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("2016.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	s
0	2016-11-01 01:00:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	28C
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	28C
2	2016-11-01 01:00:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	28C
3	2016-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	28C
4	2016-11-01 01:00:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	28C
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
209491	2016-07-01 00:00:00	NaN	0.2	NaN	NaN	2.0	29.0	73.0	NaN	NaN	NaN	NaN	NaN	28C
209492	2016-07-01 00:00:00	NaN	0.3	NaN	NaN	1.0	29.0	NaN	36.0	NaN	5.0	NaN	NaN	28C
209493	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	1.0	19.0	71.0	NaN	NaN	NaN	NaN	NaN	28C
209494	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	6.0	17.0	85.0	NaN	NaN	NaN	NaN	NaN	28C
209495	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	2.0	46.0	61.0	34.0	NaN	NaN	NaN	NaN	28C

209496 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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[6]:

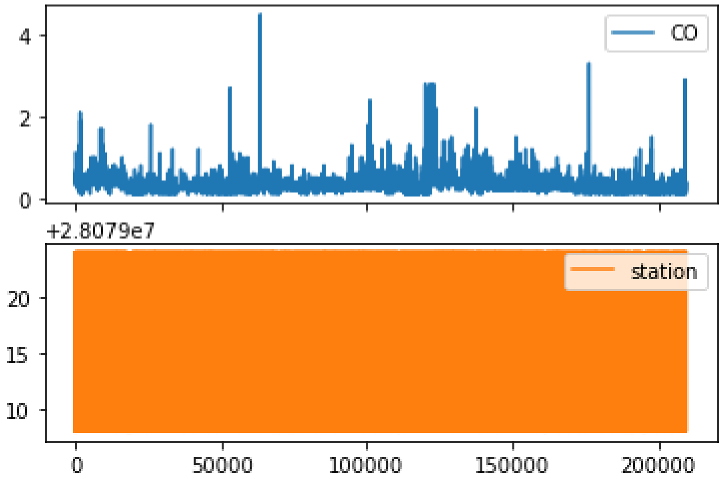
	CO	station
1	1.1	28079008
6	0.8	28079024
25	1.0	28079008
30	0.7	28079024
49	0.8	28079008
...	...	...
209430	0.2	28079024
209449	0.4	28079008
209454	0.2	28079024
209473	0.4	28079008
209478	0.2	28079024

16932 rows × 2 columns

Line chart

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

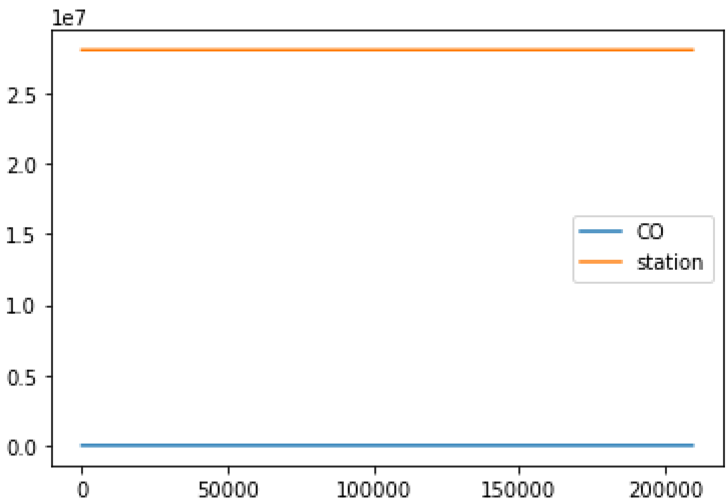
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]:
```
<AxesSubplot:>
```

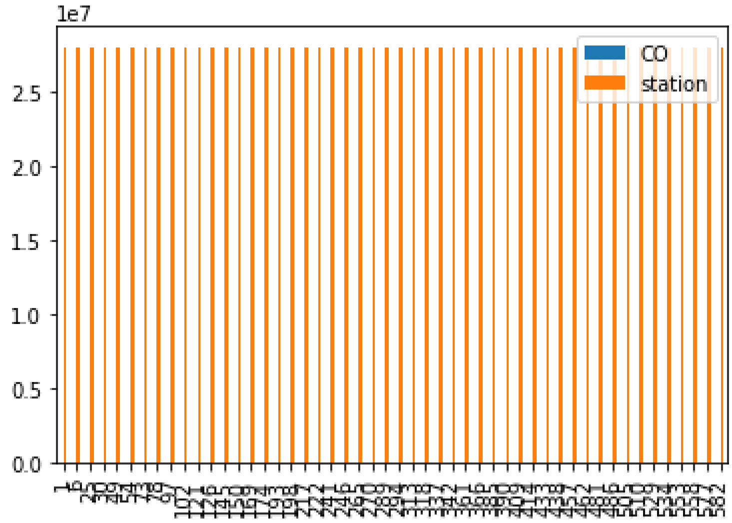


Bar chart

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

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

Out[10]:
```
<AxesSubplot:>
```

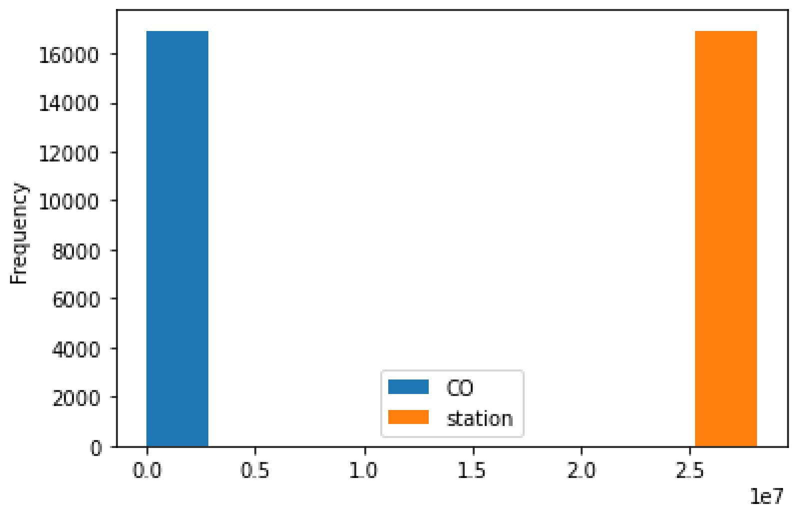


Histogram

In [11]:

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

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

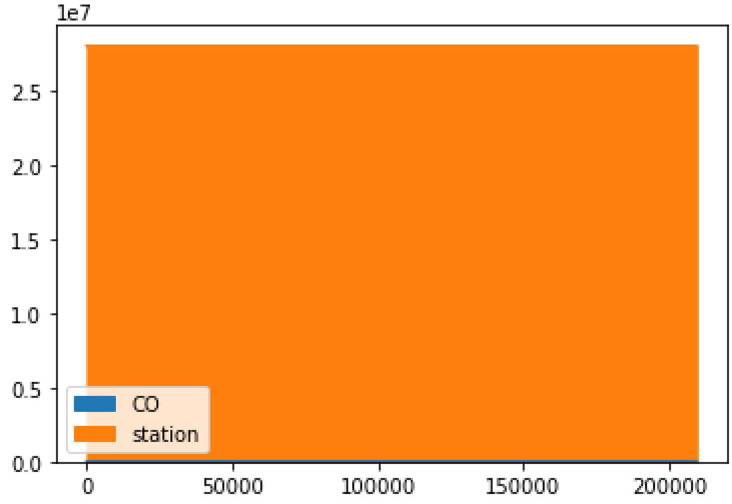


Area chart

In [12]:

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

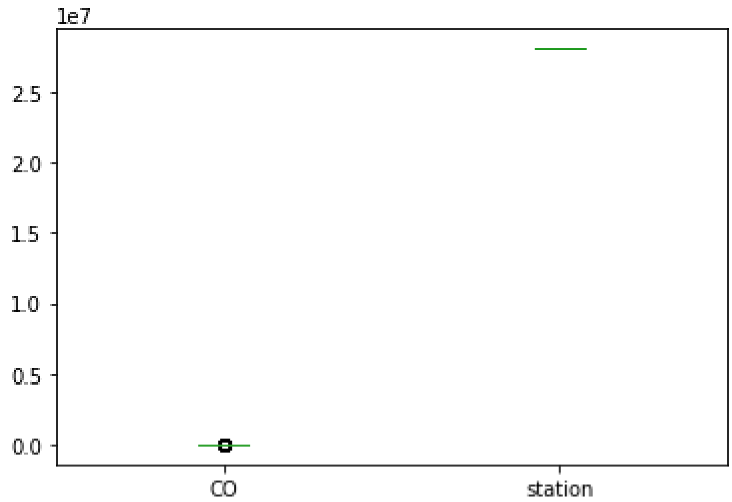
Out[12]: <AxesSubplot:>



Box chart

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

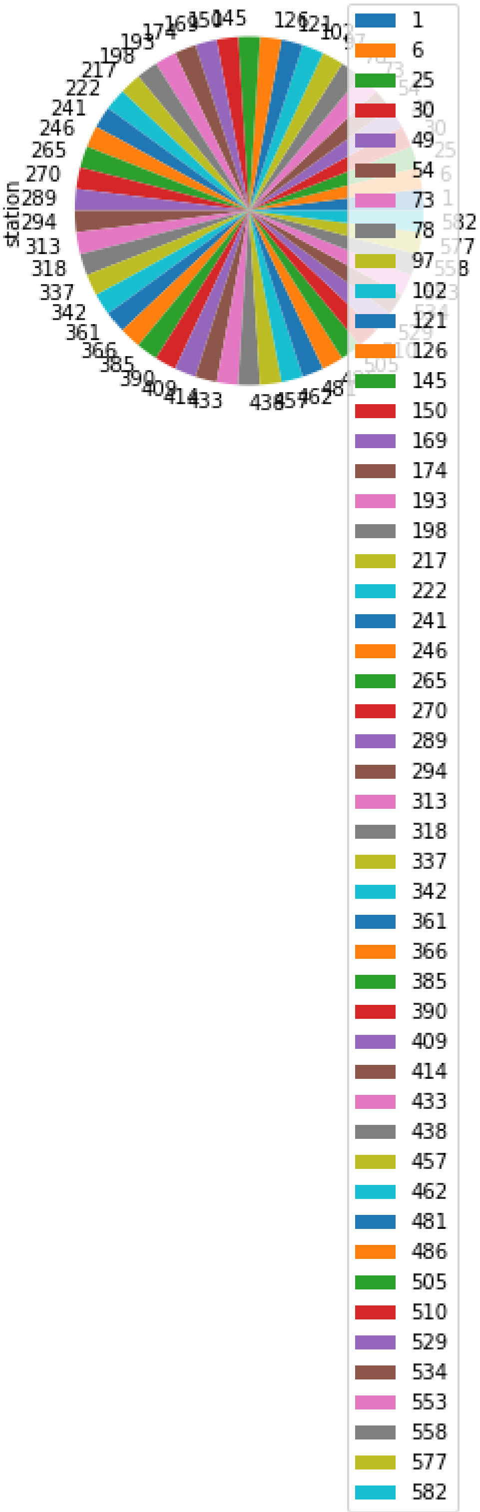
Out[13]: <AxesSubplot:>



Pie chart

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

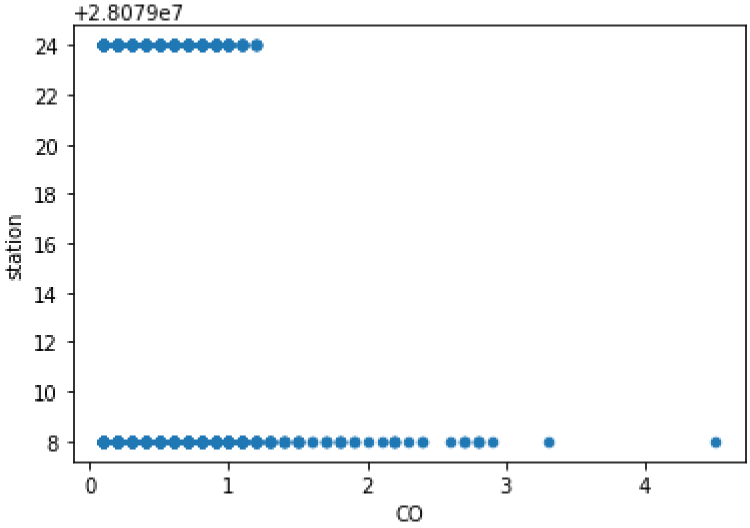
Out[14]: <AxesSubplot:ylabel='station'>



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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ 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	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.0C
mean	0.537970	0.349941	0.298955	0.099913	20.815734	39.373376	48.11
std	0.599479	0.203807	0.450204	0.079850	40.986063	31.170307	32.5€
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.0C
25%	0.200000	0.200000	0.100000	0.050000	1.000000	14.000000	21.0C
50%	0.400000	0.300000	0.200000	0.090000	7.000000	34.000000	46.0C
75%	0.700000	0.400000	0.300000	0.120000	23.000000	58.000000	69.0C

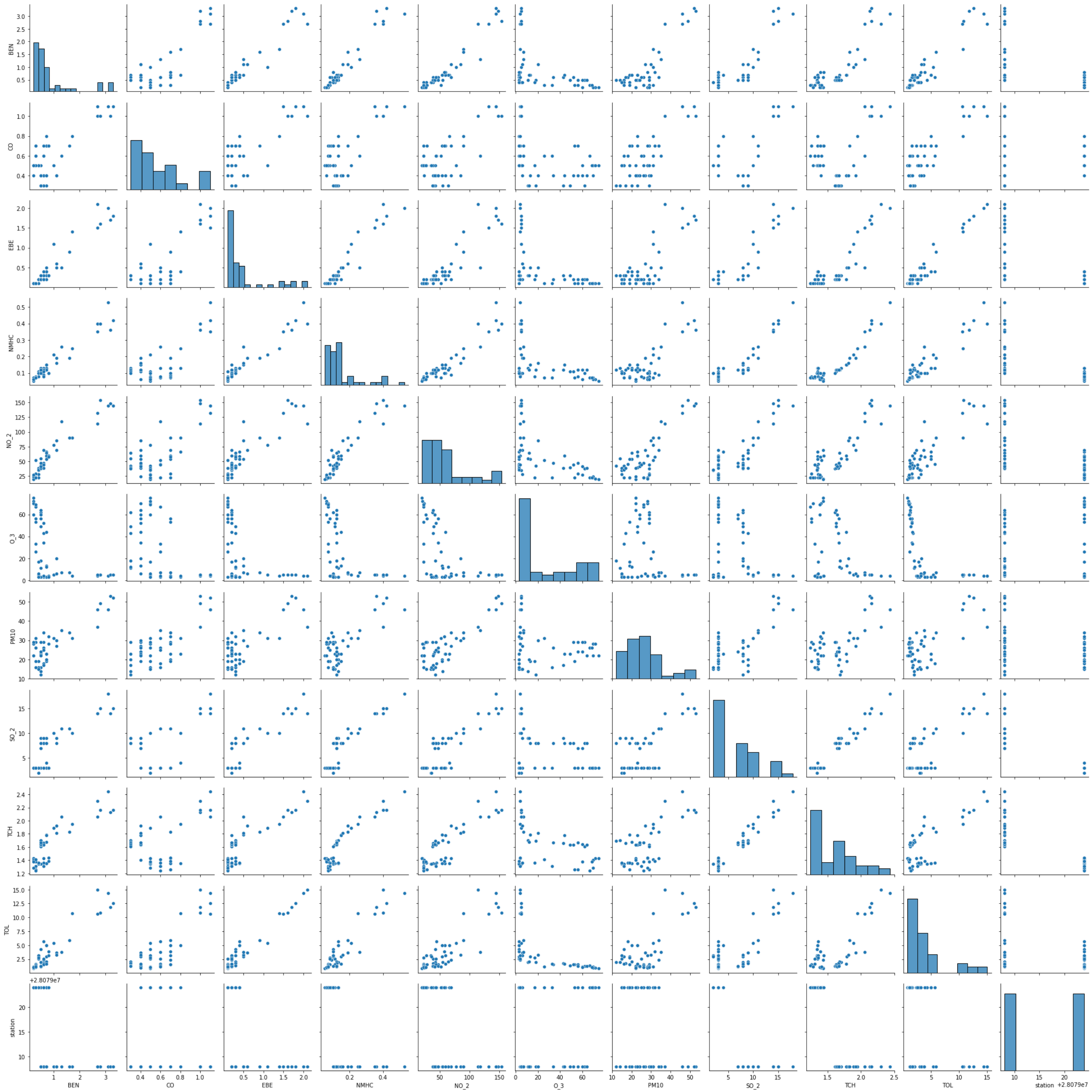
	BEN	CO	EBE	NMHC	NO	NO_2	
max	12.300000	4.500000	13.500000	2.210000	829.000000	319.000000	181.00

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 0x217d78acc70>

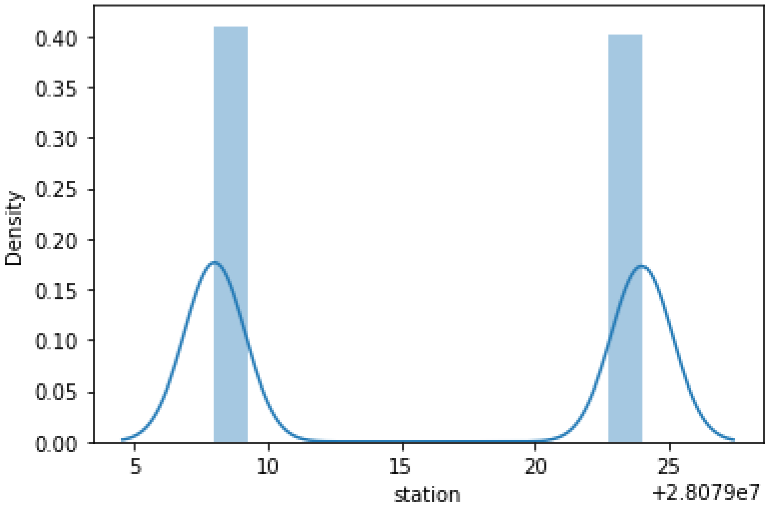


In [21]:
```
sns.distplot(df1['station'])
```

```
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
```

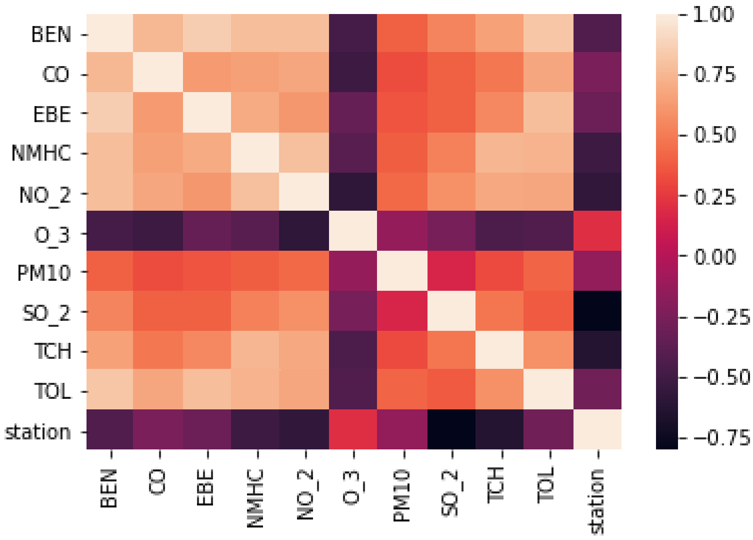
```
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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]: 28079040.05833833

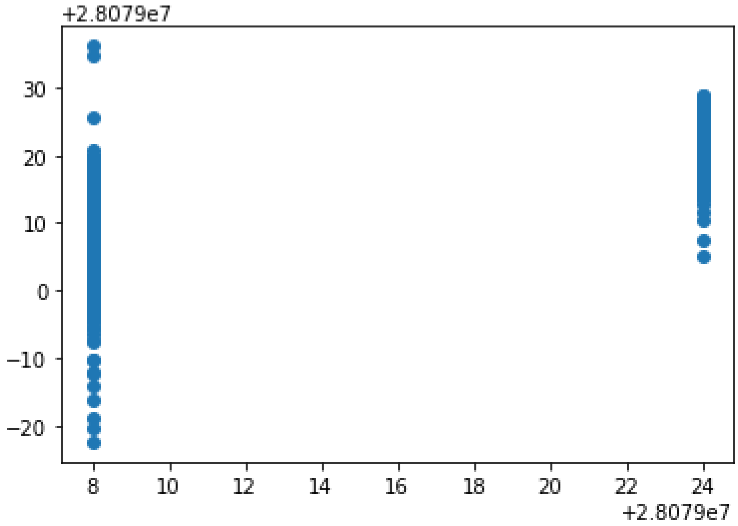
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	1.799305
CO	5.866820
EBE	0.210592
NMHC	3.652970
NO_2	-0.067171
O_3	-0.028079
PM10	0.024436
SO_2	-0.844531
TCH	-12.959269
TOL	0.258683

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

Out[28]: <matplotlib.collections.PathCollection at 0x217e0d9bee0>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```
Out[29]: 0.8011739192797894

In [30]:
```
lr.score(x_train,y_train)
```
Out[30]: 0.7916349423598931

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]: 0.8004456471536664

In [34]:
```
rr.score(x_train,y_train)
```
Out[34]: 0.7915061315853162

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]: 0.6160284065614383

Accuracy(Lasso)

In [37]:
```
la.score(x_test,y_test)
```
Out[37]: 0.6262405487220697

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.        , -0.07934312,
       -0.02660259,  0.02377698, -0.85401015, -0.        ,  0.2541252 ])
```

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

Out[40]:
```
28079025.817570496
```

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

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

Out[42]:
```
0.686409247096778
```

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)))
```

```
3.5275189590574367
20.069655755308172
4.4799169362063145
```

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]:
```
(16932, 10)
```

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

Out[47]:
```
(16932,)
```

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]:
```
0.9923812898653437
```

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

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

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

Out[56]:
```
array([[1.0000000e+00, 1.6336121e-46]])
```

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]:
```
0.9946000674991562
```

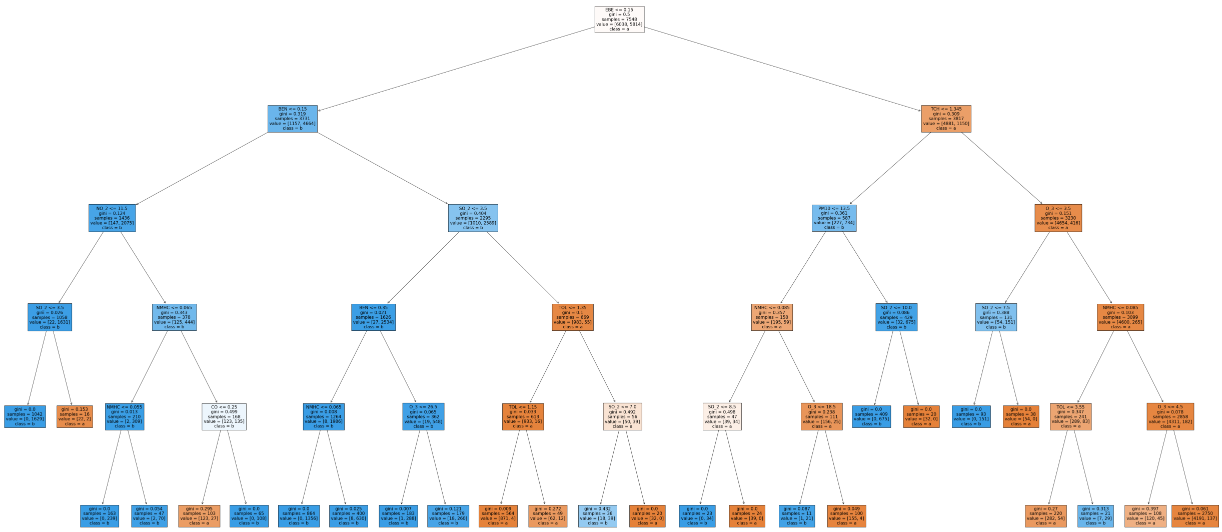
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(2266.1632653061224, 1993.2, 'EBE <= 0.15\ngini = 0.5\nsamples = 7548\nvalue =
[6038, 5814]\nclass = a'),
 Text(1070.4489795918369, 1630.8000000000002, 'BEN <= 0.15\ngini = 0.319\nsamples =
3731\nvalue = [1157, 4664]\nclass = b'),
 Text(409.9591836734694, 1268.4, 'NO_2 <= 11.5\ngini = 0.124\nsamples = 1436\nvalue
= [147, 2075]\nclass = b'),
 Text(182.20408163265307, 906.0, 'SO_2 <= 3.5\ngini = 0.026\nsamples = 1058\nvalue =
[22, 1631]\nclass = b'),
 Text(91.10204081632654, 543.5999999999999, 'gini = 0.0\nsamples = 1042\nvalue = [0,
1629]\nclass = b'),
 Text(273.30612244897964, 543.5999999999999, 'gini = 0.153\nsamples = 16\nvalue = [2
2, 2]\nclass = a'),
 Text(637.7142857142858, 906.0, 'NMHC <= 0.065\ngini = 0.343\nsamples = 378\nvalue =
[125, 444]\nclass = b'),
 Text(455.51020408163265, 543.5999999999999, 'NMHC <= 0.055\ngini = 0.013\nsamples =
210\nvalue = [2, 309]\nclass = b'),
 Text(364.40816326530614, 181.19999999999982, 'gini = 0.0\nsamples = 163\nvalue =
[0, 239]\nclass = b'),
 Text(546.6122448979593, 181.19999999999982, 'gini = 0.054\nsamples = 47\nvalue =
[2, 70]\nclass = b'),
 Text(819.9183673469388, 543.5999999999999, 'CO <= 0.25\ngini = 0.499\nsamples = 168
\nvalue = [123, 135]\nclass = b'),
 Text(728.8163265306123, 181.19999999999982, 'gini = 0.295\nsamples = 103\nvalue =
[123, 27]\nclass = a'),
 Text(911.0204081632653, 181.19999999999982, 'gini = 0.0\nsamples = 65\nvalue = [0,
108]\nclass = b'),
 Text(1730.938775510204, 1268.4, 'SO_2 <= 3.5\ngini = 0.404\nsamples = 2295\nvalue =
[1010, 2589]\nclass = b'),
 Text(1366.530612244898, 906.0, 'BEN <= 0.35\ngini = 0.021\nsamples = 1626\nvalue =
[27, 2534]\nclass = b'),
 Text(1184.326530612245, 543.5999999999999, 'NMHC <= 0.065\ngini = 0.008\nsamples =
1264\nvalue = [8, 1986]\nclass = b'),
 Text(1093.2244897959185, 181.19999999999982, 'gini = 0.0\nsamples = 864\nvalue =
[0, 1356]\nclass = b'),
 Text(1275.4285714285716, 181.19999999999982, 'gini = 0.025\nsamples = 400\nvalue =
[8, 630]\nclass = b'),
 Text(1548.734693877551, 543.5999999999999, 'O_3 <= 26.5\ngini = 0.065\nsamples = 36
2\nvalue = [19, 548]\nclass = b'),
 Text(1457.6326530612246, 181.19999999999982, 'gini = 0.007\nsamples = 183\nvalue =
[1, 288]\nclass = b'),
 Text(1639.8367346938776, 181.19999999999982, 'gini = 0.121\nsamples = 179\nvalue =
```

```
[18, 260]\nclass = b'),
 Text(2095.3469387755104, 906.0, 'TOL <= 1.35\ngini = 0.1\nsamples = 669\nvalue = [9
83, 55]\nclass = a'),
 Text(1913.1428571428573, 543.5999999999999, 'TOL <= 1.15\ngini = 0.033\nsamples = 6
13\nvalue = [933, 16]\nclass = a'),
 Text(1822.0408163265306, 181.19999999999982, 'gini = 0.009\nsamples = 564\nvalue =
[871, 4]\nclass = a'),
 Text(2004.2448979591838, 181.19999999999982, 'gini = 0.272\nsamples = 49\nvalue =
[62, 12]\nclass = a'),
 Text(2277.5510204081634, 543.5999999999999, 'SO_2 <= 7.0\ngini = 0.492\nsamples = 5
6\nvalue = [50, 39]\nclass = a'),
 Text(2186.448979591837, 181.19999999999982, 'gini = 0.432\nsamples = 36\nvalue = [1
8, 39]\nclass = b'),
 Text(2368.65306122449, 181.19999999999982, 'gini = 0.0\nsamples = 20\nvalue = [32,
0]\nclass = a'),
 Text(3461.877551020408, 1630.8000000000002, 'TCH <= 1.345\ngini = 0.309\nsamples =
3817\nvalue = [4881, 1150]\nclass = a'),
 Text(3051.918367346939, 1268.4, 'PM10 <= 13.5\ngini = 0.361\nsamples = 587\nvalue =
[227, 734]\nclass = b'),
 Text(2824.1632653061224, 906.0, 'NMHC <= 0.085\ngini = 0.357\nsamples = 158\nvalue
= [195, 59]\nclass = a'),
 Text(2641.9591836734694, 543.5999999999999, 'SO_2 <= 8.5\ngini = 0.498\nsamples = 4
7\nvalue = [39, 34]\nclass = a'),
 Text(2550.857142857143, 181.19999999999982, 'gini = 0.0\nsamples = 23\nvalue = [0,
34]\nclass = b'),
 Text(2733.061224489796, 181.19999999999982, 'gini = 0.0\nsamples = 24\nvalue = [39,
0]\nclass = a'),
 Text(3006.367346938776, 543.5999999999999, 'O_3 <= 18.5\ngini = 0.238\nsamples = 11
1\nvalue = [156, 25]\nclass = a'),
 Text(2915.265306122449, 181.19999999999982, 'gini = 0.087\nsamples = 11\nvalue =
[1, 21]\nclass = b'),
 Text(3097.469387755102, 181.19999999999982, 'gini = 0.049\nsamples = 100\nvalue =
[155, 4]\nclass = a'),
 Text(3279.673469387755, 906.0, 'SO_2 <= 10.0\ngini = 0.086\nsamples = 429\nvalue =
[32, 675]\nclass = b'),
 Text(3188.571428571429, 543.5999999999999, 'gini = 0.0\nsamples = 409\nvalue = [0,
675]\nclass = b'),
 Text(3370.775510204082, 543.5999999999999, 'gini = 0.0\nsamples = 20\nvalue = [32,
0]\nclass = a'),
 Text(3871.8367346938776, 1268.4, 'O_3 <= 3.5\ngini = 0.151\nsamples = 3230\nvalue =
[4654, 416]\nclass = a'),
 Text(3644.081632653061, 906.0, 'SO_2 <= 7.5\ngini = 0.388\nsamples = 131\nvalue =
[54, 151]\nclass = b'),
 Text(3552.979591836735, 543.5999999999999, 'gini = 0.0\nsamples = 93\nvalue = [0, 1
51]\nclass = b'),
 Text(3735.183673469388, 543.5999999999999, 'gini = 0.0\nsamples = 38\nvalue = [54,
0]\nclass = a'),
 Text(4099.591836734694, 906.0, 'NMHC <= 0.085\ngini = 0.103\nsamples = 3099\nvalue
= [4600, 265]\nclass = a'),
 Text(3917.387755102041, 543.5999999999999, 'TOL <= 3.55\ngini = 0.347\nsamples = 24
1\nvalue = [289, 83]\nclass = a'),
 Text(3826.2857142857147, 181.19999999999982, 'gini = 0.27\nsamples = 220\nvalue =
[282, 54]\nclass = a'),
 Text(4008.4897959183677, 181.19999999999982, 'gini = 0.313\nsamples = 21\nvalue =
[7, 29]\nclass = b'),
 Text(4281.795918367347, 543.5999999999999, 'O_3 <= 4.5\ngini = 0.078\nsamples = 285
8\nvalue = [4311, 182]\nclass = a'),
 Text(4190.693877551021, 181.19999999999982, 'gini = 0.397\nsamples = 108\nvalue =
[120, 45]\nclass = a'),
 Text(4372.897959183674, 181.19999999999982, 'gini = 0.061\nsamples = 2750\nvalue =
[4191, 137]\nclass = a')]
```



Conclusion

Scores

Linear Regression

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

Out[64]: 0.8011739192797894

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

Out[65]: 0.7916349423598931

Lasso

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

Out[66]: 0.6262405487220697

Ridge

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

Out[67]: 0.8004456471536664

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

Out[68]: 0.7915061315853162

Elastic Net

In [69]:

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

Out[69]: 0.686409247096778

Logistic Regression

In [70]:

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

Out[70]: 0.9923812898653437

Random Forest

In [71]:

```
grid_search.best_score_
```

Out[71]: 0.9946000674991562

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

In [ ]:

```

```