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("2003.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.
...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.

243984 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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

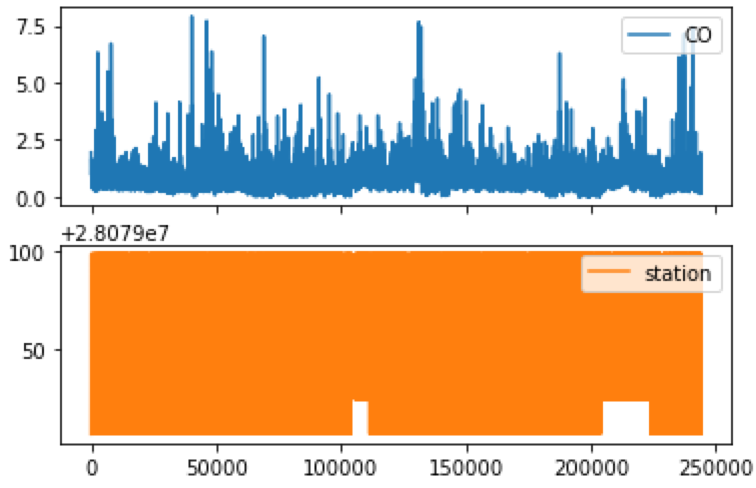
Line chart

In [7]:

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

Out[7]:

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



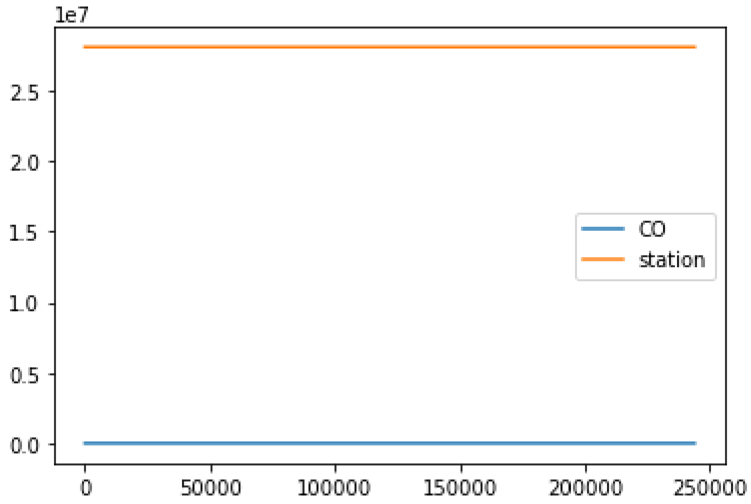
Line chart

In [8]:

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

Out[8]:

<AxesSubplot:>



Bar chart

In [9]:

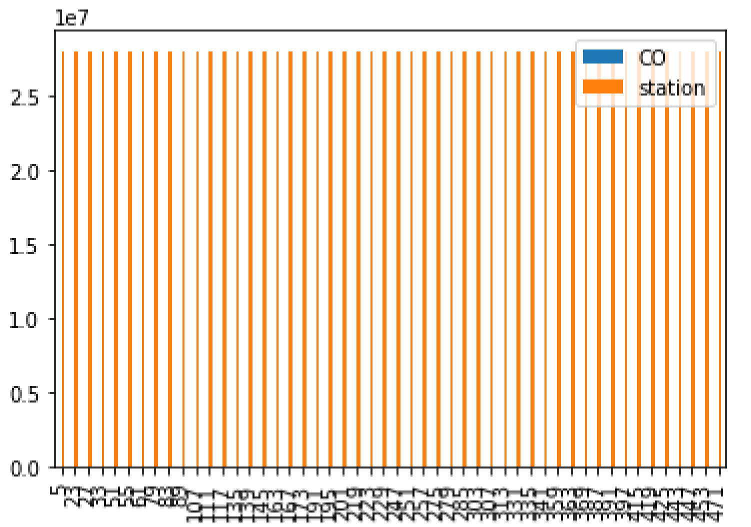
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<AxesSubplot:>



