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("2008.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37
...	...	...	...	...	...	...	...	...	...	...	
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12

226392 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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [6]:

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

Out[6]:

	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...	...	...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

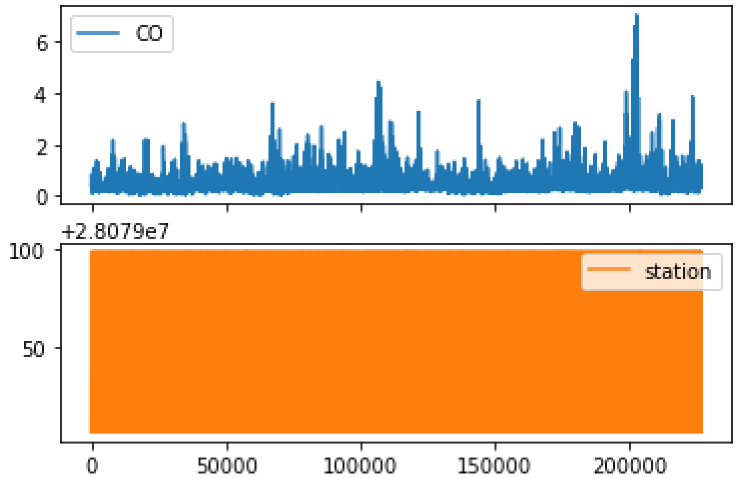
Line chart

In [7]:

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

Out[7]:

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



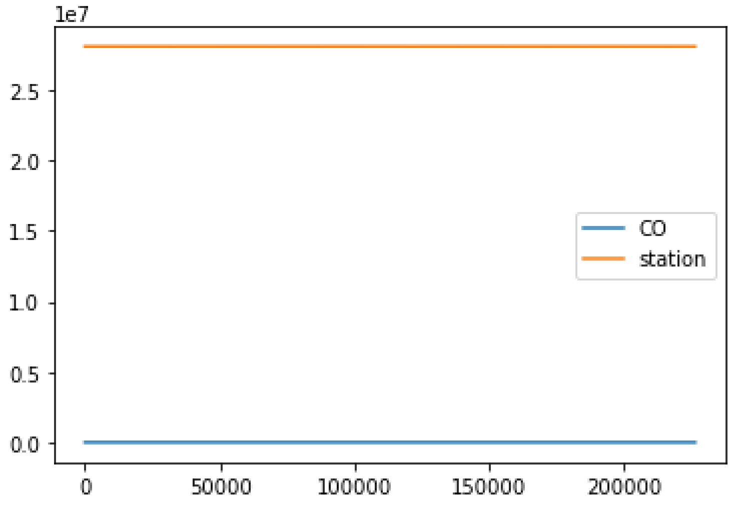
Line chart

In [8]:

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

Out[8]:

```
<AxesSubplot:>
```



Bar chart

In [9]:

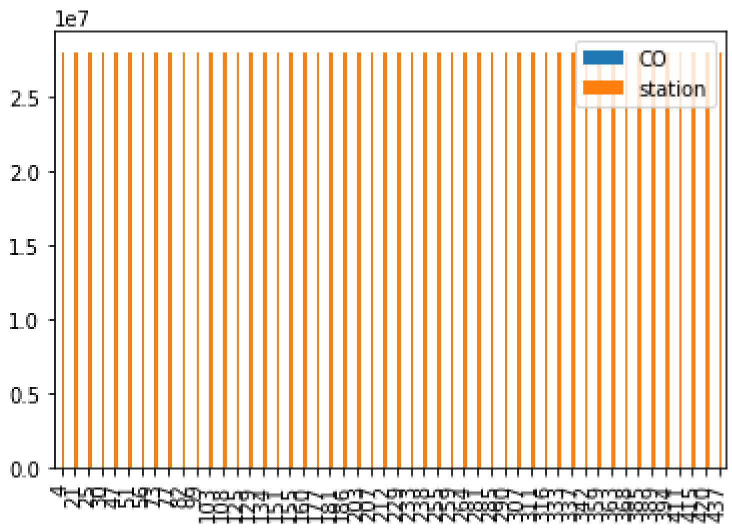
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<AxesSubplot:>
```



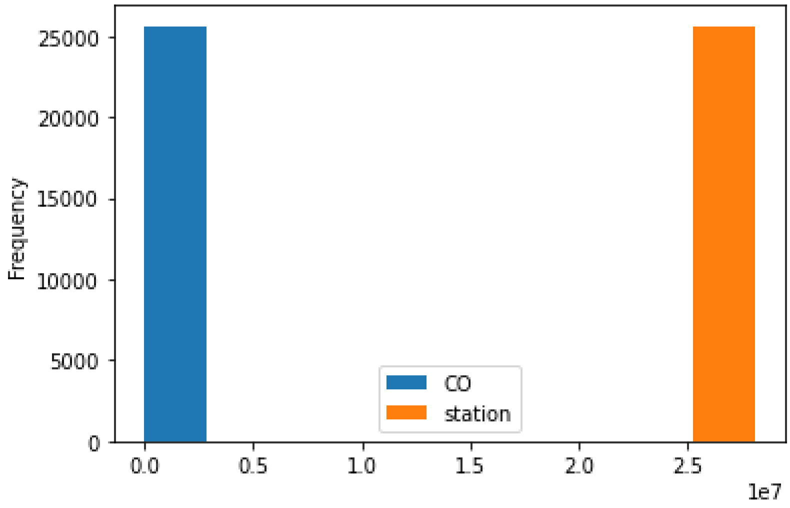
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



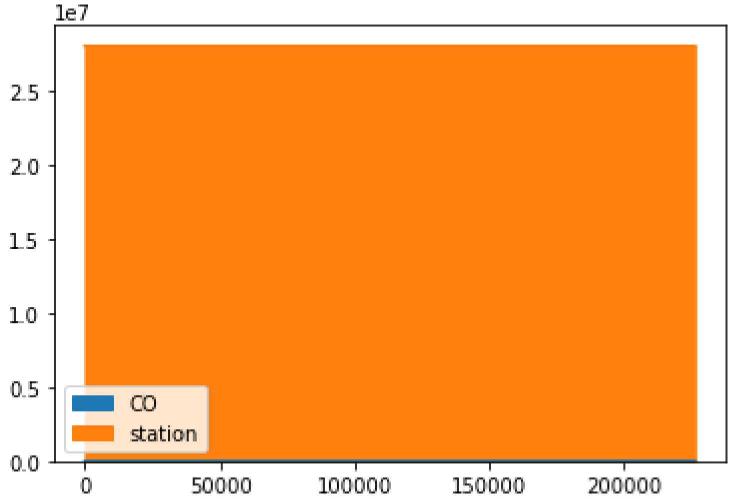
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



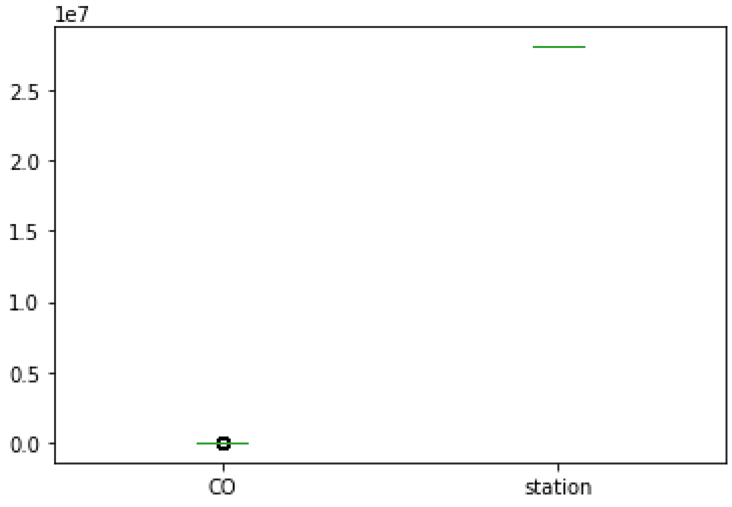
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



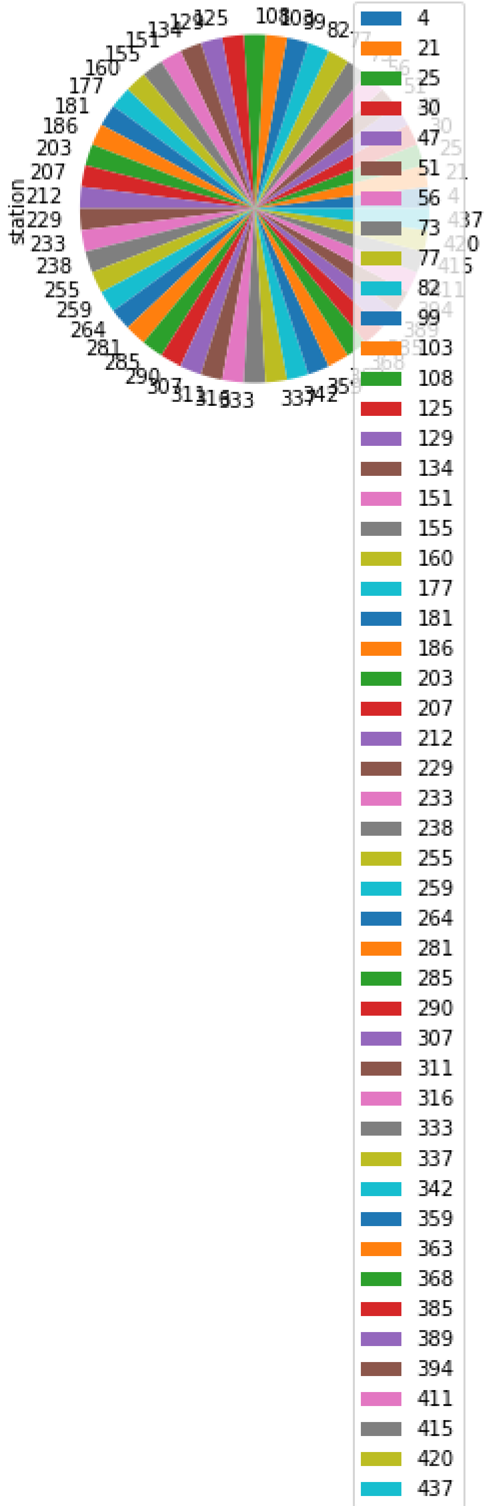
Pie chart

In [14]:

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

Out[14]:

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



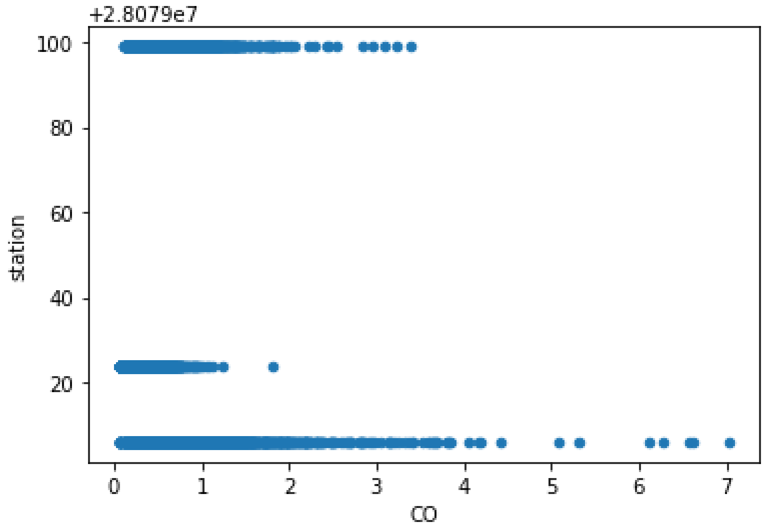
Scatter chart

In [15]:

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

Out[15]:

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



In [16]:

```
df.info()
```

```
---  ------   --------------  -----
 0   date     25631 non-null  object
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024

In [18]:

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

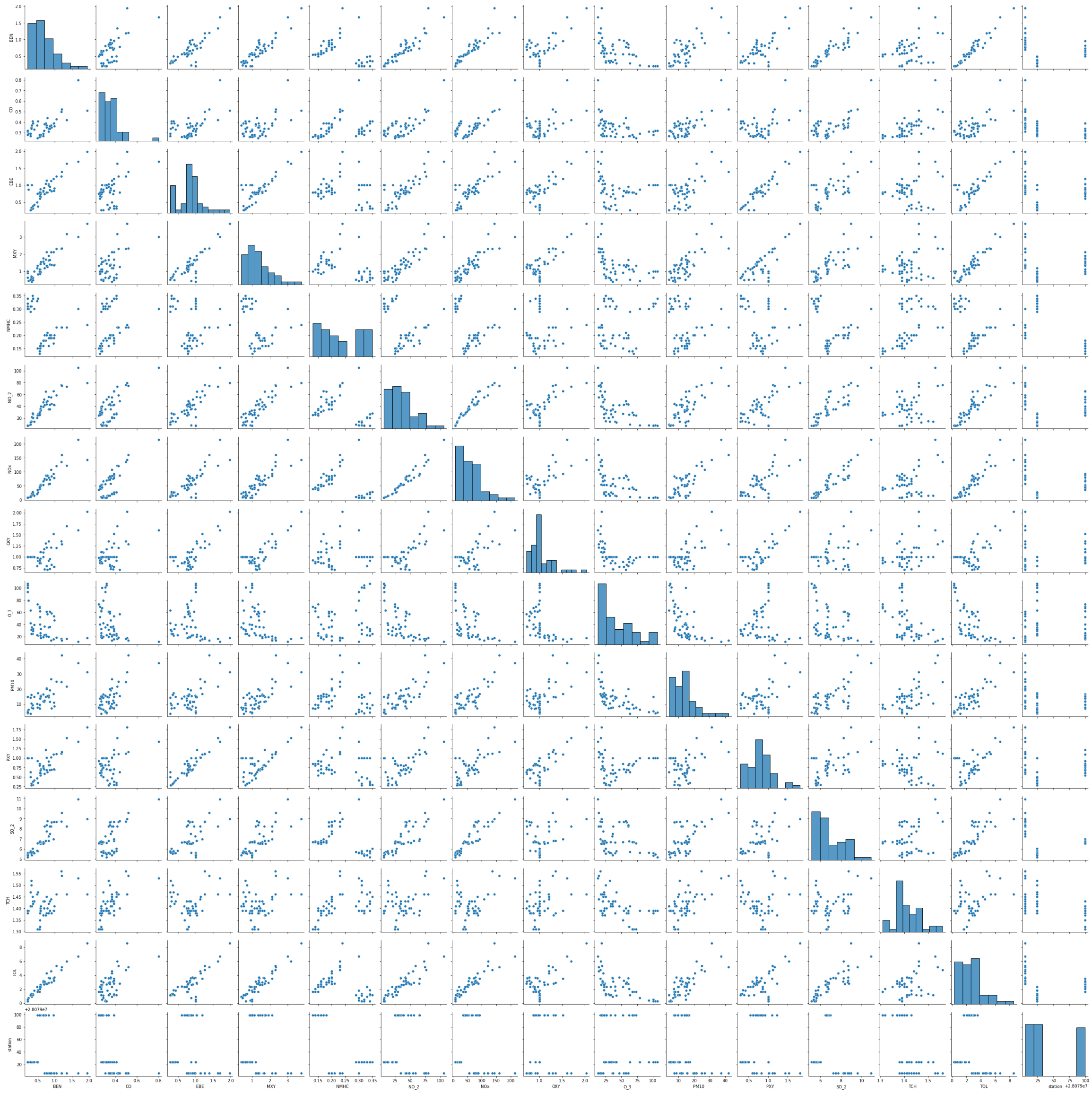
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x20da6286b20>



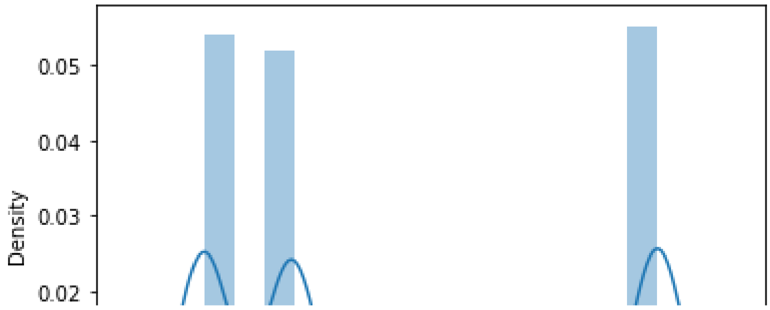
In [20]:

```
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'>

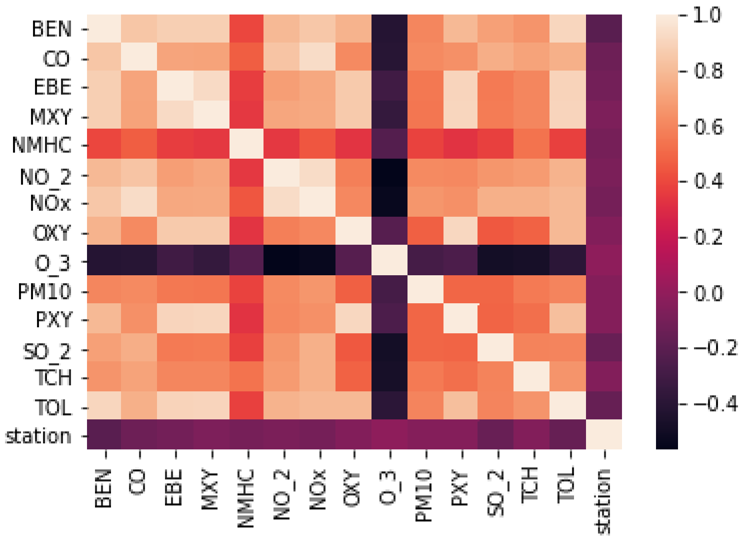


In [21]:

```
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]:

```
28079035.12432534
```

In [26]:

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

Out[26]:

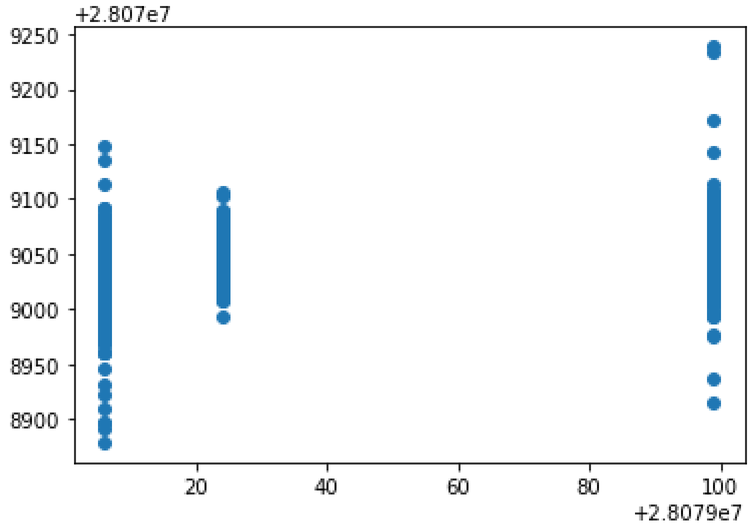
	Co-efficient
BEN	-25.677784
CO	0.039079
EBE	-0.975018
MXY	7.553562
NMHC	-27.887432
NO_2	-0.040768
NOx	0.125295
OXY	4.057897
O_3	-0.143160
PM10	0.140121
PXY	2.327563
SO_2	-0.637859
TCH	17.621654
TOL	-1.908789

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x20db4c95700>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.1425413978965746
```

In [29]:

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

Out[29]:

```
0.14416558651112388
```

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]:

```
0.14254972762293971
```

In [33]:

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

Out[33]:

```
0.14414036663149943
```

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]:

```
0.04327523622896379
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.04025531021711837
```

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([-4.64079915, -0.        ,  0.        ,  3.19190927, -0.        ,
        0.06050465,  0.02579208,  1.57297725, -0.15712899,  0.13493911,
        1.54451818, -0.95966869,  0.        , -2.49989346])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079057.330976218
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.09278636545773644
```

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)))
```

```
35.88332447042623
1499.635061630427
38.72512184138905
```

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]:

```
(25631, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(25631,)
```

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)
```

```
[28079099]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.794194530061254
```

In [54]:

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

Out[54]:

```
8.321803242555043e-09
```

In [55]:

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

Out[55]:

```
array([[8.32180324e-09, 1.19114634e-13, 9.99999992e-01]])
```

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]:

```
0.8517919874855023
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

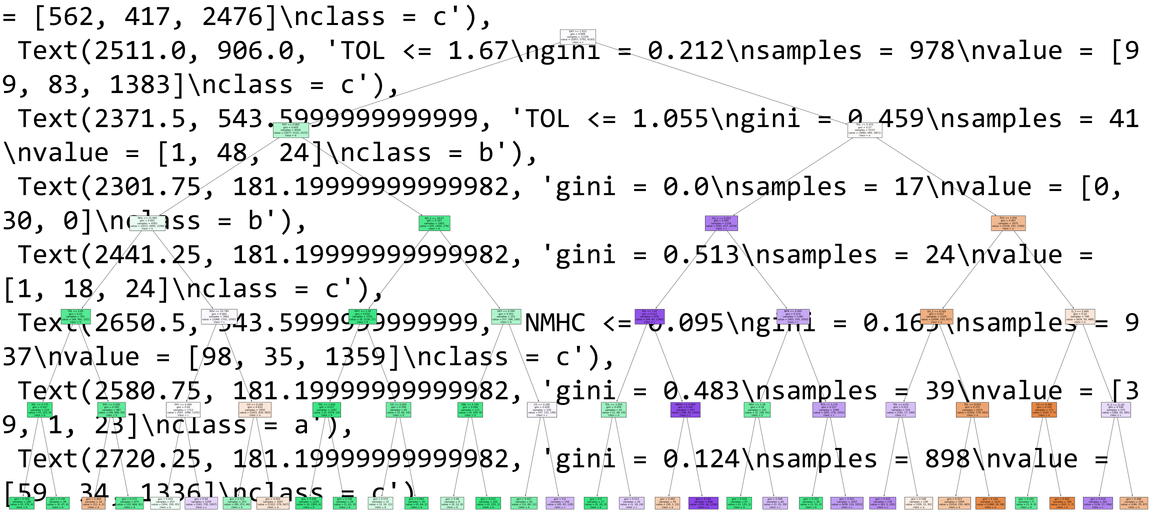
```
from sklearn.tree import plot_tree

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, 'OXY <= 1.015\ngini = 0.666\nsamples = 11259\nvalue
= [5957, 5791, 6193]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'PXY <= 0.985\ngini = 0.605\nsamples = 6
026\nvalue = [2077, 5122, 2372]\nclass = b'),
 Text(558.0, 1268.4, 'NOx <= 21.665\ngini = 0.661\nsamples = 4363\nvalue =
[2012, 2692, 2196]\nclass = b'),
 Text(279.0, 906.0, 'TOL <= 0.85\ngini = 0.23\nsamples = 701\nvalue = [44,
991, 101]\nclass = b'),
 Text(139.5, 543.5999999999999, 'TOL <= 0.775\ngini = 0.047\nsamples = 214
\nvalue = [0, 323, 8]\nclass = b'),
 Text(69.75, 181.19999999999982, 'gini = 0.028\nsamples = 186\nvalue = [0,
276, 4]\nclass = b'),
 Text(209.25, 181.19999999999982, 'gini = 0.145\nsamples = 28\nvalue = [0,
47, 4]\nclass = b'),
 Text(418.5, 543.5999999999999, 'PXY <= 0.295\ngini = 0.295\nsamples = 487
\nvalue = [44, 668, 93]\nclass = b'),
 Text(348.75, 181.19999999999982, 'gini = 0.494\nsamples = 14\nvalue = [1
2, 4, 2]\nclass = a'),
 Text(488.25, 181.19999999999982, 'gini = 0.273\nsamples = 473\nvalue = [3
2, 664, 91]\nclass = b'),
 Text(837.0, 906.0, 'NOx <= 52.785\ngini = 0.664\nsamples = 3662\nvalue =
[1968, 1701, 2095]\nclass = c'),
 Text(697.5, 543.5999999999999, 'PXY <= 0.495\ngini = 0.64\nsamples = 1712
\nvalue = [547, 1069, 1102]\nclass = c'),
 Text(627.75, 181.19999999999982, 'gini = 0.603\nsamples = 453\nvalue = [3
04, 339, 95]\nclass = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.59\nsamples = 1259\nvalue = [2
43, 730, 1007]\nclass = c'),
 Text(976.5, 543.5999999999999, 'CO <= 0.245\ngini = 0.633\nsamples = 1950
\nvalue = [1421, 632, 993]\nclass = a'),
 Text(906.75, 181.19999999999982, 'gini = 0.521\nsamples = 253\nvalue = [9
9, 253, 46]\nclass = b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.602\nsamples = 1697\nvalue =
[1322, 379, 947]\nclass = a'),
 Text(1674.0, 1268.4, 'NO_2 <= 39.67\ngini = 0.167\nsamples = 1663\nvalue
= [65, 2430, 176]\nclass = b'),
 Text(1395.0, 906.0, 'MXY <= 1.29\ngini = 0.032\nsamples = 1332\nvalue =
[8, 2134, 27]\nclass = b'),
 Text(1255.5, 543.5999999999999, 'TOL <= 1.635\ngini = 0.017\nsamples = 12
87\nvalue = [5, 2074, 13]\nclass = b'),
 Text(1185.75, 181.19999999999982, 'gini = 0.008\nsamples = 1229\nvalue =
[4, 1993, 4]\nclass = b'),
 Text(1325.25, 181.19999999999982, 'gini = 0.198\nsamples = 58\nvalue =
[1, 81, 9]\nclass = b'),
 Text(1534.5, 543.5999999999999, 'CO <= 0.265\ngini = 0.358\nsamples = 45
\nvalue = [3, 60, 14]\nclass = b'),
 Text(1464.75, 181.19999999999982, 'gini = 0.574\nsamples = 15\nvalue =
[3, 16, 12]\nclass = b'),
 Text(1604.25, 181.19999999999982, 'gini = 0.083\nsamples = 30\nvalue =
[0, 44, 2]\nclass = b'),
 Text(1953.0, 906.0, 'OXY <= 0.595\ngini = 0.551\nsamples = 331\nvalue =
[57, 296, 149]\nclass = b'),
 Text(1813.5, 543.5999999999999, 'EBE <= 0.705\ngini = 0.098\nsamples = 11
1\nvalue = [0, 165, 9]\nclass = b'),
 Text(1743.75, 181.19999999999982, 'gini = 0.48\nsamples = 9\nvalue = [0,
12, 8]\nclass = b'),
 Text(1883.25, 181.19999999999982, 'gini = 0.013\nsamples = 102\nvalue =
[0, 153, 1]\nclass = b'),
 Text(2092.5, 543.5999999999999, 'CO <= 0.285\ngini = 0.628\nsamples = 220
\nvalue = [57, 131, 140]\nclass = c'),
 Text(2022.75, 181.19999999999982, 'gini = 0.427\nsamples = 62\nvalue = [1
```

```
2, 66, 12]\nclass = b'),
 Text(2162.25, 181.19999999999982, 'gini = 0.6\nsamples = 158\nvalue = [4
5, 65, 128]\nclass = c'),
 Text(3348.0, 1630.8000000000002, 'TOL <= 5.415\ngini = 0.57\nsamples = 52
33\nvalue = [3880, 669, 3821]\nclass = a'),
 Text(2790.0, 1268.4, 'SO_2 <= 8.055\ngini = 0.445\nsamples = 2158\nvalue
= [562, 417, 2476]\nclass = c'),
 Text(2511.0, 906.0, 'TOL <= 1.67\ngini = 0.212\nsamples = 978\nvalue = [9
9, 83, 1383]\nclass = c'),
 Text(2371.5, 543.5999999999999, 'TOL <= 1.055\ngini = 0.459\nsamples = 41
\nvalue = [1, 48, 24]\nclass = b'),
 Text(2301.75, 181.19999999999982, 'gini = 0.0\nsamples = 17\nvalue = [0,
30, 0]\nclass = b'),
 Text(2441.25, 181.19999999999982, 'gini = 0.513\nsamples = 24\nvalue =
[1, 18, 24]\nclass = c'),
 Text(2650.5, 543.5999999999999, 'NMHC <= 0.095\ngini = 0.163\nsamples = 9
37\nvalue = [98, 35, 1359]\nclass = c'),
 Text(2580.75, 181.19999999999982, 'gini = 0.483\nsamples = 39\nvalue = [3
9, 1, 23]\nclass = a'),
 Text(2720.25, 181.19999999999982, 'gini = 0.124\nsamples = 898\nvalue =
[59, 34, 1336]\nclass = c'),
 Text(3069.0, 906.0, 'BEN <= 0.495\ngini = 0.574\nsamples = 1180\nvalue =
[463, 334, 1093]\nclass = c'),
 Text(2929.5, 543.5999999999999, 'MXY <= 1.38\ngini = 0.34\nsamples = 131
\nvalue = [2, 158, 41]\nclass = b'),
 Text(2859.75, 181.19999999999982, 'gini = 0.029\nsamples = 87\nvalue =
[0, 133, 2]\nclass = b'),
 Text(2999.25, 181.19999999999982, 'gini = 0.506\nsamples = 44\nvalue =
[2, 25, 39]\nclass = c'),
 Text(3208.5, 543.5999999999999, 'TCH <= 1.275\ngini = 0.527\nsamples = 10
49\nvalue = [461, 176, 1052]\nclass = c'),
 Text(3138.75, 181.19999999999982, 'gini = 0.103\nsamples = 39\nvalue =
[3, 52, 0]\nclass = b'),
 Text(3278.25, 181.19999999999982, 'gini = 0.501\nsamples = 1010\nvalue =
[458, 124, 1052]\nclass = c'),
 Text(3906.0, 1268.4, 'TCH <= 1.605\ngini = 0.467\nsamples = 3075\nvalue =
[3318, 252, 1345]\nclass = a'),
 Text(3627.0, 906.0, 'SO_2 <= 8.505\ngini = 0.427\nsamples = 2329\nvalue =
[2694, 193, 857]\nclass = a'),
 Text(3487.5, 543.5999999999999, 'TOL <= 6.63\ngini = 0.512\nsamples = 314
\nvalue = [192, 17, 294]\nclass = c'),
 Text(3417.75, 181.19999999999982, 'gini = 0.433\nsamples = 178\nvalue =
[83, 5, 202]\nclass = c'),
 Text(3557.25, 181.19999999999982, 'gini = 0.548\nsamples = 136\nvalue =
[109, 12, 92]\nclass = a'),
 Text(3766.5, 543.5999999999999, 'TOL <= 8.925\ngini = 0.371\nsamples = 20
15\nvalue = [2502, 176, 563]\nclass = a'),
 Text(3696.75, 181.19999999999982, 'gini = 0.523\nsamples = 1005\nvalue =
[1014, 150, 465]\nclass = a'),
 Text(3836.25, 181.19999999999982, 'gini = 0.144\nsamples = 1010\nvalue =
[1488, 26, 98]\nclass = a'),
 Text(4185.0, 906.0, 'O_3 <= 5.495\ngini = 0.54\nsamples = 746\nvalue = [6
24, 59, 488]\nclass = a'),
 Text(4045.5, 543.5999999999999, 'CO <= 0.465\ngini = 0.105\nsamples = 171
\nvalue = [242, 7, 7]\nclass = a'),
 Text(3975.75, 181.19999999999982, 'gini = 0.245\nsamples = 5\nvalue = [1,
6, 0]\nclass = b'),
 Text(4115.25, 181.19999999999982, 'gini = 0.062\nsamples = 166\nvalue =
[241, 1, 7]\nclass = a'),
 Text(4324.5, 543.5999999999999, 'O_3 <= 11.85\ngini = 0.546\nsamples = 57
5\nvalue = [382, 52, 481]\nclass = c'),
```



Conclusion

Scores

Linear Regression

In [64]:

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

Out[64]:

```
0.1425413978965746
```

In [65]:

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

Out[65]:

```
0.14416538651112388
```

Lasso

In [66]:

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

Out[66]:

```
0.04025531021711837
```

```
 Text(4254.75, 181.19999999999982, 'gini = 0.418\nsamples = 341\nvalue = 
[136, 17, 394]\nclass = c'),
 Text(4834.25, 181.19999999999982, 'gini = 0.488\nsamples = 234\nvalue = 
[246, 35, 87]\nclass = a')]
```

Ridge

In [68]:

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

Out[68]:

```
0.14254972762293971
```

In [69]:

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

Out[69]:

```
0.14414036663149943
```

Elastic Net

In [71]:

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

Out[71]:

```
0.09278636545773644
```

Logistic Regression

In [70]:

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

Out[70]:

```
0.794194530061254
```

Random Forest

In [72]:

```
grid_search.best_score_
```

Out[72]:

```
0.8517919874855023
```

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

In [ ]:

```
```