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_2016.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	01-11-2016 01:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	28079004
1	01-11-2016 01:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	28079008
2	01-11-2016 01:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	28079011
3	01-11-2016 01:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	01-11-2016 01:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209491	01-07-2016 00:00	NaN	0.2	NaN	NaN	2.0	29.0	73.0	NaN	NaN	NaN	NaN	NaN	28079056
209492	01-07-2016 00:00	NaN	0.3	NaN	NaN	1.0	29.0	NaN	36.0	NaN	5.0	NaN	NaN	28079057
209493	01-07-2016 00:00	NaN	NaN	NaN	NaN	1.0	19.0	71.0	NaN	NaN	NaN	NaN	NaN	28079058

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
209494	01-07-2016 00:00	NaN	NaN	NaN	NaN	6.0	17.0	85.0	NaN	NaN	NaN	NaN	NaN	28079059
209495	01-07-2016 00:00	NaN	NaN	NaN	NaN	2.0	46.0	61.0	34.0	NaN	NaN	NaN	NaN	28079060

209496 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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[6]:

	CO	station
1	1.1	28079008

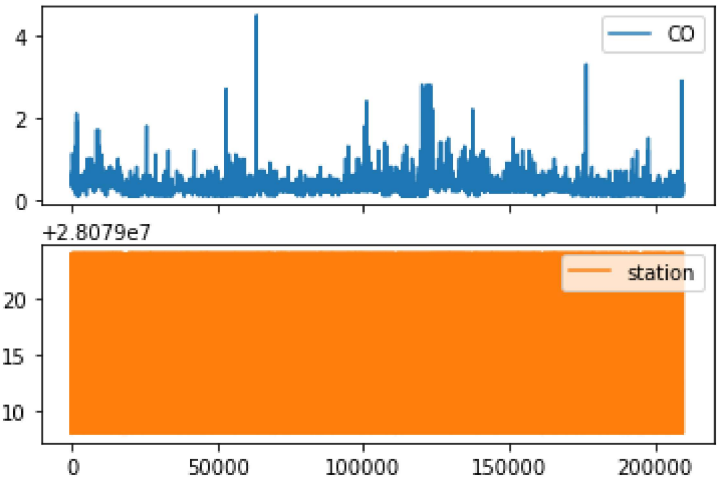
	CO	station
6	0.8	28079024
25	1.0	28079008
30	0.7	28079024
49	0.8	28079008
...	...	...
209430	0.2	28079024
209449	0.4	28079008
209454	0.2	28079024
209473	0.4	28079008
209478	0.2	28079024

16932 rows × 2 columns

Line chart

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

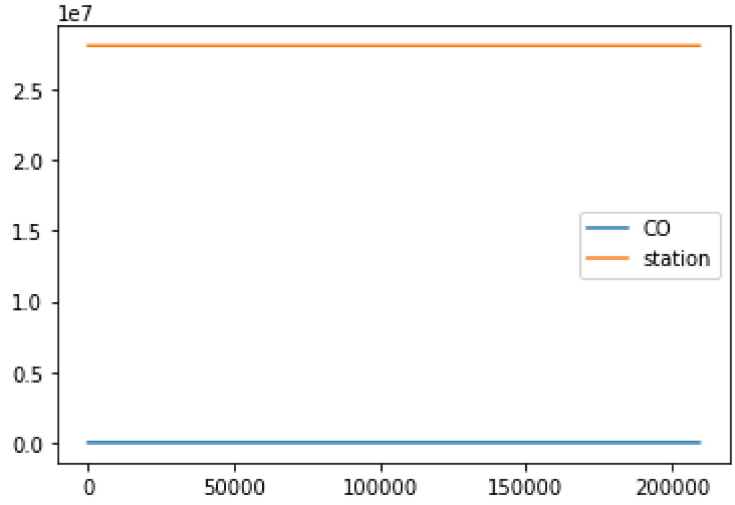
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

