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_2008.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.889999
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.040001
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.270000
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.850000
4	2008-06-01 01:00:00	1.68	0.80	1.7	3.01	0.3	105.199997	214.899994	1.61	12.120000	37.160000

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

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

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

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

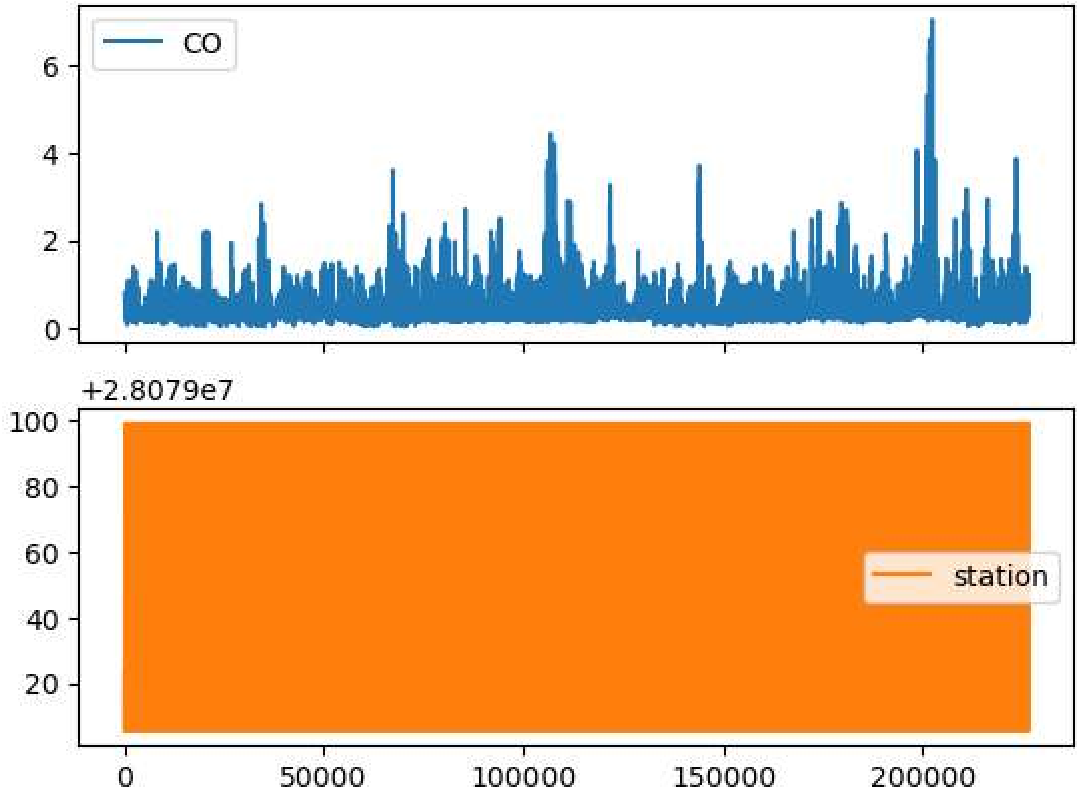
Out[7]:

	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...	...	...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

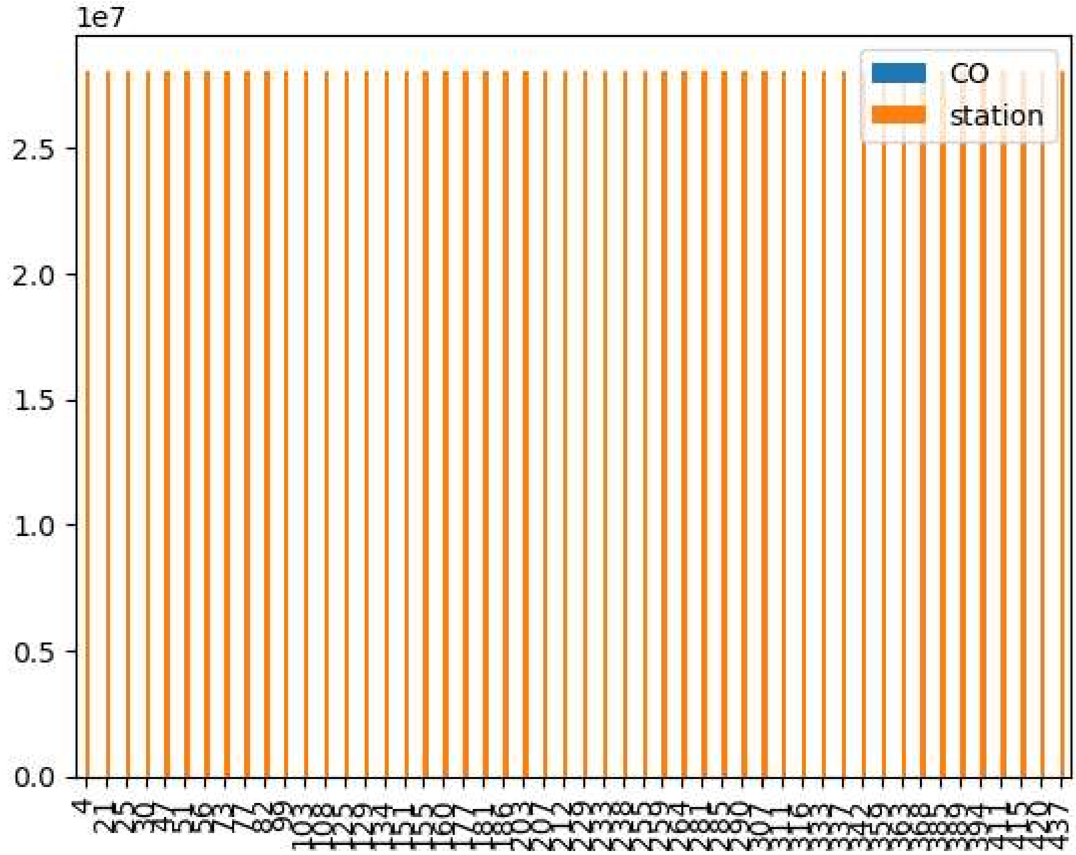
In [8]:
```
data.plot.line(subplots=True)
```

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



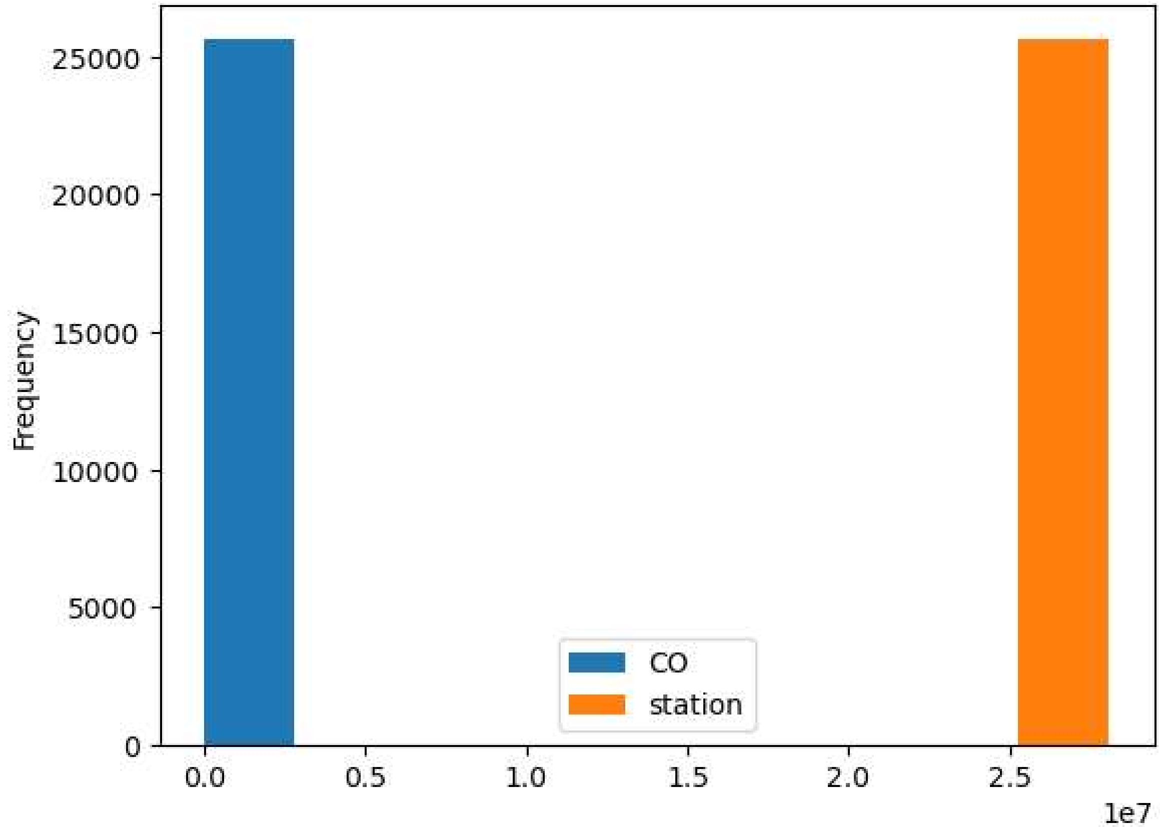
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



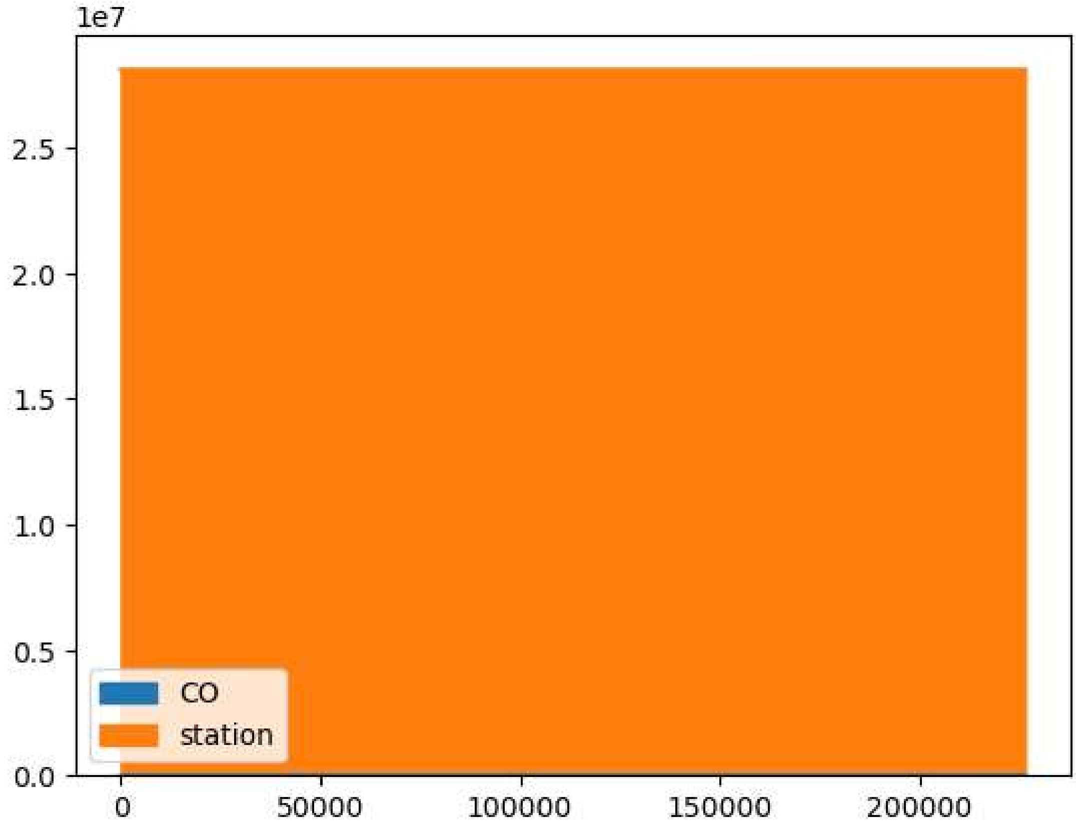
In [10]:
```
data.plot.hist()
```

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



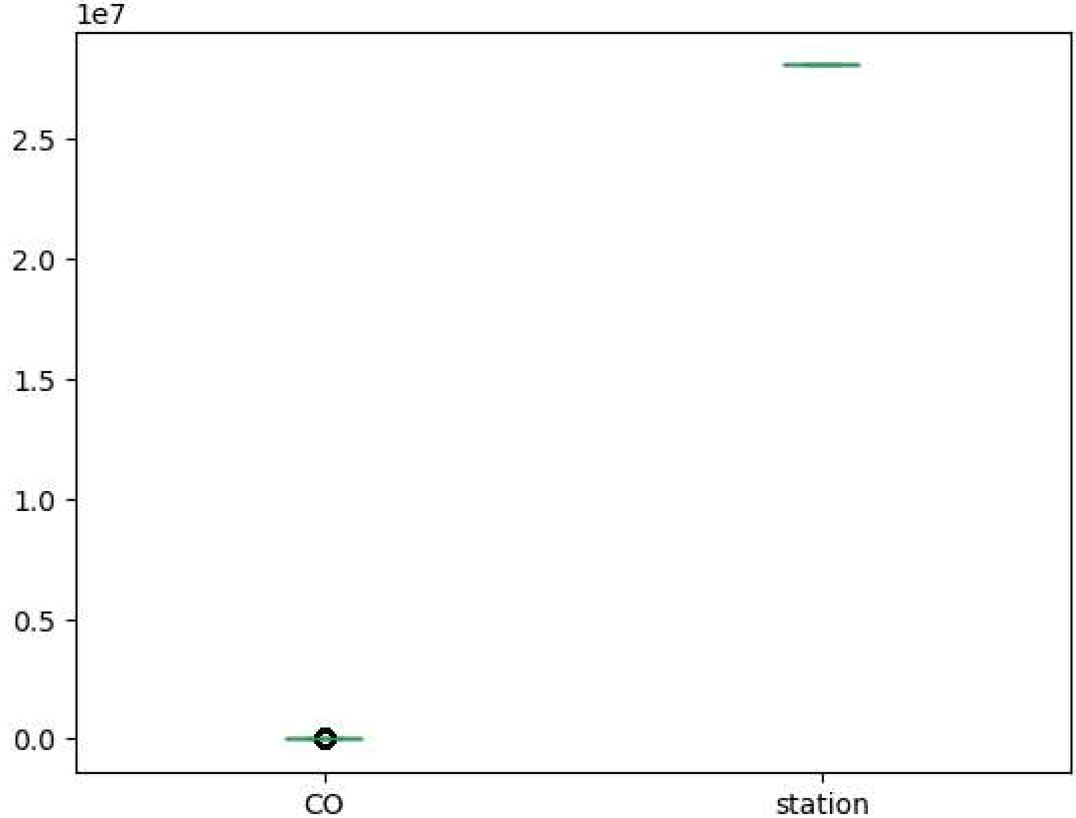
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



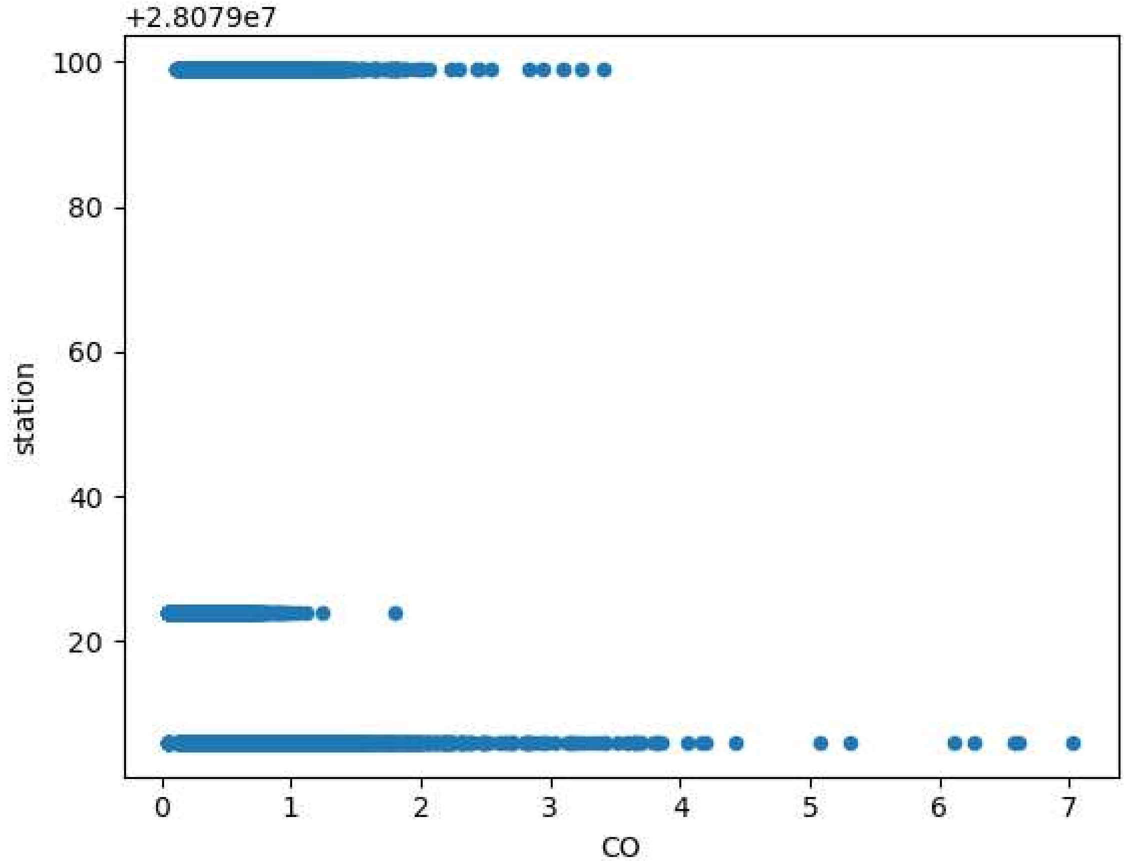
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



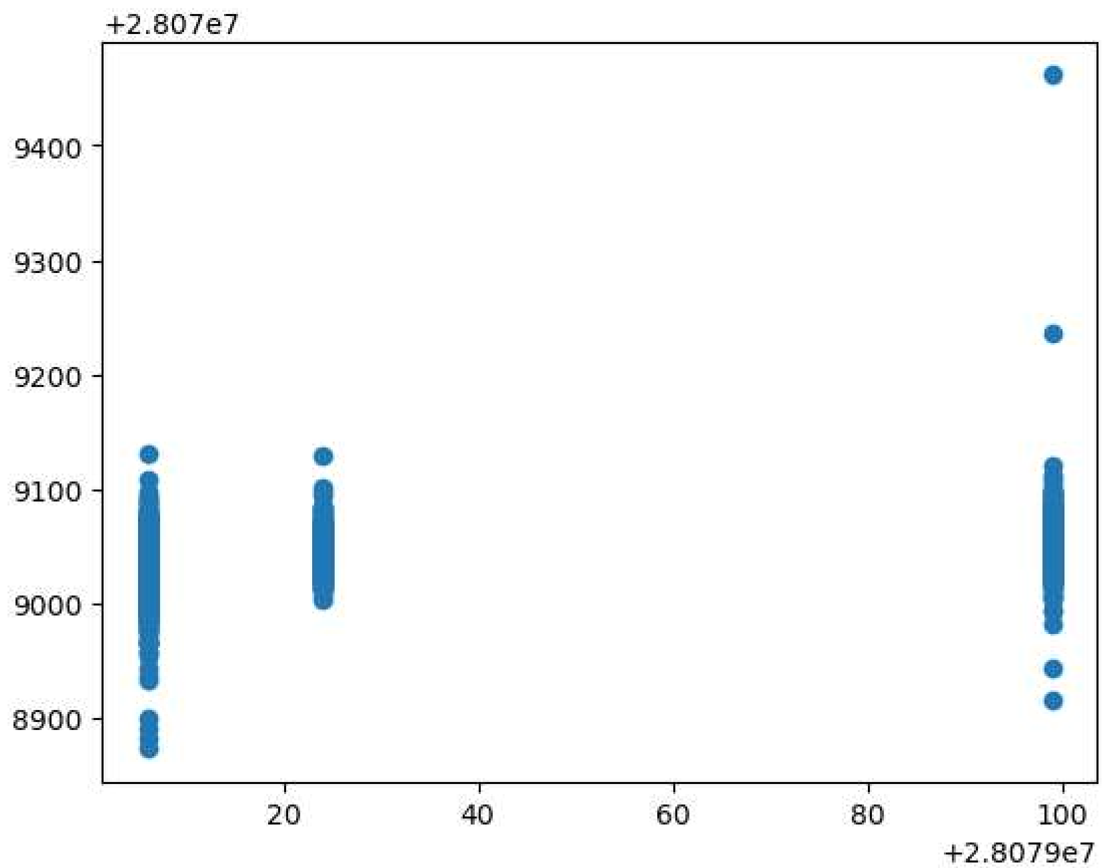
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x22bb912c390>



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

```
0.14189755406565563
0.1442773097866754
```

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)
```

```
0.14199298931038495
0.14424986480453006
```

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

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

Out[19]:
```
0.042320117094826304
```

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([-4.75245423, -0.        ,  0.        ,  3.34582701, -0.        ,
        0.04525883,  0.03148858,  1.44837803, -0.14826515,  0.13088534,
        1.59485819, -0.91313965,  0.        , -2.56737519])
```

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

Out[22]:
```
28079056.835846838
```

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

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

Out[24]:
```
0.09945769268475402
```

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)))
```

```
35.734393493612195
1488.6577699930538
38.58312804831995
```

Logistics Regression

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

In [27]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O
'PM10', 'PXY', '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)
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
logr.score(fs,target_vector)
```

Out[28]:
```
0.794194530061254
```

In [29]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
logr.predict_proba(observation)
```

Out[29]:
```
array([[8.32180727e-09, 1.19114483e-13, 9.99999992e-01]])
```

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]:

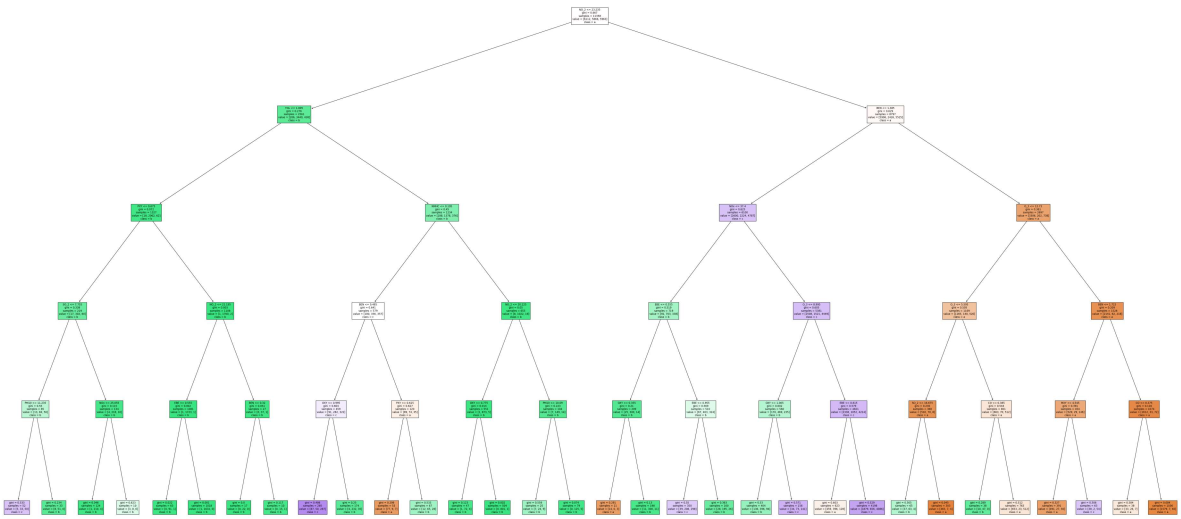
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

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(0.5, 0.9166666666666666, 'NO_2 <= 23.235\ngini = 0.667\nsamples = 11358
\nvalue = [6112, 5866, 5963]\nclass = a'),
 Text(0.25, 0.75, 'TOL <= 1.005\ngini = 0.276\nsamples = 2561\nvalue = [206,
3440, 438]\nclass = b'),
 Text(0.125, 0.5833333333333334, 'PXY <= 0.675\ngini = 0.072\nsamples = 1327
\nvalue = [18, 2062, 62]\nclass = b'),
 Text(0.0625, 0.4166666666666667, 'SO_2 <= 7.755\ngini = 0.338\nsamples = 219
\nvalue = [17, 302, 60]\nclass = b'),
 Text(0.03125, 0.25, 'PM10 <= 11.235\ngini = 0.55\nsamples = 85\nvalue = [13,
84, 50]\nclass = b'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.533\nsamples = 52\nvalue = [5,
33, 50]\nclass = c'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.234\nsamples = 33\nvalue = [8,
51, 0]\nclass = b'),
 Text(0.09375, 0.25, 'NOx <= 25.055\ngini = 0.115\nsamples = 134\nvalue = [4,
218, 10]\nclass = b'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.046\nsamples = 124\nvalue =
[1, 210, 4]\nclass = b'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.623\nsamples = 10\nvalue = [3,
8, 6]\nclass = b'),
 Text(0.1875, 0.4166666666666667, 'NO_2 <= 21.195\ngini = 0.003\nsamples = 11
08\nvalue = [1, 1760, 2]\nclass = b'),
 Text(0.15625, 0.25, 'EBE <= 0.555\ngini = 0.002\nsamples = 1081\nvalue = [1,
1723, 1]\nclass = b'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.022\nsamples = 63\nvalue = [0,
91, 1]\nclass = b'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.001\nsamples = 1018\nvalue =
[1, 1632, 0]\nclass = b'),
 Text(0.21875, 0.25, 'BEN <= 0.32\ngini = 0.051\nsamples = 27\nvalue = [0, 3
7, 1]\nclass = b'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.0\nsamples = 17\nvalue = [0, 2
2, 0]\nclass = b'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.117\nsamples = 10\nvalue = [0,
15, 1]\nclass = b'),
 Text(0.375, 0.5833333333333334, 'NMHC <= 0.195\ngini = 0.45\nsamples = 1234
\nvalue = [188, 1378, 376]\nclass = b'),
 Text(0.3125, 0.4166666666666667, 'BEN <= 0.485\ngini = 0.641\nsamples = 579
\nvalue = [180, 356, 357]\nclass = c'),
 Text(0.28125, 0.25, 'OXY <= 0.995\ngini = 0.604\nsamples = 459\nvalue = [91,
282, 322]\nclass = c'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.486\nsamples = 285\nvalue = [8
7, 50, 287]\nclass = c'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.25\nsamples = 174\nvalue = [4,
232, 35]\nclass = b'),
 Text(0.34375, 0.25, 'PXY <= 0.615\ngini = 0.627\nsamples = 120\nvalue = [89,
74, 35]\nclass = a'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.299\nsamples = 53\nvalue = [7
7, 9, 7]\nclass = a'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.533\nsamples = 67\nvalue = [1
2, 65, 28]\nclass = b'),
 Text(0.4375, 0.4166666666666667, 'NO_2 <= 20.225\ngini = 0.05\nsamples = 655
\nvalue = [8, 1022, 19]\nclass = b'),
 Text(0.40625, 0.25, 'OXY <= 0.775\ngini = 0.014\nsamples = 551\nvalue = [1,
873, 5]\nclass = b'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.123\nsamples = 47\nvalue = [1,
72, 4]\nclass = b'),
 Text(0.421875, 0.08333333333333333, 'gini = 0.002\nsamples = 504\nvalue =
```

```
[0, 801, 1]\nclass = b'),
 Text(0.46875, 0.25, 'PM10 <= 10.09\ngini = 0.223\nsamples = 104\nvalue = [7,
149, 14]\nclass = b'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.559\nsamples = 25\nvalue = [7,
24, 9]\nclass = b'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.074\nsamples = 79\nvalue = [0,
125, 5]\nclass = b'),
 Text(0.75, 0.75, 'BEN <= 1.385\ngini = 0.629\nsamples = 8797\nvalue = [5906,
2426, 5525]\nclass = a'),
 Text(0.625, 0.5833333333333334, 'NOx <= 37.4\ngini = 0.625\nsamples = 6100\n
value = [2600, 2224, 4787]\nclass = c'),
 Text(0.5625, 0.4166666666666667, 'EBE <= 0.555\ngini = 0.519\nsamples = 719
\nvalue = [92, 703, 338]\nclass = b'),
 Text(0.53125, 0.25, 'OXY <= 0.355\ngini = 0.21\nsamples = 209\nvalue = [25,
300, 14]\nclass = b'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.291\nsamples = 11\nvalue = [1
4, 0, 3]\nclass = a'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.13\nsamples = 198\nvalue = [1
1, 300, 11]\nclass = b'),
 Text(0.59375, 0.25, 'EBE <= 0.955\ngini = 0.569\nsamples = 510\nvalue = [67,
403, 324]\nclass = b'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.55\nsamples = 350\nvalue = [3
9, 208, 298]\nclass = c'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.363\nsamples = 160\nvalue = [2
8, 195, 26]\nclass = b'),
 Text(0.6875, 0.4166666666666667, 'O_3 <= 8.995\ngini = 0.605\nsamples = 5381
\nvalue = [2508, 1521, 4449]\nclass = c'),
 Text(0.65625, 0.25, 'OXY <= 1.005\ngini = 0.602\nsamples = 560\nvalue = [17
0, 469, 235]\nclass = b'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.53\nsamples = 404\nvalue = [13
6, 396, 94]\nclass = b'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.571\nsamples = 156\nvalue = [3
4, 73, 141]\nclass = c'),
 Text(0.71875, 0.25, 'EBE <= 0.615\ngini = 0.579\nsamples = 4821\nvalue = [23
38, 1052, 4214]\nclass = c'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.603\nsamples = 623\nvalue = [4
59, 396, 128]\nclass = a'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.529\nsamples = 4198\nvalue =
[1879, 656, 4086]\nclass = c'),
 Text(0.875, 0.5833333333333334, 'O_3 <= 12.73\ngini = 0.361\nsamples = 2697
\nvalue = [3306, 202, 738]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'O_3 <= 5.595\ngini = 0.505\nsamples = 1169
\nvalue = [1165, 140, 520]\nclass = a'),
 Text(0.78125, 0.25, 'SO_2 <= 16.675\ngini = 0.236\nsamples = 368\nvalue = [5
02, 70, 8]\nclass = a'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.505\nsamples = 65\nvalue = [3
7, 63, 4]\nclass = b'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.045\nsamples = 303\nvalue = [4
65, 7, 4]\nclass = a'),
 Text(0.84375, 0.25, 'CO <= 0.385\ngini = 0.544\nsamples = 801\nvalue = [663,
70, 512]\nclass = a'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.289\nsamples = 38\nvalue = [1
0, 47, 0]\nclass = b'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.512\nsamples = 763\nvalue = [6
53, 23, 512]\nclass = a'),
 Text(0.9375, 0.4166666666666667, 'BEN <= 1.715\ngini = 0.209\nsamples = 1528
\nvalue = [2141, 62, 218]\nclass = a'),
```

```
 Text(0.90625, 0.25, 'MXY <= 4.545\ngini = 0.391\nsamples = 454\nvalue = [52
9, 29, 146]\nclass = a'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.327\nsamples = 391\nvalue = [4
91, 27, 92]\nclass = a'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.506\nsamples = 63\nvalue = [3
8, 2, 54]\nclass = c'),
 Text(0.96875, 0.25, 'CO <= 0.375\ngini = 0.116\nsamples = 1074\nvalue = [161
2, 33, 72]\nclass = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.584\nsamples = 38\nvalue = [3
3, 26, 7]\nclass = a'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.084\nsamples = 1036\nvalue =
[1579, 7, 65]\nclass = a')]
```



Conclusion

In [34]:
```python
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: 0.14189755406565563
Ridge Regression: 0.14199298931038495
Lasso Regression 0.042320117094826304
ElasticNet Regression: 0.09945769268475402
Logistic Regression: 0.794194530061254
Random Forest: 0.852460771963469
```

Logistic Is Better!!!