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("2007.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	1
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	1
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	1
...	...	...	...	...	...	...	...	...	...	...	
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	

225120 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: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 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	1.86	28079006
21	0.31	28079024
25	1.42	28079099
30	1.89	28079006
47	0.30	28079024
...	...	...
225073	0.47	28079006
225094	0.45	28079099
225098	0.41	28079006
225115	0.45	28079024
225119	0.40	28079099

25443 rows × 2 columns

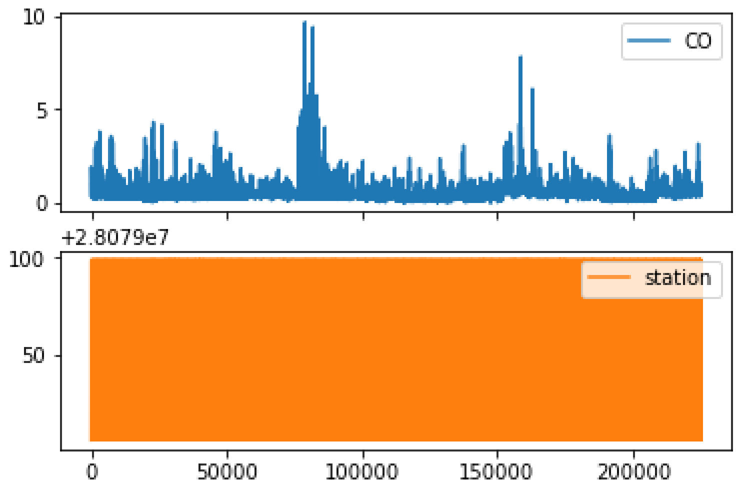
Line chart

In [7]:

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

Out[7]:

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



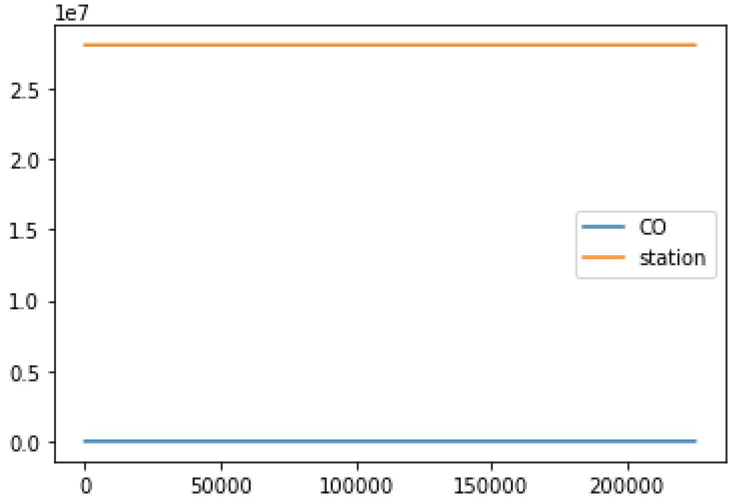
Line chart

In [8]:

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

Out[8]:

```
<AxesSubplot:>
```



Bar chart

In [9]:

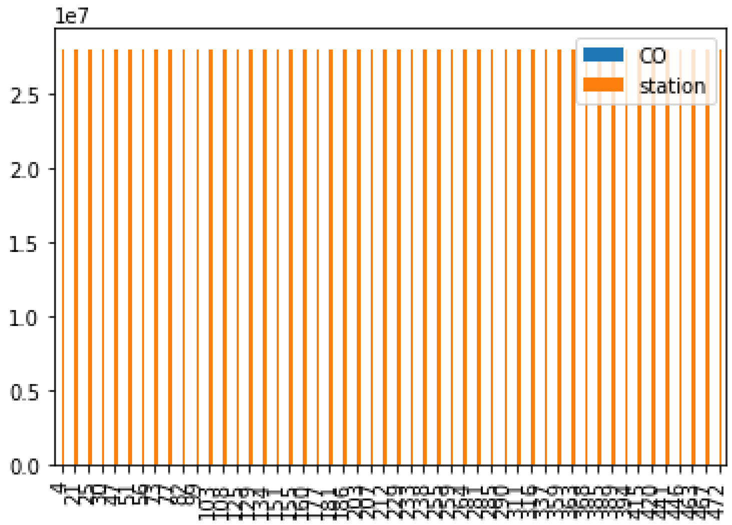
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<AxesSubplot:>
```



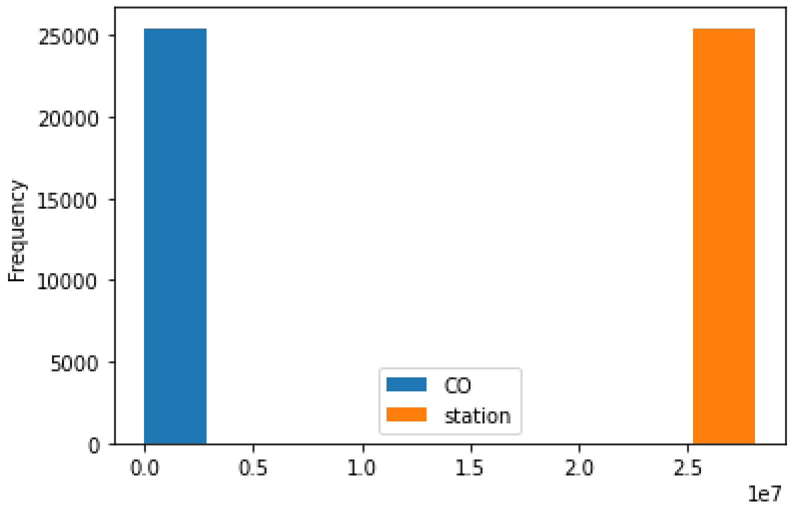
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



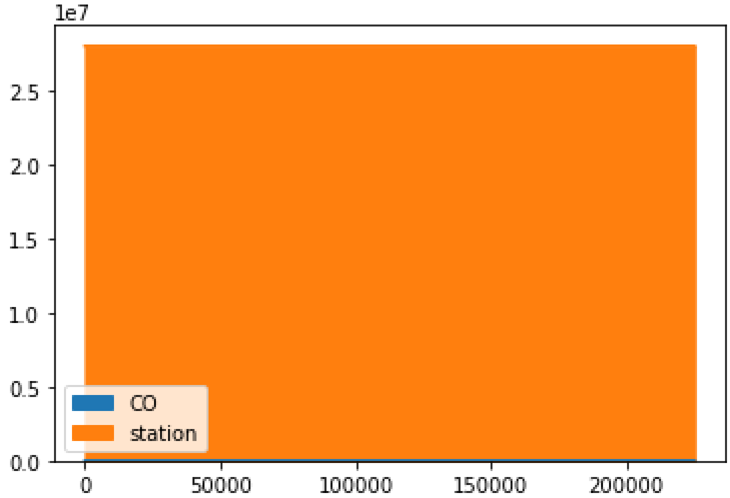
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



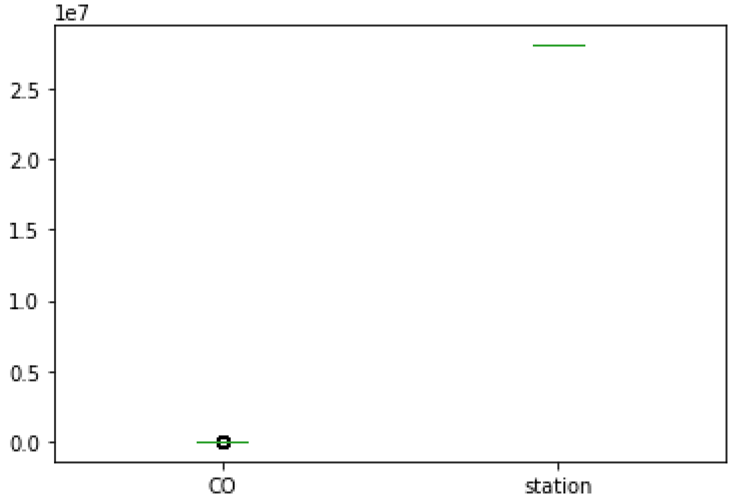
Box chart

In [13]:

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

Out[13]:

```
<AxesSubplot:>
```



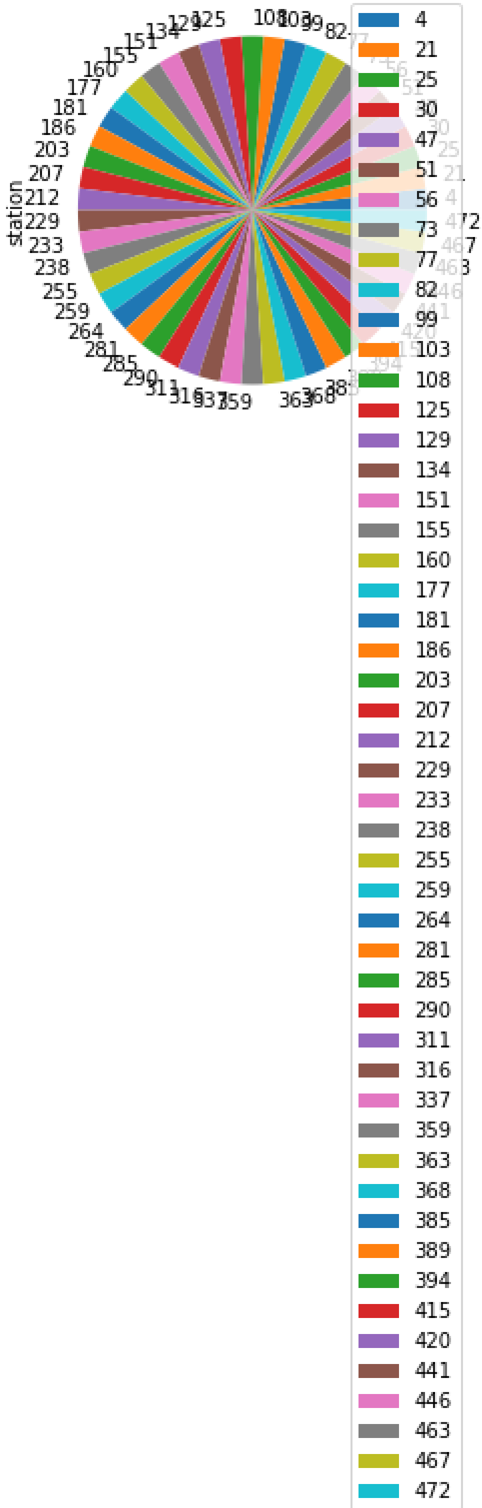
Pie chart

In [14]:

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

Out[14]:

<AxesSubplot:ylabel='station'>



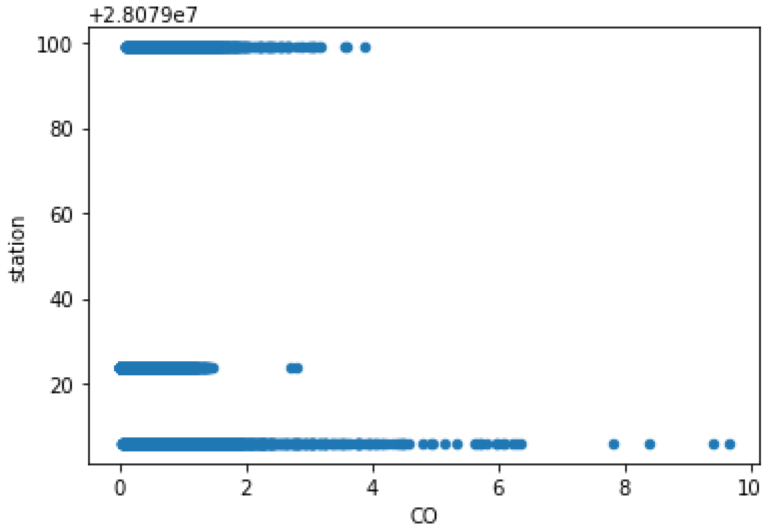
Scatter chart

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: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     25443 non-null  object
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000
25%	0.450000	0.260000	0.780000	0.960000	0.160000	31.285001
50%	0.770000	0.400000	1.000000	1.500000	0.220000	54.080002
75%	1.390000	0.640000	1.580000	2.855000	0.300000	79.230003
max	30.139999	9.660000	31.680000	65.480003	2.570000	430.299988

In [18]:

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

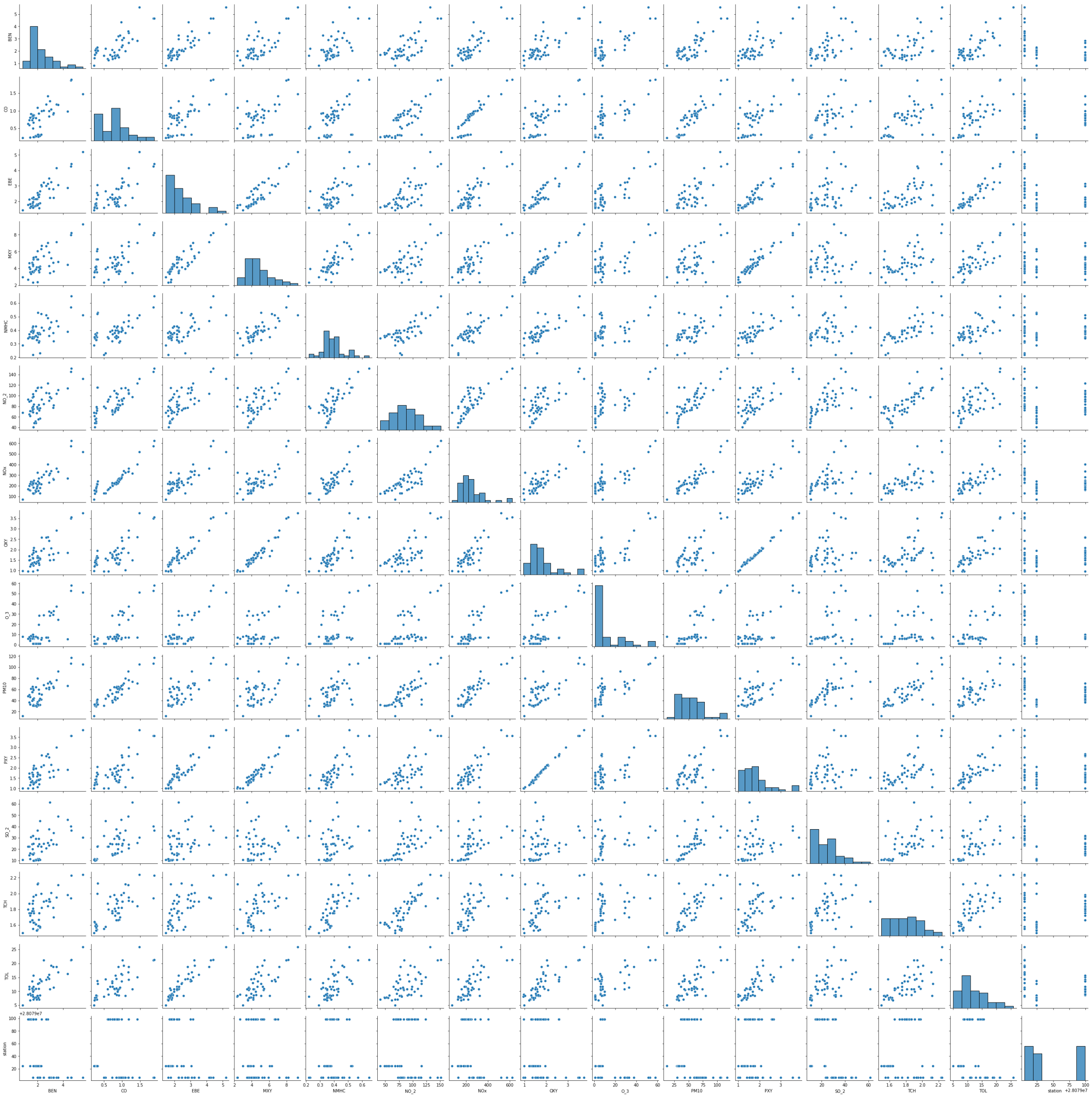
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x1adb46b4250>



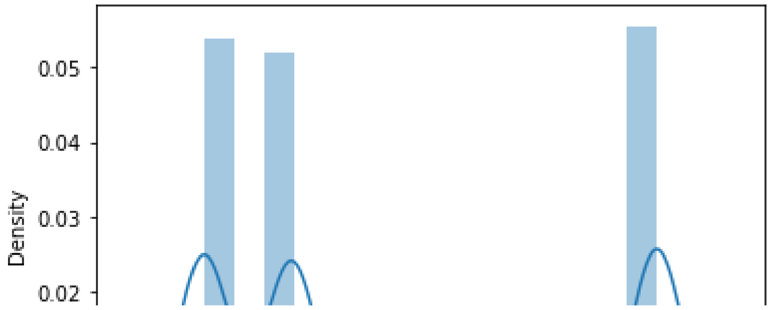
In [20]:

```
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'>

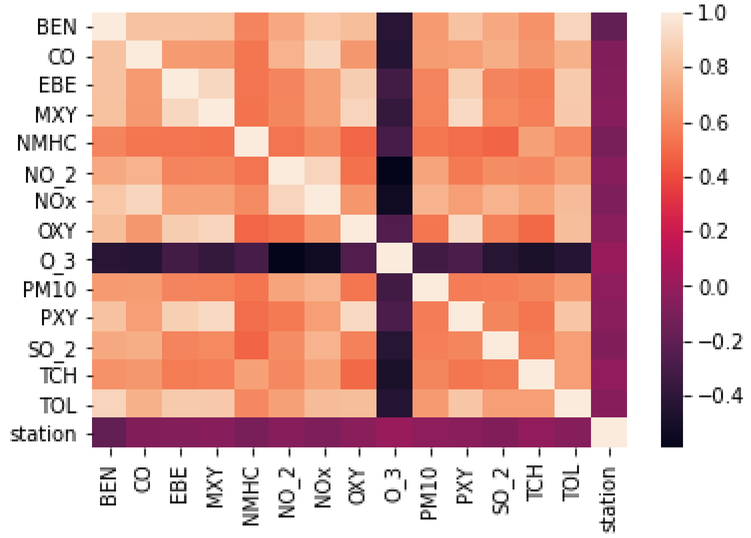


In [21]:

```
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]:

```
28079011.361880615
```

In [26]:

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

Out[26]:

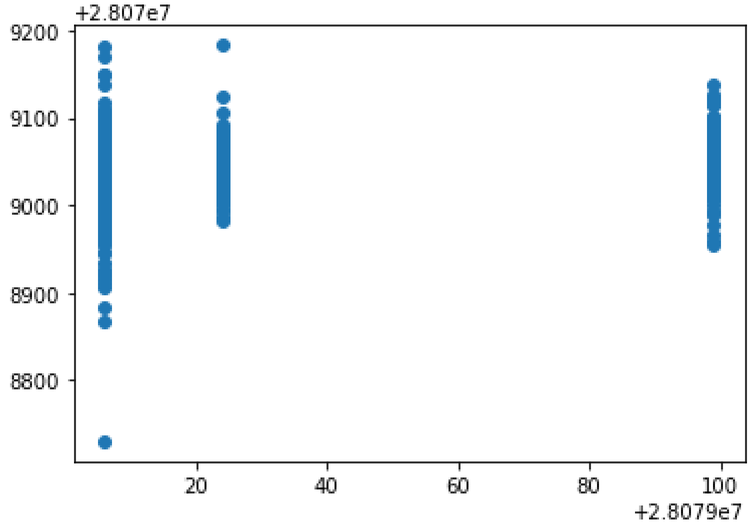
	Co-efficient
BEN	-33.020995
CO	17.178681
EBE	0.697249
MXY	-1.427224
NMHC	-40.329386
NO_2	0.095271
NOx	-0.030673
OXY	5.978076
O_3	-0.037292
PM10	0.154694
PXY	6.482152
SO_2	0.166995
TCH	24.965438
TOL	3.199910

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x1adc307df70>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.16331457098631952
```

In [29]:

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

Out[29]:

```
0.15751999889546764
```

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]:

```
0.16317654437433604
```

In [33]:

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

Out[33]:

```
0.15747044218625972
```

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]:

```
0.013356204359641799
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.013732764982463452
```

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([-7.95649726,  0.        ,  0.        ,  0.        , -0.        ,
        0.04361549, -0.04782221,  0.79287979, -0.06193482,  0.17492139,
        0.70353371, -0.01244352,  0.        ,  0.89998083])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079046.267567
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.0693172677037851
```

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)))
```

```
36.607853145883624
1531.6829499262737
39.13672124650037
```

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]:

```
(25443, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(25443,)
```

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]:

```
0.8146838030106512
```

In [54]:

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

Out[54]:

```
1.0827539771181323e-19
```

In [55]:

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

Out[55]:

```
array([[1.08275398e-19, 1.80383815e-19, 1.00000000e+00]])
```

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]:

```
0.8206625491297024
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

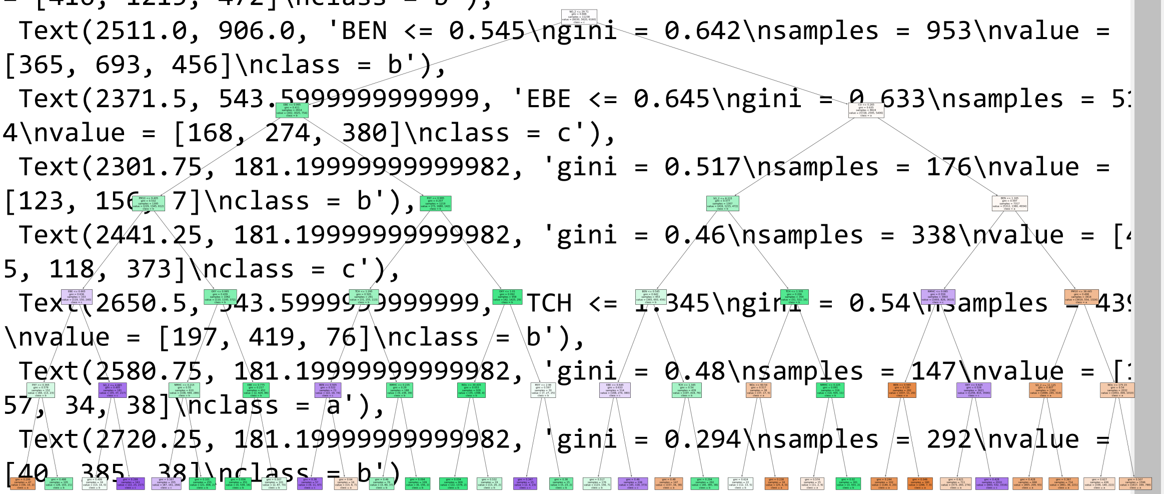
```
from sklearn.tree import plot_tree

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, 'NO_2 <= 29.71\ngini = 0.666\nsamples = 11238\nvalu
= [6030, 5620, 6160]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'EBE <= 0.995\ngini = 0.411\nsamples = 
614\nvalue = [302, 3025, 754]\nclass = b'),
 Text(558.0, 1268.4, 'PM10 <= 9.405\ngini = 0.532\nsamples = 1395\nvalue 
[229, 1345, 612]\nclass = b'),
 Text(279.0, 906.0, 'EBE <= 0.665\ngini = 0.636\nsamples = 333\nvalue = [
19, 150, 240]\nclass = c'),
 Text(139.5, 543.5999999999999, 'PXY <= 0.365\ngini = 0.579\nsamples = 15
\nvalue = [84, 113, 23]\nclass = b'),
 Text(69.75, 181.19999999999982, 'gini = 0.299\nsamples = 47\nvalue = [56
10, 2]\nclass = a'),
 Text(209.25, 181.19999999999982, 'gini = 0.488\nsamples = 105\nvalue = [
8, 103, 21]\nclass = b'),
 Text(418.5, 543.5999999999999, 'SO_2 <= 4.865\ngini = 0.405\nsamples = 1
1\nvalue = [35, 37, 217]\nclass = c'),
 Text(348.75, 181.19999999999982, 'gini = 0.499\nsamples = 18\nvalue = [1
3, 14, 0]\nclass = b'),
 Text(488.25, 181.19999999999982, 'gini = 0.299\nsamples = 163\nvalue = [
2, 23, 217]\nclass = c'),
 Text(837.0, 906.0, 'OXY <= 0.995\ngini = 0.439\nsamples = 1062\nvalue = 
[110, 1195, 372]\nclass = b'),
 Text(697.5, 543.5999999999999, 'NMHC <= 0.215\ngini = 0.55\nsamples = 61
\nvalue = [108, 569, 286]\nclass = b'),
 Text(627.75, 181.19999999999982, 'gini = 0.597\nsamples = 355\nvalue = [
7, 161, 284]\nclass = c'),
 Text(767.25, 181.19999999999982, 'gini = 0.101\nsamples = 255\nvalue = [
1, 408, 2]\nclass = b'),
 Text(976.5, 543.5999999999999, 'EBE <= 0.775\ngini = 0.217\nsamples = 45
\nvalue = [2, 626, 86]\nclass = b'),
 Text(906.75, 181.19999999999982, 'gini = 0.056\nsamples = 351\nvalue = 
[0, 539, 16]\nclass = b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.507\nsamples = 101\nvalue = 
[2, 87, 70]\nclass = b'),
 Text(1674.0, 1268.4, 'PXY <= 0.995\ngini = 0.207\nsamples = 1219\nvalue 
[73, 1680, 142]\nclass = b'),
 Text(1395.0, 906.0, 'TCH <= 1.295\ngini = 0.505\nsamples = 261\nvalue = 
[31, 255, 113]\nclass = b'),
 Text(1255.5, 543.5999999999999, 'BEN <= 0.505\ngini = 0.522\nsamples = 7
\nvalue = [22, 19, 74]\nclass = c'),
 Text(1185.75, 181.19999999999982, 'gini = 0.38\nsamples = 57\nvalue = [9
11, 67]\nclass = c'),
 Text(1325.25, 181.19999999999982, 'gini = 0.64\nsamples = 16\nvalue = [1
3, 8, 7]\nclass = a'),
 Text(1534.5, 543.5999999999999, 'NMHC <= 0.235\ngini = 0.29\nsamples = 1
8\nvalue = [9, 236, 39]\nclass = b'),
 Text(1464.75, 181.19999999999982, 'gini = 0.46\nsamples = 79\nvalue = [3
80, 37]\nclass = b'),
 Text(1604.25, 181.19999999999982, 'gini = 0.094\nsamples = 109\nvalue = 
[6, 156, 2]\nclass = b'),
 Text(1953.0, 906.0, 'OXY <= 1.01\ngini = 0.092\nsamples = 958\nvalue = [
2, 1425, 29]\nclass = b'),
 Text(1813.5, 543.5999999999999, 'NOx <= 30.435\ngini = 0.053\nsamples = 
28\nvalue = [35, 1398, 4]\nclass = b'),
 Text(1743.75, 181.19999999999982, 'gini = 0.034\nsamples = 909\nvalue = 
[22, 1378, 2]\nclass = b'),
 Text(1883.25, 181.19999999999982, 'gini = 0.532\nsamples = 19\nvalue = [
3, 20, 2]\nclass = b'),
 Text(2092.5, 543.5999999999999, 'MXY <= 2.96\ngini = 0.597\nsamples = 30
\nvalue = [7, 27, 25]\nclass = b'),
 Text(2022.75, 181.19999999999982, 'gini = 0.347\nsamples = 15\nvalue = 
```

```
[2, 4, 23]\nclass = c'),
 Text(2162.25, 181.19999999999982, 'gini = 0.38\nsamples = 15\nvalue = [5
23, 2]\nclass = b'),
 Text(3348.0, 1630.8000000000002, 'CO <= 0.265\ngini = 0.635\nsamples = 86
24\nvalue = [5728, 2595, 5406]\nclass = a'),
 Text(2790.0, 1268.4, 'SO_2 <= 8.215\ngini = 0.577\nsamples = 1307\nvalue
= [416, 1215, 472]\nclass = b'),
 Text(2511.0, 906.0, 'BEN <= 0.545\ngini = 0.642\nsamples = 953\nvalue =
[365, 693, 456]\nclass = b'),
 Text(2371.5, 543.5999999999999, 'EBE <= 0.645\ngini = 0.633\nsamples = 5
4\nvalue = [168, 274, 380]\nclass = c'),
 Text(2301.75, 181.19999999999982, 'gini = 0.517\nsamples = 176\nvalue =
[123, 156, 7]\nclass = b'),
 Text(2441.25, 181.19999999999982, 'gini = 0.46\nsamples = 338\nvalue = [4
5, 118, 373]\nclass = c'),
 Text(2650.5, 543.5999999999999, 'TCH <= 1.345\ngini = 0.54\nsamples = 439
\nvalue = [197, 419, 76]\nclass = b'),
 Text(2580.75, 181.19999999999982, 'gini = 0.48\nsamples = 147\nvalue = [1
57, 34, 38]\nclass = a'),
 Text(2720.25, 181.19999999999982, 'gini = 0.294\nsamples = 292\nvalue =
[40, 385, 38]\nclass = b'),
 Text(3069.0, 906.0, 'TCH <= 1.335\ngini = 0.206\nsamples = 354\nvalue =
[51, 522, 16]\nclass = b'),
 Text(2929.5, 543.5999999999999, 'NOx <= 60.54\ngini = 0.517\nsamples = 3
\nvalue = [37, 17, 5]\nclass = a'),
 Text(2859.75, 181.19999999999982, 'gini = 0.624\nsamples = 22\nvalue = [1
2, 13, 5]\nclass = b'),
 Text(2999.25, 181.19999999999982, 'gini = 0.238\nsamples = 16\nvalue = [2
5, 4, 0]\nclass = a'),
 Text(3208.5, 543.5999999999999, 'NMHC <= 0.225\ngini = 0.091\nsamples = 
16\nvalue = [14, 505, 11]\nclass = b'),
 Text(3138.75, 181.19999999999982, 'gini = 0.574\nsamples = 15\nvalue = [1
1, 2, 9]\nclass = a'),
 Text(3278.25, 181.19999999999982, 'gini = 0.02\nsamples = 301\nvalue =
[3, 503, 92]\nclass = b'),
 Text(3906.0, 1268.4, 'BEN <= 1.165\ngini = 0.597\nsamples = 7317\nvalue =
[5312, 1380, 4934]\nclass = a'),
 Text(3627.0, 906.0, 'NMHC <= 0.085\ngini = 0.559\nsamples = 3903\nvalue =
[1693, 826, 3610]\nclass = c'),
 Text(3487.5, 543.5999999999999, 'BEN <= 0.565\ngini = 0.126\nsamples = 28
2\nvalue = [437, 11, 20]\nclass = a'),
 Text(3417.75, 181.19999999999982, 'gini = 0.244\nsamples = 102\nvalue =
[149, 4, 20]\nclass = a'),
 Text(3557.25, 181.19999999999982, 'gini = 0.046\nsamples = 180\nvalue =
[288, 7, 0]\nclass = a'),
 Text(3766.5, 543.5999999999999, 'OXY <= 0.625\ngini = 0.528\nsamples = 36
21\nvalue = [1256, 815, 3590]\nclass = c'),
 Text(3696.75, 181.19999999999982, 'gini = 0.621\nsamples = 711\nvalue =
[573, 283, 274]\nclass = a'),
 Text(3836.25, 181.19999999999982, 'gini = 0.428\nsamples = 2910\nvalue =
[683, 532, 3316]\nclass = c'),
 Text(4185.0, 906.0, 'PM10 <= 38.445\ngini = 0.498\nsamples = 3414\nvalue
= [3619, 554, 1324]\nclass = a'),
 Text(4045.5, 543.5999999999999, 'SO_2 <= 11.125\ngini = 0.407\nsamples =
1382\nvalue = [1666, 245, 314]\nclass = a'),
 Text(3975.75, 181.19999999999982, 'gini = 0.428\nsamples = 668\nvalue =
[805, 209, 93]\nclass = a'),
 Text(4115.25, 181.19999999999982, 'gini = 0.367\nsamples = 714\nvalue =
[861, 36, 221]\nclass = a'),
 Text(4324.5, 543.5999999999999, 'NOx <= 175.15\ngini = 0.54\nsamples = 20
32\nvalue = [1953, 309, 1010]\nclass = a')
```



Conclusion

Accuracy

Linear Regression:0.16331457098631952

Ridge Regression:0.16317654437433604

Lasso Regression:0.013732764982463452

ElasticNet Regression:0.0693172677037851

Logistic Regression:0.8146838030706512

Random Forest:0.8206625491297024

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

In [ ]: