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("2006.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.880000	97
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.430000	34
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.830000	28
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.990000	54
...	...	...	...	...	...	...	...	...	...	...	
230563	2006-05-01 00:00:00	5.88	0.83	6.23	NaN	0.20	112.500000	218.000000	NaN	24.389999	93
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29
230565	2006-05-01 00:00:00	0.96	NaN	0.69	NaN	0.19	135.100006	179.199997	NaN	11.460000	64
230566	2006-05-01 00:00:00	0.50	NaN	0.67	NaN	0.10	82.599998	105.599998	NaN	NaN	94
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52

230568 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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24758 non-null  object 
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	1.69	28079006
22	0.79	28079024
25	1.47	28079099
31	0.85	28079006
48	0.79	28079024
...	...	...
230538	0.40	28079024
230541	0.94	28079099
230547	1.06	28079006
230564	0.32	28079024
230567	0.74	28079099

24758 rows × 2 columns

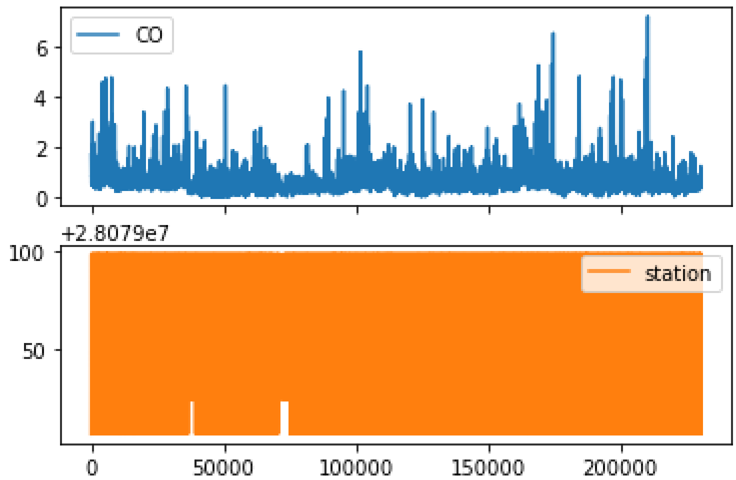
Line chart

In [7]:

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

Out[7]:

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



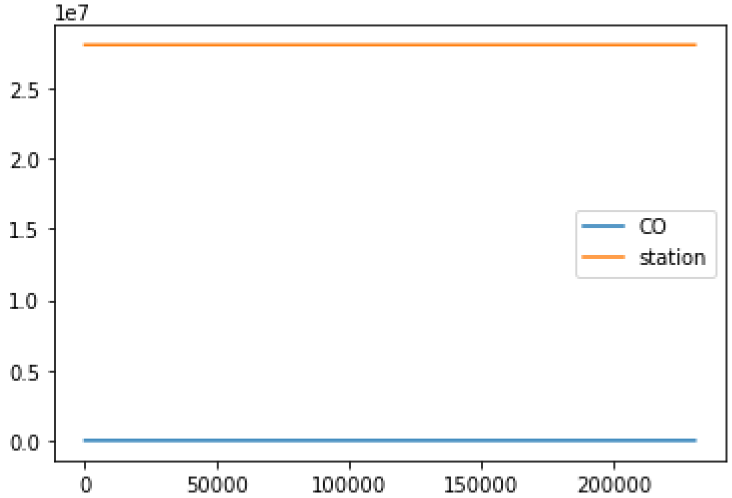
Line chart

In [8]:

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

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

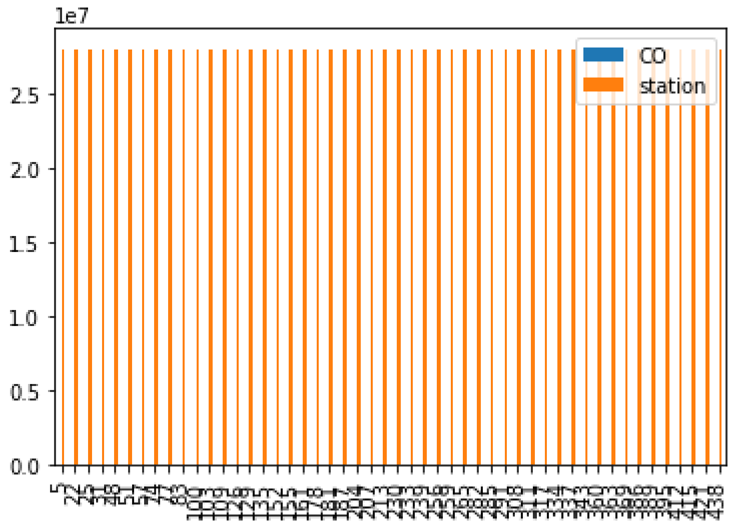
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<AxesSubplot:>



