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_2012.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	2807900
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	2807900
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	2807901
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	2807901
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	2807901

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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

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

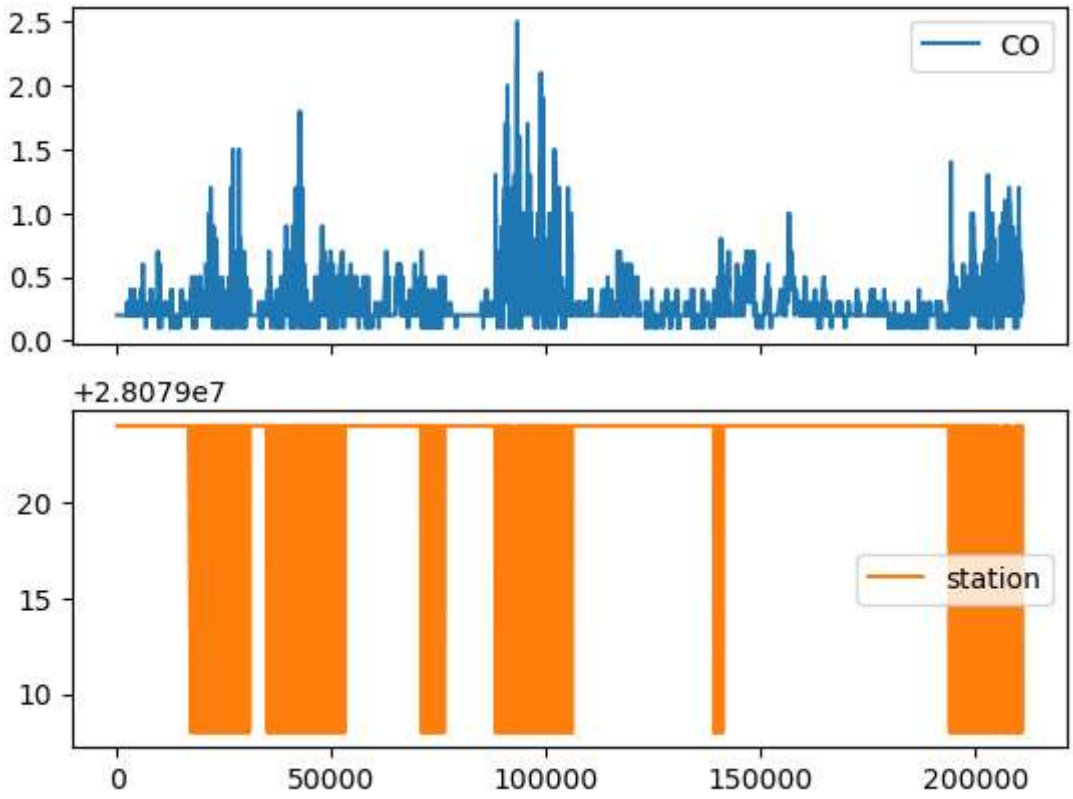
Out[7]:

	CO	station
6	0.2	28079024
30	0.2	28079024
54	0.2	28079024
78	0.2	28079024
102	0.2	28079024
...	...	...
210654	0.3	28079024
210673	0.4	28079008
210678	0.3	28079024
210697	0.4	28079008
210702	0.3	28079024

10916 rows × 2 columns

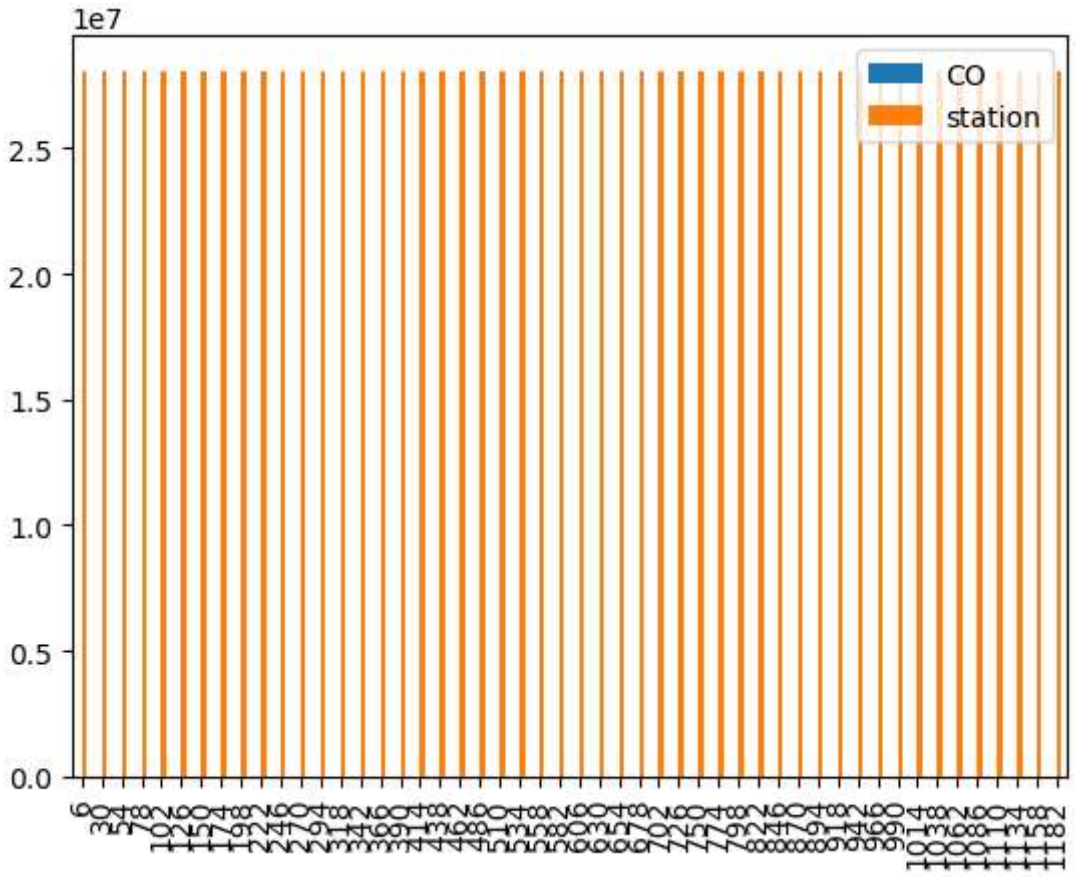
In [8]:
```
data.plot.line(subplots=True)
```

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



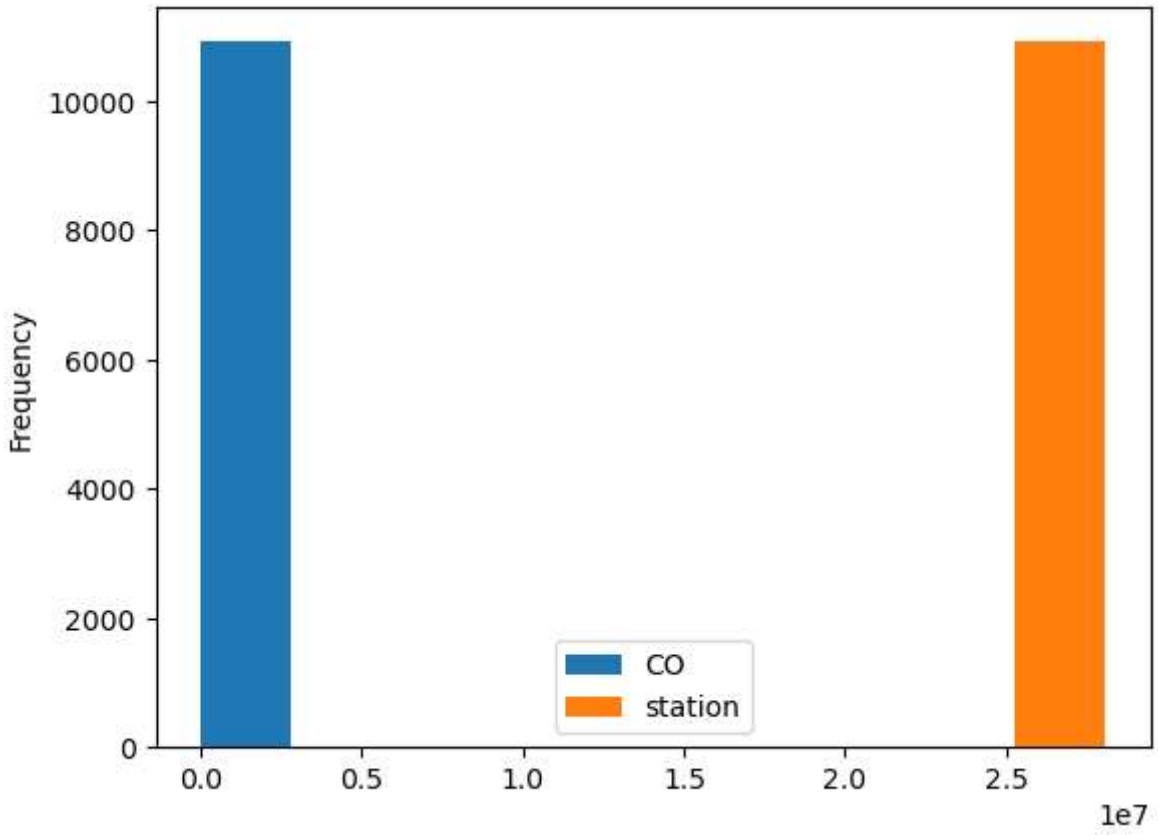
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



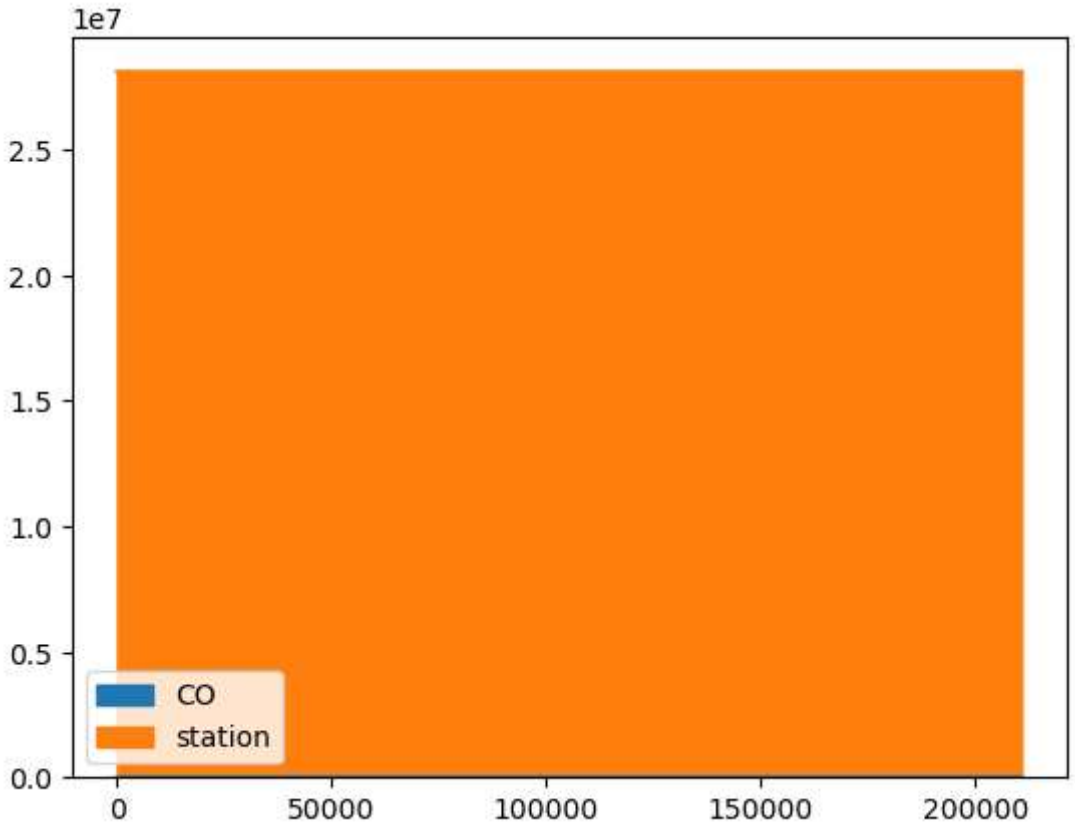
In [10]:
```
data.plot.hist()
```

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



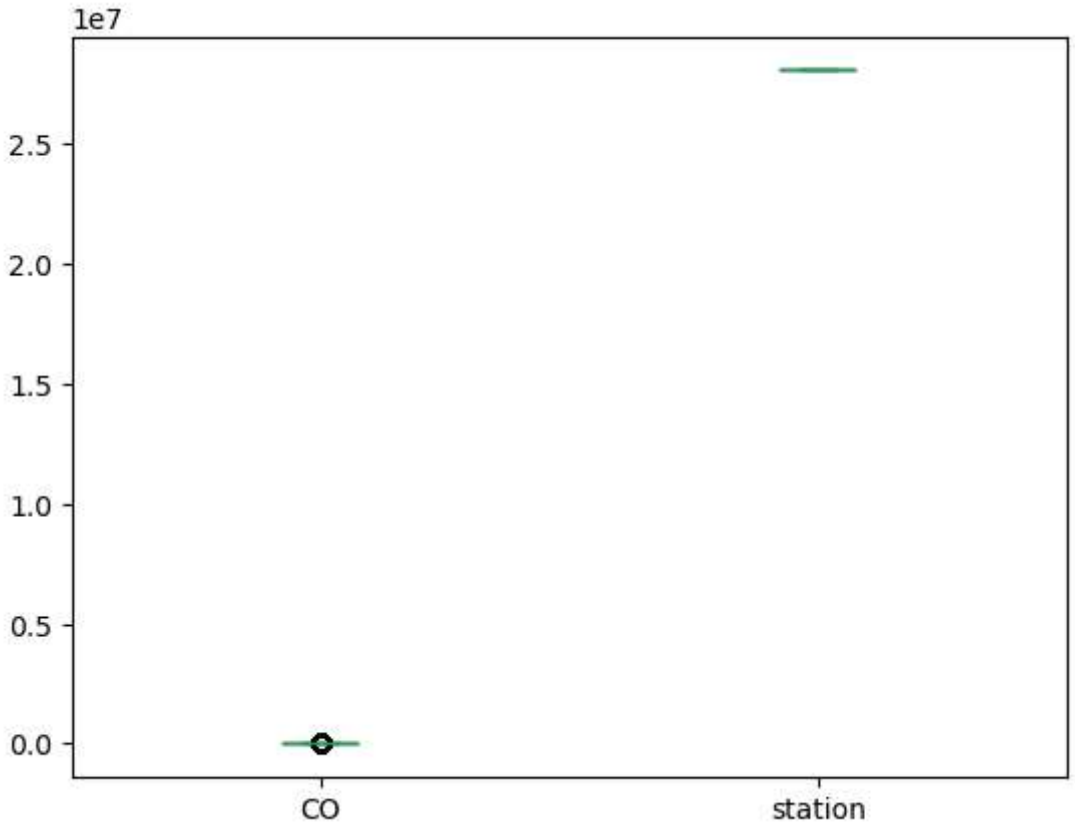
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



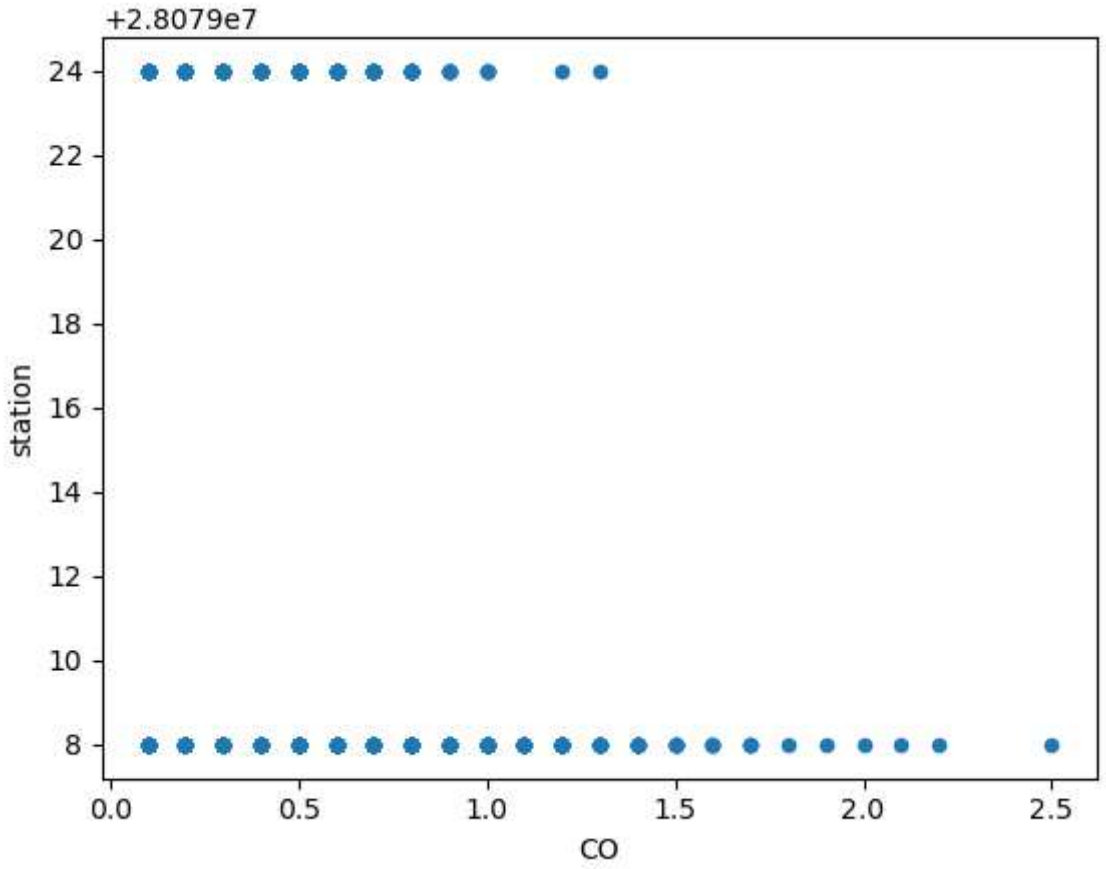
In [12]:
```
data.plot.box()
```

Out[12]:
```
<Axes: >
```



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

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



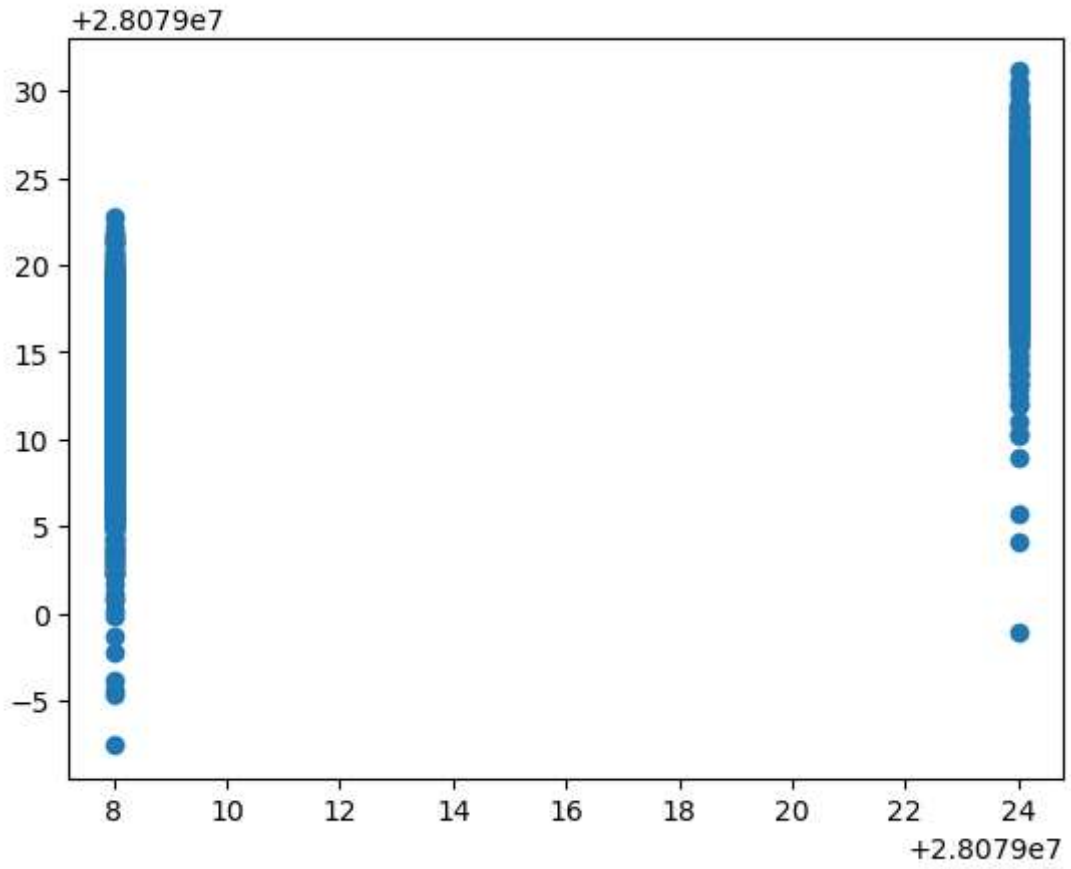
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1c8761fa850>



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

```
0.644033622206244
0.6154224643004862
```

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)
```

```
0.639799790217638
0.6112243584665173
```

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

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

Out[19]:
```
0.3716155940503091
```

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([ 0.00000000e+00,  0.00000000e+00, -0.00000000e+00,  0.00000000e+00,
       -8.18423122e-02,  6.30481885e-02, -3.73703607e-02,  3.29858334e-04,
        0.00000000e+00, -7.07346653e-01,  0.00000000e+00, -6.85485553e-01])
```

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

Out[22]:
```
28079027.605697796
```

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

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

Out[24]:
```
0.5408652684713832
```

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)))
```

```
3.4514185757639297
23.60061674316346
4.858046597467284
```

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([[1.0000000e+00, 2.0517756e-32]])
```

Random Forest

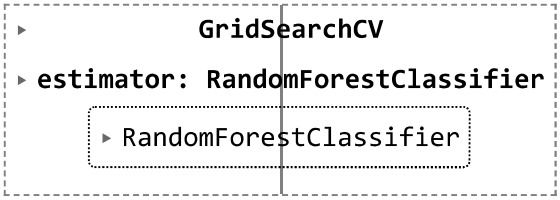
In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

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)
```

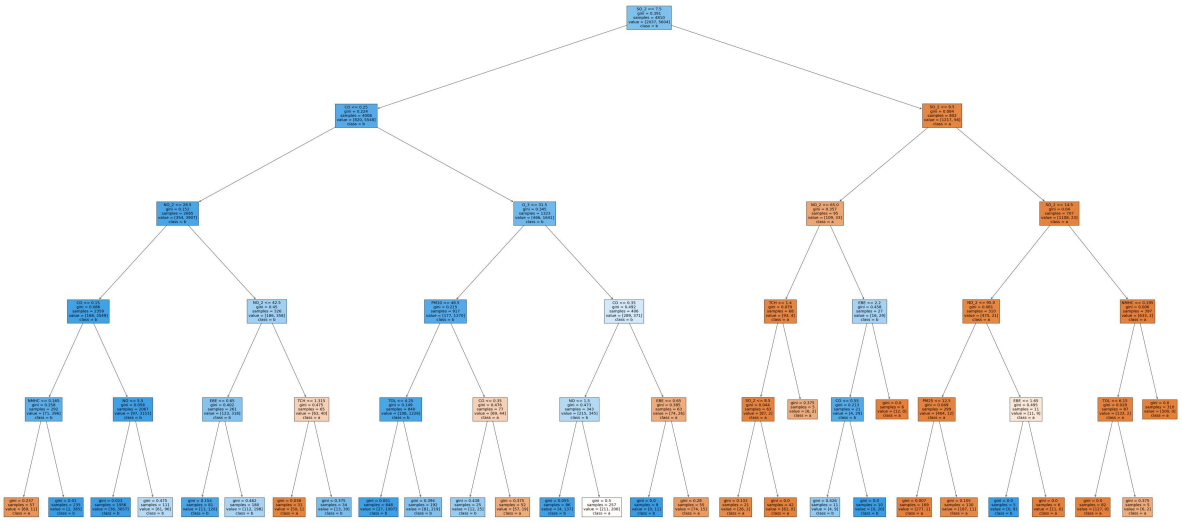
Out[32]:



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(0.5495283018867925, 0.9166666666666666, 'SO_2 <= 7.5\ngini = 0.391\nsam
ples = 4810\nvalue = [2037, 5604]\nclass = b'),
 Text(0.3018867924528302, 0.75, 'CO <= 0.25\ngini = 0.224\nsamples = 4008\nva
lue = [820, 5548]\nclass = b'),
 Text(0.1509433962264151, 0.5833333333333334, 'NO_2 <= 28.5\ngini = 0.152\nsa
mples = 2685\nvalue = [354, 3907]\nclass = b'),
 Text(0.07547169811320754, 0.4166666666666667, 'CO <= 0.15\ngini = 0.086\nsam
ples = 2359\nvalue = [168, 3549]\nclass = b'),
 Text(0.03773584905660377, 0.25, 'NMHC <= 0.165\ngini = 0.258\nsamples = 292
\nvalue = [71, 396]\nclass = b'),
 Text(0.018867924528301886, 0.08333333333333333, 'gini = 0.237\nsamples = 57
\nvalue = [69, 11]\nclass = a'),
 Text(0.05660377358490566, 0.08333333333333333, 'gini = 0.01\nsamples = 235\n
value = [2, 385]\nclass = b'),
 Text(0.11320754716981132, 0.25, 'NO <= 5.5\ngini = 0.058\nsamples = 2067\nva
lue = [97, 3153]\nclass = b'),
 Text(0.09433962264150944, 0.08333333333333333, 'gini = 0.023\nsamples = 1956
\nvalue = [36, 3057]\nclass = b'),
 Text(0.1320754716981132, 0.08333333333333333, 'gini = 0.475\nsamples = 111\n
value = [61, 96]\nclass = b'),
 Text(0.22641509433962265, 0.4166666666666667, 'NO_2 <= 42.5\ngini = 0.45\nsa
mples = 326\nvalue = [186, 358]\nclass = b'),
 Text(0.18867924528301888, 0.25, 'EBE <= 0.65\ngini = 0.402\nsamples = 261\nv
alue = [123, 318]\nclass = b'),
 Text(0.16981132075471697, 0.08333333333333333, 'gini = 0.154\nsamples = 81\n
value = [11, 120]\nclass = b'),
 Text(0.20754716981132076, 0.08333333333333333, 'gini = 0.462\nsamples = 180
\nvalue = [112, 198]\nclass = b'),
 Text(0.2641509433962264, 0.25, 'TCH <= 1.315\ngini = 0.475\nsamples = 65\nva
lue = [63, 40]\nclass = a'),
 Text(0.24528301886792453, 0.08333333333333333, 'gini = 0.038\nsamples = 31\n
value = [50, 1]\nclass = a'),
 Text(0.2830188679245283, 0.08333333333333333, 'gini = 0.375\nsamples = 34\nv
alue = [13, 39]\nclass = b'),
 Text(0.4528301886792453, 0.5833333333333334, 'O_3 <= 31.5\ngini = 0.345\nsam
ples = 1323\nvalue = [466, 1641]\nclass = b'),
 Text(0.37735849056603776, 0.4166666666666667, 'PM10 <= 48.5\ngini = 0.215\ns
amples = 917\nvalue = [177, 1270]\nclass = b'),
 Text(0.33962264150943394, 0.25, 'TOL <= 4.25\ngini = 0.149\nsamples = 840\nv
alue = [108, 1226]\nclass = b'),
 Text(0.32075471698113206, 0.08333333333333333, 'gini = 0.051\nsamples = 648
\nvalue = [27, 1007]\nclass = b'),
 Text(0.3584905660377358, 0.08333333333333333, 'gini = 0.394\nsamples = 192\n
value = [81, 219]\nclass = b'),
 Text(0.41509433962264153, 0.25, 'CO <= 0.35\ngini = 0.476\nsamples = 77\nval
ue = [69, 44]\nclass = a'),
 Text(0.39622641509433965, 0.08333333333333333, 'gini = 0.438\nsamples = 25\n
value = [12, 25]\nclass = b'),
 Text(0.4339622641509434, 0.08333333333333333, 'gini = 0.375\nsamples = 52\nv
alue = [57, 19]\nclass = a'),
 Text(0.5283018867924528, 0.4166666666666667, 'CO <= 0.35\ngini = 0.492\nsamp
les = 406\nvalue = [289, 371]\nclass = b'),
 Text(0.49056603773584906, 0.25, 'NO <= 1.5\ngini = 0.473\nsamples = 343\nval
ue = [215, 345]\nclass = b'),
 Text(0.4716981132075472, 0.08333333333333333, 'gini = 0.055\nsamples = 86\nv
alue = [4, 137]\nclass = b'),
 Text(0.5094339622641509, 0.08333333333333333, 'gini = 0.5\nsamples = 257\nva
```

```
lue = [211, 208]\nclass = a'),
 Text(0.5660377358490566, 0.25, 'EBE <= 0.65\ngini = 0.385\nsamples = 63\nval
ue = [74, 26]\nclass = a'),
 Text(0.5471698113207547, 0.08333333333333333, 'gini = 0.0\nsamples = 8\nvalu
e = [0, 11]\nclass = b'),
 Text(0.5849056603773585, 0.08333333333333333, 'gini = 0.28\nsamples = 55\nva
lue = [74, 15]\nclass = a'),
 Text(0.7971698113207547, 0.75, 'SO_2 <= 9.5\ngini = 0.084\nsamples = 802\nva
lue = [1217, 56]\nclass = a'),
 Text(0.6981132075471698, 0.5833333333333334, 'NO_2 <= 65.0\ngini = 0.357\nsa
mples = 95\nvalue = [109, 33]\nclass = a'),
 Text(0.660377358490566, 0.4166666666666667, 'TCH <= 1.4\ngini = 0.079\nsampl
es = 68\nvalue = [93, 4]\nclass = a'),
 Text(0.6415094339622641, 0.25, 'SO_2 <= 8.5\ngini = 0.044\nsamples = 63\nval
ue = [87, 2]\nclass = a'),
 Text(0.6226415094339622, 0.08333333333333333, 'gini = 0.133\nsamples = 21\nv
alue = [26, 2]\nclass = a'),
 Text(0.660377358490566, 0.08333333333333333, 'gini = 0.0\nsamples = 42\nvalu
e = [61, 0]\nclass = a'),
 Text(0.6792452830188679, 0.25, 'gini = 0.375\nsamples = 5\nvalue = [6, 2]\nc
lass = a'),
 Text(0.7358490566037735, 0.4166666666666667, 'EBE <= 2.2\ngini = 0.458\nsamp
les = 27\nvalue = [16, 29]\nclass = b'),
 Text(0.7169811320754716, 0.25, 'CO <= 0.55\ngini = 0.213\nsamples = 21\nvalu
e = [4, 29]\nclass = b'),
 Text(0.6981132075471698, 0.08333333333333333, 'gini = 0.426\nsamples = 11\nv
alue = [4, 9]\nclass = b'),
 Text(0.7358490566037735, 0.08333333333333333, 'gini = 0.0\nsamples = 10\nval
ue = [0, 20]\nclass = b'),
 Text(0.7547169811320755, 0.25, 'gini = 0.0\nsamples = 6\nvalue = [12, 0]\ncl
ass = a'),
 Text(0.8962264150943396, 0.5833333333333334, 'SO_2 <= 14.5\ngini = 0.04\nsam
ples = 707\nvalue = [1108, 23]\nclass = a'),
 Text(0.8301886792452831, 0.4166666666666667, 'NO_2 <= 95.0\ngini = 0.081\nsa
mples = 310\nvalue = [475, 21]\nclass = a'),
 Text(0.7924528301886793, 0.25, 'PM25 <= 12.5\ngini = 0.049\nsamples = 299\nv
alue = [464, 12]\nclass = a'),
 Text(0.7735849056603774, 0.08333333333333333, 'gini = 0.007\nsamples = 169\n
value = [277, 1]\nclass = a'),
 Text(0.8113207547169812, 0.08333333333333333, 'gini = 0.105\nsamples = 130\n
value = [187, 11]\nclass = a'),
 Text(0.8679245283018868, 0.25, 'EBE <= 1.65\ngini = 0.495\nsamples = 11\nval
ue = [11, 9]\nclass = a'),
 Text(0.8490566037735849, 0.08333333333333333, 'gini = 0.0\nsamples = 5\nvalu
e = [0, 9]\nclass = b'),
 Text(0.8867924528301887, 0.08333333333333333, 'gini = 0.0\nsamples = 6\nvalu
e = [11, 0]\nclass = a'),
 Text(0.9622641509433962, 0.4166666666666667, 'NMHC <= 0.195\ngini = 0.006\ns
amples = 397\nvalue = [633, 2]\nclass = a'),
 Text(0.9433962264150944, 0.25, 'TOL <= 6.15\ngini = 0.029\nsamples = 87\nval
ue = [133, 2]\nclass = a'),
 Text(0.9245283018867925, 0.08333333333333333, 'gini = 0.0\nsamples = 82\nval
ue = [127, 0]\nclass = a'),
 Text(0.9622641509433962, 0.08333333333333333, 'gini = 0.375\nsamples = 5\nva
lue = [6, 2]\nclass = a'),
 Text(0.9811320754716981, 0.25, 'gini = 0.0\nsamples = 310\nvalue = [500, 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: 0.644033622206244
Ridge Regression: 0.639799790217638
Lasso Regression 0.3716155940503091
ElasticNet Regression: 0.5408652684713832
Logistic Regression: 0.9311102968120191
Random Forest: 0.9670201600140311
```

Random Forest Is Better!!!