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").fillna(1)
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	2014-06-01 01:00:00	1.0	0.2	1.0	1.00	3.0	10.0	1.0	1.0	1.0	3.0	1.00	1.0	28079004
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	28079008
2	2014-06-01 01:00:00	0.3	1.0	0.1	1.00	2.0	6.0	1.0	1.0	1.0	1.0	1.00	1.1	28079011
3	2014-06-01 01:00:00	1.0	0.2	1.0	1.00	1.0	6.0	79.0	1.0	1.0	1.0	1.00	1.0	28079016
4	2014-06-01 01:00:00	1.0	1.0	1.0	1.00	1.0	6.0	75.0	1.0	1.0	4.0	1.00	1.0	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210019	2014-09-01 00:00:00	1.0	0.5	1.0	1.00	20.0	84.0	29.0	1.0	1.0	1.0	1.00	1.0	28079056
210020	2014-09-01 00:00:00	1.0	0.3	1.0	1.00	1.0	22.0	1.0	15.0	1.0	6.0	1.00	1.0	28079057
210021	2014-09-01 00:00:00	1.0	1.0	1.0	1.00	1.0	13.0	70.0	1.0	1.0	1.0	1.00	1.0	28079058
210022	2014-09-01 00:00:00	1.0	1.0	1.0	1.00	3.0	38.0	42.0	1.0	1.0	1.0	1.00	1.0	28079059
210023	2014-09-01 00:00:00	1.0	1.0	1.0	1.00	1.0	26.0	65.0	11.0	1.0	1.0	1.00	1.0	28079060

210024 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:
```
1 df.columns
```

Out[3]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [4]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210024 entries, 0 to 210023
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210024 non-null  object 
 1   BEN      210024 non-null  float64
 2   CO       210024 non-null  float64
 3   EBE      210024 non-null  float64
 4   NMHC     210024 non-null  float64
 5   NO       210024 non-null  float64
 6   NO_2     210024 non-null  float64
 7   O_3      210024 non-null  float64
 8   PM10     210024 non-null  float64
 9   PM25     210024 non-null  float64
 10  SO_2     210024 non-null  float64
 11  TCH      210024 non-null  float64
 12  TOL      210024 non-null  float64
 13  station  210024 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

In [5]:
```
data=df[['CO' ,'station']]
data
```

Out[5]:

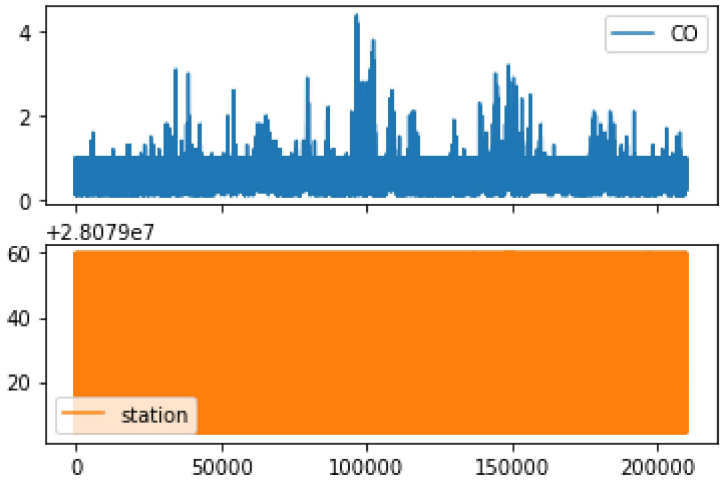
	CO	station
0	0.2	28079004
1	0.2	28079008
2	1.0	28079011
3	0.2	28079016
4	1.0	28079017
...	...	...
210019	0.5	28079056
210020	0.3	28079057
210021	1.0	28079058
210022	1.0	28079059
210023	1.0	28079060

210024 rows × 2 columns

Line chart

In [6]:
```
1 data.plot.line(subplots=True)
```

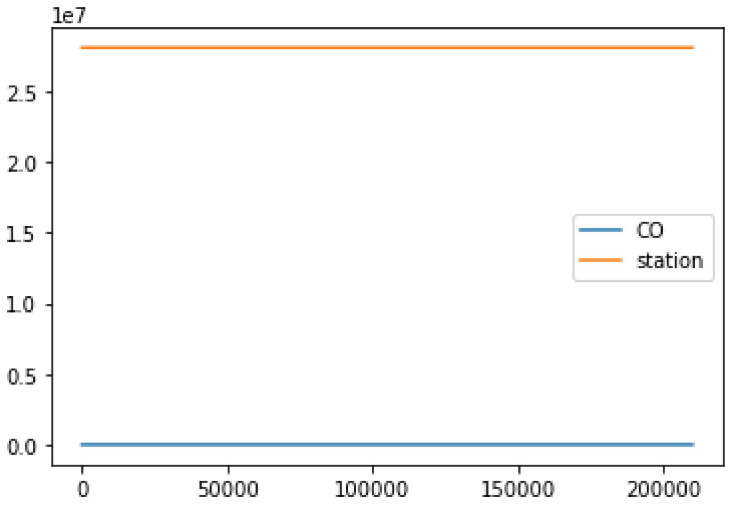
Out[6]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [7]:
```
1 data.plot.line()
```

Out[7]:
```
<AxesSubplot:>
```

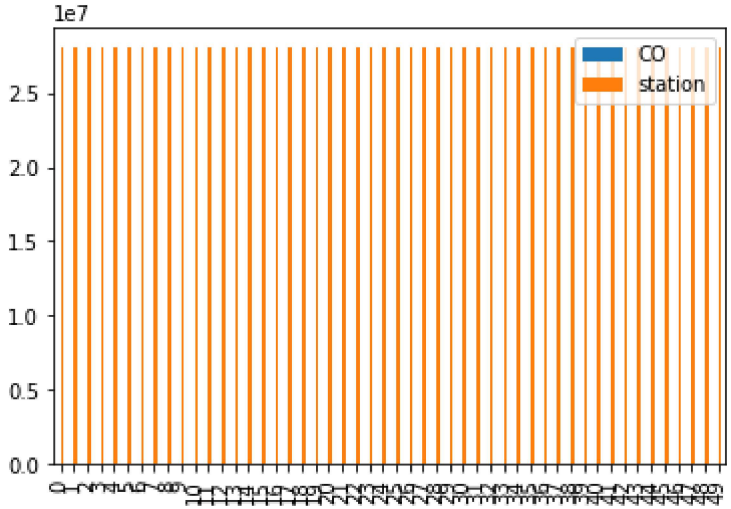


Bar chart

In [8]:
```
1 b=data[0:50]
```

In [9]:
```
1 b.plot.bar()
```

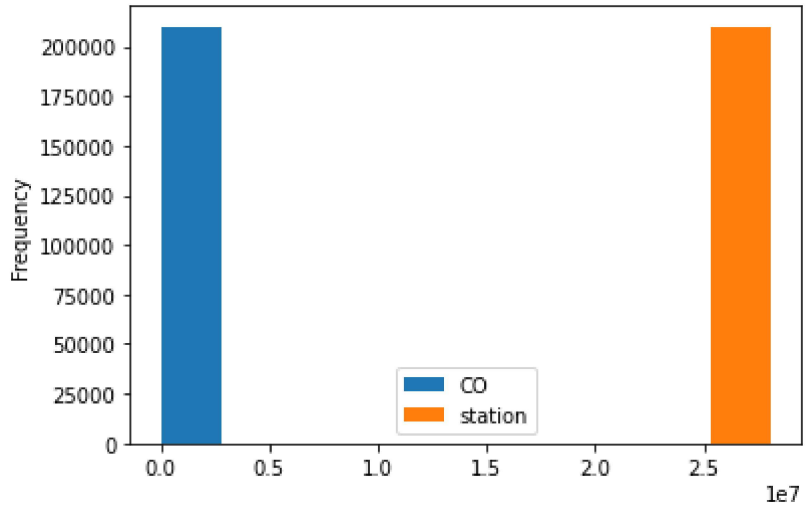
Out[9]: <AxesSubplot:>



Histogram

In [10]:
```
1 data.plot.hist()
```

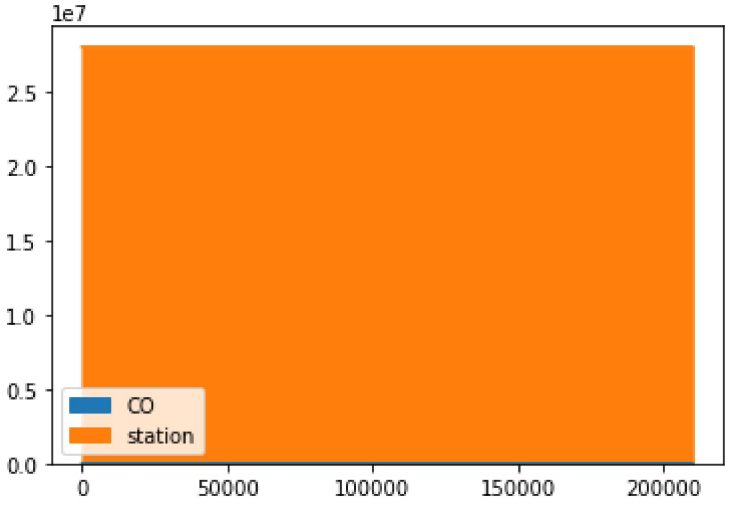
Out[10]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [11]:
```
1 data.plot.area()
```

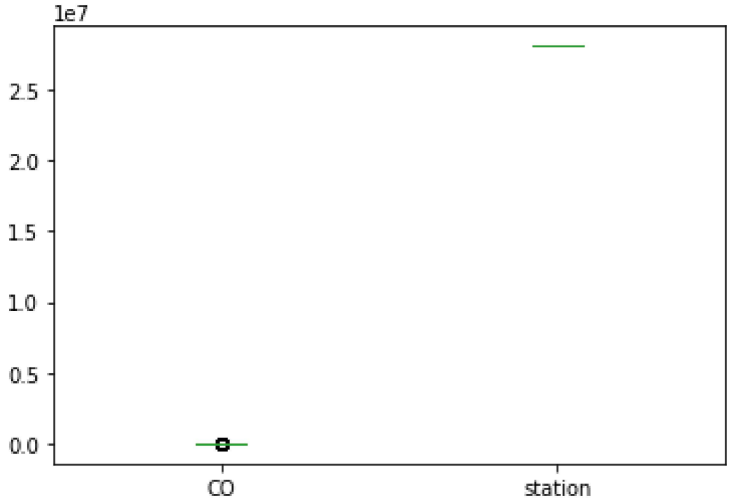
Out[11]: <AxesSubplot:>



Box chart

In [12]:
```
1 data.plot.box()
```

Out[12]: <AxesSubplot:>

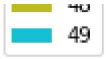


Pie chart

In [13]:
```
1 b.plot.pie(y='station' )
```

Out[13]:
```
<AxesSubplot:ylabel='station'>
```

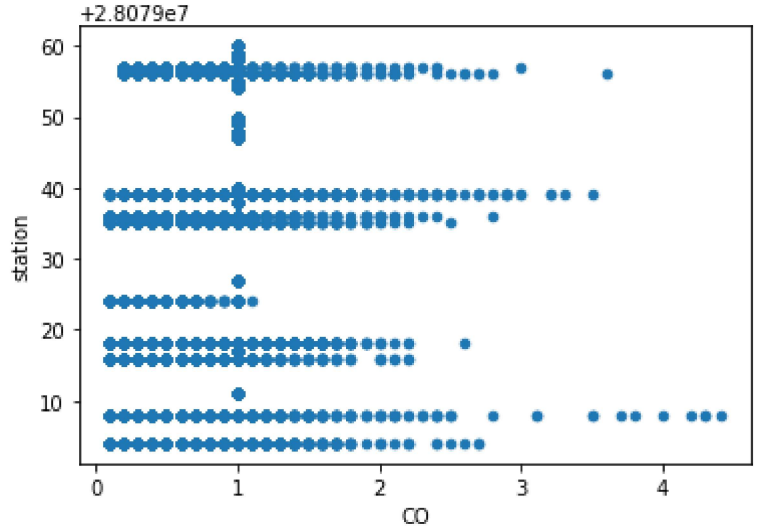

station
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47



Scatter chart

In [14]:
```
1 data.plot.scatter(x='CO' ,y='station')
```

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



In [15]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210024 entries, 0 to 210023
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210024 non-null  object 
 1   BEN      210024 non-null  float64
 2   CO       210024 non-null  float64
 3   EBE      210024 non-null  float64
 4   NMHC     210024 non-null  float64
 5   NO       210024 non-null  float64
 6   NO_2     210024 non-null  float64
 7   O_3      210024 non-null  float64
 8   PM10     210024 non-null  float64
 9   PM25     210024 non-null  float64
 10  SO_2     210024 non-null  float64
 11  TCH      210024 non-null  float64
 12  TOL      210024 non-null  float64
 13  station  210024 non-null  int64
```

In [16]:
```
1 df.describe()
```

Out[16]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	
count	210024.000000	210024.000000	210024.000000	210024.000000	210024.000000	210024.000000	210024.000000	210024.000000	21002
mean	0.929350	0.738237	0.882264	0.913692	19.894774	34.912305	30.340047	10.083376	
std	0.448200	0.348712	0.441303	0.240590	44.766042	28.420739	35.374036	14.022661	
min	0.100000	0.100000	0.100000	0.040000	1.000000	1.000000	1.000000	1.000000	
25%	1.000000	0.300000	1.000000	1.000000	2.000000	14.000000	1.000000	1.000000	
50%	1.000000	1.000000	1.000000	1.000000	5.000000	27.000000	11.000000	1.000000	
75%	1.000000	1.000000	1.000000	1.000000	17.000000	49.000000	57.000000	15.000000	
max	17.799999	4.400000	16.200001	1.590000	925.000000	416.000000	220.000000	200.000000	17

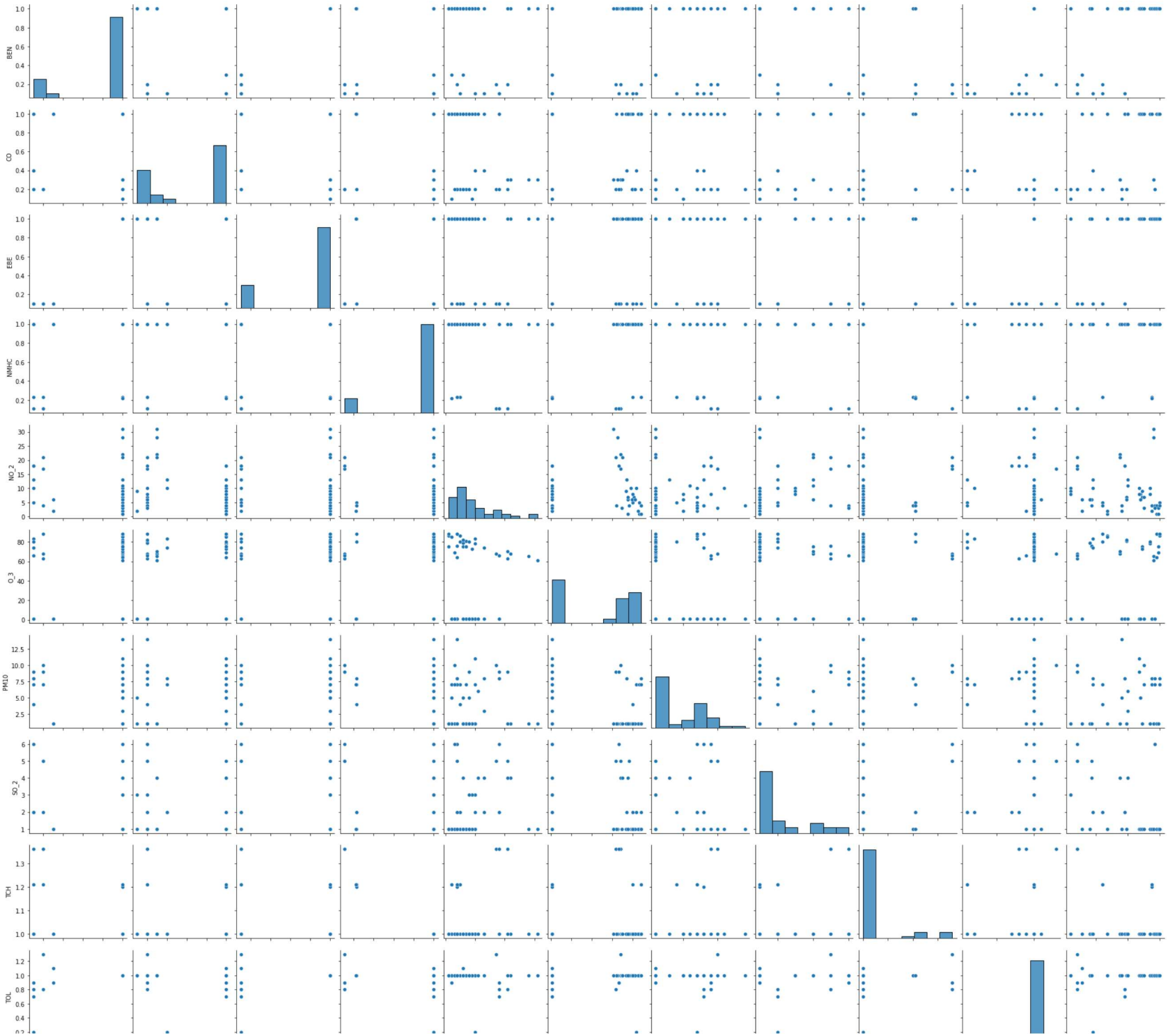
In [17]:

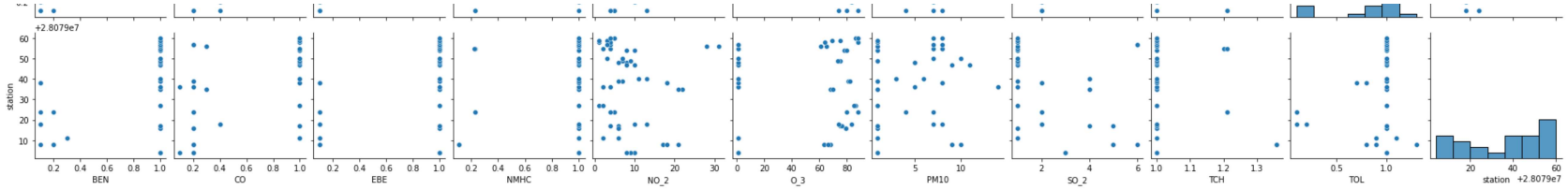
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [18]:
```
1 sns.pairplot(df1[0:50])
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x1f6d4048160>
```

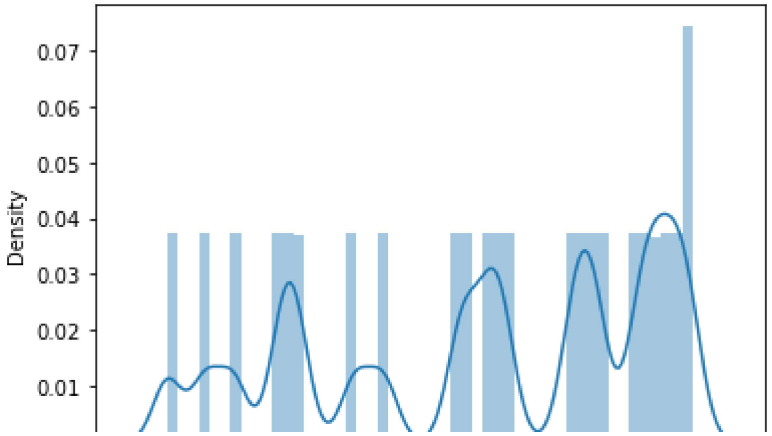

BEN
CO
EBE
NMHC
NO_2
O_3
PM10
SO_2
TCH
TOL



In [19]:
```
1 sns.distplot(df1['station'])
```

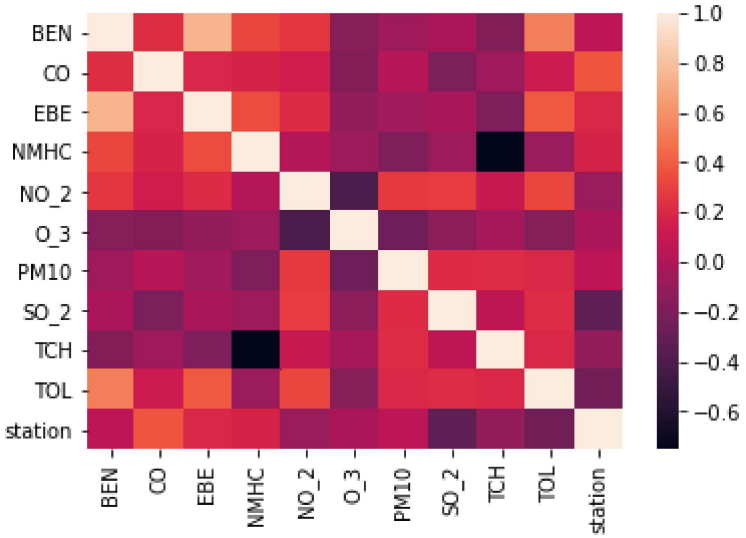
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[19]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [20]:
```
1 sns.heatmap(df1.corr())
```

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
1 x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
2        'PM10', 'SO_2', 'TCH', 'TOL']]
3 y=df['station']
```

In [22]:
```
1 from sklearn.model_selection import train_test_split
2 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Linear Regression

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

Out[23]:
```
LinearRegression()
```

In [24]:
```
lr.intercept_
```

Out[24]:
```
28079002.290398657
```

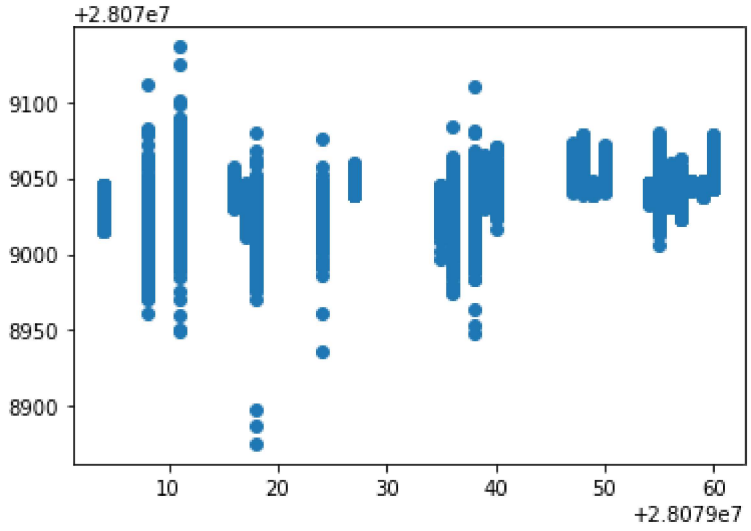
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	-1.165043
CO	16.148953
EBE	13.098257
NMHC	7.615914
NO_2	-0.052439
O_3	0.013252
PM10	0.258949
SO_2	-1.099325
TCH	12.392951
TOL	-3.687351

In [26]:
```
1 prediction =lr.predict(x_test)
2 plt.scatter(y_test,prediction)
```

Out[26]: <matplotlib.collections.PathCollection at 0x1f6e23357f0>



ACCURACY

In [27]:
```
1 lr.score(x_test,y_test)
```

Out[27]: 0.3596258819547221

In [28]:
```
1 lr.score(x_train,y_train)
```

Out[28]: 0.36177552696481163

Ridge and Lasso

In [29]:
```
1 from sklearn.linear_model import Ridge,Lasso
```

In [30]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[30]: Ridge(alpha=10)

Accuracy(Ridge)

In [31]:
```
rr.score(x_test,y_test)
```

Out[31]: 0.3596274424313378

In [32]:
```
rr.score(x_train,y_train)
```

Out[32]: 0.3617750872178387

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

Out[33]: Lasso(alpha=10)

Accuracy(Lasso)

In [34]:
```
la.score(x_train,y_train)
```

Out[34]: 0.08092617252882062

In [35]:
```
la.score(x_test,y_test)
```

Out[35]: 0.0782856061639382

In [36]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[36]: ElasticNet()

In [37]:
```
1 en.coef_
```

Out[37]:
```
array([ 0.75408637,  2.34331603,  2.22463261,  0.19730868, -0.01467279,
       -0.0104731 ,  0.20436227, -1.60710152, -0.        , -1.95064468])
```

In [38]:
```
1 en.intercept_
```

Out[38]:
```
28079038.977202713
```

In [39]:
```
1 prediction=en.predict(x_test)
```

In [40]:
```
1 en.score(x_test,y_test)
```

Out[40]:
```
0.21968666281164606
```

Evaluation Metrics

In [41]:
```
1 from sklearn import metrics
2 print(metrics.mean_absolute_error(y_test,prediction))
3 print(metrics.mean_squared_error(y_test,prediction))
4 print(np.sqrt(metrics.mean_squared_error(y_test,prediction)))
```

```
13.108874793230651
243.2640730978318
15.596925116760413
```

Logistic Regression

In [42]:
```
1 from sklearn.linear_model import LogisticRegression
```

In [43]:
```
1 feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
2        'PM10', 'SO_2', 'TCH', 'TOL']]
3 target_vector=df[ 'station']
```

In [44]:
```
1 feature_matrix.shape
```

Out[44]:
```
(210024, 10)
```

In [45]:
```
1 target_vector.shape
```

Out[45]:
```
(210024,)
```

In [46]:
```
1 from sklearn.preprocessing import StandardScaler
```

In [47]:
```
1 fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:
```
1 logr=LogisticRegression(max_iter=10000)
2 logr.fit(fs,target_vector)
```

Out[48]:
```
LogisticRegression(max_iter=10000)
```

In [49]:
```
1 observation=[[1,2,3,4,5,6,7,8,9,10]]
```

In [50]:
```
1 prediction=logr.predict(observation)
2 print(prediction)
```

```
[28079018]
```

In [51]:
```
1 logr.classes_
```

Out[51]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [52]:
```
1 logr.score(fs,target_vector)
```

Out[52]:
```
0.6760370243400754
```

In [53]:
```
1 logr.predict_proba(observation)[0][0]
```

Out[53]:
```
1.813134240601751e-204
```

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

Out[54]:
```
array([[1.81313424e-204, 4.50388872e-003, 1.85252720e-155,
        4.19032650e-117, 1.37882214e-070, 9.95496111e-001,
        8.02773684e-016, 9.03717527e-150, 6.73304007e-076,
        1.73878018e-080, 7.58142560e-099, 2.52014820e-115,
        5.67602012e-080, 8.04903369e-165, 6.26569208e-168,
        3.72030116e-169, 8.24543622e-171, 9.15991727e-160,
        4.43382154e-097, 1.99369412e-118, 5.06333324e-082,
        1.69417108e-172, 2.05952508e-168, 1.99296232e-068]])
```

Random Forest

In [55]:
```
1 from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
1 rfc=RandomForestClassifier()
2 rfc.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

In [57]:
```
1 parameters={'max_depth':[1,2,3,4,5],
2             'min_samples_leaf':[5,10,15,20,25],
3             'n_estimators':[10,20,30,40,50]
4 }
```

In [58]:
```
1 from sklearn.model_selection import GridSearchCV
2 grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accuracy")
3 grid_search.fit(x_train,y_train)
```

Out[58]:
```
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 [59]:
```
1 grid_search.best_score_
```

Out[59]:
```
0.6849730641562823
```

In [60]:
```
rfc_best=grid_search.best_estimator_
```

In [61]:
```
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','d','e','f','g','h','i
```

Out[61]:
```
[Text(2148.3, 1993.2, 'SO_2 <= 1.5\ngini = 0.958\nsamples = 92861\nvalue = [6224, 5992, 6134, 5975, 6141, 6
036, 6145, 6147, 6205\n6147, 6000, 6109, 6091, 6151, 6191, 6148, 6177, 6061\n6266, 6176, 6165, 6058, 6220,
6057]\nclass = s'),
 Text(1023.0000000000001, 1630.8000000000002, 'NMHC <= 0.995\ngini = 0.932\nsamples = 55927\nvalue = [25, 7
3, 6134, 5975, 60, 88, 545, 6147, 20, 1801\n10, 6109, 59, 6151, 6191, 6148, 6177, 6061, 6266\n6176, 13, 605
8, 6220, 6057]\nclass = s'),
 Text(558.0, 1268.4, 'EBE <= 0.95\ngini = 0.347\nsamples = 4826\nvalue = [0, 22, 0, 0, 0, 0, 435, 1136, 0,
0, 0, 0, 0\n0, 0, 0, 0, 0, 6076, 0, 0, 0, 0, 0]\nclass = s'),
 Text(297.6, 906.0, 'CO <= 0.8\ngini = 0.236\nsamples = 2078\nvalue = [0, 19, 0, 0, 0, 0, 429, 0, 0, 0, 0,
0, 0, 0\n0, 0, 0, 0, 2850, 0, 0, 0, 0, 0]\nclass = s'),
 Text(148.8, 543.5999999999999, 'TOL <= 0.85\ngini = 0.078\nsamples = 268\nvalue = [0, 18, 0, 0, 0, 0, 425,
0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(74.4, 181.19999999999982, 'gini = 0.005\nsamples = 243\nvalue = [0, 1, 0, 0, 0, 0, 400, 0, 0, 0, 0,
0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.482\nsamples = 25\nvalue = [0, 17, 0, 0, 0, 0, 25,
0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\nclass = g'),
 Text(446.40000000000003, 543.5999999999999, 'NO_2 <= 2.5\ngini = 0.003\nsamples = 1810\nvalue = [0, 1, 0,
0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 2850, 0, 0, 0, 0, 0]\nclass = s'),
 Text(372.0, 181.19999999999982, 'gini = 0.18\nsamples = 31\nvalue = [0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0,
```

Conclusion

Accuracy

Linear Regression:0.36177552696481163

Ridge Regression:0.3617750872178387

Lasso Regression:0.08092617252882062

ElasticNet Regression:0.21968666281164606

Logistic Regression:0.6760370243400754

Random Forest:0.6849730641562823

Random Forest is suitable for this dataset