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_2004.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2004-08-01 01:00:00	NaN	0.66	NaN	NaN	NaN	89.550003	118.900002	NaN	40.020000	39.990002
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22.950001
2	2004-08-01 01:00:00	NaN	1.02	NaN	NaN	NaN	93.389999	138.600006	NaN	20.860001	49.480000
3	2004-08-01 01:00:00	NaN	0.53	NaN	NaN	NaN	87.290001	105.000000	NaN	36.730000	31.070000
4	2004-08-01 01:00:00	NaN	0.17	NaN	NaN	NaN	34.910000	35.349998	NaN	86.269997	54.080002

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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     19397 non-null  object 
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

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

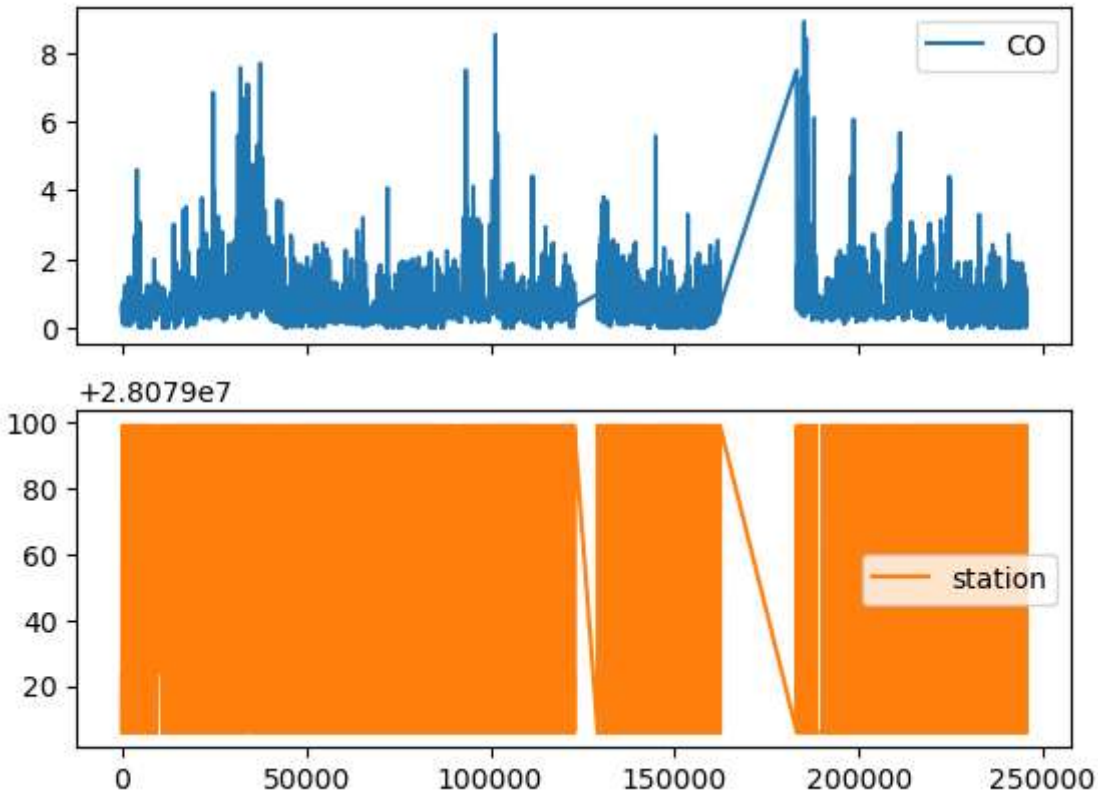
Out[7]:

	CO	station
5	0.63	28079006
22	0.36	28079024
26	0.46	28079099
32	0.67	28079006
49	0.30	28079024
...	...	...
245463	0.08	28079024
245467	0.67	28079099
245473	1.12	28079006
245491	0.21	28079024
245495	0.67	28079099

19397 rows × 2 columns

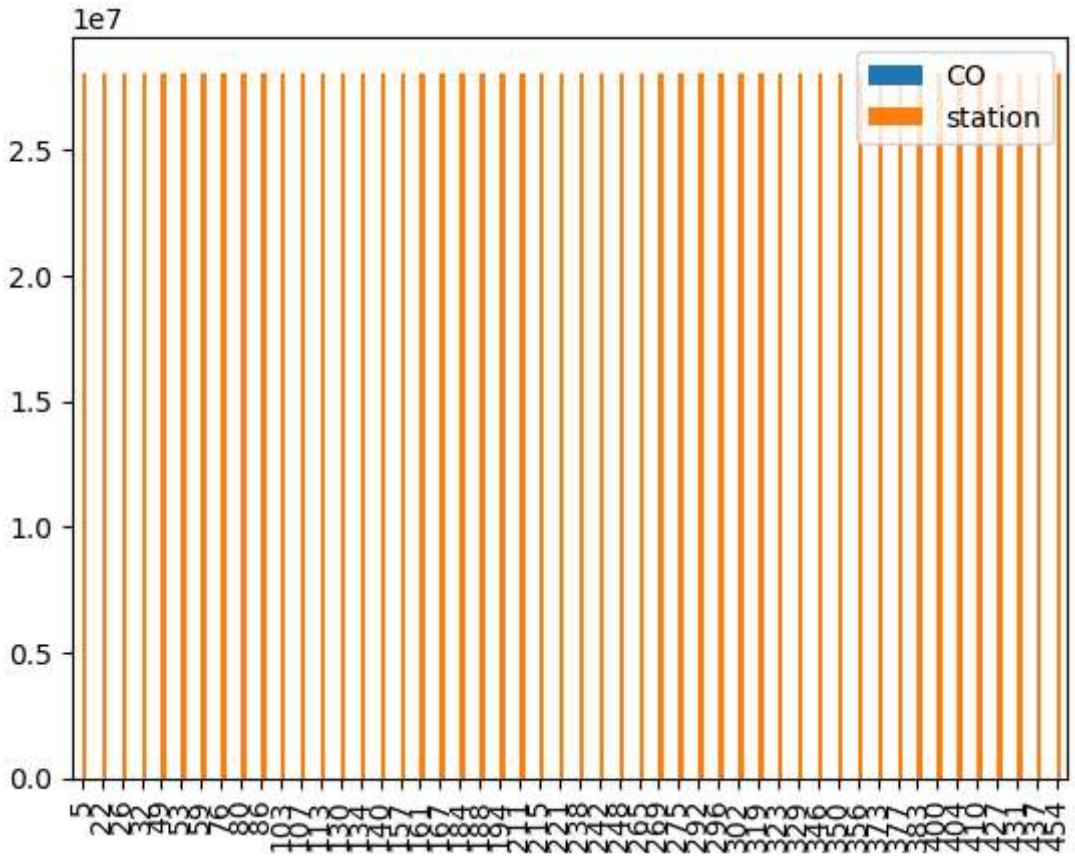
In [8]:
```
data.plot.line(subplots=True)
```

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



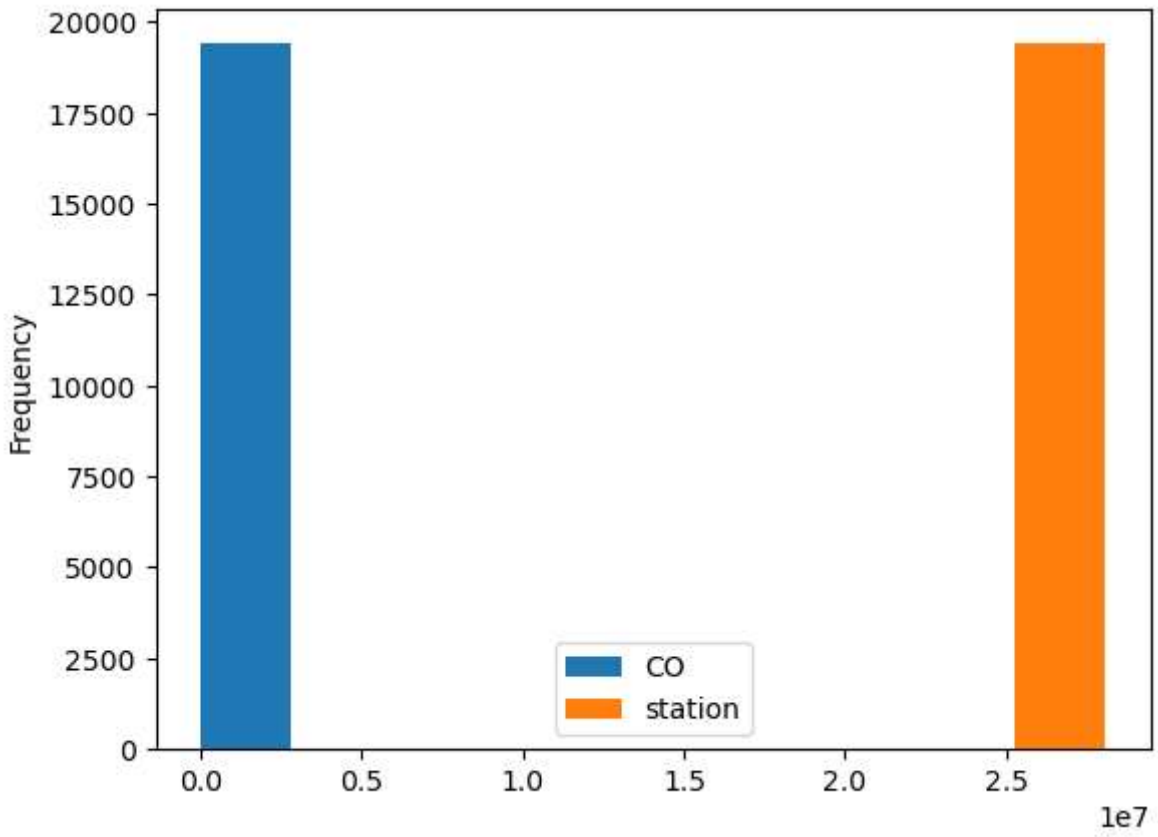
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



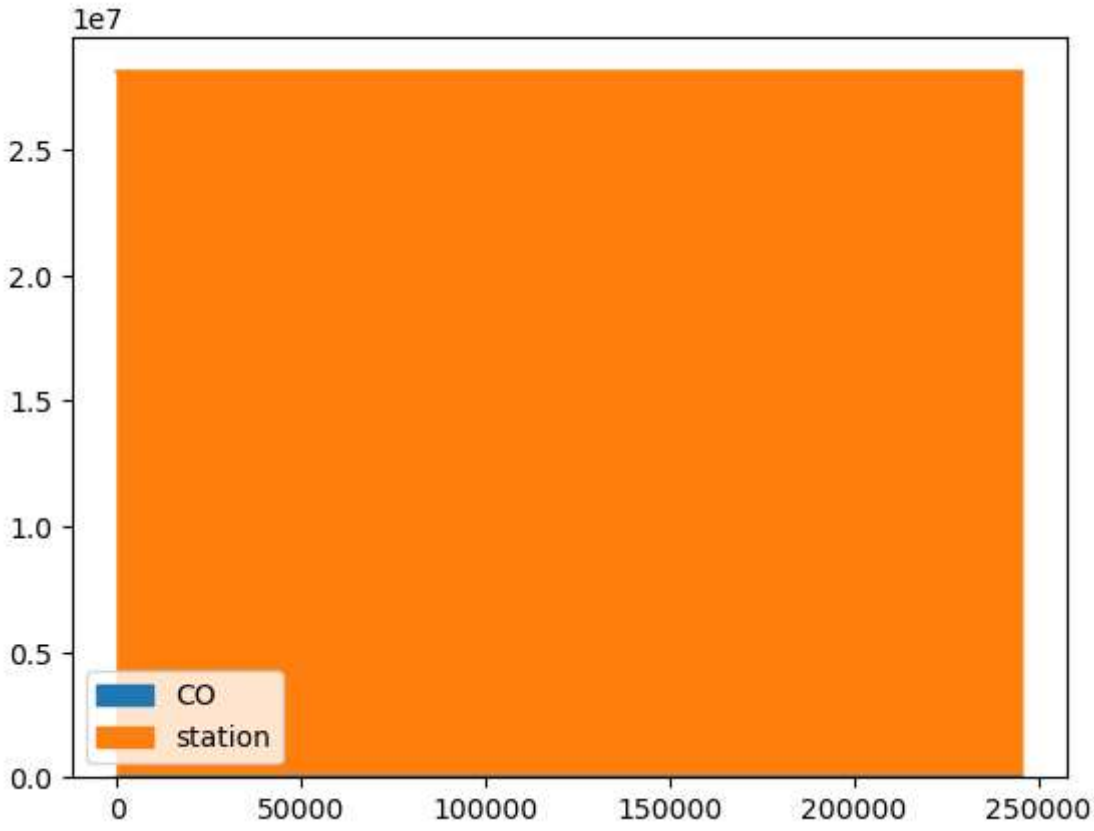
In [10]:
```
data.plot.hist()
```

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



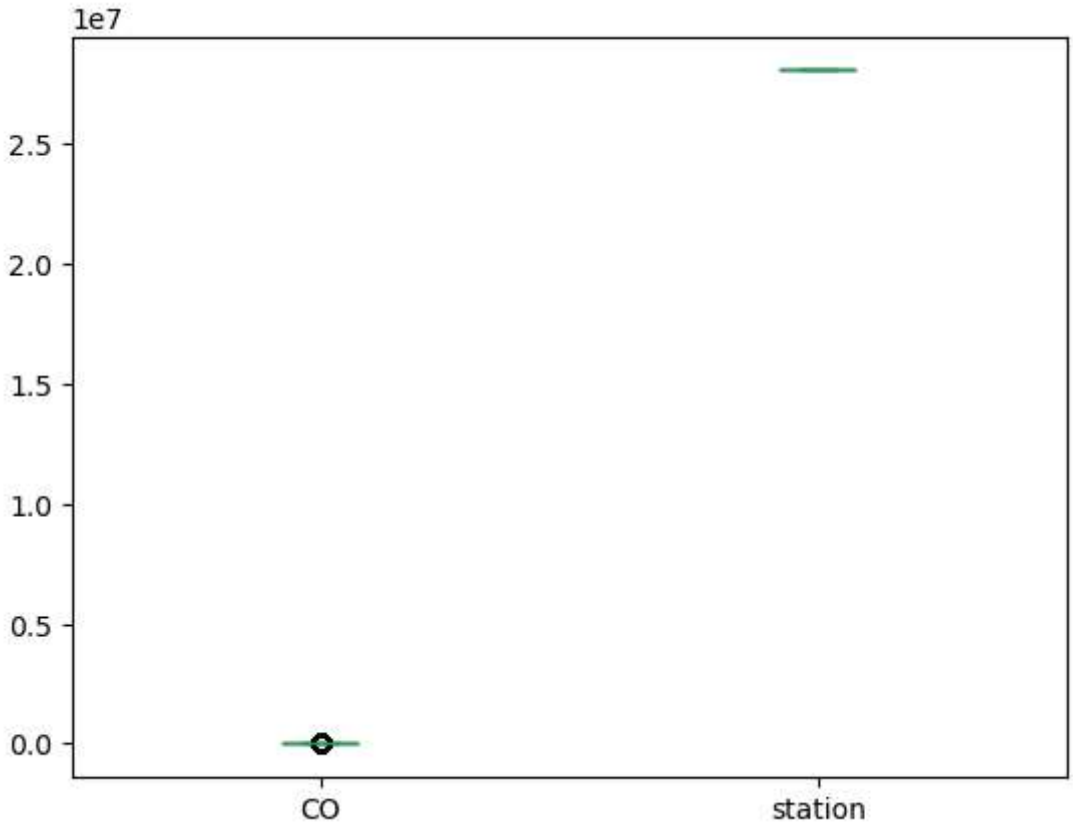
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



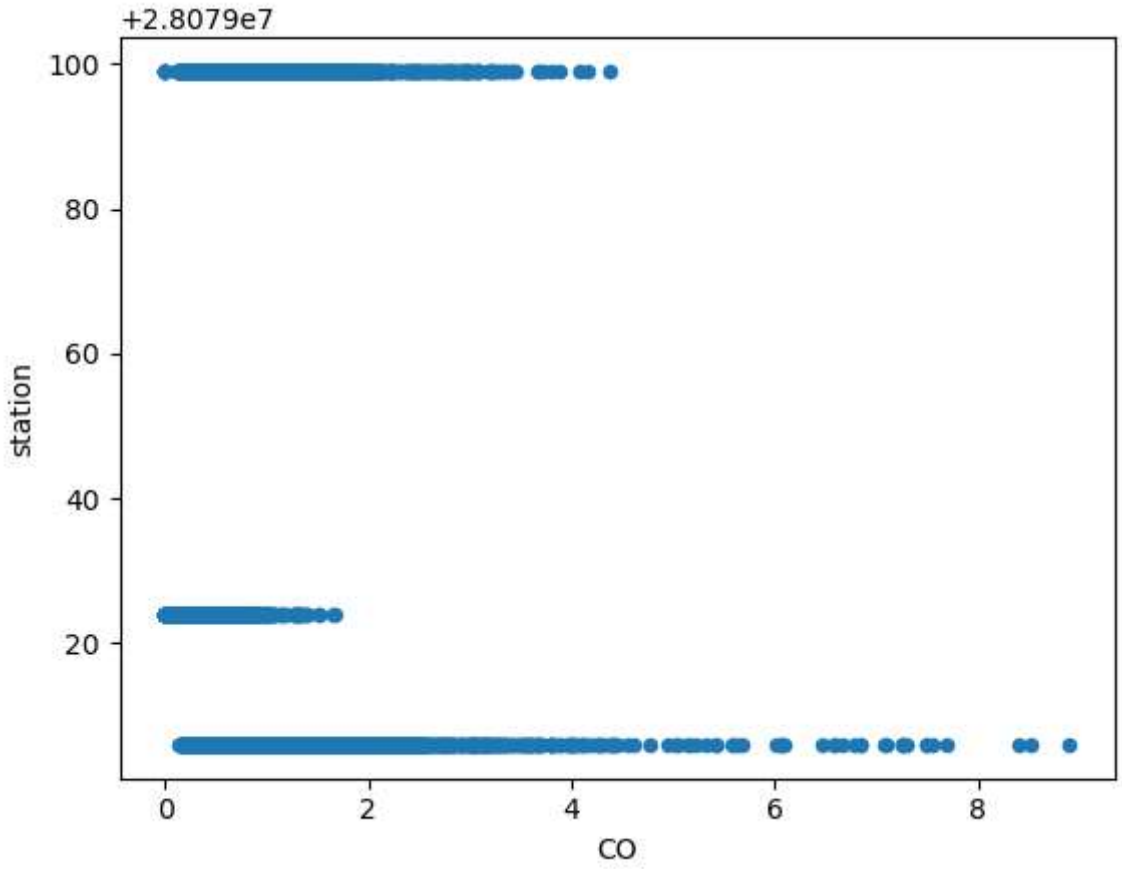
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



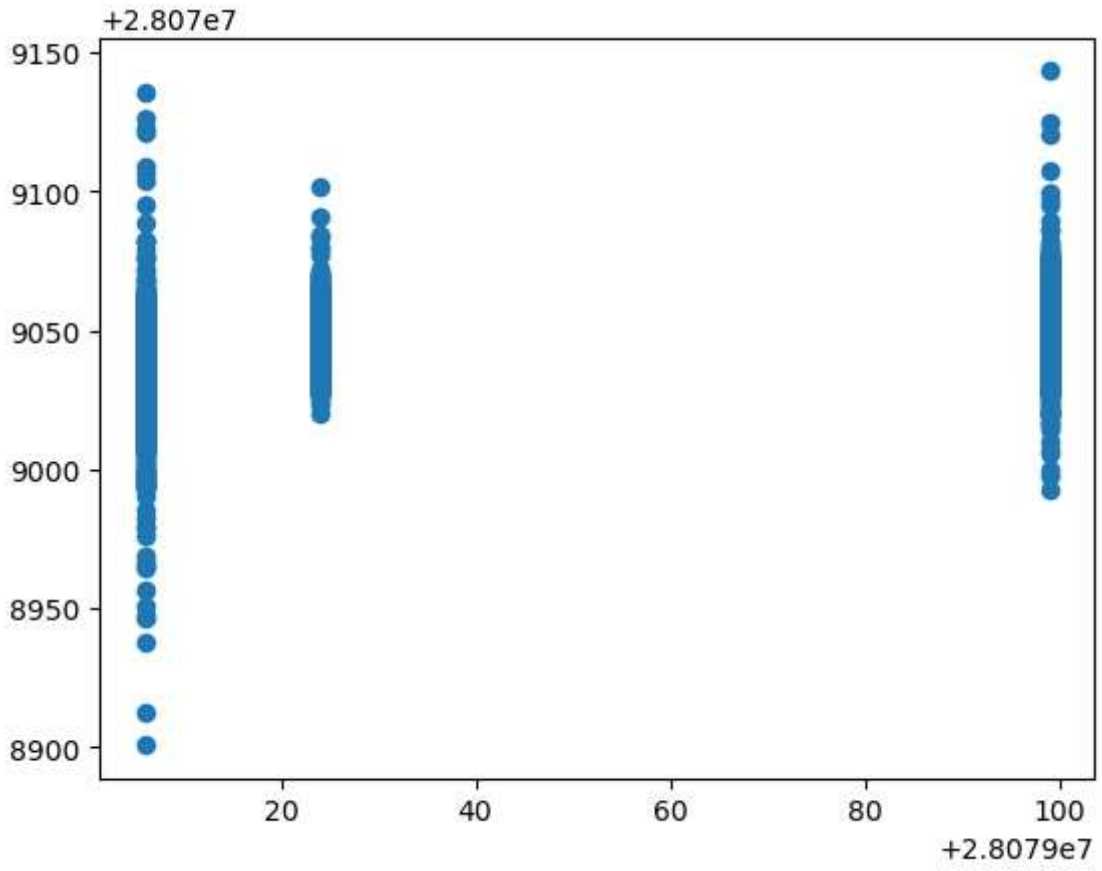
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1c4bc8cc490>



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

```
0.11799091137651152
0.10125254041546794
```

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)
```

```
0.1160044793713958
0.10093519654550476
```

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

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

Out[19]:
```
0.06045121805425291
```

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.        ,  0.50514994,  1.41453796, -1.88612795,  0.        ,
       -0.16008602, -0.08900056, -0.        , -0.20911849,  0.09001607,
        0.44982385, -0.09945283,  0.        ,  1.18881924])
```

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

Out[22]:
```
28079065.7680432
```

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

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

Out[24]:
```
0.0750481944603354
```

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)))
```

```
38.368553069312625
1642.5200013611081
40.528015018763355
```

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]:
```
0.7360416559261741
```

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

Out[29]:
```
array([[9.99997826e-01, 7.75018057e-20, 2.17444273e-06]])
```

Random Forest

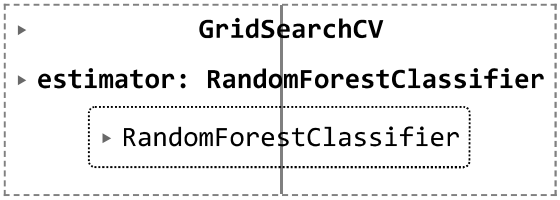
In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

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)
```

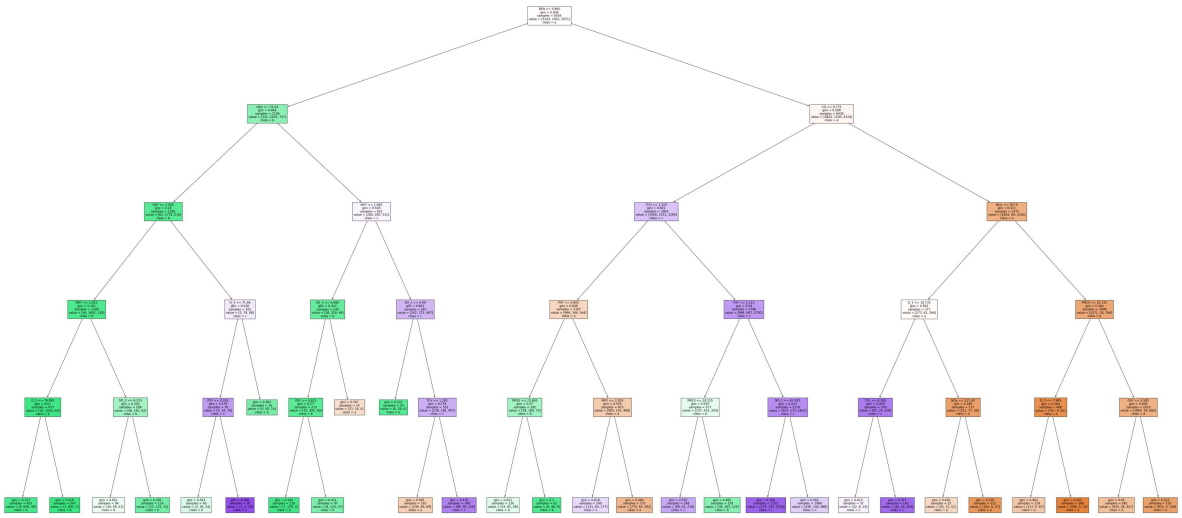
Out[32]:



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(0.4652777777777778, 0.9166666666666666, 'BEN <= 0.865\ngini = 0.656\nsa
mples = 8544\nvalue = [5143, 3363, 5071]\nclass = a'),
 Text(0.22685185185185186, 0.75, 'NOx <= 31.04\ngini = 0.484\nsamples = 2106
\nvalue = [341, 2263, 747]\nclass = b'),
 Text(0.1388888888888889, 0.5833333333333334, 'OXY <= 1.005\ngini = 0.24\nsam
ples = 1285\nvalue = [61, 1770, 216]\nclass = b'),
 Text(0.07407407407407407, 0.4166666666666667, 'MXY <= 1.015\ngini = 0.181\ns
amples = 1180\nvalue = [56, 1691, 128]\nclass = b'),
 Text(0.037037037037037035, 0.25, 'O_3 <= 76.895\ngini = 0.07\nsamples = 972
\nvalue = [10, 1500, 46]\nclass = b'),
 Text(0.018518518518518517, 0.08333333333333333, 'gini = 0.137\nsamples = 425
\nvalue = [9, 609, 39]\nclass = b'),
 Text(0.05555555555555555, 0.08333333333333333, 'gini = 0.018\nsamples = 547
\nvalue = [1, 891, 7]\nclass = b'),
 Text(0.1111111111111111, 0.25, 'SO_2 <= 6.215\ngini = 0.555\nsamples = 208\n
value = [46, 191, 82]\nclass = b'),
 Text(0.09259259259259259, 0.08333333333333333, 'gini = 0.651\nsamples = 94\n
value = [34, 59, 51]\nclass = b'),
 Text(0.12962962962962962, 0.08333333333333333, 'gini = 0.395\nsamples = 114
\nvalue = [12, 132, 31]\nclass = b'),
 Text(0.2037037037037037, 0.4166666666666667, 'O_3 <= 71.46\ngini = 0.526\nsa
mples = 105\nvalue = [5, 79, 88]\nclass = c'),
 Text(0.18518518518518517, 0.25, 'PXY <= 0.935\ngini = 0.479\nsamples = 70\nv
alue = [5, 34, 74]\nclass = c'),
 Text(0.16666666666666666, 0.08333333333333333, 'gini = 0.543\nsamples = 40\n
value = [3, 30, 24]\nclass = b'),
 Text(0.2037037037037037, 0.08333333333333333, 'gini = 0.196\nsamples = 30\nv
alue = [2, 4, 50]\nclass = c'),
 Text(0.2222222222222222, 0.25, 'gini = 0.362\nsamples = 35\nvalue = [0, 45,
14]\nclass = b'),
 Text(0.3148148148148148, 0.5833333333333334, 'MXY <= 1.065\ngini = 0.645\nsa
mples = 821\nvalue = [280, 493, 531]\nclass = c'),
 Text(0.2777777777777778, 0.4166666666666667, 'SO_2 <= 8.985\ngini = 0.347\ns
amples = 240\nvalue = [38, 318, 44]\nclass = b'),
 Text(0.25925925925925924, 0.25, 'OXY <= 0.625\ngini = 0.27\nsamples = 215\nv
alue = [15, 302, 40]\nclass = b'),
 Text(0.24074074074074073, 0.08333333333333333, 'gini = 0.102\nsamples = 118
\nvalue = [7, 178, 3]\nclass = b'),
 Text(0.2777777777777778, 0.08333333333333333, 'gini = 0.411\nsamples = 97\nv
alue = [8, 124, 37]\nclass = b'),
 Text(0.2962962962962963, 0.25, 'gini = 0.567\nsamples = 25\nvalue = [23, 16,
4]\nclass = a'),
 Text(0.35185185185185186, 0.4166666666666667, 'SO_2 <= 4.49\ngini = 0.601\ns
amples = 581\nvalue = [242, 175, 487]\nclass = c'),
 Text(0.3333333333333333, 0.25, 'gini = 0.231\nsamples = 29\nvalue = [6, 39,
0]\nclass = b'),
 Text(0.37037037037037035, 0.25, 'TCH <= 1.285\ngini = 0.578\nsamples = 552\n
value = [236, 136, 487]\nclass = c'),
 Text(0.35185185185185186, 0.08333333333333333, 'gini = 0.595\nsamples = 162
\nvalue = [138, 46, 69]\nclass = a'),
 Text(0.3888888888888889, 0.08333333333333333, 'gini = 0.476\nsamples = 390\n
value = [98, 90, 418]\nclass = c'),
 Text(0.7037037037037037, 0.75, 'CO <= 0.775\ngini = 0.589\nsamples = 6438\nv
alue = [4802, 1100, 4324]\nclass = a'),
 Text(0.5555555555555556, 0.5833333333333334, 'TCH <= 1.325\ngini = 0.601\nsa
mples = 3963\nvalue = [1958, 1031, 3294]\nclass = c'),
 Text(0.48148148148148145, 0.4166666666666667, 'PXY <= 0.805\ngini = 0.608\ns
```

```
amples = 1167\nvalue = [964, 344, 544]\nclass = a'),
 Text(0.4444444444444444, 0.25, 'PM10 <= 23.695\ngini = 0.57\nsamples = 192\n
value = [59, 189, 75]\nclass = b'),
 Text(0.42592592592592593, 0.08333333333333333, 'gini = 0.651\nsamples = 130
\nvalue = [53, 91, 69]\nclass = b'),
 Text(0.46296296296296297, 0.08333333333333333, 'gini = 0.2\nsamples = 62\nva
lue = [6, 98, 6]\nclass = b'),
 Text(0.5185185185185185, 0.25, 'MXY <= 2.505\ngini = 0.545\nsamples = 975\nv
alue = [905, 155, 469]\nclass = a'),
 Text(0.5, 0.08333333333333333, 'gini = 0.618\nsamples = 240\nvalue = [131, 6
2, 177]\nclass = c'),
 Text(0.5370370370370371, 0.08333333333333333, 'gini = 0.484\nsamples = 735\n
value = [774, 93, 292]\nclass = a'),
 Text(0.6296296296296297, 0.4166666666666667, 'PXY <= 1.215\ngini = 0.54\nsam
ples = 2796\nvalue = [994, 687, 2750]\nclass = c'),
 Text(0.5925925925925926, 0.25, 'PM10 <= 24.215\ngini = 0.615\nsamples = 577
\nvalue = [137, 415, 343]\nclass = b'),
 Text(0.5740740740740741, 0.08333333333333333, 'gini = 0.559\nsamples = 248\n
value = [99, 52, 218]\nclass = c'),
 Text(0.6111111111111112, 0.08333333333333333, 'gini = 0.462\nsamples = 329\n
value = [38, 363, 125]\nclass = b'),
 Text(0.6666666666666666, 0.25, 'NO_2 <= 62.925\ngini = 0.472\nsamples = 2219
\nvalue = [857, 272, 2407]\nclass = c'),
 Text(0.6481481481481481, 0.08333333333333333, 'gini = 0.326\nsamples = 1153
\nvalue = [227, 132, 1519]\nclass = c'),
 Text(0.6851851851851852, 0.08333333333333333, 'gini = 0.562\nsamples = 1066
\nvalue = [630, 140, 888]\nclass = c'),
 Text(0.8518518518518519, 0.5833333333333334, 'NOx <= 167.9\ngini = 0.411\nsa
mples = 2475\nvalue = [2844, 69, 1030]\nclass = a'),
 Text(0.7777777777777778, 0.4166666666666667, 'O_3 <= 19.735\ngini = 0.563\ns
amples = 377\nvalue = [273, 41, 266]\nclass = a'),
 Text(0.7407407407407407, 0.25, 'TOL <= 8.195\ngini = 0.428\nsamples = 200\nv
alue = [62, 24, 228]\nclass = c'),
 Text(0.7222222222222222, 0.08333333333333333, 'gini = 0.615\nsamples = 37\nv
alue = [22, 8, 24]\nclass = c'),
 Text(0.7592592592592593, 0.08333333333333333, 'gini = 0.357\nsamples = 163\n
value = [40, 16, 204]\nclass = c'),
 Text(0.8148148148148148, 0.25, 'NOx <= 127.45\ngini = 0.346\nsamples = 177\n
value = [211, 17, 38]\nclass = a'),
 Text(0.7962962962962963, 0.08333333333333333, 'gini = 0.636\nsamples = 25\nv
alue = [20, 11, 11]\nclass = a'),
 Text(0.8333333333333334, 0.08333333333333333, 'gini = 0.258\nsamples = 152\n
value = [191, 6, 27]\nclass = a'),
 Text(0.9259259259259259, 0.4166666666666667, 'PM10 <= 34.135\ngini = 0.364\n
samples = 2098\nvalue = [2571, 28, 764]\nclass = a'),
 Text(0.8888888888888888, 0.25, 'O_3 <= 7.945\ngini = 0.184\nsamples = 488\nv
alue = [707, 0, 81]\nclass = a'),
 Text(0.8703703703703703, 0.08333333333333333, 'gini = 0.463\nsamples = 119\n
value = [117, 0, 67]\nclass = a'),
 Text(0.9074074074074074, 0.08333333333333333, 'gini = 0.045\nsamples = 369\n
value = [590, 0, 14]\nclass = a'),
 Text(0.9629629629629629, 0.25, 'OXY <= 5.585\ngini = 0.406\nsamples = 1610\n
value = [1864, 28, 683]\nclass = a'),
 Text(0.9444444444444444, 0.08333333333333333, 'gini = 0.46\nsamples = 881\nv
alue = [933, 28, 447]\nclass = a'),
 Text(0.9814814814814815, 0.08333333333333333, 'gini = 0.323\nsamples = 729\n
value = [931, 0, 236]\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: 0.11799091137651152
Ridge Regression: 0.1160044793713958
Lasso Regression 0.06045121805425291
ElasticNet Regression: 0.0750481944603354
Logistic Regression: 0.7360416559261741
Random Forest: 0.7751342382600437
```

Logistic Is Better!!!