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("i2015.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	st
0	2015-10-01 01:00:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	2807
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	2807
2	2015-10-01 01:00:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	2807
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	2807
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	2807
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210091	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN	2807
210092	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN	2807
210093	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN	2807
210094	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN	2807
210095	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN	2807

210096 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:

```
df=df.dropna()
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

In [6]:

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

Out[6]:

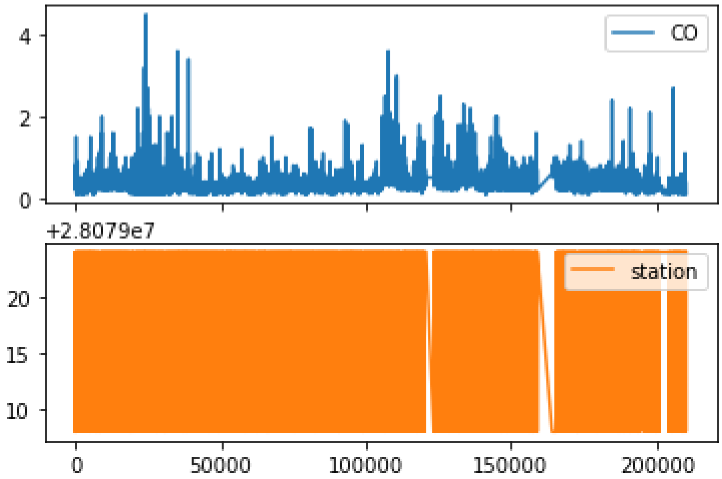
	CO	station
1	0.8	28079008
6	0.3	28079024
25	0.7	28079008
30	0.3	28079024
49	0.8	28079008
...	...	...
210030	0.1	28079024
210049	0.3	28079008
210054	0.1	28079024
210073	0.3	28079008
210078	0.1	28079024

16026 rows × 2 columns

Line chart

In [7]:
```
data.plot.line(subplots=True)
```

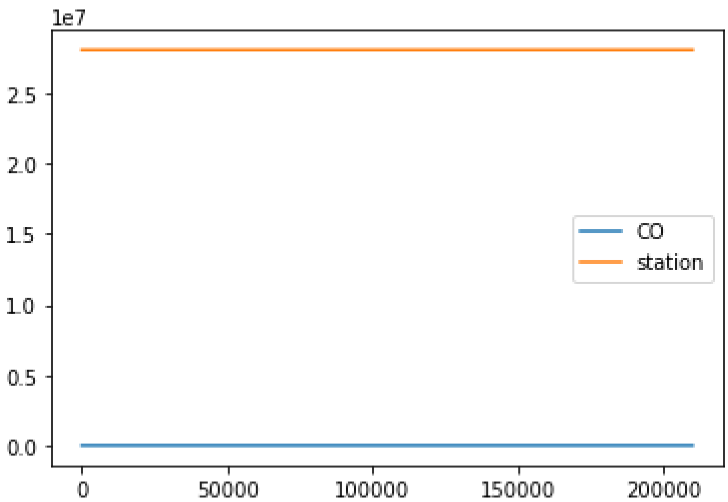
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [8]:
```
data.plot.line()
```

Out[8]: <AxesSubplot:>

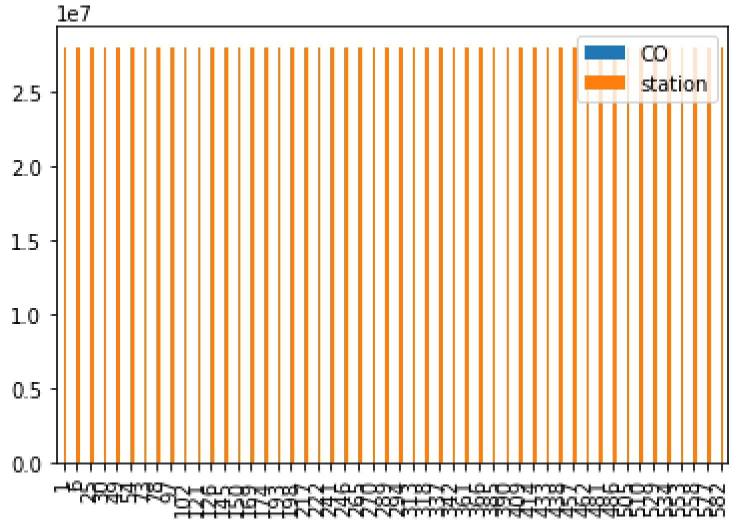


Bar chart

In [9]:
```
b=data[0:50]
```

In [10]:
```
b.plot.bar()
```

Out[10]: <AxesSubplot:>

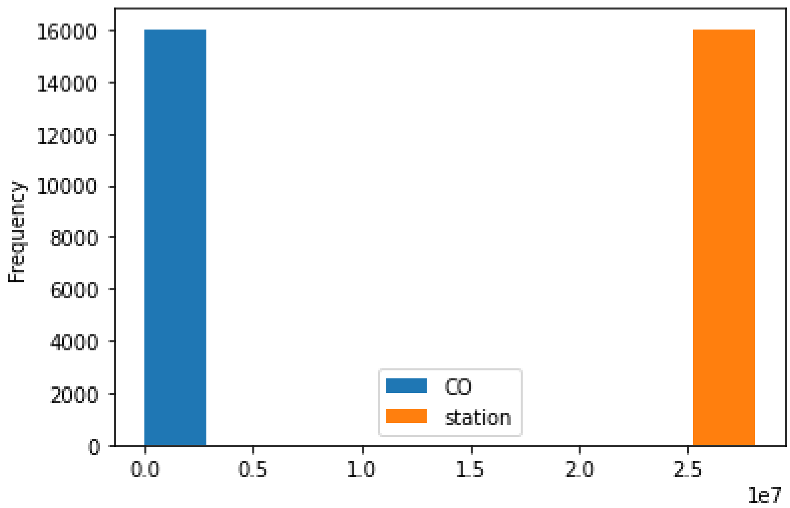


Histogram

In [11]:

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

Out[11]: <AxesSubplot:ylabel='Frequency'>

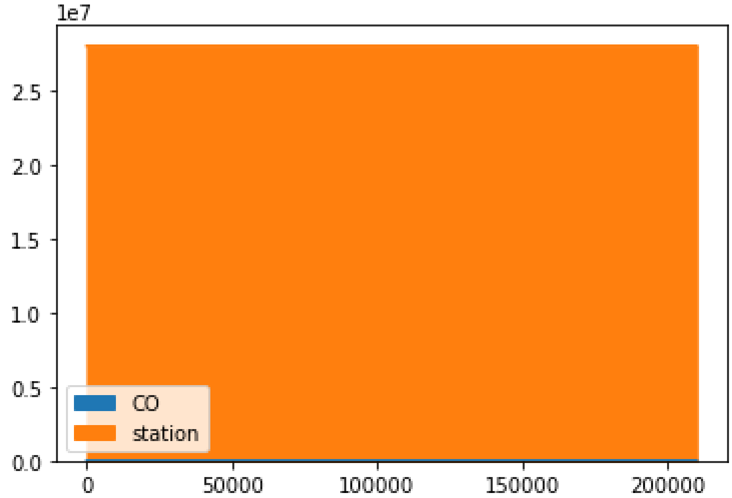


Area chart

In [12]:

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

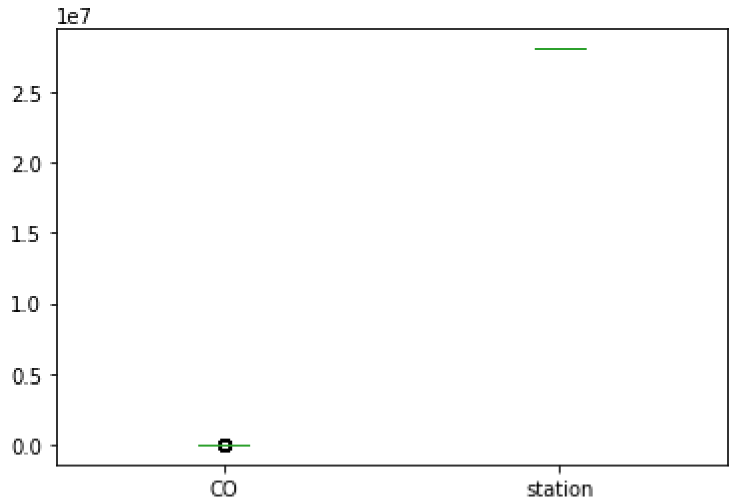
Out[12]: <AxesSubplot:>



Box chart

In [13]:
```
data.plot.box()
```

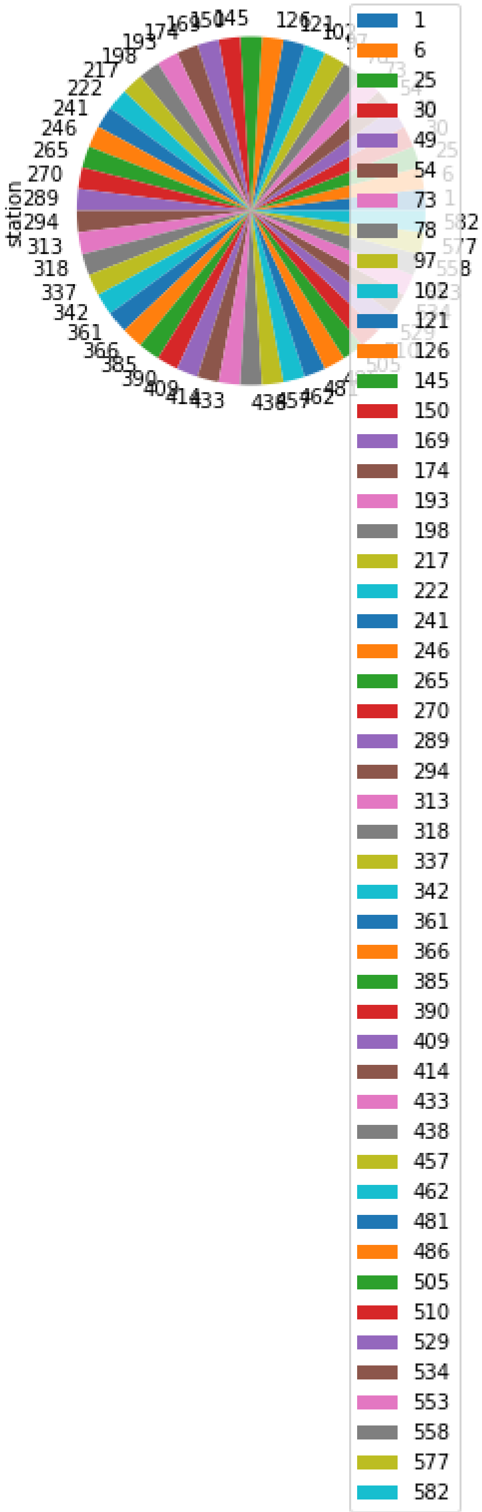
Out[13]: <AxesSubplot:>



Pie chart

In [14]:
```
b.plot.pie(y='station' )
```

Out[14]: <AxesSubplot:ylabel='station'>

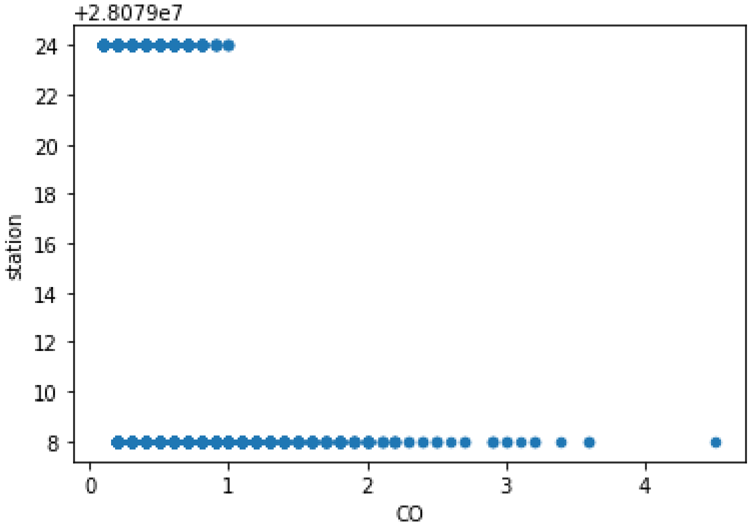


Scatter chart

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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [18]:
```
df.describe()
```

Out[18]:

	BEN	CO	EBE	NMHC	NO	NO_2	
