import numpy as np # linear algebra  
import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  
import os  
import matplotlib.pyplot as plt  
%matplotlib inline  
import seaborn as sns  
import warnings  
warnings.filterwarnings('ignore')  
print(os.listdir("../input"))

['india-air-quality-data']

aq=pd.read\_csv('../input/india-air-quality-data/data.csv',encoding="ISO-8859-1")  
aq.tail(5)  
#Data from years 1987-2015

stn\_code sampling\_date state location \  
435737 SAMP 24-12-15 West Bengal ULUBERIA   
435738 SAMP 29-12-15 West Bengal ULUBERIA   
435739 NaN NaN andaman-and-nicobar-islands NaN   
435740 NaN NaN Lakshadweep NaN   
435741 NaN NaN Tripura NaN   
  
 agency type so2 no2 rspm \  
435737 West Bengal State Pollution Control Board RIRUO 22.0 50.0 143.0   
435738 West Bengal State Pollution Control Board RIRUO 20.0 46.0 171.0   
435739 NaN NaN NaN NaN NaN   
435740 NaN NaN NaN NaN NaN   
435741 NaN NaN NaN NaN NaN   
  
 spm location\_monitoring\_station pm2\_5 date   
435737 NaN Inside Rampal Industries,ULUBERIA NaN 2015-12-24   
435738 NaN Inside Rampal Industries,ULUBERIA NaN 2015-12-29   
435739 NaN NaN NaN NaN   
435740 NaN NaN NaN NaN   
435741 NaN NaN NaN NaN

aq.shape

(435742, 13)

# Extracting Tamil Nadu state data alone  
tn = aq.query('state=="Tamil Nadu" ')  
tn.sample(2)

stn\_code sampling\_date state location \  
355251 159.0 24/07/2012 Tamil Nadu Chennai   
353962 237.0 7/5/2011 Tamil Nadu Coimbatore   
  
 agency \  
355251 National Environmental Engineering Research In...   
353962 Tamilnadu State Pollution Control Board   
  
 type so2 no2 rspm spm \  
355251 Residential, Rural and other Areas 10.0 19.0 22.0 NaN   
353962 Industrial Area 4.0 15.0 61.0 NaN   
  
 location\_monitoring\_station pm2\_5 date   
355251 Madras Medical College, Chennai NaN 2012-07-24   
353962 SIDCO Office, Coimbatore NaN 2011-05-07

tn.shape

(20597, 13)

tn.describe(include = 'all')

stn\_code sampling\_date state location \  
count 14539.0 20597 20597 20597   
unique 48.0 3559 1 11   
top 309.0 28-02-13 Tamil Nadu Chennai   
freq 811.0 17 20597 6646   
mean NaN NaN NaN NaN   
std NaN NaN NaN NaN   
min NaN NaN NaN NaN   
25% NaN NaN NaN NaN   
50% NaN NaN NaN NaN   
75% NaN NaN NaN NaN   
max NaN NaN NaN NaN   
  
 agency \  
count 14133   
unique 4   
top Tamilnadu State Pollution Control Board   
freq 11498   
mean NaN   
std NaN   
min NaN   
25% NaN   
50% NaN   
75% NaN   
max NaN   
  
 type so2 no2 \  
count 20243 19906.000000 19981.000000   
unique 6 NaN NaN   
top Residential, Rural and other Areas NaN NaN   
freq 9033 NaN NaN   
mean NaN 11.315134 21.601202   
std NaN 9.790730 11.034707   
min NaN 0.000000 0.000000   
25% NaN 6.900000 15.300000   
50% NaN 10.000000 20.600000   
75% NaN 14.000000 25.100000   
max NaN 909.000000 315.000000   
  
 rspm spm location\_monitoring\_station \  
count 18792.000000 9530.000000 18961   
unique NaN NaN 49   
top NaN NaN Sowdeswari College Building, Salem   
freq NaN NaN 772   
mean 66.585638 126.729064 NaN   
std 44.450037 81.060905 NaN   
min 3.000000 0.000000 NaN   
25% 39.500000 76.000000 NaN   
50% 55.000000 108.000000 NaN   
75% 82.000000 156.875000 NaN   
max 1183.500000 1682.000000 NaN   
  
 pm2\_5 date   
count 454.000000 20597   
unique NaN 3559   
top NaN 2013-02-28   
freq NaN 17   
mean 29.550441 NaN   
std 16.783704 NaN   
min 4.000000 NaN   
25% 18.000000 NaN   
50% 25.000000 NaN   
75% 36.000000 NaN   
max 141.000000 NaN

# Feature Engineering

##### Removing unnecessary datas

tn.drop(labels=['stn\_code','sampling\_date','agency','location\_monitoring\_station'], axis = 1, inplace = True)  
tn.sample(2)

state location type so2 no2 \  
356319 Tamil Nadu Trichy Residential, Rural and other Areas 10.0 17.0   
360456 Tamil Nadu Cuddalore Residential, Rural and other Areas 10.0 22.0   
  
 rspm spm pm2\_5 date   
356319 46.0 NaN NaN 2012-05-12   
360456 90.0 NaN NaN 2014-10-02

tn.isnull().sum()

state 0  
location 0  
type 354  
so2 691  
no2 616  
rspm 1805  
spm 11067  
pm2\_5 20143  
date 0  
dtype: int64

pm2\_5 has almost 97% data missing. So omitting pm2\_5 column

tn.drop(labels = ['pm2\_5'], axis =1, inplace = True)  
tn.head(2)

state location type so2 no2 rspm spm date  
343639 Tamil Nadu Madras Industrial 0.5 0.3 NaN 82.0 1987-01-27  
343640 Tamil Nadu Madras Industrial 12.9 1.3 NaN 290.0 1987-04-13

In order to fill the missing values, the values are first need to be sorted in Chronological order

tn.dtypes

state object  
location object  
type object  
so2 float64  
no2 float64  
rspm float64  
spm float64  
date object  
dtype: object

# To sort based on dates, the date should be of "datetime" datatype.   
#So converting "object" data type to "datetime" datatype

tn['date'] = pd.to\_datetime(tn.date,format='%Y-%m-%d')  
tn.info()

<class 'pandas.core.frame.DataFrame'>  
Int64Index: 20597 entries, 343639 to 364235  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 state 20597 non-null object   
 1 location 20597 non-null object   
 2 type 20243 non-null object   
 3 so2 19906 non-null float64   
 4 no2 19981 non-null float64   
 5 rspm 18792 non-null float64   
 6 spm 9530 non-null float64   
 7 date 20597 non-null datetime64[ns]  
dtypes: datetime64[ns](1), float64(4), object(3)  
memory usage: 1.4+ MB

tn.sort\_values(by='date')

state location type so2 \  
343641 Tamil Nadu Madras Industrial 38.8   
343642 Tamil Nadu Madras Industrial 29.7   
343643 Tamil Nadu Madras Industrial 27.5   
343656 Tamil Nadu Madras Residential 4.0   
343655 Tamil Nadu Madras Residential 8.3   
... ... ... ... ...   
363476 Tamil Nadu Cuddalore Residential, Rural and other Areas 12.0   
363989 Tamil Nadu Chennai Residential, Rural and other Areas 10.0   
362831 Tamil Nadu Madurai Residential, Rural and other Areas 9.0   
362605 Tamil Nadu Coimbatore Residential, Rural and other Areas 4.0   
363906 Tamil Nadu Chennai Residential, Rural and other Areas 12.0   
  
 no2 rspm spm date   
343641 0.9 NaN 222.0 1987-01-05   
343642 1.1 NaN 213.0 1987-01-06   
343643 1.3 NaN 223.0 1987-01-07   
343656 1.0 NaN 116.0 1987-01-12   
343655 0.8 NaN 121.0 1987-01-12   
... ... ... ... ...   
363476 18.0 53.0 NaN 2015-12-31   
363989 14.0 123.0 NaN 2015-12-31   
362831 23.0 58.0 NaN 2015-12-31   
362605 27.0 88.0 NaN 2015-12-31   
363906 31.0 234.0 NaN 2015-12-31   
  
[20597 rows x 8 columns]

tn['so2'].fillna(method='ffill',inplace = True);  
tn['no2'].fillna(method='ffill',inplace = True);  
tn['rspm'].fillna(method='ffill',inplace = True);  
tn['spm'].fillna(method='ffill',inplace = True);

tn.isnull().sum()

state 0  
location 0  
type 354  
so2 0  
no2 0  
rspm 1636  
spm 0  
date 0  
dtype: int64

# Even after replacement, we have 1636 missing values in rspm  
print(tn.iloc[[1634]],tn.iloc[[1635]],tn.iloc[[1636]],tn.iloc[[1636]])

state location type so2 no2 rspm spm date  
345273 Tamil Nadu Madurai Industrial Area 8.3 19.6 NaN 33.0 2002-11-01 state location type so2 no2 rspm spm \  
345274 Tamil Nadu Madurai Industrial Area 10.7 29.6 NaN 63.0   
  
 date   
345274 2002-12-01 state location type so2 no2 \  
345275 Tamil Nadu Chennai Residential, Rural and other Areas 6.0 12.75   
  
 rspm spm date   
345275 56.33 101.33 2004-02-01 state location type so2 no2 \  
345275 Tamil Nadu Chennai Residential, Rural and other Areas 6.0 12.75   
  
 rspm spm date   
345275 56.33 101.33 2004-02-01

# This means rspm has not been calculated till 2002. It has been measured only from 2004 onwards.  
# We can either omit it or seperate the data set into two. That is before 2004 and after 2004.  
# Here for simplicity, I am deleting the column of rspm

tn.drop(labels = ['rspm'], axis = 1, inplace = True)  
tn.head()

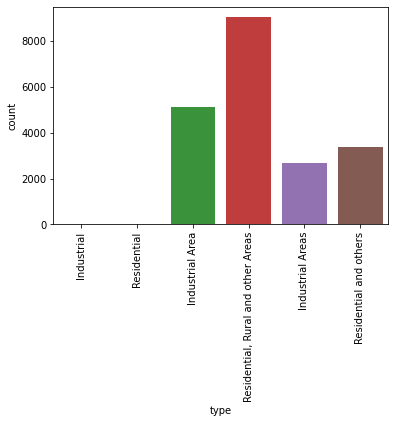
state location type so2 no2 spm date  
343639 Tamil Nadu Madras Industrial 0.5 0.3 82.0 1987-01-27  
343640 Tamil Nadu Madras Industrial 12.9 1.3 290.0 1987-04-13  
343641 Tamil Nadu Madras Industrial 38.8 0.9 222.0 1987-01-05  
343642 Tamil Nadu Madras Industrial 29.7 1.1 213.0 1987-01-06  
343643 Tamil Nadu Madras Industrial 27.5 1.3 223.0 1987-01-07

tn.isnull().sum()

state 0  
location 0  
type 354  
so2 0  
no2 0  
spm 0  
date 0  
dtype: int64

# Dealing with 354 missing values of type

typ=sns.countplot(x ="type",data = tn)  
typ.set\_xticklabels(typ.get\_xticklabels(), rotation=90);



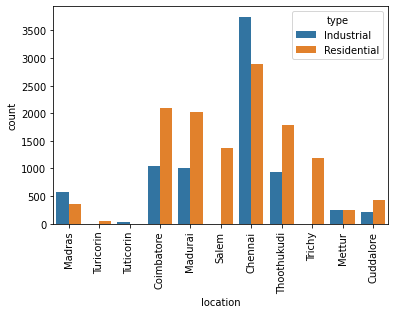
Here we have repetition of types, so replacing all to unique types

tn['type'].replace("Industrial Areas","Industrial",inplace=True)  
tn['type'].replace("Industrial Area","Industrial",inplace=True)  
tn['type'].replace("Residential and others","Residential",inplace=True)  
tn['type'].replace("Residential, Rural and other Areas","Residential",inplace=True)

typ=sns.countplot(x ="type",data = tn)  
typ.set\_xticklabels(typ.get\_xticklabels(), rotation=90);



datacount\_ty =sns.countplot(x ="location",hue = 'type',data = tn);  
datacount\_ty.set\_xticklabels(datacount\_ty.get\_xticklabels(), rotation=90);



# Rows with missing "types"  
null\_data = tn[tn.isnull().any(axis=1)]  
null\_data.head(20)

state location type so2 no2 spm date  
343962 Tamil Nadu Tuticorin NaN 10.2 16.7 194.0 1991-09-01  
343984 Tamil Nadu Tuticorin NaN 7.2 16.4 82.0 1991-12-01  
343985 Tamil Nadu Tuticorin NaN 8.2 5.4 47.0 1991-12-01  
343986 Tamil Nadu Tuticorin NaN 8.0 12.4 44.0 1991-12-01  
344142 Tamil Nadu Tuticorin NaN 3.7 17.8 80.0 1994-09-01  
344143 Tamil Nadu Tuticorin NaN 6.4 20.9 68.0 1994-09-01  
344147 Tamil Nadu Tuticorin NaN 3.7 17.8 80.0 1994-09-01  
344148 Tamil Nadu Tuticorin NaN 6.4 20.9 68.0 1994-09-01  
344155 Tamil Nadu Coimbatore NaN 6.0 16.0 55.0 1995-01-01  
344156 Tamil Nadu Coimbatore NaN 0.9 9.2 44.0 1995-03-01  
344157 Tamil Nadu Coimbatore NaN 5.1 15.8 78.0 1995-02-01  
344158 Tamil Nadu Coimbatore NaN 3.9 13.5 71.0 1995-03-01  
344159 Tamil Nadu Coimbatore NaN 2.3 9.1 56.0 1995-04-01  
344160 Tamil Nadu Coimbatore NaN 2.2 5.5 42.0 1995-05-01  
344161 Tamil Nadu Coimbatore NaN 1.4 4.9 31.0 1995-06-01  
344162 Tamil Nadu Coimbatore NaN 2.3 6.9 29.0 1995-07-01  
344163 Tamil Nadu Coimbatore NaN 3.7 8.4 51.0 1995-08-01  
344164 Tamil Nadu Coimbatore NaN 2.3 5.3 40.0 1995-09-01  
344165 Tamil Nadu Coimbatore NaN 3.1 5.9 33.0 1995-10-01  
344166 Tamil Nadu Coimbatore NaN 5.1 6.5 41.0 1995-11-01

Mode is higher for residential. So filling the missing 354 values in type by "Residential" type

# Converting NaN to zeros  
#df['DataFrame Column'] = df['DataFrame Column'].replace(np.nan, 0)  
tn['type'] = tn['type'].replace(np.nan, "Residential")

tn.isnull().sum()

state 0  
location 0  
type 0  
so2 0  
no2 0  
spm 0  
date 0  
dtype: int64

#Finding hidden missing values. (i.e. zeros)

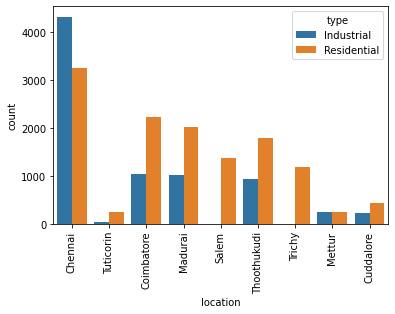
aaa = (tn == 0).astype(int).sum(axis=0)  
print(aaa)

state 0  
location 0  
type 0  
so2 16  
no2 1  
spm 66  
date 0  
dtype: int64

# Also we can see the "locations" repeated.  
# Madras - Chennai, # Turicorin-Tuticorin  
# Replacing them into single value

tn['location'].replace("Turicorin","Tuticorin",inplace=True)  
tn['location'].replace("Madras","Chennai",inplace=True)

datacount\_ty =sns.countplot(x ="location",hue = 'type',data = tn);  
datacount\_ty.set\_xticklabels(datacount\_ty.get\_xticklabels(), rotation=90);

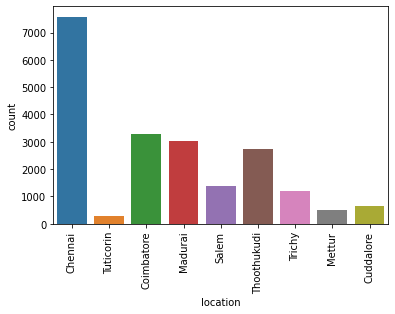


tn.head()

state location type so2 no2 spm date  
343639 Tamil Nadu Chennai Industrial 0.5 0.3 82.0 1987-01-27  
343640 Tamil Nadu Chennai Industrial 12.9 1.3 290.0 1987-04-13  
343641 Tamil Nadu Chennai Industrial 38.8 0.9 222.0 1987-01-05  
343642 Tamil Nadu Chennai Industrial 29.7 1.1 213.0 1987-01-06  
343643 Tamil Nadu Chennai Industrial 27.5 1.3 223.0 1987-01-07

# Data Visualization

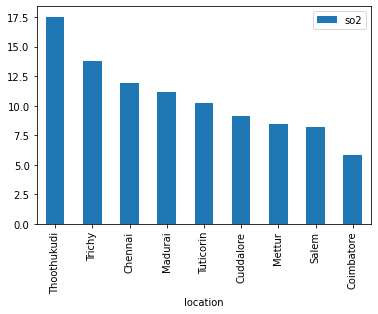
datacount =sns.countplot(x ="location",data = tn);  
datacount.set\_xticklabels(datacount.get\_xticklabels(), rotation=90);



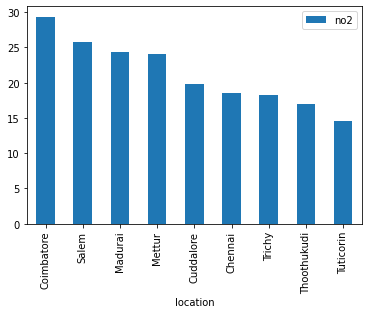
loc = pd.pivot\_table(tn, values=['so2','no2','spm'],index='location') # Aggfunc: default-np.mean()  
loc

no2 so2 spm  
location   
Chennai 18.551330 11.905157 199.767056  
Coimbatore 29.374767 5.832845 188.888683  
Cuddalore 19.772657 9.110599 267.000000  
Madurai 24.420616 11.153280 179.156298  
Mettur 24.039095 8.399177 267.000000  
Salem 25.764407 8.190645 179.440385  
Thoothukudi 16.948928 17.532772 210.858009  
Trichy 18.211327 13.753170 267.000000  
Tuticorin 14.505208 10.176389 51.322917

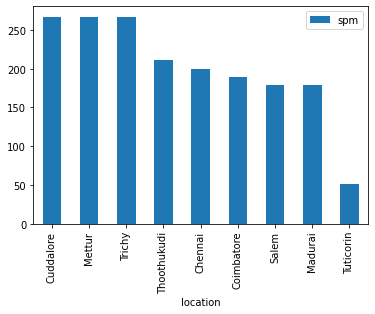
maxso2 = loc.sort\_values(by='so2',ascending=False)  
maxso2.loc[:,['so2']].head(10).plot(kind='bar'); # Based on average values



maxno2 = loc.sort\_values(by='no2',ascending=False);  
maxno2.loc[:,['no2']].head(10).plot(kind='bar');



maxspm = loc.sort\_values(by='spm',ascending=False);  
maxspm.loc[:,['spm']].head(10).plot(kind='bar');



# Calculating AQI

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  
tn['si']=tn['so2'].apply(calculate\_si)  
ds= tn[['so2','si']]  
ds.tail()

so2 si  
364231 13.0 s1  
364232 13.0 s1  
364233 14.0 s1  
364234 15.0 s1  
364235 15.0 s1

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  
tn['ni']=tn['no2'].apply(calculate\_ni)  
dn= tn[['no2','ni']]  
dn.tail()

no2 ni  
364231 22.0 n1  
364232 22.0 n1  
364233 24.0 n1  
364234 25.0 n1  
364235 24.0 n1

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  
tn['spi']=tn['spm'].apply(calculate\_spi)  
dsp= tn[['spm','spi']]  
dsp.tail()

spm spi  
364231 267.0 sp4  
364232 267.0 sp4  
364233 267.0 sp4  
364234 267.0 sp4  
364235 267.0 sp4

tn.sample(2)

state location type so2 no2 spm date si \  
363100 Tamil Nadu Thoothukudi Residential 9.0 12.0 267.0 2015-01-13 s1   
355369 Tamil Nadu Chennai Residential 4.0 15.0 267.0 2012-10-15 s1   
  
 ni spi   
363100 n1 sp4   
355369 n1 sp4

# 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  
tn['AQI']=tn.apply(lambda x:calculate\_aqi(x['so2'],x['no2'],x['spm']),axis=1)

tn.head()

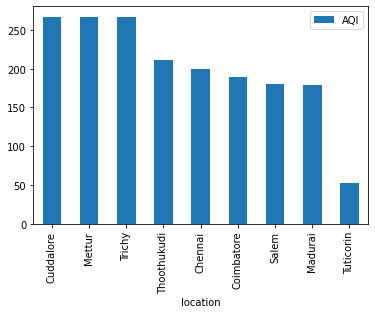
state location type so2 no2 spm date si ni \  
343639 Tamil Nadu Chennai Industrial 0.5 0.3 82.0 1987-01-27 s1 n1   
343640 Tamil Nadu Chennai Industrial 12.9 1.3 290.0 1987-04-13 s1 n1   
343641 Tamil Nadu Chennai Industrial 38.8 0.9 222.0 1987-01-05 s1 n1   
343642 Tamil Nadu Chennai Industrial 29.7 1.1 213.0 1987-01-06 s1 n1   
343643 Tamil Nadu Chennai Industrial 27.5 1.3 223.0 1987-01-07 s1 n1   
  
 spi AQI   
343639 sp3 82.0   
343640 sp5 290.0   
343641 sp4 222.0   
343642 sp4 213.0   
343643 sp4 223.0

aq\_wise = pd.pivot\_table(tn, values=['AQI'],index='location')  
aq\_wise

AQI  
location   
Chennai 200.055794  
Coimbatore 189.199613  
Cuddalore 267.000000  
Madurai 179.283224  
Mettur 267.000000  
Salem 179.550399  
Thoothukudi 210.887068  
Trichy 267.000000  
Tuticorin 52.573958

maxaqi = aq\_wise.sort\_values(by='AQI',ascending=False)  
maxaqi.loc[:,['AQI']].head(37).plot(kind='bar')

<AxesSubplot:xlabel='location'>

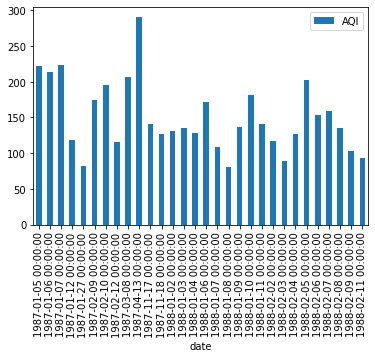


date\_wise = pd.pivot\_table(tn, values=['AQI'],index='date')  
date\_wise

AQI  
date   
1987-01-05 222.0  
1987-01-06 213.0  
1987-01-07 223.0  
1987-01-12 118.5  
1987-01-27 82.0  
... ...  
2015-12-26 267.0  
2015-12-28 267.0  
2015-12-29 267.0  
2015-12-30 267.0  
2015-12-31 267.0  
  
[3559 rows x 1 columns]

date\_wise.loc[:,['AQI']].head(30).plot(kind='bar')

<AxesSubplot:xlabel='date'>



# Training Data

dum1 = pd.get\_dummies(tn['type'])  
dum2 = pd.get\_dummies(tn['location'])  
tn['year'] = tn['date'].dt.year

td = pd.concat([tn, dum1, dum2], axis = 1)  
td.head()

state location type so2 no2 spm date si ni \  
343639 Tamil Nadu Chennai Industrial 0.5 0.3 82.0 1987-01-27 s1 n1   
343640 Tamil Nadu Chennai Industrial 12.9 1.3 290.0 1987-04-13 s1 n1   
343641 Tamil Nadu Chennai Industrial 38.8 0.9 222.0 1987-01-05 s1 n1   
343642 Tamil Nadu Chennai Industrial 29.7 1.1 213.0 1987-01-06 s1 n1   
343643 Tamil Nadu Chennai Industrial 27.5 1.3 223.0 1987-01-07 s1 n1   
  
 spi ... Residential Chennai Coimbatore Cuddalore Madurai \  
343639 sp3 ... 0 1 0 0 0   
343640 sp5 ... 0 1 0 0 0   
343641 sp4 ... 0 1 0 0 0   
343642 sp4 ... 0 1 0 0 0   
343643 sp4 ... 0 1 0 0 0   
  
 Mettur Salem Thoothukudi Trichy Tuticorin   
343639 0 0 0 0 0   
343640 0 0 0 0 0   
343641 0 0 0 0 0   
343642 0 0 0 0 0   
343643 0 0 0 0 0   
  
[5 rows x 23 columns]

td.drop(labels = ['state','location','type','so2','no2','spm','si','ni','spi','date'], axis = 1, inplace = True)  
td.sample(2)

AQI year Industrial Residential Chennai Coimbatore Cuddalore \  
350723 102.0 2008 0 1 0 0 0   
354333 267.0 2011 1 0 0 0 0   
  
 Madurai Mettur Salem Thoothukudi Trichy Tuticorin   
350723 0 0 0 1 0 0   
354333 1 0 0 0 0 0

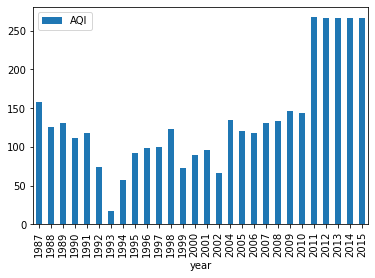
td.corr()

AQI year Industrial Residential Chennai \  
AQI 1.000000 0.646473 0.057981 -0.057981 -0.006406   
year 0.646473 1.000000 -0.085917 0.085917 -0.123071   
Industrial 0.057981 -0.085917 1.000000 -1.000000 0.300520   
Residential -0.057981 0.085917 -1.000000 1.000000 -0.300520   
Chennai -0.006406 -0.123071 0.300520 -0.300520 1.000000   
Coimbatore -0.056296 -0.056847 -0.054400 0.054400 -0.331489   
Cuddalore 0.133554 0.158258 -0.017697 0.017697 -0.137904   
Madurai -0.099989 0.011918 -0.038487 0.038487 -0.317511   
Mettur 0.114920 0.138348 0.038661 -0.038661 -0.118663   
Salem -0.063568 0.015006 -0.209332 0.209332 -0.204397   
Thoothukudi 0.043930 0.047736 -0.027929 0.027929 -0.297876   
Trichy 0.182486 0.186706 -0.192979 0.192979 -0.188430   
Tuticorin -0.197143 -0.307805 -0.064122 0.064122 -0.090901   
  
 Coimbatore Cuddalore Madurai Mettur Salem Thoothukudi \  
AQI -0.056296 0.133554 -0.099989 0.114920 -0.063568 0.043930   
year -0.056847 0.158258 0.011918 0.138348 0.015006 0.047736   
Industrial -0.054400 -0.017697 -0.038487 0.038661 -0.209332 -0.027929   
Residential 0.054400 0.017697 0.038487 -0.038661 0.209332 0.027929   
Chennai -0.331489 -0.137904 -0.317511 -0.118663 -0.204397 -0.297876   
Coimbatore 1.000000 -0.078454 -0.180633 -0.067508 -0.116282 -0.169463   
Cuddalore -0.078454 1.000000 -0.075146 -0.028084 -0.048375 -0.070499   
Madurai -0.180633 -0.075146 1.000000 -0.064661 -0.111379 -0.162317   
Mettur -0.067508 -0.028084 -0.064661 1.000000 -0.041626 -0.060663   
Salem -0.116282 -0.048375 -0.111379 -0.041626 1.000000 -0.104491   
Thoothukudi -0.169463 -0.070499 -0.162317 -0.060663 -0.104491 1.000000   
Trichy -0.107199 -0.044596 -0.102678 -0.038374 -0.066099 -0.096329   
Tuticorin -0.051714 -0.021514 -0.049533 -0.018512 -0.031887 -0.046470   
  
 Trichy Tuticorin   
AQI 0.182486 -0.197143   
year 0.186706 -0.307805   
Industrial -0.192979 -0.064122   
Residential 0.192979 0.064122   
Chennai -0.188430 -0.090901   
Coimbatore -0.107199 -0.051714   
Cuddalore -0.044596 -0.021514   
Madurai -0.102678 -0.049533   
Mettur -0.038374 -0.018512   
Salem -0.066099 -0.031887   
Thoothukudi -0.096329 -0.046470   
Trichy 1.000000 -0.029396   
Tuticorin -0.029396 1.000000

"year" has good correlation with "AQI" when compared to others

yr\_wise = pd.pivot\_table(td, values=['AQI'],index='year')  
yr\_wise.loc[:,['AQI']].head(30).plot(kind='bar')

<AxesSubplot:xlabel='year'>



from sklearn.model\_selection import train\_test\_split

X=td.drop("AQI",axis=1)  
y=td["AQI"]

X\_train,X\_test,y\_train,y\_test= train\_test\_split(X,y,test\_size=0.30,random\_state=25)

# Model fittings

### Simple Linear Regression

from sklearn.linear\_model import LinearRegression

lin\_mod = LinearRegression()  
lin\_mod.fit(X\_train, y\_train)

LinearRegression()

lin\_mod.score(X\_train, y\_train )

0.4453601500506762

lin\_mod.score(X\_test, y\_test)

0.46740661107915094

# Less Score. Underfitting

### Polynomial Regression

from sklearn.preprocessing import PolynomialFeatures  
from sklearn import linear\_model  
  
poly = PolynomialFeatures(degree=2, interaction\_only=True)  
X\_train2 = poly.fit\_transform(X\_train)  
X\_test2 = poly.fit\_transform(X\_test)  
  
poly\_clf = linear\_model.LinearRegression()  
  
poly\_clf.fit(X\_train2, y\_train)  
  
y\_pred = poly\_clf.predict(X\_test2)

print(poly\_clf.score(X\_train2, y\_train))

0.5060498134379463

print(poly\_clf.score(X\_test2, y\_test))

0.5262979517606676

# Trying with higher degrees

poly = PolynomialFeatures(degree=3, interaction\_only=True)  
X\_train2 = poly.fit\_transform(X\_train)  
X\_test2 = poly.fit\_transform(X\_test)  
  
poly\_clf = linear\_model.LinearRegression()  
  
poly\_clf.fit(X\_train2, y\_train)  
  
y\_pred = poly\_clf.predict(X\_test2)  
print(poly\_clf.score(X\_train2, y\_train))  
print(poly\_clf.score(X\_test2, y\_test))

0.5063818171823522  
0.5282058458861778

# degree = 3 has less scores than degree = 2

poly = PolynomialFeatures(degree=4, interaction\_only=True)  
X\_train2 = poly.fit\_transform(X\_train)  
X\_test2 = poly.fit\_transform(X\_test)  
  
poly\_clf = linear\_model.LinearRegression()  
  
poly\_clf.fit(X\_train2, y\_train)  
  
y\_pred = poly\_clf.predict(X\_test2)  
print(poly\_clf.score(X\_train2, y\_train))  
print(poly\_clf.score(X\_test2, y\_test))

0.4956914409476182  
0.5164060494757956

# Nearly score to degree = 2. But still less than degree = 2

poly = PolynomialFeatures(degree=5, interaction\_only=True)  
X\_train2 = poly.fit\_transform(X\_train)  
X\_test2 = poly.fit\_transform(X\_test)  
  
poly\_clf = linear\_model.LinearRegression()  
  
poly\_clf.fit(X\_train2, y\_train)  
  
y\_pred = poly\_clf.predict(X\_test2)  
print(poly\_clf.score(X\_train2, y\_train))  
print(poly\_clf.score(X\_test2, y\_test))

0.4956914409476182  
0.5164060494757956

# Score reduces as degree increases

poly = PolynomialFeatures(degree=6, interaction\_only=True)  
X\_train2 = poly.fit\_transform(X\_train)  
X\_test2 = poly.fit\_transform(X\_test)  
  
poly\_clf = linear\_model.LinearRegression()  
  
poly\_clf.fit(X\_train2, y\_train)  
  
y\_pred = poly\_clf.predict(X\_test2)  
print(poly\_clf.score(X\_train2, y\_train))  
print(poly\_clf.score(X\_test2, y\_test))

0.4956914409476182  
0.5164060494757956

# Same score as prev degree.   
#Underfitting

### K-Nearest Neighbour

from sklearn.neighbors import KNeighborsRegressor  
from scipy.stats import zscore

XScaled = X.apply(zscore)

NNH = KNeighborsRegressor(n\_neighbors = 27, metric = 'euclidean')

NNH.fit(X\_train,y\_train)

KNeighborsRegressor(metric='euclidean', n\_neighbors=27)

predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7239080609538029  
0.7706168194875633

Score Better than Linear Regression models. Trying with different n\_neighbours

NNH = KNeighborsRegressor(n\_neighbors = 30) # default metric = 'minkowski'  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.723809498088116  
0.7719586808848092

NNH = KNeighborsRegressor(n\_neighbors = 55)  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7251509518338644  
0.7705163784261739

NNH = KNeighborsRegressor(n\_neighbors = 70)  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7240609430015734  
0.7685054030238094

# if we increase n\_neigbours more than 55,   
#train fitting increases but test fit decreases. So 55 is the optimum one

Trying with different "metrics"

NNH = KNeighborsRegressor(n\_neighbors = 55, metric = 'euclidean')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7251509518338644  
0.7705163784261739

# Better result dan "minskowki"

NNH = KNeighborsRegressor(n\_neighbors = 35, metric = 'euclidean')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7235162915287796  
0.7722764205003013

NNH = KNeighborsRegressor(n\_neighbors = 30, metric = 'euclidean')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.723809498088116  
0.7719586808848092

35 is the optimum one

NNH = KNeighborsRegressor(n\_neighbors = 35, metric = 'manhattan')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7238308231819853  
0.7723429772126563

# Better than euclidean

NNH = KNeighborsRegressor(n\_neighbors = 45, metric = 'manhattan')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7253762245673958  
0.7726679102657432

NNH = KNeighborsRegressor(n\_neighbors = 55, metric = 'manhattan')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7254198551302349  
0.7706201379619615

# 45 is optimum

# Therefore best solution is for   
NNH = KNeighborsRegressor(n\_neighbors = 45, metric = 'manhattan')  
NNH.fit(X\_train,y\_train)  
predicted\_labels = NNH.predict(X\_test)  
print(NNH.score(X\_train, y\_train))  
print(NNH.score(X\_test,y\_test))

0.7253762245673958  
0.7726679102657432

### SVM

from sklearn import svm  
from sklearn.svm import SVR

reg= svm.SVR(kernel='rbf',gamma='auto', C=2)  
reg.fit(X\_train,y\_train)

SVR(C=2, gamma='auto')

predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

0.6354902391648543  
0.6830055255559128

# Score is less than KNN. Trying with other "C"

reg= svm.SVR(kernel='rbf',gamma='auto', C=150)  
reg.fit(X\_train,y\_train)  
predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

0.6828037711031538  
0.7269869947874135

reg= svm.SVR(kernel='rbf',gamma='auto', C=160)  
reg.fit(X\_train,y\_train)  
predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

0.6829412252671465  
0.7270024553951311

reg= svm.SVR(kernel='rbf',gamma='auto', C=163)  
reg.fit(X\_train,y\_train)  
predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

0.682954354642903  
0.7269849962216359

# as C increases after 160, score training score increases but test score decreases.

reg= svm.SVR(kernel='sigmoid',gamma='auto', C=80)  
reg.fit(X\_train,y\_train)  
predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

-0.526948927297983  
-0.5899658122069891

# Using poly kernel takes lot of time to run

# Optimum value for SVM is  
reg= svm.SVR(kernel='rbf',gamma='auto', C=160)  
reg.fit(X\_train,y\_train)  
predicted\_labels = reg.predict(X\_test)  
print(reg.score(X\_train,y\_train))  
print(reg.score(X\_test,y\_test))

0.6829412252671465  
0.7270024553951311

But not as good as KNN

### Decision Tree

from sklearn.tree import DecisionTreeRegressor

dTree= DecisionTreeRegressor(criterion='mse',splitter='best',random\_state=25,max\_depth=5)

dTree.fit(X\_train,y\_train)

DecisionTreeRegressor(max\_depth=5, random\_state=25)

print(dTree.score(X\_train,y\_train))   
print(dTree.score(X\_test,y\_test))

0.6987590136971868  
0.7490946656981097

Trying with different "max\_depth"

dTree= DecisionTreeRegressor(criterion='mse',splitter='best',random\_state=25,max\_depth=14)  
dTree.fit(X\_train,y\_train)  
print(dTree.score(X\_train,y\_train))   
print(dTree.score(X\_test,y\_test))

0.7320163141352926  
0.7764637553626321

# No improvements in score after "max\_depth = 14"  
# Trying with different criteria

dTree= DecisionTreeRegressor(criterion='mae',splitter='best',random\_state=25,max\_depth=20)  
dTree.fit(X\_train,y\_train)  
print(dTree.score(X\_train,y\_train))   
print(dTree.score(X\_test,y\_test))

0.7152580836801676  
0.7650663853334146

dTree= DecisionTreeRegressor(criterion='friedman\_mse',splitter='best',random\_state=25,max\_depth=15)  
dTree.fit(X\_train,y\_train)  
print(dTree.score(X\_train,y\_train))   
print(dTree.score(X\_test,y\_test))

0.7320320350190757  
0.7764581508742332

# friedman\_mse same as mse

# Optimum is   
dTree= DecisionTreeRegressor(criterion='mse',splitter='best',random\_state=25,max\_depth=14)  
dTree.fit(X\_train,y\_train)  
print(dTree.score(X\_train,y\_train))   
print(dTree.score(X\_test,y\_test))

0.7320163141352926  
0.7764637553626321

dTree= DecisionTreeRegressor(criterion='mse',splitter='best',random\_state=25,max\_depth=14)  
dTree.fit(X\_train,y\_train)  
dTree\_tr=dTree.score(X\_train,y\_train)  
dTree\_ts=dTree.score(X\_test,y\_test)

Better than KNN

### Bagging

from sklearn.ensemble import BaggingRegressor

bgr= BaggingRegressor (n\_estimators=9,base\_estimator=dTree,random\_state=25)  
bgr=bgr.fit(X\_train,y\_train)  
print(bgr.score(X\_train,y\_train))  
print(bgr.score(X\_test,y\_test))

0.7315720767332268  
0.7762732523198246

# trying with different "n\_estimators"

bgr= BaggingRegressor (n\_estimators=12,base\_estimator=dTree,random\_state=25)  
bgr=bgr.fit(X\_train,y\_train)  
print(bgr.score(X\_train,y\_train))  
print(bgr.score(X\_test,y\_test))

0.7317520849084952  
0.7759680322824837

# Increase in "n\_estimators" increases train score but decreases test score.   
#so "n\_estimators = 9" is good

Not as good as Decision Tree

### AdaBoost

from sklearn.ensemble import AdaBoostRegressor

adr= AdaBoostRegressor (n\_estimators=5,random\_state=25, loss ='linear') # loss = 'linear' is default  
adr=adr.fit(X\_train,y\_train)  
print(adr.score(X\_train,y\_train))  
print(adr.score(X\_test,y\_test))

0.5648604617431005  
0.6273545061897812

# trying with different "n\_estimators"

adr= AdaBoostRegressor (n\_estimators=15,random\_state=25,loss ='linear')  
adr=adr.fit(X\_train,y\_train)  
print(adr.score(X\_train,y\_train))  
print(adr.score(X\_test,y\_test))

0.5648604617431005  
0.6273545061897812

# Increase in "n\_estimators" increases train score but decreases test score.   
#so "n\_estimators = 5" is good

adr= AdaBoostRegressor (n\_estimators=7,random\_state=25,loss ='square')  
adr=adr.fit(X\_train,y\_train)  
print(adr.score(X\_train,y\_train))  
print(adr.score(X\_test,y\_test))

0.5806460542188621  
0.6450872934033688

adr= AdaBoostRegressor (n\_estimators=5,random\_state=25,loss ='exponential')  
adr=adr.fit(X\_train,y\_train)  
print(adr.score(X\_train,y\_train))  
print(adr.score(X\_test,y\_test))

0.672733039307424  
0.723678044480647

Not good as Decision Tree

### Gradient Boosting

from sklearn.ensemble import GradientBoostingRegressor

gbr= GradientBoostingRegressor (n\_estimators=10,random\_state=25)  
gbr=gbr.fit(X\_train,y\_train)  
print(gbr.score(X\_train,y\_train))  
print(gbr.score(X\_test,y\_test))

0.5842205537720442  
0.6290894003600735

# trying with different "n\_estimators"

gbr= GradientBoostingRegressor (n\_estimators=400,random\_state=25)  
gbr=gbr.fit(X\_train,y\_train)  
print(gbr.score(X\_train,y\_train))  
print(gbr.score(X\_test,y\_test))

0.7247539388723996  
0.7751318074743498

gbr= GradientBoostingRegressor (n\_estimators=410,random\_state=25)  
gbr=gbr.fit(X\_train,y\_train)  
print(gbr.score(X\_train,y\_train))  
print(gbr.score(X\_test,y\_test))

0.7248060365277629  
0.775075734837311

# Increase in "n\_estimators" beyond 400, increases train score but decreases test score. so "n\_estimators = 400" is good

# Optimum is   
gbr= GradientBoostingRegressor (n\_estimators=400,random\_state=25)  
gbr=gbr.fit(X\_train,y\_train)  
print(gbr.score(X\_train,y\_train))  
print(gbr.score(X\_test,y\_test))

0.7247539388723996  
0.7751318074743498

gbr= GradientBoostingRegressor (n\_estimators=400,random\_state=25)  
gbr=gbr.fit(X\_train,y\_train)  
gbr\_tr= gbr.score(X\_train,y\_train)  
gbr\_ts= gbr.score(X\_test,y\_test)

Very near to Decision Tree.

Score of Decision Tree

train - 0.7320163141352926 test - 0.7764637553626321

### Random Forest

from sklearn.ensemble import RandomForestRegressor

rfr= RandomForestRegressor (n\_estimators=10,random\_state=25,max\_features=5)  
rfr=rfr.fit(X\_train,y\_train)  
print(rfr.score(X\_train,y\_train))  
print(rfr.score(X\_test,y\_test))

0.7316928947040389  
0.7761367960037955

# trying with different "n\_estimators"

rfr= RandomForestRegressor (n\_estimators=11,random\_state=25,max\_features=5)  
rfr=rfr.fit(X\_train,y\_train)  
print(rfr.score(X\_train,y\_train))  
print(rfr.score(X\_test,y\_test))

0.7317456980511001  
0.776008309930724

# No effect

# trying with different "max\_features"

rfr= RandomForestRegressor (n\_estimators=10,random\_state=25,max\_features=10)  
rfr=rfr.fit(X\_train,y\_train)  
print(rfr.score(X\_train,y\_train))  
print(rfr.score(X\_test,y\_test))

0.7317018765899191  
0.7761620422920031

rfr= RandomForestRegressor (n\_estimators=10,random\_state=25,max\_features=10)  
rfr=rfr.fit(X\_train,y\_train)  
rfr\_tr = rfr.score(X\_train,y\_train)  
rfr\_ts = rfr.score(X\_test,y\_test)

# the above one is optimum

Score Very similar to Decision Tree, Gradient Boosting

# Therefore the models which perform well are

score\_res = pd.DataFrame({'Model':['DecisionTree','GradientBoosting','RandomForest'],  
 'Train Score':[dTree\_tr, gbr\_tr, rfr\_tr],  
 'Test Score':[dTree\_ts, gbr\_ts, rfr\_ts]  
 })  
score\_res

Model Train Score Test Score  
0 DecisionTree 0.732016 0.776464  
1 GradientBoosting 0.724754 0.775132  
2 RandomForest 0.731702 0.776162