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_2007.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PI
0	01-12-2007 01:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.199997	9
1	01-12-2007 01:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.809998	
2	01-12-2007 01:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.099998	
3	01-12-2007 01:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.300003	
4	01-12-2007 01:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.500000	1
...	...	...	...	...	...	...	...	...	...	...	...	
225115	01-03-2007 00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.760000	
225116	01-03-2007 00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	5.700000	
225117	01-03-2007 00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	13.010000	

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PI
225118	01-03-2007 00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	6.610000	
225119	01-03-2007 00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.260000	

225120 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: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

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

Out[6]:

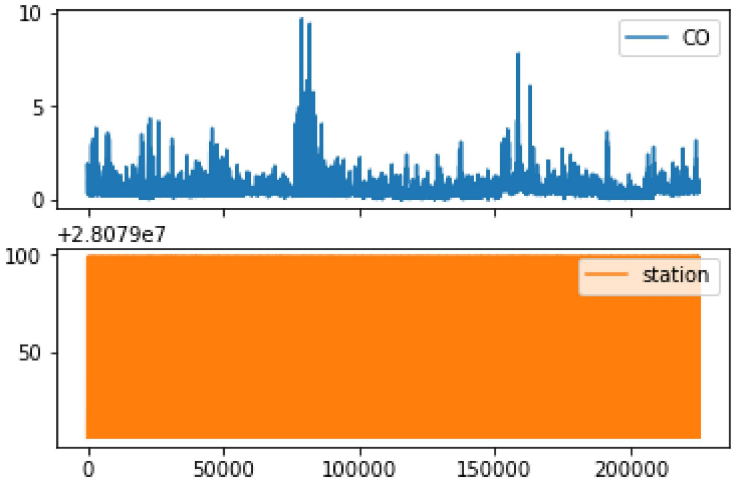
	CO	station
4	1.86	28079006
21	0.31	28079024
25	1.42	28079099
30	1.89	28079006
47	0.30	28079024
...	...	...
225073	0.47	28079006
225094	0.45	28079099
225098	0.41	28079006
225115	0.45	28079024
225119	0.40	28079099

25443 rows × 2 columns

Line chart

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

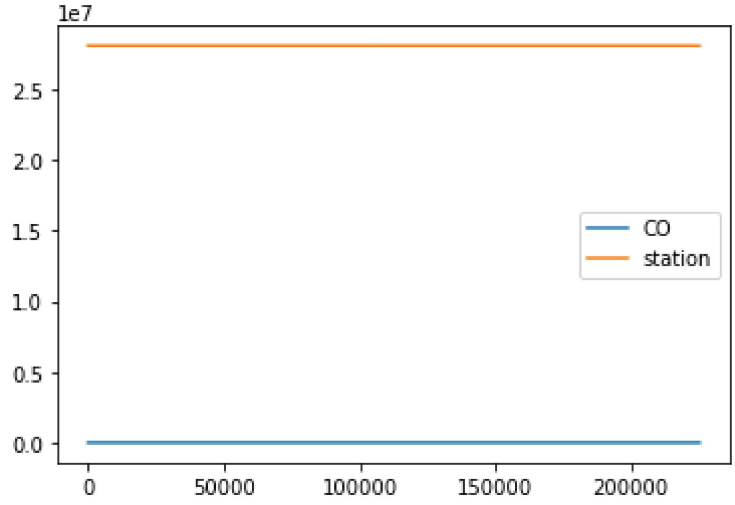
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

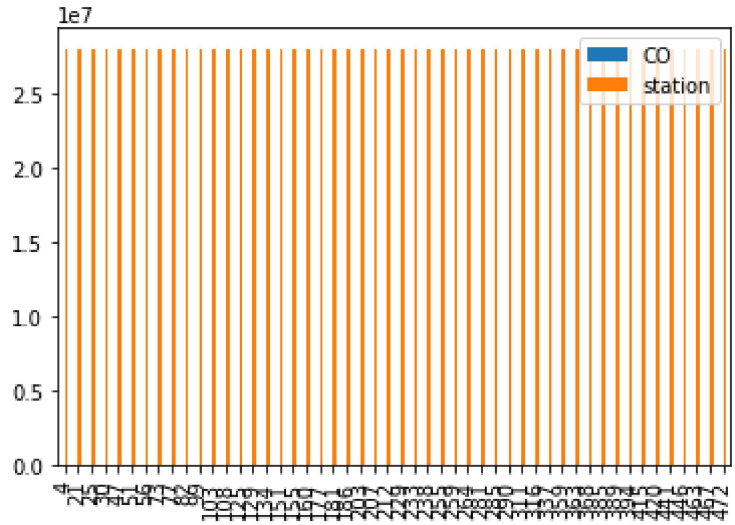


Bar chart

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

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

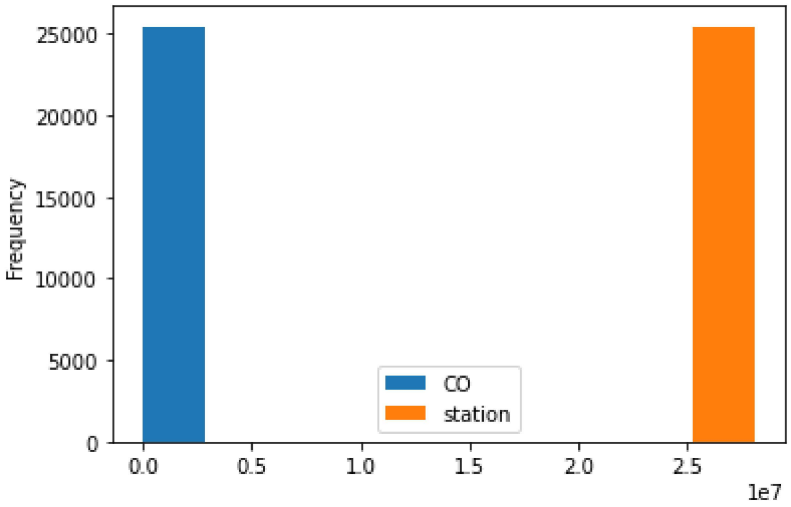
Out[10]: <AxesSubplot:>



Histogram

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

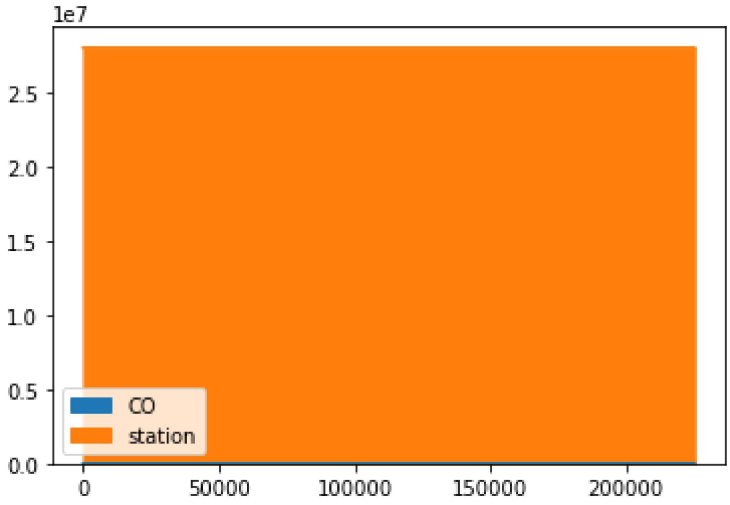
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

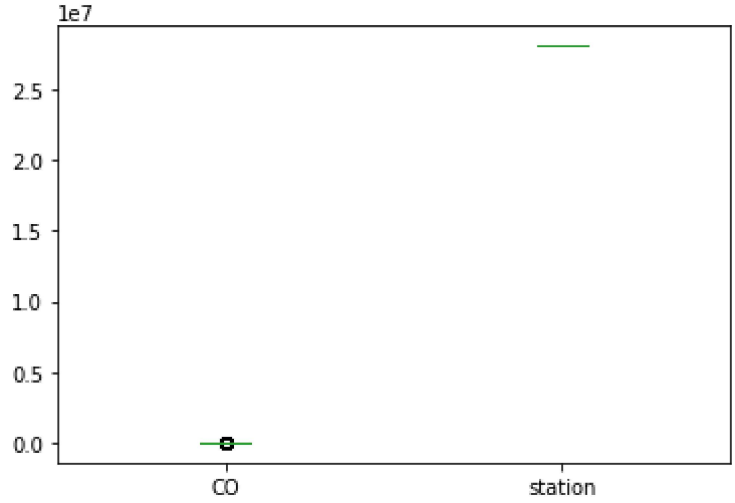
Out[12]: <AxesSubplot:>



Box chart

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

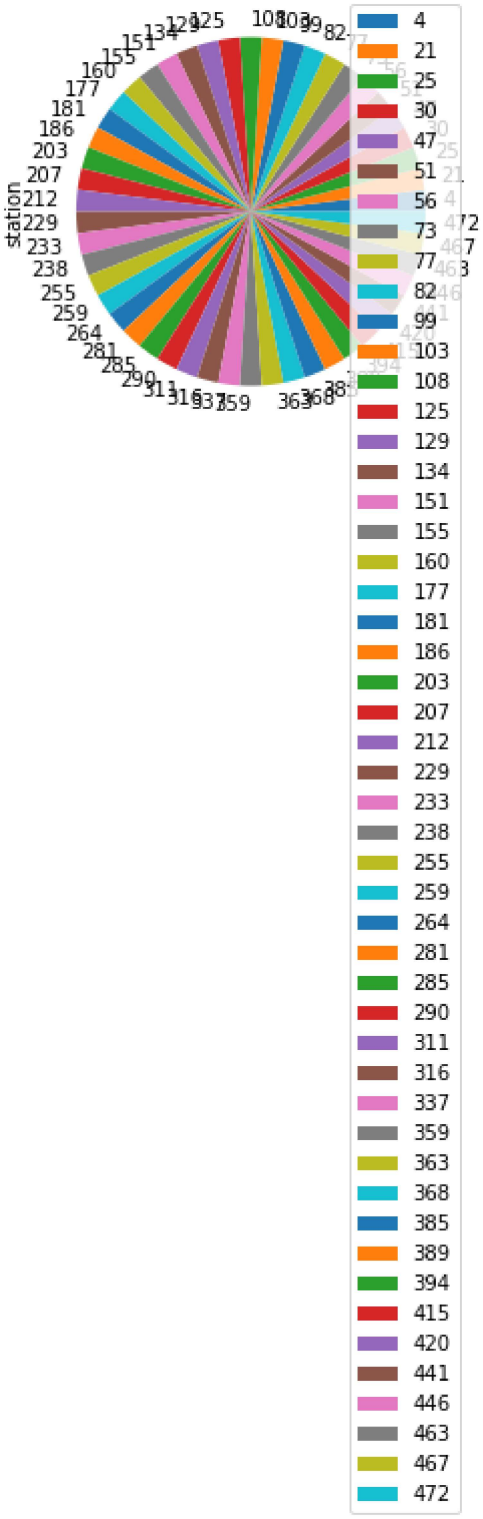
Out[13]: <AxesSubplot:>



Pie chart

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

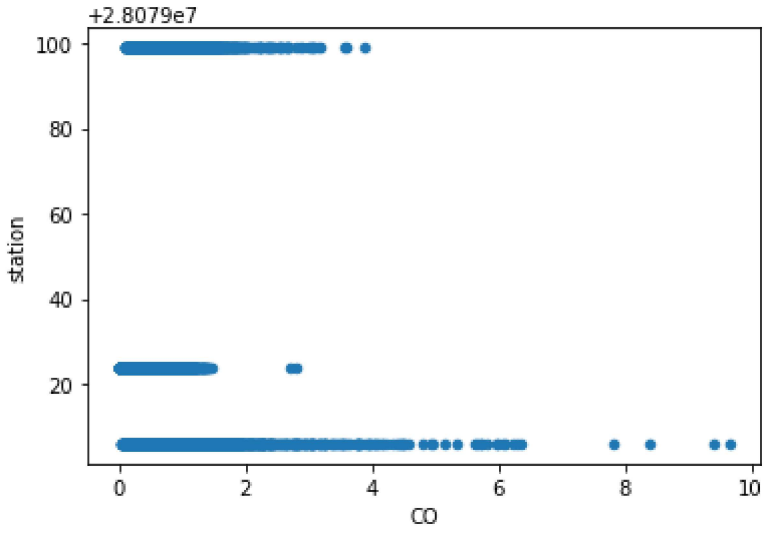
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.00000
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683	112.74186
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029	115.52700
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000	1.78000
25%	0.450000	0.260000	0.780000	0.960000	0.160000	31.285001	39.91000
50%	0.770000	0.400000	1.000000	1.500000	0.220000	54.080002	82.80999

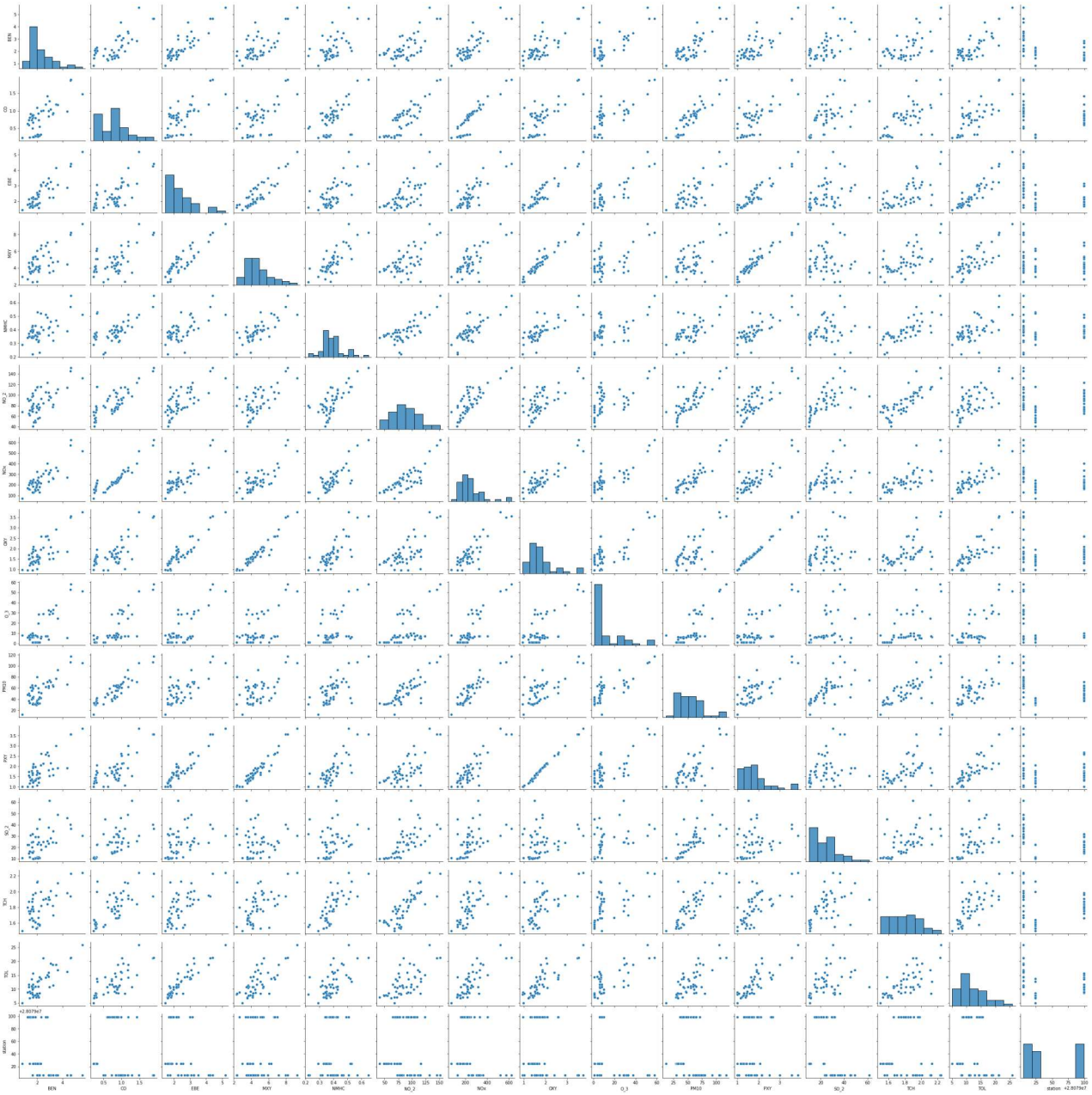
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	1.390000	0.640000	1.580000	2.855000	0.300000	79.230003	149.30000
max	30.139999	9.660000	31.680000	65.480003	2.570000	430.299988	1893.00000

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])
```

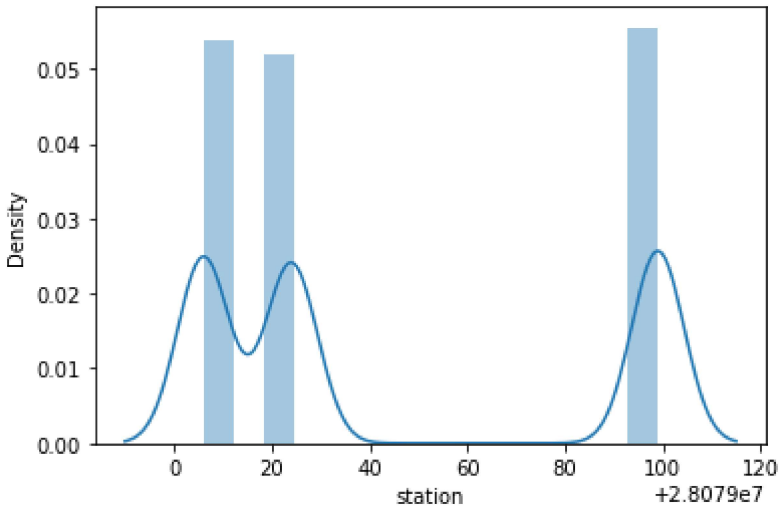
Out[19]: <seaborn.axisgrid.PairGrid at 0x1f968e91100>



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

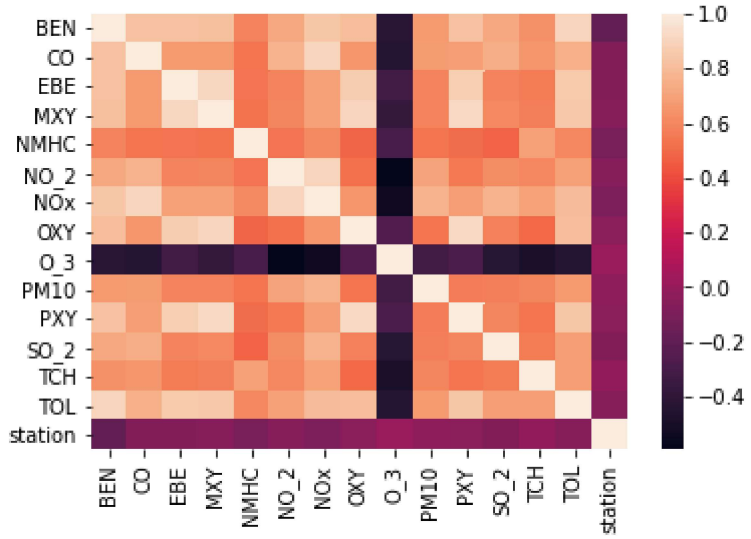
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]: <AxesSubplot:xlabel='station', ylabel='Density'>



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]:
```
28079007.64139326
```

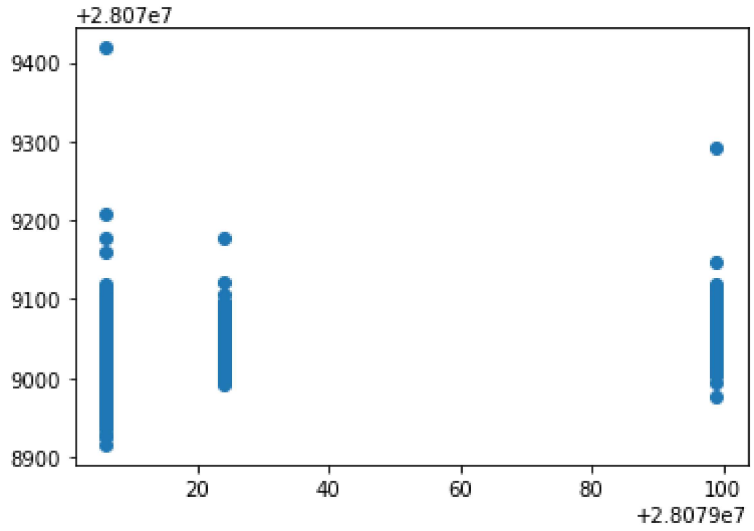
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-33.061239
CO	18.458079
EBE	0.954915
MXY	-1.464538
NMHC	-39.035707
NO_2	0.097440
NOx	-0.029132
OXY	7.402488
O_3	-0.024969
PM10	0.137297
PXY	5.795919
SO_2	0.146833
TCH	26.351619
TOL	3.085223

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1f977856bb0>
```



ACCURACY

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

Out[28]: 0.16138975732905902

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

Out[29]: 0.15733111828428403

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]: 0.1613210457141202

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

Out[33]: 0.15728104086612638

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]:
```
0.012611579967684139
```

Accuracy(Lasso)

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

Out[36]:
```
0.014671192800835842
```

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([-7.96019664,  0.        ,  0.        ,  0.139444  , -0.        ,
        0.04509635, -0.04482903,  0.60332642, -0.05243443,  0.15658336,
        0.69375031, -0.01660662,  0.        ,  0.94408825])
```

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

Out[39]:
```
28079045.686490264
```

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

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

Out[41]:
```
0.06969624440569488
```

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)))
```

```
36.70409685135137
1543.770377944848
39.29084343641465
```

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]: (25443, 14)

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

Out[46]: (25443,)

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)
```

```
[28079099]
```

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

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

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

Out[53]:
```
0.8146838030106512
```

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

Out[54]:
```
1.0827516273438634e-19
```

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

Out[55]:
```
array([[1.08275163e-19, 1.80384181e-19, 1.00000000e+00]])
```

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]:
```
0.8216732172936553
```

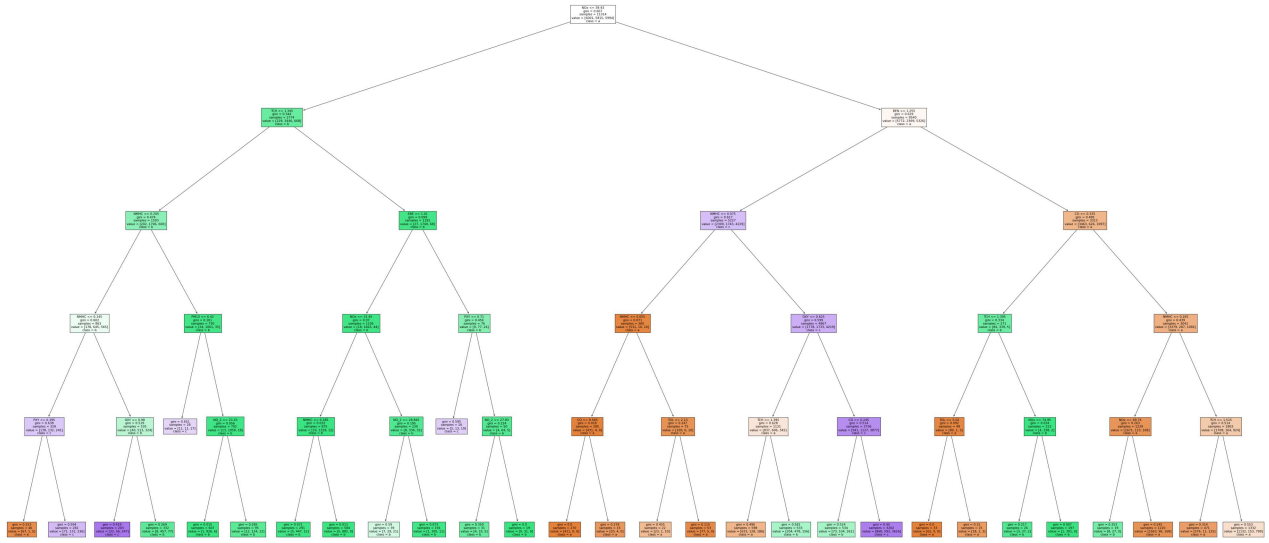
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(2092.5, 1993.2, 'NOx <= 39.43\ngini = 0.667\nsamples = 11314\nvalue = [6001, 5815,
5994]\nclass = a'),
 Text(996.4285714285713, 1630.8000000000002, 'TCH <= 1.345\ngini = 0.344\nsamples = 2774
\nvalue = [229, 3446, 668]\nclass = b'),
 Text(518.1428571428571, 1268.4, 'NMHC <= 0.205\ngini = 0.474\nsamples = 1593\nvalue =
[202, 1706, 600]\nclass = b'),
 Text(318.85714285714283, 906.0, 'NMHC <= 0.145\ngini = 0.602\nsamples = 863\nvalue = [1
78, 645, 565]\nclass = b'),
 Text(159.42857142857142, 543.5999999999999, 'PXY <= 0.395\ngini = 0.638\nsamples = 328
\nvalue = [138, 132, 241]\nclass = c'),
 Text(79.71428571428571, 181.19999999999982, 'gini = 0.153\nsamples = 46\nvalue = [67,
1, 5]\nclass = a'),
 Text(239.1428571428571, 181.19999999999982, 'gini = 0.594\nsamples = 282\nvalue = [71,
131, 236]\nclass = c'),
 Text(478.2857142857142, 543.5999999999999, 'OXY <= 0.98\ngini = 0.519\nsamples = 535\nv
alue = [40, 513, 324]\nclass = b'),
 Text(398.57142857142856, 181.19999999999982, 'gini = 0.419\nsamples = 203\nvalue = [32,
56, 247]\nclass = c'),
 Text(558.0, 181.19999999999982, 'gini = 0.269\nsamples = 332\nvalue = [8, 457, 77]\ncla
ss = b'),
 Text(717.4285714285713, 906.0, 'PM10 <= 6.42\ngini = 0.101\nsamples = 730\nvalue = [24,
1061, 35]\nclass = b'),
 Text(637.7142857142857, 543.5999999999999, 'gini = 0.651\nsamples = 28\nvalue = [11, 1
1, 17]\nclass = c'),
 Text(797.1428571428571, 543.5999999999999, 'NO_2 <= 23.25\ngini = 0.056\nsamples = 702
\nvalue = [13, 1050, 18]\nclass = b'),
 Text(717.4285714285713, 181.19999999999982, 'gini = 0.015\nsamples = 607\nvalue = [1, 9
26, 6]\nclass = b'),
 Text(876.8571428571428, 181.19999999999982, 'gini = 0.285\nsamples = 95\nvalue = [12, 1
24, 12]\nclass = b'),
 Text(1474.7142857142856, 1268.4, 'EBE <= 1.01\ngini = 0.099\nsamples = 1181\nvalue = [2
7, 1740, 68]\nclass = b'),
 Text(1275.4285714285713, 906.0, 'NOx <= 31.45\ngini = 0.07\nsamples = 1105\nvalue = [1
8, 1663, 44]\nclass = b'),
 Text(1116.0, 543.5999999999999, 'NMHC <= 0.185\ngini = 0.032\nsamples = 875\nvalue = [1
0, 1329, 12]\nclass = b'),
 Text(1036.2857142857142, 181.19999999999982, 'gini = 0.071\nsamples = 291\nvalue = [5,
447, 12]\nclass = b'),
 Text(1195.7142857142856, 181.19999999999982, 'gini = 0.011\nsamples = 584\nvalue = [5,
882, 0]\nclass = b'),
 Text(1434.8571428571427, 543.5999999999999, 'NO_2 <= 28.845\ngini = 0.195\nsamples = 23
0\nvalue = [8, 334, 32]\nclass = b'),
 Text(1355.142857142857, 181.19999999999982, 'gini = 0.59\nsamples = 39\nvalue = [7, 29,
21]\nclass = b'),
 Text(1514.5714285714284, 181.19999999999982, 'gini = 0.073\nsamples = 191\nvalue = [1,
305, 11]\nclass = b'),
 Text(1673.9999999999998, 906.0, 'PXY <= 0.71\ngini = 0.456\nsamples = 76\nvalue = [9, 7
7, 24]\nclass = b'),
 Text(1594.2857142857142, 543.5999999999999, 'gini = 0.595\nsamples = 26\nvalue = [5, 1
3, 19]\nclass = c'),
 Text(1753.7142857142856, 543.5999999999999, 'NO_2 <= 27.83\ngini = 0.224\nsamples = 50
\nvalue = [4, 64, 5]\nclass = b'),
 Text(1673.9999999999998, 181.19999999999982, 'gini = 0.359\nsamples = 31\nvalue = [4, 3
3, 5]\nclass = b'),
 Text(1833.4285714285713, 181.19999999999982, 'gini = 0.0\nsamples = 19\nvalue = [0, 31,
0]\nclass = b'),
 Text(3188.5714285714284, 1630.8000000000002, 'BEN <= 1.255\ngini = 0.629\nsamples = 854
0\nvalue = [5772, 2369, 5326]\nclass = a'),
 Text(2550.8571428571427, 1268.4, 'NMHC <= 0.075\ngini = 0.617\nsamples = 5227\nvalue =
[2309, 1743, 4229]\nclass = c'),
 Text(2232.0, 906.0, 'NMHC <= 0.055\ngini = 0.071\nsamples = 360\nvalue = [531, 10, 10]
\nclass = a'),
 Text(2072.5714285714284, 543.5999999999999, 'CO <= 0.565\ngini = 0.018\nsamples = 285\n
value = [431, 4, 0]\nclass = a'),
 Text(1992.8571428571427, 181.19999999999982, 'gini = 0.0\nsamples = 270\nvalue = [411,
```

```
0, 0]\nclass = a'),
 Text(2152.285714285714, 181.19999999999982, 'gini = 0.278\nsamples = 15\nvalue = [20,
4, 0]\nclass = a'),
 Text(2391.428571428571, 543.5999999999999, 'TOL <= 2.13\ngini = 0.247\nsamples = 75\nva
lue = [100, 6, 10]\nclass = a'),
 Text(2311.7142857142853, 181.19999999999982, 'gini = 0.455\nsamples = 22\nvalue = [23,
1, 10]\nclass = a'),
 Text(2471.142857142857, 181.19999999999982, 'gini = 0.115\nsamples = 53\nvalue = [77,
5, 0]\nclass = a'),
 Text(2869.7142857142853, 906.0, 'OXY <= 0.625\ngini = 0.599\nsamples = 4867\nvalue = [1
778, 1733, 4219]\nclass = c'),
 Text(2710.285714285714, 543.5999999999999, 'TCH <= 1.395\ngini = 0.628\nsamples = 1131
\nvalue = [837, 606, 342]\nclass = a'),
 Text(2630.5714285714284, 181.19999999999982, 'gini = 0.496\nsamples = 598\nvalue = [63
3, 128, 186]\nclass = a'),
 Text(2790.0, 181.19999999999982, 'gini = 0.581\nsamples = 533\nvalue = [204, 478, 156]
\nclass = b'),
 Text(3029.142857142857, 543.5999999999999, 'CO <= 0.245\ngini = 0.514\nsamples = 3736\n
value = [941, 1127, 3877]\nclass = c'),
 Text(2949.428571428571, 181.19999999999982, 'gini = 0.524\nsamples = 534\nvalue = [73,
534, 261]\nclass = b'),
 Text(3108.8571428571427, 181.19999999999982, 'gini = 0.45\nsamples = 3202\nvalue = [86
8, 593, 3616]\nclass = c'),
 Text(3826.2857142857138, 1268.4, 'CO <= 0.335\ngini = 0.495\nsamples = 3313\nvalue = [3
463, 626, 1097]\nclass = a'),
 Text(3507.428571428571, 906.0, 'TCH <= 1.395\ngini = 0.334\nsamples = 271\nvalue = [84,
339, 5]\nclass = b'),
 Text(3347.9999999999995, 543.5999999999999, 'TOL <= 5.02\ngini = 0.092\nsamples = 48\nv
alue = [80, 1, 3]\nclass = a'),
 Text(3268.285714285714, 181.19999999999982, 'gini = 0.0\nsamples = 33\nvalue = [62, 0,
0]\nclass = a'),
 Text(3427.7142857142853, 181.19999999999982, 'gini = 0.31\nsamples = 15\nvalue = [18,
1, 3]\nclass = a'),
 Text(3666.8571428571427, 543.5999999999999, 'NOx <= 74.95\ngini = 0.034\nsamples = 223
\nvalue = [4, 338, 2]\nclass = b'),
 Text(3587.142857142857, 181.19999999999982, 'gini = 0.217\nsamples = 26\nvalue = [3, 3
7, 2]\nclass = b'),
 Text(3746.5714285714284, 181.19999999999982, 'gini = 0.007\nsamples = 197\nvalue = [1,
301, 0]\nclass = b'),
 Text(4145.142857142857, 906.0, 'NMHC <= 0.265\ngini = 0.439\nsamples = 3042\nvalue = [3
379, 287, 1092]\nclass = a'),
 Text(3985.7142857142853, 543.5999999999999, 'NOx <= 69.74\ngini = 0.263\nsamples = 1239
\nvalue = [1671, 123, 168]\nclass = a'),
 Text(3905.9999999999995, 181.19999999999982, 'gini = 0.353\nsamples = 19\nvalue = [8, 2
7, 0]\nclass = b'),
 Text(4065.428571428571, 181.19999999999982, 'gini = 0.245\nsamples = 1220\nvalue = [166
3, 96, 168]\nclass = a'),
 Text(4304.571428571428, 543.5999999999999, 'TCH <= 1.515\ngini = 0.514\nsamples = 1803
\nvalue = [1708, 164, 924]\nclass = a'),
 Text(4224.857142857142, 181.19999999999982, 'gini = 0.314\nsamples = 471\nvalue = [576,
11, 125]\nclass = a'),
 Text(4384.285714285714, 181.19999999999982, 'gini = 0.553\nsamples = 1332\nvalue = [113
2, 153, 799]\nclass = a')]
```



Conclusion

Accuracy

linear regression

In [63]:
```
lr.score(x_test,y_test)
```

Out[63]: 0.16138975732905902

Ridge regression

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

Out[64]: 0.1613210457141202

Lasso regression

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

Out[66]: 0.014671192800835842

Elastic net regression

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

Out[67]: 0.06969624440569488

Logistic regression

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

Out[68]: 0.8146838030106512

Random forest

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

Out[69]: 0.8216732172936553

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