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_2016.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	2016-11-01 01:00:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	28079004
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	28079008
2	2016-11-01 01:00:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	28079011
3	2016-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2016-11-01 01:00:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209491	2016-07-01 00:00:00	NaN	0.2	NaN	NaN	2.0	29.0	73.0	NaN	NaN	NaN	NaN	NaN	28079056
209492	2016-07-01 00:00:00	NaN	0.3	NaN	NaN	1.0	29.0	NaN	36.0	NaN	5.0	NaN	NaN	28079057
209493	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	1.0	19.0	71.0	NaN	NaN	NaN	NaN	NaN	28079058
209494	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	6.0	17.0	85.0	NaN	NaN	NaN	NaN	NaN	28079059
209495	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	2.0	46.0	61.0	34.0	NaN	NaN	NaN	NaN	28079060

209496 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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 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	1.1	28079008
6	0.8	28079024
25	1.0	28079008
30	0.7	28079024
49	0.8	28079008
...	...	...
209430	0.2	28079024
209449	0.4	28079008
209454	0.2	28079024
209473	0.4	28079008
209478	0.2	28079024

16932 rows × 2 columns

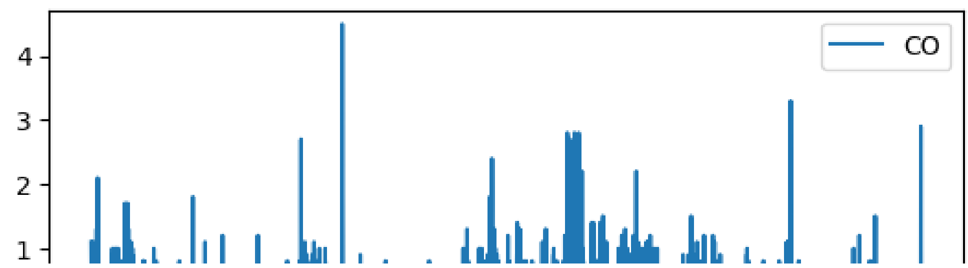
Line chart

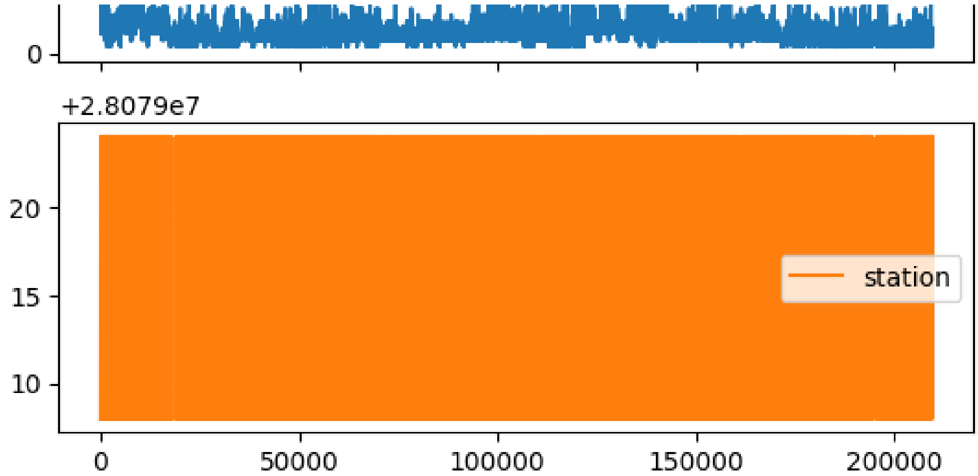
In [7]:

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

Out[7]:

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





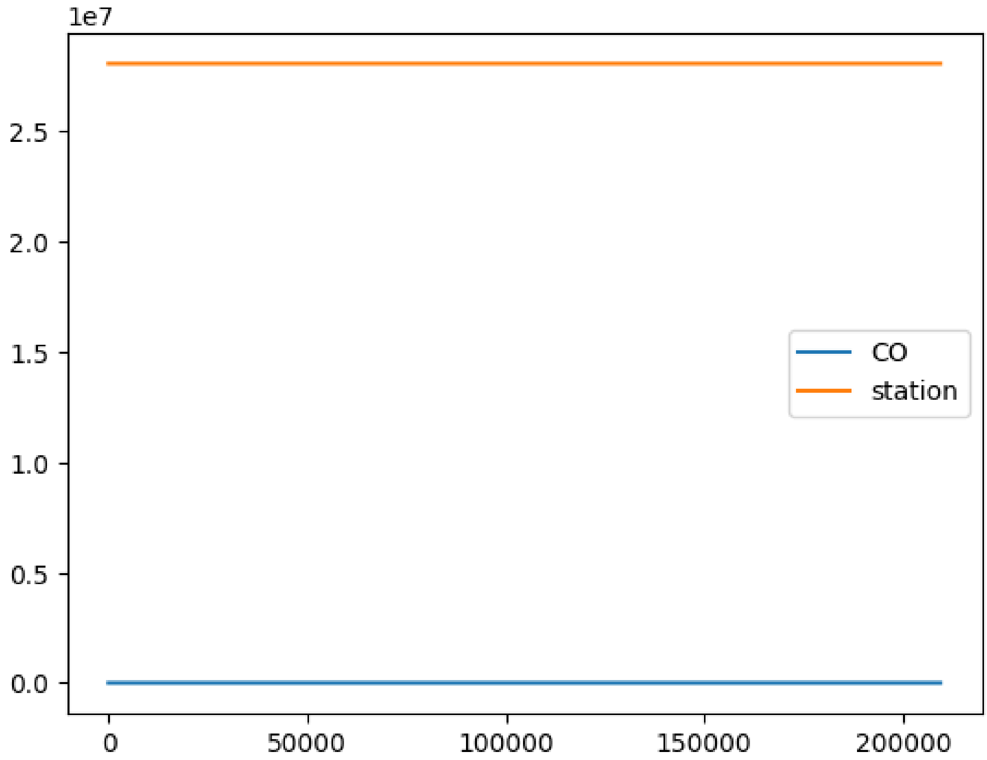
Line chart

In [8]:

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

Out[8]:

<Axes: >



Bar chart

In [9]:

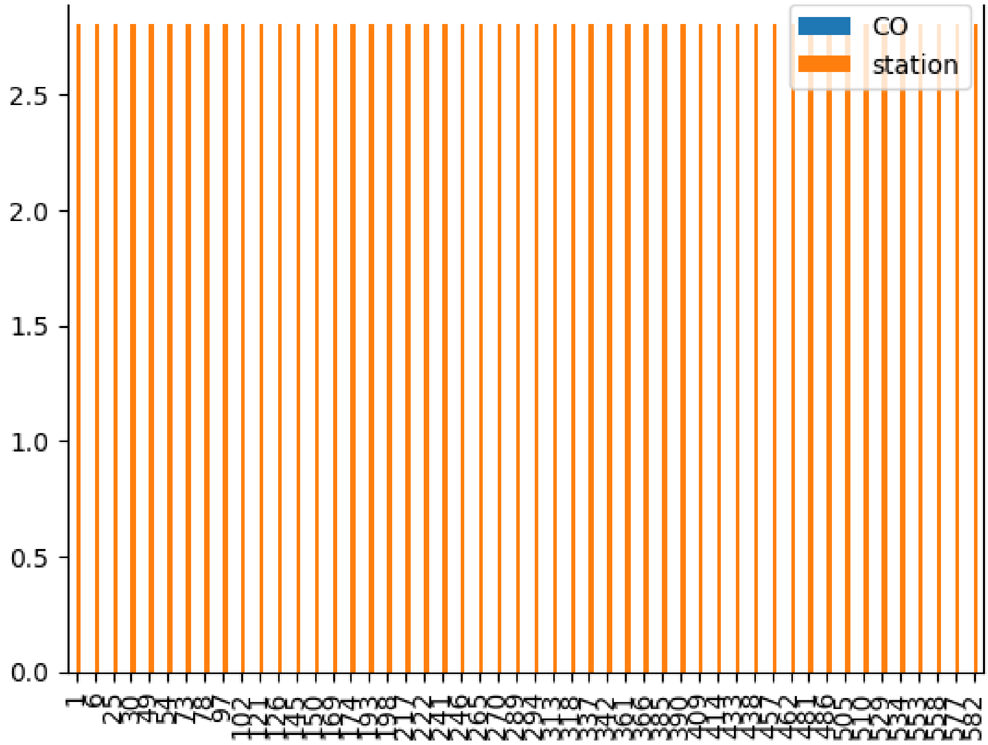
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



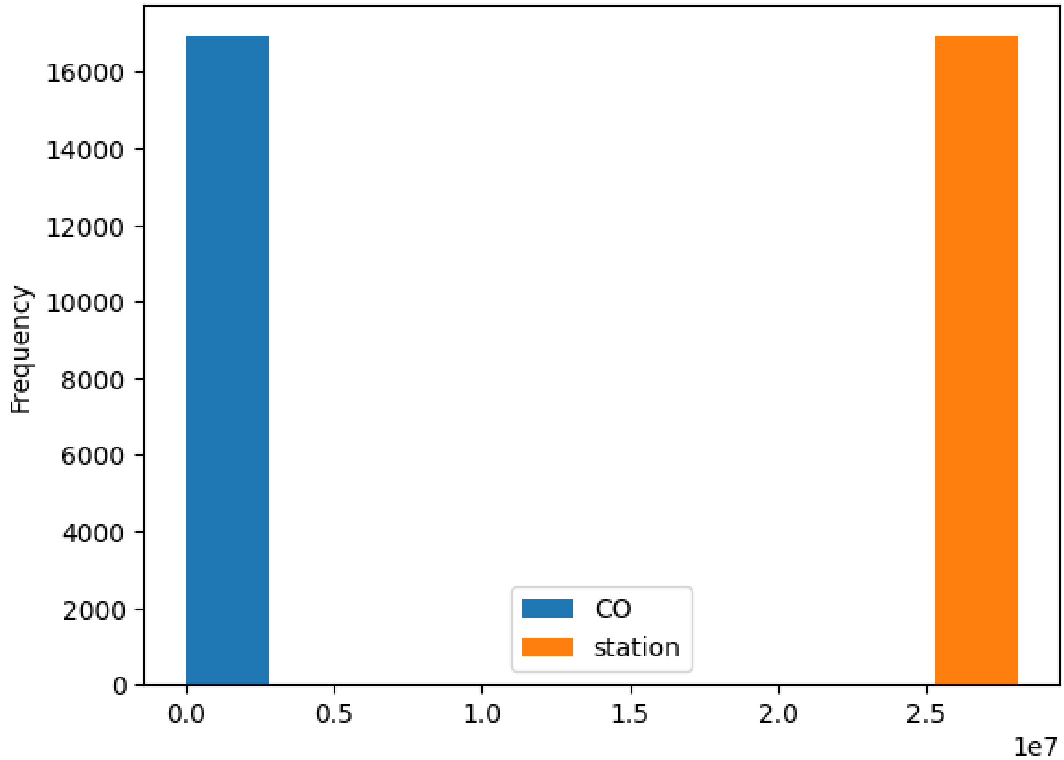
Histogram

In [11]:

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

Out[11]:

<Axes: ylabel='Frequency'>



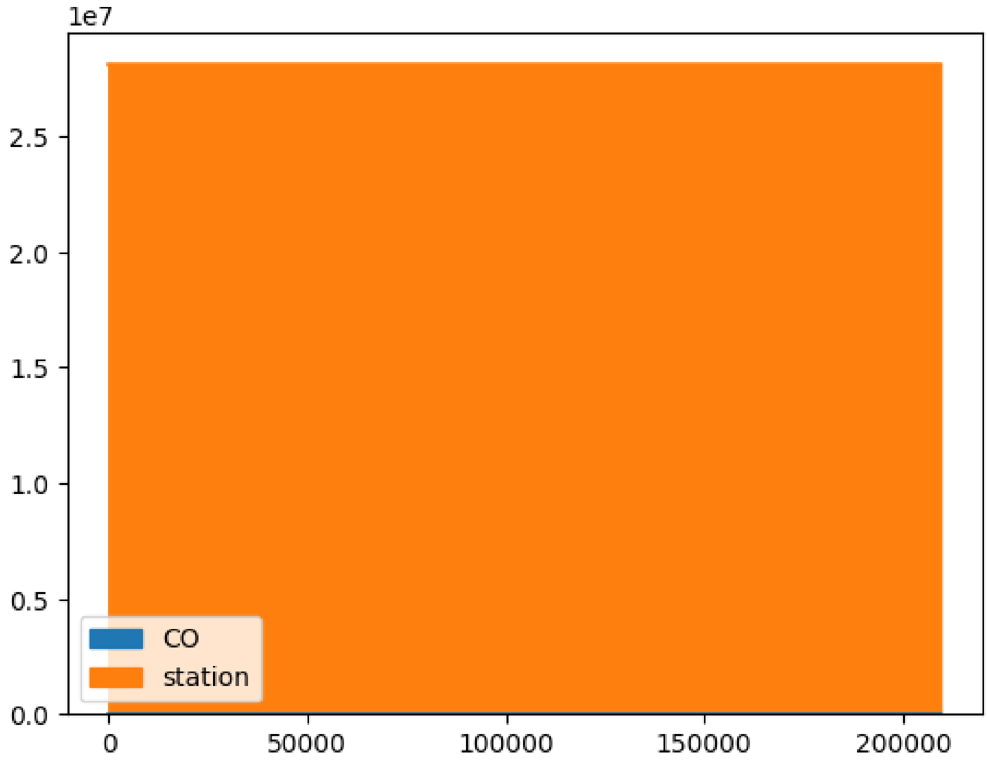
Area chart

In [12]:

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

Out[12]:

<Axes: >



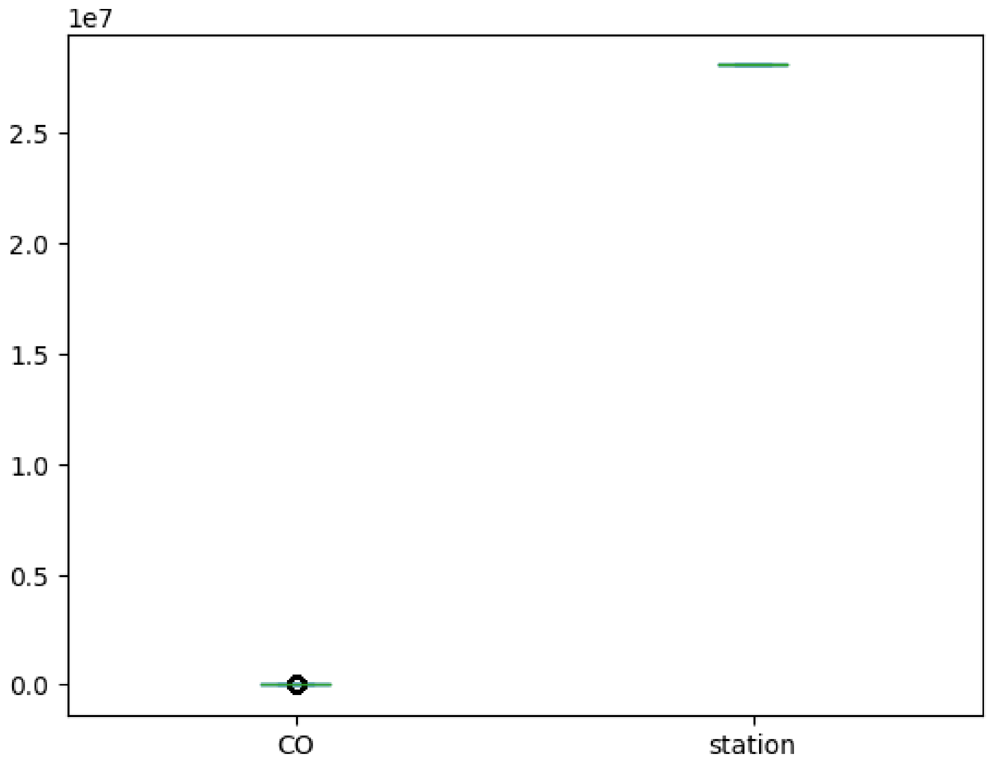
Box chart

In [13]:

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

Out[13]:

<Axes: >



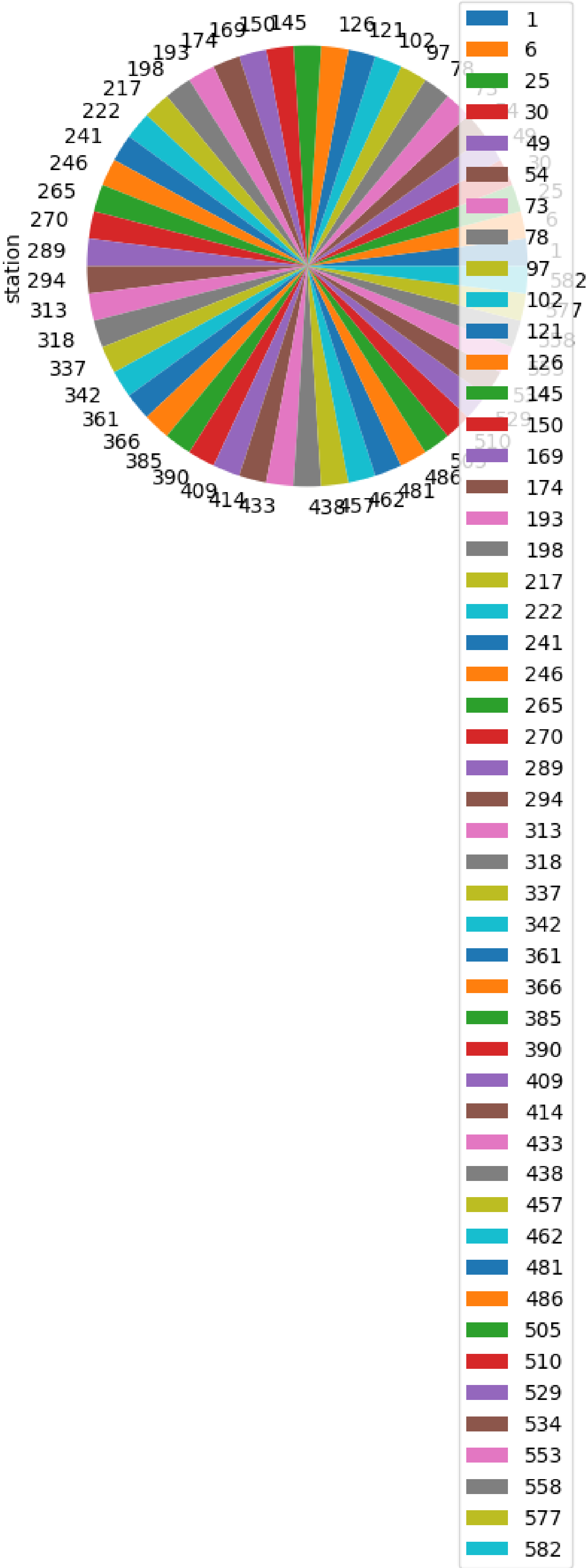
Pie chart

In [14]:

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

Out[14]:

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



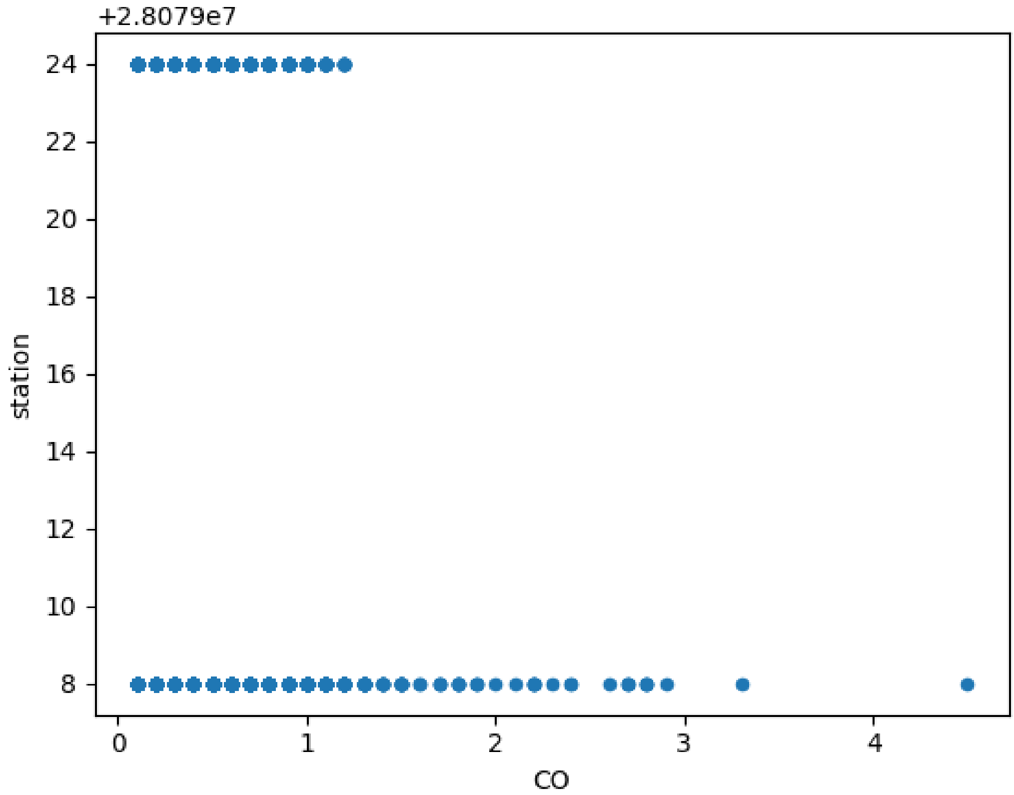
Scatter chart

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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     16932 non-null  object
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 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	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	169
mean	0.537970	0.349941	0.298955	0.099913	20.815734	39.373376	48.118474	19.248110	
std	0.599479	0.203807	0.450204	0.079850	40.986063	31.170307	32.560277	18.509093	
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000	1.000000	
25%	0.200000	0.200000	0.100000	0.050000	1.000000	14.000000	21.000000	9.000000	
50%	0.400000	0.300000	0.200000	0.090000	7.000000	34.000000	46.000000	15.000000	
75%	0.700000	0.400000	0.300000	0.120000	23.000000	58.000000	69.000000	24.000000	
max	12.300000	4.500000	13.500000	2.210000	829.000000	319.000000	181.000000	367.000000	2

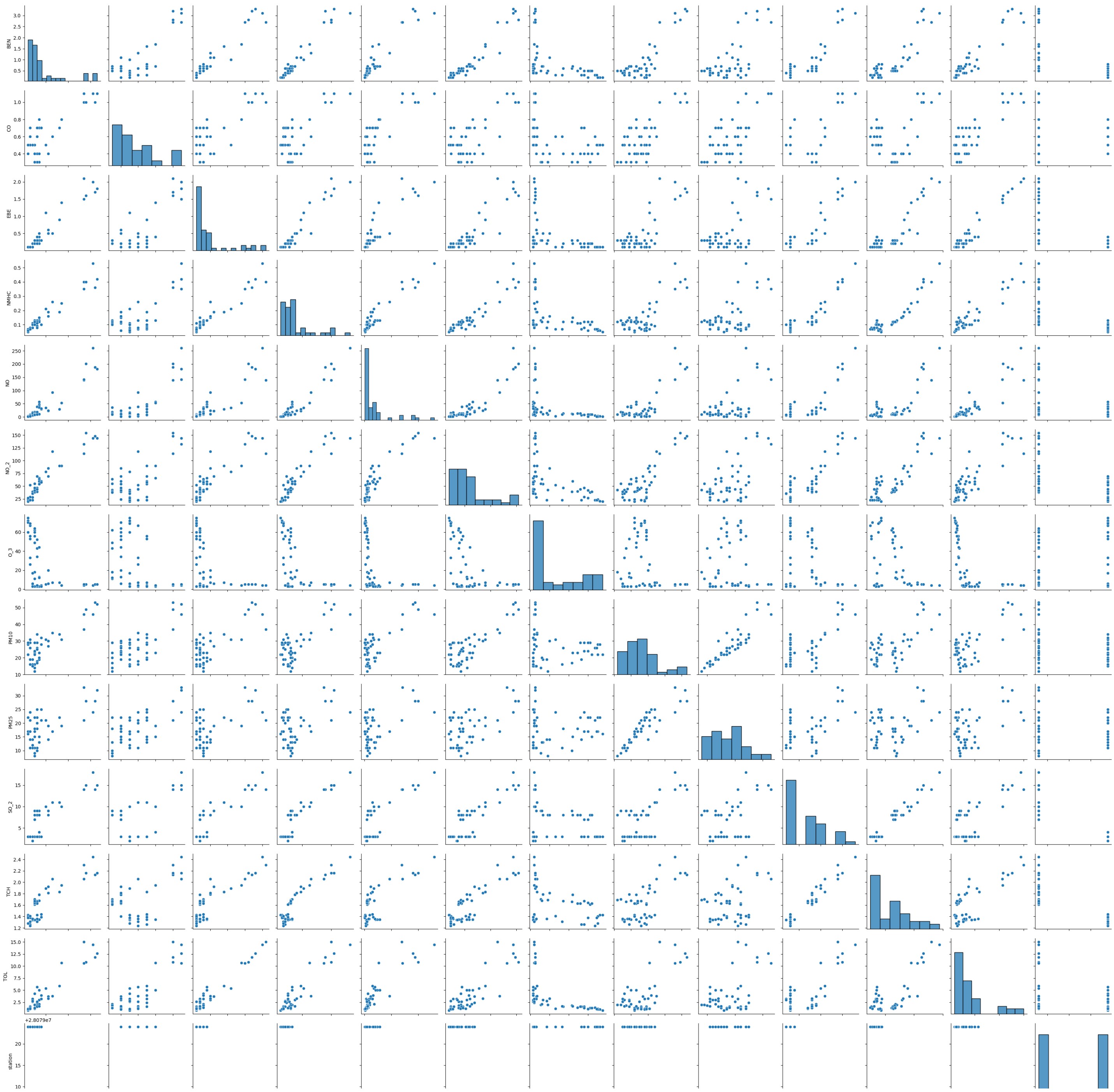
EDA AND VISUALIZATION

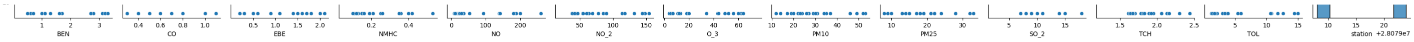
In [18]:

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

Out[18]:

<seaborn.axisgrid.PairGrid at 0x79330053a830>





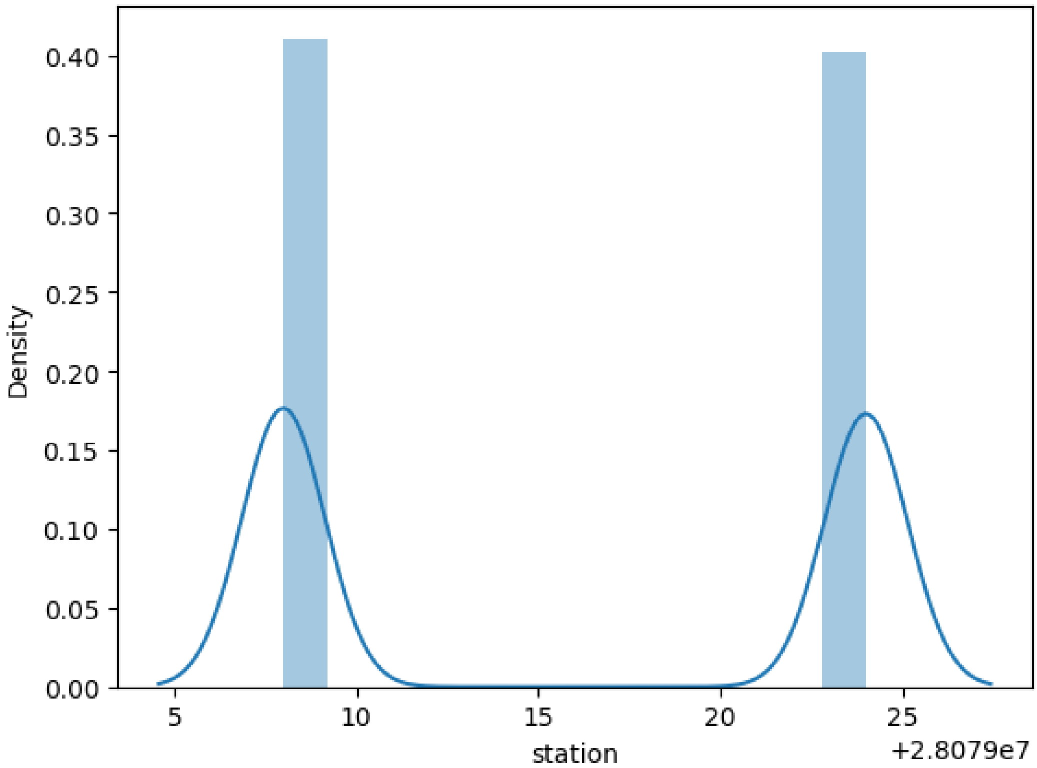
In [19]:

```
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'>
```



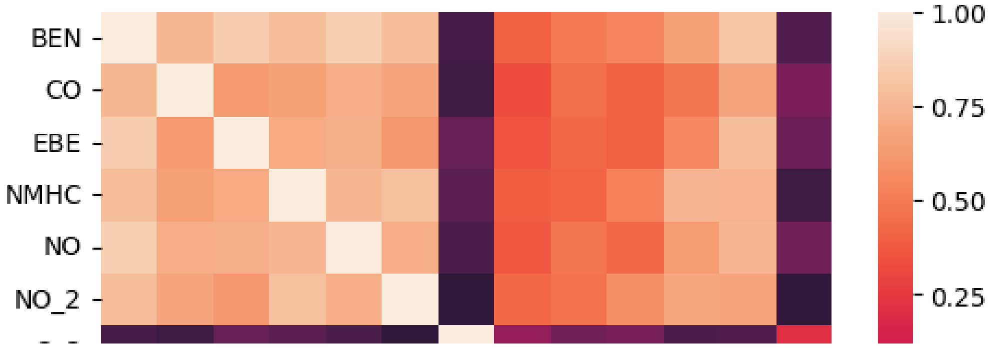
In [20]:

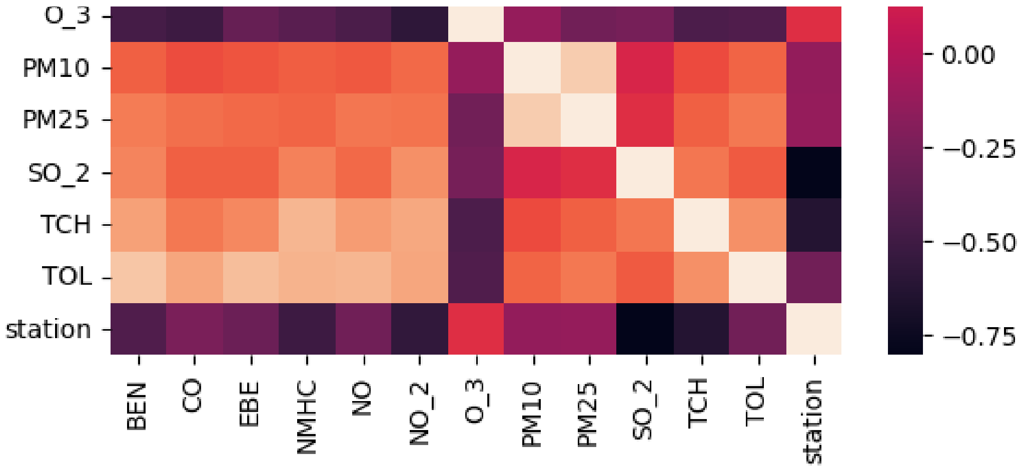
```
sns.heatmap(df.corr())
```

```
<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]:

```
28079042.21686273
```

In [25]:

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

Out[25]:

	Co-efficient
BEN	-1.993384
CO	4.487618
EBE	0.754127
NMHC	2.960473
NO	0.069834

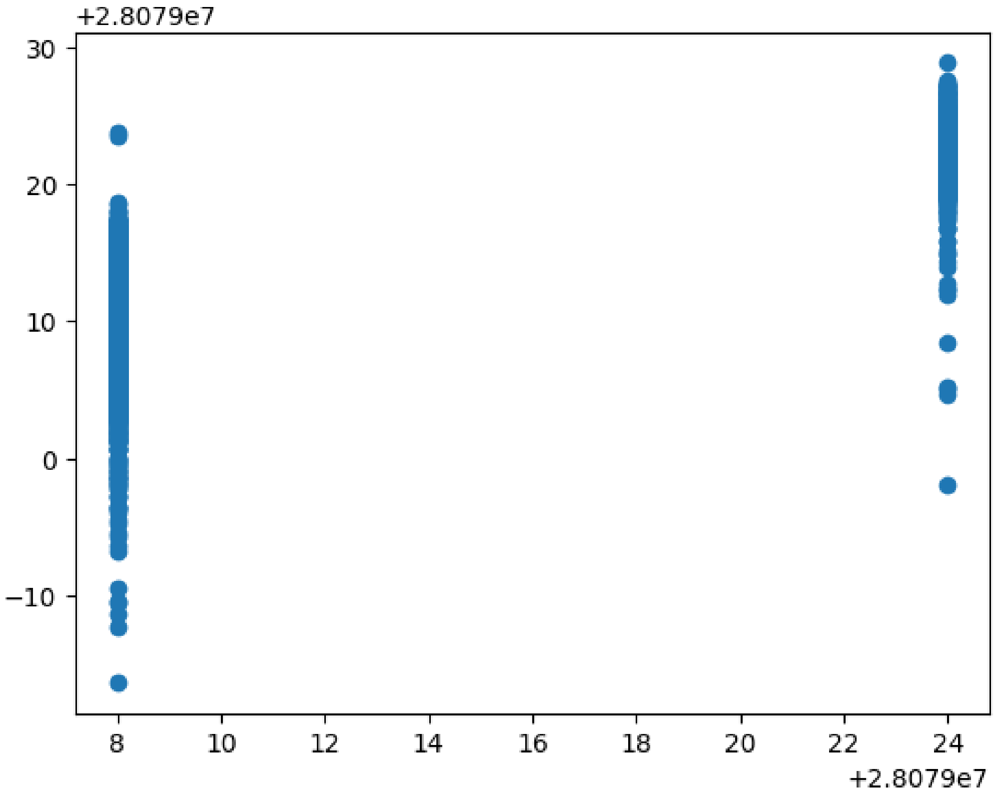
NO_2	Coefficient -0.067438
O_3	-0.024092
PM10	-0.012027
PM25	0.096521
SO_2	-0.800109
TCH	-14.218379
TOL	0.161117

In [26]:

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

Out[26]:

```
<matplotlib.collections.PathCollection at 0x79333a6511b0>
```



ACCURACY

In [27]:

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

Out[27]:

```
0.8387972184876646
```

In [28]:

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

Out[28]:

```
0.8233122579938551
```

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]:

```
0.8387200462261404
```

In [32]:

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

Out[32]:

```
0.8231913774530392
```

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]:

```
0.6421283430780567
```

Accuracy(Lasso)

In [35]:

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

Out[35]:

```
0.6503999023554409
```

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.        ,  0.04912246,
       -0.10855501, -0.02049893,  0.00325871,  0.04611462, -0.8562113 ,
       -0.02336465,  0.        ])
```

In [38]:

```
en.intercept_
```

Out[38]:

```
28079026.177051354
```

In [39]:

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

In [40]:

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

Out[40]:

```
0.7158451670288988
```

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)))
```

```
3.29210566510703
18.18320042261141
4.264176406131835
```

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]:

```
(16932, 12)
```

In [45]:

```
target_vector.shape
```

Out[45]:

```
(16932,)
```

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]:

```
0.996161115048429
```

In [53]:

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

Out[53]:

```
1.0
```

In [54]:

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

Out[54]:

```
array([[1.00000000e+00, 1.38109307e-55]])
```

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]:

```
0.9946000674991562
```

In [60]:

```
rfc_best=grid_search.best_estimator_
```

In [61]:

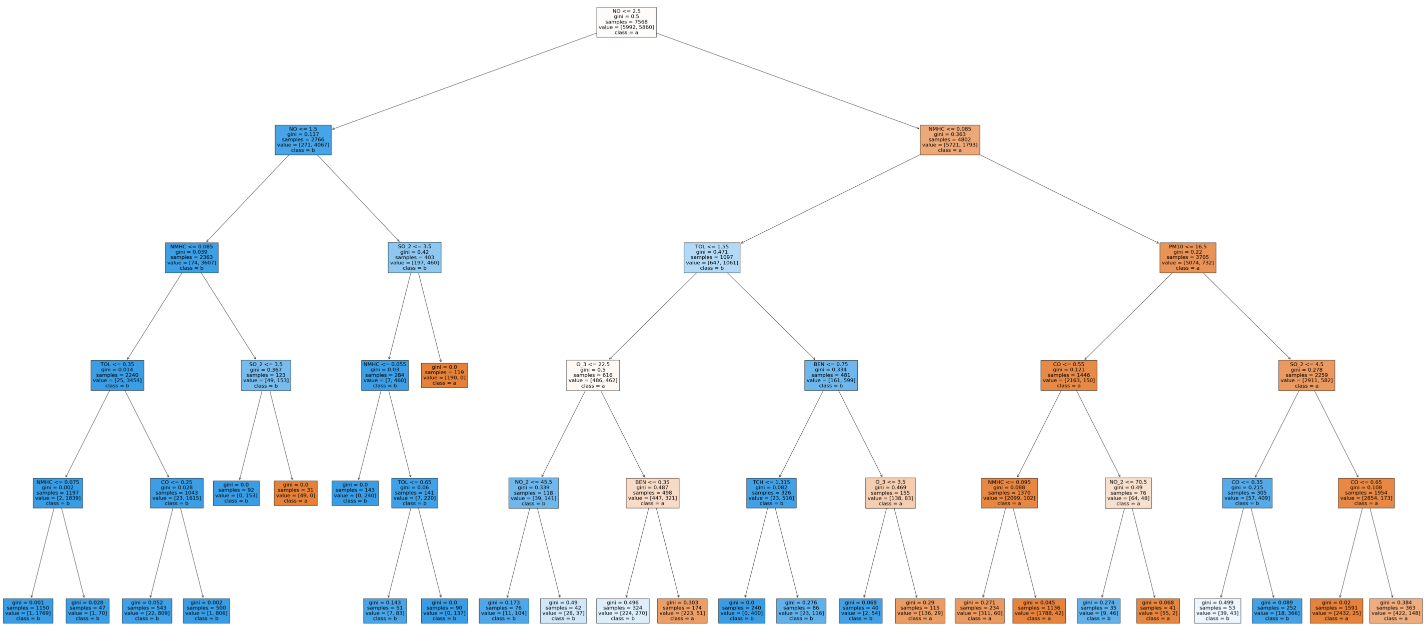
```
from sklearn.tree import plot_tree

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(0.44010416666666667, 0.9166666666666666, 'NO <= 2.5\ngini = 0.5\nsamples = 7568\nval
ue = [5992, 5860]\nclass = a'),
 Text(0.21354166666666666, 0.75, 'NO <= 1.5\ngini = 0.117\nsamples = 2766\nvalue = [271,
4067]\nclass = b'),
 Text(0.13541666666666666, 0.5833333333333334, 'NMHC <= 0.085\ngini = 0.039\nsamples = 23
63\nvalue = [74, 3607]\nclass = b'),
 Text(0.08333333333333333, 0.4166666666666667, 'TOL <= 0.35\ngini = 0.014\nsamples = 2240
\nvalue = [25, 3454]\nclass = b'),
 Text(0.041666666666666664, 0.25, 'NMHC <= 0.075\ngini = 0.002\nsamples = 1197\nvalue = [
2, 1839]\nclass = b'),
 Text(0.020833333333333332, 0.08333333333333333, 'gini = 0.001\nsamples = 1150\nvalue = [
1, 1769]\nclass = b'),
 Text(0.0625, 0.08333333333333333, 'gini = 0.028\nsamples = 47\nvalue = [1, 70]\nclass =
b'),
 Text(0.125, 0.25, 'CO <= 0.25\ngini = 0.028\nsamples = 1043\nvalue = [23, 1615]\nclass =
```

```
b'),
 Text(0.10416666666666667, 0.08333333333333333, 'gini = 0.052\nsamples = 543\nvalue = [22
, 809]\nclass = b'),
 Text(0.14583333333333334, 0.08333333333333333, 'gini = 0.002\nsamples = 500\nvalue = [1,
806]\nclass = b'),
 Text(0.1875, 0.4166666666666667, 'SO_2 <= 3.5\ngini = 0.367\nsamples = 123\nvalue = [49,
153]\nclass = b'),
 Text(0.16666666666666666, 0.25, 'gini = 0.0\nsamples = 92\nvalue = [0, 153]\nclass = b')
,
 Text(0.20833333333333334, 0.25, 'gini = 0.0\nsamples = 31\nvalue = [49, 0]\nclass = a'),
 Text(0.2916666666666667, 0.5833333333333334, 'SO_2 <= 3.5\ngini = 0.42\nsamples = 403\nv
alue = [197, 460]\nclass = b'),
 Text(0.2708333333333333, 0.4166666666666667, 'NMHC <= 0.055\ngini = 0.03\nsamples = 284\
nvalue = [7, 460]\nclass = b'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 143\nvalue = [0, 240]\nclass = b'),
 Text(0.2916666666666667, 0.25, 'TOL <= 0.65\ngini = 0.06\nsamples = 141\nvalue = [7, 220
]\nclass = b'),
 Text(0.2708333333333333, 0.08333333333333333, 'gini = 0.143\nsamples = 51\nvalue = [7, 8
3]\nclass = b'),
 Text(0.3125, 0.08333333333333333, 'gini = 0.0\nsamples = 90\nvalue = [0, 137]\nclass = b
'),
 Text(0.3125, 0.4166666666666667, 'gini = 0.0\nsamples = 119\nvalue = [190, 0]\nclass = a
'),
 Text(0.6666666666666666, 0.75, 'NMHC <= 0.085\ngini = 0.363\nsamples = 4802\nvalue = [57
21, 1793]\nclass = a'),
 Text(0.5, 0.5833333333333334, 'TOL <= 1.55\ngini = 0.471\nsamples = 1097\nvalue = [647,
1061]\nclass = b'),
 Text(0.4166666666666667, 0.4166666666666667, 'O_3 <= 22.5\ngini = 0.5\nsamples = 616\nva
lue = [486, 462]\nclass = a'),
 Text(0.375, 0.25, 'NO_2 <= 45.5\ngini = 0.339\nsamples = 118\nvalue = [39, 141]\nclass =
b'),
 Text(0.3541666666666667, 0.08333333333333333, 'gini = 0.173\nsamples = 76\nvalue = [11,
104]\nclass = b'),
 Text(0.3958333333333333, 0.08333333333333333, 'gini = 0.49\nsamples = 42\nvalue = [28, 3
7]\nclass = b'),
 Text(0.4583333333333333, 0.25, 'BEN <= 0.35\ngini = 0.487\nsamples = 498\nvalue = [447,
321]\nclass = a'),
 Text(0.4375, 0.08333333333333333, 'gini = 0.496\nsamples = 324\nvalue = [224, 270]\nclas
s = b'),
 Text(0.4791666666666667, 0.08333333333333333, 'gini = 0.303\nsamples = 174\nvalue = [223
, 51]\nclass = a'),
 Text(0.5833333333333334, 0.4166666666666667, 'BEN <= 0.75\ngini = 0.334\nsamples = 481\n
value = [161, 599]\nclass = b'),
 Text(0.5416666666666666, 0.25, 'TCH <= 1.315\ngini = 0.082\nsamples = 326\nvalue = [23,
516]\nclass = b'),
 Text(0.5208333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 240\nvalue = [0, 40
0]\nclass = b'),
 Text(0.5625, 0.08333333333333333, 'gini = 0.276\nsamples = 86\nvalue = [23, 116]\nclass
= b'),
 Text(0.625, 0.25, 'O_3 <= 3.5\ngini = 0.469\nsamples = 155\nvalue = [138, 83]\nclass = a
'),
 Text(0.6041666666666666, 0.08333333333333333, 'gini = 0.069\nsamples = 40\nvalue = [2, 5
4]\nclass = b'),
 Text(0.6458333333333334, 0.08333333333333333, 'gini = 0.29\nsamples = 115\nvalue = [136,
29]\nclass = a'),
 Text(0.8333333333333334, 0.5833333333333334, 'PM10 <= 16.5\ngini = 0.22\nsamples = 3705\
nvalue = [5074, 732]\nclass = a'),
 Text(0.75, 0.4166666666666667, 'CO <= 0.55\ngini = 0.121\nsamples = 1446\nvalue = [2163,
150]\nclass = a'),
 Text(0.7083333333333334, 0.25, 'NMHC <= 0.095\ngini = 0.088\nsamples = 1370\nvalue = [20
99, 102]\nclass = a'),
 Text(0.6875, 0.08333333333333333, 'gini = 0.271\nsamples = 234\nvalue = [311, 60]\nclass
= a'),
 Text(0.7291666666666666, 0.08333333333333333, 'gini = 0.045\nsamples = 1136\nvalue = [17
88, 42]\nclass = a'),
 Text(0.7916666666666666, 0.25, 'NO_2 <= 70.5\ngini = 0.49\nsamples = 76\nvalue = [64, 48
]\nclass = a'),
 Text(0.7708333333333334, 0.08333333333333333, 'gini = 0.274\nsamples = 35\nvalue = [9, 4
6]\nclass = b'),
 Text(0.8125, 0.08333333333333333, 'gini = 0.068\nsamples = 41\nvalue = [55, 2]\nclass =
a'),
 Text(0.9166666666666666, 0.4166666666666667, 'SO_2 <= 4.5\ngini = 0.278\nsamples = 2259\
```

```
nvalue = [2911, 582]\nclass = a'),
 Text(0.875, 0.25, 'CO <= 0.35\ngini = 0.215\nsamples = 305\nvalue = [57, 409]\nclass = b
'),
 Text(0.8541666666666666, 0.08333333333333333, 'gini = 0.499\nsamples = 53\nvalue = [39,
43]\nclass = b'),
 Text(0.8958333333333334, 0.08333333333333333, 'gini = 0.089\nsamples = 252\nvalue = [18,
366]\nclass = b'),
 Text(0.9583333333333334, 0.25, 'CO <= 0.65\ngini = 0.108\nsamples = 1954\nvalue = [2854,
173]\nclass = a'),
 Text(0.9375, 0.08333333333333333, 'gini = 0.02\nsamples = 1591\nvalue = [2432, 25]\nclas
s = a'),
 Text(0.9791666666666666, 0.08333333333333333, 'gini = 0.384\nsamples = 363\nvalue = [422
, 148]\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: 0.8387972184876646
Ridge Regression: 0.8387200462261404
Lasso Regression 0.6503999023554409
ElasticNet Regression: 0.7158451670288988
Logistic Regression: 0.996161115048429
Random Forest: 0.9946000674991562
```

Logistic Regression is suitable for this dataset