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_2003.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209999
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389999
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240002
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839996
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779999

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', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

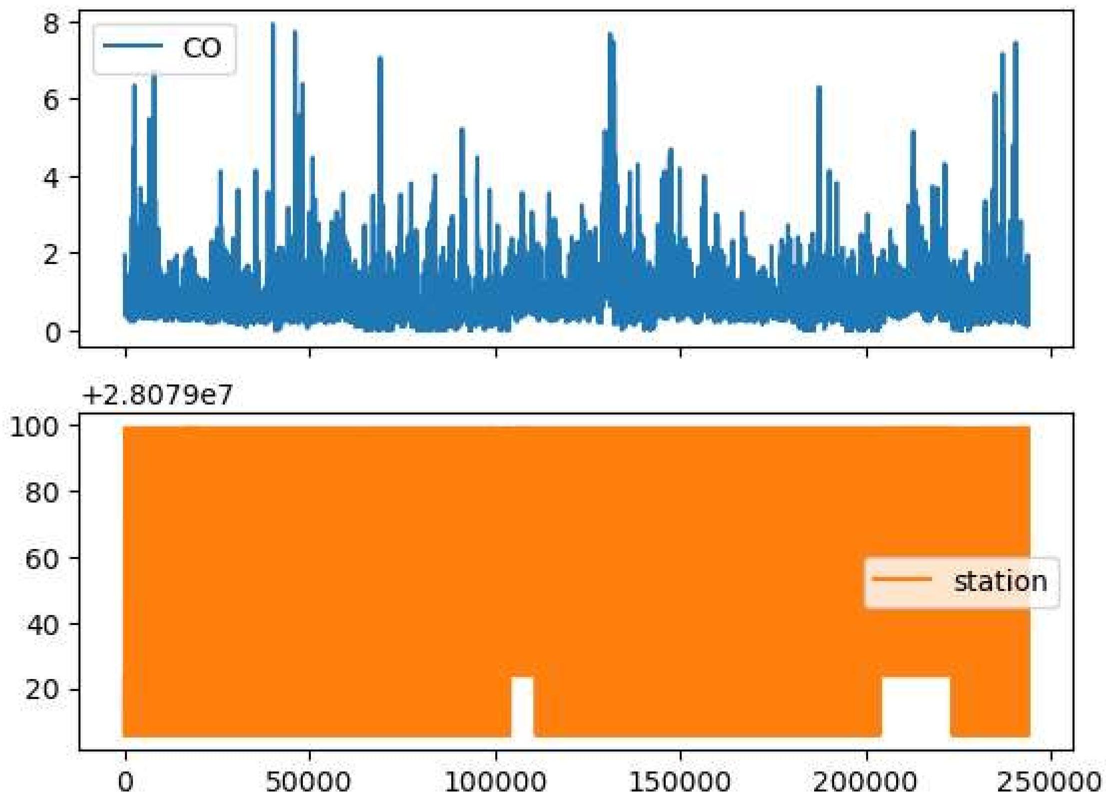
Out[7]:

	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

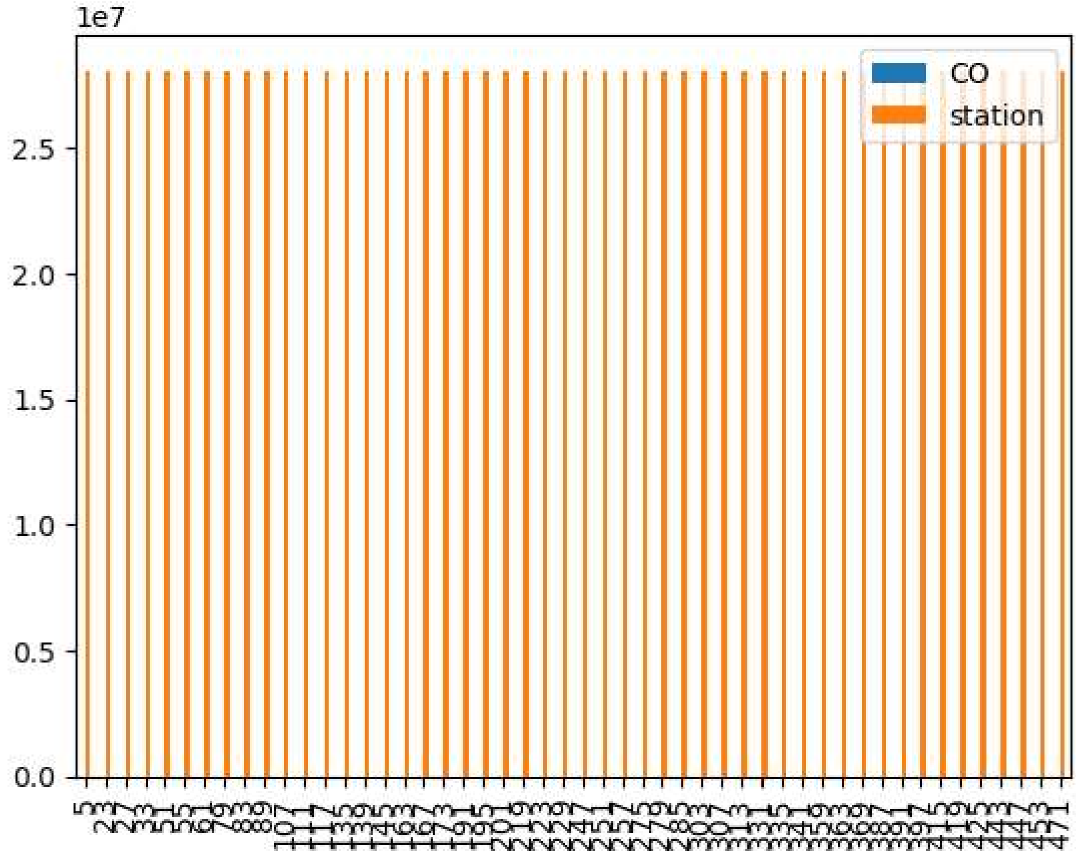
In [8]:
```
data.plot.line(subplots=True)
```

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



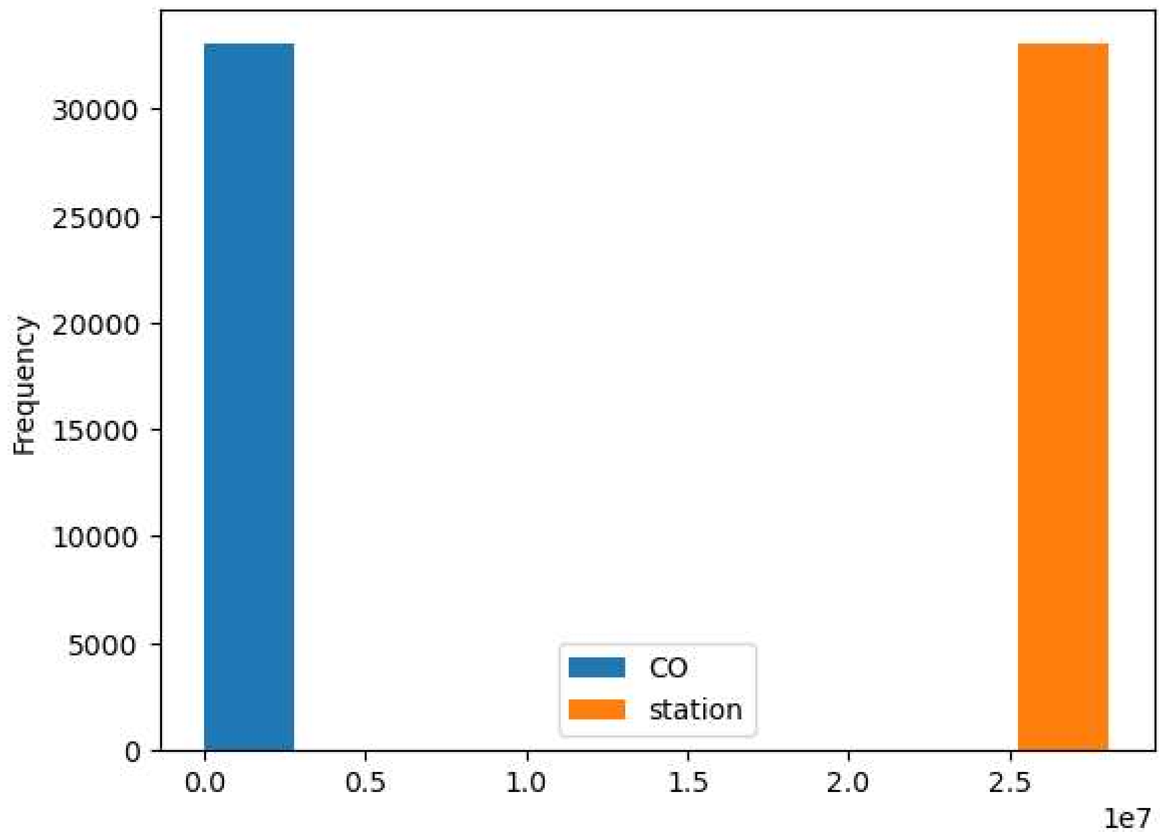
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



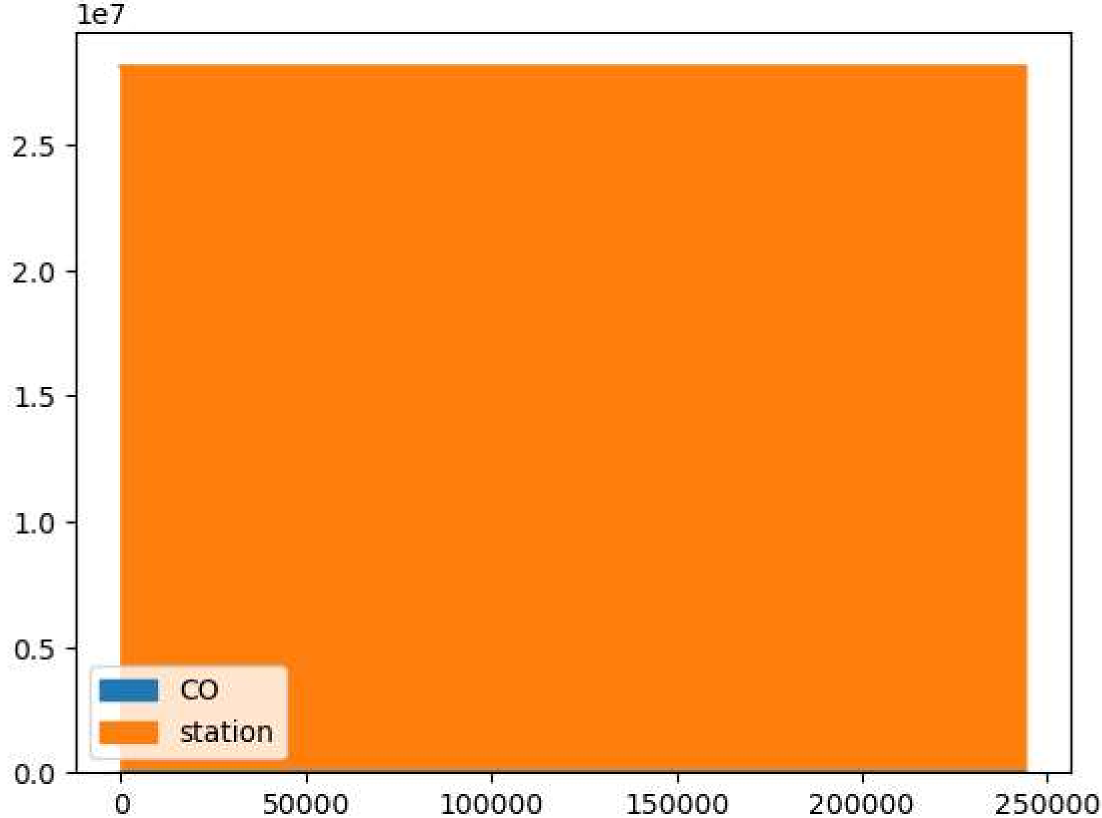
In [10]:
```
data.plot.hist()
```

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



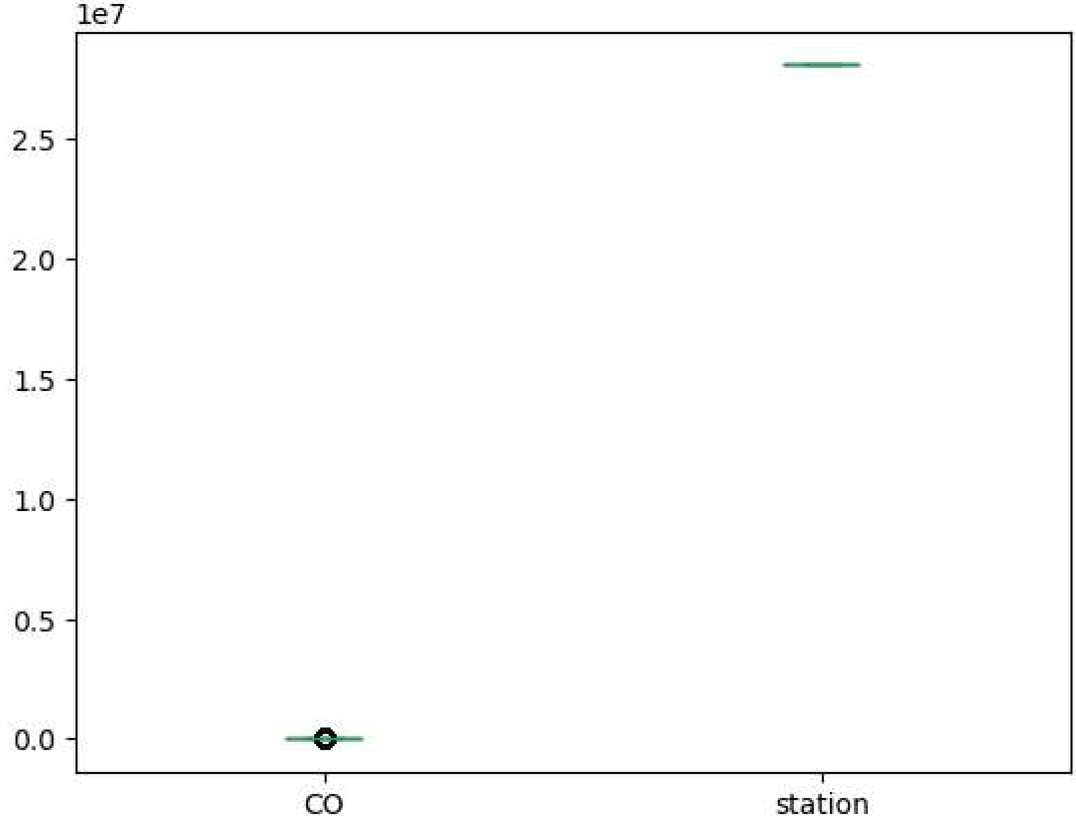
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



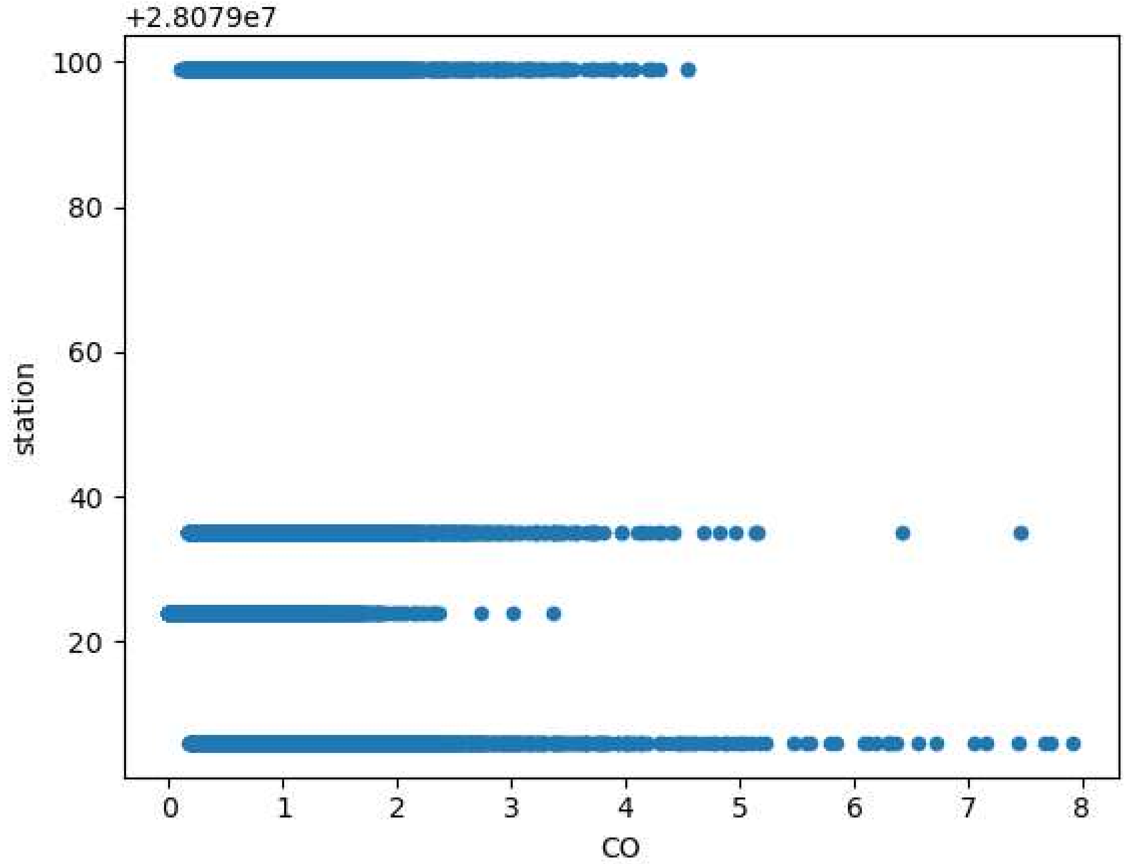
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



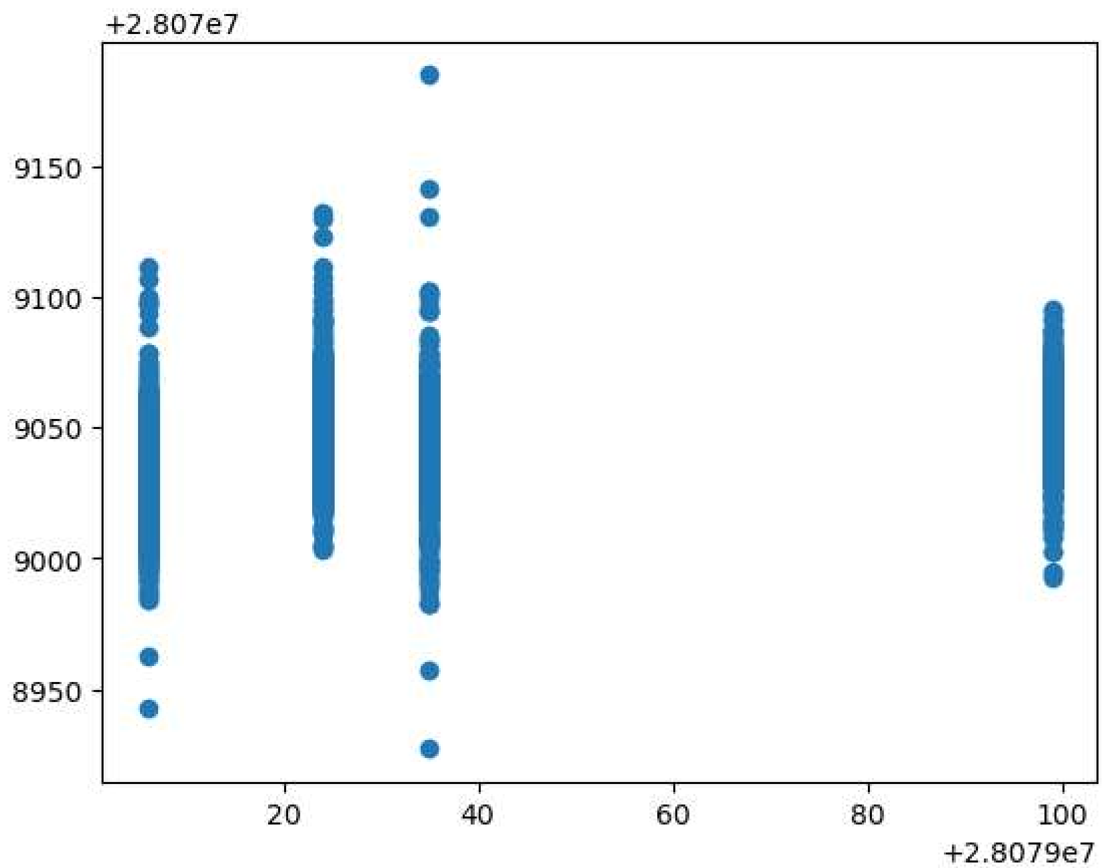
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x21507f72f90>



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

```
0.1701347321394755
0.17821539264824415
```

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)
```

```
0.16884583817917886
0.1771755777769093
```

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

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

Out[19]:
```
0.03300962238255689
```

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.        , -0.37014479,  0.00966089, -0.00265992,  0.13120338,
        0.15115898, -0.07269552, -1.38014302, -0.0343777 ,  0.07018145,
        0.31342131,  0.77048973,  1.57413284, -0.3931537 ])
```

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

Out[22]:
```
28079037.570191413
```

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

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

Out[24]:
```
0.04632926704706636
```

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)))
```

```
28.960766677698512
1167.4416002669948
34.167844536449685
```

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]:
```
0.7584974250227204
```

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

Out[29]:
```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457490e-16]])
```

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)
```

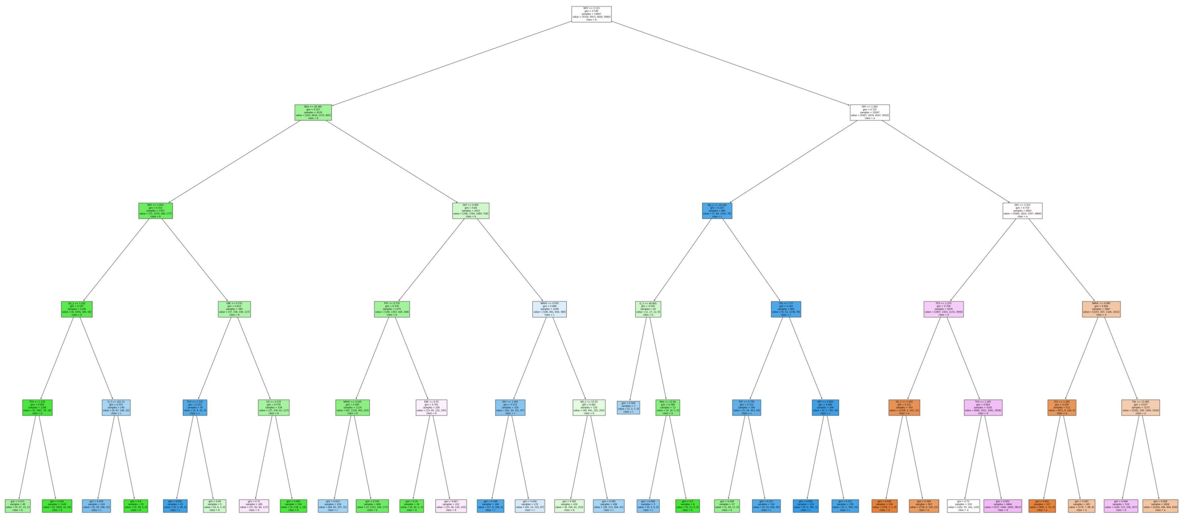
Out[32]:
```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

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(0.5020833333333333, 0.9166666666666666, 'MXY <= 2.115\ngini = 0.749\nsa
mples = 14667\nvalue = [5310, 5913, 5920, 5964]\nclass = d'),
 Text(0.26666666666666666, 0.75, 'NOx <= 28.385\ngini = 0.557\nsamples = 4120
\nvalue = [243, 4034, 1373, 905]\nclass = b'),
 Text(0.13333333333333333, 0.5833333333333334, 'MXY <= 1.015\ngini = 0.315\ns
amples = 1707\nvalue = [37, 2270, 289, 177]\nclass = b'),
 Text(0.06666666666666667, 0.4166666666666667, 'SO_2 <= 7.255\ngini = 0.197\n
samples = 1338\nvalue = [0, 1934, 185, 50]\nclass = b'),
 Text(0.03333333333333333, 0.25, 'TCH <= 1.235\ngini = 0.058\nsamples = 1189
\nvalue = [0, 1867, 39, 18]\nclass = b'),
 Text(0.016666666666666666, 0.08333333333333333, 'gini = 0.471\nsamples = 40
\nvalue = [0, 47, 23, 2]\nclass = b'),
 Text(0.05, 0.08333333333333333, 'gini = 0.034\nsamples = 1149\nvalue = [0, 1
820, 16, 16]\nclass = b'),
 Text(0.1, 0.25, 'O_3 <= 102.15\ngini = 0.553\nsamples = 149\nvalue = [0, 67,
146, 32]\nclass = c'),
 Text(0.08333333333333333, 0.08333333333333333, 'gini = 0.459\nsamples = 124
\nvalue = [0, 29, 146, 32]\nclass = c'),
 Text(0.11666666666666667, 0.08333333333333333, 'gini = 0.0\nsamples = 25\nva
lue = [0, 38, 0, 0]\nclass = b'),
 Text(0.2, 0.4166666666666667, 'EBE <= 0.535\ngini = 0.613\nsamples = 369\nva
lue = [37, 336, 104, 127]\nclass = b'),
 Text(0.16666666666666666, 0.25, 'TCH <= 1.315\ngini = 0.215\nsamples = 30\nv
alue = [0, 6, 43, 0]\nclass = c'),
 Text(0.15, 0.08333333333333333, 'gini = 0.091\nsamples = 25\nvalue = [0, 2,
40, 0]\nclass = c'),
 Text(0.18333333333333332, 0.08333333333333333, 'gini = 0.49\nsamples = 5\nva
lue = [0, 4, 3, 0]\nclass = b'),
 Text(0.23333333333333334, 0.25, 'CO <= 0.335\ngini = 0.578\nsamples = 339\nv
alue = [37, 330, 61, 127]\nclass = b'),
 Text(0.21666666666666667, 0.08333333333333333, 'gini = 0.71\nsamples = 185\n
value = [37, 92, 60, 117]\nclass = d'),
 Text(0.25, 0.08333333333333333, 'gini = 0.085\nsamples = 154\nvalue = [0, 23
8, 1, 10]\nclass = b'),
 Text(0.4, 0.5833333333333334, 'OXY <= 0.995\ngini = 0.66\nsamples = 2413\nva
lue = [206, 1764, 1084, 728]\nclass = b'),
 Text(0.3333333333333333, 0.4166666666666667, 'PXY <= 0.755\ngini = 0.576\nsa
mples = 1375\nvalue = [100, 1303, 428, 348]\nclass = b'),
 Text(0.3, 0.25, 'NMHC <= 0.045\ngini = 0.495\nsamples = 1119\nvalue = [67, 1
218, 303, 203]\nclass = b'),
 Text(0.2833333333333333, 0.08333333333333333, 'gini = 0.623\nsamples = 229\n
value = [64, 65, 197, 31]\nclass = c'),
 Text(0.31666666666666665, 0.08333333333333333, 'gini = 0.334\nsamples = 890
\nvalue = [3, 1153, 106, 172]\nclass = b'),
 Text(0.36666666666666664, 0.25, 'EBE <= 0.55\ngini = 0.701\nsamples = 256\nv
alue = [33, 85, 125, 145]\nclass = d'),
 Text(0.35, 0.08333333333333333, 'gini = 0.24\nsamples = 34\nvalue = [0, 45,
5, 2]\nclass = b'),
 Text(0.38333333333333336, 0.08333333333333333, 'gini = 0.667\nsamples = 222
\nvalue = [33, 40, 120, 143]\nclass = d'),
 Text(0.4666666666666667, 0.4166666666666667, 'NMHC <= 0.055\ngini = 0.689\ns
amples = 1038\nvalue = [106, 461, 656, 380]\nclass = c'),
 Text(0.43333333333333335, 0.25, 'OXY <= 1.005\ngini = 0.515\nsamples = 318\n
value = [62, 20, 331, 87]\nclass = c'),
 Text(0.4166666666666667, 0.08333333333333333, 'gini = 0.198\nsamples = 148\n
value = [17, 8, 206, 0]\nclass = c'),
 Text(0.45, 0.08333333333333333, 'gini = 0.649\nsamples = 170\nvalue = [45, 1
```

```
2, 125, 87]\nclass = c'),
 Text(0.5, 0.25, 'NO_2 <= 53.92\ngini = 0.681\nsamples = 720\nvalue = [44, 44
1, 325, 293]\nclass = b'),
 Text(0.48333333333333334, 0.08333333333333333, 'gini = 0.585\nsamples = 432
\nvalue = [8, 328, 61, 252]\nclass = b'),
 Text(0.5166666666666667, 0.08333333333333333, 'gini = 0.585\nsamples = 288\n
value = [36, 113, 264, 41]\nclass = c'),
 Text(0.7375, 0.75, 'OXY <= 1.005\ngini = 0.725\nsamples = 10547\nvalue = [50
67, 1879, 4547, 5059]\nclass = a'),
 Text(0.6083333333333333, 0.5833333333333334, 'NO_2 <= 24.295\ngini = 0.203\n
samples = 884\nvalue = [7, 69, 1250, 79]\nclass = c'),
 Text(0.55, 0.4166666666666667, 'O_3 <= 48.805\ngini = 0.545\nsamples = 20\nv
alue = [2, 17, 12, 0]\nclass = b'),
 Text(0.5333333333333333, 0.25, 'gini = 0.569\nsamples = 7\nvalue = [2, 3, 7,
0]\nclass = c'),
 Text(0.5666666666666667, 0.25, 'NOx <= 22.84\ngini = 0.388\nsamples = 13\nva
lue = [0, 14, 5, 0]\nclass = b'),
 Text(0.55, 0.08333333333333333, 'gini = 0.408\nsamples = 5\nvalue = [0, 2,
5, 0]\nclass = c'),
 Text(0.5833333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 8\nvalu
e = [0, 12, 0, 0]\nclass = b'),
 Text(0.6666666666666666, 0.4166666666666667, 'TOL <= 7.37\ngini = 0.183\nsam
ples = 864\nvalue = [5, 52, 1238, 79]\nclass = c'),
 Text(0.6333333333333333, 0.25, 'PXY <= 0.795\ngini = 0.314\nsamples = 356\nv
alue = [5, 50, 453, 45]\nclass = c'),
 Text(0.6166666666666667, 0.08333333333333333, 'gini = 0.438\nsamples = 37\nv
alue = [1, 40, 17, 0]\nclass = b'),
 Text(0.65, 0.08333333333333333, 'gini = 0.215\nsamples = 319\nvalue = [4, 1
0, 436, 45]\nclass = c'),
 Text(0.7, 0.25, 'OXY <= 0.805\ngini = 0.084\nsamples = 508\nvalue = [0, 2, 7
85, 34]\nclass = c'),
 Text(0.6833333333333333, 0.08333333333333333, 'gini = 0.005\nsamples = 232\n
value = [0, 0, 391, 1]\nclass = c'),
 Text(0.7166666666666667, 0.08333333333333333, 'gini = 0.151\nsamples = 276\n
value = [0, 2, 394, 33]\nclass = c'),
 Text(0.8666666666666667, 0.5833333333333334, 'OXY <= 3.315\ngini = 0.719\nsa
mples = 9663\nvalue = [5060, 1810, 3297, 4980]\nclass = a'),
 Text(0.8, 0.4166666666666667, 'TCH <= 1.255\ngini = 0.709\nsamples = 5976\nv
alue = [1807, 1503, 2133, 3958]\nclass = d'),
 Text(0.7666666666666667, 0.25, 'SO_2 <= 5.265\ngini = 0.231\nsamples = 793\n
value = [1108, 2, 142, 22]\nclass = a'),
 Text(0.75, 0.08333333333333333, 'gini = 0.026\nsamples = 226\nvalue = [378,
2, 3, 0]\nclass = a'),
 Text(0.7833333333333333, 0.08333333333333333, 'gini = 0.304\nsamples = 567\n
value = [730, 0, 139, 22]\nclass = a'),
 Text(0.8333333333333334, 0.25, 'TCH <= 1.285\ngini = 0.664\nsamples = 5183\n
value = [699, 1501, 1991, 3936]\nclass = d'),
 Text(0.8166666666666667, 0.08333333333333333, 'gini = 0.72\nsamples = 319\nv
alue = [162, 55, 161, 124]\nclass = a'),
 Text(0.85, 0.08333333333333333, 'gini = 0.652\nsamples = 4864\nvalue = [537,
1446, 1830, 3812]\nclass = d'),
 Text(0.9333333333333333, 0.4166666666666667, 'NMHC <= 0.085\ngini = 0.604\ns
amples = 3687\nvalue = [3253, 307, 1164, 1022]\nclass = a'),
 Text(0.9, 0.25, 'TCH <= 1.285\ngini = 0.259\nsamples = 512\nvalue = [671, 8,
104, 6]\nclass = a'),
 Text(0.8833333333333333, 0.08333333333333333, 'gini = 0.061\nsamples = 327\n
value = [492, 1, 15, 0]\nclass = a'),
```

```
 Text(0.9166666666666666, 0.08333333333333333, 'gini = 0.493\nsamples = 185\n
value = [179, 7, 89, 6]\nclass = a'),
 Text(0.9666666666666667, 0.25, 'TOL <= 13.485\ngini = 0.637\nsamples = 3175
\nvalue = [2582, 299, 1060, 1016]\nclass = a'),
 Text(0.95, 0.08333333333333333, 'gini = 0.694\nsamples = 519\nvalue = [164,
115, 156, 357]\nclass = d'),
 Text(0.9833333333333333, 0.08333333333333333, 'gini = 0.589\nsamples = 2656
\nvalue = [2418, 184, 904, 659]\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: 0.17013473213947555
Ridge Regression: 0.16884583817917886
Lasso Regression 0.03300962238255689
ElasticNet Regression: 0.04632926704706636
Logistic Regression: 0.7584974250227204
Random Forest: 0.7301681838070575
```

Logistic Is Better!!!