Data Analytics with Cognos-Group2

Project Title: Air Quality Analysis in Tamil Nadu

Group Members;

1) **Sivaperumalraj G (TL)**

2) Thirumurugan

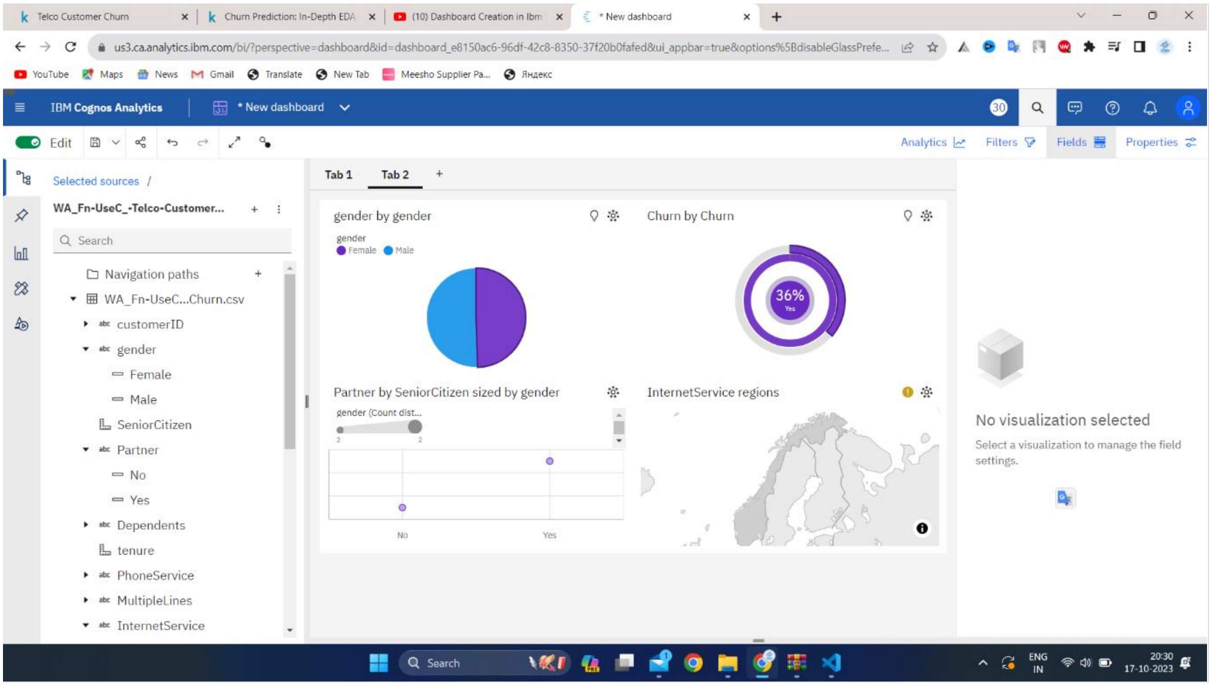
3) Vamsi

4) Tamilarasan

5) Udayakumar

Phase 4

IBM Cognos Visualization:-



IBM Cognos can be a valuable tool for air quality analysis in Tamil Nadu using machine learning. Air quality analysis typically involves collecting and analyzing various data sources, including pollution data, meteorological data, and other relevant information. Here's how you can use IBM Cognos for this purpose:

1. **Data Collection and Integration:**

- Gather historical air quality data, including pollutant concentrations (PM2.5, PM10, CO, NO2, etc.) and meteorological data (temperature, humidity, wind speed, etc.). You can obtain this data from government agencies, environmental monitoring stations, or other sources.

- Integrate this data into IBM Cognos using data modules or data sets. Cognos provides data connectors to access and import data from various sources.

2. **Data Preprocessing:**

- Clean and preprocess the data, handling missing values, outliers, and inconsistencies.

- Create calculated fields if needed, such as air quality indices or meteorological indices.

3. **Feature Engineering:**

- Identify relevant features, such as time of day, weather conditions, and pollutant concentrations, that could impact air quality.

- Engineer new features if necessary to improve model performance.

4. **Machine Learning Model Building:**

- Utilize IBM Watson Studio or other machine learning tools integrated with Cognos to build predictive models.

- Choose appropriate machine learning algorithms for air quality prediction, such as regression, time series analysis, or deep learning.

- Split your data into training and testing sets to evaluate model performance.

5. **Model Evaluation and Visualization:**

- Assess the model's performance using evaluation metrics like mean squared error (MSE), R-squared, or others relevant to air quality prediction.

- Create visualizations and dashboards in IBM Cognos to display model results and insights. Visualizations can include historical air quality trends, forecasts, and comparisons with actual measurements.

6. **Data Monitoring and Reporting:**

- Implement monitoring mechanisms to continuously collect real-time air quality data.

- Use IBM Cognos to create reports and dashboards that provide real-time insights into air quality conditions in Tamil Nadu.

7. **Alerting and Notifications:**

- Configure alerts and notifications in Cognos to trigger actions when air quality exceeds predefined thresholds. For example, you can notify relevant authorities or the public when air quality becomes hazardous.

8. **Data Sharing and Public Access:**

- Utilize Cognos to publish air quality reports and visualizations for public access, allowing citizens to stay informed about air quality conditions in their area.

9. **Policy Recommendations:**

- Use the insights generated by your analysis to make data-driven policy recommendations for improving air quality in Tamil Nadu.

10. **Continuous Improvement:**

- Continuously refine your models and data collection processes to enhance the accuracy and reliability of air quality predictions.

By combining IBM Cognos with machine learning, you can create a comprehensive air quality analysis and prediction system for Tamil Nadu. This approach allows you to make informed decisions, take preventive actions, and improve air quality management in the region.

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.linear_model import Ridge
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 sklearn.impute import SimpleImputer

%matplotlib inline

import os

# Reading the dataset
df=pd.read_csv('data.csv',encoding='unicode_escape')
```

Data Understanding

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

```
  stn_code       sampling_date           state   location agency  \
0    150.0  February - M021990  Andhra Pradesh  Hyderabad    NaN
1    151.0  February - M021990  Andhra Pradesh  Hyderabad    NaN
2    152.0  February - M021990  Andhra Pradesh  Hyderabad    NaN
3    150.0     March - M031990  Andhra Pradesh  Hyderabad    NaN
4    151.0     March - M031990  Andhra Pradesh  Hyderabad    NaN

                                 type  so2   no2  rspm  spm  \
0  Residential, Rural and other Areas  4.8  17.4   NaN  NaN
1                     Industrial Area  3.1   7.0   NaN  NaN
2  Residential, Rural and other Areas  6.2  28.5   NaN  NaN
3  Residential, Rural and other Areas  6.3  14.7   NaN  NaN
4                     Industrial Area  4.7   7.5   NaN  NaN

  location_monitoring_station  pm2_5        date
0                         NaN    NaN  1990-02-01
```

```
1                         NaN    NaN  1990-02-01
2                         NaN    NaN  1990-02-01
3                         NaN    NaN  1990-03-01
4                         NaN    NaN  1990-03-01

# As we can see that there are 4,35,742 rows and 13 columns in the
dataset
df.shape

(435742, 13)

# Checking the over all 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

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

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
```

```
# 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()
```

```
                 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
```

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

```
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
```

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

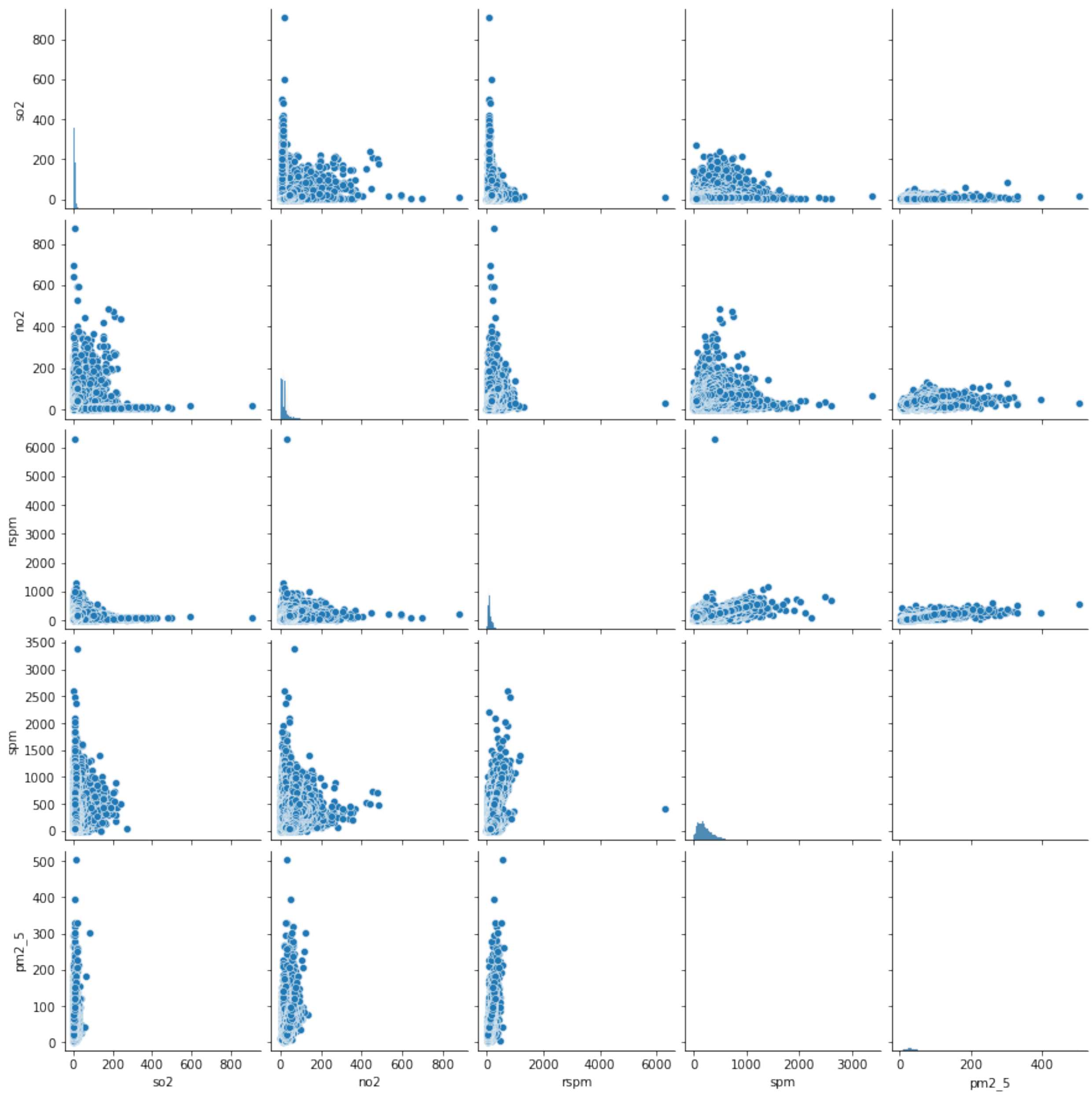
```
Index(['stn_code', 'sampling_date', 'state', 'location', 'agency', 
'type',
       'so2', 'no2', 'rspm', 'spm', 'location_monitoring_station',
```

```
'pm2_5',
       'date'],
      dtype='object')
```

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)

```
sns.pairplot(data=df)
```

```
<seaborn.axisgrid.PairGrid at 0x2c0a06a4190>
```



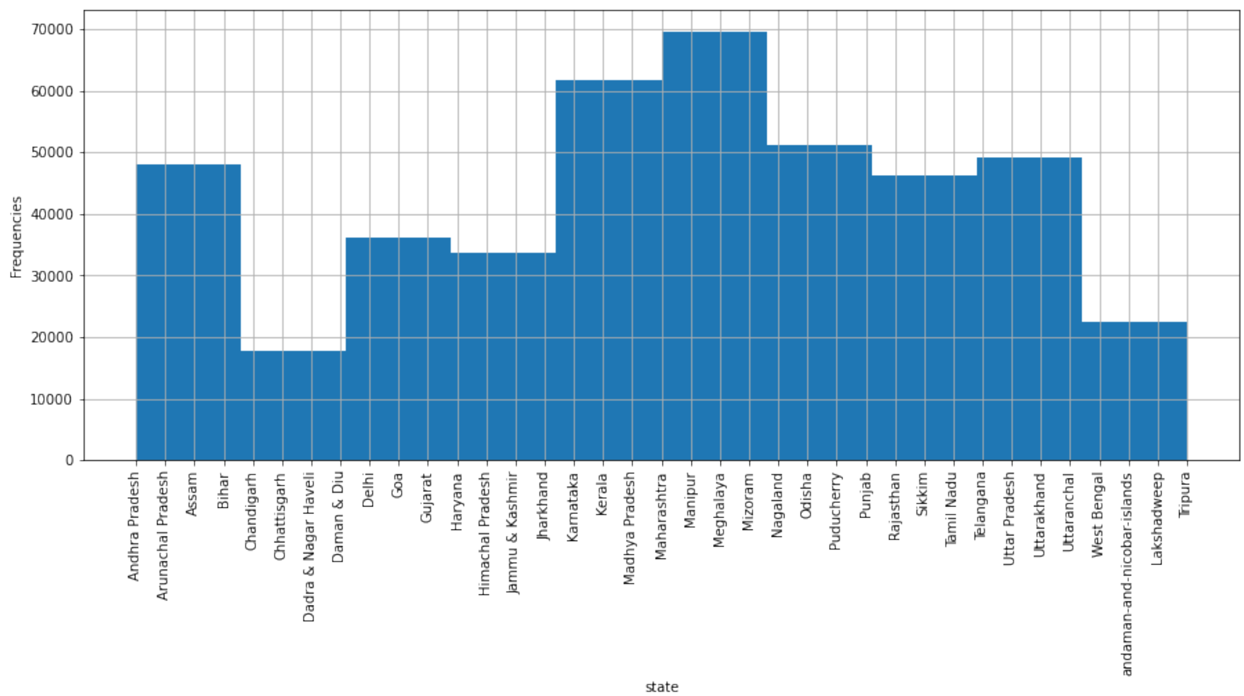

```python
# Viewing the count of values present in the state column
df['state'].value_counts()
```

```
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
```

```python
# The visualization shows us the count of states present in the dataset.
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.state.hist()
plt.xlabel('state')
plt.ylabel('Frequencies')
plt.plot()
```

```
[]
```

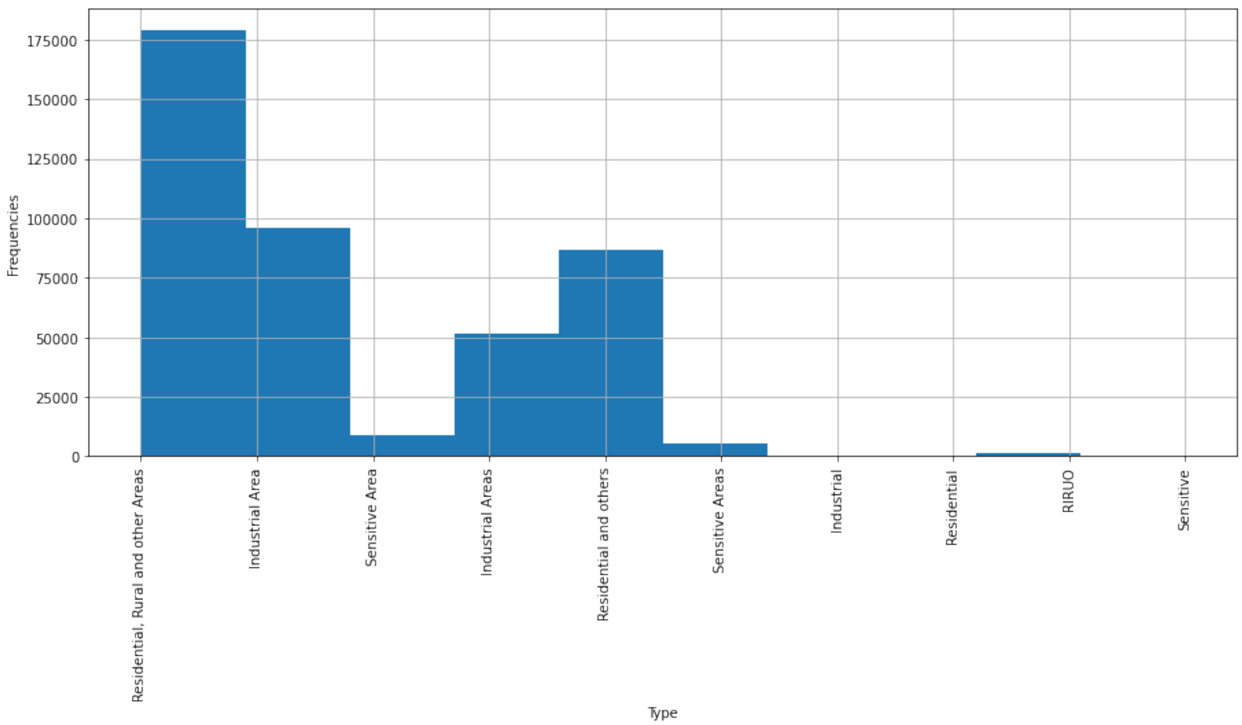



```
# Viewing the count of values present in the type column
df['type'].value_counts()
```

```
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
```

```
# The visualization shows us the count of Types present in the dataset.
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.type.hist()
plt.xlabel('Type')
plt.ylabel('Frequencies')
plt.plot()
```

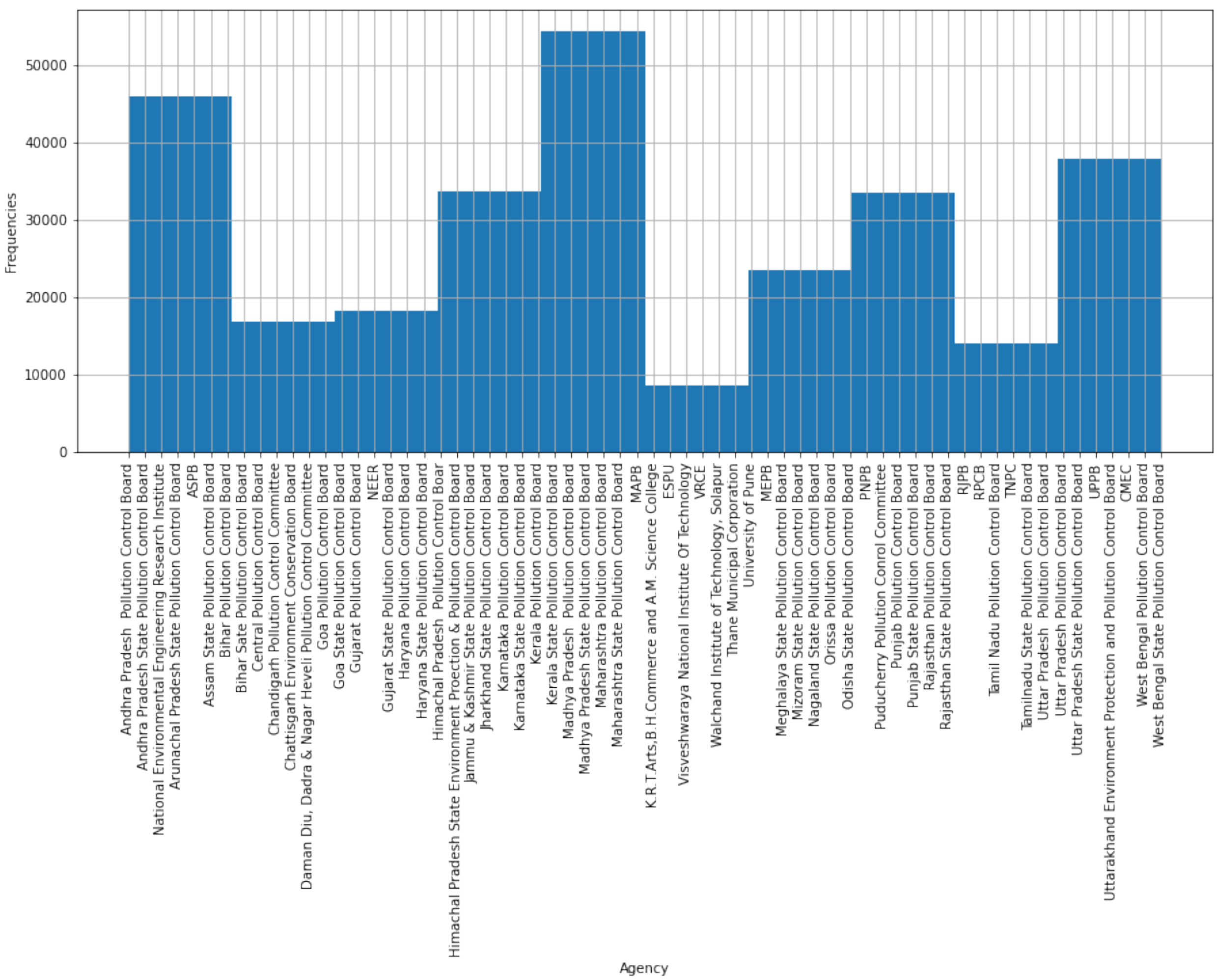
```
[]
```



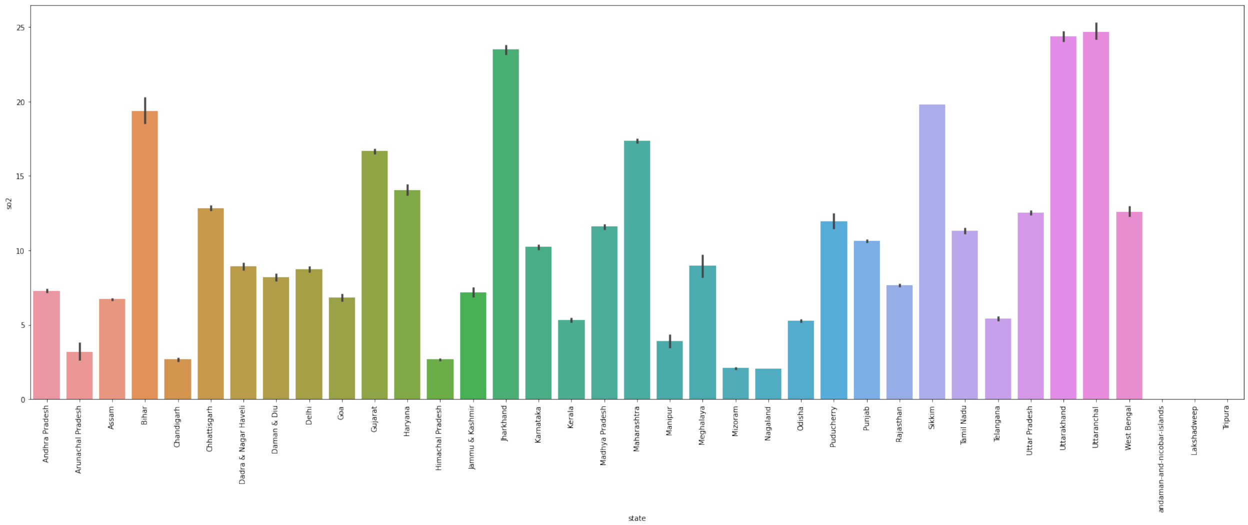

```
#Viewing the counts of values present in the agency column
df['agency'].value_counts()

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
```

```python
# The visualization shows us the count of Agency present in the dataset.
plt.figure(figsize=(15, 6))
plt.xticks(rotation=90)
df.agency.hist()
plt.xlabel('Agency')
plt.ylabel('Frequencies')
plt.plot()
```

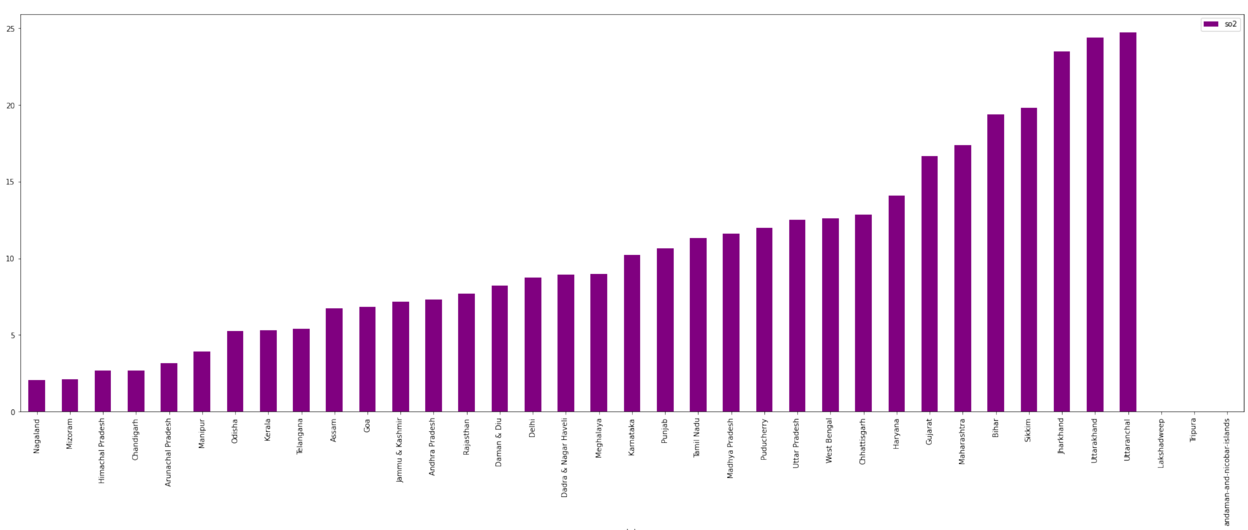
```
[]
```



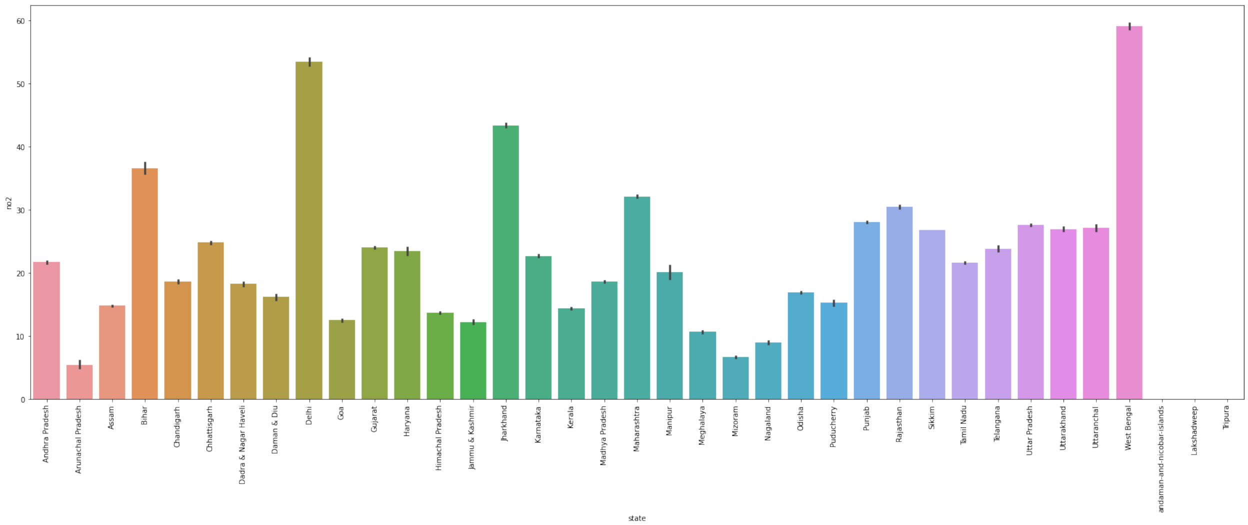

```python
# This visualization shows the name of the state having higher so2 levels in the air which is Uttaranchal followed by Uttarakhand
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='so2',data=df);
```



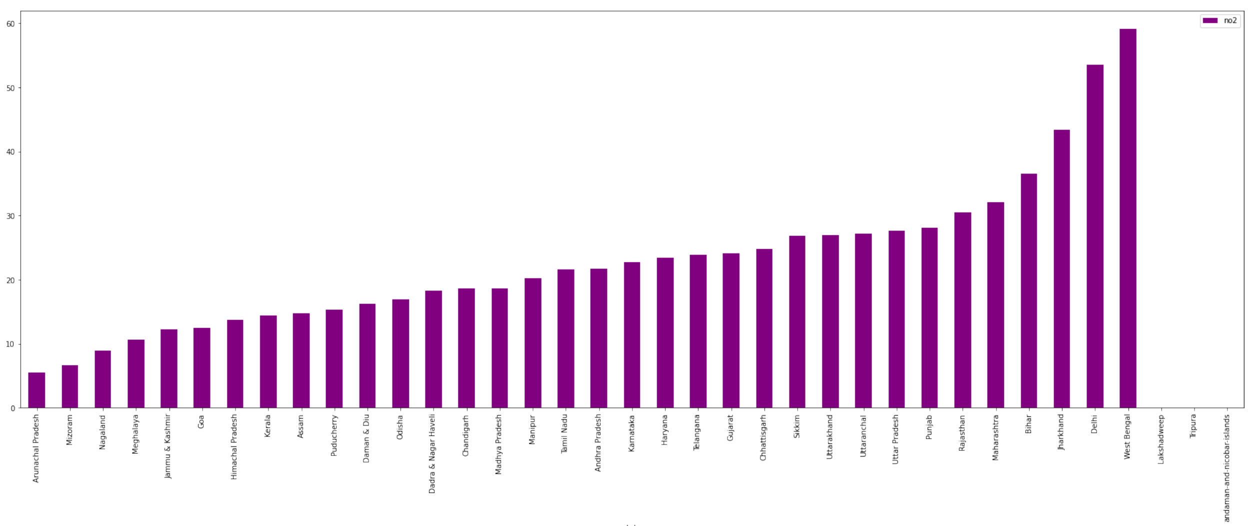

```python
plt.rcParams['figure.figsize']=(30,10)

# We can also use the groupby function to sort values in an ascending order based on the x-axis, y-axis and its keys
# Below we get a clear picture of the states in an increasing order based on their so2 levels.
df[['so2','state']].groupby(["state"]).mean().sort_values(by='so2').plot.bar(color='purple')
plt.show()
```



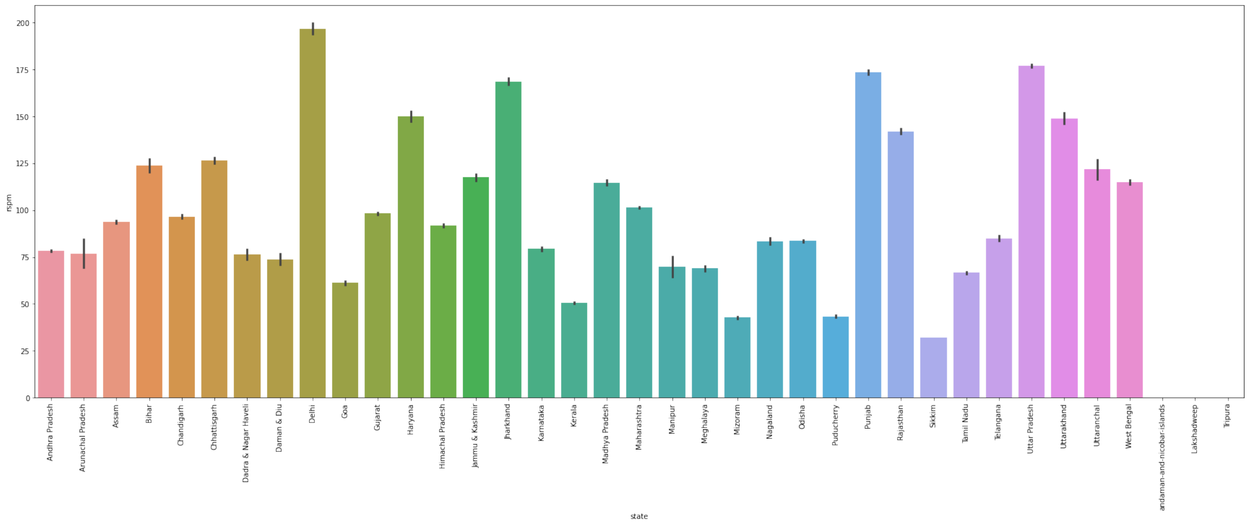

```python
# West bengal has a higher no2 level compared to other states
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='no2',data=df);
```



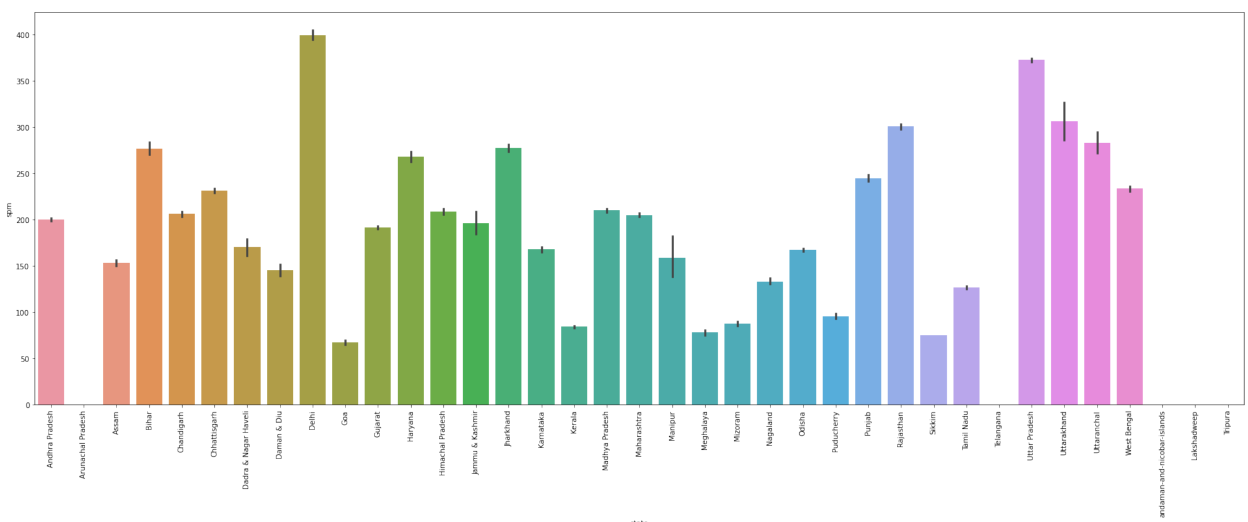

```python
# We can also use the groupby function to sort values in an ascending order based on the x-axis, y-axis and its keys
# Below we get a clear picture of the states in an increasing order based on their no2 levels.
df[['no2','state']].groupby(["state"]).mean().sort_values(by='no2').plot.bar(color='purple')
plt.show()
```



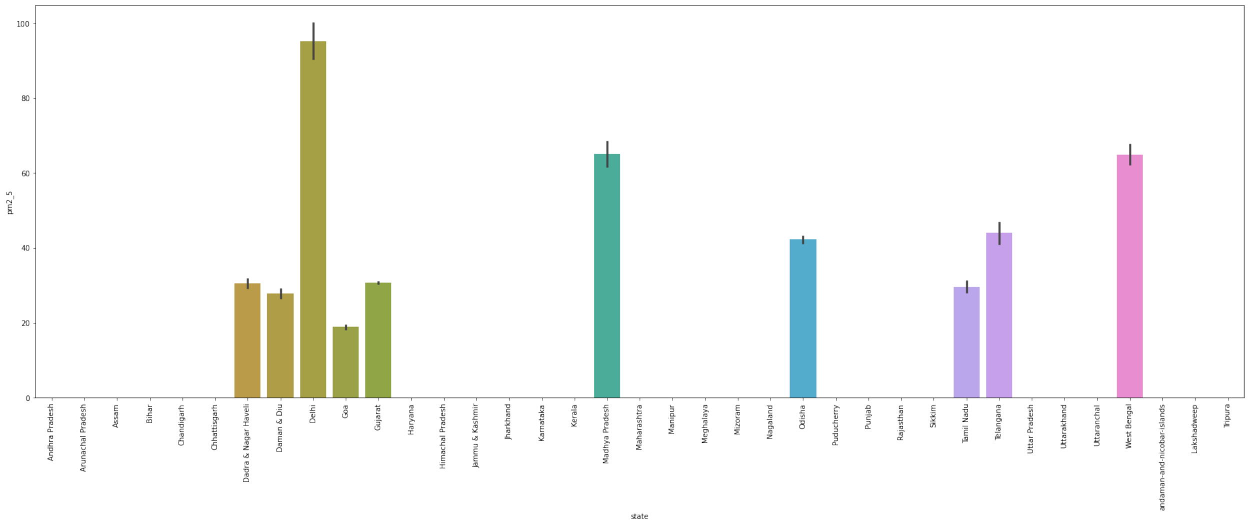

```python
# Delhi has higher rspm level compared to other states
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='rspm',data=df);
```




```python
# Delhi has higher spm level compared to other states
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='spm',data=df);
```




```python
# Delhi has higher pm2_5 level compared to other states
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='state',y='pm2_5',data=df);
```



Checking all null values and treating those null values.

```
# Checking all null values
nullvalues = df.isnull().sum().sort_values(ascending=False)

# higher null values present in pm2_5 followed by spm
nullvalues

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

#count(returns Non-NAN value)
null_values_percentage =
(df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=Fals
e)

# Concatenating total null values and their percentage of missing
values for further imputation or column deletion
missing_data_with_percentage = pd.concat([nullvalues,
null_values_percentage], axis=1, keys=['Total', 'Percent'])
```

```
# As you can see below these are the percentages of null values
present in the dataset
missing_data_with_percentage
```

```
                             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
```

```
# 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)

# Now checking the null values
df.isnull().sum()
```

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

```
df
```

```
                             state   location  \
0                   Andhra Pradesh  Hyderabad
1                   Andhra Pradesh  Hyderabad
2                   Andhra Pradesh  Hyderabad
3                   Andhra Pradesh  Hyderabad
4                   Andhra Pradesh  Hyderabad
...                            ...        ...
435737                 West Bengal   ULUBERIA
435738                 West Bengal   ULUBERIA
435739  andaman-and-nicobar-islands        NaN
```

```
435740                   Lakshadweep        NaN
435741                       Tripura        NaN

                                           type   so2   no2   rspm  spm
pm2_5
0       Residential, Rural and other Areas   4.8  17.4    NaN  NaN
NaN
1                          Industrial Area   3.1   7.0    NaN  NaN
NaN
2       Residential, Rural and other Areas   6.2  28.5    NaN  NaN
NaN
3       Residential, Rural and other Areas   6.3  14.7    NaN  NaN
NaN
4                          Industrial Area   4.7   7.5    NaN  NaN
NaN
...                                    ...   ...   ...    ...  ...
...
435737                               RIRUO  22.0  50.0  143.0  NaN
NaN
435738                               RIRUO  20.0  46.0  171.0  NaN
NaN
435739                                 NaN   NaN   NaN    NaN  NaN
NaN
435740                                 NaN   NaN   NaN    NaN  NaN
NaN
435741                                 NaN   NaN   NaN    NaN  NaN
NaN

[435742 rows x 8 columns]

# Null value Imputation for categorical data
df['location']=df['location'].fillna(df['location'].mode()[0])
df['type']=df['type'].fillna(df['type'].mode()[0])

# null values are replaced with zeros for the numerical data
df.fillna(0, inplace=True)

# Now we have successfully imputed null values which were present in
the dataset
df.isnull().sum()

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

```
# The following features are important for our machine learning
models.
df
```

```
                                 state   location  \
0                       Andhra Pradesh  Hyderabad   
1                       Andhra Pradesh  Hyderabad   
2                       Andhra Pradesh  Hyderabad   
3                       Andhra Pradesh  Hyderabad   
4                       Andhra Pradesh  Hyderabad   
...                                ...        ...   
435737                     West Bengal   ULUBERIA   
435738                     West Bengal   ULUBERIA   
435739  andaman-and-nicobar-islands   Guwahati   
435740                     Lakshadweep   Guwahati   
435741                         Tripura   Guwahati   

                                     type   so2   no2   rspm  spm
pm2_5
0       Residential, Rural and other Areas   4.8  17.4    0.0  0.0
0.0
1                         Industrial Area   3.1   7.0    0.0  0.0
0.0
2       Residential, Rural and other Areas   6.2  28.5    0.0  0.0
0.0
3       Residential, Rural and other Areas   6.3  14.7    0.0  0.0
0.0
4                         Industrial Area   4.7   7.5    0.0  0.0
0.0
...                                   ...   ...   ...    ...  ...
...
435737                              RIRUO  22.0  50.0  143.0  0.0
0.0
435738                              RIRUO  20.0  46.0  171.0  0.0
0.0
435739  Residential, Rural and other Areas   0.0   0.0    0.0  0.0
0.0
435740  Residential, Rural and other Areas   0.0   0.0    0.0  0.0
0.0
435741  Residential, Rural and other Areas   0.0   0.0    0.0  0.0
0.0

[435742 rows x 8 columns]
```

Applying Exploratorty Data Analysis

```
# Applying Exploratorty Data Analysis
```

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)

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

	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)

```
# calculating the individual pollutant index for no2(nitrogen dioxide)
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()
```

	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)

```
# calculating the individual pollutant index for rspm(respirable
suspended particualte matter concentration)
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()
```

	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)

```
# calculating the individual pollutant index for spm(suspended
particulate matter)
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()
```

	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

```
# Caluclating the Air Quality Index.
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()
```

```
             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
```

```python
# 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  \
0  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  4.8
17.4
1  Andhra Pradesh  Hyderabad                     Industrial Area  3.1
7.0
2  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.2
28.5
3  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.3
14.7
4  Andhra Pradesh  Hyderabad                     Industrial Area  4.7
7.5

   rspm  spm  pm2_5    SOi     Noi  Rpi  SPMi     AQI AQI_Range
0   0.0  0.0    0.0  6.000  21.750  0.0   0.0  21.750      Good
1   0.0  0.0    0.0  3.875   8.750  0.0   0.0   8.750      Good
2   0.0  0.0    0.0  7.750  35.625  0.0   0.0  35.625      Good
3   0.0  0.0    0.0  7.875  18.375  0.0   0.0  18.375      Good
4   0.0  0.0    0.0  5.875   9.375  0.0   0.0   9.375      Good
```

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

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

```
# we only select columns like soi, noi, rpi, spmi
X=df[['SOi','Noi','Rpi','SPMi']]
Y=df['AQI']
X.head()
```

```
     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
```

```
# the AQI column is the target column
Y.head()
```

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

```
# 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,rando
m_state=70)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
```

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

```
print(Y_test)
```

```
17593      17.500000
134413     41.250000
360229     31.250000
358484     17.500000
265920      6.250000
             ...
69766     258.000000
391744     26.250000
10306     154.666667
275551    147.333333
372655    279.000000
Name: AQI, Length: 87149, dtype: float64
```

Linear Regression

```
model=LinearRegression()
model.fit(X_train,Y_train)
```

```
LinearRegression()
```

```
#predicting train
train_pred=model.predict(X_train)
#predicting on test
test_pred=model.predict(X_test)
```

```
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 =  13.672937344789004
--------------------------------------------------
RSquared value on train: 0.9849533579250526
RSquared value on test: 0.9847286394495923
```

```
model.coef_
```

```
array([0.14480562, 0.56535211, 0.        , 0.88192549])
```

```
model.intercept_
```

```
7.325911627307576
```

Ridge Regression

```
ridge =Ridge(alpha=1)
ridge.fit(X_train, Y_train)
```

```
Ridge(alpha=1)
```

```
#predicting train
train_pred6=ridge.predict(X_train)
#predicting on test
test_pred6=ridge.predict(X_test)
```

```
RMSE_train=(np.sqrt(metrics.mean_squared_error(Y_train,train_pred6)))
RMSE_test=(np.sqrt(metrics.mean_squared_error(Y_test,test_pred6)))
print("RMSE TrainingData = ",str(RMSE_train))
print("RMSE TestData = ",str(RMSE_test))
print('-'*50)
```

```
print('RSquared value on train:',ridge.score(X_train, Y_train))
print('RSquared value on test:',ridge.score(X_test, Y_test))
```

```
RMSE TrainingData =  13.583424938613533
RMSE TestData =  13.67293734414519
--------------------------------------------------
RSquared value on train: 0.9849533579250526
RSquared value on test: 0.9847286394510305
```

Decision Tree Regressor

```
DT=DecisionTreeRegressor()
DT.fit(X_train,Y_train)
```

```
DecisionTreeRegressor()
```

```
#predicting train
train_preds=DT.predict(X_train)
#predicting on test
test_preds=DT.predict(X_test)

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 =  2.2145111619852308e-13
RMSE TestData =  1.3036957039731905
--------------------------------------------------
RSquared value on train: 1.0
RSquared value on test: 0.9998611625743694
```

Random Forest Regressor

```
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))
```

```
RMSE TrainingData =  0.415318411285987
RMSE TestData =  1.1585719092815703
--------------------------------------------------
RSquared value on train: 0.9999859335863704
RSquared value on test: 0.9998903521621562
```

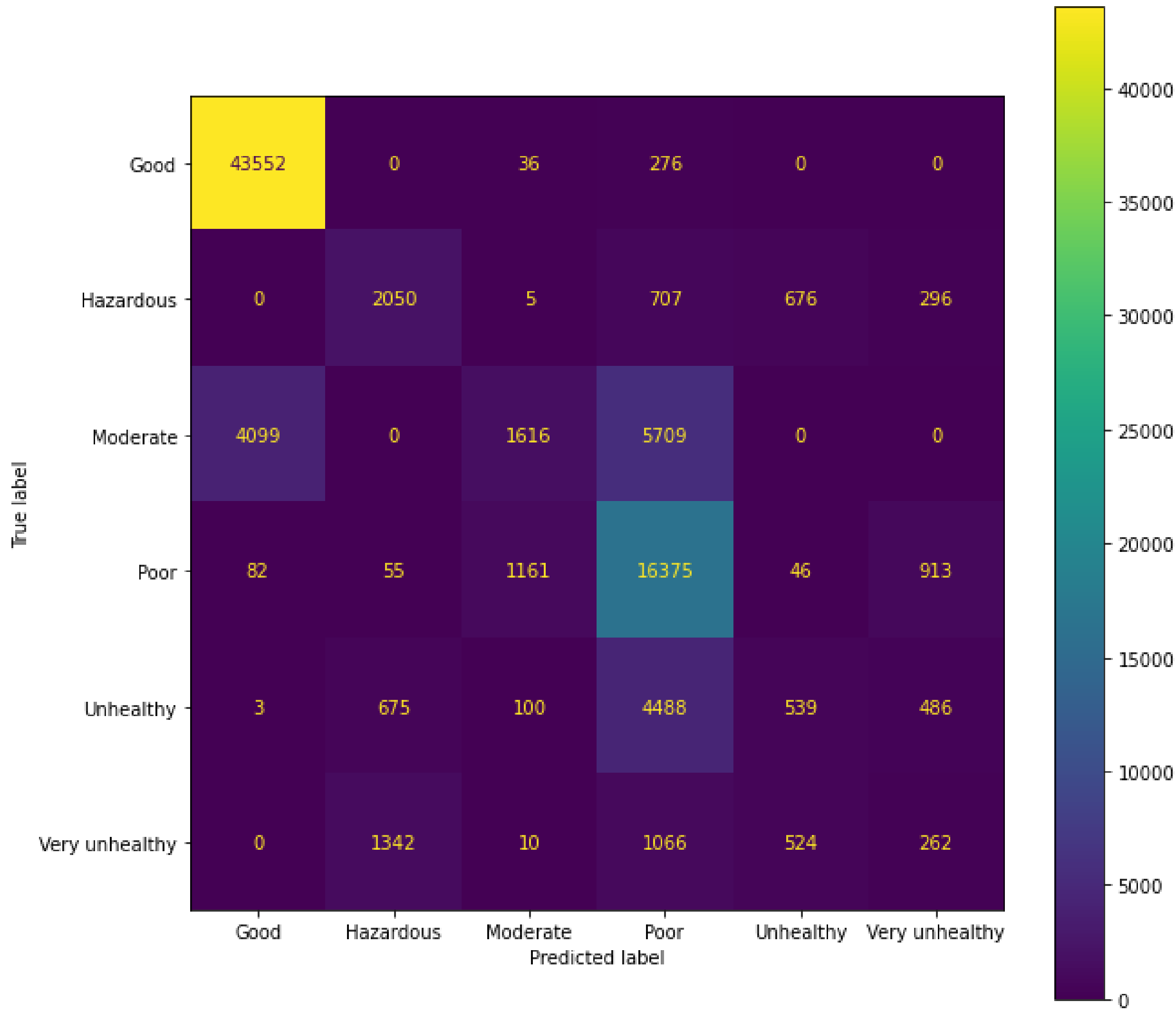
Classification Algorithms

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import RidgeClassifier

# Splitting the data into independent and dependent columns for classification
X2 = df[['SOi','Noi','Rpi','SPMi']]
Y2 = df['AQI_Range']

# Splitting the data into training and testing data
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X2
                                                        , Y2,
test_size=0.20, random_state=70)
```

Logistic Regression

```python
#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:  0.7392030247308465
Model accuracy on test is:  0.7388954549105555
--------------------------------------------------
KappaScore is:  0.5954103699644182

log_reg.predict([[727,327.55,78.2,100]])

array(['Moderate'], dtype=object)

# Predictions on random values
log_reg.predict([[2.4,47.7,78.182,100]])

array(['Poor'], dtype=object)

log_reg.predict([[2,45.8,37,32]])

array(['Moderate'], dtype=object)

from sklearn.metrics import plot_confusion_matrix
fig, ax = plt.subplots(figsize=(10, 10))
plot_confusion_matrix(log_reg, X_test2, Y_test2, ax=ax)
plt.show()
```



Decision Tree Classifier

```
#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:  0.9998508301873802
--------------------------------------------------
KappaScore is:  0.9997791306063205
```

```
DT2.predict([[727,327.55,78.2,100]])
```

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

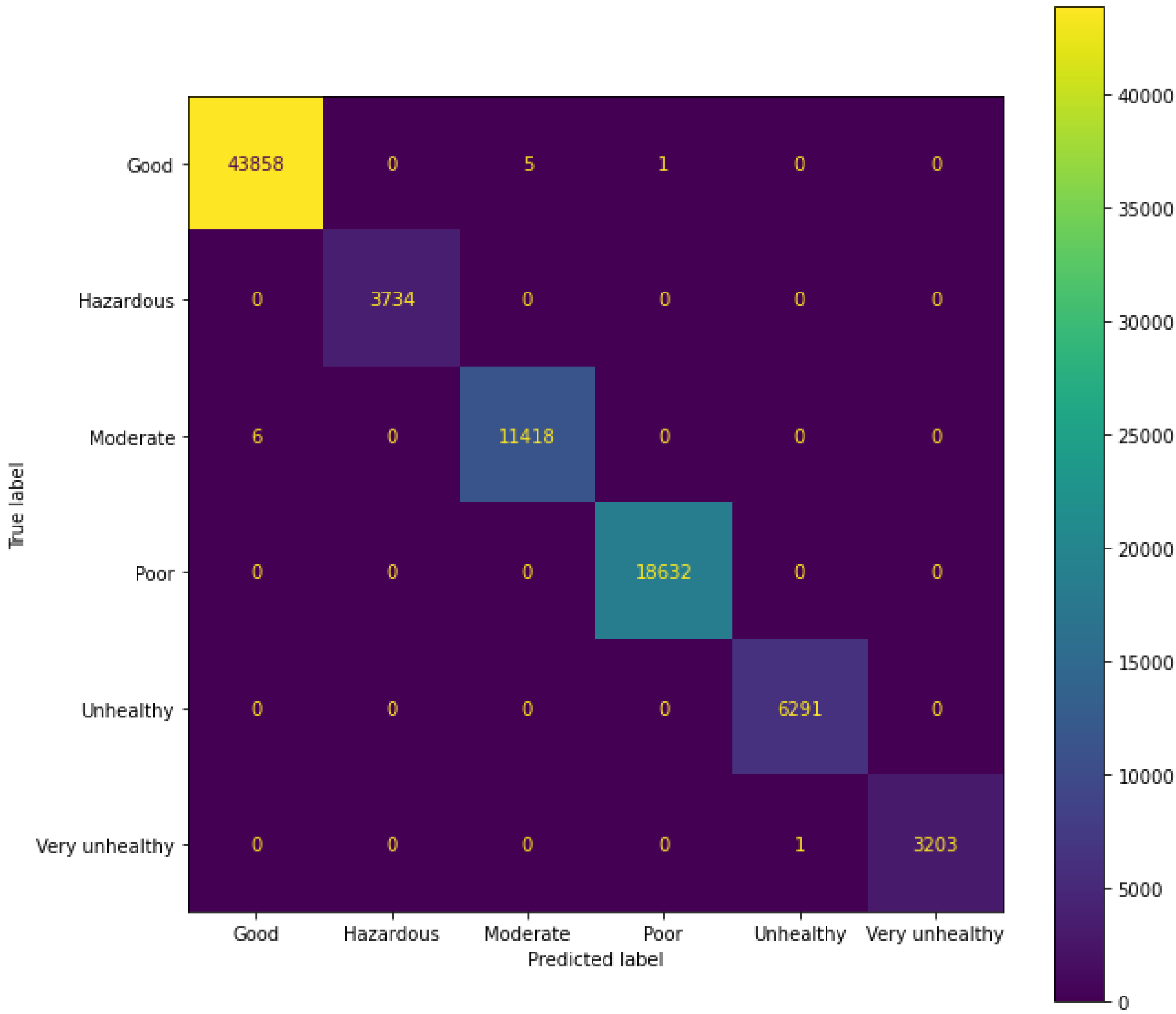
```
# Predictions on random values
DT2.predict([[2.4,47.7,78.182,100]])
```

```
array(['Moderate'], dtype=object)
```

```
DT2.predict([[2,45.8,37,32]])
```

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

```
from sklearn.metrics import plot_confusion_matrix
fig, ax = plt.subplots(figsize=(10, 10))
plot_confusion_matrix(DT2, X_test2, Y_test2, ax=ax)
plt.show()
```



Random Forest Classifier

```
#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:  0.9998508301873802
--------------------------------------------------
KappaScore is:  0.9997791332898625
```

```
RF.predict([[727,327.55,78.2,100]])
```

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

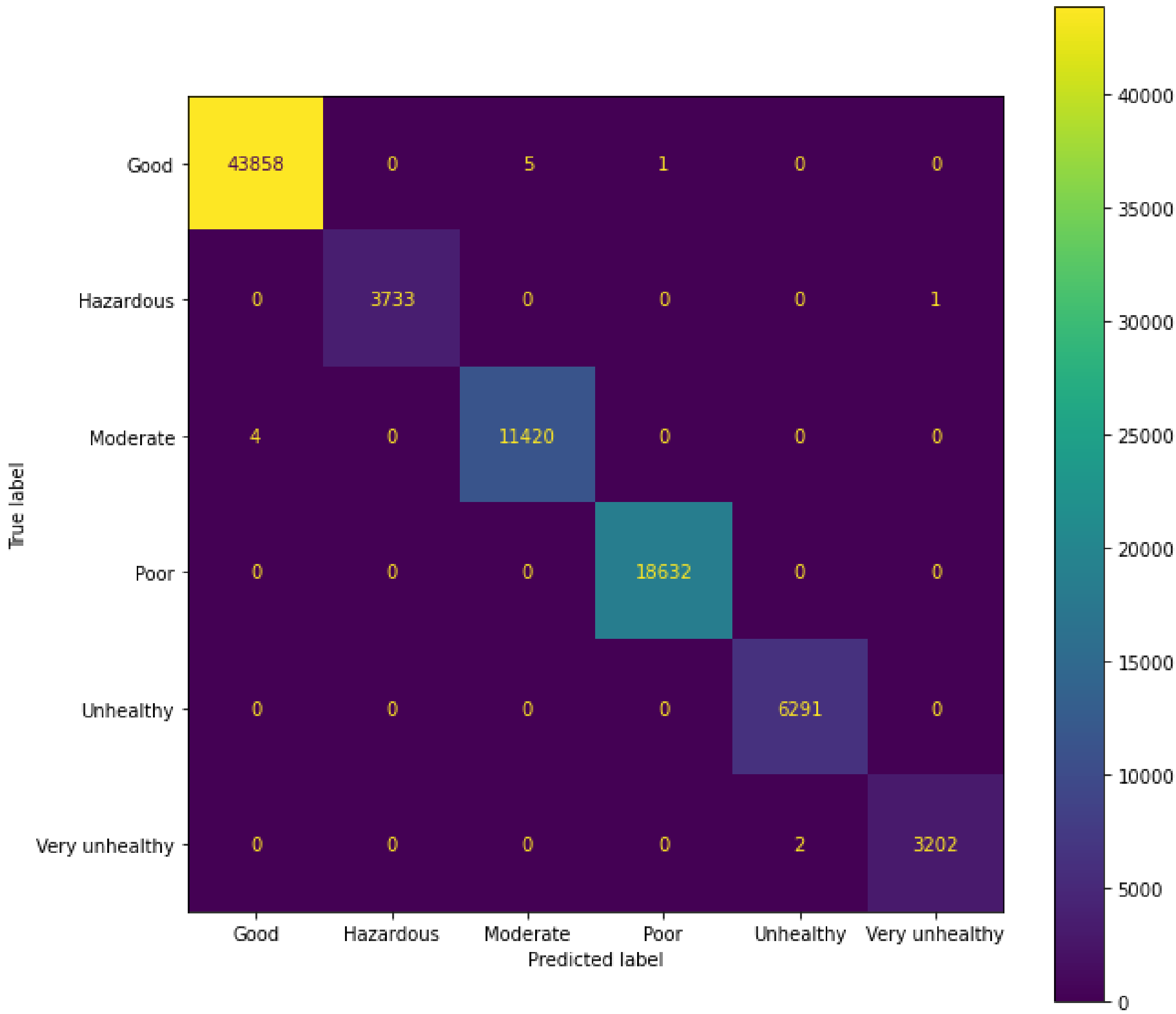
```
# Predictions on random values
RF.predict([[2.4,47.7,78.182,100]])
```

```
array(['Moderate'], dtype=object)
```

```
RF.predict([[2,45.8,37,32]])
```

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

```
from sklearn.metrics import plot_confusion_matrix
fig, ax = plt.subplots(figsize=(10, 10))
plot_confusion_matrix(RF, X_test2, Y_test2, ax=ax)
plt.show()
```



K-Nearest Neighbours

```
#fit the model on train data
KNN = KNeighborsClassifier(n_neighbors = 10).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:  0.9975042528105843
Model accuracy on test is:  0.9966953149204236
---------------------------------------------------
KappaScore is:  0.9951053795775416
```

```
KNN.predict([[727,327.55,78.2,100]])
```

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

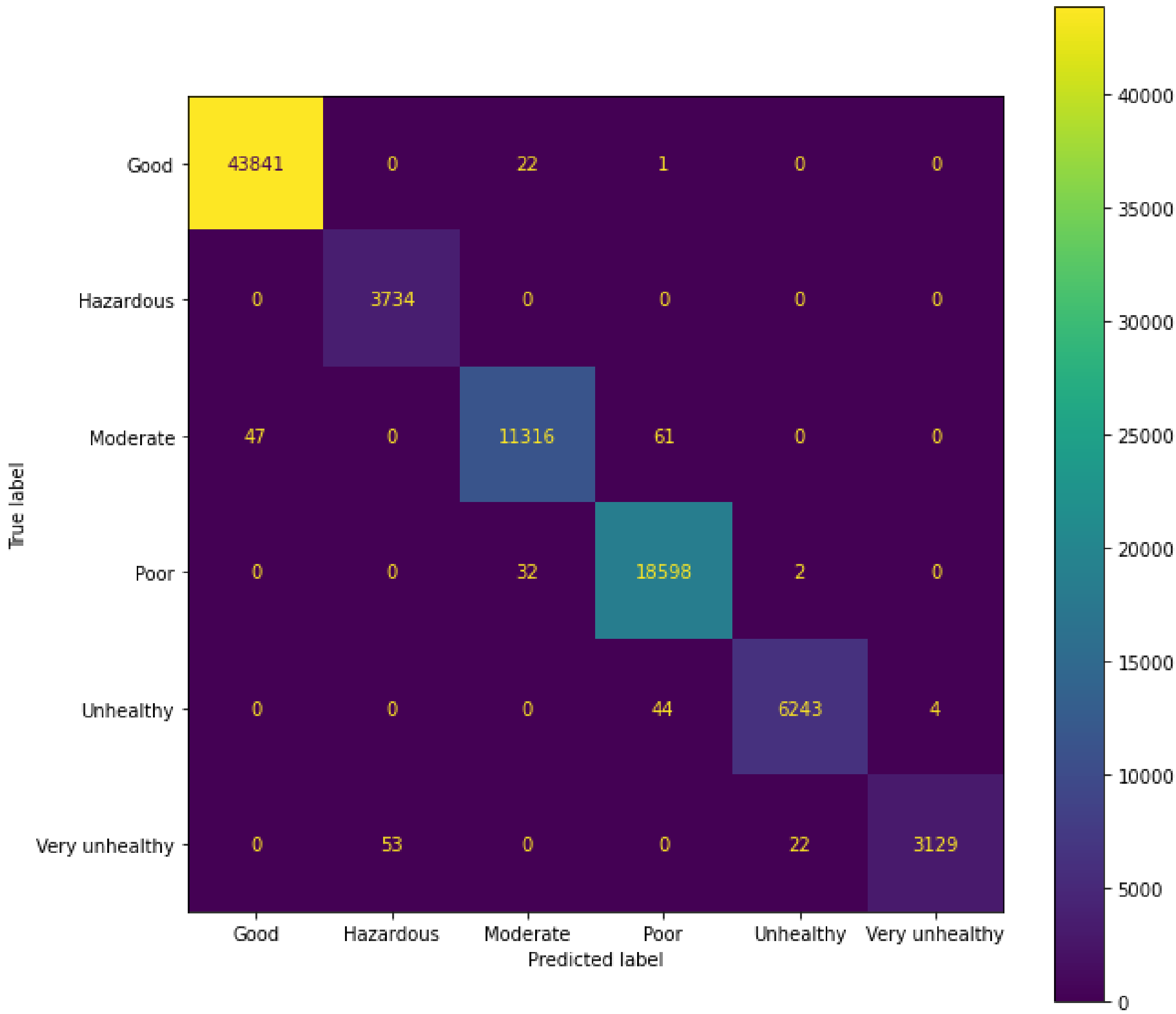
```
# Predictions on random values
KNN.predict([[2.4,47.7,78.182,100]])
```

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

```
KNN.predict([[2,45.8,37,32]])
```

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

```
from sklearn.metrics import plot_confusion_matrix
fig, ax = plt.subplots(figsize=(10, 10))
plot_confusion_matrix(KNN, X_test2, Y_test2, ax=ax)
plt.show()
```



Ridge Classifier

```
#fit the model on train data
ridge4 = RidgeClassifier(alpha = 1.0)
ridge4.fit(X_train2, Y_train2)

#predict on train
train_preds8 = ridge4.predict(X_train2)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train2,
train_preds8))

#predict on test
test_preds8 = ridge4.predict(X_test2)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test2,
test_preds8))
print('-'*50)
```

```
# Kappa Score
print('KappaScore is: ',
metrics.cohen_kappa_score(Y_test2,test_preds8))
```

```
Model accuracy on train is:  0.5626016586678447
Model accuracy on test is:  0.5622095491629279
---------------------------------------------------
KappaScore is:  0.22112609343373502
```

```
# Predictions on random values
ridge4.predict([[727,327.55,78.2,100]])
```

```
array(['Moderate'], dtype='<U14')
```

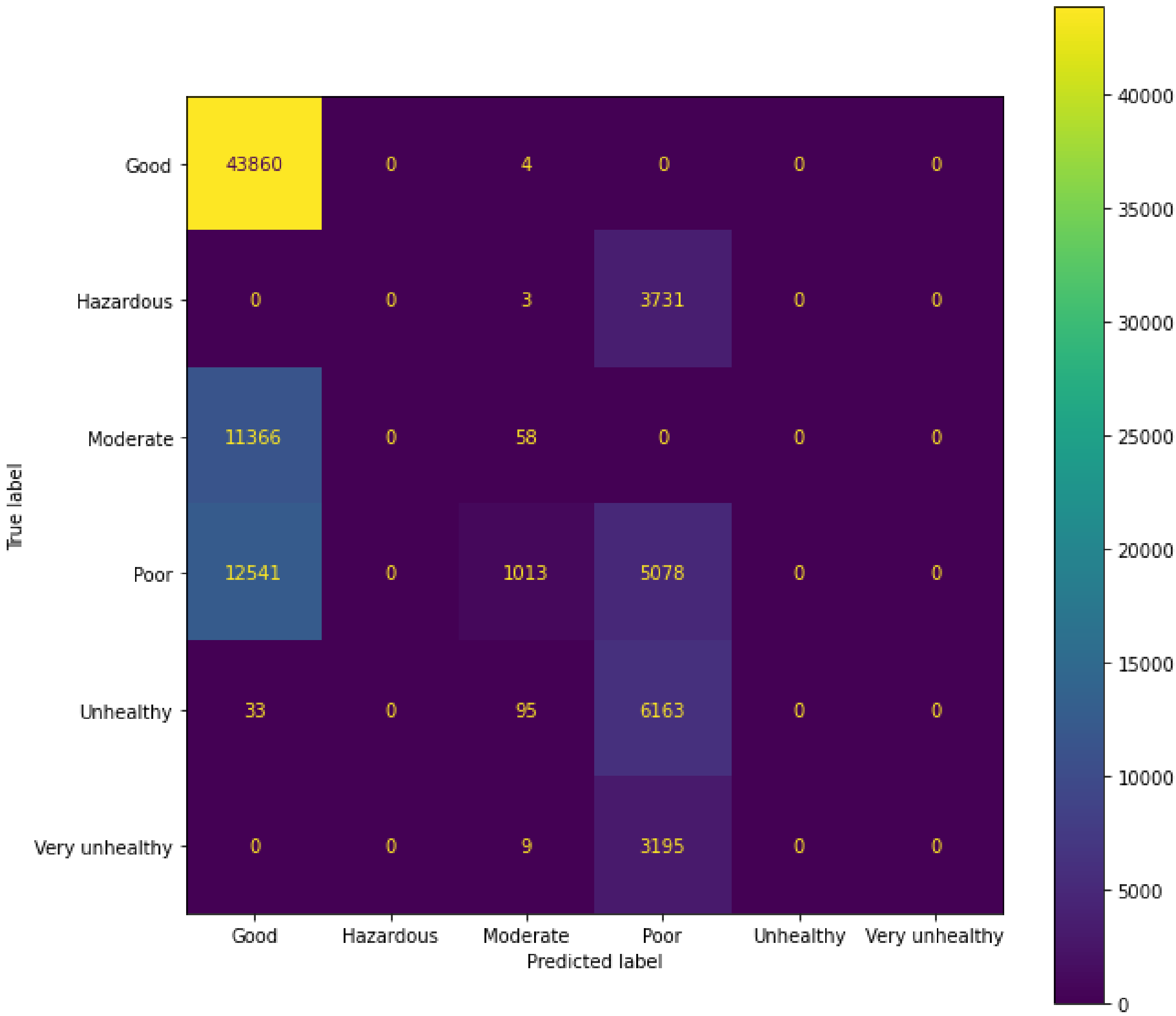
```
# Predictions on random values
ridge4.predict([[2.4,47.7,78.182,100]])
```

```
array(['Good'], dtype='<U14')
```

```
ridge4.predict([[2,45.8,37,32]])
```

```
array(['Good'], dtype='<U14')
```

```
from sklearn.metrics import plot_confusion_matrix
RidgeClassifier(random_state=40)
fig, ax = plt.subplots(figsize=(10, 10))
plot_confusion_matrix(ridge4, X_test2, Y_test2, ax=ax)
plt.show()
```



Good
Hazardous
Moderate
Poor
Unhealthy
Very unhealthy
True label
Predicted label
43860 0 4 0 0 0
0 0 3 3731 0 0
11366 0 58 0 0 0
12541 0 1013 5078 0 0
33 0 95 6163 0 0
0 0 9 3195 0 0
40000
35000
30000
25000
20000
15000
10000
5000
0