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(r"C:\Users\user\Downloads\madrid_2014.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	01-06-2014 01:00	NaN	0.2	NaN	NaN	3.0	10.0	NaN	NaN	NaN	3.0	NaN	NaN	28079004
1	01-06-2014 01:00	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	01-06-2014 01:00	0.3	NaN	0.1	NaN	2.0	6.0	NaN	NaN	NaN	NaN	NaN	1.1	28079011
3	01-06-2014 01:00	NaN	0.2	NaN	NaN	1.0	6.0	79.0	NaN	NaN	NaN	NaN	NaN	28079016
4	01-06-2014 01:00	NaN	NaN	NaN	NaN	1.0	6.0	75.0	NaN	NaN	4.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210019	01-09-2014 00:00	NaN	0.5	NaN	NaN	20.0	84.0	29.0	NaN	NaN	NaN	NaN	NaN	28079056
210020	01-09-2014 00:00	NaN	0.3	NaN	NaN	1.0	22.0	NaN	15.0	NaN	6.0	NaN	NaN	28079057
210021	01-09-2014 00:00	NaN	NaN	NaN	NaN	1.0	13.0	70.0	NaN	NaN	NaN	NaN	NaN	28079058

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
210022	01-09-2014 00:00	NaN	NaN	NaN	NaN	3.0	38.0	42.0	NaN	NaN	NaN	NaN	NaN	28079059
210023	01-09-2014 00:00	NaN	NaN	NaN	NaN	1.0	26.0	65.0	11.0	NaN	NaN	NaN	NaN	28079060

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', 'PM25',
       '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

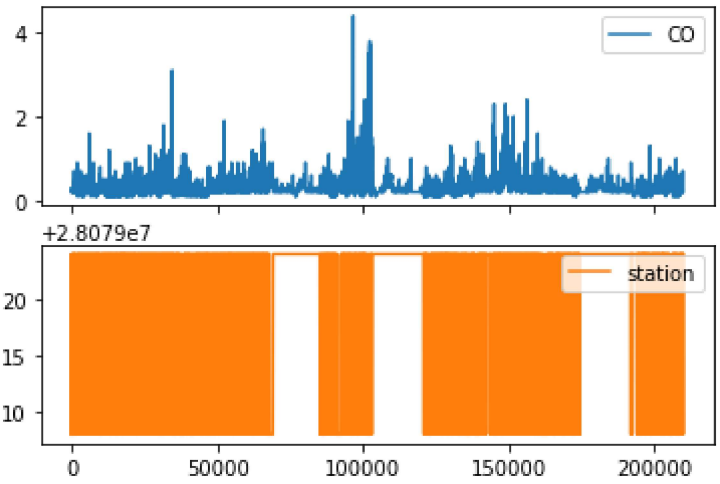
	CO	station
25	0.2	28079008
30	0.2	28079024
49	0.2	28079008
...	...	...
209958	0.2	28079024
209977	0.7	28079008
209982	0.2	28079024
210001	0.4	28079008
210006	0.2	28079024

13946 rows × 2 columns

Line chart

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

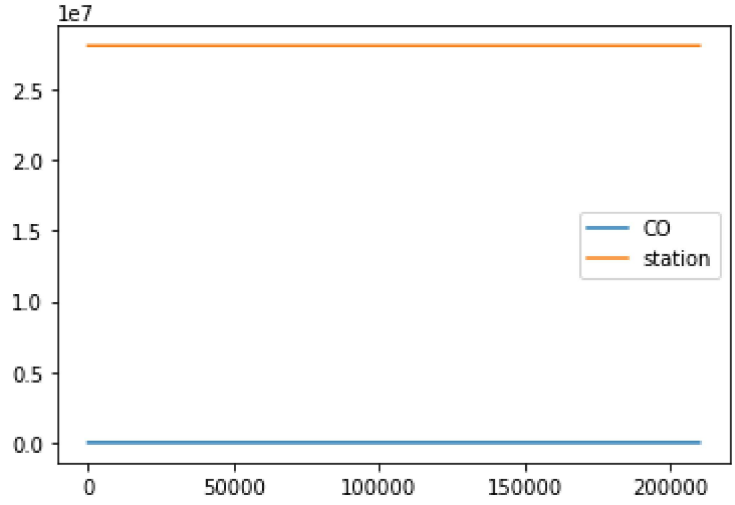
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]:
```
<AxesSubplot:>
```

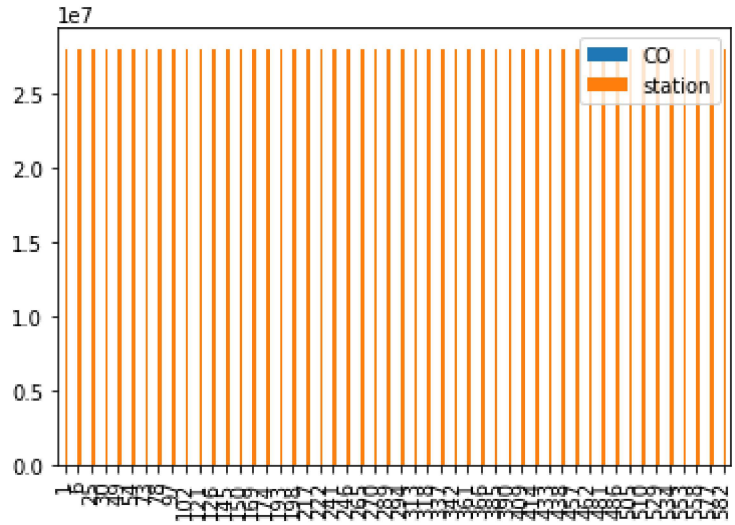


Bar chart

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

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

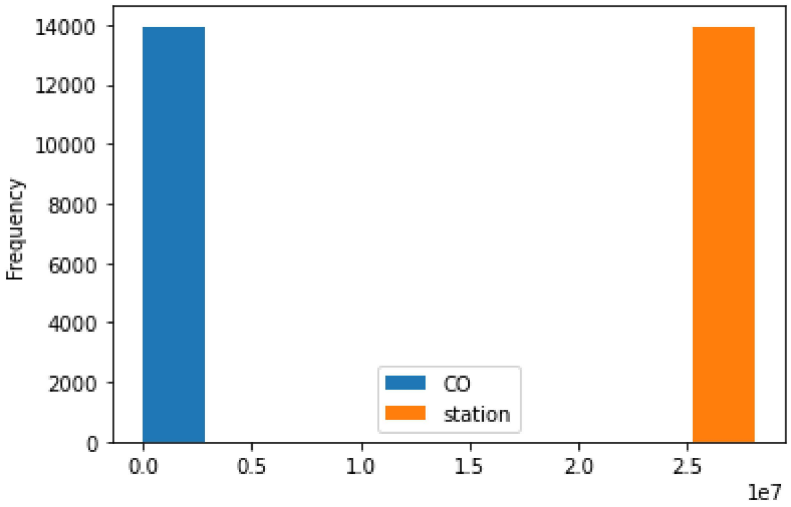
Out[10]: <AxesSubplot:>



Histogram

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

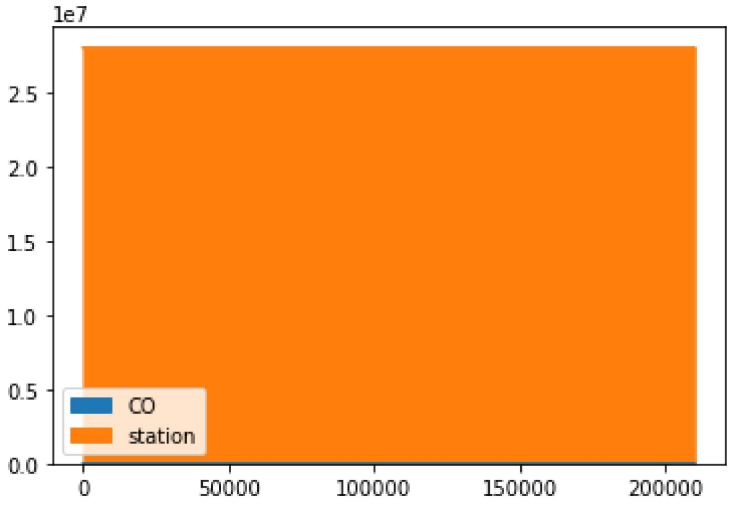
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

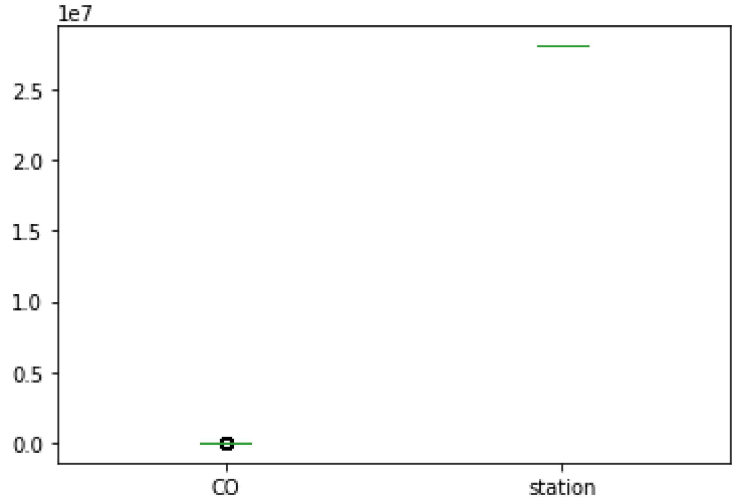
Out[12]: <AxesSubplot:>



Box chart

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

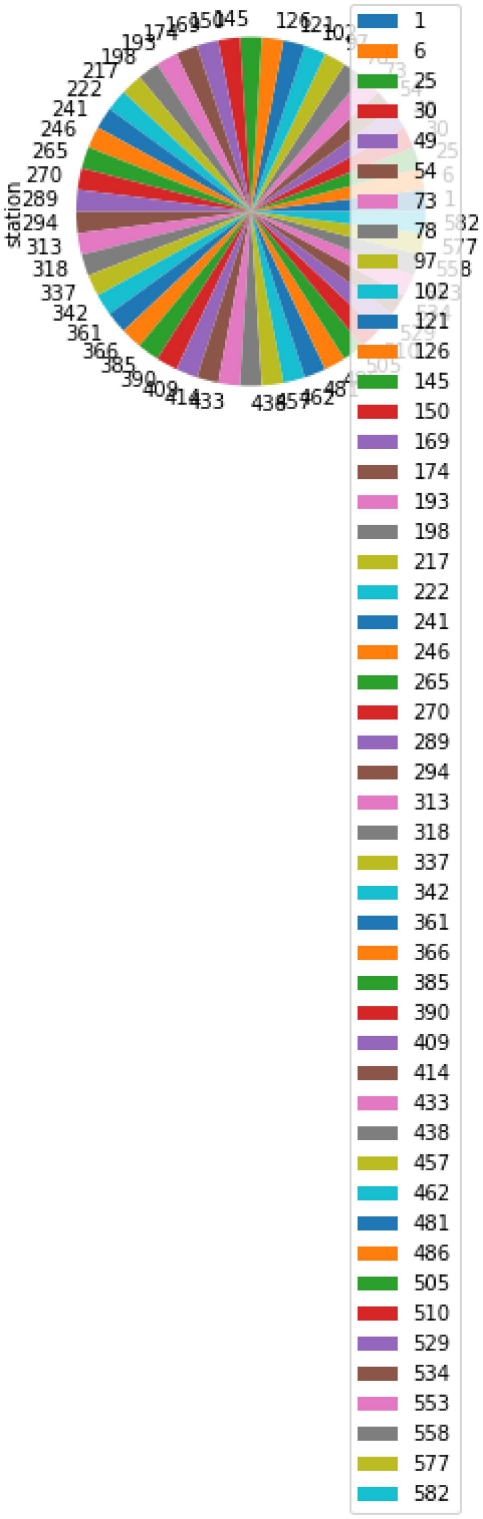
Out[13]: <AxesSubplot:>



Pie chart

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

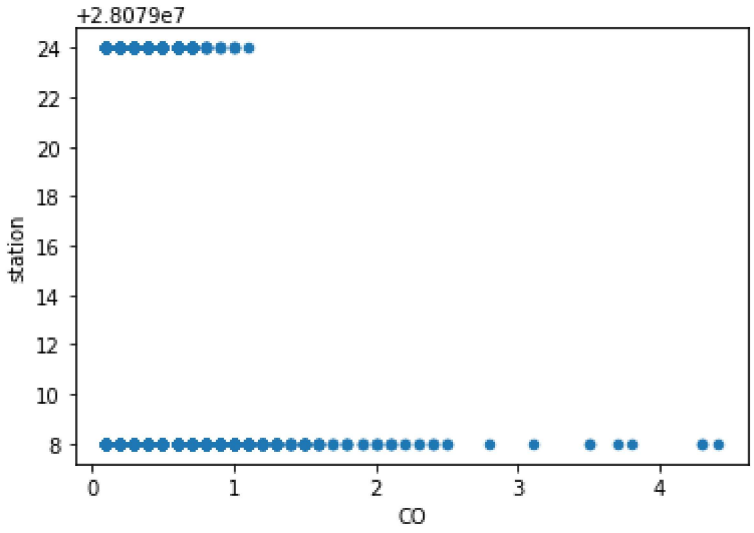
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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)
memory usage: 1.6+ MB
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [18]:
```
df.describe()
```

Out[18]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000	13946.000000
mean	0.375921	0.314793	0.306016	0.222302	17.589129	34.240929	53.082385
std	0.555093	0.207375	0.635475	0.082403	39.432216	30.654229	33.488161
min	0.100000	0.100000	0.100000	0.060000	1.000000	1.000000	1.000000

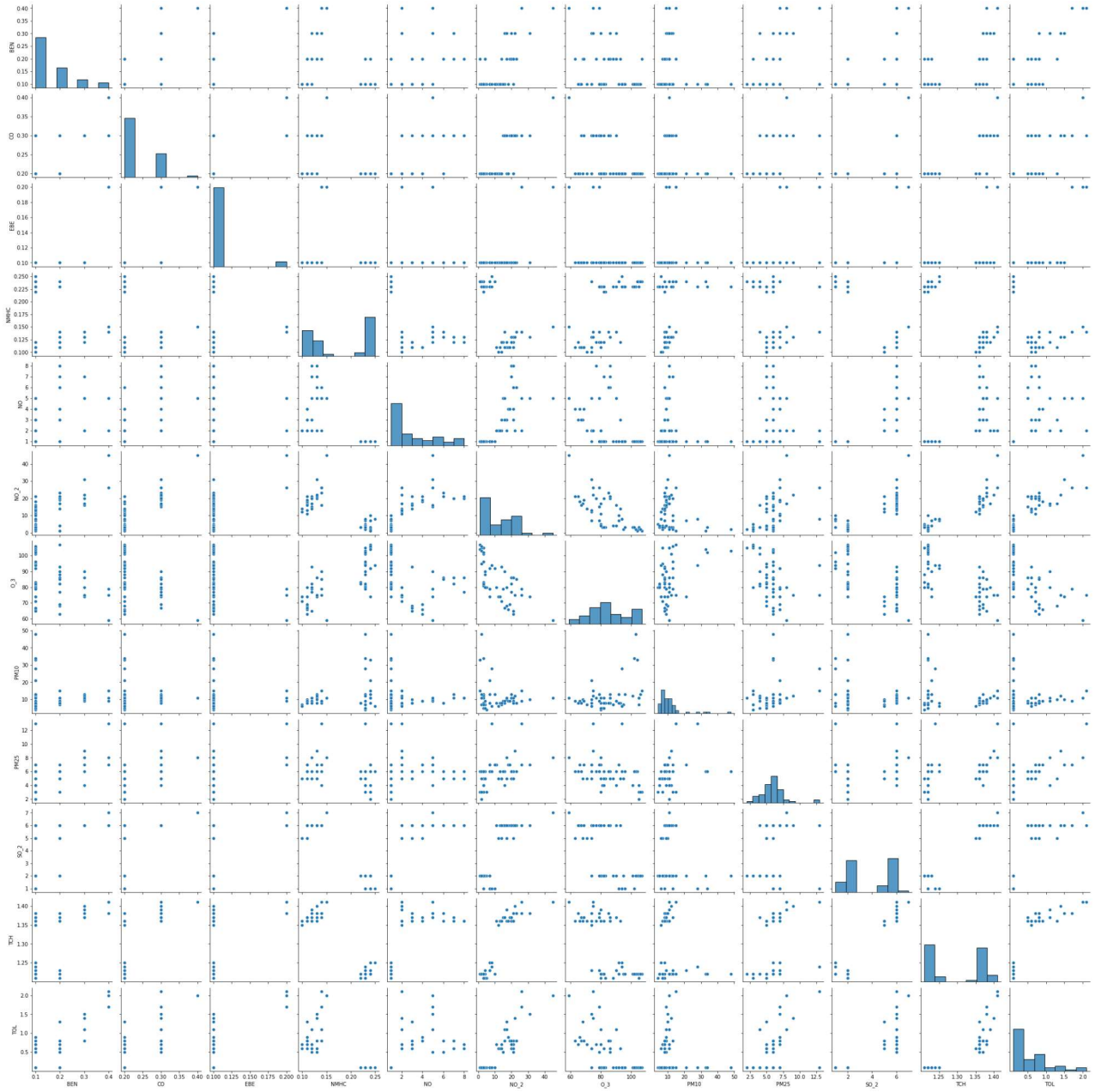
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
25%	0.100000	0.200000	0.100000	0.160000	1.000000	10.000000	25.00000
50%	0.200000	0.300000	0.100000	0.230000	4.000000	27.000000	53.00000
75%	0.400000	0.400000	0.300000	0.260000	18.000000	51.000000	75.00000
max	9.400000	4.400000	16.200001	1.290000	725.000000	346.000000	220.00000

In [19]:
```
df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
```

EDA AND VISUALIZATION

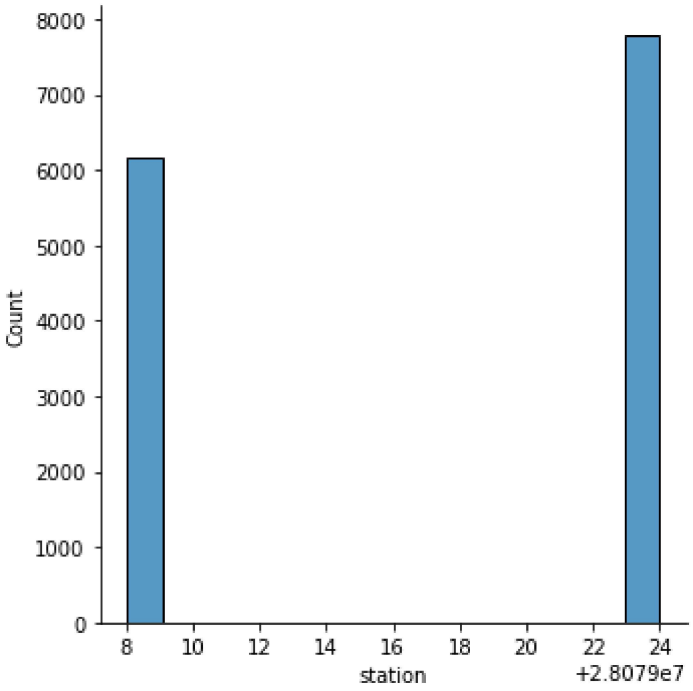
In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x1afbda2e8b0>



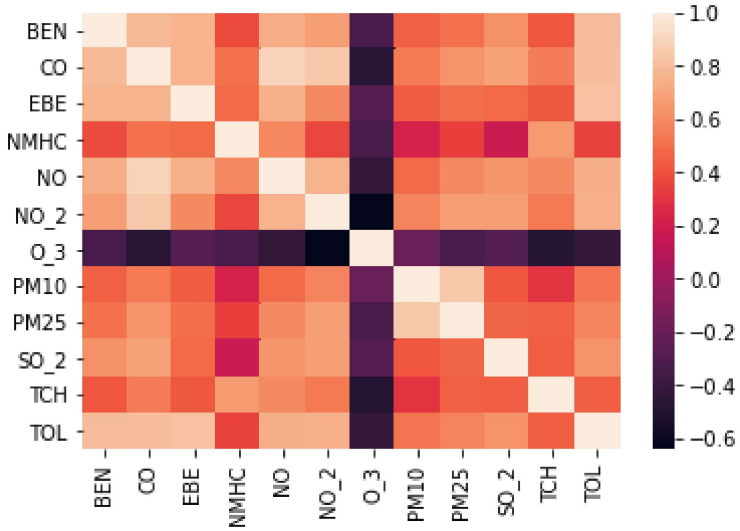
In [21]:
```
sns.displot(df['station'])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x1afbd727fa0>
```



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', 'NO_2', 'O_3', 'PM10', 'PM25',
       '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]:
```
28079024.394915693
```

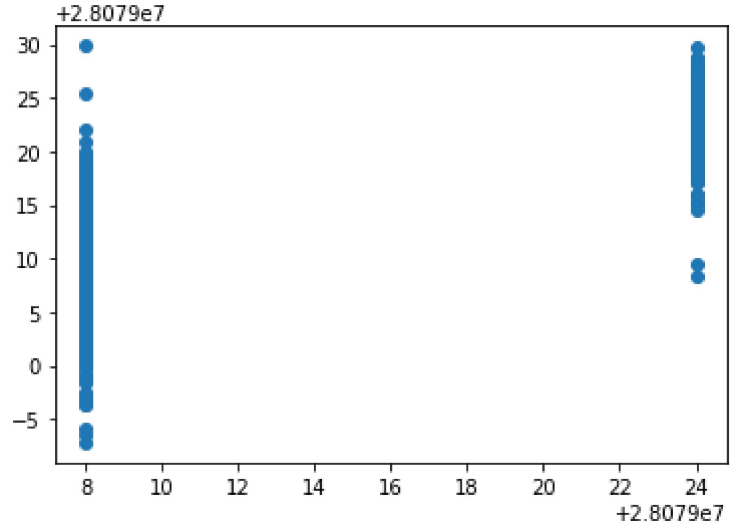
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	-1.273927
CO	-9.781674
EBE	0.067405
NMHC	81.165744
NO	0.027498
NO_2	-0.036824
O_3	0.001595
PM10	-0.022855
PM25	0.133816
SO_2	-0.886446
TCH	-12.469797
TOL	-0.436048

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

Out[28]:
```
<matplotlib.collections.PathCollection at 0x1afc7d86fd0>
```



ACCURACY

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

Out[29]: 0.8919083811903865

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

Out[30]: 0.8905312335955875

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]: 0.8721404715309519

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

Out[34]: 0.8675427616752357

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]:
```
0.29662054811945093
```

Accuracy(Lasso)

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

Out[37]:
```
0.2933629846931395
```

Elastic Net regression

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.        ,  0.10254066,
       -0.10668817, -0.01665885, -0.01175183,  0.10485822, -1.39329952,
        0.        , -0.58435978])
```

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

Out[40]:
```
28079026.63640475
```

In [41]:
```
prediction=en.predict(x_test)
```

In [42]:
```
en.score(x_test,y_test)
```

Out[42]:
```
0.6013151173557179
```

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)))
```

```
4.133590317469261
25.34133925666771
5.034018201860985
```

Logistic Regression

In [44]:
```
from sklearn.linear_model import LogisticRegression
```

In [45]:
```
feature_matrix=df[[ 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

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

Out[46]: (13946, 12)

In [47]:
```
target_vector.shape
```

Out[47]: (13946,)

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=[[1,2,3,4,5,6,7,8,9,10,11,12]]
```

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]:
```
0.9930446006023232
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 1.92542116e-22]])
```

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]:
```
0.9960049170251998
```

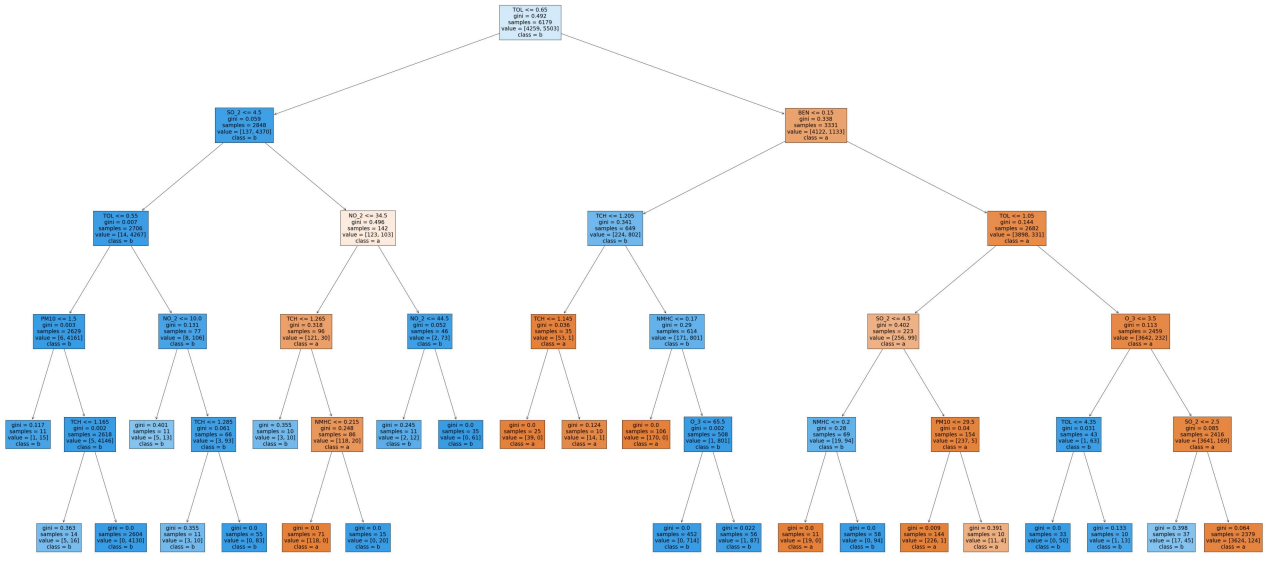
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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','d']
```

Out[63]:
```
[Text(1878.1463414634145, 1993.2, 'TOL <= 0.65\ngini = 0.492\nsamples = 6179\nvalue = [4
259, 5503]\nclass = b'),
 Text(871.0243902439024, 1630.8000000000002, 'SO_2 <= 4.5\ngini = 0.059\nsamples = 2848
\nvalue = [137, 4370]\nclass = b'),
 Text(435.5121951219512, 1268.4, 'TOL <= 0.55\ngini = 0.007\nsamples = 2706\nvalue = [1
4, 4267]\nclass = b'),
 Text(217.7560975609756, 906.0, 'PM10 <= 1.5\ngini = 0.003\nsamples = 2629\nvalue = [6,
4161]\nclass = b'),
 Text(108.8780487804878, 543.5999999999999, 'gini = 0.117\nsamples = 11\nvalue = [1, 15]
\nclass = b'),
 Text(326.6341463414634, 543.5999999999999, 'TCH <= 1.165\ngini = 0.002\nsamples = 2618
\nvalue = [5, 4146]\nclass = b'),
 Text(217.7560975609756, 181.19999999999982, 'gini = 0.363\nsamples = 14\nvalue = [5, 1
6]\nclass = b'),
 Text(435.5121951219512, 181.19999999999982, 'gini = 0.0\nsamples = 2604\nvalue = [0, 41
30]\nclass = b'),
 Text(653.2682926829268, 906.0, 'NO_2 <= 10.0\ngini = 0.131\nsamples = 77\nvalue = [8, 1
06]\nclass = b'),
 Text(544.390243902439, 543.5999999999999, 'gini = 0.401\nsamples = 11\nvalue = [5, 13]
\nclass = b'),
 Text(762.1463414634146, 543.5999999999999, 'TCH <= 1.285\ngini = 0.061\nsamples = 66\nv
alue = [3, 93]\nclass = b'),
 Text(653.2682926829268, 181.19999999999982, 'gini = 0.355\nsamples = 11\nvalue = [3, 1
0]\nclass = b'),
 Text(871.0243902439024, 181.19999999999982, 'gini = 0.0\nsamples = 55\nvalue = [0, 83]
\nclass = b'),
 Text(1306.5365853658536, 1268.4, 'NO_2 <= 34.5\ngini = 0.496\nsamples = 142\nvalue = [1
23, 103]\nclass = a'),
 Text(1088.780487804878, 906.0, 'TCH <= 1.265\ngini = 0.318\nsamples = 96\nvalue = [121,
30]\nclass = a'),
 Text(979.9024390243902, 543.5999999999999, 'gini = 0.355\nsamples = 10\nvalue = [3, 10]
\nclass = b'),
 Text(1197.6585365853657, 543.5999999999999, 'NMHC <= 0.215\ngini = 0.248\nsamples = 86
\nvalue = [118, 20]\nclass = a'),
 Text(1088.780487804878, 181.19999999999982, 'gini = 0.0\nsamples = 71\nvalue = [118, 0]
\nclass = a'),
 Text(1306.5365853658536, 181.19999999999982, 'gini = 0.0\nsamples = 15\nvalue = [0, 20]
\nclass = b'),
 Text(1524.2926829268292, 906.0, 'NO_2 <= 44.5\ngini = 0.052\nsamples = 46\nvalue = [2,
73]\nclass = b'),
 Text(1415.4146341463415, 543.5999999999999, 'gini = 0.245\nsamples = 11\nvalue = [2, 1
2]\nclass = b'),
 Text(1633.170731707317, 543.5999999999999, 'gini = 0.0\nsamples = 35\nvalue = [0, 61]\n
class = b'),
 Text(2885.2682926829266, 1630.8000000000002, 'BEN <= 0.15\ngini = 0.338\nsamples = 3331
\nvalue = [4122, 1133]\nclass = a'),
 Text(2177.560975609756, 1268.4, 'TCH <= 1.205\ngini = 0.341\nsamples = 649\nvalue = [22
4, 802]\nclass = b'),
 Text(1959.8048780487804, 906.0, 'TCH <= 1.145\ngini = 0.036\nsamples = 35\nvalue = [53,
1]\nclass = a'),
 Text(1850.9268292682927, 543.5999999999999, 'gini = 0.0\nsamples = 25\nvalue = [39, 0]
\nclass = a'),
 Text(2068.682926829268, 543.5999999999999, 'gini = 0.124\nsamples = 10\nvalue = [14, 1]
\nclass = a'),
 Text(2395.3170731707314, 906.0, 'NMHC <= 0.17\ngini = 0.29\nsamples = 614\nvalue = [17
1, 801]\nclass = b'),
 Text(2286.439024390244, 543.5999999999999, 'gini = 0.0\nsamples = 106\nvalue = [170, 0]
\nclass = a'),
 Text(2504.1951219512193, 543.5999999999999, 'O_3 <= 65.5\ngini = 0.002\nsamples = 508\n
value = [1, 801]\nclass = b'),
 Text(2395.3170731707314, 181.19999999999982, 'gini = 0.0\nsamples = 452\nvalue = [0, 71
4]\nclass = b'),
 Text(2613.0731707317073, 181.19999999999982, 'gini = 0.022\nsamples = 56\nvalue = [1, 8
7]\nclass = b'),
 Text(3592.9756097560976, 1268.4, 'TOL <= 1.05\ngini = 0.144\nsamples = 2682\nvalue = [3
```

```
898, 331]\nclass = a'),
 Text(3157.4634146341464, 906.0, 'SO_2 <= 4.5\ngini = 0.402\nsamples = 223\nvalue = [25
6, 99]\nclass = a'),
 Text(2939.7073170731705, 543.5999999999999, 'NMHC <= 0.2\ngini = 0.28\nsamples = 69\nva
lue = [19, 94]\nclass = b'),
 Text(2830.829268292683, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [19, 0]
\nclass = a'),
 Text(3048.5853658536585, 181.19999999999982, 'gini = 0.0\nsamples = 58\nvalue = [0, 94]
\nclass = b'),
 Text(3375.2195121951218, 543.5999999999999, 'PM10 <= 29.5\ngini = 0.04\nsamples = 154\n
value = [237, 5]\nclass = a'),
 Text(3266.341463414634, 181.19999999999982, 'gini = 0.009\nsamples = 144\nvalue = [226,
1]\nclass = a'),
 Text(3484.0975609756097, 181.19999999999982, 'gini = 0.391\nsamples = 10\nvalue = [11,
4]\nclass = a'),
 Text(4028.487804878049, 906.0, 'O_3 <= 3.5\ngini = 0.113\nsamples = 2459\nvalue = [364
2, 232]\nclass = a'),
 Text(3810.731707317073, 543.5999999999999, 'TOL <= 4.35\ngini = 0.031\nsamples = 43\nva
lue = [1, 63]\nclass = b'),
 Text(3701.8536585365855, 181.19999999999982, 'gini = 0.0\nsamples = 33\nvalue = [0, 50]
\nclass = b'),
 Text(3919.609756097561, 181.19999999999982, 'gini = 0.133\nsamples = 10\nvalue = [1, 1
3]\nclass = b'),
 Text(4246.243902439024, 543.5999999999999, 'SO_2 <= 2.5\ngini = 0.085\nsamples = 2416\n
value = [3641, 169]\nclass = a'),
 Text(4137.365853658536, 181.19999999999982, 'gini = 0.398\nsamples = 37\nvalue = [17, 4
5]\nclass = b'),
 Text(4355.121951219512, 181.19999999999982, 'gini = 0.064\nsamples = 2379\nvalue = [362
4, 124]\nclass = a')]
```



Conclusion

Accuracy

linear regression

In [64]:

```
lr.score(x_test,y_test)
```

Out[64]: 0.8919083811903865

Ridge regression

In [65]:
```
rr.score(x_test,y_test)
```

Out[65]: 0.8721404715309519

Lasso regression

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

Out[66]: 0.2933629846931395

Elastic net regression

In [67]:
```
en.score(x_test,y_test)
```

Out[67]: 0.6013151173557179

Logistic regression

In [68]:
```
logr.score(fs,target_vector)
```

Out[68]: 0.9930446006023232

Random forest

In [69]:
```
grid_search.best_score_
```

Out[69]: 0.9960049170251998

Accuracy for random forest is higher so it is the best fit model