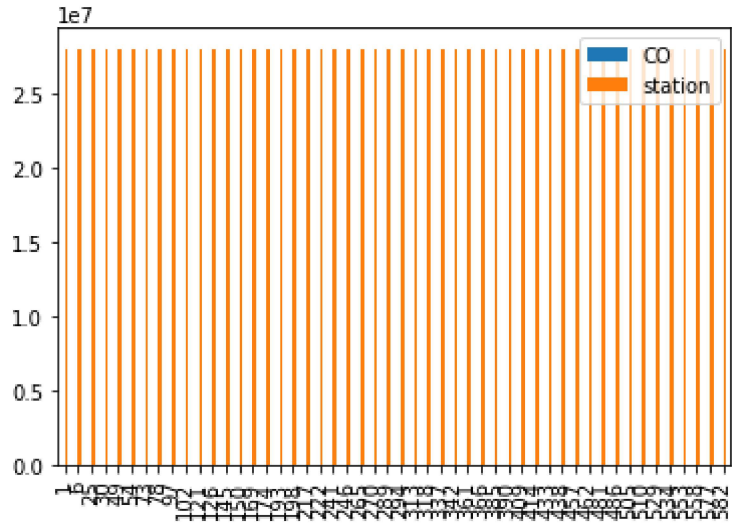


Bar chart

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

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

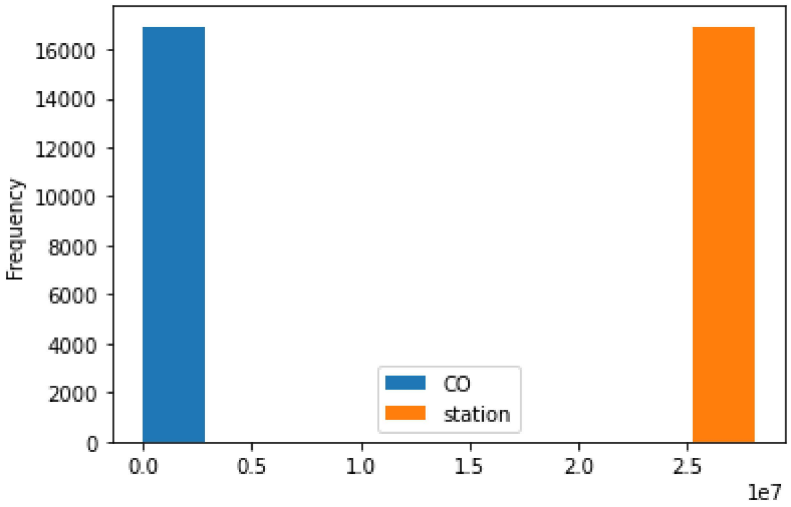
Out[10]: <AxesSubplot:>



Histogram

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

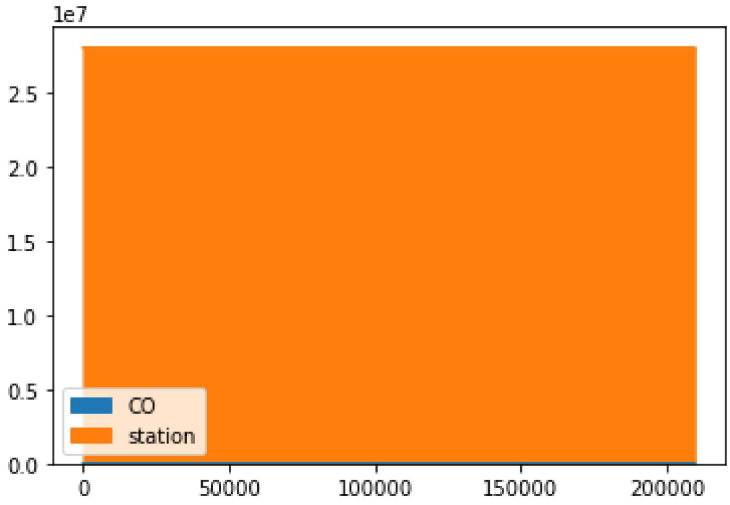
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

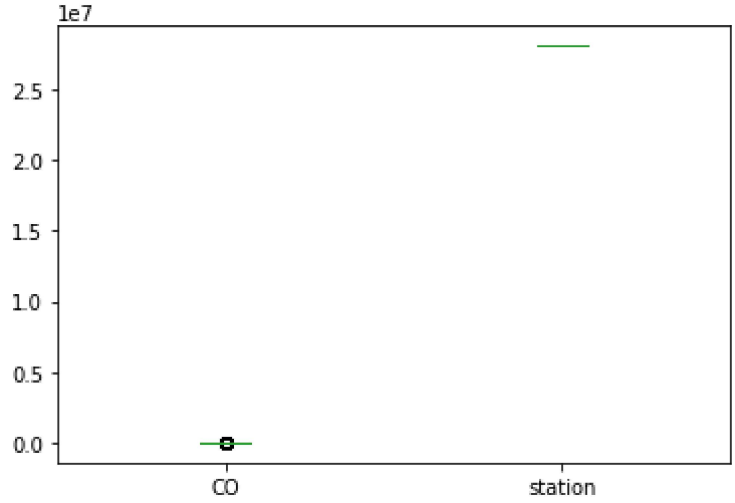
Out[12]: <AxesSubplot:>



Box chart

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

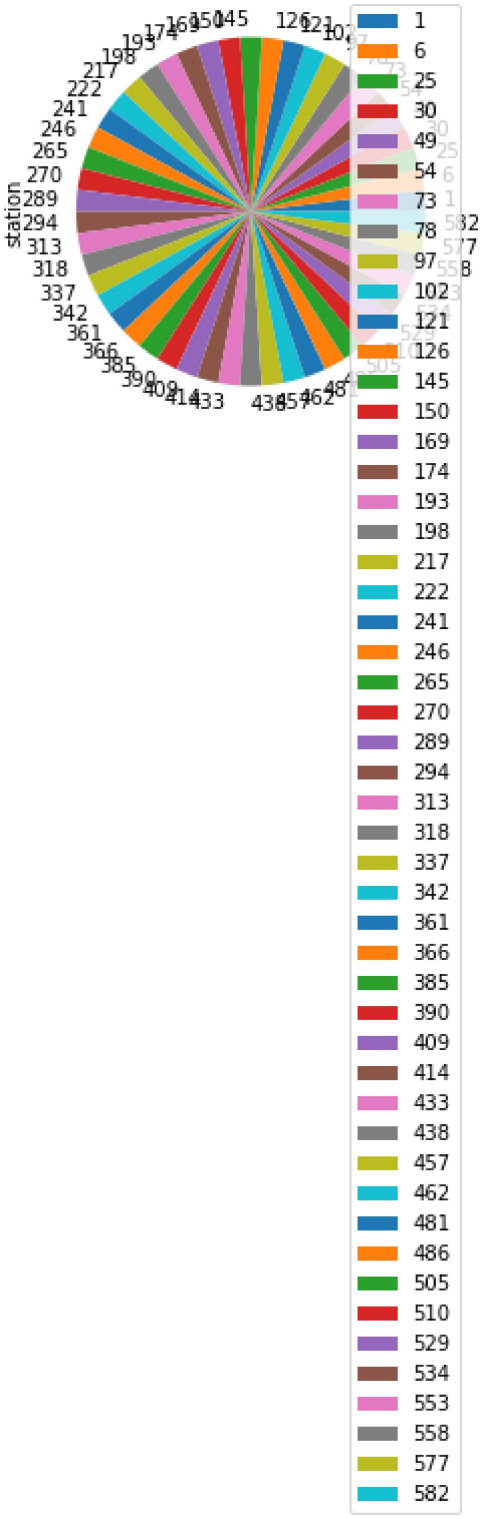
Out[13]: <AxesSubplot:>



Pie chart

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

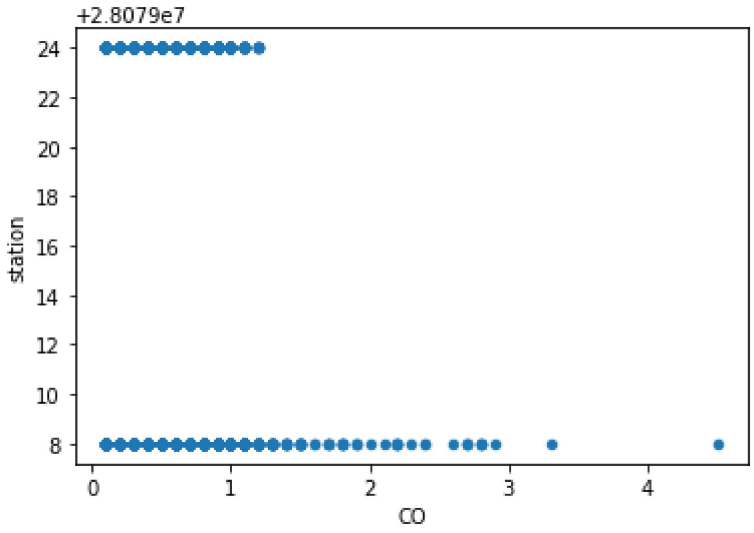
Out[14]: <AxesSubplot:ylabel='station'>



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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ 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	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000
mean	0.537970	0.349941	0.298955	0.099913	20.815734	39.373376	48.118474
std	0.599479	0.203807	0.450204	0.079850	40.986063	31.170307	32.560271
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000

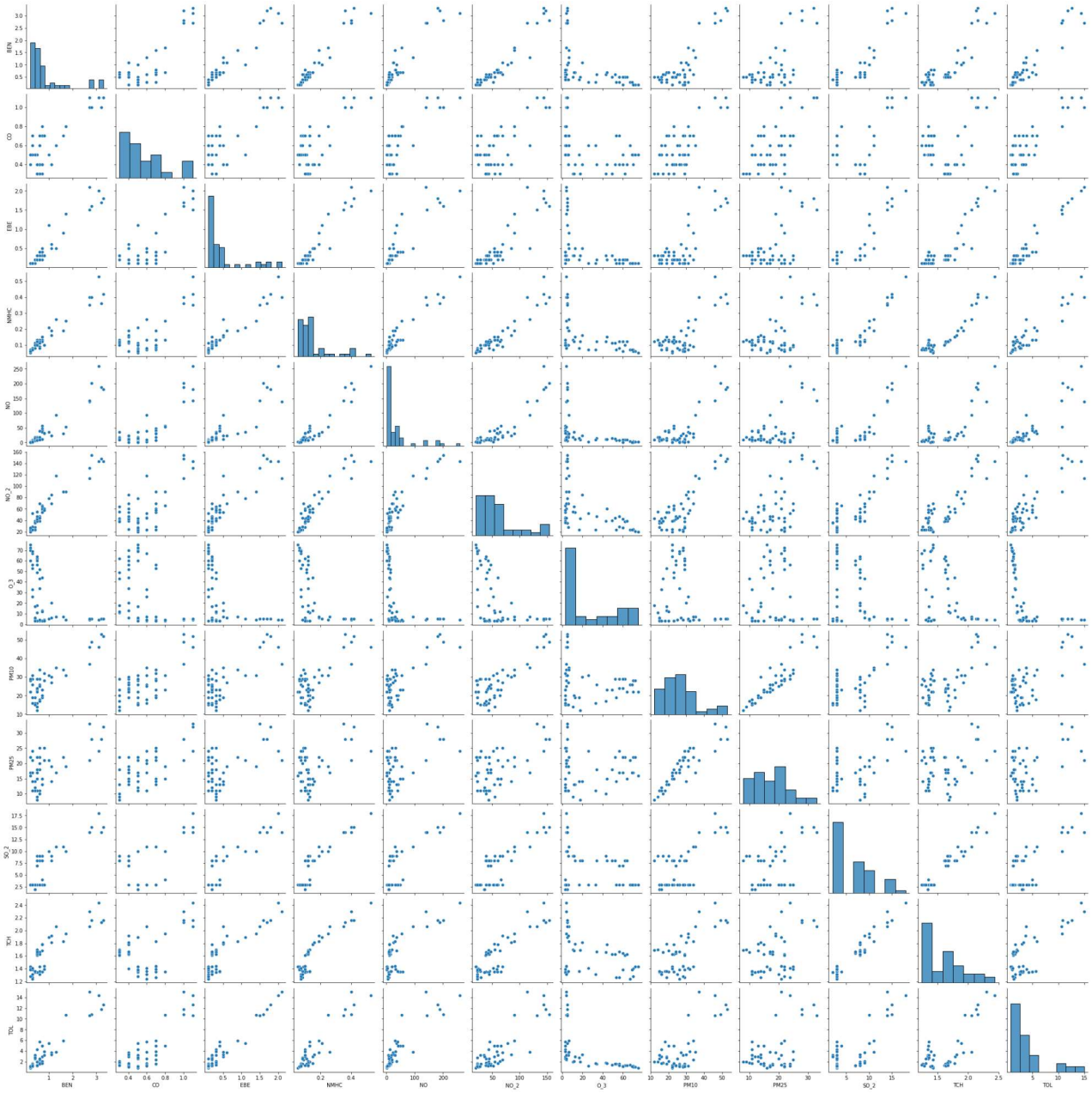
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
25%	0.200000	0.200000	0.100000	0.050000	1.000000	14.000000	21.00000
50%	0.400000	0.300000	0.200000	0.090000	7.000000	34.000000	46.00000
75%	0.700000	0.400000	0.300000	0.120000	23.000000	58.000000	69.00000
max	12.300000	4.500000	13.500000	2.210000	829.000000	319.000000	181.00000

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

EDA AND VISUALIZATION

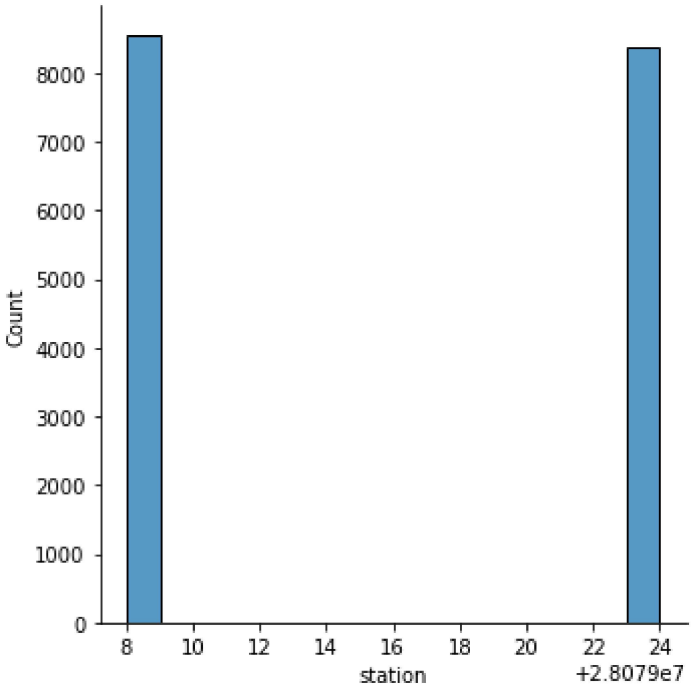
In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x20b68dfbdc0>



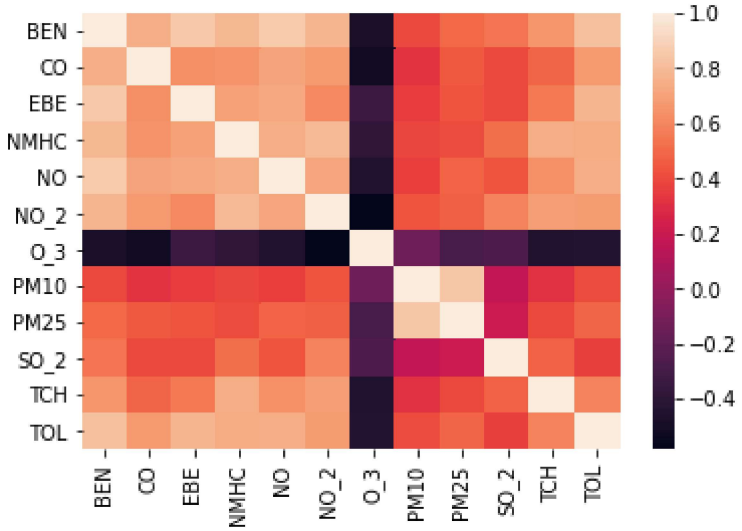
In [21]:
```
sns.displot(df['station'])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x20b6c1e71c0>
```



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]:
```
28079042.527170584
```

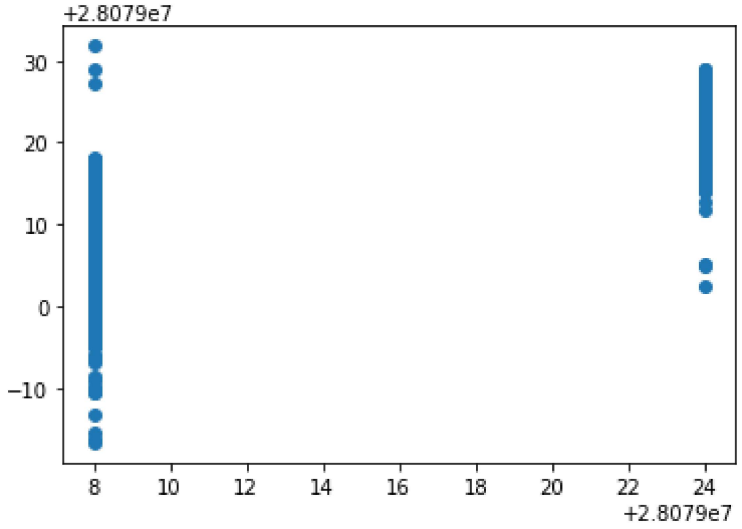
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	-2.017760
CO	3.858276
EBE	0.794343
NMHC	2.085569
NO	0.069951
NO_2	-0.065068
O_3	-0.024533
PM10	-0.013745
PM25	0.118355
SO_2	-0.802797
TCH	-14.410384
TOL	0.165420

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

Out[28]:
```
<matplotlib.collections.PathCollection at 0x20b732402e0>
```



ACCURACY

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

Out[29]: 0.8285268680585672

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

Out[30]: 0.8276013542854708

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]: 0.8283861171129381

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

Out[34]: 0.8274982513959772

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]:
```
0.6464852947292641
```

Accuracy(Lasso)

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

Out[37]:
```
0.6453921030812604
```

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.        ,  0.04728043,
       -0.10790083, -0.02048809,  0.00424075,  0.05473435, -0.86435745,
       -0.00605905,  0.        ])
```

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

Out[40]:
```
28079026.11302535
```

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

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

Out[42]:
```
0.7040306356663145
```

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)))
```

```
3.3506878408002043
18.934989928614947
4.351435387158466
```

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]: (16932, 12)

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

Out[47]: (16932,)

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]:
```
0.996161115048429
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 1.38109298e-55]])
```

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]:
```
0.9938407019912251
```

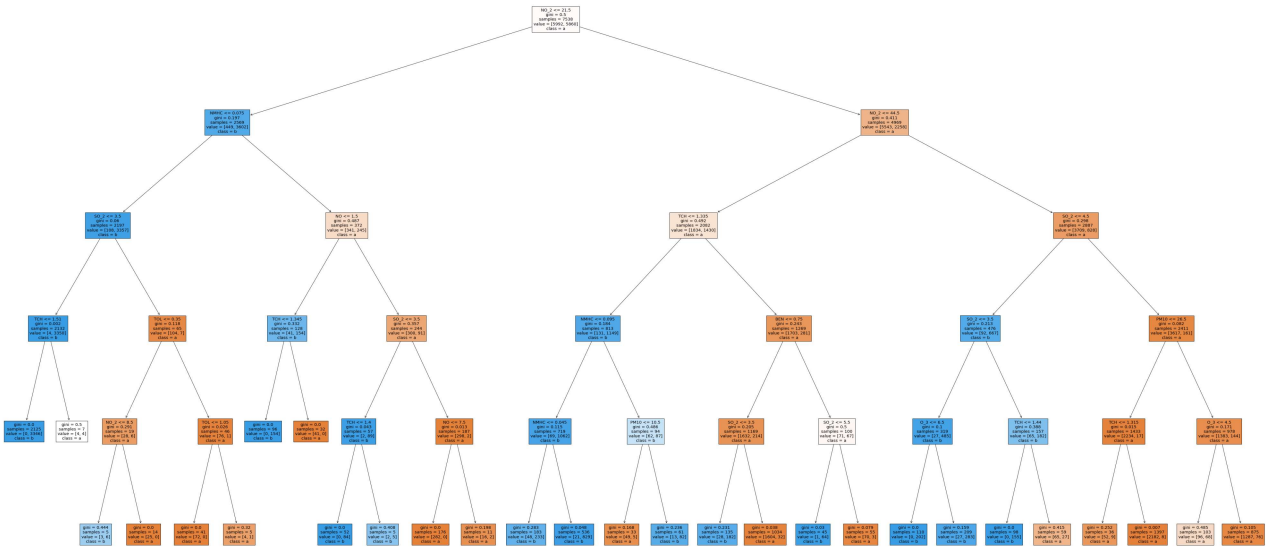
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1958.2641509433963, 1993.2, 'NO_2 <= 21.5\ngini = 0.5\nsamples = 7538\nvalue = [5992, 5860]\nclass = a'),
 Text(800.1509433962265, 1630.8000000000002, 'NMHC <= 0.075\ngini = 0.197\nsamples = 2569\nvalue = [449, 3602]\nclass = b'),
 Text(379.0188679245283, 1268.4, 'SO_2 <= 3.5\ngini = 0.06\nsamples = 2197\nvalue = [108, 3357]\nclass = b'),
 Text(168.45283018867926, 906.0, 'TCH <= 1.51\ngini = 0.002\nsamples = 2132\nvalue = [4, 3350]\nclass = b'),
 Text(84.22641509433963, 543.5999999999999, 'gini = 0.0\nsamples = 2125\nvalue = [0, 3346]\nclass = b'),
 Text(252.67924528301887, 543.5999999999999, 'gini = 0.5\nsamples = 7\nvalue = [4, 4]\nclass = a'),
 Text(589.5849056603774, 906.0, 'TOL <= 0.35\ngini = 0.118\nsamples = 65\nvalue = [104, 7]\nclass = a'),
 Text(421.13207547169816, 543.5999999999999, 'NO_2 <= 8.5\ngini = 0.291\nsamples = 19\nvalue = [28, 6]\nclass = a'),
 Text(336.9056603773585, 181.19999999999982, 'gini = 0.444\nsamples = 5\nvalue = [3, 6]\nclass = b'),
 Text(505.35849056603774, 181.19999999999982, 'gini = 0.0\nsamples = 14\nvalue = [25, 0]\nclass = a'),
 Text(758.0377358490566, 543.5999999999999, 'TOL <= 1.05\ngini = 0.026\nsamples = 46\nvalue = [76, 1]\nclass = a'),
 Text(673.811320754717, 181.19999999999982, 'gini = 0.0\nsamples = 41\nvalue = [72, 0]\nclass = a'),
 Text(842.2641509433963, 181.19999999999982, 'gini = 0.32\nsamples = 5\nvalue = [4, 1]\nclass = a'),
 Text(1221.2830188679245, 1268.4, 'NO <= 1.5\ngini = 0.487\nsamples = 372\nvalue = [341, 245]\nclass = a'),
 Text(1010.7169811320755, 906.0, 'TCH <= 1.345\ngini = 0.332\nsamples = 128\nvalue = [41, 154]\nclass = b'),
 Text(926.4905660377359, 543.5999999999999, 'gini = 0.0\nsamples = 96\nvalue = [0, 154]\nclass = b'),
 Text(1094.9433962264152, 543.5999999999999, 'gini = 0.0\nsamples = 32\nvalue = [41, 0]\nclass = a'),
 Text(1431.8490566037738, 906.0, 'SO_2 <= 3.5\ngini = 0.357\nsamples = 244\nvalue = [300, 91]\nclass = a'),
 Text(1263.3962264150944, 543.5999999999999, 'TCH <= 1.4\ngini = 0.043\nsamples = 57\nvalue = [2, 89]\nclass = b'),
 Text(1179.169811320755, 181.19999999999982, 'gini = 0.0\nsamples = 52\nvalue = [0, 84]\nclass = b'),
 Text(1347.622641509434, 181.19999999999982, 'gini = 0.408\nsamples = 5\nvalue = [2, 5]\nclass = b'),
 Text(1600.301886792453, 543.5999999999999, 'NO <= 7.5\ngini = 0.013\nsamples = 187\nvalue = [298, 2]\nclass = a'),
 Text(1516.0754716981132, 181.19999999999982, 'gini = 0.0\nsamples = 176\nvalue = [282, 0]\nclass = a'),
 Text(1684.5283018867926, 181.19999999999982, 'gini = 0.198\nsamples = 11\nvalue = [16, 2]\nclass = a'),
 Text(3116.377358490566, 1630.8000000000002, 'NO_2 <= 44.5\ngini = 0.411\nsamples = 4969\nvalue = [5543, 2258]\nclass = a'),
 Text(2442.566037735849, 1268.4, 'TCH <= 1.335\ngini = 0.492\nsamples = 2082\nvalue = [1834, 1430]\nclass = a'),
 Text(2105.6603773584907, 906.0, 'NMHC <= 0.095\ngini = 0.184\nsamples = 813\nvalue = [131, 1149]\nclass = b'),
 Text(1937.2075471698115, 543.5999999999999, 'NMHC <= 0.045\ngini = 0.115\nsamples = 719\nvalue = [69, 1062]\nclass = b'),
 Text(1852.9811320754718, 181.19999999999982, 'gini = 0.283\nsamples = 183\nvalue = [48, 233]\nclass = b'),
 Text(2021.433962264151, 181.19999999999982, 'gini = 0.048\nsamples = 536\nvalue = [21, 829]\nclass = b'),
 Text(2274.11320754717, 543.5999999999999, 'PM10 <= 10.5\ngini = 0.486\nsamples = 94\nvalue = [62, 87]\nclass = b'),
 Text(2189.8867924528304, 181.19999999999982, 'gini = 0.168\nsamples = 33\nvalue = [49, 5]\nclass = a'),
 Text(2358.33962264151, 181.19999999999982, 'gini = 0.236\nsamples = 61\nvalue = [13, 8
```

```
2]\nclass = b'),
 Text(2779.471698113208, 906.0, 'BEN <= 0.75\ngini = 0.243\nsamples = 1269\nvalue = [170
3, 281]\nclass = a'),
 Text(2611.0188679245284, 543.5999999999999, 'SO_2 <= 3.5\ngini = 0.205\nsamples = 1169
\nvalue = [1632, 214]\nclass = a'),
 Text(2526.7924528301887, 181.19999999999982, 'gini = 0.231\nsamples = 135\nvalue = [28,
182]\nclass = b'),
 Text(2695.245283018868, 181.19999999999982, 'gini = 0.038\nsamples = 1034\nvalue = [160
4, 32]\nclass = a'),
 Text(2947.924528301887, 543.5999999999999, 'SO_2 <= 5.5\ngini = 0.5\nsamples = 100\nval
ue = [71, 67]\nclass = a'),
 Text(2863.6981132075475, 181.19999999999982, 'gini = 0.03\nsamples = 45\nvalue = [1, 6
4]\nclass = b'),
 Text(3032.1509433962265, 181.19999999999982, 'gini = 0.079\nsamples = 55\nvalue = [70,
3]\nclass = a'),
 Text(3790.1886792452833, 1268.4, 'SO_2 <= 4.5\ngini = 0.298\nsamples = 2887\nvalue = [3
709, 828]\nclass = a'),
 Text(3453.283018867925, 906.0, 'SO_2 <= 3.5\ngini = 0.213\nsamples = 476\nvalue = [92,
667]\nclass = b'),
 Text(3284.8301886792456, 543.5999999999999, 'O_3 <= 6.5\ngini = 0.1\nsamples = 319\nval
ue = [27, 485]\nclass = b'),
 Text(3200.603773584906, 181.19999999999982, 'gini = 0.0\nsamples = 110\nvalue = [0, 20
2]\nclass = b'),
 Text(3369.0566037735853, 181.19999999999982, 'gini = 0.159\nsamples = 209\nvalue = [27,
283]\nclass = b'),
 Text(3621.735849056604, 543.5999999999999, 'TCH <= 1.44\ngini = 0.388\nsamples = 157\nv
alue = [65, 182]\nclass = b'),
 Text(3537.509433962264, 181.19999999999982, 'gini = 0.0\nsamples = 98\nvalue = [0, 155]
\nclass = b'),
 Text(3705.9622641509436, 181.19999999999982, 'gini = 0.415\nsamples = 59\nvalue = [65,
27]\nclass = a'),
 Text(4127.094339622642, 906.0, 'PM10 <= 26.5\ngini = 0.082\nsamples = 2411\nvalue = [36
17, 161]\nclass = a'),
 Text(3958.6415094339627, 543.5999999999999, 'TCH <= 1.315\ngini = 0.015\nsamples = 1433
\nvalue = [2234, 17]\nclass = a'),
 Text(3874.415094339623, 181.19999999999982, 'gini = 0.252\nsamples = 36\nvalue = [52,
9]\nclass = a'),
 Text(4042.867924528302, 181.19999999999982, 'gini = 0.007\nsamples = 1397\nvalue = [218
2, 8]\nclass = a'),
 Text(4295.5471698113215, 543.5999999999999, 'O_3 <= 4.5\ngini = 0.171\nsamples = 978\nv
alue = [1383, 144]\nclass = a'),
 Text(4211.320754716981, 181.19999999999982, 'gini = 0.485\nsamples = 103\nvalue = [96,
68]\nclass = a'),
 Text(4379.773584905661, 181.19999999999982, 'gini = 0.105\nsamples = 875\nvalue = [128
7, 76]\nclass = a')]
```



Conclusion

Accuracy

linear regression

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

Out[64]: 0.8285268680585672

Ridge regression

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

Out[65]: 0.8283861171129381

Lasso regression

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

Out[66]: 0.6453921030812604

Elastic net regression

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

Out[67]: 0.7040306356663145

Logistic regression

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

Out[68]: 0.996161115048429

Random forest

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

Out[69]: 0.9938407019912251

Accuracy for logistic regression is higher so it is the best fit model