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(r"C:\Users\user\Downloads\madrid_2010.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
0	01-03-2010 01:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	NaN	
1	01-03-2010 01:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	NaN	
2	01-03-2010 01:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	NaN	
3	01-03-2010 01:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.410000	7.8
4	01-03-2010 01:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.670000	22.0
...	...	...	...	...	...	...	...	...	...	...	...	...
209443	01-08-2010 00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	NaN	
209444	01-08-2010 00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.259998	
209445	01-08-2010 00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	NaN	

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
209446	01-08-2010 00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	NaN	
209447	01-08-2010 00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.150002	26.8

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]:

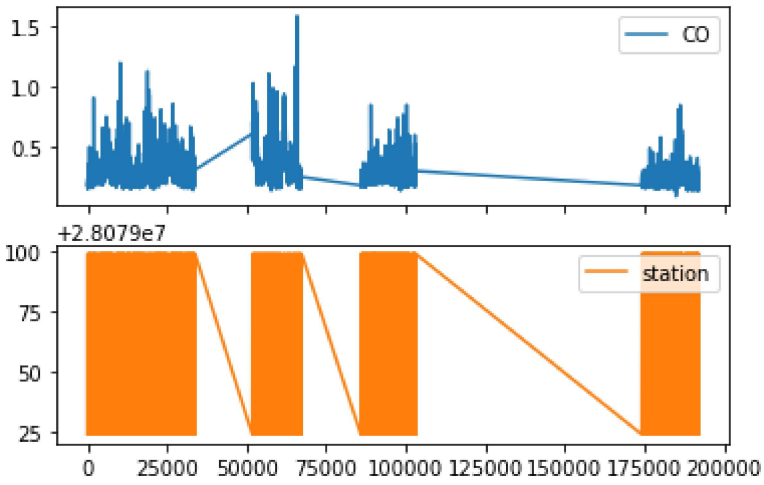
	CO	station
11	0.18	28079024
23	0.23	28079099
35	0.17	28079024
47	0.21	28079099
59	0.16	28079024
...	...	...
191879	0.26	28079099
191891	0.16	28079024
191903	0.28	28079099
191915	0.16	28079024
191927	0.25	28079099

6666 rows × 2 columns

Line chart

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

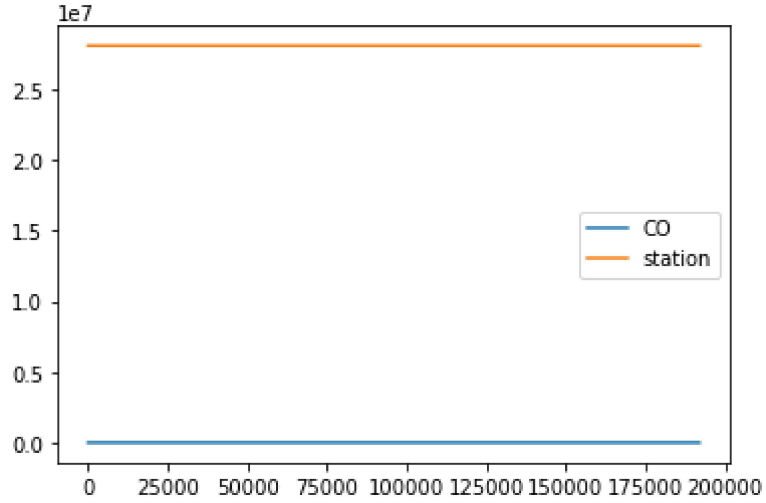
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

