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_2003.csv")
df
```

Mounted at /content/drive

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY	SO_2	TCH	T
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209999	NaN	24.299999	NaN	N
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389999	NaN	14.230000	1.55	N
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240002	NaN	17.879999	NaN	N
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839996	NaN	24.900000	NaN	N
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779999	NaN	18.750000	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380000	1.20	4.870000	1.27	1
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400000	0.50	8.360000	NaN	(
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830000	NaN	5.330000	1.55	N
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570000	NaN	6.830000	1.27	N
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350000	2.43	6.060000	1.32	5

243984 rows × 16 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', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

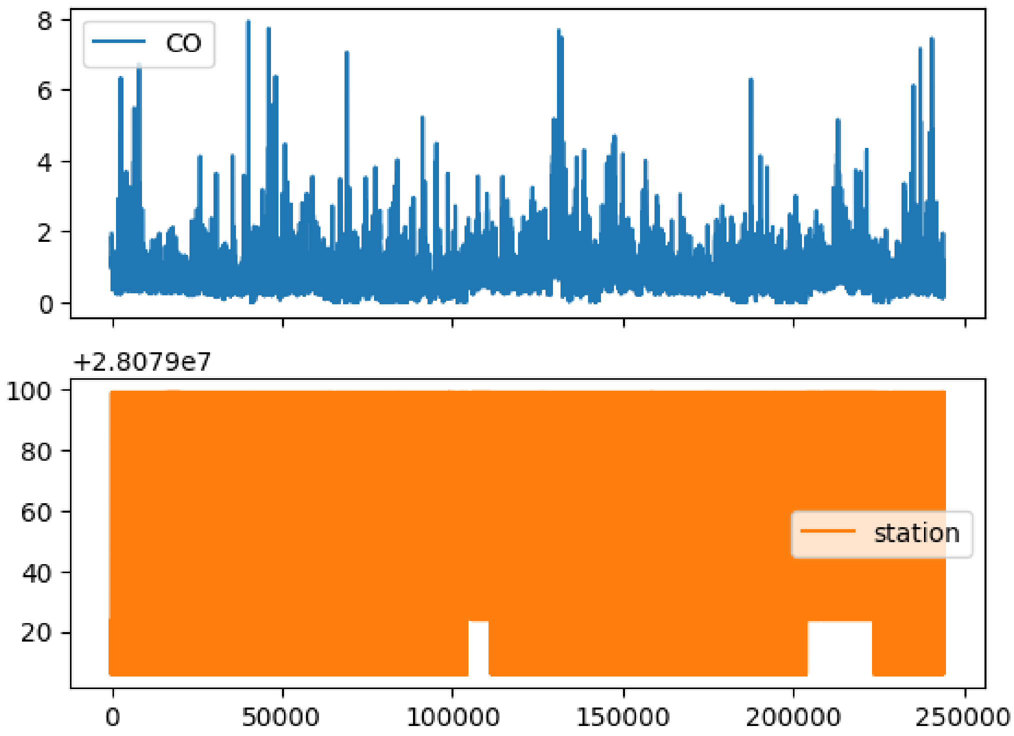
Line chart

In [7]:

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

Out[7]:

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



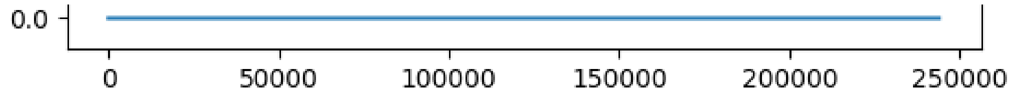
Line chart

In [8]:

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

Out[8]:

<Axes: >



Bar chart

In [9]:

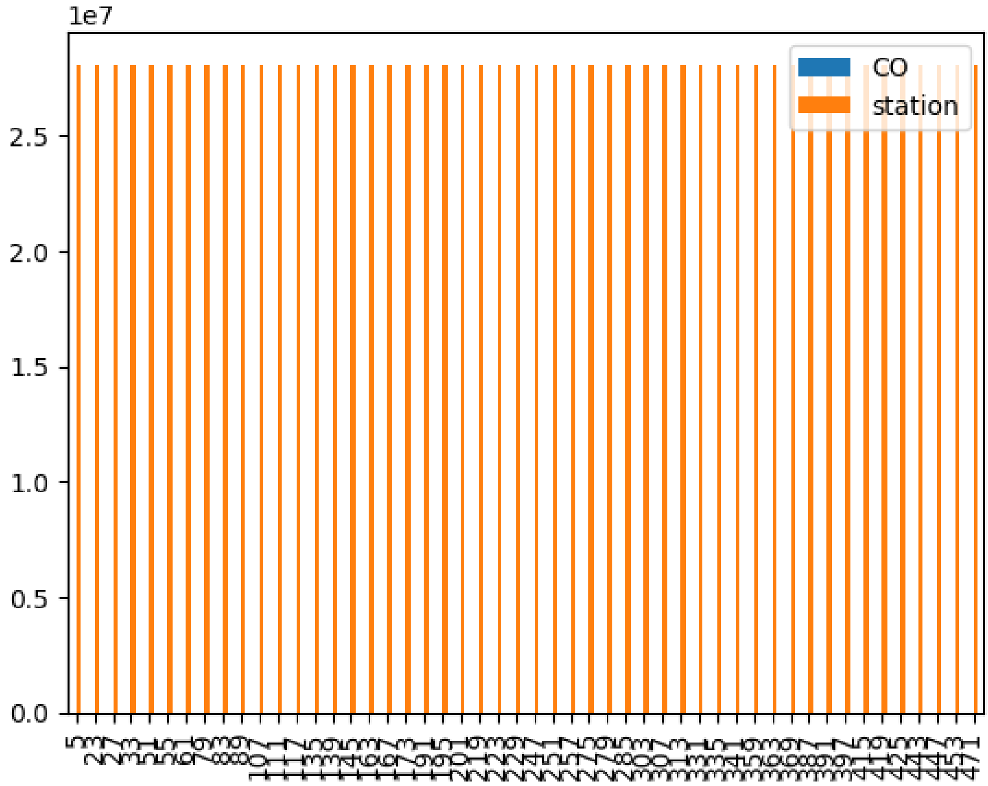
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



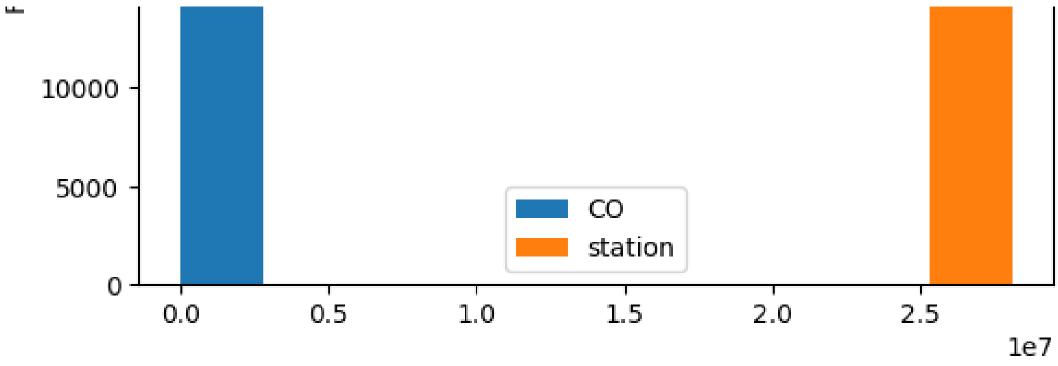
Histogram

In [11]:

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

Out[11]:

<Axes: ylabel='Frequency'>



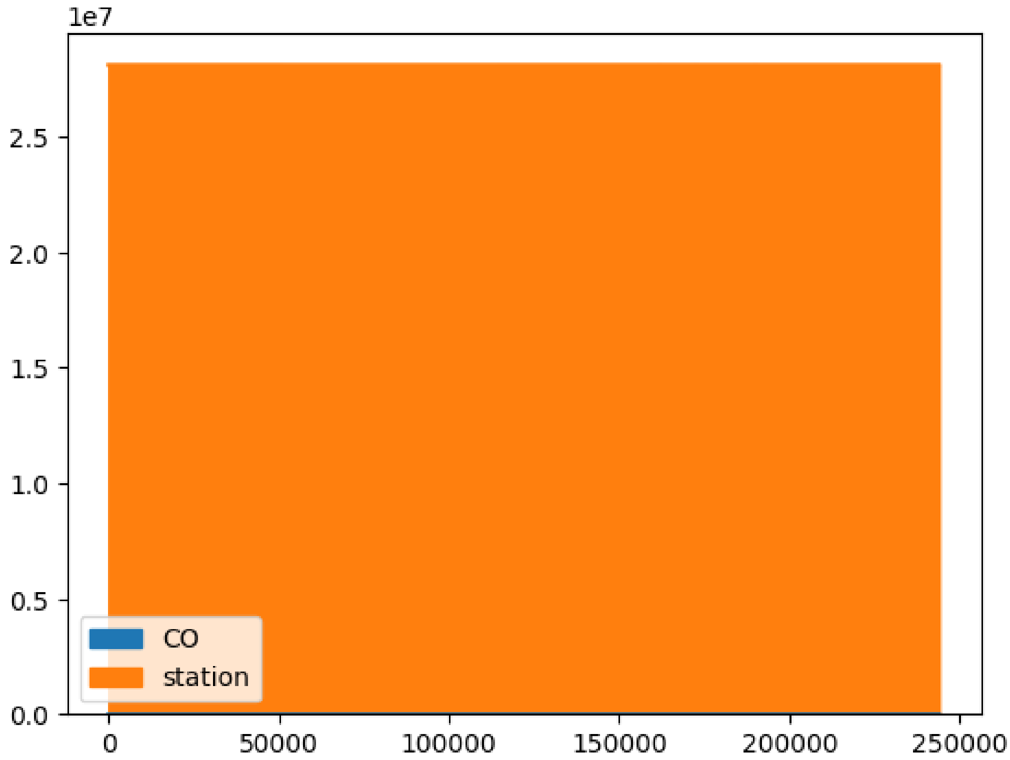
Area chart

In [12]:

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

Out[12]:

<Axes: >



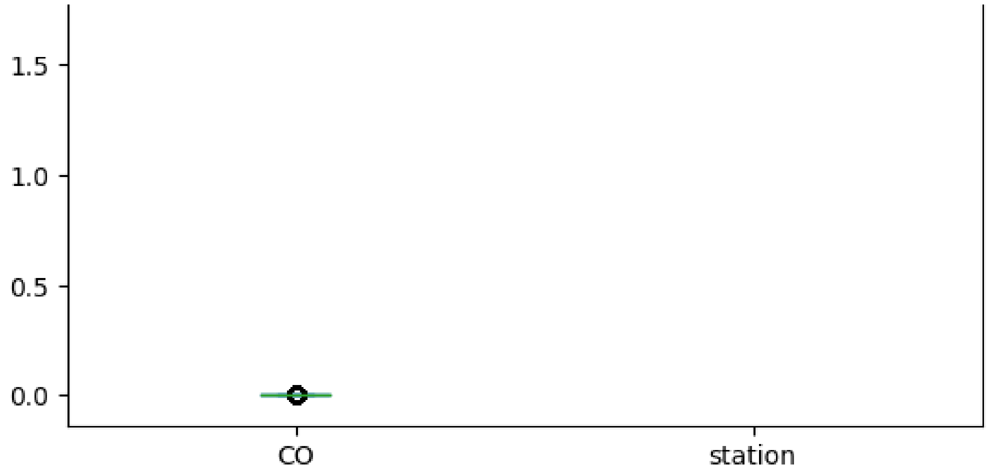
Box chart

In [13]:

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

Out[13]:

<Axes: >



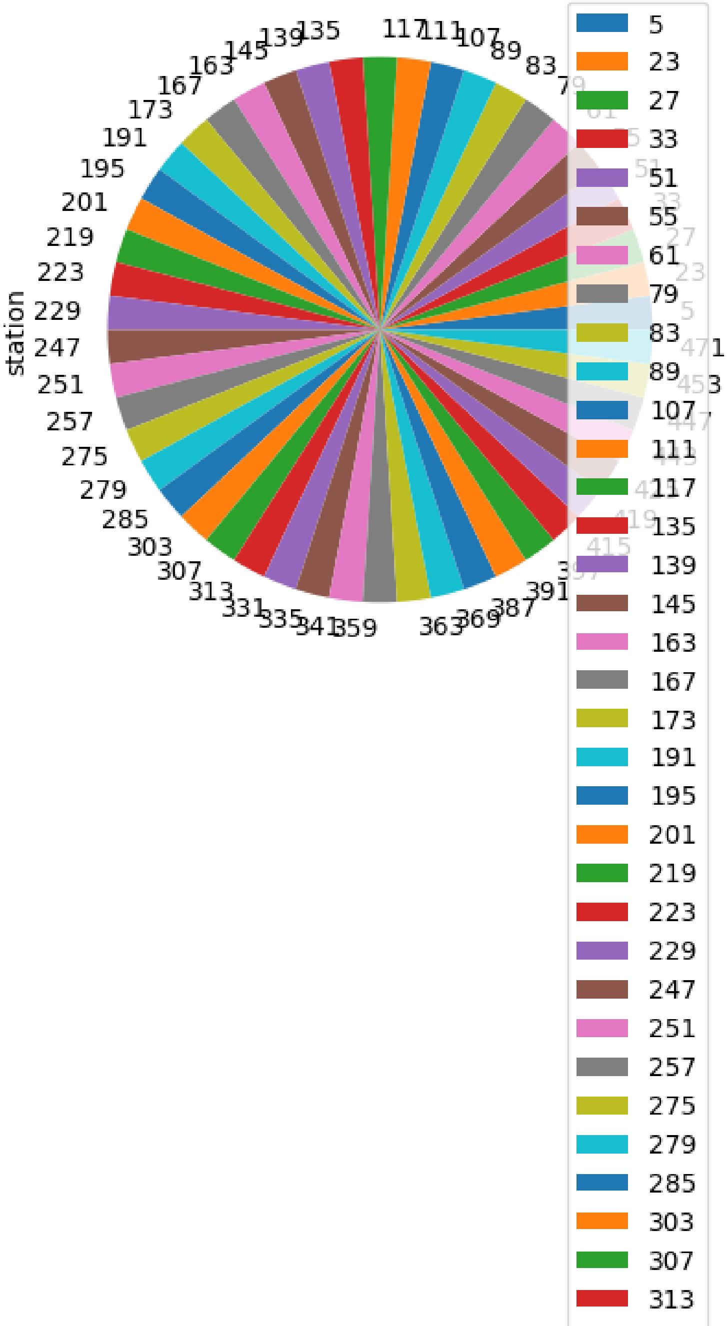
Pie chart

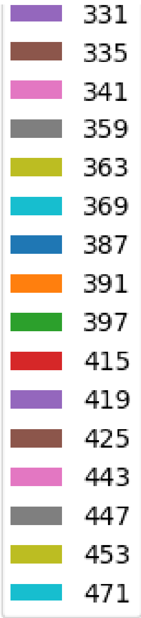
In [14]:

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

Out[14]:

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





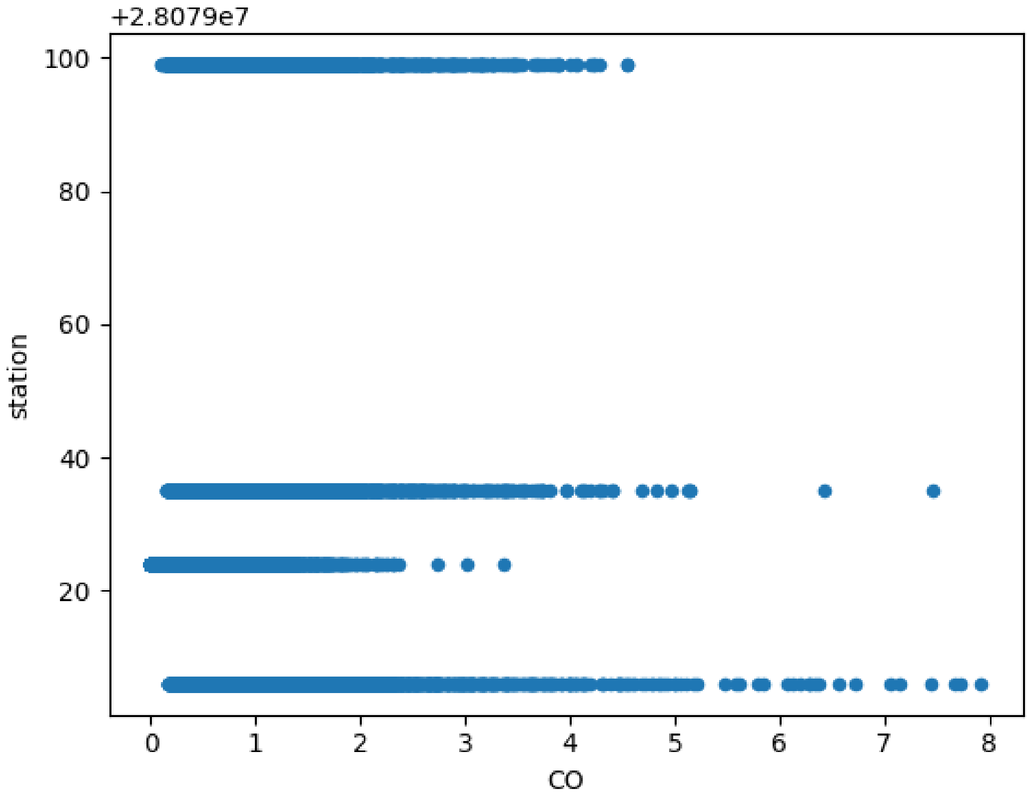
Scatter chart

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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     33010 non-null  object
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
```

```
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	330
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049	120.153676	2.684084	
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082	104.521700	2.717832	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999	49.070000	1.000000	
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000	92.779999	1.790000	
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004	160.100006	3.340000	
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012	1246.000000	88.180000	1

In [18]:

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

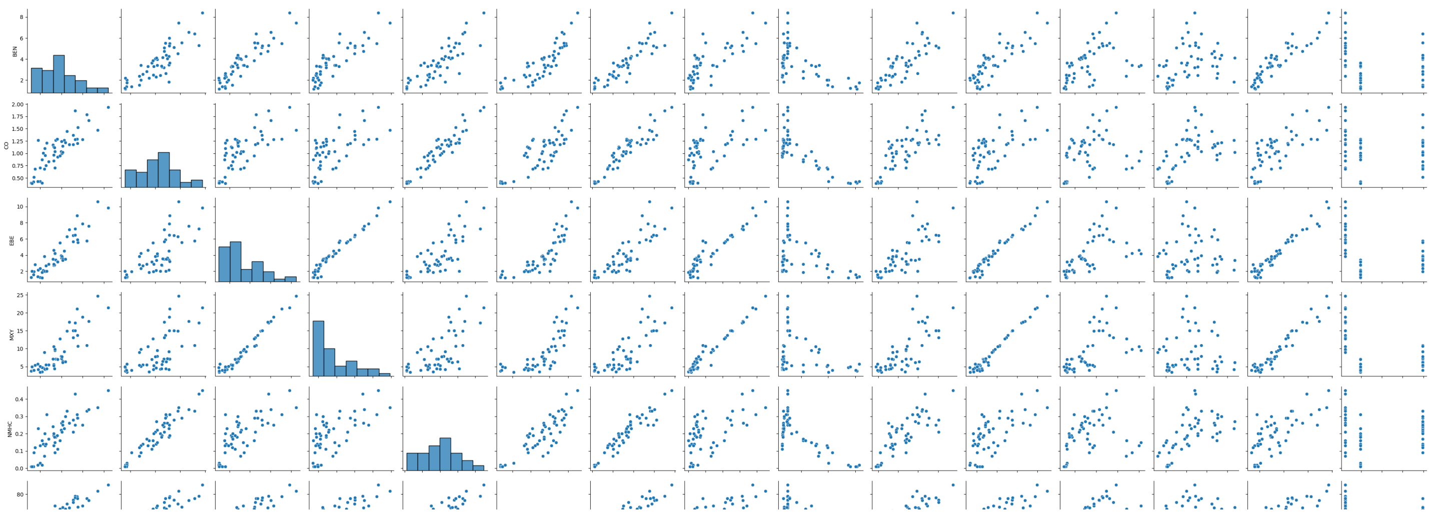
EDA AND VISUALIZATION

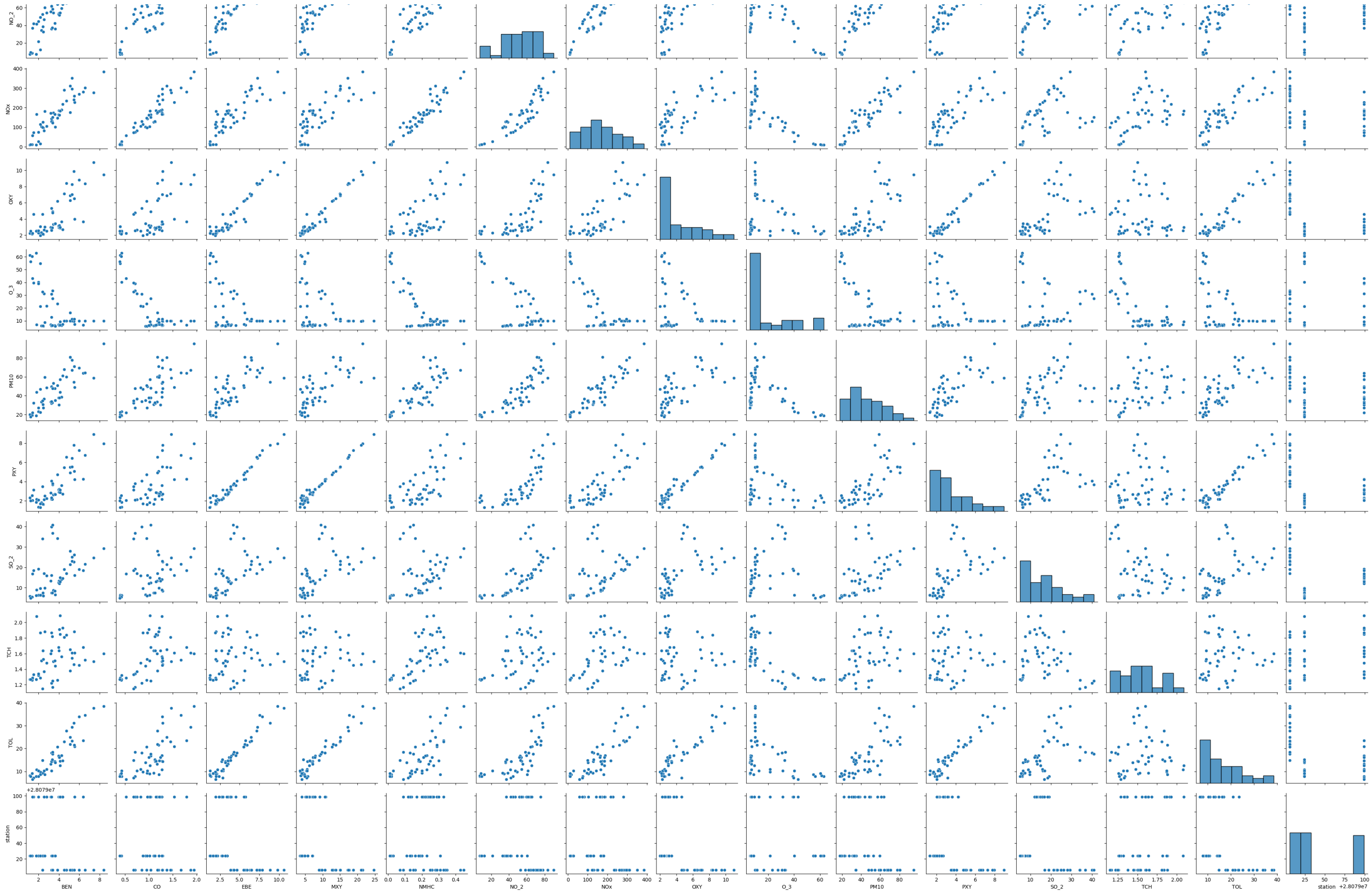
In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x789e946231f0>
```





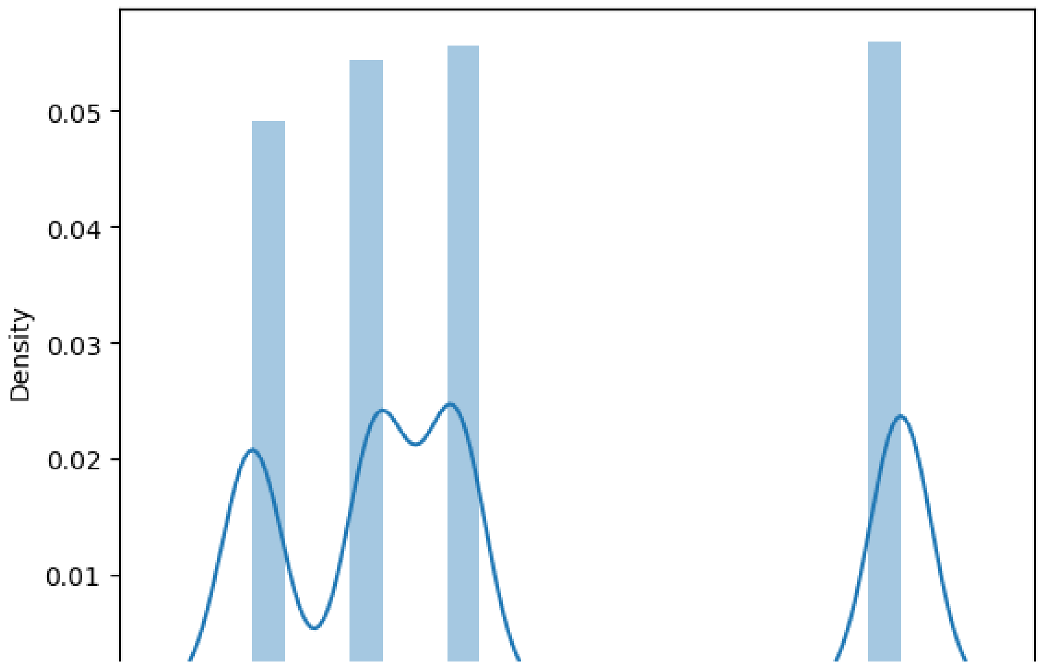
In [20]:

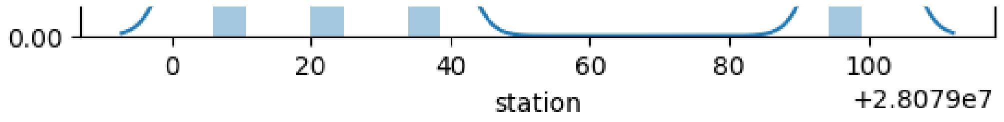
```
sns.distplot(df1['station'])
```

```
<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'>
```



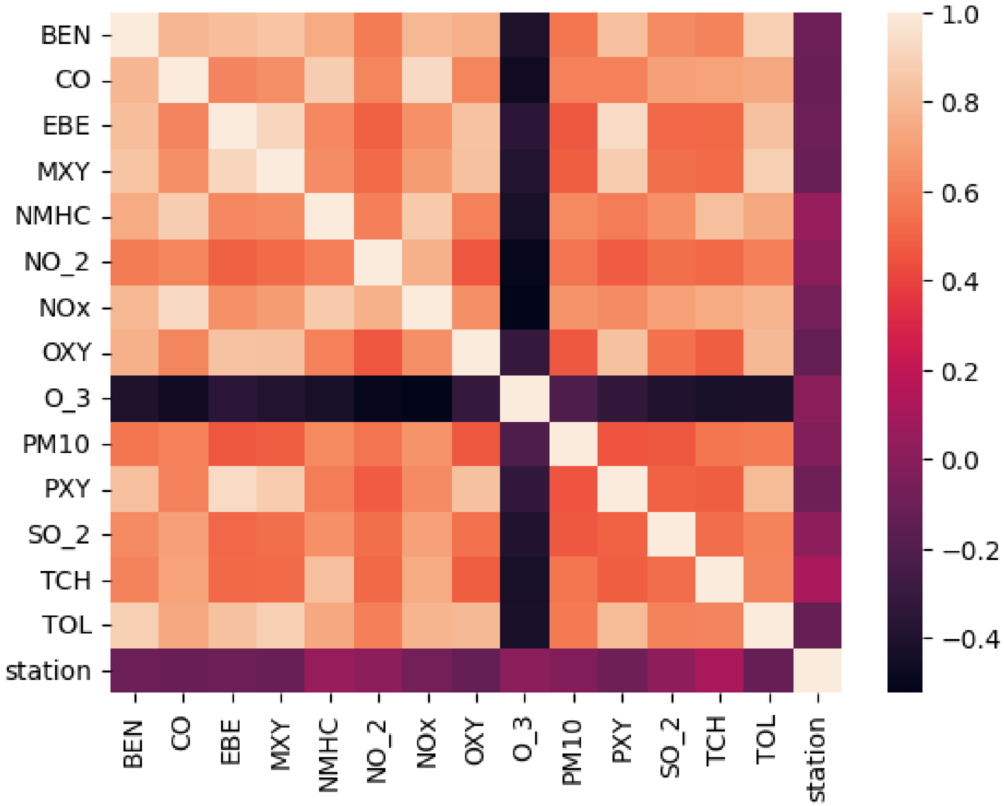


In [21]:

```
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]:

```
28079000.00250134
```

In [26]:

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

Out[26]:

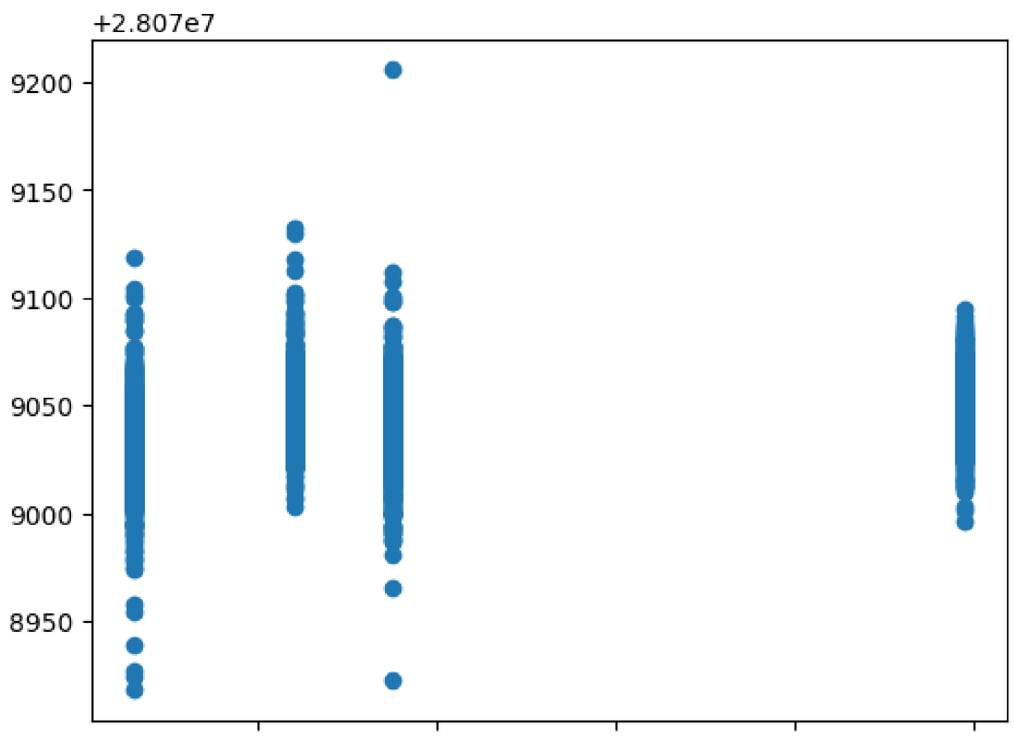
	Co-efficient
BEN	1.422345
CO	-37.828081
EBE	-1.629962
MXY	0.129792
NMHC	150.117500
NO_2	0.164140
NOx	-0.072211
OXY	-1.290530
O_3	-0.008479
PM10	-0.067689
PXY	1.548522
SO_2	0.863726
TCH	36.485332
TOL	-0.822906

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x789e7b5b5150>
```



20 40 60 80 100
+2.8079e7

ACCURACY

In [28]:

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

Out[28]:

```
0.17549329134935188
```

In [29]:

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

Out[29]:

```
0.1762470472451948
```

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]:

```
0.1732610422406159
```

In [33]:

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

Out[33]:

```
0.17524457166834073
```

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]:

```
0.036881185460887145
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.03369913728742602
```

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([ 0.        , -0.2613577 ,  0.05170798, -0.04653734,  0.1248526 ,
        0.1502707 , -0.06931803, -1.2610245 , -0.04399248,  0.06465597,
        0.23553947,  0.76815569,  1.61950313, -0.43112076])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079037.99552147
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.045639839248231784
```

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)))
```

```
29.030141230399117
```

```
29.03014125039117
1172.9763939722272
34.248742954628675
```

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]:

```
(33010, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(33010,)
```

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)
```

```
[28079035]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.7584974250227204
```

In [54]:

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

Out[54]:

```
2.3306153253214888e-23
```

In [55]:

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

Out[55]:

```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457490e-16]])
```

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]:

```
0.7290865021814479
```

In [61]:

In [61]:

```
rfc_best=grid_search.best_estimator_
```

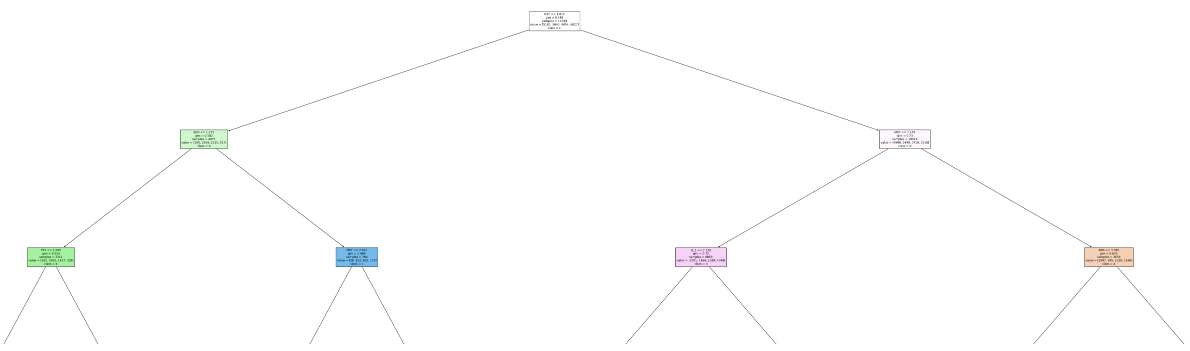
In [62]:

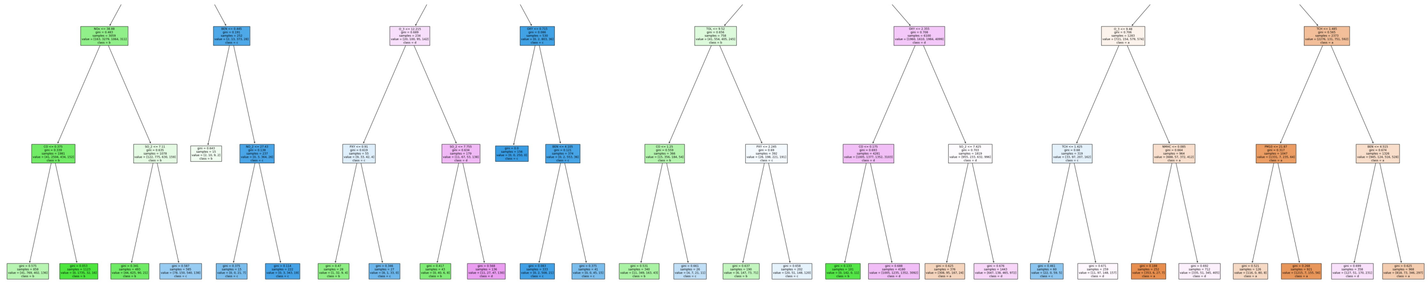
```
from sklearn.tree import plot_tree

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(0.46875, 0.9166666666666666, 'OXY <= 1.015\ngini = 0.749\nsamples = 14589\nvalue =
[5183, 5843, 6054, 6027]\nclass = c'),
 Text(0.22321428571428573, 0.75, 'BEN <= 1.235\ngini = 0.582\nsamples = 4075\nvalue = [18
5, 3394, 2335, 517]\nclass = b'),
 Text(0.11607142857142858, 0.5833333333333334, 'PXY <= 1.005\ngini = 0.524\nsamples = 331
1\nvalue = [165, 3292, 1437, 339]\nclass = b'),
 Text(0.07142857142857142, 0.4166666666666667, 'NOx <= 38.88\ngini = 0.483\nsamples = 305
9\nvalue = [163, 3279, 1064, 311]\nclass = b'),
 Text(0.03571428571428571, 0.25, 'CO <= 0.375\ngini = 0.339\nsamples = 1981\nvalue = [41,
2504, 434, 152]\nclass = b'),
 Text(0.017857142857142856, 0.08333333333333333, 'gini = 0.575\nsamples = 858\nvalue = [4
1, 769, 402, 136]\nclass = b'),
 Text(0.05357142857142857, 0.08333333333333333, 'gini = 0.053\nsamples = 1123\nvalue = [0
, 1735, 32, 16]\nclass = b'),
 Text(0.10714285714285714, 0.25, 'SO_2 <= 7.11\ngini = 0.635\nsamples = 1078\nvalue = [12
2, 775, 630, 159]\nclass = b'),
 Text(0.08928571428571429, 0.08333333333333333, 'gini = 0.341\nsamples = 493\nvalue = [44
, 625, 90, 21]\nclass = b'),
 Text(0.125, 0.08333333333333333, 'gini = 0.587\nsamples = 585\nvalue = [78, 150, 540, 13
8]\nclass = c'),
 Text(0.16071428571428573, 0.4166666666666667, 'BEN <= 0.445\ngini = 0.191\nsamples = 252
\nvalue = [2, 13, 373, 28]\nclass = c'),
 Text(0.14285714285714285, 0.25, 'gini = 0.643\nsamples = 15\nvalue = [2, 10, 9, 2]\nclas
s = b'),
 Text(0.17857142857142858, 0.25, 'NO_2 <= 27.43\ngini = 0.138\nsamples = 237\nvalue = [0,
3, 364, 26]\nclass = c'),
 Text(0.16071428571428573, 0.08333333333333333, 'gini = 0.375\nsamples = 15\nvalue = [0,
0, 21, 7]\nclass = c'),
 Text(0.19642857142857142, 0.08333333333333333, 'gini = 0.114\nsamples = 222\nvalue = [0,
3, 343, 19]\nclass = c'),
 Text(0.33035714285714285, 0.5833333333333334, 'MXY <= 2.595\ngini = 0.409\nsamples = 764
\nvalue = [20, 102, 898, 178]\nclass = c'),
 Text(0.2857142857142857, 0.4166666666666667, 'O_3 <= 12.215\ngini = 0.689\nsamples = 234
\nvalue = [20, 100, 95, 142]\nclass = d'),
 Text(0.25, 0.25, 'PXY <= 0.91\ngini = 0.619\nsamples = 55\nvalue = [9, 33, 42, 4]\nclass
= c'),
 Text(0.23214285714285715, 0.08333333333333333, 'gini = 0.47\nsamples = 28\nvalue = [1, 3
2, 9, 4]\nclass = b'),
 Text(0.26785714285714285, 0.08333333333333333, 'gini = 0.346\nsamples = 27\nvalue = [8,
1, 33, 0]\nclass = c'),
 Text(0.32142857142857145, 0.25, 'SO_2 <= 7.755\ngini = 0.634\nsamples = 179\nvalue = [11
, 67, 53, 138]\nclass = d'),
 Text(0.30357142857142855, 0.08333333333333333, 'gini = 0.417\nsamples = 43\nvalue = [0,
40, 6, 8]\nclass = b'),
 Text(0.3392857142857143, 0.08333333333333333, 'gini = 0.568\nsamples = 136\nvalue = [11,
27, 47, 130]\nclass = d'),
 Text(0.375, 0.4166666666666667, 'OXY <= 0.715\ngini = 0.086\nsamples = 530\nvalue = [0,
2, 803, 36]\nclass = c'),
 Text(0.35714285714285715, 0.25, 'gini = 0.0\nsamples = 156\nvalue = [0, 0, 250, 0]\nclas
s = c'),
 Text(0.39285714285714285, 0.25, 'BEN <= 4.105\ngini = 0.121\nsamples = 374\nvalue = [0,
2, 553, 36]\nclass = c'),
 Text(0.375, 0.08333333333333333, 'gini = 0.083\nsamples = 333\nvalue = [0, 2, 508, 21]\n
class = c'),
 Text(0.4107142857142857, 0.08333333333333333, 'gini = 0.375\nsamples = 41\nvalue = [0, 0
, 45, 15]\nclass = c'),
 Text(0.7142857142857143, 0.75, 'MXY <= 7.235\ngini = 0.73\nsamples = 10514\nvalue = [499
8, 2449, 3719, 5510]\nclass = d'),
 Text(0.5714285714285714, 0.5833333333333334, 'O_3 <= 7.535\ngini = 0.72\nsamples = 6858\
```

```
nvalue = [2001, 2164, 2389, 4344]\nclass = d'),
 Text(0.5, 0.4166666666666667, 'TOL <= 9.52\ngini = 0.656\nsamples = 758\nvalue = [41, 55
4, 405, 245]\nclass = b'),
 Text(0.4642857142857143, 0.25, 'CO <= 1.25\ngini = 0.559\nsamples = 366\nvalue = [15, 35
6, 184, 54]\nclass = b'),
 Text(0.44642857142857145, 0.08333333333333333, 'gini = 0.531\nsamples = 340\nvalue = [11
, 349, 163, 43]\nclass = b'),
 Text(0.48214285714285715, 0.08333333333333333, 'gini = 0.661\nsamples = 26\nvalue = [4,
7, 21, 11]\nclass = c'),
 Text(0.5357142857142857, 0.25, 'PXY <= 2.245\ngini = 0.69\nsamples = 392\nvalue = [26, 1
98, 221, 191]\nclass = c'),
 Text(0.5178571428571429, 0.08333333333333333, 'gini = 0.637\nsamples = 190\nvalue = [6,
147, 73, 71]\nclass = b'),
 Text(0.5535714285714286, 0.08333333333333333, 'gini = 0.658\nsamples = 202\nvalue = [20,
51, 148, 120]\nclass = c'),
 Text(0.6428571428571429, 0.4166666666666667, 'OXY <= 2.355\ngini = 0.708\nsamples = 6100
\nvalue = [1960, 1610, 1984, 4099]\nclass = d'),
 Text(0.6071428571428571, 0.25, 'CO <= 0.175\ngini = 0.693\nsamples = 4281\nvalue = [1005
, 1377, 1352, 3103]\nclass = d'),
 Text(0.5892857142857143, 0.08333333333333333, 'gini = 0.133\nsamples = 101\nvalue = [0,
142, 0, 11]\nclass = b'),
 Text(0.625, 0.08333333333333333, 'gini = 0.688\nsamples = 4180\nvalue = [1005, 1235, 135
2, 3092]\nclass = d'),
 Text(0.6785714285714286, 0.25, 'SO_2 <= 7.425\ngini = 0.703\nsamples = 1819\nvalue = [95
5, 233, 632, 996]\nclass = d'),
 Text(0.6607142857142857, 0.08333333333333333, 'gini = 0.625\nsamples = 376\nvalue = [308
, 95, 167, 24]\nclass = a'),
 Text(0.6964285714285714, 0.08333333333333333, 'gini = 0.676\nsamples = 1443\nvalue = [64
7, 138, 465, 972]\nclass = d'),
 Text(0.8571428571428571, 0.5833333333333334, 'BEN <= 3.365\ngini = 0.635\nsamples = 3656
\nvalue = [2997, 285, 1330, 1166]\nclass = a'),
 Text(0.7857142857142857, 0.4166666666666667, 'O_3 <= 9.48\ngini = 0.706\nsamples = 1283\
nvalue = [721, 154, 579, 574]\nclass = a'),
 Text(0.75, 0.25, 'TCH <= 1.425\ngini = 0.68\nsamples = 319\nvalue = [33, 97, 207, 162]\n
class = c'),
 Text(0.7321428571428571, 0.08333333333333333, 'gini = 0.461\nsamples = 60\nvalue = [22,
0, 59, 5]\nclass = c'),
 Text(0.7678571428571429, 0.08333333333333333, 'gini = 0.671\nsamples = 259\nvalue = [11,
97, 148, 157]\nclass = d'),
 Text(0.8214285714285714, 0.25, 'NMHC <= 0.085\ngini = 0.664\nsamples = 964\nvalue = [688
, 57, 372, 412]\nclass = a'),
 Text(0.8035714285714286, 0.08333333333333333, 'gini = 0.188\nsamples = 252\nvalue = [353
, 6, 27, 7]\nclass = a'),
 Text(0.8392857142857143, 0.08333333333333333, 'gini = 0.692\nsamples = 712\nvalue = [335
, 51, 345, 405]\nclass = d'),
 Text(0.9285714285714286, 0.4166666666666667, 'TCH <= 1.485\ngini = 0.565\nsamples = 2373
\nvalue = [2276, 131, 751, 592]\nclass = a'),
 Text(0.8928571428571429, 0.25, 'PM10 <= 21.87\ngini = 0.317\nsamples = 1047\nvalue = [13
31, 7, 235, 64]\nclass = a'),
 Text(0.875, 0.08333333333333333, 'gini = 0.521\nsamples = 126\nvalue = [116, 0, 80, 8]\n
class = a'),
 Text(0.9107142857142857, 0.08333333333333333, 'gini = 0.268\nsamples = 921\nvalue = [121
5, 7, 155, 56]\nclass = a'),
 Text(0.9642857142857143, 0.25, 'BEN <= 4.515\ngini = 0.674\nsamples = 1326\nvalue = [945
, 124, 516, 528]\nclass = a'),
 Text(0.9464285714285714, 0.08333333333333333, 'gini = 0.699\nsamples = 358\nvalue = [127
, 51, 170, 231]\nclass = d'),
 Text(0.9821428571428571, 0.08333333333333333, 'gini = 0.625\nsamples = 968\nvalue = [818
, 73, 346, 297]\nclass = a')]
```





Conclusion

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: 0.17549329134935188
Ridge Regression: 0.1732610422406159
Lasso Regression 0.03369913728742602
ElasticNet Regression: 0.045639839248231784
Logistic Regression: 0.7584974250227204
Random Forest: 0.7290865021814479
```

Logistic Regression is suitable for this dataset