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_2009.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	TCH
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.260000	NaN	NaN	5.55	NaN
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.580000	NaN	NaN	8.84	NaN
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.190001	NaN	NaN	6.98	NaN
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.530001	6.82	1.30	8.88	1.38
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.760000	NaN	NaN	7.82	1.41
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.830000	7.15	0.74	6.25	1.25
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.920000	NaN	NaN	4.90	NaN
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.460000	6.77	NaN	8.40	1.34
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.030000	NaN	NaN	7.84	1.42
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.360000	11.61	0.83	6.93	1.34

215688 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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 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
3	0.33	28079006
20	0.32	28079024
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...	...	...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

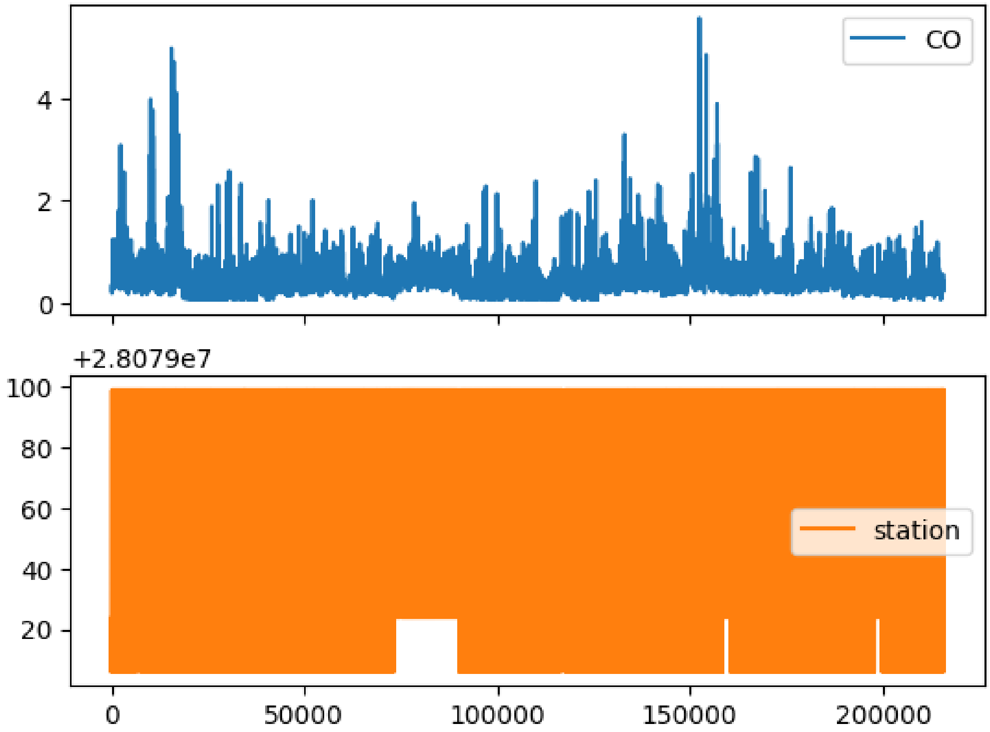
Line chart

In [7]:

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

Out[7]:

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



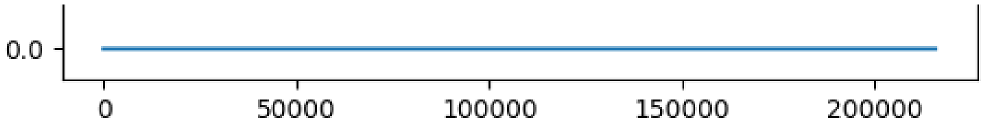
Line chart

In [8]:

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

Out[8]:

```
<Axes: >
```



Bar chart

In [9]:

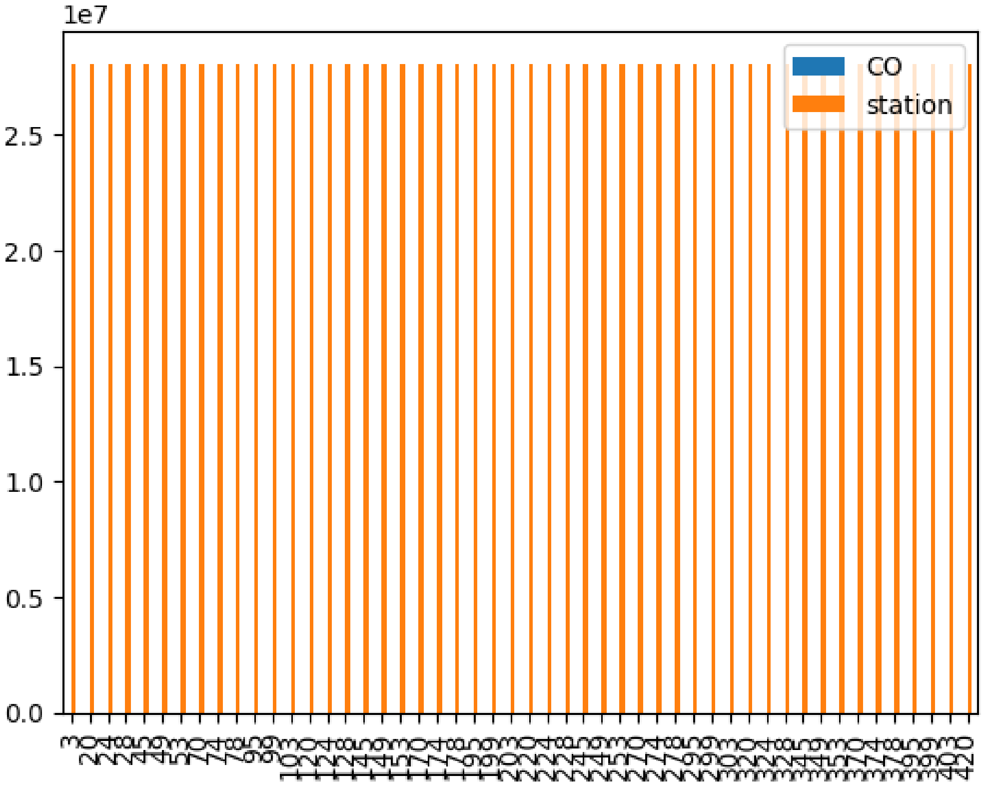
```
b=data[0:50]
```

In [10]:

