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_2015.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
0	2015-10-01 01:00:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN	2807900
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3	2807900
2	2015-10-01 01:00:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1	2807901
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN	2807901
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN	2807901
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210091	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN	2807905
210092	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN	2807905
210093	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN	2807905

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
210094	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN	2807905
210095	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN	2807906

210096 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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

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

Out[6]:

	CO	station
1	0.8	28079008
6	0.3	28079024

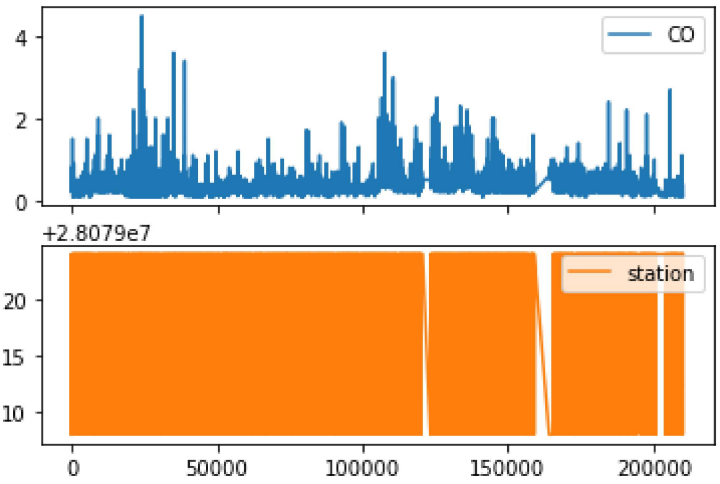
	CO	station
25	0.7	28079008
30	0.3	28079024
49	0.8	28079008
...	...	...
210030	0.1	28079024
210049	0.3	28079008
210054	0.1	28079024
210073	0.3	28079008
210078	0.1	28079024

16026 rows × 2 columns

Line chart

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

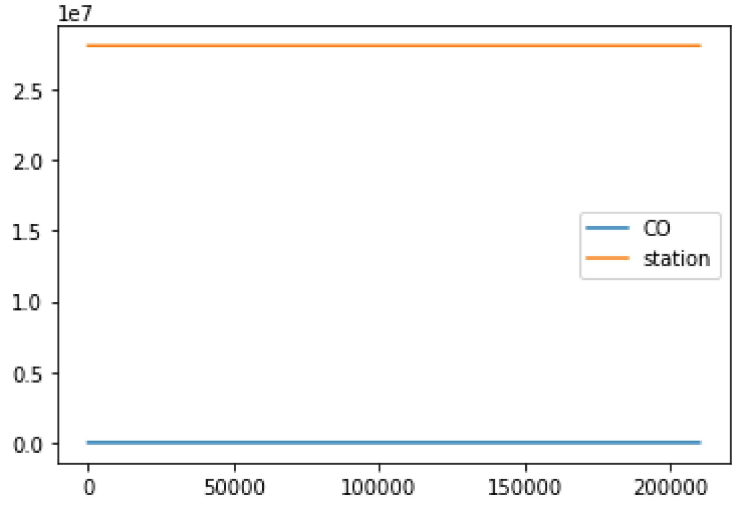
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

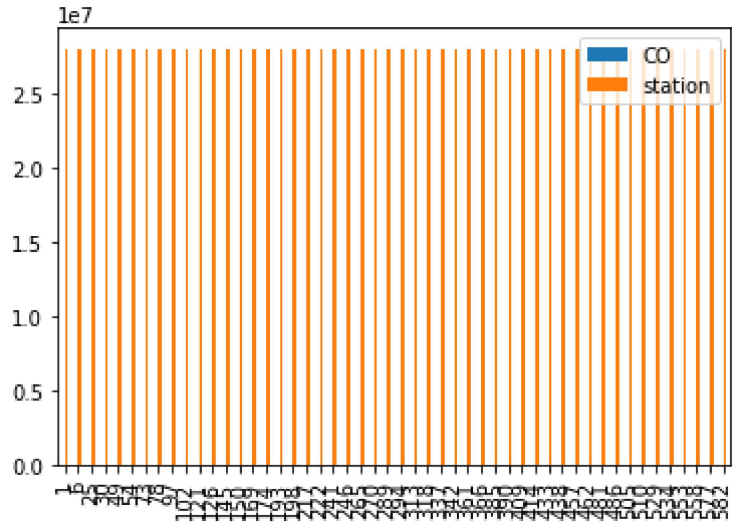


Bar chart

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

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

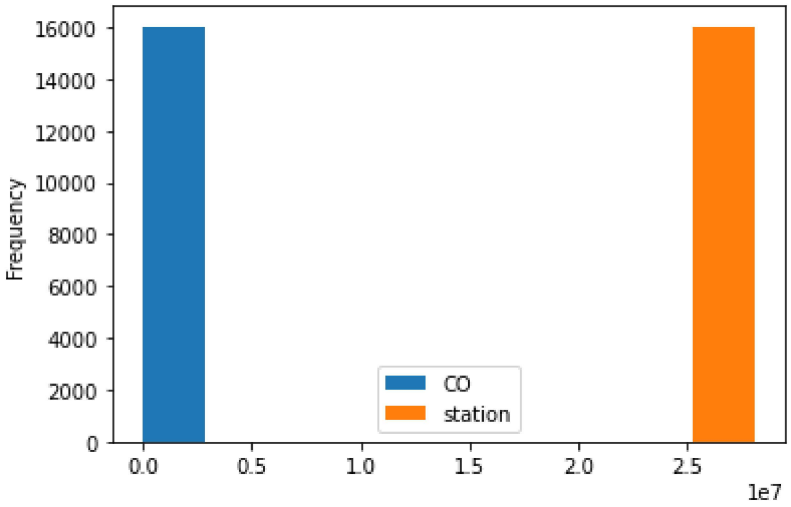
Out[10]: <AxesSubplot:>



Histogram

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

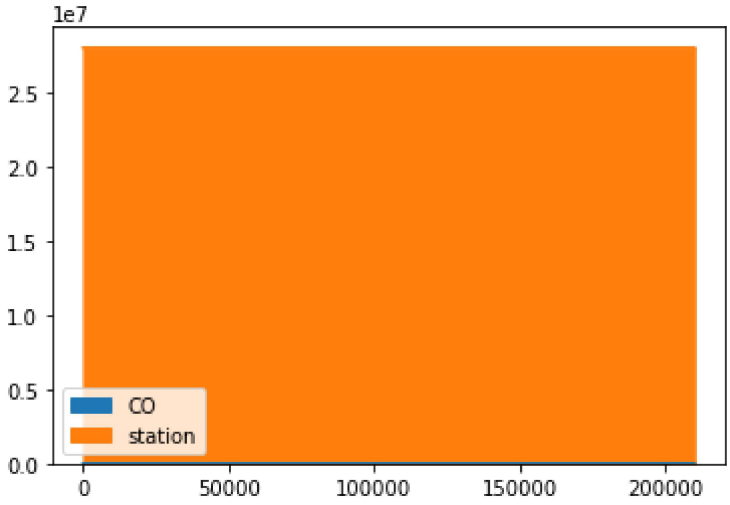
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

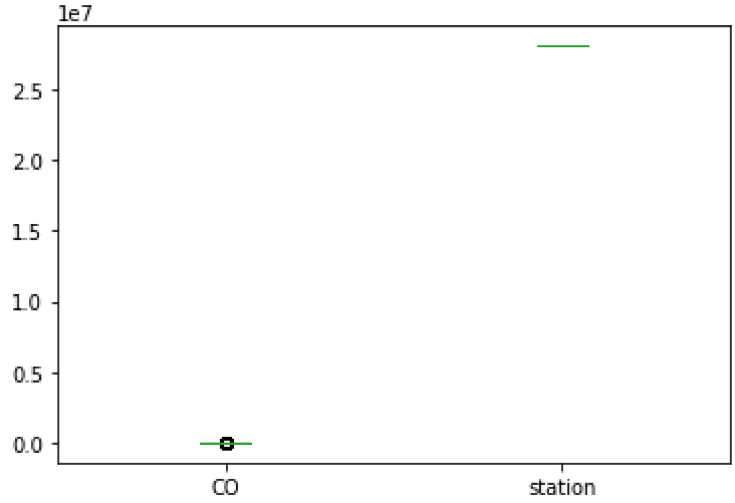
Out[12]: <AxesSubplot:>



Box chart

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

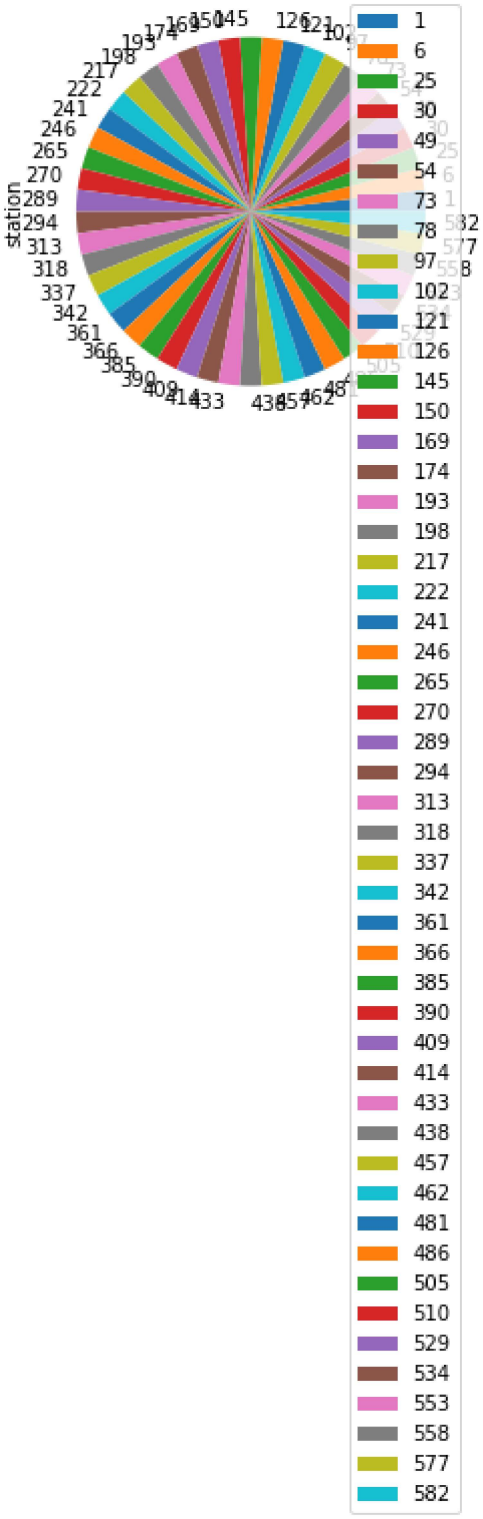
Out[13]: <AxesSubplot:>



Pie chart

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

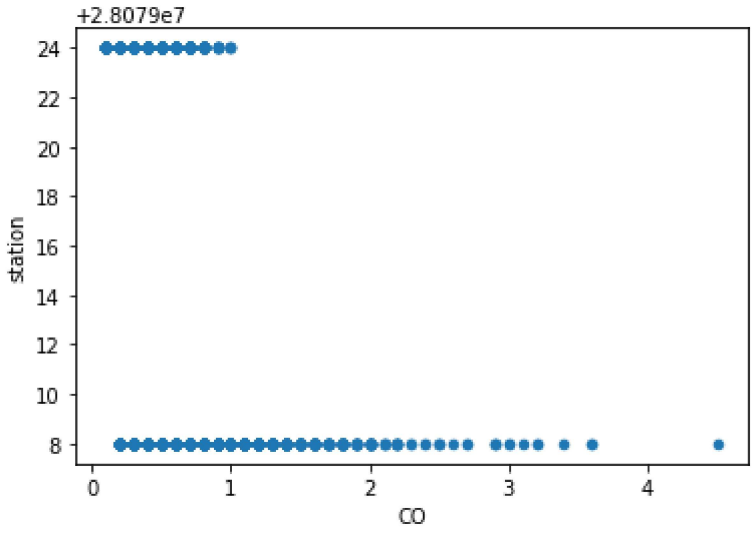
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16026 non-null  object 
 1   BEN      16026 non-null  float64
 2   CO       16026 non-null  float64
 3   EBE      16026 non-null  float64
 4   NMHC     16026 non-null  float64
 5   NO       16026 non-null  float64
 6   NO_2     16026 non-null  float64
 7   O_3      16026 non-null  float64
 8   PM10     16026 non-null  float64
 9   PM25     16026 non-null  float64
 10  SO_2     16026 non-null  float64
 11  TCH      16026 non-null  float64
 12  TOL      16026 non-null  float64
 13  station  16026 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

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

Out[17]:

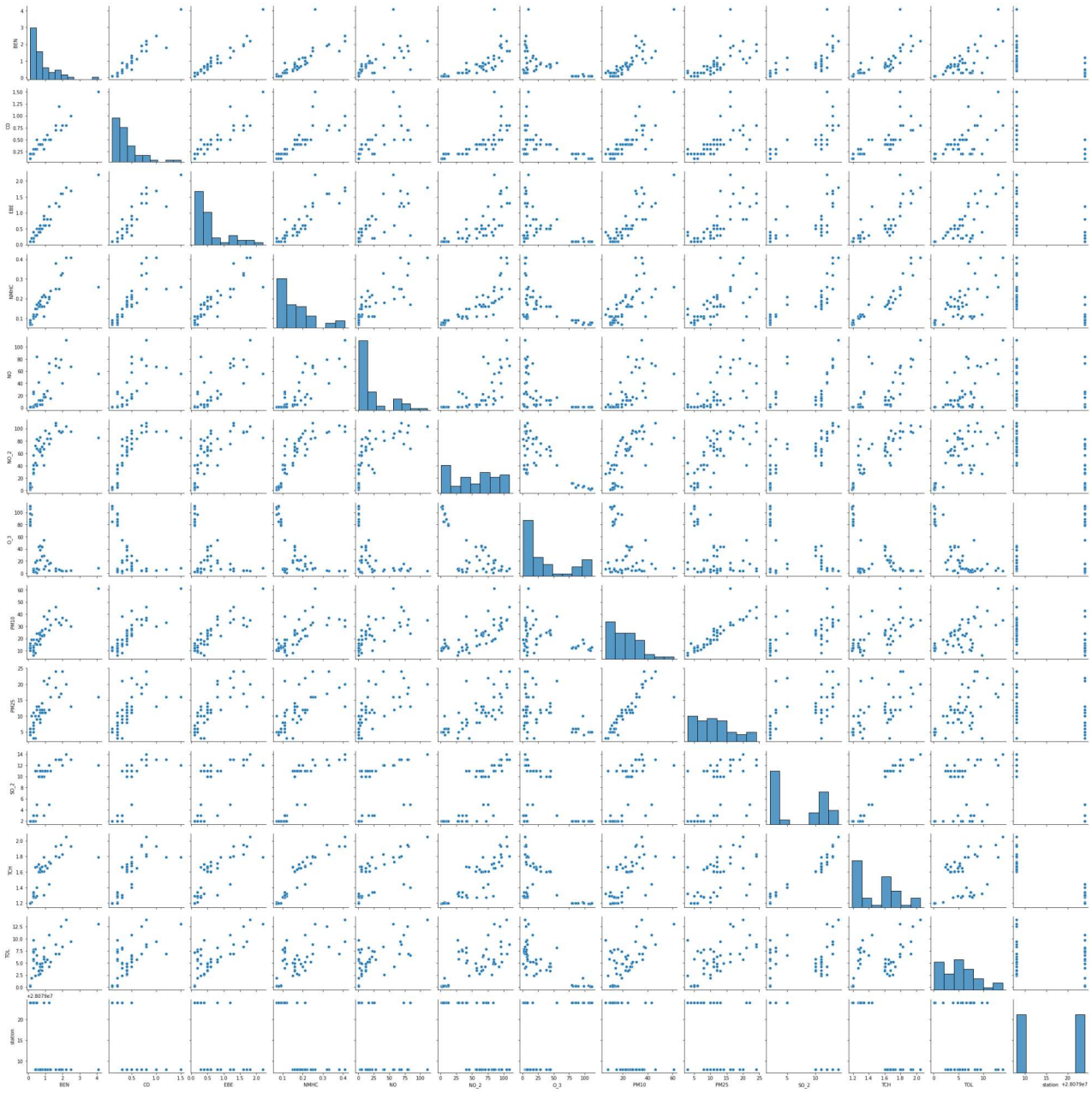
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	48.089793
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	35.847298
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	15.000000
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	46.000000
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	73.000000

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	217.00000

EDA AND VISUALIZATION

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

Out[18]: <seaborn.axisgrid.PairGrid at 0x1d50d04fb80>

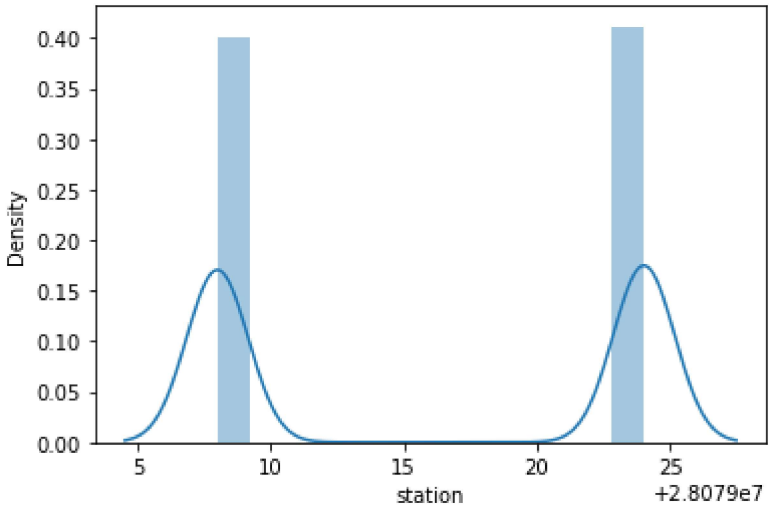


In [19]:
```
sns.distplot(df['station'])
```

```
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
```

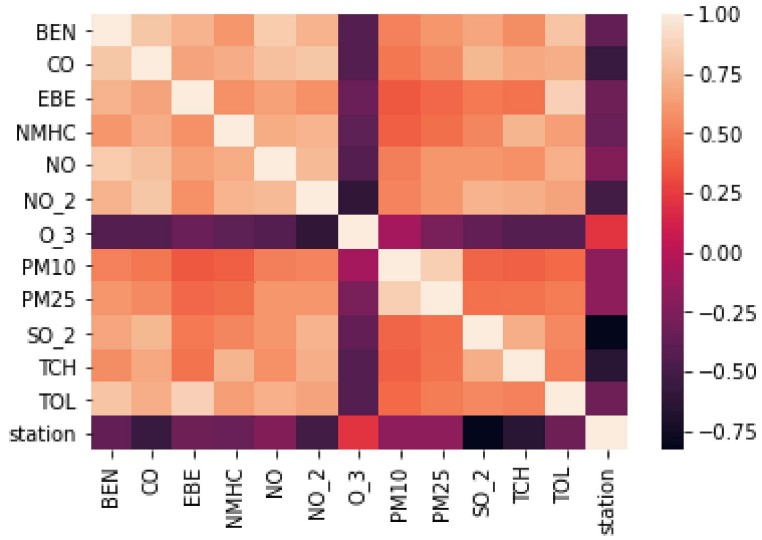
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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]:
```
28079039.314071946
```

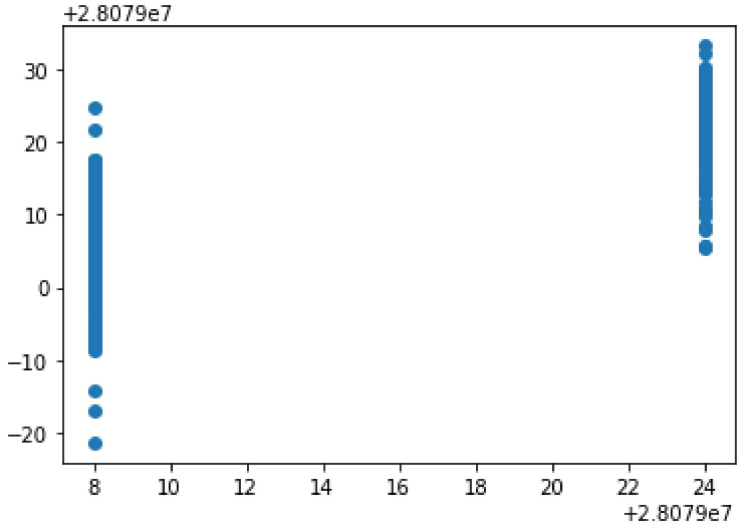
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	1.323470
CO	-9.604075
EBE	-0.717979
NMHC	14.552680
NO	0.079911
NO_2	-0.018050
O_3	-0.014215
PM10	0.015311
PM25	0.085798
SO_2	-1.115941
TCH	-10.579271
TOL	-0.094328

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

Out[26]:
```
<matplotlib.collections.PathCollection at 0x1d5180c4850>
```



ACCURACY

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

Out[27]: 0.86115815742106

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

Out[28]: 0.8759223747154492

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]: 0.8618700765500296

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

Out[32]: 0.8749717624639893

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]:
```
0.7317593857447761
```

Accuracy(Lasso)

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

Out[35]:
```
0.7321231205088827
```

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.        , -0.        , -0.        , -0.        ,  0.07529833,
       -0.05341669, -0.01216129,  0.02760881,  0.04400699, -1.32367541,
       -0.        , -0.08256708])
```

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

Out[38]:
```
28079025.972867627
```

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

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

Out[40]:
```
0.8162393550281835
```

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)))
```

```
2.525675002689289
11.760630403296968
3.429377553331941
```

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]: (16026, 12)

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

Out[45]: (16026,)

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]: 0.9971296642955197

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

Out[53]:
```
1.0
```

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

Out[54]:
```
array([[1.00000000e+00, 1.54284913e-35]])
```

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]:
```
0.993938313424853
```

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

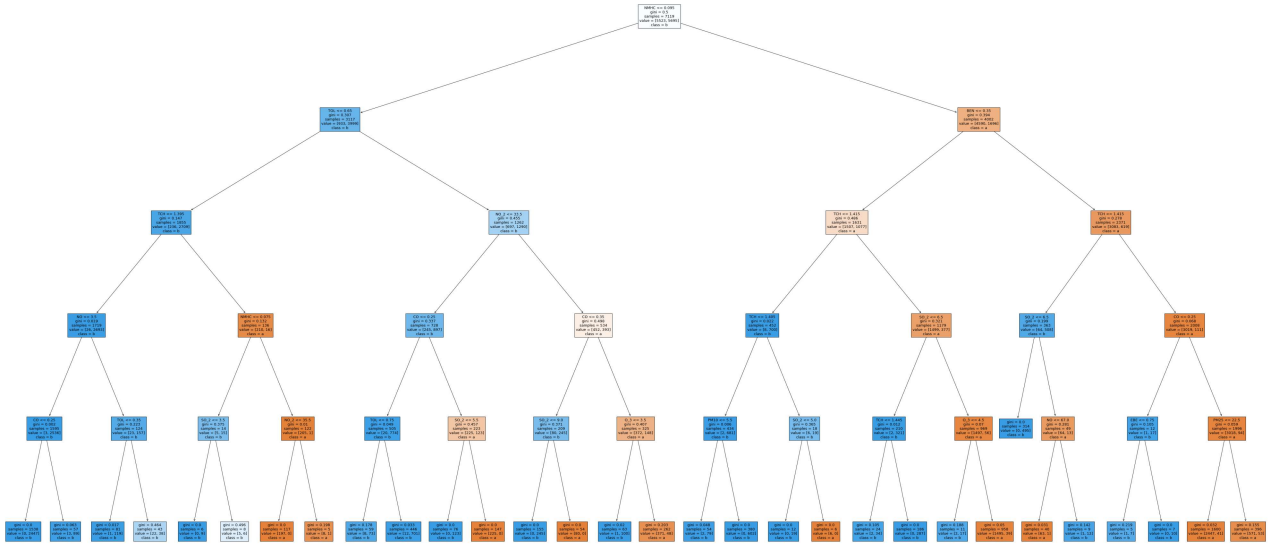
In [61]:
```
from sklearn.tree import plot_tree

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

Out[61]:
```
[Text(2315.7000000000003, 1993.2, 'NMHC <= 0.095\ngini = 0.5\nsamples = 7119\nvalue = [5
523, 5695]\nclass = b'),
 Text(1190.4, 1630.8000000000002, 'TOL <= 0.65\ngini = 0.307\nsamples = 3117\nvalue = [9
33, 3999]\nclass = b'),
```

```
 Text(595.2, 1268.4, 'TCH <= 1.395\ngini = 0.147\nsamples = 1855\nvalue = [236, 2709]\nc
lass = b'),
 Text(297.6, 906.0, 'NO <= 3.5\ngini = 0.019\nsamples = 1719\nvalue = [26, 2693]\nclass
= b'),
 Text(148.8, 543.5999999999999, 'CO <= 0.25\ngini = 0.002\nsamples = 1595\nvalue = [3, 2
536]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.0\nsamples = 1538\nvalue = [0, 2447]\nclass =
b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.063\nsamples = 57\nvalue = [3, 8
9]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'TOL <= 0.35\ngini = 0.223\nsamples = 124\n
value = [23, 157]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.017\nsamples = 81\nvalue = [1, 119]\nclass =
b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.464\nsamples = 43\nvalue = [22, 3
8]\nclass = b'),
 Text(892.8000000000001, 906.0, 'NMHC <= 0.075\ngini = 0.132\nsamples = 136\nvalue = [21
0, 16]\nclass = a'),
 Text(744.0, 543.5999999999999, 'SO_2 <= 3.5\ngini = 0.375\nsamples = 14\nvalue = [5, 1
5]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [0, 9]\nclass = b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.496\nsamples = 8\nvalue = [5, 6]
\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'NO_2 <= 35.5\ngini = 0.01\nsamples = 122\n
value = [205, 1]\nclass = a'),
 Text(967.2, 181.19999999999982, 'gini = 0.0\nsamples = 117\nvalue = [197, 0]\nclass =
a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.198\nsamples = 5\nvalue = [8, 1]\nclass =
a'),
 Text(1785.6000000000001, 1268.4, 'NO_2 <= 33.5\ngini = 0.455\nsamples = 1262\nvalue =
[697, 1290]\nclass = b'),
 Text(1488.0, 906.0, 'CO <= 0.25\ngini = 0.337\nsamples = 728\nvalue = [245, 897]\nclass
= b'),
 Text(1339.2, 543.5999999999999, 'TOL <= 0.75\ngini = 0.049\nsamples = 505\nvalue = [20,
774]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.178\nsamples = 59\nvalue = [8, 7
3]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.033\nsamples = 446\nvalue = [12,
701]\nclass = b'),
 Text(1636.8000000000002, 543.5999999999999, 'SO_2 <= 5.5\ngini = 0.457\nsamples = 223\n
value = [225, 123]\nclass = a'),
 Text(1562.4, 181.19999999999982, 'gini = 0.0\nsamples = 76\nvalue = [0, 123]\nclass =
b'),
 Text(1711.2, 181.19999999999982, 'gini = 0.0\nsamples = 147\nvalue = [225, 0]\nclass =
a'),
 Text(2083.2000000000003, 906.0, 'CO <= 0.35\ngini = 0.498\nsamples = 534\nvalue = [452,
393]\nclass = a'),
 Text(1934.4, 543.5999999999999, 'SO_2 <= 9.0\ngini = 0.371\nsamples = 209\nvalue = [80,
245]\nclass = b'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.0\nsamples = 155\nvalue = [0, 24
5]\nclass = b'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.0\nsamples = 54\nvalue = [80, 0]
\nclass = a'),
 Text(2232.0, 543.5999999999999, 'O_3 <= 3.5\ngini = 0.407\nsamples = 325\nvalue = [372,
148]\nclass = a'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.02\nsamples = 63\nvalue = [1, 10
0]\nclass = b'),
 Text(2306.4, 181.19999999999982, 'gini = 0.203\nsamples = 262\nvalue = [371, 48]\nclass
= a'),
 Text(3441.0000000000005, 1630.8000000000002, 'BEN <= 0.35\ngini = 0.394\nsamples = 4002
\nvalue = [4590, 1696]\nclass = a'),
 Text(2976.0, 1268.4, 'TCH <= 1.415\ngini = 0.486\nsamples = 1631\nvalue = [1507, 1077]
\nclass = a'),
 Text(2678.4, 906.0, 'TCH <= 1.405\ngini = 0.022\nsamples = 452\nvalue = [8, 700]\nclass
= b'),
```

```
 Text(2529.6000000000004, 543.5999999999999, 'PM10 <= 5.5\ngini = 0.006\nsamples = 434\n
value = [2, 681]\nclass = b'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.048\nsamples = 54\nvalue = [2, 7
9]\nclass = b'),
 Text(2604.0, 181.19999999999982, 'gini = 0.0\nsamples = 380\nvalue = [0, 602]\nclass =
b'),
 Text(2827.2000000000003, 543.5999999999999, 'SO_2 <= 5.0\ngini = 0.365\nsamples = 18\nv
alue = [6, 19]\nclass = b'),
 Text(2752.8, 181.19999999999982, 'gini = 0.0\nsamples = 12\nvalue = [0, 19]\nclass =
b'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]\n
class = a'),
 Text(3273.6000000000004, 906.0, 'SO_2 <= 6.5\ngini = 0.321\nsamples = 1179\nvalue = [14
99, 377]\nclass = a'),
 Text(3124.8, 543.5999999999999, 'TCH <= 1.445\ngini = 0.012\nsamples = 210\nvalue = [2,
321]\nclass = b'),
 Text(3050.4, 181.19999999999982, 'gini = 0.105\nsamples = 24\nvalue = [2, 34]\nclass =
b'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.0\nsamples = 186\nvalue = [0, 28
7]\nclass = b'),
 Text(3422.4, 543.5999999999999, 'O_3 <= 4.5\ngini = 0.07\nsamples = 969\nvalue = [1497,
56]\nclass = a'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.188\nsamples = 11\nvalue = [2, 1
7]\nclass = b'),
 Text(3496.8, 181.19999999999982, 'gini = 0.05\nsamples = 958\nvalue = [1495, 39]\nclass
= a'),
 Text(3906.0000000000005, 1268.4, 'TCH <= 1.415\ngini = 0.278\nsamples = 2371\nvalue =
[3083, 619]\nclass = a'),
 Text(3645.6000000000004, 906.0, 'SO_2 <= 6.5\ngini = 0.199\nsamples = 363\nvalue = [64,
508]\nclass = b'),
 Text(3571.2000000000003, 543.5999999999999, 'gini = 0.0\nsamples = 314\nvalue = [0, 49
5]\nclass = b'),
 Text(3720.0000000000005, 543.5999999999999, 'NO <= 67.0\ngini = 0.281\nsamples = 49\nva
lue = [64, 13]\nclass = a'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.031\nsamples = 40\nvalue = [63,
1]\nclass = a'),
 Text(3794.4, 181.19999999999982, 'gini = 0.142\nsamples = 9\nvalue = [1, 12]\nclass =
b'),
 Text(4166.400000000001, 906.0, 'CO <= 0.25\ngini = 0.068\nsamples = 2008\nvalue = [301
9, 111]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'EBE <= 0.75\ngini = 0.105\nsamples = 12\nv
alue = [1, 17]\nclass = b'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.219\nsamples = 5\nvalue = [1, 7]
\nclass = b'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.0\nsamples = 7\nvalue = [0, 10]
\nclass = b'),
 Text(4315.200000000001, 543.5999999999999, 'PM25 <= 22.5\ngini = 0.059\nsamples = 1996
\nvalue = [3018, 94]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.032\nsamples = 1600\nvalue = [2447, 41]\ncla
ss = a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.155\nsamples = 396\nvalue = [571, 53]\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: 0.86115815742106
Ridge Regression: 0.8618700765500296
Lasso Regression 0.7321231205088827
ElasticNet Regression: 0.8162393550281835
Logistic Regression: 0.9971296642955197
Random Forest: 0.993938313424853
```

Logistic Regression is suitable for this dataset