In [2]:

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

In [3]:

```
df1 = pd.read_csv(r"C:\Users\user\Desktop\c10\madrid_2002.csv")
df = df1.head(1000)
df
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.540000	41.99(
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.850000	20.98(
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.010000	28.44(
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.120000	42.18(
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.280000	76.33(
...	...	...	...	...	...	...	...	...	...	...	
995	2002-04-02 16:00:00	2.19	0.36	NaN	NaN	NaN	58.709999	93.349998	NaN	70.830002	19.85(
996	2002-04-02 16:00:00	0.87	0.30	1.00	NaN	0.09	32.580002	45.150002	NaN	77.910004	14.43(
997	2002-04-02 16:00:00	0.46	0.50	0.27	0.41	0.10	11.550000	13.290000	0.49	82.260002	19.80
998	2002-04-02 16:00:00	1.11	0.44	0.96	2.00	NaN	96.790001	209.600006	0.66	34.540001	
999	2002-04-02 16:00:00	1.98	0.45	2.12	6.27	0.11	49.419998	75.730003	2.86	76.000000	25.16(

1000 rows × 16 columns

In [4]:

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

In [5]:

```
df.columns
```

Out[5]:

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

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 160 entries, 1 to 999
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     160 non-null    object 
 1   BEN      160 non-null    float64
 2   CO       160 non-null    float64
 3   EBE      160 non-null    float64
 4   MXY      160 non-null    float64
 5   NMHC     160 non-null    float64
 6   NO_2     160 non-null    float64
 7   NOx      160 non-null    float64
 8   OXY      160 non-null    float64
 9   O_3      160 non-null    float64
 10  PM10     160 non-null    float64
 11  PXY      160 non-null    float64
 12  SO_2     160 non-null    float64
 13  TCH      160 non-null    float64
 14  TOL      160 non-null    float64
 15  station  160 non-null    int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 21.2+ KB
```

In [7]:

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

Out[7]:

	CO	station
1	0.71	28079035
5	0.72	28079006
22	0.80	28079024
24	1.04	28079099
26	0.53	28079035
...	...	...
974	0.57	28079099
976	0.40	28079035
980	0.63	28079006
997	0.50	28079024
999	0.45	28079099

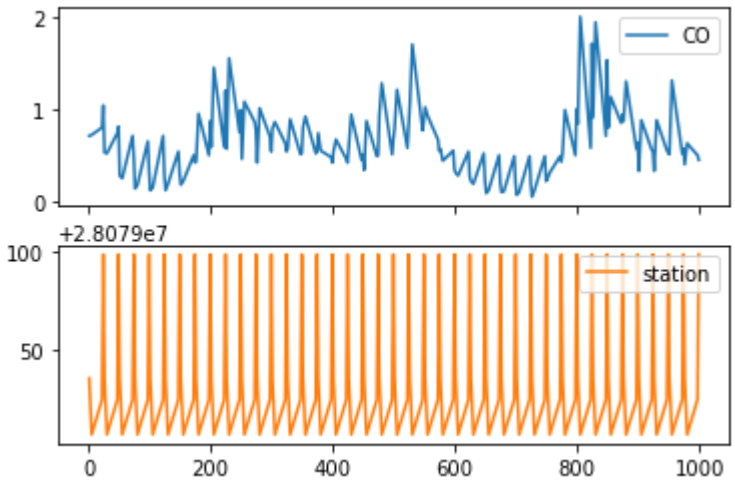
160 rows × 2 columns

In [8]:

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

Out[8]:

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

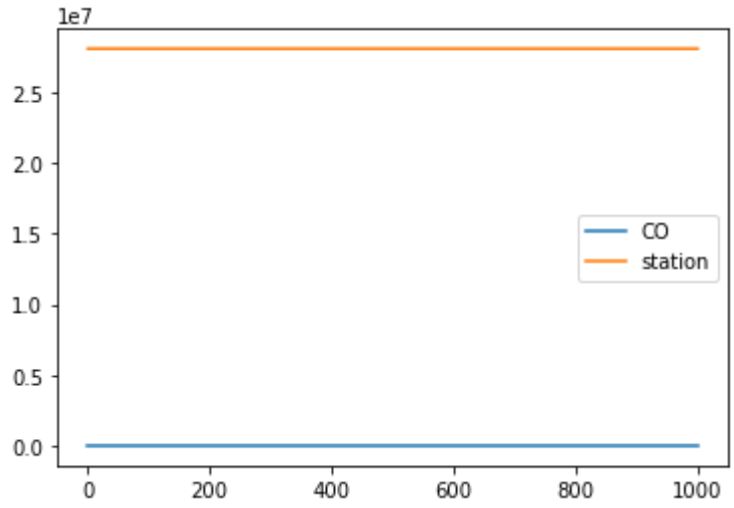


In [9]:

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

Out[9]:

<AxesSubplot:>



In [10]:

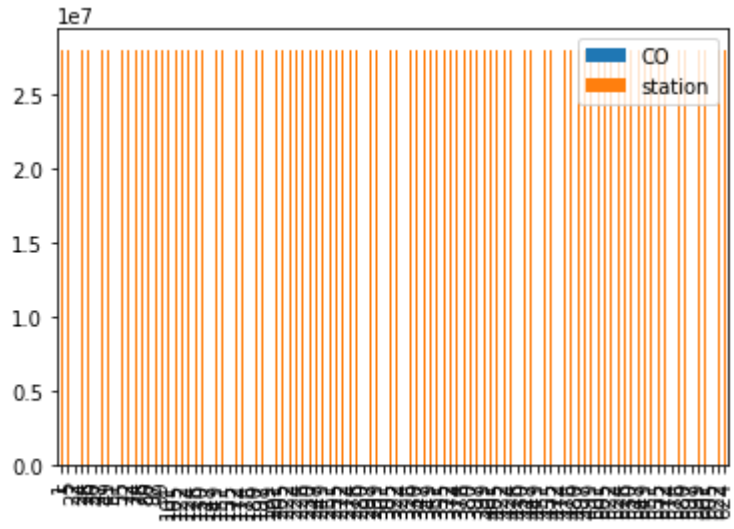
```
x = data[0:100]
```

In [11]:

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

Out[11]:

<AxesSubplot:>

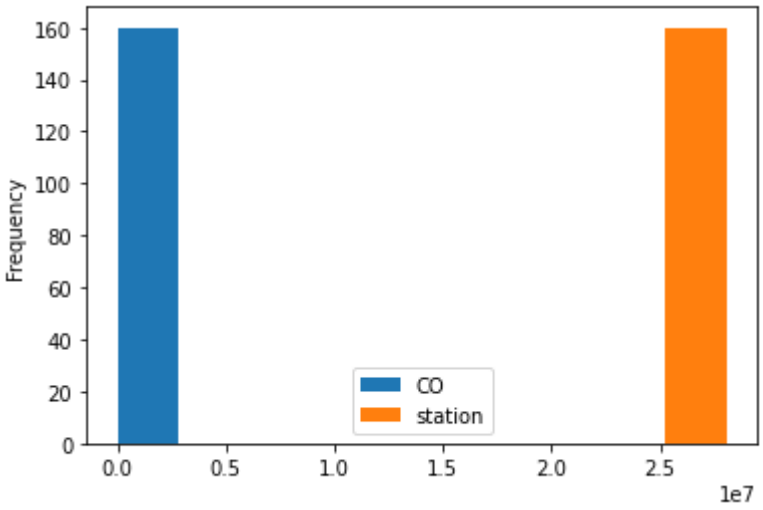


In [12]:

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

Out[12]:

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

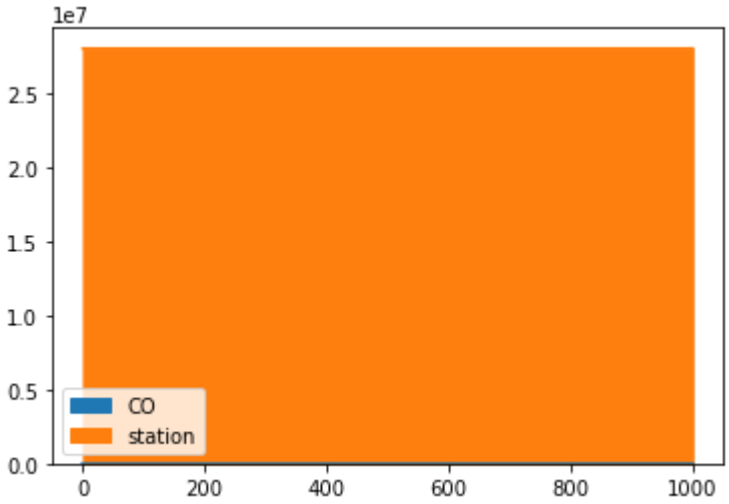


In [13]:

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

Out[13]:

```
<AxesSubplot:>
```

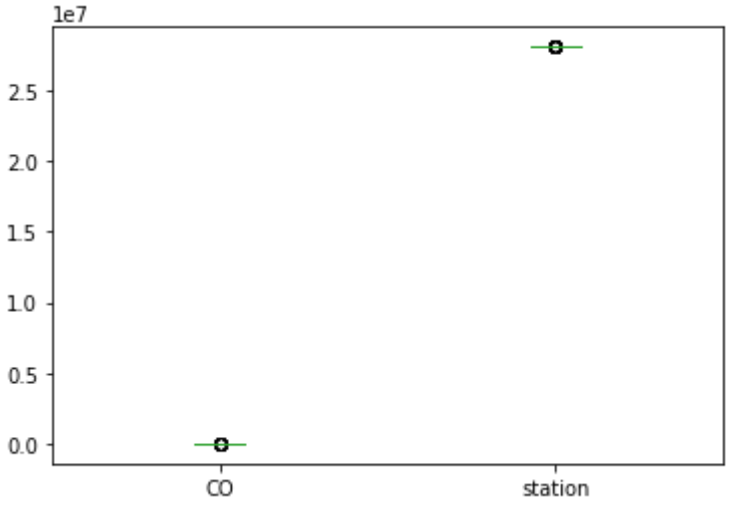


In [14]:

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

Out[14]:

<AxesSubplot:>



In [18]:

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

Out[18]:

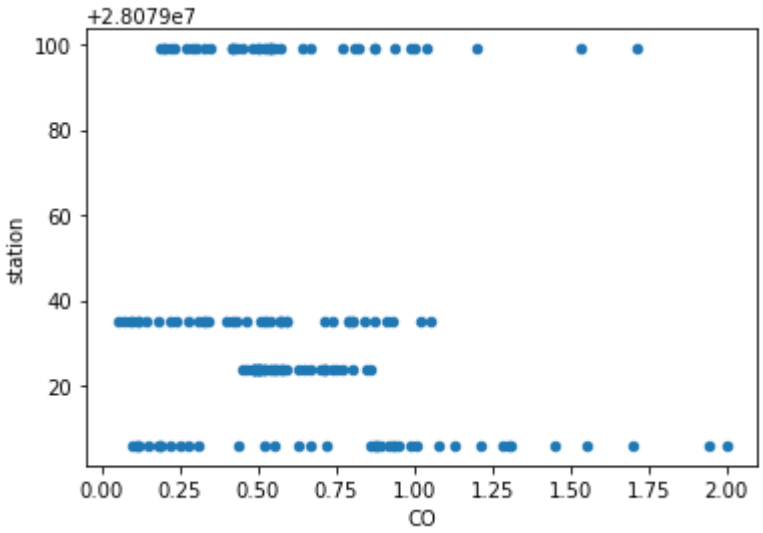
```
<AxesSubplot:ylabel='station'>
```

In [19]:

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

Out[19]:

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



In [20]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 160 entries, 1 to 999
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   date      160 non-null    object
 1   BEN       160 non-null    float64
 2   CO        160 non-null    float64
 3   EBE       160 non-null    float64
 4   MXY       160 non-null    float64
 5   NMHC      160 non-null    float64
 6   NO_2      160 non-null    float64
 7   NOx       160 non-null    float64
 8   OXY       160 non-null    float64
 9   O_3       160 non-null    float64
 10  PM10      160 non-null    float64
 11  PXY       160 non-null    float64
 12  SO_2      160 non-null    float64
 13  TCH       160 non-null    float64
```

In [21]:

```
df.describe()
```

Out[21]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx
count	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000
mean	2.131063	0.622562	2.277000	5.605812	0.125750	65.578250	113.474250
std	1.538625	0.364399	1.811231	4.637598	0.077732	31.284188	79.396254
min	0.450000	0.050000	0.270000	0.410000	0.000000	10.120000	10.370000
25%	0.892500	0.420000	0.910000	1.960000	0.090000	39.662498	45.964999
50%	1.760000	0.540000	1.875000	4.425000	0.120000	69.520000	98.540001
75%	2.877500	0.812500	2.985000	7.672500	0.172500	90.804998	167.650002
max	7.250000	2.000000	7.220000	18.129999	0.460000	130.199997	385.399994

In [22]:

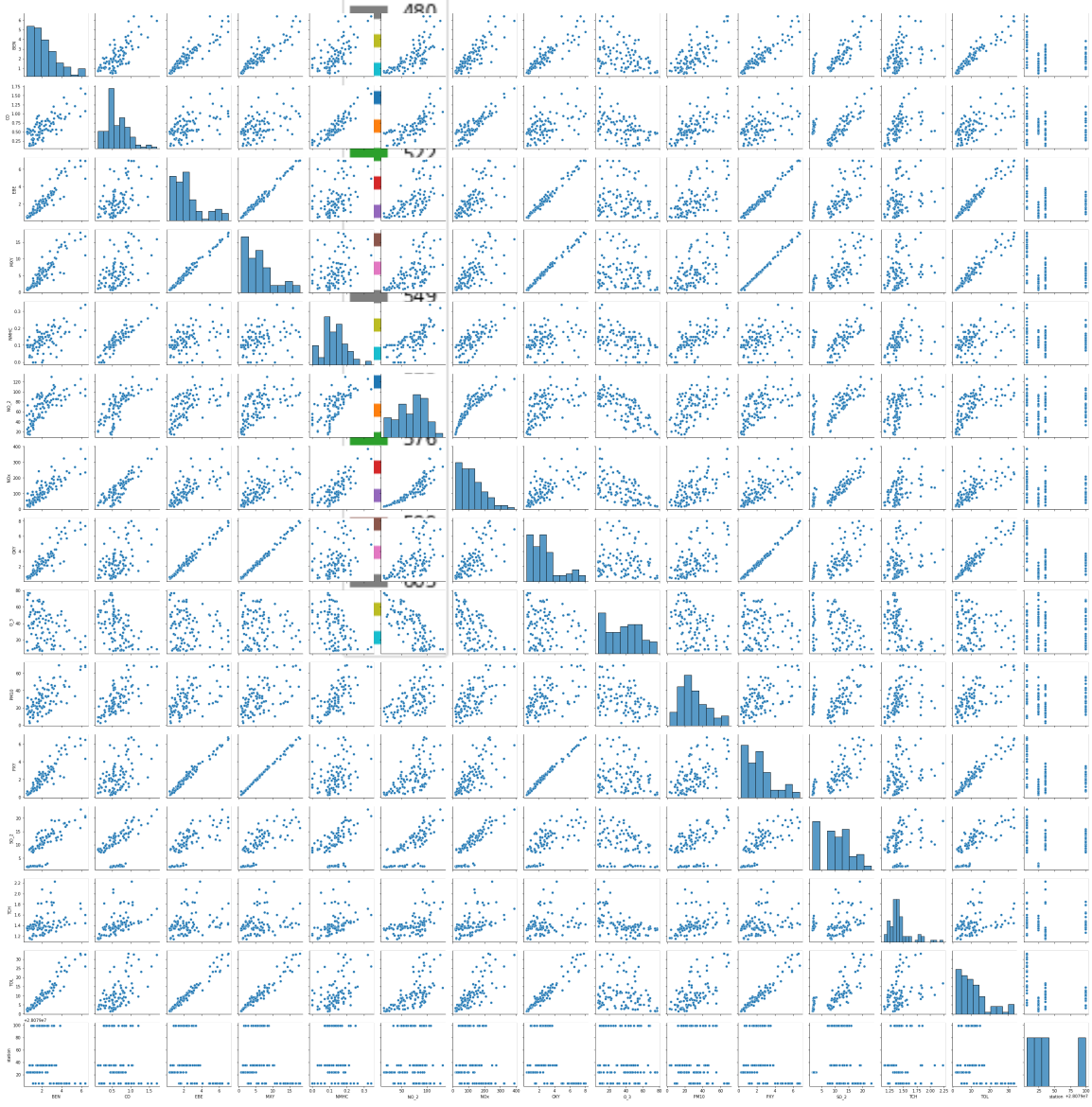
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [25]:

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

Out[25]:

```
<seaborn.axisgrid.PairGrid at 0x280fe2d37c0>
```



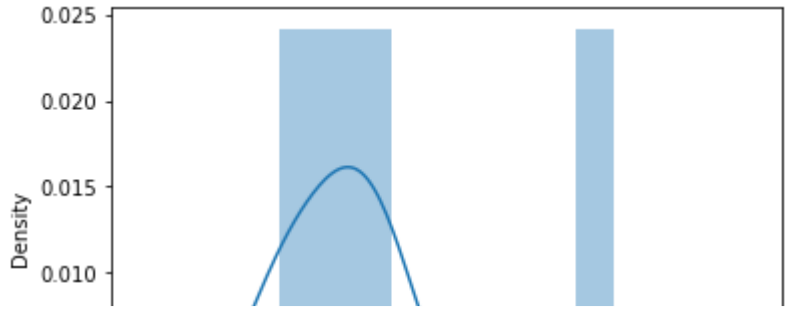
In [26]:

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

```
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[26]:

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

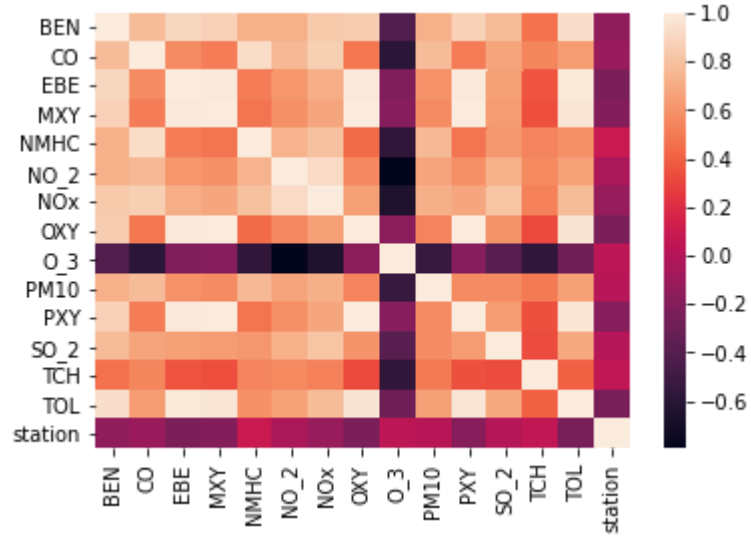


In [28]:

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

Out[28]:

```
<AxesSubplot:>
```



In [29]:

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

In [30]:

```
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 [31]:

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

Out[31]:

```
LinearRegression()
```

In [32]:

```
lr.intercept_
```

Out[32]:

```
28078977.49152369
```

In [33]:

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

Out[33]:

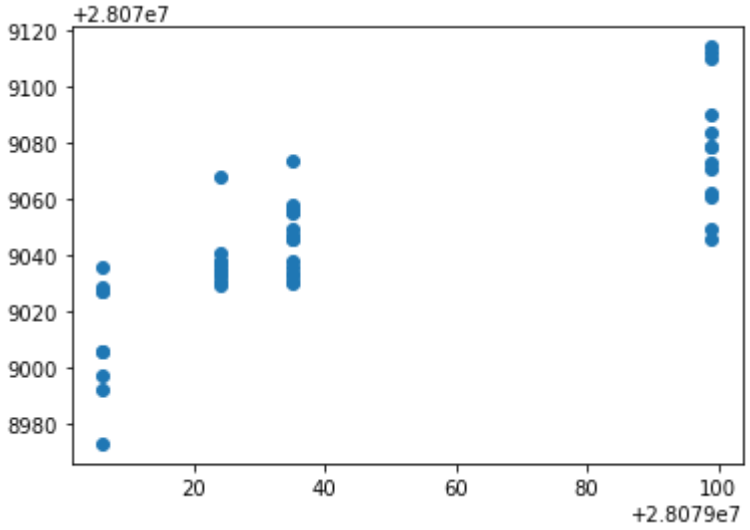
	Co-efficient
BEN	9.040153
CO	-121.059250
EBE	14.620063
MXY	8.756767
NMHC	520.629202
NO_2	0.101393
NOx	-0.092833
OXY	-59.443498
O_3	0.203557
PM10	0.501449
PXY	55.001711
SO_2	1.135682
TCH	37.868582
TOL	-8.195715

In [36]:

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

Out[36]:

```
<matplotlib.collections.PathCollection at 0x2808bfa7f70>
```



In [37]:

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

Out[37]:

```
0.610636280978353
```

In [38]:

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

Out[38]:

```
0.5708897397788171
```

In [39]:

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

In [40]:

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

Out[40]:

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

In [41]:

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

Out[41]:

```
0.43768273788146705
```

In [42]:

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

Out[42]:

```
0.3072546328793606
```

In [43]:

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

Out[43]:

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

In [44]:

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

Out[44]:

```
0.18221165600063582
```

In [45]:

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

Out[45]:

```
0.23336705493564447
```

In [46]:

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

Out[46]:

```
ElasticNet()
```

In [47]:

```
e.coef_
```

Out[47]:

```
array([ 2.44576649, -0.        , -0.        ,  2.43636761,  0.        ,
        0.76346703, -0.21802117, -1.24003822,  0.54178107,  0.77185674,
        0.65929456,  0.92239421,  0.75004521, -4.06549356])
```

In [48]:

```
e.intercept_
```

Out[48]:

```
28078983.000059154
```

In [49]:

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

In [50]:

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

Out[50]:

```
0.3339567703285454
```

In [51]:

```
from sklearn import metrics
```

In [52]:

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

```
799.2446485741735
```

In [53]:

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

```
28.27091524118336
```

In [54]:

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

```
22.51115193388735
```

In [55]:

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

In [56]:

```
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 [57]:

```
feature_matrix.shape
```

Out[57]:

```
(160, 14)
```

In [58]:

```
target_vector.shape
```

Out[58]:

```
(160,)
```

In [59]:

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

In [60]:

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

In [61]:

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

Out[61]:

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

In [62]:

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

In [63]:

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

```
[28079035]
```

In [64]:

```
logr.classes_
```

Out[64]:

```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [65]:

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

Out[65]:

```
0.91875
```

In [66]:

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

Out[66]:

```
0.06614160880263478
```

In [67]:

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

Out[67]:

```
array([[6.61416088e-02, 6.10198057e-27, 9.33856503e-01, 1.88838254e-06]])
```

In [68]:

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

In [69]:

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

Out[69]:

```
RandomForestClassifier()
```

In [70]:

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

In [71]:

```
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[71]:

```
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 [72]:

```
grid_search.best_score_
```

Out[72]:

```
0.75
```

In [73]:

```
rfc_best=grid_search.best_estimator_
```

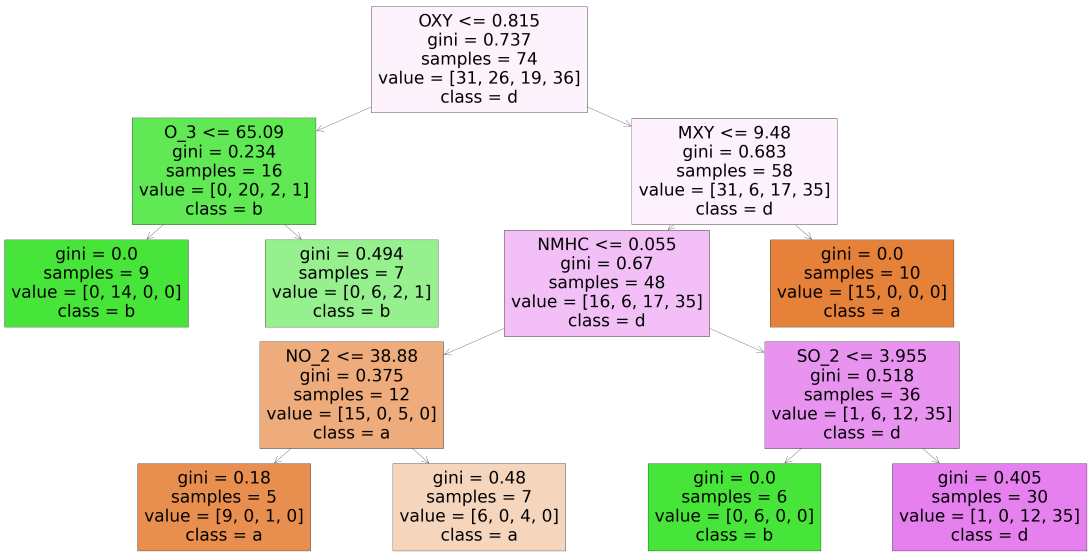
In [75]:

```
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[75]:

```
[Text(1984.0, 1956.96, 'OXY <= 0.815\ngini = 0.737\nsamples = 74\nvalue =
[31, 26, 19, 36]\nclass = d'),
 Text(992.0, 1522.0800000000002, 'O_3 <= 65.09\ngini = 0.234\nsamples = 16
\nvalue = [0, 20, 2, 1]\nclass = b'),
 Text(496.0, 1087.2, 'gini = 0.0\nsamples = 9\nvalue = [0, 14, 0, 0]\nclas
s = b'),
 Text(1488.0, 1087.2, 'gini = 0.494\nsamples = 7\nvalue = [0, 6, 2, 1]\ncl
ass = b'),
 Text(2976.0, 1522.0800000000002, 'MXY <= 9.48\ngini = 0.683\nsamples = 58
\nvalue = [31, 6, 17, 35]\nclass = d'),
 Text(2480.0, 1087.2, 'NMHC <= 0.055\ngini = 0.67\nsamples = 48\nvalue =
[16, 6, 17, 35]\nclass = d'),
 Text(1488.0, 652.3200000000002, 'NO_2 <= 38.88\ngini = 0.375\nsamples = 1
2\nvalue = [15, 0, 5, 0]\nclass = a'),
 Text(992.0, 217.44000000000005, 'gini = 0.18\nsamples = 5\nvalue = [9, 0,
1, 0]\nclass = a'),
 Text(1984.0, 217.44000000000005, 'gini = 0.48\nsamples = 7\nvalue = [6,
0, 4, 0]\nclass = a'),
 Text(3472.0, 652.3200000000002, 'SO_2 <= 3.955\ngini = 0.518\nsamples = 3
6\nvalue = [1, 6, 12, 35]\nclass = d'),
 Text(2976.0, 217.44000000000005, 'gini = 0.0\nsamples = 6\nvalue = [0, 6,
0, 0]\nclass = b'),
 Text(3968.0, 217.44000000000005, 'gini = 0.405\nsamples = 30\nvalue = [1,
0, 12, 35]\nclass = d'),
 Text(3472.0, 1087.2, 'gini = 0.0\nsamples = 10\nvalue = [15, 0, 0, 0]\ncl
ass = a')]
```



logistic regression is suitable for this dataset (0.918750000000000)

In [ ]:

```
```