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_2006.csv")
df
```

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

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	S
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.880000	97.570000	40.259998	NaN	33.779
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.820000	NaN	2.48	11.890
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.430000	34.419998	NaN	NaN	19.719
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.830000	28.260000	NaN	NaN	21.129
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.990000	54.180000	NaN	NaN	11.050
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
230563	2006-05-01 00:00:00	5.88	0.83	6.23	NaN	0.20	112.500000	218.000000	NaN	24.389999	93.120003	NaN	NaN	7.400
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29.469999	15.640000	0.50	8.840
230565	2006-05-01 00:00:00	0.96	NaN	0.69	NaN	0.19	135.100006	179.199997	NaN	11.460000	64.680000	35.000000	NaN	12.110
230566	2006-05-01 00:00:00	0.50	NaN	0.67	NaN	0.10	82.599998	105.599998	NaN	NaN	94.360001	NaN	NaN	4.890
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52.490002	27.920000	1.70	9.170

230568 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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24758 non-null  object 
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

In [6]:

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

Out[6]:

	CO	station
5	1.69	28079006
22	0.79	28079024
25	1.47	28079099
31	0.85	28079006
48	0.79	28079024
...	...	...
230538	0.40	28079024
230541	0.94	28079099
230547	1.06	28079006
230564	0.32	28079024
230567	0.74	28079099

24758 rows × 2 columns

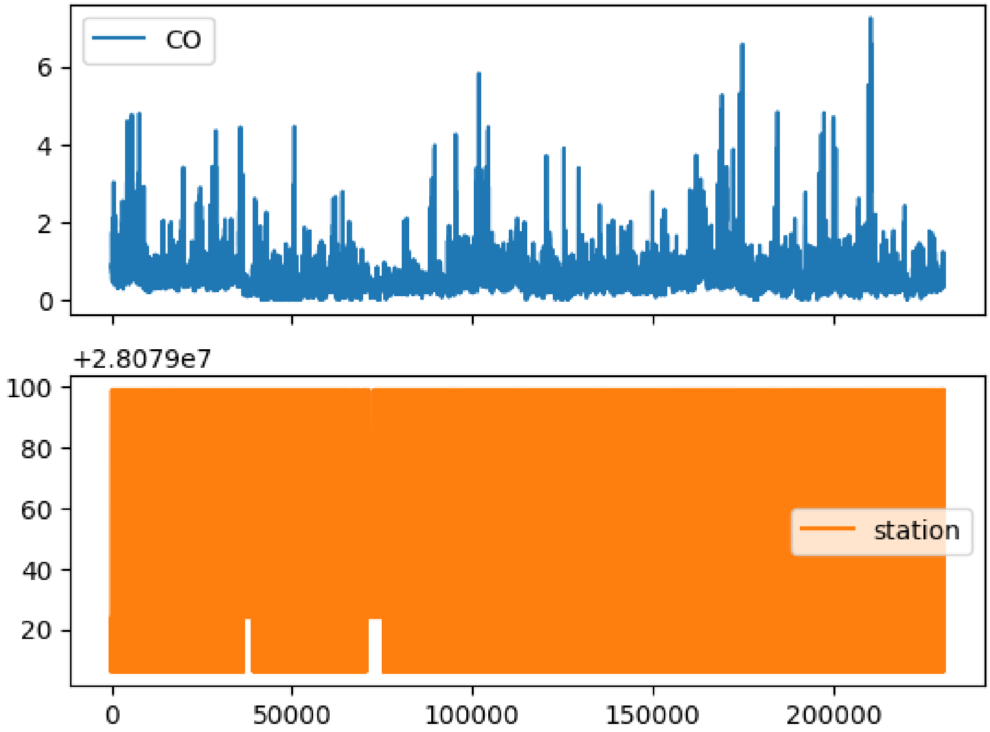
Line chart

In [7]:

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

Out[7]:

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



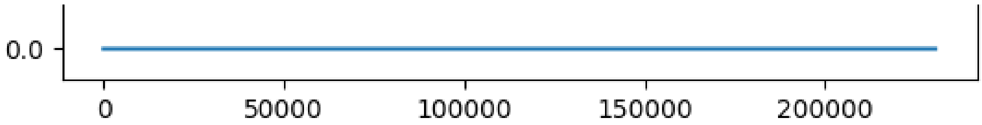
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

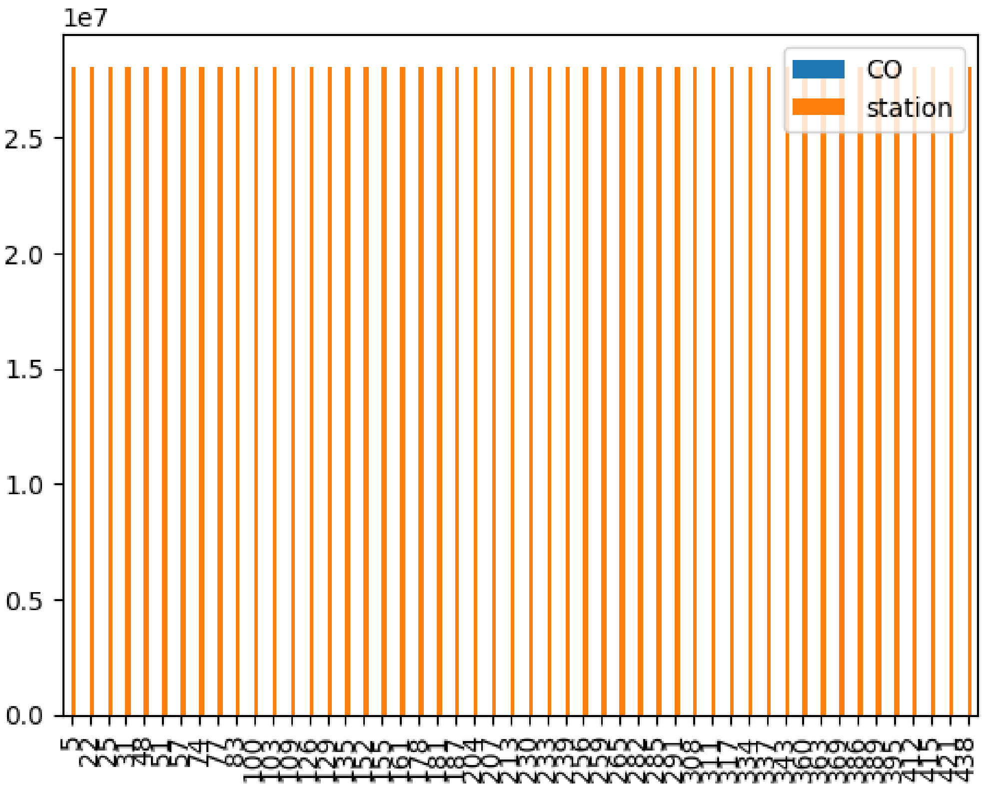
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



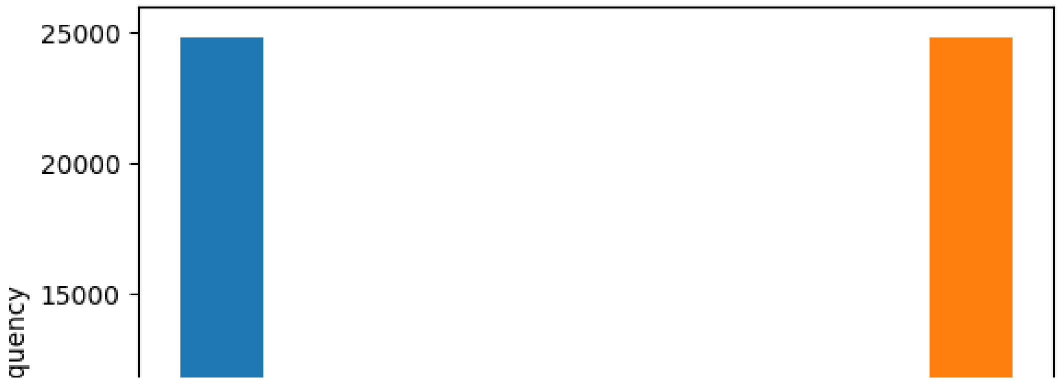
Histogram

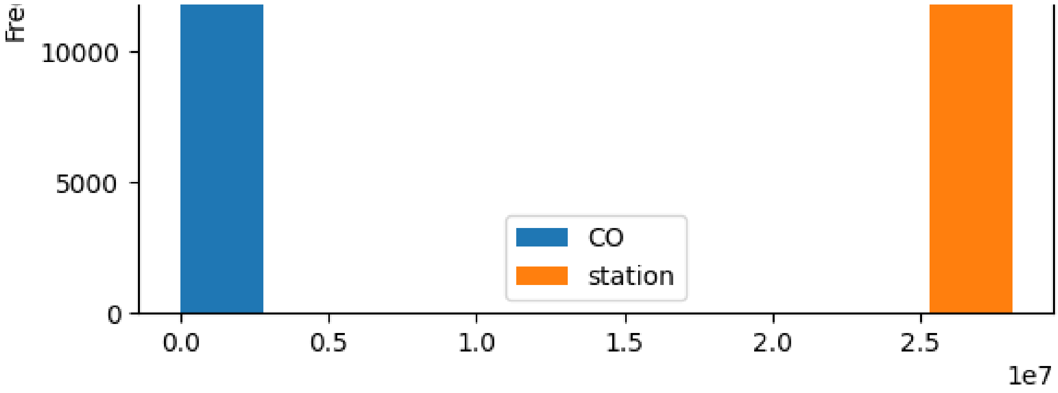
In [11]:

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

Out[11]:

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





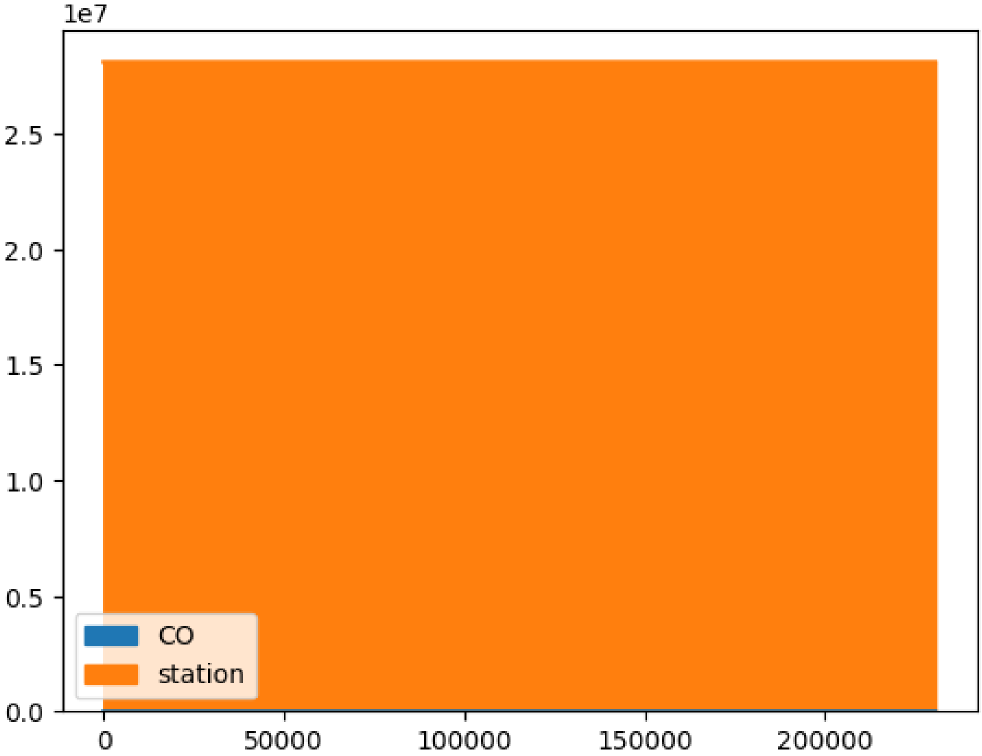
Area chart

In [12]:

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

Out[12]:

<Axes: >



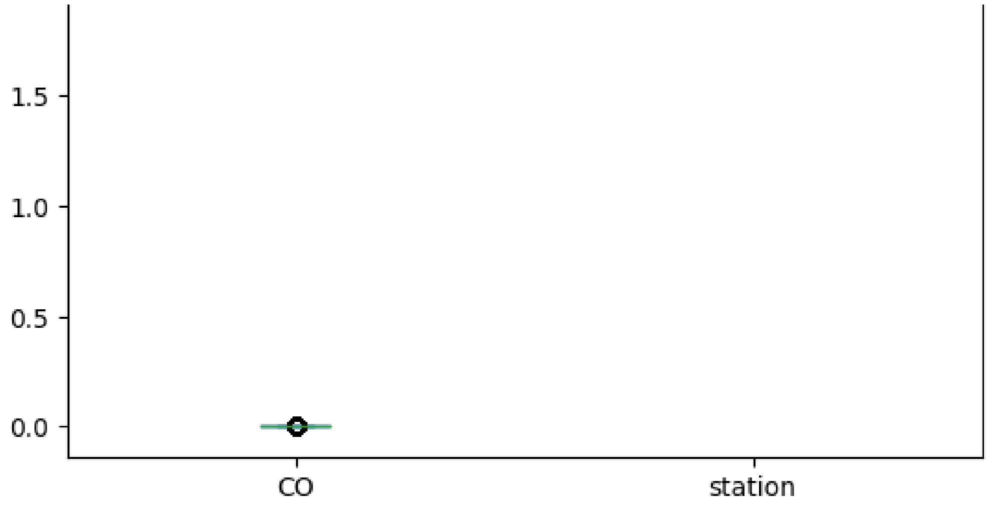
Box chart

In [13]:

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

Out[13]:

<Axes: >



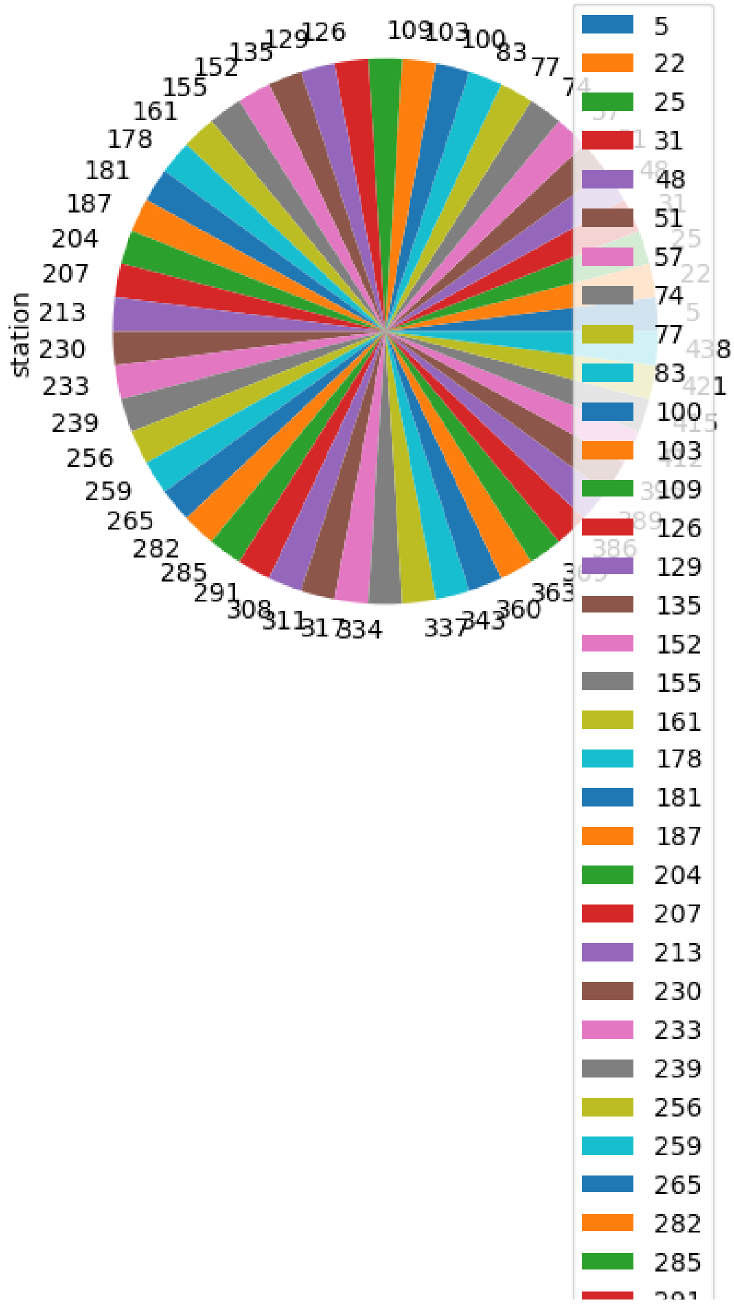
Pie chart

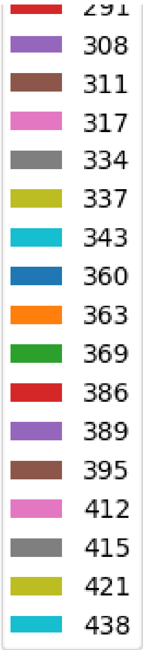
In [14]:

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

Out[14]:

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





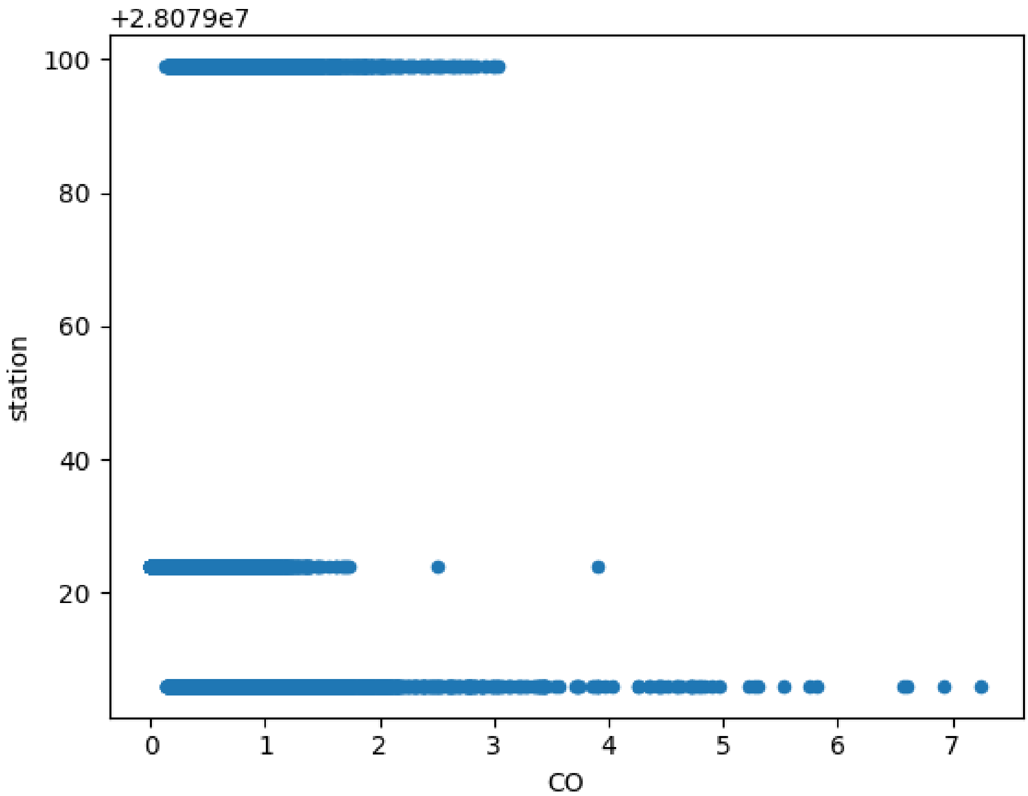
Scatter chart

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: 24758 entries, 5 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     24758 non-null  object
 1   BEN      24758 non-null  float64
```

```
 1   BEN      24758 non-null  float64
 2   CO       24758 non-null  float64
 3   EBE      24758 non-null  float64
 4   MXY      24758 non-null  float64
 5   NMHC     24758 non-null  float64
 6   NO_2     24758 non-null  float64
 7   NOx      24758 non-null  float64
 8   OXY      24758 non-null  float64
 9   O_3      24758 non-null  float64
 10  PM10     24758 non-null  float64
 11  PM25     24758 non-null  float64
 12  PXY      24758 non-null  float64
 13  SO_2     24758 non-null  float64
 14  TCH      24758 non-null  float64
 15  TOL      24758 non-null  float64
 16  station  24758 non-null  int64
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	
count	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	24758.000000	247
mean	1.350624	0.600713	1.824534	3.835034	0.176546	58.333481	116.419090	1.990347	
std	1.541636	0.419048	1.868939	4.069036	0.126683	40.529382	117.557064	1.931620	
min	0.110000	0.000000	0.170000	0.150000	0.000000	1.680000	2.020000	0.190000	
25%	0.450000	0.360000	0.810000	1.060000	0.100000	28.450001	36.882501	0.960000	
50%	0.850000	0.500000	1.130000	2.500000	0.150000	52.959999	85.180000	1.260000	
75%	1.680000	0.720000	2.160000	5.090000	0.220000	79.347498	158.300003	2.470000	
max	45.430000	7.250000	57.799999	66.900002	2.020000	461.299988	1680.000000	63.000000	1

In [18]:

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

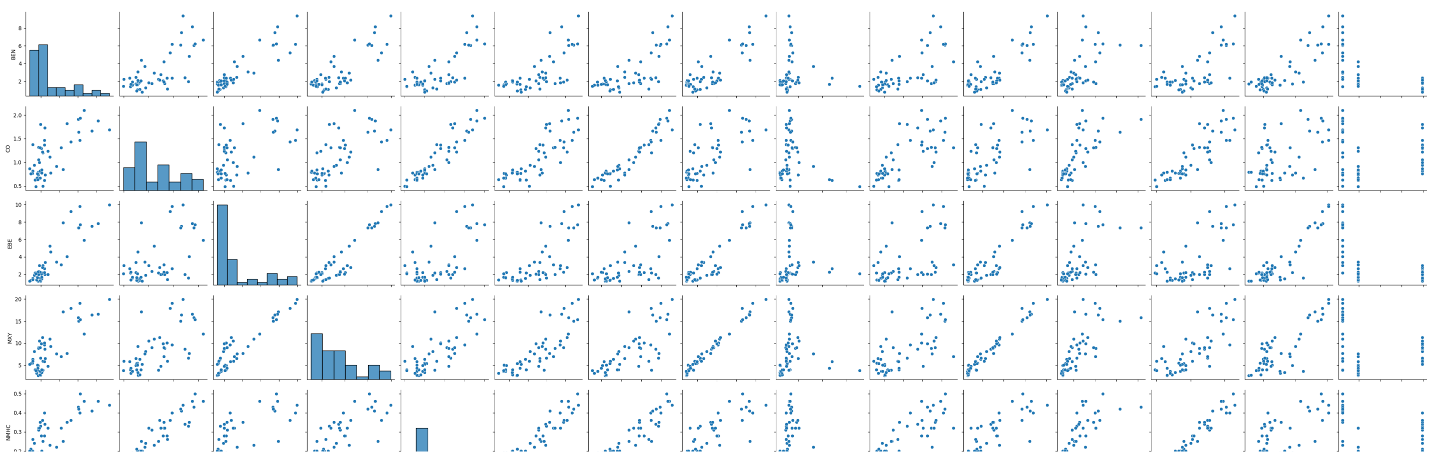
EDA AND VISUALIZATION

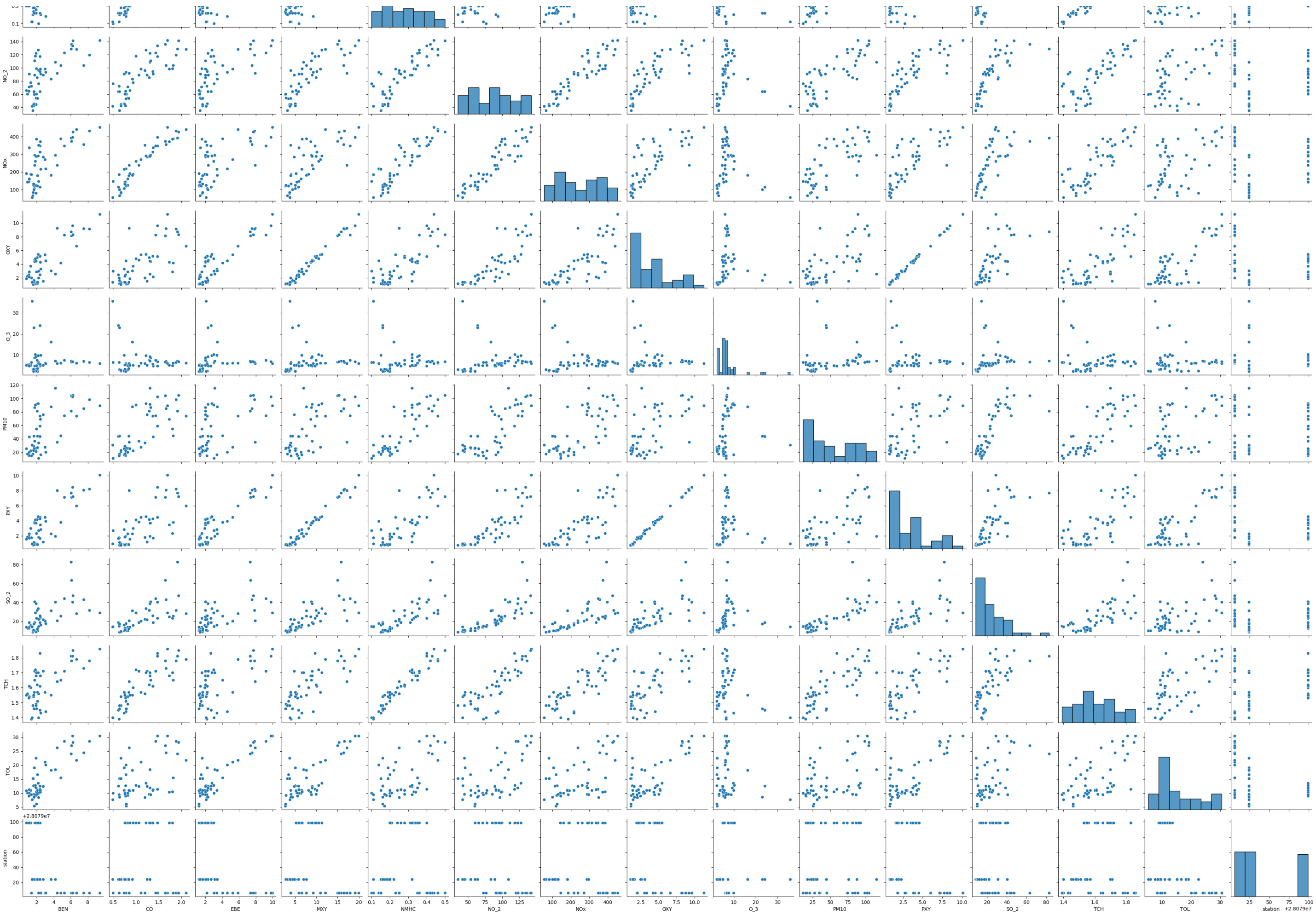
In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x79a28cf7d240>





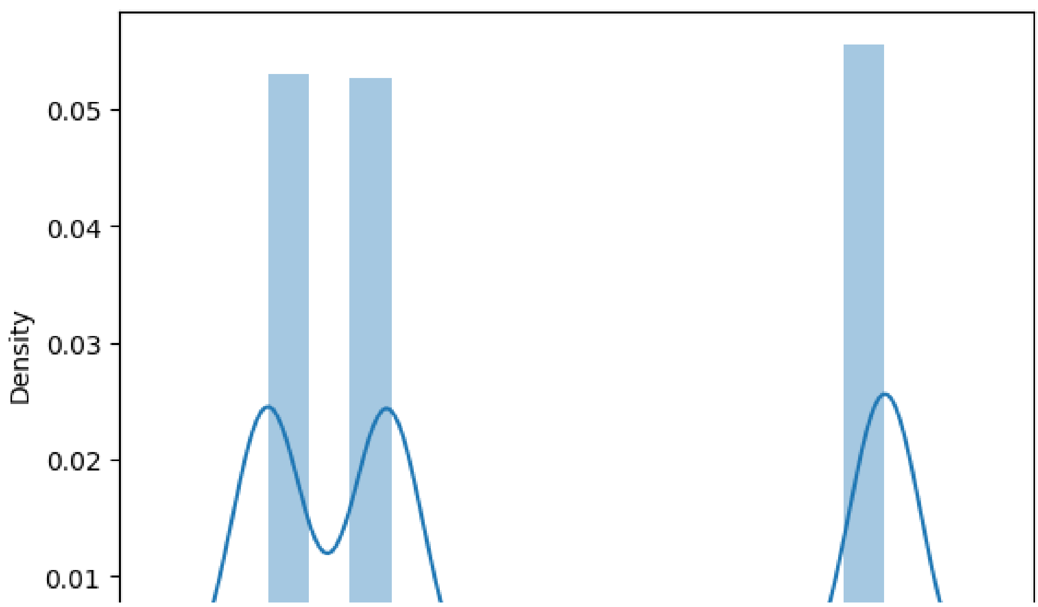
In [20]:

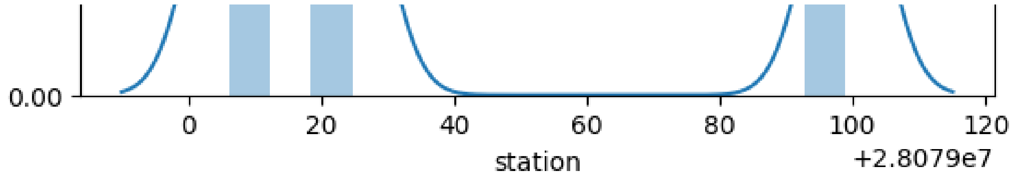
```
sns.distplot(df1['station'])
```

```
<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'>
```



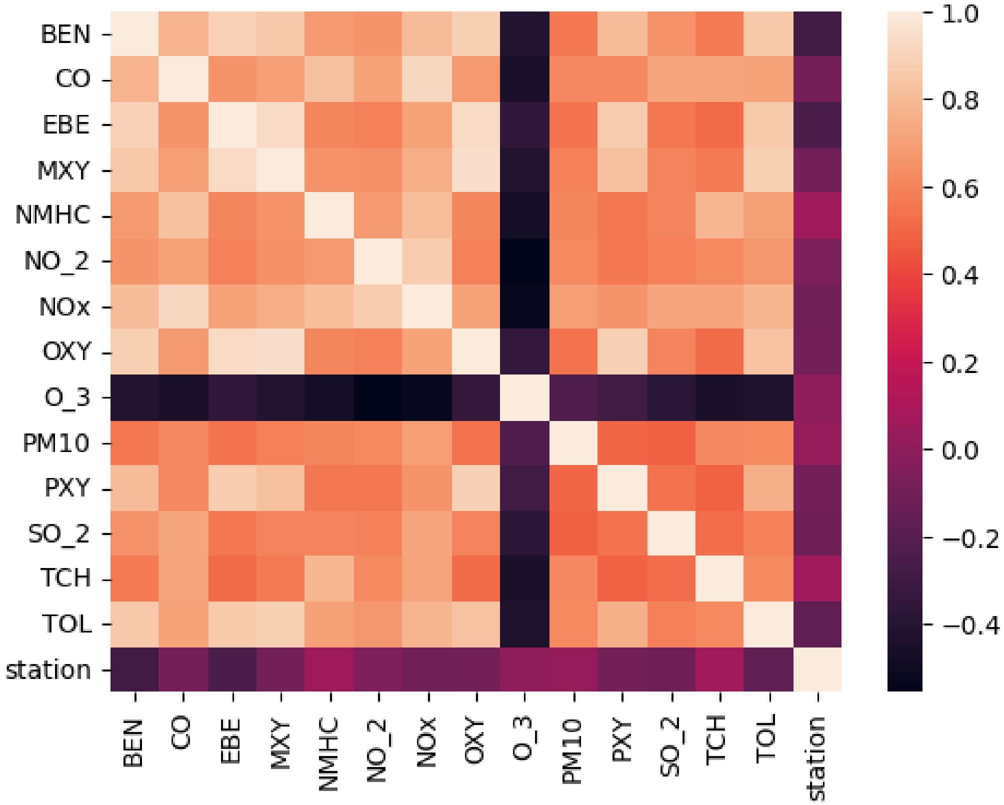


In [21]:

```
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]:

28079016.942035887

In [26]:

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

Out[26]:

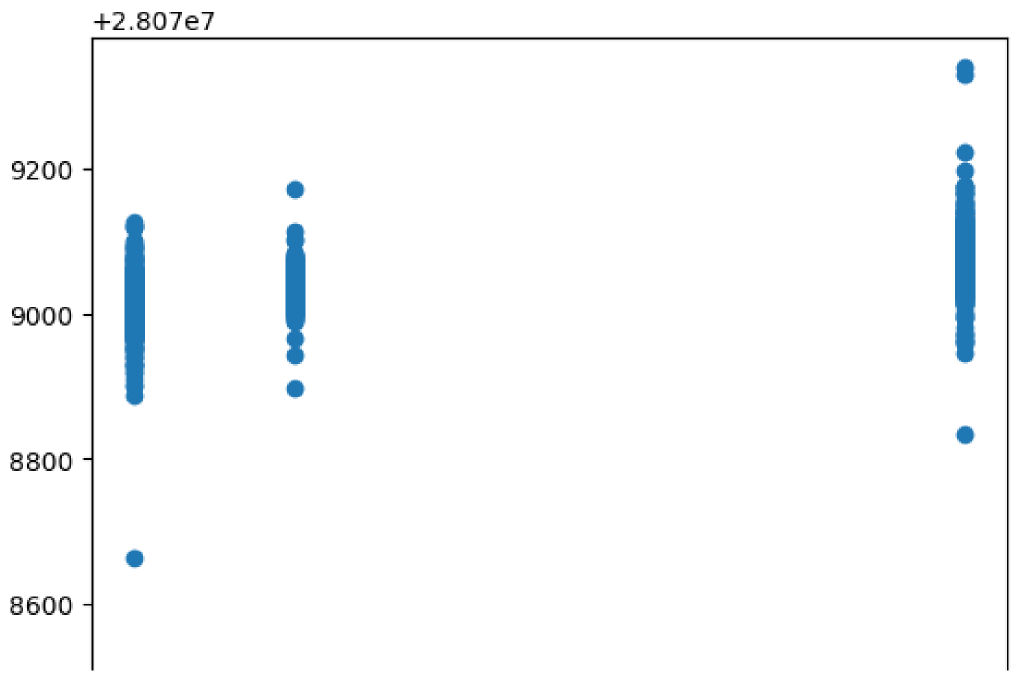
	Co-efficient
BEN	-18.937235
CO	-11.166994
EBE	-23.638521
MXY	3.451071
NMHC	126.463265
NO_2	-0.014745
NOx	-0.000677
OXY	17.358327
O_3	-0.052211
PM10	0.134200
PXY	6.232581
SO_2	-0.656112
TCH	20.991331
TOL	-0.397456

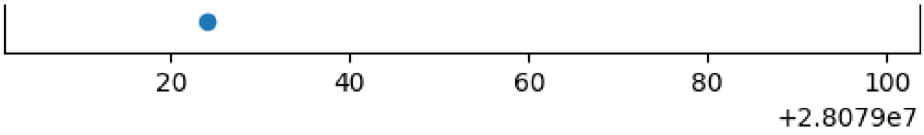
In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x79a276472530>





ACCURACY

In [28]:

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

Out[28]:

```
0.3826824977651868
```

In [29]:

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

Out[29]:

```
0.39833084657877427
```

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]:

```
0.3815141911316644
```

In [33]:

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

Out[33]:

```
0.39770107390932086
```

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]:

```
0.05963442714373823
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.06316435993388725
```

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([-8.69777368e+00,  0.00000000e+00, -8.82080365e+00,  3.38577972e+00,
        4.21564999e-01, -6.12881145e-03,  7.71205703e-03,  3.37014759e+00,
       -1.14245129e-01,  3.00779623e-01,  2.60695492e+00, -4.63953992e-01,
        5.76730869e-01, -1.04207930e+00])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079051.89303524
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.24028188740354617
```

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)))
```

```
32.15898898229611
1249.1604026955306
35.34346336588324
```

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]:

```
(24758, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(24758,)
```

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]:

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

In [53]:

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

Out[53]:

```
0.8741416915744405
```

In [54]:

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

Out[54]:

```
3.555768961192919e-15
```

In [55]:

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

Out[55]:

```
array([[3.55576896e-15, 7.80741341e-29, 1.00000000e+00]])
```

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]:

```
0.8762261973456433
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

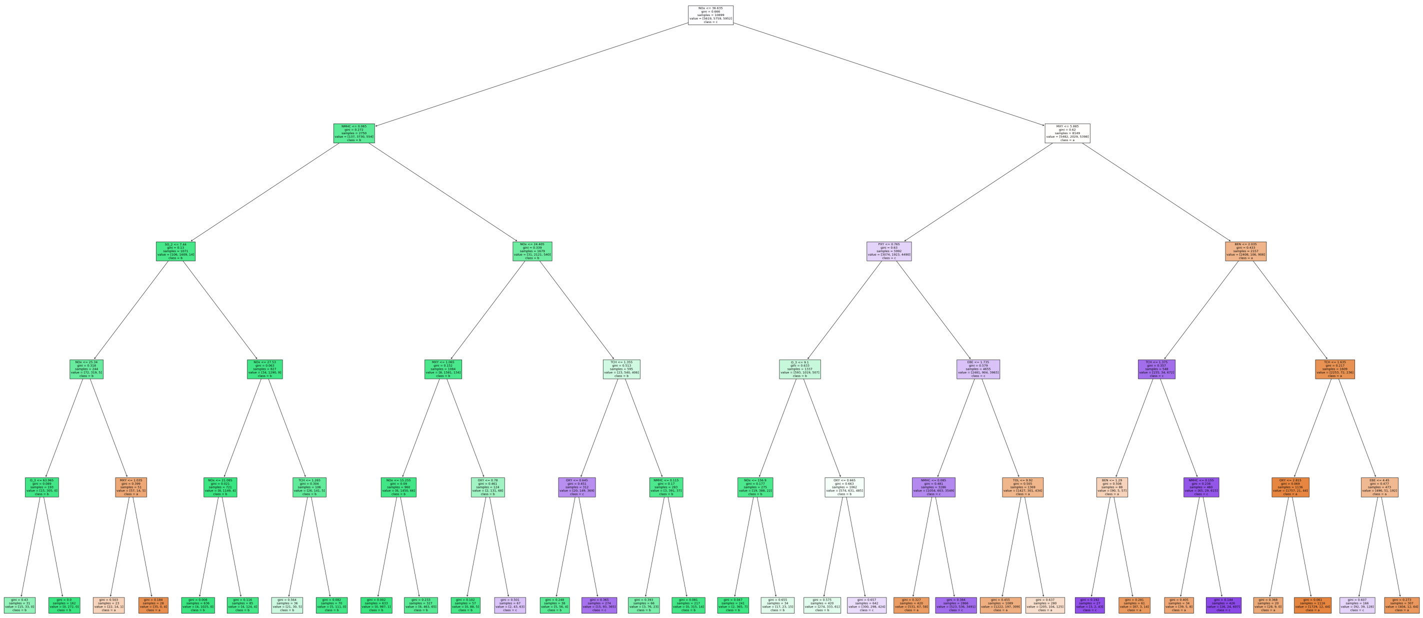
In [62]:

```
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(0.5, 0.9166666666666666, 'NOx <= 36.635\ngini = 0.666\nsamples = 10899\nvalue = [56
19, 5759, 5952]\nclass = c'),
 Text(0.25, 0.75, 'NMHC <= 0.065\ngini = 0.272\nsamples = 2750\nvalue = [137, 3730, 554]\
nclass = b'),
 Text(0.125, 0.5833333333333334, 'SO_2 <= 7.44\ngini = 0.13\nsamples = 1071\nvalue = [106
, 1609, 14]\nclass = b'),
 Text(0.0625, 0.4166666666666667, 'NOx <= 25.34\ngini = 0.318\nsamples = 244\nvalue = [72
, 319, 5]\nclass = b'),
 Text(0.03125, 0.25, 'O_3 <= 63.965\ngini = 0.089\nsamples = 193\nvalue = [15, 305, 0]\nc
lass = b'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.43\nsamples = 31\nvalue = [15, 33, 0]\ncla
ss = b'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.0\nsamples = 162\nvalue = [0, 272, 0]\ncla
ss = b'),
 Text(0.09375, 0.25, 'MXY <= 1.035\ngini = 0.399\nsamples = 51\nvalue = [57, 14, 5]\nclas
s = a'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.503\nsamples = 23\nvalue = [22, 14, 1]\ncl
ass = a'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.184\nsamples = 28\nvalue = [35, 0, 4]\ncla
ss = a'),
 Text(0.1875, 0.4166666666666667, 'NOx <= 27.53\ngini = 0.063\nsamples = 827\nvalue = [34
, 1290, 9]\nclass = b'),
 Text(0.15625, 0.25, 'NOx <= 21.085\ngini = 0.021\nsamples = 721\nvalue = [8, 1149, 4]\nc
lass = b'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.008\nsamples = 636\nvalue = [4, 1025, 0]\n
class = b'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.116\nsamples = 85\nvalue = [4, 124, 4]\ncl
ass = b'),
 Text(0.21875, 0.25, 'TCH <= 1.265\ngini = 0.304\nsamples = 106\nvalue = [26, 141, 5]\ncl
ass = b'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.564\nsamples = 36\nvalue = [21, 30, 5]\ncl
ass = b'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.082\nsamples = 70\nvalue = [5, 111, 0]\ncl
ass = b'),
 Text(0.375, 0.5833333333333334, 'NOx <= 24.405\ngini = 0.339\nsamples = 1679\nvalue = [3
1, 2121, 540]\nclass = b'),
 Text(0.3125, 0.4166666666666667, 'MXY <= 1.065\ngini = 0.152\nsamples = 1084\nvalue = [8
, 1581, 134]\nclass = b'),
 Text(0.28125, 0.25, 'NOx <= 15.255\ngini = 0.09\nsamples = 960\nvalue = [6, 1450, 66]\nc
lass = b'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.002\nsamples = 633\nvalue = [0, 987, 1]\nc
lass = b'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.233\nsamples = 327\nvalue = [6, 463, 65]\n
class = b'),
 Text(0.34375, 0.25, 'OXY <= 0.78\ngini = 0.461\nsamples = 124\nvalue = [2, 131, 68]\ncla
ss = b'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.102\nsamples = 57\nvalue = [0, 88, 5]\ncla
ss = b'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.501\nsamples = 67\nvalue = [2, 43, 63]\ncl
ass = c'),
 Text(0.4375, 0.4166666666666667, 'TCH <= 1.355\ngini = 0.513\nsamples = 595\nvalue = [23
, 540, 406]\nclass = b'),
 Text(0.40625, 0.25, 'OXY <= 0.645\ngini = 0.451\nsamples = 312\nvalue = [20, 149, 369]\n
class = c'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.248\nsamples = 38\nvalue = [5, 56, 4]\ncla
ss = b'),
```

```
 Text(0.421875, 0.08333333333333333, 'gini = 0.365\nsamples = 274\nvalue = [15, 93, 365]\
nclass = c'),
 Text(0.46875, 0.25, 'NMHC <= 0.115\ngini = 0.17\nsamples = 283\nvalue = [3, 391, 37]\ncl
ass = b'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.393\nsamples = 66\nvalue = [3, 76, 23]\ncl
ass = b'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.081\nsamples = 217\nvalue = [0, 315, 14]\n
class = b'),
 Text(0.75, 0.75, 'MXY <= 5.885\ngini = 0.62\nsamples = 8149\nvalue = [5482, 2029, 5398]\
nclass = a'),
 Text(0.625, 0.5833333333333334, 'PXY <= 0.765\ngini = 0.63\nsamples = 5992\nvalue = [307
4, 1923, 4490]\nclass = c'),
 Text(0.5625, 0.4166666666666667, 'O_3 <= 9.1\ngini = 0.633\nsamples = 1337\nvalue = [593
, 1019, 507]\nclass = b'),
 Text(0.53125, 0.25, 'NOx <= 156.9\ngini = 0.177\nsamples = 275\nvalue = [19, 388, 22]\nc
lass = b'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.047\nsamples = 241\nvalue = [2, 365, 7]\nc
lass = b'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.655\nsamples = 34\nvalue = [17, 23, 15]\nc
lass = b'),
 Text(0.59375, 0.25, 'OXY <= 0.665\ngini = 0.663\nsamples = 1062\nvalue = [574, 631, 485]
\nclass = b'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.575\nsamples = 420\nvalue = [274, 333, 61]
\nclass = b'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.657\nsamples = 642\nvalue = [300, 298, 424
]\nclass = c'),
 Text(0.6875, 0.4166666666666667, 'EBE <= 1.735\ngini = 0.579\nsamples = 4655\nvalue = [2
481, 904, 3983]\nclass = c'),
 Text(0.65625, 0.25, 'NMHC <= 0.085\ngini = 0.481\nsamples = 3286\nvalue = [1054, 603, 35
49]\nclass = c'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.327\nsamples = 420\nvalue = [531, 67, 58]\
nclass = a'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.384\nsamples = 2866\nvalue = [523, 536, 34
91]\nclass = c'),
 Text(0.71875, 0.25, 'TOL <= 9.92\ngini = 0.505\nsamples = 1369\nvalue = [1427, 301, 434]
\nclass = a'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.455\nsamples = 1089\nvalue = [1222, 197, 3
09]\nclass = a'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.637\nsamples = 280\nvalue = [205, 104, 125
]\nclass = a'),
 Text(0.875, 0.5833333333333334, 'BEN <= 2.035\ngini = 0.433\nsamples = 2157\nvalue = [24
08, 106, 908]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'TCH <= 1.375\ngini = 0.357\nsamples = 548\nvalue = [15
5, 34, 672]\nclass = c'),
 Text(0.78125, 0.25, 'BEN <= 1.29\ngini = 0.508\nsamples = 88\nvalue = [90, 5, 57]\nclass
= a'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.192\nsamples = 27\nvalue = [3, 2, 43]\ncla
ss = c'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.281\nsamples = 61\nvalue = [87, 3, 14]\ncl
ass = a'),
 Text(0.84375, 0.25, 'NMHC <= 0.155\ngini = 0.238\nsamples = 460\nvalue = [65, 29, 615]\n
class = c'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.405\nsamples = 34\nvalue = [39, 5, 8]\ncla
ss = a'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.144\nsamples = 426\nvalue = [26, 24, 607]\
nclass = c'),
 Text(0.9375, 0.4166666666666667, 'TCH <= 1.635\ngini = 0.217\nsamples = 1609\nvalue = [2
253, 72, 236]\nclass = a'),
 Text(0.90625, 0.25, 'OXY <= 2.815\ngini = 0.069\nsamples = 1136\nvalue = [1757, 21, 44]\
nclass = a'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.368\nsamples = 20\nvalue = [28, 9, 0]\ncla
ss = a'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.061\nsamples = 1116\nvalue = [1729, 12, 44
]\nclass = a'),
 Text(0.96875, 0.25, 'EBE <= 4.45\ngini = 0.477\nsamples = 473\nvalue = [496, 51, 192]\nc
lass = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.607\nsamples = 166\nvalue = [92, 39, 128]\
nclass = c'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.273\nsamples = 307\nvalue = [404, 12, 64]\
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: 0.3826824977651868
Ridge Regression: 0.3815141911316644
Lasso Regression 0.06316435993388725
ElasticNet Regression: 0.24028188740354617
Logistic Regression: 0.8741416915744405
Random Forest: 0.8762261973456433
```

Random Forest is suitable for this dataset