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_2015.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	2015-10-01 01:00:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	28079004
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	28079008
2	2015-10-01 01:00:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	28079011
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210091	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN	28079056
210092	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN	28079057
210093	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN	28079058
210094	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN	28079059
210095	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN	28079060

210096 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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

In [6]:

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

Out[6]:

	CO	station
1	0.8	28079008
6	0.3	28079024
25	0.7	28079008
30	0.3	28079024
49	0.8	28079008
...	...	...
210030	0.1	28079024
210049	0.3	28079008
210054	0.1	28079024
210073	0.3	28079008
210078	0.1	28079024

16026 rows × 2 columns

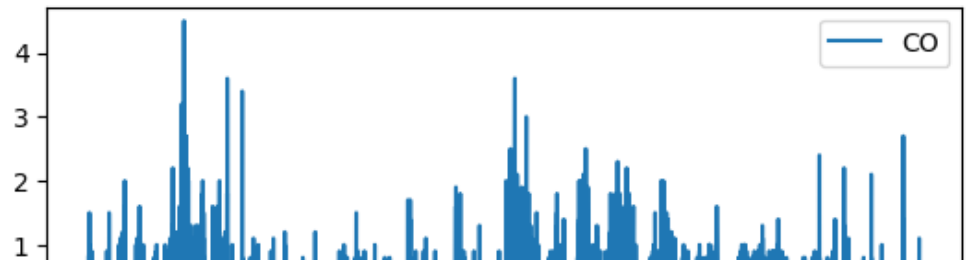
Line chart

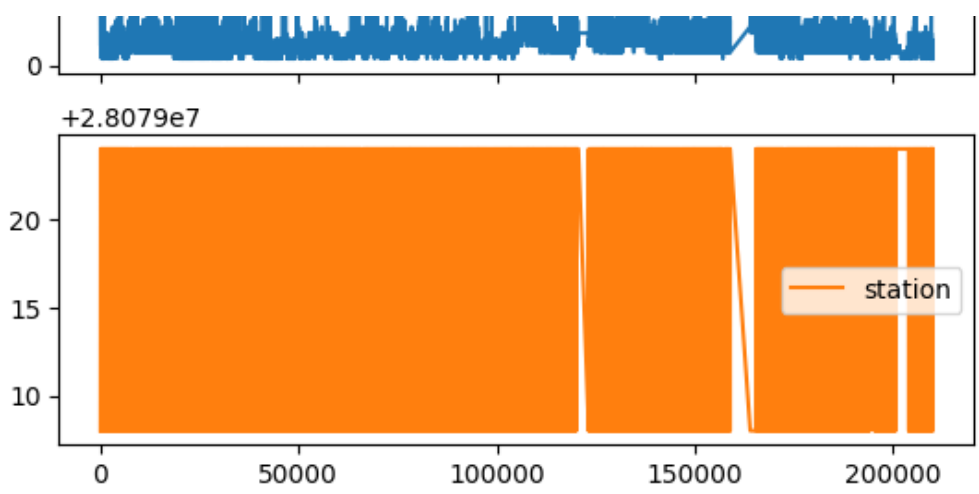
In [7]:

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

Out[7]:

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





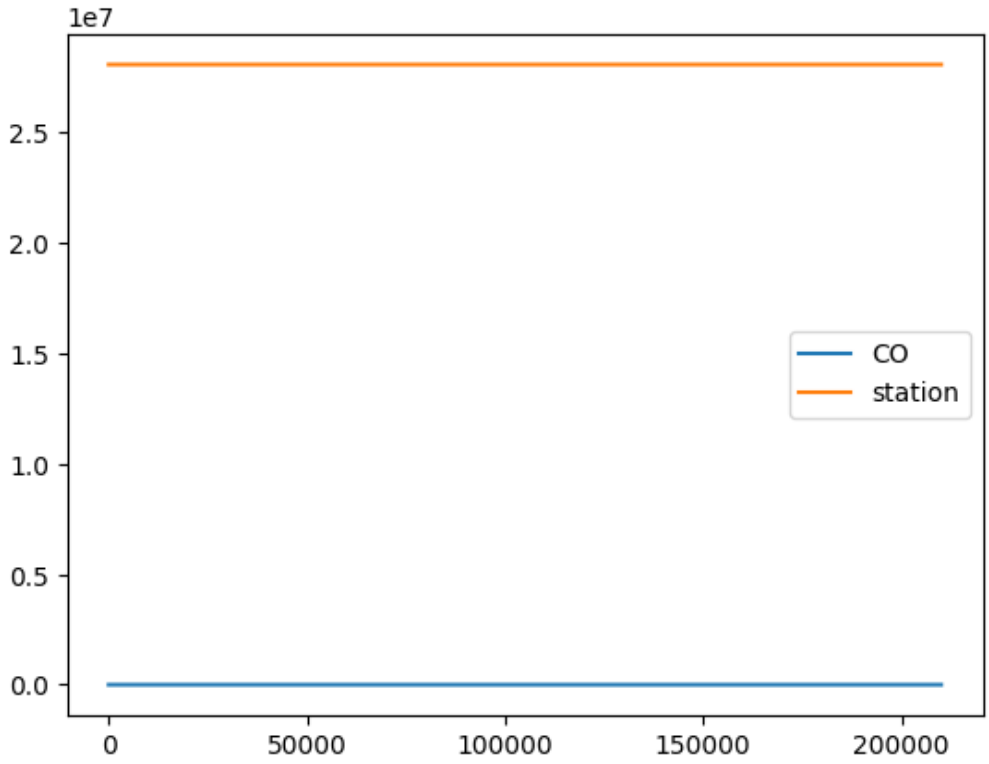
Line chart

In [8]:

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

Out[8]:

<Axes: >



Bar chart

In [9]:

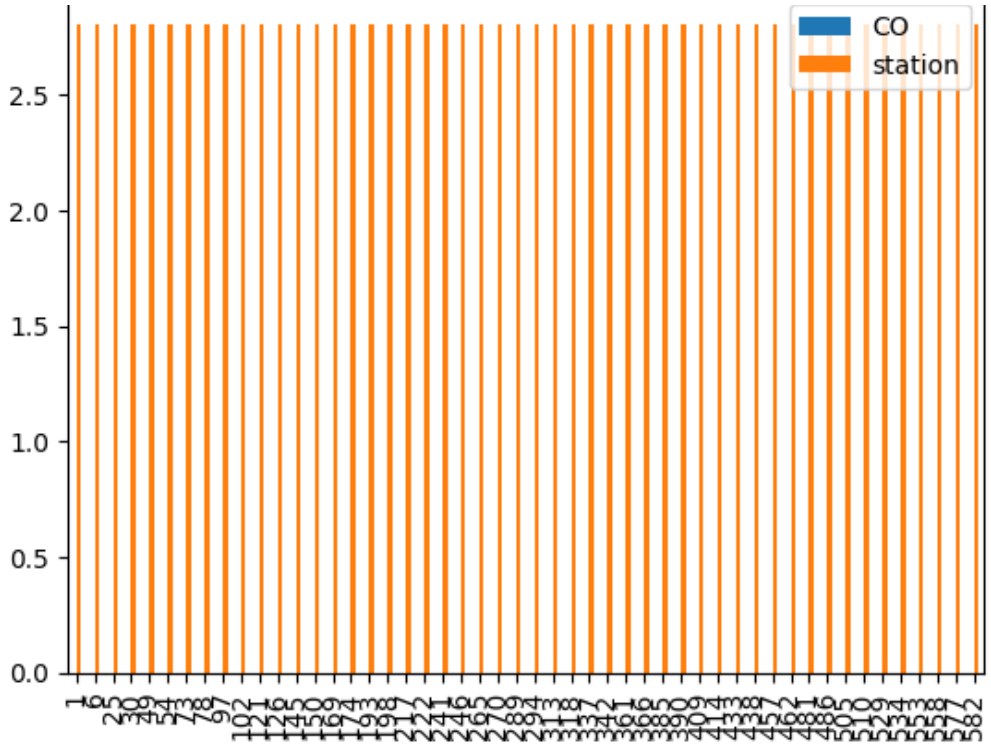
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



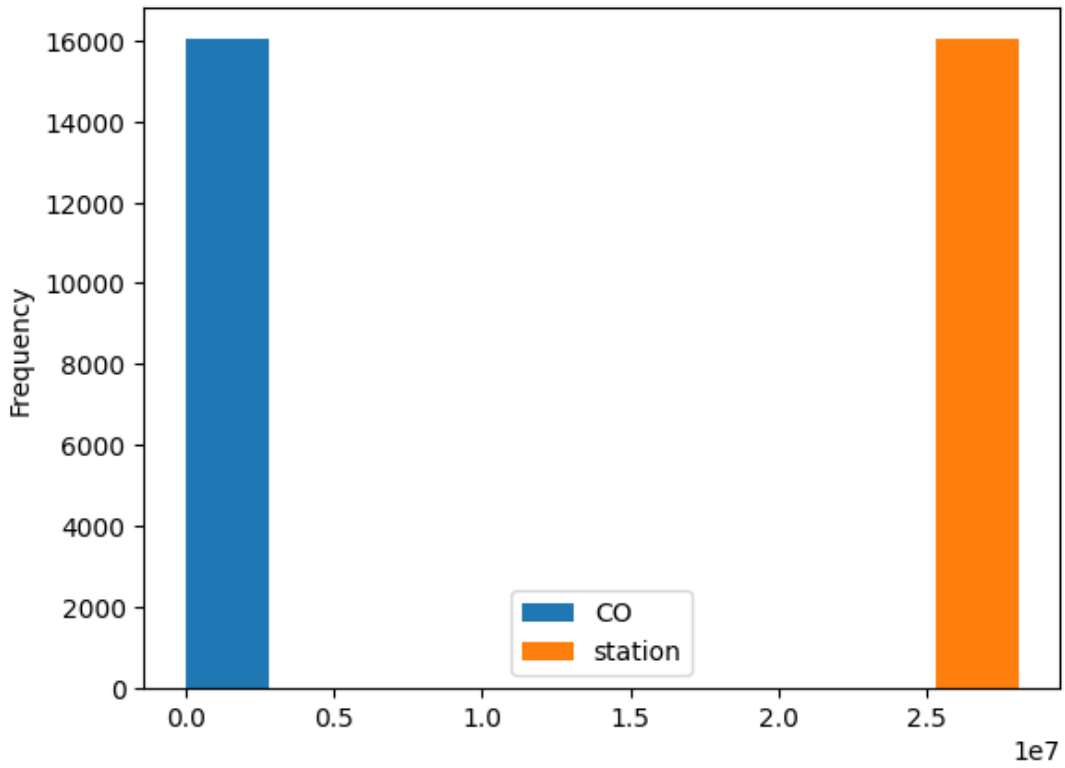
Histogram

In [11]:

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

Out[11]:

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



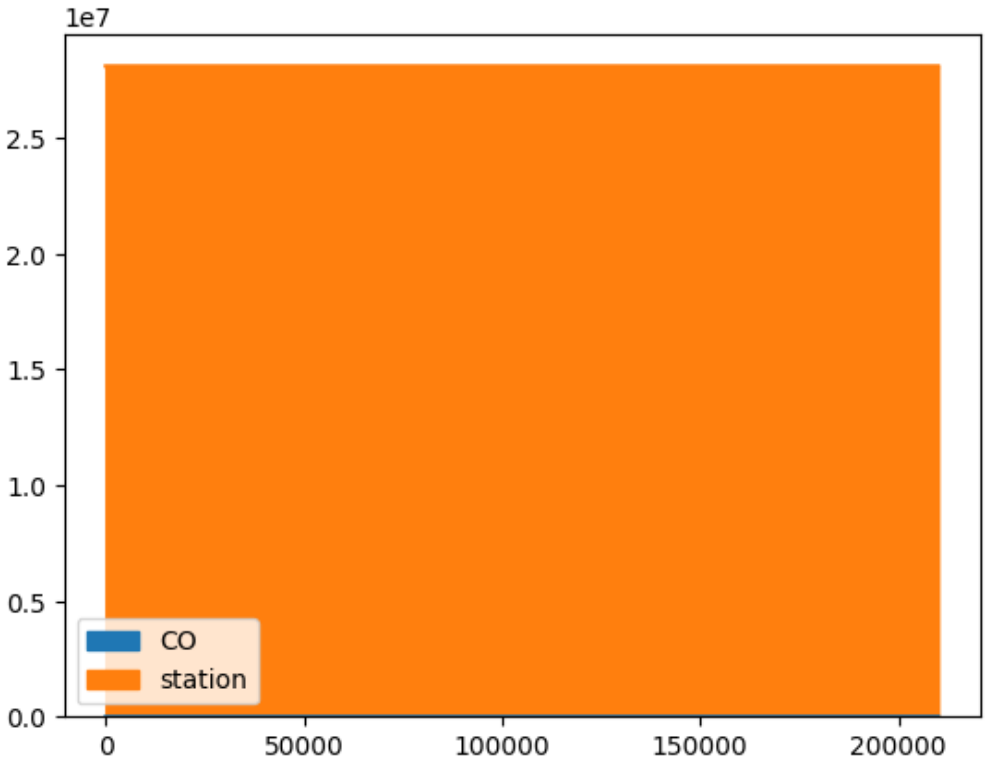
Area chart

In [12]:

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

Out[12]:

<Axes: >



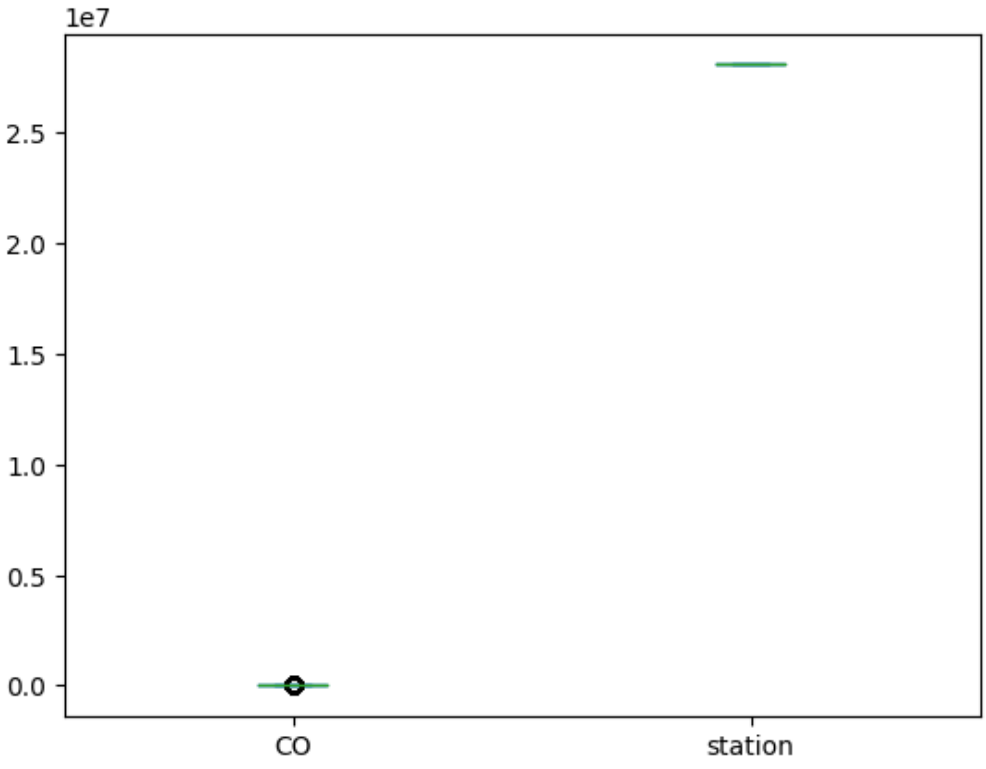
Box chart

In [13]:

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

Out[13]:

<Axes: >



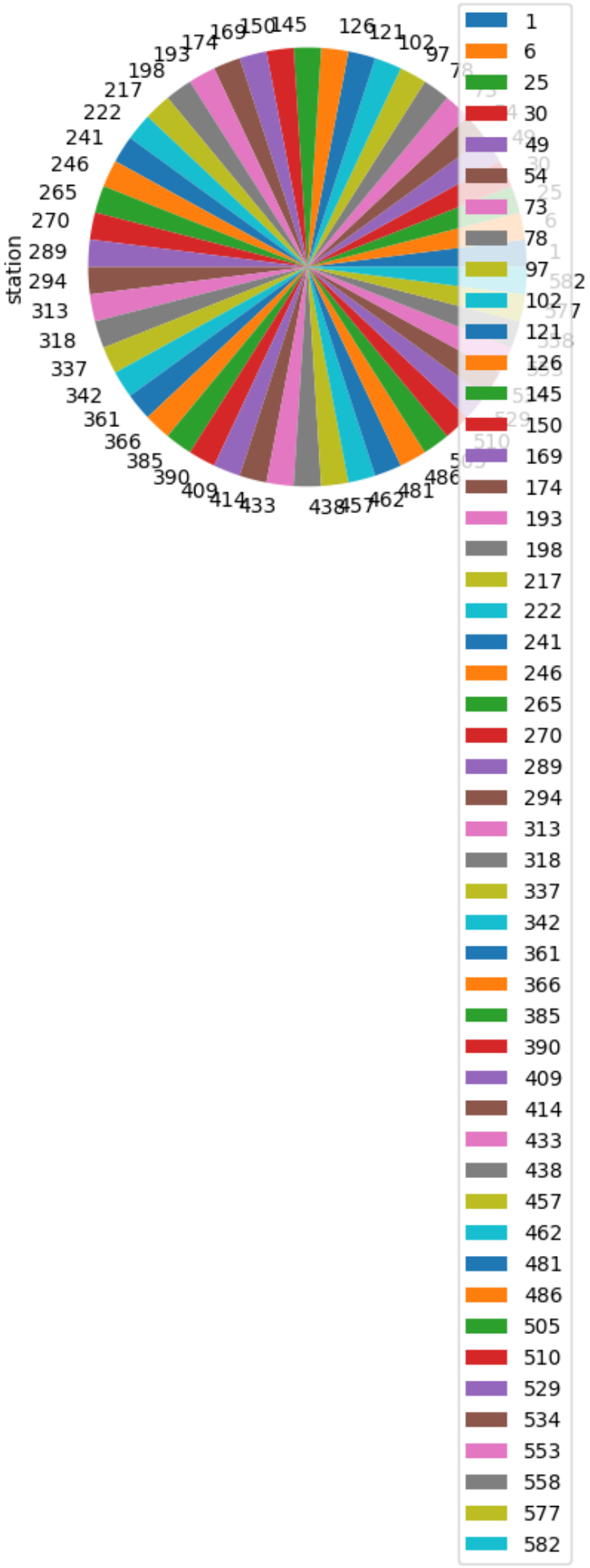
Pie chart

In [14]:

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

Out[14]:

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



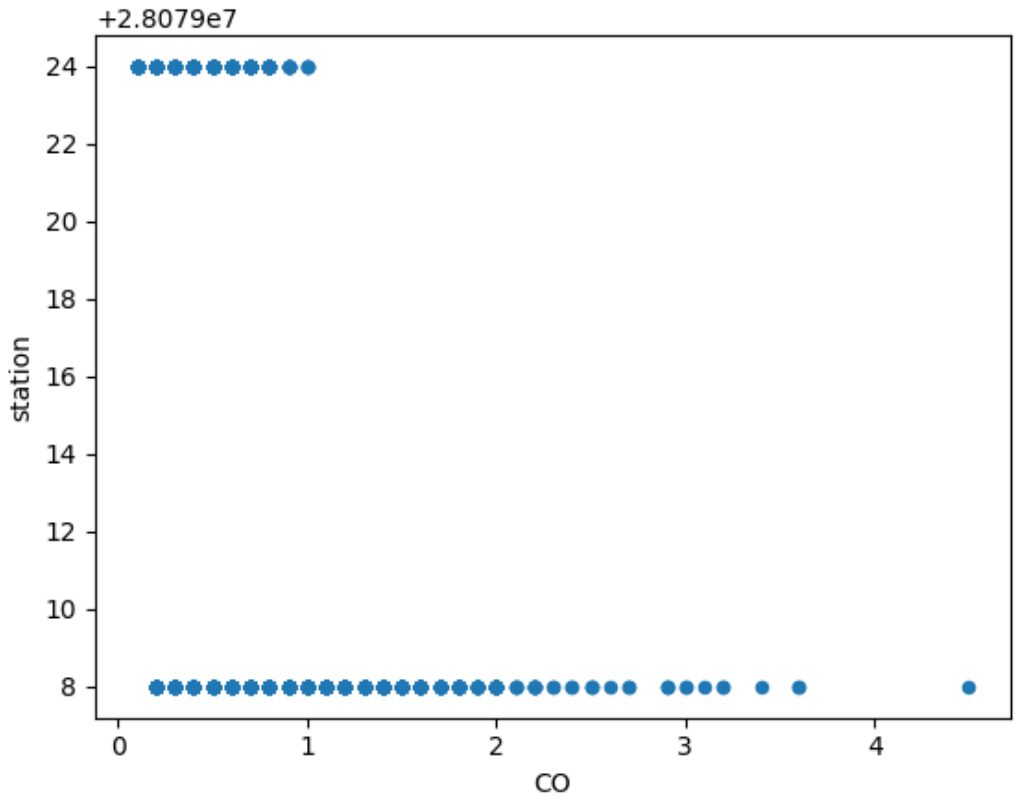
Scatter chart

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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ 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	160
count	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	48.089792	22.183764	
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	35.847298	15.993825	
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000	1.000000	
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	15.000000	11.000000	
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	46.000000	19.000000	
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	73.000000	29.000000	
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	217.000000	196.000000	

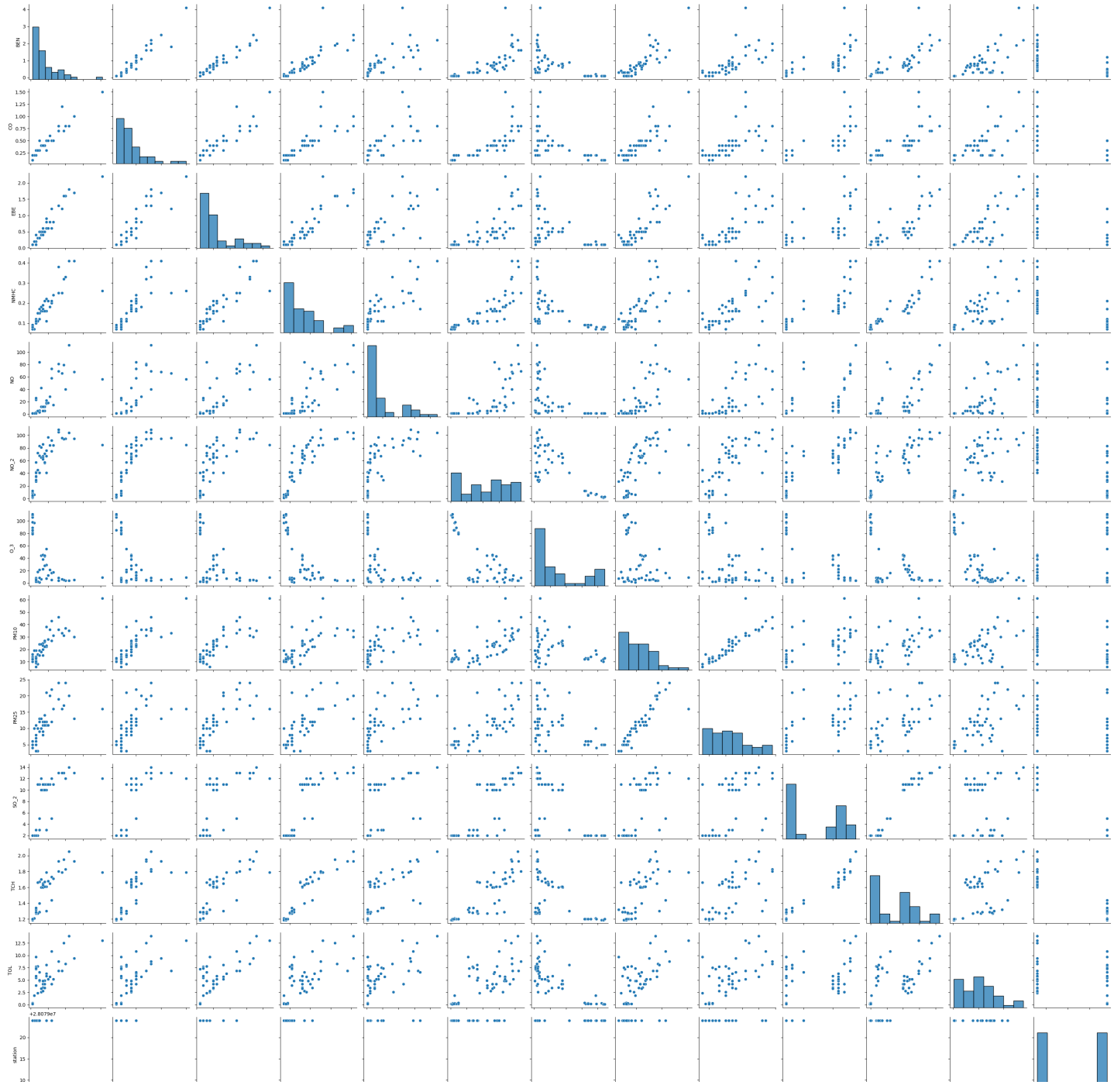
EDA AND VISUALIZATION

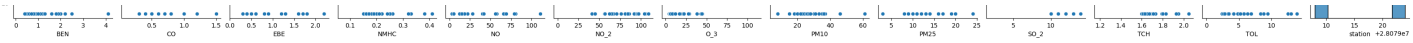
In [18]:

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

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x7a87e695a3b0>
```





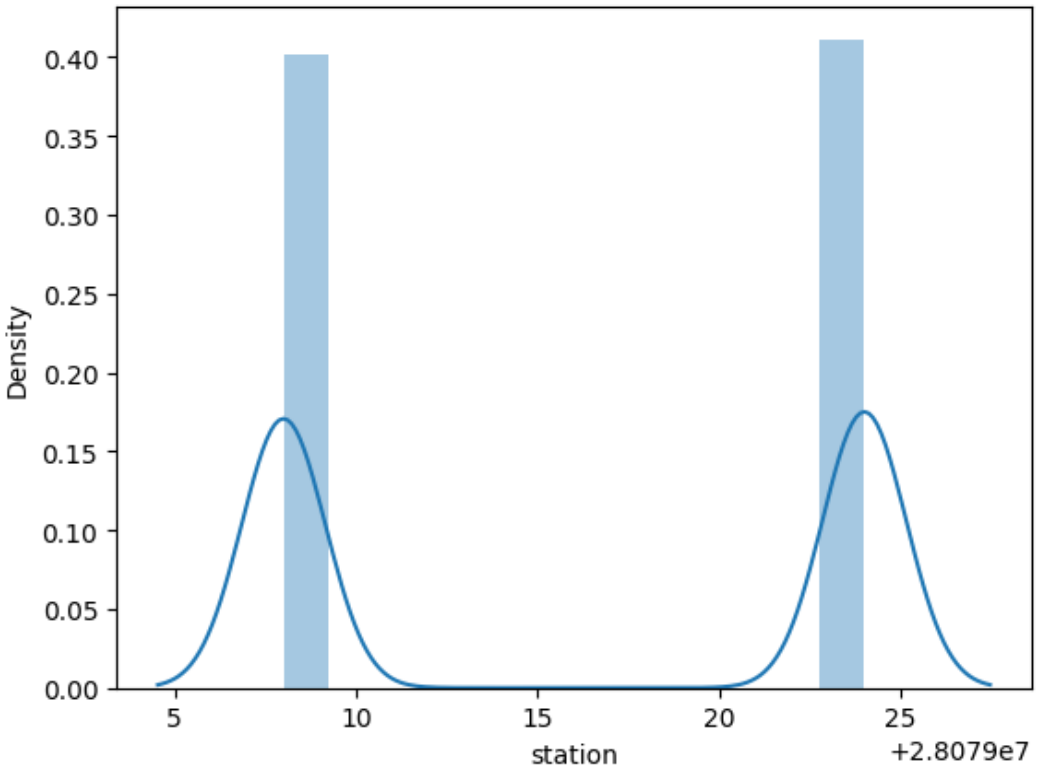
In [19]:

```
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'>
```



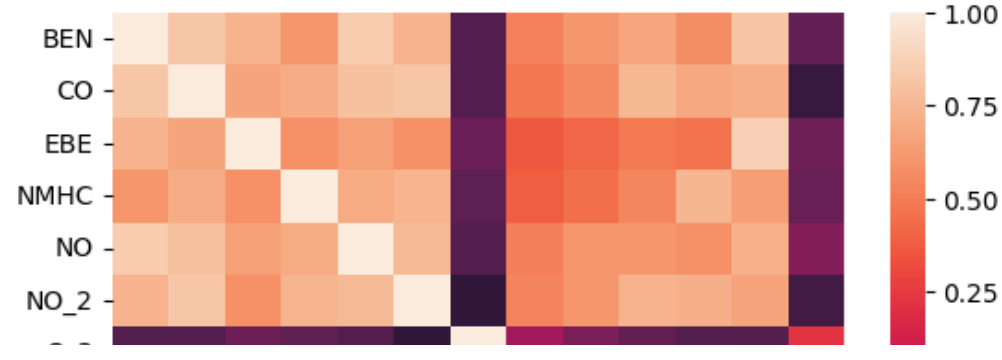
In [20]:

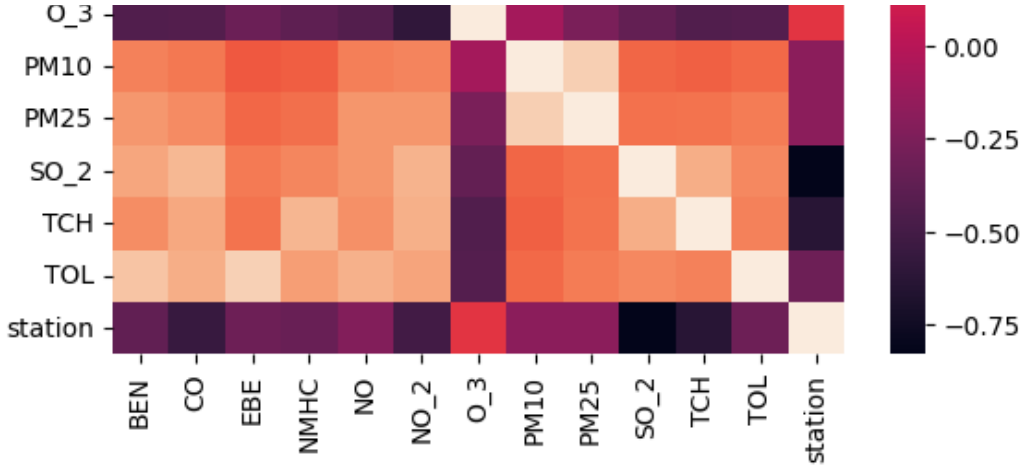
```
sns.heatmap(df.corr())
```

```
<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]:

```
28079038.931514844
```

In [25]:

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

Out[25]:

	Co-efficient
BEN	1.014327
CO	-9.653039
EBE	-0.644831
NMHC	13.306330
NO	0.081571

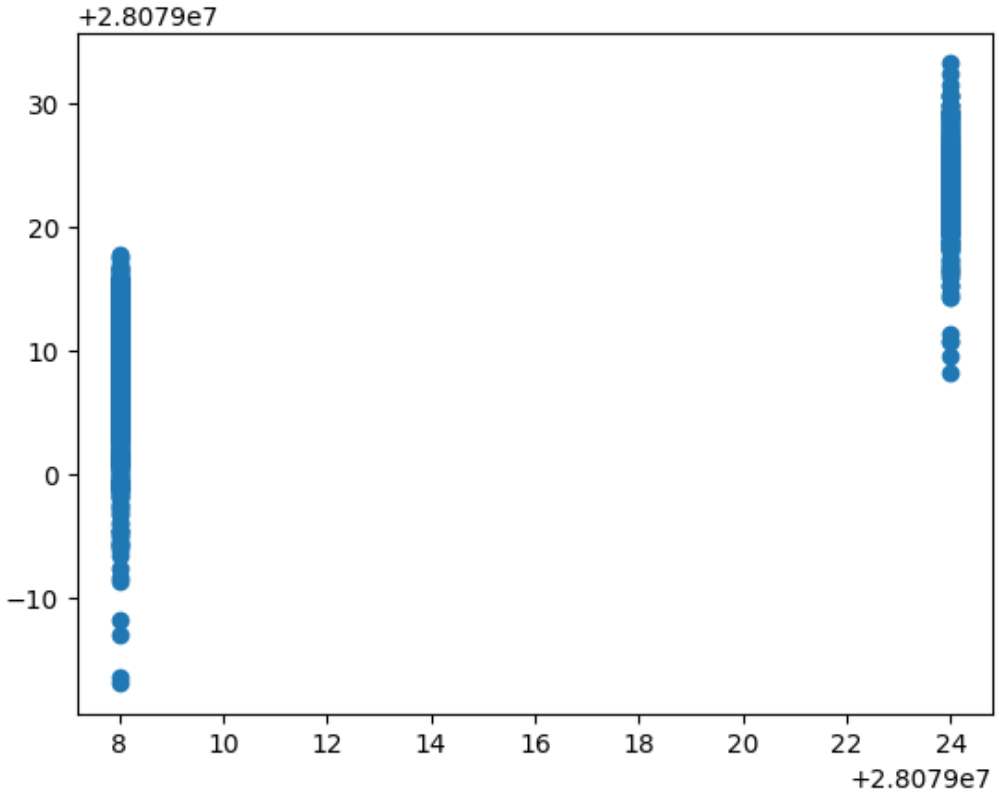
	Coefficient
NO_2	-0.017666
O_3	-0.015317
PM10	0.011994
PM25	0.091290
SO_2	-1.106813
TCH	-10.122372
TOL	-0.092657

In [26]:

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

Out[26]:

```
<matplotlib.collections.PathCollection at 0x7a882cb2bfd0>
```



ACCURACY

In [27]:

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

Out[27]:

```
0.8710527228671661
```

In [28]:

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

Out[28]:

```
0.8719155249862663
```

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]:

```
0.8703938384987232
```

In [32]:

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

Out[32]:

```
0.8711174432841791
```

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]:

```
0.729126123535089
```

Accuracy(Lasso)

In [35]:

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

Out[35]:

```
0.72972460052557
```

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.        ,  0.0742088 ,
       -0.05278603, -0.01248481,  0.02465562,  0.04391838, -1.31143922,
       -0.        , -0.08767773])
```

In [38]:

```
en.intercept_
```

Out[38]:

```
28079026.004801378
```

In [39]:

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

In [40]:

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

Out[40]:

```
0.8241762978180678
```

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)))
```

```
2.5083211469471753
11.252278841930856
3.354441658746036
```

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]:

```
(16026, 12)
```

In [45]:

```
target_vector.shape
```

Out[45]:

```
(16026,)
```

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(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]:

```
0.9971296642955197
```

In [53]:

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

Out[53]:

```
1.0
```

In [54]:

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

Out[54]:

```
array([[1.00000000e+00, 1.54284913e-35]])
```

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]:

```
0.9948297379211981
```

In [60]:

```
rfc_best=grid_search.best_estimator_
```

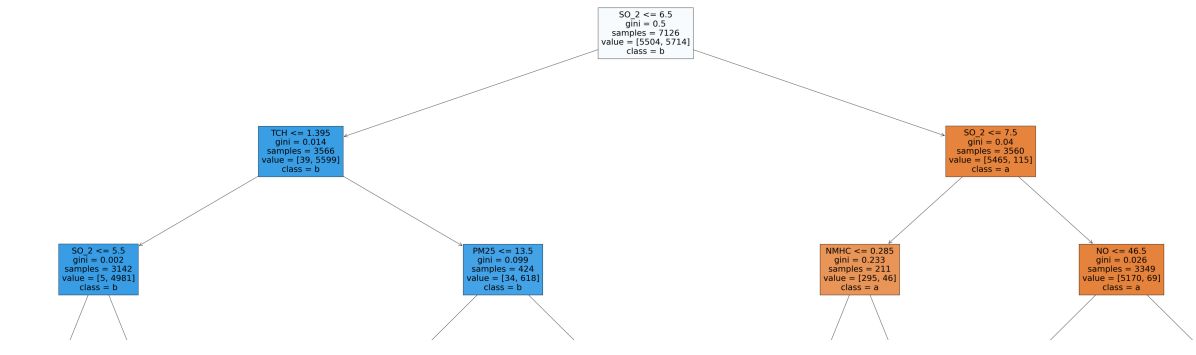
In [61]:

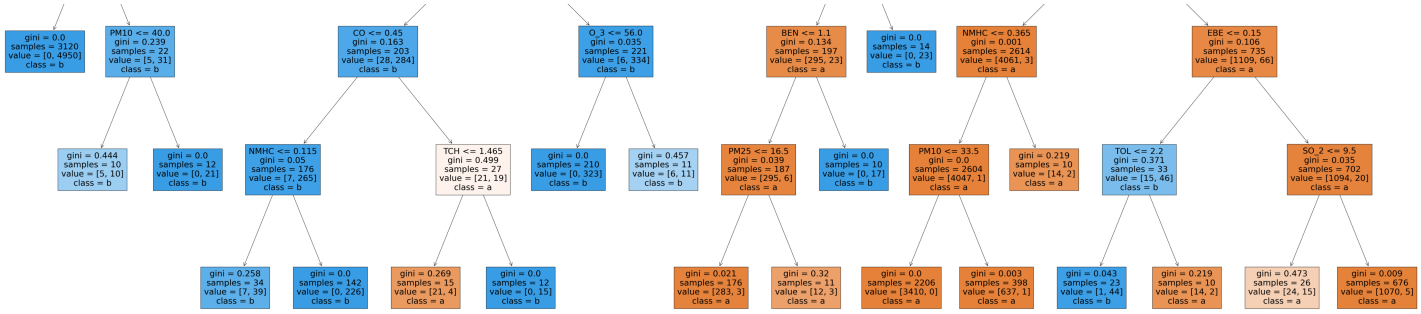
```
from sklearn.tree import plot_tree

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

Out[61]:

```
[Text(0.45, 0.9166666666666666, 'SO_2 <= 6.5\ngini = 0.5\nsamples = 7126\nvalue = [5504, 5714]\nclass = b'),
 Text(0.20833333333333334, 0.75, 'TCH <= 1.395\ngini = 0.014\nsamples = 3566\nvalue = [39, 5599]\nclass = b'),
 Text(0.06666666666666667, 0.5833333333333334, 'SO_2 <= 5.5\ngini = 0.002\nsamples = 3142\nvalue = [5, 4981]\nclass = b'),
 Text(0.03333333333333333, 0.4166666666666667, 'gini = 0.0\nsamples = 3120\nvalue = [0, 4950]\nclass = b'),
 Text(0.1, 0.4166666666666667, 'PM10 <= 40.0\ngini = 0.239\nsamples = 22\nvalue = [5, 31]\nclass = b'),
 Text(0.06666666666666667, 0.25, 'gini = 0.444\nsamples = 10\nvalue = [5, 10]\nclass = b'),
 Text(0.13333333333333333, 0.25, 'gini = 0.0\nsamples = 12\nvalue = [0, 21]\nclass = b'),
 Text(0.35, 0.5833333333333334, 'PM25 <= 13.5\ngini = 0.099\nsamples = 424\nvalue = [34, 618]\nclass = b'),
```

```
 Text(0.26666666666666666, 0.4166666666666667, 'CO <= 0.45\ngini = 0.163\nsamples = 203\n
value = [28, 284]\nclass = b'),
 Text(0.2, 0.25, 'NMHC <= 0.115\ngini = 0.05\nsamples = 176\nvalue = [7, 265]\nclass = b'
),
 Text(0.16666666666666666, 0.08333333333333333, 'gini = 0.258\nsamples = 34\nvalue = [7,
39]\nclass = b'),
 Text(0.23333333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 142\nvalue = [0, 2
26]\nclass = b'),
 Text(0.3333333333333333, 0.25, 'TCH <= 1.465\ngini = 0.499\nsamples = 27\nvalue = [21, 1
9]\nclass = a'),
 Text(0.3, 0.08333333333333333, 'gini = 0.269\nsamples = 15\nvalue = [21, 4]\nclass = a')
,
 Text(0.36666666666666664, 0.08333333333333333, 'gini = 0.0\nsamples = 12\nvalue = [0, 15
]\nclass = b'),
 Text(0.43333333333333335, 0.4166666666666667, 'O_3 <= 56.0\ngini = 0.035\nsamples = 221\
nvalue = [6, 334]\nclass = b'),
 Text(0.4, 0.25, 'gini = 0.0\nsamples = 210\nvalue = [0, 323]\nclass = b'),
 Text(0.4666666666666667, 0.25, 'gini = 0.457\nsamples = 11\nvalue = [6, 11]\nclass = b')
,
 Text(0.6916666666666667, 0.75, 'SO_2 <= 7.5\ngini = 0.04\nsamples = 3560\nvalue = [5465,
115]\nclass = a'),
 Text(0.6, 0.5833333333333334, 'NMHC <= 0.285\ngini = 0.233\nsamples = 211\nvalue = [295,
46]\nclass = a'),
 Text(0.5666666666666667, 0.4166666666666667, 'BEN <= 1.1\ngini = 0.134\nsamples = 197\nv
alue = [295, 23]\nclass = a'),
 Text(0.5333333333333333, 0.25, 'PM25 <= 16.5\ngini = 0.039\nsamples = 187\nvalue = [295,
6]\nclass = a'),
 Text(0.5, 0.08333333333333333, 'gini = 0.021\nsamples = 176\nvalue = [283, 3]\nclass = a
'),
 Text(0.5666666666666667, 0.08333333333333333, 'gini = 0.32\nsamples = 11\nvalue = [12, 3
]\nclass = a'),
 Text(0.6, 0.25, 'gini = 0.0\nsamples = 10\nvalue = [0, 17]\nclass = b'),
 Text(0.6333333333333333, 0.4166666666666667, 'gini = 0.0\nsamples = 14\nvalue = [0, 23]\
nclass = b'),
 Text(0.7833333333333333, 0.5833333333333334, 'NO <= 46.5\ngini = 0.026\nsamples = 3349\n
value = [5170, 69]\nclass = a'),
 Text(0.7, 0.4166666666666667, 'NMHC <= 0.365\ngini = 0.001\nsamples = 2614\nvalue = [406
1, 3]\nclass = a'),
 Text(0.6666666666666666, 0.25, 'PM10 <= 33.5\ngini = 0.0\nsamples = 2604\nvalue = [4047,
1]\nclass = a'),
 Text(0.6333333333333333, 0.08333333333333333, 'gini = 0.0\nsamples = 2206\nvalue = [3410
, 0]\nclass = a'),
 Text(0.7, 0.08333333333333333, 'gini = 0.003\nsamples = 398\nvalue = [637, 1]\nclass = a
'),
 Text(0.7333333333333333, 0.25, 'gini = 0.219\nsamples = 10\nvalue = [14, 2]\nclass = a')
,
 Text(0.8666666666666667, 0.4166666666666667, 'EBE <= 0.15\ngini = 0.106\nsamples = 735\n
value = [1109, 66]\nclass = a'),
 Text(0.8, 0.25, 'TOL <= 2.2\ngini = 0.371\nsamples = 33\nvalue = [15, 46]\nclass = b'),
 Text(0.7666666666666667, 0.08333333333333333, 'gini = 0.043\nsamples = 23\nvalue = [1, 4
4]\nclass = b'),
 Text(0.8333333333333334, 0.08333333333333333, 'gini = 0.219\nsamples = 10\nvalue = [14,
2]\nclass = a'),
 Text(0.9333333333333333, 0.25, 'SO_2 <= 9.5\ngini = 0.035\nsamples = 702\nvalue = [1094,
20]\nclass = a'),
 Text(0.9, 0.08333333333333333, 'gini = 0.473\nsamples = 26\nvalue = [24, 15]\nclass = a'
),
 Text(0.9666666666666667, 0.08333333333333333, 'gini = 0.009\nsamples = 676\nvalue = [107
0, 5]\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: 0.8710527228671661
Ridge Regression: 0.8703938384987232
Lasso Regression 0.72972460052557
ElasticNet Regression: 0.8241762978180678
Logistic Regression: 0.9971296642955197
Random Forest: 0.9948297379211981
```

Logistic Regression is suitable for this dataset