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_2002.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY	
0	01-04-2002 01:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.990002	NaN	21
1	01-04-2002 01:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.85	20.980000	2.53	11
2	01-04-2002 01:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.440001	NaN	13
3	01-04-2002 01:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.180000	NaN	16
4	01-04-2002 01:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.330002	NaN	15
...	...	...	...	...	...	...	...	...	...	...	...	...	...
217291	01-11-2002 00:00	4.16	1.14	NaN	NaN	NaN	81.080002	265.700012	NaN	7.21	36.750000	NaN	13
217292	01-11-2002 00:00	3.67	1.73	2.89	NaN	0.38	113.900002	373.100006	NaN	5.66	63.389999	NaN	15
217293	01-11-2002 00:00	1.37	0.58	1.17	2.37	0.15	65.389999	107.699997	1.30	9.11	9.640000	0.94	5

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY	
217294	01-11-2002 00:00	4.51	0.91	4.83	10.99	NaN	149.800003	202.199997	1.00	5.75	NaN	5.52	24
217295	01-11-2002 00:00	3.11	1.17	3.00	7.77	0.26	80.110001	180.300003	2.25	7.38	29.240000	3.35	12

217296 rows × 16 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.dropna()
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

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

Out[6]:

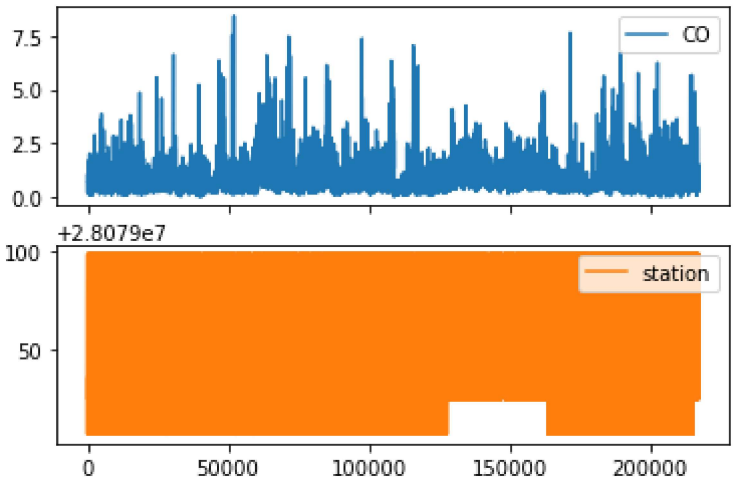
	CO	station
1	0.71	28079035
5	0.72	28079006
22	0.80	28079024
24	1.04	28079099
26	0.53	28079035
...	...	...
217269	0.28	28079024
217271	1.30	28079099
217273	0.97	28079035
217293	0.58	28079024
217295	1.17	28079099

32381 rows × 2 columns

Line chart

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

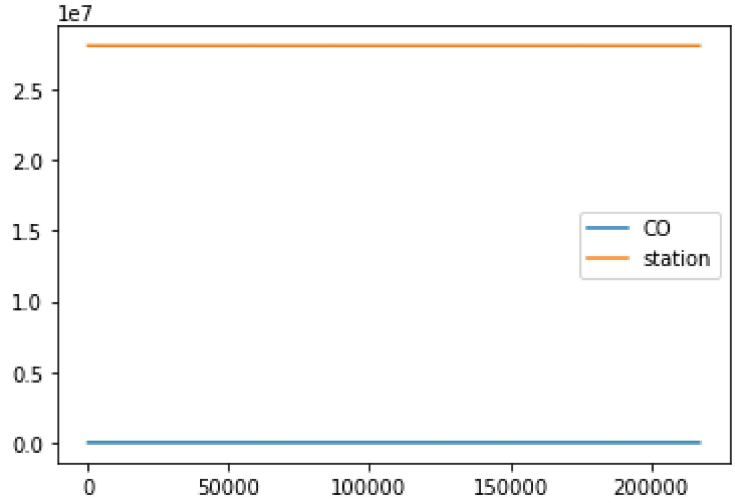
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

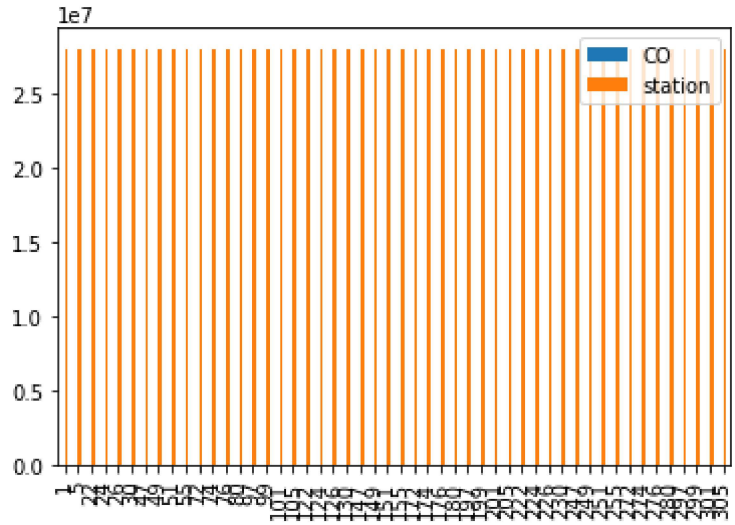


Bar chart

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

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

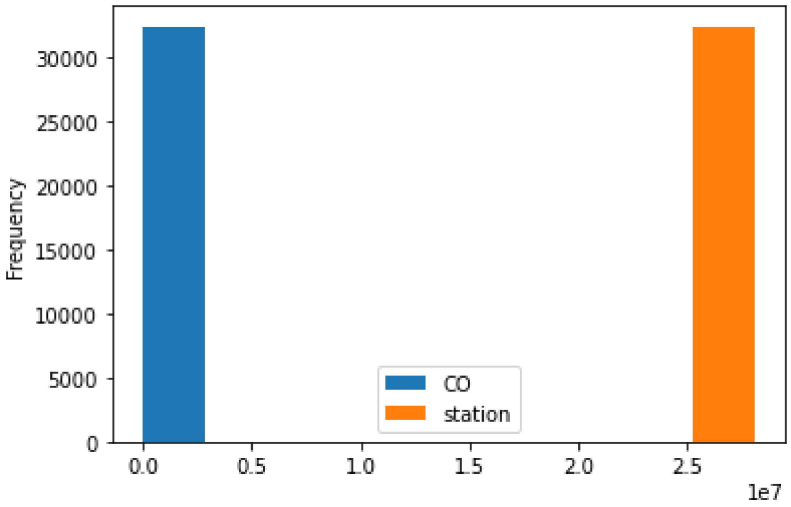
Out[10]: <AxesSubplot:>



Histogram

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

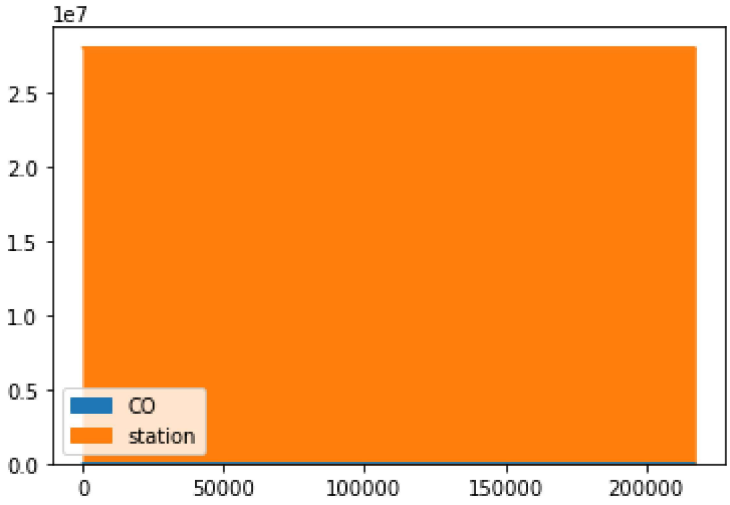
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

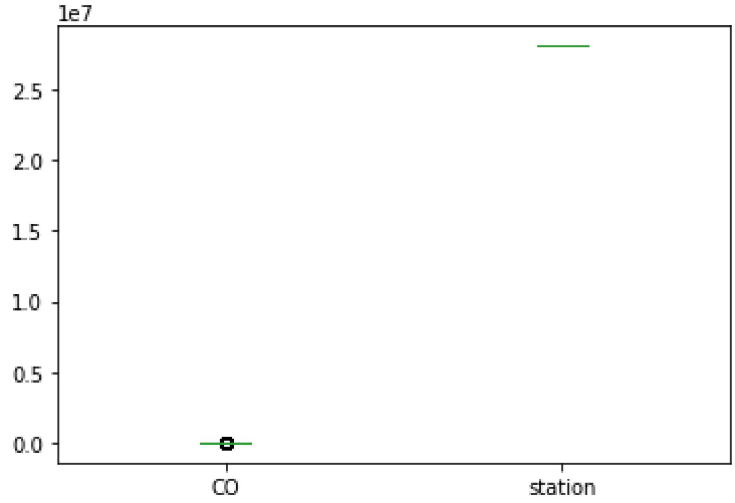
Out[12]: <AxesSubplot:>



Box chart

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

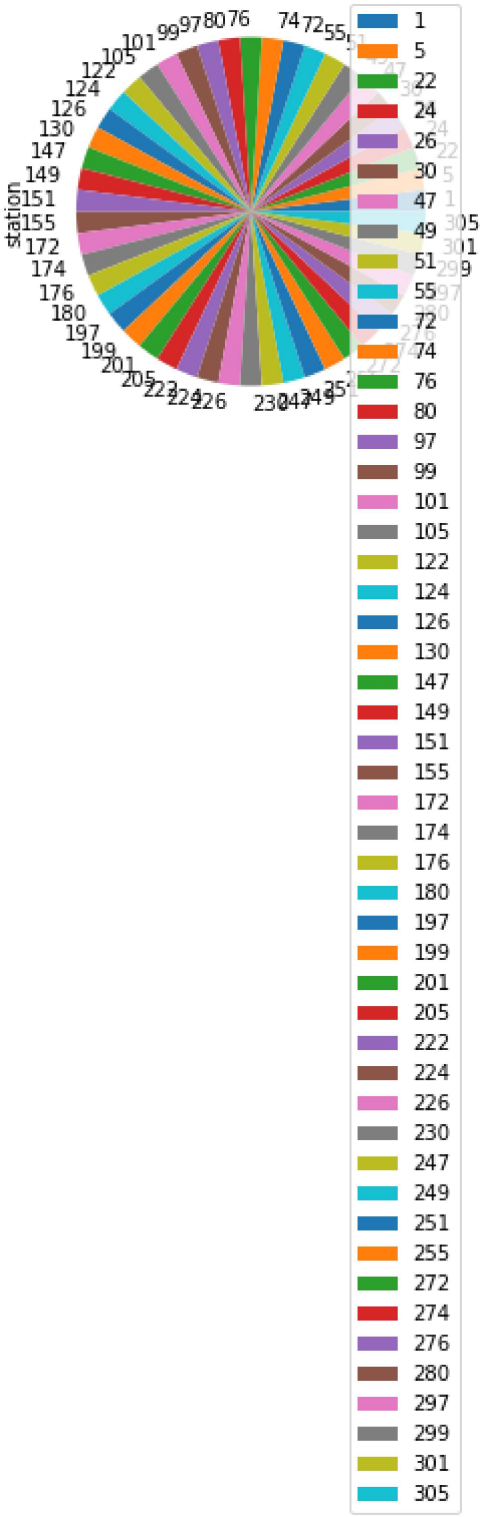
Out[13]: <AxesSubplot:>



Pie chart

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

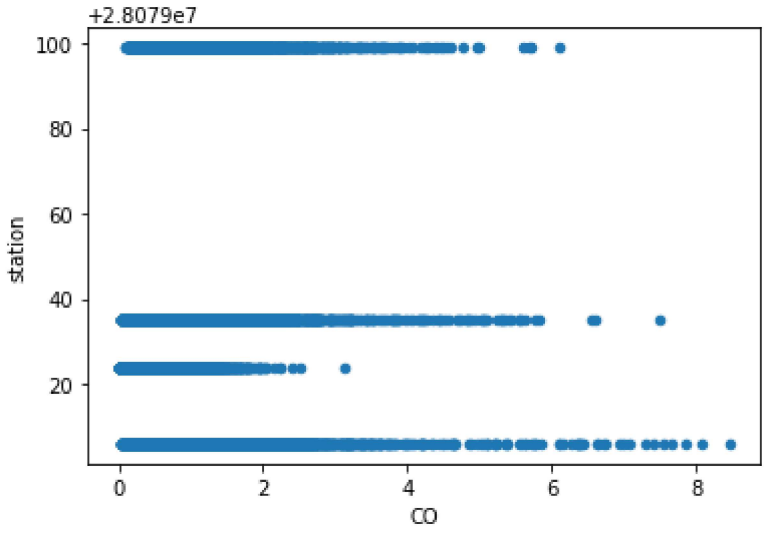
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000	32381.00000
mean	2.479155	0.787323	2.914004	7.013636	0.155827	58.936796	126.00934
std	2.280959	0.610810	2.667881	6.774365	0.135731	31.472733	114.03507
min	0.180000	0.000000	0.180000	0.190000	0.000000	0.890000	1.71000
25%	0.970000	0.420000	1.140000	2.420000	0.080000	35.660000	48.72000
50%	1.840000	0.620000	2.130000	5.140000	0.130000	57.160000	96.83000
75%	3.250000	0.980000	3.830000	9.420000	0.200000	78.769997	167.50000

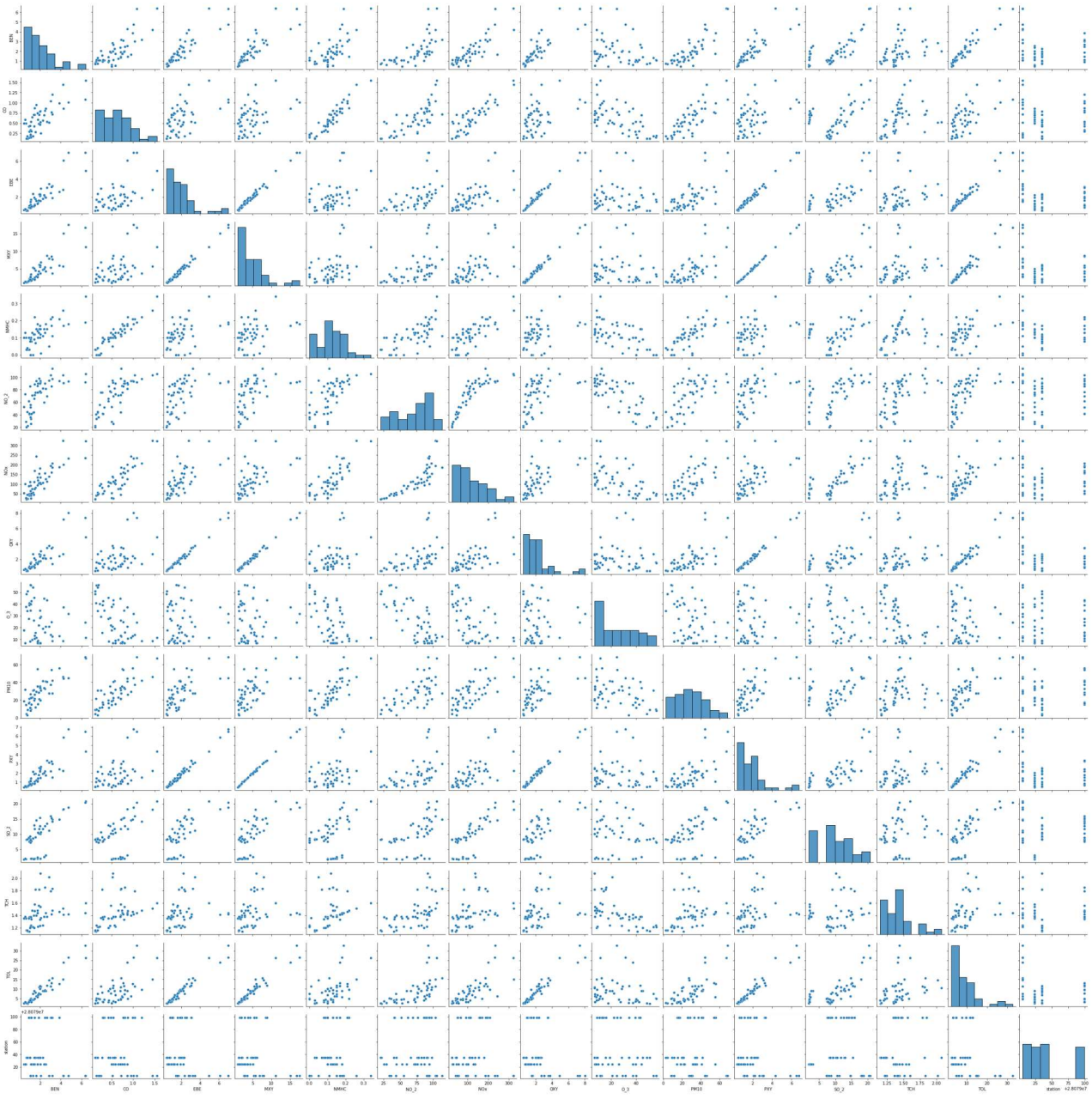
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
max	32.660000	8.460000	41.740002	99.879997	2.700000	263.600006	1336.00000

In [18]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

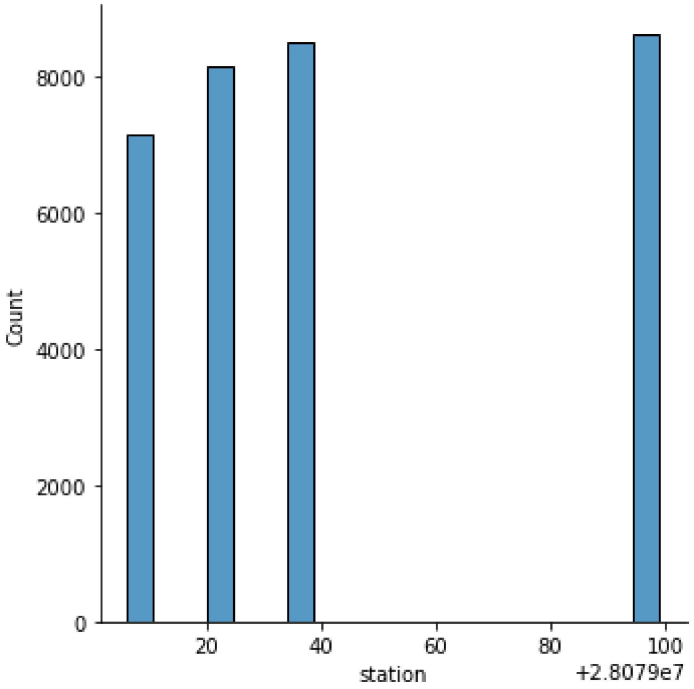
In [19]:
```
sns.pairplot(df1[0:50])
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x20020be2610>



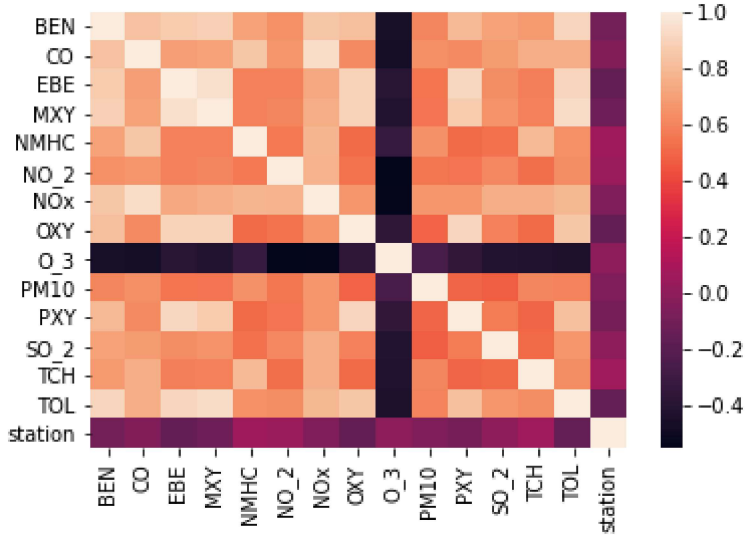
In [63]:
```
sns.displot(df1['station'])
```

Out[63]: <seaborn.axisgrid.FacetGrid at 0x2003049f040>



In [21]:
```
sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [23]:
```
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 [24]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[24]:
```
LinearRegression()
```

In [25]:
```
lr.intercept_
```

Out[25]:
```
28078988.114380907
```

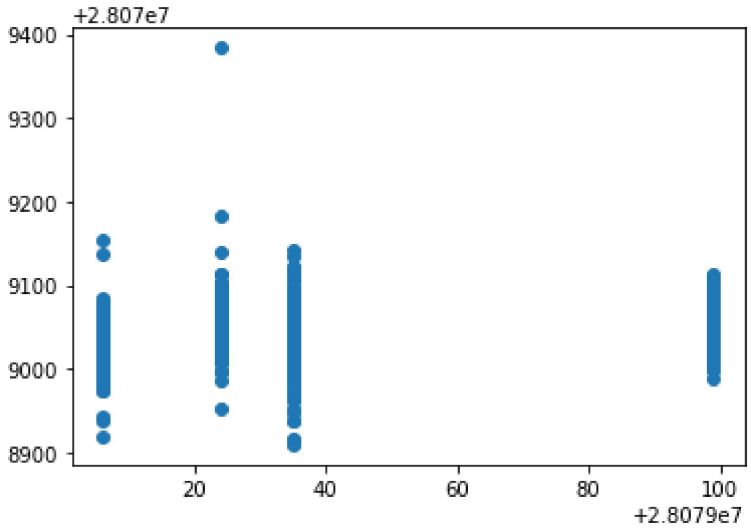
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.401736
CO	-15.128328
EBE	-10.835111
MXY	3.702200
NMHC	90.095239
NO_2	0.260170
NOx	-0.095151
OXY	-4.898513
O_3	-0.028787
PM10	-0.134892
PXY	7.420749
SO_2	0.584795
TCH	43.407324
TOL	-1.284777

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]:
```
<matplotlib.collections.PathCollection at 0x20030447ac0>
```



ACCURACY

In [28]:
```
lr.score(x_test,y_test)
```

Out[28]: 0.20586196841799131

In [29]:
```
lr.score(x_train,y_train)
```

Out[29]: 0.195097278017629

Ridge and Lasso

In [30]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [31]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[31]: Ridge(alpha=10)

Accuracy(Ridge)

In [32]:
```
rr.score(x_test,y_test)
```

Out[32]: 0.20662440552663042

In [33]:
```
rr.score(x_train,y_train)
```

Out[33]: 0.19484052262692186

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]:
```
Lasso(alpha=10)
```

In [35]:
```
la.score(x_train,y_train)
```

Out[35]:
```
0.05762328122195848
```

Accuracy(Lasso)

In [36]:
```
la.score(x_test,y_test)
```

Out[36]:
```
0.05638980897063961
```

Elastic Net regression

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

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

In [38]:
```
en.coef_
```

Out[38]:
```
array([ 7.06745495e-01,  0.00000000e+00, -2.75279598e+00,  1.33954985e+00,
        2.12263144e-01,  2.32243949e-01, -2.79195288e-02, -2.32372478e+00,
       -2.12314451e-02,  5.92239041e-04,  2.13550128e+00,  3.84179697e-01,
        1.05475468e+00, -1.02105861e+00])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
28079038.464570597
```

In [40]:
```
prediction=en.predict(x_test)
```

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

Out[41]:
```
0.10178683023266588
```

Evaluation Metrics

In [42]:
```
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)))
```

```
28.74064766263155
1121.541358877462
33.48942159663947
```

Logistic Regression

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:
```
feature_matrix.shape
```

Out[45]: (32381, 14)

In [46]:
```
target_vector.shape
```

Out[46]: (32381,)

In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [49]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[49]: LogisticRegression(max_iter=10000)

In [50]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [51]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079035]
```

In [52]:
```
logr.classes_
```

Out[52]: array([28079006, 28079024, 28079035, 28079099], dtype=int64)

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]:
```
0.8481516938945678
```

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]:
```
2.555637688425837e-10
```

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

Out[55]:
```
array([[2.55563769e-10, 3.47676748e-71, 1.00000000e+00, 1.44730625e-13]])
```

Random Forest

In [56]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[57]:
```
RandomForestClassifier()
```

In [58]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [59]:
```
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[59]:
```
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 [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.7760081178858202
```

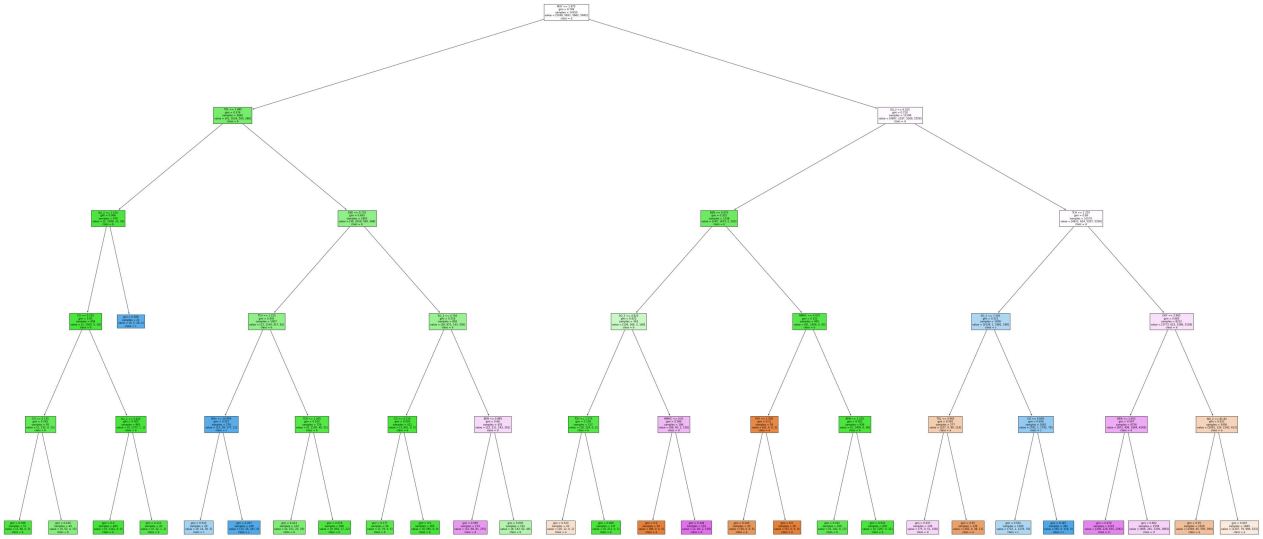
In [61]:
```
rfc_best=grid_search.best_estimator_
```

In [62]:
```
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[62]:
```
[Text(2012.7857142857142, 1993.2, 'MXY <= 1.875\ngini = 0.749\nsamples = 14350\nvalue =
[5038, 5821, 5862, 5945]\nclass = d'),
 Text(836.9999999999999, 1630.8000000000002, 'TOL <= 1.485\ngini = 0.374\nsamples = 2842
\nvalue = [41, 3524, 593, 386]\nclass = b'),
 Text(398.57142857142856, 1268.4, 'SO_2 <= 7.575\ngini = 0.066\nsamples = 979\nvalue =
[2, 1508, 33, 18]\nclass = b'),
 Text(318.85714285714283, 906.0, 'CO <= 0.235\ngini = 0.03\nsamples = 958\nvalue = [2, 1
505, 5, 16]\nclass = b'),
 Text(159.42857142857142, 543.5999999999999, 'CO <= 0.155\ngini = 0.245\nsamples = 95\nv
alue = [2, 132, 4, 15]\nclass = b'),
 Text(79.71428571428571, 181.19999999999982, 'gini = 0.048\nsamples = 51\nvalue = [2, 8
0, 0, 0]\nclass = b'),
 Text(239.1428571428571, 181.19999999999982, 'gini = 0.416\nsamples = 44\nvalue = [0, 5
2, 4, 15]\nclass = b'),
 Text(478.2857142857142, 543.5999999999999, 'SO_2 <= 5.925\ngini = 0.003\nsamples = 863
\nvalue = [0, 1373, 1, 1]\nclass = b'),
 Text(398.57142857142856, 181.19999999999982, 'gini = 0.0\nsamples = 843\nvalue = [0, 13
41, 0, 0]\nclass = b'),
 Text(558.0, 181.19999999999982, 'gini = 0.112\nsamples = 20\nvalue = [0, 32, 1, 1]\ncla
ss = b'),
 Text(478.2857142857142, 906.0, 'gini = 0.268\nsamples = 21\nvalue = [0, 3, 28, 2]\nclas
s = c'),
 Text(1275.4285714285713, 1268.4, 'EBE <= 0.735\ngini = 0.493\nsamples = 1863\nvalue =
[39, 2016, 560, 368]\nclass = b'),
 Text(956.5714285714284, 906.0, 'TCH <= 1.225\ngini = 0.445\nsamples = 1007\nvalue = [1
3, 1144, 417, 62]\nclass = b'),
 Text(797.1428571428571, 543.5999999999999, 'NOx <= 14.895\ngini = 0.257\nsamples = 278
\nvalue = [13, 39, 377, 11]\nclass = c'),
 Text(717.4285714285713, 181.19999999999982, 'gini = 0.533\nsamples = 29\nvalue = [0, 1
4, 30, 5]\nclass = c'),
 Text(876.8571428571428, 181.19999999999982, 'gini = 0.207\nsamples = 249\nvalue = [13,
25, 347, 6]\nclass = c'),
 Text(1116.0, 543.5999999999999, 'TCH <= 1.265\ngini = 0.143\nsamples = 729\nvalue = [0,
1105, 40, 51]\nclass = b'),
 Text(1036.2857142857142, 181.19999999999982, 'gini = 0.413\nsamples = 123\nvalue = [0,
151, 23, 29]\nclass = b'),
 Text(1195.7142857142856, 181.19999999999982, 'gini = 0.076\nsamples = 606\nvalue = [0,
954, 17, 22]\nclass = b'),
 Text(1594.2857142857142, 906.0, 'SO_2 <= 4.765\ngini = 0.518\nsamples = 856\nvalue = [2
6, 872, 143, 306]\nclass = b'),
 Text(1434.8571428571427, 543.5999999999999, 'CO <= 0.215\ngini = 0.024\nsamples = 421\n
value = [3, 661, 0, 5]\nclass = b'),
 Text(1355.142857142857, 181.19999999999982, 'gini = 0.177\nsamples = 56\nvalue = [3, 7
6, 0, 5]\nclass = b'),
 Text(1514.5714285714284, 181.19999999999982, 'gini = 0.0\nsamples = 365\nvalue = [0, 58
5, 0, 0]\nclass = b'),
 Text(1753.7142857142856, 543.5999999999999, 'BEN <= 0.885\ngini = 0.66\nsamples = 435\n
value = [23, 211, 143, 301]\nclass = d'),
 Text(1673.9999999999998, 181.19999999999982, 'gini = 0.565\nsamples = 274\nvalue = [15,
68, 81, 255]\nclass = d'),
 Text(1833.4285714285713, 181.19999999999982, 'gini = 0.605\nsamples = 161\nvalue = [8,
143, 62, 46]\nclass = b'),
 Text(3188.5714285714284, 1630.8000000000002, 'SO_2 <= 6.015\ngini = 0.729\nsamples = 11
508\nvalue = [4997, 2297, 5269, 5559]\nclass = d'),
 Text(2550.8571428571427, 1268.4, 'BEN <= 0.975\ngini = 0.325\nsamples = 1338\nvalue =
[185, 1673, 2, 203]\nclass = b'),
 Text(2232.0, 906.0, 'SO_2 <= 4.825\ngini = 0.622\nsamples = 343\nvalue = [104, 264, 2,
160]\nclass = b'),
 Text(2072.5714285714284, 543.5999999999999, 'TCH <= 1.275\ngini = 0.139\nsamples = 157
\nvalue = [16, 224, 0, 2]\nclass = b'),
 Text(1992.8571428571427, 181.19999999999982, 'gini = 0.523\nsamples = 20\nvalue = [16,
12, 0, 1]\nclass = a'),
```

```
 Text(2152.285714285714, 181.19999999999982, 'gini = 0.009\nsamples = 137\nvalue = [0, 2
12, 0, 1]\nclass = b'),
 Text(2391.428571428571, 543.5999999999999, 'NMHC <= 0.01\ngini = 0.586\nsamples = 186\n
value = [88, 40, 2, 158]\nclass = d'),
 Text(2311.7142857142853, 181.19999999999982, 'gini = 0.0\nsamples = 54\nvalue = [86, 0,
0, 0]\nclass = a'),
 Text(2471.142857142857, 181.19999999999982, 'gini = 0.349\nsamples = 132\nvalue = [2, 4
0, 2, 158]\nclass = d'),
 Text(2869.7142857142853, 906.0, 'NMHC <= 0.055\ngini = 0.152\nsamples = 995\nvalue = [8
1, 1409, 0, 43]\nclass = b'),
 Text(2710.285714285714, 543.5999999999999, 'PXY <= 1.595\ngini = 0.11\nsamples = 59\nva
lue = [81, 0, 0, 5]\nclass = a'),
 Text(2630.5714285714284, 181.19999999999982, 'gini = 0.245\nsamples = 20\nvalue = [30,
0, 0, 5]\nclass = a'),
 Text(2790.0, 181.19999999999982, 'gini = 0.0\nsamples = 39\nvalue = [51, 0, 0, 0]\nclas
s = a'),
 Text(3029.142857142857, 543.5999999999999, 'BEN <= 1.105\ngini = 0.051\nsamples = 936\n
value = [0, 1409, 0, 38]\nclass = b'),
 Text(2949.428571428571, 181.19999999999982, 'gini = 0.191\nsamples = 106\nvalue = [0, 1
42, 0, 17]\nclass = b'),
 Text(3108.8571428571427, 181.19999999999982, 'gini = 0.032\nsamples = 830\nvalue = [0,
1267, 0, 21]\nclass = b'),
 Text(3826.2857142857138, 1268.4, 'TCH <= 1.255\ngini = 0.69\nsamples = 10170\nvalue =
[4812, 624, 5267, 5356]\nclass = d'),
 Text(3507.428571428571, 906.0, 'SO_2 <= 7.005\ngini = 0.521\nsamples = 1958\nvalue = [1
039, 1, 1881, 198]\nclass = c'),
 Text(3347.9999999999995, 543.5999999999999, 'TOL <= 5.965\ngini = 0.592\nsamples = 277
\nvalue = [257, 0, 89, 119]\nclass = a'),
 Text(3268.285714285714, 181.19999999999982, 'gini = 0.637\nsamples = 138\nvalue = [75,
0, 51, 108]\nclass = d'),
 Text(3427.7142857142853, 181.19999999999982, 'gini = 0.35\nsamples = 139\nvalue = [182,
0, 38, 11]\nclass = a'),
 Text(3666.8571428571427, 543.5999999999999, 'CO <= 0.605\ngini = 0.456\nsamples = 1681
\nvalue = [782, 1, 1792, 79]\nclass = c'),
 Text(3587.142857142857, 181.19999999999982, 'gini = 0.501\nsamples = 1300\nvalue = [72
2, 1, 1276, 79]\nclass = c'),
 Text(3746.5714285714284, 181.19999999999982, 'gini = 0.187\nsamples = 381\nvalue = [60,
0, 516, 0]\nclass = c'),
 Text(4145.142857142857, 906.0, 'OXY <= 3.965\ngini = 0.685\nsamples = 8212\nvalue = [37
73, 623, 3386, 5158]\nclass = d'),
 Text(3985.7142857142853, 543.5999999999999, 'BEN <= 1.925\ngini = 0.597\nsamples = 4756
\nvalue = [872, 499, 1844, 4245]\nclass = d'),
 Text(3905.9999999999995, 181.19999999999982, 'gini = 0.479\nsamples = 2218\nvalue = [20
6, 218, 635, 2382]\nclass = d'),
 Text(4065.428571428571, 181.19999999999982, 'gini = 0.662\nsamples = 2538\nvalue = [66
6, 281, 1209, 1863]\nclass = d'),
 Text(4304.571428571428, 543.5999999999999, 'NO_2 <= 81.82\ngini = 0.612\nsamples = 3456
\nvalue = [2901, 124, 1542, 913]\nclass = a'),
 Text(4224.857142857142, 181.19999999999982, 'gini = 0.55\nsamples = 1616\nvalue = [159
4, 45, 556, 390]\nclass = a'),
 Text(4384.285714285714, 181.19999999999982, 'gini = 0.647\nsamples = 1840\nvalue = [130
7, 79, 986, 523]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.20586196841799131

Ridge Regression:0.20662440552663042

Lasso Regression:0.05638980897063961

ElasticNet Regression:0.10178683023266588

Logistic Regression:0.8481516938945678

Random Forest:0.7760081178858202

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