MAHATMA EDUCATION SOCIETY'S

PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE

(Autonomous)

NEW PANVEL

PROJECT REPORT ON

"AIR QUALITY INDEX PREDICTION ANALYSIS"

IN PARTIAL FULFILLMENT OF

MASTER OF DATA ANALYTICS

SEMESTER III 2024-25

PROJECT GUIDE

PROF. OMKAR SHERKHANE

SUBMITTED BY: ANAMIKA SARKAR

ROLL NO: 6904

Mahatma Education Society's
PILLAI COLLEGE OF ARTS, COMMERCE & SCIENCE (Autonomous)
Re-accredited "A" Grade by NAAC (3rd Cycle)

Project Completion Certificate

THIS IS TO CERTIFY THAT

ANAMIKA SARKAR

of **M.Sc. Data Analytics Part - II** has completed the project titled **AIR QUALITY INDEX PREDICTION ANALYSIS** of subject **MACHINE LEARNING** under our guidance and supervision during the academic year 2023-24 in the department of M.Sc. Data Analytics

Project Guide | Course Coordinator | Head of the Department

Pillai College of Arts, Commerce & Science (Autonomous)
New Panvel

INDEX

INTRODUCTION

Air quality is a vital aspect of environmental health, directly affecting human well-being, ecosystems, and climate. With rapid urbanization and industrialization, air pollution levels have been rising globally, making it essential to monitor and predict air quality to mitigate its adverse effects. The Air Quality Index (AQI) is a standardized system used to report daily air quality levels, indicating how clean or polluted the air is, along with its potential health impacts.

Traditionally, AQI is determined based on sensor data collected from various pollutants such as particulate matter (PM2.5 and PM10), nitrogen dioxide (NO2), sulfur dioxide (SO2), and ground-level ozone (O3). However, predicting future air quality levels is a complex task due to the dynamic nature of pollution sources and weather conditions.

To address this complexity, machine learning (ML) has emerged as a powerful tool for AQI prediction. ML models can learn from historical pollution and weather data to forecast future air quality levels with reasonable accuracy. By analyzing large datasets, machine learning algorithms can identify patterns, correlations, and trends that are not easily discernible through traditional statistical methods.

In parallel, web applications provide an accessible interface for users to interact with machine learning models. Using frameworks like Flask, machine learning predictions can be served on a user-friendly platform where users can input relevant data (e.g., meteorological conditions, pollution levels) and receive real-time predictions. Such an application not only helps individuals stay informed about air quality but also assists policymakers in making data-driven decisions for pollution control.

This project focuses on leveraging machine learning techniques for AQI prediction and deploying the model through a Flask-based web application. The combination of data science and web development facilitates user-friendly access to critical air quality information.

TOOLS AND TECHNIQUE USED

Machine Learning Techniques for AQI Prediction:

1. **Data Collection**: Data is collected from sensors or public datasets like weather data (temperature, humidity, wind speed) and pollutant levels (PM2.5, PM10, NO2, etc.).
    - Common sources include **Kaggle** or governmental databases like the **EPA**.
2. **Data Preprocessing**: Collected data is cleaned and processed to handle missing values, noise, and outliers. Feature scaling, encoding categorical variables, and normalizing data might also be necessary.
    - **Libraries**: Pandas, NumPy, Scikit-learn
3. **Feature Selection**: Choosing the most relevant features is essential for building efficient models. Techniques like correlation analysis or recursive feature elimination are commonly used.
    - **Libraries**: Scikit-learn, StatsModels
4. **Model Selection**: Various machine learning models can be applied to predict AQI, such as:
    - **Linear Regression**
5. **Web Application:** Flask Framework: Creating a simple web interface where users can input their annual income and spending score to predict their customer segment.

Model performance can be evaluated using metrics like **Mean Squared Error (MSE)** or **R-squared**.

CODES AND OUTPUTS

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 import metrics

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

from sklearn.metrics import accuracy_score,confusion_matrix

 # Reading the dataset

df=pd.read_csv('/content/data.csv',encoding='unicode_escape')

 # Loading the dataset

df.head()

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

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

df.shape

```
df.shape
(435742, 13)
```

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

Checking the over all information on the dataset.

df.info()

```
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
```

df.isnull().sum()

```
df.isnull().sum()
```

	0
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

There are a lot of missing values present in the dataset

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()

```
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

df.nunique()

```
df.nunique()
```

	0
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 unique values present in the dataframe

df.columns

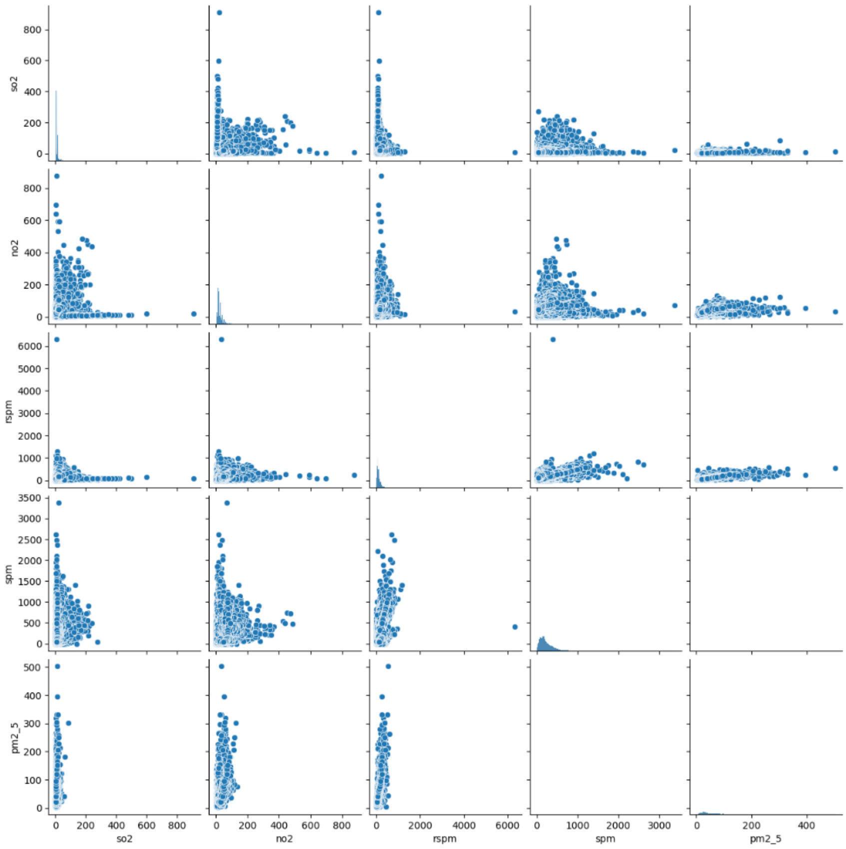
```
df.columns
```

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

These are all the columns present in the dataset.

Data Visualization

sns.pairplot(data=df)



df['state'].value_counts()

```
df['state'].value_counts()
```

state	count
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

Viewing the count of values present in the state column

plt.figure(figsize=(15, 6))

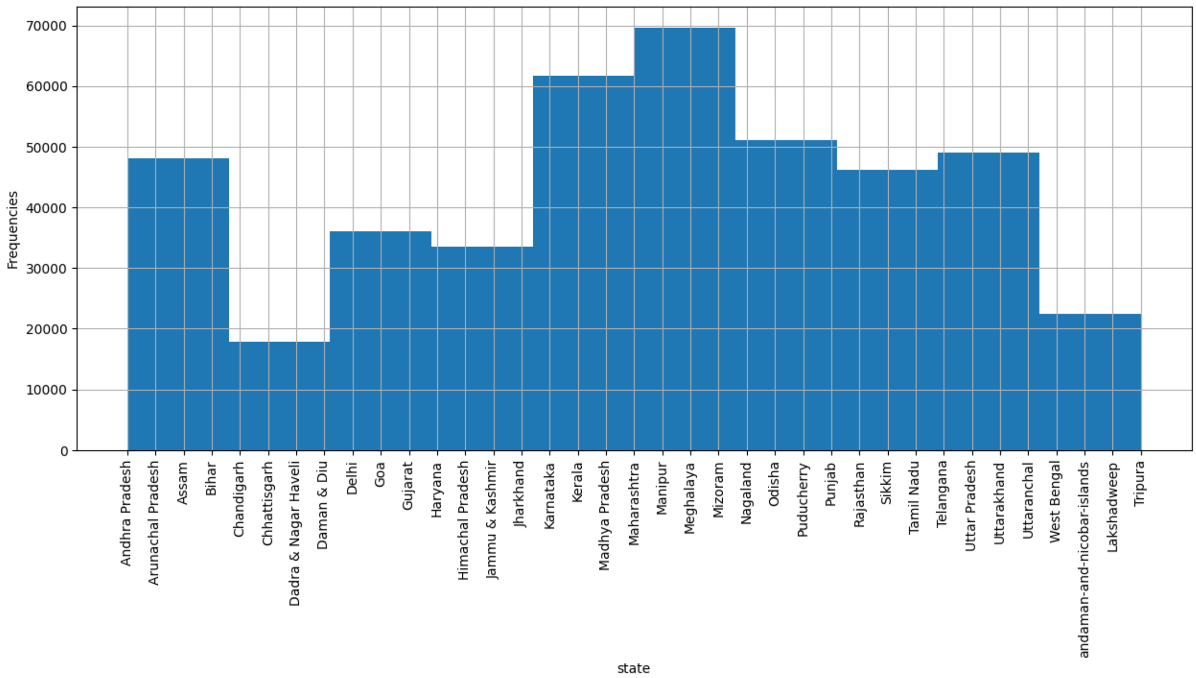
plt.xticks(rotation=90)

df.state.hist()

plt.xlabel('state')

plt.ylabel('Frequencies')

plt.plot()



The visualization shows us the count of states present in the dataset.

df['type'].value_counts()

```
df['type'].value_counts()
```

type	count
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

dtype: int64

plt.figure(figsize=(15, 6))

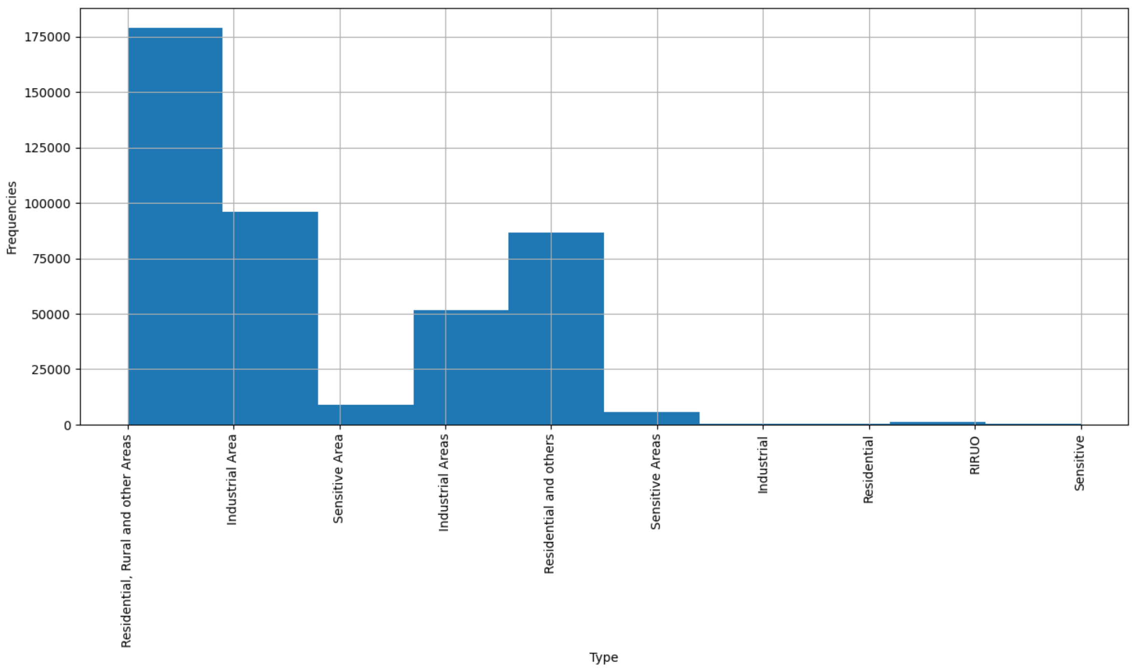
plt.xticks(rotation=90)

df.type.hist()

plt.xlabel('Type')

plt.ylabel('Frequencies')

plt.plot()



The visualization shows us the count of Types present in the dataset.

Viewing the counts of values present in the agency column

df['agency'].value_counts()

```
df['agency'].value_counts()
```

agency	count
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

64 rows × 1 columns

dtype: int64

plt.figure(figsize=(15, 6))

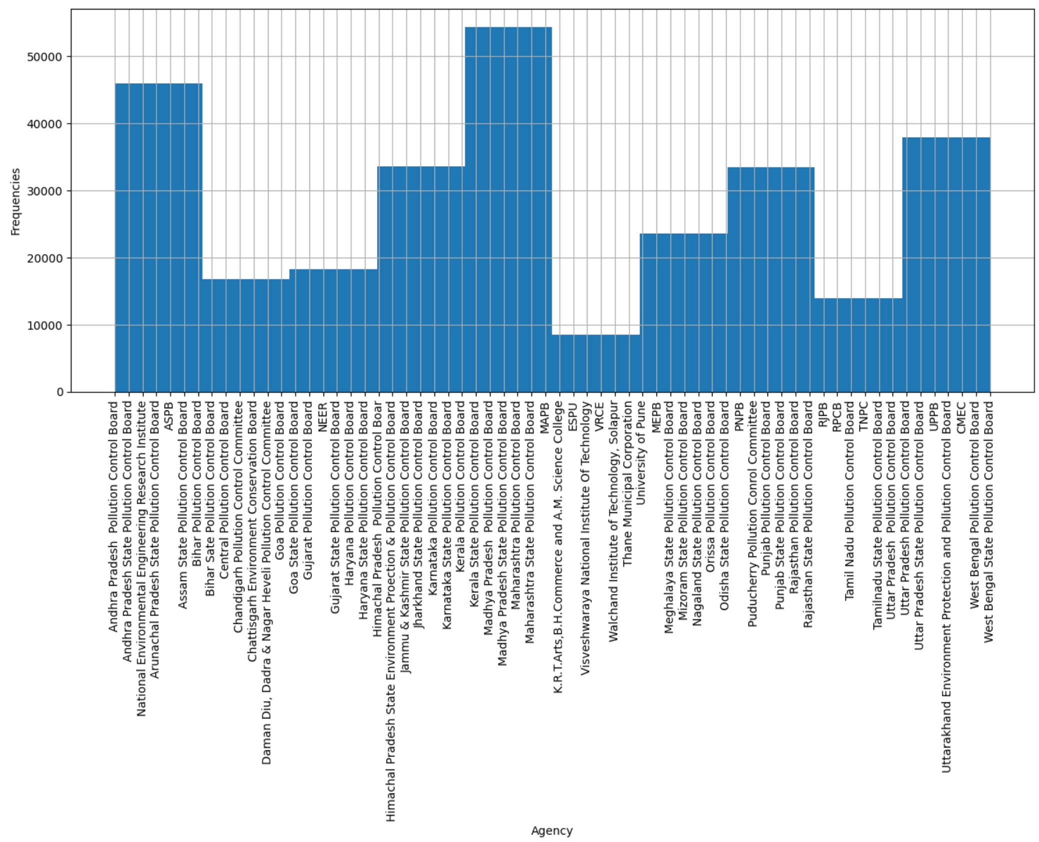
plt.xticks(rotation=90)

df.agency.hist()

plt.xlabel('Agency')

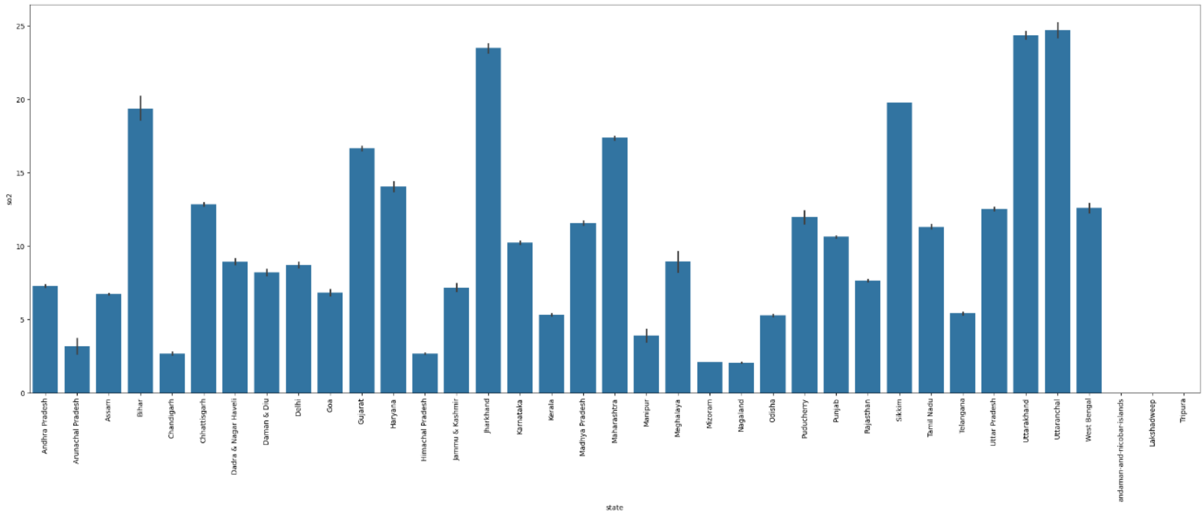
plt.ylabel('Frequencies')

plt.plot()



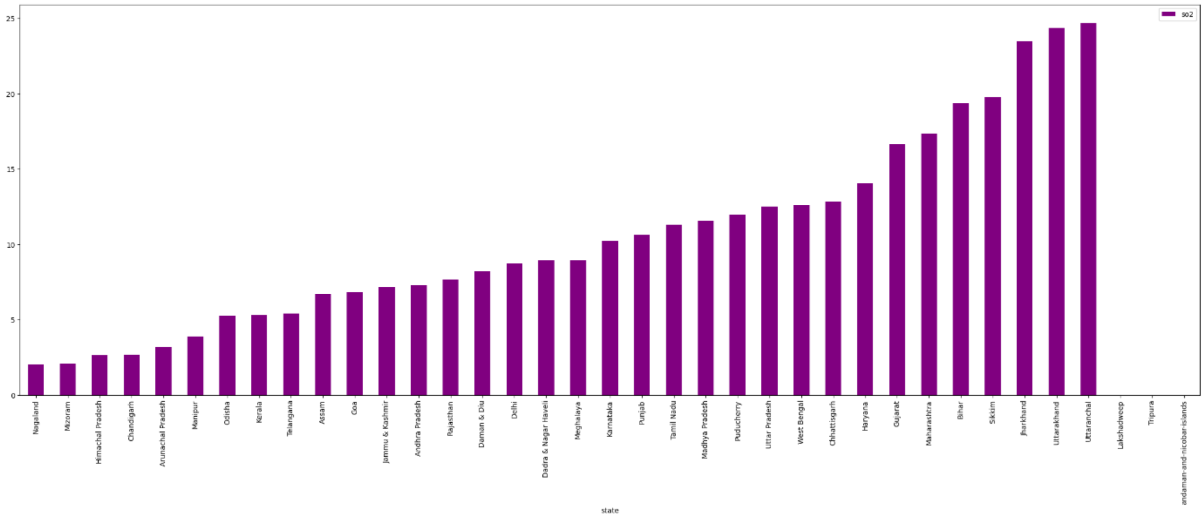
The visualization shows us the count of Agency present in the dataset.

```
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

```
plt.rcParams['figure.figsize']=(30,10)
df[['so2','state']].groupby(["state"]).mean().sort_values(by='so2').plot.bar(color='purple')
plt.show()
```

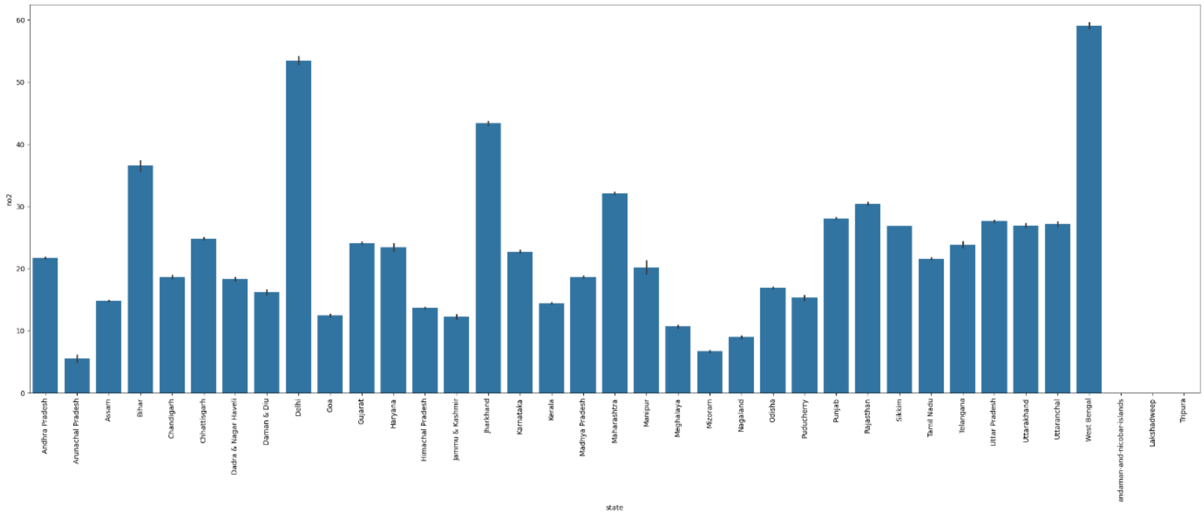


we get a clear picture of the states in an increasing order based on their so2 levels.

plt.figure(figsize=(30, 10))

plt.xticks(rotation=90)

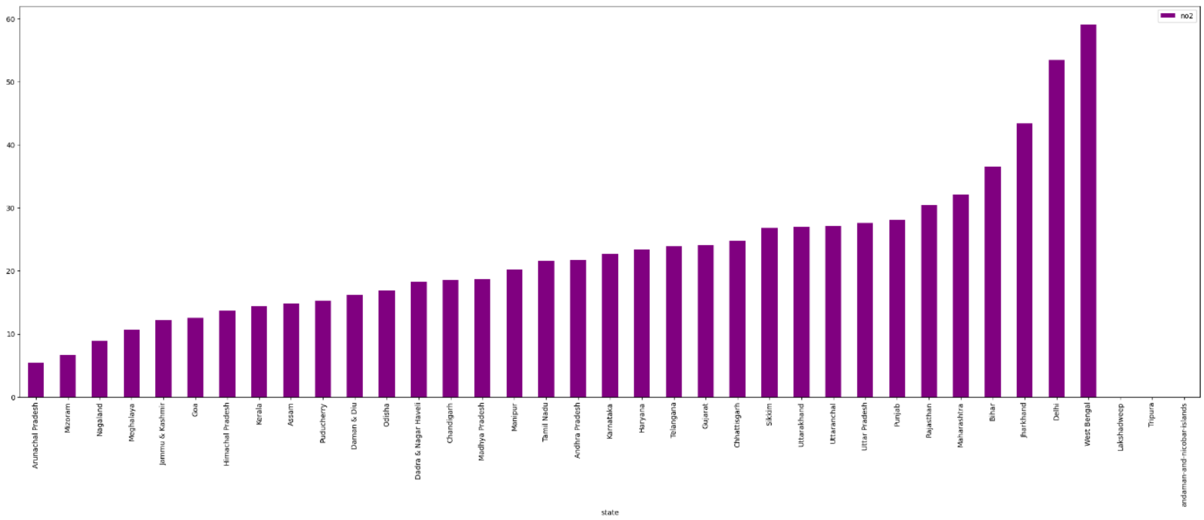
sns.barplot(x='state',y='no2',data=df);



West Bengal has a higher no2 level compared to other states.

df[['no2','state']].groupby(["state"]).mean().sort_values(by='no2').plot.bar(color='purple')

plt.show()

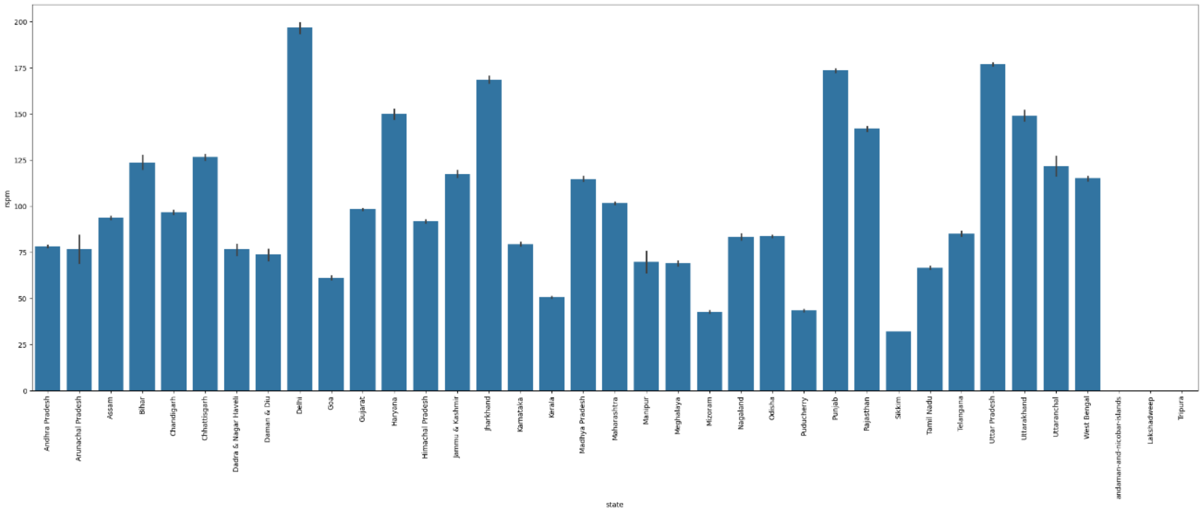


we get a clear picture of the states in an increasing order based on their no2 levels.

plt.figure(figsize=(30, 10))

plt.xticks(rotation=90)

sns.barplot(x='state',y='rspm',data=df);

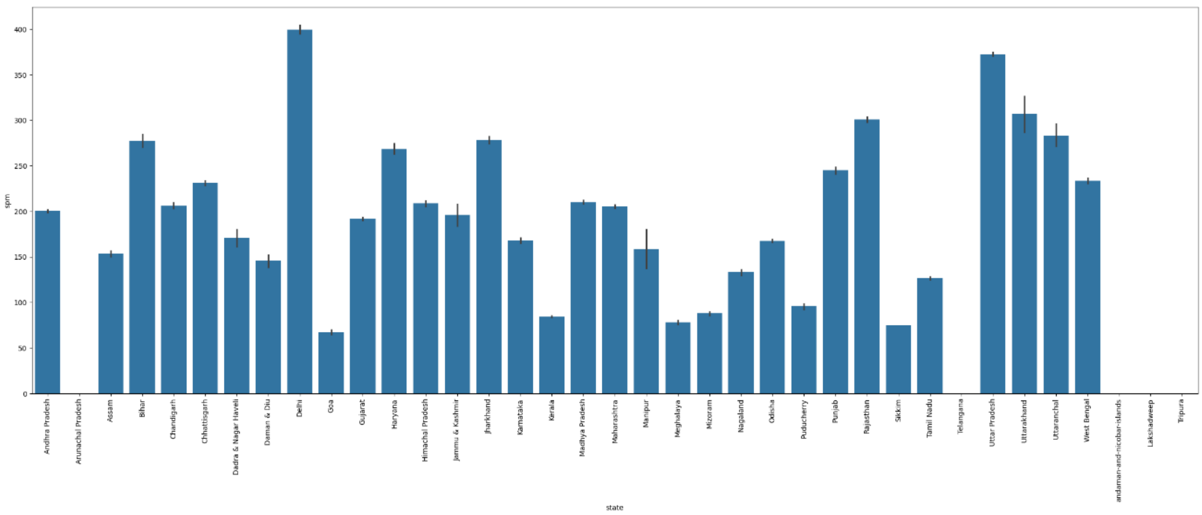


Delhi has higher rspm level compared to other states.

plt.figure(figsize=(30, 10))

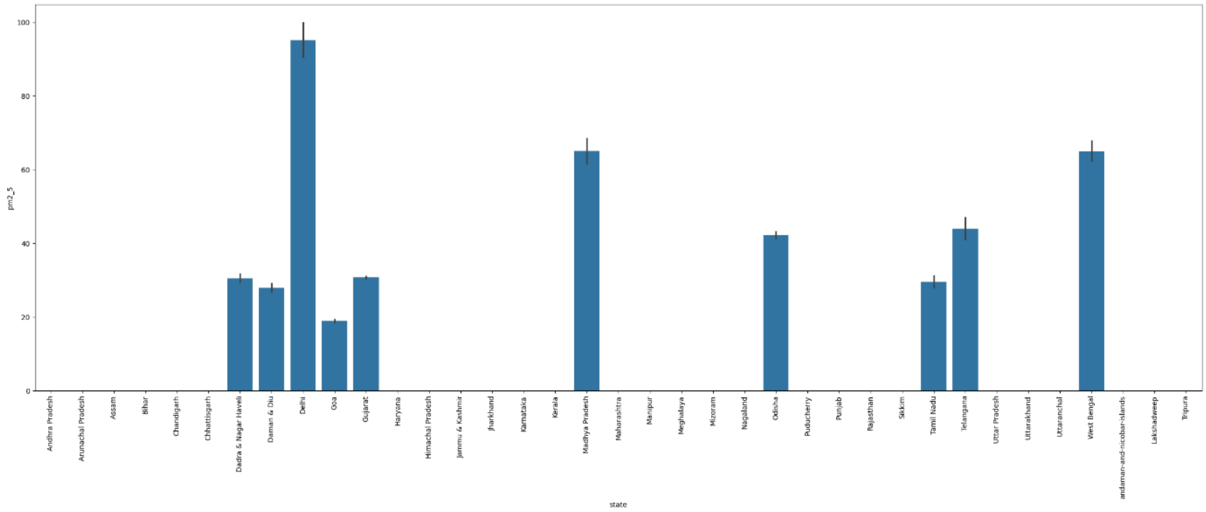
plt.xticks(rotation=90)

sns.barplot(x='state',y='spm',data=df);



Delhi has higher spm level compared to other states.

```
plt.figure(figsize=(30, 10))

plt.xticks(rotation=90)

sns.barplot(x='state',y='pm2_5',data=df);
```



Delhi has higher pm2_5 level compared to other states

Checking all null values and treating those null values.

```
nullvalues = df.isnull().sum().sort_values(ascending=False)

nullvalues
```

```
nullvalues
```

	0
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

higher null values present in pm2_5 followed by spm null_values_percentage = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)

missing_data_with_percentage = pd.concat([nullvalues, null_values_percentage], axis=1, keys=['Total', 'Percent'])

Concatenating total null values and their percentage of missing values for further imputation or column deletion

missing_data_with_percentage

```
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

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

Dropping unnecessary columns

Dropping unnecessary columns

```
[31] 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()

```
df.isnull().sum()
```

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

dtype: int64

df

```
df
```

	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

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)

df.isnull().sum()

```
df.isnull().sum()
```

	0
state	0
location	0
type	0
so2	0
no2	0
rspm	0
spm	0
pm2_5	0

dtype: int64

Now we have successfully imputed null values which were present in the dataset

```
df
```

	state	location	type	so2	no2	rspm	spm	pm2_5
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0.0
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	0.0	0.0	0.0
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0.0
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	0.0	0.0	0.0
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
435737	West Bengal	ULUBERIA	RIRUO	22.0	50.0	143.0	0.0	0.0
435738	West Bengal	ULUBERIA	RIRUO	20.0	46.0	171.0	0.0	0.0
435739	andaman-and-nicobar-islands	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0
435740	Lakshadweep	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0
435741	Tripura	Guwahati	Residential, Rural and other Areas	0.0	0.0	0.0	0.0	0.0

435742 rows × 8 columns

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)

```
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)

```
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)

```
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)

```
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

```
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

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()
```

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

	state	location	type	so2	no2	rspm	spm	pm2_5	SOi	Noi	Rpi	SPMi	AQI	AQI_Range
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0.0	6.000	21.750	0.0	0.0	21.750	Good
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	0.0	0.0	0.0	3.875	8.750	0.0	0.0	8.750	Good
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0.0	7.750	35.625	0.0	0.0	35.625	Good
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	0.0	0.0	0.0	7.875	18.375	0.0	0.0	18.375	Good
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	0.0	0.0	0.0	5.875	9.375	0.0	0.0	9.375	Good

```
df['AQI_Range'].value_counts()
```

```
df['AQI_Range'].value_counts()
```

AQI_Range	count
Good	219643
Poor	93272
Moderate	56571
Unhealthy	31733
Hazardous	18700
Very unhealthy	15823

dtype: int64

These are the counts of values present in the AQI_Range column.

Splitting the dataset into Dependent and Independent columns

X=df[['SOi','Noi','Rpi','SPMi']]

Y=df['AQI']

X.head()

```
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

we only select columns like soi, noi, rpi, spmi

Y.head()

```
Y.head()
```

	AQI
0	21.750
1	8.750
2	35.625
3	18.375
4	9.375

dtype: float64

the AQI column is the target column

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)

```
[47] 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,)
```

Linear Regression

Linear Regression

```
[48] model=LinearRegression()
     model.fit(X_train,Y_train)
```

```
▾ LinearRegression
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_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 =  13.672937344789002
--------------------------------------------------
RSquared value on train: 0.9849533579250526
RSquared value on test: 0.9847286394495923
```

```
[52] import pickle
```

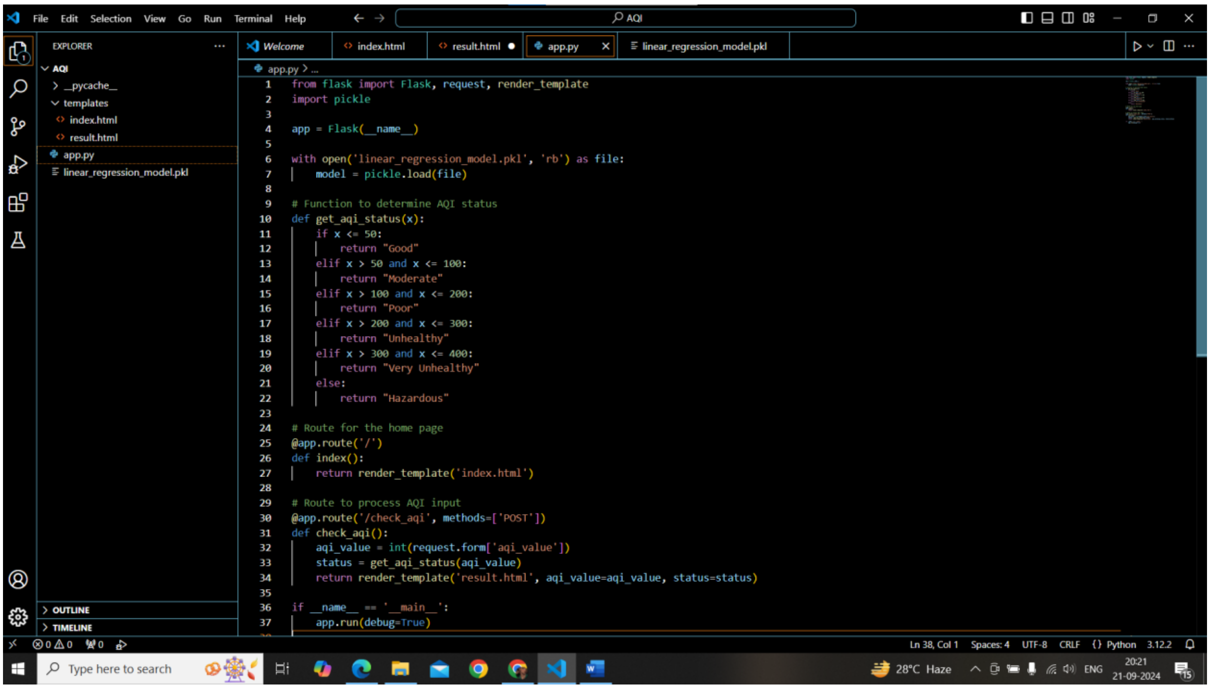
```
with open('linear_regression_model.pkl', 'wb') as file:
    pickle.dump(model, file)

print("Model saved successfully!")
```

```
Model saved successfully!
```

App.py

```
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

with open('linear_regression_model.pkl', 'rb') as file:
    model = pickle.load(file)

# Function to determine AQI status
def get_aqi_status(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"
    else:
        return "Hazardous"

# Route for the home page
@app.route('/')
def index():
    return render_template('index.html')

# Route to process AQI input
@app.route('/check_aqi', methods=['POST'])
```

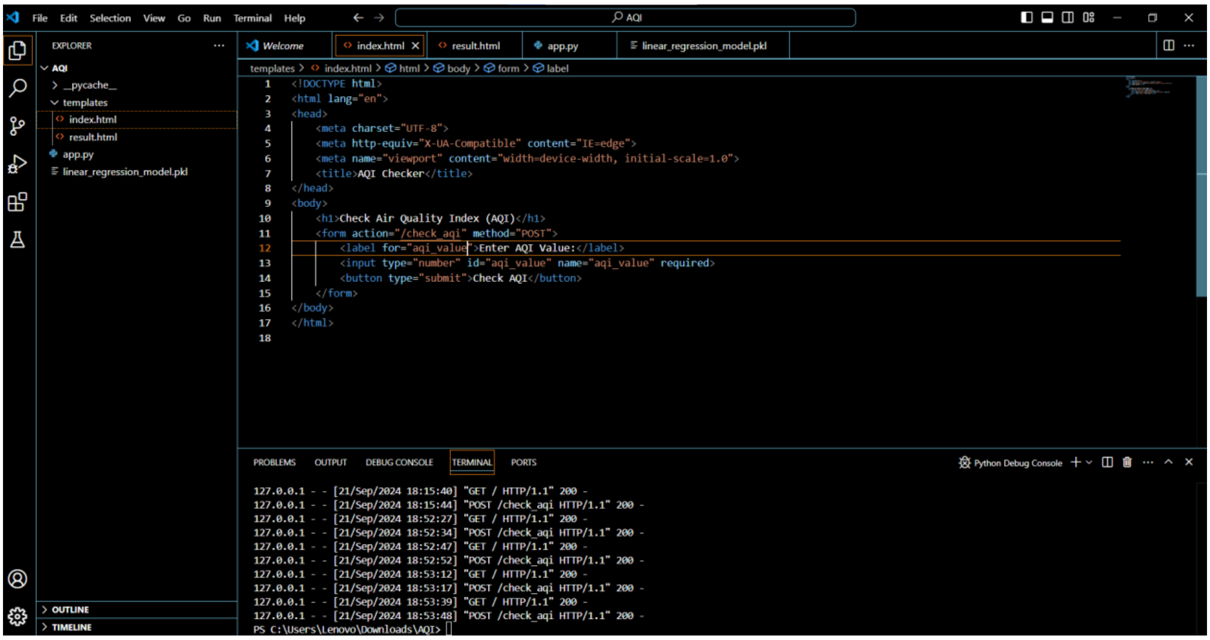
```
def check_aqi():
    aqi_value = int(request.form['aqi_value'])
    status = get_aqi_status(aqi_value)
    return render_template('result.html', aqi_value=aqi_value, status=status)


if __name__ == '__main__':
    app.run(debug=True)
```



index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AQI Checker</title>
</head>
<body>
    <h1>Check Air Quality Index (AQI)</h1>
    <form action="/check_aqi" method="POST">
```

```
        <label for="aqi_value">Enter AQI Value:</label>

        <input type="number" id="aqi_value" name="aqi_value" required>

        <button type="submit">Check AQI</button>

    </form>

</body>

</html>
```

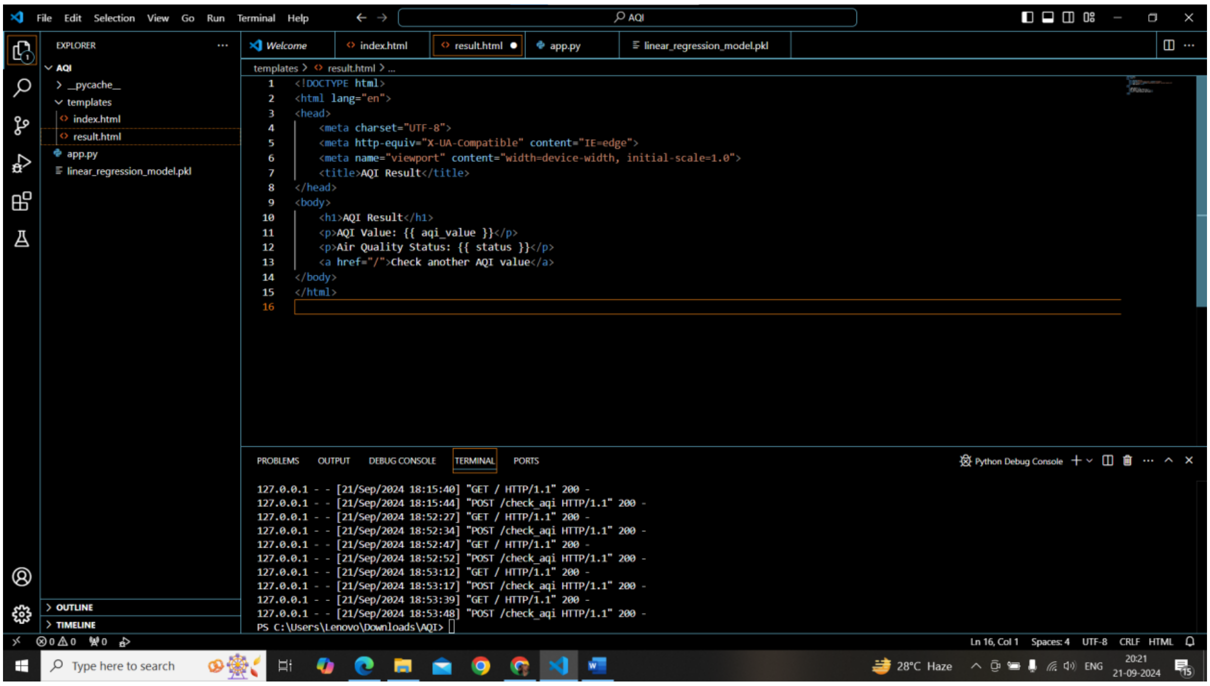


Result.html

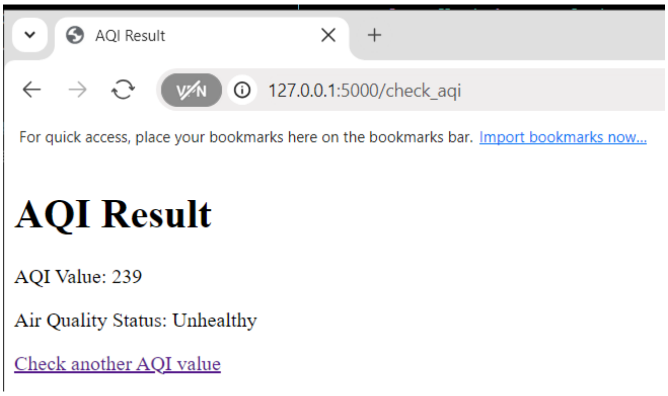
```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>AQI Result</title>

</head>

<body>

    <h1>AQI Result</h1>

    <p>AQI Value: {{ aqi_value }}</p>

    <p>Air Quality Status: {{ status }}</p>

    <a href="/">Check another AQI value</a>
```

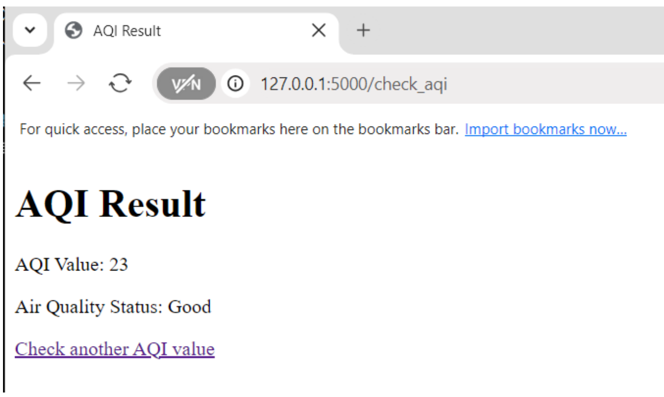
</body>

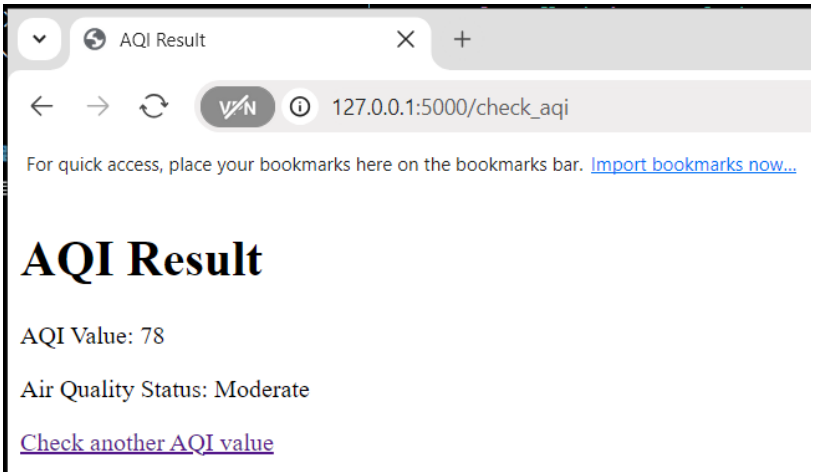
</html>

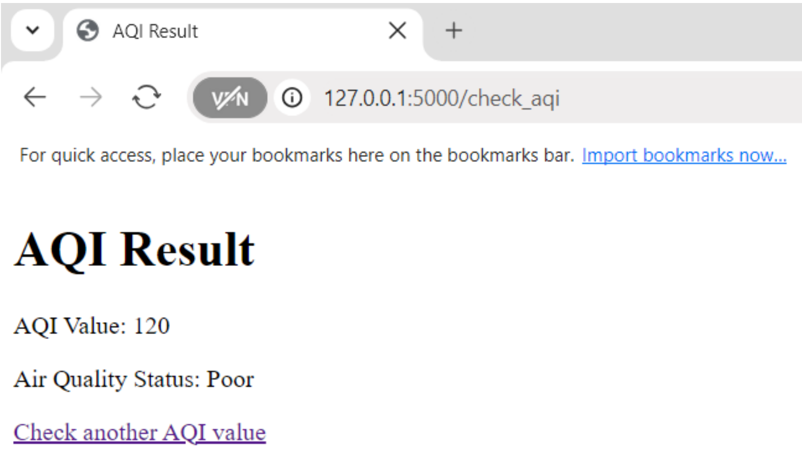


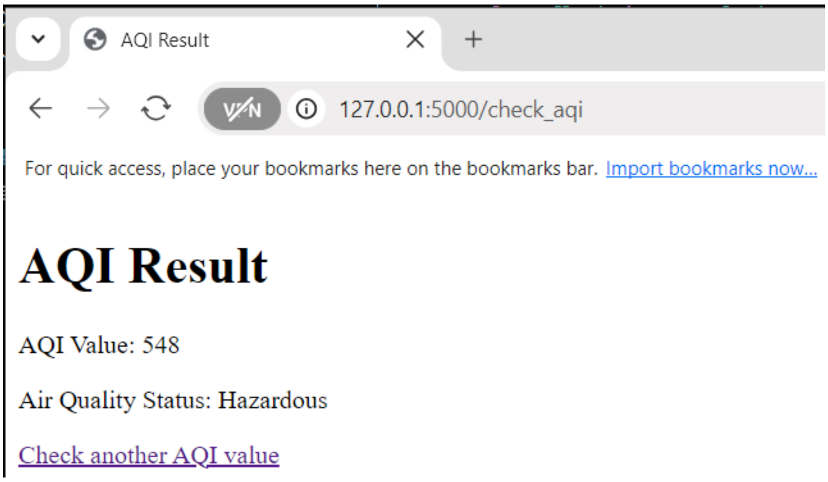
OUTPUT






AQI Result
127.0.0.1:5000/check_aqi
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
AQI Result
AQI Value: 78
Air Quality Status: Moderate
Check another AQI value


AQI Result
127.0.0.1:5000/check_aqi
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
AQI Result
AQI Value: 120
Air Quality Status: Poor
Check another AQI value


AQI Result
127.0.0.1:5000/check_aqi
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
AQI Result
AQI Value: 548
Air Quality Status: Hazardous
Check another AQI value

CONCLUSION

Predicting air quality using machine learning presents a significant opportunity to enhance environmental monitoring and public health protection. Through the application of advanced algorithms, we can process large volumes of historical and real-time data to generate accurate AQI forecasts. These predictions can help governments, industries, and individuals make informed decisions to reduce exposure to harmful pollutants and take necessary precautions during periods of poor air quality.

Furthermore, by deploying these machine learning models in a Flask web application, we bridge the gap between complex data science models and end-users. A user-friendly interface ensures that individuals without technical expertise can benefit from real-time air quality predictions, enabling them to take proactive measures to safeguard their health.

The integration of AQI prediction into accessible web platforms opens up possibilities for innovation in air quality monitoring systems. With the growing availability of data from IoT sensors and advancements in machine learning, this approach can evolve further to provide more granular and localized predictions. In turn, such applications can play a pivotal role in environmental management strategies, enabling communities and organizations to work towards cleaner, healthier air.

In conclusion, the synergy between machine learning and web development creates a powerful tool for improving air quality awareness and public health interventions. By making AQI predictions more accessible and actionable, this approach contributes to a future where data-driven decisions are instrumental in tackling air pollution and promoting sustainable living.