Importing Libraries

In [1]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:

```python
df=pd.read_csv("2018.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH
0	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.4
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69091	2018-02-01 00:00:00	NaN	NaN	0.5	NaN	NaN	66.0	91.0	192.0	1.0	35.0	22.0	NaN	NaN
69092	2018-02-01 00:00:00	NaN	NaN	0.7	NaN	NaN	87.0	107.0	241.0	NaN	29.0	NaN	15.0	NaN
69093	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	28.0	48.0	91.0	2.0	NaN	NaN	NaN	NaN
69094	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	141.0	103.0	320.0	2.0	NaN	NaN	NaN	NaN
69095	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	69.0	96.0	202.0	3.0	26.0	NaN	NaN	NaN

69096 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: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

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

Out[6]:

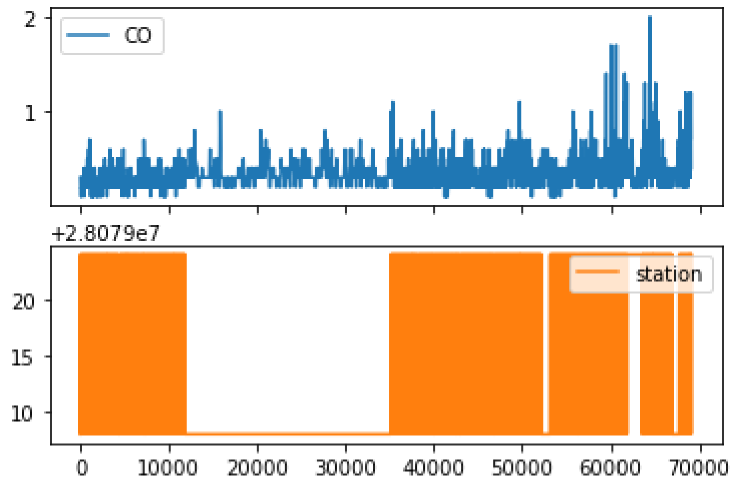
	CO	station
1	0.3	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
69030	0.7	28079024
69049	1.2	28079008
69054	0.6	28079024
69073	1.0	28079008
69078	0.4	28079024

4562 rows × 2 columns

Line chart

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

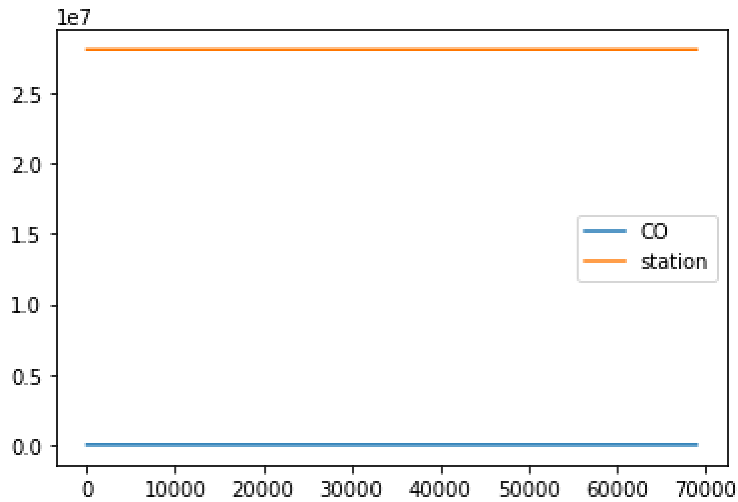
Out[7]: `array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)`



Line chart

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

Out[8]: `<AxesSubplot:>`

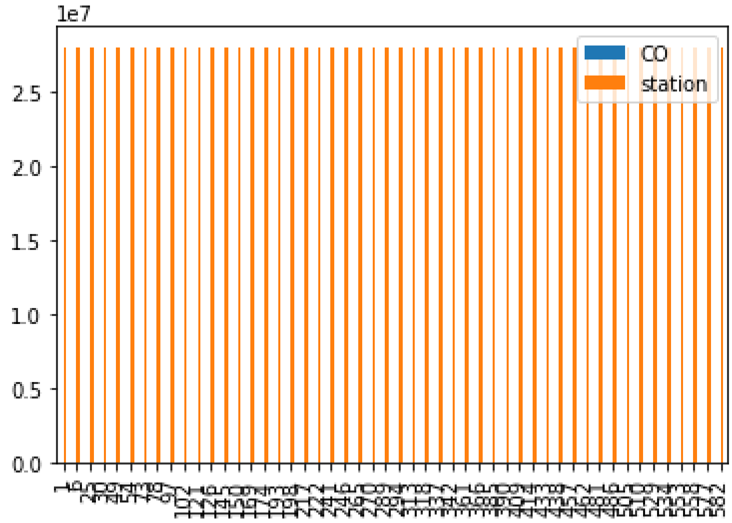


Bar chart

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

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

