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_2015.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	01-10-2015 01:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	28079004
1	01-10-2015 01:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	28079008
2	01-10-2015 01:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	28079011
3	01-10-2015 01:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	01-10-2015 01:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210091	01-08-2015 00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN	28079056
210092	01-08-2015 00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN	28079057
210093	01-08-2015 00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN	28079058

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
210094	01-08-2015 00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN	28079059
210095	01-08-2015 00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN	28079060

210096 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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

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

Out[6]:

	CO	station
1	0.8	28079008
6	0.3	28079024

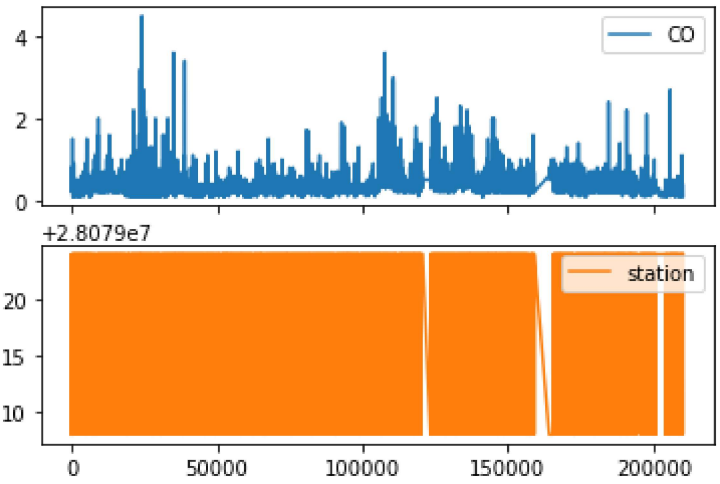
	CO	station
25	0.7	28079008
30	0.3	28079024
49	0.8	28079008
...	...	...
210030	0.1	28079024
210049	0.3	28079008
210054	0.1	28079024
210073	0.3	28079008
210078	0.1	28079024

16026 rows × 2 columns

Line chart

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

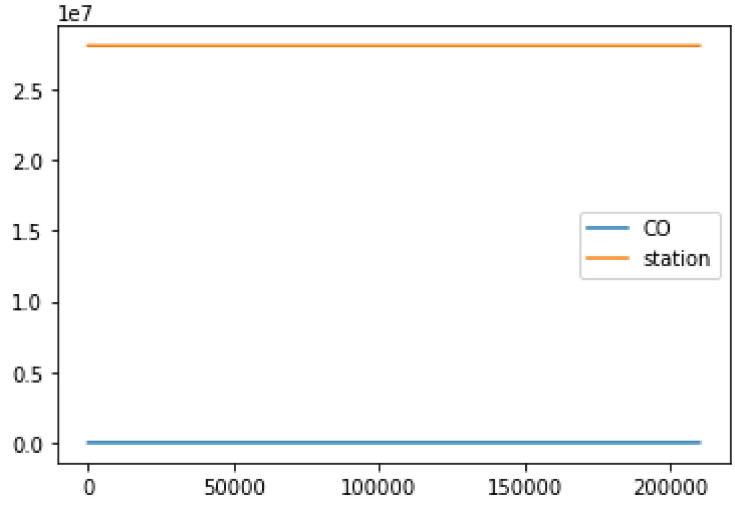
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

