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_2011.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
0	2011-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN	280790
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7	280790
2	2011-11-01 01:00:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2	280790
3	2011-11-01 01:00:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN	280790
4	2011-11-01 01:00:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN	280790
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209923	2011-09-01 00:00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN	280790
209924	2011-09-01 00:00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN	280790
209925	2011-09-01 00:00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN	280790

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
209926	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN	280790
209927	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN	280790

209928 rows × 14 columns

Data Cleaning and Data Preprocessing

In [3]:
```
df=df.dropna()
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[6]:

	CO	station
1	0.4	28079008
6	0.3	28079024

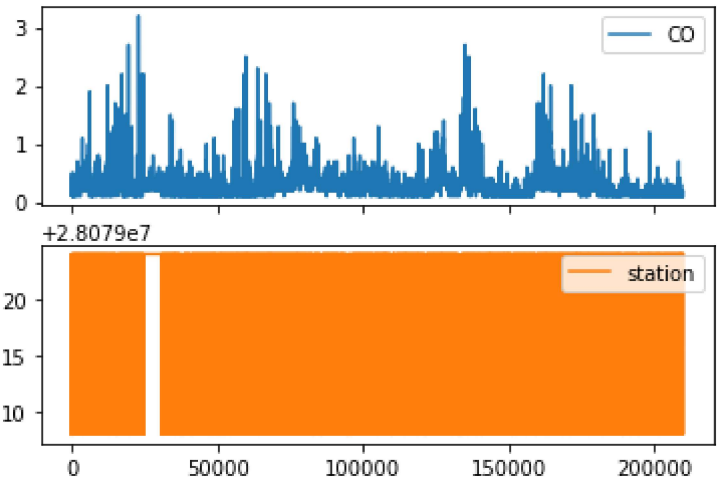
	CO	station
25	0.3	28079008
