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_2009.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	P
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.260000	
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.580000	
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.190001	
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.530001	
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.760000	

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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

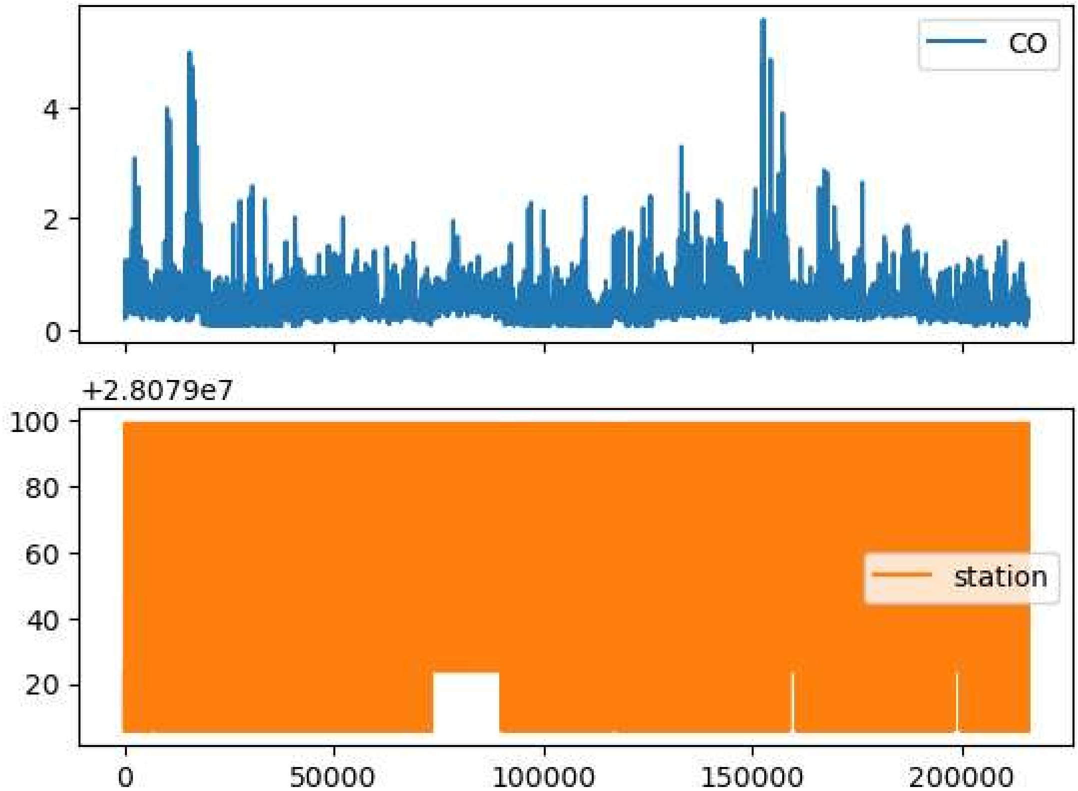
Out[7]:

	CO	station
3	0.33	28079006
20	0.32	28079024
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...	...	...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

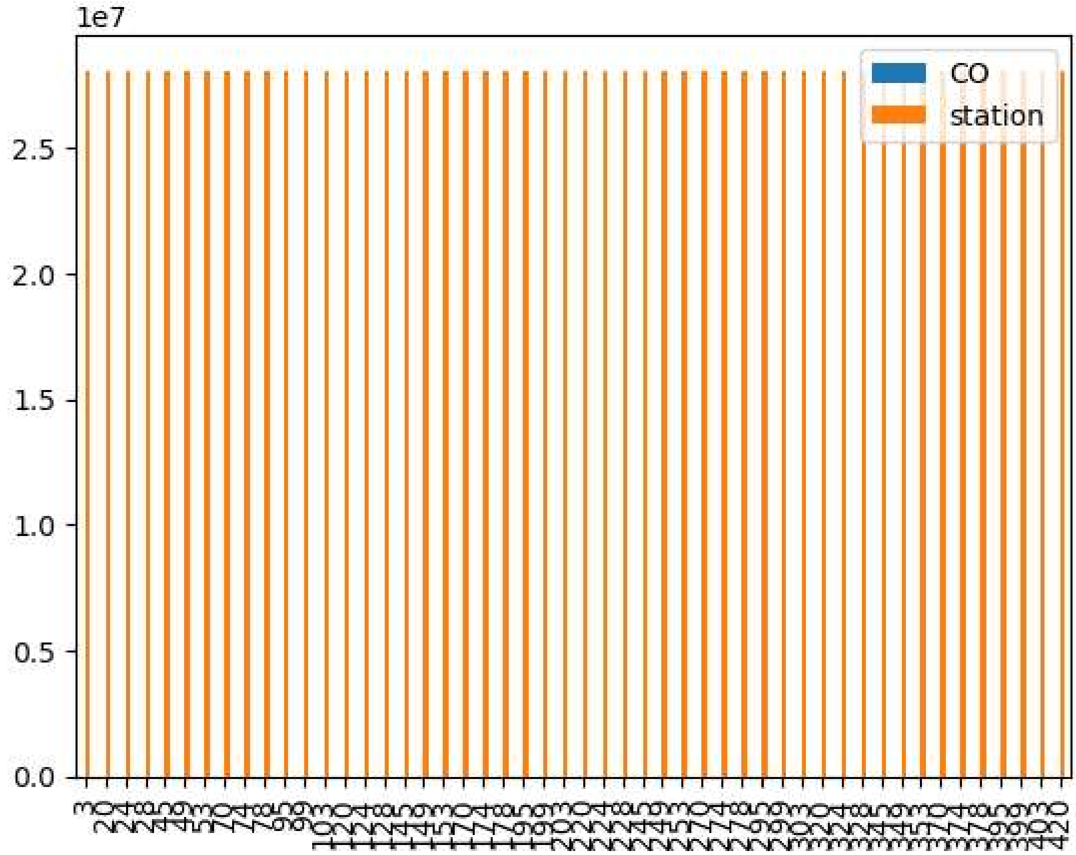
In [8]:
```
data.plot.line(subplots=True)
```

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



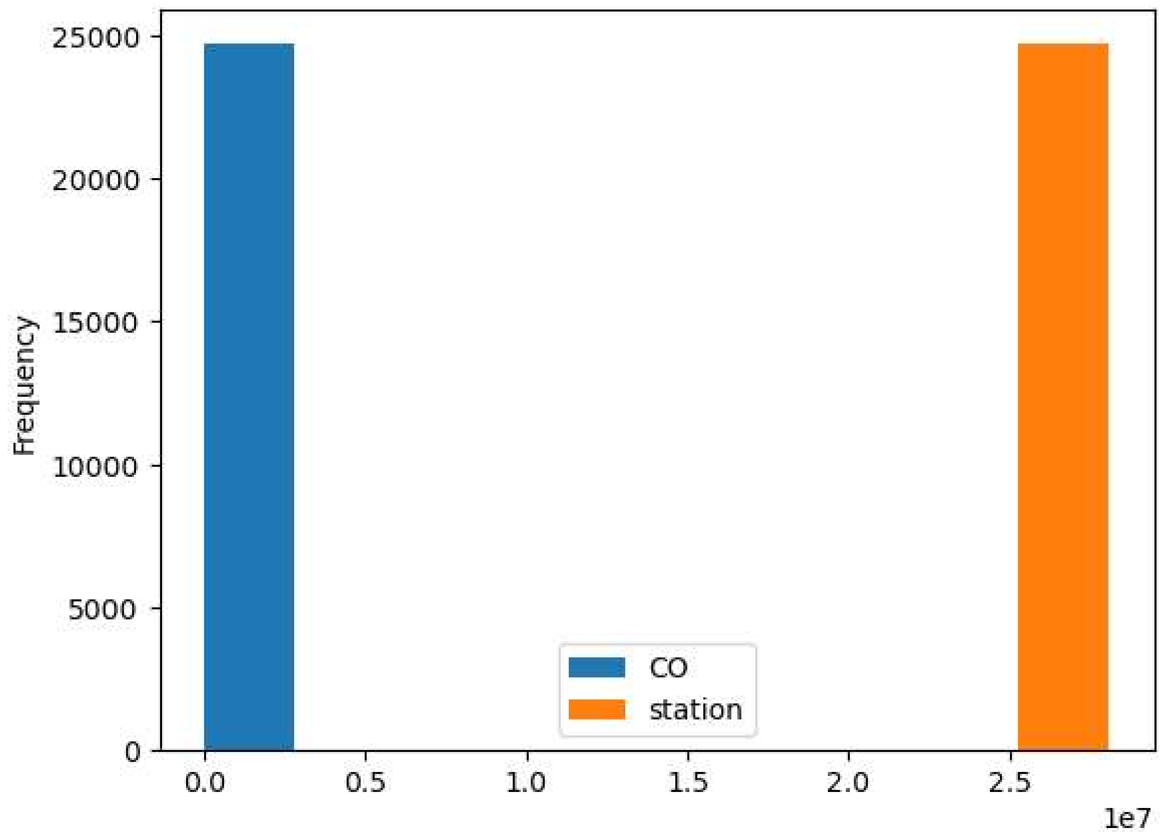
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



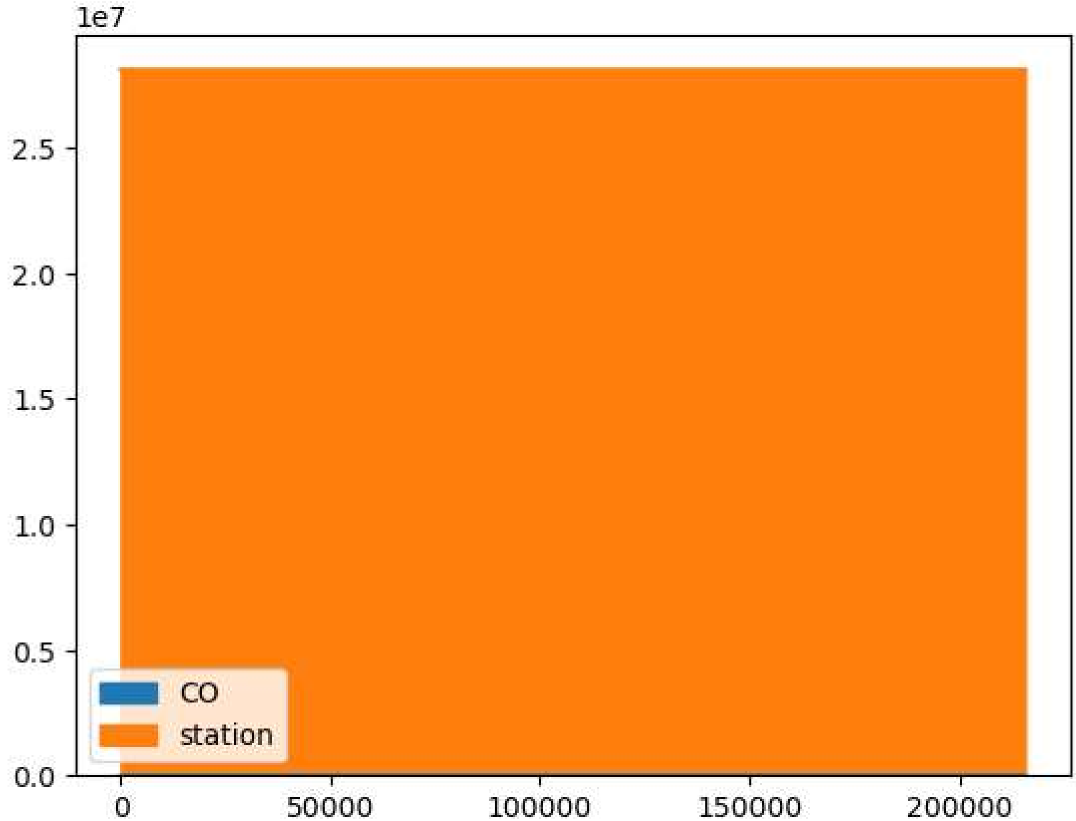
In [10]:
```
data.plot.hist()
```

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



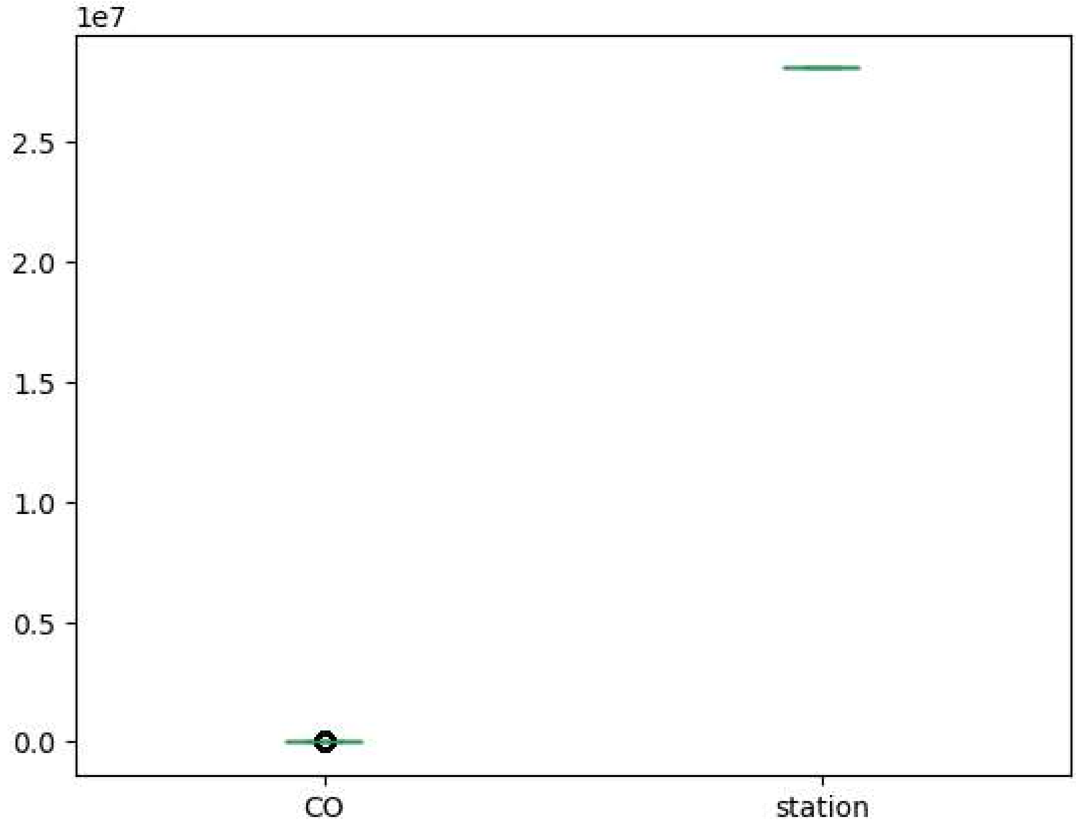
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



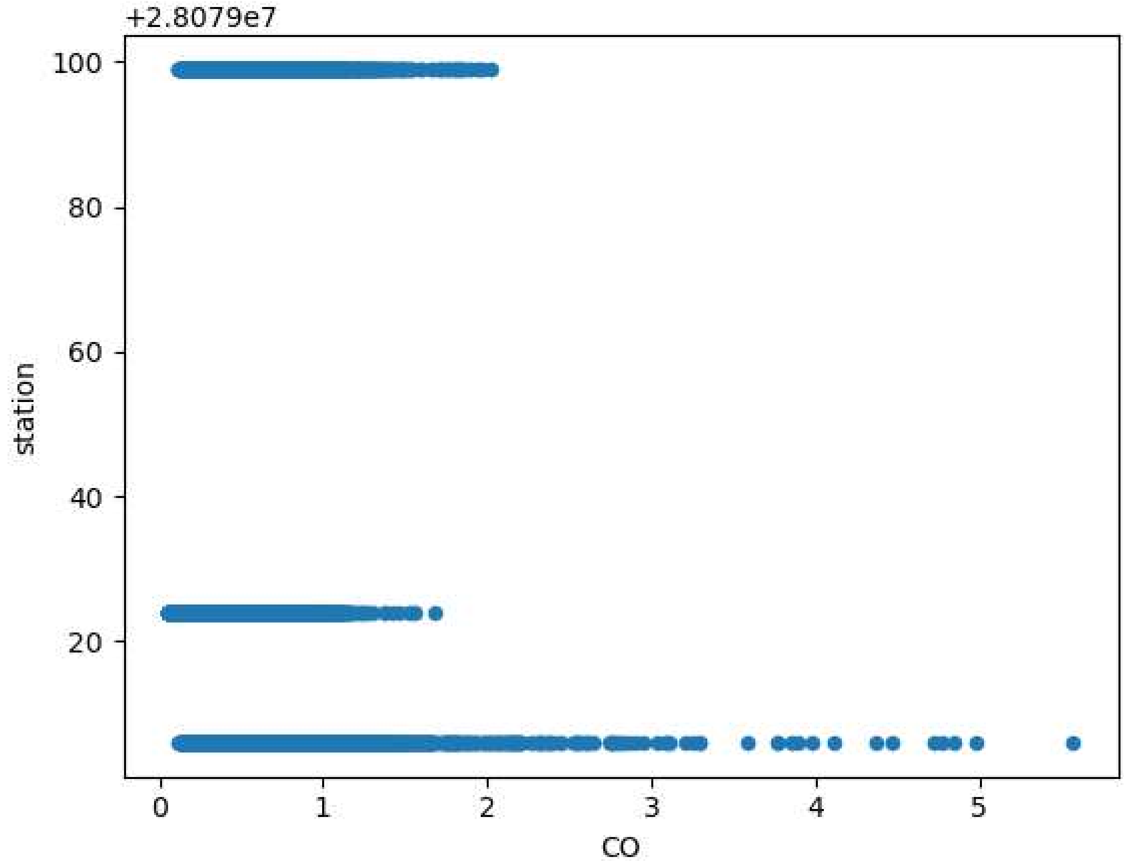
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



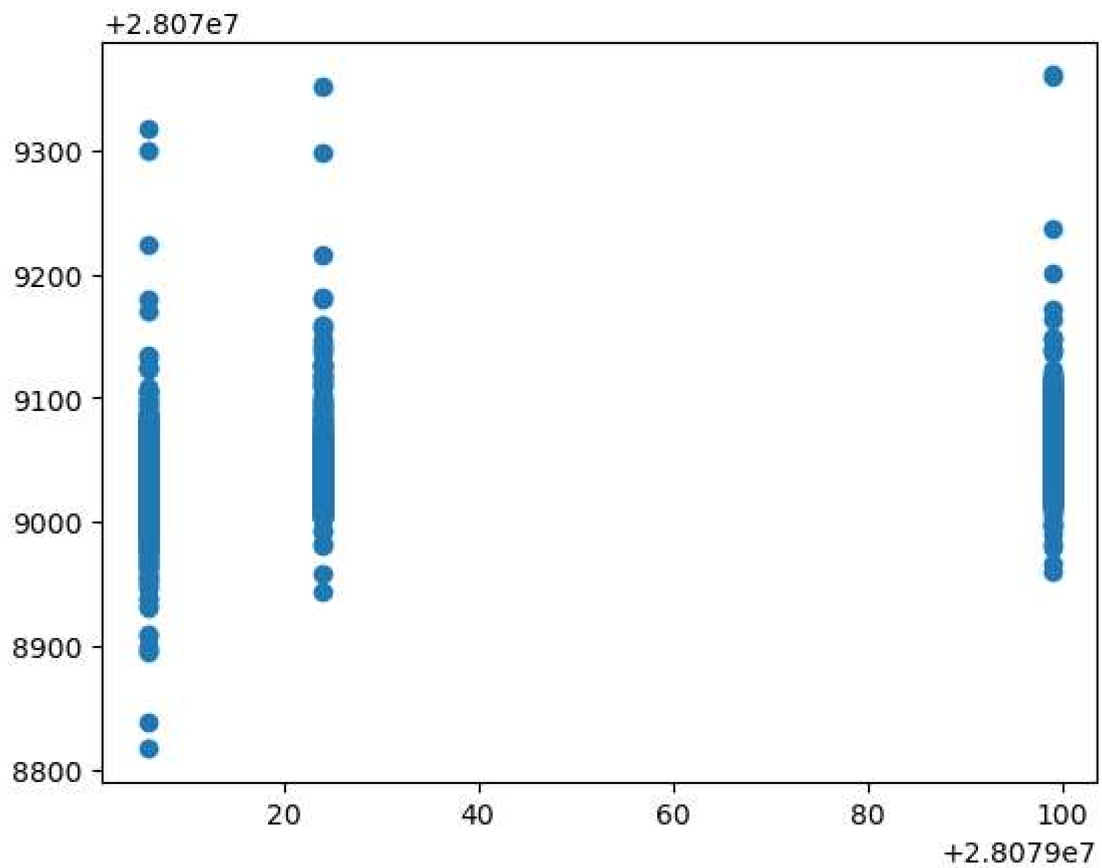
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1d8199f2d90>



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

```
0.26601739151949366
0.2933062499888983
```

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)
```

```
0.2690440139334591
0.29293537749908416
```

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

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

Out[19]:
```
0.03937735380051999
```

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([-6.87664085, -0.64461303,  0.40431261,  2.03148415,  0.        ,
       -0.22642231,  0.13089604,  1.18105579, -0.14403   ,  0.08682722,
        2.24311586, -0.79606947,  1.56173064, -2.0704175 ])
```

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

Out[22]:
```
28079063.582352076
```

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

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

Out[24]:
```
0.11075762320990701
```

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)))
```

```
35.69492881069716
1443.947038504372
37.99930313182561
```

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]:
```
0.8951733624630821
```

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

Out[29]:
```
array([[5.44721271e-13, 8.28694002e-44, 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]
}
```

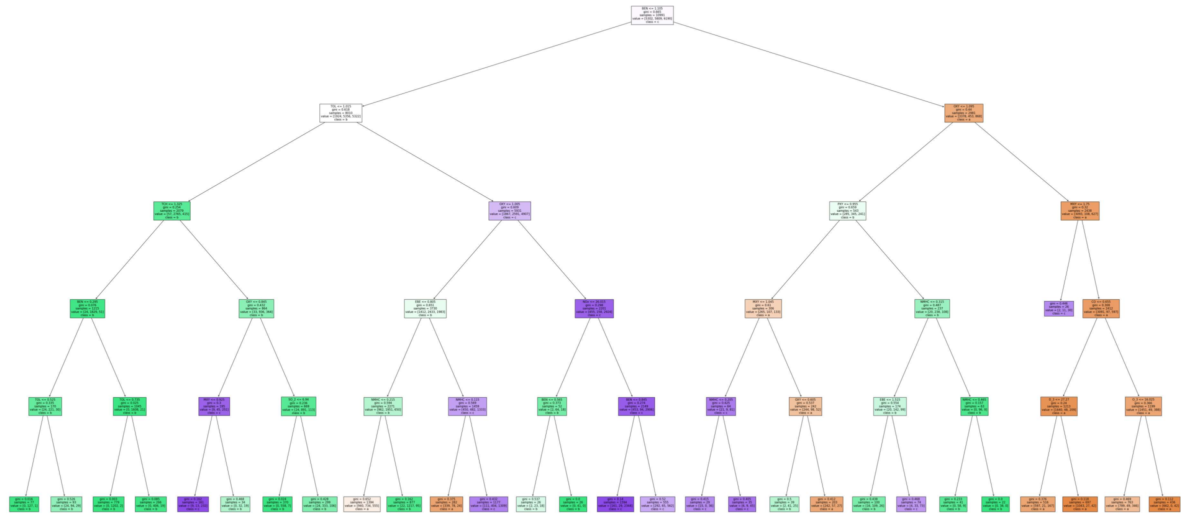
In [32]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

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(0.5491071428571429, 0.9166666666666666, 'BEN <= 1.105\ngini = 0.665\nsamples = 10991\nvalue = [5302, 5809, 6190]\nclass = c'),
 Text(0.2857142857142857, 0.75, 'TOL <= 1.015\ngini = 0.618\nsamples = 8010\nvalue = [1924, 5356, 5322]\nclass = b'),
 Text(0.14285714285714285, 0.5833333333333334, 'TCH <= 1.325\ngini = 0.254\nsamples = 2079\nvalue = [57, 2765, 415]\nclass = b'),
 Text(0.07142857142857142, 0.4166666666666667, 'BEN <= 0.295\ngini = 0.076\nsamples = 1215\nvalue = [24, 1829, 51]\nclass = b'),
 Text(0.03571428571428571, 0.25, 'TOL <= 0.525\ngini = 0.335\nsamples = 170\nvalue = [24, 221, 30]\nclass = b'),
 Text(0.017857142857142856, 0.08333333333333333, 'gini = 0.016\nsamples = 77\nvalue = [0, 127, 1]\nclass = b'),
 Text(0.05357142857142857, 0.08333333333333333, 'gini = 0.526\nsamples = 93\nvalue = [24, 94, 29]\nclass = b'),
 Text(0.10714285714285714, 0.25, 'TOL <= 0.735\ngini = 0.025\nsamples = 1045\nvalue = [0, 1608, 21]\nclass = b'),
 Text(0.08928571428571429, 0.08333333333333333, 'gini = 0.003\nsamples = 779\nvalue = [0, 1202, 2]\nclass = b'),
 Text(0.125, 0.08333333333333333, 'gini = 0.085\nsamples = 266\nvalue = [0, 406, 19]\nclass = b'),
 Text(0.21428571428571427, 0.4166666666666667, 'OXY <= 0.845\ngini = 0.432\nsamples = 864\nvalue = [33, 936, 364]\nclass = b'),
 Text(0.17857142857142858, 0.25, 'MXY <= 0.925\ngini = 0.3\nsamples = 195\nvalue = [9, 45, 251]\nclass = c'),
 Text(0.16071428571428573, 0.08333333333333333, 'gini = 0.162\nsamples = 161\nvalue = [9, 13, 232]\nclass = c'),
 Text(0.19642857142857142, 0.08333333333333333, 'gini = 0.468\nsamples = 34\nvalue = [0, 32, 19]\nclass = b'),
 Text(0.25, 0.25, 'SO_2 <= 6.94\ngini = 0.236\nsamples = 669\nvalue = [24, 891, 113]\nclass = b'),
 Text(0.23214285714285715, 0.08333333333333333, 'gini = 0.024\nsamples = 370\nvalue = [0, 558, 7]\nclass = b'),
 Text(0.26785714285714285, 0.08333333333333333, 'gini = 0.428\nsamples = 299\nvalue = [24, 333, 106]\nclass = b'),
 Text(0.42857142857142855, 0.5833333333333334, 'OXY <= 1.005\ngini = 0.609\nsamples = 5931\nvalue = [1867, 2591, 4907]\nclass = c'),
 Text(0.35714285714285715, 0.4166666666666667, 'EBE <= 0.805\ngini = 0.651\nsamples = 3730\nvalue = [1412, 2433, 1983]\nclass = b'),
 Text(0.32142857142857145, 0.25, 'NMHC <= 0.215\ngini = 0.594\nsamples = 2271\nvalue = [962, 1951, 650]\nclass = b'),
 Text(0.30357142857142855, 0.08333333333333333, 'gini = 0.652\nsamples = 1394\nvalue = [940, 734, 555]\nclass = a'),
 Text(0.3392857142857143, 0.08333333333333333, 'gini = 0.162\nsamples = 877\nvalue = [22, 1217, 95]\nclass = b'),
 Text(0.39285714285714285, 0.25, 'NMHC <= 0.115\ngini = 0.569\nsamples = 1459\nvalue = [450, 482, 1333]\nclass = c'),
 Text(0.375, 0.08333333333333333, 'gini = 0.375\nsamples = 282\nvalue = [339, 78, 24]\nclass = a'),
 Text(0.4107142857142857, 0.08333333333333333, 'gini = 0.432\nsamples = 1177\nvalue = [111, 404, 1309]\nclass = c'),
 Text(0.5, 0.4166666666666667, 'NOx <= 26.015\ngini = 0.298\nsamples = 2201\nvalue = [455, 158, 2924]\nclass = c'),
 Text(0.4642857142857143, 0.25, 'BEN <= 0.565\ngini = 0.373\nsamples = 52\nvalue = [2, 64, 18]\nclass = b'),
 Text(0.44642857142857145, 0.08333333333333333, 'gini = 0.537\nsamples = 26\nvalue = [2, 23, 18]\nclass = b'),
 Text(0.48214285714285715, 0.08333333333333333, 'gini = 0.0\nsamples = 26\nva
```

```
lue = [0, 41, 0]\nclass = b'),
 Text(0.5357142857142857, 0.25, 'BEN <= 0.845\ngini = 0.274\nsamples = 2149\n
value = [453, 94, 2906]\nclass = c'),
 Text(0.5178571428571429, 0.08333333333333333, 'gini = 0.14\nsamples = 1594\n
value = [161, 29, 2344]\nclass = c'),
 Text(0.5535714285714286, 0.08333333333333333, 'gini = 0.52\nsamples = 555\nv
alue = [292, 65, 562]\nclass = c'),
 Text(0.8125, 0.75, 'OXY <= 1.095\ngini = 0.44\nsamples = 2981\nvalue = [337
8, 453, 868]\nclass = a'),
 Text(0.7142857142857143, 0.5833333333333334, 'PXY <= 0.955\ngini = 0.659\nsa
mples = 543\nvalue = [285, 345, 241]\nclass = b'),
 Text(0.6428571428571429, 0.4166666666666667, 'MXY <= 1.045\ngini = 0.61\nsam
ples = 306\nvalue = [265, 107, 133]\nclass = a'),
 Text(0.6071428571428571, 0.25, 'NMHC <= 0.205\ngini = 0.425\nsamples = 64\nv
alue = [21, 9, 81]\nclass = c'),
 Text(0.5892857142857143, 0.08333333333333333, 'gini = 0.415\nsamples = 29\nv
alue = [15, 0, 36]\nclass = c'),
 Text(0.625, 0.08333333333333333, 'gini = 0.405\nsamples = 35\nvalue = [6, 9,
45]\nclass = c'),
 Text(0.6785714285714286, 0.25, 'OXY <= 0.605\ngini = 0.537\nsamples = 242\nv
alue = [244, 98, 52]\nclass = a'),
 Text(0.6607142857142857, 0.08333333333333333, 'gini = 0.5\nsamples = 39\nval
ue = [2, 41, 25]\nclass = b'),
 Text(0.6964285714285714, 0.08333333333333333, 'gini = 0.412\nsamples = 203\n
value = [242, 57, 27]\nclass = a'),
 Text(0.7857142857142857, 0.4166666666666667, 'NMHC <= 0.315\ngini = 0.487\ns
amples = 237\nvalue = [20, 238, 108]\nclass = b'),
 Text(0.75, 0.25, 'EBE <= 1.515\ngini = 0.554\nsamples = 174\nvalue = [20, 14
2, 99]\nclass = b'),
 Text(0.7321428571428571, 0.08333333333333333, 'gini = 0.438\nsamples = 100\n
value = [16, 109, 26]\nclass = b'),
 Text(0.7678571428571429, 0.08333333333333333, 'gini = 0.468\nsamples = 74\nv
alue = [4, 33, 73]\nclass = c'),
 Text(0.8214285714285714, 0.25, 'NMHC <= 0.465\ngini = 0.157\nsamples = 63\nv
alue = [0, 96, 9]\nclass = b'),
 Text(0.8035714285714286, 0.08333333333333333, 'gini = 0.233\nsamples = 41\nv
alue = [0, 58, 9]\nclass = b'),
 Text(0.8392857142857143, 0.08333333333333333, 'gini = 0.0\nsamples = 22\nval
ue = [0, 38, 0]\nclass = b'),
 Text(0.9107142857142857, 0.5833333333333334, 'MXY <= 1.75\ngini = 0.32\nsamp
les = 2438\nvalue = [3093, 108, 627]\nclass = a'),
 Text(0.8928571428571429, 0.4166666666666667, 'gini = 0.446\nsamples = 26\nva
lue = [2, 11, 30]\nclass = c'),
 Text(0.9285714285714286, 0.4166666666666667, 'CO <= 0.655\ngini = 0.308\nsam
ples = 2412\nvalue = [3091, 97, 597]\nclass = a'),
 Text(0.8928571428571429, 0.25, 'O_3 <= 27.27\ngini = 0.24\nsamples = 1213\nv
alue = [1640, 48, 209]\nclass = a'),
 Text(0.875, 0.08333333333333333, 'gini = 0.376\nsamples = 516\nvalue = [597,
21, 167]\nclass = a'),
 Text(0.9107142857142857, 0.08333333333333333, 'gini = 0.118\nsamples = 697\n
value = [1043, 27, 42]\nclass = a'),
 Text(0.9642857142857143, 0.25, 'O_3 <= 16.025\ngini = 0.366\nsamples = 1199
\nvalue = [1451, 49, 388]\nclass = a'),
 Text(0.9464285714285714, 0.08333333333333333, 'gini = 0.469\nsamples = 763\n
value = [789, 49, 346]\nclass = a'),
 Text(0.9821428571428571, 0.08333333333333333, 'gini = 0.112\nsamples = 436\n
value = [662, 0, 42]\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: 0.26601739151949366
Ridge Regression: 0.2690440139334591
Lasso Regression 0.03937735380051999
ElasticNet Regression: 0.11075762320990701
Logistic Regression: 0.8951733624630821
Random Forest: 0.90052581044124
```

Logistic Is Better!!!