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_2017.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TOL
0	01-06-2017 01:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN
1	01-06-2017 01:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1.4	2.9
2	01-06-2017 01:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	0.9
3	01-06-2017 01:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	NaN	NaN
4	01-06-2017 01:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210115	01-08-2017 00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	NaN	NaN
210116	01-08-2017 00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	NaN	NaN
210117	01-08-2017 00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	NaN	NaN

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TOL
210118	01-08-2017 00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	NaN	NaN
210119	01-08-2017 00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	NaN	NaN

210120 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: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

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

Out[6]:

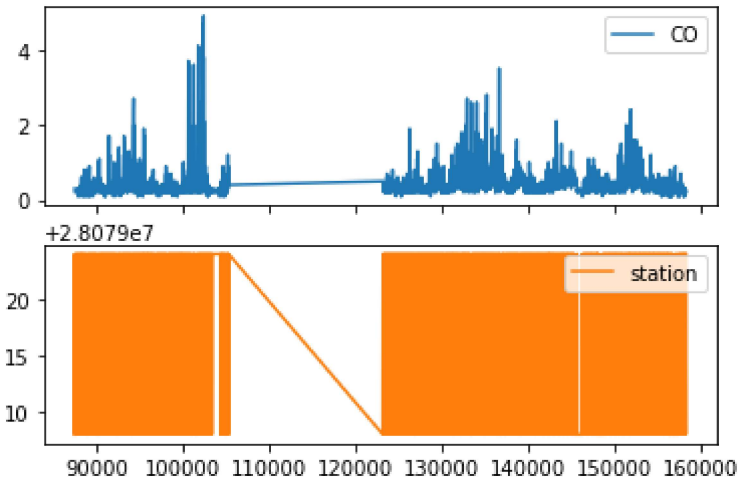
	CO	station
87457	0.3	28079008
87462	0.2	28079024
87481	0.2	28079008
87486	0.2	28079024
87505	0.2	28079008
...	...	...
158238	0.2	28079024
158257	0.3	28079008
158262	0.2	28079024
158281	0.2	28079008
158286	0.2	28079024

4127 rows × 2 columns

Line chart

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

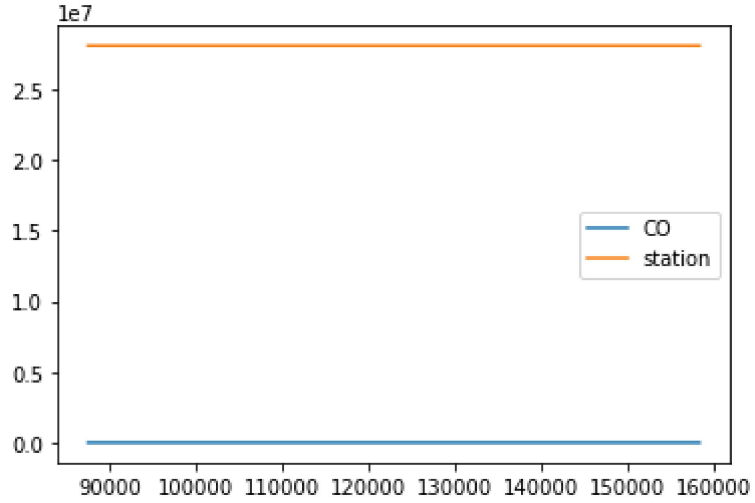
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

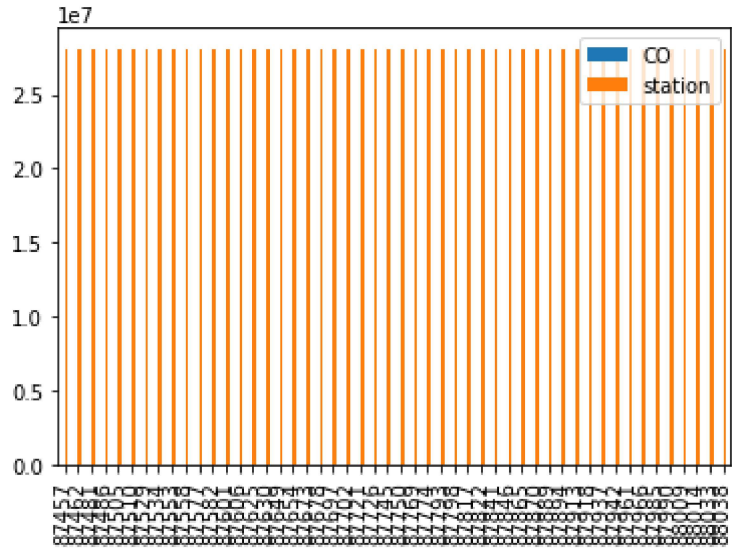


Bar chart

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

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

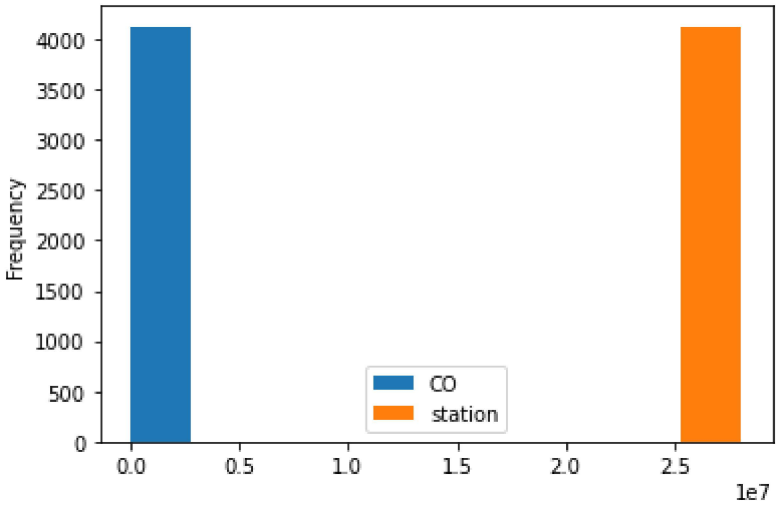
Out[10]: <AxesSubplot:>



Histogram

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

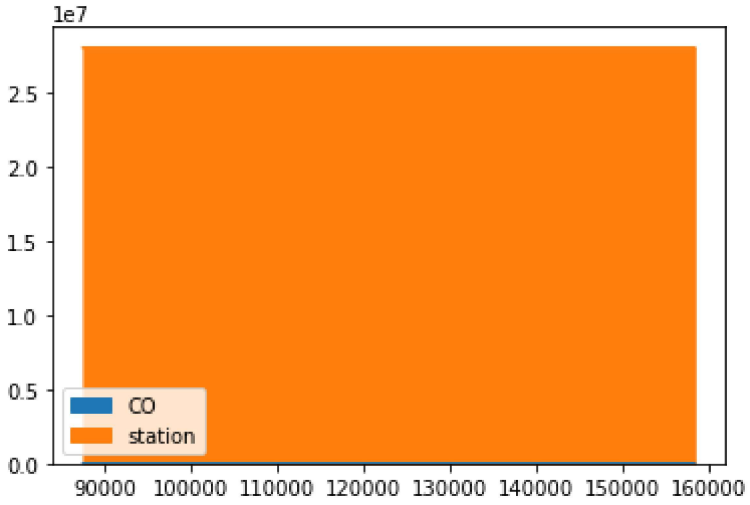
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

