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_2014.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
0	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	3.0	10.0	NaN	NaN	NaN	3.0	NaN	NaN	2807900
1	2014-06-01 01:00:00	0.2	0.2	0.1	0.11	3.0	17.0	68.0	10.0	5.0	5.0	1.36	1.3	2807900
2	2014-06-01 01:00:00	0.3	NaN	0.1	NaN	2.0	6.0	NaN	NaN	NaN	NaN	NaN	1.1	2807901
3	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	79.0	NaN	NaN	NaN	NaN	NaN	2807901
4	2014-06-01 01:00:00	NaN	NaN	NaN	NaN	1.0	6.0	75.0	NaN	NaN	4.0	NaN	NaN	2807901

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

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

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ MB
```

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

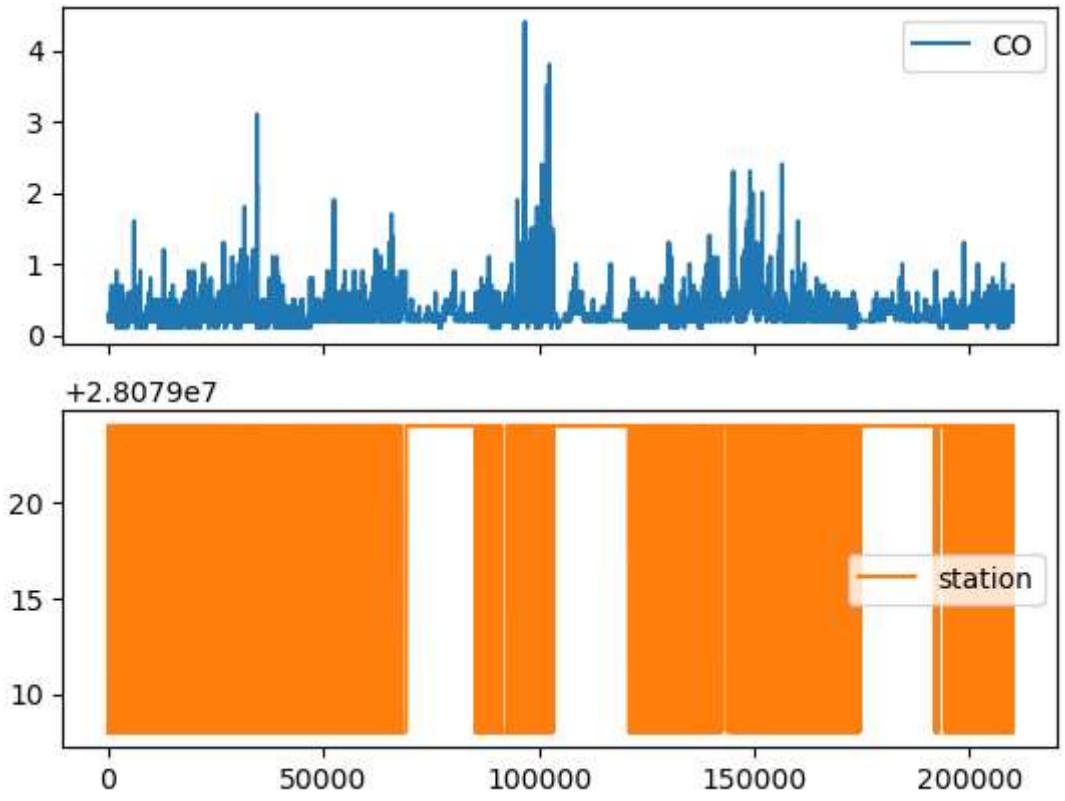
Out[7]:

	CO	station
1	0.2	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
209958	0.2	28079024
209977	0.7	28079008
209982	0.2	28079024
210001	0.4	28079008
210006	0.2	28079024

13946 rows × 2 columns

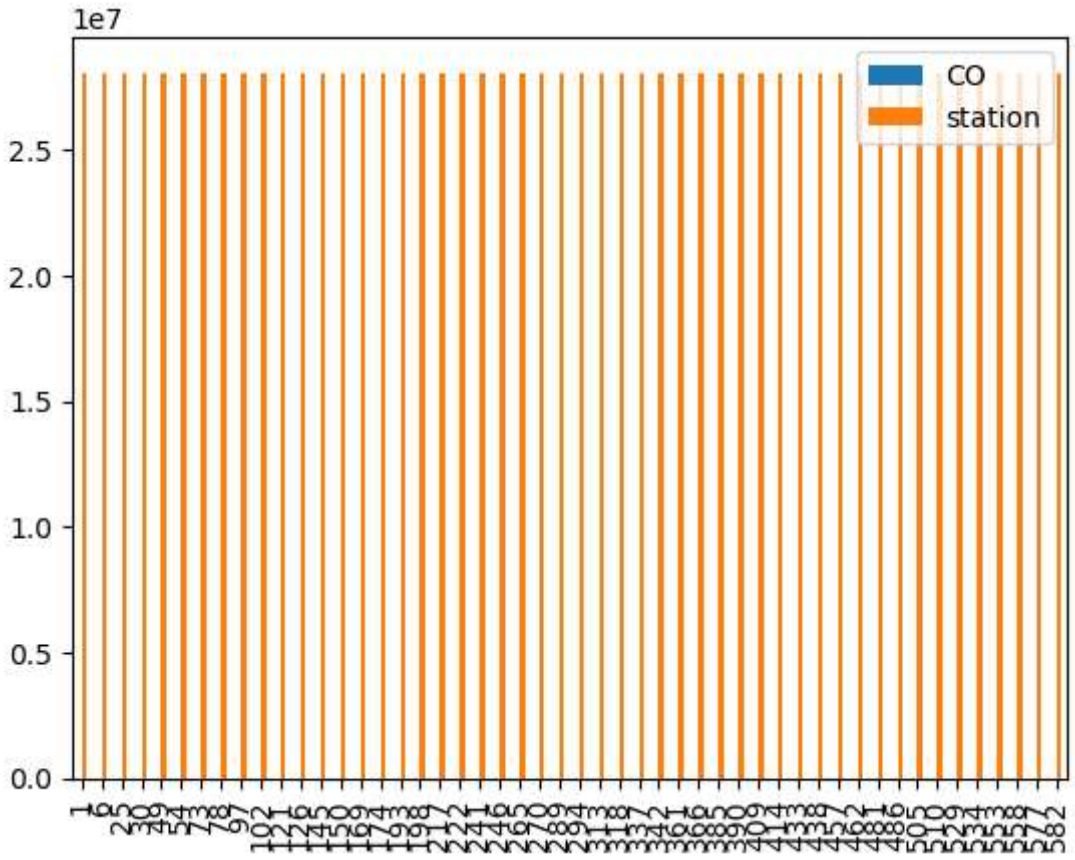
In [8]:
```
data.plot.line(subplots=True)
```

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



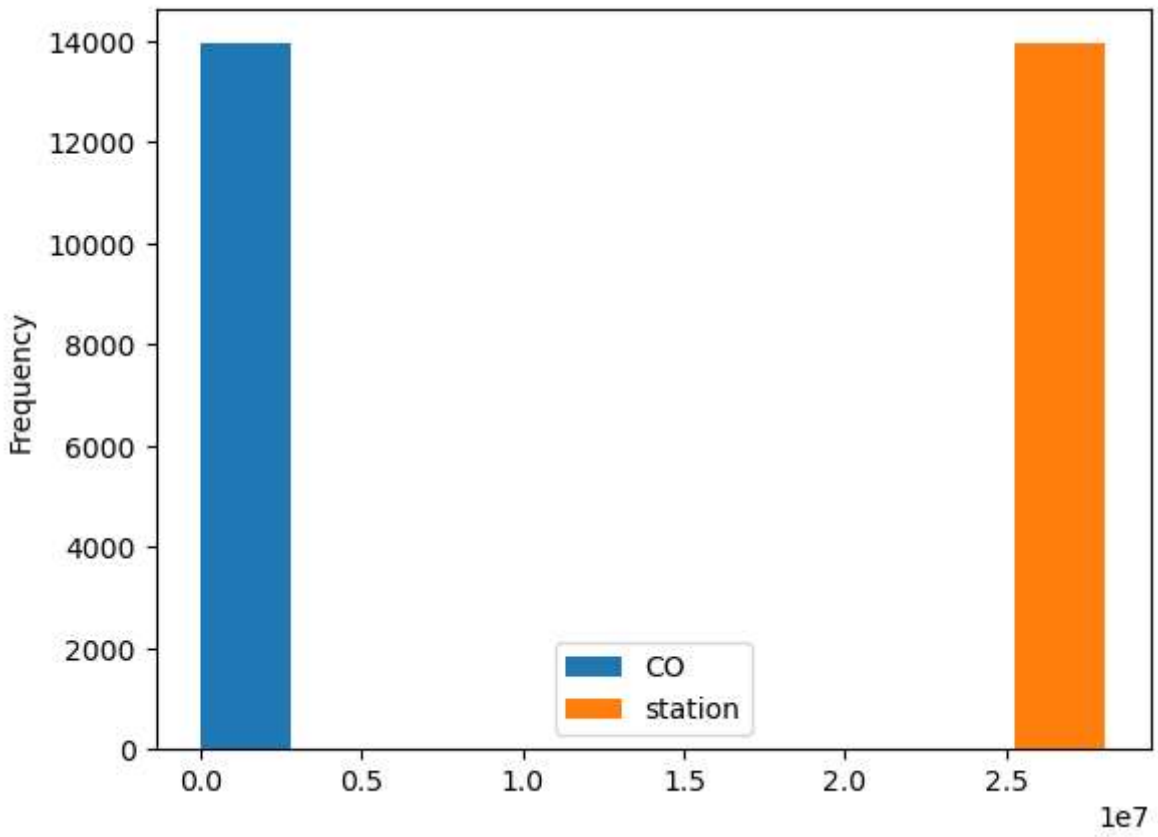
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



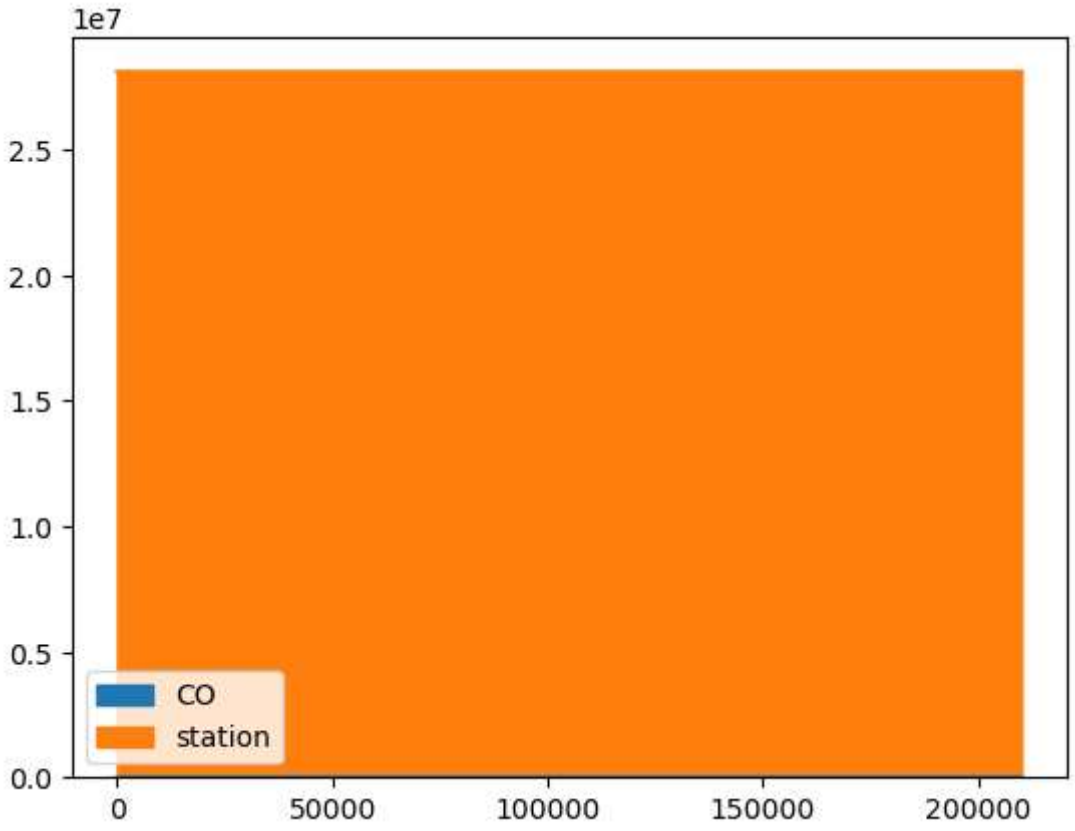
In [10]:
```
data.plot.hist()
```

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



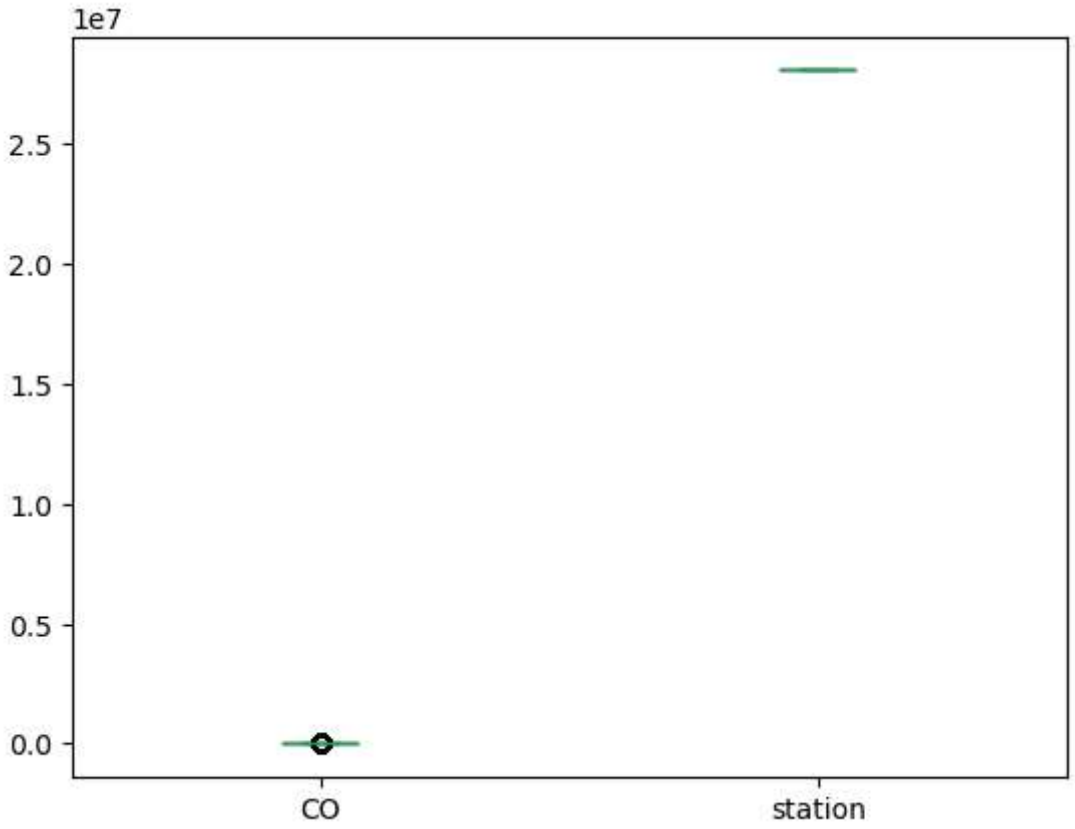
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



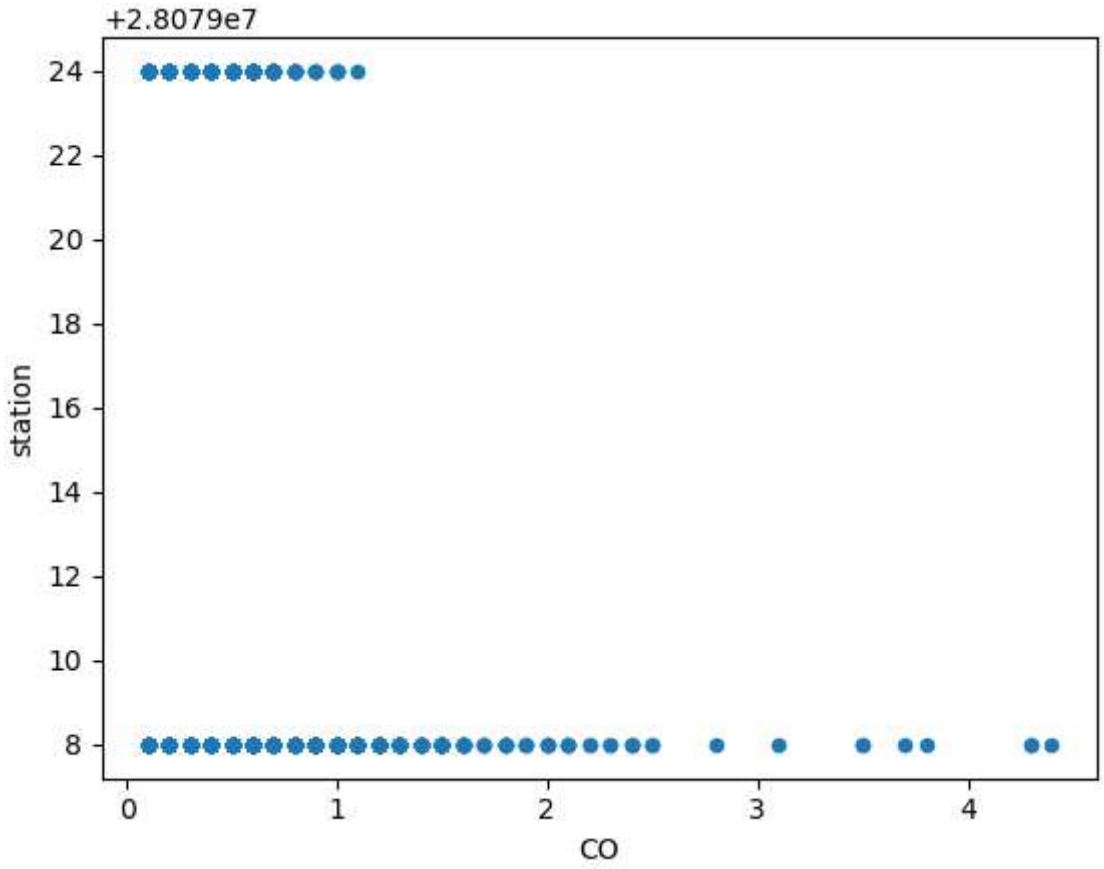
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



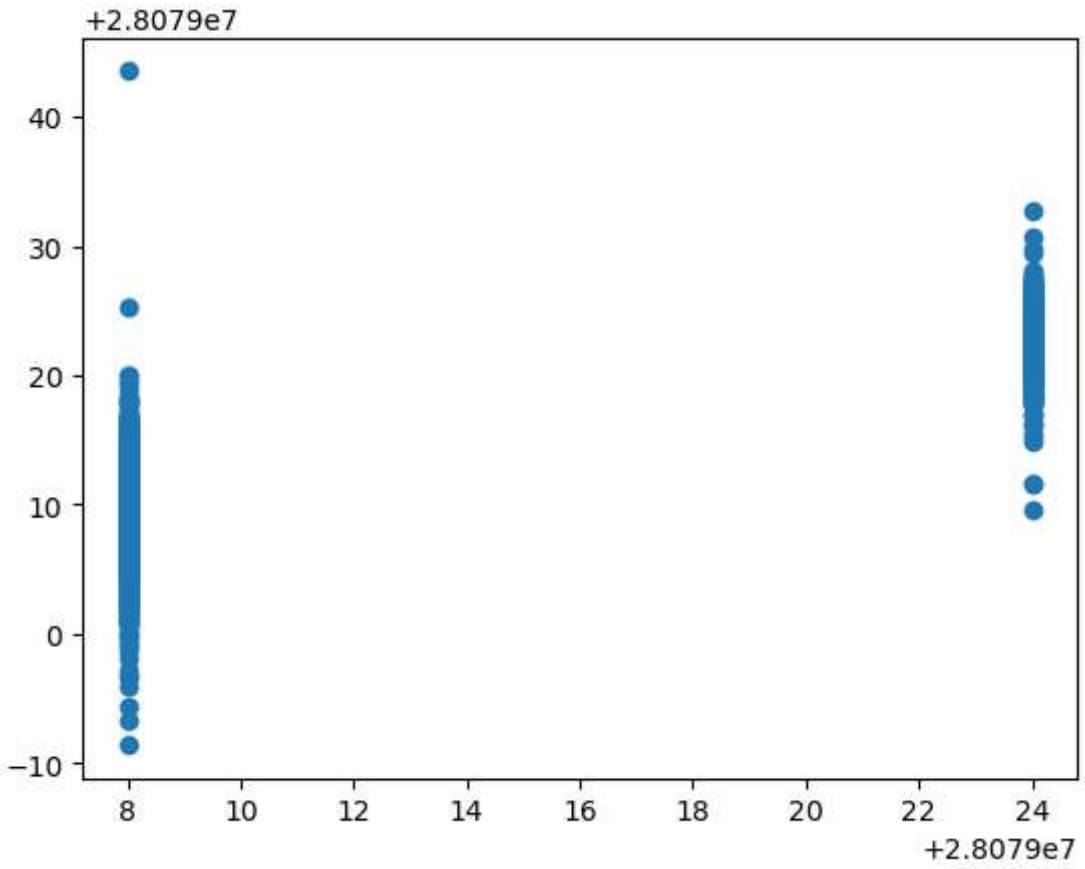
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x288f233ae50>



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

```
0.8898266972443416
0.8918220578414895
```

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)
```

```
0.8656692723177617
0.8694196840406931
```

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

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

Out[19]:
```
0.2797650321514594
```

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.        ,  0.        ,  0.        , -0.10810085,
        0.10430448, -0.01691592, -0.0172753 ,  0.12021387, -1.41015963,
        0.        , -0.56712908])
```

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

Out[22]:
```
28079026.615376983
```

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

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

Out[24]:
```
0.5926559921582942
```

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)))
```

```
4.167310216467664
25.68872336515694
5.06840442004749
```

Logistics Regression

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

In [27]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NO', 'O_3',
'PM10','PM25','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)
```

Out[28]:
```
LogisticRegression(max_iter=10000)
```

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

Out[29]:
```
array([[1.00000000e+00, 1.64445593e-21]])
```

Random Forest

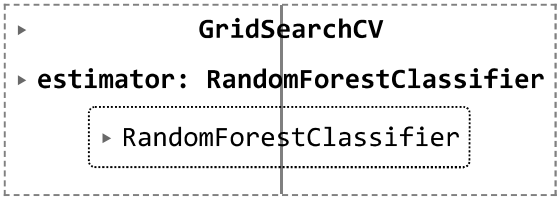
In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[30]:
```
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)
```

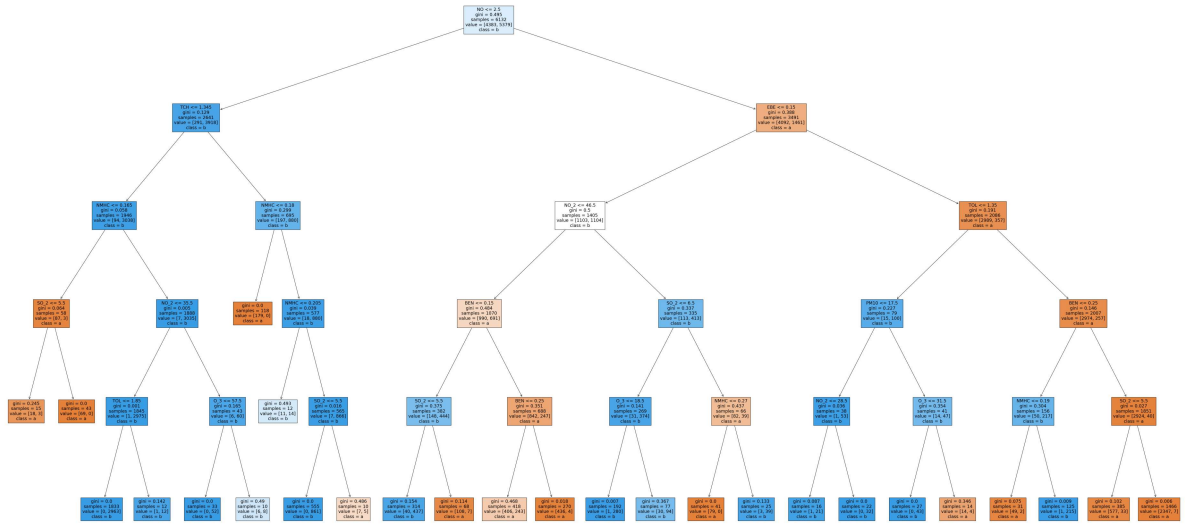
Out[32]:



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(0.4122340425531915, 0.9166666666666666, 'NO <= 2.5\ngini = 0.495\nsampl
es = 6132\nvalue = [4383, 5379]\nclass = b'),
 Text(0.16489361702127658, 0.75, 'TCH <= 1.345\ngini = 0.129\nsamples = 2641
\nvalue = [291, 3918]\nclass = b'),
 Text(0.09574468085106383, 0.5833333333333334, 'NMHC <= 0.165\ngini = 0.058\n
samples = 1946\nvalue = [94, 3038]\nclass = b'),
 Text(0.0425531914893617, 0.4166666666666667, 'SO_2 <= 5.5\ngini = 0.064\nsam
ples = 58\nvalue = [87, 3]\nclass = a'),
 Text(0.02127659574468085, 0.25, 'gini = 0.245\nsamples = 15\nvalue = [18, 3]
\nclass = a'),
 Text(0.06382978723404255, 0.25, 'gini = 0.0\nsamples = 43\nvalue = [69, 0]\n
class = a'),
 Text(0.14893617021276595, 0.4166666666666667, 'NO_2 <= 35.5\ngini = 0.005\ns
amples = 1888\nvalue = [7, 3035]\nclass = b'),
 Text(0.10638297872340426, 0.25, 'TOL <= 1.85\ngini = 0.001\nsamples = 1845\n
value = [1, 2975]\nclass = b'),
 Text(0.0851063829787234, 0.08333333333333333, 'gini = 0.0\nsamples = 1833\nv
alue = [0, 2963]\nclass = b'),
 Text(0.1276595744680851, 0.08333333333333333, 'gini = 0.142\nsamples = 12\nv
alue = [1, 12]\nclass = b'),
 Text(0.19148936170212766, 0.25, 'O_3 <= 57.5\ngini = 0.165\nsamples = 43\nva
lue = [6, 60]\nclass = b'),
 Text(0.1702127659574468, 0.08333333333333333, 'gini = 0.0\nsamples = 33\nval
ue = [0, 52]\nclass = b'),
 Text(0.2127659574468085, 0.08333333333333333, 'gini = 0.49\nsamples = 10\nva
lue = [6, 8]\nclass = b'),
 Text(0.23404255319148937, 0.5833333333333334, 'NMHC <= 0.18\ngini = 0.299\ns
amples = 695\nvalue = [197, 880]\nclass = b'),
 Text(0.2127659574468085, 0.4166666666666667, 'gini = 0.0\nsamples = 118\nval
ue = [179, 0]\nclass = a'),
 Text(0.2553191489361702, 0.4166666666666667, 'NMHC <= 0.205\ngini = 0.039\ns
amples = 577\nvalue = [18, 880]\nclass = b'),
 Text(0.23404255319148937, 0.25, 'gini = 0.493\nsamples = 12\nvalue = [11, 1
4]\nclass = b'),
 Text(0.2765957446808511, 0.25, 'SO_2 <= 5.5\ngini = 0.016\nsamples = 565\nva
lue = [7, 866]\nclass = b'),
 Text(0.2553191489361702, 0.08333333333333333, 'gini = 0.0\nsamples = 555\nva
lue = [0, 861]\nclass = b'),
 Text(0.2978723404255319, 0.08333333333333333, 'gini = 0.486\nsamples = 10\nv
alue = [7, 5]\nclass = a'),
 Text(0.6595744680851063, 0.75, 'EBE <= 0.15\ngini = 0.388\nsamples = 3491\nv
alue = [4092, 1461]\nclass = a'),
 Text(0.48936170212765956, 0.5833333333333334, 'NO_2 <= 46.5\ngini = 0.5\nsam
ples = 1405\nvalue = [1103, 1104]\nclass = b'),
 Text(0.40425531914893614, 0.4166666666666667, 'BEN <= 0.15\ngini = 0.484\nsa
mples = 1070\nvalue = [990, 691]\nclass = a'),
 Text(0.3617021276595745, 0.25, 'SO_2 <= 5.5\ngini = 0.375\nsamples = 382\nva
lue = [148, 444]\nclass = b'),
 Text(0.3404255319148936, 0.08333333333333333, 'gini = 0.154\nsamples = 314\n
value = [40, 437]\nclass = b'),
 Text(0.3829787234042553, 0.08333333333333333, 'gini = 0.114\nsamples = 68\nv
alue = [108, 7]\nclass = a'),
 Text(0.44680851063829785, 0.25, 'BEN <= 0.25\ngini = 0.351\nsamples = 688\nv
alue = [842, 247]\nclass = a'),
 Text(0.425531914893617, 0.08333333333333333, 'gini = 0.468\nsamples = 418\nv
alue = [406, 243]\nclass = a'),
 Text(0.46808510638297873, 0.08333333333333333, 'gini = 0.018\nsamples = 270
```

```
\nvalue = [436, 4]\nclass = a'),
 Text(0.574468085106383, 0.4166666666666667, 'SO_2 <= 6.5\ngini = 0.337\nsamp
les = 335\nvalue = [113, 413]\nclass = b'),
 Text(0.5319148936170213, 0.25, 'O_3 <= 18.5\ngini = 0.141\nsamples = 269\nva
lue = [31, 374]\nclass = b'),
 Text(0.5106382978723404, 0.08333333333333333, 'gini = 0.007\nsamples = 192\n
value = [1, 280]\nclass = b'),
 Text(0.5531914893617021, 0.08333333333333333, 'gini = 0.367\nsamples = 77\nv
alue = [30, 94]\nclass = b'),
 Text(0.6170212765957447, 0.25, 'NMHC <= 0.27\ngini = 0.437\nsamples = 66\nva
lue = [82, 39]\nclass = a'),
 Text(0.5957446808510638, 0.08333333333333333, 'gini = 0.0\nsamples = 41\nval
ue = [79, 0]\nclass = a'),
 Text(0.6382978723404256, 0.08333333333333333, 'gini = 0.133\nsamples = 25\nv
alue = [3, 39]\nclass = b'),
 Text(0.8297872340425532, 0.5833333333333334, 'TOL <= 1.35\ngini = 0.191\nsam
ples = 2086\nvalue = [2989, 357]\nclass = a'),
 Text(0.7446808510638298, 0.4166666666666667, 'PM10 <= 17.5\ngini = 0.227\nsa
mples = 79\nvalue = [15, 100]\nclass = b'),
 Text(0.7021276595744681, 0.25, 'NO_2 <= 28.5\ngini = 0.036\nsamples = 38\nva
lue = [1, 53]\nclass = b'),
 Text(0.6808510638297872, 0.08333333333333333, 'gini = 0.087\nsamples = 16\nv
alue = [1, 21]\nclass = b'),
 Text(0.723404255319149, 0.08333333333333333, 'gini = 0.0\nsamples = 22\nvalu
e = [0, 32]\nclass = b'),
 Text(0.7872340425531915, 0.25, 'O_3 <= 31.5\ngini = 0.354\nsamples = 41\nval
ue = [14, 47]\nclass = b'),
 Text(0.7659574468085106, 0.08333333333333333, 'gini = 0.0\nsamples = 27\nval
ue = [0, 43]\nclass = b'),
 Text(0.8085106382978723, 0.08333333333333333, 'gini = 0.346\nsamples = 14\nv
alue = [14, 4]\nclass = a'),
 Text(0.9148936170212766, 0.4166666666666667, 'BEN <= 0.25\ngini = 0.146\nsam
ples = 2007\nvalue = [2974, 257]\nclass = a'),
 Text(0.8723404255319149, 0.25, 'NMHC <= 0.19\ngini = 0.304\nsamples = 156\nv
alue = [50, 217]\nclass = b'),
 Text(0.851063829787234, 0.08333333333333333, 'gini = 0.075\nsamples = 31\nva
lue = [49, 2]\nclass = a'),
 Text(0.8936170212765957, 0.08333333333333333, 'gini = 0.009\nsamples = 125\n
value = [1, 215]\nclass = b'),
 Text(0.9574468085106383, 0.25, 'SO_2 <= 5.5\ngini = 0.027\nsamples = 1851\nv
alue = [2924, 40]\nclass = a'),
 Text(0.9361702127659575, 0.08333333333333333, 'gini = 0.102\nsamples = 385\n
value = [577, 33]\nclass = a'),
 Text(0.9787234042553191, 0.08333333333333333, 'gini = 0.006\nsamples = 1466
\nvalue = [2347, 7]\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: 0.8898266972443416
Ridge Regression: 0.8656692723177617
Lasso Regression 0.2797650321514594
ElasticNet Regression: 0.5926559921582942
Logistic Regression: 0.9930446006023232
Random Forest: 0.9960049170251998
```

Random Forest Is Better!!!