AN ANALYSIS OF AMBIENT AIR QUALITY IN STATES OF INDIA

Big Data Analytics and Visualization Using PySpark and Tableau

Student Name : Roshan Dhanasekaran

SID : 12135637

Subject Code : 7153CEM

I can confirm that all work submitted is my own : YES

Roshan Dhanasekaran

Abstract

This paper analyses and compares the pollution levels from the year 2015 till 2020 with the respective 26 states of India with the National NAAQ standards. The data is again used with machine learning algorithms like regression to predict AQI.

Big Data analytics was done with pySpark. Tableau was used to visualize the data with different chats and have an inference about the outcome.

Keywords - NAAQ, Regression, Machine learning, AQI, pySpark, Tableau

Link to the Dataset: https://www.kaggle.com/rohanrao/air-quality-data-in-india

Content Page

Introduction

The coursework was written as a part of a Big Data management and data visualization module. The course work uses necessary concepts of Big Data for analyzing the data and bringing out insights of the data with visualizing techniques which is intern used to take necessary actions over the polluting states.

The course work involves features of Apache Spark. This is then used in a more user-friendly environment by incorporating pySpark in Jupyter notebooks where in which we can see the execution time and the cores used for data management.

The main focus of this course work is to analyse ambient air pollution in India and find out which state was the major contributor of air pollution and the cause of pollution and what was the pollutant and its effects on the people and the environment.

1. PySpark for loading and processing data.
2. Analysing the data with appropriate techniques
3. Using Machine learning regression to predict AQI
4. Using the data to visualize and give inferences and feedback.

1. Installation & Implementation

1.1 Installing prerequisite

We start by installing Home Brew on the terminal. Home Brew is a package installer that is used to install packages on mac OS. Homebrew lets us install pySpark onto the local machine.

```
Last login: Thu Feb  3 16:19:02 on ttys000
roshandhanashekeran@Roshans-MacBook-Pro ~ % /bin/bash -c "$(curl -fsSL https://raw.github
==> Checking for `sudo` access (which may request your password)...
Password:
==> This script will install:
/opt/homebrew/bin/brew
/opt/homebrew/share/doc/homebrew
/opt/homebrew/share/man/man1/brew.1
/opt/homebrew/share/zsh/site-functions/_brew
/opt/homebrew/etc/bash_completion.d/brew
/opt/homebrew

```

Figure 1. Installing Home Brew In Terminal

The entire setup was done successfully. Another major prerequisite was JAVA 8. JAVA 8 was manually downloaded and installed in the local machine.

```
roshandhanashekeran@Roshans-MacBook-Pro ~ % java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.321-b07, mixed mode)
roshandhanashekeran@Roshans-MacBook-Pro ~ %
```

Figure 2. Installing JAVA 8

1.2 Installing Spark

Spark 2.3.0 was downloaded manually from the Apache website and installed in the Terminal

```
roshandhanashekeran@Roshans-MacBook-Pro ~ % cd Desktop
roshandhanashekeran@Roshans-MacBook-Pro Desktop % cd spark-2.3.0-bin-hadoop2.7
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-hadoop2.7 % ls
LICENSE         README.md       conf            jars            python
NOTICE          RELEASE         data            kubernetes      sbin
R               bin             examples        licenses        yarn
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-hadoop2.7 %
```

Figure 3. Installing spark

After installation of spark, the important thing to do is to set the path for java setting the path properly will let the user access spark-shell with ease and this is the reason why we installed JAVA8 instead of any other versions, JAVA 8 is compatible with spark 2.3.0

```
roshandhanashekeran@Roshans-MacBook-Pro ~ % cd Desktop
roshandhanashekeran@Roshans-MacBook-Pro Desktop % cd spark-2.3.0-bin-hadoop2.7
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-hadoop2.7 % ls
LICENSE         README.md       conf            jars            python
NOTICE          RELEASE         data            kubernetes      sbin
R               bin             examples        licenses        yarn
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-hadoop2.7 % export SPARK_HOME=`pw
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-hadoop2.7 % PATH=$SPARK_HOME/bin:
roshandhanashekeran@Roshans-MacBook-Pro spark-2.3.0-bin-
```

Figure 4. Setting up java path

After setting up the path we can open up spark-shell by giving the command spark-shell or bin/spark-shell on mac

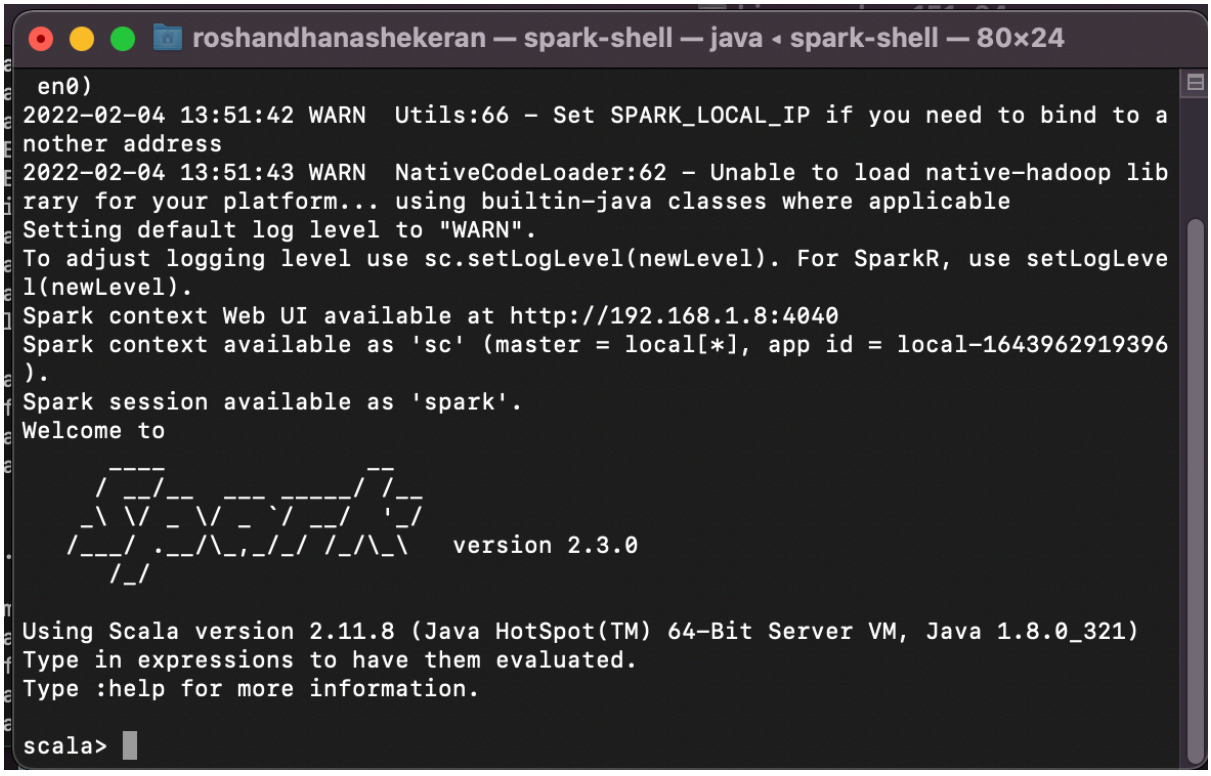


Figure 5. opening spark-shell

Now we have opened a spark session where we can import out datasets into the shell and we can also track the tasks taken in the shell by going into the spark UI. There are many ways to get into the spark UI one simple way is to copy the above URL available in the shell and paste it into the browser.

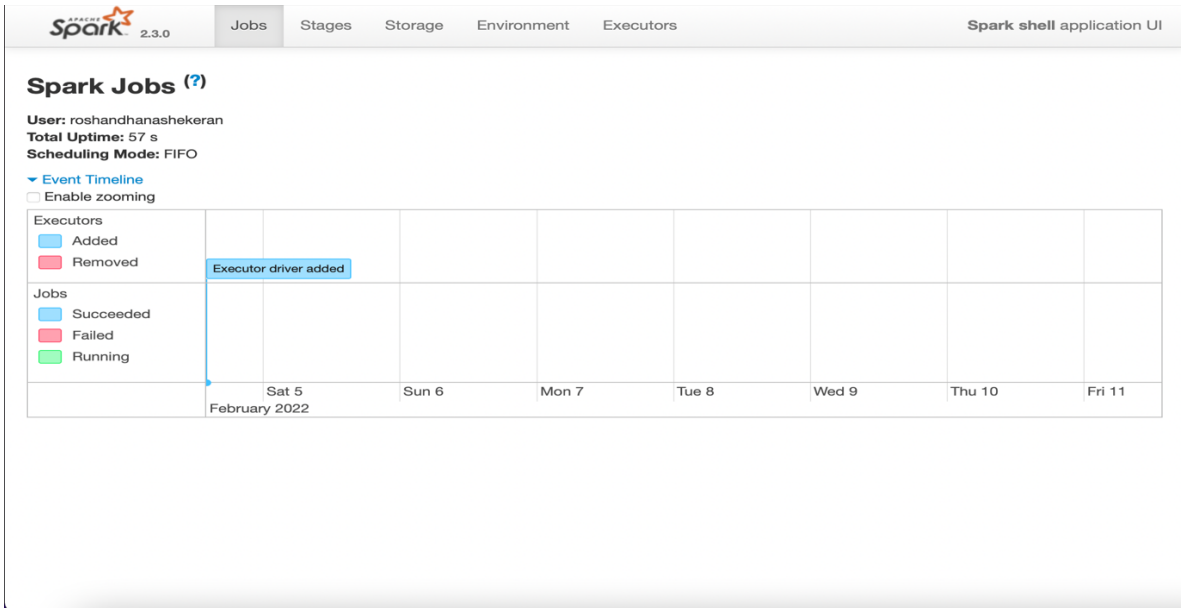


Figure 6. Spark job tracker

Since we have opened the shell it is wise to check the executers in the shell, if the shell uses fewer cores to do a task it will take significantly more time to do the job instead the machine

should be configured so that spark can take advantage of all the cores in the respective local machine.

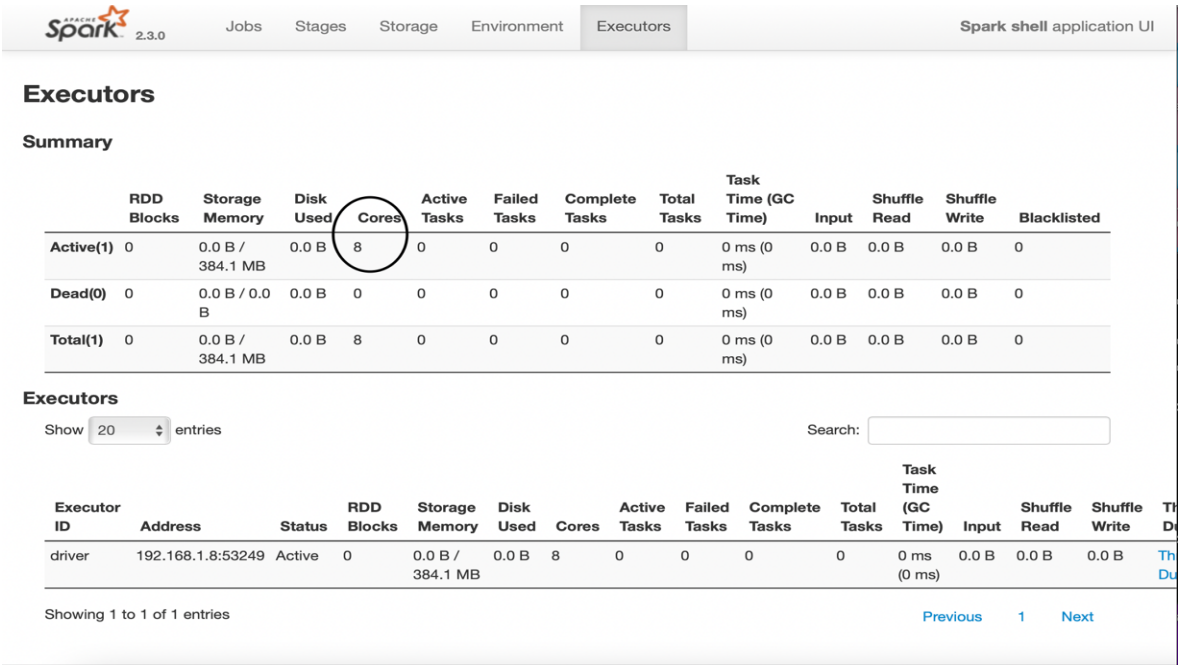


Figure 7. Checking the cores

1.3 Installing pySpark and using jupyter notebook

We start by going to the pipenv website (https://pipenv.pypa.io/en/latest/) **Pipenv** is a tool that aims to bring the best of all packaging worlds (bundler, composer, npm, cargo, yarn, etc.) to the Python world. Now we can go to the terminal and install pipenv by using a simple code %brew install pipenv since we have already had homebrew installed In the local system the macOS will capitalize the code and the installation starts automatically.

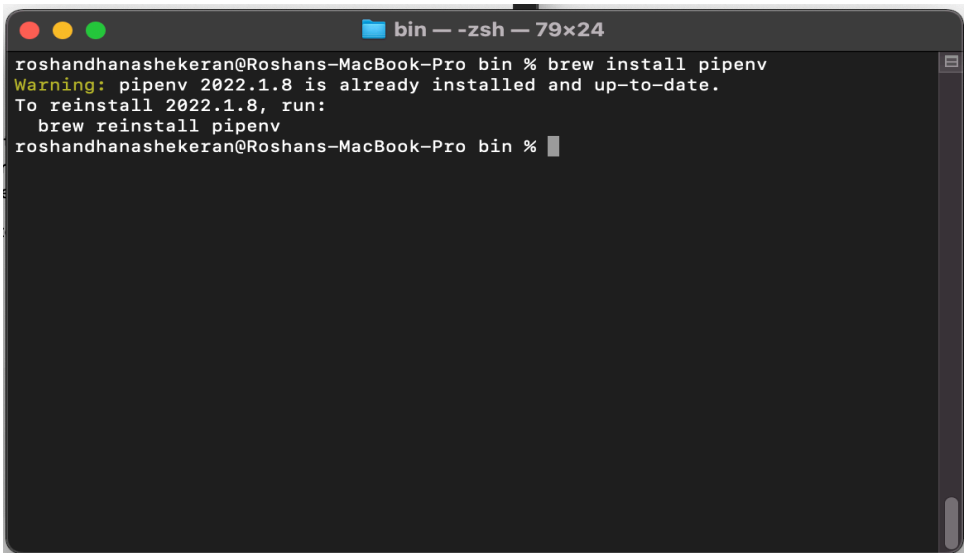


Figure 8. Installing pipenv

After installing both spark and pipenv it is time to connect both with jupyter notebook this can be done by installing jupyter first by using pipenv install jupyter we will be able to install

jupyter successfully onto the local machine. After the successful installation of Jupiter, we can start a notebook that will direct us to the jupyter session.

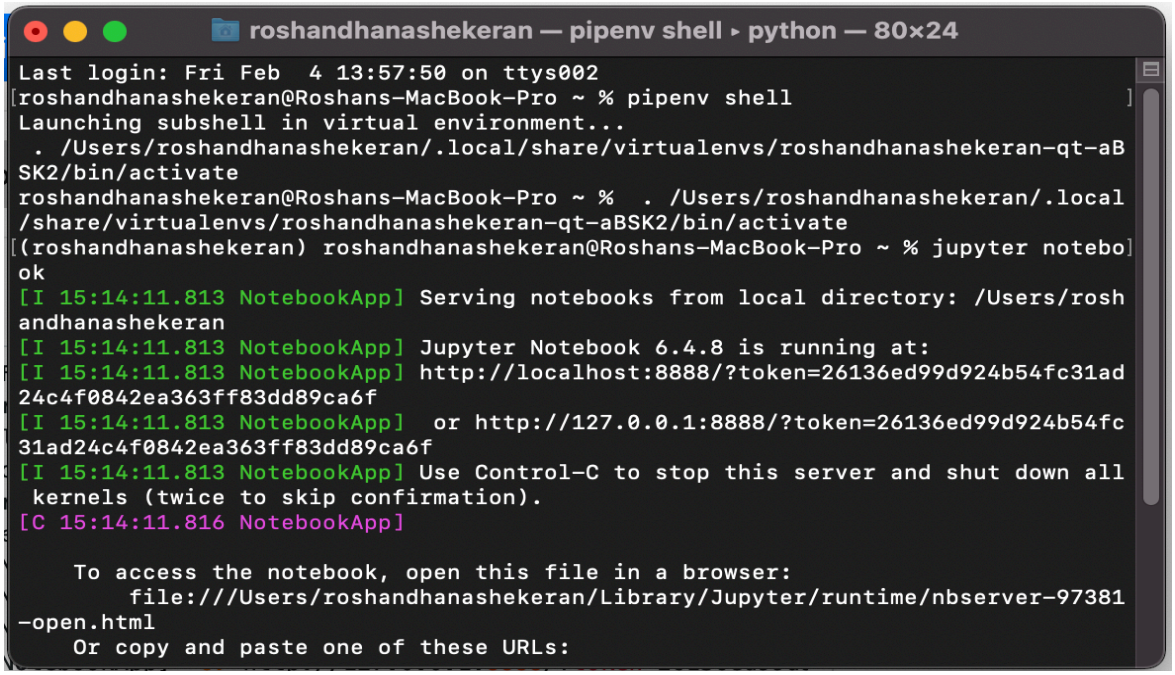


Figure 9. Opening jupyter notebook.

This command will open up our jupyter session which is more user-friendly and accessible.

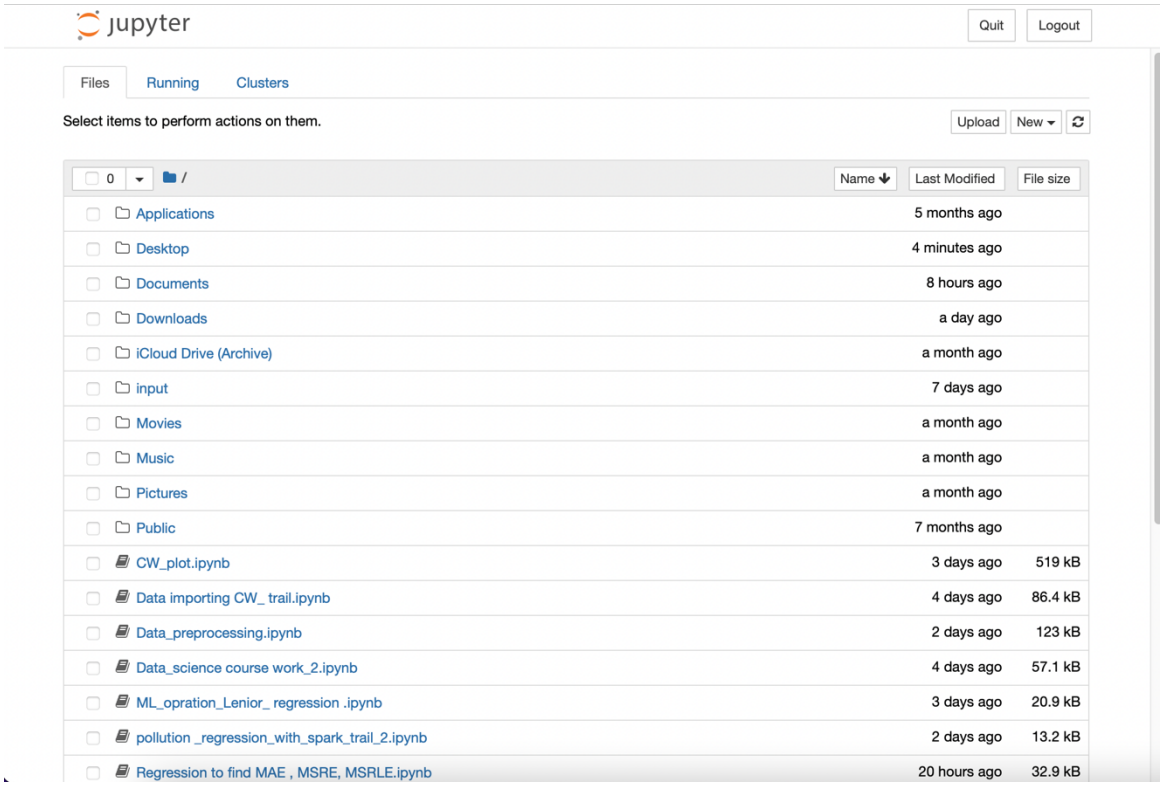


Figure 10. opening jupyter notebook

We can now create a new python notebook and import all the required spark packages like SparrkContext() and getOrCreate() and other pandas and NumPy packages to do data pre-processing and validation.

It is advisable to take a brief look at the data before importing and know about the data extensively and be aware of the measures and standards in the country this will help us to get a meaningful inference of the data. This brings us to the next part of the coursework.

2.Background & Standards

2.1 Standards

Each country has its standards of VOC exceedance. The Indian standards may vary with other countries. Primarily we focus on the pollutants given in the data sets and discuss their effects on the environment and the people and discuss yearly Indian NAAQ standards in brief.

The 12 major attributes in the datasets are PM10, PM2.5, NO, NO2, NOX, NH3, CO, SO2,03, BENZENE, TOLUENE, XYLENE. This will be compared with respective years from 2015 to 2020 and with respective cities.

Column Name	Data Type	Description	Measurement
City	Geolocation	States	N/A
Date	Timestamp	Timestamp of daily data	N/A
PM2.5	Numerical	Particulate matter 2.5	ppm
PM10	Numerical	Particulate matter 10	ppm
NO	Numerical	Nitric oxide	$\mu g/m^3$
NO2	Numerical	Nitrogen dioxide	$\mu g/m^3$
NOX	Numerical	Nitrogen oxide	$\mu g/m^3$
NH3	Numerical	Ammonia	$\mu g/m^3$
CO	Numerical	Carbon monoxide	$\mu g/m^3$
SO2	Numerical	Sulphur dioxide	$\mu g/m^3$
03	Numerical	ozone	mg/m^3
BENZENE	Numerical	Benzene	$\mu g/m^3$
XYLENE	Numerical	xylene	ppm
TOLUENE	Numerical	Toluene	ppm
AQI	Categorical	Air Quality Index	String

Table 1. Dataset basic info

National Ambient Air Quality Standards

Pollutant	Time Weighted Average	Concentration in Ambient Air	
		Industrial, Residential, Rural and Other Areas	Ecologically Sensitive Area (notified by Central Government)
Sulphur Dioxide (SO_2), $\mu g/m^3$	Annual* 24 hours**	50 80	20 80
Nitrogen Dioxide (NO_2), $\mu g/m^3$	Annual* 24 hours**	40 80	30 80
Particulate Matter (size less than 10 µm) or PM_{10} $\mu g/m^3$	Annual* 24 hours**	60 100	60 100
Particulate Matter (size less than 2.5 µm) or $PM_{2.5}$ $\mu g/m^3$	Annual* 24 hours**	40 60	40 60
Ozone (O_3) $\mu g/m^3$	8 hours* 1 hour**	100 180	100 180
Lead (Pb) $\mu g/m^3$	Annual* 24 hours**	0.50 1.0	0.50 1.0
Carbon Monoxide (CO) mg/m^3	8 hours* 1 hour**	02 04	02 04
Ammonia (NH_3) $\mu g/m^3$	Annual* 24 hours**	100 400	100 400
Benzene (C_6H_6) $\mu g/m^3$	Annual*	5	5
Benzo(a)Pyrene (BaP)-particulate phase only, ng/m^3	Annual*	1	1
Arsenic(As), ng/m^3	Annual*	6	60
Nickel (Ni), ng/m^3	Annual*	20	20
Note: * Annual arithmetic mean of minimum 104 measurements in a year at a particular site taken twice a week 24 hourly at uniform intervals. ** 24 hourly or 8 hourly or 1 hourly monitored values, as applicable, shall be complied with 98% of the time, they may exceed the limits but not on two consecutive days of monitoring.			

Table 2. Standards in India

Since we have basic information about the data and standards imposed by the government. Let us see the effects of these pollutants if exceeds the limits.

2.2 Application and Effects

NO2 - Can harm the human respiratory system and make a person more susceptible to and suffer from respiratory infections and asthma.

CO - Breathing CO-rich air limits the quantity of oxygen that can be transferred to vital organs like the heart and brain through circulation.

NO - Longer exposure to high NO2 levels may contribute to the development of asthma and perhaps increase vulnerability to respiratory infections.

SO2 - Sulphur dioxide irritates the eyes and affects the respiratory system, particularly lung function. The respiratory tract is irritated by sulphur dioxide, which raises the risk of infection.

PM10 - High levels of PM10 can cause a variety of health problems, from coughing and wheezing to asthma attacks and bronchitis

PM2.5 - Fine particle exposure can irritate the eyes, nose, throat, and lungs, as well as coughing, sneezing, runny nose, and shortness of breath.

O3 - Breathing ground-level ozone can cause chest discomfort, coughing, throat irritation, and congestion, among other symptoms.

Benzene - Benzene is toxic to the bone marrow and can induce a reduction in red blood cells, resulting in anemia.

Xylene - Inhaling xylene vapor causes central nervous system depression, resulting in symptoms such as headache, dizziness, nausea, and vomiting.

Toluene - The colorless liquid toluene ($C_6H_5CH_3$) has a pleasant, pungent odour. Eye and nose irritation, exhaustion, confusion, euphoria, dizziness, headache,

All the above data was referred from google.

3. Implementation

3.1 Importing data

We begin with using spark SQL since it is used for structured data processing, and its interface provides more details to the user. In this course work, we will be using spark SQL and pandas, and other data frames for the computational part and Tableau for visualizing the data.

We start with importing the pandas library in a new spark session and locating the CSV file in the local machine and import into the session see figure 13. And figure 14

```
In [1]: import pandas as pd

In [2]: from pyspark import SparkContext

In [3]: sc = SparkContext.getOrCreate()
22/01/31 14:58:16 WARN Utils: Your hostname, Roshans-MacBook-Pro.local resolves to a loopback address: 127.0.0.1; u
sing 192.168.1.8 instead (on interface en0)
22/01/31 14:58:16 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/01/31 14:58:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable
```

Figure 11. initializing and importing required libraries

We then continue by importing libraries from pySpark SQL like the SQLContext and sparkSession and we build our spark session. in the next command we assign File_loc_csv as our dataset which is city_day.csv, this is assigned to a variable called df which we will be using in the rest of the commands and then we use the show(). Check figure 14. and 15

```
In [5]: from pyspark import SparkConf, SparkContext
        from pyspark.sql import SQLContext ,SparkSession

In [6]: spark = SparkSession.builder.master("local[*]").getOrCreate()

In [12]: file_loc_csv = 'city_day.csv'

In [13]: df=spark.read.csv(file_loc_csv, inferSchema=True, header=True)

In [14]: print(type(df))
<class 'pyspark.sql.dataframe.DataFrame'>
```

Figure 12. importing SQL library into the session

By using the show() function we can have a brief about the data in the datasets by default the function shows 20 lines.

```
[15]: df.show()
```

```
+---------+----------+-----+----+------+-----+------+----+------+-----+------+-------+-------+------+----+---
-------+
|     City|      Date|PM2.5|PM10|    NO|  NO2|   NOx| NH3|    CO|  SO2|    O3|Benzene|Toluene|Xylene| AQI|AQI
_Bucket|
+---------+----------+-----+----+------+-----+------+----+------+-----+------+-------+-------+------+----+---
-------+
|Ahmedabad|2015-01-01| null|null|  0.92|18.22| 17.15|null|  0.92|27.64|133.36|    0.0|   0.02|   0.0|null|
null|
|Ahmedabad|2015-01-02| null|null|  0.97|15.69| 16.46|null|  0.97|24.55| 34.06|   3.68|    5.5|  3.77|null|
null|
|Ahmedabad|2015-01-03| null|null|  17.4| 19.3|  29.7|null|  17.4|29.07|  30.7|    6.8|   16.4|  2.25|null|
null|
|Ahmedabad|2015-01-04| null|null|   1.7|18.48| 17.97|null|   1.7|18.59| 36.08|   4.43|  10.14|   1.0|null|
null|
|Ahmedabad|2015-01-05| null|null|  22.1|21.42| 37.76|null|  22.1|39.33| 39.31|   7.01|  18.89|  2.78|null|
null|
|Ahmedabad|2015-01-06| null|null| 45.41|38.48|  81.5|null| 45.41|45.76| 46.51|   5.42|  10.83|  1.93|null|
null|
|Ahmedabad|2015-01-07| null|null|112.16|40.62|130.77|null|112.16|32.28| 33.47|    0.0|    0.0|   0.0|null|
null|
|Ahmedabad|2015-01-08| null|null| 80.87|36.74| 96.75|null| 80.87|38.54| 31.89|    0.0|    0.0|   0.0|null|
null|
|Ahmedabad|2015-01-09| null|null| 29.16| 31.0|  48.0|null| 29.16|58.68| 25.75|    0.0|    0.0|   0.0|null|
null|
|Ahmedabad|2015-01-10| null|null|  null| 7.04|   0.0|null|  null| 8.29|  4.55|    0.0|    0.0|   0.0|null|
null|
|Ahmedabad|2015-01-11| null|null|132.07| 55.8| 24.53|null|132.07|25.03|  6.79|    0.0|    0.0|   0.0|null|
null|
|Ahmedabad|2015-01-12| null|null| 52.04|40.67| 90.24|null| 52.04|51.84| 45.89|   2.41|   0.03|  7.88|null|
null|
|Ahmedabad|2015-01-13| null|null| 48.82| 44.2| 87.09|null| 48.82|68.21| 35.16|   9.45|  13.35|  12.5|null|
null|
|Ahmedabad|2015-01-14| null|null|  19.2|27.86| 33.05|null|  19.2|52.65| 20.96|   2.16|   2.26|  5.19|null|
null|
|Ahmedabad|2015-01-15| null|null|   0.6|16.96|  16.6|null|   0.6|28.89| 47.63|   0.14|   0.04|  1.35|null|
null|
|Ahmedabad|2015-01-16| null|null|  1.63|21.72| 22.86|null|  1.63|38.27| 46.03|   0.35|   0.05|  2.01|null|
null|
|Ahmedabad|2015-01-17| null|null| 11.44|24.73| 34.75|null| 11.44| 49.5| 52.24|   0.68|    0.0|  3.27|null|
null|
|Ahmedabad|2015-01-18| null|null|   6.1|25.77| 29.57|null|   6.1|48.43| 53.49|   0.74|   0.21|  2.75|null|
null|
|Ahmedabad|2015-01-19| null|null|  2.51|26.88| 27.45|null|  2.51|50.03| 49.48|   0.26|   0.02|   2.8|null|
null|
|Ahmedabad|2015-01-20| null|null|  7.92| 26.8|  32.4|null|  7.92|58.87| 56.37|   0.24|   0.01|  3.97|null|
null|
+---------+----------+-----+----+------+-----+------+----+------+-----+------+-------+-------+------+----+---
-------+
only showing top 20 rows
```

Figure 13. Showing the first 20 rows of the datasets

We move on to find out the schema of the dataset this will indicate what variables we are dealing with in the dataset and check for null values in the dataset.

```
In [24]: df.printSchema()
```

```
root
 |-- City: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- PM2.5: double (nullable = true)
 |-- PM10: double (nullable = true)
 |-- NO: double (nullable = true)
 |-- NO2: double (nullable = true)
 |-- NOx: double (nullable = true)
 |-- NH3: double (nullable = true)
 |-- CO: double (nullable = true)
 |-- SO2: double (nullable = true)
 |-- O3: double (nullable = true)
 |-- Benzene: double (nullable = true)
 |-- Toluene: double (nullable = true)
 |-- Xylene: double (nullable = true)
 |-- AQI: double (nullable = true)
 |-- AQI_Bucket: string (nullable = true)
```

Figure 14. showing the schema of the dataset

Now we need the check for negative values and convert them. if there are any negatives we need it to be read as 0 this has to be done in this dataset because there cannot be values below zero.

```
In [44]: import pyspark.sql.functions as F

In [45]: # CHECKING NEGATIVE VALUES
         Col_List =["PM10","NO2","O3","CO","SO2","NH3","NO","Benzene","Toluene","Xylene"]

         for n in Col_List:
             df=df.withColumn(n , F.when(F.col(n)< 0, None).otherwise(F.col(n)))
             df.show()
```

```
+---------+----------+----+------+-----+------+----+------+-----+------+-------+-------+------+----+-------
---+
|     City|      Date|PM10|    NO|  NO2|   NOx| NH3|    CO|  SO2|    O3|Benzene|Toluene|Xylene| AQI|AQI_Buc
ket|
+---------+----------+----+------+-----+------+----+------+-----+------+-------+-------+------+----+-------
---+
|Ahmedabad|2015-01-01|null|  0.92|18.22| 17.15|null|  0.92|27.64|133.36|    0.0|   0.02|   0.0|null|      n
ull|
|Ahmedabad|2015-01-02|null|  0.97|15.69| 16.46|null|  0.97|24.55| 34.06|   3.68|    5.5|  3.77|null|      n
ull|
|Ahmedabad|2015-01-03|null|  17.4| 19.3|  29.7|null|  17.4|29.07|  30.7|    6.8|   16.4|  2.25|null|      n
ull|
|Ahmedabad|2015-01-04|null|   1.7|18.48| 17.97|null|   1.7|18.59| 36.08|   4.43|  10.14|   1.0|null|      n
ull|
|Ahmedabad|2015-01-05|null|  22.1|21.42| 37.76|null|  22.1|39.33| 39.31|   7.01|  18.89|  2.78|null|      n
ull|
|Ahmedabad|2015-01-06|null| 45.41|38.48|  81.5|null| 45.41|45.76| 46.51|   5.42|  10.83|  1.93|null|      n
ull|
|Ahmedabad|2015-01-07|null|112.16|40.62|130.77|null|112.16|32.28| 33.47|    0.0|    0.0|   0.0|null|      n
ull|
|Ahmedabad|2015-01-08|null| 80.87|36.74| 96.75|null| 80.87|38.54| 31.89|    0.0|    0.0|   0.0|null|      n
```

Figure 15 Converting negative values into 0

As we can see the data set has a lot of null values and it is difficult to work with null values in the data. The final output can be misrepresented so we use the na.drop() function to remove all the null values in the data set. First and foremost we check how many null values are there in the dataset see figure 18

```
In [72]: #NULL VALUES IN THE DATASET
         from pyspark.sql.functions import isnan, when, count, col
         df_Null = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
         df_Null.show()
```

```
+----+----+-----+----+----+----+-----+----+----+----+-------+-------+------+----+-----------+
|City|Date| PM10|  NO| NO2| NOx|  NH3|  CO| SO2|  O3|Benzene|Toluene|Xylene| AQI|AQI_Bucket|
+----+----+-----+----+----+----+-----+----+----+----+-------+-------+------+----+-----------+
|   0|   0|11140|3582|3585|4185|10328|2059|3854|4022|   5623|   8041| 18109|4681|       4681|
+----+----+-----+----+----+----+-----+----+----+----+-------+-------+------+----+-----------+
```

Figure16. Checking all the null values

We now go on to filter the data and remove all the null values in the data and again check if there are any leftovers

```
In [29]: 1 df.na.drop(how ="any").show()
```

```
+---------+----------+-----+----+-----+-----+-----+----+-----+-----+-------+-------+------+-----+-----------+
|     City|      Date|PM10|  NO|  NO2|  NOx|  NH3|  CO|  SO2|   O3|Benzene|Toluene|Xylene|  AQI|AQI_Bucket|
+---------+----------+-----+----+-----+-----+-----+----+-----+-----+-------+-------+------+-----+-----------+
|Amaravati|2017-11-25| 124.5|1.44| 20.5|12.08|10.72|0.12|15.24|127.09|    0.2|    6.5|  0.06|184.0|  Moderate|
|Amaravati|2017-11-26|129.06|1.26| 26.0|14.85|10.28|0.14|26.96|117.44|   0.22|   7.95|  0.08|197.0|  Moderate|
|Amaravati|2017-11-27|135.32| 6.6|30.85|21.77|12.91|0.11|33.59|111.81|   0.29|   7.63|  0.12|198.0|  Moderate|
|Amaravati|2017-11-28|104.09|2.56|28.07|17.01|11.42|0.09| 19.0|138.18|   0.17|   5.02|  0.07|188.0|  Moderate|
|Amaravati|2017-11-29|114.84|5.23| 23.2|16.59|12.25|0.16|10.55|109.74|   0.21|   4.71|  0.08|173.0|  Moderate|
|Amaravati|2017-11-30|114.86|4.69|20.17|14.54|10.95|0.12|14.07|118.09|   0.16|   3.52|  0.06|165.0|  Moderate|
|Amaravati|2017-12-01|113.56|4.58|19.29|13.97|10.95| 0.1| 13.9| 123.8|   0.17|   2.85|  0.04|191.0|  Moderate|
|Amaravati|2017-12-02| 140.2|7.71|26.19|19.87|13.12| 0.1|19.37|128.73|   0.25|   2.79|  0.07|191.0|  Moderate|
|Amaravati|2017-12-03|130.52|0.97|21.31|12.12|14.36|0.15|11.41| 114.8|   0.23|   3.82|  0.04|227.0|      Poor|
|Amaravati|2017-12-04| 125.0|4.02|26.98|17.58|14.41|0.18| 9.84|112.41|   0.31|   3.53|  0.09|168.0|  Moderate|
|Amaravati|2017-12-05|121.77| 3.7|20.23|13.75|13.72|0.12|14.02|117.93|   0.24|   2.92|  0.03|198.0|  Moderate|
|Amaravati|2017-12-06|139.36| 1.6|25.65|14.99|15.12|0.11|16.54|117.21|   0.29|   4.45|  0.07|201.0|      Poor|
|Amaravati|2017-12-07|181.64|4.26| 41.1|25.32|17.34|0.13|28.79| 94.63|   0.36|   6.21|  0.17|252.0|      Poor|
|Amaravati|2017-12-08|208.86|5.56|54.87|33.71|17.96|0.27|22.97|  68.6|   0.36|   6.28|  0.21|310.0| Very Poor|
|Amaravati|2017-12-09|141.22| 6.1|44.97|28.88|15.73|0.09| 21.9| 60.62|   0.26|   4.79|  0.16|196.0|  Moderate|
|Amaravati|2017-12-10|102.77|1.73|33.85|19.41|12.56| 0.1|13.65| 68.15|    0.2|   4.29|   0.1|132.0|  Moderate|
|Amaravati|2017-12-11|115.27|4.93|41.64|26.15| 15.2|0.16|18.37| 73.75|   0.23|   5.51|  0.16|147.0|  Moderate|
|Amaravati|2017-12-12|131.48|7.97| 42.1|28.88|21.24|0.24| 7.42| 44.67|   0.28|   7.01|  0.19|179.0|  Moderate|
|Amaravati|2017-12-13| 99.74| 7.2|34.78|24.36|17.63|0.15| 5.81| 50.16|   0.24|   6.11|  0.14|145.0|  Moderate|
|Amaravati|2017-12-14| 98.94|5.81|29.97|20.67|19.34|0.14|  5.8|  55.0|   0.23|   5.09|  0.14|115.0|  Moderate|
+---------+----------+-----+----+-----+-----+-----+----+-----+-----+-------+-------+------+-----+-----------+
only showing top 20 rows
```

```
In [36]: from pyspark.sql.functions import isnan , when, count, col
         df2 = df.select([count(when(isnan(c), c)).alias(c) for c in df.columns])
         df2.show()

         [Stage 15:>                                                         (0 + 1) / 1]

+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
|City|Date|PM10| NO|NO2|NOx|NH3| CO|SO2| O3|Benzene|Toluene|Xylene|AQI|AQI_Bucket|
+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
|   0|   0|   0|  0|  0|  0|  0|  0|  0|  0|      0|      0|     0|  0|         0|
+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
```

Figure17. Clearing the null values.

The next step is to drop all the duplicate values from the dataset this can be achieved with the drop duplicates() function

```
In [16]: 1 df.count()
Out[16]: 29531

In [54]: 1 df.dropDuplicates()
Out[54]: DataFrame[City: string, Date: string, PM10: double, NO: double, NO2: double, NOx: double, NH3: double, CO: double, S
         O2: double, O3: double, Benzene: double, Toluene: double, Xylene: double, AQI: double, AQI_Bucket: string]

In [55]: 1 df.count()
Out[55]: 29531

In [ ]: 1
```

Figure 18. checking for duplicate values

Moving on we can find the min-max standard deviation of the data by doing a simple function describe() But in some cases, the output is cropped and cannot be viewed properly in that case we can use truncate = False or set a truncate value or we view it vertically. Find figure 21.

```
In [46]: 1 df.describe().show(truncate=3)

         [Stage 97:>                                                         (0 + 1) / 1]

+-------+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
|summary|City|Date|PM10| NO|NO2|NOx|NH3| CO|SO2| O3|Benzene|Toluene|Xylene|AQI|AQI_Bucket|
+-------+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
|    cou| 295| 295| 183|259|259|253|192|274|256|255|    239|    214|   114|248|       248|
|    mea| nul| nul| 118|17.|28.|32.|23.|2.2|14.|34.|    3.2|    8.7|   3.0|166|       nul|
|    std| nul| nul| 90.|22.|24.|31.|25.|6.9|18.|21.|    15.|    19.|   6.3|140|       nul|
|    min| Ahm| 201| 0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|    0.0|    0.0|   0.0|13.|       Goo|
|    max| Vis| 202| 100|390|362|467|352|175|193|257|    455|    454|   170|204|       Ver|
+-------+----+----+----+---+---+---+---+---+---+---+-------+-------+------+---+----------+
```

Figure 19. describing the data

Now to check if all the states have given the same amount of data in the span of 5 years to check this we use the groupBy() function and count() the available data in each state. This can indicate the missing data from the state

```
In [63]:
1 df3=df.groupBy('City').count().sort("count", asending = True).show(50)
2 df4=df.select ("City").distinct().count()
3 print(f"The total number of cities is : {df4}")
```

```
+------------------+-----+
|              City|count|
+------------------+-----+
|            Aizawl|  113|
|             Kochi|  162|
|         Ernakulam|  162|
|            Bhopal|  289|
|        Chandigarh|  304|
|          Shillong|  310|
|        Coimbatore|  386|
|          Guwahati|  502|
|           Kolkata|  814|
|           Talcher|  925|
|      Brajrajnagar|  938|
|         Amaravati|  951|
|Thiruvananthapuram| 1112|
|            Jaipur| 1114|
|        Jorapokhar| 1169|
|          Amritsar| 1221|
|     Visakhapatnam| 1462|
|          Gurugram| 1679|
|             Patna| 1858|
|         Hyderabad| 2006|
|           Chennai| 2009|
|           Lucknow| 2009|
|         Ahmedabad| 2009|
|            Mumbai| 2009|
|             Delhi| 2009|
|         Bengaluru| 2009|
+------------------+-----+

The total number of cities is : 26
```

Figure 20. Showing the number of data in each state respectively

Sub Conclusion

1. Some values are indicating very high readings which are not possible in a real situation this indicates that the data should be subjected to validation.

2. Few states have not produced data from early 2015 to 2017

3. Although it is normal to have anomalies in a dataset It is found that this dataset has several anomalies and should be addressed before implementing ML and Visualisation.

4.Machine Learning

4.1 Regression

Regression in machine learning consists of several mathematical equations which allow the data scientist to predict an outcome. In this course work, we will be using multilinear regression, random forest regression, polynomial regression, decision tree regression and find out which is the best suited for predicting AQI

The codes which are used in this part of the coursework was referred from different sources(apache spark MLlib guide, GitHub, YouTube) the code is then modified with the needs and errors in the dataset

Note: all the codes are available in the appendix to run

In [1]:
```
import numpy as np
import pandas as pd
from pyspark import SparkContext
sc = SparkContext.getOrCreate()

df = pd.read_csv("city_day.csv")
df2= df.sample(frac =.05)
df2=df2.drop(['City','Date','AQI_Bucket'], axis = 1)
df2=df2.dropna().astype(np.int64)

x1 = df2.iloc[:,:12].values
z1 = pd.DataFrame(x1)

y1 = df2.iloc[:,12:13].values
z2 = pd.DataFrame(y1)

from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder()
x_new1 = pd.DataFrame(ohe.fit_transform(x1[:,[0]]).toarray())
x_new2 = pd.DataFrame(ohe.fit_transform(x1[:,[1]]).toarray())
x_new3 = pd.DataFrame(ohe.fit_transform(x1[:,[2]]).toarray())

feature_set = pd.concat([x_new1,x_new2,x_new3,pd.DataFrame(x1[:,5:12])],axis=1,sort=False)

df2.dropna(inplace=True)
```

figure 21 importing dataset into pyspark

As we can see in the above fig 21 we are importing required packages and the dataset into pyspark and the next line indicates all the errors and anomalies addressed in the dataset.
Due to computational constraints and the new architecture of mac os, it was only possible to do the task with only 10% of the data. Using more than 10% crashed the kernel automatically

The oneHotencoder is used to convert the string input columns to numbers which can be used in the ML algorithm.

There were several errors and anomalies while working with this dataset the most difficult ones were seen in figure 22

```
    ensure_2d=ensure_2d,
    allow_nd=allow_nd,
    ensure_min_samples=ensure_min_samples,
    ensure_min_features=ensure_min_features,
    estimator=estimator,
)
y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric)
check_consistent_length(X, y)

File ~/.local/share/virtualenvs/roshandhanashekeran-qt-aBSK2/lib/python3.8/site-packages/sklearn/utils/validation.p
y:800, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, a
llow_nd, ensure_min_samples, ensure_min_features, estimator)
        raise ValueError(
            "Found array with dim %d. %s expected <= 2."
            % (array.ndim, estimator_name)
        )
    if force_all_finite:
--> 800     _assert_all_finite(array, allow_nan=force_all_finite == "allow-nan")
if ensure_min_samples > 0:
    n_samples = _num_samples(array)

File ~/.local/share/virtualenvs/roshandhanashekeran-qt-aBSK2/lib/python3.8/site-packages/sklearn/utils/validation.p
y:114, in _assert_all_finite(X, allow_nan, msg_dtype)
    if (
        allow_nan
        and np.isinf(X).any()
        or not allow_nan
        and not np.isfinite(X).all()
    ):
        type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114     raise ValueError(
            msg_err.format(
                type_err, msg_dtype if msg_dtype is not None else X.dtype
            )
        )
# for object dtype data, we only check for NaNs (GH-13254)
elif X.dtype == np.dtype("object") and not allow_nan:

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

Figure 22 Errors in the dataset

The NaN values were removed from the dataset by using a simple code df2=df2.dropna() and the data type float64 was converted inti int64 by using df2=df2.astype(np.int64) and we use the info() function to have a preview of the data

```
In [2]:  1 df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 632 entries, 7640 to 10259
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   PM2.5    632 non-null    int64
 1   PM10     632 non-null    int64
 2   NO       632 non-null    int64
 3   NO2      632 non-null    int64
 4   NOx      632 non-null    int64
 5   NH3      632 non-null    int64
 6   CO       632 non-null    int64
 7   SO2      632 non-null    int64
 8   O3       632 non-null    int64
 9   Benzene  632 non-null    int64
 10  Toluene  632 non-null    int64
 11  Xylene   632 non-null    int64
 12  AQI      632 non-null    int64
dtypes: int64(13)
memory usage: 69.1 KB
```

Figure 23 information about the data set after error correction

We start by importing all the necessary modules from sklearn.

```
In [2]:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from math import sqrt

x_train,x_test,y_train,y_test = train_test_split(feature_set,y1,test_size=0.25,random_state=0)

```

Figure 24 importing modules

Primarily we split the data into two train data and test data the train data requires 75% of the data and the test will take the remaining 25% and we name them x train and y tarin for training and x test and y test for testing the data

4.1.1 Test data prediction

4.1.2 Multiple linear regression model

```
In [3]:
#------test data prediction--------#


# multiple linear regression model
mreg = LinearRegression()
mreg.fit(x_train,y_train)

mlr_y_predict = mreg.predict(x_test)


```

Figure 25 code for MLR

MLR is also known as multi linear regression is a type of regression that uses multiple variables to predict a response variable. The drawback of using MLR comes down to the dataset if the dataset has empty data the MLR will give us a false conclusion.

```
In [40]:
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual VS Predicted')
```

```
Out[40]: Text(0.5, 1.0, 'Actual VS Predicted')
```

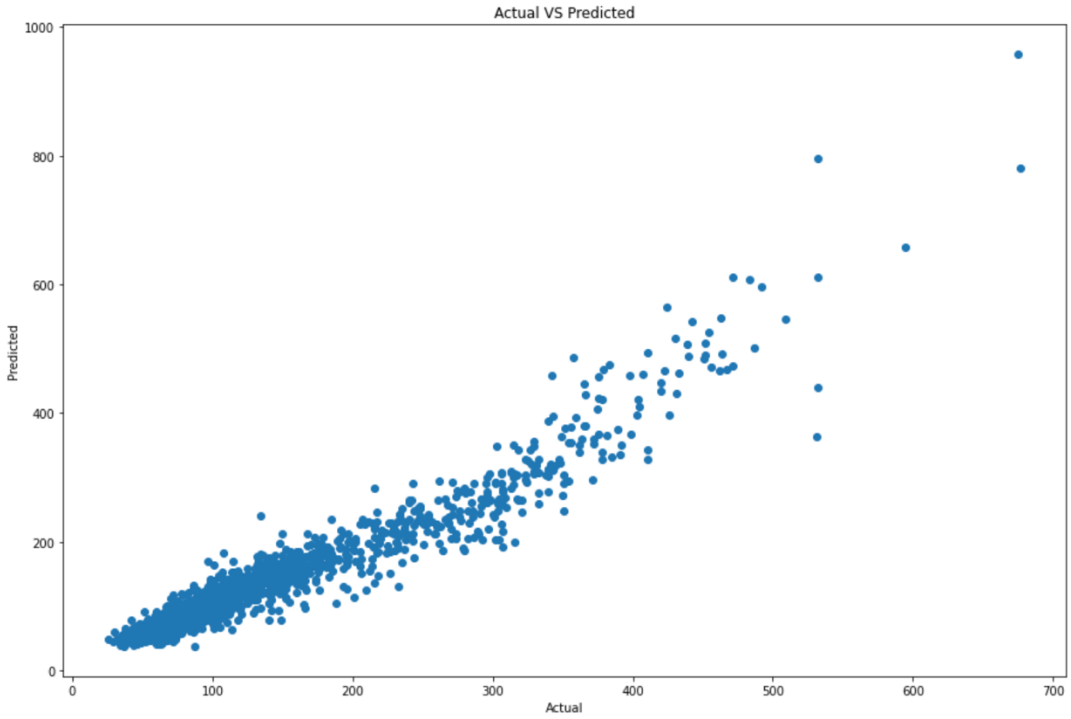


Figure 26 MLR plotting

As we can see our model has given close relation and we are able to predict AQI with less Error

```
In [44]:
pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred ,'Difference': y_test-y_pred })
pred_y_df[0:10]
```

Out[44]:

	Actual Value	Predicted value	Difference
12081	365	379.179482	-14.179482
19784	51	52.145679	-1.145679
19519	61	82.518915	-21.518915
14879	216	283.214811	-67.214811
28665	169	159.210452	9.789548
2910	63	79.981366	-16.981366
3993	64	64.120332	-0.120332
11545	102	100.455865	1.544135
10529	333	275.163956	57.836044
2343	70	63.722341	6.277659

The result shoes the actual value and predicted values are close to each other by subtracting the actual values and predicted values .

4.1.3 polynomial regression model

The polynomial regression is used to find the relationship between X and Y it provides a result with grate defined relationship between the independent and dependent variables but working with polynomial regression can sometimes be tuff because it takes a longer time to process data and needs more computational power to execute when working with huge datasets.

```
In [4]:
# ----------------------------------------
# polynomial regression model
# degree = 2
poly_reg = PolynomialFeatures(degree = 2)
preg = LinearRegression()
pf = poly_reg.fit_transform(x_train)
preg.fit(pf,y_train)

pr_y_predict = preg.predict(poly_reg.fit_transform(x_test))
```

Figure 27 code for polynomial regression model

4.1.4 Random forest regression

The random forest regression is a technique used for both regression and classification as the name suggest the random forest tree contains multiple decision trees on the various subset of the given dataset and takes the average to improve the predictive accuracy. Random forest tree also takes less training time compared with others it has high accuracy, it can also use large datasets

```
In [7]:
# random forest regression model
# random forest with 500 trees

rt_reg = RandomForestRegressor(n_estimators = 500, random_st
rt_reg.fit(x_train,y_train)

rt_y_predict = rt_reg.predict(x_test)
```

Figure 28 Random forest regression

```
In [84]:
1 pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred_rf ,'Difference': y_test-y_pred_rf })
2 pred_y_df[0:10]
```

Out[84]:

	Actual Value	Predicted value	Difference
12081	365	380.022	-15.022
19784	51	49.046	1.954
19519	61	77.462	-16.462
14879	216	448.976	-232.976
28665	169	177.674	-8.674
2910	63	61.702	1.298
3993	64	50.746	13.254
11545	102	106.070	-4.070
10529	333	304.204	28.796
2343	70	69.738	0.262

Figure 29 Difference between predicted and actual values

```
In [83]:
1
2 import matplotlib.pyplot as plt
3 plt.figure(figsize=(15,10))
4 plt.scatter(y_test,y_pred_rf)
5 plt.xlabel('Actual')
6 plt.ylabel('Predicted')
7 plt.title('Actual VS Predicted')
8
9
```

Out[83]: Text(0.5, 1.0, 'Actual VS Predicted')

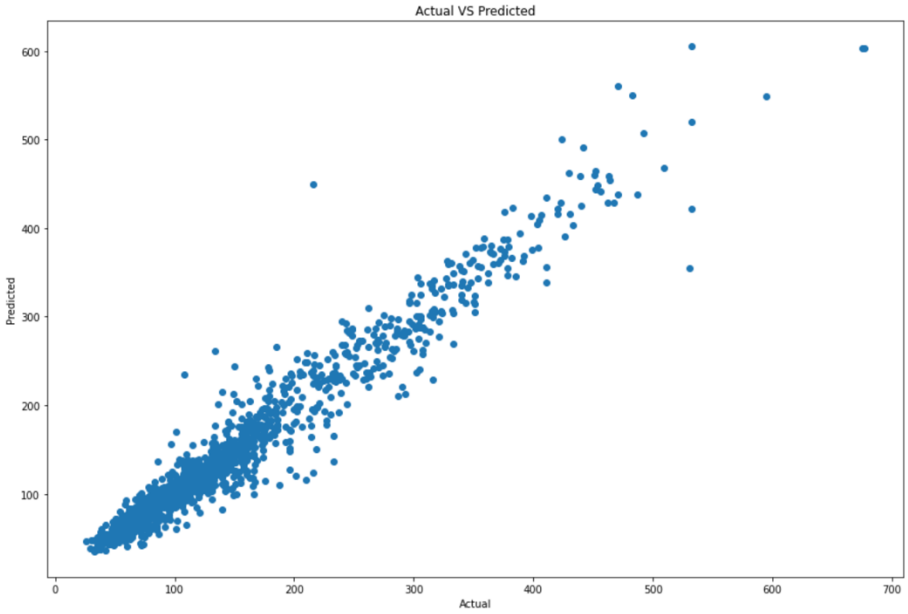


Figure 30 Random forest plot

The data and the plot is a clear evidence that random forest regression can be used to find a good relation Between the predicted and the actual values

4.1.5 Decision tree regression model

The decision tree regression model brake down the dataset into smaller subsets

```
In [6]:
#-----------------------------------------------
# decision tree regression model

dec_tree = DecisionTreeRegressor(random_state = 0)
dec_tree.fit(x_train,y_train)

dt_y_predict = dec_tree.predict(x_test)

```

Figure 31 Decision tree regression

```
In [90]:
pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred_rf ,'Difference': y_test-y_pred_dc })
pred_y_df[0:10]
```

Out[90]:

	Actual Value	Predicted value	Difference
12081	365	380.022	-1.0
19784	51	49.046	7.0
19519	61	77.462	-27.0
14879	216	448.976	-286.0
28665	169	177.674	-20.0
2910	63	61.702	-3.0
3993	64	50.746	19.0
11545	102	106.070	1.0
10529	333	304.204	35.0
2343	70	69.738	-3.0

Figure 32 difference between predicted and actual value

```
In [89]:
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred_dc)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual VS Predicted')


```

Out[89]: `Text(0.5, 1.0, 'Actual VS Predicted')`

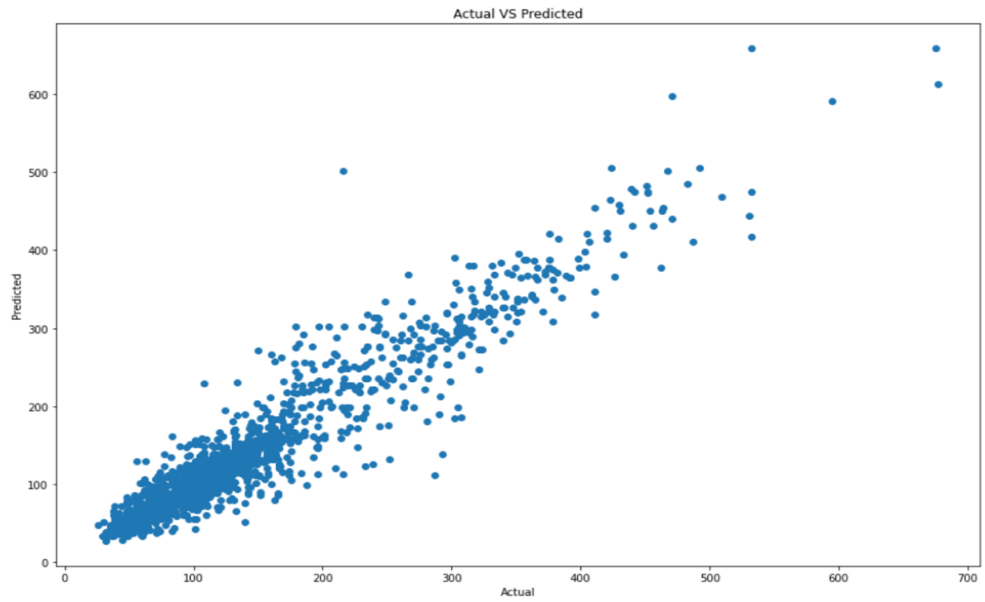


Figure 33 Decision tree plotting

By using the Decision tree regression we were able to get the required output.

Before moving forward some definitions for RMSE, MAE, RMSLE, R^2 (appendix page 44)

The residual error is defined as difference between the actual and the predicted values

```
# error estimation methods
#-----------------------------------------------

from math import sqrt
from sklearn import metrics

def rmsle(real, predicted):
    sum=0.0
    for x in range(len(predicted)):
        if predicted[x]<0 or real[x]<0:
            continue
        p = np.log(predicted[x]+1)
        r = np.log(real[x]+1)
        sum = sum + (p - r)**2
    return ((sum/len(predicted))**0.5)[0]
```

4.1.6 Testing and training data prediction

```
In [11]:
#----- multiple linear regresion -------
rmse_mlr = sqrt(metrics.mean_squared_error(y_test, mlr_y_predict))
mae_mlr = metrics.mean_absolute_error(y_test, mlr_y_predict)
r2_mlr = metrics.r2_score(y_test,mlr_y_predict)
rmsle_mlr = rmsle(y_test,mlr_y_predict)

#----- polynomial regression ------------
rmse_pr = sqrt(metrics.mean_squared_error(y_test, pr_y_predict))
mae_pr = metrics.mean_absolute_error(y_test, pr_y_predict)
r2_pr = metrics.r2_score(y_test,pr_y_predict)
rmsle_pr = rmsle(y_test,pr_y_predict)

#----- decision tree regression ---------
rmse_dt = sqrt(metrics.mean_squared_error(y_test, dt_y_predict))
mae_dt = metrics.mean_absolute_error(y_test, dt_y_predict)
r2_dt = metrics.r2_score(y_test,dt_y_predict)
rmsle_dt = rmsle(y_test,dt_y_predict)

#----- random forest regression ---------
rmse_rt = sqrt(metrics.mean_squared_error(y_test, rt_y_predict))
mae_rt = metrics.mean_absolute_error(y_test, rt_y_predict)
r2_rt = metrics.r2_score(y_test,rt_y_predict)
rmsle_rt = rmsle(y_test,rt_y_predict)

```

Figure 34 Testing data prediction code

We do the same with the training data and bring out the results

In [12]:

```
# ---- MLR ------
mlr_ytp_rmse = sqrt(metrics.mean_squared_error(y_train, mreg.predict(x_train)))
mlr_ytp_mae = metrics.mean_absolute_error(y_train, mreg.predict(x_train))
mlr_ytp_r2 = metrics.r2_score(y_train, mreg.predict(x_train))
m1 = mreg.predict(x_train)
mlr_ytp_rmsle = rmsle(y_train, m1)
#------ polynomial regression ---------
pr_ytp_rmse = sqrt(metrics.mean_squared_error(y_train, preg.predict(poly_reg.fit_transform(x_train))))
pr_ytp_mae = metrics.mean_absolute_error(y_train, preg.predict(poly_reg.fit_transform(x_train)))
pr_ytp_r2 = metrics.r2_score(y_train, preg.predict(poly_reg.fit_transform(x_train)))
pr_ytp_rmsle = rmsle(y_train, preg.predict(poly_reg.fit_transform(x_train)))

#mxp = preg.predict(poly_reg.fit_transform(x_train))

# ----- decision tree reg ------
dt_ytp_rmse = sqrt(metrics.mean_squared_error(y_train, dec_tree.predict(x_train)))
dt_ytp_mae = metrics.mean_absolute_error(y_train, dec_tree.predict(x_train))
dt_ytp_r2 = metrics.r2_score(y_train, dec_tree.predict(x_train))
dt_ytp_rmsle = rmsle(y_train, dec_tree.predict(x_train))

# ----- random forest reg -----
rf_ytp_rmse = sqrt(metrics.mean_squared_error(y_train, rt_reg.predict(x_train)))
rf_ytp_mae = metrics.mean_absolute_error(y_train, rt_reg.predict(x_train))
rf_ytp_r2 = metrics.r2_score(y_train, rt_reg.predict(x_train))
rf_ytp_rmsle = rmsle(y_train, rt_reg.predict(x_train))

```

Figure 35 Training data prediction code

Since we have all the values let's print out the results and compare which model is better suited for predicting AQI

In [49]:

```

#---------- RESULT ----------


print("evaluating on training data:")
print("models\tR^2\tRMSE\tMAE\tRMSLE")
print("MLR\t{3:.2f}\t{3:.2f}\t{3:.2f}\t{3:.2f}".format(mlr_ytp_r2,mlr_ytp_rmse,mlr_ytp_mae,mlr_ytp_rmsle))
print("PR\t{1:.2f}\t{1:.2f}\t{1:.3f}\t{1:.4f}".format(pr_ytp_r2,pr_ytp_rmse,pr_ytp_mae,pr_ytp_rmsle))
print("DTR\t{1:.4f}\t{1:.4f}\t{1:.4f}\t{1:.4f}".format(dt_ytp_r2,dt_ytp_rmse,dt_ytp_mae,dt_ytp_rmsle))
print("RFR\t{1:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(rf_ytp_r2,rf_ytp_rmse,rf_ytp_mae,rf_ytp_rmsle))
```

```
evaluating on training data:
models  R^2     RMSE    MAE     RMSLE
MLR     0.12    0.12    0.12    0.12
PR      0.00    0.00    0.000   0.0000
DTR     0.0000  0.0000  0.0000  0.0000
RFR     18.2648 18.2648 11.7249 0.1163
```

In [50]:

```
print("evaluating on testing data:")
print("models\tR^2\tRMSE\tMAE\tRMSLE")
print("MLR\t{3:.2f}\t{3:.2f}\t{3:.2f}\t{3:.2f}".format(r2_mlr,rmse_mlr,mae_mlr,rmsle_mlr))
print("PR\t{0:.2f}\t{1:.2f}\t{2:.3f}\t{3:.4f}".format(r2_pr,rmse_pr,mae_pr,rmsle_pr))
print("DTR\t{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(r2_dt,rmse_dt,mae_dt,rmsle_dt))
print("RFR\t{1:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(r2_rt,rmse_rt,mae_rt,rmsle_rt))
```

```
evaluating on testing data:
models  R^2     RMSE    MAE     RMSLE
MLR     7.55    7.55    7.55    7.55
PR      0.49    63.58   41.536  0.4738
DTR     0.4805  64.0374 42.8226 0.4526
RFR     50.4290 50.4290 33.2146 0.3377
```

Figure 36. Results of testing and training data

As we can see the testing data had zero values in polynomial regression and decision tree regression which means The regression model couldn't establish a relationship between the dependent variable y and the independent variable x. This may be the result of using only fewer data. Whereas the training data has found a good relationship.

Sub conclusion

1) It was clear that regression models can help to predict AQI.

2) Computational power plays a major role in doing so

3) It was very difficult to find a relationship since the data is inconsistent

4) It was found that polynomial regression was difficult to plot and run the regression due to lack of memory

5. Data Visualization

5.1 Importing data into Tableau

Tableau is a tool used for interactive data visualization software, Tableau helps make Big Data small, and small data insightful and actionable. The main use of tableau software is to help people see and understand their data.

First and foremost we open tableau and import our CSV file into the tableau and look for necessary misinterpretation

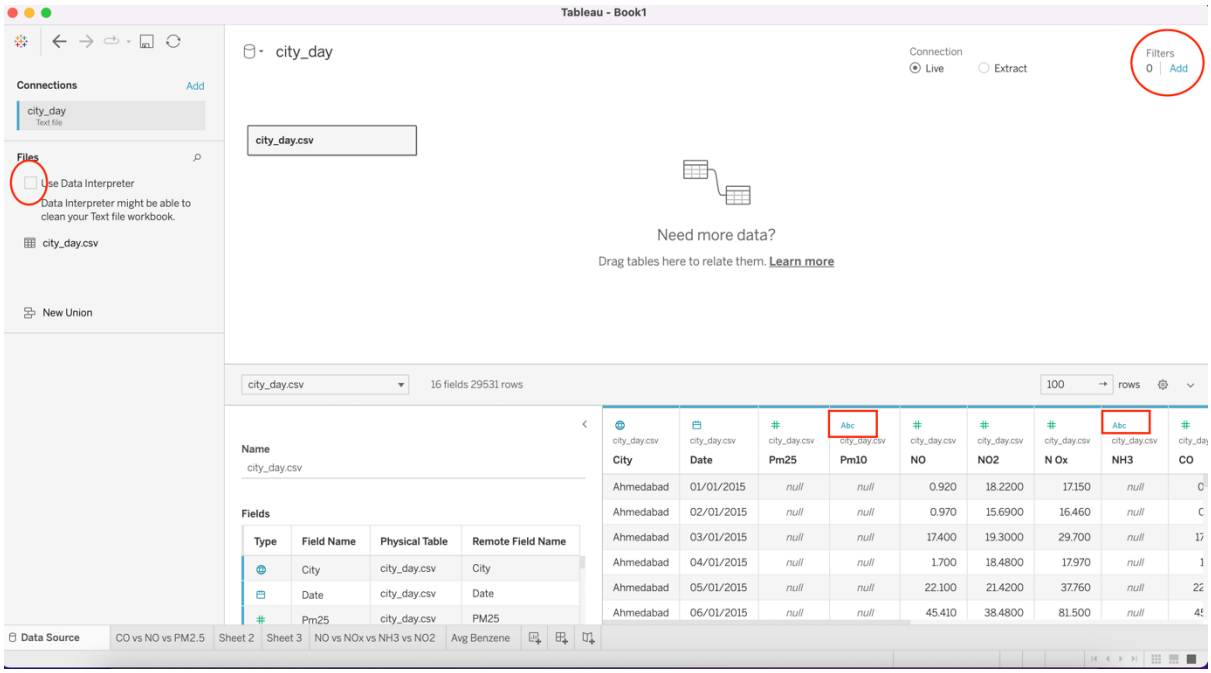


Figure 37.Tableau dashboard

As shown in the figure we can use the data interpreter option this will automatically check our data or we can also use the filter option in the top right corner to clear out the null data in the dataset. The main thing to be noticed before starting to work with the data is we need to check if all the data variables are correct sometime tableau interprets int as strings and vice versa we can toggle the settings and change if necessary.

5.2.1 Geographical representation

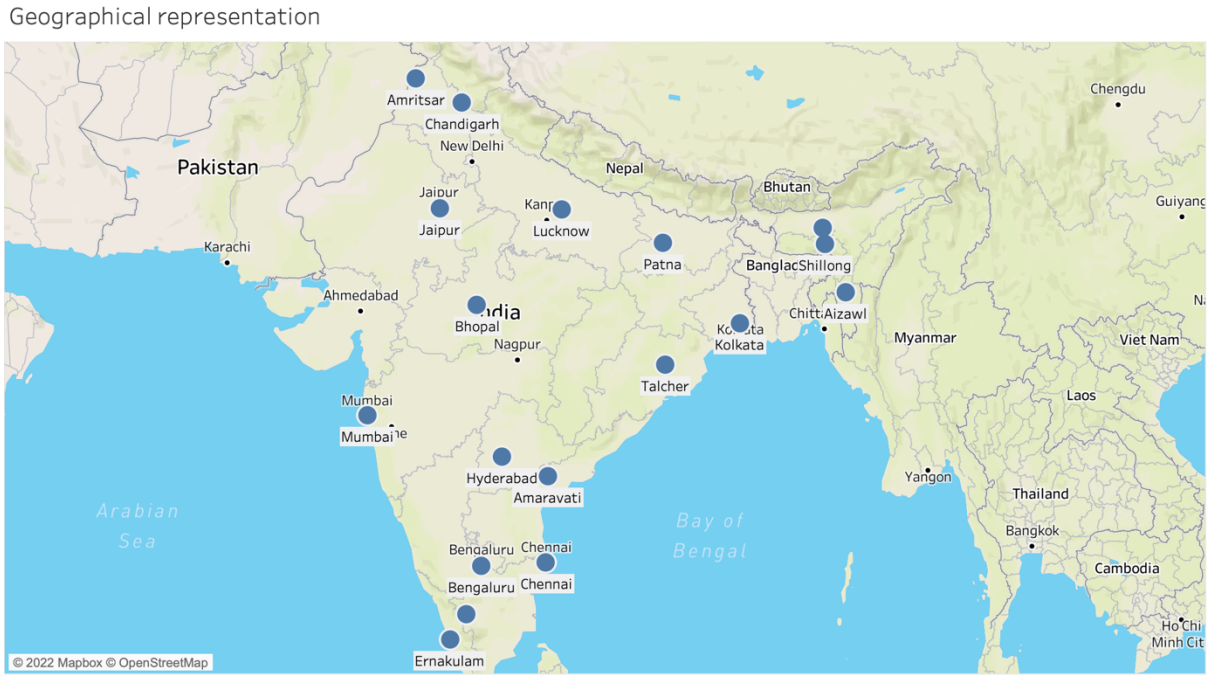


Figure 38. India outline

In this chart, we locate all the 26 states in India and we will be using all the stations for our visualization. This chart will later be useful to know why a particulate pollutant was very high in a particular place and not in other states, geographical location plays a major role in analysing pollution and deriving the cause.

5.2.2 Representation Of Particulate Matter

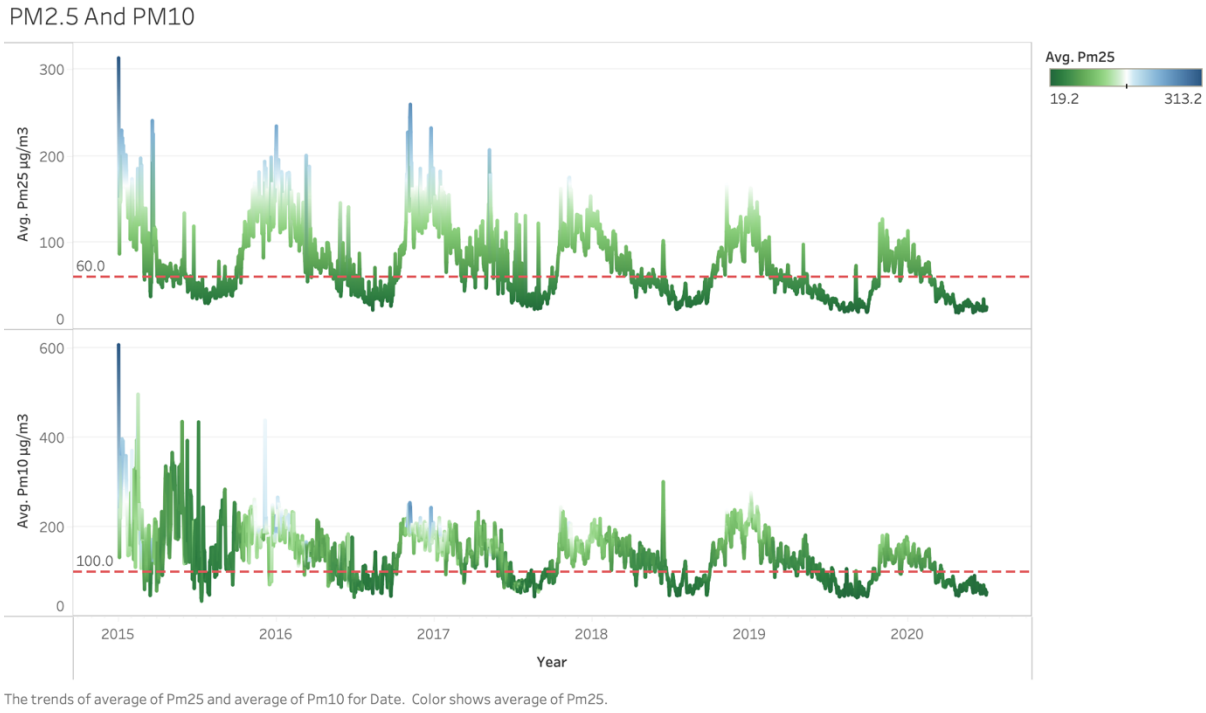


Figure Representation of particulate matter from the year 2015-2020

The PM2.5 and PM10 levels in ambient air for the period 2015-20 reveal that there is a decreasing trend in pollution levels of Pm 2.5 and Pm10. It is also found that the PM 2.5 and Pm10 levels exceed the 24 h NAAQ standards of 60 and 100 µg/m^3 respectively.

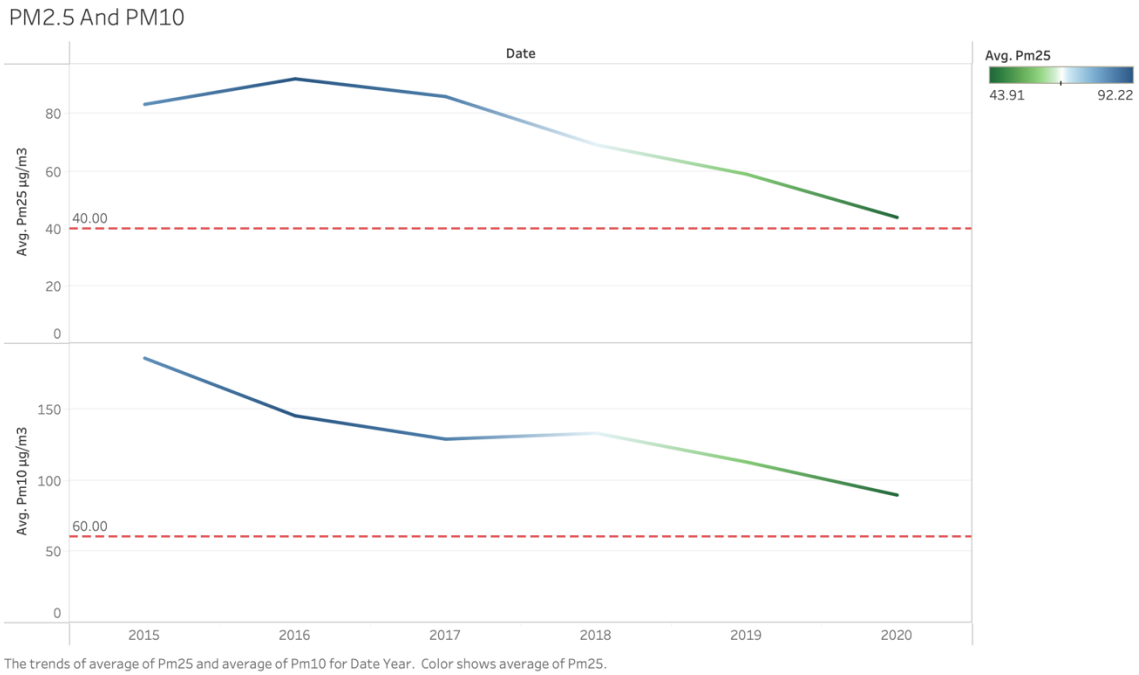
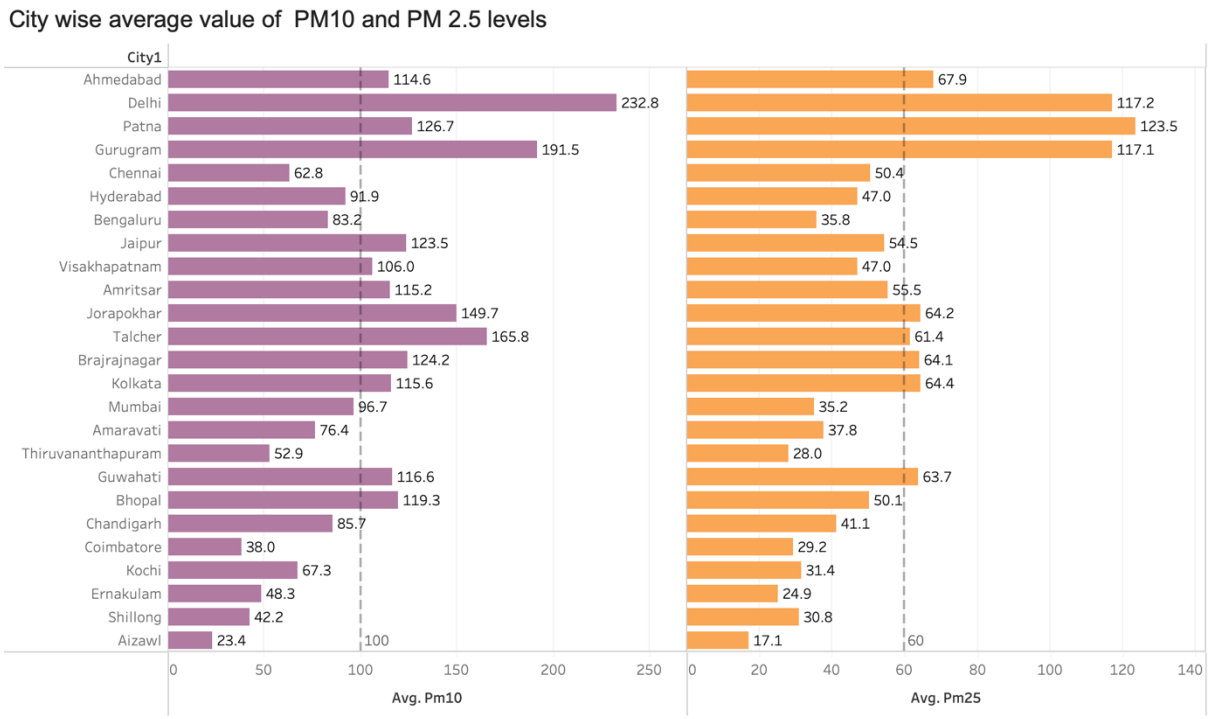


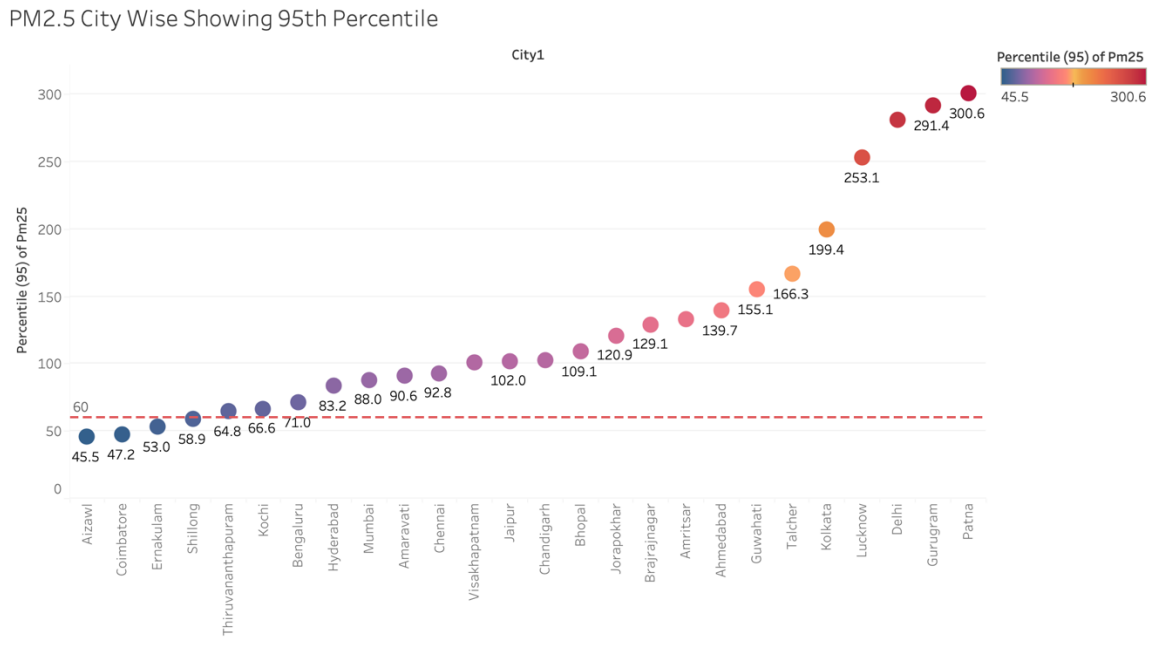
Figure39 Representation yearly data of particulate matter



Average of Pm10 and average of Pm25 for each City1. For pane Average of Pm10: The marks are labeled by average of Pm10. For pane Average of Pm25: The marks are labeled by average of Pm25. The view is filtered on average of Pm10, which keeps non-Null values only.

Figure 40 Average PM10 and PM2.5 city wise

The PM2.5 and PM10 levels in ambient air for the period 2015-20 reveal that there is a decreasing trend in pollution levels of PM 2.5 and PM10. It is also found that the PM 2.5 and Pm10 levels exceed the Annual NAAQ standards of 40 and 60 µg/m^3 respectively. Hens the dataset reveals that the above cities fall under the non-attainment to comply with ambient air quality standards in ambient air.

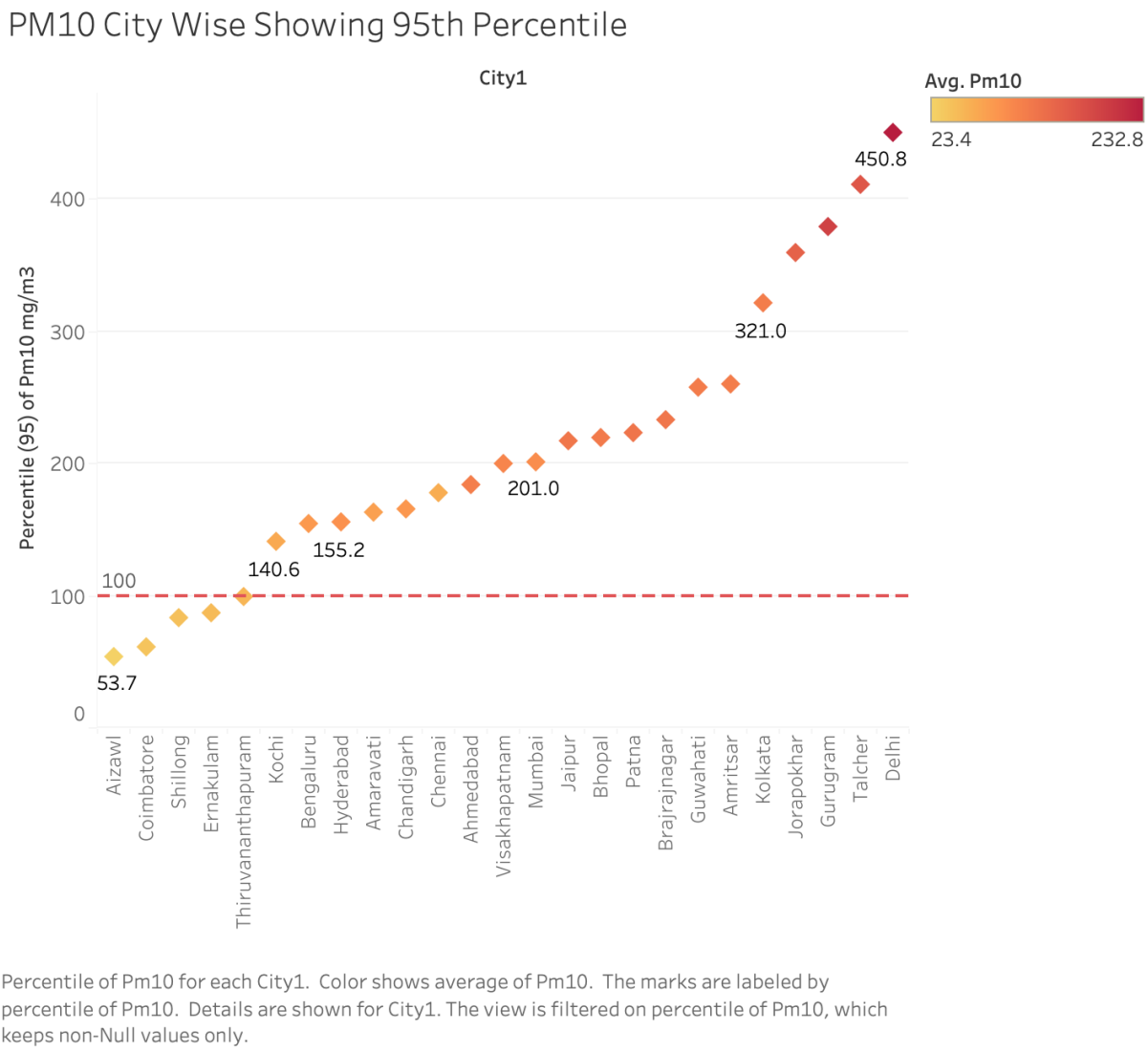


Percentile of Pm25 for each City1. Color shows percentile of Pm25. The marks are labeled by percentile of Pm25.

Figure41. 95th percentile of PM2.5

The 95th percentile reveals a value recorded 95% of the time in each city while 5% of the value exceeded the 95th percentile of the value. As the graph reveals only 4 cities comply with the 24 hours NAAQ standards with respective 95Th percentile. The rest of the cities need to be addressed and prioritized to comply with the NAAQ standards

The data also shows Delhi and Gurugram emitting high levels of PM2.5 consistently for 5 years. The main reason behind high levels of PM2.5 emission can be related to vehicular pollution. Another aspect to be noticed is the downtrend which indicates that the introduction of EV vehicles has played a major impact on reducing PM levels in ambient air.



The figure42 *95th percentile of PM10*

Emissions from the combustion of gasoline, oil, diesel fuel, or wood are major contributors of PM10 we can see the data indicating 5 states well within the limits are less populated cities on the contrary densely packed cities like Delhi, Bangalore, Kolkata indicate high levels of PM10 emission and should be addressed.

5.2.3 Representation Nitrogen And its Oxides

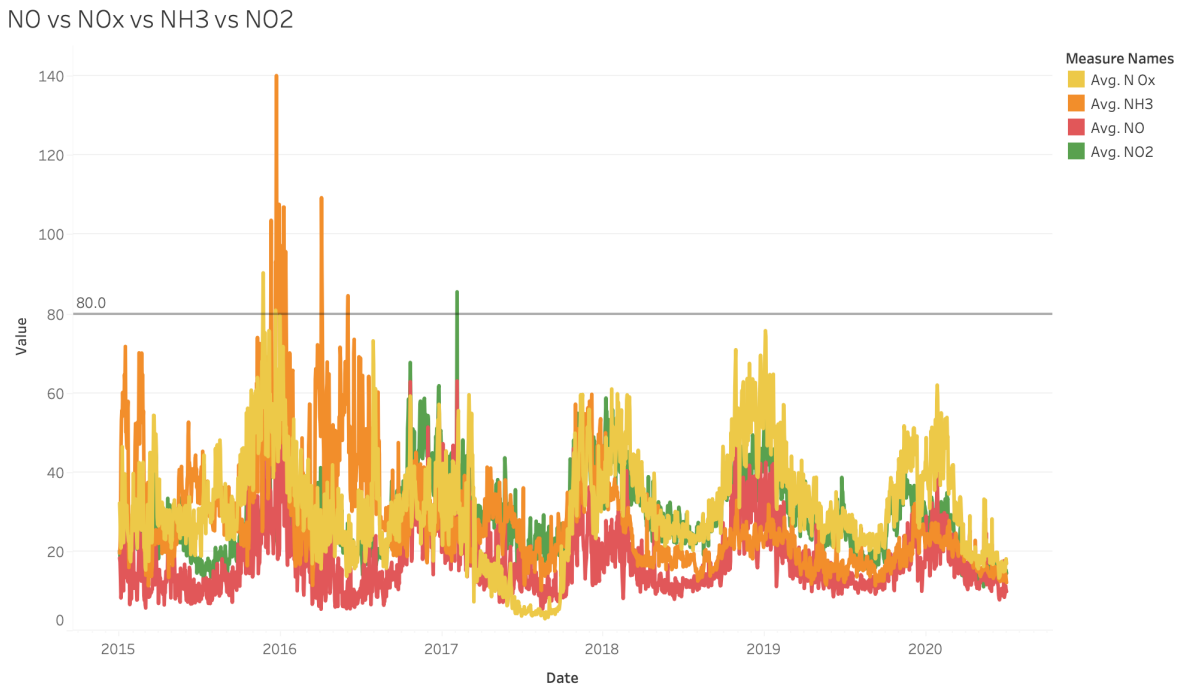


Figure43 Average NO, NOx, NH3, NO2 for the period 2015-20

As the chart depicts the NO2 levels comply with the 24-hour NAAQ standard of 80µg/m^3 The other parameters namely NO, NOX, NH3, NO2 reveal that there is decreasing trend from the year 2015 to 2020. The ammonia in ambient air for the period 2015-20 reveals that it complies with the 24 hours NAAQs standards of 400µg/m^3

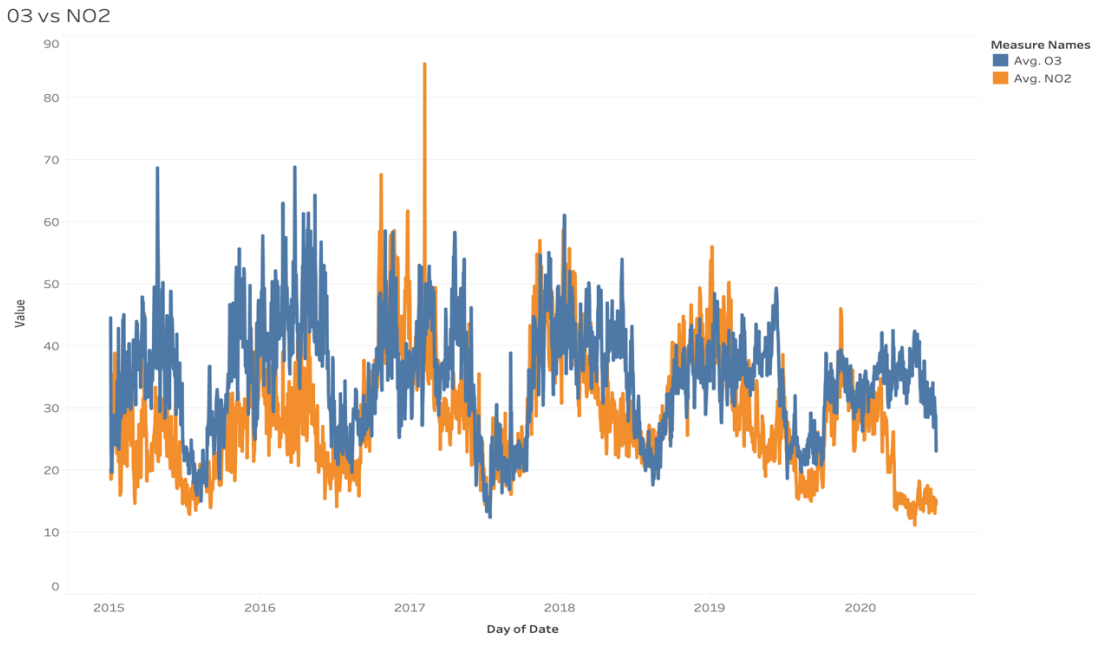


Figure 44 Correlation between NO2 and Ozone

The above data shows that O3 comes under the National standards and the correlation between NO2 and ozone(O3) reveals that as ozone increases NO2 decreases. and the data shows there is not much significance to the ambient air quality deterioration of ozone levels.

5.2.4 Representation of sulphur and Carbon monoxide

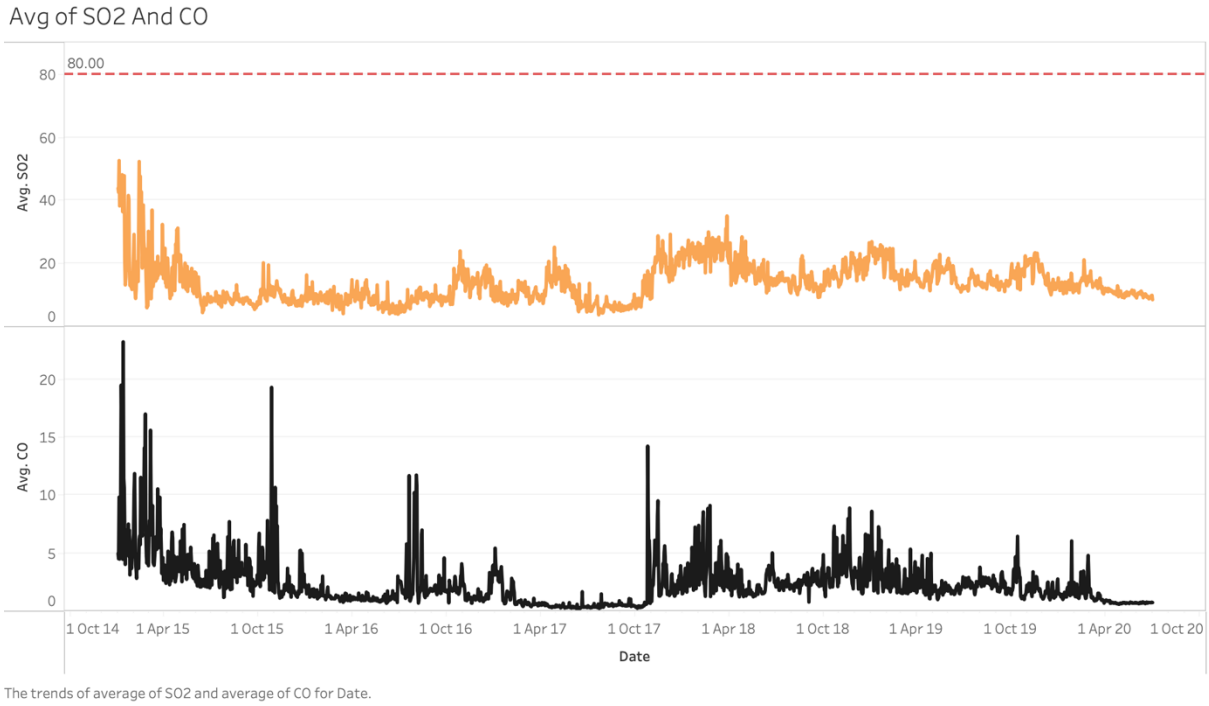


Figure 45 Average SO2 and CO

SO2 levels in ambient air comply with the 24 hours standards of NAAQ of 80 µg/m^3 Therefore the levels of SO2 in ambient air are insignificant.

The AAQ standers for CO is 4 mg/m^3 for 8 hours and 2 mg/m^3 for 1 hour. The data reveals 24 hours average data for the period 2015 to 2020. The maximum data reported is greater than 25 mg/m^3, if such high levels of CO existed in ambient air would have resulted in morbidity and mortality in sensitive locations it is felt that the data needs to be validated for CO as very high values are recorded in the given data.

5.2.5 Representation of Benzene, Toluene, Xylene

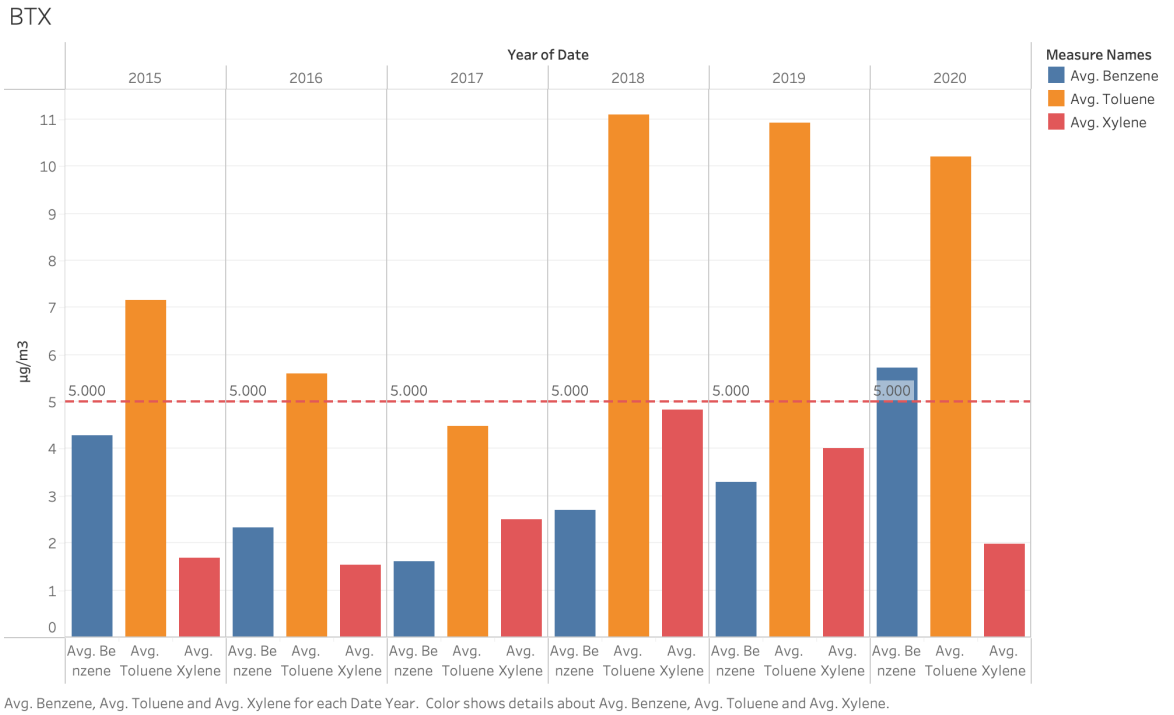


Figure 46 Annual Average values of benzene, toluene, xylene

Benzene, Toluene, Xylene (BTX) exists in ambient air because of vehicular pollution. The Benzene levels from the data set show an increasing trend from the year 2017 and are also found to exceed the annual standard of 5 mg/m3 for the year 2020.

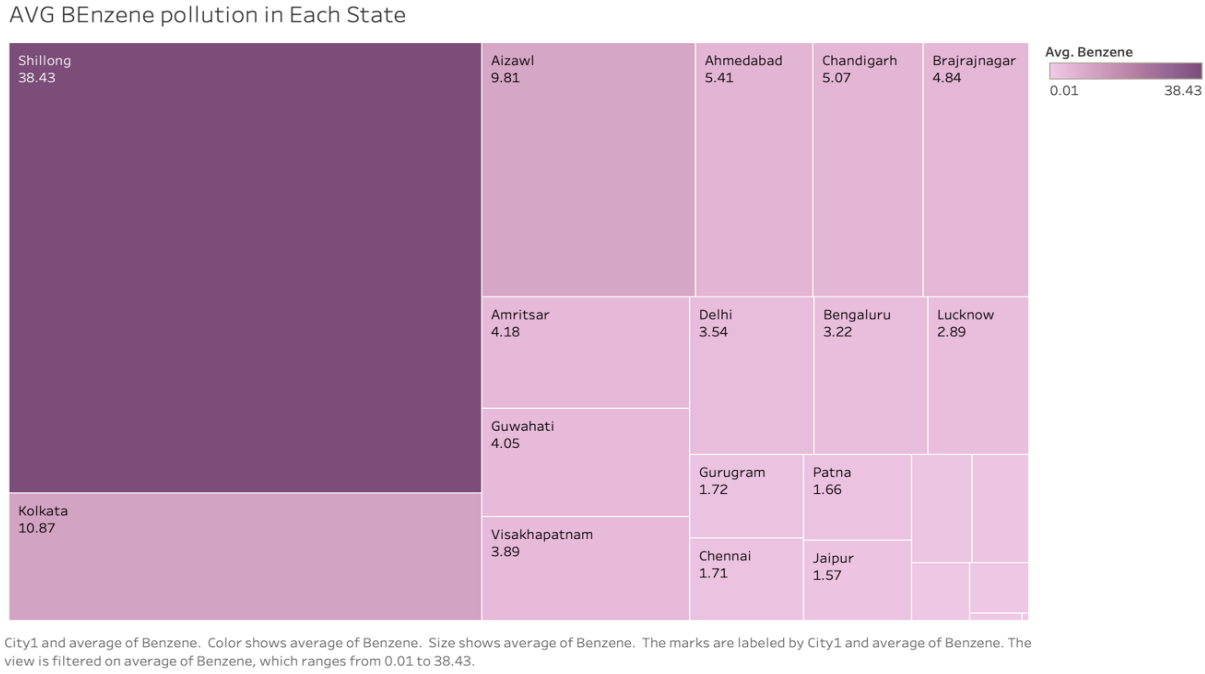
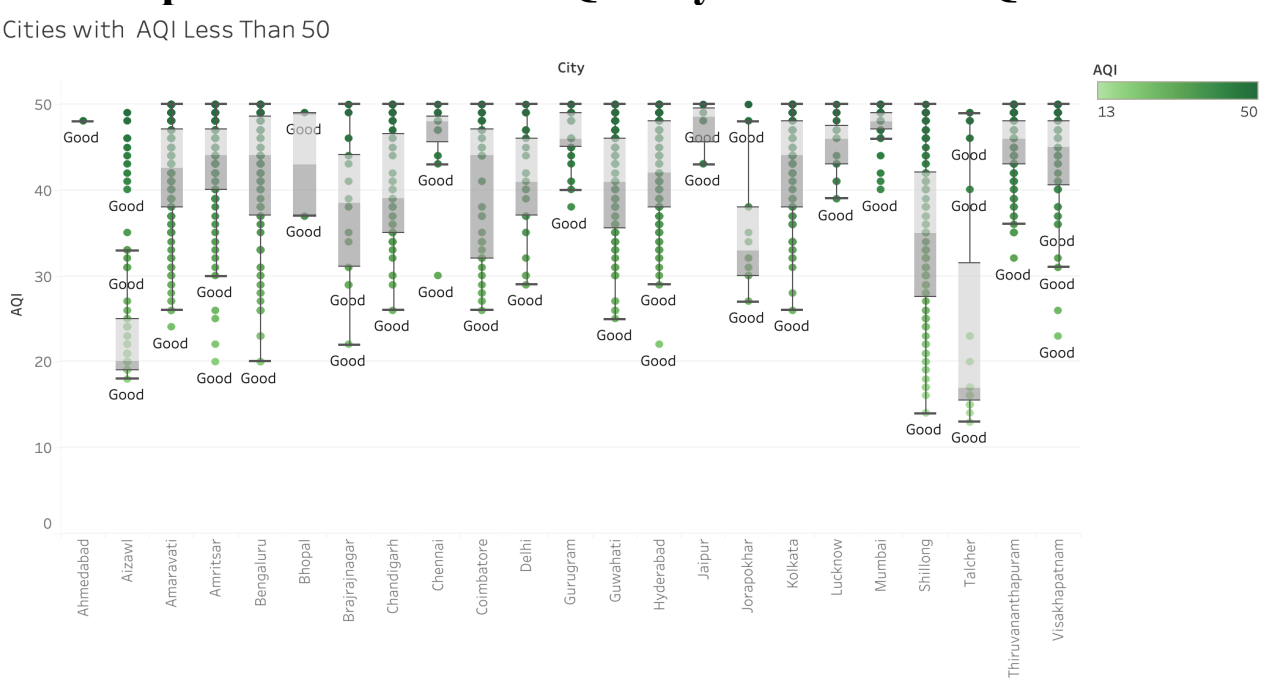


Figure 47 Average benzene pollution in each state.

The above graph shows that the city of Shillong, Kolkata, and Aizawl have only crossed the annual average where areas others have stayed well within the limits. The reason behind Shillong being the highest emitter of benzene even though it is located in a hill station and less populated can be because of lack of airflow therefore the pollutant doesn't travel and stays in the atmosphere this will intern show abnormal readings in the particular place.

5.2.6 Representation of Air Quality Index and AQI Bucket



AQI for each City. Color shows AQI. The marks are labeled by AQI Bucket. Details are shown for City. The data is filtered on AQI, which keeps non-Null values only. The view is filtered on AQI Bucket, which excludes Moderate, Poor, Satisfactory, Severe and Very Poor.

Figure 48 shows Air Quality Index That Falls Under Good

The term AQI represents the air quality index and the term AQI Bucket is represented by 6 categories shown in table 3

AQI	Associated Health Impacts
Good (0–50)	Minimal Impact
Satisfactory (51–100)	May cause minor breathing discomfort to sensitive people.
Moderately polluted (101–200)	May cause breathing discomfort to people with lung disease such as asthma, and discomfort to people with heart disease, children, and older adults.
Poor (201–300)	May cause breathing discomfort to people on prolonged exposure, and discomfort to people with heart disease
Very Poor (301–400)	May cause respiratory illness to the people on prolonged exposure. The effect may be more pronounced in people with lung and heart diseases.
Severe (401-500)	May cause respiratory impact even on healthy people, and serious health impacts on people with lung/heart disease. The health impacts may be experienced even during light physical activity.

Table 3 AQI Standards

The above box plot reveals that the above cities AQI is found to be within 50 µg/m^3

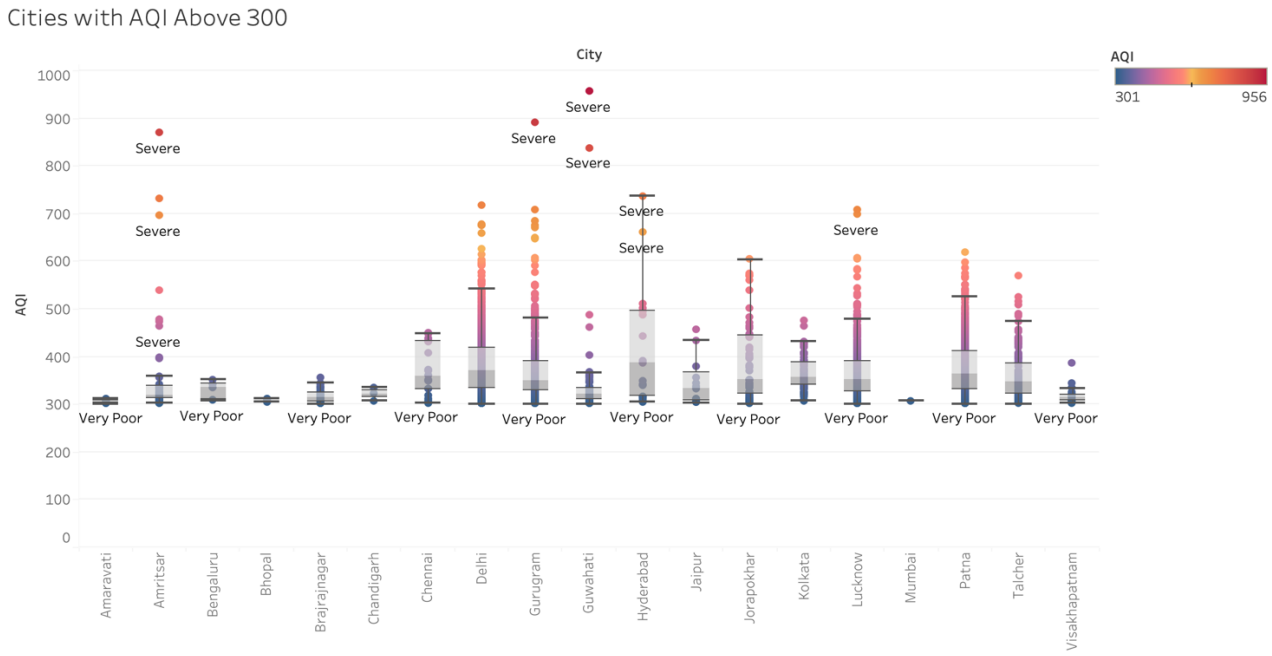


Figure 49 Showing Air Quality Index for cities with very poor and severe AQI

The above box plot reveals that the AQI is found to be greater than 300 µg/m^3 and falls in the category of severe to very poor. The above cities AQI will have a morbidity effect on sensitive populations in the above cities.

6. Discussion

1) Because of dynamic and volatile nature and unpredictable nature of the location and time of the pollutants and particles predicting air quality is very Difficult (A Machine Learning Approach to Predict Air Quality in California)

2) On keen inspection, it was found that the data had high variability with very unusual readings which are not applicable in a real scenario

3) Due to computational power and time constraints in this model, only 20% of the data was used and the regression methods used were multilinear regression, random forest regression, decision tree regression, polynomial regression to find the outputs.

4) The best results were taken from this model where the random forest regression and multilinear regression gave the best results and the rest of the regression model was not able to find a relation between x and y this may be the result of using fewer data in the testing case.

5) It was a surprise to find the polynomial and decision tree didn't give results as expected in the training data.

6) Overall it was seen that multi linear regression satisfies the requirements. But we cannot conclude the results with only 20% of the data being used therefore upon the usage of the entirety of the data we may get a perfect result

7) Future study needs to be done with support vector regression (SVR) which is a promising method for forecasting, as it offers several advantages

8) Lack of understanding of massive data can be a drawback of big data since all dataset are different and have various data types it can be very nerve-racking to decide on the right tools to use.

7. Conclusion

The project mainly focused on data pre-processing and data visualization and using machine learning models for predicting the AQI

It was found that the data was very inconsistent and had a lot of missing data it was also found that some states did not install analysers to give data therefore there was a lot of missing data in certain states from 2015-to 2017

There were some instances with extremely high reading recorded in certain places which are not possible and the data or the analyser needs to be checked for malfunction.

The highest polluting states were the densely populated cities like Delhi Hyderabad Kolkata and Chennai regards of pm10 and pm2.5

It was also found that BTX were recorded highly in hill stations when compared with major cities this can indicate a high rate of tourism and non-introduction of electric vehicles in these areas and wind speed and wind direction can also have a major impact on the reported data.

Carbon emission has been consistent in all the states throughout 5 years and has not breached the national standards

All this conclude the particulate matter needs to be addressed all over the country and necessary measures need to be taken for the reduction of particulate matter

Upon applying machine learning regression we were partially successful in predicting the AQI and the RFR and MLR gave us good results although these results cannot be considered perfect.

It is found that the regression is highly dependent on the equality of data and accuracy of the data and also the precision of the sensor readings Although it is not possible to have perfect data since there can be several aspects affecting the reading of data like power cut and sensor malfunction. Considering all the aspects best results were given with the given data.

References

Spark Documentation and MLlib documentation.
https://spark.apache.org/docs/latest/api/python/

Spark operation and tutorials
https://www.youtube.com/watch?v=_C8kWso4ne4&t=1413s

Spark installation and Jupiter installation on mac os
https://www.youtube.com/watch?v=MLXOy-OhWRY&t=199s

Air pollution India dataset retrieved from Kaggle January 27
https://www.kaggle.com/rohanrao/air-quality-data-in-india

Air pollution standards for India table reference (2017)
https://www.transportpolicy.net/standard/india-air-quality-standards/

Ambient air pollution in India
http://www.arthapedia.in/index.php?title=Ambient_Air_Quality_Standards_in_India

National air quality standards
https://cpcb.nic.in/National-Air-Quality-Index/

European air quality standers
https://ec.europa.eu/environment/air/quality/standards.htm

A Machine Learning Approach to Predict Air Quality in California Article ID 8049504 | https://doi.org/10.1155/2020/8049504

Machine learning algorithms and references
https://www.youtube.com/watch?v=K490SP-_H0U&t=333s

Marc Peter Deiseroth, A.Aldo Faisal, Cheng Soon Ong Mathematics for machine learning https://mml-book.github.io/book/mml-book.pdf

Christopher M. Bishop Pattern Recognition and machine learning.

Vneogi199(Github) Air Quality Index prediction using python
https://github.com/vneogi199/Air-Quality-Index-Prediction-using-Python

rishika02809(Github) Air Quality Index prediction using ml and lstm
https://github.com/rishika028/Air-Quality-Index-prediction-using-ML-and-LSTM

viraj1131(Github) Indian Air Quality Data Visualization _using _Tableau
https://github.com/viraj1131/India-Air-Quality-Data-Visualization-Using-Tableau

8. Appendix

9.1 Appendix - A

Codes For machine learning

Code part 1

```
1. import numpy as np
2. import pandas as pd
3. from pyspark import SparkContext
4. sc = SparkContext.getOrCreate()
5.
6. df = pd.read_csv("city_day.csv")
7. df2= df.sample(frac =.10)
8. df2=df2.drop(['City','Date','AQI_Bucket'], axis = 1)
9. df2=df2.dropna().astype(np.int64)
10.
11. x1 = df2.iloc[:,:12].values
12. z1 = pd.DataFrame(x1)
13.
14. y1 = df2.iloc[:,12:13].values
15. z2 = pd.DataFrame(y1)
16.
17. from sklearn.preprocessing import OneHotEncoder
18.
19. ohe = OneHotEncoder()
20. x_new1 = pd.DataFrame(ohe.fit_transform(x1[:,[0]]).toarray())
21. x_new2 = pd.DataFrame(ohe.fit_transform(x1[:,[1]]).toarray())
22. x_new3 = pd.DataFrame(ohe.fit_transform(x1[:,[2]]).toarray())
23.
24. feature_set =
    pd.concat([x_new1,x_new2,x_new3,pd.DataFrame(x1[:,5:12])],axis=1,sort=False)
25.
26. df2.dropna(inplace=True)
```

Code part 2

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.SVM import SVR
from math import sqrt

x_train,x_test,y_train,y_test =
train_test_split(feature_set,y1,test_size=0.25,random_state=0)
```

code part 3

```
#------test data prediction--------#


# multiple linear regression model
mreg = LinearRegression()
mreg.fit(x_train,y_train)

mlr_y_predict = mreg.predict(x_test)

y_pred=ml.predict(X_test)
print(y_pred)

#plot
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred_rf)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual VS Predicted')

#compare
pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred_rf
,'Difference': y_test-y_pred })
pred_y_df[0:10]

```

code part 4

```

# polynomial regression model
# degree = 2
poly_reg = PolynomialFeatures(degree = 2)
preg = LinearRegression()
pf = poly_reg.fit_transform(x_train)
```

```
preg.fit(pf,y_train)

pr_y_predict = preg.predict(poly_reg.fit_transform(x_test))
```

code part 5

```
#-----------------------------------------------
# decision tree regression model

dec_tree = DecisionTreeRegressor(random_state = 0)
dec_tree.fit(x_train,y_train)

dt_y_predict = dec_tree.predict(x_test)

y_pred_dc = dec_tree.predict(X_test)
print(y_pred_dc)

#plot
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred_dc)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual VS Predicted')

#compare
pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred_rf
,'Difference': y_test-y_pred_dc })
pred_y_df[0:10]
```

code part 6

```
# random forest regression model
# random forest with 500 trees

rt_reg = RandomForestRegressor(n_estimators = 500, random_state = 0)
rt_reg.fit(x_train,y_train)

rt_y_predict = rt_reg.predict(x_test)

y_pred_rf = rt_reg.predict(X_test)
print(y_pred_rf)

#plot
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
plt.scatter(y_test,y_pred_rf)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual VS Predicted')
#compare
```

```
pred_y_df=pd.DataFrame({'Actual Value':y_test,'Predicted value':y_pred_rf
,'Difference': y_test-y_pred_rf })
pred_y_df[0:10]
```

code part 7

```
        from math import sqrt
from sklearn import metrics

def rmsle(real, predicted):
    sum=0.0
    for x in range(len(predicted)):
        if predicted[x]<0 or real[x]<0:
            continue
        p = np.log(predicted[x]+1)
        r = np.log(real[x]+1)
        sum = sum + (p - r)**2
    return ((sum/len(predicted))**0.5)[0]
```

code part 8

```
        #----- multiple linear regresion -------
rmse_mlr = sqrt(metrics.mean_squared_error(y_test, mlr_y_predict))
mae_mlr = metrics.mean_absolute_error(y_test, mlr_y_predict)
r2_mlr = metrics.r2_score(y_test,mlr_y_predict)
rmsle_mlr = rmsle(y_test,mlr_y_predict)

#----- polynomial regression ------------
rmse_pr = sqrt(metrics.mean_squared_error(y_test, pr_y_predict))
mae_pr = metrics.mean_absolute_error(y_test, pr_y_predict)
r2_pr = metrics.r2_score(y_test,pr_y_predict)
rmsle_pr = rmsle(y_test,pr_y_predict)

#----- decision tree regression ---------
rmse_dt = sqrt(metrics.mean_squared_error(y_test, dt_y_predict))
mae_dt = metrics.mean_absolute_error(y_test, dt_y_predict)
r2_dt = metrics.r2_score(y_test,dt_y_predict)
rmsle_dt = rmsle(y_test,dt_y_predict)

#----- random forest regression ---------
rmse_rt = sqrt(metrics.mean_squared_error(y_test, rt_y_predict))
mae_rt = metrics.mean_absolute_error(y_test, rt_y_predict)
r2_rt = metrics.r2_score(y_test,rt_y_predict)
rmsle_rt = rmsle(y_test,rt_y_predict)
```

code part 9

```
# ---- MLR ------
mlr_ytp_rmse = sqrt(metrics.mean_squared_error(y_train,
mreg.predict(x_train)))
mlr_ytp_mae = metrics.mean_absolute_error(y_train, mreg.predict(x_train))
mlr_ytp_r2 = metrics.r2_score(y_train, mreg.predict(x_train))
m1 = mreg.predict(x_train)
mlr_ytp_rmsle = rmsle(y_train, m1)
#------ polynomial regression ---------
pr_ytp_rmse = sqrt(metrics.mean_squared_error(y_train,
preg.predict(poly_reg.fit_transform(x_train))))
pr_ytp_mae = metrics.mean_absolute_error(y_train,
preg.predict(poly_reg.fit_transform(x_train)))
pr_ytp_r2 = metrics.r2_score(y_train,
preg.predict(poly_reg.fit_transform(x_train)))
pr_ytp_rmsle = rmsle(y_train, preg.predict(poly_reg.fit_transform(x_train)))

#mxp = preg.predict(poly_reg.fit_transform(x_train))

# ----- decision tree reg ------
dt_ytp_rmse = sqrt(metrics.mean_squared_error(y_train,
dec_tree.predict(x_train)))
dt_ytp_mae = metrics.mean_absolute_error(y_train, dec_tree.predict(x_train))
dt_ytp_r2 = metrics.r2_score(y_train, dec_tree.predict(x_train))
dt_ytp_rmsle = rmsle(y_train, dec_tree.predict(x_train))

# ----- random forest reg -----
rf_ytp_rmse = sqrt(metrics.mean_squared_error(y_train,
rt_reg.predict(x_train)))
rf_ytp_mae = metrics.mean_absolute_error(y_train, rt_reg.predict(x_train))
rf_ytp_r2 = metrics.r2_score(y_train, rt_reg.predict(x_train))
rf_ytp_rmsle = rmsle(y_train, rt_reg.predict(x_train))

```

code part 10

```
        #---------- RESULT ----------


```

```
print("evaluating on training data:")
print("models\tR^2\tRMSE\tMAE\tRMSLE")
print("MLR\t{:.2f}\t{3:.2f}\t{1:.2f}\t{3:.2f}".format(mlr_ytp_r2,mlr_ytp_rmse,mlr_ytp_mae,mlr_ytp_rmsle))
print("PR\t{1:.2f}\t{1:.2f}\t{1:.3f}\t{1:.4f}".format(pr_ytp_r2,pr_ytp_rmse,pr_ytp_mae,pr_ytp_rmsle))
print("DTR\t{1:.4f}\t{1:.4f}\t{1:.4f}\t{1:.4f}".format(dt_ytp_r2,dt_ytp_rmse,dt_ytp_mae,dt_ytp_rmsle))
print("RFR\t{1:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(rf_ytp_r2,rf_ytp_rmse,rf_ytp_mae,rf_ytp_rmsle))


print("evaluating on testing data:")
print("models\tR^2\tRMSE\tMAE\tRMSLE")
print("MLR\t{2:.2f}\t{2:.2f}\t{2:.2f}\t{3:.2f}".format(r2_mlr,rmse_mlr,mae_mlr,rmsle_mlr))
print("PR\t{0:.2f}\t{1:.2f}\t{2:.3f}\t{3:.4f}".format(r2_pr,rmse_pr,mae_pr,rmsle_pr))
print("DTR\t{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(r2_dt,rmse_dt,mae_dt,rmsle_dt))
print("RFR\t{1:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}".format(r2_rt,rmse_rt,mae_rt,rmsle_rt))
```

MAE – Mean absolute error is used to find the absolute error the negative values from the error are not considered and only the positive value is considered.

RMSE – Root mean square error in this we will find the residual error and take an average this average value is then squared and the root of the number will be taken as the output.

RMSLE – Root mean Square log error follows the same steps as taken as the RMSE but the log of the final value is taken to give the output.

R^2 - R-squared is a statistical measure of how near the data are to the regression line that has been fitted. The coefficient of determination is another name for this figure.

By importing sqrt and metrics we will solve the errors which come on later

9.2 Appendix -B

Visualisation outputs

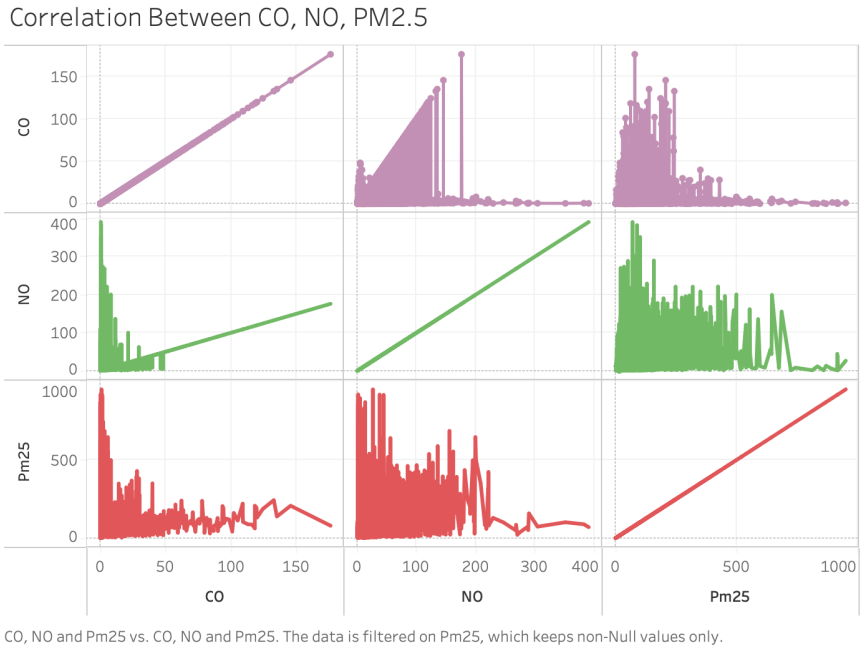


Figure 45 correlation plot

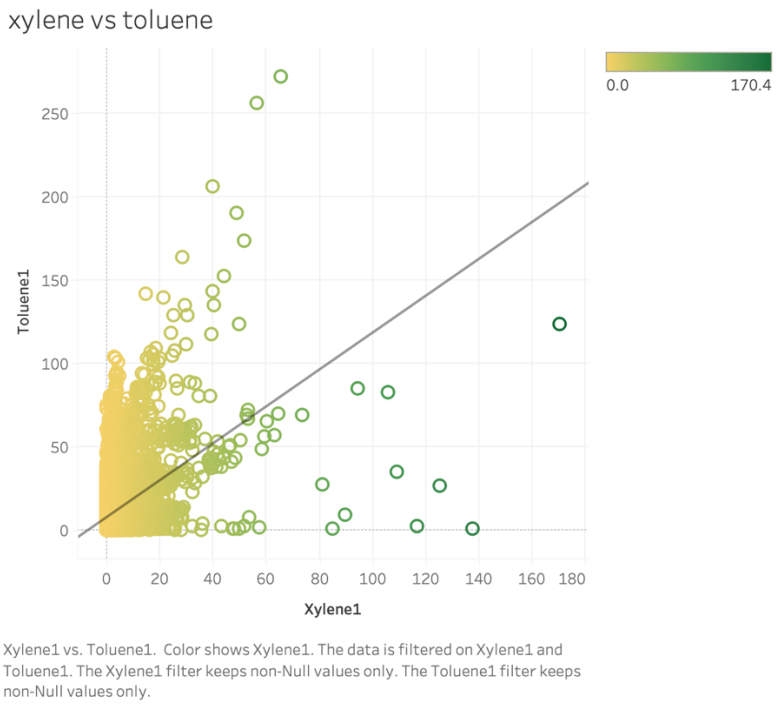


Figure 46 scatter plot

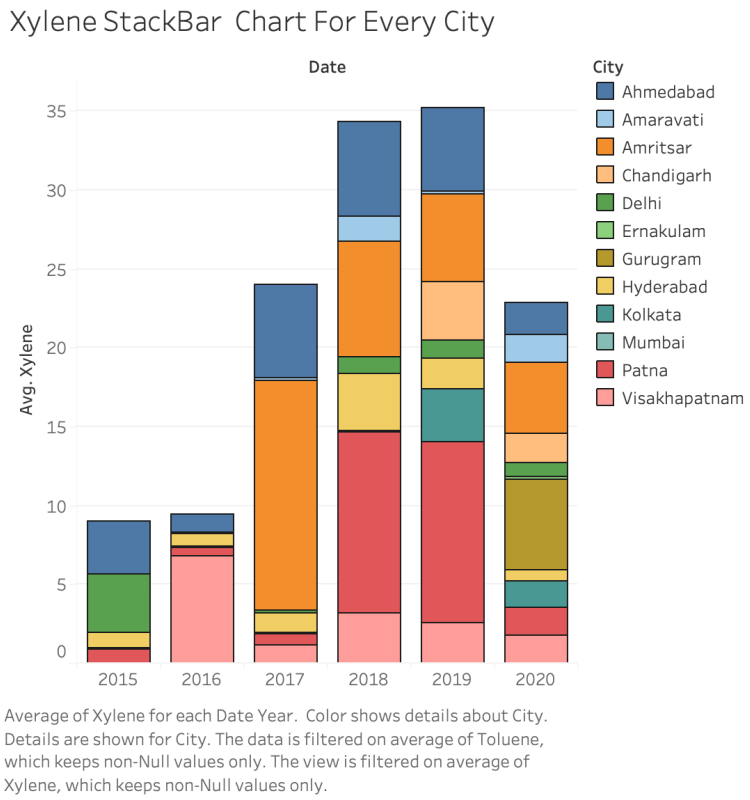


Figure 47 bar chart for BTX

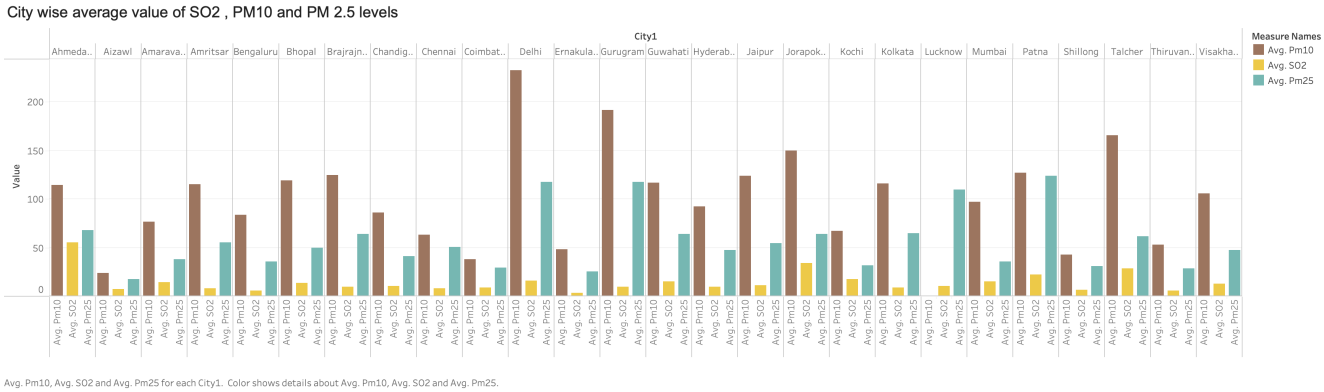


Figure 48 city-wise average levels

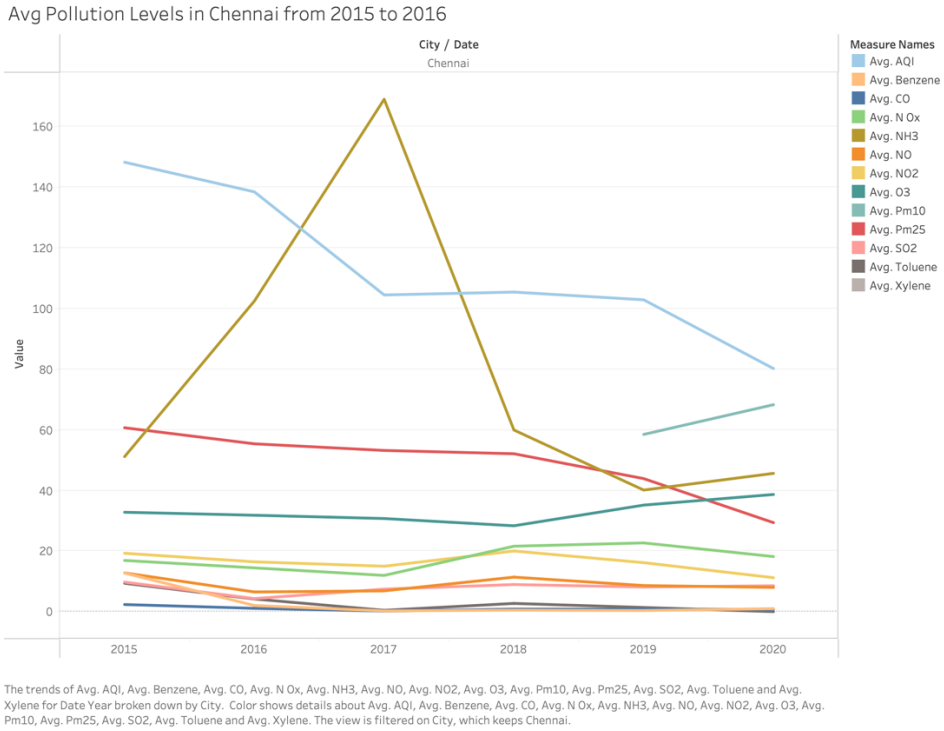


Figure 49Average levels of pollution from 2015 to 2020

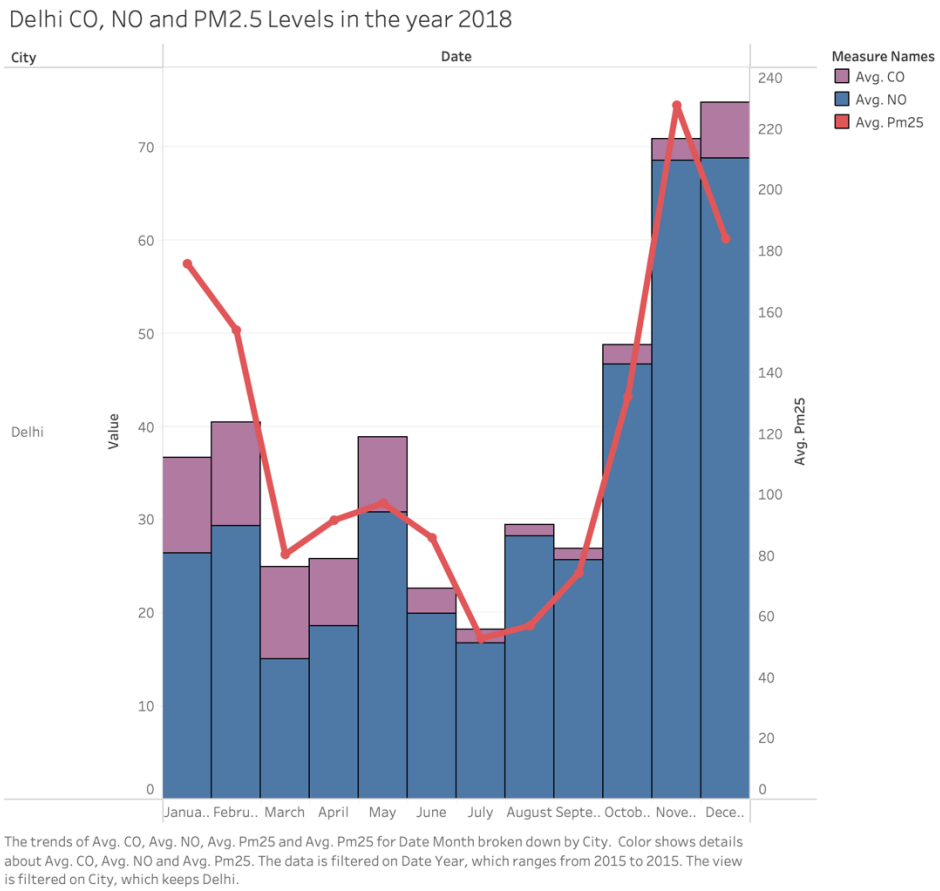
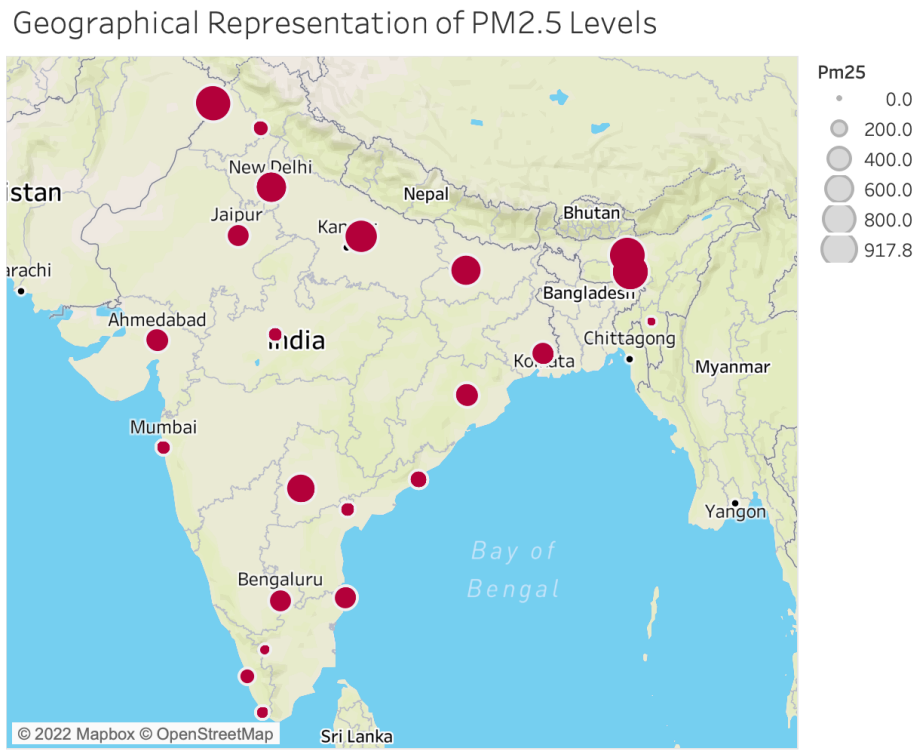


Figure 50 Average pollution in Delhi in the year 2018



Map based on Longitude (generated) and Latitude (generated). Size shows Pm25. Details are shown for City1. The view is filtered on Latitude (generated) and Longitude (generated). The Latitude (generated) filter keeps non-Null values only. The Longitude (generated) filter keeps non-Null values only.

Figure 50 geographical representation of pm2.5

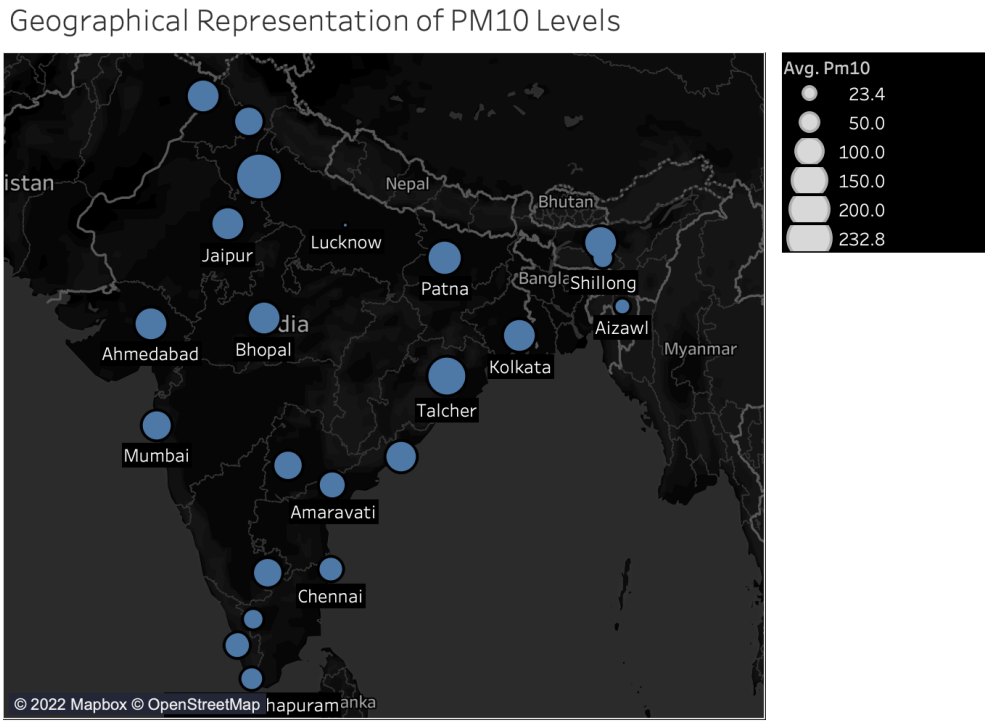


Figure 51 geographical representation of pm10

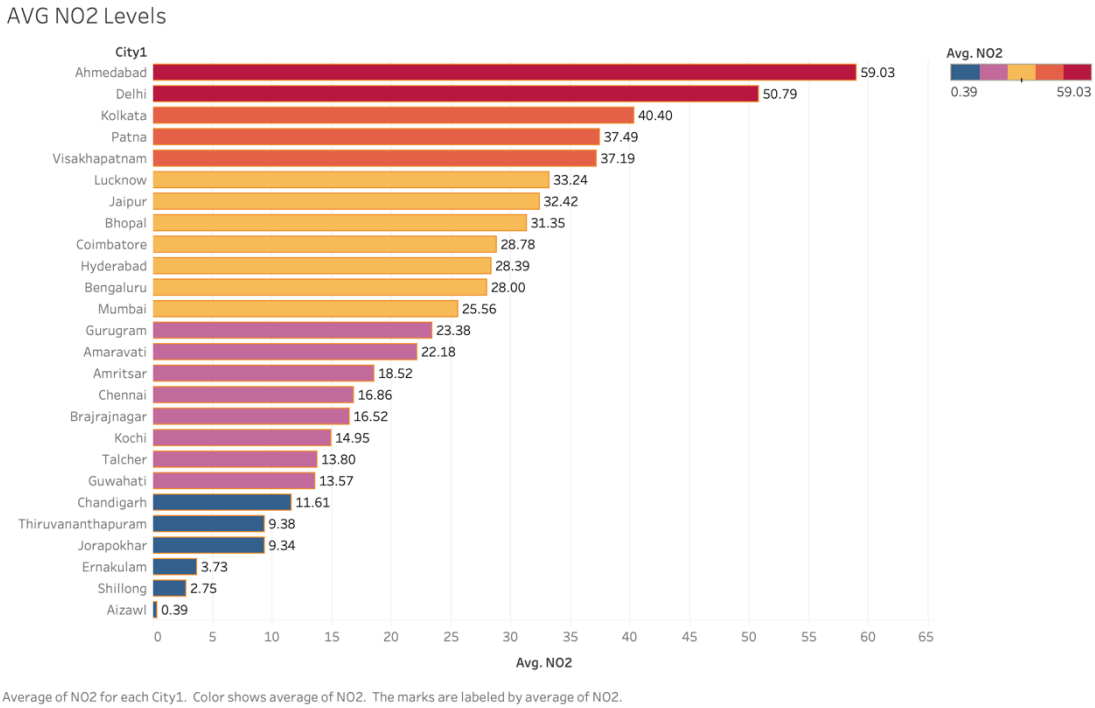
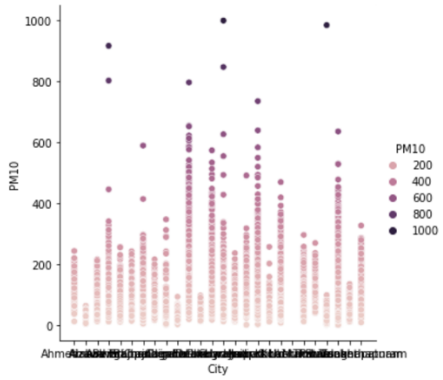


Figure 52 average no levels each state

```
Out[15]: <seaborn.axisgrid.FacetGrid at 0x11dd14550>
```




```
Out[43]: <seaborn.axisgrid.FacetGrid at 0x291601c10>
```

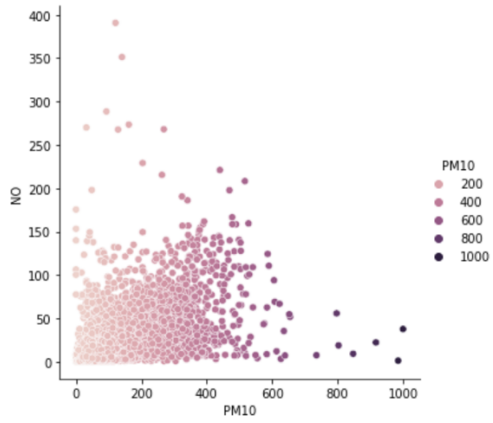
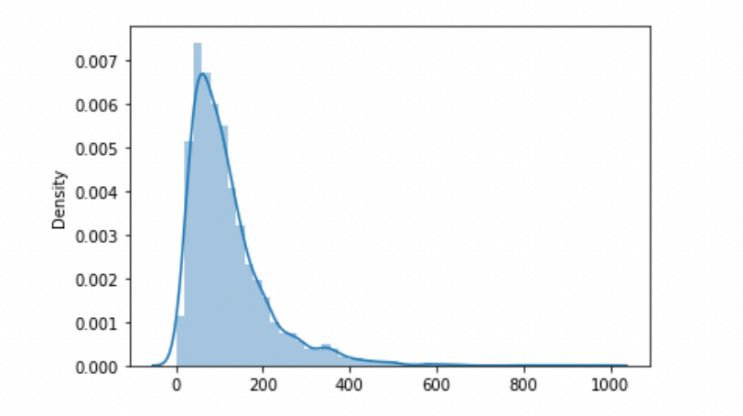


Figure 53 seaborn representation




```
Out[40]: <seaborn.axisgrid.FacetGrid at 0x157c29970>
```

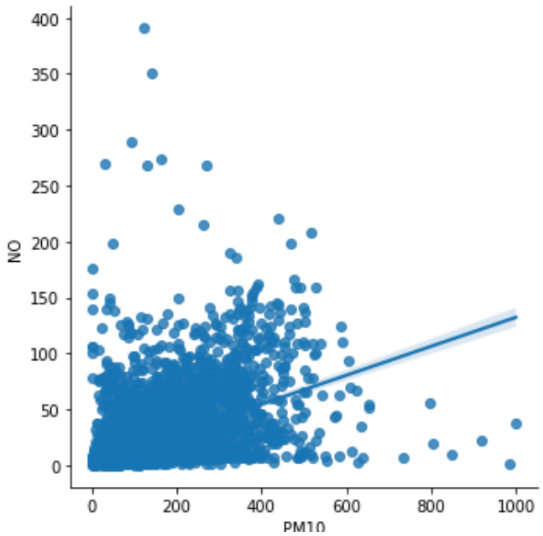


Figure 54 seaborn representation 2

```
Out[64]: <seaborn.axisgrid.FacetGrid at 0x290f074c0>
```

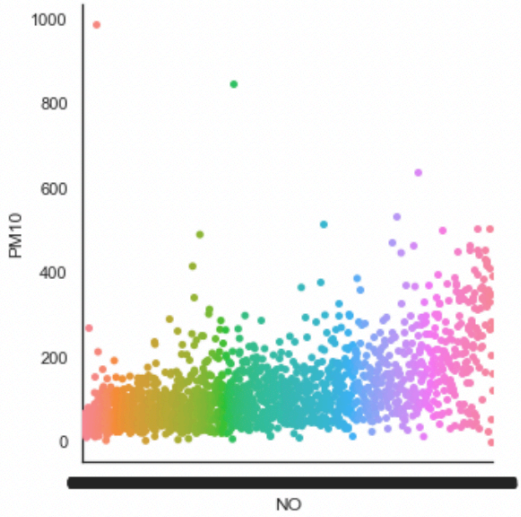
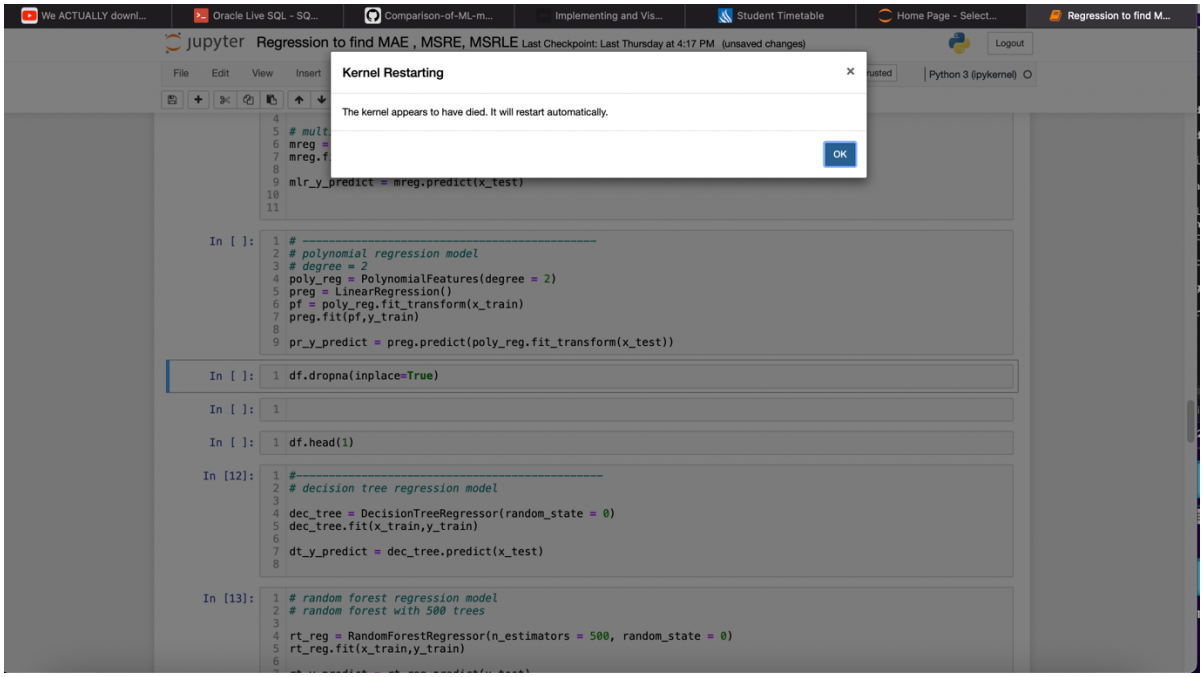


Figure 55 scatterplot using seaborn


We ACTUALLY downl...
Oracle Live SQL - SQ...
Comparison-of-ML-m...
Implementing and Vis...
Student Timetable
Home Page - Select...
Regression to find M...
jupyter Regression to find MAE , MSRE, MSRLE Last Checkpoint: Last Thursday at 4:17 PM (unsaved changes)
Logout
File Edit View Insert
Python 3 (ipykernel)
Kernel Restarting
The kernel appears to have died. It will restart automatically.
OK
mlr_y_predict = mreg.predict(x_test)
In [ ]:
# polynomial regression model
# degree = 2
poly_reg = PolynomialFeatures(degree = 2)
preg = LinearRegression()
pf = poly_reg.fit_transform(x_train)
preg.fit(pf,y_train)
pr_y_predict = preg.predict(poly_reg.fit_transform(x_test))
In [ ]:
df.dropna(inplace=True)
In [ ]:
In [ ]:
df.head(1)
In [12]:
# decision tree regression model
dec_tree = DecisionTreeRegressor(random_state = 0)
dec_tree.fit(x_train,y_train)
dt_y_predict = dec_tree.predict(x_test)
In [13]:
# random forest regression model
# random forest with 500 trees
rt_reg = RandomForestRegressor(n_estimators = 500, random_state = 0)
rt_reg.fit(x_train,y_train)