20104028

Heamnath N

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_2008.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	P
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.889999	
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.040001	
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.270000	
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.850000	
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.160000	
...	...	...	...	...	...	...	...	...	...	...	...	...
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5.450000	
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15.020000	
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17.540001	

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	P
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11.910000	
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12.690000	

226392 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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 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	0.80	28079006

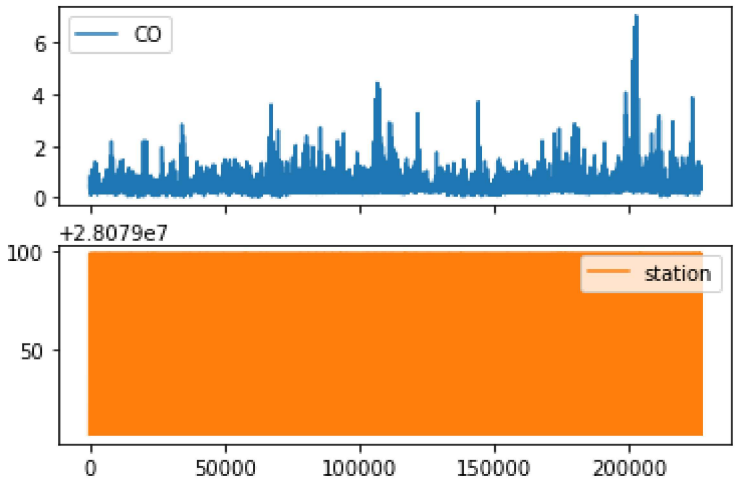
	CO	station
21	0.39	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...	...	...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

Line chart

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

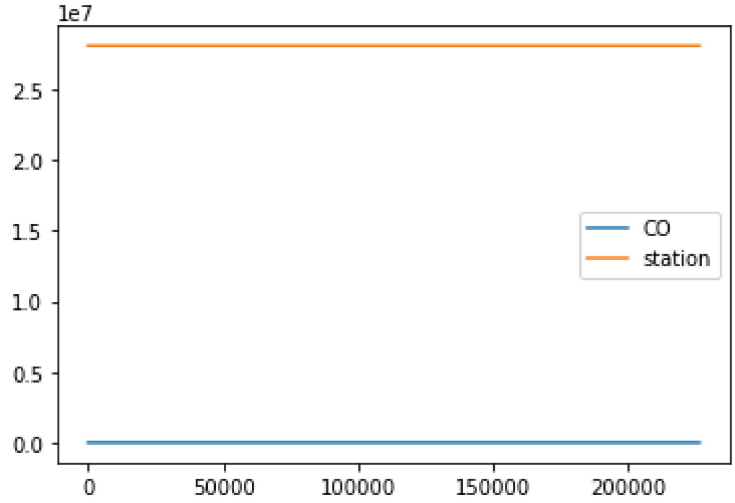
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

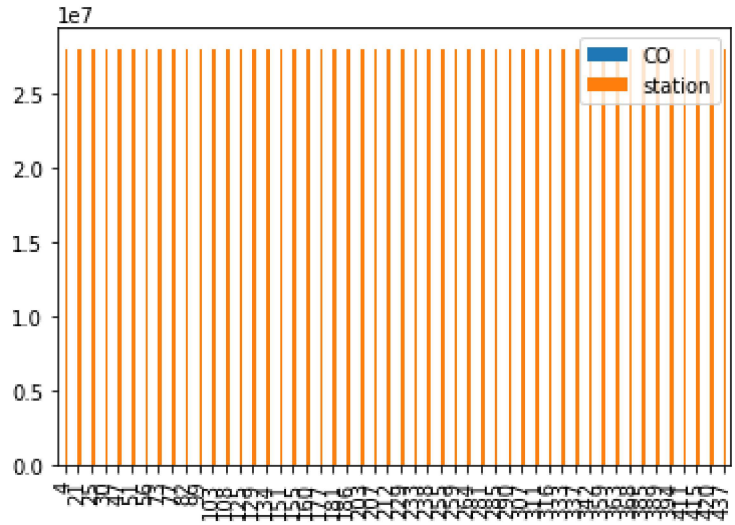


Bar chart

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

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

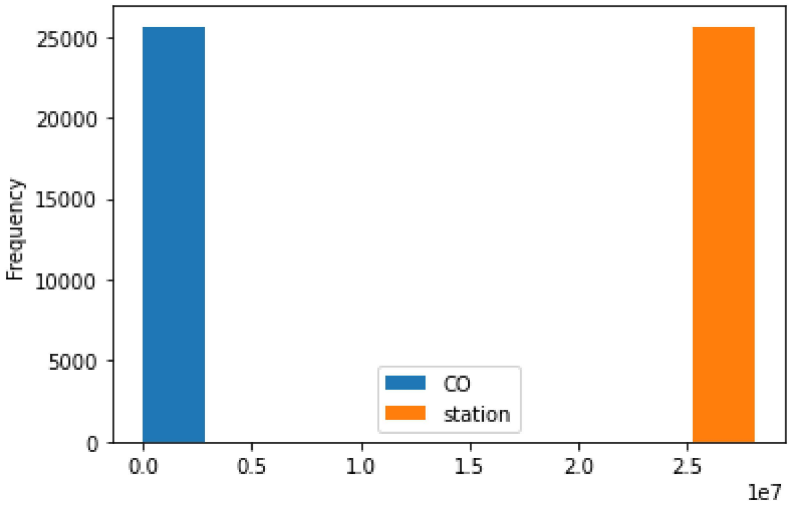
Out[10]: <AxesSubplot:>



Histogram

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

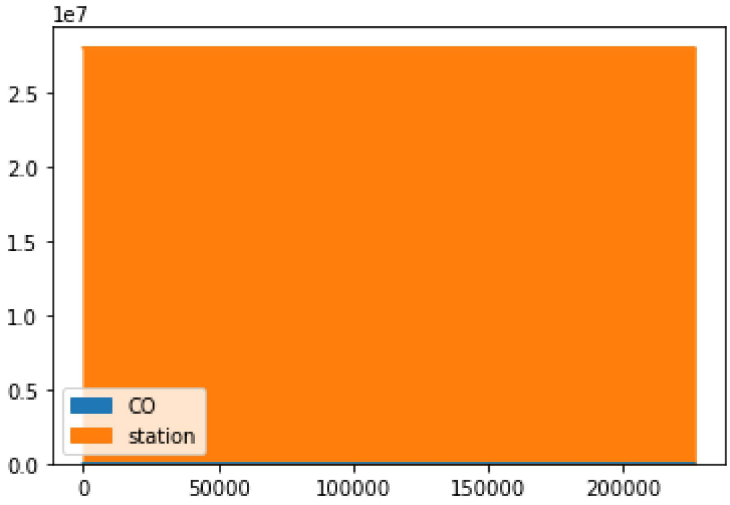
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

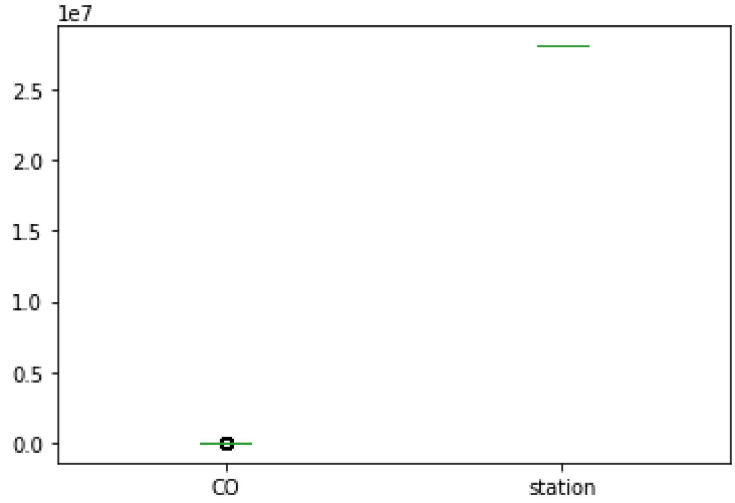
Out[12]: <AxesSubplot:>



Box chart

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

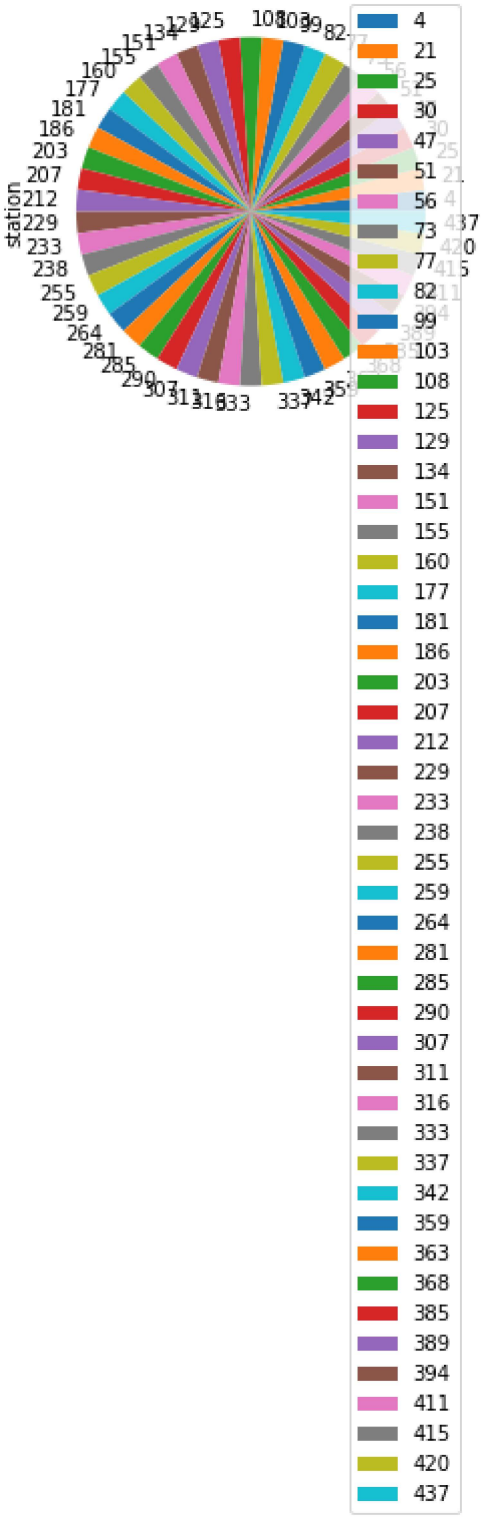
Out[13]: <AxesSubplot:>



Pie chart

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

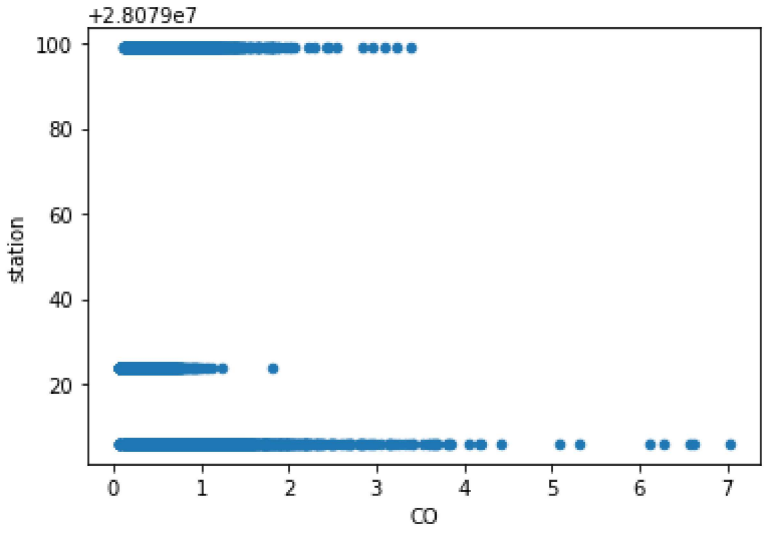
Out[14]: <AxesSubplot:ylabel='station'>



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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 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	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.00000
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261	98.00773
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647	101.44823
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000	2.11000
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999	32.63500
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000	71.11000

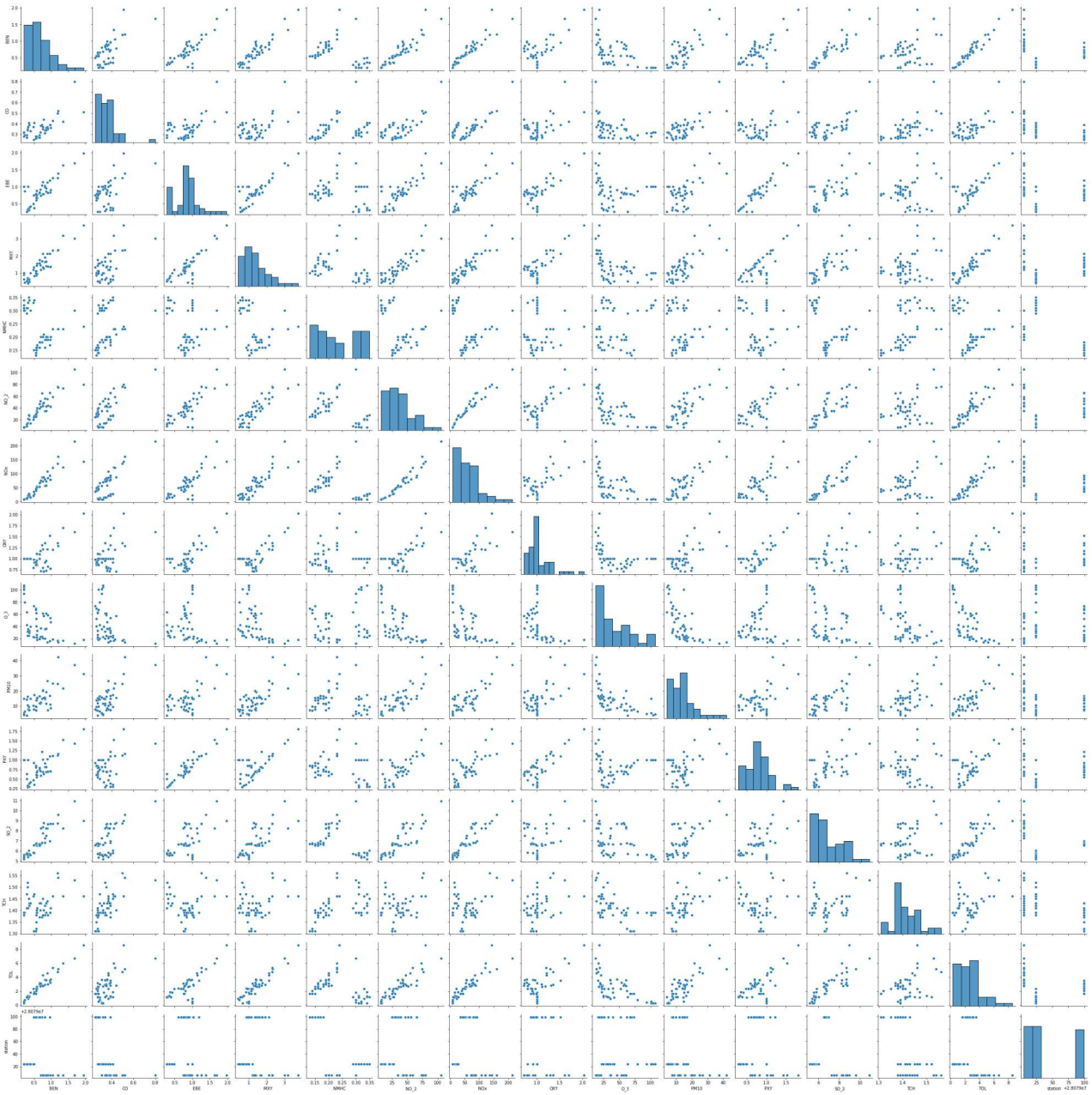
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999	131.55000
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024	2004.00000

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])
```

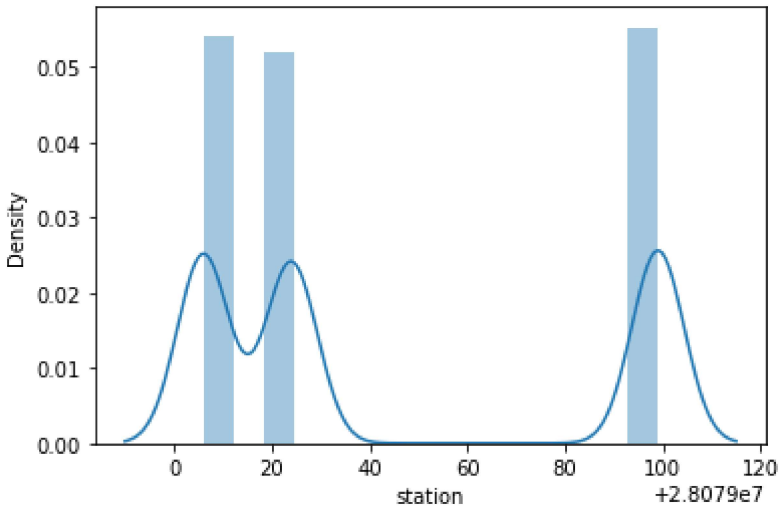
Out[19]: <seaborn.axisgrid.PairGrid at 0x24d8280f6d0>



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

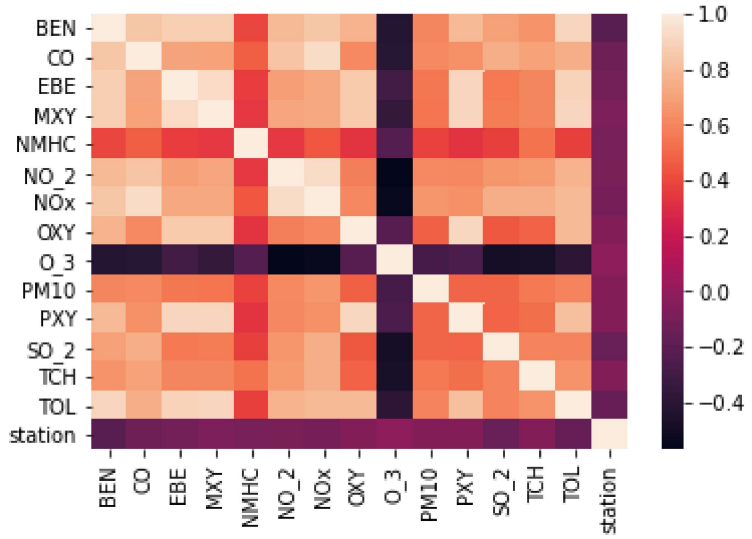
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28079036.42478702
```

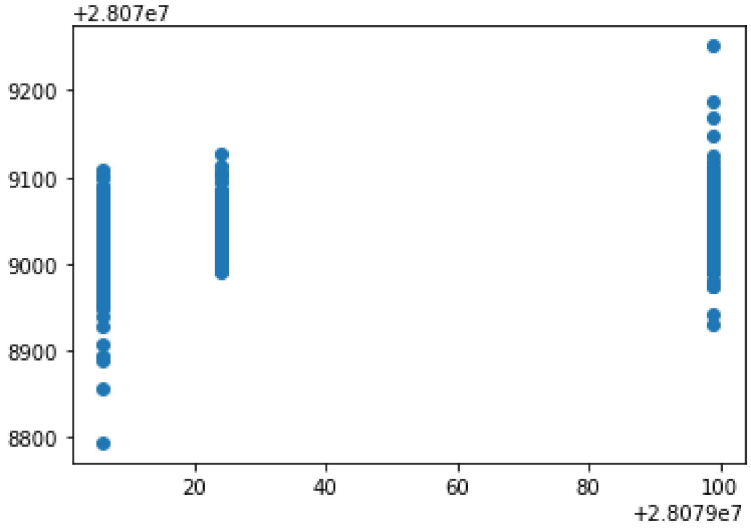
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-26.597342
CO	-0.637339
EBE	-1.518449
MXY	7.900139
NMHC	-27.260871
NO_2	-0.046701
NOx	0.136103
OXY	3.779547
O_3	-0.137764
PM10	0.149288
PXY	2.734614
SO_2	-0.621577
TCH	16.641957
TOL	-1.938340

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x24d911ba820>
```



ACCURACY

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

Out[28]: 0.13557396666490573

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

Out[29]: 0.14679836459932194

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]: 0.13557710254120614

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

Out[33]: 0.14677484243940597

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]:
```
0.04230979290865633
```

Accuracy(Lasso)

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

Out[36]:
```
0.041769918579957044
```

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([-4.71027737, -0.        , -0.        ,  3.3791079 , -0.        ,
        0.05793323,  0.02865665,  1.44651612, -0.15181064,  0.14269973,
        1.5330996 , -0.96148094,  0.        , -2.60212333])
```

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

Out[39]:
```
28079057.271412406
```

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

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

Out[41]:
```
0.09290372083540299
```

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)))
```

```
35.649377816155436
1481.6748546806912
38.49252985555368
```

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]: (25631, 14)

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

Out[46]: (25631,)

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]: 0.794194530061254

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

Out[54]:
```
8.321803242555043e-09
```

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

Out[55]:
```
array([[8.32180324e-09, 1.19114634e-13, 9.99999992e-01]])
```

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]:
```
0.8463849699272535
```

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

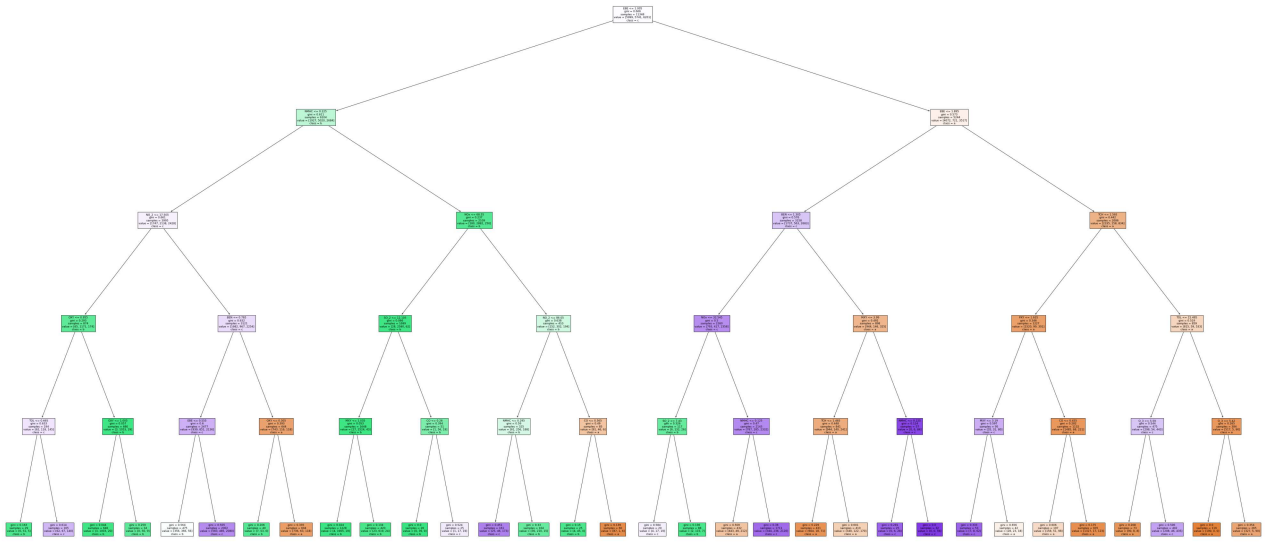
In [62]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','d']
```

Out[62]:
```
[Text(2232.0, 1993.2, 'EBE <= 1.005\ngini = 0.666\nsamples = 11348\nvalue = [5999, 5741,
6201]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'NMHC <= 0.225\ngini = 0.611\nsamples = 6104\nvalue =
[1927, 5020, 2684]\nclass = b'),
```

```
 Text(558.0, 1268.4, 'NO_2 <= 17.505\ngini = 0.661\nsamples = 3995\nvalue = [1747, 2138,
2428]\nclass = c'),
 Text(279.0, 906.0, 'OXY <= 0.955\ngini = 0.293\nsamples = 874\nvalue = [65, 1171, 174]
\nclass = b'),
 Text(139.5, 543.5999999999999, 'TOL <= 0.665\ngini = 0.633\nsamples = 194\nvalue = [62,
118, 145]\nclass = c'),
 Text(69.75, 181.19999999999982, 'gini = 0.163\nsamples = 29\nvalue = [0, 51, 5]\nclass
= b'),
 Text(209.25, 181.19999999999982, 'gini = 0.614\nsamples = 165\nvalue = [62, 67, 140]\nc
lass = c'),
 Text(418.5, 543.5999999999999, 'OXY <= 1.005\ngini = 0.057\nsamples = 680\nvalue = [3,
1053, 29]\nclass = b'),
 Text(348.75, 181.19999999999982, 'gini = 0.044\nsamples = 646\nvalue = [3, 1003, 20]\nc
lass = b'),
 Text(488.25, 181.19999999999982, 'gini = 0.259\nsamples = 34\nvalue = [0, 50, 9]\nclass
= b'),
 Text(837.0, 906.0, 'BEN <= 0.765\ngini = 0.632\nsamples = 3121\nvalue = [1682, 967, 225
4]\nclass = c'),
 Text(697.5, 543.5999999999999, 'EBE <= 0.555\ngini = 0.6\nsamples = 2477\nvalue = [939,
851, 2136]\nclass = c'),
 Text(627.75, 181.19999999999982, 'gini = 0.564\nsamples = 475\nvalue = [356, 366, 56]\n
class = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.505\nsamples = 2002\nvalue = [583, 485, 208
0]\nclass = c'),
 Text(976.5, 543.5999999999999, 'OXY <= 0.305\ngini = 0.393\nsamples = 644\nvalue = [74
3, 116, 118]\nclass = a'),
 Text(906.75, 181.19999999999982, 'gini = 0.206\nsamples = 40\nvalue = [7, 53, 0]\nclass
= b'),
 Text(1046.25, 181.19999999999982, 'gini = 0.335\nsamples = 604\nvalue = [736, 63, 118]
\nclass = a'),
 Text(1674.0, 1268.4, 'NOx <= 68.55\ngini = 0.237\nsamples = 2109\nvalue = [180, 2882, 2
56]\nclass = b'),
 Text(1395.0, 906.0, 'SO_2 <= 12.105\ngini = 0.066\nsamples = 1699\nvalue = [28, 2580, 6
2]\nclass = b'),
 Text(1255.5, 543.5999999999999, 'MXY <= 1.035\ngini = 0.053\nsamples = 1648\nvalue = [2
7, 2524, 43]\nclass = b'),
 Text(1185.75, 181.19999999999982, 'gini = 0.024\nsamples = 1228\nvalue = [4, 1905, 19]
\nclass = b'),
 Text(1325.25, 181.19999999999982, 'gini = 0.134\nsamples = 420\nvalue = [23, 619, 24]\n
class = b'),
 Text(1534.5, 543.5999999999999, 'CO <= 0.26\ngini = 0.394\nsamples = 51\nvalue = [1, 5
6, 19]\nclass = b'),
 Text(1464.75, 181.19999999999982, 'gini = 0.0\nsamples = 26\nvalue = [0, 39, 0]\nclass
= b'),
 Text(1604.25, 181.19999999999982, 'gini = 0.524\nsamples = 25\nvalue = [1, 17, 19]\ncla
ss = c'),
 Text(1953.0, 906.0, 'NO_2 <= 86.05\ngini = 0.638\nsamples = 410\nvalue = [152, 302, 19
4]\nclass = b'),
 Text(1813.5, 543.5999999999999, 'NMHC <= 0.295\ngini = 0.59\nsamples = 325\nvalue = [6
1, 256, 188]\nclass = b'),
 Text(1743.75, 181.19999999999982, 'gini = 0.451\nsamples = 161\nvalue = [25, 46, 173]\n
class = c'),
 Text(1883.25, 181.19999999999982, 'gini = 0.33\nsamples = 164\nvalue = [36, 210, 15]\nc
lass = b'),
 Text(2092.5, 543.5999999999999, 'CO <= 0.565\ngini = 0.49\nsamples = 85\nvalue = [91, 4
6, 6]\nclass = a'),
 Text(2022.75, 181.19999999999982, 'gini = 0.15\nsamples = 25\nvalue = [4, 45, 0]\nclass
= b'),
 Text(2162.25, 181.19999999999982, 'gini = 0.139\nsamples = 60\nvalue = [87, 1, 6]\nclas
s = a'),
 Text(3348.0, 1630.8000000000002, 'EBE <= 1.895\ngini = 0.573\nsamples = 5244\nvalue =
[4072, 721, 3517]\nclass = a'),
 Text(2790.0, 1268.4, 'BEN <= 1.305\ngini = 0.576\nsamples = 3158\nvalue = [1737, 563, 2
683]\nclass = c'),
 Text(2511.0, 906.0, 'NOx <= 32.545\ngini = 0.5\nsamples = 2260\nvalue = [793, 417, 235
```

```
8]\nclass = c'),
 Text(2371.5, 543.5999999999999, 'SO_2 <= 7.43\ngini = 0.326\nsamples = 117\nvalue = [6,
132, 26]\nclass = b'),
 Text(2301.75, 181.19999999999982, 'gini = 0.584\nsamples = 29\nvalue = [4, 17, 19]\ncla
ss = c'),
 Text(2441.25, 181.19999999999982, 'gini = 0.136\nsamples = 88\nvalue = [2, 115, 7]\ncla
ss = b'),
 Text(2650.5, 543.5999999999999, 'NMHC <= 0.125\ngini = 0.47\nsamples = 2143\nvalue = [7
87, 285, 2332]\nclass = c'),
 Text(2580.75, 181.19999999999982, 'gini = 0.509\nsamples = 432\nvalue = [443, 49, 212]
\nclass = a'),
 Text(2720.25, 181.19999999999982, 'gini = 0.36\nsamples = 1711\nvalue = [344, 236, 212
0]\nclass = c'),
 Text(3069.0, 906.0, 'MXY <= 3.99\ngini = 0.492\nsamples = 898\nvalue = [944, 146, 325]
\nclass = a'),
 Text(2929.5, 543.5999999999999, 'TCH <= 1.465\ngini = 0.448\nsamples = 841\nvalue = [94
4, 140, 241]\nclass = a'),
 Text(2859.75, 181.19999999999982, 'gini = 0.229\nsamples = 431\nvalue = [604, 18, 71]\n
class = a'),
 Text(2999.25, 181.19999999999982, 'gini = 0.601\nsamples = 410\nvalue = [340, 122, 170]
\nclass = a'),
 Text(3208.5, 543.5999999999999, 'NMHC <= 0.225\ngini = 0.124\nsamples = 57\nvalue = [0,
6, 84]\nclass = c'),
 Text(3138.75, 181.19999999999982, 'gini = 0.291\nsamples = 25\nvalue = [0, 6, 28]\nclas
s = c'),
 Text(3278.25, 181.19999999999982, 'gini = 0.0\nsamples = 32\nvalue = [0, 0, 56]\nclass
= c'),
 Text(3906.0, 1268.4, 'TCH <= 1.565\ngini = 0.442\nsamples = 2086\nvalue = [2335, 158, 8
34]\nclass = a'),
 Text(3627.0, 906.0, 'PXY <= 1.635\ngini = 0.346\nsamples = 1227\nvalue = [1520, 99, 30
1]\nclass = a'),
 Text(3487.5, 543.5999999999999, 'MXY <= 3.19\ngini = 0.597\nsamples = 95\nvalue = [35,
31, 80]\nclass = c'),
 Text(3417.75, 181.19999999999982, 'gini = 0.333\nsamples = 53\nvalue = [7, 8, 62]\nclas
s = c'),
 Text(3557.25, 181.19999999999982, 'gini = 0.656\nsamples = 42\nvalue = [28, 23, 18]\ncl
ass = a'),
 Text(3766.5, 543.5999999999999, 'CO <= 0.455\ngini = 0.282\nsamples = 1132\nvalue = [14
85, 68, 221]\nclass = a'),
 Text(3696.75, 181.19999999999982, 'gini = 0.606\nsamples = 197\nvalue = [158, 51, 98]\n
class = a'),
 Text(3836.25, 181.19999999999982, 'gini = 0.175\nsamples = 935\nvalue = [1327, 17, 123]
\nclass = a'),
 Text(4185.0, 906.0, 'TOL <= 15.495\ngini = 0.519\nsamples = 859\nvalue = [815, 59, 533]
\nclass = a'),
 Text(4045.5, 543.5999999999999, 'O_3 <= 5.59\ngini = 0.544\nsamples = 475\nvalue = [29
8, 54, 443]\nclass = c'),
 Text(3975.75, 181.19999999999982, 'gini = 0.268\nsamples = 73\nvalue = [90, 8, 8]\nclas
s = a'),
 Text(4115.25, 181.19999999999982, 'gini = 0.506\nsamples = 402\nvalue = [208, 46, 435]
\nclass = c'),
 Text(4324.5, 543.5999999999999, 'O_3 <= 5.42\ngini = 0.265\nsamples = 384\nvalue = [51
7, 5, 90]\nclass = a'),
 Text(4254.75, 181.19999999999982, 'gini = 0.0\nsamples = 119\nvalue = [190, 0, 0]\nclas
s = a'),
 Text(4394.25, 181.19999999999982, 'gini = 0.354\nsamples = 265\nvalue = [327, 5, 90]\nc
lass = a')]
```



Conclusion

Accuracy

In [63]:
```
print("Linear Regression:",lr.score(x_test,y_test))
print("Ridge Regression:",rr.score(x_test,y_test))
print("Lasso Regression",la.score(x_test,y_test))
print("ElasticNet Regression:",en.score(x_test,y_test))
print("Logistic Regression:",logr.score(fs,target_vector))
print("Random Forest:",grid_search.best_score_)
```

```
Linear Regression: 0.13557396666490573
Ridge Regression: 0.13557710254120614
Lasso Regression 0.041769918579957044
ElasticNet Regression: 0.09290372083540299
Logistic Regression: 0.794194530061254
Random Forest: 0.8463849699272535
```

Random Forest is suitable for this dataset