count	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.0C
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	48.08
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	35.84
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.0C
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	15.0C
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	46.0C
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	73.0C

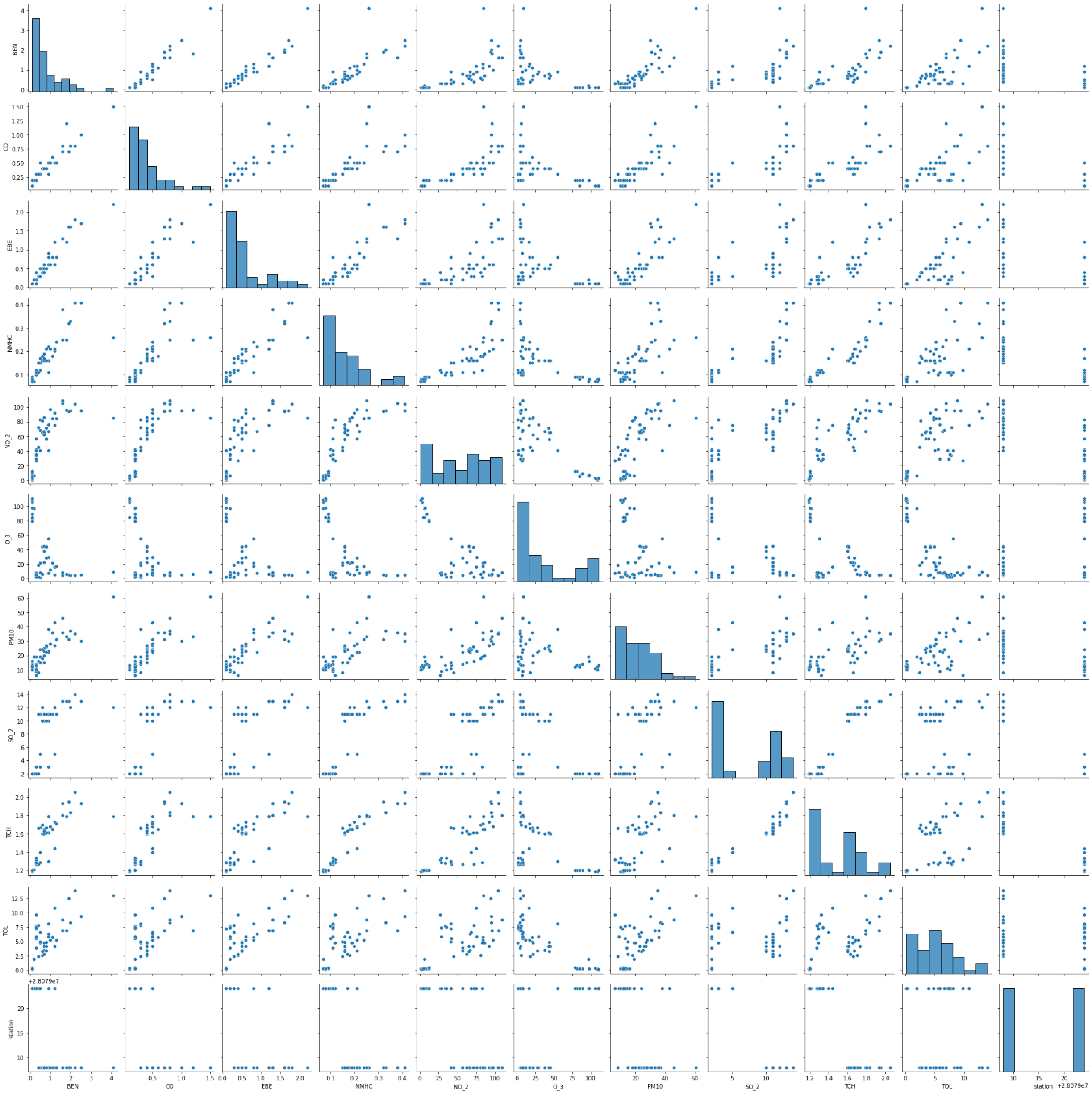
	BEN	CO	EBE	NMHC	NO	NO_2	
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	217.0C

In [19]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2','O_3',
        'PM10','SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x2b88a8dc880>

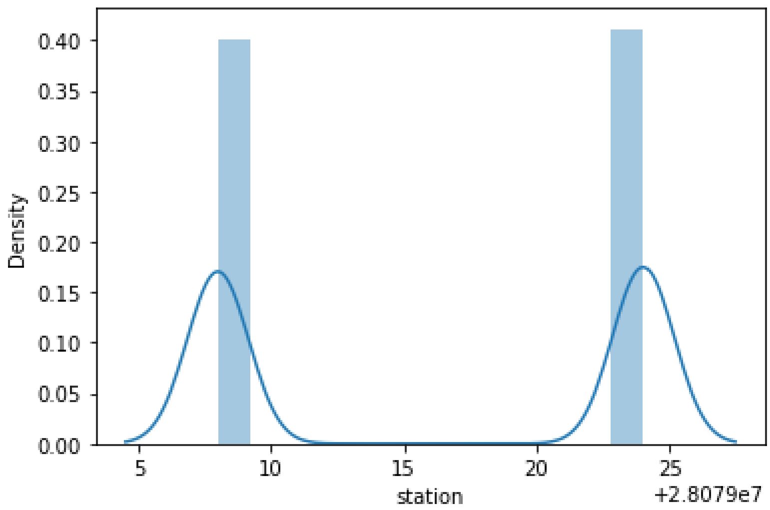


In [21]:
```
sns.distplot(df1['station'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
```

```
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

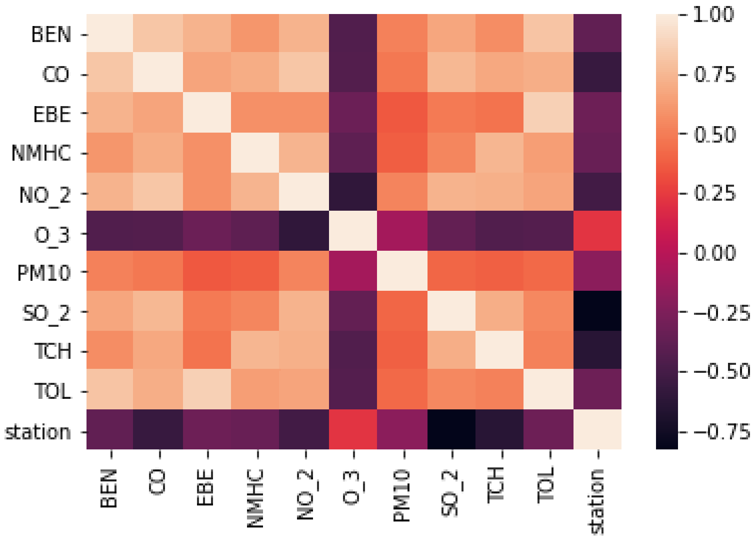
Out[21]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [22]:

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

Out[22]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [23]:

```
x=df[['BEN', 'CO', 'EBE','NMHC', 'NO_2','O_3',
       'PM10','SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [24]:

```
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 [25]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[25]: LinearRegression()

In [26]:
```
lr.intercept_
```

Out[26]: 28079037.971513327

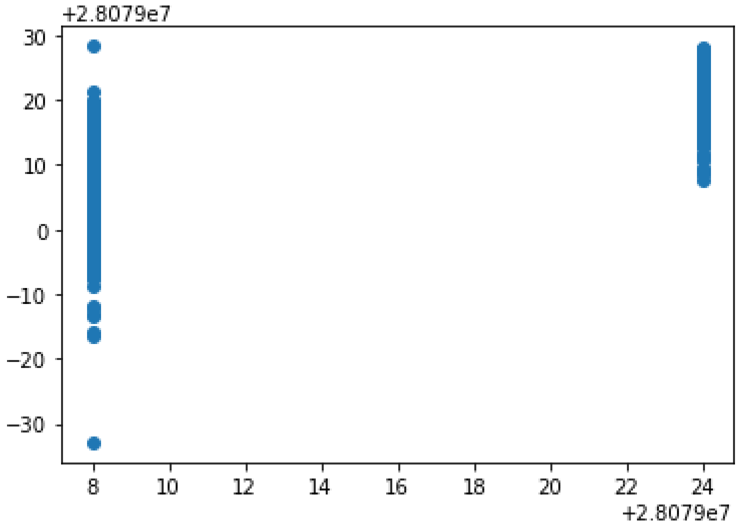
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	4.829039
CO	-7.201094
EBE	-1.221655
NMHC	24.353949
NO_2	-0.001239
O_3	-0.017437
PM10	0.057771
SO_2	-1.173156
TCH	-10.943129
TOL	-0.070548

In [28]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[28]: <matplotlib.collections.PathCollection at 0x2b892efa7c0>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```
Out[29]: 0.8058280829064649

In [30]:
```
lr.score(x_train,y_train)
```
Out[30]: 0.8089532029954628

Ridge and Lasso

In [31]:
```
from sklearn.linear_model import Ridge,Lasso
```
In [32]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```
Out[32]: Ridge(alpha=10)

Accuracy(Ridge)

In [33]:
```
rr.score(x_test,y_test)
```
Out[33]: 0.8042402331444318

In [34]:
```
rr.score(x_train,y_train)
```
Out[34]: 0.8068821169448739

In [35]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[35]: Lasso(alpha=10)

In [36]:
```
la.score(x_train,y_train)
```
Out[36]: 0.6361709027833888

Accuracy(Lasso)

In [37]:
```
la.score(x_test,y_test)
```
Out[37]: 0.6343603719349661

Elastic Net

In [38]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[38]:
```
ElasticNet()
```

In [39]:
```
en.coef_
```

Out[39]:
```
array([ 0.        , -0.        , -0.        ,  0.        , -0.        ,
       -0.01573327,  0.07181082, -1.25810757, -0.        ,  0.15780668])
```

In [40]:
```
en.intercept_
```

Out[40]:
```
28079024.1919828
```

In [41]:
```
prediction=en.predict(x_test)
```

In [42]:
```
en.score(x_test,y_test)
```

Out[42]:
```
0.7348213384433098
```

Evaluation Metrics

In [43]:
```
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)))
```

```
3.234674387959339
16.967628463726648
4.11917812964269
```

Logistic Regression

In [44]:
```
from sklearn.linear_model import LogisticRegression
```

In [45]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2','O_3',
       'PM10','SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [46]:
```
feature_matrix.shape
```

Out[46]:
```
(16026, 10)
```

In [47]:
```
target_vector.shape
```

Out[47]:
```
(16026,)
```

In [48]:
```
from sklearn.preprocessing import StandardScaler
```

In [49]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [50]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[50]:
```
LogisticRegression(max_iter=10000)
```

In [51]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [52]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [53]:
```
logr.classes_
```

Out[53]:
```
array([28079008, 28079024], dtype=int64)
```

In [54]:
```
logr.score(fs,target_vector)
```

Out[54]:
```
0.9947585174092101
```

In [55]:
```
logr.predict_proba(observation)[0][0]
```

Out[55]:
```
1.0
```

In [56]:
```
logr.predict_proba(observation)
```

Out[56]:
```
array([[1.00000000e+00, 5.69793111e-39]])
```

Random Forest

In [57]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [58]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[58]:
```
RandomForestClassifier()
```

In [59]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [60]:

```
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[60]:

```
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 [61]:

```
grid_search.best_score_
```

Out[61]:

```
0.9948297379211981
```

In [62]:

```
rfc_best=grid_search.best_estimator_
```

In [63]:

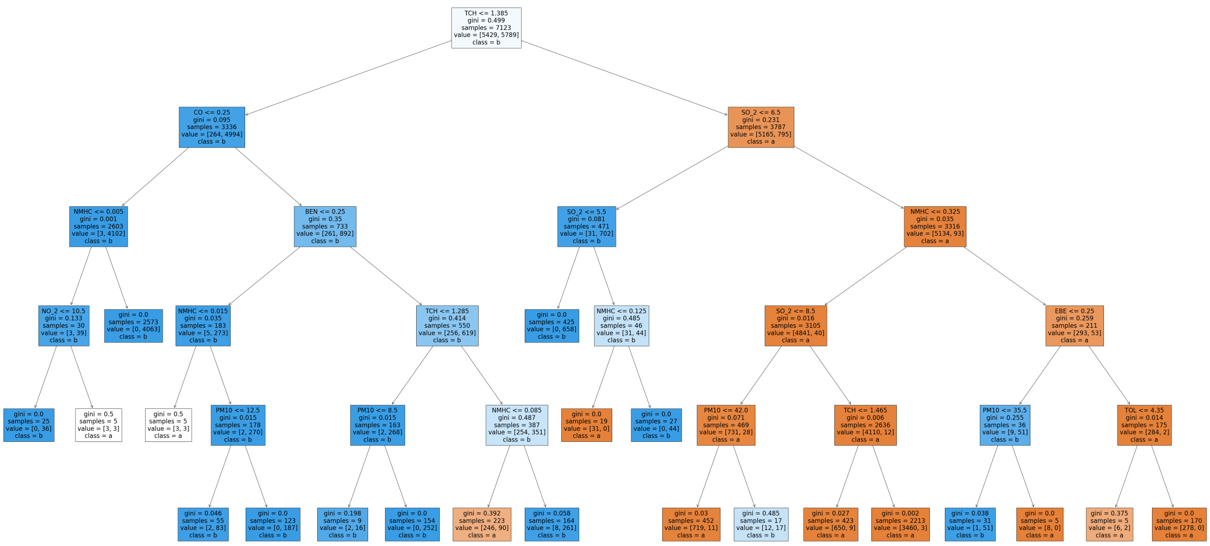
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','
```

Out[63]:

