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_2007.csv")
df
```

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

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	SO_
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.199997	97.43	NaN	64.5199
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.809998	NaN	NaN	35.4199
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.099998	NaN	NaN	19.0800
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.300003	NaN	NaN	17.6700
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.500000	15.90	3.56	40.2300
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.760000	5.14	1.00	7.4200
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	5.700000	NaN	NaN	7.1300
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	13.010000	6.95	NaN	8.7400
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	6.610000	NaN	NaN	9.8900
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.260000	7.08	1.23	9.8900

225120 rows × 17 columns

Data Cleaning and Data Preprocessing

In [3]:

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

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [6]:

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

Out[6]:

	CO	station
4	1.86	28079006
21	0.31	28079024
25	1.42	28079099
30	1.89	28079006
47	0.30	28079024
...	...	...
225073	0.47	28079006
225094	0.45	28079099
225098	0.41	28079006
225115	0.45	28079024
225119	0.40	28079099

25443 rows × 2 columns

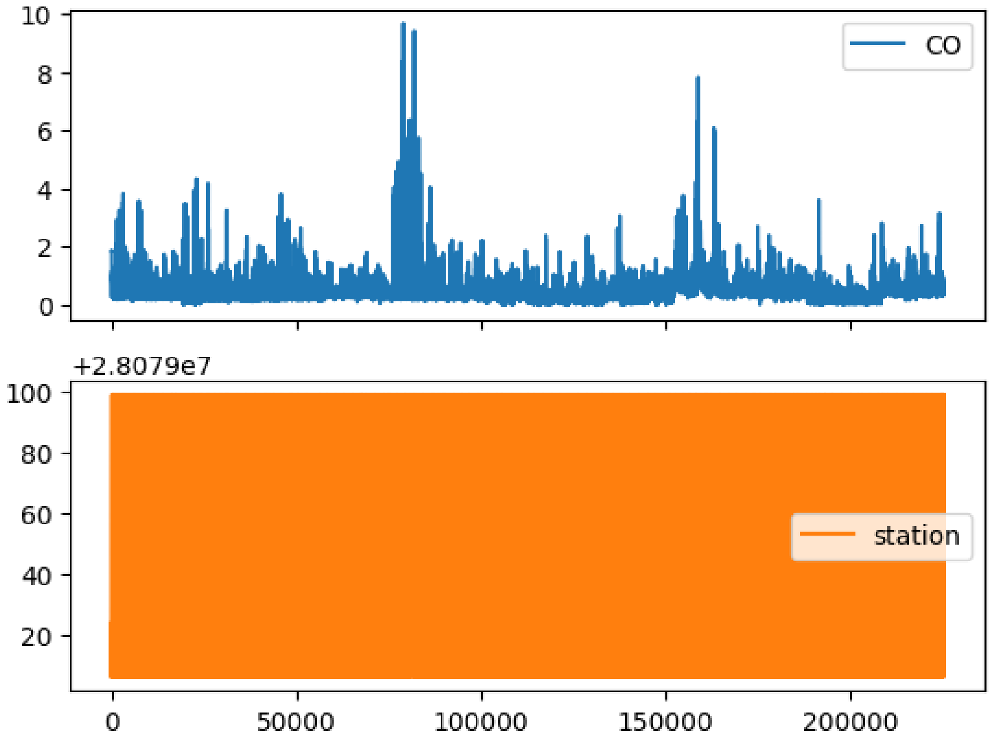
Line chart

In [7]:

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

Out[7]:

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



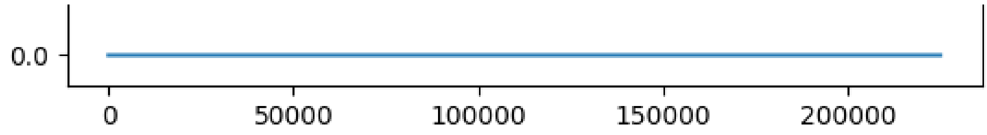
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

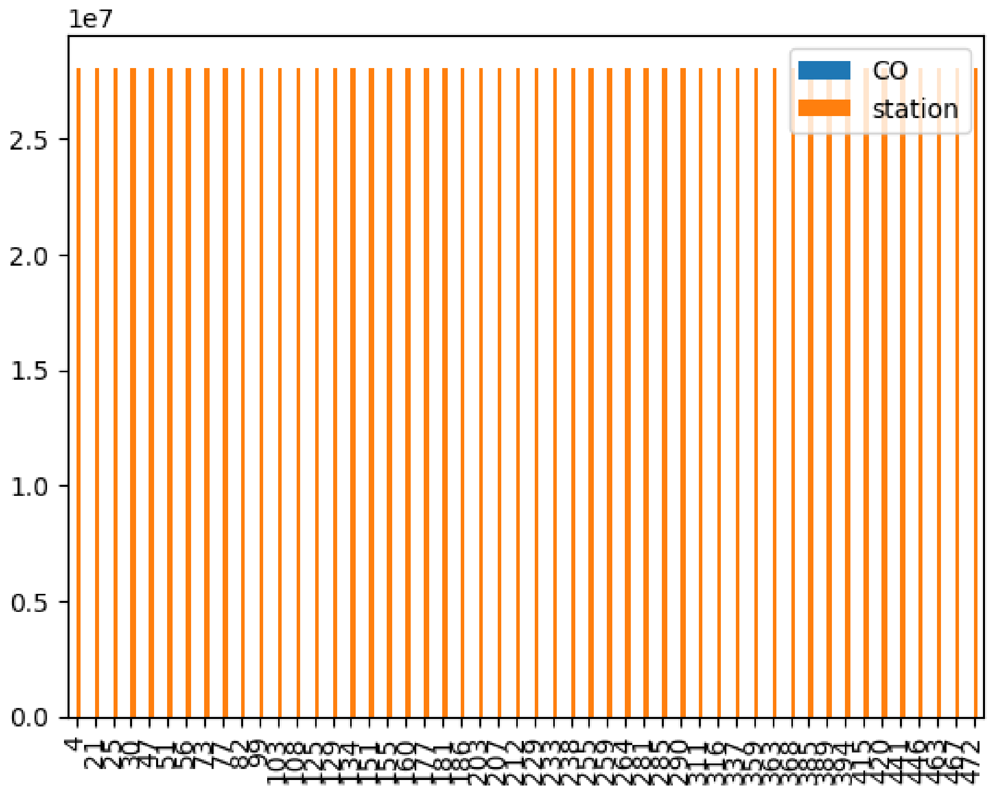
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



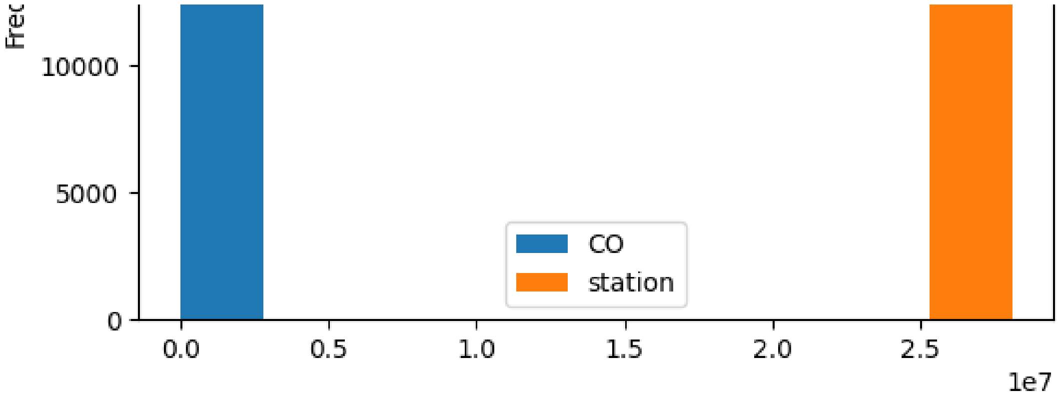
Histogram

In [11]:

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

Out[11]:

<Axes: ylabel='Frequency'>



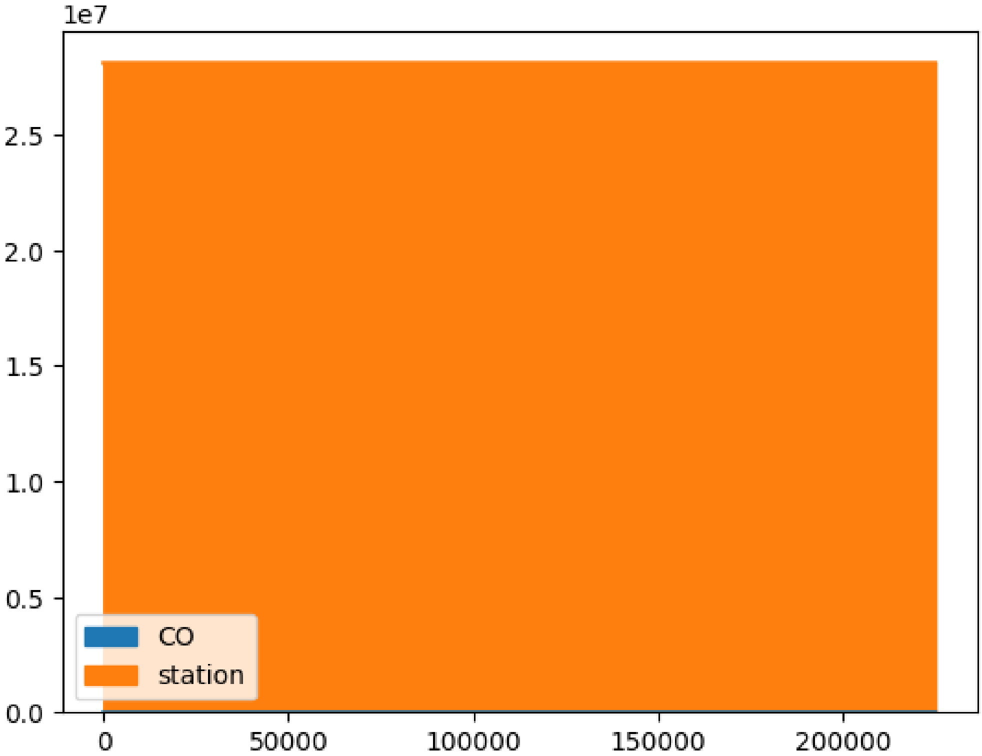
Area chart

In [12]:

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

Out[12]:

<Axes: >



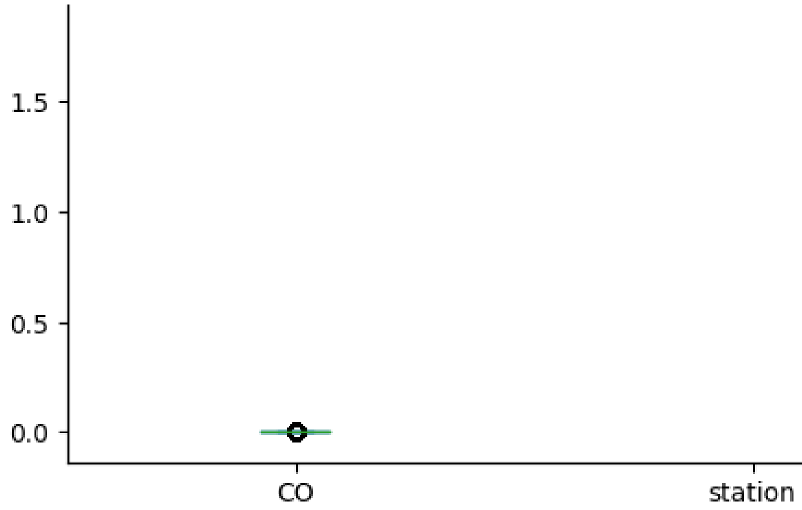
Box chart

In [13]:

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

Out[13]:

<Axes: >



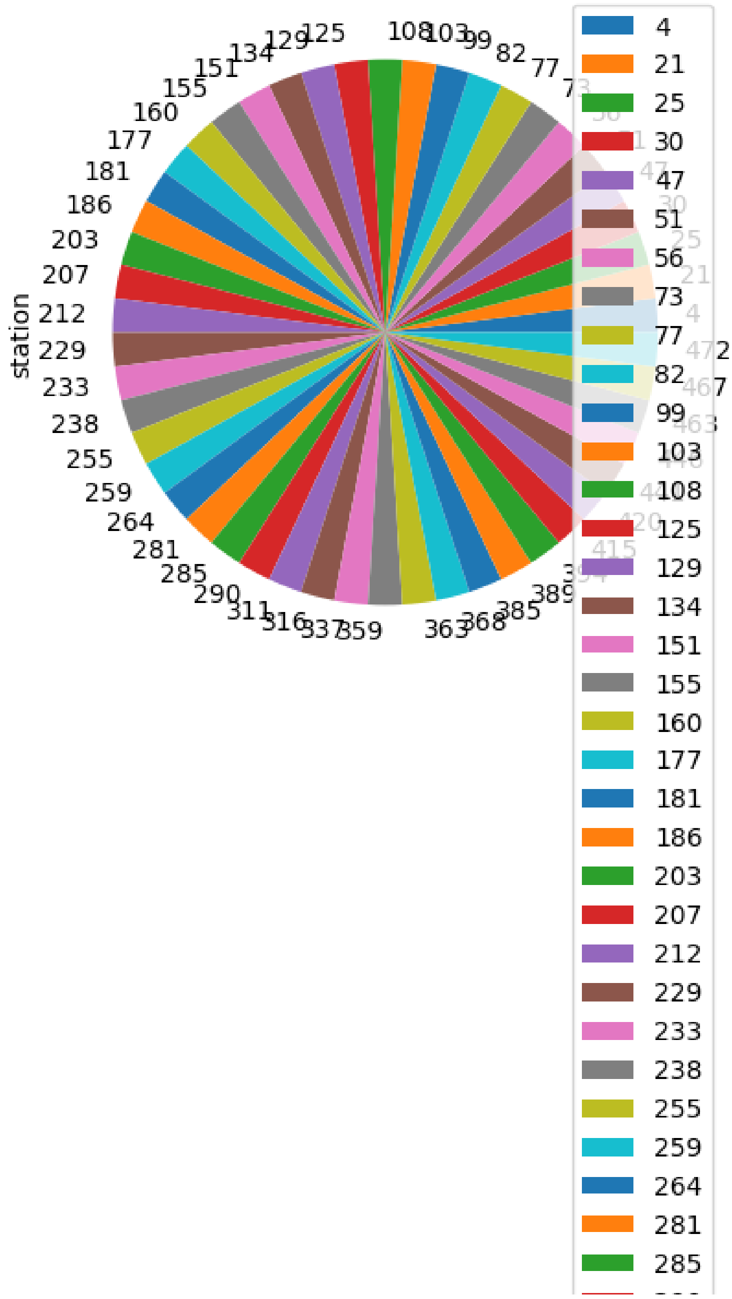
Pie chart

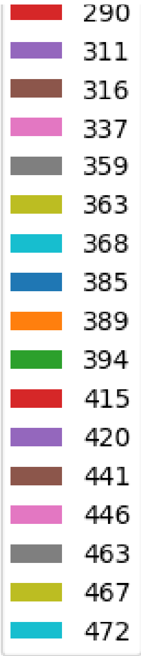
In [14]:

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

Out[14]:

<Axes: ylabel='station'>





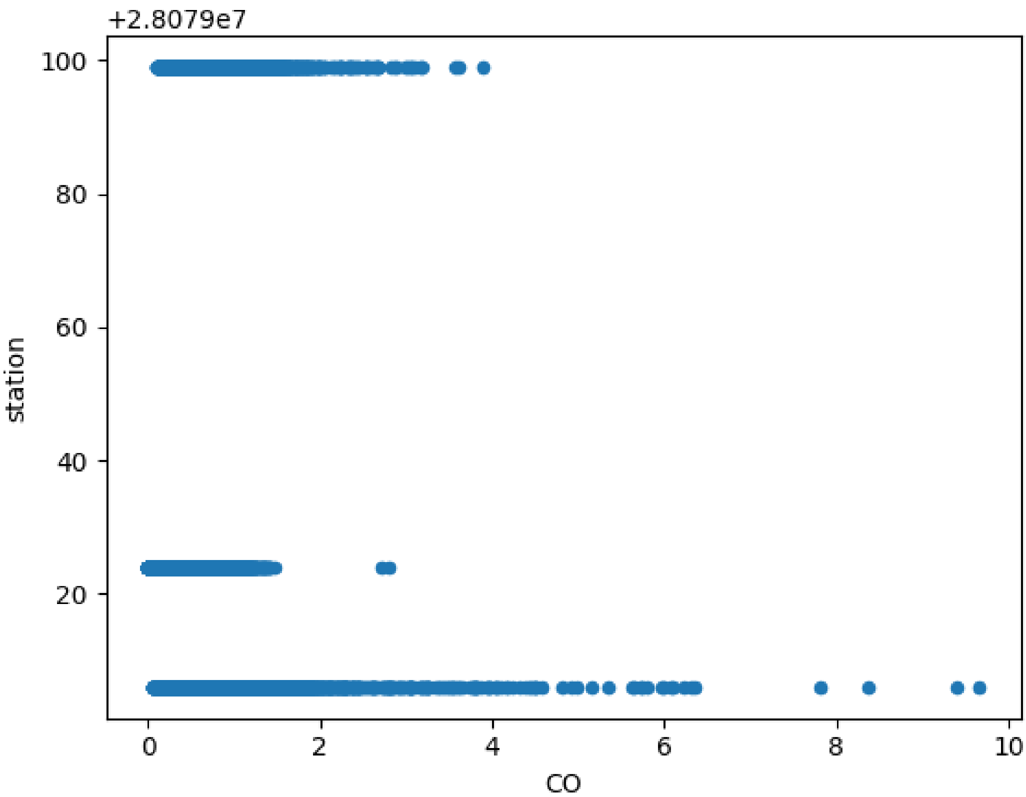
Scatter chart

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: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     25443 non-null  object
```

```
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	254
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683	112.741861	1.270278	
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029	115.527006	1.143188	
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000	1.780000	0.110000	
25%	0.450000	0.260000	0.780000	0.960000	0.160000	31.285001	39.910000	0.740000	
50%	0.770000	0.400000	1.000000	1.500000	0.220000	54.080002	82.809998	1.000000	
75%	1.390000	0.640000	1.580000	2.855000	0.300000	79.230003	149.300003	1.360000	
max	30.139999	9.660000	31.680000	65.480003	2.570000	430.299988	1893.000000	32.540001	1

In [18]:

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

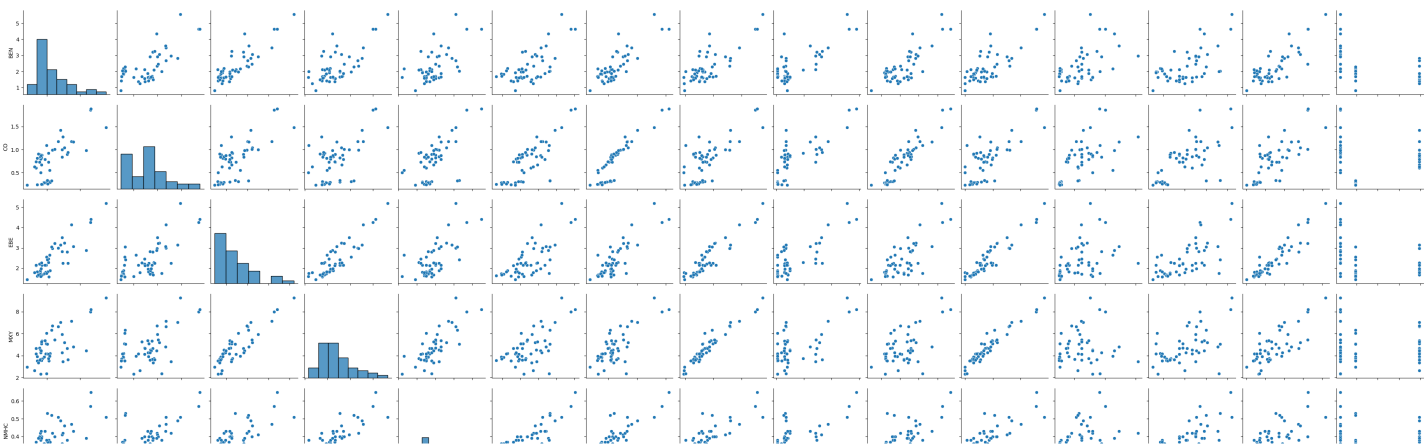
EDA AND VISUALIZATION

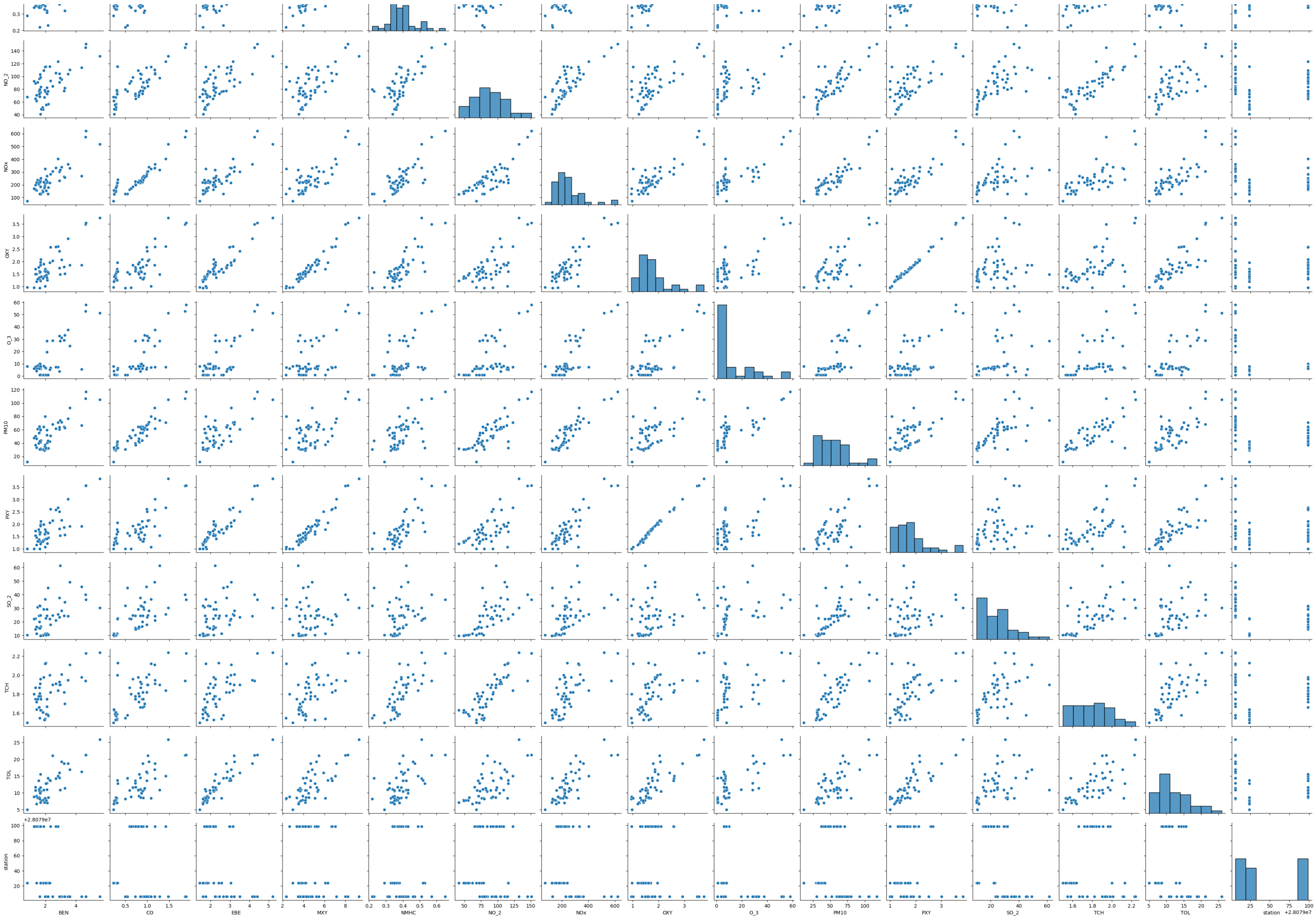
In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x7c28d80a6080>





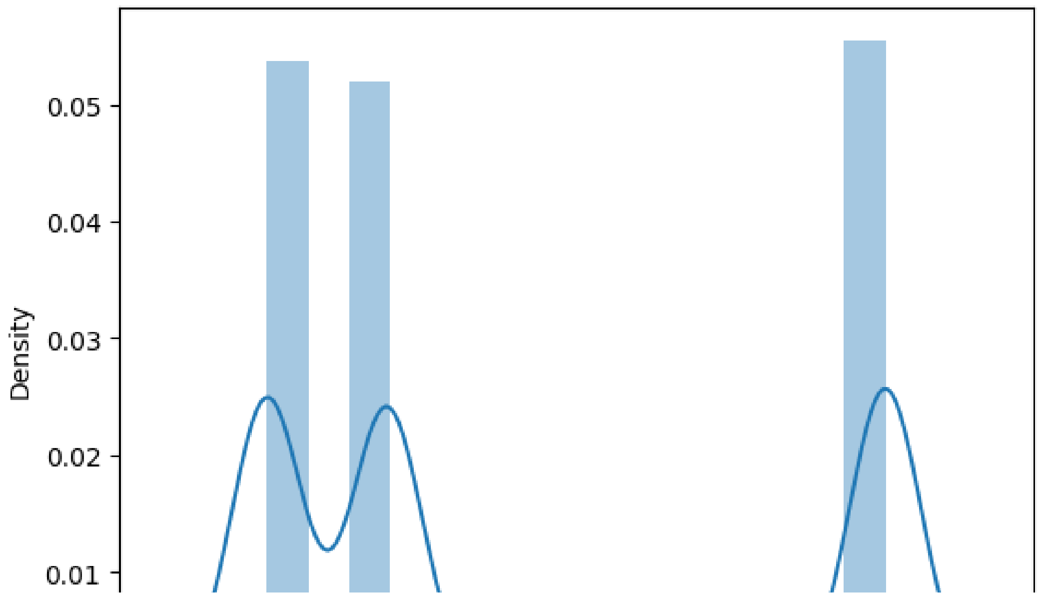
In [20]:

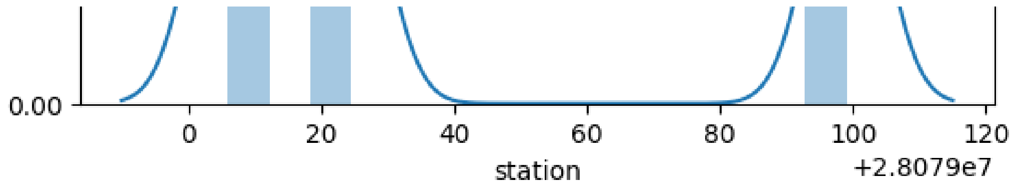
```
sns.distplot(df1['station'])
```

```
<ipython-input-20-4bc330f7257f>: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[20]:

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



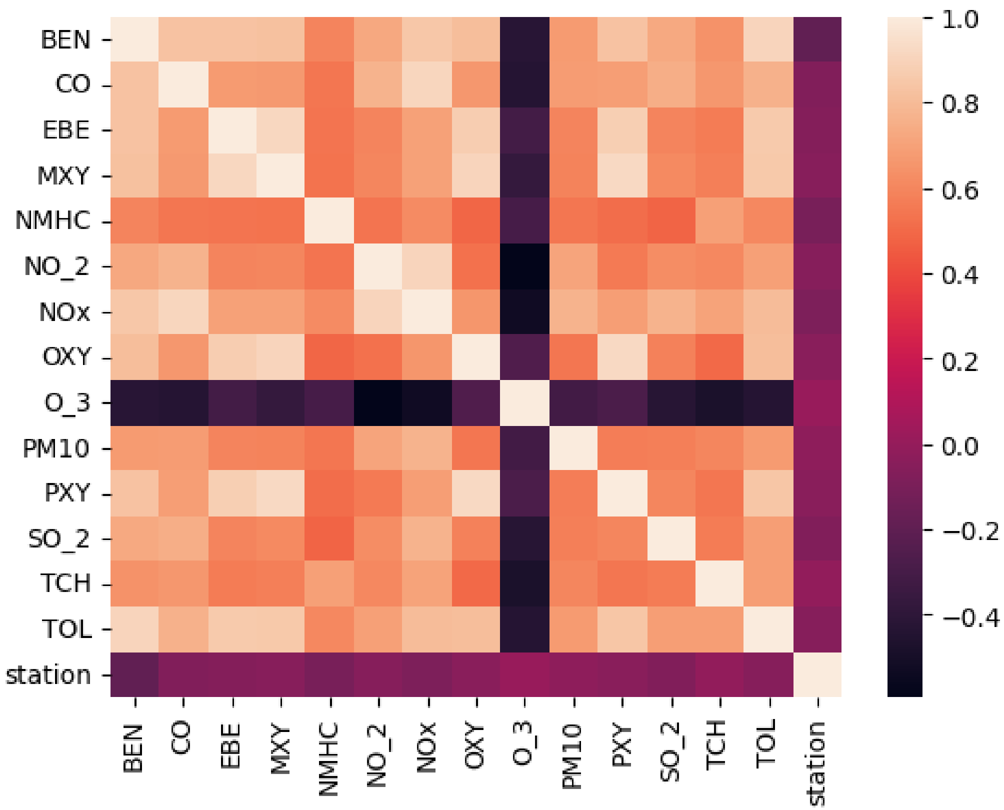


In [21]:

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

Out[21]:

```
<Axes: >
```



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:

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

In [23]:

```
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 [24]:

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

Out[24]:

```
▾ LinearRegression
LinearRegression()
```

In [25]:

```
lr.intercept_
```

Out[25]:

```
28079009.537545096
```

In [26]:

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

Out[26]:

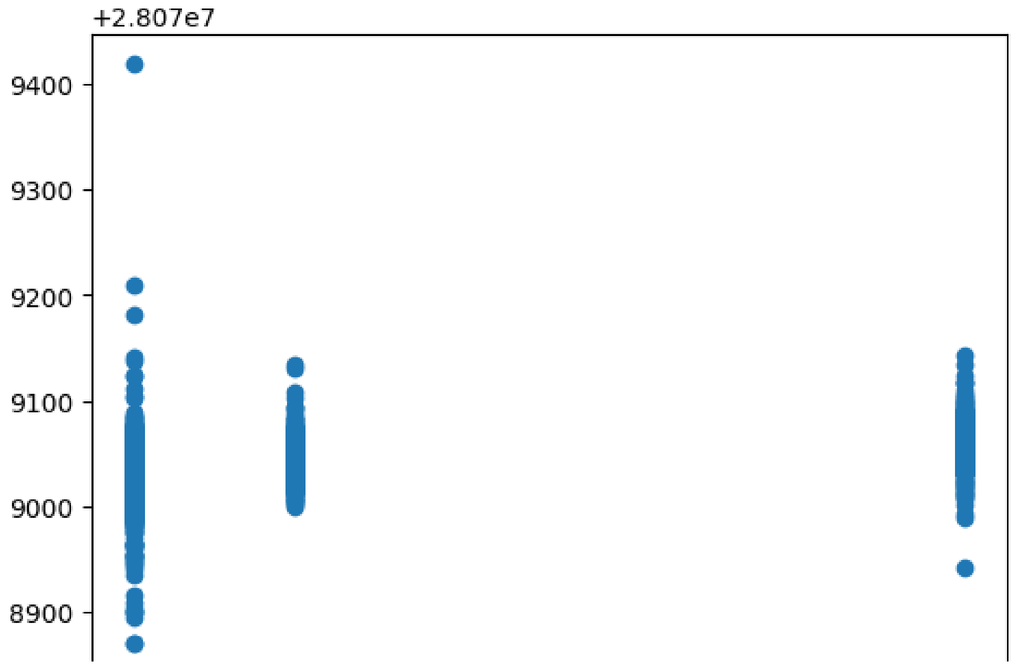
	Co-efficient
BEN	-33.878090
CO	16.498366
EBE	1.482030
MXY	-1.353057
NMHC	-40.412680
NO_2	0.121295
NOx	-0.037018
OXY	6.095063
O_3	-0.036906
PM10	0.143900
PXY	7.256940
SO_2	0.246843
TCH	25.289281
TOL	3.010203

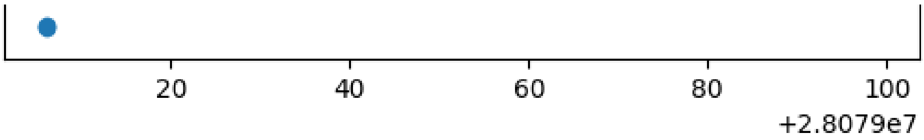
In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x7c28c503e3e0>
```





ACCURACY

In [28]:

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

Out[28]:

```
0.15525827694238103
```

In [29]:

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

Out[29]:

```
0.16056031664547565
```

Ridge and Lasso

In [30]:

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

In [31]:

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

Out[31]:

```
▾ Ridge
Ridge(alpha=10)
```

Accuracy(Ridge)

In [32]:

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

Out[32]:

```
0.1552057727588495
```

In [33]:

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

Out[33]:

```
0.16050892632951208
```

In [34]:

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

Out[34]:

```
▾ Lasso
```

```
Lasso(alpha=10)
```

In [35]:

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

Out[35]:

```
0.01325224953187909
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.014198799959927011
```

In [37]:

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

Out[37]:

```
▾ ElasticNet
ElasticNet()
```

In [38]:

```
en.coef_
```

Out[38]:

```
array([-7.99626026,  0.        ,  0.        ,  0.21923228, -0.        ,
        0.06336065, -0.0554393 ,  0.8249646 , -0.05906873,  0.16760411,
        0.72498793,  0.01837265,  0.        ,  0.84622624])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079045.564106826
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.06927782673837335
```

Evaluation Metrics

In [42]:

```
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)))
```

```
36.70207687149274
1535.6845903561043
39.18781175768946
```

Logistic Regression

In [43]:

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

In [44]:

```
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 [45]:

```
feature_matrix.shape
```

Out[45]:

```
(25443, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(25443,)
```

In [47]:

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

In [48]:

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

In [49]:

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

Out[49]:

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

In [50]:

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

In [51]:

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

```
[28079099]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

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

In [53]:

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

Out[53]:

```
0.8146838030106512
```

In [54]:

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

Out[54]:

```
1.0827539764163807e-19
```

In [55]:

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

Out[55]:

```
array([[1.08275398e-19, 1.80383815e-19, 1.00000000e+00]])
```

Random Forest

In [56]:

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

In [57]:

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

Out[57]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [58]:

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

In [59]:

```
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[59]:

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

In [60]:

```
grid_search.best_score_
```

Out[60]:

```
0.8214486243683324
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

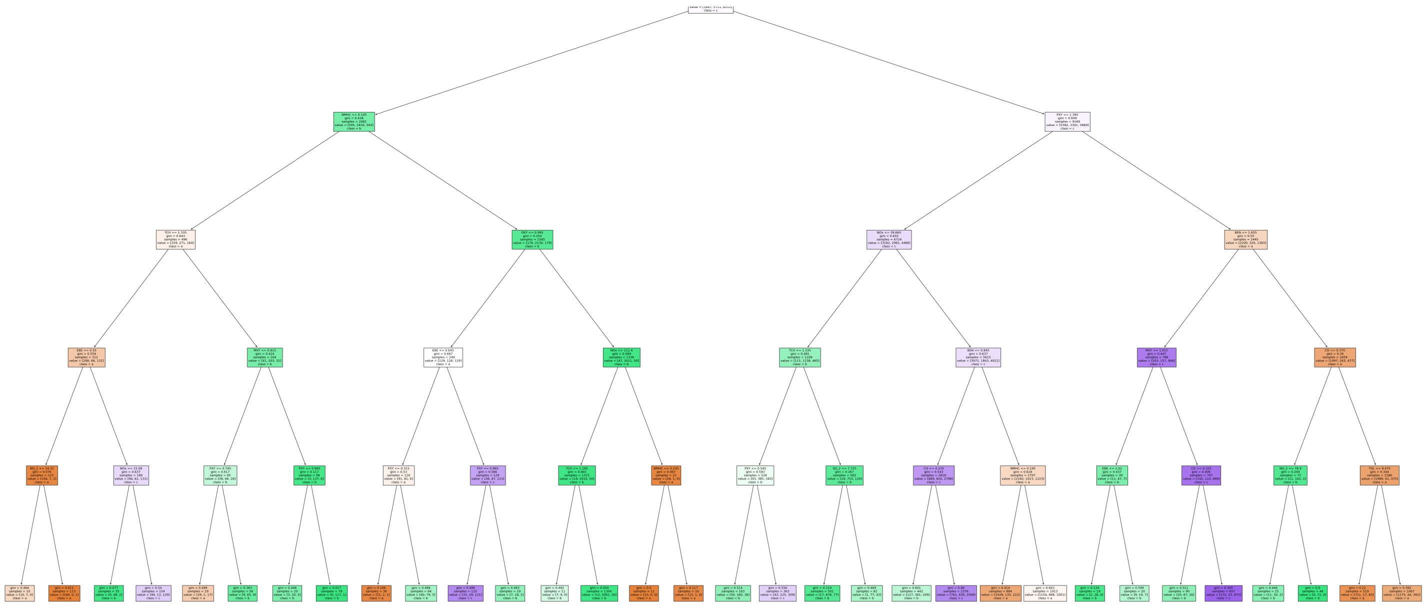
In [62]:

```
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[62]:

```
[Text(0.5, 0.9166666666666666, 'TOL <= 1.325\ngini = 0.666\nsamples = 11250\nvalue = [588
7, 5711, 6212]\nclass = c'),
 Text(0.25, 0.75, 'NMHC <= 0.145\ngini = 0.418\nsamples = 2081\nvalue = [505, 2410, 343]\
nclass = b'),
 Text(0.125, 0.5833333333333334, 'TCH <= 1.335\ngini = 0.643\nsamples = 496\nvalue = [329
, 271, 164]\nclass = a'),
 Text(0.0625, 0.4166666666666667, 'EBE <= 0.55\ngini = 0.559\nsamples = 312\nvalue = [288
, 68, 132]\nclass = a'),
 Text(0.03125, 0.25, 'NO_2 <= 14.31\ngini = 0.076\nsamples = 123\nvalue = [194, 7, 1]\ncl
ass = a'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.484\nsamples = 10\nvalue = [10, 7, 0]\ncla
ss = a'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.011\nsamples = 113\nvalue = [184, 0, 1]\nc
lass = a'),
 Text(0.09375, 0.25, 'NOx <= 15.08\ngini = 0.637\nsamples = 189\nvalue = [94, 61, 131]\nc
lass = c'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.077\nsamples = 35\nvalue = [0, 48, 2]\ncla
ss = b'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.54\nsamples = 154\nvalue = [94, 13, 129]\n
class = c'),
 Text(0.1875, 0.4166666666666667, 'MXY <= 0.815\ngini = 0.424\nsamples = 184\nvalue = [41
, 203, 32]\nclass = b'),
 Text(0.15625, 0.25, 'PXY <= 0.745\ngini = 0.617\nsamples = 85\nvalue = [38, 66, 26]\ncla
ss = b'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.488\nsamples = 29\nvalue = [29, 1, 17]\ncl
ass = a'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.363\nsamples = 56\nvalue = [9, 65, 9]\ncla
ss = b'),
 Text(0.21875, 0.25, 'PXY <= 0.965\ngini = 0.117\nsamples = 99\nvalue = [3, 137, 6]\nclas
s = b'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.446\nsamples = 20\nvalue = [3, 20, 5]\ncla
ss = b'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.017\nsamples = 79\nvalue = [0, 117, 1]\ncl
ass = b'),
 Text(0.375, 0.5833333333333334, 'OXY <= 0.995\ngini = 0.254\nsamples = 1585\nvalue = [17
6, 2139, 179]\nclass = b'),
 Text(0.3125, 0.4166666666666667, 'EBE <= 0.645\ngini = 0.667\nsamples = 249\nvalue = [12
9, 128, 129]\nclass = a'),
 Text(0.28125, 0.25, 'PXY <= 0.315\ngini = 0.53\nsamples = 120\nvalue = [91, 81, 6]\nclas
s = a'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.106\nsamples = 36\nvalue = [51, 2, 1]\ncla
ss = a'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.488\nsamples = 84\nvalue = [40, 79, 5]\ncl
ass = b'),
 Text(0.34375, 0.25, 'PXY <= 0.985\ngini = 0.566\nsamples = 129\nvalue = [38, 47, 123]\nc
lass = c'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.498\nsamples = 110\nvalue = [31, 29, 121]\
nclass = c'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.483\nsamples = 19\nvalue = [7, 18, 2]\ncla
ss = b'),
 Text(0.4375, 0.4166666666666667, 'NOx <= 111.4\ngini = 0.089\nsamples = 1336\nvalue = [4
7, 2011, 50]\nclass = b'),
 Text(0.40625, 0.25, 'TCH <= 1.185\ngini = 0.065\nsamples = 1315\nvalue = [19, 2010, 50]\
nclass = b'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.492\nsamples = 11\nvalue = [7, 9, 0]\nclas
s = b'),
 Text(0.421875, 0.08333333333333333, 'gini = 0.059\nsamples = 1304\nvalue = [12, 2001, 50
```

```
]\nclass = b'),
 Text(0.46875, 0.25, 'NMHC <= 0.235\ngini = 0.067\nsamples = 21\nvalue = [28, 1, 0]\nclas
s = a'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.0\nsamples = 11\nvalue = [13, 0, 0]\nclass
= a'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.117\nsamples = 10\nvalue = [15, 1, 0]\ncla
ss = a'),
 Text(0.75, 0.75, 'PXY <= 1.385\ngini = 0.649\nsamples = 9169\nvalue = [5382, 3301, 5869]
\nclass = c'),
 Text(0.625, 0.5833333333333334, 'NOx <= 39.665\ngini = 0.655\nsamples = 6724\nvalue = [3
182, 2981, 4486]\nclass = c'),
 Text(0.5625, 0.4166666666666667, 'TCH <= 1.335\ngini = 0.481\nsamples = 1109\nvalue = [1
11, 1138, 465]\nclass = b'),
 Text(0.53125, 0.25, 'PXY <= 0.545\ngini = 0.593\nsamples = 526\nvalue = [93, 385, 345]\n
class = b'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.514\nsamples = 163\nvalue = [50, 160, 36]\
nclass = b'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.556\nsamples = 363\nvalue = [43, 225, 309]
\nclass = c'),
 Text(0.59375, 0.25, 'SO_2 <= 7.725\ngini = 0.267\nsamples = 583\nvalue = [18, 753, 120]\
nclass = b'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.219\nsamples = 501\nvalue = [17, 676, 77]\
nclass = b'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.469\nsamples = 82\nvalue = [1, 77, 43]\ncl
ass = b'),
 Text(0.6875, 0.4166666666666667, 'BEN <= 0.845\ngini = 0.637\nsamples = 5615\nvalue = [3
071, 1843, 4021]\nclass = c'),
 Text(0.65625, 0.25, 'CO <= 0.235\ngini = 0.543\nsamples = 2818\nvalue = [889, 820, 2798]
\nclass = c'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.601\nsamples = 442\nvalue = [127, 381, 209
]\nclass = b'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.48\nsamples = 2376\nvalue = [762, 439, 258
9]\nclass = c'),
 Text(0.71875, 0.25, 'NMHC <= 0.195\ngini = 0.628\nsamples = 2797\nvalue = [2182, 1023, 1
223]\nclass = a'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.414\nsamples = 884\nvalue = [1028, 135, 22
2]\nclass = a'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.663\nsamples = 1913\nvalue = [1154, 888, 1
001]\nclass = a'),
 Text(0.875, 0.5833333333333334, 'BEN <= 1.655\ngini = 0.55\nsamples = 2445\nvalue = [220
0, 320, 1383]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'MXY <= 1.015\ngini = 0.447\nsamples = 786\nvalue = [20
3, 157, 906]\nclass = c'),
 Text(0.78125, 0.25, 'EBE <= 1.01\ngini = 0.437\nsamples = 39\nvalue = [11, 47, 7]\nclass
= b'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.124\nsamples = 19\nvalue = [2, 28, 0]\ncla
ss = b'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.599\nsamples = 20\nvalue = [9, 19, 7]\ncla
ss = b'),
 Text(0.84375, 0.25, 'CO <= 0.325\ngini = 0.406\nsamples = 747\nvalue = [192, 110, 899]\n
class = c'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.511\nsamples = 90\nvalue = [20, 87, 26]\nc
lass = b'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.305\nsamples = 657\nvalue = [172, 23, 873]
\nclass = c'),
 Text(0.9375, 0.4166666666666667, 'CO <= 0.375\ngini = 0.39\nsamples = 1659\nvalue = [199
7, 163, 477]\nclass = a'),
 Text(0.90625, 0.25, 'NO_2 <= 78.9\ngini = 0.204\nsamples = 73\nvalue = [11, 102, 2]\ncla
ss = b'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.446\nsamples = 25\nvalue = [11, 30, 2]\ncl
ass = b'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.0\nsamples = 48\nvalue = [0, 72, 0]\nclass
= b'),
 Text(0.96875, 0.25, 'TOL <= 9.475\ngini = 0.344\nsamples = 1586\nvalue = [1986, 61, 475]
\nclass = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.22\nsamples = 519\nvalue = [711, 17, 83]\n
class = a'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.392\nsamples = 1067\nvalue = [1275, 44, 39
2]\nclass = a')]
```



Conclusion

Accuracy

In [63]:

```
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: 0.15525827694238103
Ridge Regression: 0.1552057727588495
Lasso Regression 0.014198799959927011
ElasticNet Regression: 0.06927782673837335
Logistic Regression: 0.8146838030106512
Random Forest: 0.8214486243683324
```

Random Forest is suitable for this dataset