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("2011.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	s
0	2011-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN	28C
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7	28C
2	2011-11-01 01:00:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2	28C
3	2011-11-01 01:00:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN	28C
4	2011-11-01 01:00:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN	28C
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
209923	2011-09-01 00:00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN	28C
209924	2011-09-01 00:00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN	28C
209925	2011-09-01 00:00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN	28C
209926	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN	28C
209927	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN	28C

209928 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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[6]:

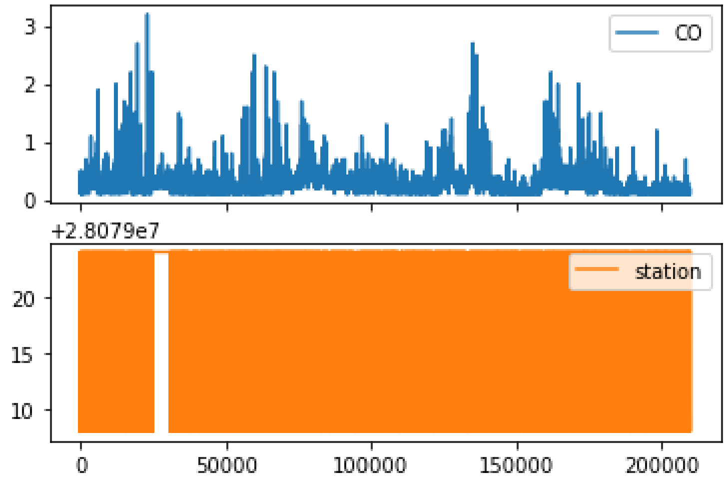
	CO	station
1	0.4	28079008
6	0.3	28079024
25	0.3	28079008
30	0.4	28079024
49	0.2	28079008
...	...	...
209862	0.1	28079024
209881	0.1	28079008
209886	0.1	28079024
209905	0.1	28079008
209910	0.1	28079024

16460 rows × 2 columns

Line chart

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

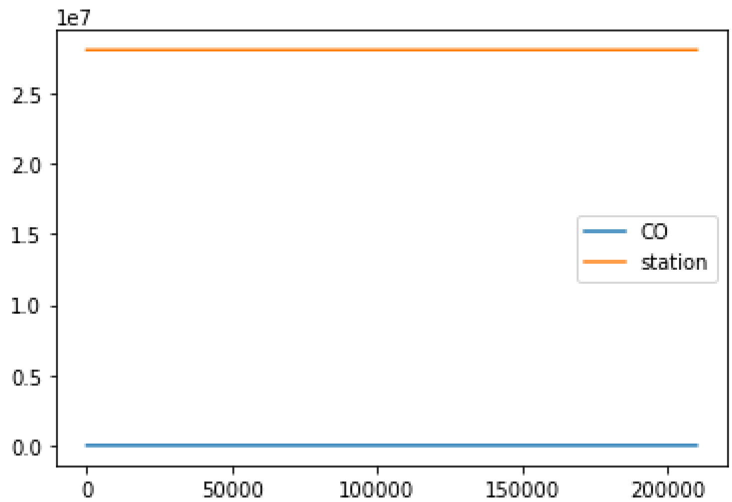
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

Out[8]: <AxesSubplot:>

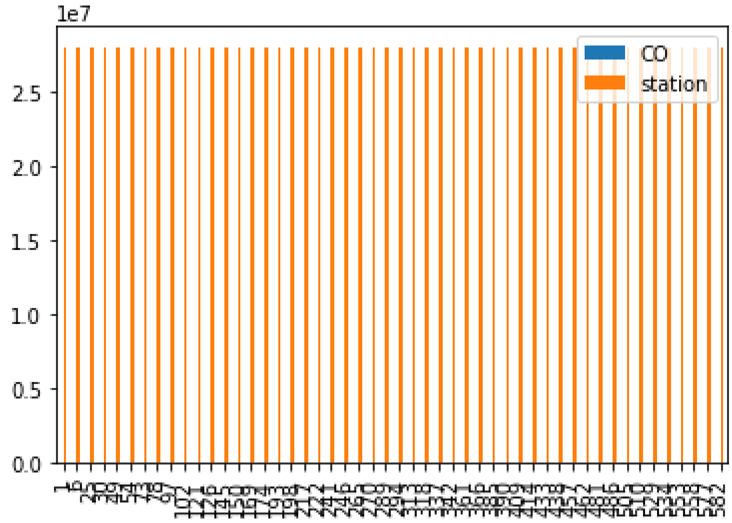


Bar chart

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

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

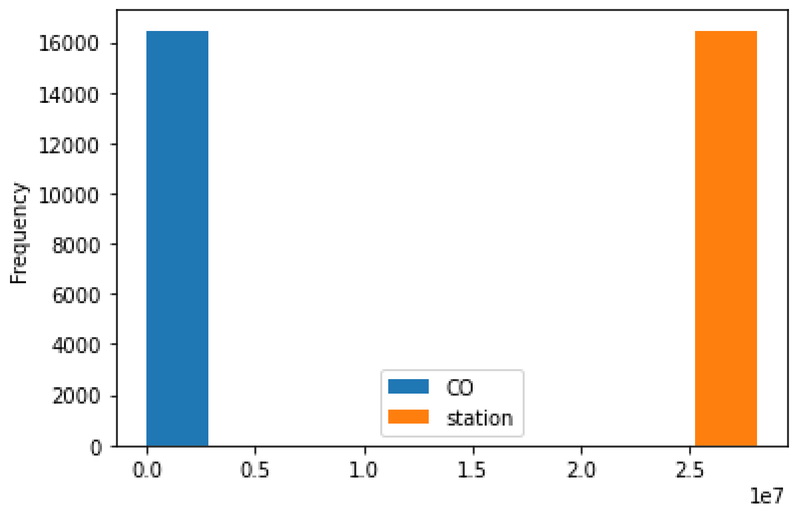
Out[10]: <AxesSubplot:>



Histogram

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

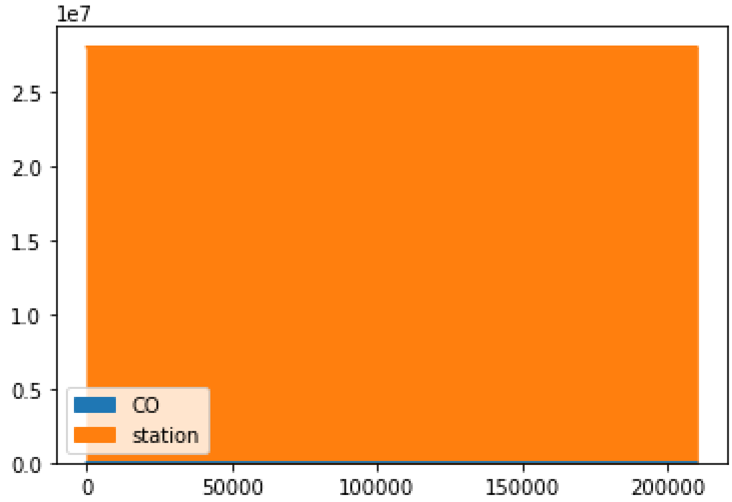
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

