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("2014.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	st
0	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	3.0	10.0	NaN	NaN	NaN	3.0	NaN	NaN	2807
1	2014-06-01 01:00:00	0.2	0.2	0.1	0.11	3.0	17.0	68.0	10.0	5.0	5.0	1.36	1.3	2807
2	2014-06-01 01:00:00	0.3	NaN	0.1	NaN	2.0	6.0	NaN	NaN	NaN	NaN	NaN	1.1	2807
3	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	79.0	NaN	NaN	NaN	NaN	NaN	2807
4	2014-06-01 01:00:00	NaN	NaN	NaN	NaN	1.0	6.0	75.0	NaN	NaN	4.0	NaN	NaN	2807
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210019	2014-09-01 00:00:00	NaN	0.5	NaN	NaN	20.0	84.0	29.0	NaN	NaN	NaN	NaN	NaN	2807
210020	2014-09-01 00:00:00	NaN	0.3	NaN	NaN	1.0	22.0	NaN	15.0	NaN	6.0	NaN	NaN	2807
210021	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	13.0	70.0	NaN	NaN	NaN	NaN	NaN	2807
210022	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	38.0	42.0	NaN	NaN	NaN	NaN	NaN	2807
210023	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	26.0	65.0	11.0	NaN	NaN	NaN	NaN	2807

210024 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: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ MB
```

In [6]:
```
data=df[['CO' ,'station']]
data
```

Out[6]:

	CO	station
1	0.2	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
209958	0.2	28079024
209977	0.7	28079008
209982	0.2	28079024
210001	0.4	28079008
210006	0.2	28079024

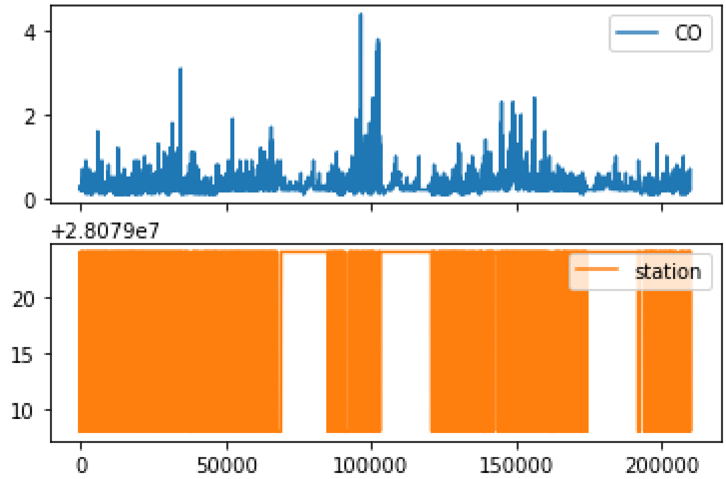
13946 rows × 2 columns

Line chart

In [7]:

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

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

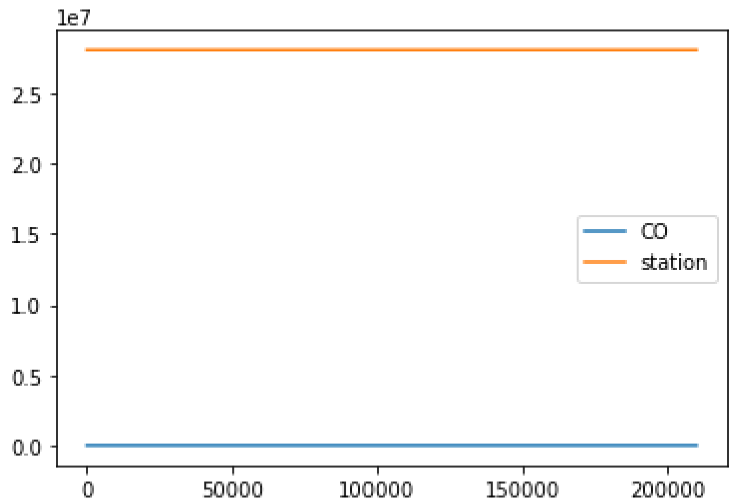


