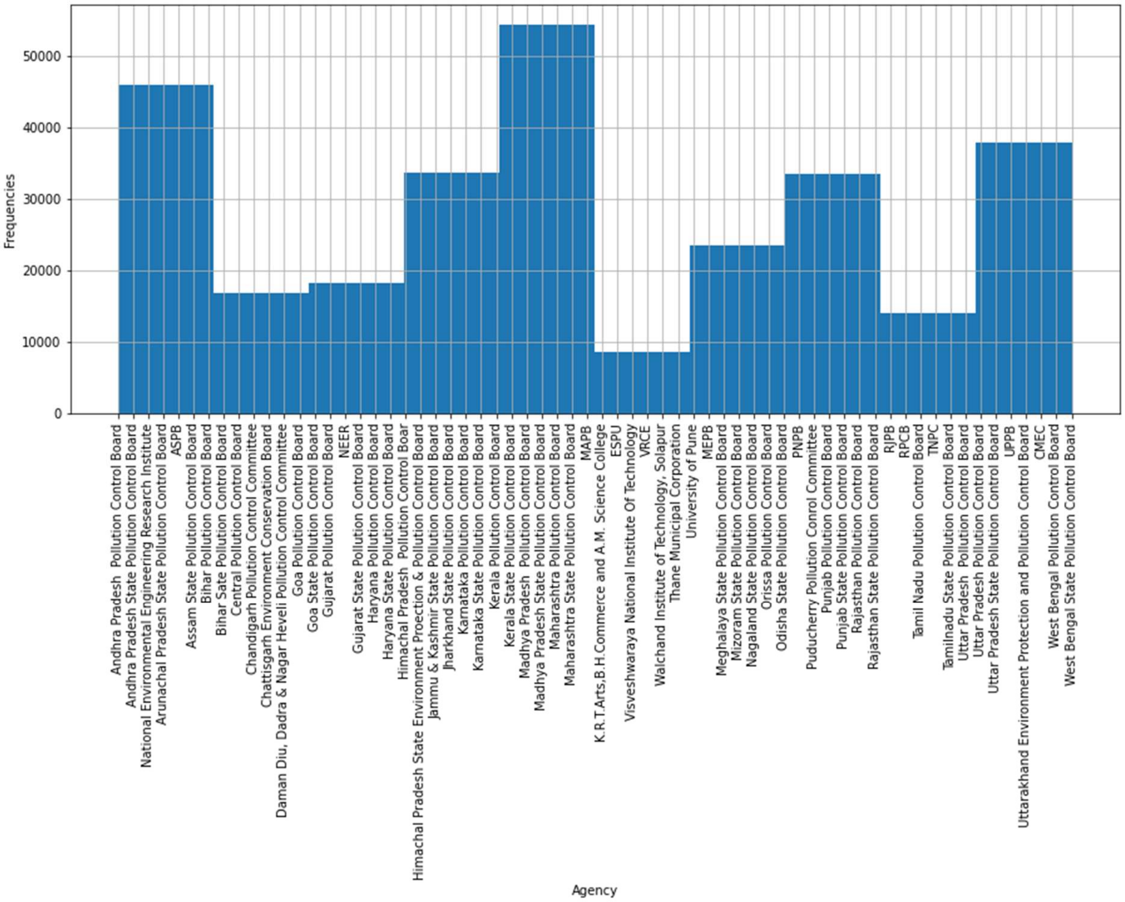
Viewing the count of values present in the state column,



