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_2010.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	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	NaN	NaN	NaN	10.15
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	NaN	NaN	NaN	12.24
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	NaN	NaN	NaN	NaN
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.410000	7.870000	NaN	10.06
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.670000	22.030001	NaN	10.68
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
209443	2010-08-01 00:00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	NaN	NaN	NaN	NaN
209444	2010-08-01 00:00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.259998	NaN	NaN	7.26
209445	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	NaN	NaN	NaN	NaN
209446	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	NaN	NaN	NaN	NaN
209447	2010-08-01 00:00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.150002	26.860001	NaN	7.03

209448 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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     6666 non-null   object 
 1   BEN      6666 non-null   float64
 2   CO       6666 non-null   float64
 3   EBE      6666 non-null   float64
 4   MXY      6666 non-null   float64
 5   NMHC     6666 non-null   float64
 6   NO_2     6666 non-null   float64
 7   NOx      6666 non-null   float64
 8   OXY      6666 non-null   float64
 9   O_3      6666 non-null   float64
 10  PM10     6666 non-null   float64
 11  PM25     6666 non-null   float64
 12  PXY      6666 non-null   float64
 13  SO_2     6666 non-null   float64
 14  TCH      6666 non-null   float64
 15  TOL      6666 non-null   float64
 16  station  6666 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 937.4+ KB
```

In [6]:

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

Out[6]:

	CO	station
11	0.18	28079024
23	0.23	28079099
35	0.17	28079024
47	0.21	28079099
59	0.16	28079024
...	...	...
191879	0.26	28079099
191891	0.16	28079024
191903	0.28	28079099
191915	0.16	28079024
191927	0.25	28079099

6666 rows × 2 columns

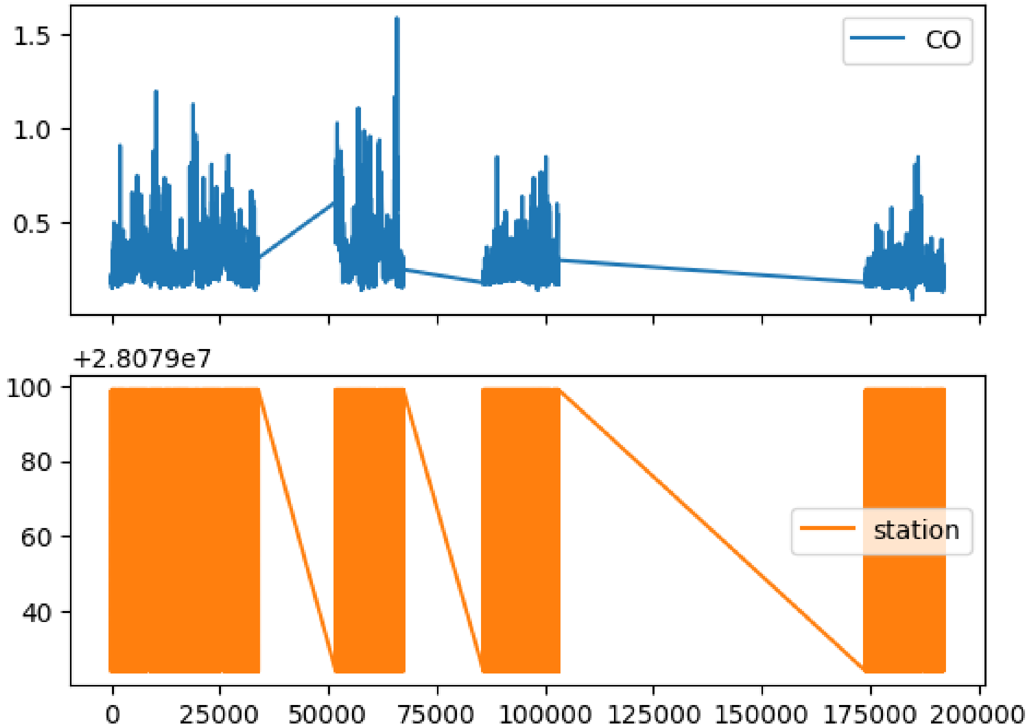
Line chart

In [7]:

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

Out[7]:

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



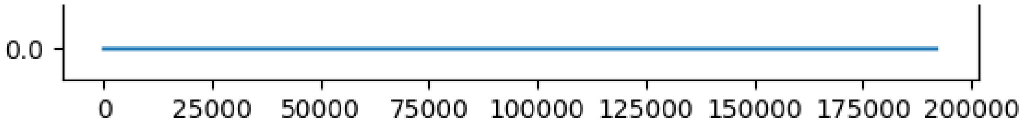
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

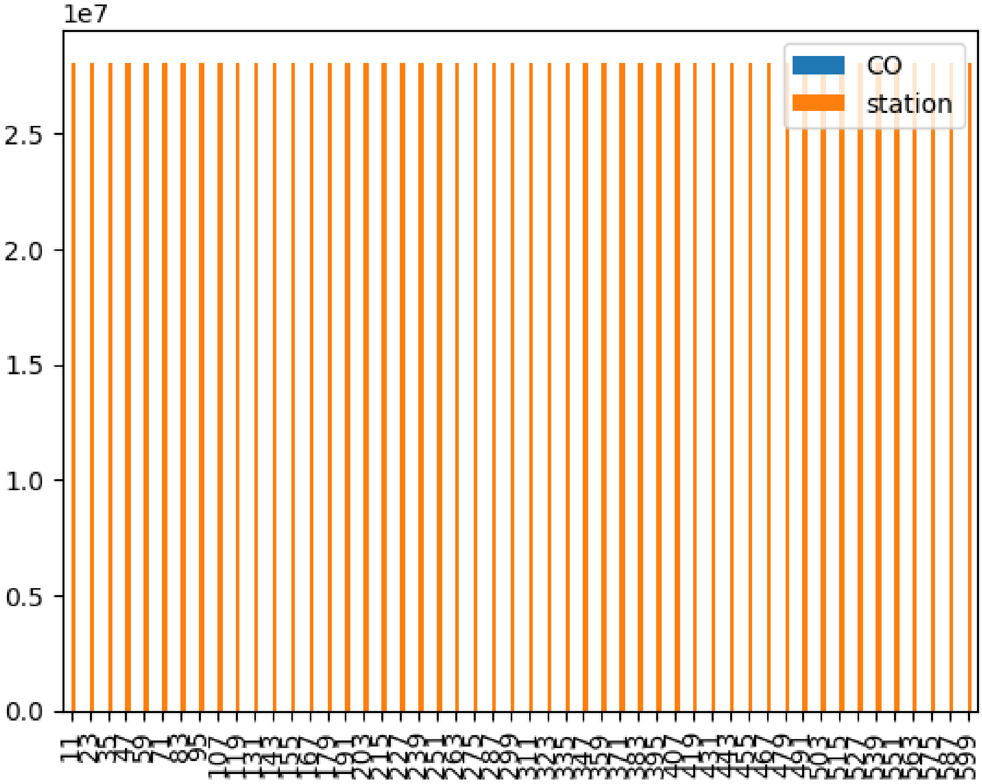
```
b=data[0:50]
```

In [10]:

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

Out[10]:

<Axes: >



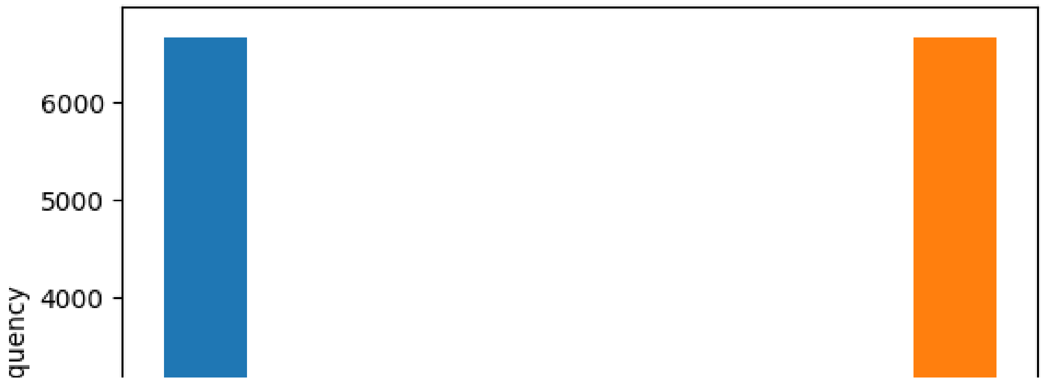
Histogram

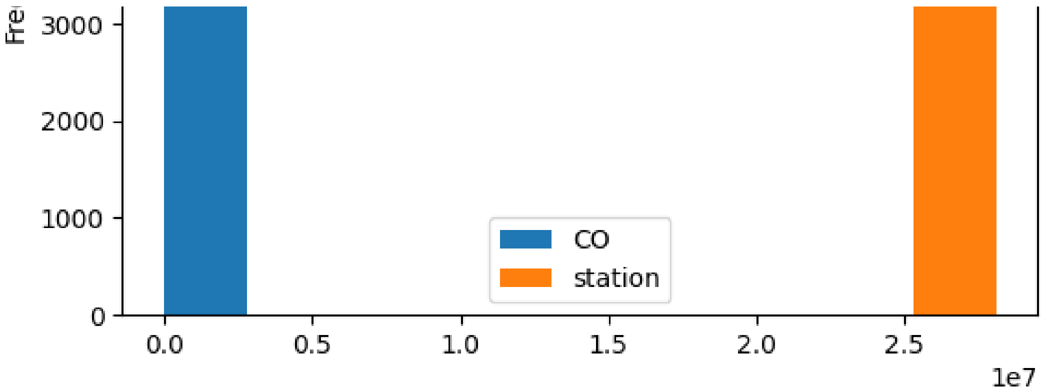
In [11]:

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

Out[11]:

<Axes: ylabel='Frequency'>





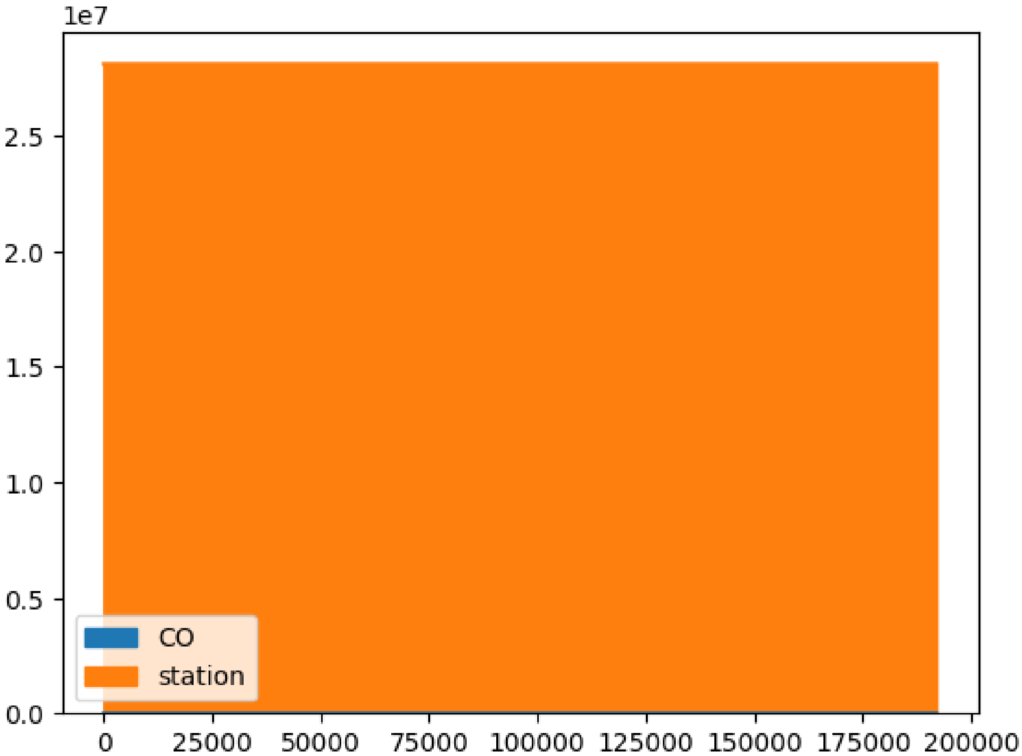
Area chart

In [12]:

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

Out[12]:

<Axes: >



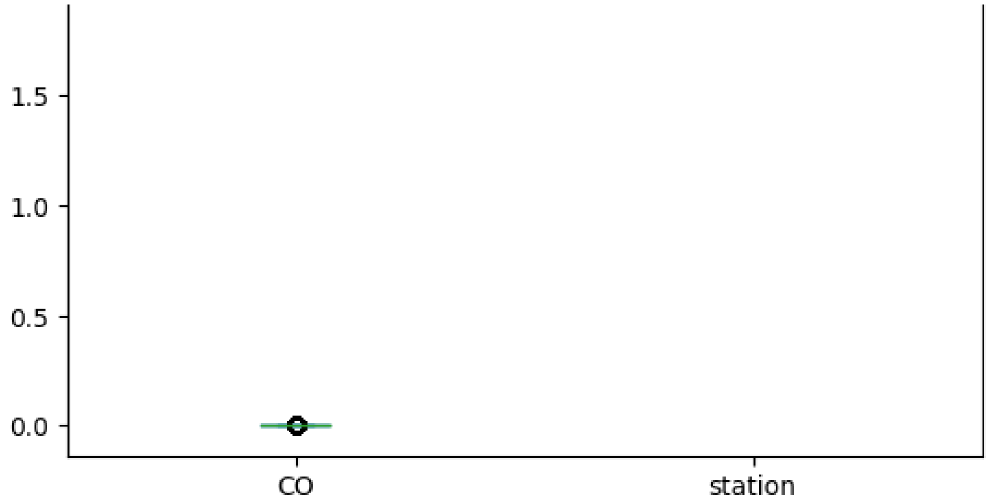
Box chart

In [13]:

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

Out[13]:

<Axes: >



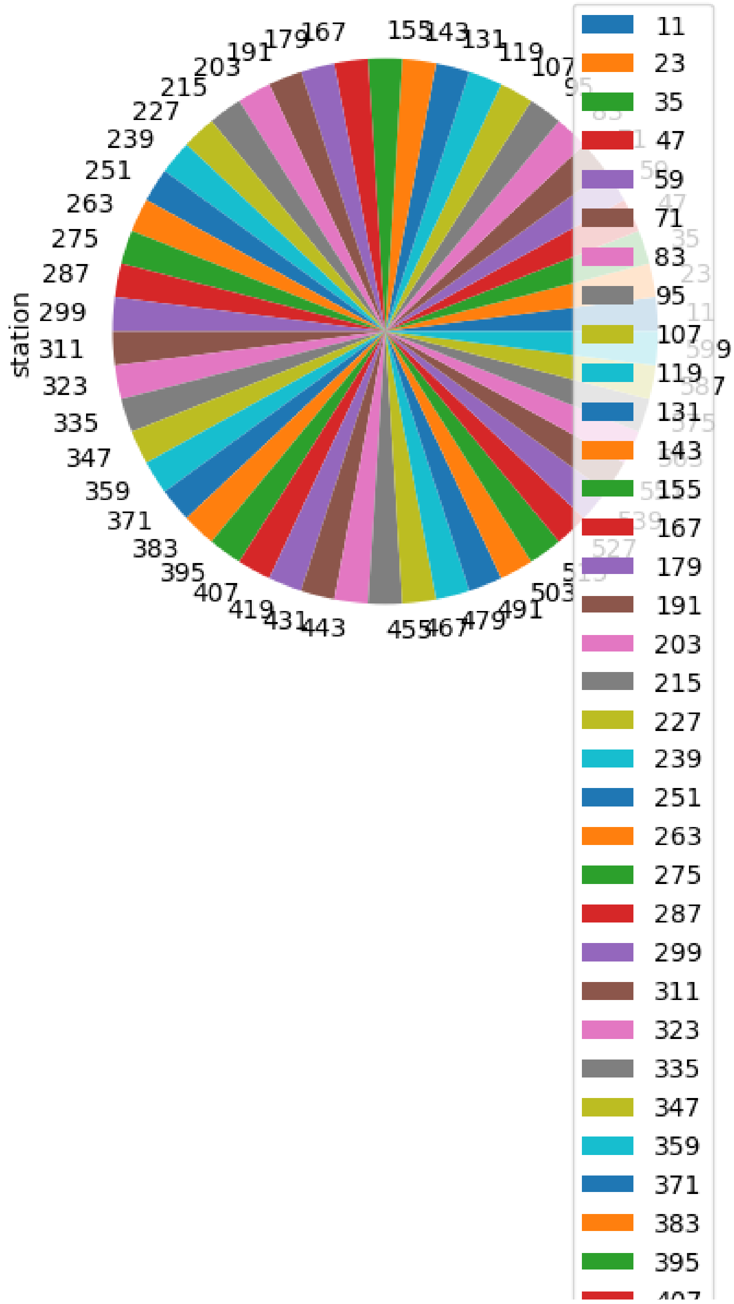
Pie chart

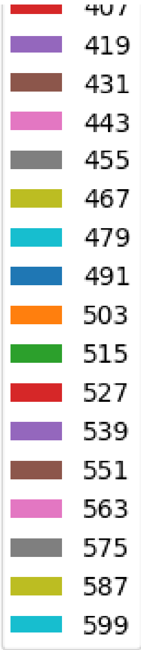
In [14]:

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

Out[14]:

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





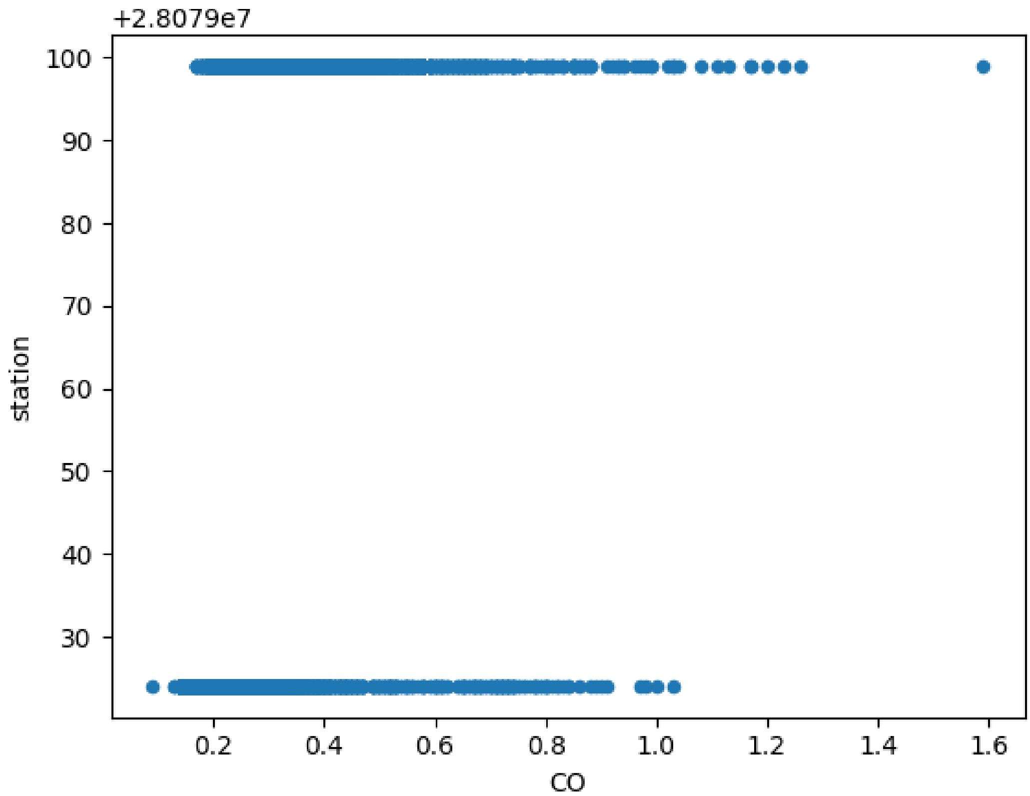
Scatter chart

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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     6666 non-null   object
 1   BEN      6666 non-null   float64
```

```
 1   BEN      6666 non-null   float64
 2   CO       6666 non-null   float64
 3   EBE      6666 non-null   float64
 4   MXY      6666 non-null   float64
 5   NMHC     6666 non-null   float64
 6   NO_2     6666 non-null   float64
 7   NOx      6666 non-null   float64
 8   OXY      6666 non-null   float64
 9   O_3      6666 non-null   float64
 10  PM10     6666 non-null   float64
 11  PM25     6666 non-null   float64
 12  PXY      6666 non-null   float64
 13  SO_2     6666 non-null   float64
 14  TCH      6666 non-null   float64
 15  TOL      6666 non-null   float64
 16  station  6666 non-null   int64
dtypes: float64(15), int64(1), object(1)
memory usage: 937.4+ KB
```

In [17]:

```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3
count	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000
mean	0.648425	0.296280	0.840585	0.839959	0.243378	33.888744	47.540617	0.916668	56.246101
std	0.395346	0.133296	0.508031	0.382263	0.115730	23.465169	41.230578	0.192521	30.380535
min	0.170000	0.090000	0.140000	0.110000	0.000000	1.290000	2.760000	0.200000	0.600000
25%	0.380000	0.200000	0.470000	0.590000	0.180000	15.752500	19.442501	1.000000	31.740000
50%	0.540000	0.260000	0.755000	1.000000	0.220000	29.320000	36.770000	1.000000	57.750000
75%	0.810000	0.340000	1.000000	1.000000	0.280000	47.657500	62.102501	1.000000	79.489998
max	5.110000	1.590000	5.190000	6.810000	0.930000	133.399994	409.299988	2.760000	161.100006

In [18]:

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

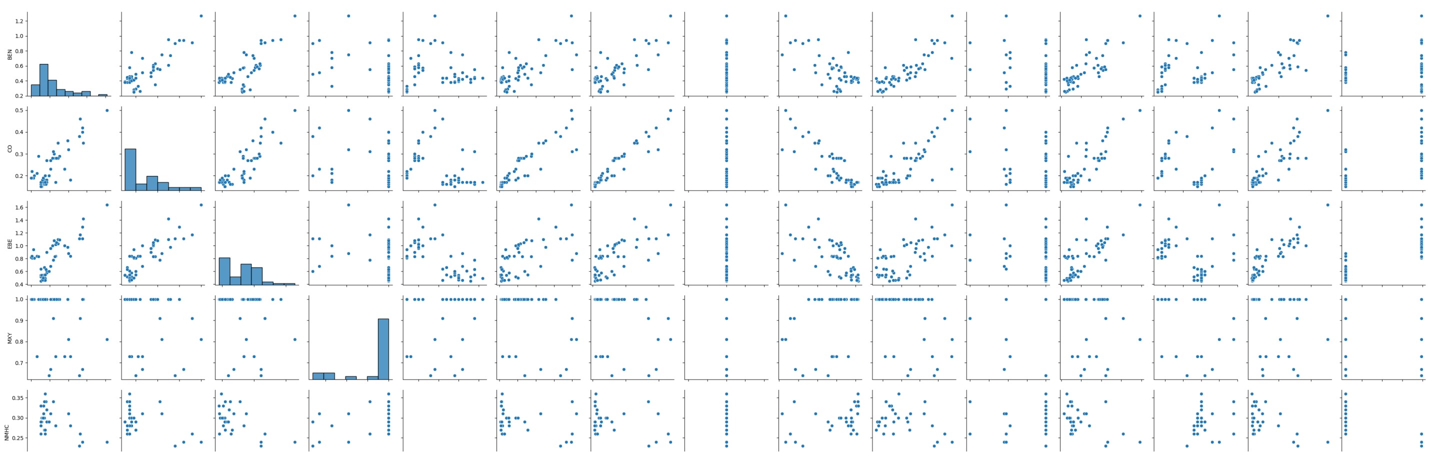
EDA AND VISUALIZATION

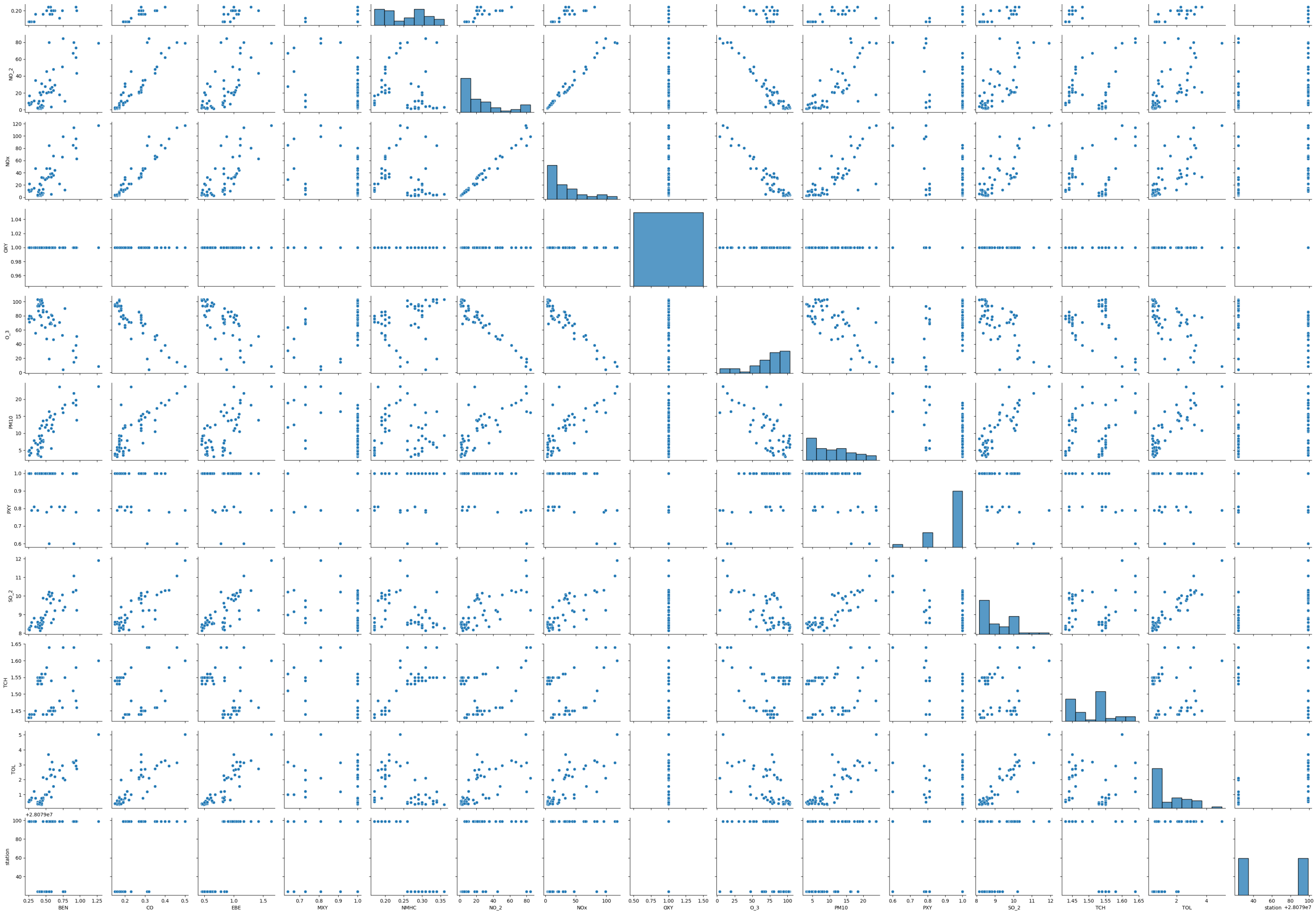
In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x7f1221c9a0e0>
```





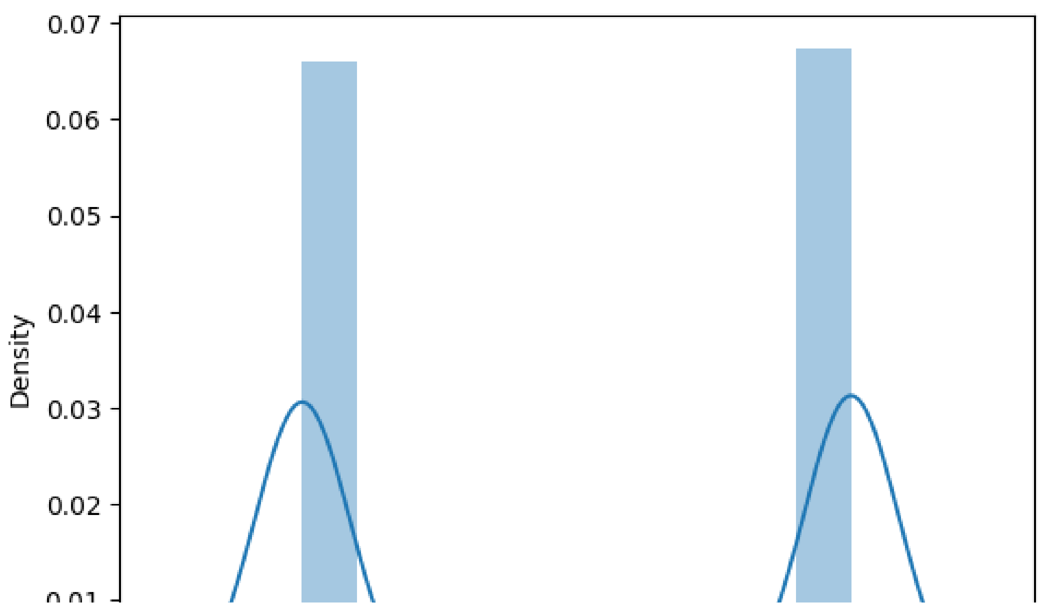
In [20]:

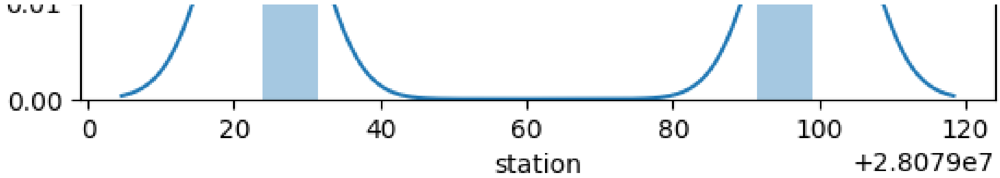
```
sns.distplot(df1['station'])
```

```
<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'>
```



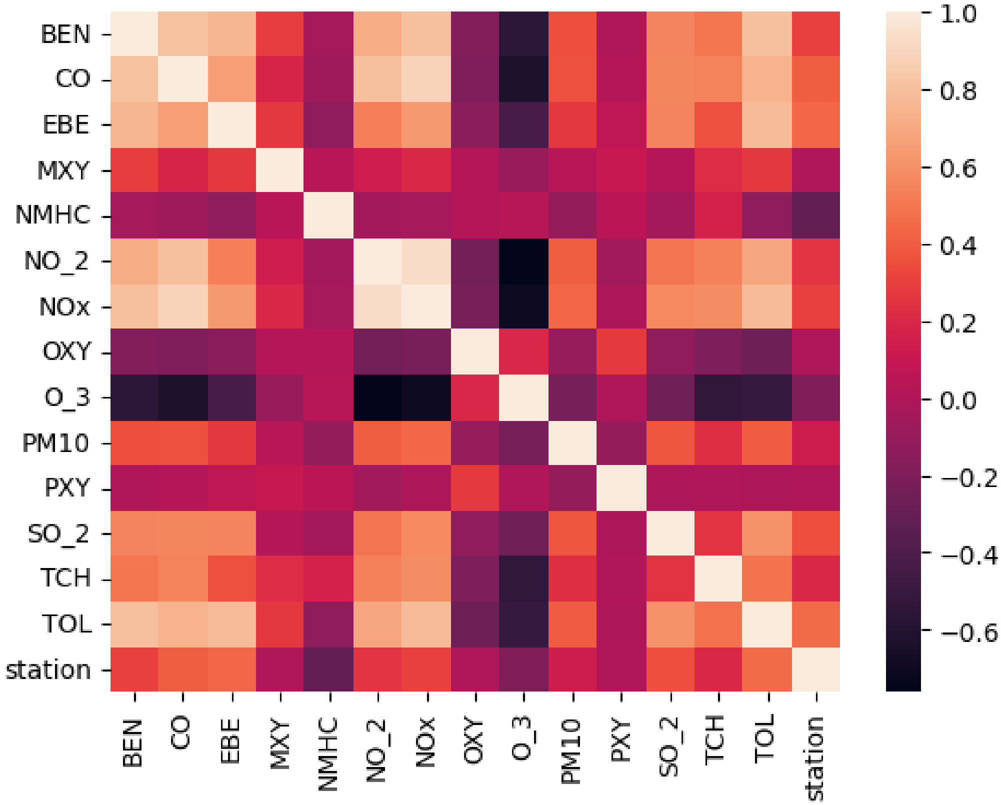


In [21]:

```
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]:

28078937.985878214

In [26]:

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

Out[26]:

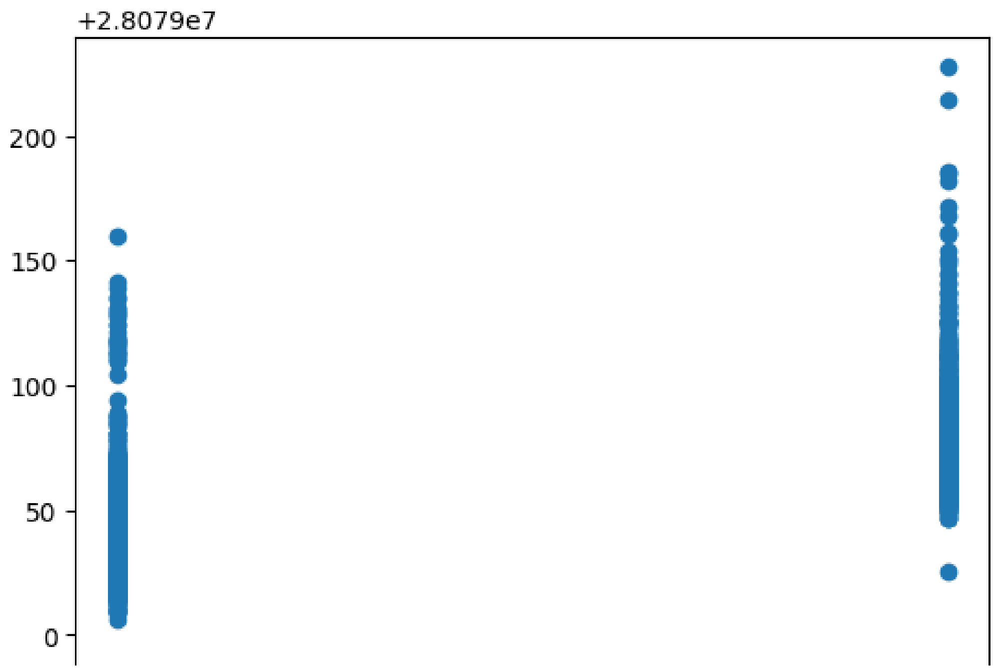
	Co-efficient
BEN	-36.709145
CO	187.134078
EBE	16.226167
MXY	-10.108055
NMHC	-71.228235
NO_2	0.279667
NOx	-0.647778
OXY	31.635234
O_3	0.067167
PM10	-0.171634
PXY	-5.699804
SO_2	1.731514
TCH	42.565878
TOL	10.181513

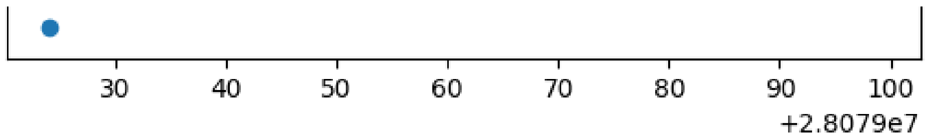
In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x7f1209b6c6a0>





ACCURACY

In [28]:

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

Out[28]:

0.3973996559206925

In [29]:

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

Out[29]:

0.439048183216168

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]:

0.4000976864620943

In [33]:

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

Out[33]:

0.4243968983294256

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]:

0.18330001884692382

Accuracy(Lasso)

In [36]:

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

Out[36]:

0.1808772022742714

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.        ,  0.22380028,  2.74088109, -1.33795864, -1.2035281 ,
        0.0106859 , -0.09520738,  0.34663853, -0.02829414, -0.11197929,
       -0.        ,  2.52733251,  0.        ,  7.14688317])
```

In [39]:

```
en.intercept_
```

Out[39]:

28079026.65612885

In [40]:

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

In [41]:

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

Out[41]:

0.23395551898080547

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)))
```

```
30.841453990375623
1076.214814133815
32.80571313252945
```

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]:

```
(6666, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(6666,)
```

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([28079024, 28079099])
```

array([28079024, 28079099])

In [53]:

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

Out[53]:

0.8660366036603661

In [54]:

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

Out[54]:

0.0

In [55]:

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

Out[55]:

array([[0., 1.]])

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]:

0.9305615087869696

In [61]:

```
rfc_best=grid_search.best_estimator_
```

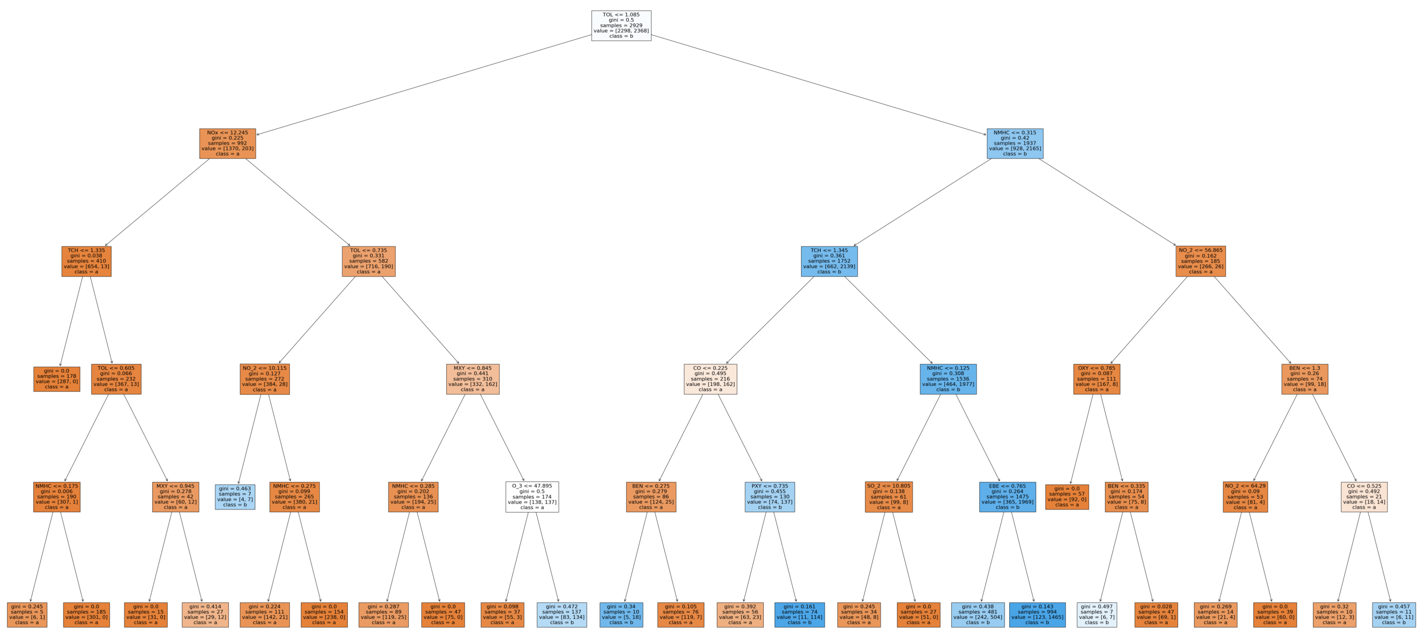
In [62]:

```
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(0.4375, 0.9166666666666666, 'TOL <= 1.085\ngini = 0.5\nsamples = 2929\nvalue = [229
8, 2368]\nclass = b'),
 Text(0.16145833333333334, 0.75, 'NOx <= 12.245\ngini = 0.225\nsamples = 992\nvalue = [13
70, 203]\nclass = a'),
 Text(0.0625, 0.5833333333333334, 'TCH <= 1.335\ngini = 0.038\nsamples = 410\nvalue = [65
4, 13]\nclass = a'),
 Text(0.041666666666666664, 0.4166666666666667, 'gini = 0.0\nsamples = 178\nvalue = [287,
0]\nclass = a'),
 Text(0.08333333333333333, 0.4166666666666667, 'TOL <= 0.605\ngini = 0.066\nsamples = 232
\nvalue = [367, 13]\nclass = a'),
 Text(0.041666666666666664, 0.25, 'NMHC <= 0.175\ngini = 0.006\nsamples = 190\nvalue = [3
07, 1]\nclass = a'),
 Text(0.020833333333333332, 0.08333333333333333, 'gini = 0.245\nsamples = 5\nvalue = [6,
1]\nclass = a'),
 Text(0.0625, 0.08333333333333333, 'gini = 0.0\nsamples = 185\nvalue = [301, 0]\nclass =
a'),
 Text(0.125, 0.25, 'MXY <= 0.945\ngini = 0.278\nsamples = 42\nvalue = [60, 12]\nclass = a
'),
 Text(0.10416666666666667, 0.08333333333333333, 'gini = 0.0\nsamples = 15\nvalue = [31, 0
]\nclass = a'),
 Text(0.14583333333333334, 0.08333333333333333, 'gini = 0.414\nsamples = 27\nvalue = [29,
12]\nclass = a'),
 Text(0.2604166666666667, 0.5833333333333334, 'TOL <= 0.735\ngini = 0.331\nsamples = 582\
nvalue = [716, 190]\nclass = a'),
 Text(0.1875, 0.4166666666666667, 'NO_2 <= 10.115\ngini = 0.127\nsamples = 272\nvalue = [
384, 28]\nclass = a'),
 Text(0.16666666666666666, 0.25, 'gini = 0.463\nsamples = 7\nvalue = [4, 7]\nclass = b'),
 Text(0.20833333333333334, 0.25, 'NMHC <= 0.275\ngini = 0.099\nsamples = 265\nvalue = [38
0, 21]\nclass = a'),
 Text(0.1875, 0.08333333333333333, 'gini = 0.224\nsamples = 111\nvalue = [142, 21]\nclass
= a'),
 Text(0.22916666666666666, 0.08333333333333333, 'gini = 0.0\nsamples = 154\nvalue = [238,
0]\nclass = a'),
 Text(0.3333333333333333, 0.4166666666666667, 'MXY <= 0.845\ngini = 0.441\nsamples = 310\
nvalue = [332, 162]\nclass = a'),
 Text(0.2916666666666667, 0.25, 'NMHC <= 0.285\ngini = 0.202\nsamples = 136\nvalue = [194
, 25]\nclass = a'),
 Text(0.2708333333333333, 0.08333333333333333, 'gini = 0.287\nsamples = 89\nvalue = [119,
25]\nclass = a'),
 Text(0.3125, 0.08333333333333333, 'gini = 0.0\nsamples = 47\nvalue = [75, 0]\nclass = a'
),
 Text(0.375, 0.25, 'O_3 <= 47.895\ngini = 0.5\nsamples = 174\nvalue = [138, 137]\nclass =
a'),
 Text(0.3541666666666667, 0.08333333333333333, 'gini = 0.098\nsamples = 37\nvalue = [55,
3]\nclass = a'),
 Text(0.3958333333333333, 0.08333333333333333, 'gini = 0.472\nsamples = 137\nvalue = [83,
134]\nclass = b'),
 Text(0.7135416666666666, 0.75, 'NMHC <= 0.315\ngini = 0.42\nsamples = 1937\nvalue = [928
, 2165]\nclass = b'),
 Text(0.5833333333333334, 0.5833333333333334, 'TCH <= 1.345\ngini = 0.361\nsamples = 1752
\nvalue = [662, 2139]\nclass = b'),
 Text(0.5, 0.4166666666666667, 'CO <= 0.225\ngini = 0.495\nsamples = 216\nvalue = [198, 1
62]\nclass = a'),
 Text(0.4583333333333333, 0.25, 'BEN <= 0.275\ngini = 0.279\nsamples = 86\nvalue = [124,
25]\nclass = a'),
 Text(0.4375, 0.08333333333333333, 'gini = 0.34\nsamples = 10\nvalue = [5, 18]\nclass = b
'),
```

```
 Text(0.4791666666666667, 0.08333333333333333, 'gini = 0.105\nsamples = 76\nvalue = [119,
7]\nclass = a'),
 Text(0.5416666666666666, 0.25, 'PXY <= 0.735\ngini = 0.455\nsamples = 130\nvalue = [74,
137]\nclass = b'),
 Text(0.5208333333333334, 0.08333333333333333, 'gini = 0.392\nsamples = 56\nvalue = [63,
23]\nclass = a'),
 Text(0.5625, 0.08333333333333333, 'gini = 0.161\nsamples = 74\nvalue = [11, 114]\nclass
= b'),
 Text(0.6666666666666666, 0.4166666666666667, 'NMHC <= 0.125\ngini = 0.308\nsamples = 153
6\nvalue = [464, 1977]\nclass = b'),
 Text(0.625, 0.25, 'SO_2 <= 10.805\ngini = 0.138\nsamples = 61\nvalue = [99, 8]\nclass =
a'),
 Text(0.6041666666666666, 0.08333333333333333, 'gini = 0.245\nsamples = 34\nvalue = [48,
8]\nclass = a'),
 Text(0.6458333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 27\nvalue = [51, 0]
\nclass = a'),
 Text(0.7083333333333334, 0.25, 'EBE <= 0.765\ngini = 0.264\nsamples = 1475\nvalue = [365
, 1969]\nclass = b'),
 Text(0.6875, 0.08333333333333333, 'gini = 0.438\nsamples = 481\nvalue = [242, 504]\nclas
s = b'),
 Text(0.7291666666666666, 0.08333333333333333, 'gini = 0.143\nsamples = 994\nvalue = [123
, 1465]\nclass = b'),
 Text(0.84375, 0.5833333333333334, 'NO_2 <= 56.865\ngini = 0.162\nsamples = 185\nvalue =
[266, 26]\nclass = a'),
 Text(0.7708333333333334, 0.4166666666666667, 'OXY <= 0.785\ngini = 0.087\nsamples = 111\
nvalue = [167, 8]\nclass = a'),
 Text(0.75, 0.25, 'gini = 0.0\nsamples = 57\nvalue = [92, 0]\nclass = a'),
 Text(0.7916666666666666, 0.25, 'BEN <= 0.335\ngini = 0.174\nsamples = 54\nvalue = [75, 8
]\nclass = a'),
 Text(0.7708333333333334, 0.08333333333333333, 'gini = 0.497\nsamples = 7\nvalue = [6, 7]
\nclass = b'),
 Text(0.8125, 0.08333333333333333, 'gini = 0.028\nsamples = 47\nvalue = [69, 1]\nclass =
a'),
 Text(0.9166666666666666, 0.4166666666666667, 'BEN <= 1.3\ngini = 0.26\nsamples = 74\nval
ue = [99, 18]\nclass = a'),
 Text(0.875, 0.25, 'NO_2 <= 64.29\ngini = 0.09\nsamples = 53\nvalue = [81, 4]\nclass = a'
),
 Text(0.8541666666666666, 0.08333333333333333, 'gini = 0.269\nsamples = 14\nvalue = [21,
4]\nclass = a'),
 Text(0.8958333333333334, 0.08333333333333333, 'gini = 0.0\nsamples = 39\nvalue = [60, 0]
\nclass = a'),
 Text(0.9583333333333334, 0.25, 'CO <= 0.525\ngini = 0.492\nsamples = 21\nvalue = [18, 14
]\nclass = a'),
 Text(0.9375, 0.08333333333333333, 'gini = 0.32\nsamples = 10\nvalue = [12, 3]\nclass = a
'),
 Text(0.9791666666666666, 0.08333333333333333, 'gini = 0.457\nsamples = 11\nvalue = [6, 1
1]\nclass = b')]
```



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: 0.3973996559206925
Ridge Regression: 0.4000976864620943
Lasso Regression 0.1808772022742714
ElasticNet Regression: 0.23395551898080547
Logistic Regression: 0.8660366036603661
Random Forest: 0.9305615087869696
```

Random Forest is suitable for this dataset