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_2017.csv")
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TC
0	2017-06-01 01:00:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	NaN	Na
1	2017-06-01 01:00:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1.4	2
2	2017-06-01 01:00:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	C
3	2017-06-01 01:00:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	NaN	Na
4	2017-06-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210115	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	NaN	Na
210116	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	NaN	Na
210117	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	NaN	Na

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TC
210118	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	NaN	Na
210119	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	NaN	Na

210120 rows × 16 columns

Data Cleaning and Data Preprocessing

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

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

Out[4]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

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

Out[6]:

	CO	station
87457	0.3	28079008

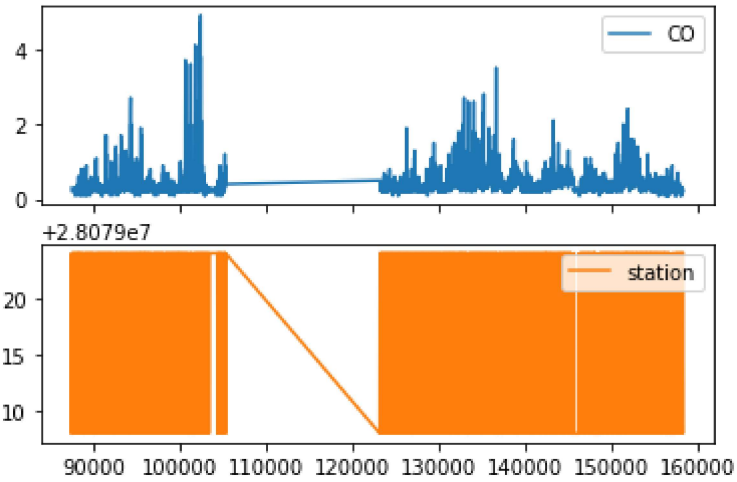
	CO	station
87462	0.2	28079024
87481	0.2	28079008
87486	0.2	28079024
87505	0.2	28079008
...	...	...
158238	0.2	28079024
158257	0.3	28079008
158262	0.2	28079024
158281	0.2	28079008
158286	0.2	28079024

4127 rows × 2 columns

Line chart

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

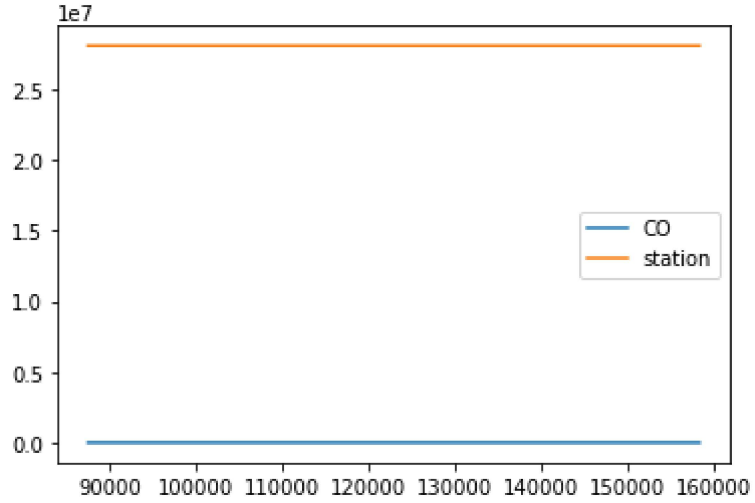
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

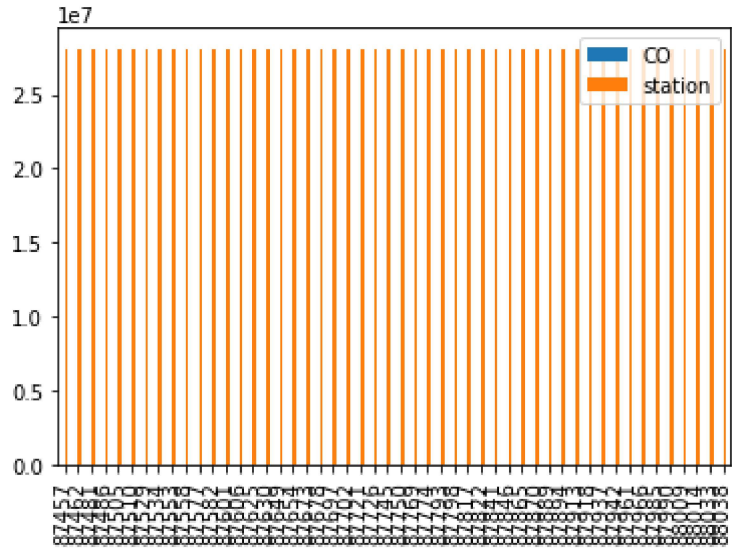


Bar chart

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

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

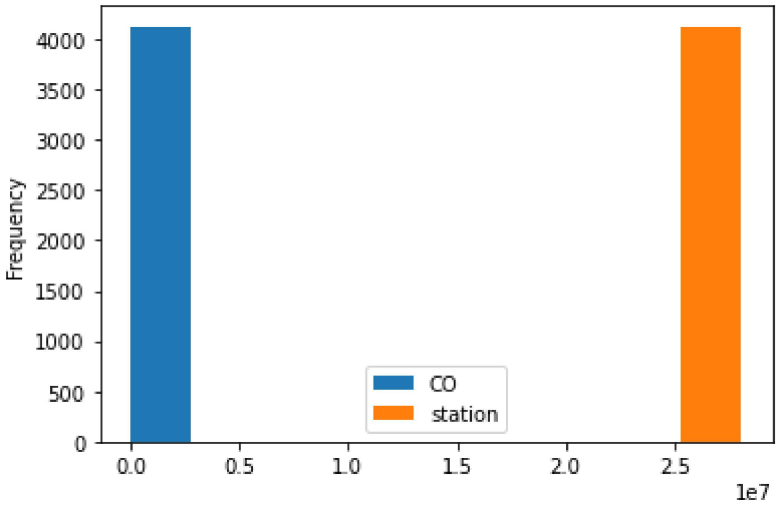
Out[10]: <AxesSubplot:>



Histogram

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

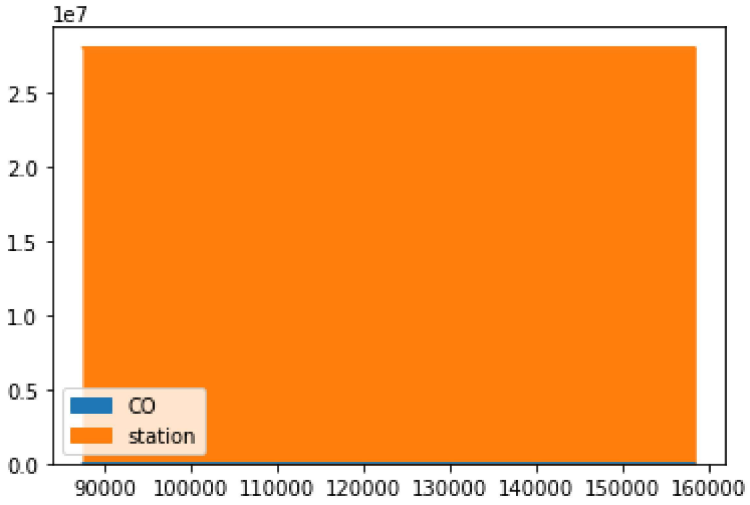
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

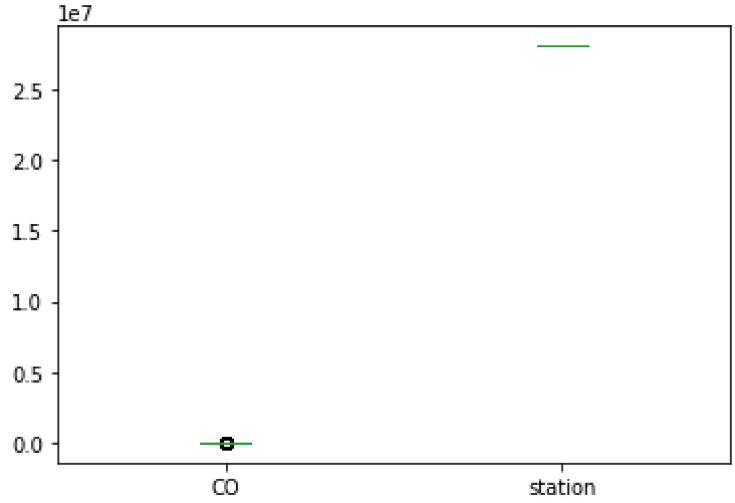
Out[12]: <AxesSubplot:>



Box chart

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

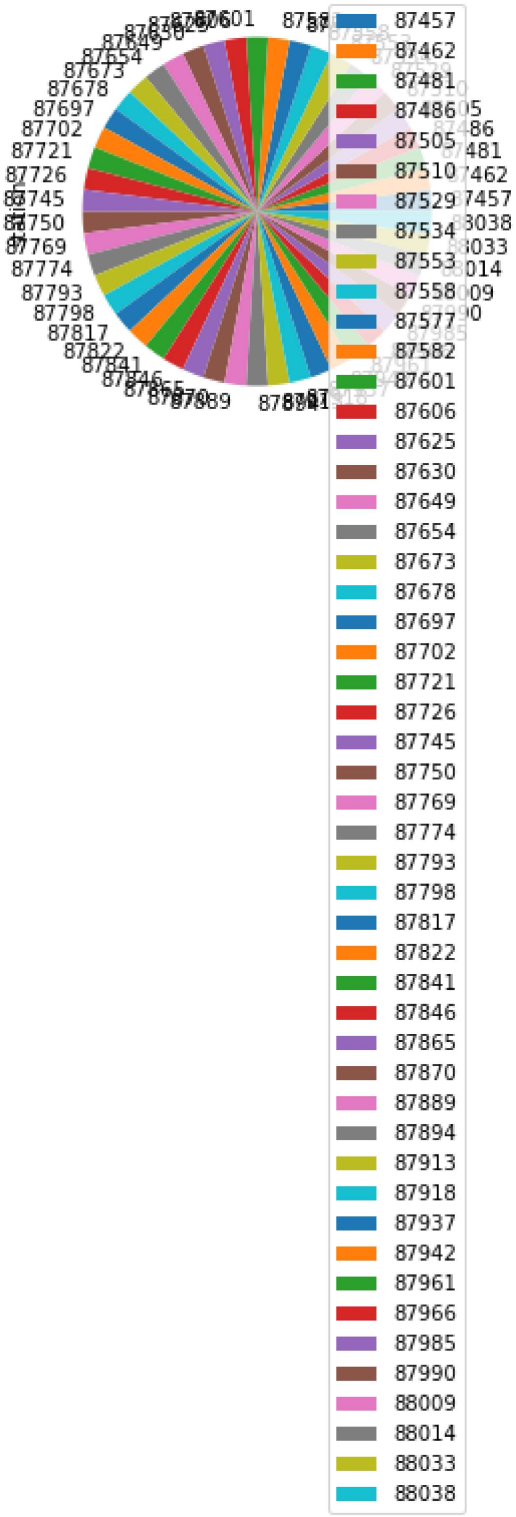
Out[13]: <AxesSubplot:>



Pie chart

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

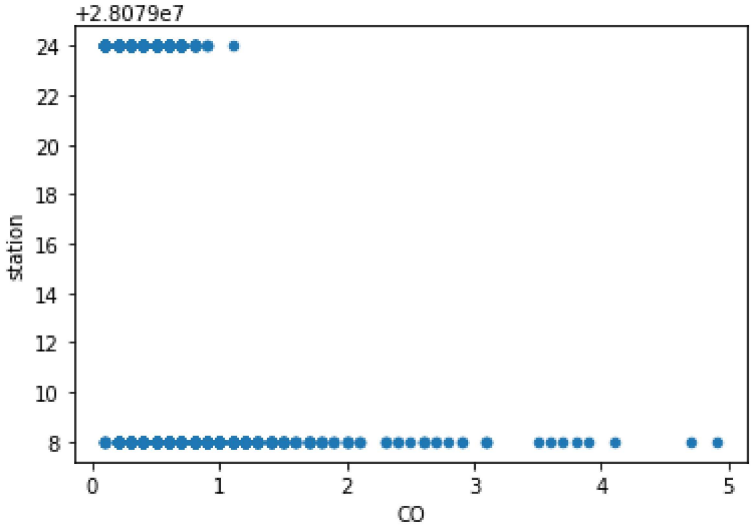
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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: 4127 entries, 87457 to 158286
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     4127 non-null   object 
 1   BEN      4127 non-null   float64
 2   CH4      4127 non-null   float64
 3   CO       4127 non-null   float64
 4   EBE      4127 non-null   float64
 5   NMHC     4127 non-null   float64
 6   NO       4127 non-null   float64
 7   NO_2     4127 non-null   float64
 8   NOx      4127 non-null   float64
 9   O_3      4127 non-null   float64
 10  PM10     4127 non-null   float64
 11  PM25     4127 non-null   float64
 12  SO_2     4127 non-null   float64
 13  TCH      4127 non-null   float64
 14  TOL      4127 non-null   float64
 15  station  4127 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 548.1+ KB
```

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

Out[17]:

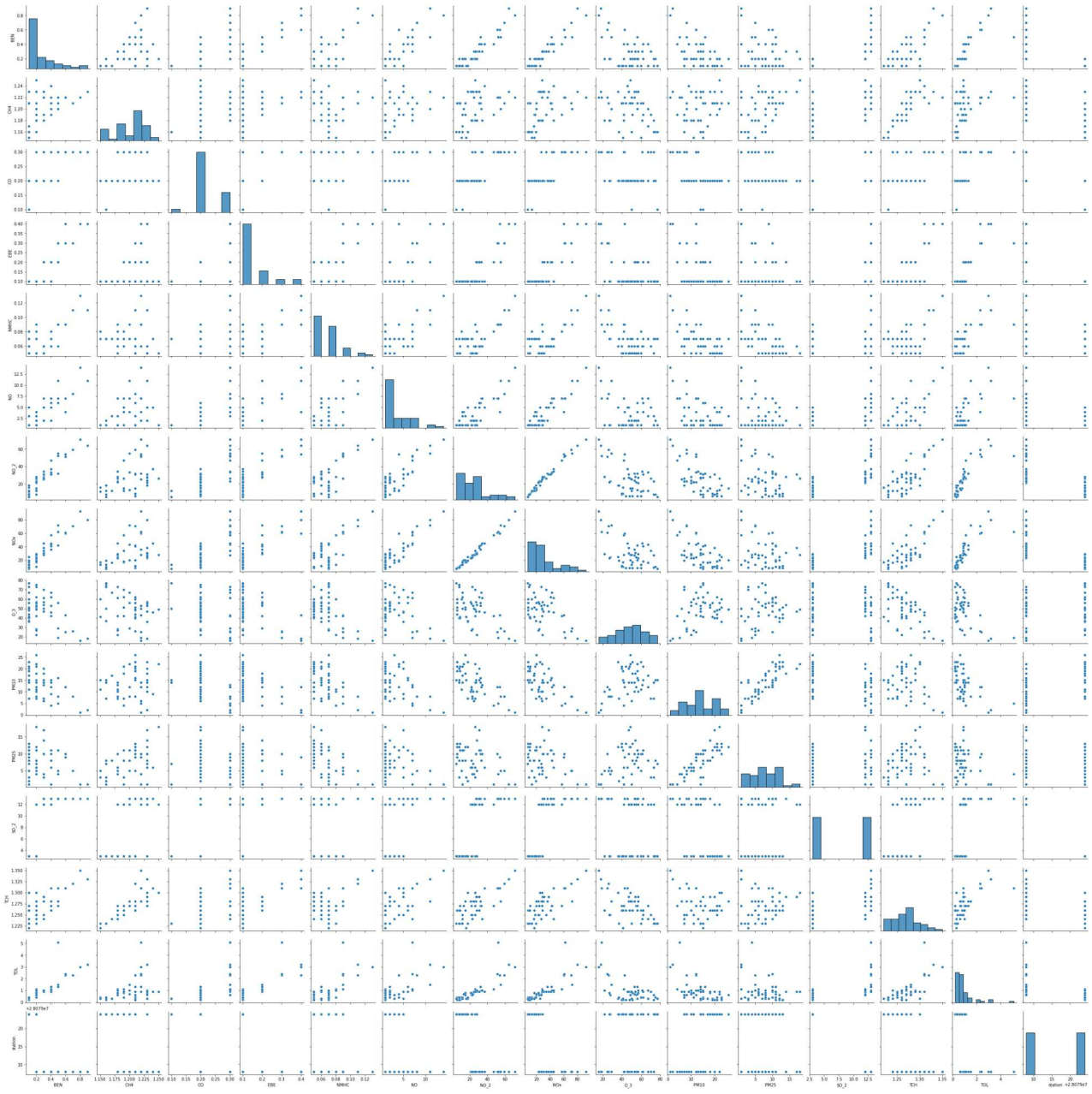
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
count	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.000000	4127.
mean	0.919918	1.323732	0.417858	0.578168	0.097269	41.785316	58.069057	122.
std	1.123078	0.215742	0.342871	0.962000	0.094035	71.118499	38.974112	142.
min	0.100000	1.100000	0.100000	0.100000	0.000000	1.000000	1.000000	2.
25%	0.300000	1.180000	0.200000	0.100000	0.050000	3.000000	30.000000	37.
50%	0.600000	1.270000	0.300000	0.300000	0.080000	16.000000	54.000000	80.
75%	1.100000	1.400000	0.500000	0.700000	0.110000	50.000000	78.000000	153.

	BEN	CH4	CO	EBE	NMHC	NO	NO_2	
max	19.600000	3.630000	4.900000	16.700001	1.420000	879.000000	349.000000	1681.

EDA AND VISUALIZATION

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

Out[18]: <seaborn.axisgrid.PairGrid at 0x1938bd3cc10>

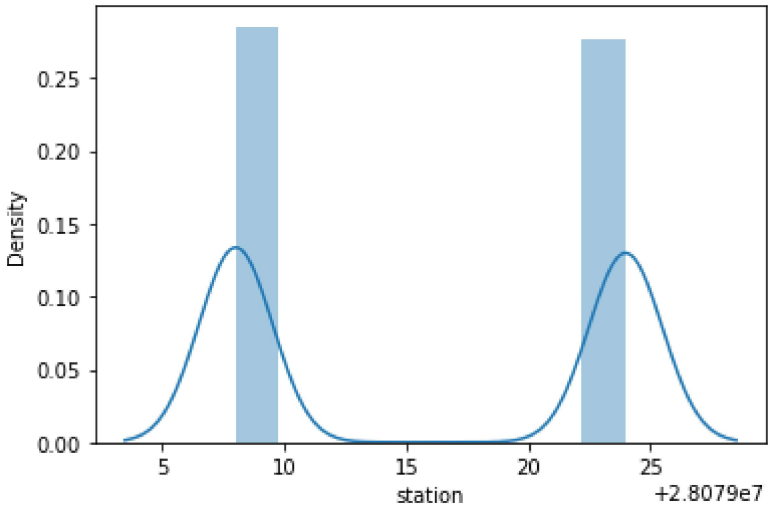


In [19]:
```
sns.distplot(df['station'])
```

```
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
```

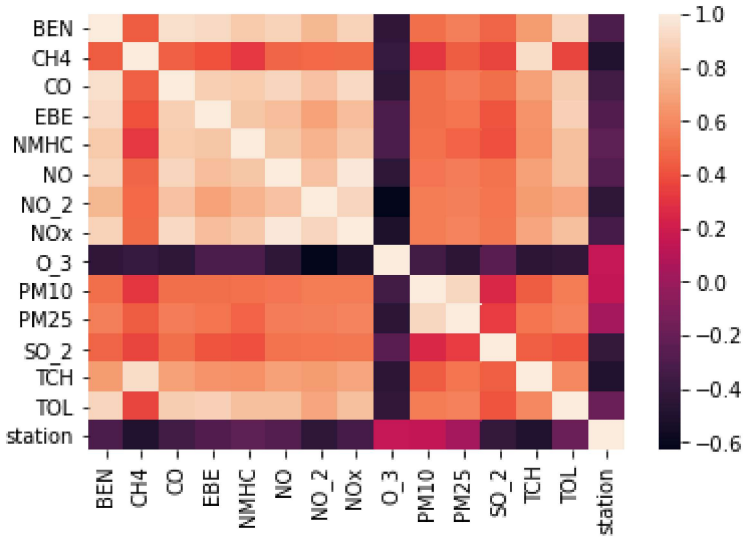
```
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

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]:
```
28079041.429375373
```

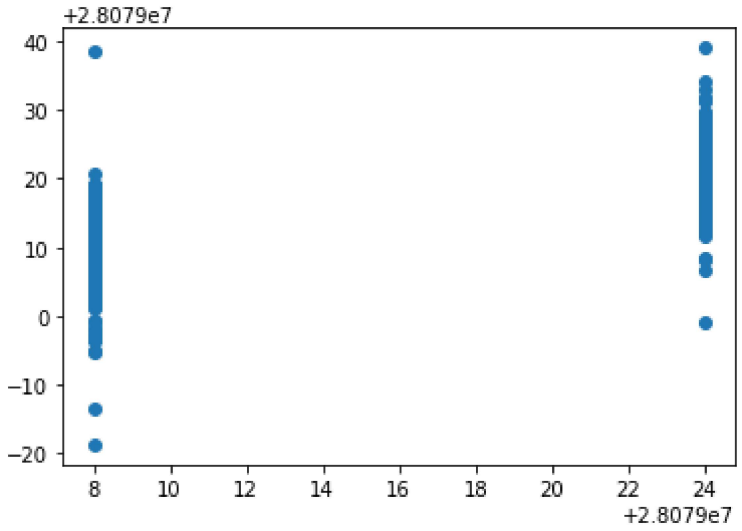
In [25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[25]:

	Co-efficient
BEN	0.023010
CO	-4.562207
EBE	-3.244087
NMHC	27.262080
NO	0.050540
NO_2	-0.183788
O_3	-0.085203
PM10	0.406205
PM25	-0.137726
SO_2	-0.275413
TCH	-13.419832
TOL	0.377682

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

Out[26]:
```
<matplotlib.collections.PathCollection at 0x19399accdc0>
```



ACCURACY

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

Out[27]: 0.6105325193243455

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

Out[28]: 0.6414900667299002

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]: 0.6007731100091809

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

Out[32]: 0.6320426244697528

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]:
```
0.41028558350989697
```

Accuracy(Lasso)

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

Out[35]:
```
0.38653046258925305
```

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.        ,  0.03099954,
       -0.20495532, -0.08378987,  0.54101678, -0.38511888, -0.29555376,
       -0.        ,  0.        ])
```

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

Out[38]:
```
28079025.268195264
```

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

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

Out[40]:
```
0.49910972111641516
```

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)))
```

```
4.860353989645394
32.03941576818532
5.6603370719582875
```

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]: (4127, 12)

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

Out[45]: (4127,)

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]: 0.9520232614489944

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

Out[53]:
```
0.9999999999999389
```

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

Out[54]:
```
array([[1.00000000e+00, 6.10312359e-14]])
```

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]:
```
0.9684903047091413
```

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

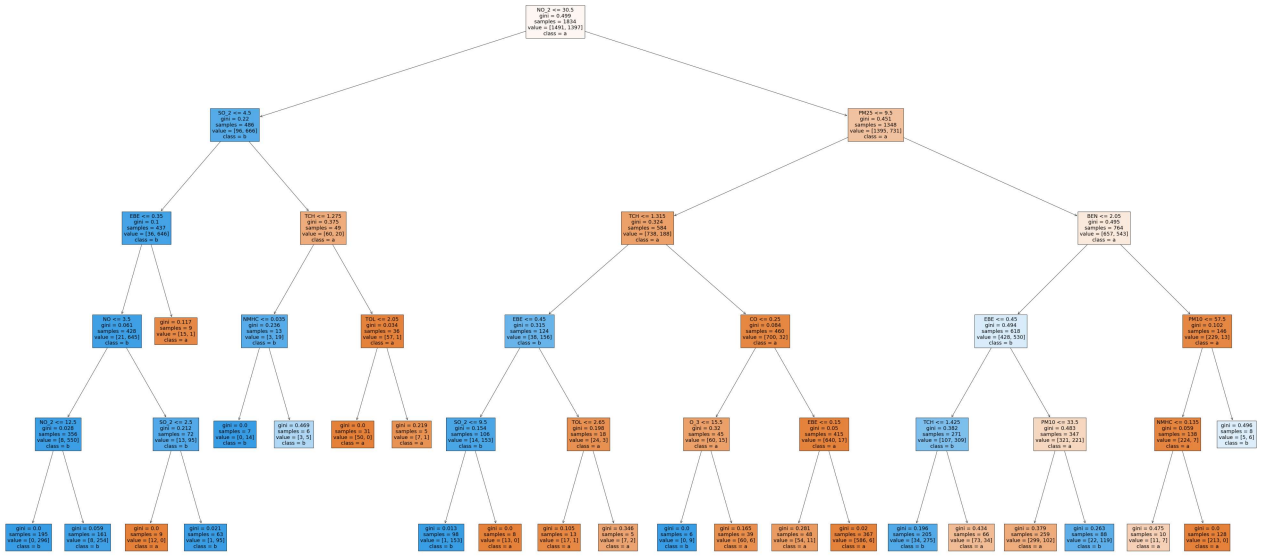
In [61]:
```
from sklearn.tree import plot_tree

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

Out[61]:
```
[Text(1959.4883720930231, 1993.2, 'NO_2 <= 30.5\ngini = 0.499\nsamples = 1834\nvalue =
[1491, 1397]\nclass = a'),
 Text(830.5116279069767, 1630.8000000000002, 'SO_2 <= 4.5\ngini = 0.22\nsamples = 486\nv
alue = [96, 666]\nclass = b'),
```

```
 Text(519.0697674418604, 1268.4, 'EBE <= 0.35\ngini = 0.1\nsamples = 437\nvalue = [36, 6
46]\nclass = b'),
 Text(415.25581395348837, 906.0, 'NO <= 3.5\ngini = 0.061\nsamples = 428\nvalue = [21, 6
45]\nclass = b'),
 Text(207.62790697674419, 543.5999999999999, 'NO_2 <= 12.5\ngini = 0.028\nsamples = 356
\nvalue = [8, 550]\nclass = b'),
 Text(103.81395348837209, 181.19999999999982, 'gini = 0.0\nsamples = 195\nvalue = [0, 29
6]\nclass = b'),
 Text(311.4418604651163, 181.19999999999982, 'gini = 0.059\nsamples = 161\nvalue = [8, 2
54]\nclass = b'),
 Text(622.8837209302326, 543.5999999999999, 'SO_2 <= 2.5\ngini = 0.212\nsamples = 72\nva
lue = [13, 95]\nclass = b'),
 Text(519.0697674418604, 181.19999999999982, 'gini = 0.0\nsamples = 9\nvalue = [12, 0]\n
class = a'),
 Text(726.6976744186046, 181.19999999999982, 'gini = 0.021\nsamples = 63\nvalue = [1, 9
5]\nclass = b'),
 Text(622.8837209302326, 906.0, 'gini = 0.117\nsamples = 9\nvalue = [15, 1]\nclass =
a'),
 Text(1141.953488372093, 1268.4, 'TCH <= 1.275\ngini = 0.375\nsamples = 49\nvalue = [60,
20]\nclass = a'),
 Text(934.3255813953489, 906.0, 'NMHC <= 0.035\ngini = 0.236\nsamples = 13\nvalue = [3,
19]\nclass = b'),
 Text(830.5116279069767, 543.5999999999999, 'gini = 0.0\nsamples = 7\nvalue = [0, 14]\nc
lass = b'),
 Text(1038.139534883721, 543.5999999999999, 'gini = 0.469\nsamples = 6\nvalue = [3, 5]\n
class = b'),
 Text(1349.5813953488373, 906.0, 'TOL <= 2.05\ngini = 0.034\nsamples = 36\nvalue = [57,
1]\nclass = a'),
 Text(1245.7674418604652, 543.5999999999999, 'gini = 0.0\nsamples = 31\nvalue = [50, 0]
\nclass = a'),
 Text(1453.3953488372092, 543.5999999999999, 'gini = 0.219\nsamples = 5\nvalue = [7, 1]
\nclass = a'),
 Text(3088.4651162790697, 1630.8000000000002, 'PM25 <= 9.5\ngini = 0.451\nsamples = 1348
\nvalue = [1395, 731]\nclass = a'),
 Text(2283.906976744186, 1268.4, 'TCH <= 1.315\ngini = 0.324\nsamples = 584\nvalue = [73
8, 188]\nclass = a'),
 Text(1868.6511627906978, 906.0, 'EBE <= 0.45\ngini = 0.315\nsamples = 124\nvalue = [38,
156]\nclass = b'),
 Text(1661.0232558139535, 543.5999999999999, 'SO_2 <= 9.5\ngini = 0.154\nsamples = 106\n
value = [14, 153]\nclass = b'),
 Text(1557.2093023255813, 181.19999999999982, 'gini = 0.013\nsamples = 98\nvalue = [1, 1
53]\nclass = b'),
 Text(1764.8372093023256, 181.19999999999982, 'gini = 0.0\nsamples = 8\nvalue = [13, 0]
\nclass = a'),
 Text(2076.279069767442, 543.5999999999999, 'TOL <= 2.65\ngini = 0.198\nsamples = 18\nva
lue = [24, 3]\nclass = a'),
 Text(1972.4651162790697, 181.19999999999982, 'gini = 0.105\nsamples = 13\nvalue = [17,
1]\nclass = a'),
 Text(2180.093023255814, 181.19999999999982, 'gini = 0.346\nsamples = 5\nvalue = [7, 2]
\nclass = a'),
 Text(2699.1627906976746, 906.0, 'CO <= 0.25\ngini = 0.084\nsamples = 460\nvalue = [700,
32]\nclass = a'),
 Text(2491.5348837209303, 543.5999999999999, 'O_3 <= 15.5\ngini = 0.32\nsamples = 45\nva
lue = [60, 15]\nclass = a'),
 Text(2387.720930232558, 181.19999999999982, 'gini = 0.0\nsamples = 6\nvalue = [0, 9]\nc
lass = b'),
 Text(2595.3488372093025, 181.19999999999982, 'gini = 0.165\nsamples = 39\nvalue = [60,
6]\nclass = a'),
 Text(2906.7906976744184, 543.5999999999999, 'EBE <= 0.15\ngini = 0.05\nsamples = 415\nv
alue = [640, 17]\nclass = a'),
 Text(2802.9767441860463, 181.19999999999982, 'gini = 0.281\nsamples = 48\nvalue = [54,
11]\nclass = a'),
 Text(3010.6046511627906, 181.19999999999982, 'gini = 0.02\nsamples = 367\nvalue = [586,
6]\nclass = a'),
 Text(3893.0232558139533, 1268.4, 'BEN <= 2.05\ngini = 0.495\nsamples = 764\nvalue = [65
```

```
7, 543]\nclass = a'),
 Text(3529.6744186046512, 906.0, 'EBE <= 0.45\ngini = 0.494\nsamples = 618\nvalue = [42
8, 530]\nclass = b'),
 Text(3322.046511627907, 543.5999999999999, 'TCH <= 1.425\ngini = 0.382\nsamples = 271\n
value = [107, 309]\nclass = b'),
 Text(3218.232558139535, 181.19999999999982, 'gini = 0.196\nsamples = 205\nvalue = [34,
275]\nclass = b'),
 Text(3425.860465116279, 181.19999999999982, 'gini = 0.434\nsamples = 66\nvalue = [73, 3
4]\nclass = a'),
 Text(3737.3023255813955, 543.5999999999999, 'PM10 <= 33.5\ngini = 0.483\nsamples = 347
\nvalue = [321, 221]\nclass = a'),
 Text(3633.4883720930234, 181.19999999999982, 'gini = 0.379\nsamples = 259\nvalue = [29
9, 102]\nclass = a'),
 Text(3841.1162790697676, 181.19999999999982, 'gini = 0.263\nsamples = 88\nvalue = [22,
119]\nclass = b'),
 Text(4256.372093023256, 906.0, 'PM10 <= 57.5\ngini = 0.102\nsamples = 146\nvalue = [22
9, 13]\nclass = a'),
 Text(4152.558139534884, 543.5999999999999, 'NMHC <= 0.135\ngini = 0.059\nsamples = 138
\nvalue = [224, 7]\nclass = a'),
 Text(4048.7441860465115, 181.19999999999982, 'gini = 0.475\nsamples = 10\nvalue = [11,
7]\nclass = a'),
 Text(4256.372093023256, 181.19999999999982, 'gini = 0.0\nsamples = 128\nvalue = [213,
0]\nclass = a'),
 Text(4360.186046511628, 543.5999999999999, 'gini = 0.496\nsamples = 8\nvalue = [5, 6]\n
class = b')]
```



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: 0.6105325193243455
Ridge Regression: 0.6007731100091809
```

```
Lasso Regression 0.38653046258925305
ElasticNet Regression: 0.49910972111641516
Logistic Regression: 0.9520232614489944
Random Forest: 0.9684903047091413
```

Random Forest is suitable for this dataset