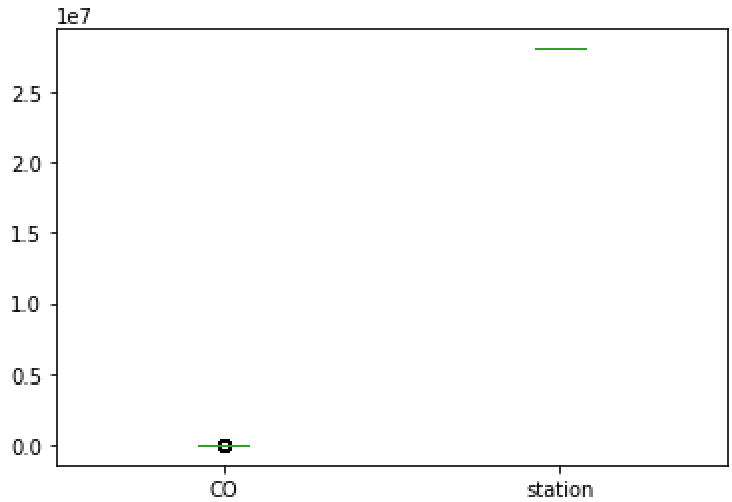
Out[12]: <AxesSubplot:>



Box chart

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

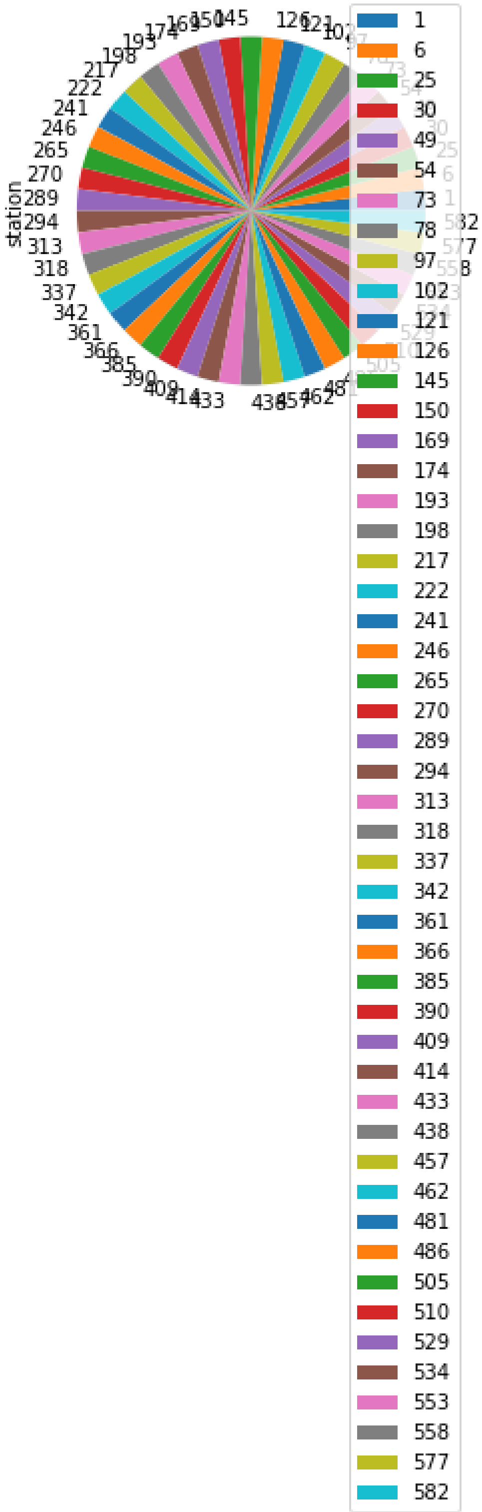
Out[13]: <AxesSubplot:>



Pie chart

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

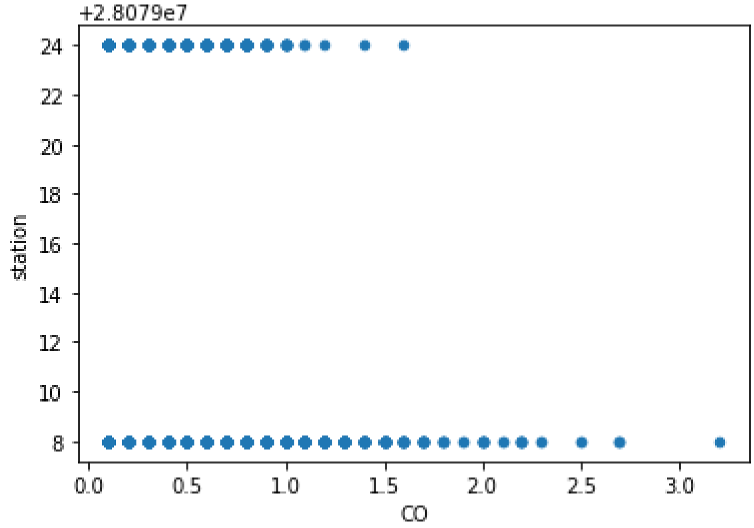
Out[14]: <AxesSubplot:ylabel='station'>



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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [17]:
```
df.describe()
```

Out[17]:

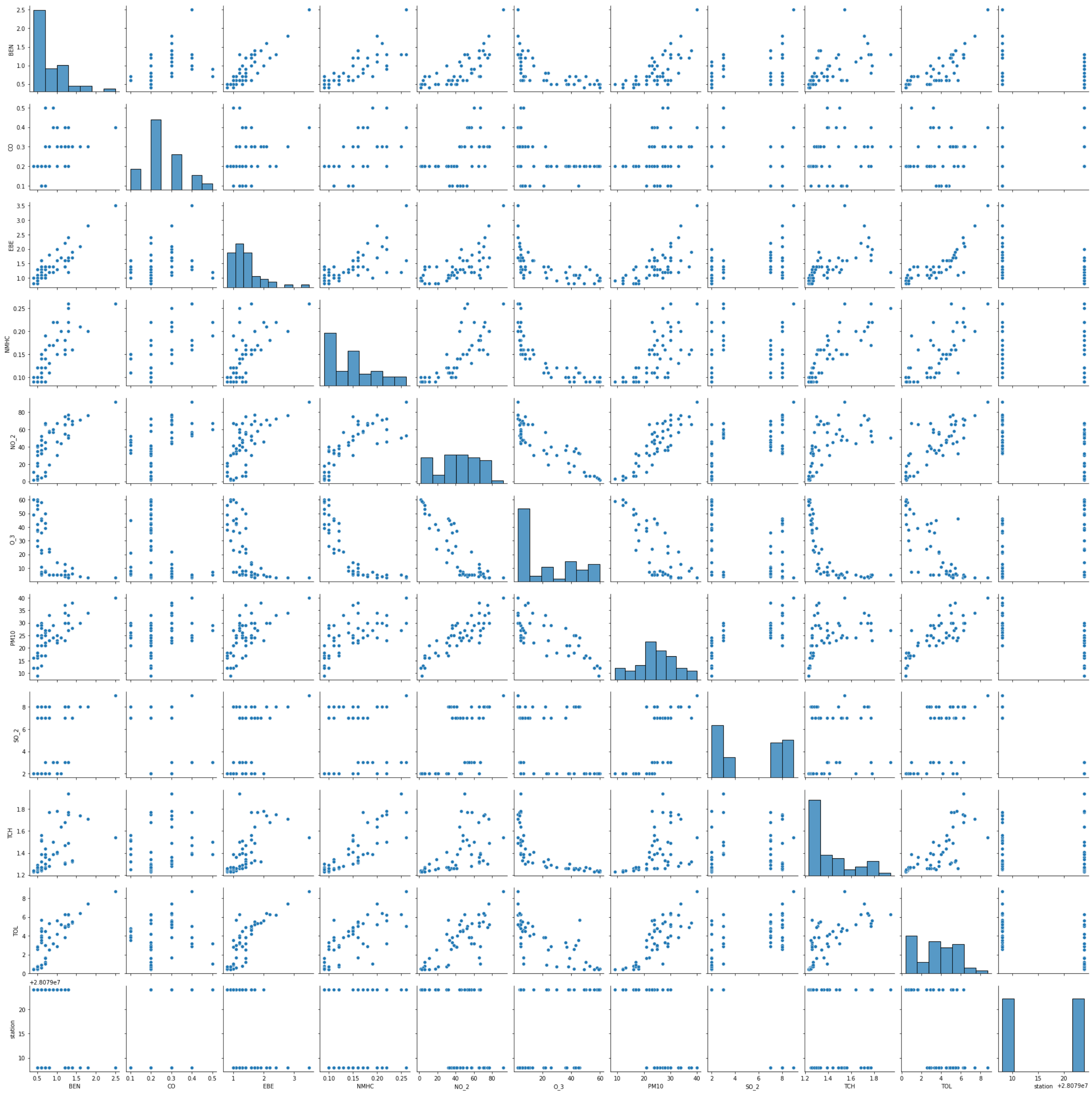
	BEN	CO	EBE	NMHC	NO	NO_2	
count	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.0C
mean	0.900680	0.277758	1.471871	0.167043	23.671810	44.583961	41.58
std	0.768892	0.206143	1.051004	0.075068	44.362859	31.569185	28.11
min	0.100000	0.100000	0.200000	0.010000	1.000000	1.000000	1.0C
25%	0.500000	0.200000	0.800000	0.120000	2.000000	19.000000	17.0C
50%	0.700000	0.200000	1.200000	0.160000	7.000000	40.000000	39.0C
75%	1.100000	0.300000	1.700000	0.200000	25.000000	63.000000	61.0C
max	9.500000	3.200000	12.800000	0.840000	615.000000	289.000000	154.0C

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])
```

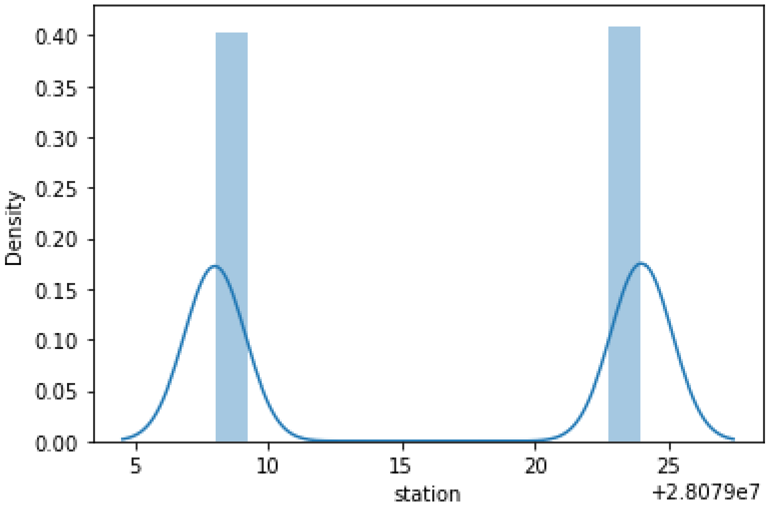
Out[20]: <seaborn.axisgrid.PairGrid at 0x2b838ba11c0>



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

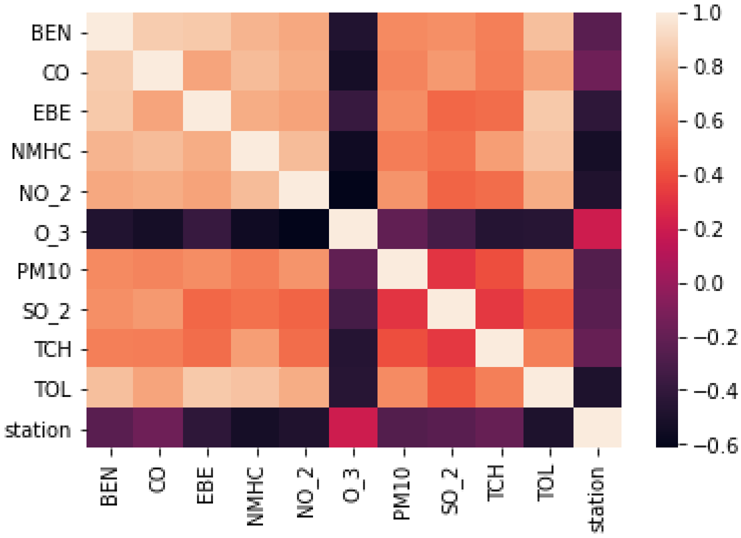
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function 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 [24]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
y=df['station']
```

In [25]:
```
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 [26]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[26]: LinearRegression()

In [27]:
```
lr.intercept_
```

Out[27]: 3.725290298461914e-09

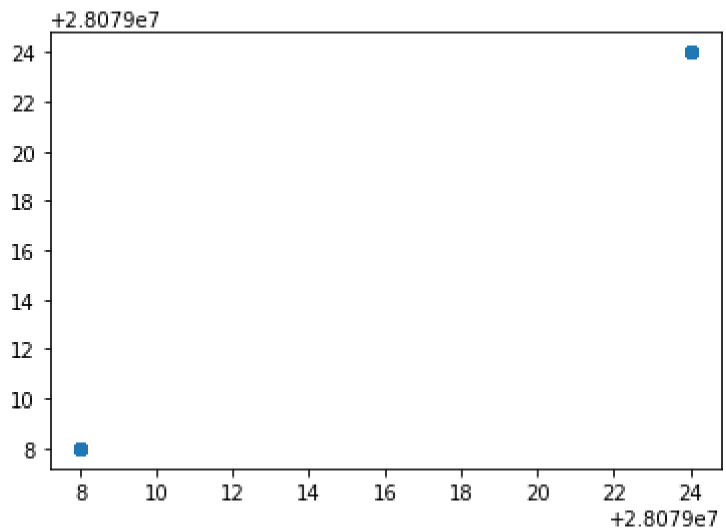
In [28]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[28]:

	Co-efficient
BEN	5.890901e-15
CO	-3.520731e-14
EBE	-4.213255e-15
NMHC	8.625282e-14
NO_2	-3.680550e-16
O_3	-2.440437e-16
PM10	4.011955e-16
SO_2	-2.949478e-16
TCH	5.710240e-15
TOL	-2.745079e-16
station	1.000000e+00

In [29]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[29]: <matplotlib.collections.PathCollection at 0x2b8415ebac0>



ACCURACY

In [30]:
```
lr.score(x_test,y_test)
```

Out[30]: 1.0

In [31]:
```
lr.score(x_train,y_train)
```

Out[31]: 1.0

Ridge and Lasso

In [32]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [33]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[33]: Ridge(alpha=10)

Accuracy(Ridge)

In [34]:
```
rr.score(x_test,y_test)
```

Out[34]: 0.999999999611746

In [35]:
```
rr.score(x_train,y_train)
```

Out[35]: 0.9999999996219582

In [36]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[36]: Lasso(alpha=10)

In [37]:
```
la.score(x_train,y_train)
```

Out[37]: 0.97386196894652

Accuracy(Lasso)

In [38]:
```
la.score(x_test,y_test)
```

Out[38]: 0.9735494647088159

Elastic Net

In [39]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[39]:
```
ElasticNet()
```

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

Out[40]:
```
array([ 0.        ,  0.        , -0.        , -0.        , -0.0018391 ,
       -0.        , -0.        , -0.        ,  0.        , -0.        ,
        0.98100195])
```

In [41]:
```
en.intercept_
```

Out[41]:
```
533446.6172212809
```

In [42]:
```
prediction=en.predict(x_test)
```

In [43]:
```
en.score(x_test,y_test)
```

Out[43]:
```
0.9997150865067317
```

Evaluation Metrics

In [44]:
```
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)))
```

```
0.12655833940809677
0.018234272130008482
0.13503433685551422
```

Logistic Regression

In [45]:
```
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]:
```
(16460, 10)
```

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

Out[60]:
```
(16460,)
```

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 [64]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [65]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [66]:
```
logr.classes_
```

Out[66]:
```
array([28079008, 28079024], dtype=int64)
```

In [67]:
```
logr.score(fs,target_vector)
```

Out[67]:
```
0.9237545565006076
```

In [68]:
```
logr.predict_proba(observation)[0][0]
```

Out[68]:
```
0.9999999999999966
```

In [69]:
```
logr.predict_proba(observation)
```

Out[69]:
```
array([[1.00000000e+00, 3.47334507e-15]])
```

Random Forest

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [71]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[71]:
```
RandomForestClassifier()
```

In [72]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [73]:
```
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[73]:
```
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 [74]:
```
grid_search.best_score_
```

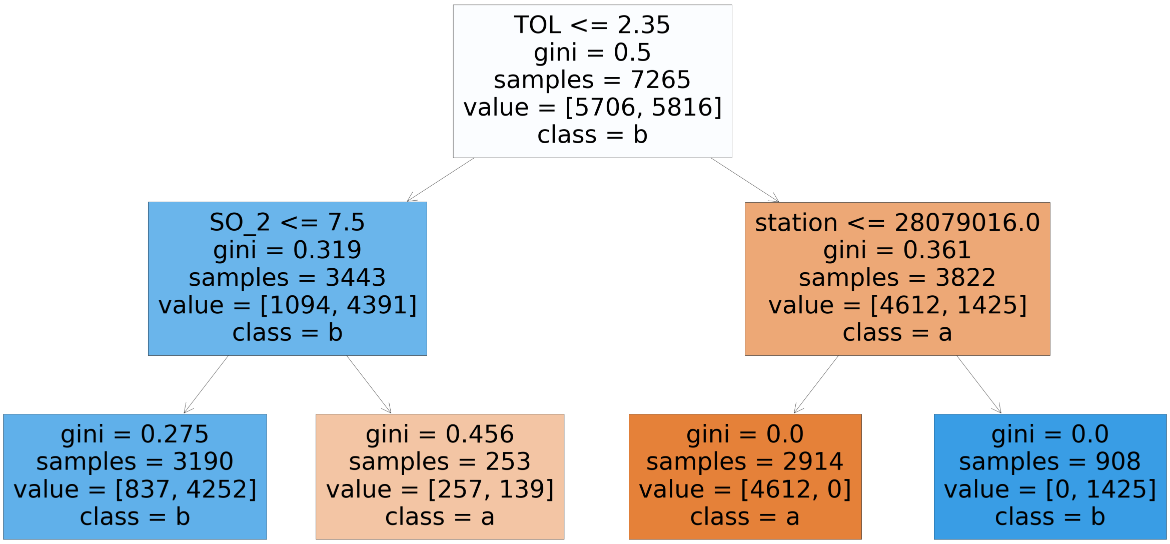
Out[74]:
```
1.0
```

In [75]:
```
rfc_best=grid_search.best_estimator_
```

In [76]:
```
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[76]:
```
[Text(2232.0, 1812.0, 'TOL <= 2.35\ngini = 0.5\nsamples = 7265\nvalue = [5706, 5816]
\nclass = b'),
 Text(1116.0, 1087.2, 'SO_2 <= 7.5\ngini = 0.319\nsamples = 3443\nvalue = [1094, 439
1]\nclass = b'),
 Text(558.0, 362.39999999999986, 'gini = 0.275\nsamples = 3190\nvalue = [837, 4252]
\nclass = b'),
 Text(1674.0, 362.39999999999986, 'gini = 0.456\nsamples = 253\nvalue = [257, 139]\n
class = a'),
 Text(3348.0, 1087.2, 'station <= 28079016.0\ngini = 0.361\nsamples = 3822\nvalue =
[4612, 1425]\nclass = a'),
 Text(2790.0, 362.39999999999986, 'gini = 0.0\nsamples = 2914\nvalue = [4612, 0]\ncl
ass = a'),
 Text(3906.0, 362.39999999999986, 'gini = 0.0\nsamples = 908\nvalue = [0, 1425]\ncla
ss = b')]
```



Conclusion

Scores

Linear Regression

In [77]:
```
lr.score(x_test,y_test)
```
Out[77]: 1.0

In [78]:
```
lr.score(x_train,y_train)
```
Out[78]: 1.0

Lasso

In [79]:
```
la.score(x_test,y_test)
```
Out[79]: 0.9735494647088159

Ridge

In [80]:
```
rr.score(x_test,y_test)
```
Out[80]: 0.999999999611746

In [81]:
```
rr.score(x_train,y_train)
```
Out[81]: 0.9999999996219582

Elastic Net

In [82]:
```
en.score(x_test,y_test)
```
Out[82]: 0.9997150865067317

Logistic Regression

In [83]:
```
logr.score(fs,target_vector)
```
Out[83]: 0.9237545565006076

Random Forest

In [84]:
```
grid_search.best_score_
```
Out[84]: 1.0

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

```
In [ ]:
```