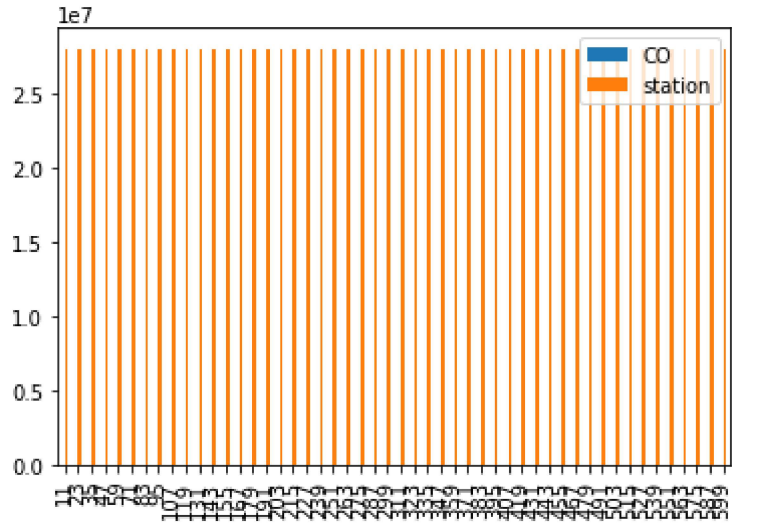


Bar chart

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

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

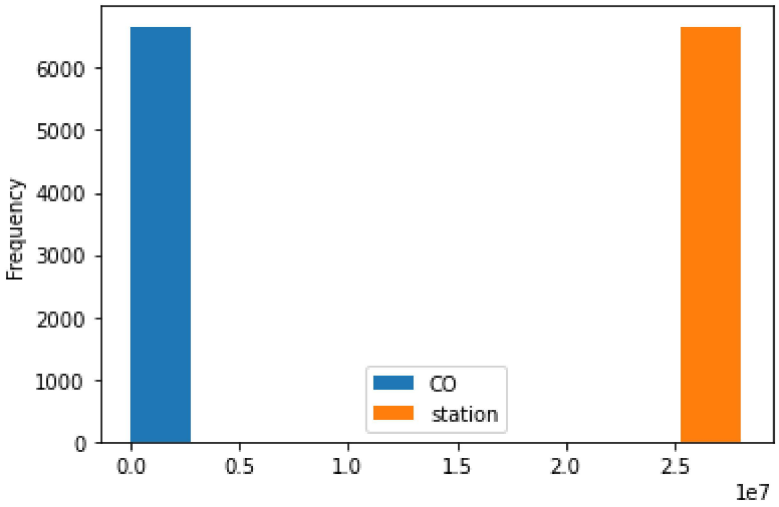
Out[10]: <AxesSubplot:>



Histogram

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

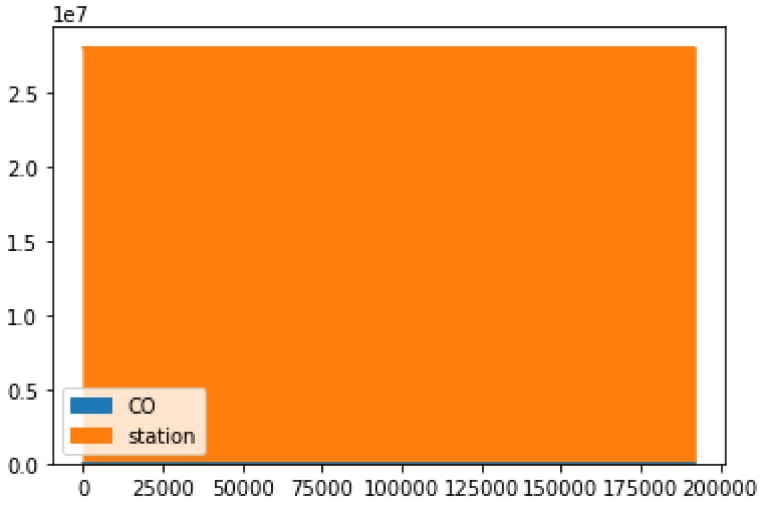
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

