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_2005.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	P
0	01-11-2005 01:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91	10.65	N
1	01-11-2005 01:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.93	NaN	1
2	01-11-2005 01:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60	NaN	N
3	01-11-2005 01:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16	NaN	N
4	01-11-2005 01:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
236995	01-01-2006 00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00	NaN	N
236996	01-01-2006 00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95	1.49	1
236997	01-01-2006 00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31	2.93	N

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	F
236998	01-01-2006 00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00	NaN	N
236999	01-01-2006 00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67	2.11	1

237000 rows × 17 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', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

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

Out[6]:

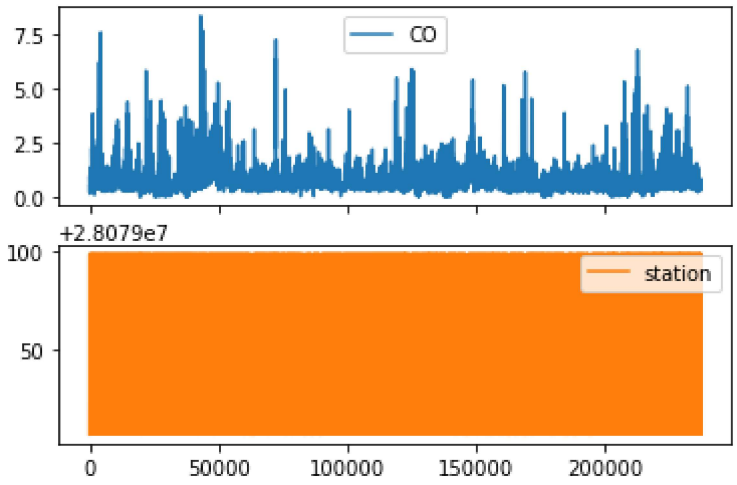
	CO	station
5	0.88	28079006
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

Line chart

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

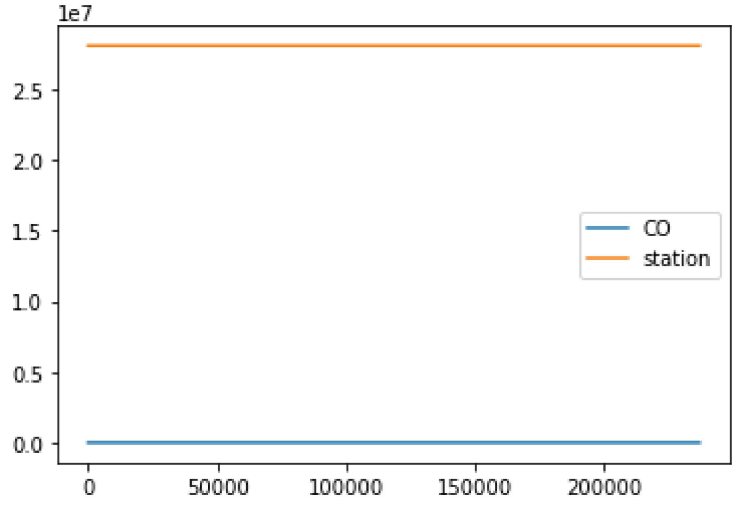
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

