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("2004.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2004-08-01 01:00:00	NaN	0.66	NaN	NaN	NaN	89.550003	118.900002	NaN	40.020000	39
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22
2	2004-08-01 01:00:00	NaN	1.02	NaN	NaN	NaN	93.389999	138.600006	NaN	20.860001	49
3	2004-08-01 01:00:00	NaN	0.53	NaN	NaN	NaN	87.290001	105.000000	NaN	36.730000	31
4	2004-08-01 01:00:00	NaN	0.17	NaN	NaN	NaN	34.910000	35.349998	NaN	86.269997	54
...	...	...	...	...	...	...	...	...	...	...	
245491	2004-06-01 00:00:00	0.75	0.21	0.85	1.55	0.07	59.580002	64.389999	0.66	33.029999	30
245492	2004-06-01 00:00:00	2.49	0.75	2.44	4.57	NaN	97.139999	146.899994	2.34	7.740000	37
245493	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.13	102.699997	132.600006	NaN	17.809999	22
245494	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.09	82.599998	102.599998	NaN	NaN	45
245495	2004-06-01 00:00:00	3.01	0.67	2.78	5.12	0.20	92.550003	141.000000	2.60	11.460000	24

245496 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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     19397 non-null  object 
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	0.63	28079006
22	0.36	28079024
26	0.46	28079099
32	0.67	28079006
49	0.30	28079024
...	...	...
245463	0.08	28079024
245467	0.67	28079099
245473	1.12	28079006
245491	0.21	28079024
245495	0.67	28079099

19397 rows × 2 columns

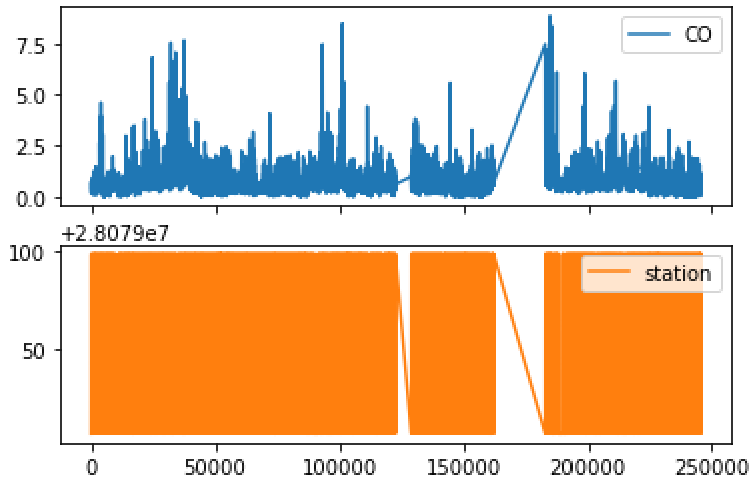
Line chart

In [7]:

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

Out[7]:

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



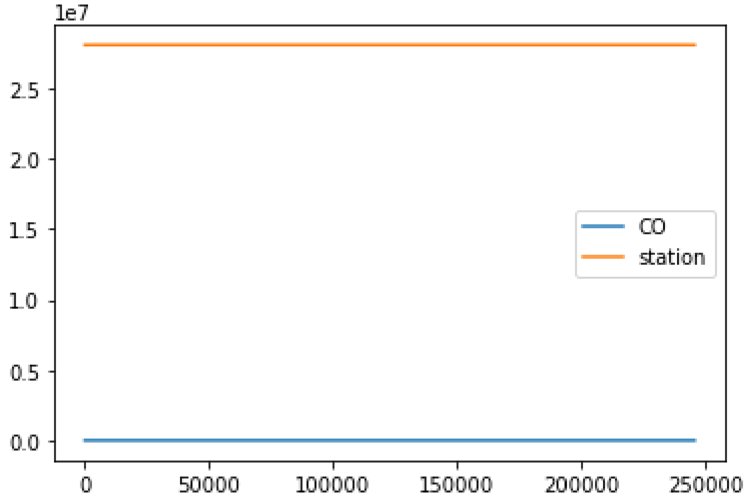
Line chart

In [8]:

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

Out[8]:

```
<AxesSubplot:>
```



Bar chart

In [9]:

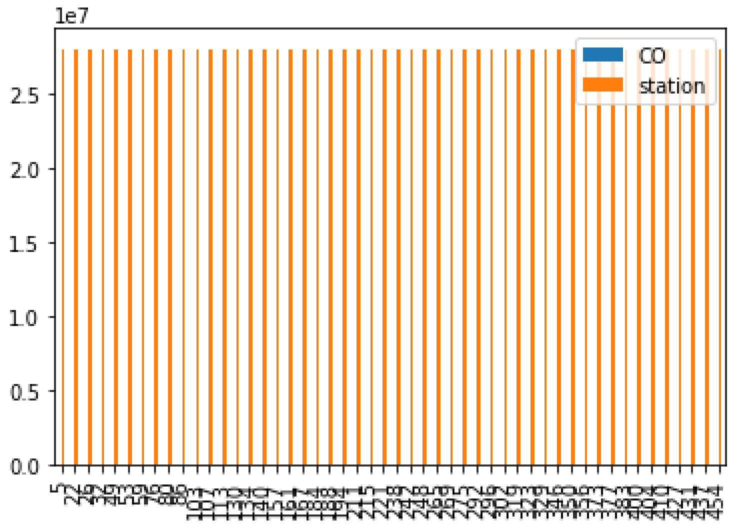
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<AxesSubplot:>
```



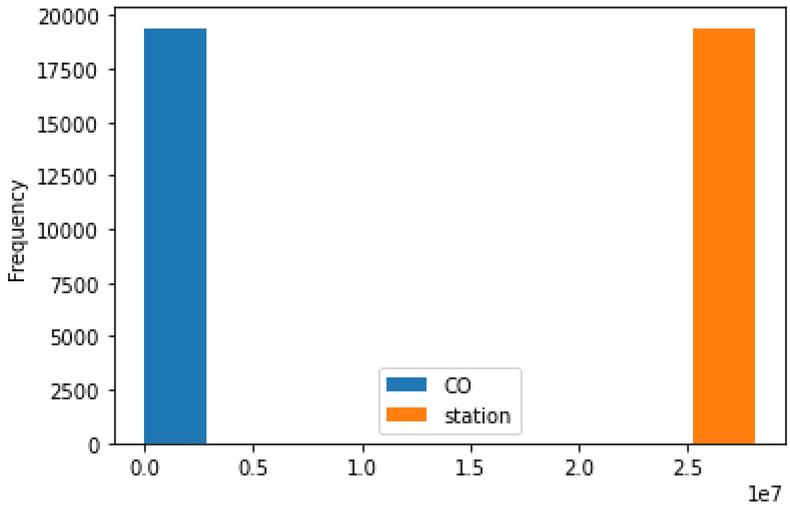
Histogram

In [11]:

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

Out[11]:

<AxesSubplot:ylabel='Frequency'>



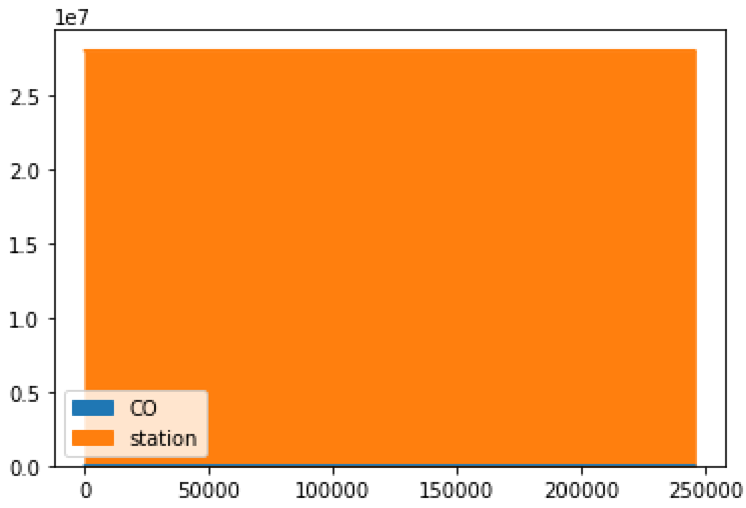
Area chart

In [12]:

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

Out[12]:

<AxesSubplot:>



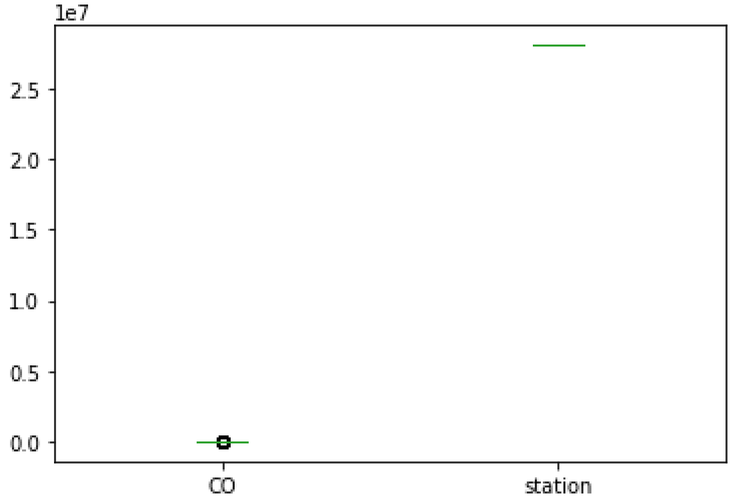
Box chart

In [13]:

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

Out[13]:

```
<AxesSubplot:>
```



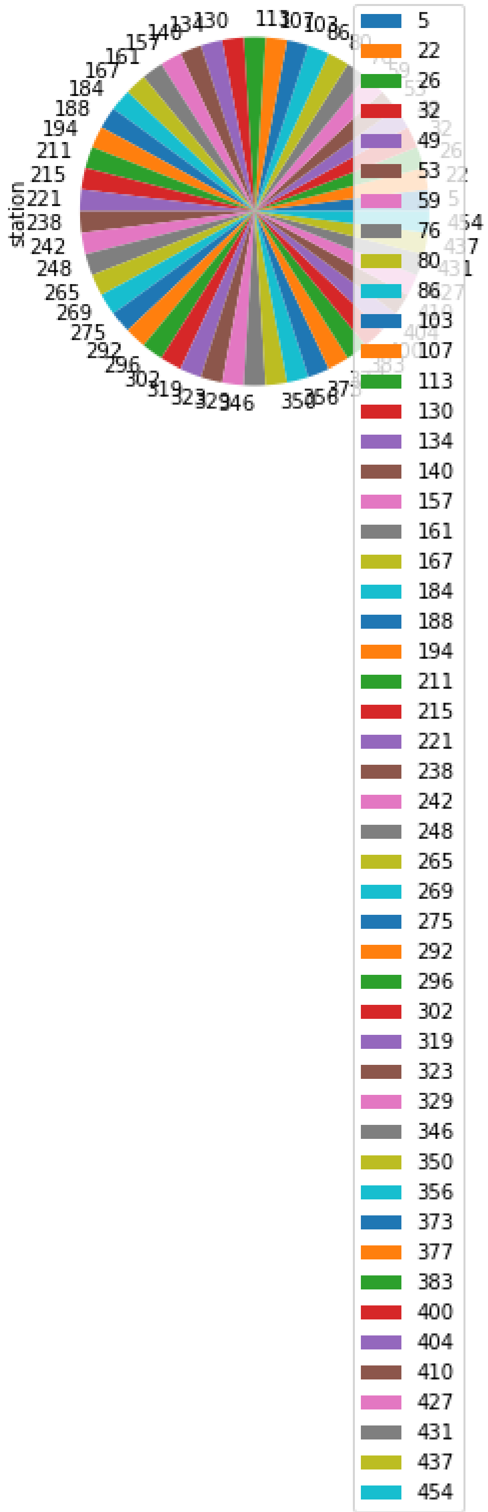
Pie chart

In [14]:

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

Out[14]:

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



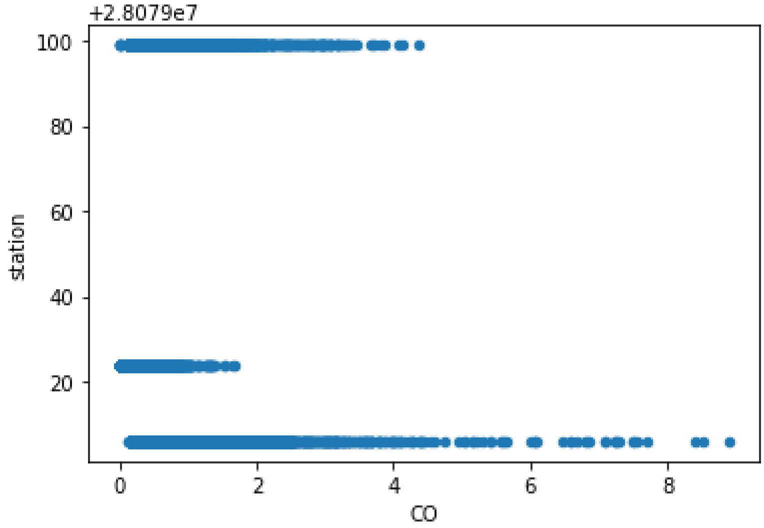
Scatter chart

In [15]:

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

Out[15]:

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



In [16]:

```
df.info()
```

```
 0   date     19397 non-null  object
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2
count	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000
mean	2.250781	0.675347	2.775913	5.424809	0.151024	62.887023
std	2.184724	0.591026	2.729622	5.554358	0.158603	37.952255
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.090000
25%	0.870000	0.320000	1.020000	1.780000	0.060000	35.150002
50%	1.620000	0.520000	1.970000	3.800000	0.110000	58.310001
75%	2.910000	0.860000	3.580000	7.260000	0.200000	85.730003
max	34.180000	8.900000	41.880001	91.599998	4.810000	355.100006

In [18]:

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

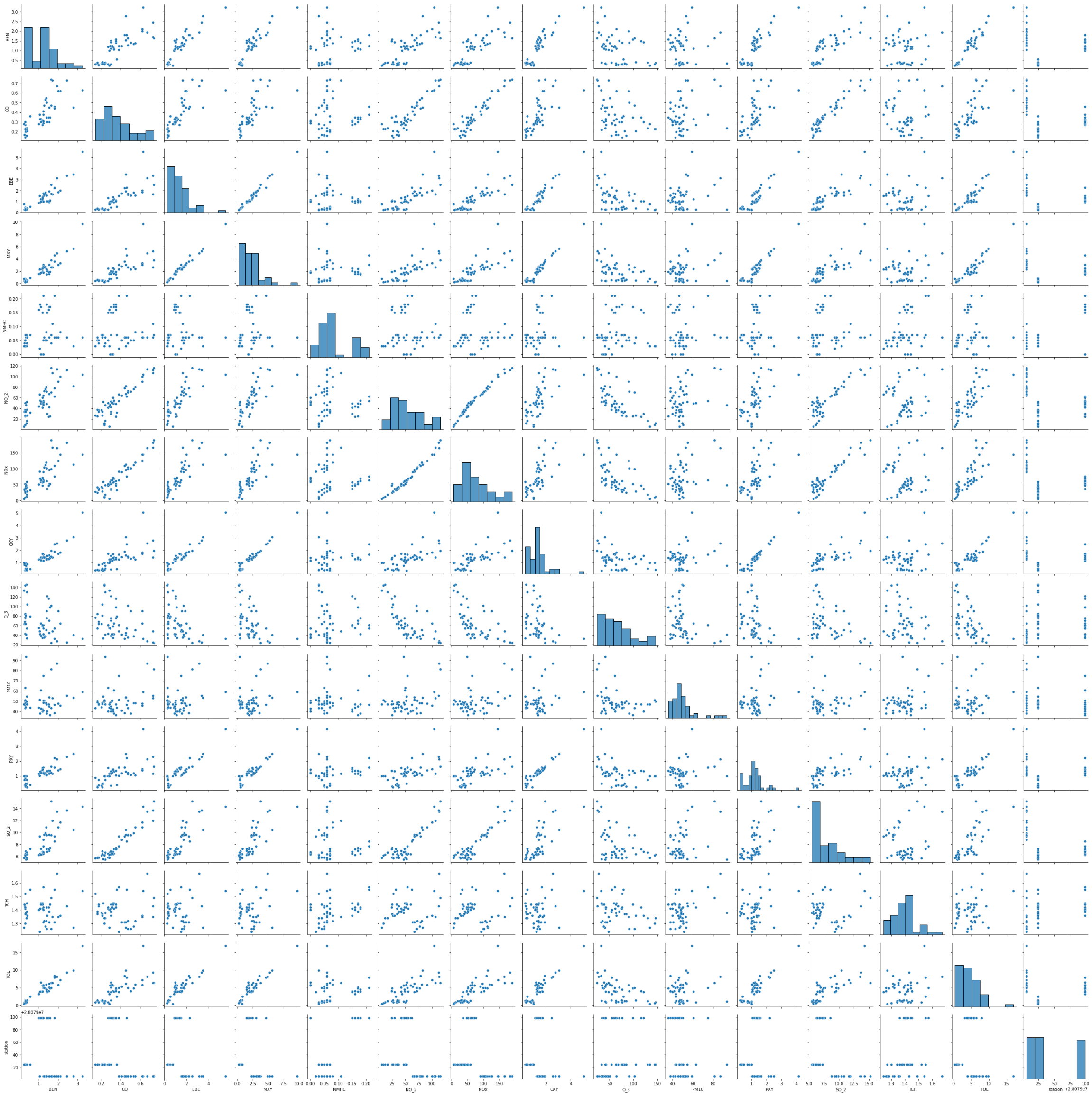
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x14897c308b0>
```



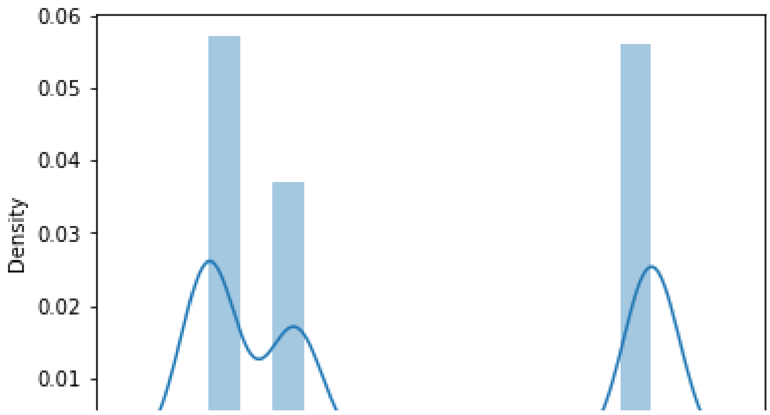
In [20]:

```
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]:

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

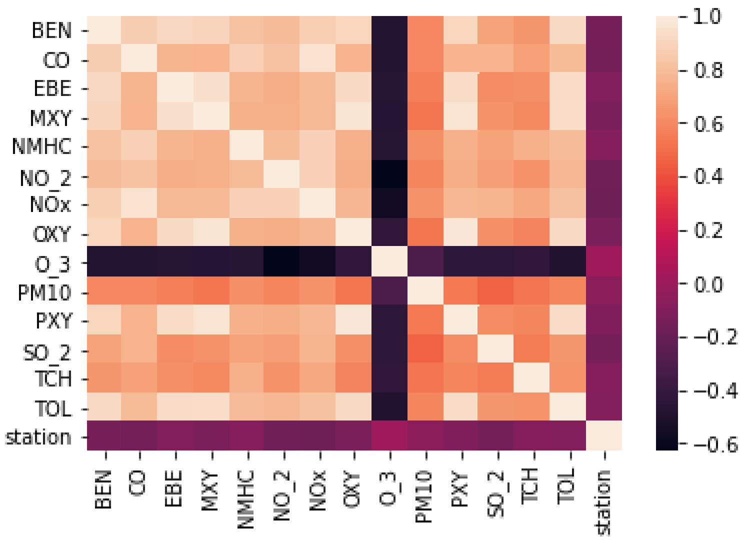


In [21]:

```
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]:

```
28079078.038366888
```

In [26]:

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

Out[26]:

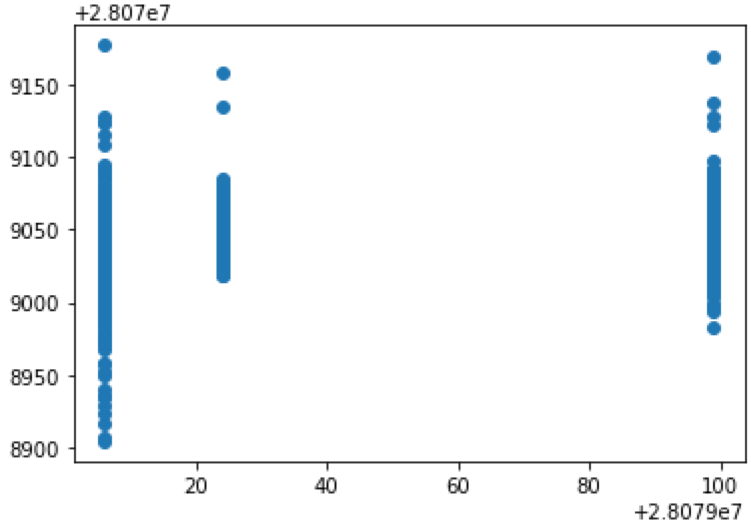
	Co-efficient
BEN	-4.904216
CO	28.448162
EBE	4.273867
MXY	-3.543865
NMHC	81.109230
NO_2	-0.155101
NOx	-0.264007
OXY	-2.021141
O_3	-0.290927
PM10	0.067048
PXY	5.557141
SO_2	-0.161654
TCH	-8.688692
TOL	1.356833

In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x148a66c47f0>
```



ACCURACY

In [28]:

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

Out[28]:

```
0.09590137311518288
```

In [29]:

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

Out[29]:

```
0.11045993310581825
```

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]:

0.09408174881015519

In [33]:

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

Out[33]:

0.11011845641033136

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]:

0.05790297062460381

Accuracy(Lasso)

In [36]:

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

Out[36]:

0.044541624988104433

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([-0.28263423,  0.41220025,  1.45157432, -1.92392772,  0.        ,
       -0.16950735, -0.0980621 , -0.        , -0.22385107,  0.09147307,
        0.3423934 , -0.07929535,  0.        ,  1.32279133])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079067.505576894
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.055622313951294466
```

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)))
```

```
38.70642817269281
1677.7450358836077
40.96028608156451
```

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]:

```
(19397, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(19397,)
```

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)
```

```
[28079006]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.7360416559261741
```

In [54]:

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

Out[54]:

```
0.9999978255573396
```

In [55]:

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

Out[55]:

```
array([[9.99997826e-01, 7.75018107e-20, 2.17444266e-06]])
```

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]:

```
0.7754293098484298
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

In [62]:

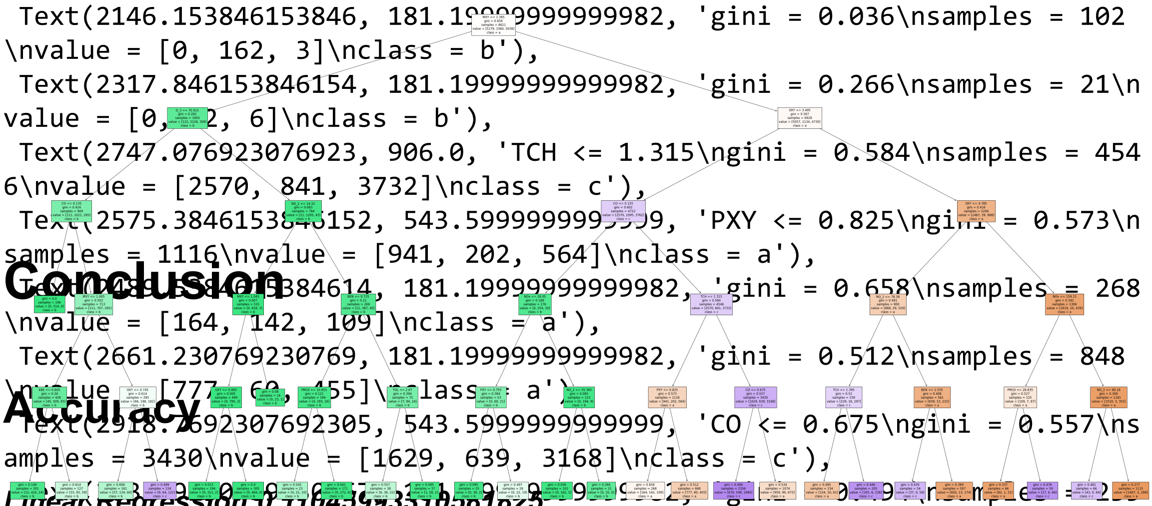
```
from sklearn.tree import plot_tree

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(1899.3461538461538, 1993.2, 'MXY <= 1.385\ngini = 0.656\nsamples = 8
621\nvalue = [5179, 3360, 5038]\nclass = a'),
 Text(708.2307692307692, 1630.8000000000002, 'O_3 <= 70.915\ngini = 0.282
\nsamples = 1693\nvalue = [122, 2226, 308]\nclass = b'),
 Text(257.53846153846155, 1268.4, 'CO <= 0.135\ngini = 0.424\nsamples = 90
9\nvalue = [111, 1021, 265]\nclass = b'),
 Text(171.69230769230768, 906.0, 'gini = 0.0\nsamples = 196\nvalue = [0, 3
14, 0]\nclass = b'),
 Text(343.38461538461536, 906.0, 'MXY <= 1.005\ngini = 0.503\nsamples = 71
3\nvalue = [111, 707, 265]\nclass = b'),
 Text(171.69230769230768, 543.5999999999999, 'EBE <= 0.605\ngini = 0.34\ns
amples = 428\nvalue = [45, 509, 83]\nclass = b'),
 Text(85.84615384615384, 181.19999999999982, 'gini = 0.149\nsamples = 301
\nvalue = [12, 416, 24]\nclass = b'),
 Text(257.53846153846155, 181.19999999999982, 'gini = 0.614\nsamples = 127
\nvalue = [33, 93, 59]\nclass = b'),
 Text(515.0769230769231, 543.5999999999999, 'OXY <= 0.745\ngini = 0.614\ns
amples = 285\nvalue = [66, 198, 182]\nclass = b'),
 Text(429.23076923076917, 181.19999999999982, 'gini = 0.606\nsamples = 161
\nvalue = [57, 134, 60]\nclass = b'),
 Text(600.9230769230769, 181.19999999999982, 'gini = 0.499\nsamples = 124
\nvalue = [9, 64, 122]\nclass = c'),
 Text(1158.923076923077, 1268.4, 'NO_2 <= 14.32\ngini = 0.083\nsamples = 7
84\nvalue = [11, 1205, 43]\nclass = b'),
 Text(944.3076923076923, 906.0, 'MXY <= 1.045\ngini = 0.007\nsamples = 515
\nvalue = [0, 818, 3]\nclass = b'),
 Text(858.4615384615383, 543.5999999999999, 'OXY <= 0.865\ngini = 0.005\ns
amples = 499\nvalue = [0, 795, 2]\nclass = b'),
 Text(772.6153846153845, 181.19999999999982, 'gini = 0.013\nsamples = 194
\nvalue = [0, 311, 2]\nclass = b'),
 Text(944.3076923076923, 181.19999999999982, 'gini = 0.0\nsamples = 305\nv
alue = [0, 484, 0]\nclass = b'),
 Text(1030.1538461538462, 543.5999999999999, 'gini = 0.08\nsamples = 16\nv
alue = [0, 23, 1]\nclass = b'),
 Text(1373.5384615384614, 906.0, 'BEN <= 0.725\ngini = 0.21\nsamples = 269
\nvalue = [11, 387, 40]\nclass = b'),
 Text(1201.8461538461538, 543.5999999999999, 'PM10 <= 10.855\ngini = 0.121
\nsamples = 194\nvalue = [4, 293, 16]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.545\nsamples = 23\nvalue = [4,
21, 10]\nclass = b'),
 Text(1287.6923076923076, 181.19999999999982, 'gini = 0.042\nsamples = 171
\nvalue = [0, 272, 6]\nclass = b'),
 Text(1545.230769230769, 543.5999999999999, 'TOL <= 2.67\ngini = 0.394\nsa
mples = 75\nvalue = [7, 94, 24]\nclass = b'),
 Text(1459.3846153846152, 181.19999999999982, 'gini = 0.557\nsamples = 38
\nvalue = [6, 36, 22]\nclass = b'),
 Text(1631.0769230769229, 181.19999999999982, 'gini = 0.095\nsamples = 37
\nvalue = [1, 58, 2]\nclass = b'),
 Text(3090.461538461538, 1630.8000000000002, 'OXY <= 3.495\ngini = 0.587\n
samples = 6928\nvalue = [5057, 1134, 4730]\nclass = a'),
 Text(2403.6923076923076, 1268.4, 'CO <= 0.155\ngini = 0.602\nsamples = 47
22\nvalue = [2570, 1095, 3762]\nclass = c'),
 Text(2060.3076923076924, 906.0, 'NOx <= 28.05\ngini = 0.189\nsamples = 17
6\nvalue = [0, 254, 30]\nclass = b'),
 Text(1888.6153846153845, 543.5999999999999, 'PXY <= 0.755\ngini = 0.384\n
samples = 53\nvalue = [0, 60, 21]\nclass = b'),
 Text(1802.7692307692307, 181.19999999999982, 'gini = 0.095\nsamples = 25
\nvalue = [0, 38, 2]\nclass = b'),
 Text(1974.4615384615383, 181.19999999999982, 'gini = 0.497\nsamples = 28
\nvalue = [0, 22, 19]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'NO_2 <= 55.365\ngini = 0.085\nsamples =
```

```
123\nvalue = [0, 194, 9]\nclass = b'),
 Text(2146.153846153846, 181.19999999999982, 'gini = 0.036\nsamples = 102
\nvalue = [0, 162, 3]\nclass = b'),
 Text(2317.846153846154, 181.19999999999982, 'gini = 0.266\nsamples = 21\n
value = [0, 32, 6]\nclass = b'),
 Text(2747.076923076923, 906.0, 'TCH <= 1.315\ngini = 0.584\nsamples = 454
6\nvalue = [2570, 841, 3732]\nclass = c'),
 Text(2575.3846153846152, 543.5999999999999, 'PXY <= 0.825\ngini = 0.573\n
samples = 1116\nvalue = [941, 202, 564]\nclass = a'),
 Text(2489.5384615384614, 181.19999999999982, 'gini = 0.658\nsamples = 268
\nvalue = [164, 142, 109]\nclass = a'),
 Text(2661.230769230769, 181.19999999999982, 'gini = 0.512\nsamples = 848
\nvalue = [777, 60, 455]\nclass = a'),
 Text(2918.7692307692305, 543.5999999999999, 'CO <= 0.675\ngini = 0.557\ns
amples = 3430\nvalue = [1629, 639, 3168]\nclass = c'),
6\nvalue = [670, 549, 2495]\nclass = c'),
 Text(3004.6153846153843, 181.19999999999982, 'gini = 0.534\nsamples = 107
4\nvalue = [959, 90, 673]\nclass = a'),
 Text(3777.230769230769, 1268.4, 'OXY <= 4.785\ngini = 0.416\nsamples = 22
06\nvalue = [2487, 39, 968]\nclass = a'),
 Text(3433.8461538461534, 906.0, 'NO_2 <= 78.39\ngini = 0.491\nsamples = 9
00\nvalue = [868, 29, 529]\nclass = a'),
 Text(3262.1538461538457, 543.5999999999999, 'TCH <= 1.395\ngini = 0.52\ns
amples = 339\nvalue = [229, 16, 297]\nclass = c'),
 Text(3176.3076923076924, 181.19999999999982, 'gini = 0.495\nsamples = 134
\nvalue = [124, 10, 61]\nclass = a'),
 Text(3347.9999999999995, 181.19999999999982, 'gini = 0.446\nsamples = 205
\nvalue = [105, 6, 236]\nclass = c'),
 Text(3605.5384615384614, 543.5999999999999, 'BEN <= 2.535\ngini = 0.408\n
samples = 561\nvalue = [639, 13, 232]\nclass = a'),
 Text(3519.6923076923076, 181.19999999999982, 'gini = 0.476\nsamples = 54
\nvalue = [37, 0, 58]\nclass = c'),
 Text(3691.3846153846152, 181.19999999999982, 'gini = 0.369\nsamples = 507
\nvalue = [602, 13, 174]\nclass = a'),
 Text(4120.615384615385, 906.0, 'NOx <= 159.15\ngini = 0.342\nsamples = 13
06\nvalue = [1619, 10, 439]\nclass = a'),
 Text(3948.9230769230767, 543.5999999999999, 'PM10 <= 28.835\ngini = 0.527
\nsamples = 125\nvalue = [109, 7, 87]\nclass = a'),
 Text(3863.076923076923, 181.19999999999982, 'gini = 0.337\nsamples = 66\n
value = [82, 1, 21]\nclass = a'),
 Text(4034.7692307692305, 181.19999999999982, 'gini = 0.478\nsamples = 59
\nvalue = [27, 6, 66]\nclass = c'),
 Text(4292.307692307692, 543.5999999999999, 'NO_2 <= 80.16\ngini = 0.309\n
samples = 1181\nvalue = [1519, 3, 352]\nclass = a')
```



Conclusion

Accuracy

Linear Regression:0.11045993310581823

Ridge Regression:0.11011845641033136

Lasso Regression:0.04454762498104433

ElasticNet Regression:0.05562231395129446

Logistic Regression:0.7360416559261741

Random Forest:0.7754293098484298

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

In [ ]: