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_2010.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	P
0	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	NaN	
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	NaN	
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	NaN	
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.41	7.87(
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.67	22.03(

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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     6666 non-null   object 
 1   BEN      6666 non-null   float64
 2   CO       6666 non-null   float64
 3   EBE      6666 non-null   float64
 4   MXY      6666 non-null   float64
 5   NMHC     6666 non-null   float64
 6   NO_2     6666 non-null   float64
 7   NOx      6666 non-null   float64
 8   OXY      6666 non-null   float64
 9   O_3      6666 non-null   float64
 10  PM10     6666 non-null   float64
 11  PM25     6666 non-null   float64
 12  PXY      6666 non-null   float64
 13  SO_2     6666 non-null   float64
 14  TCH      6666 non-null   float64
 15  TOL      6666 non-null   float64
 16  station  6666 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 937.4+ KB
```

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

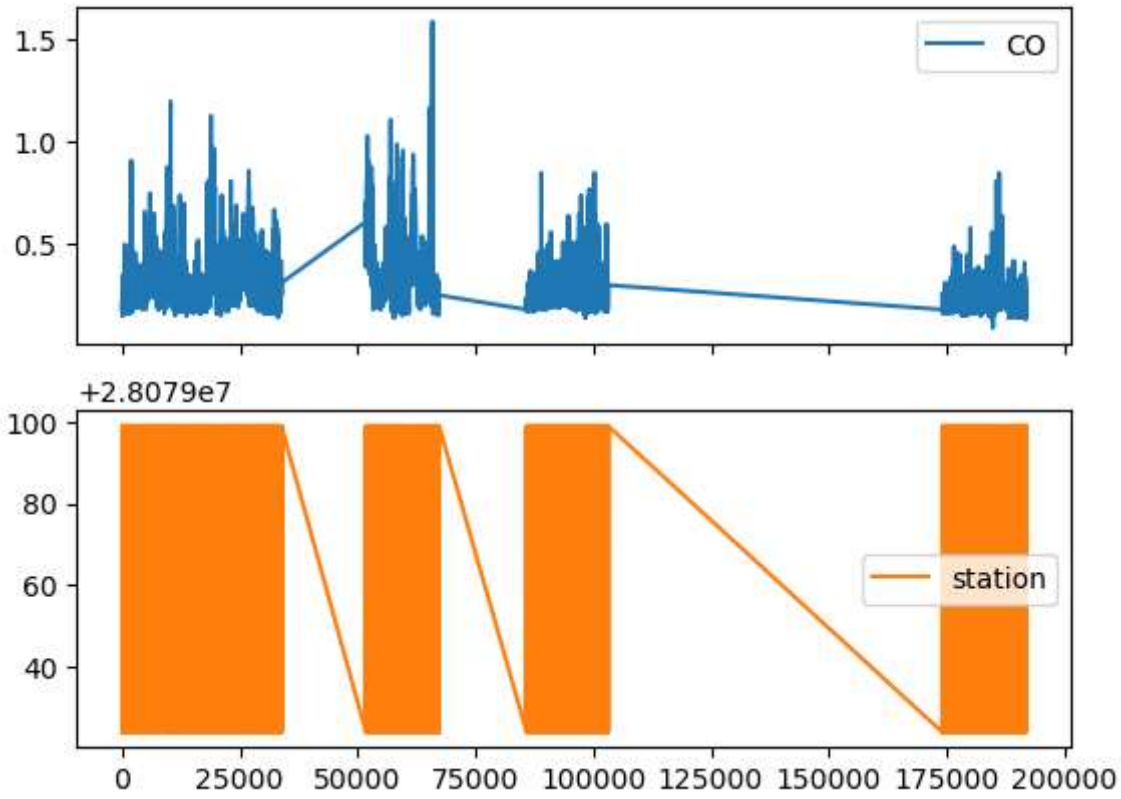
Out[7]:

	CO	station
11	0.18	28079024
23	0.23	28079099
35	0.17	28079024
47	0.21	28079099
59	0.16	28079024
...	...	...
191879	0.26	28079099
191891	0.16	28079024
191903	0.28	28079099
191915	0.16	28079024
191927	0.25	28079099

6666 rows × 2 columns

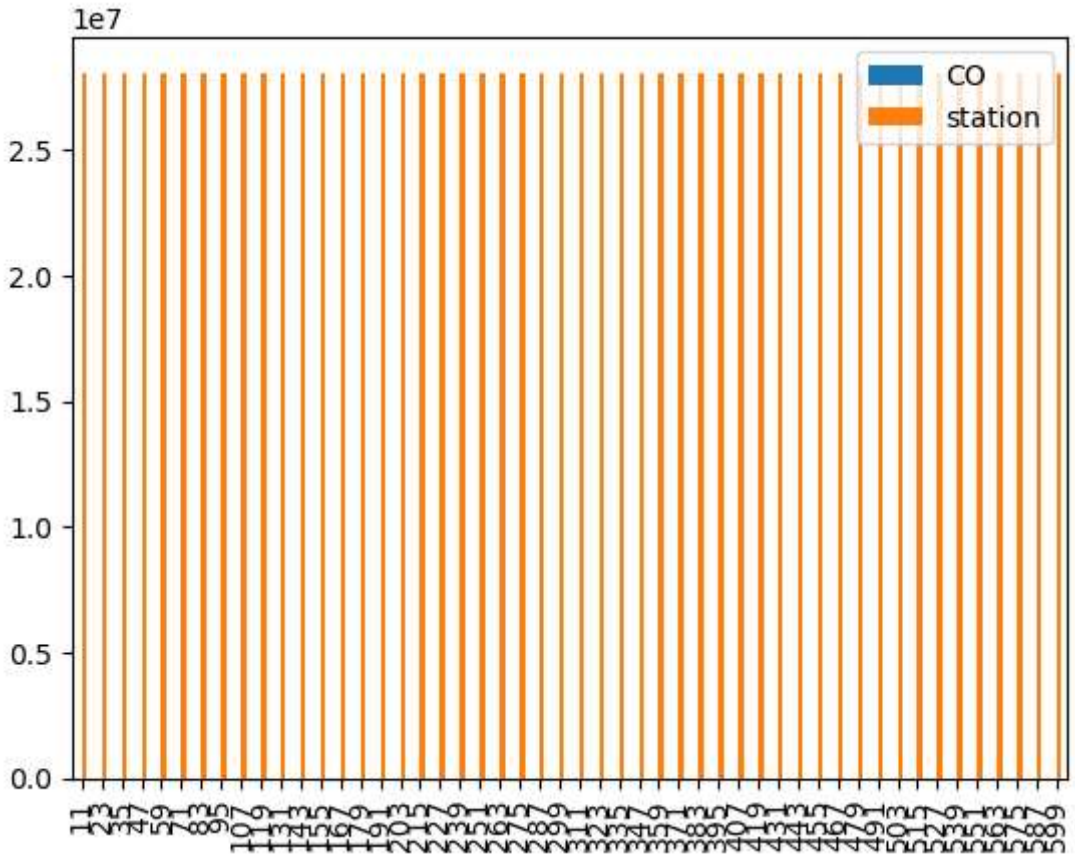
In [8]:
```
data.plot.line(subplots=True)
```

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



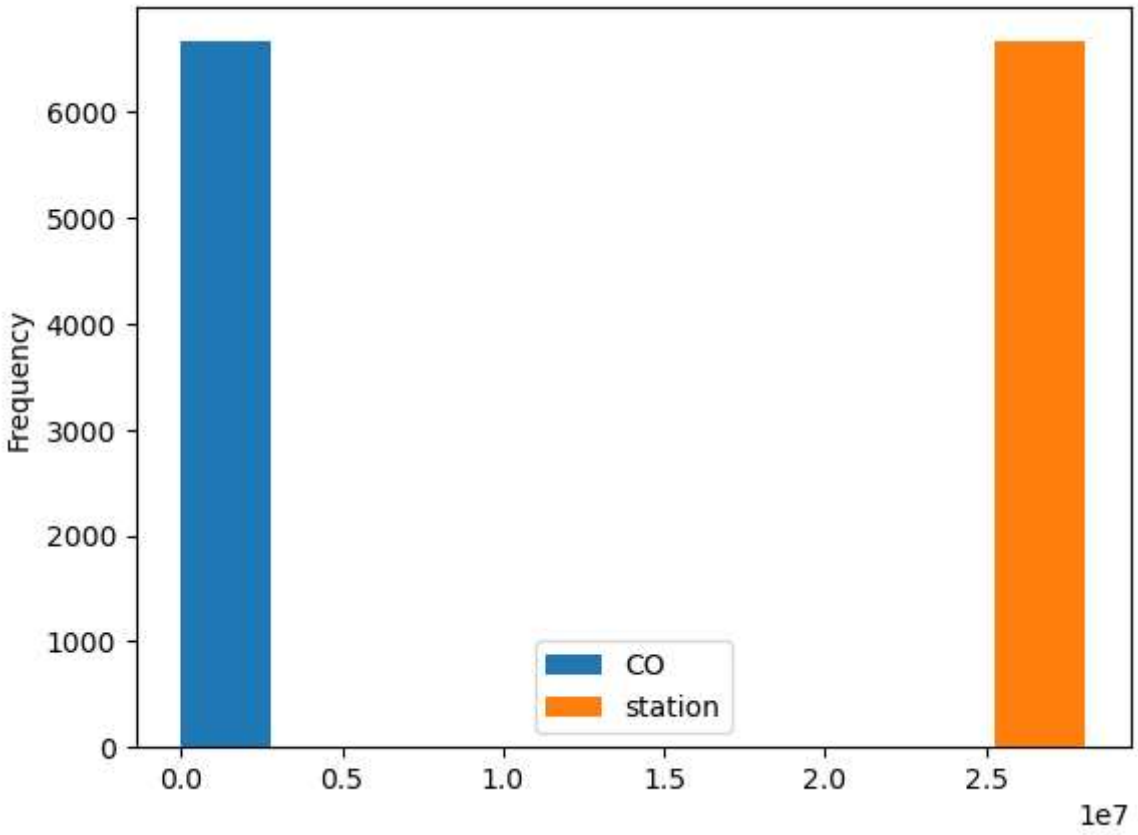
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



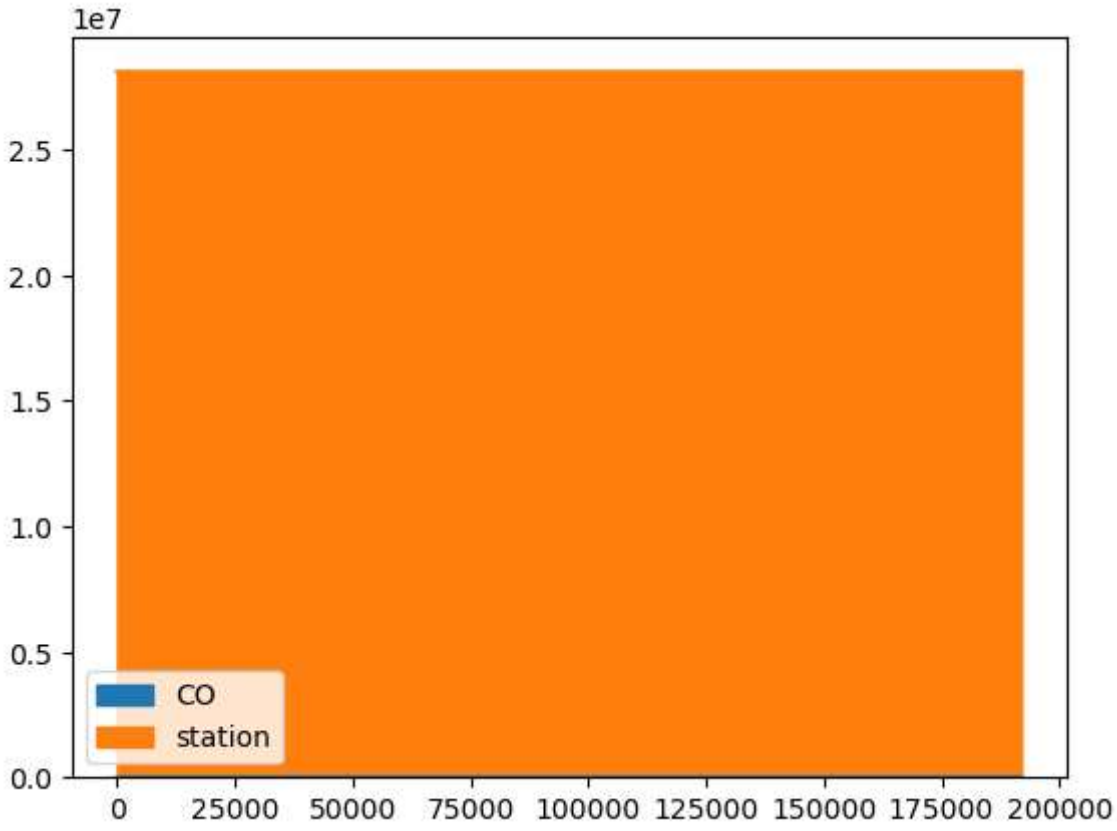
In [10]:
```
data.plot.hist()
```

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



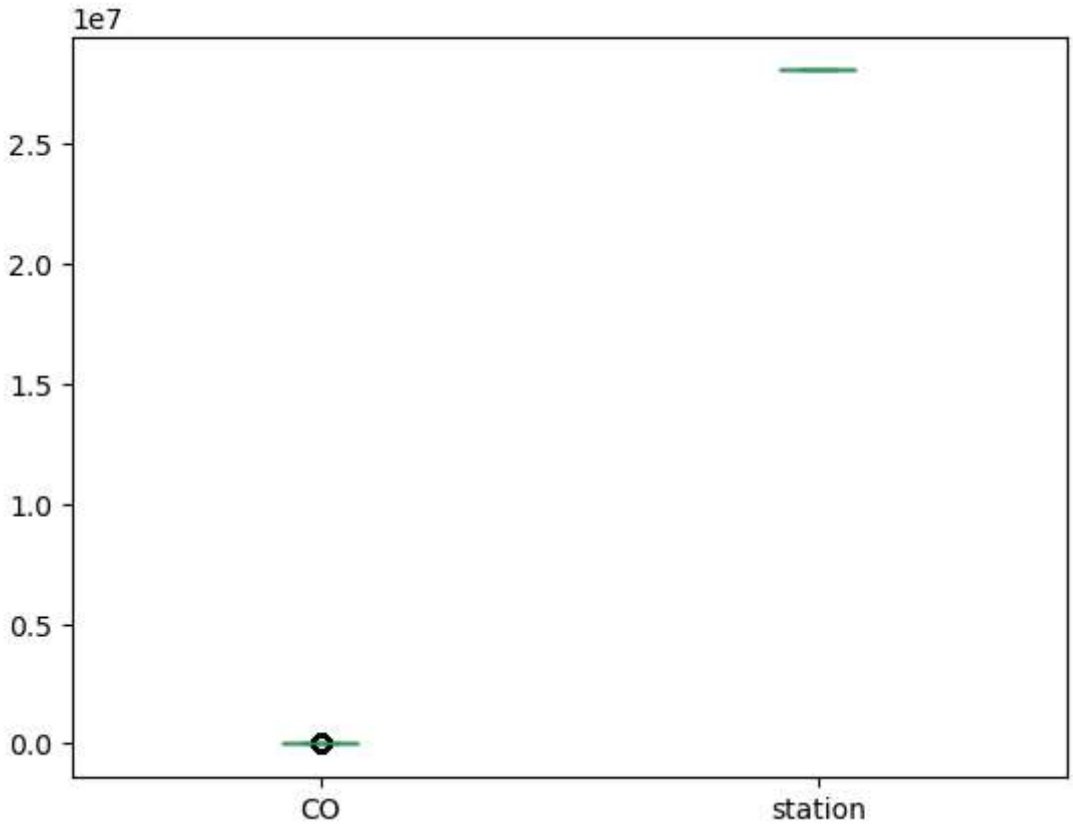
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



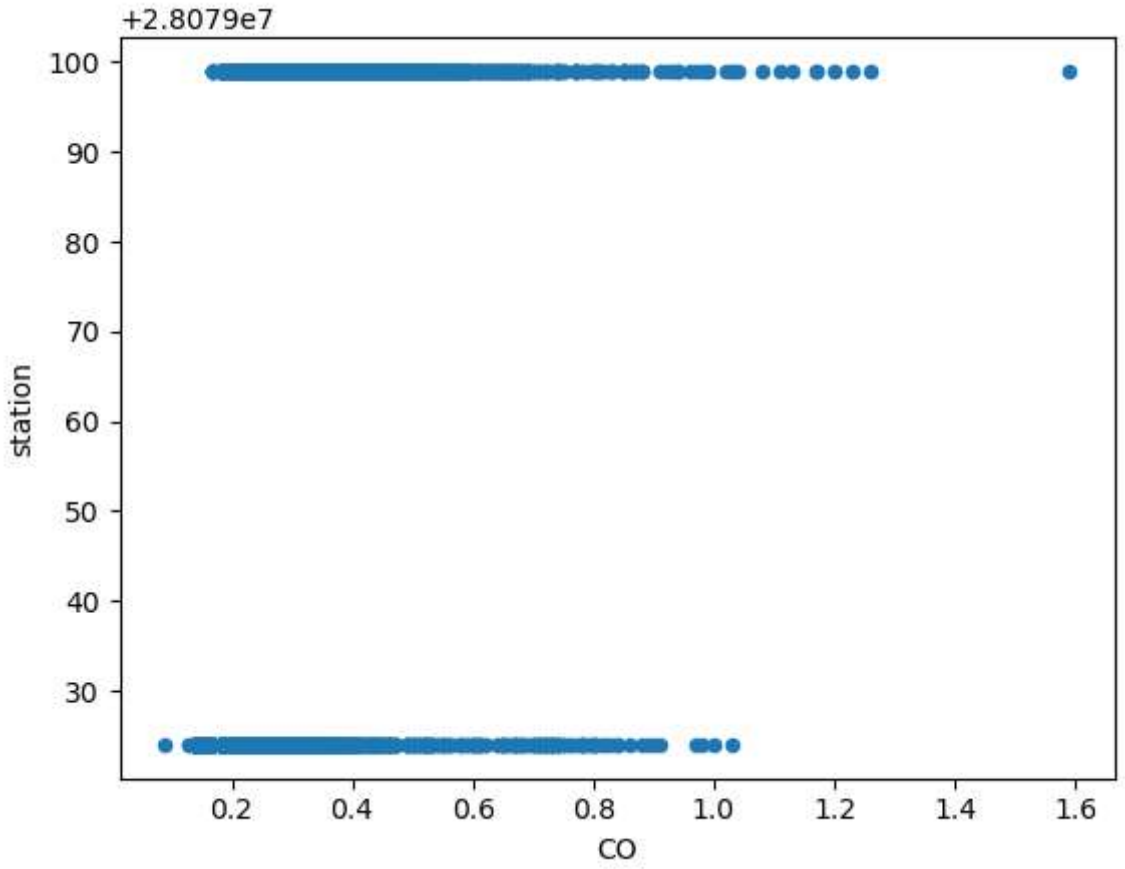
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



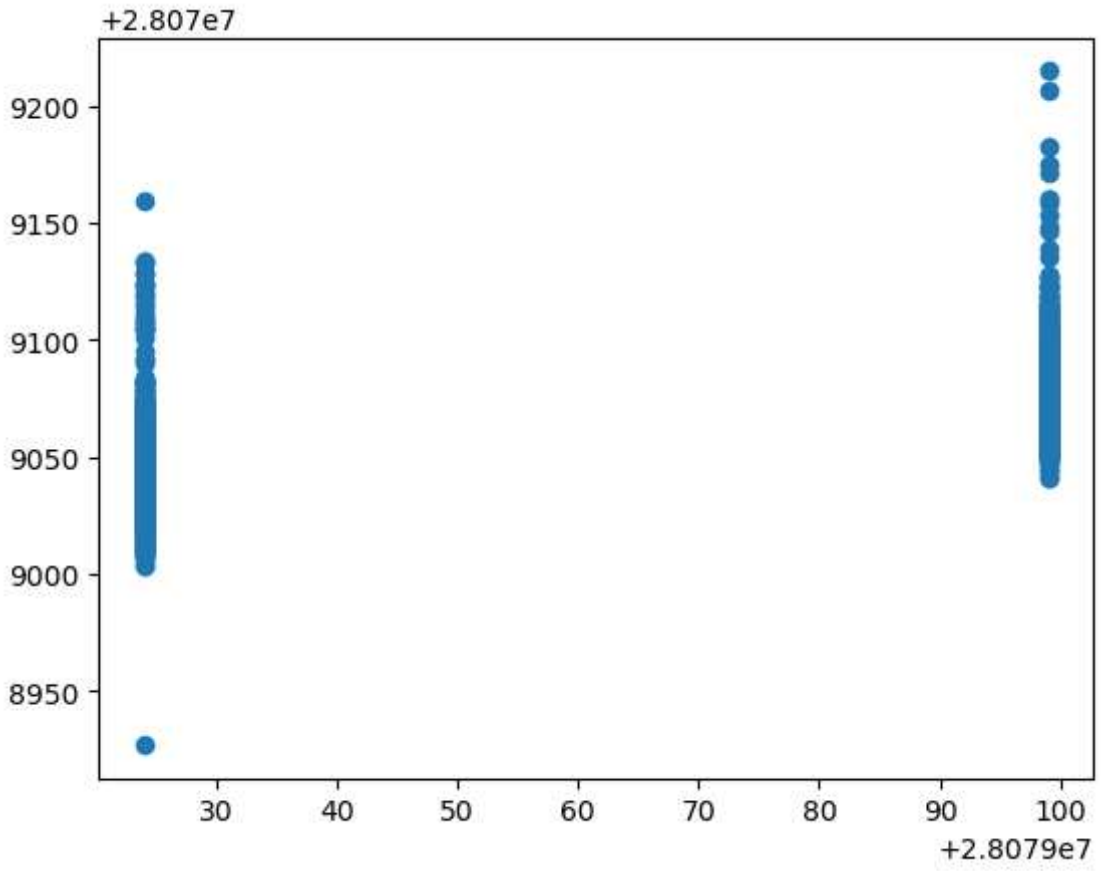
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x13c4b393510>



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

```
0.43366150940207937
0.42459842684297244
```

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)
```

```
0.42233021912700675
0.41222990289857164
```

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

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

Out[19]:
```
0.18385205544759098
```

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([-0.00000000e+00,  2.05990497e-01,  2.54434006e+00, -9.08623254e-01,
       -1.15272971e+00,  2.95491345e-03, -1.23823147e-01,  4.79258603e-01,
       -6.47235893e-02, -1.40631262e-01,  0.00000000e+00,  2.39630962e+00,
        0.00000000e+00,  7.34208674e+00])
```

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

Out[22]:
```
28079031.66796336
```

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

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

Out[24]:
```
0.233761726800773
```

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)))
```

```
30.937956313297153
1077.5214941638412
32.82562252515314
```

Logistics Regression

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

In [27]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'C
'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]:
```
0.8660366036603661
```

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

Out[29]:
```
array([[0., 1.]])
```

Random Forest

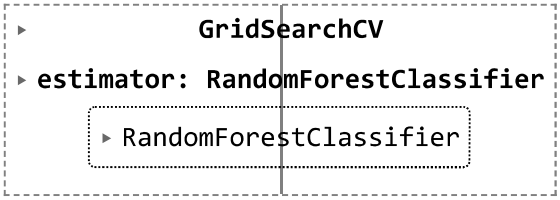
In [30]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

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)
```

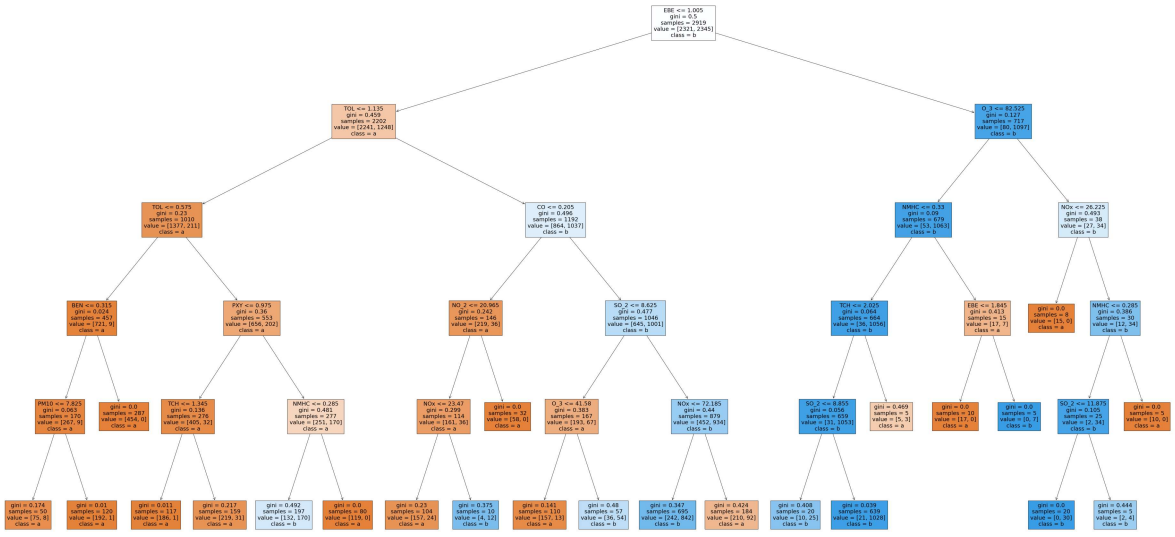
Out[32]:



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(0.581081081081081, 0.9166666666666666, 'EBE <= 1.005\ngini = 0.5\nsampl
es = 2919\nvalue = [2321, 2345]\nclass = b'),
 Text(0.3108108108108108, 0.75, 'TOL <= 1.135\ngini = 0.459\nsamples = 2202\n
value = [2241, 1248]\nclass = a'),
 Text(0.14864864864864866, 0.5833333333333334, 'TOL <= 0.575\ngini = 0.23\nsa
mples = 1010\nvalue = [1377, 211]\nclass = a'),
 Text(0.08108108108108109, 0.4166666666666667, 'BEN <= 0.315\ngini = 0.024\ns
amples = 457\nvalue = [721, 9]\nclass = a'),
 Text(0.05405405405405406, 0.25, 'PM10 <= 7.825\ngini = 0.063\nsamples = 170
\nvalue = [267, 9]\nclass = a'),
 Text(0.02702702702702703, 0.08333333333333333, 'gini = 0.174\nsamples = 50\n
value = [75, 8]\nclass = a'),
 Text(0.08108108108108109, 0.08333333333333333, 'gini = 0.01\nsamples = 120\n
value = [192, 1]\nclass = a'),
 Text(0.10810810810810811, 0.25, 'gini = 0.0\nsamples = 287\nvalue = [454, 0]
\nclass = a'),
 Text(0.21621621621621623, 0.4166666666666667, 'PXY <= 0.975\ngini = 0.36\nsa
mples = 553\nvalue = [656, 202]\nclass = a'),
 Text(0.16216216216216217, 0.25, 'TCH <= 1.345\ngini = 0.136\nsamples = 276\n
value = [405, 32]\nclass = a'),
 Text(0.13513513513513514, 0.08333333333333333, 'gini = 0.011\nsamples = 117
\nvalue = [186, 1]\nclass = a'),
 Text(0.1891891891891892, 0.08333333333333333, 'gini = 0.217\nsamples = 159\n
value = [219, 31]\nclass = a'),
 Text(0.2702702702702703, 0.25, 'NMHC <= 0.285\ngini = 0.481\nsamples = 277\n
value = [251, 170]\nclass = a'),
 Text(0.24324324324324326, 0.08333333333333333, 'gini = 0.492\nsamples = 197
\nvalue = [132, 170]\nclass = b'),
 Text(0.2972972972972973, 0.08333333333333333, 'gini = 0.0\nsamples = 80\nval
ue = [119, 0]\nclass = a'),
 Text(0.47297297297297297, 0.5833333333333334, 'CO <= 0.205\ngini = 0.496\nsa
mples = 1192\nvalue = [864, 1037]\nclass = b'),
 Text(0.40540540540540543, 0.4166666666666667, 'NO_2 <= 20.965\ngini = 0.242
\nsamples = 146\nvalue = [219, 36]\nclass = a'),
 Text(0.3783783783783784, 0.25, 'NOx <= 23.47\ngini = 0.299\nsamples = 114\nv
alue = [161, 36]\nclass = a'),
 Text(0.35135135135135137, 0.08333333333333333, 'gini = 0.23\nsamples = 104\n
value = [157, 24]\nclass = a'),
 Text(0.40540540540540543, 0.08333333333333333, 'gini = 0.375\nsamples = 10\n
value = [4, 12]\nclass = b'),
 Text(0.43243243243243246, 0.25, 'gini = 0.0\nsamples = 32\nvalue = [58, 0]\n
class = a'),
 Text(0.5405405405405406, 0.4166666666666667, 'SO_2 <= 8.625\ngini = 0.477\ns
amples = 1046\nvalue = [645, 1001]\nclass = b'),
 Text(0.4864864864864865, 0.25, 'O_3 <= 41.58\ngini = 0.383\nsamples = 167\nv
alue = [193, 67]\nclass = a'),
 Text(0.4594594594594595, 0.08333333333333333, 'gini = 0.141\nsamples = 110\n
value = [157, 13]\nclass = a'),
 Text(0.5135135135135135, 0.08333333333333333, 'gini = 0.48\nsamples = 57\nva
lue = [36, 54]\nclass = b'),
 Text(0.5945945945945946, 0.25, 'NOx <= 72.185\ngini = 0.44\nsamples = 879\nv
alue = [452, 934]\nclass = b'),
 Text(0.5675675675675675, 0.08333333333333333, 'gini = 0.347\nsamples = 695\n
value = [242, 842]\nclass = b'),
 Text(0.6216216216216216, 0.08333333333333333, 'gini = 0.424\nsamples = 184\n
value = [210, 92]\nclass = a'),
 Text(0.8513513513513513, 0.75, 'O_3 <= 82.525\ngini = 0.127\nsamples = 717\n

```
value = [80, 1097]\nclass = b'),
 Text(0.7837837837837838, 0.5833333333333334, 'NMHC <= 0.33\ngini = 0.09\nsam
ples = 679\nvalue = [53, 1063]\nclass = b'),
 Text(0.7297297297297297, 0.4166666666666667, 'TCH <= 2.025\ngini = 0.064\nsa
mples = 664\nvalue = [36, 1056]\nclass = b'),
 Text(0.7027027027027027, 0.25, 'SO_2 <= 8.855\ngini = 0.056\nsamples = 659\n
value = [31, 1053]\nclass = b'),
 Text(0.6756756756756757, 0.08333333333333333, 'gini = 0.408\nsamples = 20\nv
alue = [10, 25]\nclass = b'),
 Text(0.7297297297297297, 0.08333333333333333, 'gini = 0.039\nsamples = 639\n
value = [21, 1028]\nclass = b'),
 Text(0.7567567567567568, 0.25, 'gini = 0.469\nsamples = 5\nvalue = [5, 3]\nc
lass = a'),
 Text(0.8378378378378378, 0.4166666666666667, 'EBE <= 1.845\ngini = 0.413\nsa
mples = 15\nvalue = [17, 7]\nclass = a'),
 Text(0.8108108108108109, 0.25, 'gini = 0.0\nsamples = 10\nvalue = [17, 0]\nc
lass = a'),
 Text(0.8648648648648649, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [0, 7]\ncla
ss = b'),
 Text(0.918918918918919, 0.5833333333333334, 'NOx <= 26.225\ngini = 0.493\nsa
mples = 38\nvalue = [27, 34]\nclass = b'),
 Text(0.8918918918918919, 0.4166666666666667, 'gini = 0.0\nsamples = 8\nvalue
= [15, 0]\nclass = a'),
 Text(0.9459459459459459, 0.4166666666666667, 'NMHC <= 0.285\ngini = 0.386\ns
amples = 30\nvalue = [12, 34]\nclass = b'),
 Text(0.918918918918919, 0.25, 'SO_2 <= 11.875\ngini = 0.105\nsamples = 25\nv
alue = [2, 34]\nclass = b'),
 Text(0.8918918918918919, 0.08333333333333333, 'gini = 0.0\nsamples = 20\nval
ue = [0, 30]\nclass = b'),
 Text(0.9459459459459459, 0.08333333333333333, 'gini = 0.444\nsamples = 5\nva
lue = [2, 4]\nclass = b'),
 Text(0.972972972972973, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [10, 0]\ncla
ss = 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: 0.43366150940207937
Ridge Regression: 0.42233021912700675
Lasso Regression 0.18385205544759098
ElasticNet Regression: 0.233761726800773
Logistic Regression: 0.8660366036603661
Random Forest: 0.9297042434633519
```

Logistic Is Better!!!