Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:

```
df=pd.read_csv("madrid_2005")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	SO_2	TCH	TOL	static
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	NaN	2807900
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	7.98	2807903
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	NaN	2807900
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	NaN	2807900
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	NaN	2807903
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	2.75	2807902
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	0.38	2807902
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	0.56	2807902
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	0.22	2807902
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	1.29	2807909

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]:

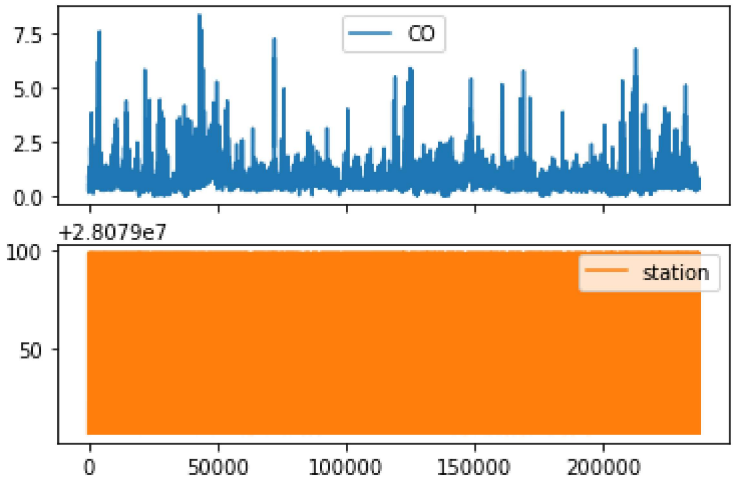
	CO	station
5	0.88	28079006
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

Line chart

In [7]:
```
1 data.plot.line(subplots=True)
```

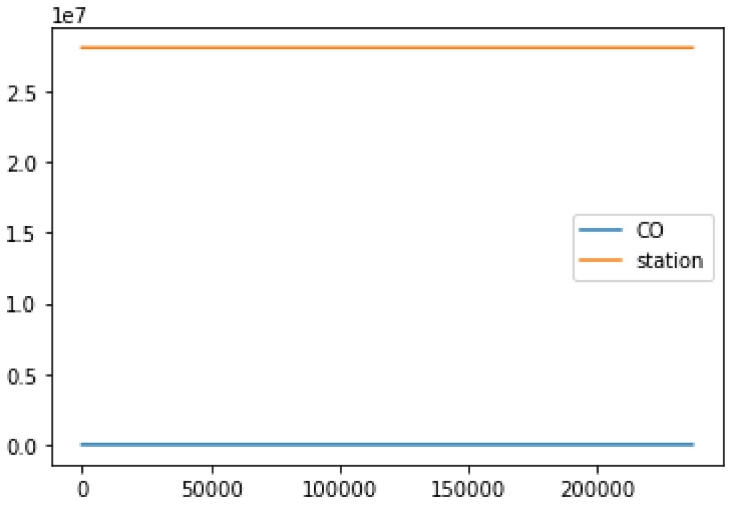
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [8]:
```
1 data.plot.line()
```

Out[8]:
```
<AxesSubplot:>
```

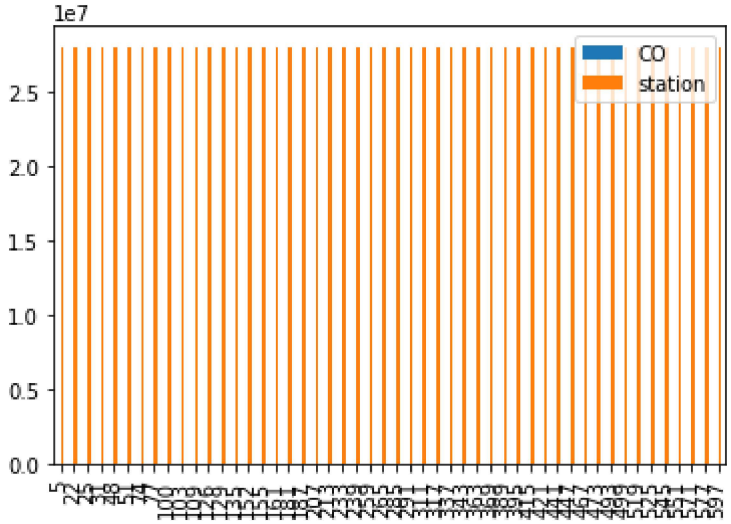


Bar chart

In [9]:
```
1 b=data[0:50]
```

In [10]:
```
1 b.plot.bar()
```

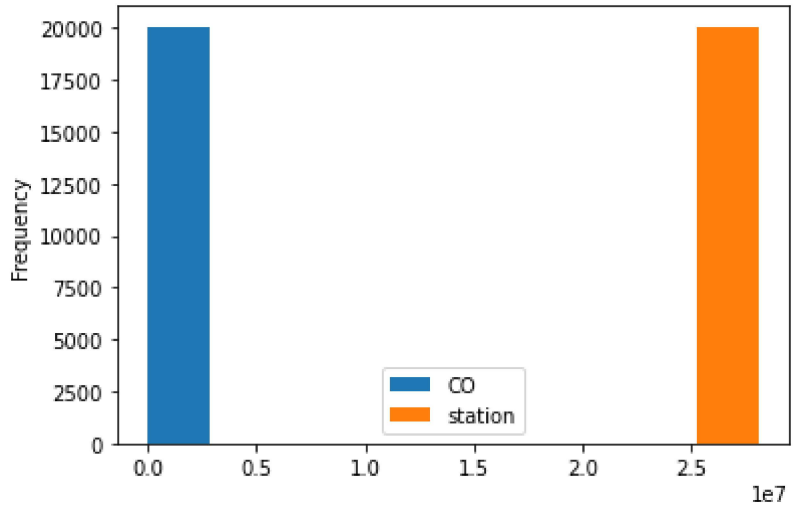
Out[10]: <AxesSubplot:>



Histogram

In [11]:
```
1 data.plot.hist()
```

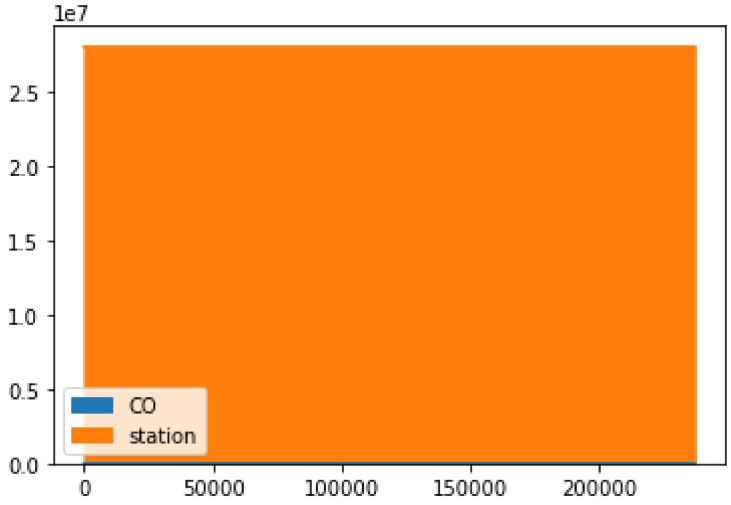
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [12]:
```
1 data.plot.area()
```

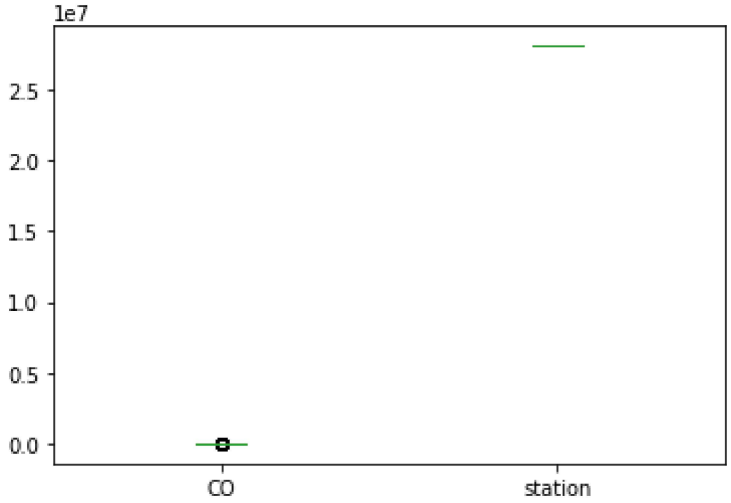
Out[12]: <AxesSubplot:>



Box chart

In [13]:
```
1 data.plot.box()
```

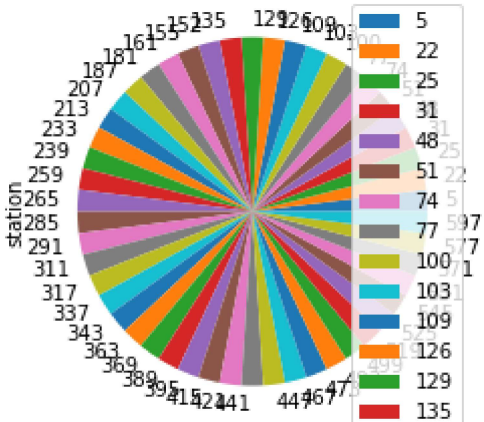
Out[13]: <AxesSubplot:>



Pie chart

In [14]:
```
1 b.plot.pie(y='station' )
```

Out[14]:
```
<AxesSubplot:ylabel='station'>
```

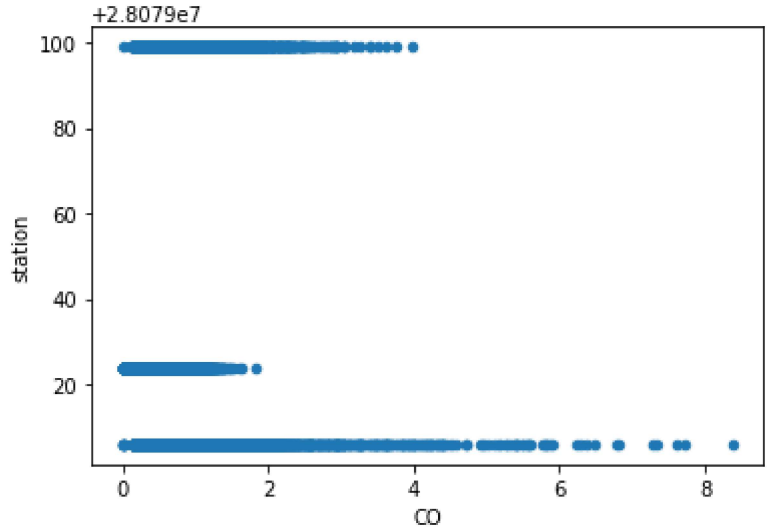

station
5
22
25
31
48
51
74
77
100
103
109
126
129
135
152
155
161
181
187
207
213
233
239
259
265
285
291
311
317
337
343
363
369
389
395
415
421
441
447
467
473
493
499
519
525
545
551
571
577

597

Scatter chart

In [15]:
```
1 data.plot.scatter(x='CO' ,y='station')
```

Out[15]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [16]:
```
1 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
```

In [17]:
```
1 df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	143.046536	2.774935	32.121323
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	136.582521	2.705508	23.954714
min	0.000000	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	11.540000
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	105.699997	1.890000	26.255000
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	190.100006	3.620000	47.150002
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	1774.000000	38.680000	149.800003

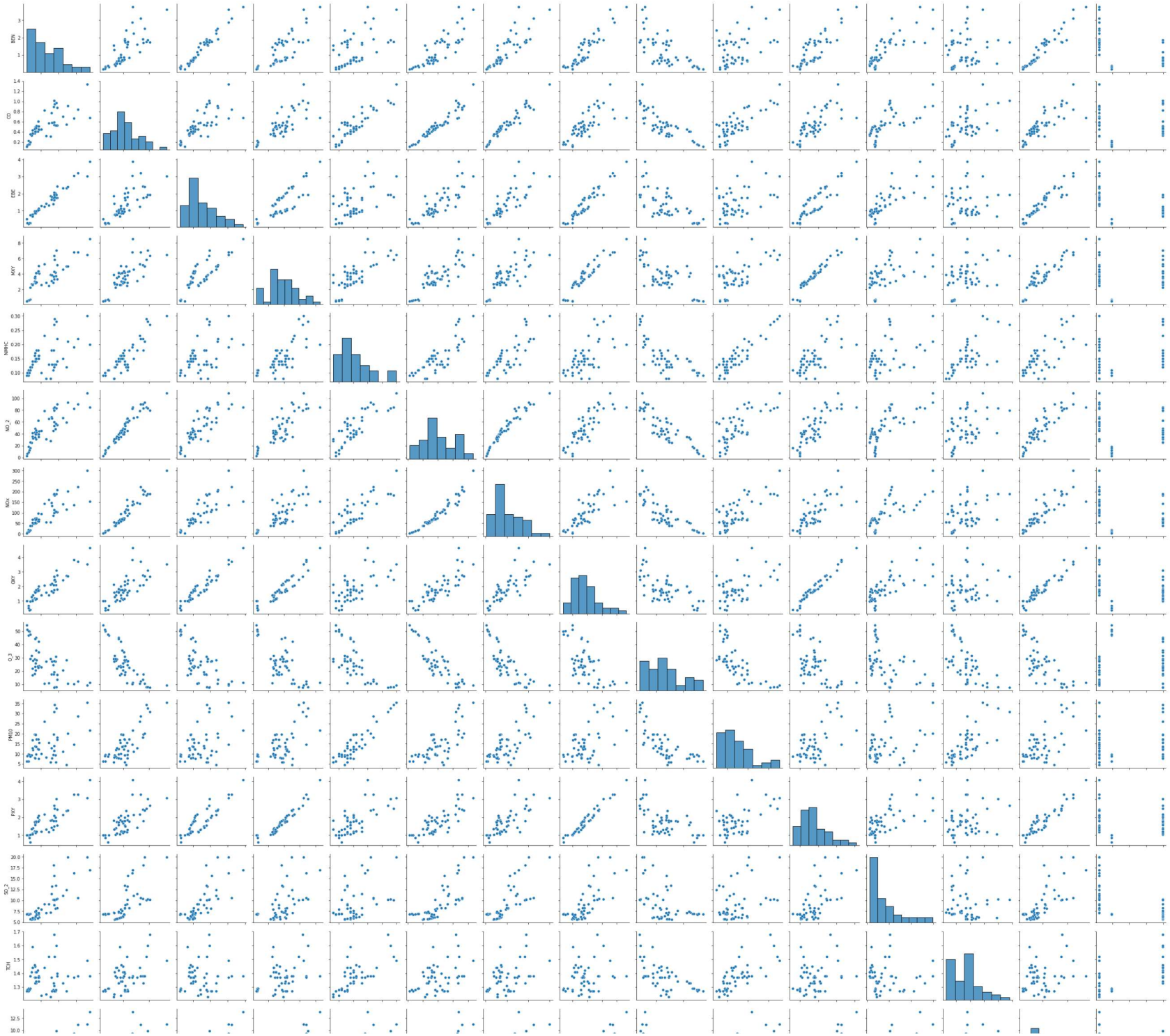
In [18]:

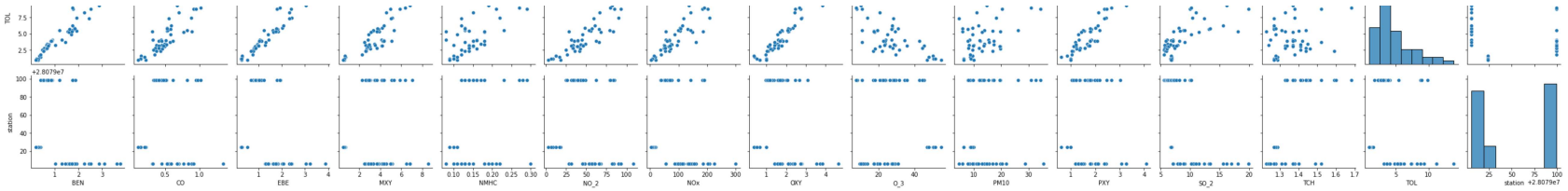
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [19]:
```
1 sns.pairplot(df1[0:50])
```

Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x2338ba80af0>
```

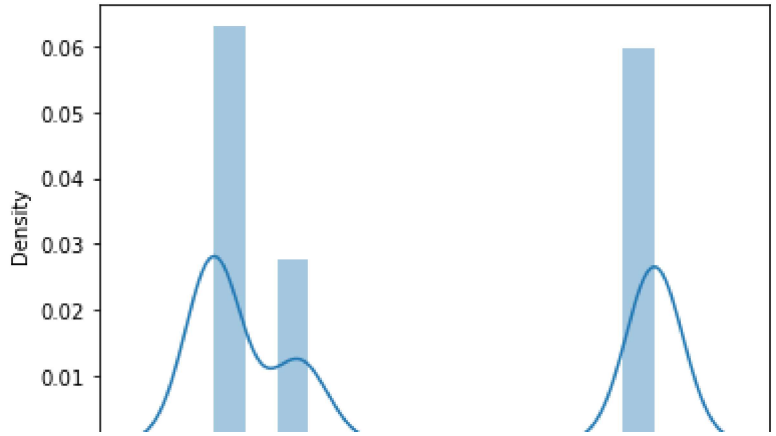

BEN
CO
EBE
MXY
NMHC
NO_2
NOx
OXY
O_3
PM10
PXY
SO_2
TCH



In [20]:
```
1 sns.distplot(df1['station'])
```

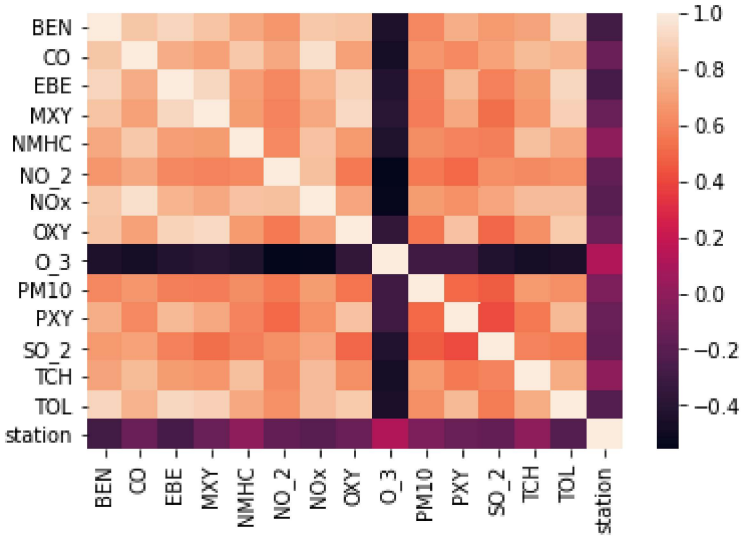
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a de
precated function and will be removed in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [21]:
```
1 sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
1 x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
2        'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
3 y=df['station']
```

In [23]:
```
1 from sklearn.model_selection import train_test_split
2 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()
```

In [25]:
```
lr.intercept_
```

Out[25]:
```
28078959.43807614
```

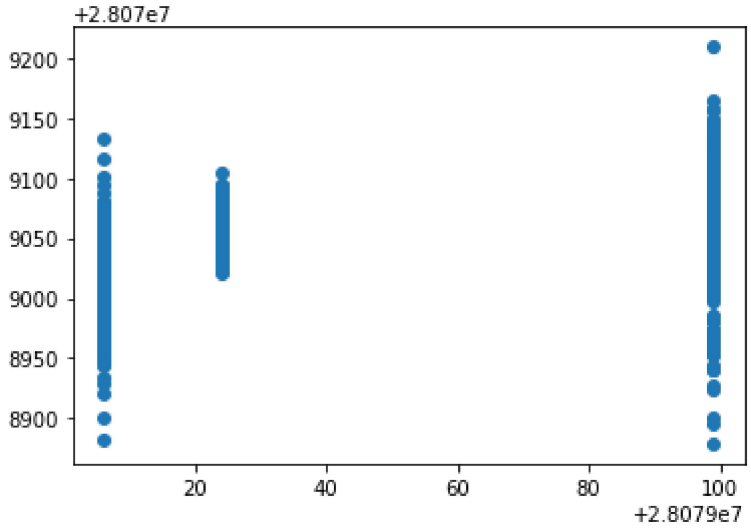
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-8.957694
CO	38.918415
EBE	-13.198253
MXY	3.642700
NMHC	77.780575
NO_2	0.125812
NOx	-0.274325
OXY	3.167298
O_3	0.008482
PM10	0.048127
PXY	2.543436
SO_2	0.180207
TCH	62.510949
TOL	-0.544119

In [27]:
```
1 prediction =lr.predict(x_test)
2 plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x2339a38f580>



ACCURACY

In [28]:
```
1 lr.score(x_test,y_test)
```

Out[28]: 0.31417673670713897

In [29]:
```
1 lr.score(x_train,y_train)
```

Out[29]: 0.2998431212306839

Ridge and Lasso

In [30]:
```
1 from sklearn.linear_model import Ridge,Lasso
```

In [31]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[31]: Ridge(alpha=10)

Accuracy(Ridge)

In [32]:
```
rr.score(x_test,y_test)
```

Out[32]: 0.31355614379783414

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]: 0.2996231647869222

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]: Lasso(alpha=10)

Accuracy(Lasso)

In [35]:
```
la.score(x_train,y_train)
```

Out[35]: 0.062201095160138054

In [36]:
```
la.score(x_test,y_test)
```

Out[36]: 0.0691094813594253

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[37]: ElasticNet()

In [38]:
```
1 en.coef_
```

Out[38]:
```
array([-5.44544857,  1.50574277, -7.32302084,  2.61297051,  0.92964019,
       -0.0573147 , -0.00983528,  1.89670049, -0.02333062,  0.22792507,
        1.40928848,  0.13714068,  1.59571901, -0.77222586])
```

In [39]:
```
1 en.intercept_
```

Out[39]:
```
28079050.10880432
```

In [40]:
```
1 prediction=en.predict(x_test)
```

In [41]:
```
1 en.score(x_test,y_test)
```

Out[41]:
```
0.18421578404776207
```

Evaluation Metrics

In [42]:
```
1 from sklearn import metrics
2 print(metrics.mean_absolute_error(y_test,prediction))
3 print(metrics.mean_squared_error(y_test,prediction))
4 print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
36.82616234889504
1527.3078091416137
39.080785677128006
```

Logistic Regression

In [43]:
```
1 from sklearn.linear_model import LogisticRegression
```

In [44]:
```
1 feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
2        'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
3 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(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], dtype=int64)

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]: 0.879023418036871

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]: 0.9998967601812779

In [55]:
```
logr.predict_proba(observation)
```

Out[55]: array([[9.99896760e-01, 3.21124597e-30, 1.03239819e-04]])

Random Forest

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[57]:
```
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(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.8659692442383612
```

In [61]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:

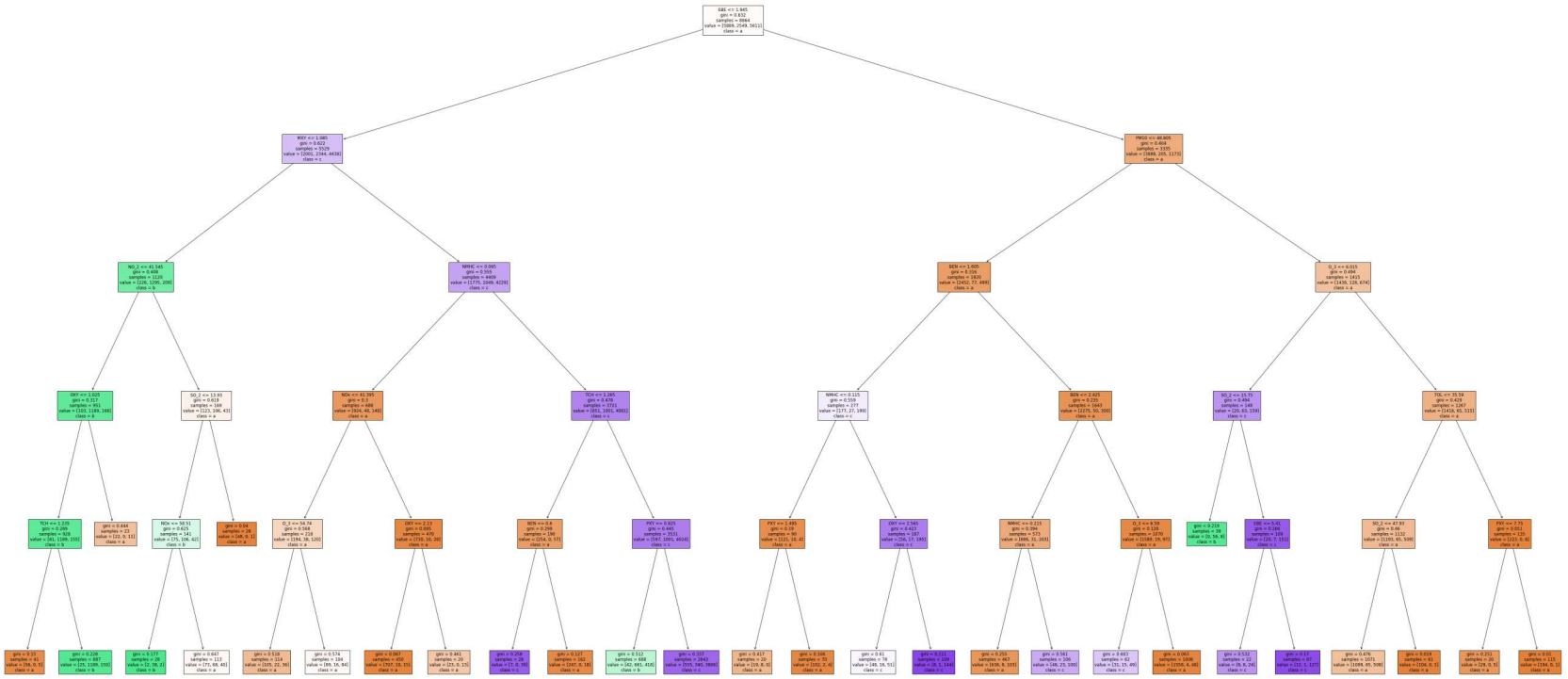
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2092.5, 1993.2, 'EBE <= 1.945\ngini = 0.632\nsamples = 8864\nvalue = [5889, 2549, 5611]\nclass = a'),
 Text(901.3846153846154, 1630.8000000000002, 'MXY <= 1.085\ngini = 0.622\nsamples = 5529\nvalue = [2001, 2344, 4438]\nclass = c'),
 Text(429.23076923076917, 1268.4, 'NO_2 <= 41.545\ngini = 0.408\nsamples = 1120\nvalue = [226, 1295, 209]\nclass = b'),
 Text(257.53846153846155, 906.0, 'OXY <= 1.025\ngini = 0.317\nsamples = 951\nvalue = [103, 1189, 166]\nclass = b'),
 Text(171.69230769230768, 543.5999999999999, 'TCH <= 1.235\ngini = 0.289\nsamples = 928\nvalue = [81, 1189, 155]\nclass = b'),
 Text(85.84615384615384, 181.19999999999982, 'gini = 0.15\nsamples = 41\nvalue = [56, 0, 5]\nclass = a'),
 Text(257.53846153846155, 181.19999999999982, 'gini = 0.228\nsamples = 887\nvalue = [25, 1189, 150]\nclass = b'),
 Text(343.38461538461536, 543.5999999999999, 'gini = 0.444\nsamples = 23\nvalue = [22, 0, 11]\nclass = a'),
 Text(600.9230769230769, 906.0, 'SO_2 <= 13.93\ngini = 0.619\nsamples = 169\nvalue = [123, 106, 43]\nclass = a'),
 Text(515.0769230769231, 543.5999999999999, 'NOx <= 58.51\ngini = 0.625\nsamples = 141\nvalue = [75, 106, 42]\nclass = b'),
 Text(429.23076923076917, 181.19999999999982, 'gini = 0.177\nsamples = 28\nvalue = [2, 38, 2]\nclass = b'),
 Text(600.9230769230769, 181.19999999999982, 'gini = 0.647\nsamples = 113\nvalue = [73, 68, 40]\nclass = a'),
 Text(686.7692307692307, 543.5999999999999, 'gini = 0.04\nsamples = 28\nvalue = [48, 0, 1]\nclass = a'),
 Text(1373.5384615384614, 1268.4, 'NMHC <= 0.065\ngini = 0.555\nsamples = 4409\nvalue = [1775, 1049, 4229]\nclass = c'),
 Text(1030.1538461538462, 906.0, 'NOx <= 41.395\ngini = 0.3\nsamples = 688\nvalue = [924, 48, 148]\nclass = a'),
 Text(858.4615384615383, 543.5999999999999, 'O_3 <= 54.74\ngini = 0.568\nsamples = 218\nvalue = [194, 38, 120]\nclass = a'),
 Text(772.6153846153845, 181.19999999999982, 'gini = 0.518\nsamples = 114\nvalue = [105, 22, 36]\nclass = a'),
 Text(944.3076923076923, 181.19999999999982, 'gini = 0.574\nsamples = 104\nvalue = [89, 16, 84]\nclass = a'),
 Text(1201.8461538461538, 543.5999999999999, 'OXY <= 2.13\ngini = 0.095\nsamples = 470\nvalue = [730, 10, 28]\nclass = a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.067\nsamples = 450\nvalue = [707, 10, 15]\nclass = a'),
 Text(1287.6923076923076, 181.19999999999982, 'gini = 0.461\nsamples = 20\nvalue = [23, 0, 13]\nclass = a'),
 Text(1716.9230769230767, 906.0, 'TCH <= 1.265\ngini = 0.478\nsamples = 3721\nvalue = [851, 1001, 4081]\nclass = c'),
 Text(1545.230769230769, 543.5999999999999, 'BEN <= 0.6\ngini = 0.299\nsamples = 190\nvalue = [254, 0, 57]\nclass = a'),
 Text(1459.3846153846152, 181.19999999999982, 'gini = 0.258\nsamples = 28\nvalue = [7, 0, 39]\nclass = c'),
 Text(1631.0769230769229, 181.19999999999982, 'gini = 0.127\nsamples = 162\nvalue = [247, 0, 18]\nclass = a'),
 Text(1888.6153846153845, 543.5999999999999, 'PXY <= 0.825\ngini = 0.445\nsamples = 3531\nvalue = [597, 100
```

```
1, 4024]\nclass = c'),
 Text(1802.7692307692307, 181.19999999999982, 'gini = 0.512\nsamples = 688\nvalue = [42, 661, 418]\nclass =
b'),
 Text(1974.4615384615383, 181.19999999999982, 'gini = 0.337\nsamples = 2843\nvalue = [555, 340, 3606]\nclass
= c'),
 Text(3283.6153846153843, 1630.8000000000002, 'PM10 <= 48.805\ngini = 0.404\nsamples = 3335\nvalue = [3888,
205, 1173]\nclass = a'),
 Text(2747.076923076923, 1268.4, 'BEN <= 1.605\ngini = 0.316\nsamples = 1920\nvalue = [2452, 77, 499]\nclass
= a'),
 Text(2403.6923076923076, 906.0, 'NMHC <= 0.115\ngini = 0.559\nsamples = 277\nvalue = [177, 27, 199]\nclass
= c'),
 Text(2232.0, 543.5999999999999, 'PXY <= 1.495\ngini = 0.19\nsamples = 90\nvalue = [121, 10, 4]\nclass =
a'),
 Text(2146.153846153846, 181.19999999999982, 'gini = 0.417\nsamples = 20\nvalue = [19, 8, 0]\nclass = a'),
 Text(2317.846153846154, 181.19999999999982, 'gini = 0.106\nsamples = 70\nvalue = [102, 2, 4]\nclass = a'),
 Text(2575.3846153846152, 543.5999999999999, 'OXY <= 2.565\ngini = 0.423\nsamples = 187\nvalue = [56, 17, 19
5]\nclass = c'),
 Text(2489.5384615384614, 181.19999999999982, 'gini = 0.61\nsamples = 78\nvalue = [48, 16, 51]\nclass = c'),
 Text(2661.230769230769, 181.19999999999982, 'gini = 0.111\nsamples = 109\nvalue = [8, 1, 144]\nclass = c'),
 Text(3090.461538461538, 906.0, 'BEN <= 2.425\ngini = 0.235\nsamples = 1643\nvalue = [2275, 50, 300]\nclass
= a'),
 Text(2918.7692307692305, 543.5999999999999, 'NMHC <= 0.215\ngini = 0.394\nsamples = 573\nvalue = [686, 31,
203]\nclass = a'),
 Text(2832.9230769230767, 181.19999999999982, 'gini = 0.255\nsamples = 467\nvalue = [638, 8, 103]\nclass =
a'),
 Text(3004.6153846153843, 181.19999999999982, 'gini = 0.561\nsamples = 106\nvalue = [48, 23, 100]\nclass =
c'),
 Text(3262.1538461538457, 543.5999999999999, 'O_3 <= 6.59\ngini = 0.128\nsamples = 1070\nvalue = [1589, 19,
97]\nclass = a'),
 Text(3176.307692307692, 181.19999999999982, 'gini = 0.603\nsamples = 62\nvalue = [31, 15, 49]\nclass = c'),
 Text(3347.9999999999995, 181.19999999999982, 'gini = 0.063\nsamples = 1008\nvalue = [1558, 4, 48]\nclass =
a'),
 Text(3820.1538461538457, 1268.4, 'O_3 <= 6.015\ngini = 0.494\nsamples = 1415\nvalue = [1436, 128, 674]\ncla
ss = a'),
 Text(3519.6923076923076, 906.0, 'SO_2 <= 15.75\ngini = 0.494\nsamples = 148\nvalue = [20, 63, 159]\nclass =
c'),
 Text(3433.8461538461534, 543.5999999999999, 'gini = 0.219\nsamples = 39\nvalue = [0, 56, 8]\nclass = b'),
 Text(3605.5384615384614, 543.5999999999999, 'EBE <= 5.41\ngini = 0.266\nsamples = 109\nvalue = [20, 7, 151]
\nclass = c'),
 Text(3519.6923076923076, 181.19999999999982, 'gini = 0.532\nsamples = 22\nvalue = [8, 6, 24]\nclass = c'),
 Text(3691.3846153846152, 181.19999999999982, 'gini = 0.17\nsamples = 87\nvalue = [12, 1, 127]\nclass = c'),
 Text(4120.615384615385, 906.0, 'TOL <= 35.59\ngini = 0.429\nsamples = 1267\nvalue = [1416, 65, 515]\nclass
= a'),
```

```
 Text(3948.9230769230767, 543.5999999999999, 'SO_2 <= 47.93\ngini = 0.46\nsamples = 1132\nvalue = [1193, 65,
509]\nclass = a'),
 Text(3863.076923076923, 181.19999999999982, 'gini = 0.476\nsamples = 1071\nvalue = [1089, 65, 508]\nclass =
a'),
 Text(4034.7692307692305, 181.19999999999982, 'gini = 0.019\nsamples = 61\nvalue = [104, 0, 1]\nclass = a'),
 Text(4292.307692307692, 543.5999999999999, 'PXY <= 7.75\ngini = 0.051\nsamples = 135\nvalue = [223, 0, 6]\n
class = a'),
 Text(4206.461538461538, 181.19999999999982, 'gini = 0.251\nsamples = 20\nvalue = [29, 0, 5]\nclass = a'),
 Text(4378.153846153846, 181.19999999999982, 'gini = 0.01\nsamples = 115\nvalue = [194, 0, 1]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression0.2998431212306839

Ridge Regression:0.2996231647869222

Lasso Regression:0.062201095160138054

ElasticNet Regression:0.18421578404776207

Logistic Regression:0.879023418036871

Random Forest:0.8659692442383612

Logistic Regression is suitable for this dataset