```
[Text(1801.5428571428572, 1993.2, 'TCH <= 1.385\ngini = 0.499\nsamples = 7123\nvalue
= [5429, 5789]\nclass = b'),
 Text(797.1428571428571, 1630.8000000000002, 'CO <= 0.25\ngini = 0.095\nsamples = 33
36\nvalue = [264, 4994]\nclass = b'),
 Text(382.62857142857143, 1268.4, 'NMHC <= 0.005\ngini = 0.001\nsamples = 2603\nvalu
e = [3, 4102]\nclass = b'),
 Text(255.0857142857143, 906.0, 'NO_2 <= 10.5\ngini = 0.133\nsamples = 30\nvalue =
[3, 39]\nclass = b'),
 Text(127.54285714285714, 543.5999999999999, 'gini = 0.0\nsamples = 25\nvalue = [0,
36]\nclass = b'),
 Text(382.62857142857143, 543.5999999999999, 'gini = 0.5\nsamples = 5\nvalue = [3,
3]\nclass = a'),
 Text(510.1714285714286, 906.0, 'gini = 0.0\nsamples = 2573\nvalue = [0, 4063]\nclas
s = b'),
 Text(1211.6571428571428, 1268.4, 'BEN <= 0.25\ngini = 0.35\nsamples = 733\nvalue =
[261, 892]\nclass = b'),
 Text(765.2571428571429, 906.0, 'NMHC <= 0.015\ngini = 0.035\nsamples = 183\nvalue =
[5, 273]\nclass = b'),
 Text(637.7142857142858, 543.5999999999999, 'gini = 0.5\nsamples = 5\nvalue = [3, 3]
\nclass = a'),
 Text(892.8, 543.5999999999999, 'PM10 <= 12.5\ngini = 0.015\nsamples = 178\nvalue =
[2, 270]\nclass = b'),
 Text(765.2571428571429, 181.19999999999982, 'gini = 0.046\nsamples = 55\nvalue =
[2, 83]\nclass = b'),
 Text(1020.3428571428572, 181.19999999999982, 'gini = 0.0\nsamples = 123\nvalue =
[0, 187]\nclass = b'),
 Text(1658.057142857143, 906.0, 'TCH <= 1.285\ngini = 0.414\nsamples = 550\nvalue =
[256, 619]\nclass = b'),
 Text(1402.9714285714285, 543.5999999999999, 'PM10 <= 8.5\ngini = 0.015\nsamples = 1
63\nvalue = [2, 268]\nclass = b'),
 Text(1275.4285714285716, 181.19999999999982, 'gini = 0.198\nsamples = 9\nvalue =
[2, 16]\nclass = b'),
 Text(1530.5142857142857, 181.19999999999982, 'gini = 0.0\nsamples = 154\nvalue =
[0, 252]\nclass = b'),
 Text(1913.142857142857, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.487\nsamples =
387\nvalue = [254, 351]\nclass = b'),
 Text(1785.6, 181.19999999999982, 'gini = 0.392\nsamples = 223\nvalue = [246, 90]\nc
lass = a'),
 Text(2040.6857142857143, 181.19999999999982, 'gini = 0.058\nsamples = 164\nvalue =
[8, 261]\nclass = b'),
 Text(2805.942857142857, 1630.8000000000002, 'SO_2 <= 6.5\ngini = 0.231\nsamples = 3
```

