In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as pp
```

In [2]:

```
df1 = pd.read_csv(r"C:\Users\user\Desktop\c10\madrid_2007.csv")
df = df1.head(1000)
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.
...	...	...	...	...	...	...	...	...	...	...	
995	2007-12-02 15:00:00	NaN	0.42	NaN	NaN	NaN	51.220001	88.709999	NaN	49.360001	16.
996	2007-12-02 15:00:00	NaN	0.44	NaN	NaN	NaN	NaN	NaN	NaN	26.090000	24.
997	2007-12-02 15:00:00	NaN	0.37	NaN	NaN	0.35	58.570000	99.980003	NaN	26.480000	14.
998	2007-12-02 15:00:00	NaN	0.61	NaN	NaN	NaN	54.150002	109.500000	NaN	21.650000	17.
999	2007-12-02 15:00:00	NaN	0.10	NaN	NaN	NaN	32.230000	43.970001	NaN	35.799999	9.

1000 rows × 17 columns

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: 101 entries, 4 to 987
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     101 non-null    object 
 1   BEN      101 non-null    float64
 2   CO       101 non-null    float64
 3   EBE      101 non-null    float64
 4   MXY      101 non-null    float64
 5   NMHC     101 non-null    float64
 6   NO_2     101 non-null    float64
 7   NOx      101 non-null    float64
 8   OXY      101 non-null    float64
 9   O_3      101 non-null    float64
 10  PM10     101 non-null    float64
 11  PM25     101 non-null    float64
 12  PXY      101 non-null    float64
 13  SO_2     101 non-null    float64
 14  TCH      101 non-null    float64
 15  TOL      101 non-null    float64
 16  station  101 non-null    int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 14.2+ KB
```

In [6]:

```
data=df[['CO' ,'station']]
data
```

Out[6]:

	CO	station
4	1.86	28079006
21	0.31	28079024
25	1.42	28079099
30	1.89	28079006
47	0.30	28079024
...	...	...
935	0.58	28079099
957	0.28	28079024
961	0.54	28079099
983	0.25	28079024
987	0.47	28079099

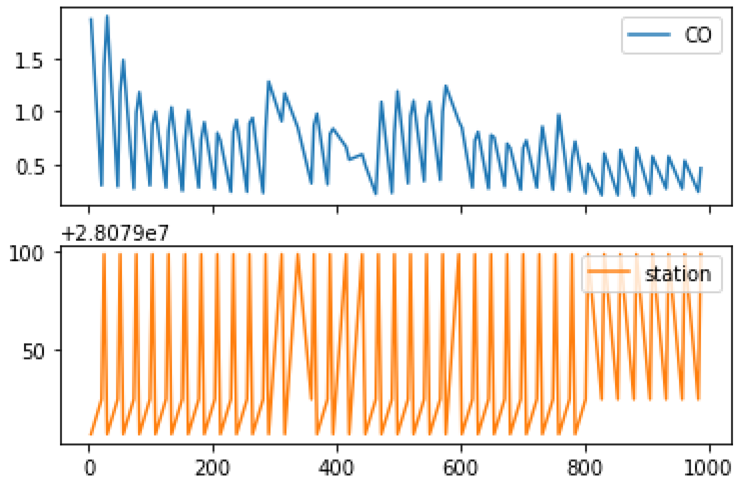
101 rows × 2 columns

In [7]:

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

Out[7]:

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

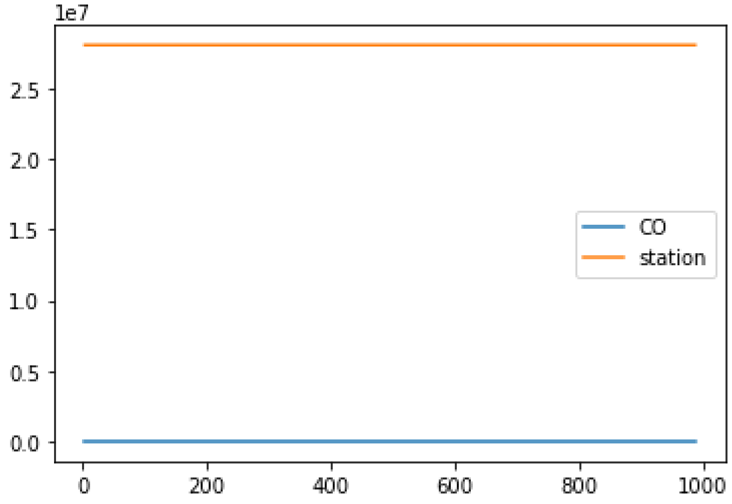


In [8]:

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

Out[8]:

<AxesSubplot:>



In [9]:

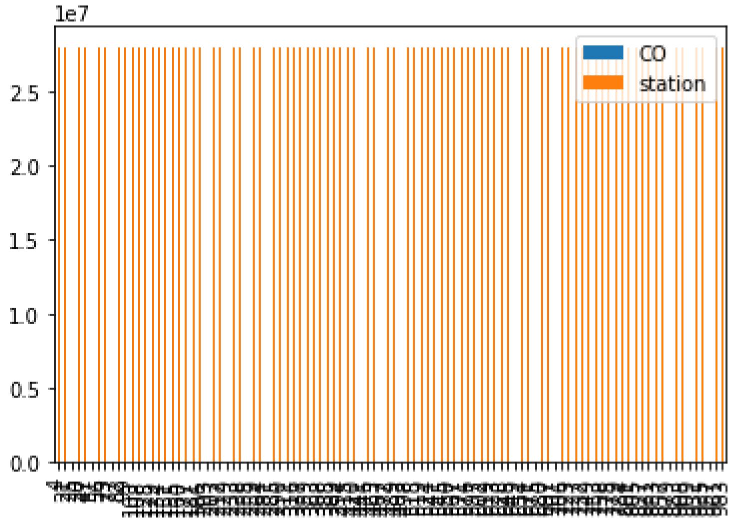
```
x = data[0:100]
```

In [10]:

```
x.plot.bar()
```

Out[10]:

<AxesSubplot:>

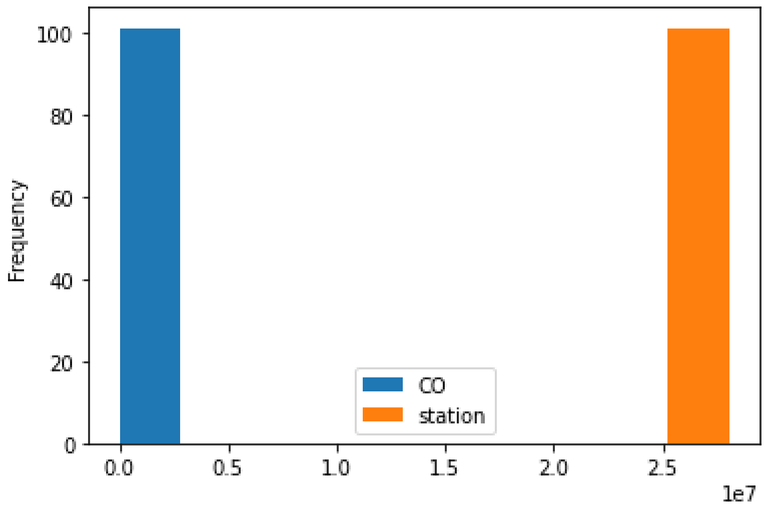


In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>

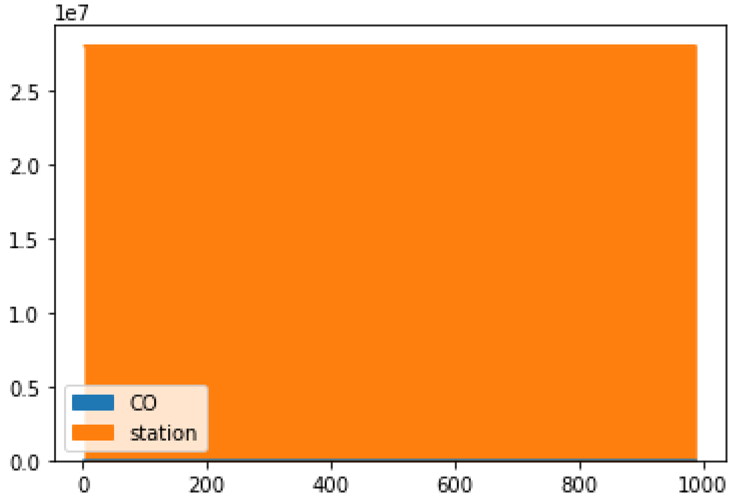


In [12]:

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

Out[12]:

<AxesSubplot:>

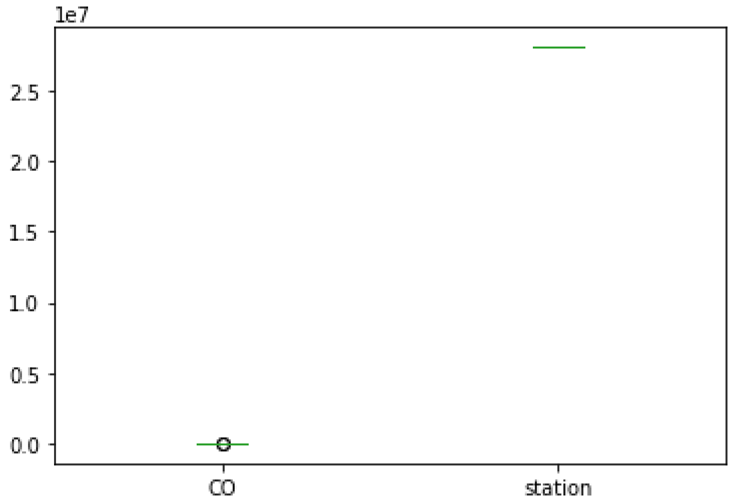


In [13]:

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

Out[13]:

<AxesSubplot:>



In [14]:

```
x.plot.pie(y='station' )
```

Out[14]:

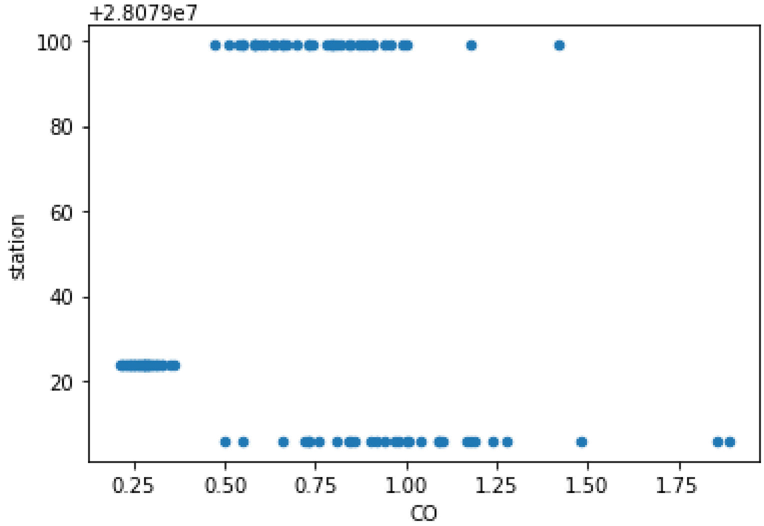
```
<AxesSubplot:ylabel='station'>
```

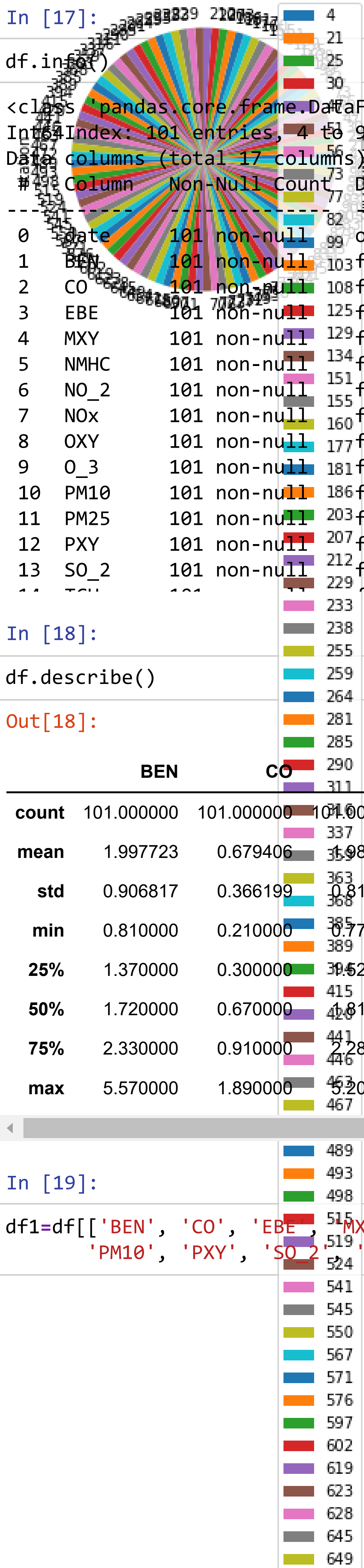
In [16]:

```
data.plot.scatter(x='CO' ,y='station')
```

Out[16]:

<AxesSubplot:xlabel='CO', ylabel='station'>





In [17]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101 entries, 4 to 987
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     101 non-null    object
 1   BEN      101 non-null    float64
 2   CO       101 non-null    float64
 3   EBE      101 non-null    float64
 4   MXY      101 non-null    float64
 5   NMHC     101 non-null    float64
 6   NO_2     101 non-null    float64
 7   NOx      101 non-null    float64
 8   OXY      101 non-null    float64
 9   O_3      101 non-null    float64
 10  PM10     101 non-null    float64
 11  PM25     101 non-null    float64
 12  PXY      101 non-null    float64
 13  SO_2     101 non-null    float64
```

In [18]:

```
df.describe()
```

Out[18]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000
mean	1.997723	0.679406	1.985446	3.786436	0.345446	80.299010	208.948714
std	0.906817	0.366199	0.816284	1.561502	0.082250	26.741420	104.795058
min	0.810000	0.210000	0.770000	1.450000	0.220000	23.910000	47.470001
25%	1.370000	0.300000	1.520000	2.510000	0.280000	56.930000	129.000000
50%	1.720000	0.670000	1.810000	3.660000	0.350000	77.330002	208.899994
75%	2.330000	0.910000	2.280000	4.520000	0.390000	101.500000	263.000000
max	5.570000	1.890000	5.200000	9.270000	0.650000	151.000000	622.700012

In [19]:

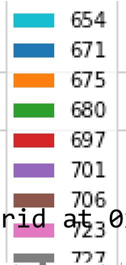
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

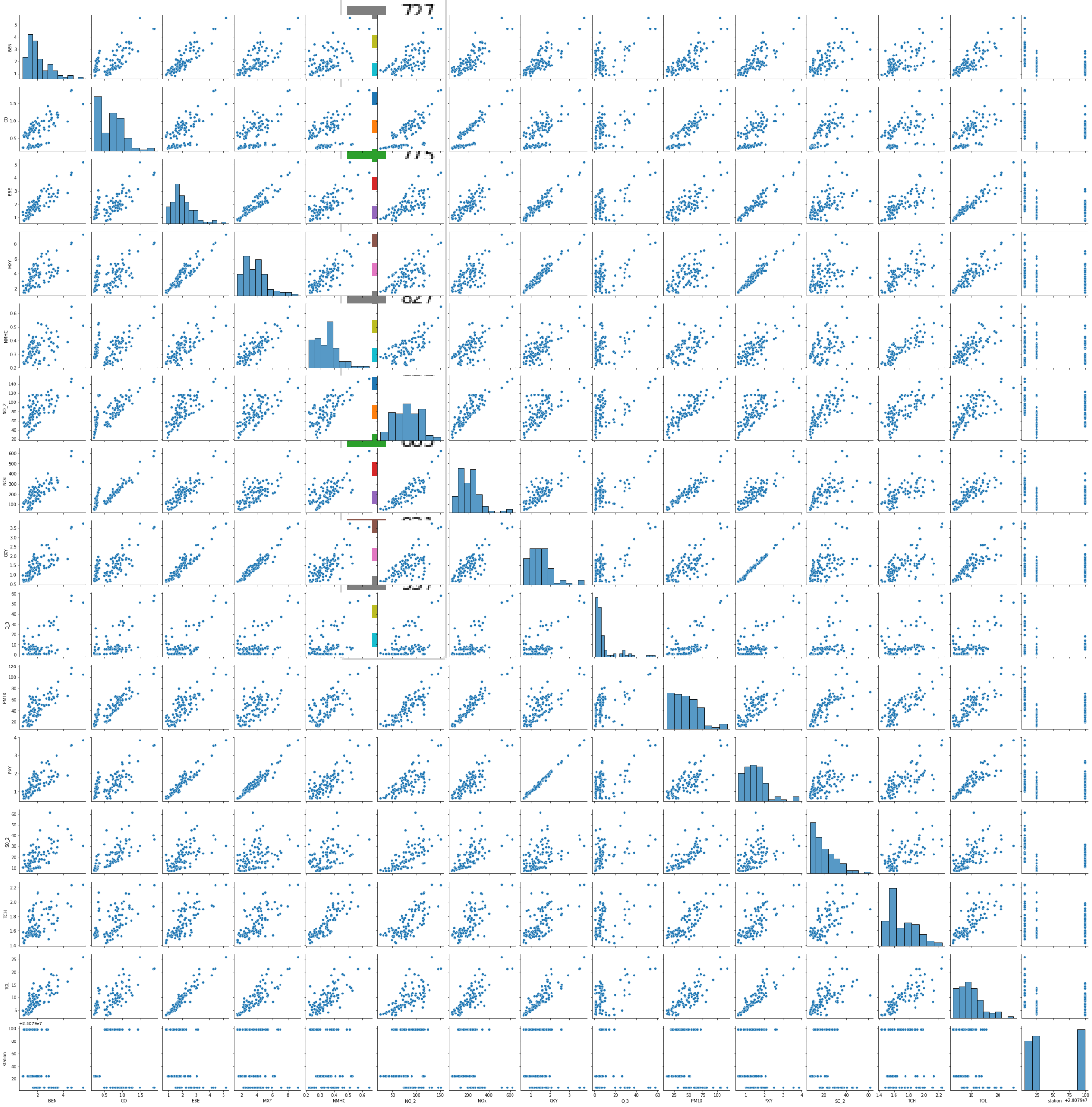
In [20]:

```
sb.pairplot(df1[0:100])
```

Out[20]:

```
<seaborn.axisgrid.PairGrid at 0x22934f2ee80>
```





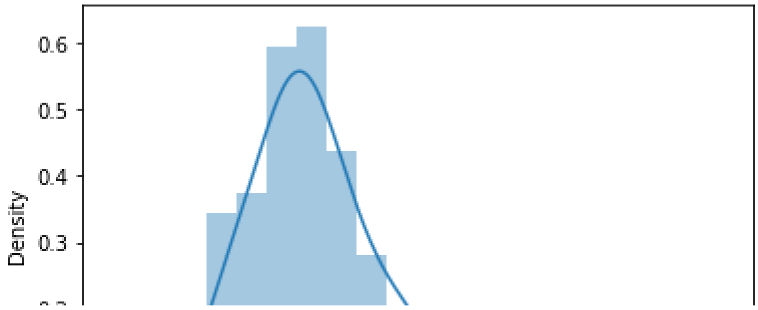
In [21]:

```
sb.distplot(df1['EBE'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-l
evel function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[21]:

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

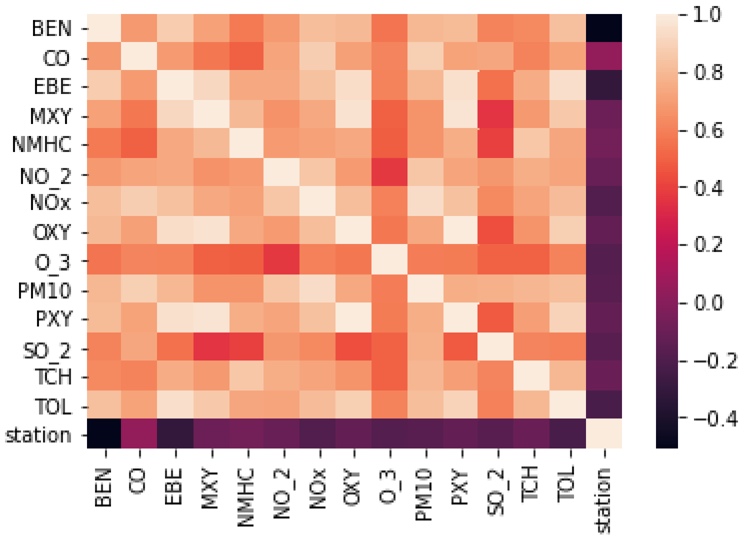


In [22]:

