# 4zlyggvsv

August 3, 2023

# 1 20104169 - SUMESH R

# 2 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid_2018.
↪csv")
df
```

```
Mounted at /content/drive
```

[2]:
```
                      date  BEN   CH4   CO  EBE  NMHC     NO   NO_2    NOx  \
0      2018-03-01 01:00:00  NaN   NaN  0.3  NaN   NaN    1.0   29.0   31.0
1      2018-03-01 01:00:00  0.5  1.39  0.3  0.2  0.02    6.0   40.0   49.0
2      2018-03-01 01:00:00  0.4   NaN  NaN  0.2   NaN    4.0   41.0   47.0
3      2018-03-01 01:00:00  NaN   NaN  0.3  NaN   NaN    1.0   35.0   37.0
4      2018-03-01 01:00:00  NaN   NaN  NaN  NaN   NaN    1.0   27.0   29.0
…                      …    …    …    …    …    …      …      …      …
69091  2018-02-01 00:00:00  NaN   NaN  0.5  NaN   NaN   66.0   91.0  192.0
69092  2018-02-01 00:00:00  NaN   NaN  0.7  NaN   NaN   87.0  107.0  241.0
69093  2018-02-01 00:00:00  NaN   NaN  NaN  NaN   NaN   28.0   48.0   91.0
69094  2018-02-01 00:00:00  NaN   NaN  NaN  NaN   NaN  141.0  103.0  320.0
69095  2018-02-01 00:00:00  NaN   NaN  NaN  NaN   NaN   69.0   96.0  202.0

        O_3  PM10  PM25  SO_2   TCH  TOL   station
0       NaN   NaN   NaN   2.0   NaN  NaN  28079004
1      52.0   5.0   4.0   3.0  1.41  0.8  28079008
2       NaN   NaN   NaN   NaN   NaN  1.1  28079011
3      54.0   NaN   NaN   NaN   NaN  NaN  28079016
4      49.0   NaN   NaN   3.0   NaN  NaN  28079017
…       …    …    …    …    …    …         …
```

```
69091   1.0  35.0  22.0   NaN   NaN  NaN  28079056
69092   NaN  29.0   NaN  15.0   NaN  NaN  28079057
69093   2.0   NaN   NaN   NaN   NaN  NaN  28079058
69094   2.0   NaN   NaN   NaN   NaN  NaN  28079059
69095   3.0  26.0   NaN   NaN   NaN  NaN  28079060

[69096 rows x 16 columns]
```

## 3 Data Cleaning and Data Preprocessing

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

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

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

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     4562 non-null   object
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

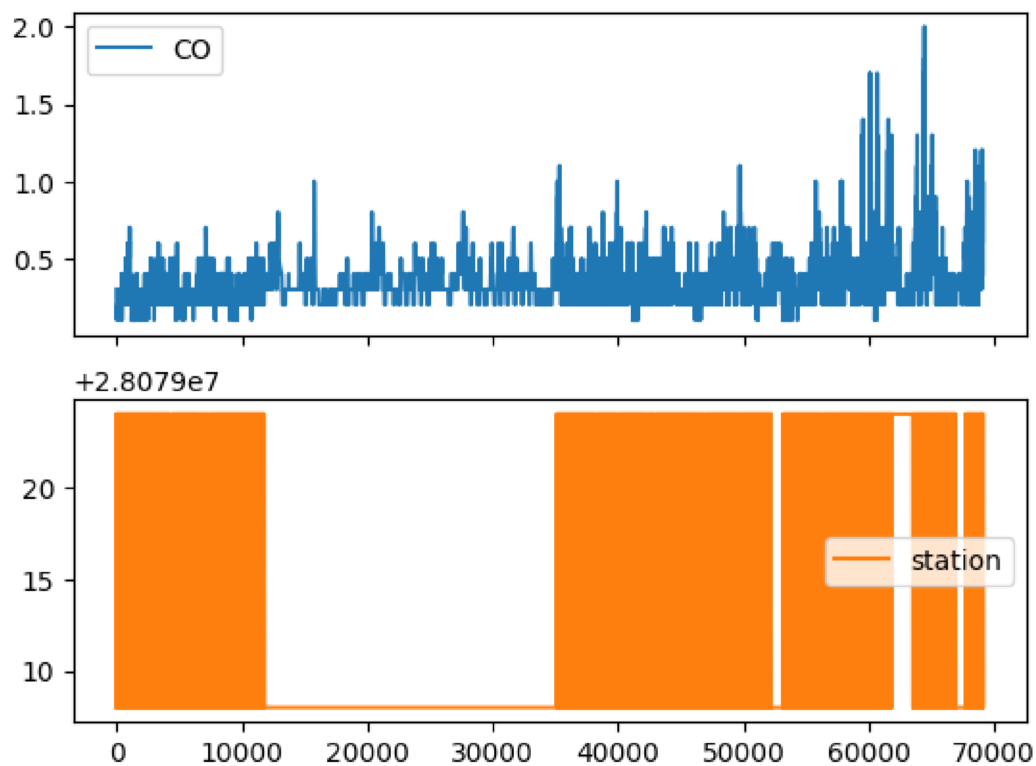
[6]:
```
data=df[['CO' ,'station']]
data
```

[6]:
```
          CO   station
1        0.3  28079008
6        0.2  28079024
25       0.2  28079008
30       0.2  28079024
49       0.2  28079008
…     …          …
69030  0.7  28079024
69049  1.2  28079008
69054  0.6  28079024
69073  1.0  28079008
69078  0.4  28079024

[4562 rows x 2 columns]
```

## 4 Line chart

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

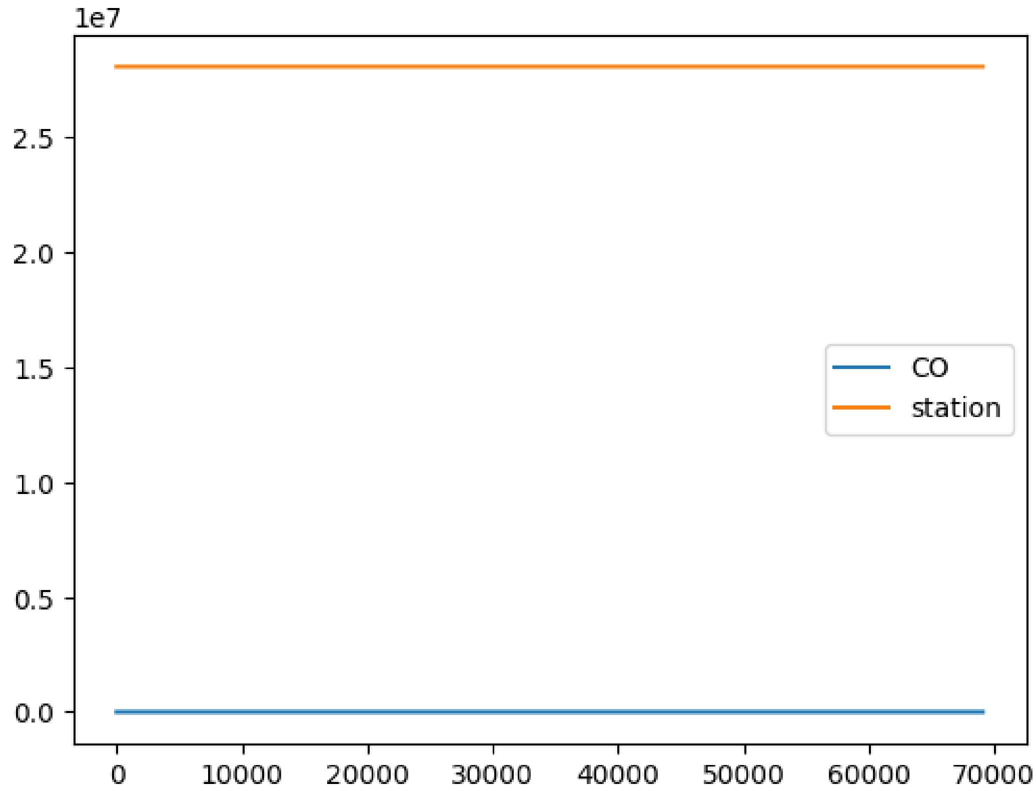
[7]:
```
array([<Axes: >, <Axes: >], dtype=object)
```



## 5 Line chart

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

[8]: <Axes: >

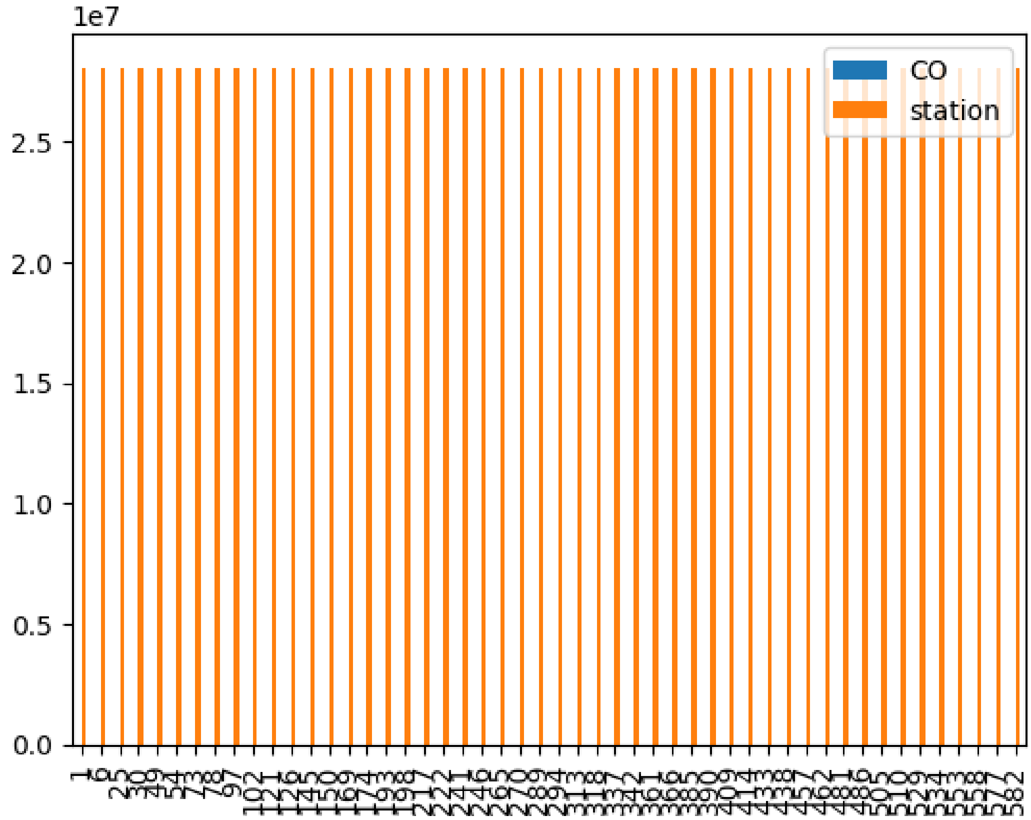



## 6 Bar chart

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

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

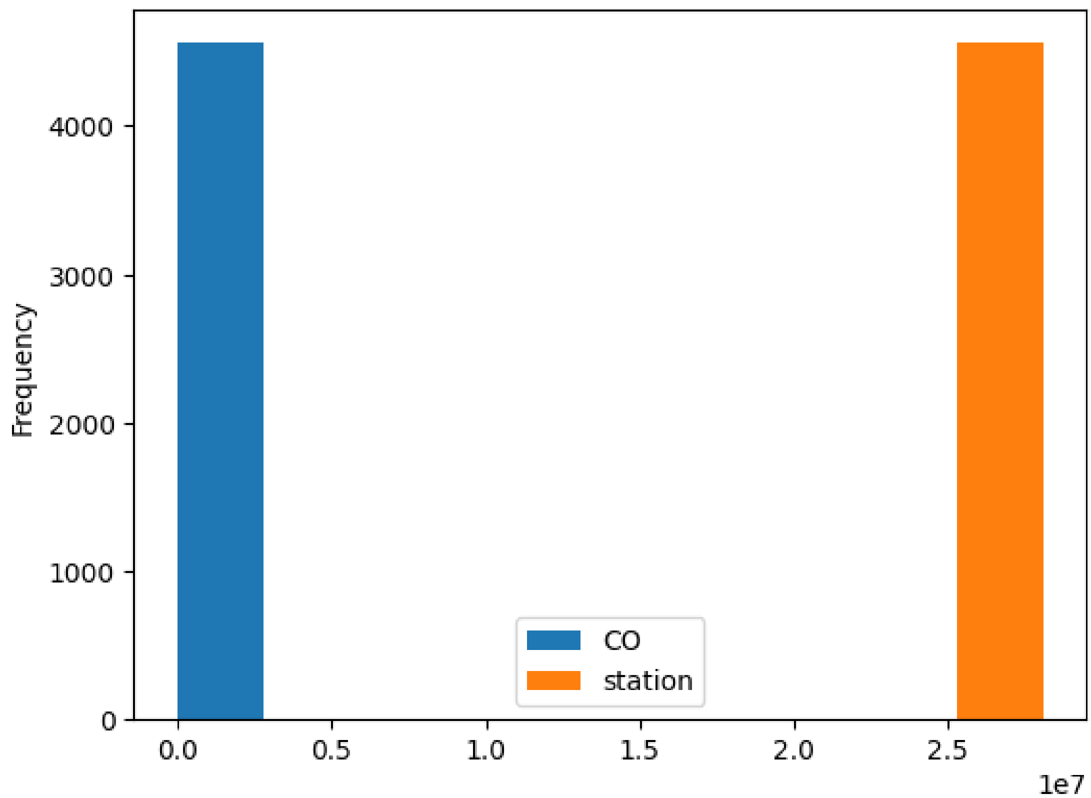
[10]: <Axes: >




## 7 Histogram

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

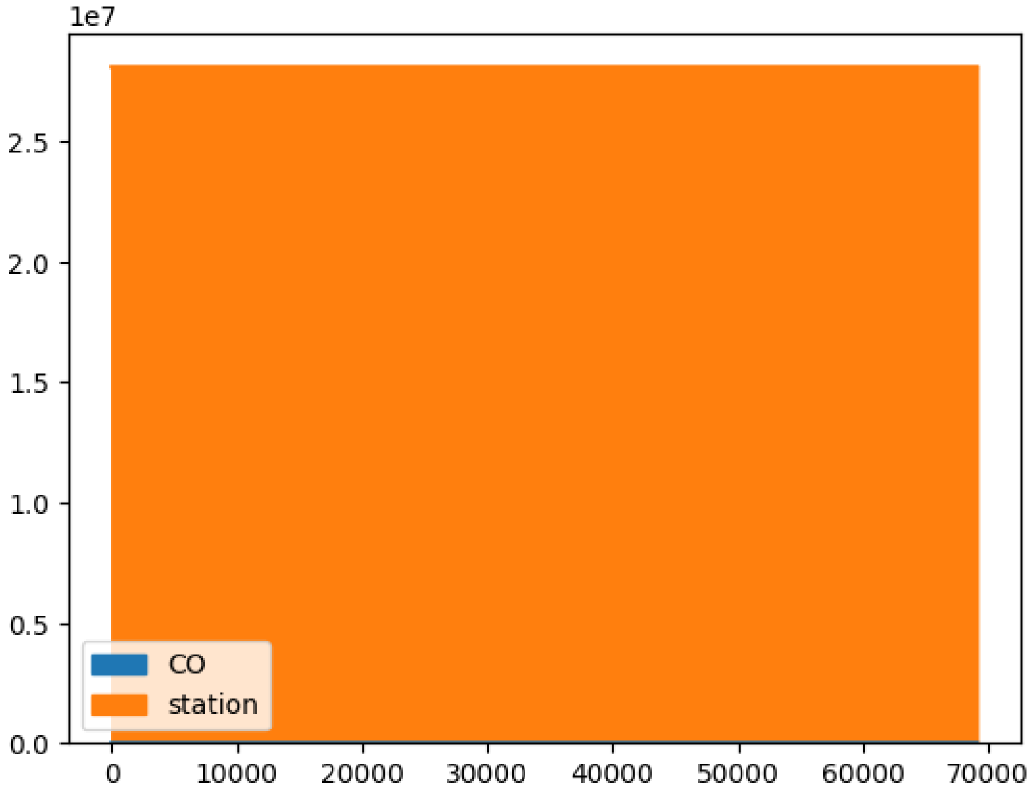
[11]: <Axes: ylabel='Frequency'>




## 8 Area chart

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

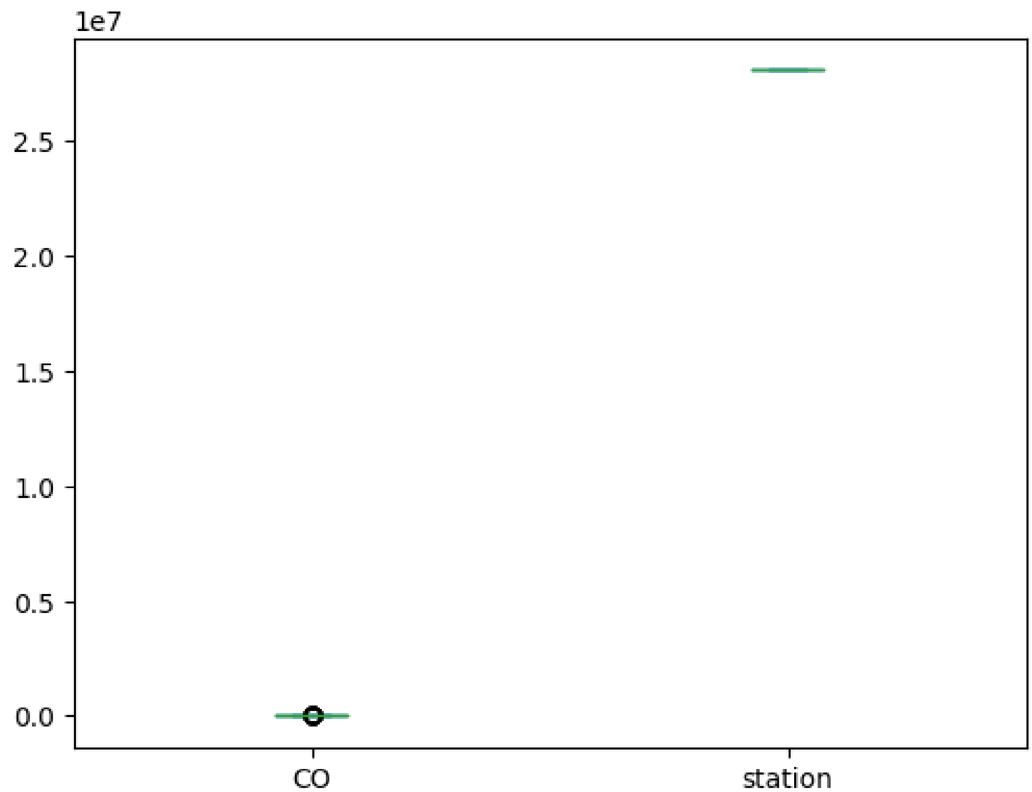
[12]: <Axes: >




## 9 Box chart

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

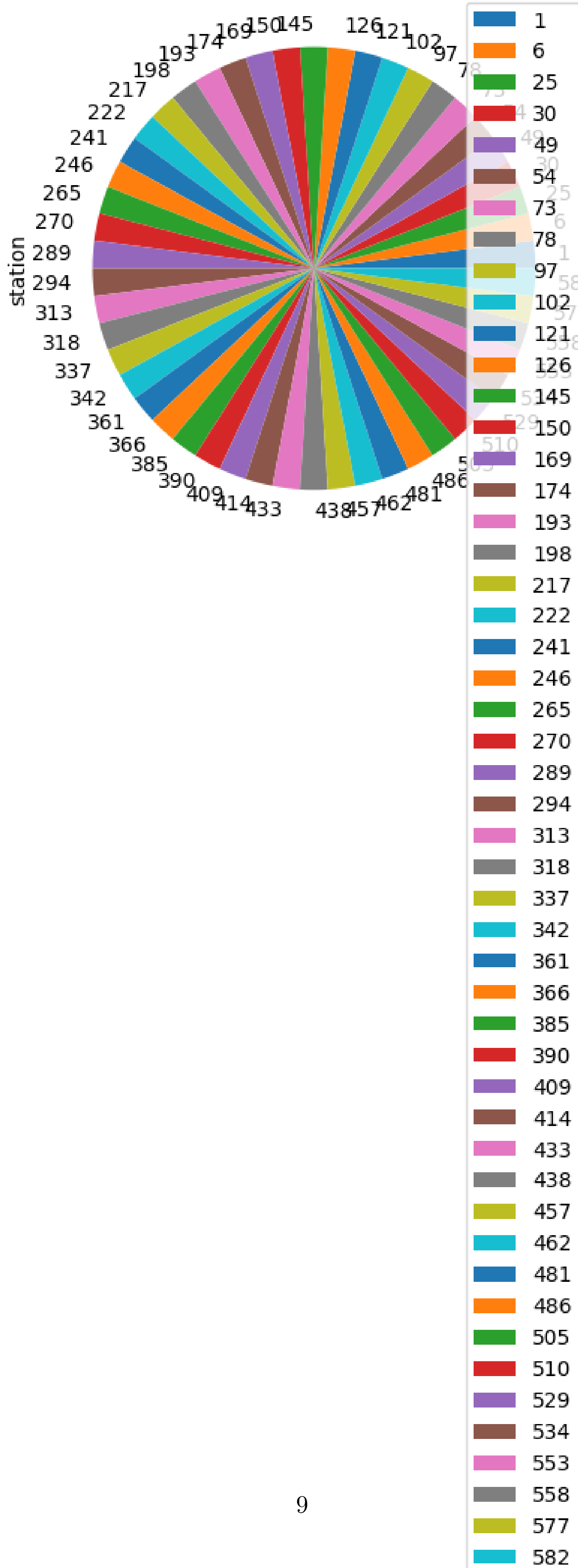
[13]: <Axes: >




## 10 Pie chart

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

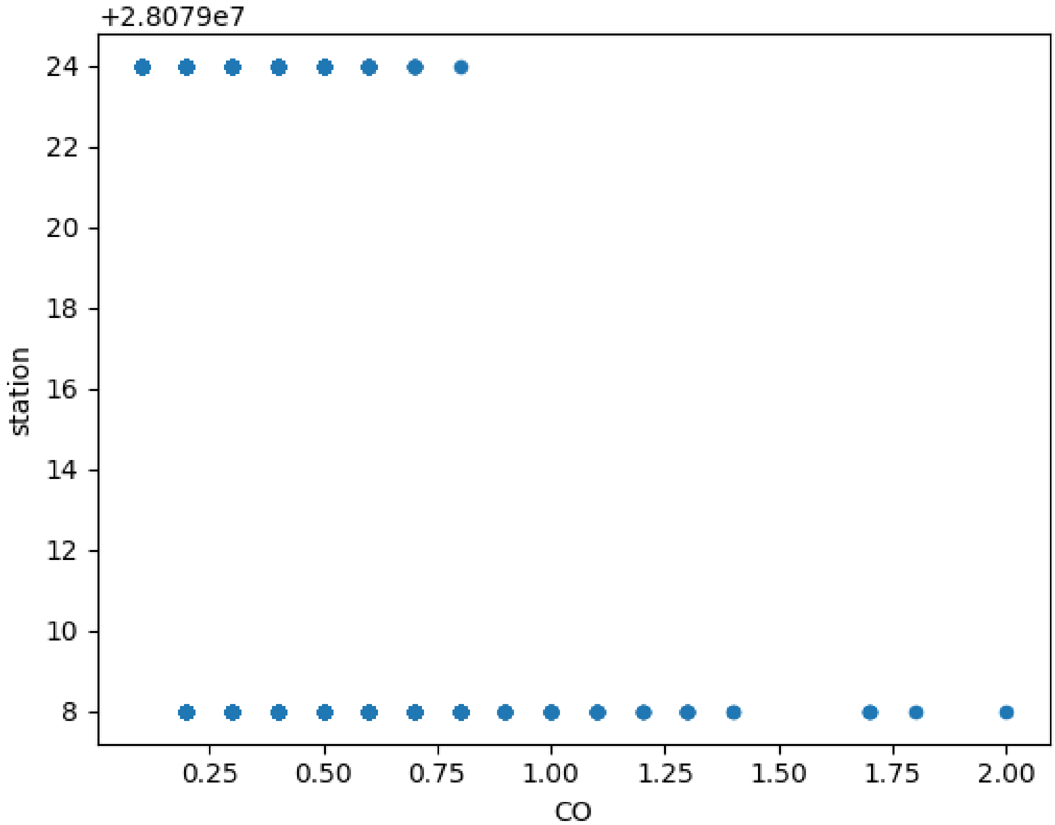
[14]: <Axes: ylabel='station'>


station
1
6
25
30
49
54
73
78
97
102
121
126
145
150
169
174
193
198
217
222
241
246
265
270
289
294
313
318
337
342
361
366
385
390
409
414
433
438
457
462
481
486
505
510
529
534
553
558
577
582

## 11 Scatter chart

[15]:
```
data.plot.scatter(x='CO' ,y='station')
```

[15]: <Axes: xlabel='CO', ylabel='station'>




[16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     4562 non-null   object
 1   BEN      4562 non-null   float64
 2   CH4      4562 non-null   float64
 3   CO       4562 non-null   float64
```

```
 4   EBE      4562 non-null   float64
 5   NMHC     4562 non-null   float64
 6   NO       4562 non-null   float64
 7   NO_2     4562 non-null   float64
 8   NOx      4562 non-null   float64
 9   O_3      4562 non-null   float64
 10  PM10     4562 non-null   float64
 11  PM25     4562 non-null   float64
 12  SO_2     4562 non-null   float64
 13  TCH      4562 non-null   float64
 14  TOL      4562 non-null   float64
 15  station  4562 non-null   int64
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

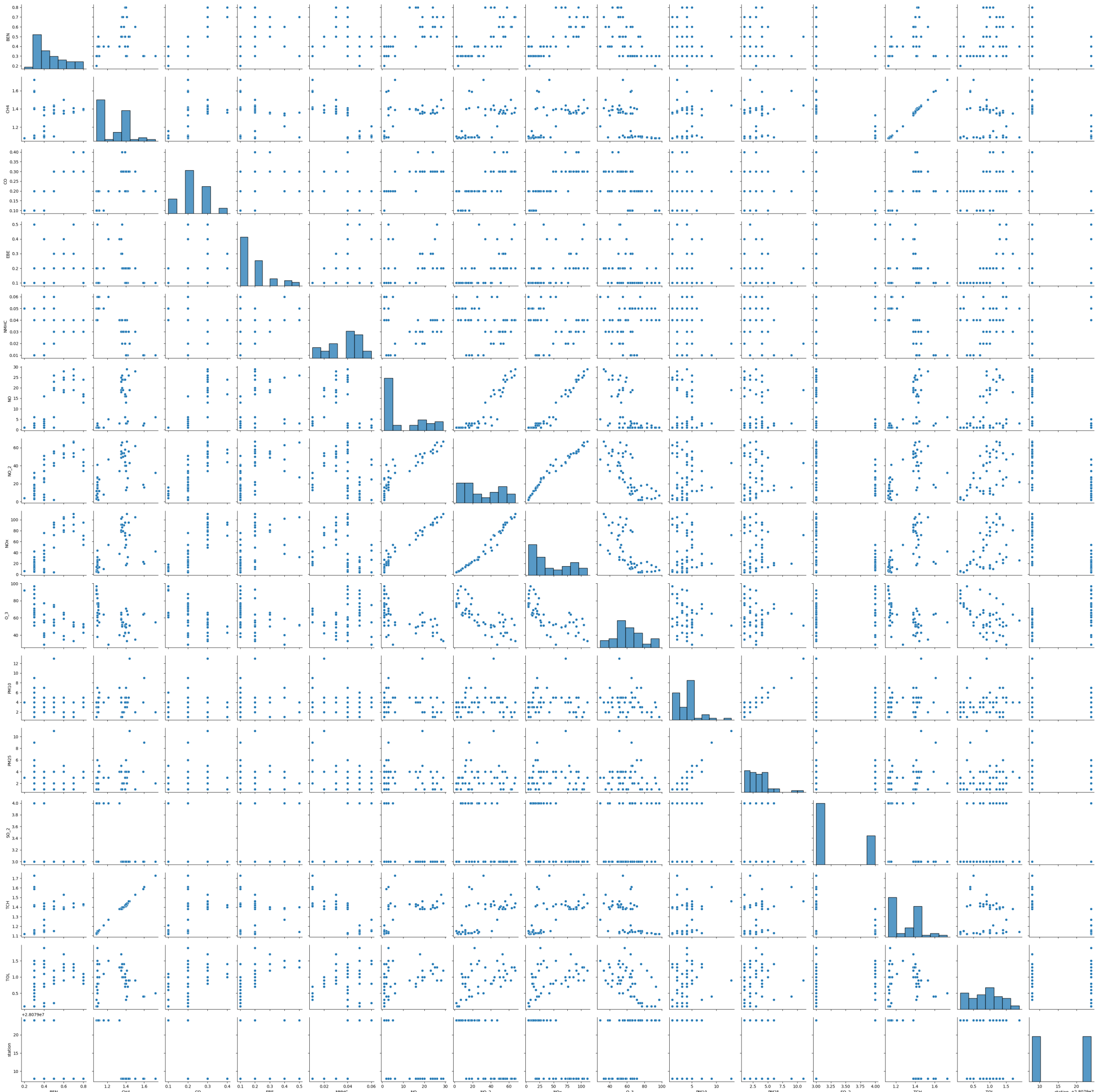
[17]:
```
df.describe()
```

[17]:
```
              BEN          CH4           CO          EBE         NMHC  \
count  4562.00000  4562.000000  4562.000000  4562.000000  4562.000000
mean      0.69349     1.329163     0.330579     0.286782     0.056773
std       0.46832     0.214399     0.161489     0.354442     0.037711
min       0.10000     0.020000     0.100000     0.100000     0.000000
25%       0.40000     1.120000     0.200000     0.100000     0.030000
50%       0.60000     1.390000     0.300000     0.200000     0.050000
75%       0.90000     1.420000     0.400000     0.300000     0.070000
max       6.60000     3.920000     2.000000     7.400000     0.490000

                NO         NO_2          NOx          O_3         PM10  \
count  4562.000000  4562.000000  4562.000000  4562.000000  4562.000000
mean     21.742218    44.152126    77.494739    41.279702    10.656291
std      35.539531    30.234015    79.218558    26.298770     8.734093
min       1.000000     1.000000     2.000000     1.000000     0.000000
25%       1.000000    20.000000    24.000000    18.000000     4.000000
50%       9.000000    41.000000    56.000000    42.000000     8.000000
75%      27.000000    64.000000   106.000000    63.000000    15.000000
max     431.000000   184.000000   844.000000   113.000000    64.000000

              PM25         SO_2          TCH          TOL       station
count  4562.000000  4562.000000  4562.000000  4562.000000  4.562000e+03
mean      7.126480     4.080447     1.385296     1.882288  2.807901e+07
std       5.965405     2.515964     0.227030     2.184735  7.829190e+00
min       0.000000     1.000000     0.030000     0.100000  2.807901e+07
25%       3.000000     3.000000     1.180000     0.500000  2.807901e+07
50%       5.000000     4.000000     1.420000     1.200000  2.807901e+07
75%      10.000000     5.000000     1.480000     2.500000  2.807902e+07
max      42.000000    22.000000     4.120000    26.700001  2.807902e+07
```

## 12 EDA AND VISUALIZATION

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

[18]: <seaborn.axisgrid.PairGrid at 0x79d3310f1b40>



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

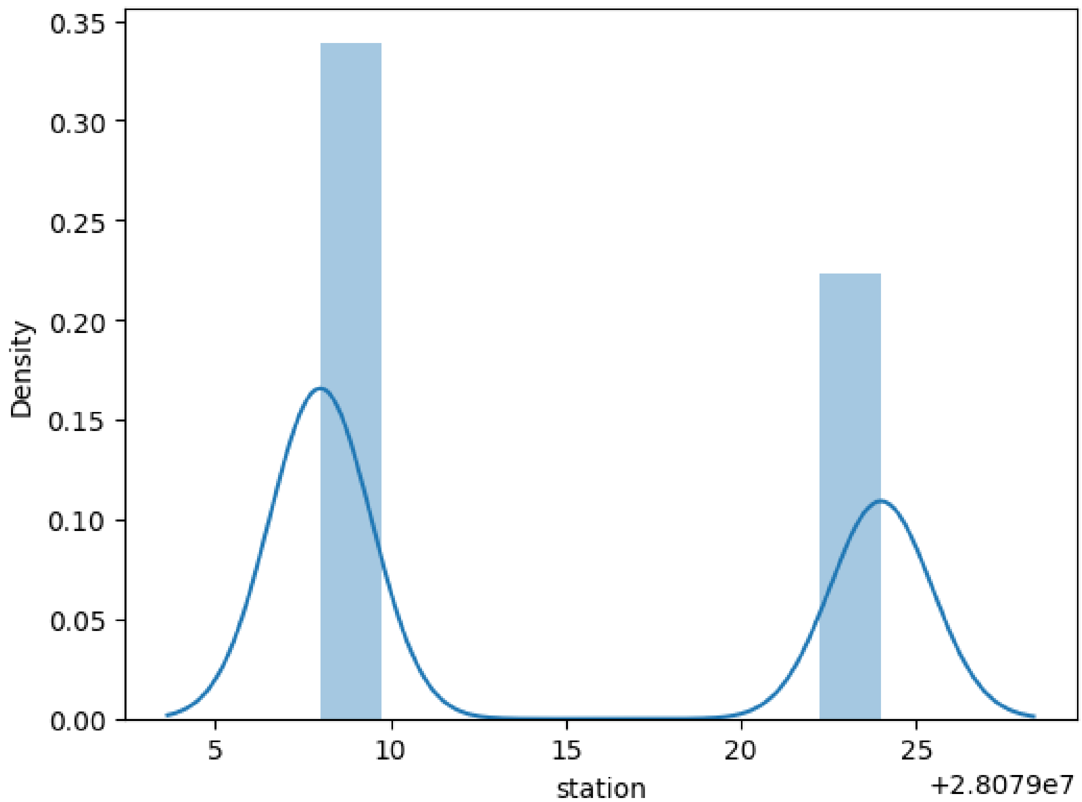
<ipython-input-19-6e2460d4583e>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['station'])
```

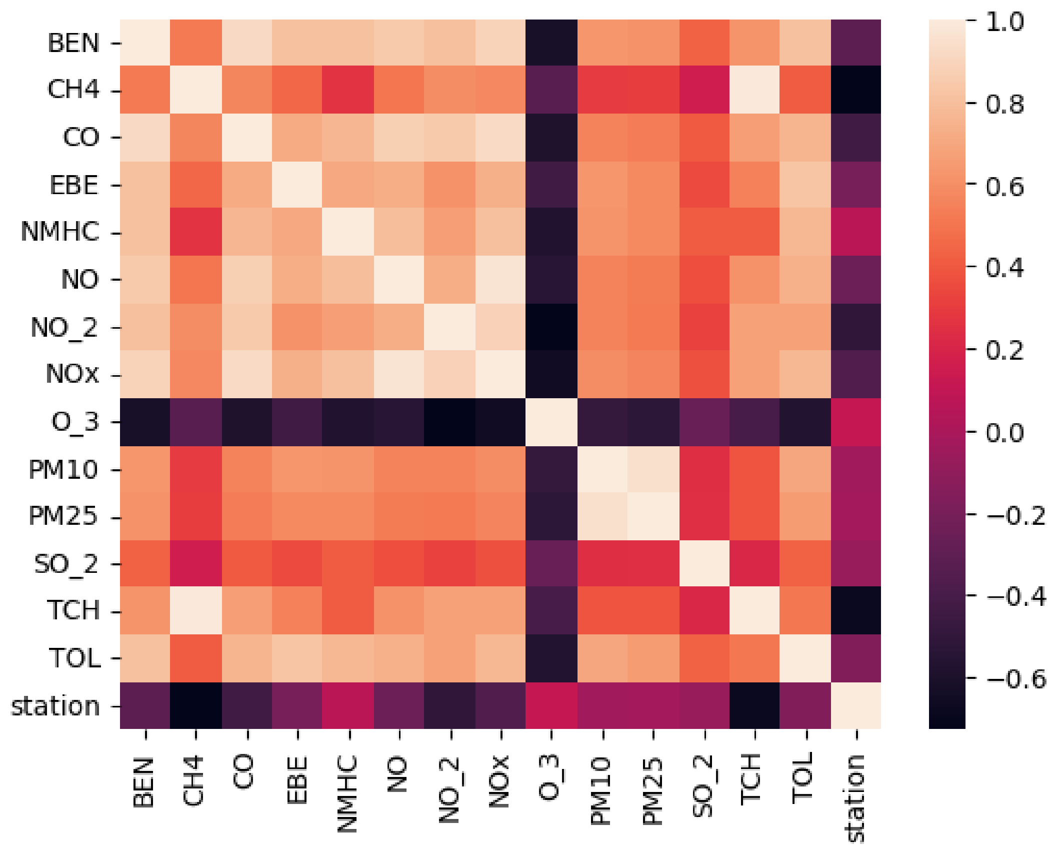
[19]: <Axes: xlabel='station', ylabel='Density'>




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

```
<ipython-input-20-aa4f4450a243>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  sns.heatmap(df.corr())
```

[20]: <Axes: >




## 13 TO TRAIN THE MODEL AND MODEL BULDING

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

[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)
```

## 14 Linear Regression

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

[23]: LinearRegression()

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

[24]: 28079045.805657078

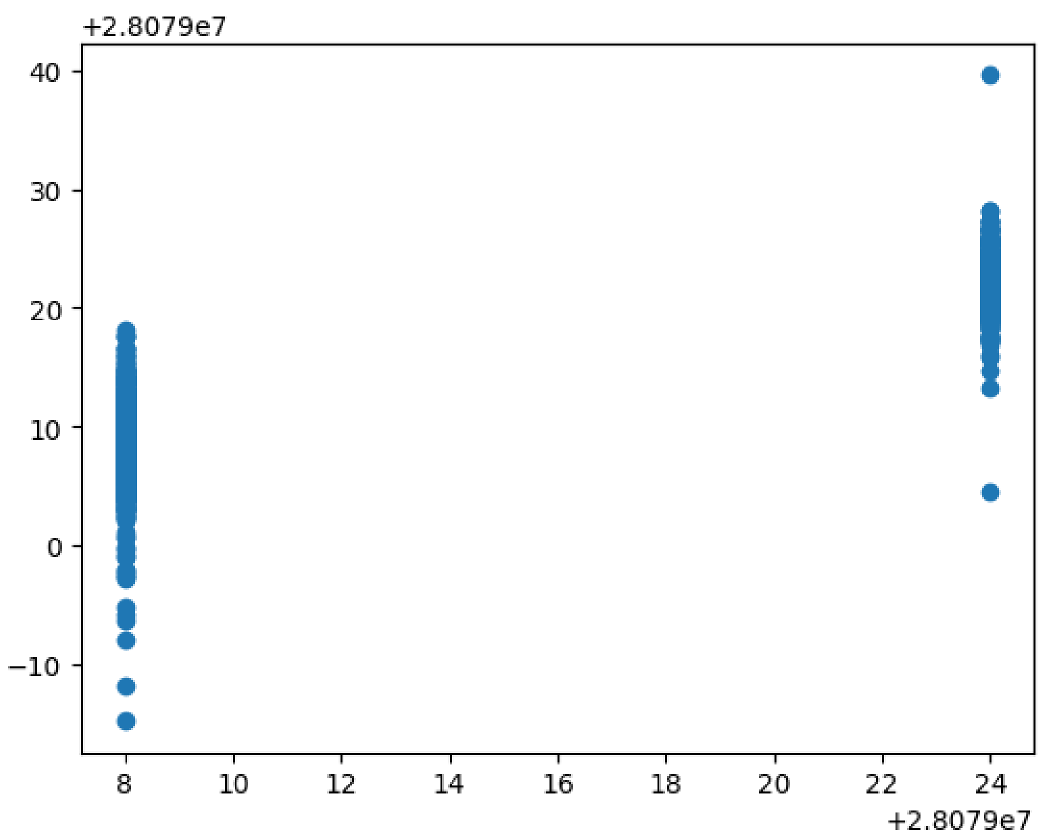
[25]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

[25]:

| | Co-efficient |
|---|---|
| BEN | -0.769274 |
| CO | -24.425017 |
| EBE | 0.782613 |
| NMHC | 143.013368 |
| NO | 0.034748 |
| NO_2 | -0.142726 |
| O_3 | -0.082258 |
| PM10 | 0.036115 |
| PM25 | 0.118408 |
| SO_2 | -0.029073 |
| TCH | -16.628788 |
| TOL | -0.179569 |

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

[26]: <matplotlib.collections.PathCollection at 0x79d31c035060>



## 15 ACCURACY

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

[27]: 0.8197532674838165

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

[28]: 0.8055332557920627

## 16 Ridge and Lasso

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

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

[30]: Ridge(alpha=10)

## 17 Accuracy(Ridge)

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

[31]: 0.715146237574983

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

[32]: 0.7045608388139539

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

[33]: Lasso(alpha=10)

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

[34]: 0.4160112646980627

## 18 Accuracy(Lasso)

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

[35]: 0.42341162759586803

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

[36]: ElasticNet()

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

[37]:
```
array([-0.        , -0.        , -0.        ,  0.        ,  0.03164662,
       -0.28757476, -0.14986394,  0.2734702 , -0.09326112,  0.04575556,
       -0.11681796,  0.        ])
```

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

[38]: 28079030.2848804

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

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

[40]: 0.4847448049621037

## 19 Evaluation Metrics

[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)))
```

```
4.762223432114017
31.163300264103892
5.5824098975356415
```

## 20 Logistic Regression

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

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

```
target_vector=df[ 'station']
```

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

[44]: (4562, 12)

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

[45]: (4562,)

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

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

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

[48]: LogisticRegression(max_iter=10000)

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

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

[28079008]

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

[51]: array([28079008, 28079024])

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

[52]: 0.9890398947829899

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

[53]: 1.0

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

[54]: array([[1.00000000e+00, 1.09190927e-22]])

## 21 Random Forest

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

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

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

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

[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)
```

[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')
```

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

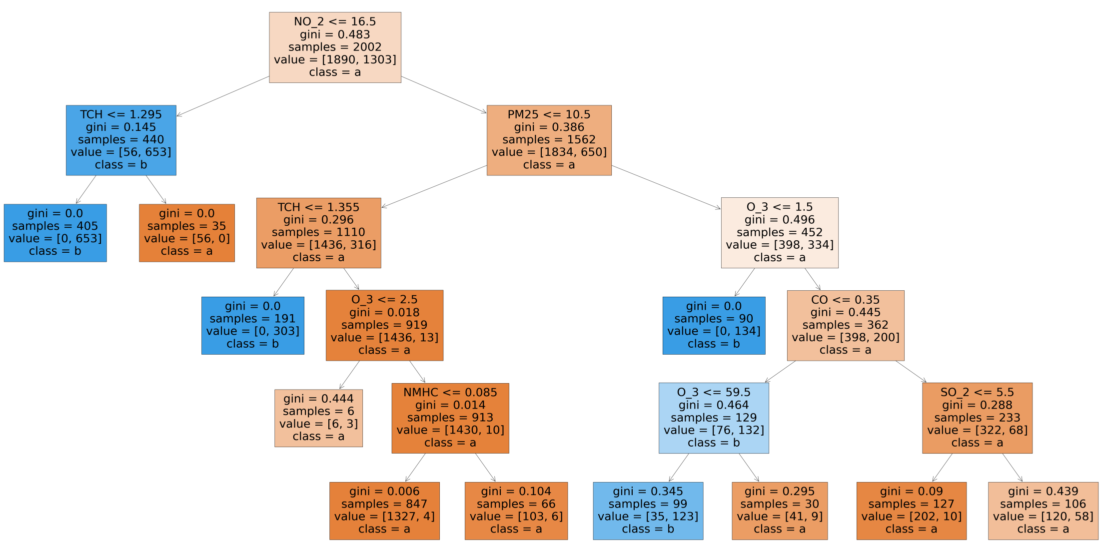
[59]:
```
0.9924833608755765
```

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

[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'],filled=True)
```

[61]:
```
[Text(0.3088235294117647, 0.9166666666666666, 'NO_2 <= 16.5\ngini =
0.483\nsamples = 2002\nvalue = [1890, 1303]\nclass = a'),
 Text(0.11764705882352941, 0.75, 'TCH <= 1.295\ngini = 0.145\nsamples =
440\nvalue = [56, 653]\nclass = b'),
 Text(0.058823529411764705, 0.5833333333333334, 'gini = 0.0\nsamples =
405\nvalue = [0, 653]\nclass = b'),
 Text(0.17647058823529413, 0.5833333333333334, 'gini = 0.0\nsamples = 35\nvalue
= [56, 0]\nclass = a'),
 Text(0.5, 0.75, 'PM25 <= 10.5\ngini = 0.386\nsamples = 1562\nvalue = [1834,
650]\nclass = a'),
 Text(0.29411764705882354, 0.5833333333333334, 'TCH <= 1.355\ngini =
0.296\nsamples = 1110\nvalue = [1436, 316]\nclass = a'),
 Text(0.23529411764705882, 0.4166666666666667, 'gini = 0.0\nsamples = 191\nvalue
= [0, 303]\nclass = b'),
```

```
 Text(0.35294117647058826, 0.4166666666666667, 'O_3 <= 2.5\ngini =
0.018\nsamples = 919\nvalue = [1436, 13]\nclass = a'),
 Text(0.29411764705882354, 0.25, 'gini = 0.444\nsamples = 6\nvalue = [6,
3]\nclass = a'),
 Text(0.4117647058823529, 0.25, 'NMHC <= 0.085\ngini = 0.014\nsamples =
913\nvalue = [1430, 10]\nclass = a'),
 Text(0.35294117647058826, 0.08333333333333333, 'gini = 0.006\nsamples =
847\nvalue = [1327, 4]\nclass = a'),
 Text(0.47058823529411764, 0.08333333333333333, 'gini = 0.104\nsamples =
66\nvalue = [103, 6]\nclass = a'),
 Text(0.7058823529411765, 0.5833333333333334, 'O_3 <= 1.5\ngini = 0.496\nsamples
= 452\nvalue = [398, 334]\nclass = a'),
 Text(0.6470588235294118, 0.4166666666666667, 'gini = 0.0\nsamples = 90\nvalue =
[0, 134]\nclass = b'),
 Text(0.7647058823529411, 0.4166666666666667, 'CO <= 0.35\ngini = 0.445\nsamples
= 362\nvalue = [398, 200]\nclass = a'),
 Text(0.6470588235294118, 0.25, 'O_3 <= 59.5\ngini = 0.464\nsamples = 129\nvalue
= [76, 132]\nclass = b'),
 Text(0.5882352941176471, 0.08333333333333333, 'gini = 0.345\nsamples =
99\nvalue = [35, 123]\nclass = b'),
 Text(0.7058823529411765, 0.08333333333333333, 'gini = 0.295\nsamples =
30\nvalue = [41, 9]\nclass = a'),
 Text(0.8823529411764706, 0.25, 'SO_2 <= 5.5\ngini = 0.288\nsamples = 233\nvalue
= [322, 68]\nclass = a'),
 Text(0.8235294117647058, 0.08333333333333333, 'gini = 0.09\nsamples =
127\nvalue = [202, 10]\nclass = a'),
 Text(0.9411764705882353, 0.08333333333333333, 'gini = 0.439\nsamples =
106\nvalue = [120, 58]\nclass = a')]
```



## 22 Conclusion

## 23 Accuracy

[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: 0.8197532674838165
Ridge Regression: 0.715146237574983
Lasso Regression 0.42341162759586803
ElasticNet Regression: 0.4847448049621037
Logistic Regression: 0.9890398947829899
Random Forest: 0.9924833608755765
```

## 24 Random Forest is suitable for this dataset