20104169 - SUMESH R

Importing Libraries

In [10]:

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

In [11]:

```
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid_2001.csv")
df
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

Out[11]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY	SO_2	TCH
0	2001-08-01 01:00:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	105.000000	NaN	6.340000	NaN
1	2001-08-01 01:00:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.599998	1.73	8.110000	1.24
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.099998	NaN	7.850000	NaN
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.779999	NaN	6.460000	NaN
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.180000	NaN	8.800000	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
217867	2001-04-01 00:00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	47.880001	NaN	39.910000	NaN
217868	2001-04-01 00:00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	26.809999	NaN	13.450000	1.32
217869	2001-04-01 00:00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.770000	0.61	14.700000	1.40
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	37.889999	4.31	39.919998	NaN
217871	2001-04-01 00:00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.369999	4.95	27.340000	1.41

217872 rows × 16 columns

Data Cleaning and Data Preprocessing

In [12]:

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

In [13]:

```
df.columns
```

Out[13]:

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

In [14]:

```
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 [15]:

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

Out[15]:

	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

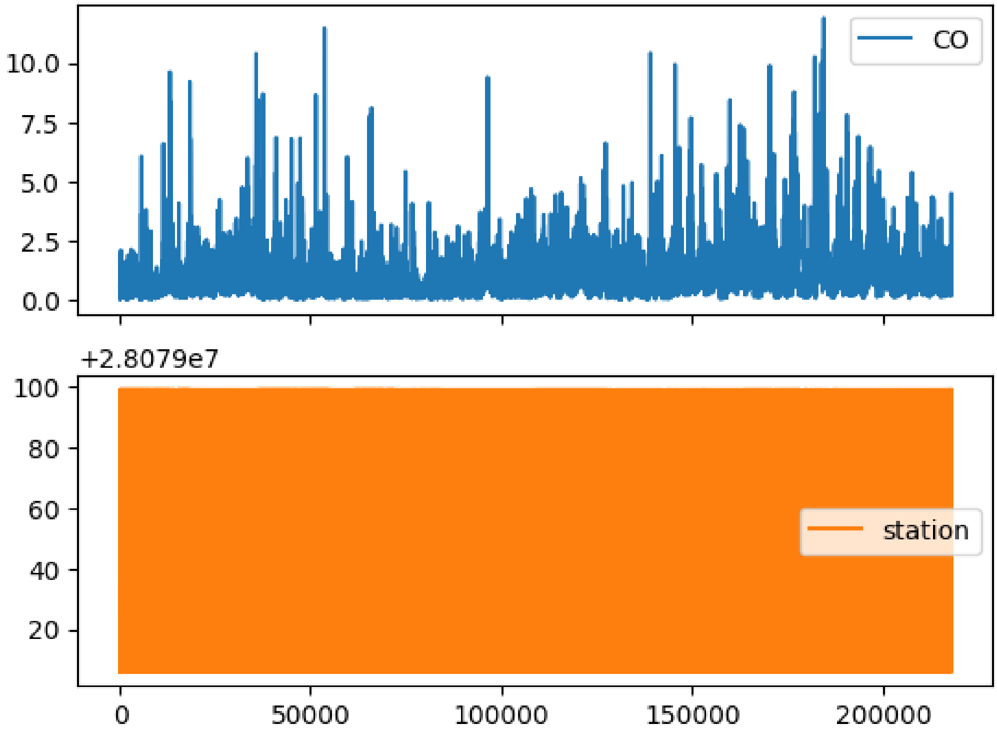
Line chart

In [16]:

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

Out[16]:

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



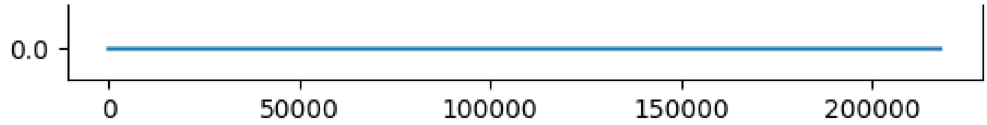
Line chart

In [17]:

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

Out[17]:

```
<Axes: >
```



Bar chart

In [18]:

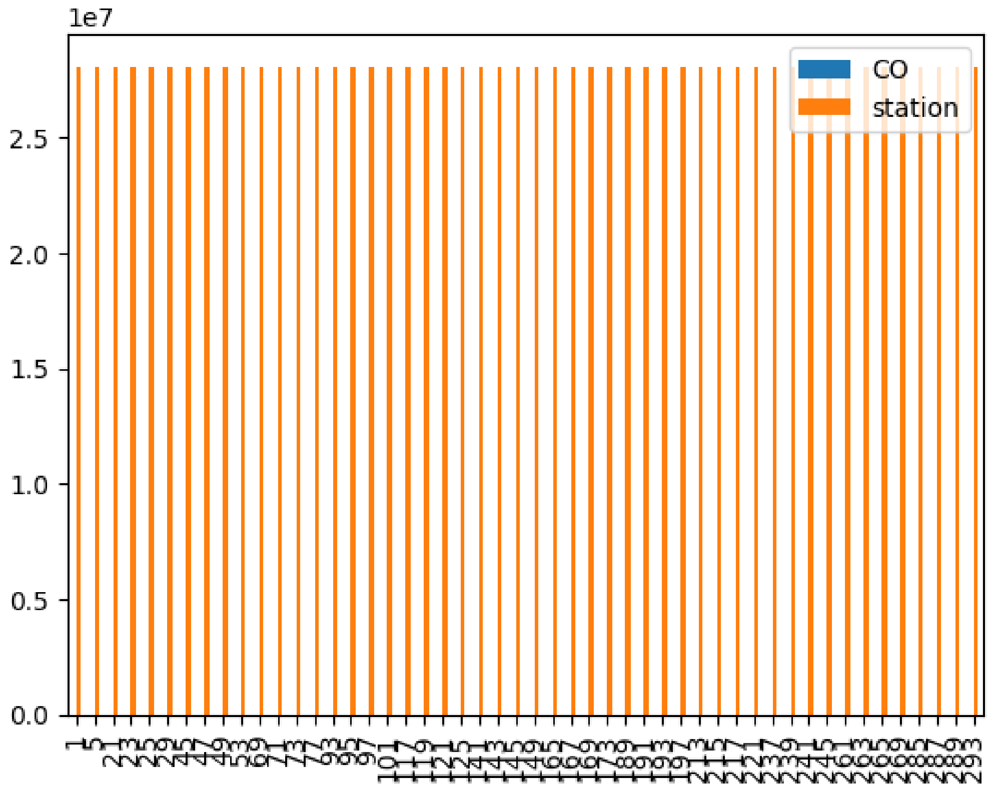
```
b=data[0:50]
```

In [19]:

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

Out[19]:

```
<Axes: >
```



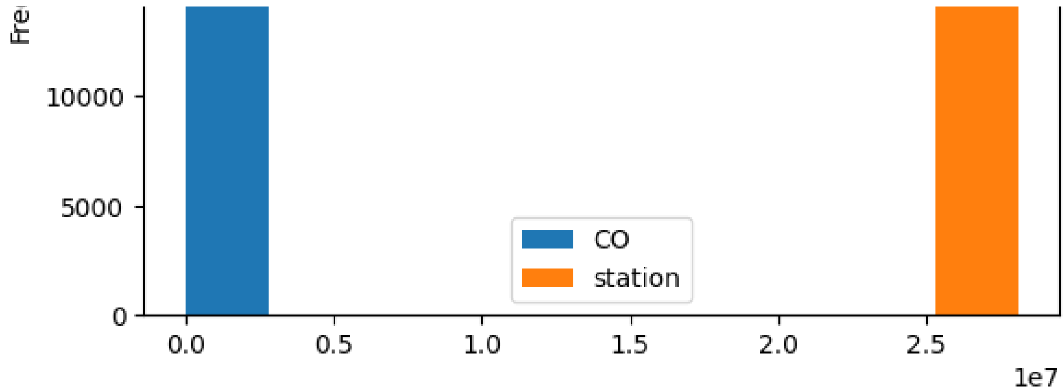
Histogram

In [20]:

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

Out[20]:

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



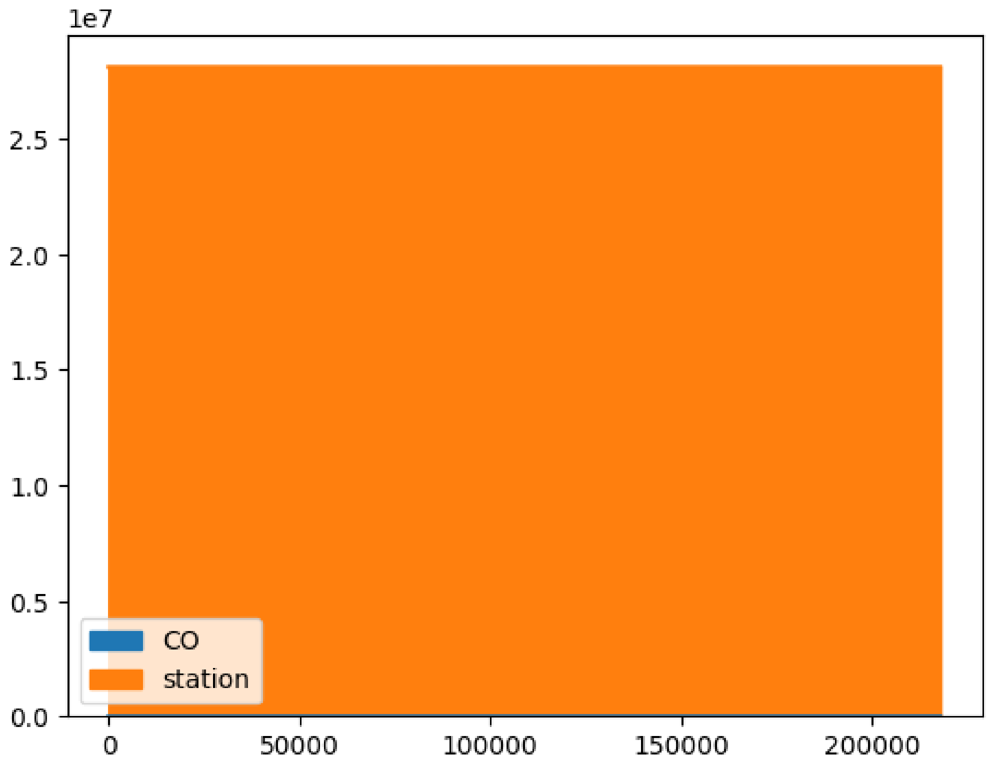
Area chart

In [21]:

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

Out[21]:

<Axes: >



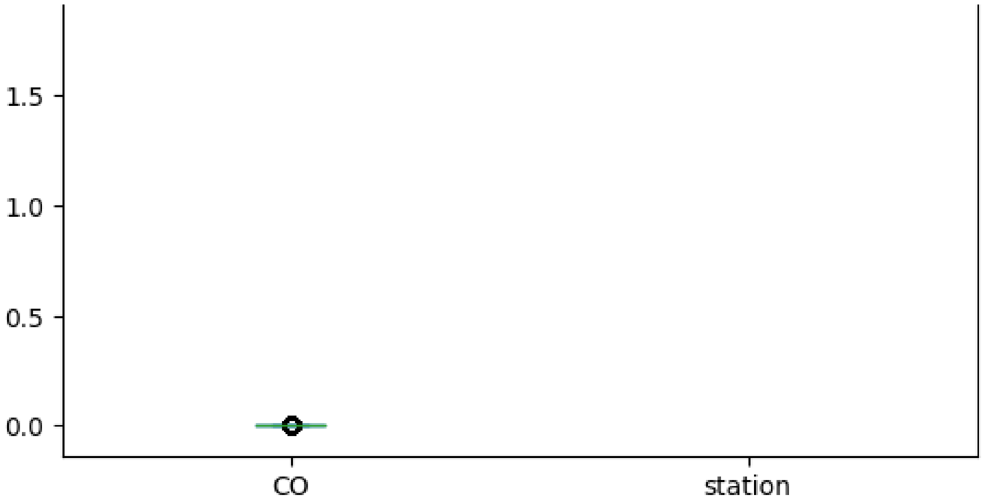
Box chart

In [22]:

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

Out[22]:

<Axes: >



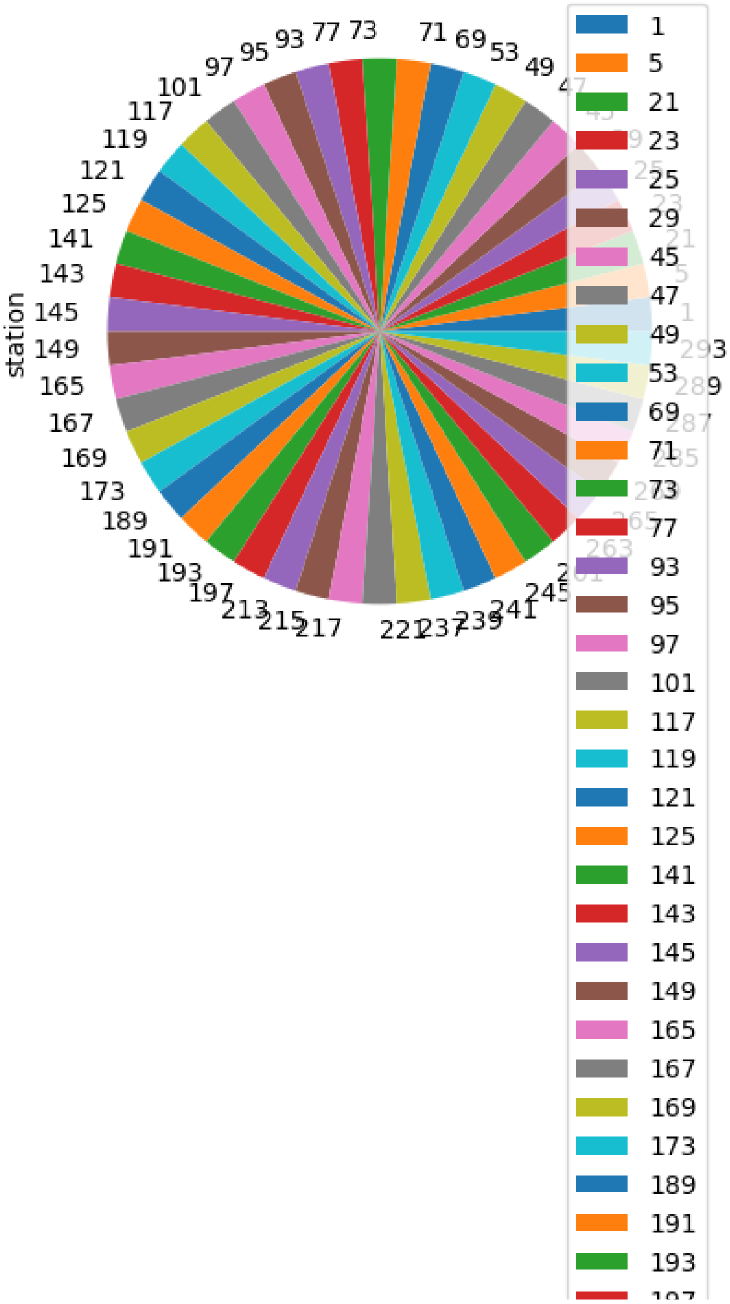
Pie chart

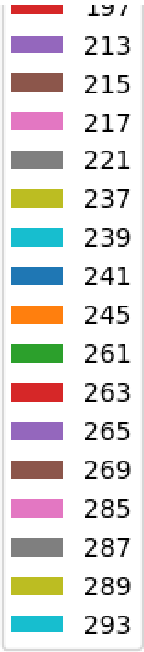
In [23]:

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

Out[23]:

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





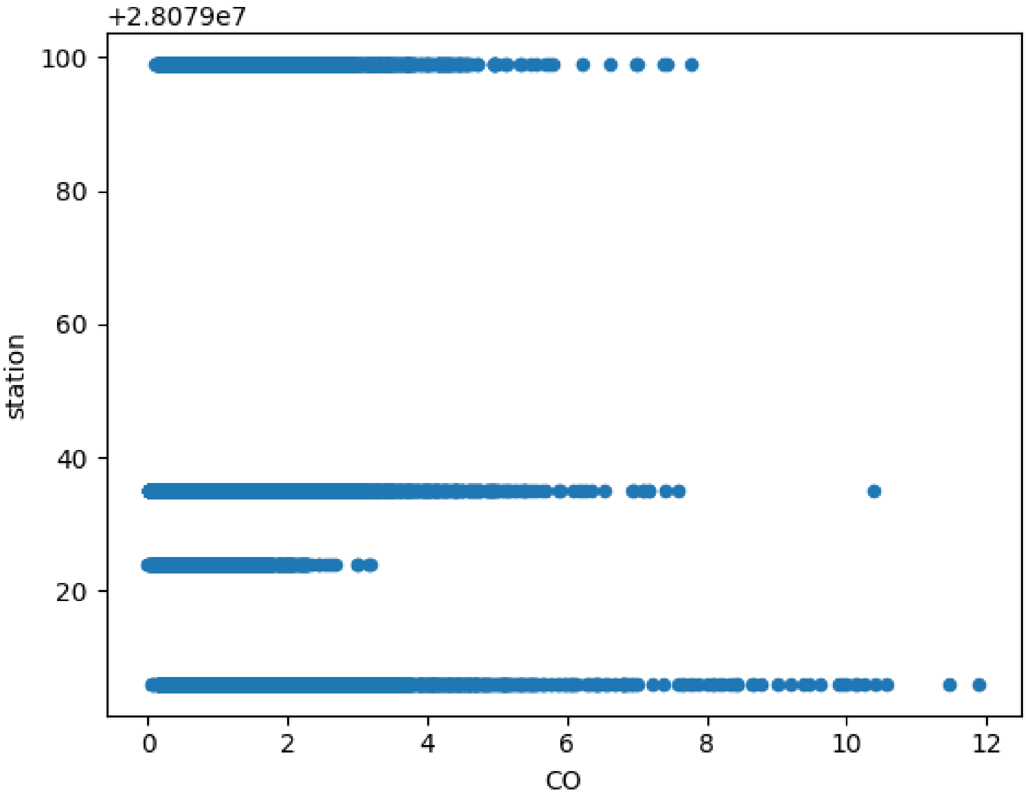
Scatter chart

In [24]:

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

Out[24]:

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



In [25]:

```
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
```

```
 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 [26]:

```
df.describe()
```

Out[26]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	296
mean	3.361895	1.005413	3.580229	8.113086	0.195222	67.652292	163.004858	3.736694	
std	3.176669	0.863135	3.744496	7.909701	0.192585	34.003120	147.491777	3.702559	
min	0.100000	0.000000	0.140000	0.210000	0.000000	1.180000	1.280000	0.190000	
25%	1.280000	0.470000	1.390000	3.040000	0.080000	44.299999	68.089996	1.480000	
50%	2.510000	0.760000	2.600000	5.830000	0.140000	64.449997	123.699997	2.730000	
75%	4.420000	1.270000	4.580000	10.640000	0.250000	86.540001	213.199997	4.830000	
max	54.560001	11.890000	77.260002	150.600006	2.880000	292.700012	1940.000000	89.510002	1

In [27]:

```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
        'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

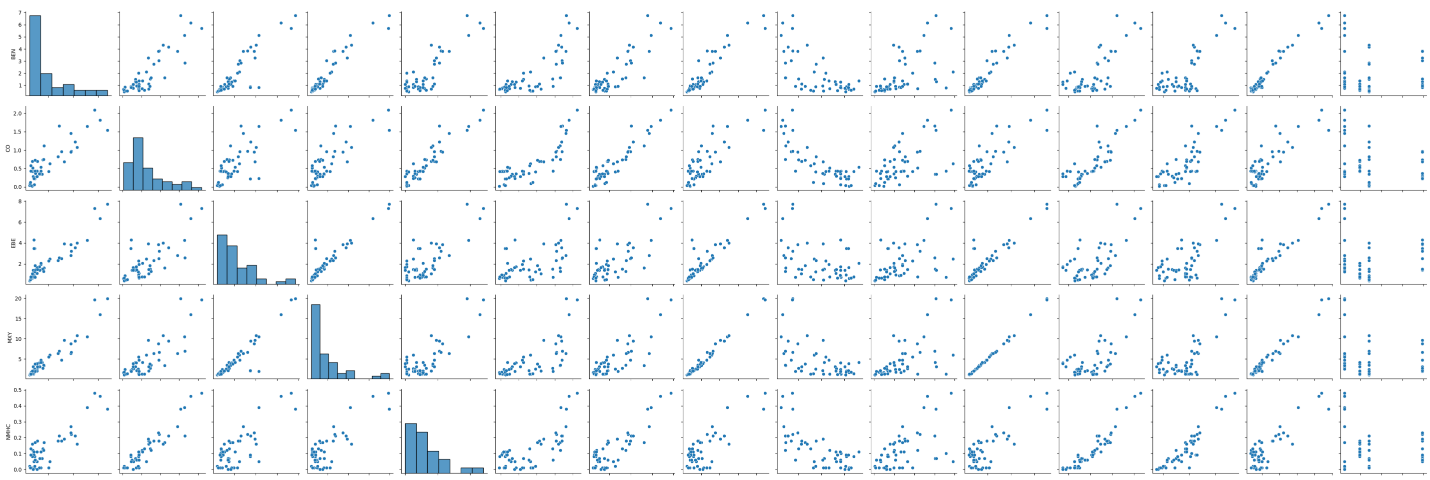
EDA AND VISUALIZATION

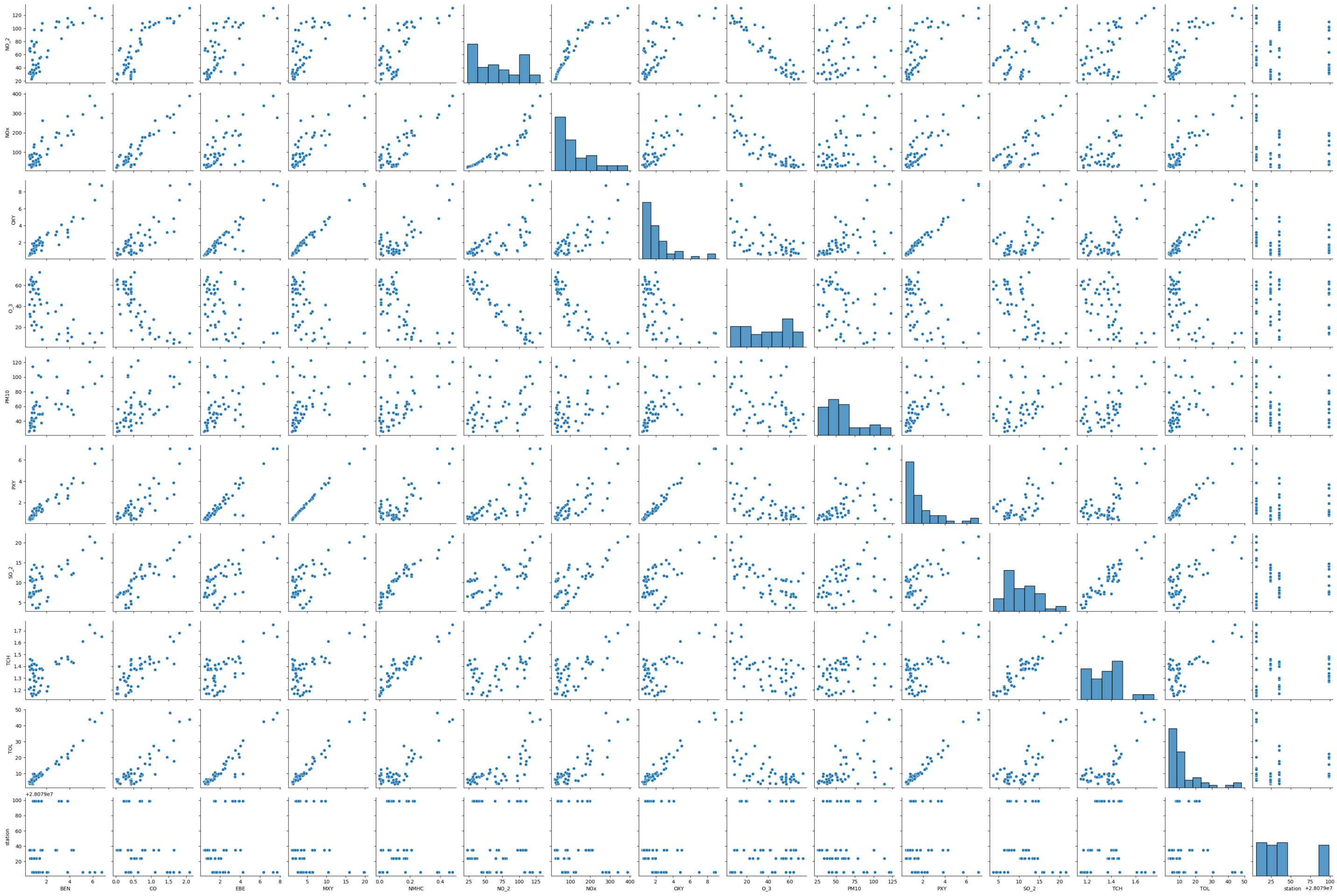
In [28]:

```
sns.pairplot(df1[0:50])
```

Out[28]:

<seaborn.axisgrid.PairGrid at 0x7f253b3678e0>





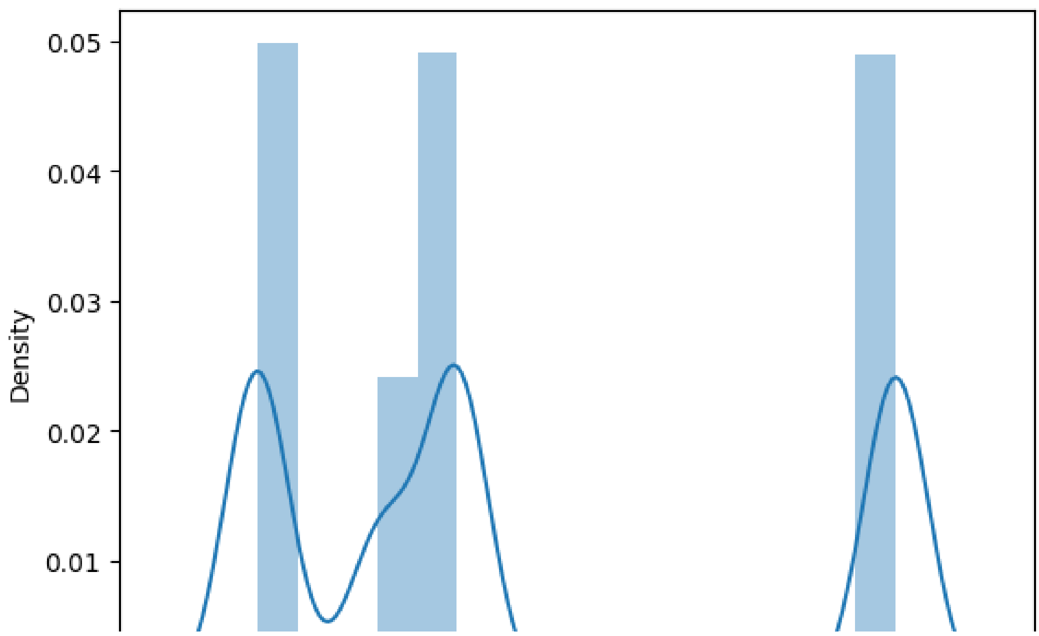
In [29]:

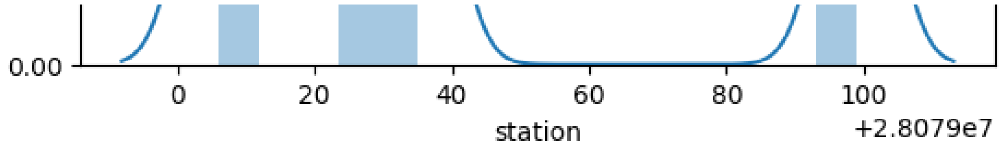
```
sns.distplot(df1['station'])
```

```
<ipython-input-29-4bc330f7257f>: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(df1['station'])
```

Out[29]:

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



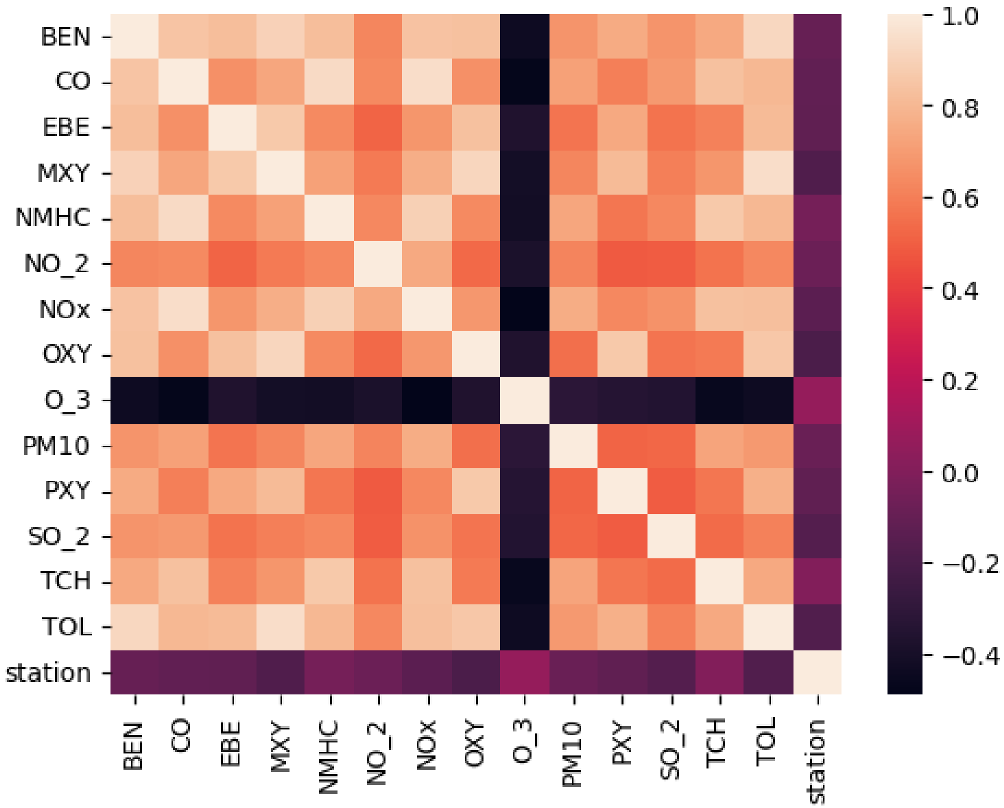


In [30]:

```
sns.heatmap(df1.corr())
```

Out[30]:

```
<Axes: >
```



TO TRAIN THE MODEL AND MODEL BULDING

In [31]:

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

In [32]:

```
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 [33]:

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

Out[33]:

```
▾ LinearRegression
LinearRegression()
```

In [34]:

```
lr.intercept_
```

Out[34]:

```
28079009.16482375
```

In [35]:

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

Out[35]:

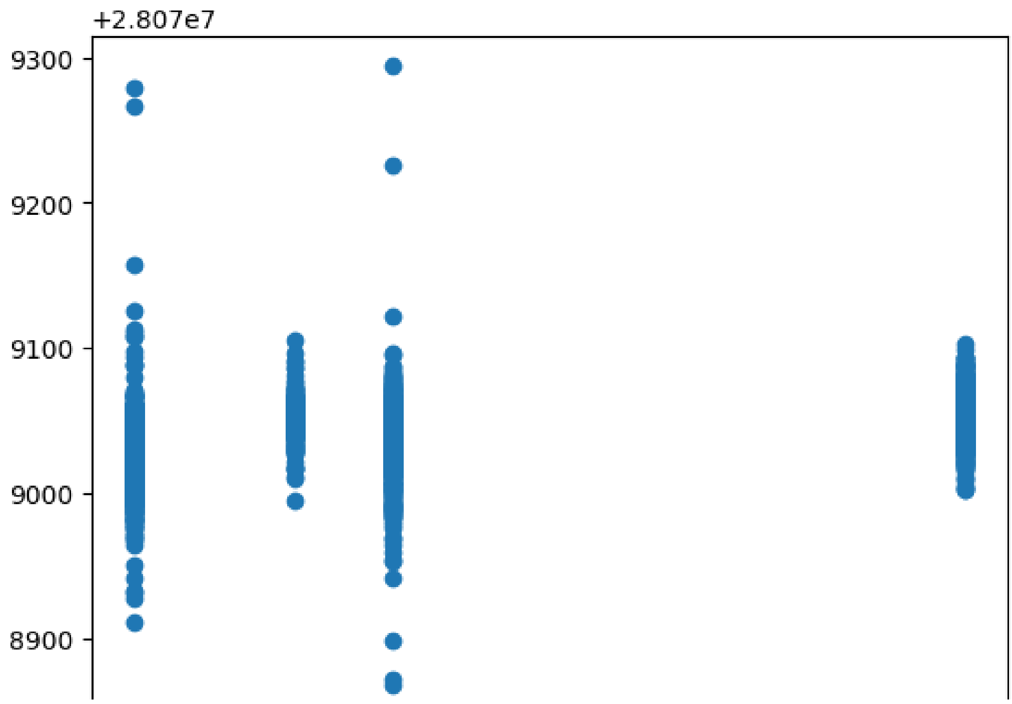
	Co-efficient
BEN	7.795875
CO	-17.273985
EBE	0.516764
MXY	0.066244
NMHC	89.263149
NO_2	0.110278
NOx	-0.078486
OXY	-2.942643
O_3	-0.024871
PM10	-0.061212
PXY	1.175102
SO_2	-0.309099
TCH	35.328353
TOL	-1.362196

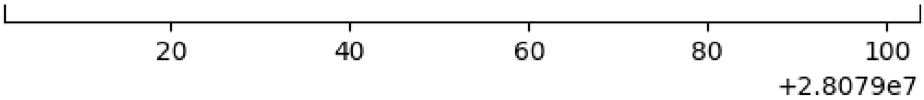
In [36]:

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

Out[36]:

```
<matplotlib.collections.PathCollection at 0x7f2520f67a00>
```





ACCURACY

In [37]:

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

Out[37]:

```
0.1560981914762234
```

In [38]:

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

Out[38]:

```
0.16747734739320896
```

Ridge and Lasso

In [39]:

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

In [40]:

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

Out[40]:

```
▾ Ridge
Ridge(alpha=10)
```

Accuracy(Ridge)

In [41]:

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

Out[41]:

```
0.15714896786525434
```

In [42]:

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

Out[42]:

```
0.16717492584275784
```

In [43]:

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

Out[43]:

```
▾ Lasso
Lasso(alpha=10)
```

In [44]:

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

Out[44]:

```
0.036138853604019805
```

Accuracy(Lasso)

In [45]:

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

Out[45]:

```
0.04533993694040095
```

In [46]:

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

Out[46]:

```
▾ ElasticNet
ElasticNet()
```

In [47]:

```
en.coef_
```

Out[47]:

```
array([ 5.26902786,  0.        ,  0.51992889, -0.1321379 ,  0.11319397,
        0.06472817, -0.03285392, -2.41130199, -0.02691663,  0.07280831,
        0.73040554, -0.32944933,  1.1859783 , -0.73889907])
```

In [48]:

```
en.intercept_
```

Out[48]:

```
28079048.462195482
```

In [49]:

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

In [50]:

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

Out[50]:

```
0.10478612906640628
```

Evaluation Metrics

In [51]:

```
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)))
```

```
30.32339812894667
1211.7776248895525
34.81059644547264
```

Logistic Regression

In [52]:

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

In [53]:

```
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 [54]:

```
feature_matrix.shape
```

Out[54]:

```
(29669, 14)
```

In [55]:

```
target_vector.shape
```

Out[55]:

```
(29669,)
```

In [56]:

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

In [57]:

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

In [58]:

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

Out[58]:

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

In [59]:

```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [60]:

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

```
[28079035]
```

In [61]:

```
logr.classes_
```

Out[61]:

```
array([28079006, 28079024, 28079035, 28079099])
```

In [62]:

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

Out[62]:

```
0.8086892042198928
```

In [63]:

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

Out[63]:

```
1.802621471329542e-43
```

In [64]:

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

Out[64]:

```
array([[1.80262147e-43, 2.18514449e-56, 9.99998556e-01, 1.44365621e-06]])
```

Random Forest

In [65]:

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

In [66]:

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

Out[66]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [67]:

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

In [68]:

```
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[68]:

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

In [69]:

```
grid_search.best_score_
```

Out[69]:

```
0.7368547765793528
```

In [70]:

```
rfc_best=grid_search.best_estimator_
```

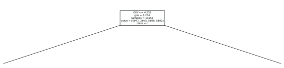
In [71]:

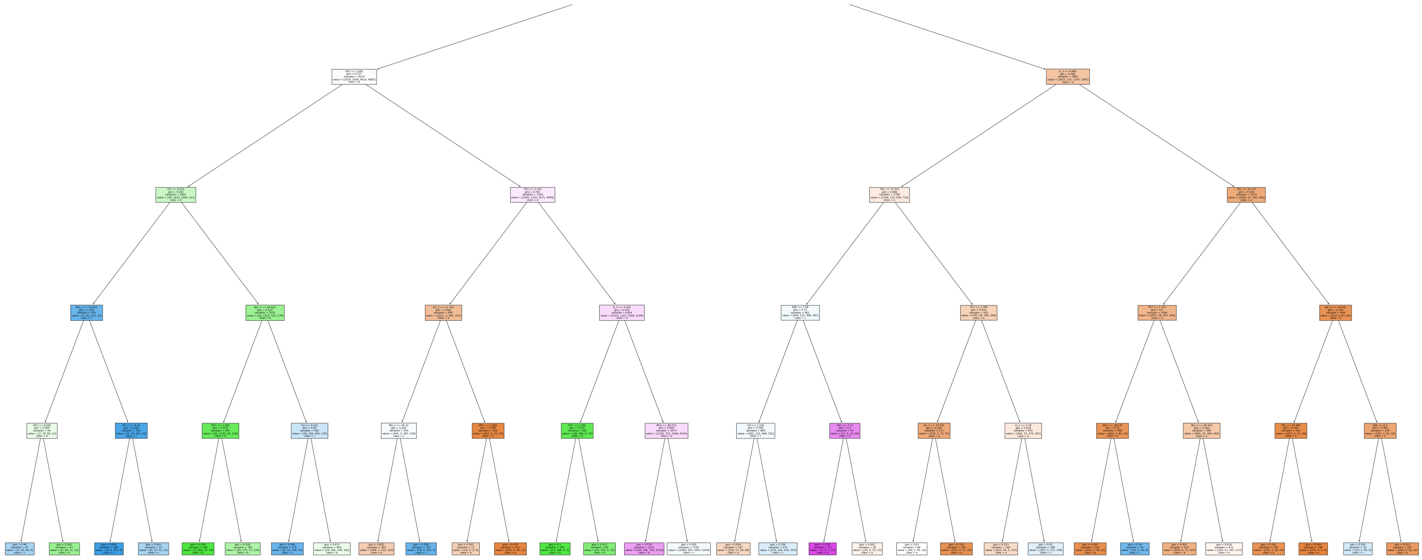
```
from sklearn.tree import plot_tree

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[71]:

```
[Text(0.5, 0.9166666666666666, 'OXY <= 4.255\ngini = 0.734\nsamples = 13155\nvalue = [594
7, 2943, 5986, 5892]\nclass = c'),
 Text(0.25, 0.75, 'OXY <= 1.005\ngini = 0.727\nsamples = 9214\nvalue = [2314, 2768, 4616,
4805]\nclass = d'),
 Text(0.125, 0.5833333333333334, 'CO <= 0.215\ngini = 0.583\nsamples = 1960\nvalue = [49,
1618, 1045, 311]\nclass = b'),
 Text(0.0625, 0.4166666666666667, 'NO_2 <= 19.815\ngini = 0.341\nsamples = 435\nvalue = [
7, 93, 525, 33]\nclass = c'),
 Text(0.03125, 0.25, 'OXY <= 0.645\ngini = 0.604\nsamples = 94\nvalue = [7, 70, 60, 11]\n
class = b'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.48\nsamples = 47\nvalue = [2, 26, 49, 0]\n
class = c'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.563\nsamples = 47\nvalue = [5, 44, 11, 11]
\nclass = b'),
 Text(0.09375, 0.25, 'SO_2 <= 8.43\ngini = 0.165\nsamples = 341\nvalue = [0, 23, 465, 22]
\nclass = c'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.042\nsamples = 290\nvalue = [0, 0, 413, 9]
\nclass = c'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.561\nsamples = 51\nvalue = [0, 23, 52, 13]
\nclass = c'),
 Text(0.1875, 0.4166666666666667, 'NO_2 <= 38.025\ngini = 0.522\nsamples = 1525\nvalue =
[42, 1525, 520, 278]\nclass = b'),
 Text(0.15625, 0.25, 'MXY <= 1.045\ngini = 0.274\nsamples = 943\nvalue = [22, 1231, 64, 1
39]\nclass = b'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.076\nsamples = 640\nvalue = [2, 956, 27, 1
0]\nclass = b'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.558\nsamples = 303\nvalue = [20, 275, 37,
129]\nclass = b'),
 Text(0.21875, 0.25, 'CO <= 0.415\ngini = 0.62\nsamples = 582\nvalue = [20, 294, 456, 139
]\nclass = c'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.346\nsamples = 178\nvalue = [6, 13, 218, 3
7]\nclass = c'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.637\nsamples = 404\nvalue = [14, 281, 238,
102]\nclass = b'),
 Text(0.375, 0.5833333333333334, 'TCH <= 1.235\ngini = 0.701\nsamples = 7254\nvalue = [22
65, 1150, 3571, 4494]\nclass = d'),
 Text(0.3125, 0.4166666666666667, 'SO_2 <= 11.705\ngini = 0.506\nsamples = 990\nvalue = [
1011, 3, 403, 145]\nclass = a'),
 Text(0.28125, 0.25, 'NO_2 <= 46.32\ngini = 0.615\nsamples = 564\nvalue = [364, 3, 387, 1
26]\nclass = c'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.602\nsamples = 363\nvalue = [308, 3, 132,
123]\nclass = a'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.309\nsamples = 201\nvalue = [56, 0, 255, 3
]\nclass = c'),
 Text(0.34375, 0.25, 'MXY <= 2.045\ngini = 0.099\nsamples = 426\nvalue = [647, 0, 16, 19]
\nclass = a'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.542\nsamples = 13\nvalue = [14, 0, 2, 8]\n
class = a'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.074\nsamples = 413\nvalue = [633, 0, 14, 1
1]\nclass = a'),
 Text(0.4375, 0.4166666666666667, 'O_3 <= 4.265\ngini = 0.676\nsamples = 6264\nvalue = [1
254, 1147, 3168, 4349]\nclass = d'),
 Text(0.40625, 0.25, 'PXY <= 2.395\ngini = 0.215\nsamples = 291\nvalue = [38, 394, 0, 15]
\nclass = b'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.1\nsamples = 192\nvalue = [13, 290, 0, 3]\
nclass = b'),
 Text(0.421875, 0.08333333333333333, 'gini = 0.417\nsamples = 99\nvalue = [25, 104, 0, 12
]\nclass = b'),
```

```
 Text(0.46875, 0.25, 'NOx <= 96.515\ngini = 0.656\nsamples = 5973\nvalue = [1216, 753, 31
68, 4334]\nclass = d'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.578\nsamples = 2231\nvalue = [156, 566, 71
6, 2110]\nclass = d'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.655\nsamples = 3742\nvalue = [1060, 187, 2
452, 2224]\nclass = c'),
 Text(0.75, 0.75, 'O_3 <= 8.805\ngini = 0.585\nsamples = 3941\nvalue = [3633, 175, 1370,
1087]\nclass = a'),
 Text(0.625, 0.5833333333333334, 'TOL <= 37.015\ngini = 0.686\nsamples = 1788\nvalue = [1
164, 153, 830, 723]\nclass = a'),
 Text(0.5625, 0.4166666666666667, 'EBE <= 7.58\ngini = 0.71\nsamples = 863\nvalue = [419,
115, 484, 387]\nclass = c'),
 Text(0.53125, 0.25, 'CO <= 1.165\ngini = 0.709\nsamples = 800\nvalue = [402, 115, 468, 3
21]\nclass = c'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.654\nsamples = 167\nvalue = [130, 13, 58,
68]\nclass = a'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.706\nsamples = 633\nvalue = [272, 102, 410
, 253]\nclass = c'),
 Text(0.59375, 0.25, 'OXY <= 7.27\ngini = 0.5\nsamples = 63\nvalue = [17, 0, 16, 66]\ncla
ss = d'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.132\nsamples = 35\nvalue = [1, 0, 3, 53]\n
class = d'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.663\nsamples = 28\nvalue = [16, 0, 13, 13]
\nclass = a'),
 Text(0.6875, 0.4166666666666667, 'TCH <= 1.785\ngini = 0.632\nsamples = 925\nvalue = [74
5, 38, 346, 336]\nclass = a'),
 Text(0.65625, 0.25, 'SO_2 <= 24.935\ngini = 0.419\nsamples = 271\nvalue = [319, 5, 73, 3
5]\nclass = a'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.63\nsamples = 88\nvalue = [56, 5, 56, 16]\
nclass = a'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.219\nsamples = 183\nvalue = [263, 0, 17, 1
9]\nclass = a'),
 Text(0.71875, 0.25, 'O_3 <= 5.29\ngini = 0.674\nsamples = 654\nvalue = [426, 33, 273, 30
1]\nclass = a'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.553\nsamples = 266\nvalue = [223, 30, 1, 1
52]\nclass = a'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.65\nsamples = 388\nvalue = [203, 3, 272, 1
49]\nclass = c'),
 Text(0.875, 0.5833333333333334, 'TOL <= 32.175\ngini = 0.434\nsamples = 2153\nvalue = [2
469, 22, 540, 364]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'TCH <= 1.375\ngini = 0.5\nsamples = 1484\nvalue = [155
5, 18, 453, 304]\nclass = a'),
 Text(0.78125, 0.25, 'NOx <= 204.45\ngini = 0.25\nsamples = 488\nvalue = [664, 3, 84, 22]
\nclass = a'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.185\nsamples = 454\nvalue = [654, 3, 48, 2
2]\nclass = a'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.34\nsamples = 34\nvalue = [10, 0, 36, 0]\n
class = c'),
 Text(0.84375, 0.25, 'NO_2 <= 89.325\ngini = 0.583\nsamples = 996\nvalue = [891, 15, 369,
282]\nclass = a'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.461\nsamples = 524\nvalue = [558, 4, 72, 1
65]\nclass = a'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.629\nsamples = 472\nvalue = [333, 11, 297,
117]\nclass = a'),
 Text(0.9375, 0.4166666666666667, 'NO_2 <= 109.95\ngini = 0.254\nsamples = 669\nvalue = [
914, 4, 87, 60]\nclass = a'),
 Text(0.90625, 0.25, 'TOL <= 40.495\ngini = 0.166\nsamples = 430\nvalue = [627, 2, 31, 28
]\nclass = a'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.241\nsamples = 250\nvalue = [351, 2, 28, 2
4]\nclass = a'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.049\nsamples = 180\nvalue = [276, 0, 3, 4]
\nclass = a'),
 Text(0.96875, 0.25, 'EBE <= 6.2\ngini = 0.391\nsamples = 239\nvalue = [287, 2, 56, 32]\n
class = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.526\nsamples = 32\nvalue = [23, 2, 30, 0]\
nclass = c'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.311\nsamples = 207\nvalue = [264, 0, 26, 3
2]\nclass = a')]
```





Conclusion

In [72]:

```
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: 0.1560981914762234
Ridge Regression: 0.15714896786525434
Lasso Regression 0.04533993694040095
ElasticNet Regression: 0.10478612906640628
Logistic Regression: 0.8086892042198928
Random Forest: 0.7368547765793528
```

Logistic Regression is suitable for this dataset