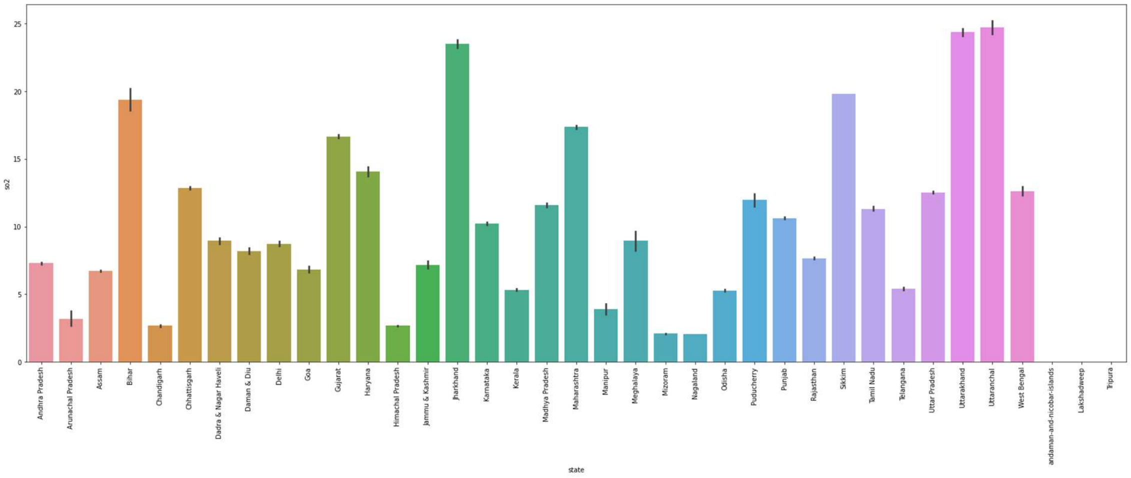
Viewing the count of values present in the type column,



Viewing the counts of values present in the agency column,

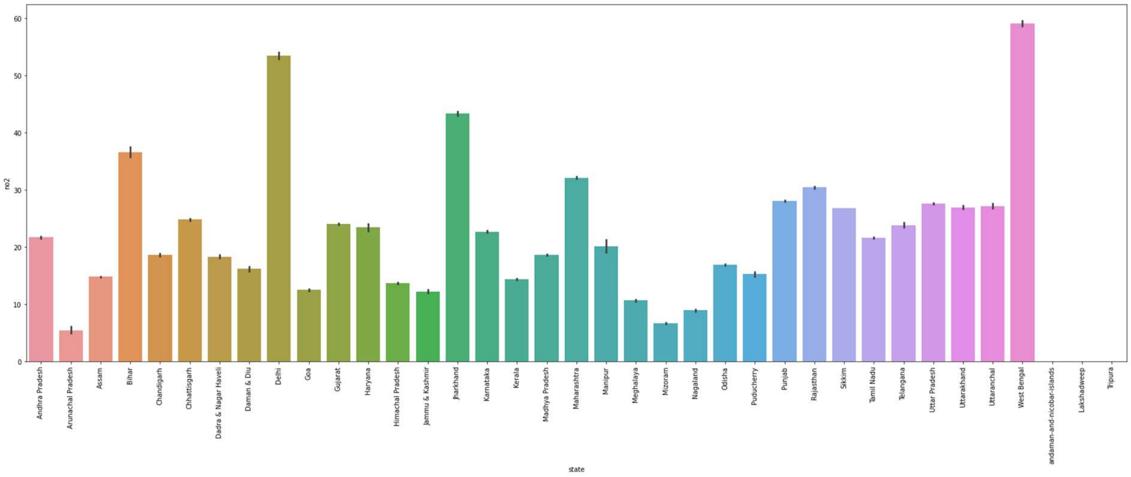


Visualizing the name of the state having higher so2 levels in the air,

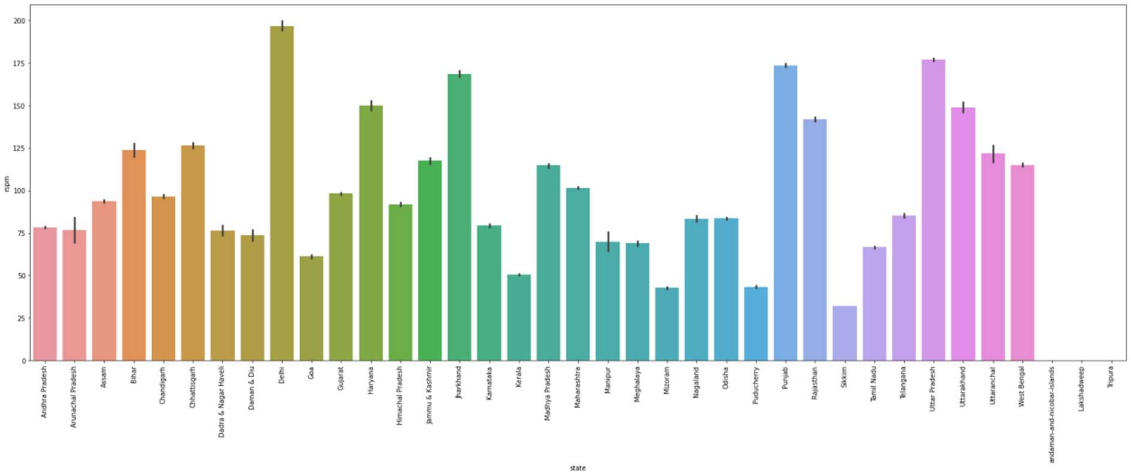


From the above visualization we can see that the state of Uttaranchal has highest so2 level fol lowed by Uttarakhand.

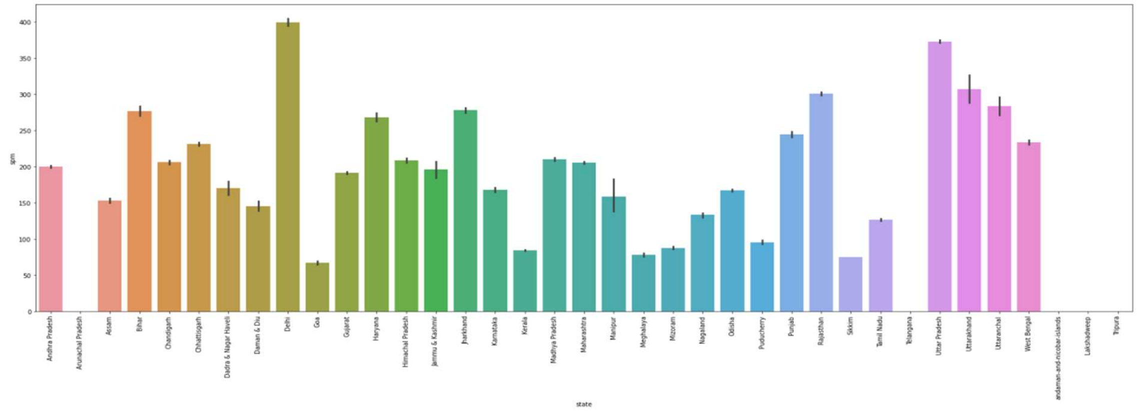
Similarly, for no2 level we get West Bengal has the highest no2 level,



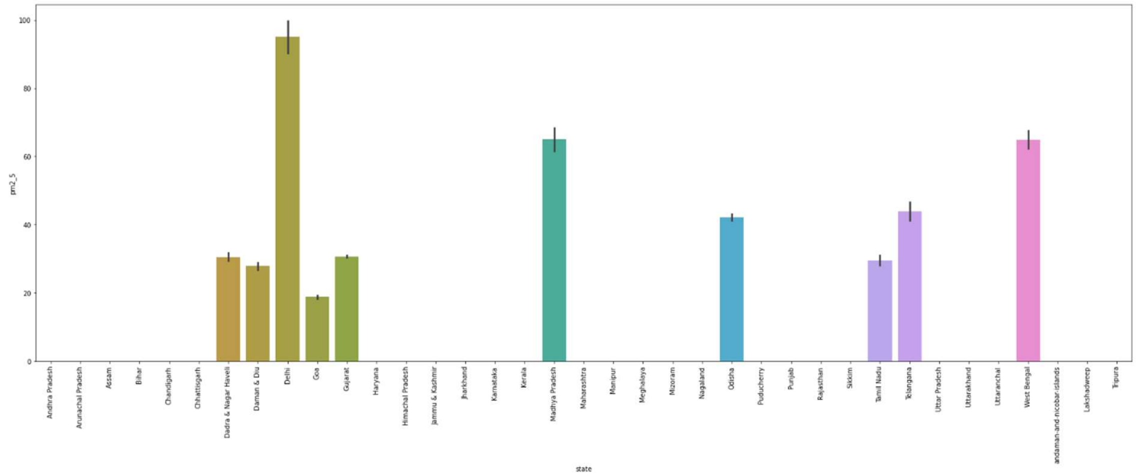
Delhi has higher rspm levels than other states,



