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_2002.csv")
```

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.990002	NaN
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.2	0.15	98.150002	153.399994	2.67	6.85	20.980000	2.53
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.440001	NaN
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.180000	NaN
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.330002	NaN

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: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

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

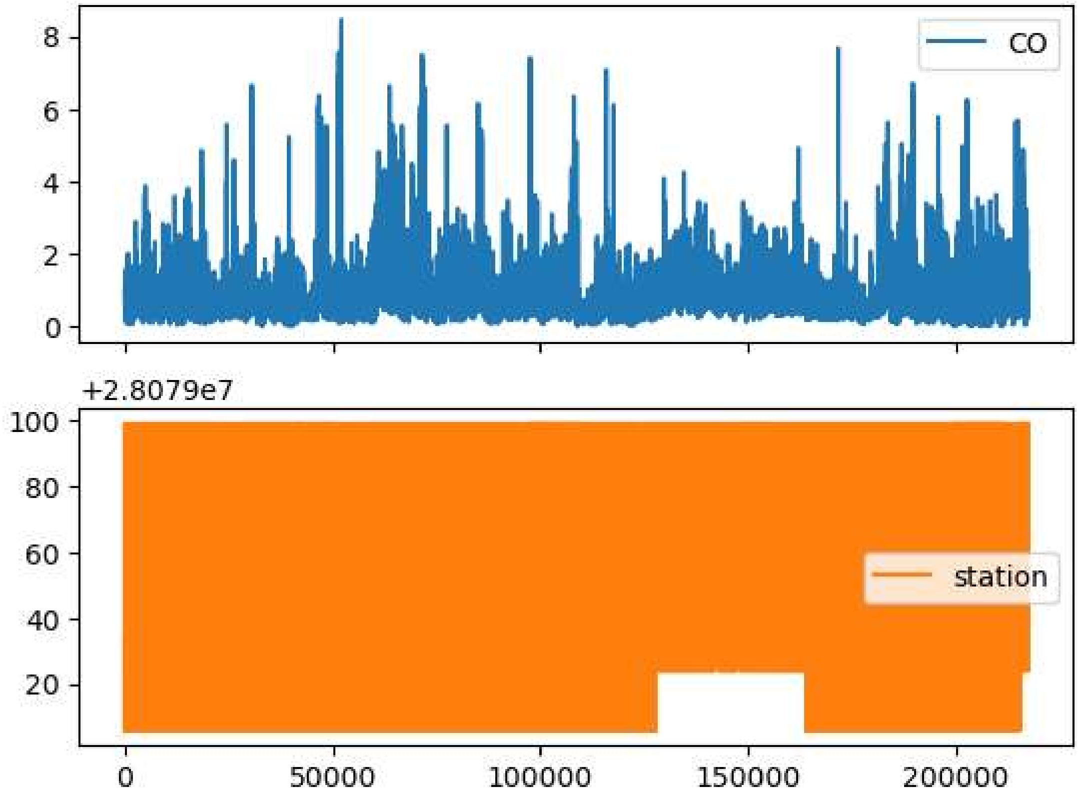
Out[7]:

	CO	station
1	0.71	28079035
5	0.72	28079006
22	0.80	28079024
24	1.04	28079099
26	0.53	28079035
...	...	...
217269	0.28	28079024
217271	1.30	28079099
217273	0.97	28079035
217293	0.58	28079024
217295	1.17	28079099

32381 rows × 2 columns

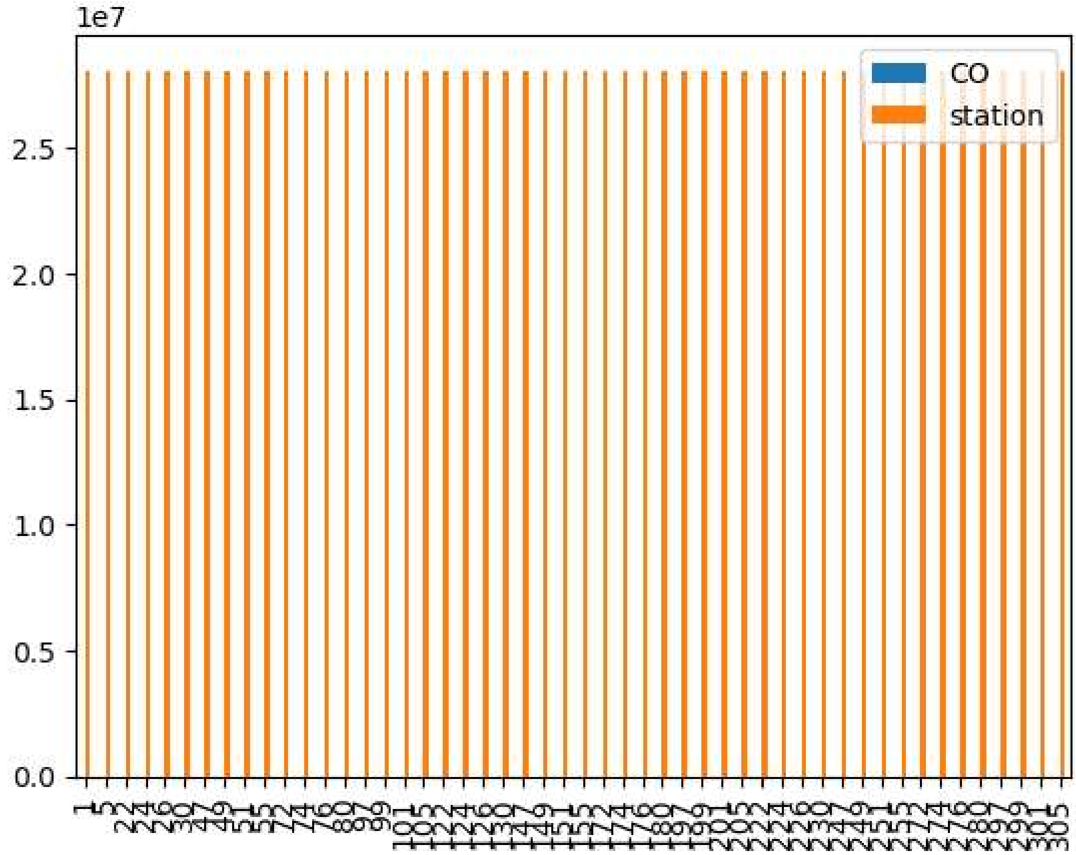
In [8]:
```
data.plot.line(subplots=True)
```

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



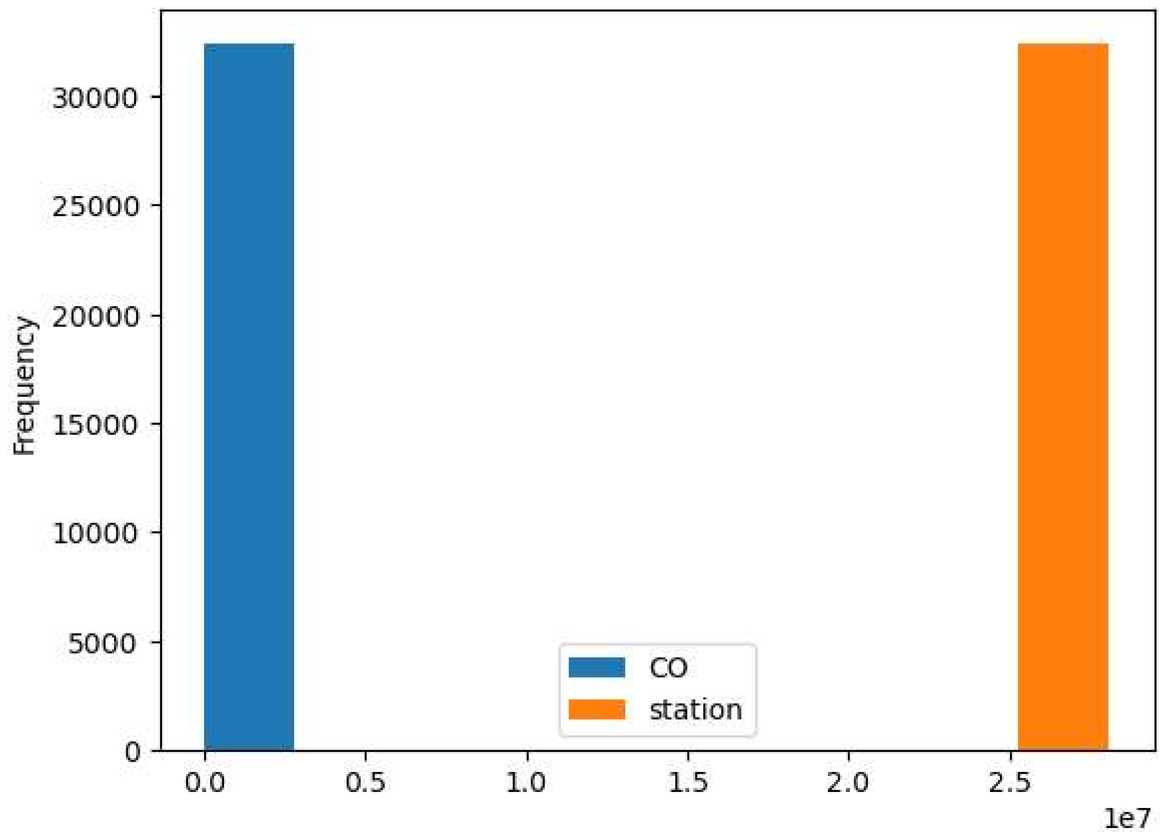
In [9]:
```
b=data[0:50]
b.plot.bar()
```

Out[9]: <Axes: >



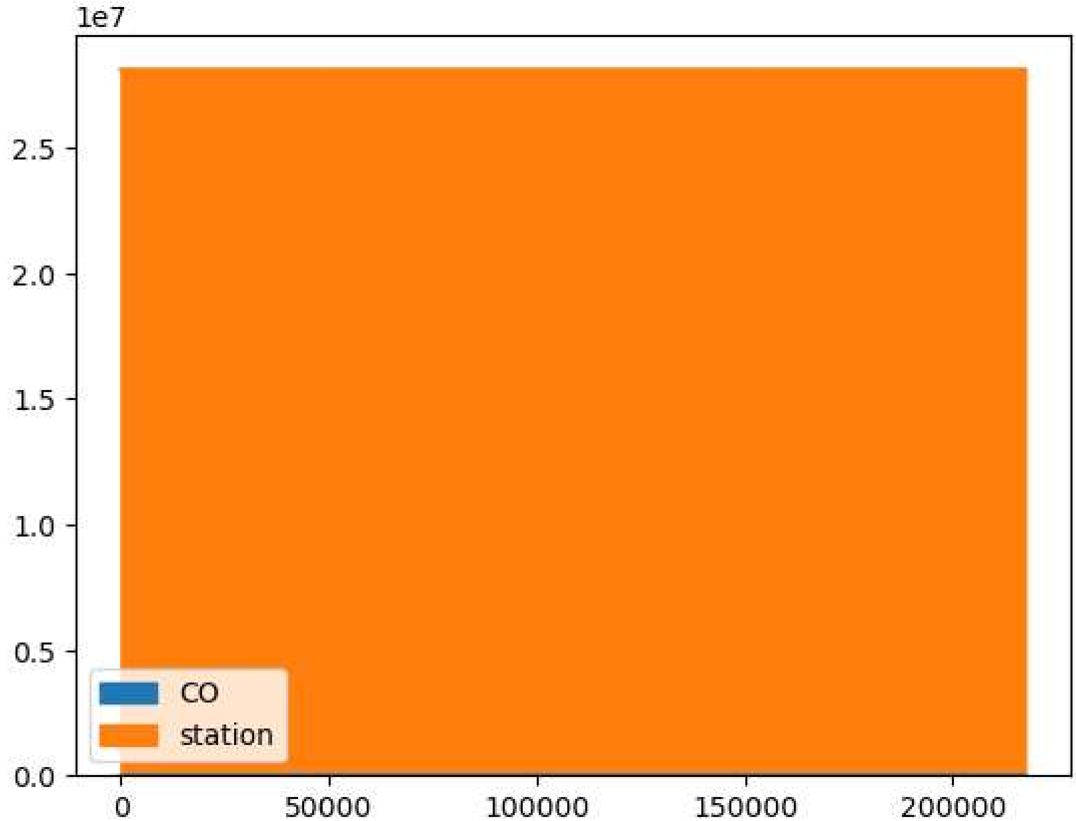
In [10]:
```
data.plot.hist()
```

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



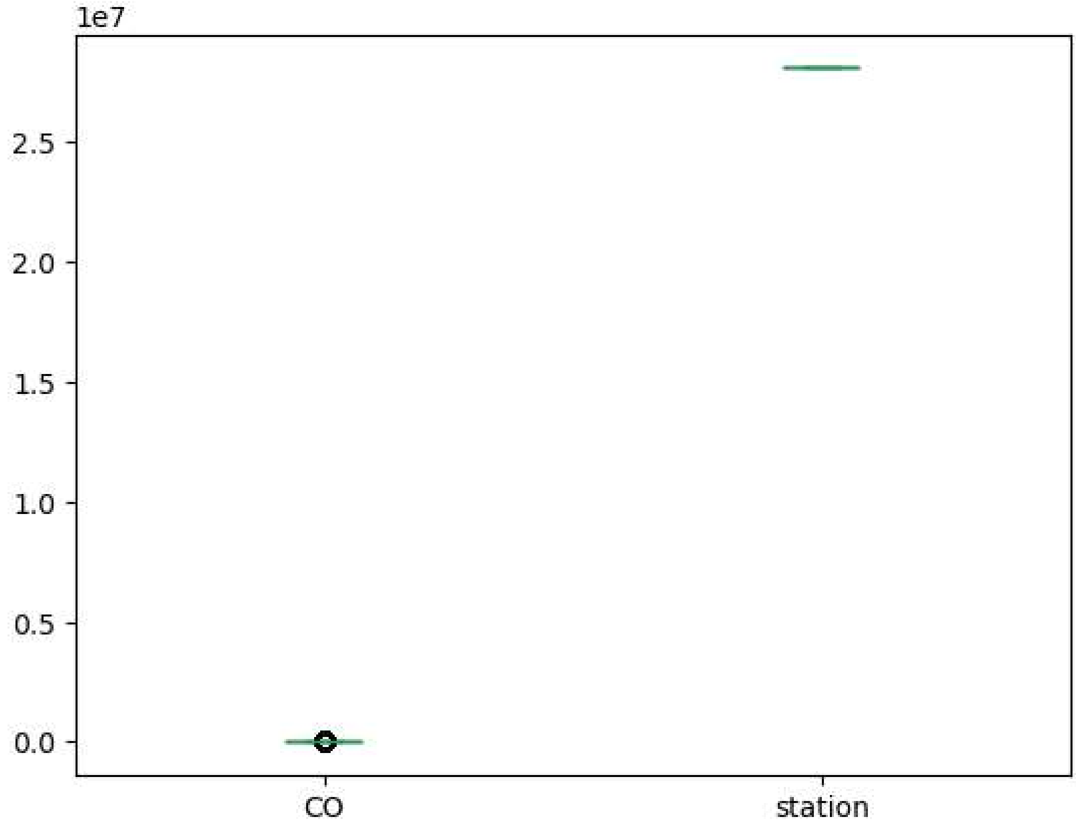
In [11]:
```
data.plot.area()
```

Out[11]: <Axes: >



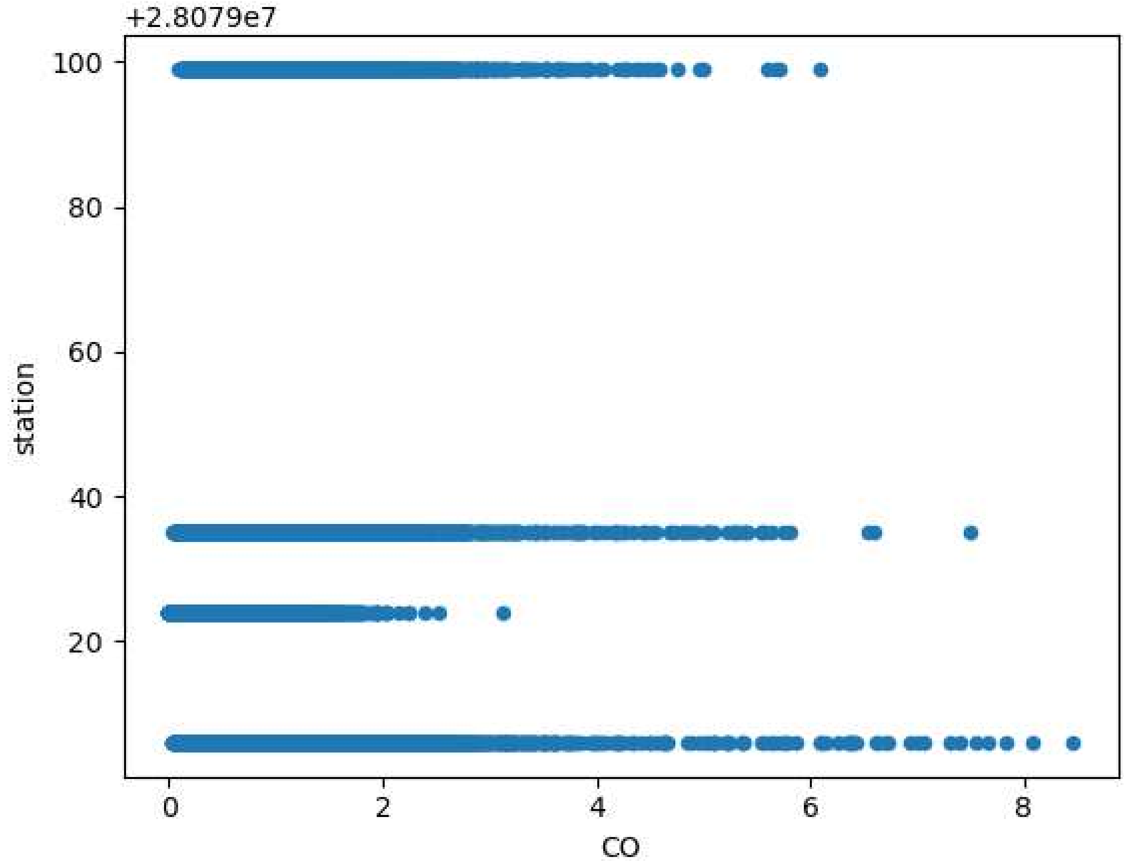
In [12]:
```
data.plot.box()
```

Out[12]: <Axes: >



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

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



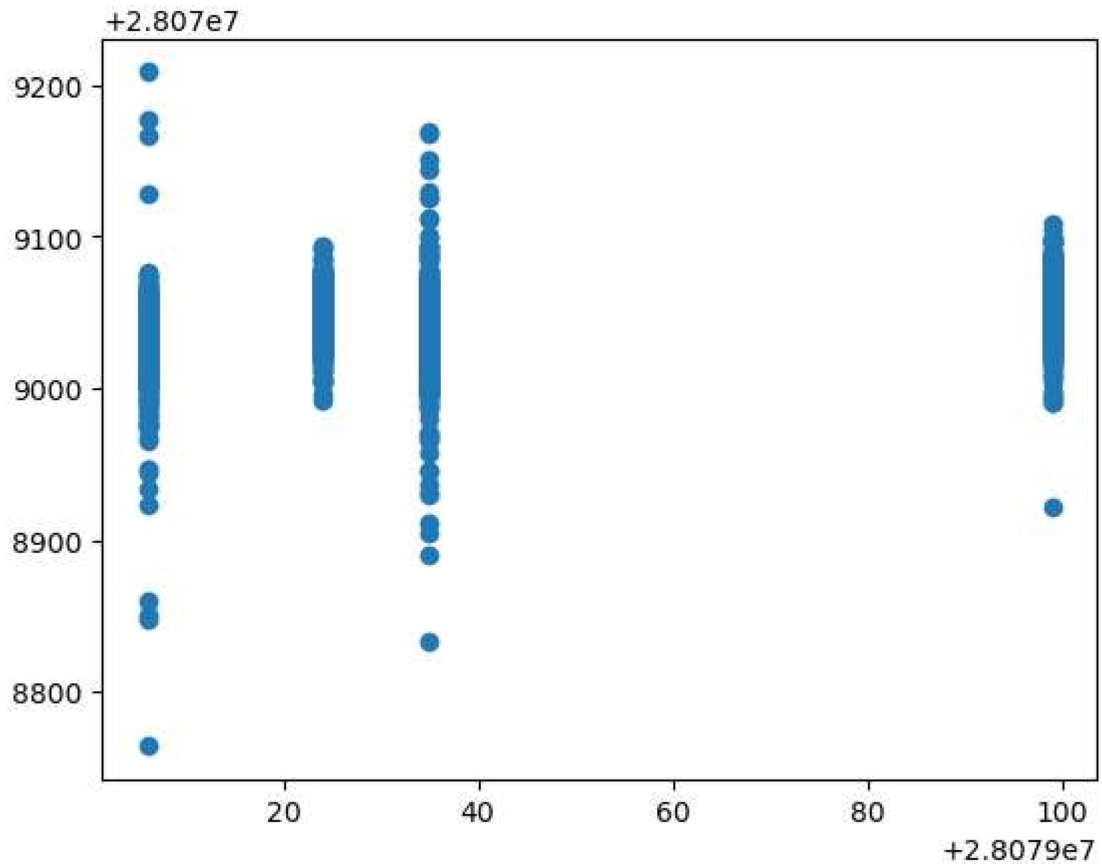
In [14]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

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 0x1491a728910>



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

```
0.19808264306622458
0.1988584434706827
```

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)
```

```
0.19746964338217032
0.1986451008739084
```

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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

Out[19]:
```
0.05959650402235683
```

ElasticNet

In [20]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[20]:
```
ElasticNet()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [21]:
```
en.coef_
```

Out[21]:
```
array([ 0.92575093,  0.        , -2.95775812,  1.56224474,  0.19104727,
        0.22680299, -0.03020663, -2.425336  , -0.02546159,  0.00742185,
        2.35459024,  0.39870298,  1.05678995, -1.1689577 ])
```

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

Out[22]:
```
28079038.632254932
```

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

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

Out[24]: 0.1000461075926431

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)))
```

```
28.59476137351143
1126.0759677707854
33.55705540971653
```

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]: 0.8480899292795158

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

Out[29]: array([[2.53067744e-10, 3.44181204e-71, 1.00000000e+00, 1.42366623e-13]])

Random Forest

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

Out[30]:
```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

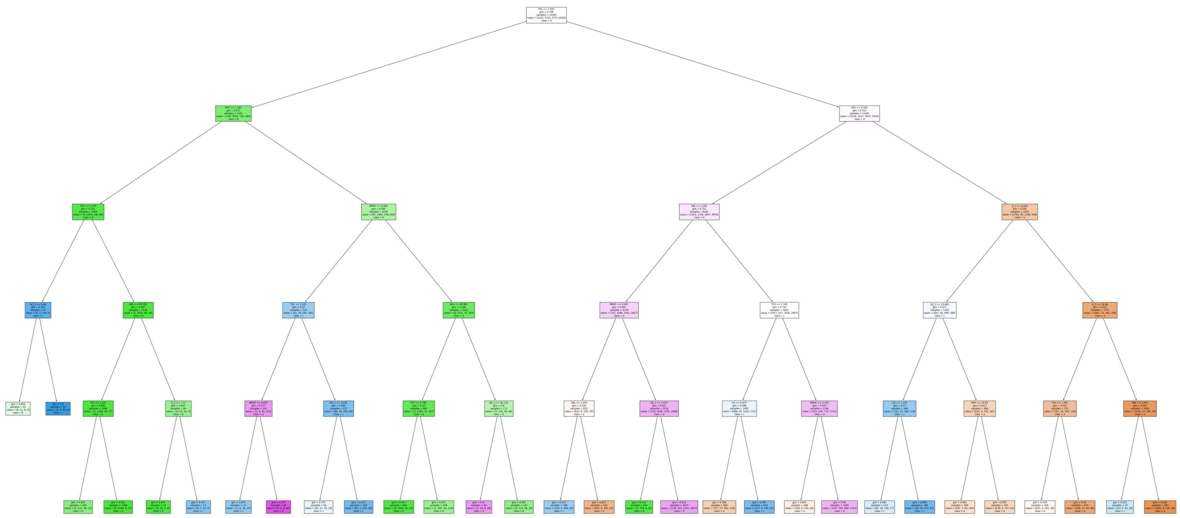
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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(0.4639830508474576, 0.9166666666666666, 'PXY <= 1.005\ngini = 0.749\nsa
mples = 14380\nvalue = [5122, 5733, 5773, 6038]\nclass = d'),
 Text(0.19915254237288135, 0.75, 'MXY <= 1.105\ngini = 0.413\nsamples = 3283
\nvalue = [106, 3916, 736, 484]\nclass = b'),
 Text(0.07627118644067797, 0.5833333333333334, 'TCH <= 1.175\ngini = 0.151\ns
amples = 1605\nvalue = [9, 2354, 158, 40]\nclass = b'),
 Text(0.03389830508474576, 0.4166666666666667, 'SO_2 <= 6.66\ngini = 0.305\ns
amples = 67\nvalue = [8, 11, 89, 0]\nclass = c'),
 Text(0.01694915254237288, 0.25, 'gini = 0.658\nsamples = 15\nvalue = [8, 11,
8, 0]\nclass = b'),
 Text(0.05084745762711865, 0.25, 'gini = 0.0\nsamples = 52\nvalue = [0, 0, 8
1, 0]\nclass = c'),
 Text(0.11864406779661017, 0.4166666666666667, 'NOx <= 89.385\ngini = 0.087\n
samples = 1538\nvalue = [1, 2343, 69, 40]\nclass = b'),
 Text(0.0847457627118644, 0.25, 'TCH <= 1.225\ngini = 0.068\nsamples = 1494\n
value = [1, 2286, 45, 37]\nclass = b'),
 Text(0.06779661016949153, 0.08333333333333333, 'gini = 0.442\nsamples = 106
\nvalue = [1, 122, 36, 12]\nclass = b'),
 Text(0.1016949152542373, 0.08333333333333333, 'gini = 0.031\nsamples = 1388
\nvalue = [0, 2164, 9, 25]\nclass = b'),
 Text(0.15254237288135594, 0.25, 'O_3 <= 7.23\ngini = 0.457\nsamples = 44\nva
lue = [0, 57, 24, 3]\nclass = b'),
 Text(0.13559322033898305, 0.08333333333333333, 'gini = 0.074\nsamples = 27\n
value = [0, 50, 2, 0]\nclass = b'),
 Text(0.1694915254237288, 0.08333333333333333, 'gini = 0.471\nsamples = 17\nv
alue = [0, 7, 22, 3]\nclass = c'),
 Text(0.3220338983050847, 0.5833333333333334, 'NMHC <= 0.065\ngini = 0.585\ns
amples = 1678\nvalue = [97, 1562, 578, 444]\nclass = b'),
 Text(0.2542372881355932, 0.4166666666666667, 'TOL <= 2.435\ngini = 0.55\nsam
ples = 516\nvalue = [91, 30, 503, 191]\nclass = c'),
 Text(0.22033898305084745, 0.25, 'NMHC <= 0.035\ngini = 0.477\nsamples = 101
\nvalue = [3, 4, 45, 101]\nclass = d'),
 Text(0.2033898305084746, 0.08333333333333333, 'gini = 0.475\nsamples = 37\nv
alue = [3, 0, 36, 15]\nclass = c'),
 Text(0.23728813559322035, 0.08333333333333333, 'gini = 0.235\nsamples = 64\n
value = [0, 4, 9, 86]\nclass = d'),
 Text(0.288135593220339, 0.25, 'NO_2 <= 19.82\ngini = 0.484\nsamples = 415\nv
alue = [88, 26, 458, 90]\nclass = c'),
 Text(0.2711864406779661, 0.08333333333333333, 'gini = 0.715\nsamples = 56\nv
alue = [26, 21, 33, 10]\nclass = c'),
 Text(0.3050847457627119, 0.08333333333333333, 'gini = 0.417\nsamples = 359\n
value = [62, 5, 425, 80]\nclass = c'),
 Text(0.3898305084745763, 0.4166666666666667, 'NOx <= 80.985\ngini = 0.306\ns
amples = 1162\nvalue = [6, 1532, 75, 253]\nclass = b'),
 Text(0.3559322033898305, 0.25, 'PXY <= 0.785\ngini = 0.24\nsamples = 990\nva
lue = [1, 1391, 31, 187]\nclass = b'),
 Text(0.3389830508474576, 0.08333333333333333, 'gini = 0.101\nsamples = 681\n
value = [0, 1046, 15, 43]\nclass = b'),
 Text(0.3728813559322034, 0.08333333333333333, 'gini = 0.453\nsamples = 309\n
value = [1, 345, 16, 144]\nclass = b'),
 Text(0.423728813559322, 0.25, 'NO_2 <= 62.125\ngini = 0.6\nsamples = 172\nva
lue = [5, 141, 44, 66]\nclass = b'),
 Text(0.4067796610169492, 0.08333333333333333, 'gini = 0.53\nsamples = 49\nva
lue = [1, 18, 8, 46]\nclass = d'),
 Text(0.4406779661016949, 0.08333333333333333, 'gini = 0.497\nsamples = 123\n
value = [4, 123, 36, 20]\nclass = b'),
 Text(0.7288135593220338, 0.75, 'OXY <= 4.785\ngini = 0.721\nsamples = 11097
```

```
\nvalue = [5016, 1817, 5037, 5554]\nclass = d'),
 Text(0.5932203389830508, 0.5833333333333334, 'EBE <= 2.285\ngini = 0.711\nsa
mples = 8182\nvalue = [2314, 1726, 3847, 4970]\nclass = d'),
 Text(0.5254237288135594, 0.4166666666666667, 'NMHC <= 0.045\ngini = 0.682\ns
amples = 4339\nvalue = [757, 1099, 1931, 3067]\nclass = d'),
 Text(0.4915254237288136, 0.25, 'EBE <= 1.555\ngini = 0.556\nsamples = 823\nv
alue = [621, 0, 576, 79]\nclass = a'),
 Text(0.4745762711864407, 0.08333333333333333, 'gini = 0.521\nsamples = 399\n
value = [156, 0, 389, 69]\nclass = c'),
 Text(0.5084745762711864, 0.08333333333333333, 'gini = 0.427\nsamples = 424\n
value = [465, 0, 187, 10]\nclass = a'),
 Text(0.559322033898305, 0.25, 'SO_2 <= 5.625\ngini = 0.615\nsamples = 3516\n
value = [136, 1099, 1355, 2988]\nclass = d'),
 Text(0.5423728813559322, 0.08333333333333333, 'gini = 0.135\nsamples = 543\n
value = [0, 779, 0, 61]\nclass = b'),
 Text(0.576271186440678, 0.08333333333333333, 'gini = 0.531\nsamples = 2973\n
value = [136, 320, 1355, 2927]\nclass = d'),
 Text(0.6610169491525424, 0.4166666666666667, 'TCH <= 1.345\ngini = 0.719\nsa
mples = 3843\nvalue = [1557, 627, 1916, 1903]\nclass = c'),
 Text(0.6271186440677966, 0.25, 'CO <= 0.675\ngini = 0.586\nsamples = 1495\nv
alue = [994, 63, 1142, 172]\nclass = c'),
 Text(0.6101694915254238, 0.08333333333333333, 'gini = 0.588\nsamples = 862\n
value = [777, 57, 402, 150]\nclass = a'),
 Text(0.6440677966101694, 0.08333333333333333, 'gini = 0.387\nsamples = 633\n
value = [217, 6, 740, 22]\nclass = c'),
 Text(0.6949152542372882, 0.25, 'NMHC <= 0.105\ngini = 0.679\nsamples = 2348
\nvalue = [563, 564, 774, 1731]\nclass = d'),
 Text(0.6779661016949152, 0.08333333333333333, 'gini = 0.605\nsamples = 168\n
value = [126, 6, 105, 28]\nclass = a'),
 Text(0.711864406779661, 0.08333333333333333, 'gini = 0.66\nsamples = 2180\nv
alue = [437, 558, 669, 1703]\nclass = d'),
 Text(0.864406779661017, 0.5833333333333334, 'O_3 <= 8.695\ngini = 0.565\nsam
ples = 2915\nvalue = [2702, 91, 1190, 584]\nclass = a'),
 Text(0.7966101694915254, 0.4166666666666667, 'SO_2 <= 20.965\ngini = 0.671\n
samples = 1158\nvalue = [641, 60, 699, 390]\nclass = c'),
 Text(0.7627118644067796, 0.25, 'CO <= 1.335\ngini = 0.57\nsamples = 498\nval
ue = [111, 57, 469, 129]\nclass = c'),
 Text(0.7457627118644068, 0.08333333333333333, 'gini = 0.683\nsamples = 213\n
value = [92, 19, 136, 77]\nclass = c'),
 Text(0.7796610169491526, 0.08333333333333333, 'gini = 0.409\nsamples = 285\n
value = [19, 38, 333, 52]\nclass = c'),
 Text(0.8305084745762712, 0.25, 'MXY <= 20.83\ngini = 0.617\nsamples = 660\nv
alue = [530, 3, 230, 261]\nclass = a'),
 Text(0.8135593220338984, 0.08333333333333333, 'gini = 0.605\nsamples = 369\n
value = [291, 3, 83, 200]\nclass = a'),
 Text(0.847457627118644, 0.08333333333333333, 'gini = 0.587\nsamples = 291\nv
alue = [239, 0, 147, 61]\nclass = a'),
 Text(0.9322033898305084, 0.4166666666666667, 'O_3 <= 18.08\ngini = 0.413\nsa
mples = 1757\nvalue = [2061, 31, 491, 194]\nclass = a'),
 Text(0.8983050847457628, 0.25, 'TCH <= 1.465\ngini = 0.522\nsamples = 753\nv
alue = [767, 18, 305, 116]\nclass = a'),
 Text(0.8813559322033898, 0.08333333333333333, 'gini = 0.539\nsamples = 329\n
value = [261, 4, 241, 18]\nclass = a'),
 Text(0.9152542372881356, 0.08333333333333333, 'gini = 0.42\nsamples = 424\nv
alue = [506, 14, 64, 98]\nclass = a'),
 Text(0.9661016949152542, 0.25, 'EBE <= 3.855\ngini = 0.305\nsamples = 1004\n
value = [1294, 13, 186, 78]\nclass = a'),
```

```
 Text(0.9491525423728814, 0.08333333333333333, 'gini = 0.625\nsamples = 73\nv
alue = [13, 5, 52, 34]\nclass = c'),
 Text(0.9830508474576272, 0.08333333333333333, 'gini = 0.228\nsamples = 931\n
value = [1281, 8, 134, 44]\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: 0.19808264306622458
Ridge Regression: 0.19746964338217032
Lasso Regression 0.05959650402235683
ElasticNet Regression: 0.1000461075926431
Logistic Regression: 0.8480899292795158
Random Forest: 0.7738462895967528
```

Logistic Is Better!!!