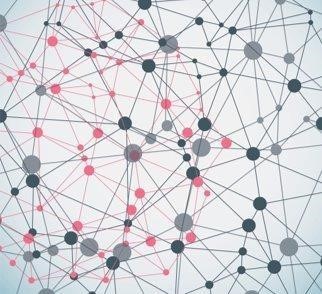
AIR QUALITY ANALYSIS AND PREDICTION IN TAMIL NADU



PROJECT REPORT PHASE- 4

By

* GIRIPRASATH S - 420421104020
* GUGAN M -420421104024
* HEMANTH RAO BK -420421104027
* JANAKIRAMAN S -420421104029

INTRODUCTION

Air pollution is one of the greatest environmental risk to health. By reducing air pollution levels, countriescan reduce the burden of disease from stroke, heart disease, lung cancer, and both chronic and acute respiratory diseases, including asthma. Here we are studied about the air quality analysis methods in Tamil Nadu

Content for Project Phase 4:

For analyzing data, we need some libraries. In this

section, we are importing all the required libraries like pandas NumPy, matplotlib, plotly, seaborn, and word cloud that are required for data analysis. Check the below code to import all the required libraries Data Source: A good data source for credit card fraud detection should be accurate,complete, Covering the geographic area of interest, Accessible.

EXPLORATORY DATA ANALYSIS

Exploratory data analysis is performed on the raw data. The insights gained from the analysis helps to identify the pre- processing tasks that need to be performed to form the dataset for building the air quality prediction model.

import numpy as np *# linear algebra import* pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*import* matplotlib.pyplot as plt import seaborn as sns

%matplotlib inline plt.rcParams['figure.figsize'] = (10, 7)

# Warnings import warnings warnings.filterwarnings('ignore')

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.

['lat-lon-indianstates', 'india-air-quality-data', 'indian-states-lat- lon']

In [2]: data=pd.read\_csv('../input/india-air- quality-data/data.csv',encoding="ISO-8859-

1") data.fillna(0, inplace=True) data.head()

Out[2]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | stn\_c ode | samp  ling\_ date | state | locati on | agen cy | type | so2 | no2 | rspm | spm | locati  on\_m  onito ring\_ statio n | pm2\_  5 | date |
| 0 | 150 | Febru  ary -  M021  990 | Andh  ra  Prade sh | Hyde rabad | 0 | Resid  ential  ,  Rural and other Areas | 4.8 | 17.4 | 0.0 | 0.0 | 0 | 0.0 | 1990-  02-01 |
| 1 | 151 | Febru  ary -  M021  990 | Andh  ra  Prade sh | Hyde rabad | 0 | Indus  trial  Area | 3.1 | 7.0 | 0.0 | 0.0 | 0 | 0.0 | 1990-  02-01 |
| 2 | 152 | Febru  ary -  M021  990 | Andh  ra  Prade sh | Hyde rabad | 0 | Resid  ential  ,  Rural and other Areas | 6.2 | 28.5 | 0.0 | 0.0 | 0 | 0.0 | 1990-  02-01 |
| 3 | 150 | Marc  h -  M031  990 | Andh  ra  Prade sh | Hyde rabad | 0 | Resid  ential  ,  Rural and other Areas | 6.3 | 14.7 | 0.0 | 0.0 | 0 | 0.0 | 1990-  03-01 |
| 4 | 151 | Marc  h -  M031  990 | Andh  ra  Prade sh | Hyde rabad | 0 | Indus  trial  Area | 4.7 | 7.5 | 0.0 | 0.0 | 0 | 0.0 | 1990-  03-01 |

In [3]:

*#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

data['si']=data['so2'].apply(calculate\_si) df= data[['so2','si']] df.head()

Out[3]:

|  |  |  |
| --- | --- | --- |
|  | 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 [4]:

*#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

data['ni']=data['no2'].apply(calculate\_ni) df= data[['no2','ni']]

df.head()

Out[4]:

|  |  |  |
| --- | --- | --- |
|  | 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 [5]:

*#Function to calculate no2 individual pollutant index(rpi) 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 data['rpi']=data['rspm'].apply(calculate\_si

) df= data[['rspm','rpi']] df.tail()

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

Out[5]:

|  |  |  |
| --- | --- | --- |
|  | 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 [6]:

*#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

data['spi']=data['spm'].apply(calculate\_spi

) df= data[['spm','spi']] df.tail()

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

Out[6]:

|  |  |  |
| --- | --- | --- |
|  | 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 [7]:

*#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

data['AQI']=data.apply(lambda x:calculate\_aqi(x['si'],x['ni'],x['spi'],x[

'rpi']),axis=1) df= data[['sampling\_date','state','si','ni','rp i','spi','AQI']]

df.head()

Out[7]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | sampling\_da te | 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 [8]:

df.state.unique()

Out[8]:

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 [9]:

state=pd.read\_csv("../input/indian-states- lat-lon/lat.csv") state.head() df.head()

Out[9]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | sampling\_da te | 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]:

dff=pd.merge(state.set\_index("state"),df.se t\_index("state"), right\_index=True, left\_index=True).reset\_index() dff.head()

Out[10]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | state | lat | lon | sampling \_date | si | ni | rpi | spi | AQI |
| 0 | Andhra  Pradesh | 14.75042  9 | 78.57002  6 | February  -  M02199  0 | 6.000 | 21.750 | 0.0 | 0.0 | 21.750 |
| 1 | Andhra  Pradesh | 14.75042  9 | 78.57002  6 | February  -  M02199  0 | 3.875 | 8.750 | 0.0 | 0.0 | 8.750 |
| 2 | Andhra  Pradesh | 14.75042  9 | 78.57002  6 | February  -  M02199  0 | 7.750 | 35.625 | 0.0 | 0.0 | 35.625 |
| 3 | Andhra Pradesh | 14.75042  9 | 78.57002  6 | March - M03199  0 | 7.875 | 18.375 | 0.0 | 0.0 | 18.375 |
| 4 | Andhra Pradesh | 14.75042  9 | 78.57002  6 | March - M03199  0 | 5.875 | 9.375 | 0.0 | 0.0 | 9.375 |

In [11]:

from mpl\_toolkits.basemap import Basemap

%matplotlib inline import warnings warnings.filterwarnings('ignore')

%config InlineBackend.figure\_format =

'retina'

In [12]:

m =

Basemap(projection='mill',llcrnrlat=5,urcrn rlat=40, llcrnrlon=60,urcrnrlon=110,lat\_ts=20,resolu tion='c')

In [13]:

longitudes = dff["lon"].tolist() latitudes = dff["lat"].tolist()

#m =

Basemap(width=12000000,height=9000000,proje ction='lcc',

*#resolution=None,lat\_1=80.,lat\_2=55,lat\_0=80*

*,lon\_0=-107.) x,y* = m(longitudes,latitudes)

In [14]:

fig = plt.figure(figsize=(12,10)) plt.title("All affected areas")

m.plot(x, y, "o", markersize = 3, color =

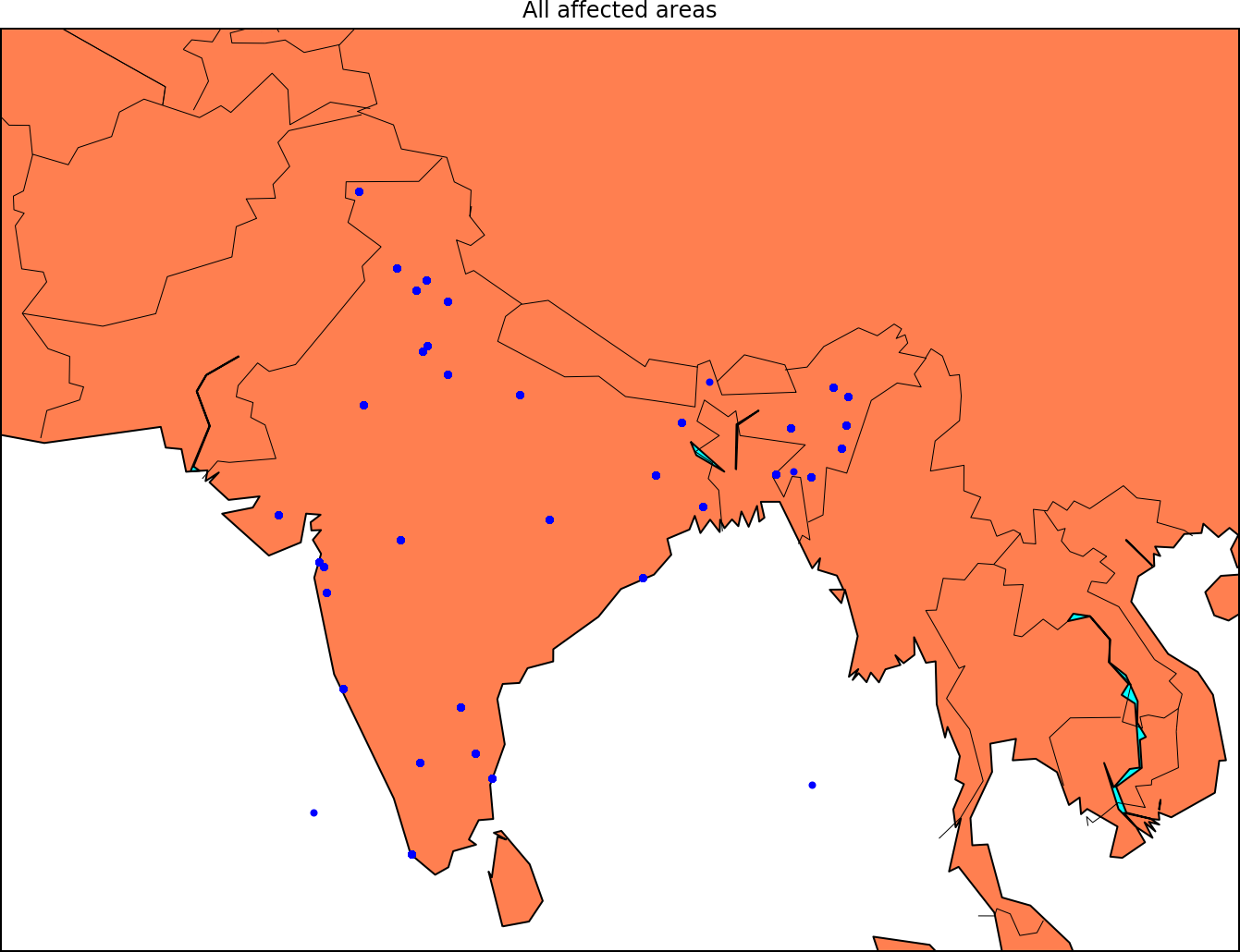
'blue')

m.drawcoastlines()

m.fillcontinents(color='coral',lake\_color=' aqua')

m.drawmapboundary()

m.drawcountries() plt.show()



data['date'] =

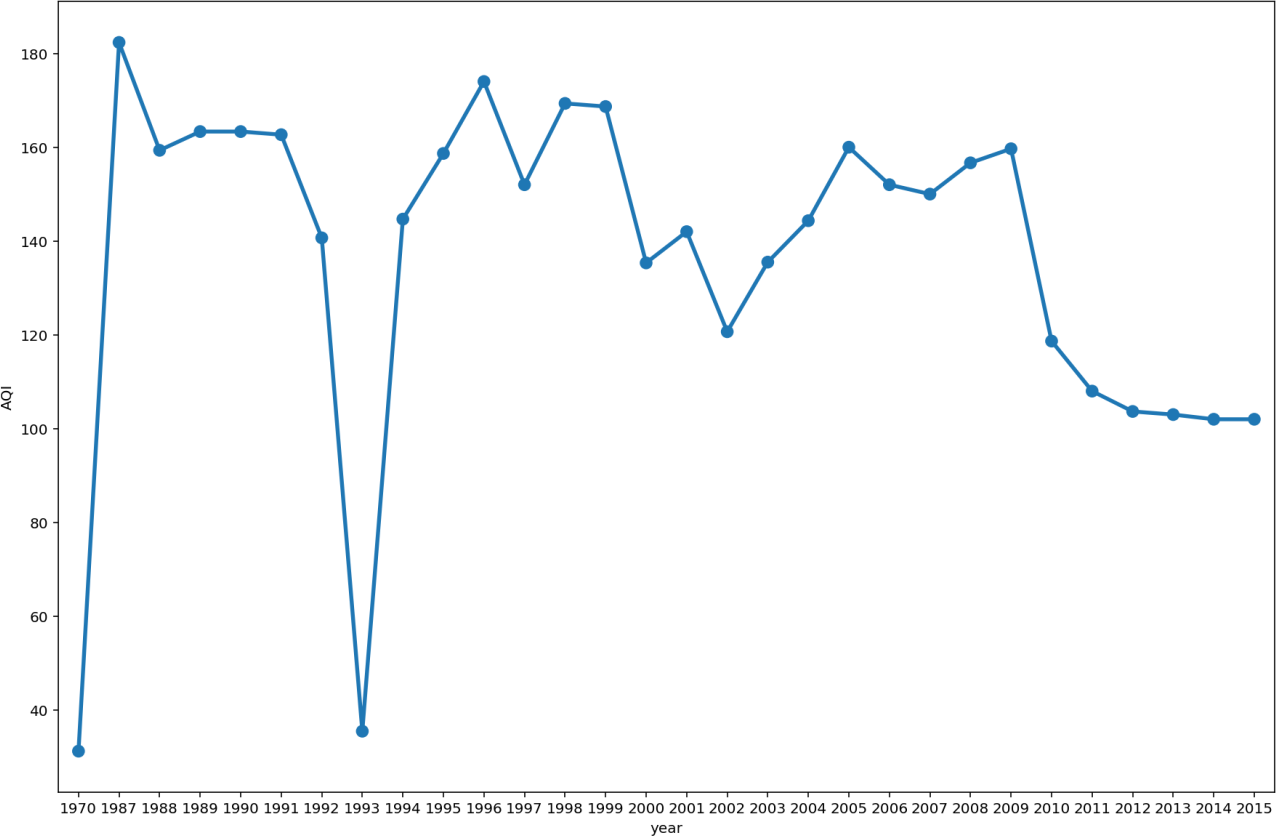
pd.to\_datetime(data['date'],format='%Y-%m-

**%d**') *# date parse*

*data[*'year'] = data['date'].dt.year *# year data[*'year'] = data['year'].fillna(0.0).astype(int) data = data[(data['year']>0)] df = data[['AQI','year','state']].groupby(["year "]).median().reset\_index().sort\_values(by=' year',ascending=False) f,ax=plt.subplots(figsize=(15,10)) sns.pointplot(x='year', y='AQI', data=df)

Out[15]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f72c28f4cc0>



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 df=data[['AQI','date']] df["date"] = pd.to\_datetime(df['date']) df.tail(20)

Out[16]:

|  |  |  |
| --- | --- | --- |
|  | 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 |
| 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 [17]:

*#Calculating the yearly mean for the data df*=df.set\_index('date').resample('M')["AQI"

].mean() df.head()

Out[17]: 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 [18]:

*#preprocessing the data values*

*data*=df.reset\_index(level=0, inplace=False) data = data[np.isfinite(data['AQI'])] data=data[data.date != '1970-01-31'] data = data.reset\_index(drop=True) data.head()

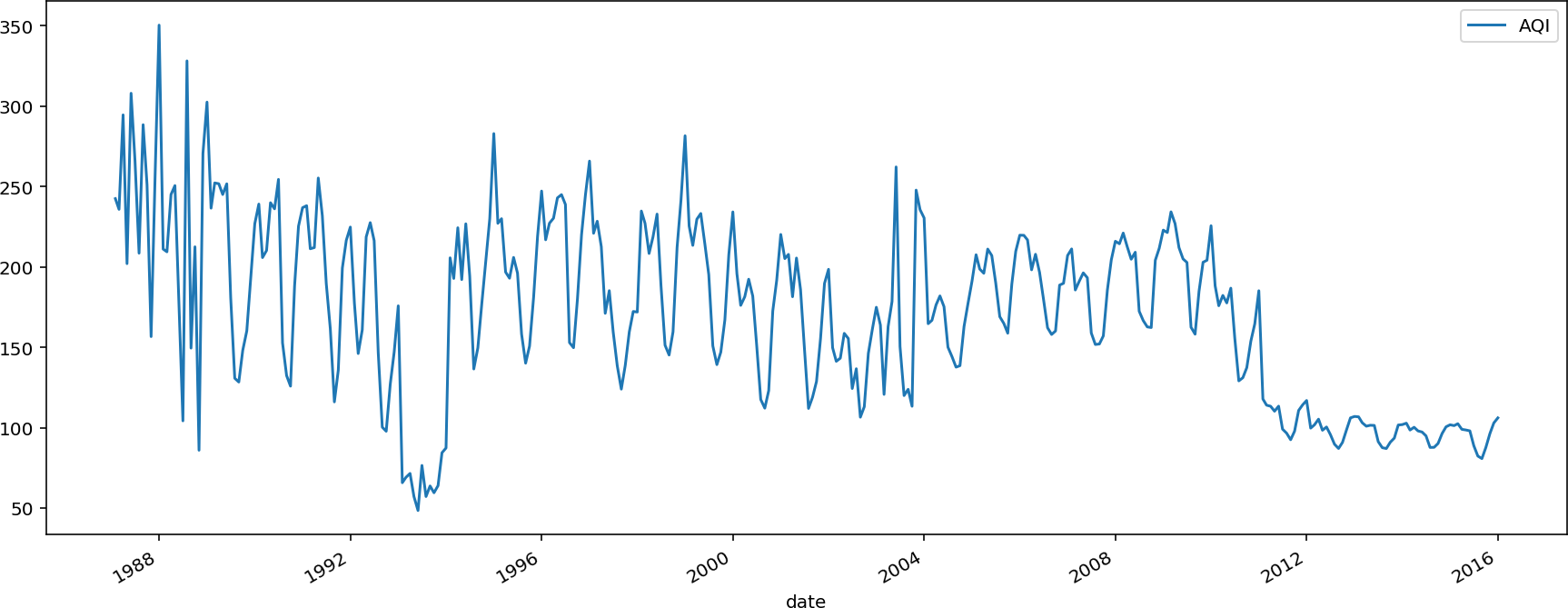
Out[18]:

|  |  |  |
| --- | --- | --- |
|  | 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 [19]:

*#visualizing the processed data of AQI*

*df*=data.set\_index('date') df.sort\_values(by='date',ascending=False) df.plot(figsize=(15, 6)) plt.show() y=df.AQI



In [20]:

*#exctracting knowledge about data*

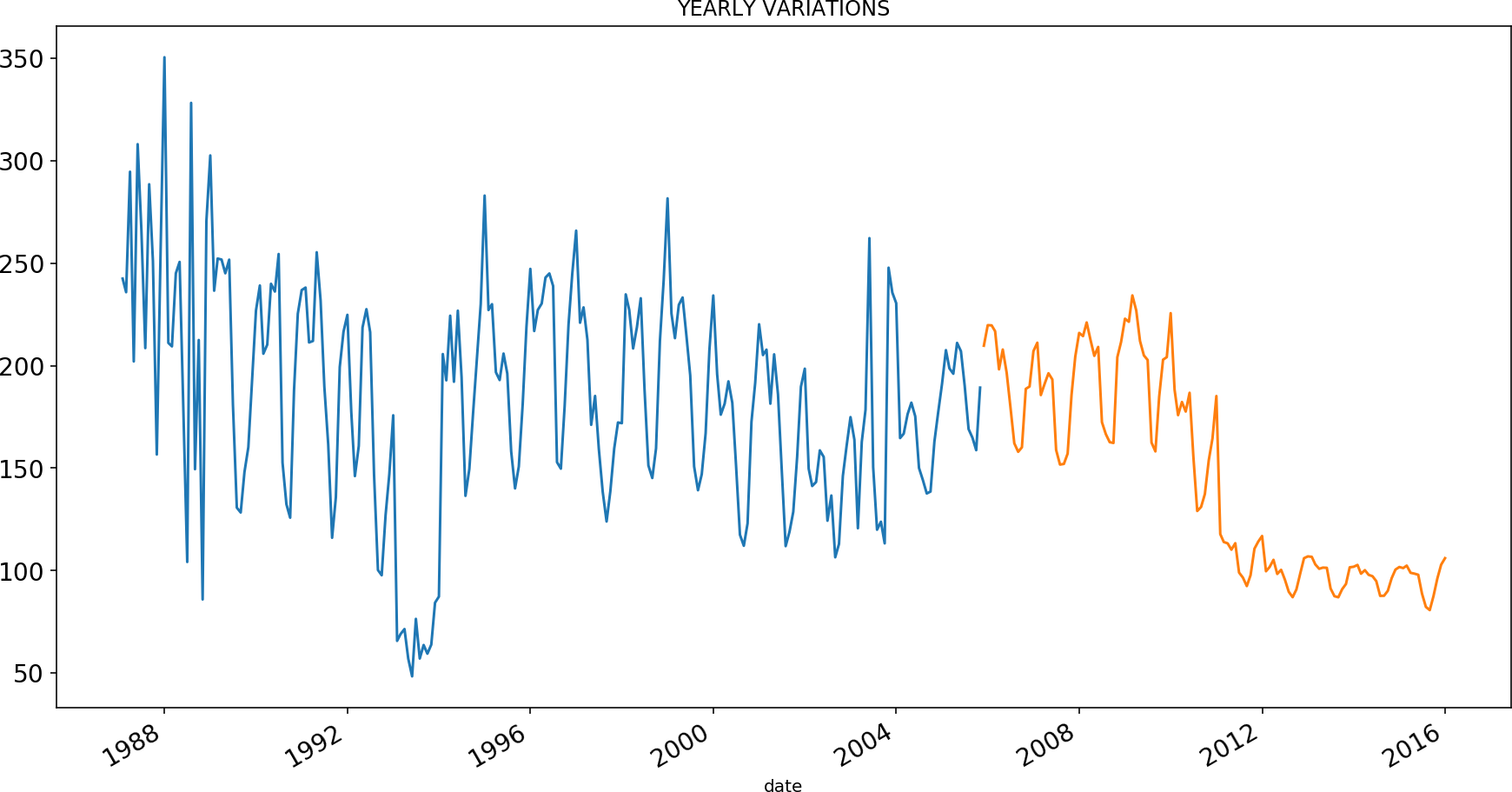
*#spliting dataframes into test and train n* = df.shape[0] train\_size = 0.65

features\_dataframe = df.sort\_values('date') train = df.iloc[:int(n \* train\_size)] test = df.iloc[int(n \* train\_size):]

In [21]:

*#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.show()

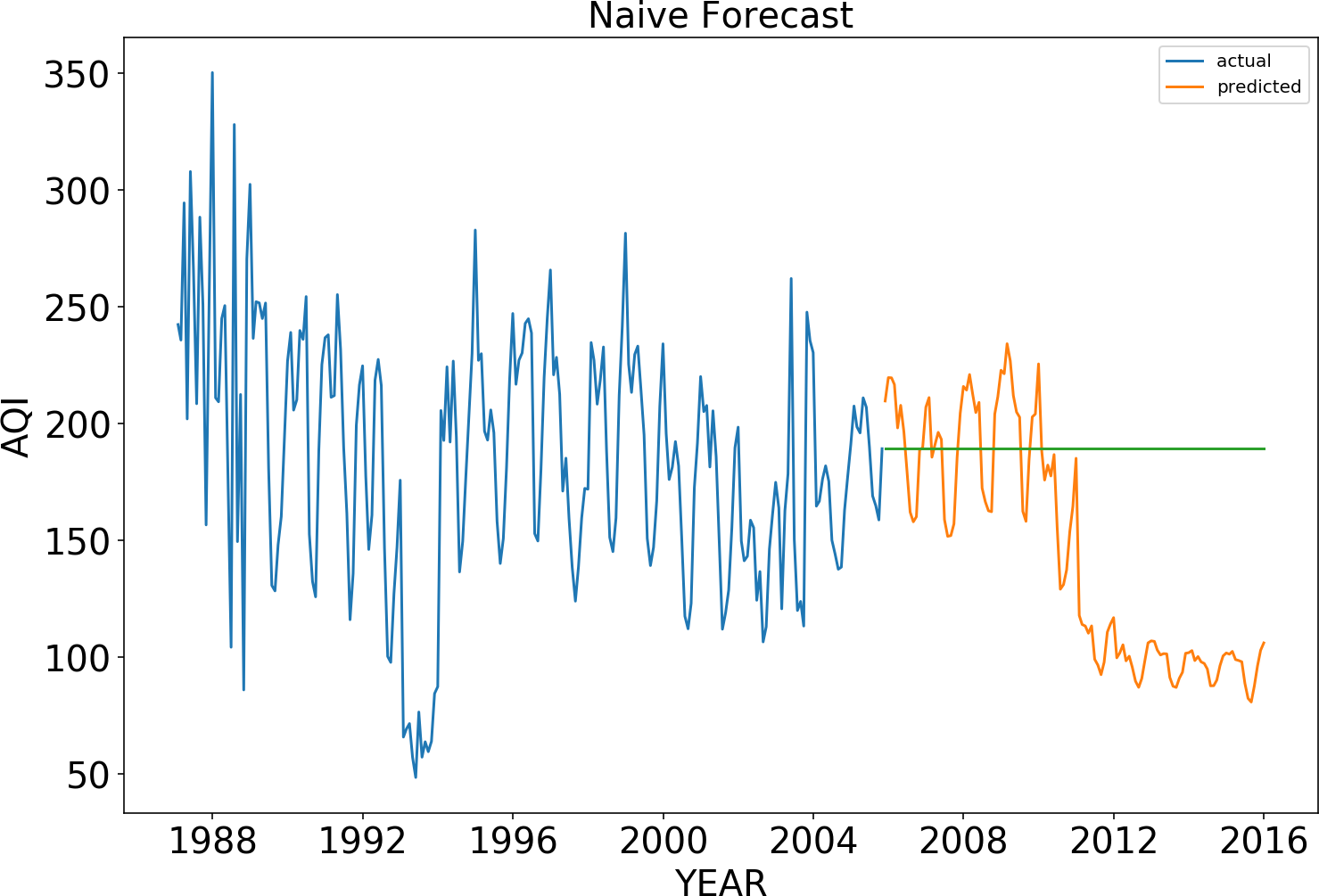


In [22]:

*#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)

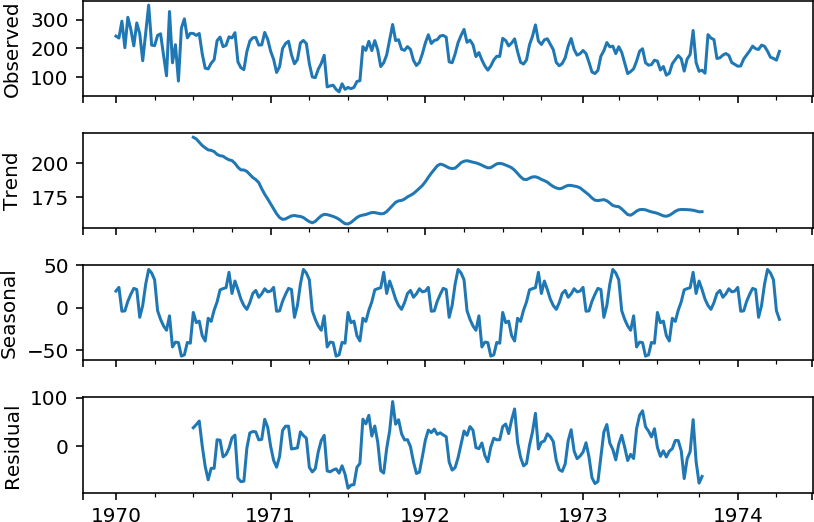
plt.legend(["actual ","predicted"]) plt.xlabel("YEAR",fontsize=20) plt.ylabel("AQI",fontsize=20) plt.tick\_params(labelsize=20) plt.show()



In [23]:

*#various statmodel to identity huge variations od data values import* statsmodels.api as sm train.index=pd.DatetimeIndex(freq="w", start=0 ,periods=224)

sm.tsa.seasonal\_decompose(train.AQI).plot() result = sm.tsa.stattools.adfuller(train.AQI) plt.show()



In [24]:

*#resampling the data to predict monthly AQI of india df*=data[['AQI','date']]

df['date']=pd.to\_datetime(df['date']) date=df.groupby(pd.Grouper(key='date',freq=

'1MS'))["AQI"].mean() df.count()

Out[24]:

AQI 346

date 346

dtype: int64

In [25]:

*#splitting the sampling date into month and year accordingly*

*data[*'month'] = data['date'].dt.month data['year'] = data['date'].dt.year data=data[['AQI','date','month','year']] data.head()

Out[25]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 [26]:

*#predicting JANUARY-AQI across india data*=data[data['month']==1] data.head()

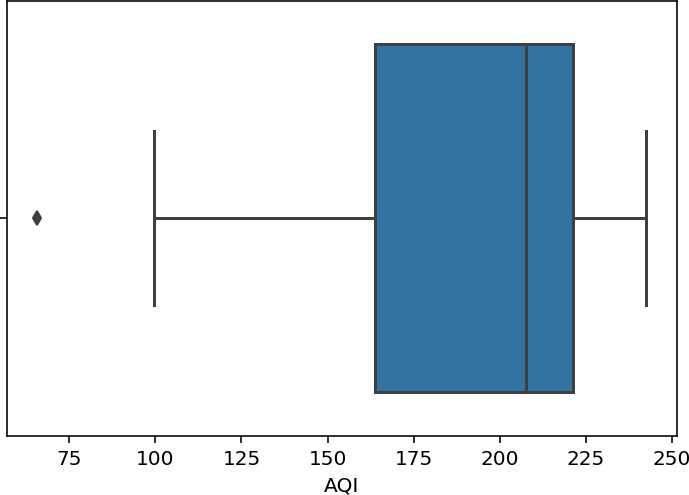
Out[26]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 [27]:

*#Appling BOXPLOT analysis df* = data[['AQI','year']].groupby(["year"]).mean ().reset\_index().sort\_values(by='year',asce nding=False) df=df.dropna() dd=df df.describe()

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | |
| import seaborn as sns sns.boxplot(x=df['AQI']) | | | | In [28]: | |
| <matplotlib.axes.\_subplots.AxesSubplot at 0x7f72c268cd30> | | | | Out[28]: | |



In [29]:

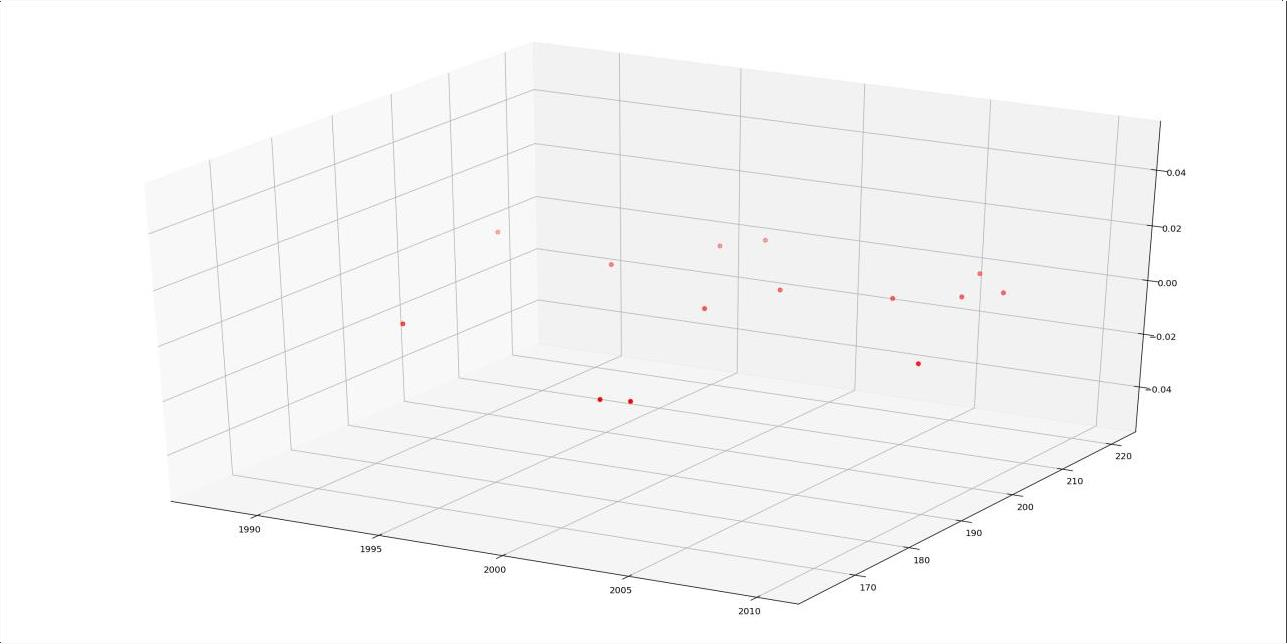
*#removing Outliers df* = df[np.isfinite(df['AQI'])] df=df[df.AQI >153] df=df[df.AQI <221]

In [30]:

*#visualizing the filttered data*

*year*=df['year'].values AQI=df['AQI'].values df['AQI']=pd.to\_numeric(df['AQI'],errors='coerce') df['year']=pd.to\_numeric(df['year'],errors='coerce')

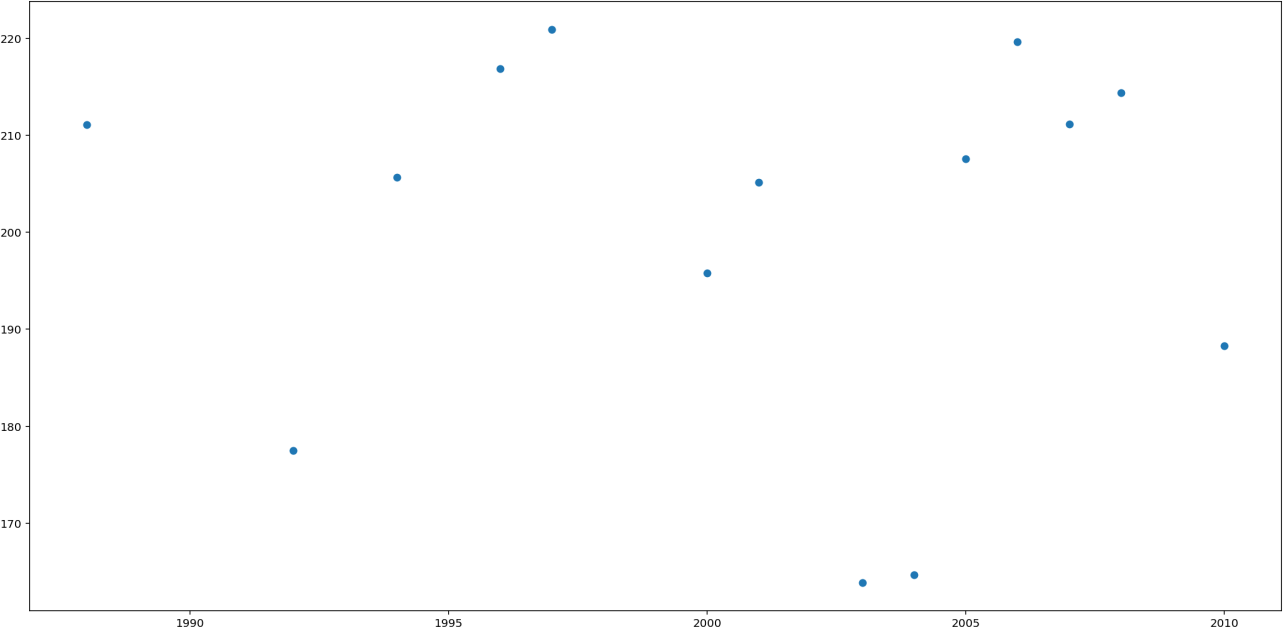
import matplotlib.pyplot as plt plt.rcParams['figure.figsize'] = (20.0, 10.0) from mpl\_toolkits.mplot3d import Axes3D fig = plt.figure() ax = Axes3D(fig) ax.scatter(year,AQI, color='red') plt.show()



In [31]:

*#scatter plot of data points cols* =['year'] y = df['AQI'] x=df[cols]

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



In [32]: x = (x - x.mean()) / x.std() x = np.c\_[np.ones(x.shape[0]), x] x

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | Out[32]: |
| 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. |  | , -0.11967512], |  |  |
| [ 1. |  | , -0.57661648], |  |  |
| [ 1. |  | , -0.72893027], |  |  |
| [ 1. |  | , -1.03355785], |  |  |
| [ 1. |  | , -1.33818543], |  |  |
| [ 1. |  | , -1.94744058]]) |  |  |
|  |  |  |  |  |

In [33]:

*# 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)

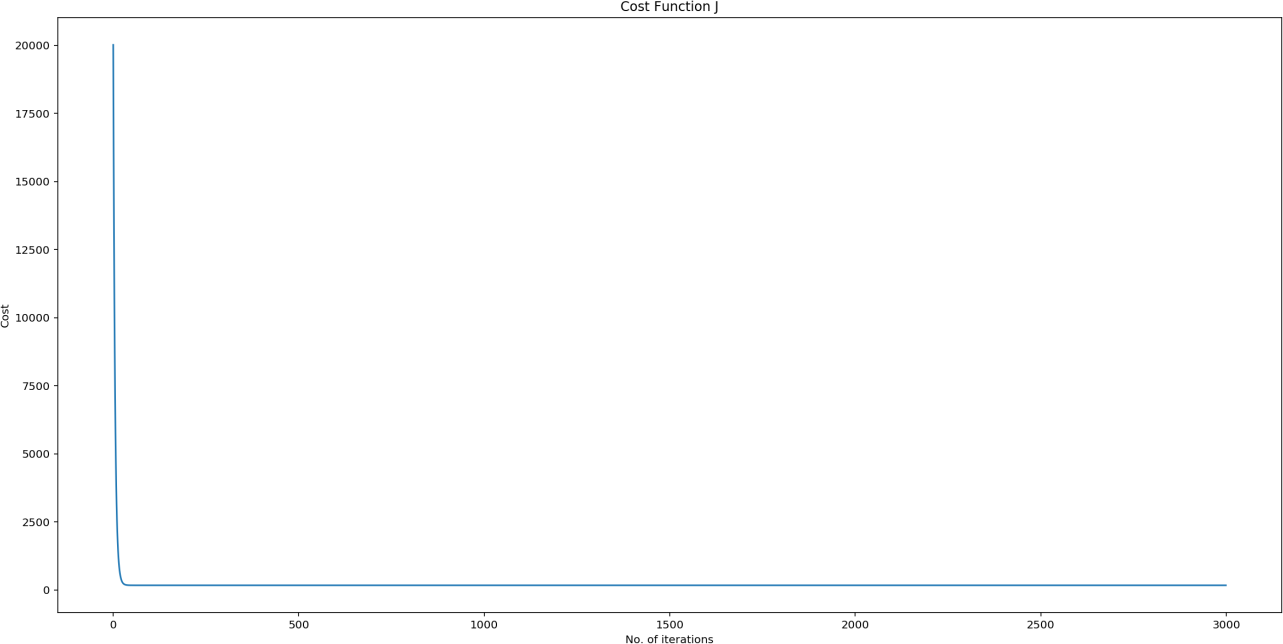
return past\_thetas, past\_costs

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

In [34]: *#Plotting the cost function... plt*.title('Cost Function J') plt.xlabel('No. of iterations') plt.ylabel('Cost') plt.plot(past\_costs) plt.show()



In [35]:

*#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([df, dt], axis=1) x x

Out[35]:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 |
| 5 | 1992 | 177.485106 | 177.485106 | 202.230806 |
| 1 | 1988 | 211.076502 | 211.076502 | 203.169058 |

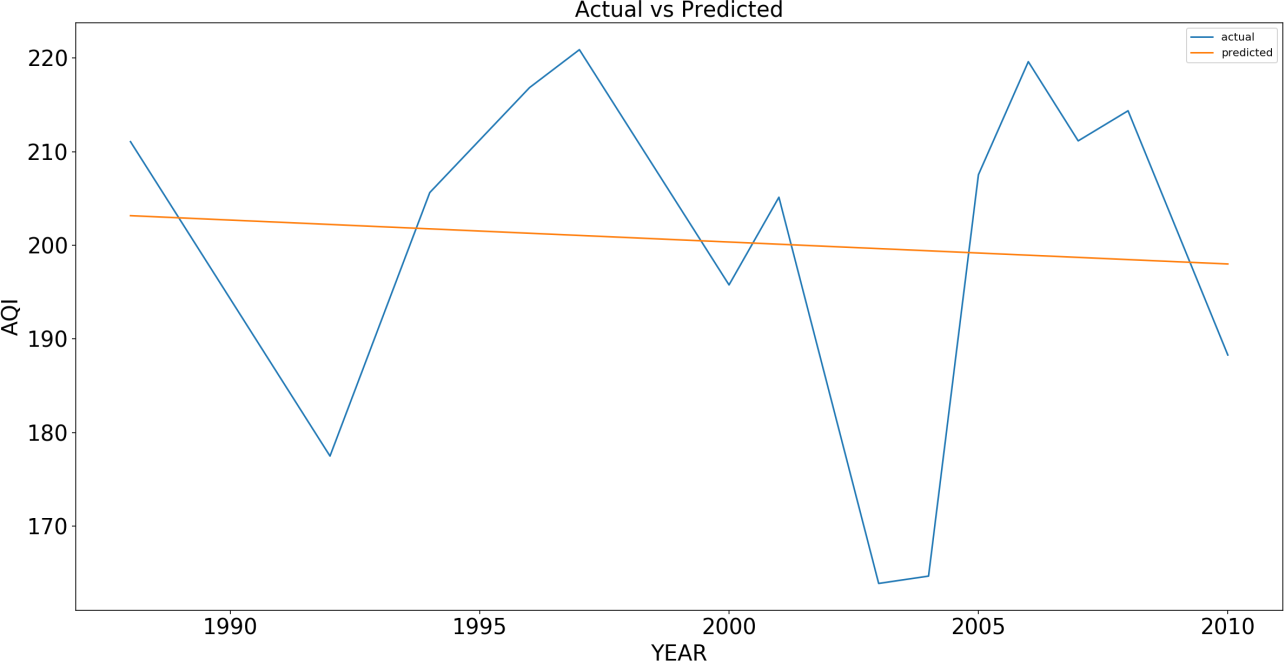
In [36]:

*#calculating the root mean squared error for the predicted AQi values from* sklearn import metrics

print(np.sqrt(metrics.mean\_squared\_error(y,y\_pred)))

18.630885273104628

In [37]: 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 [38]:

*#improving the accuracy by splitting the data on heavy variations*

*df*=dd[['year','AQI']]

#huge variations aqi accures on year 2009-2010 (by moving average graph) df=df[df.year<2011] df.describe()

Out[38]:

|  |  |  |
| --- | --- | --- |
|  | 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 |

CONCLUTION

In conclusion, ambient air pollution is a health hazard. It is a global challenge, as evidence shows that adverse effects still exist even at relatively low air pollutant concentrations, and so no threshold values for classical air pollutants can be established based on the available data.