# 20104169 - SUMESH R

In [1]:

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

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv("/content/drive/MyDrive/mydatasets/csvs_per_year/madrid_2002.csv")
df
```

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

Out[2]:

| | date | BEN | CO | EBE | MXY | NMHC | NO_2 | NOx | OXY | O_3 | PM10 | PXY | SO_2 | TCH | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2002-04-01 01:00:00 | NaN | 1.39 | NaN | NaN | NaN | 145.100006 | 352.100006 | NaN | 6.54 | 41.990002 | NaN | 21.320000 | NaN | |
| 1 | 2002-04-01 01:00:00 | 1.93 | 0.71 | 2.33 | 6.20 | 0.15 | 98.150002 | 153.399994 | 2.67 | 6.85 | 20.980000 | 2.53 | 11.660000 | 1.82 | 10.9 |
| 2 | 2002-04-01 01:00:00 | NaN | 0.80 | NaN | NaN | NaN | 103.699997 | 134.000000 | NaN | 13.01 | 28.440001 | NaN | 13.670000 | NaN | |
| 3 | 2002-04-01 01:00:00 | NaN | 1.61 | NaN | NaN | NaN | 97.599998 | 268.000000 | NaN | 5.12 | 42.180000 | NaN | 16.990000 | NaN | |
| 4 | 2002-04-01 01:00:00 | NaN | 1.90 | NaN | NaN | NaN | 92.089996 | 237.199997 | NaN | 7.28 | 76.330002 | NaN | 15.260000 | NaN | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 217291 | 2002-11-01 00:00:00 | 4.16 | 1.14 | NaN | NaN | NaN | 81.080002 | 265.700012 | NaN | 7.21 | 36.750000 | NaN | 13.210000 | NaN | |
| 217292 | 2002-11-01 00:00:00 | 3.67 | 1.73 | 2.89 | NaN | 0.38 | 113.900002 | 373.100006 | NaN | 5.66 | 63.389999 | NaN | 15.640000 | 1.78 | 15.6 |
| 217293 | 2002-11-01 00:00:00 | 1.37 | 0.58 | 1.17 | 2.37 | 0.15 | 65.389999 | 107.699997 | 1.30 | 9.11 | 9.640000 | 0.94 | 5.620000 | 1.43 | 4.3 |
| 217294 | 2002-11-01 00:00:00 | 4.51 | 0.91 | 4.83 | 10.99 | NaN | 149.800003 | 202.199997 | 1.00 | 5.75 | NaN | 5.52 | 24.219999 | NaN | 22.1 |
| 217295 | 2002-11-01 00:00:00 | 3.11 | 1.17 | 3.00 | 7.77 | 0.26 | 80.110001 | 180.300003 | 2.25 | 7.38 | 29.240000 | 3.35 | 12.910000 | 1.54 | 15.5 |

217296 rows × 16 columns

# Data Cleaning and Data Preprocessing

In [3]:

```
df=df.dropna()
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     32381 non-null  object
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

In [6]:

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

Out[6]:

| | CO | station |
|---|---|---|
| **1** | 0.71 | 28079035 |
| **5** | 0.72 | 28079006 |
| **22** | 0.80 | 28079024 |
| **24** | 1.04 | 28079099 |
| **26** | 0.53 | 28079035 |
| **...** | ... | ... |
| **217269** | 0.28 | 28079024 |
| **217271** | 1.30 | 28079099 |
| **217273** | 0.97 | 28079035 |
| **217293** | 0.58 | 28079024 |
| **217295** | 1.17 | 28079099 |

32381 rows × 2 columns

# Line chart

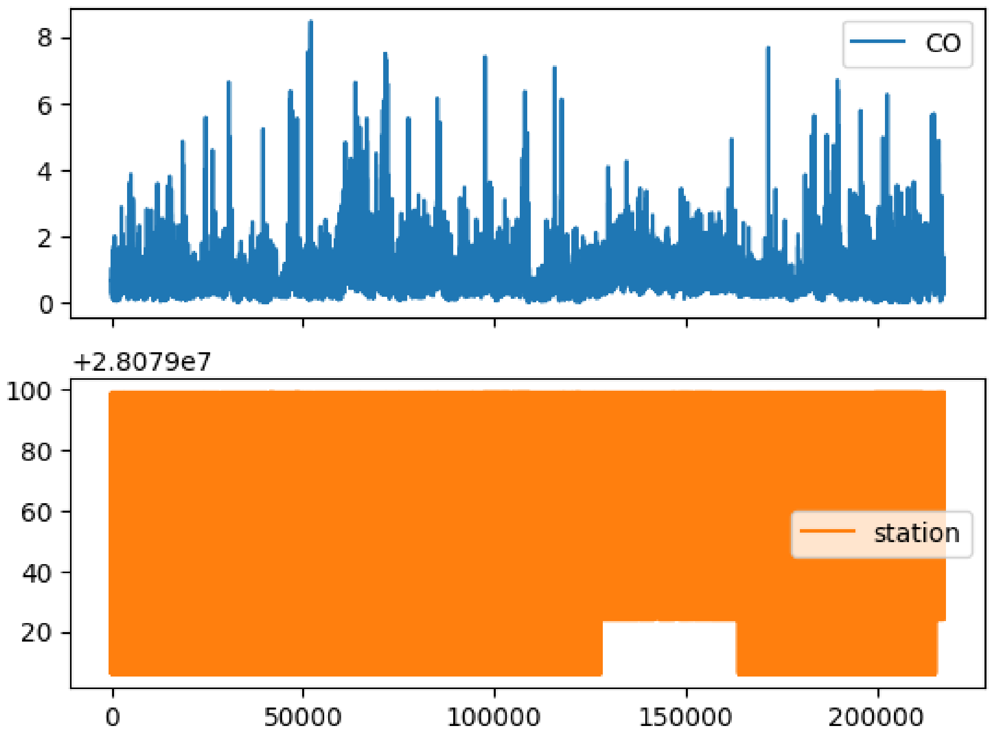
## Line chart

In [7]:

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

Out[7]:

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



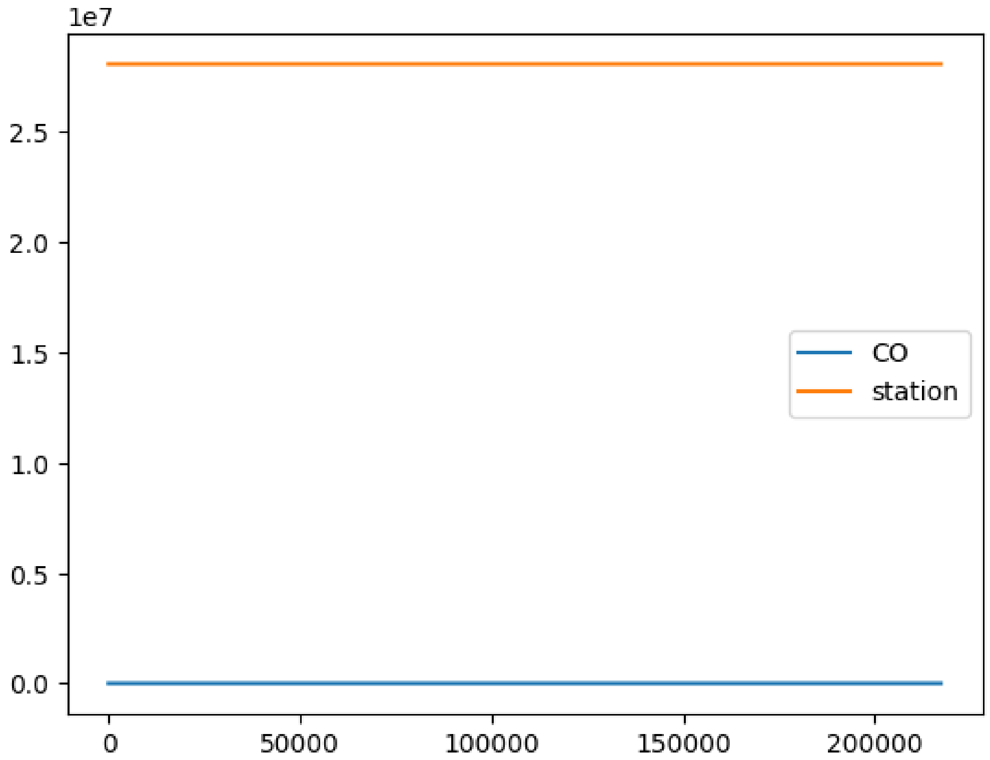

## Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



# Bar chart

In [9]:

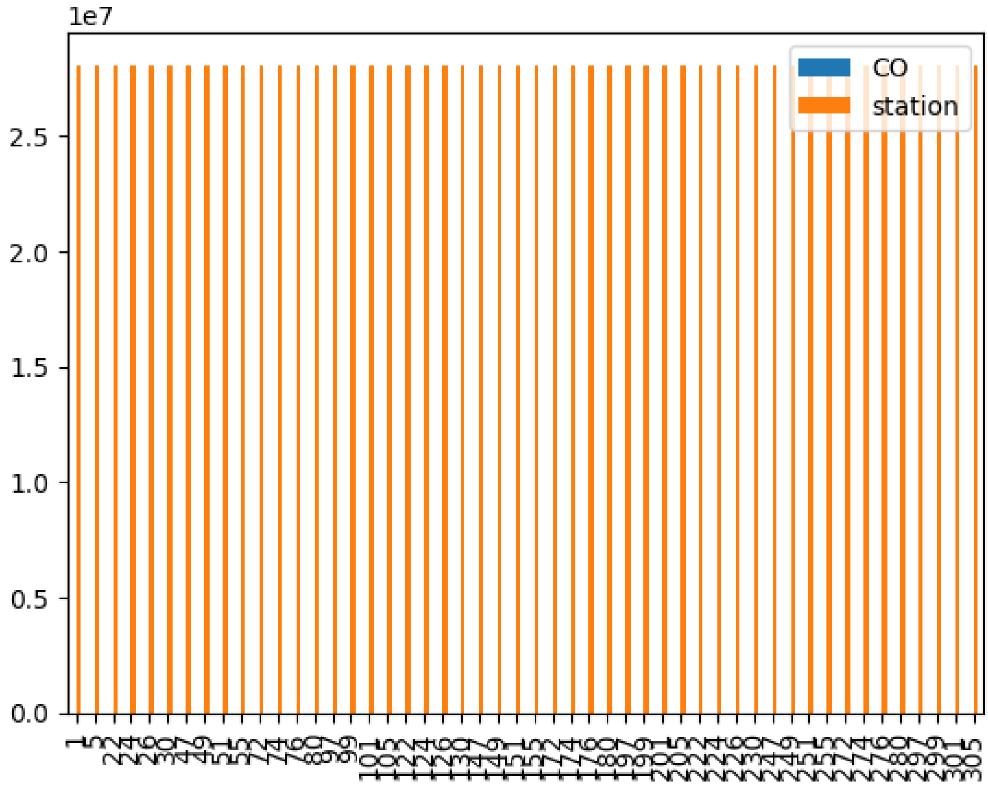
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



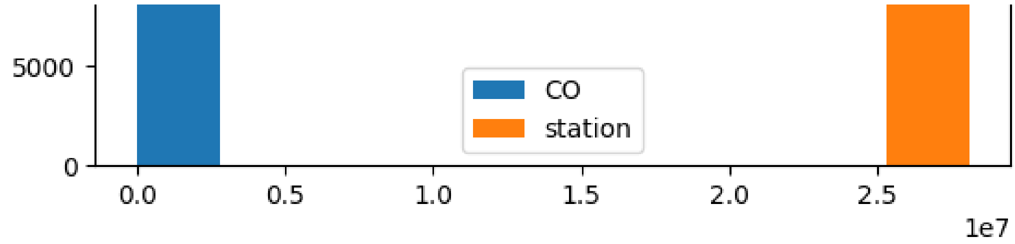

# Histogram

In [11]:

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

Out[11]:

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



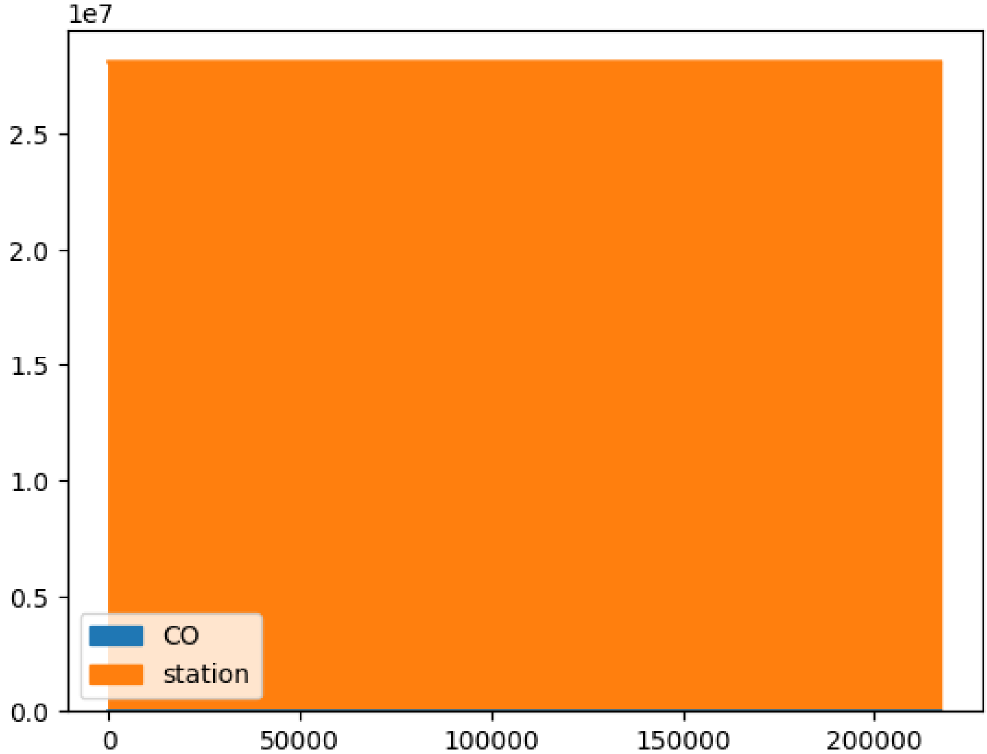

## Area chart

In [12]:

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

Out[12]:

<Axes: >




## Box chart

In [13]:

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

Out[13]:

<Axes: >

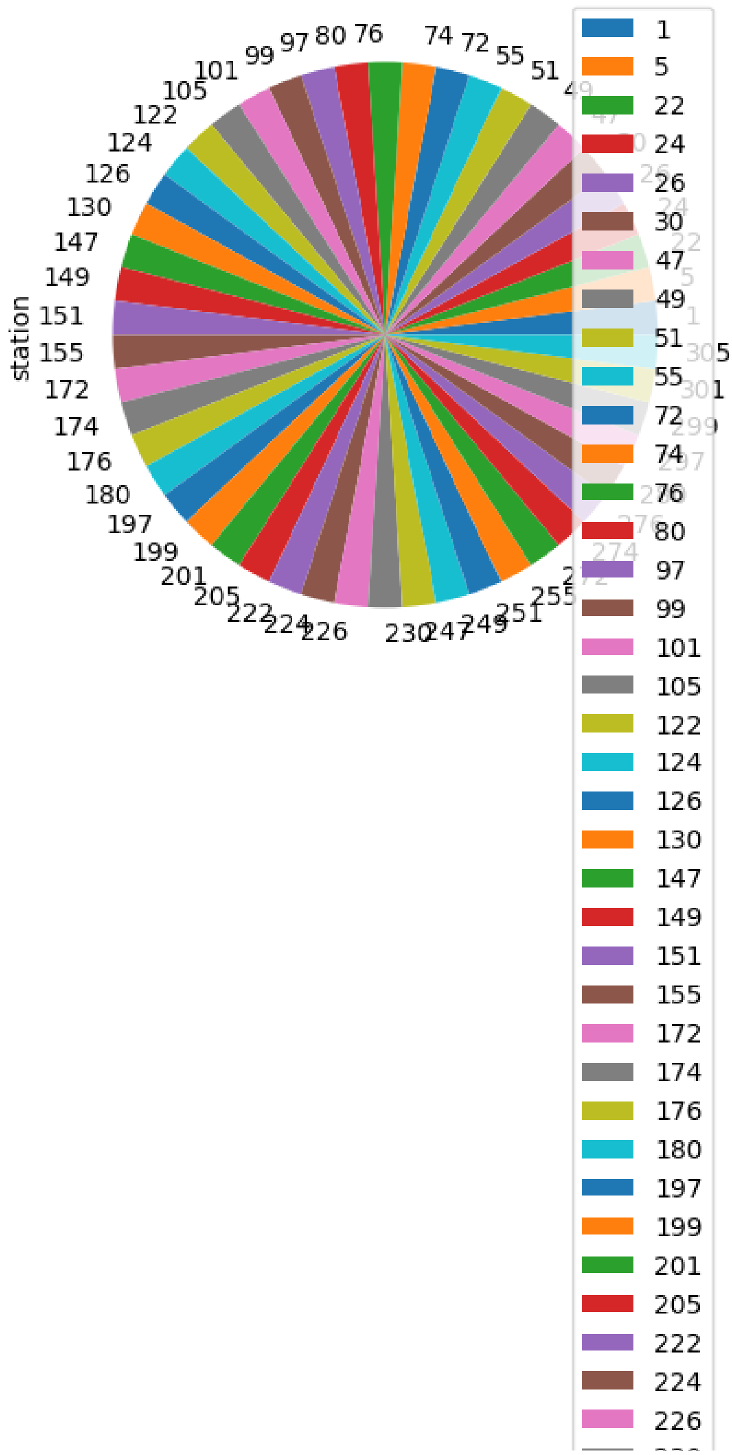

# Pie chart

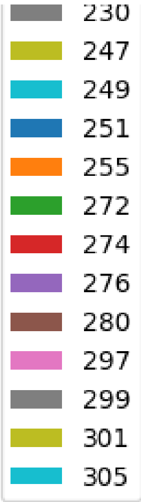
In [14]:

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

Out[14]:

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





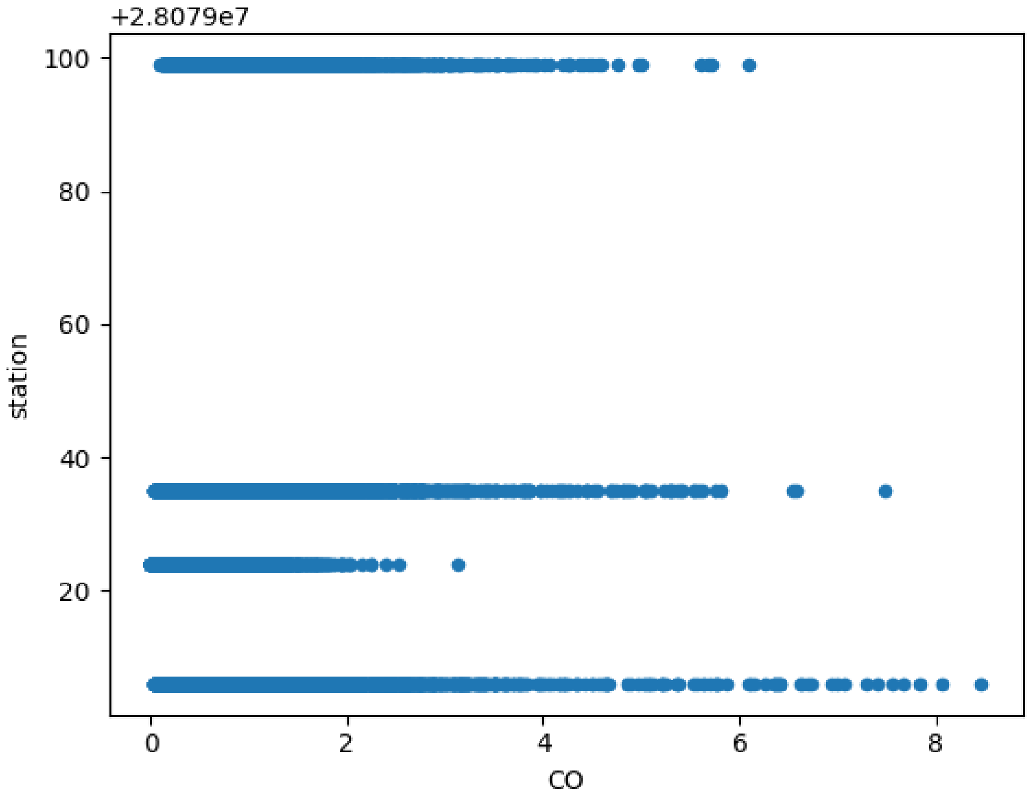

## Scatter chart

In [15]:

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

Out[15]:

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




In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
```

 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB

In [17]:

```
df.describe()
```

Out[17]:

| | BEN | CO | EBE | MXY | NMHC | NO_2 | NOx | OXY | |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 32381.000000 | 32381.000000 | 32381.000000 | 32381.000000 | 32381.000000 | 32381.000000 | 32381.000000 | 32381.000000 | 323 |
| **mean** | 2.479155 | 0.787323 | 2.914004 | 7.013636 | 0.155827 | 58.936796 | 126.009340 | 3.169093 | |
| **std** | 2.280959 | 0.610810 | 2.667881 | 6.774365 | 0.135731 | 31.472733 | 114.035078 | 2.950771 | |
| **min** | 0.180000 | 0.000000 | 0.180000 | 0.190000 | 0.000000 | 0.890000 | 1.710000 | 0.180000 | |
| **25%** | 0.970000 | 0.420000 | 1.140000 | 2.420000 | 0.080000 | 35.660000 | 48.720001 | 1.190000 | |
| **50%** | 1.840000 | 0.620000 | 2.130000 | 5.140000 | 0.130000 | 57.160000 | 96.830002 | 2.340000 | |
| **75%** | 3.250000 | 0.980000 | 3.830000 | 9.420000 | 0.200000 | 78.769997 | 167.500000 | 4.130000 | |
| **max** | 32.660000 | 8.460000 | 41.740002 | 99.879997 | 2.700000 | 263.600006 | 1336.000000 | 76.339996 | 1 |

In [18]:

```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
        'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

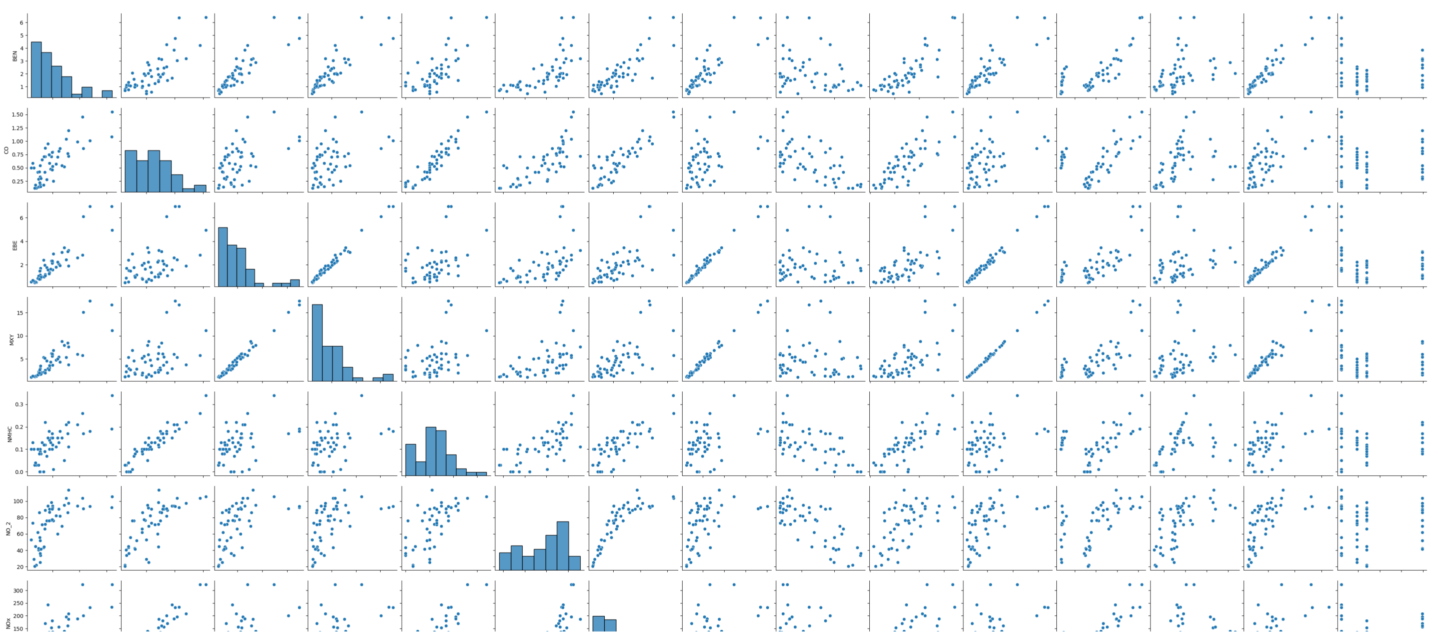
# EDA AND VISUALIZATION

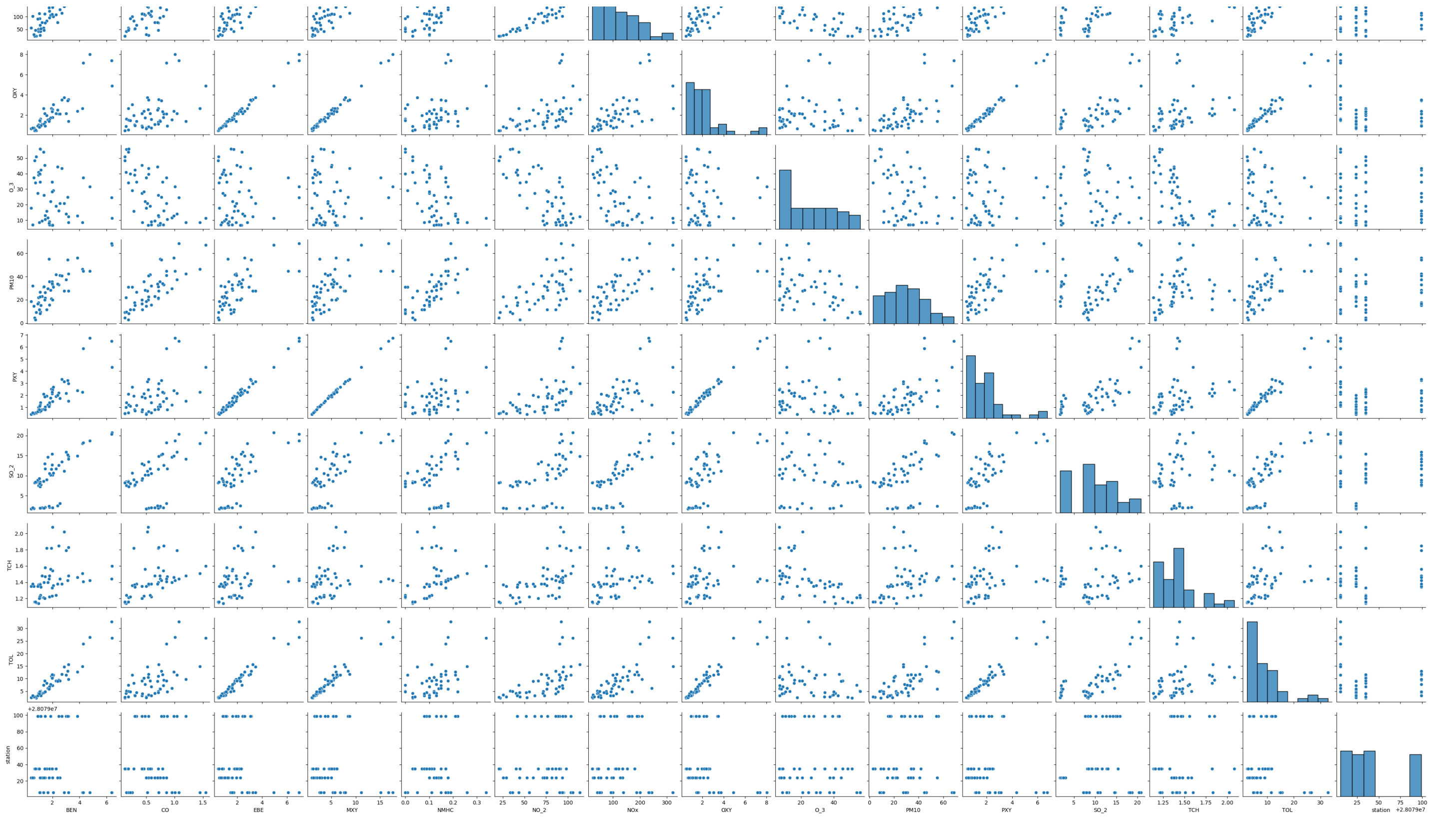
In [19]:

```
sns.pairplot(df1[0:50])
```

Out[19]:

<seaborn.axisgrid.PairGrid at 0x7f7c7914fc10>





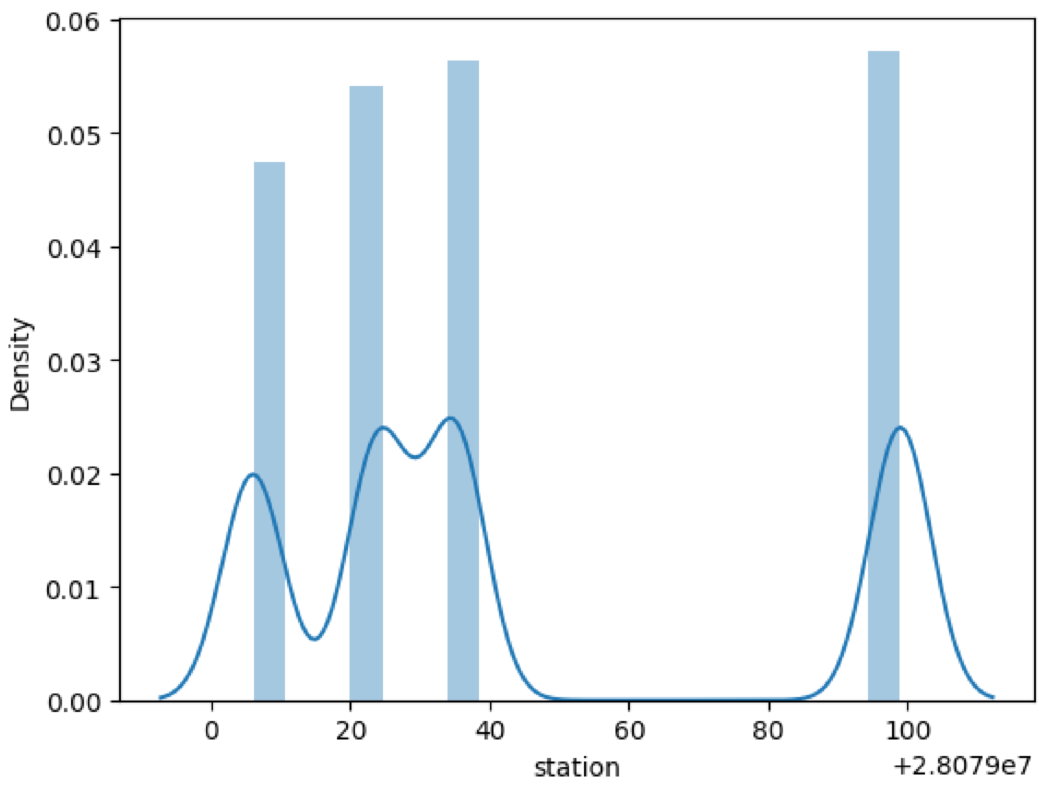
In [20]:

```
sns.distplot(df1['station'])
```

```
<ipython-input-20-4bc330f7257f>: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(df1['station'])
```

Out[20]:

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

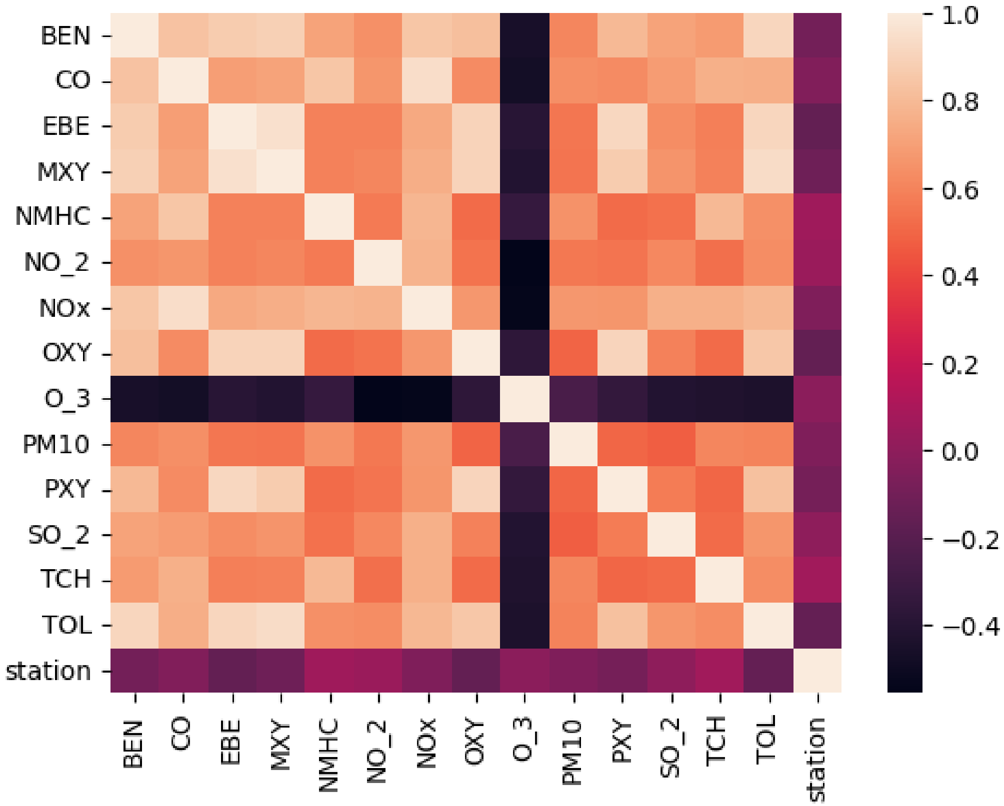


In [21]:

```
sns.heatmap(df1.corr())
```

Out[21]:

```
<Axes: >
```




## TO TRAIN THE MODEL AND MODEL BULDING

In [22]:

```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
        'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [23]:

```
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 [24]:

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

Out[24]:

```
▾ LinearRegression
LinearRegression()
```

In [25]:

```
lr.intercept_
```

Out[25]:

28078990.232478637

In [26]:

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

Out[26]:

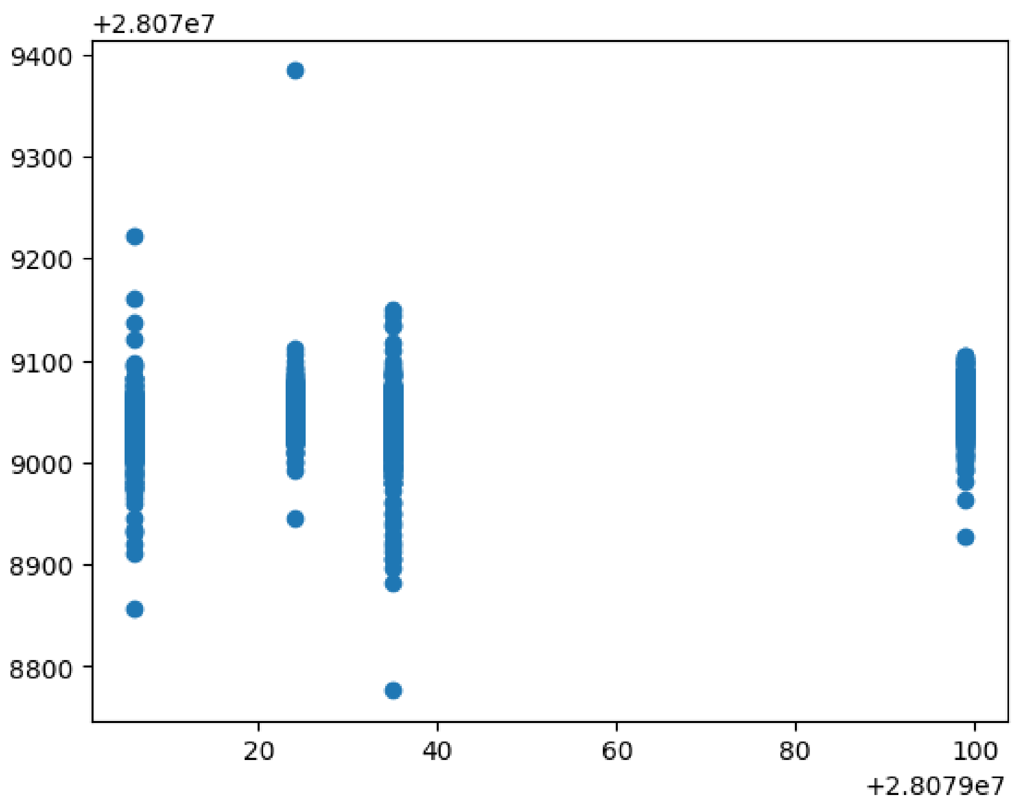
| | Co-efficient |
|---|---|
| **BEN** | 1.809828 |
| **CO** | -14.733289 |
| **EBE** | -11.672195 |
| **MXY** | 4.279424 |
| **NMHC** | 92.418335 |
| **NO_2** | 0.257456 |
| **NOx** | -0.100192 |
| **OXY** | -5.274840 |
| **O_3** | -0.035001 |
| **PM10** | -0.126043 |
| **PXY** | 7.538213 |
| **SO_2** | 0.633011 |
| **TCH** | 41.784911 |
| **TOL** | -1.467695 |

In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x7f7c6070bdf0>



# ACCURACY

In [28]:

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

Out[28]:

```
0.18255900377828316
```

In [29]:

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

Out[29]:

```
0.2055390283677454
```

# Ridge and Lasso

In [30]:

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

In [31]:

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

Out[31]:

```
▾ Ridge
Ridge(alpha=10)
```

# Accuracy(Ridge)

In [32]:

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

Out[32]:

```
0.18366223136467708
```

In [33]:

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

Out[33]:

```
0.20526531163757678
```

In [34]:

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

Out[34]:

```
▾ Lasso
Lasso(alpha=10)
```

In [35]:

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

Out[35]:

```
0.06014145811842009
```

# Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.054957863738079316
```

In [37]:

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

Out[37]:

```
▾ ElasticNet
ElasticNet()
```

In [38]:

```
en.coef_
```

Out[38]:

```
array([ 0.90764863,  0.        , -3.01272993,  1.62945896,  0.19807839,
        0.23285916, -0.03001406, -2.51883221, -0.02637025,  0.0042756 ,
        2.22628373,  0.4138471 ,  1.02558249, -1.16834097])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079038.733090658
```

In [40]:

```
prediction=en.predict(x_test)
```

In [41]:

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

Out[41]:

```
0.09188933991869308
```

# Evaluation Metrics

In [42]:

```
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)))
```

```
28.73836839998537
1134.0308585375371
33.67537466068547
```

# Logistic Regression

In [43]:

```
from sklearn.linear_model import LogisticRegression
```

In [44]:

```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [45]:

```
feature_matrix.shape
```

Out[45]:

```
(32381, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(32381,)
```

In [47]:

```
from sklearn.preprocessing import StandardScaler
```

In [48]:

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

In [49]:

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

Out[49]:

```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

In [50]:

```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [51]:

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

```
[28079035]
```

In [52]:

```
logr.classes_
```

Out[52]:

```
array([28079006, 28079024, 28079035, 28079099])
```

In [53]:

```
logr.score(fs,target_vector)
```

Out[53]:

```
0.8481825762020938
```

In [54]:

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

Out[54]:

```
2.548453802498866e-10
```

In [55]:

```
logr.predict_proba(observation)
```

Out[55]:

```
array([[2.54845380e-10, 3.39752911e-71, 1.00000000e+00, 1.42865321e-13]])
```

# Random Forest

In [56]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [57]:

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

Out[57]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [58]:

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

In [59]:

```
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[59]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
▸ RandomForestClassifier
```

In [60]:

```
grid_search.best_score_
```

Out[60]:

```
0.773493338039354
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

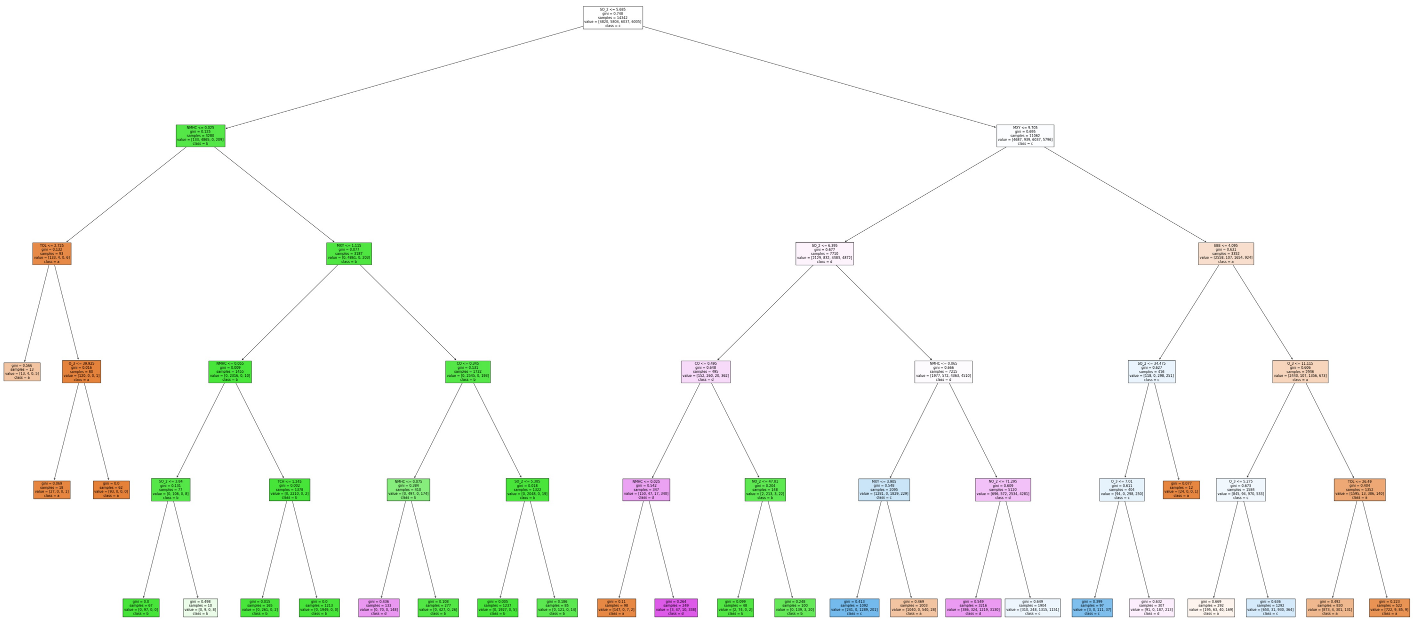
In [62]:

```
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'],f
illed=True)
```

Out[62]:

```
[Text(0.4348958333333333, 0.9166666666666666, 'SO_2 <= 5.685\ngini = 0.748\nsamples = 143
42\nvalue = [4820, 5804, 6037, 6005]\nclass = c'),
 Text(0.14583333333333334, 0.75, 'NMHC <= 0.025\ngini = 0.125\nsamples = 3280\nvalue = [1
33, 4865, 0, 209]\nclass = b'),
 Text(0.041666666666666664, 0.5833333333333334, 'TOL <= 2.725\ngini = 0.132\nsamples = 93
\nvalue = [133, 4, 0, 6]\nclass = a'),
 Text(0.020833333333333332, 0.4166666666666667, 'gini = 0.566\nsamples = 13\nvalue = [13,
4, 0, 5]\nclass = a'),
 Text(0.0625, 0.4166666666666667, 'O_3 <= 39.925\ngini = 0.016\nsamples = 80\nvalue = [12
0, 0, 0, 1]\nclass = a'),
 Text(0.041666666666666664, 0.25, 'gini = 0.069\nsamples = 18\nvalue = [27, 0, 0, 1]\ncla
ss = a'),
 Text(0.08333333333333333, 0.25, 'gini = 0.0\nsamples = 62\nvalue = [93, 0, 0, 0]\nclass
= a'),
 Text(0.25, 0.5833333333333334, 'MXY <= 1.115\ngini = 0.077\nsamples = 3187\nvalue = [0,
4861, 0, 203]\nclass = b'),
 Text(0.16666666666666666, 0.4166666666666667, 'NMHC <= 0.055\ngini = 0.009\nsamples = 14
55\nvalue = [0, 2316, 0, 10]\nclass = b'),
 Text(0.125, 0.25, 'SO_2 <= 3.84\ngini = 0.131\nsamples = 77\nvalue = [0, 106, 0, 8]\ncla
ss = b'),
 Text(0.10416666666666667, 0.08333333333333333, 'gini = 0.0\nsamples = 67\nvalue = [0, 97
, 0, 0]\nclass = b'),
 Text(0.14583333333333334, 0.08333333333333333, 'gini = 0.498\nsamples = 10\nvalue = [0,
9, 0, 8]\nclass = b'),
 Text(0.20833333333333334, 0.25, 'TCH <= 1.245\ngini = 0.002\nsamples = 1378\nvalue = [0,
2210, 0, 2]\nclass = b'),
 Text(0.1875, 0.08333333333333333, 'gini = 0.015\nsamples = 165\nvalue = [0, 261, 0, 2]\n
class = b'),
 Text(0.22916666666666666, 0.08333333333333333, 'gini = 0.0\nsamples = 1213\nvalue = [0,
1949, 0, 0]\nclass = b'),
 Text(0.3333333333333333, 0.4166666666666667, 'CO <= 0.345\ngini = 0.131\nsamples = 1732\
nvalue = [0, 2545, 0, 193]\nclass = b'),
 Text(0.2916666666666667, 0.25, 'NMHC <= 0.075\ngini = 0.384\nsamples = 410\nvalue = [0,
497, 0, 174]\nclass = b'),
 Text(0.2708333333333333, 0.08333333333333333, 'gini = 0.436\nsamples = 133\nvalue = [0,
70, 0, 148]\nclass = d'),
 Text(0.3125, 0.08333333333333333, 'gini = 0.108\nsamples = 277\nvalue = [0, 427, 0, 26]\
nclass = b'),
 Text(0.375, 0.25, 'SO_2 <= 5.385\ngini = 0.018\nsamples = 1322\nvalue = [0, 2048, 0, 19]
\nclass = b'),
 Text(0.3541666666666667, 0.08333333333333333, 'gini = 0.005\nsamples = 1237\nvalue = [0,
1927, 0, 5]\nclass = b'),
 Text(0.3958333333333333, 0.08333333333333333, 'gini = 0.186\nsamples = 85\nvalue = [0, 1
21, 0, 14]\nclass = b'),
 Text(0.7239583333333334, 0.75, 'MXY <= 9.705\ngini = 0.695\nsamples = 11062\nvalue = [46
87, 939, 6037, 5796]\nclass = c'),
 Text(0.5833333333333334, 0.5833333333333334, 'SO_2 <= 6.395\ngini = 0.677\nsamples = 771
0\nvalue = [2129, 832, 4383, 4872]\nclass = d'),
 Text(0.5, 0.4166666666666667, 'CO <= 0.495\ngini = 0.648\nsamples = 495\nvalue = [152, 2
60, 20, 362]\nclass = d'),
 Text(0.4583333333333333, 0.25, 'NMHC <= 0.025\ngini = 0.542\nsamples = 347\nvalue = [150
, 47, 17, 340]\nclass = d'),
 Text(0.4375, 0.08333333333333333, 'gini = 0.11\nsamples = 98\nvalue = [147, 0, 7, 2]\ncl
ass = a'),
 Text(0.4791666666666667, 0.08333333333333333, 'gini = 0.264\nsamples = 249\nvalue = [3,
47, 10, 338]\nclass = d'),
 Text(0.5416666666666666, 0.25, 'NO_2 <= 47.81\ngini = 0.204\nsamples = 148\nvalue = [2,
213, 3, 22]\nclass = b'),
 Text(0.5208333333333334, 0.08333333333333333, 'gini = 0.099\nsamples = 48\nvalue = [2, 7
4, 0, 2]\nclass = b'),
 Text(0.5625, 0.08333333333333333, 'gini = 0.248\nsamples = 100\nvalue = [0, 139, 3, 20]\
nclass = b'),
 Text(0.6666666666666666, 0.4166666666666667, 'NMHC <= 0.065\ngini = 0.666\nsamples = 721
```

```
5\nvalue = [1977, 572, 4363, 4510]\nclass = d'),
 Text(0.625, 0.25, 'MXY <= 3.905\ngini = 0.548\nsamples = 2095\nvalue = [1281, 0, 1829, 2
29]\nclass = c'),
 Text(0.6041666666666666, 0.08333333333333333, 'gini = 0.413\nsamples = 1092\nvalue = [24
1, 0, 1289, 201]\nclass = c'),
 Text(0.6458333333333334, 0.08333333333333333, 'gini = 0.469\nsamples = 1003\nvalue = [10
40, 0, 540, 28]\nclass = a'),
 Text(0.7083333333333334, 0.25, 'NO_2 <= 71.295\ngini = 0.609\nsamples = 5120\nvalue = [6
96, 572, 2534, 4281]\nclass = d'),
 Text(0.6875, 0.08333333333333333, 'gini = 0.549\nsamples = 3216\nvalue = [386, 324, 1219
, 3130]\nclass = d'),
 Text(0.7291666666666666, 0.08333333333333333, 'gini = 0.649\nsamples = 1904\nvalue = [31
0, 248, 1315, 1151]\nclass = c'),
 Text(0.8645833333333334, 0.5833333333333334, 'EBE <= 4.095\ngini = 0.631\nsamples = 3352
\nvalue = [2558, 107, 1654, 924]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'SO_2 <= 34.475\ngini = 0.627\nsamples = 416\nvalue = [
118, 0, 298, 251]\nclass = c'),
 Text(0.7916666666666666, 0.25, 'O_3 <= 7.01\ngini = 0.611\nsamples = 404\nvalue = [94, 0
, 298, 250]\nclass = c'),
 Text(0.7708333333333334, 0.08333333333333333, 'gini = 0.399\nsamples = 97\nvalue = [3, 0
, 111, 37]\nclass = c'),
 Text(0.8125, 0.08333333333333333, 'gini = 0.632\nsamples = 307\nvalue = [91, 0, 187, 213
]\nclass = d'),
 Text(0.8333333333333334, 0.25, 'gini = 0.077\nsamples = 12\nvalue = [24, 0, 0, 1]\nclass
 = a'),
 Text(0.9166666666666666, 0.4166666666666667, 'O_3 <= 11.115\ngini = 0.606\nsamples = 293
6\nvalue = [2440, 107, 1356, 673]\nclass = a'),
 Text(0.875, 0.25, 'O_3 <= 5.275\ngini = 0.673\nsamples = 1584\nvalue = [845, 94, 970, 53
3]\nclass = c'),
 Text(0.8541666666666666, 0.08333333333333333, 'gini = 0.669\nsamples = 292\nvalue = [195
, 63, 40, 169]\nclass = a'),
 Text(0.8958333333333334, 0.08333333333333333, 'gini = 0.636\nsamples = 1292\nvalue = [65
0, 31, 930, 364]\nclass = c'),
 Text(0.9583333333333334, 0.25, 'TOL <= 26.49\ngini = 0.404\nsamples = 1352\nvalue = [159
5, 13, 386, 140]\nclass = a'),
 Text(0.9375, 0.08333333333333333, 'gini = 0.492\nsamples = 830\nvalue = [873, 4, 301, 13
1]\nclass = a'),
 Text(0.9791666666666666, 0.08333333333333333, 'gini = 0.223\nsamples = 522\nvalue = [722
, 9, 85, 9]\nclass = a')]
```



## Conclusion

In [63]:

```
print("Linear Regression:",lr.score(x_test,y_test))
```

```python
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: 0.18255900377828316
Ridge Regression: 0.18366223136467708
Lasso Regression 0.054957863738079316
ElasticNet Regression: 0.09188933991869308
Logistic Regression: 0.8481825762020938
Random Forest: 0.773493338039354
```

# Logistic Regression is suitable for this dataset