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_2010.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.41(
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.67(
...	...	...	...	...	...	...	...	...	...	...	
209443	2010-08-01 00:00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	
209444	2010-08-01 00:00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.25(
209445	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	
209446	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	
209447	2010-08-01 00:00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.15(

209448 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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     6666 non-null   object 
 1   BEN      6666 non-null   float64
 2   CO       6666 non-null   float64
 3   EBE      6666 non-null   float64
 4   MXY      6666 non-null   float64
 5   NMHC     6666 non-null   float64
 6   NO_2     6666 non-null   float64
 7   NOx      6666 non-null   float64
 8   OXY      6666 non-null   float64
 9   O_3      6666 non-null   float64
 10  PM10     6666 non-null   float64
 11  PM25     6666 non-null   float64
 12  PXY      6666 non-null   float64
 13  SO_2     6666 non-null   float64
 14  TCH      6666 non-null   float64
 15  TOL      6666 non-null   float64
 16  station  6666 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 937.4+ KB
```

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

Out[6]:

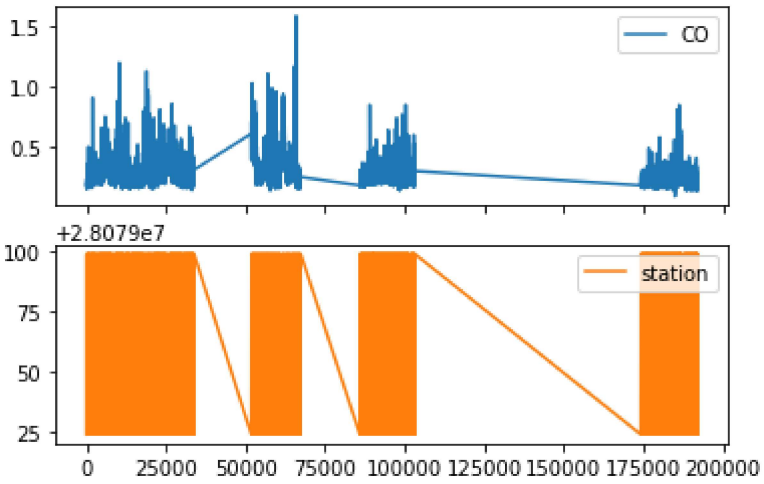
	CO	station
11	0.18	28079024
23	0.23	28079099
35	0.17	28079024
47	0.21	28079099
59	0.16	28079024
...	...	...
191879	0.26	28079099
191891	0.16	28079024
191903	0.28	28079099
191915	0.16	28079024
191927	0.25	28079099

6666 rows × 2 columns

Line chart

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

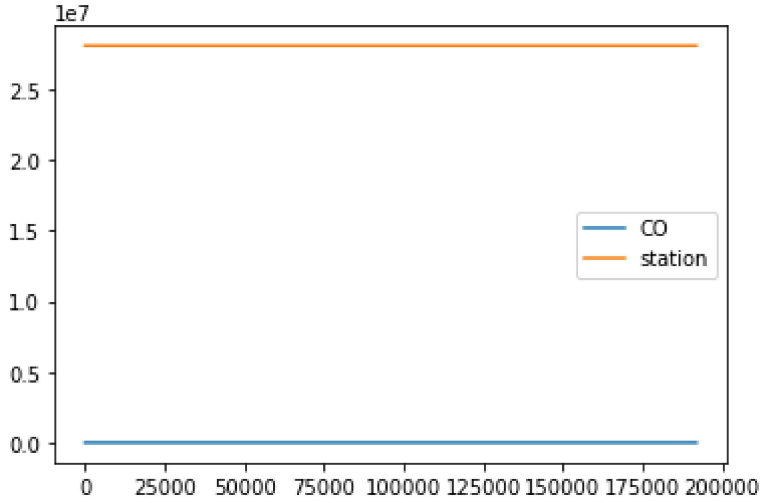
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]: <AxesSubplot:>

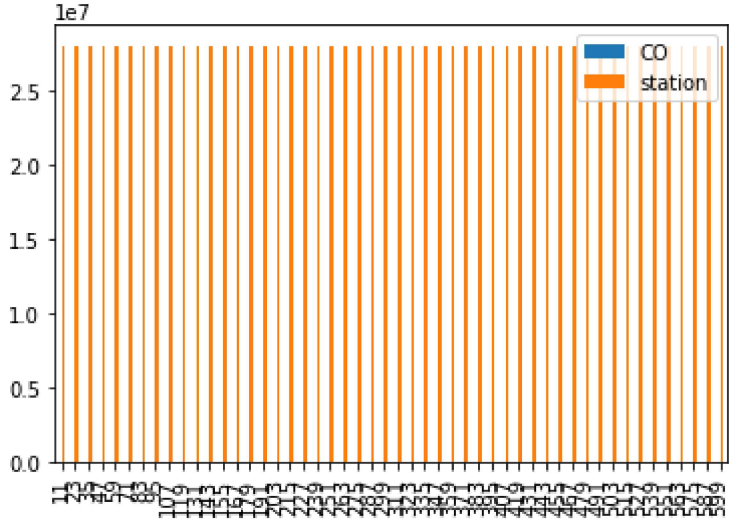


Bar chart

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

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

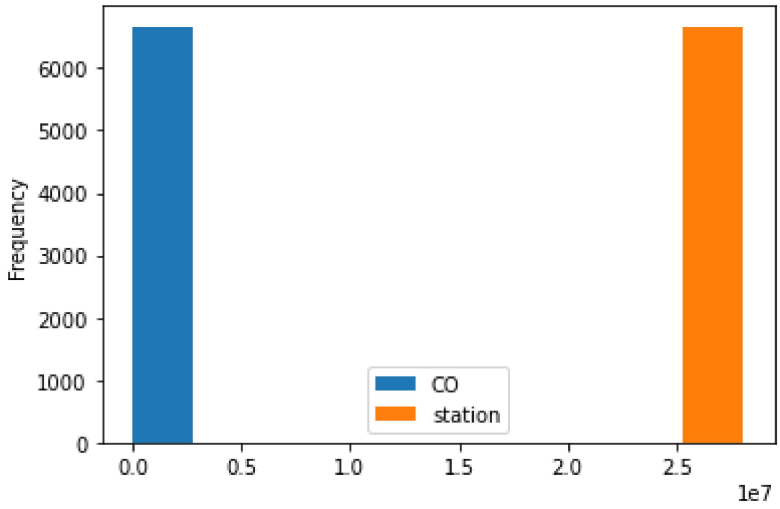
Out[10]: <AxesSubplot:>



Histogram

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

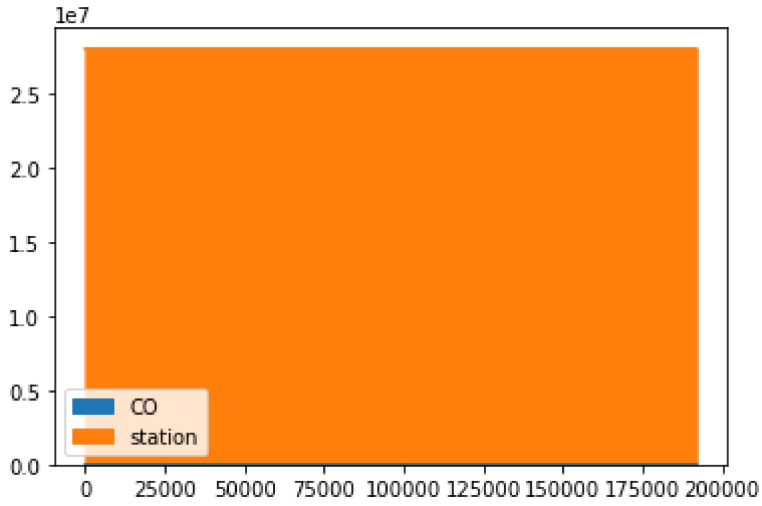
Out[11]:
```
<AxesSubplot:ylabel='Frequency'>
```



Area chart

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

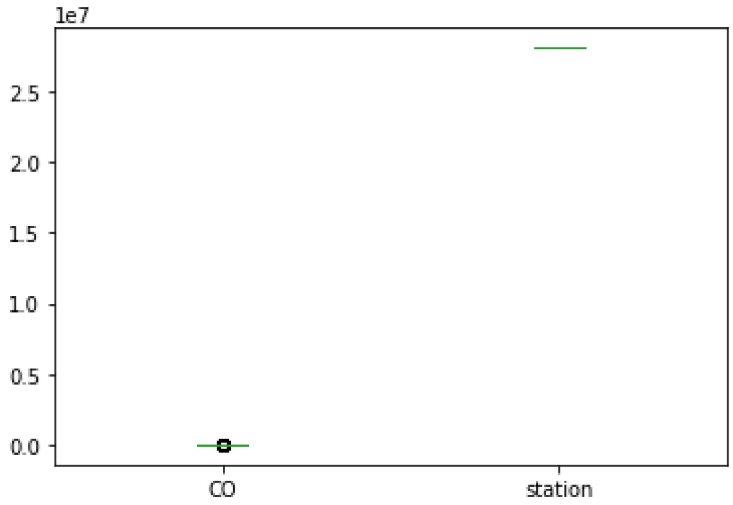
Out[12]:
```
<AxesSubplot:>
```



Box chart

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

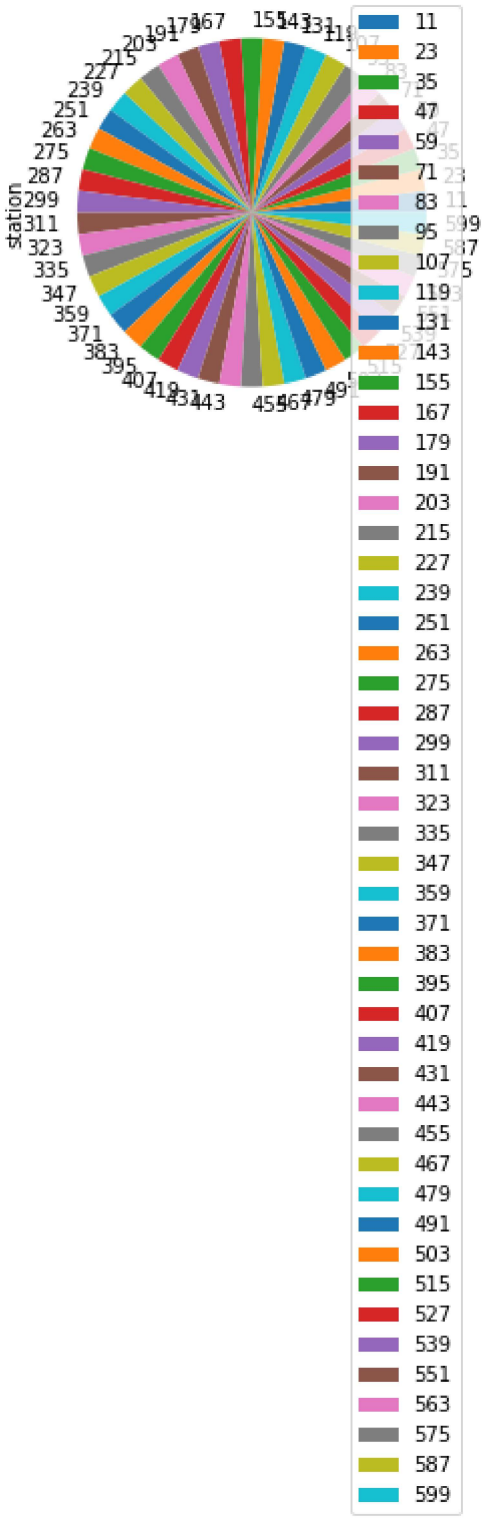
Out[13]:
```
<AxesSubplot:>
```



Pie chart

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

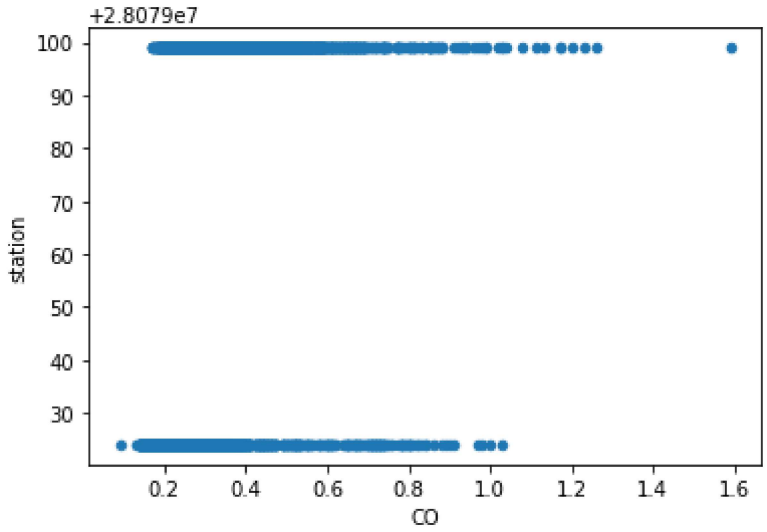
Out[14]: <AxesSubplot:ylabel='station'>



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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     6666 non-null   object 
 1   BEN      6666 non-null   float64
 2   CO       6666 non-null   float64
 3   EBE      6666 non-null   float64
 4   MXY      6666 non-null   float64
 5   NMHC     6666 non-null   float64
 6   NO_2     6666 non-null   float64
 7   NOx      6666 non-null   float64
 8   OXY      6666 non-null   float64
 9   O_3      6666 non-null   float64
 10  PM10     6666 non-null   float64
 11  PM25     6666 non-null   float64
 12  PXY      6666 non-null   float64
 13  SO_2     6666 non-null   float64
```

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

Out[17]:

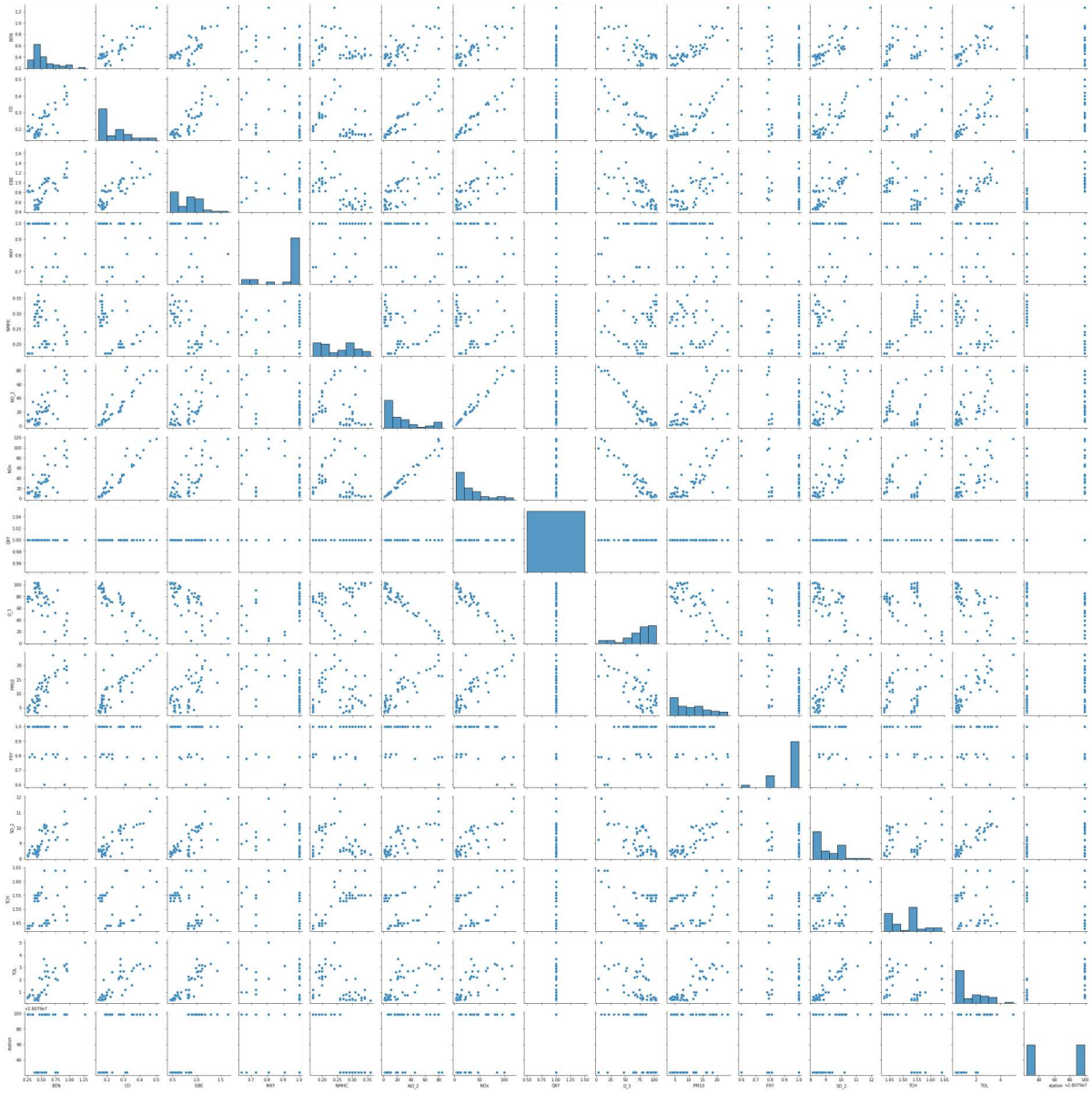
	BEN	CO	EBE	MXY	NMHC	NO_2	NC
count	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.0000
mean	0.648425	0.296280	0.840585	0.839959	0.243378	33.888744	47.5406
std	0.395346	0.133296	0.508031	0.382263	0.115730	23.465169	41.2305
min	0.170000	0.090000	0.140000	0.110000	0.000000	1.290000	2.7600
25%	0.380000	0.200000	0.470000	0.590000	0.180000	15.752500	19.4425
50%	0.540000	0.260000	0.755000	1.000000	0.220000	29.320000	36.7700
75%	0.810000	0.340000	1.000000	1.000000	0.280000	47.657500	62.1025
max	5.110000	1.590000	5.190000	6.810000	0.930000	133.399994	409.2999

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])
```

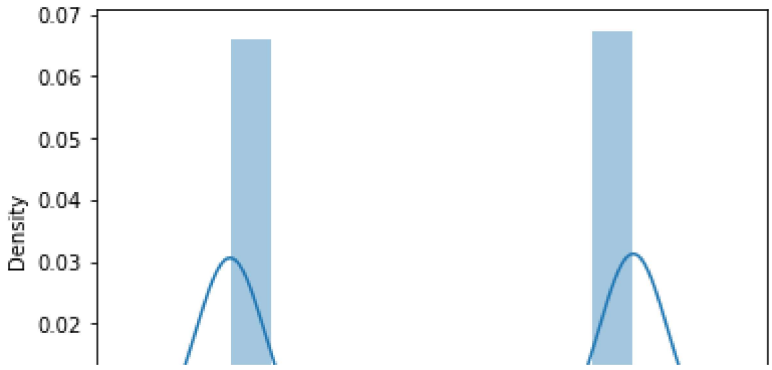
Out[19]: <seaborn.axisgrid.PairGrid at 0x201be8d59a0>



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

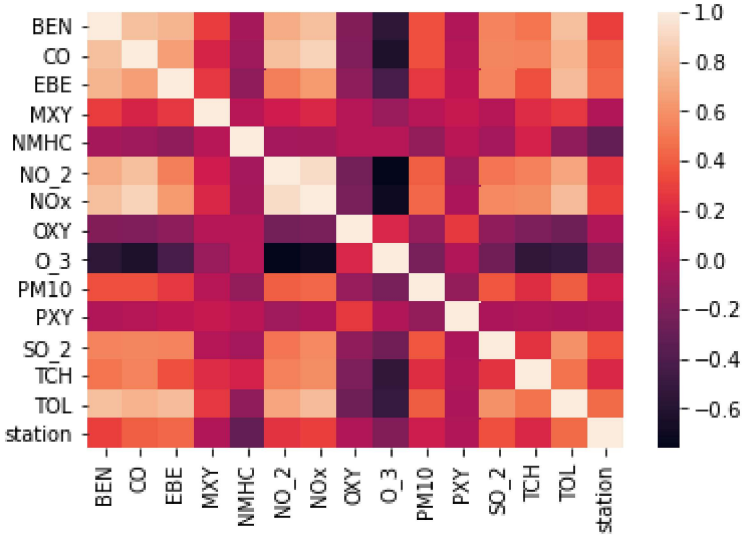
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: F
utureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
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]:
```
28078939.152284157
```

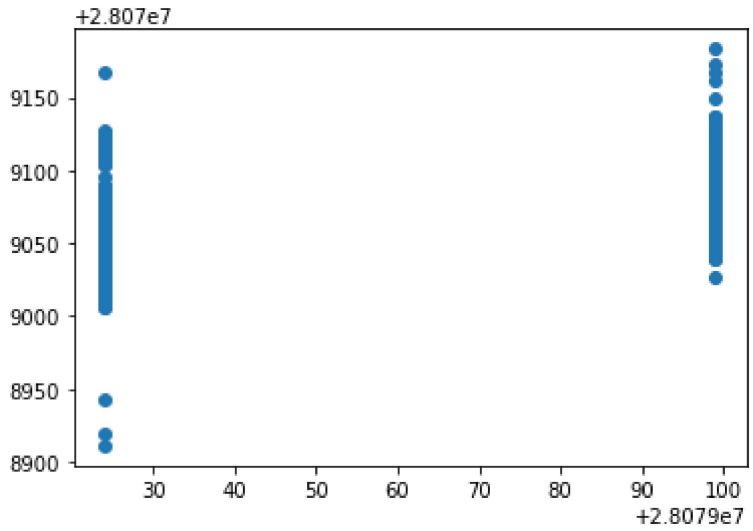
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-40.377766
CO	176.315549
EBE	15.735204
MXY	-8.626732
NMHC	-70.943699
NO_2	0.367366
NOx	-0.677048
OXY	30.041183
O_3	0.066617
PM10	-0.142680
PXY	-5.369936
SO_2	1.497095
TCH	44.624701
TOL	10.867809

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

Out[27]: <matplotlib.collections.PathCollection at 0x201cc8fe520>



ACCURACY

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

Out[28]: 0.41346460594020007

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

Out[29]: 0.43129748182765937

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]: 0.40572304859039277

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

Out[33]:
```
0.41793114685125476
```

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]:
```
0.19216073756348984
```

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

Out[36]:
```
0.16648517018839903
```

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.        ,  0.18708066,  2.61064504, -1.07072423, -1.13990337,
        0.08357608, -0.14546488,  0.42073229, -0.01341173, -0.0960109 ,
       -0.        ,  2.20075866,  0.        ,  7.65424251])
```

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

Out[39]:
```
28079027.114858843
```

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

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

Out[41]:
```
0.21892518042084452
```

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)))
```

```
31.048280765108764
1095.9601293319292
33.10528854023069
```

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]:
```
(6666, 14)
```

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

Out[46]:
```
(6666,)
```

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([28079024, 28079099], dtype=int64)
```

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]:
```
0.8660366036603661
```

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

Out[54]:
```
0.0
```

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

Out[55]:
```
array([[0., 1.]])
```

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]:
```
0.9312044577796827
```

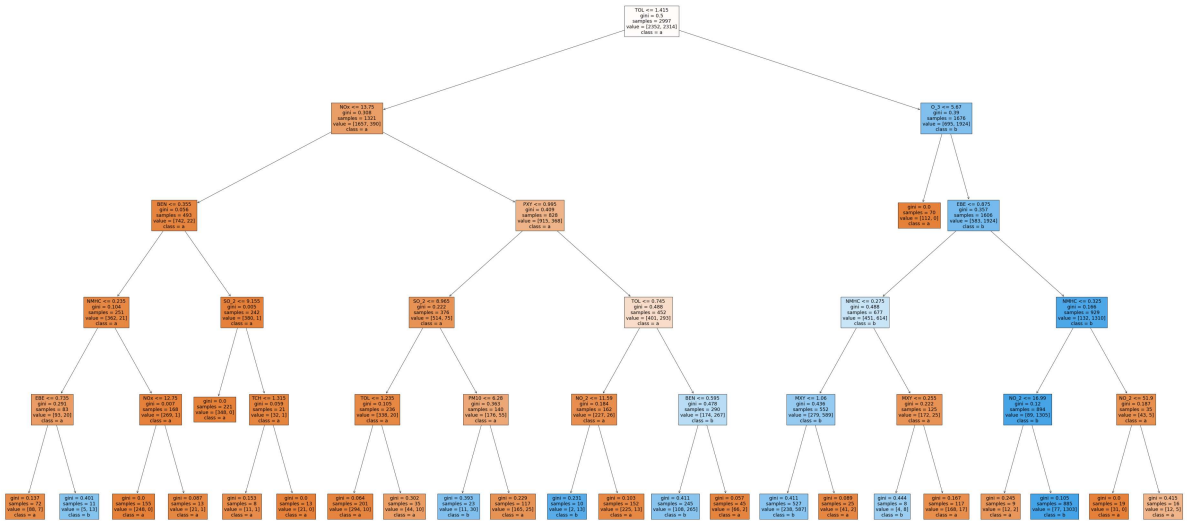
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(2447.590909090909, 1993.2, 'TOL <= 1.415\ngini = 0.5\nsamples = 2997\nv
alue = [2352, 2314]\nclass = a'),
 Text(1344.2727272727273, 1630.8000000000002, 'NOx <= 13.75\ngini = 0.308\nsa
mples = 1321\nvalue = [1657, 390]\nclass = a'),
 Text(659.4545454545455, 1268.4, 'BEN <= 0.355\ngini = 0.056\nsamples = 493\n
value = [742, 22]\nclass = a'),
 Text(405.8181818181818, 906.0, 'NMHC <= 0.235\ngini = 0.104\nsamples = 251\n
value = [362, 21]\nclass = a'),
 Text(202.9090909090909, 543.5999999999999, 'EBE <= 0.735\ngini = 0.291\nsamp
les = 83\nvalue = [93, 20]\nclass = a'),
 Text(101.45454545454545, 181.19999999999982, 'gini = 0.137\nsamples = 72\nva
lue = [88, 7]\nclass = a'),
 Text(304.3636363636364, 181.19999999999982, 'gini = 0.401\nsamples = 11\nval
ue = [5, 13]\nclass = b'),
 Text(608.7272727272727, 543.5999999999999, 'NOx <= 12.75\ngini = 0.007\nsamp
les = 168\nvalue = [269, 1]\nclass = a'),
 Text(507.27272727272725, 181.19999999999982, 'gini = 0.0\nsamples = 155\nval
ue = [248, 0]\nclass = a'),
 Text(710.1818181818181, 181.19999999999982, 'gini = 0.087\nsamples = 13\nval
ue = [21, 1]\nclass = a'),
 Text(913.0909090909091, 906.0, 'SO_2 <= 9.155\ngini = 0.005\nsamples = 242\n
value = [380, 1]\nclass = a'),
 Text(811.6363636363636, 543.5999999999999, 'gini = 0.0\nsamples = 221\nvalue
= [348, 0]\nclass = a'),
 Text(1014.5454545454545, 543.5999999999999, 'TCH <= 1.315\ngini = 0.059\nsam
ples = 21\nvalue = [32, 1]\nclass = a'),
 Text(913.0909090909091, 181.19999999999982, 'gini = 0.153\nsamples = 8\nvalu
e = [11, 1]\nclass = a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.0\nsamples = 13\nvalue = [21, 0]
\nclass = a'),
 Text(2029.090909090909, 1268.4, 'PXY <= 0.995\ngini = 0.409\nsamples = 828\n
value = [915, 368]\nclass = a'),
 Text(1623.2727272727273, 906.0, 'SO_2 <= 8.965\ngini = 0.222\nsamples = 376
\nvalue = [514, 75]\nclass = a'),
 Text(1420.3636363636363, 543.5999999999999, 'TOL <= 1.235\ngini = 0.105\nsam
ples = 236\nvalue = [338, 20]\nclass = a'),
 Text(1318.909090909091, 181.19999999999982, 'gini = 0.064\nsamples = 201\nva
lue = [294, 10]\nclass = a'),
 Text(1521.8181818181818, 181.19999999999982, 'gini = 0.302\nsamples = 35\nva
lue = [44, 10]\nclass = a'),
 Text(1826.1818181818182, 543.5999999999999, 'PM10 <= 6.28\ngini = 0.363\nsam
ples = 140\nvalue = [176, 55]\nclass = a'),
 Text(1724.7272727272727, 181.19999999999982, 'gini = 0.393\nsamples = 23\nva
lue = [11, 30]\nclass = b'),
 Text(1927.6363636363635, 181.19999999999982, 'gini = 0.229\nsamples = 117\nv
alue = [165, 25]\nclass = a'),
 Text(2434.909090909091, 906.0, 'TOL <= 0.745\ngini = 0.488\nsamples = 452\nv
alue = [401, 293]\nclass = a'),
 Text(2232.0, 543.5999999999999, 'NO_2 <= 11.59\ngini = 0.184\nsamples = 162
\nvalue = [227, 26]\nclass = a'),
 Text(2130.5454545454545, 181.19999999999982, 'gini = 0.231\nsamples = 10\nva
lue = [2, 13]\nclass = b'),
 Text(2333.4545454545455, 181.19999999999982, 'gini = 0.103\nsamples = 152\nv
alue = [225, 13]\nclass = a'),
 Text(2637.818181818182, 543.5999999999999, 'BEN <= 0.595\ngini = 0.478\nsamp
les = 290\nvalue = [174, 267]\nclass = b'),
 Text(2536.3636363636365, 181.19999999999982, 'gini = 0.411\nsamples = 245\nv
```

```
alue = [108, 265]\nclass = b'),
 Text(2739.272727272727, 181.19999999999982, 'gini = 0.057\nsamples = 45\nval
ue = [66, 2]\nclass = a'),
 Text(3550.909090909091, 1630.8000000000002, 'O_3 <= 5.67\ngini = 0.39\nsampl
es = 1676\nvalue = [695, 1924]\nclass = b'),
 Text(3449.4545454545455, 1268.4, 'gini = 0.0\nsamples = 70\nvalue = [112, 0]
\nclass = a'),
 Text(3652.3636363636365, 1268.4, 'EBE <= 0.875\ngini = 0.357\nsamples = 1606
\nvalue = [583, 1924]\nclass = b'),
 Text(3246.5454545454545, 906.0, 'NMHC <= 0.275\ngini = 0.488\nsamples = 677
\nvalue = [451, 614]\nclass = b'),
 Text(3043.6363636363635, 543.5999999999999, 'MXY <= 1.06\ngini = 0.436\nsamp
les = 552\nvalue = [279, 589]\nclass = b'),
 Text(2942.181818181818, 181.19999999999982, 'gini = 0.411\nsamples = 527\nva
lue = [238, 587]\nclass = b'),
 Text(3145.090909090909, 181.19999999999982, 'gini = 0.089\nsamples = 25\nval
ue = [41, 2]\nclass = a'),
 Text(3449.4545454545455, 543.5999999999999, 'MXY <= 0.255\ngini = 0.222\nsam
ples = 125\nvalue = [172, 25]\nclass = a'),
 Text(3348.0, 181.19999999999982, 'gini = 0.444\nsamples = 8\nvalue = [4, 8]
\nclass = b'),
 Text(3550.909090909091, 181.19999999999982, 'gini = 0.167\nsamples = 117\nva
lue = [168, 17]\nclass = a'),
 Text(4058.181818181818, 906.0, 'NMHC <= 0.325\ngini = 0.166\nsamples = 929\n
value = [132, 1310]\nclass = b'),
 Text(3855.272727272727, 543.5999999999999, 'NO_2 <= 16.99\ngini = 0.12\nsamp
les = 894\nvalue = [89, 1305]\nclass = b'),
 Text(3753.818181818182, 181.19999999999982, 'gini = 0.245\nsamples = 9\nvalu
e = [12, 2]\nclass = a'),
 Text(3956.7272727272725, 181.19999999999982, 'gini = 0.105\nsamples = 885\nv
alue = [77, 1303]\nclass = b'),
 Text(4261.090909090909, 543.5999999999999, 'NO_2 <= 51.9\ngini = 0.187\nsamp
les = 35\nvalue = [43, 5]\nclass = a'),
 Text(4159.636363636364, 181.19999999999982, 'gini = 0.0\nsamples = 19\nvalue
= [31, 0]\nclass = a'),
 Text(4362.545454545454, 181.19999999999982, 'gini = 0.415\nsamples = 16\nval
ue = [12, 5]\nclass = a')]
```



Accuracy

Linear Regression

In [63]:
```
lr.score(x_train,y_train)
```

Out[63]: 0.43129748182765937

Ridge Regression

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

Out[64]: 0.41793114685125476

Lasso Regression

In [65]:
```
la.score(x_test,y_test)
```

Out[65]: 0.16648517018839903

ElasticNet Regression

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

Out[66]: 0.21892518042084452

Logistic Regression

In [67]:
```
logr.score(fs,target_vector)
```

Out[67]: 0.8660366036603661

Random Forest

In [68]:
```
grid_search.best_score_
```

Out[68]: 0.9312044577796827

Conclusion

Random Forest is suitable for this dataset