A

SUMMER TRAINING PROJECT REPORT

ON

"DATA ANALYSIS OF AIR POLLUTION BEFORE AND AFTER COVID-19 LOCKDOWN AND PREDICT AQI OF 2020"

Submitted in partial fulfillment of

Requirement for the award of

Degree of Bachelor of Technology in

COMPUTER SCIENCE

AT

SESSION (2019-2020)

SUBMITTED TO: -

MR. VIKAS KUCHHAL

(Asst. professor of CSE)

SUBMITTED BY:-

MEENU RAWAT

CSE/17/157

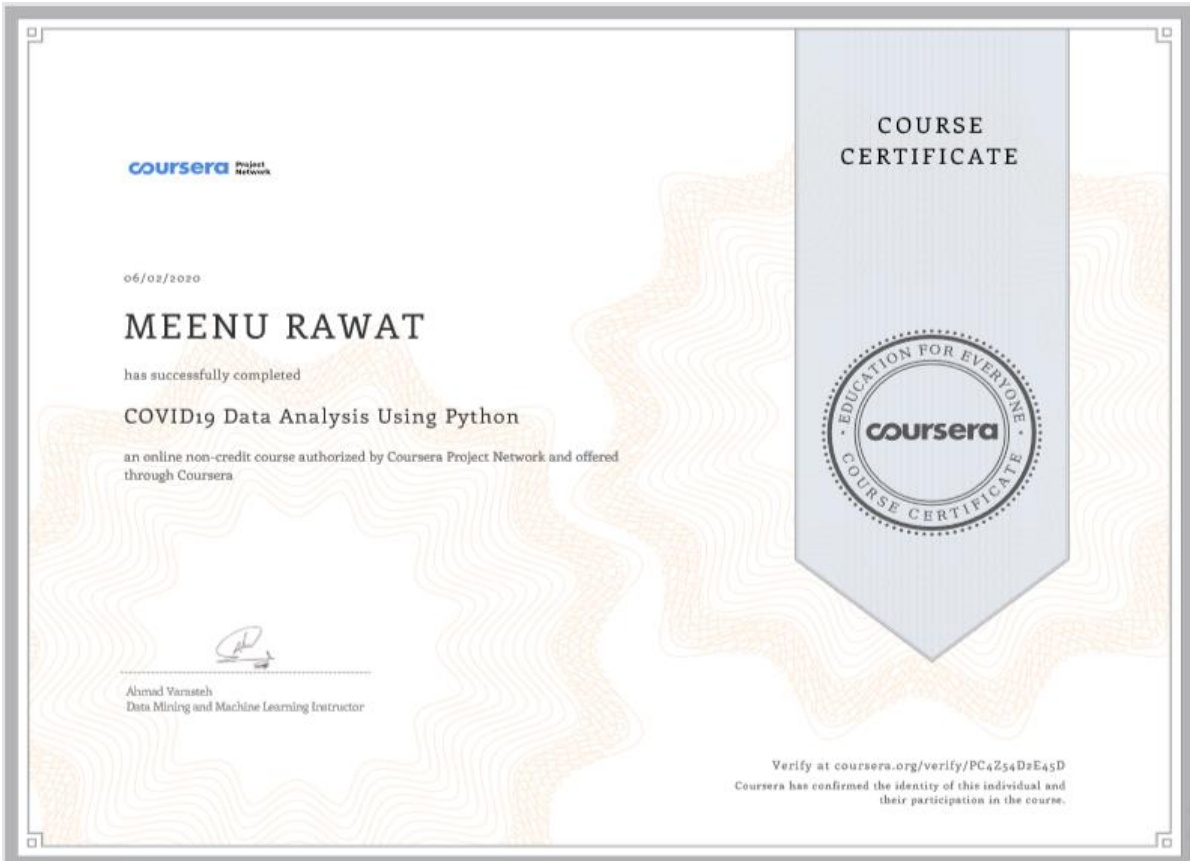
4th year(7th semester)

ACKNOWLEDGEMENT

It is my pleasure to be indebted to various people, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior, and acts during the course of study. I am thankful to **Mr. Ahmad Varasteh** for his support, cooperation, and motivation provided to me during the training for constant inspiration, presence and blessings. I also extend my sincere appreciation to **Mr. Vikas Kuchhal** who provided his valuable suggestions and precious time in accomplishing my project report.

Lastly, I would like to thank the almighty and my parents for their moral support and my friends with whom I shared my day-to-day experience and received lots of suggestions that improved my quality of work.

CERTIFICATE



DECLARATION

I hereby declare that the Industrial Training Report entitled ("Data Analysis of covid-19before and after lockdown and predict AQI of 2020") is an authentic record of my own work as requirements of Industrial Training during the period from 15/04/2020 to 2/06/2020 for the award of degree of B.Tech. (Computer Science), B. M. Institute of Engineering & Technology College, Sonipat, under the guidance of **(MR. VIKAS KUCHHAL**).

Meenu Rawat

Date: 11/01/2021 **cse/17/157**

TABLE OF CONTENTS

INTRODUCTION:

Air pollution in India is emerging as a major factor that seems to contribute to many of the health hazards that people are facing such as asthma, lung and chest infections, and allergies. With the increasing number of vehicles used in metropolitan cities the quality of air that we breathe every second is turning out to be very harmful to our health. This has led to the growing need to conduct periodic measurements of air quality data especially the calculation of air pollutant concentrations along with other data captured at various places and time zones and analysis of it can help to understand the pointers such as the major contributors of air pollution in a particular location, what kind of air pollution carries more hazardous air pollutant, during what time or season the air pollutants are more active, etc. These important resultant data will, in turn, help people to know the areas that are more prone to particular air pollutants and the source of air pollution that is causing this. This data will also help environmentalist to work towards their goal of maintaining ecological balance. There were 7 million deaths in 2012 (WHO, 2015) because of air contamination. Nine out of 10 individuals on the planet inhale contaminated air.

Recently, the air contamination status in Delhi has experienced numerous adjustments as far as the levels of toxins and the control estimates taken to diminish them. The database of the World Health Organization in September 2011 shows that Delhi has the most extreme PM10 value by very nearly 10-times at 198 μg/m3. Vehicles discharges were observed to be related to indoor and open-air contamination in Delhi. [1] Contaminated air has a negative effect on a human, influencing various distinctive frameworks and organs. It shows intense respiratory contaminations in young and constant bronchitis in grown-ups, lung infection, and asthma. Change in vaporous and particulate air toxin fixations have a checked and close transient relationship with unfavorable results

Air pollutants are classified as suspended particulate matter (dust, fumes, mists, and smokes); gaseous pollutants (gases and vapors); and odors. PM10 is smaller than 10 microns (10μm) and is most hazardous. PM2.5 occurs by the condensation of gaseous pollutants such as SO2, Sulfur trioxide, and Carbon monoxide; Nitrogen compounds such as nitric oxide, NO2, and ammonia; Organic compounds such as hydrocarbons; Volatile organic compounds; polycyclic aromatic hydrocarbons and halogen derivatives such as aldehydes; and odorous substances. [4] Air Pollution effects the immune system of the body, which acts as a host defence system that protects against the disease Particulate Matter (PM) - Particulates, alternatively referred to as particulate matter (PM), atmospheric particulate matter, or fine particles, are tiny particles of solid or liquid suspended in a gas. In contrast, aerosol refers to combined particles and gas. Some particulates occur naturally, originating from volcanoe2s, dust storms, forest and grassland fires, living vegetation, and sea spray. Human activities, such as the burning of fossil fuels in vehicles, power plants and various industrial processes also generate significant amounts of aerosols. Averaged worldwide, anthropogenic aerosols—those made by human activities— currently account for approximately 10 percent of our atmosphere. Increased levels of fine particles in the air are linked to health hazards such as heart disease, altered lung function and lung cancer.

EFFECT AFTER LOCKDOWN:

The year 2020 started off on the wrong foot, thanks to the new coronavirus virtually shutting down the world. India also took some necessary precautions, one of them in the form of a nationwide lockdown that Prime Minister Narendra Modi kicked off on March 24 for 21 days, and which was extended further. March is the start of spring season in India – when most cities, especially in the north, experience good air quality. Just past the cold winter haze, it is a chance to open windows and spend some time in the sun. It is typically welcome relief to citizens who, after many months, don't have to worry about wearing masks, checking sensors that indicate when it's safe to go out or huddle around the one purifier in the bedroom.

This spring, thanks to the lockdown, the skies are blue and the air quality index is in code green in most parts of the country. However, it is still no time to step out and enjoy the fresh air. Specifically, the lockdown is an opportunity to understand the extent of air quality changes in various sectors. Cities have very little traffic or commercial and industrial movement.

PM2.5

PM2.5 is the critical pollutant of Delhi, and the rest of India. During the winter haze, this graph is permanently shaded red or brown, notwithstanding the occasional rain that leads to some sedimentation. Before the lockdown, the high winds of spring had already lowered the PM2.5 levels. After the lockdown began, the lowest daily average was 20 μg/m3, with an average of 35 μg/m3. The main contributor to PM2.5 is combustion. With most local activities at a minimum, what we are seeing is the background baseline concentration.

PM10

Resuspended road dust and construction dust are major PM10 contributors. With limited traffic, road dust is under control, and with a complete ban on construction activities, the overall PM10 level has dropped to its lowest day-wise average, at 35 μg/m3. However, dust storms from the west briefly increased the levels since April 10, For 2017-2019, the annual average PM10 level in Delhi was 200 μg/m3.

Sulphur dioxide

For a long time, sulphur dioxide (SO2) was the only pollutant that consistently complied with the national standards. In Delhi, there is no discernible difference between its levels before and after the lockdown. The principal source of SO2 is diesel and coal combustion. Since Delhi already has access to BS-6 fuel, with lowest sulphur content possible, a drop in traffic alone wouldn't – and expectedly didn't – affect the data. What we see in the graph is the background contribution to SO2 levels from coal used at power plants, some industries and cooking activities.

Nitrogen dioxide

the drop in nitrogen dioxide (NO2) is dramatic enough as well as more discernible compared to the other pollutants. This is because its main source is vehicle exhaust. With nearly 90% of vehicles off the road, the change is evident at ground stations and visible even in satellites' columnar observations over China, Italy and the USA.

Carbon monoxide

Carbon monoxide (CO) has the longest tropospheric lifetime – approximately two months – of the criteria pollutants, and its levels are incapable of changing quickly. The daytime averages during February-March ranged from 600 to 1000 μg/m3 and during the lockdown period, from 500 to 800 μg/m3.

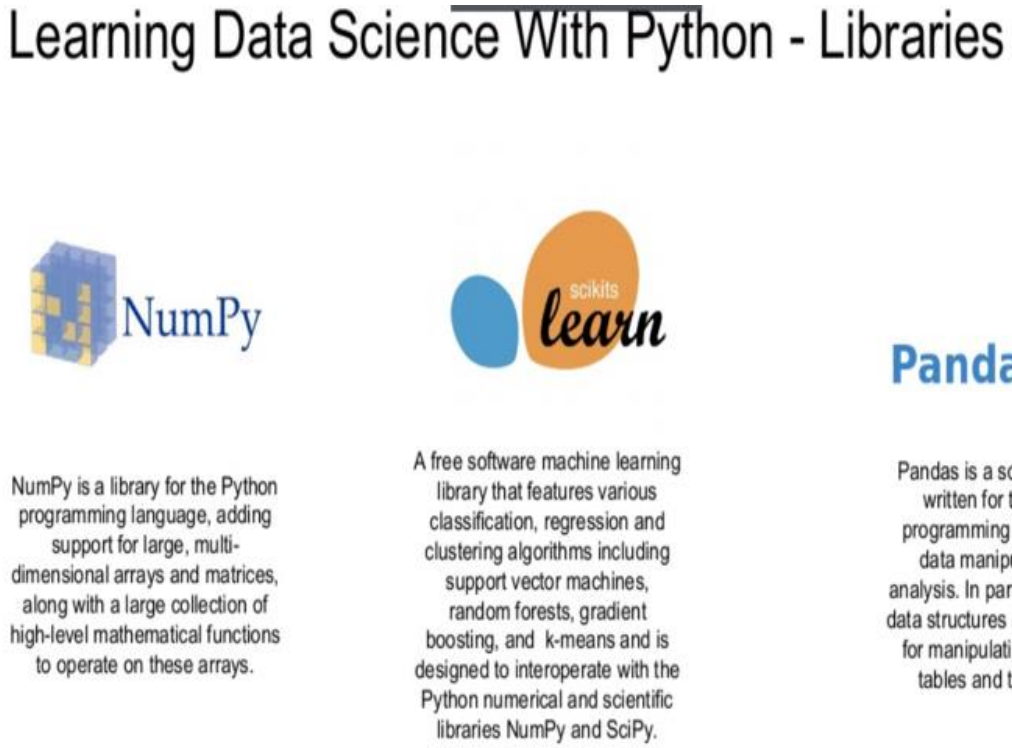
It would be interesting to examine how the pollution levels change in the absence of lockdown conditions. These counterfactual studies will need some additional modelling based on yet-to-come data from various sectors.

AIR QUALITY INDEX (AQI)

An AQI is defined as an overall scheme that transforms weighted values of individual air pollution related parameters (SO2, CO, etc.) into a single number or set of numbers. AQI is useful for the general public to know air quality in a simplified way. Air quality standards are the basic foundation that provides a legal framework for air pollution control. An air quality standard is a description of a level of air quality that is adopted by a regulatory authority as enforceable.

Air Quality Index Levels of Health Concern	Numerical Value	Meaning
Good	0 to 50	Air quality is considered satisfactory, and air pollution poses little or no risk.
Moderate	51 to 100	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
Unhealthy	151 to 200	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
Very Unhealthy	201 to 300	Health warnings of emergency conditions. The entire population is more likely to be affected.
Hazardous	301 to 500	Health alert: everyone may experience more serious health effects.

Technology used:



Python and its libraries:

Python is a general purpose language and is often used for things other than data analysis and data science.

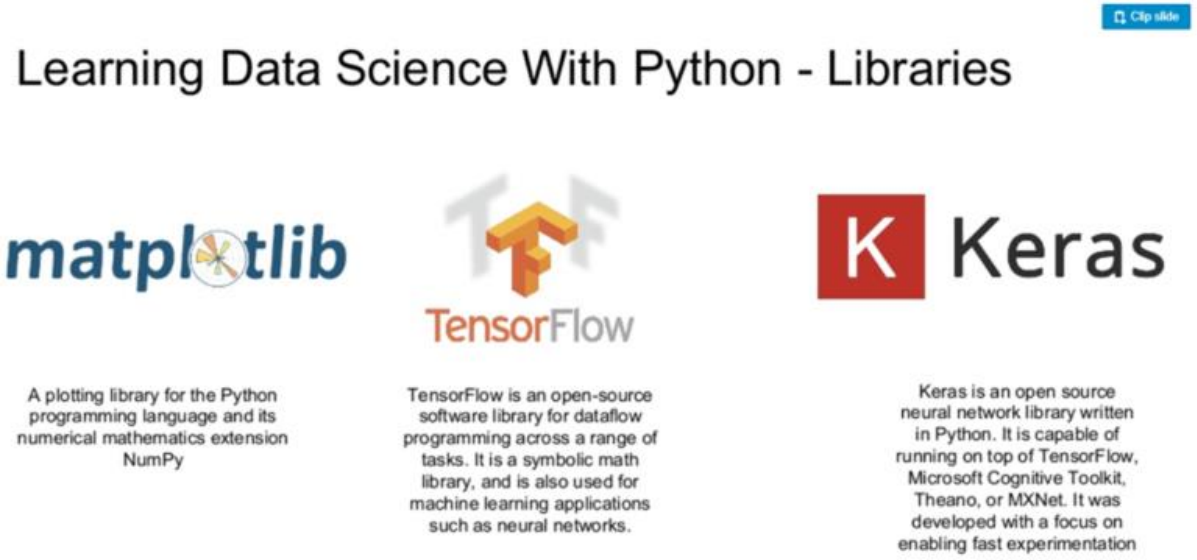
There are libraries that give users the necessary functionality when crunching data.

NUMPY: Fundamental Scientific Computing

NumPy stands for Numerical Python. The most powerful feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran, C and C++.

PANDAS: Data Manipulation and Analysis

Pandas for structured data operations and manipulations. It is extensively used for data munging and preparation. Pandas were added relatively recently to Python and have been instrumental in boosting Python's usage in data scientist community.



MATPLOTLIB: Plotting and Visualization

Matplotlib for plotting vast variety of graphs, starting from histograms to line plots to heat plots.. You can use Pylab feature in ipython notebook (ipython notebook –pylab = inline) to use these plotting features inline. If you ignore the inline option, then pylab converts ipython environment to an environment, very similar to Matlab.

SCIKIT-LEARN: Machine Learning and Data Mining

Scikit Learn for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensional reduction.

SEABORN: For Statistical Data Visualization

Seaborn for statistical data visualization. It is a library for making attractive and informative statistical graphics in Python. It is based on matplotlib. Seaborn aims to make visualization a central part of exploring and understanding data.

Folium:

Folium is a powerful data visualization library in Python that was built primarily to help people visualize geospatial data. With Folium, one can create a map of any location in the world if its latitude and longitude values are known.

Installation:

```
$ pip install folium
```

Data Analysis Steps:

- Data Requirement Gathering. Ask yourself why you're doing this analysis, what type of data analysis you want to use, and what data you are planning on analyzing.
- Data Collection. Guided by the requirements you've identified, it's time to collect the data from your sources. Sources include case studies, surveys, interviews, questionnaires, direct observation, and focus groups. Make sure to organize the collected data for analysis.
- Data Cleaning. Not all of the data you collect will be useful, so it's time to clean it up. This process is where you remove white spaces, duplicate records, and basic errors. Data cleaning is mandatory before sending the information on for analysis.
- Data Analysis. Here is where you use data analysis software and other tools to help you interpret and understand the data and arrive at conclusions. Data analysis tools include Excel, Python, R, Looker, Rapid Miner, Chartio, Metabase, Redash, and Microsoft Power BI.
- Data Interpretation. Now that you have your results, you need to interpret them and come up with the best courses of action, based on your findings.
- Data Visualization. Data visualization is a fancy way of saying, "graphically show your information in a way that people can read and understand it." You can use charts, graphs, maps, bullet points, or a host of other methods. Visualization helps you derive valuable insights by helping you compare datasets and observe relationships.

Although there are many data analysis methods available, they all fall into one of two primary types: qualitative analysis and quantitative analysis.

- Qualitative Data Analysis. The qualitative data analysis method derives data via words, symbols, pictures, and observations. This method doesn't use statistics. The most common qualitative methods include:
  - o Content Analysis, for analyzing behavioral and verbal data.
  - o Grounded Theory, for developing causal explanations of a given event by studying and extrapolating from one or more past cases.

- Quantitative Data Analysis. Statistical data analysis methods collect raw data and process it into numerical data. Quantitative analysis methods include:
  - o Hypothesis Testing, for assessing the truth of a given hypothesis or theory for a data set or demographic.
  - o Mean, or average determines a subject's overall trend by dividing the sum of a list of numbers by the number of items on the list.

AIR QUALITY INDEX PREDICTION MODEL

System analysis:

Fine material (PM2.5) could be a important one as a result of it's a giant concern to people's health once its level within the air is comparatively high. PM2.5 refers to little particles within the air that scale back visibility and cause the air to look hazy once levels are elevated. But in the proposed system we calculate the air quality index of all the pollutants using the AQI formulae to know the air quality level in a particular city using gradient descent and Box-Plot analysis.

Experimental analysis

Data sources

To predict the air quality index of a particular region, we need the pollutant concentration of all the gases which will be available in the cpcb.nic.in website, which holds all the data that pollutes the cities every year. The AQI formulae will be applied in order to calculate the AQI by using the linear regression algorithm for a particular year. Several datasets will be imported inside the directory and null values will be set to the infinite data. The predicted and actual values will be represented using the Box-Plot analysis in order to remove the outliers.

Pre-processing the data

In this dataset the outliers are mainly of faulty sensor or transmission errors, these errors have huge variation than the normal valid results. We know the standard range of pollutants occurs on a particular areaso to remove the outliers from the data we use boundary value analysis. By using BVA we found the upper quartile range and lower quartile range of a given data

AQI simulation and calculation

We acquired the dataset with various columns of sensor data from various places in India. Wwe have International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 11, 2019 (Special Issue) © Research India Publications. http://www.ripublication.com Page 182 of 186 the average readings of ambient air quality with respect to air quality parameters, like Sulphur dioxide (So2), Nitrogen dioxide (No2), Respirable Suspended Particulate Matter (RSPM) and Suspended Particulate Matter (SPM). Data acquired from the source has more noisy data since few of the data from the stations have been shifted or closed the period were marked as NAN or not available.so we have to pre-process the data in order to remove the outliers.Each individual pollutant indexes, gives the relationship between the pollutant concentration and their corresponding individual index

The air quality index of a particular data point is the aggregate of maximum indexed pollutant on that particular area. That pollutants maxsub index is taken as the air quality index of that particular location

Prediction of air quality index

Using Naïve Forecast approach, we spitted the dataset into two parts of first 75% and rest 25% data into test and train datasets to identify the huge seasonal variations and trend.

We calculated the moving average of our datapoints and plotted the moving average.

Plotted the graph of train and test dataset with their moving average and analyzed the moving average.

Results analysis
Box plot is one of common graphical systems utilized in EDA.A crate plot or boxplot is a helpful method for graphically portraying gatherings of numerical information through their quartiles. Box plots may likewise have lines broadening vertically from the containers(bristles) demonstrating inconstancy outside the upper and lower quartiles, henceforth the terms box-and-hair plot and box-and stubble graph. Exceptions might be plotted as individual focuses.
Box Plot gives fundamental data about a dispersion. It graphically delineates a gathering of numerical information
By this data analysis we came to know that there are seasonal variations and trend, in order to reduce these metrics, we resample the data month wise to predict it month wise. By resampling the data, we can reduce the outlier more efficiently than raw data. After removing the outlier's linear regression is applied to the filtered data and to fit the trend line on the data points gradient descent hyper parameters are used to optimize the model.

Linear regression
While doing straight relapse our goal is to fit a line through the dissemination which is closest to the majority of the focuses. Subsequently lessening the separation (mistake term) of information focuses from the fitted line.

Gradient boost algorithm
The principle issue influenced by individuals is air contamination since air contains numerous substances which might be made by manmade or regular procedure. The Air substances present most organic atoms, points of interest and perilous material into the air. Boosting Algorithm is a victor among the most prevalent
learning insights showed over the most recent twenty years.

Conclusion and future enhancements:
Since our model is capable of predicting the current data with 95% accuracy it will successfully predict the upcoming air quality index of any particular data within a given region. With this model we can forecast the AQI and alert the respected region of the country also it a progressive learning model it is capable of tracing back to the particular location needed attention provided the time series data of every possible region needed attention.

RESULTS:

After declaration of three weeks of lockdown starting from 24th of March 2020, pollution of the megacity Delhi has witnessed substantial diminution of the pollutants. Especially, during the study period PM 10, PM2.5, NO2, and CO concentration have shown significant declining trends. Averaged concentrations of PM10and PM2.5 have reduced.Other pollutants that have shown considerable variation between pre and during lockdown are NO2 (−52.68%) and CO (−30.35%). However, for SO2 (−17.97%), and NH3 (−12.33%) the reduction have counted very low in comparison to the others and the trend also not evidencing prominent definite trend . 8 h average daily maximum concentration of O3 (+0.78% overall variation) in the study period shows negligible increase within significant rising trend. In this case, considering that April to August in Indian subcontinent is the usual high O3 period due

to the increase in insolation is quite feasible. Important to note that, the concentration of O3 increases especially in the industrial and transport dominated locations (N10% increase).

This is a clear indication that substantial improvement of the air quality could be expected if strict implementation of air quality control measures like lockdown is put into practice.

FUTURE SCOPE:

- Future scope of this work is to explore and work upon various data analytical techniques to build a forecasting model which is adaptable to dynamic atmospheric variables. Appropriate forecasting model of ambient air pollution is prerequisite in developing stringent pollution control technology and measures thereof.

REFERENCES

Amann, M., Purohit, P., Bhanarkar, A.D., Bertok, I., Borken-Kleefeld, J., Cofala, J., Heyes, C., Kiesewetter, G., Klimont, Z., Liu, J., Majumdar, D., 2017. Managing future air quality in megacities: a case study for Delhi. Atmos. Environ. 161, 99–111.

Anjum, N.A., 2020. Good in the Worst: COVID-19 Restrictions and Ease in Global Air Pollution. Preprints. https://doi.org/10.20944/preprints202004.0069.v1 2020040069.

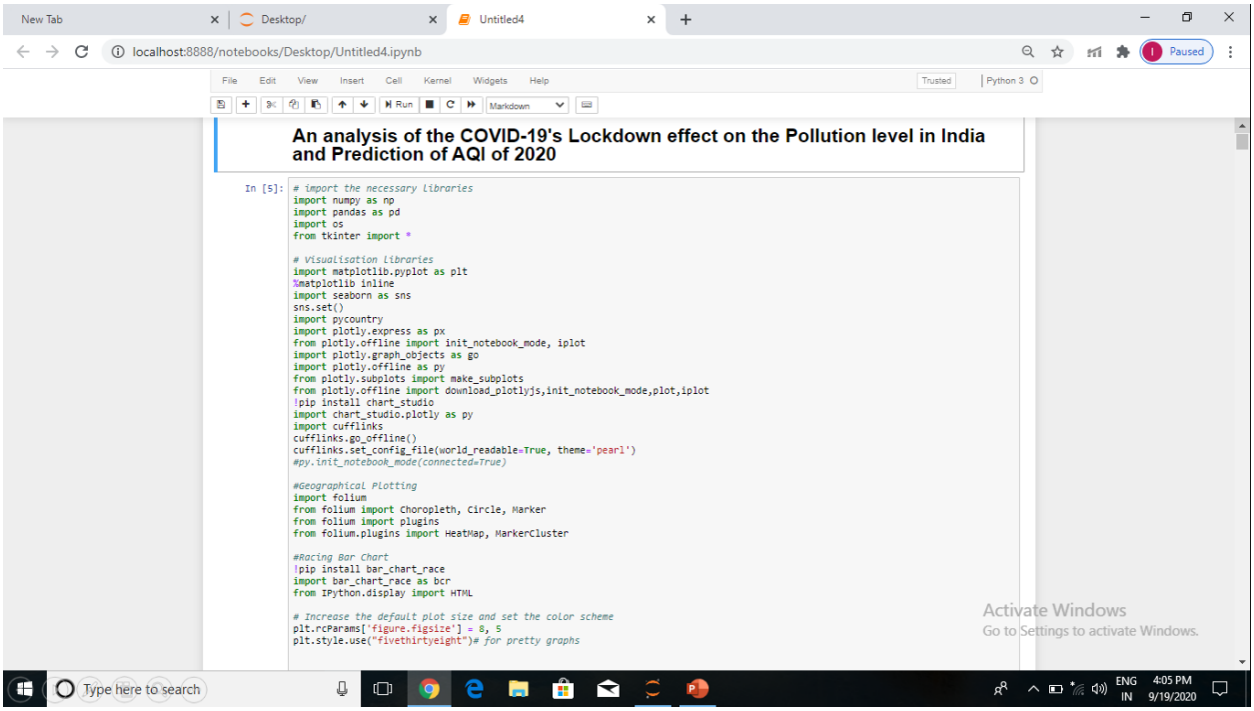
Bashir, M.F., Ma, B., Komal, B., Bashir, M.A., Tan, D., Bashir, M., 2020. Correlation between climate indicators and COVID-19 pandemic in New York, USA. Sci. Total Environ. 728, 138835.

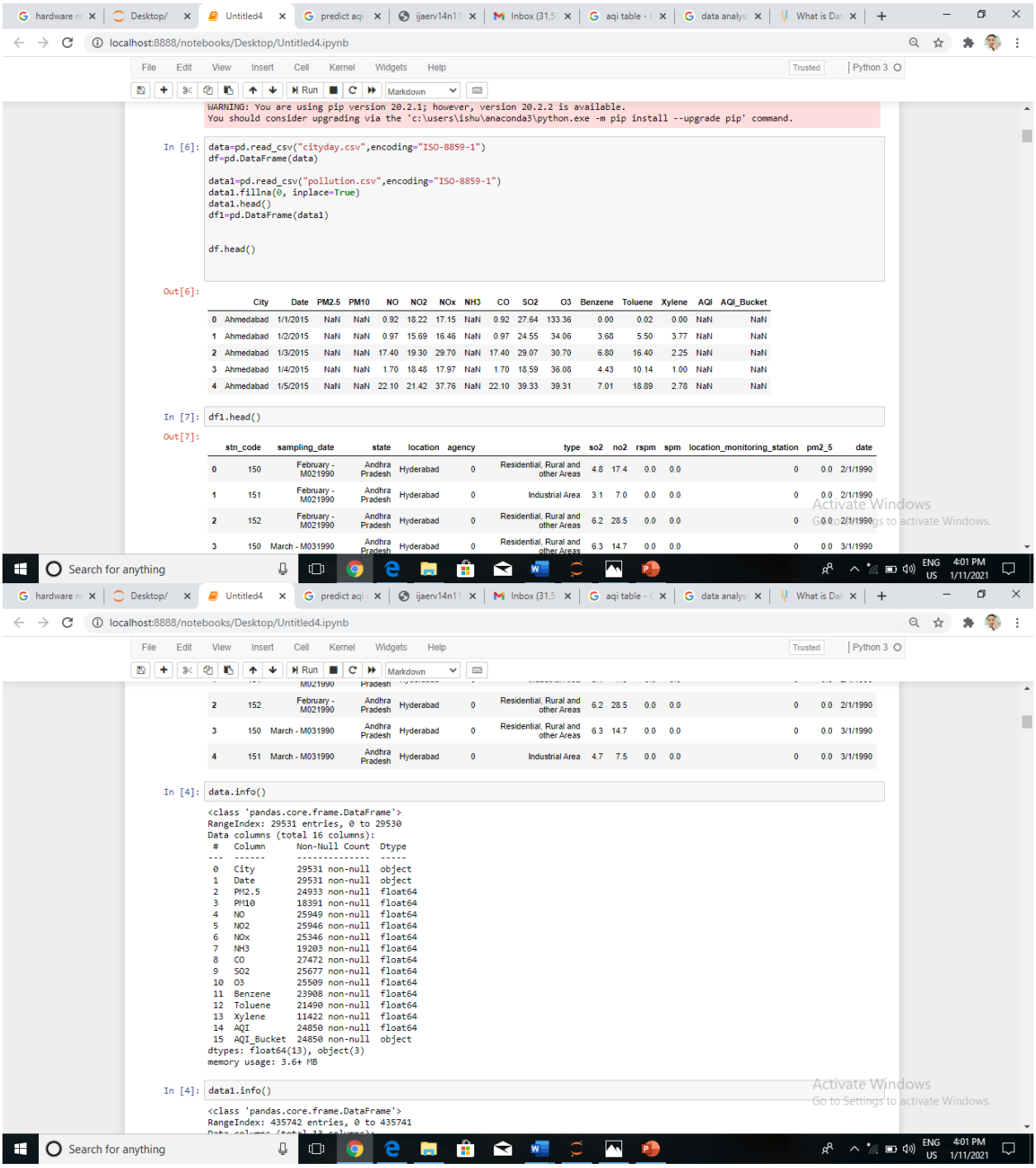
Beig, G., Ghude, S.D., Deshpande, A., 2010. Scientific Evaluation of Air Quality Standards and Defining Air Quality Index for India. Indian Institute of Tropical Meteorology.

Bezuglaya, E.Y., Shchutskaya, A.B., Smirnova, I.V., 1993. Air pollution index and interpretation of measurements of toxic pollutant concentrations. Atmos. Environ. Part A 27 (5), 773–779.

Bishoi, B., Prakash, A., Jain, V.K., 2009. A comparative study of air quality index based on factor analysis and US-EPA methods for an urban environment. Aerosol Air Qual. Res. 9 (1), 1–17.

SCREENSHOTS OF CODE:





```
WARNING: You are using pip version 20.2.1; however, version 20.2.2 is available.
You should consider upgrading via the 'c:\users\ishu\anaconda3\python.exe -m pip install --upgrade pip' command.
```

In [6]:
```
data=pd.read_csv("cityday.csv",encoding="ISO-8859-1")
df=pd.DataFrame(data)

data1=pd.read_csv("pollution.csv",encoding="ISO-8859-1")
data1.fillna(0, inplace=True)
data1.head()
df1=pd.DataFrame(data1)


df.head()
```

Out[6]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket
0	Ahmedabad	1/1/2015	NaN	NaN	0.92	18.22	17.15	NaN	0.92	27.64	133.36	0.00	0.02	0.00	NaN	NaN
1	Ahmedabad	1/2/2015	NaN	NaN	0.97	15.69	16.46	NaN	0.97	24.55	34.06	3.68	5.50	3.77	NaN	NaN
2	Ahmedabad	1/3/2015	NaN	NaN	17.40	19.30	29.70	NaN	17.40	29.07	30.70	6.80	16.40	2.25	NaN	NaN
3	Ahmedabad	1/4/2015	NaN	NaN	1.70	18.48	17.97	NaN	1.70	18.59	36.08	4.43	10.14	1.00	NaN	NaN
4	Ahmedabad	1/5/2015	NaN	NaN	22.10	21.42	37.76	NaN	22.10	39.33	39.31	7.01	18.89	2.78	NaN	NaN

In [7]:
```
df1.head()
```

Out[7]:

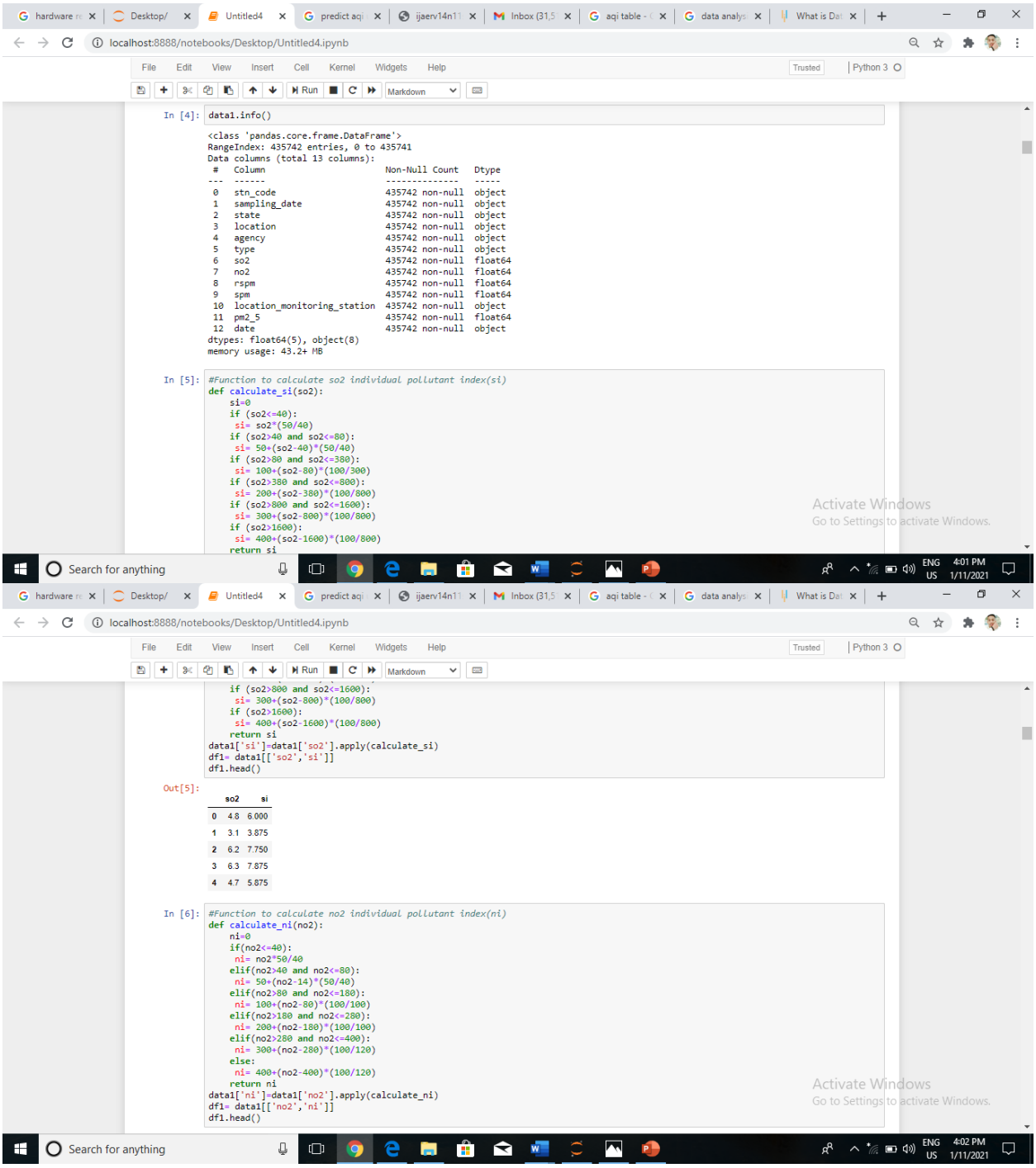
	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date
0	150	February - M021990	Andhra Pradesh	Hyderabad	0	Residential, Rural and other Areas	4.8	17.4	0.0	0.0	0	0.0	2/1/1990
1	151	February - M021990	Andhra Pradesh	Hyderabad	0	Industrial Area	3.1	7.0	0.0	0.0	0	0.0	2/1/1990
2	152	February - M021990	Andhra Pradesh	Hyderabad	0	Residential, Rural and other Areas	6.2	28.5	0.0	0.0	0	0.0	2/1/1990
3	150	March - M031990	Andhra Pradesh	Hyderabad	0	Residential, Rural and other Areas	6.3	14.7	0.0	0.0	0	0.0	3/1/1990
4	151	March - M031990	Andhra Pradesh	Hyderabad	0	Industrial Area	4.7	7.5	0.0	0.0	0	0.0	3/1/1990

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29531 entries, 0 to 29530
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   City        29531 non-null  object
 1   Date        29531 non-null  object
 2   PM2.5       24933 non-null  float64
 3   PM10        18391 non-null  float64
 4   NO          25949 non-null  float64
 5   NO2         25946 non-null  float64
 6   NOx         25346 non-null  float64
 7   NH3         19203 non-null  float64
 8   CO          27472 non-null  float64
 9   SO2         25677 non-null  float64
 10  O3          25509 non-null  float64
 11  Benzene     23908 non-null  float64
 12  Toluene     21490 non-null  float64
 13  Xylene      11422 non-null  float64
 14  AQI         24850 non-null  float64
 15  AQI_Bucket  24850 non-null  object
dtypes: float64(13), object(3)
memory usage: 3.6+ MB
```

In [4]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435742 entries, 0 to 435741
```



In [4]:
```
data1.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                    435742 non-null  object 
 1   sampling_date               435742 non-null  object 
 2   state                       435742 non-null  object 
 3   location                    435742 non-null  object 
 4   agency                      435742 non-null  object 
 5   type                        435742 non-null  object 
 6   so2                         435742 non-null  float64
 7   no2                         435742 non-null  float64
 8   rspm                        435742 non-null  float64
 9   spm                         435742 non-null  float64
 10  location_monitoring_station  435742 non-null  object 
 11  pm2_5                       435742 non-null  float64
 12  date                        435742 non-null  object 
dtypes: float64(5), object(8)
memory usage: 43.2+ MB
```

In [5]:
```
#Function to calculate so2 individual pollutant index(si)
def calculate_si(so2):
    si=0
    if (so2<=40):
     si= so2*(50/40)
    if (so2>40 and so2<=80):
     si= 50+(so2-40)*(50/40)
    if (so2>80 and so2<=380):
     si= 100+(so2-80)*(100/300)
    if (so2>380 and so2<=800):
     si= 200+(so2-380)*(100/800)
    if (so2>800 and so2<=1600):
     si= 300+(so2-800)*(100/800)
    if (so2>1600):
     si= 400+(so2-1600)*(100/800)
    return si
```

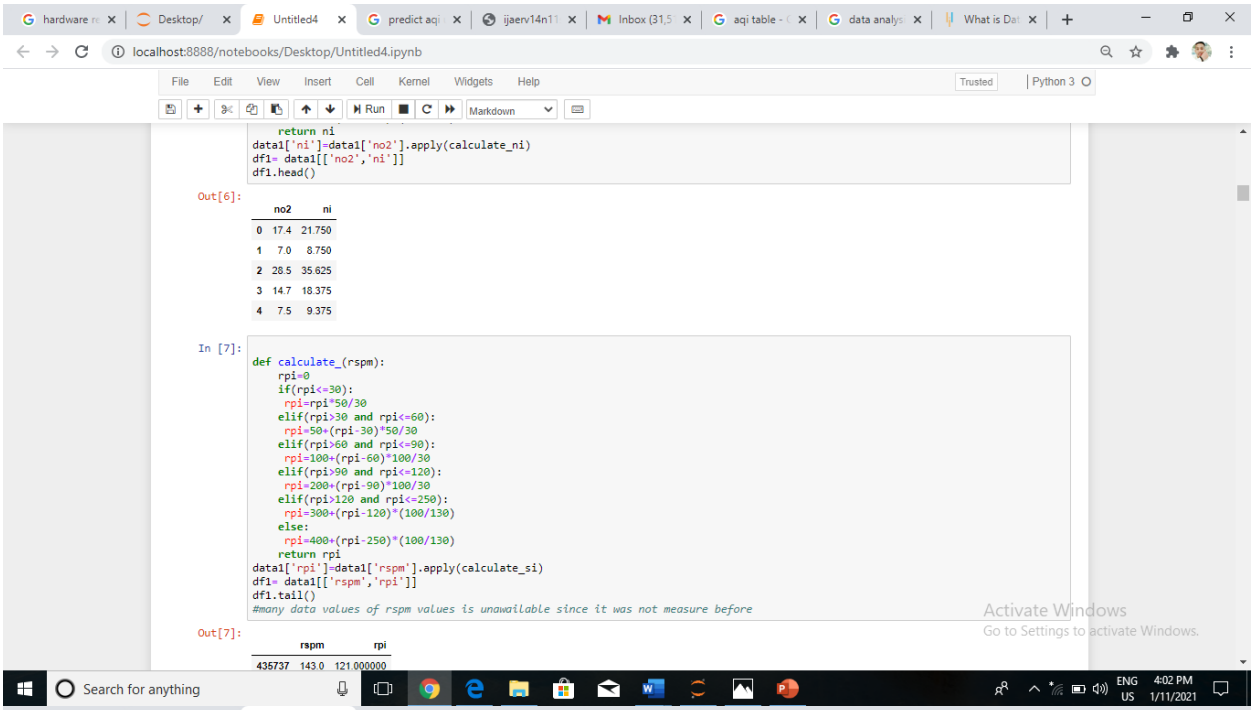

In [5]:
```
    if (so2>800 and so2<=1600):
     si= 300+(so2-800)*(100/800)
    if (so2>1600):
     si= 400+(so2-1600)*(100/800)
    return si
data1['si']=data1['so2'].apply(calculate_si)
df1= data1[['so2','si']]
df1.head()
```

Out[5]:

	so2	si
0	4.8	6.000
1	3.1	3.875
2	6.2	7.750
3	6.3	7.875
4	4.7	5.875

In [6]:
```
#Function to calculate no2 individual pollutant index(ni)
def calculate_ni(no2):
    ni=0
    if(no2<=40):
     ni= no2*50/40
    elif(no2>40 and no2<=80):
     ni= 50+(no2-14)*(50/40)
    elif(no2>80 and no2<=180):
     ni= 100+(no2-80)*(100/100)
    elif(no2>180 and no2<=280):
     ni= 200+(no2-180)*(100/100)
    elif(no2>280 and no2<=400):
     ni= 300+(no2-280)*(100/120)
    else:
     ni= 400+(no2-400)*(100/120)
    return ni
data1['ni']=data1['no2'].apply(calculate_ni)
df1= data1[['no2','ni']]
df1.head()
```

Activate Windows
Go to Settings to activate Windows.



```
    return ni
data1['ni']=data1['no2'].apply(calculate_ni)
df1= data1[['no2','ni']]
df1.head()
```

Out[6]:

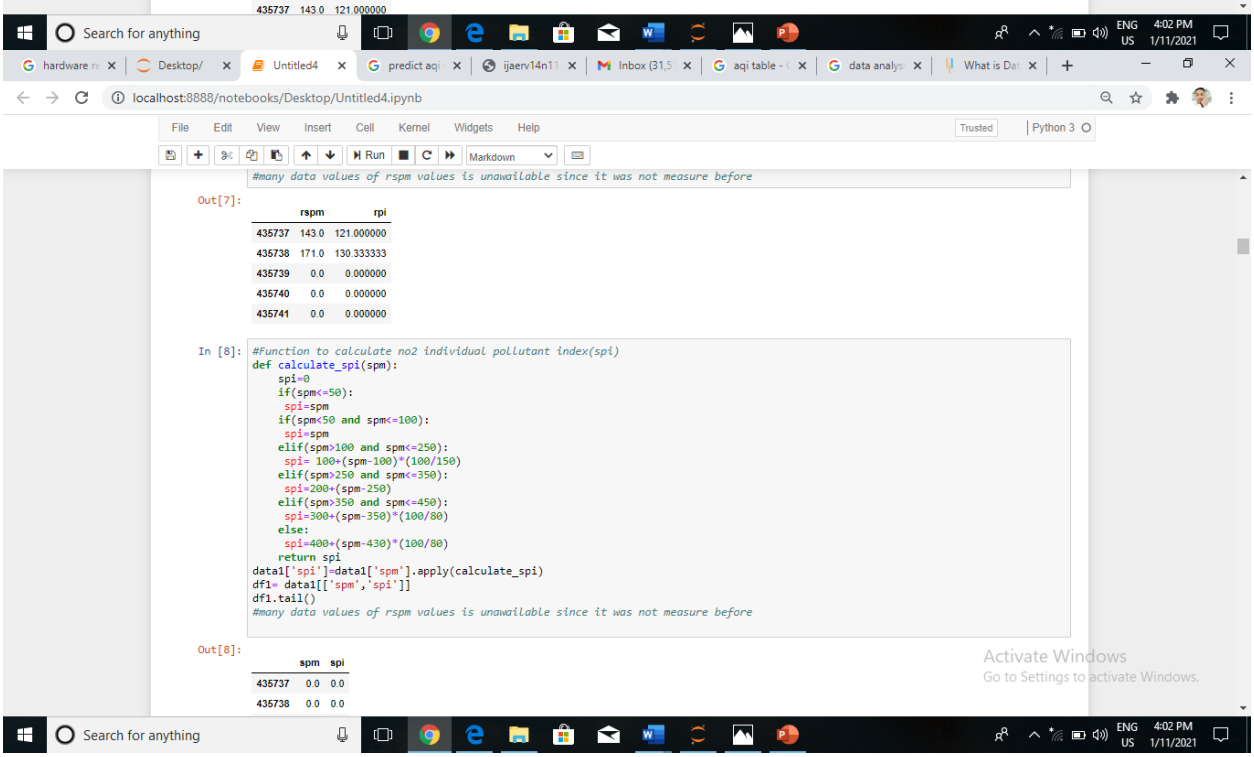
	no2	ni
0	17.4	21.750
1	7.0	8.750
2	28.5	35.625
3	14.7	18.375
4	7.5	9.375

In [7]:

```
def calculate_(rspm):
    rpi=0
    if(rpi<=30):
     rpi=rpi*50/30
    elif(rpi>30 and rpi<=60):
     rpi=50+(rpi-30)*50/30
    elif(rpi>60 and rpi<=90):
     rpi=100+(rpi-60)*100/30
    elif(rpi>90 and rpi<=120):
     rpi=200+(rpi-90)*100/30
    elif(rpi>120 and rpi<=250):
     rpi=300+(rpi-120)*(100/130)
    else:
     rpi=400+(rpi-250)*(100/130)
     return rpi
data1['rpi']=data1['rspm'].apply(calculate_si)
df1= data1[['rspm','rpi']]
df1.tail()
#many data values of rspm values is unawailable since it was not measure before
```

Out[7]:

	rspm	rpi
435737	143.0	121.000000



```
#many data values of rspm values is unawailable since it was not measure before
```

Out[7]:

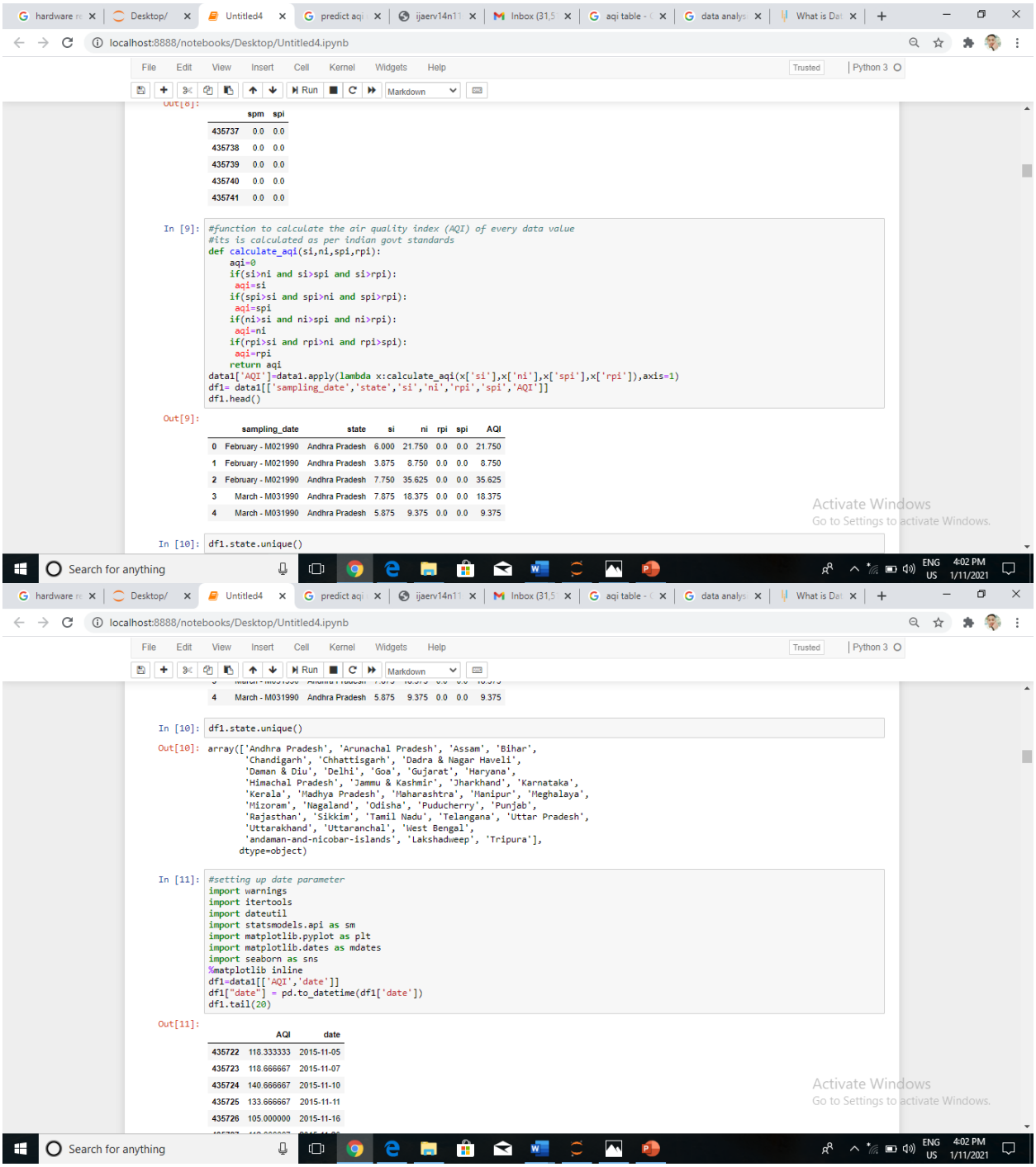
	rspm	rpi
435737	143.0	121.000000
435738	171.0	130.333333
435739	0.0	0.000000
435740	0.0	0.000000
435741	0.0	0.000000

In [8]:

```
#Function to calculate no2 individual pollutant index(spi)
def calculate_spi(spm):
    spi=0
    if(spm<=50):
     spi=spm
    if(spm<50 and spm<=100):
     spi=spm
    elif(spm>100 and spm<=250):
     spi= 100+(spm-100)*(100/150)
    elif(spm>250 and spm<=350):
     spi=200+(spm-250)
    elif(spm>350 and spm<=450):
     spi=300+(spm-350)*(100/80)
    else:
     spi=400+(spm-430)*(100/80)
     return spi
data1['spi']=data1['spm'].apply(calculate_spi)
df1= data1[['spm','spi']]
df1.tail()
#many data values of rspm values is unawailable since it was not measure before
```

Out[8]:

	spm	spi
435737	0.0	0.0
435738	0.0	0.0



Out[8]:

	spm	spi
435737	0.0	0.0
435738	0.0	0.0
435739	0.0	0.0
435740	0.0	0.0
435741	0.0	0.0

In [9]:
```
#function to calculate the air quality index (AQI) of every data value
#its is calculated as per indian govt standards
def calculate_aqi(si,ni,spi,rpi):
    aqi=0
    if(si>ni and si>spi and si>rpi):
     aqi=si
    if(spi>si and spi>ni and spi>rpi):
     aqi=spi
    if(ni>si and ni>spi and ni>rpi):
     aqi=ni
    if(rpi>si and rpi>ni and rpi>spi):
     aqi=rpi
    return aqi
data1['AQI']=data1.apply(lambda x:calculate_aqi(x['si'],x['ni'],x['spi'],x['rpi']),axis=1)
df1= data1[['sampling_date','state','si','ni','rpi','spi','AQI']]
df1.head()
```

Out[9]:

	sampling_date	state	si	ni	rpi	spi	AQI
0	February - M021990	Andhra Pradesh	6.000	21.750	0.0	0.0	21.750
1	February - M021990	Andhra Pradesh	3.875	8.750	0.0	0.0	8.750
2	February - M021990	Andhra Pradesh	7.750	35.625	0.0	0.0	35.625
3	March - M031990	Andhra Pradesh	7.875	18.375	0.0	0.0	18.375
4	March - M031990	Andhra Pradesh	5.875	9.375	0.0	0.0	9.375

In [10]:
```
df1.state.unique()
```

Search for anything
ENG US 4:02 PM 1/11/2021
localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Markdown

4	March - M031990	Andhra Pradesh	5.875	9.375	0.0	0.0	9.375

In [10]:
```
df1.state.unique()
```

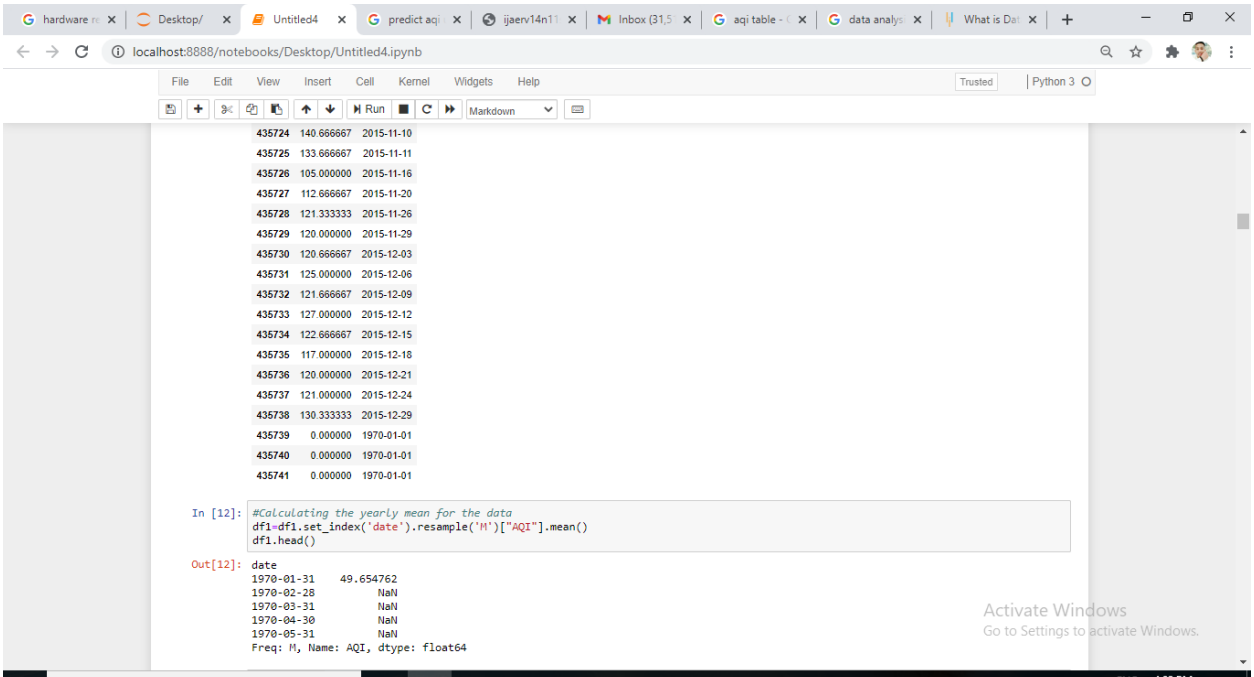
Out[10]:
```
array(['Andhra Pradesh', 'Arunachal Pradesh', 'Assam', 'Bihar',
       'Chandigarh', 'Chhattisgarh', 'Dadra & Nagar Haveli',
       'Daman & Diu', 'Delhi', 'Goa', 'Gujarat', 'Haryana',
       'Himachal Pradesh', 'Jammu & Kashmir', 'Jharkhand', 'Karnataka',
       'Kerala', 'Madhya Pradesh', '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)
```

In [11]:
```
#setting up date parameter
import warnings
import itertools
import dateutil
import statsmodels.api as sm
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
%matplotlib inline
df1=data1[['AQI','date']]
df1["date"] = pd.to_datetime(df1['date'])
df1.tail(20)
```

Out[11]:

	AQI	date
435722	118.333333	2015-11-05
435723	118.666667	2015-11-07
435724	140.666667	2015-11-10
435725	133.666667	2015-11-11
435726	105.000000	2015-11-16

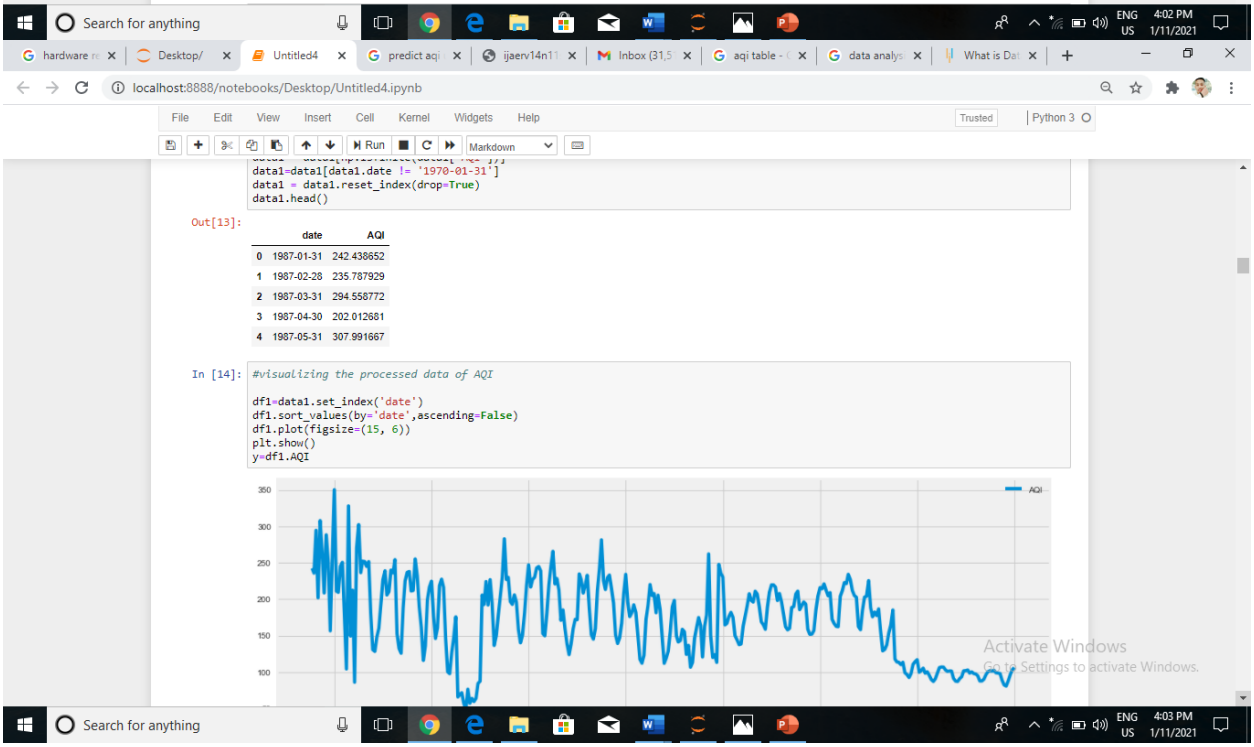
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US 4:02 PM 1/11/2021



435724	140.666667	2015-11-10
435725	133.666667	2015-11-11
435726	105.000000	2015-11-16
435727	112.666667	2015-11-20
435728	121.333333	2015-11-26
435729	120.000000	2015-11-29
435730	120.666667	2015-12-03
435731	125.000000	2015-12-06
435732	121.666667	2015-12-09
435733	127.000000	2015-12-12
435734	122.666667	2015-12-15
435735	117.000000	2015-12-18
435736	120.000000	2015-12-21
435737	121.000000	2015-12-24
435738	130.333333	2015-12-29
435739	0.000000	1970-01-01
435740	0.000000	1970-01-01
435741	0.000000	1970-01-01

In [12]:
```
#Calculating the yearly mean for the data
df1=df1.set_index('date').resample('M')["AQI"].mean()
df1.head()
```

Out[12]:
```
date
1970-01-31    49.654762
1970-02-28          NaN
1970-03-31          NaN
1970-04-30          NaN
1970-05-31          NaN
Freq: M, Name: AQI, dtype: float64
```

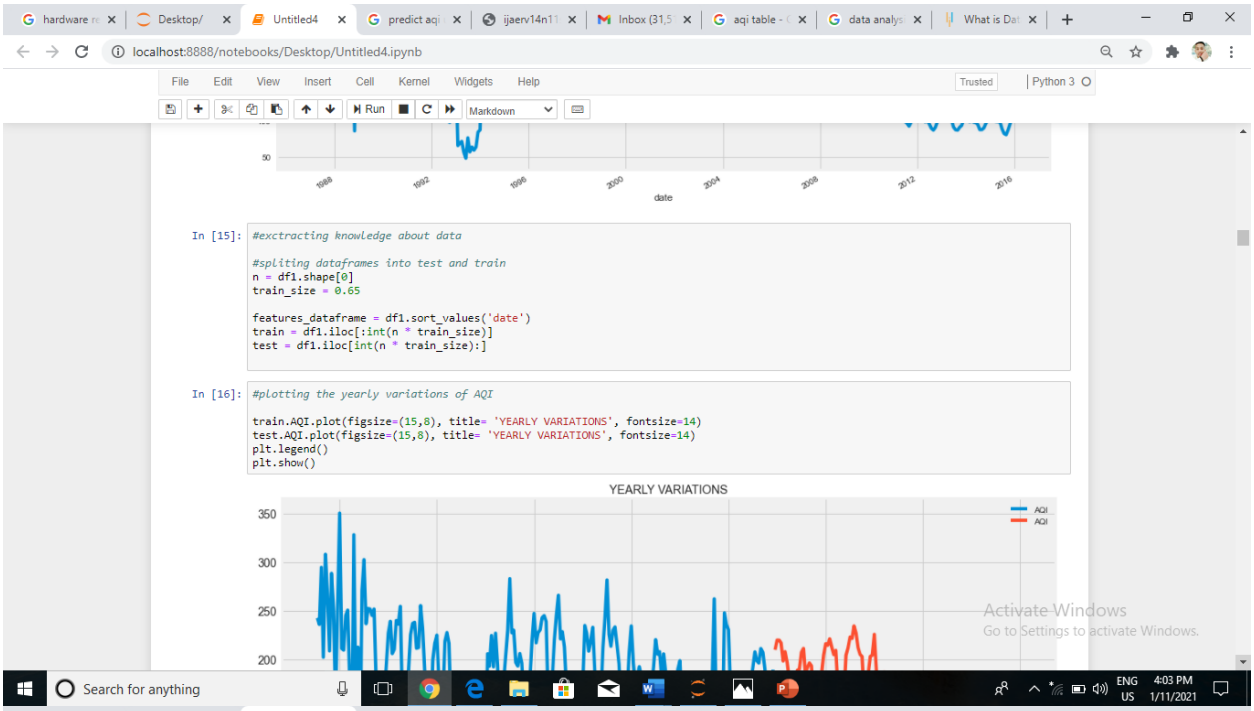



In [13]:
```
data1=data1[data1.date != '1970-01-31']
data1 = data1.reset_index(drop=True)
data1.head()
```

Out[13]:

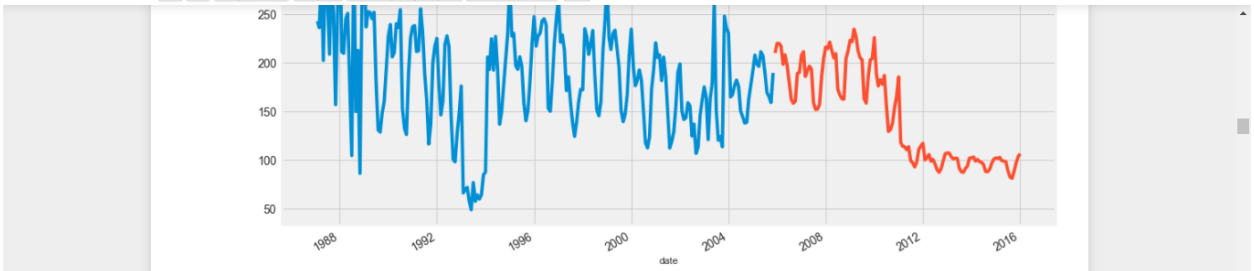
	date	AQI
0	1987-01-31	242.438652
1	1987-02-28	235.787929
2	1987-03-31	294.558772
3	1987-04-30	202.012681
4	1987-05-31	307.991667

In [14]:
```
#visualizing the processed data of AQI

df1=data1.set_index('date')
df1.sort_values(by='date',ascending=False)
df1.plot(figsize=(15, 6))
plt.show()
y=df1.AQI
```

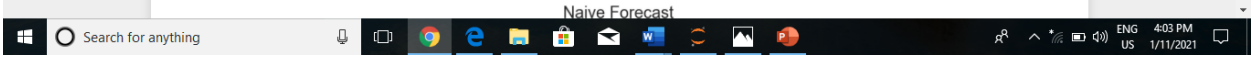


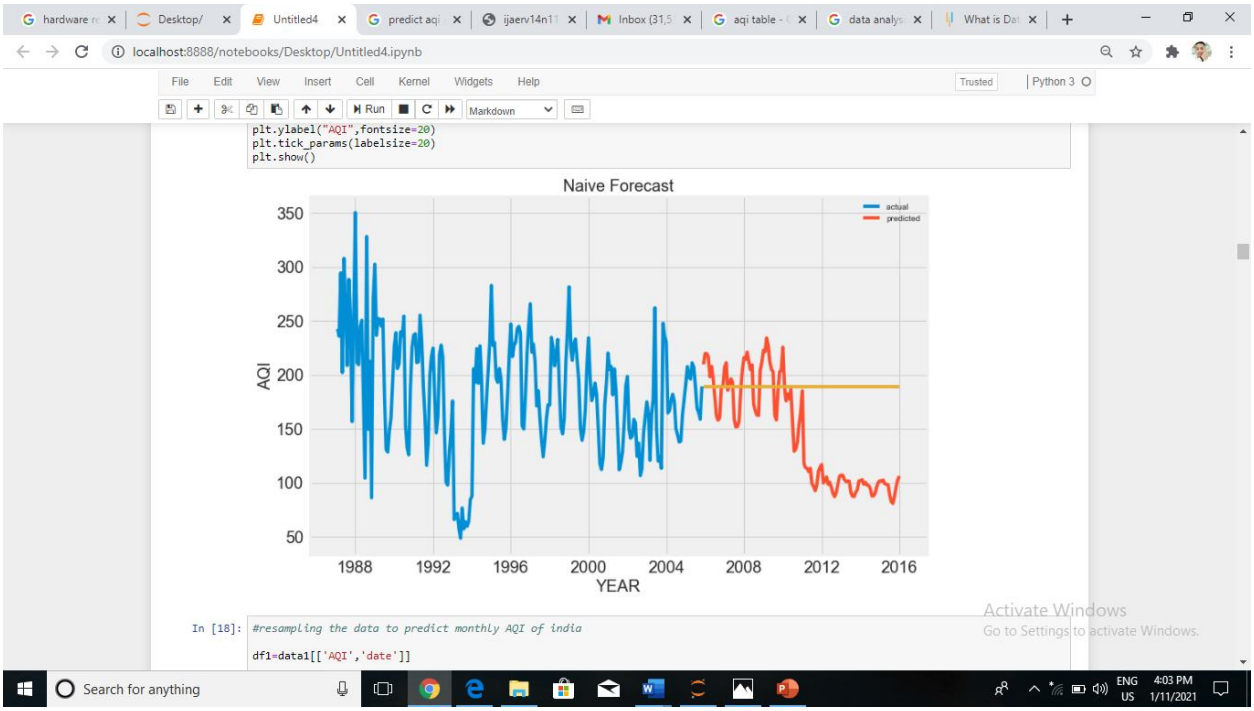
In [15]:
```
#exctracting knowledge about data

#spliting dataframes into test and train
n = df1.shape[0]
train_size = 0.65

features_dataframe = df1.sort_values('date')
train = df1.iloc[:int(n * train_size)]
test = df1.iloc[int(n * train_size):]
```

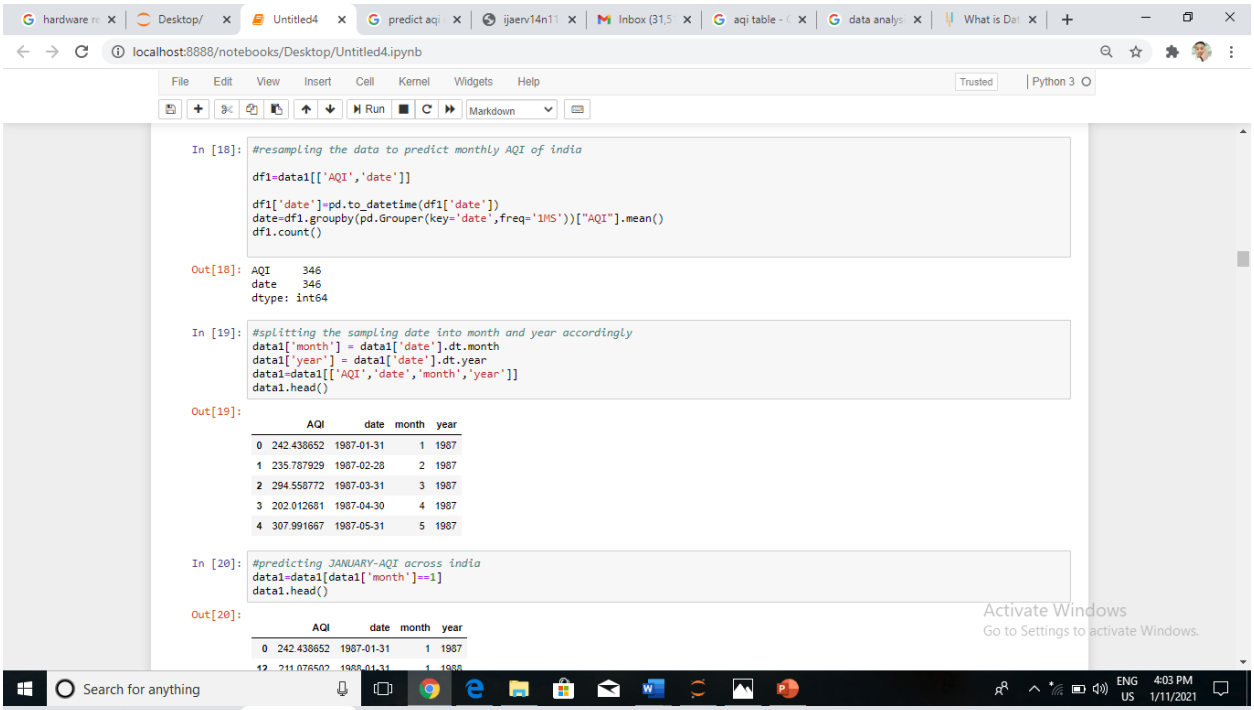
In [16]:
```
#plotting the yearly variations of AQI

train.AQI.plot(figsize=(15,8), title= 'YEARLY VARIATIONS', fontsize=14)
test.AQI.plot(figsize=(15,8), title= 'YEARLY VARIATIONS', fontsize=14)
plt.legend()
plt.show()
```

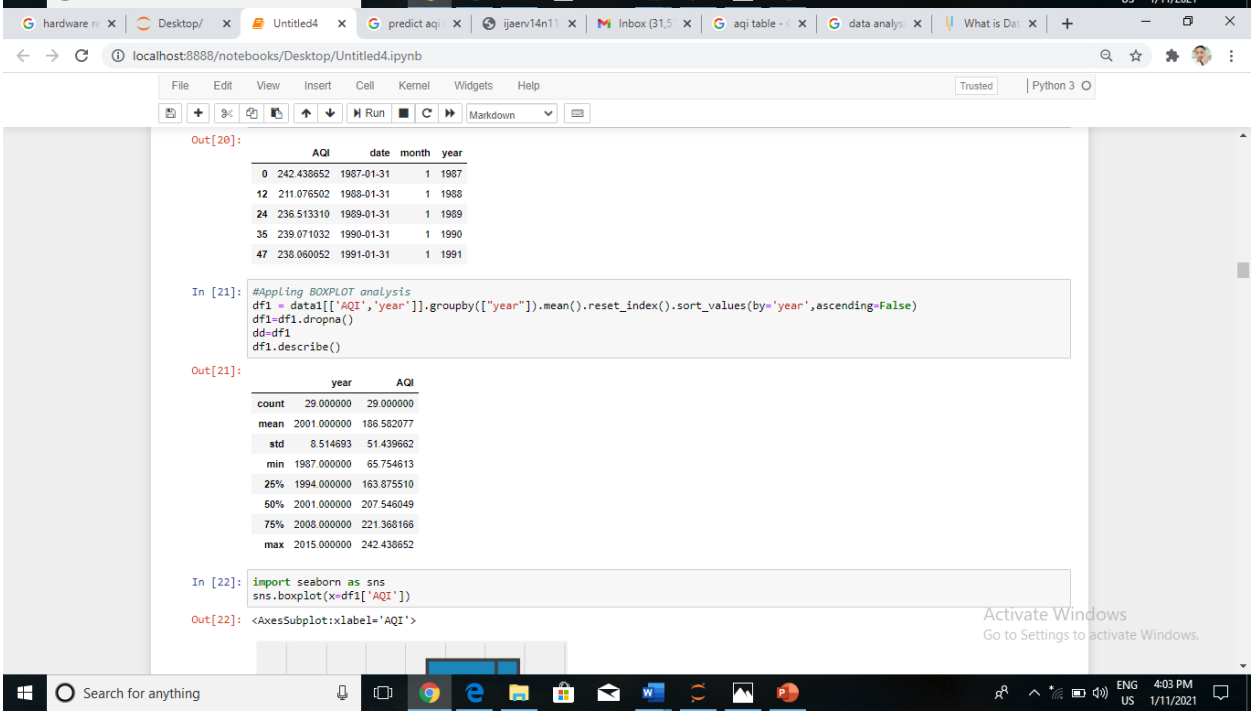



In [17]:
```
#Naive Forecast Approach to find the variations(trend)

dd= np.asarray(train.AQI)
y_hat = test.copy()
y_hat['naive'] = dd[len(dd)-1]
plt.figure(figsize=(12,8))
plt.plot(train.index, train['AQI'], label='Train')
plt.plot(test.index,test['AQI'], label='Test')
plt.plot(y_hat.index,y_hat['naive'], label='Naive Forecast')
plt.legend(loc='best')
plt.title("Naive Forecast",fontsize=20)

plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
```

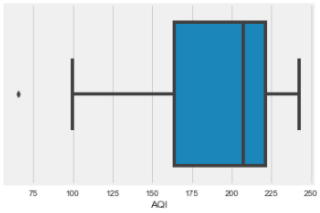



localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
Naive Forecast
actual
predicted
AQI
YEAR
350
300
250
200
150
100
50
1988
1992
1996
2000
2004
2008
2012
2016
In [18]:
#resampling the data to predict monthly AQI of india
df1=data1[['AQI','date']]
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
US
4:03 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
In [18]:
#resampling the data to predict monthly AQI of india
df1=data1[['AQI','date']]
df1['date']=pd.to_datetime(df1['date'])
date=df1.groupby(pd.Grouper(key='date',freq='1MS'))["AQI"].mean()
df1.count()
Out[18]:
AQI 346
date 346
dtype: int64
In [19]:
#splitting the sampling date into month and year accordingly
data1['month'] = data1['date'].dt.month
data1['year'] = data1['date'].dt.year
data1=data1[['AQI','date','month','year']]
data1.head()
Out[19]:
AQI date month year
0 242.438652 1987-01-31 1 1987
1 235.787929 1987-02-28 2 1987
2 294.558772 1987-03-31 3 1987
3 202.012681 1987-04-30 4 1987
4 307.991667 1987-05-31 5 1987
In [20]:
#predicting JANUARY-AQI across india
data1=data1[data1['month']==1]
data1.head()
Out[20]:
AQI date month year
0 242.438652 1987-01-31 1 1987
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:03 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Out[20]:
AQI date month year
0 242.438652 1987-01-31 1 1987
12 211.076502 1988-01-31 1 1988
24 236.513310 1989-01-31 1 1989
35 239.071032 1990-01-31 1 1990
47 238.060052 1991-01-31 1 1991
In [21]:
#Appling BOXPLOT analysis
df1 = data1[['AQI','year']].groupby(["year"]).mean().reset_index().sort_values(by='year',ascending=False)
df1=df1.dropna()
dd=df1
df1.describe()
Out[21]:
year AQI
count 29.000000 29.000000
mean 2001.000000 186.582077
std 8.514693 51.439662
min 1987.000000 65.754613
25% 1994.000000 163.875510
50% 2001.000000 207.546049
75% 2008.000000 221.368166
max 2015.000000 242.438652
In [22]:
import seaborn as sns
sns.boxplot(x=df1['AQI'])
Out[22]:
<AxesSubplot:xlabel='AQI'>
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:03 PM
1/11/2021

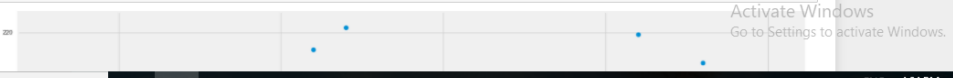
Out[22]: <AxesSubplot:xlabel='AQI'>

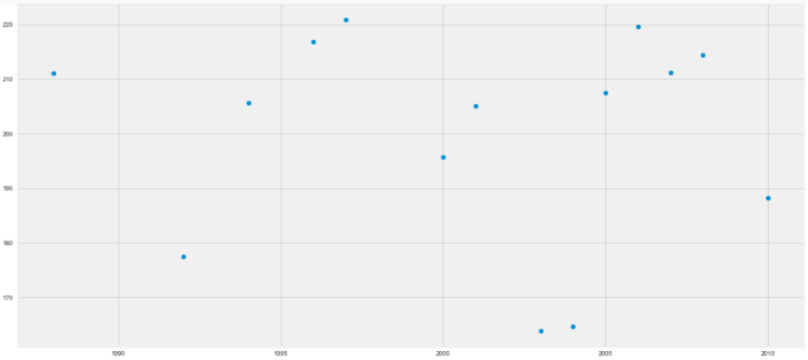


In [23]:
```
#removing Outliers
df1 = df1[np.isfinite(df1['AQI'])]
df1=df1[df1.AQI >153]
df1=df1[df1.AQI <221]
```

In [ ]:

In [25]:
```
#scatter plot of data points
cols =['year']
y = df1['AQI']
x=df1[cols]

plt.scatter(x,y)
plt.show()
```





In [26]:
```
x = (x - x.mean()) / x.std()
x = np.c_[np.ones(x.shape[0]), x]
x
```

Out[26]:
```
array([[ 1.        ,  1.40346276],
       [ 1.        ,  1.09883519],
       [ 1.        ,  0.9465214 ],
       [ 1.        ,  0.79420761],
       [ 1.        ,  0.64189382],
       [ 1.        ,  0.48958003],
       [ 1.        ,  0.33726625],
       [ 1.        ,  0.03263867],
```

```
[ 1.        ,  0.79420761],
[ 1.        ,  0.64189382],
[ 1.        ,  0.48958003],
[ 1.        ,  0.33726625],
[ 1.        ,  0.03263867],
[ 1.        , -0.11967512],
[ 1.        , -0.57661648],
[ 1.        , -0.72893027],
[ 1.        , -1.03355785],
[ 1.        , -1.33818543],
[ 1.        , -1.94744058]])
```

In [27]:

```
# Applying GRADIENT DESCENT

alpha = 0.1 #Step size
iterations = 3000 #No. of iterations
m = y.size #No. of data points
np.random.seed(4) #Setting the seed
theta = np.random.rand(2) #Picking random values to start with

def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs
past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]
#Printing the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

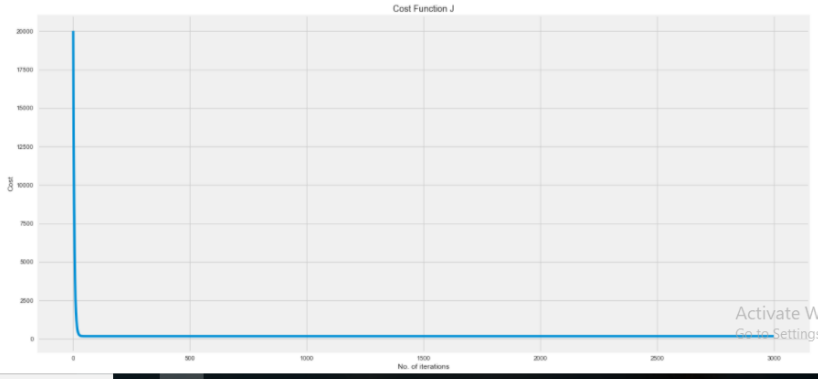
```
Gradient Descent: 200.17, -1.54
```


#Printing the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))

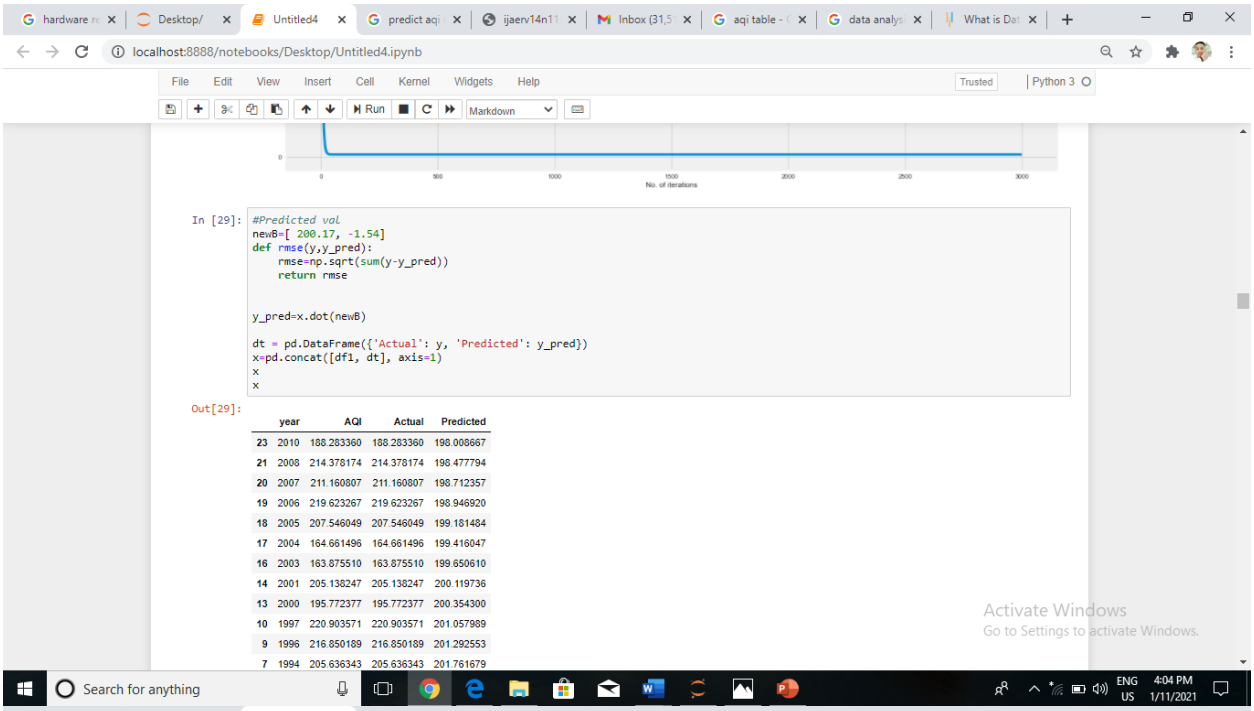
Gradient Descent: 200.17, -1.54

In [28]:

```
#Plotting the cost function...
plt.title('Cost Function J')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(past_costs)
plt.show()
```



Activate Windows
Go to Settings to activate Windows.

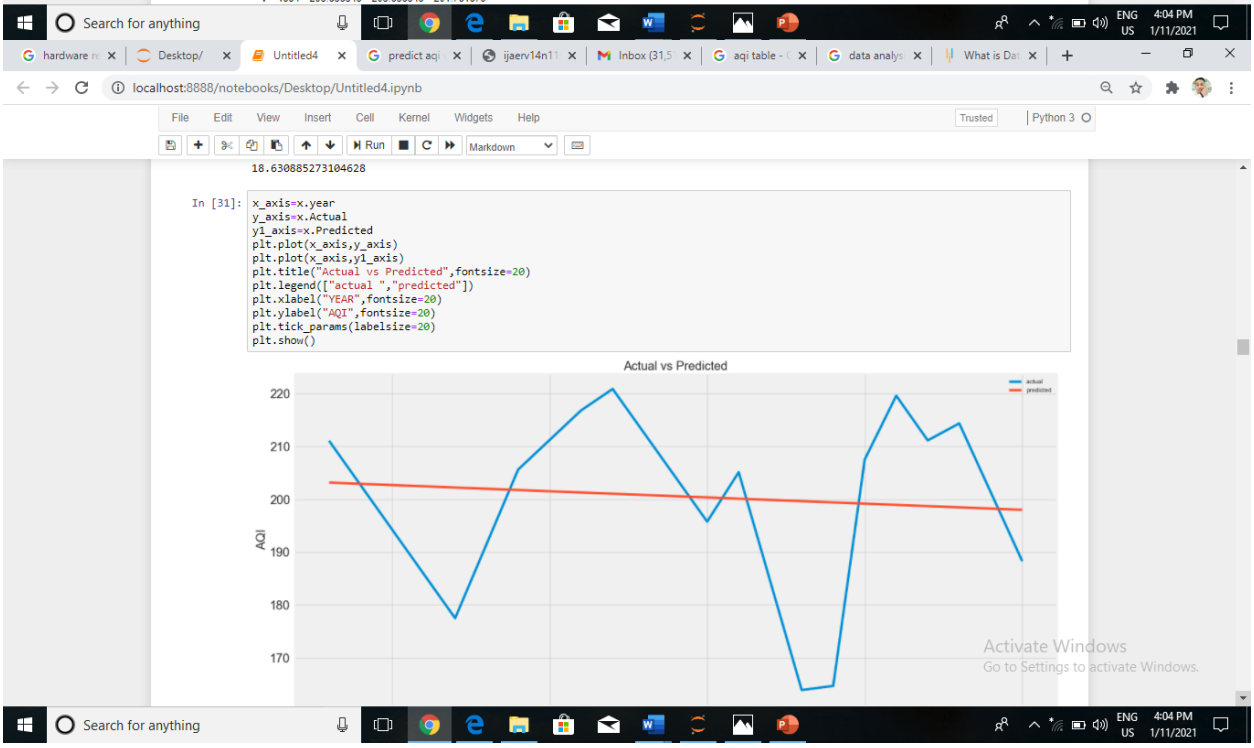


In [29]:

```
#Predicted val
newB=[ 200.17, -1.54]
def rmse(y,y_pred):
    rmse=np.sqrt(sum(y-y_pred))
    return rmse


y_pred=x.dot(newB)

dt = pd.DataFrame({'Actual': y, 'Predicted': y_pred})
x=pd.concat([df1, dt], axis=1)
x
x
```

Out[29]:

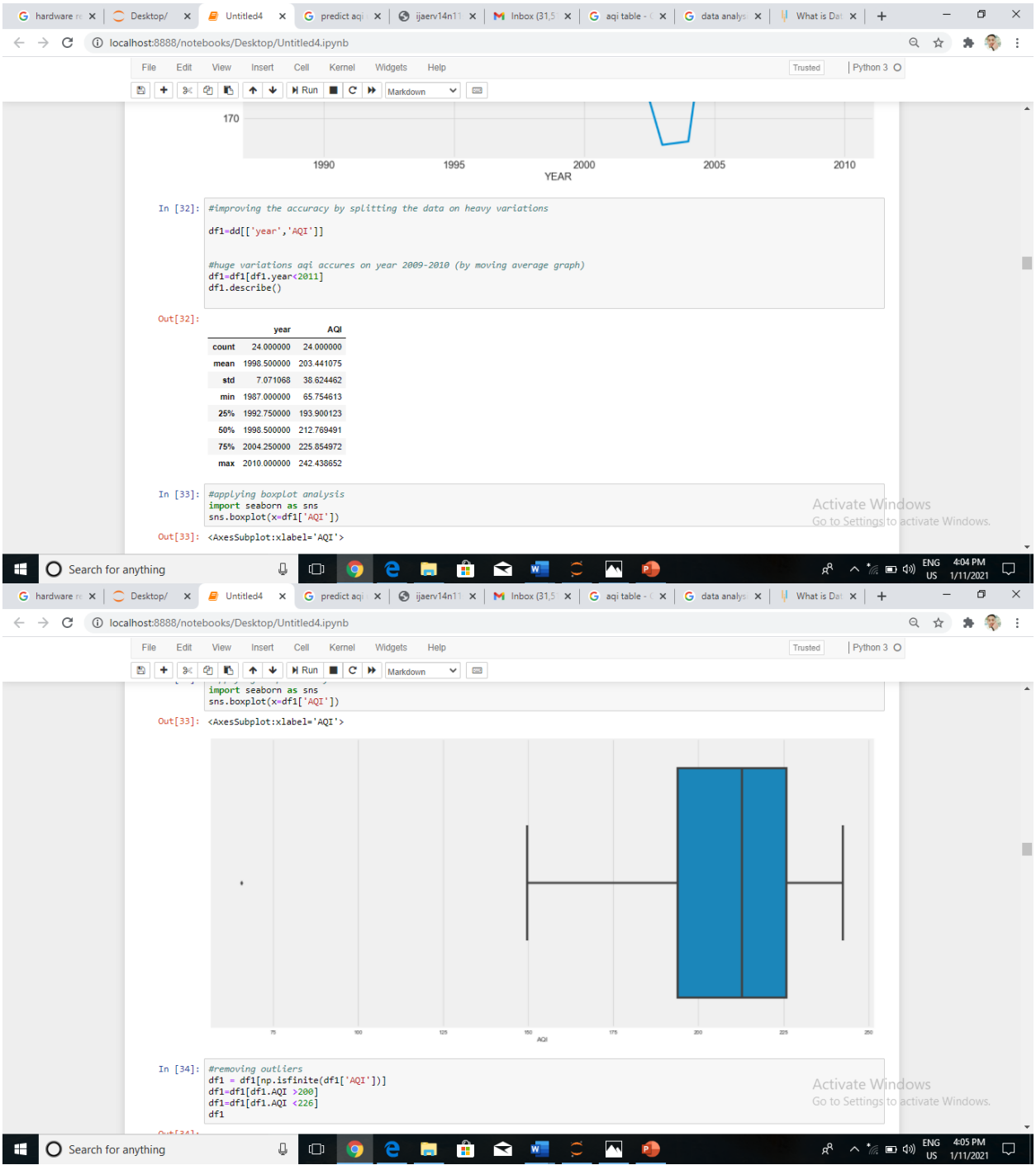
	year	AQI	Actual	Predicted
23	2010	188.283360	188.283360	198.008667
21	2008	214.378174	214.378174	198.477794
20	2007	211.160807	211.160807	198.712357
19	2006	219.623267	219.623267	198.946920
18	2005	207.546049	207.546049	199.181484
17	2004	164.661496	164.661496	199.416047
16	2003	163.875510	163.875510	199.650610
14	2001	205.138247	205.138247	200.119736
13	2000	195.772377	195.772377	200.354300
10	1997	220.903571	220.903571	201.057989
9	1996	216.850189	216.850189	201.292553
7	1994	205.636343	205.636343	201.761679



18.630885273104628

In [31]:

```
x_axis=x.year
y_axis=x.Actual
y1_axis=x.Predicted
plt.plot(x_axis,y_axis)
plt.plot(x_axis,y1_axis)
plt.title("Actual vs Predicted",fontsize=20)
plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
```


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
170
1990
1995
2000
2005
2010
YEAR
In [32]:
#improving the accuracy by splitting the data on heavy variations
df1=dd[['year','AQI']]
#huge variations aqi accures on year 2009-2010 (by moving average graph)
df1=df1[df1.year<2011]
df1.describe()
Out[32]:
year AQI
count 24.000000 24.000000
mean 1998.500000 203.441075
std 7.071068 38.624462
min 1987.000000 65.754613
25% 1992.750000 193.900123
50% 1998.500000 212.769491
75% 2004.250000 225.854972
max 2010.000000 242.438652
In [33]:
#applying boxplot analysis
import seaborn as sns
sns.boxplot(x=df1['AQI'])
Out[33]:
<AxesSubplot:xlabel='AQI'>
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:04 PM
1/11/2021

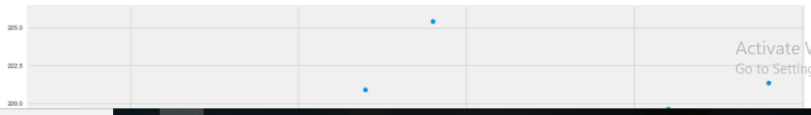
localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
import seaborn as sns
sns.boxplot(x=df1['AQI'])
Out[33]:
<AxesSubplot:xlabel='AQI'>
75
100
125
150
175
200
225
250
AQI
In [34]:
#removing outliers
df1 = df1[np.isfinite(df1['AQI'])]
df1=df1[df1.AQI >200]
df1=df1[df1.AQI <226]
df1
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:05 PM
1/11/2021

```
df1 = df1[np.isfinite(df1['AQI'])]
df1=df1[df1.AQI >200]
df1=df1[df1.AQI <226]
df1
```

Out[34]:

	year	AQI
22	2009	221.368166
21	2008	214.378174
20	2007	211.160807
19	2006	219.623267
18	2005	207.546049
14	2001	205.138247
12	1999	225.439218
10	1997	220.903571
9	1996	216.850189
7	1994	205.636343
1	1988	211.076502

In [35]:
```
#plotting data points
cols =['year']
y = df1['AQI']
x=df1[cols]

plt.scatter(x,y)
plt.show()
```



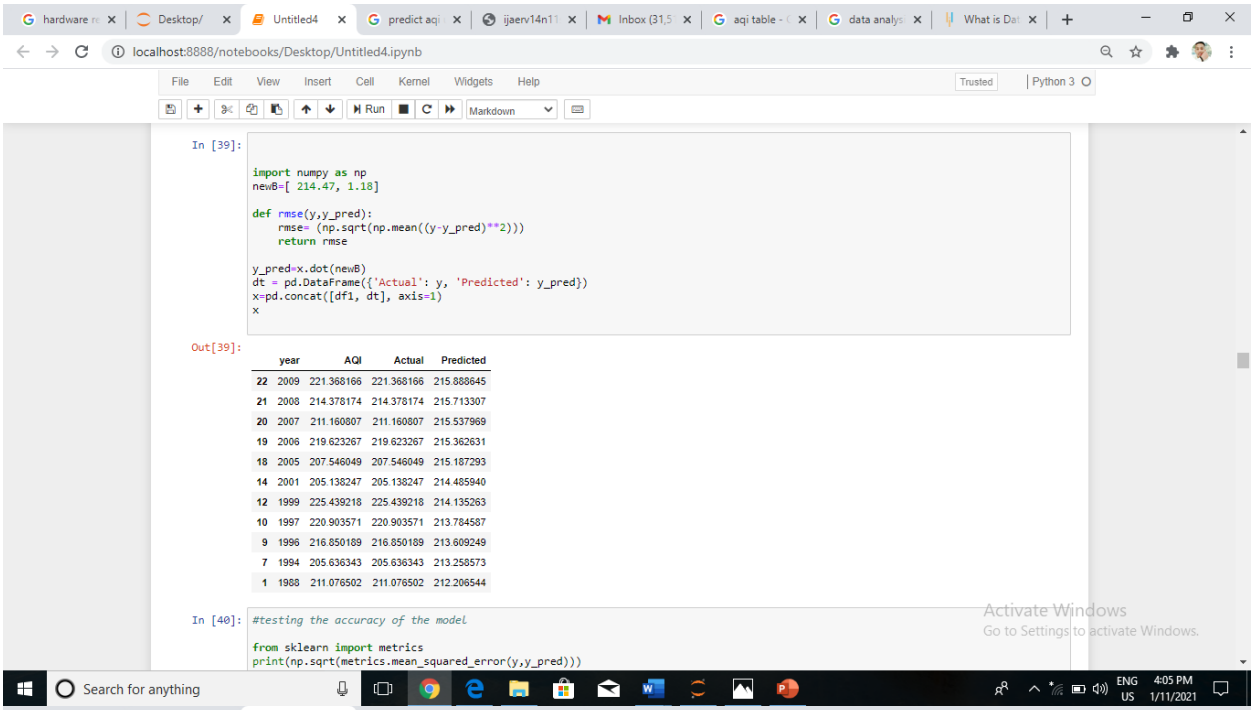
hardware re | Desktop/ | Untitled4 | predict aqi | ijaerv14n11 | Inbox (31,5 | aqi table - | data analys | What is Dat
localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted | Python 3
Run | Markdown

```
[ 1.        , -0.58085837],
[ 1.        , -0.72945005],
[ 1.        , -1.0266334 ],
[ 1.        , -1.91818345]])
```

In [37]:
```
#Tunning model with GRADIENT DESCENT

alpha = 0.1 #Step size
iterations = 3000 #No. of iterations
m = y.size #No. of data points
np.random.seed(4) #Setting the seed
theta = np.random.rand(2) #Picking some random values to start with

def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs

past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Print the results...#Print the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

```
Gradient Descent: 214.47, 1.18
```

In [38]:
```
#Plotting the cost function...
plt.title('Cost Function J')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
```



In [39]:

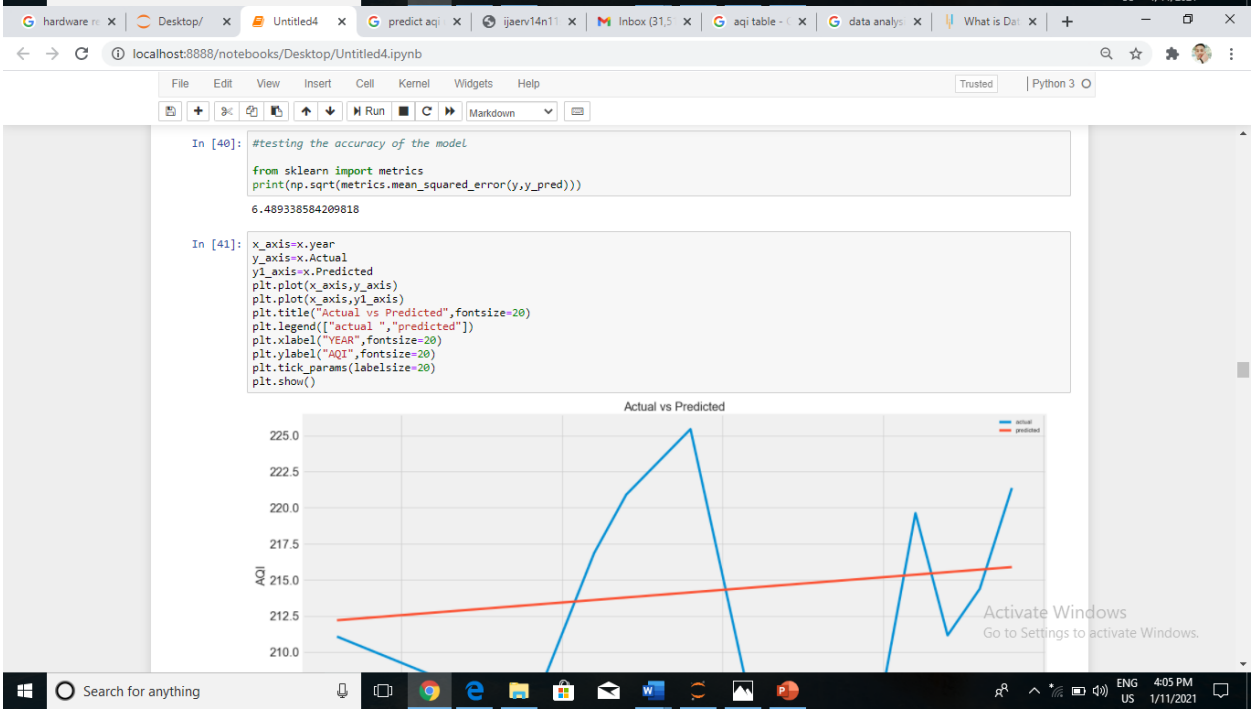
```
import numpy as np
newB=[ 214.47, 1.18]

def rmse(y,y_pred):
    rmse= (np.sqrt(np.mean((y-y_pred)**2)))
    return rmse

y_pred=x.dot(newB)
dt = pd.DataFrame({'Actual': y, 'Predicted': y_pred})
x=pd.concat([df1, dt], axis=1)
x
```

Out[39]:

	year	AQI	Actual	Predicted
22	2009	221.368166	221.368166	215.888645
21	2008	214.378174	214.378174	215.713307
20	2007	211.160807	211.160807	215.537969
19	2006	219.623267	219.623267	215.362631
18	2005	207.546049	207.546049	215.187293
14	2001	205.138247	205.138247	214.485940
12	1999	225.439218	225.439218	214.135263
10	1997	220.903571	220.903571	213.784587
9	1996	216.850189	216.850189	213.609249
7	1994	205.636343	205.636343	213.258573
1	1988	211.076502	211.076502	212.206544

In [40]:

```
#testing the accuracy of the model

from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y,y_pred)))
```

In [40]:

```
#testing the accuracy of the model

from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y,y_pred)))
```

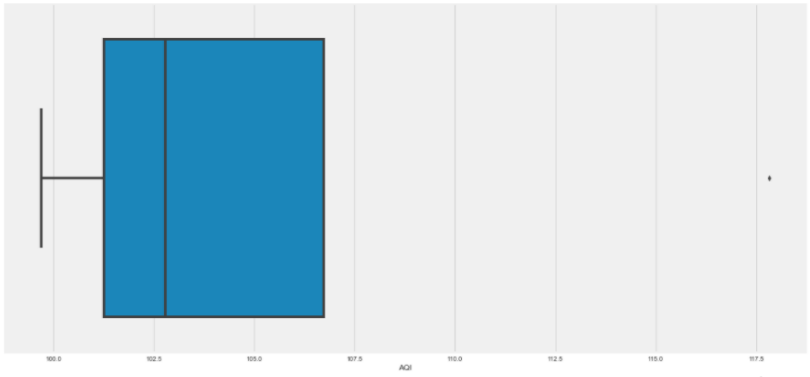
```
6.489338584209818
```

In [41]:

```
x_axis=x.year
y_axis=x.Actual
y1_axis=x.Predicted
plt.plot(x_axis,y_axis)
plt.plot(x_axis,y1_axis)
plt.title("Actual Vs Predicted",fontsize=20)
plt.legend(["actual ","predicted"])
plt.xlabel("YEAR",fontsize=20)
plt.ylabel("AQI",fontsize=20)
plt.tick_params(labelsize=20)
plt.show()
```

```
In [43]: #applying boxplot
         import seaborn as sns
         sns.boxplot(x=df1['AQI'])
```

```
Out[43]: <AxesSubplot:xlabel='AQI'>
```




```
In [44]: df1 = df1[np.isfinite(df1['AQI'])]
         df1=df1[df1.AQI >101]
         df1=df1[df1.AQI <107]
         df1.head()
```



```
        for i in range(iterations):
            prediction = np.dot(x, theta)
            error = prediction - y
            cost = 1/(2*m) * np.dot(error.T, error)
            past_costs.append(cost)
            theta = theta - (alpha * (1/m) * np.dot(x.T, error))
            past_thetas.append(theta)

    return past_thetas, past_costs

past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Print the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))
```

```
Gradient Descent: 103.59, -2.74
```

```
In [48]: #Plotting the cost function...
         plt.title('Cost Function J',fontsize=28)
         plt.xlabel('No. of iterations',fontsize=25)
         plt.ylabel('Cost',fontsize=25)
         plt.plot(past_costs)
         plt.tick_params(labelsize=30)
         plt.show()
```

In [52]:
```
#Prediction for the future
from sklearn.preprocessing import MinMaxScaler
```

In [53]:
```
#feeding in the x value-years
data1=[[-1,2016],[-1,2017],[-1,2018],[-1,2019],[-1,2020]]
```

In [54]:
```
#normalization
scaler=MinMaxScaler(feature_range=(-1,1))
scaler.fit(data1)
x=scaler.transform(data1)
```

In [55]:
```
#calculations
newB=[103.59,-2.74]
ypred=-(x.dot(newB))
```


In [55]:
```
#calculations
newB=[103.59,-2.74]
ypred=-(x.dot(newB))
```

In [56]:
```
#AQI for the year 2020
print("AQI for the year 2020===>",ypred[-1])
```

```
AQI for the year 2020===> 106.33
```

PART 2

In [5]:
```
# Missing values
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
```

```
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns

missing_values= missing_values_table(data)
missing_values.style.background_gradient(cmap='Reds')
```

```
Your selected dataframe has 16 columns.
There are 14 columns that have missing values.
```

Out[5]:

	Missing Values	% of Total Values
Xylene	18109	61.300000
PM10	11140	37.700000
NH3	10328	35.000000
Toluene	8041	27.200000
Benzene	5623	19.000000
AQI	4681	15.900000
AQI_Bucket	4681	15.900000
PM2.5	4598	15.600000
NOx	4185	14.200000
O3	4022	13.600000
SO2	3854	13.100000
NO2	3585	12.100000
NO	3582	12.100000
CO	2059	7.000000

CITIES IN THE DATASET

CITIES IN THE DATASET

In [6]:
```
df.City.unique()
```

Out[6]:
```
array(['Ahmedabad', 'Aizawl', 'Amaravati', 'Amritsar', 'Bengaluru',
       'Bhopal', 'Brajrajnagar', 'Chandigarh', 'Chennai', 'Coimbatore',
       'Delhi', 'Ernakulam', 'Gurugram', 'Guwahati', 'Hyderabad',
       'Jaipur', 'Jorapokhar', 'Kochi', 'Kolkata', 'Lucknow', 'Mumbai',
       'Patna', 'Shillong', 'Talcher', 'Thiruvananthapuram',
       'Visakhapatnam'], dtype=object)
```

CONVERT THE DATETIME FORMAT

In [7]:
```
# Convert string to datetime64
data['Date'] = pd.to_datetime(data['Date'])
#city_day.set_index('Date',inplace=True)
```

DATA AVAILABILITY

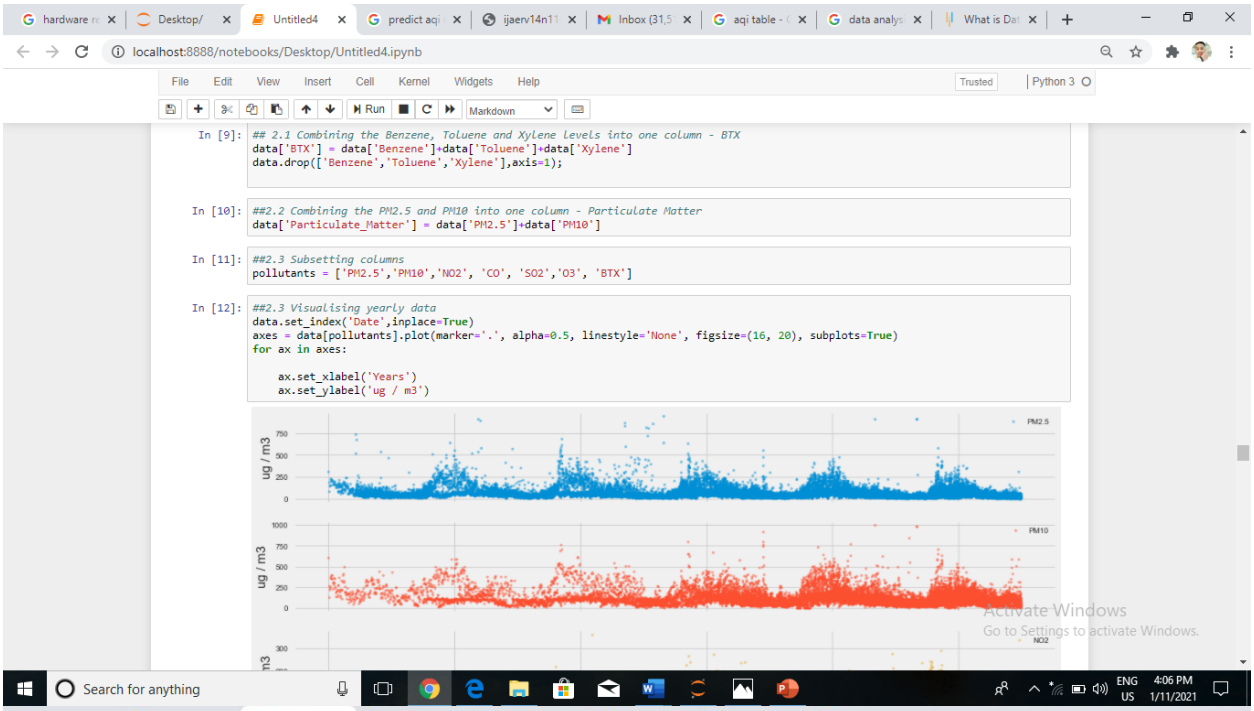
In [8]:
```
print(f"The available data is between {data['Date'].min()} and {data['Date'].max()}")
```

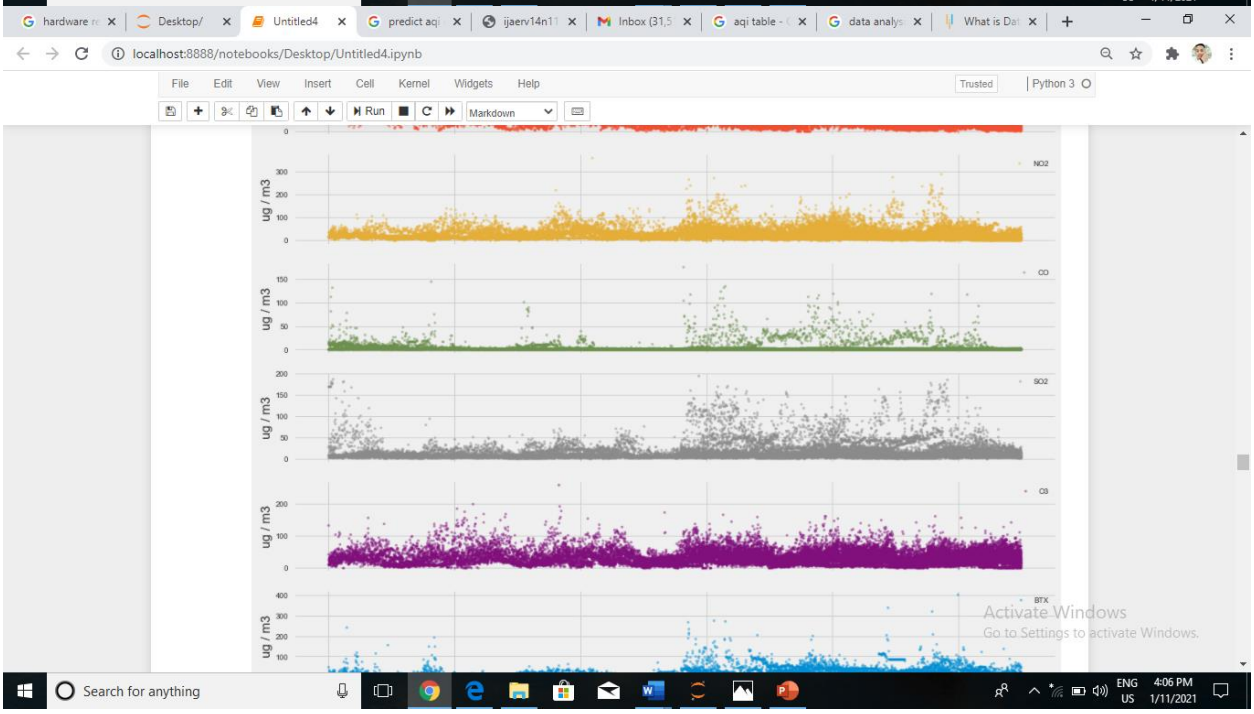
```
The available data is between 2015-01-01 00:00:00 and 2020-07-01 00:00:00
```

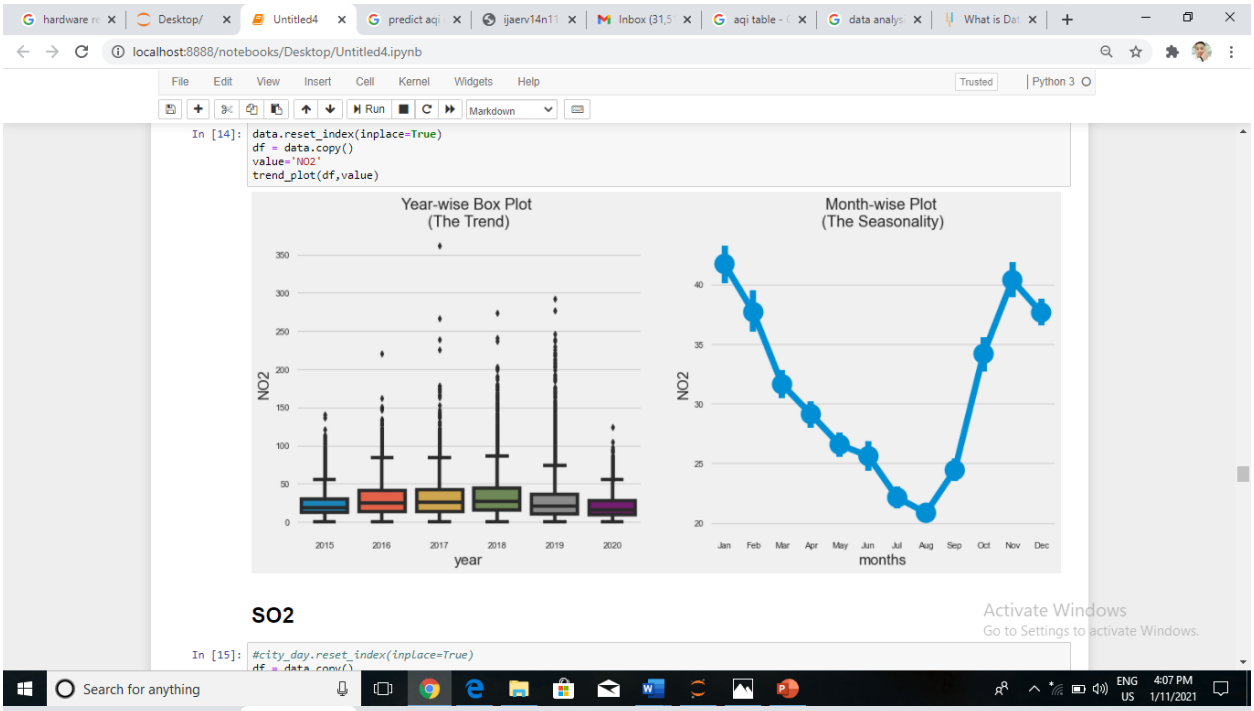
ANALYSING THE COMPLETE CITY LEVEL DAILY DATA

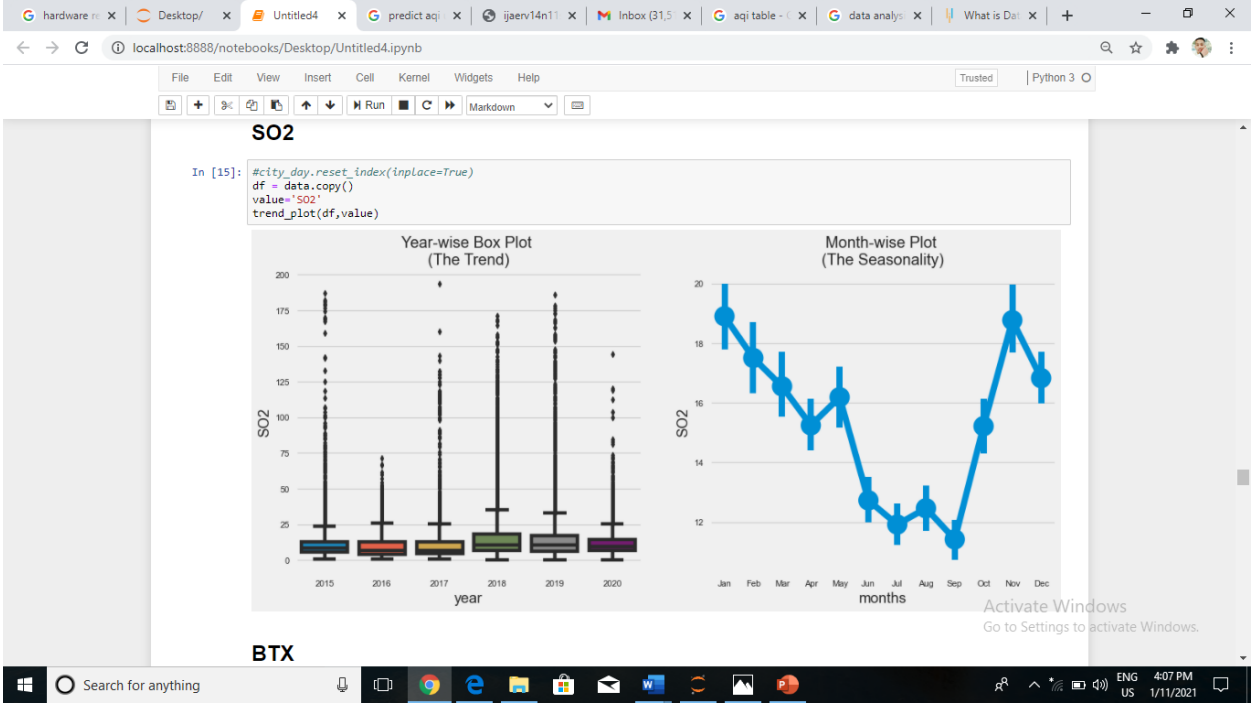
In [9]:
```
## 2.1 Combining the Benzene, Toluene and Xylene levels into one column - BTX
data['BTX'] = data['Benzene']+data['Toluene']+data['Xylene']
data.drop(['Benzene','Toluene','Xylene'],axis=1);
```

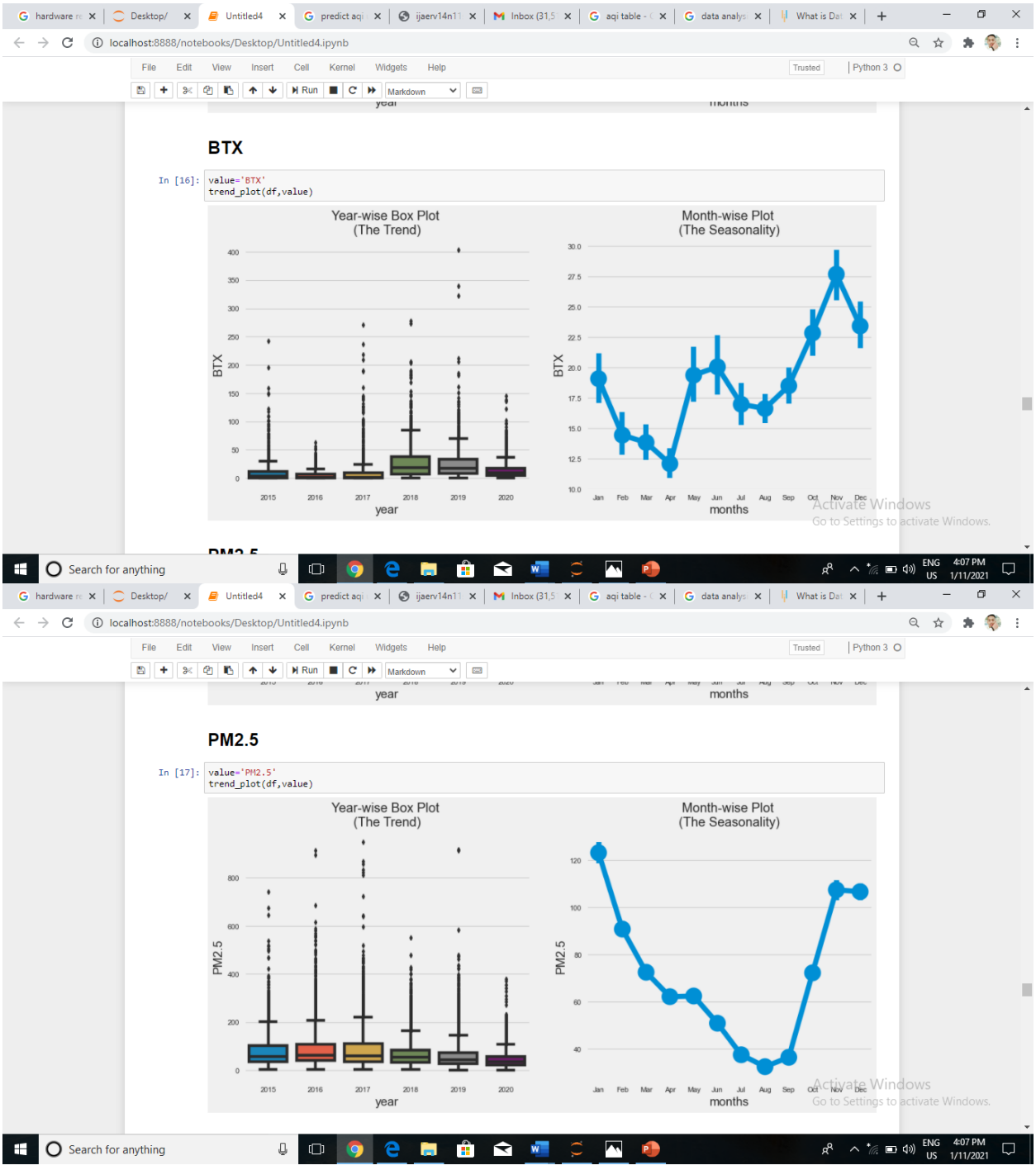
Activate Windows
Go to Settings to activate Windows.


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
In [9]:
2.1 Combining the Benzene, Toluene and Xylene levels into one column - BTX
data['BTX'] = data['Benzene']+data['Toluene']+data['Xylene']
data.drop(['Benzene','Toluene','Xylene'],axis=1);
In [10]:
##2.2 Combining the PM2.5 and PM10 into one column - Particulate Matter
data['Particulate_Matter'] = data['PM2.5']+data['PM10']
In [11]:
##2.3 Subsetting columns
pollutants = ['PM2.5','PM10','NO2', 'CO', 'SO2','O3', 'BTX']
In [12]:
##2.3 Visualising yearly data
data.set_index('Date',inplace=True)
axes = data[pollutants].plot(marker='.', alpha=0.5, linestyle='None', figsize=(16, 20), subplots=True)
for ax in axes:
ax.set_xlabel('Years')
ax.set_ylabel('ug / m3')
PM2.5
PM10
NO2
ug / m3
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
US
4:06 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
NO2
CO
SO2
O3
BTX
ug / m3
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
US
4:06 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
In [14]:
data.reset_index(inplace=True)
df = data.copy()
value='NO2'
trend_plot(df,value)
Year-wise Box Plot
(The Trend)
NO2
year
Month-wise Plot
(The Seasonality)
NO2
months
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
SO2
In [15]:
#city_day.reset_index(inplace=True)
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:07 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
SO2
In [15]:
#city_day.reset_index(inplace=True)
df = data.copy()
value='SO2'
trend_plot(df,value)
Year-wise Box Plot
(The Trend)
SO2
year
Month-wise Plot
(The Seasonality)
SO2
months
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Activate Windows
Go to Settings to activate Windows.
BTX
Search for anything
ENG US
4:07 PM
1/11/2021


localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
BTX
In [16]:
value='BTX'
trend_plot(df,value)
Year-wise Box Plot
(The Trend)
Month-wise Plot
(The Seasonality)
BTX
year
months
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG
US
4:07 PM
1/11/2021
PM2.5
In [17]:
value='PM2.5'
trend_plot(df,value)
PM2.5

MOST POLLUTED INDIAN CITIES

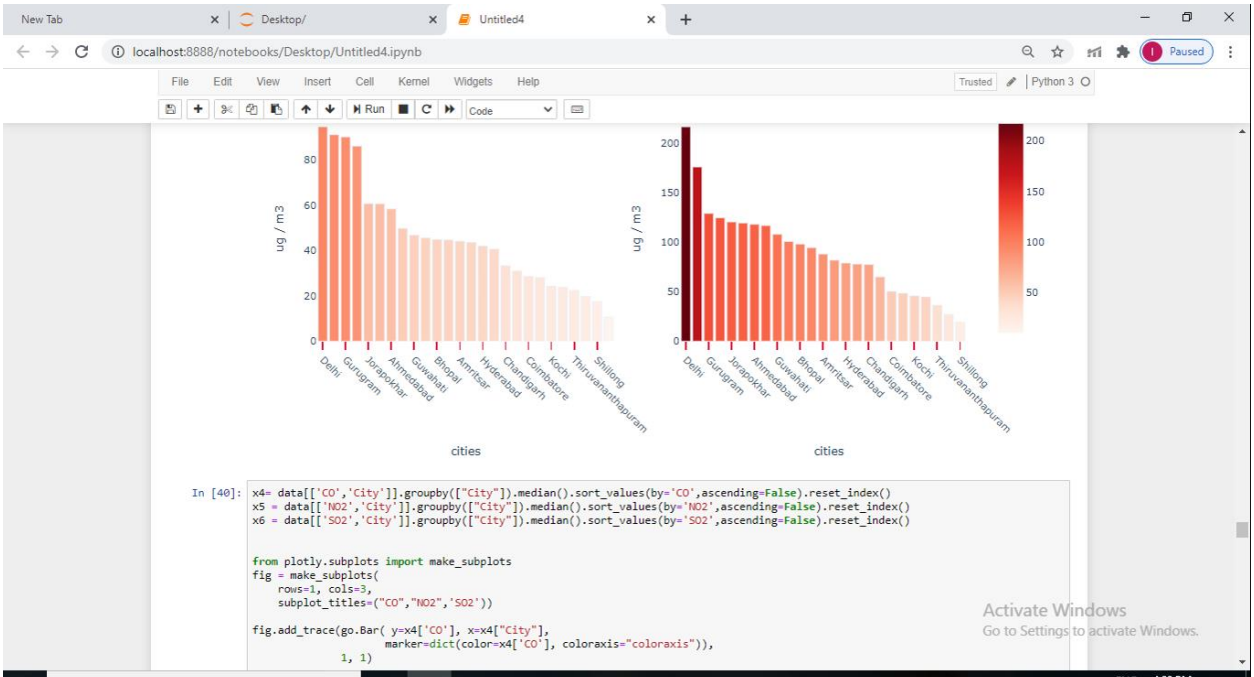
In [19]:
```
def max_polluted_city(pollutant):
    x1 = data[[pollutant,'City']].groupby(["City"]).mean().sort_values(by=pollutant,ascending=False).reset_index()
    x1[pollutant] = round(x1[pollutant],2)
    return x1[:10].style.background_gradient(cmap='OrRd')
```

In [20]:
```
#source: https://stackoverflow.com/questions/38783027/jupyter-notebook-display-two-pandas-tables-side-by-side
from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.render()
    display_html(html_str.replace('table','table style="display:inline"'),raw=True)
```

In [21]:
```
pm2_5 = max_polluted_city('PM2.5')
pm10 = max_polluted_city('PM10')
no2 = max_polluted_city('NO2')
so2 = max_polluted_city('SO2')
co = max_polluted_city('CO')
btx = max_polluted_city('BTX')


display_side_by_side(pm2_5,pm10,no2,so2,co,btx)
```

	City	PM2.5		City	PM10		City	NO2		City	SO2		City	CO
0	Patna	123.500000	0	Delhi	232.810000	0	Ahmedabad	59.030000	0	Ahmedabad	55.250000	0	Ahmedabad	22.190000
1	Delhi	117.200000	1	Gurugram	191.500000	1	Delhi	50.790000	1	Jorapokhar	33.650000	1	Lucknow	2.130000
2	Gurugram	117.100000	2	Talcher	165.770000	2	Kolkata	40.400000	2	Talcher	28.490000	2	Delhi	1.980000
3	Lucknow	109.710000	3	Jorapokhar	149.660000	3	Patna	37.490000	3	Patna	22.130000	3	Talcher	1.850000
4	Ahmedabad	67.850000	4	Patna	126.750000	4	Visakhapatnam	37.190000	4	Kochi	17.600000	4	Bengaluru	1.840000
5	Kolkata	64.360000	5	Brajrajnagar	124.220000	5	Lucknow	33.240000	5	Delhi	15.900000	5	Brajrajnagar	1.800000
6	Jorapokhar	64.230000	6	Jaipur	123.480000	6	Jaipur	32.420000	6	Mumbai	15.200000	6	Ernakulam	1.630000

In [4]:
```
x2= data[['PM2.5','City']].groupby(["City"]).median().sort_values(by='PM2.5',ascending=False).reset_index()
x3 = data[['PM10','City']].groupby(["City"]).median().sort_values(by='PM10',ascending=False).reset_index()

import plotly.io as pio
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import plot, iplot


fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=("PM2.5","PM10"))

fig.add_trace(go.Bar( y=x2['PM2.5'], x=x2["City"],
                      marker=dict(color=x2['PM2.5'], coloraxis="coloraxis")),
              1, 1)


fig.add_trace(go.Bar( y=x3['PM10'], x=x2["City"],
                      marker=dict(color=x3['PM10'], coloraxis="coloraxis")),
              1, 2)
fig.update_layout(coloraxis=dict(colorscale='reds'), showlegend=False,plot_bgcolor='white')
fig.update_xaxes(ticks="outside", tickwidth=2,tickangle=45, tickcolor='crimson', ticklen=10,title_text="cities")
fig.update_yaxes(title_text="ug / m3", row=1, col=1)
fig.update_yaxes(title_text="ug / m3", row=1, col=2)
fig.show();
```




```
In [40]: x4= data[['CO','City']].groupby(["City"]).median().sort_values(by='CO',ascending=False).reset_index()
         x5 = data[['NO2','City']].groupby(["City"]).median().sort_values(by='NO2',ascending=False).reset_index()
         x6 = data[['SO2','City']].groupby(["City"]).median().sort_values(by='SO2',ascending=False).reset_index()


         from plotly.subplots import make_subplots
         fig = make_subplots(
             rows=1, cols=3,
             subplot_titles=("CO","NO2",'SO2'))

         fig.add_trace(go.Bar( y=x4['CO'], x=x4["City"],
                               marker=dict(color=x4['CO'], coloraxis="coloraxis")),
                       1, 1)
```


EFFECT OF LOCKDOWN IN AQI

AQI for some of the major cities of India

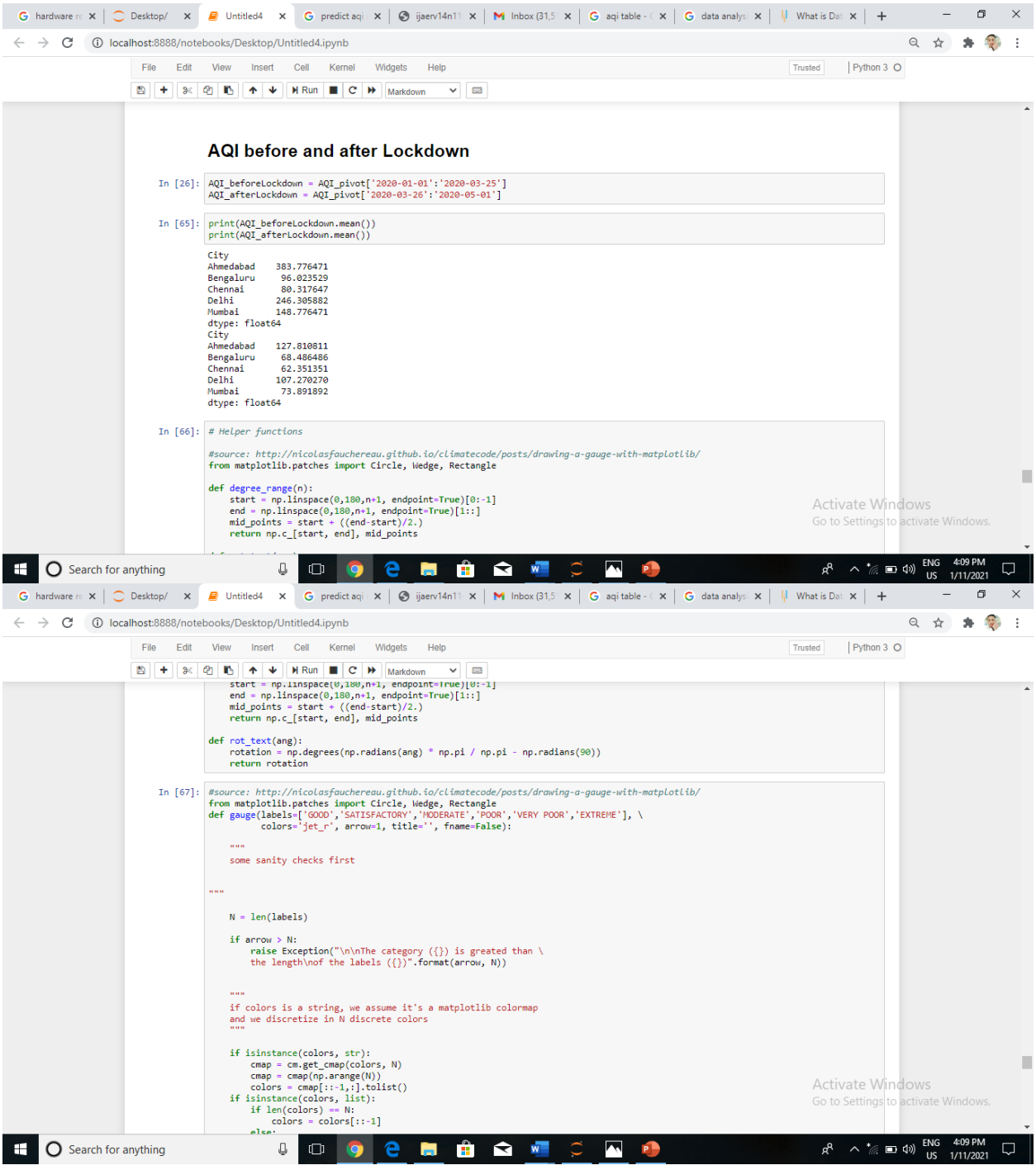
```
In [24]: cities = ['Ahmedabad','Delhi','Bengaluru','Mumbai','Chennai']

         filtered_data = data[data['Date'] >= '2019-01-01']
         AQI = filtered_data[filtered_data.City.isin(cities)][['Date','City','AQI','AQI_Bucket']]
         AQI.head()
```

Out[24]:

	Date	City	AQI	AQI_Bucket
1461	2019-01-01	Ahmedabad	1474.0	Severe
1462	2019-01-02	Ahmedabad	1246.0	Severe
1463	2019-01-03	Ahmedabad	1719.0	Severe
1464	2019-01-04	Ahmedabad	1264.0	Severe
1465	2019-01-05	Ahmedabad	1127.0	Severe

```
In [64]: AQI_pivot = AQI.pivot(index='Date', columns='City', values='AQI')
         AQI_pivot.fillna(method='bfill',inplace=True)


         from plotly.subplots import make_subplots
         import plotly.graph_objects as go

         fig = make_subplots(
             rows=6, cols=1,
             #specs=[[{}, {}],
                    # [{"colspan": 6}, None]],
             subplot_titles=("Ahmedabad","Bengaluru","Chennai","Delhi",'Mumbai'))

         fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Ahmedabad'],
                              marker=dict(color=AQI_pivot['Ahmedabad'],coloraxis="coloraxis")),
```

```
        # [{"colspan": 6}, None]],
    subplot_titles=("Ahmedabad","Bengaluru","Chennai","Delhi",'Mumbai'))

fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Ahmedabad'],
                     marker=dict(color=AQI_pivot['Ahmedabad'],coloraxis="coloraxis")),
              1, 1)
fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Bengaluru'],
                     marker=dict(color=AQI_pivot['Bengaluru'], coloraxis="coloraxis")),
              2, 1)
fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Chennai'],
                     marker=dict(color=AQI_pivot['Chennai'], coloraxis="coloraxis")),
              3, 1)
fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Delhi'],
                     marker=dict(color=AQI_pivot['Delhi'], coloraxis="coloraxis")),
              4, 1)

fig.add_trace(go.Bar(x=AQI_pivot.index, y=AQI_pivot['Mumbai'],
                     marker=dict(color=AQI_pivot['Mumbai'], coloraxis="coloraxis")),
              5, 1)

fig.update_layout(coloraxis=dict(colorscale='Temps'),showlegend=False,title_text="AQI Levels")

fig.update_layout(plot_bgcolor='white')

fig.update_layout( width=800,height=1200,shapes=[
      dict(
      type= 'line',
      yref= 'paper', y0= 0, y1= 1,
      xref= 'x', x0= '2020-03-25', x1= '2020-03-25'
      )
])

fig.show()
```

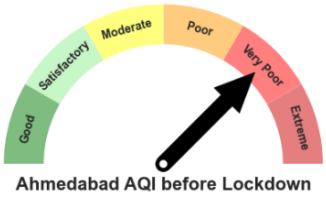

Untitled4
localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
AQI before and after Lockdown
In [26]:
AQI_beforeLockdown = AQI_pivot['2020-01-01':'2020-03-25']
AQI_afterLockdown = AQI_pivot['2020-03-26':'2020-05-01']
In [65]:
print(AQI_beforeLockdown.mean())
print(AQI_afterLockdown.mean())
City
Ahmedabad 383.776471
Bengaluru 96.023529
Chennai 80.317647
Delhi 246.305882
Mumbai 148.776471
dtype: float64
City
Ahmedabad 127.810811
Bengaluru 68.486486
Chennai 62.351351
Delhi 107.270270
Mumbai 73.891892
dtype: float64
In [66]:
Helper functions
#source: http://nicolasfauchereau.github.io/climatecode/posts/drawing-a-gauge-with-matplotlib/
from matplotlib.patches import Circle, Wedge, Rectangle
def degree_range(n):
start = np.linspace(0,180,n+1, endpoint=True)[0:-1]
end = np.linspace(0,180,n+1, endpoint=True)[1::]
mid_points = start + ((end-start)/2.)
return np.c_[start, end], mid_points
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:09 PM
1/11/2021

Untitled4
localhost:8888/notebooks/Desktop/Untitled4.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
start = np.linspace(0,180,n+1, endpoint=True)[0:-1]
end = np.linspace(0,180,n+1, endpoint=True)[1::]
mid_points = start + ((end-start)/2.)
return np.c_[start, end], mid_points
def rot_text(ang):
rotation = np.degrees(np.radians(ang) * np.pi / np.pi - np.radians(90))
return rotation
In [67]:
#source: http://nicolasfauchereau.github.io/climatecode/posts/drawing-a-gauge-with-matplotlib/
from matplotlib.patches import Circle, Wedge, Rectangle
def gauge(labels=['GOOD','SATISFACTORY','MODERATE','POOR','VERY POOR','EXTREME'], \
colors='jet_r', arrow=1, title='', fname=False):
"""
some sanity checks first
"""
N = len(labels)
if arrow > N:
raise Exception("\n\nThe category ({}) is greated than \
the length\nof the labels ({})".format(arrow, N))
"""
if colors is a string, we assume it's a matplotlib colormap
and we discretize in N discrete colors
"""
if isinstance(colors, str):
cmap = cm.get_cmap(colors, N)
cmap = cmap(np.arange(N))
colors = cmap[::-1,:].tolist()
if isinstance(colors, list):
if len(colors) == N:
colors = colors[::-1]
else:
Activate Windows
Go to Settings to activate Windows.
Search for anything
ENG US
4:09 PM
1/11/2021

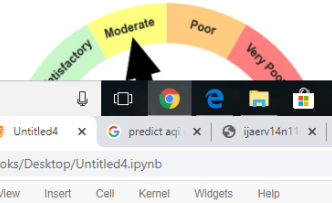
```
    ax.axis('equal')
    plt.tight_layout()
    if fname:
        fig.savefig(fname, dpi=200)
```

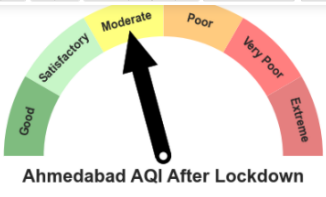
In [68]:
```
display("Ahmedabad's AQI levels")

gauge(labels=['Good','Satisfactory','Moderate','Poor','Very Poor','Extreme'], \
      colors=['#007A00','#90EE90','#ffff00','#FF9900','#ff0000','#CC0000'], arrow=5, title='Ahmedabad AQI before Lockdown')

gauge(labels=['Good','Satisfactory','Moderate','Poor','Very Poor','Extreme'], \
      colors=['#007A00','#90EE90','#ffff00','#FF9900','#ff0000','#CC0000'], arrow=3, title='Ahmedabad AQI After Lockdown')
```

"Ahmedabad's AQI levels"



Ahmedabad AQI before Lockdown

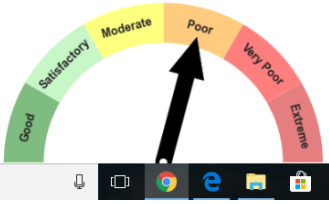




Ahmedabad AQI After Lockdown

In [69]:
```
display("Delhi's AQI levels")

gauge(labels=['Good','Satisfactory','Moderate','Poor','Very Poor','Extreme'], \
      colors=['#007A00','#90EE90','#ffff00','#FF9900','#ff0000','#CC0000'], arrow=4, title="Delhi's AQI before Lockdown")

gauge(labels=['Good','Satisfactory','Moderate','Poor','Very Poor','Extreme'], \
      colors=['#007A00','#90EE90','#ffff00','#FF9900','#ff0000','#CC0000'], arrow=3, title="Delhi's AQI after Lockdown")
```

"Delhi's AQI levels"



"Mumbai's AQI levels"

Mumbai's AQI before Lockdown

Mumbai's AQI after Lockdown

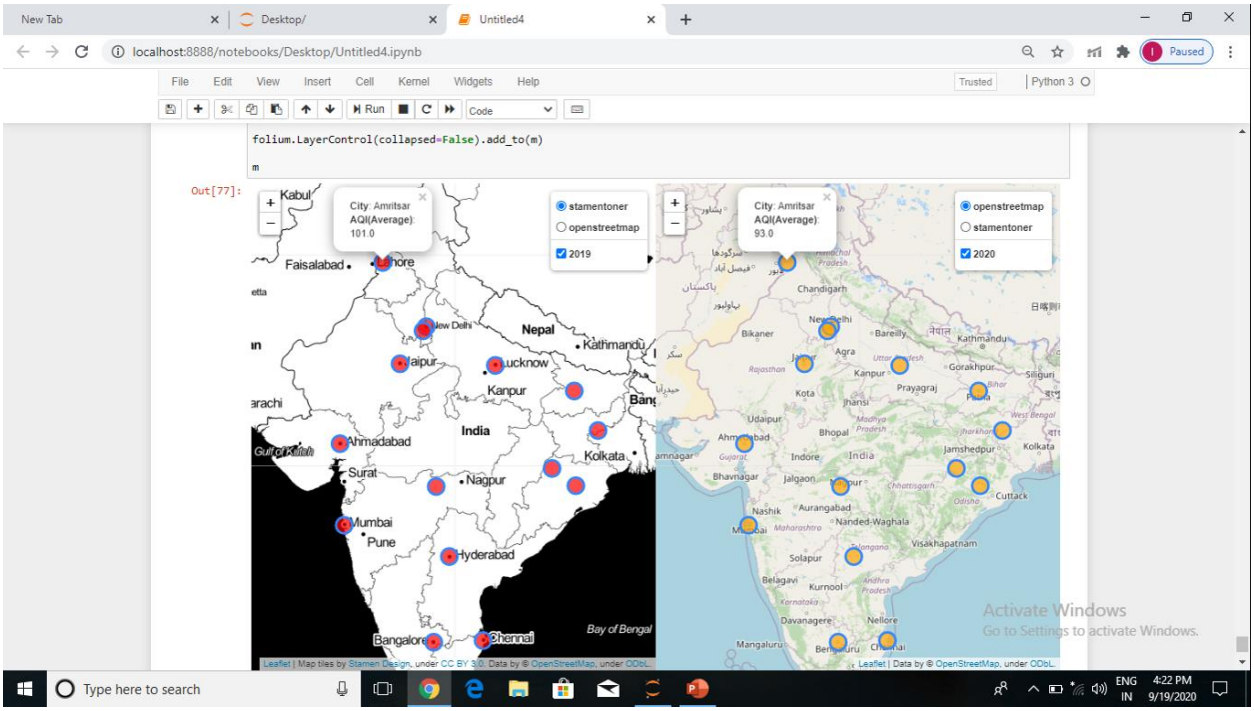
Mumbai's AQI after Lockdown

GEOLOGICAL ANALYSIS USING DUAL MAPS

In [75]:
```
df11 = pollutants_filtered_2019[['City','AQI']]
df22 = pollutants_filtered_2020[['City','AQI']]

df_2019_coord = df11.groupby('City')['AQI'].mean().to_frame().reset_index()
df_2020_coord = df22.groupby('City')['AQI'].mean().to_frame().reset_index()

df_2019_AQI = pd.merge(df_2019_coord,cities_db,on='City')
df_2019_AQI['AQI'] = df_2019_AQI['AQI'].round(0)
df_2020_AQI = pd.merge(df_2020_coord,cities_db,on='City')
df_2020_AQI['AQI'] = df_2020_AQI['AQI'].round(0)
```

In [77]:
```
m = plugins.DualMap(location=(22.9734, 78.6569), tiles=None, zoom_start=5)

folium.TileLayer('Stamen Toner').add_to(m)
folium.TileLayer('openstreetmap').add_to(m)


fg_1 = folium.FeatureGroup(name='2019').add_to(m.m1)
fg_2 = folium.FeatureGroup(name='2020').add_to(m.m2)



for lat, lon, value, name in zip(df_2019_AQI['Lat'], df_2019_AQI['Long'], df_2019_AQI['AQI'], df_2019_AQI['City']):
    folium.CircleMarker([lat, lon],
                        radius=10,
                        icon=folium.Icon(color='red'),
                        popup = ('<strong>City</strong>: ' + str(name).capitalize() + '<br>'
                                 '<strong>AQI(Average)</strong>: ' + str(value) + '<br>'),
                        fill_color='red',
                        fill_opacity=0.7 ).add_to(fg_1)
```

Activate Windows
Go to Settings to activate Windows.


New Tab
Desktop/
Untitled4
localhost:8888/notebooks/Desktop/Untitled4.ipynb
Paused
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
folium.LayerControl(collapsed=False).add_to(m)
m
Out[77]:
City: Amritsar
AQI(Average):
101.0
stamentoner
openstreetmap
2019
Kabul
Faisalabad
Lahore
New Delhi
Nepal
Kathmandu
Jaipur
Lucknow
Kanpur
India
Ahmadabad
Surat
Nagpur
Kolkata
Mumbai
Pune
Hyderabad
Bangalore
Chennai
Bay of Bengal
Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by © OpenStreetMap, under ODbL.
City: Amritsar
AQI(Average):
93.0
openstreetmap
stamentoner
2020
Chandigarh
Bikaner
Bareilly
Agra
Kanpur
Gorakhpur
Kota
Prayagraj
Udaipur
Bhopal
Indore
India
Jamshedpur
Kolkata
Bhavnagar
Jalgaon
Cuttack
Nashik
Aurangabad
Nanded-Waghala
Visakhapatnam
Solapur
Belagavi
Kurnool
Davanagere
Nellore
Mangaluru
Leaflet | Data by © OpenStreetMap, under ODbL.
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
IN
4:22 PM
9/19/2020