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("2017.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH
0	2017-06-01 01:00:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	NaN
1	2017-06-01 01:00:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1.4
2	2017-06-01 01:00:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
3	2017-06-01 01:00:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	NaN
4	2017-06-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210115	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	NaN
210116	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	NaN
210117	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	NaN
210118	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	NaN
210119	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	NaN

210120 rows × 16 columns

Data Cleaning and Data Preprocessing

In [3]:

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

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

In [6]:

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

Out[6]:

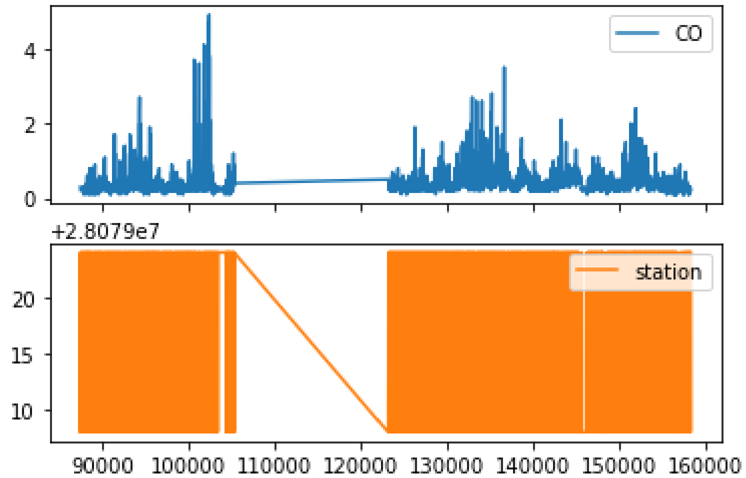
	CO	station
87457	0.3	28079008
87462	0.2	28079024
87481	0.2	28079008
87486	0.2	28079024
87505	0.2	28079008
...	...	...
158238	0.2	28079024
158257	0.3	28079008
158262	0.2	28079024
158281	0.2	28079008
158286	0.2	28079024

4127 rows × 2 columns

Line chart

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

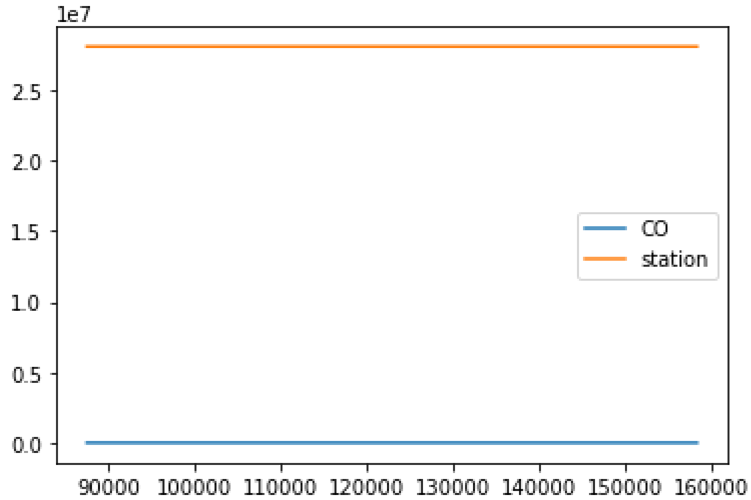
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

Out[8]: <AxesSubplot:>

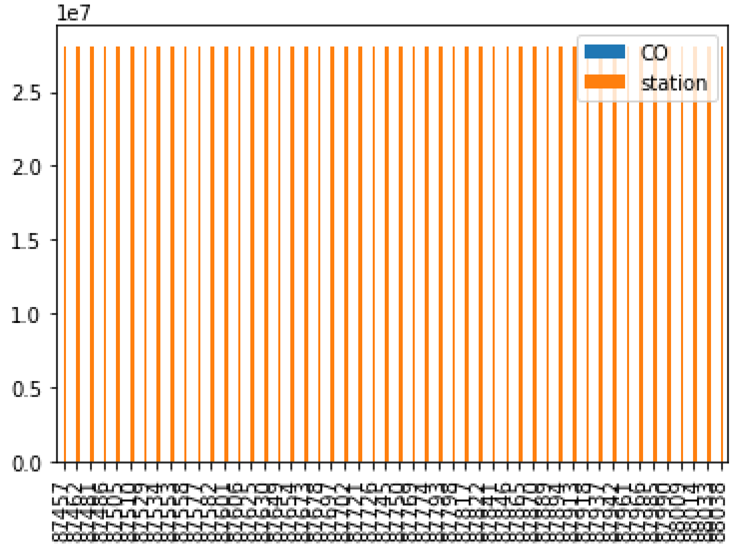


Bar chart

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

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

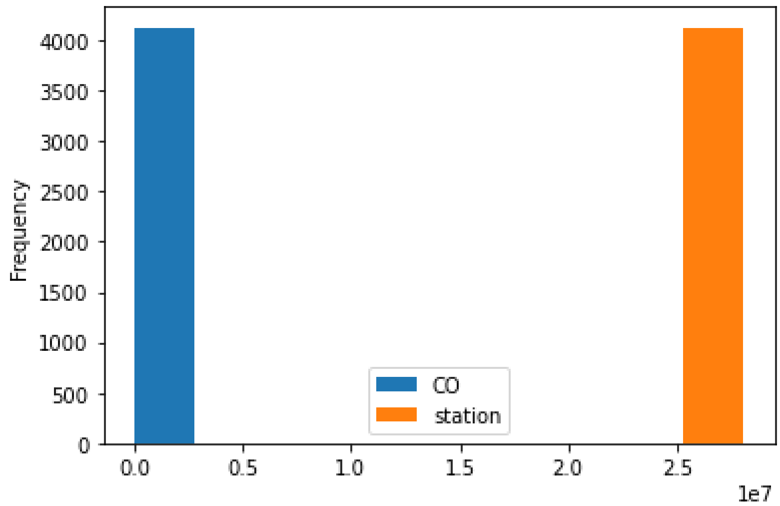
Out[10]: <AxesSubplot:>



Histogram

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

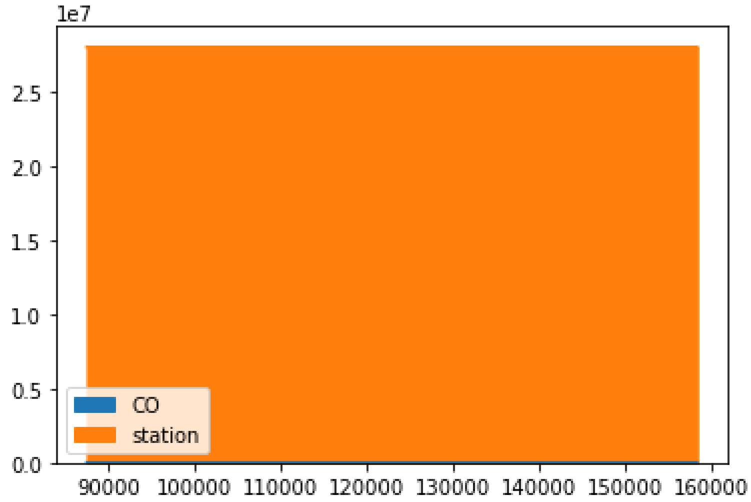
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

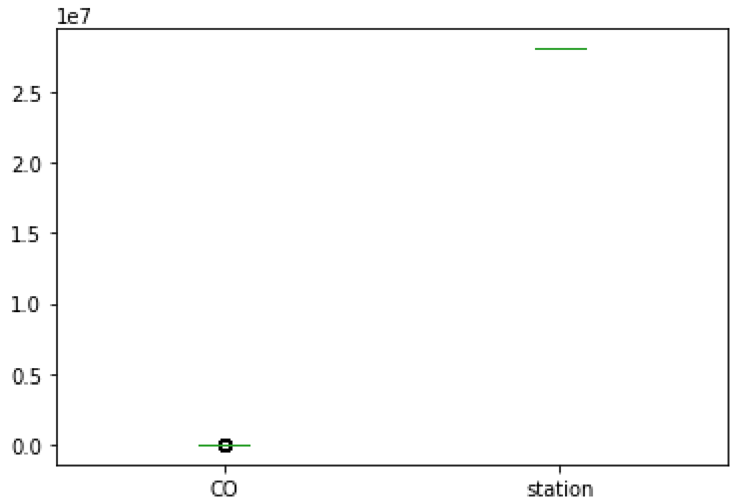
Out[12]: <AxesSubplot:>



Box chart

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

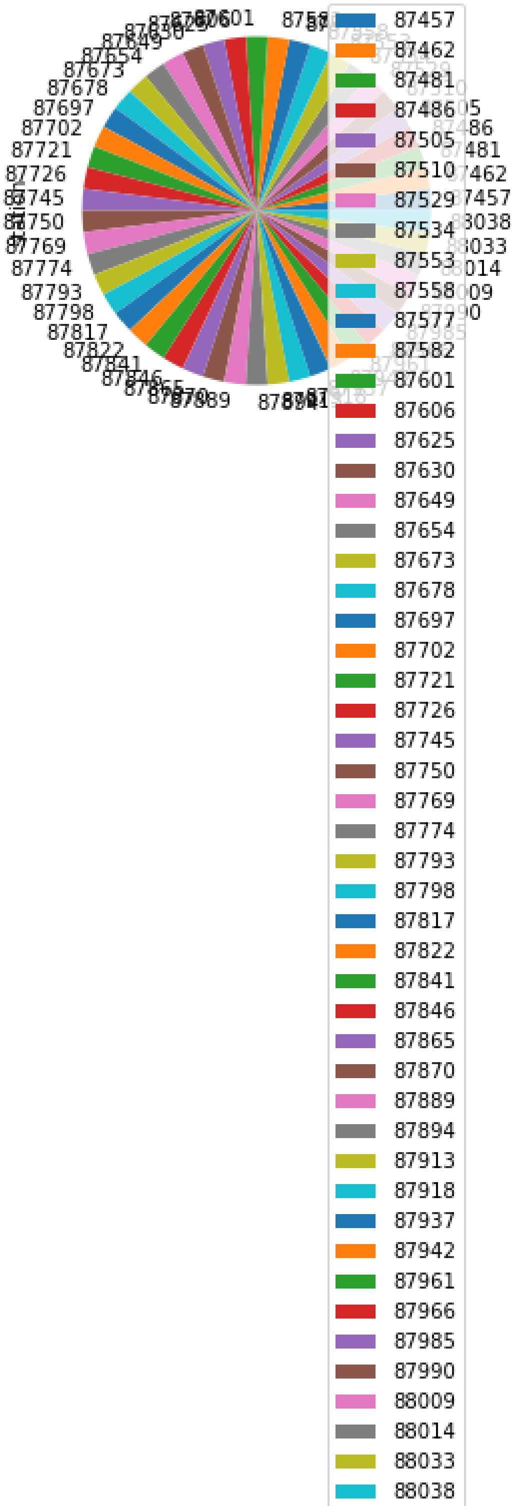
Out[13]: <AxesSubplot:>



Pie chart

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

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

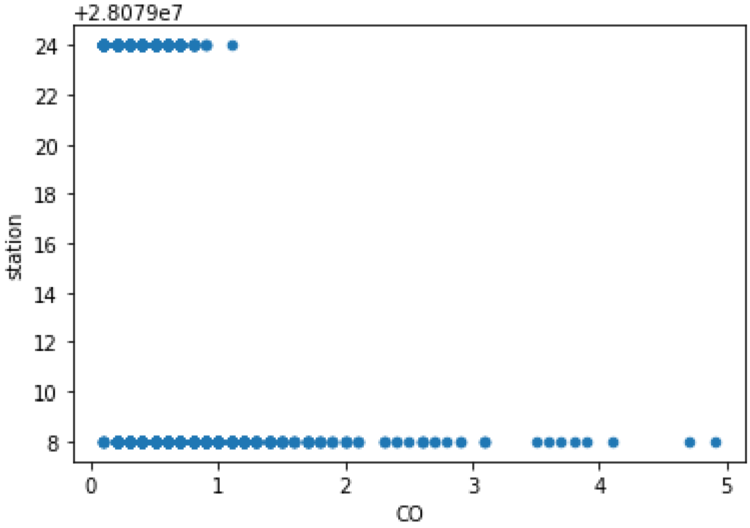


Scatter chart

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: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

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

Out[17]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

Out[18]:

	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
count	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4
mean	0.919918	1.323732	0.417858	0.578168	0.097269	41.785316	58.069057	
std	1.123078	0.215742	0.342871	0.962000	0.094035	71.118499	38.974112	
min	0.100000	1.100000	0.100000	0.100000	0.000000	1.000000	1.000000	
25%	0.300000	1.180000	0.200000	0.100000	0.050000	3.000000	30.000000	

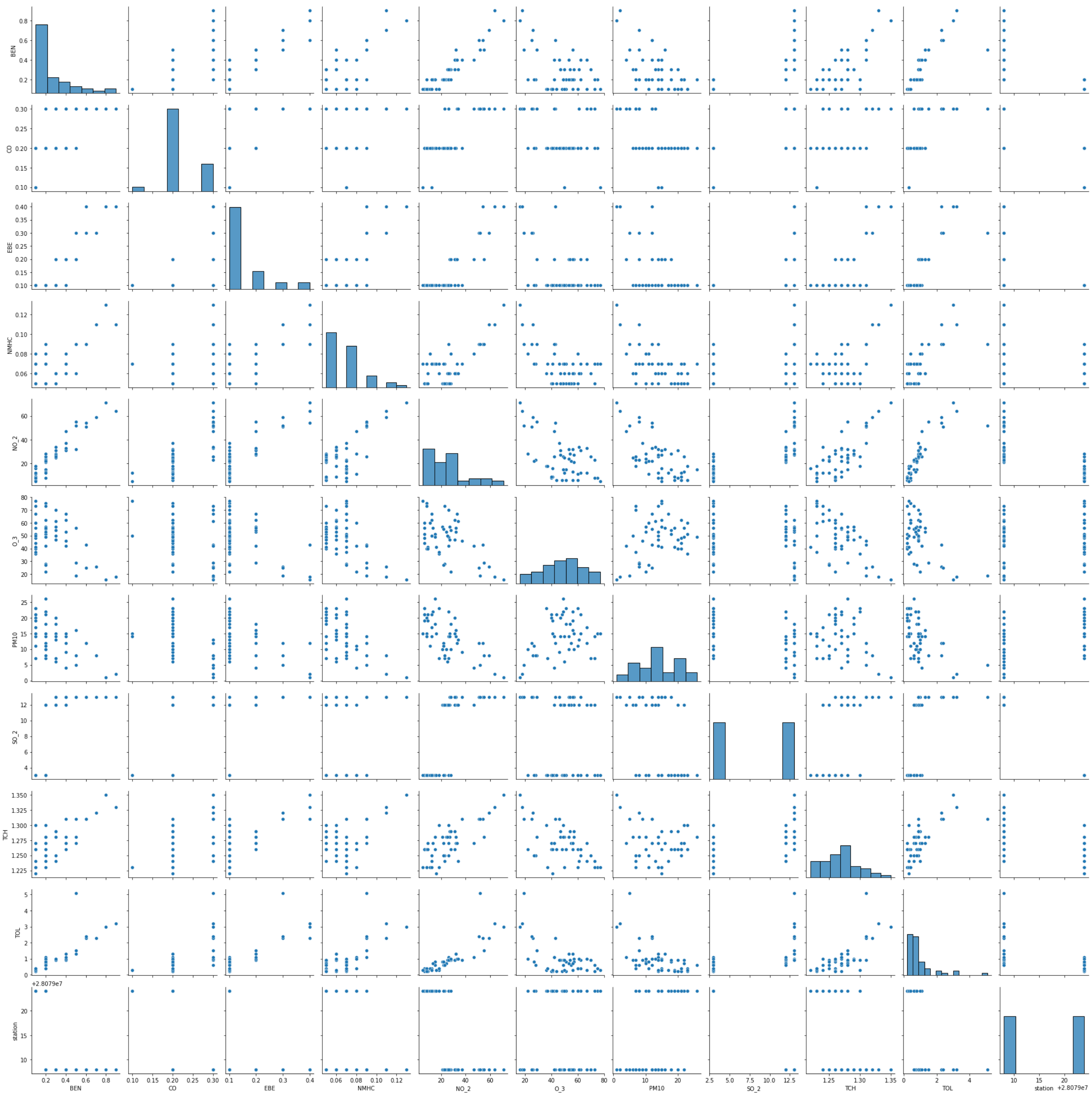
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
50%	0.600000	1.270000	0.300000	0.300000	0.080000	16.000000	54.000000	
75%	1.100000	1.400000	0.500000	0.700000	0.110000	50.000000	78.000000	
max	19.600000	3.630000	4.900000	16.700001	1.420000	879.000000	349.000000	1

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])
```

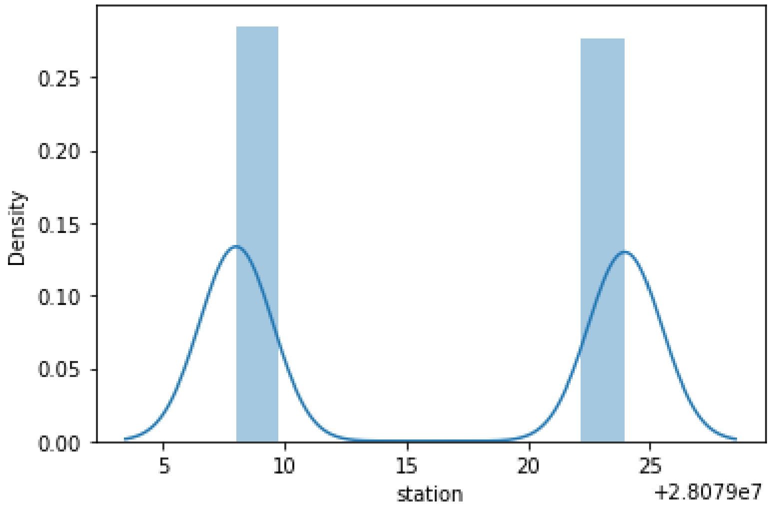
Out[20]: <seaborn.axisgrid.PairGrid at 0x2eda3b5af10>



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

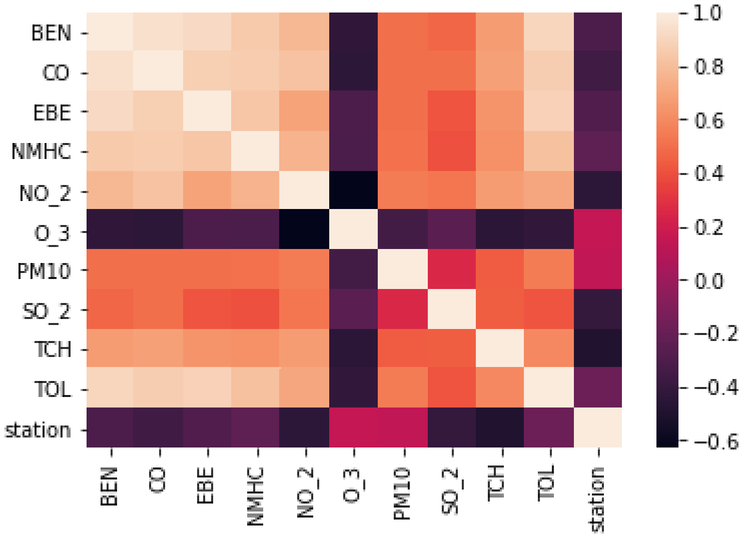
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function 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]: 28079039.742447674

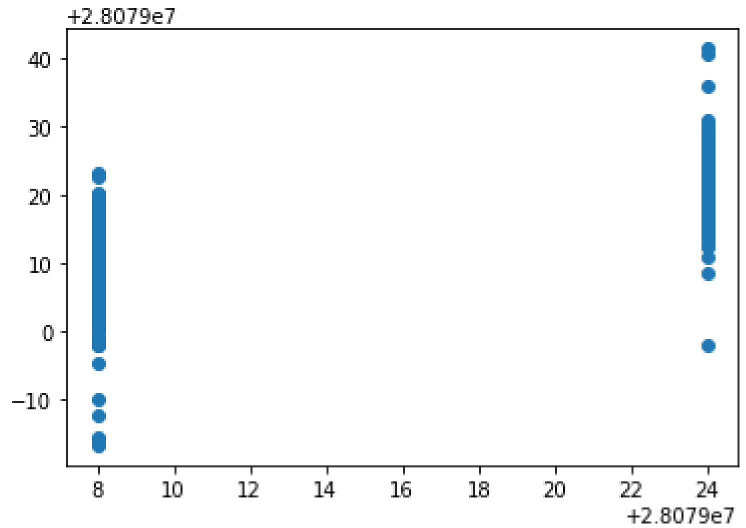
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	1.085227
CO	-2.089988
EBE	-2.720277
NMHC	36.240965
NO_2	-0.163846
O_3	-0.083555
PM10	0.353406
SO_2	-0.250277
TCH	-13.987635
TOL	0.231713

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

Out[28]: <matplotlib.collections.PathCollection at 0x2edac169580>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```
Out[29]: 0.5900771355947994

In [30]:
```
lr.score(x_train,y_train)
```
Out[30]: 0.6106445957418349

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]: 0.5715248038847451

In [34]:
```
rr.score(x_train,y_train)
```
Out[34]: 0.5938530282276366

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]: 0.40482712909937557

Accuracy(Lasso)

In [37]:
```
la.score(x_test,y_test)
```
Out[37]: 0.39778549225268334

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.        , -0.16620446,
       -0.0579603 ,  0.33131595, -0.34871237, -0.        ,  0.11185378])
```

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

Out[40]:
```
28079022.7783964
```

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

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

Out[42]:
```
0.4418261548618335
```

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)))
```

```
5.145347586020383
35.72198583137868
5.976787249967885
```

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]:
```
(4127, 10)
```

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

Out[47]:
```
(4127,)
```

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]: `0.9437848315968016`

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

Out[55]: `0.9999999999725541`

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

Out[56]: `array([[1.00000000e+00, 2.74458959e-11]])`

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]:
```
0.9698753462603877
```

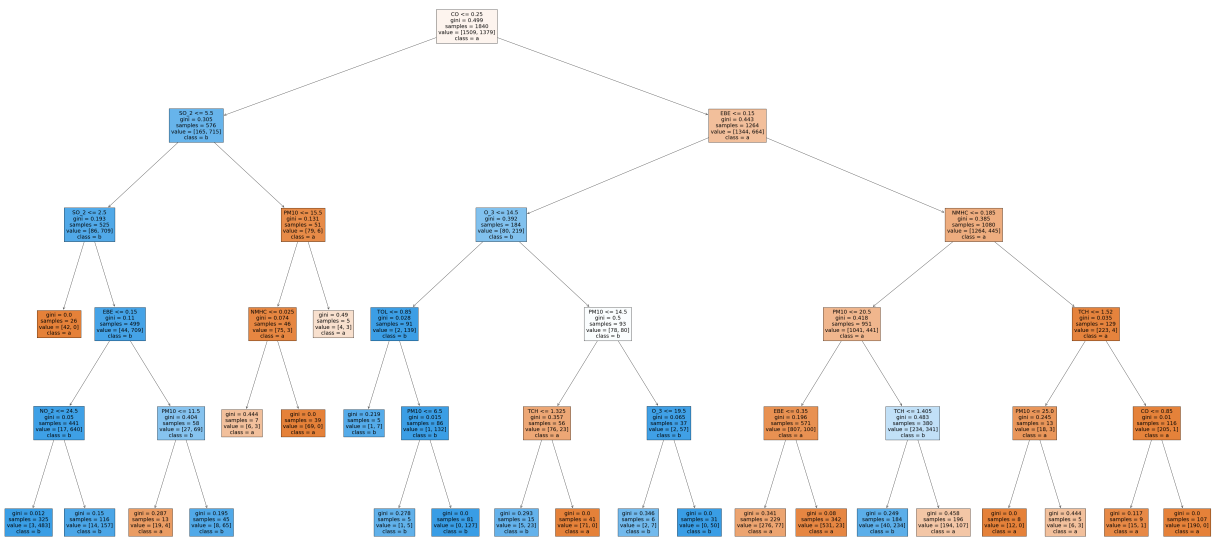
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1715.85, 1993.2, 'CO <= 0.25\ngini = 0.499\nsamples = 1840\nvalue = [1509, 137
9]\nclass = a'),
 Text(725.4, 1630.8000000000002, 'SO_2 <= 5.5\ngini = 0.305\nsamples = 576\nvalue =
[165, 715]\nclass = b'),
 Text(334.79999999999995, 1268.4, 'SO_2 <= 2.5\ngini = 0.193\nsamples = 525\nvalue =
[86, 709]\nclass = b'),
 Text(223.2, 906.0, 'gini = 0.0\nsamples = 26\nvalue = [42, 0]\nclass = a'),
 Text(446.4, 906.0, 'EBE <= 0.15\ngini = 0.11\nsamples = 499\nvalue = [44, 709]\ncla
ss = b'),
 Text(223.2, 543.5999999999999, 'NO_2 <= 24.5\ngini = 0.05\nsamples = 441\nvalue =
[17, 640]\nclass = b'),
 Text(111.6, 181.19999999999982, 'gini = 0.012\nsamples = 325\nvalue = [3, 483]\ncla
ss = b'),
 Text(334.79999999999995, 181.19999999999982, 'gini = 0.15\nsamples = 116\nvalue =
[14, 157]\nclass = b'),
 Text(669.5999999999999, 543.5999999999999, 'PM10 <= 11.5\ngini = 0.404\nsamples = 5
8\nvalue = [27, 69]\nclass = b'),
 Text(558.0, 181.19999999999982, 'gini = 0.287\nsamples = 13\nvalue = [19, 4]\nclass
= a'),
 Text(781.1999999999999, 181.19999999999982, 'gini = 0.195\nsamples = 45\nvalue =
[8, 65]\nclass = b'),
 Text(1116.0, 1268.4, 'PM10 <= 15.5\ngini = 0.131\nsamples = 51\nvalue = [79, 6]\ncl
ass = a'),
 Text(1004.4, 906.0, 'NMHC <= 0.025\ngini = 0.074\nsamples = 46\nvalue = [75, 3]\ncl
ass = a'),
 Text(892.8, 543.5999999999999, 'gini = 0.444\nsamples = 7\nvalue = [6, 3]\nclass =
a'),
 Text(1116.0, 543.5999999999999, 'gini = 0.0\nsamples = 39\nvalue = [69, 0]\nclass =
a'),
 Text(1227.6, 906.0, 'gini = 0.49\nsamples = 5\nvalue = [4, 3]\nclass = a'),
 Text(2706.2999999999997, 1630.8000000000002, 'EBE <= 0.15\ngini = 0.443\nsamples =
1264\nvalue = [1344, 664]\nclass = a'),
 Text(1841.3999999999999, 1268.4, 'O_3 <= 14.5\ngini = 0.392\nsamples = 184\nvalue =
[80, 219]\nclass = b'),
 Text(1450.8, 906.0, 'TOL <= 0.85\ngini = 0.028\nsamples = 91\nvalue = [2, 139]\ncla
ss = b'),
 Text(1339.1999999999998, 543.5999999999999, 'gini = 0.219\nsamples = 5\nvalue = [1,
7]\nclass = b'),
 Text(1562.3999999999999, 543.5999999999999, 'PM10 <= 6.5\ngini = 0.015\nsamples = 8
6\nvalue = [1, 132]\nclass = b'),
 Text(1450.8, 181.19999999999982, 'gini = 0.278\nsamples = 5\nvalue = [1, 5]\nclass
```

```
= b'),
 Text(1674.0, 181.19999999999982, 'gini = 0.0\nsamples = 81\nvalue = [0, 127]\nclass
= b'),
 Text(2232.0, 906.0, 'PM10 <= 14.5\ngini = 0.5\nsamples = 93\nvalue = [78, 80]\nclas
s = b'),
 Text(2008.8, 543.5999999999999, 'TCH <= 1.325\ngini = 0.357\nsamples = 56\nvalue =
[76, 23]\nclass = a'),
 Text(1897.1999999999998, 181.19999999999982, 'gini = 0.293\nsamples = 15\nvalue =
[5, 23]\nclass = b'),
 Text(2120.4, 181.19999999999982, 'gini = 0.0\nsamples = 41\nvalue = [71, 0]\nclass
= a'),
 Text(2455.2, 543.5999999999999, 'O_3 <= 19.5\ngini = 0.065\nsamples = 37\nvalue =
[2, 57]\nclass = b'),
 Text(2343.6, 181.19999999999982, 'gini = 0.346\nsamples = 6\nvalue = [2, 7]\nclass
= b'),
 Text(2566.7999999999997, 181.19999999999982, 'gini = 0.0\nsamples = 31\nvalue = [0,
50]\nclass = b'),
 Text(3571.2, 1268.4, 'NMHC <= 0.185\ngini = 0.385\nsamples = 1080\nvalue = [1264, 4
45]\nclass = a'),
 Text(3124.7999999999997, 906.0, 'PM10 <= 20.5\ngini = 0.418\nsamples = 951\nvalue =
[1041, 441]\nclass = a'),
 Text(2901.6, 543.5999999999999, 'EBE <= 0.35\ngini = 0.196\nsamples = 571\nvalue =