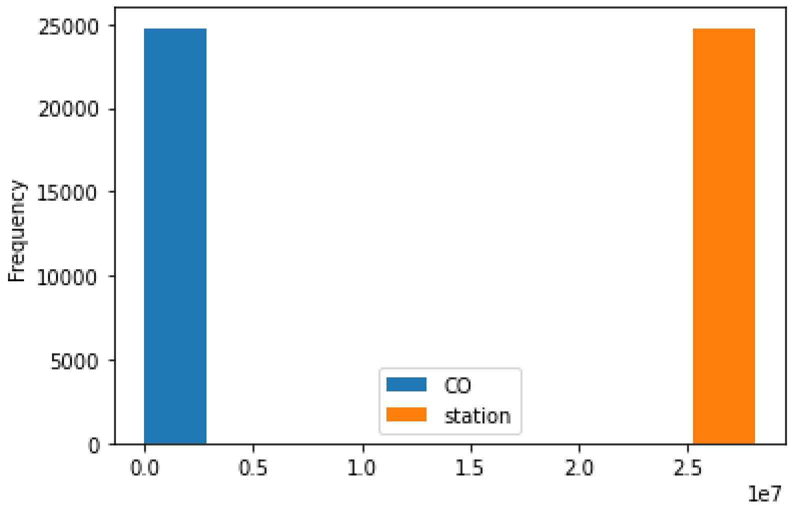
Histogram

In [11]:

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

Out[11]:

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



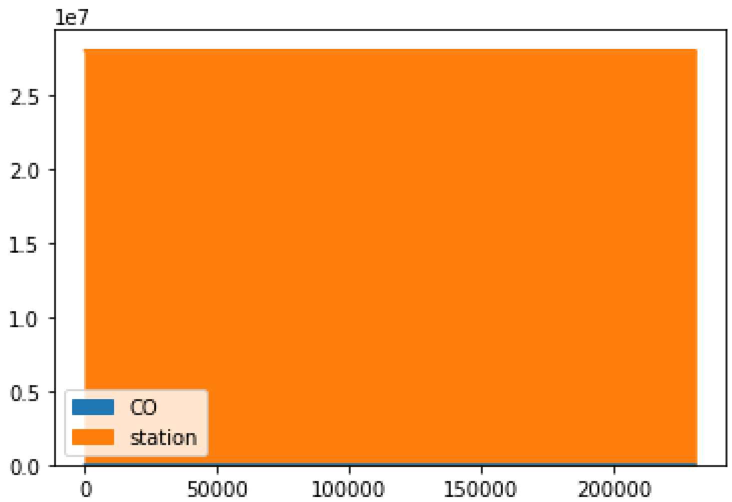
Area chart

In [12]:

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

Out[12]:

```
<AxesSubplot:>
```



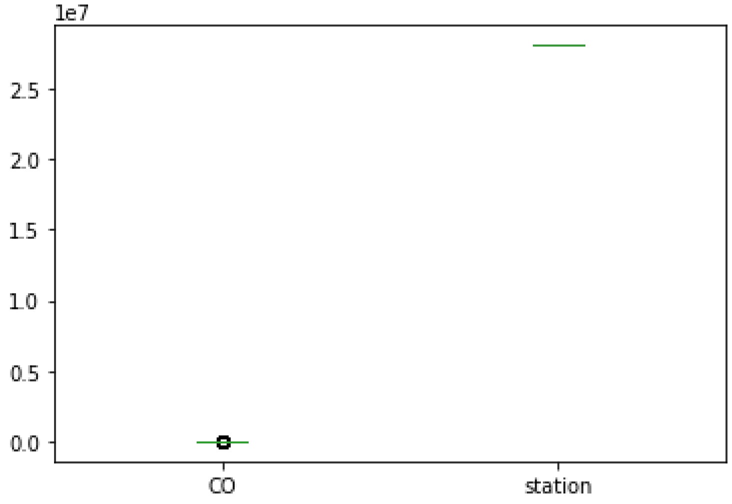
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



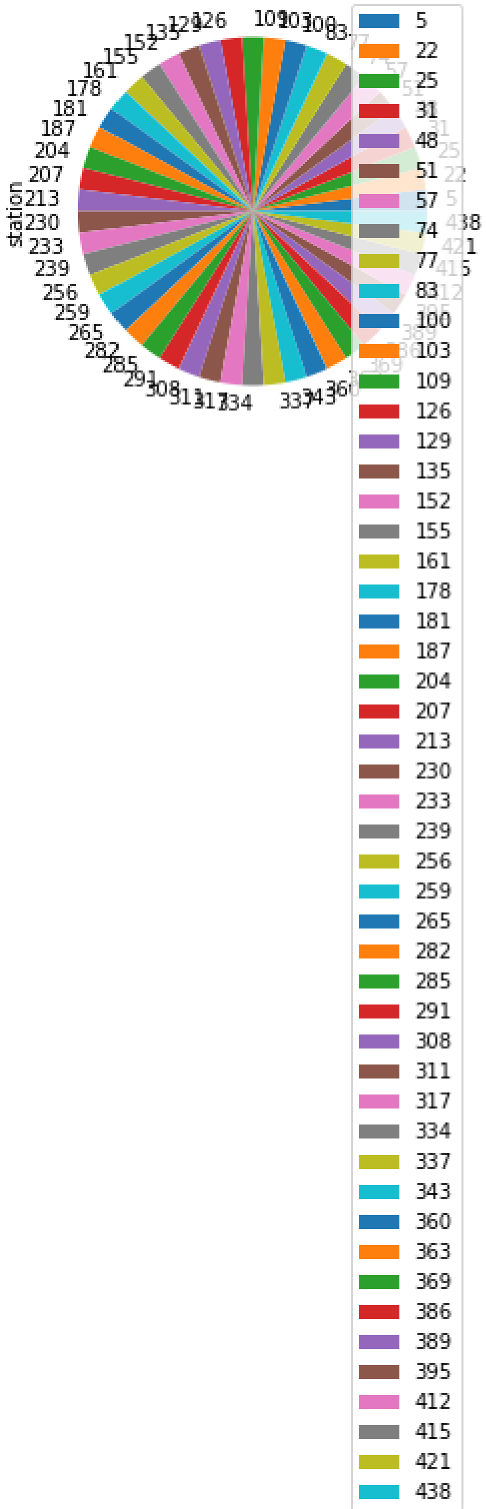
Pie chart

In [14]:

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

Out[14]:

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



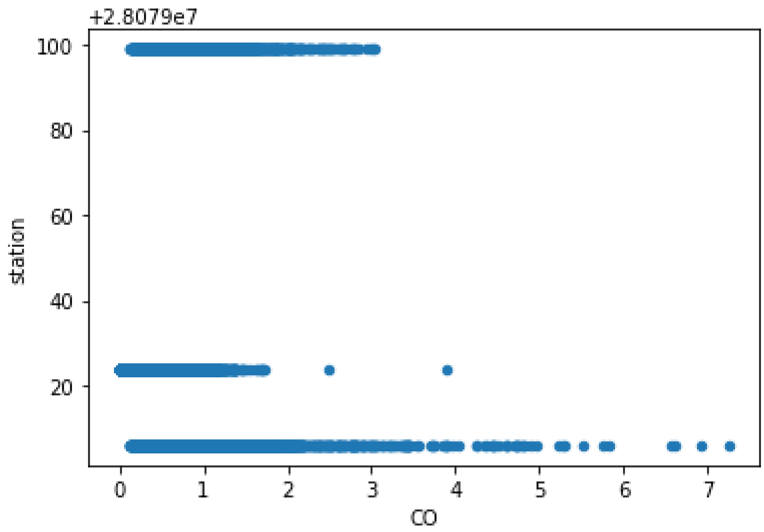
Scatter chart

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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     24758 non-null  object
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000
mean	1.350624	0.600713	1.824534	3.835034	0.176546	58.333481
std	1.541636	0.419048	1.868939	4.069036	0.126683	40.529382
min	0.110000	0.000000	0.170000	0.150000	0.000000	1.680000
25%	0.450000	0.360000	0.810000	1.060000	0.100000	28.450001
50%	0.850000	0.500000	1.130000	2.500000	0.150000	52.959999
75%	1.680000	0.720000	2.160000	5.090000	0.220000	79.347498
max	45.430000	7.250000	57.799999	66.900002	2.020000	461.299988

In [18]:

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

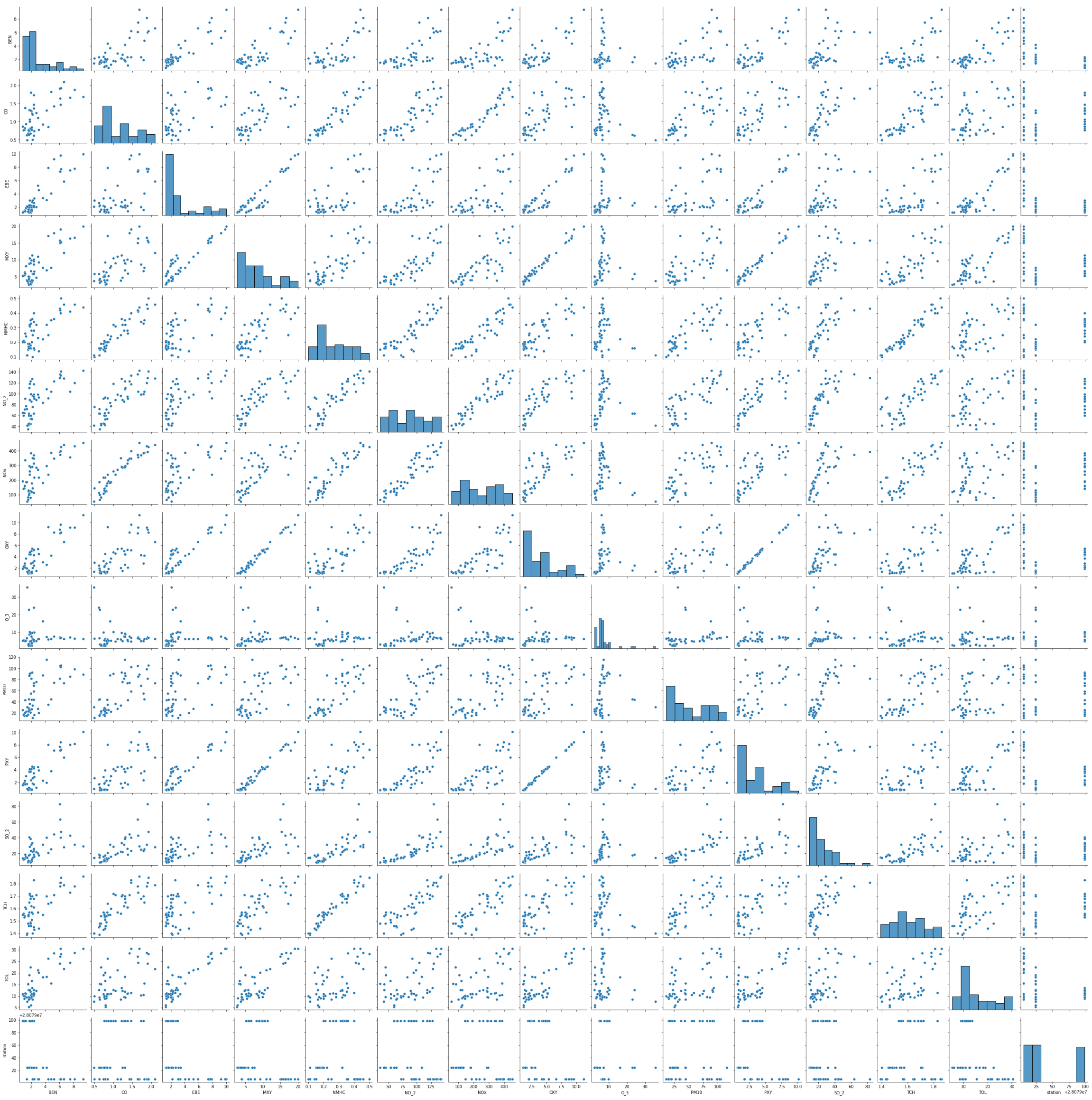
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x2d5b63f3f10>
```



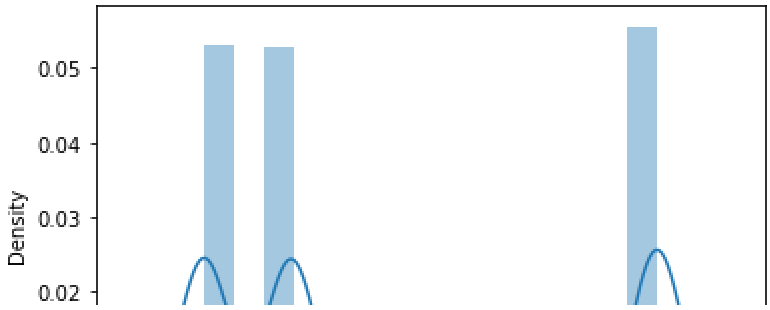
In [20]:

```
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'>

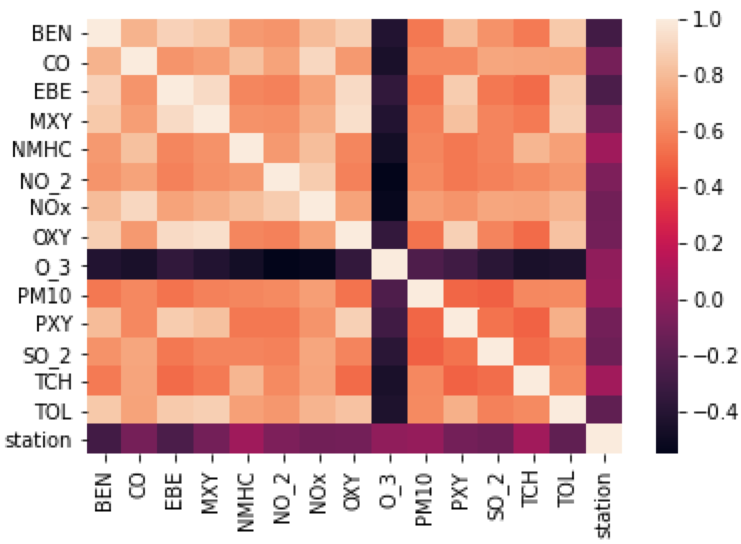


In [21]:

```
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]:

```
28079018.802358855
```

In [26]:

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

Out[26]:

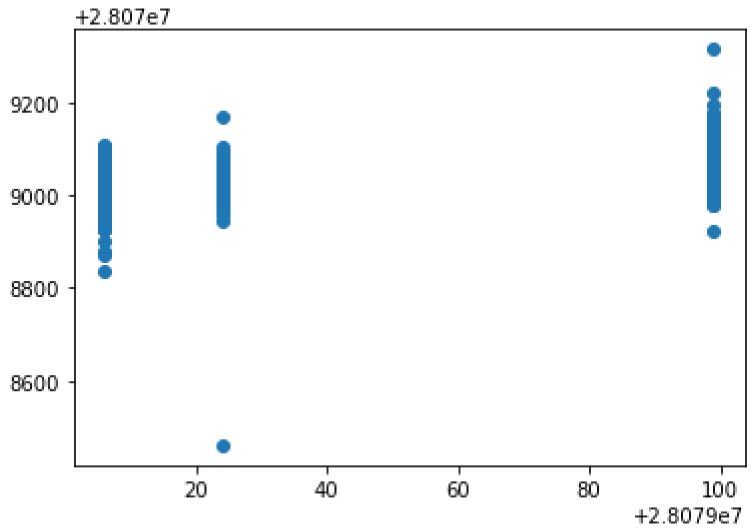
	Co-efficient
BEN	-19.415615
CO	-9.906295
EBE	-21.594129
MXY	4.013644
NMHC	126.094214
NO_2	-0.021909
NOx	0.000721
OXY	15.127191
O_3	-0.066023
PM10	0.138395
PXY	6.161678
SO_2	-0.623250
TCH	19.615309
TOL	-0.586153

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x2d5c4dc9b80>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.4012427537752702
```

In [29]:

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

Out[29]:

```
0.39005226870021625
```

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]:

```
0.3996835768174142
```

In [33]:

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

Out[33]:

```
0.3894233323360523
```

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]:

```
0.06375935096974827
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.05704880629900244
```

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([-8.6429254 ,  0.        , -8.58588669,  3.24227377,  0.41177302,
       -0.01034274,  0.00973438,  3.29408513, -0.13069957,  0.30630113,
        2.72111449, -0.4564863 ,  0.56287406, -1.09350568])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079052.300598077
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.23600965343781022
```

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)))
```

```
32.42264622331316
1272.2301177828479
35.66833494547857
```

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]:

```
(24758, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(24758,)
```

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]:

```
0.8741416915744405
```

In [54]:

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

Out[54]:

```
3.5557727473608076e-15
```

In [55]:

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

Out[55]:

```
array([[3.55577275e-15, 7.80743173e-29, 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]:

```
0.8748413156376227
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

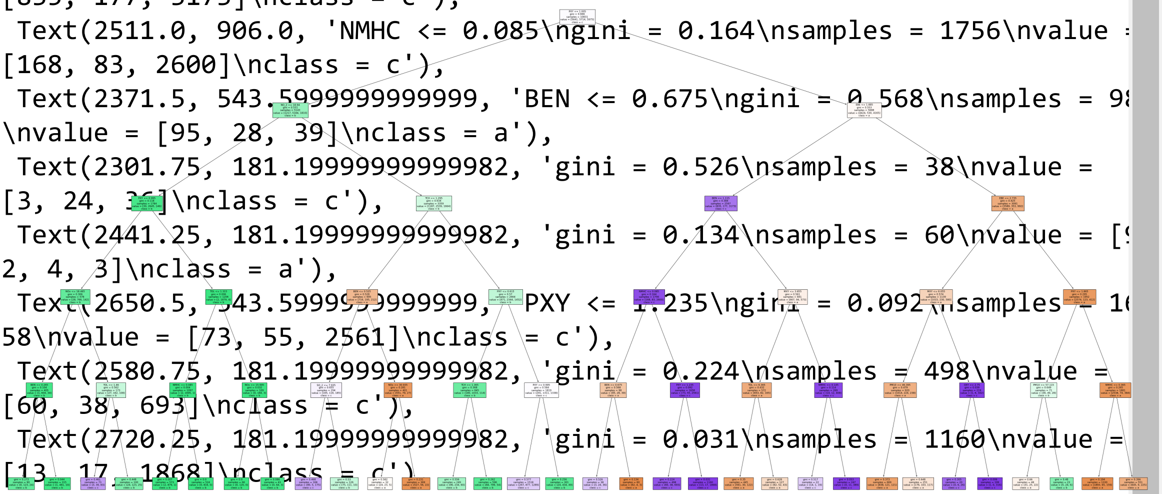
```
from sklearn.tree import plot_tree

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, 'PXY <= 1.005\ngini = 0.666\nsamples = 10831\nvalue
= [5641, 5714, 5975]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'NO_2 <= 18.94\ngini = 0.531\nsamples =
5143\nvalue = [1217, 5184, 1810]\nclass = b'),
 Text(558.0, 1268.4, 'PXY <= 0.995\ngini = 0.118\nsamples = 1784\nvalue =
[30, 2645, 146]\nclass = b'),
 Text(279.0, 906.0, 'NOx <= 18.465\ngini = 0.306\nsamples = 578\nvalue =
[28, 766, 142]\nclass = b'),
 Text(139.5, 543.5999999999999, 'BEN <= 0.265\ngini = 0.104\nsamples = 40
\nvalue = [1, 604, 34]\nclass = b'),
 Text(69.75, 181.19999999999982, 'gini = 0.273\nsamples = 90\nvalue = [0,
123, 24]\nclass = b'),
 Text(209.25, 181.19999999999982, 'gini = 0.044\nsamples = 315\nvalue =
[1, 481, 10]\nclass = b'),
 Text(418.5, 543.5999999999999, 'TOL <= 1.46\ngini = 0.562\nsamples = 173
\nvalue = [27, 162, 108]\nclass = b'),
 Text(348.75, 181.19999999999982, 'gini = 0.461\nsamples = 73\nvalue = [3
39, 83]\nclass = c'),
 Text(488.25, 181.19999999999982, 'gini = 0.448\nsamples = 100\nvalue = [
4, 123, 25]\nclass = b'),
 Text(837.0, 906.0, 'TOL <= 1.315\ngini = 0.006\nsamples = 1206\nvalue =
[2, 1879, 4]\nclass = b'),
 Text(697.5, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.004\nsamples = 1
97\nvalue = [2, 1695, 1]\nclass = b'),
 Text(627.75, 181.19999999999982, 'gini = 0.007\nsamples = 573\nvalue =
[2, 879, 1]\nclass = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.0\nsamples = 524\nvalue = [0,
816, 0]\nclass = b'),
 Text(976.5, 543.5999999999999, 'NOx <= 15.995\ngini = 0.032\nsamples = 1
9\nvalue = [0, 184, 3]\nclass = b'),
 Text(906.75, 181.19999999999982, 'gini = 0.0\nsamples = 66\nvalue = [0,
20, 0]\nclass = b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.086\nsamples = 43\nvalue =
[0, 64, 3]\nclass = b'),
 Text(1674.0, 1268.4, 'TCH <= 1.295\ngini = 0.634\nsamples = 3359\nvalue
[1187, 2539, 1664]\nclass = b'),
 Text(1395.0, 906.0, 'BEN <= 0.525\ngini = 0.528\nsamples = 695\nvalue =
[716, 195, 212]\nclass = a'),
 Text(1255.5, 543.5999999999999, 'SO_2 <= 7.935\ngini = 0.655\nsamples =
94\nvalue = [165, 116, 185]\nclass = c'),
 Text(1185.75, 181.19999999999982, 'gini = 0.468\nsamples = 168\nvalue =
[90, 5, 175]\nclass = c'),
 Text(1325.25, 181.19999999999982, 'gini = 0.53\nsamples = 126\nvalue = [
5, 111, 10]\nclass = b'),
 Text(1534.5, 543.5999999999999, 'NOx <= 30.935\ngini = 0.281\nsamples =
01\nvalue = [551, 79, 27]\nclass = a'),
 Text(1464.75, 181.19999999999982, 'gini = 0.582\nsamples = 32\nvalue = [
4, 23, 5]\nclass = a'),
 Text(1604.25, 181.19999999999982, 'gini = 0.231\nsamples = 369\nvalue =
[527, 56, 22]\nclass = a'),
 Text(1953.0, 906.0, 'PXY <= 0.615\ngini = 0.57\nsamples = 2664\nvalue =
[471, 2344, 1452]\nclass = b'),
 Text(1813.5, 543.5999999999999, 'TCH <= 1.365\ngini = 0.368\nsamples = 8
5\nvalue = [180, 1033, 114]\nclass = b'),
 Text(1743.75, 181.19999999999982, 'gini = 0.556\nsamples = 249\nvalue =
[96, 234, 60]\nclass = b'),
 Text(1883.25, 181.19999999999982, 'gini = 0.262\nsamples = 596\nvalue =
[84, 799, 54]\nclass = b'),
 Text(2092.5, 543.5999999999999, 'PXY <= 0.995\ngini = 0.584\nsamples = 1
19\nvalue = [291, 1311, 1338]\nclass = c'),
 Text(2022.75, 181.19999999999982, 'gini = 0.577\nsamples = 1518\nvalue =
```

```
[267, 877, 1289]\nclass = c'),
 Text(2162.25, 181.19999999999982, 'gini = 0.256\nsamples = 301\nvalue =
[24, 434, 49]\nclass = b'),
 Text(3348.0, 1630.8000000000002, 'EBE <= 1.885\ngini = 0.553\nsamples = 5
688\nvalue = [4424, 530, 4165]\nclass = a'),
 Text(2790.0, 1268.4, 'BEN <= 1.115\ngini = 0.384\nsamples = 2597\nvalue =
[835, 177, 3173]\nclass = c'),
 Text(2511.0, 906.0, 'NMHC <= 0.085\ngini = 0.164\nsamples = 1756\nvalue =
[168, 83, 2600]\nclass = c'),
 Text(2371.5, 543.5999999999999, 'BEN <= 0.675\ngini = 0.568\nsamples = 98
\nvalue = [95, 28, 39]\nclass = a'),
 Text(2301.75, 181.19999999999982, 'gini = 0.526\nsamples = 38\nvalue =
[3, 24, 36]\nclass = c'),
 Text(2441.25, 181.19999999999982, 'gini = 0.134\nsamples = 60\nvalue = [9
2, 4, 3]\nclass = a'),
 Text(2650.5, 543.5999999999999, 'PXY <= 1.235\ngini = 0.092\nsamples = 16
58\nvalue = [73, 55, 2561]\nclass = c'),
 Text(2580.75, 181.19999999999982, 'gini = 0.224\nsamples = 498\nvalue =
[60, 38, 693]\nclass = c'),
 Text(2720.25, 181.19999999999982, 'gini = 0.031\nsamples = 1160\nvalue =
[13, 17, 1868]\nclass = c'),
 Text(3069.0, 906.0, 'MXY <= 3.855\ngini = 0.561\nsamples = 841\nvalue =
[667, 94, 573]\nclass = a'),
 Text(2929.5, 543.5999999999999, 'TOL <= 6.365\ngini = 0.432\nsamples = 57
2\nvalue = [653, 82, 165]\nclass = a'),
 Text(2859.75, 181.19999999999982, 'gini = 0.35\nsamples = 445\nvalue = [5
61, 30, 122]\nclass = a'),
 Text(2999.25, 181.19999999999982, 'gini = 0.628\nsamples = 127\nvalue =
[92, 52, 43]\nclass = a'),
 Text(3208.5, 543.5999999999999, 'NMHC <= 0.145\ngini = 0.114\nsamples = 2
69\nvalue = [14, 12, 408]\nclass = c'),
 Text(3138.75, 181.19999999999982, 'gini = 0.517\nsamples = 22\nvalue = [1
4, 1, 19]\nclass = c'),
 Text(3278.25, 181.19999999999982, 'gini = 0.053\nsamples = 247\nvalue =
[0, 11, 389]\nclass = c'),
 Text(3906.0, 1268.4, 'EBE <= 2.735\ngini = 0.425\nsamples = 3091\nvalue =
[3589, 353, 992]\nclass = a'),
 Text(3627.0, 906.0, 'MXY <= 6.055\ngini = 0.574\nsamples = 1139\nvalue =
[1015, 230, 580]\nclass = a'),
 Text(3487.5, 543.5999999999999, 'PM10 <= 46.395\ngini = 0.479\nsamples =
920\nvalue = [1014, 224, 238]\nclass = a'),
 Text(3417.75, 181.19999999999982, 'gini = 0.373\nsamples = 669\nvalue =
[838, 121, 121]\nclass = a'),
 Text(3557.25, 181.19999999999982, 'gini = 0.648\nsamples = 251\nvalue =
[176, 103, 117]\nclass = a'),
 Text(3766.5, 543.5999999999999, 'OXY <= 2.75\ngini = 0.039\nsamples = 21
\nvalue = [1, 6, 342]\nclass = c'),
 Text(3696.75, 181.19999999999982, 'gini = 0.365\nsamples = 20\nvalue =
[0, 6, 26]\nclass = c'),
 Text(3836.25, 181.19999999999982, 'gini = 0.006\nsamples = 199\nvalue =
[1, 0, 316]\nclass = c'),
 Text(4185.0, 906.0, 'PXY <= 1.865\ngini = 0.295\nsamples = 1952\nvalue =
[2574, 123, 412]\nclass = a'),
 Text(4045.5, 543.5999999999999, 'PM10 <= 57.115\ngini = 0.649\nsamples =
71\nvalue = [36, 49, 28]\nclass = b'),
 Text(3975.75, 181.19999999999982, 'gini = 0.66\nsamples = 48\nvalue = [3
1, 25, 22]\nclass = a'),
 Text(4115.25, 181.19999999999982, 'gini = 0.48\nsamples = 23\nvalue = [5
24, 6]\nclass = b'),
 Text(4324.5, 543.5999999999999, 'NMHC <= 0.365\ngini = 0.265\nsamples = 1
881\nvalue = [2538, 74, 384]\nclass = a')
```



Conclusion

Accuracy

Linear Regression:0.40124275377752702

Ridge Regression:0.3996835768174142

Lasso Regression:0.3996835768174142

ElasticNet Regression:0.23600965343781022

Logistic Regression:0.8741416915744405

Random Forest:0.8748413156376227

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

In [0]: