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_2016.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	stat
0	2016-11-01 01:00:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	28079
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	28079
2	2016-11-01 01:00:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	28079
3	2016-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	28079
4	2016-11-01 01:00:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	28079

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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

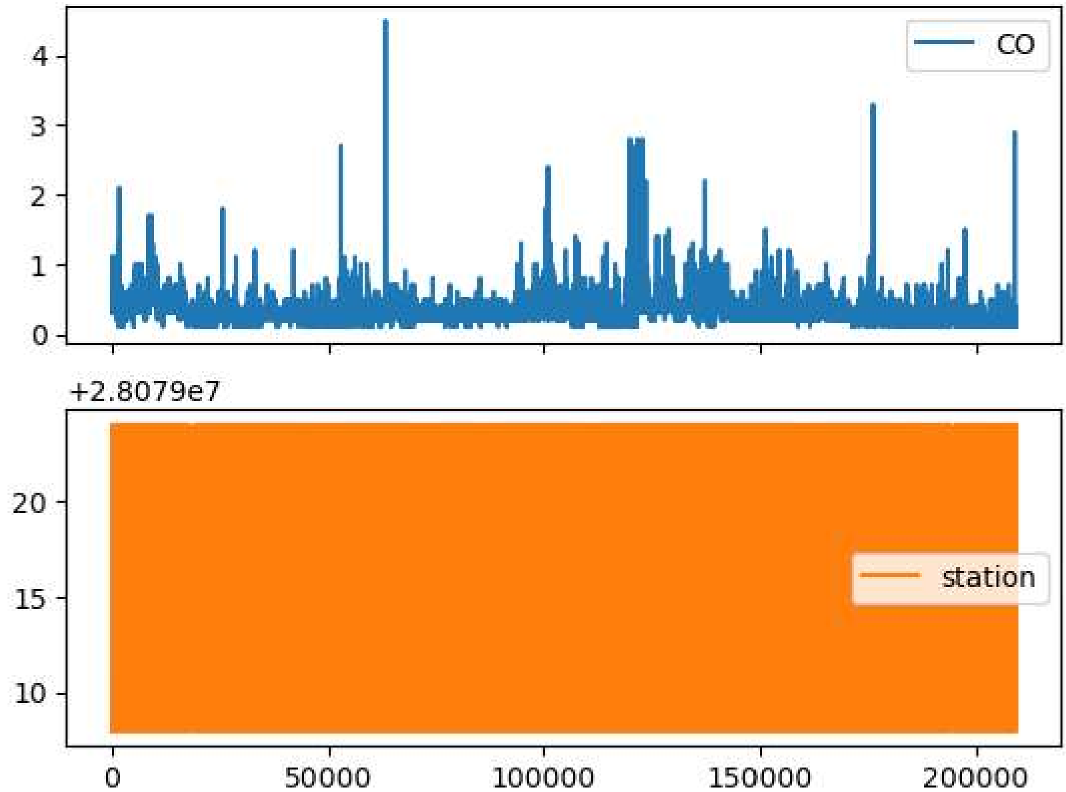
Out[7]:

	CO	station
1	1.1	28079008
6	0.8	28079024
25	1.0	28079008
30	0.7	28079024
49	0.8	28079008
...	...	...
209430	0.2	28079024
209449	0.4	28079008
209454	0.2	28079024
209473	0.4	28079008
209478	0.2	28079024

16932 rows × 2 columns

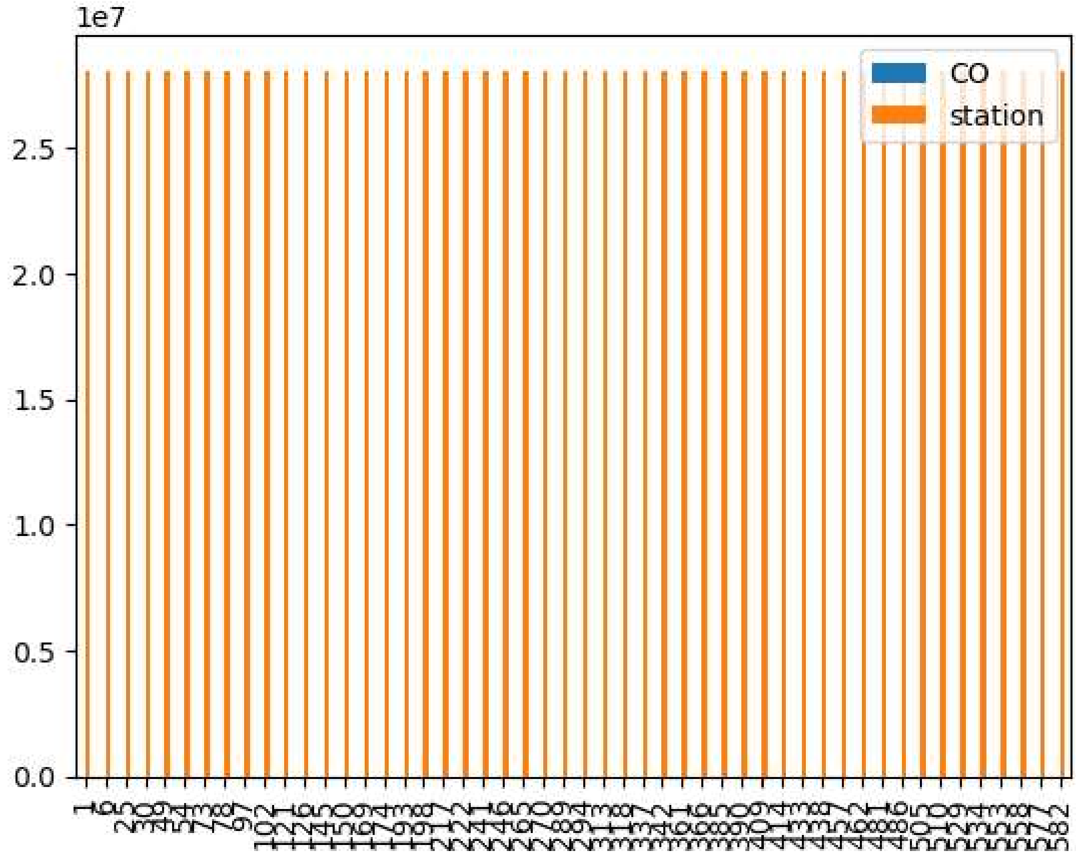
In [8]:
```
data.plot.line(subplots=True)
```

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



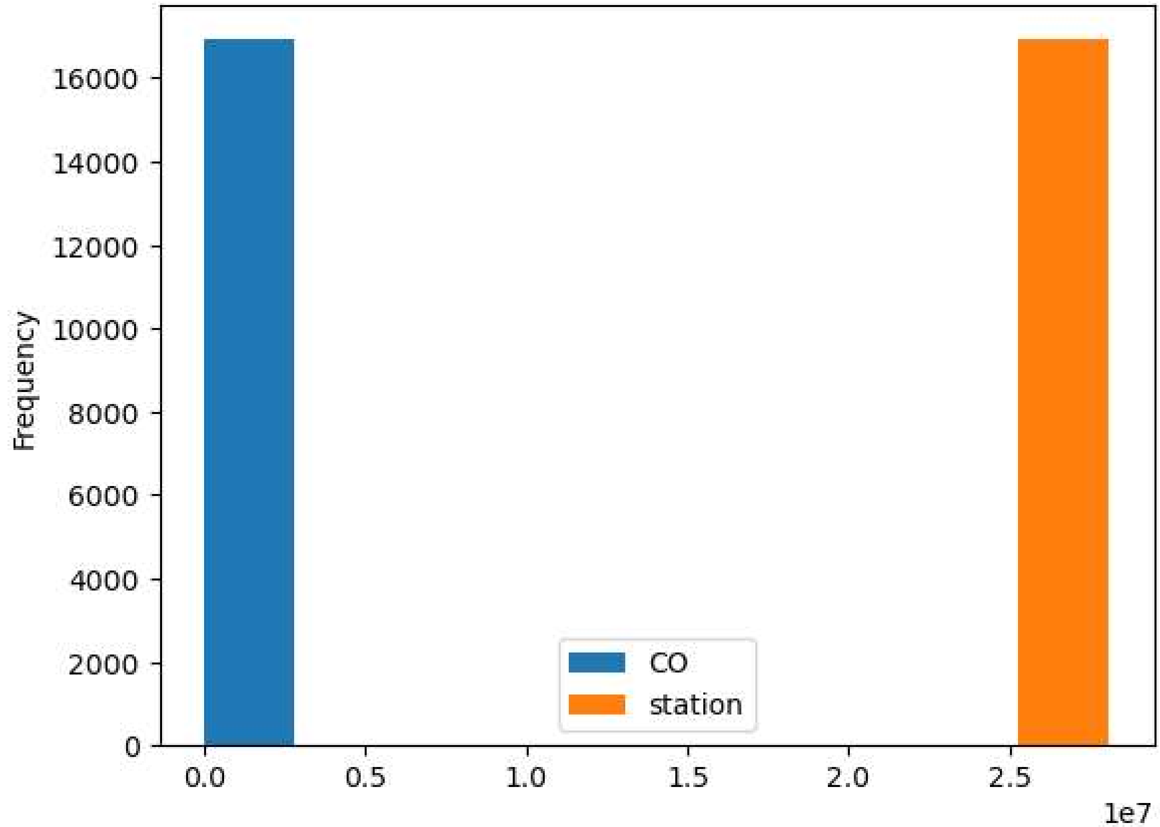
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



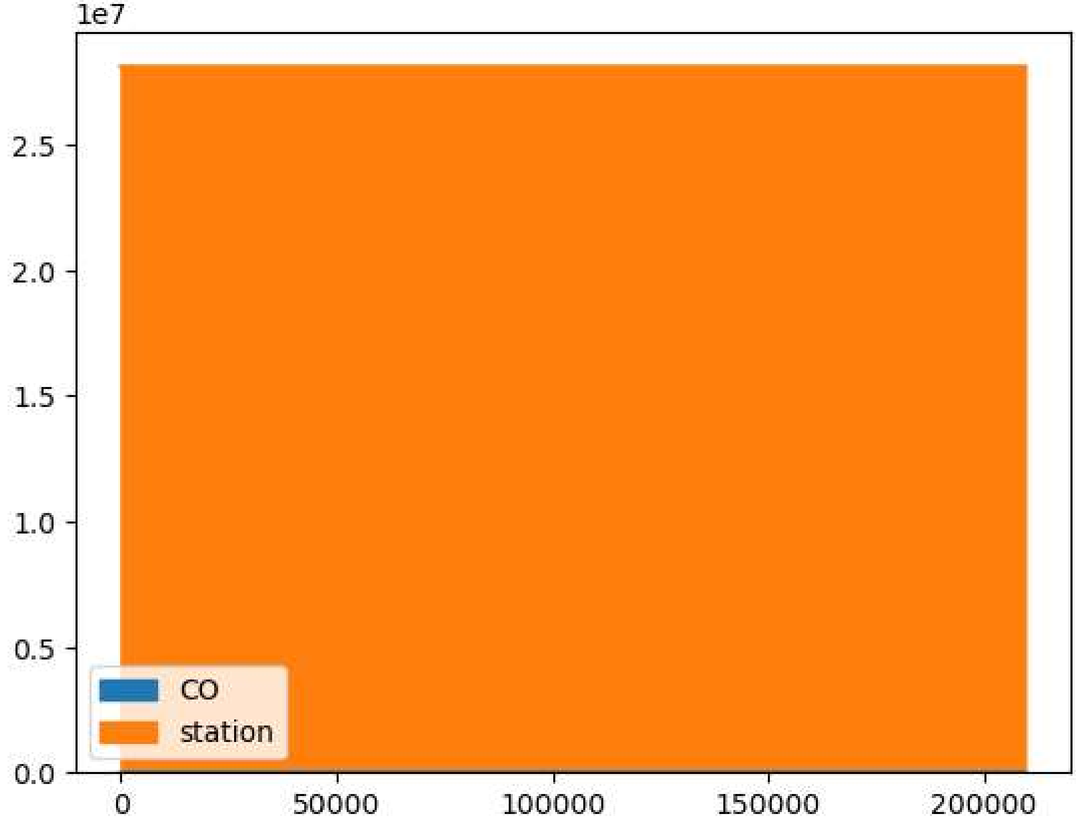
In [10]:
```
data.plot.hist()
```

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



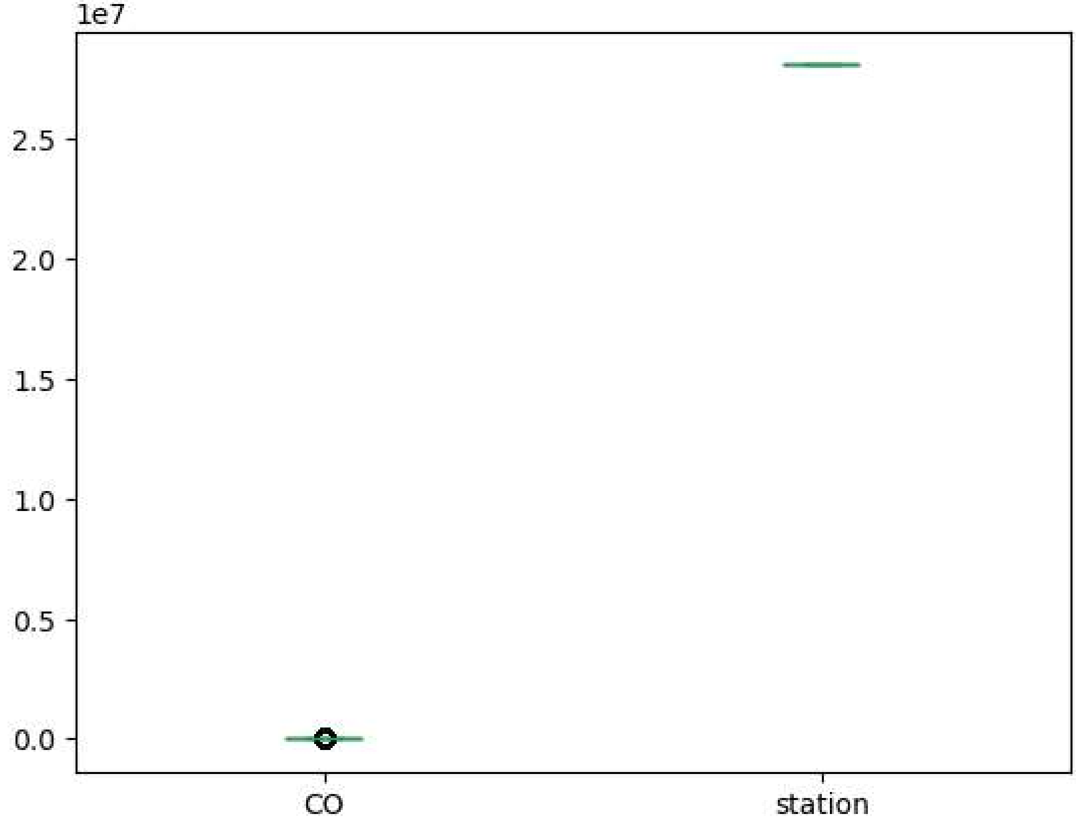
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



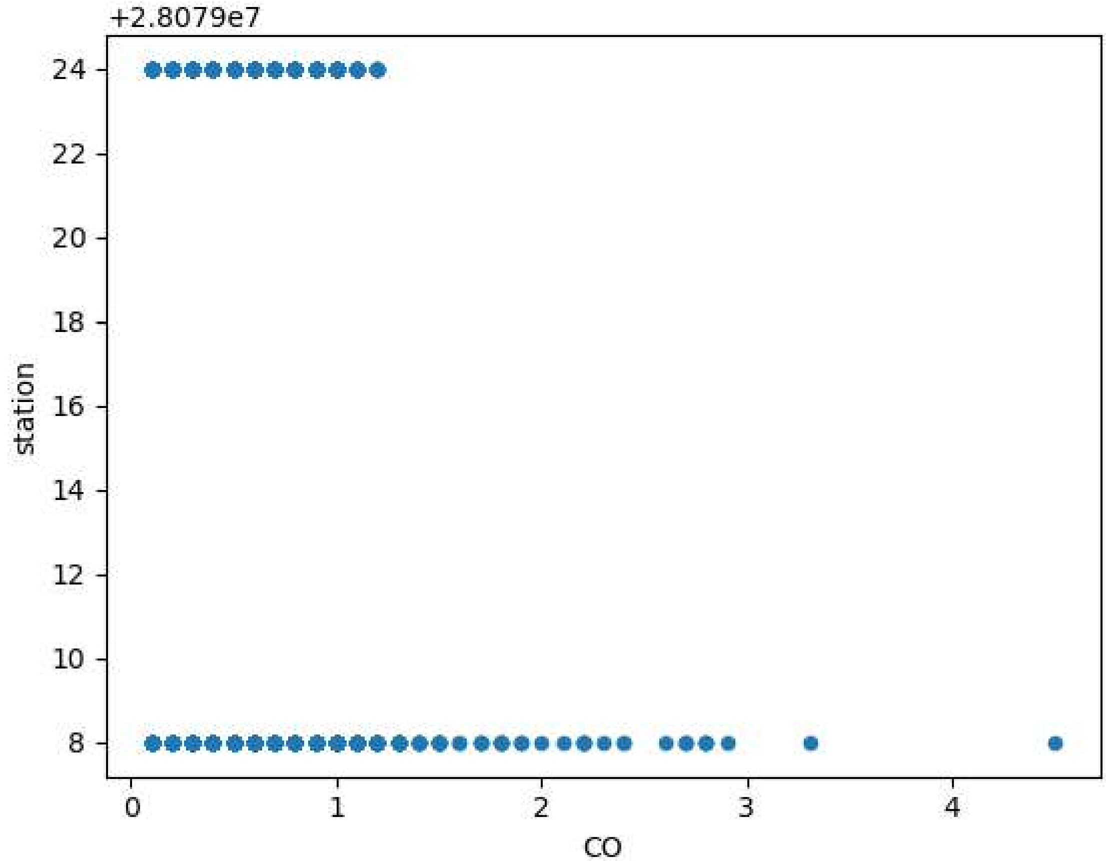
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



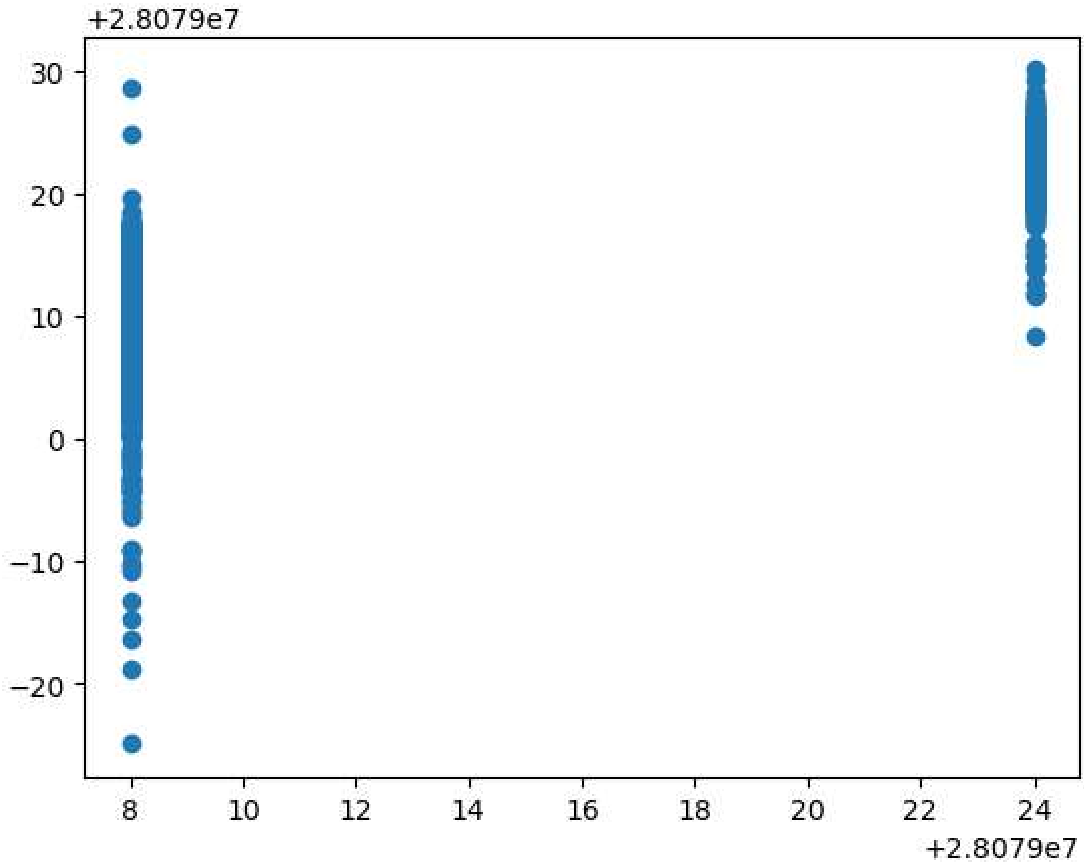
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1f37805afd0>



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

```
0.8307069955786595
0.8268326961423023
```

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)
```

```
0.8307958700138998
0.8267460978017672
```

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

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

Out[19]:
```
0.6552591684765237
```

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.        , -0.10673492,
        0.04673306, -0.02155889,  0.003281  ,  0.05059301, -0.86163134,
       -0.02501396,  0.        ])
```

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

Out[22]:
```
28079026.179314196
```

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

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

Out[24]:
```
0.7176989302087358
```

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)))
```

```
3.295905291957
18.066361126603837
4.250454225915607
```

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, 1.86372661e-53]])
```

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)
```

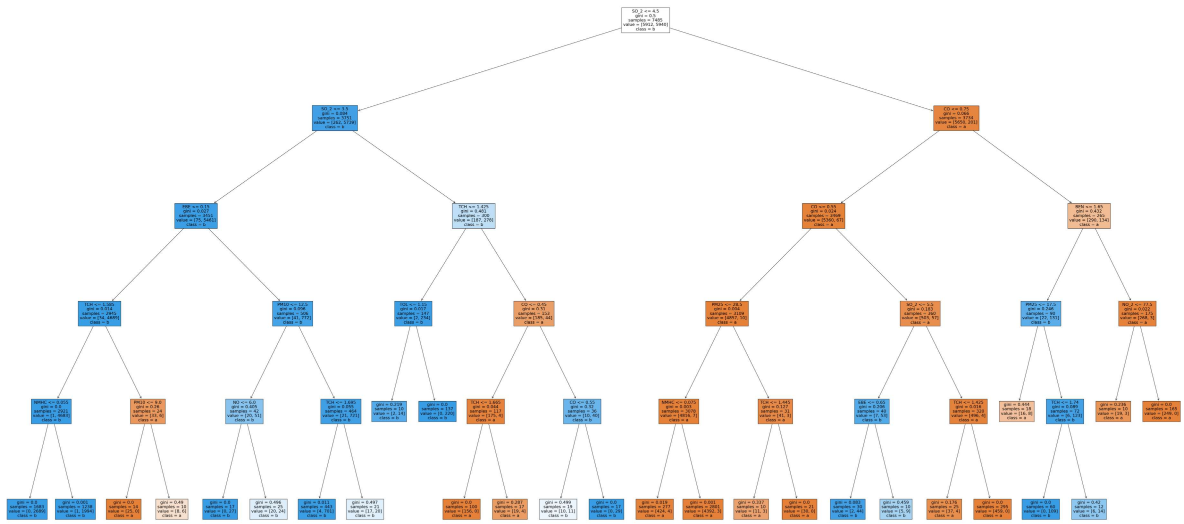
Out[32]:

```
▸ 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(0.5433673469387755, 0.9166666666666666, 'SO_2 <= 4.5\ngini = 0.5\nsampl
es = 7485\nvalue = [5912, 5940]\nclass = b'),
 Text(0.28061224489795916, 0.75, 'SO_2 <= 3.5\ngini = 0.084\nsamples = 3751\n
value = [262, 5739]\nclass = b'),
 Text(0.16326530612244897, 0.5833333333333334, 'EBE <= 0.15\ngini = 0.027\nsa
mples = 3451\nvalue = [75, 5461]\nclass = b'),
 Text(0.08163265306122448, 0.4166666666666667, 'TCH <= 1.585\ngini = 0.014\ns
amples = 2945\nvalue = [34, 4689]\nclass = b'),
 Text(0.04081632653061224, 0.25, 'NMHC <= 0.055\ngini = 0.0\nsamples = 2921\n
value = [1, 4683]\nclass = b'),
 Text(0.02040816326530612, 0.08333333333333333, 'gini = 0.0\nsamples = 1683\n
value = [0, 2689]\nclass = b'),
 Text(0.061224489795918366, 0.08333333333333333, 'gini = 0.001\nsamples = 123
8\nvalue = [1, 1994]\nclass = b'),
 Text(0.12244897959183673, 0.25, 'PM10 <= 9.0\ngini = 0.26\nsamples = 24\nval
ue = [33, 6]\nclass = a'),
 Text(0.10204081632653061, 0.08333333333333333, 'gini = 0.0\nsamples = 14\nva
lue = [25, 0]\nclass = a'),
 Text(0.14285714285714285, 0.08333333333333333, 'gini = 0.49\nsamples = 10\nv
alue = [8, 6]\nclass = a'),
 Text(0.24489795918367346, 0.4166666666666667, 'PM10 <= 12.5\ngini = 0.096\ns
amples = 506\nvalue = [41, 772]\nclass = b'),
 Text(0.20408163265306123, 0.25, 'NO <= 6.0\ngini = 0.405\nsamples = 42\nvalu
e = [20, 51]\nclass = b'),
 Text(0.1836734693877551, 0.08333333333333333, 'gini = 0.0\nsamples = 17\nval
ue = [0, 27]\nclass = b'),
 Text(0.22448979591836735, 0.08333333333333333, 'gini = 0.496\nsamples = 25\n
value = [20, 24]\nclass = b'),
 Text(0.2857142857142857, 0.25, 'TCH <= 1.695\ngini = 0.055\nsamples = 464\nv
alue = [21, 721]\nclass = b'),
 Text(0.2653061224489796, 0.08333333333333333, 'gini = 0.011\nsamples = 443\n
value = [4, 701]\nclass = b'),
 Text(0.30612244897959184, 0.08333333333333333, 'gini = 0.497\nsamples = 21\n
value = [17, 20]\nclass = b'),
 Text(0.3979591836734694, 0.5833333333333334, 'TCH <= 1.425\ngini = 0.481\nsa
mples = 300\nvalue = [187, 278]\nclass = b'),
 Text(0.3469387755102041, 0.4166666666666667, 'TOL <= 1.15\ngini = 0.017\nsam
ples = 147\nvalue = [2, 234]\nclass = b'),
 Text(0.32653061224489793, 0.25, 'gini = 0.219\nsamples = 10\nvalue = [2, 14]
\nclass = b'),
 Text(0.3673469387755102, 0.25, 'gini = 0.0\nsamples = 137\nvalue = [0, 220]
\nclass = b'),
 Text(0.4489795918367347, 0.4166666666666667, 'CO <= 0.45\ngini = 0.31\nsampl
es = 153\nvalue = [185, 44]\nclass = a'),
 Text(0.40816326530612246, 0.25, 'TCH <= 1.665\ngini = 0.044\nsamples = 117\n
value = [175, 4]\nclass = a'),
 Text(0.3877551020408163, 0.08333333333333333, 'gini = 0.0\nsamples = 100\nva
lue = [156, 0]\nclass = a'),
 Text(0.42857142857142855, 0.08333333333333333, 'gini = 0.287\nsamples = 17\n
value = [19, 4]\nclass = a'),
 Text(0.4897959183673469, 0.25, 'CO <= 0.55\ngini = 0.32\nsamples = 36\nvalue
= [10, 40]\nclass = b'),
 Text(0.46938775510204084, 0.08333333333333333, 'gini = 0.499\nsamples = 19\n
value = [10, 11]\nclass = b'),
 Text(0.5102040816326531, 0.08333333333333333, 'gini = 0.0\nsamples = 17\nval
ue = [0, 29]\nclass = b'),
 Text(0.8061224489795918, 0.75, 'CO <= 0.75\ngini = 0.066\nsamples = 3734\nva
```

```
lue = [5650, 201]\nclass = a'),
 Text(0.6938775510204082, 0.5833333333333334, 'CO <= 0.55\ngini = 0.024\nsamp
les = 3469\nvalue = [5360, 67]\nclass = a'),
 Text(0.6122448979591837, 0.4166666666666667, 'PM25 <= 28.5\ngini = 0.004\nsa
mples = 3109\nvalue = [4857, 10]\nclass = a'),
 Text(0.5714285714285714, 0.25, 'NMHC <= 0.075\ngini = 0.003\nsamples = 3078
\nvalue = [4816, 7]\nclass = a'),
 Text(0.5510204081632653, 0.08333333333333333, 'gini = 0.019\nsamples = 277\n
value = [424, 4]\nclass = a'),
 Text(0.5918367346938775, 0.08333333333333333, 'gini = 0.001\nsamples = 2801
\nvalue = [4392, 3]\nclass = a'),
 Text(0.6530612244897959, 0.25, 'TCH <= 1.445\ngini = 0.127\nsamples = 31\nva
lue = [41, 3]\nclass = a'),
 Text(0.6326530612244898, 0.08333333333333333, 'gini = 0.337\nsamples = 10\nv
alue = [11, 3]\nclass = a'),
 Text(0.673469387755102, 0.08333333333333333, 'gini = 0.0\nsamples = 21\nvalu
e = [30, 0]\nclass = a'),
 Text(0.7755102040816326, 0.4166666666666667, 'SO_2 <= 5.5\ngini = 0.183\nsam
ples = 360\nvalue = [503, 57]\nclass = a'),
 Text(0.7346938775510204, 0.25, 'EBE <= 0.65\ngini = 0.206\nsamples = 40\nval
ue = [7, 53]\nclass = b'),
 Text(0.7142857142857143, 0.08333333333333333, 'gini = 0.083\nsamples = 30\nv
alue = [2, 44]\nclass = b'),
 Text(0.7551020408163265, 0.08333333333333333, 'gini = 0.459\nsamples = 10\nv
alue = [5, 9]\nclass = b'),
 Text(0.8163265306122449, 0.25, 'TCH <= 1.425\ngini = 0.016\nsamples = 320\nv
alue = [496, 4]\nclass = a'),
 Text(0.7959183673469388, 0.08333333333333333, 'gini = 0.176\nsamples = 25\nv
alue = [37, 4]\nclass = a'),
 Text(0.8367346938775511, 0.08333333333333333, 'gini = 0.0\nsamples = 295\nva
lue = [459, 0]\nclass = a'),
 Text(0.9183673469387755, 0.5833333333333334, 'BEN <= 1.65\ngini = 0.432\nsam
ples = 265\nvalue = [290, 134]\nclass = a'),
 Text(0.8775510204081632, 0.4166666666666667, 'PM25 <= 17.5\ngini = 0.246\nsa
mples = 90\nvalue = [22, 131]\nclass = b'),
 Text(0.8571428571428571, 0.25, 'gini = 0.444\nsamples = 18\nvalue = [16, 8]
\nclass = a'),
 Text(0.8979591836734694, 0.25, 'TCH <= 1.74\ngini = 0.089\nsamples = 72\nval
ue = [6, 123]\nclass = b'),
 Text(0.8775510204081632, 0.08333333333333333, 'gini = 0.0\nsamples = 60\nval
ue = [0, 109]\nclass = b'),
 Text(0.9183673469387755, 0.08333333333333333, 'gini = 0.42\nsamples = 12\nva
lue = [6, 14]\nclass = b'),
 Text(0.9591836734693877, 0.4166666666666667, 'NO_2 <= 77.5\ngini = 0.022\nsa
mples = 175\nvalue = [268, 3]\nclass = a'),
 Text(0.9387755102040817, 0.25, 'gini = 0.236\nsamples = 10\nvalue = [19, 3]
\nclass = a'),
 Text(0.9795918367346939, 0.25, 'gini = 0.0\nsamples = 165\nvalue = [249, 0]
\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: 0.8307069955786595
Ridge Regression: 0.8307958700138998
Lasso Regression 0.6552591684765237
ElasticNet Regression: 0.7176989302087358
Logistic Regression: 0.996161115048429
Random Forest: 0.9947688153898075
```

Random Forest Is Better!!!