In [2]:
```
#Importing necessary libraries
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
from prettytable import PrettyTable
```

In [1]:
```
!pip install prettytable
```

```
Requirement already satisfied: prettytable in c:\users\tanima\anaconda3\lib\site-p
ackages (3.6.0)
Requirement already satisfied: wcwidth in c:\users\tanima\anaconda3\lib\site-packa
ges (from prettytable) (0.2.5)
```

In [3]:
```
#Reading the dataset
df=pd.read_csv('AQIdata.csv',encoding='unicode_escape')
```

In [4]:
```
#Loading the dataset
df.head()
```

Out[4]:

| | stn_code | sampling_date | state | location | agency | type | so2 | no2 | rspm | spm | loca |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 [5]:
```
df.shape
# As we can see that there are 45742 rows and 13 columns in the dataset
```

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

In [6]:
```
# Checking the overall information on the dataset
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()
# There are a lot of missing values present in the dataset
```

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]:
```
# Checking the descriptive stats of the numeric values present in the data like mea
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]:
```
# These are all the unique values present in the dataframe
```

```
df.nunique()
```

Out[9]:
```
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
```

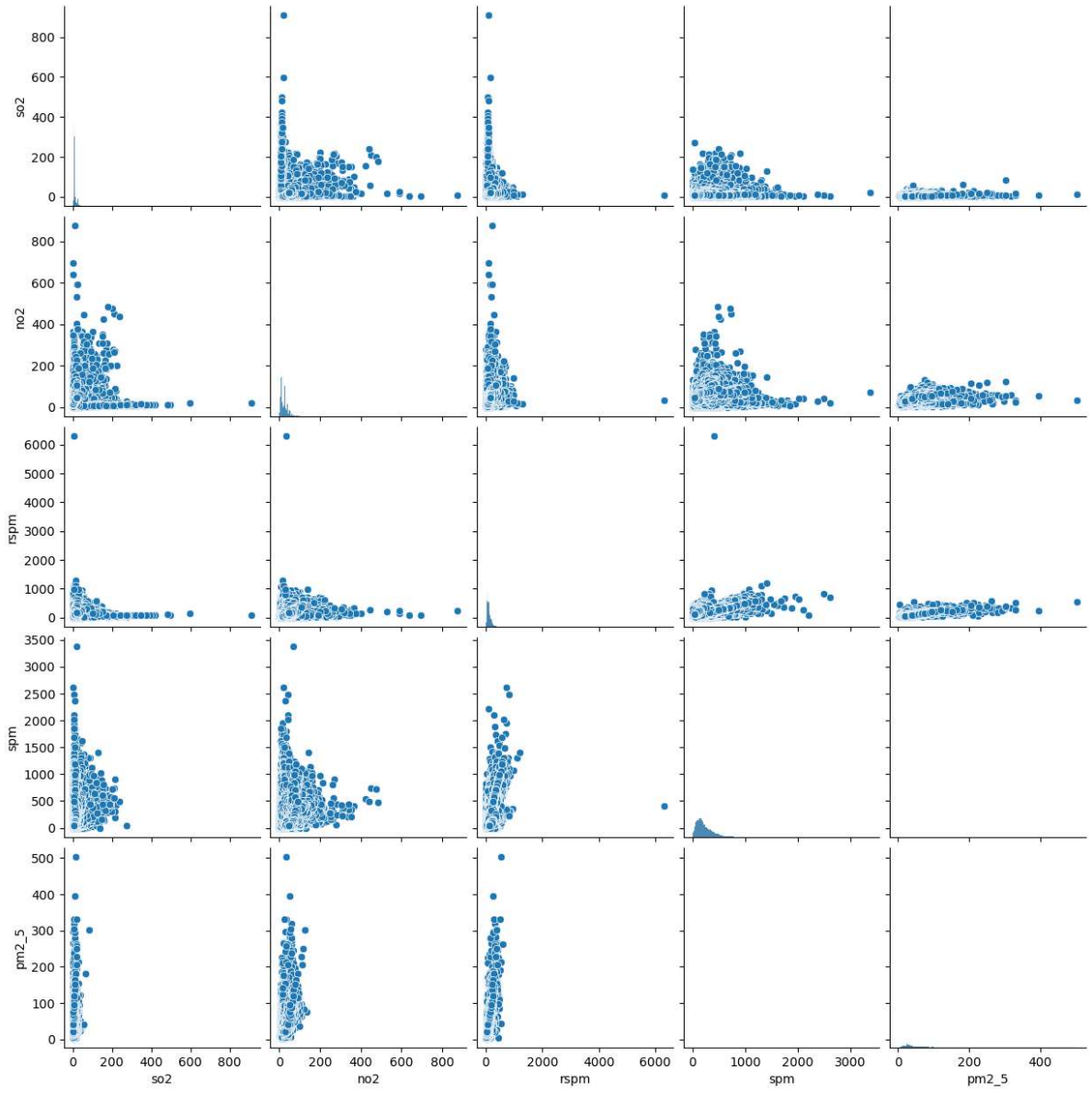
In [10]:
```
df.columns
# These are all columns present in the dataset
```

Out[10]:
```
Index(['stn_code', 'sampling_date', 'state', 'location', 'agency', 'type',
       'so2', 'no2', 'rspm', 'spm', 'location_monitoring_station', 'pm2_5',
       'date'],
      dtype='object')
```

In [11]:
```
# Data Visualization
```

In [12]:
```
sns.pairplot(data=df)
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x22200039df0>
```




In [13]:
```
# Checking all nullvalues and treating those null values
```

In [14]:
```
# Checking all null values
nullvalues=df.isnull().sum().sort_values(ascending=False)
```

In [15]:
```
nullvalues
# Higher null values present in pm2_5 followed by spm
```

Out[15]:
```
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 [16]:
```
null_values_percentage = (df.isnull().sum()/df.isnull().count()*100).sort_values(a
# Count (returns non-NAN value)
```

In [17]:
```
missing_data_with_percentage = pd.concat([nullvalues, null_values_percentage], axis
# Concatenating total null values and their percentage of missing values for furthe
```

In [18]:
```
missing_data_with_percentage
# Percentage of null values present in the dataset
```

Out[18]:

| | Total | Percentage |
|---|---|---|
| **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 |

In [19]:
```
# 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 [20]:
```
# Now checking the null values
df.isnull().sum()
```

Out[20]:
```
state            0
location         3
type          5393
so2          34646
no2          16233
rspm         40222
spm         237387
pm2_5       426428
dtype: int64
```

In [21]:
```
df
```

Out[21]:

| | 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 [22]:
```
# null value imputation for catagorical data
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])
```

In [23]:
```
# null values are replaced with zeros for the numerical data
df.fillna(0, inplace=True)
```

In [24]:
```
# now we have successfully imputed null values which were present in the dataset
df.isnull().sum()
```

Out[24]:
```
state       0
location    0
type        0
so2         0
no2         0
rspm        0
spm         0
pm2_5       0
dtype: int64
```

In [25]:
```
# Function to calculate so2 individual pollutant index(si)
```

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

| | 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 [27]:
```
# Function to calculate no2 individual pollutant index(ni)
```

In [28]:
```
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<=380):
        ni=100+(no2-80)*(100/300)
    elif(no2>380 and no2<=800):
        ni=200+(no2-380)*(100/420)
    elif(no2>800 and no2<1600):
        ni=300+(no2-800)*(100/800)
    elif(no2>1600):
        ni=400+(no2-1600)*(100/800)
    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[28]:

| | 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 [29]:
```
# Function to calculate rspm individual pollutant index(rpi)
```

In [30]:
```
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 particul
```

Out[30]:

| | 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 [31]:
```
# Function to calculate spm individual pollutant index(spi)
```

In [32]:
```
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)
```

Out[32]:

| | 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 |

In [33]:
```
# Function to calculate the air quality index (AQI) of every data value
```

In [34]:
```
def cal_aqi(si,ni,rpi,spi):
    aqi=0
```

```
    if(si>ni and si>rpi and si>spi):
        aqi=si
    if(ni>si and ni>rpi and ni>spi):
        aqi=ni
    if(rpi>si and rpi>ni and rpi>spi):
        aqi=rpi
    if(spi>si and spi>ni and spi>rpi):
        aqi=spi
    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()
#Calculating the air quality index
```

Out[34]:

| | 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 [35]:

```
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 "Heatwave"

df['AQI_Range']=df['AQI'] .apply(AQI_Range)
df.head()
# using threshold values to classify a particular values as good, moderate, poor, u
```

Out[35]:

| | state | location | type | so2 | no2 | rspm | spm | pm2_5 | SOi | NOi | Rpi | SPMI | 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 | 21.7 |
| **1** | Andhra Pradesh | Hyderabad | Industrial Area | 3.1 | 7.0 | 0.0 | 0.0 | 0.0 | 3.875 | 8.750 | 0.0 | 0.0 | 8.7 |
| **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 | 35.6 |
| **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 | 18.3 |
| **4** | Andhra Pradesh | Hyderabad | Industrial Area | 4.7 | 7.5 | 0.0 | 0.0 | 0.0 | 5.875 | 9.375 | 0.0 | 0.0 | 9.3 |

In [36]:
```
df['AQI_Range'].value_counts()
# These are the counts of values present in the AQI_Range column
```

Out[36]:
```
Good               219663
Poor                93454
Moderate            56571
Unhealthy           31572
Heatwave            18692
Very Unhealthy      15790
Name: AQI_Range, dtype: int64
```

In [37]:
```
# Splitting the dataset into Dependent and Independent columns
```

In [38]:
```
# we only select columns like soi, noi, rspi, spi
X=df[['SOi', 'NOi', 'Rpi', 'SPMI']]
Y=df['AQI']
X.head()
```

Out[38]:

| | 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 [39]:
```
# the AQI column is the target column
Y.head()
```

Out[39]:
```
0    21.750
1     8.750
2    35.625
3    18.375
4     9.375
Name: AQI, dtype: float64
```

In [40]:
```
# Splitting the data into training and testing data
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,)
```

In [41]:
```
#Linear Regression model
```

In [42]:
```
model=LinearRegression()
model.fit(X_train,Y_train)
```

Out[42]:
```
LinearRegression()
```

In [43]:
```
#Predicting on train
train_pred=model.predict(X_train)
#predicting on test
test_pred=model.predict(X_test)
```

In [44]:
```
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 =  13.036579259471658
RMSE TestData =  13.16451366168132
--------------------------------------------------
RSquared value on train: 0.9861161278938458
RSquared value on test: 0.9858222098899132
```

In [45]:
```
#Decision Tree Regressor
```

In [46]:
```
DT=DecisionTreeRegressor()
DT.fit(X_train,Y_train)
```

Out[46]:
```
DecisionTreeRegressor()
```

In [47]:
```
#predicting train
train_preds=DT.predict(X_train)
#predicting test
test_preds=DT.predict(X_test)
```

In [48]:
```
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 =  2.2618569275593674e-13
RMSE TestData =  1.1937728681513136
--------------------------------------------------
RSquared value on train: 1.0
RSquared value on test: 0.999883415098039
```

In [49]:
```
#Random Forest Regressor
```

In [50]:
```
RF=RandomForestRegressor().fit(X_train,Y_train)
```

In [51]:
```
#predicting train
train_preds1=RF.predict(X_train)
#predicting test
test_preds1=RF.predict(X_test)
```

In [52]:
```
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 =  0.4082428949234372
RMSE TestData =  1.0248199432160114
--------------------------------------------------
RSquared value on train: 0.9999863849081467
RSquared value on test: 0.9999140800451395
```

In [53]:
```
#Classification Algorithms
```

In [54]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [55]:
```
# Splitting the data into independent and dependent columns for classification
X2 = df[['SOi','NOi','Rpi','SPMI']]
Y2 = df[['AQI_Range']]
```

In [56]:
```
# Splitting the data into training and testing data
X_train2,X_test2,Y_train2,Y_test2=train_test_split(X2,Y2,test_size=0.33,random_sta
```

In [57]:
```
#Logistic Regression model
```

In [58]:
```
#fit the model on train data
log_reg = LogisticRegression().fit(X_train2,Y_train2)

#predict on train
train_preds2=log_reg.predict(X_train2)
#accuracy on train
print("Model accuracy on train is:", accuracy_score(Y_train2,train_preds2))

#predict on test
test_preds2=log_reg.predict(X_test2)
#accuracy on test
print("Model accuracy on test is:", accuracy_score(Y_test2,test_preds2))
print('-'*50)

# Kappa score
print ('KappaScore is:', metrics.cohen_kappa_score(Y_test2,test_preds2))
```

```
Model accuracy on train is: 0.7540615248658147
Model accuracy on test is: 0.7545881289335512
--------------------------------------------------
KappaScore is: 0.6239966563223742
```

In [59]:
```
#prediction on random values
log_reg.predict([[727,327.55,78.2,100]])
```

Out[59]:
```
array(['Unhealthy'], dtype=object)
```

In [60]:
```
#prediction on random values
log_reg.predict([[2.7,45,35.16,23]])
```

Out[60]:
```
array(['Good'], dtype=object)
```

In [61]:
```
#Decision Tree Classifier
```

In [62]:
```
#fit the model on train data
DT2 = DecisionTreeClassifier().fit(X_train2,Y_train2)

#predict on train
train_preds3=DT2.predict(X_train2)
#accuracy on train
print("Model accuracy on train is:", accuracy_score(Y_train2,train_preds3))

#predict on test
test_preds3=DT2.predict(X_test2)
#accuracy on test
print("Model accuracy on test is:", accuracy_score(Y_test2,test_preds3))
print('-'*50)

# Kappa score
print ('KappaScore is:', metrics.cohen_kappa_score(Y_test2,test_preds3))
```

```
Model accuracy on train is: 1.0
Model accuracy on test is: 0.9997774609687402
--------------------------------------------------
KappaScore is: 0.9996698510519874
```

In [63]:
```
#Random Forest Classifier
```

In [64]:
```
#fit the model on train data
RF = RandomForestClassifier().fit(X_train2,Y_train2)

#predict on train
train_preds4=RF.predict(X_train2)
#accuracy on train
print("Model accuracy on train is:", accuracy_score(Y_train2,train_preds4))

#predict on test
test_preds4=RF.predict(X_test2)
#accuracy on test
print("Model accuracy on test is:", accuracy_score(Y_test2,test_preds4))
print('-'*50)

# Kappa score
print ('KappaScore is:', metrics.cohen_kappa_score(Y_test2,test_preds4))
```

```
Model accuracy on train is: 1.0
Model accuracy on test is: 0.9998400500712821
--------------------------------------------------
KappaScore is: 0.9997627093380352
```

In [65]:
```
#K-Nearest Neighbours
```

In [66]:
```
#fit the model on train data
KNN = KNeighborsClassifier().fit(X_train2,Y_train2)
```

```
#predict on train
train_preds5=KNN.predict(X_train2)
#accuracy on train
print("Model accuracy on train is:", accuracy_score(Y_train2,train_preds5))

#predict on test
test_preds5=KNN.predict(X_test2)
#accuracy on test
print("Model accuracy on test is:", accuracy_score(Y_test2,test_preds5))
print('-'*50)

# Kappa score
print ('KappaScore is:', metrics.cohen_kappa_score(Y_test2,test_preds5))
```

```
Model accuracy on train is: 0.9980818436223013
Model accuracy on test is: 0.996627142807469
--------------------------------------------------
KappaScore is: 0.9949952864587799
```

In [67]:
```
#prediction on random values
KNN.predict([[7.4,47.7,78.182,100]])
```

Out[67]:
```
array(['Poor'], dtype=object)
```

In [68]:
```
#prediction on random values
KNN.predict([[325.7,345,798.182,203]])
```

Out[68]:
```
array(['Unhealthy'], dtype=object)
```

In [ ]: