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_2006.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.880000	97.570000	4
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.820000	
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.430000	34.419998	
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.830000	28.260000	
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.990000	54.180000	
...	...	...	...	...	...	...	...	...	...	...	...	...
230563	2006-05-01 00:00:00	5.88	0.83	6.23	NaN	0.20	112.500000	218.000000	NaN	24.389999	93.120003	
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29.469999	1
230565	2006-05-01 00:00:00	0.96	NaN	0.69	NaN	0.19	135.100006	179.199997	NaN	11.460000	64.680000	3

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
230566	2006-05-01 00:00:00	0.50	NaN	0.67	NaN	0.10	82.599998	105.599998	NaN	NaN	94.360001	
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52.490002	2

230568 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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24758 non-null  object 
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 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
5	1.69	28079006

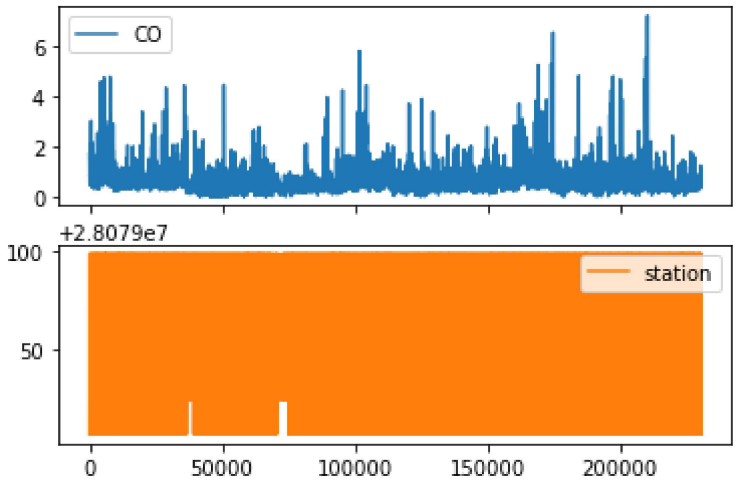
	CO	station
22	0.79	28079024
25	1.47	28079099
31	0.85	28079006
48	0.79	28079024
...	...	...
230538	0.40	28079024
230541	0.94	28079099
230547	1.06	28079006
230564	0.32	28079024
230567	0.74	28079099

24758 rows × 2 columns

Line chart

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

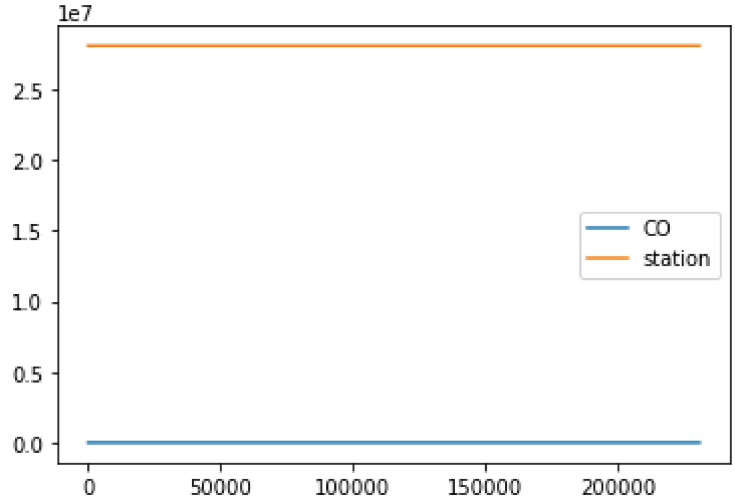
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

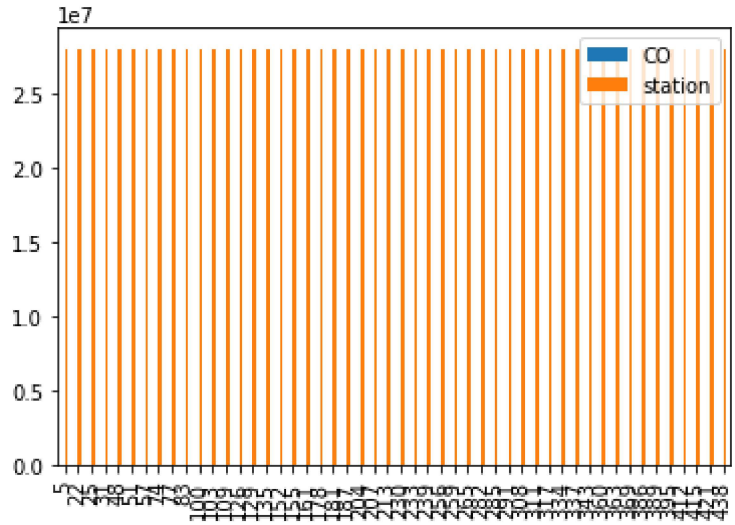


Bar chart

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

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

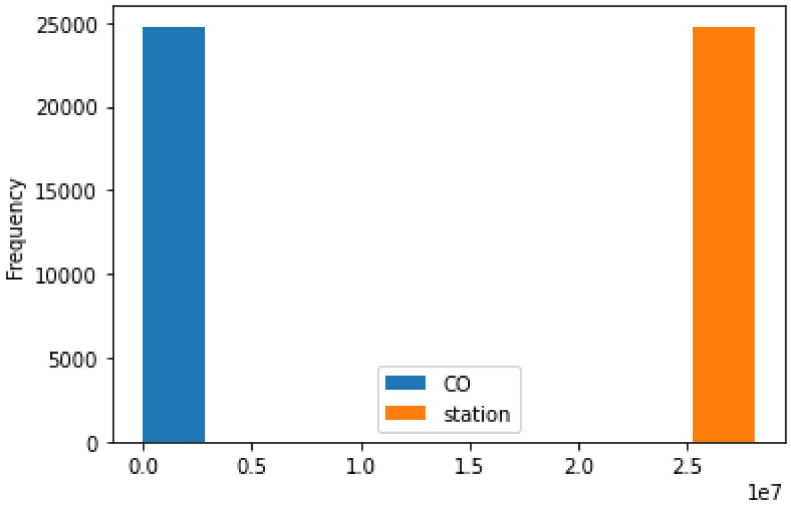
Out[10]: <AxesSubplot:>



Histogram

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

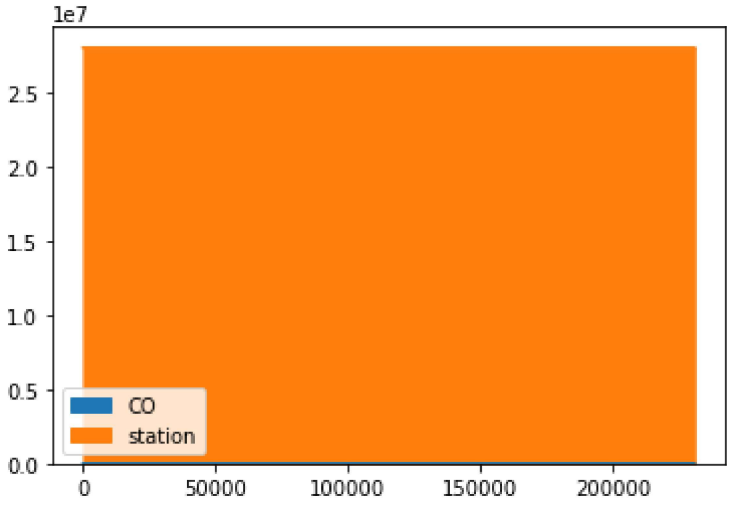
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

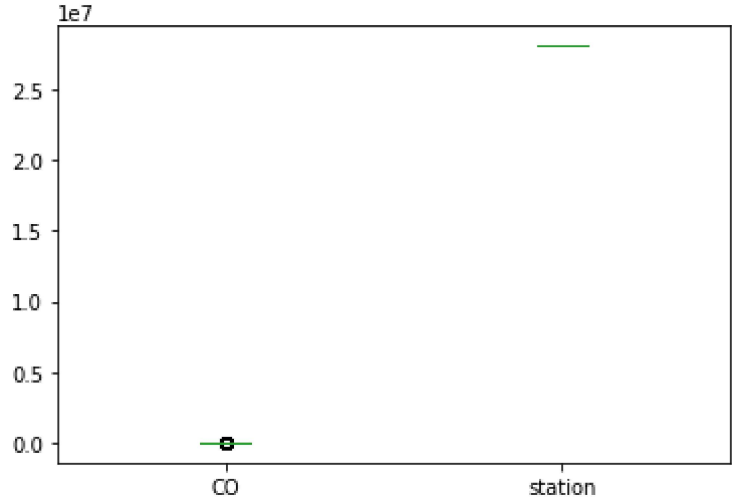
Out[12]: <AxesSubplot:>



Box chart

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

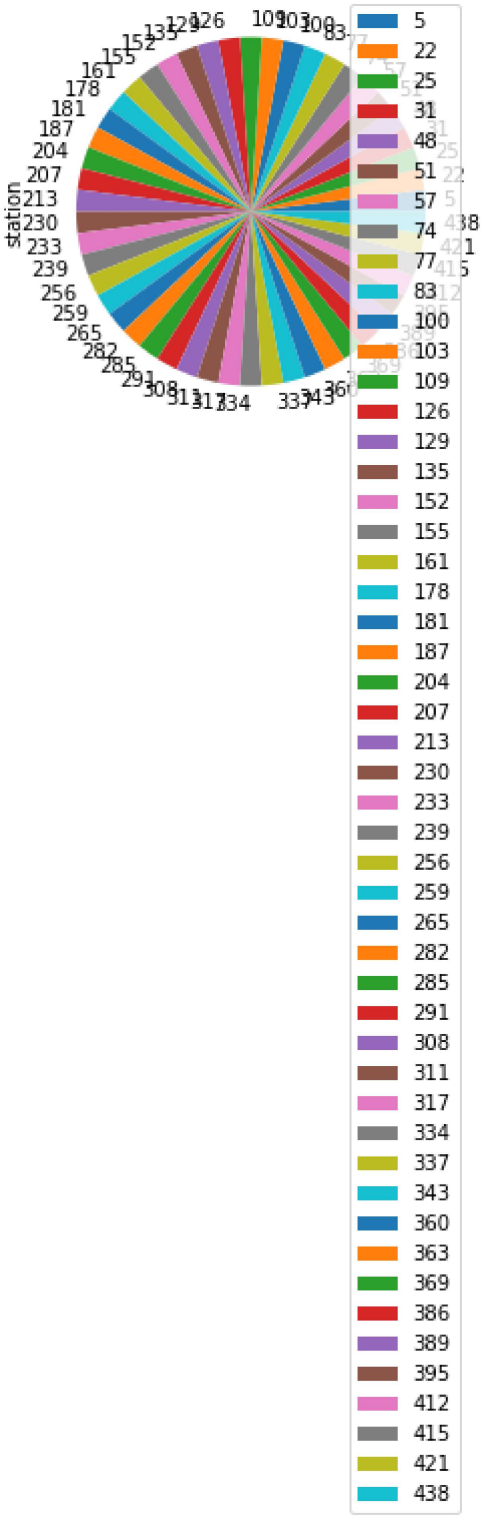
Out[13]: <AxesSubplot:>



Pie chart

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

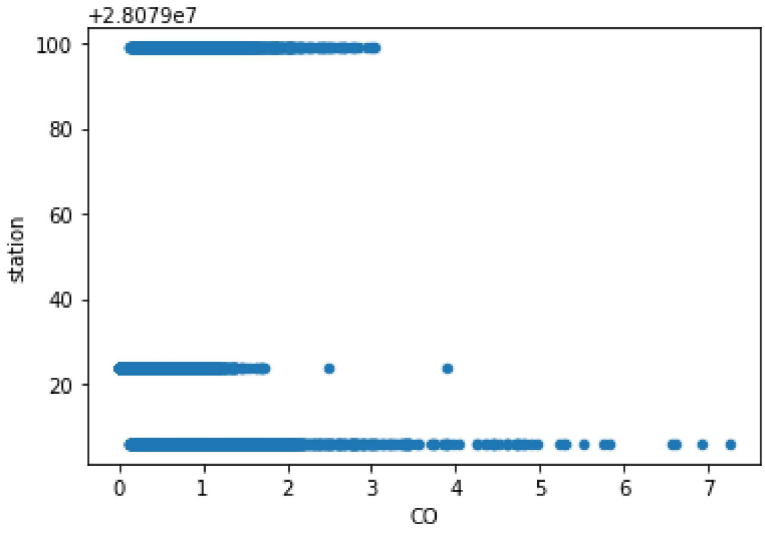
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24758 non-null  object 
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 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	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.00000
mean	1.350624	0.600713	1.824534	3.835034	0.176546	58.333481	116.41909
std	1.541636	0.419048	1.868939	4.069036	0.126683	40.529382	117.55706
min	0.110000	0.000000	0.170000	0.150000	0.000000	1.680000	2.02000
25%	0.450000	0.360000	0.810000	1.060000	0.100000	28.450001	36.88250
50%	0.850000	0.500000	1.130000	2.500000	0.150000	52.959999	85.18000

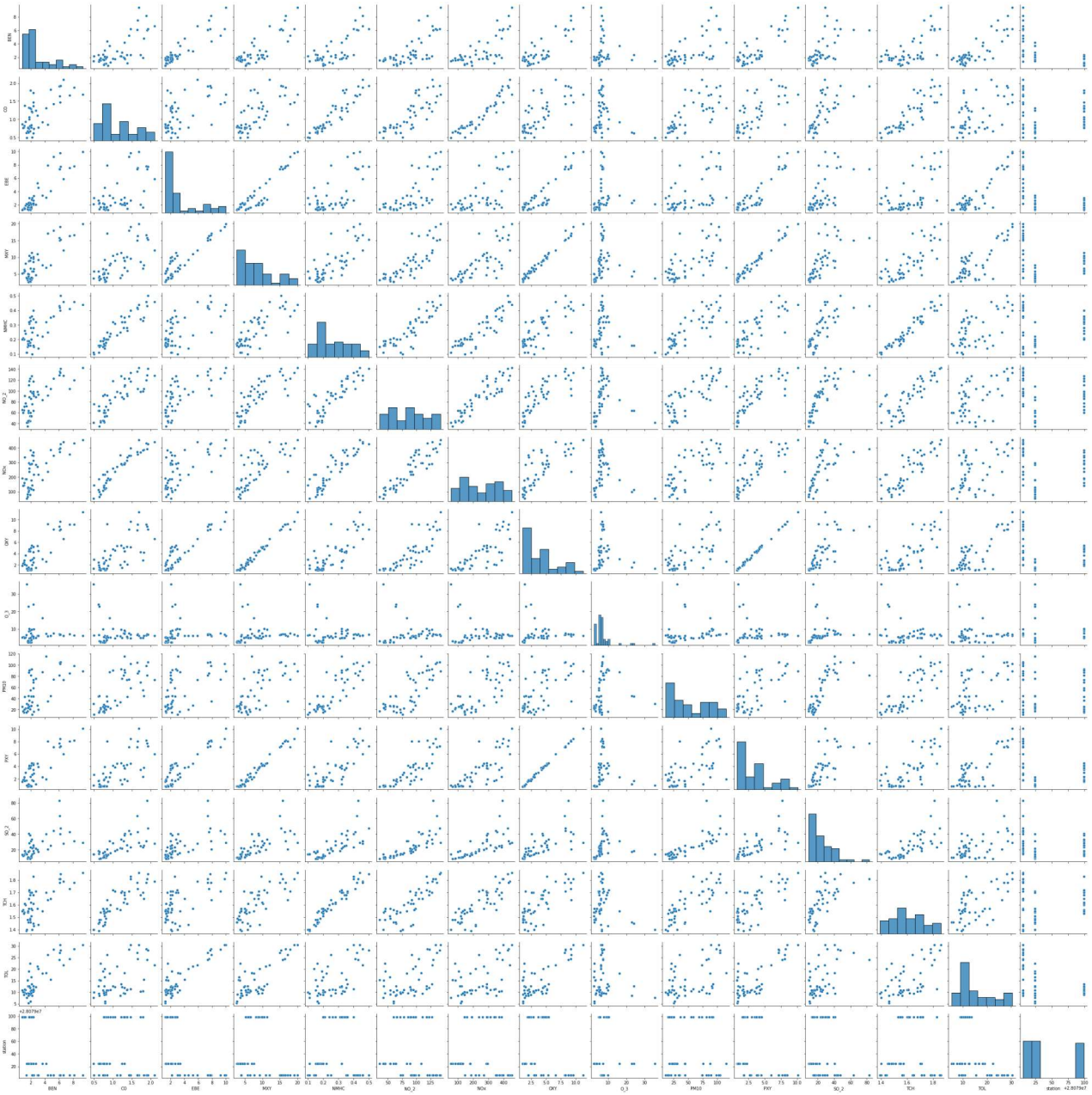
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	1.680000	0.720000	2.160000	5.090000	0.220000	79.347498	158.30000
max	45.430000	7.250000	57.799999	66.900002	2.020000	461.299988	1680.00000

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])
```

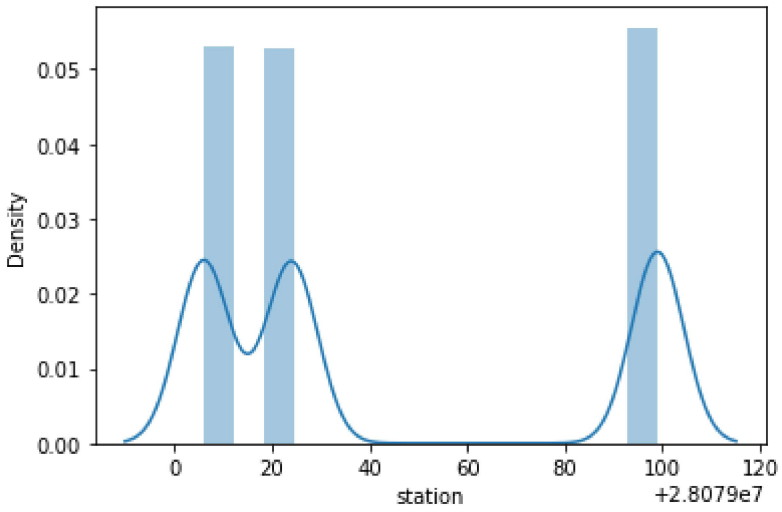
Out[19]: <seaborn.axisgrid.PairGrid at 0x1aa936594f0>



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

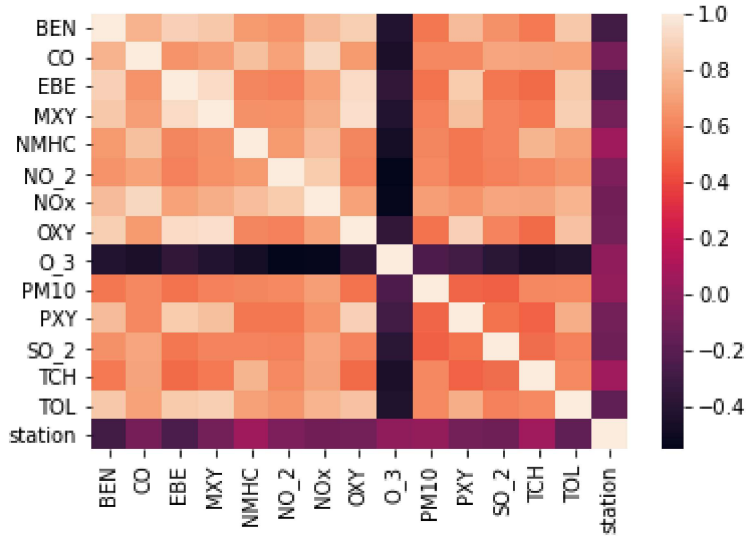
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28079019.30668338
```

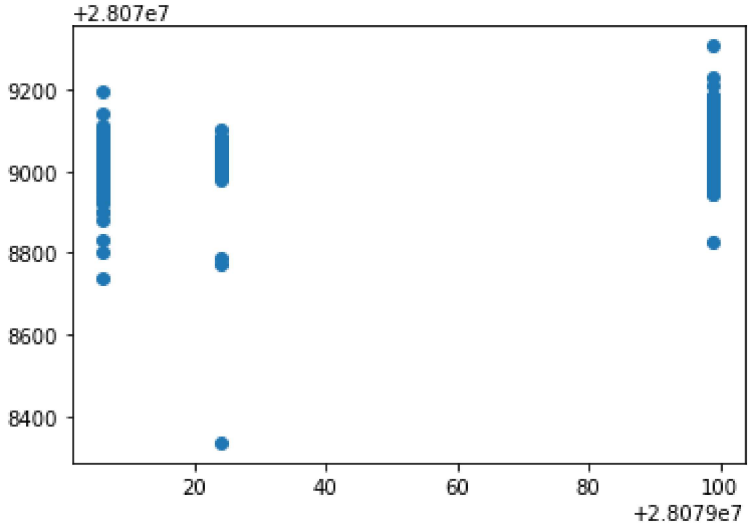
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-19.353050
CO	-11.851839
EBE	-23.768054
MXY	5.086710
NMHC	125.009137
NO_2	-0.025754
NOx	0.013150
OXY	15.332449
O_3	-0.042112
PM10	0.129803
PXY	5.143647
SO_2	-0.615478
TCH	19.230722
TOL	-0.487797

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1aaa2009ca0>
```



ACCURACY

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

Out[28]: 0.3678236838296889

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

Out[29]: 0.4007488405200039

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]: 0.3662072101375202

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

Out[33]: 0.40014204757597593

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]:
```
0.061956048088042004
```

Accuracy(Lasso)

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

Out[36]:
```
0.05954870455102246
```

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([-8.78188649e+00,  0.00000000e+00, -9.08045289e+00,  3.50188837e+00,
        4.11475659e-01, -1.72906950e-03,  1.05117053e-02,  3.33006247e+00,
       -1.15603040e-01,  2.96489676e-01,  2.22781817e+00, -4.27955409e-01,
        5.43787582e-01, -1.00362244e+00])
```

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

Out[39]:
```
28079051.61951336
```

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

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

Out[41]:
```
0.23408583464683075
```

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)))
```

```
32.05616047198387
1260.0014673130768
35.496499367023176
```

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]:
```
(24758, 14)
```

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

Out[46]:
```
(24758,)
```

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]:
```
0.8741416915744405
```

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

Out[54]:
```
3.5557727473608076e-15
```

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

Out[55]:
```
array([[3.55577275e-15, 7.80743173e-29, 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]:
```
0.878534333525678
```

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

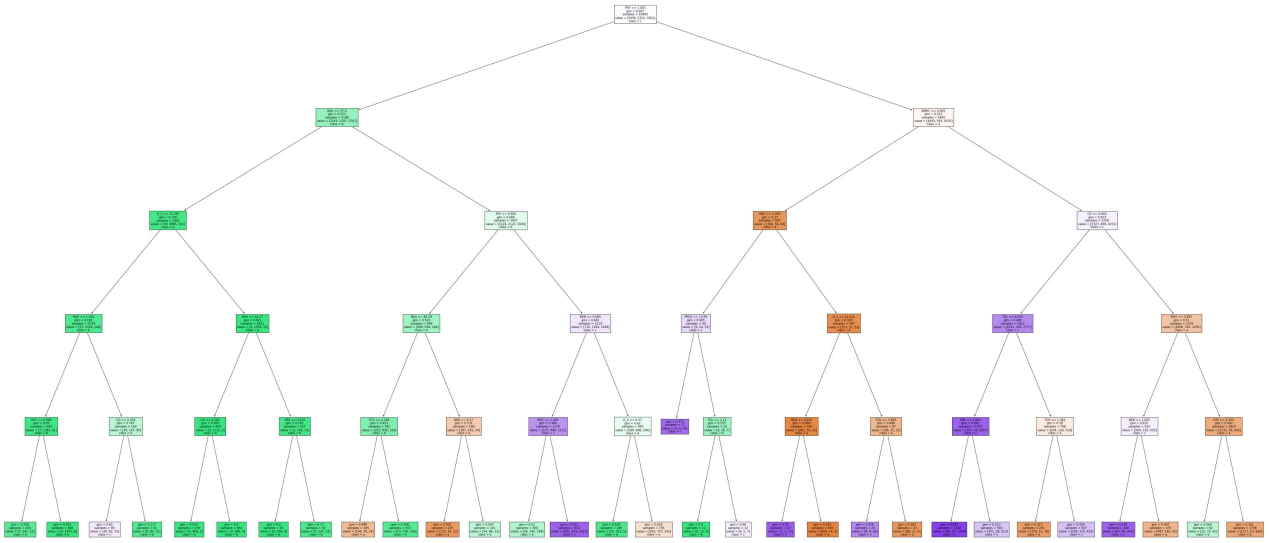
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2241.3, 1993.2, 'PXY <= 1.005\ngini = 0.667\nsamples = 10993\nvalue = [5656, 5752,
5922]\nclass = c'),
 Text(1190.4, 1630.8000000000002, 'NOx <= 27.0\ngini = 0.523\nsamples = 5148\nvalue = [1
163, 5207, 1767]\nclass = b'),
```

```
 Text(595.2, 1268.4, 'O_3 <= 71.255\ngini = 0.116\nsamples = 2081\nvalue = [39, 3085, 16
1]\nclass = b'),
 Text(297.6, 906.0, 'MXY <= 1.035\ngini = 0.195\nsamples = 1070\nvalue = [37, 1529, 146]
\nclass = b'),
 Text(148.8, 543.5999999999999, 'OXY <= 0.995\ngini = 0.09\nsamples = 910\nvalue = [7, 1
382, 61]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.259\nsamples = 250\nvalue = [7, 341, 53]\nclas
s = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.015\nsamples = 660\nvalue = [0,
1041, 8]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'CO <= 0.355\ngini = 0.567\nsamples = 160\n
value = [30, 147, 85]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.63\nsamples = 99\nvalue = [30, 62, 73]\nclass
= c'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.217\nsamples = 61\nvalue = [0, 8
5, 12]\nclass = b'),
 Text(892.8000000000001, 906.0, 'NOx <= 18.77\ngini = 0.021\nsamples = 1011\nvalue = [2,
1556, 15]\nclass = b'),
 Text(744.0, 543.5999999999999, 'CO <= 0.265\ngini = 0.005\nsamples = 854\nvalue = [0, 1
315, 3]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.014\nsamples = 270\nvalue = [0, 409, 3]\nclas
s = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.0\nsamples = 584\nvalue = [0, 90
6, 0]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'EBE <= 0.575\ngini = 0.105\nsamples = 157
\nvalue = [2, 241, 12]\nclass = b'),
 Text(967.2, 181.19999999999982, 'gini = 0.0\nsamples = 65\nvalue = [0, 104, 0]\nclass =
b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.17\nsamples = 92\nvalue = [2, 137, 12]\nclas
s = b'),
 Text(1785.6000000000001, 1268.4, 'PXY <= 0.605\ngini = 0.646\nsamples = 3067\nvalue =
[1124, 2122, 1606]\nclass = b'),
 Text(1488.0, 906.0, 'NOx <= 86.19\ngini = 0.522\nsamples = 948\nvalue = [409, 938, 148]
\nclass = b'),
 Text(1339.2, 543.5999999999999, 'TCH <= 1.295\ngini = 0.453\nsamples = 742\nvalue = [22
2, 836, 124]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.489\nsamples = 185\nvalue = [19
9, 78, 24]\nclass = a'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.246\nsamples = 557\nvalue = [23,
758, 100]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'BEN <= 0.57\ngini = 0.531\nsamples = 206\n
value = [187, 102, 24]\nclass = a'),
 Text(1562.4, 181.19999999999982, 'gini = 0.302\nsamples = 105\nvalue = [133, 16, 12]\nc
lass = a'),
 Text(1711.2, 181.19999999999982, 'gini = 0.547\nsamples = 101\nvalue = [54, 86, 12]\ncl
ass = b'),
 Text(2083.2000000000003, 906.0, 'BEN <= 0.685\ngini = 0.642\nsamples = 2119\nvalue = [7
15, 1184, 1458]\nclass = c'),
 Text(1934.4, 543.5999999999999, 'MXY <= 1.005\ngini = 0.484\nsamples = 1174\nvalue = [1
27, 488, 1212]\nclass = c'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.51\nsamples = 360\nvalue = [26,
345, 199]\nclass = b'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.331\nsamples = 814\nvalue = [10
1, 143, 1013]\nclass = c'),
 Text(2232.0, 543.5999999999999, 'O_3 <= 6.37\ngini = 0.62\nsamples = 945\nvalue = [588,
696, 246]\nclass = b'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.158\nsamples = 209\nvalue = [25,
323, 5]\nclass = b'),
 Text(2306.4, 181.19999999999982, 'gini = 0.629\nsamples = 736\nvalue = [563, 373, 241]
\nclass = a'),
 Text(3292.2000000000003, 1630.8000000000002, 'NMHC <= 0.095\ngini = 0.553\nsamples = 58
45\nvalue = [4493, 545, 4155]\nclass = a'),
 Text(2715.6000000000004, 1268.4, 'MXY <= 2.055\ngini = 0.27\nsamples = 587\nvalue = [76
6, 55, 83]\nclass = a'),
 Text(2455.2000000000003, 906.0, 'PM10 <= 13.69\ngini = 0.605\nsamples = 40\nvalue = [9,
```

```
24, 31]\nclass = c'),
 Text(2380.8, 543.5999999999999, 'gini = 0.375\nsamples = 17\nvalue = [3, 4, 24]\nclass
= c'),
 Text(2529.6000000000004, 543.5999999999999, 'TOL <= 1.11\ngini = 0.555\nsamples = 23\nv
alue = [6, 20, 7]\nclass = b'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [0, 15,
0]\nclass = b'),
 Text(2604.0, 181.19999999999982, 'gini = 0.66\nsamples = 12\nvalue = [6, 5, 7]\nclass =
c'),
 Text(2976.0, 906.0, 'O_3 <= 52.015\ngini = 0.183\nsamples = 547\nvalue = [757, 31, 52]
\nclass = a'),
 Text(2827.2000000000003, 543.5999999999999, 'BEN <= 0.635\ngini = 0.099\nsamples = 460
\nvalue = [661, 16, 20]\nclass = a'),
 Text(2752.8, 181.19999999999982, 'gini = 0.29\nsamples = 10\nvalue = [1, 2, 15]\nclass
= c'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.055\nsamples = 450\nvalue = [66
0, 14, 5]\nclass = a'),
 Text(3124.8, 543.5999999999999, 'TOL <= 3.995\ngini = 0.488\nsamples = 87\nvalue = [96,
15, 32]\nclass = a'),
 Text(3050.4, 181.19999999999982, 'gini = 0.476\nsamples = 24\nvalue = [8, 4, 26]\nclass
= c'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.283\nsamples = 63\nvalue = [88,
11, 6]\nclass = a'),
 Text(3868.8, 1268.4, 'CO <= 0.665\ngini = 0.553\nsamples = 5258\nvalue = [3727, 490, 40
72]\nclass = c'),
 Text(3571.2000000000003, 906.0, 'TOL <= 6.925\ngini = 0.448\nsamples = 2523\nvalue = [1
031, 185, 2777]\nclass = c'),
 Text(3422.4, 543.5999999999999, 'EBE <= 1.405\ngini = 0.285\nsamples = 1725\nvalue = [3
97, 59, 2267]\nclass = c'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.081\nsamples = 1160\nvalue = [4
6, 31, 1754]\nclass = c'),
 Text(3496.8, 181.19999999999982, 'gini = 0.513\nsamples = 565\nvalue = [351, 28, 513]\n
class = c'),
 Text(3720.0000000000005, 543.5999999999999, 'TCH <= 1.365\ngini = 0.58\nsamples = 798\n
value = [634, 126, 510]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.317\nsamples = 291\nvalue = [37
6, 11, 78]\nclass = a'),
 Text(3794.4, 181.19999999999982, 'gini = 0.589\nsamples = 507\nvalue = [258, 115, 432]
\nclass = c'),
 Text(4166.400000000001, 906.0, 'MXY <= 5.855\ngini = 0.51\nsamples = 2735\nvalue = [269
6, 305, 1295]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'BEN <= 1.625\ngini = 0.615\nsamples = 915
\nvalue = [564, 210, 633]\nclass = c'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.335\nsamples = 440\nvalue = [67,
65, 541]\nclass = c'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.487\nsamples = 475\nvalue = [49
7, 145, 92]\nclass = a'),
 Text(4315.200000000001, 543.5999999999999, 'PXY <= 2.355\ngini = 0.402\nsamples = 1820
\nvalue = [2132, 95, 662]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.569\nsamples = 82\nvalue = [15, 72, 42]\ncla
ss = b'),
 Text(4389.6, 181.19999999999982, 'gini = 0.361\nsamples = 1738\nvalue = [2117, 23, 620]
\nclass = 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: 0.3678236838296889
Ridge Regression: 0.3662072101375202
Lasso Regression 0.05954870455102246
ElasticNet Regression: 0.23408583464683075
Logistic Regression: 0.8741416915744405
Random Forest: 0.878534333525678
```

Random Forest is suitable for this dataset