```
sb.heatmap(df1.corr())
```

Out[22]:

```
<AxesSubplot:>
```



In [23]:

```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', '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)
```

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]:

```
28079012.969358567
```

In [27]:

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

Out[27]:

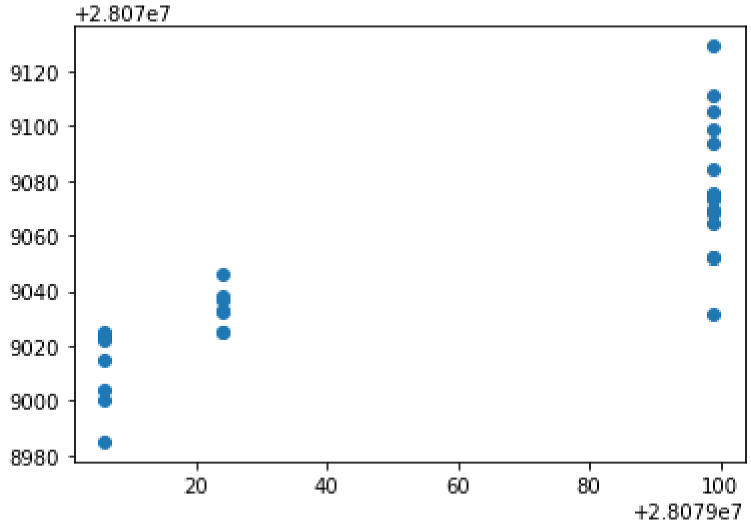
	Co-efficient
BEN	-18.322228
CO	129.622852
EBE	-95.975102
MXY	41.080598
NMHC	-18.405558
NO_2	0.587834
NOx	-0.459477
OXY	20.942919
O_3	0.188272
PM10	0.635570
PXY	-28.919749
SO_2	-1.122275
TCH	23.172145
TOL	3.750273

In [28]:

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

Out[28]:

```
<matplotlib.collections.PathCollection at 0x229421c6400>
```



In [29]:

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

Out[29]:

```
0.6714930835249275
```

In [30]:

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

Out[30]:

```
0.8377936458050441
```

In [31]:

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

In [32]:

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

Out[32]:

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

In [33]:

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

Out[33]:

```
0.26554594914406027
```

In [34]:

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

Out[34]:

```
0.5858982564362569
```

In [35]:

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

Out[35]:

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

In [36]:

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

Out[36]:

```
0.12918524864848024
```

In [37]:

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

Out[37]:

```
0.0013479668890272745
```

In [38]:

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

Out[38]:

```
ElasticNet()
```

In [39]:

```
e.coef_
```

Out[39]:

```
array([-14.14851847,   4.7213111 ,  -3.73536822,   8.93718374,
         0.        ,   0.12875784,  -0.14188838,   2.6067886 ,
        -0.28057423,   1.08483308,   2.3995999 ,  -0.4650452 ,
        -0.        ,  -2.76623723])
```

In [40]:

```
e.intercept_
```

Out[40]:

```
28079042.397540484
```

In [41]:

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

In [42]:

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

Out[42]:

```
0.09407581702575019
```

In [43]:

```
from sklearn import metrics
```

In [44]:

```
print(metrics.mean_squared_error(y_test,prediction))
```

```
1654.6229144051315
```

In [45]:

```
print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
40.67705636357099
```

In [46]:

```
print(metrics.mean_absolute_error(y_test,prediction))
```

```
36.71744495029411
```

In [47]:

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

In [48]:

```
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 [49]:

```
feature_matrix.shape
```

Out[49]:

```
(101, 14)
```

In [50]:

```
target_vector.shape
```

Out[50]:

```
(101,)
```

In [51]:

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

In [52]:

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

In [53]:

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

Out[53]:

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

In [54]:

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

In [55]:

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

```
[28079006]
```

In [56]:

```
logr.classes_
```

Out[56]:

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

In [57]:

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

Out[57]:

```
1.0
```

In [58]:

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

Out[58]:

```
0.9999964046241867
```

In [59]:

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

Out[59]:

```
array([[9.99996405e-01, 6.15542634e-20, 3.59537581e-06]])
```

In [60]:

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

In [61]:

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

Out[61]:

```
RandomForestClassifier()
```

In [62]:

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

In [63]:

```
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[63]:

```
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 [64]:

```
grid_search.best_score_
```

Out[64]:

```
0.8428571428571429
```

In [65]:

```
rfc_best=grid_search.best_estimator_
```

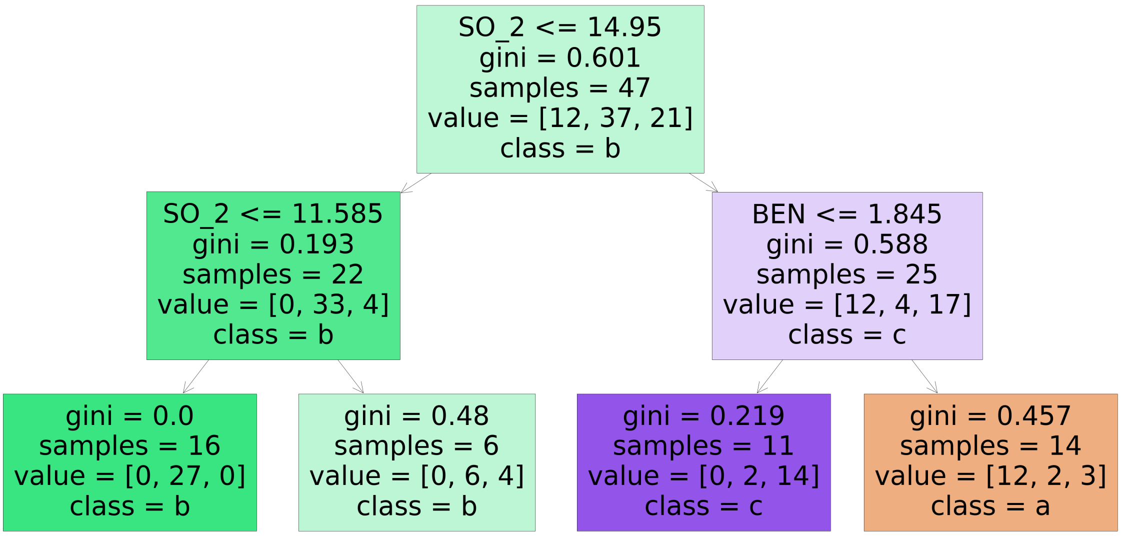
In [66]:

```
from sklearn.tree import plot_tree

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

Out[66]:

```
[Text(2232.0, 1812.0, 'SO_2 <= 14.95\ngini = 0.601\nsamples = 47\nvalue =
[12, 37, 21]\nclass = b'),
 Text(1116.0, 1087.2, 'SO_2 <= 11.585\ngini = 0.193\nsamples = 22\nvalue =
[0, 33, 4]\nclass = b'),
 Text(558.0, 362.39999999999986, 'gini = 0.0\nsamples = 16\nvalue = [0, 2
7, 0]\nclass = b'),
 Text(1674.0, 362.39999999999986, 'gini = 0.48\nsamples = 6\nvalue = [0,
6, 4]\nclass = b'),
 Text(3348.0, 1087.2, 'BEN <= 1.845\ngini = 0.588\nsamples = 25\nvalue =
[12, 4, 17]\nclass = c'),
 Text(2790.0, 362.39999999999986, 'gini = 0.219\nsamples = 11\nvalue = [0,
2, 14]\nclass = c'),
 Text(3906.0, 362.39999999999986, 'gini = 0.457\nsamples = 14\nvalue = [1
2, 2, 3]\nclass = a')]
```



random forest is best suitable for this data set

In [ ]: