20104169 - SUMESH R

Importing Libraries

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_2008.csv")
df
```

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

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	SO_2	TCI
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.889999	10.40	NaN	8.98	Nal
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.040001	NaN	NaN	5.85	Nal
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.270000	NaN	NaN	6.95	Nal
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.850000	NaN	NaN	5.96	Nal
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.160000	21.90	1.43	10.92	1.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5.450000	5.15	1.86	9.68	1.2
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15.020000	NaN	NaN	8.90	Nal
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17.540001	13.91	NaN	7.00	1.5
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11.910000	NaN	NaN	8.02	1.5
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12.690000	11.42	1.98	8.74	1.4

226392 rows × 17 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', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [6]:

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

Out[6]:

	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...	...	...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

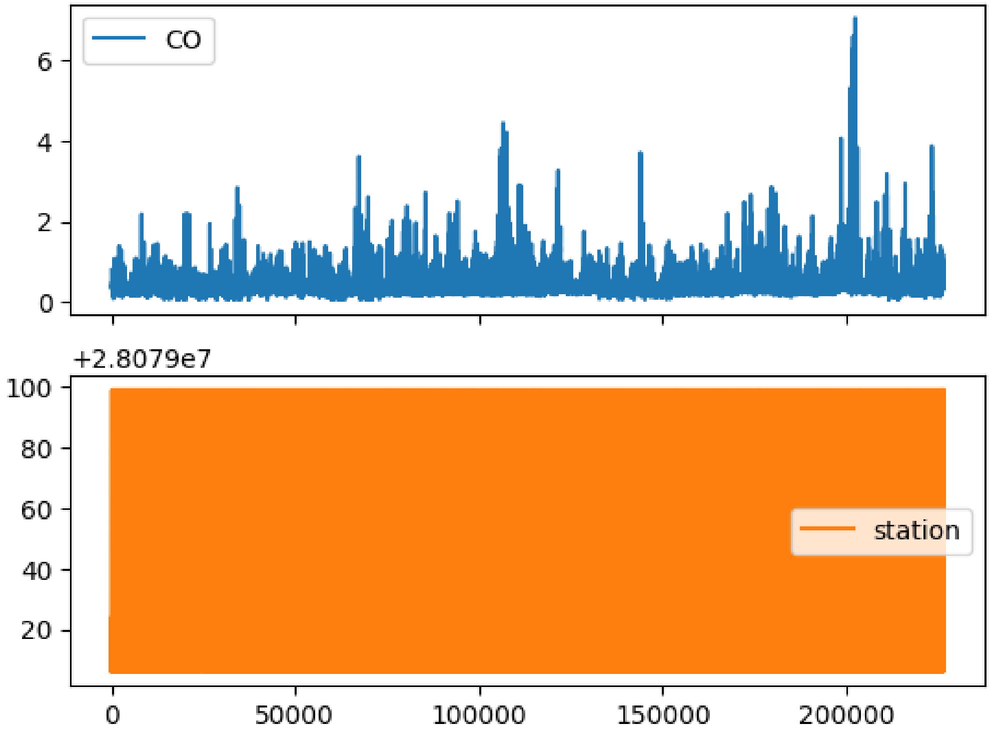
Line chart

In [7]:

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

Out[7]:

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



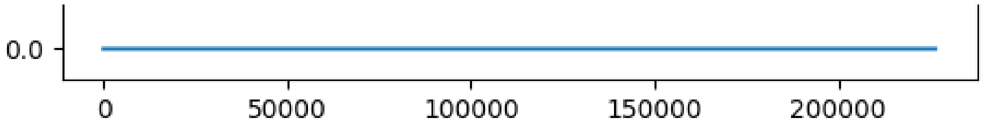
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

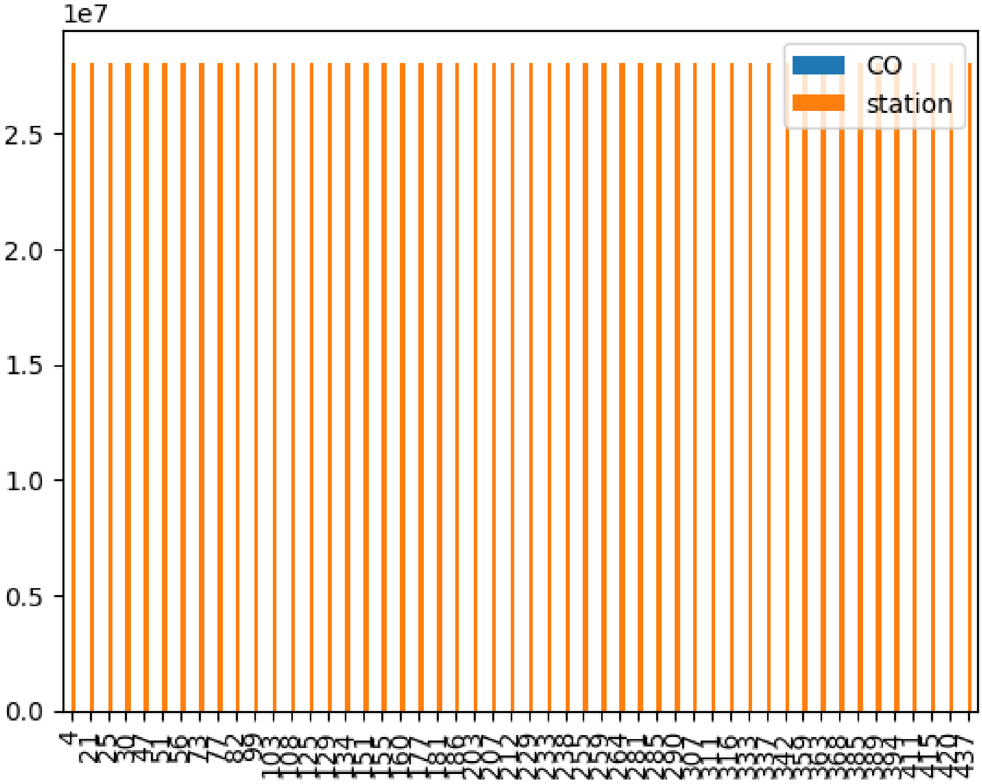
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



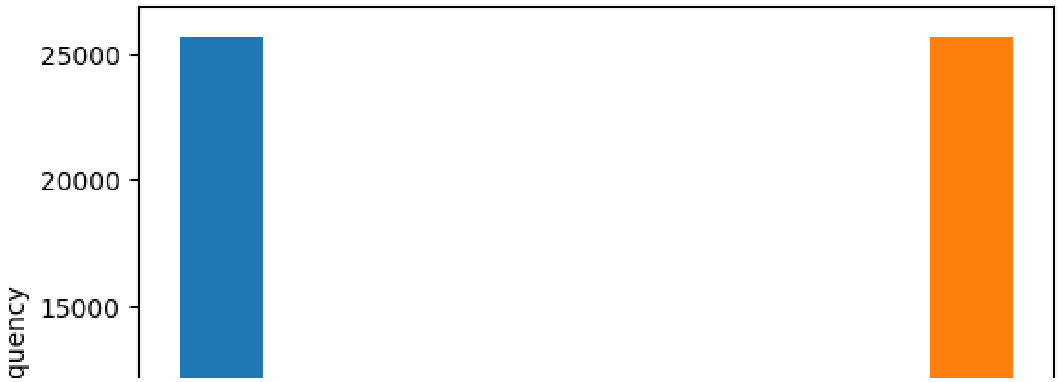
Histogram

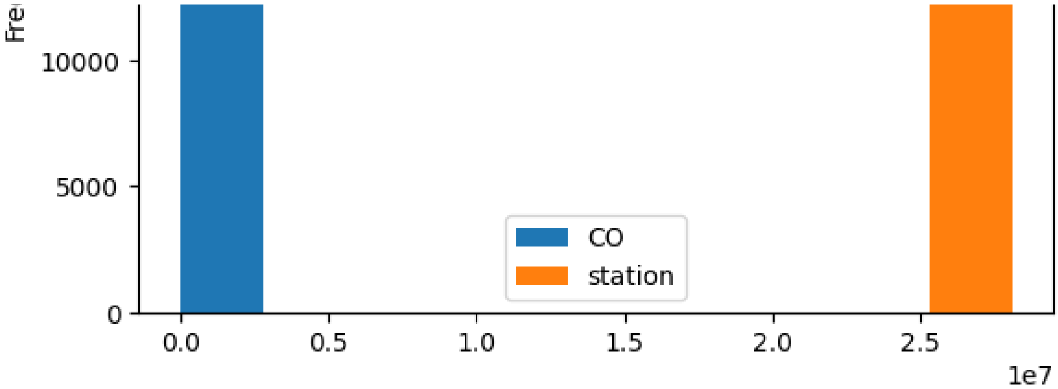
In [11]:

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

Out[11]:

<Axes: ylabel='Frequency'>





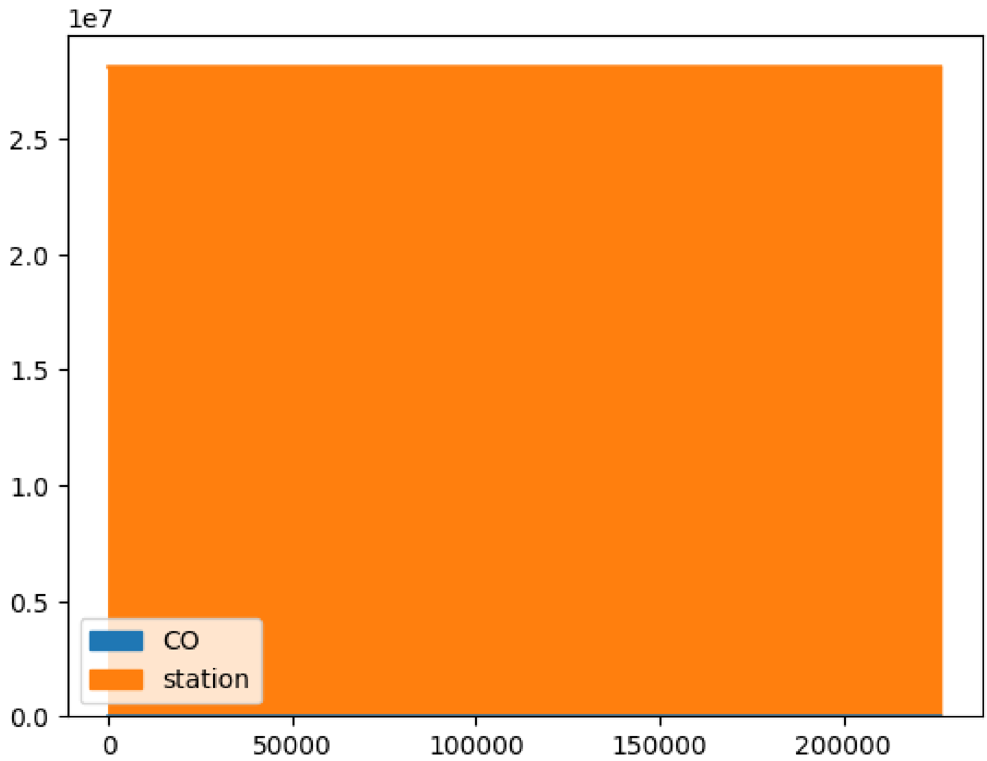
Area chart

In [12]:

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

Out[12]:

<Axes: >



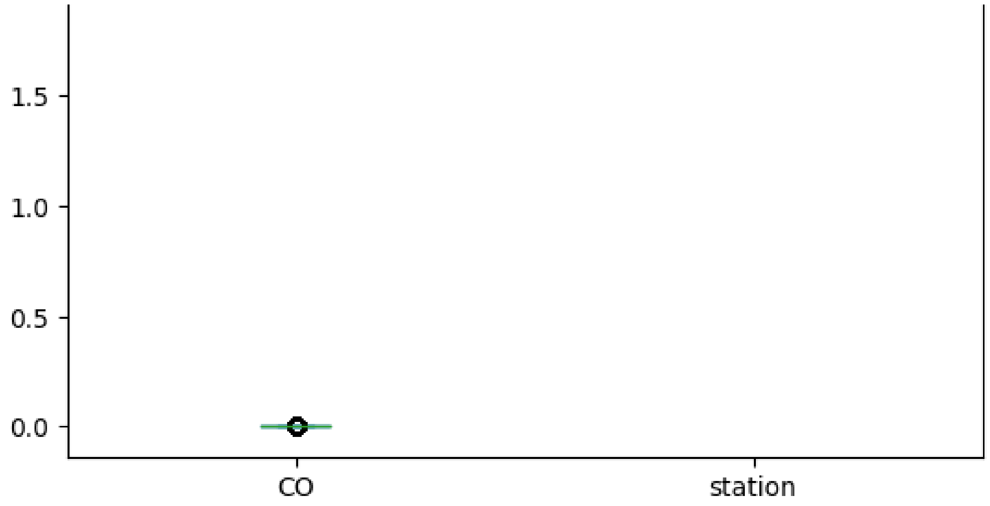
Box chart

In [13]:

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

Out[13]:

<Axes: >



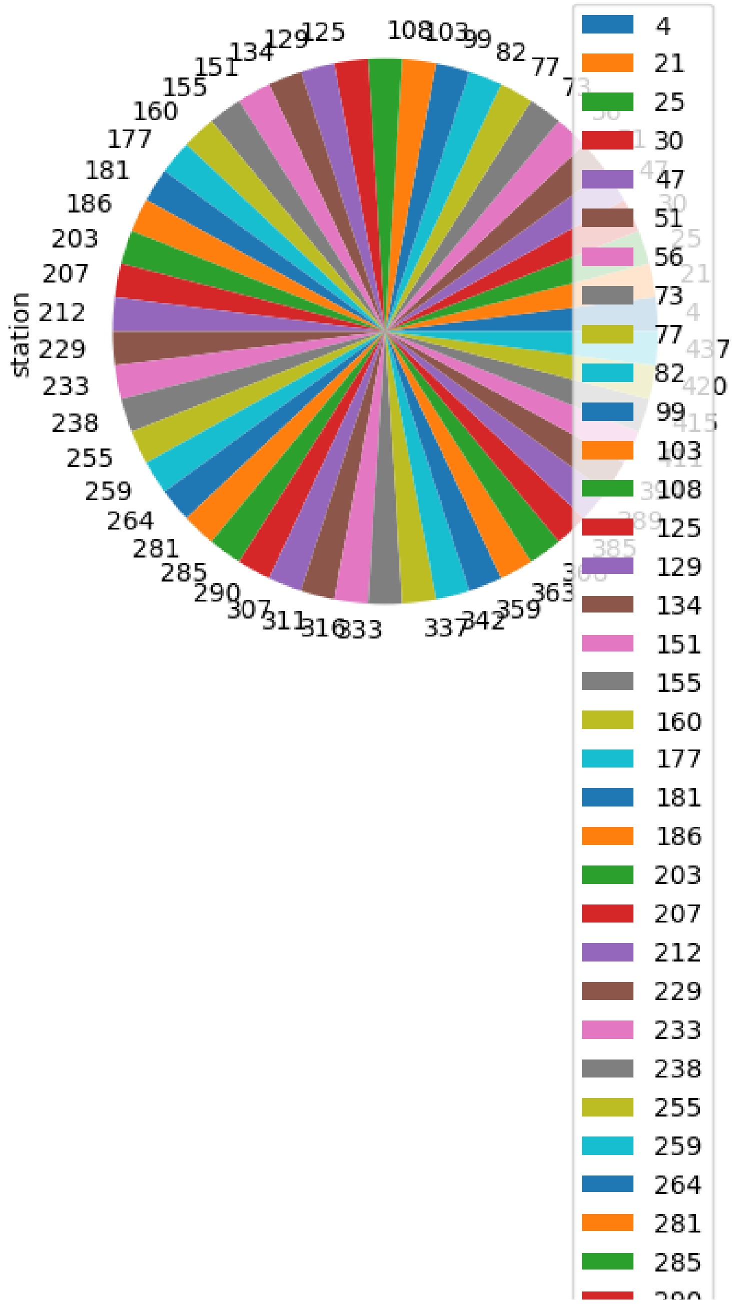
Pie chart

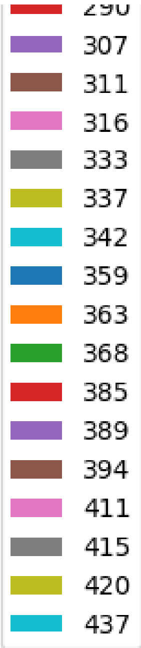
In [14]:

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

Out[14]:

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





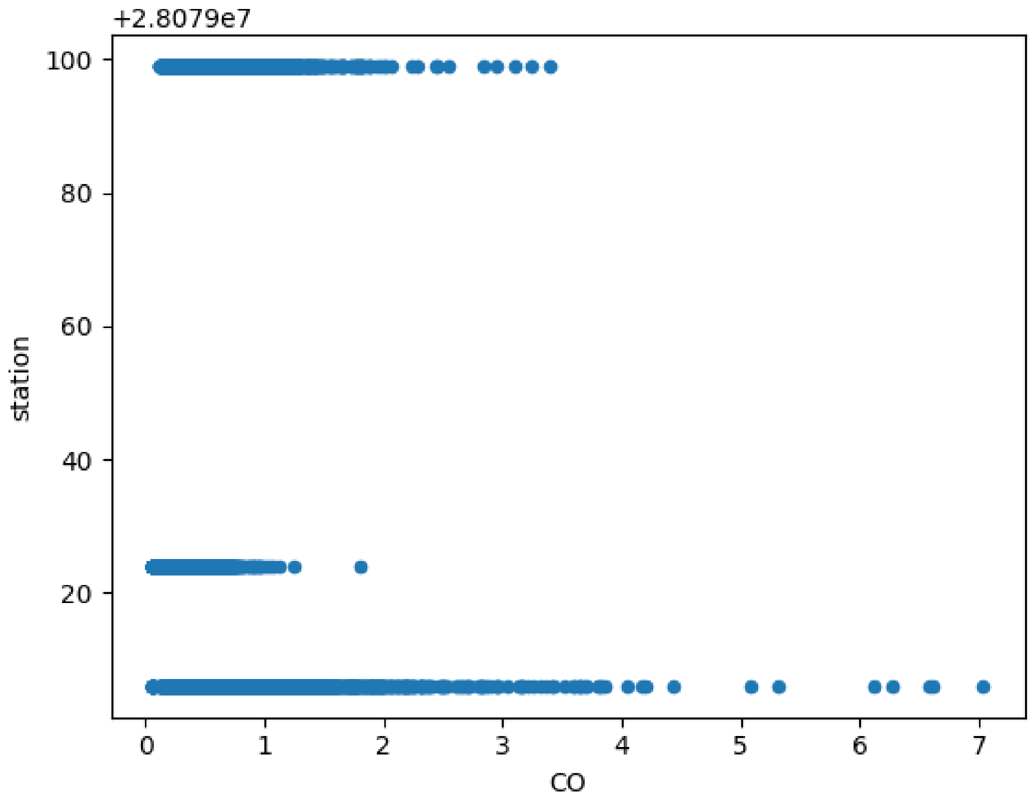
Scatter chart

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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     25631 non-null  object
 1   BEN      25631 non-null  float64
```

```
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	256
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261	98.007732	1.479964	
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647	101.448238	1.258928	
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000	2.110000	0.140000	
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999	32.635000	0.870000	
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000	71.110001	1.000000	
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999	131.550003	1.760000	
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024	2004.000000	28.020000	2

In [18]:

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

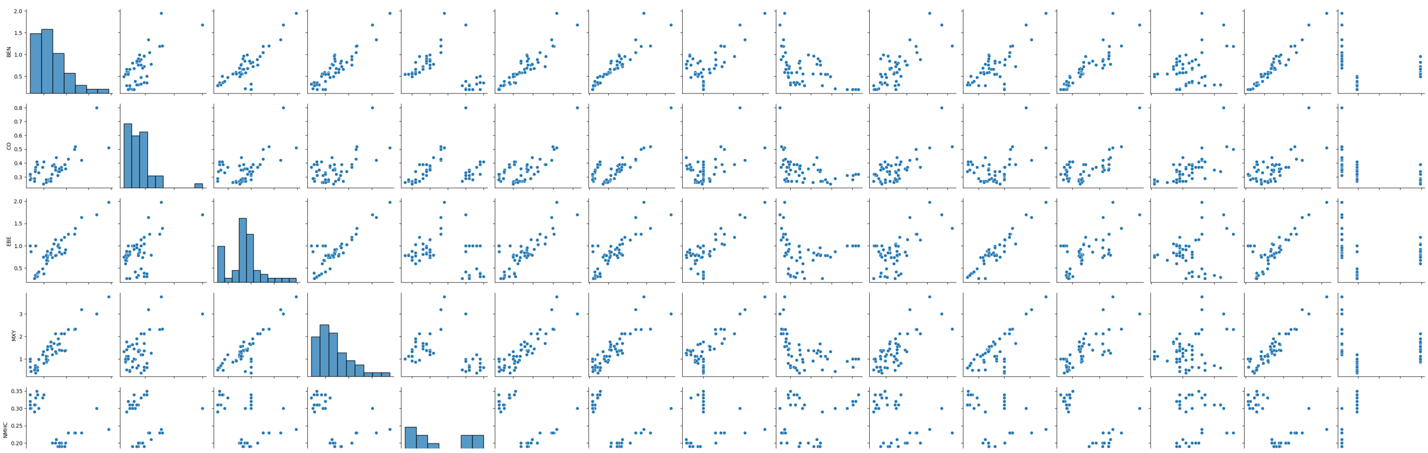
EDA AND VISUALIZATION

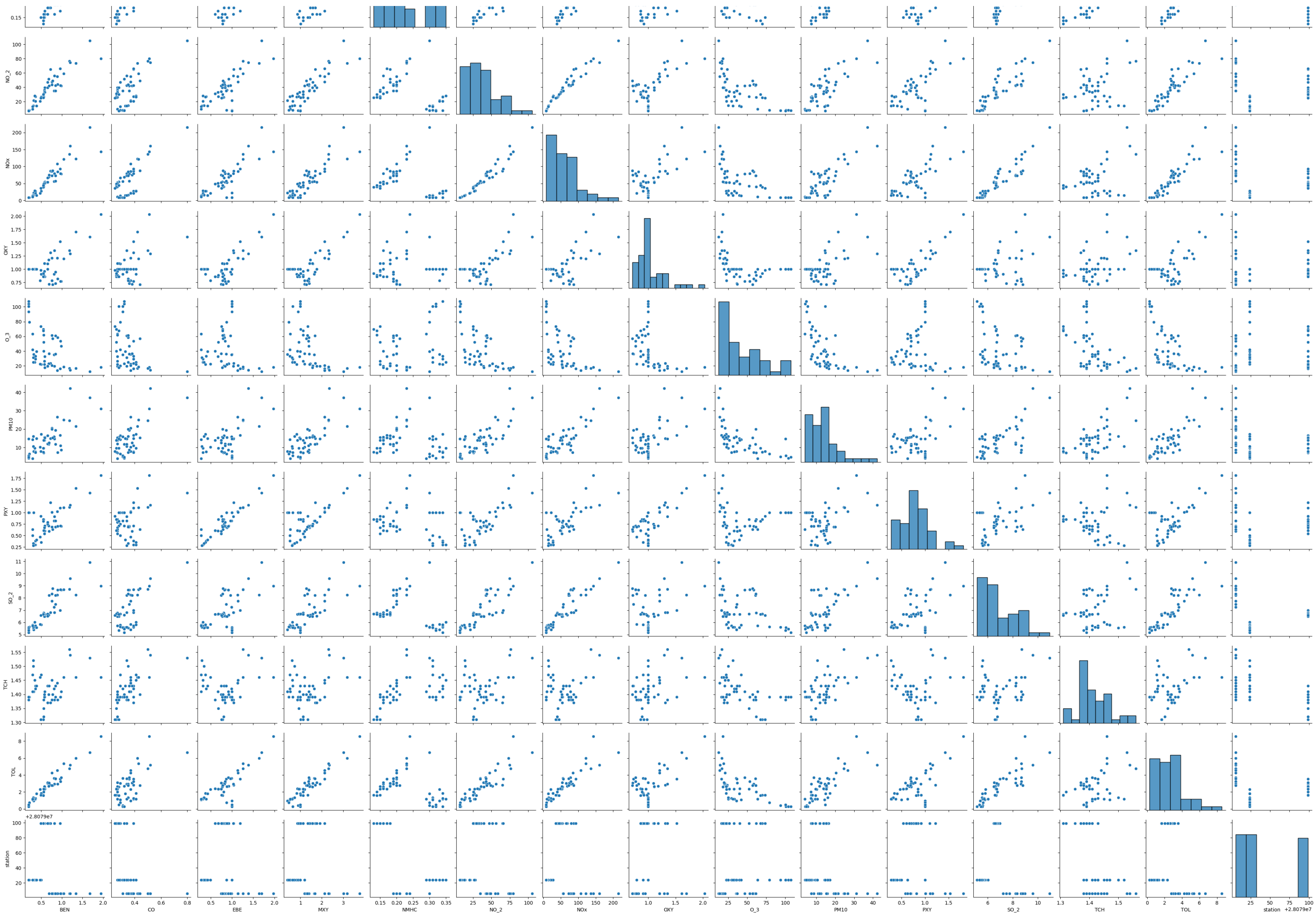
In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x7ef0eb316830>





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.

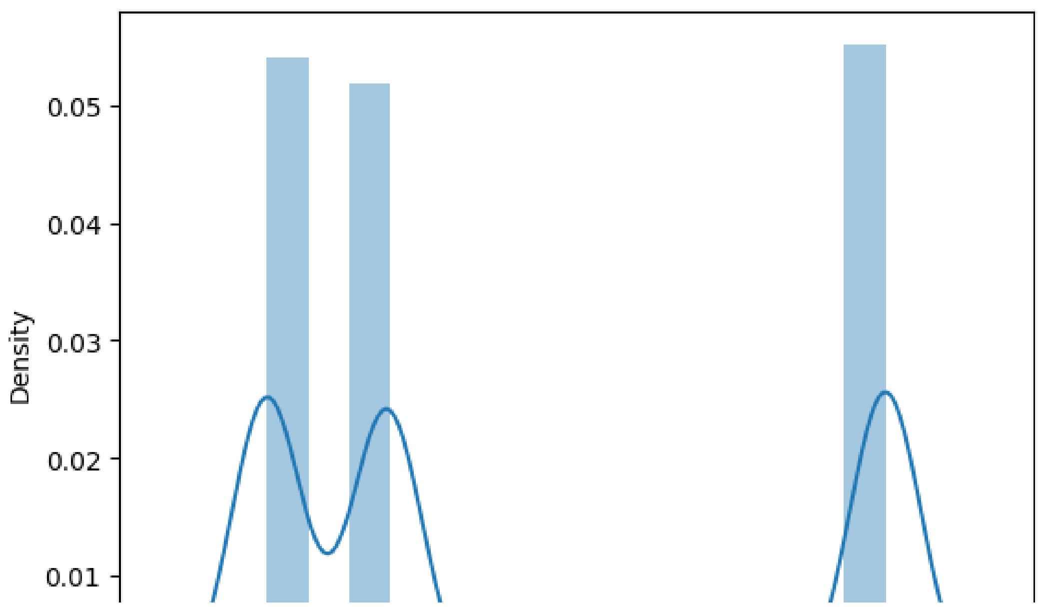
Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

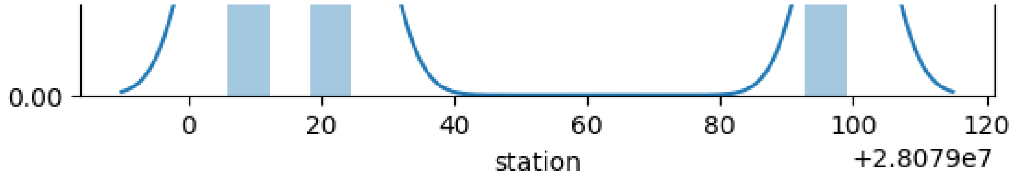
For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

    sns.distplot(df1['station'])

Out[20]:

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



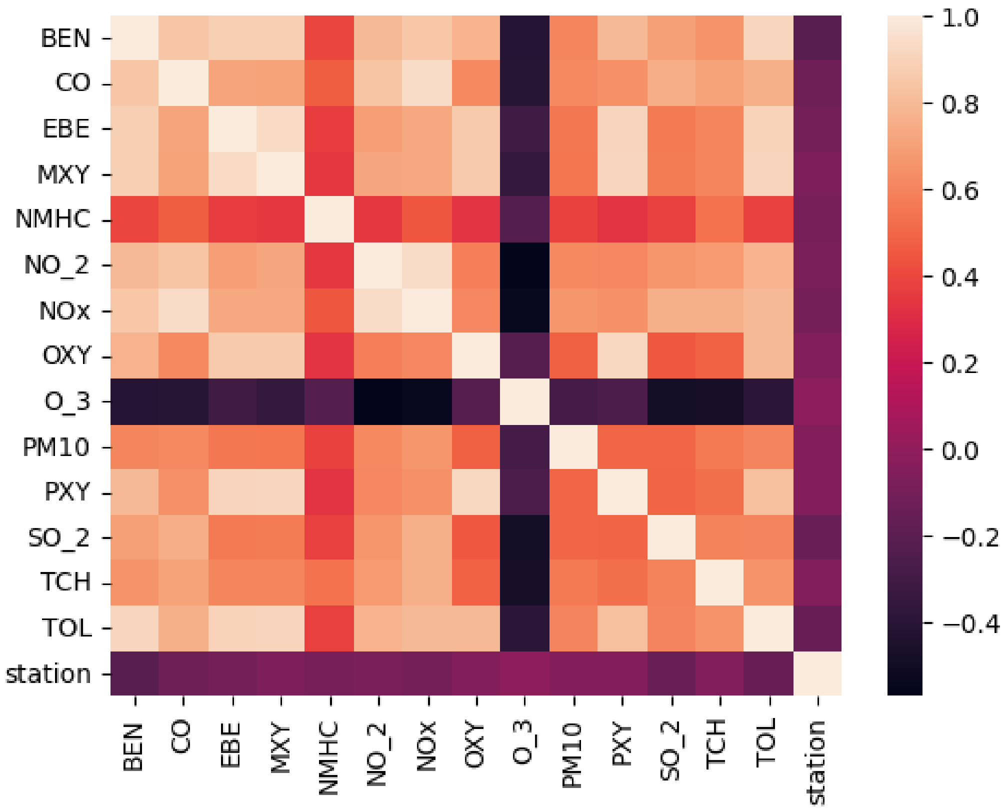


In [21]:

```
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]:

28079032.671049826

In [26]:

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

Out[26]:

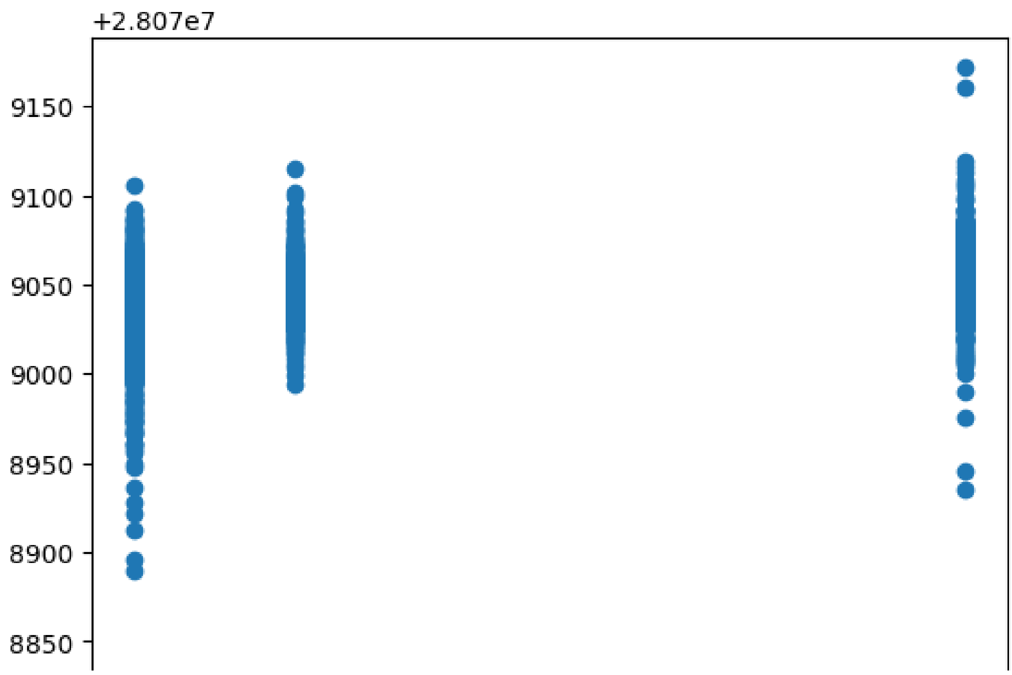
	Co-efficient
BEN	-24.990234
CO	-0.056987
EBE	-1.552171
MXY	7.716946
NMHC	-24.900897
NO_2	-0.033259
NOx	0.118478
OXY	3.892108
O_3	-0.140412
PM10	0.141664
PXY	1.960910
SO_2	-0.598225
TCH	19.110382
TOL	-1.946012

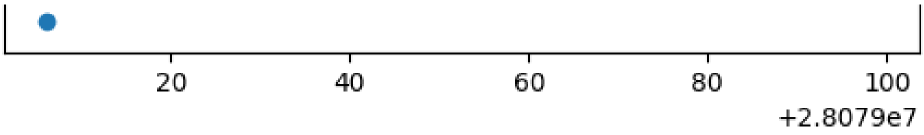
In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x7ef0d45fdde0>





ACCURACY

In [28]:

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

Out[28]:

0.15290366306738545

In [29]:

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

Out[29]:

0.13948682050013284

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]:

0.15276766850837376

In [33]:

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

Out[33]:

0.13946286669960595

In [34]:

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

Out[34]:

▾ Lasso

Lasso(alpha=10)

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

In [35]:

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

Out[35]:

```
0.04093060208584942
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.04114787529283537
```

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([-4.52382059, -0.        , -0.        ,  3.2861883 , -0.        ,
        0.05853171,  0.02289057,  1.5037067 , -0.15964927,  0.1370066 ,
        1.47937709, -0.92099626,  0.        , -2.55540006])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079057.486030195
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.09645231136315036
```

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)))
```

```
35.69170143740743
1481.8712126387766
38.49508036929883
```

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]:

```
(25631, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(25631,)
```

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)
```

```
[28079099]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

array([28079006, 28079024, 28079099])

In [53]:

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

Out[53]:

0.794194530061254

In [54]:

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

Out[54]:

8.321801665678893e-09

In [55]:

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

Out[55]:

array([[8.32180167e-09, 1.19114695e-13, 9.99999992e-01]])

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]:

0.8502310802788671

In [61]:

```
rfc_best=grid_search.best_estimator_
```

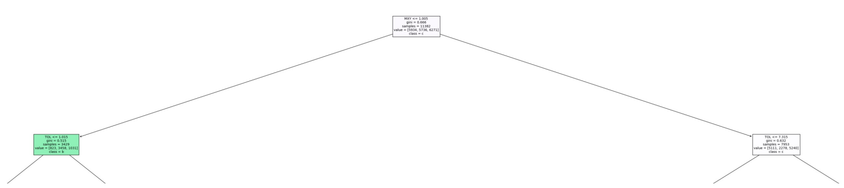
In [62]:

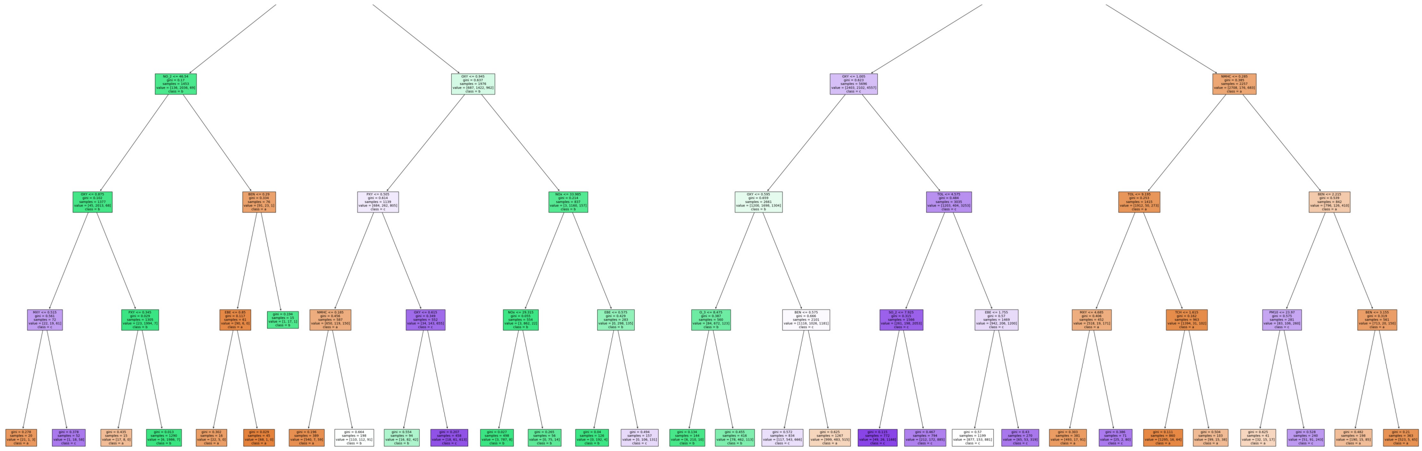
```
from sklearn.tree import plot_tree

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(0.48125, 0.9166666666666666, 'MXY <= 1.005\ngini = 0.666\nsamples = 11382\nvalue =
[5934, 5736, 6271]\nclass = c'),
 Text(0.22916666666666666, 0.75, 'TOL <= 1.015\ngini = 0.515\nsamples = 3429\nvalue = [82
3, 3458, 1031]\nclass = b'),
 Text(0.125, 0.5833333333333334, 'NO_2 <= 46.54\ngini = 0.17\nsamples = 1453\nvalue = [13
6, 2036, 69]\nclass = b'),
 Text(0.06666666666666667, 0.4166666666666667, 'OXY <= 0.875\ngini = 0.102\nsamples = 137
7\nvalue = [45, 2013, 68]\nclass = b'),
 Text(0.03333333333333333, 0.25, 'MXY <= 0.515\ngini = 0.561\nsamples = 72\nvalue = [22,
19, 61]\nclass = c'),
 Text(0.016666666666666666, 0.08333333333333333, 'gini = 0.278\nsamples = 20\nvalue = [21
, 1, 3]\nclass = a'),
 Text(0.05, 0.08333333333333333, 'gini = 0.378\nsamples = 52\nvalue = [1, 18, 58]\nclass
= c'),
 Text(0.1, 0.25, 'PXY <= 0.345\ngini = 0.029\nsamples = 1305\nvalue = [23, 1994, 7]\nclas
s = b'),
 Text(0.08333333333333333, 0.08333333333333333, 'gini = 0.435\nsamples = 15\nvalue = [17,
8, 0]\nclass = a'),
 Text(0.11666666666666667, 0.08333333333333333, 'gini = 0.013\nsamples = 1290\nvalue = [6
, 1986, 7]\nclass = b'),
 Text(0.18333333333333332, 0.4166666666666667, 'BEN <= 0.29\ngini = 0.334\nsamples = 76\n
value = [91, 23, 1]\nclass = a'),
 Text(0.16666666666666666, 0.25, 'EBE <= 0.85\ngini = 0.117\nsamples = 61\nvalue = [90, 6
, 0]\nclass = a'),
 Text(0.15, 0.08333333333333333, 'gini = 0.302\nsamples = 16\nvalue = [22, 5, 0]\nclass =
a'),
 Text(0.18333333333333332, 0.08333333333333333, 'gini = 0.029\nsamples = 45\nvalue = [68,
1, 0]\nclass = a'),
 Text(0.2, 0.25, 'gini = 0.194\nsamples = 15\nvalue = [1, 17, 1]\nclass = b'),
 Text(0.3333333333333333, 0.5833333333333334, 'OXY <= 0.945\ngini = 0.637\nsamples = 1976
\nvalue = [687, 1422, 962]\nclass = b'),
 Text(0.26666666666666666, 0.4166666666666667, 'PXY <= 0.505\ngini = 0.614\nsamples = 113
9\nvalue = [684, 262, 805]\nclass = c'),
 Text(0.23333333333333334, 0.25, 'NMHC <= 0.185\ngini = 0.456\nsamples = 587\nvalue = [65
0, 119, 150]\nclass = a'),
 Text(0.21666666666666667, 0.08333333333333333, 'gini = 0.196\nsamples = 389\nvalue = [54
0, 7, 59]\nclass = a'),
 Text(0.25, 0.08333333333333333, 'gini = 0.664\nsamples = 198\nvalue = [110, 112, 91]\ncl
ass = b'),
 Text(0.3, 0.25, 'OXY <= 0.615\ngini = 0.349\nsamples = 552\nvalue = [34, 143, 655]\nclas
s = c'),
 Text(0.2833333333333333, 0.08333333333333333, 'gini = 0.554\nsamples = 94\nvalue = [16,
82, 42]\nclass = b'),
 Text(0.31666666666666665, 0.08333333333333333, 'gini = 0.207\nsamples = 458\nvalue = [18
, 61, 613]\nclass = c'),
 Text(0.4, 0.4166666666666667, 'NOx <= 33.985\ngini = 0.214\nsamples = 837\nvalue = [3, 1
160, 157]\nclass = b'),
 Text(0.36666666666666664, 0.25, 'NOx <= 29.315\ngini = 0.055\nsamples = 554\nvalue = [3,
862, 22]\nclass = b'),
 Text(0.35, 0.08333333333333333, 'gini = 0.027\nsamples = 498\nvalue = [3, 787, 8]\nclass
= b'),
 Text(0.3833333333333336, 0.08333333333333333, 'gini = 0.265\nsamples = 56\nvalue = [0,
75, 14]\nclass = b'),
 Text(0.43333333333333335, 0.25, 'EBE <= 0.575\ngini = 0.429\nsamples = 283\nvalue = [0,
298, 135]\nclass = b'),
 Text(0.4166666666666667, 0.08333333333333333, 'gini = 0.04\nsamples = 126\nvalue = [0, 1
92, 4]\nclass = b'),
```

```
 Text(0.45, 0.08333333333333333, 'gini = 0.494\nsamples = 157\nvalue = [0, 106, 131]\ncla
ss = c'),
 Text(0.7333333333333333, 0.75, 'TOL <= 7.315\ngini = 0.632\nsamples = 7953\nvalue = [511
1, 2278, 5240]\nclass = c'),
 Text(0.6, 0.5833333333333334, 'OXY <= 1.005\ngini = 0.623\nsamples = 5696\nvalue = [2403
, 2102, 4557]\nclass = c'),
 Text(0.5333333333333333, 0.4166666666666667, 'OXY <= 0.595\ngini = 0.659\nsamples = 2661
\nvalue = [1200, 1698, 1304]\nclass = b'),
 Text(0.5, 0.25, 'O_3 <= 8.475\ngini = 0.387\nsamples = 560\nvalue = [84, 672, 123]\nclas
s = b'),
 Text(0.48333333333333334, 0.08333333333333333, 'gini = 0.134\nsamples = 144\nvalue = [6,
210, 10]\nclass = b'),
 Text(0.5166666666666667, 0.08333333333333333, 'gini = 0.455\nsamples = 416\nvalue = [78,
462, 113]\nclass = b'),
 Text(0.5666666666666667, 0.25, 'BEN <= 0.575\ngini = 0.666\nsamples = 2101\nvalue = [111
6, 1026, 1181]\nclass = c'),
 Text(0.55, 0.08333333333333333, 'gini = 0.572\nsamples = 834\nvalue = [117, 543, 666]\nc
lass = c'),
 Text(0.5833333333333334, 0.08333333333333333, 'gini = 0.625\nsamples = 1267\nvalue = [99
9, 483, 515]\nclass = a'),
 Text(0.6666666666666666, 0.4166666666666667, 'TOL <= 4.575\ngini = 0.484\nsamples = 3035
\nvalue = [1203, 404, 3253]\nclass = c'),
 Text(0.6333333333333333, 0.25, 'SO_2 <= 7.925\ngini = 0.315\nsamples = 1566\nvalue = [26
1, 198, 2053]\nclass = c'),
 Text(0.6166666666666667, 0.08333333333333333, 'gini = 0.115\nsamples = 772\nvalue = [49,
26, 1168]\nclass = c'),
 Text(0.65, 0.08333333333333333, 'gini = 0.467\nsamples = 794\nvalue = [212, 172, 885]\nc
lass = c'),
 Text(0.7, 0.25, 'EBE <= 1.755\ngini = 0.57\nsamples = 1469\nvalue = [942, 206, 1200]\ncl
ass = c'),
 Text(0.6833333333333333, 0.08333333333333333, 'gini = 0.57\nsamples = 1199\nvalue = [877
, 153, 881]\nclass = c'),
 Text(0.7166666666666667, 0.08333333333333333, 'gini = 0.43\nsamples = 270\nvalue = [65,
53, 319]\nclass = c'),
 Text(0.8666666666666667, 0.5833333333333334, 'NMHC <= 0.285\ngini = 0.385\nsamples = 225
7\nvalue = [2708, 176, 683]\nclass = a'),
 Text(0.8, 0.4166666666666667, 'TOL <= 9.195\ngini = 0.253\nsamples = 1415\nvalue = [1912
, 50, 273]\nclass = a'),
 Text(0.7666666666666667, 0.25, 'MXY <= 4.685\ngini = 0.406\nsamples = 452\nvalue = [518,
19, 171]\nclass = a'),
 Text(0.75, 0.08333333333333333, 'gini = 0.303\nsamples = 381\nvalue = [493, 17, 91]\ncla
ss = a'),
 Text(0.7833333333333333, 0.08333333333333333, 'gini = 0.386\nsamples = 71\nvalue = [25,
2, 80]\nclass = c'),
 Text(0.8333333333333334, 0.25, 'TCH <= 1.615\ngini = 0.162\nsamples = 963\nvalue = [1394
, 31, 102]\nclass = a'),
 Text(0.8166666666666667, 0.08333333333333333, 'gini = 0.111\nsamples = 860\nvalue = [129
5, 16, 64]\nclass = a'),
 Text(0.85, 0.08333333333333333, 'gini = 0.504\nsamples = 103\nvalue = [99, 15, 38]\nclas
s = a'),
 Text(0.9333333333333333, 0.4166666666666667, 'BEN <= 2.215\ngini = 0.539\nsamples = 842\
nvalue = [796, 126, 410]\nclass = a'),
 Text(0.9, 0.25, 'PM10 <= 23.97\ngini = 0.575\nsamples = 281\nvalue = [83, 106, 260]\ncla
ss = c'),
 Text(0.8833333333333333, 0.08333333333333333, 'gini = 0.625\nsamples = 41\nvalue = [32,
15, 17]\nclass = a'),
 Text(0.9166666666666666, 0.08333333333333333, 'gini = 0.528\nsamples = 240\nvalue = [51,
91, 243]\nclass = c'),
 Text(0.9666666666666667, 0.25, 'BEN <= 3.155\ngini = 0.319\nsamples = 561\nvalue = [713,
20, 150]\nclass = a'),
 Text(0.95, 0.08333333333333333, 'gini = 0.482\nsamples = 198\nvalue = [190, 15, 85]\ncla
ss = a'),
 Text(0.9833333333333333, 0.08333333333333333, 'gini = 0.21\nsamples = 363\nvalue = [523,
5, 65]\nclass = a')]
```





Conclusion

Accuracy

In [63]:

```
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: 0.15290366306738545
Ridge Regression: 0.15276766850837376
Lasso Regression 0.04114787529283537
ElasticNet Regression: 0.09645231136315036
Logistic Regression: 0.794194530061254
Random Forest: 0.8502310802788671
```

Random Forest is suitable for this dataset