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_2018.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	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN	N
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.41	(
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN	
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN	N
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	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: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

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

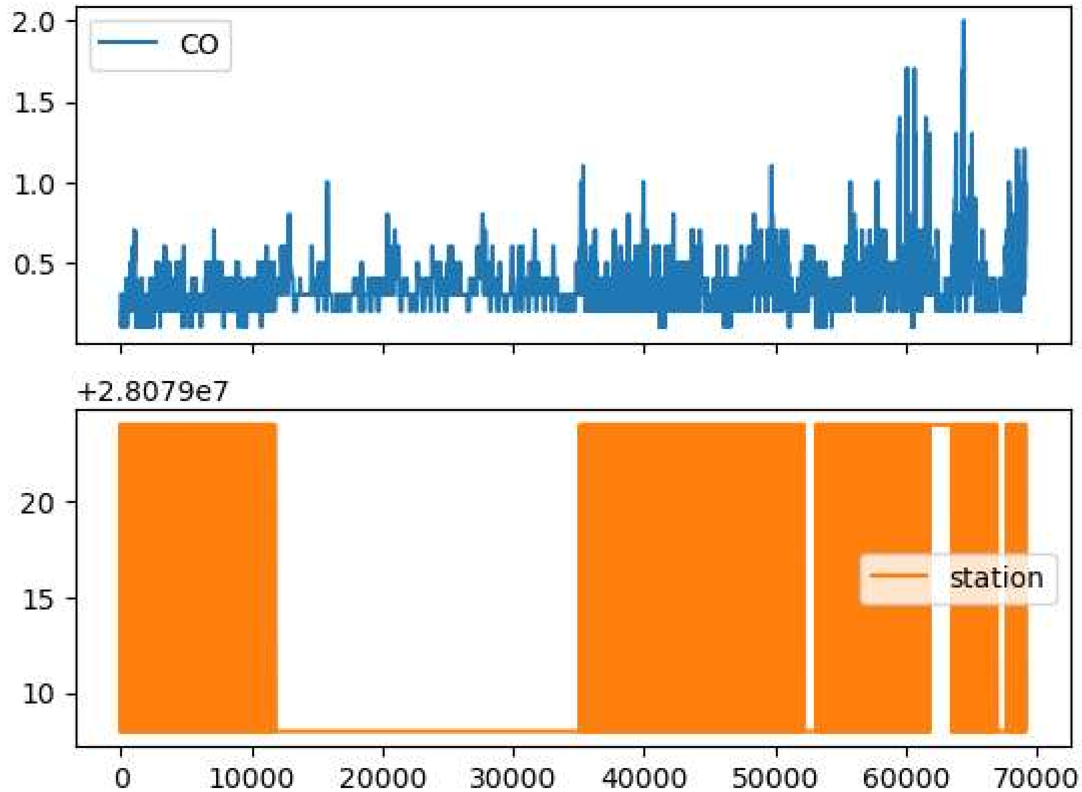
Out[7]:

	CO	station
1	0.3	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
69030	0.7	28079024
69049	1.2	28079008
69054	0.6	28079024
69073	1.0	28079008
69078	0.4	28079024

