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("2012.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	st
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	2807
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	2807
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	2807
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	2807
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	2807
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210715	2012-03-01 00:00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN	2807
210716	2012-03-01 00:00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN	2807
210717	2012-03-01 00:00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN	2807
210718	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN	2807
210719	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN	2807

210720 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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

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

Out[6]:

	CO	station
6	0.2	28079024
30	0.2	28079024
54	0.2	28079024
78	0.2	28079024
102	0.2	28079024
...	...	...
210654	0.3	28079024
210673	0.4	28079008
210678	0.3	28079024
210697	0.4	28079008
210702	0.3	28079024

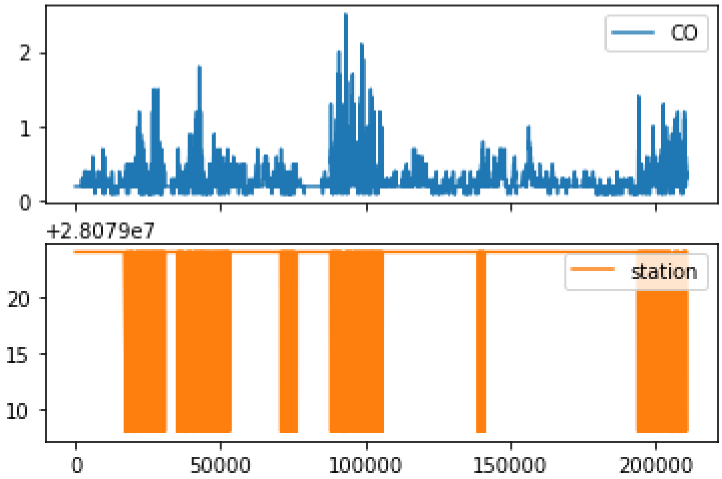
10916 rows × 2 columns

Line chart

In [7]:

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

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

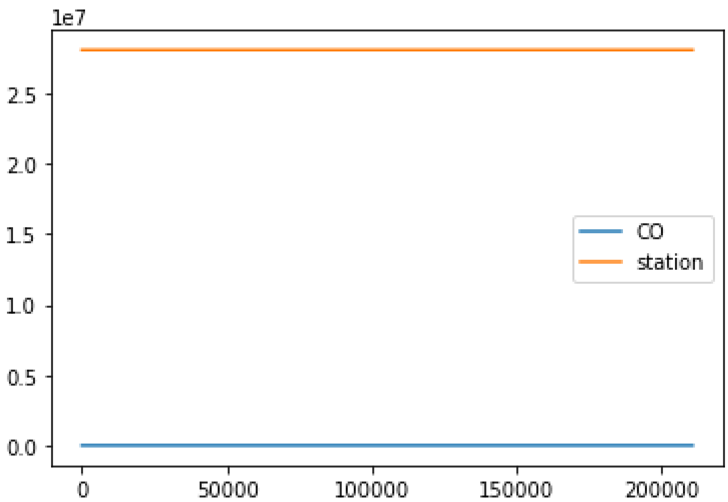


Line chart

In [8]:

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

Out[8]: <AxesSubplot:>



Bar chart

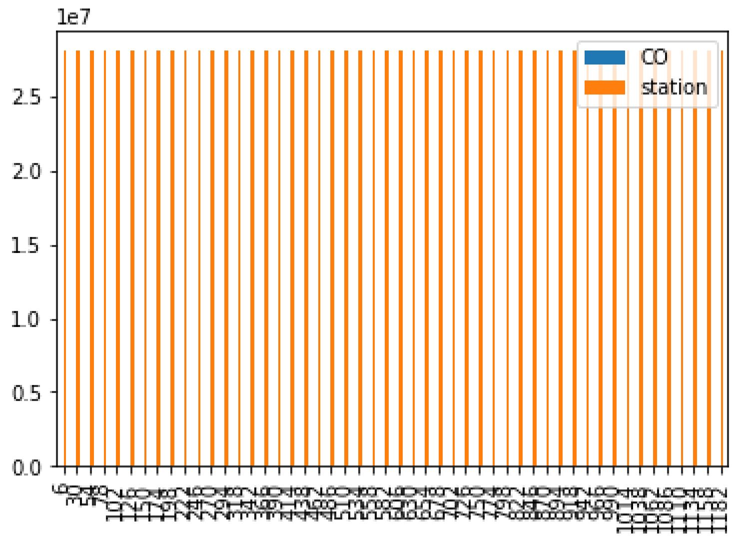
In [9]:

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

In [10]:

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

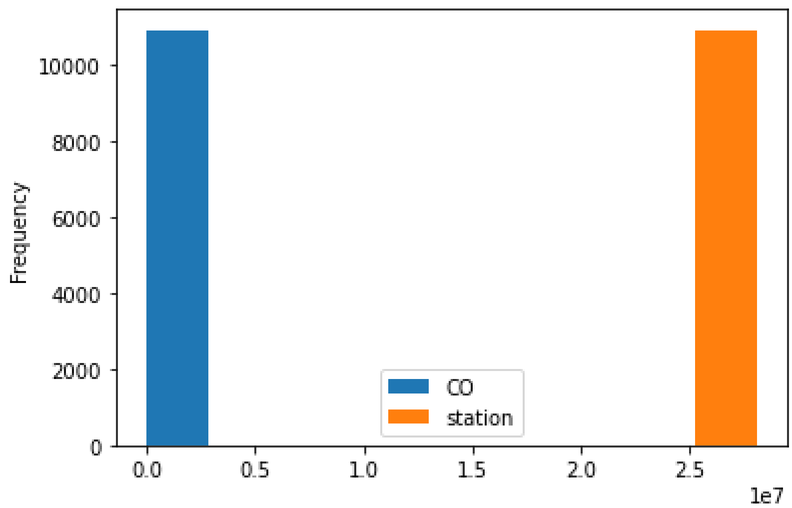
Out[10]: <AxesSubplot:>



Histogram

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

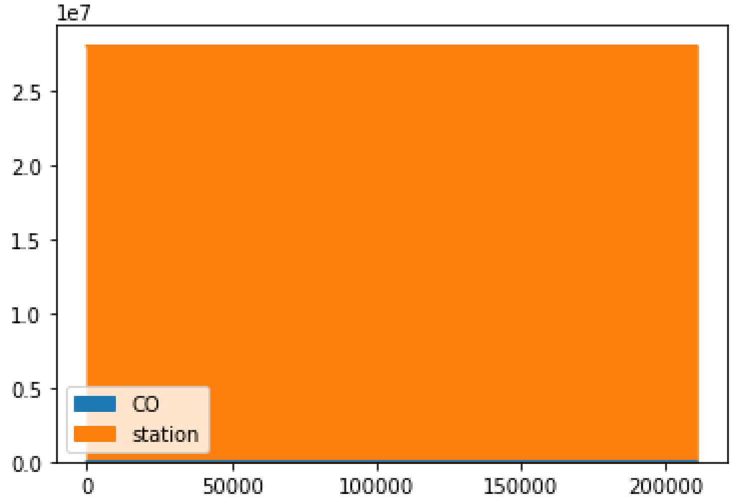
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

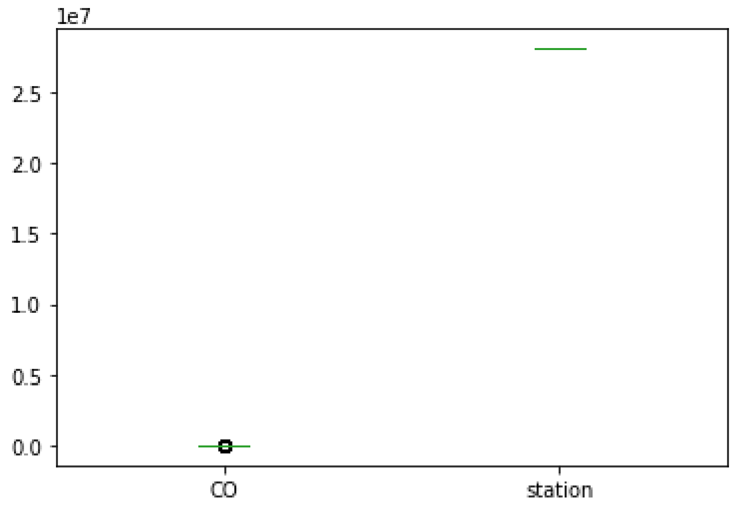
Out[12]: <AxesSubplot:>



Box chart

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

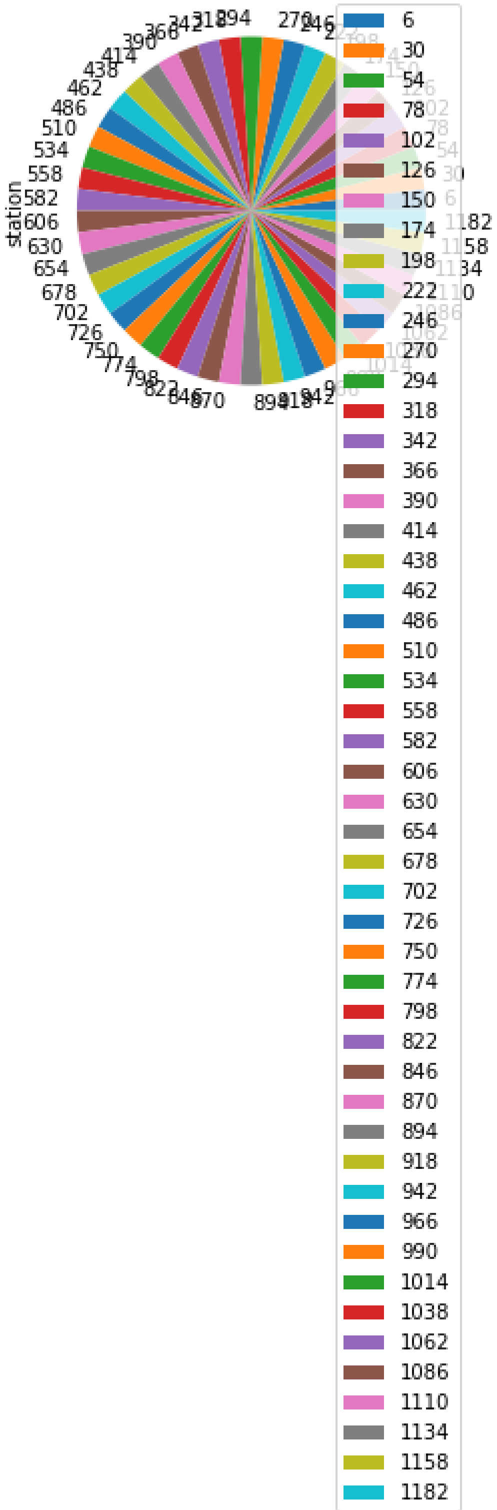
Out[13]: <AxesSubplot:>



Pie chart

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

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

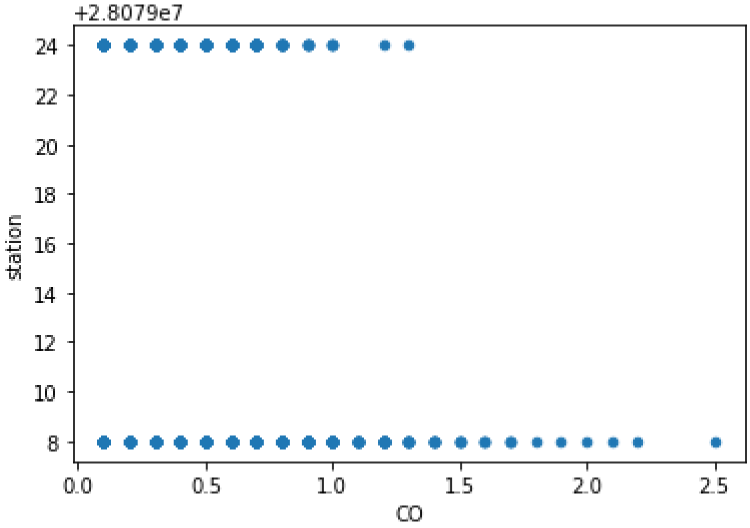


Scatter chart

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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

In [19]:

```
df.columns
```

Out[19]:

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

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	
count	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.0C
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642	44.23
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732	29.53
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000	1.0C
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000	18.0C
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000	44.0C
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000	65.0C

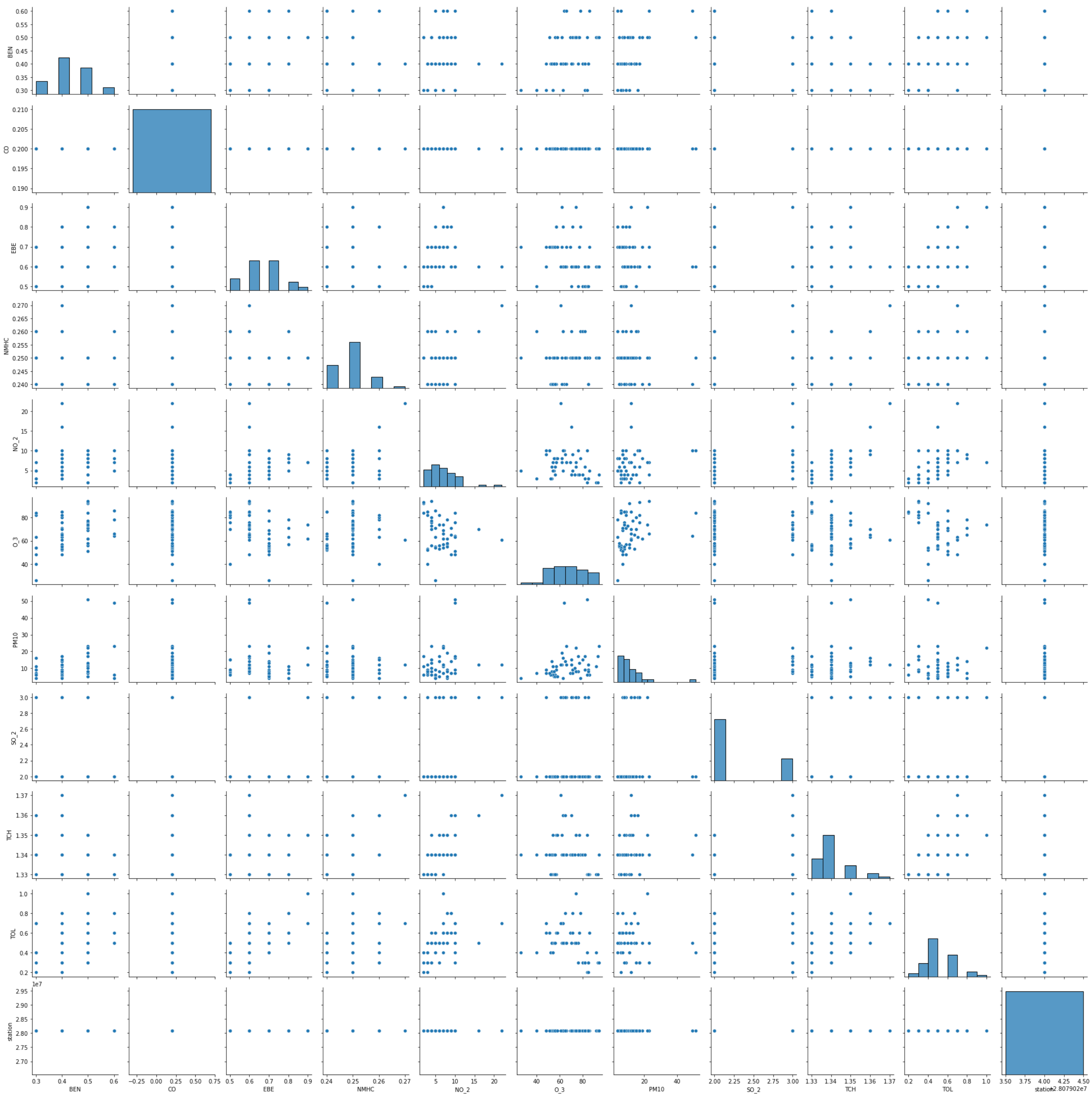
	BEN	CO	EBE	NMHC	NO	NO_2	
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000	157.0C

In [29]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2','O_3',
       'PM10','SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [30]:
```
sns.pairplot(df1[0:50])
```

Out[30]: <seaborn.axisgrid.PairGrid at 0x22b58e44c70>

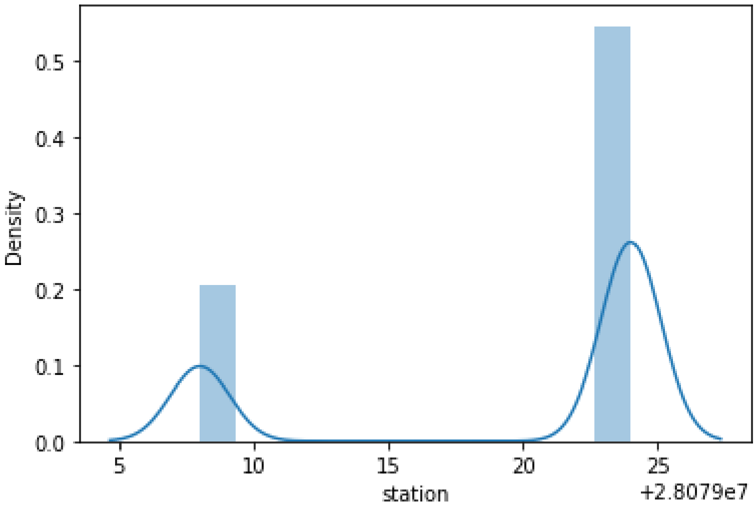


In [31]:
```
sns.distplot(df1['station'])
```

```
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
```

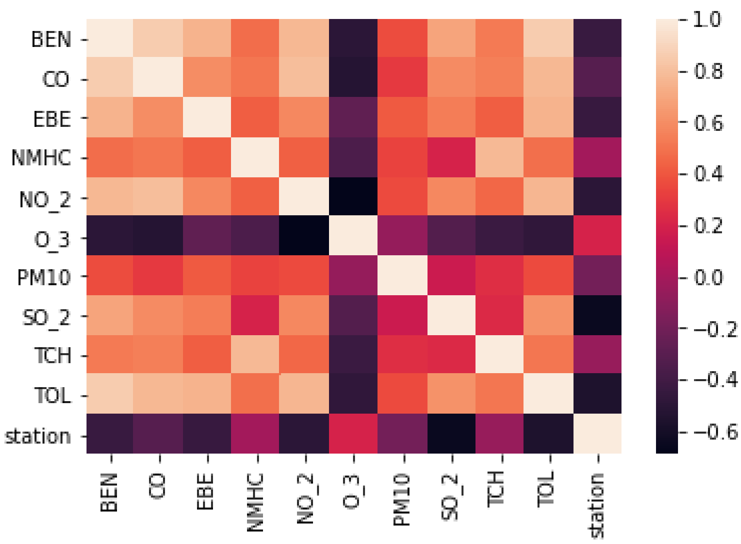
```
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[31]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [32]:
```
sns.heatmap(df1.corr())
```

Out[32]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [33]:
```
x=df[['BEN', 'CO', 'EBE','NMHC', 'NO_2','O_3',
       'PM10','SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [34]:
```
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 [35]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[35]: LinearRegression()

In [36]:
```
lr.intercept_
```

Out[36]: 28079019.165581908

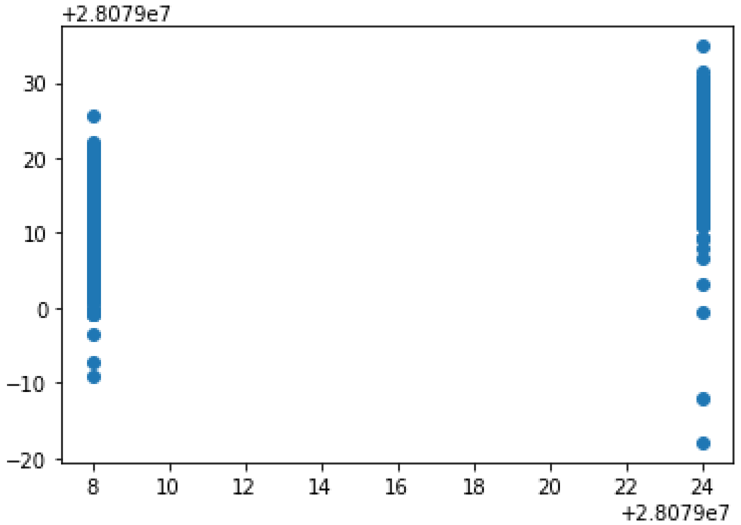
In [37]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[37]:

	Co-efficient
BEN	3.571543
CO	17.031631
EBE	-0.394357
NMHC	17.056148
NO_2	-0.116879
O_3	-0.030933
PM10	-0.008837
SO_2	-0.695280
TCH	1.217509
TOL	-1.466106

In [38]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[38]: <matplotlib.collections.PathCollection at 0x22b63c44df0>



ACCURACY

In [39]:
```
lr.score(x_test,y_test)
```
Out[39]: 0.6086004152156592

In [40]:
```
lr.score(x_train,y_train)
```
Out[40]: 0.6273759315249119

Ridge and Lasso

In [41]:
```
from sklearn.linear_model import Ridge,Lasso
```
In [42]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```
Out[42]: Ridge(alpha=10)

Accuracy(Ridge)

In [43]:
```
rr.score(x_test,y_test)
```
Out[43]: 0.6048567528892628

In [44]:
```
rr.score(x_train,y_train)
```
Out[44]: 0.6249236689518839

In [45]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[45]: Lasso(alpha=10)

In [46]:
```
la.score(x_train,y_train)
```
Out[46]: 0.37364827503353815

Accuracy(Lasso)

In [47]:
```
la.score(x_test,y_test)
```
Out[47]: 0.36239748001773564

Elastic Net

In [48]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[48]:
```
ElasticNet()
```

In [49]:
```
en.coef_
```

Out[49]:
```
array([ 0.        ,  0.        ,  0.        ,  0.        , -0.06309774,
       -0.04242463,  0.00461886, -0.59054753,  0.        , -0.36246202])
```

In [50]:
```
en.intercept_
```

Out[50]:
```
28079026.97350473
```

In [51]:
```
prediction=en.predict(x_test)
```

In [52]:
```
en.score(x_test,y_test)
```

Out[52]:
```
0.4525528104685921
```

Evaluation Metrics

In [53]:
```
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)))
```

```
3.8630474262235275
27.92966396938047
5.284852312920435
```

Logistic Regression

In [54]:
```
from sklearn.linear_model import LogisticRegression
```

In [58]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2','O_3',
       'PM10','SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [59]:
```
feature_matrix.shape
```

Out[59]:
```
(10916, 10)
```

In [60]:
```
target_vector.shape
```

Out[60]:
```
(10916,)
```

In [61]:
```
from sklearn.preprocessing import StandardScaler
```

In [62]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [63]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[63]:
```
LogisticRegression(max_iter=10000)
```

In [70]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [71]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [72]:
```
logr.classes_
```

Out[72]:
```
array([28079008, 28079024], dtype=int64)
```

In [73]:
```
logr.score(fs,target_vector)
```

Out[73]:
```
0.9293697325027482
```

In [74]:
```
logr.predict_proba(observation)[0][0]
```

Out[74]:
```
1.0
```

In [75]:
```
logr.predict_proba(observation)
```

Out[75]:
```
array([[1.00000000e+00, 3.50349553e-26]])
```

Random Forest

In [76]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [77]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[77]:
```
RandomForestClassifier()
```

In [78]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [79]:
```
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[79]:
```
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 [80]:
```
grid_search.best_score_
```

Out[80]:
```
0.9611307585114501
```

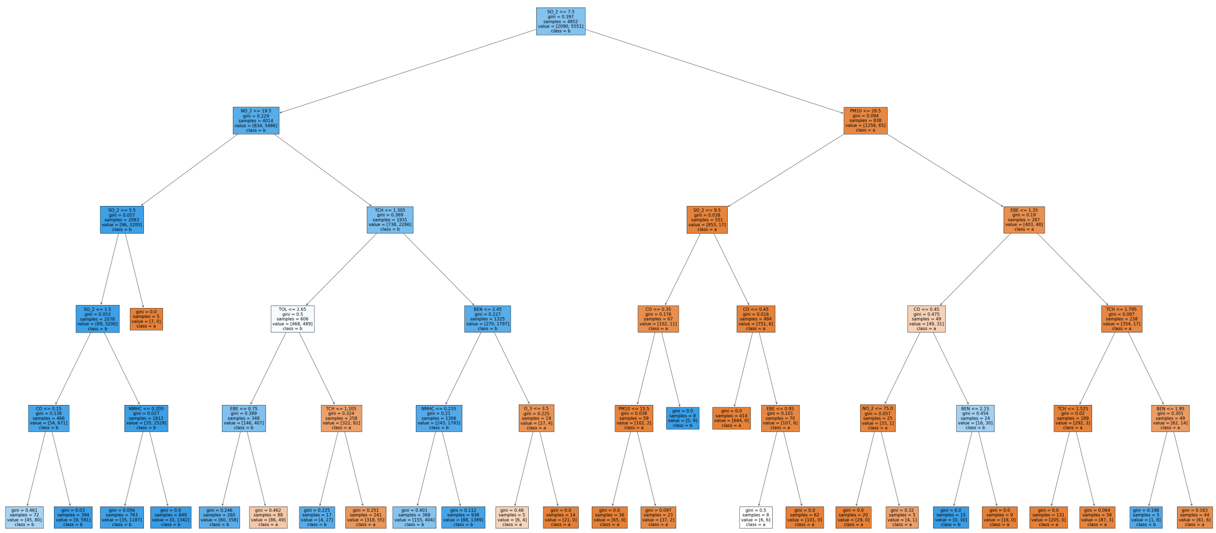
In [81]:
```
rfc_best=grid_search.best_estimator_
```

In [82]:
```
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[82]:
```
[Text(2053.44, 1993.2, 'SO_2 <= 7.5\ngini = 0.397\nsamples = 4852\nvalue = [2090, 55
51]\nclass = b'),
 Text(937.44, 1630.8000000000002, 'NO_2 <= 19.5\ngini = 0.229\nsamples = 4014\nvalue
= [834, 5486]\nclass = b'),
 Text(446.4, 1268.4, 'SO_2 <= 5.5\ngini = 0.057\nsamples = 2083\nvalue = [96, 3200]
\nclass = b'),
 Text(357.12, 906.0, 'SO_2 <= 1.5\ngini = 0.053\nsamples = 2078\nvalue = [89, 3200]
\nclass = b'),
 Text(178.56, 543.5999999999999, 'CO <= 0.15\ngini = 0.138\nsamples = 466\nvalue =
[54, 671]\nclass = b'),
 Text(89.28, 181.19999999999982, 'gini = 0.461\nsamples = 72\nvalue = [45, 80]\nclas
s = b'),
 Text(267.84000000000003, 181.19999999999982, 'gini = 0.03\nsamples = 394\nvalue =
[9, 591]\nclass = b'),
 Text(535.6800000000001, 543.5999999999999, 'NMHC <= 0.205\ngini = 0.027\nsamples =
1612\nvalue = [35, 2529]\nclass = b'),
 Text(446.4, 181.19999999999982, 'gini = 0.056\nsamples = 763\nvalue = [35, 1187]\nc
lass = b'),
 Text(624.96, 181.19999999999982, 'gini = 0.0\nsamples = 849\nvalue = [0, 1342]\ncla
ss = b'),
 Text(535.6800000000001, 906.0, 'gini = 0.0\nsamples = 5\nvalue = [7, 0]\nclass =
a'),
 Text(1428.48, 1268.4, 'TCH <= 1.305\ngini = 0.369\nsamples = 1931\nvalue = [738, 22
86]\nclass = b'),
 Text(1071.3600000000001, 906.0, 'TOL <= 2.65\ngini = 0.5\nsamples = 606\nvalue = [4
68, 489]\nclass = b'),
 Text(892.8, 543.5999999999999, 'EBE <= 0.75\ngini = 0.389\nsamples = 348\nvalue =
[146, 407]\nclass = b'),
 Text(803.52, 181.19999999999982, 'gini = 0.246\nsamples = 260\nvalue = [60, 358]\nc
lass = b'),
 Text(982.08, 181.19999999999982, 'gini = 0.462\nsamples = 88\nvalue = [86, 49]\ncla
ss = a'),
 Text(1249.92, 543.5999999999999, 'TCH <= 1.205\ngini = 0.324\nsamples = 258\nvalue
= [322, 82]\nclass = a'),
 Text(1160.64, 181.19999999999982, 'gini = 0.225\nsamples = 17\nvalue = [4, 27]\ncla
ss = b'),
 Text(1339.2, 181.19999999999982, 'gini = 0.251\nsamples = 241\nvalue = [318, 55]\nc
lass = a'),
 Text(1785.6, 906.0, 'BEN <= 2.45\ngini = 0.227\nsamples = 1325\nvalue = [270, 1797]
\nclass = b'),
 Text(1607.04, 543.5999999999999, 'NMHC <= 0.235\ngini = 0.21\nsamples = 1306\nvalue
```

```
= [243, 1793]\nclass = b'),
 Text(1517.76, 181.19999999999982, 'gini = 0.401\nsamples = 368\nvalue = [155, 404]
\nclass = b'),
 Text(1696.32, 181.19999999999982, 'gini = 0.112\nsamples = 938\nvalue = [88, 1389]
\nclass = b'),
 Text(1964.16, 543.5999999999999, 'O_3 <= 3.5\ngini = 0.225\nsamples = 19\nvalue =
[27, 4]\nclass = a'),
 Text(1874.88, 181.19999999999982, 'gini = 0.48\nsamples = 5\nvalue = [6, 4]\nclass
= a'),
 Text(2053.44, 181.19999999999982, 'gini = 0.0\nsamples = 14\nvalue = [21, 0]\nclass
= a'),
 Text(3169.44, 1630.8000000000002, 'PM10 <= 28.5\ngini = 0.094\nsamples = 838\nvalue
= [1256, 65]\nclass = a'),
 Text(2589.12, 1268.4, 'SO_2 <= 9.5\ngini = 0.038\nsamples = 551\nvalue = [853, 17]
\nclass = a'),
 Text(2410.56, 906.0, 'CO <= 0.35\ngini = 0.176\nsamples = 67\nvalue = [102, 11]\ncl
ass = a'),
 Text(2321.28, 543.5999999999999, 'PM10 <= 15.5\ngini = 0.038\nsamples = 59\nvalue =
[102, 2]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.0\nsamples = 36\nvalue = [65, 0]\nclass
= a'),
 Text(2410.56, 181.19999999999982, 'gini = 0.097\nsamples = 23\nvalue = [37, 2]\ncla
ss = a'),
 Text(2499.84, 543.5999999999999, 'gini = 0.0\nsamples = 8\nvalue = [0, 9]\nclass =
b'),
 Text(2767.68, 906.0, 'CO <= 0.45\ngini = 0.016\nsamples = 484\nvalue = [751, 6]\ncl
ass = a'),
 Text(2678.4, 543.5999999999999, 'gini = 0.0\nsamples = 414\nvalue = [644, 0]\nclass
= a'),
 Text(2856.96, 543.5999999999999, 'EBE <= 0.95\ngini = 0.101\nsamples = 70\nvalue =
[107, 6]\nclass = a'),
 Text(2767.68, 181.19999999999982, 'gini = 0.5\nsamples = 8\nvalue = [6, 6]\nclass =
a'),
 Text(2946.2400000000002, 181.19999999999982, 'gini = 0.0\nsamples = 62\nvalue = [10
1, 0]\nclass = a'),
 Text(3749.76, 1268.4, 'EBE <= 1.35\ngini = 0.19\nsamples = 287\nvalue = [403, 48]\n
class = a'),
 Text(3392.64, 906.0, 'CO <= 0.45\ngini = 0.475\nsamples = 49\nvalue = [49, 31]\ncla
ss = a'),
 Text(3214.08, 543.5999999999999, 'NO_2 <= 75.0\ngini = 0.057\nsamples = 25\nvalue =
[33, 1]\nclass = a'),
 Text(3124.8, 181.19999999999982, 'gini = 0.0\nsamples = 20\nvalue = [29, 0]\nclass
= a'),
 Text(3303.36, 181.19999999999982, 'gini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass
= a'),
 Text(3571.2, 543.5999999999999, 'BEN <= 2.15\ngini = 0.454\nsamples = 24\nvalue =
[16, 30]\nclass = b'),
 Text(3481.92, 181.19999999999982, 'gini = 0.0\nsamples = 15\nvalue = [0, 30]\nclass
= b'),
 Text(3660.48, 181.19999999999982, 'gini = 0.0\nsamples = 9\nvalue = [16, 0]\nclass
= a'),
 Text(4106.88, 906.0, 'TCH <= 1.795\ngini = 0.087\nsamples = 238\nvalue = [354, 17]
\nclass = a'),
 Text(3928.32, 543.5999999999999, 'TCH <= 1.525\ngini = 0.02\nsamples = 189\nvalue =
[292, 3]\nclass = a'),
 Text(3839.04, 181.19999999999982, 'gini = 0.0\nsamples = 131\nvalue = [205, 0]\ncla
ss = a'),
 Text(4017.6, 181.19999999999982, 'gini = 0.064\nsamples = 58\nvalue = [87, 3]\nclas
s = a'),
 Text(4285.4400000000005, 543.5999999999999, 'BEN <= 1.95\ngini = 0.301\nsamples = 4
9\nvalue = [62, 14]\nclass = a'),
 Text(4196.16, 181.19999999999982, 'gini = 0.198\nsamples = 5\nvalue = [1, 8]\nclass
= b'),
 Text(4374.72, 181.19999999999982, 'gini = 0.163\nsamples = 44\nvalue = [61, 6]\ncla
ss = a')]
```



Conclusion

Scores

Linear Regression

In [83]:
```
lr.score(x_test,y_test)
```

Out[83]: 0.6086004152156592

In [84]:
```
lr.score(x_train,y_train)
```

Out[84]: 0.6273759315249119

Lasso

In [85]:
```
la.score(x_test,y_test)
```

Out[85]: 0.36239748001773564

Ridge

In [86]:
```
rr.score(x_test,y_test)
```

Out[86]: 0.6048567528892628

In [87]:
```
rr.score(x_train,y_train)
```

Out[87]: 0.6249236689518839

Elastic Net

In [88]:
```
en.score(x_test,y_test)
```

Out[88]: 0.4525528104685921

Logistic Regression

In [89]:
```
logr.score(fs,target_vector)
```

Out[89]: 0.9293697325027482

Random Forest

In [90]:
```
grid_search.best_score_
```

Out[90]: 0.9611307585114501

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

In [ ]:
```

```