Delhi has higher spm level compared to other states,

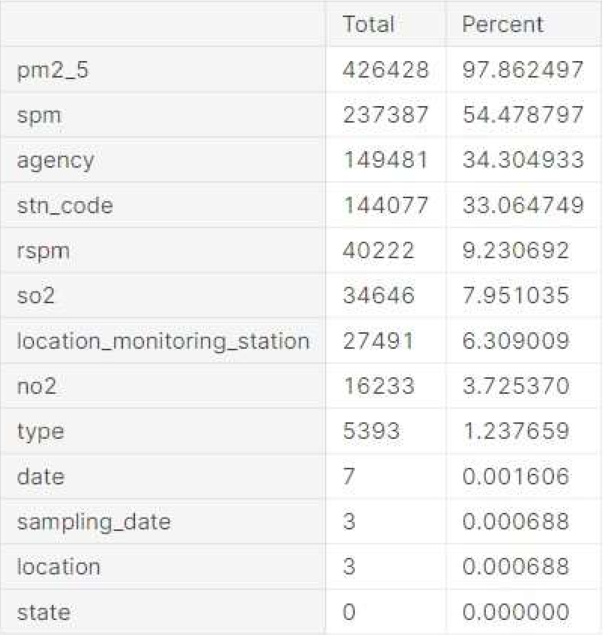


Delhi has higher pm2_5 level compared to other states,



Checking all null values and treating those null values

As you can see below these are the percentages of null values present in the dataset,

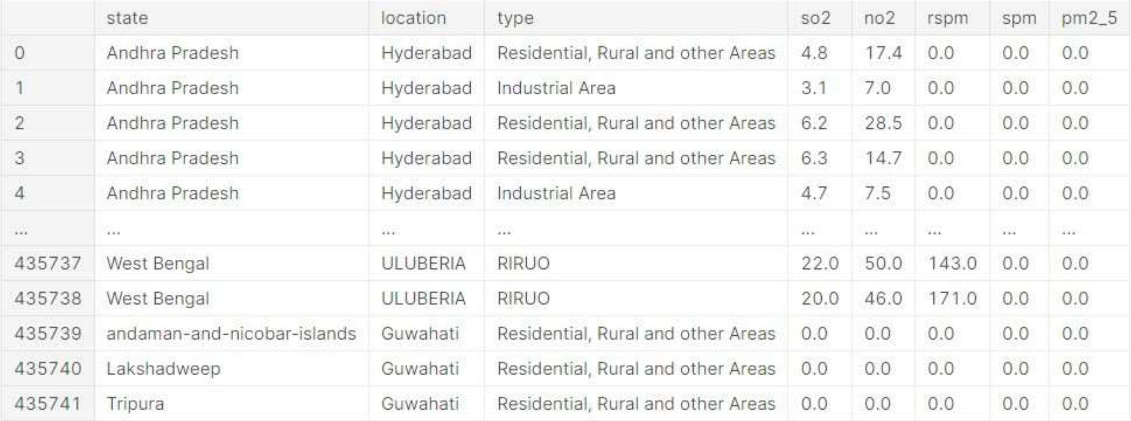


	Total	Percent
pm2_5	426428	97.862497
spm	237387	54.478797
agency	149481	34.304933
stn_code	144077	33.064749
rspm	40222	9.230692
so2	34646	7.951035
location_monitoring_station	27491	6.309009
no2	16233	3.725370
type	5393	1.237659
date	7	0.001606
sampling_date	3	0.000688
location	3	0.000688
state	0	0.000000

Next step,

```
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])
# Null value Imputation for categorical data
df.fillna(0, inplace=True)
# null values are replaced with zeros for the numerical data
```

After imputation the dataset is as follows,

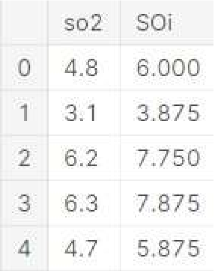


	state	location	type	so2	no2	rspm	spm	pm2_5
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0.0
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	0.0	0.0	0.0
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0.0
3	Andhra Pradesh	Hyderabad	Residential, 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

CALCULATE AIR QUALITY INDEX FOR SO2 BASED ON FORMULA

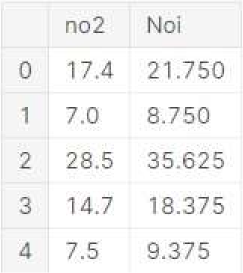
The air quality index is a piecewise linear function of the pollutant concentration. At the boun dary between AQI categories, there is a discontinuous jump of one AQI unit. To convert from concentration to AQI this equation is used,

```
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()
# calculating the individual pollutant index for so2(sulphur dioxide)
```



