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_2013.csv").fillna(1)
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2013-11-01 01:00:00	1.0	0.6	1.0	1.0	135.0	74.0	1.0	1.0	1.0	7.0	1.0	1.0	28079004
1	2013-11-01 01:00:00	1.5	0.5	1.3	1.0	71.0	83.0	2.0	23.0	16.0	12.0	1.0	8.3	28079008
2	2013-11-01 01:00:00	3.9	1.0	2.8	1.0	49.0	70.0	1.0	1.0	1.0	1.0	1.0	9.0	28079011
3	2013-11-01 01:00:00	1.0	0.5	1.0	1.0	82.0	87.0	3.0	1.0	1.0	1.0	1.0	1.0	28079016
4	2013-11-01 01:00:00	1.0	1.0	1.0	1.0	242.0	111.0	2.0	1.0	1.0	12.0	1.0	1.0	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209875	2013-03-01 00:00:00	1.0	0.4	1.0	1.0	8.0	39.0	52.0	1.0	1.0	1.0	1.0	1.0	28079056
209876	2013-03-01 00:00:00	1.0	0.4	1.0	1.0	1.0	11.0	1.0	6.0	1.0	2.0	1.0	1.0	28079057
209877	2013-03-01 00:00:00	1.0	1.0	1.0	1.0	2.0	4.0	75.0	1.0	1.0	1.0	1.0	1.0	28079058
209878	2013-03-01 00:00:00	1.0	1.0	1.0	1.0	2.0	11.0	52.0	1.0	1.0	1.0	1.0	1.0	28079059
209879	2013-03-01 00:00:00	1.0	1.0	1.0	1.0	1.0	10.0	75.0	3.0	1.0	1.0	1.0	1.0	28079060

209880 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.dropna()
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 209880 entries, 0 to 209879
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209880 non-null  object 
 1   BEN      209880 non-null  float64
 2   CO       209880 non-null  float64
 3   EBE      209880 non-null  float64
 4   NMHC     209880 non-null  float64
 5   NO       209880 non-null  float64
 6   NO_2     209880 non-null  float64
 7   O_3      209880 non-null  float64
 8   PM10     209880 non-null  float64
 9   PM25     209880 non-null  float64
 10  SO_2     209880 non-null  float64
 11  TCH      209880 non-null  float64
 12  TOL      209880 non-null  float64
 13  station  209880 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 24.0+ MB
```

In [6]:
```
data=df[['CO' ,'station']]
data
```

Out[6]:

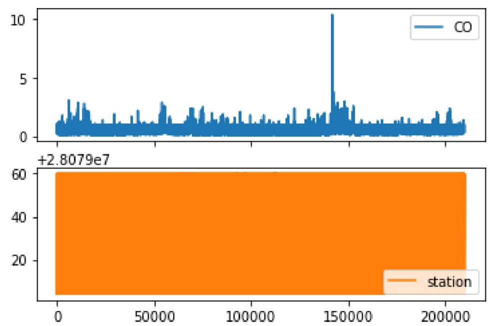
	CO	station
0	0.6	28079004
1	0.5	28079008
2	1.0	28079011
3	0.5	28079016
4	1.0	28079017
...	...	...
209875	0.4	28079056
209876	0.4	28079057
209877	1.0	28079058
209878	1.0	28079059
209879	1.0	28079060

209880 rows × 2 columns

Line chart

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

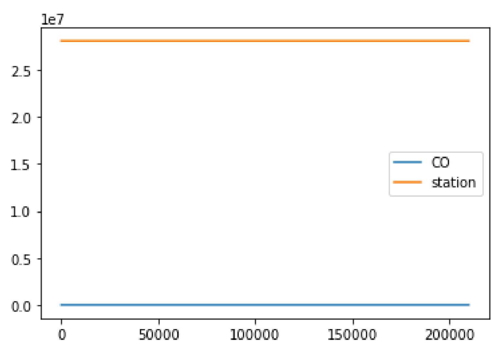
Out[7]: `array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)`



Line chart

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

Out[8]: `<AxesSubplot:>`

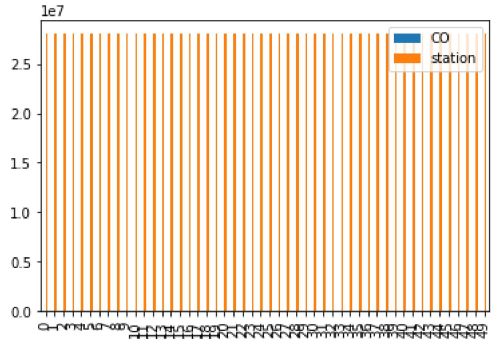


Bar chart

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

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

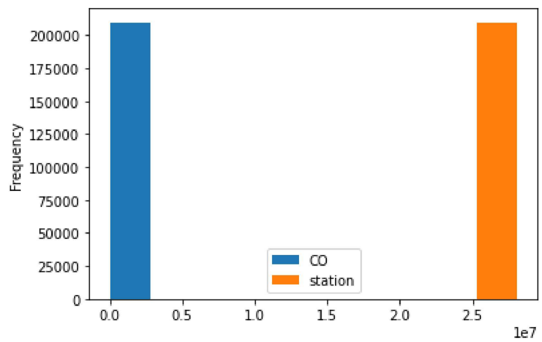
Out[10]: <AxesSubplot:>



Histogram

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

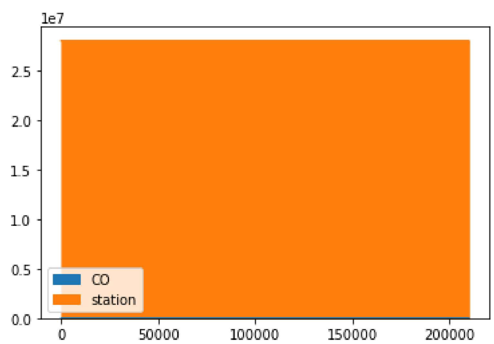
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

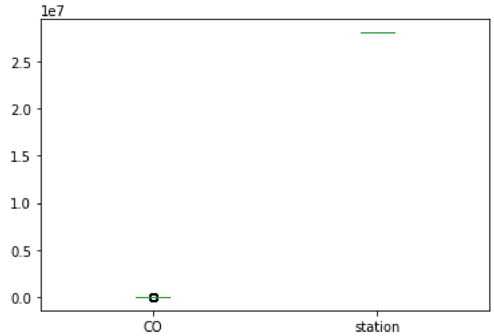
Out[12]: <AxesSubplot:>



Box chart

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

Out[13]: <AxesSubplot:>



Pie chart

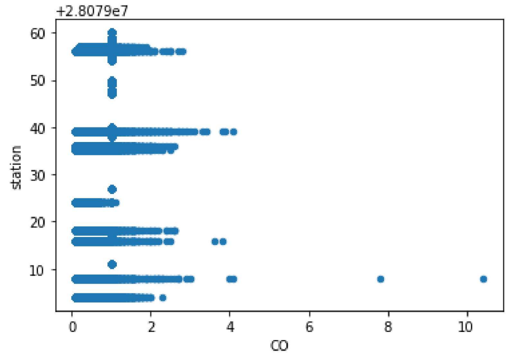
In [14]:
```
b.plot.pie(y='station' )
```

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

Scatter chart

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

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



In [16]:
```
df.info()
```

```
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   date     209880 non-null  object
 1   BEN      209880 non-null  float64
 2   CO       209880 non-null  float64
 3   EBE      209880 non-null  float64
 4   NMHC     209880 non-null  float64
 5   NO       209880 non-null  float64
 6   NO_2     209880 non-null  float64
 7   O_3      209880 non-null  float64
 8   PM10     209880 non-null  float64
 9   PM25     209880 non-null  float64
 10  SO_2     209880 non-null  float64
 11  TCH      209880 non-null  float64
 12  TOL      209880 non-null  float64
 13  station  209880 non-null  int64
dtypes: float64(12), int64(1), object(1)
memory usage: 24.0+ MB
```

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

Out[17]:

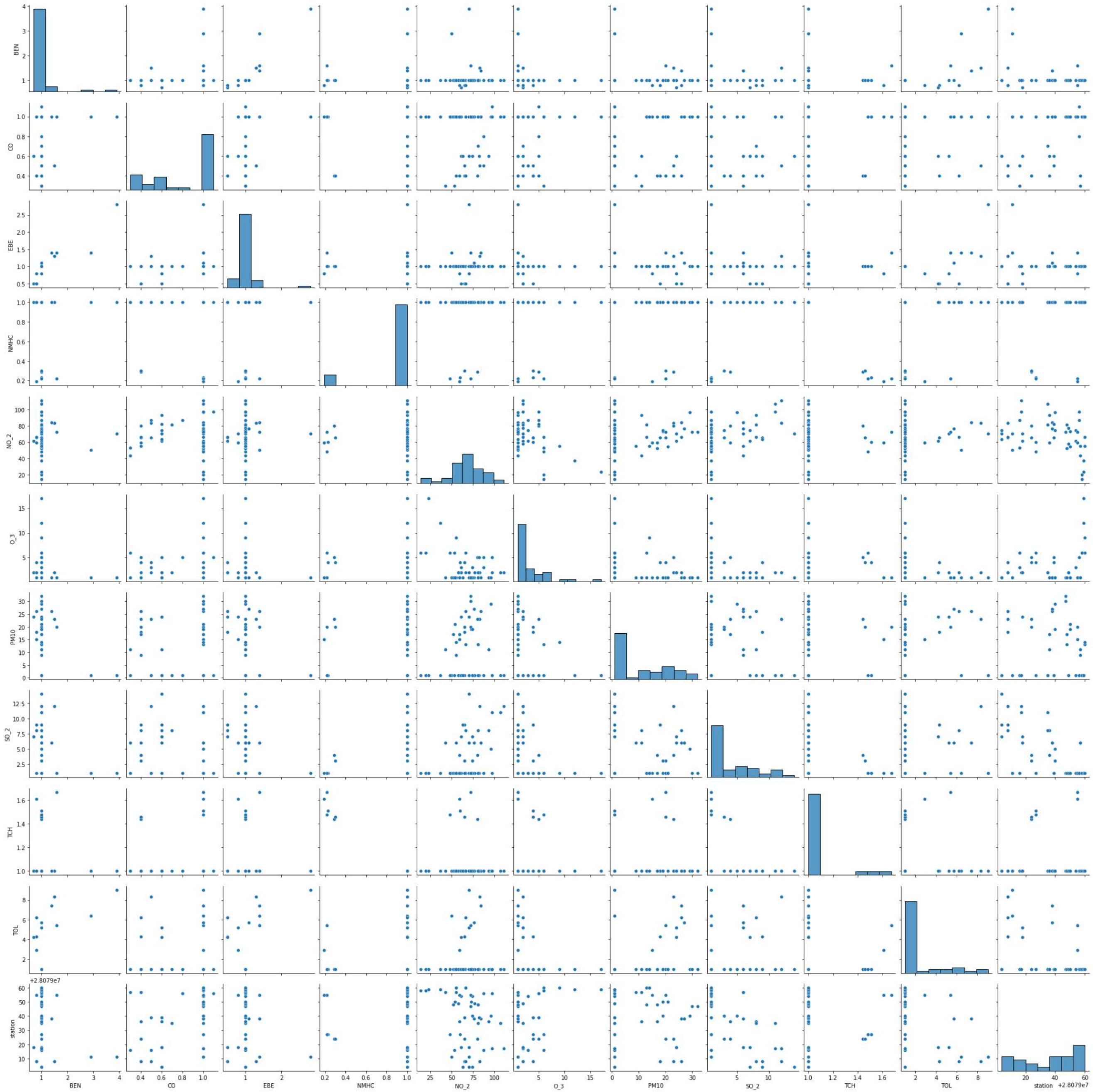
	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_
count	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.000000	209880.00000
mean	0.931014	0.721695	0.954744	0.900223	20.101401	34.586402	29.461235	9.636635	3.213098	2.41724
std	0.430684	0.361528	0.301074	0.267139	44.319112	27.866588	35.362880	13.492716	5.044685	3.09325
min	0.100000	0.100000	0.100000	0.040000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
25%	1.000000	0.300000	1.000000	1.000000	2.000000	14.000000	1.000000	1.000000	1.000000	1.00000
50%	1.000000	1.000000	1.000000	1.000000	5.000000	27.000000	8.000000	1.000000	1.000000	1.00000
75%	1.000000	1.000000	1.000000	1.000000	17.000000	48.000000	54.000000	14.000000	1.000000	3.00000
max	12.100000	10.400000	11.800000	1.000000	1081.000000	388.000000	226.000000	232.000000	63.000000	89.00000

In [18]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
        'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

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

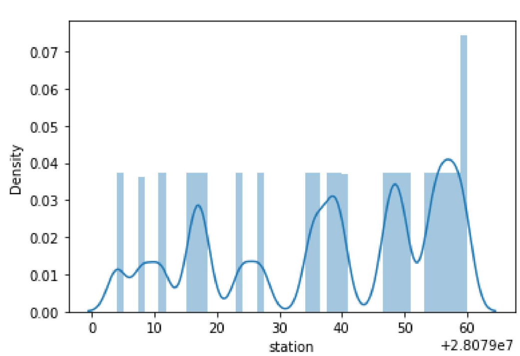
Out[19]: <seaborn.axisgrid.PairGrid at 0x20f06d4d2e0>



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

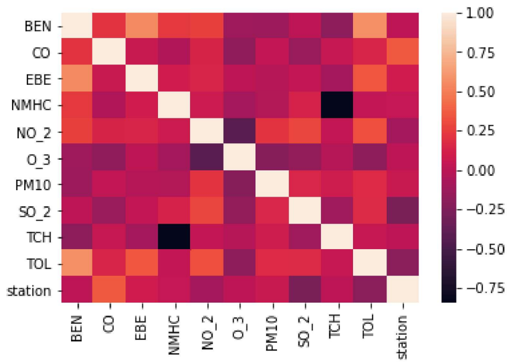
```
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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



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

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', '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()

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

Out[25]: 28078974.406198326

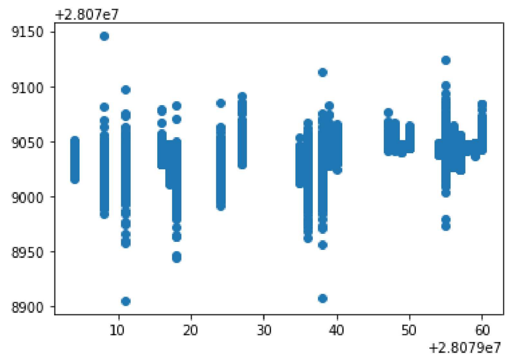
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	2.108241
CO	18.451959
EBE	9.960444
NMHC	18.763590
NO_2	-0.056328
O_3	0.009023
PM10	0.202416
SO_2	-0.925580
TCH	27.314417
TOL	-3.697850

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

Out[27]: <matplotlib.collections.PathCollection at 0x20f14ece700>



ACCURACY

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

Out[28]: 0.2967569182916857

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

Out[29]: 0.3010933192756732

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(alpha=10)

Accuracy(Ridge)

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

Out[32]: 0.29677442776901075

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

Out[33]: 0.30109007377580155

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

Out[34]: Lasso(alpha=10)

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

Out[35]: 0.04456572422360783

Accuracy(Lasso)

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

Out[36]: 0.044227173527790375

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

Out[37]: ElasticNet()

In [38]:
```
en.coef_
```
Out[38]:
```
array([ 0.40075863,  2.69805747,  0.51308103,  0.        , -0.02150688,
       -0.01761615,  0.15879854, -1.27015822, -0.        , -1.688371  ])
```

In [39]:
```
en.intercept_
```
Out[39]:
```
28079040.081229556
```

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

In [41]:
```
en.score(x_test,y_test)
```
Out[41]:
```
0.15292454711115522
```

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)))
```
```
13.759835789967182
263.2214202944504
16.22409998411161
```

Logistic Regression

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

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df['station']
```

In [45]:
```
feature_matrix.shape
```
Out[45]:
```
(209880, 10)
```

In [46]:
```
target_vector.shape
```
Out[46]:
```
(209880,)
```

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]]
```

In [51]:
```
prediction=logr.predict(observation)
print(prediction)
```
```
[28079008]
```

In [52]:
```
logr.classes_
```
Out[52]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [53]:
```
logr.score(fs,target_vector)
```
Out[53]:
```
0.6612921669525443
```

In [54]:
```
logr.predict_proba(observation)[0][0]
```
Out[54]:
```
9.49253547859177e-217
```

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

Out[55]:
```
array([[9.49253548e-217, 6.03969072e-001, 1.69773000e-169,
        1.44179094e-134, 1.71060740e-074, 3.96021369e-001,
        9.55808997e-006, 5.22717178e-089, 5.48319507e-081,
        1.32436170e-079, 1.07294134e-076, 3.50636612e-129,
        1.69529056e-079, 3.82520459e-158, 4.22872970e-161,
        3.57928159e-187, 2.10845766e-164, 8.33937392e-188,
        1.12752042e-082, 7.42692411e-129, 7.66872499e-080,
        6.30044443e-191, 4.32093567e-191, 3.26054498e-071]])
```

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]:
```
0.692443300933867
```

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

In [63]:
```
ture_names=x.columns,class_names=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x'
```

Out[63]:
```
[Text(2258.571428571429, 1993.2, 'O_3 <= 1.5\ngini = 0.958\nsamples = 92901\nvalue = [6161, 5810, 6152, 5922, 6088, 6175, 626
6, 6282, 6141\n5911, 6071, 6180, 6094, 6239, 6149, 6271, 6136, 6174\n6223, 6009, 6147, 6204, 5977, 6134]\nclass = h'),
 Text(1328.5714285714287, 1630.8000000000002, 'TCH <= 1.27\ngini = 0.904\nsamples = 39495\nvalue = [6161, 36, 6152, 10, 54, 1
50, 23, 6, 200, 5911\n6071, 46, 6094, 6239, 6149, 387, 6136, 31, 6223, 62\n6147, 100, 35, 76]\nclass = n'),
 Text(690.8571428571429, 1268.4, 'CO <= 0.95\ngini = 0.894\nsamples = 35689\nvalue = [6161, 36, 6152, 10, 54, 150, 17, 6, 20
0, 5911\n6071, 46, 6094, 6239, 6149, 387, 6136, 31, 75, 62\n6147, 100, 35, 76]\nclass = n'),
 Text(425.14285714285717, 906.0, 'NO_2 <= 23.5\ngini = 0.679\nsamples = 11417\nvalue = [5905, 4, 0, 8, 0, 127, 6, 0, 142, 572
1, 0, 32\n0, 0, 0, 0, 0, 0, 0, 15, 6092, 0, 0, 0]\nclass = u'),
 Text(212.57142857142858, 543.5999999999999, 'PM10 <= 1.5\ngini = 0.639\nsamples = 4612\nvalue = [1446, 0, 0, 2, 0, 3, 6, 0,
0, 2692, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 3100, 0, 0, 0]\nclass = u'),
 Text(106.28571428571429, 181.19999999999982, 'gini = 0.031\nsamples = 941\nvalue = [1446, 0, 0, 2, 0, 0, 0, 0, 0, 7, 0, 0,
0, 0\n0, 0, 0, 0, 0, 0, 14, 0, 0, 0]\nclass = a'),
 Text(318.8571428571429, 181.19999999999982, 'gini = 0.499\nsamples = 3671\nvalue = [0, 0, 0, 0, 0, 3, 6, 0, 0, 2685, 0, 0,
0, 0\n0, 0, 0, 0, 0, 0, 3086, 0, 0, 0]\nclass = u'),
 Text(637.7142857142858, 543.5999999999999, 'TOL <= 1.25\ngini = 0.674\nsamples = 6805\nvalue = [4459, 4, 0, 6, 0, 124, 0, 0,
142, 3029, 0, 32\n0, 0, 0, 0, 0, 0, 0, 15, 2992, 0, 0, 0]\nclass = a'),
 Text(531.4285714285714, 181.19999999999982, 'gini = 0.667\nsamples = 6737\nvalue = [4459, 2, 0, 6, 0, 13, 0, 0, 142, 3029,
0, 32\n0, 0, 0, 0, 0, 0, 0, 15, 2992, 0, 0, 0]\nclass = a'),
 Text(744.0, 181.19999999999982, 'gini = 0.035\nsamples = 68\nvalue = [0, 2, 0, 0, 0, 111, 0, 0, 0, 0, 0, 0, 0\n0, 0,
```

Conclusion

Accuracy

In [64]:
```
lr.score(x_train,y_train)
```

Out[64]:
```
0.3010933192756732
```

In [65]:
```
rr.score(x_train,y_train)
```
Out[65]: 0.30109007377580155

In [66]:
```
la.score(x_train,y_train)
```
Out[66]: 0.044565724223607834

In [67]:
```
en.score(x_test,y_test)
```
Out[67]: 0.15292454711115522

In [68]:
```
logr.score(fs,target_vector)
```
Out[68]: 0.6612921669525443

In [69]:
```
grid_search.best_score_
```
Out[69]: 0.692443300933867

Random Forest is suitable for this dataset

In [ ]: