In [ ]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
df = pd.read_csv('airQuality.csv')
df.head(5)
```

Out[ ]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	loc
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 [ ]:
```
df.describe()
```

Out[ ]:

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

```
<class 'pandas.core.frame.DataFrame'>
```

```
<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 [ ]:
```
df.shape
```

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

In [ ]:
```
# Dropping unnecessary columns
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 [ ]:
```
df.isnull().sum()
```

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

In [ ]:
```
df
```

Out[ ]:

	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
435740	Lakshadweep	NaN	NaN	NaN	NaN	NaN	NaN	NaN
435741	Tripura	NaN	NaN	NaN	NaN	NaN	NaN	NaN

435742 rows × 8 columns

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

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

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

In [ ]:
```
df
```

Out[ ]:

	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
			Residential, Rural					

435740	Lakshadweep	Guwahati	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

In [ ]:
```
# Function to calculate so2 individual pollutant index(si)¶
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']]
```

In [ ]:
```
#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']]
```

In [ ]:
```
# 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):
```

```
    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']]
```

In [ ]:

```
# 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']]
```

In [ ]:

```
# 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']]
```

In [ ]:

```
# function to calculate the air quality index range
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"
```

```
        return "Hazardous"

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

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

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=70
```

In [ ]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,Y_train)
```

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

In [ ]:
```
#predicting train
train_pred=model.predict(X_train)
```