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("madrid_2014.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	3.0	10.0	NaN	NaN	NaN	3.0	NaN	NaN	28
1	2014-06-01 01:00:00	0.2	0.2	0.1	0.11	3.0	17.0	68.0	10.0	5.0	5.0	1.36	1.3	28
2	2014-06-01 01:00:00	0.3	NaN	0.1	NaN	2.0	6.0	NaN	NaN	NaN	NaN	NaN	1.1	28
3	2014-06-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	79.0	NaN	NaN	NaN	NaN	NaN	28
4	2014-06-01 01:00:00	NaN	NaN	NaN	NaN	1.0	6.0	75.0	NaN	NaN	4.0	NaN	NaN	28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210019	2014-09-01 00:00:00	NaN	0.5	NaN	NaN	20.0	84.0	29.0	NaN	NaN	NaN	NaN	NaN	28
210020	2014-09-01 00:00:00	NaN	0.3	NaN	NaN	1.0	22.0	NaN	15.0	NaN	6.0	NaN	NaN	28
210021	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	13.0	70.0	NaN	NaN	NaN	NaN	NaN	28
210022	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	38.0	42.0	NaN	NaN	NaN	NaN	NaN	28
210023	2014-09-01 00:00:00	NaN	NaN	NaN	NaN	1.0	26.0	65.0	11.0	NaN	NaN	NaN	NaN	28

210024 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', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.6+ MB
```

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

Out[6]:

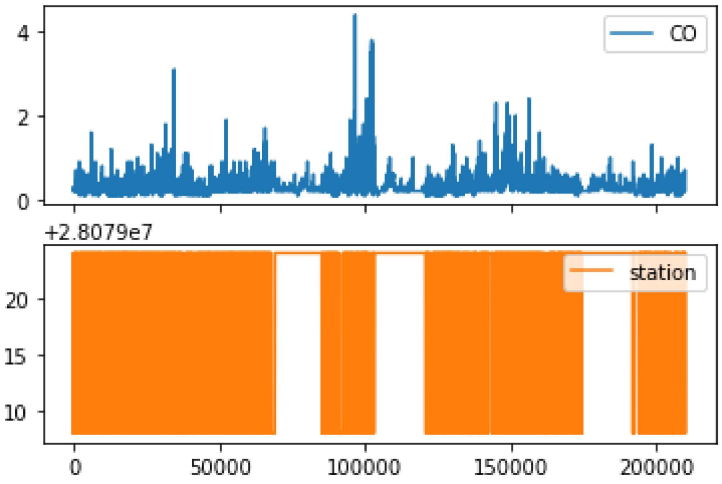
	CO	station
1	0.2	28079008
6	0.2	28079024
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
209958	0.2	28079024
209977	0.7	28079008
209982	0.2	28079024
210001	0.4	28079008
210006	0.2	28079024

13946 rows × 2 columns

Line chart

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

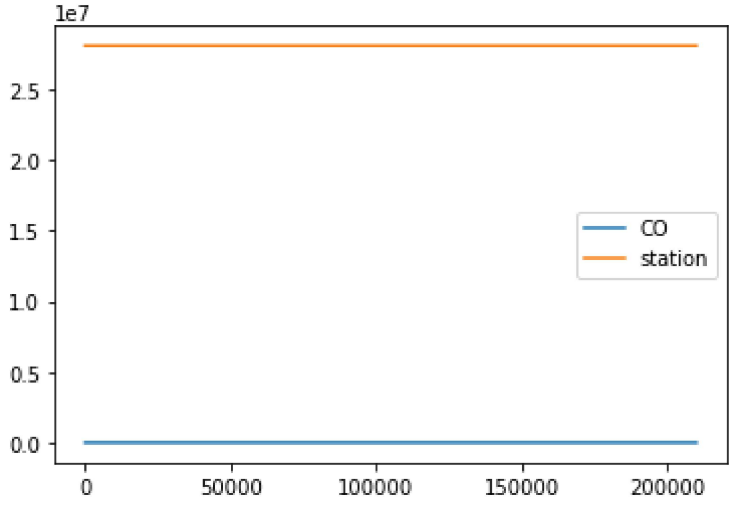
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]: <AxesSubplot:>

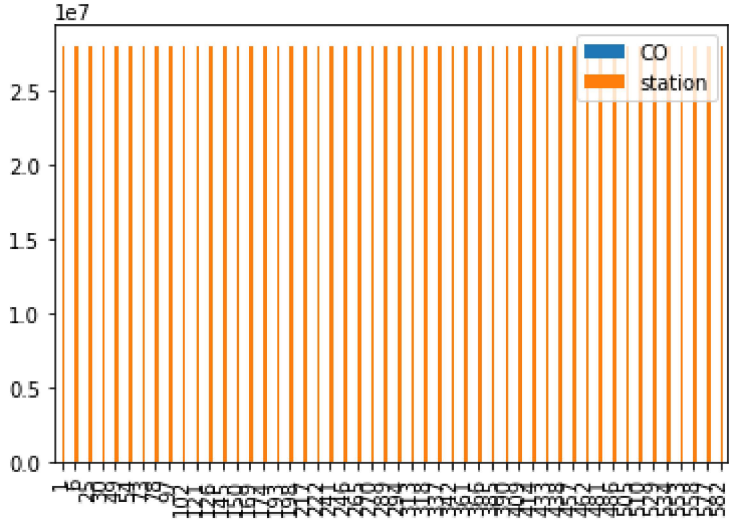


Bar chart

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

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

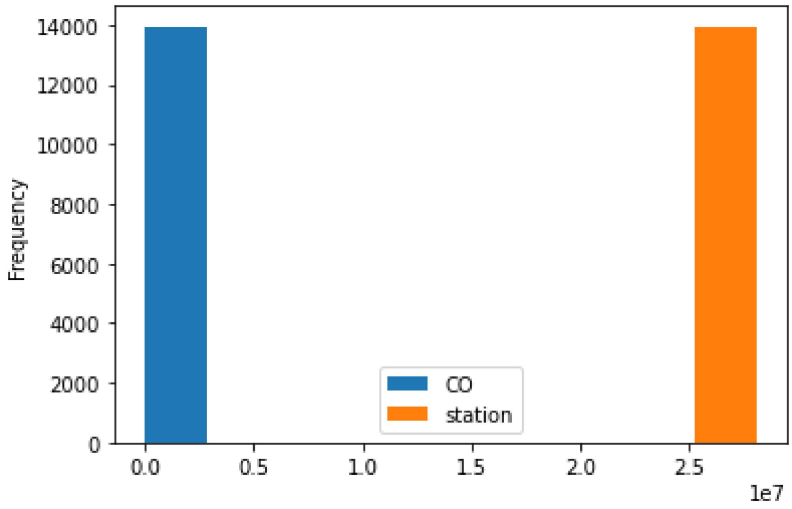
Out[10]: <AxesSubplot:>



Histogram

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

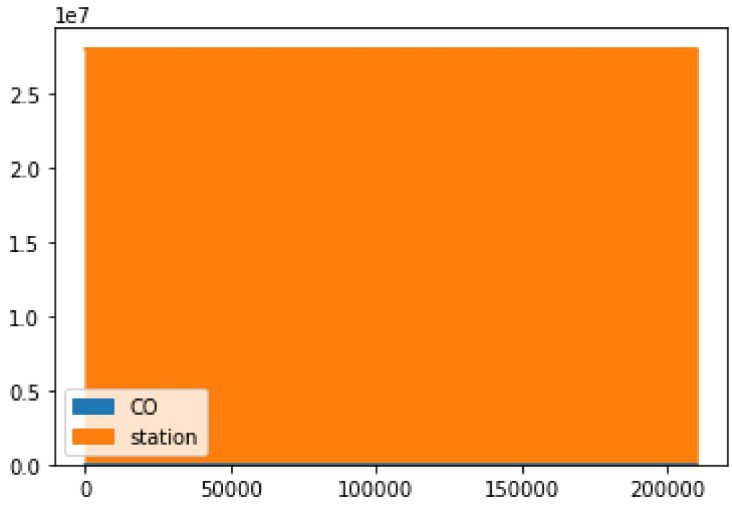
Out[11]:
```
<AxesSubplot:ylabel='Frequency'>
```



Area chart

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

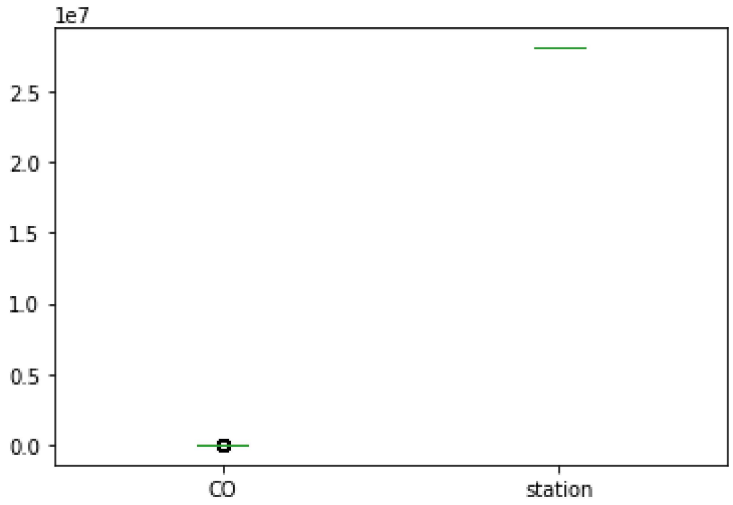
Out[12]:
```
<AxesSubplot:>
```



Box chart

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

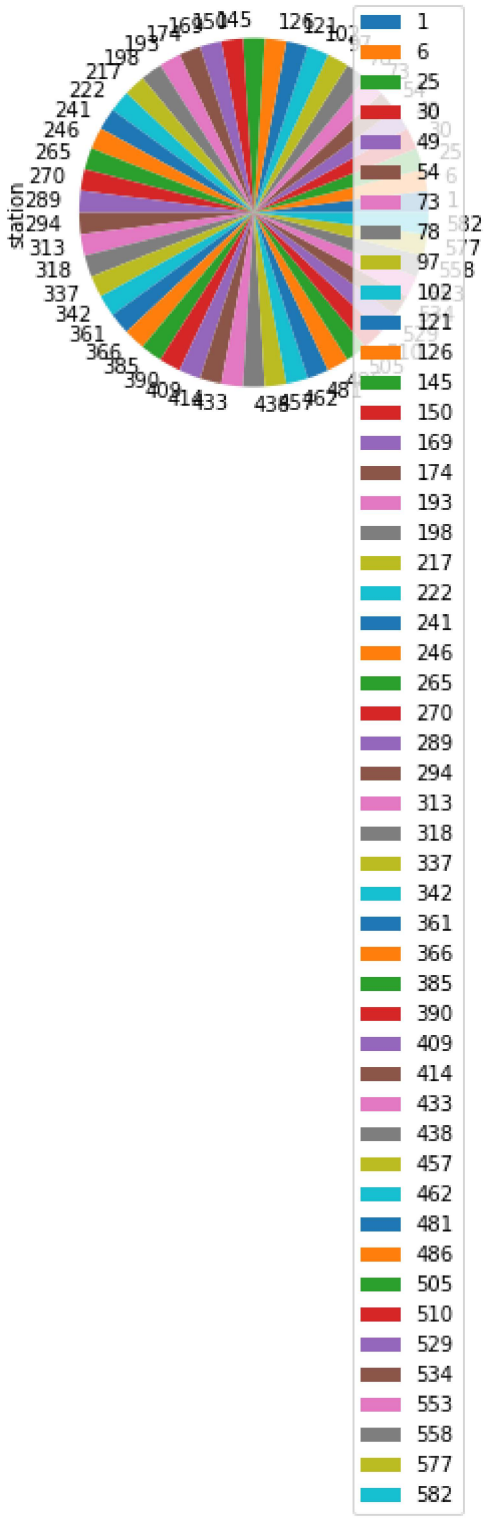
Out[13]: <AxesSubplot:>



Pie chart

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

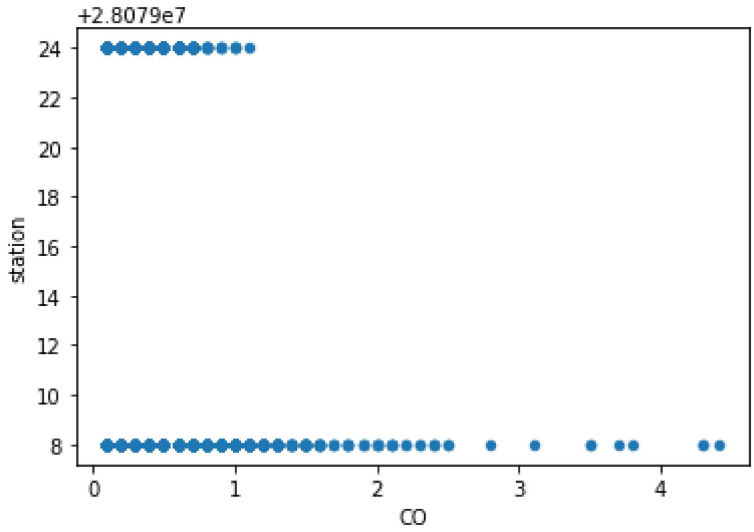
Out[14]: <AxesSubplot:ylabel='station'>



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: 13946 entries, 1 to 210006
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     13946 non-null  object 
 1   BEN      13946 non-null  float64
 2   CO       13946 non-null  float64
 3   EBE      13946 non-null  float64
 4   NMHC     13946 non-null  float64
 5   NO       13946 non-null  float64
 6   NO_2     13946 non-null  float64
 7   O_3      13946 non-null  float64
 8   PM10     13946 non-null  float64
 9   PM25     13946 non-null  float64
 10  SO_2     13946 non-null  float64
 11  TCH      13946 non-null  float64
 12  TOL      13946 non-null  float64
 13  station  13946 non-null  int64  
dtypes: float64(12), int64(1), object(1)
```

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

Out[17]:

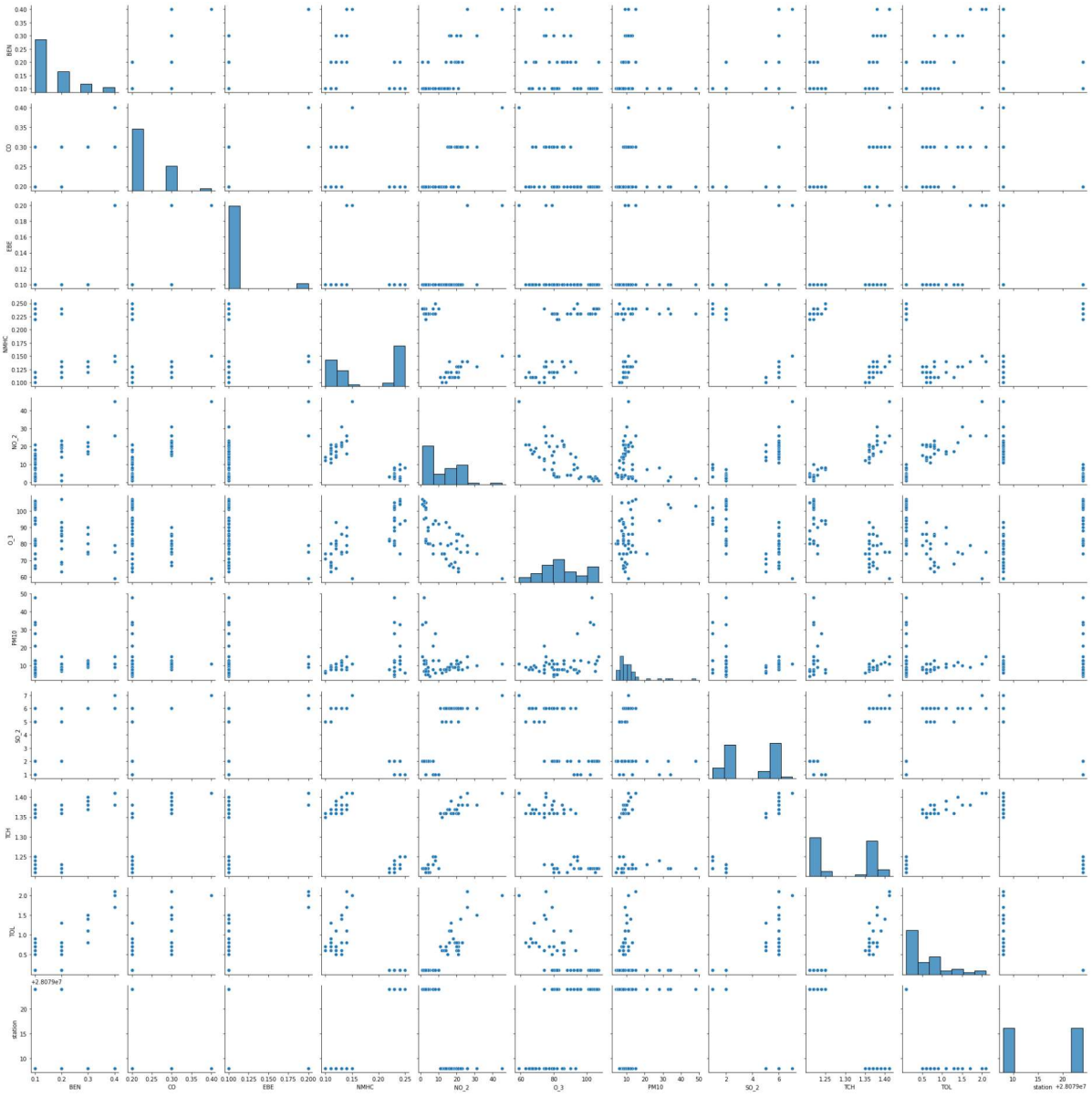
	BEN	CO	EBE	NMHC	NO	NO_2	
count	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	139
mean	0.375921	0.314793	0.306016	0.222302	17.589129	34.240929	
std	0.555093	0.207375	0.635475	0.082403	39.432216	30.654229	
min	0.100000	0.100000	0.100000	0.060000	1.000000	1.000000	
25%	0.100000	0.200000	0.100000	0.160000	1.000000	10.000000	
50%	0.200000	0.300000	0.100000	0.230000	4.000000	27.000000	
75%	0.400000	0.400000	0.300000	0.260000	18.000000	51.000000	
max	9.400000	4.400000	16.200001	1.290000	725.000000	346.000000	2

In [18]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL','station']]
```

EDA AND VISUALIZATION

In [19]:
```
sns.pairplot(df1[0:50])
```

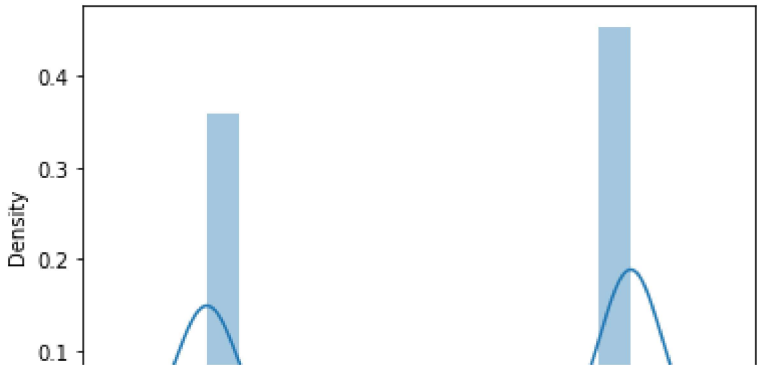
Out[19]: <seaborn.axisgrid.PairGrid at 0x22cf819a070>



In [20]:
```
sns.distplot(df1['station'])
```

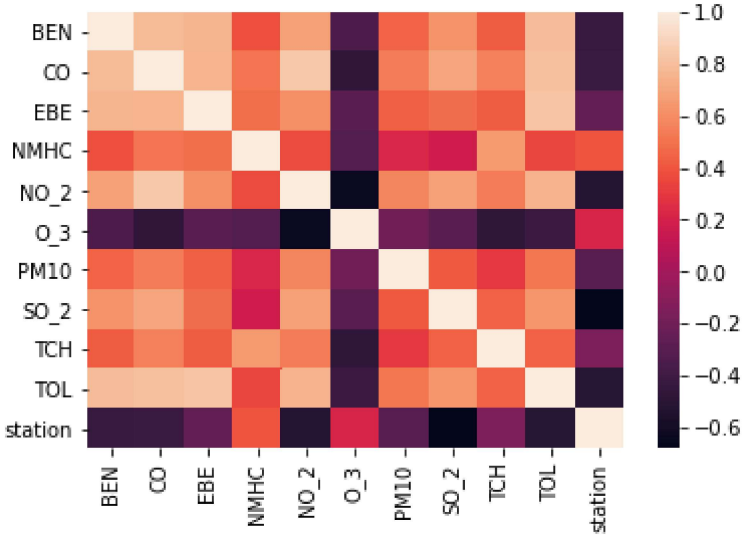
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [21]:
```
sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [23]:
```
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 [24]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
lr.intercept_
```

Out[25]:
```
28079022.090173364
```

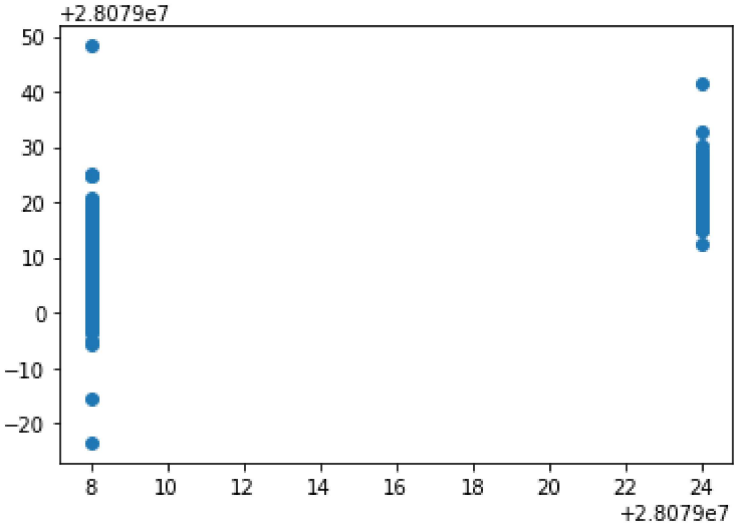
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-1.451791
CO	-5.020005
EBE	0.303522
NMHC	82.905443
NO_2	-0.032328
O_3	0.004185
PM10	0.013945
SO_2	-0.927889
TCH	-11.492904
TOL	-0.411399

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x22c828180a0>



ACCURACY

In [28]:
```
lr.score(x_test,y_test)
```

Out[28]: 0.8737212184307477

In [29]:
```
lr.score(x_train,y_train)
```

Out[29]: 0.8884206360581255

Ridge and Lasso

In [30]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [31]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[31]: Ridge(alpha=10)

Accuracy(Ridge)

In [32]:
```
rr.score(x_test,y_test)
```

Out[32]: 0.8473425406501551

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]:
```
0.8655640891434795
```

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]:
```
Lasso(alpha=10)
```

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

Out[35]:
```
0.2760473094771422
```

Accuracy(Lasso)

In [36]:
```
la.score(x_test,y_test)
```

Out[36]:
```
0.2627026380997839
```

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[37]:
```
ElasticNet()
```

In [38]:
```
en.coef_
```

Out[38]:
```
array([ 0.        ,  0.        ,  0.18751801,  0.        , -0.03810084,
       -0.00850831,  0.01480864, -1.30352408,  0.        , -0.17074891])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
28079024.803057823
```

In [40]:
```
prediction=en.predict(x_test)
```

In [41]:
```
en.score(x_test,y_test)
```

Out[41]:
```
0.45585961404763753
```

Evaluation Metrics

In [42]:
```
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)))
```

```
5.0144061560842275
34.58406846428023
5.880822090854524
```

Logistic Regression

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:
```
feature_matrix.shape
```

Out[45]:
```
(13946, 10)
```

In [46]:
```
target_vector.shape
```

Out[46]:
```
(13946,)
```

In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [49]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[49]:
```
LogisticRegression(max_iter=10000)
```

In [50]:
```
observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [51]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [52]:
```
logr.classes_
```

Out[52]:
```
array([28079008, 28079024], dtype=int64)
```

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]:
```
0.9926143697117453
```

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]:
```
1.0
```

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

Out[55]:
```
array([[1.00000000e+00, 5.27113072e-18]])
```

Random Forest

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[57]:
```
RandomForestClassifier()
```

In [58]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [59]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="ac
grid_search.fit(x_train,y_train)
```

Out[59]:
```
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 [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.9956976029502151
```

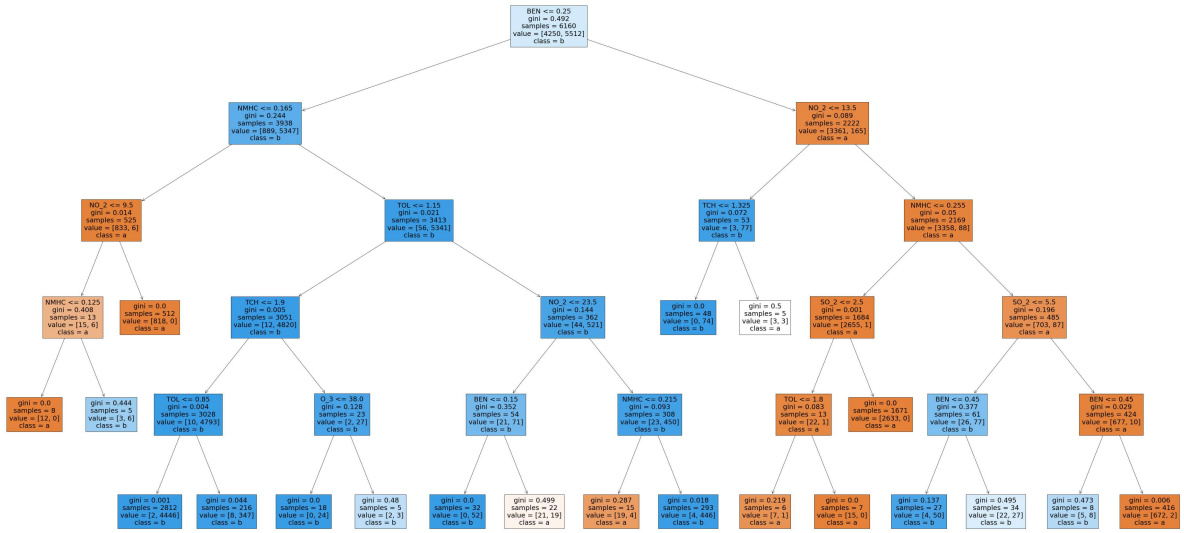
In [61]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:
```
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'
```

Out[62]:
```
[Text(2070.0, 1993.2, 'BEN <= 0.25\ngini = 0.492\nsamples = 6160\nvalue = [4250, 5512]\nclass = b'),
 Text(1008.0, 1630.8000000000002, 'NMHC <= 0.165\ngini = 0.244\nsamples = 3938\nvalue = [889, 5347]\nclass = b'),
 Text(432.0, 1268.4, 'NO_2 <= 9.5\ngini = 0.014\nsamples = 525\nvalue = [833, 6]\nclass = a'),
 Text(288.0, 906.0, 'NMHC <= 0.125\ngini = 0.408\nsamples = 13\nvalue = [15, 6]\nclass = a'),
 Text(144.0, 543.5999999999999, 'gini = 0.0\nsamples = 8\nvalue = [12, 0]\nclass = a'),
 Text(432.0, 543.5999999999999, 'gini = 0.444\nsamples = 5\nvalue = [3, 6]\nclass = b'),
 Text(576.0, 906.0, 'gini = 0.0\nsamples = 512\nvalue = [818, 0]\nclass = a'),
 Text(1584.0, 1268.4, 'TOL <= 1.15\ngini = 0.021\nsamples = 3413\nvalue = [56, 5341]\nclass = b'),
 Text(1008.0, 906.0, 'TCH <= 1.9\ngini = 0.005\nsamples = 3051\nvalue = [12, 4820]\nclass = b'),
 Text(720.0, 543.5999999999999, 'TOL <= 0.85\ngini = 0.004\nsamples = 3028\nvalue = [10, 4793]\nclass = b'),
 Text(576.0, 181.19999999999982, 'gini = 0.001\nsamples = 2812\nvalue = [2, 4446]\nclass = b'),
 Text(864.0, 181.19999999999982, 'gini = 0.044\nsamples = 216\nvalue = [8, 347]\nclass = b'),
 Text(1296.0, 543.5999999999999, 'O_3 <= 38.0\ngini = 0.128\nsamples = 23\nvalue = [2, 27]\nclass = b'),
 Text(1152.0, 181.19999999999982, 'gini = 0.0\nsamples = 18\nvalue = [0, 24]\nclass = b'),
 Text(1440.0, 181.19999999999982, 'gini = 0.48\nsamples = 5\nvalue = [2, 3]\nclass = b'),
 Text(2160.0, 906.0, 'NO_2 <= 23.5\ngini = 0.144\nsamples = 362\nvalue = [44, 521]\nclass = b'),
 Text(1872.0, 543.5999999999999, 'BEN <= 0.15\ngini = 0.352\nsamples = 54\nvalue = [21, 71]\nclass = b'),
 Text(1728.0, 181.19999999999982, 'gini = 0.0\nsamples = 32\nvalue = [0, 52]\nclass = b'),
 Text(2016.0, 181.19999999999982, 'gini = 0.499\nsamples = 22\nvalue = [21, 19]\nclass = a'),
 Text(2448.0, 543.5999999999999, 'NMHC <= 0.215\ngini = 0.093\nsamples = 308\nvalue = [23, 450]\nclass = b'),
 Text(2304.0, 181.19999999999982, 'gini = 0.287\nsamples = 15\nvalue = [19, 4]\nclass = a'),
 Text(2592.0, 181.19999999999982, 'gini = 0.018\nsamples = 293\nvalue = [4, 446]\nclass = b'),
 Text(3132.0, 1630.8000000000002, 'NO_2 <= 13.5\ngini = 0.089\nsamples = 2222\nvalue = [3361, 165]\nclass = a'),
 Text(2736.0, 1268.4, 'TCH <= 1.325\ngini = 0.072\nsamples = 53\nvalue = [3, 77]\nclass = b'),
 Text(2592.0, 906.0, 'gini = 0.0\nsamples = 48\nvalue = [0, 74]\nclass = b'),
 Text(2880.0, 906.0, 'gini = 0.5\nsamples = 5\nvalue = [3, 3]\nclass = a'),
 Text(3528.0, 1268.4, 'NMHC <= 0.255\ngini = 0.05\nsamples = 2169\nvalue = [3358, 88]\nclass = a'),
 Text(3168.0, 906.0, 'SO_2 <= 2.5\ngini = 0.001\nsamples = 1684\nvalue = [2655, 1]\nclass = a'),
 Text(3024.0, 543.5999999999999, 'TOL <= 1.8\ngini = 0.083\nsamples = 13\nvalue = [22, 1]\nclass = a'),
 Text(2880.0, 181.19999999999982, 'gini = 0.219\nsamples = 6\nvalue = [7, 1]
```

```
\nclass = a'),
 Text(3168.0, 181.19999999999982, 'gini = 0.0\nsamples = 7\nvalue = [15, 0]\n
class = a'),
 Text(3312.0, 543.5999999999999, 'gini = 0.0\nsamples = 1671\nvalue = [2633,
0]\nclass = a'),
 Text(3888.0, 906.0, 'SO_2 <= 5.5\ngini = 0.196\nsamples = 485\nvalue = [703,
87]\nclass = a'),
 Text(3600.0, 543.5999999999999, 'BEN <= 0.45\ngini = 0.377\nsamples = 61\nva
lue = [26, 77]\nclass = b'),
 Text(3456.0, 181.19999999999982, 'gini = 0.137\nsamples = 27\nvalue = [4, 5
0]\nclass = b'),
 Text(3744.0, 181.19999999999982, 'gini = 0.495\nsamples = 34\nvalue = [22, 2
7]\nclass = b'),
 Text(4176.0, 543.5999999999999, 'BEN <= 0.45\ngini = 0.029\nsamples = 424\nv
alue = [677, 10]\nclass = a'),
 Text(4032.0, 181.19999999999982, 'gini = 0.473\nsamples = 8\nvalue = [5, 8]
\nclass = b'),
 Text(4320.0, 181.19999999999982, 'gini = 0.006\nsamples = 416\nvalue = [672,
2]\nclass = a')]
```



Conclusion

Accuracy

In [63]:
```
lr.score(x_train,y_train)
```

Out[63]: 0.8884206360581255

In [64]:
```
rr.score(x_train,y_train)
```

Out[64]: 0.8655640891434795

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

Out[65]: 0.2760473094771422

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

Out[66]: 0.45585961404763753

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

Out[67]: 0.9926143697117453

In [68]:
```
grid_search.best_score_
```

Out[68]: 0.9956976029502151

Random Forest is suitable for this dataset

In [ ]:
```

```