JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

B.TECH V SEMESTER

REPORT FOR

Open Source Software Lab Project

TITLE OF PROJECT

AirQ

Supervision of:

Dr. Alka Singhal

Assistant Professor (Senior Grade)

Department of Computer &

Science Engineering

JIIT, Sector 62, Noida

Submitted by:

Enrollment No.	**Name**
21103093	**Akash Sharma**
21103105	**Mukund Sarda**
21103096	**Karanjot Singh**

OSS LAB REPORT

PROJECT SYNOPSIS

Introduction

Abstract of the Project:

The provided Python script is part of a comprehensive air quality monitoring and prediction project. It begins by loading air quality data from a CSV file, performing data preprocessing to impute missing values and engineer new features. The script offers various data visualization tools for analyzing air quality parameters, trends, and AQI values in specific cities. It also utilizes seasonal decomposition and SARIMA modeling for time series analysis, forecasting AQI levels, and calculating performance metrics. Additionally, the code conducts ANOVA tests to compare AQI levels across different cities. In summary, this script combines data analysis, time series modeling, and statistical testing to provide valuable insights into air quality variations and trends among different urban areas.

Objective:

The main objective of this project is to python script serves the purpose of comprehensive air quality monitoring and prediction. Its goals encompass data preprocessing for missing value handling and feature engineering, visualizing air quality parameters and trends, conducting time series analysis with seasonal decomposition and SARIMA modeling, and assessing AQI levels among different cities using ANOVA tests.

Dataset Selection:

Dataset contains air quality data and AQI (Air Quality Index) at hourly and daily level of various stations across multiple cities in India.

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

Pre-Processing:

Pre-processing involved several steps to prepare the dataset for modeling:

Data Preprocessing

Handling Missing Values: The code begins by loading a dataset ('city_day.csv') using Pandas and then performs imputation of missing values using the Iterative Imputer from scikit-learn. The missing values for various air quality parameters (e.g., PM2.5, NO2, CO) are imputed using the MICE (Multivariate Imputation by Chained Equations) algorithm.

Feature Engineering: The code contains functions for feature engineering. It creates new columns like 'BTX' (sum of Benzene, Toluene, and Xylene) and 'Particulate_Matter' (sum of PM2.5 and PM10) to derive meaningful features from the original dataset.

Data Transformation: The code normalizes some columns, such as 'Particulate_Matter', 'NO2', 'CO', 'SO2', 'O3', and 'BTX', by scaling their values between 0 and 1. This transformation can help in better modeling and analysis.

AQI Bucketing: The code categorizes air quality into different AQI buckets, such as 'Good,' 'Satisfactory,' 'Moderate,' 'Poor,' 'Very Poor,' and 'Severe,' based on the AQI value.

Data Visualization: The code provides functions to visualize pollutant levels and trends, which can assist in understanding and interpreting the air quality data.

Time Series Analysis: Time series analysis is performed using SARIMA (Seasonal AutoRegressive Integrated Moving Average) to make predictions and evaluate the model's performance.

Statistical Analysis: The code conducts statistical analysis, such as ANOVA tests, to compare the AQI values of different cities and determine if there are significant differences among them.

Data Export: Finally, the code exports the preprocessed data to 'newCityData.csv' for further analysis.

Overall, the code covers data cleaning, feature engineering, normalization, visualization, time series analysis, and statistical tests to prepare and analyze air quality data.

Implementation:

Three regression models were implemented, and their performances were evaluated:

1. Linear Regression: Rectifying missing values.
2. Random Forest Regressor: A more complex model which can capture non-linear relationships and interactions between features.
3. Ridge Regression: This model was used to analyse the impact of regularization on the prediction accuracy .

Code & Working:

```
import csv
```

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime

import warnings

import seaborn as sns

# from plotly.subplots import make_subplots

# import plotly.graph_objects as go

warnings.filterwarnings("ignore")

warnings.filterwarnings("ignore")

city_day = pd.read_csv('city_day.csv')

del city_day['Unnamed: 0']

city_day

city_day.isnull().sum()

from sklearn.experimental import enable_iterative_imputer

from sklearn.impute import IterativeImputer

city_day = city_day.copy(deep=True)



mice_imputer = IterativeImputer()

city_day['PM2.5'] = mice_imputer.fit_transform(city_day[['PM2.5']])

city_day['PM10'] = mice_imputer.fit_transform(city_day[['PM10']])
```

```
city_day['NO'] = mice_imputer.fit_transform(city_day[['NO']])

city_day['NOx'] = mice_imputer.fit_transform(city_day[['NOx']])

city_day['NH3'] = mice_imputer.fit_transform(city_day[['NH3']])

city_day['CO'] = mice_imputer.fit_transform(city_day[['CO']])

city_day['SO2'] = mice_imputer.fit_transform(city_day[['SO2']])

city_day['O3'] = mice_imputer.fit_transform(city_day[['O3']])

city_day['Benzene'] = mice_imputer.fit_transform(city_day[['Benzene']])

city_day['Toluene'] = mice_imputer.fit_transform(city_day[['Toluene']])

city_day['Xylene'] = mice_imputer.fit_transform(city_day[['Xylene']])

city_day['AQI'] = mice_imputer.fit_transform(city_day[['AQI']])

city_day['NO2'] = mice_imputer.fit_transform(city_day[['NO2']])

city_day.isnull().sum()

city_day['AQI_Bucket'] = np.where(

  (city_day['AQI'] <=50) & (city_day['AQI'] >=0) ,'Good' , city_day['AQI_Bucket']

  )

city_day['AQI_Bucket'] = np.where(

  (city_day['AQI'] <=100) & (city_day['AQI'] >=51) ,'Satisfactory' , city_day['AQI_Bucket']

  )

city_day['AQI_Bucket'] = np.where(

  (city_day['AQI'] <=200) & (city_day['AQI'] >=101) ,'Moderate' , city_day['AQI_Bucket']

  )
```

```
city_day['AQI_Bucket'] = np.where(
   (city_day['AQI'] <=300) & (city_day['AQI'] >=201) ,'Poor' , city_day['AQI_Bucket']
   )
city_day['AQI_Bucket'] = np.where(
   (city_day['AQI'] <=400) & (city_day['AQI'] >=301) ,'Very Poor' , city_day['AQI_Bucket']
   )
city_day['AQI_Bucket'] = np.where(
   (city_day['AQI'] <=500) & (city_day['AQI'] >=401) ,'Severe' , city_day['AQI_Bucket']
   )
city_day
city_day.to_csv('city_day.csv')
def mergeColumns(data):
    data['Date'] = pd.to_datetime(data['Date'])
    data['BTX'] = data['Benzene'] + data['Toluene'] + data['Xylene']
    data.drop(['Benzene','Toluene','Xylene'], axis=1)
    data['Particulate_Matter'] = data['PM2.5'] + data['PM10']
    return data


def subsetColumns(data):
    pollutants = ['Particulate_Matter', 'NO2', 'CO','SO2', 'O3', 'BTX']
    columns =  ['Date', 'City', 'AQI', 'AQI_Bucket'] + pollutants
```

```
    data = data[columns]

    return data, pollutants

def handleMissingValues(data):

   # missing_values = getMissingValues(data)

    newCityData = mergeColumns(data)

    newCityData, pollutants = subsetColumns(newCityData)

    return newCityData, pollutants



newCityData, newColumns = handleMissingValues(city_day)

newCityData

newCityData.isnull().sum()

min_Particulate_Matter = newCityData['Particulate_Matter'].min()

max_Particulate_Matter = newCityData['Particulate_Matter'].max()

newCityData['Particulate_Matter_new'] = [(x-min_Particulate_Matter)/(max_Particulate_Matter-
min_Particulate_Matter)

                                         for x in newCityData['Particulate_Matter']]



min_NO2= newCityData['NO2'].min()

max_NO2 = newCityData['NO2'].max()

newCityData['NO2_new'] = [(x-min_NO2)/(max_NO2-min_NO2) for x in newCityData['NO2']]



min_CO= newCityData['CO'].min()
```

```
max_CO = newCityData['CO'].max()

newCityData['CO_new'] = [(x-min_CO)/(max_CO-min_CO) for x in newCityData['CO']]



min_SO2= newCityData['SO2'].min()

max_SO2 = newCityData['SO2'].max()

newCityData['SO2_new'] = [(x-min_SO2)/(max_SO2-min_SO2) for x in newCityData['SO2']]



min_O3= newCityData['O3'].min()

max_O3 = newCityData['O3'].max()

newCityData['O3_new'] = [(x-min_O3)/(max_O3-min_O3) for x in newCityData['O3']]



min_BTX = newCityData['BTX'].min()

max_BTX = newCityData['BTX'].max()

newCityData['BTX_new'] = [(x-min_BTX)/(max_BTX-min_BTX) for x in newCityData['BTX']]

newCityData

newCityData= newCityData[['City','Date','AQI','AQI_Bucket',
'Particulate_Matter_new','NO2_new','CO_new','SO2_new','O3_new',

                          'BTX_new']]

newCityData.to_csv('newCityData.csv')

def visualisepollutants(udata, column):

    data = udata.copy()

    data.set_index('Date',inplace=True)
```

```
    axes = data[column].plot(marker='.', alpha=0.5, linestyle='None', figsize=(16, 15), subplots=True)
    for ax in axes:
        ax.set_xlabel('Years')
        ax.set_ylabel('ug/m3')
visualisepollutants(nCityData, pollutant)
def trend_plot(nCityData, value):
    data = nCityData.copy()
    data['Year'] = [d.year for d in data.Date]
    data['Month'] = [d.strftime('%b') for d in data.Date]
    years = data['Year'].unique()
    fig, axes = plt.subplots(1, 2, figsize=(12,3), dpi= 80)
    sns.boxplot(x='Year', y=value, data=data, ax=axes[0])
    sns.lineplot(x='Month', y=value, data=data.loc[~data.Year.isin([2015, 2020]), :])

    axes[0].set_title('Year-wise Plot i.e. the trend', fontsize=18);
    axes[1].set_title('Month-wise Plot i.e. the seasonality', fontsize=18)
    plt.show()
value='Particulate_Matter_new'
trend_plot(nCityData,value)
value='NO2_new'
```

```
trend_plot(nCityData,value)

def visualiseAQI(udata, column):

    data = udata.copy()

    data.set_index('Date',inplace=True)



    axes = data[column].plot(marker='.', alpha=0.5, linestyle='None', figsize=(16, 3), subplots=True)

    for ax in axes:

        ax.set_xlabel('Years')

        ax.set_ylabel('AQI')

visualiseAQI(nCityData, ['AQI'])

from pandas import DataFrame

from pandas import concat

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):

    n_vars = 1 if type(data) is list else data.shape[1]

    df = DataFrame(data)

    cols, names = list(), list()

    # input sequence (t-n, ... t-1)

    for i in range(n_in, 0, -1):

        cols.append(df.shift(i))

        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

    # forecast sequence (t, t+1, ... t+n)
```

```
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
newCityData = pd.read_csv('newCityData.csv', header=0, index_col=0)
values = newCityData.values
values[:]
cities = ['Mumbai','Shillong','Lucknow','Delhi','Visakhapatnam','Patna','Bhopal']
somecityday = newCityData[newCityData['Date'] >= '2015-01-01']
AQI = somecityday[somecityday.City.isin(cities)][['Date','City','AQI','AQI_Bucket']]
AQI_pivot = AQI.pivot(index='Date', columns='City', values='AQI')
def getColorBar(city):
```

```
    col = []
    for val in AQI_pivot[city]:
        if val < 50:
            col.append('royalblue')
        elif val > 50 and val < 101:
            col.append('lightskyblue') #cornflowerblue
        elif val > 100 and val < 201:
            col.append('lightsteelblue')
        elif val > 200 and val < 301:
            col.append('peachpuff')
        elif val > 300 and val < 401:
            col.append('lightcoral')
        elif val> 400:
            col.append('firebrick')
        else:
            col.append('white')
    return col


ah = getColorBar('Mumbai')
de = getColorBar('Shillong')
```

```
mu = getColorBar('Lucknow')

ko = getColorBar('Delhi')

hy = getColorBar('Visakhapatnam')

ch = getColorBar('Patna')

bp=getColorBar('Bhopal')




colors = {'Good':'royalblue', 'Satisfactory':'lightskyblue', 'Moderate':'lightsteelblue', 'Poor':'peachpuff', 'Very Poor':'lightcoral', 'Severe':'firebrick'}

labels = list(colors.keys())

handles = [plt.Rectangle((0,0),1,1, color=colors[label]) for label in labels]



f, ((ax1, ax2, ax3, ax4, ax5, ax6, ax7)) = plt.subplots(7, 1, sharex='col', sharey='row', figsize=(15,18))

ax1.bar(AQI_pivot.index, AQI_pivot['Mumbai'], color = ah, width = 0.75)

ax2.bar(AQI_pivot.index, AQI_pivot['Shillong'], color = de, width = 0.75)

ax3.bar(AQI_pivot.index, AQI_pivot['Lucknow'], color = mu, width = 0.75)

ax4.bar(AQI_pivot.index, AQI_pivot['Delhi'], color = ko, width = 0.75)

ax5.bar(AQI_pivot.index, AQI_pivot['Visakhapatnam'], color = hy, width = 0.75)

ax6.bar(AQI_pivot.index, AQI_pivot['Patna'], color = ch, width = 0.75)

ax7.bar(AQI_pivot.index, AQI_pivot['Bhopal'], color = bp, width = 0.75)



ax1.legend(handles, labels, loc='upper left')
```

```
ax2.legend(handles, labels, loc='upper left')
ax3.legend(handles, labels, loc='upper left')
ax4.legend(handles, labels, loc='upper left')
ax5.legend(handles, labels, loc='upper left')
ax6.legend(handles, labels, loc='upper left')
ax7.legend(handles, labels, loc='upper left')

ax1.title.set_text('Mumbai')
ax2.title.set_text('Shillong')
ax3.title.set_text('Lucknow')
ax4.title.set_text('Delhi')
ax5.title.set_text('Visakhapatnam')
ax6.title.set_text('Patna')
ax7.title.set_text('Bhopal')


ax1.set_ylabel('AQI')
ax2.set_ylabel('AQI')
ax3.set_ylabel('AQI')
ax4.set_ylabel('AQI')
ax5.set_ylabel('AQI')
```

```
ax6.set_ylabel('AQI')

ax7.set_ylabel('AQI')



AQI_beforeLockdown = AQI_pivot['2015-01-01':'2020-03-25']

AQI_afterLockdown = AQI_pivot['2020-03-26':'2020-07-01']

limits = [50, 100, 200, 300, 400, 510]

#palette = sns.light_palette("Spectral", len(limits), reverse = True)

palette = sns.color_palette("coolwarm", len(limits))

for city in cities:

    aqi_before = AQI_beforeLockdown[city].mean()

    aqi_after = AQI_afterLockdown[city].mean()

    fig, (ax1, ax2) = plt.subplots(1,2,figsize=(27, 1.5))

    ax1.set_yticks([1])

    ax1.set_yticklabels([city])

    ax1.spines['bottom'].set_visible(False)

    ax1.spines['top'].set_visible(False)

    ax1.spines['right'].set_visible(False)

    ax1.spines['left'].set_visible(False)



    prev_limit = 0

    for idx, lim in enumerate(limits):
```

```
    ax1.barh([1], lim-prev_limit, left=prev_limit, height=15, color=palette[idx])
    prev_limit = lim

ax1.barh([1], aqi_before, color='black', height=5)

# after lockdown
ax2.set_yticks([1])
ax2.set_yticklabels([city])
ax2.spines['bottom'].set_visible(False)
ax2.spines['top'].set_visible(False)
ax2.spines['right'].set_visible(False)
ax2.spines['left'].set_visible(False)

prev_limit = 0
for idx, lim in enumerate(limits):
    ax2.barh([1], lim-prev_limit, left=prev_limit, height=15, color=palette[idx])
    prev_limit = lim

ax2.barh([1], aqi_after, color='black', height=5)

ax1.set_title('Before Lockdown')
```

```
ax2.set_title('After Lockdown')


rects = ax1.patches

labels=["Good", "Satisfactory", "Moderate", "Poor", 'Very Poor', 'Severe']


for rect, label in zip(rects, labels):

    height = rect.get_height()

    ax1.text(

        rect.get_x() + rect.get_width() / 2,

        -height * .4,

        label,

        ha='center',

        va='bottom',

        color='black')

    ax2.text(

        rect.get_x() + rect.get_width() / 2,

        -height * .4,

        label,

        ha='center',

        va='bottom',

        color='black')
```

```
Delhi_data = newCityData[newCityData['City']=='Delhi']

Delhi_data.set_index('Date',inplace=True, drop = False)

Delhi_data

val = 'AQI'

date_range = np.arange('2015-01-01', '2020-07-02', dtype='datetime64[D]')

values = np.random.rand(len(date_range))

final_data = pd.DataFrame(index=date_range, columns=[val], data=values)



# Assuming Delhi_data has 'Date' as the index

Delhi_data['Date'] = pd.to_datetime(Delhi_data['Date'])

Delhi_data = Delhi_data.set_index('Date')



# Reindex final_data to match Delhi_data's index

final_data = final_data.reindex(Delhi_data.index)



# Assign values

final_data[val] = Delhi_data[val]



print(final_data[val])
```

```
final_data=final_data.astype('float64')

final_data[val] = final_data[val].fillna(final_data[val].mean(axis=0))

seasonal_data = final_data

seasonal_data = seasonal_data.resample(rule='MS').mean()

seasonal_data

from statsmodels.tsa.seasonal import seasonal_decompose

Delhi_AQI = seasonal_data[val]

result = seasonal_decompose(Delhi_AQI, model='multiplicative')

result.plot();

import pmdarima as pm

from statsmodels.tsa.statespace.sarimax import SARIMAX

from pmdarima import auto_arima;

auto_arima(y=Delhi_AQI,start_p=0,start_P=0,start_q=0,start_Q=0,seasonal=True, m=12)

train = Delhi_AQI[:41] #from 2015-2018

test = Delhi_AQI[42:54]# july 2018-june 2019

test

model=SARIMAX(train,order=(1,0,0),seasonal_order=(1,0,1,12),)

results=model.fit()

results.summary()

predictions = results.predict(start=42, end=53, typ='levels').rename('Predictions')

predictions.plot(legend=True)
```

```
test.plot(legend=True,title="Delhi Prediction data");

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_variance_score,
max_error,mean_poisson_deviance,mean_gamma_deviance

import math

RMSE=np.sqrt(mean_squared_error(predictions,test))

print('Root Mean Squared Error: ', RMSE)

print('Mean AQI:',test.mean())

forecast_errors = [test[i]-predictions[i] for i in range(len(test))]

bias = sum(forecast_errors) * 1.0/len(test)

print('Bias: %f' % bias)

mse = mean_squared_error(test, predictions)

print('MSE: '+str(mse))

mae = mean_absolute_error(test, predictions)

print('MAE: '+str(mae))

rmse = math.sqrt(mean_squared_error(test, predictions))

print('RMSE: '+str(rmse))

mape = np.mean(np.abs(predictions - test)/np.abs(test))

print('MAPE: '+str(mape))

r2score=r2_score(test, predictions)

print('r2score: '+str(r2score))

explainedVariance_score=explained_variance_score(test, predictions)

print('explainedVariance_score: '+str(explainedVariance_score))
```

```
me=max_error(test, predictions)

print('me: '+str(me))

mpd=mean_poisson_deviance(test, predictions)

print('mpd: '+str(mpd))

mgd=mean_gamma_deviance(test, predictions)

print('mgd: '+str(mgd))

# Forming the model:

final_model = SARIMAX(train,order=(1,0,0),seasonal_order=(1,0,1,12))

results = final_model.fit()

#Obtaining predicted values:

predictions = results.predict(start=64, end=77, typ='levels').rename('Predictions')

#Plotting predicted values against the true values:

predictions.plot(legend=True)

Delhi_AQI.plot(legend=True,figsize=(12,8),grid=True,title="Delhi AQI");

df_anova = pd.read_csv('newCityData.csv')

df_anova = df_anova[['AQI','City']]

from scipy import stats

Citys = pd.unique(df_anova.City.values)

d_data = {city:df_anova['AQI'][df_anova.City == city] for city in Citys}



F, p = stats.f_oneway(d_data['Mumbai'], d_data['Shillong'], d_data['Lucknow'])
```

```
print("p-value for significance is: ", p)

if p<0.05:
    print("We reject the null hypothesis")
else:
    print("We accept the null hypothesis")
from scipy import stats
Citys = pd.unique(df_anova.City.values)
d_data = {city:df_anova['AQI'][df_anova.City == city] for city in Citys}

F, p = stats.f_oneway(d_data['Delhi'], d_data['Visakhapatnam'])

print("p-value for significance is: ", p)

if p<0.05:
    print("We reject the null hypothesis")
else:
    print("We accept the null hypothesis")
from scipy import stats
Citys = pd.unique(df_anova.City.values)
```

```
d_data = {city:df_anova['AQI'][df_anova.City == city] for city in Citys}

F, p = stats.f_oneway(d_data['Patna'], d_data['Bhopal'])

print("p-value for significance is: ", p)

if p<0.05:
    print("We reject the null hypothesis")
else:
    print("We accept the null hypothesis")
```

[2]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket
0	Ahmedabad	2015-01-01	67.450578	118.127103	0.92	18.22	17.15	23.483476	0.92	27.64	133.36	0.00000	0.020000	0.000000	166.463581	Moderate
1	Ahmedabad	2015-01-02	67.450578	118.127103	0.97	15.69	16.46	23.483476	0.97	24.55	34.06	3.68000	5.500000	3.770000	166.463581	Moderate
2	Ahmedabad	2015-01-03	67.450578	118.127103	17.40	19.30	29.70	23.483476	17.40	29.07	30.70	6.80000	16.400000	2.250000	166.463581	Moderate
3	Ahmedabad	2015-01-04	67.450578	118.127103	1.70	18.48	17.97	23.483476	1.70	18.59	36.08	4.43000	10.140000	1.000000	166.463581	Moderate
4	Ahmedabad	2015-01-05	67.450578	118.127103	22.10	21.42	37.76	23.483476	22.10	39.33	39.31	7.01000	18.890000	2.780000	166.463581	Moderate
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29526	Visakhapatnam	2020-06-27	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30	2.24000	12.070000	0.730000	41.000000	Good
29527	Visakhapatnam	2020-06-28	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14	0.74000	2.210000	0.380000	70.000000	Satisfactory
29528	Visakhapatnam	2020-06-29	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96	0.01000	0.010000	0.000000	68.000000	Satisfactory
29529	Visakhapatnam	2020-06-30	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30	0.00000	0.000000	0.000000	54.000000	Satisfactory
29530	Visakhapatnam	2020-07-01	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05	3.28084	8.700972	3.070128	50.000000	Good

29531 rows × 16 columns

[6]:

	City	Date	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	Benzene	Toluene	Xylene	AQI	AQI_Bucket
0	Ahmedabad	2015-01-01	67.450578	118.127103	0.92	18.22	17.15	23.483476	0.92	27.64	133.36	0.00000	0.020000	0.000000	166.463581	Moderate
1	Ahmedabad	2015-01-02	67.450578	118.127103	0.97	15.69	16.46	23.483476	0.97	24.55	34.06	3.68000	5.500000	3.770000	166.463581	Moderate
2	Ahmedabad	2015-01-03	67.450578	118.127103	17.40	19.30	29.70	23.483476	17.40	29.07	30.70	6.80000	16.400000	2.250000	166.463581	Moderate
3	Ahmedabad	2015-01-04	67.450578	118.127103	1.70	18.48	17.97	23.483476	1.70	18.59	36.08	4.43000	10.140000	1.000000	166.463581	Moderate
4	Ahmedabad	2015-01-05	67.450578	118.127103	22.10	21.42	37.76	23.483476	22.10	39.33	39.31	7.01000	18.890000	2.780000	166.463581	Moderate
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29526	Visakhapatnam	2020-06-27	15.020000	50.940000	7.68	25.06	19.54	12.470000	0.47	8.55	23.30	2.24000	12.070000	0.730000	41.000000	Good
29527	Visakhapatnam	2020-06-28	24.380000	74.090000	3.42	26.06	16.53	11.990000	0.52	12.72	30.14	0.74000	2.210000	0.380000	70.000000	Satisfactory
29528	Visakhapatnam	2020-06-29	22.910000	65.730000	3.45	29.53	18.33	10.710000	0.48	8.42	30.96	0.01000	0.010000	0.000000	68.000000	Satisfactory
29529	Visakhapatnam	2020-06-30	16.640000	49.970000	4.05	29.26	18.80	10.030000	0.52	9.84	28.30	0.00000	0.000000	0.000000	54.000000	Satisfactory
29530	Visakhapatnam	2020-07-01	15.000000	66.000000	0.40	26.85	14.05	5.200000	0.59	2.10	17.05	3.28084	8.700972	3.070128	50.000000	Good

29531 rows × 16 columns

```
#print(newcolumns)
```

[8]:

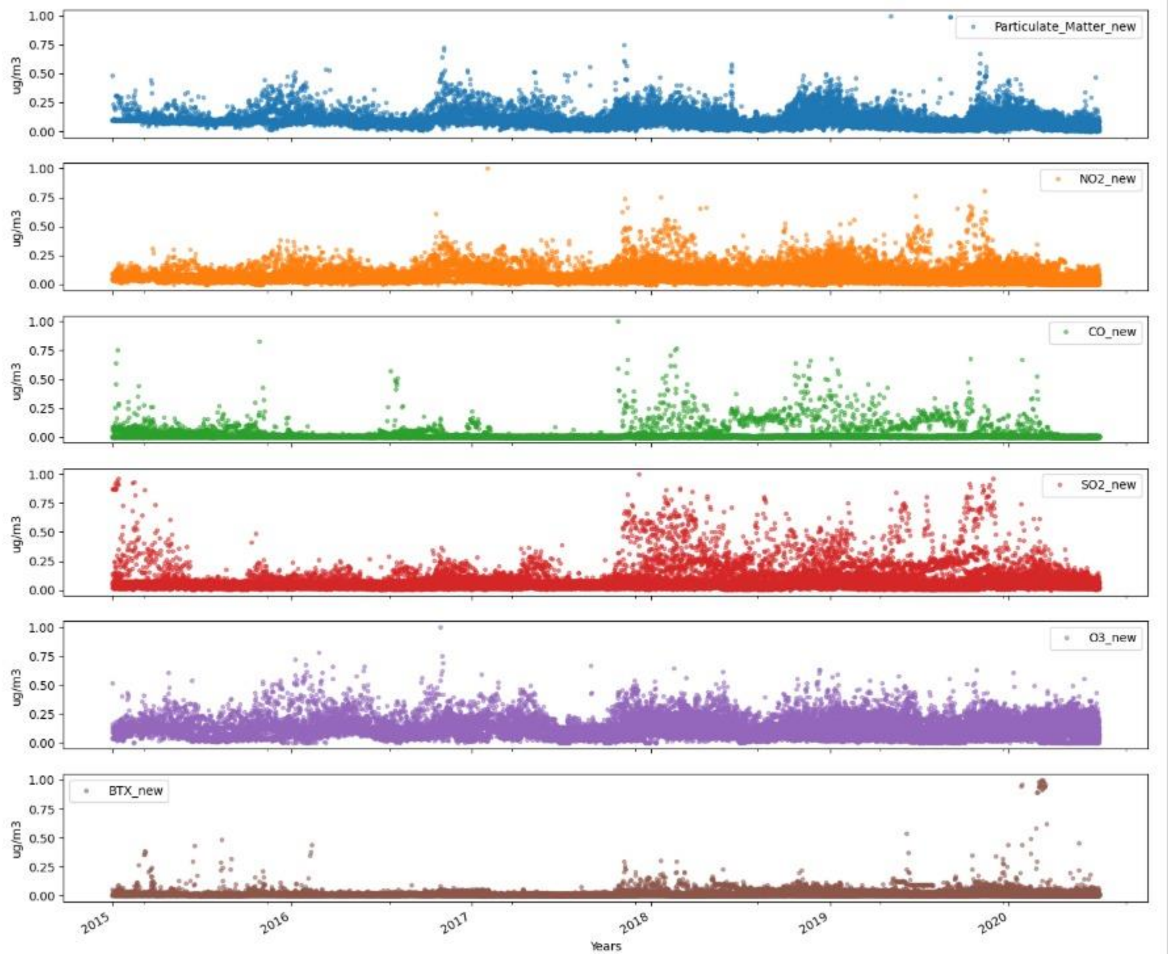
	Date	City	AQI	AQI_Bucket	Particulate_Matter	NO2	CO	SO2	O3	BTX
0	2015-01-01	Ahmedabad	166.463581	Moderate	185.577681	18.22	0.92	27.64	133.36	0.02000
1	2015-01-02	Ahmedabad	166.463581	Moderate	185.577681	15.69	0.97	24.55	34.06	12.95000
2	2015-01-03	Ahmedabad	166.463581	Moderate	185.577681	19.30	17.40	29.07	30.70	25.45000
3	2015-01-04	Ahmedabad	166.463581	Moderate	185.577681	18.48	1.70	18.59	36.08	15.57000
4	2015-01-05	Ahmedabad	166.463581	Moderate	185.577681	21.42	22.10	39.33	39.31	28.68000
...	...	...	...	...	...	...	...	...	...	...
29526	2020-06-27	Visakhapatnam	41.000000	Good	65.960000	25.06	0.47	8.55	23.30	15.04000
29527	2020-06-28	Visakhapatnam	70.000000	Satisfactory	98.470000	26.06	0.52	12.72	30.14	3.33000
29528	2020-06-29	Visakhapatnam	68.000000	Satisfactory	88.640000	29.53	0.48	8.42	30.96	0.02000
29529	2020-06-30	Visakhapatnam	54.000000	Satisfactory	66.610000	29.26	0.52	9.84	28.30	0.00000
29530	2020-07-01	Visakhapatnam	50.000000	Good	81.000000	26.85	0.59	2.10	17.05	15.05194

29531 rows × 10 columns

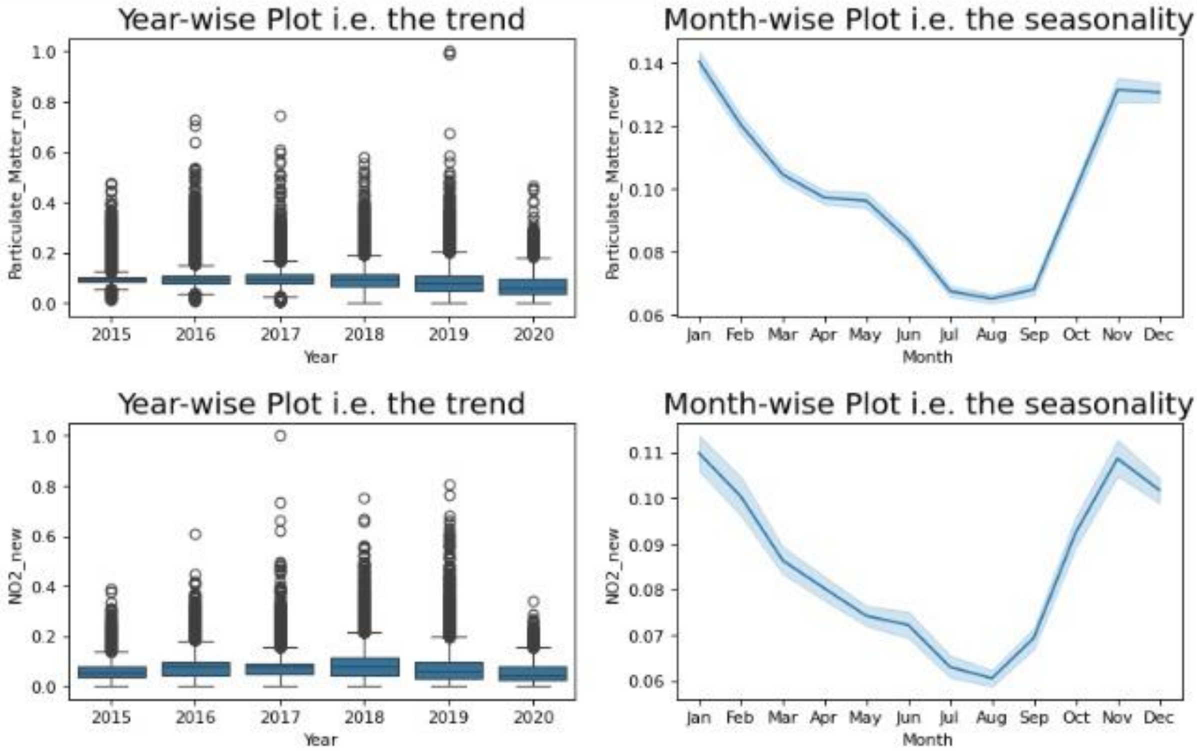
[10]:

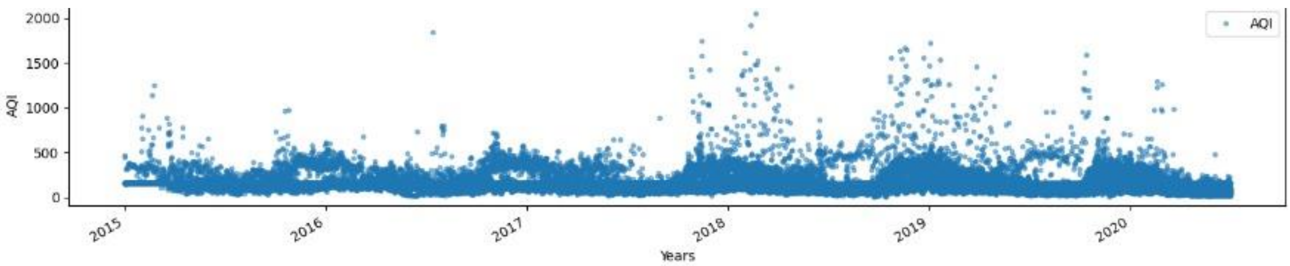
	Date	City	AQI	AQI_Bucket	Particulate_Matter	NO2	CO	SO2	O3	BTX	Particulate_Matter_new	NO2_new	CO_new	SO2_new
0	2015-01-01	Ahmedabad	166.463581	Moderate	185.577681	18.22	0.92	27.64	133.36	0.02000	0.096167	0.050276	0.005233	0.14253
1	2015-01-02	Ahmedabad	166.463581	Moderate	185.577681	15.69	0.97	24.55	34.06	12.95000	0.096167	0.043291	0.005517	0.12659
2	2015-01-03	Ahmedabad	166.463581	Moderate	185.577681	19.30	17.40	29.07	30.70	25.45000	0.096167	0.053258	0.098970	0.14991
3	2015-01-04	Ahmedabad	166.463581	Moderate	185.577681	18.48	1.70	18.59	36.08	15.57000	0.096167	0.050994	0.009670	0.09584
4	2015-01-05	Ahmedabad	166.463581	Moderate	185.577681	21.42	22.10	39.33	39.31	28.68000	0.096167	0.059111	0.125704	0.20283
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29526	2020-06-27	Visakhapatnam	41.000000	Good	65.960000	25.06	0.47	8.55	23.30	15.04000	0.033713	0.069161	0.002673	0.04405
29527	2020-06-28	Visakhapatnam	70.000000	Satisfactory	98.470000	26.06	0.52	12.72	30.14	3.33000	0.050687	0.071922	0.002958	0.06556
29528	2020-06-29	Visakhapatnam	68.000000	Satisfactory	88.640000	29.53	0.48	8.42	30.96	0.02000	0.045555	0.081502	0.002730	0.04338
29529	2020-06-30	Visakhapatnam	54.000000	Satisfactory	66.610000	29.26	0.52	9.84	28.30	0.00000	0.034052	0.080756	0.002958	0.05070
29530	2020-07-01	Visakhapatnam	50.000000	Good	81.000000	26.85	0.59	2.10	17.05	15.05194	0.041566	0.074103	0.003356	0.01078

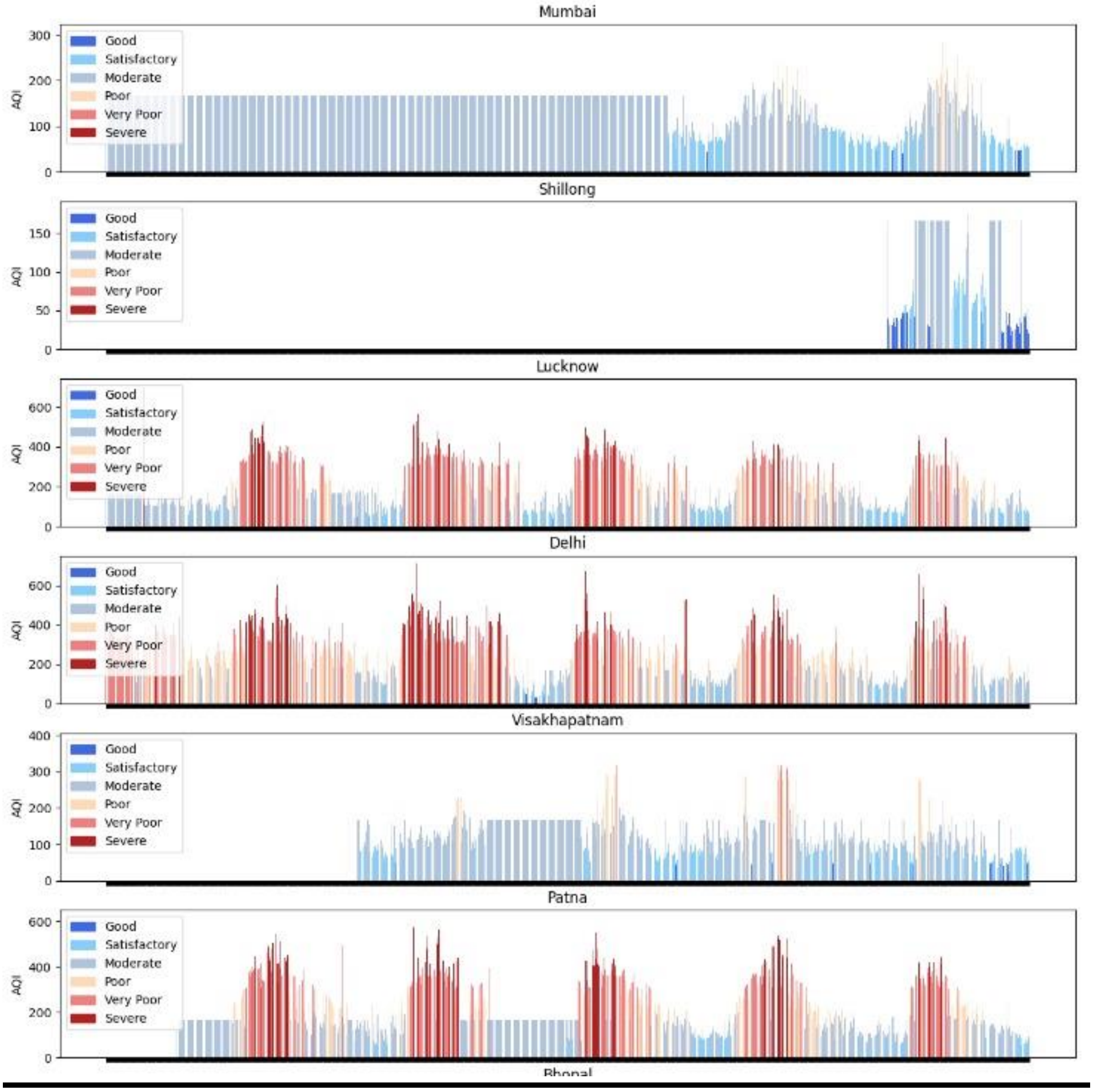
29531 rows × 16 columns

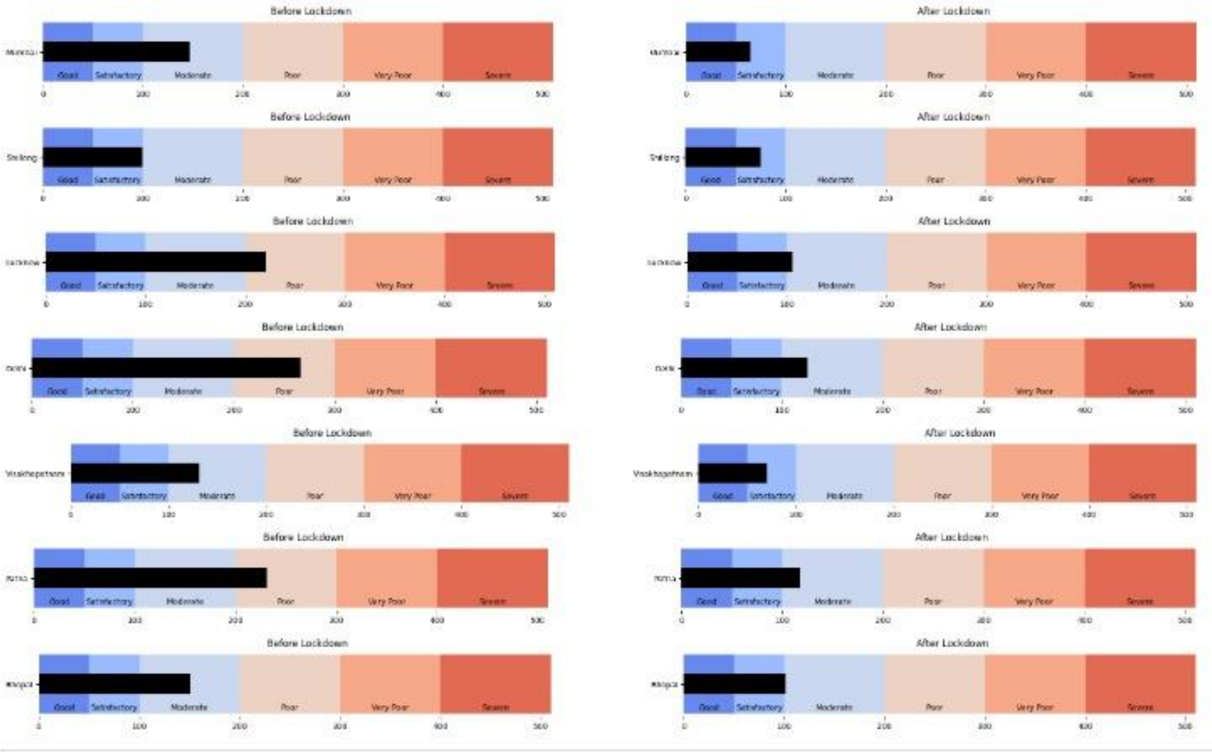


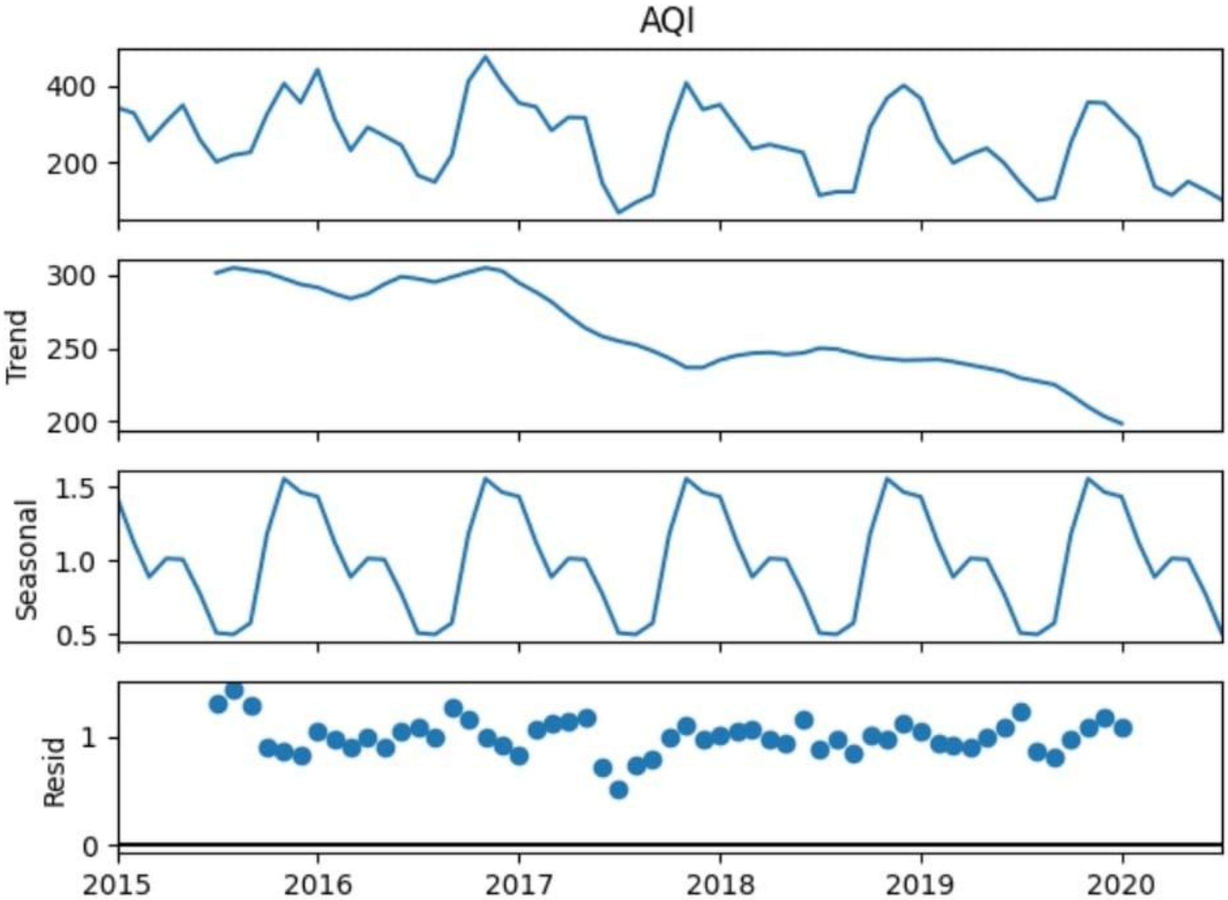
```
value='NO2_new'
trend_plot(nCityData,value)
```

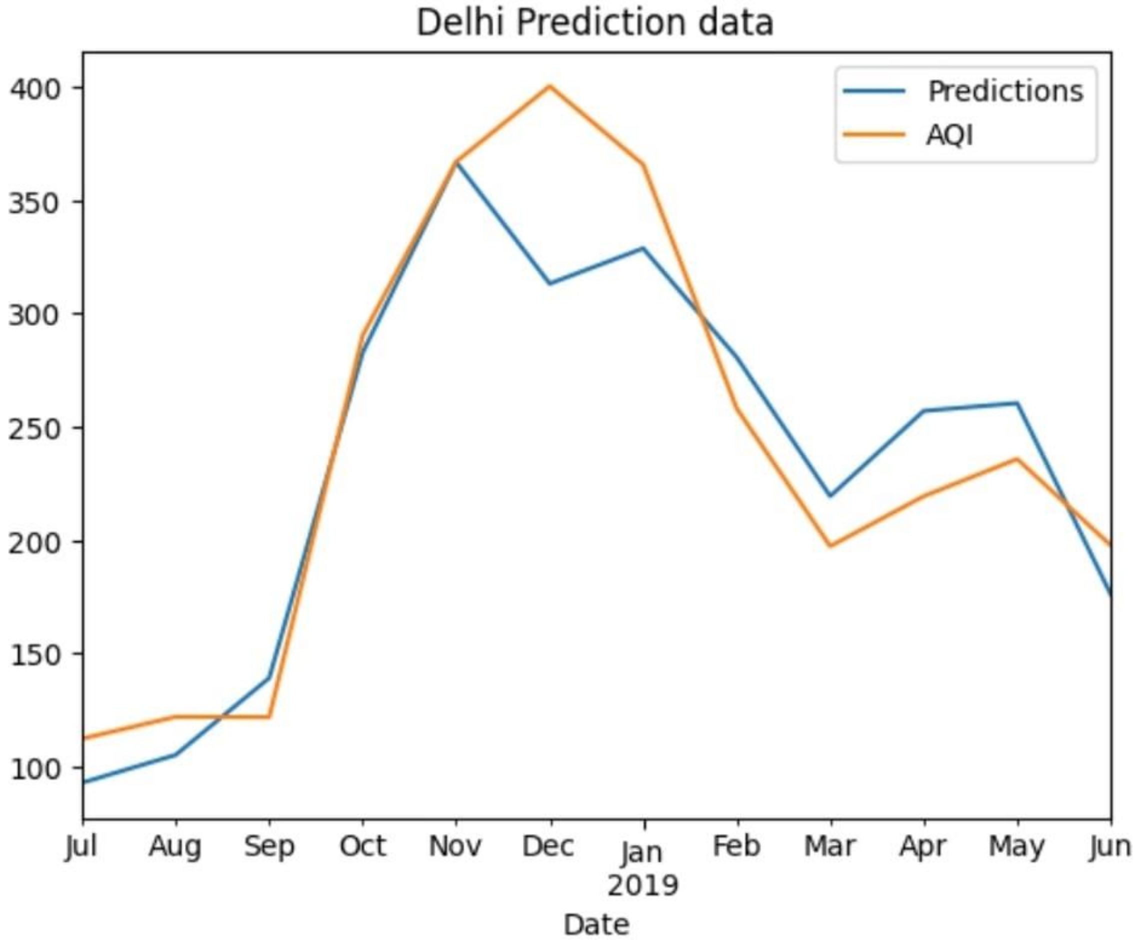


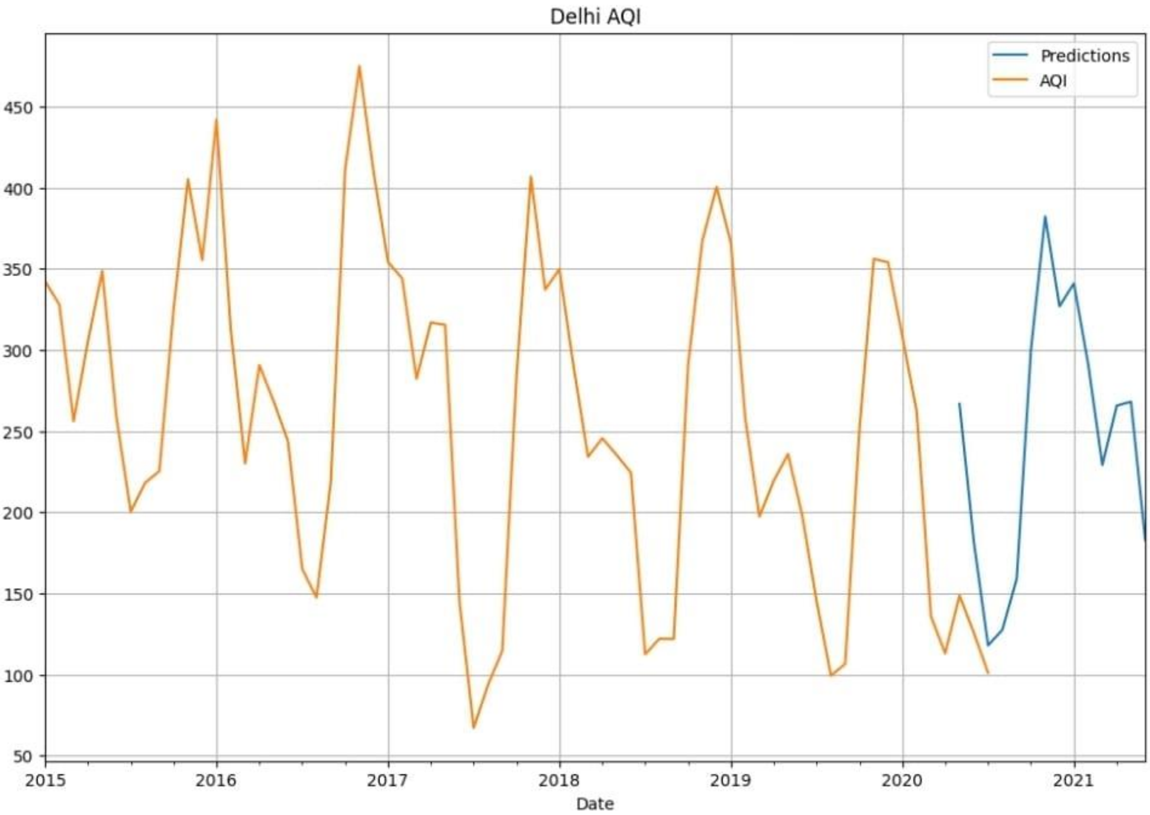



Mumbai
Shillong
Lucknow
Delhi
Visakhapatnam
Patna
AQI
Good
Satisfactory
Moderate
Poor
Very Poor
Severe


Before Lockdown
After Lockdown
Good
Satisfactory
Moderate
Poor
Very Poor
Severe
0
100
200
300
400
500


AQI
400
200
Trend
300
250
200
Seasonal
1.5
1.0
0.5
Resid
1
0
2015
2016
2017
2018
2019
2020


Delhi Prediction data
Predictions
AQI
400
350
300
250
200
150
100
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2019
Feb
Mar
Apr
May
Jun
Date


Delhi AQI
Predictions
AQI
450
400
350
300
250
200
150
100
50
2015
2016
2017
2018
2019
2020
2021
Date

References:

1. https://www.kaggle.com/datasets/rohanrao/air-quality-data-in-india
2. https://m.rp5.ru/Weather_archive_in_New_Delhi,_Safdarjung_(airport)
3. https://pandas.pydata.org/docs/
4. https://numpy.org/doc/1.26/
5. https://scikit-learn.org/0.21/documentation.html
6. https://colab.research.google.com/?utm_source=scs-index