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_2003.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY
0	01-03-2003 01:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209999	NaN
1	01-03-2003 01:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389999	NaN
2	01-03-2003 01:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240002	NaN
3	01-03-2003 01:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839996	NaN
4	01-03-2003 01:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779999	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
243979	01-10-2003 00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380000	1.20
243980	01-10-2003 00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400000	0.50
243981	01-10-2003 00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830000	NaN

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY
243982	01-10-2003 00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570000	NaN
243983	01-10-2003 00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350000	2.43

243984 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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

Out[6]:

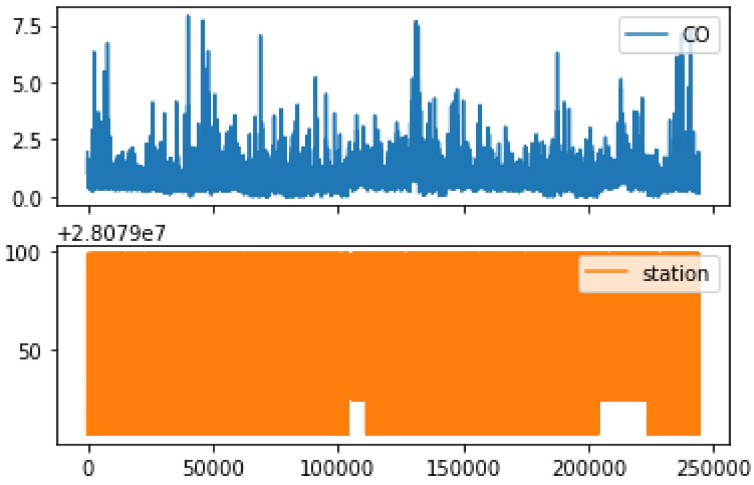
	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

Line chart

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

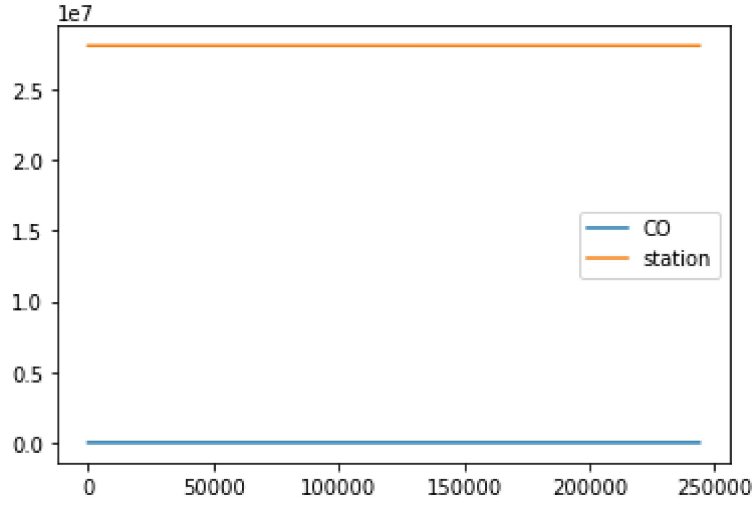
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

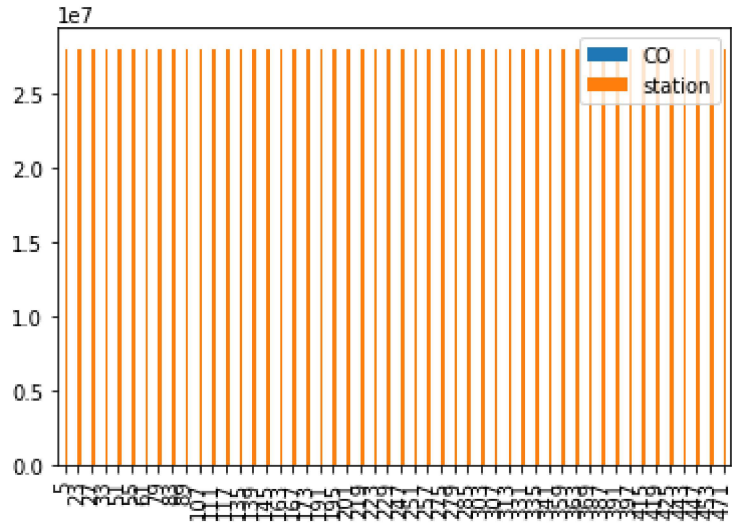


Bar chart

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

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

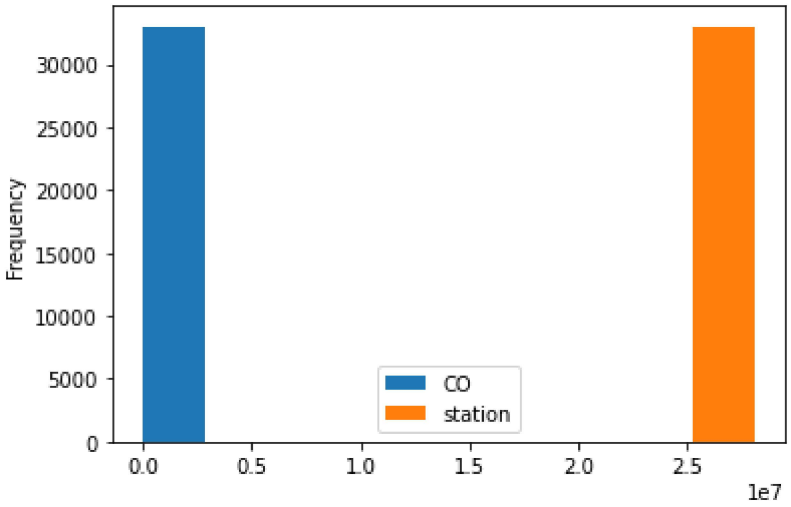
Out[10]: <AxesSubplot:>



Histogram

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

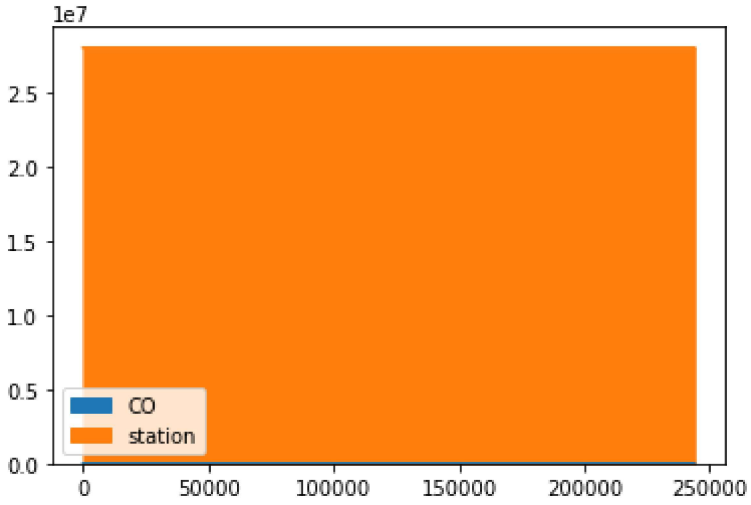
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

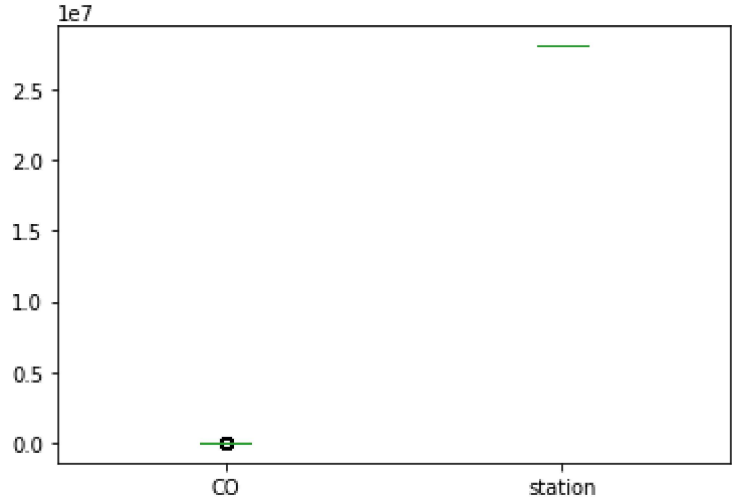
Out[12]: <AxesSubplot:>



Box chart

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

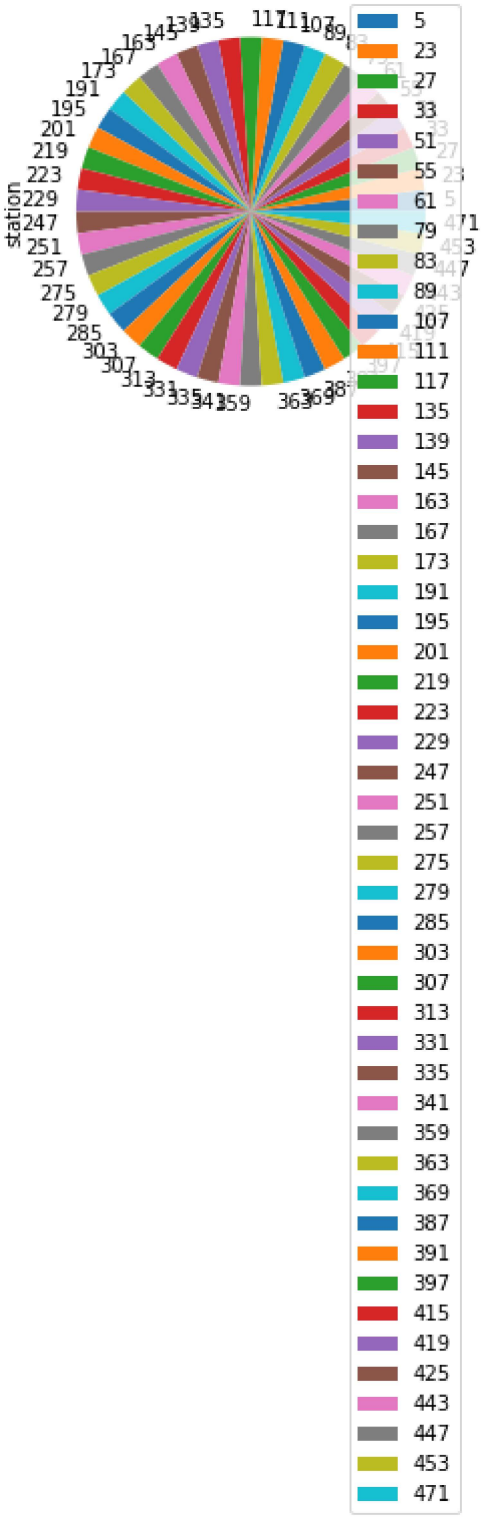
Out[13]: <AxesSubplot:>



Pie chart

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

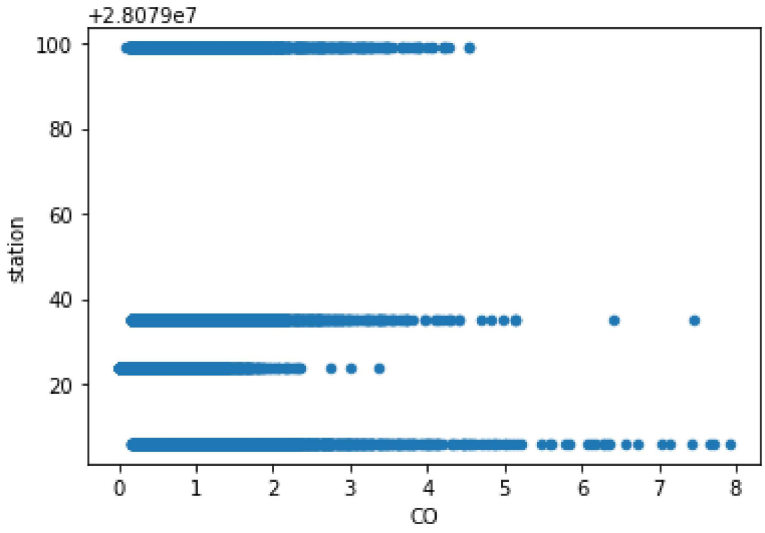
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049	120.15367
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082	104.52170
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999	49.07000
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000	92.77999
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004	160.10000

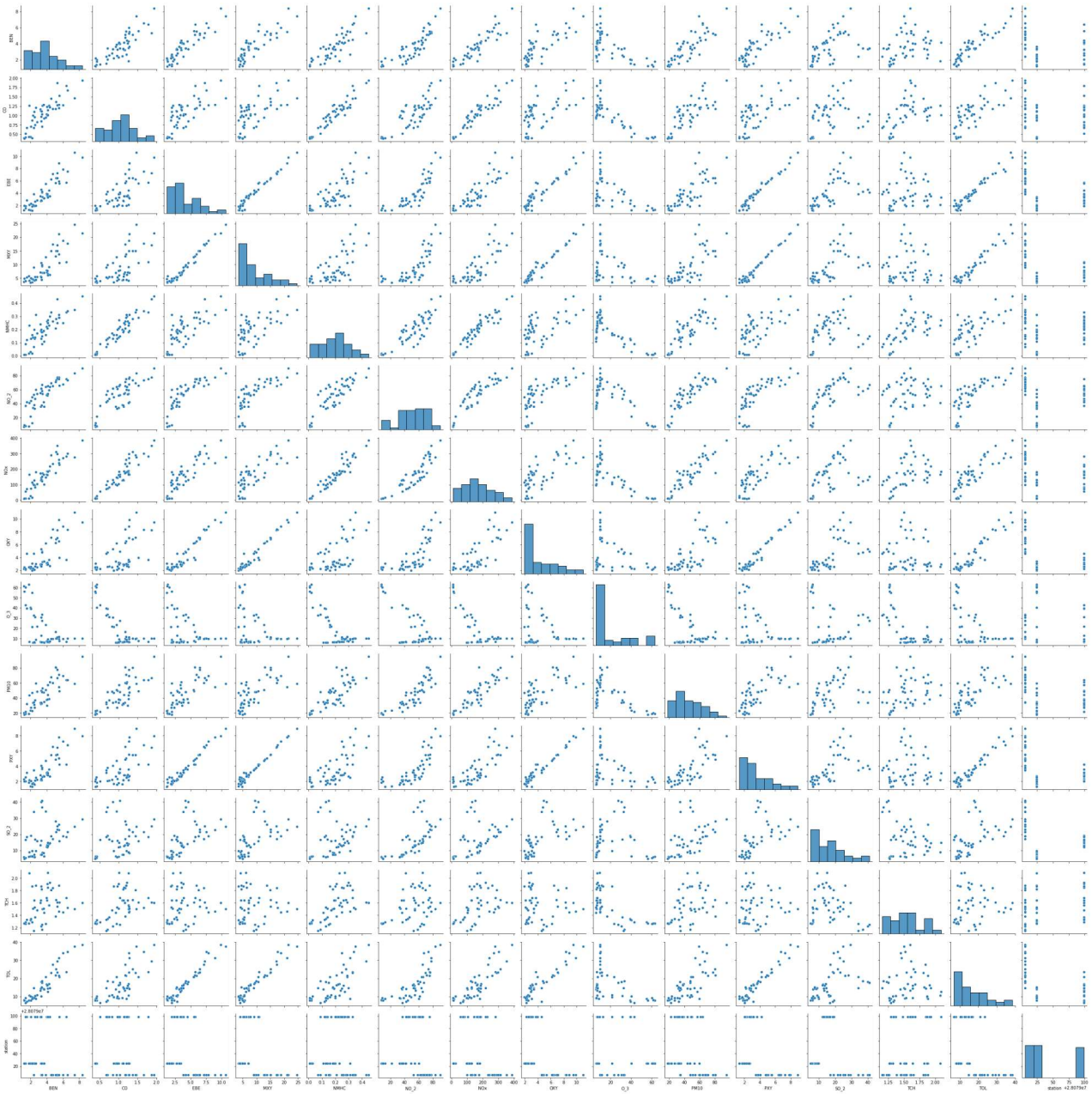
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012	1246.00000

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

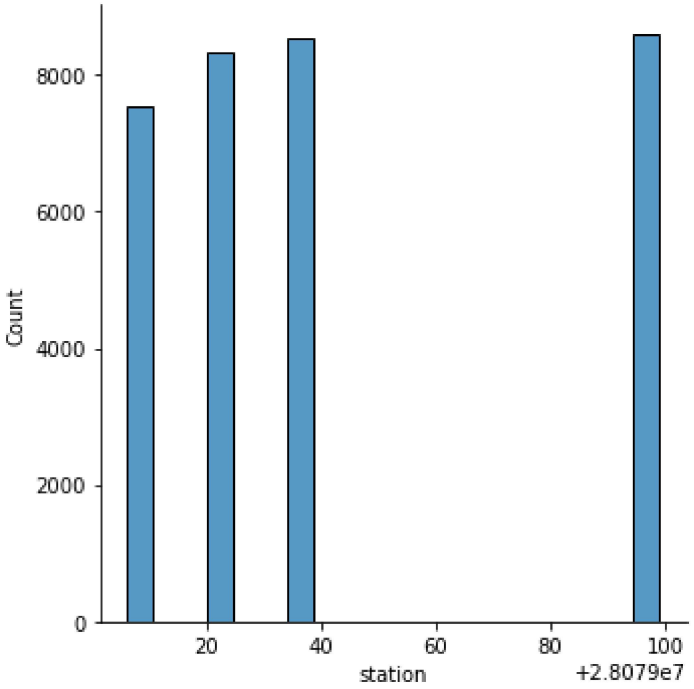
In [19]:
```
sns.pairplot(df1[0:50])
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x1986fd34730>



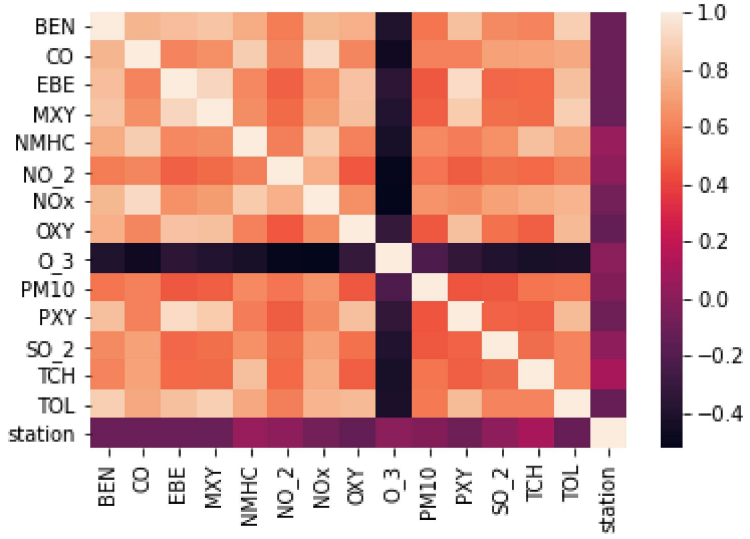
In [20]:
```
sns.displot(df1['station'])
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x198702d3ca0>



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]:
```
28079001.705791198
```

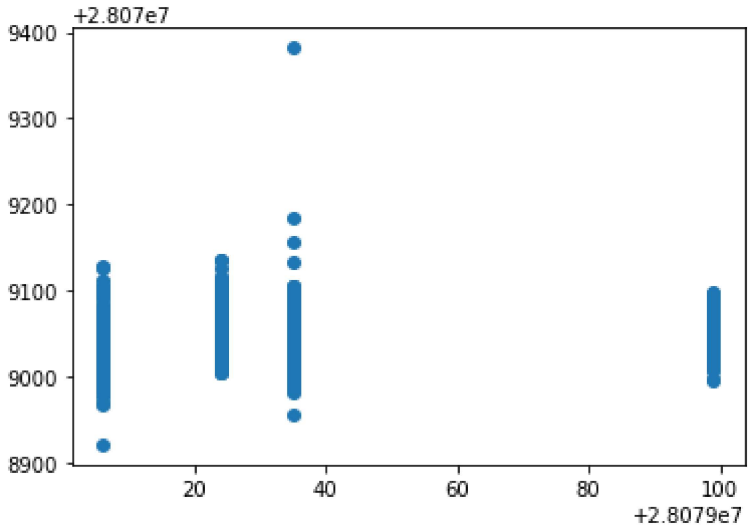
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.716726
CO	-40.340134
EBE	-1.691670
MXY	0.049299
NMHC	160.338149
NO_2	0.158254
NOx	-0.073205
OXY	-1.381130
O_3	-0.013500
PM10	-0.058948
PXY	1.865123
SO_2	0.847128
TCH	35.635134
TOL	-0.797461

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1987fa36730>
```



ACCURACY

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

Out[28]: 0.16672091906394193

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

Out[29]: 0.1798509962774265

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]: 0.1674968474856896

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

Out[33]: 0.1786612443473352

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]:
```
0.03629374637949523
```

Accuracy(Lasso)

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

Out[36]:
```
0.03451252116913206
```

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([ 0.        , -0.30448294,  0.09232789, -0.11725017,  0.11276744,
        0.1526912 , -0.07341294, -1.20739406, -0.04598254,  0.08088606,
        0.30356586,  0.75909519,  1.54993948, -0.40026107])
```

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

Out[39]:
```
28079037.861311376
```

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

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

Out[41]:
```
0.047227086151283015
```

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)))
```

```
29.08043222878616
1170.5345422485757
34.213075603467395
```

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]: (33010, 14)

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

Out[46]: (33010,)

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]:
```
0.758467131172372
```

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

Out[54]:
```
2.3461772071787448e-23
```

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

Out[55]:
```
array([[2.34617721e-23, 1.48032414e-55, 1.00000000e+00, 6.73763516e-16]])
```

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]:
```
0.7230709172578377
```

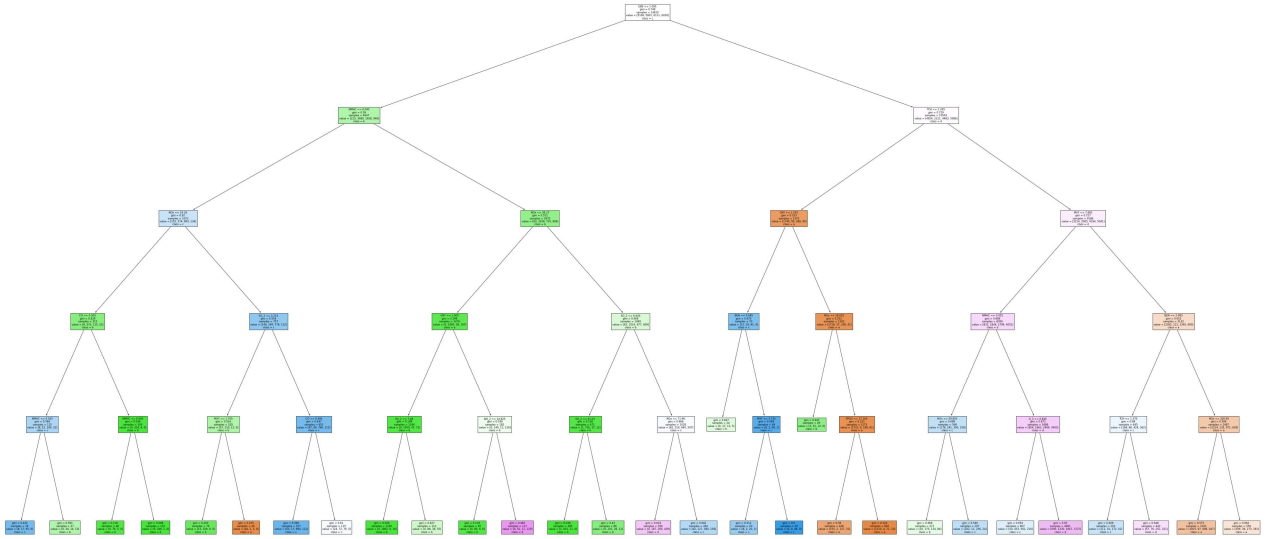
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(2291.785714285714, 1993.2, 'EBE <= 1.095\ngini = 0.749\nsamples = 14610\nvalue =
[5169, 5801, 6111, 6026]\nclass = c'),
 Text(1275.4285714285713, 1630.8000000000002, 'NMHC <= 0.045\ngini = 0.59\nsamples = 404
7\nvalue = [215, 3680, 1628, 940]\nclass = b'),
 Text(637.7142857142857, 1268.4, 'NOx <= 19.26\ngini = 0.62\nsamples = 1072\nvalue = [15
2, 574, 893, 134]\nclass = c'),
 Text(318.85714285714283, 906.0, 'CO <= 0.305\ngini = 0.429\nsamples = 315\nvalue = [8,
375, 115, 22]\nclass = b'),
 Text(159.42857142857142, 543.5999999999999, 'NMHC <= 0.025\ngini = 0.584\nsamples = 115
\nvalue = [8, 51, 109, 22]\nclass = c'),
 Text(79.71428571428571, 181.19999999999982, 'gini = 0.432\nsamples = 78\nvalue = [8, 1
7, 95, 9]\nclass = c'),
 Text(239.1428571428571, 181.19999999999982, 'gini = 0.591\nsamples = 37\nvalue = [0, 3
4, 14, 13]\nclass = b'),
 Text(478.2857142857142, 543.5999999999999, 'NMHC <= 0.005\ngini = 0.036\nsamples = 200
\nvalue = [0, 324, 6, 0]\nclass = b'),
 Text(398.57142857142856, 181.19999999999982, 'gini = 0.116\nsamples = 48\nvalue = [0, 7
6, 5, 0]\nclass = b'),
 Text(558.0, 181.19999999999982, 'gini = 0.008\nsamples = 152\nvalue = [0, 248, 1, 0]\nc
lass = b'),
 Text(956.5714285714284, 906.0, 'SO_2 <= 5.215\ngini = 0.554\nsamples = 757\nvalue = [14
4, 199, 778, 112]\nclass = c'),
 Text(797.1428571428571, 543.5999999999999, 'MXY <= 1.705\ngini = 0.516\nsamples = 105\n
value = [57, 110, 12, 0]\nclass = b'),
 Text(717.4285714285713, 181.19999999999982, 'gini = 0.293\nsamples = 79\nvalue = [13, 1
09, 9, 0]\nclass = b'),
 Text(876.8571428571428, 181.19999999999982, 'gini = 0.155\nsamples = 26\nvalue = [44,
1, 3, 0]\nclass = a'),
 Text(1116.0, 543.5999999999999, 'CO <= 0.495\ngini = 0.447\nsamples = 652\nvalue = [87,
89, 766, 112]\nclass = c'),
 Text(1036.2857142857142, 181.19999999999982, 'gini = 0.366\nsamples = 537\nvalue = [63,
17, 690, 112]\nclass = c'),
 Text(1195.7142857142856, 181.19999999999982, 'gini = 0.61\nsamples = 115\nvalue = [24,
72, 76, 0]\nclass = c'),
 Text(1913.1428571428569, 1268.4, 'NOx <= 36.27\ngini = 0.511\nsamples = 2975\nvalue =
[63, 3106, 735, 806]\nclass = b'),
 Text(1594.2857142857142, 906.0, 'OXY <= 1.005\ngini = 0.198\nsamples = 1476\nvalue =
[0, 2090, 58, 197]\nclass = b'),
 Text(1434.8571428571427, 543.5999999999999, 'SO_2 <= 7.48\ngini = 0.109\nsamples = 1294
\nvalue = [0, 1950, 47, 71]\nclass = b'),
 Text(1355.142857142857, 181.19999999999982, 'gini = 0.026\nsamples = 1182\nvalue = [0,
1862, 9, 16]\nclass = b'),
 Text(1514.5714285714284, 181.19999999999982, 'gini = 0.627\nsamples = 112\nvalue = [0,
88, 38, 55]\nclass = b'),
 Text(1753.7142857142856, 543.5999999999999, 'NO_2 <= 14.825\ngini = 0.536\nsamples = 18
2\nvalue = [0, 140, 11, 126]\nclass = b'),
 Text(1673.9999999999998, 181.19999999999982, 'gini = 0.118\nsamples = 65\nvalue = [0, 8
9, 0, 6]\nclass = b'),
 Text(1833.4285714285713, 181.19999999999982, 'gini = 0.483\nsamples = 117\nvalue = [0,
51, 11, 120]\nclass = d'),
 Text(2232.0, 906.0, 'SO_2 <= 6.645\ngini = 0.666\nsamples = 1499\nvalue = [63, 1016, 67
7, 609]\nclass = b'),
 Text(2072.5714285714284, 543.5999999999999, 'SO_2 <= 6.115\ngini = 0.126\nsamples = 471
\nvalue = [1, 702, 37, 12]\nclass = b'),
 Text(1992.8571428571427, 181.19999999999982, 'gini = 0.038\nsamples = 385\nvalue = [1,
601, 11, 0]\nclass = b'),
 Text(2152.285714285714, 181.19999999999982, 'gini = 0.43\nsamples = 86\nvalue = [0, 10
1, 26, 12]\nclass = b'),
 Text(2391.428571428571, 543.5999999999999, 'NOx <= 71.96\ngini = 0.666\nsamples = 1028
\nvalue = [62, 314, 640, 597]\nclass = c'),
 Text(2311.7142857142853, 181.19999999999982, 'gini = 0.624\nsamples = 566\nvalue = [0,
187, 260, 439]\nclass = d'),
 Text(2471.142857142857, 181.19999999999982, 'gini = 0.642\nsamples = 462\nvalue = [62,
127, 380, 158]\nclass = c'),
 Text(3308.142857142857, 1630.8000000000002, 'TCH <= 1.265\ngini = 0.729\nsamples = 1056
```

```
3\nvalue = [4954, 2121, 4483, 5086]\nclass = d'),
 Text(2790.0, 1268.4, 'OXY <= 1.025\ngini = 0.313\nsamples = 1375\nvalue = [1740, 56, 28
9, 45]\nclass = a'),
 Text(2630.5714285714284, 906.0, 'BEN <= 0.685\ngini = 0.474\nsamples = 73\nvalue = [12,
19, 81, 4]\nclass = c'),
 Text(2550.8571428571427, 543.5999999999999, 'gini = 0.647\nsamples = 24\nvalue = [4, 1
7, 13, 3]\nclass = b'),
 Text(2710.285714285714, 543.5999999999999, 'MXY <= 3.735\ngini = 0.248\nsamples = 49\nv
alue = [8, 2, 68, 1]\nclass = c'),
 Text(2630.5714285714284, 181.19999999999982, 'gini = 0.512\nsamples = 20\nvalue = [8,
2, 20, 1]\nclass = c'),
 Text(2790.0, 181.19999999999982, 'gini = 0.0\nsamples = 29\nvalue = [0, 0, 48, 0]\nclas
s = c'),
 Text(2949.428571428571, 906.0, 'NOx <= 18.635\ngini = 0.252\nsamples = 1302\nvalue = [1
728, 37, 208, 41]\nclass = a'),
 Text(2869.7142857142853, 543.5999999999999, 'gini = 0.447\nsamples = 29\nvalue = [3, 3
1, 10, 0]\nclass = b'),
 Text(3029.142857142857, 543.5999999999999, 'PM10 <= 17.165\ngini = 0.223\nsamples = 127
3\nvalue = [1725, 6, 198, 41]\nclass = a'),
 Text(2949.428571428571, 181.19999999999982, 'gini = 0.38\nsamples = 428\nvalue = [515,
2, 127, 31]\nclass = a'),
 Text(3108.8571428571427, 181.19999999999982, 'gini = 0.124\nsamples = 845\nvalue = [121
0, 4, 71, 10]\nclass = a'),
 Text(3826.2857142857138, 1268.4, 'MXY <= 7.885\ngini = 0.727\nsamples = 9188\nvalue =
[3214, 2065, 4194, 5041]\nclass = d'),
 Text(3507.428571428571, 906.0, 'NMHC <= 0.055\ngini = 0.688\nsamples = 6056\nvalue = [8
32, 1844, 2799, 4051]\nclass = d'),
 Text(3347.9999999999995, 543.5999999999999, 'NOx <= 59.855\ngini = 0.689\nsamples = 568
\nvalue = [178, 181, 390, 108]\nclass = c'),
 Text(3268.285714285714, 181.19999999999982, 'gini = 0.668\nsamples = 271\nvalue = [16,
170, 134, 84]\nclass = b'),
 Text(3427.7142857142853, 181.19999999999982, 'gini = 0.549\nsamples = 297\nvalue = [16
2, 11, 256, 24]\nclass = c'),
 Text(3666.8571428571427, 543.5999999999999, 'O_3 <= 6.845\ngini = 0.673\nsamples = 5488
\nvalue = [654, 1663, 2409, 3943]\nclass = d'),
 Text(3587.142857142857, 181.19999999999982, 'gini = 0.654\nsamples = 807\nvalue = [54,
433, 562, 216]\nclass = c'),
 Text(3746.5714285714284, 181.19999999999982, 'gini = 0.65\nsamples = 4681\nvalue = [60
0, 1230, 1847, 3727]\nclass = d'),
 Text(4145.142857142857, 906.0, 'BEN <= 2.985\ngini = 0.652\nsamples = 3132\nvalue = [23
82, 221, 1395, 990]\nclass = a'),
 Text(3985.7142857142853, 543.5999999999999, 'TCH <= 1.375\ngini = 0.68\nsamples = 645\n
value = [168, 86, 424, 362]\nclass = c'),
 Text(3905.9999999999995, 181.19999999999982, 'gini = 0.608\nsamples = 202\nvalue = [11
1, 10, 172, 41]\nclass = c'),
 Text(4065.428571428571, 181.19999999999982, 'gini = 0.648\nsamples = 443\nvalue = [57,
76, 252, 321]\nclass = d'),
 Text(4304.571428571428, 543.5999999999999, 'NOx <= 320.95\ngini = 0.599\nsamples = 2487
\nvalue = [2214, 135, 971, 628]\nclass = a'),
 Text(4224.857142857142, 181.19999999999982, 'gini = 0.573\nsamples = 1931\nvalue = [181
5, 97, 698, 447]\nclass = a'),
 Text(4384.285714285714, 181.19999999999982, 'gini = 0.662\nsamples = 556\nvalue = [399,
38, 273, 181]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.1798509962774265

Ridge Regression:0.1786612443473352

Lasso Regression:0.1786612443473352

ElasticNet Regression:0.047227086151283015

Logistic Regression:0.758467131172372

Random Forest:0.7230709172578377

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