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_2016.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	stati
0	2016-11-01 01:00:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	280790
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	280790
2	2016-11-01 01:00:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	280790
3	2016-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	280790
4	2016-11-01 01:00:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	280790
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209491	2016-07-01 00:00:00	NaN	0.2	NaN	NaN	2.0	29.0	73.0	NaN	NaN	NaN	NaN	NaN	280790
209492	2016-07-01 00:00:00	NaN	0.3	NaN	NaN	1.0	29.0	NaN	36.0	NaN	5.0	NaN	NaN	280790
209493	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	1.0	19.0	71.0	NaN	NaN	NaN	NaN	NaN	280790

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	statio
209494	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	6.0	17.0	85.0	NaN	NaN	NaN	NaN	NaN	280790
209495	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	2.0	46.0	61.0	34.0	NaN	NaN	NaN	NaN	280790

209496 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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 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	1.1	28079008
6	0.8	28079024

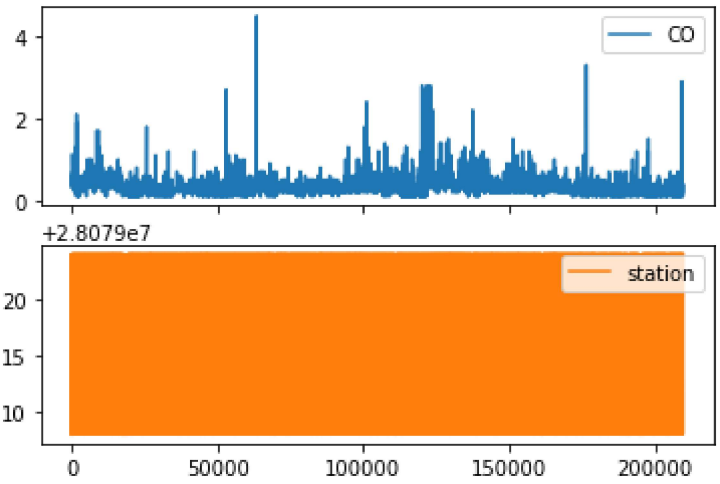
	CO	station
25	1.0	28079008
30	0.7	28079024
49	0.8	28079008
...	...	...
209430	0.2	28079024
209449	0.4	28079008
209454	0.2	28079024
209473	0.4	28079008
209478	0.2	28079024

16932 rows × 2 columns

Line chart

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

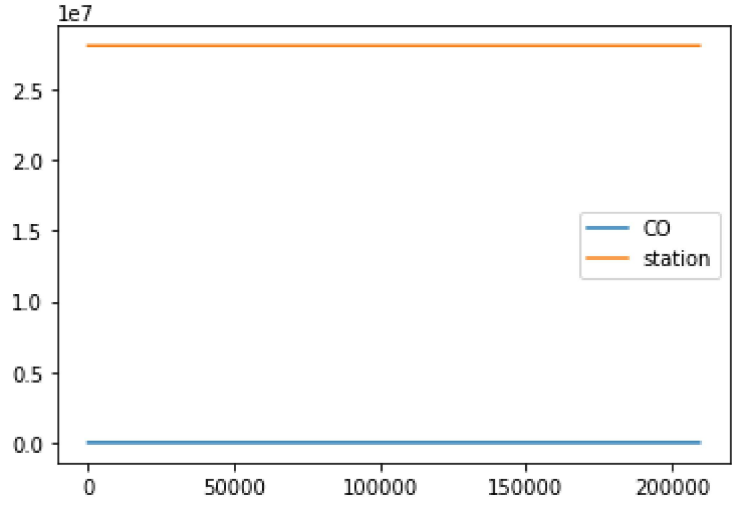
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

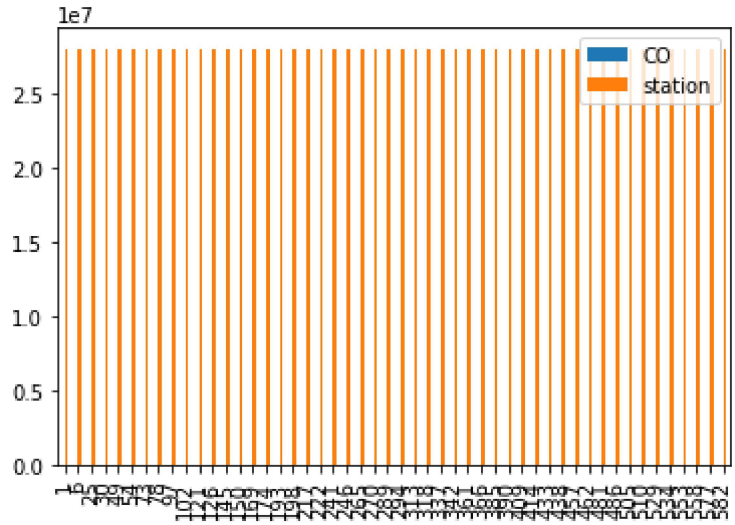


Bar chart

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

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

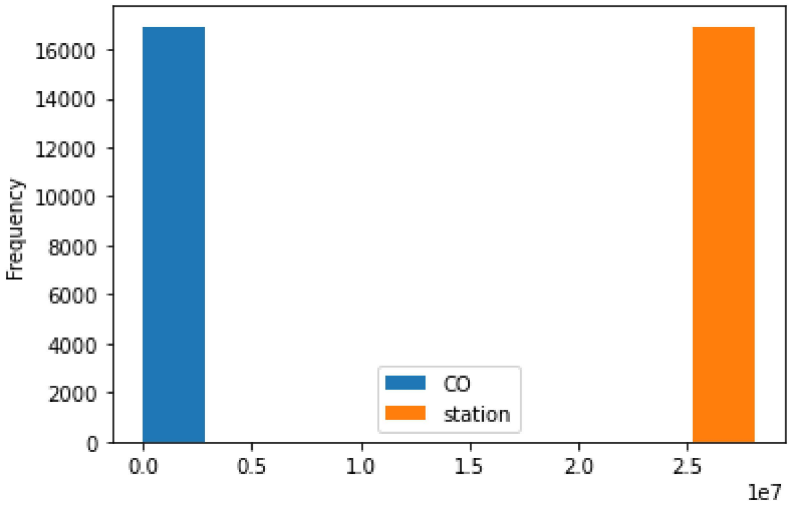
Out[10]: <AxesSubplot:>



Histogram

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

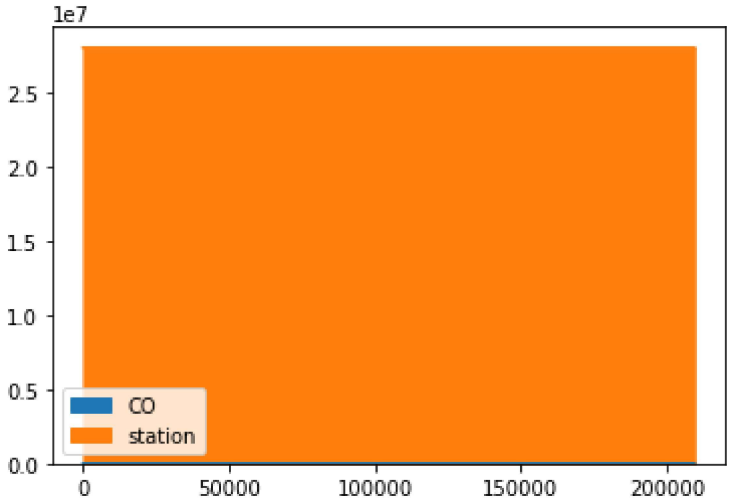
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

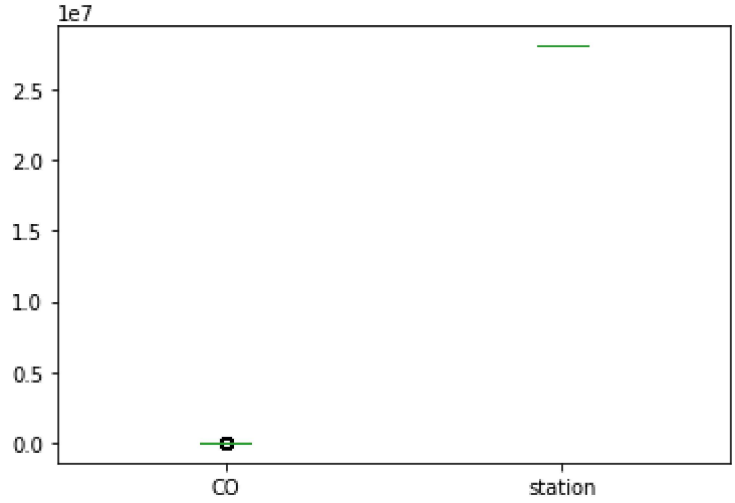
Out[12]: <AxesSubplot:>



Box chart

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

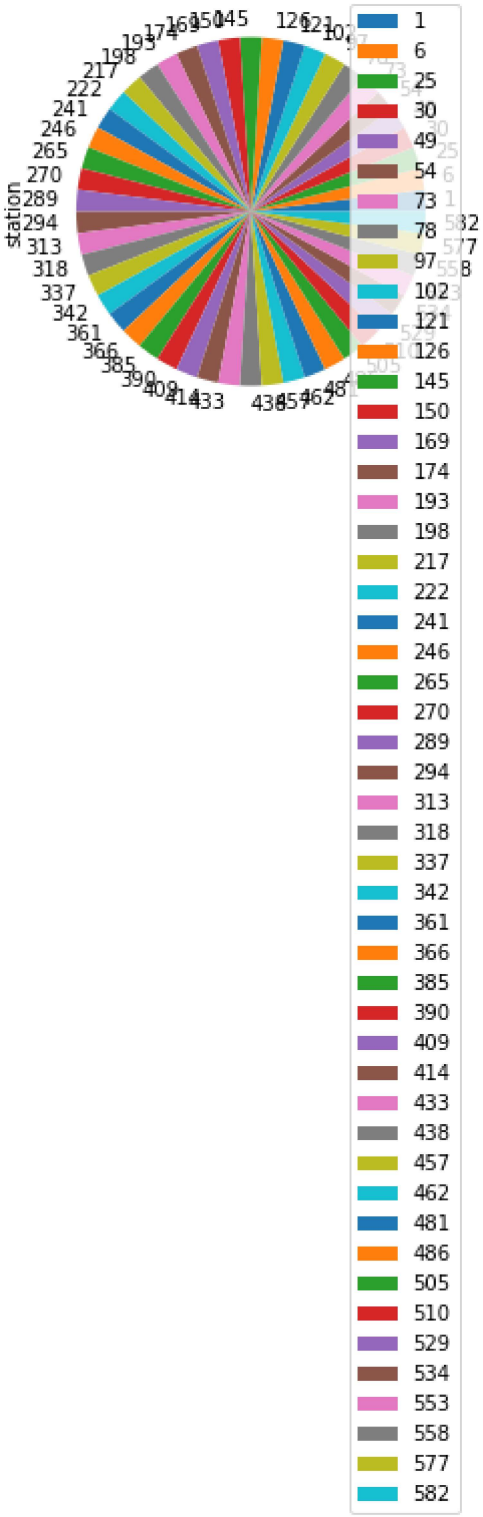
Out[13]: <AxesSubplot:>



Pie chart

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

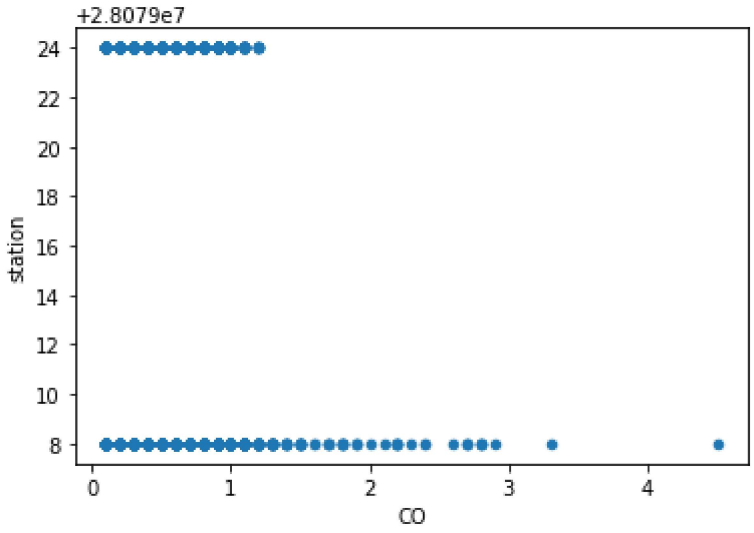
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 16932 entries, 1 to 209478
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     16932 non-null  object 
 1   BEN      16932 non-null  float64
 2   CO       16932 non-null  float64
 3   EBE      16932 non-null  float64
 4   NMHC     16932 non-null  float64
 5   NO       16932 non-null  float64
 6   NO_2     16932 non-null  float64
 7   O_3      16932 non-null  float64
 8   PM10     16932 non-null  float64
 9   PM25     16932 non-null  float64
 10  SO_2     16932 non-null  float64
 11  TCH      16932 non-null  float64
 12  TOL      16932 non-null  float64
 13  station  16932 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

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

Out[17]:

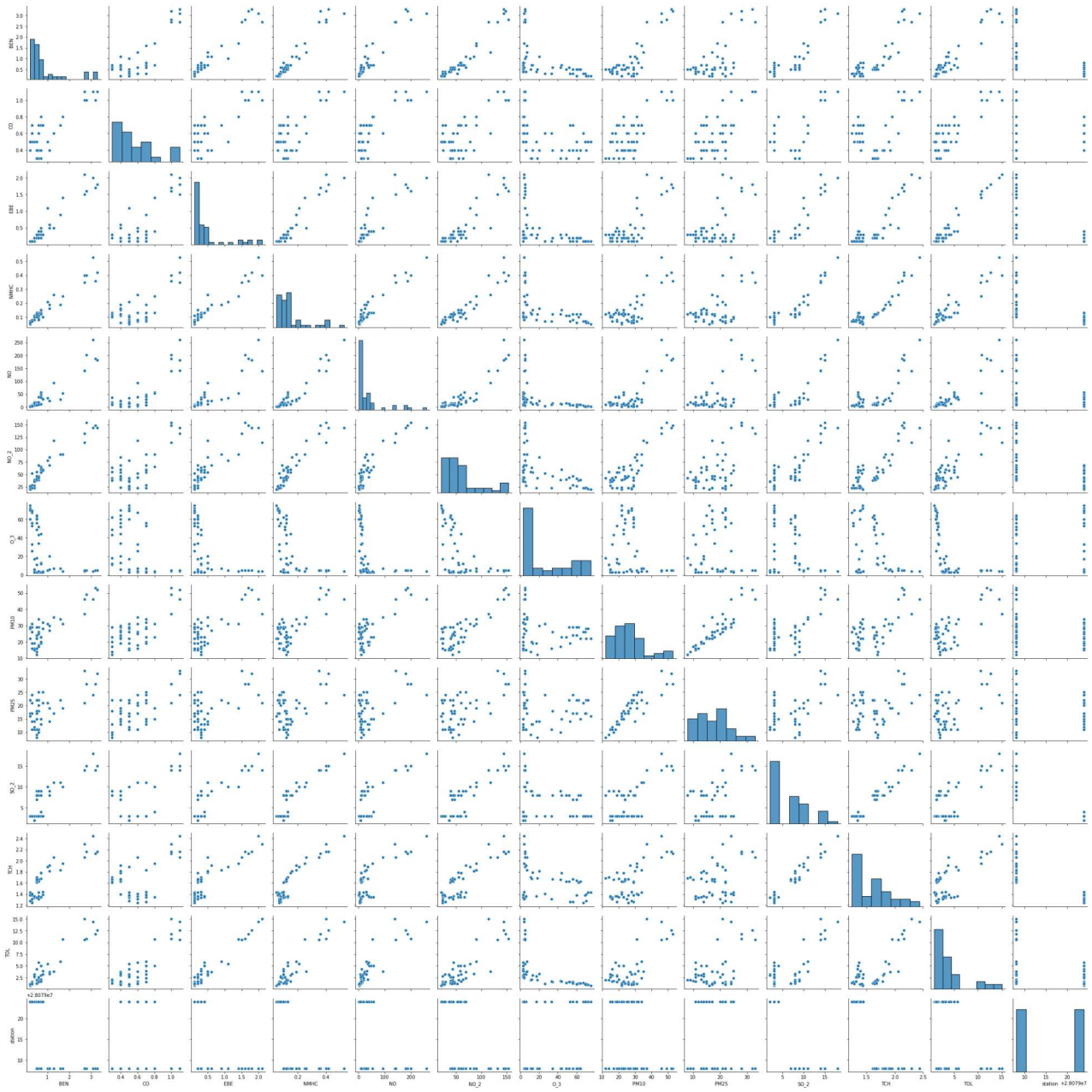
	BEN	CO	EBE	NMHC	NO	NO_2	O_3
count	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000	16932.000000
mean	0.537970	0.349941	0.298955	0.099913	20.815734	39.373376	48.118474
std	0.599479	0.203807	0.450204	0.079850	40.986063	31.170307	32.560271
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	1.000000
25%	0.200000	0.200000	0.100000	0.050000	1.000000	14.000000	21.000000
50%	0.400000	0.300000	0.200000	0.090000	7.000000	34.000000	46.000000
75%	0.700000	0.400000	0.300000	0.120000	23.000000	58.000000	69.000000

	BEN	CO	EBE	NMHC	NO	NO_2	O_3
max	12.300000	4.500000	13.500000	2.210000	829.000000	319.000000	181.00000

EDA AND VISUALIZATION

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

Out[18]: <seaborn.axisgrid.PairGrid at 0x27a607aff70>

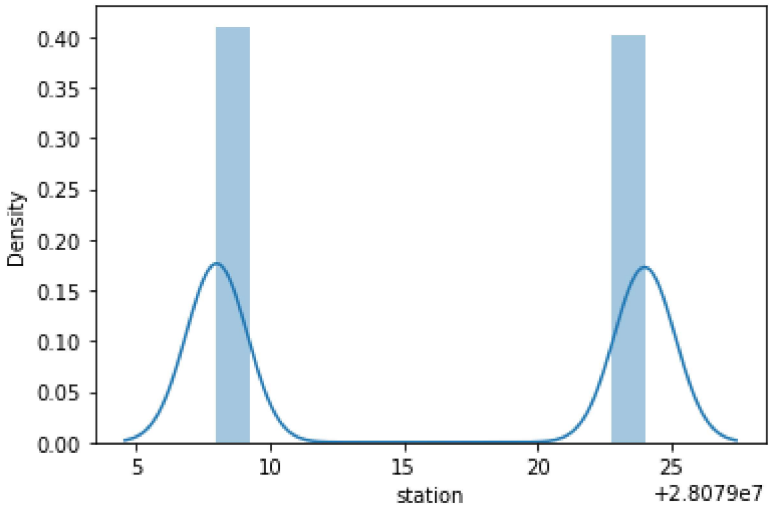


In [19]:
```
sns.distplot(df['station'])
```

```
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
```

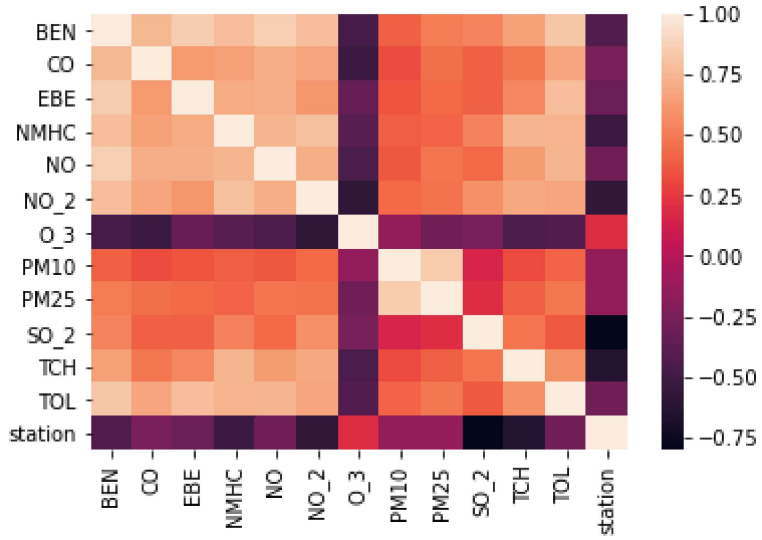
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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]:
```
28079042.232149646
```

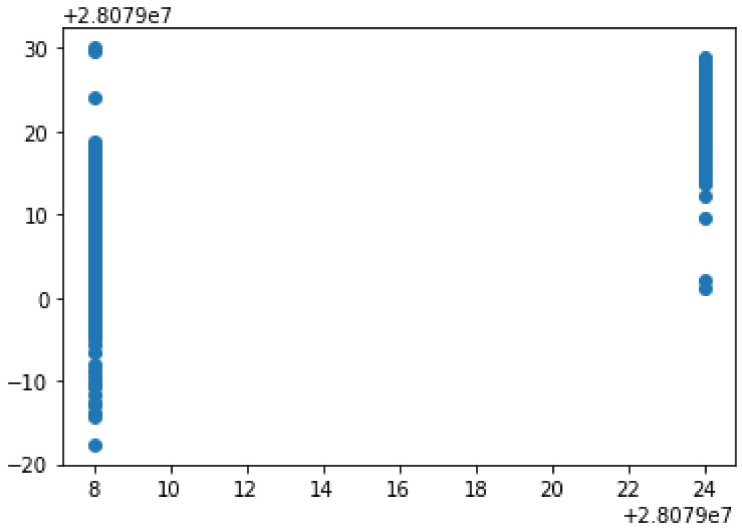
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	-1.789614
CO	4.703265
EBE	0.530774
NMHC	3.029672
NO	0.068447
NO_2	-0.068550
O_3	-0.023620
PM10	-0.012122
PM25	0.104467
SO_2	-0.810548
TCH	-14.290087
TOL	0.162564

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

Out[26]:
```
<matplotlib.collections.PathCollection at 0x27a6b6f83a0>
```



ACCURACY

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

Out[27]: 0.8240445151487018

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

Out[28]: 0.8295970183403071

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]: 0.8241246518374151

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

Out[32]: 0.8294725009318911

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]:
```
0.6490580260003715
```

Accuracy(Lasso)

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

Out[35]:
```
0.64336258517085
```

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.        ,  0.04978231,
       -0.10896416, -0.01995372,  0.00425888,  0.0489163 , -0.86624891,
       -0.00441998,  0.        ])
```

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

Out[38]:
```
28079026.163120836
```

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

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

Out[40]:
```
0.6985652534021703
```

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)))
```

```
3.371799136888177
19.288761777866362
4.391897286807418
```

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]: (16932, 12)

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

Out[45]: (16932,)

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]: 0.996161115048429

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

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

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

Out[54]:
```
array([[1.00000000e+00, 1.38109307e-55]])
```

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]:
```
0.9947688153898075
```

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

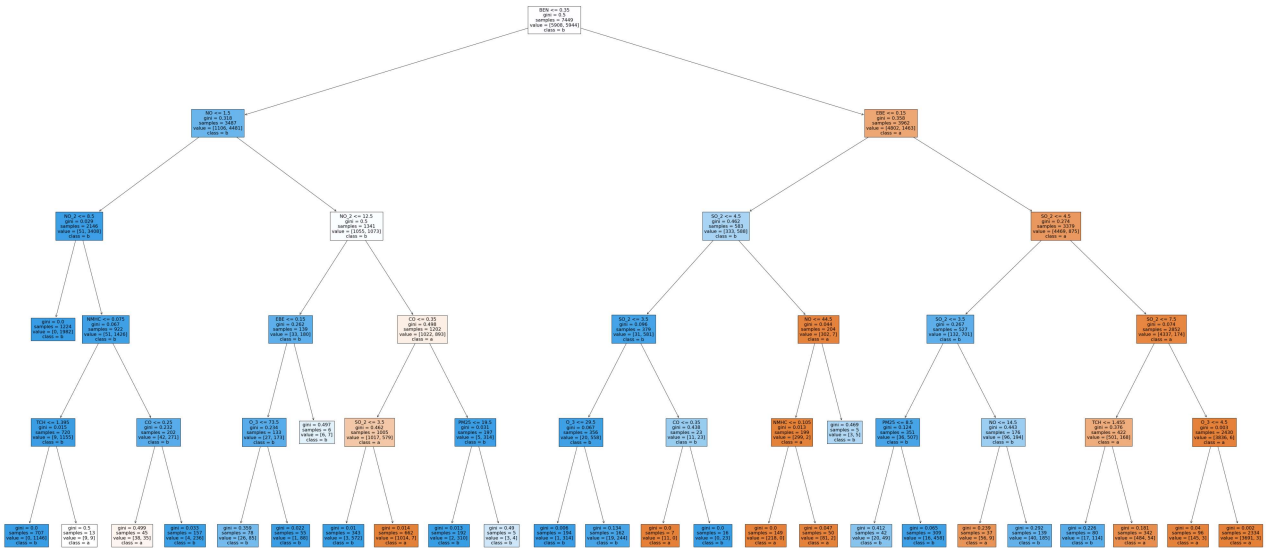
In [61]:
```
from sklearn.tree import plot_tree

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

Out[61]:
```
[Text(1953.0, 1993.2, 'BEN <= 0.35\ngini = 0.5\nsamples = 7449\nvalue = [5908, 5944]\ncl
ass = b'),
 Text(767.25, 1630.8000000000002, 'NO <= 1.5\ngini = 0.318\nsamples = 3487\nvalue = [110
6, 4481]\nclass = b'),
```

```
 Text(279.0, 1268.4, 'NO_2 <= 8.5\ngini = 0.029\nsamples = 2146\nvalue = [51, 3408]\ncla
ss = b'),
 Text(186.0, 906.0, 'gini = 0.0\nsamples = 1224\nvalue = [0, 1982]\nclass = b'),
 Text(372.0, 906.0, 'NMHC <= 0.075\ngini = 0.067\nsamples = 922\nvalue = [51, 1426]\ncla
ss = b'),
 Text(186.0, 543.5999999999999, 'TCH <= 1.395\ngini = 0.015\nsamples = 720\nvalue = [9,
1155]\nclass = b'),
 Text(93.0, 181.19999999999982, 'gini = 0.0\nsamples = 707\nvalue = [0, 1146]\nclass =
b'),
 Text(279.0, 181.19999999999982, 'gini = 0.5\nsamples = 13\nvalue = [9, 9]\nclass = a'),
 Text(558.0, 543.5999999999999, 'CO <= 0.25\ngini = 0.232\nsamples = 202\nvalue = [42, 2
71]\nclass = b'),
 Text(465.0, 181.19999999999982, 'gini = 0.499\nsamples = 45\nvalue = [38, 35]\nclass =
a'),
 Text(651.0, 181.19999999999982, 'gini = 0.033\nsamples = 157\nvalue = [4, 236]\nclass =
b'),
 Text(1255.5, 1268.4, 'NO_2 <= 12.5\ngini = 0.5\nsamples = 1341\nvalue = [1055, 1073]\nc
lass = b'),
 Text(1023.0, 906.0, 'EBE <= 0.15\ngini = 0.262\nsamples = 139\nvalue = [33, 180]\nclass
= b'),
 Text(930.0, 543.5999999999999, 'O_3 <= 73.5\ngini = 0.234\nsamples = 133\nvalue = [27,
173]\nclass = b'),
 Text(837.0, 181.19999999999982, 'gini = 0.359\nsamples = 78\nvalue = [26, 85]\nclass =
b'),
 Text(1023.0, 181.19999999999982, 'gini = 0.022\nsamples = 55\nvalue = [1, 88]\nclass =
b'),
 Text(1116.0, 543.5999999999999, 'gini = 0.497\nsamples = 6\nvalue = [6, 7]\nclass =
b'),
 Text(1488.0, 906.0, 'CO <= 0.35\ngini = 0.498\nsamples = 1202\nvalue = [1022, 893]\ncla
ss = a'),
 Text(1302.0, 543.5999999999999, 'SO_2 <= 3.5\ngini = 0.462\nsamples = 1005\nvalue = [10
17, 579]\nclass = a'),
 Text(1209.0, 181.19999999999982, 'gini = 0.01\nsamples = 343\nvalue = [3, 572]\nclass =
b'),
 Text(1395.0, 181.19999999999982, 'gini = 0.014\nsamples = 662\nvalue = [1014, 7]\nclass
= a'),
 Text(1674.0, 543.5999999999999, 'PM25 <= 19.5\ngini = 0.031\nsamples = 197\nvalue = [5,
314]\nclass = b'),
 Text(1581.0, 181.19999999999982, 'gini = 0.013\nsamples = 192\nvalue = [2, 310]\nclass
= b'),
 Text(1767.0, 181.19999999999982, 'gini = 0.49\nsamples = 5\nvalue = [3, 4]\nclass =
b'),
 Text(3138.75, 1630.8000000000002, 'EBE <= 0.15\ngini = 0.358\nsamples = 3962\nvalue =
[4802, 1463]\nclass = a'),
 Text(2557.5, 1268.4, 'SO_2 <= 4.5\ngini = 0.462\nsamples = 583\nvalue = [333, 588]\ncla
ss = b'),
 Text(2232.0, 906.0, 'SO_2 <= 3.5\ngini = 0.096\nsamples = 379\nvalue = [31, 581]\nclass
= b'),
 Text(2046.0, 543.5999999999999, 'O_3 <= 29.5\ngini = 0.067\nsamples = 356\nvalue = [20,
558]\nclass = b'),
 Text(1953.0, 181.19999999999982, 'gini = 0.006\nsamples = 194\nvalue = [1, 314]\nclass
= b'),
 Text(2139.0, 181.19999999999982, 'gini = 0.134\nsamples = 162\nvalue = [19, 244]\nclass
= b'),
 Text(2418.0, 543.5999999999999, 'CO <= 0.35\ngini = 0.438\nsamples = 23\nvalue = [11, 2
3]\nclass = b'),
 Text(2325.0, 181.19999999999982, 'gini = 0.0\nsamples = 7\nvalue = [11, 0]\nclass =
a'),
 Text(2511.0, 181.19999999999982, 'gini = 0.0\nsamples = 16\nvalue = [0, 23]\nclass =
b'),
 Text(2883.0, 906.0, 'NO <= 44.5\ngini = 0.044\nsamples = 204\nvalue = [302, 7]\nclass =
a'),
 Text(2790.0, 543.5999999999999, 'NMHC <= 0.105\ngini = 0.013\nsamples = 199\nvalue = [2
99, 2]\nclass = a'),
 Text(2697.0, 181.19999999999982, 'gini = 0.0\nsamples = 149\nvalue = [218, 0]\nclass =
```

```
a'),
 Text(2883.0, 181.19999999999982, 'gini = 0.047\nsamples = 50\nvalue = [81, 2]\nclass =
a'),
 Text(2976.0, 543.5999999999999, 'gini = 0.469\nsamples = 5\nvalue = [3, 5]\nclass =
b'),
 Text(3720.0, 1268.4, 'SO_2 <= 4.5\ngini = 0.274\nsamples = 3379\nvalue = [4469, 875]\nc
lass = a'),
 Text(3348.0, 906.0, 'SO_2 <= 3.5\ngini = 0.267\nsamples = 527\nvalue = [132, 701]\nclas
s = b'),
 Text(3162.0, 543.5999999999999, 'PM25 <= 8.5\ngini = 0.124\nsamples = 351\nvalue = [36,
507]\nclass = b'),
 Text(3069.0, 181.19999999999982, 'gini = 0.412\nsamples = 42\nvalue = [20, 49]\nclass =
b'),
 Text(3255.0, 181.19999999999982, 'gini = 0.065\nsamples = 309\nvalue = [16, 458]\nclass
= b'),
 Text(3534.0, 543.5999999999999, 'NO <= 14.5\ngini = 0.443\nsamples = 176\nvalue = [96,
194]\nclass = b'),
 Text(3441.0, 181.19999999999982, 'gini = 0.239\nsamples = 37\nvalue = [56, 9]\nclass =
a'),
 Text(3627.0, 181.19999999999982, 'gini = 0.292\nsamples = 139\nvalue = [40, 185]\nclass
= b'),
 Text(4092.0, 906.0, 'SO_2 <= 7.5\ngini = 0.074\nsamples = 2852\nvalue = [4337, 174]\ncl
ass = a'),
 Text(3906.0, 543.5999999999999, 'TCH <= 1.455\ngini = 0.376\nsamples = 422\nvalue = [50
1, 168]\nclass = a'),
 Text(3813.0, 181.19999999999982, 'gini = 0.226\nsamples = 80\nvalue = [17, 114]\nclass
= b'),
 Text(3999.0, 181.19999999999982, 'gini = 0.181\nsamples = 342\nvalue = [484, 54]\nclass
= a'),
 Text(4278.0, 543.5999999999999, 'O_3 <= 4.5\ngini = 0.003\nsamples = 2430\nvalue = [383
6, 6]\nclass = a'),
 Text(4185.0, 181.19999999999982, 'gini = 0.04\nsamples = 96\nvalue = [145, 3]\nclass =
a'),
 Text(4371.0, 181.19999999999982, 'gini = 0.002\nsamples = 2334\nvalue = [3691, 3]\nclas
s = 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: 0.8240445151487018
Ridge Regression: 0.8241246518374151
Lasso Regression 0.64336258517085
ElasticNet Regression: 0.6985652534021703
Logistic Regression: 0.996161115048429
Random Forest: 0.9947688153898075
```

Logistic Regression is suitable for this dataset