Air Quality Index for Urban Areas

Panchajanya Sarkar [2021MSBDA028]

Central University of Rajasthan
August 2023

Overview

Air Quality Index for Urban Areas

- Introduction
- Problem Statement
- Objectives
- Methodology
- Exploratory Data Analysis
- Model Development
- Web Dashboard
- Result and Analysis

Illustrations by Pixeltrue on icons8

Introduction

Air Quality Importance

Air Quality profoundly affects human health and the environment. The air we breathe affects our well-being and monitoring air quality becomes crucial.

→ **Health Impact:**
Poor air quality leads to respiratory and cardiovascular problems. Particulate Matter and Ozone harm vulnerable groups.

→ **Environmental Effects:**
Air Pollution causes smog, acid rain and climate change. Ecosystems suffer, disrupting natural balances.

→ **Urban Challenges:**
Urban areas face heightened pollution due to vehicles and industries.

Introduction

Motivation

→ Rapid urbanization and industrial growth have led to an urgent need for real-time air quality information.

→ Poor air quality poses serious health risks, particularly in densely populated areas. The lack of immediate data hinders informers decision making for residents and policymakers alike.

→ This project aims t bridge this gap by providing a user-friendly web dashboard that offers a predicted Air Quality Index value by predicting from the values provided by the user with an accuracy of ~96% .

Problem Statement

Challenges Faced

→ **Densely Populated Areas**

Urban regions host high population densities, increasing pollution sources and complexity

→ **Dynamic Environment**

Air Quality changes rapidly due to traffic. Industries and weather conditions, demanding real time predictions.

→ **Limited Coverage**

Conventional Static monitoring stations offer limited coverage and may not capture local capture variations.

→ **Public Health Impact**

Lack of immediate data affects health decisions, especially for vulnerable individuals.

Objectives

Key Objectives

The key objectives to describe the intended results and impacts of the efforts -

→ **A Real-Time Web dashboard**
A user-friendly web dashboard to display the predicted AQI value.
→ **Empower Residents**
Provide Information to individuals who wants a future plan by AQI values
→ **Urban Planning**
Equip policymakers with near-accurate data to guide urban development, transportation etc.
→ **Bridge Information Gap**
Address the lack of AQI information and promote data-driven decisions.

Methodology

Data Collection

→ Data Source is obtained from a reputable source, Kaggle in this instance.

→ The data source is read using pandas and ISO-8859-1 encoding. A screenshot of data reading is shown below:

```
# read the data from the csv file located inside datasets folder
df = pd.read_csv('datasets/data.csv', encoding = "ISO-8859-1", low_memory=False)
```

→ 'df' is a variable which stores the dataframe.

Methodology

Data Preprocessing

Data Viewing

→ The data is first viewed. To view the data we either check the *head* or the *tail*. By deafult, *head* returns the top 5 rows of a data frame.

```
df.head()
```

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date
0	150	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	NaN	1990-02-01
1	151	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	3.1	7.0	NaN	NaN	NaN	NaN	1990-02-01
2	152	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	NaN	1990-02-01
3	150	March - M031990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	NaN	1990-03-01
4	151	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.7	7.5	NaN	NaN	NaN	NaN	1990-03-01

Methodology

Data Preprocessing

Checking Shape of the dataframe

→ The shape of the dataframe is figured by *shape* function.

```python
# check the shape of the dataframe
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns in the dataframe")
```

```
There are 435742 rows and 13 columns in the dataframe
```

→ We can see fromhere that the dataframe holds 435742 rows and 13 columns.

Methodology

Data Preprocessing

Dealing with NULL/missing values

→ NULL values or missing values appear whenever a data entry is missed or something subsequent mishaps.

```
# check for null values in descending order
df.isnull().sum().sort_values(ascending=False)

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
```

```
# check the basic info of the dataframe
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
```

Methodology

Data Preprocessing

Dealing with NULL/missing values

→ To fill the NULL values, we will fill the NULL values with **Median** values of each column.

→ The purpose of using **Median** instead of **Mean** is dealing with **Outliers.** Since **Median** is less sensitive to extreme values and provides a more robust measure of Central Tendency.

→ We should not remove the outliers from the data since we have no physical idea what exactly was the original value when compared to the read value. Also it affects accuracy in some cases.

Methodology

Data Preprocessing

Dealing with NULL/missing values

→ The NULL values are replaced with **median** values.

```python
# for the float64 columns, fill the null values with the mean of the column
# make a list of the float64 columns
float64_cols = df.select_dtypes(include=['float64']).columns.tolist()
# find the mean and median of each column
mean = df[float64_cols].mean()
#print(mean)
median = df[float64_cols].median()
#print(median)

# fill the null values with the median of the column
df[float64_cols] = df[float64_cols].fillna(median)

# check for null values in descending order
df.isnull().sum().sort_values(ascending=False)
```

```
agency                        149481
stn_code                      144077
location_monitoring_station    27491
type                            5393
date                               7
sampling_date                      3
location                           3
state                              0
so2                                0
no2                                0
rspm                               0
spm                                0
pm2_5                              0
dtype: int64
```

Methodology

Data Preprocessing

Dealing with duplicate rows

→ Duplicate rows has no effect overall and it is a toll to memory usage. Dropping the duplicate rows is essential to make the dataframe smaller and reduce memory usage.
→ Uniqueness of the dataframe remains unchanged after dropping duplicate rows.

```
# check for duplicate rows in the dataframe
print(df.duplicated().sum())

# drop the duplicate rows
df.drop_duplicates(inplace=True)

# check rows and columns after dropping the duplicates
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns in the dataframe")

# are the unique values in the datraframe got affected?
df.nunique()

# they are same as before, so we can proceed
```

```
674
There are 435068 rows and 13 columns in the dataframe

stn_code                      745
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
```

Methodology

Data Preprocessing

Dealing with optional columns

→ Optional columns exists in the dataframe but it has no purpose in the prediction part. Dropping the column would save us memory and computing time.

```
# drop the columns which are not required
# create a list of columns to be dropped
cols_to_drop = ['agency', 'stn_code', 'date', 'sampling_date', 'location_monitoring_station']

# drop the columns
df.drop(columns=cols_to_drop, inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 435068 entries, 0 to 435741
Data columns (total 8 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   state     435068 non-null  object
 1   location  435065 non-null  object
 2   type      429711 non-null  object
 3   so2       435068 non-null  float64
 4   no2       435068 non-null  float64
 5   rspm      435068 non-null  float64
 6   spm       435068 non-null  float64
 7   pm2_5     435068 non-null  float64
dtypes: float64(5), object(3)
memory usage: 29.9+ MB
```

Methodology

Feature Engineering

Creating SO2 Individual Index

→ The purpose of creating SO2 Index is to assess whether it is exceeding acceptale levels and posing potential health risks.

```python
# calculate the so2 individual index (si)
def cal_si(so2):
    si = 0
    if (so2 <= 40):
        si = so2 * 50 / 40
    elif (so2 <= 80):
        si = 50 + (so2 - 40) * 50 / 40
    elif (so2 <= 380):
        si = 100 + (so2 - 80) * 100 / 300
    elif (so2 <= 800):
        si = 200 + (so2 - 380) * 100 / 420
    elif (so2 <= 1600):
        si = 300 + (so2 - 800) * 100 / 800
    else:
        si = 400 + (so2 - 1600) * 100 / 800
    return si

# apply the function to calculate si
df['si'] = df['so2'].apply(cal_si)
```

Here's how the SI levels correlate with air quality:

- **SI <= 50:** Acceptable - Good air quality.
- **50 < SI <= 100:** Moderate - Moderate air quality, posing a slight health concern for a very small number of people who are unusually sensitive to air pollution.
- **100 < SI <= 200:** Unhealthy for Sensitive Groups - People with respiratory or heart conditions, children, and older adults are at a greater risk. The general public is not likely to be affected.
- **200 < SI <= 300:** Unhealthy - Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
- **300 < SI <= 400:** Very Unhealthy - Health alert: everyone may experience more serious health effects.
- **SI > 400:** Hazardous - Health warnings of emergency conditions. The entire population is more likely to be affected.

Methodology

Feature Engineering

Creating NO2 Individual Index

→ The purpose of creating NO2 Index is to assess whether it is exceeding acceptale levels and posing potential health risks.

```python
# calculate the no2 individual index (ni)
def cal_ni(no2):
    ni = 0
    if (no2 <= 40):
        ni = no2 * 50 / 40
    elif (no2 <= 80):
        ni = 50 + (no2 - 40) * 50 / 40
    elif (no2 <= 180):
        ni = 100 + (no2 - 80) * 100 / 100
    elif (no2 <= 280):
        ni = 200 + (no2 - 180) * 100 / 100
    elif (no2 <= 400):
        ni = 300 + (no2 - 280) * 100 / 120
    else:
        ni = 400 + (no2 - 400) * 100 / 120
    return ni

# apply the function to calculate ni
df['ni'] = df['no2'].apply(cal_ni)
```

Here's how the NI levels correlate with air quality:

- **NI <= 50:** Acceptable - Good air quality.
- **50 < NI <= 100:** Moderate - Moderate air quality, posing a slight health concern for a very small number of people who are unusually sensitive to air pollution.
- **100 < NI <= 200:** Unhealthy for Sensitive Groups - People with respiratory or heart conditions, children, and older adults are at a greater risk. The general public is not likely to be affected.
- **200 < NI <= 300:** Unhealthy - Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
- **300 < NI <= 400:** Very Unhealthy - Health alert: everyone may experience more serious health effects.
- **NI > 400:** Hazardous - Health warnings of emergency conditions. The entire population is more likely to be affected.

Methodology

Feature Engineering

Creating RSPM Individual Index

→ The purpose of creating RSPM Index is to assess whether it is exceeding acceptale levels and posing potential health risks.

```python
# calculate the rspm individual index (rpi)
def cal_rpi(rspm):
    rpi = 0
    if (rspm <= 30):
        rpi = rspm * 50 / 30
    elif (rspm <= 60):
        rpi = 50 + (rspm - 30) * 50 / 30
    elif (rspm <= 90):
        rpi = 100 + (rspm - 60) * 100 / 30
    elif (rspm <= 120):
        rpi = 200 + (rspm - 90) * 100 / 30
    elif (rspm <= 250):
        rpi = 300 + (rspm - 120) * 100 / 130
    else:
        rpi = 400 + (rspm - 250) * 100 / 130
    return rpi

# apply the function to calculate rpi
df['rpi'] = df['rspm'].apply(cal_rpi)
```

[28] ✓ 0.3s

Here's how the RPI levels correlate with air quality:

- **RPI <= 50:** Acceptable - Good air quality.
- **50 < RPI <= 100:** Moderate - Moderate air quality, posing a slight health concern for a very small number of people who are unusually sensitive to air pollution.
- **100 < RPI <= 200:** Poor - People with respiratory or heart conditions, children, and older adults are at a greater risk. The general public is not likely to be affected.
- **200 < RPI <= 300:** Very Poor - Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
- **300 < RPI <= 400:** Severe - Health alert: everyone may experience more serious health effects.
- **RPI > 400:** Very Severe - Health warnings of emergency conditions. The entire population is more likely to be affected.

Methodology

Feature Engineering

Creating SPM Individual Index

→ The purpose of creating SPM Index is to assess whether it is exceeding acceptale levels and posing potential health risks.

```python
# calculate the spm individual index (spi)
def cal_spi(spm):
    spi = 0
    if (spm <= 50):
        spi = spm
    elif (spm <= 100):
        spi = spm
    elif (spm <= 250):
        spi = 100 + (spm - 100) * 100 / 150
    elif (spm <= 350):
        spi = 200 + (spm - 250)
    elif (spm <= 450):
        spi = 300 + (spm - 350) * 100 / 100
    else:
        spi = 400 + (spm - 450) * 100 / 100
    return spi

# apply the function to calculate spi
df['spi'] = df['spm'].apply(cal_spi)
```

[29] ✓ 0.3s

Here's how the SPI levels correlate with air quality:

- **SPI <= 50:** Acceptable - Good air quality.
- **50 < SPI <= 100:** Acceptable - Satisfactory air quality.
- **100 < SPI <= 200:** Moderate - Moderate air quality, posing a slight health concern for a very small number of people who are unusually sensitive to air pollution.
- **200 < SPI <= 300:** Poor - Poor air quality, unhealthy for sensitive groups.
- **300 < SPI <= 400:** Very Poor - Very poor air quality, everyone may begin to experience health effects.
- **SPI > 400:** Severe - Health warnings of emergency conditions, the entire population is more likely to be affected.

Methodology

Feature Engineering

Creating Air Quality Individual Index

→ The purpose of creating AQII is to mathematically calculate the values from other attributes and use it to train and test models.

```python
# calculate the aqi index
def cal_aqi(si, ni, rpi, spi):
    aqi = 0
    if (si > ni and si > rpi and si > spi):
        aqi = si
    elif (ni > si and ni > rpi and ni > spi):
        aqi = ni
    elif (rpi > si and rpi > ni and rpi > spi):
        aqi = rpi
    else:
        aqi = spi
    return aqi

# apply the function to calculate aqi
df['AQI'] = df.apply(lambda x: cal_aqi(x['si'], x['ni'], x['rpi'], x['spi']), axis=1)
```

[30] ✓ 5.6s

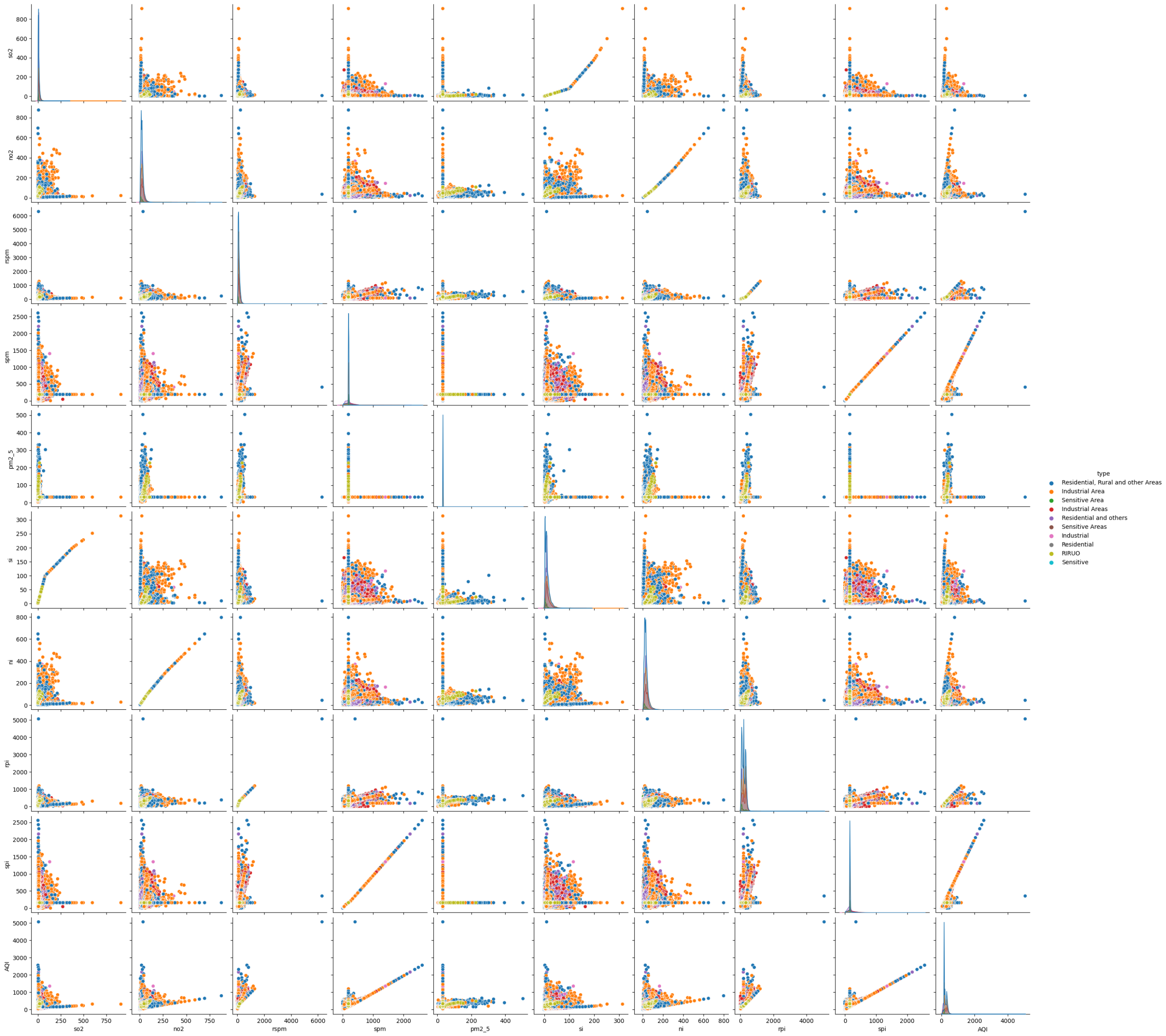
Exploratory Data Analysis

Data Visualization

Pairplot against *type* column

→ pairpot provides an insight into both relationship between variables and how those relationships differ across different categories.

→ The image is shown on the next slide.



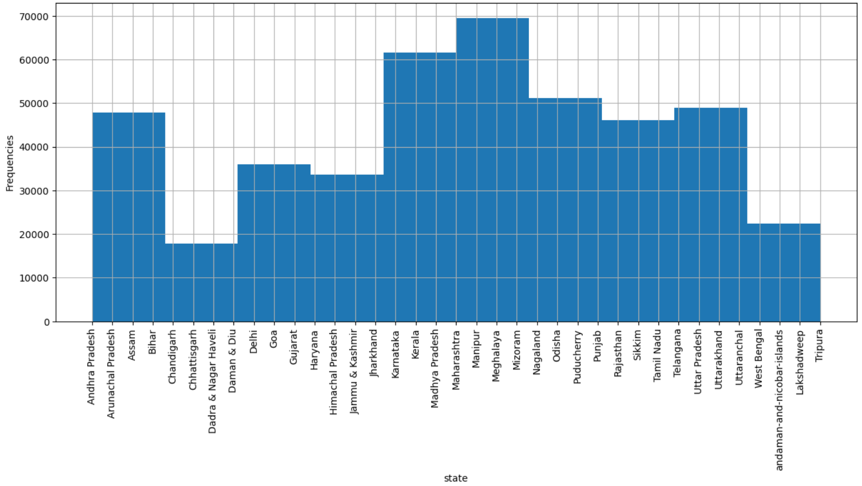
so2
no2
rspm
spm
pm2_5
si
ni
rpi
spi
AQI
type
Residential, Rural and other Areas
Industrial Area
Sensitive Area
Industrial Areas
Residential and others
Sensitive Areas
Industrial
Residential
RIRUO
Sensitive

Exploratory Data Analysis

Data Visualization

Frequency of Reading from each state

→ This visualizes the number of readings collected from each state.

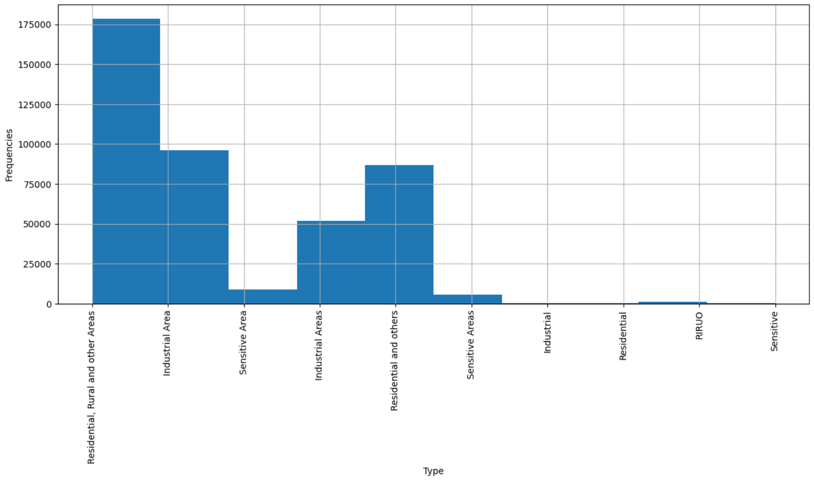


Exploratory Data Analysis

Data Visualization

Count of *types* present in the dataset

→ This visualizes the count of types in the dataset..

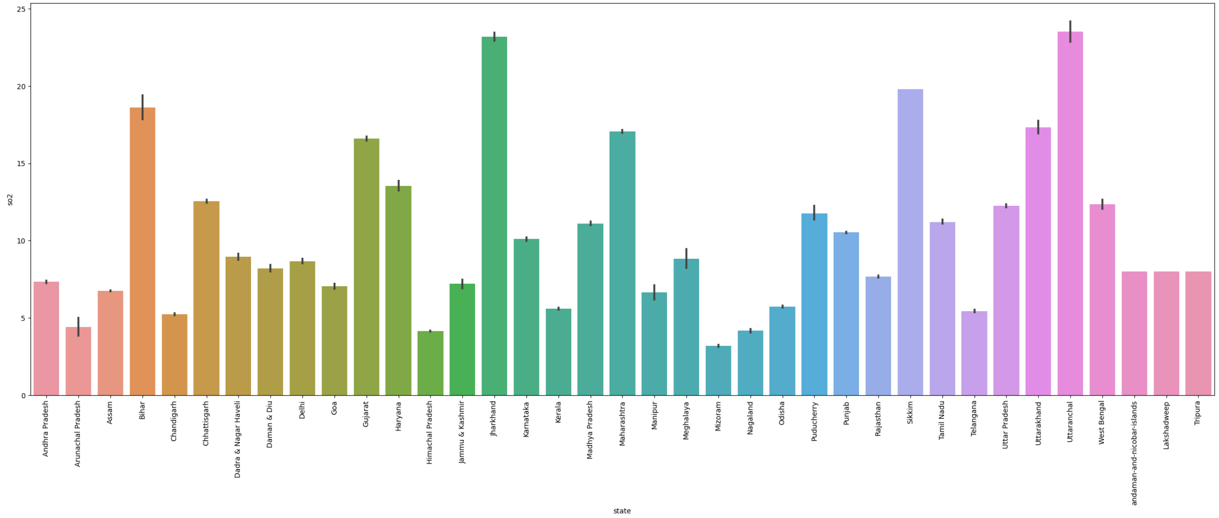


Exploratory Data Analysis

Data Visualization

Higher Sulphur-di-oxide levels in the air

→ This visualizes the SO2 levels in the air for each state.

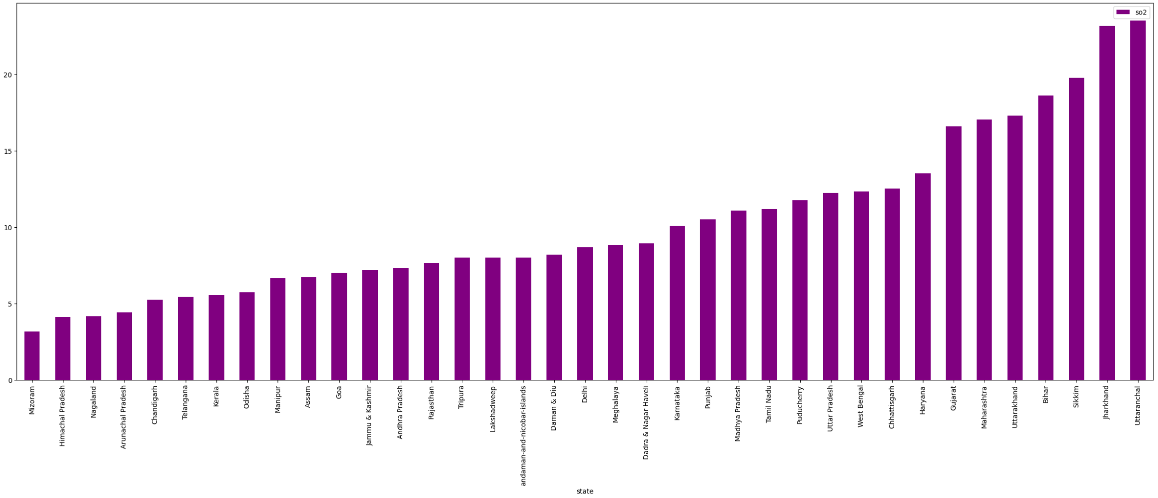


Exploratory Data Analysis

Data Visualization

Higher Sulphur-di-oxide levels in the air

→ This visualizes the SO2 levels in the air for each state (ascending).

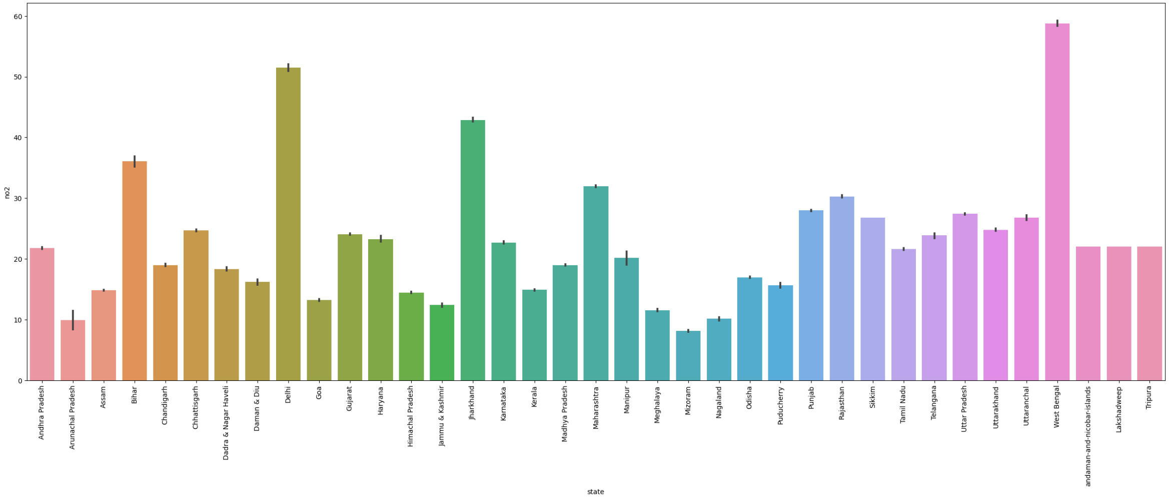


Exploratory Data Analysis

Data Visualization

Higher Nitrogen-di-oxide levels in the air

→ This visualizes the NO2 levels in the air for each state.

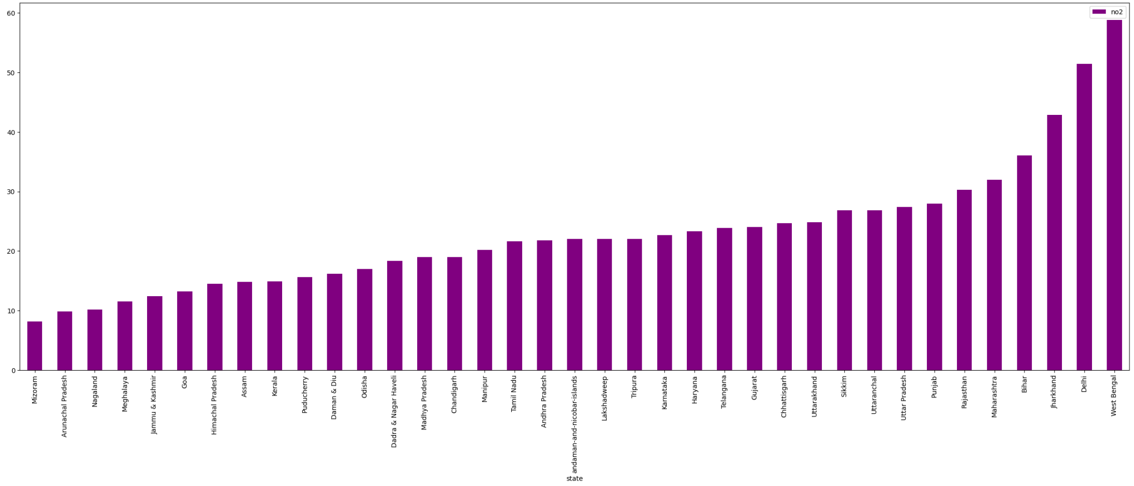


Exploratory Data Analysis

Data Visualization

Higher Nitrogen-di-oxide levels in the air

→ This visualizes the NO2 levels in the air for each state (ascending).

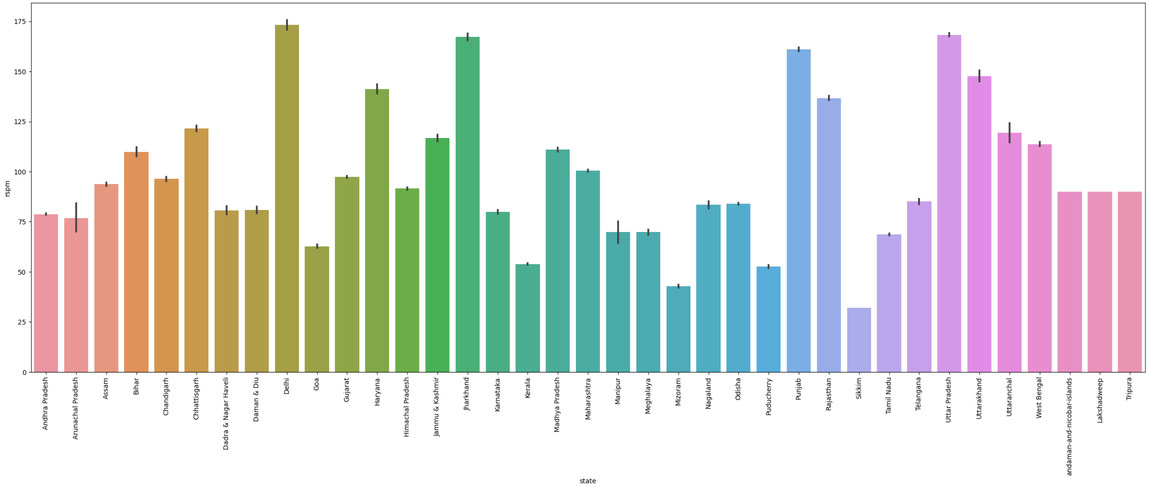


Exploratory Data Analysis

Data Visualization

Higher Respirable SPM levels in the air

→ This visualizes the RSPM levels in the air for each state.

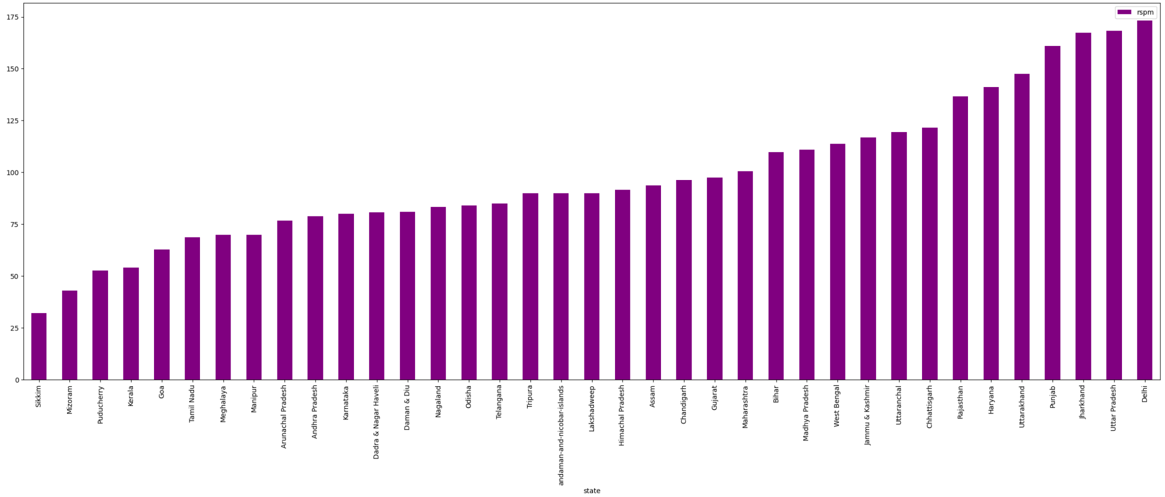


Exploratory Data Analysis

Data Visualization

Higher Respirable SPM levels in the air

→ This visualizes the RSPM levels in the air for each state (ascending).

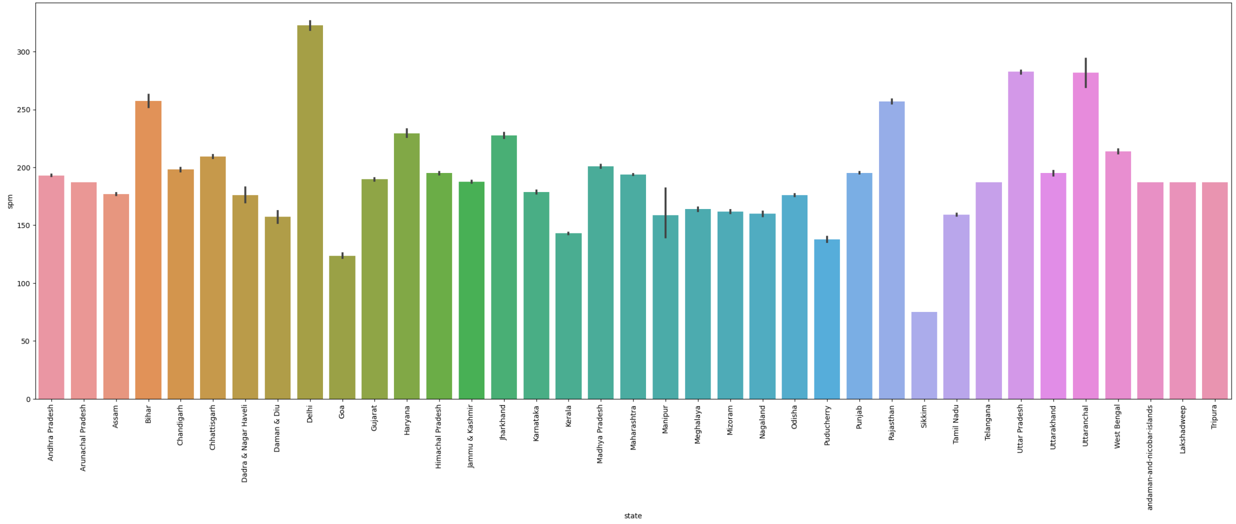


Exploratory Data Analysis

Data Visualization

Higher SPM levels in the air

→ This visualizes the SPM levels in the air for each state.

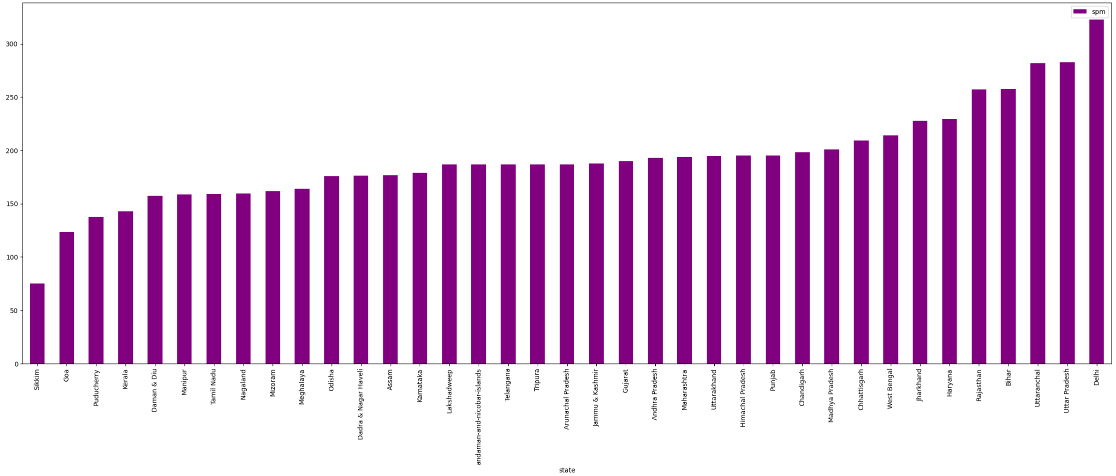


Exploratory Data Analysis

Data Visualization

Higher SPM levels in the air

→ This visualizes the SPM levels in the air for each state (ascending).

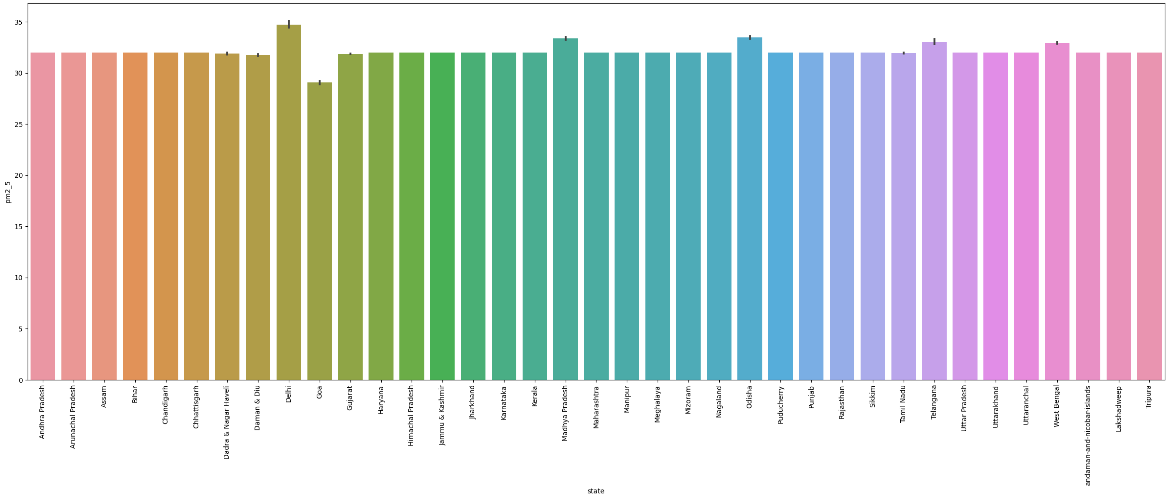


Exploratory Data Analysis

Data Visualization

Higher PM2.5 levels in the air

→ This visualizes the PM > 2.5 micrometer levels in the air for each state.

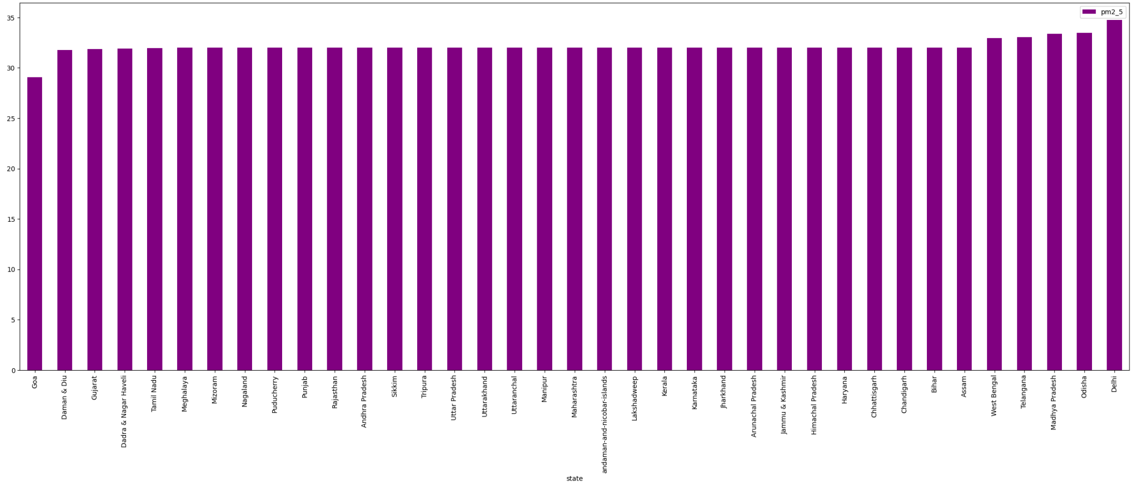


Exploratory Data Analysis

Data Visualization

Higher PM2.5 levels in the air

→ This visualizes the PM > 2.5 micrometer levels in the air for each state (ascending).

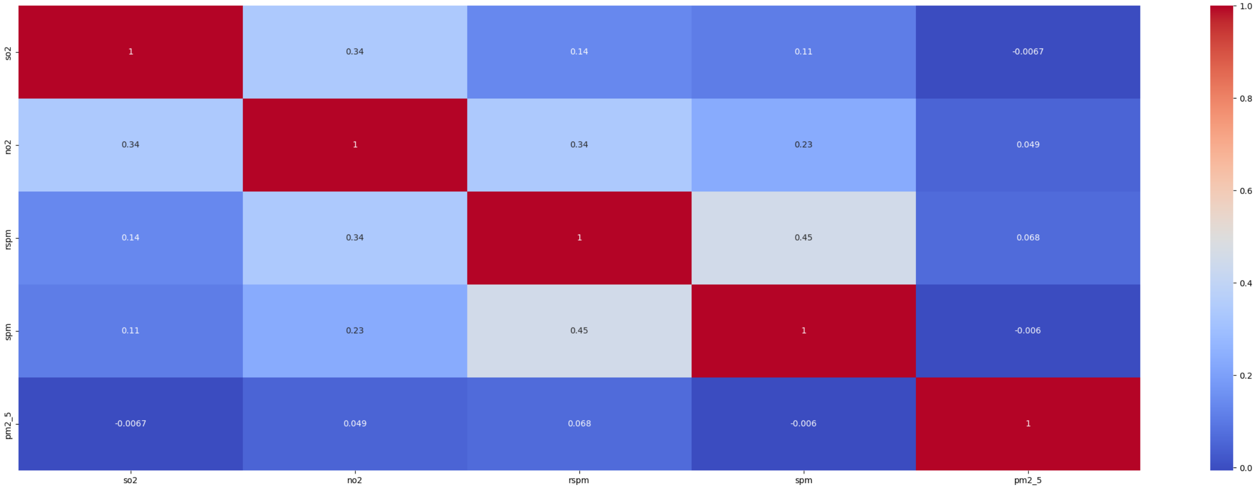


Exploratory Data Analysis

Data Visualization

Correlation Heatmap

→ Heatmap to visualize the correlation between the features of the dataset.



Predictive Model Development

Location Selection

Prediction for Kolkata, West Bengal

→ Select an Urban area from the states. Here we will be using Kokata, West Bengal.
→ The following snippet shows the states and the cities from a selected state.

```
# find all the unique states in df
df['state'].unique()
[34]  ✓ 0.0s

... array(['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',
       'Chandigarh', 'Chhattisgarh', 'Dadra & Nagar Haveli',
       'Daman & Diu', 'Delhi', 'Goa', 'Gujarat', 'Haryana',
       'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka',
       'Kerala', 'Madhya Pradesh', 'Maharashtra', 'Manipur', 'Meghalaya',
       'Mizoram', 'Nagaland', 'Odisha', 'Puducherry', 'Punjab',
       'Rajasthan', 'Sikkim', 'Tamil Nadu', 'Telangana', 'Uttar Pradesh',
       'Uttarakhand', 'Uttaranchal', 'West Bengal',
       'andaman-and-nicobar-islands', 'Lakshadweep', 'Tripura'],
      dtype=object)
```

```
# locate the rows where state is West Bengal
df.loc[df['state'] == 'West Bengal'].location.unique()
[35]  ✓ 0.1s

... array(['Haldia', 'Howrah', 'Calcutta', 'Durgapur', 'Asansol', 'Kolkata',
       'Durgapur (WB)', 'Baruipur', 'Barrackpore', 'Raniganj', 'Sankrail',
       'South Suburban', 'DANKUNI', 'HALDIA', 'Kalyani', 'MALDAH',
       'SILIGURI', 'ULUBERIA'], dtype=object)
```

Predictive Model Development

Location Selection

Prediction for Kolkata, West Bengal

→ Select an Urban area from the states. Here we will be using Kokata, West Bengal.
→ The following snippet shows the selection of Kolkata

```python
# create a dataframe with the data of Kolkata
df_kolkata = df.loc[df['location'] == 'Kolkata']

# drop the object columns
df_kolkata.drop(columns=['state', 'location', 'type'], inplace=True)

# check the shape of the dataframe
print(df_kolkata.shape)
```

```
[44]  ✓ 0.1s

... (7733, 10)
```

Predictive Model Development

Train and Test data split

Splitting data using scikit-learn

→ Split the data into train and test with 80% data into train and remaining 20% into test set.

```python
# craete test and train dataframes
# import train_test_split from sklearn
from sklearn.model_selection import train_test_split

# create the test and train dataframes
train, test = train_test_split(df_kolkata, test_size=0.2, random_state=42)

# check the shape of the train and test dataframes
print(train.shape)
print(test.shape)
```

```
(6186, 10)
(1547, 10)
```

Predictive Model Development

LazyPredict – A modern Library to run ML models

Creating and fitting models into LazyRegressor

→ Use LazyPredict to find the best model for this dataset

```python
# use lazypredict to find the best model for this dataset
# import LazyRegressor

from lazypredict.Supervised import LazyRegressor

# create the model
reg = LazyRegressor(verbose=0, ignore_warnings=False, custom_metric=None)

# fit the model
models, predictions = reg.fit(train.drop(columns=['AQI']), test.drop(columns=['AQI']), train['AQI'], test
['AQI'])

# print the models
print(models)
```

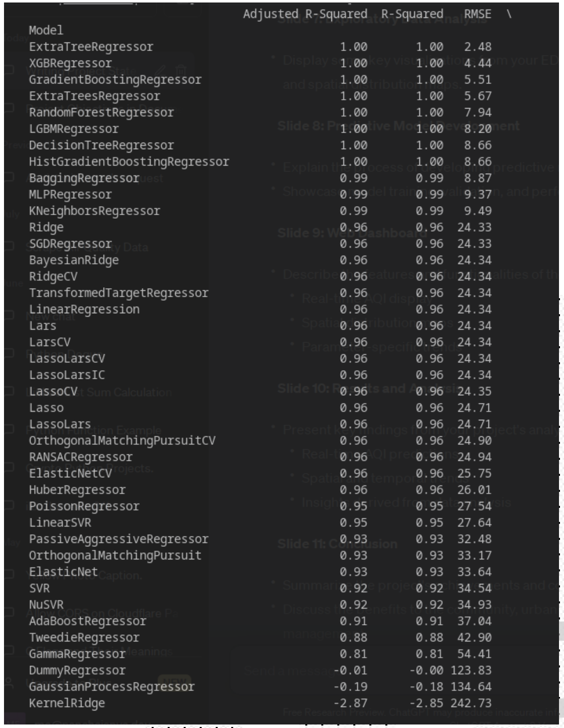
✓ 39.0s Python

Predictive Model Development

LazyPredict – A modern Library to run ML models

Creating and fitting models into LazyRegressor

→ Use LazyPredict to find the best model for this dataset



Model	Adjusted R-Squared	R-Squared	RMSE \
ExtraTreeRegressor	1.00	1.00	2.48
XGBRegressor	1.00	1.00	4.44
GradientBoostingRegressor	1.00	1.00	5.51
ExtraTreesRegressor	1.00	1.00	5.67
RandomForestRegressor	1.00	1.00	7.94
LGBMRegressor	1.00	1.00	8.20
DecisionTreeRegressor	1.00	1.00	8.66
HistGradientBoostingRegressor	1.00	1.00	8.66
BaggingRegressor	0.99	0.99	8.87
MLPRegressor	0.99	0.99	9.37
KNeighborsRegressor	0.99	0.99	9.49
Ridge	0.96	0.96	24.33
SGDRegressor	0.96	0.96	24.33
BayesianRidge	0.96	0.96	24.34
RidgeCV	0.96	0.96	24.34
TransformedTargetRegressor	0.96	0.96	24.34
LinearRegression	0.96	0.96	24.34
Lars	0.96	0.96	24.34
LarsCV	0.96	0.96	24.34
LassoLarsCV	0.96	0.96	24.34
LassoLarsIC	0.96	0.96	24.34
LassoCV	0.96	0.96	24.35
Lasso	0.96	0.96	24.71
LassoLars	0.96	0.96	24.71
OrthogonalMatchingPursuitCV	0.96	0.96	24.90
RANSACRegressor	0.96	0.96	24.94
ElasticNetCV	0.96	0.96	25.75
HuberRegressor	0.96	0.96	26.01
PoissonRegressor	0.95	0.95	27.54
LinearSVR	0.95	0.95	27.64
PassiveAggressiveRegressor	0.93	0.93	32.48
OrthogonalMatchingPursuit	0.93	0.93	33.17
ElasticNet	0.93	0.93	33.64
SVR	0.92	0.92	34.54
NuSVR	0.92	0.92	34.93
AdaBoostRegressor	0.91	0.91	37.04
TweedieRegressor	0.88	0.88	42.90
GammaRegressor	0.81	0.81	54.41
DummyRegressor	-0.01	-0.00	123.83
GaussianProcessRegressor	-0.19	-0.18	134.64
KernelRidge	-2.87	-2.85	242.73

Predictive Model Development

Use Linear Regression to predict AQI

Creating and fitting models into Linear Regresior

→ Linear Regresion can predict AQI with 96% accuracy

```
# use LinearRegression to predict the aqi for Kolkata
# import LinearRegression from sklearn\
from sklearn.linear_model import LinearRegression

# create the model
model = LinearRegression()

# fit the model
model.fit(train.drop(columns=['AQI']), train['AQI'])

# predict the aqi for test data
pred = model.predict(test.drop(columns=['AQI']))

# check the accuracy of the model
from sklearn.metrics import r2_score

print(r2_score(test['AQI'], pred)*100 , '%')
```

```
[38] ✓ 0.0s
96.12736161095054 %
```

Web Dashboard

Gradio – A Library to setup web dashboard for ML

Setting up Gradio Interface for users to interact on web

→ Gradio helps to create a basic interface for users to react with. It provides a dedicated URL to share for users while the instance is still active and running.
→ Fllowing shows a snippet to setup Gradio for this project.

```python
# implement a gradio interface to take user input and predict the aqi
# the boxes to take user input for these variables - so2, no2, rspm, spm, pm2_5, si, ni, rpi, spi, aqi

# import gradio
import gradio as gr

# create a function to predict the aqi
def predict_aqi(so2, no2, rspm, spm, pm2_5, si, ni, rpi, spi, aqi):
    # create a dataframe with the user input
    user_input = pd.DataFrame({'so2': [so2], 'no2': [no2], 'rspm': [rspm], 'spm': [spm], 'pm2_5':
    [pm2_5], 'si': [si], 'ni': [ni], 'rpi': [rpi], 'spi': [spi], 'AQI': [aqi]})
    # predict the aqi for the user input
    pred = model.predict(user_input.drop(columns=['AQI']))
    # return the predicted aqi
    return pred[0]

# create the interface
iface = gr.Interface(fn=predict_aqi, inputs=['number', 'number', 'number', 'number', 'number', 'number',
'number', 'number', 'number', 'number'], outputs='number')

# launch the interface
iface.launch(share=True)
```

✓ 5.0s Python

Web Dashboard

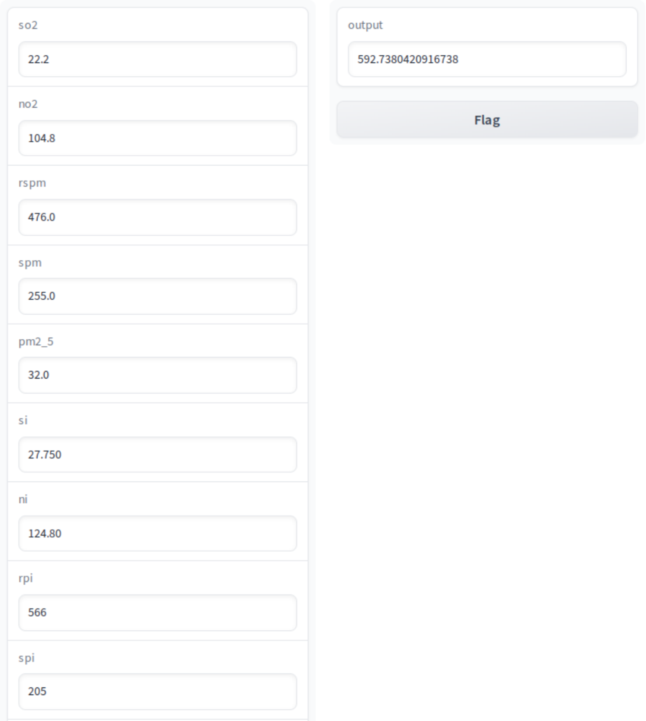
Gradio – A Library to setup web dashboard for ML

Web Interface to react with the model

→ User needs to give the input from the sensor data they have collected

→ The model will return the predicted AQI value.

→ On the snippet on the right, we have taken data from an Industrial Area and we Are getting a higher AQI which shows the area is poluted and it is totlally unhealthy.



Results and Analysis

Results and Analysis of the Project

→ **Real Time AQI Predicions:** With sensor data and calculated data, one can easily predict the AQI for the data provided.

→ **Health Implications:** Linked high AQI values to potential health risks, highlighting the importance of timely air quality information.

→ **Testing Model Validation:** Rigorous model testing gives us a better validation on the model we are working on.

thank you.

Photo by Dave Hoefler on Unsplash