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_2012.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	01-09-2012 01:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	28079004
1	01-09-2012 01:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	28079008
2	01-09-2012 01:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	28079011
3	01-09-2012 01:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	28079016
4	01-09-2012 01:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210715	01-03-2012 00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN	28079056
210716	01-03-2012 00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN	28079057
210717	01-03-2012 00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN	28079058

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
210718	01-03-2012 00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN	28079059
210719	01-03-2012 00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN	28079060

210720 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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ MB
```

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

Out[6]:

	CO	station
6	0.2	28079024
30	0.2	28079024

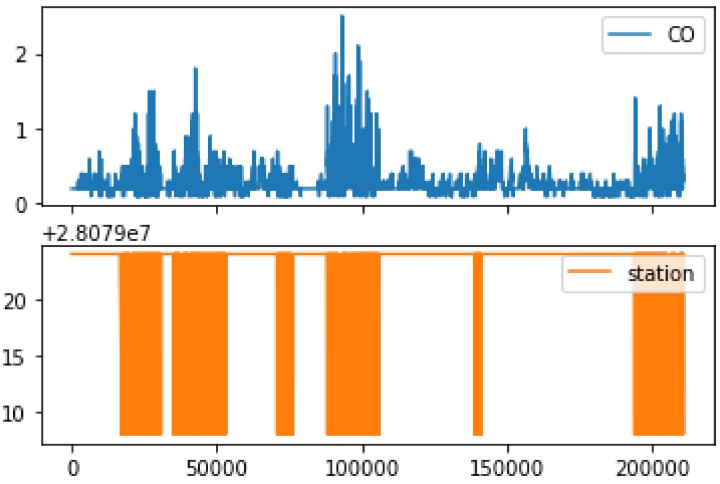
	CO	station
54	0.2	28079024
78	0.2	28079024
102	0.2	28079024
...	...	...
210654	0.3	28079024
210673	0.4	28079008
210678	0.3	28079024
210697	0.4	28079008
210702	0.3	28079024

10916 rows × 2 columns

Line chart

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

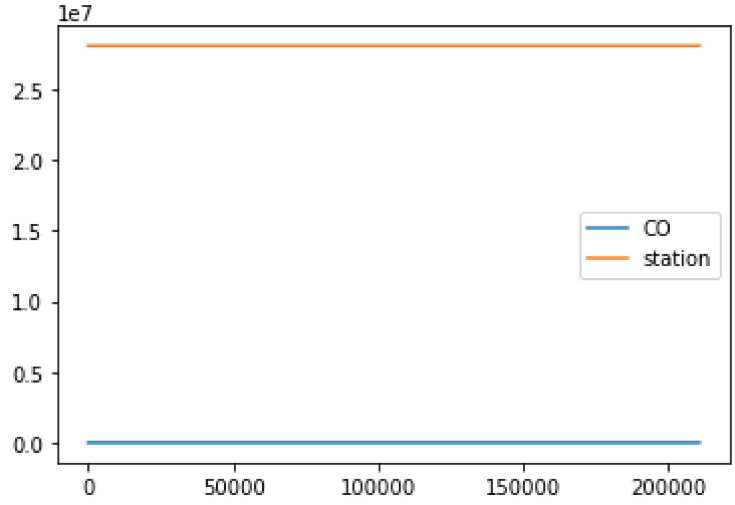
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

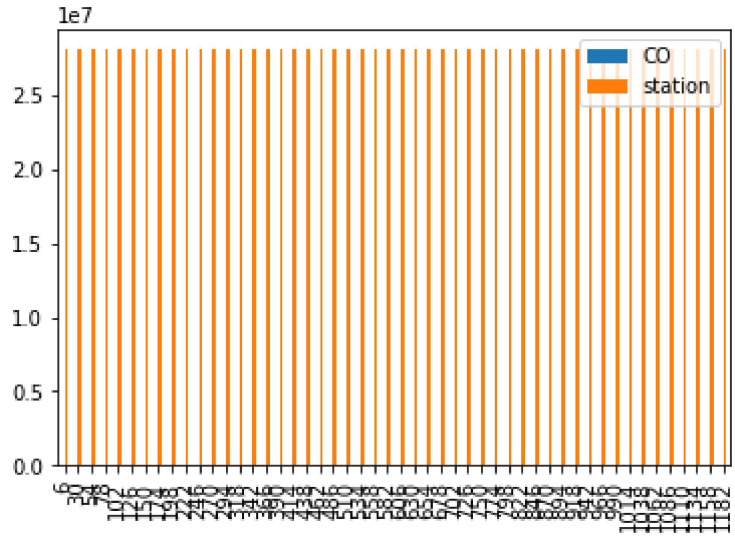


Bar chart

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

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

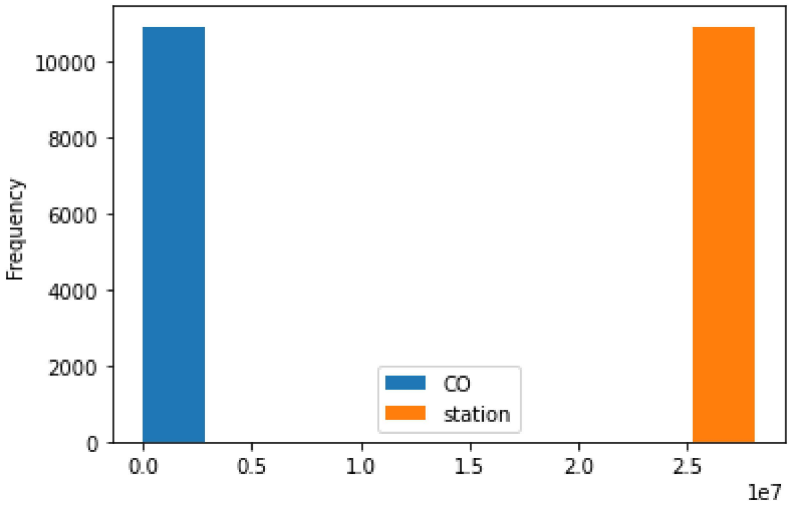
Out[10]: <AxesSubplot:>



Histogram

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

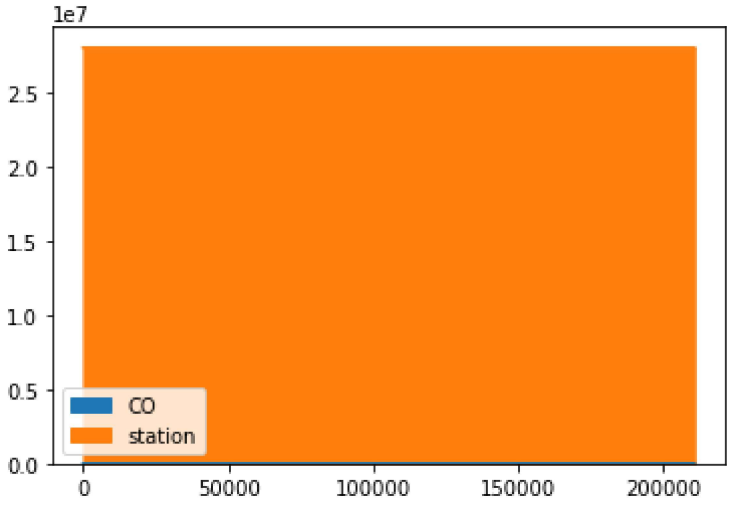
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

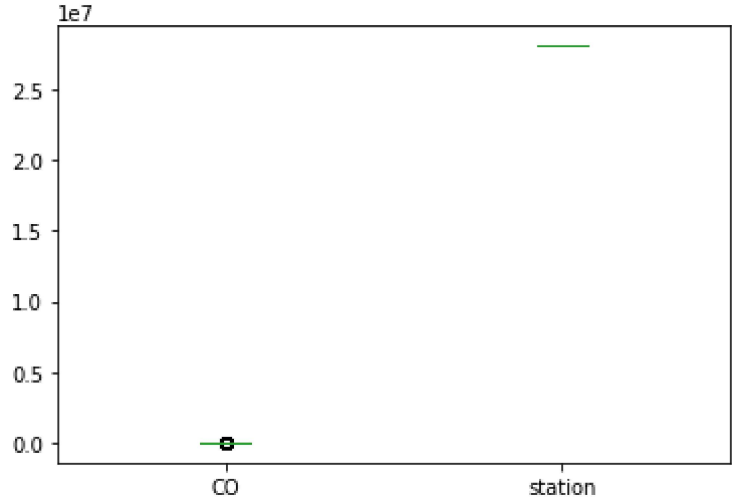
Out[12]: <AxesSubplot:>



Box chart

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

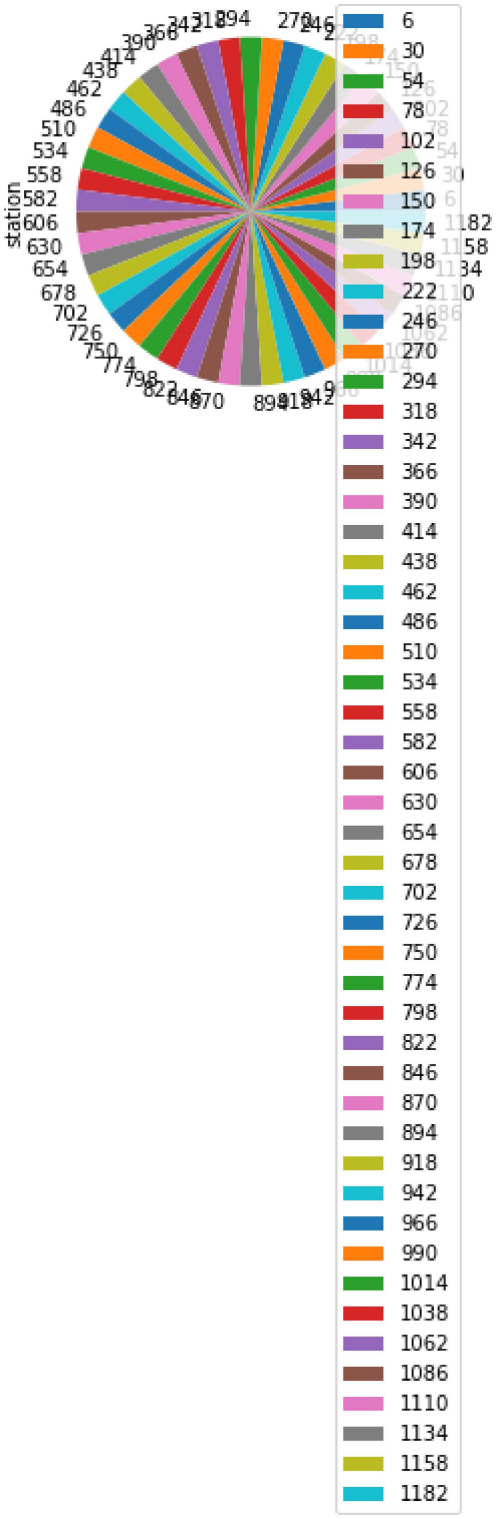
Out[13]: <AxesSubplot:>



Pie chart

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

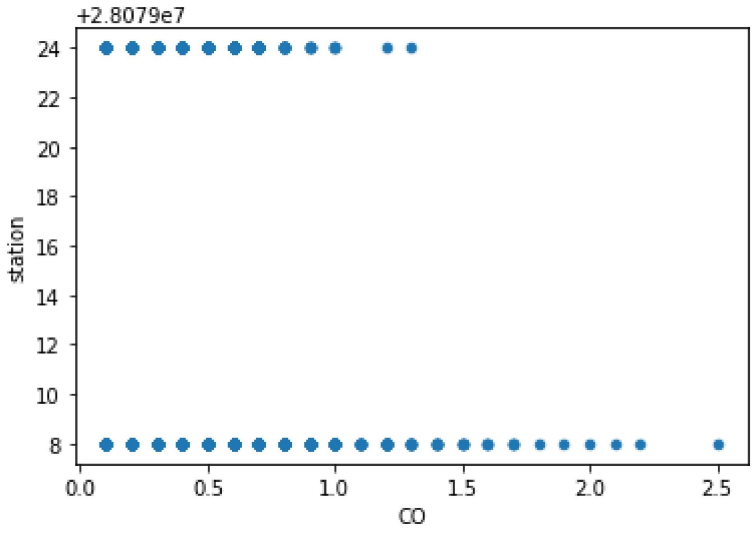
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 10916 entries, 6 to 210702
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     10916 non-null  object 
 1   BEN      10916 non-null  float64
 2   CO       10916 non-null  float64
 3   EBE      10916 non-null  float64
 4   NMHC     10916 non-null  float64