[807, 100]\nclass = a'),
 Text(2790.0, 181.19999999999982, 'gini = 0.341\nsamples = 229\nvalue = [276, 77]\nc
lass = a'),
 Text(3013.2, 181.19999999999982, 'gini = 0.08\nsamples = 342\nvalue = [531, 23]\ncl
ass = a'),
 Text(3348.0, 543.5999999999999, 'TCH <= 1.405\ngini = 0.483\nsamples = 380\nvalue =
[234, 341]\nclass = b'),
 Text(3236.3999999999996, 181.19999999999982, 'gini = 0.249\nsamples = 184\nvalue =
[40, 234]\nclass = b'),
 Text(3459.6, 181.19999999999982, 'gini = 0.458\nsamples = 196\nvalue = [194, 107]\n
class = a'),
 Text(4017.6, 906.0, 'TCH <= 1.52\ngini = 0.035\nsamples = 129\nvalue = [223, 4]\ncl
ass = a'),
 Text(3794.3999999999996, 543.5999999999999, 'PM10 <= 25.0\ngini = 0.245\nsamples =
13\nvalue = [18, 3]\nclass = a'),
 Text(3682.7999999999997, 181.19999999999982, 'gini = 0.0\nsamples = 8\nvalue = [12,
0]\nclass = a'),
 Text(3906.0, 181.19999999999982, 'gini = 0.444\nsamples = 5\nvalue = [6, 3]\nclass
= a'),
 Text(4240.8, 543.5999999999999, 'CO <= 0.85\ngini = 0.01\nsamples = 116\nvalue = [2
05, 1]\nclass = a'),
 Text(4129.2, 181.19999999999982, 'gini = 0.117\nsamples = 9\nvalue = [15, 1]\nclass
= a'),
 Text(4352.4, 181.19999999999982, 'gini = 0.0\nsamples = 107\nvalue = [190, 0]\nclas
s = a')]
```



Conclusion

Scores

Linear Regression

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

Out[64]: 0.5900771355947994

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

Out[65]: 0.6106445957418349

Lasso

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

Out[66]: 0.39778549225268334

Ridge

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

Out[67]: 0.5715248038847451

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

Out[68]: 0.5938530282276366

Elastic Net

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

Out[69]: 0.4418261548618335

Logistic Regression

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

Out[70]: 0.9437848315968016

Random Forest

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

Out[71]:
```
0.9698753462603877
```

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

In [ ]:
```

```