```
787\nvalue = [5165, 795]\nclass = a'),
 Text(2168.2285714285713, 1268.4, 'SO_2 <= 5.5\ngini = 0.081\nsamples = 471\nvalue =
[31, 702]\nclass = b'),
 Text(2040.6857142857143, 906.0, 'gini = 0.0\nsamples = 425\nvalue = [0, 658]\nclass
= b'),
 Text(2295.7714285714287, 906.0, 'NMHC <= 0.125\ngini = 0.485\nsamples = 46\nvalue =
[31, 44]\nclass = b'),
 Text(2168.2285714285713, 543.5999999999999, 'gini = 0.0\nsamples = 19\nvalue = [31,
0]\nclass = a'),
 Text(2423.3142857142857, 543.5999999999999, 'gini = 0.0\nsamples = 27\nvalue = [0,
44]\nclass = b'),
 Text(3443.657142857143, 1268.4, 'NMHC <= 0.325\ngini = 0.035\nsamples = 3316\nvalue
= [5134, 93]\nclass = a'),
 Text(2933.4857142857145, 906.0, 'SO_2 <= 8.5\ngini = 0.016\nsamples = 3105\nvalue =
[4841, 40]\nclass = a'),
 Text(2678.4, 543.5999999999999, 'PM10 <= 42.0\ngini = 0.071\nsamples = 469\nvalue =
[731, 28]\nclass = a'),
 Text(2550.857142857143, 181.19999999999982, 'gini = 0.03\nsamples = 452\nvalue = [7
19, 11]\nclass = a'),
 Text(2805.942857142857, 181.19999999999982, 'gini = 0.485\nsamples = 17\nvalue = [1
2, 17]\nclass = b'),
 Text(3188.5714285714284, 543.5999999999999, 'TCH <= 1.465\ngini = 0.006\nsamples =
2636\nvalue = [4110, 12]\nclass = a'),
 Text(3061.0285714285715, 181.19999999999982, 'gini = 0.027\nsamples = 423\nvalue =
[650, 9]\nclass = a'),
 Text(3316.114285714286, 181.19999999999982, 'gini = 0.002\nsamples = 2213\nvalue =
[3460, 3]\nclass = a'),
 Text(3953.8285714285716, 906.0, 'EBE <= 0.25\ngini = 0.259\nsamples = 211\nvalue =
[293, 53]\nclass = a'),
 Text(3698.7428571428572, 543.5999999999999, 'PM10 <= 35.5\ngini = 0.255\nsamples =
36\nvalue = [9, 51]\nclass = b'),
 Text(3571.2, 181.19999999999982, 'gini = 0.038\nsamples = 31\nvalue = [1, 51]\nclas
s = b'),
 Text(3826.285714285714, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [8,
0]\nclass = a'),
 Text(4208.914285714286, 543.5999999999999, 'TOL <= 4.35\ngini = 0.014\nsamples = 17
5\nvalue = [284, 2]\nclass = a'),
 Text(4081.3714285714286, 181.19999999999982, 'gini = 0.375\nsamples = 5\nvalue =
[6, 2]\nclass = a'),
 Text(4336.457142857143, 181.19999999999982, 'gini = 0.0\nsamples = 170\nvalue = [27
8, 0]\nclass = a')]
```



Conclusion

Scores

Linear Regression

In [64]:
```
lr.score(x_test,y_test)
```

Out[64]: 0.8058280829064649

In [65]:
```
lr.score(x_train,y_train)
```

Out[65]: 0.8089532029954628

Lasso

In [66]:
```
la.score(x_test,y_test)
```

Out[66]: 0.6343603719349661

Ridge

In [67]:
```
rr.score(x_test,y_test)
```

Out[67]: 0.8042402331444318

In [68]:
```
rr.score(x_train,y_train)
```

Out[68]: 0.8068821169448739

Elastic Net

In [69]:
```
en.score(x_test,y_test)
```

Out[69]: 0.7348213384433098

Logistic Regression

In [70]:
```
logr.score(fs,target_vector)
```

Out[70]: 0.9947585174092101

Random Forest

In [71]:
```
grid_search.best_score_
```

Out[71]: 0.9948297379211981

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

In [ ]: