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("2001.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2001-08-01 01:00:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	1(
1	2001-08-01 01:00:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	1(
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	1(
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	(
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	7
...	...	...	...	...	...	...	...	...	...	...	
217867	2001-04-01 00:00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	4
217868	2001-04-01 00:00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	2
217869	2001-04-01 00:00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	4
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	3
217871	2001-04-01 00:00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	3

217872 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: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

In [6]:

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

Out[6]:

	CO	station
1	0.34	28079035
5	0.63	28079006
21	0.43	28079024
23	0.34	28079099
25	0.06	28079035
...	...	...
217829	4.48	28079006
217847	2.65	28079099
217849	1.22	28079035
217853	1.83	28079006
217871	1.62	28079099

29669 rows × 2 columns

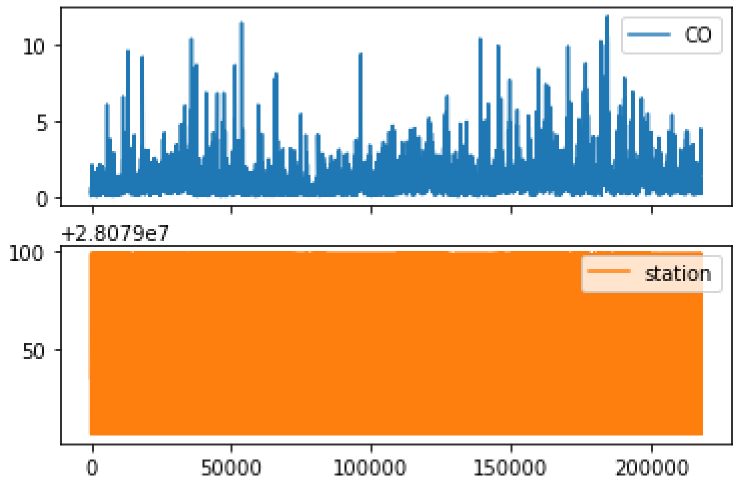
Line chart

In [7]:

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

Out[7]:

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



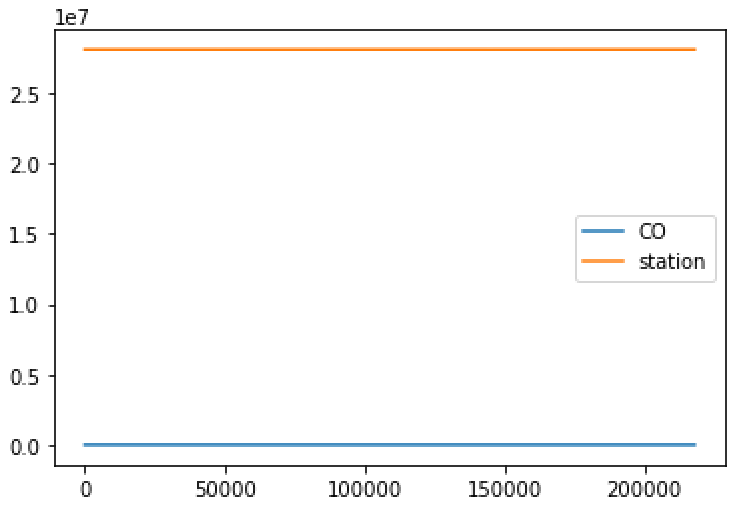
Line chart

In [8]:

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

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

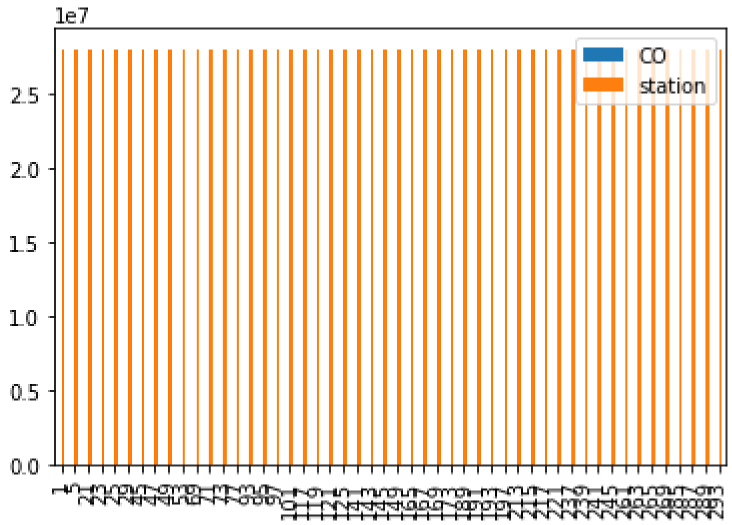
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<AxesSubplot:>



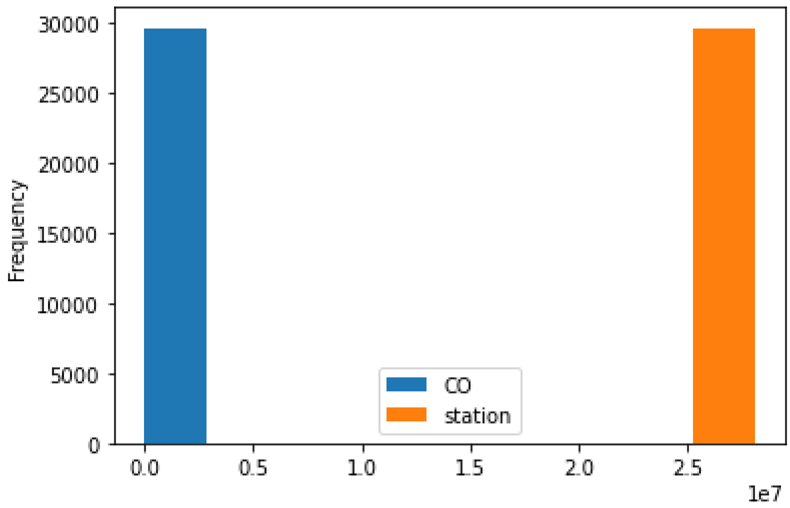
Histogram

In [11]:

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

Out[11]:

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



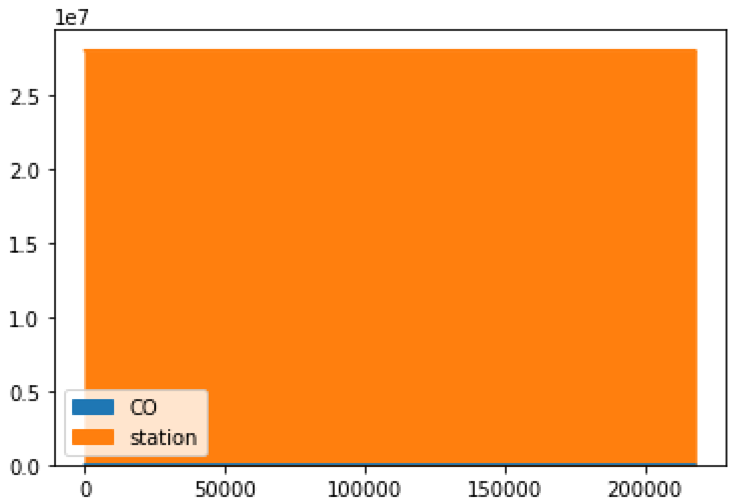
Area chart

In [12]:

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

Out[12]:

```
<AxesSubplot:>
```



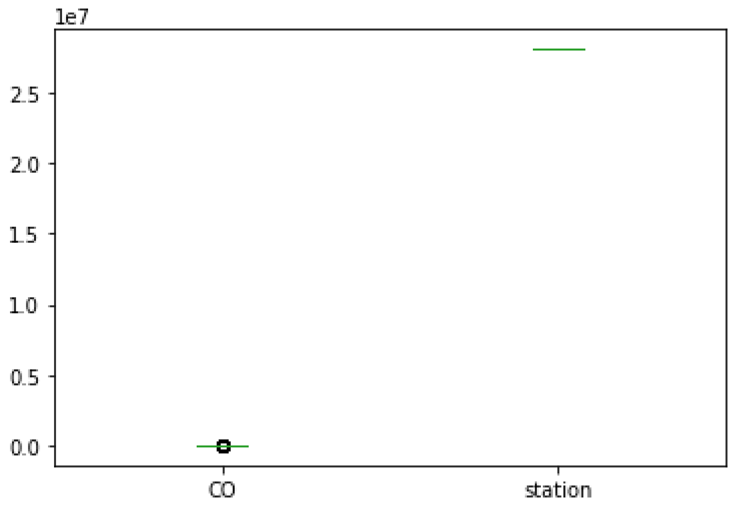
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



Pie chart

In [14]:

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

Out[14]:

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

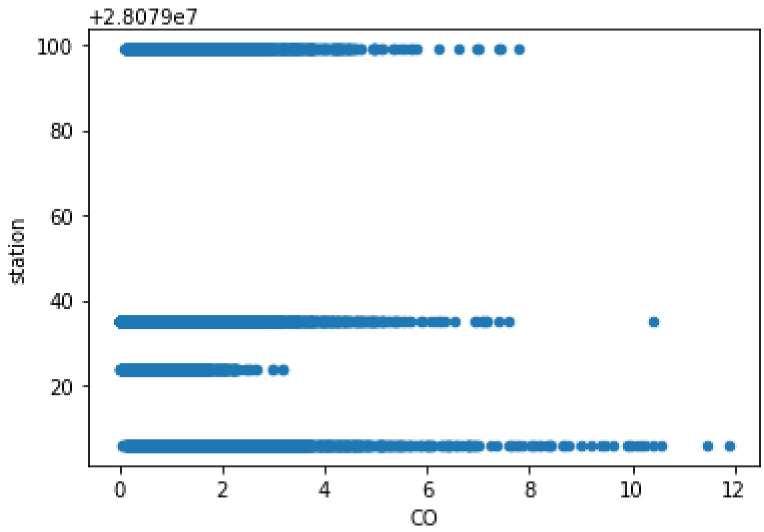
Scatter chart

In [15]:

```
data.plot.scatter(x='CO' ,y='station')
```

Out[15]:

```
<AxesSubplot:xlabel='CO', ylabel='station'>
```



In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000
mean	3.361895	1.005413	3.580229	8.113086	0.195222	67.652292
std	3.176669	0.863135	3.744496	7.909701	0.192585	34.003120
min	0.100000	0.000000	0.140000	0.210000	0.000000	1.180000
25%	1.280000	0.470000	1.390000	3.040000	0.080000	44.299999
50%	2.510000	0.760000	2.600000	5.830000	0.140000	64.449997
75%	4.420000	1.270000	4.580000	10.640000	0.250000	86.540001
max	54.560001	11.890000	77.260002	150.600006	2.880000	292.700012

In [18]:

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

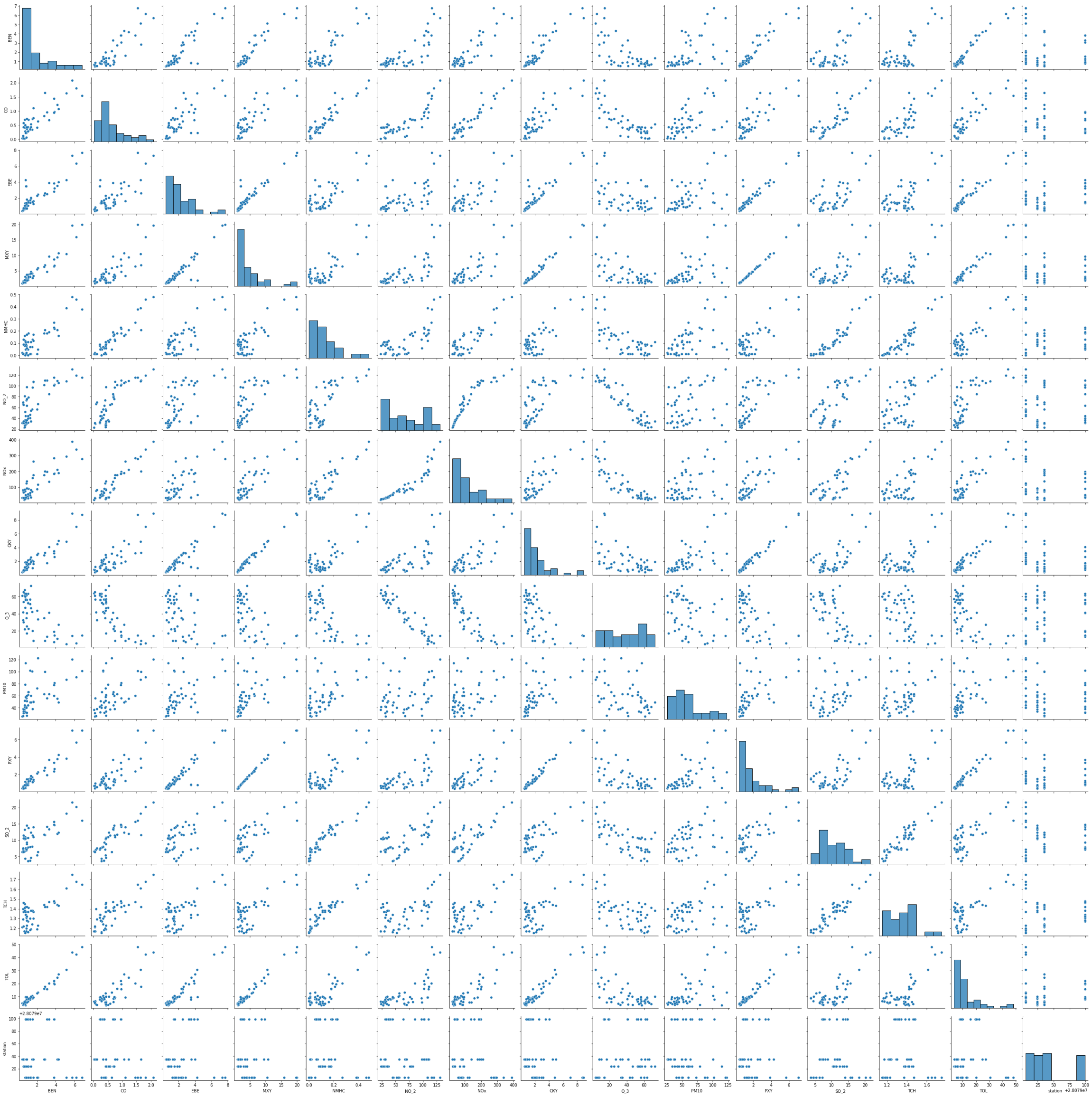
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x1f5a86cfdf0>
```



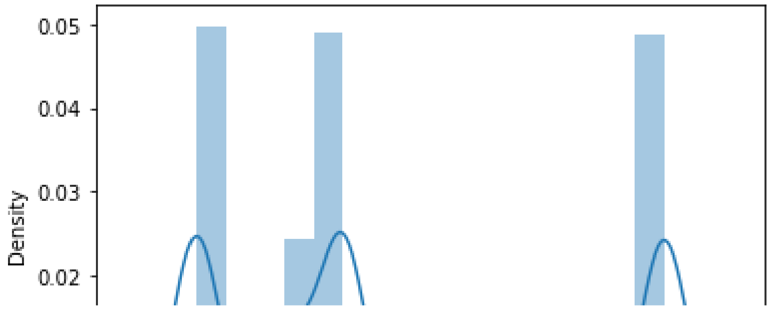
In [20]:

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

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:255
7: FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-l
evel function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]:

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

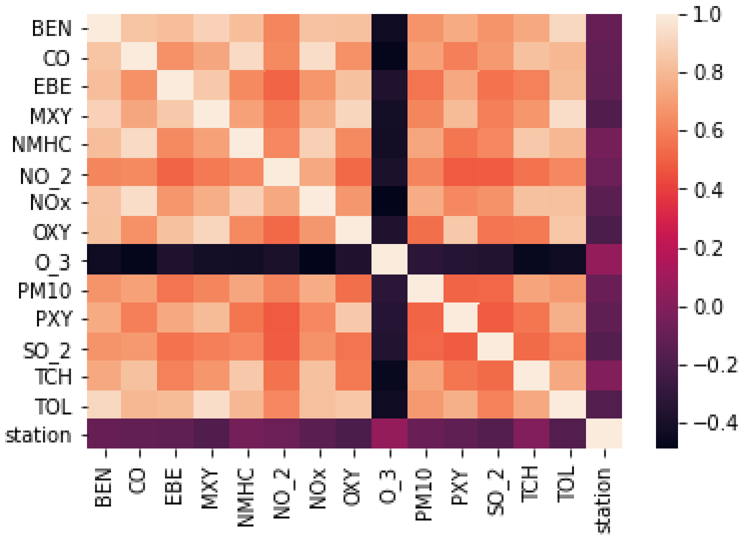


In [21]:

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

Out[21]:

```
<AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:

```
x=df[['BEN', 'CO', 'EBE', '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()
```

In [25]:

```
lr.intercept_
```

Out[25]:

```
28079008.55913765
```

In [26]:

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

Out[26]:

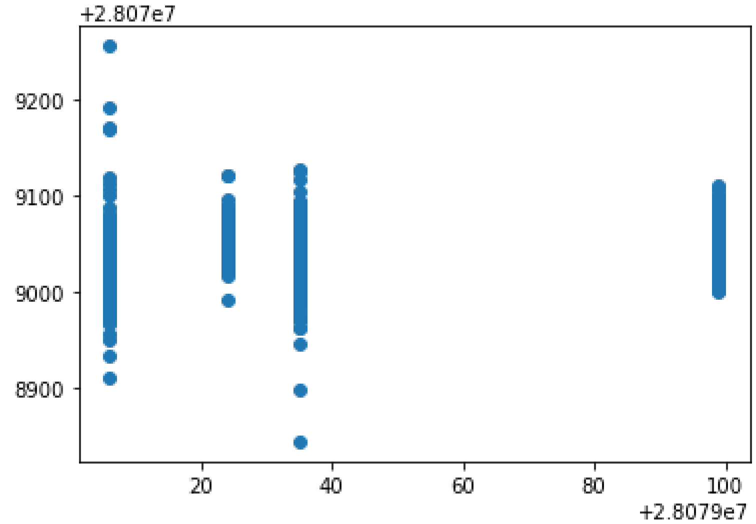
	Co-efficient
BEN	6.619859
CO	-15.989876
EBE	0.872545
MXY	0.017424
NMHC	88.710184
NO_2	0.112348
NOx	-0.080611
OXY	-3.244159
O_3	-0.025626
PM10	-0.070470
PXY	1.239961
SO_2	-0.312370
TCH	35.962288
TOL	-1.189127

In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x1f5b7e58be0>



ACCURACY

In [28]:

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

Out[28]:

0.16286149554552198

In [29]:

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

Out[29]:

0.16520772737246636

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]:

```
0.16372670617299712
```

In [33]:

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

Out[33]:

```
0.16489635135341363
```

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]:

```
0.038993466925703824
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.04002946671090035
```

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([ 4.7444565 ,  0.        ,  0.82948713, -0.23953317,  0.07631236,
        0.06701353, -0.02985224, -2.57310218, -0.03186297,  0.06550741,
        0.76329119, -0.32984626,  1.21356543, -0.6647353 ])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079048.852104515
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.10236914940351627
```

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)))
```

```
30.61342063435188
1229.2000073743877
35.05994876457163
```

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]:

```
(29669, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(29669,)
```

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)
```

```
[28079035]
```

In [52]:

```
logr.classes_
```

Out[52]:

```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [53]:

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

Out[53]:

```
0.8087229094340894
```

In [54]:

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

Out[54]:

```
1.7245277771444498e-43
```

In [55]:

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

Out[55]:

```
array([[1.72452778e-43, 2.43756289e-56, 9.99998565e-01, 1.43537418e-06]])
```

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]:

```
0.7372399845916795
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

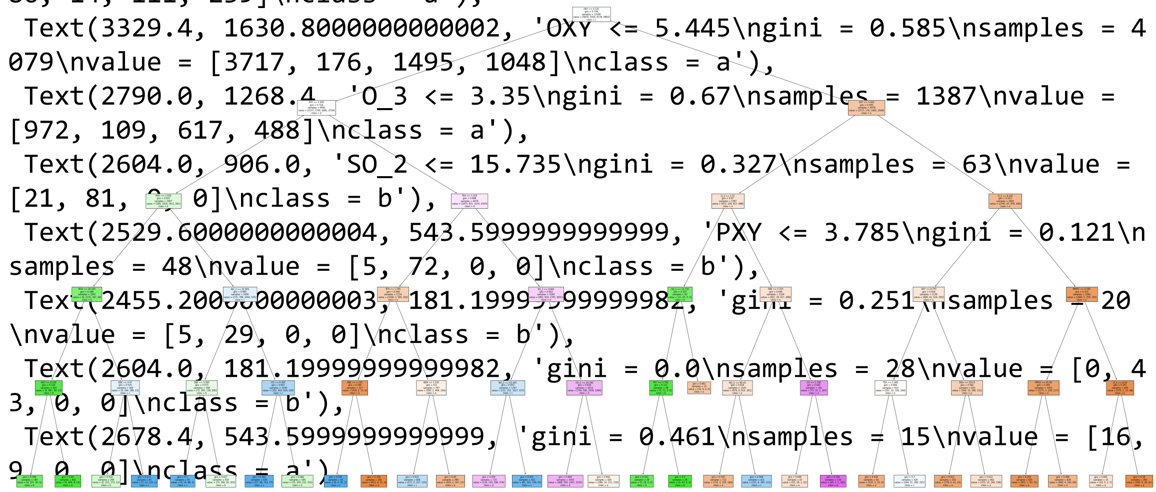
```
from sklearn.tree import plot_tree

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

Out[62]:

```
[Text(2259.9, 1993.2, 'OXY <= 4.125\ngini = 0.734\nsamples = 13160\nvalue
= [5874, 2916, 6176, 5802]\nclass = c'),
 Text(1190.4, 1630.8000000000002, 'PXY <= 1.005\ngini = 0.724\nsamples = 9
081\nvalue = [2157, 2740, 4681, 4754]\nclass = d'),
 Text(595.2, 1268.4, 'MXY <= 1.035\ngini = 0.637\nsamples = 2607\nvalue =
[184, 1919, 1411, 561]\nclass = b'),
 Text(297.6, 906.0, 'NOx <= 26.245\ngini = 0.388\nsamples = 1001\nvalue =
[9, 1131, 347, 26]\nclass = b'),
 Text(148.8, 543.5999999999999, 'OXY <= 0.535\ngini = 0.139\nsamples = 641
\nvalue = [0, 887, 58, 13]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.296\nsamples = 199\nvalue = [0,
227, 50, 0]\nclass = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.06\nsamples = 442
\nvalue = [0, 660, 8, 13]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'EBE <= 0.97\ngini = 0.535\ns
amples = 360\nvalue = [9, 244, 289, 13]\nclass = c'),
 Text(372.0, 181.19999999999982, 'gini = 0.522\nsamples = 269\nvalue = [2,
231, 173, 12]\nclass = b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.271\nsamples = 91\n
value = [7, 13, 116, 1]\nclass = c'),
 Text(892.8000000000001, 906.0, 'NO_2 <= 32.905\ngini = 0.685\nsamples = 1
606\nvalue = [175, 788, 1064, 535]\nclass = c'),
 Text(744.0, 543.5999999999999, 'EBE <= 0.585\ngini = 0.673\nsamples = 568
\nvalue = [73, 385, 138, 306]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.3\nsamples = 65\nvalue = [0, 1
6, 88, 3]\nclass = c'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.627\nsamples = 503
\nvalue = [73, 369, 50, 303]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'CO <= 0.395\ngini = 0.607\ns
amples = 1038\nvalue = [102, 403, 926, 229]\nclass = c'),
 Text(967.2, 181.19999999999982, 'gini = 0.302\nsamples = 530\nvalue = [2
7, 44, 714, 77]\nclass = c'),
 Text(1116.0, 181.19999999999982, 'gini = 0.682\nsamples = 508\nvalue = [7
5, 359, 212, 152]\nclass = b'),
 Text(1785.6000000000001, 1268.4, 'TCH <= 1.255\ngini = 0.688\nsamples = 6
474\nvalue = [1973, 821, 3270, 4193]\nclass = d'),
 Text(1488.0, 906.0, 'TCH <= 1.195\ngini = 0.549\nsamples = 1114\nvalue =
[1008, 2, 565, 162]\nclass = a'),
 Text(1339.2, 543.5999999999999, 'EBE <= 1.235\ngini = 0.266\nsamples = 31
7\nvalue = [418, 0, 71, 6]\nclass = a'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.245\nsamples = 34
\nvalue = [6, 0, 50, 2]\nclass = c'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.109\nsamples = 283
\nvalue = [412, 0, 21, 4]\nclass = a'),
 Text(1636.8000000000002, 543.5999999999999, 'BEN <= 1.535\ngini = 0.6\nsa
mples = 797\nvalue = [590, 2, 494, 156]\nclass = a'),
 Text(1562.4, 181.19999999999982, 'gini = 0.615\nsamples = 408\nvalue = [1
71, 0, 327, 135]\nclass = c'),
 Text(1711.2, 181.19999999999982, 'gini = 0.45\nsamples = 389\nvalue = [41
9, 2, 167, 21]\nclass = a'),
 Text(2083.2000000000003, 906.0, 'SO_2 <= 9.845\ngini = 0.653\nsamples = 5
360\nvalue = [965, 819, 2705, 4031]\nclass = d'),
 Text(1934.4, 543.5999999999999, 'NO_2 <= 62.445\ngini = 0.653\nsamples =
1347\nvalue = [211, 251, 1027, 637]\nclass = c'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.654\nsamples = 725
\nvalue = [130, 150, 288, 574]\nclass = d'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.415\nsamples = 622
\nvalue = [81, 101, 739, 63]\nclass = c'),
 Text(2232.0, 543.5999999999999, 'SO_2 <= 29.185\ngini = 0.628\nsamples =
4013\nvalue = [754, 568, 1678, 3394]\nclass = d'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.608\nsamples = 359
```

```
3\nvalue = [468, 554, 1567, 3155]\nclass = d'),
 Text(2306.4, 181.19999999999982, 'gini = 0.642\nsamples = 420\nvalue = [2
86, 14, 111, 239]\nclass = a'),
 Text(3329.4, 1630.8000000000002, 'OXY <= 5.445\ngini = 0.585\nsamples = 4
079\nvalue = [3717, 176, 1495, 1048]\nclass = a'),
 Text(2790.0, 1268.4, 'O_3 <= 3.35\ngini = 0.67\nsamples = 1387\nvalue =
[972, 109, 617, 488]\nclass = a'),
 Text(2604.0, 906.0, 'SO_2 <= 15.735\ngini = 0.327\nsamples = 63\nvalue =
[21, 81, 0, 0]\nclass = b'),
 Text(2529.6000000000004, 543.5999999999999, 'PXY <= 3.785\ngini = 0.121\n
samples = 48\nvalue = [5, 72, 0, 0]\nclass = b'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.251\nsamples = 20
\nvalue = [5, 29, 0, 0]\nclass = b'),
 Text(2604.0, 181.19999999999982, 'gini = 0.0\nsamples = 28\nvalue = [0, 4
3, 0, 0]\nclass = b'),
 Text(2678.4, 543.5999999999999, 'gini = 0.461\nsamples = 15\nvalue = [16,
9, 0, 0]\nclass = a'),
 Text(2976.0, 906.0, 'EBE <= 5.315\ngini = 0.649\nsamples = 1324\nvalue =
[951, 28, 617, 488]\nclass = a'),
 Text(2827.2000000000003, 543.5999999999999, 'NO_2 <= 86.43\ngini = 0.618
\nsamples = 1143\nvalue = [876, 8, 615, 301]\nclass = a'),
 Text(2752.8, 181.19999999999982, 'gini = 0.526\nsamples = 722\nvalue = [7
25, 2, 220, 186]\nclass = a'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.568\nsamples = 421
\nvalue = [151, 6, 395, 115]\nclass = c'),
 Text(3124.8, 543.5999999999999, 'CO <= 1.185\ngini = 0.492\nsamples = 181
\nvalue = [75, 20, 2, 187]\nclass = d'),
 Text(3050.4, 181.19999999999982, 'gini = 0.648\nsamples = 47\nvalue = [3
5, 18, 1, 22]\nclass = a'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.334\nsamples = 134
\nvalue = [40, 2, 1, 165]\nclass = d'),
 Text(3868.8, 1268.4, 'O_3 <= 8.245\ngini = 0.523\nsamples = 2692\nvalue =
[2745, 67, 878, 560]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'OXY <= 8.755\ngini = 0.654\nsamples = 12
30\nvalue = [899, 62, 619, 395]\nclass = a'),
 Text(3422.4, 543.5999999999999, 'TCH <= 1.485\ngini = 0.694\nsamples = 61
4\nvalue = [351, 52, 329, 264]\nclass = a'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.439\nsamples = 94
\nvalue = [107, 1, 33, 9]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.7\nsamples = 520\nvalue = [24
4, 51, 296, 255]\nclass = c'),
 Text(3720.0000000000005, 543.5999999999999, 'NOx <= 353.8\ngini = 0.581\n
samples = 616\nvalue = [548, 10, 290, 131]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.421\nsamples = 154
\nvalue = [178, 0, 41, 23]\nclass = a'),
 Text(3794.4, 181.19999999999982, 'gini = 0.612\nsamples = 462\nvalue = [3
70, 10, 249, 108]\nclass = a'),
 Text(4166.400000000001, 906.0, 'NMHC <= 0.365\ngini = 0.323\nsamples = 14
62\nvalue = [1846, 5, 259, 165]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'PM10 <= 45.49\ngini = 0.365
\nsamples = 1053\nvalue = [1271, 5, 236, 117]\nclass = a'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.263\nsamples = 625
\nvalue = [831, 1, 93, 51]\nclass = a'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.488\nsamples = 428
\nvalue = [440, 4, 143, 66]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'O_3 <= 8.39\ngini = 0.201\nsa
mples = 409\nvalue = [575, 0, 23, 48]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.605\nsamples = 15\nvalue = [1
0, 0, 3, 7]\nclass = a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.18\nsamples = 394\nvalue = [56
5, 0, 20, 41]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.16520772737246636

Ridge Regression:0.16489635135341363

Lasso Regression:0.04002946671090035

ElasticNet Regression:0.10236914940351627

Logistic Regression:0.8087229094340894

Random Forest:0.7372399845916795

From the above data, we can conclude that logistic regression is preferrable to other regression types

In [ ]:

In [ ]: