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_2015.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	stati
0	2015-10-01 01:00:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	280790
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	280790
2	2015-10-01 01:00:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	280790
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	280790
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	280790

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

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

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     16026 non-null  object
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

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

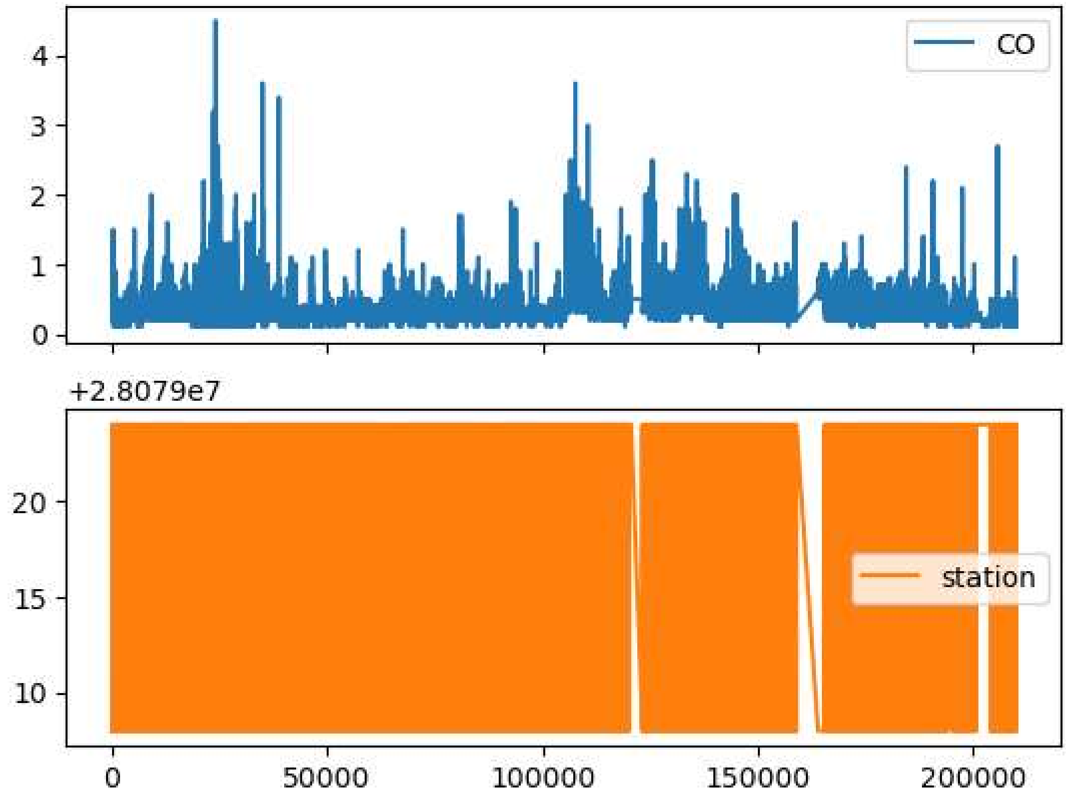
Out[7]:

	CO	station
1	0.8	28079008
6	0.3	28079024
25	0.7	28079008
30	0.3	28079024
49	0.8	28079008
...	...	...
210030	0.1	28079024
210049	0.3	28079008
210054	0.1	28079024
210073	0.3	28079008
210078	0.1	28079024

16026 rows × 2 columns

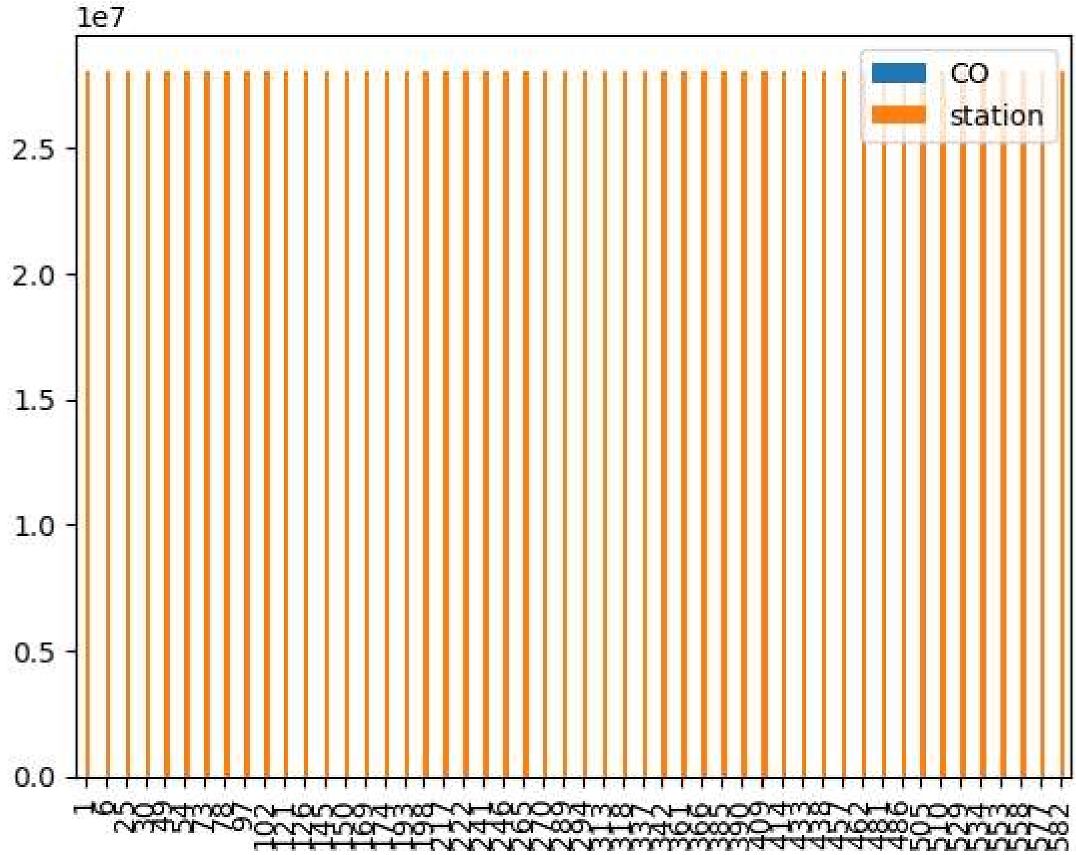
In [8]:
```
data.plot.line(subplots=True)
```

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



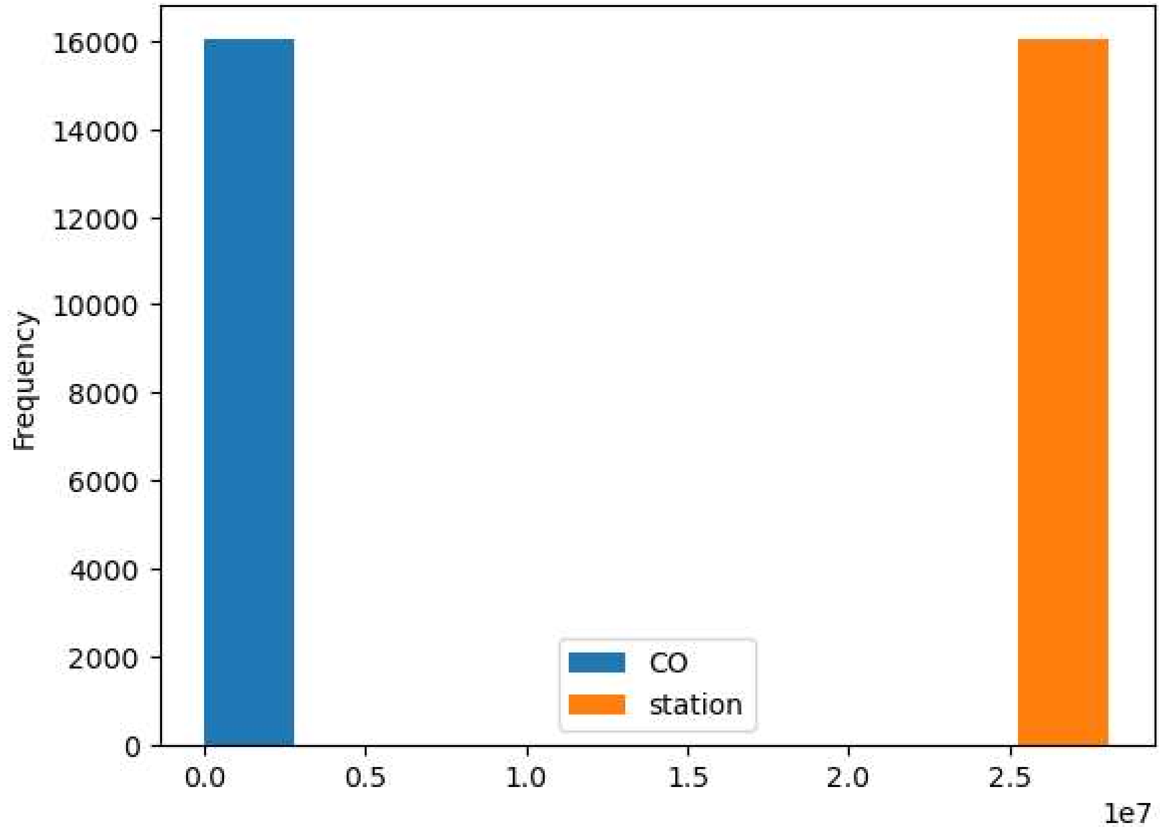
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



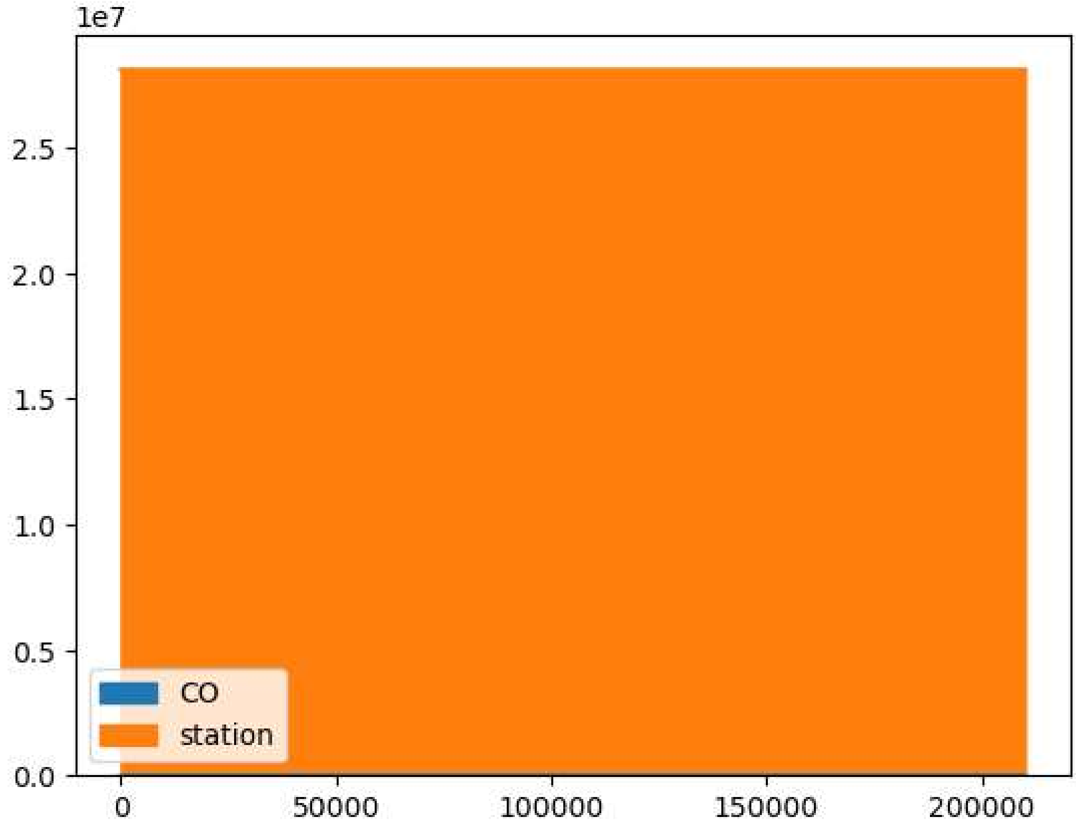
In [10]:
```
data.plot.hist()
```

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



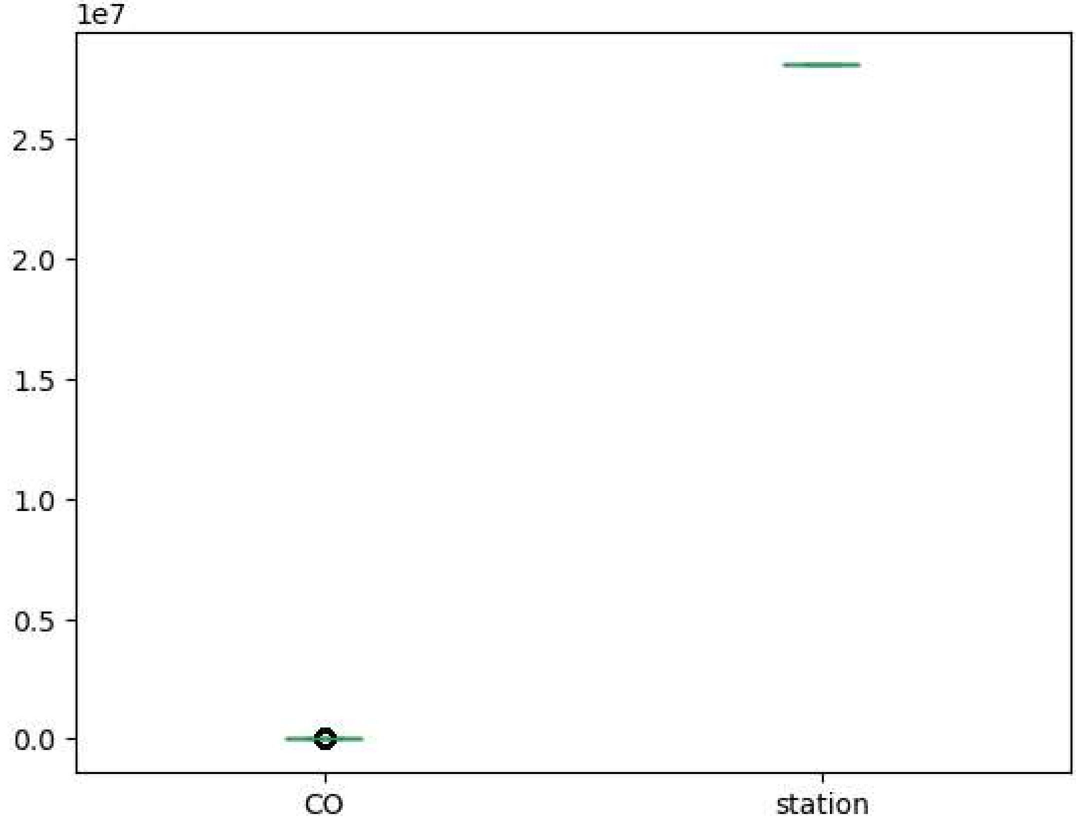
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



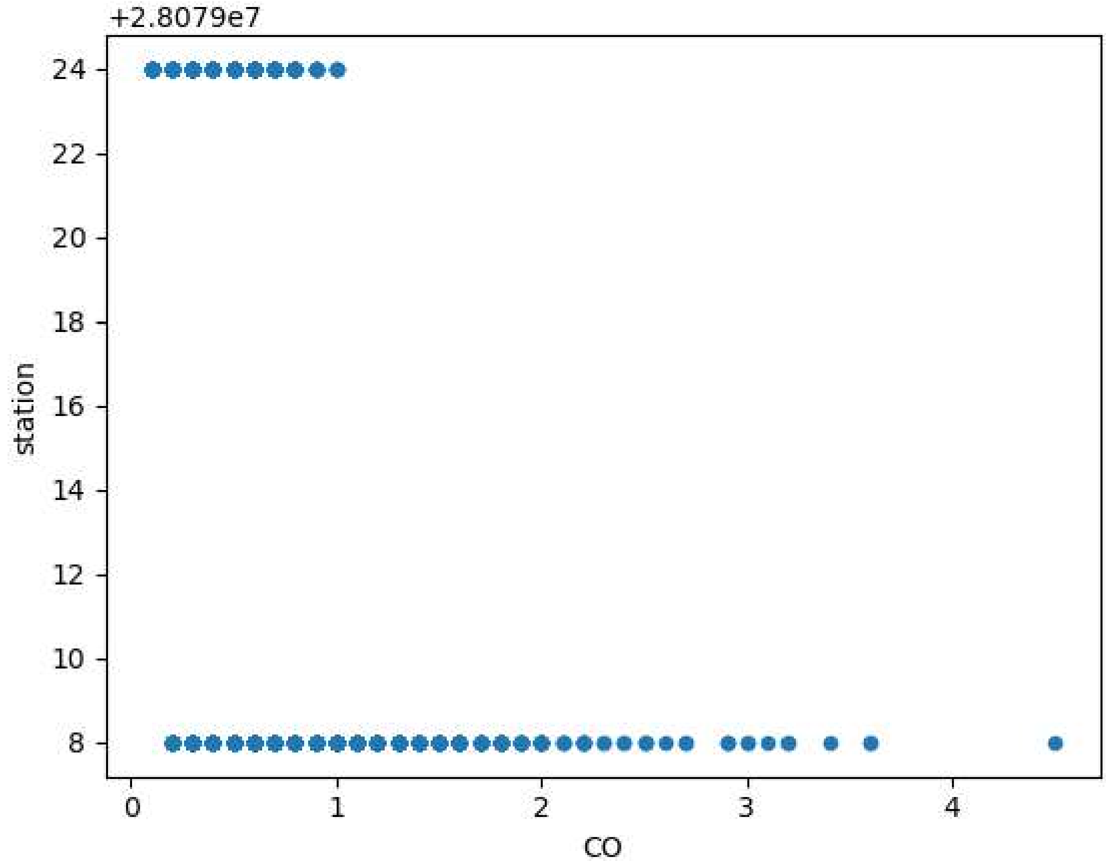
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



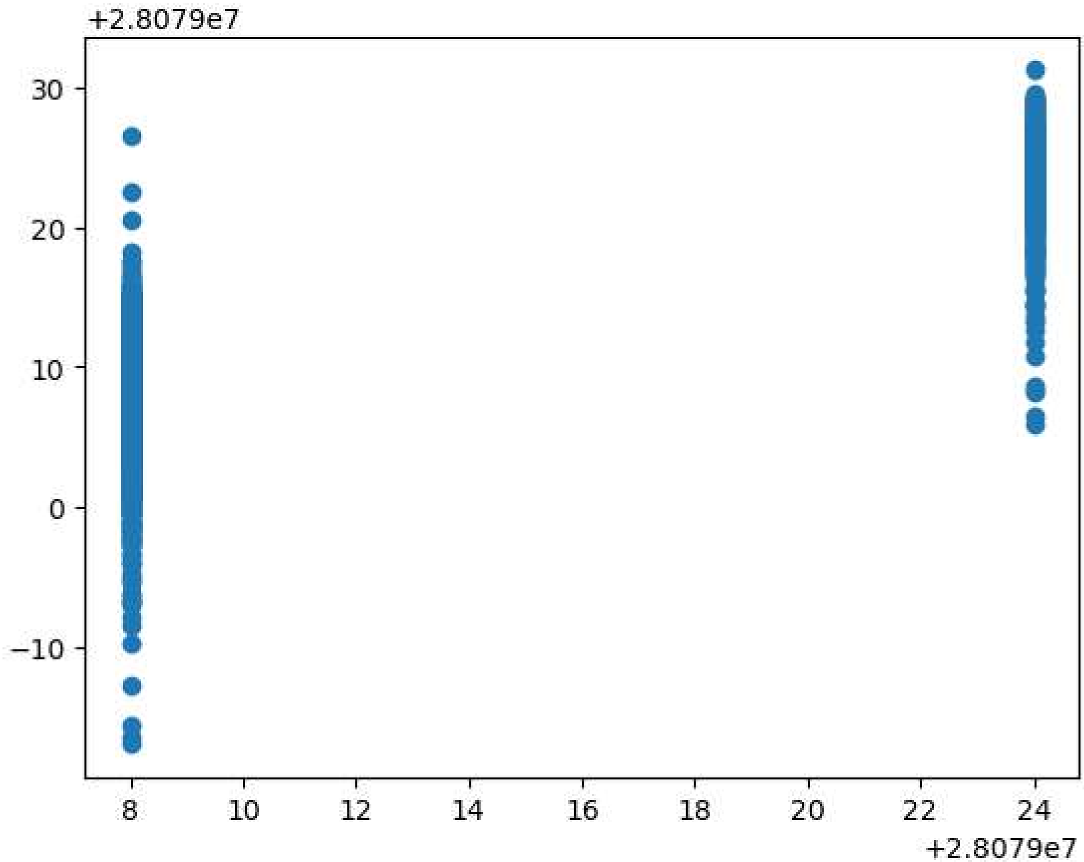
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1afef4b2fd0>
```



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

```
0.8628532861459661
0.8753082441077431
```

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)
```

```
0.86360150367386
0.8743848257290687
```

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

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

Out[19]:
```
0.7249293275039095
```

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.        , -0.05331612,
        0.07546272, -0.01119862,  0.01660429,  0.05265972, -1.31359866,
       -0.        , -0.07270684])
```

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

Out[22]:
```
28079025.97322588
```

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

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

Out[24]:
```
0.818659081862931
```

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)))
```

```
2.5414626818272703
11.598828218265224
3.4057052453589143
```

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, 9.56732829e-34]])
```

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]
}
```

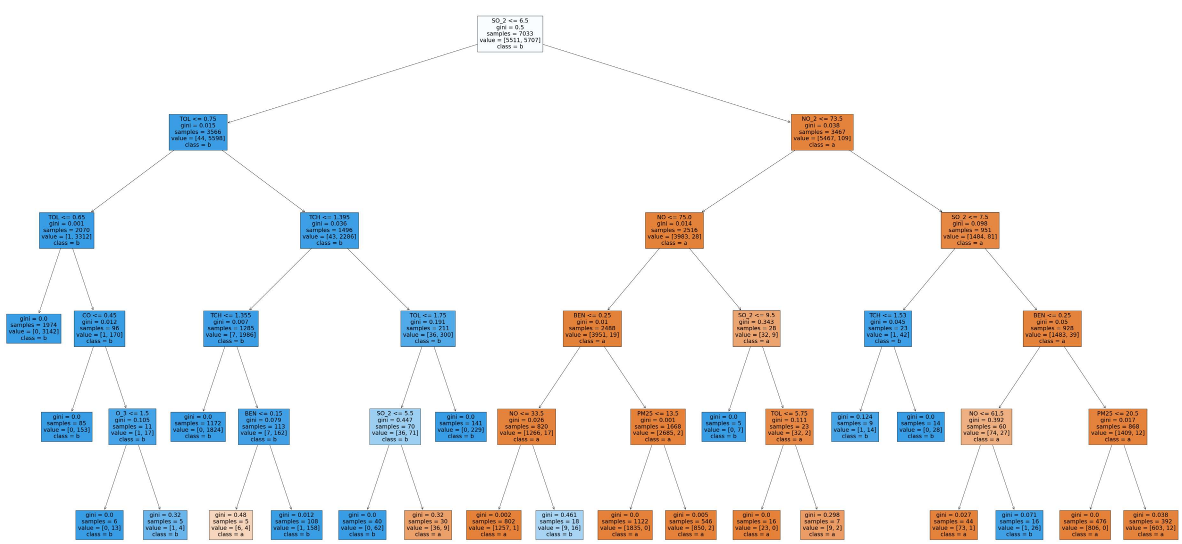
In [32]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

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(0.4305555555555556, 0.9166666666666666, 'SO_2 <= 6.5\ngini = 0.5\nsampl
es = 7033\nvalue = [5511, 5707]\nclass = b'),
 Text(0.16666666666666666, 0.75, 'TOL <= 0.75\ngini = 0.015\nsamples = 3566\n
value = [44, 5598]\nclass = b'),
 Text(0.05555555555555555, 0.5833333333333334, 'TOL <= 0.65\ngini = 0.001\nsa
mples = 2070\nvalue = [1, 3312]\nclass = b'),
 Text(0.027777777777777776, 0.4166666666666667, 'gini = 0.0\nsamples = 1974\n
value = [0, 3142]\nclass = b'),
 Text(0.08333333333333333, 0.4166666666666667, 'CO <= 0.45\ngini = 0.012\nsam
ples = 96\nvalue = [1, 170]\nclass = b'),
 Text(0.05555555555555555, 0.25, 'gini = 0.0\nsamples = 85\nvalue = [0, 153]
\nclass = b'),
 Text(0.1111111111111111, 0.25, 'O_3 <= 1.5\ngini = 0.105\nsamples = 11\nvalu
e = [1, 17]\nclass = b'),
 Text(0.08333333333333333, 0.08333333333333333, 'gini = 0.0\nsamples = 6\nval
ue = [0, 13]\nclass = b'),
 Text(0.1388888888888889, 0.08333333333333333, 'gini = 0.32\nsamples = 5\nval
ue = [1, 4]\nclass = b'),
 Text(0.2777777777777778, 0.5833333333333334, 'TCH <= 1.395\ngini = 0.036\nsa
mples = 1496\nvalue = [43, 2286]\nclass = b'),
 Text(0.19444444444444445, 0.4166666666666667, 'TCH <= 1.355\ngini = 0.007\ns
amples = 1285\nvalue = [7, 1986]\nclass = b'),
 Text(0.16666666666666666, 0.25, 'gini = 0.0\nsamples = 1172\nvalue = [0, 182
4]\nclass = b'),
 Text(0.2222222222222222, 0.25, 'BEN <= 0.15\ngini = 0.079\nsamples = 113\nva
lue = [7, 162]\nclass = b'),
 Text(0.19444444444444445, 0.08333333333333333, 'gini = 0.48\nsamples = 5\nva
lue = [6, 4]\nclass = a'),
 Text(0.25, 0.08333333333333333, 'gini = 0.012\nsamples = 108\nvalue = [1, 15
8]\nclass = b'),
 Text(0.3611111111111111, 0.4166666666666667, 'TOL <= 1.75\ngini = 0.191\nsam
ples = 211\nvalue = [36, 300]\nclass = b'),
 Text(0.3333333333333333, 0.25, 'SO_2 <= 5.5\ngini = 0.447\nsamples = 70\nval
ue = [36, 71]\nclass = b'),
 Text(0.3055555555555556, 0.08333333333333333, 'gini = 0.0\nsamples = 40\nval
ue = [0, 62]\nclass = b'),
 Text(0.3611111111111111, 0.08333333333333333, 'gini = 0.32\nsamples = 30\nva
lue = [36, 9]\nclass = a'),
 Text(0.3888888888888889, 0.25, 'gini = 0.0\nsamples = 141\nvalue = [0, 229]
\nclass = b'),
 Text(0.6944444444444444, 0.75, 'NO_2 <= 73.5\ngini = 0.038\nsamples = 3467\n
value = [5467, 109]\nclass = a'),
 Text(0.5694444444444444, 0.5833333333333334, 'NO <= 75.0\ngini = 0.014\nsamp
les = 2516\nvalue = [3983, 28]\nclass = a'),
 Text(0.5, 0.4166666666666667, 'BEN <= 0.25\ngini = 0.01\nsamples = 2488\nval
ue = [3951, 19]\nclass = a'),
 Text(0.4444444444444444, 0.25, 'NO <= 33.5\ngini = 0.026\nsamples = 820\nval
ue = [1266, 17]\nclass = a'),
 Text(0.4166666666666667, 0.08333333333333333, 'gini = 0.002\nsamples = 802\n
value = [1257, 1]\nclass = a'),
 Text(0.4722222222222222, 0.08333333333333333, 'gini = 0.461\nsamples = 18\nv
alue = [9, 16]\nclass = b'),
 Text(0.5555555555555556, 0.25, 'PM25 <= 13.5\ngini = 0.001\nsamples = 1668\n
value = [2685, 2]\nclass = a'),
 Text(0.5277777777777778, 0.08333333333333333, 'gini = 0.0\nsamples = 1122\nv
alue = [1835, 0]\nclass = a'),
 Text(0.5833333333333334, 0.08333333333333333, 'gini = 0.005\nsamples = 546\n
```

```
value = [850, 2]\nclass = a'),
 Text(0.6388888888888888, 0.4166666666666667, 'SO_2 <= 9.5\ngini = 0.343\nsam
ples = 28\nvalue = [32, 9]\nclass = a'),
 Text(0.6111111111111112, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [0, 7]\ncla
ss = b'),
 Text(0.6666666666666666, 0.25, 'TOL <= 5.75\ngini = 0.111\nsamples = 23\nval
ue = [32, 2]\nclass = a'),
 Text(0.6388888888888888, 0.08333333333333333, 'gini = 0.0\nsamples = 16\nval
ue = [23, 0]\nclass = a'),
 Text(0.6944444444444444, 0.08333333333333333, 'gini = 0.298\nsamples = 7\nva
lue = [9, 2]\nclass = a'),
 Text(0.8194444444444444, 0.5833333333333334, 'SO_2 <= 7.5\ngini = 0.098\nsam
ples = 951\nvalue = [1484, 81]\nclass = a'),
 Text(0.75, 0.4166666666666667, 'TCH <= 1.53\ngini = 0.045\nsamples = 23\nval
ue = [1, 42]\nclass = b'),
 Text(0.7222222222222222, 0.25, 'gini = 0.124\nsamples = 9\nvalue = [1, 14]\n
class = b'),
 Text(0.7777777777777778, 0.25, 'gini = 0.0\nsamples = 14\nvalue = [0, 28]\nc
lass = b'),
 Text(0.8888888888888888, 0.4166666666666667, 'BEN <= 0.25\ngini = 0.05\nsamp
les = 928\nvalue = [1483, 39]\nclass = a'),
 Text(0.8333333333333334, 0.25, 'NO <= 61.5\ngini = 0.392\nsamples = 60\nvalu
e = [74, 27]\nclass = a'),
 Text(0.8055555555555556, 0.08333333333333333, 'gini = 0.027\nsamples = 44\nv
alue = [73, 1]\nclass = a'),
 Text(0.8611111111111112, 0.08333333333333333, 'gini = 0.071\nsamples = 16\nv
alue = [1, 26]\nclass = b'),
 Text(0.9444444444444444, 0.25, 'PM25 <= 20.5\ngini = 0.017\nsamples = 868\nv
alue = [1409, 12]\nclass = a'),
 Text(0.9166666666666666, 0.08333333333333333, 'gini = 0.0\nsamples = 476\nva
lue = [806, 0]\nclass = a'),
 Text(0.9722222222222222, 0.08333333333333333, 'gini = 0.038\nsamples = 392\n
value = [603, 12]\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: 0.8628532861459661
Ridge Regression: 0.86360150367386
Lasso Regression 0.7249293275039095
ElasticNet Regression: 0.818659081862931
Logistic Regression: 0.9971296642955197
Random Forest: 0.9950080228204672
```

Random Forest Is Better!!!