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_2001.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PX
0	01-08-2001 01:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	105.000000	NaN
1	01-08-2001 01:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.599998	1.7
2	01-08-2001 01:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.099998	NaN
3	01-08-2001 01:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.779999	NaN
4	01-08-2001 01:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.180000	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
217867	01-04-2001 00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	47.880001	NaN
217868	01-04-2001 00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	26.809999	NaN
217869	01-04-2001 00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.770000	0.6

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PX
217870	01-04-2001 00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	37.889999	4.3
217871	01-04-2001 00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.369999	4.9

217872 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: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

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

Out[6]:

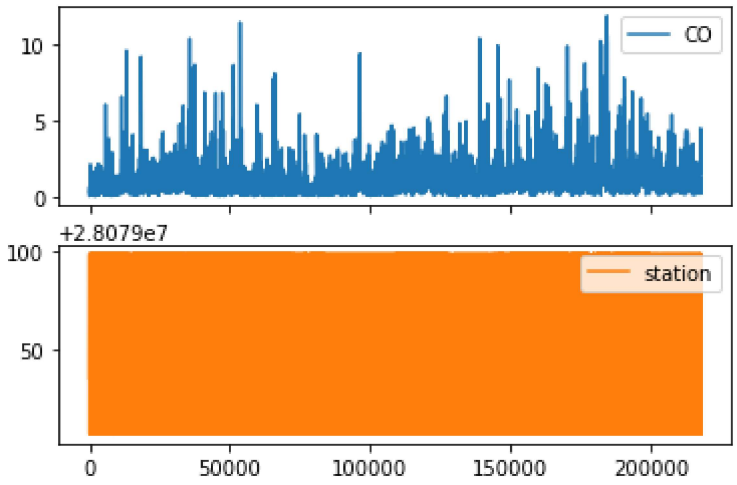
	CO	station
1	0.34	28079035
5	0.63	28079006
21	0.43	28079024
23	0.34	28079099
25	0.06	28079035
...	...	...
217829	4.48	28079006
217847	2.65	28079099
217849	1.22	28079035
217853	1.83	28079006
217871	1.62	28079099

29669 rows × 2 columns

Line chart

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

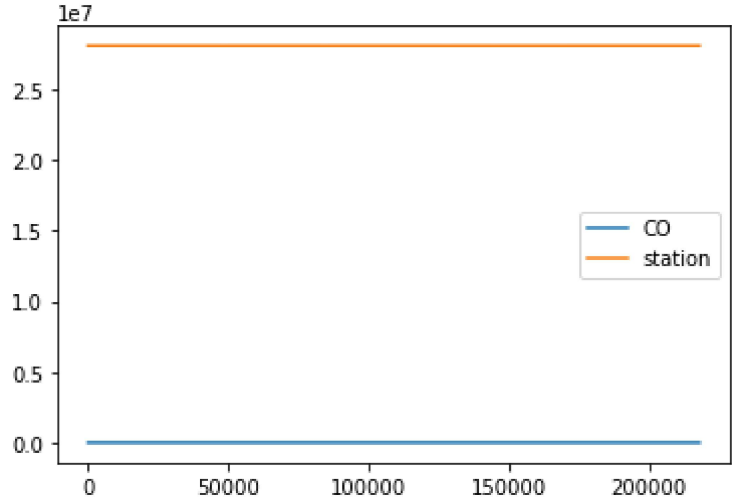
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

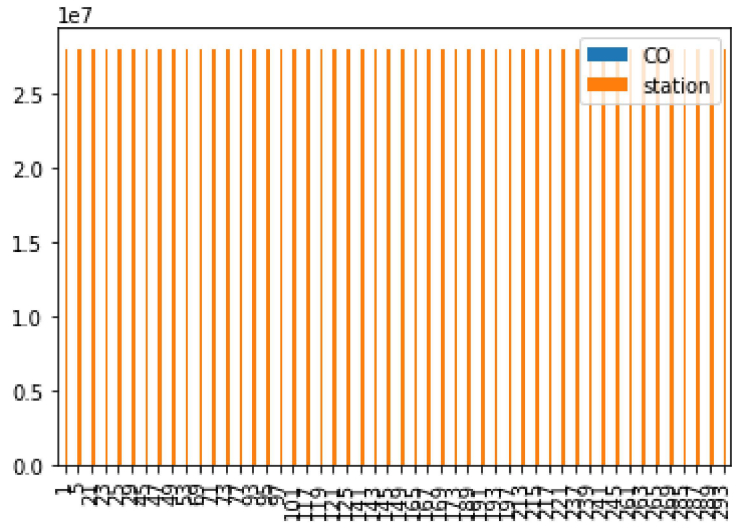


Bar chart

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

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

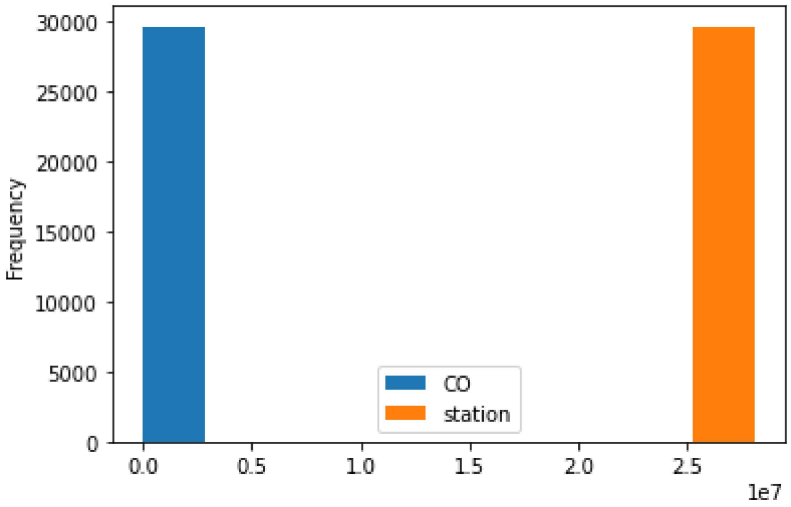
Out[10]: <AxesSubplot:>



Histogram

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

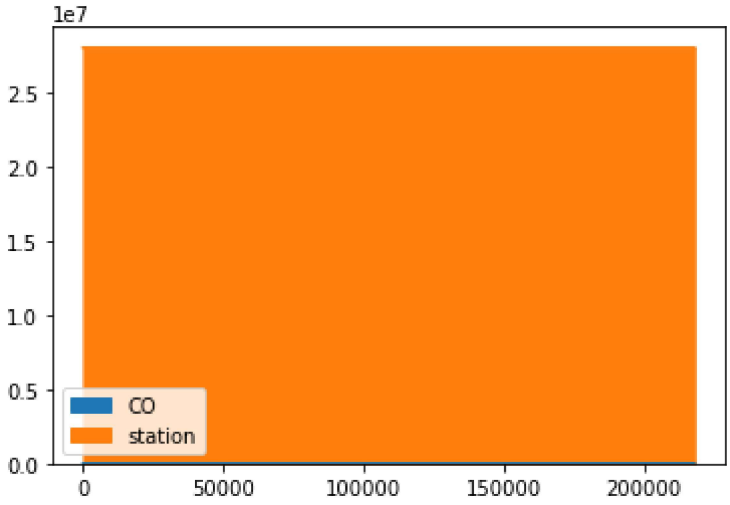
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

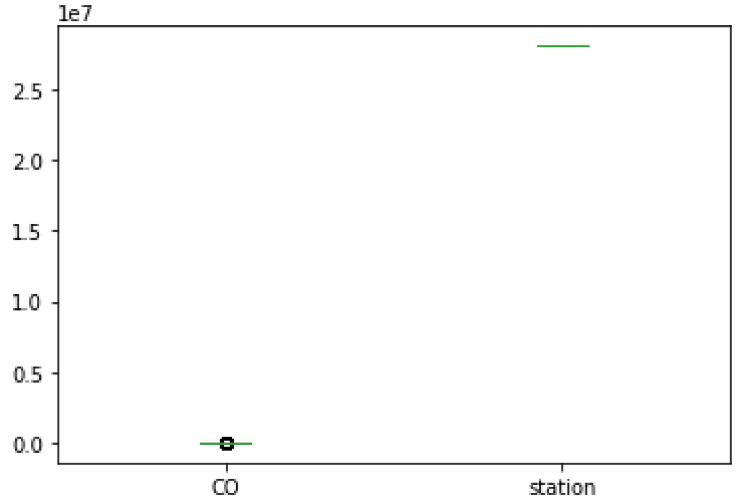
Out[12]: <AxesSubplot:>



Box chart

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

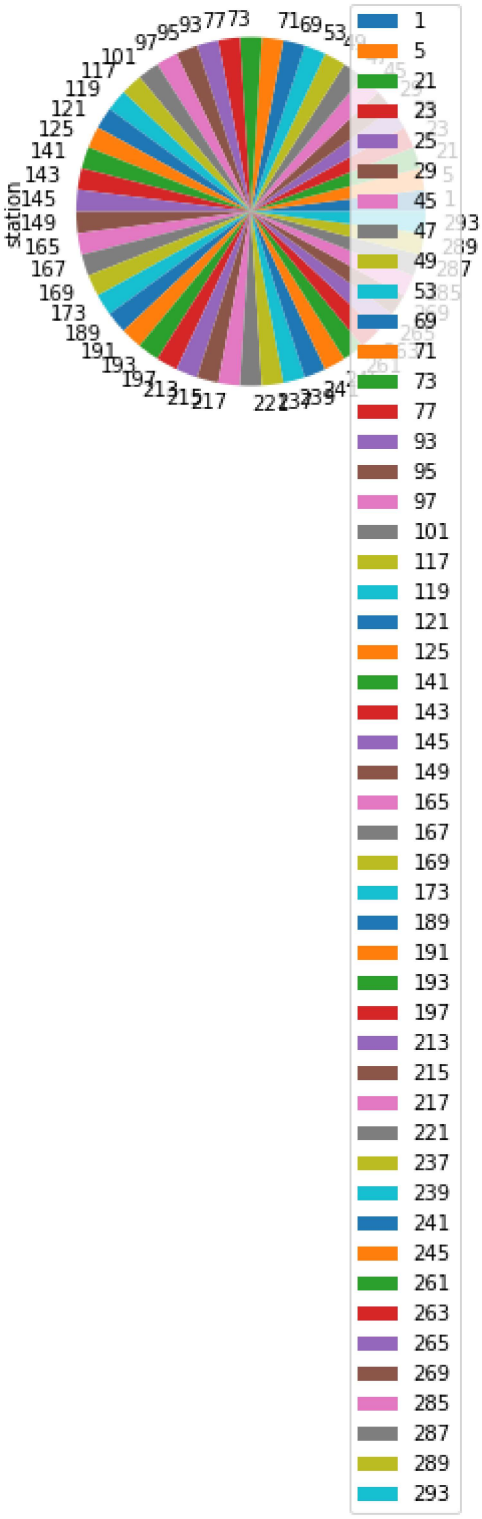
Out[13]: <AxesSubplot:>



Pie chart

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

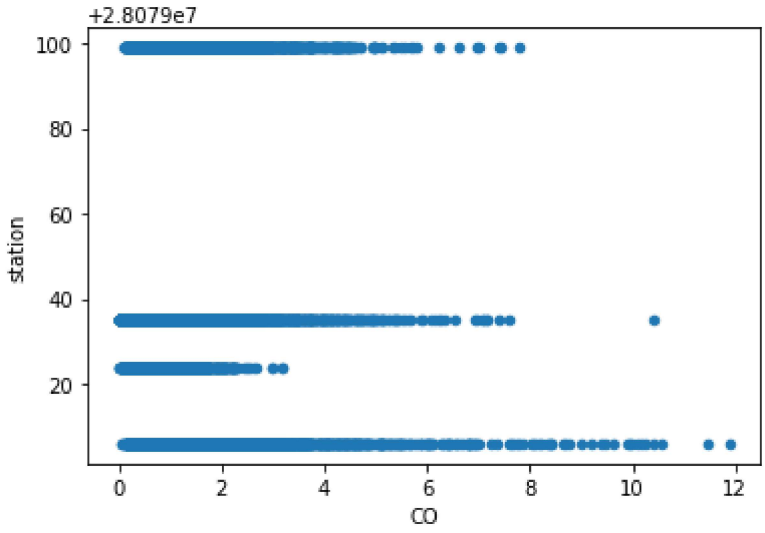
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 29669 entries, 1 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     29669 non-null  object 
 1   BEN      29669 non-null  float64
 2   CO       29669 non-null  float64
 3   EBE      29669 non-null  float64
 4   MXY      29669 non-null  float64
 5   NMHC     29669 non-null  float64
 6   NO_2     29669 non-null  float64
 7   NOx      29669 non-null  float64
 8   OXY      29669 non-null  float64
 9   O_3      29669 non-null  float64
 10  PM10     29669 non-null  float64
 11  PXY      29669 non-null  float64
 12  SO_2     29669 non-null  float64
 13  TCH      29669 non-null  float64
 14  TOL      29669 non-null  float64
 15  station  29669 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 3.8+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.000000	29669.00000
mean	3.361895	1.005413	3.580229	8.113086	0.195222	67.652292	163.00485
std	3.176669	0.863135	3.744496	7.909701	0.192585	34.003120	147.49177
min	0.100000	0.000000	0.140000	0.210000	0.000000	1.180000	1.28000
25%	1.280000	0.470000	1.390000	3.040000	0.080000	44.299999	68.08999
50%	2.510000	0.760000	2.600000	5.830000	0.140000	64.449997	123.69999
75%	4.420000	1.270000	4.580000	10.640000	0.250000	86.540001	213.19999

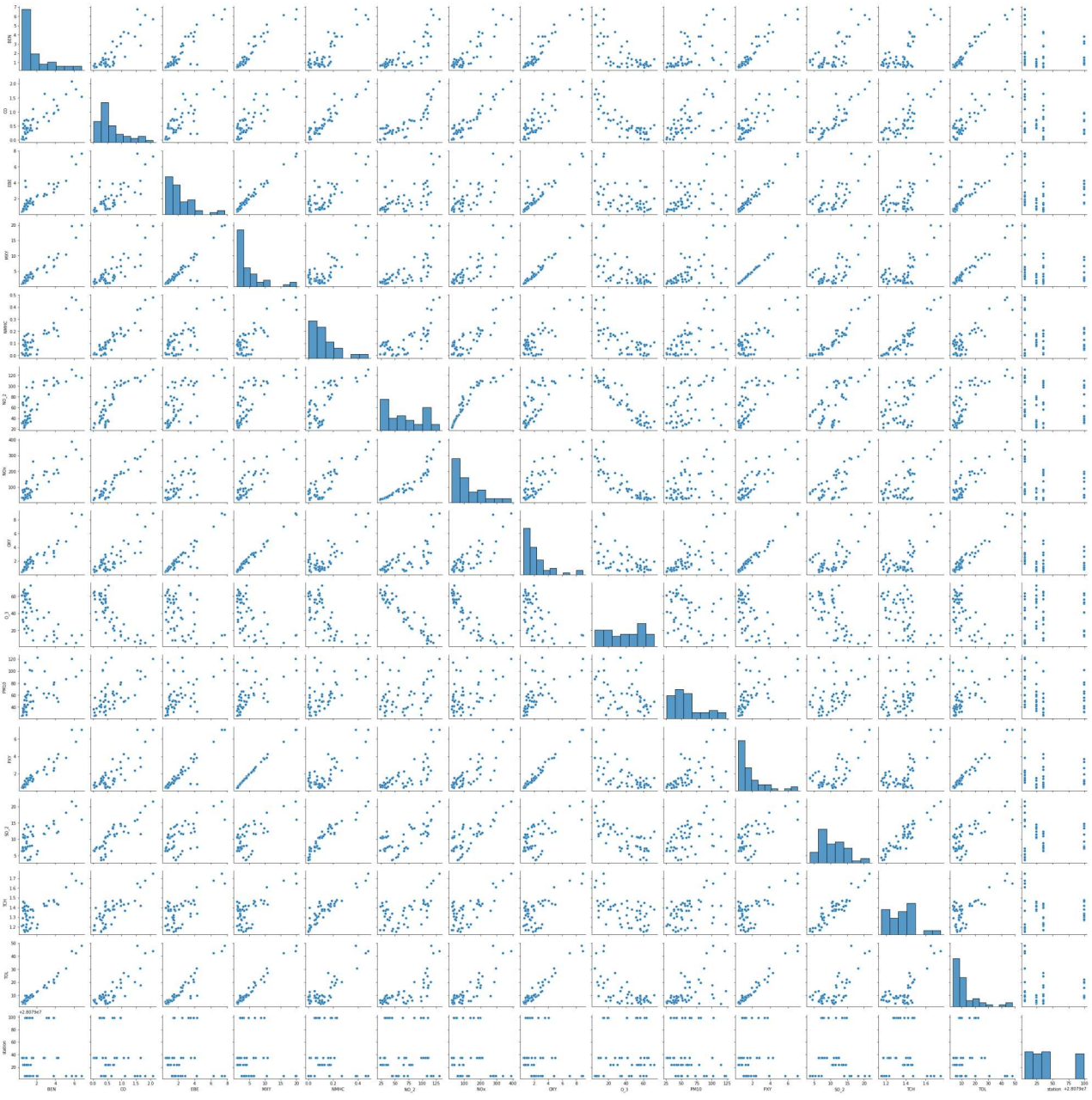
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
max	54.560001	11.890000	77.260002	150.600006	2.880000	292.700012	1940.00000

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])
```

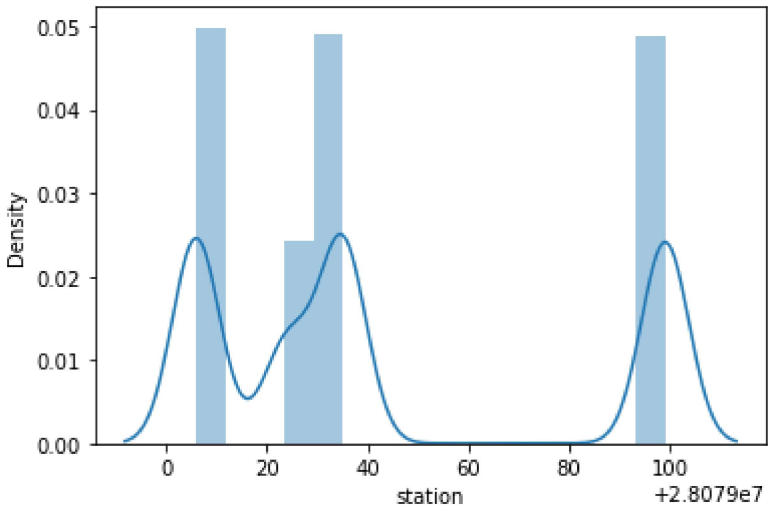
Out[19]: <seaborn.axisgrid.PairGrid at 0x1222eafe190>



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

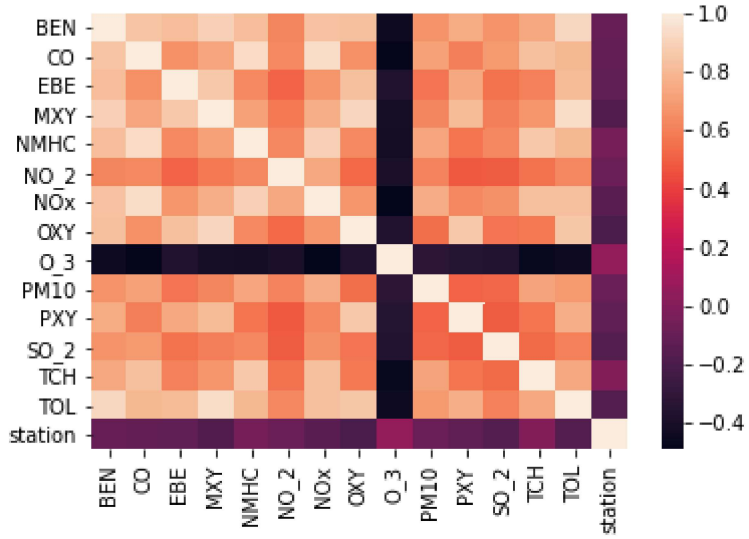
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28079006.360363092
```

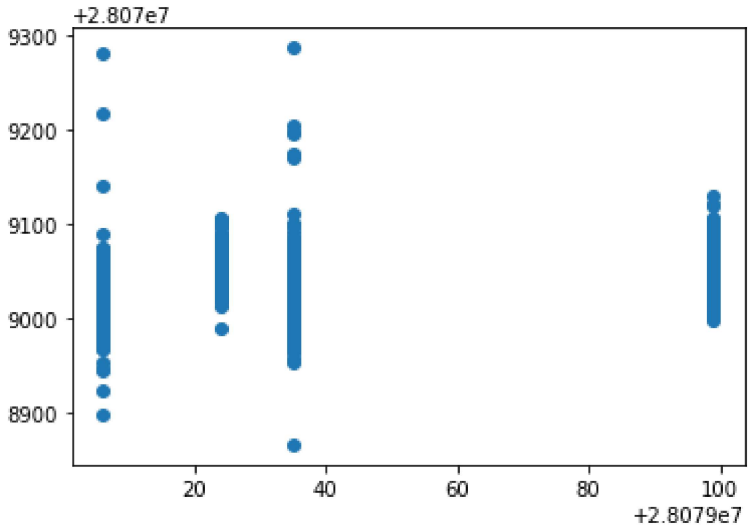
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	7.413989
CO	-15.923675
EBE	0.753554
MXY	-0.278728
NMHC	77.942437
NO_2	0.117329
NOx	-0.085042
OXY	-3.321241
O_3	-0.021930
PM10	-0.047747
PXY	1.866746
SO_2	-0.302305
TCH	37.708288
TOL	-1.217249

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1223e3b9a90>
```



ACCURACY

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

Out[28]: 0.15413624199466525

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

Out[29]: 0.16861010524615405

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]: 0.1534098188973947

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

Out[33]: 0.16838829498437113

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]:
```
0.0427417811160149
```

Accuracy(Lasso)

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

Out[36]:
```
0.032119074438977324
```

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([ 4.9316136 ,  0.        ,  0.70385615, -0.35548025,  0.03323425,
        0.06641308, -0.0359027 , -2.58965433, -0.02803092,  0.08564567,
        1.17100148, -0.32354108,  1.19636252, -0.68553536])
```

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

Out[39]:
```
28079048.901556093
```

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

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

Out[41]:
```
0.09345029566665797
```

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)))
```

```
30.33471531909616
1214.1266076379716
34.844319589252585
```

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]: (29669, 14)

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

Out[46]: (29669,)

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]:
```
0.8087566146482861
```

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

Out[54]:
```
1.8247648812614132e-43
```

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

Out[55]:
```
array([[1.82476488e-43, 2.40167593e-56, 9.99998565e-01, 1.43549338e-06]])
```

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]:
```
0.7278505392912172
```

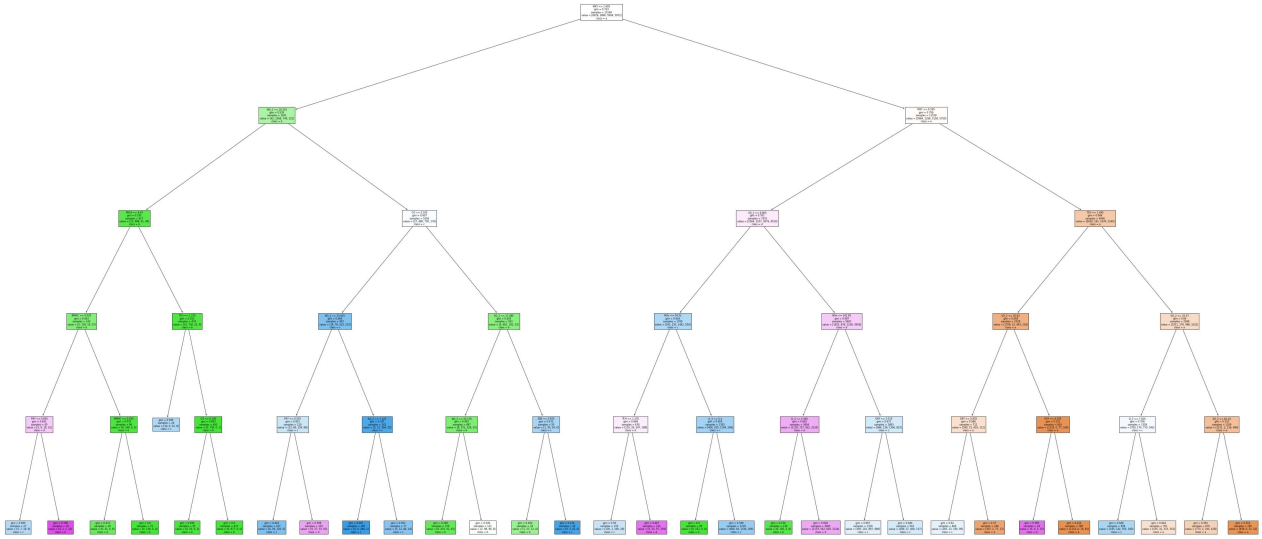
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(2129.7000000000003, 1993.2, 'MXY <= 1.605\ngini = 0.733\nsamples = 13149\nvalue =
[6026, 2866, 5904, 5972]\nclass = a'),
 Text(985.8000000000001, 1630.8000000000002, 'NO_2 <= 18.325\ngini = 0.539\nsamples = 16
21\nvalue = [42, 1568, 746, 222]\nclass = b'),
 Text(483.6, 1268.4, 'PM10 <= 8.67\ngini = 0.191\nsamples = 617\nvalue = [15, 888, 41, 4
6]\nclass = b'),
 Text(297.6, 906.0, 'NMHC <= 0.035\ngini = 0.443\nsamples = 143\nvalue = [5, 158, 19, 3
7]\nclass = b'),
 Text(148.8, 543.5999999999999, 'PXY <= 0.635\ngini = 0.651\nsamples = 45\nvalue = [5,
9, 19, 31]\nclass = d'),
 Text(74.4, 181.19999999999982, 'gini = 0.665\nsamples = 27\nvalue = [5, 7, 18, 6]\nclas
s = c'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.196\nsamples = 18\nvalue = [0,
2, 1, 25]\nclass = d'),
 Text(446.40000000000003, 543.5999999999999, 'NMHC <= 0.055\ngini = 0.074\nsamples = 98
\nvalue = [0, 149, 0, 6]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.272\nsamples = 26\nvalue = [0, 31, 0, 6]\ncla
ss = b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 72\nvalue = [0, 118,
0, 0]\nclass = b'),
 Text(669.6, 906.0, 'TCH <= 1.235\ngini = 0.102\nsamples = 474\nvalue = [10, 730, 22, 9]
\nclass = b'),
 Text(595.2, 543.5999999999999, 'gini = 0.595\nsamples = 24\nvalue = [10, 0, 22, 8]\ncla
ss = c'),
 Text(744.0, 543.5999999999999, 'CO <= 0.185\ngini = 0.003\nsamples = 450\nvalue = [0, 7
30, 0, 1]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.036\nsamples = 35\nvalue = [0, 53, 0, 1]\ncla
ss = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 415\nvalue = [0, 67
7, 0, 0]\nclass = b'),
 Text(1488.0, 1268.4, 'CO <= 0.345\ngini = 0.607\nsamples = 1004\nvalue = [27, 680, 705,
176]\nclass = c'),
 Text(1190.4, 906.0, 'NO_2 <= 33.035\ngini = 0.464\nsamples = 452\nvalue = [18, 78, 523,
123]\nclass = c'),
 Text(1041.6000000000001, 543.5999999999999, 'PXY <= 0.535\ngini = 0.651\nsamples = 210
\nvalue = [13, 66, 159, 98]\nclass = c'),
 Text(967.2, 181.19999999999982, 'gini = 0.424\nsamples = 107\nvalue = [4, 39, 126, 5]\n
class = c'),
 Text(1116.0, 181.19999999999982, 'gini = 0.598\nsamples = 103\nvalue = [9, 27, 33, 93]
\nclass = d'),
 Text(1339.2, 543.5999999999999, 'SO_2 <= 7.125\ngini = 0.191\nsamples = 242\nvalue =
[5, 12, 364, 25]\nclass = c'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.007\nsamples = 165\nvalue = [0,
0, 280, 1]\nclass = c'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.501\nsamples = 77\nvalue = [5, 1
2, 84, 24]\nclass = c'),
 Text(1785.6000000000001, 906.0, 'SO_2 <= 17.485\ngini = 0.443\nsamples = 552\nvalue =
[9, 602, 182, 53]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'NO_2 <= 53.175\ngini = 0.402\nsamples = 49
7\nvalue = [8, 572, 128, 53]\nclass = b'),
 Text(1562.4, 181.19999999999982, 'gini = 0.266\nsamples = 376\nvalue = [6, 474, 32, 45]
\nclass = b'),
 Text(1711.2, 181.19999999999982, 'gini = 0.546\nsamples = 121\nvalue = [2, 98, 96, 8]\n
class = b'),
 Text(1934.4, 543.5999999999999, 'EBE <= 0.925\ngini = 0.472\nsamples = 55\nvalue = [1,
30, 54, 0]\nclass = c'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.454\nsamples = 29\nvalue = [1, 2
7, 12, 0]\nclass = b'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.124\nsamples = 26\nvalue = [0,
3, 42, 0]\nclass = c'),
 Text(3273.6000000000004, 1630.8000000000002, 'MXY <= 9.165\ngini = 0.706\nsamples = 115
28\nvalue = [5984, 1298, 5158, 5750]\nclass = a'),
 Text(2678.4, 1268.4, 'SO_2 <= 8.805\ngini = 0.701\nsamples = 7432\nvalue = [2364, 1107,
3679, 4510]\nclass = d'),
 Text(2380.8, 906.0, 'NOx <= 54.52\ngini = 0.644\nsamples = 1765\nvalue = [541, 231, 144
```

```
1, 554]\nclass = c'),
 Text(2232.0, 543.5999999999999, 'TCH <= 1.235\ngini = 0.664\nsamples = 430\nvalue = [13
5, 26, 247, 288]\nclass = d'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.56\nsamples = 216\nvalue = [130,
1, 190, 29]\nclass = c'),
 Text(2306.4, 181.19999999999982, 'gini = 0.407\nsamples = 214\nvalue = [5, 25, 57, 259]
\nclass = d'),
 Text(2529.6000000000004, 543.5999999999999, 'O_3 <= 5.3\ngini = 0.603\nsamples = 1335\n
value = [406, 205, 1194, 266]\nclass = c'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 96\nvalue = [0, 15
1, 0, 0]\nclass = b'),
 Text(2604.0, 181.19999999999982, 'gini = 0.549\nsamples = 1239\nvalue = [406, 54, 1194,
266]\nclass = c'),
 Text(2976.0, 906.0, 'NOx <= 142.65\ngini = 0.687\nsamples = 5667\nvalue = [1823, 876, 2
238, 3956]\nclass = d'),
 Text(2827.2000000000003, 543.5999999999999, 'O_3 <= 5.595\ngini = 0.645\nsamples = 3804
\nvalue = [1139, 737, 932, 3133]\nclass = d'),
 Text(2752.8, 181.19999999999982, 'gini = 0.159\nsamples = 139\nvalue = [6, 195, 3, 9]\n
class = b'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.628\nsamples = 3665\nvalue = [11
33, 542, 929, 3124]\nclass = d'),
 Text(3124.8, 543.5999999999999, 'OXY <= 3.515\ngini = 0.671\nsamples = 1863\nvalue = [6
84, 139, 1306, 823]\nclass = c'),
 Text(3050.4, 181.19999999999982, 'gini = 0.667\nsamples = 1320\nvalue = [395, 102, 897,
686]\nclass = c'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.644\nsamples = 543\nvalue = [28
9, 37, 409, 137]\nclass = c'),
 Text(3868.8, 1268.4, 'TCH <= 1.495\ngini = 0.604\nsamples = 4096\nvalue = [3620, 191, 1
479, 1240]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'SO_2 <= 20.63\ngini = 0.458\nsamples = 1528\nvalue =
[1709, 15, 493, 218]\nclass = a'),
 Text(3422.4, 543.5999999999999, 'OXY <= 5.835\ngini = 0.584\nsamples = 712\nvalue = [59
0, 15, 416, 112]\nclass = a'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.62\nsamples = 466\nvalue = [283,
13, 339, 99]\nclass = c'),
 Text(3496.8, 181.19999999999982, 'gini = 0.37\nsamples = 246\nvalue = [307, 2, 77, 13]
\nclass = a'),
 Text(3720.0000000000005, 543.5999999999999, 'OXY <= 4.205\ngini = 0.251\nsamples = 816
\nvalue = [1119, 0, 77, 106]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.395\nsamples = 18\nvalue = [5,
0, 3, 25]\nclass = d'),
 Text(3794.4, 181.19999999999982, 'gini = 0.222\nsamples = 798\nvalue = [1114, 0, 74, 8
1]\nclass = a'),
 Text(4166.400000000001, 906.0, 'SO_2 <= 32.57\ngini = 0.66\nsamples = 2568\nvalue = [19
11, 176, 986, 1022]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'O_3 <= 7.555\ngini = 0.706\nsamples = 1359
\nvalue = [700, 174, 770, 542]\nclass = c'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.646\nsamples = 658\nvalue = [16
5, 142, 555, 190]\nclass = c'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.644\nsamples = 701\nvalue = [53
5, 32, 215, 352]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'SO_2 <= 62.19\ngini = 0.522\nsamples = 1209
\nvalue = [1211, 2, 216, 480]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.581\nsamples = 879\nvalue = [773, 2, 195, 42
8]\nclass = a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.253\nsamples = 330\nvalue = [438, 0, 21, 52]
\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.15413624199466525

Ridge Regression:0.15413624199466525

Lasso Regression:0.032119074438977324

ElasticNet Regression:0.09871426228846358

Logistic Regression:0.8087229094340894

Random Forest:0.7331953004622496

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