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_2003.csv")
df = df1.head(1000)
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779
...	...	...	...	...	...	...	...	...	...	...	
995	2003-03-02 12:00:00	NaN	0.62	NaN	NaN	NaN	38.060001	52.660000	NaN	26.180000	11.350
996	2003-03-02 12:00:00	NaN	0.31	NaN	NaN	NaN	16.330000	18.770000	NaN	44.869999	14.100
997	2003-03-02 12:00:00	NaN	0.41	NaN	NaN	0.17	20.590000	26.580000	NaN	45.130001	10.780
998	2003-03-02 12:00:00	NaN	0.44	NaN	NaN	NaN	23.540001	42.919998	NaN	33.930000	18.379
999	2003-03-02 12:00:00	NaN	0.52	NaN	NaN	NaN	31.379999	43.660000	NaN	30.920000	16.770

1000 rows × 16 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', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106 entries, 5 to 985
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     106 non-null    object 
 1   BEN      106 non-null    float64
 2   CO       106 non-null    float64
 3   EBE      106 non-null    float64
 4   MXY      106 non-null    float64
 5   NMHC     106 non-null    float64
 6   NO_2     106 non-null    float64
 7   NOx      106 non-null    float64
 8   OXY      106 non-null    float64
 9   O_3      106 non-null    float64
 10  PM10     106 non-null    float64
 11  PXY      106 non-null    float64
 12  SO_2     106 non-null    float64
 13  TCH      106 non-null    float64
 14  TOL      106 non-null    float64
 15  station  106 non-null    int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 14.1+ KB
```

In [6]:

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

Out[6]:

	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
951	0.45	28079099
957	0.71	28079006
975	0.41	28079024
979	0.50	28079099
985	0.70	28079006

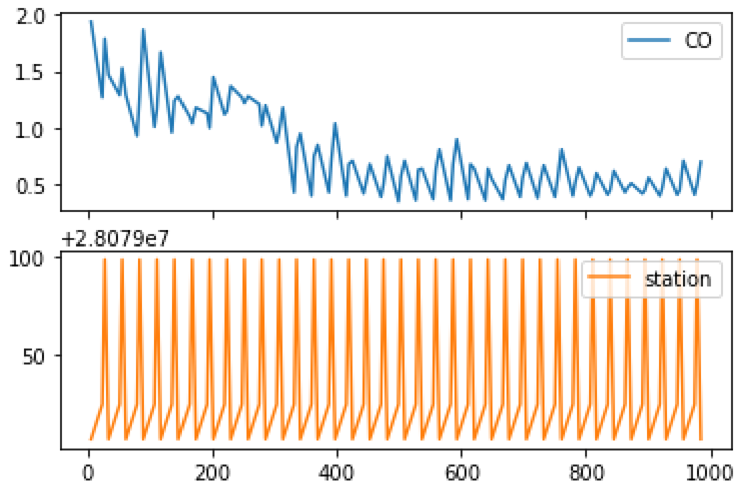
106 rows × 2 columns

In [7]:

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

Out[7]:

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

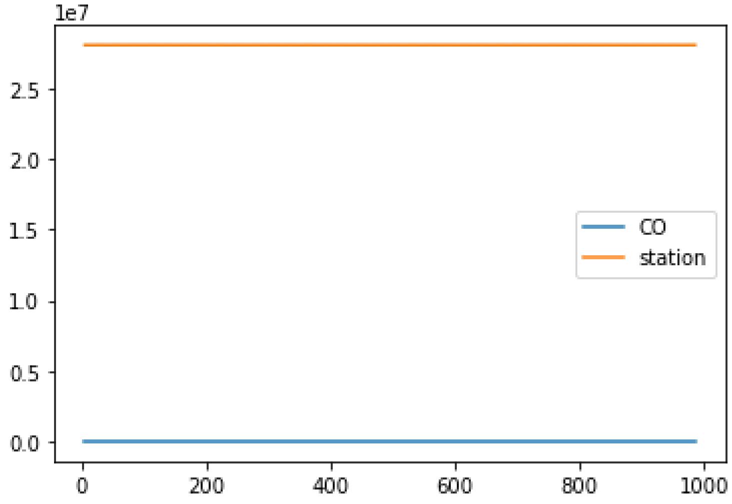


In [8]:

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

Out[8]:

```
<AxesSubplot:>
```



In [9]:

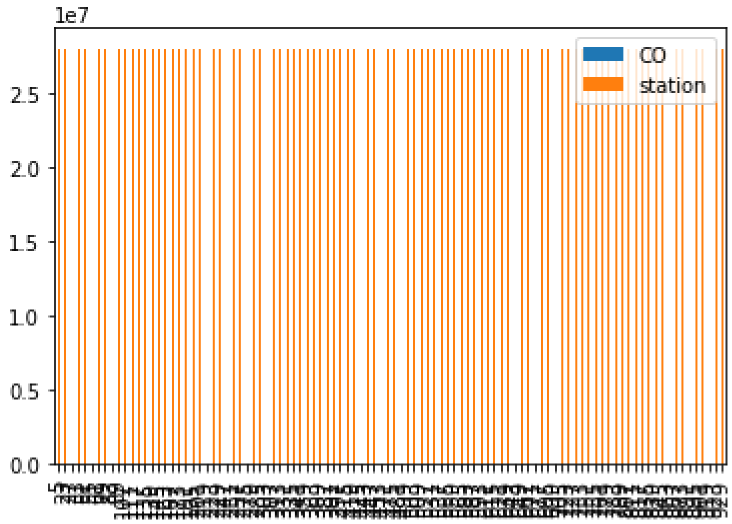
```
x = data[0:100]
```

In [10]:

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

Out[10]:

```
<AxesSubplot:>
```

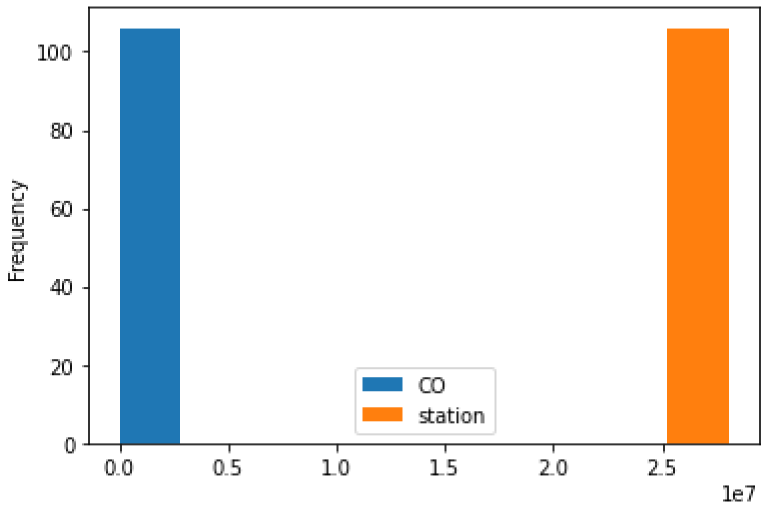


In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>

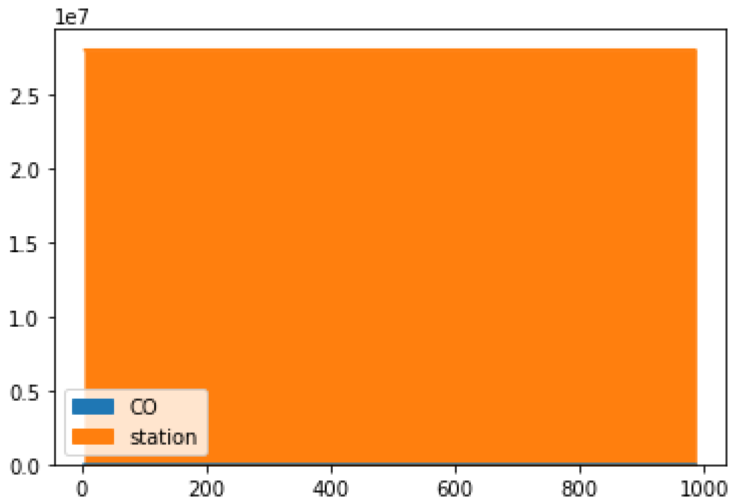


In [12]:

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

Out[12]:

<AxesSubplot:>

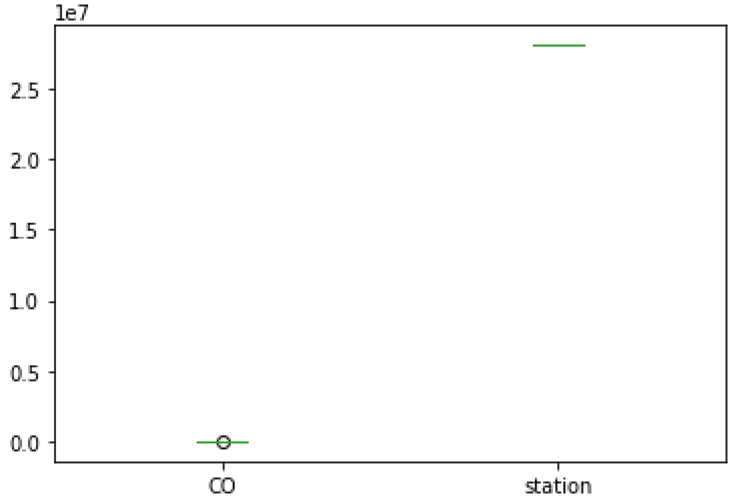


In [13]:

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

Out[13]:

<AxesSubplot:>



In [14]:

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

Out[14]:

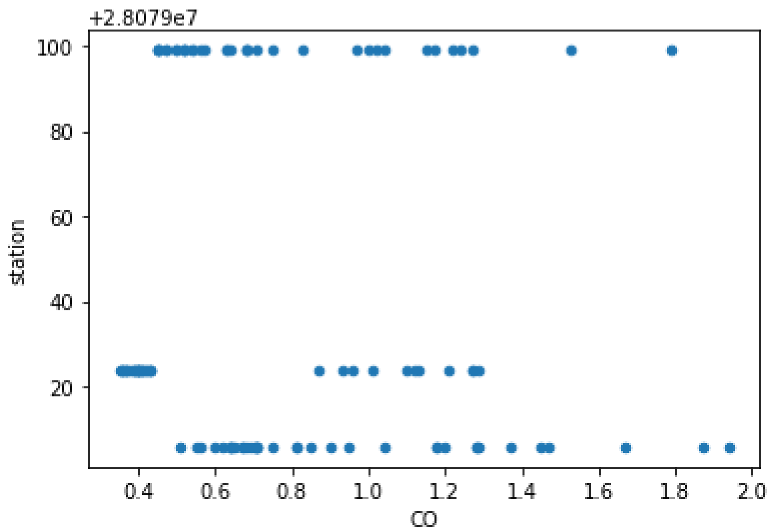
```
<AxesSubplot:ylabel='station'>
```

In [15]:

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

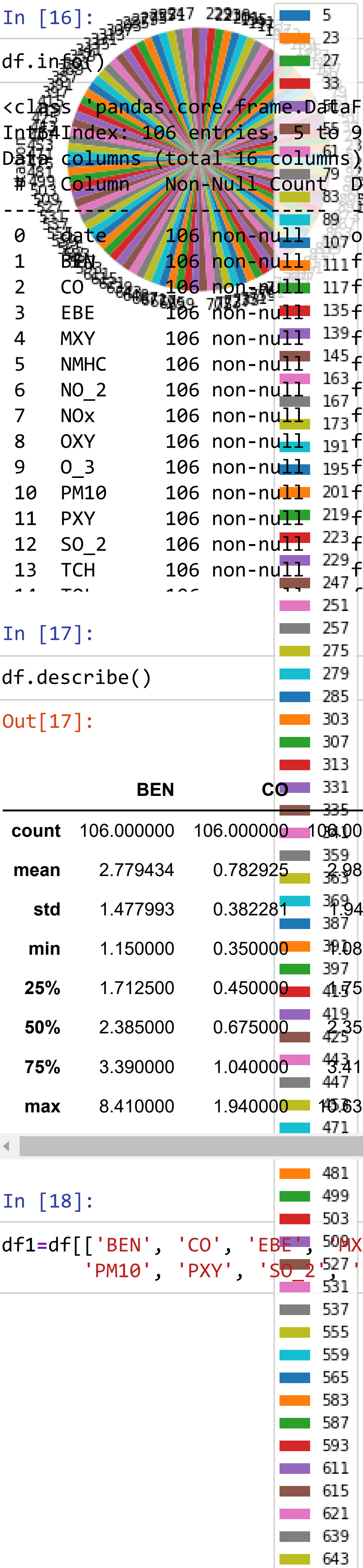
Out[15]:

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



In [16]:

