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("madrid_2002.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PXY
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.990002	NaN
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.85	20.980000	2.53
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.440001	NaN
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.180000	NaN
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.330002	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
217291	2002-11-01 00:00:00	4.16	1.14	NaN	NaN	NaN	81.080002	265.700012	NaN	7.21	36.750000	NaN
217292	2002-11-01 00:00:00	3.67	1.73	2.89	NaN	0.38	113.900002	373.100006	NaN	5.66	63.389999	NaN
217293	2002-11-01 00:00:00	1.37	0.58	1.17	2.37	0.15	65.389999	107.699997	1.30	9.11	9.640000	0.94
217294	2002-11-01 00:00:00	4.51	0.91	4.83	10.99	NaN	149.800003	202.199997	1.00	5.75	NaN	5.52
217295	2002-11-01 00:00:00	3.11	1.17	3.00	7.77	0.26	80.110001	180.300003	2.25	7.38	29.240000	3.35

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

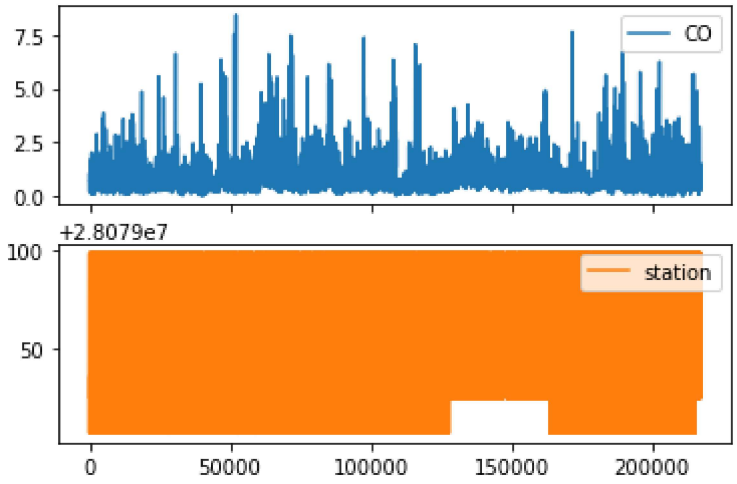
	CO	station
217271	1.30	28079099
217273	0.97	28079035
217293	0.58	28079024
217295	1.17	28079099

32381 rows × 2 columns

Line chart

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

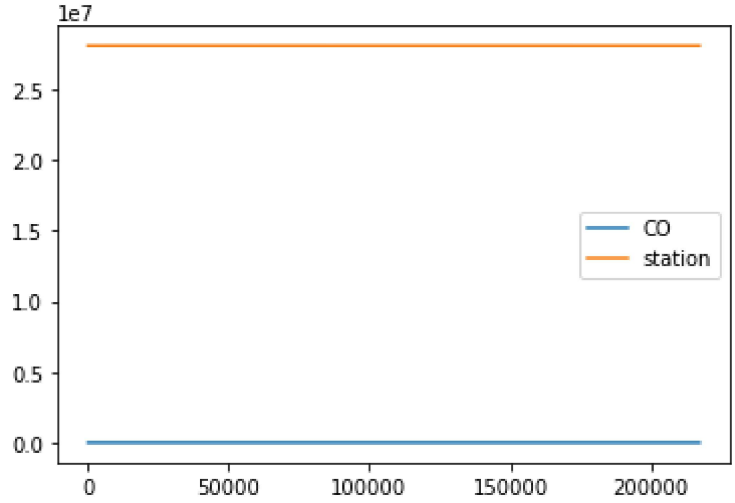
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

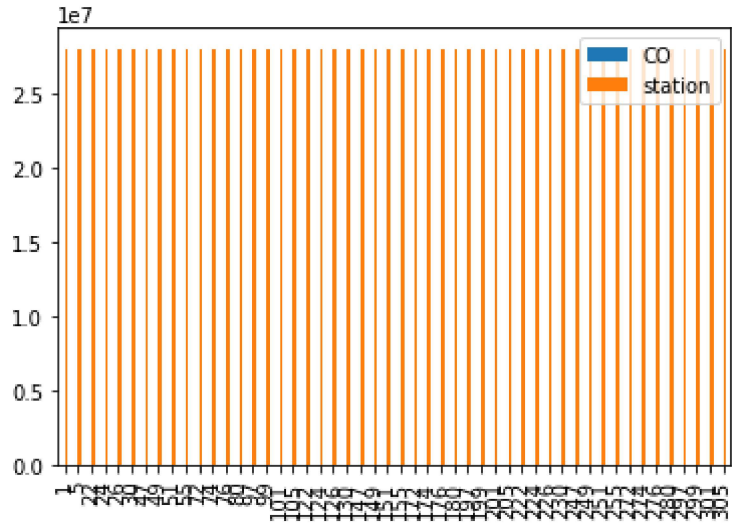


Bar chart

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

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

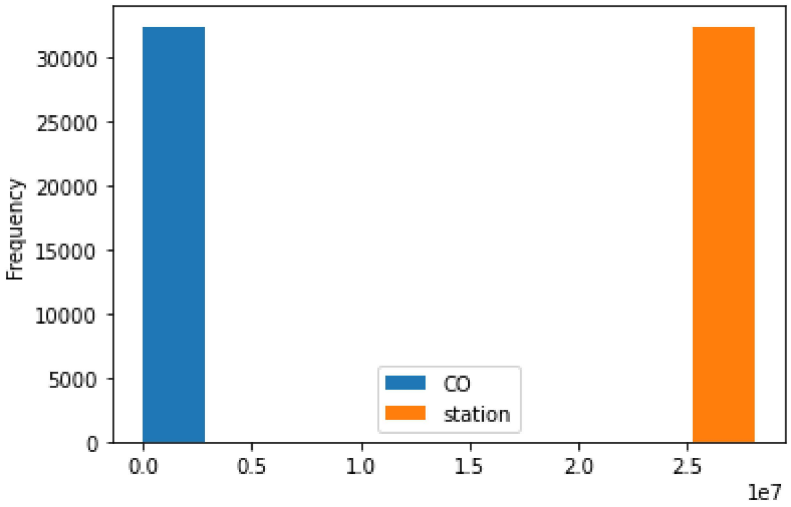
Out[10]: <AxesSubplot:>



Histogram

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

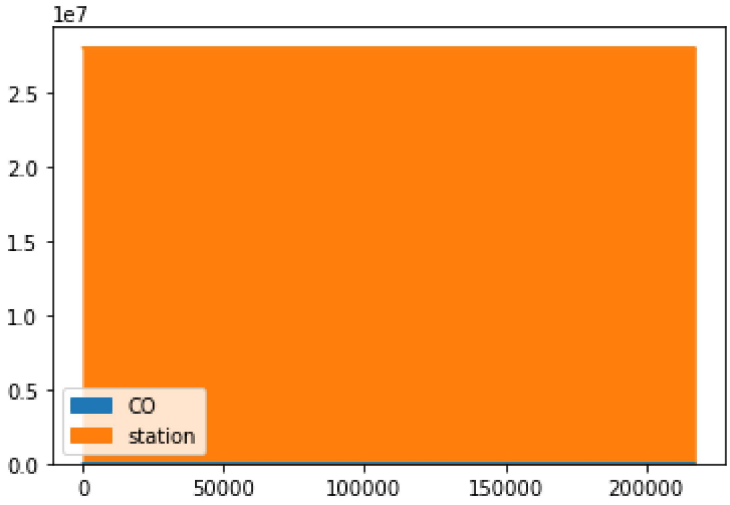
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

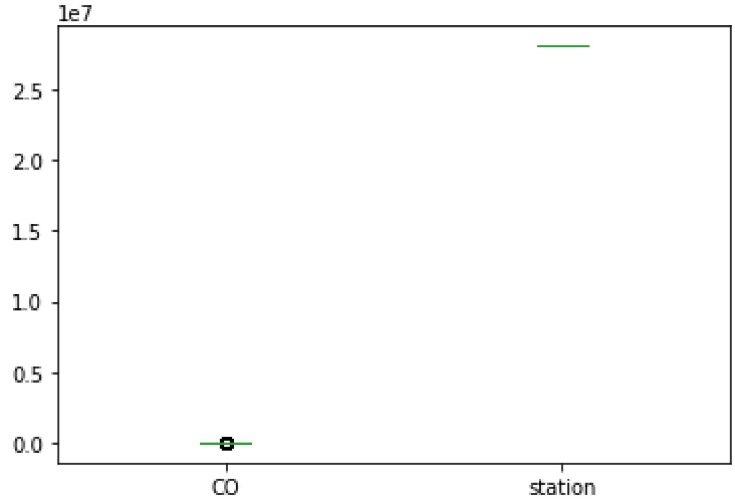
Out[12]: <AxesSubplot:>



Box chart

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

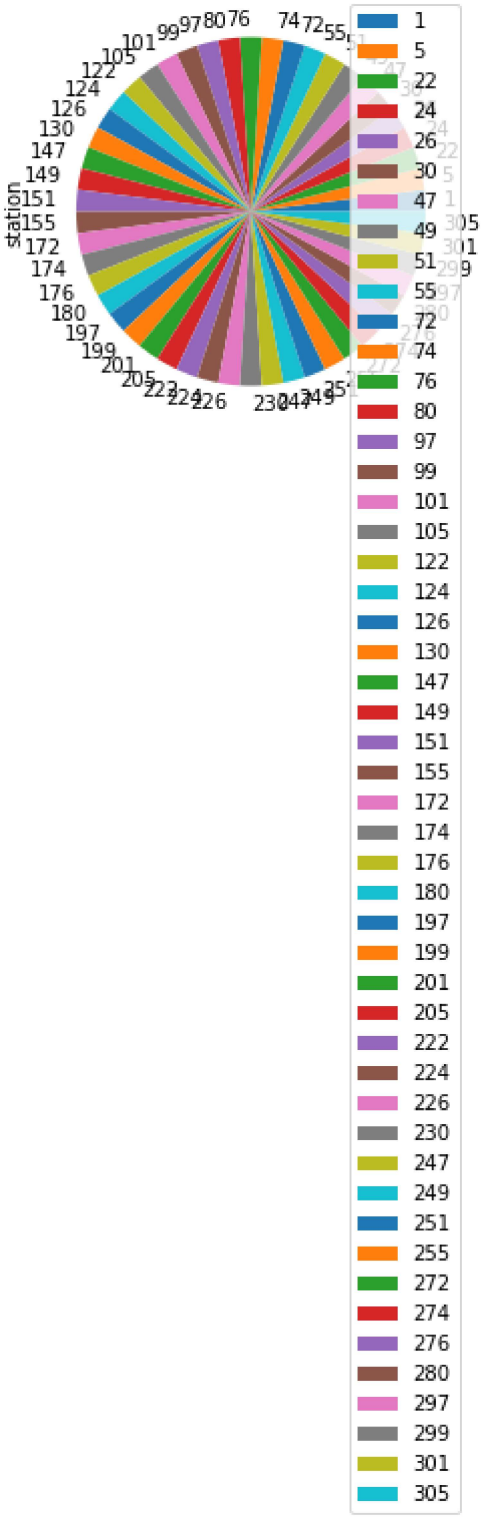
Out[13]: <AxesSubplot:>



Pie chart

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

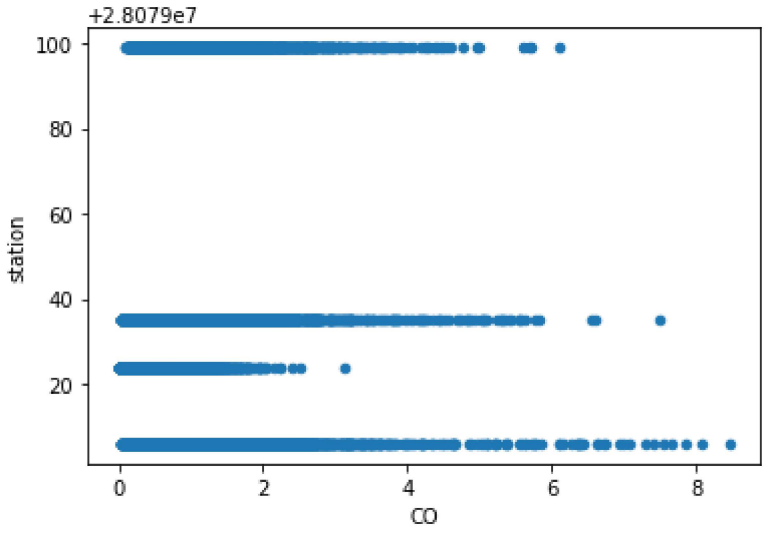
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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

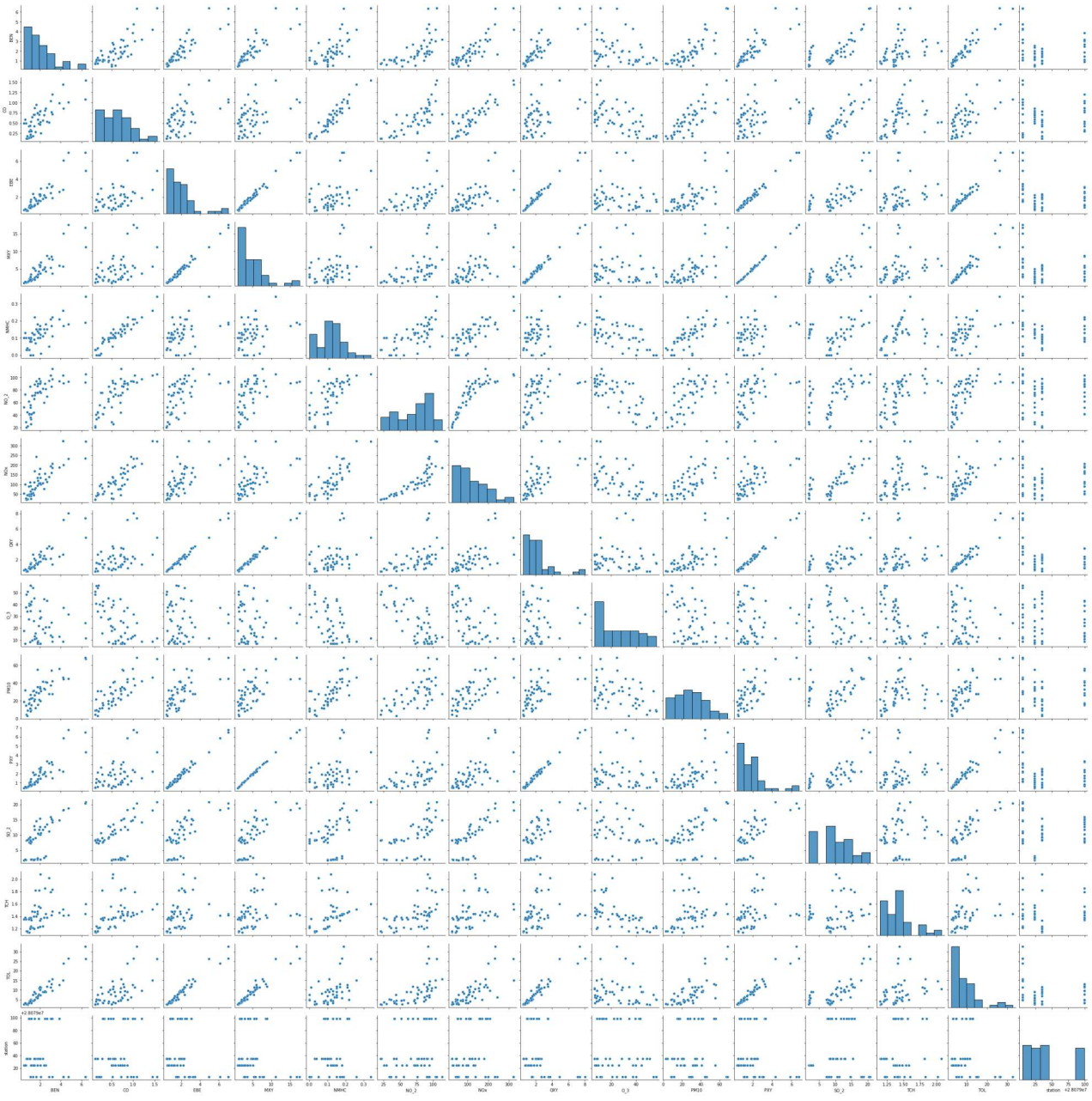
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
max	32.660000	8.460000	41.740002	99.879997	2.700000	263.600006	1336.00000

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])
```

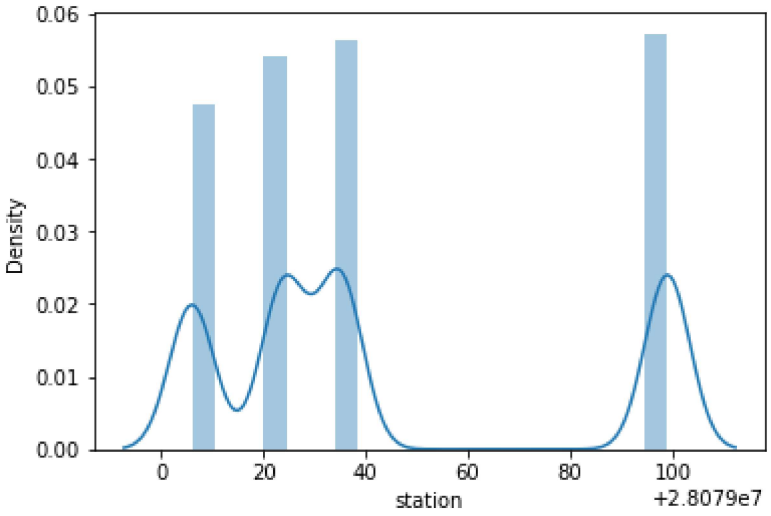
Out[19]: <seaborn.axisgrid.PairGrid at 0x19c559617c0>



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

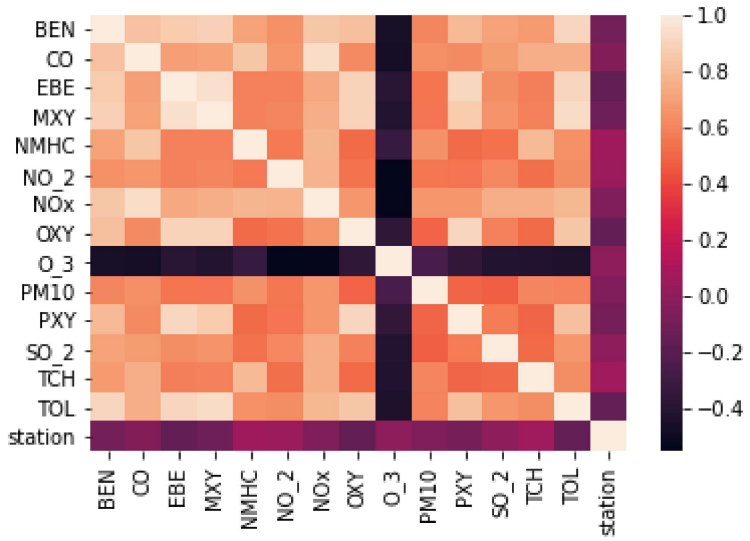
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28078987.25931952
```

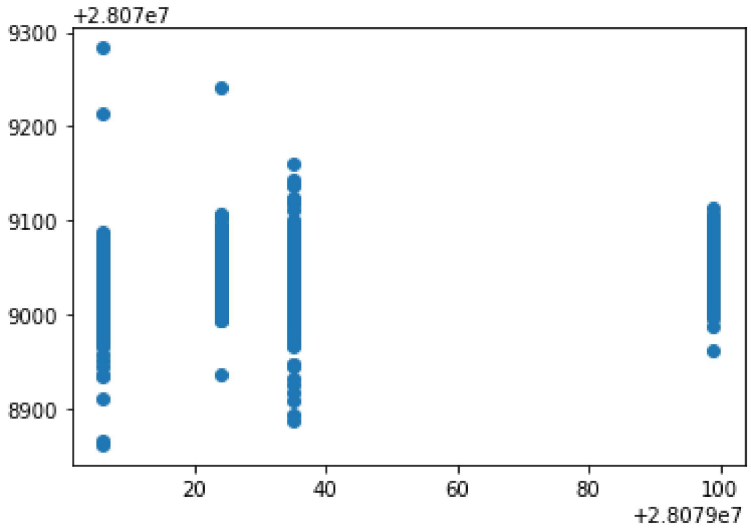
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.640497
CO	-12.020551
EBE	-11.612133
MXY	3.971244
NMHC	85.043934
NO_2	0.254878
NOx	-0.105815
OXY	-5.029007
O_3	-0.026584
PM10	-0.132277
PXY	7.836114
SO_2	0.584089
TCH	43.763759
TOL	-1.361998

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x19c651b7580>
```



ACCURACY

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

Out[28]: 0.19971411089471758

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

Out[29]: 0.19840597173570573

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]: 0.19953985630901738

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

Out[33]: 0.19818854070673386

Accuracy(Lasso)

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]:
```
0.055289238259287776
```

ElasticNet

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

Out[36]:
```
0.0611745974963912
```

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([ 0.85558994,  0.        , -2.97028182,  1.46884267,  0.22377329,
        0.22791616, -0.02661111, -2.32895622, -0.01845213, -0.        ,
        2.27313903,  0.37553058,  1.12817032, -1.0911772 ])
```

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

Out[39]:
```
28079038.239885956
```

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

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

Out[41]:
```
0.10599972753833409
```

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)))
```

```
28.380169156675358
1105.4055720042734
33.247640096768876
```

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]:
```
0.8480899292795158
```

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

Out[54]:
```
2.5638972732451705e-10
```

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

Out[55]:
```
array([[2.56389727e-10, 3.44199742e-71, 1.00000000e+00, 1.43898646e-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]:
```
0.7711550339715874
```

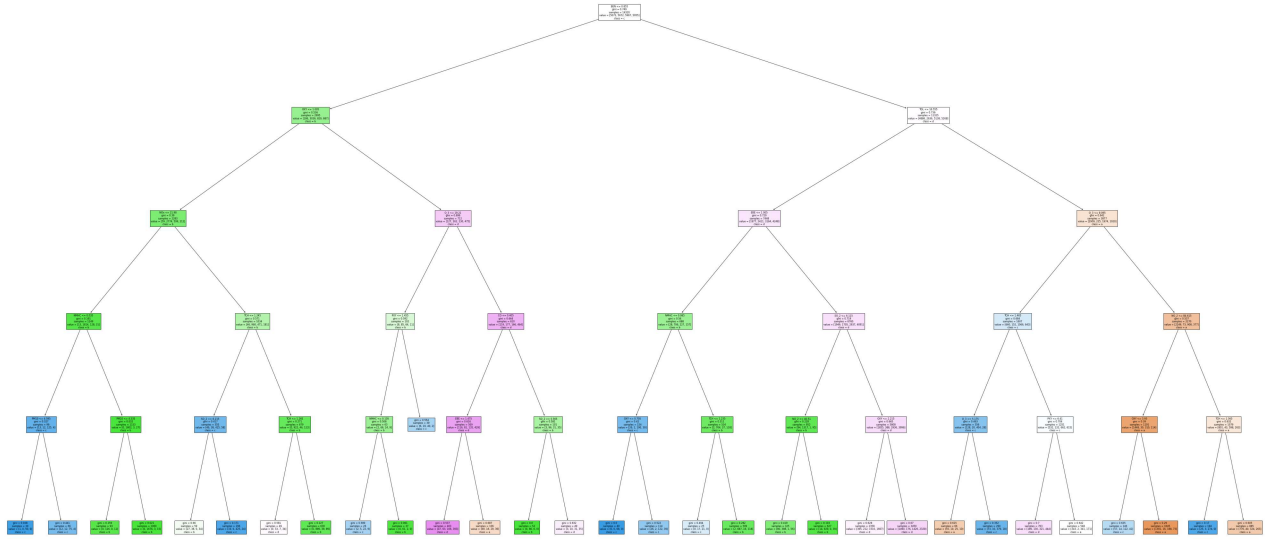
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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, 'BEN <= 0.855\ngini = 0.749\nsamples = 14320\nvalue = [5072, 5672,
5967, 5955]\nclass = c'),
 Text(1097.4, 1630.8000000000002, 'OXY <= 1.005\ngini = 0.536\nsamples = 2995\nvalue =
[186, 3036, 829, 687]\nclass = b'),
 Text(595.2, 1268.4, 'NOx <= 21.98\ngini = 0.39\nsamples = 2283\nvalue = [59, 2774, 599,
212]\nclass = b'),
 Text(297.6, 906.0, 'NMHC <= 0.035\ngini = 0.161\nsamples = 1249\nvalue = [13, 1814, 12
8, 31]\nclass = b'),
 Text(148.8, 543.5999999999999, 'PM10 <= 4.595\ngini = 0.327\nsamples = 96\nvalue = [13,
12, 125, 4]\nclass = c'),
 Text(74.4, 181.19999999999982, 'gini = 0.038\nsamples = 30\nvalue = [1, 0, 50, 0]\nclas
s = c'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.441\nsamples = 66\nvalue = [12,
12, 75, 4]\nclass = c'),
 Text(446.40000000000003, 543.5999999999999, 'PM10 <= 4.535\ngini = 0.032\nsamples = 115
3\nvalue = [0, 1802, 3, 27]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.159\nsamples = 93\nvalue = [0, 126, 0, 12]\nc
lass = b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.021\nsamples = 1060\nvalue = [0,
1676, 3, 15]\nclass = b'),
 Text(892.8000000000001, 906.0, 'TCH <= 1.245\ngini = 0.571\nsamples = 1034\nvalue = [4
6, 960, 471, 181]\nclass = b'),
 Text(744.0, 543.5999999999999, 'SO_2 <= 6.215\ngini = 0.417\nsamples = 355\nvalue = [4
6, 38, 425, 58]\nclass = c'),
 Text(669.6, 181.19999999999982, 'gini = 0.66\nsamples = 59\nvalue = [27, 38, 0, 34]\ncl
ass = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.171\nsamples = 296\nvalue = [19,
0, 425, 24]\nclass = c'),
 Text(1041.6000000000001, 543.5999999999999, 'TCH <= 1.265\ngini = 0.271\nsamples = 679
\nvalue = [0, 922, 46, 123]\nclass = b'),
 Text(967.2, 181.19999999999982, 'gini = 0.581\nsamples = 49\nvalue = [0, 33, 7, 34]\ncl
ass = d'),
 Text(1116.0, 181.19999999999982, 'gini = 0.227\nsamples = 630\nvalue = [0, 889, 39, 89]
\nclass = b'),
 Text(1599.6000000000001, 1268.4, 'O_3 <= 19.21\ngini = 0.696\nsamples = 712\nvalue = [1
27, 262, 230, 475]\nclass = d'),
 Text(1413.6000000000001, 906.0, 'PXY <= 1.455\ngini = 0.592\nsamples = 102\nvalue = [8,
85, 64, 11]\nclass = b'),
 Text(1339.2, 543.5999999999999, 'NMHC <= 0.105\ngini = 0.508\nsamples = 63\nvalue = [2,
66, 24, 9]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.589\nsamples = 26\nvalue = [2,
5, 22, 9]\nclass = c'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.061\nsamples = 37\nvalue = [0, 6
1, 2, 0]\nclass = b'),
 Text(1488.0, 543.5999999999999, 'gini = 0.554\nsamples = 39\nvalue = [6, 19, 40, 2]\ncl
ass = c'),
 Text(1785.6000000000001, 906.0, 'CO <= 0.405\ngini = 0.664\nsamples = 610\nvalue = [11
9, 177, 166, 464]\nclass = d'),
 Text(1636.8000000000002, 543.5999999999999, 'EBE <= 1.475\ngini = 0.616\nsamples = 509
\nvalue = [116, 81, 135, 429]\nclass = d'),
 Text(1562.4, 181.19999999999982, 'gini = 0.537\nsamples = 403\nvalue = [47, 63, 106, 39
3]\nclass = d'),
 Text(1711.2, 181.19999999999982, 'gini = 0.687\nsamples = 106\nvalue = [69, 18, 29, 36]
\nclass = a'),
 Text(1934.4, 543.5999999999999, 'SO_2 <= 5.565\ngini = 0.581\nsamples = 101\nvalue =
[3, 96, 31, 35]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.0\nsamples = 52\nvalue = [0, 86,
0, 0]\nclass = b'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.632\nsamples = 49\nvalue = [3, 1
0, 31, 35]\nclass = d'),
 Text(3273.6000000000004, 1630.8000000000002, 'TOL <= 16.555\ngini = 0.736\nsamples = 11
325\nvalue = [4886, 2636, 5138, 5268]\nclass = d'),
 Text(2678.4, 1268.4, 'EBE <= 1.005\ngini = 0.729\nsamples = 7448\nvalue = [1977, 2411,
3164, 4248]\nclass = d'),
 Text(2380.8, 906.0, 'NMHC <= 0.065\ngini = 0.54\nsamples = 688\nvalue = [28, 706, 227,
```

```
157]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'OXY <= 0.705\ngini = 0.42\nsamples = 154\nvalue = [26,
2, 190, 39]\nclass = c'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.0\nsamples = 40\nvalue = [0, 0,
68, 0]\nclass = c'),
 Text(2306.4, 181.19999999999982, 'gini = 0.522\nsamples = 114\nvalue = [26, 2, 122, 39]
\nclass = c'),
 Text(2529.6000000000004, 543.5999999999999, 'TCH <= 1.235\ngini = 0.311\nsamples = 534
\nvalue = [2, 704, 37, 118]\nclass = b'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.494\nsamples = 25\nvalue = [0, 1
7, 21, 0]\nclass = c'),
 Text(2604.0, 181.19999999999982, 'gini = 0.282\nsamples = 509\nvalue = [2, 687, 16, 11
8]\nclass = b'),
 Text(2976.0, 906.0, 'SO_2 <= 6.115\ngini = 0.719\nsamples = 6760\nvalue = [1949, 1705,
2937, 4091]\nclass = d'),
 Text(2827.2000000000003, 543.5999999999999, 'NO_2 <= 40.51\ngini = 0.228\nsamples = 952
\nvalue = [94, 1317, 1, 95]\nclass = b'),
 Text(2752.8, 181.19999999999982, 'gini = 0.419\nsamples = 325\nvalue = [80, 388, 1, 56]
\nclass = b'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.103\nsamples = 627\nvalue = [14,
929, 0, 39]\nclass = b'),
 Text(3124.8, 543.5999999999999, 'OXY <= 2.215\ngini = 0.665\nsamples = 5808\nvalue = [1
855, 388, 2936, 3996]\nclass = d'),
 Text(3050.4, 181.19999999999982, 'gini = 0.628\nsamples = 2358\nvalue = [365, 212, 151
0, 1667]\nclass = d'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.67\nsamples = 3450\nvalue = [149
0, 176, 1426, 2329]\nclass = d'),
 Text(3868.8, 1268.4, 'O_3 <= 8.095\ngini = 0.642\nsamples = 3877\nvalue = [2909, 225, 1
974, 1020]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'TCH <= 1.465\ngini = 0.684\nsamples = 1607\nvalue = [6
60, 152, 1066, 643]\nclass = c'),
 Text(3422.4, 543.5999999999999, 'O_3 <= 5.175\ngini = 0.463\nsamples = 356\nvalue = [12
8, 20, 404, 28]\nclass = c'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.615\nsamples = 66\nvalue = [55,
10, 25, 10]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.352\nsamples = 290\nvalue = [73, 10, 379, 1
8]\nclass = c'),
 Text(3720.0000000000005, 543.5999999999999, 'PXY <= 6.41\ngini = 0.704\nsamples = 1251
\nvalue = [532, 132, 662, 615]\nclass = c'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.7\nsamples = 703\nvalue = [189,
130, 321, 444]\nclass = d'),
 Text(3794.4, 181.19999999999982, 'gini = 0.642\nsamples = 548\nvalue = [343, 2, 341, 17
1]\nclass = a'),
 Text(4166.400000000001, 906.0, 'NO_2 <= 84.615\ngini = 0.537\nsamples = 2270\nvalue =
[2249, 73, 908, 377]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'OXY <= 3.93\ngini = 0.39\nsamples = 1191\n
value = [1448, 30, 310, 114]\nclass = a'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.635\nsamples = 146\nvalue = [57,
14, 122, 41]\nclass = c'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.29\nsamples = 1045\nvalue = [139
1, 16, 188, 73]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'TCH <= 1.365\ngini = 0.632\nsamples = 1079
\nvalue = [801, 43, 598, 263]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.17\nsamples = 194\nvalue = [25, 3, 274, 0]\n
class = c'),
 Text(4389.6, 181.19999999999982, 'gini = 0.605\nsamples = 885\nvalue = [776, 40, 324, 2
63]\nclass = a')]
```



Conclusion

Accuracy

In [64]:
```
lr.score(x_train,y_train)
```

Out[64]: 0.19840597173570573

In [65]:
```
rr.score(x_train,y_train)
```

Out[65]: 0.19818854070673386

In [66]:
```
la.score(x_train,y_train)
```

Out[66]: 0.055289238259287776

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

Out[67]: 0.10599972753833409

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

Out[68]: 0.8480899292795158

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

Out[69]: 0.7711550339715874

Logistic Regression is suitable for this dataset