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("2002.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.99(
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.85	20.98(
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.44(
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.18(
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.33(
...	...	...	...	...	...	...	...	...	...	...	
217291	2002-11-01 00:00:00	4.16	1.14	NaN	NaN	NaN	81.080002	265.700012	NaN	7.21	36.75(
217292	2002-11-01 00:00:00	3.67	1.73	2.89	NaN	0.38	113.900002	373.100006	NaN	5.66	63.38(
217293	2002-11-01 00:00:00	1.37	0.58	1.17	2.37	0.15	65.389999	107.699997	1.30	9.11	9.64(
217294	2002-11-01 00:00:00	4.51	0.91	4.83	10.99	NaN	149.800003	202.199997	1.00	5.75	
217295	2002-11-01 00:00:00	3.11	1.17	3.00	7.77	0.26	80.110001	180.300003	2.25	7.38	29.24(

217296 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: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

In [6]:

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

Out[6]:

	CO	station
1	0.71	28079035
5	0.72	28079006
22	0.80	28079024
24	1.04	28079099
26	0.53	28079035
...	...	...
217269	0.28	28079024
217271	1.30	28079099
217273	0.97	28079035
217293	0.58	28079024
217295	1.17	28079099

32381 rows × 2 columns

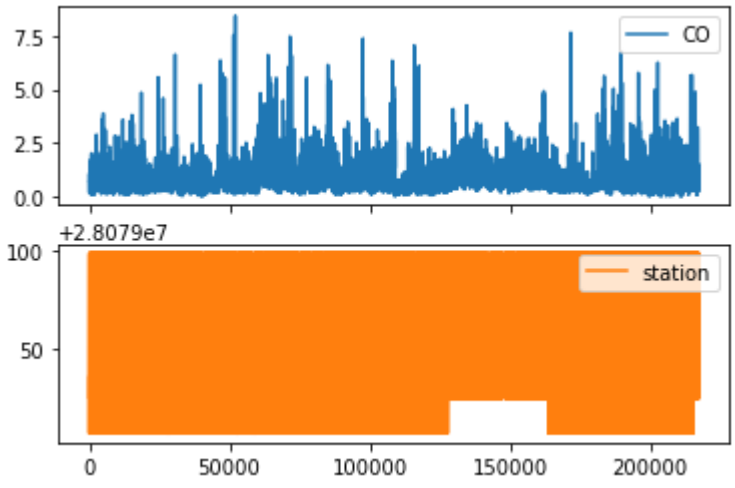
Line chart

In [7]:

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

Out[7]:

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



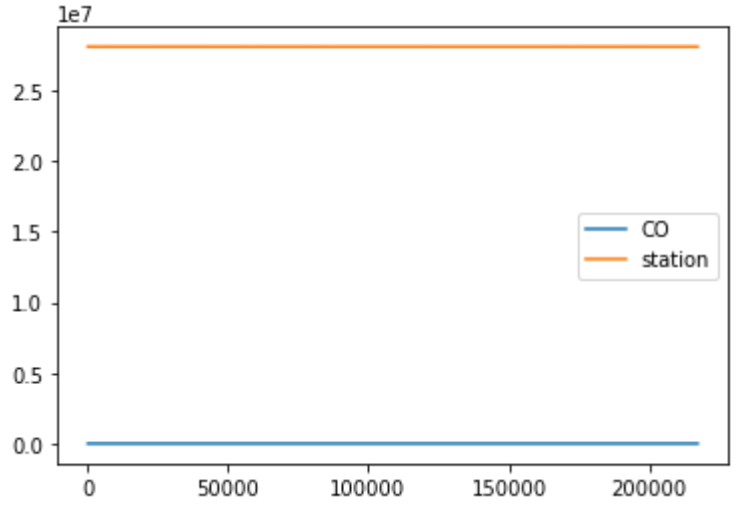
Line chart

In [8]:

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

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

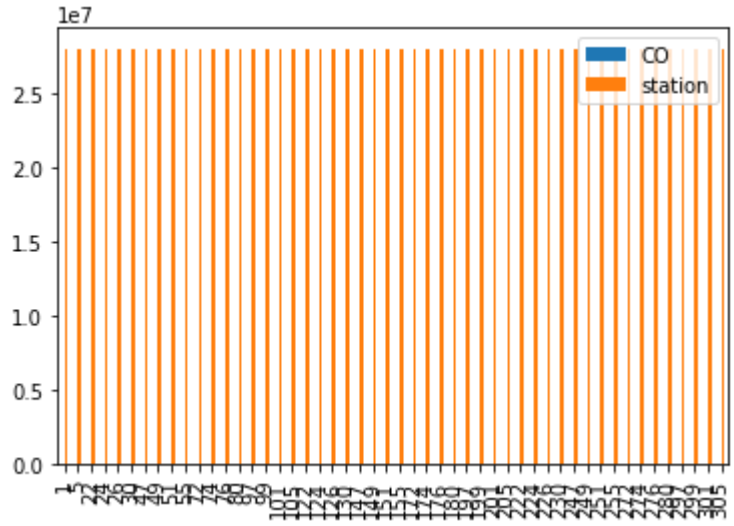
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<AxesSubplot:>



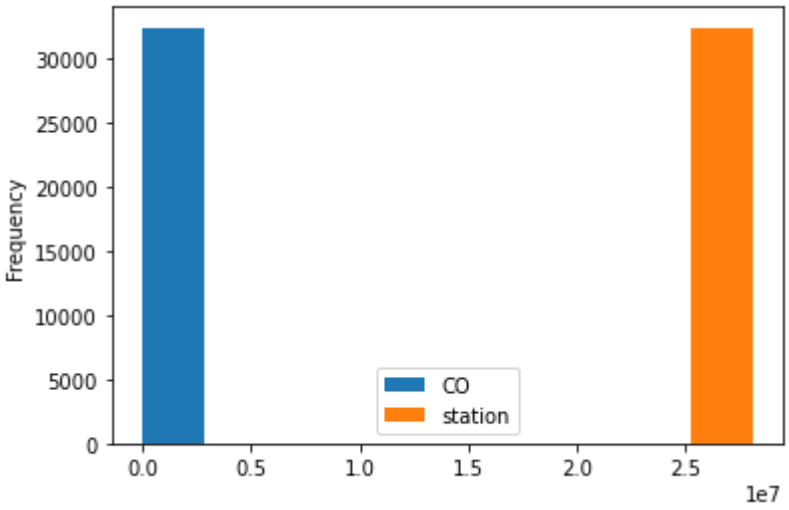
Histogram

In [11]:

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

Out[11]:

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



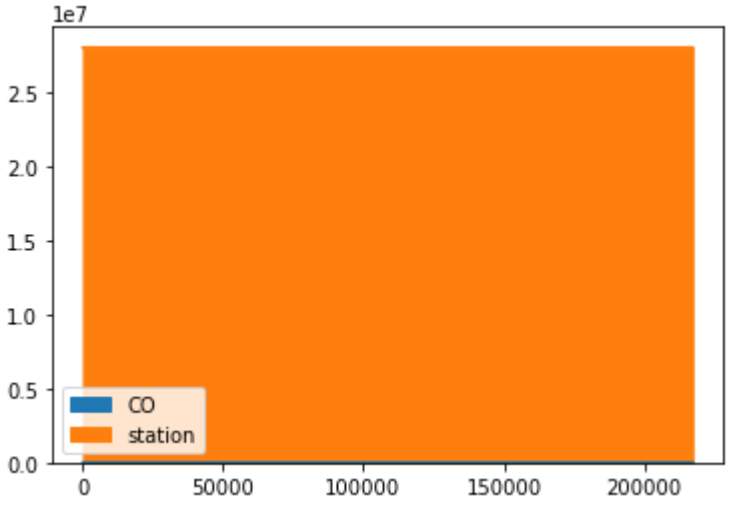
Area chart

In [12]:

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

Out[12]:

```
<AxesSubplot:>
```



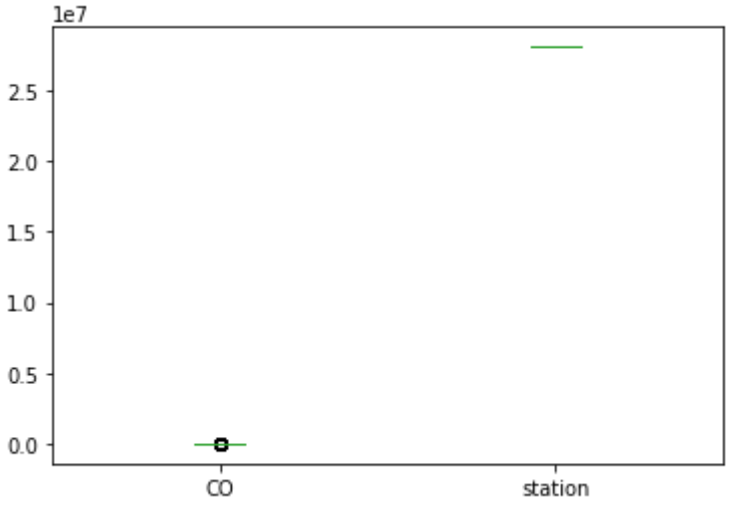
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



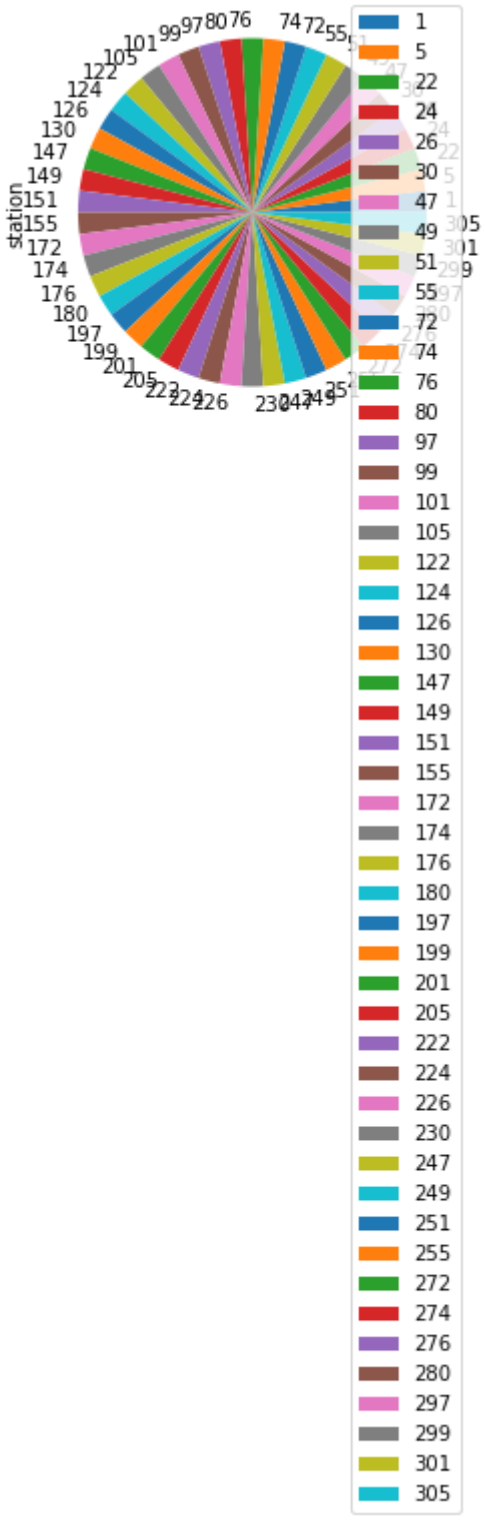
Pie chart

In [14]:

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

Out[14]:

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



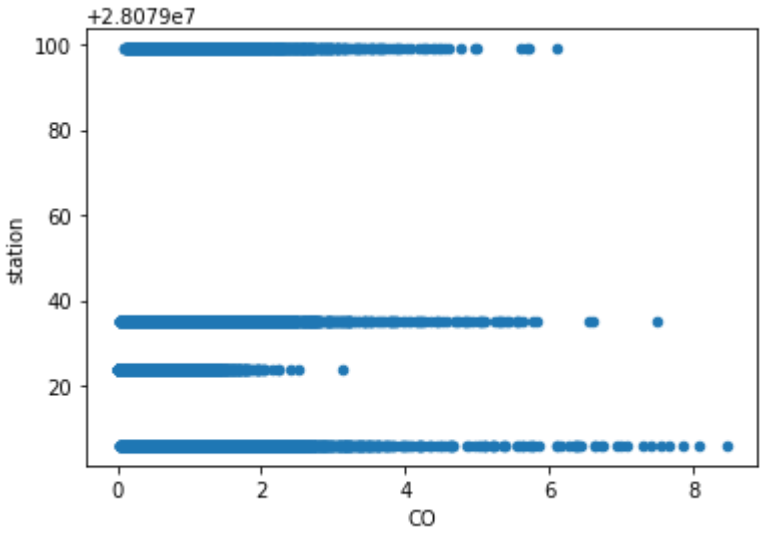
Scatter chart

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: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     32381 non-null  object
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000
mean	2.479155	0.787323	2.914004	7.013636	0.155827	58.936796
std	2.280959	0.610810	2.667881	6.774365	0.135731	31.472733
min	0.180000	0.000000	0.180000	0.190000	0.000000	0.890000
25%	0.970000	0.420000	1.140000	2.420000	0.080000	35.660000
50%	1.840000	0.620000	2.130000	5.140000	0.130000	57.160000
75%	3.250000	0.980000	3.830000	9.420000	0.200000	78.769997
max	32.660000	8.460000	41.740002	99.879997	2.700000	263.600006

In [18]:

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

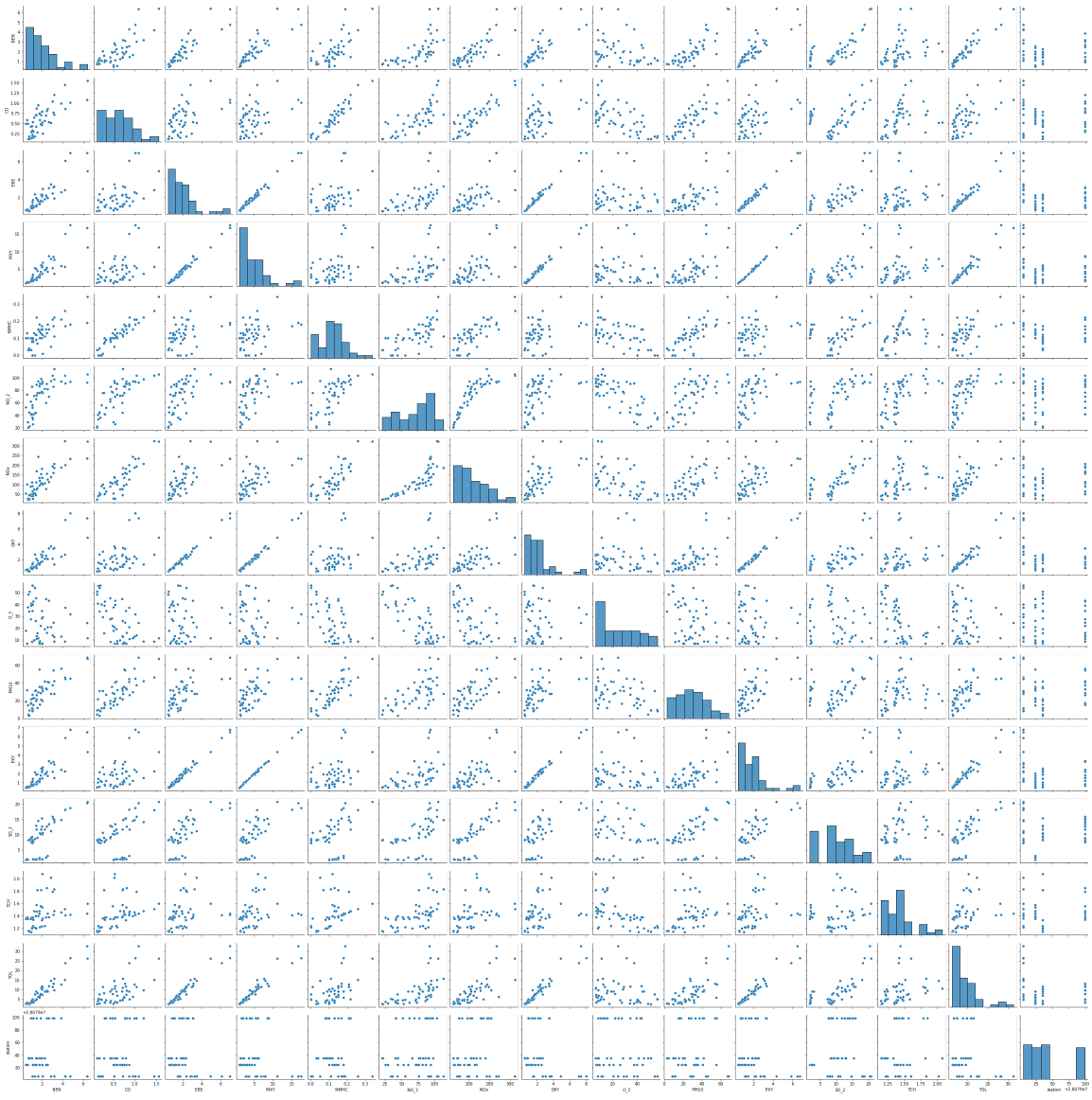
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x1368f903a90>
```



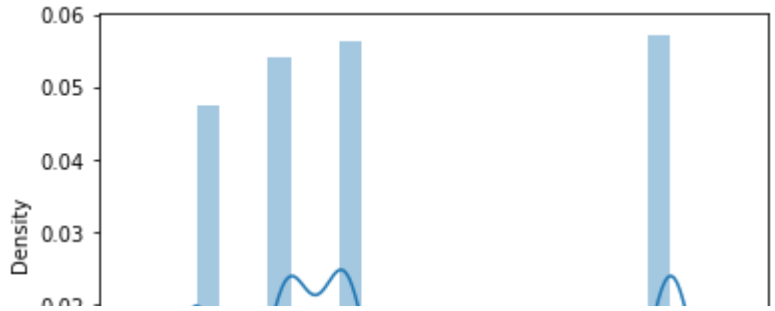
In [20]:

```
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'>
```

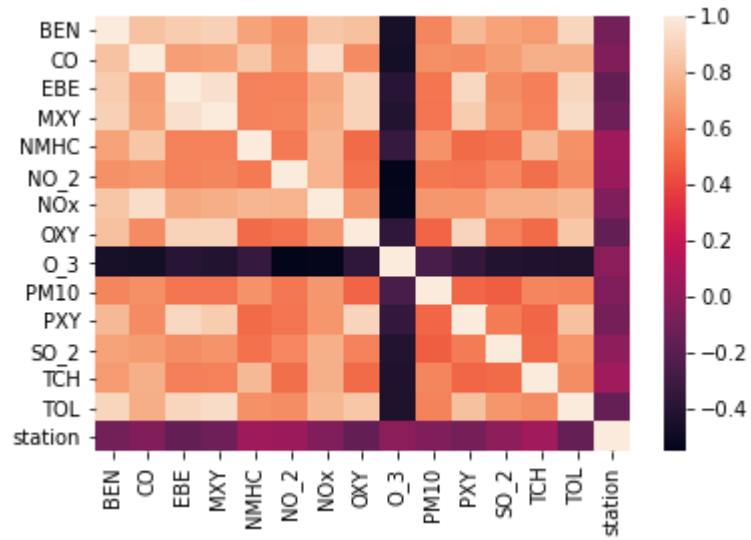


In [21]:

```
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]:

```
28078990.622461464
```

In [26]:

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

Out[26]:

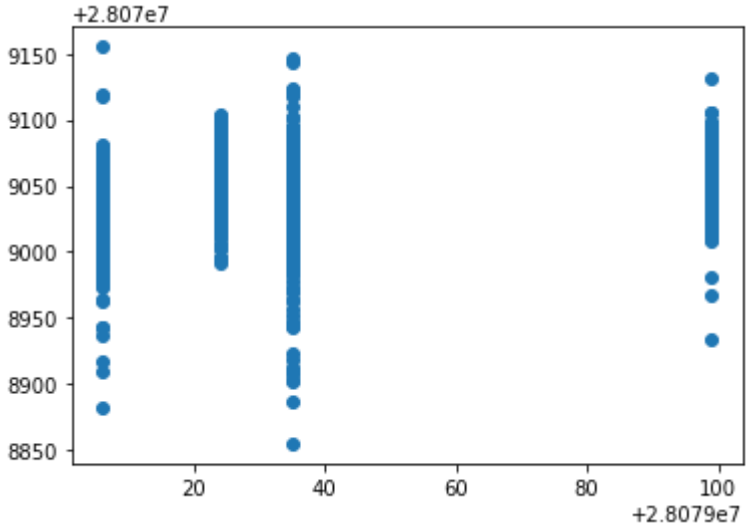
	Co-efficient
BEN	2.250573
CO	-13.356136
EBE	-11.032766
MXY	4.037415
NMHC	82.950541
NO_2	0.250248
NOx	-0.101087
OXY	-5.331560
O_3	-0.036990
PM10	-0.125154
PXY	7.582413
SO_2	0.620478
TCH	41.563998
TOL	-1.488182

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x136a00cad30>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.1994901005045402
```

In [29]:

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

Out[29]:

```
0.19844468408144533
```

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]:

```
0.19877953137155935
```

In [33]:

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

Out[33]:

```
0.19824348970169514
```

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]:

```
0.05821399342354372
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.058987010961436104
```

Elastic Net

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([ 1.04329139e+00,  0.00000000e+00, -2.77495813e+00,  1.58080172e+00,
        2.24925739e-01,  2.24065730e-01, -2.91614786e-02, -2.48915117e+00,
       -3.22825319e-02,  1.68616249e-03,  2.19676136e+00,  4.15511513e-01,
        1.12433805e+00, -1.19291369e+00])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079038.842092328
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.09846653794618532
```

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)))
```

```
28.45980093897777
1117.586783105173
33.4303272958129
```

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]:

```
(32381, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(32381,)
```

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]:

```
0.8480899292795158
```

In [54]:

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

Out[54]:

```
2.5638972732451705e-10
```

In [55]:

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

Out[55]:

```
array([[2.56389727e-10, 3.44199742e-71, 1.00000000e+00, 1.43898646e-13]])
```

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]:

```
0.7732727433159798
```

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(2232.0, 1993.2, 'MXY <= 1.725\ngini = 0.749\nsamples = 14309\nvalue
= [5056, 5682, 5910, 6018]\nclass = d'),
 Text(1116.0, 1630.8000000000002, 'CO <= 0.315\ngini = 0.342\nsamples = 25
97\nvalue = [29, 3292, 478, 321]\nclass = b'),
 Text(558.0, 1268.4, 'NMHC <= 0.045\ngini = 0.612\nsamples = 784\nvalue =
[22, 653, 354, 211]\nclass = b'),
 Text(279.0, 906.0, 'OXY <= 0.815\ngini = 0.399\nsamples = 275\nvalue = [2
2, 27, 325, 53]\nclass = c'),
 Text(139.5, 543.5999999999999, 'NOx <= 44.83\ngini = 0.188\nsamples = 204
\nvalue = [12, 12, 286, 8]\nclass = c'),
 Text(69.75, 181.19999999999982, 'gini = 0.122\nsamples = 179\nvalue = [0
10, 264, 8]\nclass = c'),
 Text(209.25, 181.19999999999982, 'gini = 0.512\nsamples = 25\nvalue = [1
2, 2, 22, 0]\nclass = c'),
 Text(418.5, 543.5999999999999, 'MXY <= 1.13\ngini = 0.674\nsamples = 71\n
value = [10, 15, 39, 45]\nclass = d'),
 Text(348.75, 181.19999999999982, 'gini = 0.635\nsamples = 25\nvalue = [0
15, 18, 8]\nclass = c'),
 Text(488.25, 181.19999999999982, 'gini = 0.587\nsamples = 46\nvalue = [1
0, 0, 21, 37]\nclass = d'),
 Text(837.0, 906.0, 'SO_2 <= 4.735\ngini = 0.368\nsamples = 509\nvalue =
[0, 626, 29, 158]\nclass = b'),
 Text(697.5, 543.5999999999999, 'SO_2 <= 4.275\ngini = 0.014\nsamples = 35
7\nvalue = [0, 579, 0, 4]\nclass = b'),
 Text(627.75, 181.19999999999982, 'gini = 0.0\nsamples = 320\nvalue = [0,
527, 0, 0]\nclass = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.133\nsamples = 37\nvalue = [0
52, 0, 4]\nclass = b'),
 Text(976.5, 543.5999999999999, 'PXY <= 0.555\ngini = 0.494\nsamples = 152
\nvalue = [0, 47, 29, 154]\nclass = d'),
 Text(906.75, 181.19999999999982, 'gini = 0.455\nsamples = 28\nvalue = [0
26, 0, 14]\nclass = b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.422\nsamples = 124\nvalue =
[0, 21, 29, 140]\nclass = d'),
 Text(1674.0, 1268.4, 'TCH <= 1.215\ngini = 0.157\nsamples = 1813\nvalue =
[7, 2639, 124, 110]\nclass = b'),
 Text(1395.0, 906.0, 'TOL <= 2.175\ngini = 0.511\nsamples = 115\nvalue =
[1, 89, 88, 1]\nclass = b'),
 Text(1255.5, 543.5999999999999, 'OXY <= 0.54\ngini = 0.28\nsamples = 59\n
value = [0, 79, 16, 0]\nclass = b'),
 Text(1185.75, 181.19999999999982, 'gini = 0.14\nsamples = 29\nvalue = [0
49, 4, 0]\nclass = b'),
 Text(1325.25, 181.19999999999982, 'gini = 0.408\nsamples = 30\nvalue =
[0, 30, 12, 0]\nclass = b'),
 Text(1534.5, 543.5999999999999, 'NO_2 <= 38.705\ngini = 0.251\nsamples =
56\nvalue = [1, 10, 72, 1]\nclass = c'),
 Text(1464.75, 181.19999999999982, 'gini = 0.431\nsamples = 25\nvalue =
[0, 9, 24, 1]\nclass = c'),
 Text(1604.25, 181.19999999999982, 'gini = 0.078\nsamples = 31\nvalue =
[1, 1, 48, 0]\nclass = c'),
 Text(1953.0, 906.0, 'MXY <= 1.155\ngini = 0.107\nsamples = 1698\nvalue =
[6, 2550, 36, 109]\nclass = b'),
 Text(1813.5, 543.5999999999999, 'NO_2 <= 72.7\ngini = 0.025\nsamples = 12
95\nvalue = [0, 2020, 16, 10]\nclass = b'),
 Text(1743.75, 181.19999999999982, 'gini = 0.013\nsamples = 1258\nvalue =
[0, 1979, 4, 9]\nclass = b'),
 Text(1883.25, 181.19999999999982, 'gini = 0.374\nsamples = 37\nvalue =
[0, 41, 12, 1]\nclass = b'),
 Text(2092.5, 543.5999999999999, 'PXY <= 0.785\ngini = 0.321\nsamples = 40
3\nvalue = [6, 530, 20, 99]\nclass = b'),
 Text(2022.75, 181.19999999999982, 'gini = 0.163\nsamples = 340\nvalue =
```

[5, 509, 8, 36]\nclass = b'),
Text(2162.25, 181.19999999999982, 'gini = 0.516\nsamples = 63\nvalue =
[1, 21, 12, 63]\nclass = d'),
Text(3348.0, 1630.8000000000002, 'EBE <= 3.245\ngini = 0.73\nsamples = 1
712\nvalue = [5027, 2390, 5432, 5697]\nclass = d'),
Text(2790.0, 1268.4, 'SO_2 <= 5.625\ngini = 0.716\nsamples = 7251\nvalue
= [1731, 2071, 3306, 4413]\nclass = d'),
Text(2511.0, 906.0, 'NOx <= 41.925\ngini = 0.247\nsamples = 1123\nvalue =
[132, 1511, 0, 109]\nclass = b'),
Text(2371.5, 543.5999999999999, 'CO <= 0.285\ngini = 0.497\nsamples = 33
\nvalue = [103, 354, 0, 73]\nclass = b'),
Text(2301.75, 181.19999999999982, 'gini = 0.639\nsamples = 128\nvalue =
[100, 57, 0, 56]\nclass = a'),
Text(2441.25, 181.19999999999982, 'gini = 0.119\nsamples = 205\nvalue =
[3, 297, 0, 17]\nclass = b'),
Text(2650.5, 543.5999999999999, 'EBE <= 1.125\ngini = 0.102\nsamples = 79
0\nvalue = [29, 1157, 0, 36]\nclass = b'),
Text(2580.75, 181.19999999999982, 'gini = 0.177\nsamples = 195\nvalue =
[6, 273, 0, 23]\nclass = b'),
Text(2720.25, 181.19999999999982, 'gini = 0.076\nsamples = 595\nvalue =
[23, 884, 0, 13]\nclass = b'),
Text(3069.0, 906.0, 'SO_2 <= 6.985\ngini = 0.661\nsamples = 6128\nvalue =
[1599, 560, 3306, 4304]\nclass = d'),
Text(2929.5, 543.5999999999999, 'NMHC <= 0.025\ngini = 0.664\nsamples = 8
25\nvalue = [286, 293, 101, 637]\nclass = d'),
Text(2859.75, 181.19999999999982, 'gini = 0.054\nsamples = 177\nvalue =
[279, 0, 7, 1]\nclass = a'),
Text(2999.25, 181.19999999999982, 'gini = 0.529\nsamples = 648\nvalue =
[7, 293, 94, 636]\nclass = d'),
Text(3208.5, 543.5999999999999, 'PM10 <= 8.175\ngini = 0.643\nsamples = 5
303\nvalue = [1313, 267, 3205, 3667]\nclass = d'),
Text(3138.75, 181.19999999999982, 'gini = 0.558\nsamples = 407\nvalue =
[115, 4, 401, 144]\nclass = c'),
Text(3278.25, 181.19999999999982, 'gini = 0.641\nsamples = 4896\nvalue =
[1198, 263, 2804, 3523]\nclass = d'),
Text(3906.0, 1268.4, 'NMHC <= 0.145\ngini = 0.653\nsamples = 4461\nvalue
= [3296, 319, 2126, 1284]\nclass = a'),
Text(3627.0, 906.0, 'NOx <= 181.2\ngini = 0.495\nsamples = 1274\nvalue =
[1282, 23, 644, 73]\nclass = a'),
Text(3487.5, 543.5999999999999, 'TCH <= 1.255\ngini = 0.471\nsamples = 10
25\nvalue = [1100, 23, 437, 70]\nclass = a'),
Text(3417.75, 181.19999999999982, 'gini = 0.513\nsamples = 273\nvalue =
[208, 3, 226, 3]\nclass = c'),
Text(3557.25, 181.19999999999982, 'gini = 0.403\nsamples = 752\nvalue =
[892, 20, 211, 67]\nclass = a'),
Text(3766.5, 543.5999999999999, 'SO_2 <= 30.22\ngini = 0.506\nsamples = 2
49\nvalue = [182, 0, 207, 3]\nclass = c'),
Text(3696.75, 181.19999999999982, 'gini = 0.481\nsamples = 208\nvalue =
[126, 0, 204, 3]\nclass = c'),
Text(3836.25, 181.19999999999982, 'gini = 0.097\nsamples = 41\nvalue = [5
6, 0, 3, 0]\nclass = a'),
Text(4185.0, 906.0, 'TCH <= 1.365\ngini = 0.688\nsamples = 3187\nvalue =
[2014, 296, 1482, 1211]\nclass = a'),
Text(4045.5, 543.5999999999999, 'NMHC <= 0.155\ngini = 0.25\nsamples = 27
2\nvalue = [24, 11, 368, 24]\nclass = c'),
Text(3975.75, 181.19999999999982, 'gini = 0.592\nsamples = 47\nvalue = [2

Conclusion

Accuracy

Linear Regression:0.19949010050345402

Ridge Regression:0.19877953137155935

Lasso Regression:0.05898701096143610 4

ElasticNet Regression:0.09846653794618532

Logistic Regression:0.8480699292793158

Random Forest:0.7732727433159798

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