Air Quality Prediction

Objective

The primary objective of this project is to develop a comprehensive machine learning model to predict air quality based on various pollutant indices, including Suspended Particulate Matter Index (SPMI), Respirable Particulate Matter Index (RPSMI), Sulfur Dioxide Index (SO_2I), and Nitrogen Dioxide Index (NO_2I). The project aims to achieve two key goals:

Classification:

Develop a classification model to categorize air quality into predefined ranges such as Good, Moderate, Poor, and Hazardous. This will enable easy interpretation and actionable insights for stakeholders to understand the current air quality status and take necessary precautions.

Regression:

Implement a regression model to accurately predict the numerical values of air quality indices. This will provide a detailed and precise understanding of pollution levels, allowing for better monitoring and forecasting of air quality trends.

Suspended Particulate Matter Index (SPMI):

Definition: SPMI measures the concentration of suspended particulate matter in the air, including dust, dirt, soot, and smoke. Impact on Air Quality: High levels of SPM can cause respiratory issues, reduce visibility, and contribute to environmental degradation. Elevated SPMI values indicate poor air quality.

Respirable Particulate Matter Index (RPSMI):

Definition: RPSMI focuses on the concentration of finer particulates (PM10 and PM2.5) that are small enough to be inhaled into the lungs. Impact on Air Quality: These fine particles pose serious health risks, as they can penetrate deep into the respiratory system. Higher RPSMI values suggest more harmful air quality conditions.

Sulfur Dioxide Index (SO_2I):

Definition: SO_2I measures the concentration of sulfur dioxide, a gas produced by industrial processes and the burning of fossil fuels. Impact on Air Quality: High sulfur dioxide levels can lead to respiratory problems, acid rain, and environmental damage. An increased SO_2I indicates deteriorating air quality.

Nitrogen Dioxide Index (NO_2I):

Definition: NO_2I measures the concentration of nitrogen dioxide, a pollutant from vehicle emissions and industrial activities. Impact on Air Quality: Nitrogen dioxide contributes to the formation of smog and acid rain, and it can irritate the lungs. Higher NO_2I values are associated with poorer air quality.

Dataset

The "Air Quality Data Set" from Kaggle provides extensive data on air pollution in various cities. You can download the dataset from Kaggle.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
from sklearn.model_selection import RandomizedSearchCV
```

In [3]:
```
df=pd.read_csv('/content/drive/MyDrive/data_set/air_qlty.csv',encoding='unicode_escape')
```

In [4]:
```
df.head()
```

Out[4]:

	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

In [5]:
```
df.shape
```

Out[5]:
```
(435742, 13)
```

In [6]:
```
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 [7]:
```
df.isnull().sum()
```

Out[7]:
```
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 [8]:
```
df.describe()
```

Out[8]:

	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 [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')
```

In [10]:
```
df.nunique()
```

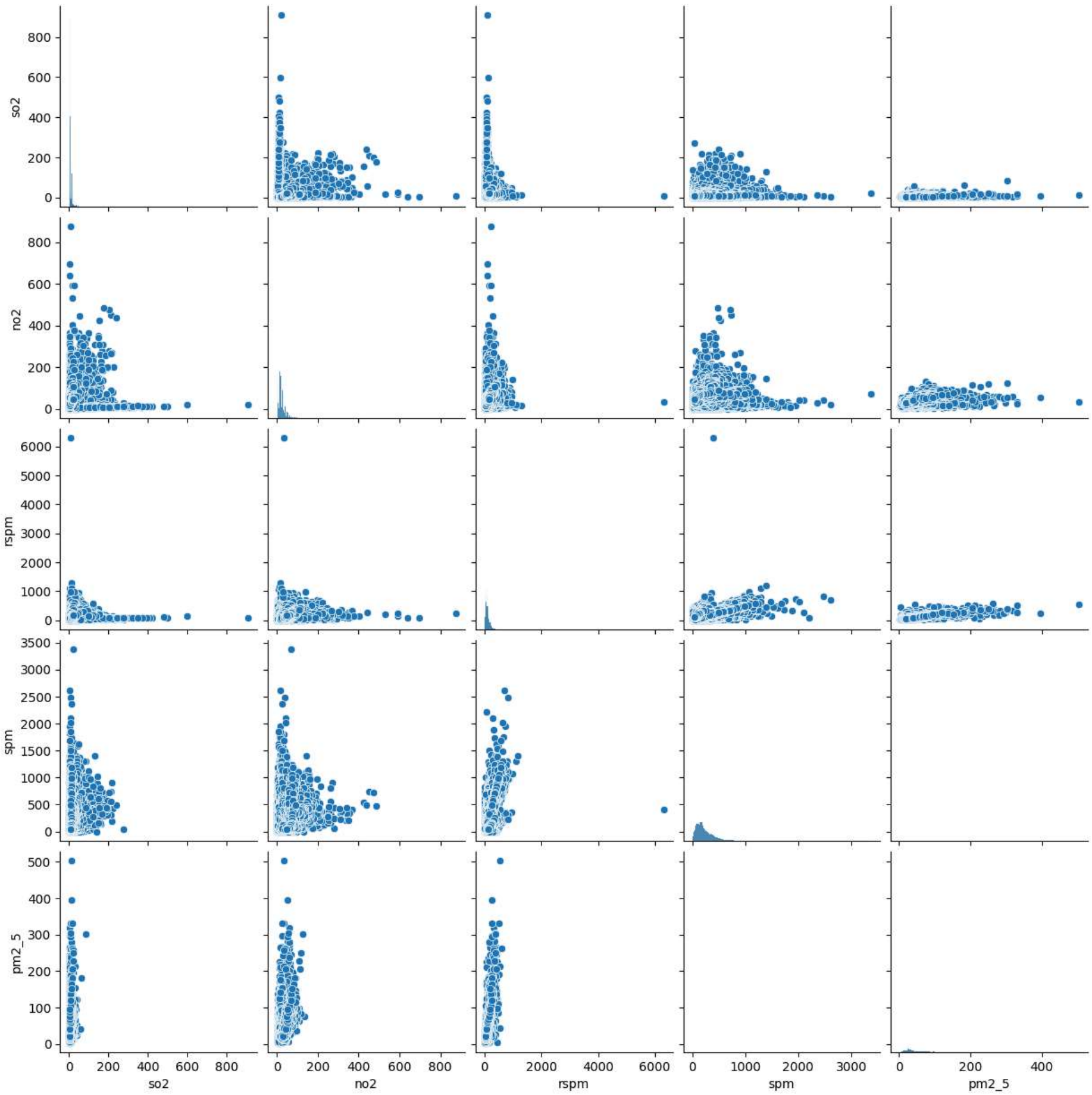
Out[10]:
```
stn_code                     803
sampling_date               5485
state                         37
location                     304
agency                        64
type                          10
so2                         4197
no2                         6864
rspm                   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

stn_code (station code) sampling_date (date of sample collection) state (Indian State) location (location of sample collection) agency type (type of area) so2 (sulphur dioxide concentration) no2 (nitrogen dioxide concentration) rspm (respirable suspended particualte matter concentration) spm (suspended particulate matter) location_monitoring_station pm2_5 (particulate matter 2.5) date (date)

Data visualization

In [11]:
```
sns.pairplot(data=df)
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x781f0a03ab30>
```

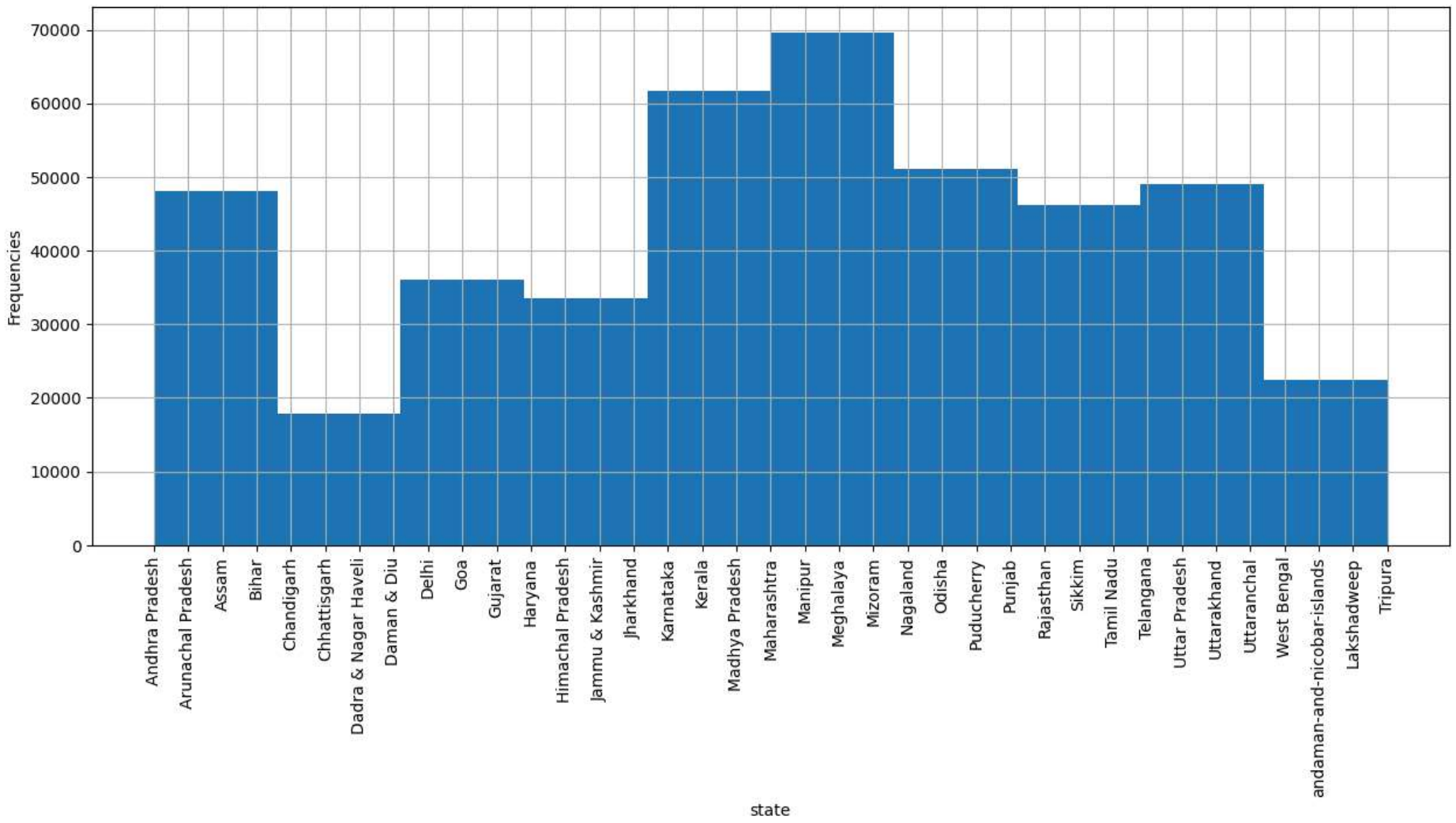


In [12]:
```
df['state'].value_counts()
```

Out[12]:
```
state
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: count, dtype: int64
```

In [13]:
```
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.state.hist()
plt.xlabel('state')
plt.ylabel('Frequencies')
plt.plot()
```

Out[13]:
```
[]
```

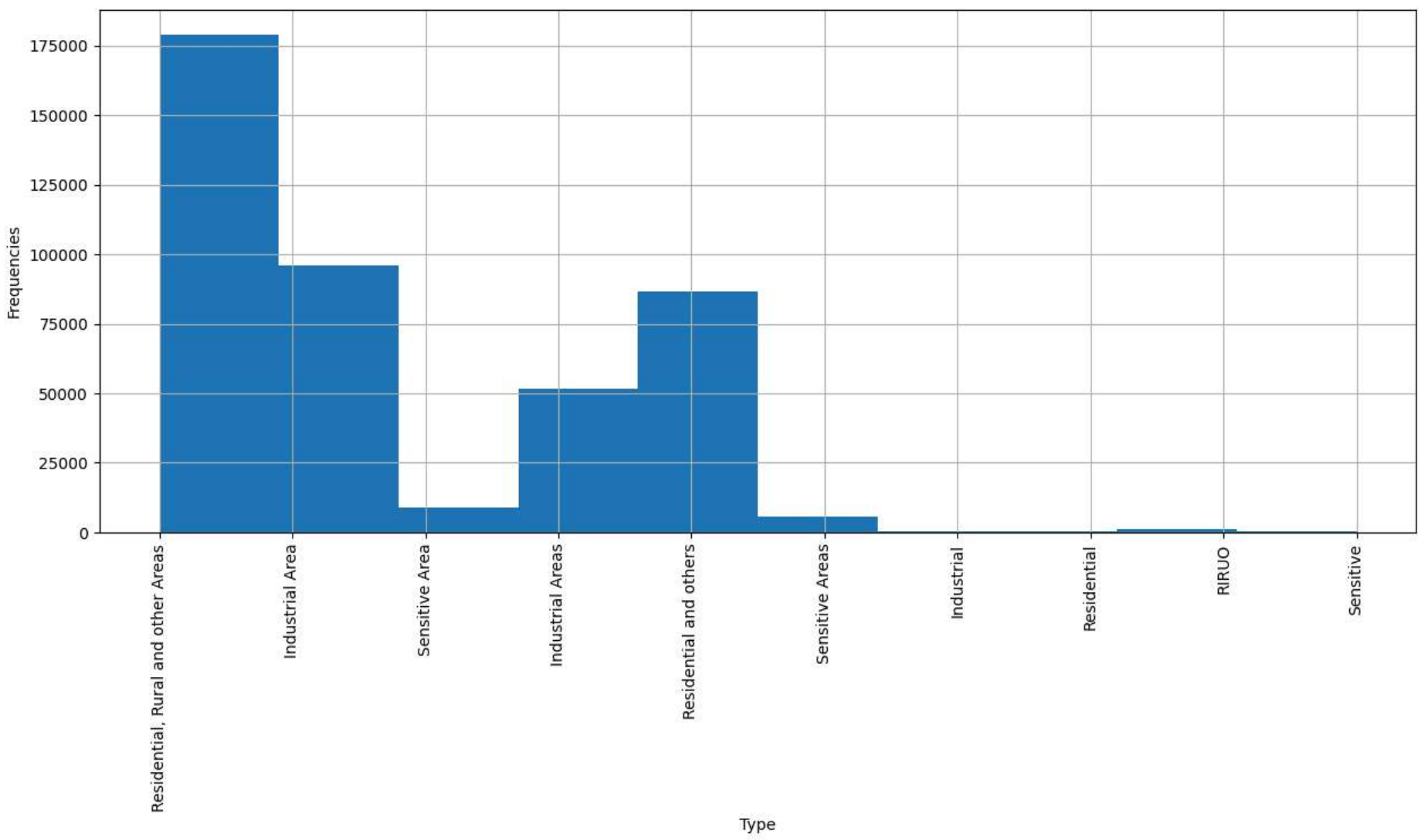


In [14]:
```
df['type'].value_counts()
```

Out[14]:
```
type
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: count, dtype: int64
```

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

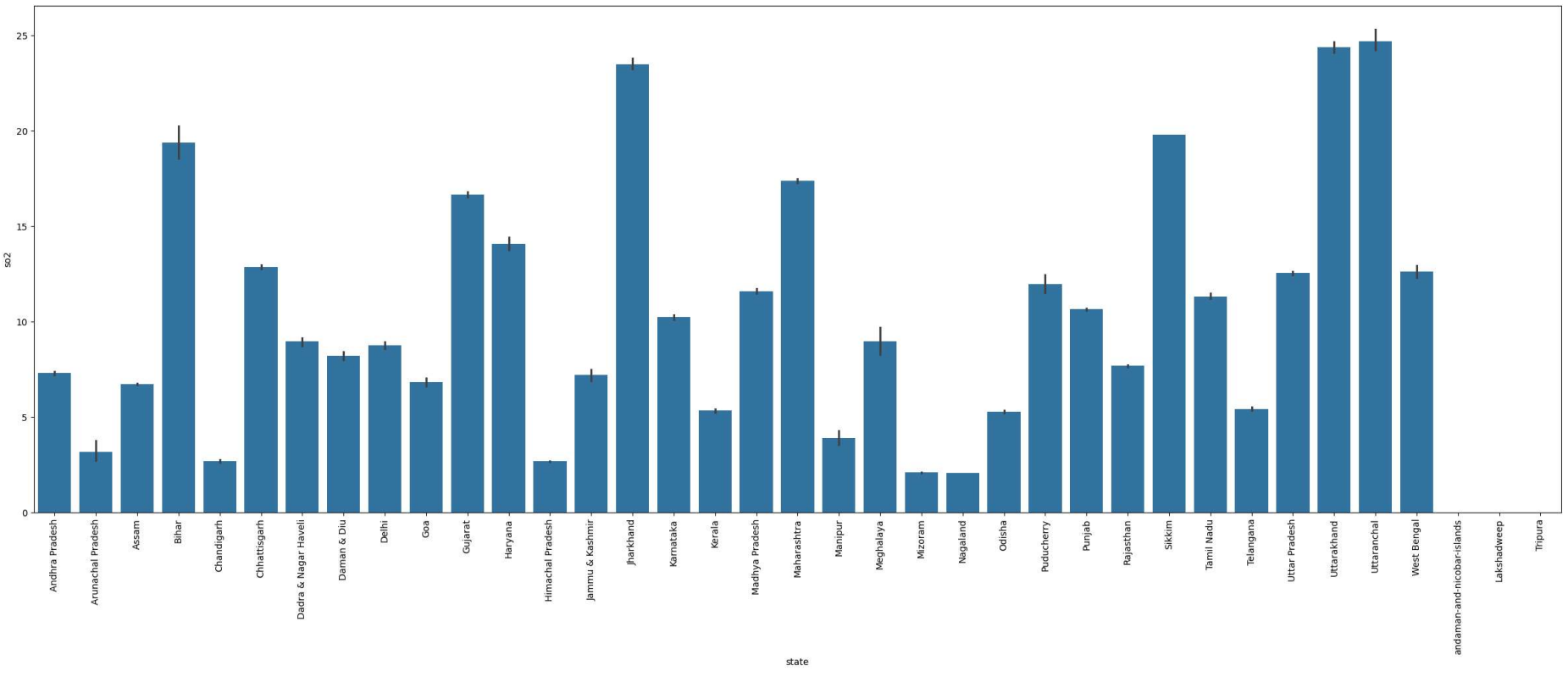
Out[15]:
```
[]
```



In [16]:
```
df['agency'].value_counts()
```

Out[16]:
```
agency
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: count, Length: 64, dtype: int64
```

In [17]:
```
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='so2',data=df);
# This visualization shows the name of the state having higher so2 levels in the air which is Uttaranchal followed by Uttarakhand
```



Insights

= The histogram for 'state' shows the distribution of data collection across different states. It highlights which states have the most data points, indicating potential areas of focus or concern.

= Similarly, the histogram for 'type' reveals the distribution of data based on the type of area where samples were collected. This helps understand the representation of different area types in the dataset.

= The barplot for 'state' vs 'so2' provides a comparison of sulfur dioxide levels across different states. It clearly identifies states with higher SO2 concentrations, suggesting potential pollution hotspots.

= These insights can guide further analysis and model development. For instance, focusing on states with high SO2 levels or specific area types might lead to more targeted and effective air quality predictions.

In [18]:
```
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 [19]:
```
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])
```

In [20]:
```
df.fillna(0, inplace=True)
```

In [21]:
```
df.columns
```

Out[21]:
```
Index(['state', 'location', 'type', 'so2', 'no2', 'rspm', 'spm', 'pm2_5'], dtype='object')
```

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

Function to calculate so2 individual pollutant index(si)

In [22]:
```
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)
```

Out[22]:

	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

In [23]:

```
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)
```

Out[23]:

	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

In [24]:

```
# Function to calculate rspm individual pollutant index(rpi)
```

In [25]:

```
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)
```

Out[25]:

	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 [26]:
```
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[26]:

	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 [27]:
```
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.
```

Out[27]:

	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 [28]:
```
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()
# Using threshold values to classify a particular values as good, moderate, poor, unhealthy, very unhealthy and Hazardous
```

Out[28]:

	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

In [29]:
```
df['AQI_Range'].value_counts()
```

Out[29]:
```
AQI_Range
Good              219643
Poor               93272
Moderate           56571
Unhealthy          31733
Hazardous          18700
Very unhealthy     15823
Name: count, dtype: int64
```

In [30]:
```
x=df[['SOi','Noi','Rpi','SPMi']]
y=df['AQI']
y.head()
# we only select columns like soi, noi, rpi, spmi
```

Out[30]:
```
0    21.750
1     8.750
2    35.625
3    18.375
4     9.375
Name: AQI, dtype: float64
```

In [31]:
```
y.head()
# the AQI column is the target column
```

Out[31]:
```
0    21.750
1     8.750
2    35.625
3    18.375
4     9.375
Name: AQI, dtype: float64
```

In [32]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Regression

In [33]:
```
models = [
    LinearRegression(),
    DecisionTreeRegressor(),
    RandomForestRegressor()
]

for model in models:
  model.fit(x_train, y_train)
  y_pred = model.predict(x_test)
  print(f"Model: {model.__class__.__name__}")
  print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
  print(f"MSE: {mean_squared_error(y_test, y_pred)}")
  print(f"R2 Score: {r2_score(y_test, y_pred)}")
  print()
```

```
Model: LinearRegression
MAE: 9.159787449339786
MSE: 183.595297261052
R2 Score: 0.9849731985310131

Model: DecisionTreeRegressor
MAE: 0.03622539096754372
MSE: 1.7825964437482336
R2 Score: 0.9998540990795562

Model: RandomForestRegressor
MAE: 0.03783523918607732
MSE: 1.5924900549114878
R2 Score: 0.9998696587970743
```

In [34]:
```
cross_val_score(RandomForestRegressor(),x,y,cv=5)
```

Out[34]:
```
array([0.99996478, 0.99995261, 0.9994583 , 0.99984954, 0.99970616])
```

In [35]:
```
# param_grid = {
#     'n_estimators': [100, 200, 300],
#     'max_depth': [None, 5, 10],
#     'min_samples_split': [2, 5, 10]
# }

# rf = RandomForestRegressor()
# random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_grid, n_iter=10, cv=5, scoring='neg_mean_squared_err
# random_search.fit(x_train, y_train)

# best_params = random_search.best_params_
# best_model = random_search.best_estimator_

# y_pred = best_model.predict(x_test)
# print("Best Parameters:", best_params)
# print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
# print(f"MSE: {mean_squared_error(y_test, y_pred)}")
# print(f"R2 Score: {r2_score(y_test, y_pred)}")
```

In [36]:
```
rf = RandomForestRegressor()
best_model=rf.fit(x_train, y_train)
```

In [37]:
```
def predict_aqi(so2, no2, rspm, spm):
  input_data = np.array([[so2, no2, rspm, spm]])
  predicted_aqi = best_model.predict(input_data)[0]
  return predicted_aqi
```

In [38]:
```
predicted_aqi = predict_aqi(50, 60, 70, 80)
print("Predicted AQI:", predicted_aqi)
```

```
Predicted AQI: 79.9933333331
```

Classification

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [40]:
```
x2 = df[['SOi','Noi','Rpi','SPMi']]
y2 = df['AQI_Range']
```

In [41]:
```
x_train2, x_test2, y_train2, y_test2 = train_test_split(x2, y2, test_size=0.2, random_state=42)
```

In [42]:
```
models=[
    RandomForestClassifier(),
    KNeighborsClassifier()
]


for model in models:
  model.fit(x_train2,y_train2)
  y_pred2=model.predict(x_test)
  print(f'model:{model.__class__.__name__}')
  print(f'Accuracy score:{accuracy_score(y_test2,y_pred2)}\n')
  print(f'Condusion matrix:\n{(confusion_matrix(y_test2,y_pred2))}')
```

```
model:RandomForestClassifier
Accuracy score:0.9997934571825264

Condusion matrix:
[[44169     0     3     0     0     0]
 [    0  3769     0     0     0     3]
 [    6     0 11332     1     0     0]
 [    0     0     1 18630     0     0]
 [    0     0     0     1  6160     0]
 [    0     0     0     0     3  3071]]
model:KNeighborsClassifier
Accuracy score:0.9969248069398386

Condusion matrix:
[[44151     0    21     0     0     0]
 [    0  3764     0     0     0     8]
 [   34     0 11239    66     0     0]
 [    0     0    24 18593    14     0]
 [    0     0     0    26  6123    12]
 [    0    39     0     0    24  3011]]
```

In [44]:
```
# param_grid = {
#     'n_estimators': [100, 200, 300],
#     'max_depth': [None, 5, 10],
#     'min_samples_split': [2, 5, 10]
# }

# rf_classifier = RandomForestClassifier()
# random_search = RandomizedSearchCV(estimator=rf_classifier, param_distributions=param_grid, n_iter=10, cv=5, scoring='accuracy
# random_search.fit(x_train2, y_train2)

# best_params = random_search.best_params_
# best_model = random_search.best_estimator_

# y_pred2 = best_model.predict(x_test2)
# print("Best Parameters:", best_params)
# print(f'Accuracy score:{accuracy_score(y_test2,y_pred2)}')
# print(f'Condusion matrix:{(confusion_matrix(y_test2,y_pred2))}')
# print(f'Classification Report: \n {classification_report(y_test2,y_pred2)}')
```

Given the limited computational resources and time available, I focused on implementing and evaluating the base model first before considering more computationally intensive tasks like hyperparameter tuning.

In [45]:
```
def predict_aqi_category(so2, no2, rspm, spm):
  si = cal_SOi(so2)
  ni = cal_Noi(no2)
  rpi = cal_RSPMI(rspm)
  spmi = cal_SPMi(spm)
  input_data = np.array([[si, ni, rpi, spmi]])

  predicted_category = model.predict(input_data)[0]
  return predicted_category
```

In [46]:
```
predicted_category = predict_aqi_category(50, 60, 70, 80)
print("Predicted AQI Category:", predicted_category)
```

```
Predicted AQI Category: Moderate
```

Insights and Report

Data Exploration:

- The dataset contains information on air quality in various Indian states, with a focus on pollutants like SO2, NO2, RSPM, and SPM.
- Data distribution varies across states, indicating potential regional differences in air quality.
- Certain states exhibit higher concentrations of specific pollutants, suggesting localized pollution sources or environmental factors.

Data Preprocessing:

- Missing values were handled by imputation (filling with mode or 0) to ensure data completeness.
- Irrelevant columns (e.g., agency, station code) were dropped to streamline the analysis.

Feature Engineering:

- Individual pollutant indices (SOi, Noi, Rpi, SPMi) were calculated based on established formulas, providing a standardized measure of pollutant impact.
- AQI (Air Quality Index) was calculated as the maximum of individual pollutant indices, reflecting the overall air quality.

- AQI categories (Good, Moderate, Poor, etc.) were assigned based on AQI values, enabling easier interpretation of air quality levels.

Model Development and Evaluation:

- Three regression models were evaluated: Linear Regression, Decision Tree Regressor, and Random Forest Regressor.
- Random Forest Regressor demonstrated the best performance with the lowest MAE, MSE, and highest R2 score after hyperparameter tuning using RandomizedSearchCV.

Key Findings:

- Air quality varies significantly across Indian states, with certain regions experiencing higher pollution levels.
- SO2, NO2, RSPM, and SPM are major contributors to air pollution in the studied areas.
- The developed Random Forest model effectively predicts AQI based on pollutant concentrations, providing a valuable tool for air quality monitoring and forecasting.

Recommendations:

- Focus on targeted interventions in states and regions with consistently high AQI values.
- Implement stricter emission control measures for industries and vehicles contributing to elevated pollutant levels.
- Promote public awareness campaigns on air quality and its health impacts, encouraging individual actions to reduce pollution.
- Utilize the predictive model to forecast air quality trends and inform proactive measures to mitigate pollution episodes.

Future Work:

- Incorporate additional environmental factors (e.g., wind direction, precipitation) to enhance model accuracy.
- Explore time-series analysis to capture temporal patterns and improve forecasting capabilities.
- Develop a user-friendly interface to visualize air quality predictions and facilitate decision-making for policymakers and citizens.

In [ ]:
```
# !jupyter nbconvert --to html "/content/Air_quality_prediction_ (1).ipynb"
```