Line chart

In [8]:

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

Out[8]: <AxesSubplot:>



Bar chart

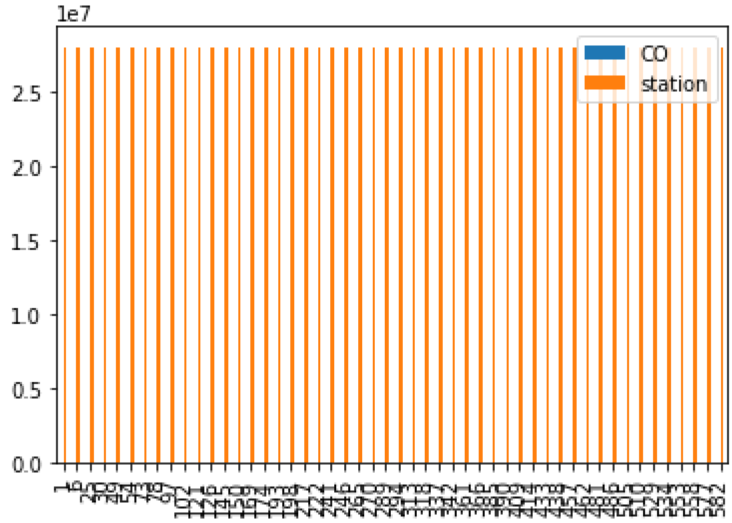
In [9]:

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

In [10]:

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

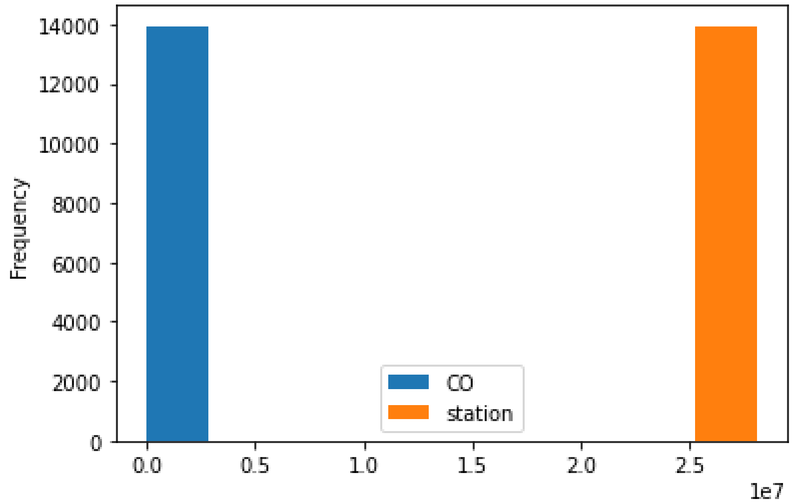
Out[10]: <AxesSubplot:>



Histogram

In [11]:
```
data.plot.hist()
```

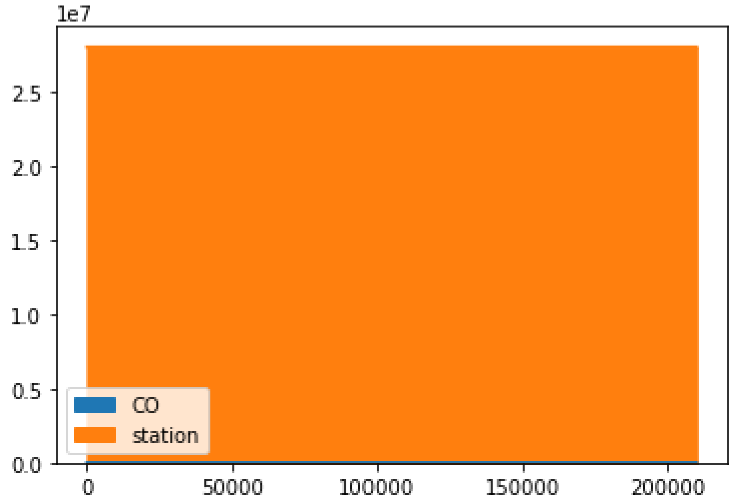
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [12]:
```
data.plot.area()
```

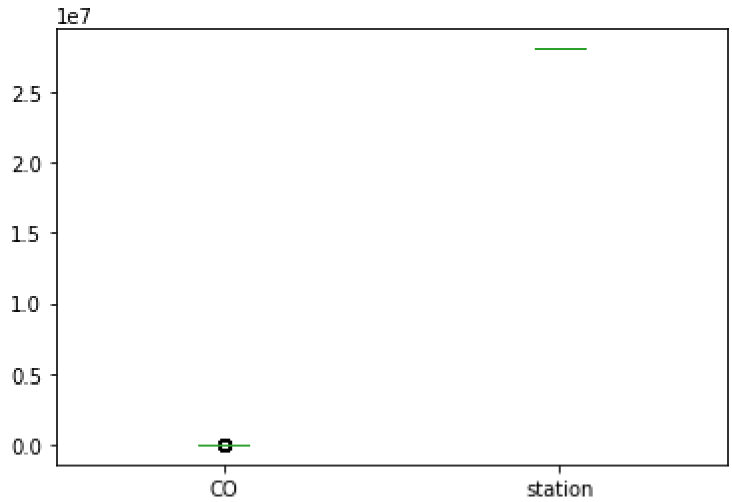
Out[12]: <AxesSubplot:>



Box chart

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

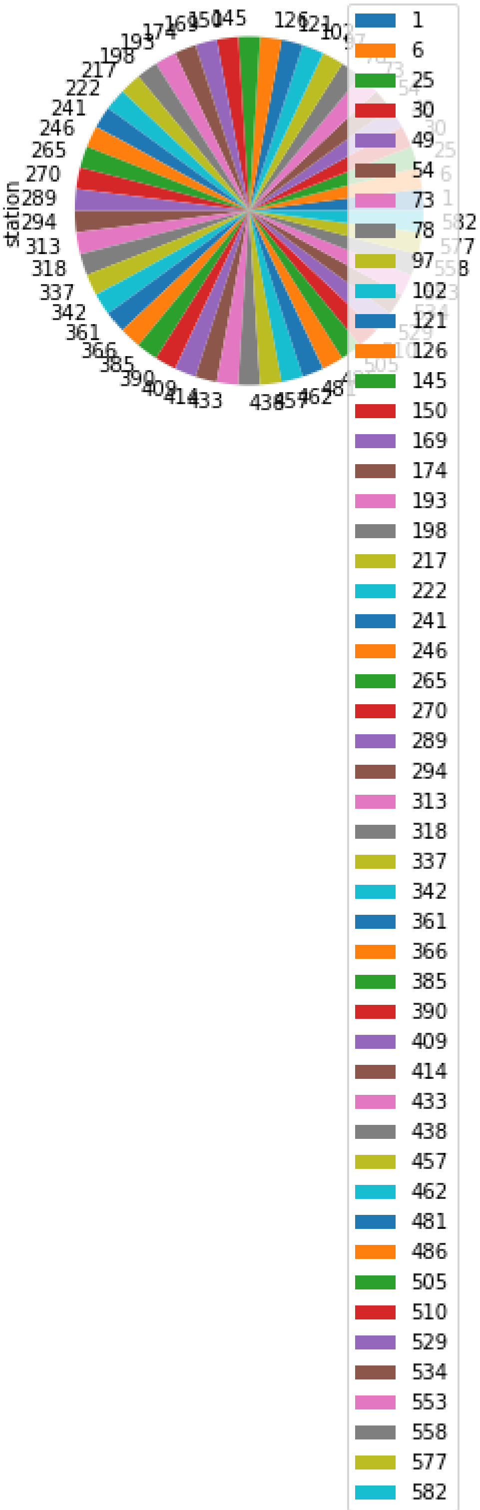
Out[13]: <AxesSubplot:>



Pie chart

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

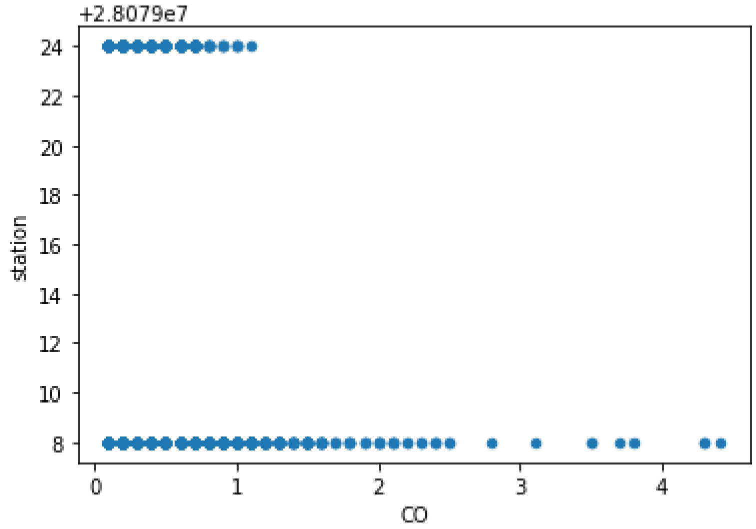
Out[14]: <AxesSubplot:ylabel='station'>



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: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ 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	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.0C
mean	0.375921	0.314793	0.306016	0.222302	17.589129	34.240929	53.08
std	0.555093	0.207375	0.635475	0.082403	39.432216	30.654229	33.48
min	0.100000	0.100000	0.100000	0.060000	1.000000	1.000000	1.0C
25%	0.100000	0.200000	0.100000	0.160000	1.000000	10.000000	25.0C
50%	0.200000	0.300000	0.100000	0.230000	4.000000	27.000000	53.0C
75%	0.400000	0.400000	0.300000	0.260000	18.000000	51.000000	75.0C

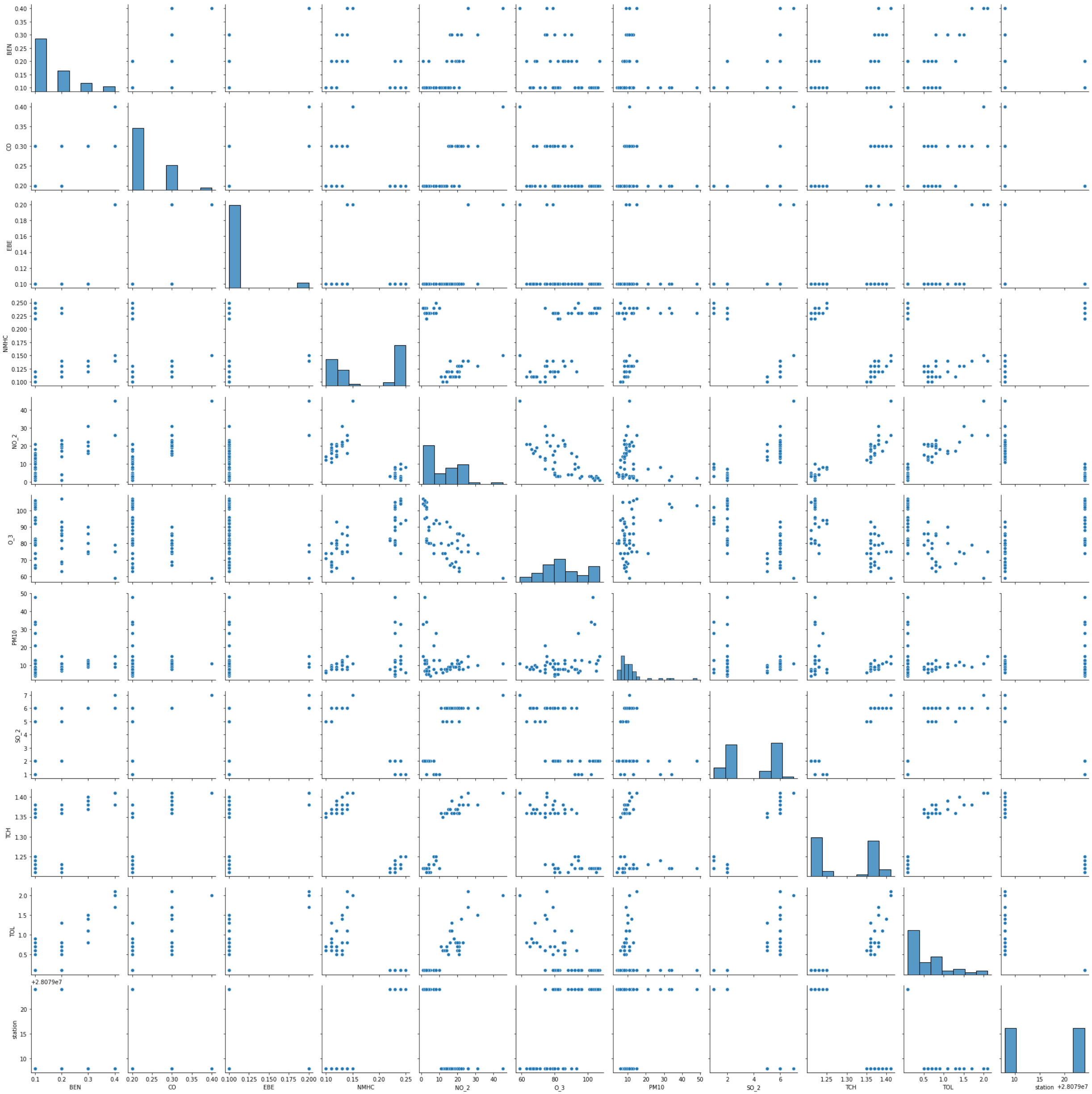
	BEN	CO	EBE	NMHC	NO	NO_2	
max	9.400000	4.400000	16.200001	1.290000	725.000000	346.000000	220.0C

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 0x23c198717f0>

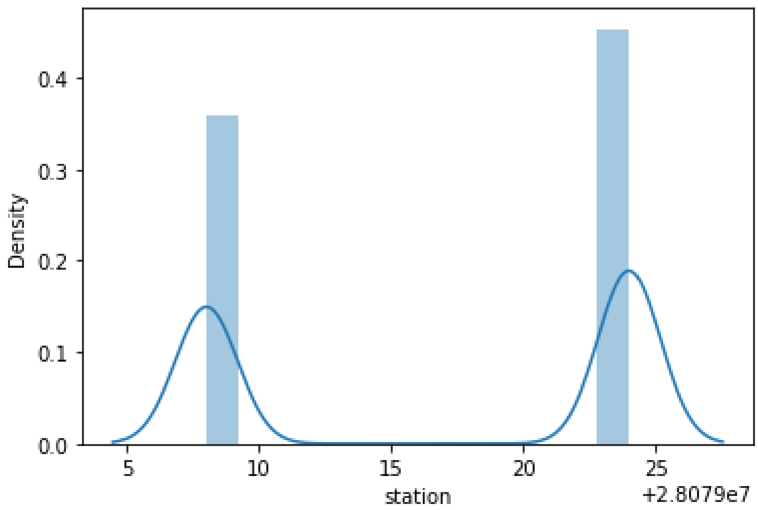


In [21]:
```
sns.distplot(df1['station'])
```

```
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)
```

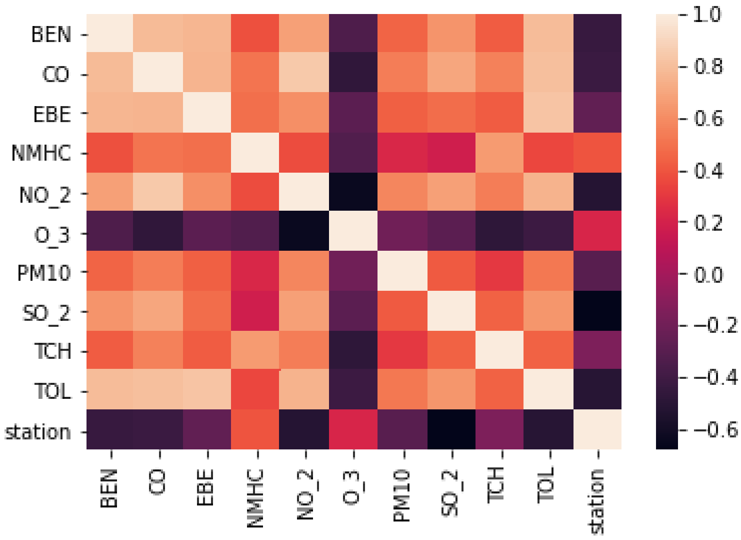
Out[21]: <AxesSubplot:xlabel='station', ylabel='Density'>



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]: 28079022.485831894

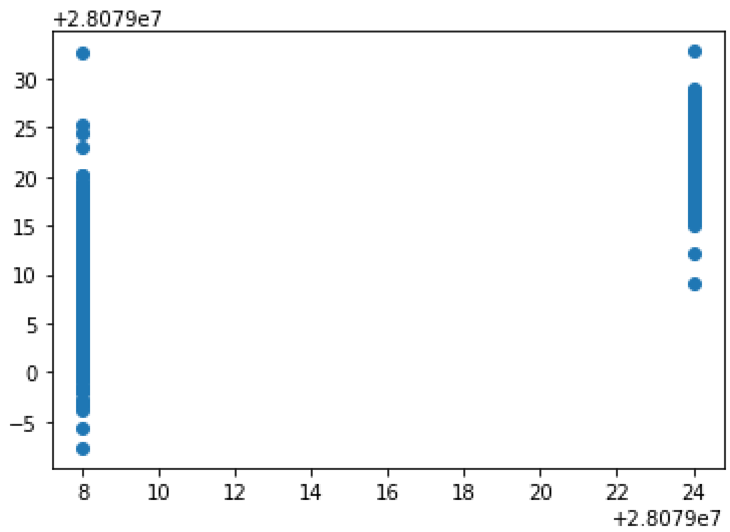
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	-1.457775
CO	-6.062174
EBE	0.340813
NMHC	83.433732
NO_2	-0.030062
O_3	0.002398
PM10	0.016894
SO_2	-0.889786
TCH	-11.796646
TOL	-0.415177

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

Out[28]: <matplotlib.collections.PathCollection at 0x23c22e183d0>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```
Out[29]: 0.8864850758116184

In [30]:
```
lr.score(x_train,y_train)
```
Out[30]: 0.8831207101050416

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]: 0.8640411021771052

In [34]:
```
rr.score(x_train,y_train)
```
Out[34]: 0.8600208413950009

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]: 0.2631383776944629

Accuracy(Lasso)

In [37]:
```
la.score(x_test,y_test)
```
Out[37]: 0.2912395128676807

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.        ,  0.17685506,  0.        , -0.0388346 ,
       -0.00955803,  0.01961017, -1.27800418,  0.        , -0.17094365])
```

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

Out[40]: 28079024.58615365

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

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

Out[42]: 0.48615561969899956

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)))
```

```
4.996357973422819
32.38585882534258
5.690857477159534
```

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]: (13946, 10)

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

Out[47]: (13946,)

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]:
```
0.9926143697117453
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 5.27113072e-18]])
```

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]:
```
0.9953902888752305
```

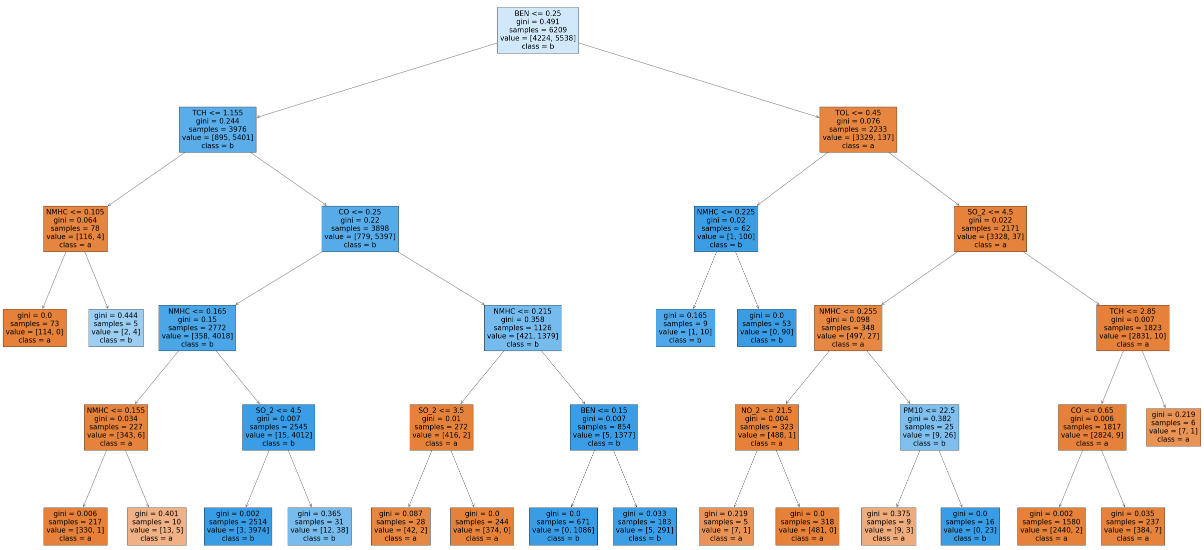
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1990.2, 1993.2, 'BEN <= 0.25\ngini = 0.491\nsamples = 6209\nvalue = [4224, 553
8]\nclass = b'),
 Text(818.4000000000001, 1630.8000000000002, 'TCH <= 1.155\ngini = 0.244\nsamples =
3976\nvalue = [895, 5401]\nclass = b'),
 Text(297.6, 1268.4, 'NMHC <= 0.105\ngini = 0.064\nsamples = 78\nvalue = [116, 4]\nc
lass = a'),
 Text(148.8, 906.0, 'gini = 0.0\nsamples = 73\nvalue = [114, 0]\nclass = a'),
 Text(446.40000000000003, 906.0, 'gini = 0.444\nsamples = 5\nvalue = [2, 4]\nclass =
b'),
 Text(1339.2, 1268.4, 'CO <= 0.25\ngini = 0.22\nsamples = 3898\nvalue = [779, 5397]
\nclass = b'),
 Text(744.0, 906.0, 'NMHC <= 0.165\ngini = 0.15\nsamples = 2772\nvalue = [358, 4018]
\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'NMHC <= 0.155\ngini = 0.034\nsamples =
227\nvalue = [343, 6]\nclass = a'),
 Text(297.6, 181.19999999999982, 'gini = 0.006\nsamples = 217\nvalue = [330, 1]\ncla
ss = a'),
 Text(595.2, 181.19999999999982, 'gini = 0.401\nsamples = 10\nvalue = [13, 5]\nclass
= a'),
 Text(1041.6000000000001, 543.5999999999999, 'SO_2 <= 4.5\ngini = 0.007\nsamples = 2
545\nvalue = [15, 4012]\nclass = b'),
 Text(892.8000000000001, 181.19999999999982, 'gini = 0.002\nsamples = 2514\nvalue =
[3, 3974]\nclass = b'),
 Text(1190.4, 181.19999999999982, 'gini = 0.365\nsamples = 31\nvalue = [12, 38]\ncla
ss = b'),
 Text(1934.4, 906.0, 'NMHC <= 0.215\ngini = 0.358\nsamples = 1126\nvalue = [421, 137
9]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'SO_2 <= 3.5\ngini = 0.01\nsamples = 27
2\nvalue = [416, 2]\nclass = a'),
 Text(1488.0, 181.19999999999982, 'gini = 0.087\nsamples = 28\nvalue = [42, 2]\nclas
s = a'),
 Text(1785.6000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 244\nvalue = [3
74, 0]\nclass = a'),
 Text(2232.0, 543.5999999999999, 'BEN <= 0.15\ngini = 0.007\nsamples = 854\nvalue =
[5, 1377]\nclass = b'),
 Text(2083.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 671\nvalue =
[0, 1086]\nclass = b'),
 Text(2380.8, 181.19999999999982, 'gini = 0.033\nsamples = 183\nvalue = [5, 291]\ncl
ass = b'),
 Text(3162.0000000000005, 1630.8000000000002, 'TOL <= 0.45\ngini = 0.076\nsamples =
2233\nvalue = [3329, 137]\nclass = a'),
```

```
 Text(2678.4, 1268.4, 'NMHC <= 0.225\ngini = 0.02\nsamples = 62\nvalue = [1, 100]\nc
lass = b'),
 Text(2529.6000000000004, 906.0, 'gini = 0.165\nsamples = 9\nvalue = [1, 10]\nclass
= b'),
 Text(2827.2000000000003, 906.0, 'gini = 0.0\nsamples = 53\nvalue = [0, 90]\nclass =
b'),
 Text(3645.6000000000004, 1268.4, 'SO_2 <= 4.5\ngini = 0.022\nsamples = 2171\nvalue
= [3328, 37]\nclass = a'),
 Text(3124.8, 906.0, 'NMHC <= 0.255\ngini = 0.098\nsamples = 348\nvalue = [497, 27]
\nclass = a'),
 Text(2827.2000000000003, 543.5999999999999, 'NO_2 <= 21.5\ngini = 0.004\nsamples =
323\nvalue = [488, 1]\nclass = a'),
 Text(2678.4, 181.19999999999982, 'gini = 0.219\nsamples = 5\nvalue = [7, 1]\nclass
= a'),
 Text(2976.0, 181.19999999999982, 'gini = 0.0\nsamples = 318\nvalue = [481, 0]\nclas
s = a'),
 Text(3422.4, 543.5999999999999, 'PM10 <= 22.5\ngini = 0.382\nsamples = 25\nvalue =
[9, 26]\nclass = b'),
 Text(3273.6000000000004, 181.19999999999982, 'gini = 0.375\nsamples = 9\nvalue =
[9, 3]\nclass = a'),
 Text(3571.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 16\nvalue = [0,
23]\nclass = b'),
 Text(4166.400000000001, 906.0, 'TCH <= 2.85\ngini = 0.007\nsamples = 1823\nvalue =
[2831, 10]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'CO <= 0.65\ngini = 0.006\nsamples = 18
17\nvalue = [2824, 9]\nclass = a'),
 Text(3868.8, 181.19999999999982, 'gini = 0.002\nsamples = 1580\nvalue = [2440, 2]\n
class = a'),
 Text(4166.400000000001, 181.19999999999982, 'gini = 0.035\nsamples = 237\nvalue =
[384, 7]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'gini = 0.219\nsamples = 6\nvalue = [7,
1]\nclass = a')]
```



Conclusion

Scores

Linear Regression

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

Out[64]: 0.8864850758116184

In [65]:
```
lr.score(x_train,y_train)
```
Out[65]: 0.8831207101050416

Lasso

In [66]:
```
la.score(x_test,y_test)
```
Out[66]: 0.2912395128676807

Ridge

In [67]:
```
rr.score(x_test,y_test)
```
Out[67]: 0.8640411021771052

In [68]:
```
rr.score(x_train,y_train)
```
Out[68]: 0.8600208413950009

Elastic Net

In [69]:
```
en.score(x_test,y_test)
```
Out[69]: 0.48615561969899956

Logistic Regression

In [70]:
```
logr.score(fs,target_vector)
```
Out[70]: 0.9926143697117453

Random Forest

In [71]:
```
grid_search.best_score_
```
Out[71]: 0.9953902888752305

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

In [ ]:
```

```