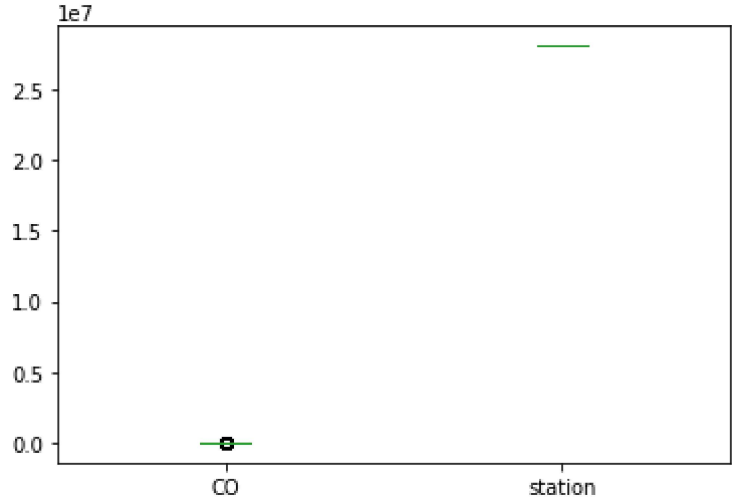
Out[12]: <AxesSubplot:>



Box chart

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

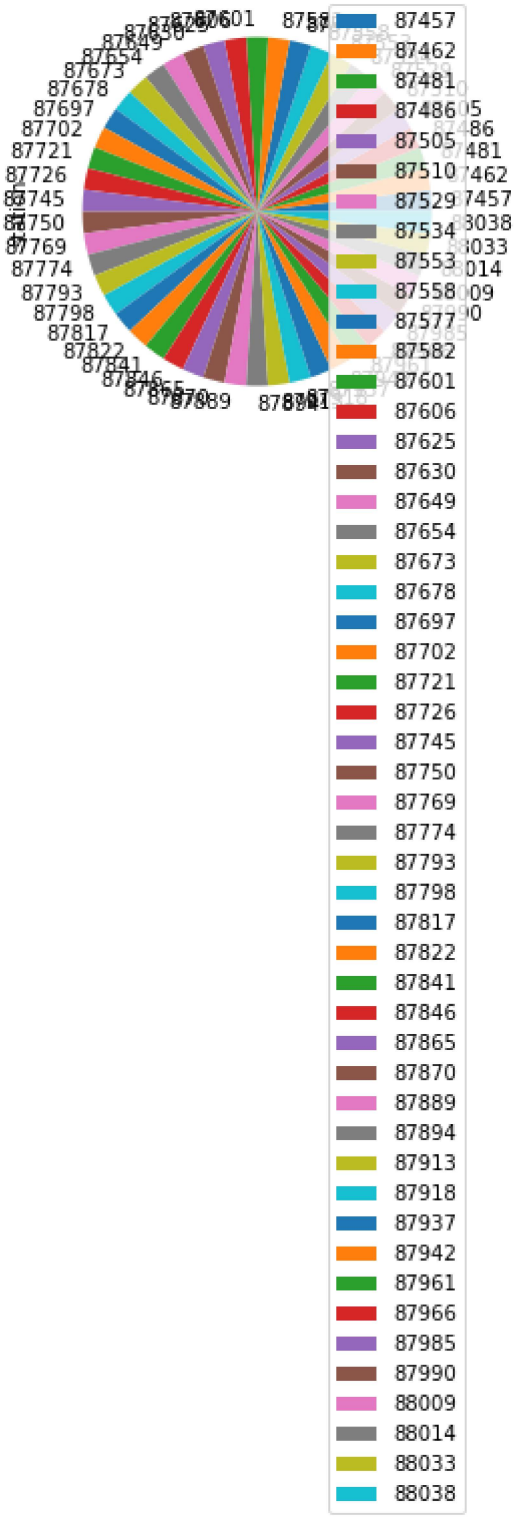
Out[13]: <AxesSubplot:>



Pie chart

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

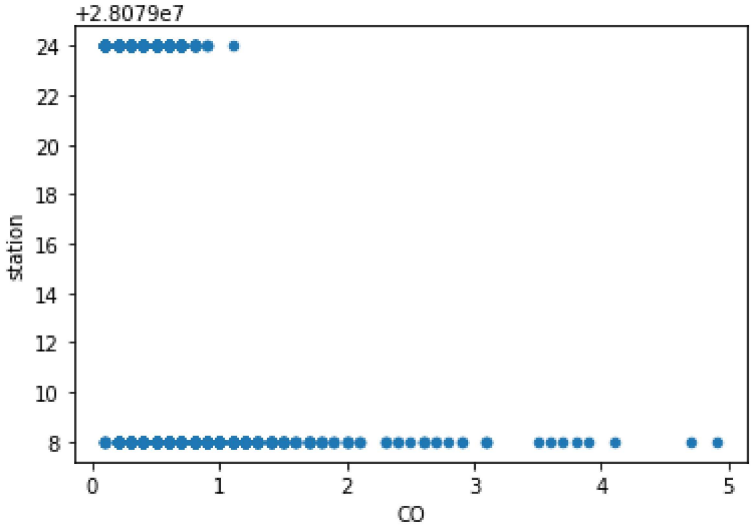
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ 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	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.
mean	0.919918	1.323732	0.417858	0.578168	0.097269	41.785316	58.069057	122.
std	1.123078	0.215742	0.342871	0.962000	0.094035	71.118499	38.974112	142.

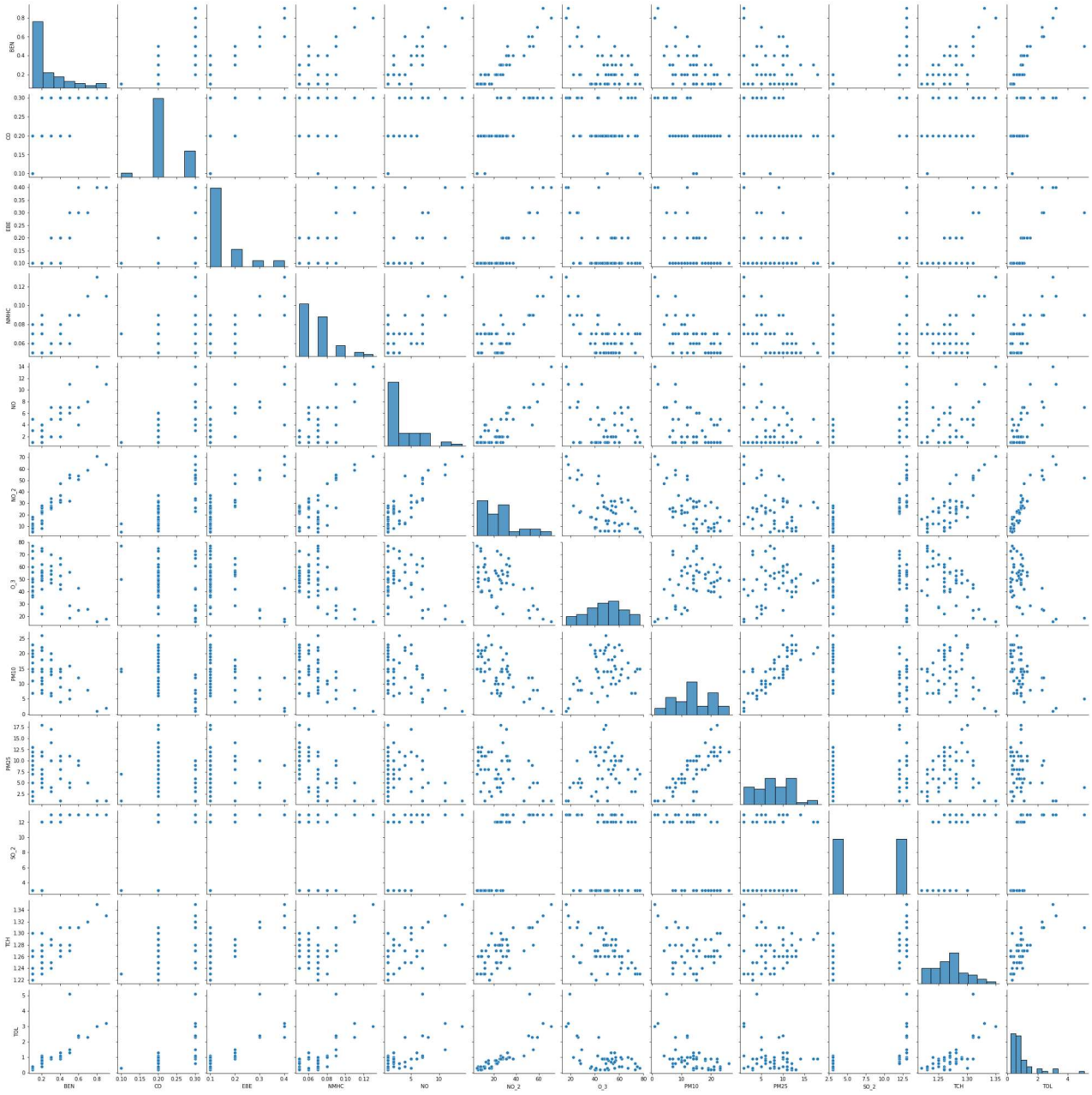
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
min	0.100000	1.100000	0.100000	0.100000	0.000000	1.000000	1.000000	2.
25%	0.300000	1.180000	0.200000	0.100000	0.050000	3.000000	30.000000	37.
50%	0.600000	1.270000	0.300000	0.300000	0.080000	16.000000	54.000000	80.
75%	1.100000	1.400000	0.500000	0.700000	0.110000	50.000000	78.000000	153.
max	19.600000	3.630000	4.900000	16.700001	1.420000	879.000000	349.000000	1681.