30	0.4	28079024
49	0.2	28079008
...	...	...
209862	0.1	28079024
209881	0.1	28079008
209886	0.1	28079024
209905	0.1	28079008
209910	0.1	28079024

16460 rows × 2 columns

Line chart

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

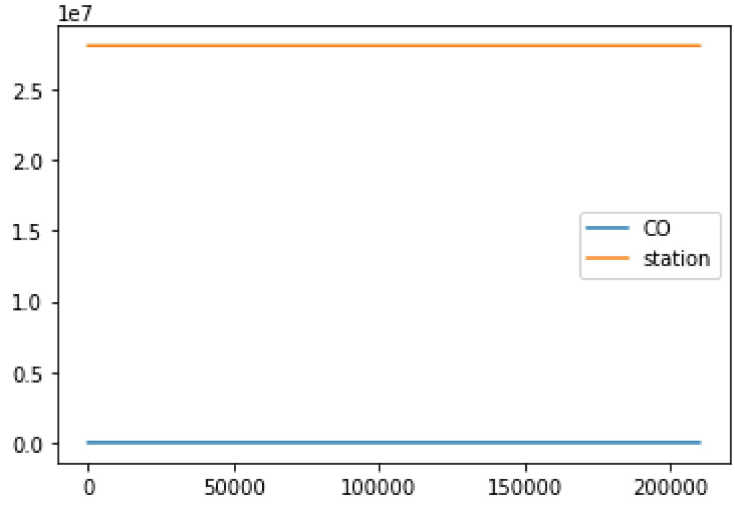
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

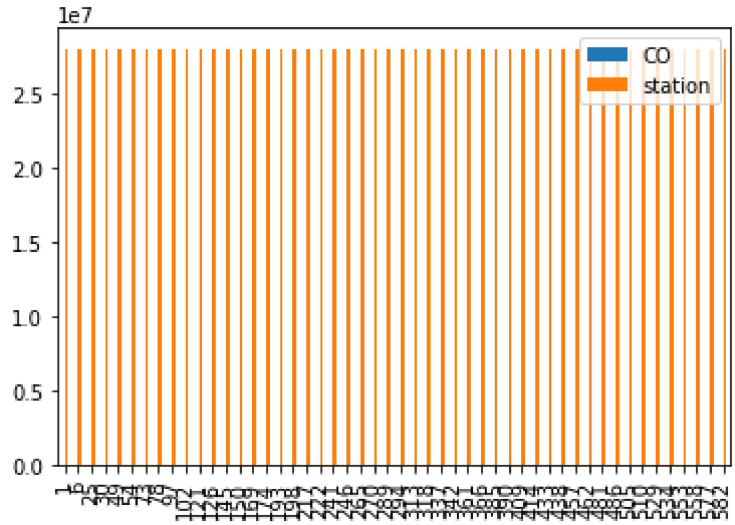


Bar chart

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

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

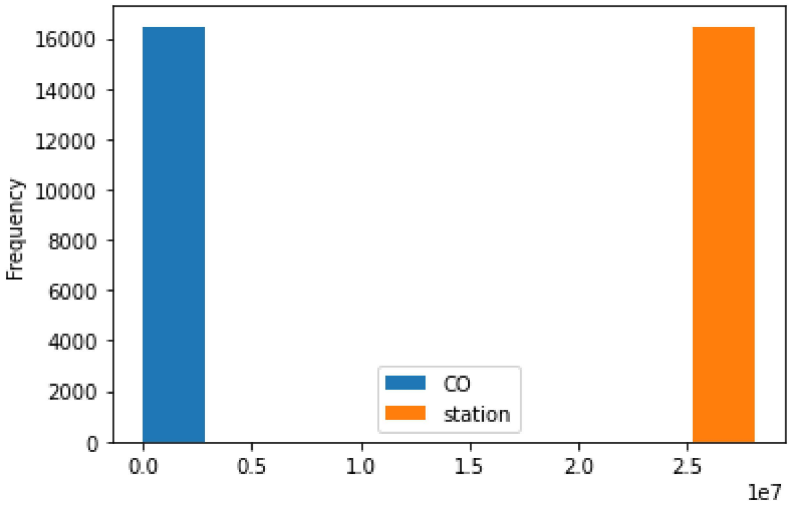
Out[10]: <AxesSubplot:>



Histogram

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

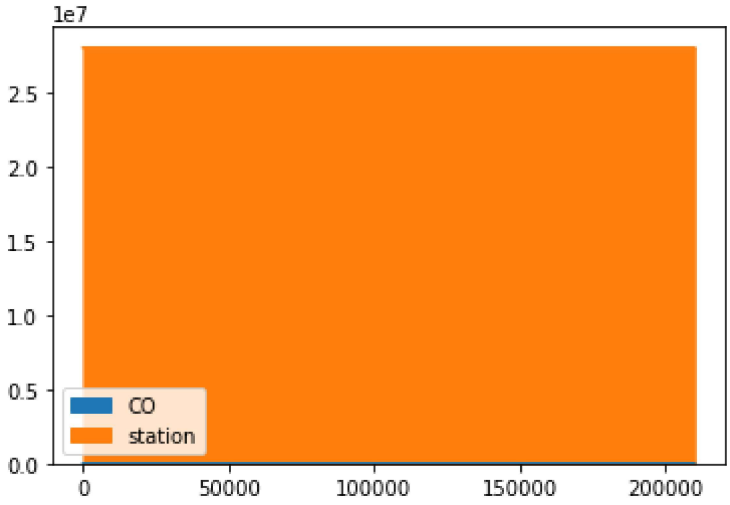
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

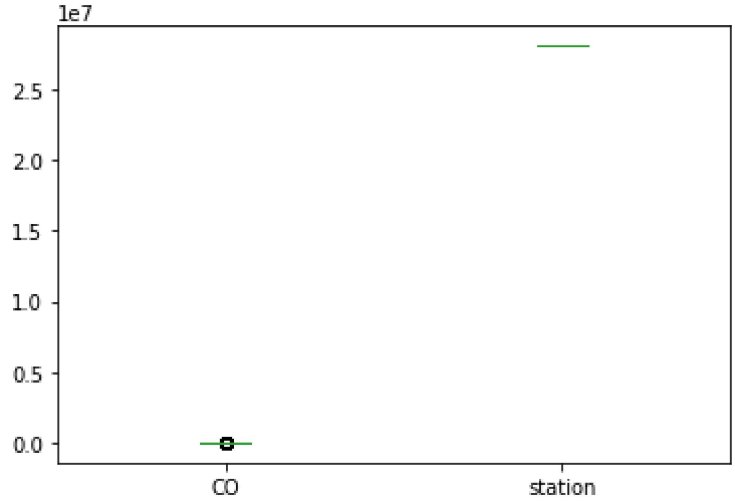
Out[12]: <AxesSubplot:>



Box chart

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

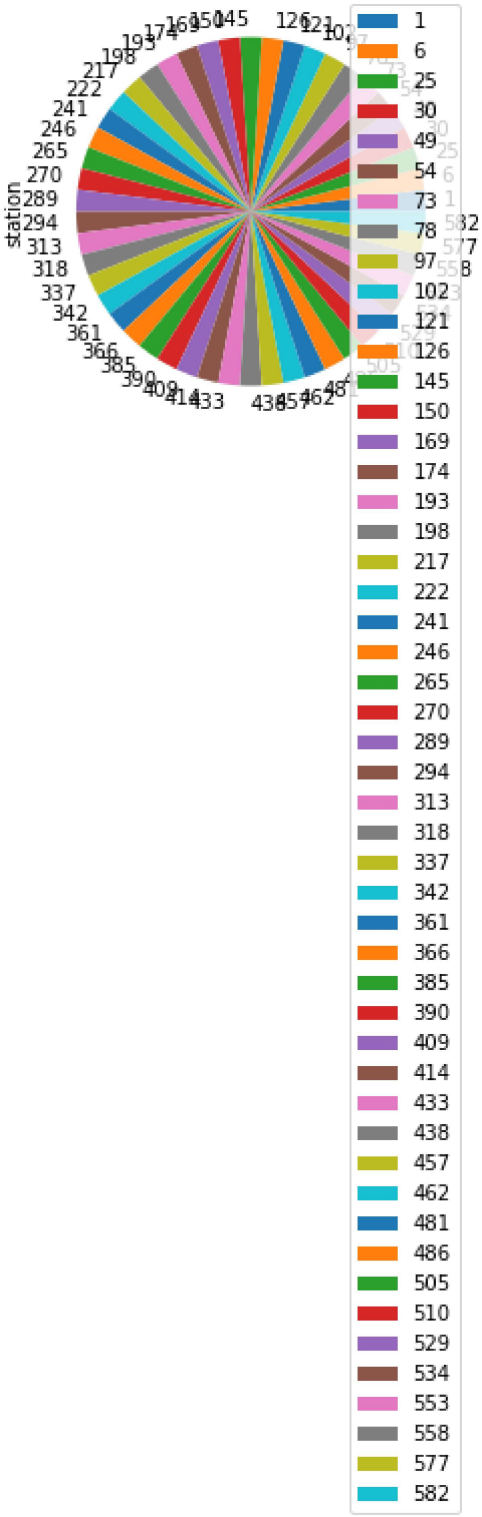
Out[13]: <AxesSubplot:>



Pie chart

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

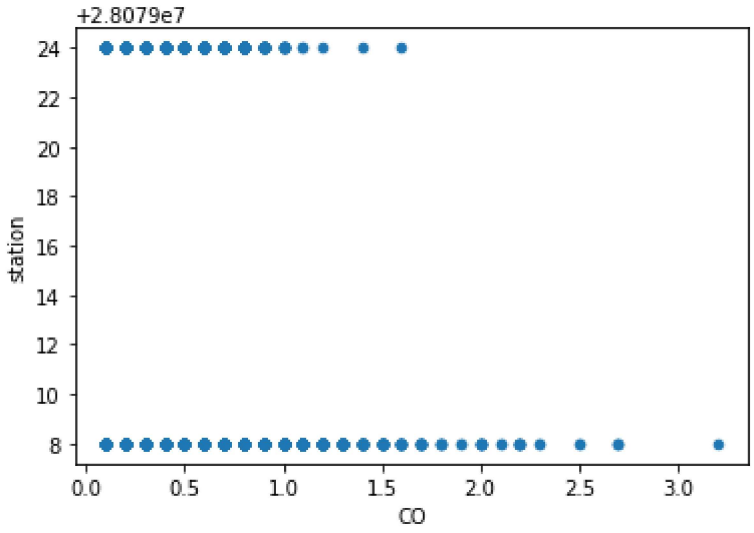
Out[14]: <AxesSubplot:ylabel='station'>



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: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16460 non-null  object 
 1   BEN      16460 non-null  float64
 2   CO       16460 non-null  float64
 3   EBE      16460 non-null  float64
 4   NMHC     16460 non-null  float64
 5   NO       16460 non-null  float64
 6   NO_2     16460 non-null  float64
 7   O_3      16460 non-null  float64
 8   PM10     16460 non-null  float64
 9   PM25     16460 non-null  float64
 10  SO_2     16460 non-null  float64
 11  TCH      16460 non-null  float64
 12  TOL      16460 non-null  float64
 13  station  16460 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[17]:

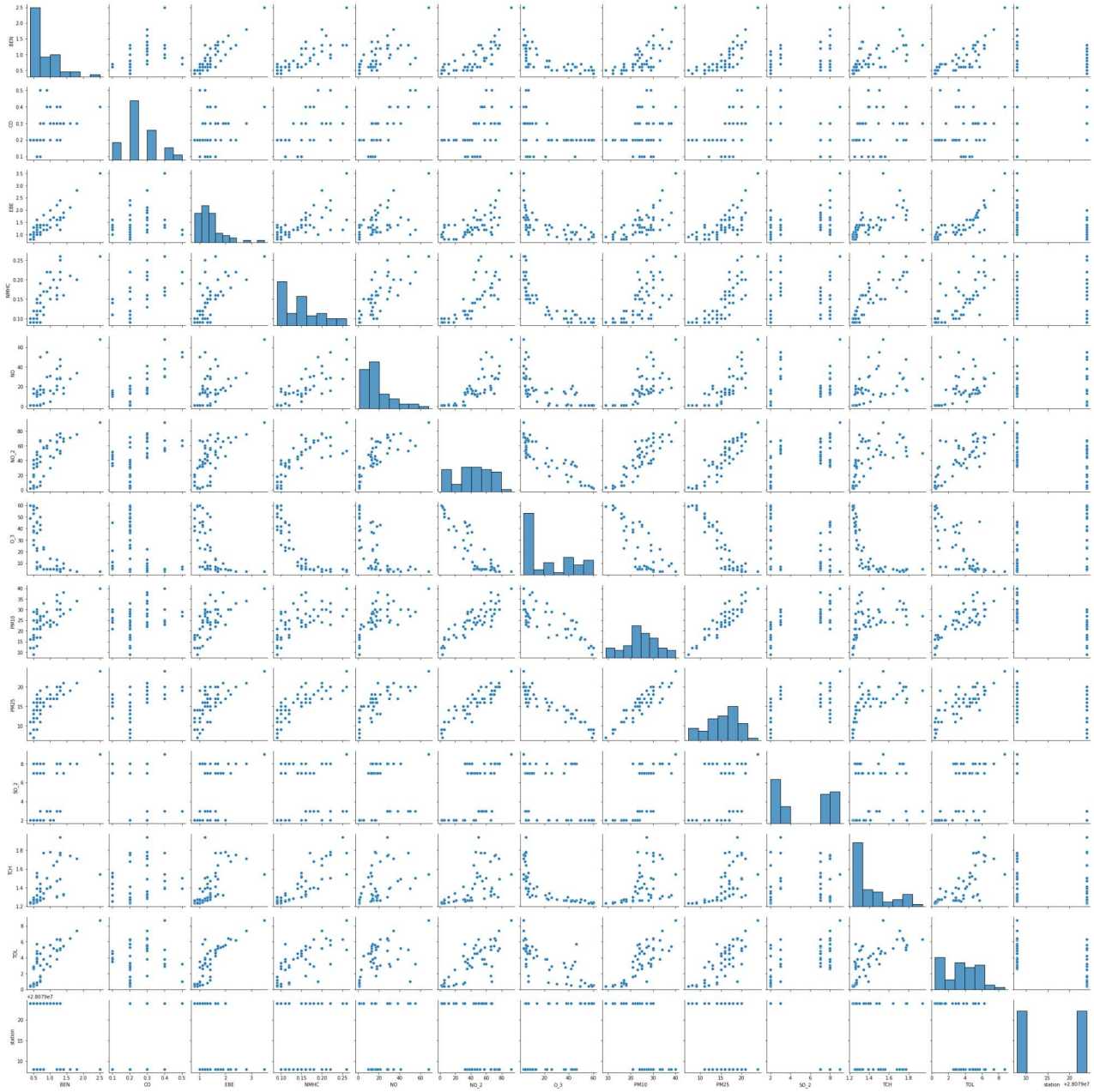
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000
mean	0.900680	0.277758	1.471871	0.167043	23.671810	44.583961	41.580377
std	0.768892	0.206143	1.051004	0.075068	44.362859	31.569185	28.113385
min	0.100000	0.100000	0.200000	0.010000	1.000000	1.000000	1.000000
25%	0.500000	0.200000	0.800000	0.120000	2.000000	19.000000	17.000000
50%	0.700000	0.200000	1.200000	0.160000	7.000000	40.000000	39.000000
75%	1.100000	0.300000	1.700000	0.200000	25.000000	63.000000	61.000000

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
max	9.500000	3.200000	12.800000	0.840000	615.000000	289.000000	154.00000

EDA AND VISUALIZATION

In [18]:
```
sns.pairplot(df[0:50])
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x23f15525940>

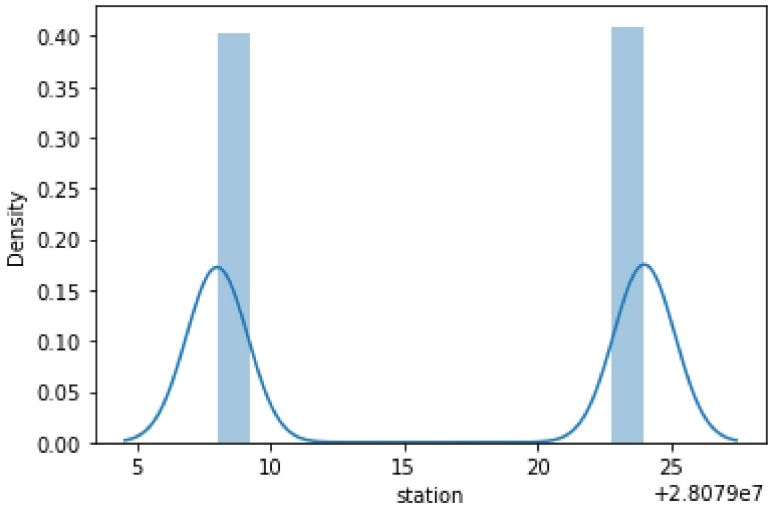


In [19]:
```
sns.distplot(df['station'])
```

```
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
```

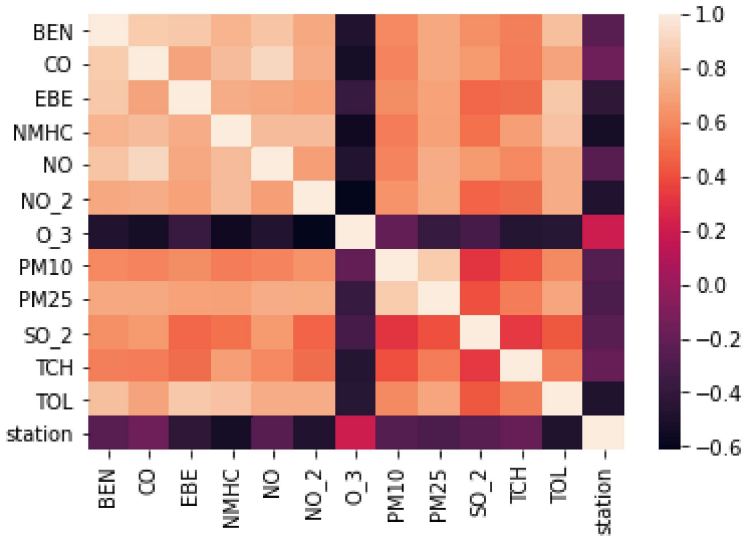
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [20]:
```
sns.heatmap(df.corr())
```

Out[20]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [21]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [22]:
```
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 [23]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[23]:
```
LinearRegression()
```

In [24]:
```
lr.intercept_
```

Out[24]:
```
28079015.14760062
```

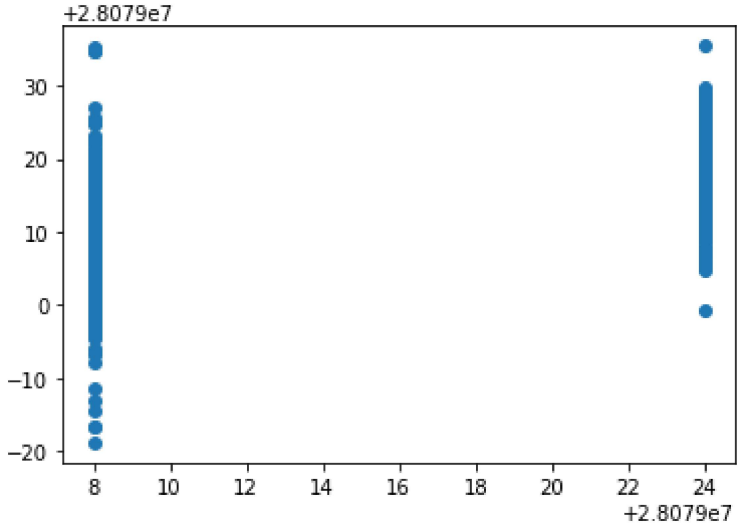
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	3.450067
CO	39.586966
EBE	-1.867066
NMHC	-93.239614
NO	-0.039815
NO_2	-0.096005
O_3	-0.017379
PM10	0.026322
PM25	-0.057732
SO_2	-0.450617
TCH	11.044752
TOL	-0.366356

In [26]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x23f2047dd00>
```



ACCURACY

In [27]:
```
lr.score(x_test,y_test)
```

Out[27]: 0.6255021013908312

In [28]:
```
lr.score(x_train,y_train)
```

Out[28]: 0.6276704921640321

Ridge and Lasso

In [29]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [30]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[30]: Ridge(alpha=10)

Accuracy(Ridge)

In [31]:
```
rr.score(x_test,y_test)
```

Out[31]: 0.5949135360319011

In [32]:
```
rr.score(x_train,y_train)
```

Out[32]: 0.5938147442105148

In [33]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[33]:
```
Lasso(alpha=10)
```

In [34]:
```
la.score(x_train,y_train)
```

Out[34]:
```
0.23481826129809824
```

Accuracy(Lasso)

In [35]:
```
la.score(x_test,y_test)
```

Out[35]:
```
0.24416009527439764
```

In [36]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[36]:
```
ElasticNet()
```

In [37]:
```
en.coef_
```

Out[37]:
```
array([ 0.28551945,  0.        , -0.        , -0.        ,  0.05091213,
       -0.1349548 , -0.04404403,  0.03035085,  0.07781285, -0.17479693,
        0.        , -0.97139213])
```

In [38]:
```
en.intercept_
```

Out[38]:
```
28079025.16102611
```

In [39]:
```
prediction=en.predict(x_test)
```

In [40]:
```
en.score(x_test,y_test)
```

Out[40]:
```
0.35934636599484293
```

Evaluation Metrics

In [41]:
```
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)))
```

```
5.599206095179652
40.978419142550166
6.4014388337740264
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [44]:
```
feature_matrix.shape
```

Out[44]: (16460, 12)

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

Out[45]: (16460,)

In [46]:
```
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [48]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[48]: LogisticRegression(max_iter=10000)

In [49]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12]]
```

In [50]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079008]
```

In [51]:
```
logr.classes_
```

Out[51]: array([28079008, 28079024], dtype=int64)

In [52]:
```
logr.score(fs,target_vector)
```

Out[52]: 0.9262454434993924

In [53]:
```
logr.predict_proba(observation)[0][0]
```

Out[53]:
```
0.9999999999999999
```

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

Out[54]:
```
array([[1.00000000e+00, 9.78522268e-17]])
```

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[56]:
```
RandomForestClassifier()
```

In [57]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]
}
```

In [58]:
```
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[58]:
```
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 [59]:
```
grid_search.best_score_
```

Out[59]:
```
0.9352542961291442
```

In [60]:
```
rfc_best=grid_search.best_estimator_
```

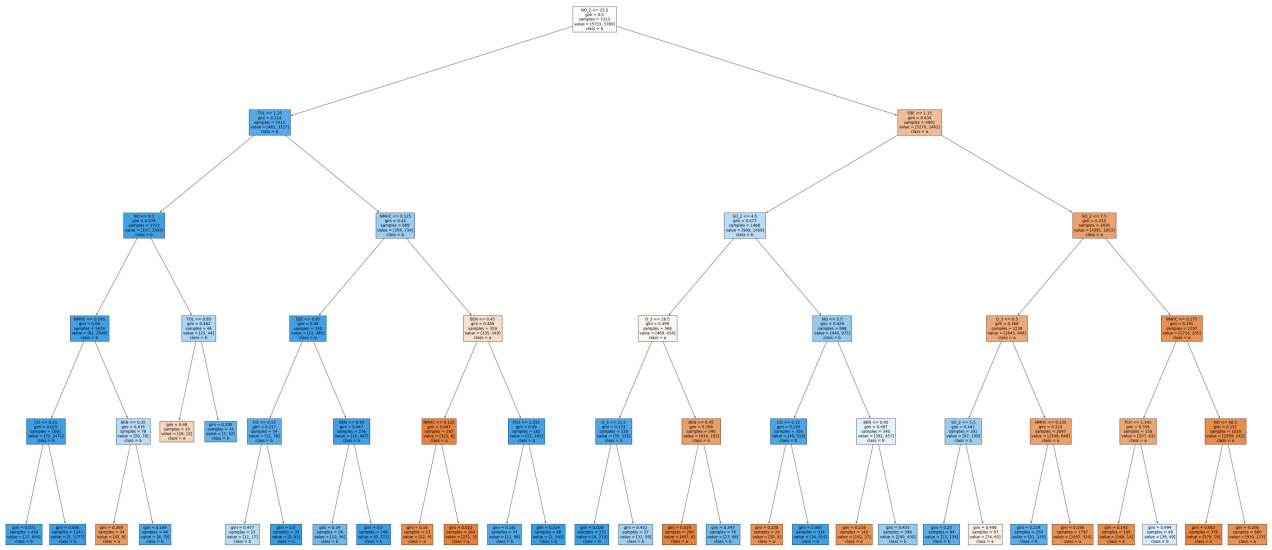
In [61]:
```
from sklearn.tree import plot_tree

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

Out[61]:
```
[Text(2087.689655172414, 1993.2, 'NO_2 <= 25.5\ngini = 0.5\nsamples = 7313\nvalue = [573
3, 5789]\nclass = b'),
 Text(942.8275862068966, 1630.8000000000002, 'TOL <= 1.25\ngini = 0.214\nsamples = 2412
\nvalue = [463, 3327]\nclass = b'),
```

```
 Text(500.2758620689656, 1268.4, 'NO <= 6.5\ngini = 0.076\nsamples = 1723\nvalue = [107,
2593]\nclass = b'),
 Text(307.86206896551727, 906.0, 'NMHC <= 0.145\ngini = 0.06\nsamples = 1679\nvalue = [8
2, 2549]\nclass = b'),
 Text(153.93103448275863, 543.5999999999999, 'CO <= 0.15\ngini = 0.025\nsamples = 1601\n
value = [32, 2471]\nclass = b'),
 Text(76.96551724137932, 181.19999999999982, 'gini = 0.072\nsamples = 458\nvalue = [27,
694]\nclass = b'),
 Text(230.89655172413796, 181.19999999999982, 'gini = 0.006\nsamples = 1143\nvalue = [5,
1777]\nclass = b'),
 Text(461.79310344827593, 543.5999999999999, 'BEN <= 0.35\ngini = 0.476\nsamples = 78\nv
alue = [50, 78]\nclass = b'),
 Text(384.82758620689657, 181.19999999999982, 'gini = 0.269\nsamples = 34\nvalue = [42,
8]\nclass = a'),
 Text(538.7586206896552, 181.19999999999982, 'gini = 0.184\nsamples = 44\nvalue = [8, 7
0]\nclass = b'),
 Text(692.6896551724138, 906.0, 'TOL <= 0.85\ngini = 0.462\nsamples = 44\nvalue = [25, 4
4]\nclass = b'),
 Text(615.7241379310345, 543.5999999999999, 'gini = 0.48\nsamples = 20\nvalue = [18, 12]
\nclass = a'),
 Text(769.6551724137931, 543.5999999999999, 'gini = 0.295\nsamples = 24\nvalue = [7, 32]
\nclass = b'),
 Text(1385.3793103448277, 1268.4, 'NMHC <= 0.125\ngini = 0.44\nsamples = 689\nvalue = [3
56, 734]\nclass = b'),
 Text(1077.5172413793105, 906.0, 'EBE <= 0.85\ngini = 0.08\nsamples = 330\nvalue = [21,
485]\nclass = b'),
 Text(923.5862068965519, 543.5999999999999, 'CO <= 0.15\ngini = 0.217\nsamples = 54\nval
ue = [11, 78]\nclass = b'),
 Text(846.6206896551724, 181.19999999999982, 'gini = 0.477\nsamples = 15\nvalue = [11, 1
7]\nclass = b'),
 Text(1000.5517241379312, 181.19999999999982, 'gini = 0.0\nsamples = 39\nvalue = [0, 61]
\nclass = b'),
 Text(1231.448275862069, 543.5999999999999, 'BEN <= 0.45\ngini = 0.047\nsamples = 276\nv
alue = [10, 407]\nclass = b'),
 Text(1154.4827586206898, 181.19999999999982, 'gini = 0.34\nsamples = 28\nvalue = [10, 3
6]\nclass = b'),
 Text(1308.4137931034484, 181.19999999999982, 'gini = 0.0\nsamples = 248\nvalue = [0, 37
1]\nclass = b'),
 Text(1693.2413793103449, 906.0, 'BEN <= 0.45\ngini = 0.489\nsamples = 359\nvalue = [33
5, 249]\nclass = a'),
 Text(1539.3103448275863, 543.5999999999999, 'NMHC <= 0.135\ngini = 0.047\nsamples = 197
\nvalue = [323, 8]\nclass = a'),
 Text(1462.344827586207, 181.19999999999982, 'gini = 0.16\nsamples = 33\nvalue = [52, 5]
\nclass = a'),
 Text(1616.2758620689656, 181.19999999999982, 'gini = 0.022\nsamples = 164\nvalue = [27
1, 3]\nclass = a'),
 Text(1847.1724137931037, 543.5999999999999, 'TCH <= 1.335\ngini = 0.09\nsamples = 162\n
value = [12, 241]\nclass = b'),
 Text(1770.2068965517242, 181.19999999999982, 'gini = 0.181\nsamples = 74\nvalue = [11,
98]\nclass = b'),
 Text(1924.137931034483, 181.19999999999982, 'gini = 0.014\nsamples = 88\nvalue = [1, 14
3]\nclass = b'),
 Text(3232.551724137931, 1630.8000000000002, 'EBE <= 1.15\ngini = 0.434\nsamples = 4901
\nvalue = [5270, 2462]\nclass = a'),
 Text(2616.8275862068967, 1268.4, 'SO_2 <= 4.5\ngini = 0.477\nsamples = 1466\nvalue = [9
09, 1409]\nclass = b'),
 Text(2308.9655172413795, 906.0, 'O_3 <= 28.5\ngini = 0.499\nsamples = 568\nvalue = [46
9, 434]\nclass = a'),
 Text(2155.034482758621, 543.5999999999999, 'O_3 <= 21.5\ngini = 0.173\nsamples = 228\nv
alue = [35, 332]\nclass = b'),
 Text(2078.0689655172414, 181.19999999999982, 'gini = 0.028\nsamples = 171\nvalue = [4,
273]\nclass = b'),
 Text(2232.0, 181.19999999999982, 'gini = 0.452\nsamples = 57\nvalue = [31, 59]\nclass =
b'),
 Text(2462.896551724138, 543.5999999999999, 'BEN <= 0.45\ngini = 0.308\nsamples = 340\nv
```

```
alue = [434, 102]\nclass = a'),
 Text(2385.9310344827586, 181.19999999999982, 'gini = 0.029\nsamples = 264\nvalue = [40
7, 6]\nclass = a'),
 Text(2539.8620689655177, 181.19999999999982, 'gini = 0.343\nsamples = 76\nvalue = [27,
96]\nclass = b'),
 Text(2924.689655172414, 906.0, 'NO <= 5.5\ngini = 0.429\nsamples = 898\nvalue = [440, 9
75]\nclass = b'),
 Text(2770.7586206896553, 543.5999999999999, 'CO <= 0.15\ngini = 0.155\nsamples = 358\nv
alue = [48, 518]\nclass = b'),
 Text(2693.7931034482763, 181.19999999999982, 'gini = 0.208\nsamples = 24\nvalue = [30,
4]\nclass = a'),
 Text(2847.724137931035, 181.19999999999982, 'gini = 0.065\nsamples = 334\nvalue = [18,
514]\nclass = b'),
 Text(3078.6206896551726, 543.5999999999999, 'BEN <= 0.45\ngini = 0.497\nsamples = 540\n
value = [392, 457]\nclass = b'),
 Text(3001.6551724137935, 181.19999999999982, 'gini = 0.216\nsamples = 142\nvalue = [19
2, 27]\nclass = a'),
 Text(3155.586206896552, 181.19999999999982, 'gini = 0.433\nsamples = 398\nvalue = [200,
430]\nclass = b'),
 Text(3848.275862068966, 1268.4, 'SO_2 <= 7.5\ngini = 0.313\nsamples = 3435\nvalue = [43
61, 1053]\nclass = a'),
 Text(3540.4137931034484, 906.0, 'O_3 <= 6.5\ngini = 0.368\nsamples = 2238\nvalue = [264
5, 848]\nclass = a'),
 Text(3386.4827586206898, 543.5999999999999, 'SO_2 <= 5.5\ngini = 0.441\nsamples = 191\n
value = [97, 199]\nclass = b'),
 Text(3309.5172413793107, 181.19999999999982, 'gini = 0.25\nsamples = 94\nvalue = [23, 1
34]\nclass = b'),
 Text(3463.4482758620693, 181.19999999999982, 'gini = 0.498\nsamples = 97\nvalue = [74,
65]\nclass = a'),
 Text(3694.3448275862074, 543.5999999999999, 'NMHC <= 0.135\ngini = 0.324\nsamples = 204
7\nvalue = [2548, 649]\nclass = a'),
 Text(3617.379310344828, 181.19999999999982, 'gini = 0.229\nsamples = 250\nvalue = [51,
335]\nclass = b'),
 Text(3771.3103448275865, 181.19999999999982, 'gini = 0.198\nsamples = 1797\nvalue = [24
97, 314]\nclass = a'),
 Text(4156.137931034483, 906.0, 'NMHC <= 0.175\ngini = 0.191\nsamples = 1197\nvalue = [1
716, 205]\nclass = a'),
 Text(4002.2068965517246, 543.5999999999999, 'TCH <= 1.345\ngini = 0.358\nsamples = 158
\nvalue = [207, 63]\nclass = a'),
 Text(3925.241379310345, 181.19999999999982, 'gini = 0.142\nsamples = 109\nvalue = [168,
14]\nclass = a'),
 Text(4079.1724137931037, 181.19999999999982, 'gini = 0.494\nsamples = 49\nvalue = [39,
49]\nclass = b'),
 Text(4310.068965517242, 543.5999999999999, 'NO <= 48.5\ngini = 0.157\nsamples = 1039\nv
alue = [1509, 142]\nclass = a'),
 Text(4233.103448275862, 181.19999999999982, 'gini = 0.062\nsamples = 379\nvalue = [579,
19]\nclass = a'),
 Text(4387.034482758621, 181.19999999999982, 'gini = 0.206\nsamples = 660\nvalue = [930,
123]\nclass = a')]
```



Conclusion

Accuracy

In [62]:
```
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: 0.6255021013908312
Ridge Regression: 0.5949135360319011
Lasso Regression 0.24416009527439764
ElasticNet Regression: 0.35934636599484293
Logistic Regression: 0.9262454434993924
Random Forest: 0.9352542961291442
```

Random Forest is suitable for this dataset