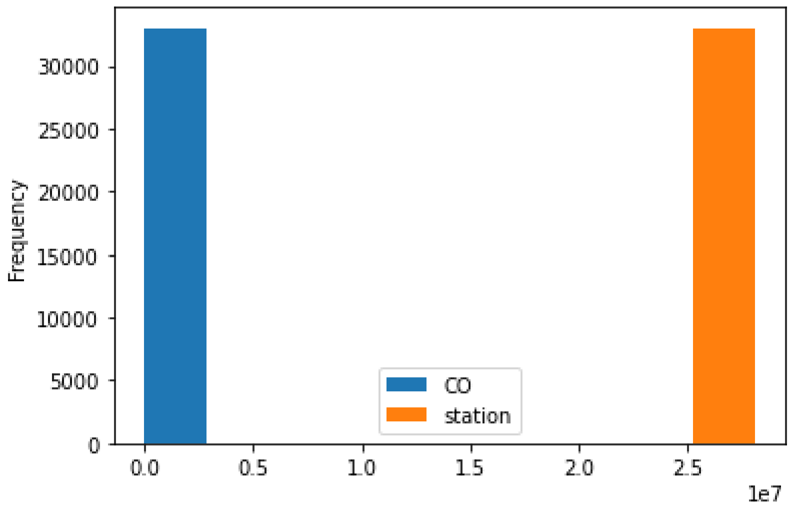
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



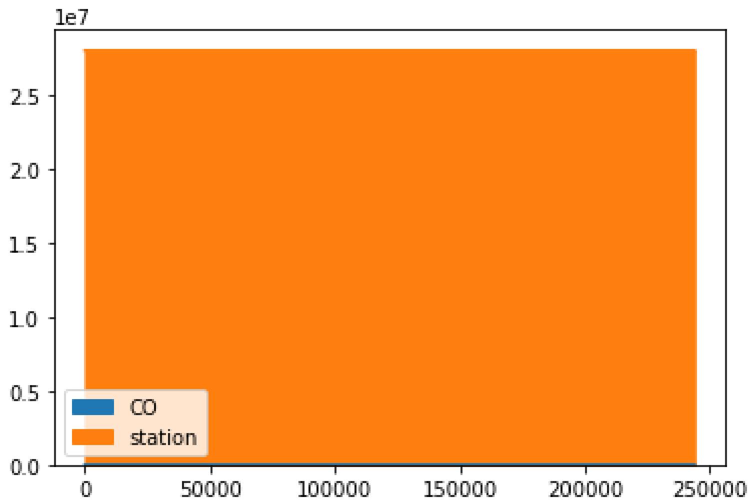
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



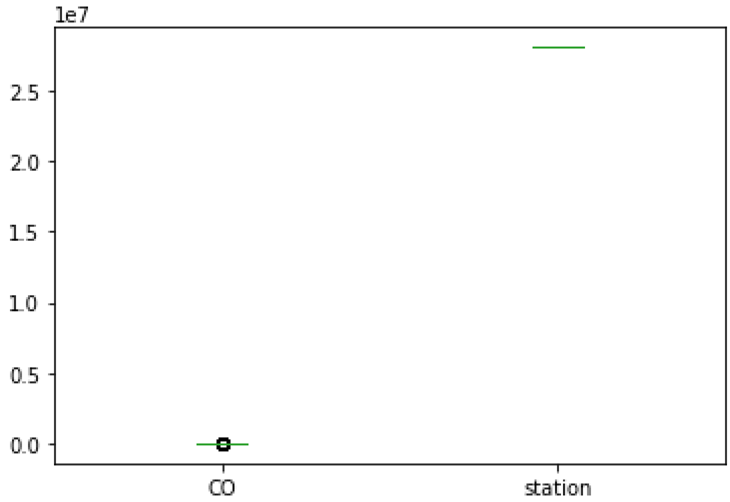
Box chart

In [13]:

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

Out[13]:

<AxesSubplot:>



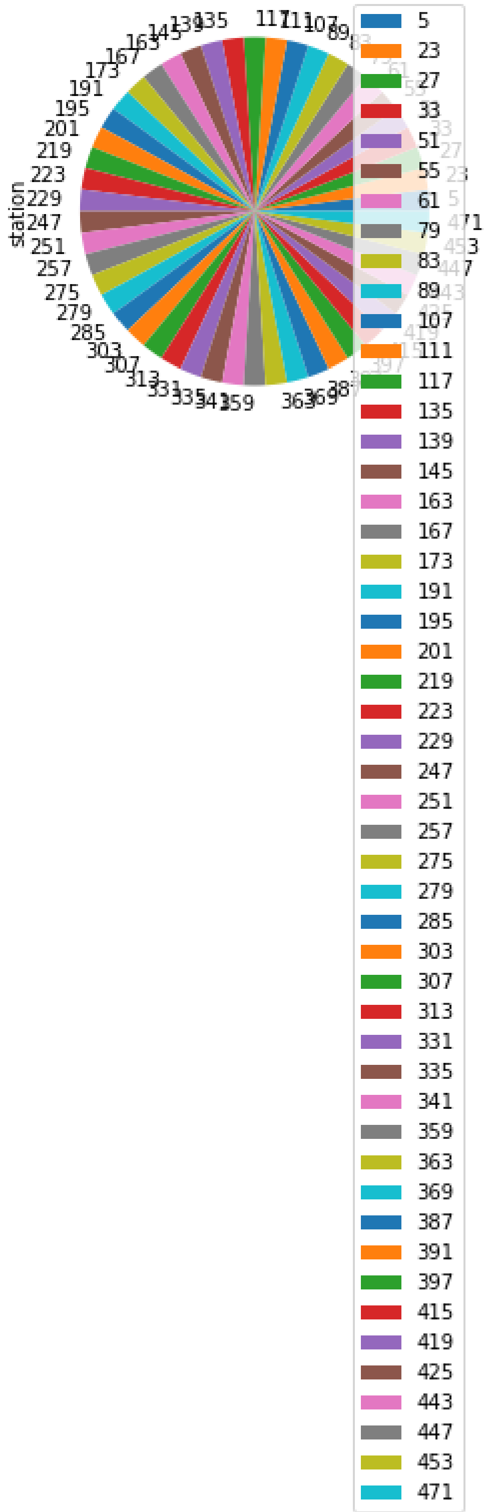
Pie chart

In [14]:

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

Out[14]:

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



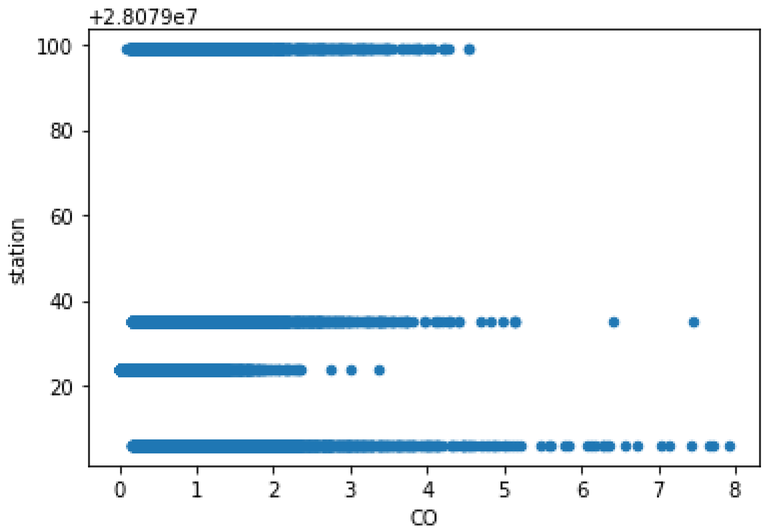
Scatter chart

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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     33010 non-null  object
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012

In [18]:

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

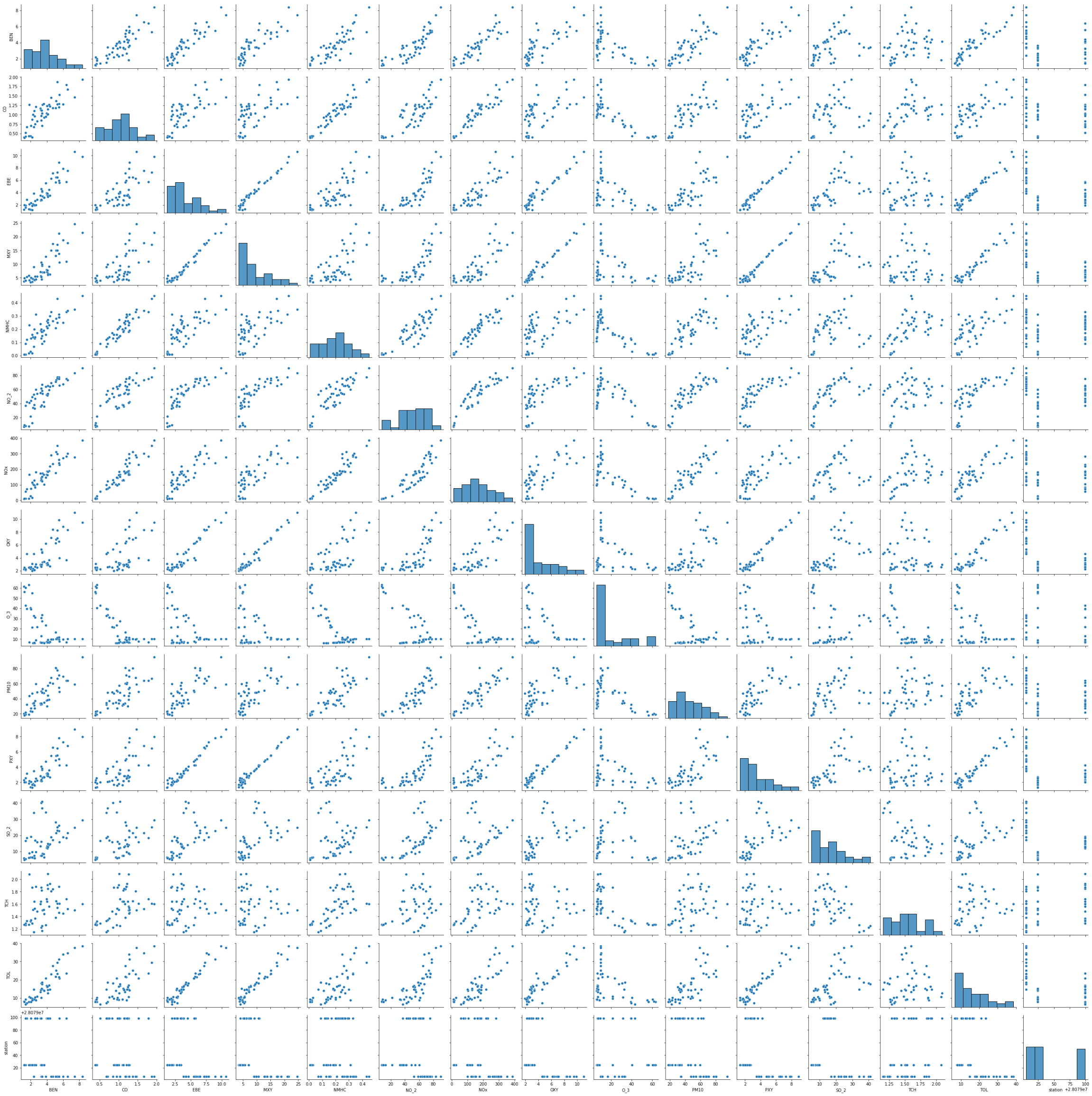
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x23a3dac3310>



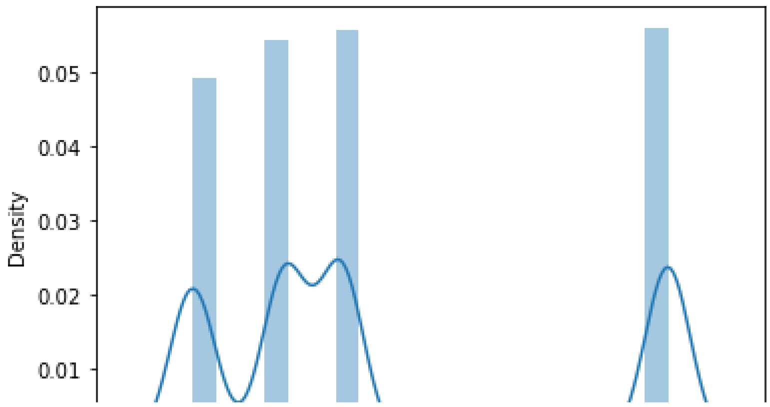
In [20]:

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

```
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]:

```
&lt;AxesSubplot:xlabel='station', ylabel='Density'&gt;
```

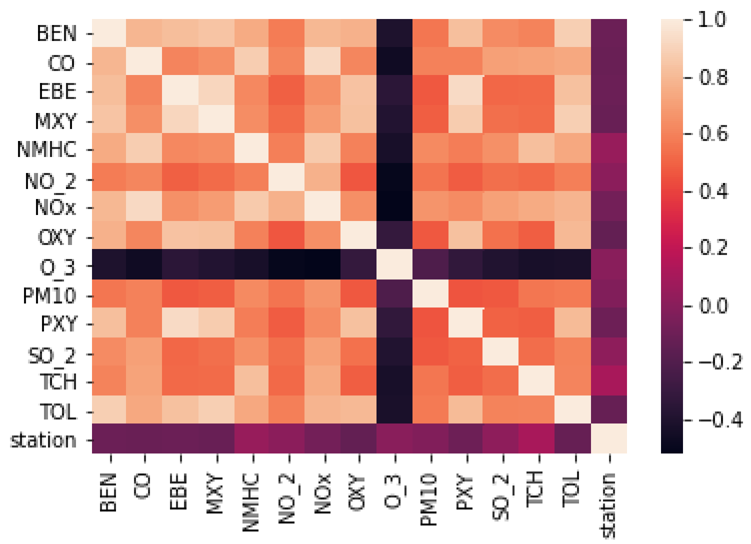


In [21]:

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

Out[21]:

```
&lt;AxesSubplot:&gt;
```



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]:

```
28079000.956622433
```

In [26]:

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

Out[26]:

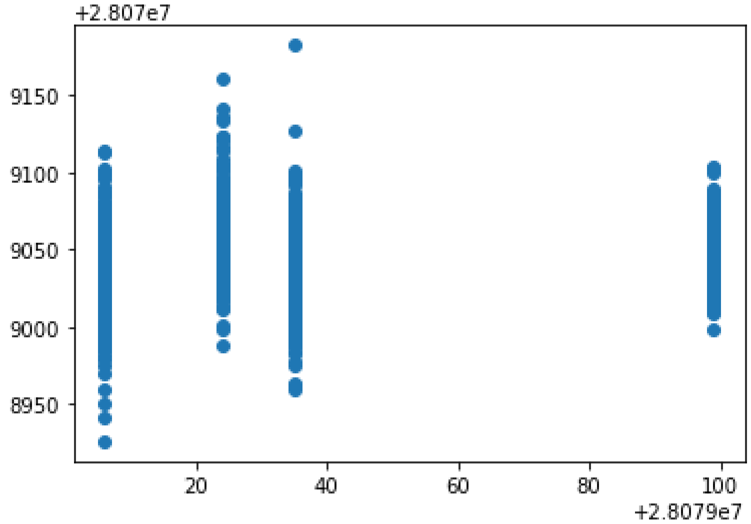
	Co-efficient
BEN	1.435520
CO	-40.195568
EBE	-1.718343
MXY	0.167218
NMHC	157.085159
NO_2	0.165226
NOx	-0.069149
OXY	-1.123471
O_3	-0.010631
PM10	-0.061677
PXY	1.653670
SO_2	0.859409
TCH	36.062127
TOL	-0.873464

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x23a4d7a2670>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.1665159379212826
```

In [29]:

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

Out[29]:

```
0.18001211504197578
```

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]:

```
0.16649970610262066
```

In [33]:

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

Out[33]:

```
0.17891931754207202
```

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]:

```
0.03581429332066366
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.03428948704282486
```

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([-0.        , -0.30198222,  0.        , -0.        ,  0.1508955 ,
        0.1595836 , -0.07058833, -1.16189269, -0.0381492 ,  0.06219809,
        0.28440877,  0.75964187,  1.61387558, -0.44919497])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079037.37589473
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.04941857713580211
```

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)))
```

```
28.823269020775424
1156.459493859106
34.006756591287946
```

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]:

```
(33010, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(33010,)
```

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]:

```
0.7584974250227204
```

In [54]:

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

Out[54]:

```
2.3306153265290618e-23
```

In [55]:

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

Out[55]:

```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457491e-16]])
```

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]:

```
0.7293464034866008
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

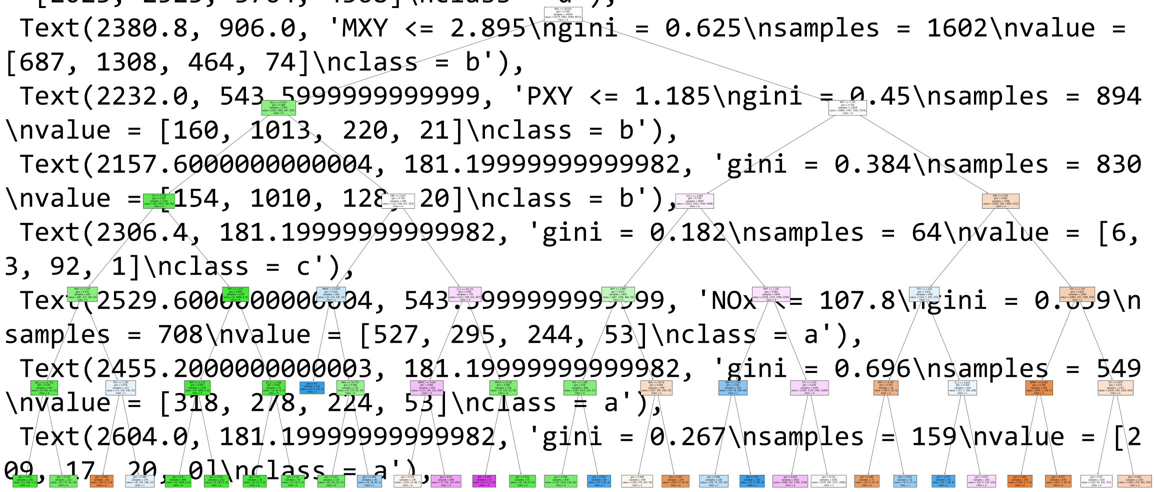
```
from sklearn.tree import plot_tree

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(2166.9, 1993.2, 'NOx <= 39.615\ngini = 0.749\nsamples = 14618\nvalue
= [5175, 5852, 6009, 6071]\nclass = d'),
 Text(1060.2, 1630.8000000000002, 'SO_2 <= 6.335\ngini = 0.489\nsamples =
2790\nvalue = [193, 3065, 647, 552]\nclass = b'),
 Text(595.2, 1268.4, 'CO <= 0.335\ngini = 0.205\nsamples = 1794\nvalue =
[82, 2557, 190, 49]\nclass = b'),
 Text(297.6, 906.0, 'MXY <= 1.275\ngini = 0.475\nsamples = 635\nvalue = [8
0, 713, 190, 43]\nclass = b'),
 Text(148.8, 543.5999999999999, 'NO_2 <= 11.755\ngini = 0.197\nsamples = 4
14\nvalue = [9, 593, 41, 21]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.025\nsamples = 192\nvalue = [1,
316, 1, 2]\nclass = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.334\nsamples = 222
\nvalue = [8, 277, 40, 19]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'TCH <= 1.195\ngini = 0.679\n
samples = 221\nvalue = [71, 120, 149, 22]\nclass = c'),
 Text(372.0, 181.19999999999982, 'gini = 0.087\nsamples = 45\nvalue = [63,
0, 3, 0]\nclass = a'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.586\nsamples = 176
\nvalue = [8, 120, 146, 22]\nclass = c'),
 Text(892.8000000000001, 906.0, 'PXY <= 1.215\ngini = 0.009\nsamples = 115
9\nvalue = [2, 1844, 0, 6]\nclass = b'),
 Text(744.0, 543.5999999999999, 'MXY <= 1.115\ngini = 0.007\nsamples = 112
8\nvalue = [0, 1804, 0, 6]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.0\nsamples = 906\nvalue = [0, 1
449, 0, 0]\nclass = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.033\nsamples = 222
\nvalue = [0, 355, 0, 6]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'SO_2 <= 4.88\ngini = 0.091\n
samples = 31\nvalue = [2, 40, 0, 0]\nclass = b'),
 Text(967.2, 181.19999999999982, 'gini = 0.0\nsamples = 16\nvalue = [0, 2
2, 0, 0]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.18\nsamples = 15\nvalue = [2,
18, 0, 0]\nclass = b'),
 Text(1525.2, 1268.4, 'EBE <= 0.615\ngini = 0.706\nsamples = 996\nvalue =
[111, 508, 457, 503]\nclass = b'),
 Text(1264.8000000000002, 906.0, 'NMHC <= 0.035\ngini = 0.524\nsamples = 2
62\nvalue = [0, 170, 235, 16]\nclass = c'),
 Text(1190.4, 543.5999999999999, 'gini = 0.0\nsamples = 111\nvalue = [0,
0, 172, 0]\nclass = c'),
 Text(1339.2, 543.5999999999999, 'NOx <= 32.775\ngini = 0.466\nsamples = 1
51\nvalue = [0, 170, 63, 16]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.313\nsamples = 115
\nvalue = [0, 152, 23, 11]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.509\nsamples = 36
\nvalue = [0, 18, 40, 5]\nclass = c'),
 Text(1785.6000000000001, 906.0, 'O_3 <= 95.375\ngini = 0.692\nsamples = 7
34\nvalue = [111, 338, 222, 487]\nclass = d'),
 Text(1636.8000000000002, 543.5999999999999, 'NMHC <= 0.025\ngini = 0.683
\nsamples = 628\nvalue = [111, 204, 212, 454]\nclass = d'),
 Text(1562.4, 181.19999999999982, 'gini = 0.578\nsamples = 139\nvalue = [1
10, 13, 98, 7]\nclass = a'),
 Text(1711.2, 181.19999999999982, 'gini = 0.56\nsamples = 489\nvalue = [1,
191, 114, 447]\nclass = d'),
 Text(1934.4, 543.5999999999999, 'PM10 <= 21.82\ngini = 0.389\nsamples = 1
06\nvalue = [0, 134, 10, 33]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.147\nsamples = 15
\nvalue = [0, 0, 2, 23]\nclass = d'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.216\nsamples = 91
\nvalue = [0, 134, 8, 10]\nclass = b'),
 Text(3273.6000000000004, 1630.8000000000002, 'MXY <= 7.255\ngini = 0.736
```

```
\nsamples = 11828\nvalue = [4982, 2787, 5362, 5519]\nclass = d'),
 Text(2678.4, 1268.4, 'SO_2 <= 6.695\ngini = 0.728\nsamples = 8060\nvalue
= [2023, 2523, 3764, 4368]\nclass = d'),
 Text(2380.8, 906.0, 'MXY <= 2.895\ngini = 0.625\nsamples = 1602\nvalue =
[687, 1308, 464, 74]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'PXY <= 1.185\ngini = 0.45\nsamples = 894
\nvalue = [160, 1013, 220, 21]\nclass = b'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.384\nsamples = 830
\nvalue = [154, 1010, 128, 20]\nclass = b'),
 Text(2306.4, 181.19999999999982, 'gini = 0.182\nsamples = 64\nvalue = [6,
3, 92, 1]\nclass = c'),
 Text(2529.6000000000004, 543.5999999999999, 'NOx <= 107.8\ngini = 0.659\n
samples = 708\nvalue = [527, 295, 244, 53]\nclass = a'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.696\nsamples = 549
\nvalue = [318, 278, 224, 53]\nclass = a'),
 Text(2604.0, 181.19999999999982, 'gini = 0.267\nsamples = 159\nvalue = [2
09, 17, 20, 0]\nclass = a'),
 Text(2976.0, 906.0, 'MXY <= 1.105\ngini = 0.683\nsamples = 6458\nvalue =
[1336, 1215, 3300, 4294]\nclass = d'),
 Text(2827.2000000000003, 543.5999999999999, 'EBE <= 1.055\ngini = 0.541\n
samples = 354\nvalue = [76, 76, 359, 46]\nclass = c'),
 Text(2752.8, 181.19999999999982, 'gini = 0.681\nsamples = 229\nvalue = [7
6, 73, 170, 43]\nclass = c'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.06\nsamples = 125
\nvalue = [0, 3, 189, 3]\nclass = c'),
 Text(3124.8, 543.5999999999999, 'CO <= 0.725\ngini = 0.678\nsamples = 610
4\nvalue = [1260, 1139, 2941, 4248]\nclass = d'),
 Text(3050.4, 181.19999999999982, 'gini = 0.635\nsamples = 3922\nvalue =
[686, 552, 1764, 3158]\nclass = d'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.724\nsamples = 218
2\nvalue = [574, 587, 1177, 1090]\nclass = c'),
 Text(3868.8, 1268.4, 'EBE <= 3.255\ngini = 0.644\nsamples = 3768\nvalue =
[2959, 264, 1598, 1151]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'TCH <= 1.325\ngini = 0.631\nsamples = 40
8\nvalue = [119, 1, 292, 233]\nclass = c'),
 Text(3422.4, 543.5999999999999, 'MXY <= 8.165\ngini = 0.435\nsamples = 92
\nvalue = [98, 0, 36, 5]\nclass = a'),
 Text(3348.000000000005, 181.19999999999982, 'gini = 0.284\nsamples = 71
\nvalue = [92, 0, 19, 0]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.554\nsamples = 21\nvalue = [6,
0, 17, 5]\nclass = c'),
 Text(3720.0000000000005, 543.5999999999999, 'O_3 <= 6.415\ngini = 0.539\n
samples = 316\nvalue = [21, 1, 256, 228]\nclass = c'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.219\nsamples = 53
\nvalue = [2, 1, 80, 8]\nclass = c'),
 Text(3794.4, 181.19999999999982, 'gini = 0.537\nsamples = 261\nvalue = [1
9, 0, 176, 220]\nclass = d'),
 Text(4166.400000000001, 906.0, 'TCH <= 1.335\ngini = 0.624\nsamples = 336
0\nvalue = [2840, 263, 1306, 918]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'NMHC <= 0.055\ngini = 0.196
\nsamples = 639\nvalue = [910, 5, 102, 4]\nclass = a'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.065\nsamples = 165
```



Conclusion

Accuracy

Linear Regression:0.18001211504197578

Ridge Regression:0.03581429332066366

Lasso Regression:0.03428948704282486

ElasticNet Regression:0.04941857713580211

Logistic Regression:0.8480899292795158

Random Forest:0.7293464034866008

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