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

EDA AND VISUALIZATION

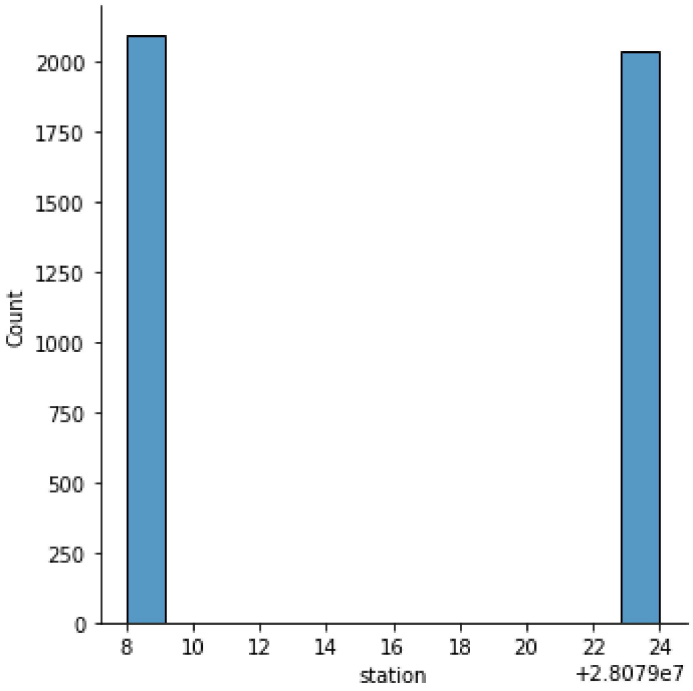
In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x182637371f0>



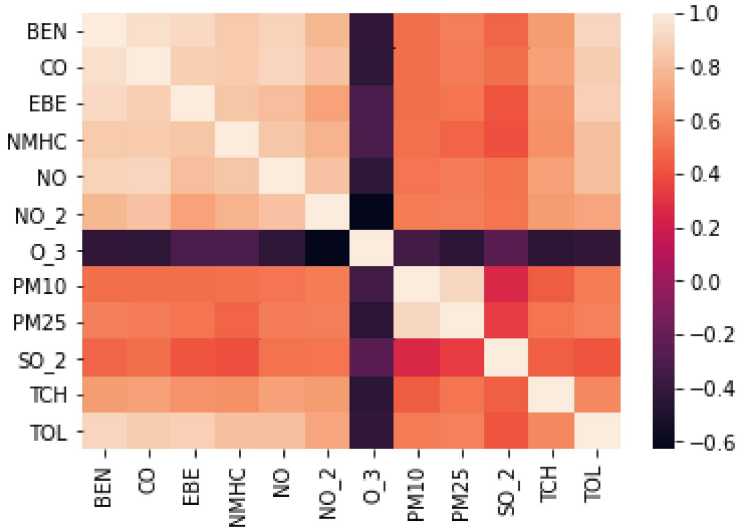
In [21]:
```
sns.displot(df['station'])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x1826aa28940>
```



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]:
```
28079043.14435622
```

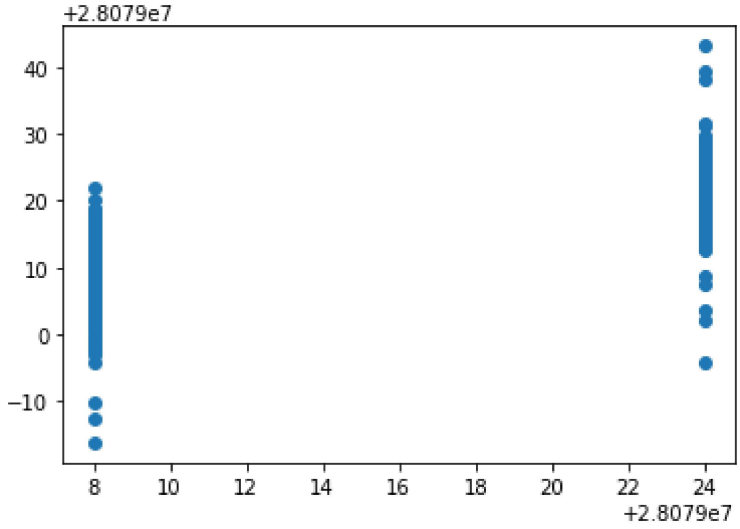
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	0.204797
CO	-4.850359
EBE	-2.554834
NMHC	27.052798
NO	0.052550
NO_2	-0.185640
O_3	-0.095932
PM10	0.427617
PM25	-0.157428
SO_2	-0.257152
TCH	-14.571244
TOL	0.244562

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

Out[28]:
```
<matplotlib.collections.PathCollection at 0x1826da6a730>
```



ACCURACY

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

Out[29]: 0.6277708455635972

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

Out[30]: 0.6376027726071255

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]: 0.6220028733584428

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

Out[34]: 0.62797708207299

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]:
```
0.40599824183762034
```

Accuracy(Lasso)

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

Out[37]:
```
0.4080539385511591
```

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.        ,  0.0304487 ,
       -0.2069857 , -0.09218015,  0.5484427 , -0.39389361, -0.27727376,
       -0.        ,  0.        ])
```

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

Out[40]:
```
28079025.433698453
```

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

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

Out[42]:
```
0.5096944365072741
```

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)))
```

```
4.7835291357075045
31.37217682780064
5.601087111249087
```

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]: (4127, 12)

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

Out[47]: (4127,)

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]:
```
0.9520232614489944
```

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

Out[55]:
```
0.9999999999999389
```

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

Out[56]:
```
array([[1.00000000e+00, 6.10312368e-14]])
```

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]:
```
0.9705678670360111
```

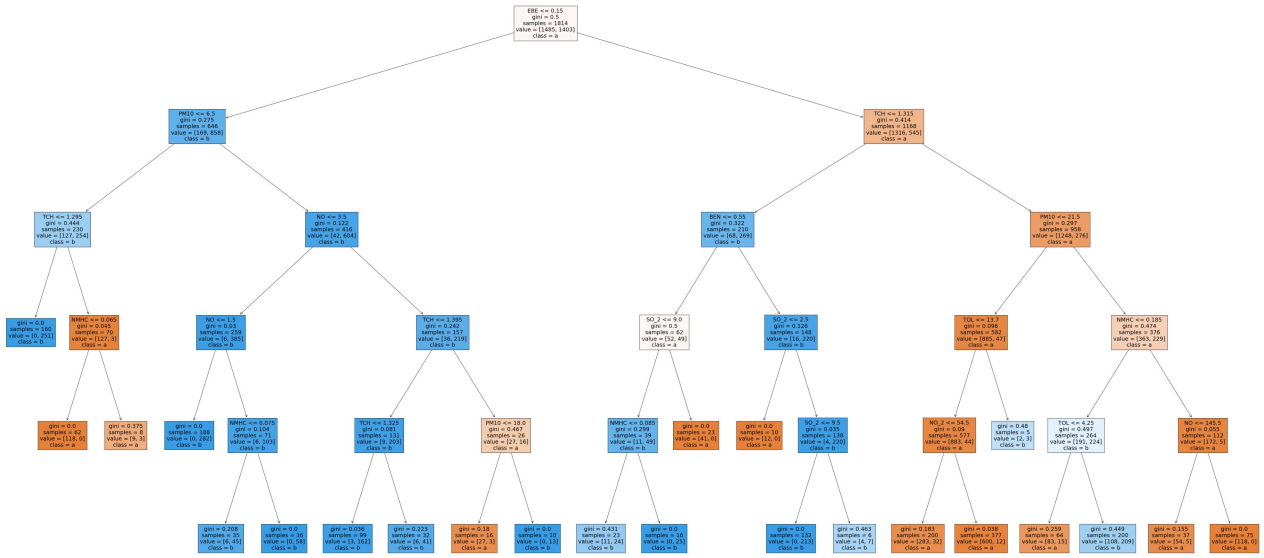
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1925.1, 1993.2, 'EBE <= 0.15\ngini = 0.5\nsamples = 1814\nvalue = [1485, 1403]\ncl
ass = a'),
 Text(697.5, 1630.8000000000002, 'PM10 <= 6.5\ngini = 0.275\nsamples = 646\nvalue = [16
9, 858]\nclass = b'),
 Text(223.2, 1268.4, 'TCH <= 1.295\ngini = 0.444\nsamples = 230\nvalue = [127, 254]\ncla
ss = b'),
 Text(111.6, 906.0, 'gini = 0.0\nsamples = 160\nvalue = [0, 251]\nclass = b'),
 Text(334.79999999999995, 906.0, 'NMHC <= 0.065\ngini = 0.045\nsamples = 70\nvalue = [12
7, 3]\nclass = a'),
 Text(223.2, 543.5999999999999, 'gini = 0.0\nsamples = 62\nvalue = [118, 0]\nclass =
a'),
 Text(446.4, 543.5999999999999, 'gini = 0.375\nsamples = 8\nvalue = [9, 3]\nclass = a'),
 Text(1171.8, 1268.4, 'NO <= 3.5\ngini = 0.122\nsamples = 416\nvalue = [42, 604]\nclass
= b'),
 Text(781.1999999999999, 906.0, 'NO <= 1.5\ngini = 0.03\nsamples = 259\nvalue = [6, 385]
\nclass = b'),
 Text(669.5999999999999, 543.5999999999999, 'gini = 0.0\nsamples = 188\nvalue = [0, 282]
\nclass = b'),
 Text(892.8, 543.5999999999999, 'NMHC <= 0.075\ngini = 0.104\nsamples = 71\nvalue = [6,
103]\nclass = b'),
 Text(781.1999999999999, 181.19999999999982, 'gini = 0.208\nsamples = 35\nvalue = [6, 4
5]\nclass = b'),
 Text(1004.4, 181.19999999999982, 'gini = 0.0\nsamples = 36\nvalue = [0, 58]\nclass =
b'),
 Text(1562.3999999999999, 906.0, 'TCH <= 1.395\ngini = 0.242\nsamples = 157\nvalue = [3
6, 219]\nclass = b'),
 Text(1339.1999999999998, 543.5999999999999, 'TCH <= 1.325\ngini = 0.081\nsamples = 131
\nvalue = [9, 203]\nclass = b'),
 Text(1227.6, 181.19999999999982, 'gini = 0.036\nsamples = 99\nvalue = [3, 162]\nclass =
b'),
 Text(1450.8, 181.19999999999982, 'gini = 0.223\nsamples = 32\nvalue = [6, 41]\nclass =
b'),
 Text(1785.6, 543.5999999999999, 'PM10 <= 18.0\ngini = 0.467\nsamples = 26\nvalue = [27,
16]\nclass = a'),
 Text(1674.0, 181.19999999999982, 'gini = 0.18\nsamples = 16\nvalue = [27, 3]\nclass =
a'),
 Text(1897.1999999999998, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalue = [0, 13]
\nclass = b'),
 Text(3152.7, 1630.8000000000002, 'TCH <= 1.315\ngini = 0.414\nsamples = 1168\nvalue =
[1316, 545]\nclass = a'),
 Text(2566.7999999999997, 1268.4, 'BEN <= 0.55\ngini = 0.322\nsamples = 210\nvalue = [6
8, 269]\nclass = b'),
 Text(2343.6, 906.0, 'SO_2 <= 9.0\ngini = 0.5\nsamples = 62\nvalue = [52, 49]\nclass =
a'),
 Text(2232.0, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.299\nsamples = 39\nvalue = [1
1, 49]\nclass = b'),
 Text(2120.4, 181.19999999999982, 'gini = 0.431\nsamples = 23\nvalue = [11, 24]\nclass =
b'),
 Text(2343.6, 181.19999999999982, 'gini = 0.0\nsamples = 16\nvalue = [0, 25]\nclass =
b'),
 Text(2455.2, 543.5999999999999, 'gini = 0.0\nsamples = 23\nvalue = [41, 0]\nclass =
a'),
 Text(2790.0, 906.0, 'SO_2 <= 2.5\ngini = 0.126\nsamples = 148\nvalue = [16, 220]\nclass
= b'),
 Text(2678.3999999999996, 543.5999999999999, 'gini = 0.0\nsamples = 10\nvalue = [12, 0]
\nclass = a'),
 Text(2901.6, 543.5999999999999, 'SO_2 <= 9.5\ngini = 0.035\nsamples = 138\nvalue = [4,
220]\nclass = b'),
 Text(2790.0, 181.19999999999982, 'gini = 0.0\nsamples = 132\nvalue = [0, 213]\nclass =
b'),
 Text(3013.2, 181.19999999999982, 'gini = 0.463\nsamples = 6\nvalue = [4, 7]\nclass =
b'),
 Text(3738.6, 1268.4, 'PM10 <= 21.5\ngini = 0.297\nsamples = 958\nvalue = [1248, 276]\nc
lass = a'),
 Text(3459.6, 906.0, 'TOL <= 13.7\ngini = 0.096\nsamples = 582\nvalue = [885, 47]\nclass
```

```
= a'),
 Text(3348.0, 543.5999999999999, 'NO_2 <= 54.5\ngini = 0.09\nsamples = 577\nvalue = [88
3, 44]\nclass = a'),
 Text(3236.3999999999996, 181.19999999999982, 'gini = 0.183\nsamples = 200\nvalue = [28
3, 32]\nclass = a'),
 Text(3459.6, 181.19999999999982, 'gini = 0.038\nsamples = 377\nvalue = [600, 12]\nclass
= a'),
 Text(3571.2, 543.5999999999999, 'gini = 0.48\nsamples = 5\nvalue = [2, 3]\nclass = b'),
 Text(4017.6, 906.0, 'NMHC <= 0.185\ngini = 0.474\nsamples = 376\nvalue = [363, 229]\ncl
ass = a'),
 Text(3794.3999999999996, 543.5999999999999, 'TOL <= 4.25\ngini = 0.497\nsamples = 264\n
value = [191, 224]\nclass = b'),
 Text(3682.7999999999997, 181.19999999999982, 'gini = 0.259\nsamples = 64\nvalue = [83,
15]\nclass = a'),
 Text(3906.0, 181.19999999999982, 'gini = 0.449\nsamples = 200\nvalue = [108, 209]\nclas
s = b'),
 Text(4240.8, 543.5999999999999, 'NO <= 145.5\ngini = 0.055\nsamples = 112\nvalue = [17
2, 5]\nclass = a'),
 Text(4129.2, 181.19999999999982, 'gini = 0.155\nsamples = 37\nvalue = [54, 5]\nclass =
a'),
 Text(4352.4, 181.19999999999982, 'gini = 0.0\nsamples = 75\nvalue = [118, 0]\nclass =
a')]
```



Conclusion

Accuracy

linear regression

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

Out[64]:
```
0.6277708455635972
```

Ridge regression

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

Out[65]: 0.6220028733584428

Lasso regression

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

Out[66]: 0.4080539385511591

Elastic net regression

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

Out[67]: 0.5096944365072741

Logistic regression

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

Out[68]: 0.9520232614489944

Random forest

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

Out[69]: 0.9705678670360111

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