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_2011.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	2011-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN	28079004
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7	28079008
2	2011-11-01 01:00:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2	28079011
3	2011-11-01 01:00:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2011-11-01 01:00:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209923	2011-09-01 00:00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN	28079056
209924	2011-09-01 00:00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN	28079057
209925	2011-09-01 00:00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN	28079058
209926	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN	28079059
209927	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN	28079060

209928 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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [6]:

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

Out[6]:

	CO	station
1	0.4	28079008
6	0.3	28079024
25	0.3	28079008
30	0.4	28079024
49	0.2	28079008
...	...	...
209862	0.1	28079024
209881	0.1	28079008
209886	0.1	28079024
209905	0.1	28079008
209910	0.1	28079024

16460 rows × 2 columns

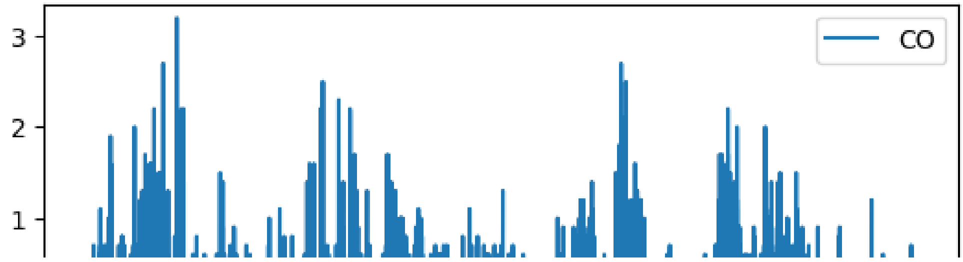
Line chart

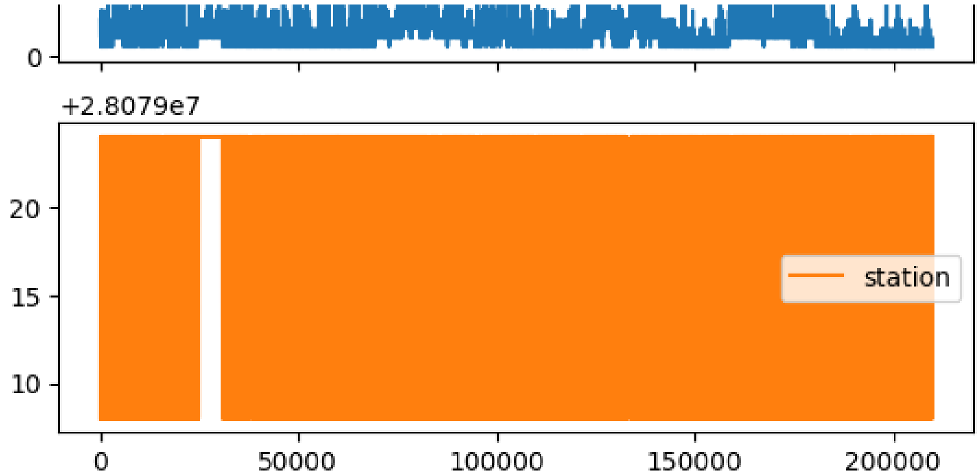
In [7]:

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

Out[7]:

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





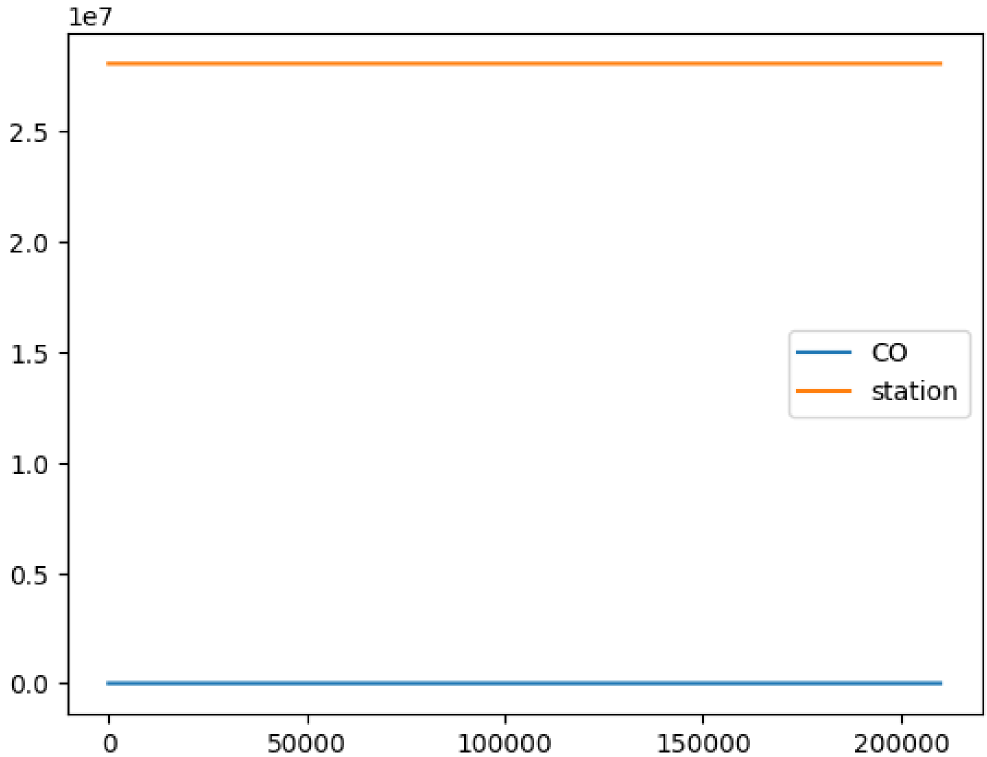
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

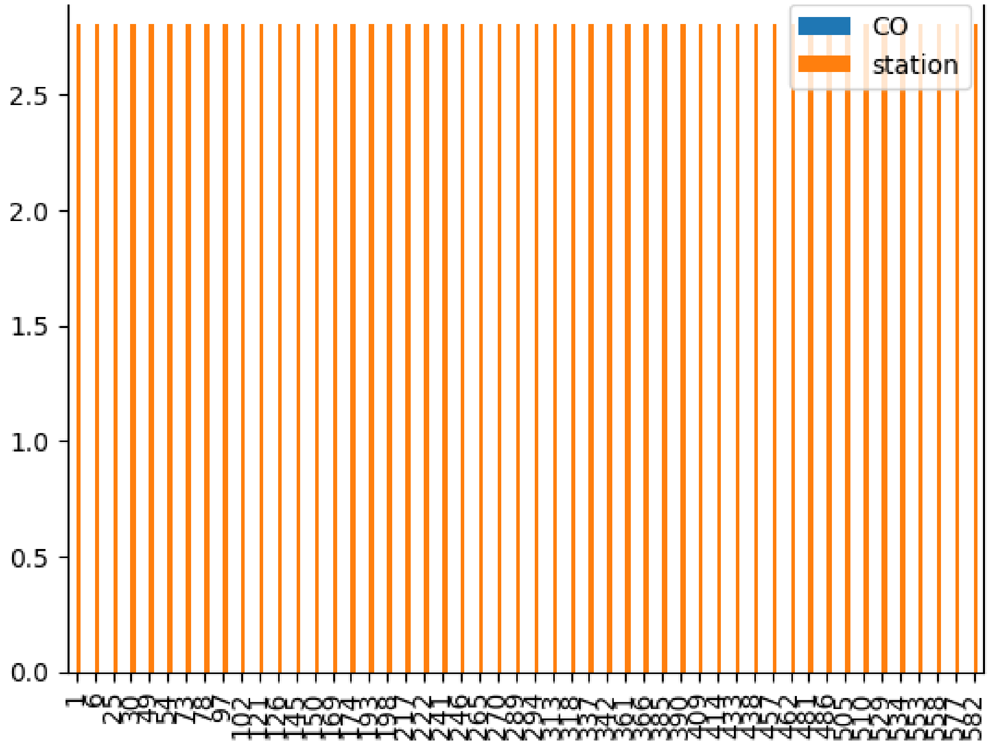
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



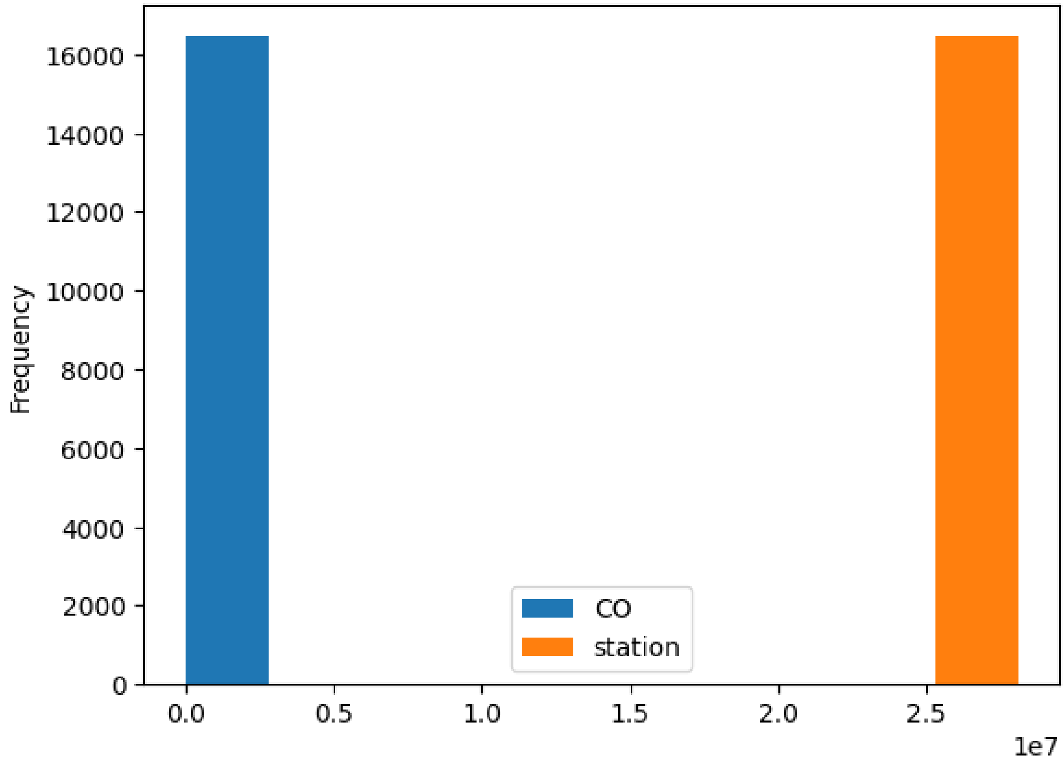
Histogram

In [11]:

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

Out[11]:

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



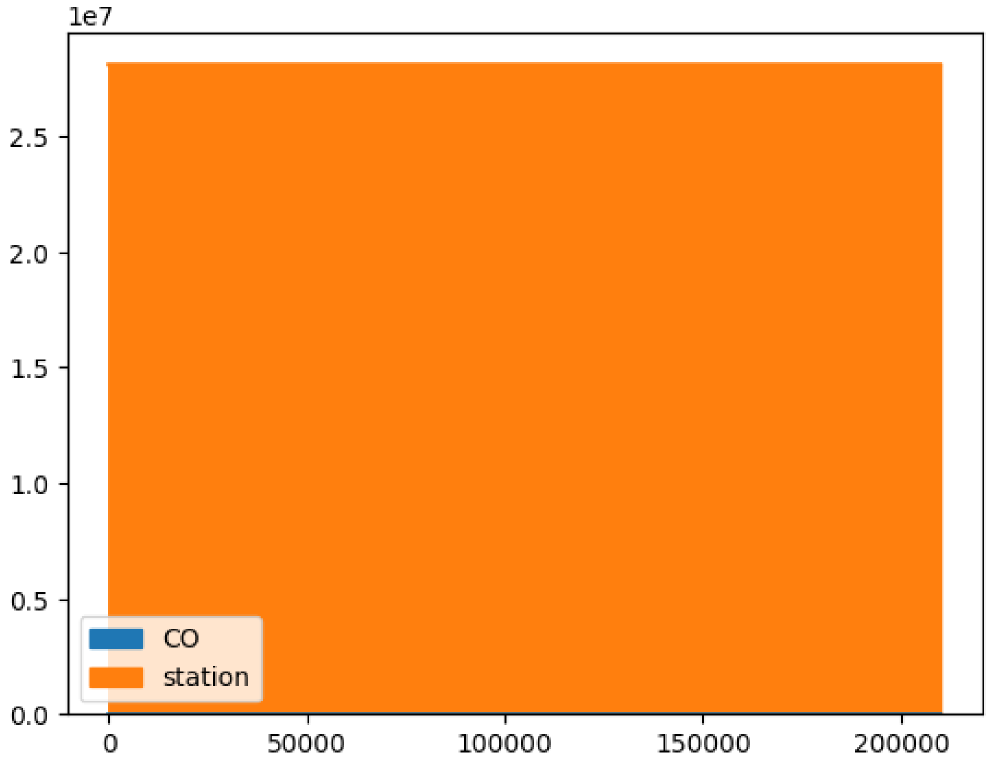
Area chart

In [12]:

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

Out[12]:

<Axes: >



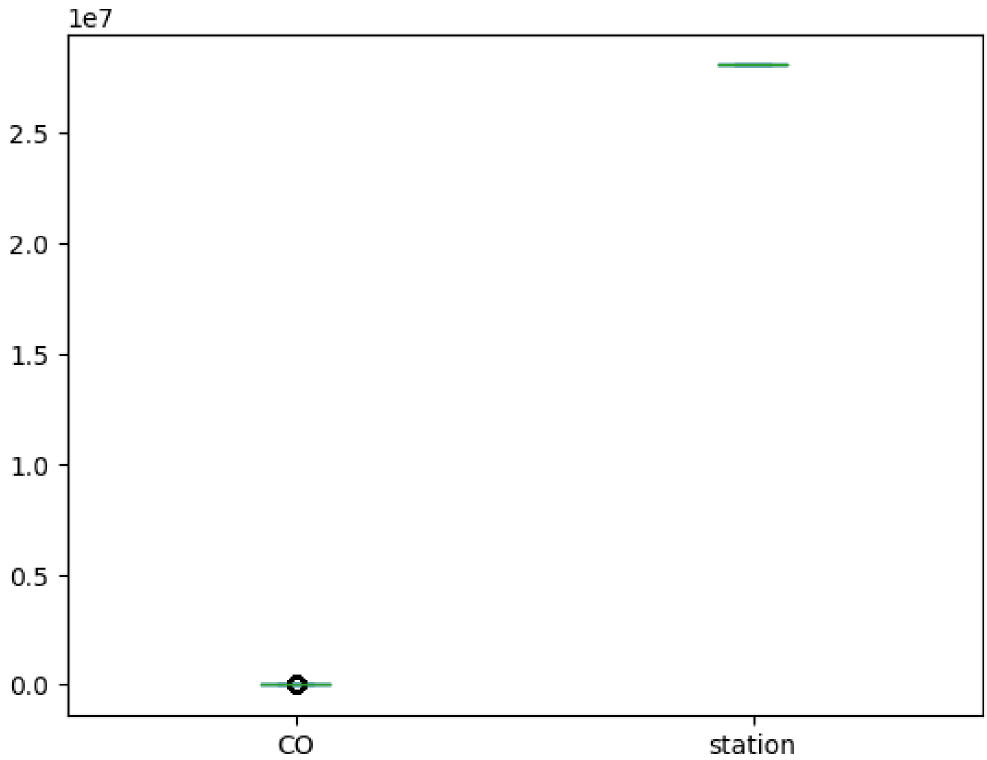
Box chart

In [13]:

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

Out[13]:

<Axes: >



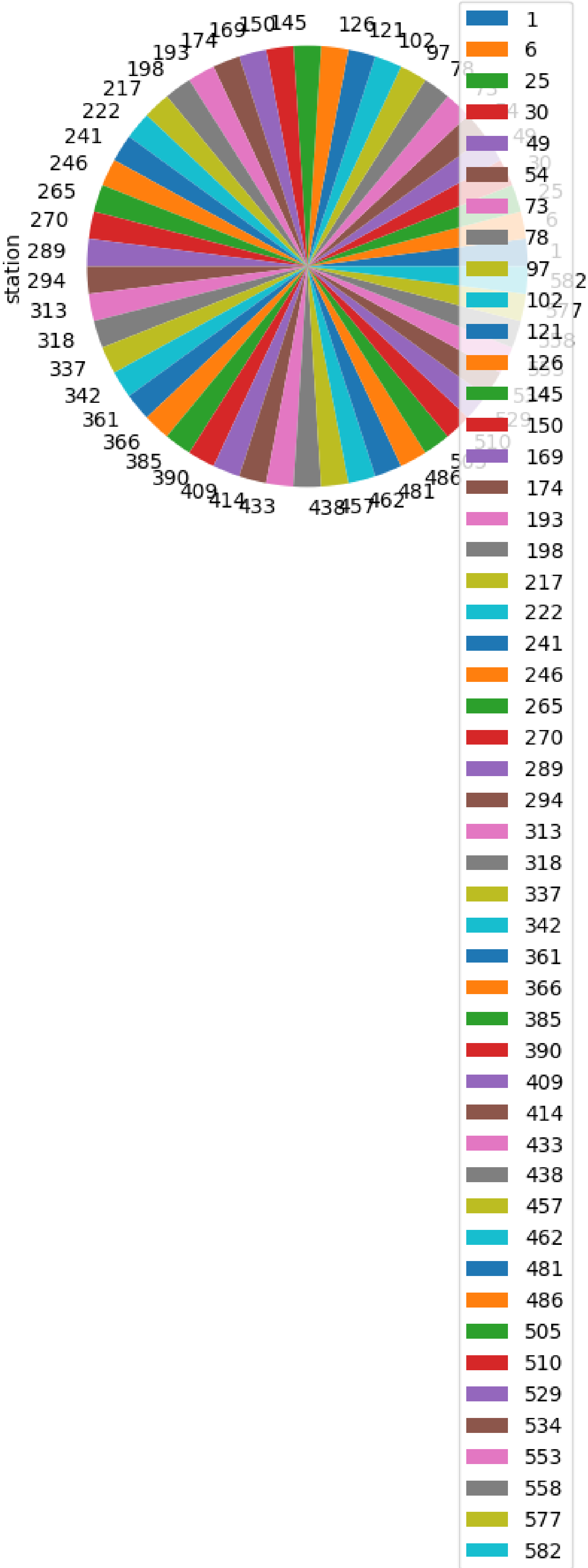
Pie chart

In [14]:

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

Out[14]:

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



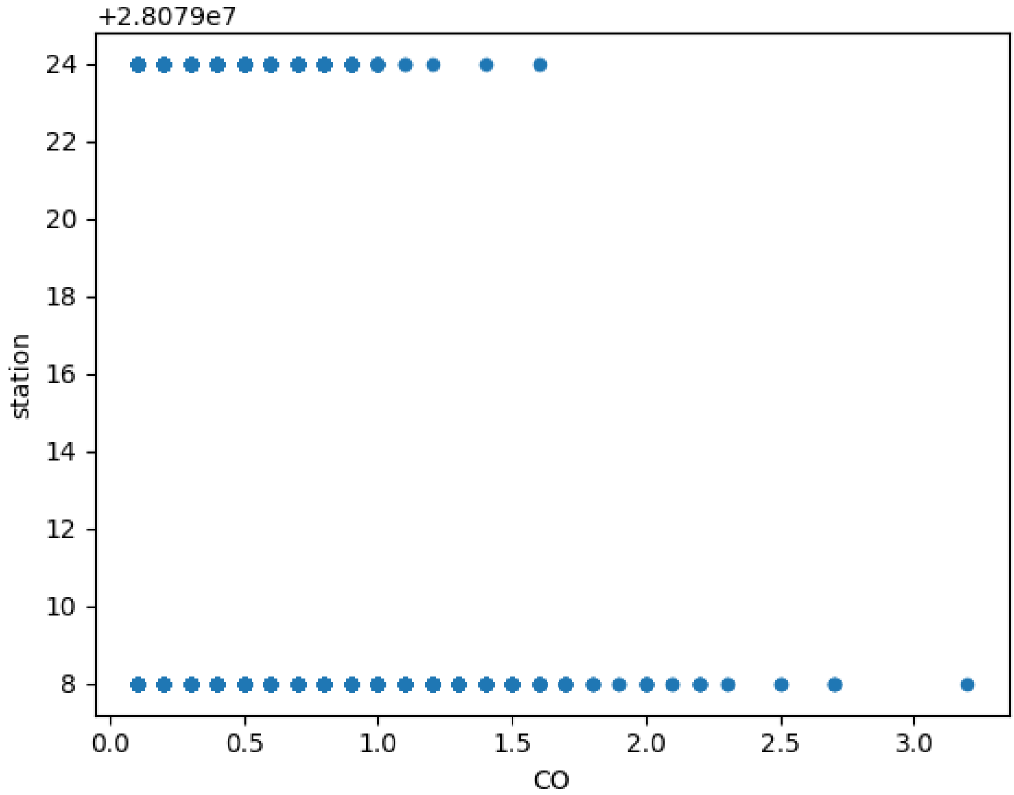
Scatter chart

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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10
count	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000
mean	0.900680	0.277758	1.471871	0.167043	23.671810	44.583961	41.580377	24.670109
std	0.768892	0.206143	1.051004	0.075068	44.362859	31.569185	28.113385	18.758383
min	0.100000	0.100000	0.200000	0.010000	1.000000	1.000000	1.000000	1.000000
25%	0.500000	0.200000	0.800000	0.120000	2.000000	19.000000	17.000000	13.000000
50%	0.700000	0.200000	1.200000	0.160000	7.000000	40.000000	39.000000	20.000000
75%	1.100000	0.300000	1.700000	0.200000	25.000000	63.000000	61.000000	31.000000
max	9.500000	3.200000	12.800000	0.840000	615.000000	289.000000	154.000000	281.000000

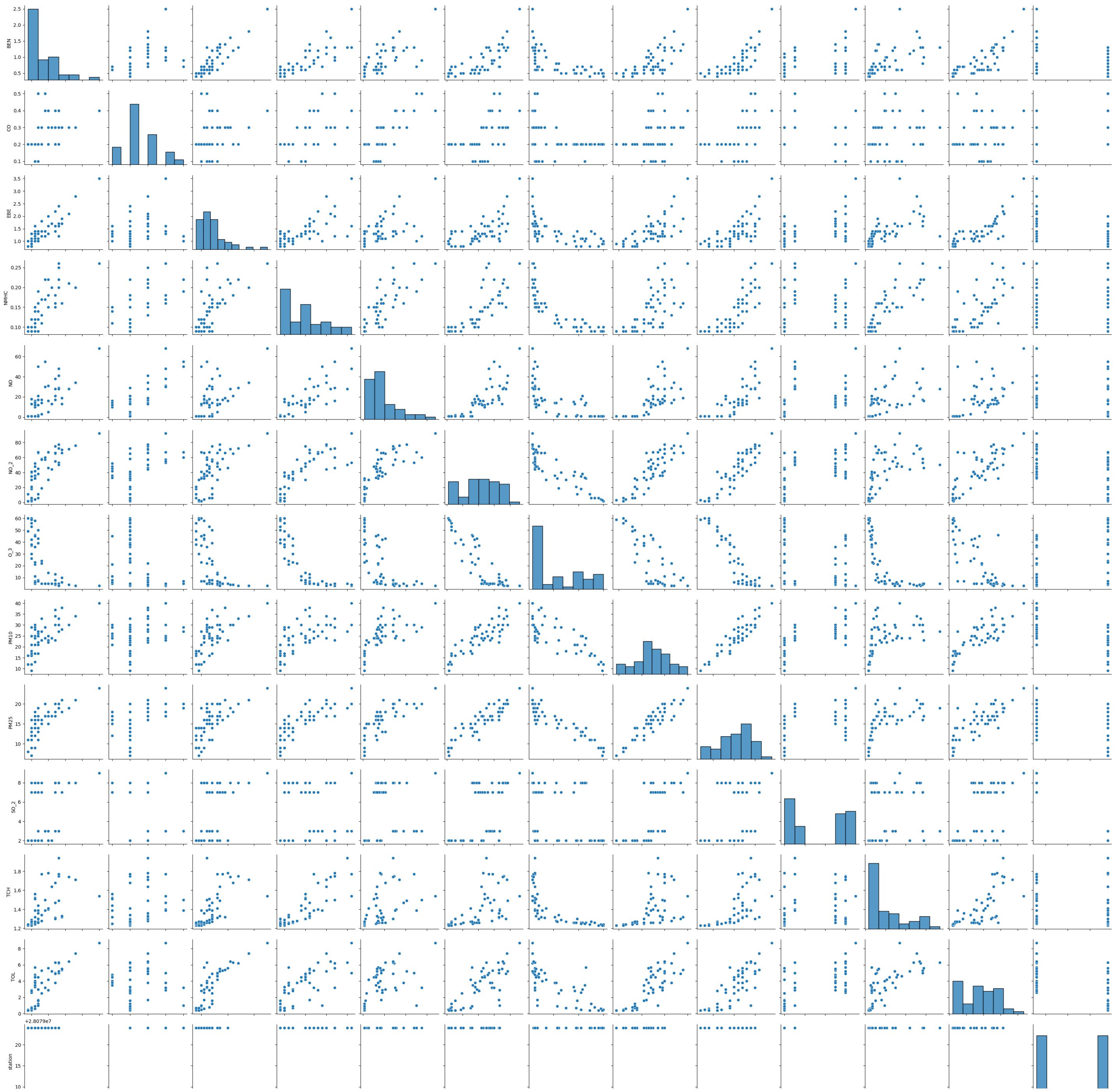
EDA AND VISUALIZATION

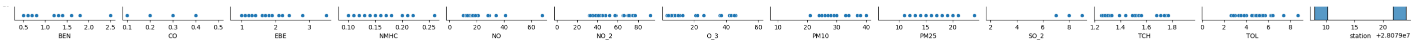
In [18]:

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

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x7b6064e95d80>
```





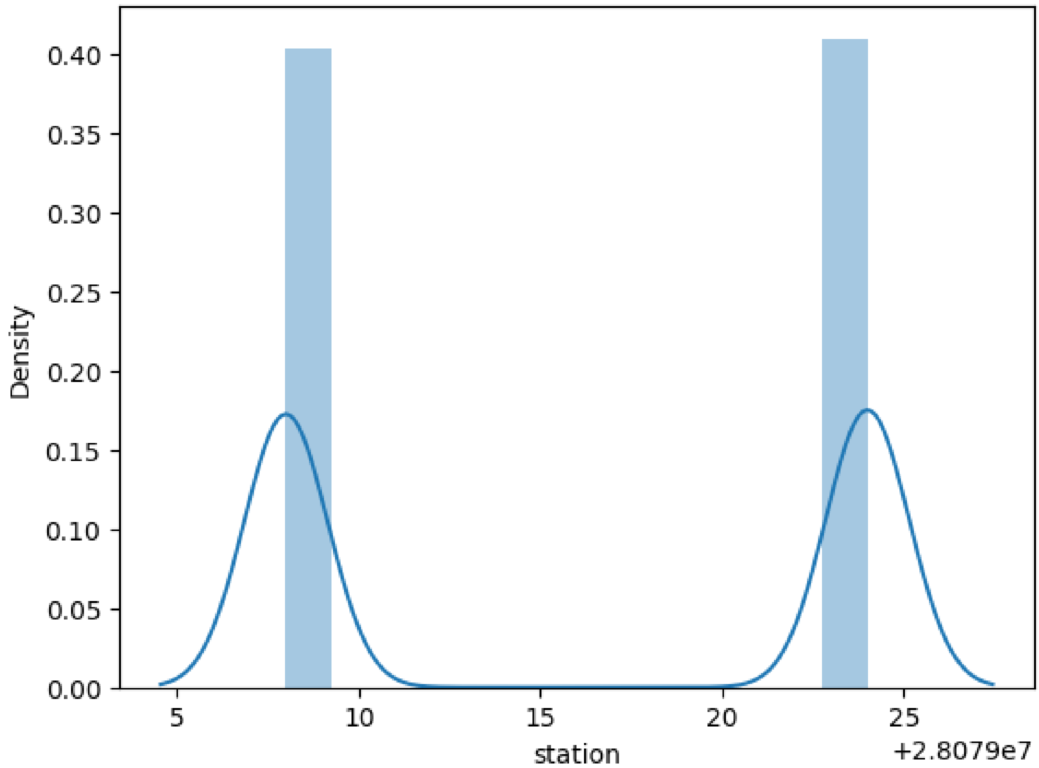
In [19]:

```
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'>
```



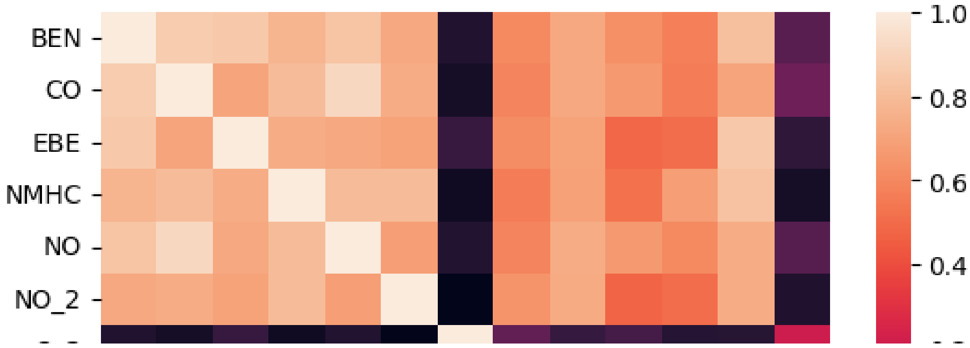
In [20]:

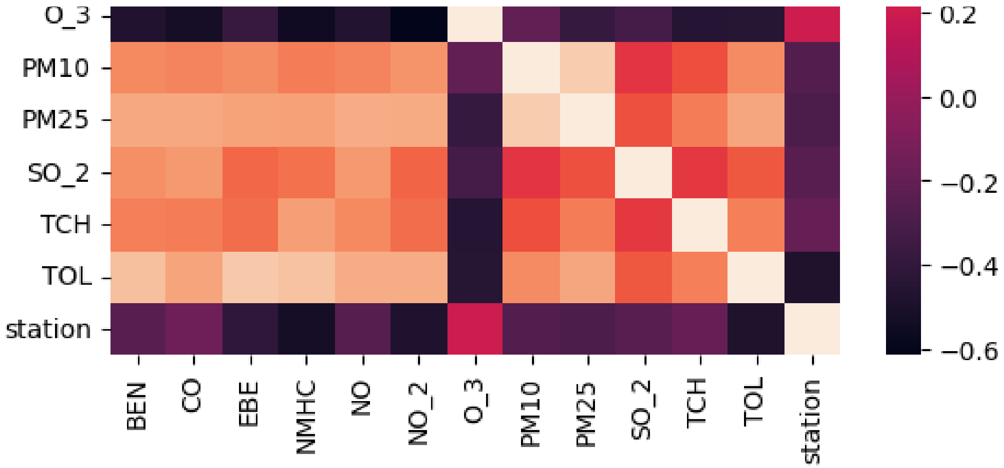
```
sns.heatmap(df.corr())
```

```
<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]:

```
28079015.040081598
```

In [25]:

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

Out[25]:

	Co-efficient
BEN	3.612437
CO	38.987211
EBE	-1.782485
NMHC	-90.336629
NO	-0.039297

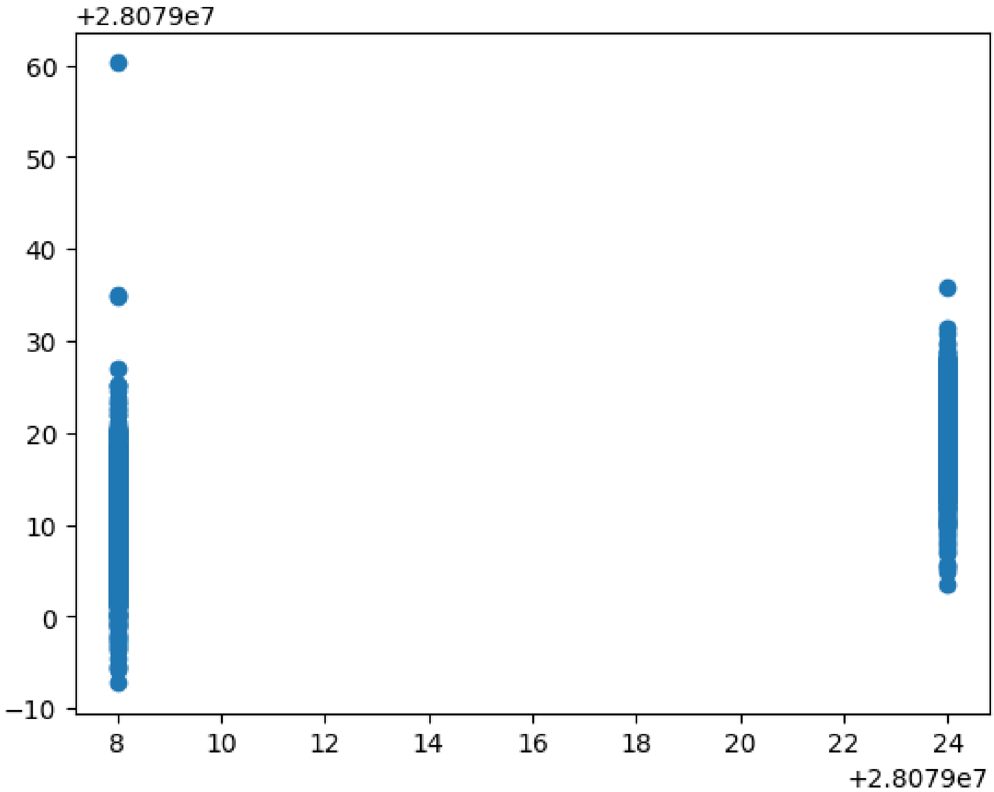
	Coefficient
NO_2	[illegible]
O_3	-0.013953
PM10	0.022116
PM25	-0.053640
SO_2	-0.468810
TCH	10.783457
TOL	-0.393568

In [26]:

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

Out[26]:

```
<matplotlib.collections.PathCollection at 0x7b609ba06080>
```



ACCURACY

In [27]:

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

Out[27]:

```
0.6305094914108933
```

In [28]:

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

Out[28]:

```
0.6254257161736327
```

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]:

```
0.5914405623037331
```

In [32]:

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

Out[32]:

```
0.5934413824701297
```

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]:

```
0.23975637701513786
```

Accuracy(Lasso)

In [35]:

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

Out[35]:

```
0.2322602002623505
```

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([ 0.30105633,  0.        , -0.        , -0.        ,  0.05259405,
       -0.13551684, -0.0423652 ,  0.03242499,  0.0690559 , -0.18573489,
        0.        , -0.95448992])
```

In [38]:

```
en.intercept_
```

Out[38]:

```
28079025.16335977
```

In [39]:

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

In [40]:

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

Out[40]:

```
0.3463848208346624
```

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)))
```

```
5.654696143495199
41.828345260407275
6.467483688453128
```

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]:

```
(16460, 12)
```

In [45]:

```
target_vector.shape
```

Out[45]:

```
(16460,)
```

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]:

```
0.9262454434993924
```

In [53]:

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

Out[53]:

```
0.9999999999999999
```

In [54]:

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

Out[54]:

```
array([[1.00000000e+00, 9.78522268e-17]])
```

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]:

```
0.9376844297864955
```

In [60]:

```
rfc_best=grid_search.best_estimator_
```

In [61]:

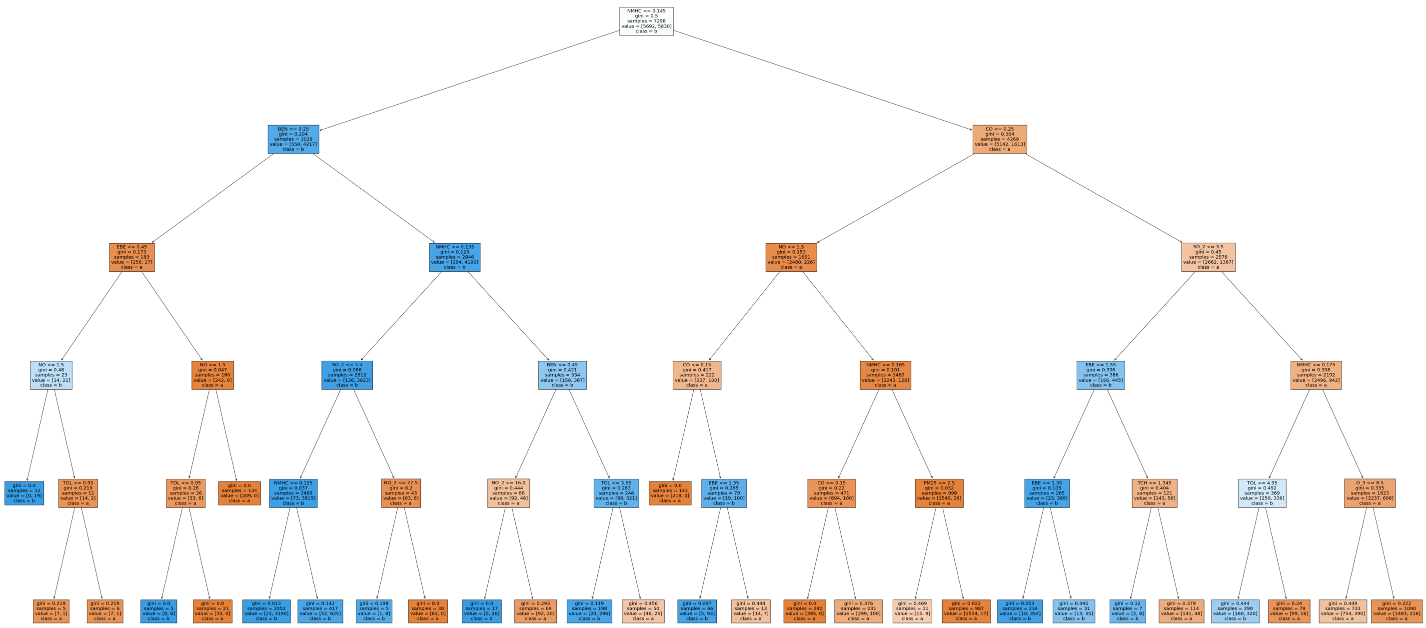
```
from sklearn.tree import plot_tree

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(0.455188679245283, 0.9166666666666666, 'NMHC <= 0.145\ngini = 0.5\nsamples = 7298\n
value = [5692, 5830]\nclass = b'),
 Text(0.20754716981132076, 0.75, 'BEN <= 0.25\ngini = 0.204\nsamples = 3029\nvalue = [550
, 4217]\nclass = b'),
 Text(0.09433962264150944, 0.5833333333333334, 'EBE <= 0.45\ngini = 0.173\nsamples = 183\
nvalue = [256, 27]\nclass = a'),
 Text(0.03773584905660377, 0.4166666666666667, 'NO <= 1.5\ngini = 0.48\nsamples = 23\nval
ue = [14, 21]\nclass = b'),
 Text(0.018867924528301886, 0.25, 'gini = 0.0\nsamples = 12\nvalue = [0, 19]\nclass = b')
,
 Text(0.05660377358490566, 0.25, 'TOL <= 0.95\ngini = 0.219\nsamples = 11\nvalue = [14, 2
]\nclass = a'),
 Text(0.03773584905660377, 0.08333333333333333, 'gini = 0.219\nsamples = 5\nvalue = [7, 1
]\nclass = a'),
 Text(0.07547169811320754, 0.08333333333333333, 'gini = 0.219\nsamples = 6\nvalue = [7, 1
```

```
]\nclass = a'),
 Text(0.1509433962264151, 0.4166666666666667, 'NO <= 1.5\ngini = 0.047\nsamples = 160\nva
lue = [242, 6]\nclass = a'),
 Text(0.1320754716981132, 0.25, 'TOL <= 0.95\ngini = 0.26\nsamples = 26\nvalue = [33, 6]\
nclass = a'),
 Text(0.11320754716981132, 0.08333333333333333, 'gini = 0.0\nsamples = 5\nvalue = [0, 6]\
nclass = b'),
 Text(0.1509433962264151, 0.08333333333333333, 'gini = 0.0\nsamples = 21\nvalue = [33, 0]
\nclass = a'),
 Text(0.16981132075471697, 0.25, 'gini = 0.0\nsamples = 134\nvalue = [209, 0]\nclass = a'
),
 Text(0.32075471698113206, 0.5833333333333334, 'NMHC <= 0.135\ngini = 0.123\nsamples = 28
46\nvalue = [294, 4190]\nclass = b'),
 Text(0.24528301886792453, 0.4166666666666667, 'SO_2 <= 7.5\ngini = 0.066\nsamples = 2512
\nvalue = [136, 3823]\nclass = b'),
 Text(0.20754716981132076, 0.25, 'NMHC <= 0.125\ngini = 0.037\nsamples = 2469\nvalue = [7
3, 3815]\nclass = b'),
 Text(0.18867924528301888, 0.08333333333333333, 'gini = 0.013\nsamples = 2052\nvalue = [2
1, 3190]\nclass = b'),
 Text(0.22641509433962265, 0.08333333333333333, 'gini = 0.142\nsamples = 417\nvalue = [52
, 625]\nclass = b'),
 Text(0.2830188679245283, 0.25, 'NO_2 <= 27.5\ngini = 0.2\nsamples = 43\nvalue = [63, 8]\
nclass = a'),
 Text(0.2641509433962264, 0.08333333333333333, 'gini = 0.198\nsamples = 5\nvalue = [1, 8]
\nclass = b'),
 Text(0.3018867924528302, 0.08333333333333333, 'gini = 0.0\nsamples = 38\nvalue = [62, 0]
\nclass = a'),
 Text(0.39622641509433965, 0.4166666666666667, 'BEN <= 0.45\ngini = 0.421\nsamples = 334\
nvalue = [158, 367]\nclass = b'),
 Text(0.3584905660377358, 0.25, 'NO_2 <= 16.0\ngini = 0.444\nsamples = 86\nvalue = [92, 4
6]\nclass = a'),
 Text(0.33962264150943394, 0.08333333333333333, 'gini = 0.0\nsamples = 17\nvalue = [0, 26
]\nclass = b'),
 Text(0.37735849056603776, 0.08333333333333333, 'gini = 0.293\nsamples = 69\nvalue = [92,
20]\nclass = a'),
 Text(0.4339622641509434, 0.25, 'TOL <= 3.55\ngini = 0.283\nsamples = 248\nvalue = [66, 3
21]\nclass = b'),
 Text(0.41509433962264153, 0.08333333333333333, 'gini = 0.119\nsamples = 198\nvalue = [20
, 296]\nclass = b'),
 Text(0.4528301886792453, 0.08333333333333333, 'gini = 0.456\nsamples = 50\nvalue = [46,
25]\nclass = a'),
 Text(0.7028301886792453, 0.75, 'CO <= 0.25\ngini = 0.364\nsamples = 4269\nvalue = [5142,
1613]\nclass = a'),
 Text(0.5566037735849056, 0.5833333333333334, 'NO <= 1.5\ngini = 0.153\nsamples = 1691\nv
alue = [2480, 226]\nclass = a'),
 Text(0.49056603773584906, 0.4166666666666667, 'CO <= 0.15\ngini = 0.417\nsamples = 222\n
value = [237, 100]\nclass = a'),
 Text(0.4716981132075472, 0.25, 'gini = 0.0\nsamples = 143\nvalue = [218, 0]\nclass = a')
,
 Text(0.5094339622641509, 0.25, 'EBE <= 1.35\ngini = 0.268\nsamples = 79\nvalue = [19, 10
0]\nclass = b'),
 Text(0.49056603773584906, 0.08333333333333333, 'gini = 0.097\nsamples = 66\nvalue = [5,
93]\nclass = b'),
 Text(0.5283018867924528, 0.08333333333333333, 'gini = 0.444\nsamples = 13\nvalue = [14,
7]\nclass = a'),
 Text(0.6226415094339622, 0.4166666666666667, 'NMHC <= 0.165\ngini = 0.101\nsamples = 146
9\nvalue = [2243, 126]\nclass = a'),
 Text(0.5849056603773585, 0.25, 'CO <= 0.15\ngini = 0.22\nsamples = 471\nvalue = [694, 10
0]\nclass = a'),
 Text(0.5660377358490566, 0.08333333333333333, 'gini = 0.0\nsamples = 240\nvalue = [395,
0]\nclass = a'),
 Text(0.6037735849056604, 0.08333333333333333, 'gini = 0.376\nsamples = 231\nvalue = [299
, 100]\nclass = a'),
 Text(0.660377358490566, 0.25, 'PM25 <= 2.5\ngini = 0.032\nsamples = 998\nvalue = [1549,
26]\nclass = a'),
 Text(0.6415094339622641, 0.08333333333333333, 'gini = 0.469\nsamples = 11\nvalue = [15,
9]\nclass = a'),
 Text(0.6792452830188679, 0.08333333333333333, 'gini = 0.022\nsamples = 987\nvalue = [153
4, 17]\nclass = a'),
 Text(0.8490566037735849, 0.5833333333333334, 'SO_2 <= 3.5\ngini = 0.45\nsamples = 2578\n
value = [2662, 1387]\nclass = a'),
 Text(0.7735849056603774, 0.4166666666666667, 'EBE <= 1.55\ngini = 0.396\nsamples = 386\n
```

```
value = [166, 445]\nclass = b'),
 Text(0.7358490566037735, 0.25, 'EBE <= 1.35\ngini = 0.105\nsamples = 265\nvalue = [23, 3
89]\nclass = b'),
 Text(0.7169811320754716, 0.08333333333333333, 'gini = 0.053\nsamples = 234\nvalue = [10,
354]\nclass = b'),
 Text(0.7547169811320755, 0.08333333333333333, 'gini = 0.395\nsamples = 31\nvalue = [13,
35]\nclass = b'),
 Text(0.8113207547169812, 0.25, 'TCH <= 1.345\ngini = 0.404\nsamples = 121\nvalue = [143,
56]\nclass = a'),
 Text(0.7924528301886793, 0.08333333333333333, 'gini = 0.32\nsamples = 7\nvalue = [2, 8]\
nclass = b'),
 Text(0.8301886792452831, 0.08333333333333333, 'gini = 0.379\nsamples = 114\nvalue = [141
, 48]\nclass = a'),
 Text(0.9245283018867925, 0.4166666666666667, 'NMHC <= 0.175\ngini = 0.398\nsamples = 219
2\nvalue = [2496, 942]\nclass = a'),
 Text(0.8867924528301887, 0.25, 'TOL <= 4.95\ngini = 0.492\nsamples = 369\nvalue = [259,
336]\nclass = b'),
 Text(0.8679245283018868, 0.08333333333333333, 'gini = 0.444\nsamples = 290\nvalue = [160
, 320]\nclass = b'),
 Text(0.9056603773584906, 0.08333333333333333, 'gini = 0.24\nsamples = 79\nvalue = [99, 1
6]\nclass = a'),
 Text(0.9622641509433962, 0.25, 'O_3 <= 8.5\ngini = 0.335\nsamples = 1823\nvalue = [2237,
606]\nclass = a'),
 Text(0.9433962264150944, 0.08333333333333333, 'gini = 0.449\nsamples = 733\nvalue = [754
, 390]\nclass = a'),
 Text(0.9811320754716981, 0.08333333333333333, 'gini = 0.222\nsamples = 1090\nvalue = [14
83, 216]\nclass = 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: 0.6305094914108933
Ridge Regression: 0.5914405623037331
Lasso Regression 0.2322602002623505
```

```
ElasticNet Regression: 0.3463848208346624
Logistic Regression: 0.9262454434993924
Random Forest: 0.9376844297864955
```

Random Forest is suitable for this dataset