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_2005.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.93
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00
...	...	...	...	...	...	...	...	...	...	...	...
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67

237000 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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

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

Out[6]:

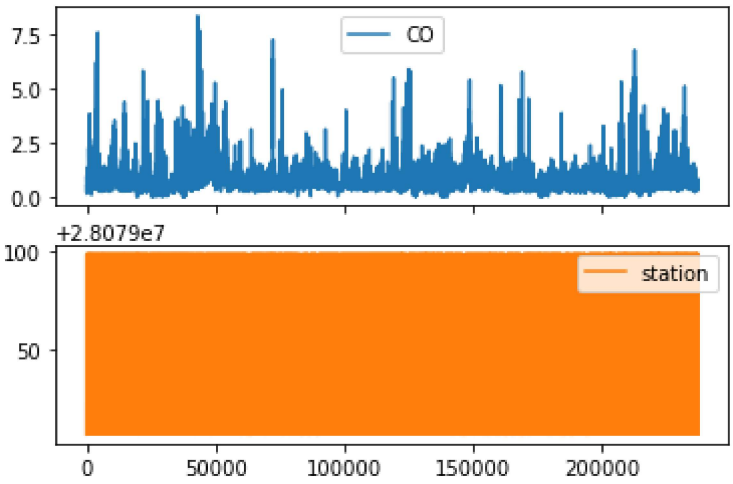
	CO	station
5	0.88	28079006
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

Line chart

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

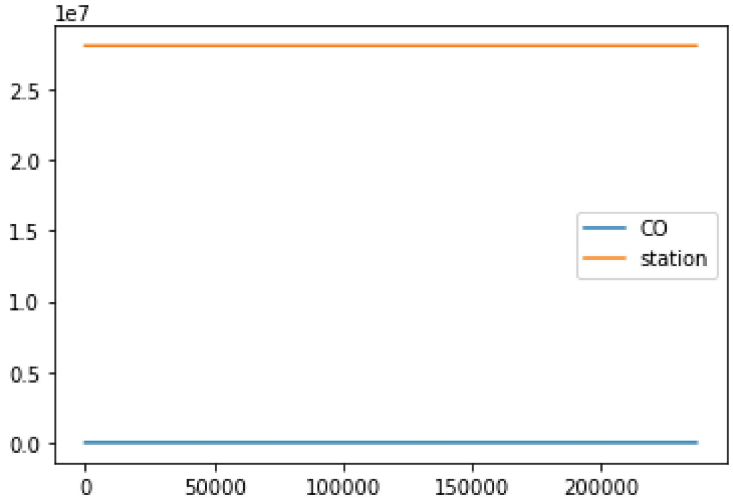
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

Out[8]: <AxesSubplot:>

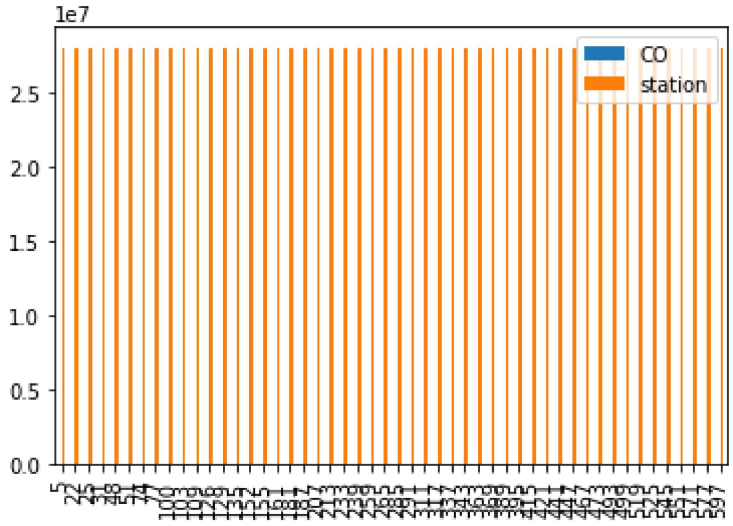


Bar chart

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

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

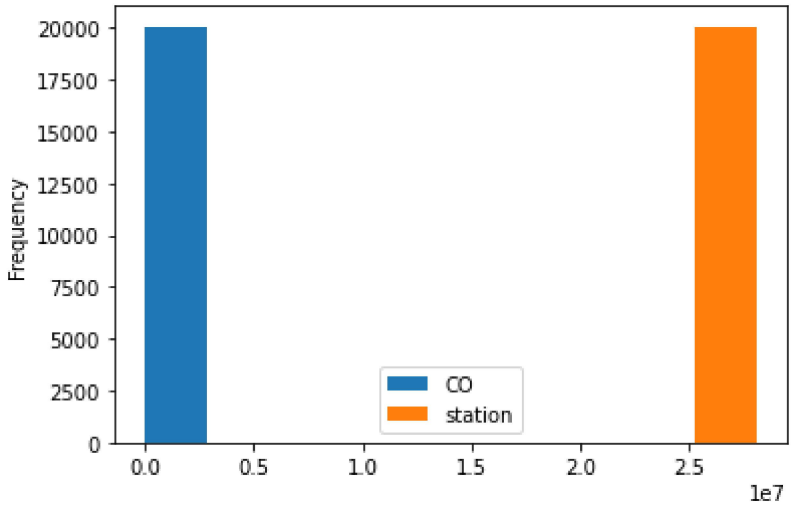
Out[10]: <AxesSubplot:>



Histogram

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

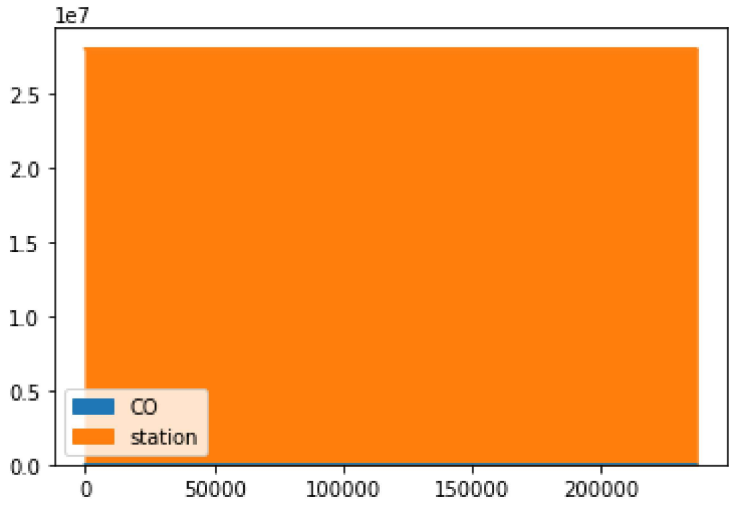
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

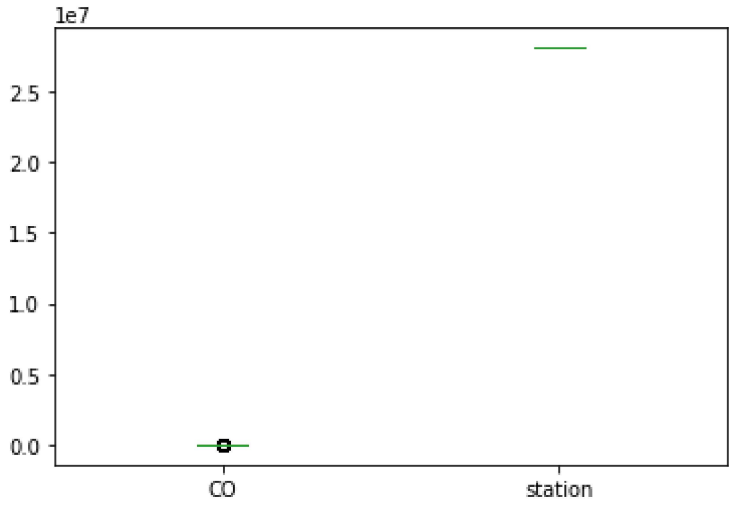
Out[12]: <AxesSubplot:>



Box chart

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

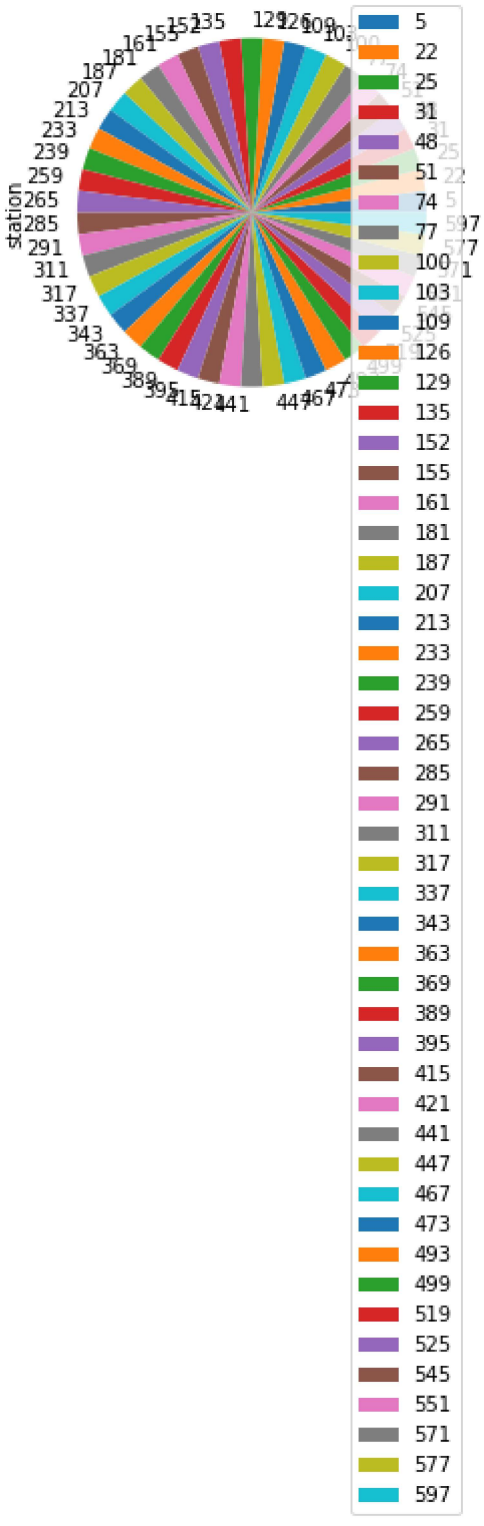
Out[13]: <AxesSubplot:>



Pie chart

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

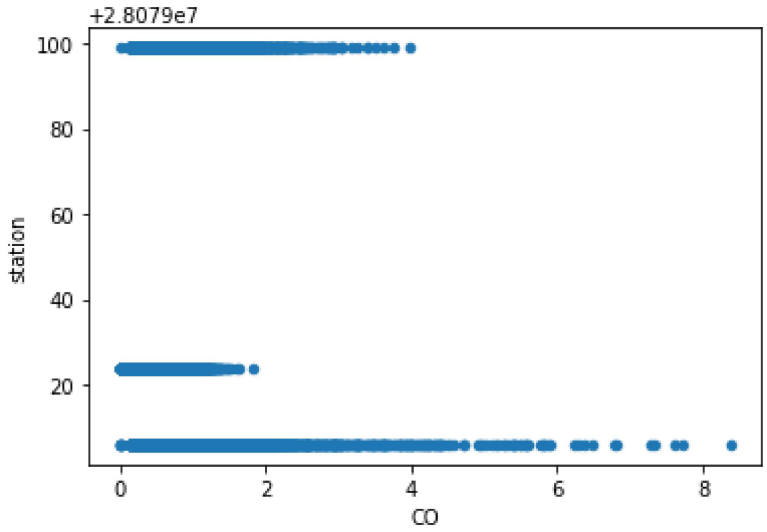
Out[14]: <AxesSubplot:ylabel='station'>



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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     20070 non-null  object
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
```

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

Out[17]:

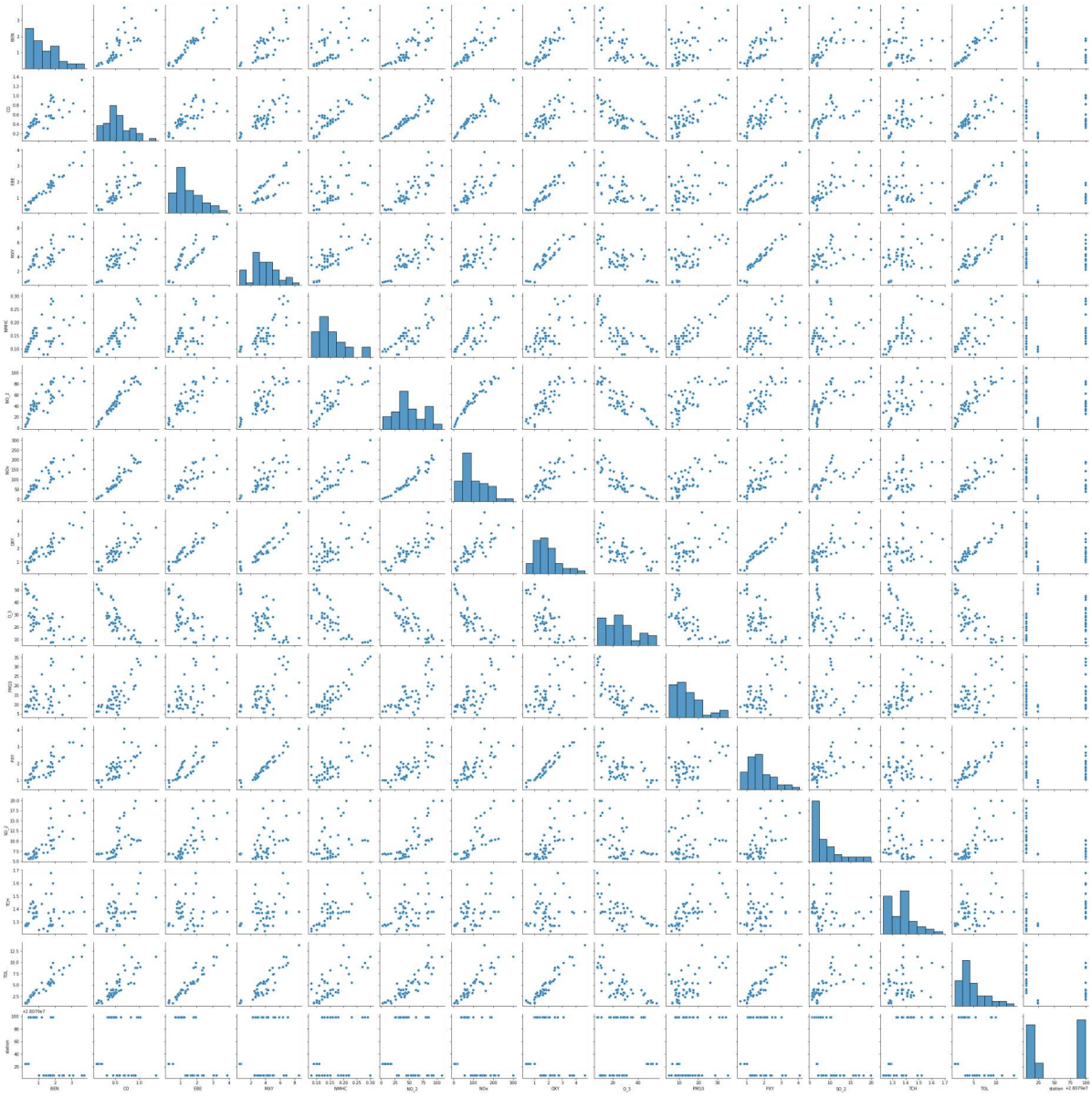
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	200
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	1
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499	
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	1
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	1
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	17

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])
```

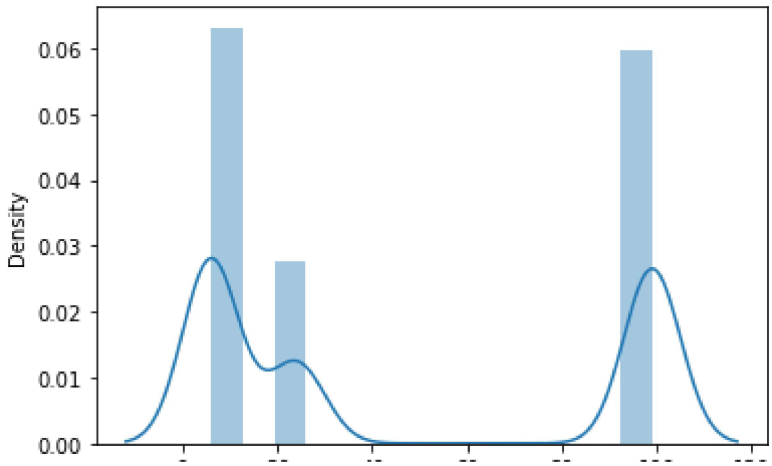
Out[19]: <seaborn.axisgrid.PairGrid at 0x2cdebf13820>



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

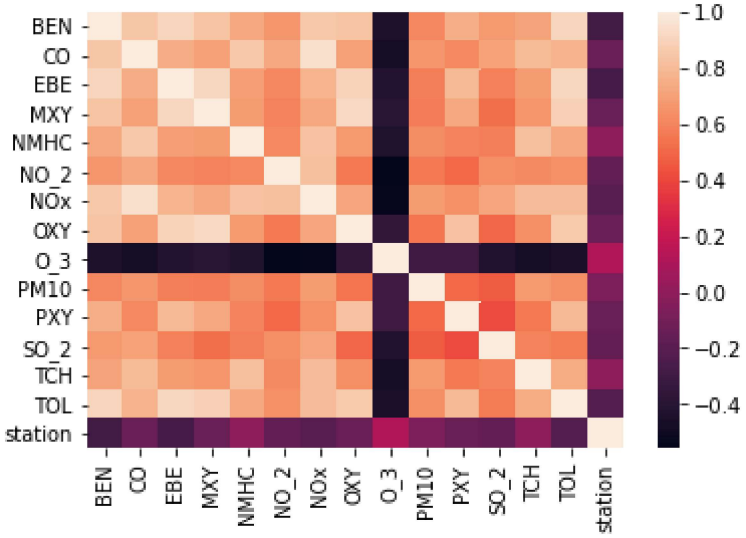
```
future version. Please adapt your code to use either `displot` (a figure-le
vel function with similar flexibility) or `histplot` (an axes-level functio
n for histograms).
  warnings.warn(msg, FutureWarning)
```

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]:
```
28078951.808543332
```

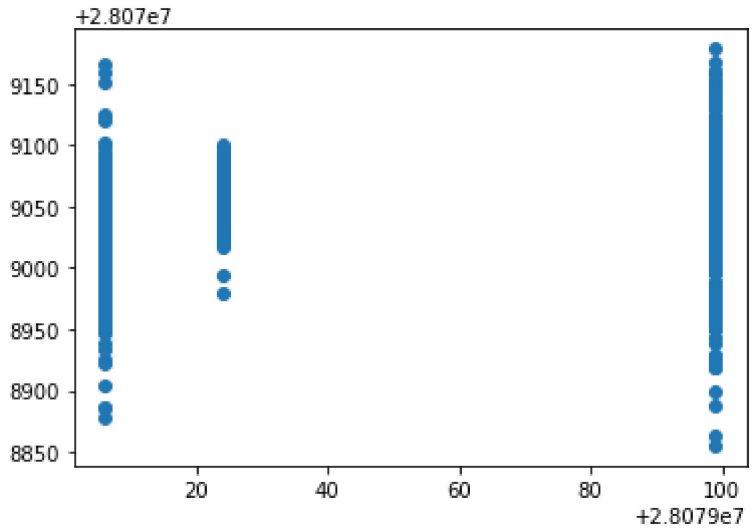
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-9.465611
CO	38.269204
EBE	-13.753835
MXY	3.609269
NMHC	78.684788
NO_2	0.121025
NOx	-0.264759
OXY	3.559233
O_3	0.023548
PM10	0.023321
PXY	2.566572
SO_2	0.185111
TCH	68.698585
TOL	-0.566203

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x2cdfa882250>
```



ACCURACY

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

Out[28]:
```
0.28233362743052093
```

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

Out[29]:
```
0.31282390667177273
```

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]:
```
0.28275844440462194
```

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

Out[33]:
```
0.3125741505664318
```

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]:
```
0.06293246283731035
```

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

Out[36]:
```
0.06685746388029112
```

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([-5.73672840e+00,  1.52481358e+00, -7.59109090e+00,  2.67067424e+00,
        8.76231211e-01, -6.12971883e-02, -2.17032038e-03,  2.00575865e+00,
       -1.84742649e-02,  2.14922961e-01,  1.42783112e+00,  1.27260444e-01,
        1.60778992e+00, -7.90151637e-01])
```

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

Out[39]:
```
28079050.583618924
```

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

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

Out[41]:
```
0.17339846972558737
```

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)))
```

```
36.81280710465069
1534.686303051333
39.175072470275445
```

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]:
```
(20070, 14)
```

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

Out[46]:
```
(20070,)
```

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)
```

```
[28079006]
```

In [52]:
```
logr.classes_
```

Out[52]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

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

Out[53]:
```
0.879023418036871
```

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

Out[54]:
```
0.9998967601812779
```

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

Out[55]:
```
array([[9.99896760e-01, 3.21124597e-30, 1.03239819e-04]])
```

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]:
```
0.8633356706847494
```

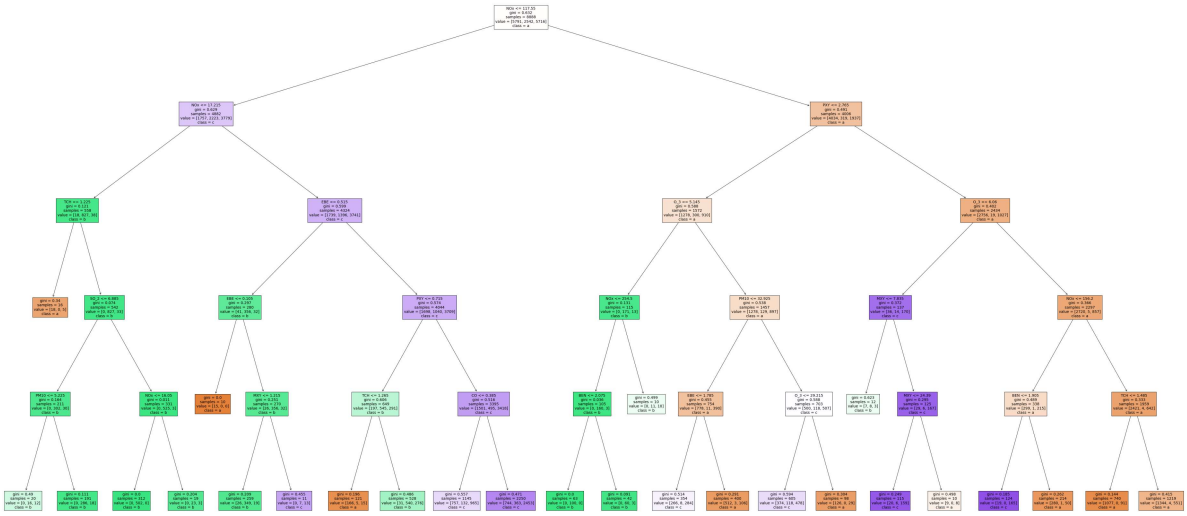
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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(1965.6818181818182, 1993.2, 'NOx <= 117.55\ngini = 0.632\nsamples = 8888\nvalue = [5791, 2542, 5716]\nclass = a'),
 Text(786.2727272727273, 1630.8000000000002, 'NOx <= 17.215\ngini = 0.629\nsamples = 4882\nvalue = [1757, 2223, 3779]\nclass = c'),
 Text(304.3636363636364, 1268.4, 'TCH <= 1.225\ngini = 0.121\nsamples = 558\nvalue = [18, 827, 38]\nclass = b'),
 Text(202.9090909090909, 906.0, 'gini = 0.34\nsamples = 16\nvalue = [18, 0, 5]\nclass = a'),
 Text(405.8181818181818, 906.0, 'SO_2 <= 6.885\ngini = 0.074\nsamples = 542\nvalue = [0, 827, 33]\nclass = b'),
 Text(202.9090909090909, 543.5999999999999, 'PM10 <= 5.225\ngini = 0.164\nsamples = 211\nvalue = [0, 302, 30]\nclass = b'),
 Text(101.45454545454545, 181.19999999999982, 'gini = 0.49\nsamples = 20\nvalue = [0, 16, 12]\nclass = b'),
 Text(304.3636363636364, 181.19999999999982, 'gini = 0.111\nsamples = 191\nvalue = [0, 286, 18]\nclass = b'),
 Text(608.7272727272727, 543.5999999999999, 'NOx <= 16.05\ngini = 0.011\nsamples = 331\nvalue = [0, 525, 3]\nclass = b'),
 Text(507.27272727272725, 181.19999999999982, 'gini = 0.0\nsamples = 312\nvalue = [0, 502, 0]\nclass = b'),
 Text(710.1818181818181, 181.19999999999982, 'gini = 0.204\nsamples = 19\nvalue = [0, 23, 3]\nclass = b'),
 Text(1268.1818181818182, 1268.4, 'EBE <= 0.515\ngini = 0.599\nsamples = 4324\nvalue = [1739, 1396, 3741]\nclass = c'),
 Text(913.0909090909091, 906.0, 'EBE <= 0.105\ngini = 0.297\nsamples = 280\nvalue = [41, 356, 32]\nclass = b'),
 Text(811.6363636363636, 543.5999999999999, 'gini = 0.0\nsamples = 10\nvalue = [15, 0, 0]\nclass = a'),
 Text(1014.5454545454545, 543.5999999999999, 'MXY <= 1.215\ngini = 0.251\nsamples = 270\nvalue = [26, 356, 32]\nclass = b'),
 Text(913.0909090909091, 181.19999999999982, 'gini = 0.209\nsamples = 259\nvalue = [26, 349, 19]\nclass = b'),
 Text(1116.0, 181.19999999999982, 'gini = 0.455\nsamples = 11\nvalue = [0, 7, 13]\nclass = c'),
 Text(1623.2727272727273, 906.0, 'PXY <= 0.715\ngini = 0.574\nsamples = 4044\nvalue = [1698, 1040, 3709]\nclass = c'),
 Text(1420.3636363636363, 543.5999999999999, 'TCH <= 1.265\ngini = 0.606\nsamples = 649\nvalue = [197, 545, 291]\nclass = b'),
 Text(1318.909090909091, 181.19999999999982, 'gini = 0.196\nsamples = 121\nvalue = [166, 5, 15]\nclass = a'),
 Text(1521.8181818181818, 181.19999999999982, 'gini = 0.486\nsamples = 528\nvalue = [31, 540, 276]\nclass = b'),
 Text(1826.1818181818182, 543.5999999999999, 'CO <= 0.385\ngini = 0.516\nsamples = 3395\nvalue = [1501, 495, 3418]\nclass = c'),
 Text(1724.7272727272727, 181.19999999999982, 'gini = 0.557\nsamples = 1145\nvalue = [757, 132, 965]\nclass = c'),
 Text(1927.6363636363635, 181.19999999999982, 'gini = 0.471\nsamples = 2250\nvalue = [744, 363, 2453]\nclass = c'),
 Text(3145.090909090909, 1630.8000000000002, 'PXY <= 2.765\ngini = 0.491\nsamples = 4006\nvalue = [4034, 319, 1937]\nclass = a'),
 Text(2587.090909090909, 1268.4, 'O_3 <= 5.145\ngini = 0.588\nsamples = 1572\nvalue = [1278, 300, 910]\nclass = a'),
 Text(2333.4545454545455, 906.0, 'NOx <= 254.5\ngini = 0.131\nsamples = 115\nvalue = [0, 171, 13]\nclass = b'),
 Text(2232.0, 543.5999999999999, 'BEN <= 2.075\ngini = 0.036\nsamples = 105\nvalue = [0, 160, 3]\nclass = b'),
 Text(2130.5454545454545, 181.19999999999982, 'gini = 0.0\nsamples = 63\nvalu
```

```
e = [0, 100, 0]\nclass = b'),
 Text(2333.4545454545455, 181.19999999999982, 'gini = 0.091\nsamples = 42\nva
lue = [0, 60, 3]\nclass = b'),
 Text(2434.909090909091, 543.5999999999999, 'gini = 0.499\nsamples = 10\nvalu
e = [0, 11, 10]\nclass = b'),
 Text(2840.7272727272725, 906.0, 'PM10 <= 32.925\ngini = 0.538\nsamples = 145
7\nvalue = [1278, 129, 897]\nclass = a'),
 Text(2637.818181818182, 543.5999999999999, 'EBE <= 1.785\ngini = 0.455\nsamp
les = 754\nvalue = [778, 11, 390]\nclass = a'),
 Text(2536.3636363636365, 181.19999999999982, 'gini = 0.514\nsamples = 354\nv
alue = [266, 8, 284]\nclass = c'),
 Text(2739.272727272727, 181.19999999999982, 'gini = 0.291\nsamples = 400\nva
lue = [512, 3, 106]\nclass = a'),
 Text(3043.6363636363635, 543.5999999999999, 'O_3 <= 29.215\ngini = 0.588\nsa
mples = 703\nvalue = [500, 118, 507]\nclass = c'),
 Text(2942.181818181818, 181.19999999999982, 'gini = 0.594\nsamples = 605\nva
lue = [374, 118, 478]\nclass = c'),
 Text(3145.090909090909, 181.19999999999982, 'gini = 0.304\nsamples = 98\nval
ue = [126, 0, 29]\nclass = a'),
 Text(3703.090909090909, 1268.4, 'O_3 <= 6.06\ngini = 0.402\nsamples = 2434\n
value = [2756, 19, 1027]\nclass = a'),
 Text(3348.0, 906.0, 'MXY <= 7.835\ngini = 0.372\nsamples = 137\nvalue = [36,
14, 170]\nclass = c'),
 Text(3246.5454545454545, 543.5999999999999, 'gini = 0.623\nsamples = 12\nval
ue = [7, 8, 3]\nclass = b'),
 Text(3449.4545454545455, 543.5999999999999, 'MXY <= 24.39\ngini = 0.295\nsam
ples = 125\nvalue = [29, 6, 167]\nclass = c'),
 Text(3348.0, 181.19999999999982, 'gini = 0.249\nsamples = 115\nvalue = [20,
6, 159]\nclass = c'),
 Text(3550.909090909091, 181.19999999999982, 'gini = 0.498\nsamples = 10\nval
ue = [9, 0, 8]\nclass = a'),
 Text(4058.181818181818, 906.0, 'NOx <= 156.2\ngini = 0.366\nsamples = 2297\n
value = [2720, 5, 857]\nclass = a'),
 Text(3855.272727272727, 543.5999999999999, 'BEN <= 1.905\ngini = 0.489\nsamp
les = 338\nvalue = [299, 1, 215]\nclass = a'),
 Text(3753.818181818182, 181.19999999999982, 'gini = 0.185\nsamples = 124\nva
lue = [19, 0, 165]\nclass = c'),
 Text(3956.7272727272725, 181.19999999999982, 'gini = 0.262\nsamples = 214\nv
alue = [280, 1, 50]\nclass = a'),
 Text(4261.090909090909, 543.5999999999999, 'TCH <= 1.485\ngini = 0.333\nsamp
les = 1959\nvalue = [2421, 4, 642]\nclass = a'),
 Text(4159.636363636364, 181.19999999999982, 'gini = 0.144\nsamples = 740\nva
lue = [1077, 0, 91]\nclass = a'),
 Text(4362.545454545454, 181.19999999999982, 'gini = 0.415\nsamples = 1219\nv
alue = [1344, 4, 551]\nclass = a')]
```



Accuracy

Linear Regression

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

Out[63]: 0.31282390667177273

Ridge Regression

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

Out[64]: 0.3125741505664318

Lasso Regression

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

Out[65]: 0.06685746388029112

ElasticNet Regression

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

Out[66]: 0.17339846972558737

Logistic Regression

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

Out[67]:
```
0.879023418036871
```

Random Forest

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

Out[68]:
```
0.8633356706847494
```

Conclusion

Logistic Regression is suitable for this dataset