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_2004.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
0	2004-08-01 01:00:00	NaN	0.66	NaN	NaN	NaN	89.550003	118.900002	NaN	40.020000	39.990002	25.860001	NaN	12.20
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22.950001	NaN	3.38	6.12
2	2004-08-01 01:00:00	NaN	1.02	NaN	NaN	NaN	93.389999	138.600006	NaN	20.860001	49.480000	NaN	NaN	8.99
3	2004-08-01 01:00:00	NaN	0.53	NaN	NaN	NaN	87.290001	105.000000	NaN	36.730000	31.070000	NaN	NaN	8.82
4	2004-08-01 01:00:00	NaN	0.17	NaN	NaN	NaN	34.910000	35.349998	NaN	86.269997	54.080002	NaN	NaN	8.71
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
245491	2004-06-01 00:00:00	0.75	0.21	0.85	1.55	0.07	59.580002	64.389999	0.66	33.029999	30.900000	14.860000	0.52	6.62
245492	2004-06-01 00:00:00	2.49	0.75	2.44	4.57	NaN	97.139999	146.899994	2.34	7.740000	37.689999	NaN	2.35	6.92
245493	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.13	102.699997	132.600006	NaN	17.809999	22.840000	12.040000	NaN	7.82
245494	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.09	82.599998	102.599998	NaN	NaN	45.630001	NaN	NaN	5.53
245495	2004-06-01 00:00:00	3.01	0.67	2.78	5.12	0.20	92.550003	141.000000	2.60	11.460000	24.389999	17.959999	2.29	8.68

245496 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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     19397 non-null  object 
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	0.63	28079006
22	0.36	28079024
26	0.46	28079099
32	0.67	28079006
49	0.30	28079024
...	...	...
245463	0.08	28079024
245467	0.67	28079099
245473	1.12	28079006
245491	0.21	28079024
245495	0.67	28079099

19397 rows × 2 columns

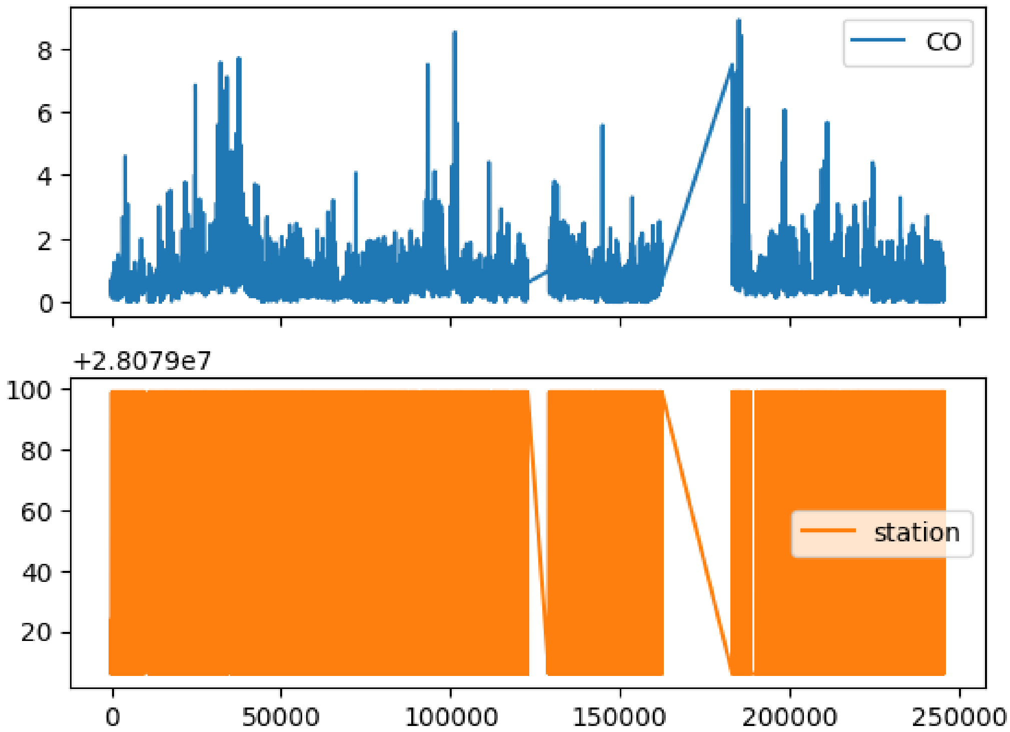
Line chart

In [7]:

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

Out[7]:

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



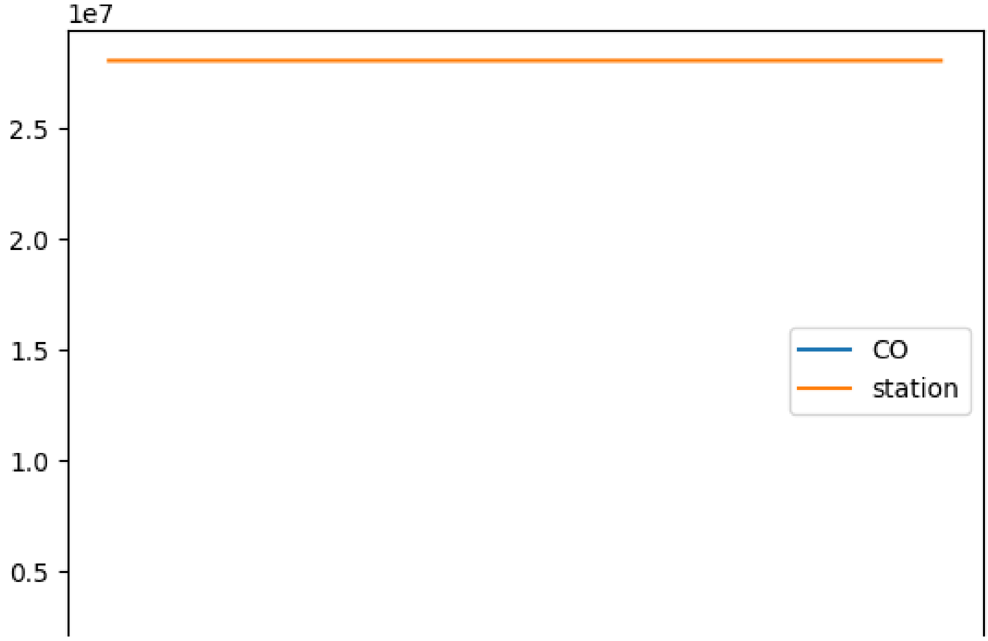
Line chart

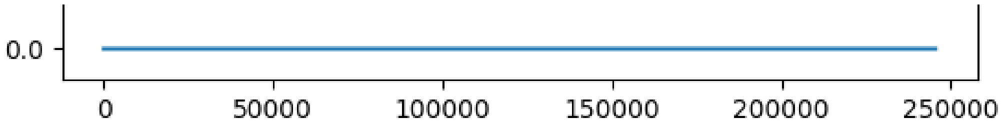
In [8]:

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

Out[8]:

```
<Axes: >
```





Bar chart

In [9]:

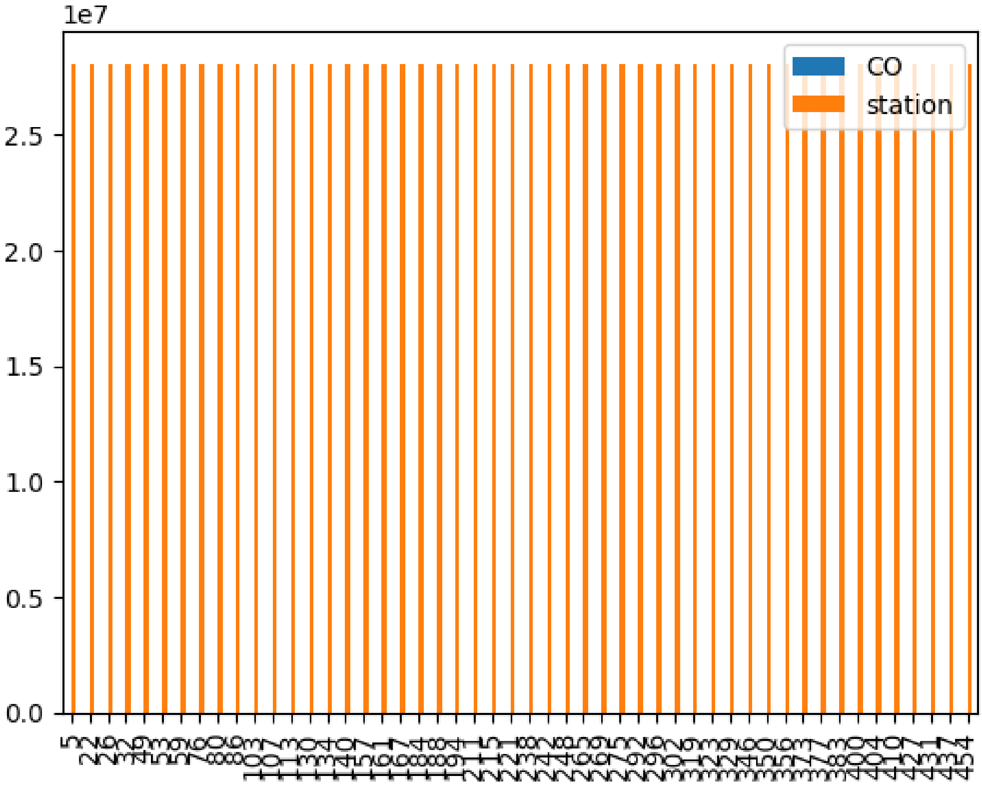
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



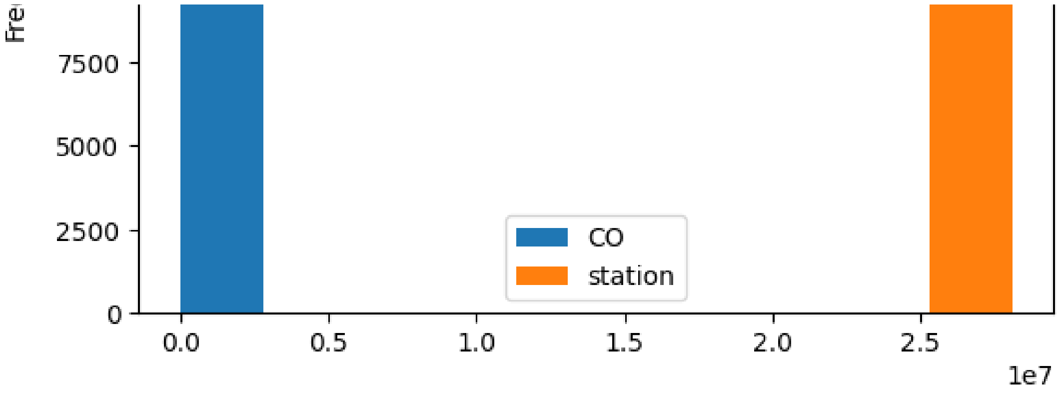
Histogram

In [11]:

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

Out[11]:

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



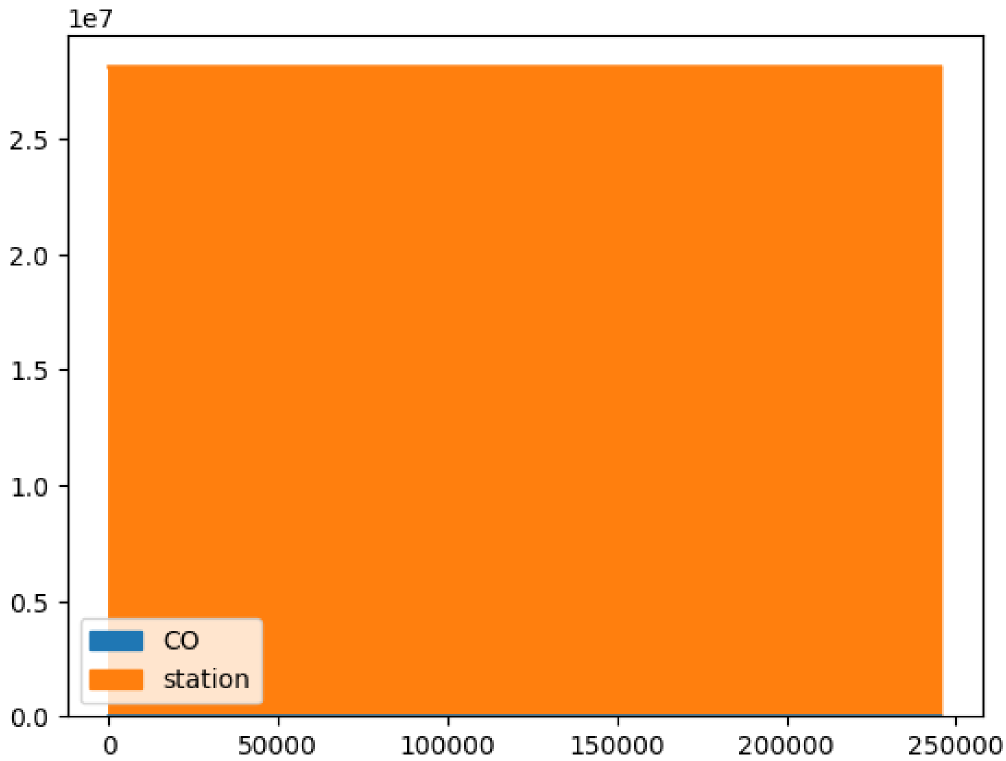
Area chart

In [12]:

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

Out[12]:

```
<Axes: >
```



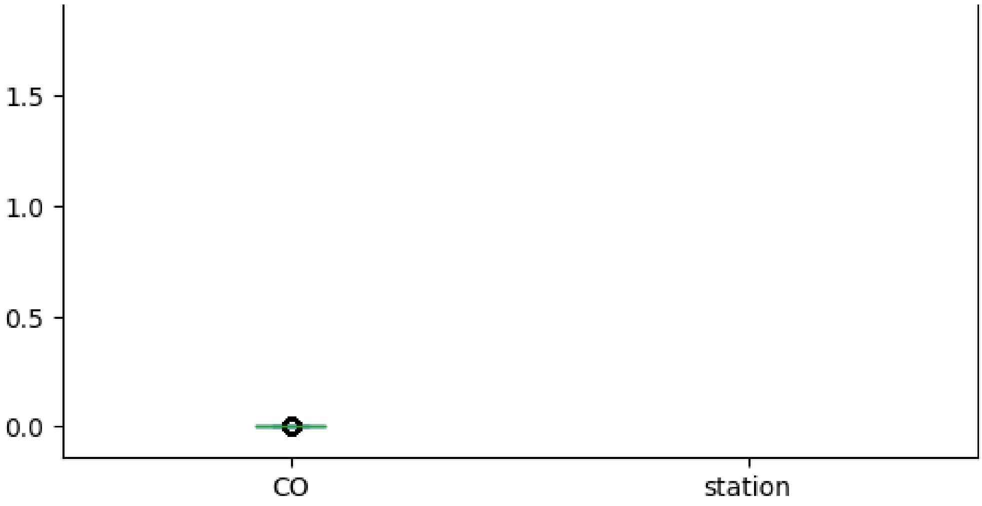
Box chart

In [13]:

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

Out[13]:

```
<Axes: >
```



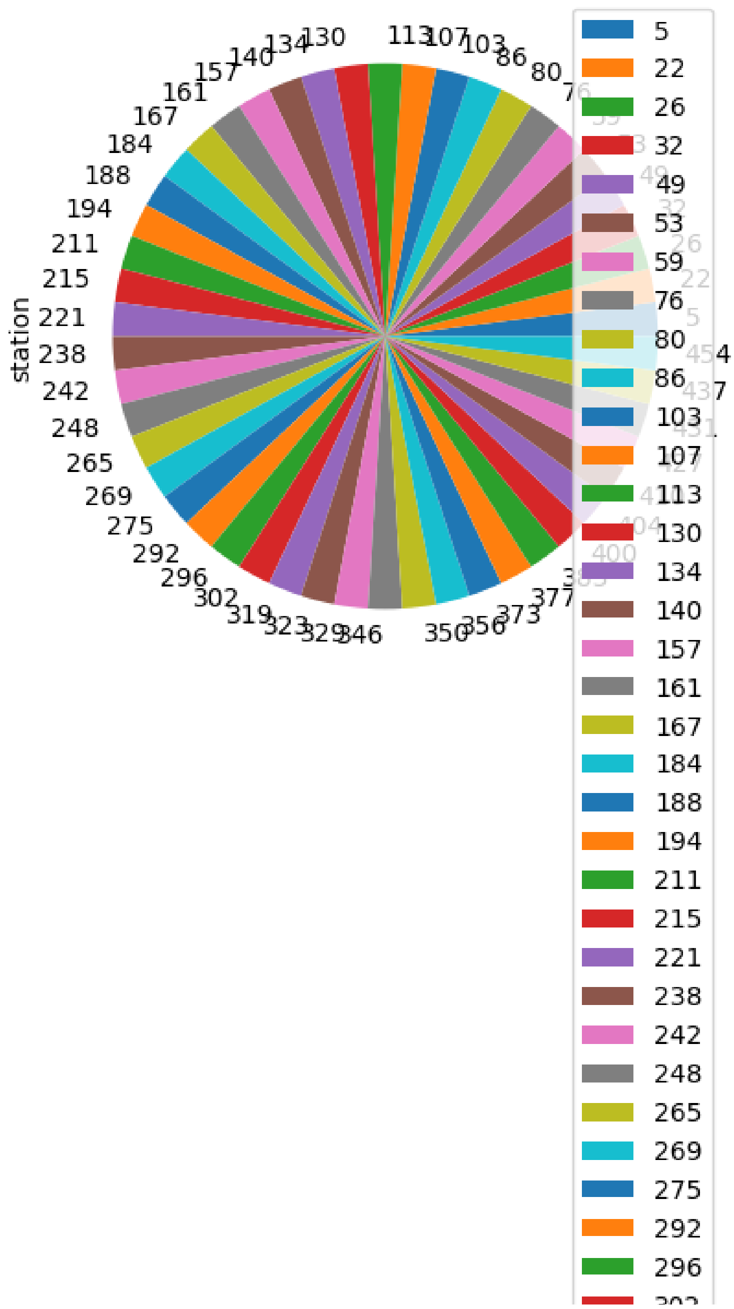
Pie chart

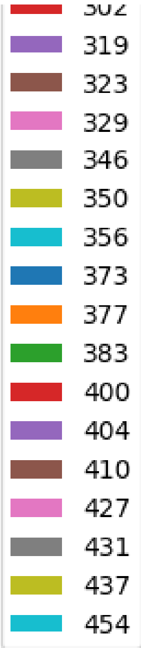
In [14]:

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

Out[14]:

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





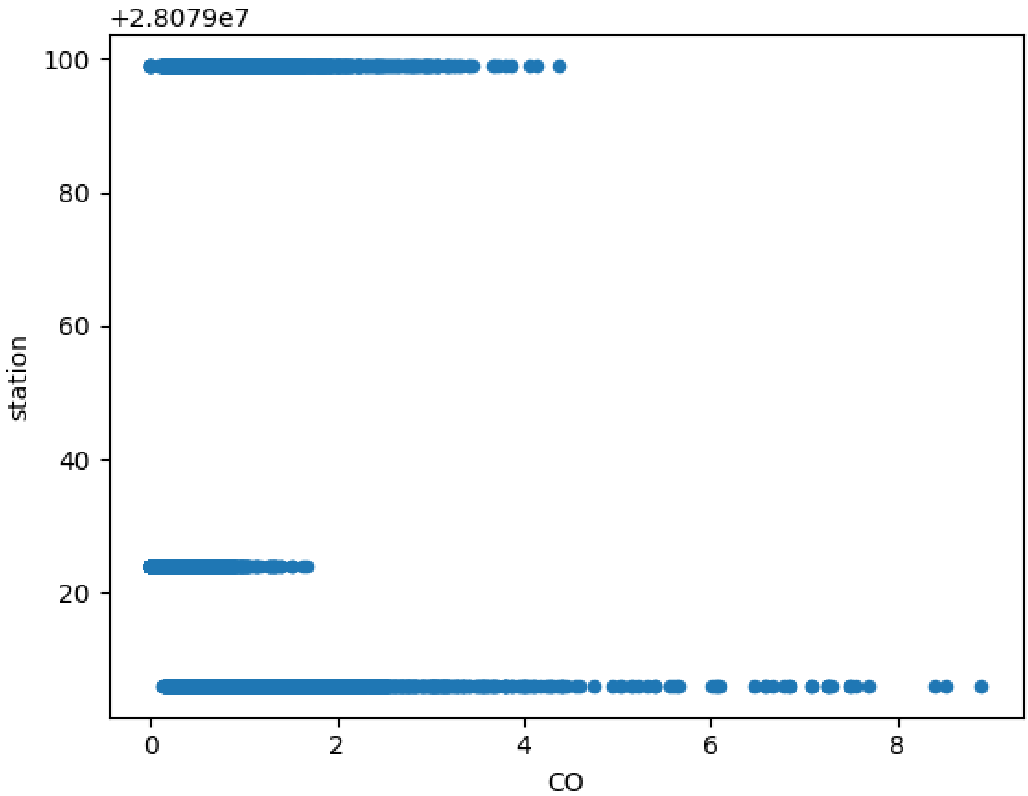
Scatter chart

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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     19397 non-null  object
 1   BEN      19397 non-null  float64
```

```
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.00000	1939
mean	2.250781	0.675347	2.775913	5.424809	0.151024	62.887023	128.554023	2.71421	3
std	2.184724	0.591026	2.729622	5.554358	0.158603	37.952255	127.912411	2.54485	2
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.090000	2.410000	0.00000	
25%	0.870000	0.320000	1.020000	1.780000	0.060000	35.150002	45.209999	1.00000	1
50%	1.620000	0.520000	1.970000	3.800000	0.110000	58.310001	93.220001	1.93000	3
75%	2.910000	0.860000	3.580000	7.260000	0.200000	85.730003	174.300003	3.55000	5
max	34.180000	8.900000	41.880001	91.599998	4.810000	355.100006	1700.000000	45.34000	19

In [18]:

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

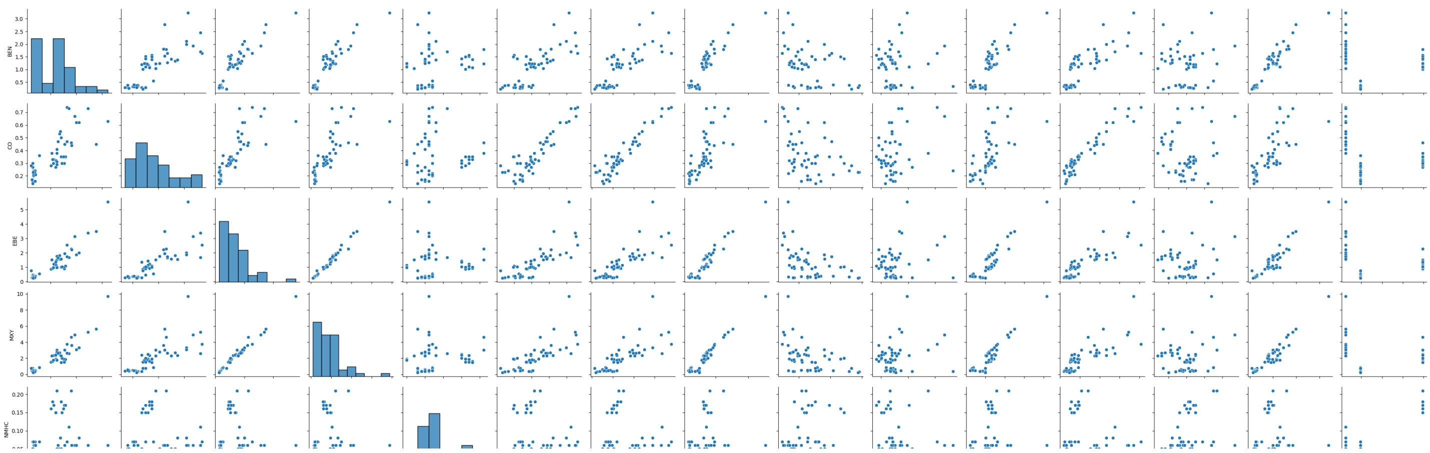
EDA AND VISUALIZATION

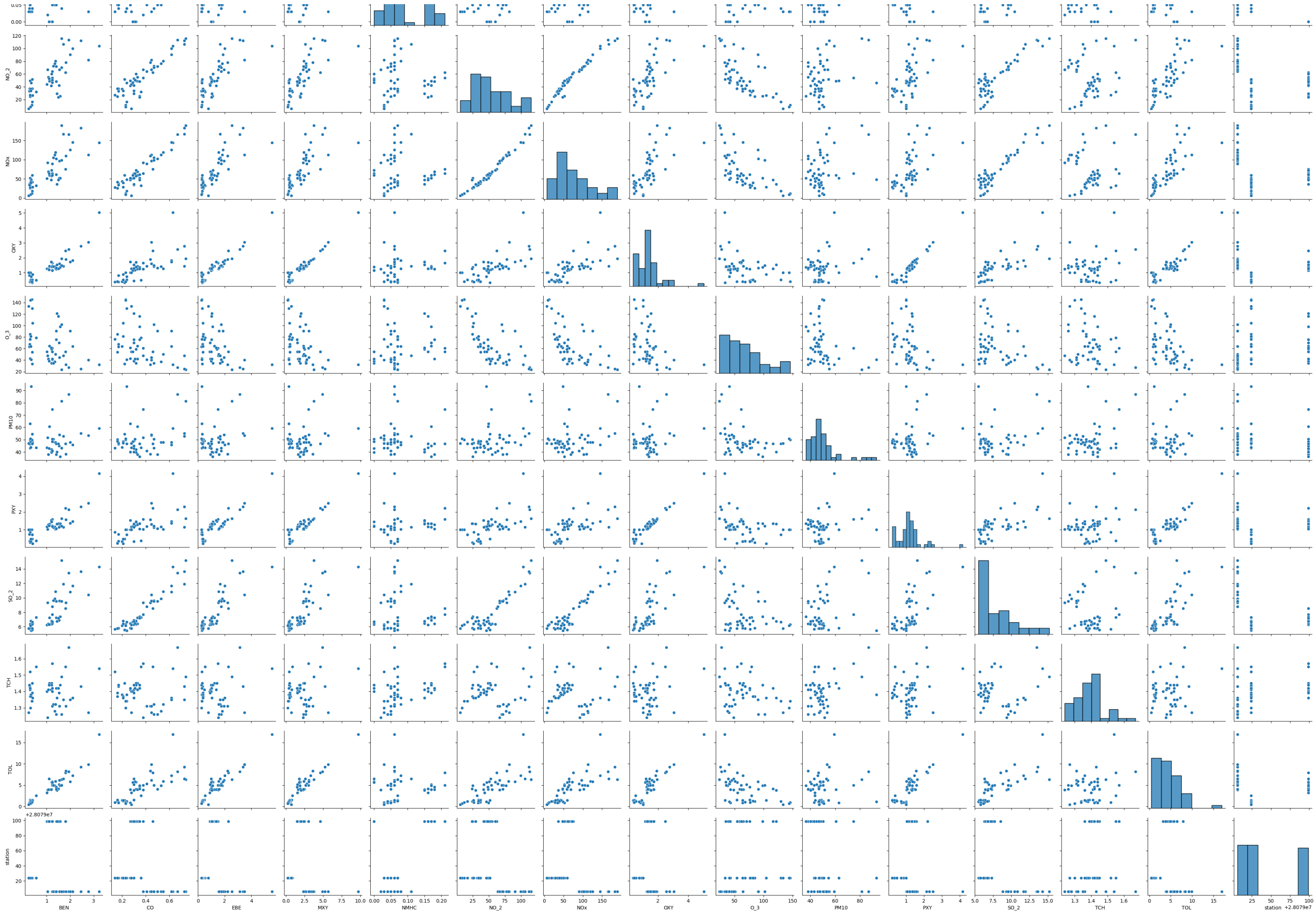
In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x7907cc907ee0>





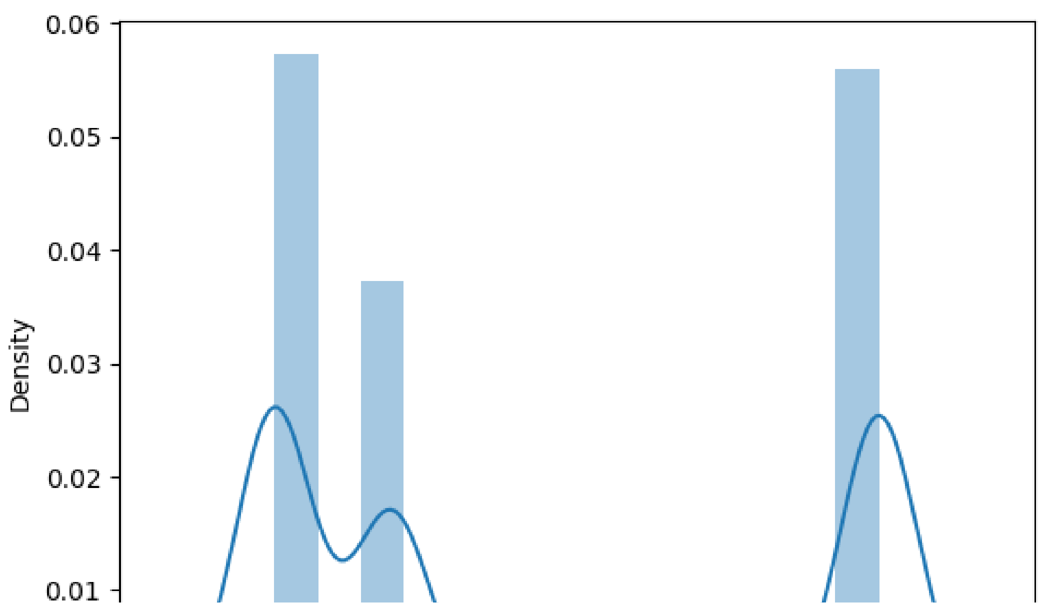
In [20]:

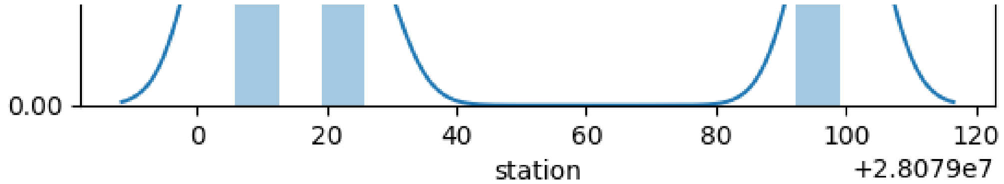
```
sns.distplot(df1['station'])
```

```
<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'>
```



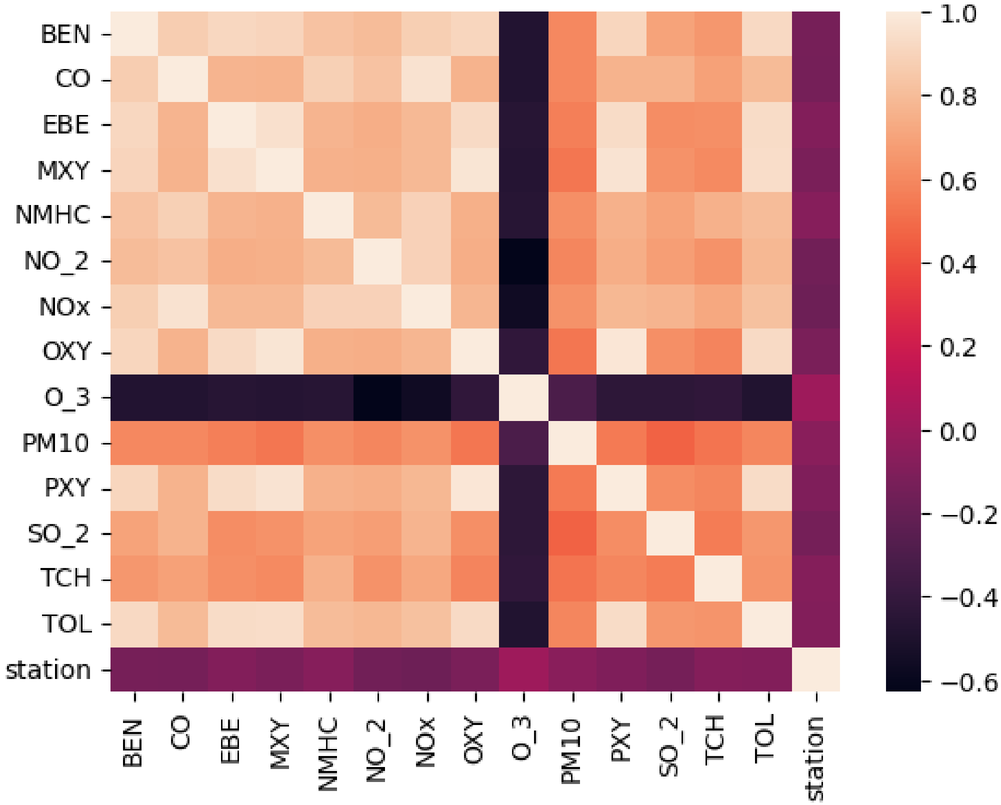


In [21]:

```
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]:

```
28079072.389001545
```

In [26]:

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

Out[26]:

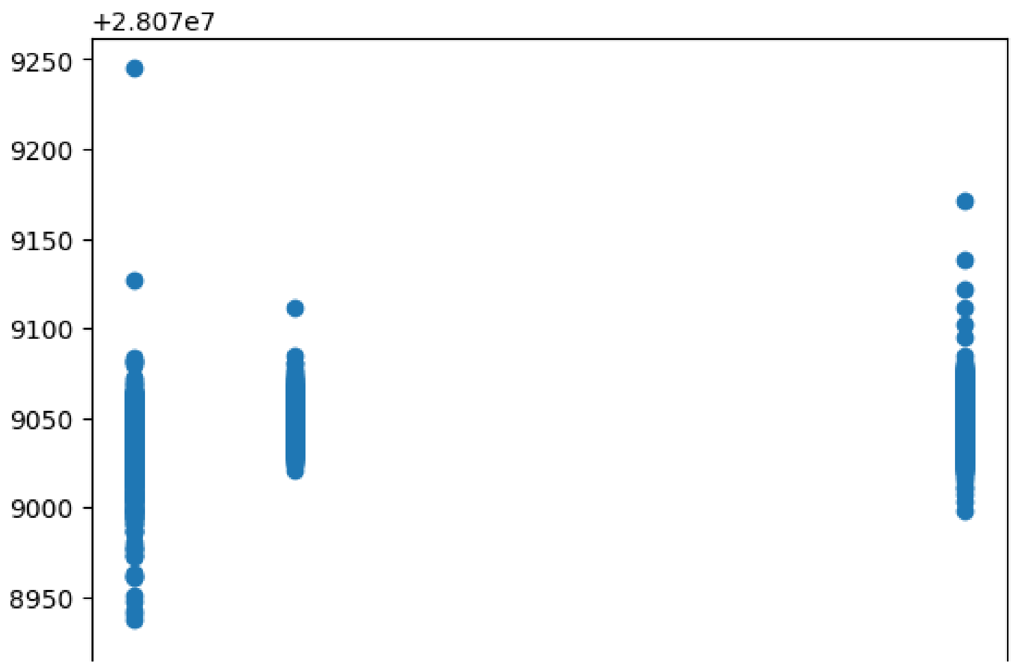
	Co-efficient
BEN	-4.078883
CO	28.022564
EBE	3.428224
MXY	-2.932984
NMHC	83.650557
NO_2	-0.133557
NOx	-0.269837
OXY	-1.446672
O_3	-0.292951
PM10	0.096099
PXY	4.588427
SO_2	-0.175789
TCH	-4.784355
TOL	1.023210

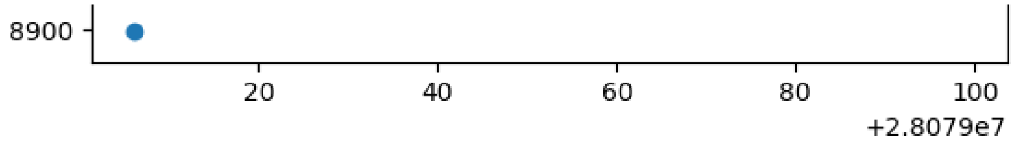
In [27]:

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

Out[27]:

```
<matplotlib.collections.PathCollection at 0x7907b5d9fb20>
```





ACCURACY

In [28]:

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

Out[28]:

0.11581401716609241

In [29]:

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

Out[29]:

0.10221719202092838

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]:

0.11416036631430615

In [33]:

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

Out[33]:

0.10185220734297917

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]:

```
0.05291859334666238
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.05492062256231334
```

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.        ,  0.43510628,  1.39905496, -1.63370056,  0.        ,
       -0.15701499, -0.09492158, -0.        , -0.22257975,  0.12384369,
        0.27490777, -0.09570018,  0.        ,  1.06168663])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079066.16218218
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.0703349520640485
```

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)))
```

```
38.48806374086726
1647.647197059977
40.59122068945423
```

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]:

```
(19397, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(19397,)
```

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)
```

```
[28079006]
```

In [52]:

```
logr.classes_
```

Out[52]:

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

In [53]:

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

Out[53]:

```
0.7360416559261741
```

In [54]:

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

Out[54]:

```
0.9999978255572566
```

In [55]:

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

Out[55]:

```
array([[9.99997826e-01, 7.75018043e-20, 2.17444274e-06]])
```

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]:

```
0.7722625198844573
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

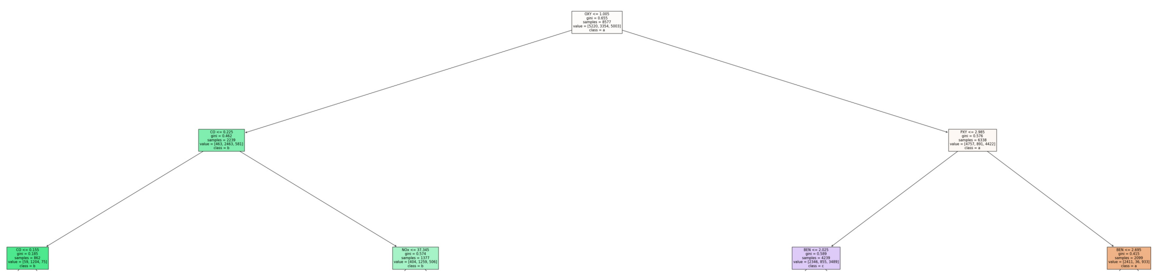
In [62]:

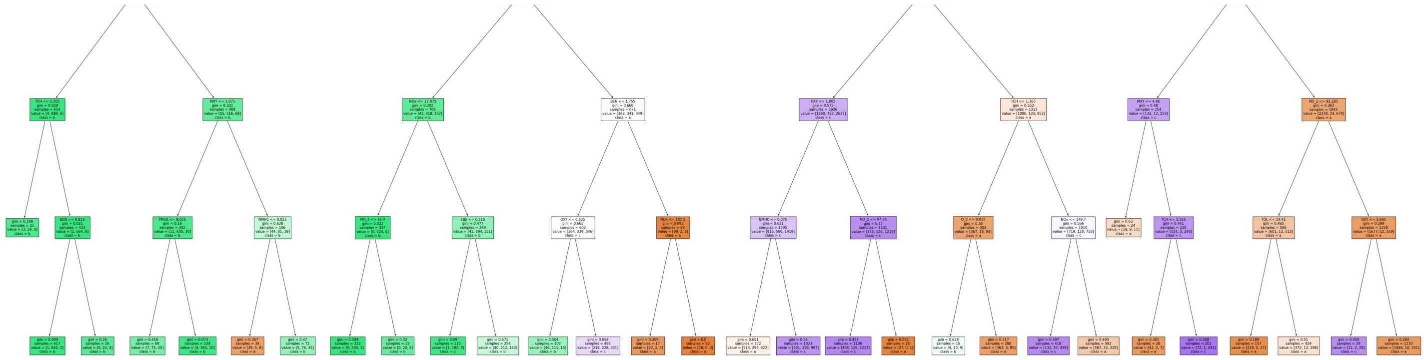
```
from sklearn.tree import plot_tree

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(0.4956140350877193, 0.9166666666666666, 'OXY <= 1.005\ngini = 0.655\nsamples = 8577
\nvalue = [5220, 3354, 5003]\nclass = a'),
 Text(0.2324561403508772, 0.75, 'CO <= 0.225\ngini = 0.462\nsamples = 2239\nvalue = [463,
2463, 581]\nclass = b'),
 Text(0.09649122807017543, 0.5833333333333334, 'CO <= 0.155\ngini = 0.185\nsamples = 862\
nvalue = [59, 1204, 75]\nclass = b'),
 Text(0.03508771929824561, 0.4166666666666667, 'TCH <= 1.205\ngini = 0.028\nsamples = 454
\nvalue = [4, 688, 6]\nclass = b'),
 Text(0.017543859649122806, 0.25, 'gini = 0.198\nsamples = 21\nvalue = [3, 24, 0]\nclass
= b'),
 Text(0.05263157894736842, 0.25, 'BEN <= 0.915\ngini = 0.021\nsamples = 433\nvalue = [1,
664, 6]\nclass = b'),
 Text(0.03508771929824561, 0.08333333333333333, 'gini = 0.009\nsamples = 417\nvalue = [1,
642, 2]\nclass = b'),
 Text(0.07017543859649122, 0.08333333333333333, 'gini = 0.26\nsamples = 16\nvalue = [0, 2
2, 4]\nclass = b'),
 Text(0.15789473684210525, 0.4166666666666667, 'MXY <= 1.075\ngini = 0.331\nsamples = 408
\nvalue = [55, 516, 69]\nclass = b'),
 Text(0.12280701754385964, 0.25, 'PM10 <= 8.025\ngini = 0.16\nsamples = 302\nvalue = [11,
435, 30]\nclass = b'),
 Text(0.10526315789473684, 0.08333333333333333, 'gini = 0.416\nsamples = 68\nvalue = [7,
75, 20]\nclass = b'),
 Text(0.14035087719298245, 0.08333333333333333, 'gini = 0.073\nsamples = 234\nvalue = [4,
360, 10]\nclass = b'),
 Text(0.19298245614035087, 0.25, 'NMHC <= 0.025\ngini = 0.628\nsamples = 106\nvalue = [44
, 81, 39]\nclass = b'),
 Text(0.17543859649122806, 0.08333333333333333, 'gini = 0.367\nsamples = 34\nvalue = [39,
5, 6]\nclass = a'),
 Text(0.21052631578947367, 0.08333333333333333, 'gini = 0.47\nsamples = 72\nvalue = [5, 7
6, 33]\nclass = b'),
 Text(0.3684210526315789, 0.5833333333333334, 'NOx <= 37.345\ngini = 0.574\nsamples = 137
7\nvalue = [404, 1259, 506]\nclass = b'),
 Text(0.2982456140350877, 0.4166666666666667, 'NOx <= 17.875\ngini = 0.302\nsamples = 706
\nvalue = [41, 918, 157]\nclass = b'),
 Text(0.2631578947368421, 0.25, 'NO_2 <= 16.4\ngini = 0.022\nsamples = 337\nvalue = [0, 5
24, 6]\nclass = b'),
 Text(0.24561403508771928, 0.08333333333333333, 'gini = 0.004\nsamples = 322\nvalue = [0,
504, 1]\nclass = b'),
 Text(0.2807017543859649, 0.08333333333333333, 'gini = 0.32\nsamples = 15\nvalue = [0, 20
, 5]\nclass = b'),
 Text(0.3333333333333333, 0.25, 'EBE <= 0.515\ngini = 0.477\nsamples = 369\nvalue = [41,
394, 151]\nclass = b'),
 Text(0.3157894736842105, 0.08333333333333333, 'gini = 0.09\nsamples = 115\nvalue = [1, 1
82, 8]\nclass = b'),
 Text(0.3508771929824561, 0.08333333333333333, 'gini = 0.571\nsamples = 254\nvalue = [40,
212, 143]\nclass = b'),
 Text(0.43859649122807015, 0.4166666666666667, 'BEN <= 1.755\ngini = 0.666\nsamples = 671
\nvalue = [363, 341, 349]\nclass = a'),
 Text(0.40350877192982454, 0.25, 'OXY <= 0.625\ngini = 0.662\nsamples = 602\nvalue = [264
, 339, 346]\nclass = c'),
 Text(0.38596491228070173, 0.08333333333333333, 'gini = 0.504\nsamples = 107\nvalue = [46
, 111, 15]\nclass = b'),
 Text(0.42105263157894735, 0.08333333333333333, 'gini = 0.654\nsamples = 495\nvalue = [21
8, 228, 331]\nclass = c'),
 Text(0.47368421052631576, 0.25, 'NOx <= 147.5\ngini = 0.093\nsamples = 69\nvalue = [99,
2, 3]\nclass = a'),
 Text(0.45614035087719296, 0.08333333333333333, 'gini = 0.309\nsamples = 17\nvalue = [23,
```

```
2, 3]\nclass = a'),
 Text(0.49122807017543857, 0.08333333333333333, 'gini = 0.0\nsamples = 52\nvalue = [76, 0
, 0]\nclass = a'),
 Text(0.7587719298245614, 0.75, 'PXY <= 2.985\ngini = 0.576\nsamples = 6338\nvalue = [475
7, 891, 4422]\nclass = a'),
 Text(0.6491228070175439, 0.5833333333333334, 'BEN <= 2.025\ngini = 0.589\nsamples = 4239
\nvalue = [2346, 855, 3489]\nclass = c'),
 Text(0.5789473684210527, 0.4166666666666667, 'OXY <= 1.885\ngini = 0.575\nsamples = 2926
\nvalue = [1260, 722, 2637]\nclass = c'),
 Text(0.543859649122807, 0.25, 'NMHC <= 0.075\ngini = 0.621\nsamples = 1795\nvalue = [815
, 596, 1419]\nclass = c'),
 Text(0.5263157894736842, 0.08333333333333333, 'gini = 0.651\nsamples = 772\nvalue = [514
, 297, 422]\nclass = a'),
 Text(0.5614035087719298, 0.08333333333333333, 'gini = 0.54\nsamples = 1023\nvalue = [301
, 299, 997]\nclass = c'),
 Text(0.6140350877192983, 0.25, 'NO_2 <= 97.26\ngini = 0.47\nsamples = 1131\nvalue = [445
, 126, 1218]\nclass = c'),
 Text(0.5964912280701754, 0.08333333333333333, 'gini = 0.457\nsamples = 1106\nvalue = [40
8, 126, 1217]\nclass = c'),
 Text(0.631578947368421, 0.08333333333333333, 'gini = 0.051\nsamples = 25\nvalue = [37, 0
, 1]\nclass = a'),
 Text(0.7192982456140351, 0.4166666666666667, 'TCH <= 1.365\ngini = 0.552\nsamples = 1313
\nvalue = [1086, 133, 852]\nclass = a'),
 Text(0.6842105263157895, 0.25, 'O_3 <= 8.815\ngini = 0.36\nsamples = 303\nvalue = [367,
13, 94]\nclass = a'),
 Text(0.6666666666666666, 0.08333333333333333, 'gini = 0.628\nsamples = 15\nvalue = [4, 1
0, 9]\nclass = b'),
 Text(0.7017543859649122, 0.08333333333333333, 'gini = 0.317\nsamples = 288\nvalue = [363
, 3, 85]\nclass = a'),
 Text(0.7543859649122807, 0.25, 'NOx <= 149.7\ngini = 0.566\nsamples = 1010\nvalue = [719
, 120, 758]\nclass = c'),
 Text(0.7368421052631579, 0.08333333333333333, 'gini = 0.497\nsamples = 418\nvalue = [132
, 87, 439]\nclass = c'),
 Text(0.7719298245614035, 0.08333333333333333, 'gini = 0.493\nsamples = 592\nvalue = [587
, 33, 319]\nclass = a'),
 Text(0.868421052631579, 0.5833333333333334, 'BEN <= 2.695\ngini = 0.415\nsamples = 2099\
nvalue = [2411, 36, 933]\nclass = a'),
 Text(0.8070175438596491, 0.4166666666666667, 'MXY <= 4.46\ngini = 0.48\nsamples = 254\nv
alue = [133, 12, 259]\nclass = c'),
 Text(0.7894736842105263, 0.25, 'gini = 0.63\nsamples = 24\nvalue = [19, 9, 11]\nclass =
a'),
 Text(0.8245614035087719, 0.25, 'TCH <= 1.355\ngini = 0.441\nsamples = 230\nvalue = [114,
3, 248]\nclass = c'),
 Text(0.8070175438596491, 0.08333333333333333, 'gini = 0.301\nsamples = 28\nvalue = [42,
2, 7]\nclass = a'),
 Text(0.8421052631578947, 0.08333333333333333, 'gini = 0.358\nsamples = 202\nvalue = [72,
1, 241]\nclass = c'),
 Text(0.9298245614035088, 0.4166666666666667, 'NO_2 <= 91.205\ngini = 0.363\nsamples = 18
45\nvalue = [2278, 24, 674]\nclass = a'),
 Text(0.8947368421052632, 0.25, 'TOL <= 14.41\ngini = 0.465\nsamples = 586\nvalue = [601,
12, 315]\nclass = a'),
 Text(0.8771929824561403, 0.08333333333333333, 'gini = 0.189\nsamples = 157\nvalue = [229
, 0, 27]\nclass = a'),
 Text(0.9122807017543859, 0.08333333333333333, 'gini = 0.51\nsamples = 429\nvalue = [372,
12, 288]\nclass = a'),
 Text(0.9649122807017544, 0.25, 'OXY <= 3.685\ngini = 0.299\nsamples = 1259\nvalue = [167
7, 12, 359]\nclass = a'),
 Text(0.9473684210526315, 0.08333333333333333, 'gini = 0.459\nsamples = 29\nvalue = [11,
2, 28]\nclass = c'),
 Text(0.9824561403508771, 0.08333333333333333, 'gini = 0.284\nsamples = 1230\nvalue = [16
66, 10, 331]\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: 0.11581401716609241
Ridge Regression: 0.11416036631430615
Lasso Regression 0.05492062256231334
ElasticNet Regression: 0.07033495206404850
Logistic Regression: 0.7360416559261741
Random Forest: 0.7722625198844573
```

Random Forest is suitable for this dataset