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_2006.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	I
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.88	97.570000	40.2[illegible]
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.10	25.820000	
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.43	34.419998	
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.83	28.260000	
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.99	54.180000	

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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24758 non-null  object 
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

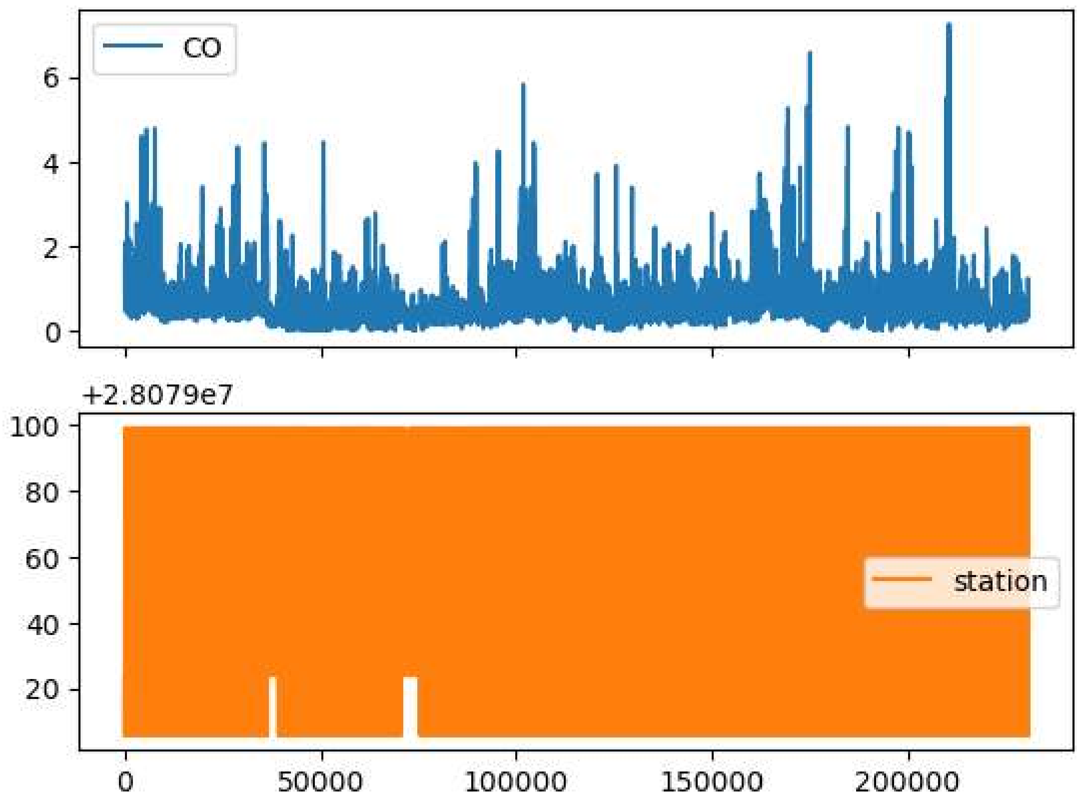
Out[7]:

	CO	station
5	1.69	28079006
22	0.79	28079024
25	1.47	28079099
31	0.85	28079006
48	0.79	28079024
...	...	...
230538	0.40	28079024
230541	0.94	28079099
230547	1.06	28079006
230564	0.32	28079024
230567	0.74	28079099

24758 rows × 2 columns

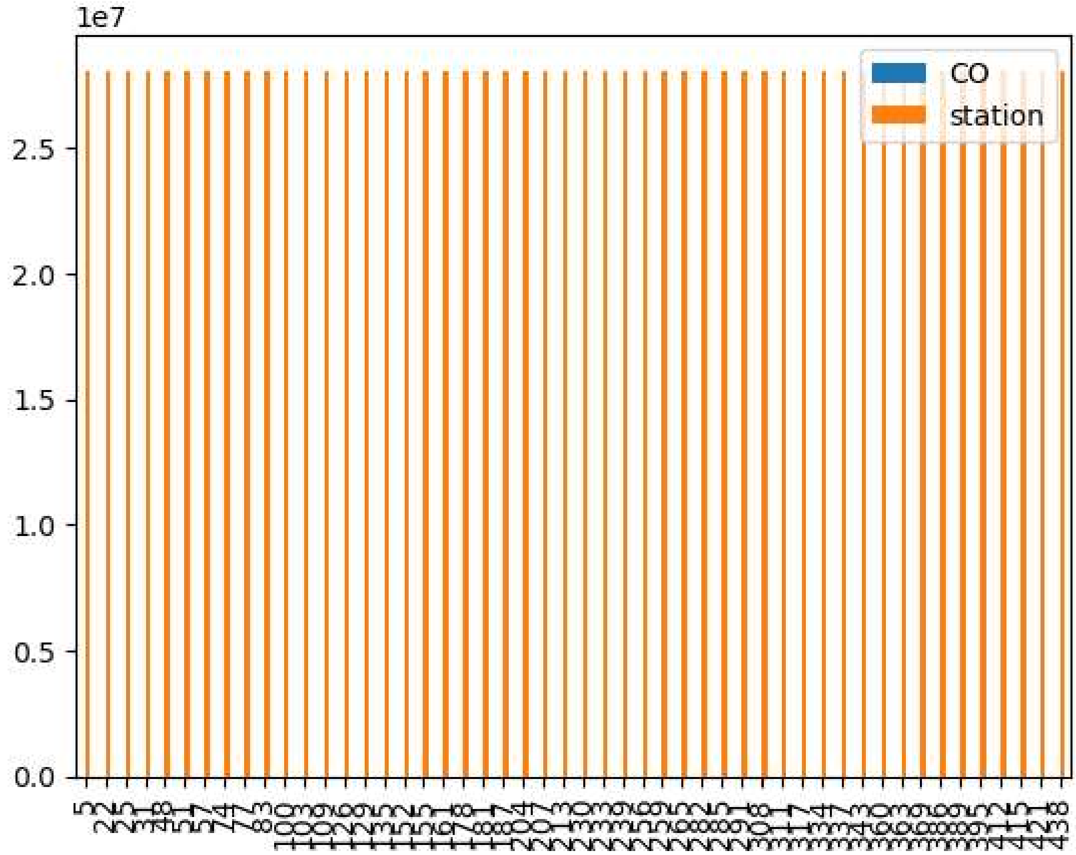
In [8]:
```
data.plot.line(subplots=True)
```

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



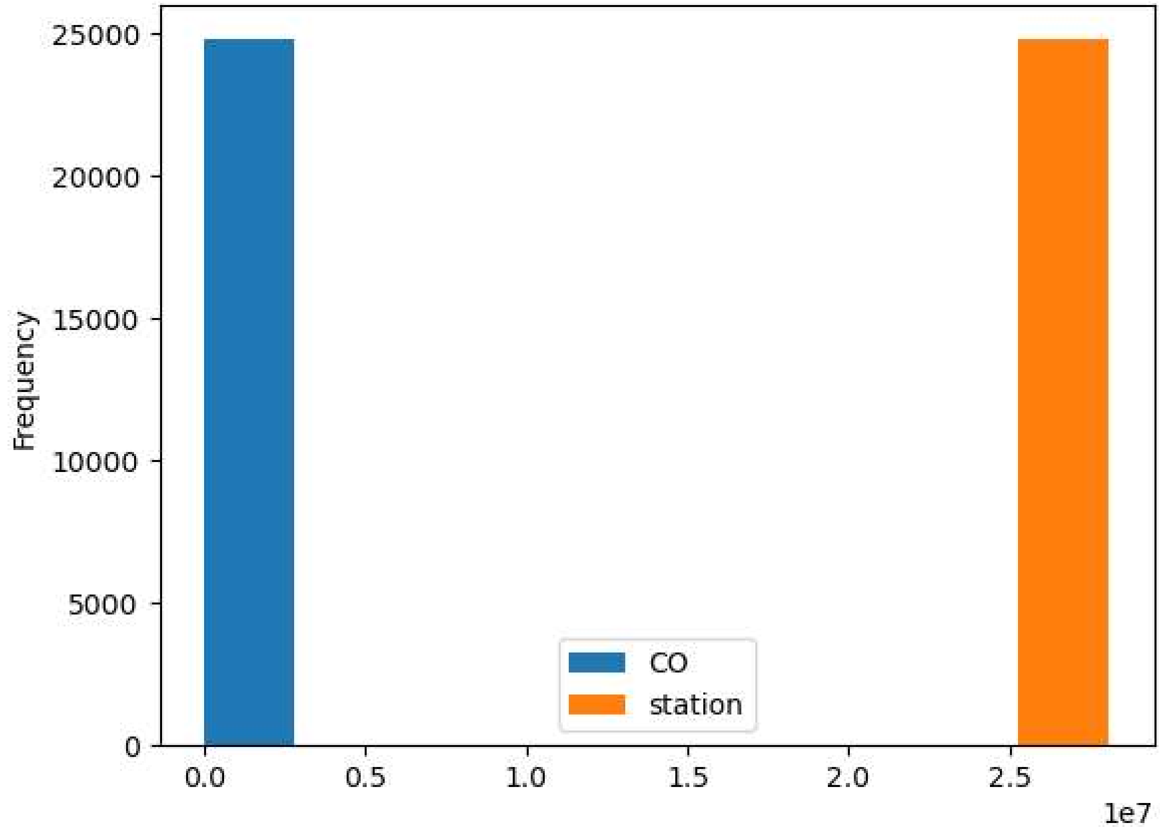
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



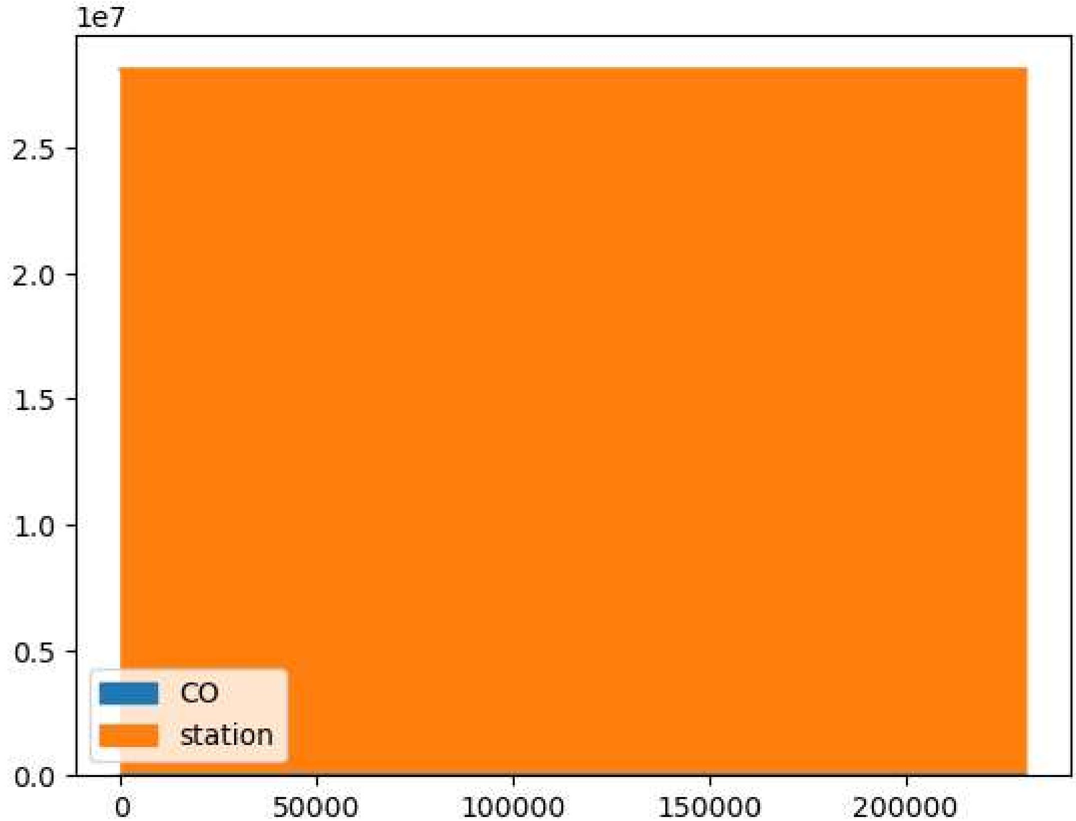
In [10]:
```
data.plot.hist()
```

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



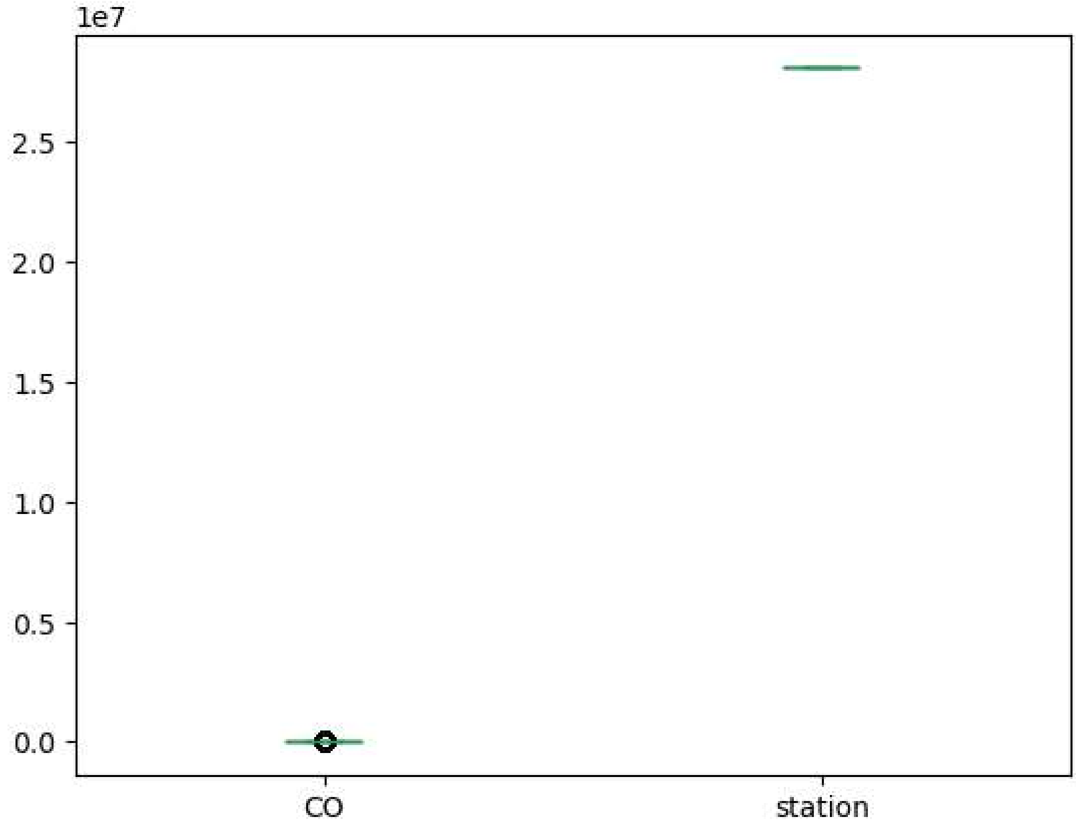
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



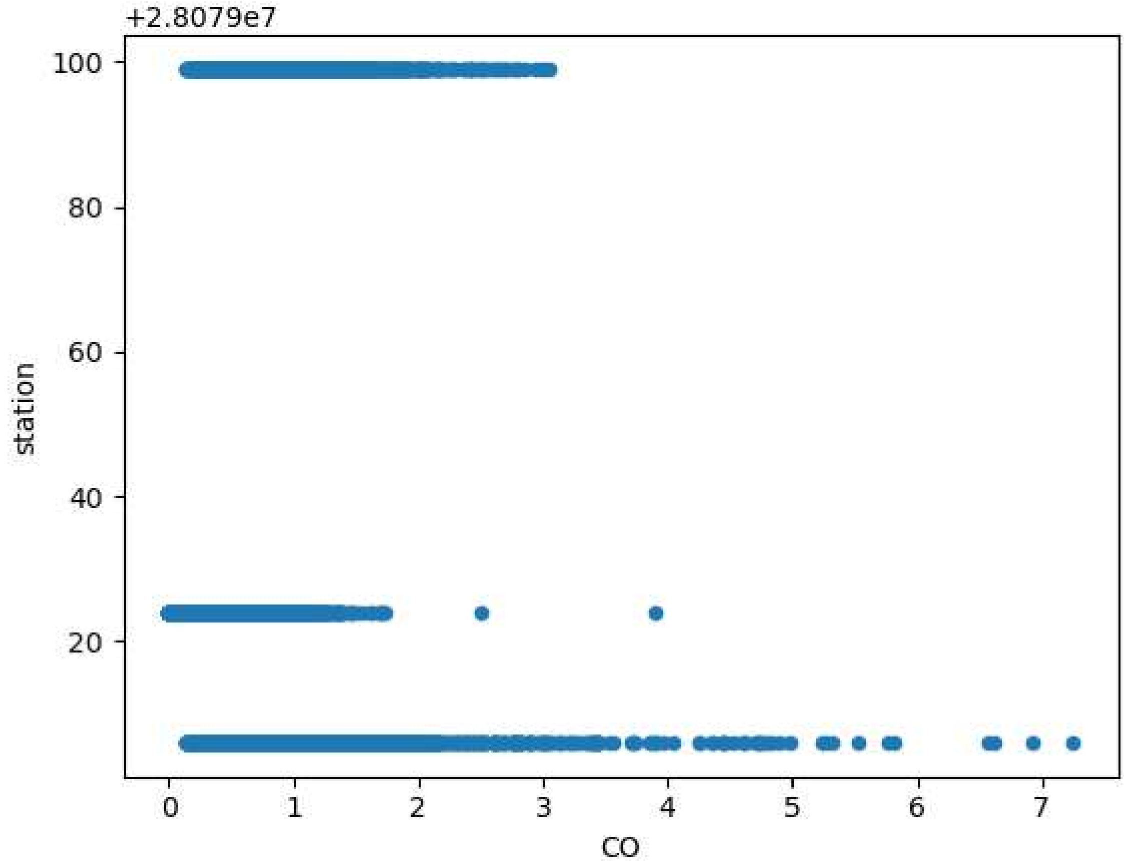
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



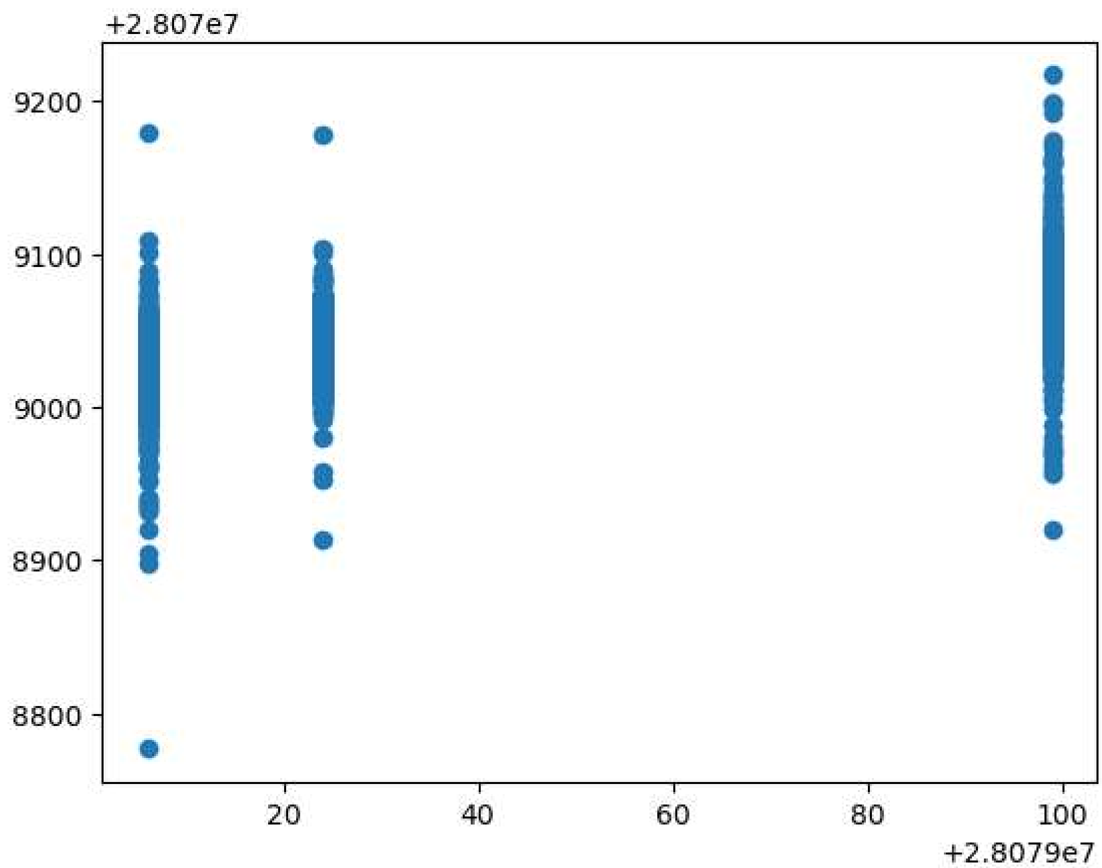
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x283881b2990>



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

```
0.40921831646492834
0.3864296243722516
```

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)
```

```
0.40824098689997745
0.3857869451715713
```

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

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

Out[19]:
```
0.05572457582786561
```

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([-8.18801519e+00,  0.00000000e+00, -8.53752527e+00,  3.03242238e+00,
        4.05325260e-01, -8.04906072e-03,  2.15852552e-03,  3.98642648e+00,
       -1.30554825e-01,  3.24195094e-01,  2.60821471e+00, -4.83891365e-01,
        5.69689624e-01, -1.11244241e+00])
```

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

Out[22]:
```
28079052.01883103
```

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

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

Out[24]:
```
0.23339678727544355
```

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)))
```

```
32.521672931825265
1263.3368086694531
35.543449588770265
```

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]:
```
0.8741416915744405
```

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

Out[29]:
```
array([[3.55577517e-15, 7.80744348e-29, 1.00000000e+00]])
```

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]:

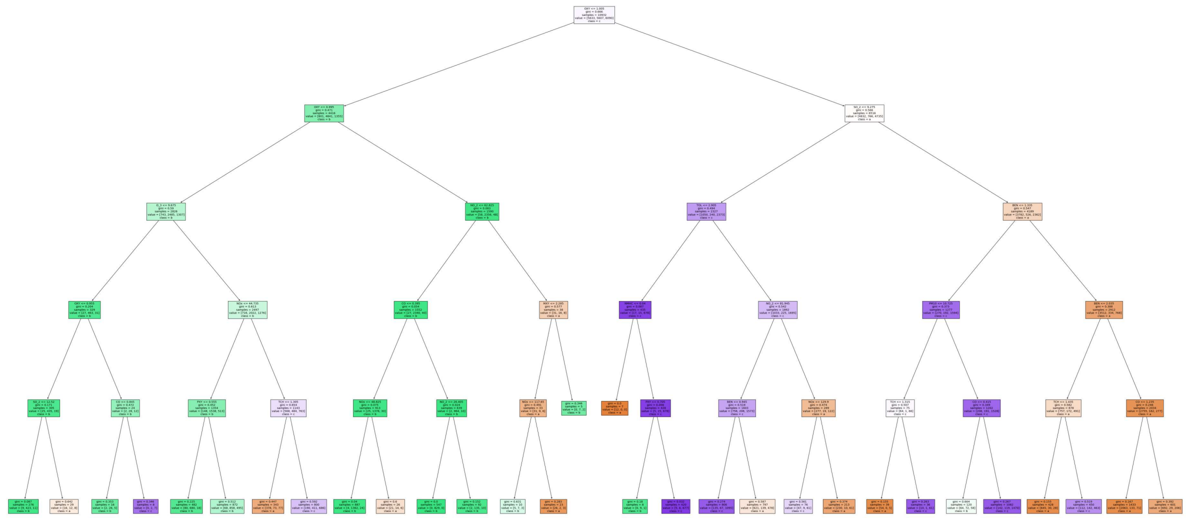
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

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, 'OXY <= 1.005\ngini = 0.666\nsamples = 10932\n
value = [5633, 5607, 6090]\nclass = c'),
 Text(0.27155172413793105, 0.75, 'OXY <= 0.995\ngini = 0.471\nsamples = 4416
\nvalue = [801, 4841, 1355]\nclass = b'),
 Text(0.13793103448275862, 0.5833333333333334, 'O_3 <= 9.675\ngini = 0.59\nsa
mples = 2826\nvalue = [743, 2485, 1307]\nclass = b'),
 Text(0.06896551724137931, 0.4166666666666667, 'OXY <= 0.955\ngini = 0.204\ns
amples = 329\nvalue = [27, 463, 31]\nclass = b'),
 Text(0.034482758620689655, 0.25, 'SO_2 <= 12.52\ngini = 0.171\nsamples = 305
\nvalue = [25, 435, 19]\nclass = b'),
 Text(0.017241379310344827, 0.08333333333333333, 'gini = 0.087\nsamples = 276
\nvalue = [9, 423, 11]\nclass = b'),
 Text(0.05172413793103448, 0.08333333333333333, 'gini = 0.642\nsamples = 29\n
value = [16, 12, 8]\nclass = a'),
 Text(0.10344827586206896, 0.25, 'CO <= 0.845\ngini = 0.472\nsamples = 24\nva
lue = [2, 28, 12]\nclass = b'),
 Text(0.08620689655172414, 0.08333333333333333, 'gini = 0.353\nsamples = 16\n
value = [2, 26, 5]\nclass = b'),
 Text(0.1206896551724138, 0.08333333333333333, 'gini = 0.346\nsamples = 8\nva
lue = [0, 2, 7]\nclass = c'),
 Text(0.20689655172413793, 0.4166666666666667, 'NOx <= 44.735\ngini = 0.613\n
samples = 2497\nvalue = [716, 2022, 1276]\nclass = b'),
 Text(0.1724137931034483, 0.25, 'PXY <= 0.555\ngini = 0.452\nsamples = 1354\n
value = [148, 1538, 513]\nclass = b'),
 Text(0.15517241379310345, 0.08333333333333333, 'gini = 0.225\nsamples = 482
\nvalue = [80, 680, 18]\nclass = b'),
 Text(0.1896551724137931, 0.08333333333333333, 'gini = 0.512\nsamples = 872\n
value = [68, 858, 495]\nclass = b'),
 Text(0.2413793103448276, 0.25, 'TCH <= 1.305\ngini = 0.654\nsamples = 1143\n
value = [568, 484, 763]\nclass = c'),
 Text(0.22413793103448276, 0.08333333333333333, 'gini = 0.447\nsamples = 343
\nvalue = [378, 73, 77]\nclass = a'),
 Text(0.25862068965517243, 0.08333333333333333, 'gini = 0.592\nsamples = 800
\nvalue = [190, 411, 686]\nclass = c'),
 Text(0.4051724137931034, 0.5833333333333334, 'NO_2 <= 62.825\ngini = 0.083\n
samples = 1590\nvalue = [58, 2356, 48]\nclass = b'),
 Text(0.3448275862068966, 0.4166666666666667, 'CO <= 0.395\ngini = 0.054\nsam
ples = 1552\nvalue = [27, 2340, 40]\nclass = b'),
 Text(0.3103448275862069, 0.25, 'NOx <= 48.825\ngini = 0.075\nsamples = 913\n
value = [25, 1376, 30]\nclass = b'),
 Text(0.29310344827586204, 0.08333333333333333, 'gini = 0.04\nsamples = 887\n
value = [4, 1362, 24]\nclass = b'),
 Text(0.3275862068965517, 0.08333333333333333, 'gini = 0.6\nsamples = 26\nval
ue = [21, 14, 6]\nclass = a'),
 Text(0.3793103448275862, 0.25, 'NO_2 <= 26.405\ngini = 0.024\nsamples = 639
\nvalue = [2, 964, 10]\nclass = b'),
 Text(0.3620689655172414, 0.08333333333333333, 'gini = 0.0\nsamples = 547\nva
lue = [0, 829, 0]\nclass = b'),
 Text(0.39655172413793105, 0.08333333333333333, 'gini = 0.152\nsamples = 92\n
value = [2, 135, 10]\nclass = b'),
 Text(0.46551724137931033, 0.4166666666666667, 'MXY <= 2.285\ngini = 0.577\ns
amples = 38\nvalue = [31, 16, 8]\nclass = a'),
 Text(0.4482758620689655, 0.25, 'NOx <= 117.65\ngini = 0.491\nsamples = 33\nv
alue = [31, 9, 6]\nclass = a'),
 Text(0.43103448275862066, 0.08333333333333333, 'gini = 0.631\nsamples = 10\n
value = [5, 7, 3]\nclass = b'),
 Text(0.46551724137931033, 0.08333333333333333, 'gini = 0.283\nsamples = 23\n
```

```
value = [26, 2, 3]\nclass = a'),
 Text(0.4827586206896552, 0.25, 'gini = 0.346\nsamples = 5\nvalue = [0, 7, 2]
\nclass = b'),
 Text(0.728448275862069, 0.75, 'SO_2 <= 9.275\ngini = 0.566\nsamples = 6516\n
value = [4832, 766, 4735]\nclass = a'),
 Text(0.5948275862068966, 0.5833333333333334, 'TOL <= 2.905\ngini = 0.494\nsa
mples = 2327\nvalue = [1050, 240, 2373]\nclass = c'),
 Text(0.5344827586206896, 0.4166666666666667, 'NMHC <= 0.04\ngini = 0.087\nsa
mples = 435\nvalue = [17, 15, 678]\nclass = c'),
 Text(0.5172413793103449, 0.25, 'gini = 0.0\nsamples = 7\nvalue = [12, 0, 0]
\nclass = a'),
 Text(0.5517241379310345, 0.25, 'PXY <= 0.705\ngini = 0.056\nsamples = 428\nv
alue = [5, 15, 678]\nclass = c'),
 Text(0.5344827586206896, 0.08333333333333333, 'gini = 0.18\nsamples = 8\nval
ue = [0, 9, 1]\nclass = b'),
 Text(0.5689655172413793, 0.08333333333333333, 'gini = 0.032\nsamples = 420\n
value = [5, 6, 677]\nclass = c'),
 Text(0.6551724137931034, 0.4166666666666667, 'NO_2 <= 81.945\ngini = 0.542\n
samples = 1892\nvalue = [1033, 225, 1695]\nclass = c'),
 Text(0.6206896551724138, 0.25, 'BEN <= 0.945\ngini = 0.519\nsamples = 1603\n
value = [756, 206, 1573]\nclass = c'),
 Text(0.603448275862069, 0.08333333333333333, 'gini = 0.274\nsamples = 806\nv
alue = [135, 67, 1095]\nclass = c'),
 Text(0.6379310344827587, 0.08333333333333333, 'gini = 0.587\nsamples = 797\n
value = [621, 139, 478]\nclass = a'),
 Text(0.6896551724137931, 0.25, 'NOx <= 129.9\ngini = 0.474\nsamples = 289\nv
alue = [277, 19, 122]\nclass = a'),
 Text(0.6724137931034483, 0.08333333333333333, 'gini = 0.561\nsamples = 76\nv
alue = [47, 9, 61]\nclass = c'),
 Text(0.7068965517241379, 0.08333333333333333, 'gini = 0.374\nsamples = 213\n
value = [230, 10, 61]\nclass = a'),
 Text(0.8620689655172413, 0.5833333333333334, 'BEN <= 1.335\ngini = 0.547\nsa
mples = 4189\nvalue = [3782, 526, 2362]\nclass = a'),
 Text(0.7931034482758621, 0.4166666666666667, 'PM10 <= 10.725\ngini = 0.373\n
samples = 1277\nvalue = [270, 192, 1594]\nclass = c'),
 Text(0.7586206896551724, 0.25, 'TCH <= 1.315\ngini = 0.507\nsamples = 75\nva
lue = [64, 1, 66]\nclass = c'),
 Text(0.7413793103448276, 0.08333333333333333, 'gini = 0.155\nsamples = 39\nv
alue = [54, 0, 5]\nclass = a'),
 Text(0.7758620689655172, 0.08333333333333333, 'gini = 0.263\nsamples = 36\nv
alue = [10, 1, 61]\nclass = c'),
 Text(0.8275862068965517, 0.25, 'CO <= 0.415\ngini = 0.349\nsamples = 1202\nv
alue = [206, 191, 1528]\nclass = c'),
 Text(0.8103448275862069, 0.08333333333333333, 'gini = 0.664\nsamples = 120\n
value = [64, 72, 58]\nclass = b'),
 Text(0.8448275862068966, 0.08333333333333333, 'gini = 0.267\nsamples = 1082
\nvalue = [142, 119, 1470]\nclass = c'),
 Text(0.9310344827586207, 0.4166666666666667, 'BEN <= 2.035\ngini = 0.388\nsa
mples = 2912\nvalue = [3512, 334, 768]\nclass = a'),
 Text(0.896551724137931, 0.25, 'TCH <= 1.435\ngini = 0.582\nsamples = 878\nva
lue = [757, 172, 491]\nclass = a'),
 Text(0.8793103448275862, 0.08333333333333333, 'gini = 0.155\nsamples = 428\n
value = [645, 30, 28]\nclass = a'),
 Text(0.9137931034482759, 0.08333333333333333, 'gini = 0.519\nsamples = 450\n
value = [112, 142, 463]\nclass = c'),
 Text(0.9655172413793104, 0.25, 'CO <= 1.235\ngini = 0.246\nsamples = 2034\nv
alue = [2755, 162, 277]\nclass = a'),
```

```
 Text(0.9482758620689655, 0.08333333333333333, 'gini = 0.167\nsamples = 1433
\nvalue = [2063, 133, 71]\nclass = a'),
 Text(0.9827586206896551, 0.08333333333333333, 'gini = 0.392\nsamples = 601\n
value = [692, 29, 206]\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: 0.40921831646492834
Ridge Regression: 0.40824098689997745
Lasso Regression 0.05572457582786561
ElasticNet Regression: 0.23339678727544355
Logistic Regression: 0.8741416915744405
Random Forest: 0.877668782458165
```

Logistic Is Better!!!