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_2009.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.260000	N
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.580000	N
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.190001	N
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.530001	6
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.760000	N
...	...	...	...	...	...	...	...	...	...	...	...	
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.830000	7
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.920000	N
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.460000	6

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.030000	N
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.360000	1

215688 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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

Out[6]:

	CO	station
3	0.33	28079006

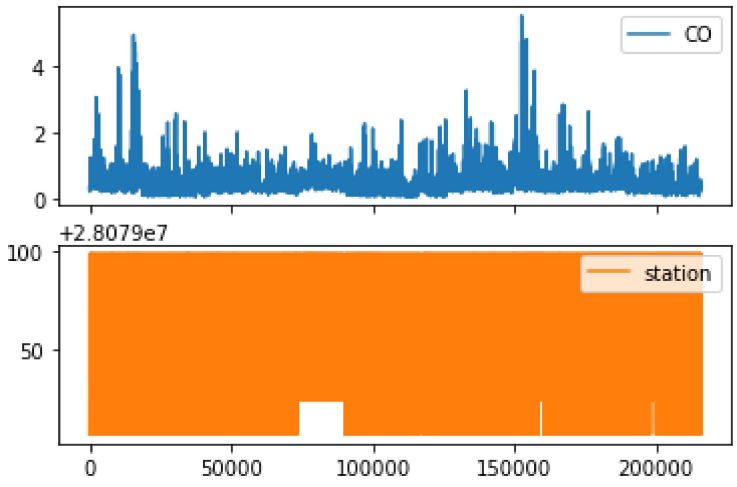
	CO	station
20	0.32	28079099
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...	...	...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

Line chart

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

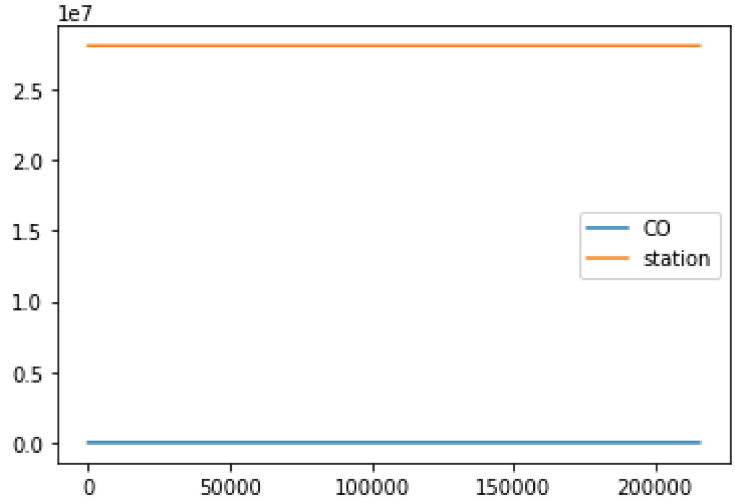
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

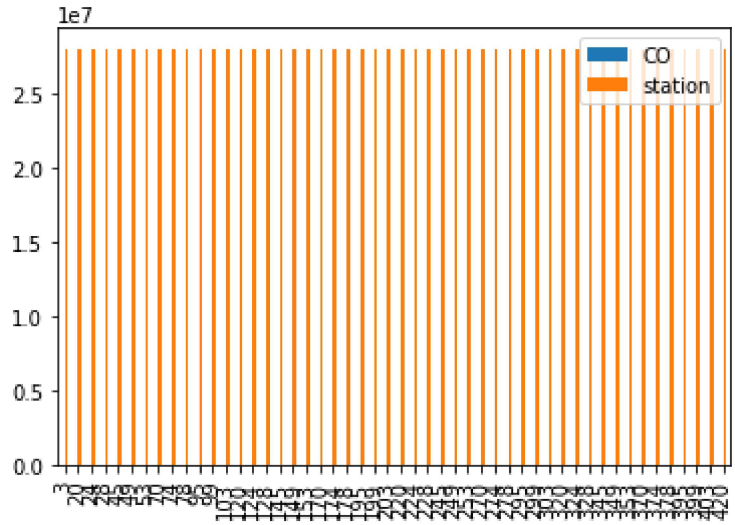


Bar chart

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

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

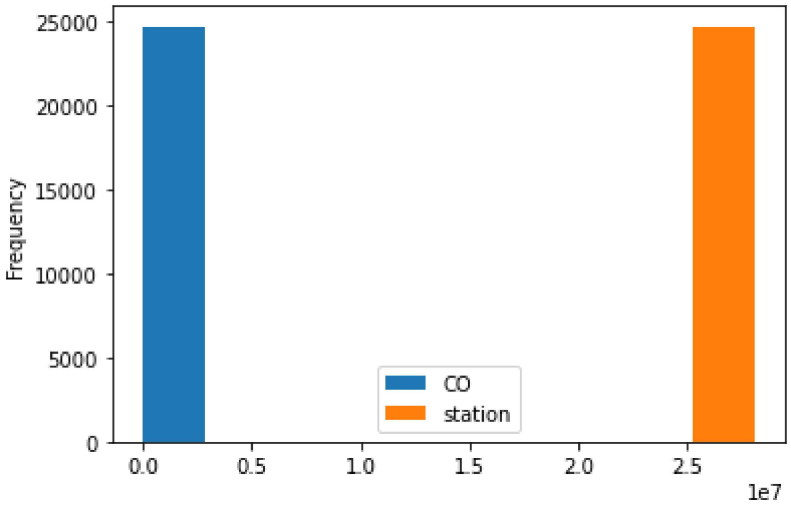
Out[10]: <AxesSubplot:>



Histogram

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

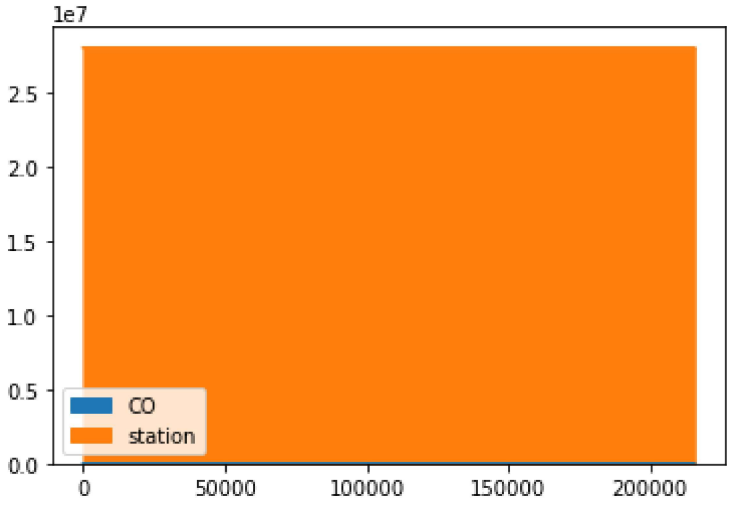
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

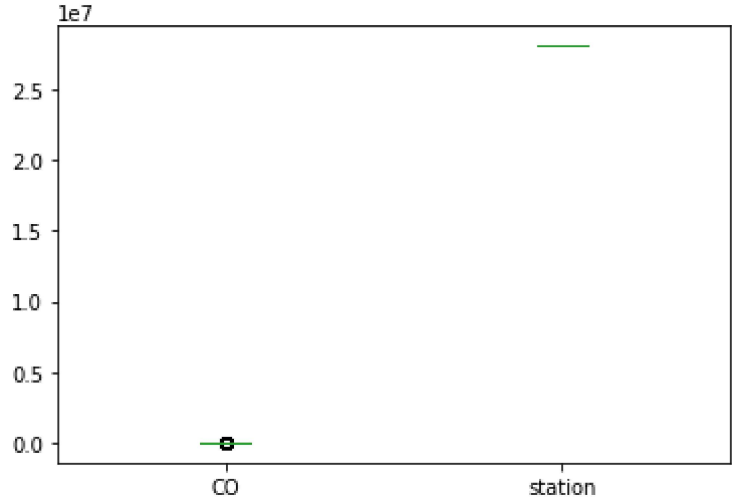
Out[12]: <AxesSubplot:>



Box chart

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

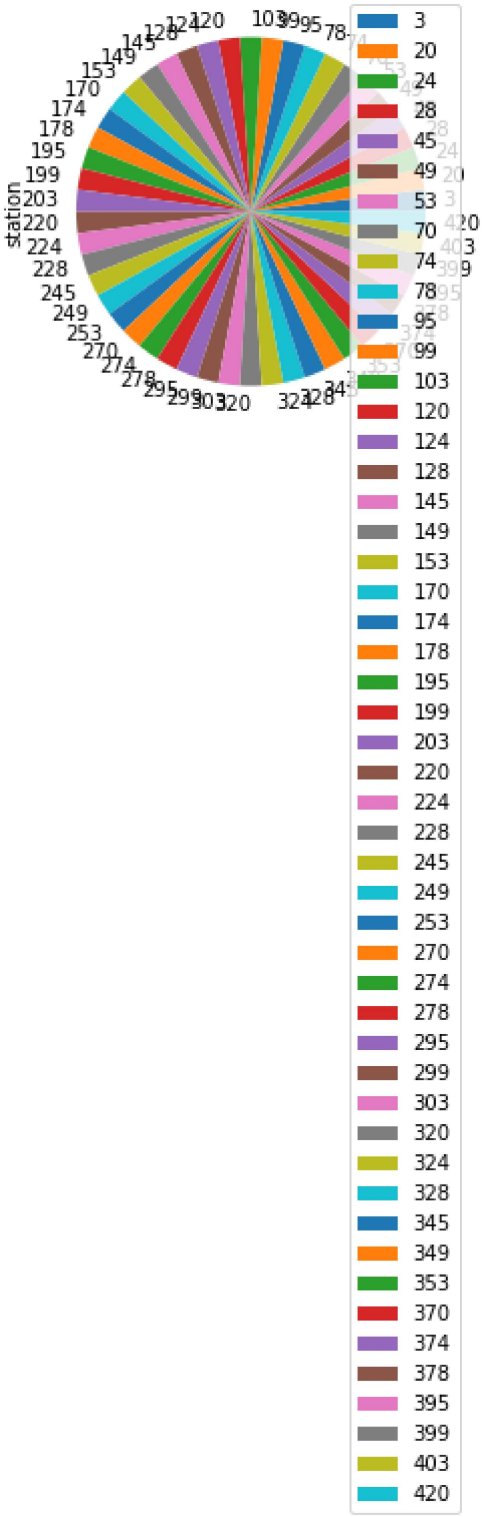
Out[13]: <AxesSubplot:>



Pie chart

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

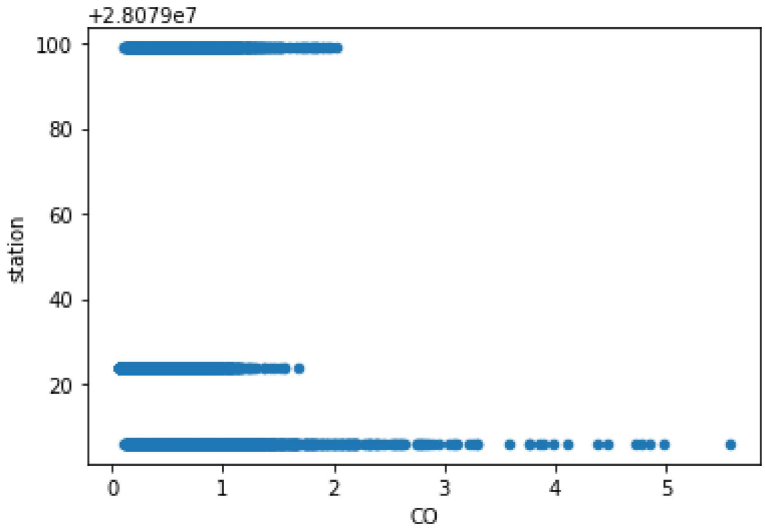
Out[14]: <AxesSubplot:ylabel='station'>



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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.00000
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929	92.90718
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677	91.98535
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000	2.25000
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000	33.00999
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000	67.01000

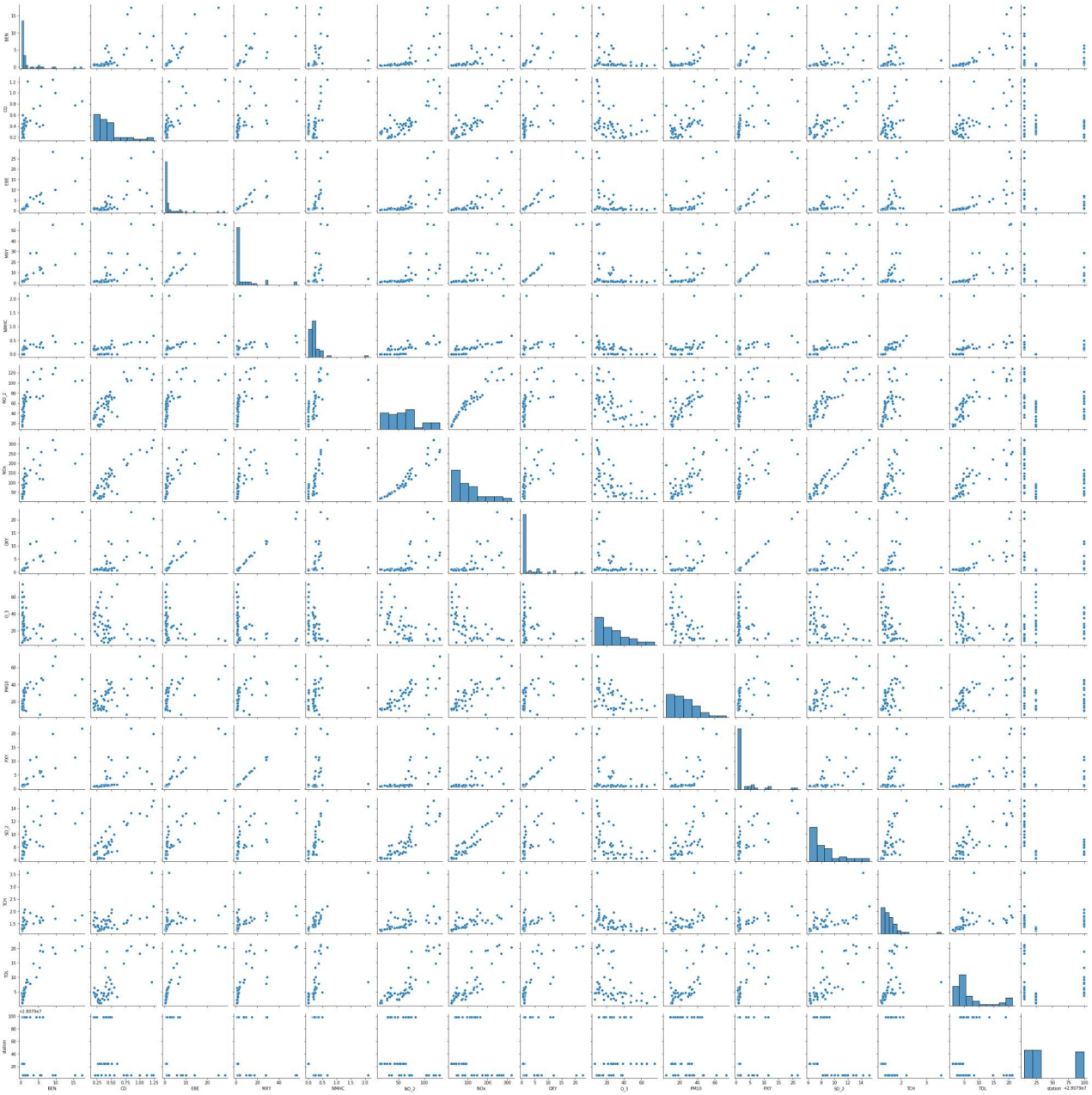
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997	124.69999
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994	1438.00000

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])
```

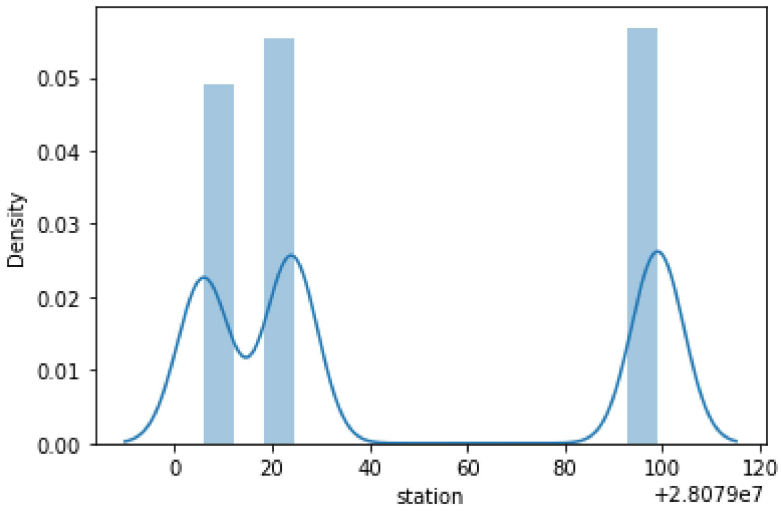
Out[19]: <seaborn.axisgrid.PairGrid at 0x1164aaf27c0>



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

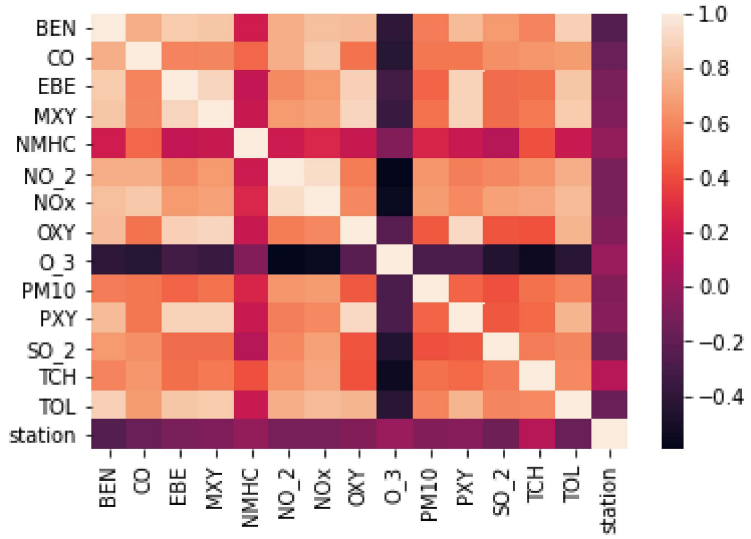
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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]:
```
28078892.665296998
```

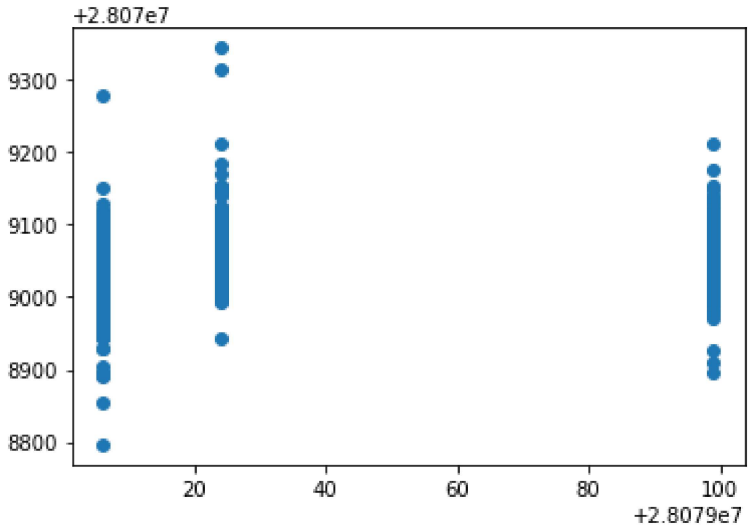
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-34.899220
CO	-28.765877
EBE	5.803456
MXY	-0.190398
NMHC	-19.084382
NO_2	-0.152268
NOx	0.177402
OXY	11.781123
O_3	0.025219
PM10	-0.050724
PXY	2.821465
SO_2	-0.387518
TCH	125.534611
TOL	-1.071650

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x11654cc98e0>
```



ACCURACY

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

Out[28]: 0.27787771499038305

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

Out[29]: 0.29072992168581624

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]: 0.2794352407863486

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

Out[33]: 0.29038083663310943

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]:
```
0.03875363015642119
```

Accuracy(Lasso)

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

Out[36]:
```
0.03495869696400822
```

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([-7.00192409, -0.60387107,  0.1409905 ,  2.08113437, -0.        ,
       -0.23004935,  0.13044212,  1.15080126, -0.14535201,  0.08614803,
        2.07352642, -0.84794817,  1.54538454, -1.9358594 ])
```

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

Out[39]:
```
28079064.34733883
```

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

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

Out[41]:
```
0.10348190610556163
```

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)))
```

```
35.86803561911397
1462.6597580464495
38.244735036949194
```

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]:
```
(24717, 14)
```

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

Out[46]:
```
(24717,)
```

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]:
```
0.8951733624630821
```

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

Out[54]:
```
5.447205522232353e-13
```

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

Out[55]:
```
array([[5.44720552e-13, 8.28692830e-44, 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]:
```
0.9047453433496612
```

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

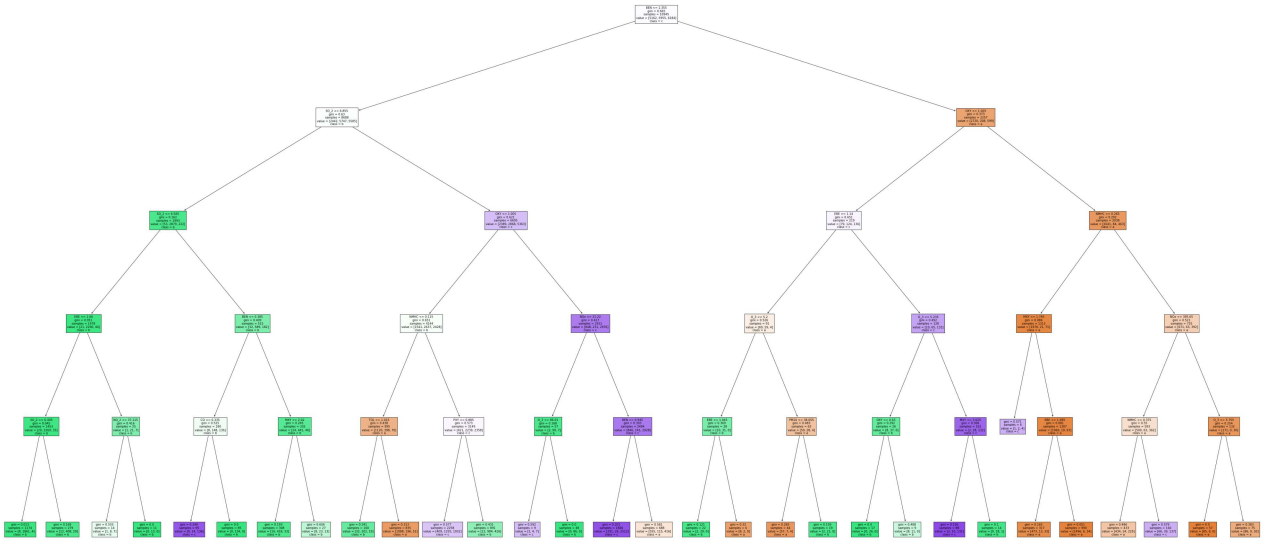
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2315.7000000000003, 1993.2, 'BEN <= 1.355\ngini = 0.665\nsamples = 10945\nvalue =
[5162, 5955, 6184]\nclass = c'),
 Text(1190.4, 1630.8000000000002, 'SO_2 <= 6.855\ngini = 0.63\nsamples = 8688\nvalue =
[2442, 5747, 5585]\nclass = b'),
```

```
 Text(595.2, 1268.4, 'SO_2 <= 6.565\ngini = 0.162\nsamples = 1993\nvalue = [53, 2879, 22
2]\nclass = b'),
 Text(297.6, 906.0, 'EBE <= 1.06\ngini = 0.051\nsamples = 1478\nvalue = [21, 2290, 40]\n
class = b'),
 Text(148.8, 543.5999999999999, 'SO_2 <= 6.405\ngini = 0.045\nsamples = 1453\nvalue = [2
0, 2269, 33]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.013\nsamples = 1174\nvalue = [8, 1861, 4]\ncla
ss = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.169\nsamples = 279\nvalue = [12,
408, 29]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'NO_2 <= 37.115\ngini = 0.416\nsamples = 25
\nvalue = [1, 21, 7]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.555\nsamples = 14\nvalue = [1, 8, 7]\nclass =
b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [0, 13,
0]\nclass = b'),
 Text(892.8000000000001, 906.0, 'BEN <= 0.385\ngini = 0.409\nsamples = 515\nvalue = [32,
589, 182]\nclass = b'),
 Text(744.0, 543.5999999999999, 'CO <= 0.335\ngini = 0.525\nsamples = 180\nvalue = [8, 1
48, 136]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.249\nsamples = 95\nvalue = [8, 14, 136]\nclas
s = c'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 85\nvalue = [0, 134,
0]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'MXY <= 2.02\ngini = 0.245\nsamples = 335\n
value = [24, 441, 46]\nclass = b'),
 Text(967.2, 181.19999999999982, 'gini = 0.193\nsamples = 308\nvalue = [16, 418, 33]\ncl
ass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.606\nsamples = 27\nvalue = [8, 23, 13]\nclas
s = b'),
 Text(1785.6000000000001, 1268.4, 'OXY <= 1.005\ngini = 0.621\nsamples = 6695\nvalue =
[2389, 2868, 5363]\nclass = c'),
 Text(1488.0, 906.0, 'NMHC <= 0.115\ngini = 0.651\nsamples = 4144\nvalue = [1541, 2637,
2428]\nclass = b'),
 Text(1339.2, 543.5999999999999, 'TOL <= 1.015\ngini = 0.438\nsamples = 995\nvalue = [11
20, 398, 70]\nclass = a'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.341\nsamples = 160\nvalue = [32,
202, 19]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.313\nsamples = 835\nvalue = [108
8, 196, 51]\nclass = a'),
 Text(1636.8000000000002, 543.5999999999999, 'PXY <= 0.885\ngini = 0.573\nsamples = 3149
\nvalue = [421, 2239, 2358]\nclass = c'),
 Text(1562.4, 181.19999999999982, 'gini = 0.577\nsamples = 2258\nvalue = [409, 1255, 193
2]\nclass = c'),
 Text(1711.2, 181.19999999999982, 'gini = 0.431\nsamples = 891\nvalue = [12, 984, 426]\n
class = b'),
 Text(2083.2000000000003, 906.0, 'NOx <= 21.22\ngini = 0.417\nsamples = 2551\nvalue = [8
48, 231, 2935]\nclass = c'),
 Text(1934.4, 543.5999999999999, 'O_3 <= 86.01\ngini = 0.168\nsamples = 57\nvalue = [2,
90, 7]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.592\nsamples = 9\nvalue = [2, 4,
7]\nclass = c'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.0\nsamples = 48\nvalue = [0, 86,
0]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'BEN <= 0.945\ngini = 0.393\nsamples = 2494\nvalue = [8
46, 141, 2928]\nclass = c'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.201\nsamples = 1806\nvalue = [29
1, 26, 2512]\nclass = c'),
 Text(2306.4, 181.19999999999982, 'gini = 0.581\nsamples = 688\nvalue = [555, 115, 416]
\nclass = a'),
 Text(3441.0000000000005, 1630.8000000000002, 'OXY <= 1.005\ngini = 0.373\nsamples = 225
7\nvalue = [2720, 208, 599]\nclass = a'),
 Text(2976.0, 1268.4, 'EBE <= 1.14\ngini = 0.651\nsamples = 219\nvalue = [79, 124, 136]
\nclass = c'),
 Text(2678.4, 906.0, 'O_3 <= 5.2\ngini = 0.526\nsamples = 91\nvalue = [69, 59, 4]\nclass
```

```
= a'),
 Text(2529.6000000000004, 543.5999999999999, 'EBE <= 1.045\ngini = 0.369\nsamples = 28\n
value = [10, 31, 0]\nclass = b'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.121\nsamples = 22\nvalue = [2, 2
9, 0]\nclass = b'),
 Text(2604.0, 181.19999999999982, 'gini = 0.32\nsamples = 6\nvalue = [8, 2, 0]\nclass =
a'),
 Text(2827.2000000000003, 543.5999999999999, 'PM10 <= 38.055\ngini = 0.483\nsamples = 63
\nvalue = [59, 28, 4]\nclass = a'),
 Text(2752.8, 181.19999999999982, 'gini = 0.283\nsamples = 44\nvalue = [57, 7, 4]\nclass
= a'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.159\nsamples = 19\nvalue = [2, 2
1, 0]\nclass = b'),
 Text(3273.6000000000004, 906.0, 'O_3 <= 5.235\ngini = 0.492\nsamples = 128\nvalue = [1
0, 65, 132]\nclass = c'),
 Text(3124.8, 543.5999999999999, 'OXY <= 0.63\ngini = 0.292\nsamples = 26\nvalue = [8, 3
7, 0]\nclass = b'),
 Text(3050.4, 181.19999999999982, 'gini = 0.0\nsamples = 17\nvalue = [0, 26, 0]\nclass =
b'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.488\nsamples = 9\nvalue = [8, 1
1, 0]\nclass = b'),
 Text(3422.4, 543.5999999999999, 'MXY <= 3.025\ngini = 0.306\nsamples = 102\nvalue = [2,
28, 132]\nclass = c'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.156\nsamples = 88\nvalue = [2, 1
0, 131]\nclass = c'),
 Text(3496.8, 181.19999999999982, 'gini = 0.1\nsamples = 14\nvalue = [0, 18, 1]\nclass =
b'),
 Text(3906.0000000000005, 1268.4, 'NMHC <= 0.265\ngini = 0.292\nsamples = 2038\nvalue =
[2641, 84, 463]\nclass = a'),
 Text(3645.6000000000004, 906.0, 'MXY <= 1.745\ngini = 0.086\nsamples = 1313\nvalue = [1
970, 21, 71]\nclass = a'),
 Text(3571.2000000000003, 543.5999999999999, 'gini = 0.571\nsamples = 6\nvalue = [1, 2,
4]\nclass = c'),
 Text(3720.0000000000005, 543.5999999999999, 'EBE <= 1.695\ngini = 0.081\nsamples = 1307
\nvalue = [1969, 19, 67]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.165\nsamples = 317\nvalue = [47
3, 13, 33]\nclass = a'),
 Text(3794.4, 181.19999999999982, 'gini = 0.051\nsamples = 990\nvalue = [1496, 6, 34]\nc
lass = a'),
 Text(4166.400000000001, 906.0, 'NOx <= 395.65\ngini = 0.521\nsamples = 725\nvalue = [67
1, 63, 392]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'NMHC <= 0.375\ngini = 0.55\nsamples = 593
\nvalue = [500, 63, 362]\nclass = a'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.486\nsamples = 433\nvalue = [43
4, 24, 225]\nclass = a'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.579\nsamples = 160\nvalue = [66,
39, 137]\nclass = c'),
 Text(4315.200000000001, 543.5999999999999, 'O_3 <= 5.755\ngini = 0.254\nsamples = 132\n
value = [171, 0, 30]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.0\nsamples = 57\nvalue = [85, 0, 0]\nclass =
a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.383\nsamples = 75\nvalue = [86, 0, 30]\nclas
s = 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: 0.27787771499038305
Ridge Regression: 0.2794352407863486
Lasso Regression 0.03495869696400822
ElasticNet Regression: 0.10348190610556163
Logistic Regression: 0.8951733624630821
Random Forest: 0.9047453433496612
```

Random Forest is suitable for this dataset