Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [2]:

```
df=pd.read_csv("2010.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM1(
0	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	Na
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	Na
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	Na
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.41000(
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.67000(
...	...	...	...	...	...	...	...	...	...	...	..
209443	2010-08-01 00:00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	Na
209444	2010-08-01 00:00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.25999ɛ
209445	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	Na
209446	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	Na
209447	2010-08-01 00:00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.15000ż

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

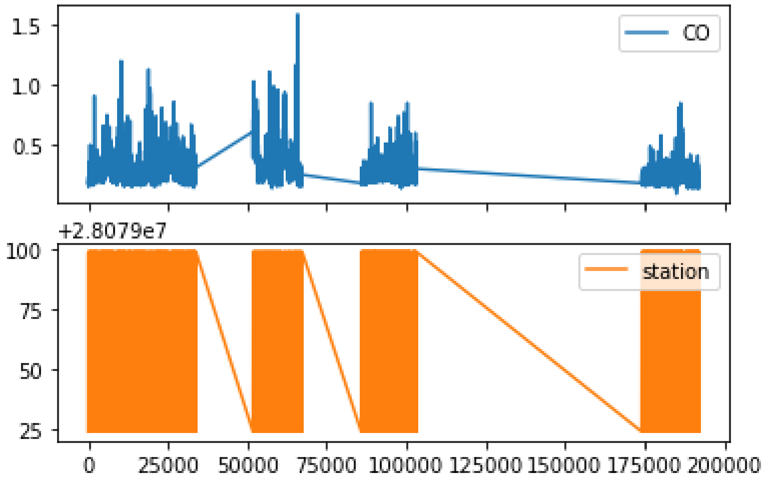
	CO	station
191927	0.25	28079099

6666 rows × 2 columns

Line chart

In [7]:
```
data.plot.line(subplots=True)
```

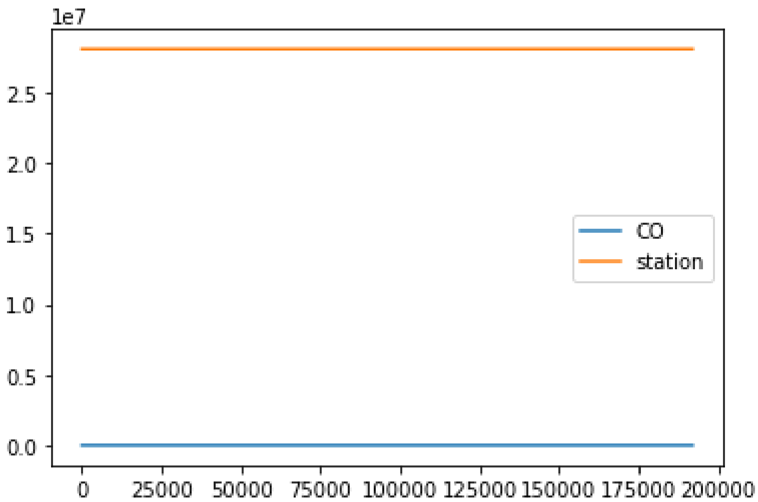
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [8]:
```
data.plot.line()
```

Out[8]:
```
<AxesSubplot:>
```

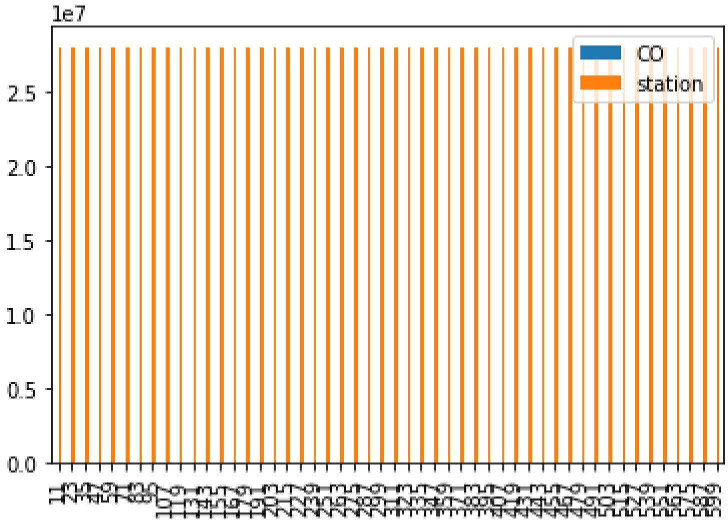


Bar chart

In [9]:
```
b=data[0:50]
```

In [10]:
```
b.plot.bar()
```

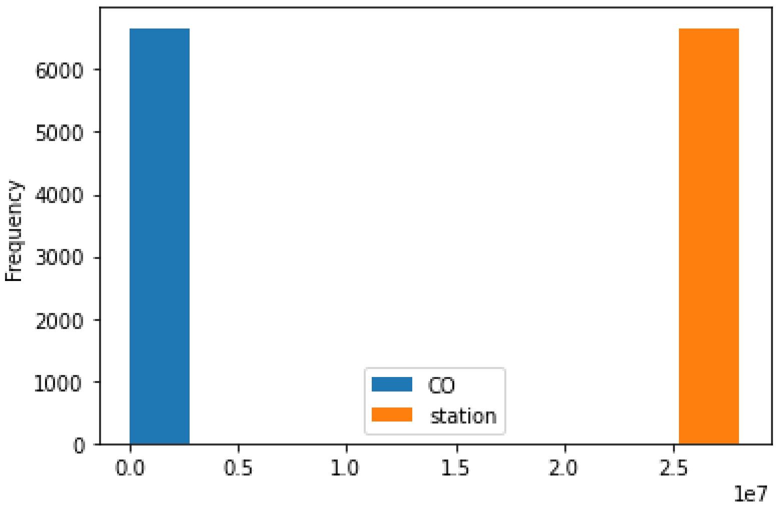
Out[10]:
```
<AxesSubplot:>
```



Histogram

In [11]:
```
data.plot.hist()
```

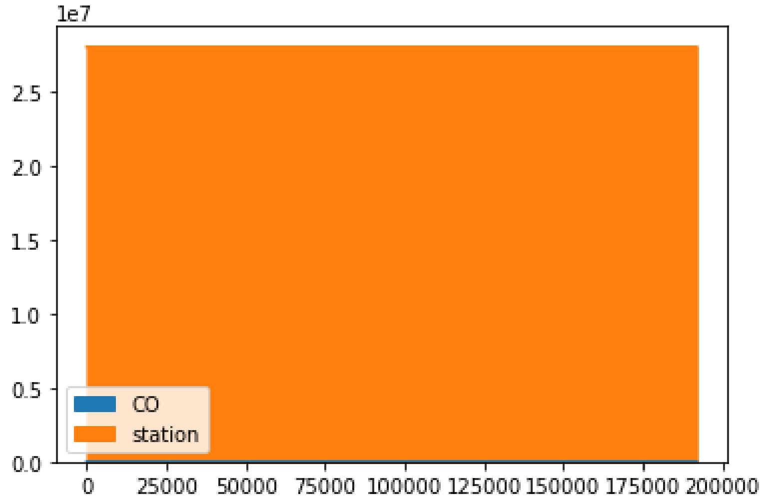
Out[11]:
```
<AxesSubplot:ylabel='Frequency'>
```



Area chart

In [12]:
```
data.plot.area()
```

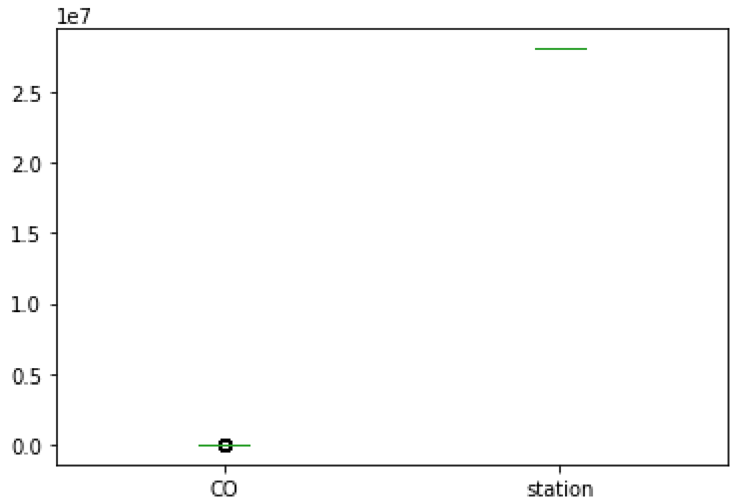
Out[12]:
```
<AxesSubplot:>
```



Box chart

In [13]:
```
data.plot.box()
```

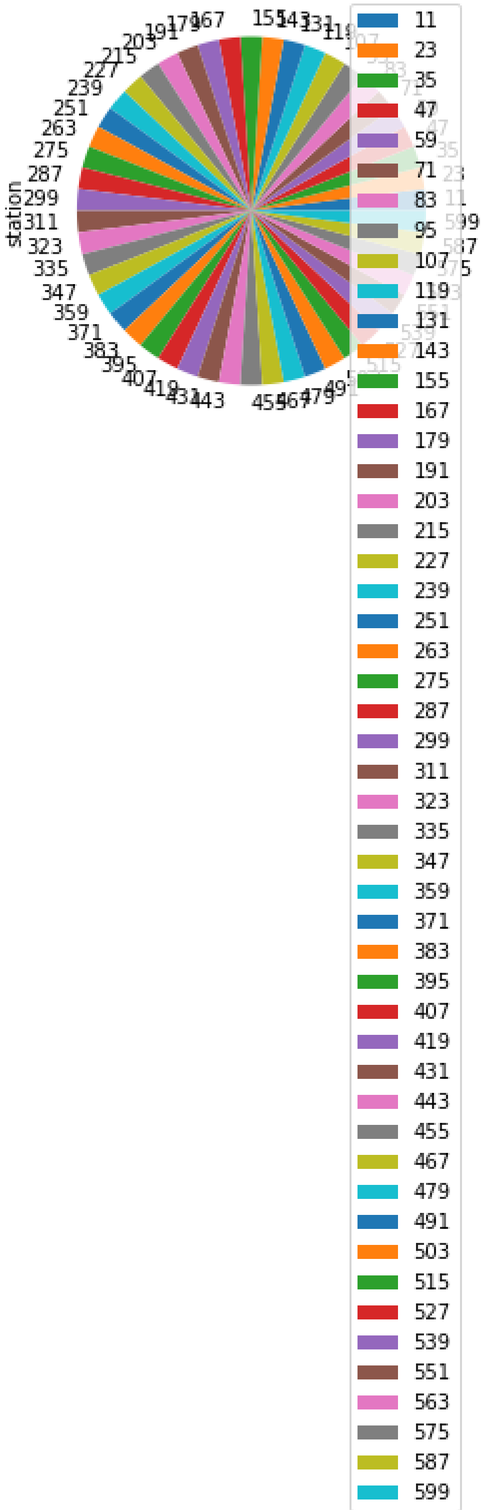
Out[13]: <AxesSubplot:>



Pie chart

In [14]:
```
b.plot.pie(y='station' )
```

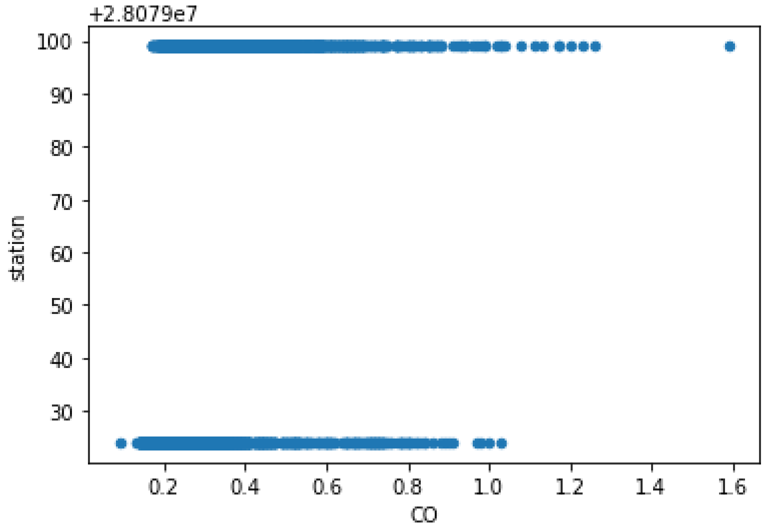
Out[14]: <AxesSubplot:ylabel='station'>



Scatter chart

In [15]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[15]: <AxesSubplot: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
 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	
count	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6
mean	0.648425	0.296280	0.840585	0.839959	0.243378	33.888744	47.540617	
std	0.395346	0.133296	0.508031	0.382263	0.115730	23.465169	41.230578	
min	0.170000	0.090000	0.140000	0.110000	0.000000	1.290000	2.760000	
25%	0.380000	0.200000	0.470000	0.590000	0.180000	15.752500	19.442501	
50%	0.540000	0.260000	0.755000	1.000000	0.220000	29.320000	36.770000	
75%	0.810000	0.340000	1.000000	1.000000	0.280000	47.657500	62.102501	
max	5.110000	1.590000	5.190000	6.810000	0.930000	133.399994	409.299988	

In [18]:

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

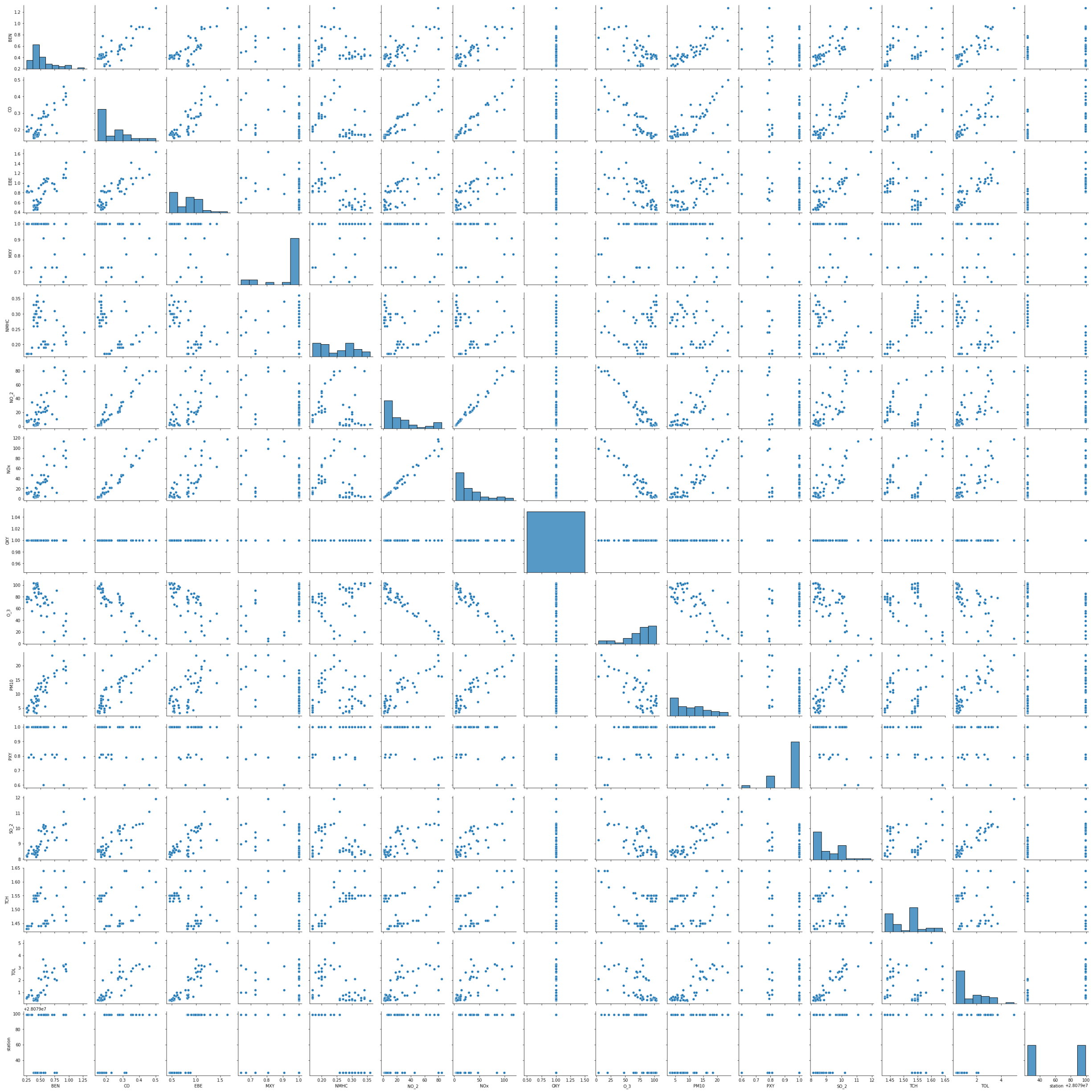
EDA AND VISUALIZATION

In [19]:

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

Out[19]:

```
<seaborn.axisgrid.PairGrid at 0x164013a3940>
```



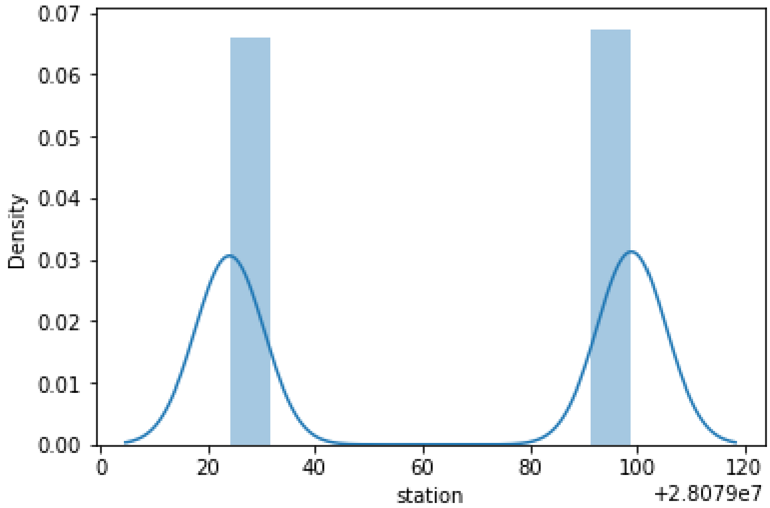
In [20]:

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

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

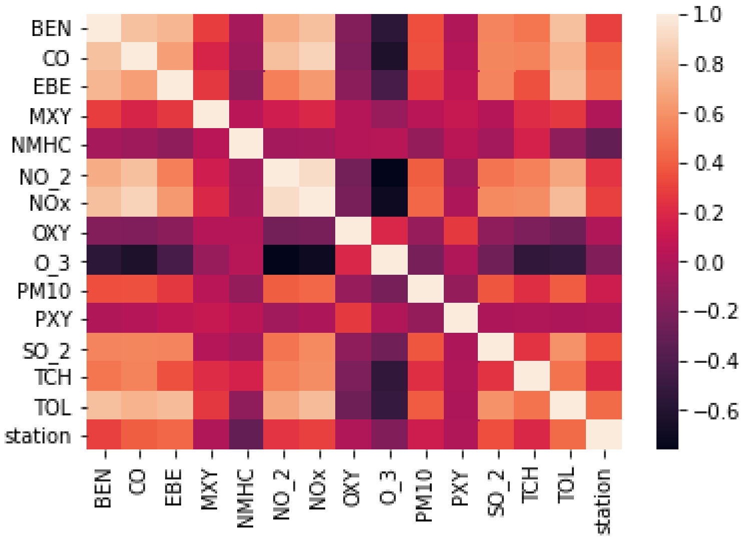
Out[20]:

```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



In [21]:
```
sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



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()

In [25]:
```
lr.intercept_
```

Out[25]: 28078935.5346144

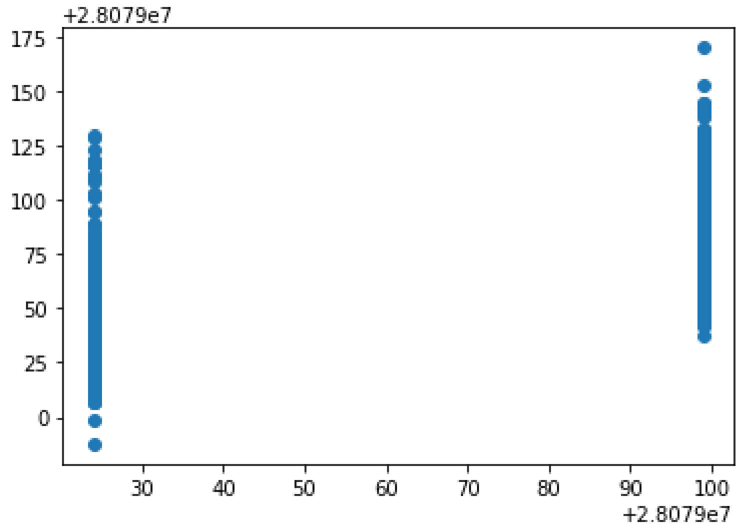
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-33.412832
CO	159.729915
EBE	17.318311
MXY	-9.867133
NMHC	-71.873011
NO_2	0.317269
NOx	-0.628945
OXY	28.536777
O_3	0.063989
PM10	-0.180838
PXY	-5.886308
SO_2	1.837711
TCH	48.674841
TOL	9.483652

In [27]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[27]: <matplotlib.collections.PathCollection at 0x1640f3d9af0>



ACCURACY

In [28]:
```
lr.score(x_test,y_test)
```
Out[28]: 0.440285701113897

In [29]:
```
lr.score(x_train,y_train)
```
Out[29]: 0.42177656427194166

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(alpha=10)

Accuracy(Ridge)

In [32]:
```
rr.score(x_test,y_test)
```
Out[32]: 0.42066682345241824

In [33]:
```
rr.score(x_train,y_train)
```
Out[33]: 0.4106663168409096

In [34]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[34]: Lasso(alpha=10)

In [35]:
```
la.score(x_train,y_train)
```
Out[35]: 0.18477025063618868

Accuracy(Lasso)

In [36]:
```
la.score(x_test,y_test)
```
Out[36]: 0.17862590308954918

Elastic Net

In [37]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[37]:
```
ElasticNet()
```

In [38]:
```
en.coef_
```

Out[38]:
```
array([-0.        ,  0.17752022,  2.88985346, -1.23889796, -1.17115414,
        0.0540545 , -0.13542696,  0.27729277, -0.04681747, -0.11967135,
       -0.        ,  2.4915881 ,  0.        ,  7.12960446])
```

In [39]:
```
en.intercept_
```

Out[39]:
```
28079028.584218573
```

In [40]:
```
prediction=en.predict(x_test)
```

In [41]:
```
en.score(x_test,y_test)
```

Out[41]:
```
0.2308125912534129
```

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)))
```

```
30.90783399728127
1081.6427518050493
32.888337626049896
```

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(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], dtype=int64)
```

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()
```

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(cv=2, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 2, 3, 4, 5],
                         'min_samples_leaf': [5, 10, 15, 20, 25],
                         'n_estimators': [10, 20, 30, 40, 50]},
             scoring='accuracy')
```

In [60]:
```
grid_search.best_score_
```

Out[60]:
```
0.9273467638234033
```

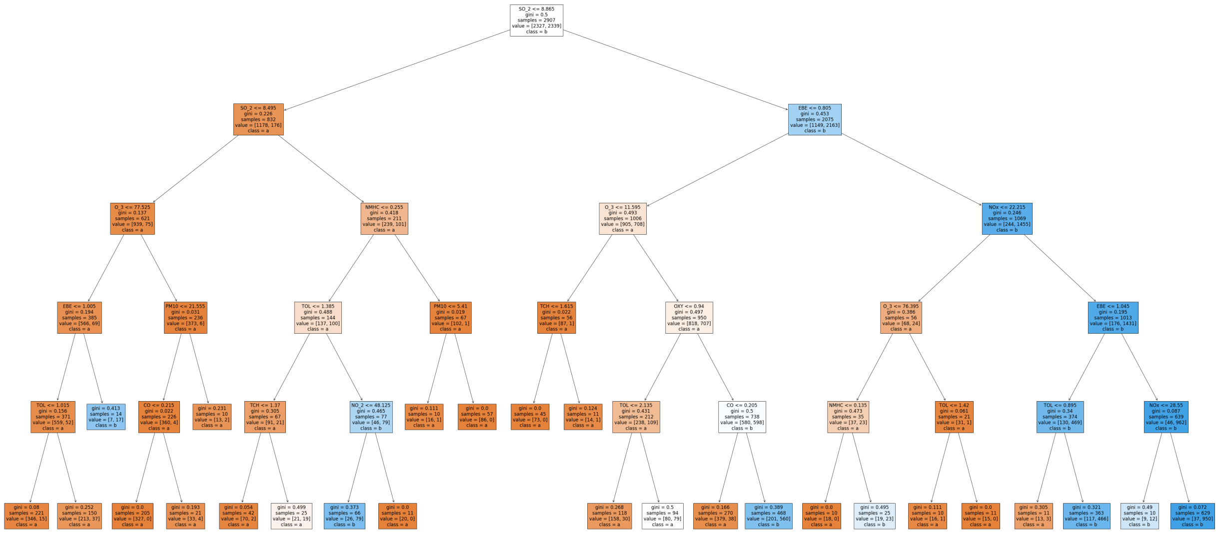
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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','
```

Out[62]:
```
[Text(1965.1304347826087, 1993.2, 'SO_2 <= 8.865\ngini = 0.5\nsamples = 2907\nvalue
= [2327, 2339]\nclass = b'),
 Text(946.1739130434783, 1630.8000000000002, 'SO_2 <= 8.495\ngini = 0.226\nsamples =
832\nvalue = [1178, 176]\nclass = a'),
 Text(485.2173913043478, 1268.4, 'O_3 <= 77.525\ngini = 0.137\nsamples = 621\nvalue
= [939, 75]\nclass = a'),
 Text(291.1304347826087, 906.0, 'EBE <= 1.005\ngini = 0.194\nsamples = 385\nvalue =
[566, 69]\nclass = a'),
 Text(194.08695652173913, 543.5999999999999, 'TOL <= 1.015\ngini = 0.156\nsamples =
371\nvalue = [559, 52]\nclass = a'),
 Text(97.04347826086956, 181.19999999999982, 'gini = 0.08\nsamples = 221\nvalue = [3
46, 15]\nclass = a'),
 Text(291.1304347826087, 181.19999999999982, 'gini = 0.252\nsamples = 150\nvalue =
[213, 37]\nclass = a'),
 Text(388.17391304347825, 543.5999999999999, 'gini = 0.413\nsamples = 14\nvalue =
[7, 17]\nclass = b'),
 Text(679.304347826087, 906.0, 'PM10 <= 21.555\ngini = 0.031\nsamples = 236\nvalue =
[373, 6]\nclass = a'),
 Text(582.2608695652174, 543.5999999999999, 'CO <= 0.215\ngini = 0.022\nsamples = 22
6\nvalue = [360, 4]\nclass = a'),
 Text(485.2173913043478, 181.19999999999982, 'gini = 0.0\nsamples = 205\nvalue = [32
7, 0]\nclass = a'),
 Text(679.304347826087, 181.19999999999982, 'gini = 0.193\nsamples = 21\nvalue = [3
3, 4]\nclass = a'),
 Text(776.3478260869565, 543.5999999999999, 'gini = 0.231\nsamples = 10\nvalue = [1
3, 2]\nclass = a'),
 Text(1407.1304347826087, 1268.4, 'NMHC <= 0.255\ngini = 0.418\nsamples = 211\nvalue
= [239, 101]\nclass = a'),
 Text(1164.5217391304348, 906.0, 'TOL <= 1.385\ngini = 0.488\nsamples = 144\nvalue =
[137, 100]\nclass = a'),
 Text(970.4347826086956, 543.5999999999999, 'TCH <= 1.37\ngini = 0.305\nsamples = 67
\nvalue = [91, 21]\nclass = a'),
 Text(873.391304347826, 181.19999999999982, 'gini = 0.054\nsamples = 42\nvalue = [7
0, 2]\nclass = a'),
```

```
 Text(1067.4782608695652, 181.19999999999982, 'gini = 0.499\nsamples = 25\nvalue =
[21, 19]\nclass = a'),
 Text(1358.608695652174, 543.5999999999999, 'NO_2 <= 48.125\ngini = 0.465\nsamples =
77\nvalue = [46, 79]\nclass = b'),
 Text(1261.5652173913043, 181.19999999999982, 'gini = 0.373\nsamples = 66\nvalue =
[26, 79]\nclass = b'),
 Text(1455.6521739130435, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [2
0, 0]\nclass = a'),
 Text(1649.7391304347825, 906.0, 'PM10 <= 5.41\ngini = 0.019\nsamples = 67\nvalue =
[102, 1]\nclass = a'),
 Text(1552.695652173913, 543.5999999999999, 'gini = 0.111\nsamples = 10\nvalue = [1
6, 1]\nclass = a'),
 Text(1746.782608695652, 543.5999999999999, 'gini = 0.0\nsamples = 57\nvalue = [86,
0]\nclass = a'),
 Text(2984.086956521739, 1630.8000000000002, 'EBE <= 0.805\ngini = 0.453\nsamples =
2075\nvalue = [1149, 2163]\nclass = b'),
 Text(2280.5217391304345, 1268.4, 'O_3 <= 11.595\ngini = 0.493\nsamples = 1006\nvalu
e = [905, 708]\nclass = a'),
 Text(2037.9130434782608, 906.0, 'TCH <= 1.615\ngini = 0.022\nsamples = 56\nvalue =
[87, 1]\nclass = a'),
 Text(1940.8695652173913, 543.5999999999999, 'gini = 0.0\nsamples = 45\nvalue = [73,
0]\nclass = a'),
 Text(2134.9565217391305, 543.5999999999999, 'gini = 0.124\nsamples = 11\nvalue = [1
4, 1]\nclass = a'),
 Text(2523.1304347826085, 906.0, 'OXY <= 0.94\ngini = 0.497\nsamples = 950\nvalue =
[818, 707]\nclass = a'),
 Text(2329.0434782608695, 543.5999999999999, 'TOL <= 2.135\ngini = 0.431\nsamples =
212\nvalue = [238, 109]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.268\nsamples = 118\nvalue = [158, 30]\nc
lass = a'),
 Text(2426.086956521739, 181.19999999999982, 'gini = 0.5\nsamples = 94\nvalue = [80,
79]\nclass = a'),
 Text(2717.217391304348, 543.5999999999999, 'CO <= 0.205\ngini = 0.5\nsamples = 738
\nvalue = [580, 598]\nclass = b'),
 Text(2620.173913043478, 181.19999999999982, 'gini = 0.166\nsamples = 270\nvalue =
[379, 38]\nclass = a'),
 Text(2814.2608695652175, 181.19999999999982, 'gini = 0.389\nsamples = 468\nvalue =
[201, 560]\nclass = b'),
 Text(3687.6521739130435, 1268.4, 'NOx <= 22.215\ngini = 0.246\nsamples = 1069\nvalu
e = [244, 1455]\nclass = b'),
 Text(3299.478260869565, 906.0, 'O_3 <= 76.395\ngini = 0.386\nsamples = 56\nvalue =
[68, 24]\nclass = a'),
 Text(3105.391304347826, 543.5999999999999, 'NMHC <= 0.135\ngini = 0.473\nsamples =
35\nvalue = [37, 23]\nclass = a'),
 Text(3008.3478260869565, 181.19999999999982, 'gini = 0.0\nsamples = 10\nvalue = [1
8, 0]\nclass = a'),
 Text(3202.4347826086955, 181.19999999999982, 'gini = 0.495\nsamples = 25\nvalue =
[19, 23]\nclass = b'),
 Text(3493.565217391304, 543.5999999999999, 'TOL <= 1.42\ngini = 0.061\nsamples = 21
\nvalue = [31, 1]\nclass = a'),
 Text(3396.5217391304345, 181.19999999999982, 'gini = 0.111\nsamples = 10\nvalue =
[16, 1]\nclass = a'),
 Text(3590.608695652174, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [15,
0]\nclass = a'),
 Text(4075.8260869565215, 906.0, 'EBE <= 1.045\ngini = 0.195\nsamples = 1013\nvalue
= [176, 1431]\nclass = b'),
 Text(3881.7391304347825, 543.5999999999999, 'TOL <= 0.895\ngini = 0.34\nsamples = 3
74\nvalue = [130, 469]\nclass = b'),
 Text(3784.695652173913, 181.19999999999982, 'gini = 0.305\nsamples = 11\nvalue = [1
3, 3]\nclass = a'),
 Text(3978.782608695652, 181.19999999999982, 'gini = 0.321\nsamples = 363\nvalue =
[117, 466]\nclass = b'),
 Text(4269.913043478261, 543.5999999999999, 'NOx <= 28.55\ngini = 0.087\nsamples = 6
39\nvalue = [46, 962]\nclass = b'),
 Text(4172.869565217391, 181.19999999999982, 'gini = 0.49\nsamples = 10\nvalue = [9,
12]\nclass = b'),
 Text(4366.95652173913, 181.19999999999982, 'gini = 0.072\nsamples = 629\nvalue = [3
7, 950]\nclass = b')]
```



Conclusion

Scores

Linear Regression

In [63]:
```
lr.score(x_test,y_test)
```
Out[63]: 0.440285701113897

In [64]:
```
lr.score(x_train,y_train)
```
Out[64]: 0.42177656427194166

Lasso

In [65]:
```
la.score(x_test,y_test)
```
Out[65]: 0.17862590308954918

Ridge

In [66]:
```
rr.score(x_test,y_test)
```
Out[66]: 0.42066682345241824

In [67]:
```
rr.score(x_train,y_train)
```
Out[67]: 0.4106663168409096

Elastic Net

In [68]:
```
en.score(x_test,y_test)
```
Out[68]: 0.2308125912534129

Logistic Regression

In [69]:
```
logr.score(fs,target_vector)
```
Out[69]: 0.8660366036603661

Random Forest

In [70]:
```
grid_search.best_score_
```
Out[70]: 0.9273467638234033

From the above data, we can conclude that random forest regression is preferrable to other regression types

In [ ]:
```

```