In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv("madrid_2017.csv")
```

In [3]:
```
df.head()
```

Out[3]:

| | date | BEN | CH4 | CO | EBE | NMHC | NO | NO_2 | NOx | O_3 | PM10 | PM25 | SO_2 | TCH | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2017-06-01 01:00:00 | NaN | NaN | 0.3 | NaN | NaN | 4.0 | 38.0 | NaN | NaN | NaN | NaN | 5.0 | NaN | N |
| **1** | 2017-06-01 01:00:00 | 0.6 | NaN | 0.3 | 0.4 | 0.08 | 3.0 | 39.0 | NaN | 71.0 | 22.0 | 9.0 | 7.0 | 1.4 | : |
| **2** | 2017-06-01 01:00:00 | 0.2 | NaN | NaN | 0.1 | NaN | 1.0 | 14.0 | NaN | NaN | NaN | NaN | NaN | NaN | ( |
| **3** | 2017-06-01 01:00:00 | NaN | NaN | 0.2 | NaN | NaN | 1.0 | 9.0 | NaN | 91.0 | NaN | NaN | NaN | NaN | N |
| **4** | 2017-06-01 01:00:00 | NaN | NaN | NaN | NaN | NaN | 1.0 | 19.0 | NaN | 69.0 | NaN | NaN | 2.0 | NaN | N |

In [4]:
```
df=df.dropna()
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

In [7]:
```
data=df[['CO','station']]
data
```

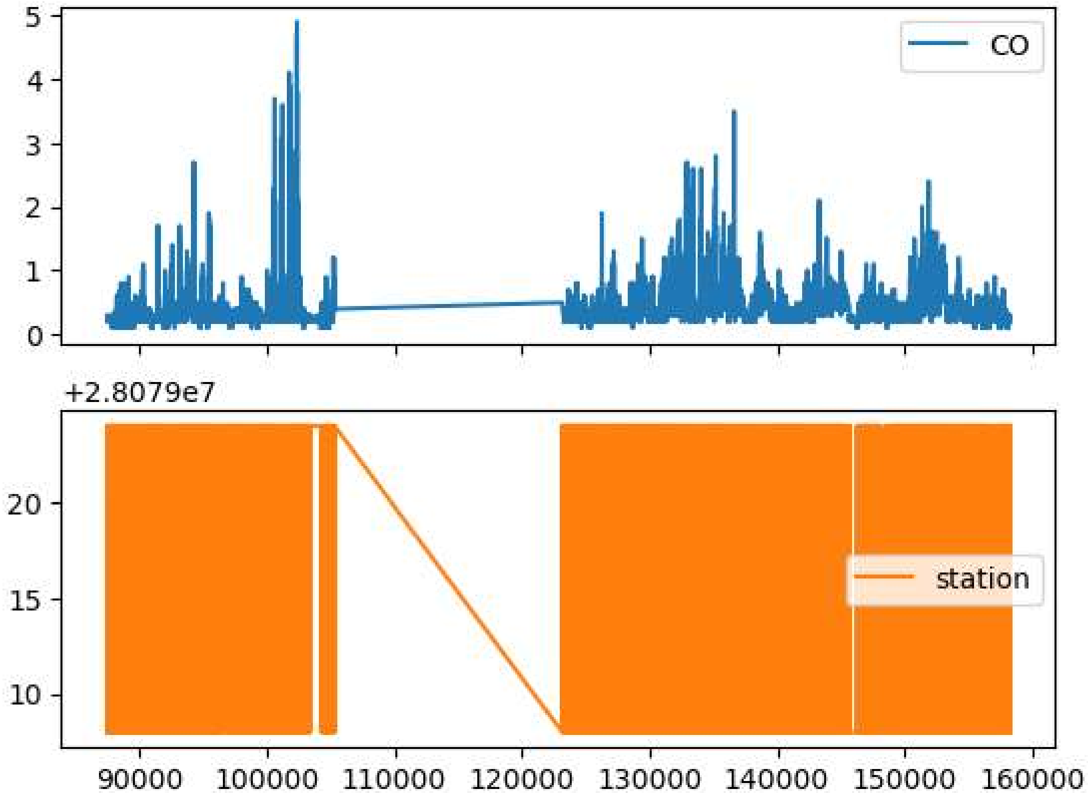
Out[7]:

| | CO | station |
|---|---|---|
| **87457** | 0.3 | 28079008 |
| **87462** | 0.2 | 28079024 |
| **87481** | 0.2 | 28079008 |
| **87486** | 0.2 | 28079024 |
| **87505** | 0.2 | 28079008 |
| **...** | ... | ... |
| **158238** | 0.2 | 28079024 |
| **158257** | 0.3 | 28079008 |
| **158262** | 0.2 | 28079024 |
| **158281** | 0.2 | 28079008 |
| **158286** | 0.2 | 28079024 |

4127 rows × 2 columns

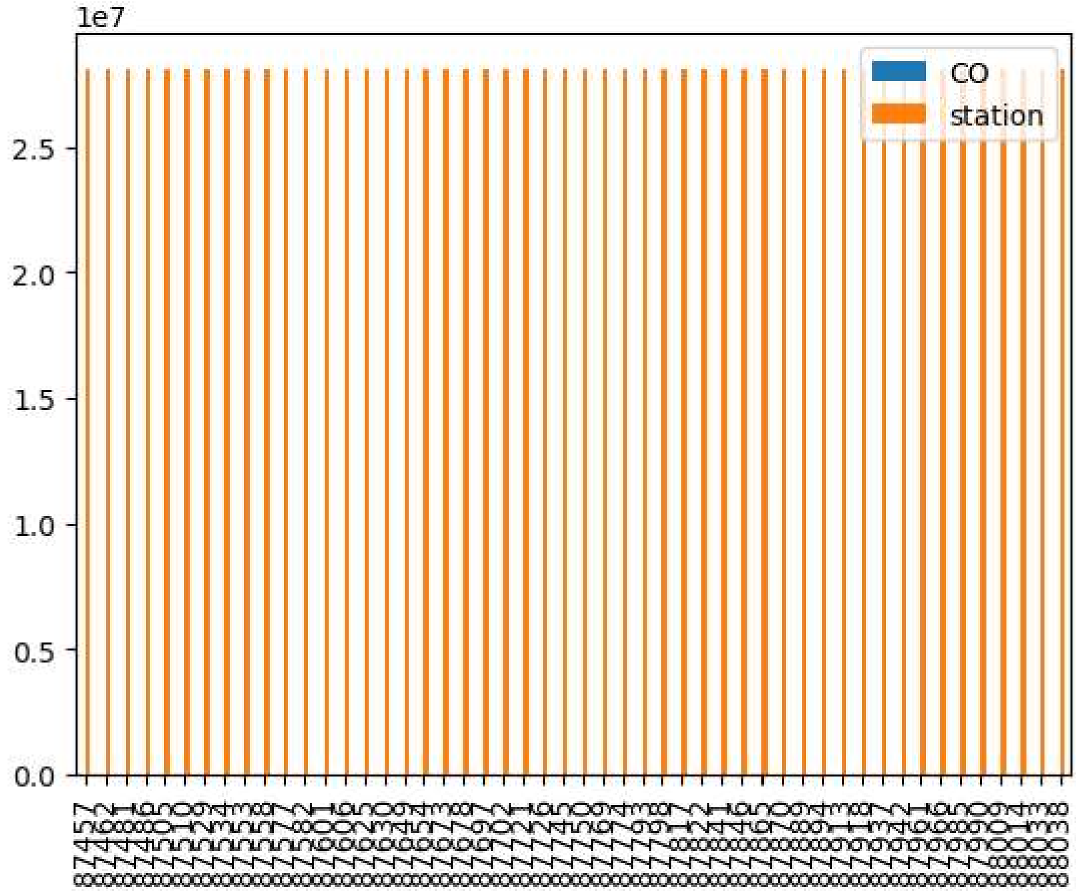
In [8]:
```
data.plot.line(subplots=True)
```

Out[8]:
```
array([<Axes: >, <Axes: >], dtype=object)
```



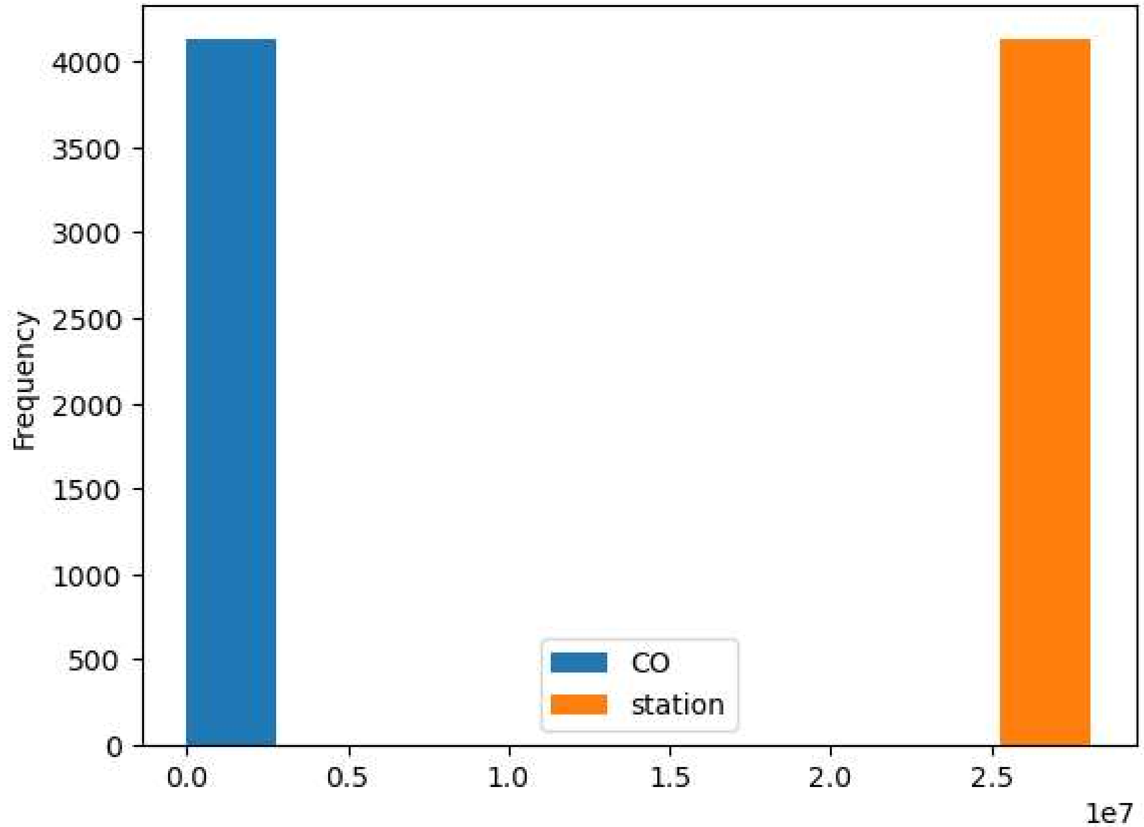
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



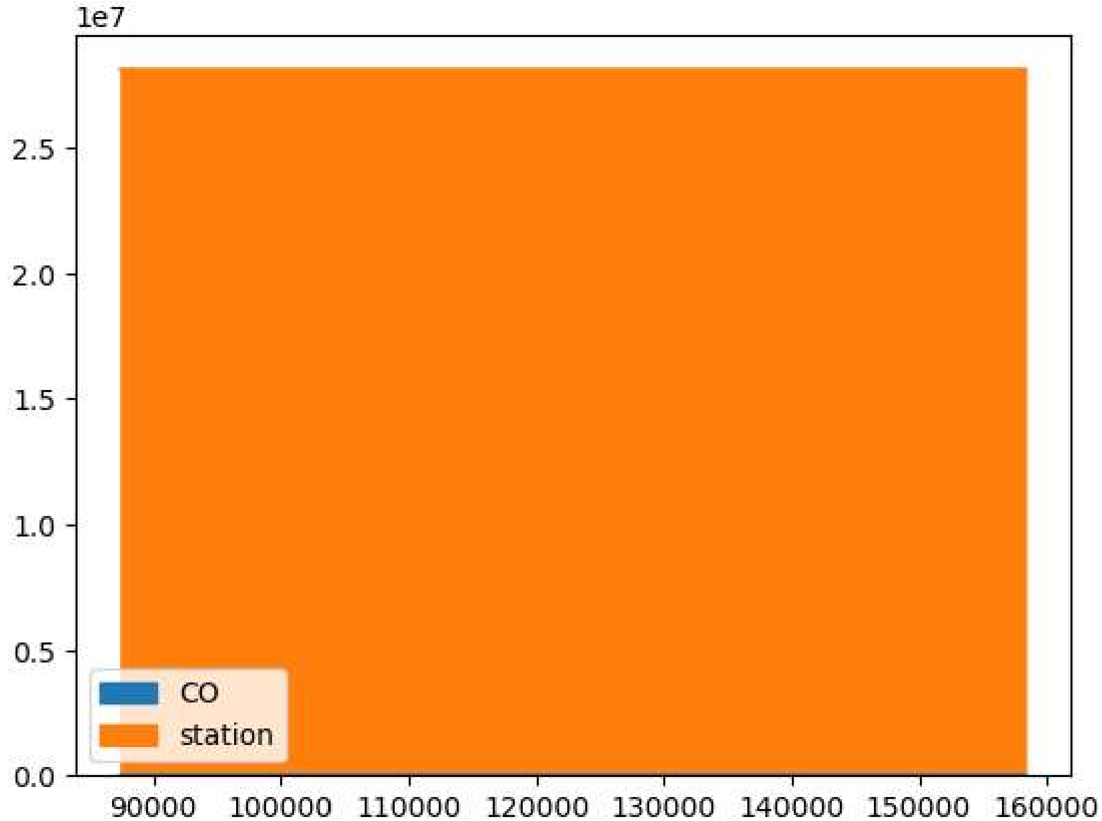
In [10]:
```
data.plot.hist()
```

Out[10]: <Axes: ylabel='Frequency'>



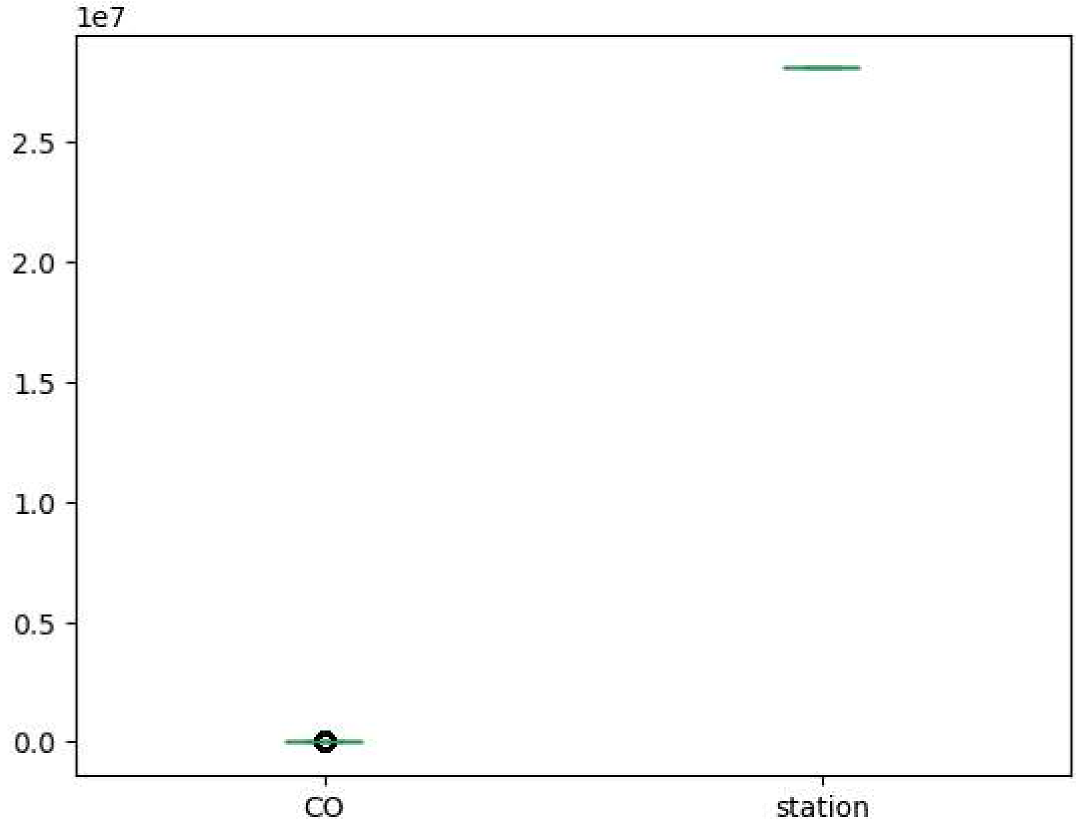
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



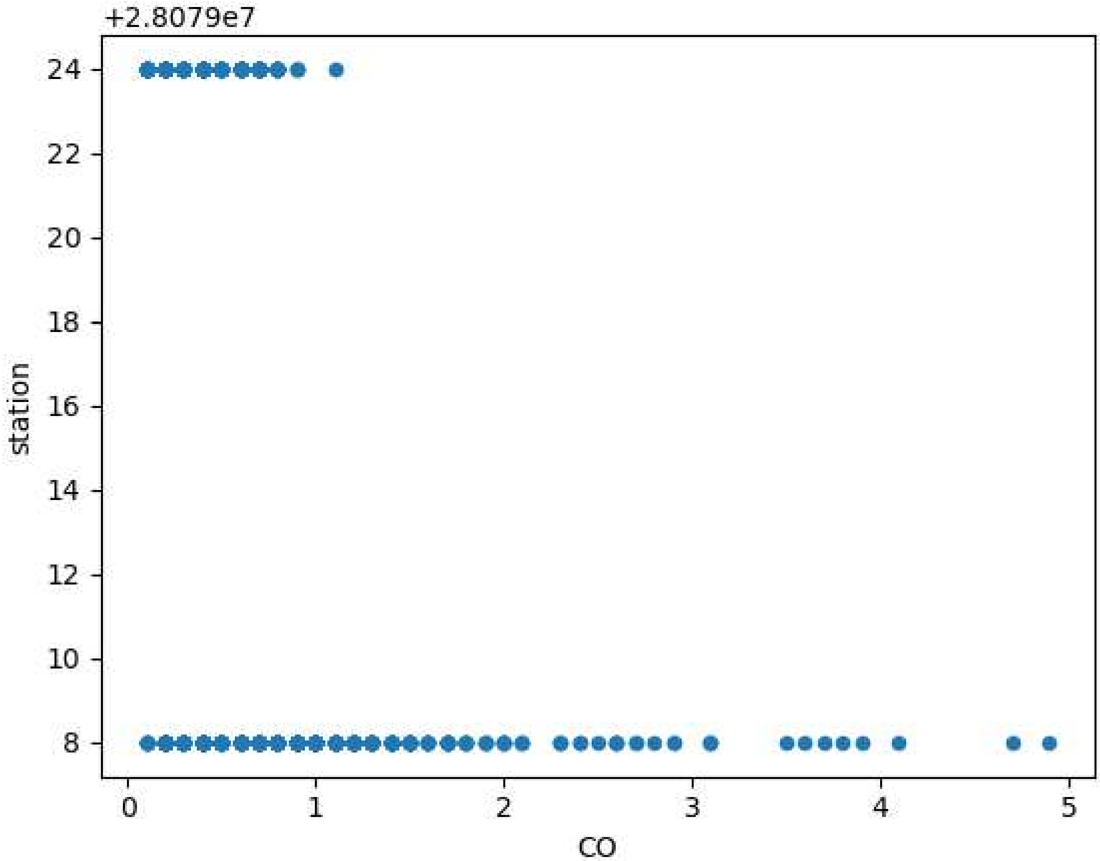
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



In [13]:
```
data.plot.scatter(x='CO',y='station')
```

Out[13]: <Axes: xlabel='CO', ylabel='station'>



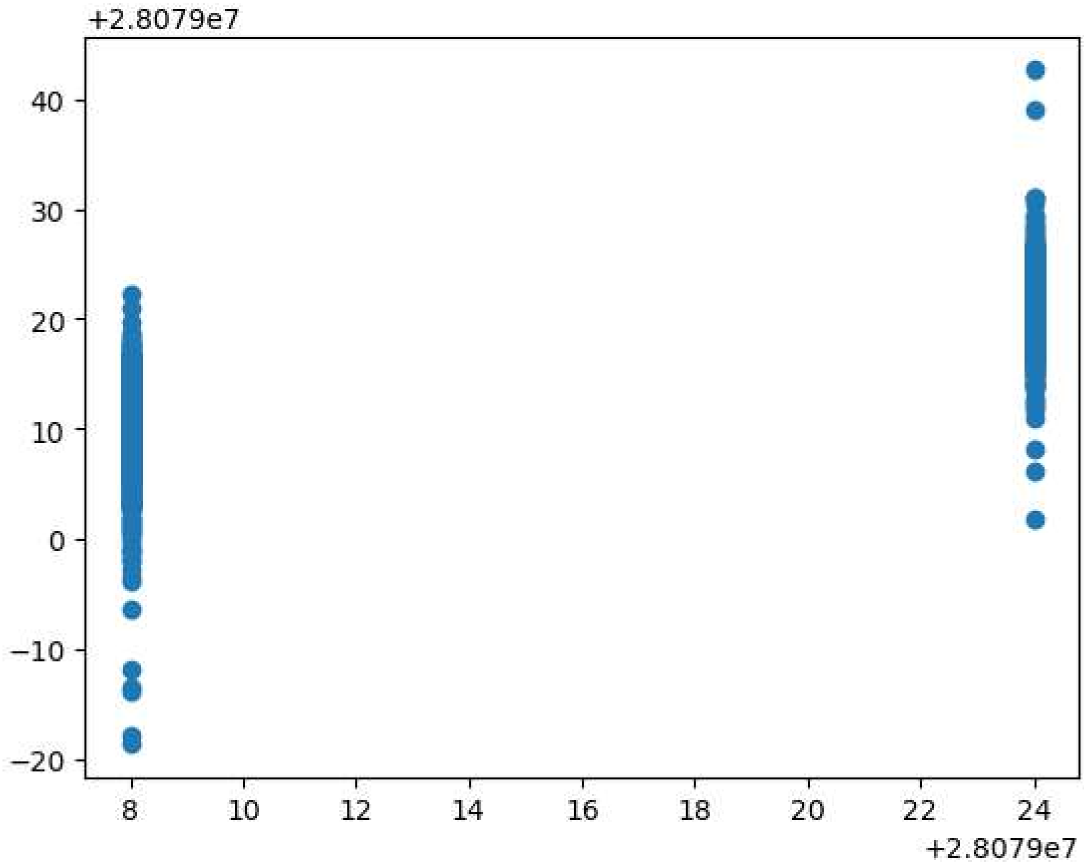

In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [15]:
```
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 [16]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
lr.intercept_
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x19add634190>
```




In [17]:
```
print(lr.score(x_test,y_test))
print(lr.score(x_train,y_train))
```

```
0.6266064332370529
0.6354532898583181
```

# Ridge and Lasso

In [18]:
```
from sklearn.linear_model import Ridge,Lasso
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
print(rr.score(x_test,y_test))
print(rr.score(x_train,y_train))
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

```
0.6267417967400755
0.6255645706561448
```

Out[18]:
```
▾     Lasso
Lasso(alpha=10)
```

In [19]:
```
la.score(x_test,y_test)
```

Out[19]:
```
0.4007652853363365
```

# ElasticNet

In [20]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[20]:
```
▾ ElasticNet
ElasticNet()
```

In [21]:
```
en.coef_
```

Out[21]:
```
array([-0.        , -0.        , -0.        ,  0.        , -0.21033405,
        0.03318752, -0.08640823,  0.52794605, -0.37262767, -0.31282917,
       -0.        ,  0.        ])
```

In [22]:
```
en.intercept_
```

Out[22]:
```
28079025.708727796
```

In [23]:
```
prediction=en.predict(x_test)
```

In [24]:
```
en.score(x_test,y_test)
```

Out[24]:
```
0.5144988330589095
```

# Evaluation Metrics

In [25]:
```
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)))
```

```
4.717827276519896
31.06865398929979
5.57392626335331
```

# Logistics Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
```

In [27]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [28]:
```
from sklearn.preprocessing import StandardScaler
fs=StandardScaler().fit_transform(feature_matrix)
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[28]:
```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

In [29]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12]]
logr.predict_proba(observation)
```

Out[29]:
```
array([[1.00000000e+00, 1.00193473e-10]])
```

# Random Forest

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[30]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [31]:
```
parameters={'max_depth':[1,2,3,4,5],
'min_samples_leaf':[5,10,15,20,25],
'n_estimators':[10,20,30,40,50]
}
```

In [32]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

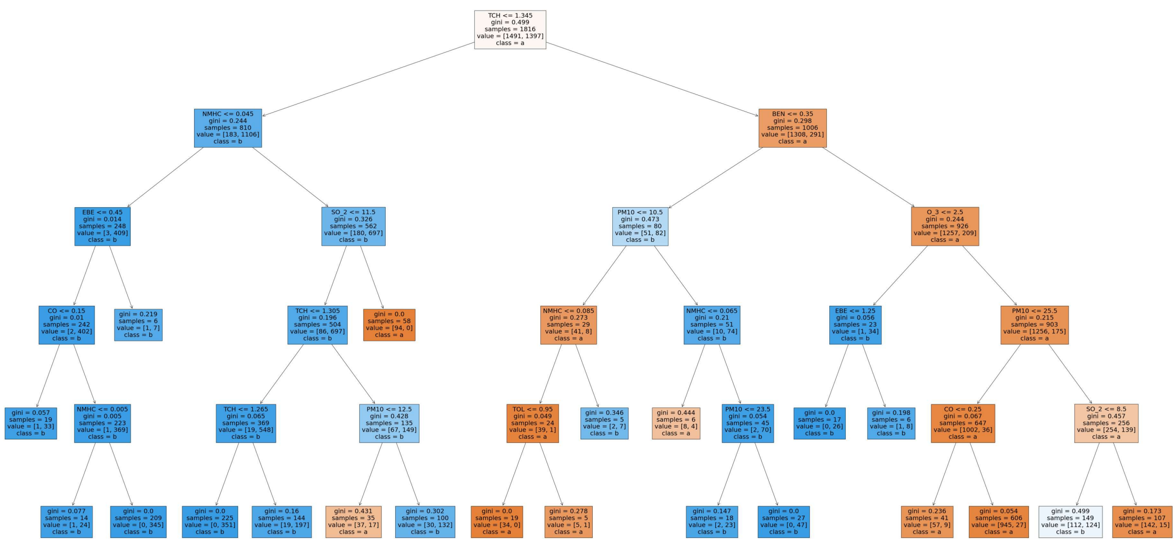
Out[32]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [33]:
```
rfc_best=grid_search.best_estimator_
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'
```

```
Out[33]: [Text(0.4356060606060606, 0.9166666666666666, 'TCH <= 1.345\ngini = 0.499\nsa
mples = 1816\nvalue = [1491, 1397]\nclass = a'),
 Text(0.19696969696969696, 0.75, 'NMHC <= 0.045\ngini = 0.244\nsamples = 810
\nvalue = [183, 1106]\nclass = b'),
 Text(0.09090909090909091, 0.5833333333333334, 'EBE <= 0.45\ngini = 0.014\nsa
mples = 248\nvalue = [3, 409]\nclass = b'),
 Text(0.06060606060606061, 0.4166666666666667, 'CO <= 0.15\ngini = 0.01\nsamp
les = 242\nvalue = [2, 402]\nclass = b'),
 Text(0.030303030303030304, 0.25, 'gini = 0.057\nsamples = 19\nvalue = [1, 3
3]\nclass = b'),
 Text(0.09090909090909091, 0.25, 'NMHC <= 0.005\ngini = 0.005\nsamples = 223
\nvalue = [1, 369]\nclass = b'),
 Text(0.06060606060606061, 0.08333333333333333, 'gini = 0.077\nsamples = 14\n
value = [1, 24]\nclass = b'),
 Text(0.12121212121212122, 0.08333333333333333, 'gini = 0.0\nsamples = 209\nv
alue = [0, 345]\nclass = b'),
 Text(0.12121212121212122, 0.4166666666666667, 'gini = 0.219\nsamples = 6\nva
lue = [1, 7]\nclass = b'),
 Text(0.30303030303030304, 0.5833333333333334, 'SO_2 <= 11.5\ngini = 0.326\ns
amples = 562\nvalue = [180, 697]\nclass = b'),
 Text(0.2727272727272727, 0.4166666666666667, 'TCH <= 1.305\ngini = 0.196\nsa
mples = 504\nvalue = [86, 697]\nclass = b'),
 Text(0.21212121212121213, 0.25, 'TCH <= 1.265\ngini = 0.065\nsamples = 369\n
value = [19, 548]\nclass = b'),
 Text(0.18181818181818182, 0.08333333333333333, 'gini = 0.0\nsamples = 225\nv
alue = [0, 351]\nclass = b'),
 Text(0.24242424242424243, 0.08333333333333333, 'gini = 0.16\nsamples = 144\n
value = [19, 197]\nclass = b'),
 Text(0.3333333333333333, 0.25, 'PM10 <= 12.5\ngini = 0.428\nsamples = 135\nv
alue = [67, 149]\nclass = b'),
 Text(0.30303030303030304, 0.08333333333333333, 'gini = 0.431\nsamples = 35\n
value = [37, 17]\nclass = a'),
 Text(0.36363636363636365, 0.08333333333333333, 'gini = 0.302\nsamples = 100
\nvalue = [30, 132]\nclass = b'),
 Text(0.3333333333333333, 0.4166666666666667, 'gini = 0.0\nsamples = 58\nvalu
e = [94, 0]\nclass = a'),
 Text(0.6742424242424242, 0.75, 'BEN <= 0.35\ngini = 0.298\nsamples = 1006\nv
alue = [1308, 291]\nclass = a'),
 Text(0.5454545454545454, 0.5833333333333334, 'PM10 <= 10.5\ngini = 0.473\nsa
mples = 80\nvalue = [51, 82]\nclass = b'),
 Text(0.48484848484848486, 0.4166666666666667, 'NMHC <= 0.085\ngini = 0.273\n
samples = 29\nvalue = [41, 8]\nclass = a'),
 Text(0.45454545454545453, 0.25, 'TOL <= 0.95\ngini = 0.049\nsamples = 24\nva
lue = [39, 1]\nclass = a'),
 Text(0.42424242424242425, 0.08333333333333333, 'gini = 0.0\nsamples = 19\nva
lue = [34, 0]\nclass = a'),
 Text(0.48484848484848486, 0.08333333333333333, 'gini = 0.278\nsamples = 5\nv
alue = [5, 1]\nclass = a'),
 Text(0.5151515151515151, 0.25, 'gini = 0.346\nsamples = 5\nvalue = [2, 7]\nc
lass = b'),
 Text(0.6060606060606061, 0.4166666666666667, 'NMHC <= 0.065\ngini = 0.21\nsa
mples = 51\nvalue = [10, 74]\nclass = b'),
 Text(0.5757575757575758, 0.25, 'gini = 0.444\nsamples = 6\nvalue = [8, 4]\nc
lass = a'),
 Text(0.6363636363636364, 0.25, 'PM10 <= 23.5\ngini = 0.054\nsamples = 45\nva
lue = [2, 70]\nclass = b'),
 Text(0.6060606060606061, 0.08333333333333333, 'gini = 0.147\nsamples = 18\nv
```

```
alue = [2, 23]\nclass = b'),
 Text(0.6666666666666666, 0.08333333333333333, 'gini = 0.0\nsamples = 27\nval
ue = [0, 47]\nclass = b'),
 Text(0.803030303030303, 0.5833333333333334, 'O_3 <= 2.5\ngini = 0.244\nsampl
es = 926\nvalue = [1257, 209]\nclass = a'),
 Text(0.7272727272727273, 0.4166666666666667, 'EBE <= 1.25\ngini = 0.056\nsam
ples = 23\nvalue = [1, 34]\nclass = b'),
 Text(0.696969696969697, 0.25, 'gini = 0.0\nsamples = 17\nvalue = [0, 26]\ncl
ass = b'),
 Text(0.7575757575757576, 0.25, 'gini = 0.198\nsamples = 6\nvalue = [1, 8]\nc
lass = b'),
 Text(0.8787878787878788, 0.4166666666666667, 'PM10 <= 25.5\ngini = 0.215\nsa
mples = 903\nvalue = [1256, 175]\nclass = a'),
 Text(0.8181818181818182, 0.25, 'CO <= 0.25\ngini = 0.067\nsamples = 647\nval
ue = [1002, 36]\nclass = a'),
 Text(0.7878787878787878, 0.08333333333333333, 'gini = 0.236\nsamples = 41\nv
alue = [57, 9]\nclass = a'),
 Text(0.8484848484848485, 0.08333333333333333, 'gini = 0.054\nsamples = 606\n
value = [945, 27]\nclass = a'),
 Text(0.9393939393939394, 0.25, 'SO_2 <= 8.5\ngini = 0.457\nsamples = 256\nva
lue = [254, 139]\nclass = a'),
 Text(0.9090909090909091, 0.08333333333333333, 'gini = 0.499\nsamples = 149\n
value = [112, 124]\nclass = b'),
 Text(0.9696969696969697, 0.08333333333333333, 'gini = 0.173\nsamples = 107\n
value = [142, 15]\nclass = a')]
```



# Conclusion

In [34]:
```
print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
print("Random Forest:",grid_search.best_score_)
```

```
Linear Regression: 0.6266064332370529
Ridge Regression: 0.6267417967400755
Lasso Regression 0.4007652853363365
ElasticNet Regression: 0.5144988330589095
Logistic Regression: 0.9520232614489944
Random Forest: 0.96398891966759
```

# Random Forest Is Better!!!