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_2011.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
0	01-11-2011 01:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN	28079004
1	01-11-2011 01:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7	28079008
2	01-11-2011 01:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2	28079011
3	01-11-2011 01:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN	28079016
4	01-11-2011 01:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN	28079017
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209923	01-09-2011 00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN	28079056
209924	01-09-2011 00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN	28079057
209925	01-09-2011 00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN	28079058

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	station
209926	01-09-2011 00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN	28079059
209927	01-09-2011 00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN	28079060

209928 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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 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	0.4	28079008

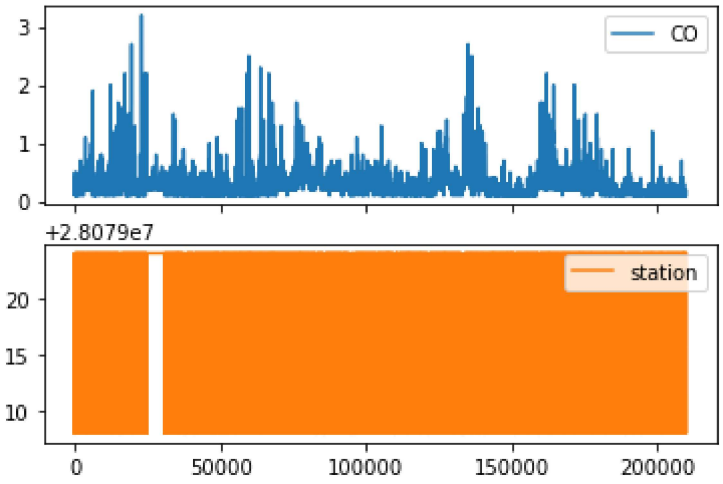
	CO	station
6	0.3	28079024
25	0.3	28079008
30	0.4	28079024
49	0.2	28079008
...	...	...
209862	0.1	28079024
209881	0.1	28079008
209886	0.1	28079024
209905	0.1	28079008
209910	0.1	28079024

16460 rows × 2 columns

Line chart

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

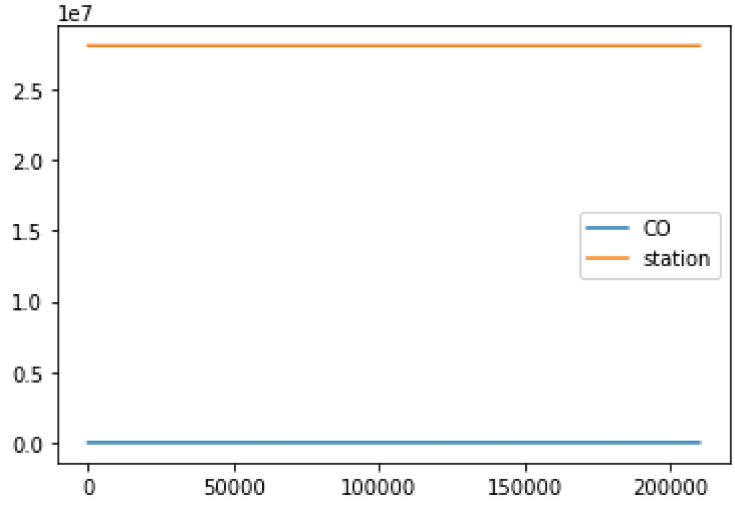
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

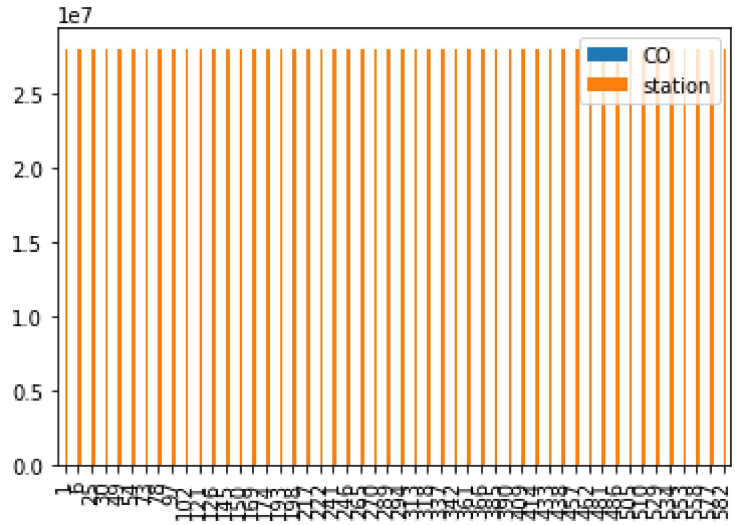


Bar chart

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

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

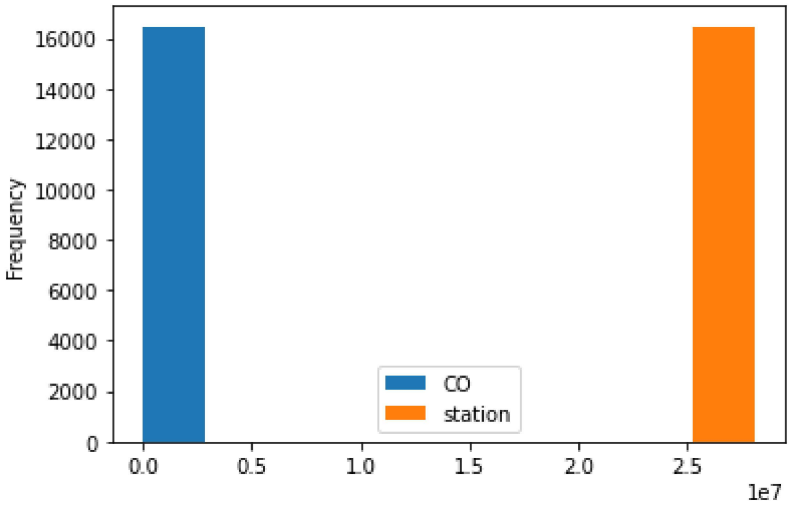
Out[10]: <AxesSubplot:>



Histogram

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

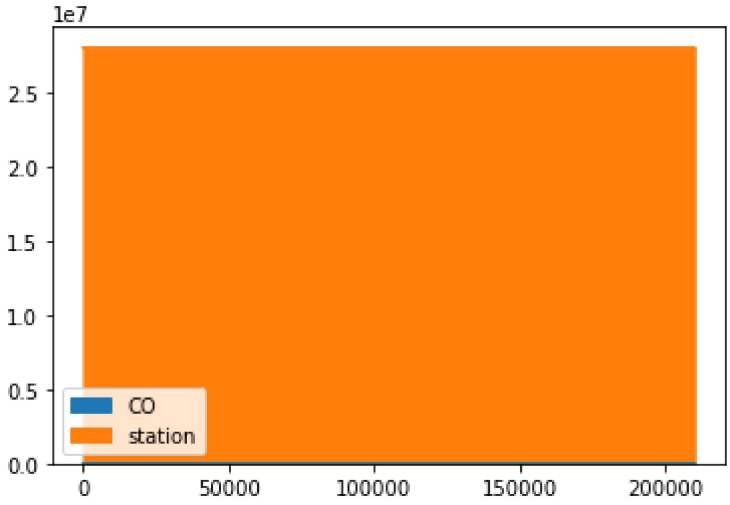
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

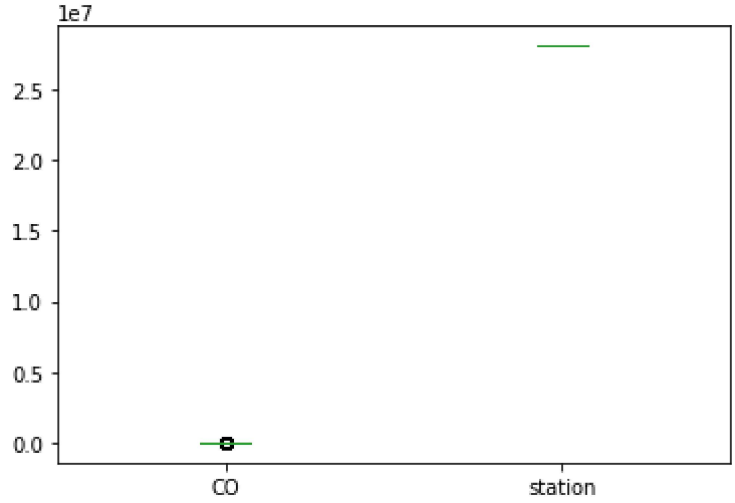
Out[12]: <AxesSubplot:>



Box chart

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

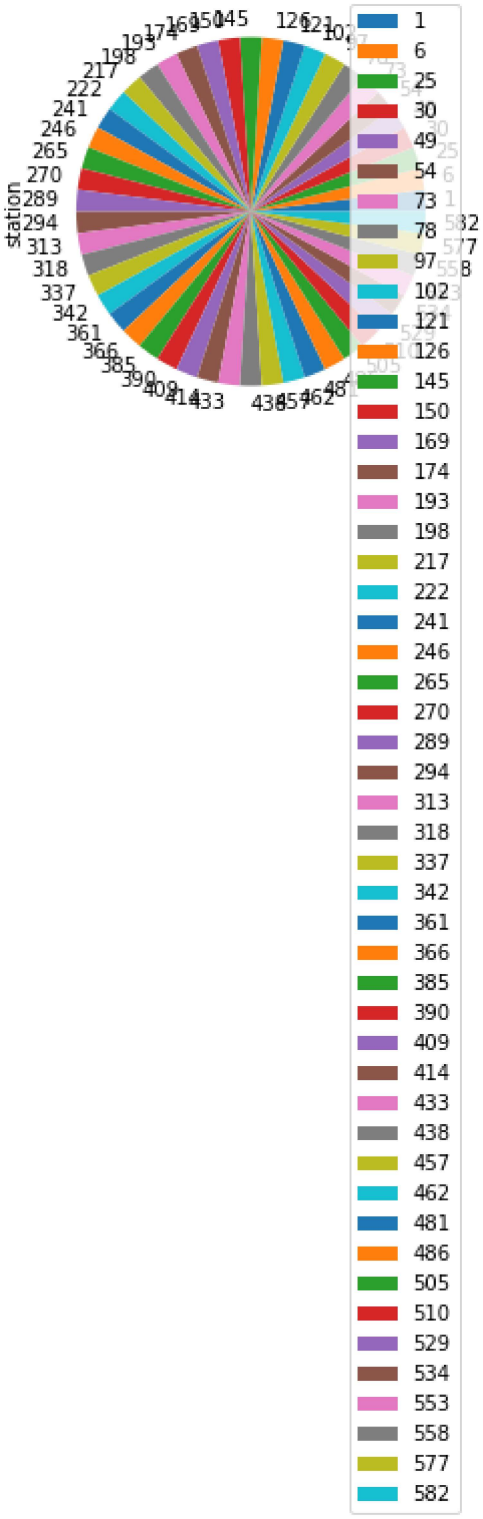
Out[13]: <AxesSubplot:>



Pie chart

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

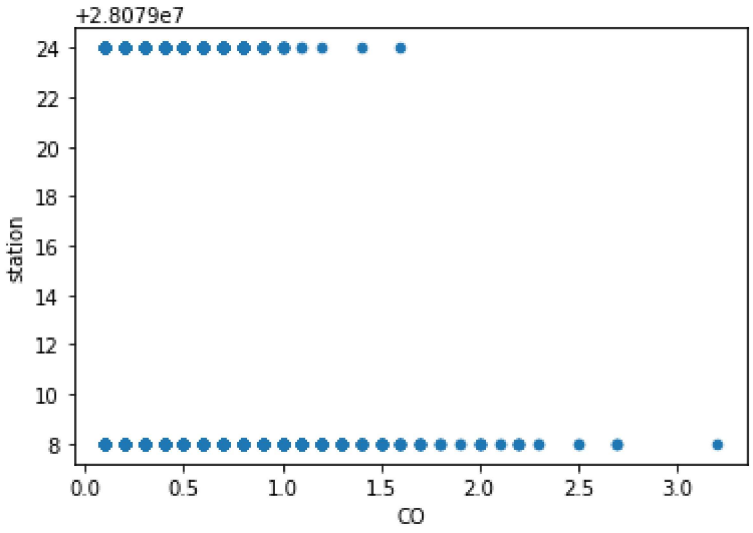
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [19]:
```
df.columns
```

Out[19]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [17]:
```
df.describe()
```

Out[17]:

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000
mean	0.900680	0.277758	1.471871	0.167043	23.671810	44.583961	41.580377
std	0.768892	0.206143	1.051004	0.075068	44.362859	31.569185	28.113385
min	0.100000	0.100000	0.200000	0.010000	1.000000	1.000000	1.000000

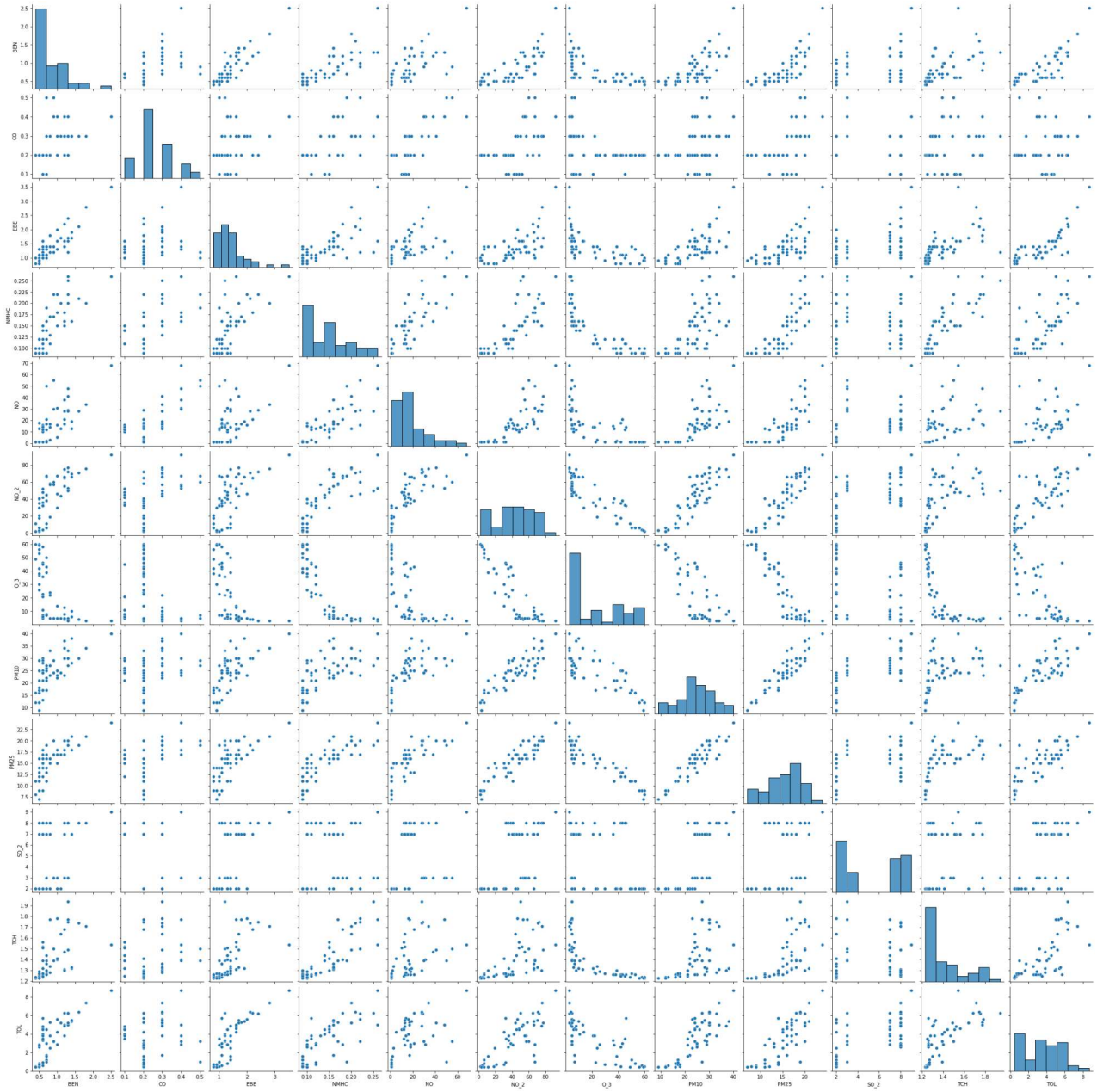
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
25%	0.500000	0.200000	0.800000	0.120000	2.000000	19.000000	17.00000
50%	0.700000	0.200000	1.200000	0.160000	7.000000	40.000000	39.00000
75%	1.100000	0.300000	1.700000	0.200000	25.000000	63.000000	61.00000
max	9.500000	3.200000	12.800000	0.840000	615.000000	289.000000	154.00000

In [21]:
```
df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
```

EDA AND VISUALIZATION

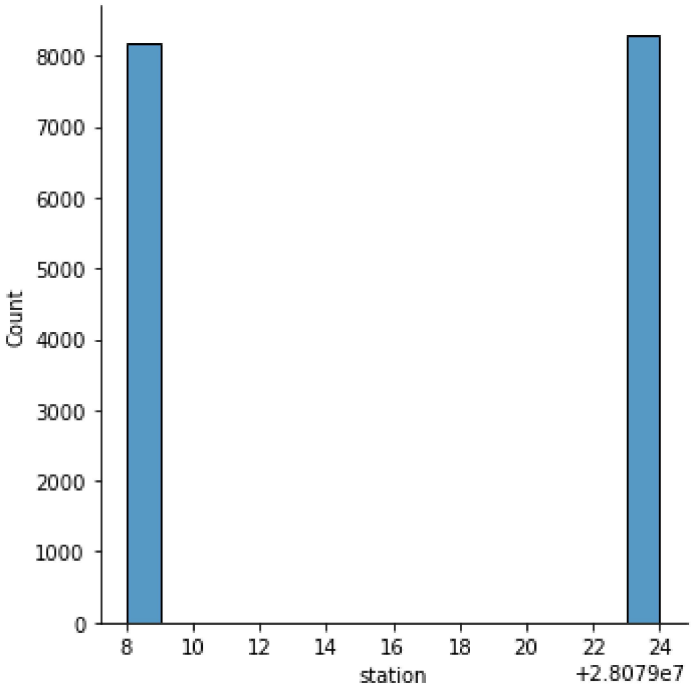
In [22]:
```
sns.pairplot(df1[0:50])
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x23115838c40>



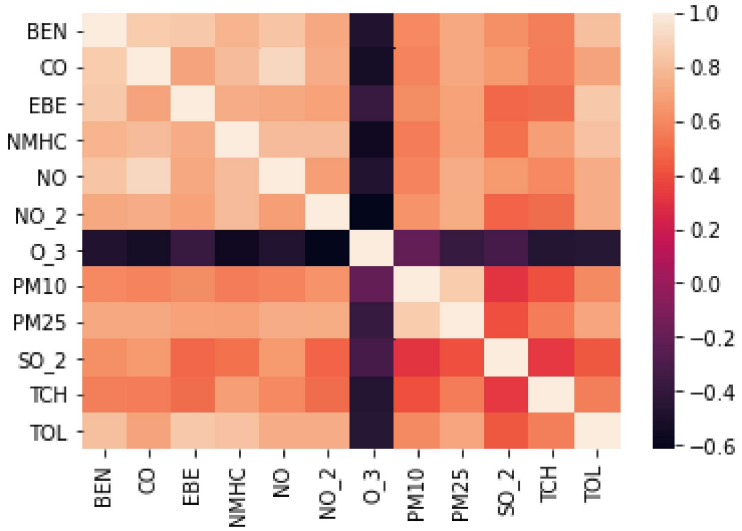
In [29]:
```
sns.displot(df['station'])
```

Out[29]:
```
<seaborn.axisgrid.FacetGrid at 0x2311e831850>
```



In [30]:
```
sns.heatmap(df1.corr())
```

Out[30]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [35]:
```
x=df[[ 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [36]:
```
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 [37]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[37]:
```
LinearRegression()
```

In [38]:
```
lr.intercept_
```

Out[38]:
```
28079014.800954293
```

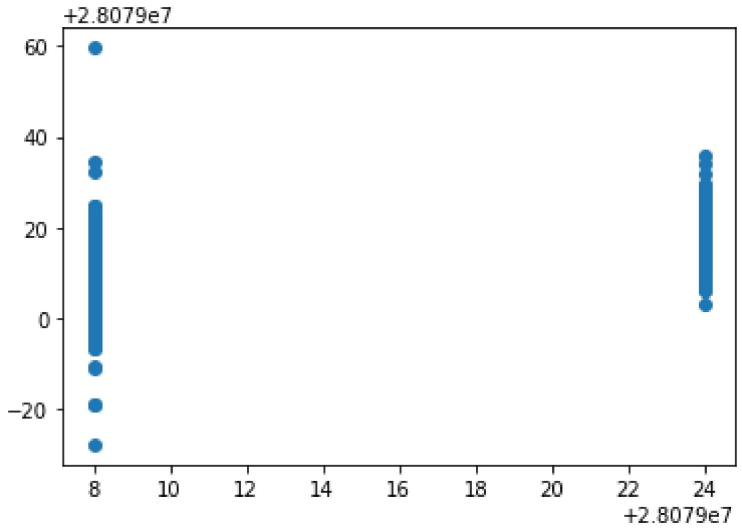
In [39]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[39]:

	Co-efficient
BEN	3.551138
CO	38.920245
EBE	-1.882766
NMHC	-93.502262
NO	-0.040682
NO_2	-0.093570
O_3	-0.014937
PM10	0.018023
PM25	-0.040450
SO_2	-0.454622
TCH	11.260512
TOL	-0.349183

In [40]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[40]:
```
<matplotlib.collections.PathCollection at 0x23120cd8730>
```



ACCURACY

In [41]:
```
lr.score(x_test,y_test)
```

Out[41]: 0.6269067768178905

In [42]:
```
lr.score(x_train,y_train)
```

Out[42]: 0.6273097523182556

Ridge and Lasso

In [43]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [44]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[44]: Ridge(alpha=10)

Accuracy(Ridge)

In [45]:
```
rr.score(x_test,y_test)
```

Out[45]: 0.5952306165233255

In [46]:
```
rr.score(x_train,y_train)
```

Out[46]: 0.5934382500380108

In [47]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[47]:
```
Lasso(alpha=10)
```

In [48]:
```
la.score(x_train,y_train)
```

Out[48]:
```
0.23859740120625683
```

Accuracy(Lasso)

In [49]:
```
la.score(x_test,y_test)
```

Out[49]:
```
0.23474093117386008
```

Elastic Net regression

In [50]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[50]:
```
ElasticNet()
```

In [51]:
```
en.coef_
```

Out[51]:
```
array([ 0.26616533,  0.        , -0.        , -0.        ,  0.05009986,
       -0.13965149, -0.04265966,  0.0339862 ,  0.083345  , -0.169781  ,
        0.        , -0.93929648])
```

In [52]:
```
en.intercept_
```

Out[52]:
```
28079025.066647008
```

In [53]:
```
prediction=en.predict(x_test)
```

In [54]:
```
en.score(x_test,y_test)
```

Out[54]:
```
0.34955662987518976
```

Evaluation Metrics

In [55]:
```
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)))
```

```
5.613010088002944
41.61329076833454
6.450836439434388
```

Logistic Regression

In [72]:
```
from sklearn.linear_model import LogisticRegression
```

In [77]:
```
feature_matrix=df[[ 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [78]:
```
feature_matrix.shape
```

Out[78]: (16460, 12)

In [79]:
```
target_vector.shape
```

Out[79]: (16460,)

In [80]:
```
from sklearn.preprocessing import StandardScaler
```

In [81]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [82]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[82]: LogisticRegression(max_iter=10000)

In [87]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12]]
```

In [88]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [89]:
```
logr.classes_
```

Out[89]: array([28079008, 28079024], dtype=int64)

In [90]:
```
logr.score(fs,target_vector)
```

Out[90]: 0.9262454434993924

In [91]:
```
logr.predict_proba(observation)[0][0]
```

Out[91]: 0.9999999999999999

In [92]:
```
logr.predict_proba(observation)
```

Out[92]:
```
array([[1.00000000e+00, 9.78522297e-17]])
```

Random Forest

In [93]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [94]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[94]:
```
RandomForestClassifier()
```

In [95]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [96]:
```
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[96]:
```
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 [97]:
```
grid_search.best_score_
```

Out[97]: 0.9367297344211074

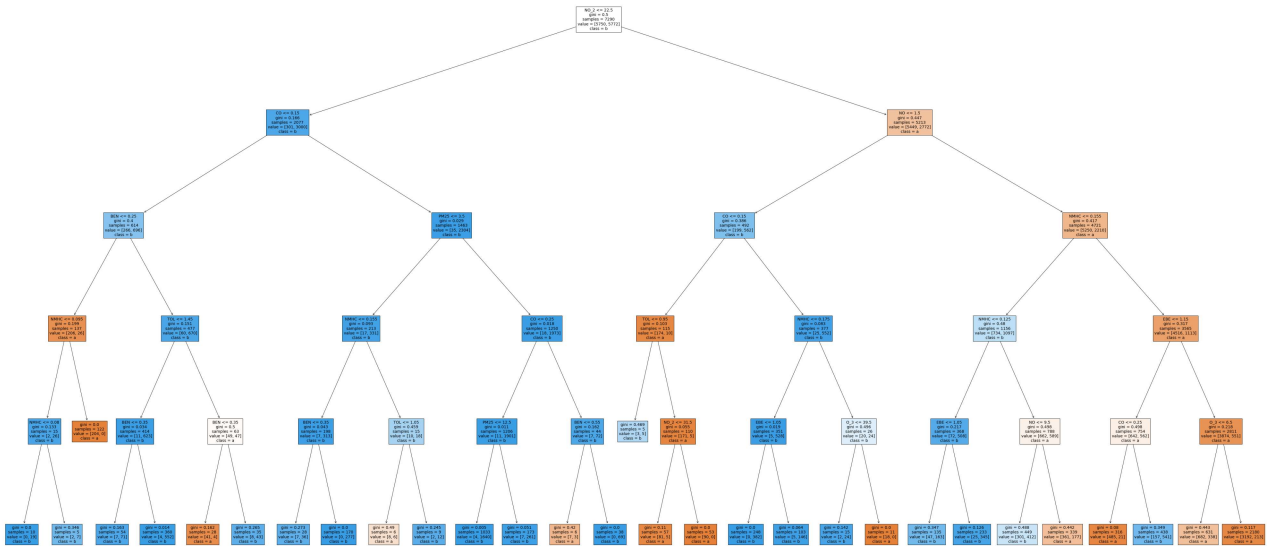
In [98]:
```
rfc_best=grid_search.best_estimator_
```

In [99]:
```
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[99]:
```
[Text(2112.428571428571, 1993.2, 'NO_2 <= 22.5\ngini = 0.5\nsamples = 7290\nvalue = [575
0, 5772]\nclass = b'),
 Text(1016.3571428571428, 1630.8000000000002, 'CO <= 0.15\ngini = 0.166\nsamples = 2077
\nvalue = [301, 3000]\nclass = b'),
 Text(438.4285714285714, 1268.4, 'BEN <= 0.25\ngini = 0.4\nsamples = 614\nvalue = [266,
696]\nclass = b'),
 Text(239.1428571428571, 906.0, 'NMHC <= 0.095\ngini = 0.199\nsamples = 137\nvalue = [20
6, 26]\nclass = a'),
 Text(159.42857142857142, 543.5999999999999, 'NMHC <= 0.08\ngini = 0.133\nsamples = 15\n
value = [2, 26]\nclass = b'),
 Text(79.71428571428571, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalue = [0, 19]
\nclass = b'),
 Text(239.1428571428571, 181.19999999999982, 'gini = 0.346\nsamples = 5\nvalue = [2, 7]
\nclass = b'),
 Text(318.85714285714283, 543.5999999999999, 'gini = 0.0\nsamples = 122\nvalue = [204,
0]\nclass = a'),
 Text(637.7142857142857, 906.0, 'TOL <= 1.45\ngini = 0.151\nsamples = 477\nvalue = [60,
670]\nclass = b'),
 Text(478.2857142857142, 543.5999999999999, 'BEN <= 0.35\ngini = 0.034\nsamples = 414\nv
alue = [11, 623]\nclass = b'),
 Text(398.57142857142856, 181.19999999999982, 'gini = 0.163\nsamples = 54\nvalue = [7, 7
1]\nclass = b'),
 Text(558.0, 181.19999999999982, 'gini = 0.014\nsamples = 360\nvalue = [4, 552]\nclass =
b'),
 Text(797.1428571428571, 543.5999999999999, 'BEN <= 0.35\ngini = 0.5\nsamples = 63\nvalu
e = [49, 47]\nclass = a'),
 Text(717.4285714285713, 181.19999999999982, 'gini = 0.162\nsamples = 28\nvalue = [41,
4]\nclass = a'),
 Text(876.8571428571428, 181.19999999999982, 'gini = 0.265\nsamples = 35\nvalue = [8, 4
3]\nclass = b'),
 Text(1594.2857142857142, 1268.4, 'PM25 <= 3.5\ngini = 0.029\nsamples = 1463\nvalue = [3
5, 2304]\nclass = b'),
 Text(1275.4285714285713, 906.0, 'NMHC <= 0.155\ngini = 0.093\nsamples = 213\nvalue = [1
7, 331]\nclass = b'),
 Text(1116.0, 543.5999999999999, 'BEN <= 0.35\ngini = 0.043\nsamples = 198\nvalue = [7,
313]\nclass = b'),
 Text(1036.2857142857142, 181.19999999999982, 'gini = 0.273\nsamples = 28\nvalue = [7, 3
6]\nclass = b'),
 Text(1195.7142857142856, 181.19999999999982, 'gini = 0.0\nsamples = 170\nvalue = [0, 27
7]\nclass = b'),
 Text(1434.8571428571427, 543.5999999999999, 'TOL <= 1.05\ngini = 0.459\nsamples = 15\nv
alue = [10, 18]\nclass = b'),
 Text(1355.142857142857, 181.19999999999982, 'gini = 0.49\nsamples = 6\nvalue = [8, 6]\n
class = a'),
 Text(1514.5714285714284, 181.19999999999982, 'gini = 0.245\nsamples = 9\nvalue = [2, 1
2]\nclass = b'),
 Text(1913.1428571428569, 906.0, 'CO <= 0.25\ngini = 0.018\nsamples = 1250\nvalue = [18,
1973]\nclass = b'),
 Text(1753.7142857142856, 543.5999999999999, 'PM25 <= 12.5\ngini = 0.011\nsamples = 1206
\nvalue = [11, 1901]\nclass = b'),
 Text(1673.9999999999998, 181.19999999999982, 'gini = 0.005\nsamples = 1033\nvalue = [4,
1640]\nclass = b'),
 Text(1833.4285714285713, 181.19999999999982, 'gini = 0.051\nsamples = 173\nvalue = [7,
261]\nclass = b'),
 Text(2072.5714285714284, 543.5999999999999, 'BEN <= 0.55\ngini = 0.162\nsamples = 44\nv
alue = [7, 72]\nclass = b'),
 Text(1992.8571428571427, 181.19999999999982, 'gini = 0.42\nsamples = 6\nvalue = [7, 3]
\nclass = a'),
 Text(2152.285714285714, 181.19999999999982, 'gini = 0.0\nsamples = 38\nvalue = [0, 69]
\nclass = b'),
 Text(3208.4999999999995, 1630.8000000000002, 'NO <= 1.5\ngini = 0.447\nsamples = 5213\n
value = [5449, 2772]\nclass = a'),
 Text(2590.7142857142853, 1268.4, 'CO <= 0.15\ngini = 0.386\nsamples = 492\nvalue = [19
9, 562]\nclass = b'),
 Text(2311.7142857142853, 906.0, 'TOL <= 0.95\ngini = 0.103\nsamples = 115\nvalue = [17
```

```
4, 10]\nclass = a'),
 Text(2232.0, 543.5999999999999, 'gini = 0.469\nsamples = 5\nvalue = [3, 5]\nclass =
b'),
 Text(2391.428571428571, 543.5999999999999, 'NO_2 <= 31.5\ngini = 0.055\nsamples = 110\n
value = [171, 5]\nclass = a'),
 Text(2311.7142857142853, 181.19999999999982, 'gini = 0.11\nsamples = 57\nvalue = [81,
5]\nclass = a'),
 Text(2471.142857142857, 181.19999999999982, 'gini = 0.0\nsamples = 53\nvalue = [90, 0]
\nclass = a'),
 Text(2869.7142857142853, 906.0, 'NMHC <= 0.175\ngini = 0.083\nsamples = 377\nvalue = [2
5, 552]\nclass = b'),
 Text(2710.285714285714, 543.5999999999999, 'EBE <= 1.05\ngini = 0.019\nsamples = 351\nv
alue = [5, 528]\nclass = b'),
 Text(2630.5714285714284, 181.19999999999982, 'gini = 0.0\nsamples = 248\nvalue = [0, 38
2]\nclass = b'),
 Text(2790.0, 181.19999999999982, 'gini = 0.064\nsamples = 103\nvalue = [5, 146]\nclass
= b'),
 Text(3029.142857142857, 543.5999999999999, 'O_3 <= 39.5\ngini = 0.496\nsamples = 26\nva
lue = [20, 24]\nclass = b'),
 Text(2949.428571428571, 181.19999999999982, 'gini = 0.142\nsamples = 15\nvalue = [2, 2
4]\nclass = b'),
 Text(3108.8571428571427, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [18, 0]
\nclass = a'),
 Text(3826.2857142857138, 1268.4, 'NMHC <= 0.155\ngini = 0.417\nsamples = 4721\nvalue =
[5250, 2210]\nclass = a'),
 Text(3507.428571428571, 906.0, 'NMHC <= 0.125\ngini = 0.48\nsamples = 1156\nvalue = [73
4, 1097]\nclass = b'),
 Text(3347.9999999999995, 543.5999999999999, 'EBE <= 1.05\ngini = 0.217\nsamples = 368\n
value = [72, 508]\nclass = b'),
 Text(3268.285714285714, 181.19999999999982, 'gini = 0.347\nsamples = 135\nvalue = [47,
163]\nclass = b'),
 Text(3427.7142857142853, 181.19999999999982, 'gini = 0.126\nsamples = 233\nvalue = [25,
345]\nclass = b'),
 Text(3666.8571428571427, 543.5999999999999, 'NO <= 9.5\ngini = 0.498\nsamples = 788\nva
lue = [662, 589]\nclass = a'),
 Text(3587.142857142857, 181.19999999999982, 'gini = 0.488\nsamples = 449\nvalue = [301,
412]\nclass = b'),
 Text(3746.5714285714284, 181.19999999999982, 'gini = 0.442\nsamples = 339\nvalue = [36
1, 177]\nclass = a'),
 Text(4145.142857142857, 906.0, 'EBE <= 1.15\ngini = 0.317\nsamples = 3565\nvalue = [451
6, 1113]\nclass = a'),
 Text(3985.7142857142853, 543.5999999999999, 'CO <= 0.25\ngini = 0.498\nsamples = 754\nv
alue = [642, 562]\nclass = a'),
 Text(3905.9999999999995, 181.19999999999982, 'gini = 0.08\nsamples = 316\nvalue = [485,
21]\nclass = a'),
 Text(4065.428571428571, 181.19999999999982, 'gini = 0.349\nsamples = 438\nvalue = [157,
541]\nclass = b'),
 Text(4304.571428571428, 543.5999999999999, 'O_3 <= 6.5\ngini = 0.218\nsamples = 2811\nv
alue = [3874, 551]\nclass = a'),
 Text(4224.857142857142, 181.19999999999982, 'gini = 0.443\nsamples = 631\nvalue = [682,
338]\nclass = a'),
 Text(4384.285714285714, 181.19999999999982, 'gini = 0.117\nsamples = 2180\nvalue = [319
2, 213]\nclass = a')]
```



Conclusion

Accuracy

linear regression

In [100…

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

Out[100…

```
0.6269067768178905
```

Ridge regression

In [101…

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

Out[101…

```
0.5952306165233255
```

Lasso regression

In [102…

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

Out[102…

```
0.23474093117386008
```

Elastic net regression

In [103…

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

Out[103… 0.34955662987518976

Logistic regression

In [104…
```
logr.score(fs,target_vector)
```

Out[104… 0.9262454434993924

Random forest

In [105…
```
grid_search.best_score_
```

Out[105… 0.9367297344211074

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