4562 rows × 2 columns

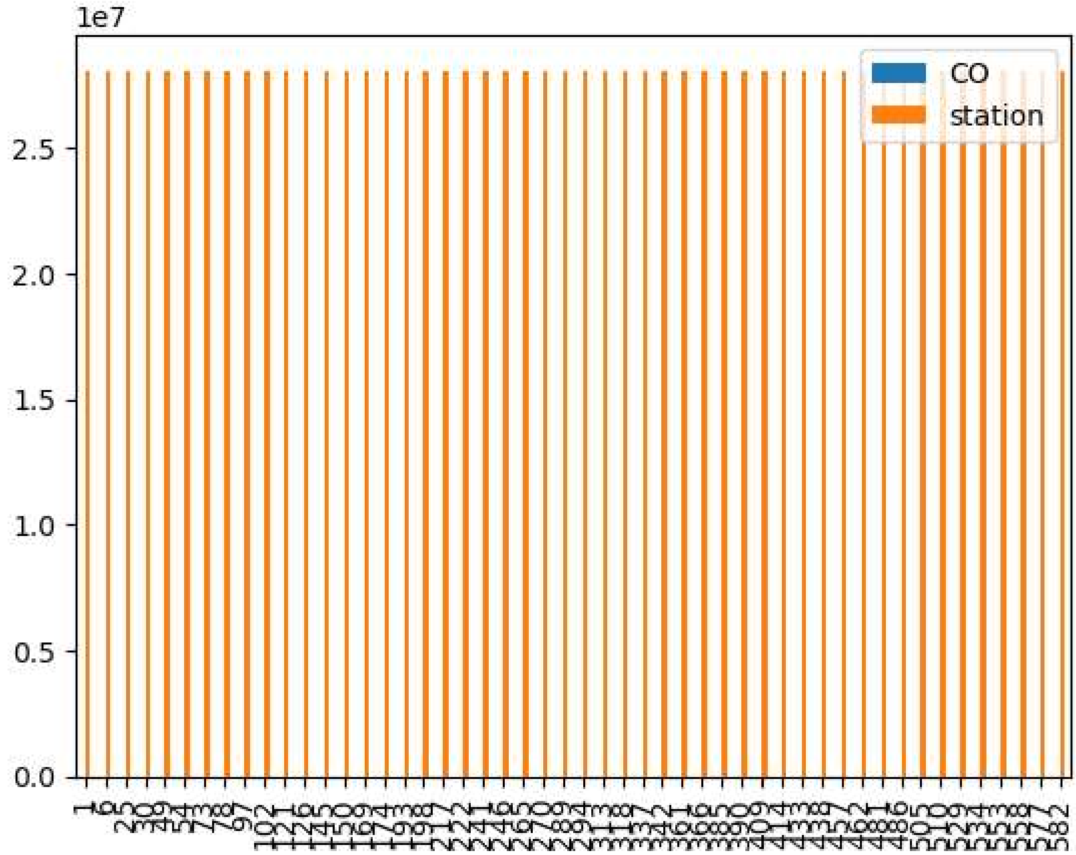
In [8]:
```
data.plot.line(subplots=True)
```

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



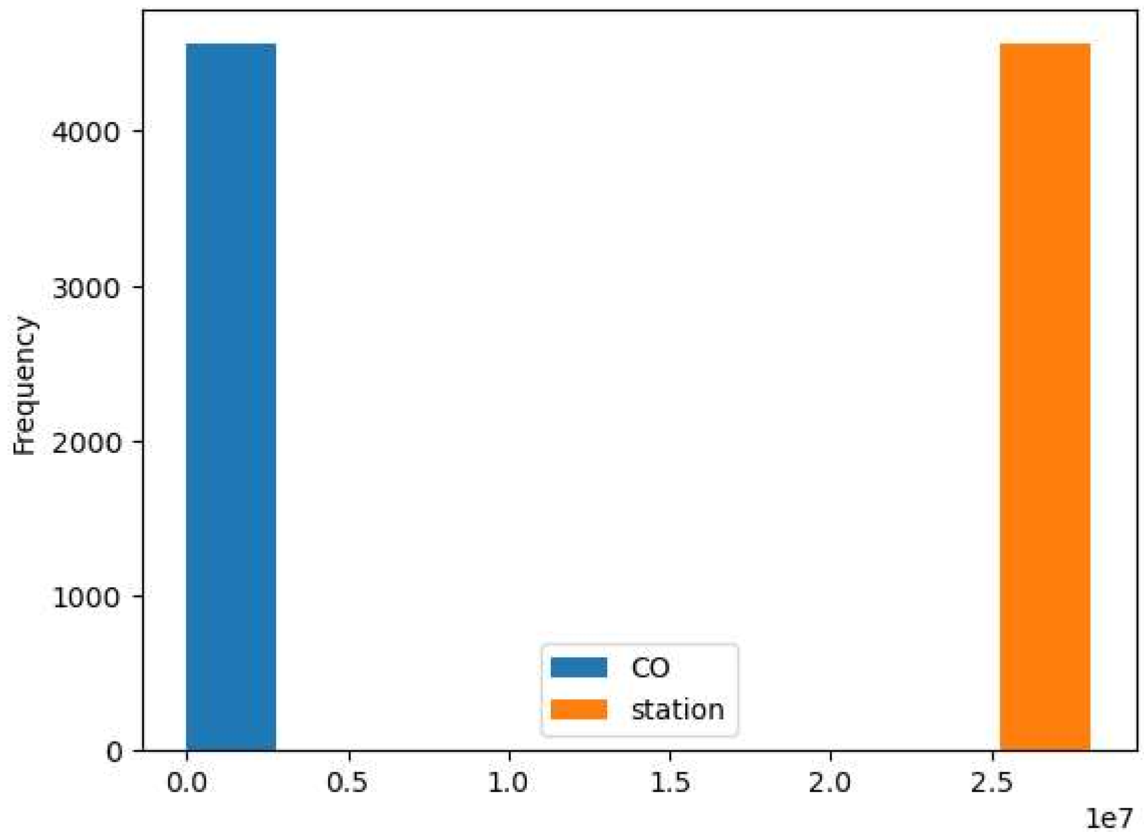
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



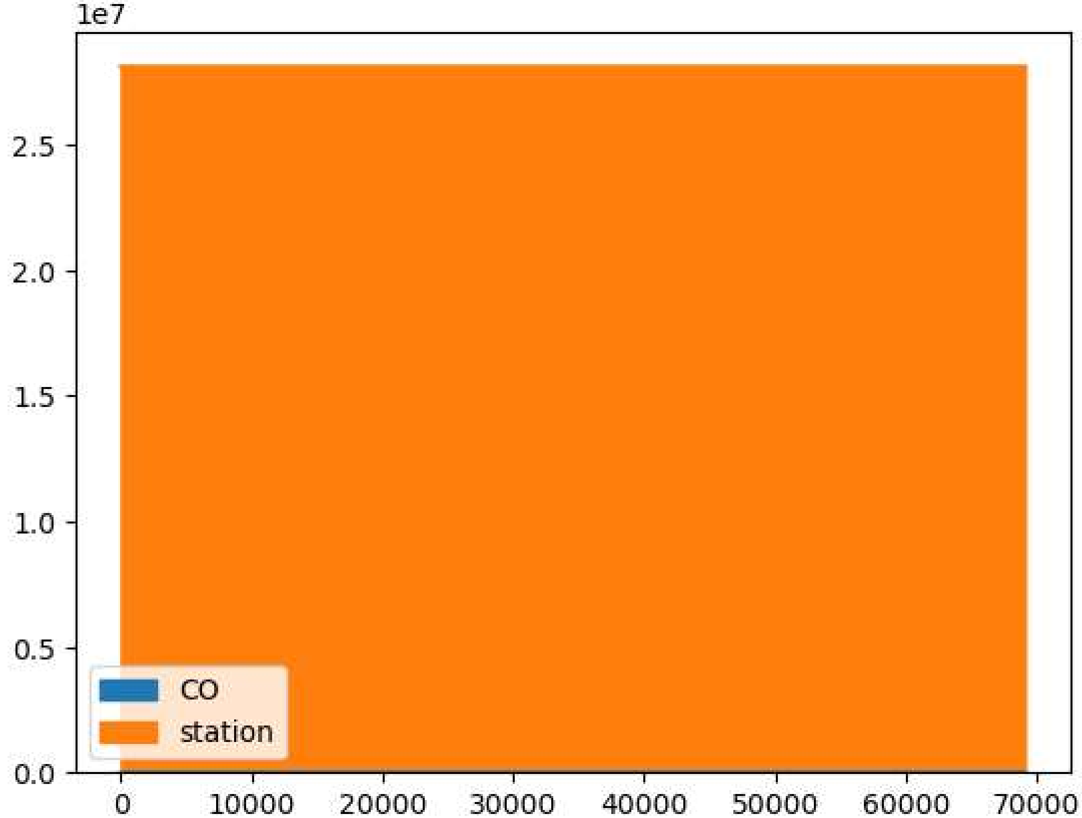
In [10]:
```
data.plot.hist()
```

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



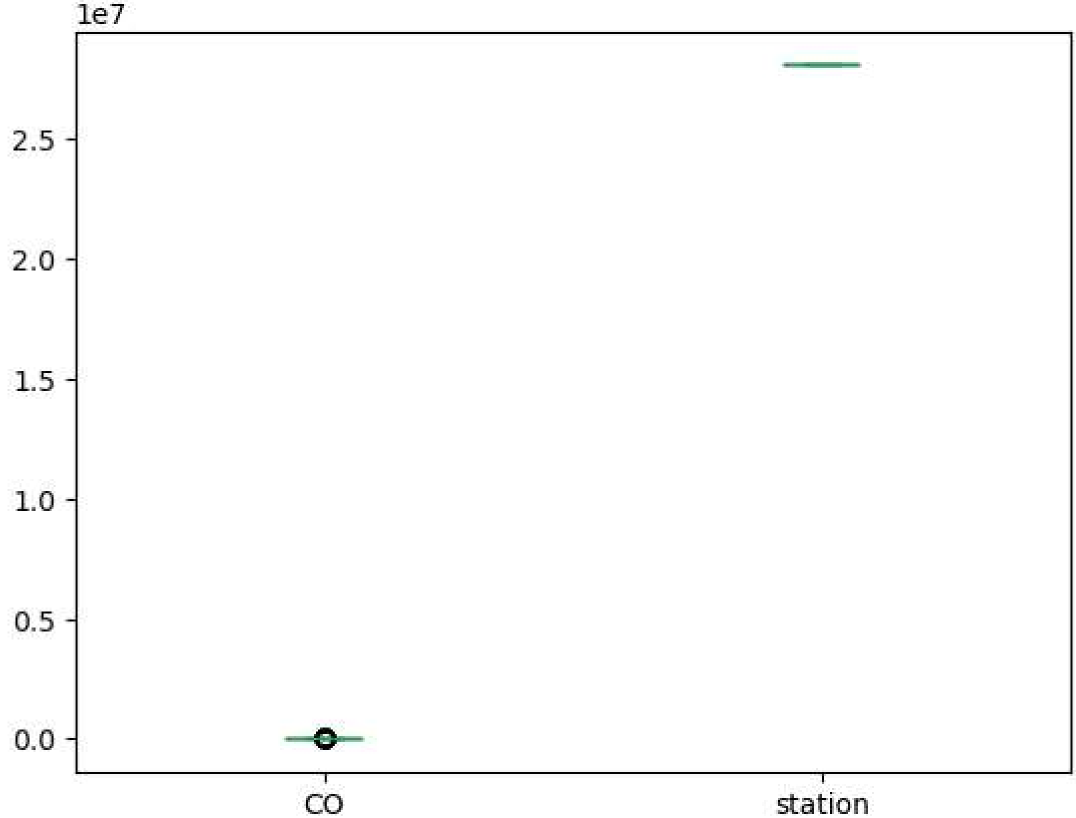
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



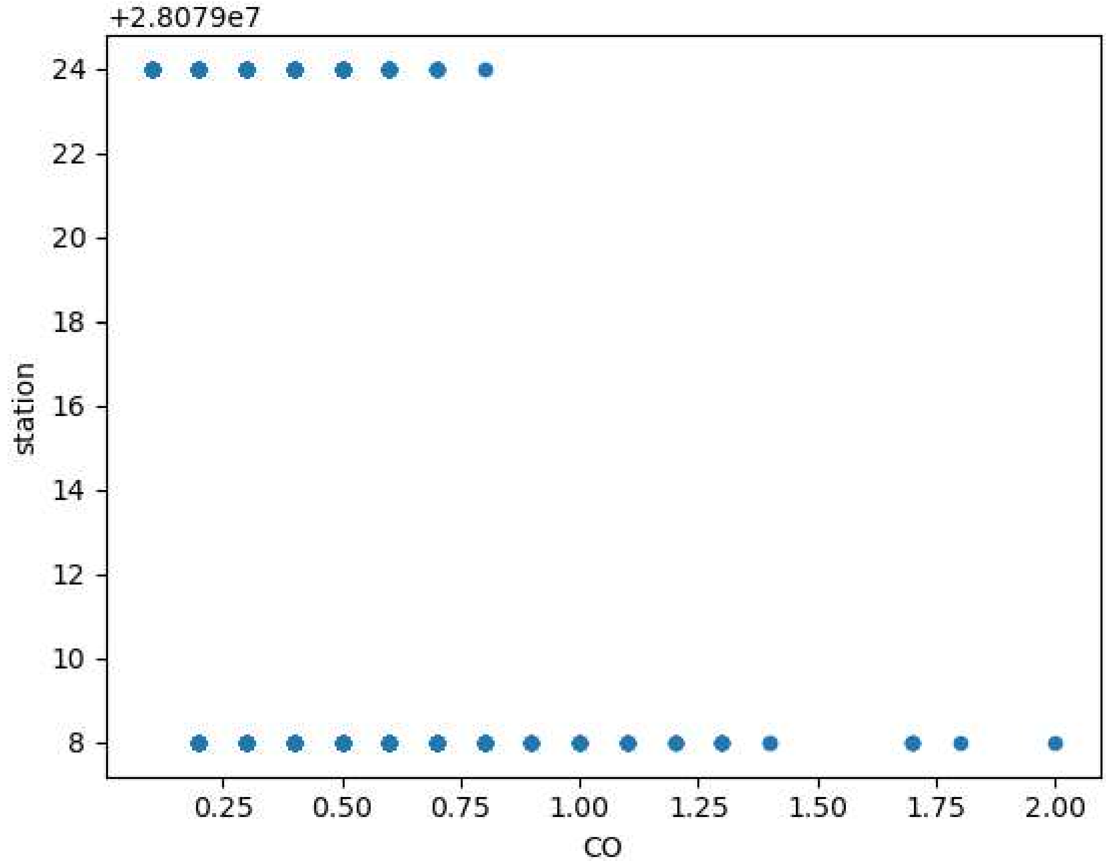
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



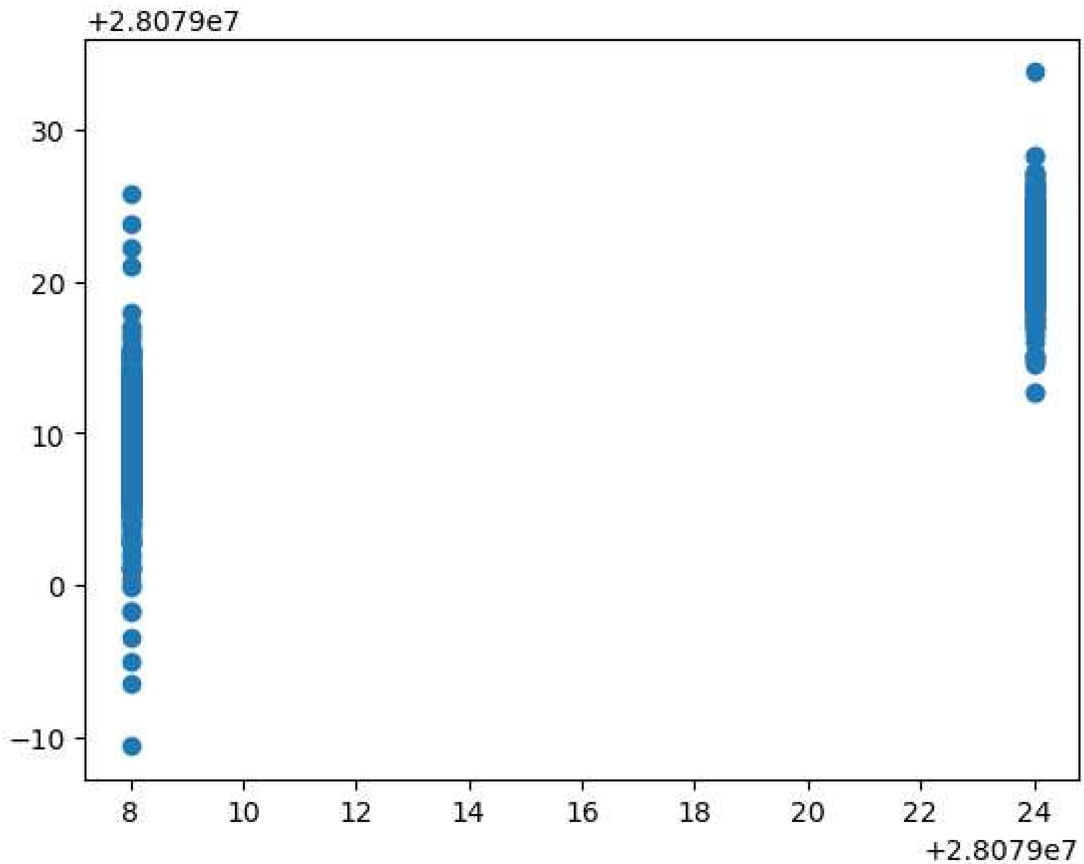
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1898cbfab90>



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

```
0.8244767636678925
0.8027984588266978
```

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)
```

```
0.7150863943178736
0.6992780061799038
```

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

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

Out[19]: 0.41439299623014336

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.        , -0.28845039,
        0.03420383, -0.14450119,  0.27151285, -0.06886965,  0.06094714,
       -0.10779873,  0.        ])
```

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

Out[22]: 28079029.799183168

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

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

Out[24]: 0.4597247930319087

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)))
```

```
4.918455148925304
33.18235851505395
5.760413050732903
```

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, 1.57204801e-20]])
```

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]
}
```

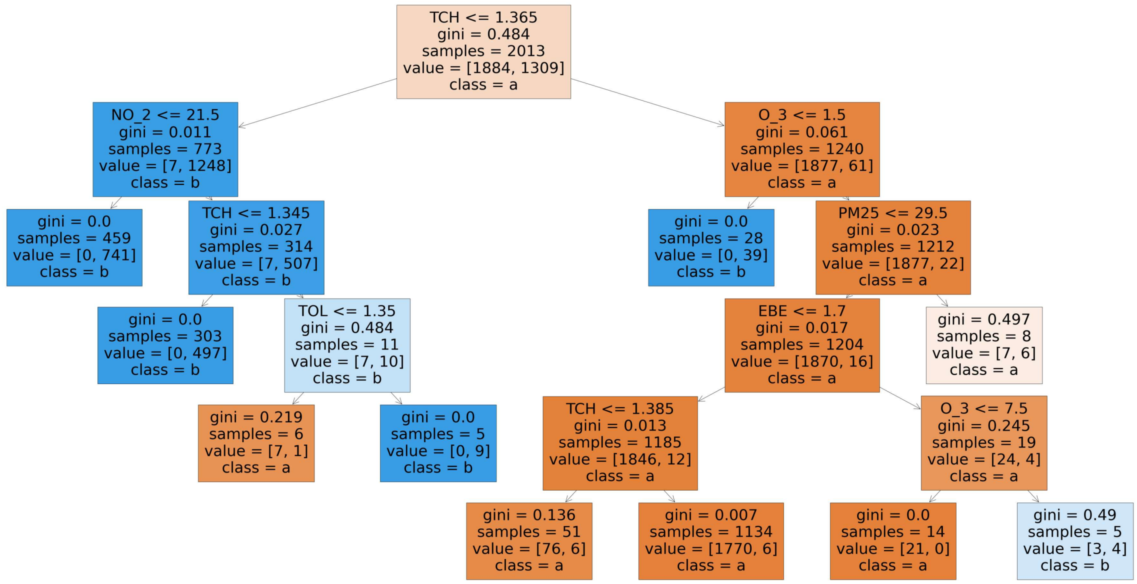
In [32]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

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(0.4230769230769231, 0.9166666666666666, 'TCH <= 1.365\ngini = 0.484\nsa
mples = 2013\nvalue = [1884, 1309]\nclass = a'),
 Text(0.15384615384615385, 0.75, 'NO_2 <= 21.5\ngini = 0.011\nsamples = 773\n
value = [7, 1248]\nclass = b'),
 Text(0.07692307692307693, 0.5833333333333334, 'gini = 0.0\nsamples = 459\nva
lue = [0, 741]\nclass = b'),
 Text(0.23076923076923078, 0.5833333333333334, 'TCH <= 1.345\ngini = 0.027\ns
amples = 314\nvalue = [7, 507]\nclass = b'),
 Text(0.15384615384615385, 0.4166666666666667, 'gini = 0.0\nsamples = 303\nva
lue = [0, 497]\nclass = b'),
 Text(0.3076923076923077, 0.4166666666666667, 'TOL <= 1.35\ngini = 0.484\nsam
ples = 11\nvalue = [7, 10]\nclass = b'),
 Text(0.23076923076923078, 0.25, 'gini = 0.219\nsamples = 6\nvalue = [7, 1]\n
class = a'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [0, 9]\ncl
ass = b'),
 Text(0.6923076923076923, 0.75, 'O_3 <= 1.5\ngini = 0.061\nsamples = 1240\nva
lue = [1877, 61]\nclass = a'),
 Text(0.6153846153846154, 0.5833333333333334, 'gini = 0.0\nsamples = 28\nvalu
e = [0, 39]\nclass = b'),
 Text(0.7692307692307693, 0.5833333333333334, 'PM25 <= 29.5\ngini = 0.023\nsa
mples = 1212\nvalue = [1877, 22]\nclass = a'),
 Text(0.6923076923076923, 0.4166666666666667, 'EBE <= 1.7\ngini = 0.017\nsamp
les = 1204\nvalue = [1870, 16]\nclass = a'),
 Text(0.5384615384615384, 0.25, 'TCH <= 1.385\ngini = 0.013\nsamples = 1185\n
value = [1846, 12]\nclass = a'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.136\nsamples = 51\n
value = [76, 6]\nclass = a'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.007\nsamples = 1134
\nvalue = [1770, 6]\nclass = a'),
 Text(0.8461538461538461, 0.25, 'O_3 <= 7.5\ngini = 0.245\nsamples = 19\nvalu
e = [24, 4]\nclass = a'),
 Text(0.7692307692307693, 0.08333333333333333, 'gini = 0.0\nsamples = 14\nval
ue = [21, 0]\nclass = a'),
 Text(0.9230769230769231, 0.08333333333333333, 'gini = 0.49\nsamples = 5\nval
ue = [3, 4]\nclass = b'),
 Text(0.8461538461538461, 0.4166666666666667, 'gini = 0.497\nsamples = 8\nval
ue = [7, 6]\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: 0.8244767636678925
Ridge Regression: 0.7150863943178736
Lasso Regression 0.41439299623014336
ElasticNet Regression: 0.4597247930319087
Logistic Regression: 0.9890398947829899
Random Forest: 0.9934232104996368
```

Random Forest Is Better!!!