In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
df=pd.read_csv("madrid_2001.csv")
```

In [4]:
```
df.head()
```

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY	SO_2	T
0	2001-08-01 01:00:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	105.000000	NaN	6.34	M
1	2001-08-01 01:00:00	1.5	0.34	1.49	4.1	0.07	56.250000	75.169998	2.11	42.160000	100.599998	1.73	8.11	
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.099998	NaN	7.85	M
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.779999	NaN	6.46	M
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.180000	NaN	8.80	M

In [5]:
```
df=df.dropna()
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

In [9]:
```
data=df[['CO','station']]
data
```

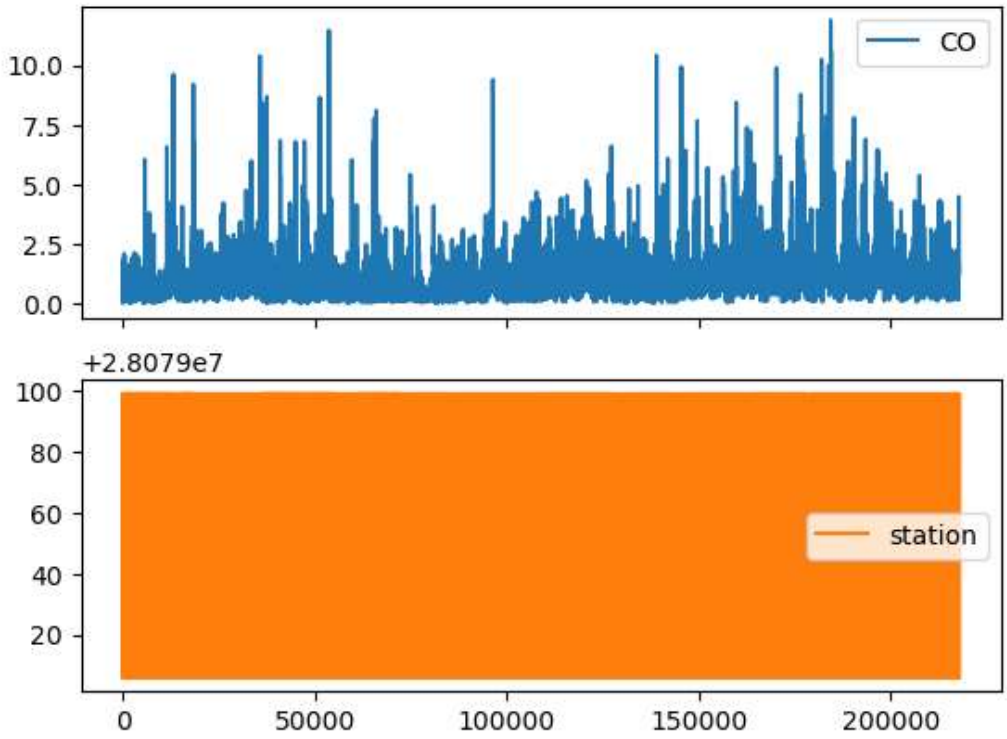
Out[9]:

	CO	station
1	0.34	28079035
5	0.63	28079006
21	0.43	28079024
23	0.34	28079099
25	0.06	28079035
...	...	...
217829	4.48	28079006
217847	2.65	28079099
217849	1.22	28079035
217853	1.83	28079006
217871	1.62	28079099

29669 rows × 2 columns

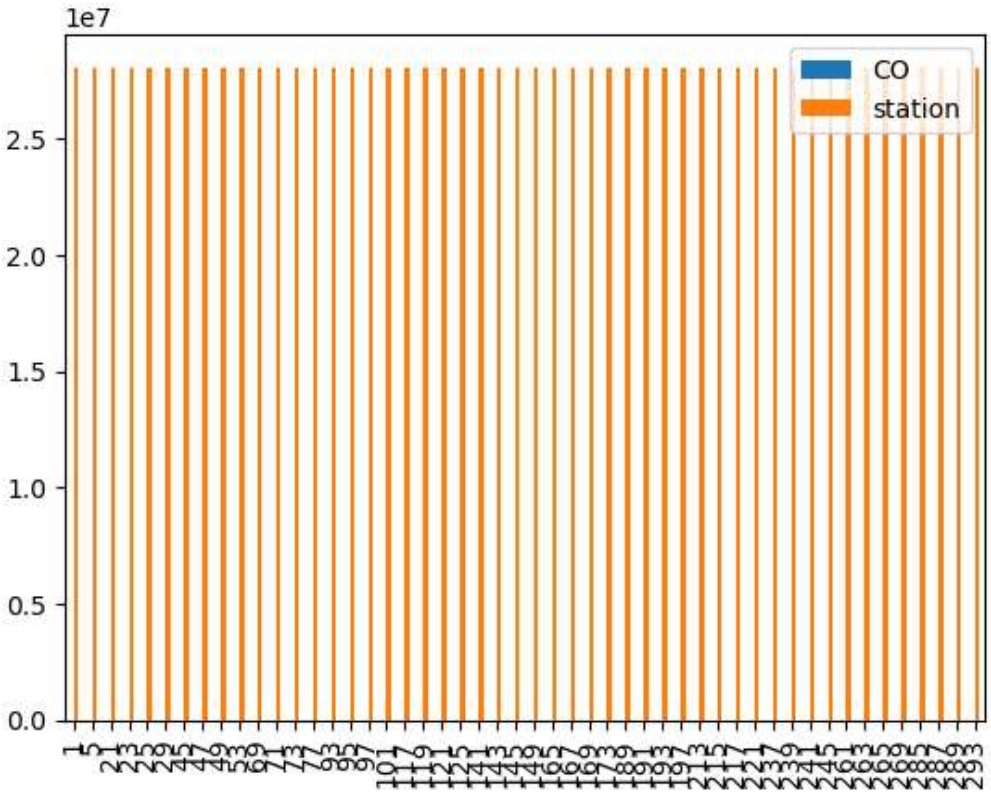
In [10]:
```
data.plot.line(subplots=True)
```

Out[10]:
```
array([<Axes: >, <Axes: >], dtype=object)
```



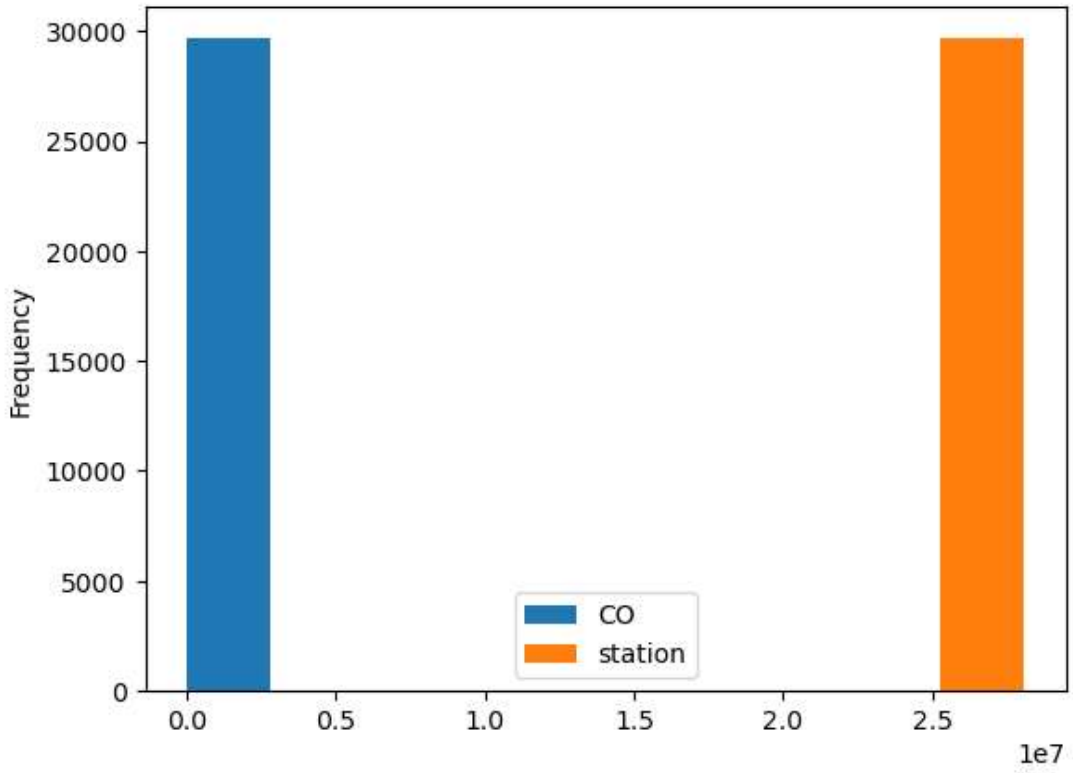
In [11]:
```
b=data[0:50]
b.plot.bar()
```

Out[11]:
```
<Axes: >
```



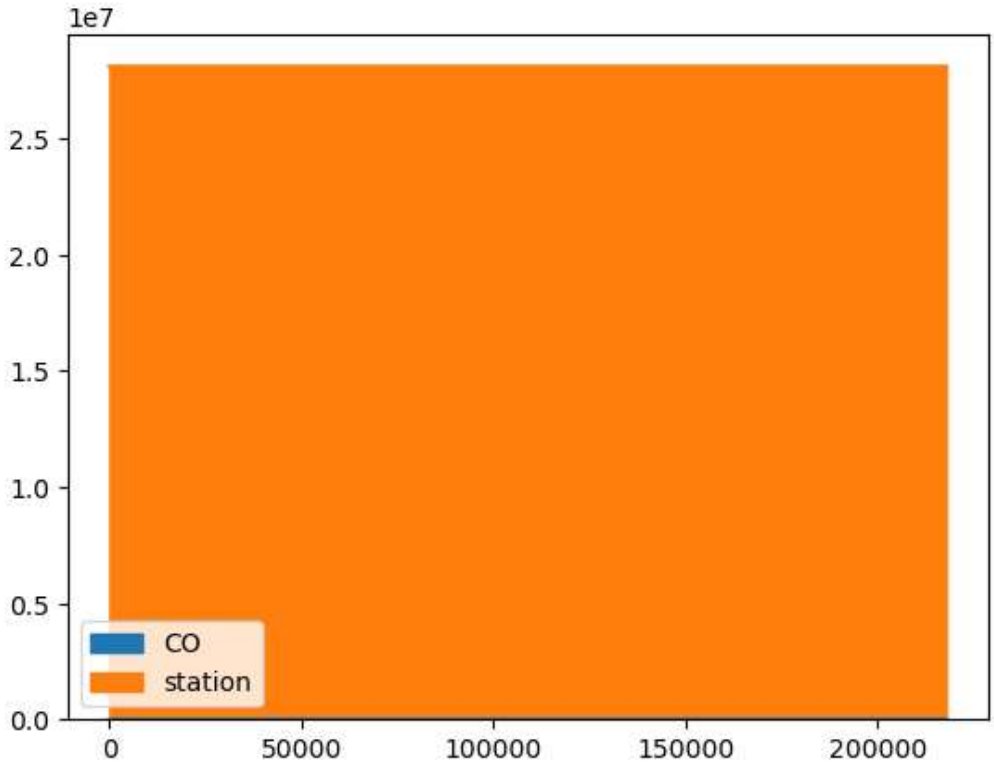
In [12]:
```
data.plot.hist()
```

Out[12]: <Axes: ylabel='Frequency'>



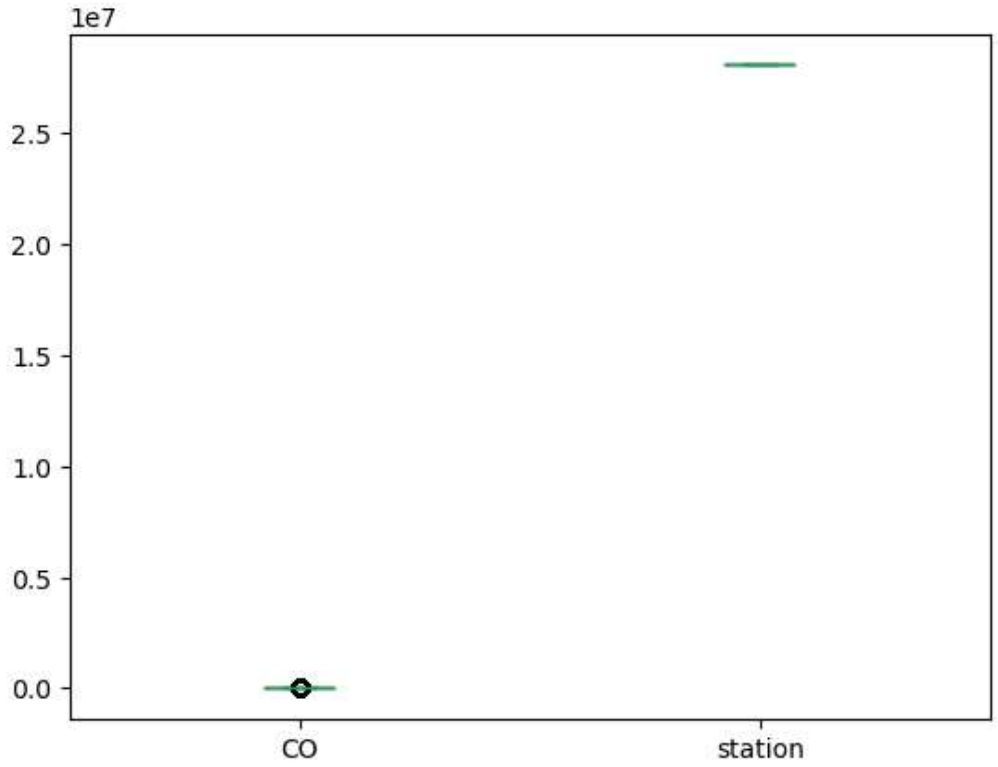
In [13]:
```
data.plot.area()
```

Out[13]: <Axes: >



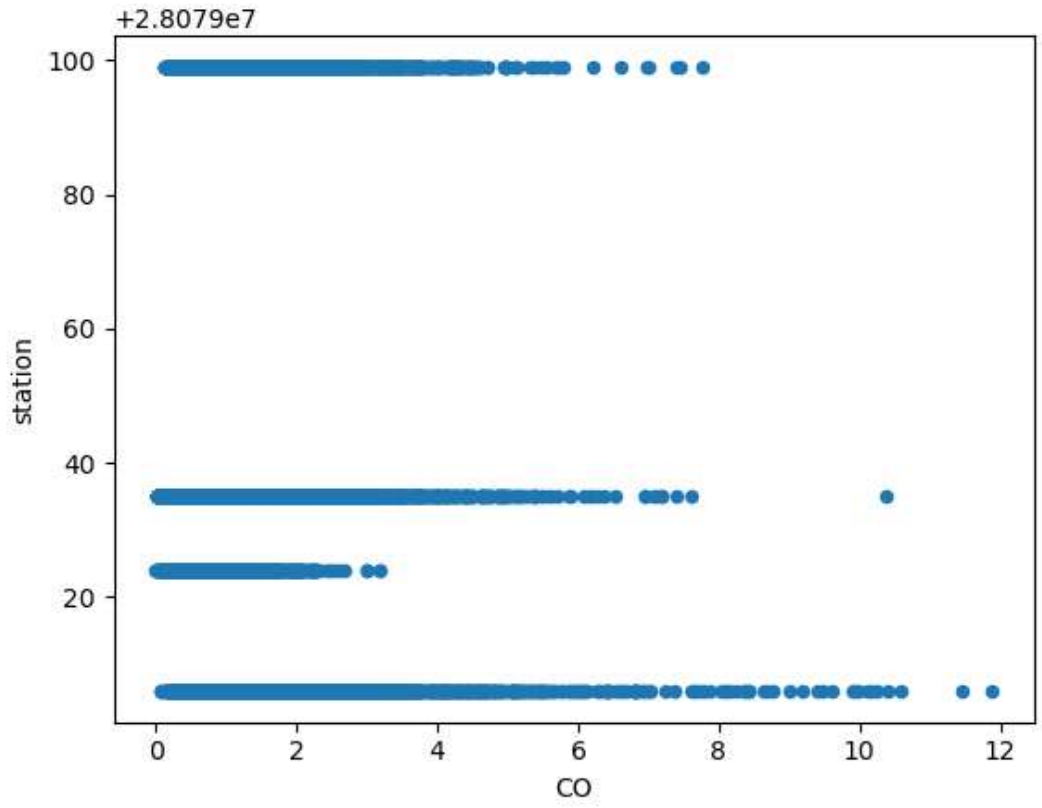
In [14]:
```
data.plot.box()
```

Out[14]: <Axes: >



In [16]:
```
data.plot.scatter(x='CO',y='station')
```

Out[16]: <Axes: xlabel='CO', ylabel='station'>



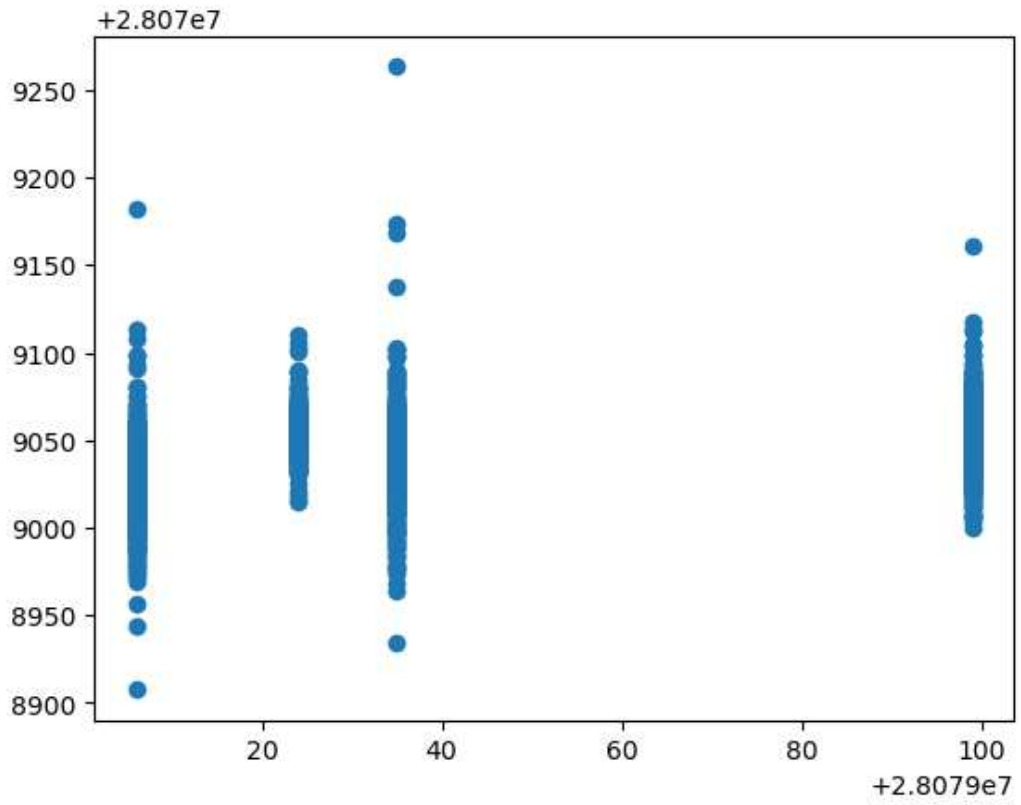
In [17]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [18]:
```
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 [20]:
```
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[20]: <matplotlib.collections.PathCollection at 0x21ce8b306d0>



In [22]:
```
print(lr.score(x_test,y_test))
print(lr.score(x_train,y_train))
```

```
0.16529626012112064
0.16441788984542627
```

Ridge and Lasso

In [24]:
```
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)
```

```
0.16504074776559996
0.16417700127652424
```

Out[24]:
```
Lasso
Lasso(alpha=10)
```

In [25]:
```
la.score(x_test,y_test)
```

Out[25]:
```
0.03790606534109997
```

ElasticNet

In [26]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[26]:
```
ElasticNet
ElasticNet()
```

In [27]:
```
en.coef_
```

Out[27]:
```
array([ 4.76903737,  0.        ,  0.74926708, -0.31698972,  0.08031853,
        0.06328982, -0.03190797, -2.3510525 , -0.03548704,  0.08078767,
        0.89069196, -0.33797515,  1.24648687, -0.67772164])
```

In [29]:
```
en.intercept_
```

In [30]:
```
prediction=en.predict(x_test)
```

In [31]:
```
en.score(x_test,y_test)
```

Out[31]:
```
0.10267841623775409
```

Evaluation Metrics

In [32]:
```
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)))
```

```
30.2880056318025
1206.9839620462847
34.74167471562482
```

Logistics Regression

In [35]:
```
from sklearn.linear_model import LogisticRegression
```

In [33]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [37]:
```
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[37]:
```
0.8087566146482861
```

In [39]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
logr.predict_proba(observation)
```

Out[39]:
```
array([[1.81867786e-43, 2.20093655e-56, 9.99998562e-01, 1.43829516e-06]])
```

Random Forest

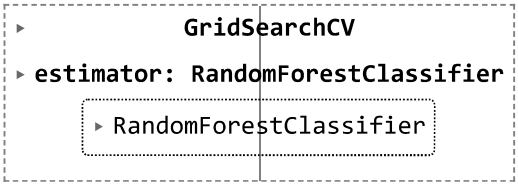
In [41]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[41]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [42]:
```
parameters={'max_depth':[1,2,3,4,5],
'min_samples_leaf':[5,10,15,20,25],
'n_estimators':[10,20,30,40,50]
}
```

In [43]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy")
grid_search.fit(x_train,y_train)
```

Out[43]:

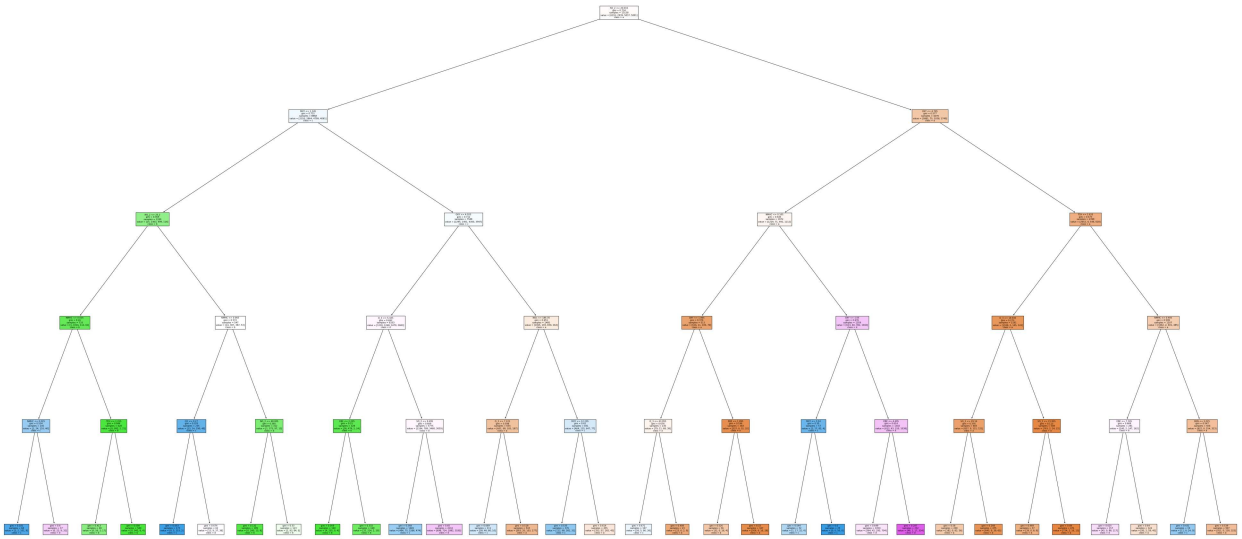


In [49]:
```
rfc_best=grid_search.best_estimator_
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','d'],
```

Out[49]:
```
[Text(0.5, 0.9166666666666666, 'SO_2 <= 20.015\ngini = 0.734\nsamples = 13138\nvalue =
[6191, 2939, 5837, 5801]\nclass = a'),
 Text(0.25, 0.75, 'MXY <= 1.345\ngini = 0.731\nsamples = 8868\nvalue = [2310, 2864, 479
9, 4061]\nclass = c'),
 Text(0.125, 0.5833333333333334, 'NO_2 <= 25.2\ngini = 0.459\nsamples = 1280\nvalue =
[15, 1403, 499, 114]\nclass = b'),
 Text(0.0625, 0.4166666666666667, 'NMHC <= 0.035\ngini = 0.26\nsamples = 733\nvalue =
[1, 1016, 112, 61]\nclass = b'),
 Text(0.03125, 0.25, 'NMHC <= 0.025\ngini = 0.534\nsamples = 105\nvalue = [1, 24, 110,
40]\nclass = c'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.181\nsamples = 68\nvalue = [1, 2, 101,
8]\nclass = c'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.6\nsamples = 37\nvalue = [0, 22, 9, 32]
\nclass = d'),
 Text(0.09375, 0.25, 'TCH <= 1.255\ngini = 0.044\nsamples = 628\nvalue = [0, 992, 2, 2
1]\nclass = b'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.413\nsamples = 48\nvalue = [0, 50, 2, 1
7]\nclass = b'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.008\nsamples = 580\nvalue = [0, 942, 0,
4]\nclass = b'),
 Text(0.1875, 0.4166666666666667, 'NMHC <= 0.065\ngini = 0.572\nsamples = 547\nvalue =
[14, 387, 387, 53]\nclass = b'),
 Text(0.15625, 0.25, 'CO <= 0.27\ngini = 0.325\nsamples = 235\nvalue = [13, 14, 290, 4
0]\nclass = c'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.053\nsamples = 173\nvalue = [0, 5, 253,
2]\nclass = c'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.674\nsamples = 62\nvalue = [13, 9, 37, 3
8]\nclass = d'),
 Text(0.21875, 0.25, 'NO_2 <= 48.005\ngini = 0.365\nsamples = 312\nvalue = [1, 373, 97,
13]\nclass = b'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.132\nsamples = 189\nvalue = [0, 280, 13,
8]\nclass = b'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.53\nsamples = 123\nvalue = [1, 93, 84,
5]\nclass = b'),
 Text(0.375, 0.5833333333333334, 'OXY <= 4.025\ngini = 0.712\nsamples = 7588\nvalue =
[2295, 1461, 4300, 3947]\nclass = c'),
 Text(0.3125, 0.4166666666666667, 'O_3 <= 5.435\ngini = 0.692\nsamples = 6103\nvalue =
[1230, 1268, 3470, 3683]\nclass = d'),
 Text(0.28125, 0.25, 'EBE <= 2.205\ngini = 0.21\nsamples = 328\nvalue = [36, 479, 2, 2
4]\nclass = b'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.059\nsamples = 160\nvalue = [4, 255, 0,
4]\nclass = b'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.332\nsamples = 168\nvalue = [32, 224, 2,
20]\nclass = b'),
 Text(0.34375, 0.25, 'SO_2 <= 8.495\ngini = 0.669\nsamples = 5775\nvalue = [1194, 789,
3468, 3659]\nclass = d'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.568\nsamples = 1665\nvalue = [496, 75, 1
586, 474]\nclass = c'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.65\nsamples = 4110\nvalue = [698, 714, 1
882, 3185]\nclass = d'),
 Text(0.4375, 0.4166666666666667, 'NOx <= 186.35\ngini = 0.651\nsamples = 1485\nvalue =
[1065, 193, 830, 264]\nclass = a'),
 Text(0.40625, 0.25, 'O_3 <= 7.225\ngini = 0.598\nsamples = 654\nvalue = [601, 68, 183,
187]\nclass = a'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.669\nsamples = 111\nvalue = [38, 48, 80,
10]\nclass = c'),
 Text(0.421875, 0.08333333333333333, 'gini = 0.518\nsamples = 543\nvalue = [563, 20, 10
3, 177]\nclass = a'),
 Text(0.46875, 0.25, 'MXY <= 12.205\ngini = 0.62\nsamples = 831\nvalue = [464, 125, 64
7, 77]\nclass = c'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.535\nsamples = 376\nvalue = [112, 68, 38
```

```
2, 32]\nclass = c'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.614\nsamples = 455\nvalue = [352, 57, 26
5, 45]\nclass = a'),
 Text(0.75, 0.75, 'OXY <= 4.795\ngini = 0.577\nsamples = 4270\nvalue = [3881, 75, 1038,
1740]\nclass = a'),
 Text(0.625, 0.5833333333333334, 'NMHC <= 0.105\ngini = 0.645\nsamples = 1872\nvalue =
[1229, 71, 492, 1112]\nclass = a'),
 Text(0.5625, 0.4166666666666667, 'OXY <= 1.865\ngini = 0.379\nsamples = 513\nvalue =
[616, 11, 100, 70]\nclass = a'),
 Text(0.53125, 0.25, 'O_3 <= 43.055\ngini = 0.676\nsamples = 130\nvalue = [79, 11, 68,
38]\nclass = a'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.671\nsamples = 107\nvalue = [56, 5, 66,
38]\nclass = c'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.408\nsamples = 23\nvalue = [23, 6, 2, 0]
\nclass = a'),
 Text(0.59375, 0.25, 'EBE <= 1.895\ngini = 0.196\nsamples = 383\nvalue = [537, 0, 32, 3
2]\nclass = a'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.526\nsamples = 36\nvalue = [31, 0, 16,
4]\nclass = a'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.15\nsamples = 347\nvalue = [506, 0, 16,
28]\nclass = a'),
 Text(0.6875, 0.4166666666666667, 'OXY <= 1.05\ngini = 0.635\nsamples = 1359\nvalue =
[613, 60, 392, 1042]\nclass = d'),
 Text(0.65625, 0.25, 'MXY <= 3.54\ngini = 0.35\nsamples = 57\nvalue = [1, 17, 82, 4]\nc
lass = c'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.544\nsamples = 32\nvalue = [1, 17, 32,
4]\nclass = c'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.0\nsamples = 25\nvalue = [0, 0, 50, 0]\n
class = c'),
 Text(0.71875, 0.25, 'EBE <= 4.405\ngini = 0.614\nsamples = 1302\nvalue = [612, 43, 31
0, 1038]\nclass = d'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.649\nsamples = 1050\nvalue = [564, 42, 2
93, 704]\nclass = d'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.287\nsamples = 252\nvalue = [48, 1, 17,
334]\nclass = d'),
 Text(0.875, 0.5833333333333334, 'TCH <= 1.625\ngini = 0.473\nsamples = 2398\nvalue =
[2652, 4, 546, 628]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'O_3 <= 18.945\ngini = 0.273\nsamples = 1191\nvalue =
[1590, 2, 145, 143]\nclass = a'),
 Text(0.78125, 0.25, 'SO_2 <= 29.14\ngini = 0.376\nsamples = 683\nvalue = [827, 0, 121,
121]\nclass = a'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.58\nsamples = 208\nvalue = [182, 0, 82,
56]\nclass = a'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.248\nsamples = 475\nvalue = [645, 0, 39,
65]\nclass = a'),
 Text(0.84375, 0.25, 'SO_2 <= 21.955\ngini = 0.113\nsamples = 508\nvalue = [763, 2, 24,
22]\nclass = a'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.469\nsamples = 37\nvalue = [39, 0, 8, 9]
\nclass = a'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.08\nsamples = 471\nvalue = [724, 2, 16,
13]\nclass = a'),
 Text(0.9375, 0.4166666666666667, 'NMHC <= 0.405\ngini = 0.599\nsamples = 1207\nvalue =
[1062, 2, 401, 485]\nclass = a'),
 Text(0.90625, 0.25, 'EBE <= 7.305\ngini = 0.668\nsamples = 281\nvalue = [140, 2, 147,
162]\nclass = d'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.625\nsamples = 157\nvalue = [45, 0, 88,
117]\nclass = d'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.64\nsamples = 124\nvalue = [95, 2, 59, 4
5]\nclass = a'),
 Text(0.96875, 0.25, 'BEN <= 5.025\ngini = 0.547\nsamples = 926\nvalue = [922, 0, 254,
323]\nclass = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.431\nsamples = 19\nvalue = [11, 0, 24,
```

```
0]\nclass = c'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.539\nsamples = 907\nvalue = [911, 0, 23
0, 323]\nclass = a')]
```



Conclusion

In [48]:
```
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: 0.16529626012112064
Ridge Regression: 0.16504074776559996
Lasso Regression 0.03790606534109997
ElasticNet Regression: 0.10267841623775409
Logistic Regression: 0.8087566146482861
Random Forest: 0.7303543913713406
```

Logistic Is Better!!!