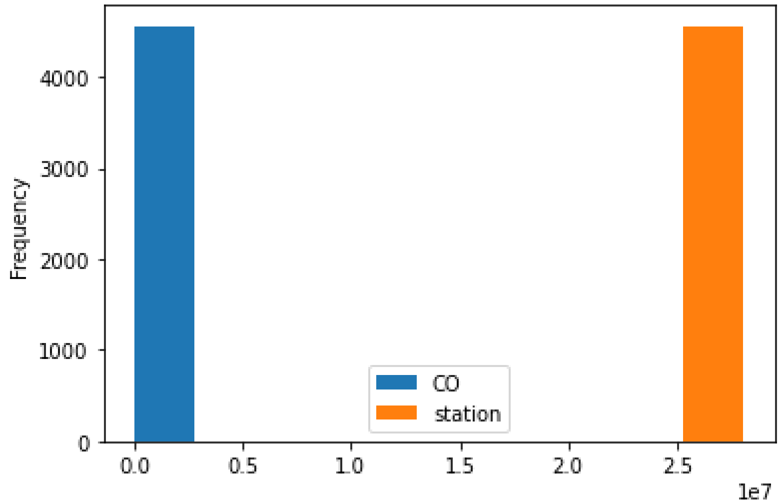
Out[10]: `<AxesSubplot:>`



Histogram

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

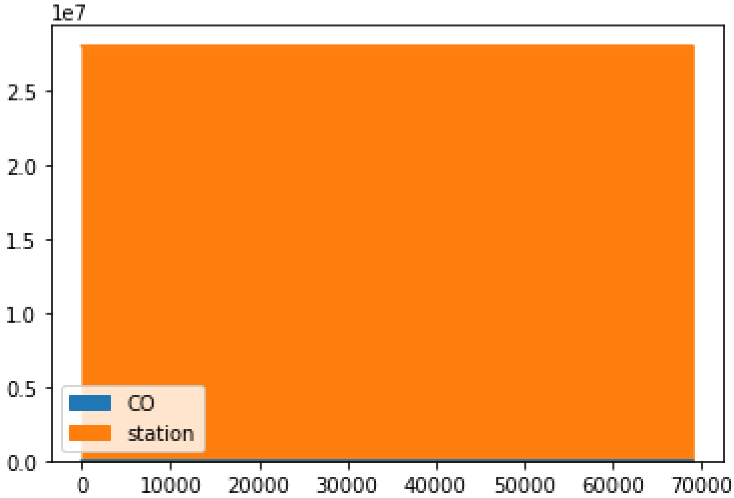
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

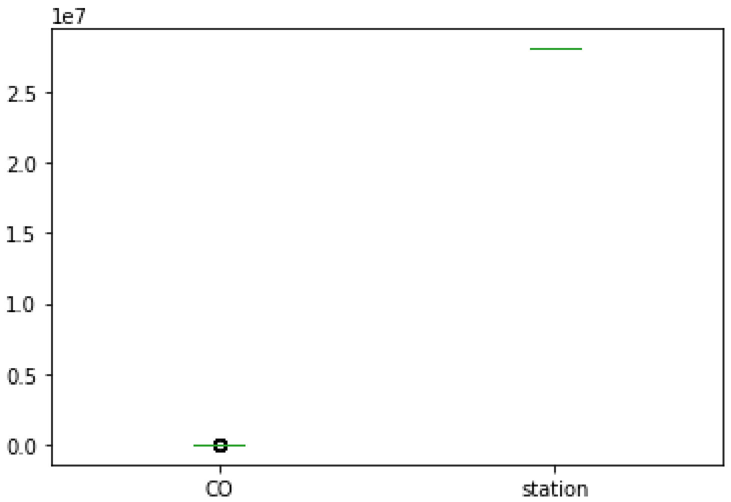
Out[12]: <AxesSubplot:>



Box chart

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

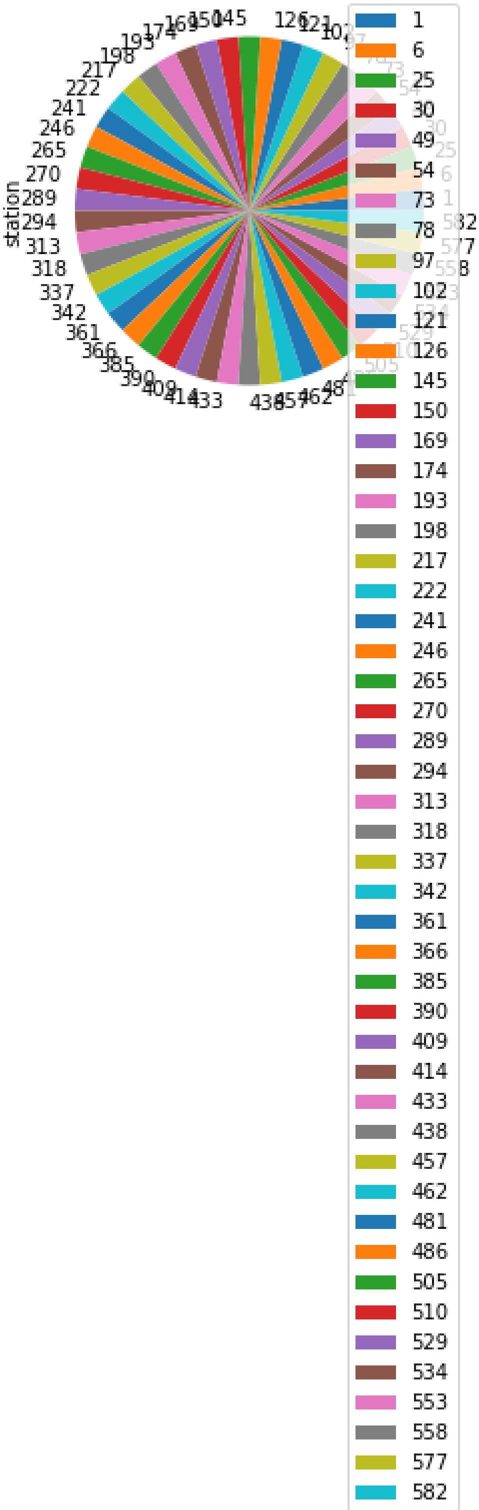
Out[13]: <AxesSubplot:>



Pie chart

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

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

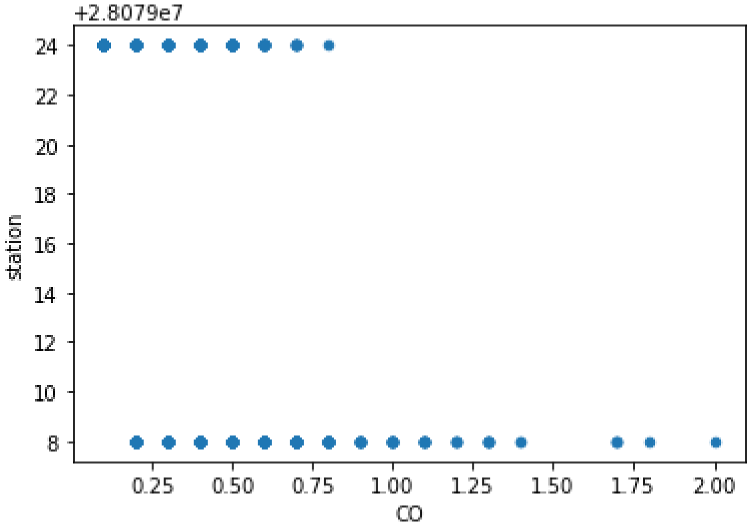


Scatter chart

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: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4562 non-null   object 
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ 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	4562.00000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	45
mean	0.69349	1.329163	0.330579	0.286782	0.056773	21.742218	44.152126	
std	0.46832	0.214399	0.161489	0.354442	0.037711	35.539531	30.234015	
min	0.10000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000	
25%	0.40000	1.120000	0.200000	0.100000	0.030000	1.000000	20.000000	

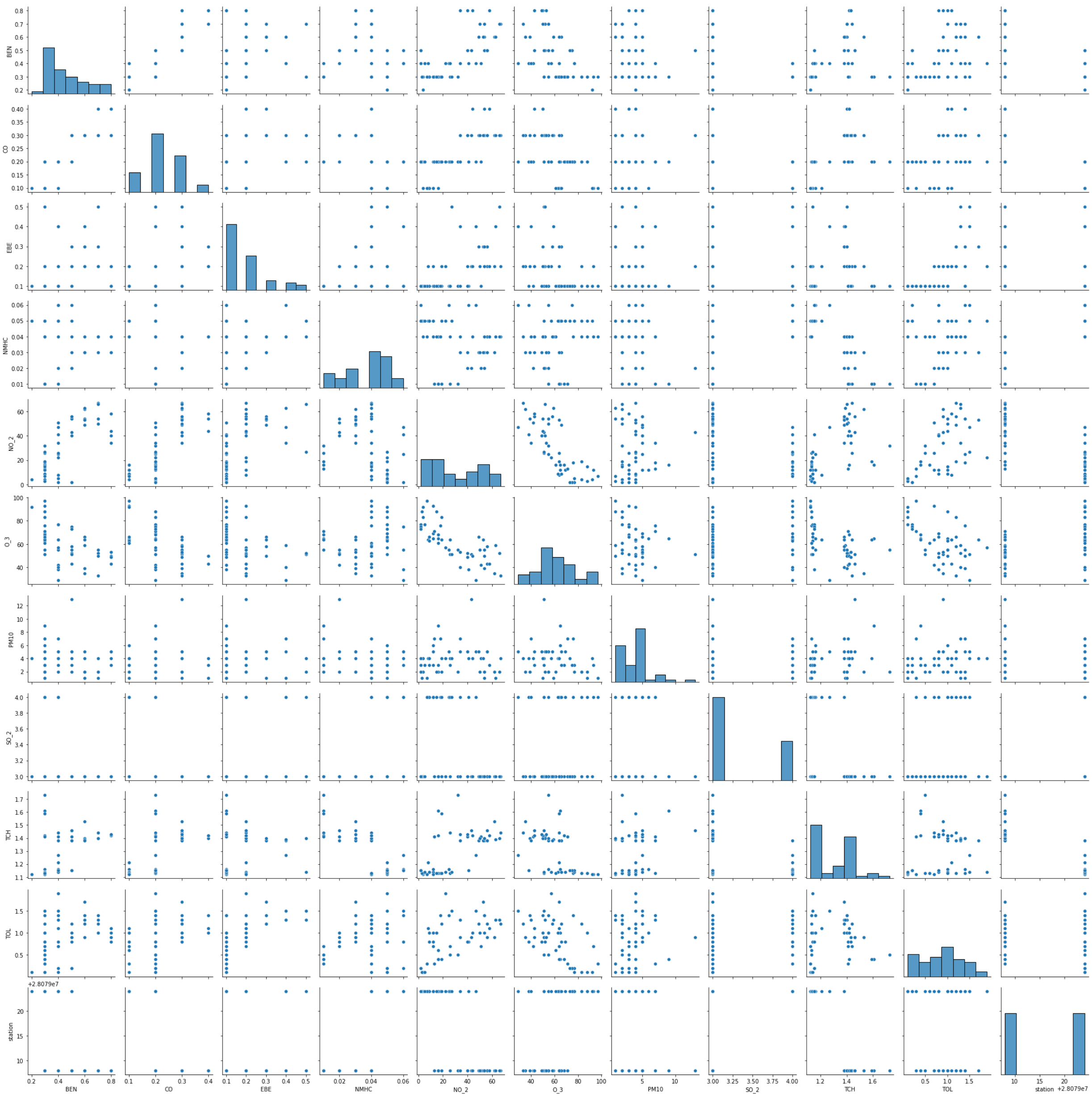
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
50%	0.60000	1.390000	0.300000	0.200000	0.050000	9.000000	41.000000	
75%	0.90000	1.420000	0.400000	0.300000	0.070000	27.000000	64.000000	1
max	6.60000	3.920000	2.000000	7.400000	0.490000	431.000000	184.000000	8

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])
```

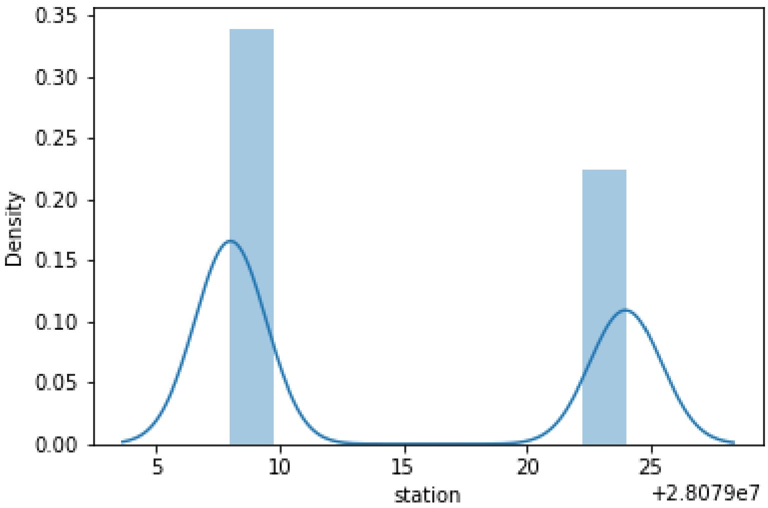
Out[20]: <seaborn.axisgrid.PairGrid at 0x1a52cb4e160>



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)
```

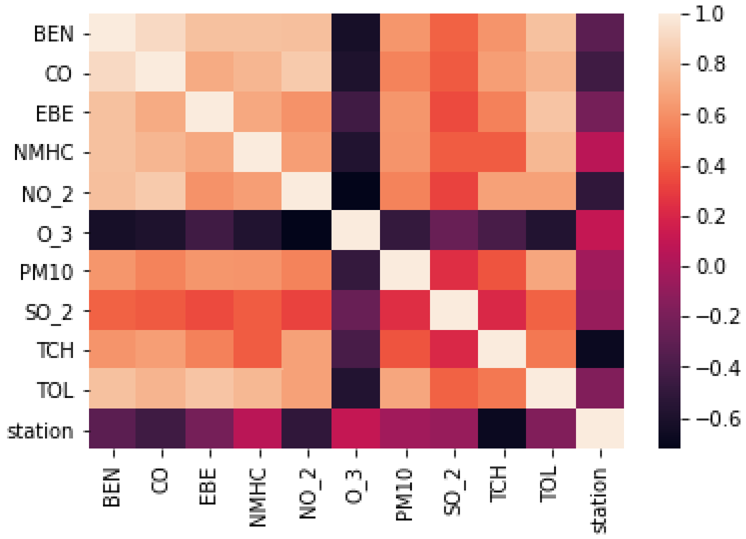
Out[21]: <AxesSubplot:xlabel='station', ylabel='Density'>



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]: 28079042.5912627

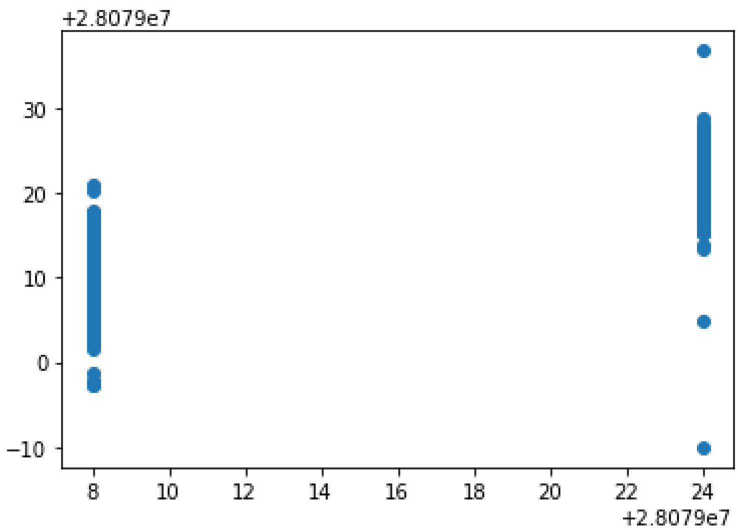
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	-0.499146
CO	-18.095949
EBE	0.768948
NMHC	156.176135
NO_2	-0.157094
O_3	-0.086085
PM10	0.100411
SO_2	-0.036225
TCH	-15.074253
TOL	-0.264953

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

Out[28]: <matplotlib.collections.PathCollection at 0x1a533f45460>



ACCURACY

In [29]:
```
lr.score(x_test,y_test)
```
Out[29]: 0.808816625773953

In [30]:
```
lr.score(x_train,y_train)
```
Out[30]: 0.8029483548008225

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]: 0.6740227370581086

In [34]:
```
rr.score(x_train,y_train)
```
Out[34]: 0.6802682051752258

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]: 0.41555219010251554

Accuracy(Lasso)

In [37]:
```
la.score(x_test,y_test)
```
Out[37]: 0.38477209623527864

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.        , -0.26753332,
       -0.14424067,  0.22426734,  0.08680002, -0.03829776,  0.09185533])
```
In [40]:
```
en.intercept_
```
Out[40]:
```
28079029.276944675
```
In [41]:
```
prediction=en.predict(x_test)
```
In [42]:
```
en.score(x_test,y_test)
```
Out[42]:
```
0.44958825409900893
```
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)))
```
```
4.983571887710196
33.69966303808441
5.805141086837116
```
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]:
```
(4562, 10)
```
In [47]:
```
target_vector.shape
```
Out[47]:
```
(4562,)
```

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]:
```
0.9888206926786497
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 1.42669593e-19]])
```

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]:
```
0.9921700776675565
```

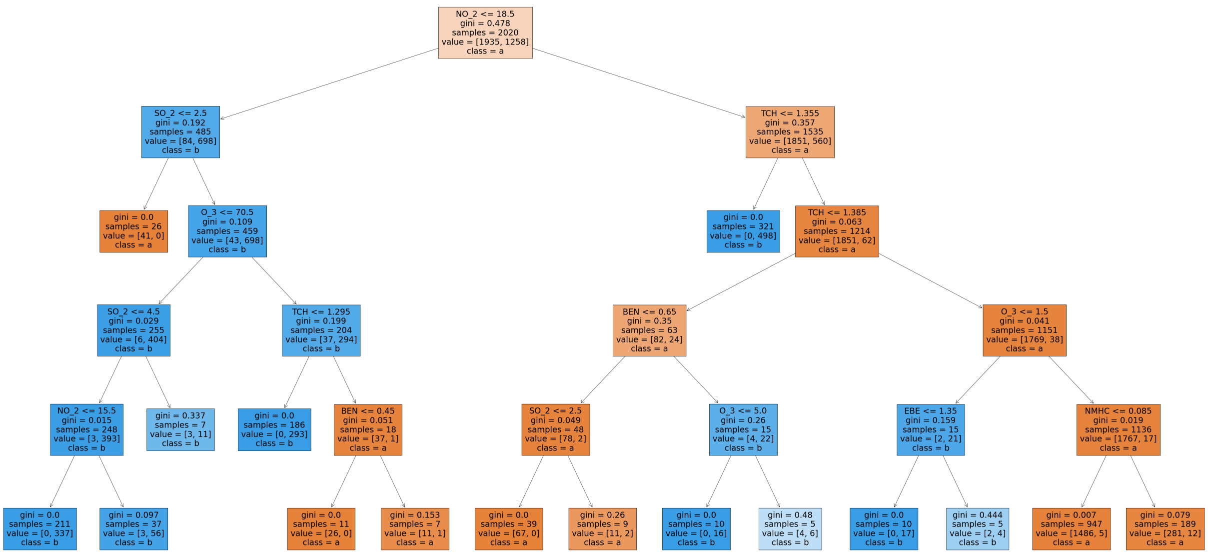
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1802.7692307692307, 1993.2, 'NO_2 <= 18.5\ngini = 0.478\nsamples = 2020\nvalue
= [1935, 1258]\nclass = a'),
 Text(686.7692307692307, 1630.8000000000002, 'SO_2 <= 2.5\ngini = 0.192\nsamples = 4
85\nvalue = [84, 698]\nclass = b'),
 Text(515.0769230769231, 1268.4, 'gini = 0.0\nsamples = 26\nvalue = [41, 0]\nclass =
a'),
 Text(858.4615384615383, 1268.4, 'O_3 <= 70.5\ngini = 0.109\nsamples = 459\nvalue =
[43, 698]\nclass = b'),
 Text(515.0769230769231, 906.0, 'SO_2 <= 4.5\ngini = 0.029\nsamples = 255\nvalue =
[6, 404]\nclass = b'),
 Text(343.38461538461536, 543.5999999999999, 'NO_2 <= 15.5\ngini = 0.015\nsamples =
248\nvalue = [3, 393]\nclass = b'),
 Text(171.69230769230768, 181.19999999999982, 'gini = 0.0\nsamples = 211\nvalue =
[0, 337]\nclass = b'),
 Text(515.0769230769231, 181.19999999999982, 'gini = 0.097\nsamples = 37\nvalue =
[3, 56]\nclass = b'),
 Text(686.7692307692307, 543.5999999999999, 'gini = 0.337\nsamples = 7\nvalue = [3,
11]\nclass = b'),
 Text(1201.8461538461538, 906.0, 'TCH <= 1.295\ngini = 0.199\nsamples = 204\nvalue =
[37, 294]\nclass = b'),
 Text(1030.1538461538462, 543.5999999999999, 'gini = 0.0\nsamples = 186\nvalue = [0,
293]\nclass = b'),
 Text(1373.5384615384614, 543.5999999999999, 'BEN <= 0.45\ngini = 0.051\nsamples = 1
8\nvalue = [37, 1]\nclass = a'),
 Text(1201.8461538461538, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [2
6, 0]\nclass = a'),
 Text(1545.230769230769, 181.19999999999982, 'gini = 0.153\nsamples = 7\nvalue = [1
1, 1]\nclass = a'),
 Text(2918.7692307692305, 1630.8000000000002, 'TCH <= 1.355\ngini = 0.357\nsamples =
1535\nvalue = [1851, 560]\nclass = a'),
 Text(2747.076923076923, 1268.4, 'gini = 0.0\nsamples = 321\nvalue = [0, 498]\nclass
= b'),
 Text(3090.461538461538, 1268.4, 'TCH <= 1.385\ngini = 0.063\nsamples = 1214\nvalue
= [1851, 62]\nclass = a'),
 Text(2403.6923076923076, 906.0, 'BEN <= 0.65\ngini = 0.35\nsamples = 63\nvalue = [8
2, 24]\nclass = a'),
 Text(2060.3076923076924, 543.5999999999999, 'SO_2 <= 2.5\ngini = 0.049\nsamples = 4
8\nvalue = [78, 2]\nclass = a'),
 Text(1888.6153846153845, 181.19999999999982, 'gini = 0.0\nsamples = 39\nvalue = [6
7, 0]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.26\nsamples = 9\nvalue = [11, 2]\nclass
```

```
= a'),
 Text(2747.076923076923, 543.5999999999999, 'O_3 <= 5.0\ngini = 0.26\nsamples = 15\n
value = [4, 22]\nclass = b'),
 Text(2575.3846153846152, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalue = [0,
16]\nclass = b'),
 Text(2918.7692307692305, 181.19999999999982, 'gini = 0.48\nsamples = 5\nvalue = [4,
6]\nclass = b'),
 Text(3777.230769230769, 906.0, 'O_3 <= 1.5\ngini = 0.041\nsamples = 1151\nvalue =
[1769, 38]\nclass = a'),
 Text(3433.8461538461534, 543.5999999999999, 'EBE <= 1.35\ngini = 0.159\nsamples = 1
5\nvalue = [2, 21]\nclass = b'),
 Text(3262.1538461538457, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalue = [0,
17]\nclass = b'),
 Text(3605.5384615384614, 181.19999999999982, 'gini = 0.444\nsamples = 5\nvalue =
[2, 4]\nclass = b'),
 Text(4120.615384615385, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.019\nsamples =
1136\nvalue = [1767, 17]\nclass = a'),
 Text(3948.9230769230767, 181.19999999999982, 'gini = 0.007\nsamples = 947\nvalue =
[1486, 5]\nclass = a'),
 Text(4292.307692307692, 181.19999999999982, 'gini = 0.079\nsamples = 189\nvalue =
[281, 12]\nclass = a')]
```



Conclusion

Scores

Linear Regression

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

Out[64]: 0.808816625773953

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

Out[65]: 0.8029483548008225

Lasso

In [66]:
```
la.score(x_test,y_test)
```
Out[66]: 0.38477209623527864

Ridge

In [67]:
```
rr.score(x_test,y_test)
```
Out[67]: 0.6740227370581086

In [68]:
```
rr.score(x_train,y_train)
```
Out[68]: 0.6802682051752258

Elastic Net

In [69]:
```
en.score(x_test,y_test)
```
Out[69]: 0.44958825409900893

Logistic Regression

In [70]:
```
logr.score(fs,target_vector)
```
Out[70]: 0.9888206926786497

Random Forest

In [71]:
```
grid_search.best_score_
```
Out[71]: 0.9921700776675565

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

In [ ]:
```

```