Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:

```
df=pd.read_csv(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\madrid_2005.
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.
...	...	...	...	...	...	...	...	...	...	...	
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.

237000 rows × 17 columns

Data Cleaning and Data Preprocessing

In [3]:

```
df=df.dropna()
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O
_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

In [7]:

```
data=df[['TCH', 'SO_2', 'PM25']]
data
```

Out[7]:

	TCH	SO_2	PM25
5	1.38	10.39	17.600000
22	1.29	6.94	6.020000
25	1.45	6.20	10.260000
31	1.38	10.60	21.870001
48	1.29	6.89	5.350000
...	...	...	...
236970	1.28	7.13	6.380000
236973	1.33	10.94	10.270000
236979	1.31	26.65	0.860000
236996	1.28	7.06	1.490000
236999	1.30	11.07	2.110000

20070 rows × 3 columns

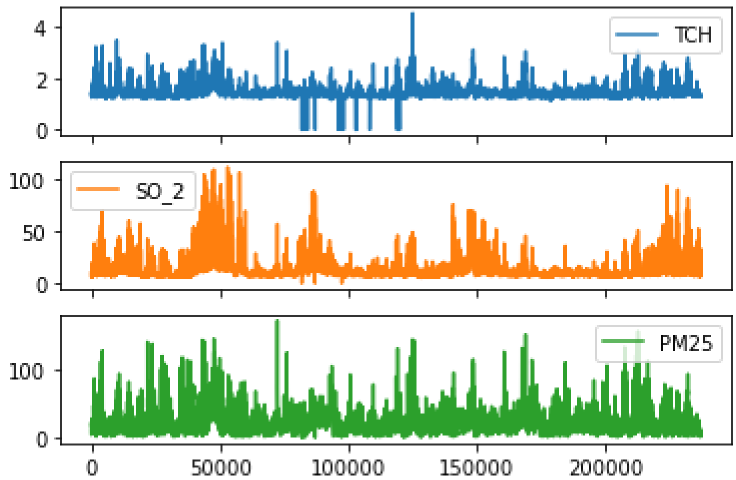
Line chart

In [8]:

```
data.plot.line(subplots=True)
```

Out[8]:

```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



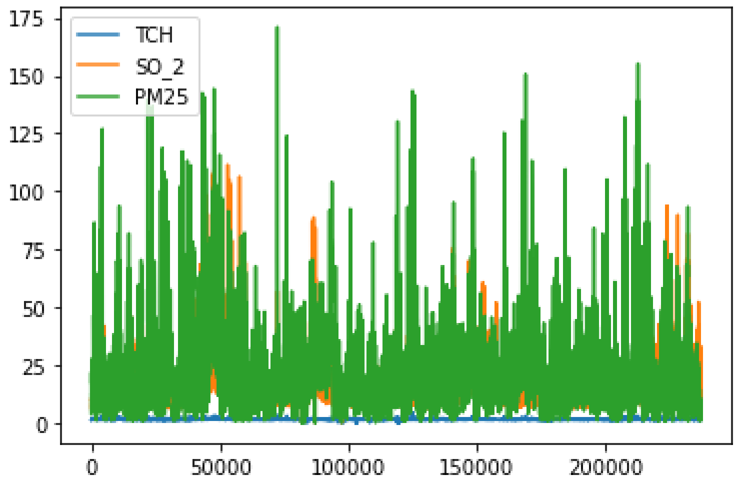
Line chart

In [9]:

```
data.plot.line()
```

Out[9]:

<AxesSubplot:>



Bar chart

In [10]:

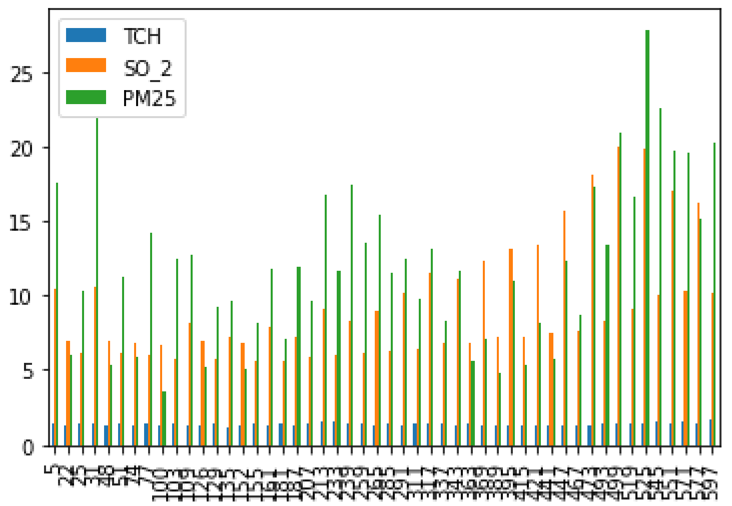
```
b=data[0:50]
```

In [11]:

```
b.plot.bar()
```

Out[11]:

<AxesSubplot:>



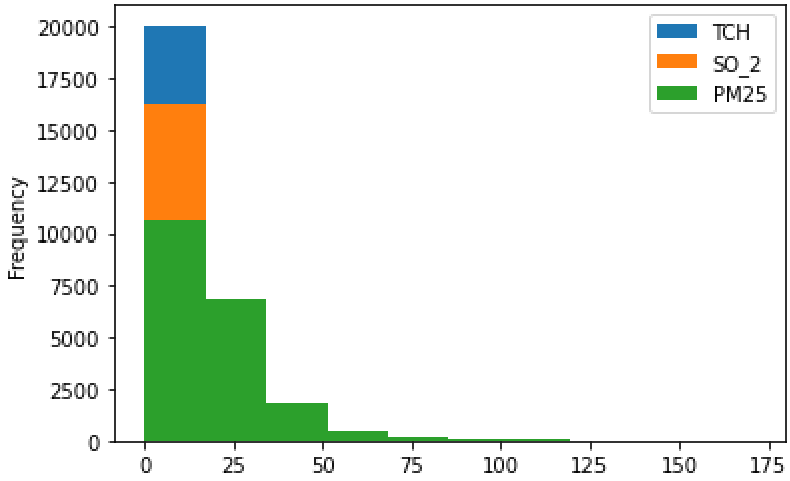
Histogram

In [12]:

```
data.plot.hist()
```

Out[12]:

```
<AxesSubplot:ylabel='Frequency'>
```



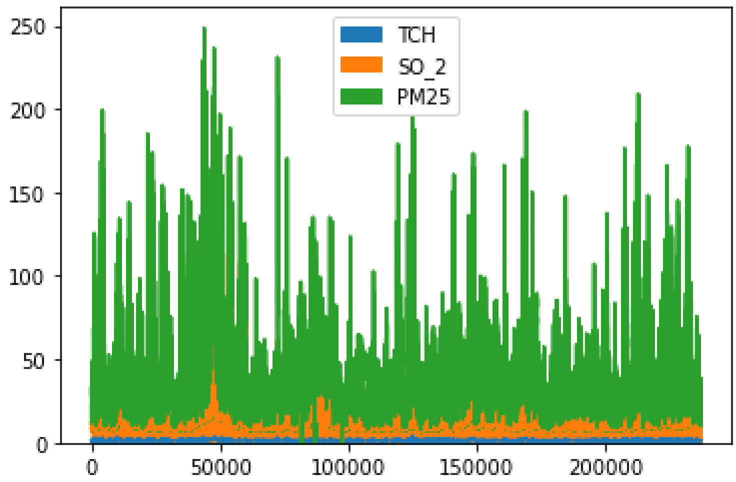
Area chart

In [13]:

```
data.plot.area()
```

Out[13]:

```
<AxesSubplot:>
```



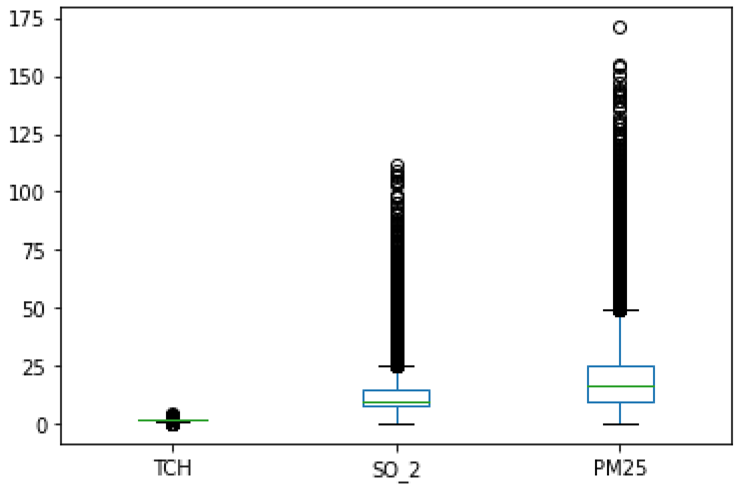
Box chart

In [14]:

```
data.plot.box()
```

Out[14]:

<AxesSubplot:>



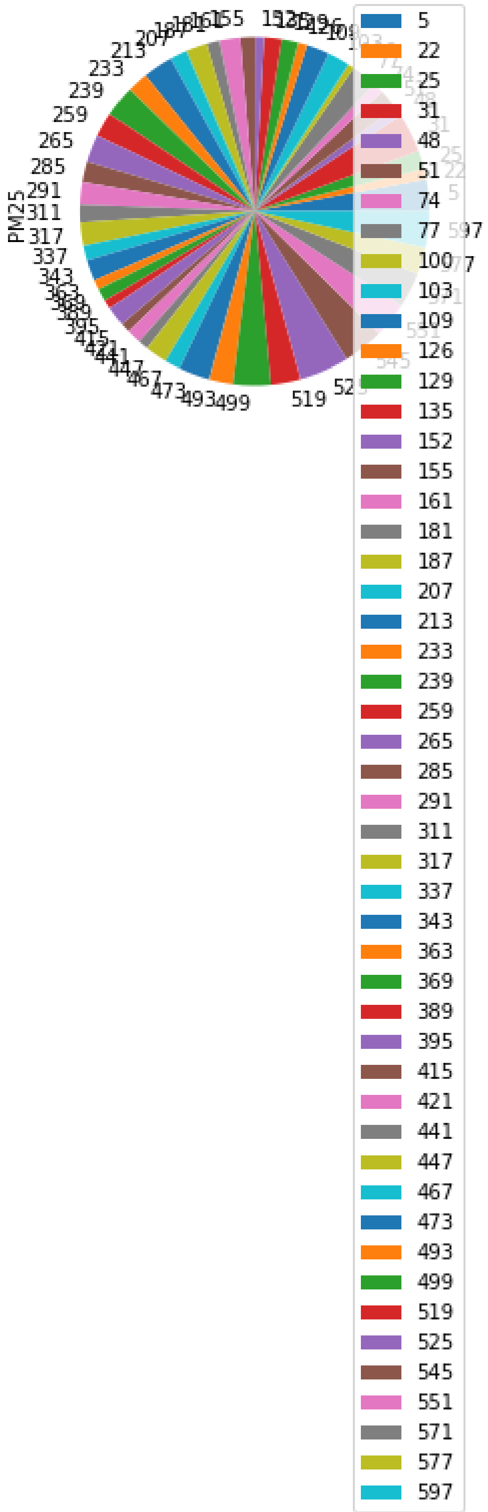
Pie chart

In [16]:

```
b.plot.pie(y='PM25' )
```

Out[16]:

```
<AxesSubplot:ylabel='PM25'>
```



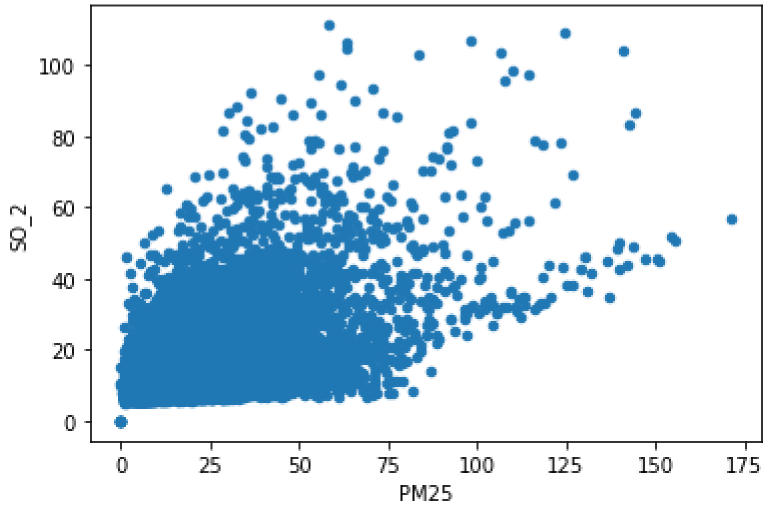
Scatter chart

In [17]:

```
data.plot.scatter(x='PM25' ,y='SO_2')
```

Out[17]:

<AxesSubplot:xlabel='PM25', ylabel='SO_2'>



In [18]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

In [19]:

```
df.describe()
```

Out[19]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000

In [20]:

```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

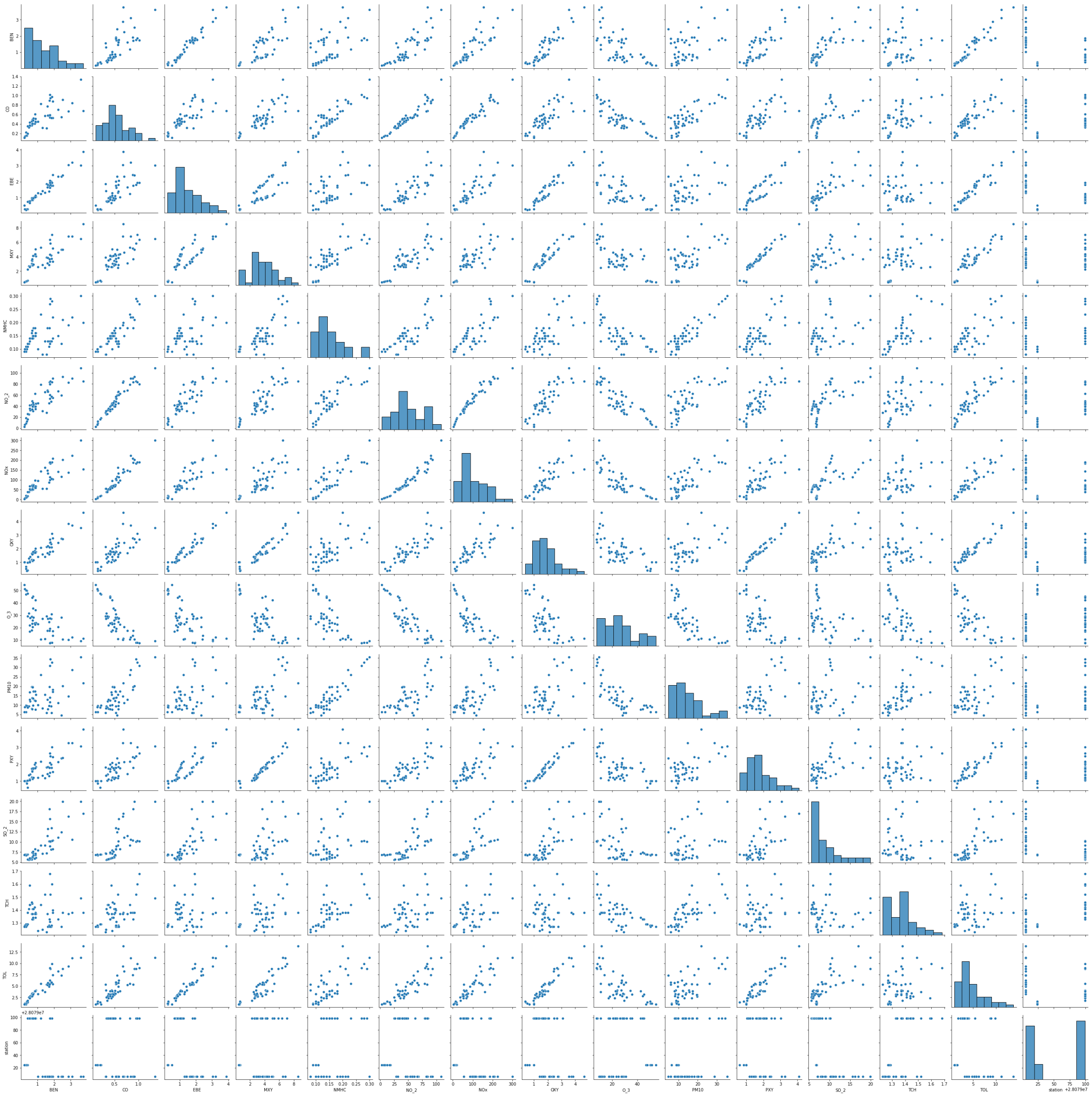
EDA AND VISUALIZATION

In [21]:

```
sns.pairplot(df1[0:50])
```

Out[21]:

```
<seaborn.axisgrid.PairGrid at 0x1f1a19f5880>
```



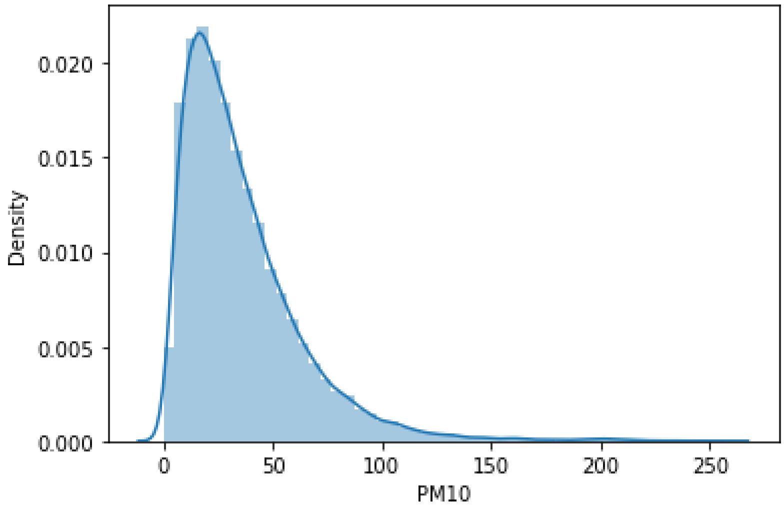
In [23]:

```
sns.distplot(df1['PM10'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure
-level function with similar flexibility) or `histplot` (an axes-level fun
ction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[23]:

```
<AxesSubplot:xlabel='PM10', ylabel='Density'>
```

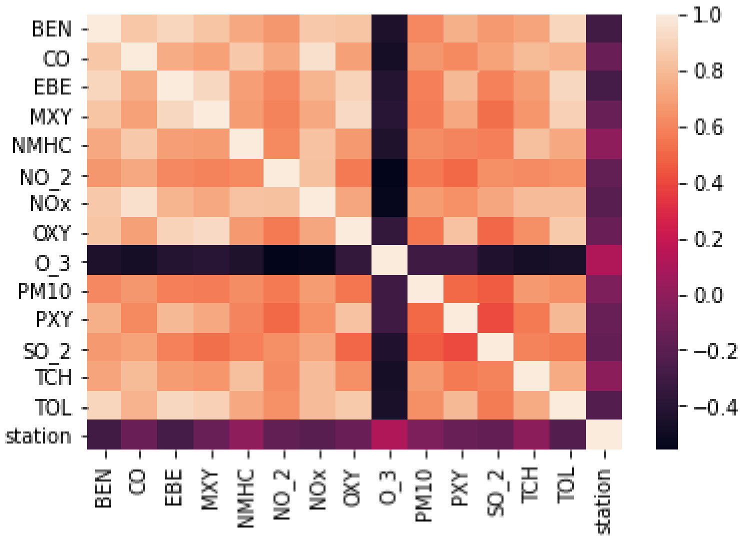


In [24]:

```
sns.heatmap(df1.corr())
```

Out[24]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BULDING

In [25]:

```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [26]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Linear Regression

In [27]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[27]:

```
LinearRegression()
```

In [28]:

```
lr.intercept_
```

Out[28]:

```
28078953.562257644
```

In [29]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[29]:

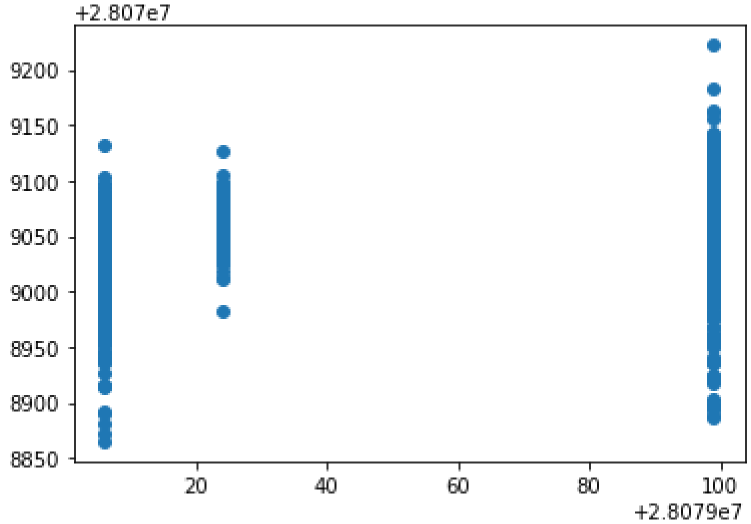
	Co-efficient
BEN	-9.534329
CO	39.995397
EBE	-13.644591
MXY	3.903344
NMHC	75.785405
NO_2	0.134611
NOx	-0.277371
OXY	3.049895
O_3	0.009189
PM10	0.048081
PXY	2.947113
SO_2	0.184869
TCH	66.848427
TOL	-0.661424

In [30]:

```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[30]:

<matplotlib.collections.PathCollection at 0x1f1af3ad970>



ACCURACY

In [31]:

```
lr.score(x_test,y_test)
```

Out[31]:

```
0.2814926859369915
```

In [32]:

```
lr.score(x_train,y_train)
```

Out[32]:

```
0.31357804533999745
```

Ridge and Lasso

In [33]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [34]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[34]:

```
Ridge(alpha=10)
```

Accuracy(Ridge)

In [35]:

```
rr.score(x_test,y_test)
```

Out[35]:

```
0.28138194302671393
```

In [36]:

```
rr.score(x_train,y_train)
```

Out[36]:

```
0.3133587604071677
```

In [37]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[37]:

```
Lasso(alpha=10)
```

In [38]:

```
la.score(x_train,y_train)
```

Out[38]:

```
0.06383146639556181
```

Accuracy(Lasso)

In [39]:

```
la.score(x_test,y_test)
```

Out[39]:

```
0.06585478293735003
```

Accuracy(Elastic Net)

In [40]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[40]:

```
ElasticNet()
```

In [41]:

```
en.coef_
```

Out[41]:

```
array([-5.69351254,  1.52343274, -7.6310123 ,  2.70676302,  0.923983  ,
       -0.04889445, -0.00986264,  1.91316733, -0.02603427,  0.23243237,
        1.56816065,  0.13729301,  1.61583742, -0.80348922])
```

In [42]:

```
en.intercept_
```

Out[42]:

```
28079049.89809796
```

In [43]:

```
prediction=en.predict(x_test)
```

In [44]:

```
en.score(x_test,y_test)
```

Out[44]:

```
0.17213950569078307
```

Evaluation Metrics

In [45]:

```
from sklearn import metrics
print(metrics.mean_absolute_error(y_test,prediction))
print(metrics.mean_squared_error(y_test,prediction))
print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
36.80409153704738
1541.142825495358
39.257391985400126
```

Logistic Regression

In [46]:

```
from sklearn.linear_model import LogisticRegression
```

In [47]:

```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [48]:

```
feature_matrix.shape
```

Out[48]:

```
(20070, 14)
```

In [49]:

```
target_vector.shape
```

Out[49]:

```
(20070,)
```

In [50]:

```
from sklearn.preprocessing import StandardScaler
```

In [51]:

```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [52]:

```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[52]:

```
LogisticRegression(max_iter=10000)
```

In [53]:

```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [54]:

```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079006]
```

In [55]:

```
logr.classes_
```

Out[55]:

```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [56]:

```
logr.score(fs,target_vector)
```

Out[56]:

```
0.879023418036871
```

In [57]:

```
logr.predict_proba(observation)[0][0]
```

Out[57]:

```
0.9998967601812779
```

In [58]:

```
logr.predict_proba(observation)
```

Out[58]:

```
array([[9.99896760e-01, 3.21124597e-30, 1.03239819e-04]])
```

Random Forest

In [59]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [60]:

```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[60]:

```
RandomForestClassifier()
```

In [61]:

```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [62]:

```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy")
grid_search.fit(x_train,y_train)
```

Out[62]:

```
GridSearchCV(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [63]:

```
grid_search.best_score_
```

Out[63]:

```
0.8684603271751554
```

In [64]:

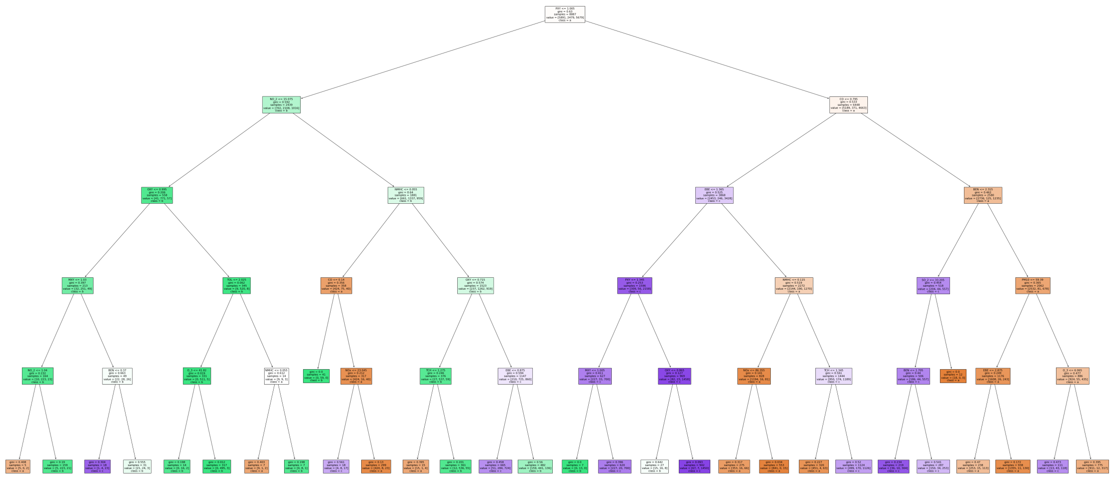
```
rfc_best=grid_search.best_estimator_
```

In [65]:

```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','d'],f
```

```
1\nvalue = [13, 43, 118]\nclass = c'),
 Text(4384.285714285714, 181.19999999999982, 'gini = 0.395\nsamples = 77
5\nvalue = [911, 12, 317]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.313578045339997 45

Ridge Regression:0.3133587604071677

Lasso Regression:0.065854782937 35003

ElasticNet Regression:0.17213950569078307

Logistic Regression:0.879023418036871

Random Forest:0.0.8684603271751554

Logistic Regression is suitable for this dataset