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_2012.
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
210715	2012-03-01 00:00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN
210716	2012-03-01 00:00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN
210717	2012-03-01 00:00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN
210718	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN
210719	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN

210720 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:

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

In [5]:

```
df.columns
```

Out[5]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'P
M25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

In [7]:

```
data=df[['BEN', 'TOL', 'TCH']]
data
```

Out[7]:

	BEN	TOL	TCH
6	0.4	0.6	1.33
30	0.4	0.5	1.33
54	0.4	0.5	1.33
78	0.3	0.4	1.34
102	0.4	0.5	1.33
...	...	...	...
210654	0.6	2.3	1.12
210673	2.0	6.2	1.33
210678	0.7	1.9	1.11
210697	1.5	4.9	1.34
210702	0.6	1.2	1.11

10916 rows × 3 columns

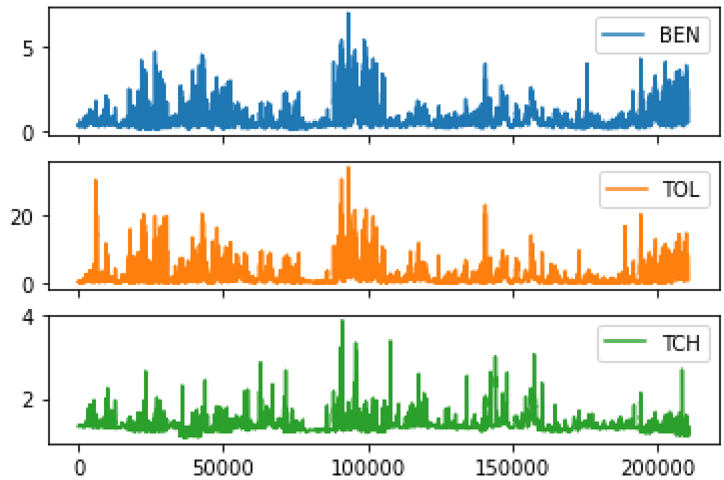
Line chart

In [8]:

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

Out[8]:

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



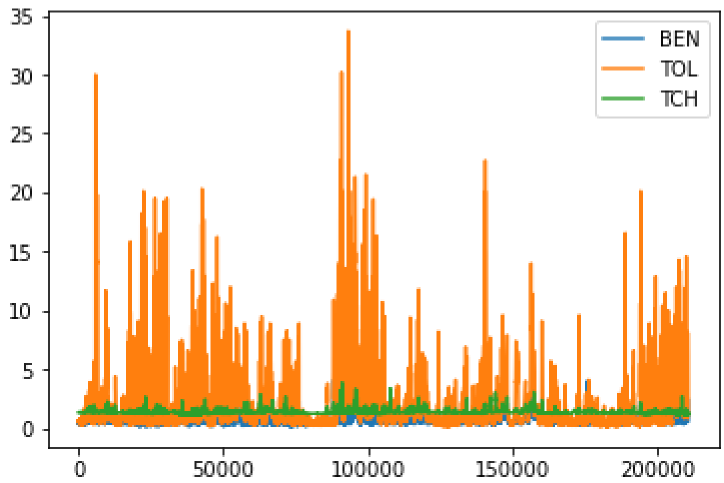
Line chart

In [9]:

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

Out[9]:

<AxesSubplot:>



Bar chart

In [10]:

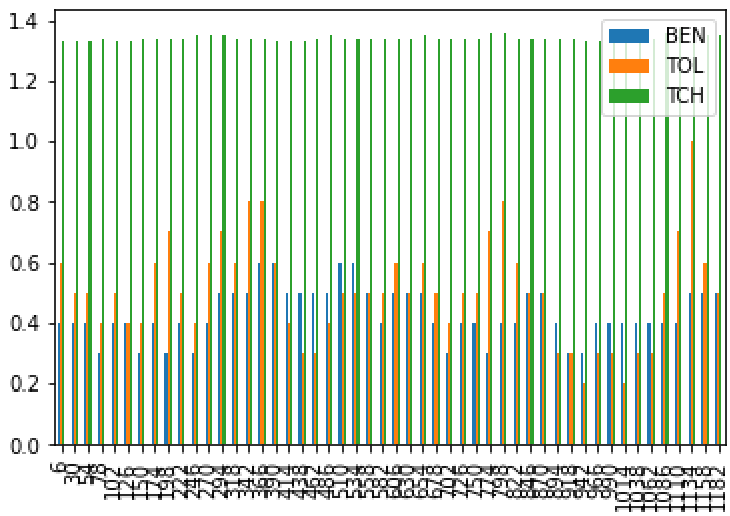
```
b=data[0:50]
```

In [11]:

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

Out[11]:

<AxesSubplot:>



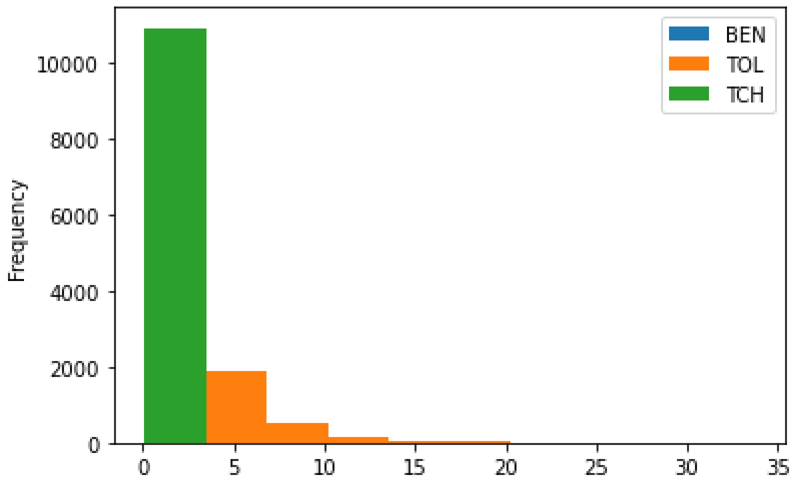
Histogram

In [12]:

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

Out[12]:

<AxesSubplot:ylabel='Frequency'>



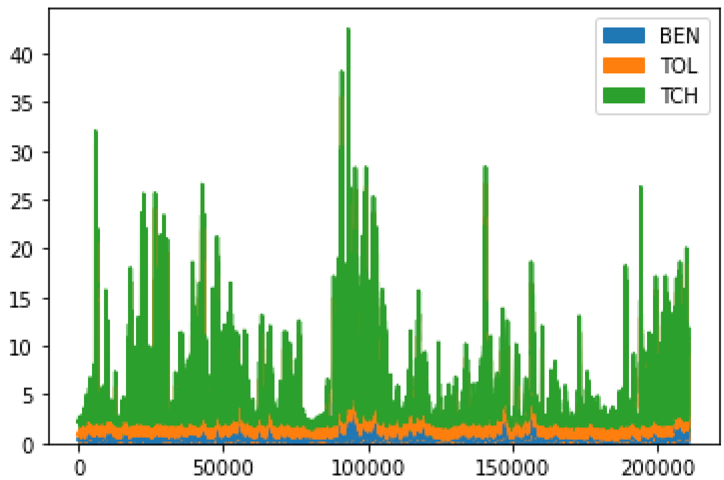
Area chart

In [13]:

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

Out[13]:

<AxesSubplot:>



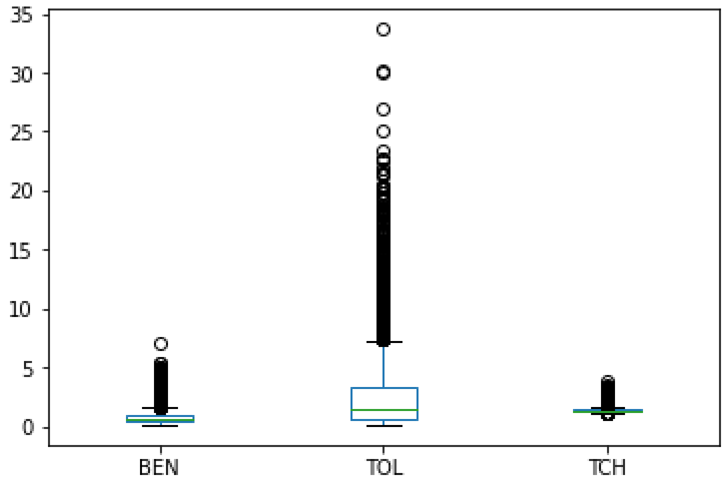
Box chart

In [14]:

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

Out[14]:

```
<AxesSubplot:>
```



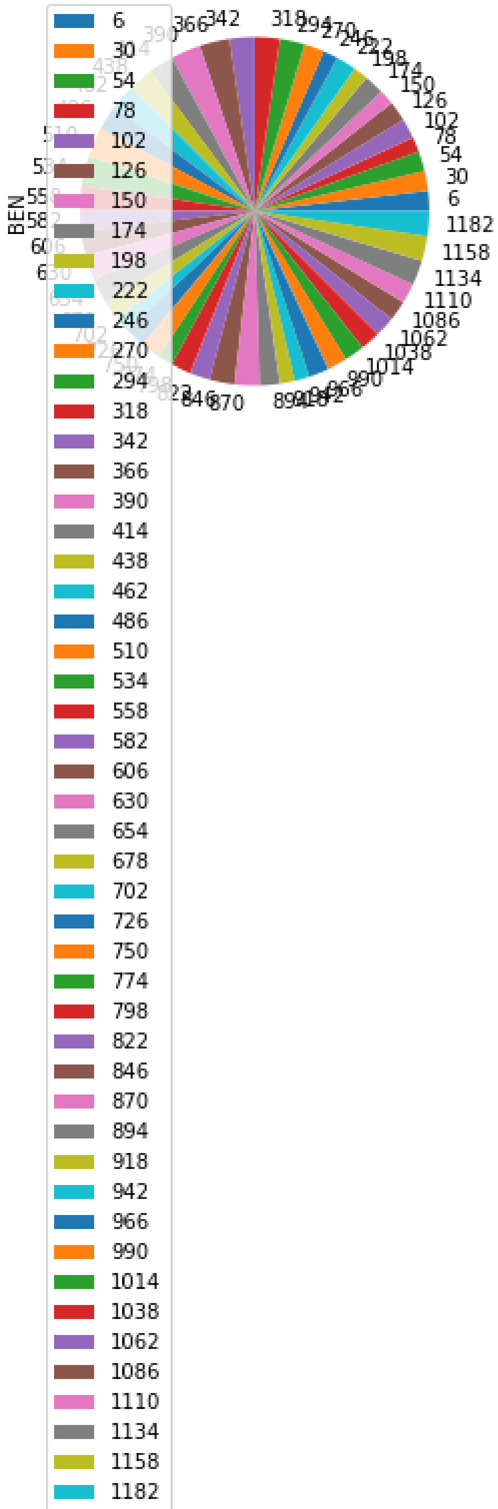
Pie chart

In [15]:

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

Out[15]:

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



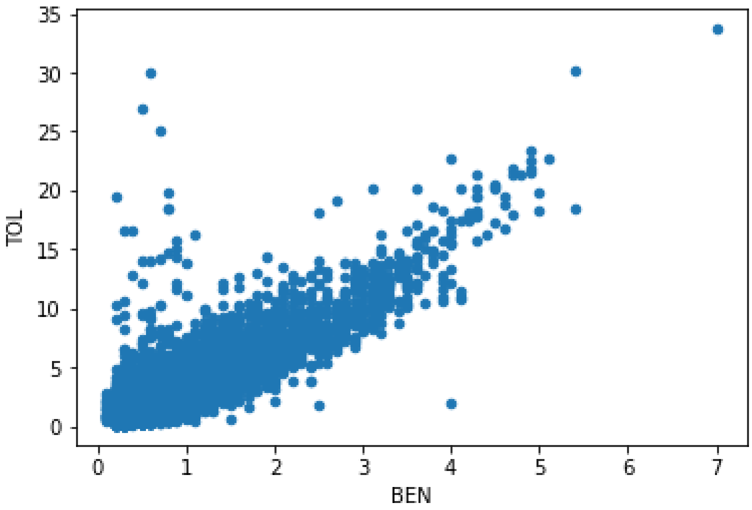
Scatter chart

In [16]:

```
data.plot.scatter(x='BEN' ,y='TOL')
```

Out[16]:

```
<AxesSubplot:xlabel='BEN', ylabel='TOL'>
```



In [17]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

In [18]:

```
df.describe()
```

Out[18]:

	BEN	CO	EBE	NMHC	NO	NO_2
count	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000

In [19]:

```
df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
```

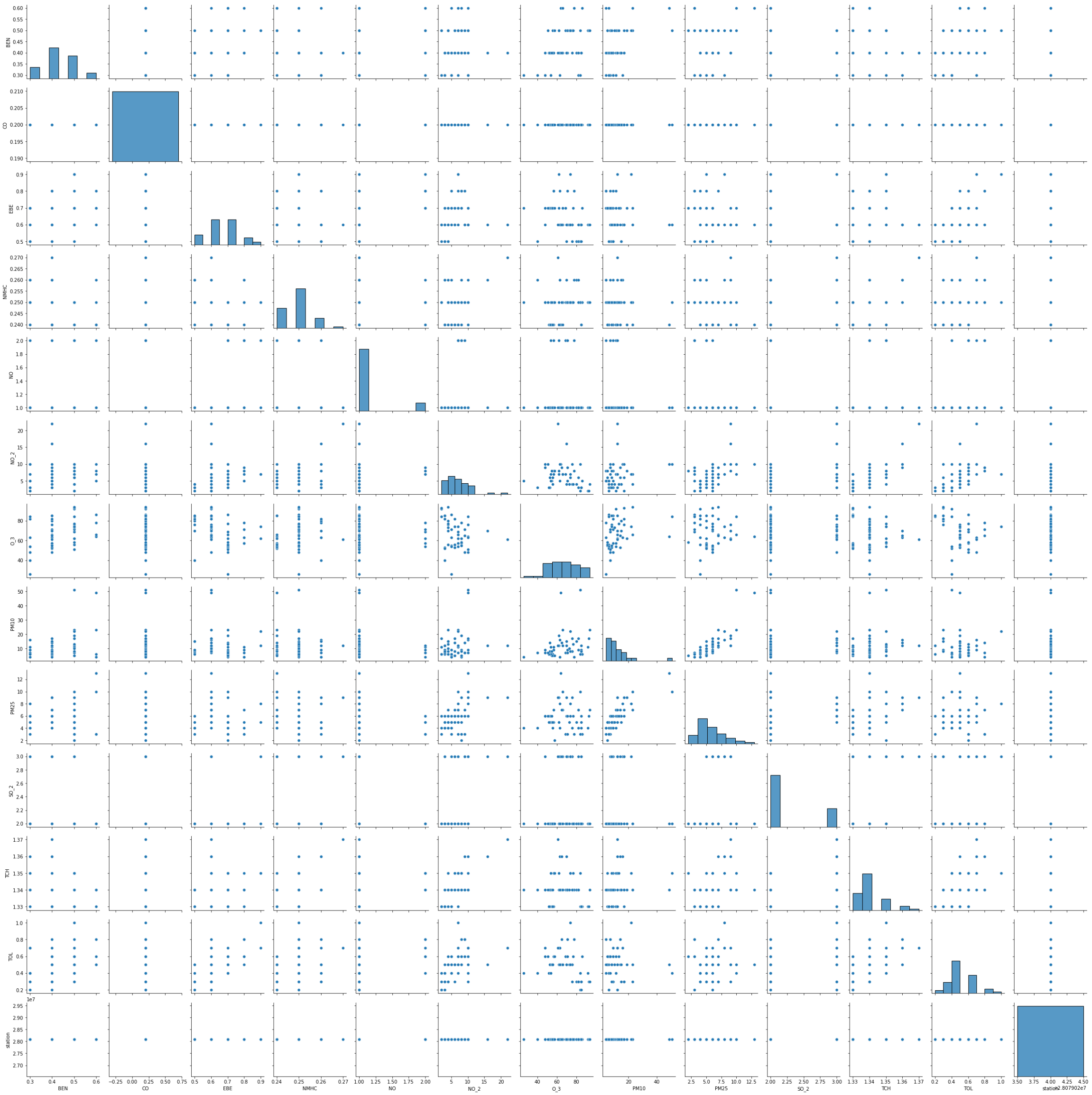
EDA AND VISUALIZATION

In [20]:

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

Out[20]:

```
<seaborn.axisgrid.PairGrid at 0x18fd7c01760>
```



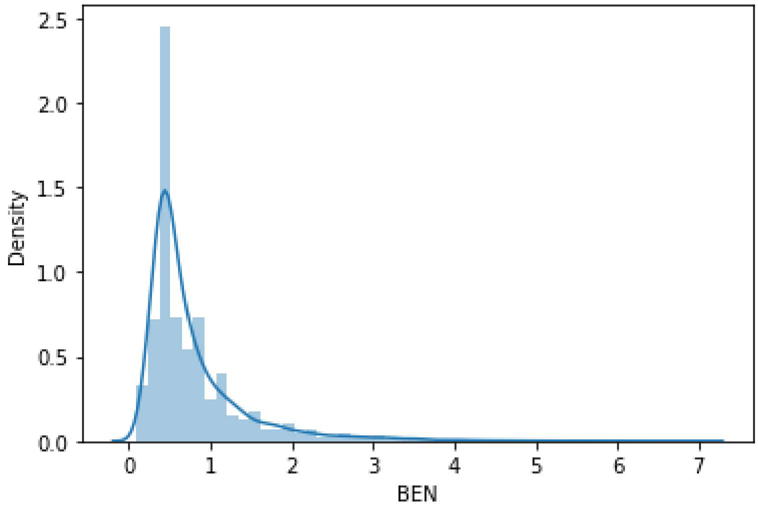
In [21]:

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

```
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[21]:

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

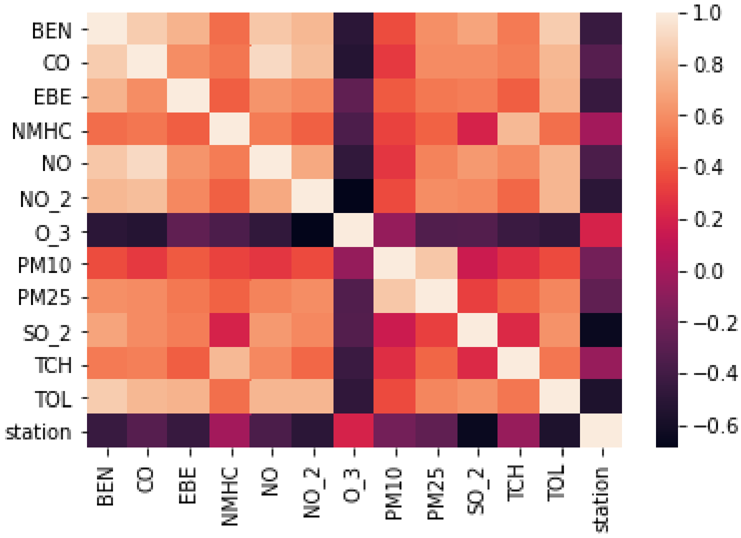


In [22]:

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

Out[22]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BULDING

In [23]:

```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [24]:

```
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 [25]:

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

Out[25]:

```
LinearRegression()
```

In [26]:

```
lr.intercept_
```

Out[26]:

```
28079017.331148606
```

In [27]:

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

Out[27]:

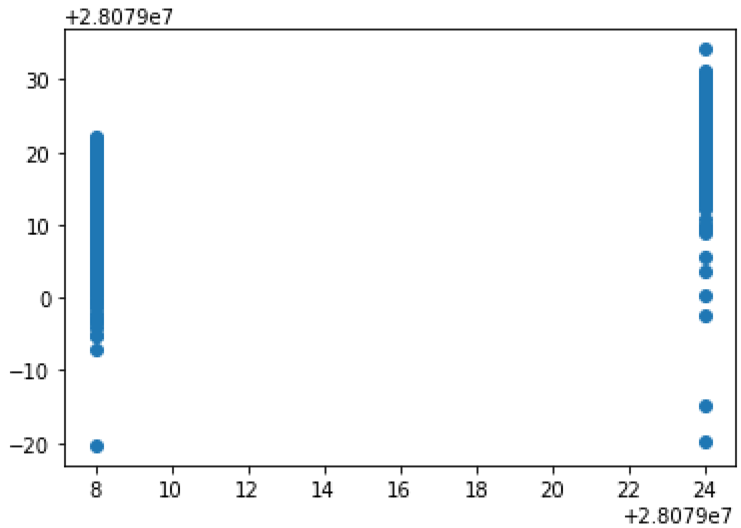
	Co-efficient
BEN	4.053746
CO	21.639243
EBE	-0.299229
NMHC	16.593507
NO	-0.023593
NO_2	-0.115481
O_3	-0.031714
PM10	0.005100
PM25	-0.057039
SO_2	-0.686152
TCH	1.976342
TOL	-1.535403

In [28]:

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

Out[28]:

```
<matplotlib.collections.PathCollection at 0x18fe4c39100>
```



ACCURACY

In [29]:

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

Out[29]:

```
0.6173246480033722
```

In [30]:

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

Out[30]:

```
0.626814373777909
```

Ridge and Lasso

In [31]:

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

In [32]:

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

Out[32]:

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

Accuracy(Ridge)

In [33]:

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

Out[33]:

```
0.6123186216863385
```

In [34]:

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

Out[34]:

```
0.6228871250229713
```

In [35]:

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

Out[35]:

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

In [36]:

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

Out[36]:

```
0.37384023853831394
```

Accuracy(Lasso)

In [37]:

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

Out[37]:

```
0.3655097139415614
```

Accuracy(Elastic Net)

In [38]:

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

Out[38]:

```
ElasticNet()
```

In [39]:

```
en.coef_
```

Out[39]:

```
array([ 0.        ,  0.        , -0.        ,  0.        ,  0.06620803,
       -0.08549274, -0.039121  ,  0.        ,  0.        , -0.70807648,
        0.        , -0.71182233])
```

In [40]:

```
en.intercept_
```

Out[40]:

```
28079027.755586464
```

In [41]:

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

In [42]:

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

Out[42]:

```
0.5392059744304518
```

Evaluation Metrics

In [43]:

```
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)))
```

```
3.4507079618543615
23.378308914389724
4.835112089123656
```

Logistic Regression

In [44]:

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

In [45]:

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

In [46]:

```
feature_matrix.shape
```

Out[46]:

```
(10916, 10)
```

In [47]:

```
target_vector.shape
```

Out[47]:

```
(10916,)
```

In [48]:

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

In [49]:

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

In [50]:

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

Out[50]:

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

In [51]:

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

In [52]:

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

```
[28079008]
```

In [53]:

```
logr.classes_
```

Out[53]:

```
array([28079008, 28079024], dtype=int64)
```

In [54]:

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

Out[54]:

```
0.9293697325027482
```

In [55]:

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

Out[55]:

```
1.0
```

In [56]:

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

Out[56]:

```
array([[1.00000000e+00, 3.50349553e-26]])
```

Random Forest

In [57]:

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

In [58]:

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

Out[58]:

```
RandomForestClassifier()
```

In [59]:

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

In [60]:

```
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[60]:

```
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 [61]:

```
grid_search.best_score_
```

Out[61]:

```
0.9646636937508478
```

In [62]:

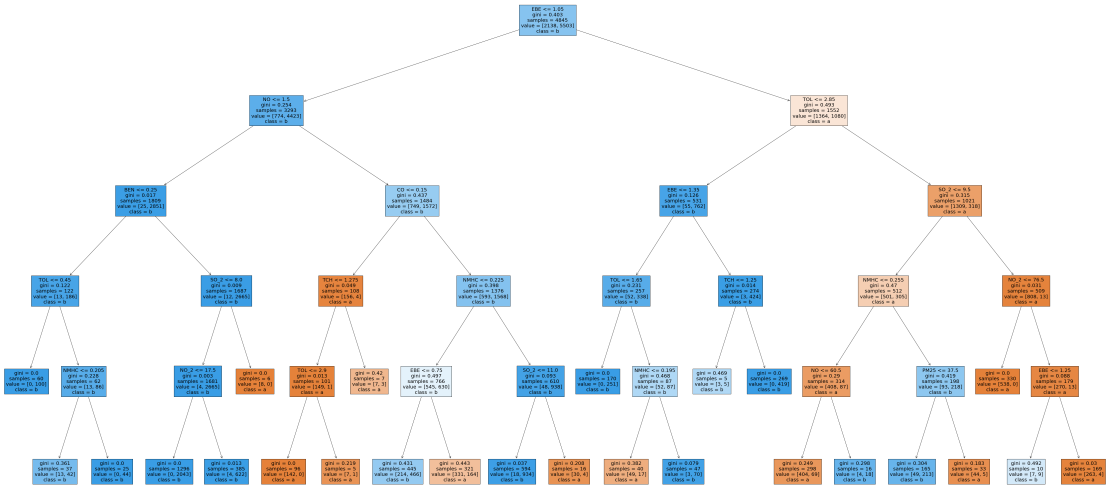
```
rfc_best=grid_search.best_estimator_
```

In [63]:

```
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
```

```
\nvalue = [7, 9]\nclass = b'),
 Text(4349.538461538462, 181.19999999999982, 'gini = 0.03\nsamples = 169
\nvalue = [263, 4]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.626814373777909

Ridge Regression:0.6228871250229713

Lasso Regression:0.3655097139415614

ElasticNet Regression:0.5392059744304518

Logistic Regression:0.9293697325027482

Random Forest:0.9646636937508478

Random Forest is suitable for this dataset