 5   NO       10916 non-null  float64
 6   NO_2     10916 non-null  float64
 7   O_3      10916 non-null  float64
 8   PM10     10916 non-null  float64
 9   PM25     10916 non-null  float64
 10  SO_2     10916 non-null  float64
 11  TCH      10916 non-null  float64
 12  TOL      10916 non-null  float64
 13  station  10916 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.2+ 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	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000	10916.000000
mean	0.784014	0.279333	0.992213	0.215755	18.795529	31.262642	44.239557
std	0.632755	0.167922	0.804554	0.075169	40.038872	27.234732	29.535566
min	0.100000	0.100000	0.100000	0.050000	0.000000	1.000000	1.000000

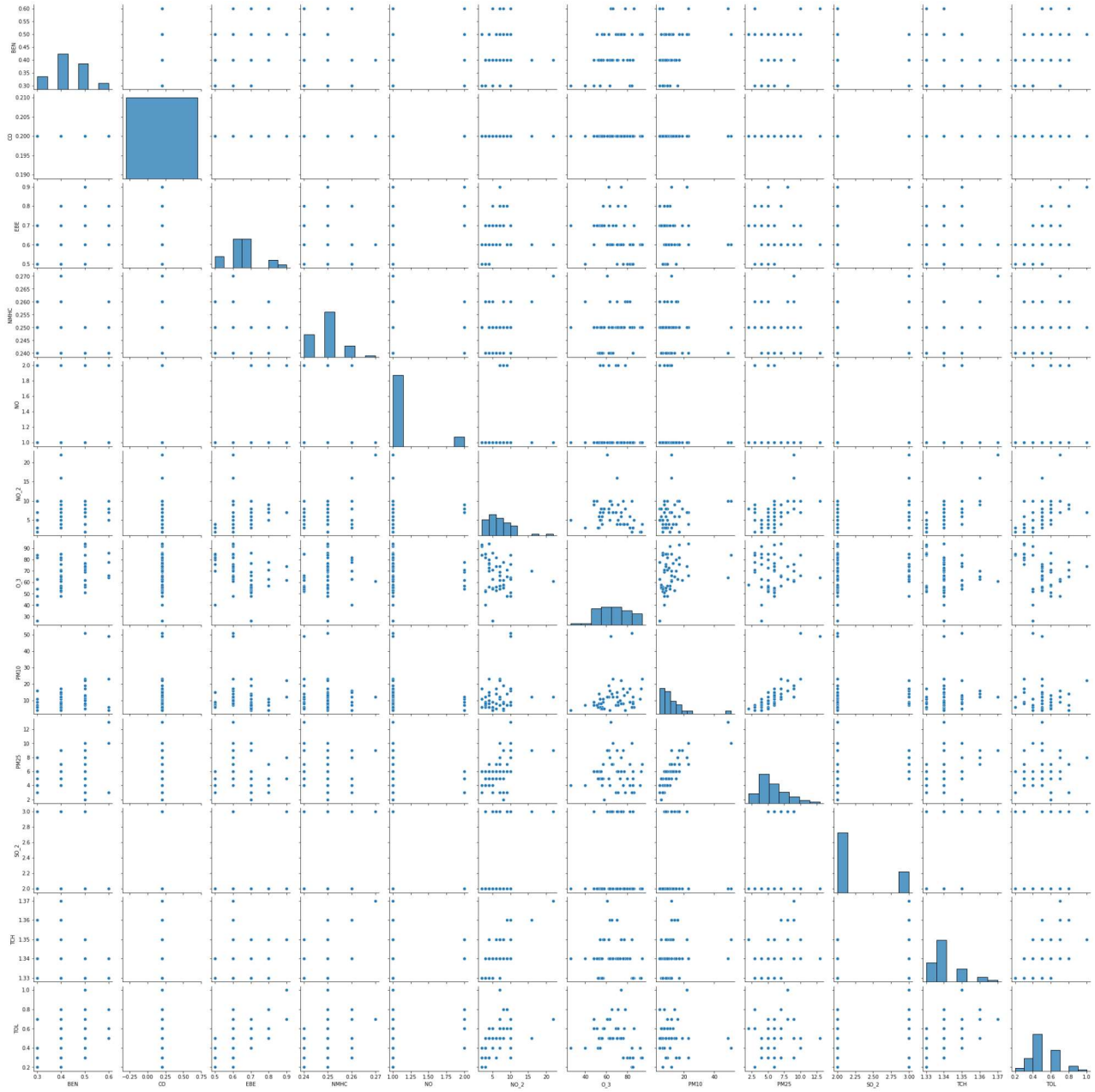
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
25%	0.400000	0.200000	0.500000	0.160000	1.000000	9.000000	18.00000
50%	0.600000	0.200000	0.800000	0.220000	3.000000	24.000000	44.00000
75%	0.900000	0.300000	1.200000	0.250000	18.000000	47.000000	65.00000
max	7.000000	2.500000	9.700000	0.670000	525.000000	225.000000	157.00000

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

EDA AND VISUALIZATION

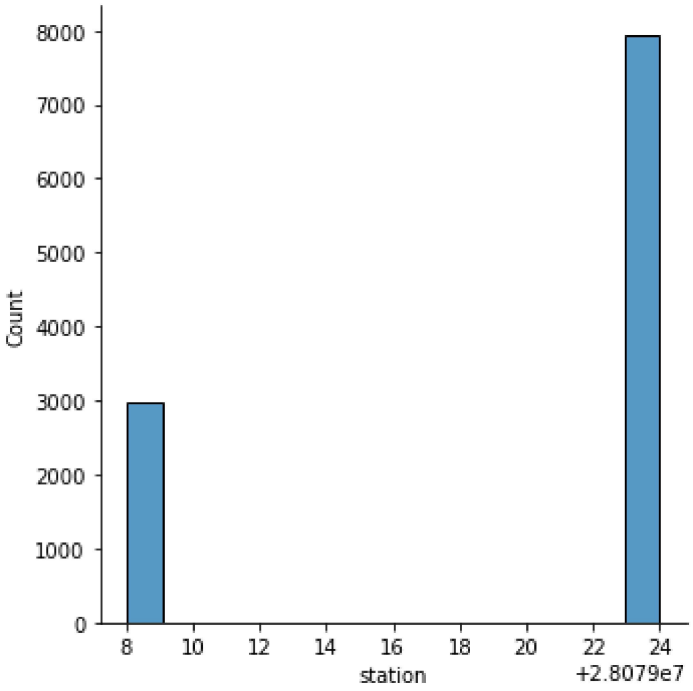
In [20]:
```
sns.pairplot(df1[0:50])
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x298e53db730>



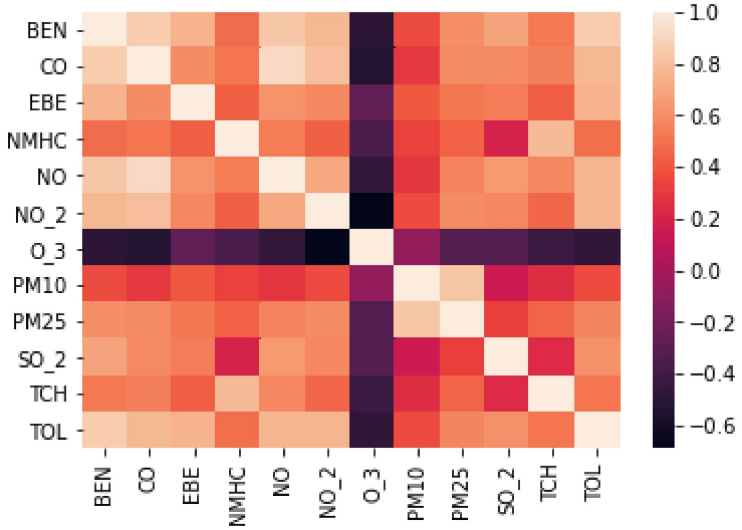
In [21]:
```
sns.displot(df['station'])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x298ec9504f0>
```



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]:
```
28079017.61714522
```

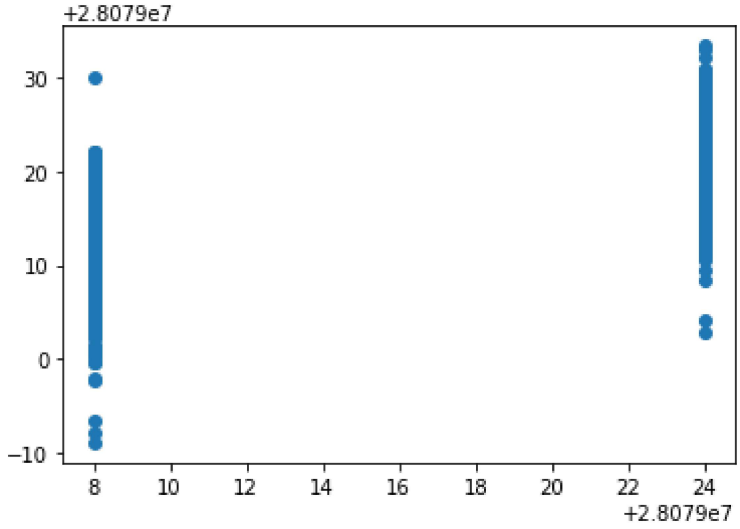
In [27]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[27]:

	Co-efficient
BEN	3.570210
CO	22.107921
EBE	-0.442427
NMHC	18.195569
NO	-0.023312
NO_2	-0.124208
O_3	-0.030521
PM10	-0.001523
PM25	-0.038328
SO_2	-0.686357
TCH	1.575592
TOL	-1.336603

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

Out[28]:
```
<matplotlib.collections.PathCollection at 0x298ef684a00>
```



ACCURACY

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

Out[29]: 0.6306803749037257

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

Out[30]: 0.621473393337072

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]: 0.6289080365797595

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

Out[34]: 0.617111303088054

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]:
```
0.36280836303852226
```

Accuracy(Lasso)

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

Out[37]:
```
0.36169540854582505
```

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.        ,  0.06722606,
       -0.09017481, -0.03724533, -0.        ,  0.        , -0.7156085 ,
        0.        , -0.65934029])
```

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

Out[40]:
```
28079027.71277003
```

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

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

Out[42]:
```
0.5368065685385757
```

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)))
```

```
3.4472610145340896
23.417346977795216
4.839147339955172
```

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]: (10916, 12)

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

Out[47]: (10916,)

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]:
```
0.9311102968120191
```

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

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

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

Out[56]:
```
array([[1.00000000e+00, 7.14919064e-33]])
```

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]:
```
0.9647950976348671
```

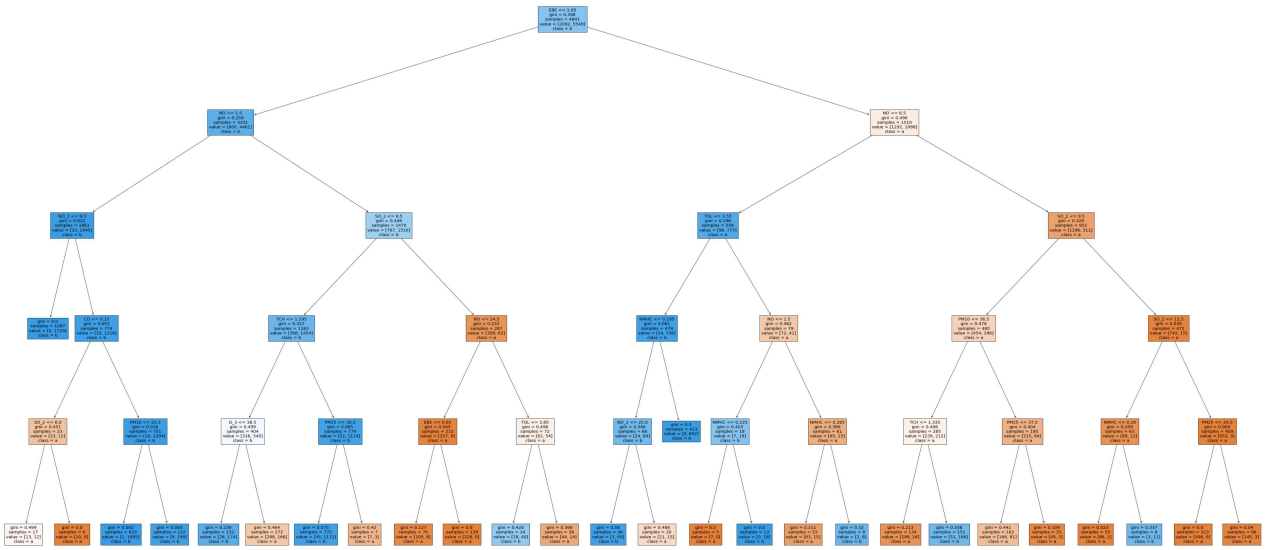
In [62]:
```
rfc_best=grid_search.best_estimator_
```

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(1985.1923076923076, 1993.2, 'EBE <= 1.05\ngini = 0.398\nsamples = 4841\nvalue = [2
092, 5549]\nclass = b'),
 Text(815.5384615384614, 1630.8000000000002, 'NO <= 1.5\ngini = 0.258\nsamples = 3331\nv
alue = [800, 4461]\nclass = b'),
 Text(257.53846153846155, 1268.4, 'NO_2 <= 9.5\ngini = 0.022\nsamples = 1861\nvalue = [3
3, 2945]\nclass = b'),
 Text(171.69230769230768, 906.0, 'gini = 0.0\nsamples = 1087\nvalue = [0, 1729]\nclass =
b'),
 Text(343.38461538461536, 906.0, 'CO <= 0.15\ngini = 0.051\nsamples = 774\nvalue = [33,
1216]\nclass = b'),
 Text(171.69230769230768, 543.5999999999999, 'SO_2 <= 6.0\ngini = 0.451\nsamples = 23\nv
alue = [23, 12]\nclass = a'),
 Text(85.84615384615384, 181.19999999999982, 'gini = 0.499\nsamples = 17\nvalue = [13, 1
2]\nclass = a'),
 Text(257.53846153846155, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [10, 0]
\nclass = a'),
 Text(515.0769230769231, 543.5999999999999, 'PM10 <= 25.5\ngini = 0.016\nsamples = 751\n
value = [10, 1204]\nclass = b'),
 Text(429.23076923076917, 181.19999999999982, 'gini = 0.002\nsamples = 629\nvalue = [1,
1005]\nclass = b'),
 Text(600.9230769230769, 181.19999999999982, 'gini = 0.083\nsamples = 122\nvalue = [9, 1
99]\nclass = b'),
 Text(1373.5384615384614, 1268.4, 'SO_2 <= 6.5\ngini = 0.446\nsamples = 1470\nvalue = [7
67, 1516]\nclass = b'),
 Text(1030.1538461538462, 906.0, 'TCH <= 1.295\ngini = 0.322\nsamples = 1183\nvalue = [3
68, 1454]\nclass = b'),
 Text(858.4615384615383, 543.5999999999999, 'O_3 <= 38.5\ngini = 0.499\nsamples = 404\nv
alue = [316, 340]\nclass = b'),
 Text(772.6153846153845, 181.19999999999982, 'gini = 0.239\nsamples = 132\nvalue = [28,
174]\nclass = b'),
 Text(944.3076923076923, 181.19999999999982, 'gini = 0.464\nsamples = 272\nvalue = [288,
166]\nclass = a'),
 Text(1201.8461538461538, 543.5999999999999, 'PM25 <= 30.5\ngini = 0.085\nsamples = 779
\nvalue = [52, 1114]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.075\nsamples = 772\nvalue = [45, 1111]\nclas
s = b'),
 Text(1287.6923076923076, 181.19999999999982, 'gini = 0.42\nsamples = 7\nvalue = [7, 3]
\nclass = a'),
 Text(1716.9230769230767, 906.0, 'NO <= 24.5\ngini = 0.233\nsamples = 287\nvalue = [399,
62]\nclass = a'),
 Text(1545.230769230769, 543.5999999999999, 'EBE <= 0.65\ngini = 0.045\nsamples = 215\nv
alue = [337, 8]\nclass = a'),
 Text(1459.3846153846152, 181.19999999999982, 'gini = 0.127\nsamples = 76\nvalue = [109,
8]\nclass = a'),
 Text(1631.0769230769229, 181.19999999999982, 'gini = 0.0\nsamples = 139\nvalue = [228,
0]\nclass = a'),
 Text(1888.6153846153845, 543.5999999999999, 'TOL <= 3.85\ngini = 0.498\nsamples = 72\nv
alue = [62, 54]\nclass = a'),
 Text(1802.7692307692307, 181.19999999999982, 'gini = 0.428\nsamples = 34\nvalue = [18,
40]\nclass = b'),
 Text(1974.4615384615383, 181.19999999999982, 'gini = 0.366\nsamples = 38\nvalue = [44,
14]\nclass = a'),
 Text(3154.846153846154, 1630.8000000000002, 'NO <= 6.5\ngini = 0.496\nsamples = 1510\nv
alue = [1292, 1088]\nclass = a'),
 Text(2532.461538461538, 1268.4, 'TOL <= 3.55\ngini = 0.196\nsamples = 558\nvalue = [96,
777]\nclass = b'),
 Text(2317.846153846154, 906.0, 'NMHC <= 0.195\ngini = 0.061\nsamples = 479\nvalue = [2
4, 736]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'NO_2 <= 25.0\ngini = 0.346\nsamples = 66\nvalue = [24,
84]\nclass = b'),
 Text(2146.153846153846, 181.19999999999982, 'gini = 0.08\nsamples = 46\nvalue = [3, 69]
\nclass = b'),
 Text(2317.846153846154, 181.19999999999982, 'gini = 0.486\nsamples = 20\nvalue = [21, 1
5]\nclass = a'),
 Text(2403.6923076923076, 543.5999999999999, 'gini = 0.0\nsamples = 413\nvalue = [0, 65
```

```
2]\nclass = b'),
 Text(2747.076923076923, 906.0, 'NO <= 1.5\ngini = 0.462\nsamples = 79\nvalue = [72, 41]
\nclass = a'),
 Text(2575.3846153846152, 543.5999999999999, 'NMHC <= 0.225\ngini = 0.403\nsamples = 18
\nvalue = [7, 18]\nclass = b'),
 Text(2489.5384615384614, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [7, 0]\n
class = a'),
 Text(2661.230769230769, 181.19999999999982, 'gini = 0.0\nsamples = 13\nvalue = [0, 18]
\nclass = b'),
 Text(2918.7692307692305, 543.5999999999999, 'NMHC <= 0.265\ngini = 0.386\nsamples = 61
\nvalue = [65, 23]\nclass = a'),
 Text(2832.9230769230767, 181.19999999999982, 'gini = 0.311\nsamples = 53\nvalue = [63,
15]\nclass = a'),
 Text(3004.6153846153843, 181.19999999999982, 'gini = 0.32\nsamples = 8\nvalue = [2, 8]
\nclass = b'),
 Text(3777.230769230769, 1268.4, 'SO_2 <= 9.5\ngini = 0.328\nsamples = 952\nvalue = [119
6, 311]\nclass = a'),
 Text(3433.8461538461534, 906.0, 'PM10 <= 36.5\ngini = 0.478\nsamples = 480\nvalue = [45
4, 296]\nclass = a'),
 Text(3262.1538461538457, 543.5999999999999, 'TCH <= 1.335\ngini = 0.498\nsamples = 285
\nvalue = [239, 212]\nclass = a'),
 Text(3176.307692307692, 181.19999999999982, 'gini = 0.213\nsamples = 134\nvalue = [188,
26]\nclass = a'),
 Text(3347.9999999999995, 181.19999999999982, 'gini = 0.338\nsamples = 151\nvalue = [51,
186]\nclass = b'),
 Text(3605.5384615384614, 543.5999999999999, 'PM25 <= 37.5\ngini = 0.404\nsamples = 195
\nvalue = [215, 84]\nclass = a'),
 Text(3519.6923076923076, 181.19999999999982, 'gini = 0.441\nsamples = 162\nvalue = [16
6, 81]\nclass = a'),
 Text(3691.3846153846152, 181.19999999999982, 'gini = 0.109\nsamples = 33\nvalue = [49,
3]\nclass = a'),
 Text(4120.615384615385, 906.0, 'SO_2 <= 11.5\ngini = 0.039\nsamples = 472\nvalue = [74
2, 15]\nclass = a'),
 Text(3948.9230769230767, 543.5999999999999, 'NMHC <= 0.29\ngini = 0.209\nsamples = 63\n
value = [89, 12]\nclass = a'),
 Text(3863.076923076923, 181.19999999999982, 'gini = 0.023\nsamples = 55\nvalue = [86,
1]\nclass = a'),
 Text(4034.7692307692305, 181.19999999999982, 'gini = 0.337\nsamples = 8\nvalue = [3, 1
1]\nclass = b'),
 Text(4292.307692307692, 543.5999999999999, 'PM25 <= 24.5\ngini = 0.009\nsamples = 409\n
value = [653, 3]\nclass = a'),
 Text(4206.461538461538, 181.19999999999982, 'gini = 0.0\nsamples = 313\nvalue = [508,
0]\nclass = a'),
 Text(4378.153846153846, 181.19999999999982, 'gini = 0.04\nsamples = 96\nvalue = [145,
3]\nclass = a')]
```



Conclusion

Accuracy

linear regression

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

Out[64]: 0.6306803749037257

Ridge regression

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

Out[65]: 0.6289080365797595

Lasso regression

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

Out[66]: 0.36169540854582505

Elastic net regression

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

Out[67]: 0.5368065685385757

Logistic regression

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

Out[68]: 0.9311102968120191

Random forest

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

Out[69]: 0.9647950976348671

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