In [82]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
%matplotlib Inline
```

In [2]:
```
data = 'E:\\SIMPLILEARN\\KAGGLE\\Air_Pollution\\data.csv'
with open(data) as f:
    print(f)

data = pd.read_csv(data, encoding='ISO-8859-1')
```

```
<_io.TextIOWrapper name='E:\\SIMPLILEARN\\KAGGLE\\Air_Pollution\\data.csv' mode=
'r' encoding='cp1252'>
```

In [3]:
```
# Extracting Tamil Nadu state data alone
df = data.query("state == 'Maharashtra'")
```

In [4]:
```
df.head(2)
```

Out[4]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm
197237	53.0	5/2/1987	Maharashtra	Pune	Maharashtra Pollution Control Board	NaN	NaN	10.4	NaN
197238	52.0	5/3/1987	Maharashtra	Pune	Maharashtra Pollution Control Board	Industrial	18.2	35.8	NaN

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 60384 entries, 197237 to 257620
Data columns (total 13 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   stn_code                     42976 non-null  object
 1   sampling_date                60384 non-null  object
 2   state                        60384 non-null  object
 3   location                     60384 non-null  object
 4   agency                       41695 non-null  object
 5   type                         60090 non-null  object
 6   so2                          58322 non-null  float64
 7   no2                          59167 non-null  float64
 8   rspm                         55439 non-null  float64
 9   spm                          22345 non-null  float64
 10  location_monitoring_station  57529 non-null  object
 11  pm2_5                        0 non-null      float64
```

```
 12  date                        60382 non-null  object
dtypes: float64(5), object(8)
```

In [6]:
```
df.isna().sum().sort_values()
```

Out[6]:
```
sampling_date                    0
state                            0
location                         0
date                             2
type                           294
no2                           1217
so2                           2062
location_monitoring_station   2855
rspm                          4945
stn_code                     17408
agency                       18689
spm                          38039
pm2_5                        60384
dtype: int64
```

In [7]:
```
df.tail()
```

Out[7]:

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm
257616	711.0	27-12-15	Maharashtra	Nagpur	Maharashtra State Pollution Control Board	Residential, Rural and other Areas	12.0	48.0	134.0
257617	711.0	28-12-15	Maharashtra	Nagpur	Maharashtra State Pollution Control Board	Residential, Rural and other Areas	15.0	63.0	77.0
257618	711.0	29-12-15	Maharashtra	Nagpur	Maharashtra State Pollution Control Board	Residential, Rural and other Areas	11.0	33.0	51.0
257619	711.0	30-12-15	Maharashtra	Nagpur	Maharashtra State Pollution Control Board	Residential, Rural and other Areas	12.0	38.0	46.0
257620	711.0	31-12-15	Maharashtra	Nagpur	Maharashtra State Pollution Control Board	Residential, Rural and other Areas	12.0	38.0	51.0

In [8]:
```
df.describe()
```

Out[8]:

	so2	no2	rspm	spm	pm2_5
count	58322.000000	59167.000000	55439.000000	22345.000000	0.0

	so2	no2	rspm	spm	pm2_5
mean	17.366863	32.115370	101.479608	205.255823	NaN
std	13.541155	19.062558	61.460782	130.053447	NaN
min	0.000000	1.000000	1.000000	4.000000	NaN
25%	9.000000	18.000000	59.000000	114.000000	NaN
50%	14.000000	29.000000	90.000000	180.000000	NaN
75%	22.000000	41.000000	128.000000	265.000000	NaN

In [9]:
```
df['type'].replace("Industrial Areas","Industrial",inplace=True)
df['type'].replace("Industrial Area","Industrial",inplace=True)
df['type'].replace("Residential and others","Residential",inplace=True)
df['type'].replace("Residential, Rural and other Areas","Residential",inplace=Tr
df['type'].replace("Sensitive Areas", "Sensitive Area", inplace=True)
```

In [10]:
```
df['so2'].fillna(df['so2'].mean(), inplace=True)
df['no2'].fillna(df['no2'].mean(), inplace=True)
```

In [11]:
```
df['rspm'].fillna(df['rspm'].median(),inplace = True)
df['spm'].fillna(df['spm'].median(),inplace = True)
```

In [12]:
```
df['type']=df['type'].fillna(df['type'].mode()[0])
```

In [13]:
```
df.isna().sum()
```

Out[13]:
```
stn_code                       17408
sampling_date                      0
state                              0
location                           0
agency                         18689
type                               0
so2                                0
no2                                0
rspm                               0
spm                                0
location_monitoring_station     2855
pm2_5                          60384
date                               2
dtype: int64
```

In [14]:
```
df = df.sort_values(by='date')
```

In [15]:
```
df['date'].ffill(inplace=True)
```

In [16]:
```
df['date'] = pd.to_datetime(df['date'],format='%Y-%m-%d')
```

Dropping Columns like location_monitoring_station, pm2_5, date, Agency and stn_code

In [17]:
```
df.drop(columns=['location_monitoring_station', 'pm2_5', 'sampling_date', 'agenc
```

In [18]:
```
df.head()
```

Out[18]:

	state	location	type	so2	no2	rspm	spm	date
197243	Maharashtra	Pune	Residential	8.100000	9.0	90.0	180.0	1987-01-07
197244	Maharashtra	Pune	Residential	0.700000	47.6	90.0	293.0	1987-01-12
197237	Maharashtra	Pune	Residential	17.366863	10.4	90.0	180.0	1987-02-05
197239	Maharashtra	Pune	Industrial	11.300000	16.3	90.0	180.0	1987-02-06
197242	Maharashtra	Pune	Industrial	9.700000	10.9	90.0	180.0	1987-02-07

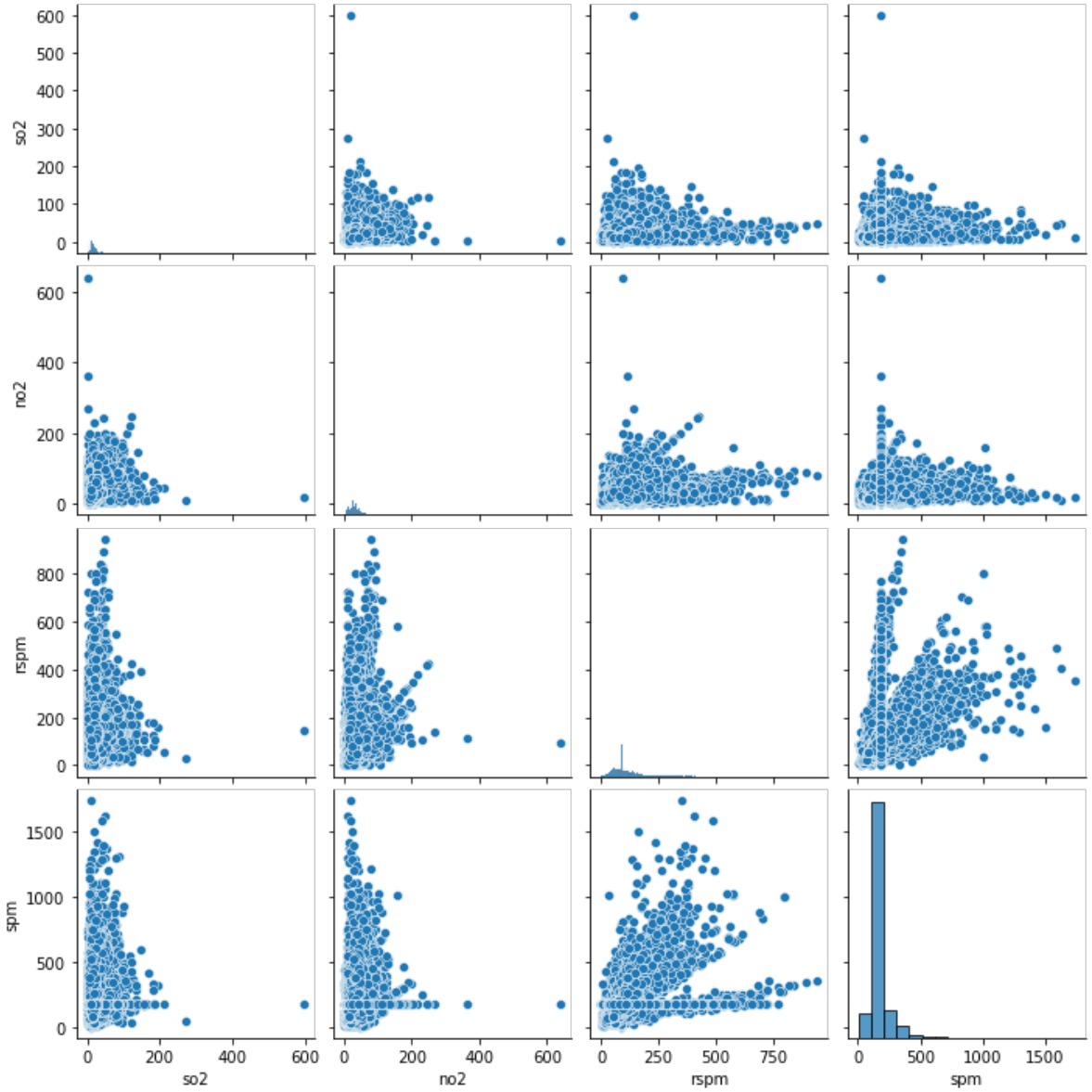
Exploratory Data Analysis

In [19]:
```
plt.figure(figsize=(10,10))
plt.plot(df['date'], df[['no2', 'so2']])
plt.xlabel("Year")
plt.ylabel("SO2 & NO2")
```

Out[19]:
```
Text(0, 0.5, 'SO2 & NO2')
```

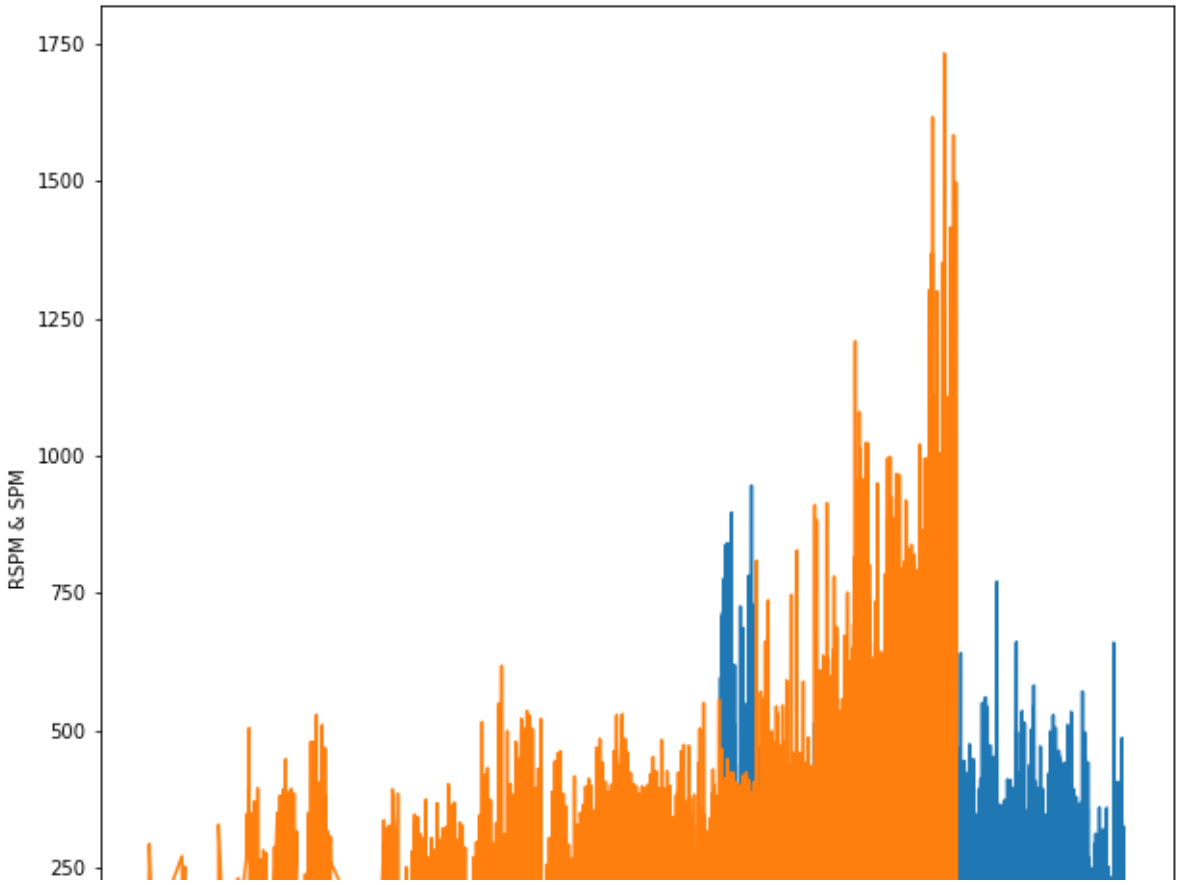
In [20]:
```
sns.pairplot(data=df)
```

Out[20]:
```
<seaborn.axisgrid.PairGrid at 0x1e0386dbe50>
```



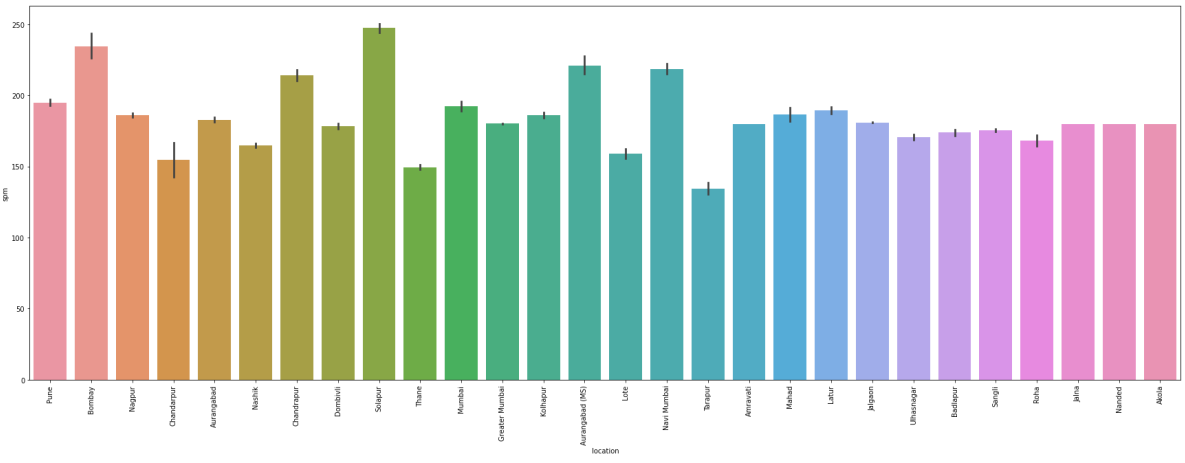
In [21]:
```
plt.figure(figsize=(10,10))
plt.plot(df['date'], df[['rspm', 'spm']])
plt.xlabel("Year")
plt.ylabel("RSPM & SPM")
```

Out[21]:
```
Text(0, 0.5, 'RSPM & SPM')
```



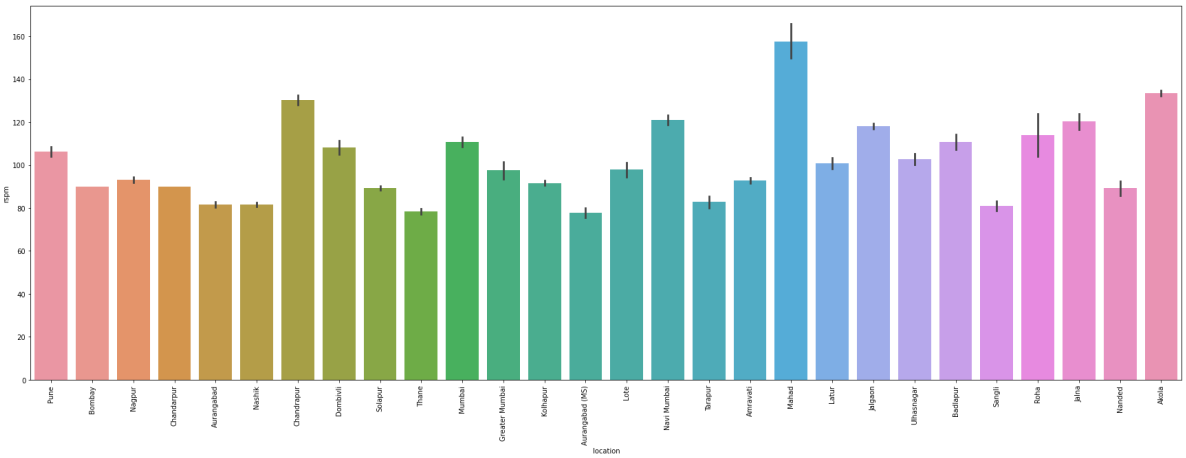
In [22]:

```
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='location',y='spm',data=df);
```

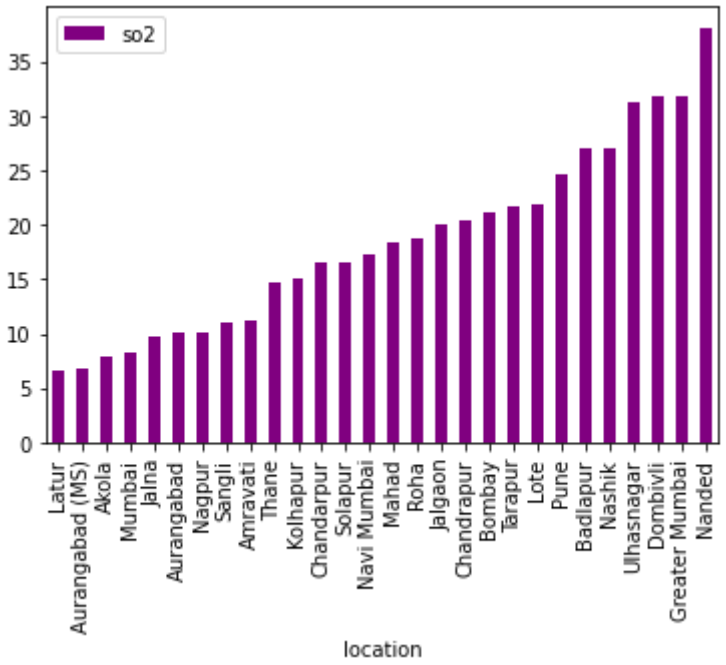


In [23]:

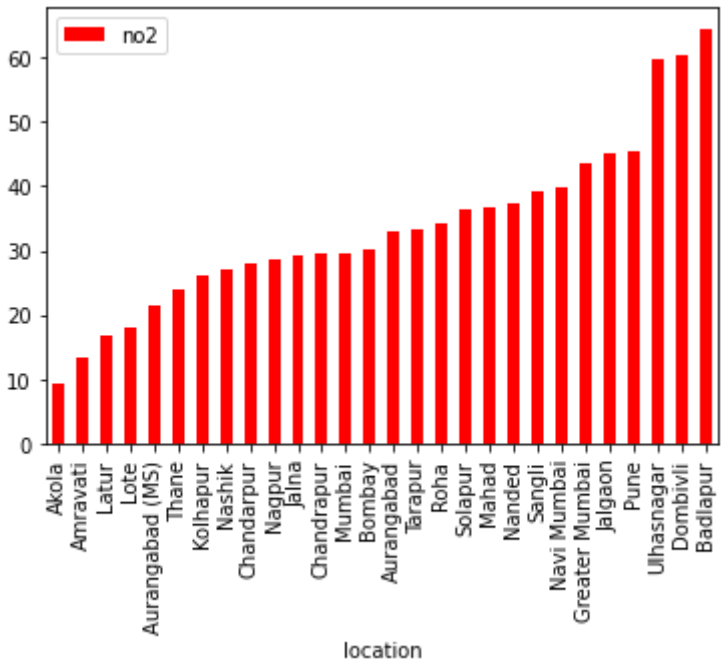
```
plt.figure(figsize=(30, 10))
plt.xticks(rotation=90)
sns.barplot(x='location',y='rspm',data=df);
```



In [24]:
```
df[['so2','location']].groupby(["location"]).mean().sort_values(by='so2').plot.b
plt.show()
```



In [25]:
```
df[['no2','location']].groupby(["location"]).mean().sort_values(by='no2').plot.b
plt.show()
```



In [26]:
```
loc = pd.pivot_table(df, values=['so2','no2','spm'],index='location')
loc
```

Out[26]:

location	no2	so2	spm
Akola	9.357824	7.937023	180.000000
Amravati	13.380021	11.138559	180.000000
Aurangabad	33.018726	10.016638	182.865231
Aurangabad (MS)	21.351419	6.815767	220.934524
Badlapur	64.451282	27.041370	173.935043
Bombay	30.061229	21.169740	234.602837
Chandarpur	27.834365	16.567540	154.779221
Chandrapur	29.483155	20.313778	214.248013
Dombivli	60.499191	31.839702	178.267231
Greater Mumbai	43.545930	31.843011	180.058055
Jalgaon	45.192121	20.052887	180.984350
Jalna	29.304250	9.686299	180.000000
Kolhapur	26.175916	15.091260	186.186164
Latur	16.856614	6.587331	189.265390
Lote	18.043252	21.907315	158.863114
Mahad	36.815866	18.395647	186.471455
Mumbai	29.660227	8.245033	192.484112
Nagpur	28.544353	10.032638	185.874141

location	no2	so2	spm
Nanded	37.185754	38.083357	180.000000
Nashik	27.205053	27.070612	164.791254
Navi Mumbai	39.788710	17.270234	218.691271
Pune	45.333006	24.659954	194.917552
Roha	34.291339	18.759843	168.283465
Sangli	39.231521	10.976787	175.430055
Solapur	36.335870	16.611888	247.254144
Tarapur	33.193643	21.770421	134.472973

Air Quality index

In [27]:
```
# Calculate so2
def calculate_si(so2):
    si=0
    if (so2<=40):
     si= "s1"
    if (so2>40 and so2<=80):
     si= "s2"
    if (so2>80 and so2<=380):
     si= "s3"
    if (so2>380 and so2<=800):
     si= "s4"
    if (so2>800 and so2<=1600):
     si= "s5"
    if (so2>1600):
     si= "s6"
    return si
df['si']= df['so2'].apply(calculate_si)
ds= df[['so2','si']]
```

In [28]:
```
# Calculate No2
def calculate_ni(no2):
    ni=0
    if (no2<=40):
     ni= "n1"
    if (no2>40 and no2<=80):
     ni= "n2"
    if (no2>80 and no2<=180):
     ni= "n3"
    if (no2>180 and no2<=280):
     ni= "n4"
    if (no2>280 and no2<=400):
     ni= "n5"
    if (no2>400):
     ni= "n6"
    return ni
df['ni']= df['no2'].apply(calculate_ni)
dn= df[['no2','ni']]
```

In [29]:
```
# Calculate SPM
def calculate_spi(spm):
    spi=0
    if (spm<=40):
      spi= "sp1"
    if (spm>40 and spm<=80):
      spi= "sp2"
    if (spm>80 and spm<=180):
      spi= "sp3"
    if (spm>180 and spm<=280):
      spi= "sp4"
    if (spm>280 and spm<=400):
      spi= "sp5"
    if (spm>400):
      spi= "sp6"
    return  spi
df['spi']= df['spm'].apply(calculate_spi)
dsp= df[['spm','spi']]
```

In [30]:
```
# Calculate AQI
def calculate_aqi(si,ni,spi):
    aqi=0
    if(si>ni and si>spi):
     aqi=si
    if (spi>ni and spi>si):
     aqi=spi
    if(ni>si and ni>spi):
     aqi= ni
    return aqi
df['AQI']= df.apply(lambda x:calculate_aqi(x['so2'],x['no2'],x['spm']),axis=1)
```

In [31]:
```
df.head(3)
```

Out[31]:

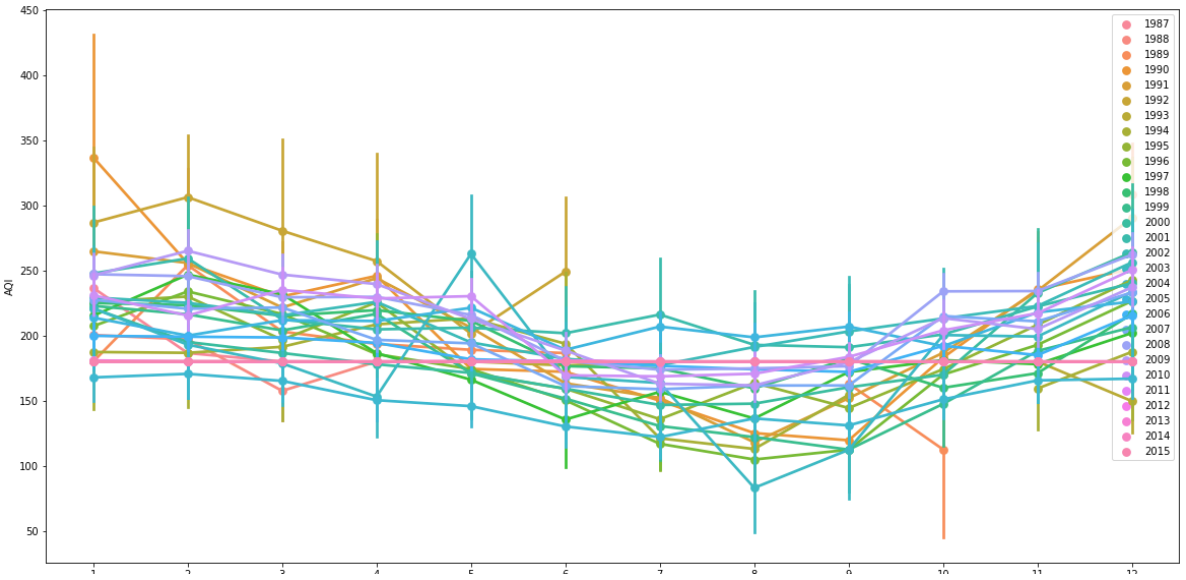
	state	location	type	so2	no2	rspm	spm	date	si	ni	spi
197243	Maharashtra	Pune	Residential	8.100000	9.0	90.0	180.0	1987-01-07	s1	n1	sp3
197244	Maharashtra	Pune	Residential	0.700000	47.6	90.0	293.0	1987-01-12	s1	n2	sp5
197237	Maharashtra	Pune	Residential	17.366863	10.4	90.0	180.0	1987-02-05	s1	n1	sp3

In [32]:
```
df['year'] = df['date'].dt.year
```

In [33]:
```
df['month'] = df['date'].dt.month
```

In [34]:
```
plt.figure(figsize=(20, 10))
sns.pointplot(x="month", y='AQI', hue="year", data=df, )
plt.xlabel("month")
plt.ylabel('AQI')
plt.legend(loc='upper right')
```

Out[34]:
```
<matplotlib.legend.Legend at 0x1e03f67ae50>
```



In [35]:
```
df.head()
```

Out[35]:

	state	location	type	so2	no2	rspm	spm	date	si	ni	spi
197243	Maharashtra	Pune	Residential	8.100000	9.0	90.0	180.0	1987-01-07	s1	n1	sp3
197244	Maharashtra	Pune	Residential	0.700000	47.6	90.0	293.0	1987-01-12	s1	n2	sp5
197237	Maharashtra	Pune	Residential	17.366863	10.4	90.0	180.0	1987-02-05	s1	n1	sp3
197239	Maharashtra	Pune	Industrial	11.300000	16.3	90.0	180.0	1987-02-06	s1	n1	sp3
197242	Maharashtra	Pune	Industrial	9.700000	10.9	90.0	180.0	1987-02-07	s1	n1	sp3

In [36]:
```
# Decomposing to see white noiise
decompose = df[['AQI']]
decompose.index =  df['date']
decompose = decompose[['AQI']]
decompose.head()
```

Out[36]:

	AQI
date	
1987-01-07	180.0
1987-01-12	293.0
1987-02-05	180.0
1987-02-06	180.0
1987-02-07	180.0

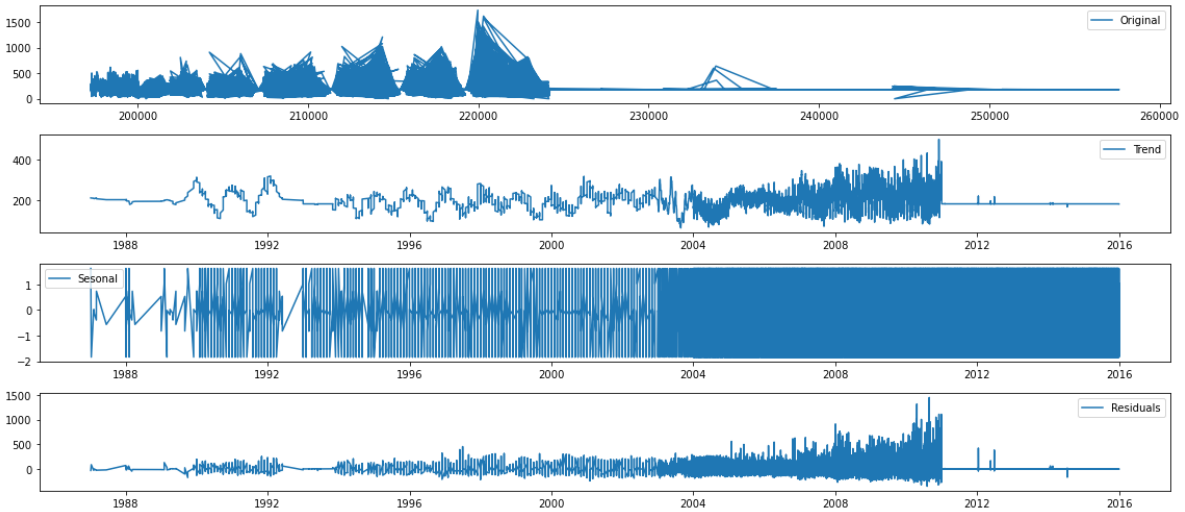
Importing Decomposition model and plotting graph

Trend, Sesonality & Residual

In [37]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(decompose, model='additive', freq=12, extrapo
```

In [38]:
```
trend = decomposition.trend
sesonal = decomposition.seasonal
resid = decomposition.resid
```

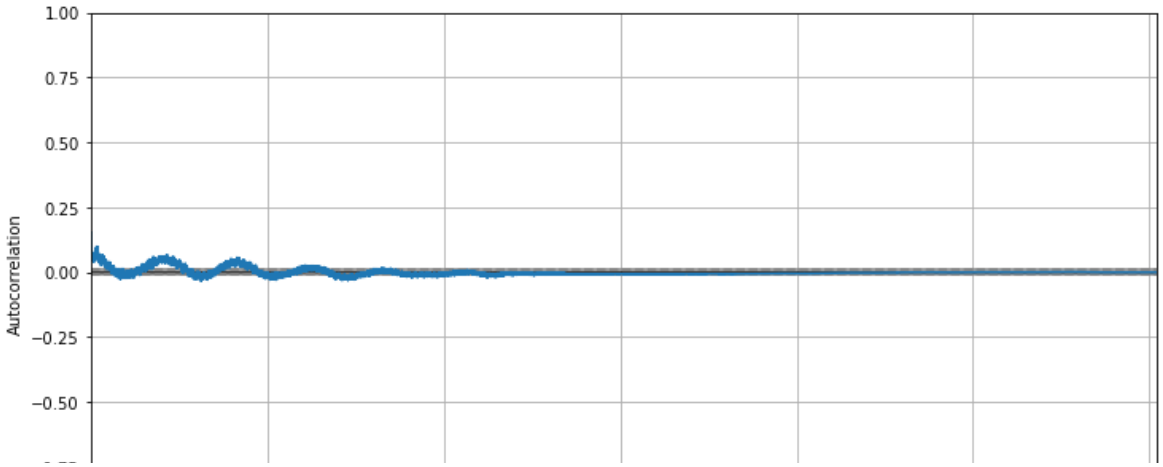
In [39]:
```
plt.figure(figsize=(16,7))
plt.subplot(411)
plt.plot(df['AQI'], label='Original')
plt.legend(loc='best')

plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')

plt.subplot(413)
plt.plot(sesonal, label='Sesonal')
plt.legend(loc='best')

plt.subplot(414)
plt.plot(resid, label='Residuals')
plt.legend(loc='best')

plt.tight_layout()
```



AutoCorrelation

In [40]:
```
# Plotting Graph for autocorelation
from pandas.plotting import autocorrelation_plot
plt.figure(figsize=(12,6))
autocorrelation_plot(df['AQI'])
```

Out[40]:
```
<AxesSubplot:xlabel='Lag', ylabel='Autocorrelation'>
```



Importing data

In [41]:
```
import datetime as dt
import math
from sklearn.metrics import mean_squared_error
```

In [64]:
```
ts = pd.pivot_table(df, values=['AQI'],index='date')
ts.head()
ts.shape
```

Out[64]:
```
(4686, 1)
```

In [66]:
```
# training Data and testing data
x_train = ts[ts.index < dt.datetime(2013,1,1,0,0,0)]
x_test = ts[ts.index >= dt.datetime(2013,1,1,0,0,0)]
print(x_train.shape)
print(x_test.shape)
```

```
(3591, 1)
(1095, 1)
```

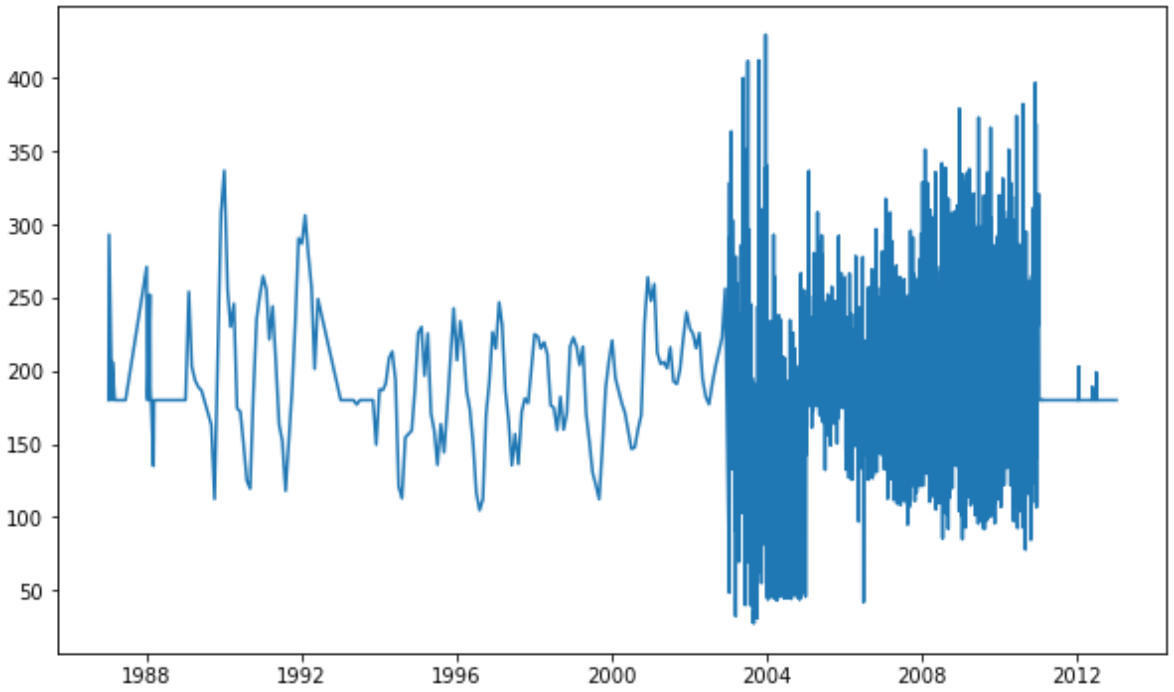
Stationarity Check

For stationarity check we will use Augmented Dicky Fuller Test

In [67]:
```
from statsmodels.tsa.stattools import adfuller

def stationarity_check(data):
    df_test = adfuller(data.AQI, autolag='AIC')
    df_output = pd.Series(df_test[0:4], index= ['Test Static: ', 'P-Value: ','#I
    for key,value in df_test[4].items():
        df_output['Criticat Value (%s)'%key] = value
    print(df_output)

    plt.figure(figsize=(10,6))
    plt.plot(data.index, data.AQI)
    plt.show()
```

In [68]:
```
stationarity_check(x_train)
```

```
Test Static:                -3.616469
P-Value:                     0.005454
#Lags Used :                30.000000
No. Of Observation :      3560.000000
Criticat Value (1%)          -3.432188
Criticat Value (5%)          -2.862352
Criticat Value (10%)         -2.567202
dtype: float64
```

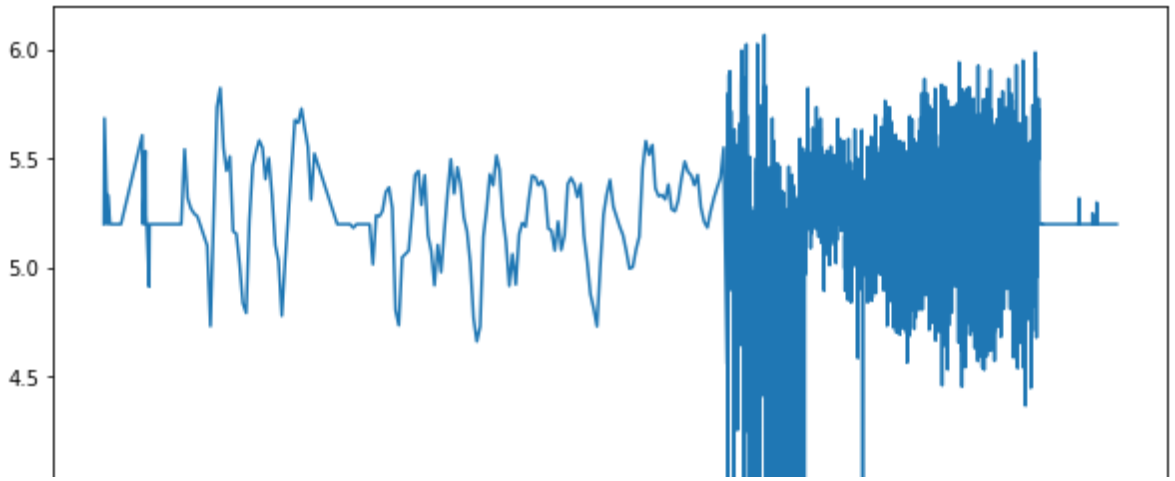


- Data is Highly stationary
- we need to apply log transformation to make variance constant

In [69]:
```
log_train = x_train
log_train = log_train['AQI'].apply(lambda x : math.log(x+1))
log_train = pd.DataFrame(log_train)
```

In [70]:
```
stationarity_check(log_train)
```

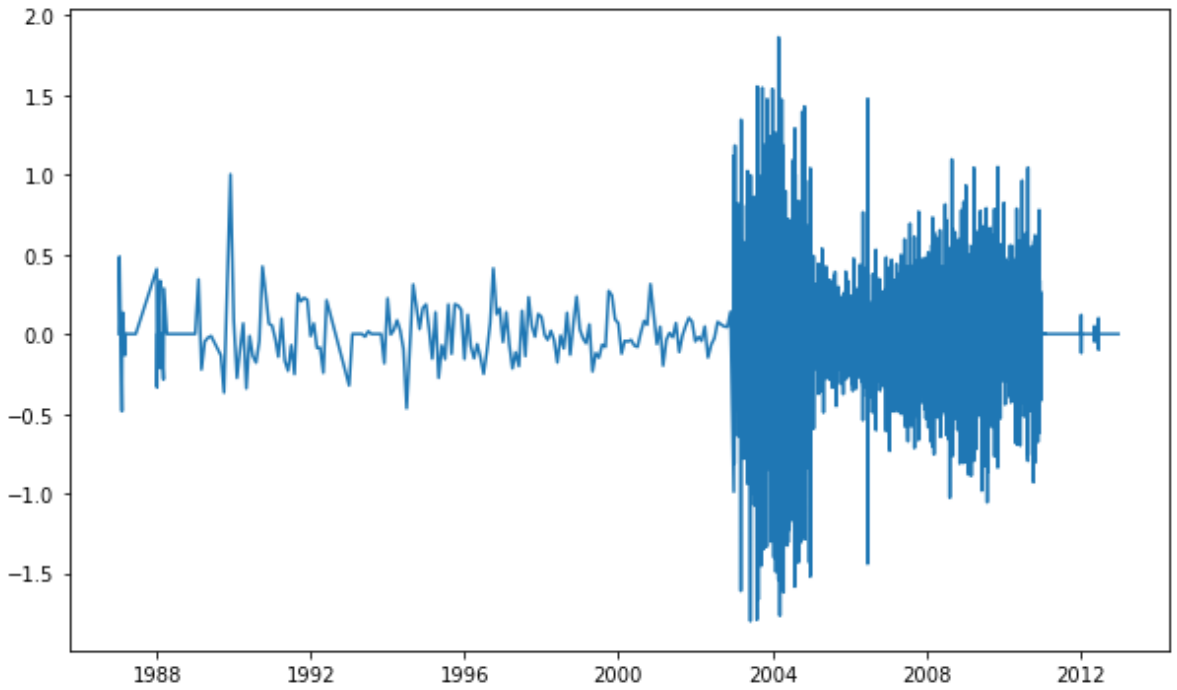
```
Test Static:                -3.814677
P-Value:                     0.002761
#Lags Used :                30.000000
No. Of Observation :      3560.000000
Criticat Value (1%)          -3.432188
Criticat Value (5%)          -2.862352
Criticat Value (10%)         -2.567202
dtype: float64
```



In [71]:
```
first_diff = log_train['AQI'] - log_train['AQI'].shift(1)
first_diff = first_diff.fillna(0)
first_diff = pd.DataFrame(first_diff)
```

In [72]:
```
stationarity_check(first_diff)
```

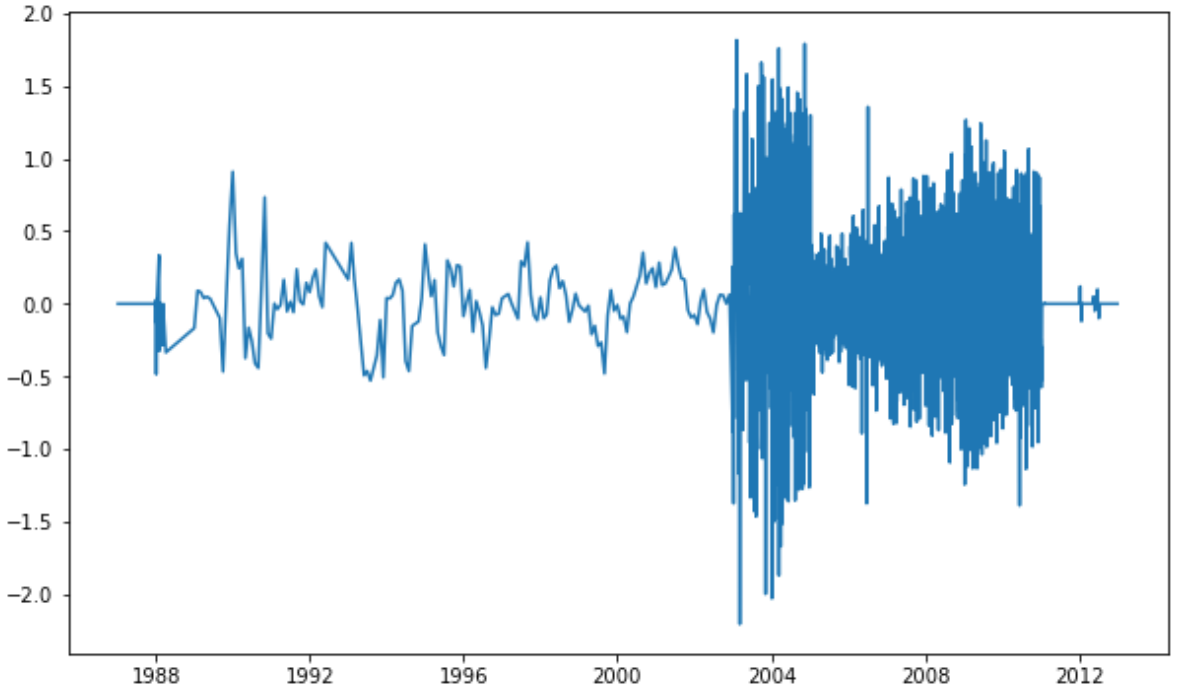
```
Test Static:             -21.733371
P-Value:                   0.000000
#Lags Used :              30.000000
No. Of Observation :    3560.000000
Criticat Value (1%)        -3.432188
Criticat Value (5%)        -2.862352
Criticat Value (10%)       -2.567202
dtype: float64
```



In [73]:
```
# Using Sesonality Difference
seasonal_diff = log_train['AQI'] - log_train['AQI'].shift(12)
seasonal_diff = seasonal_diff.fillna(0)
seasonal_diff = pd.DataFrame(seasonal_diff)
```

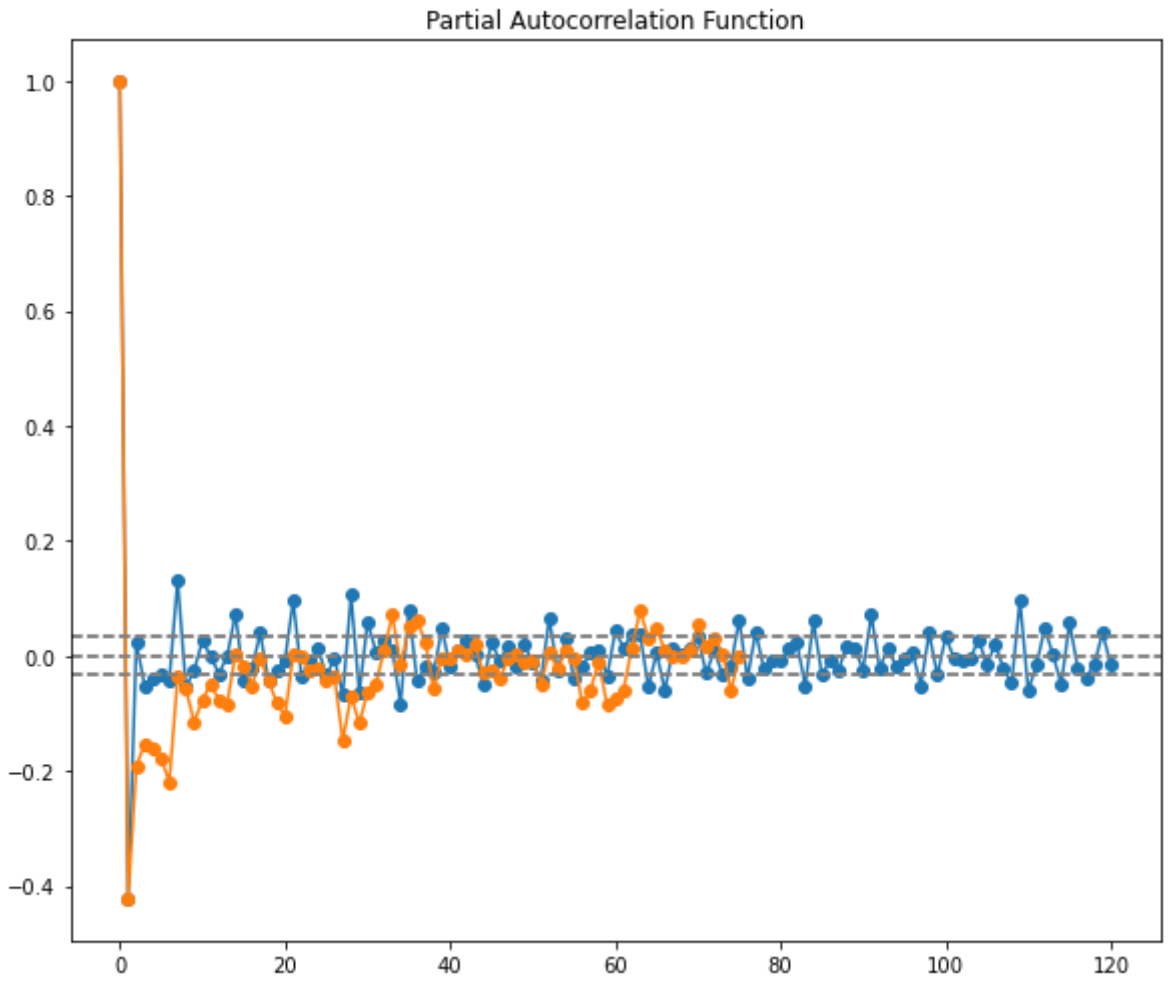
In [74]:
```
stationarity_check(seasonal_diff)
```

```
Test Static:           -1.132752e+01
P-Value:                1.133852e-20
#Lags Used :            3.000000e+01
No. Of Observation :    3.560000e+03
Criticat Value (1%)    -3.432188e+00
Criticat Value (5%)    -2.862352e+00
Criticat Value (10%)   -2.567202e+00
dtype: float64
```



Plotting Acf and Pacf plots:

In [75]:
```
from statsmodels.tsa.stattools import acf, pacf
lag_Acf = acf(first_diff, nlags=120)
lag_Pacf = pacf(first_diff, nlags=75, method='ols')
plt.figure(figsize=(16,7))

# ACF Plot
plt.subplot(121)
plt.plot(lag_Acf, marker='o')
plt.axhline(y=0, linestyle='--', color='grey')
plt.axhline(y=-1.96/np.sqrt(len(first_diff)), linestyle='--', color='grey')
plt.axhline(y=1.96/np.sqrt(len(first_diff)), linestyle='--', color='grey')
plt.title("Autocorrelation Function")

# PACF Plot
plt.subplot(121)
plt.plot(lag_Pacf, marker='o')
plt.axhline(y=0, linestyle='--', color='grey')
plt.axhline(y=-1.96/np.sqrt(len(first_diff)), linestyle='--', color='grey')
plt.axhline(y=1.96/np.sqrt(len(first_diff)), linestyle='--', color='grey')
plt.title("Partial Autocorrelation Function")

plt.tight_layout()
```



AR Model

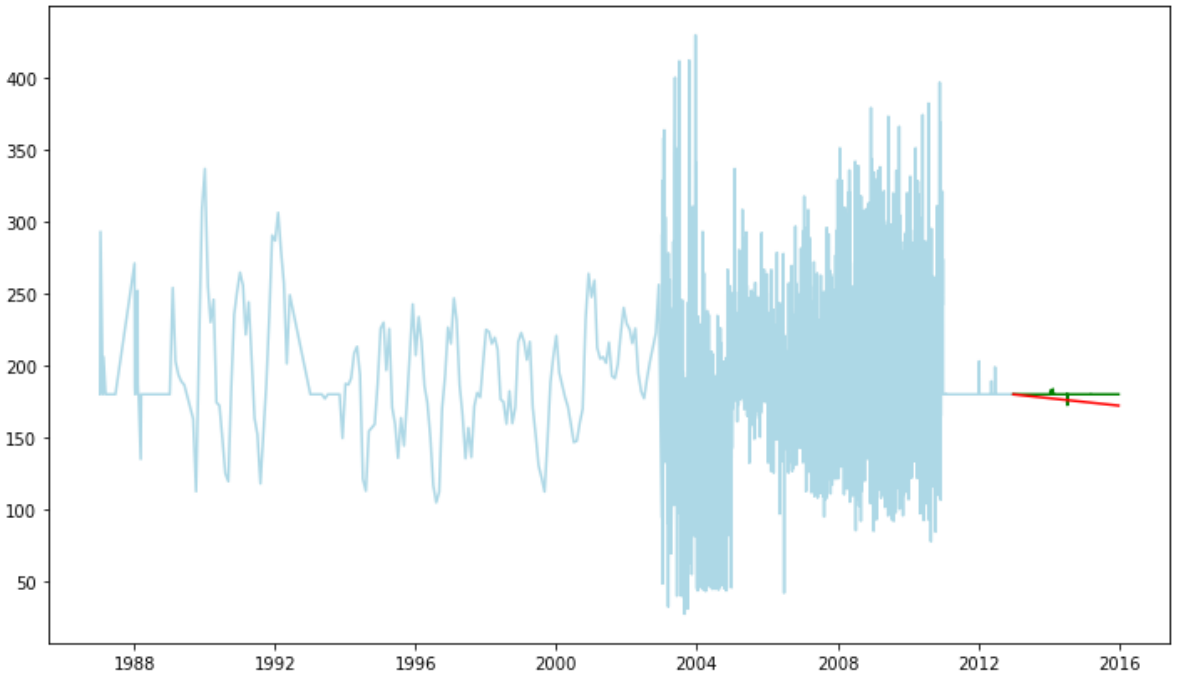
In [87]:
```
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(log_train, order=(1,1,0))
result_arima = model.fit(disp=-1)
print(result_arima.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                   D.AQI   No. Observations:                 3590
Model:                 ARIMA(1, 1, 0)   Log Likelihood                -599.077
Method:                       css-mle   S.D. of innovations              0.286
Date:                Sat, 19 Nov 2022   AIC                           1204.154
Time:                        18:00:02   BIC                           1222.712
Sample:                             1   HQIC                          1210.768

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       -4.025e-05      0.003     -0.012      0.990      -0.007       0.007
ar.L1.D.AQI    -0.4240      0.015    -28.049      0.000      -0.454      -0.394
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -2.3583           +0.0000j            2.3583            0.5000
-----------------------------------------------------------------------------
```

In [77]:

```
plt.figure(figsize=(12,7))
plt.plot(x_train.index, x_train.values, color='lightblue')
plt.plot(x_test.index, x_test.values, color='green')

# For checking forecast
pred = pd.DataFrame(result_arima.forecast(len(x_test))[0])
pred.columns = ['yhat']
pred.index = x_test.index

# Converting from log to norm
pred['yhat'] = pred['yhat'].apply(lambda x : math.exp(x)-1)

measure = math.pow(mean_squared_error(x_test.values, pred.values), 0.5)
print(measure)
plt.plot(pred.index, pred.fillna(0).values, color='red')
plt.show()
```

```
4.5432006310227715
```



MA Model

In [85]:

```
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(log_train, order=(0,1,1))
result_ma = model.fit(disp=-1)
print(result_ma.summary())
```

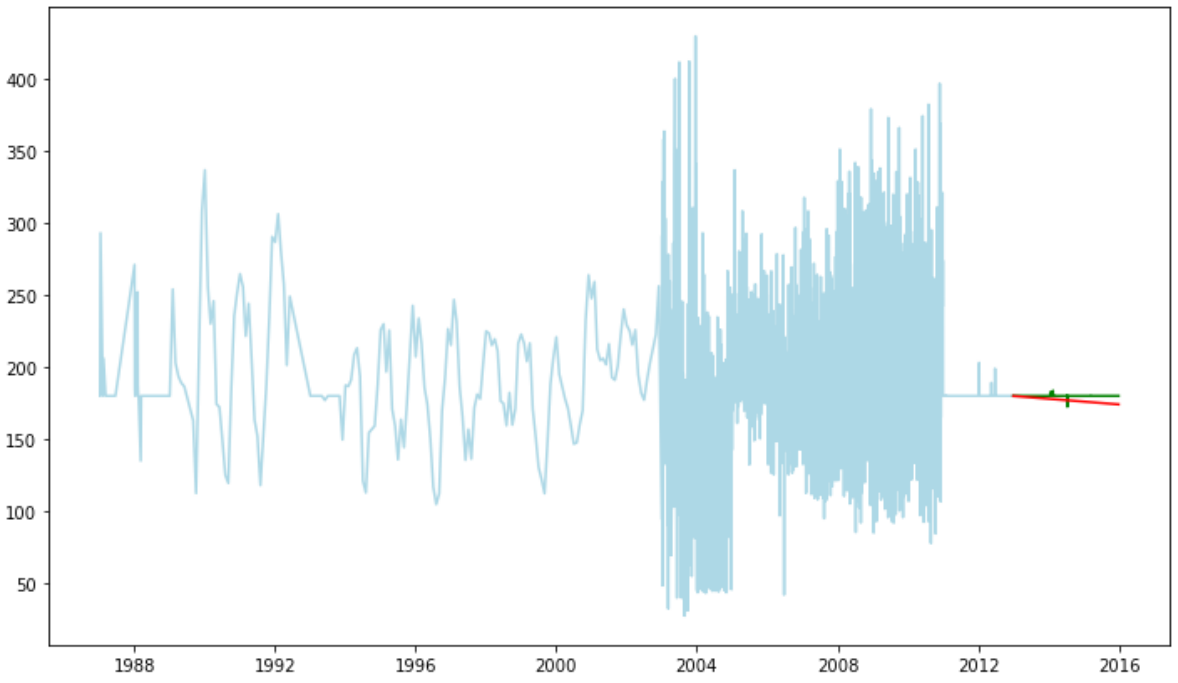
```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                  D.AQI   No. Observations:                 3590
Model:                 ARIMA(0, 1, 1)   Log Likelihood                -350.959
Method:                       css-mle   S.D. of innovations              0.267
Date:                Sat, 19 Nov 2022   AIC                            707.918
Time:                        17:59:53   BIC                            726.475
Sample:                             1   HQIC                           714.532

==============================================================================
```

```
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       -3.003e-05      0.001     -0.041      0.967      -0.001       0.001
ma.L1.D.AQI    -0.8348      0.018    -45.578      0.000      -0.871      -0.799
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1            1.1979           +0.0000j            1.1979            0.0000
```

In [86]:

```
plt.figure(figsize=(12,7))
plt.plot(x_train.index, x_train.values, color='lightblue')
plt.plot(x_test.index, x_test.values, color='green')

# For checking forecast
pred = pd.DataFrame(result_ma.forecast(len(x_test))[0])
pred.columns = ['yhat']
pred.index = x_test.index

# Converting from log to norm
pred['yhat'] = pred['yhat'].apply(lambda x : math.exp(x)-1)

measure = math.pow(mean_squared_error(x_test.values, pred.values), 0.5)
print(measure)
plt.plot(pred.index, pred.fillna(0).values, color='red')
plt.show()
```

```
3.431366850928005
```



ARIMA Model

In [83]:

```
from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(log_train, order=(1,1,1))
result_arima = model.fit(disp=-1)
```

In [84]:

```
plt.figure(figsize=(12,7))
plt.plot(x_train.index, x_train.values, color='lightblue')
plt.plot(x_test.index, x_test.values, color='green')

# For checking forecast
pred = pd.DataFrame(result_arima.forecast(len(x_test))[0])
pred.columns = ['yhat']
pred.index = x_test.index

# Converting from log to norm
pred['yhat'] = pred['yhat'].apply(lambda x : math.exp(x)-1)

measure = math.pow(mean_squared_error(x_test.values, pred.values), 0.5)
print(measure)
plt.plot(pred.index, pred.fillna(0).values, color='red')
plt.show()
```

```
2.828188771834781
```