20104169 - SUMESH R

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid_2012.csv")
df
```

```
Mounted at /content/drive
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	28079004
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	28079008
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	28079011
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210715	2012-03-01 00:00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN	28079056
210716	2012-03-01 00:00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN	28079057
210717	2012-03-01 00:00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN	28079058
210718	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN	28079059
210719	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN	28079060

210720 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:

```
df=df.dropna()
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

In [6]:

```
data=df[['CO' ,'station']]
data
```

Out[6]:

	CO	station
6	0.2	28079024
30	0.2	28079024
54	0.2	28079024
78	0.2	28079024
102	0.2	28079024
...	...	...
210654	0.3	28079024
210673	0.4	28079008
210678	0.3	28079024
210697	0.4	28079008
210702	0.3	28079024

10916 rows × 2 columns

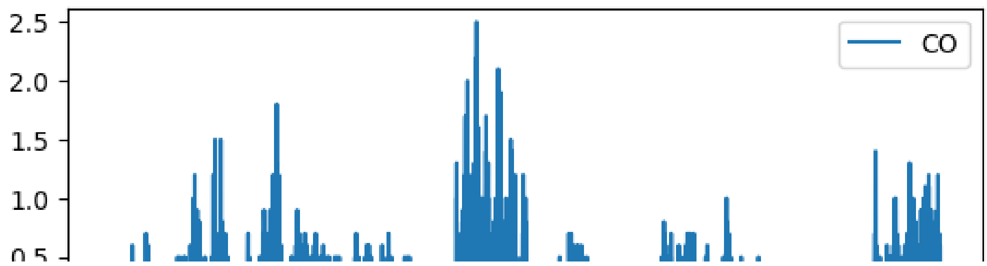
Line chart

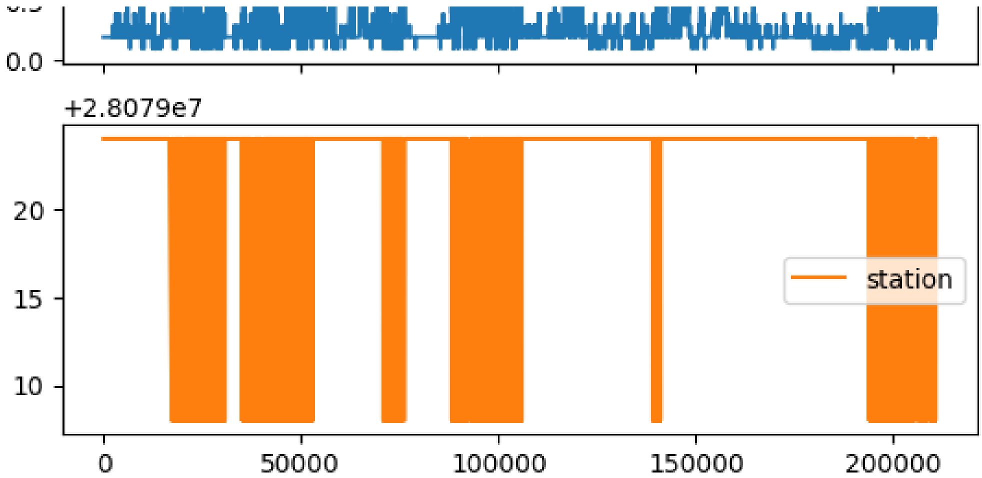
In [7]:

```
data.plot.line(subplots=True)
```

Out[7]:

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





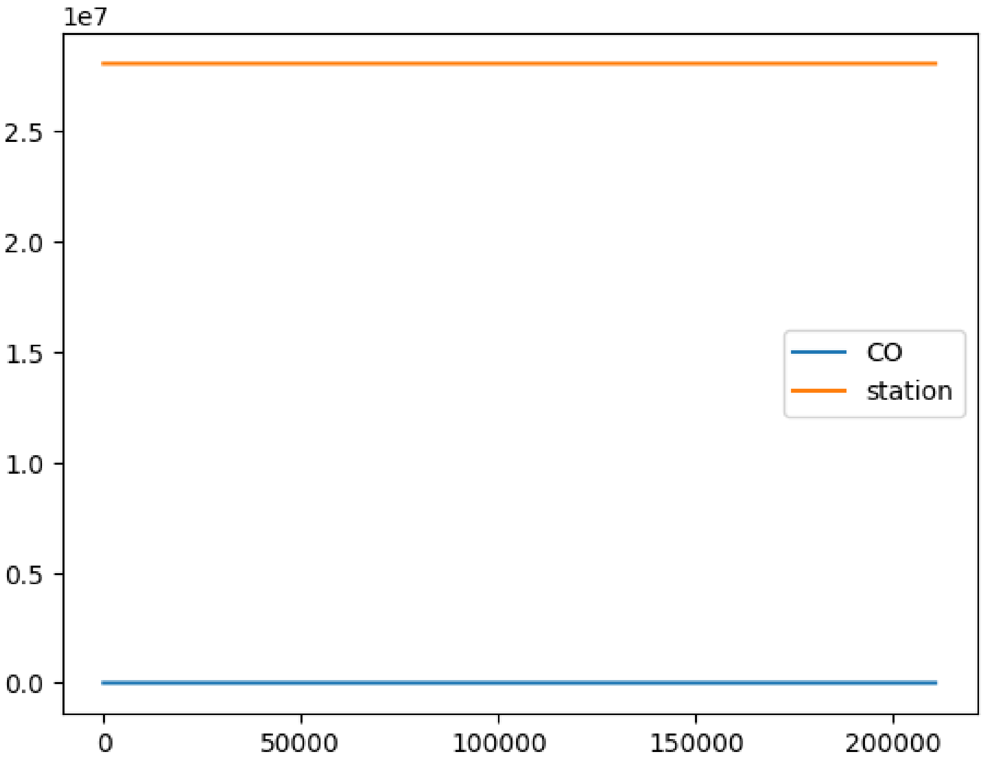
Line chart

In [8]:

```
data.plot.line()
```

Out[8]:

<Axes: >



Bar chart

In [9]:

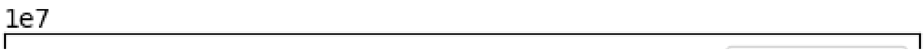
```
b=data[0:50]
```

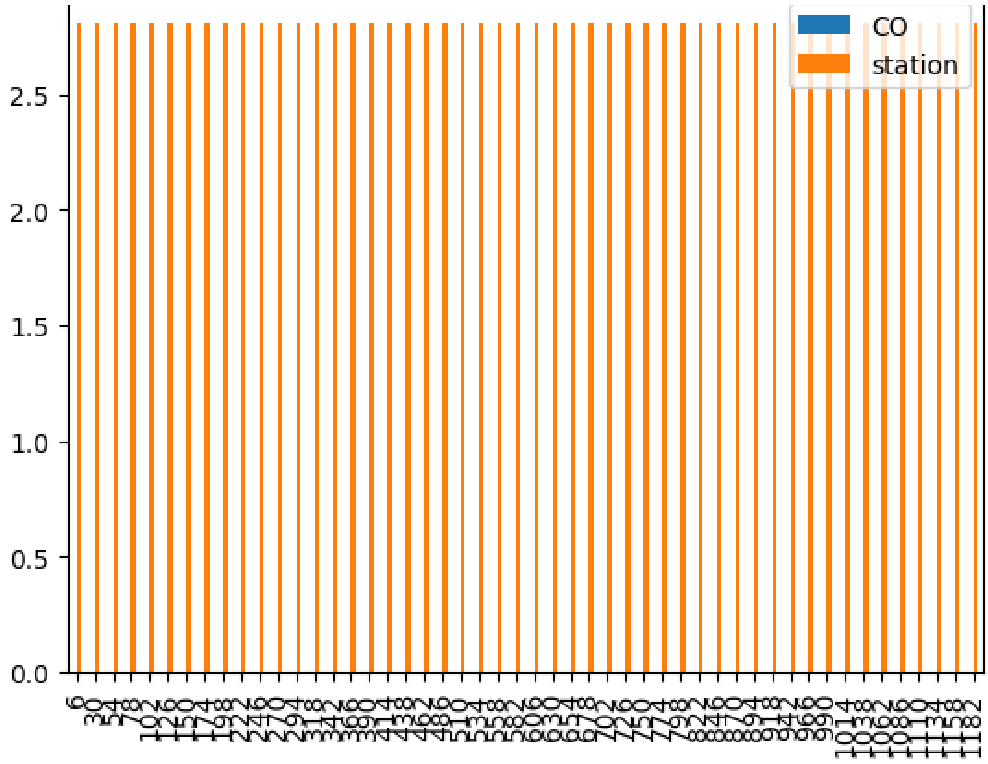
In [10]:

```
b.plot.bar()
```

Out[10]:

<Axes: >





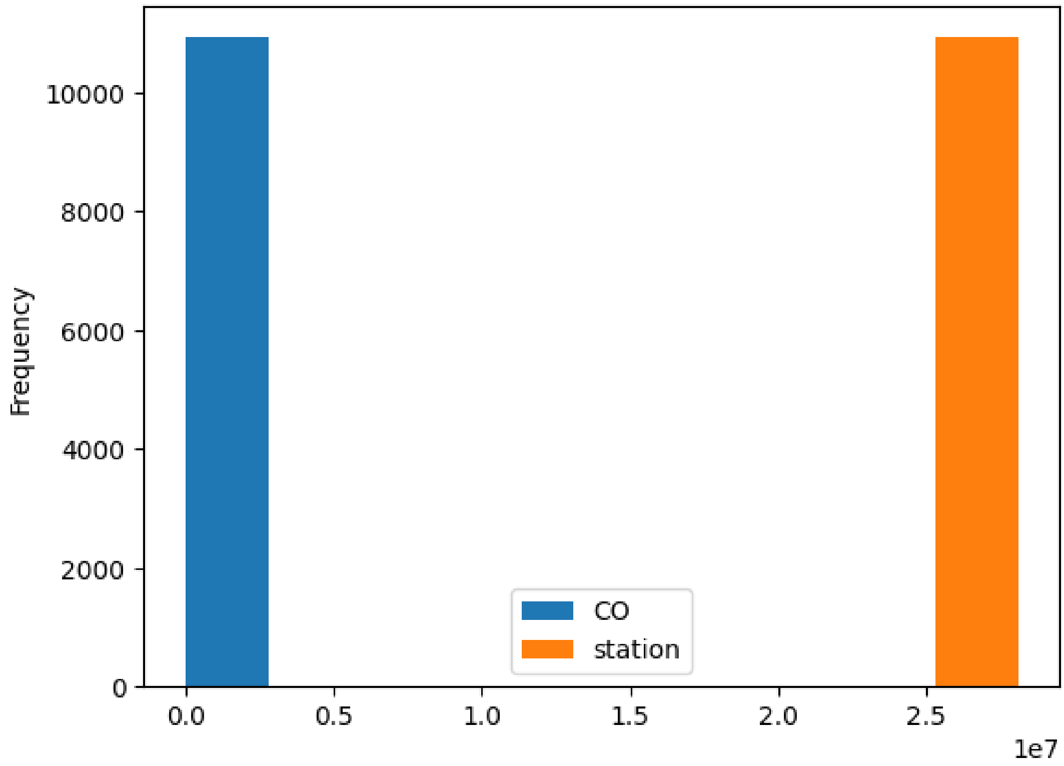
Histogram

In [11]:

```
data.plot.hist()
```

Out[11]:

```
<Axes: ylabel='Frequency'>
```



Area chart

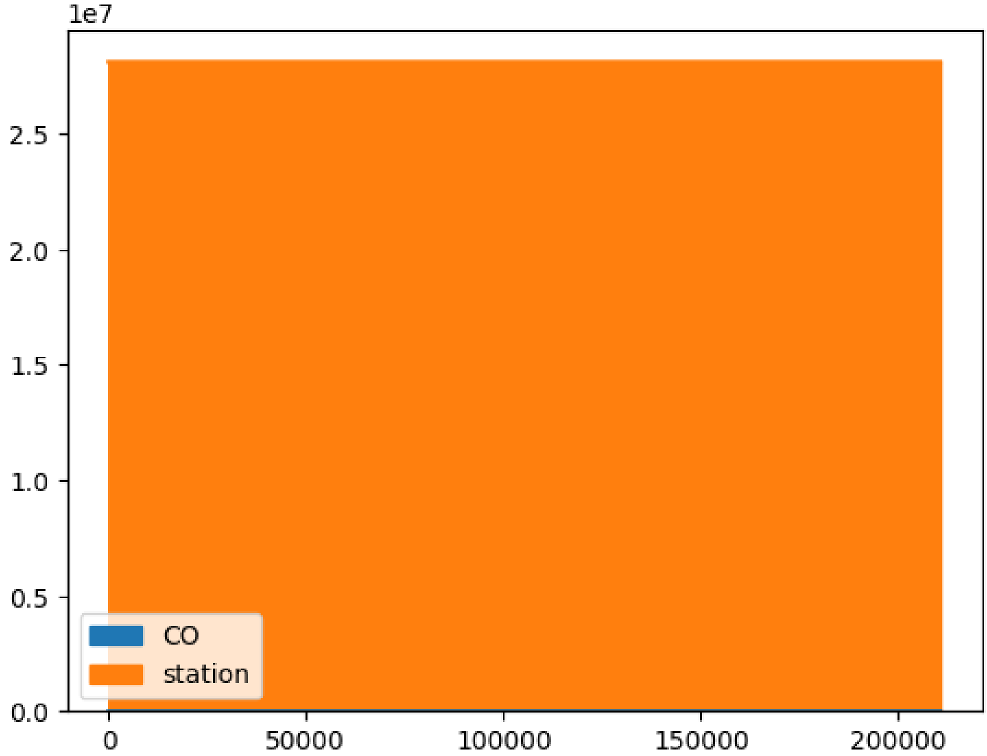
In [12]:

```
data.plot.area()
```

Out[12]:

Out[12]:

<Axes: >



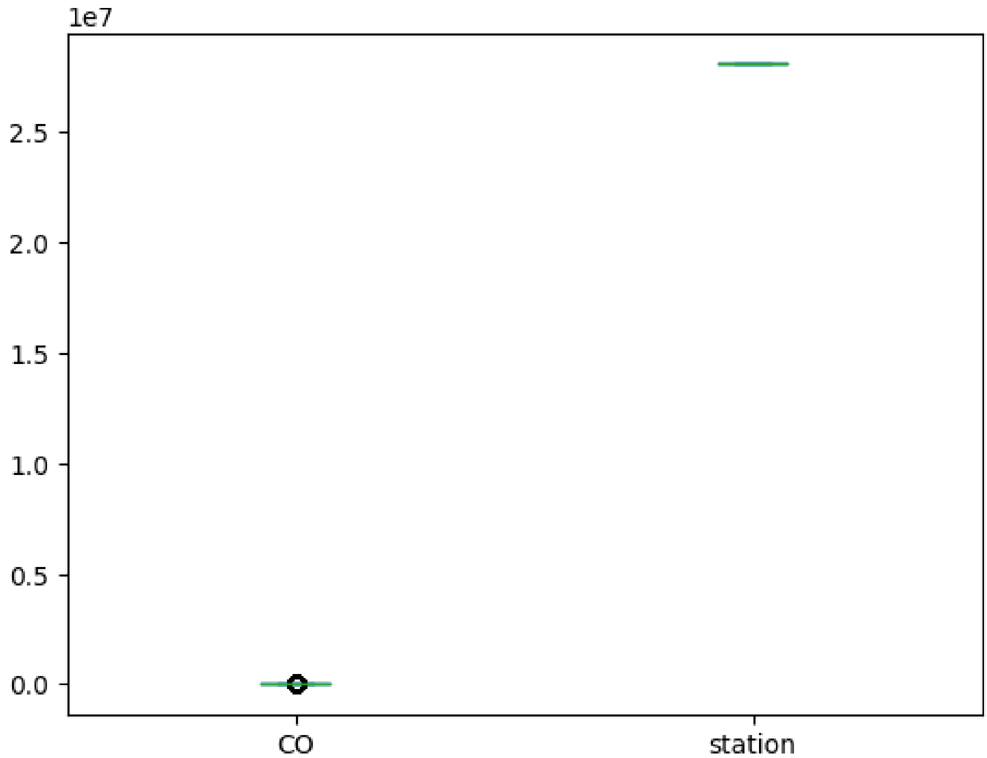
Box chart

In [13]:

```
data.plot.box()
```

Out[13]:

<Axes: >



Pie chart

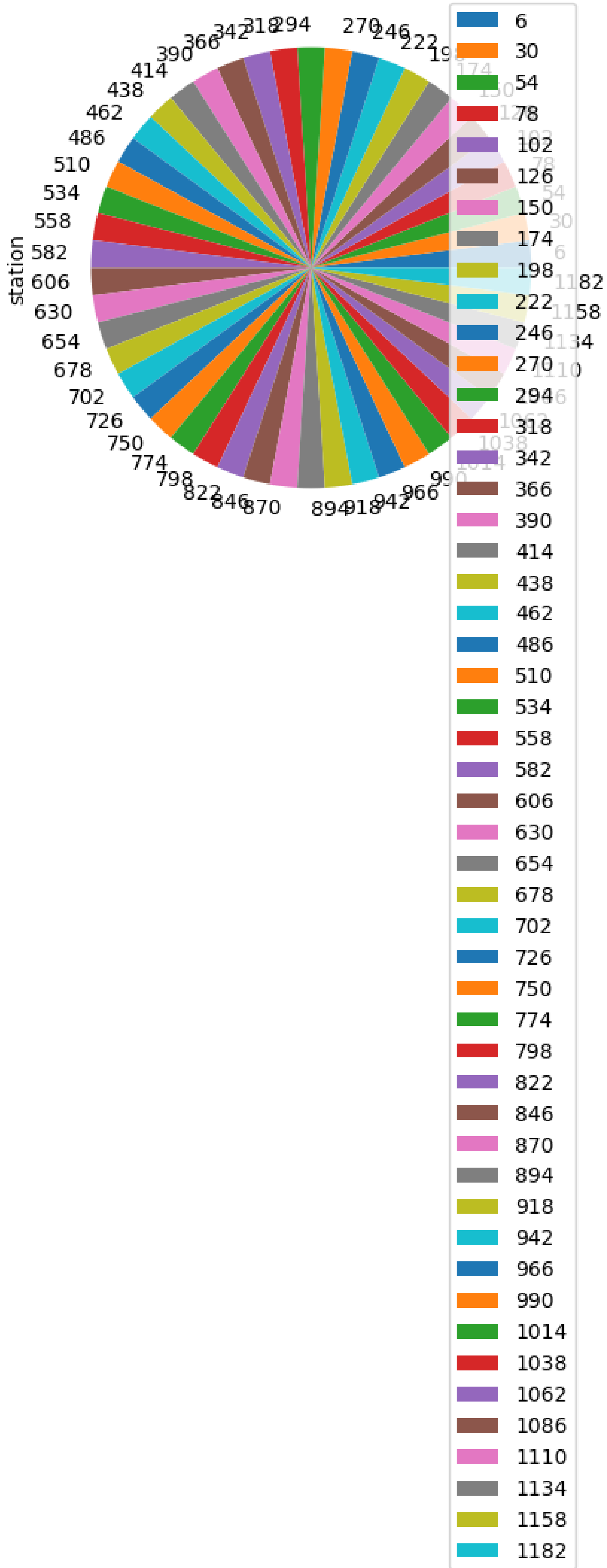
In [14]:

In [14]:

```
b.plot.pie(y='station' )
```

Out[14]:

```
<Axes: ylabel='station'>
```



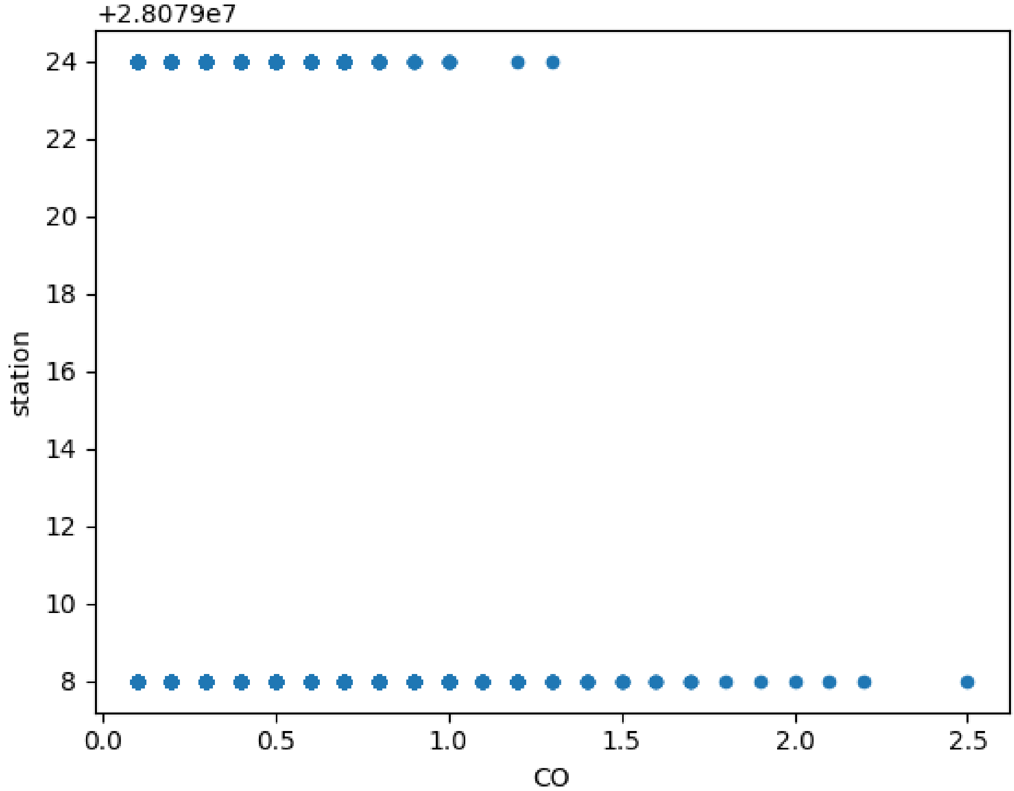
Scatter chart

In [15]:

```
data.plot.scatter(x='CO' ,y='station')
```

Out[15]:

```
<Axes: xlabel='CO', ylabel='station'>
```



In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	
count	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	109
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642	44.239557	22.875687	
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732	29.535560	22.266862	
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000	1.000000	1.000000	
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000	18.000000	10.000000	
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000	44.000000	17.000000	
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000	65.000000	28.000000	
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000	157.000000	267.000000	

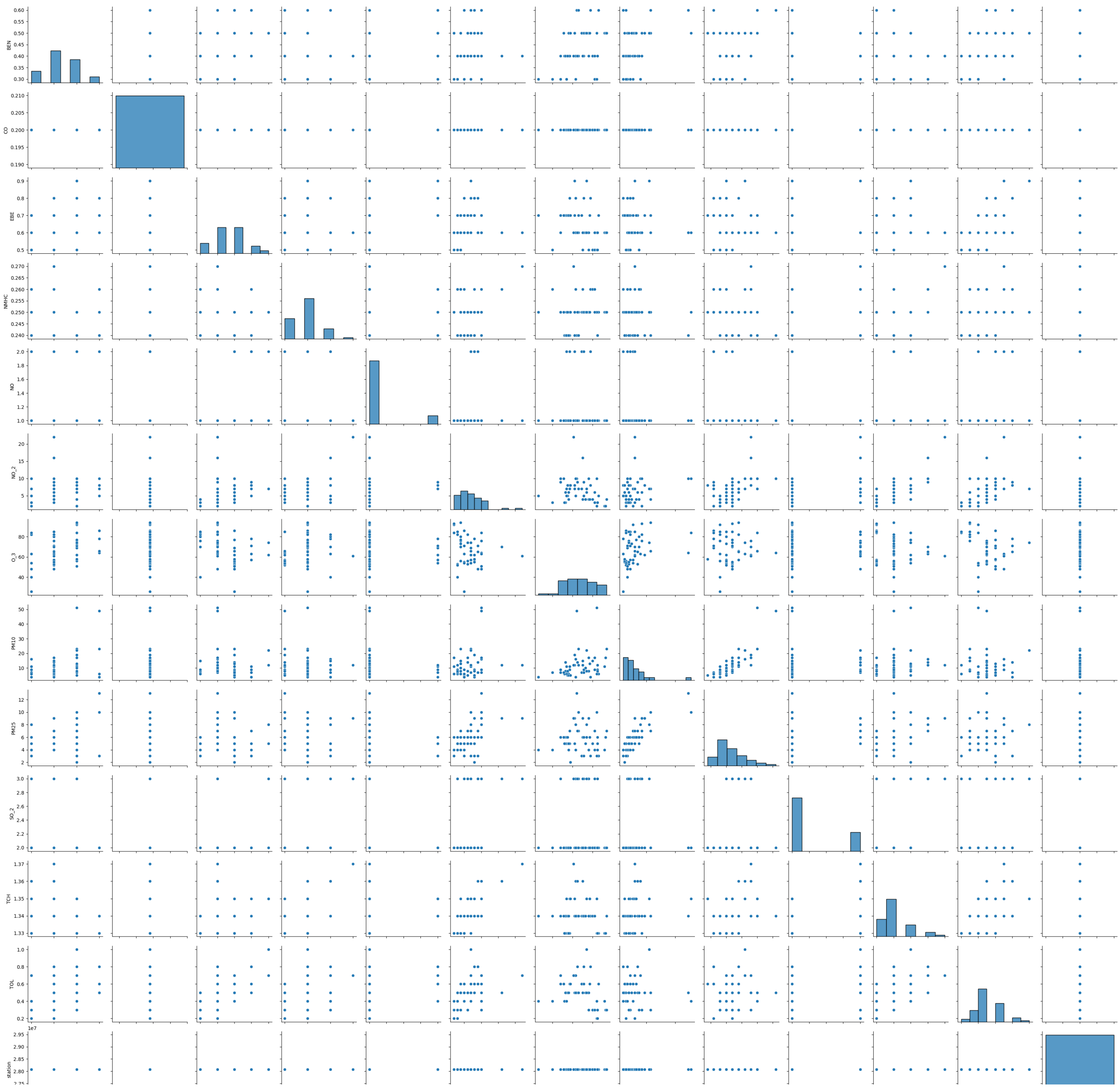
EDA AND VISUALIZATION

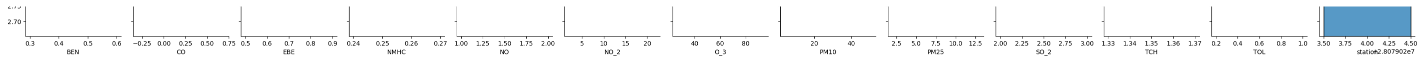
In [18]:

```
sns.pairplot(df[0:50])
```

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x7b55c19f1bd0>
```





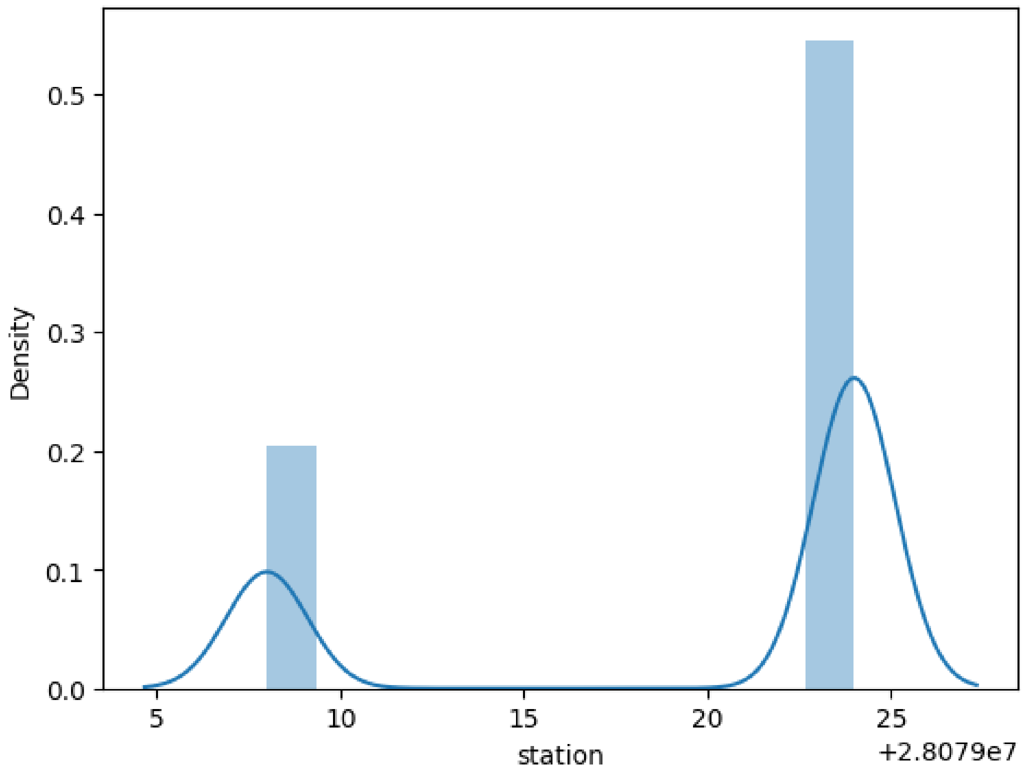
In [19]:

```
sns.distplot(df['station'])
```

```
<ipython-input-19-6e2460d4583e>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['station'])
```

Out[19]:

```
<Axes: xlabel='station', ylabel='Density'>
```



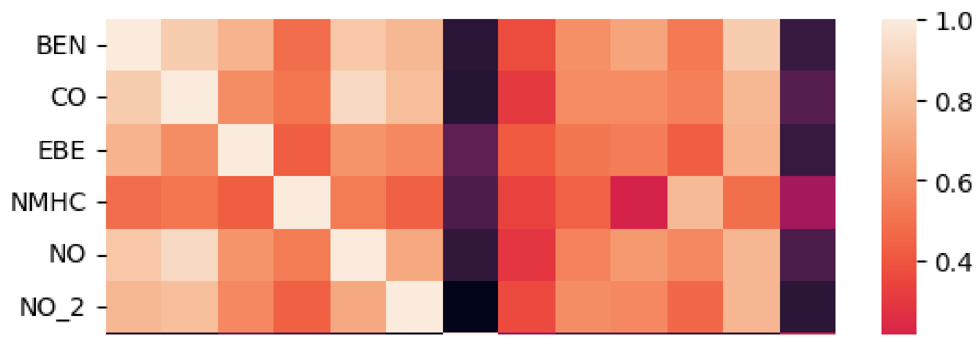
In [20]:

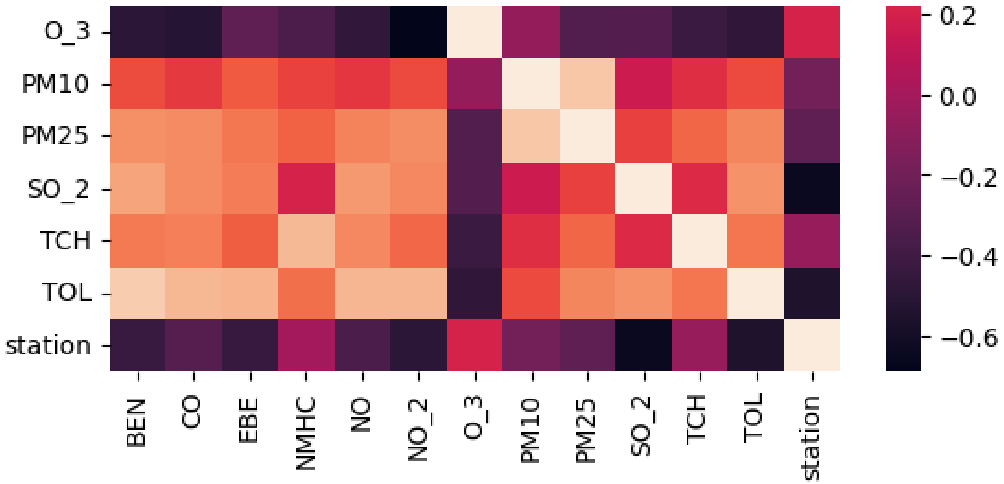
```
sns.heatmap(df.corr())
```

```
<ipython-input-20-aa4f4450a243>:1: FutureWarning: The default value of numeric_only in Da
taFrame.corr is deprecated. In a future version, it will default to False. Select only va
lid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr())
```

Out[20]:

```
<Axes: >
```





TO TRAIN THE MODEL AND MODEL BULDING

In [21]:

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

In [22]:

```
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 [23]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[23]:

```
▾ LinearRegression
LinearRegression()
```

In [24]:

```
lr.intercept_
```

Out[24]:

```
28079016.56357889
```

In [25]:

```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	4.128526
CO	22.656931
EBE	-0.355293
NMHC	16.665677
NO	-0.026850

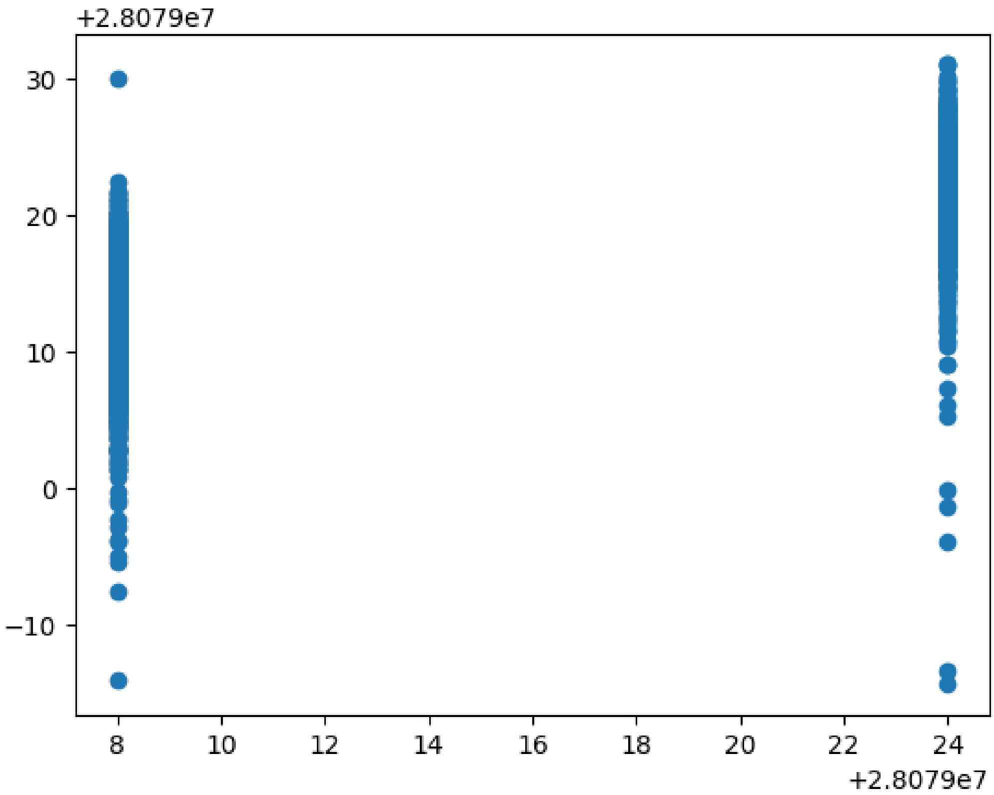
	Co-efficient
NO_2	-0.121328
O_3	-0.029489
PM10	0.005533
PM25	-0.059807
SO_2	-0.667808
TCH	2.347965
TOL	-1.517752

In [26]:

```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[26]:

```
<matplotlib.collections.PathCollection at 0x7b55b17b5450>
```



ACCURACY

In [27]:

```
lr.score(x_test,y_test)
```

Out[27]:

```
0.6178488329817193
```

In [28]:

```
lr.score(x_train,y_train)
```

Out[28]:

```
0.6265443111435998
```

Ridge and Lasso

In [29]:

```
from sklearn.linear_model import Ridge,Lasso
```

In [30]:

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[30]:

```
▾ Ridge
Ridge(alpha=10)
```

Accuracy(Ridge)

In [31]:

```
rr.score(x_test,y_test)
```

Out[31]:

```
0.6152190481161524
```

In [32]:

```
rr.score(x_train,y_train)
```

Out[32]:

```
0.6223236671405741
```

In [33]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[33]:

```
▾ Lasso
Lasso(alpha=10)
```

In [34]:

```
la.score(x_train,y_train)
```

Out[34]:

```
0.36094830764963526
```

Accuracy(Lasso)

In [35]:

```
la.score(x_test,y_test)
```

Out[35]:

```
0.3646755234090291
```

In [36]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[36]:

```
▾ ElasticNet
ElasticNet()
```

In [37]:

```
en.coef_
```

Out[37]:

```
array([ 0.        ,  0.        , -0.        ,  0.        ,  0.0666659 ,
       -0.08986738, -0.03785497,  0.        ,  0.        , -0.69996377,
        0.        , -0.69127665])
```

In [38]:

```
en.intercept_
```

Out[38]:

```
28079027.736731425
```

In [39]:

```
prediction=en.predict(x_test)
```

In [40]:

```
en.score(x_test,y_test)
```

Out[40]:

```
0.5470783923837095
```

Evaluation Metrics

In [41]:

```
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)))
```

```
3.438876448474995
23.043203144300286
4.800333649268588
```

Logistic Regression

In [42]:

```
from sklearn.linear_model import LogisticRegression
```

In [43]:

```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [44]:

```
feature_matrix.shape
```

Out[44]:

```
(10916, 12)
```

In [45]:

```
target_vector.shape
```

Out[45]:

```
(10916,)
```

In [46]:

```
from sklearn.preprocessing import StandardScaler
```

In [47]:

```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:

```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[48]:

```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

In [49]:

```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12]]
```

In [50]:

```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [51]:

```
logr.classes_
```

Out[51]:

```
array([28079008, 28079024])
```

In [52]:

```
logr.score(fs,target_vector)
```

Out[52]:

```
0.9311102968120191
```

In [53]:

```
logr.predict_proba(observation)[0][0]
```

Out[53]:

```
1.0
```

In [54]:

```
logr.predict_proba(observation)
```

Out[54]:

```
array([[1.00000000e+00, 7.14919073e-33]])
```

Random Forest

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:

```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[56]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [57]:

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

In [58]:

```
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[58]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
▸ RandomForestClassifier
```

In [59]:

```
grid_search.best_score_
```

Out[59]:

```
0.966496771081828
```

In [60]:

```
rfc_best=grid_search.best_estimator_
```

In [61]:

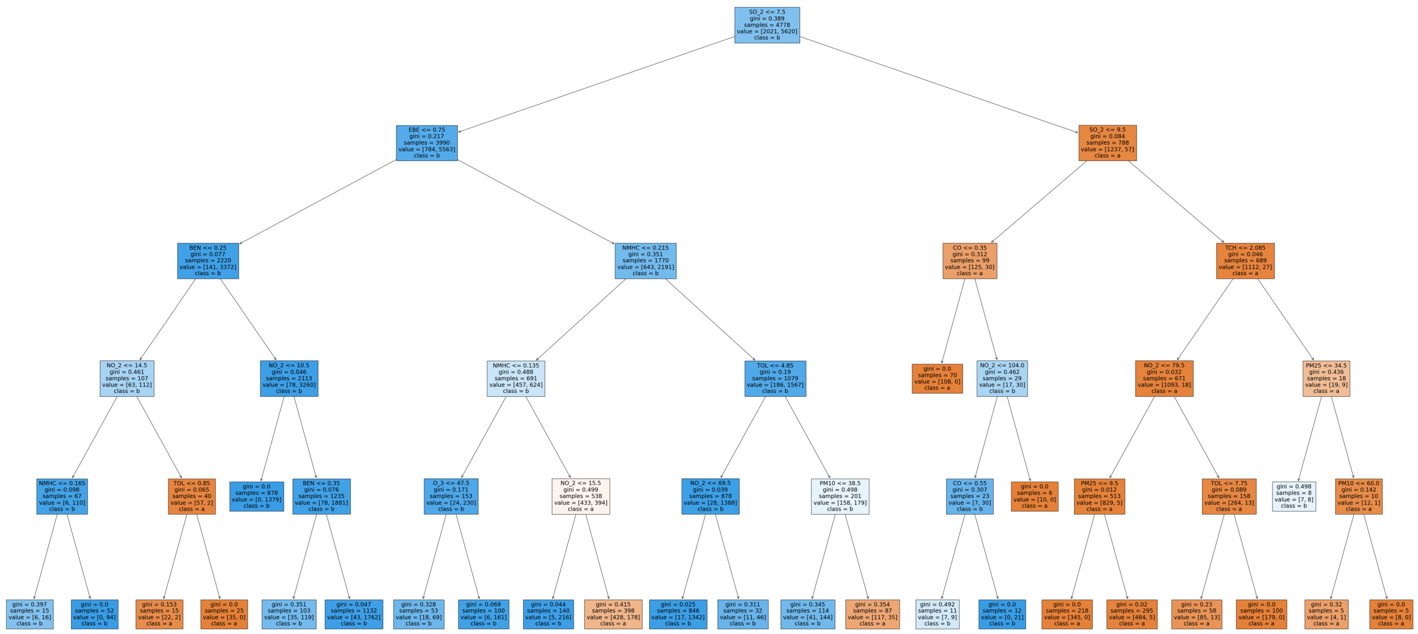
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','d'],f
illed=True)
```

Out[61]:

```
[Text(0.5397727272727273, 0.9166666666666666, 'SO_2 <= 7.5\ngini = 0.389\nsamples = 4778\
nvalue = [2021, 5620]\nclass = b'),
 Text(0.30113636363636365, 0.75, 'EBE <= 0.75\ngini = 0.217\nsamples = 3990\nvalue = [784
, 5563]\nclass = b'),
 Text(0.14772727272727273, 0.5833333333333334, 'BEN <= 0.25\ngini = 0.077\nsamples = 2220
\nvalue = [141, 3372]\nclass = b'),
 Text(0.09090909090909091, 0.4166666666666667, 'NO_2 <= 14.5\ngini = 0.461\nsamples = 107
\nvalue = [63, 112]\nclass = b'),
 Text(0.045454545454545456, 0.25, 'NMHC <= 0.165\ngini = 0.098\nsamples = 67\nvalue = [6,
110]\nclass = b'),
 Text(0.022727272727272728, 0.08333333333333333, 'gini = 0.397\nsamples = 15\nvalue = [6,
16]\nclass = b'),
 Text(0.06818181818181818, 0.08333333333333333, 'gini = 0.0\nsamples = 52\nvalue = [0, 94
]\nclass = b'),
```

```
 Text(0.13636363636363635, 0.25, 'TOL <= 0.85\ngini = 0.065\nsamples = 40\nvalue = [57, 2
]\nclass = a'),
 Text(0.11363636363636363, 0.08333333333333333, 'gini = 0.153\nsamples = 15\nvalue = [22,
2]\nclass = a'),
 Text(0.1590909090909091, 0.08333333333333333, 'gini = 0.0\nsamples = 25\nvalue = [35, 0]
\nclass = a'),
 Text(0.20454545454545456, 0.4166666666666667, 'NO_2 <= 10.5\ngini = 0.046\nsamples = 211
3\nvalue = [78, 3260]\nclass = b'),
 Text(0.18181818181818182, 0.25, 'gini = 0.0\nsamples = 878\nvalue = [0, 1379]\nclass = b
'),
 Text(0.22727272727272727, 0.25, 'BEN <= 0.35\ngini = 0.076\nsamples = 1235\nvalue = [78,
1881]\nclass = b'),
 Text(0.20454545454545456, 0.08333333333333333, 'gini = 0.351\nsamples = 103\nvalue = [35
, 119]\nclass = b'),
 Text(0.25, 0.08333333333333333, 'gini = 0.047\nsamples = 1132\nvalue = [43, 1762]\nclass
= b'),
 Text(0.45454545454545453, 0.5833333333333334, 'NMHC <= 0.215\ngini = 0.351\nsamples = 17
70\nvalue = [643, 2191]\nclass = b'),
 Text(0.36363636363636365, 0.4166666666666667, 'NMHC <= 0.135\ngini = 0.488\nsamples = 69
1\nvalue = [457, 624]\nclass = b'),
 Text(0.3181818181818182, 0.25, 'O_3 <= 47.5\ngini = 0.171\nsamples = 153\nvalue = [24, 2
30]\nclass = b'),
 Text(0.29545454545454547, 0.08333333333333333, 'gini = 0.328\nsamples = 53\nvalue = [18,
69]\nclass = b'),
 Text(0.3409090909090909, 0.08333333333333333, 'gini = 0.069\nsamples = 100\nvalue = [6,
161]\nclass = b'),
 Text(0.4090909090909091, 0.25, 'NO_2 <= 15.5\ngini = 0.499\nsamples = 538\nvalue = [433,
394]\nclass = a'),
 Text(0.38636363636363635, 0.08333333333333333, 'gini = 0.044\nsamples = 140\nvalue = [5,
216]\nclass = b'),
 Text(0.4318181818181818, 0.08333333333333333, 'gini = 0.415\nsamples = 398\nvalue = [428
, 178]\nclass = a'),
 Text(0.5454545454545454, 0.4166666666666667, 'TOL <= 4.85\ngini = 0.19\nsamples = 1079\n
value = [186, 1567]\nclass = b'),
 Text(0.5, 0.25, 'NO_2 <= 69.5\ngini = 0.039\nsamples = 878\nvalue = [28, 1388]\nclass =
b'),
 Text(0.4772727272727273, 0.08333333333333333, 'gini = 0.025\nsamples = 846\nvalue = [17,
1342]\nclass = b'),
 Text(0.5227272727272727, 0.08333333333333333, 'gini = 0.311\nsamples = 32\nvalue = [11,
46]\nclass = b'),
 Text(0.5909090909090909, 0.25, 'PM10 <= 38.5\ngini = 0.498\nsamples = 201\nvalue = [158,
179]\nclass = b'),
 Text(0.5681818181818182, 0.08333333333333333, 'gini = 0.345\nsamples = 114\nvalue = [41,
144]\nclass = b'),
 Text(0.6136363636363636, 0.08333333333333333, 'gini = 0.354\nsamples = 87\nvalue = [117,
35]\nclass = a'),
 Text(0.7784090909090909, 0.75, 'SO_2 <= 9.5\ngini = 0.084\nsamples = 788\nvalue = [1237,
57]\nclass = a'),
 Text(0.6818181818181818, 0.5833333333333334, 'CO <= 0.35\ngini = 0.312\nsamples = 99\nva
lue = [125, 30]\nclass = a'),
 Text(0.6590909090909091, 0.4166666666666667, 'gini = 0.0\nsamples = 70\nvalue = [108, 0]
\nclass = a'),
 Text(0.7045454545454546, 0.4166666666666667, 'NO_2 <= 104.0\ngini = 0.462\nsamples = 29\
nvalue = [17, 30]\nclass = b'),
 Text(0.6818181818181818, 0.25, 'CO <= 0.55\ngini = 0.307\nsamples = 23\nvalue = [7, 30]\
nclass = b'),
 Text(0.6590909090909091, 0.08333333333333333, 'gini = 0.492\nsamples = 11\nvalue = [7, 9
]\nclass = b'),
 Text(0.7045454545454546, 0.08333333333333333, 'gini = 0.0\nsamples = 12\nvalue = [0, 21]
\nclass = b'),
 Text(0.7272727272727273, 0.25, 'gini = 0.0\nsamples = 6\nvalue = [10, 0]\nclass = a'),
 Text(0.875, 0.5833333333333334, 'TCH <= 2.085\ngini = 0.046\nsamples = 689\nvalue = [111
2, 27]\nclass = a'),
 Text(0.8181818181818182, 0.4166666666666667, 'NO_2 <= 79.5\ngini = 0.032\nsamples = 671\
nvalue = [1093, 18]\nclass = a'),
 Text(0.7727272727272727, 0.25, 'PM25 <= 9.5\ngini = 0.012\nsamples = 513\nvalue = [829,
5]\nclass = a'),
 Text(0.75, 0.08333333333333333, 'gini = 0.0\nsamples = 218\nvalue = [345, 0]\nclass = a'
),
 Text(0.7954545454545454, 0.08333333333333333, 'gini = 0.02\nsamples = 295\nvalue = [484,
5]\nclass = a'),
 Text(0.8636363636363636, 0.25, 'TOL <= 7.75\ngini = 0.089\nsamples = 158\nvalue = [264,
```

```
13]\nclass = a'),
 Text(0.8409090909090909, 0.08333333333333333, 'gini = 0.23\nsamples = 58\nvalue = [85, 1
3]\nclass = a'),
 Text(0.8863636363636364, 0.08333333333333333, 'gini = 0.0\nsamples = 100\nvalue = [179,
0]\nclass = a'),
 Text(0.9318181818181818, 0.4166666666666667, 'PM25 <= 34.5\ngini = 0.436\nsamples = 18\n
value = [19, 9]\nclass = a'),
 Text(0.9090909090909091, 0.25, 'gini = 0.498\nsamples = 8\nvalue = [7, 8]\nclass = b'),
 Text(0.9545454545454546, 0.25, 'PM10 <= 60.0\ngini = 0.142\nsamples = 10\nvalue = [12, 1
]\nclass = a'),
 Text(0.9318181818181818, 0.08333333333333333, 'gini = 0.32\nsamples = 5\nvalue = [4, 1]\
nclass = a'),
 Text(0.9772727272727273, 0.08333333333333333, 'gini = 0.0\nsamples = 5\nvalue = [8, 0]\n
class = a')]
```



Conclusion

Accuracy

In [62]:

```
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: 0.6178488329817193
Ridge Regression: 0.6152190481161524
Lasso Regression 0.3646755234090291
ElasticNet Regression: 0.5470783923837095
Logistic Regression: 0.9311102968120191
Random Forest: 0.966496771081828
```

Random Forest is suitable for this dataset