	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

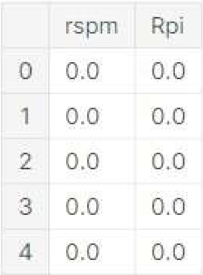
```
Function to calculate no2 individual pollutant index(ni) def
cal_Noi(no2):
    ni=0     if(no2<=40):
ni= no2*50/40     elif(no2>40
and no2<=80):      ni= 50+(no2-
40)*(50/40)     elif(no2>80 and
no2<=180):      ni= 100+(no2-
80)*(100/100)     elif(no2>180
and no2<=280):      ni=
200+(no2-180)*(100/100)
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()
# calculating the individual pollutant index for no2(nitrogen dioxide)
```



	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

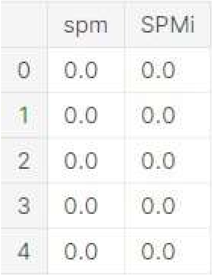
```
Function to calculate rspm individual pollutant index(rpi) def
cal_RSPMI(rspm):
    rpi=0          if(rpi<=30):
rpi=rpi*50/30    elif(rpi>30 and
rpi<=60):          rpi=50+(rpi-
30)*50/30       elif(rpi>60  and
rpi<=90):         rpi=100+(rpi-
60)*100/30      elif(rpi>90  and
rpi<=120):        rpi=200+(rpi-
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()
# calculating the individual pollutant index for rspm(respirable suspended
particualte matter concentration)
```



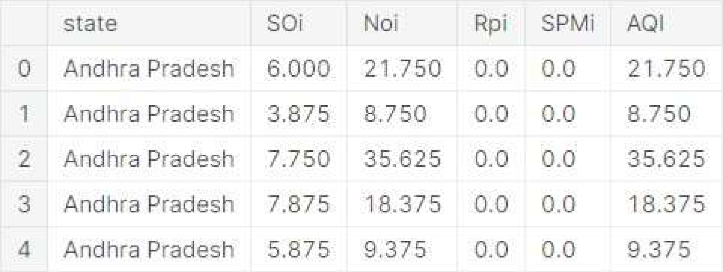
	rspm	Rpi
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

```
Function to calculate spm individual pollutant index(spi) 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()
# calculating the individual pollutant index for spm(suspended particulate
matter)
```



	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

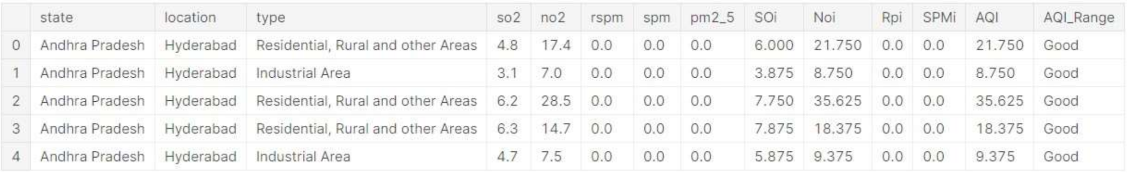
```
function to calculate the air quality index (AQI) of every data value def
cal_aqi(si,ni,rspmi,spmi):
    aqi=0     if(si>ni and si>rspmi and
si>spmi):
     aqi=si     if(ni>si and ni>rspmi
and ni>spmi):
     aqi=ni     if(rspmi>si and rspmi>ni and
rspmi>spmi):
     aqi=rspmi     if(spmi>si and spmi>ni
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()
# Caluclating the Air Quality Index.
```



	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

Now Using threshold values to classify a particular values as good, moderate, poor, unhealthy , very unhealthy and Hazardous.

```
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()
```



	state	location	type	so2	no2	rspm	spm	pm2_5	SOi	Noi	Rpi	SPMi	AQI	AQI_Range
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	Good
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	Good
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	Good
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	Good
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	Good

Split data into independent and dependent values

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

Where X is independent value and Y is dependent value.

Finding Optimum Algorithm

We try algorithms such as,

- Decision Tree Classifier
- Random Forest Classifier
- Random Forest Regressor
- Decision Tree Regressor

From above algorithms we get Random Forest Regressor as the one with highest accuracy

- RSquared value on train : 0.9999860110739481
- RSquared value on test : 0.9998918038557246

Code :

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_st
ate=70)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape) #
splitting the data into training and testing data

RF=RandomForestRegressor().fit(X_train,Y_train)
#predicting train
train_preds1=RF.predict(X_train)
#predicting on test
test_preds1=RF.predict(X_test)
```

```
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))
```