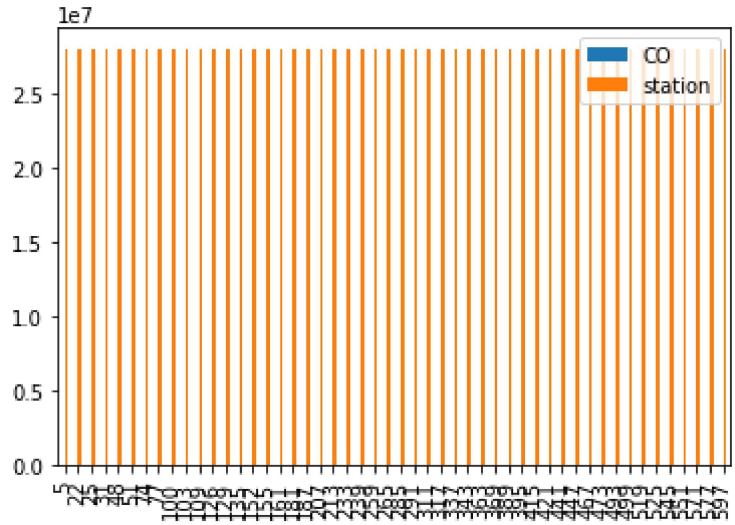


Bar chart

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

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

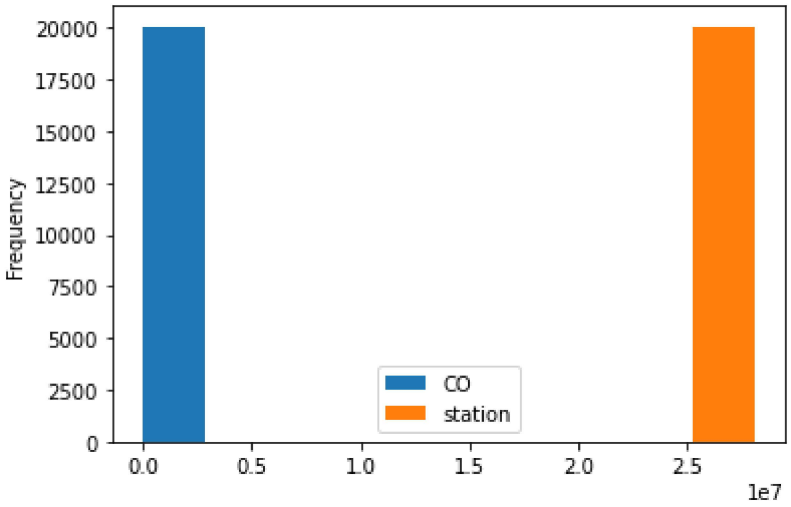
Out[10]: <AxesSubplot:>



Histogram

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

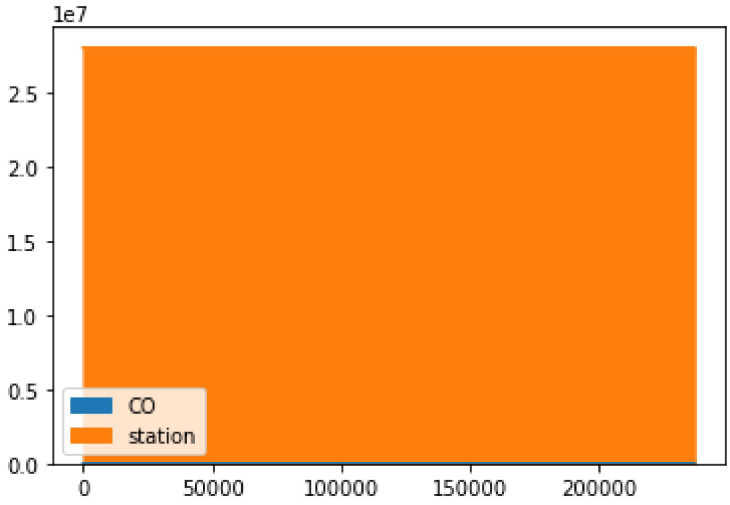
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

