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"E:\154\C10_air\csvs_per_year\csvs_per_year\madrid_2003.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779
...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350

243984 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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

Out[6]:

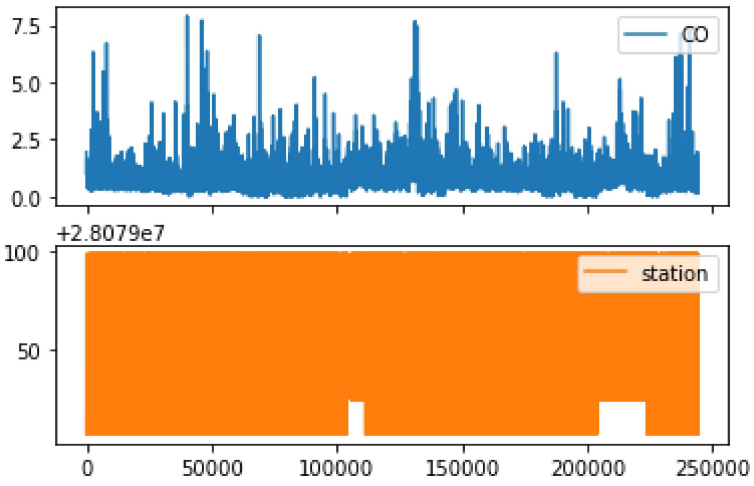
	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

Line chart

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

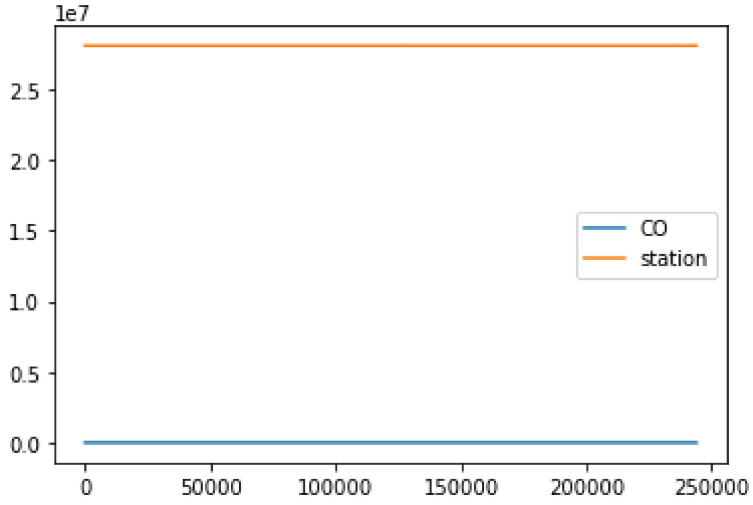
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]: <AxesSubplot:>

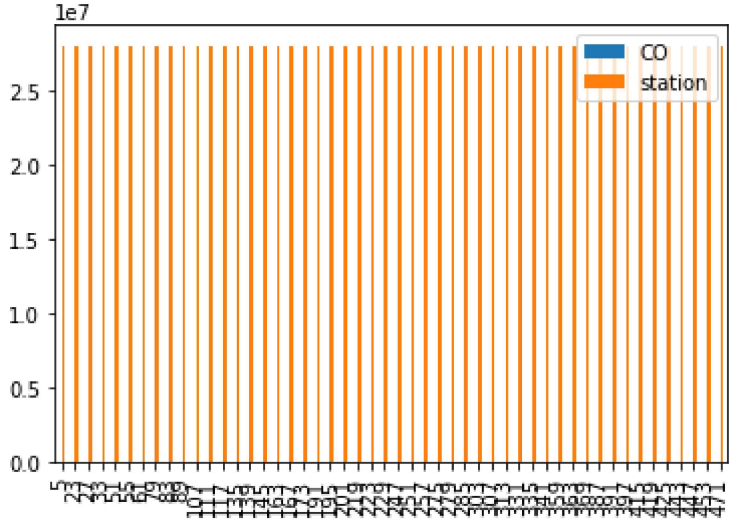


Bar chart

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

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

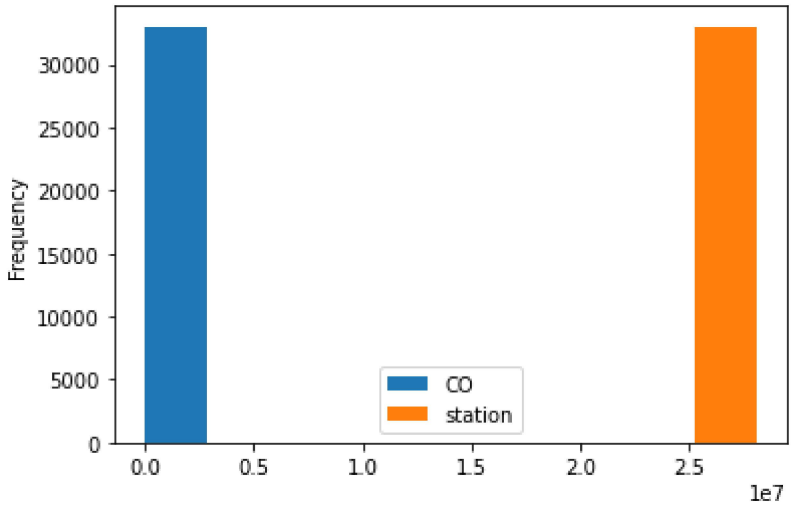
Out[10]: <AxesSubplot:>



Histogram

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

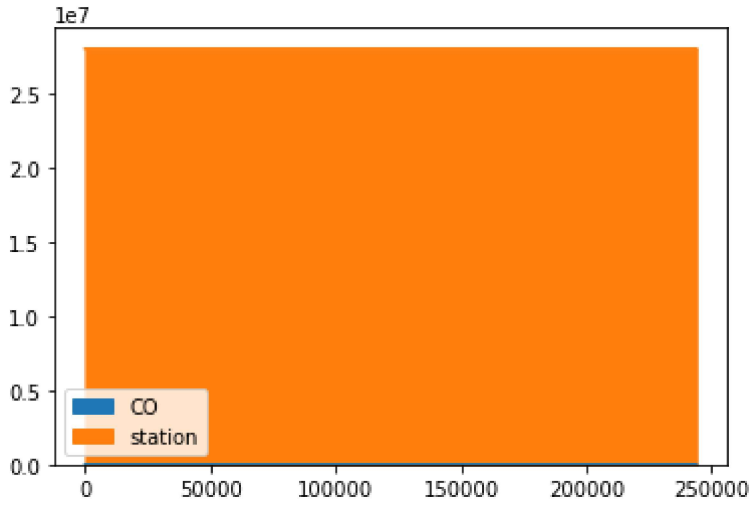
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

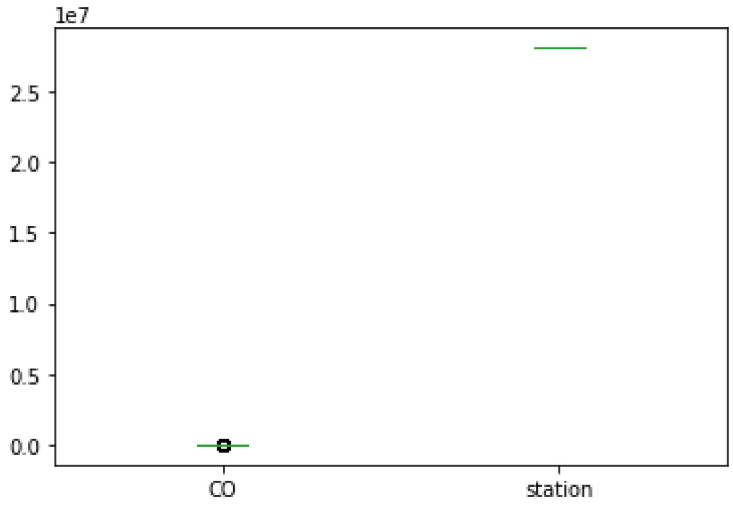
Out[12]: <AxesSubplot:>



Box chart

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

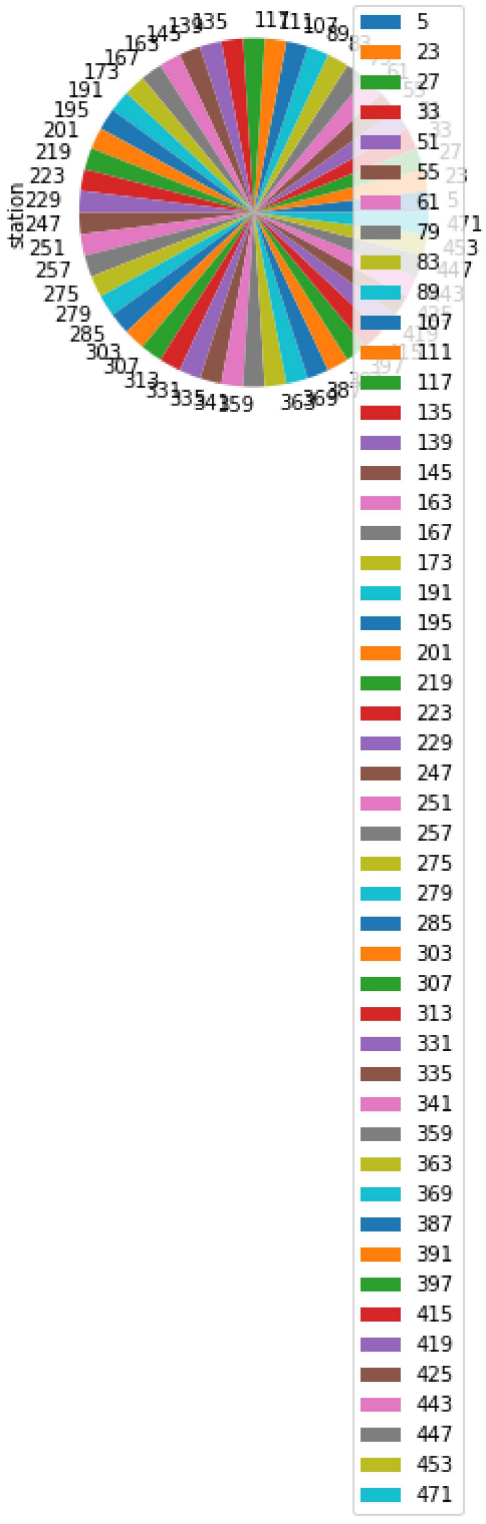
Out[13]:
```
<AxesSubplot:>
```



Pie chart

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

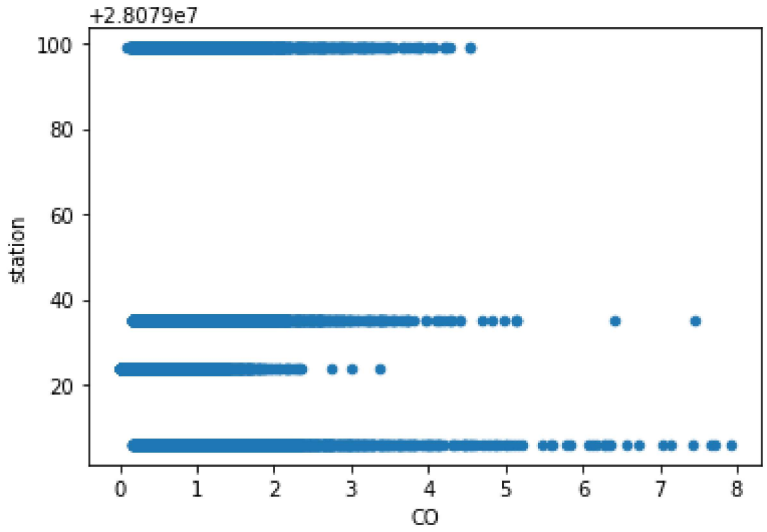
Out[14]: <AxesSubplot:ylabel='station'>



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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
```

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

Out[17]:

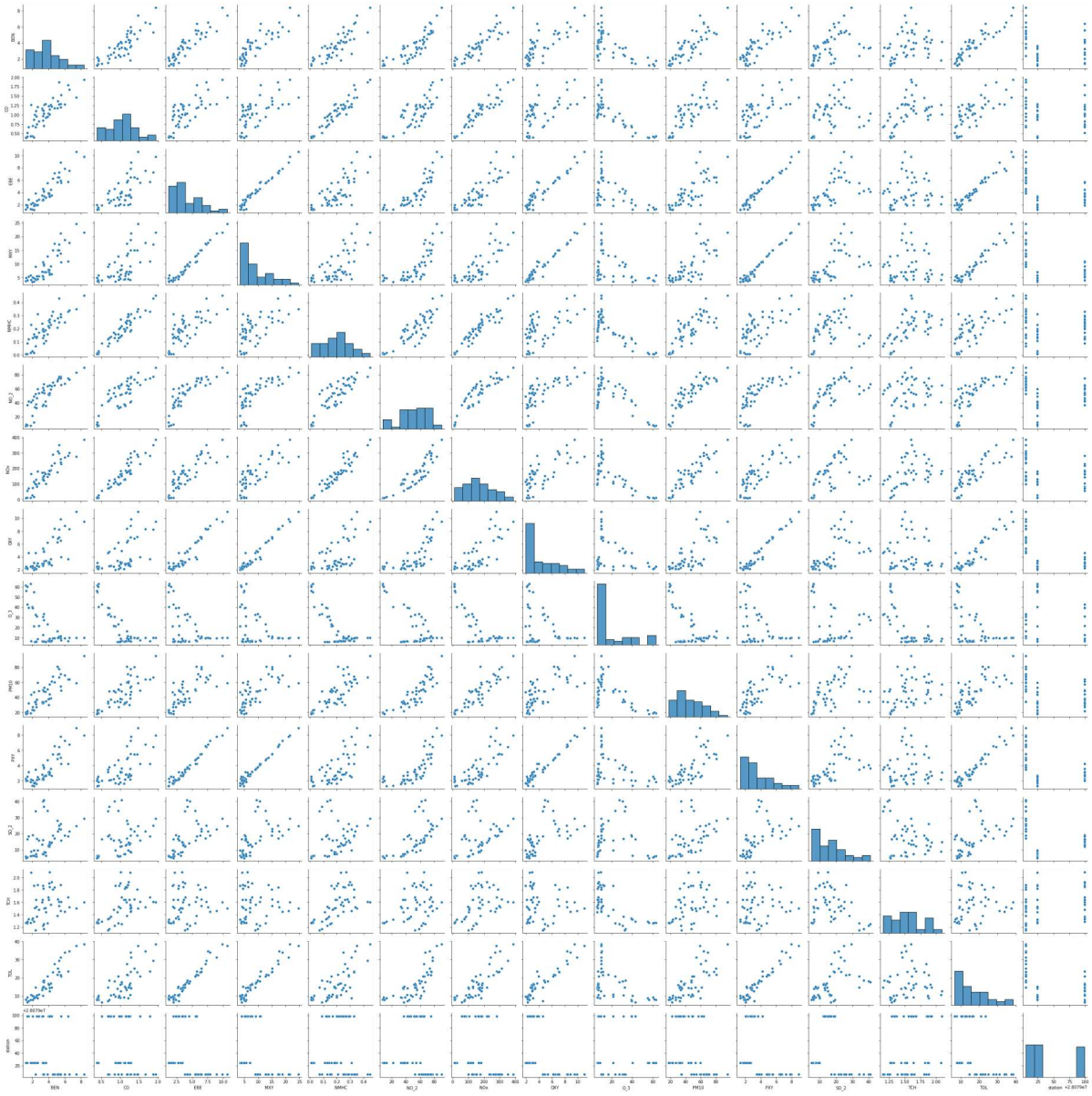
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	330
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049	1
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082	1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999	
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000	
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004	1
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012	12

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])
```

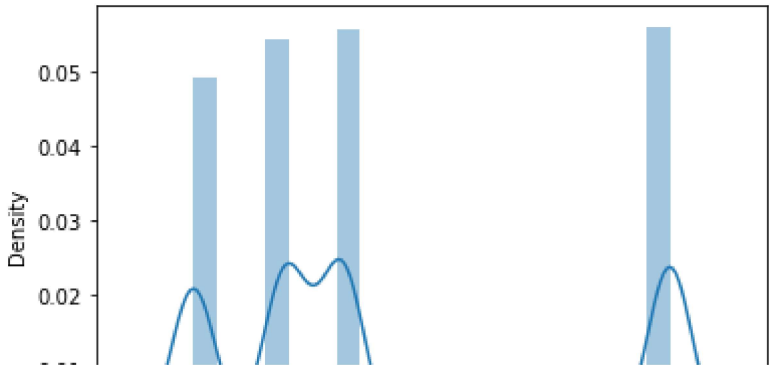
Out[19]: <seaborn.axisgrid.PairGrid at 0x22fc04f3130>



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

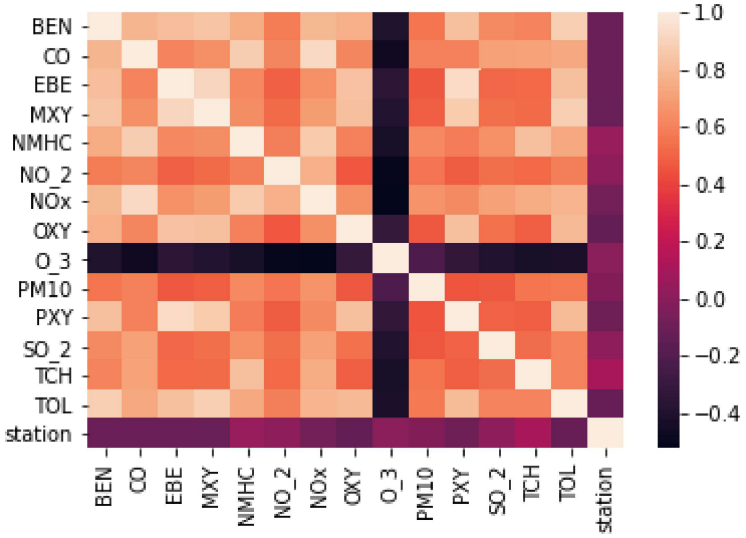
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n 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]:
```
28078998.27129176
```

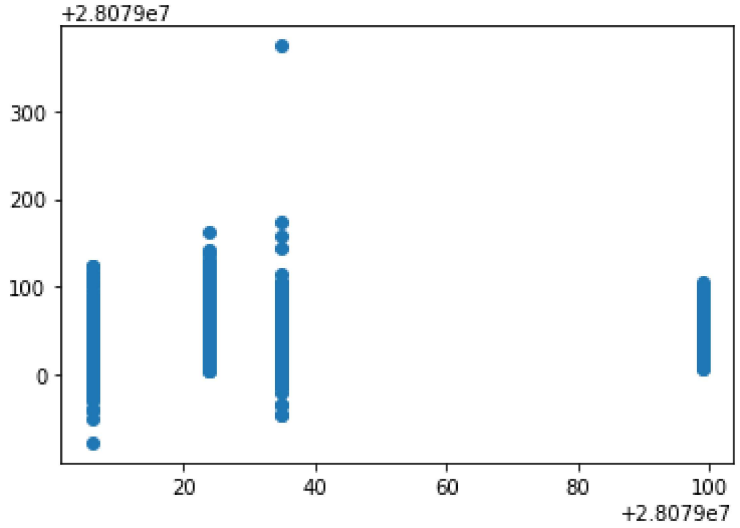
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.718572
CO	-39.642082
EBE	-1.954972
MXY	0.135129
NMHC	157.563030
NO_2	0.158084
NOx	-0.071547
OXY	-1.343282
O_3	-0.011835
PM10	-0.059124
PXY	2.184553
SO_2	0.886083
TCH	37.719728
TOL	-0.914018

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x22fd01f4700>
```



ACCURACY

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

Out[28]:
```
0.1622915136543771
```

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

Out[29]:
```
0.1816069852008838
```

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]:
```
0.16237517062390638
```

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

Out[33]:
```
0.18046734811263498
```

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[34]:
```
Lasso(alpha=10)
```

Accuracy(Lasso)

In [35]:
```
la.score(x_train,y_train)
```

Out[35]:
```
0.037290057892523265
```

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

Out[36]:
```
0.03504239491387462
```

ElasticNet

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.        , -0.29527316,  0.        , -0.01674255,  0.10642374,
        0.14440061, -0.067129  , -1.22671931, -0.04768134,  0.08080325,
        0.40578361,  0.77598315,  1.55183148, -0.49893422])
```

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

Out[39]:
```
28079037.740166444
```

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

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

Out[41]:
```
0.04543638768596503
```

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)))
```

```
29.208533545385542
1185.995883631697
34.43829095108666
```

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
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:
```
feature_matrix.shape
```

Out[45]:
```
(33010, 14)
```

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

Out[46]:
```
(33010,)
```

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]:
```
0.7584974250227204
```

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

Out[54]:
```
2.3306153265290618e-23
```

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

Out[55]:
```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457491e-16]])
```

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="ac
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]:
```
0.7366161896641471
```

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

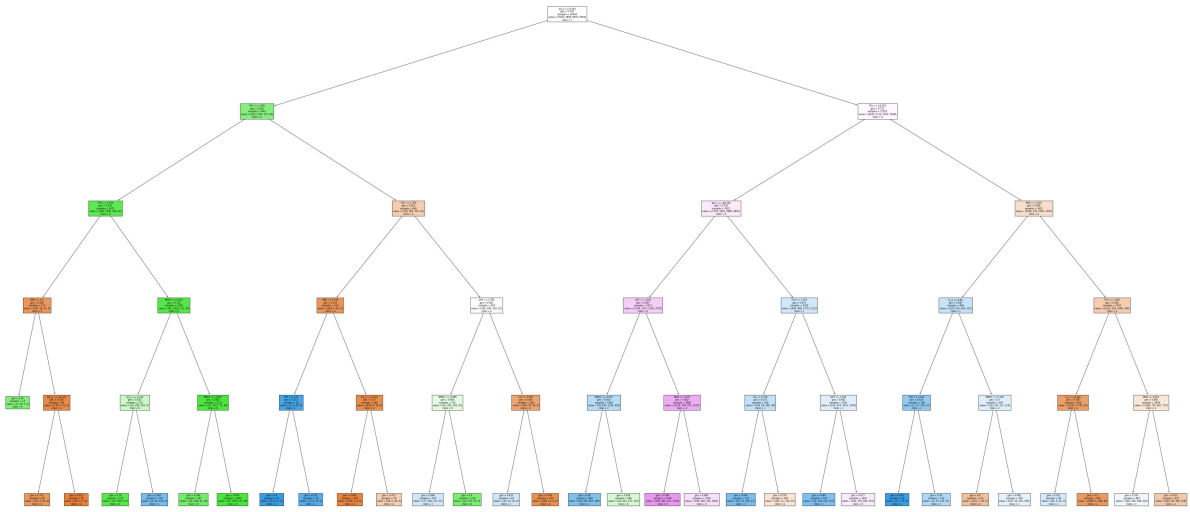
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2131.377049180328, 1993.2, 'SO_2 <= 6.335\ngini = 0.749\nsamples = 1459
4\nvalue = [5203, 5858, 6053, 5993]\nclass = c'),
 Text(969.639344262295, 1630.8000000000002, 'PXY <= 1.185\ngini = 0.455\nsamp
les = 3341\nvalue = [910, 3744, 531, 85]\nclass = b'),
 Text(402.4918032786885, 1268.4, 'TCH <= 1.225\ngini = 0.215\nsamples = 2472
\nvalue = [184, 3439, 228, 44]\nclass = b'),
 Text(146.36065573770492, 906.0, 'BEN <= 0.4\ngini = 0.294\nsamples = 113\nva
lue = [143, 16, 13, 0]\nclass = a'),
 Text(73.18032786885246, 543.5999999999999, 'gini = 0.43\nsamples = 14\nvalue
= [4, 16, 2, 0]\nclass = b'),
 Text(219.54098360655738, 543.5999999999999, 'NO_2 <= 26.125\ngini = 0.136\ns
amples = 99\nvalue = [139, 0, 11, 0]\nclass = a'),
 Text(146.36065573770492, 181.19999999999982, 'gini = 0.271\nsamples = 40\nva
lue = [52, 0, 10, 0]\nclass = a'),
 Text(292.72131147540983, 181.19999999999982, 'gini = 0.022\nsamples = 59\nva
lue = [87, 0, 1, 0]\nclass = a'),
 Text(658.6229508196722, 906.0, 'NMHC <= 0.015\ngini = 0.151\nsamples = 2359
\nvalue = [41, 3423, 215, 44]\nclass = b'),
 Text(512.2622950819672, 543.5999999999999, 'SO_2 <= 5.195\ngini = 0.521\nsam
ples = 222\nvalue = [14, 206, 143, 0]\nclass = b'),
 Text(439.08196721311475, 181.19999999999982, 'gini = 0.19\nsamples = 112\nva
lue = [10, 166, 9, 0]\nclass = b'),
 Text(585.4426229508197, 181.19999999999982, 'gini = 0.382\nsamples = 110\nva
lue = [4, 40, 134, 0]\nclass = c'),
 Text(804.983606557377, 543.5999999999999, 'NMHC <= 0.035\ngini = 0.083\nsamp
les = 2137\nvalue = [27, 3217, 72, 44]\nclass = b'),
 Text(731.8032786885246, 181.19999999999982, 'gini = 0.364\nsamples = 187\nva
lue = [12, 244, 41, 14]\nclass = b'),
 Text(878.1639344262295, 181.19999999999982, 'gini = 0.049\nsamples = 1950\nv
alue = [15, 2973, 31, 30]\nclass = b'),
 Text(1536.7868852459017, 1268.4, 'TCH <= 1.255\ngini = 0.623\nsamples = 869
\nvalue = [726, 305, 303, 41]\nclass = a'),
 Text(1244.0655737704917, 906.0, 'BEN <= 0.705\ngini = 0.254\nsamples = 310\n
value = [426, 9, 64, 0]\nclass = a'),
 Text(1097.704918032787, 543.5999999999999, 'OXY <= 1.27\ngini = 0.117\nsampl
es = 20\nvalue = [2, 0, 30, 0]\nclass = c'),
 Text(1024.5245901639344, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalu
e = [0, 0, 17, 0]\nclass = c'),
 Text(1170.8852459016393, 181.19999999999982, 'gini = 0.231\nsamples = 10\nva
lue = [2, 0, 13, 0]\nclass = c'),
 Text(1390.4262295081967, 543.5999999999999, 'SO_2 <= 5.525\ngini = 0.17\nsam
ples = 290\nvalue = [424, 9, 34, 0]\nclass = a'),
 Text(1317.2459016393443, 181.19999999999982, 'gini = 0.065\nsamples = 240\nv
alue = [376, 9, 4, 0]\nclass = a'),
 Text(1463.6065573770493, 181.19999999999982, 'gini = 0.473\nsamples = 50\nva
lue = [48, 0, 30, 0]\nclass = a'),
 Text(1829.5081967213114, 906.0, 'OXY <= 2.785\ngini = 0.692\nsamples = 559\n
value = [300, 296, 239, 41]\nclass = a'),
 Text(1683.1475409836066, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.664\nsa
mples = 375\nvalue = [91, 261, 205, 41]\nclass = b'),
 Text(1609.967213114754, 181.19999999999982, 'gini = 0.686\nsamples = 247\nva
lue = [77, 104, 172, 37]\nclass = c'),
 Text(1756.327868852459, 181.19999999999982, 'gini = 0.4\nsamples = 128\nvalu
e = [14, 157, 33, 4]\nclass = b'),
 Text(1975.8688524590164, 543.5999999999999, 'CO <= 0.555\ngini = 0.404\nsamp
les = 184\nvalue = [209, 35, 34, 0]\nclass = a'),
 Text(1902.688524590164, 181.19999999999982, 'gini = 0.633\nsamples = 44\nval
```

```
ue = [19, 14, 30, 0]\nclass = c'),
 Text(2049.0491803278687, 181.19999999999982, 'gini = 0.209\nsamples = 140\nv
alue = [190, 21, 4, 0]\nclass = a'),
 Text(3293.1147540983607, 1630.8000000000002, 'TOL <= 14.035\ngini = 0.722\ns
amples = 11253\nvalue = [4293, 2114, 5522, 5908]\nclass = d'),
 Text(2707.6721311475408, 1268.4, 'NO_2 <= 66.745\ngini = 0.703\nsamples = 78
21\nvalue = [1753, 1835, 3980, 4861]\nclass = d'),
 Text(2414.9508196721313, 906.0, 'OXY <= 1.005\ngini = 0.697\nsamples = 5471
\nvalue = [1294, 1337, 2205, 3739]\nclass = d'),
 Text(2268.590163934426, 543.5999999999999, 'NMHC <= 0.075\ngini = 0.619\nsam
ples = 1483\nvalue = [78, 593, 1234, 420]\nclass = c'),
 Text(2195.409836065574, 181.19999999999982, 'gini = 0.49\nsamples = 894\nval
ue = [64, 180, 963, 195]\nclass = c'),
 Text(2341.7704918032787, 181.19999999999982, 'gini = 0.654\nsamples = 589\nv
alue = [14, 413, 271, 225]\nclass = b'),
 Text(2561.311475409836, 543.5999999999999, 'BEN <= 1.675\ngini = 0.642\nsamp
les = 3988\nvalue = [1216, 744, 971, 3319]\nclass = d'),
 Text(2488.1311475409834, 181.19999999999982, 'gini = 0.584\nsamples = 2444\n
value = [456, 460, 617, 2314]\nclass = d'),
 Text(2634.4918032786886, 181.19999999999982, 'gini = 0.689\nsamples = 1544\n
value = [760, 284, 354, 1005]\nclass = d'),
 Text(3000.3934426229507, 906.0, 'TCH <= 1.355\ngini = 0.672\nsamples = 2350
\nvalue = [459, 498, 1775, 1122]\nclass = c'),
 Text(2854.032786885246, 543.5999999999999, 'CO <= 0.735\ngini = 0.573\nsampl
es = 419\nvalue = [234, 24, 362, 44]\nclass = c'),
 Text(2780.8524590163934, 181.19999999999982, 'gini = 0.444\nsamples = 210\nv
alue = [53, 13, 226, 21]\nclass = c'),
 Text(2927.2131147540986, 181.19999999999982, 'gini = 0.579\nsamples = 209\nv
alue = [181, 11, 136, 23]\nclass = a'),
 Text(3146.754098360656, 543.5999999999999, 'OXY <= 1.285\ngini = 0.663\nsamp
les = 1931\nvalue = [225, 474, 1413, 1078]\nclass = c'),
 Text(3073.5737704918033, 181.19999999999982, 'gini = 0.486\nsamples = 529\nv
alue = [43, 104, 613, 127]\nclass = c'),
 Text(3219.934426229508, 181.19999999999982, 'gini = 0.677\nsamples = 1402\nv
alue = [182, 370, 800, 951]\nclass = d'),
 Text(3878.55737704918, 1268.4, 'BEN <= 3.035\ngini = 0.658\nsamples = 3432\n
value = [2540, 279, 1542, 1047]\nclass = a'),
 Text(3585.8360655737706, 906.0, 'O_3 <= 6.65\ngini = 0.687\nsamples = 669\nv
alue = [224, 120, 493, 251]\nclass = c'),
 Text(3439.4754098360654, 543.5999999999999, 'TCH <= 1.415\ngini = 0.519\nsam
ples = 160\nvalue = [2, 57, 176, 37]\nclass = c'),
 Text(3366.2950819672133, 181.19999999999982, 'gini = 0.034\nsamples = 28\nva
lue = [1, 0, 57, 0]\nclass = c'),
 Text(3512.655737704918, 181.19999999999982, 'gini = 0.59\nsamples = 132\nval
ue = [1, 57, 119, 37]\nclass = c'),
 Text(3732.1967213114754, 543.5999999999999, 'NMHC <= 0.105\ngini = 0.7\nsamp
les = 509\nvalue = [222, 63, 317, 214]\nclass = c'),
 Text(3659.0163934426228, 181.19999999999982, 'gini = 0.5\nsamples = 114\nval
ue = [121, 7, 56, 5]\nclass = a'),
 Text(3805.377049180328, 181.19999999999982, 'gini = 0.682\nsamples = 395\nva
lue = [101, 56, 261, 209]\nclass = c'),
 Text(4171.2786885245905, 906.0, 'TCH <= 1.425\ngini = 0.618\nsamples = 2763
\nvalue = [2316, 159, 1049, 796]\nclass = a'),
 Text(4024.9180327868853, 543.5999999999999, 'O_3 <= 9.285\ngini = 0.336\nsam
ples = 904\nvalue = [1128, 4, 242, 43]\nclass = a'),
 Text(3951.7377049180327, 181.19999999999982, 'gini = 0.522\nsamples = 68\nva
lue = [42, 0, 54, 3]\nclass = c'),
```

```
 Text(4098.098360655737, 181.19999999999982, 'gini = 0.3\nsamples = 836\nvalu
e = [1086, 4, 188, 40]\nclass = a'),
 Text(4317.639344262295, 543.5999999999999, 'BEN <= 4.925\ngini = 0.685\nsamp
les = 1859\nvalue = [1188, 155, 807, 753]\nclass = a'),
 Text(4244.459016393443, 181.19999999999982, 'gini = 0.705\nsamples = 887\nva
lue = [363, 106, 498, 424]\nclass = c'),
 Text(4390.819672131148, 181.19999999999982, 'gini = 0.612\nsamples = 972\nva
lue = [825, 49, 309, 329]\nclass = a')]
```



Accuracy ¶

Linear Regression

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

Out[64]: 0.1816069852008838

Ridge Regression

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

Out[65]: 0.18046734811263498

Lasso Regression

In [66]:
```
la.score(x_test,y_test)
```

Out[66]: 0.03504239491387462

ElasticNet Regression

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

Out[67]: 0.04543638768596503

Logistic Regression

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

Out[68]: 0.7584974250227204

Random Forest

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

Out[69]: 0.7366161896641471

Conclusion

Logistic Regression is suitable for this dataset