```
df.info()
```




```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106 entries, 5 to 985
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     106 non-null    object
 1   BEN      106 non-null    float64
 2   CO       106 non-null    float64
 3   EBE      106 non-null    float64
 4   MXY      106 non-null    float64
 5   NMHC     106 non-null    float64
 6   NO_2     106 non-null    float64
 7   NOx      106 non-null    float64
 8   OXY      106 non-null    float64
 9   O_3      106 non-null    float64
 10  PM10     106 non-null    float64
 11  PXY      106 non-null    float64
 12  SO_2     106 non-null    float64
 13  TCH      106 non-null    float64
```

In [17]:

```
df.describe()
```

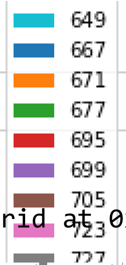
Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx
count	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000	106.000000
mean	2.779434	0.782925	2.981887	6.249340	0.122075	41.216038	105.746509
std	1.477993	0.382281	1.941199	4.741908	0.105746	21.244063	87.204718
min	1.150000	0.350000	1.080000	1.150000	0.000000	4.640000	7.260000
25%	1.712500	0.450000	1.752500	3.190000	0.040000	30.875000	47.670000
50%	2.385000	0.675000	2.350000	4.835000	0.090000	40.709999	79.869999
75%	3.390000	1.040000	3.410000	7.055000	0.190000	57.837500	150.350006
max	8.410000	1.940000	15.630000	24.730000	0.450000	90.300003	384.899994

In [18]:

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

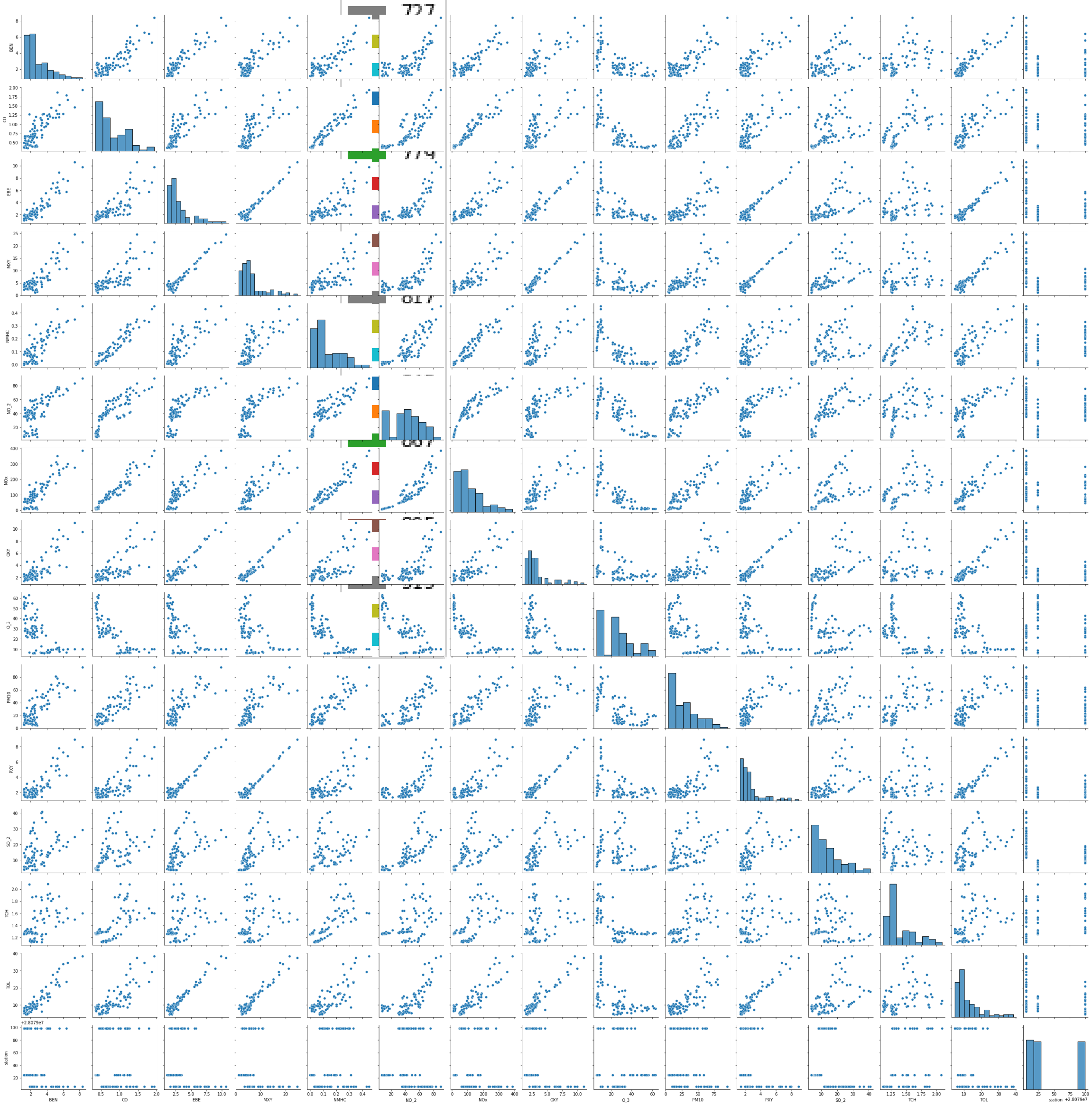
In [19]:




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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x1873c3a98b0>
```



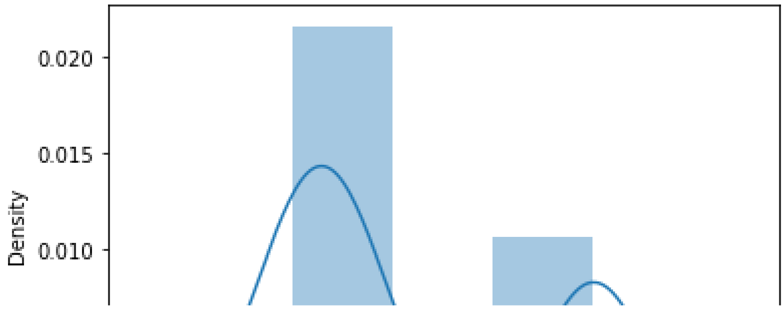
In [20]:

```
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[20]:

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

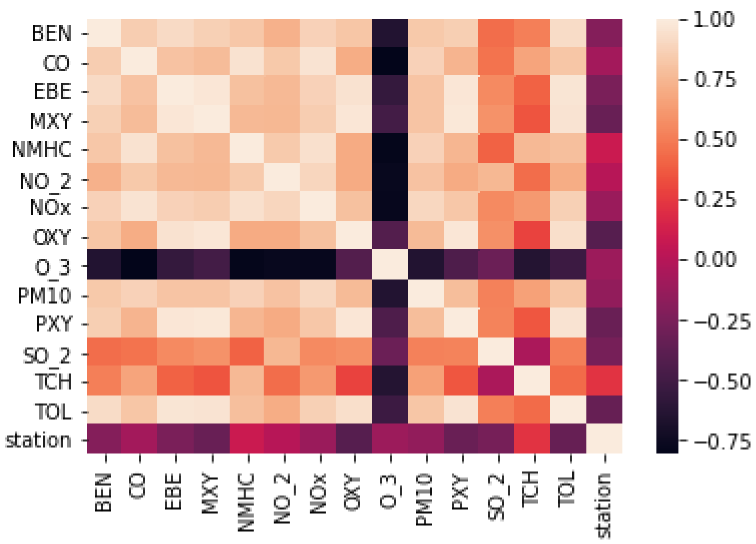


In [21]:

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

Out[21]:

```
<AxesSubplot:>
```



In [22]:

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

In [23]:

```
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 [24]:

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

Out[24]:

```
LinearRegression()
```

In [25]:

```
lr.intercept_
```

Out[25]:

```
28079018.148535907
```

In [26]:

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

Out[26]:

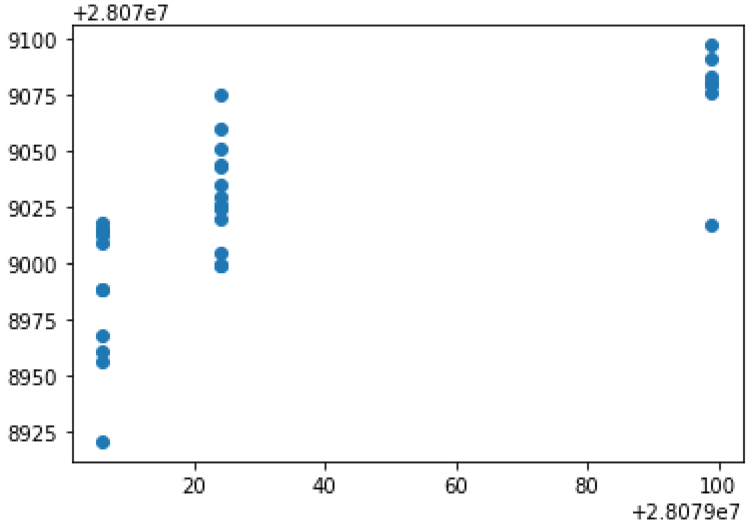
	Co-efficient
BEN	2.388074
CO	-62.598510
EBE	35.421177
MXY	5.476653
NMHC	750.239002
NO_2	1.746939
NOx	-0.606390
OXY	-29.920585
O_3	1.266920
PM10	-0.737225
PXY	-27.537977
SO_2	-1.076649
TCH	12.215117
TOL	-1.483049

In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x187497e3ac0>



In [28]:

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

Out[28]:

0.2490260633695307

In [29]:

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

Out[29]:

0.8499804236425818

In [30]:

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

In [31]:

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

Out[31]:

Ridge(alpha=10)

In [32]:

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

Out[32]:

0.280912622151427

In [33]:

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

Out[33]:

```
0.6577556464942915
```

In [34]:

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

Out[34]:

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

In [35]:

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

Out[35]:

```
0.4989396734836101
```

In [36]:

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

Out[36]:

```
0.28746779780544174
```

In [37]:

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

Out[37]:

```
ElasticNet()
```

In [38]:

```
e.coef_
```

Out[38]:

```
array([ 3.53078775,  0.        ,  3.63333222, -3.36545282,  0.        ,
        3.651599  , -0.0814537 , -7.61063762,  1.35627953, -0.28308469,
       -0.70702439, -4.65747152,  0.9105135 , -0.80954474])
```

In [39]:

```
e.intercept_
```

Out[39]:

```
28078973.84356044
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.27924764285932946
```

In [42]:

```
from sklearn import metrics
```

In [43]:

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

```
891.8740293239543
```

In [44]:

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

```
29.864260066573795
```

In [45]:

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

```
23.20164062792901
```

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]:

```
(106, 14)
```

In [49]:

```
target_vector.shape
```

Out[49]:

```
(106,)
```

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]:

```
0.9905660377358491
```

In [57]:

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

Out[57]:

```
0.9989660875154927
```

In [58]:

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

Out[58]:

```
array([[9.98966088e-01, 8.83359174e-07, 1.03302913e-03]])
```

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]:

```
0.8513513513513513
```

In [64]:

```
rfc_best=grid_search.best_estimator_
```

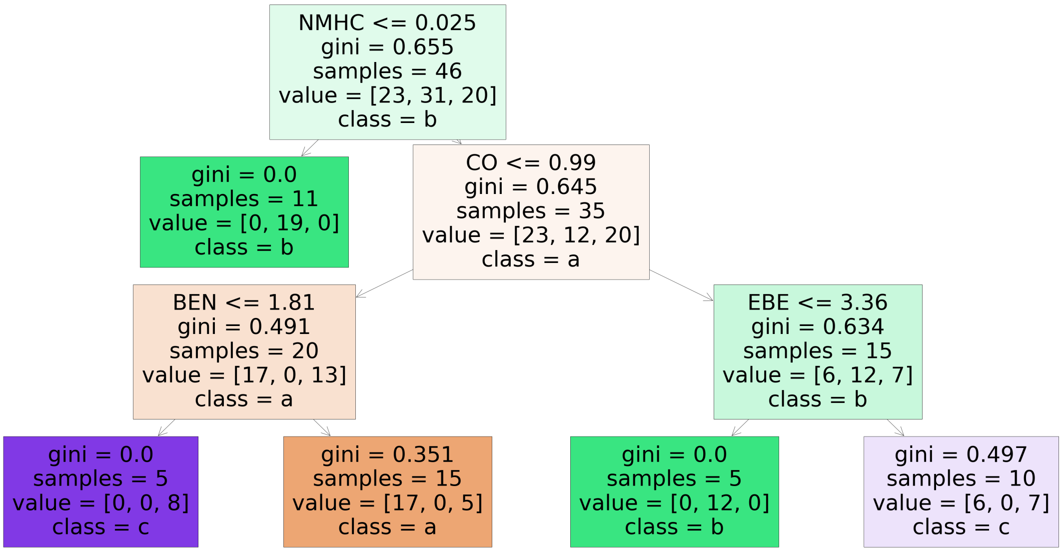
In [65]:

```
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[65]:

```
[Text(1674.0, 1902.6000000000001, 'NMHC <= 0.025\ngini = 0.655\nsamples = 
46\nvalue = [23, 31, 20]\nclass = b'),
 Text(1116.0, 1359.0, 'gini = 0.0\nsamples = 11\nvalue = [0, 19, 0]\nclass 
= b'),
 Text(2232.0, 1359.0, 'CO <= 0.99\ngini = 0.645\nsamples = 35\nvalue = [2
3, 12, 20]\nclass = a'),
 Text(1116.0, 815.4000000000001, 'BEN <= 1.81\ngini = 0.491\nsamples = 20
\nvalue = [17, 0, 13]\nclass = a'),
 Text(558.0, 271.79999999999995, 'gini = 0.0\nsamples = 5\nvalue = [0, 0, 
8]\nclass = c'),
 Text(1674.0, 271.79999999999995, 'gini = 0.351\nsamples = 15\nvalue = [1
7, 0, 5]\nclass = a'),
 Text(3348.0, 815.4000000000001, 'EBE <= 3.36\ngini = 0.634\nsamples = 15
\nvalue = [6, 12, 7]\nclass = b'),
 Text(2790.0, 271.79999999999995, 'gini = 0.0\nsamples = 5\nvalue = [0, 1
2, 0]\nclass = b'),
 Text(3906.0, 271.79999999999995, 'gini = 0.497\nsamples = 10\nvalue = [6, 
0, 7]\nclass = c')]
```



logistic regression is suitable for this dataset (0.918750000000000)

In [ ]: