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_2005.csv")
df
```

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

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	SO_2	TCH	TO
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91	10.65	NaN	4.62	NaN	Na
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.93	NaN	1.59	7.80	1.35	7.
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60	NaN	NaN	5.76	NaN	Na
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16	NaN	NaN	6.60	NaN	Na
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00	NaN	NaN	3.00	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00	NaN	NaN	6.68	1.37	2.
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95	1.49	1.00	7.06	1.28	0.
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31	2.93	NaN	13.20	1.28	0.
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00	NaN	NaN	5.81	1.25	0.
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67	2.11	1.09	11.07	1.30	1.

237000 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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	0.88	28079006
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

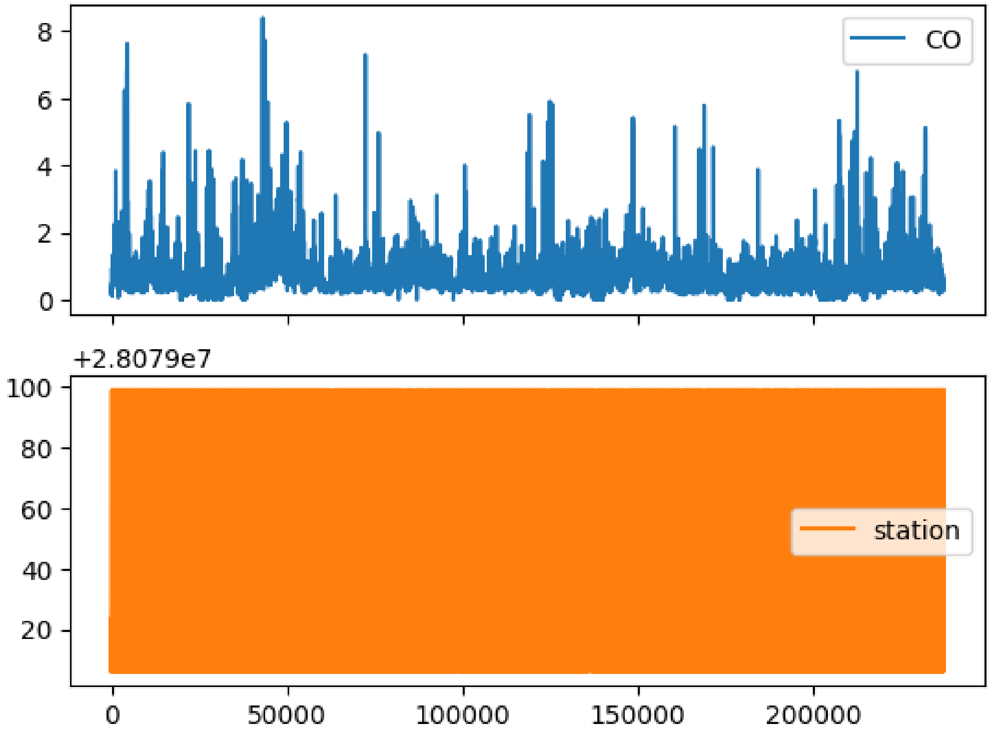
Line chart

In [7]:

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

Out[7]:

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



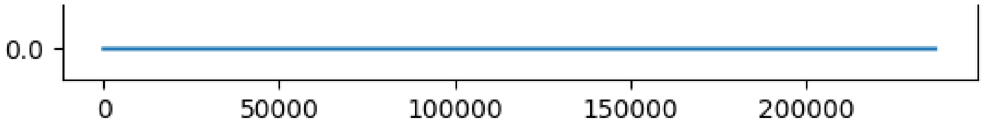
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

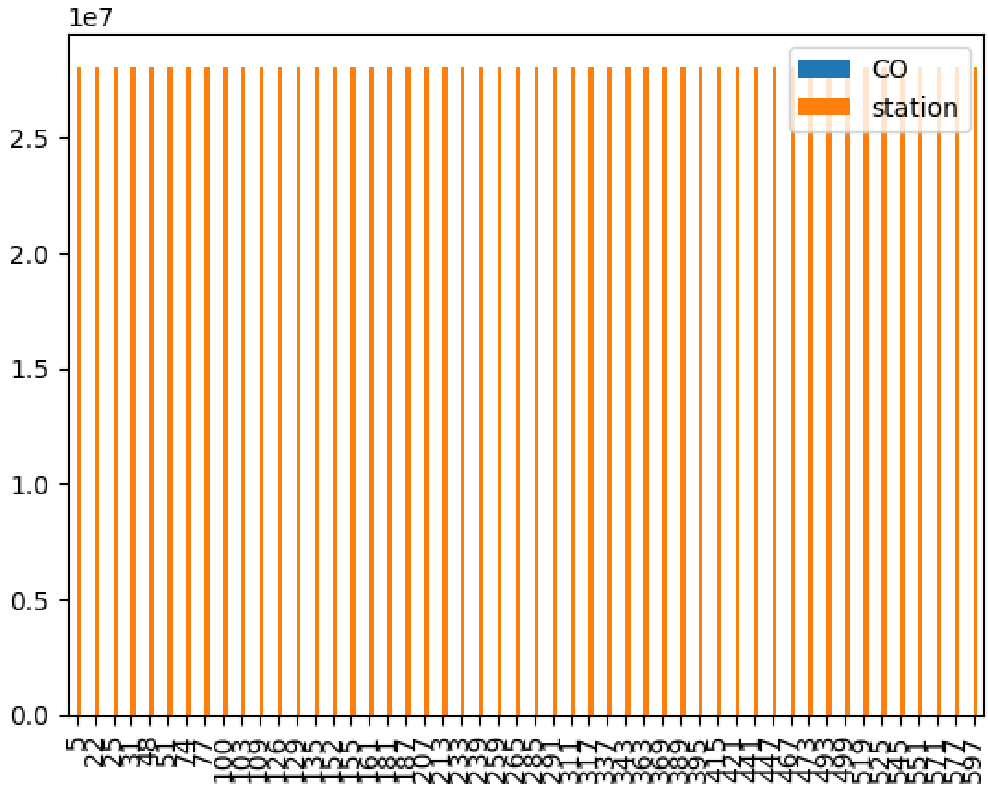
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



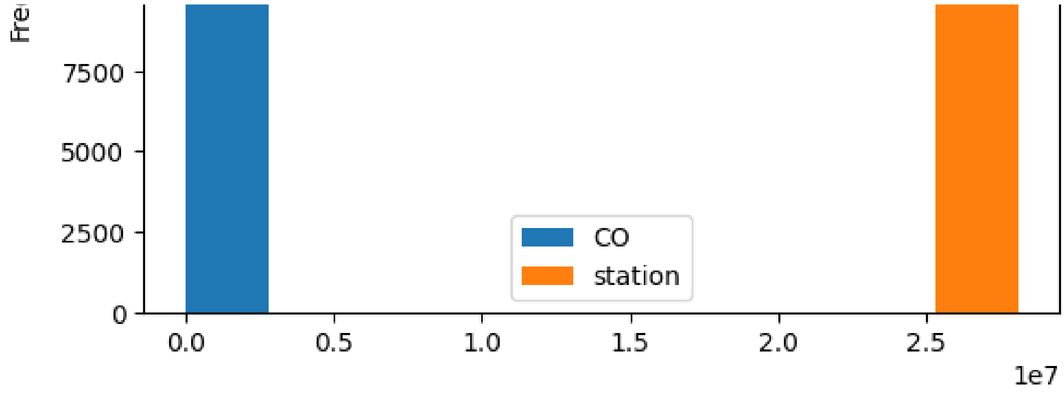
Histogram

In [11]:

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

Out[11]:

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



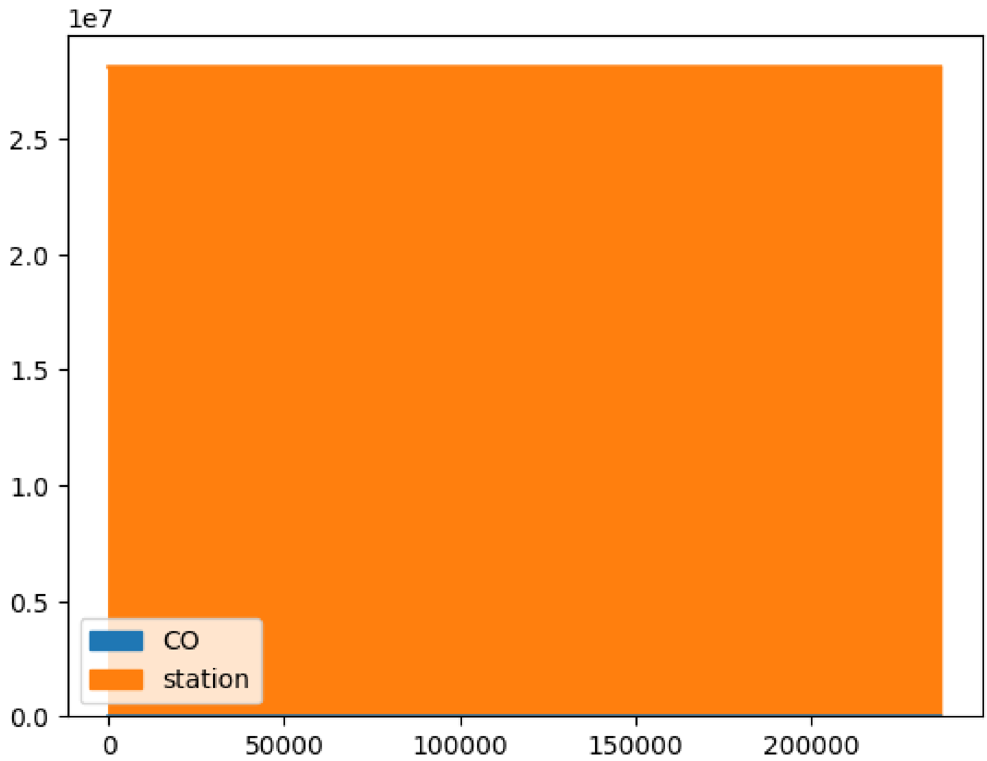
Area chart

In [12]:

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

Out[12]:

```
<Axes: >
```



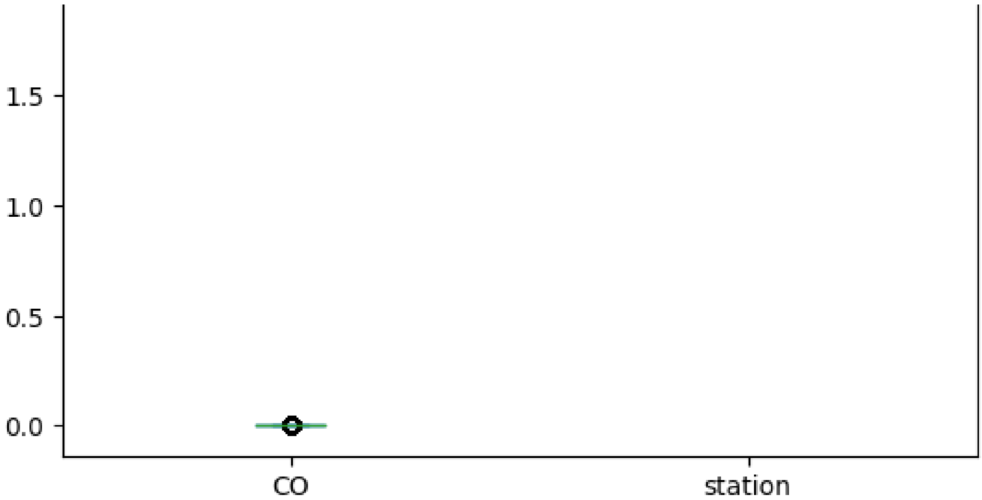
Box chart

In [13]:

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

Out[13]:

```
<Axes: >
```



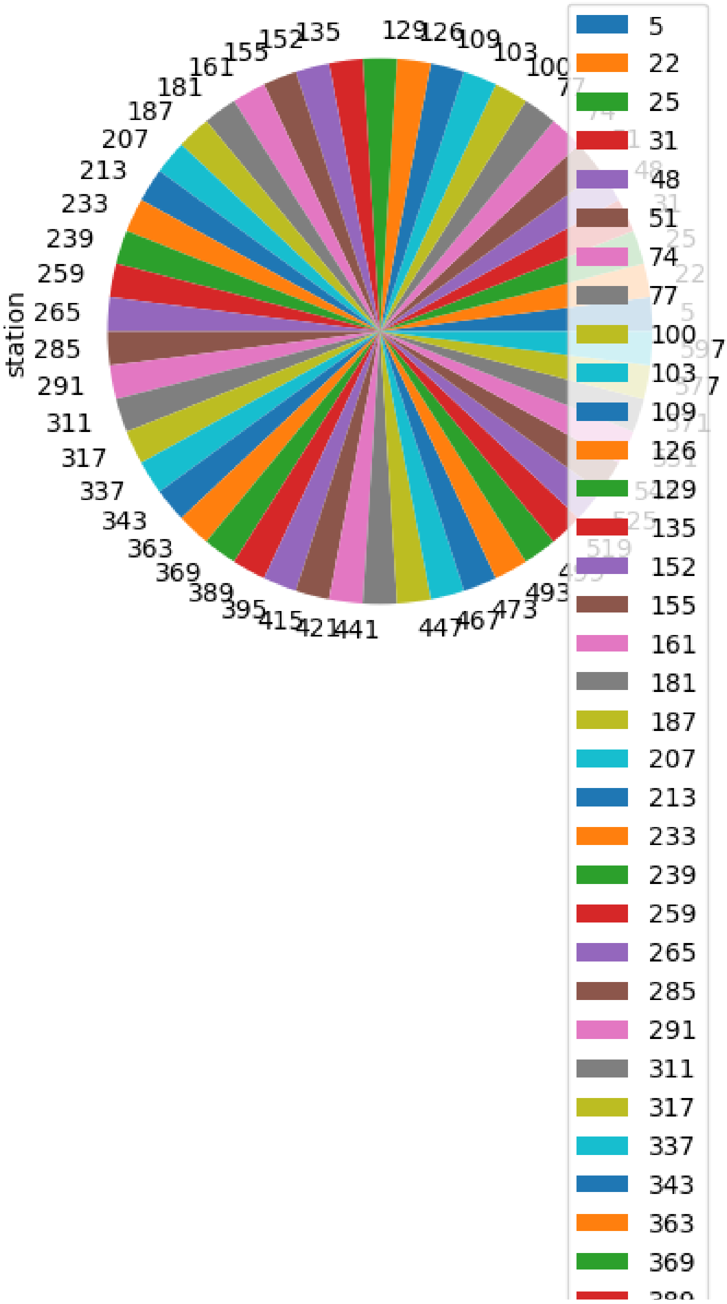
Pie chart

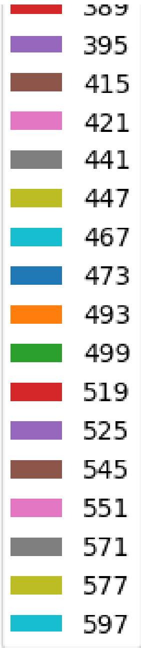
In [14]:

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

Out[14]:

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





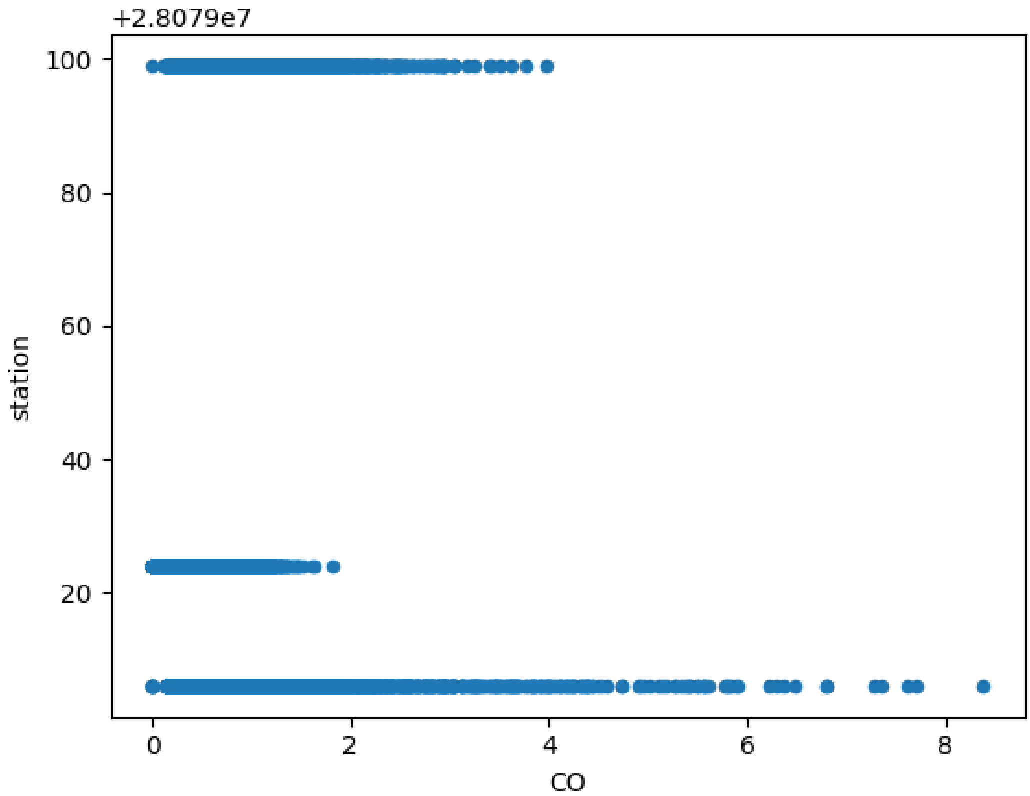
Scatter chart

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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     20070 non-null  object
 1   BEN      20070 non-null  float64
```

```
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	200
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	143.046536	2.774935	
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	136.582521	2.705508	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499	56.102499	1.000000	
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	105.699997	1.890000	
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	190.100006	3.620000	
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	1774.000000	38.680000	1

In [18]:

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

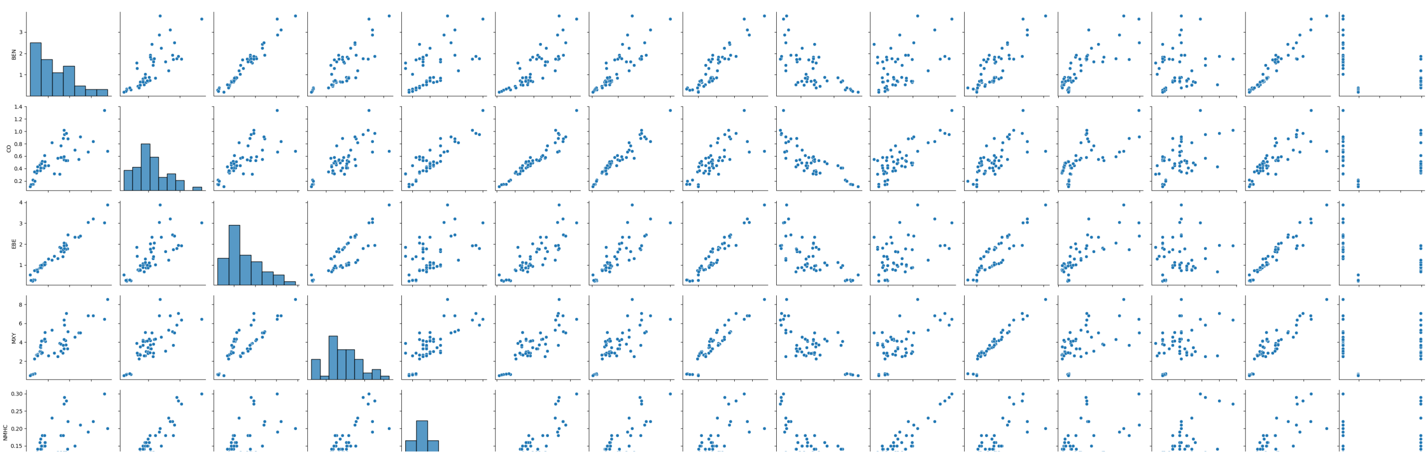
EDA AND VISUALIZATION

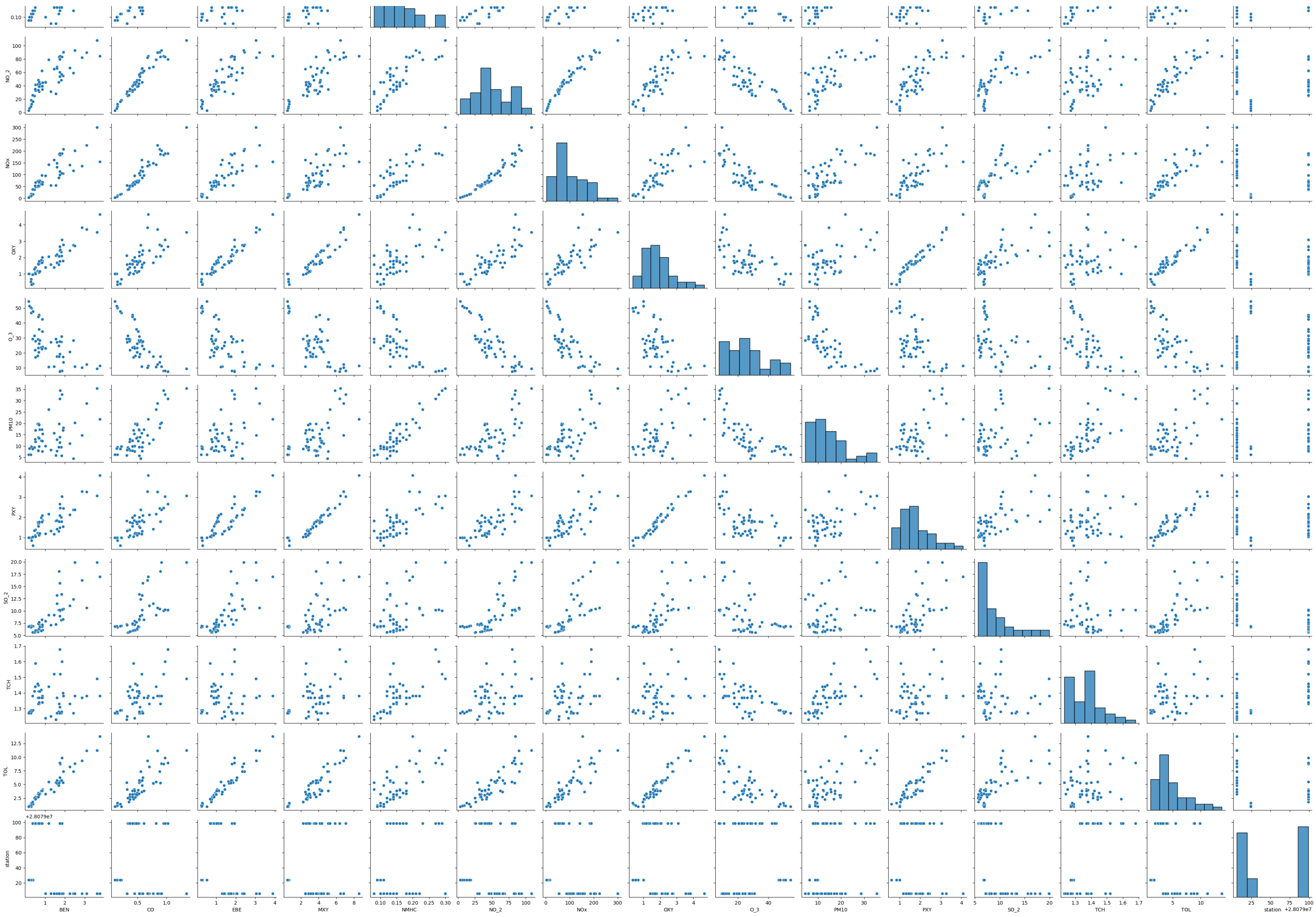
In [19]:

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

Out[19]:

```
&lt;seaborn.axisgrid.PairGrid at 0x7f748350aad0&gt;
```





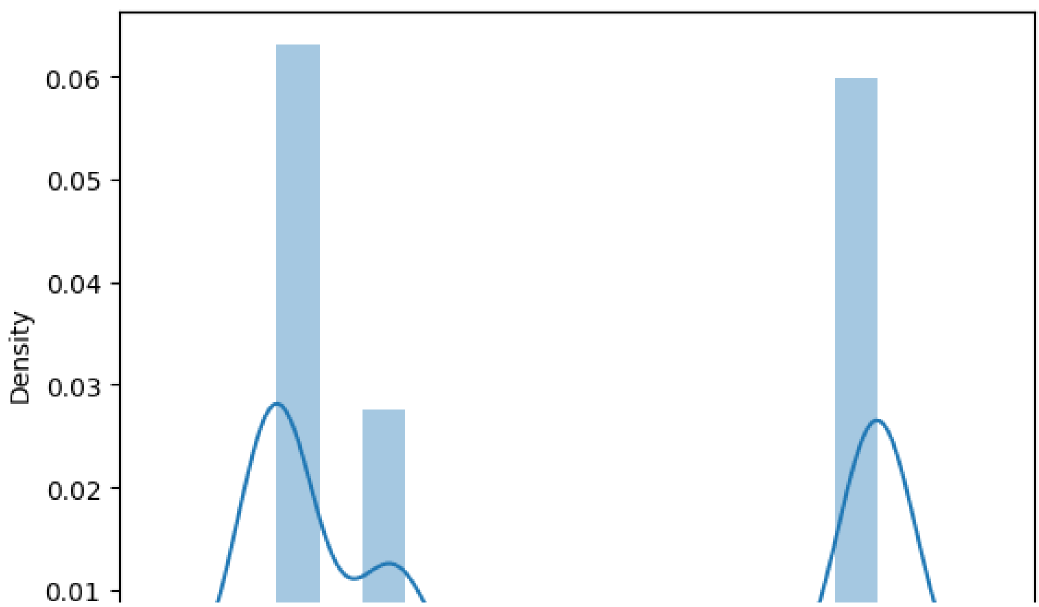
In [20]:

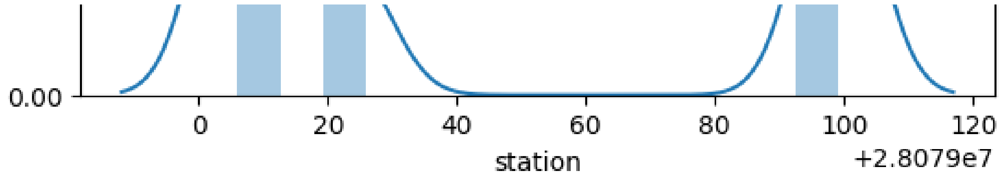
```
sns.distplot(df1['station'])
```

```
<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'>
```



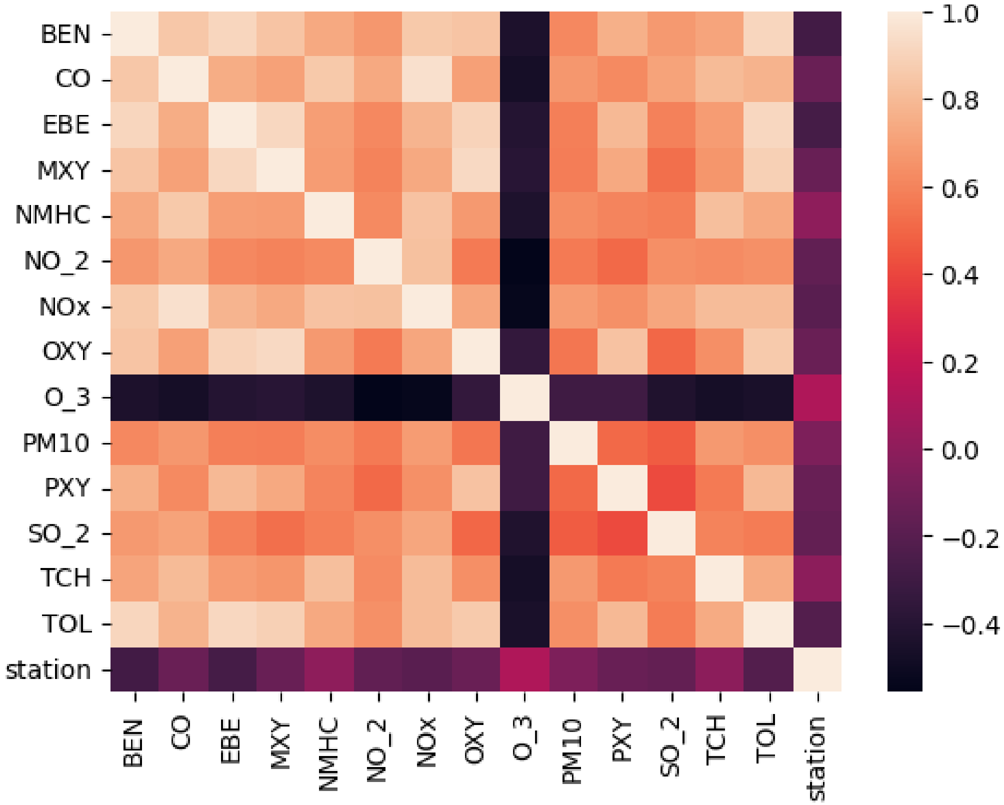


In [21]:

```
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]:

28078959.479894195

In [26]:

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

Out[26]:

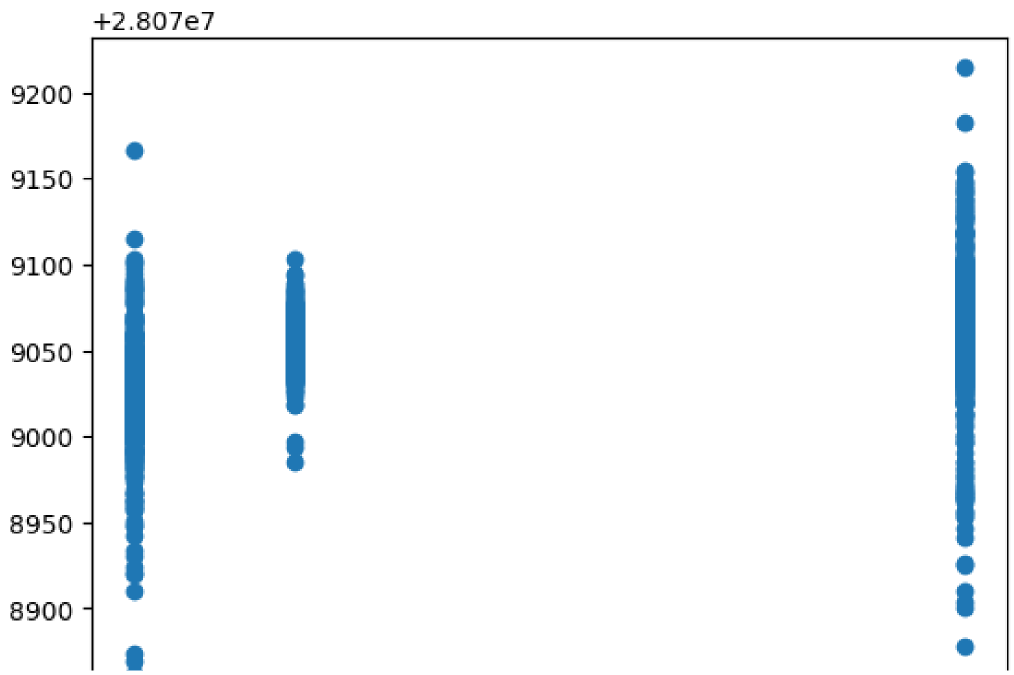
	Co-efficient
BEN	-10.011793
CO	37.347031
EBE	-12.403162
MXY	3.659439
NMHC	80.914883
NO_2	0.102935
NOx	-0.252799
OXY	2.984563
O_3	0.011059
PM10	0.040474
PXY	2.645328
SO_2	0.206198
TCH	63.296975
TOL	-0.748106

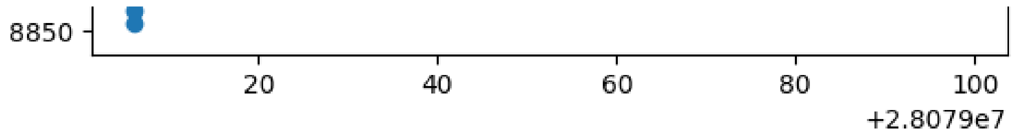
In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x7f746c99e3b0>





ACCURACY

In [28]:

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

Out[28]:

```
0.31020466985765904
```

In [29]:

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

Out[29]:

```
0.3010901501046742
```

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]:

```
0.3100017708242503
```

In [33]:

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

Out[33]:

```
0.30084307644038644
```

In [34]:

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

Out[34]:

```
▾ Lasso
Lasso(alpha=10)
```

Lasso(alpha=10)

In [35]:

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

Out[35]:

```
0.0675303013230486
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.05878207020106385
```

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([-5.73858158e+00,  1.42311526e+00, -7.04594809e+00,  2.59712093e+00,
        8.79781241e-01, -7.05217108e-02, -1.33839275e-03,  1.76533792e+00,
       -2.09839743e-02,  2.32167362e-01,  1.39167719e+00,  1.54356269e-01,
        1.57454738e+00, -8.77724188e-01])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079050.85067011
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.17596282162432753
```

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)))
```

```
36.86428888023068
1532.3099527649367
39.144730842923636
```

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]:

```
(20070, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(20070,)
```

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)
```

```
[28079006]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.879023418036871
```

In [54]:

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

Out[54]:

```
0.9998966777270014
```

In [55]:

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

Out[55]:

```
array([[9.99896678e-01, 3.20032883e-30, 1.03322273e-04]])
```

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]:

0.8624098160653053

In [61]:

```
rfc_best=grid_search.best_estimator_
```

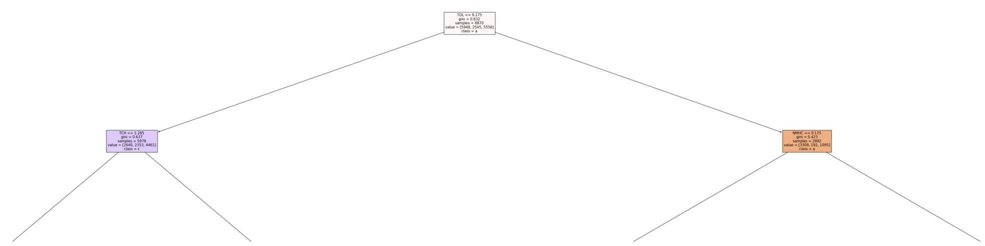
In [62]:

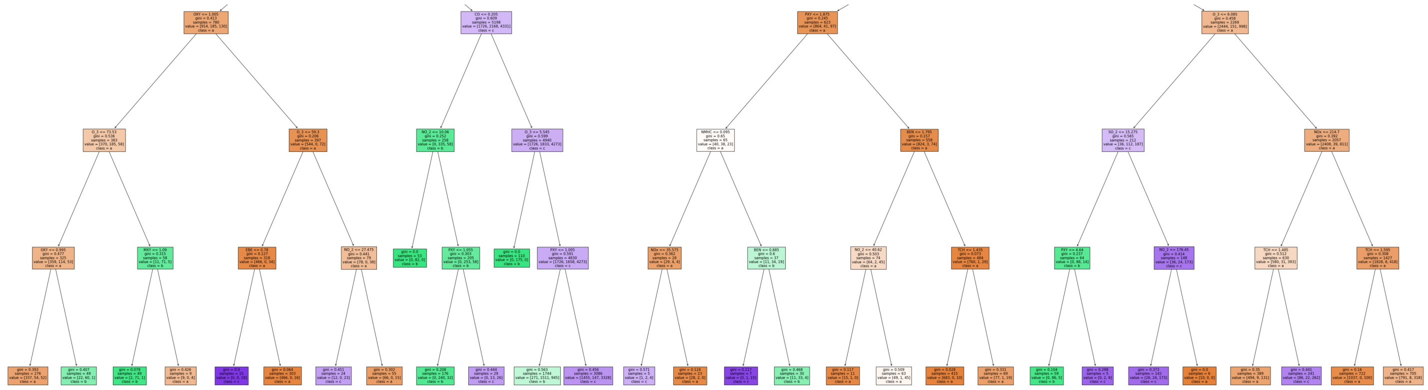
```
from sklearn.tree import plot_tree

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(0.47767857142857145, 0.9166666666666666, 'TOL <= 9.175\ngini = 0.632\nsamples = 887
0\nvalue = [5948, 2545, 5556]\nclass = a'),
 Text(0.24107142857142858, 0.75, 'TCH <= 1.265\ngini = 0.637\nsamples = 5978\nvalue = [26
40, 2353, 4461]\nclass = c'),
 Text(0.14285714285714285, 0.5833333333333334, 'OXY <= 1.005\ngini = 0.413\nsamples = 780
\nvalue = [914, 185, 130]\nclass = a'),
 Text(0.07142857142857142, 0.4166666666666667, 'O_3 <= 73.53\ngini = 0.536\nsamples = 383
\nvalue = [370, 185, 58]\nclass = a'),
 Text(0.03571428571428571, 0.25, 'OXY <= 0.995\ngini = 0.477\nsamples = 325\nvalue = [359
, 114, 53]\nclass = a'),
 Text(0.017857142857142856, 0.08333333333333333, 'gini = 0.393\nsamples = 276\nvalue = [3
37, 54, 52]\nclass = a'),
 Text(0.05357142857142857, 0.08333333333333333, 'gini = 0.407\nsamples = 49\nvalue = [22,
60, 1]\nclass = b'),
 Text(0.10714285714285714, 0.25, 'MXY <= 1.09\ngini = 0.315\nsamples = 58\nvalue = [11, 7
1, 5]\nclass = b'),
 Text(0.08928571428571429, 0.08333333333333333, 'gini = 0.079\nsamples = 49\nvalue = [2,
71, 1]\nclass = b'),
 Text(0.125, 0.08333333333333333, 'gini = 0.426\nsamples = 9\nvalue = [9, 0, 4]\nclass =
a'),
 Text(0.21428571428571427, 0.4166666666666667, 'O_3 <= 59.3\ngini = 0.206\nsamples = 397\
nvalue = [544, 0, 72]\nclass = a'),
 Text(0.17857142857142858, 0.25, 'EBE <= 0.78\ngini = 0.127\nsamples = 318\nvalue = [466,
0, 34]\nclass = a'),
 Text(0.16071428571428573, 0.08333333333333333, 'gini = 0.0\nsamples = 15\nvalue = [0, 0,
18]\nclass = c'),
 Text(0.19642857142857142, 0.08333333333333333, 'gini = 0.064\nsamples = 303\nvalue = [46
6, 0, 16]\nclass = a'),
 Text(0.25, 0.25, 'NO_2 <= 27.475\ngini = 0.441\nsamples = 79\nvalue = [78, 0, 38]\nclass
= a'),
 Text(0.23214285714285715, 0.08333333333333333, 'gini = 0.451\nsamples = 24\nvalue = [12,
0, 23]\nclass = c'),
 Text(0.26785714285714285, 0.08333333333333333, 'gini = 0.302\nsamples = 55\nvalue = [66,
0, 15]\nclass = a'),
 Text(0.3392857142857143, 0.5833333333333334, 'CO <= 0.205\ngini = 0.609\nsamples = 5198\
nvalue = [1726, 2168, 4331]\nclass = c'),
 Text(0.30357142857142855, 0.4166666666666667, 'NO_2 <= 10.06\ngini = 0.252\nsamples = 25
8\nvalue = [0, 335, 58]\nclass = b'),
 Text(0.2857142857142857, 0.25, 'gini = 0.0\nsamples = 53\nvalue = [0, 82, 0]\nclass = b'
),
 Text(0.32142857142857145, 0.25, 'PXY <= 1.055\ngini = 0.303\nsamples = 205\nvalue = [0,
253, 58]\nclass = b'),
 Text(0.30357142857142855, 0.08333333333333333, 'gini = 0.208\nsamples = 176\nvalue = [0,
240, 32]\nclass = b'),
 Text(0.3392857142857143, 0.08333333333333333, 'gini = 0.444\nsamples = 29\nvalue = [0, 1
3, 26]\nclass = c'),
 Text(0.375, 0.4166666666666667, 'O_3 <= 5.545\ngini = 0.599\nsamples = 4940\nvalue = [17
26, 1833, 4273]\nclass = c'),
 Text(0.35714285714285715, 0.25, 'gini = 0.0\nsamples = 110\nvalue = [0, 175, 0]\nclass =
b'),
 Text(0.39285714285714285, 0.25, 'PXY <= 1.005\ngini = 0.591\nsamples = 4830\nvalue = [17
26, 1658, 4273]\nclass = c'),
 Text(0.375, 0.08333333333333333, 'gini = 0.563\nsamples = 1744\nvalue = [271, 1511, 945]
\nclass = b'),
 Text(0.4107142857142857, 0.08333333333333333, 'gini = 0.456\nsamples = 3086\nvalue = [14
55, 147, 3328]\nclass = c'),
```

```
 Text(0.7142857142857143, 0.75, 'NMHC <= 0.175\ngini = 0.423\nsamples = 2892\nvalue = [33
08, 192, 1095]\nclass = a'),
 Text(0.5714285714285714, 0.5833333333333334, 'PXY <= 1.875\ngini = 0.245\nsamples = 623\
nvalue = [864, 41, 97]\nclass = a'),
 Text(0.5, 0.4166666666666667, 'NMHC <= 0.095\ngini = 0.65\nsamples = 65\nvalue = [40, 38
, 23]\nclass = a'),
 Text(0.4642857142857143, 0.25, 'NOx <= 35.575\ngini = 0.362\nsamples = 28\nvalue = [29,
4, 4]\nclass = a'),
 Text(0.44642857142857145, 0.08333333333333333, 'gini = 0.571\nsamples = 5\nvalue = [1, 2
, 4]\nclass = c'),
 Text(0.48214285714285715, 0.08333333333333333, 'gini = 0.124\nsamples = 23\nvalue = [28,
2, 0]\nclass = a'),
 Text(0.5357142857142857, 0.25, 'BEN <= 0.885\ngini = 0.6\nsamples = 37\nvalue = [11, 34,
19]\nclass = b'),
 Text(0.5178571428571429, 0.08333333333333333, 'gini = 0.117\nsamples = 7\nvalue = [0, 1,
15]\nclass = c'),
 Text(0.5535714285714286, 0.08333333333333333, 'gini = 0.468\nsamples = 30\nvalue = [11,
33, 4]\nclass = b'),
 Text(0.6428571428571429, 0.4166666666666667, 'BEN <= 1.795\ngini = 0.157\nsamples = 558\
nvalue = [824, 3, 74]\nclass = a'),
 Text(0.6071428571428571, 0.25, 'NO_2 <= 40.62\ngini = 0.503\nsamples = 74\nvalue = [64,
2, 45]\nclass = a'),
 Text(0.5892857142857143, 0.08333333333333333, 'gini = 0.117\nsamples = 11\nvalue = [15,
1, 0]\nclass = a'),
 Text(0.625, 0.08333333333333333, 'gini = 0.509\nsamples = 63\nvalue = [49, 1, 45]\nclass
= a'),
 Text(0.6785714285714286, 0.25, 'TCH <= 1.435\ngini = 0.073\nsamples = 484\nvalue = [760,
1, 29]\nclass = a'),
 Text(0.6607142857142857, 0.08333333333333333, 'gini = 0.028\nsamples = 415\nvalue = [683
, 0, 10]\nclass = a'),
 Text(0.6964285714285714, 0.08333333333333333, 'gini = 0.331\nsamples = 69\nvalue = [77,
1, 19]\nclass = a'),
 Text(0.8571428571428571, 0.5833333333333334, 'O_3 <= 6.085\ngini = 0.458\nsamples = 2269
\nvalue = [2444, 151, 998]\nclass = a'),
 Text(0.7857142857142857, 0.4166666666666667, 'SO_2 <= 15.275\ngini = 0.565\nsamples = 21
2\nvalue = [36, 112, 187]\nclass = c'),
 Text(0.75, 0.25, 'PXY <= 4.64\ngini = 0.237\nsamples = 64\nvalue = [0, 88, 14]\nclass =
b'),
 Text(0.7321428571428571, 0.08333333333333333, 'gini = 0.104\nsamples = 59\nvalue = [0, 8
6, 5]\nclass = b'),
 Text(0.7678571428571429, 0.08333333333333333, 'gini = 0.298\nsamples = 5\nvalue = [0, 2,
9]\nclass = c'),
 Text(0.8214285714285714, 0.25, 'NO_2 <= 176.45\ngini = 0.414\nsamples = 148\nvalue = [36
, 24, 173]\nclass = c'),
 Text(0.8035714285714286, 0.08333333333333333, 'gini = 0.373\nsamples = 142\nvalue = [26,
24, 173]\nclass = c'),
 Text(0.8392857142857143, 0.08333333333333333, 'gini = 0.0\nsamples = 6\nvalue = [10, 0,
0]\nclass = a'),
 Text(0.9285714285714286, 0.4166666666666667, 'NOx <= 214.7\ngini = 0.392\nsamples = 2057
\nvalue = [2408, 39, 811]\nclass = a'),
 Text(0.8928571428571429, 0.25, 'TCH <= 1.485\ngini = 0.512\nsamples = 630\nvalue = [580,
31, 393]\nclass = a'),
 Text(0.875, 0.08333333333333333, 'gini = 0.35\nsamples = 389\nvalue = [494, 9, 131]\ncla
ss = a'),
 Text(0.9107142857142857, 0.08333333333333333, 'gini = 0.441\nsamples = 241\nvalue = [86,
22, 262]\nclass = c'),
 Text(0.9642857142857143, 0.25, 'TCH <= 1.595\ngini = 0.308\nsamples = 1427\nvalue = [182
8, 8, 418]\nclass = a'),
 Text(0.9464285714285714, 0.08333333333333333, 'gini = 0.16\nsamples = 722\nvalue = [1037
, 0, 100]\nclass = a'),
 Text(0.9821428571428571, 0.08333333333333333, 'gini = 0.417\nsamples = 705\nvalue = [791
, 8, 318]\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: 0.31020466985765904
Ridge Regression: 0.3100017708242503
Lasso Regression 0.05878207020106385
ElasticNet Regression: 0.17596282162432753
Logistic Regression: 0.879023418036871
Random Forest: 0.8624098160653053
```

Logistic Regression is suitable for this dataset