20104169 - SUMESH R

Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid
_2017.csv")
df
```

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

```
                       date  BEN  CH4   CO  EBE  NMHC   NO  NO_2  NOx
O_3  \
0       2017-06-01 01:00:00  NaN  NaN  0.3  NaN   NaN  4.0  38.0  NaN
NaN
1       2017-06-01 01:00:00  0.6  NaN  0.3  0.4  0.08  3.0  39.0  NaN
71.0
2       2017-06-01 01:00:00  0.2  NaN  NaN  0.1   NaN  1.0  14.0  NaN
NaN
3       2017-06-01 01:00:00  NaN  NaN  0.2  NaN   NaN  1.0   9.0  NaN
91.0
4       2017-06-01 01:00:00  NaN  NaN  NaN  NaN   NaN  1.0  19.0  NaN
69.0
...                     ...  ...  ...  ...  ...   ...  ...   ...  ...
...
210115  2017-08-01 00:00:00  NaN  NaN  0.2  NaN   NaN  1.0  27.0  NaN
65.0
210116  2017-08-01 00:00:00  NaN  NaN  0.2  NaN   NaN  1.0  14.0  NaN
NaN
210117  2017-08-01 00:00:00  NaN  NaN  NaN  NaN   NaN  1.0   4.0  NaN
83.0
210118  2017-08-01 00:00:00  NaN  NaN  NaN  NaN   NaN  1.0  11.0  NaN
78.0
210119  2017-08-01 00:00:00  NaN  NaN  NaN  NaN   NaN  1.0  14.0  NaN
77.0

        PM10  PM25  SO_2  TCH  TOL   station
0        NaN   NaN   5.0  NaN  NaN  28079004
1       22.0   9.0   7.0  1.4  2.9  28079008
2        NaN   NaN   NaN  NaN  0.9  28079011
```

```
3         NaN   NaN   NaN  NaN  NaN  28079016
4         NaN   NaN   2.0  NaN  NaN  28079017
...       ...   ...   ...  ...  ...       ...
210115    NaN   NaN   NaN  NaN  NaN  28079056
210116   73.0   NaN   7.0  NaN  NaN  28079057
210117    NaN   NaN   NaN  NaN  NaN  28079058
210118    NaN   NaN   NaN  NaN  NaN  28079059
210119   60.0   NaN   NaN  NaN  NaN  28079060

[210120 rows x 16 columns]
```

Data Cleaning and Data Preprocessing

```
df=df.dropna()

df.columns

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

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
```

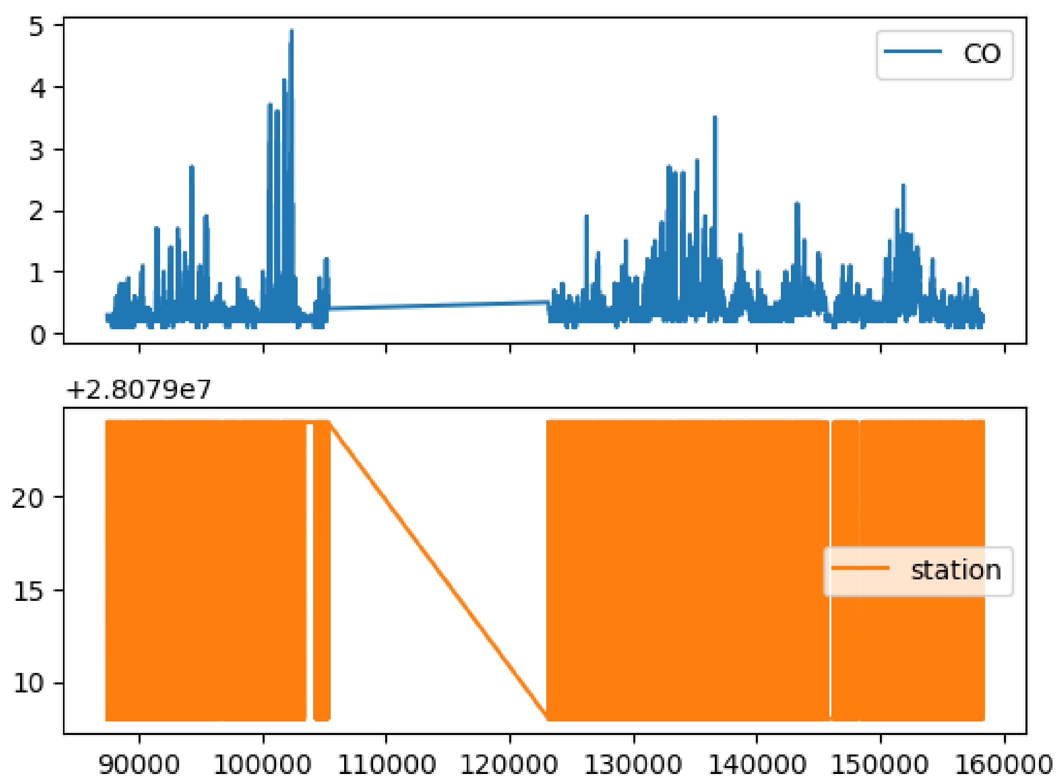
```
data=df[['CO' ,'station']]
data
```

```
          CO   station
87457    0.3  28079008
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 x 2 columns]
```

Line chart

```
data.plot.line(subplots=True)
```

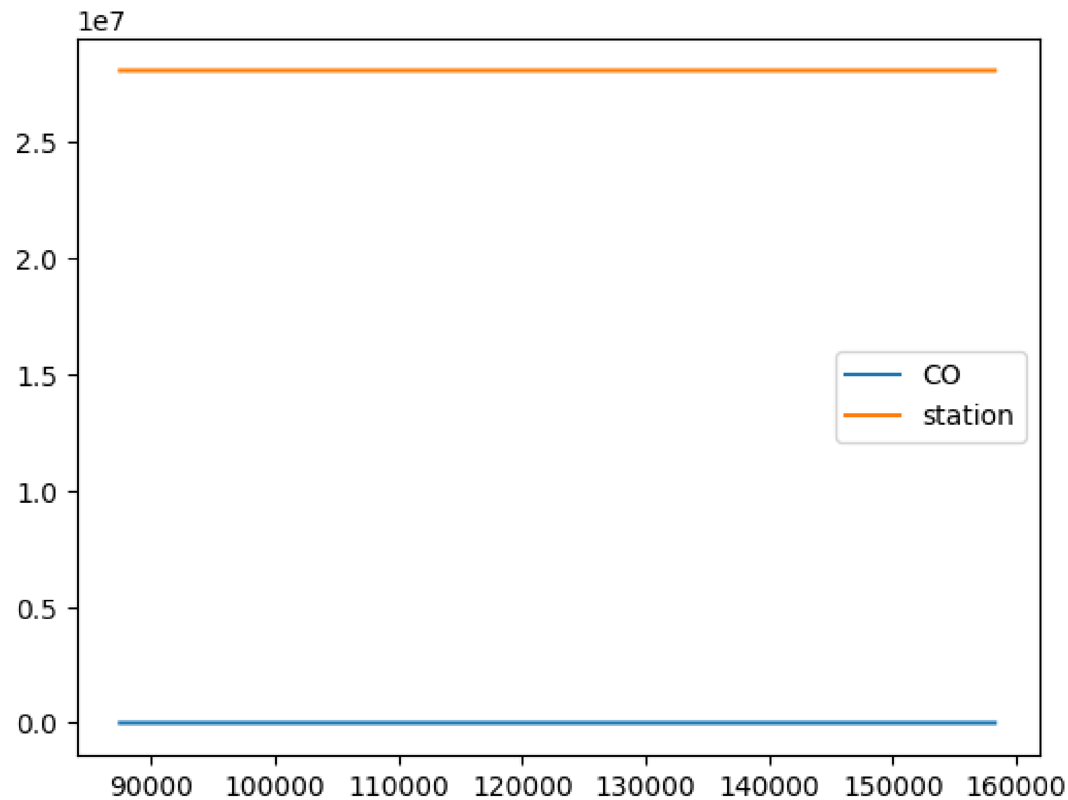
```
array([<Axes: >, <Axes: >], dtype=object)
```



Line chart

```
data.plot.line()
```

```
<Axes: >
```

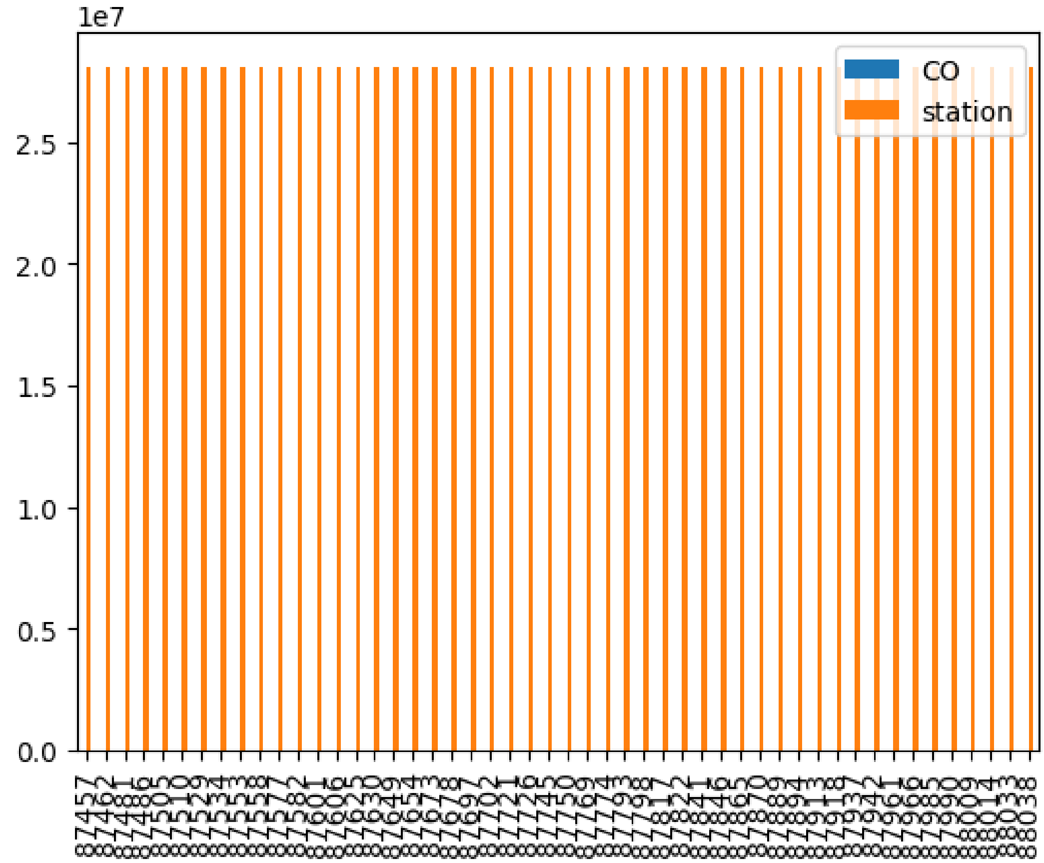


Bar chart

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

```
b.plot.bar()
```

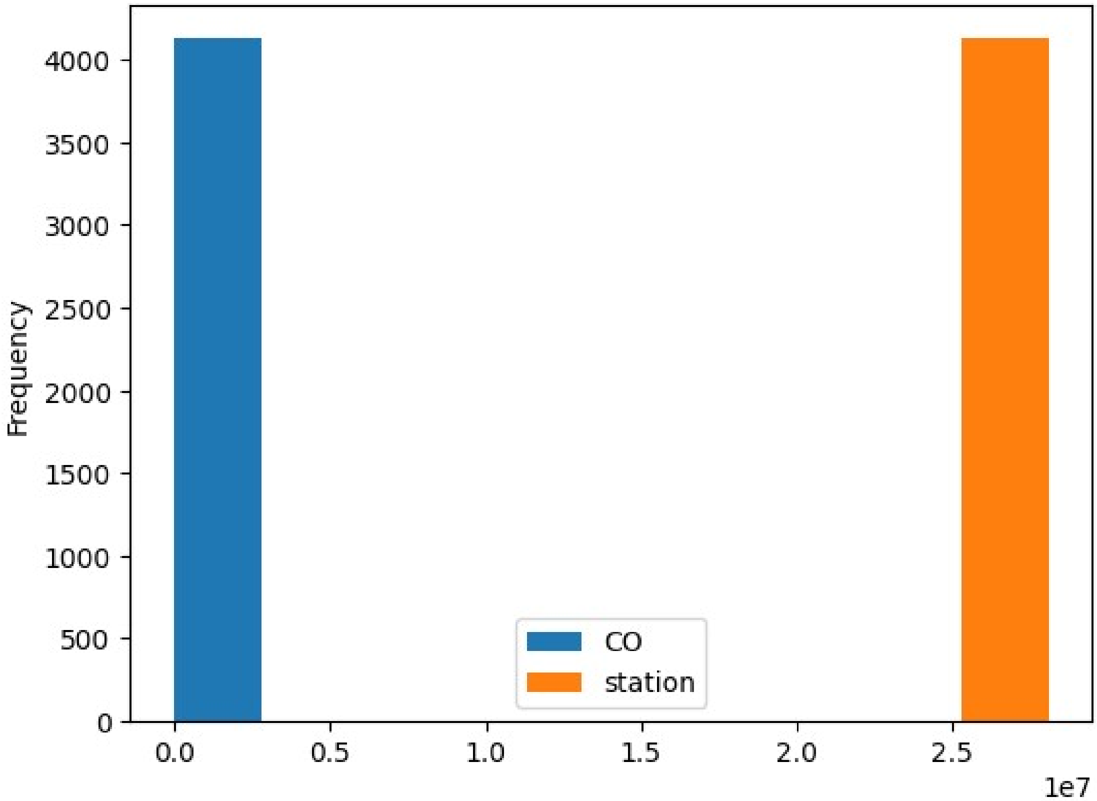
```
<Axes: >
```



Histogram

```
data.plot.hist()
```

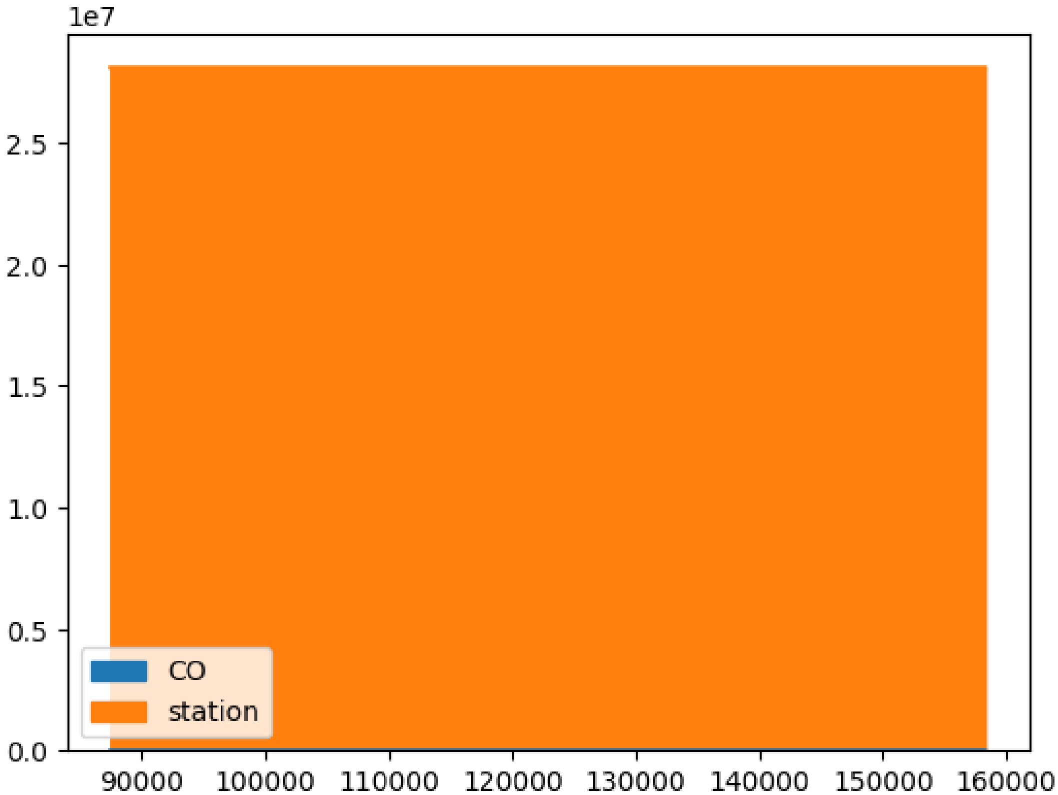
```
<Axes: ylabel='Frequency'>
```



Area chart

```
data.plot.area()
```

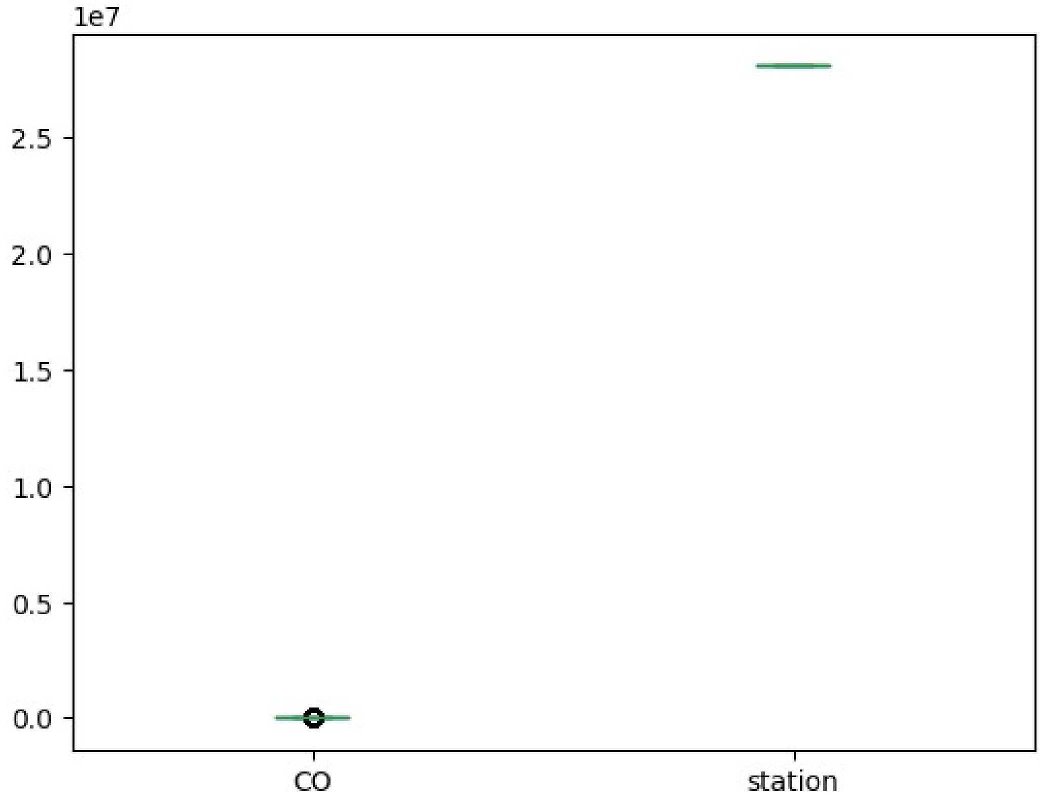
```
<Axes: >
```



Box chart

```
data.plot.box()
```

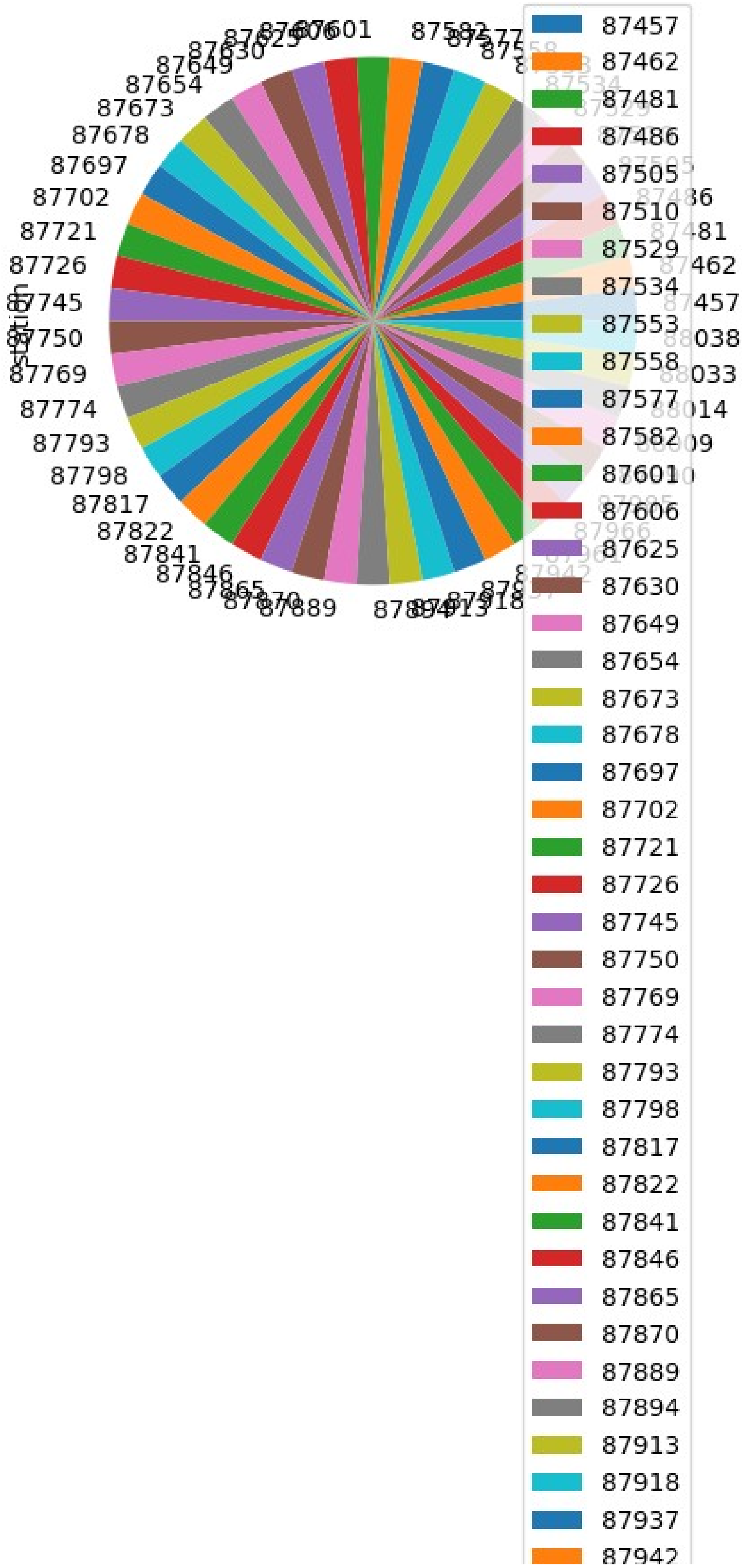
```
<Axes: >
```



Pie chart

```
b.plot.pie(y='station' )
```

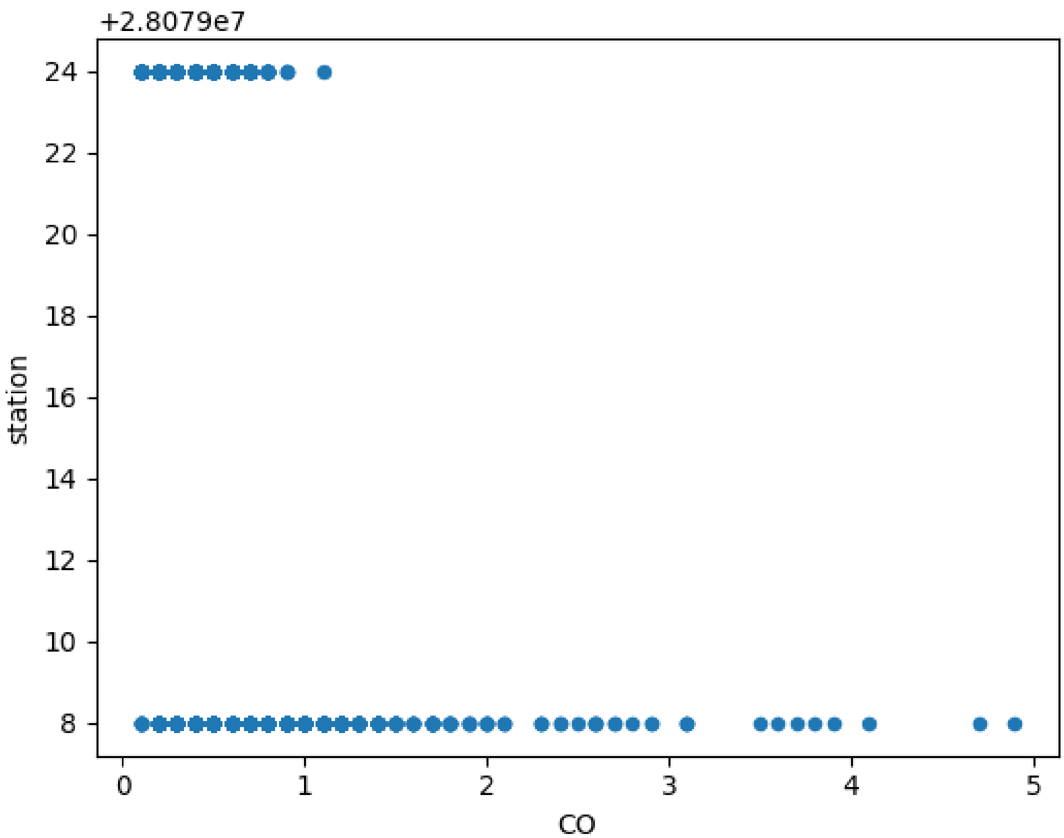
```
<Axes: ylabel='station'>
```


87457
87462
87481
87486
87505
87510
87529
87534
87553
87558
87577
87582
87601
87606
87625
87630
87649
87654
87673
87678
87697
87702
87721
87726
87745
87750
87769
87774
87793
87798
87817
87822
87841
87846
87865
87870
87889
87894
87913
87918
87937
87942

Scatter chart

```
data.plot.scatter(x='CO' ,y='station')
```

```
<Axes: xlabel='CO', ylabel='station'>
```




```
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
```

```
df.describe()
```

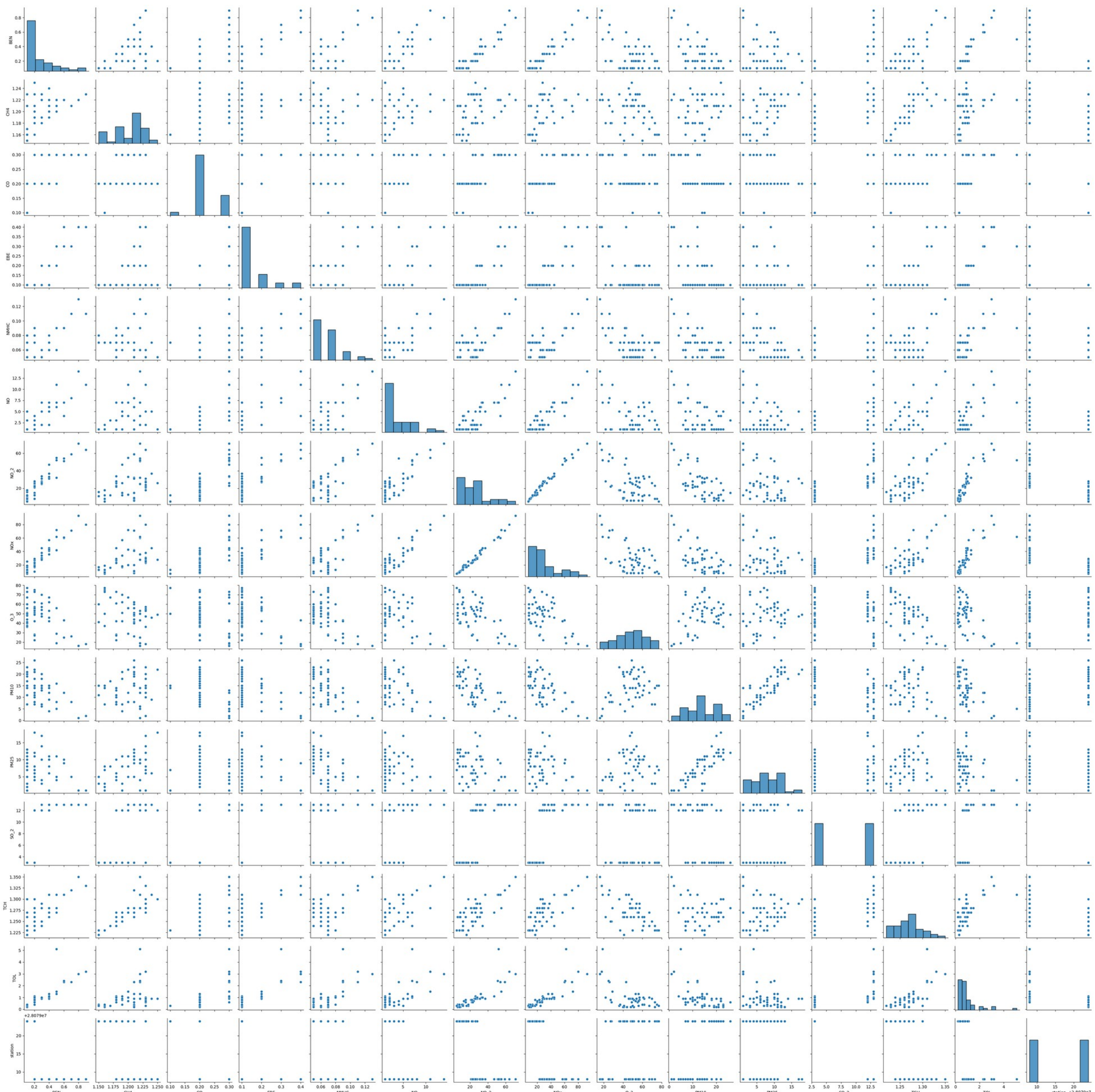
```
               BEN          CH4           CO          EBE         NMHC  \
count  4127.000000  4127.000000  4127.000000  4127.000000  4127.000000

mean      0.919918     1.323732     0.417858     0.578168     0.097269

std       1.123078     0.215742     0.342871     0.962000     0.094035

min       0.100000     1.100000     0.100000     0.100000     0.000000

25%       0.300000     1.180000     0.200000     0.100000     0.050000

50%       0.600000     1.270000     0.300000     0.300000     0.080000

75%       1.100000     1.400000     0.500000     0.700000     0.110000

max      19.600000     3.630000     4.900000    16.700001     1.420000


                NO         NO_2          NOx          O_3         PM10  \
count  4127.000000  4127.000000  4127.000000  4127.000000  4127.000000

mean     41.785316    58.069057   122.125515    28.716501    17.582021

std      71.118499    38.974112   142.828344    25.304909    12.735860

min       1.000000     1.000000     2.000000     1.000000     1.000000

25%       3.000000    30.000000    37.000000     6.000000     8.000000

50%      16.000000    54.000000    80.000000    22.000000    14.000000

75%      50.000000    78.000000   153.000000    46.000000    25.000000

max     879.000000   349.000000  1681.000000   140.000000    80.000000


              PM25         SO_2          TCH          TOL
station
```

```
count  4127.000000  4127.000000  4127.000000  4127.000000
4.127000e+03
mean     10.942816     5.689120     1.420417     4.162830
2.807902e+07
std       8.511526     3.848442     0.261857     5.689394
8.000152e+00
min       1.000000     1.000000     1.100000     0.100000
2.807901e+07
25%       5.000000     3.000000     1.260000     1.000000
2.807901e+07
50%       9.000000     4.000000     1.370000     2.500000
2.807901e+07
75%      15.000000     7.000000     1.480000     5.200000
2.807902e+07
max      58.000000    32.000000     3.700000    84.800003
2.807902e+07
```

EDA AND VISUALIZATION

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

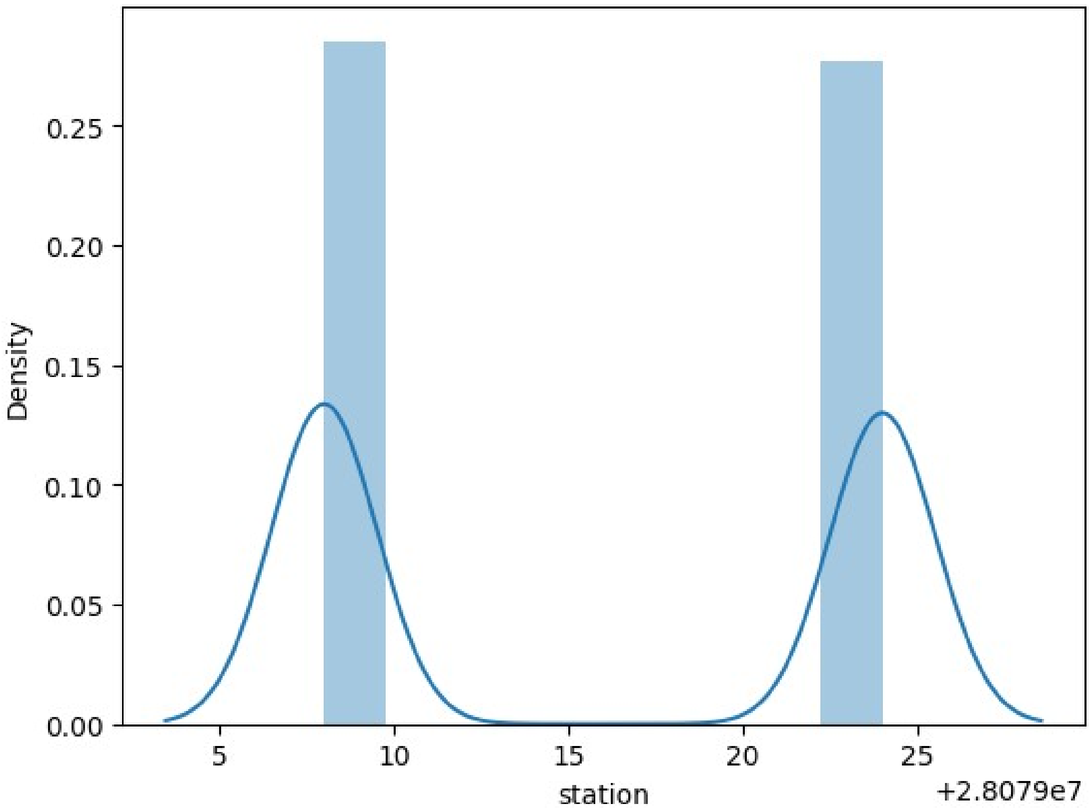
```
<seaborn.axisgrid.PairGrid at 0x7981b67df430>
```



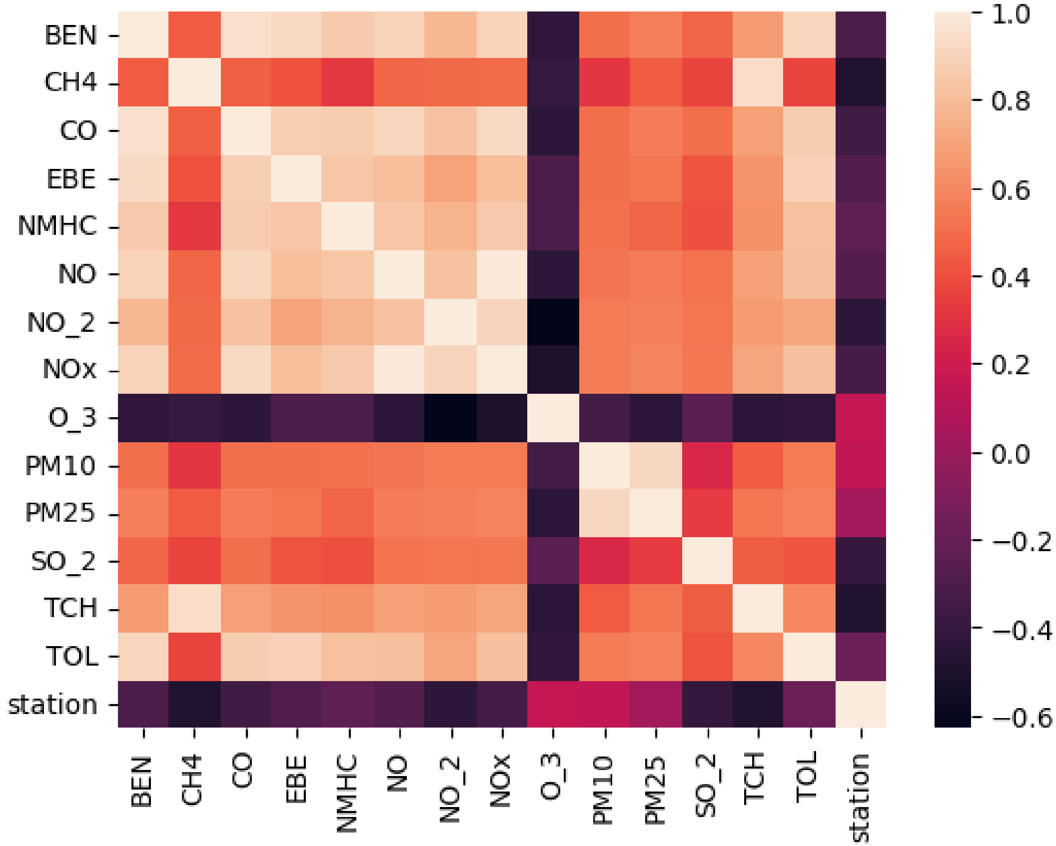
```
sns.distplot(df['station'])
```

```
<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'])

<Axes: xlabel='station', ylabel='Density'>
```




```
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())

<Axes: >
```



TO TRAIN THE MODEL AND MODEL BULDING

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

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

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
lr.intercept_
```

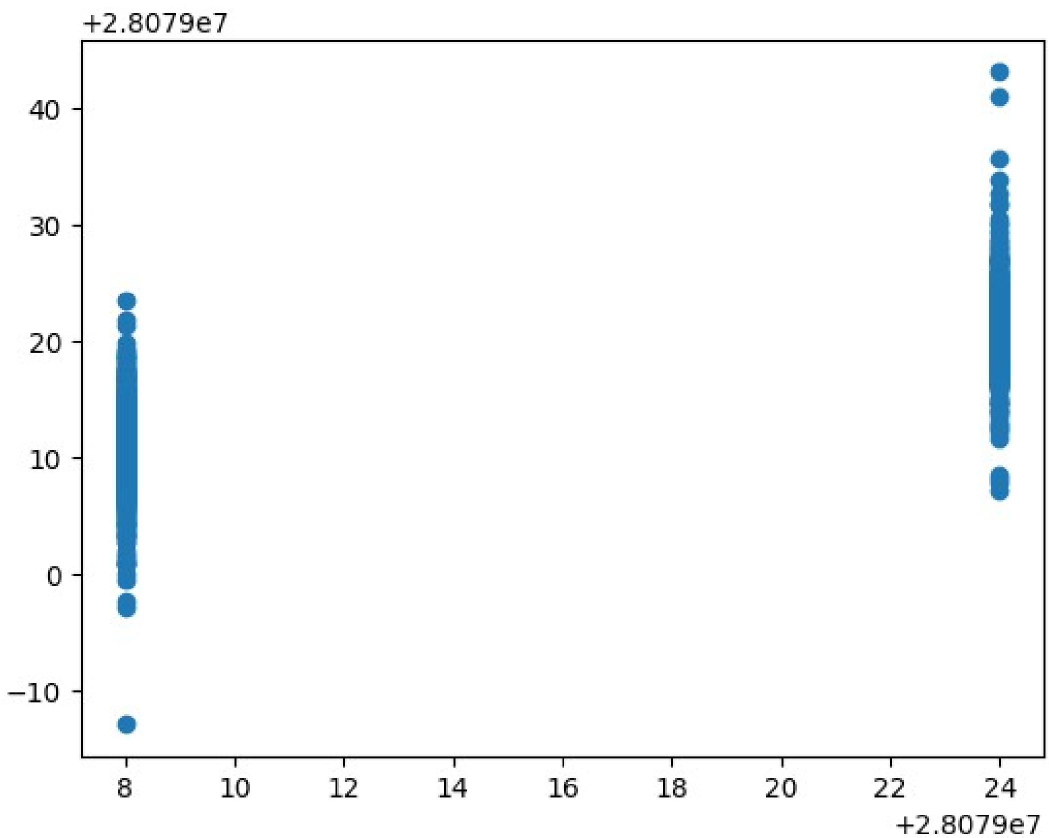
```
28079042.226563875
```

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

	Co-efficient
BEN	0.043547
CO	-5.098540
EBE	-1.826949
NMHC	22.952181
NO	0.053629
NO_2	-0.183132
O_3	-0.092828
PM10	0.456248
PM25	-0.203910
SO_2	-0.260461
TCH	-13.705300
TOL	0.200241

```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

```
<matplotlib.collections.PathCollection at 0x79819dee8cd0>
```



ACCURACY

```
lr.score(x_test,y_test)
```

```
0.6415831920868991
```

```
lr.score(x_train,y_train)
```

```
0.632157725410589
```

Ridge and Lasso

```
from sklearn.linear_model import Ridge,Lasso
```

```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
Ridge(alpha=10)
```

Accuracy(Ridge)

```
rr.score(x_test,y_test)
```

```
0.6253250346804637
```

```
rr.score(x_train,y_train)
```

```
0.6249671707210255
```

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

```
Lasso(alpha=10)
```

```
la.score(x_train,y_train)
```

```
0.40470624691546153
```

Accuracy(Lasso)

```
la.score(x_test,y_test)
```

```
0.40569400390890953
```

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

```
ElasticNet()

en.coef_

array([-0.        , -0.        , -0.        ,  0.        ,
0.03093261,
       -0.20550351, -0.08746041,  0.56446666, -0.41952604, -
0.28904518,
       -0.        ,  0.        ])

en.intercept_

28079025.37399155

prediction=en.predict(x_test)

en.score(x_test,y_test)

0.5044230795721029
```

Evaluation Metrics

```
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)))

4.814433281391835
31.697067819145467
5.63001490398964
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

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

feature_matrix.shape

(4127, 12)

target_vector.shape

(4127,)

from sklearn.preprocessing import StandardScaler
```

```
fs=StandardScaler().fit_transform(feature_matrix)

logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)

LogisticRegression(max_iter=10000)

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

prediction=logr.predict(observation)
print(prediction)

[28079008]

logr.classes_

array([28079008, 28079024])

logr.score(fs,target_vector)

0.9520232614489944

logr.predict_proba(observation)[0][0]

0.9999999999999389

logr.predict_proba(observation)

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

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)

RandomForestClassifier()

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

from sklearn.model_selection import GridSearchCV
grid_search
=GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="accura
cy")
grid_search.fit(x_train,y_train)
```

```
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')
```

```
grid_search.best_score_
```

```
0.9719529085872576
```

```
rfc_best=grid_search.best_estimator_
```

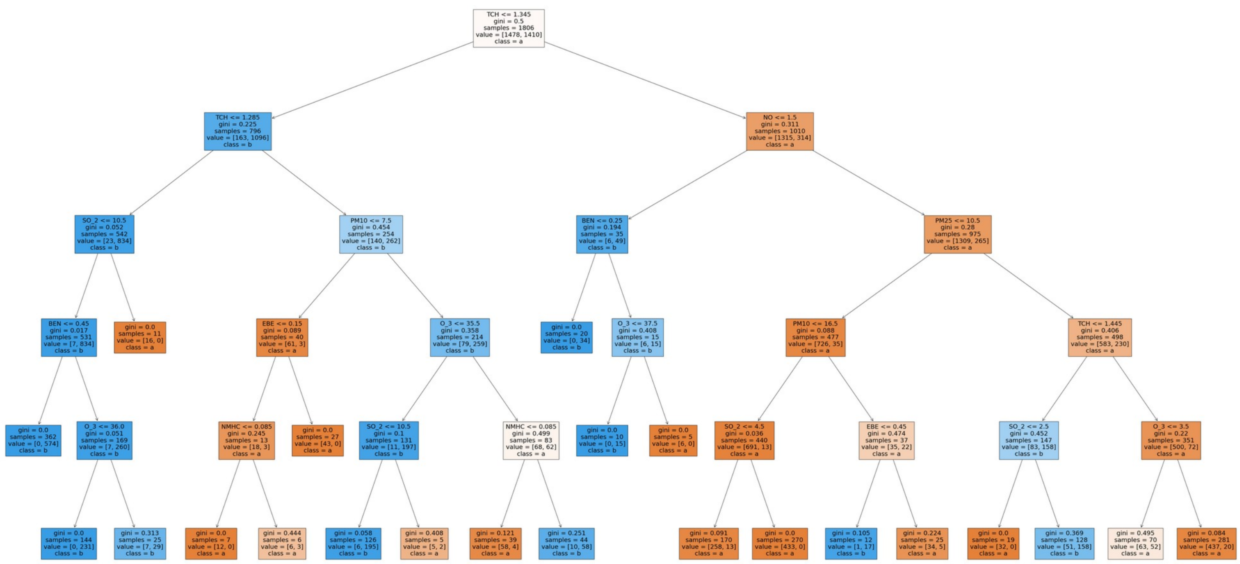
```
from sklearn.tree import plot_tree
```

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

```
[Text(0.4107142857142857, 0.9166666666666666, 'TCH <= 1.345\ngini =
0.5\nsamples = 1806\nvalue = [1478, 1410]\nclass = a'),
 Text(0.19285714285714287, 0.75, 'TCH <= 1.285\ngini = 0.225\nsamples
= 796\nvalue = [163, 1096]\nclass = b'),
 Text(0.08571428571428572, 0.5833333333333334, 'SO_2 <= 10.5\ngini =
0.052\nsamples = 542\nvalue = [23, 834]\nclass = b'),
 Text(0.05714285714285714, 0.4166666666666667, 'BEN <= 0.45\ngini =
0.017\nsamples = 531\nvalue = [7, 834]\nclass = b'),
 Text(0.02857142857142857, 0.25, 'gini = 0.0\nsamples = 362\nvalue =
[0, 574]\nclass = b'),
 Text(0.08571428571428572, 0.25, 'O_3 <= 36.0\ngini = 0.051\nsamples =
169\nvalue = [7, 260]\nclass = b'),
 Text(0.05714285714285714, 0.08333333333333333, 'gini = 0.0\nsamples =
144\nvalue = [0, 231]\nclass = b'),
 Text(0.11428571428571428, 0.08333333333333333, 'gini = 0.313\nsamples
= 25\nvalue = [7, 29]\nclass = b'),
 Text(0.11428571428571428, 0.4166666666666667, 'gini = 0.0\nsamples =
11\nvalue = [16, 0]\nclass = a'),
 Text(0.3, 0.5833333333333334, 'PM10 <= 7.5\ngini = 0.454\nsamples =
254\nvalue = [140, 262]\nclass = b'),
 Text(0.22857142857142856, 0.4166666666666667, 'EBE <= 0.15\ngini =
0.089\nsamples = 40\nvalue = [61, 3]\nclass = a'),
 Text(0.2, 0.25, 'NMHC <= 0.085\ngini = 0.245\nsamples = 13\nvalue =
[18, 3]\nclass = a'),
 Text(0.17142857142857143, 0.08333333333333333, 'gini = 0.0\nsamples =
7\nvalue = [12, 0]\nclass = a'),
 Text(0.22857142857142856, 0.08333333333333333, 'gini = 0.444\nsamples
= 6\nvalue = [6, 3]\nclass = a'),
 Text(0.2571428571428571, 0.25, 'gini = 0.0\nsamples = 27\nvalue =
[43, 0]\nclass = a'),
 Text(0.37142857142857144, 0.4166666666666667, 'O_3 <= 35.5\ngini =
0.358\nsamples = 214\nvalue = [79, 259]\nclass = b'),
```

```
 Text(0.3142857142857143, 0.25, 'SO_2 <= 10.5\ngini = 0.1\nsamples =
131\nvalue = [11, 197]\nclass = b'),
 Text(0.2857142857142857, 0.08333333333333333, 'gini = 0.058\nsamples
= 126\nvalue = [6, 195]\nclass = b'),
 Text(0.34285714285714286, 0.08333333333333333, 'gini = 0.408\nsamples
= 5\nvalue = [5, 2]\nclass = a'),
 Text(0.42857142857142855, 0.25, 'NMHC <= 0.085\ngini = 0.499\nsamples
= 83\nvalue = [68, 62]\nclass = a'),
 Text(0.4, 0.08333333333333333, 'gini = 0.121\nsamples = 39\nvalue =
[58, 4]\nclass = a'),
 Text(0.45714285714285713, 0.08333333333333333, 'gini = 0.251\nsamples
= 44\nvalue = [10, 58]\nclass = b'),
 Text(0.6285714285714286, 0.75, 'NO <= 1.5\ngini = 0.311\nsamples =
1010\nvalue = [1315, 314]\nclass = a'),
 Text(0.4857142857142857, 0.5833333333333334, 'BEN <= 0.25\ngini =
0.194\nsamples = 35\nvalue = [6, 49]\nclass = b'),
 Text(0.45714285714285713, 0.4166666666666667, 'gini = 0.0\nsamples =
20\nvalue = [0, 34]\nclass = b'),
 Text(0.5142857142857142, 0.4166666666666667, 'O_3 <= 37.5\ngini =
0.408\nsamples = 15\nvalue = [6, 15]\nclass = b'),
 Text(0.4857142857142857, 0.25, 'gini = 0.0\nsamples = 10\nvalue = [0,
15]\nclass = b'),
 Text(0.5428571428571428, 0.25, 'gini = 0.0\nsamples = 5\nvalue = [6,
0]\nclass = a'),
 Text(0.7714285714285715, 0.5833333333333334, 'PM25 <= 10.5\ngini =
0.28\nsamples = 975\nvalue = [1309, 265]\nclass = a'),
 Text(0.6571428571428571, 0.4166666666666667, 'PM10 <= 16.5\ngini =
0.088\nsamples = 477\nvalue = [726, 35]\nclass = a'),
 Text(0.6, 0.25, 'SO_2 <= 4.5\ngini = 0.036\nsamples = 440\nvalue =
[691, 13]\nclass = a'),
 Text(0.5714285714285714, 0.08333333333333333, 'gini = 0.091\nsamples
= 170\nvalue = [258, 13]\nclass = a'),
 Text(0.6285714285714286, 0.08333333333333333, 'gini = 0.0\nsamples =
270\nvalue = [433, 0]\nclass = a'),
 Text(0.7142857142857143, 0.25, 'EBE <= 0.45\ngini = 0.474\nsamples =
37\nvalue = [35, 22]\nclass = a'),
 Text(0.6857142857142857, 0.08333333333333333, 'gini = 0.105\nsamples
= 12\nvalue = [1, 17]\nclass = b'),
 Text(0.7428571428571429, 0.08333333333333333, 'gini = 0.224\nsamples
= 25\nvalue = [34, 5]\nclass = a'),
 Text(0.8857142857142857, 0.4166666666666667, 'TCH <= 1.445\ngini =
0.406\nsamples = 498\nvalue = [583, 230]\nclass = a'),
 Text(0.8285714285714286, 0.25, 'SO_2 <= 2.5\ngini = 0.452\nsamples =
147\nvalue = [83, 158]\nclass = b'),
 Text(0.8, 0.08333333333333333, 'gini = 0.0\nsamples = 19\nvalue =
[32, 0]\nclass = a'),
 Text(0.8571428571428571, 0.08333333333333333, 'gini = 0.369\nsamples
= 128\nvalue = [51, 158]\nclass = b'),
 Text(0.9428571428571428, 0.25, 'O_3 <= 3.5\ngini = 0.22\nsamples =
```

```
351\nvalue = [500, 72]\nclass = a'),
 Text(0.9142857142857143, 0.08333333333333333, 'gini = 0.495\nsamples
= 70\nvalue = [63, 52]\nclass = a'),
 Text(0.9714285714285714, 0.08333333333333333, 'gini = 0.084\nsamples
= 281\nvalue = [437, 20]\nclass = a')]
```



Conclusion

Accuracy

```
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: 0.6415831920868991
Ridge Regression: 0.6253250346804637
Lasso Regression 0.40569400390890953
ElasticNet Regression: 0.5044230795721029
Logistic Regression: 0.9520232614489944
Random Forest: 0.9719529085872576
```

Random Forest is suitable for this dataset