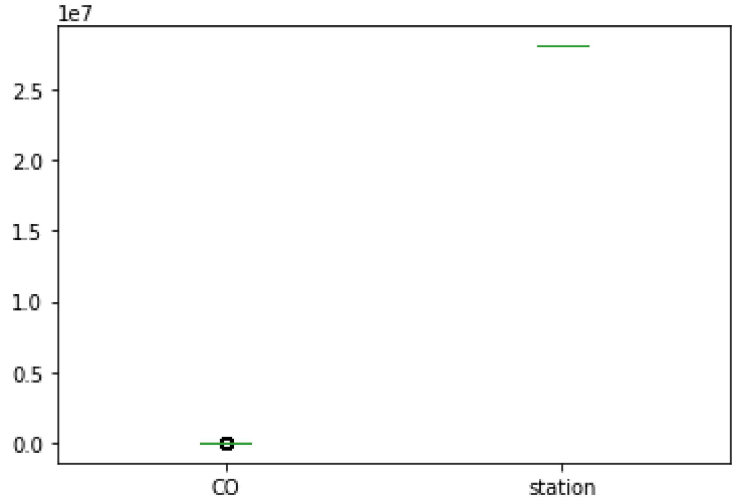
Out[12]: <AxesSubplot:>



Box chart

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

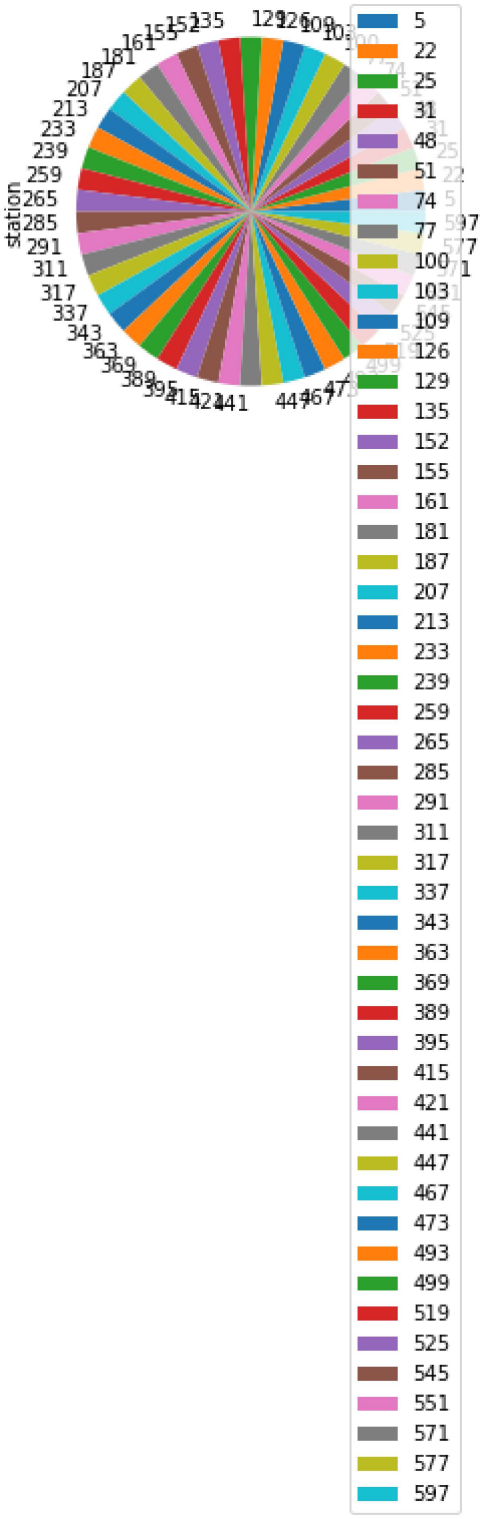
Out[13]: <AxesSubplot:>



Pie chart

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

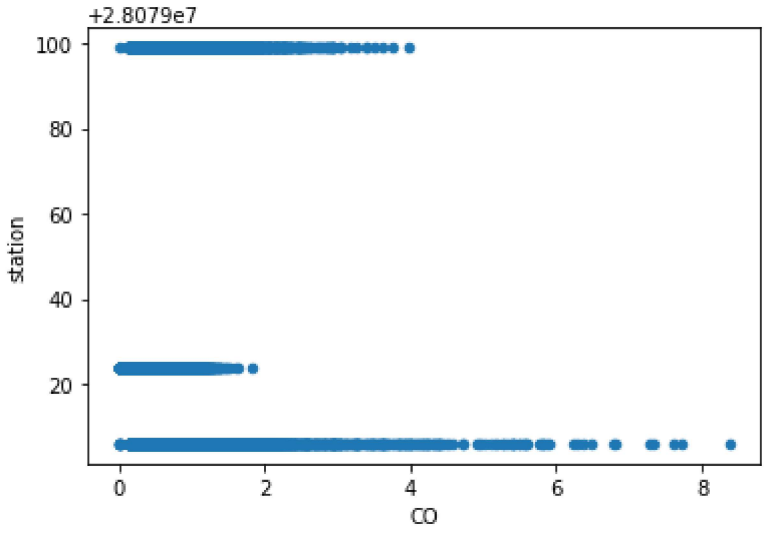
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.00000
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	143.04653
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	136.58252
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499	56.10249
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	105.69999

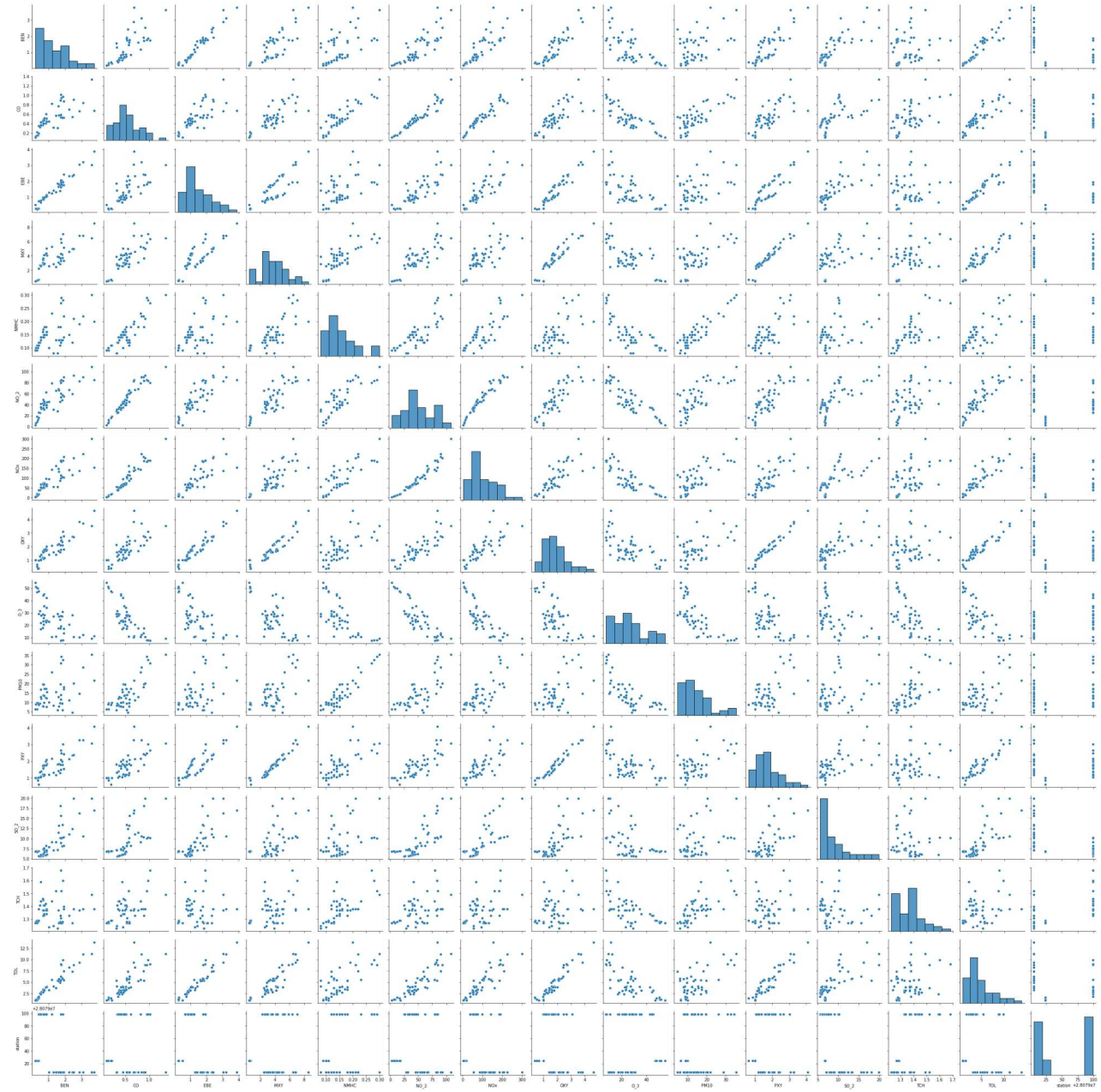
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	190.10000
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	1774.00000

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

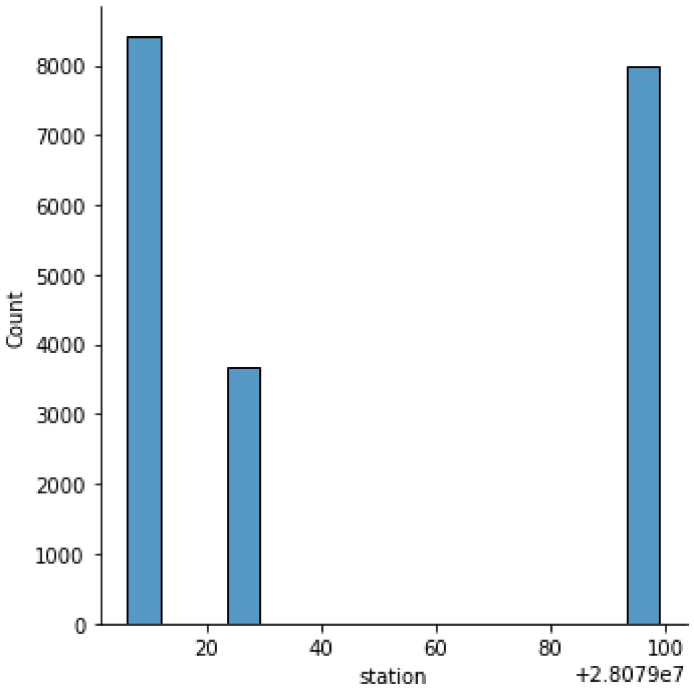
In [19]:
```
sns.pairplot(df1[0:50])
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x1f5986c1a00>



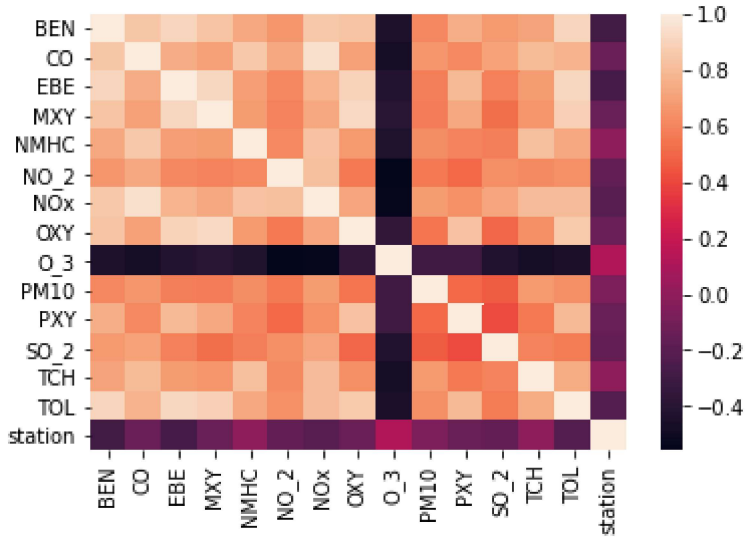
In [20]:
```
sns.displot(df1['station'])
```

Out[20]: <seaborn.axisgrid.FacetGrid at 0x1f5a25f6370>



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]:
```
28078962.89847
```

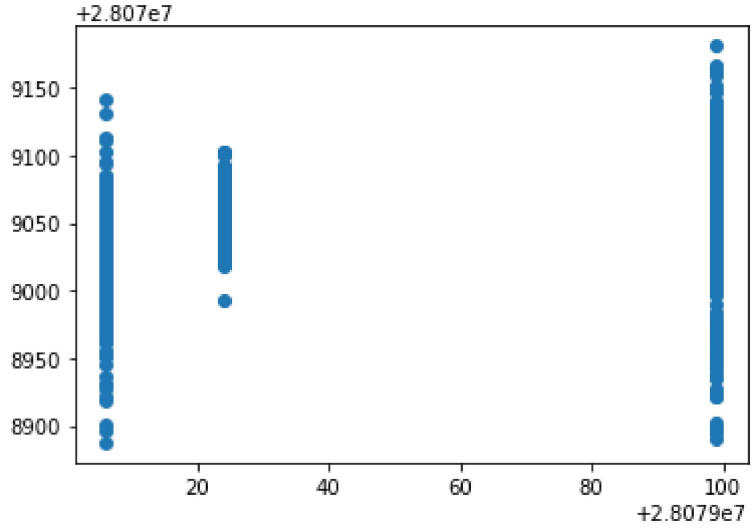
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-9.377578
CO	38.586651
EBE	-13.813828
MXY	3.803719
NMHC	78.540060
NO_2	0.108445
NOx	-0.266097
OXY	3.001159
O_3	0.004926
PM10	0.042065
PXY	2.856464
SO_2	0.169366
TCH	60.638026
TOL	-0.465624

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1f5a6faee50>
```



ACCURACY

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

Out[28]: 0.30504540761393817

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

Out[29]: 0.3035099019693833

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]: 0.30422996261464375

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

Out[33]: 0.3032785841437585

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]:
```
0.0645968056124504
```

Accuracy(Lasso)

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

Out[36]:
```
0.06282460793724132
```

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([-5.64719119,  1.43757114, -7.45833617,  2.64133603,  0.86078394,
       -0.06607035, -0.00802068,  1.77061463, -0.02225169,  0.2270263 ,
        1.5542768 ,  0.12597671,  1.53495543, -0.72054115])
```

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

Out[39]:
```
28079050.560312312
```

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

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

Out[41]:
```
0.1749626882401636
```

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)))
```

```
36.974561495274884
1552.1223586131443
39.396984130935
```

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]: (20070, 14)

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

Out[46]: (20070,)

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)
```

```
[28079006]
```

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

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

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

Out[53]:
```
0.879023418036871
```

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

Out[54]:
```
0.9998962857081471
```

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

Out[55]:
```
array([[9.99896286e-01, 3.19624619e-30, 1.03714292e-04]])
```

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]:
```
0.8708810970419669
```

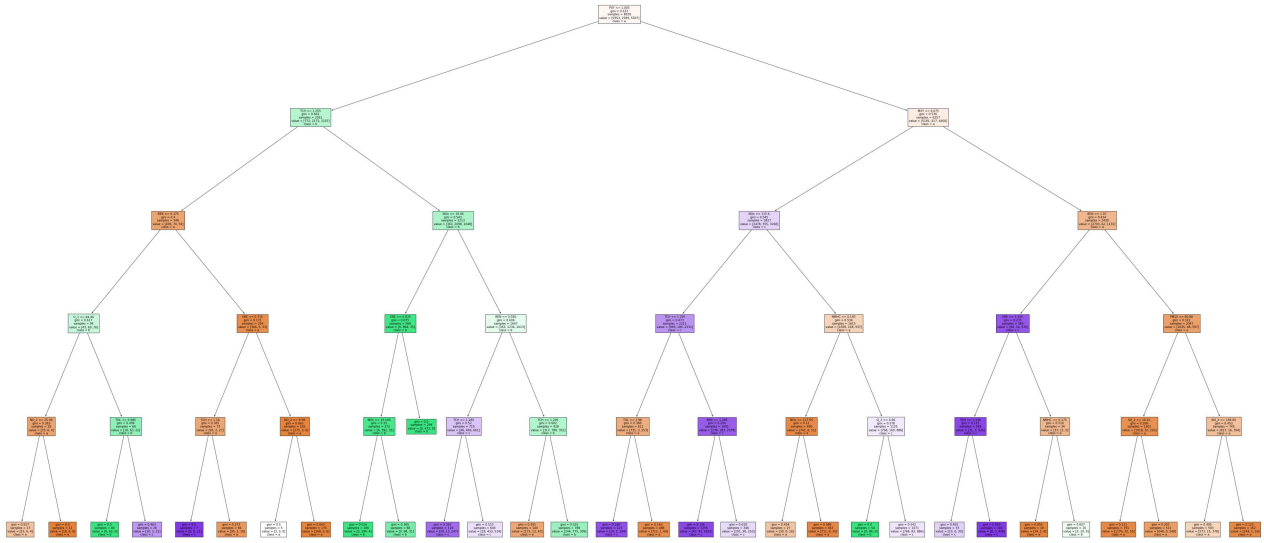
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(2185.5, 1993.2, 'PXY <= 1.005\ngini = 0.633\nsamples = 8818\nvalue = [5953, 2589,
5507]\nclass = a'),
 Text(1097.4, 1630.8000000000002, 'TCH <= 1.255\ngini = 0.602\nsamples = 2561\nvalue =
[772, 2172, 1107]\nclass = b'),
 Text(595.2, 1268.4, 'BEN <= 0.375\ngini = 0.4\nsamples = 348\nvalue = [409, 74, 59]\ncl
ass = a'),
 Text(297.6, 906.0, 'O_3 <= 44.46\ngini = 0.617\nsamples = 94\nvalue = [43, 69, 26]\ncla
ss = b'),
 Text(148.8, 543.5999999999999, 'NO_2 <= 25.04\ngini = 0.383\nsamples = 28\nvalue = [33,
6, 4]\nclass = a'),
 Text(74.4, 181.19999999999982, 'gini = 0.557\nsamples = 17\nvalue = [15, 6, 4]\nclass =
a'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [18, 0,
0]\nclass = a'),
 Text(446.40000000000003, 543.5999999999999, 'TOL <= 0.985\ngini = 0.496\nsamples = 66\n
value = [10, 63, 22]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.0\nsamples = 40\nvalue = [0, 62, 0]\nclass =
b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.463\nsamples = 26\nvalue = [10,
1, 22]\nclass = c'),
 Text(892.8000000000001, 906.0, 'EBE <= 0.715\ngini = 0.172\nsamples = 254\nvalue = [36
6, 5, 33]\nclass = a'),
 Text(744.0, 543.5999999999999, 'TCH <= 1.16\ngini = 0.365\nsamples = 73\nvalue = [95,
2, 27]\nclass = a'),
 Text(669.6, 181.19999999999982, 'gini = 0.0\nsamples = 7\nvalue = [0, 0, 11]\nclass =
c'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.273\nsamples = 66\nvalue = [95,
2, 16]\nclass = a'),
 Text(1041.6000000000001, 543.5999999999999, 'NO_2 <= 9.58\ngini = 0.063\nsamples = 181
\nvalue = [271, 3, 6]\nclass = a'),
 Text(967.2, 181.19999999999982, 'gini = 0.5\nsamples = 5\nvalue = [3, 3, 0]\nclass =
a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.043\nsamples = 176\nvalue = [268, 0, 6]\ncla
ss = a'),
 Text(1599.6000000000001, 1268.4, 'NOx <= 19.06\ngini = 0.543\nsamples = 2213\nvalue =
[363, 2098, 1048]\nclass = b'),
 Text(1413.6000000000001, 906.0, 'EBE <= 0.835\ngini = 0.075\nsamples = 566\nvalue = [0,
864, 35]\nclass = b'),
 Text(1339.2, 543.5999999999999, 'NOx <= 15.105\ngini = 0.15\nsamples = 272\nvalue = [0,
392, 35]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.026\nsamples = 186\nvalue = [0,
294, 4]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.365\nsamples = 86\nvalue = [0, 9
8, 31]\nclass = b'),
 Text(1488.0, 543.5999999999999, 'gini = 0.0\nsamples = 294\nvalue = [0, 472, 0]\nclass
= b'),
 Text(1785.6000000000001, 906.0, 'BEN <= 0.585\ngini = 0.606\nsamples = 1647\nvalue = [3
63, 1234, 1013]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'TCH <= 1.285\ngini = 0.52\nsamples = 719\n
value = [46, 446, 661]\nclass = c'),
 Text(1562.4, 181.19999999999982, 'gini = 0.362\nsamples = 119\nvalue = [28, 13, 147]\nc
lass = c'),
 Text(1711.2, 181.19999999999982, 'gini = 0.515\nsamples = 600\nvalue = [18, 433, 514]\n
class = c'),
 Text(1934.4, 543.5999999999999, 'TCH <= 1.295\ngini = 0.602\nsamples = 928\nvalue = [31
7, 788, 352]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.391\nsamples = 140\nvalue = [17
3, 13, 43]\nclass = a'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.525\nsamples = 788\nvalue = [14
4, 775, 309]\nclass = b'),
 Text(3273.6000000000004, 1630.8000000000002, 'MXY <= 6.675\ngini = 0.536\nsamples = 625
7\nvalue = [5181, 417, 4400]\nclass = a'),
 Text(2678.4, 1268.4, 'NOx <= 115.6\ngini = 0.545\nsamples = 3827\nvalue = [2478, 355, 3
268]\nclass = c'),
 Text(2380.8, 906.0, 'TCH <= 1.295\ngini = 0.473\nsamples = 2212\nvalue = [969, 186, 233
```

```
1]\nclass = c'),
 Text(2232.0, 543.5999999999999, 'TOL <= 2.98\ngini = 0.386\nsamples = 611\nvalue = [73
1, 3, 253]\nclass = a'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.186\nsamples = 123\nvalue = [19,
2, 184]\nclass = c'),
 Text(2306.4, 181.19999999999982, 'gini = 0.163\nsamples = 488\nvalue = [712, 1, 69]\ncl
ass = a'),
 Text(2529.6000000000004, 543.5999999999999, 'BEN <= 1.285\ngini = 0.294\nsamples = 1601
\nvalue = [238, 183, 2078]\nclass = c'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.135\nsamples = 1255\nvalue = [4
7, 93, 1825]\nclass = c'),
 Text(2604.0, 181.19999999999982, 'gini = 0.619\nsamples = 346\nvalue = [191, 90, 253]\n
class = c'),
 Text(2976.0, 906.0, 'NMHC <= 0.145\ngini = 0.534\nsamples = 1615\nvalue = [1509, 169, 9
37]\nclass = a'),
 Text(2827.2000000000003, 543.5999999999999, 'NOx <= 117.55\ngini = 0.12\nsamples = 490
\nvalue = [741, 0, 51]\nclass = a'),
 Text(2752.8, 181.19999999999982, 'gini = 0.454\nsamples = 27\nvalue = [30, 0, 16]\nclas
s = a'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.089\nsamples = 463\nvalue = [71
1, 0, 35]\nclass = a'),
 Text(3124.8, 543.5999999999999, 'O_3 <= 4.66\ngini = 0.578\nsamples = 1125\nvalue = [76
8, 169, 886]\nclass = c'),
 Text(3050.4, 181.19999999999982, 'gini = 0.0\nsamples = 54\nvalue = [0, 86, 0]\nclass =
b'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.542\nsamples = 1071\nvalue = [76
8, 83, 886]\nclass = c'),
 Text(3868.8, 1268.4, 'BEN <= 1.97\ngini = 0.434\nsamples = 2430\nvalue = [2703, 62, 113
2]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'EBE <= 3.355\ngini = 0.235\nsamples = 383\nvalue = [6
8, 14, 535]\nclass = c'),
 Text(3422.4, 543.5999999999999, 'TCH <= 1.335\ngini = 0.115\nsamples = 349\nvalue = [3
1, 3, 526]\nclass = c'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.491\nsamples = 33\nvalue = [23,
0, 30]\nclass = c'),
 Text(3496.8, 181.19999999999982, 'gini = 0.043\nsamples = 316\nvalue = [8, 3, 496]\ncla
ss = c'),
 Text(3720.0000000000005, 543.5999999999999, 'NMHC <= 0.175\ngini = 0.516\nsamples = 34
\nvalue = [37, 11, 9]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.056\nsamples = 18\nvalue = [34,
1, 0]\nclass = a'),
 Text(3794.4, 181.19999999999982, 'gini = 0.607\nsamples = 16\nvalue = [3, 10, 9]\nclass
= b'),
 Text(4166.400000000001, 906.0, 'PM10 <= 60.86\ngini = 0.321\nsamples = 2047\nvalue = [2
635, 48, 597]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'SO_2 <= 16.41\ngini = 0.206\nsamples = 130
2\nvalue = [1818, 32, 203]\nclass = a'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.131\nsamples = 791\nvalue = [117
0, 32, 55]\nclass = a'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.303\nsamples = 511\nvalue = [64
8, 0, 148]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'NO_2 <= 158.65\ngini = 0.453\nsamples = 745
\nvalue = [817, 16, 394]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.495\nsamples = 583\nvalue = [573, 15, 378]\n
class = a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.122\nsamples = 162\nvalue = [244, 1, 16]\ncl
ass = a')]
```



Conclusion

Accuracy

Linear Regression:0.30504540761393817

Ridge Regression:0.30422996261464375

Lasso Regression:0.06282460793724132

ElasticNet Regression:0.1749626882401636

Logistic Regression:0.879023418036871

Random Forest:0.8708810970419669

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