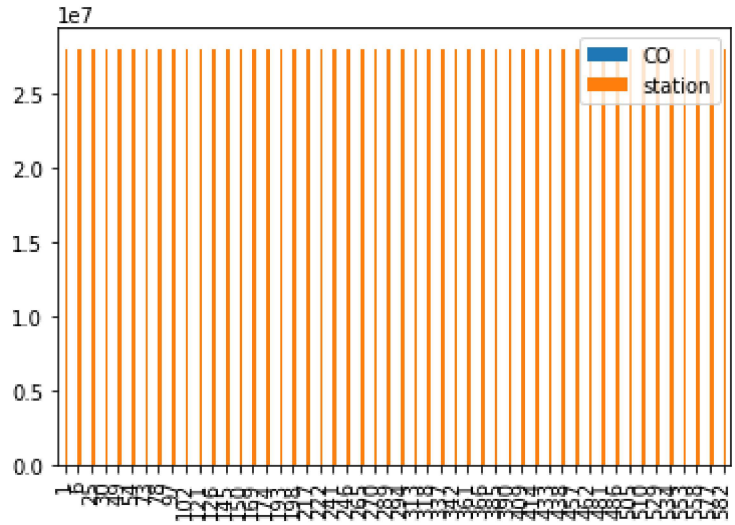


Bar chart

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

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

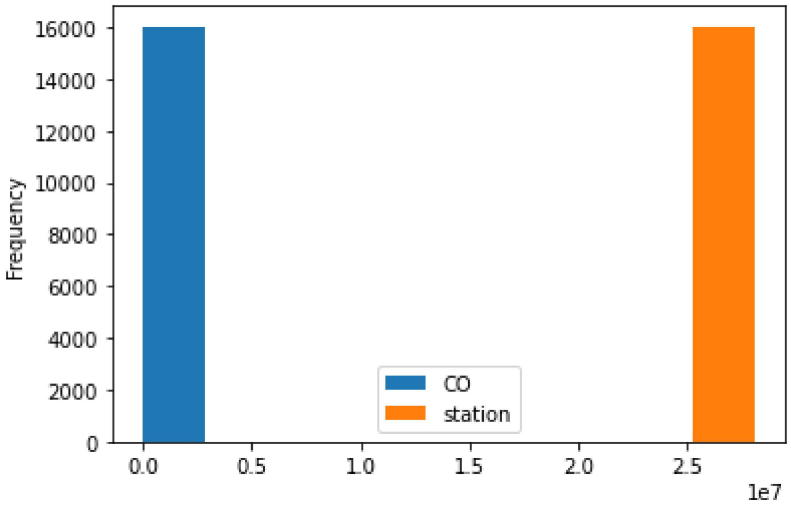
Out[10]: <AxesSubplot:>



Histogram

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

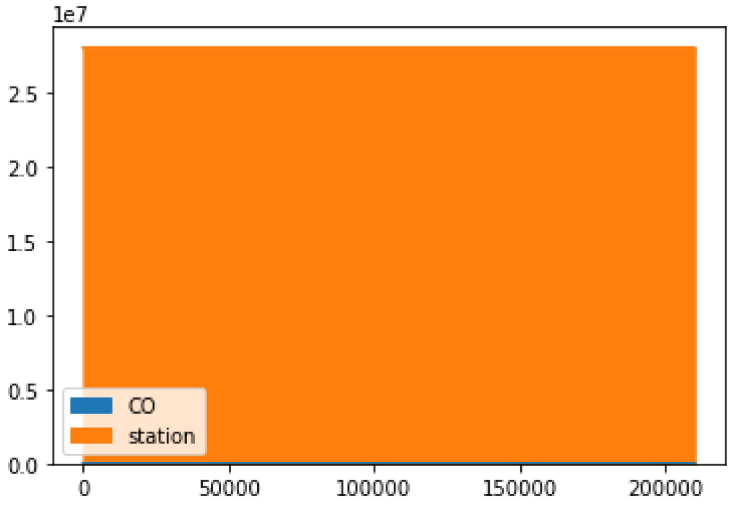
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

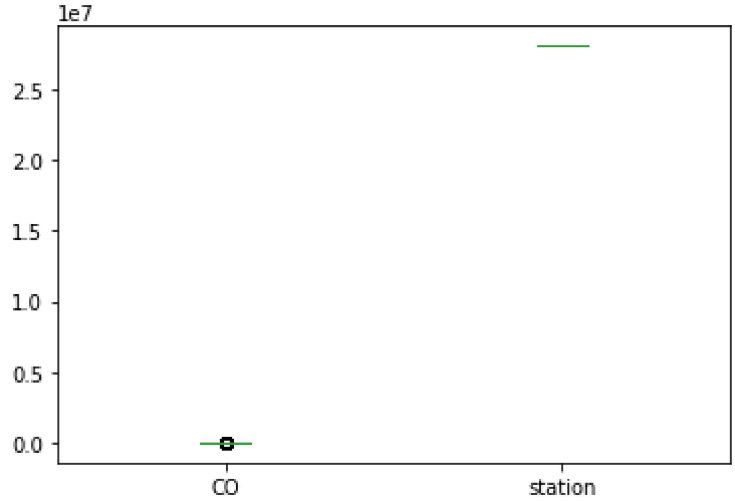
Out[12]: <AxesSubplot:>



Box chart

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

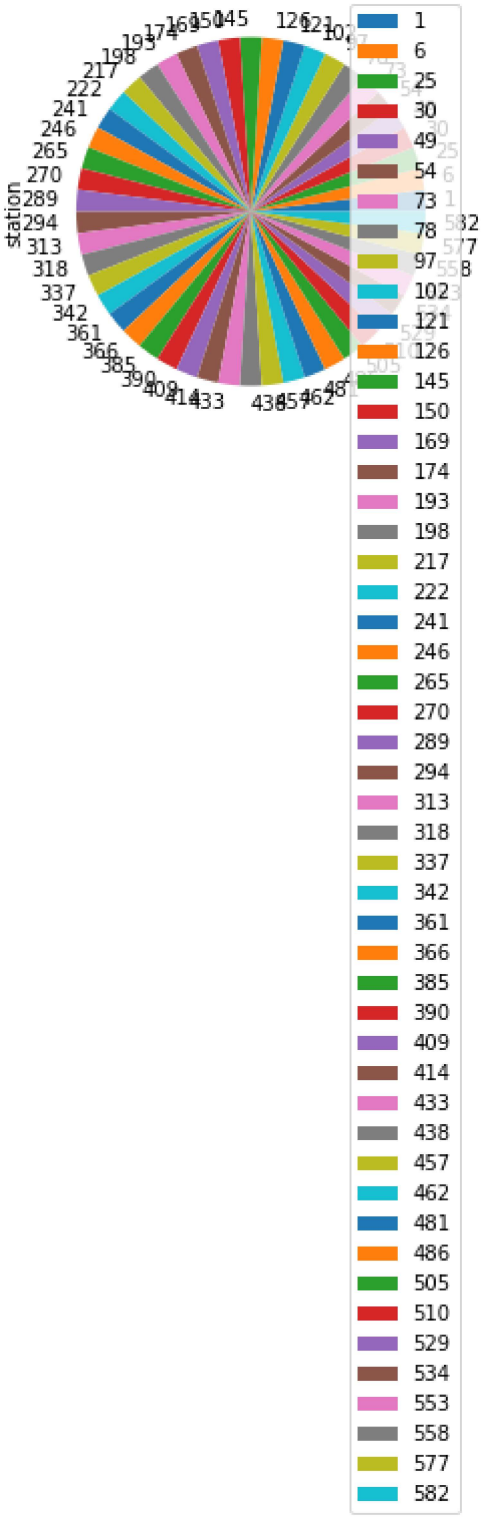
Out[13]: <AxesSubplot:>



Pie chart

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

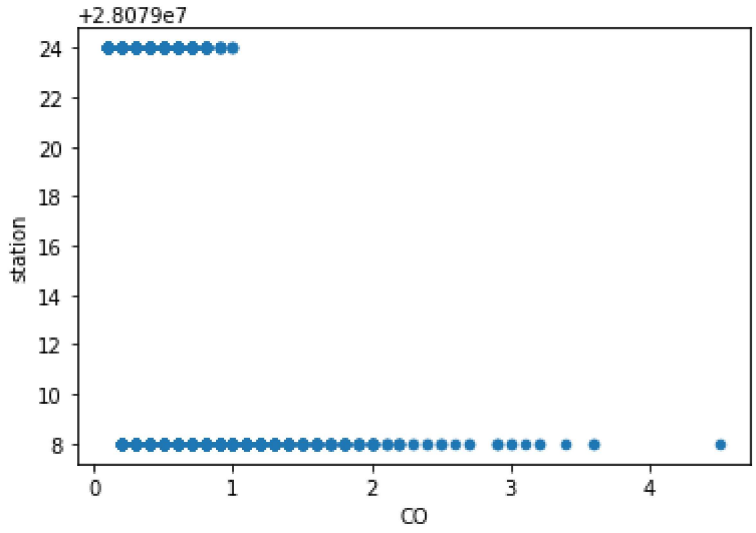
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ 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	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	48.089793
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	35.847298
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000

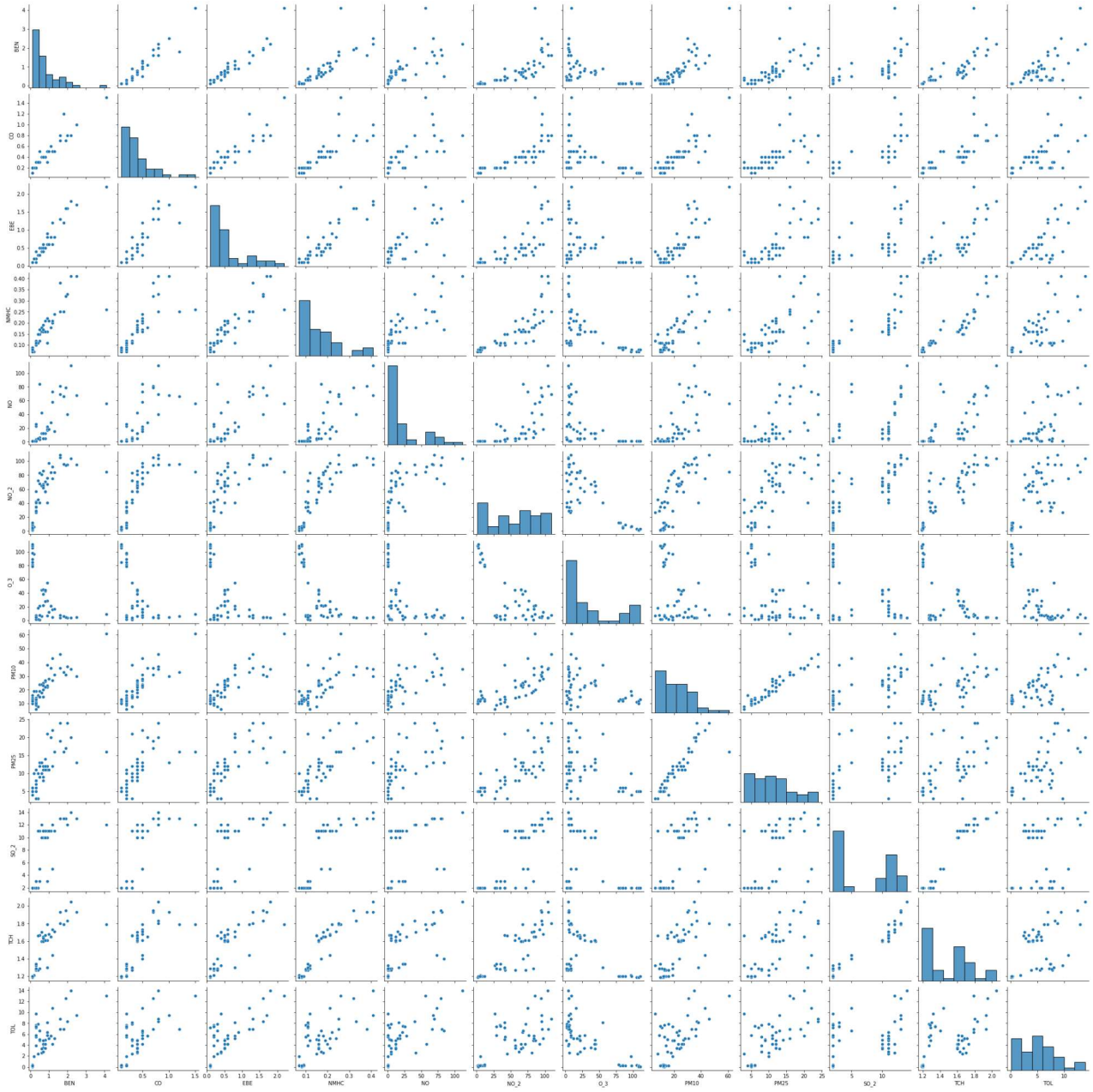
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	15.00000
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	46.00000
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	73.00000
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	217.00000

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

EDA AND VISUALIZATION

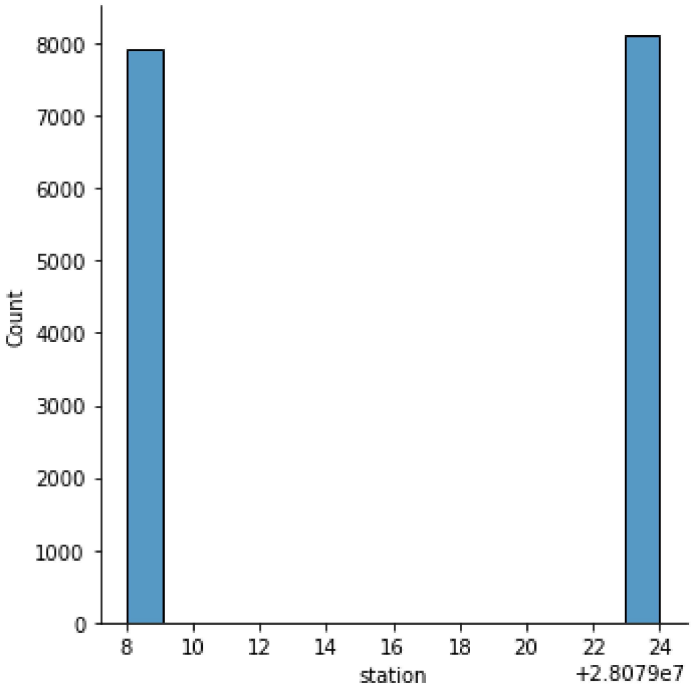
In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x1fd7be8da30>



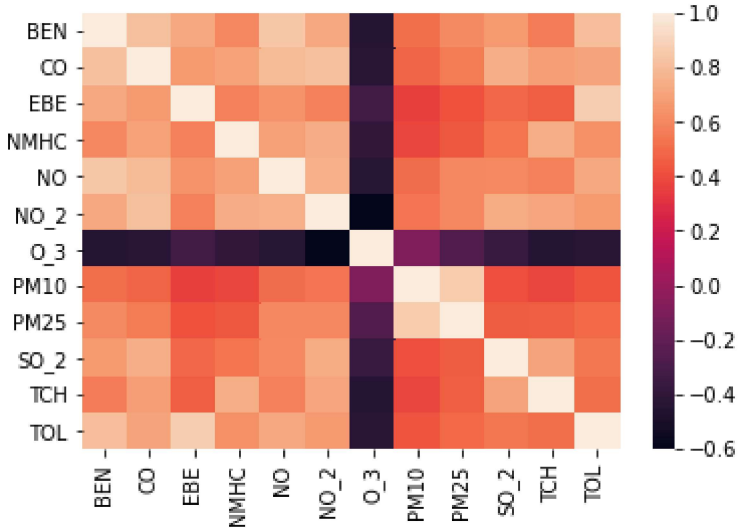
In [21]:
```
sns.displot(df['station'])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x1fd7be8d460>
```



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]:
```
28079038.34809353
```

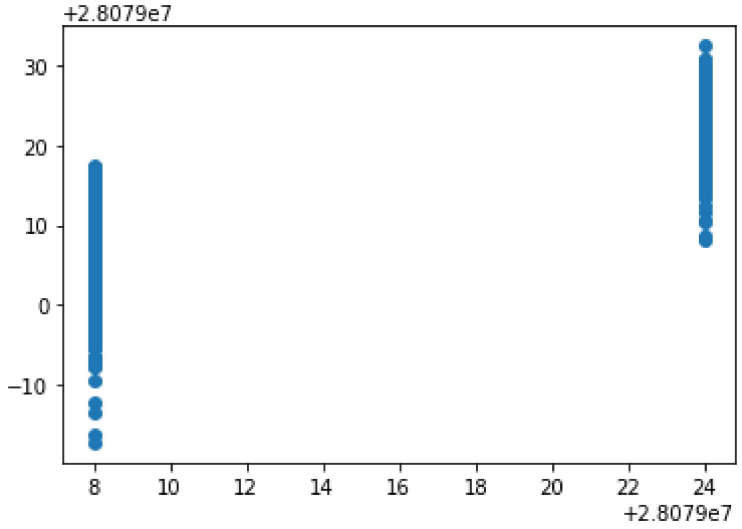
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	1.045512
CO	-10.111712
EBE	-0.681508
NMHC	13.086840
NO	0.081929
NO_2	-0.016513
O_3	-0.013979
PM10	0.007801
PM25	0.101031
SO_2	-1.108261
TCH	-9.660729
TOL	-0.093679

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

Out[28]:
```
<matplotlib.collections.PathCollection at 0x1fd07426ac0>
```



ACCURACY

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

Out[29]: 0.8679734814182677

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

Out[30]: 0.8731964427153875

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]: 0.8671870120660073

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

Out[34]: 0.8724173411097744

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]:
```
0.7288597557396053
```

Accuracy(Lasso)

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

Out[37]:
```
0.7238748028710862
```

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.        ,  0.0741257 ,
       -0.05394271, -0.01242968,  0.02406781,  0.05071896, -1.30735765,
       -0.        , -0.09495649])
```

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

Out[40]:
```
28079025.963690557
```

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

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

Out[42]:
```
0.8224453702627968
```

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)))
```

```
2.506361219588785
11.360505607232426
3.3705349141096916
```

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]: (16026, 12)

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

Out[47]: (16026,)

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]:
```
0.9971296642955197
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 1.54284914e-35]])
```

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]:
```
0.99447316812266
```

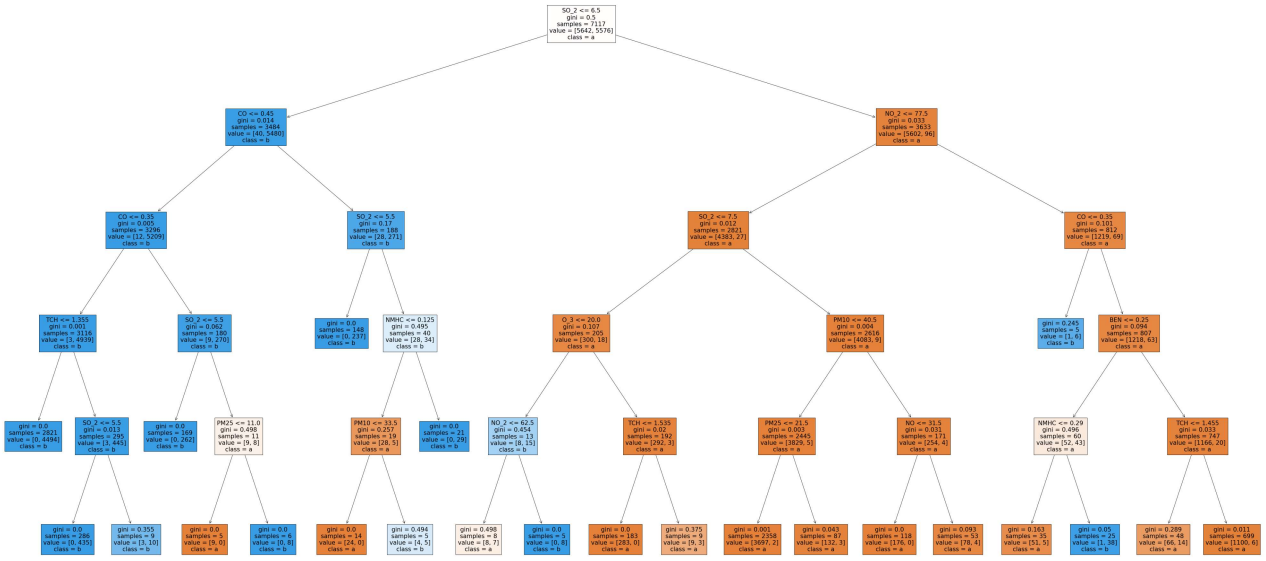
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(2051.027027027027, 1993.2, 'SO_2 <= 6.5\ngini = 0.5\nsamples = 7117\nvalue = [5642, 5576]\nclass = a'),
 Text(904.8648648648648, 1630.8000000000002, 'CO <= 0.45\ngini = 0.014\nsamples = 3484\nvalue = [40, 5480]\nclass = b'),
 Text(482.5945945945946, 1268.4, 'CO <= 0.35\ngini = 0.005\nsamples = 3296\nvalue = [12, 5209]\nclass = b'),
 Text(241.2972972972973, 906.0, 'TCH <= 1.355\ngini = 0.001\nsamples = 3116\nvalue = [3, 4939]\nclass = b'),
 Text(120.64864864864865, 543.5999999999999, 'gini = 0.0\nsamples = 2821\nvalue = [0, 4494]\nclass = b'),
 Text(361.94594594594594, 543.5999999999999, 'SO_2 <= 5.5\ngini = 0.013\nsamples = 295\nvalue = [3, 445]\nclass = b'),
 Text(241.2972972972973, 181.19999999999982, 'gini = 0.0\nsamples = 286\nvalue = [0, 435]\nclass = b'),
 Text(482.5945945945946, 181.19999999999982, 'gini = 0.355\nsamples = 9\nvalue = [3, 10]\nclass = b'),
 Text(723.8918918918919, 906.0, 'SO_2 <= 5.5\ngini = 0.062\nsamples = 180\nvalue = [9, 270]\nclass = b'),
 Text(603.2432432432432, 543.5999999999999, 'gini = 0.0\nsamples = 169\nvalue = [0, 262]\nclass = b'),
 Text(844.5405405405405, 543.5999999999999, 'PM25 <= 11.0\ngini = 0.498\nsamples = 11\nvalue = [9, 8]\nclass = a'),
 Text(723.8918918918919, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [9, 0]\nclass = a'),
 Text(965.1891891891892, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [0, 8]\nclass = b'),
 Text(1327.135135135135, 1268.4, 'SO_2 <= 5.5\ngini = 0.17\nsamples = 188\nvalue = [28, 271]\nclass = b'),
 Text(1206.4864864864865, 906.0, 'gini = 0.0\nsamples = 148\nvalue = [0, 237]\nclass = b'),
 Text(1447.7837837837837, 906.0, 'NMHC <= 0.125\ngini = 0.495\nsamples = 40\nvalue = [28, 34]\nclass = b'),
 Text(1327.135135135135, 543.5999999999999, 'PM10 <= 33.5\ngini = 0.257\nsamples = 19\nvalue = [28, 5]\nclass = a'),
 Text(1206.4864864864865, 181.19999999999982, 'gini = 0.0\nsamples = 14\nvalue = [24, 0]\nclass = a'),
 Text(1447.7837837837837, 181.19999999999982, 'gini = 0.494\nsamples = 5\nvalue = [4, 5]\nclass = b'),
 Text(1568.4324324324325, 543.5999999999999, 'gini = 0.0\nsamples = 21\nvalue = [0, 29]\nclass = b'),
 Text(3197.189189189189, 1630.8000000000002, 'NO_2 <= 77.5\ngini = 0.033\nsamples = 3633\nvalue = [5602, 96]\nclass = a'),
 Text(2533.6216216216217, 1268.4, 'SO_2 <= 7.5\ngini = 0.012\nsamples = 2821\nvalue = [4383, 27]\nclass = a'),
 Text(2051.027027027027, 906.0, 'O_3 <= 20.0\ngini = 0.107\nsamples = 205\nvalue = [300, 18]\nclass = a'),
 Text(1809.7297297297296, 543.5999999999999, 'NO_2 <= 62.5\ngini = 0.454\nsamples = 13\nvalue = [8, 15]\nclass = b'),
 Text(1689.081081081081, 181.19999999999982, 'gini = 0.498\nsamples = 8\nvalue = [8, 7]\nclass = a'),
 Text(1930.3783783783783, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [0, 8]\nclass = b'),
 Text(2292.324324324324, 543.5999999999999, 'TCH <= 1.535\ngini = 0.02\nsamples = 192\nvalue = [292, 3]\nclass = a'),
 Text(2171.675675675676, 181.19999999999982, 'gini = 0.0\nsamples = 183\nvalue = [283, 0]\nclass = a'),
 Text(2412.972972972973, 181.19999999999982, 'gini = 0.375\nsamples = 9\nvalue = [9, 3]\nclass = a'),
 Text(3016.2162162162163, 906.0, 'PM10 <= 40.5\ngini = 0.004\nsamples = 2616\nvalue = [4083, 9]\nclass = a'),
 Text(2774.9189189189187, 543.5999999999999, 'PM25 <= 21.5\ngini = 0.003\nsamples = 2445\nvalue = [3829, 5]\nclass = a'),
 Text(2654.27027027027, 181.19999999999982, 'gini = 0.001\nsamples = 2358\nvalue = [3697, 2]\nclass = a'),
 Text(2895.5675675675675, 181.19999999999982, 'gini = 0.043\nsamples = 87\nvalue = [132,
```

```
3]\nclass = a'),
 Text(3257.5135135135133, 543.5999999999999, 'NO <= 31.5\ngini = 0.031\nsamples = 171\nv
alue = [254, 4]\nclass = a'),
 Text(3136.864864864865, 181.19999999999982, 'gini = 0.0\nsamples = 118\nvalue = [176,
0]\nclass = a'),
 Text(3378.162162162162, 181.19999999999982, 'gini = 0.093\nsamples = 53\nvalue = [78,
4]\nclass = a'),
 Text(3860.7567567567567, 1268.4, 'CO <= 0.35\ngini = 0.101\nsamples = 812\nvalue = [121
9, 69]\nclass = a'),
 Text(3740.108108108108, 906.0, 'gini = 0.245\nsamples = 5\nvalue = [1, 6]\nclass = b'),
 Text(3981.4054054054054, 906.0, 'BEN <= 0.25\ngini = 0.094\nsamples = 807\nvalue = [121
8, 63]\nclass = a'),
 Text(3740.108108108108, 543.5999999999999, 'NMHC <= 0.29\ngini = 0.496\nsamples = 60\nv
alue = [52, 43]\nclass = a'),
 Text(3619.459459459459, 181.19999999999982, 'gini = 0.163\nsamples = 35\nvalue = [51,
5]\nclass = a'),
 Text(3860.7567567567567, 181.19999999999982, 'gini = 0.05\nsamples = 25\nvalue = [1, 3
8]\nclass = b'),
 Text(4222.7027027027025, 543.5999999999999, 'TCH <= 1.455\ngini = 0.033\nsamples = 747
\nvalue = [1166, 20]\nclass = a'),
 Text(4102.054054054054, 181.19999999999982, 'gini = 0.289\nsamples = 48\nvalue = [66, 1
4]\nclass = a'),
 Text(4343.351351351352, 181.19999999999982, 'gini = 0.011\nsamples = 699\nvalue = [110
0, 6]\nclass = a')]
```



Conclusion

Accuracy

linear regression

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

Out[64]: 0.8679734814182677

Ridge regression

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

Out[65]: 0.8671870120660073

Lasso regression

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

Out[66]: 0.7238748028710862

Elastic net regression

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

Out[67]: 0.8224453702627968

Logistic regression

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

Out[68]: 0.9971296642955197

Random forest

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

Out[69]: 0.99447316812266

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