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

Out[10]:

```
<Axes: >
```



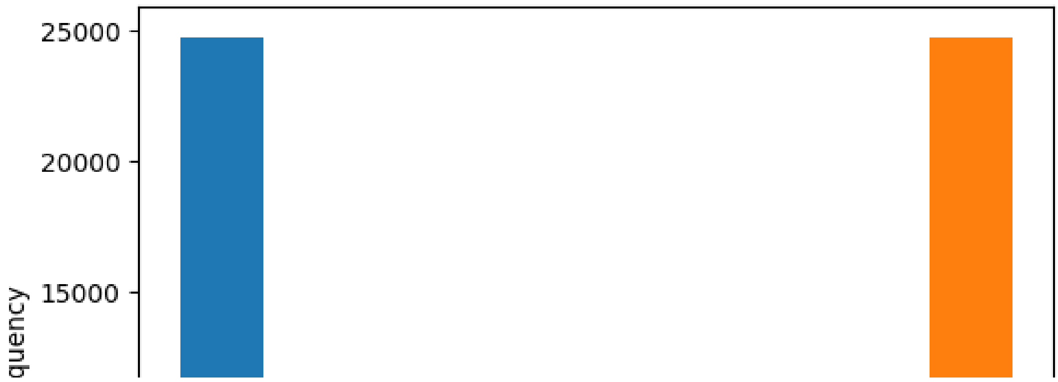
Histogram

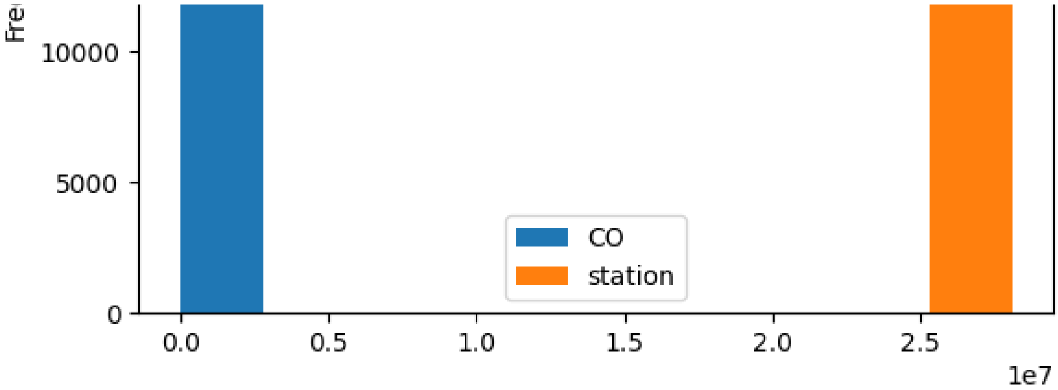
In [11]:

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

Out[11]:

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





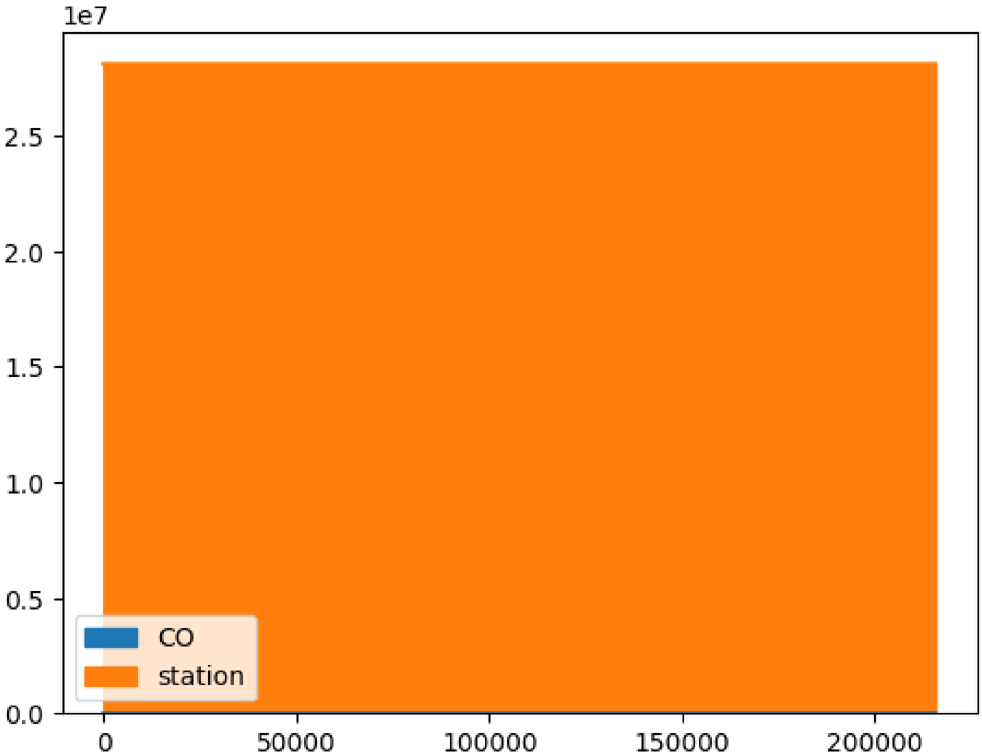
Area chart

In [12]:

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

Out[12]:

<Axes: >



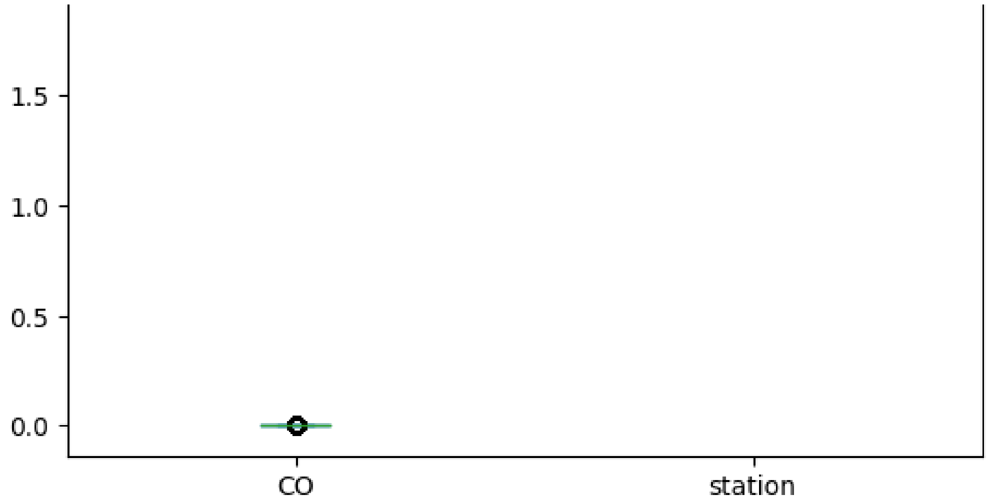
Box chart

In [13]:

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

Out[13]:

<Axes: >



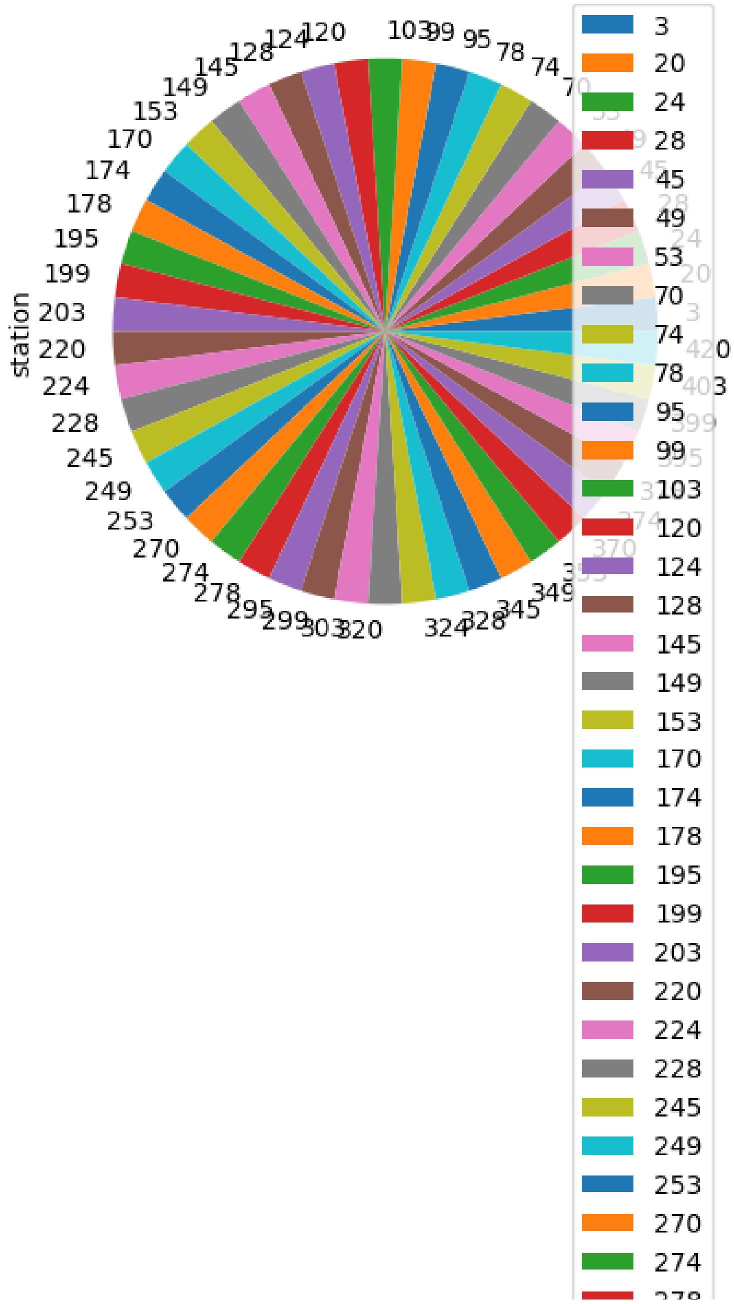
Pie chart

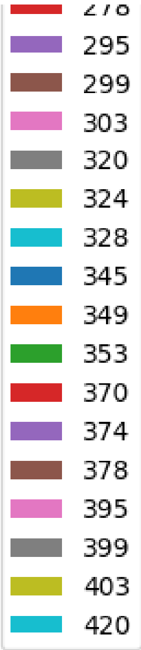
In [14]:

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

Out[14]:

<Axes: ylabel='station'>





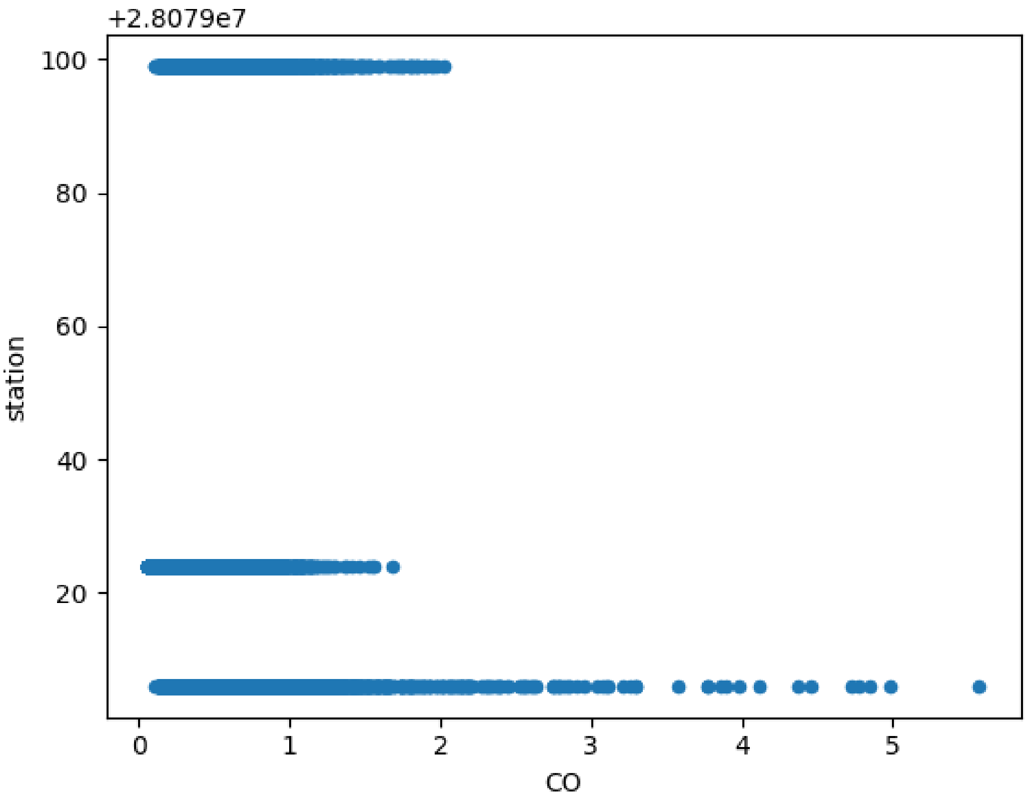
Scatter chart

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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   date     24717 non-null  object
 1   BEN      24717 non-null  float64
```

```
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 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	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	247
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929	92.907188	1.356536	
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677	91.985352	1.078515	
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000	2.250000	0.150000	
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000	33.009998	0.870000	
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000	67.010002	1.000000	
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997	124.699997	1.550000	
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994	1438.000000	45.349998	1

In [18]:

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

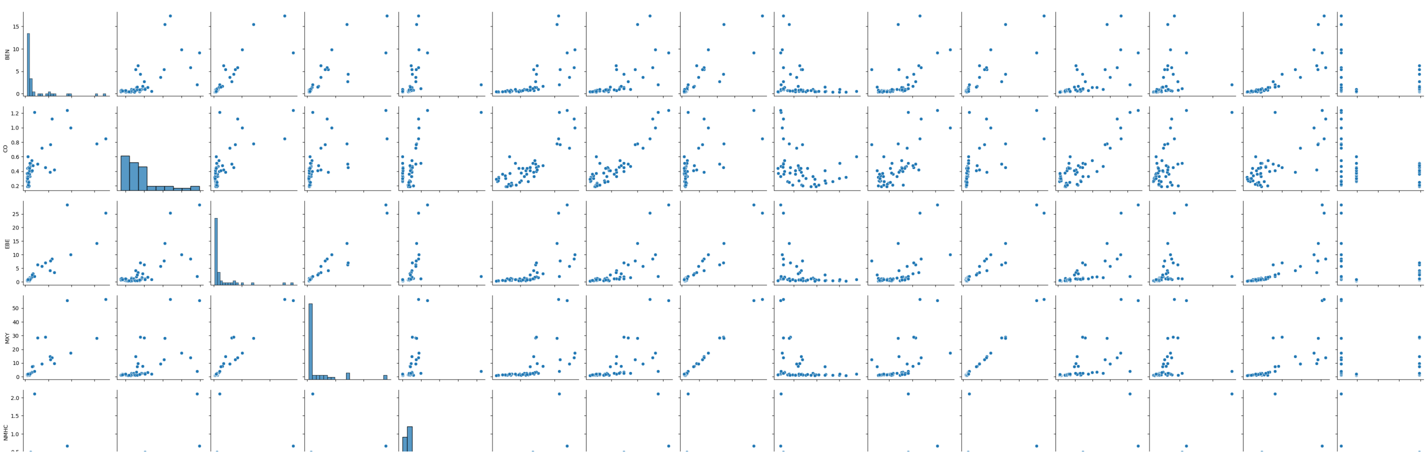
EDA AND VISUALIZATION

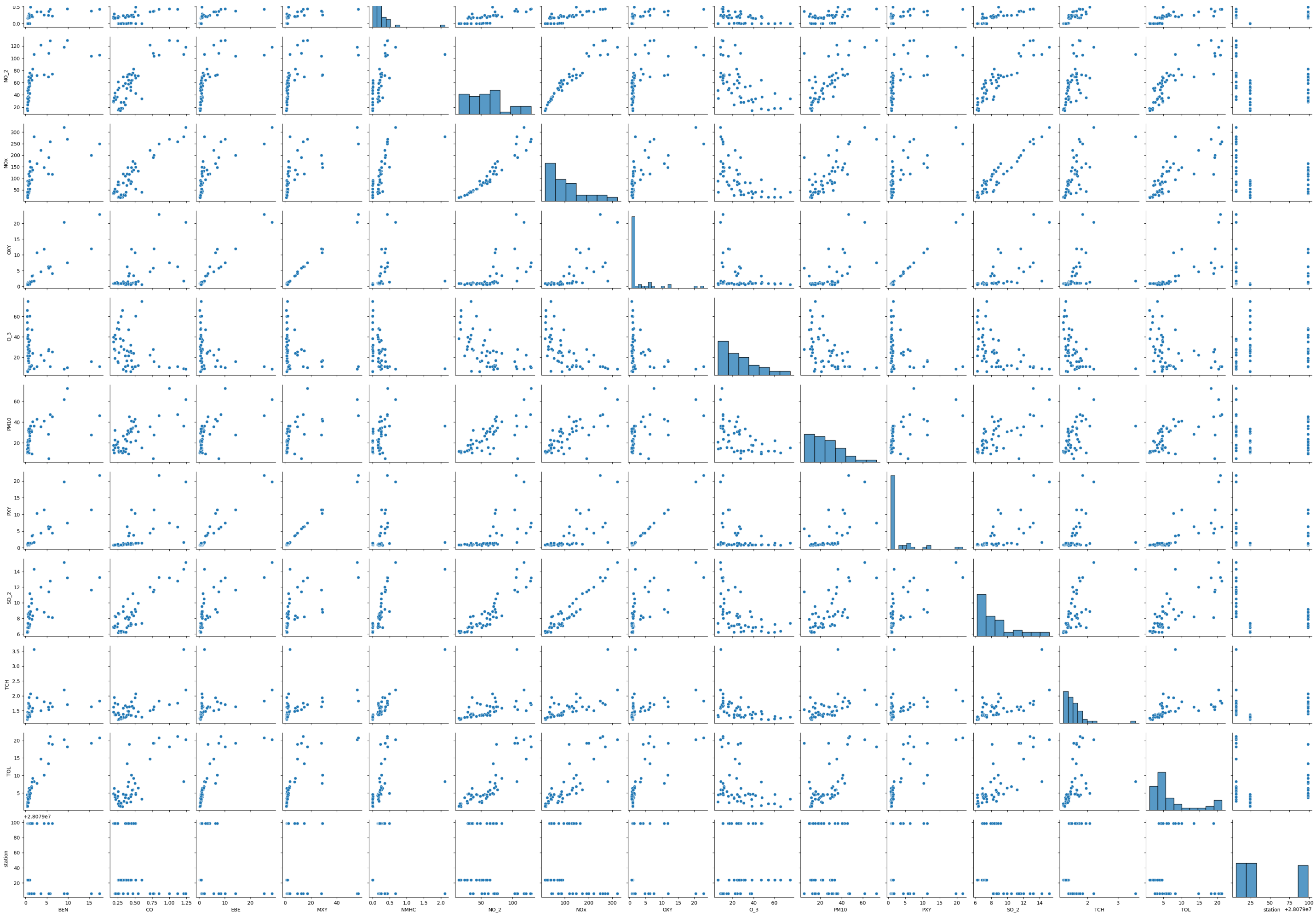
In [19]:

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

Out[19]:

<seaborn.axisgrid.PairGrid at 0x7ca9ff22b4c0>





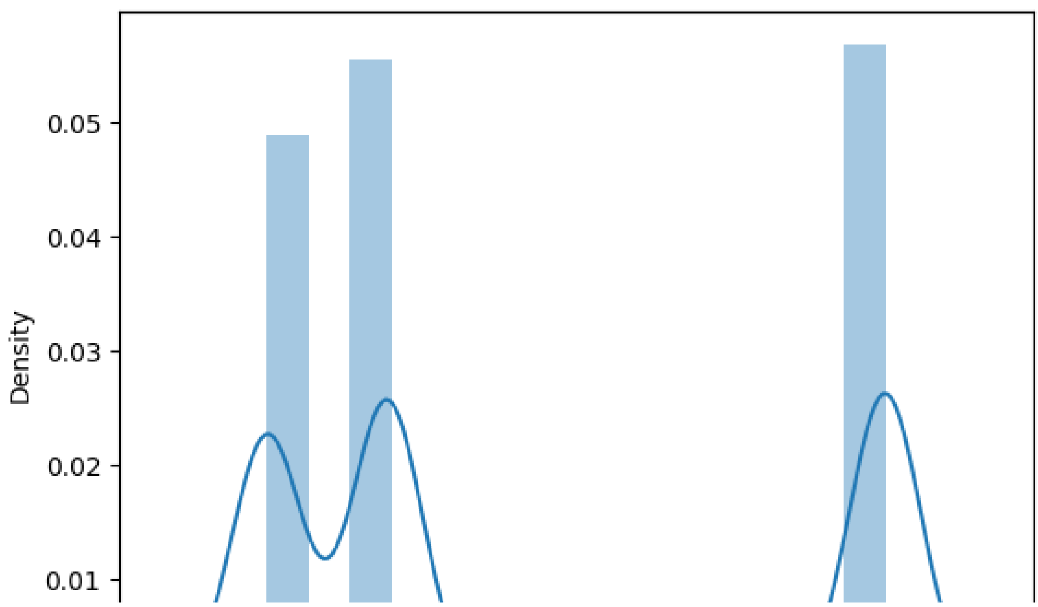
In [20]:

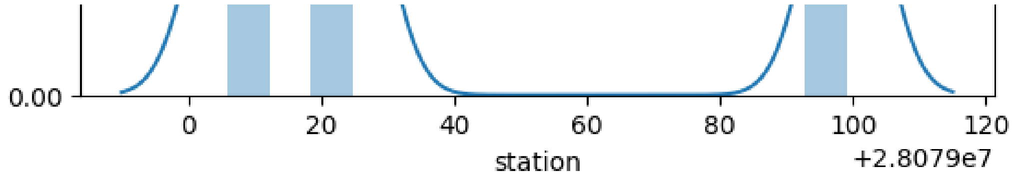
```
sns.distplot(df1['station'])
```

```
<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'>
```



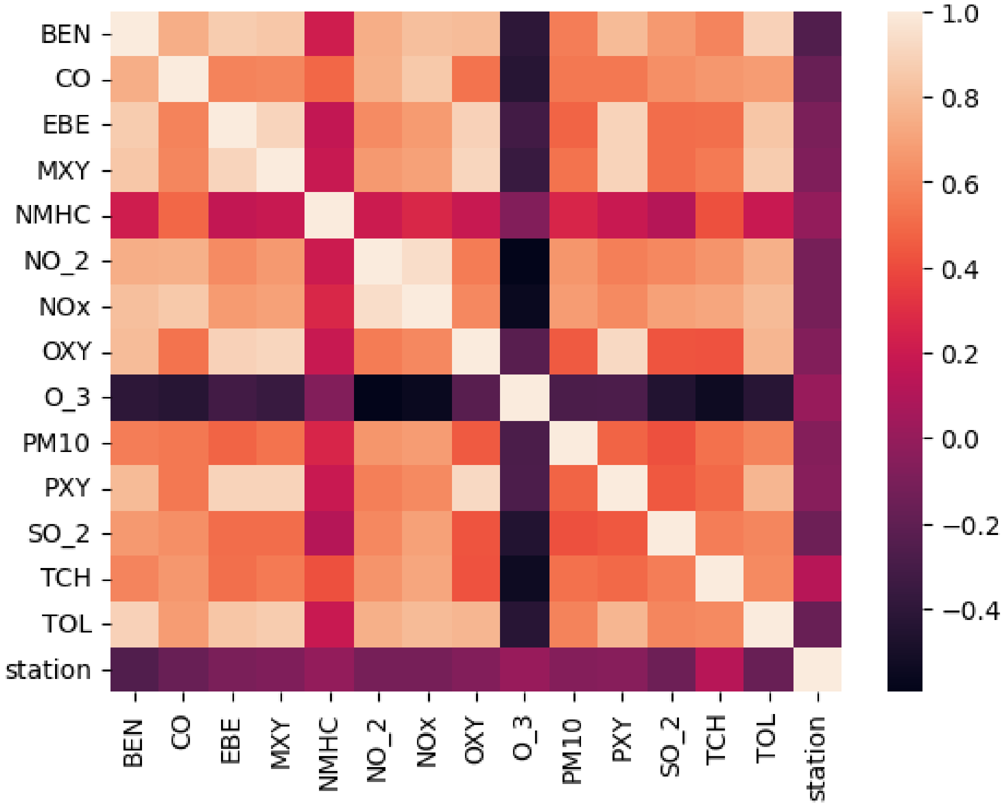


In [21]:

```
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]:

28078904.97239285

In [26]:

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

Out[26]:

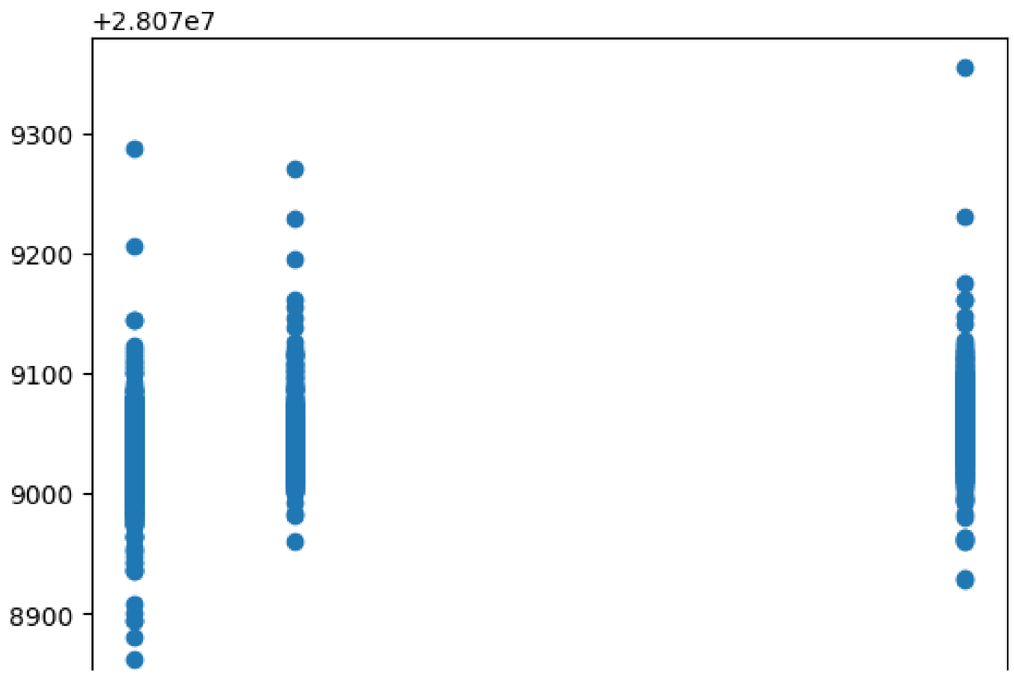
	Co-efficient
BEN	-35.248500
CO	-29.921359
EBE	6.257394
MXY	-1.547774
NMHC	-16.548767
NO_2	-0.181738
NOx	0.205752
OXY	14.358166
O_3	-0.004864
PM10	-0.046038
PXY	4.050908
SO_2	-0.319152
TCH	115.664965
TOL	-1.301777

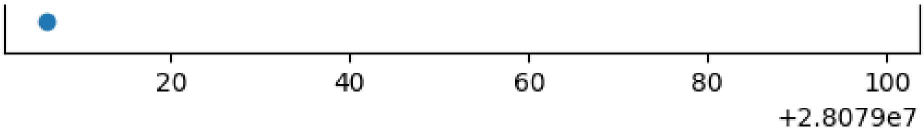
In [27]:

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

Out[27]:

<matplotlib.collections.PathCollection at 0x7ca9e66aa350>





ACCURACY

In [28]:

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

Out[28]:

```
0.2889123628906135
```

In [29]:

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

Out[29]:

```
0.28554354189040687
```

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]:

```
0.2879116805035996
```

In [33]:

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

Out[33]:

```
0.28525424301646574
```

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]:

```
0.03815013378203646б
```

Accuracy(Lasso)

In [36]:

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

Out[36]:

```
0.035039526198036186
```

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([-7.04254528, -0.66218786,  0.29331947,  2.0809855 ,  0.        ,
       -0.23706237,  0.13048947,  1.31448982, -0.16022722,  0.08861188,
        2.14157142, -0.80119548,  1.44566157, -2.00729612])
```

In [39]:

```
en.intercept_
```

Out[39]:

```
28079064.76802195
```

In [40]:

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

In [41]:

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

Out[41]:

```
0.10297648164787698
```

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)))
```

```
35.80465613731044
1464.057773404086
38.2630078980219
```

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]:

```
(24717, 14)
```

In [46]:

```
target_vector.shape
```

Out[46]:

```
(24717,)
```

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])
```

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

In [53]:

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

Out[53]:

```
0.8951733624630821
```

In [54]:

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

Out[54]:

```
5.44720687651328e-13
```

In [55]:

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

Out[55]:

```
array([[5.44720688e-13, 8.28693051e-44, 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]:

```
0.8943989768966534
```

In [61]:

```
rfc_best=grid_search.best_estimator_
```

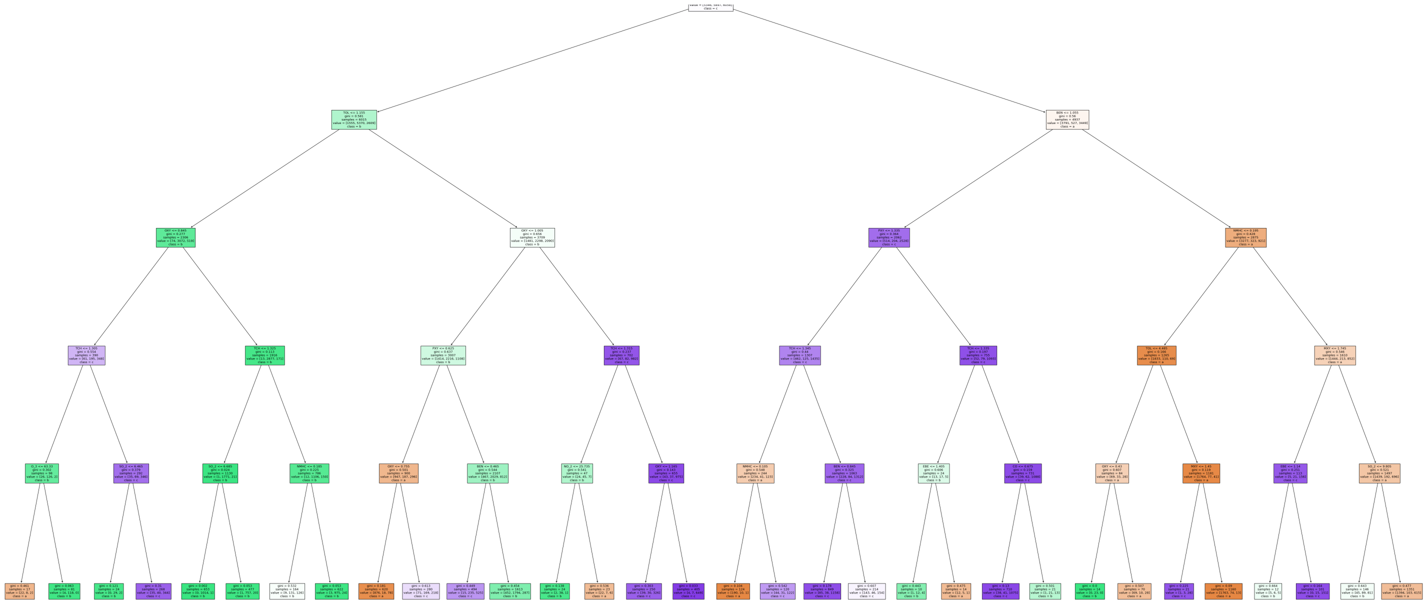
In [62]:

```
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, 'EBE <= 1.005\ngini = 0.666\nsamples = 10952\nvalue = [534
6, 5897, 6058]\nclass = c'),
 Text(0.25, 0.75, 'TOL <= 1.155\ngini = 0.581\nsamples = 6015\nvalue = [1555, 5370, 2609]
\nclass = b'),
 Text(0.125, 0.5833333333333334, 'OXY <= 0.845\ngini = 0.277\nsamples = 2306\nvalue = [74
, 3072, 519]\nclass = b'),
 Text(0.0625, 0.4166666666666667, 'TCH <= 1.305\ngini = 0.554\nsamples = 390\nvalue = [61
, 195, 348]\nclass = c'),
 Text(0.03125, 0.25, 'O_3 <= 63.33\ngini = 0.302\nsamples = 98\nvalue = [26, 126, 2]\ncla
ss = b'),
 Text(0.015625, 0.08333333333333333, 'gini = 0.461\nsamples = 17\nvalue = [22, 8, 2]\ncla
ss = a'),
 Text(0.046875, 0.08333333333333333, 'gini = 0.063\nsamples = 81\nvalue = [4, 118, 0]\ncl
ass = b'),
 Text(0.09375, 0.25, 'SO_2 <= 6.465\ngini = 0.379\nsamples = 292\nvalue = [35, 69, 346]\n
class = c'),
 Text(0.078125, 0.08333333333333333, 'gini = 0.121\nsamples = 24\nvalue = [0, 29, 2]\ncla
ss = b'),
 Text(0.109375, 0.08333333333333333, 'gini = 0.31\nsamples = 268\nvalue = [35, 40, 344]\n
class = c'),
 Text(0.1875, 0.4166666666666667, 'TCH <= 1.325\ngini = 0.113\nsamples = 1916\nvalue = [1
3, 2877, 171]\nclass = b'),
 Text(0.15625, 0.25, 'SO_2 <= 6.685\ngini = 0.024\nsamples = 1130\nvalue = [1, 1771, 21]\
nclass = b'),
 Text(0.140625, 0.08333333333333333, 'gini = 0.002\nsamples = 653\nvalue = [0, 1014, 1]\n
class = b'),
 Text(0.171875, 0.08333333333333333, 'gini = 0.053\nsamples = 477\nvalue = [1, 757, 20]\n
class = b'),
 Text(0.21875, 0.25, 'NMHC <= 0.185\ngini = 0.225\nsamples = 786\nvalue = [12, 1106, 150]
\nclass = b'),
 Text(0.203125, 0.08333333333333333, 'gini = 0.532\nsamples = 164\nvalue = [9, 131, 126]\
nclass = b'),
 Text(0.234375, 0.08333333333333333, 'gini = 0.053\nsamples = 622\nvalue = [3, 975, 24]\n
class = b'),
 Text(0.375, 0.5833333333333334, 'OXY <= 1.005\ngini = 0.656\nsamples = 3709\nvalue = [14
81, 2298, 2090]\nclass = b'),
 Text(0.3125, 0.4166666666666667, 'PXY <= 0.625\ngini = 0.637\nsamples = 3007\nvalue = [1
414, 2216, 1108]\nclass = b'),
 Text(0.28125, 0.25, 'OXY <= 0.755\ngini = 0.501\nsamples = 900\nvalue = [947, 187, 296]\
nclass = a'),
 Text(0.265625, 0.08333333333333333, 'gini = 0.181\nsamples = 620\nvalue = [876, 18, 78]\
nclass = a'),
 Text(0.296875, 0.08333333333333333, 'gini = 0.613\nsamples = 280\nvalue = [71, 169, 218]
\nclass = c'),
 Text(0.34375, 0.25, 'BEN <= 0.465\ngini = 0.544\nsamples = 2107\nvalue = [467, 2029, 812
]\nclass = b'),
 Text(0.328125, 0.08333333333333333, 'gini = 0.449\nsamples = 494\nvalue = [15, 235, 525]
\nclass = c'),
 Text(0.359375, 0.08333333333333333, 'gini = 0.454\nsamples = 1613\nvalue = [452, 1794, 2
87]\nclass = b'),
 Text(0.4375, 0.4166666666666667, 'TCH <= 1.315\ngini = 0.237\nsamples = 702\nvalue = [67
, 82, 982]\nclass = c'),
 Text(0.40625, 0.25, 'NO_2 <= 25.735\ngini = 0.541\nsamples = 47\nvalue = [24, 45, 7]\ncl
ass = b'),
 Text(0.390625, 0.08333333333333333, 'gini = 0.138\nsamples = 24\nvalue = [2, 38, 1]\ncla
ss = b'),
 Text(0.421875, 0.08333333333333333, 'gini = 0.536\nsamples = 23\nvalue = [22, 7, 6]\ncla
```

```
ss = a'),
 Text(0.46875, 0.25, 'OXY <= 1.165\ngini = 0.143\nsamples = 655\nvalue = [43, 37, 975]\nc
lass = c'),
 Text(0.453125, 0.08333333333333333, 'gini = 0.303\nsamples = 250\nvalue = [39, 30, 326]\
nclass = c'),
 Text(0.484375, 0.08333333333333333, 'gini = 0.033\nsamples = 405\nvalue = [4, 7, 649]\nc
lass = c'),
 Text(0.75, 0.75, 'BEN <= 1.055\ngini = 0.56\nsamples = 4937\nvalue = [3791, 527, 3449]\n
class = a'),
 Text(0.625, 0.5833333333333334, 'PXY <= 1.335\ngini = 0.364\nsamples = 2062\nvalue = [51
4, 204, 2528]\nclass = c'),
 Text(0.5625, 0.4166666666666667, 'TCH <= 1.345\ngini = 0.44\nsamples = 1307\nvalue = [46
2, 125, 1435]\nclass = c'),
 Text(0.53125, 0.25, 'NMHC <= 0.105\ngini = 0.548\nsamples = 244\nvalue = [234, 41, 123]\
nclass = a'),
 Text(0.515625, 0.08333333333333333, 'gini = 0.104\nsamples = 124\nvalue = [190, 10, 1]\n
class = a'),
 Text(0.546875, 0.08333333333333333, 'gini = 0.542\nsamples = 120\nvalue = [44, 31, 122]\
nclass = c'),
 Text(0.59375, 0.25, 'BEN <= 0.845\ngini = 0.325\nsamples = 1063\nvalue = [228, 84, 1312]
\nclass = c'),
 Text(0.578125, 0.08333333333333333, 'gini = 0.178\nsamples = 849\nvalue = [85, 38, 1158]
\nclass = c'),
 Text(0.609375, 0.08333333333333333, 'gini = 0.607\nsamples = 214\nvalue = [143, 46, 154]
\nclass = c'),
 Text(0.6875, 0.4166666666666667, 'TCH <= 1.335\ngini = 0.197\nsamples = 755\nvalue = [52
, 79, 1093]\nclass = c'),
 Text(0.65625, 0.25, 'EBE <= 1.405\ngini = 0.606\nsamples = 24\nvalue = [13, 17, 5]\nclas
s = b'),
 Text(0.640625, 0.08333333333333333, 'gini = 0.443\nsamples = 10\nvalue = [1, 12, 4]\ncla
ss = b'),
 Text(0.671875, 0.08333333333333333, 'gini = 0.475\nsamples = 14\nvalue = [12, 5, 1]\ncla
ss = a'),
 Text(0.71875, 0.25, 'CO <= 0.675\ngini = 0.159\nsamples = 731\nvalue = [39, 62, 1088]\nc
lass = c'),
 Text(0.703125, 0.08333333333333333, 'gini = 0.13\nsamples = 710\nvalue = [38, 41, 1075]\
nclass = c'),
 Text(0.734375, 0.08333333333333333, 'gini = 0.501\nsamples = 21\nvalue = [1, 21, 13]\ncl
ass = b'),
 Text(0.875, 0.5833333333333334, 'NMHC <= 0.195\ngini = 0.428\nsamples = 2875\nvalue = [3
277, 323, 921]\nclass = a'),
 Text(0.8125, 0.4166666666666667, 'TOL <= 4.485\ngini = 0.166\nsamples = 1265\nvalue = [1
833, 110, 69]\nclass = a'),
 Text(0.78125, 0.25, 'OXY <= 0.43\ngini = 0.607\nsamples = 84\nvalue = [69, 33, 28]\nclas
s = a'),
 Text(0.765625, 0.08333333333333333, 'gini = 0.0\nsamples = 14\nvalue = [0, 23, 0]\nclass
= b'),
 Text(0.796875, 0.08333333333333333, 'gini = 0.507\nsamples = 70\nvalue = [69, 10, 28]\nc
lass = a'),
 Text(0.84375, 0.25, 'MXY <= 1.45\ngini = 0.119\nsamples = 1181\nvalue = [1764, 77, 41]\n
class = a'),
 Text(0.828125, 0.08333333333333333, 'gini = 0.225\nsamples = 21\nvalue = [1, 3, 28]\ncla
ss = c'),
 Text(0.859375, 0.08333333333333333, 'gini = 0.09\nsamples = 1160\nvalue = [1763, 74, 13]
\nclass = a'),
 Text(0.9375, 0.4166666666666667, 'MXY <= 1.745\ngini = 0.546\nsamples = 1610\nvalue = [1
444, 213, 852]\nclass = a'),
 Text(0.90625, 0.25, 'EBE <= 1.14\ngini = 0.251\nsamples = 113\nvalue = [5, 21, 156]\ncla
ss = c'),
 Text(0.890625, 0.08333333333333333, 'gini = 0.664\nsamples = 12\nvalue = [5, 6, 5]\nclas
s = b'),
 Text(0.921875, 0.08333333333333333, 'gini = 0.164\nsamples = 101\nvalue = [0, 15, 151]\n
class = c'),
 Text(0.96875, 0.25, 'SO_2 <= 9.805\ngini = 0.521\nsamples = 1497\nvalue = [1439, 192, 69
6]\nclass = a'),
 Text(0.953125, 0.08333333333333333, 'gini = 0.643\nsamples = 146\nvalue = [45, 89, 81]\n
class = b'),
 Text(0.984375, 0.08333333333333333, 'gini = 0.477\nsamples = 1351\nvalue = [1394, 103, 6
15]\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: 0.2889123628906135
Ridge Regression: 0.2879116805035996
Lasso Regression 0.035039526198036186
ElasticNet Regression: 0.10297648164787698
Logistic Regression: 0.8951733624630821
Random Forest: 0.8943989768966534
```

Logistic Regression is suitable for this dataset