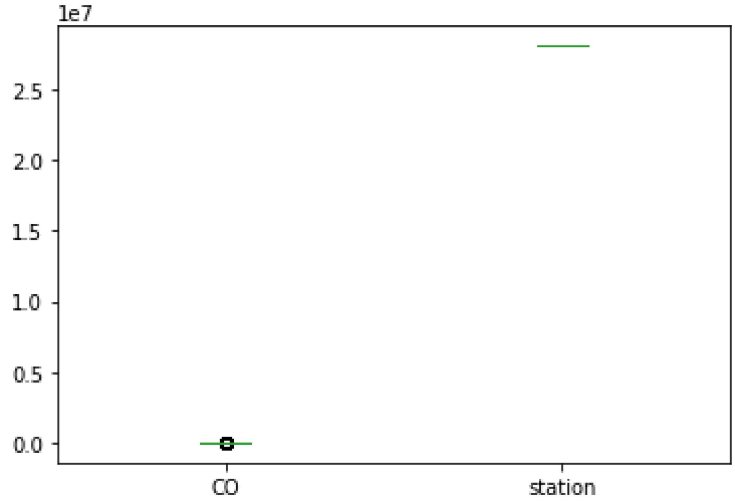
Out[12]: <AxesSubplot:>



Box chart

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

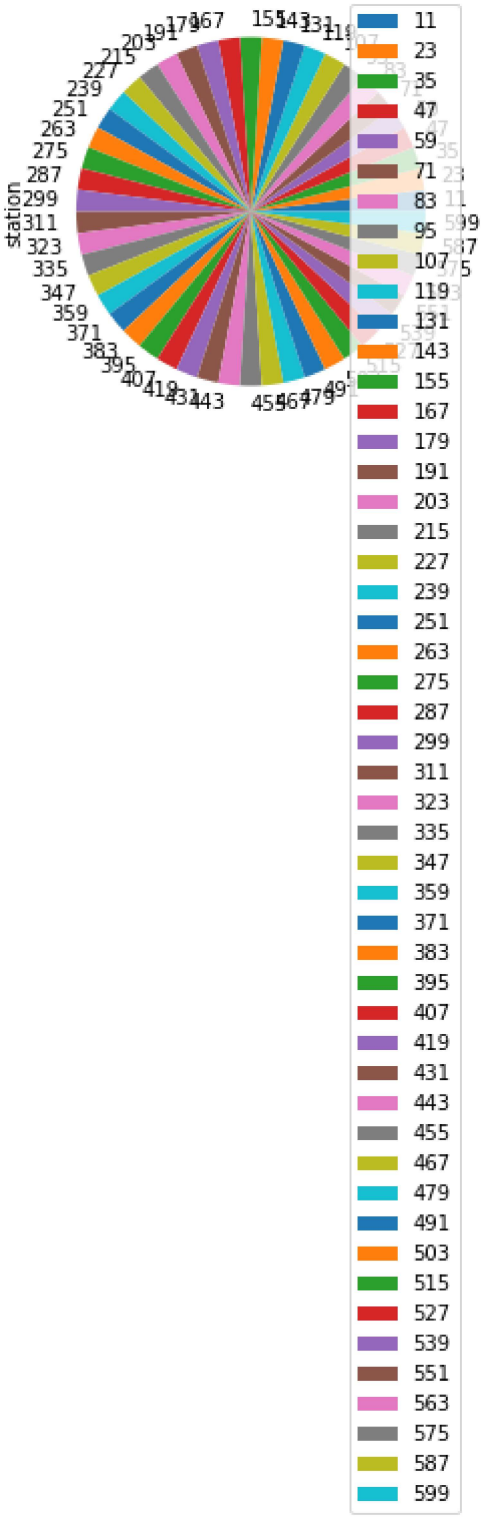
Out[13]: <AxesSubplot:>



Pie chart

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

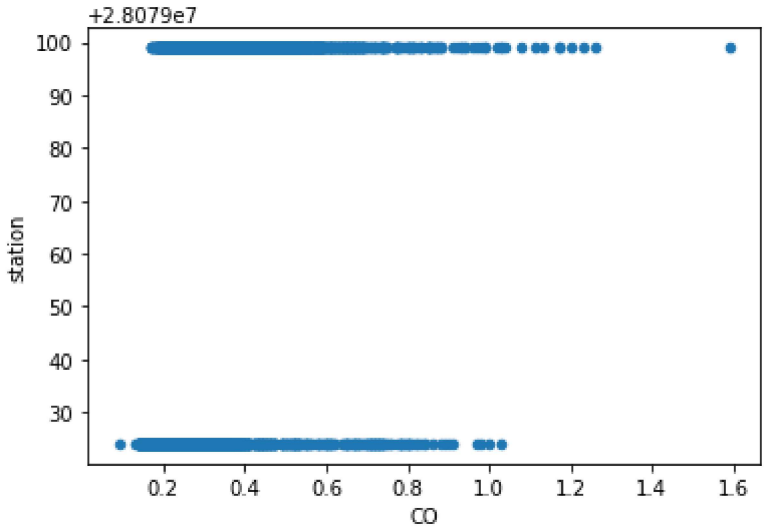
Out[14]:
```
<AxesSubplot:ylabel='station'>
```



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	6666.
mean	0.648425	0.296280	0.840585	0.839959	0.243378	33.888744	47.540617	0.
std	0.395346	0.133296	0.508031	0.382263	0.115730	23.465169	41.230578	0.
min	0.170000	0.090000	0.140000	0.110000	0.000000	1.290000	2.760000	0.
25%	0.380000	0.200000	0.470000	0.590000	0.180000	15.752500	19.442501	1.
50%	0.540000	0.260000	0.755000	1.000000	0.220000	29.320000	36.770000	1.

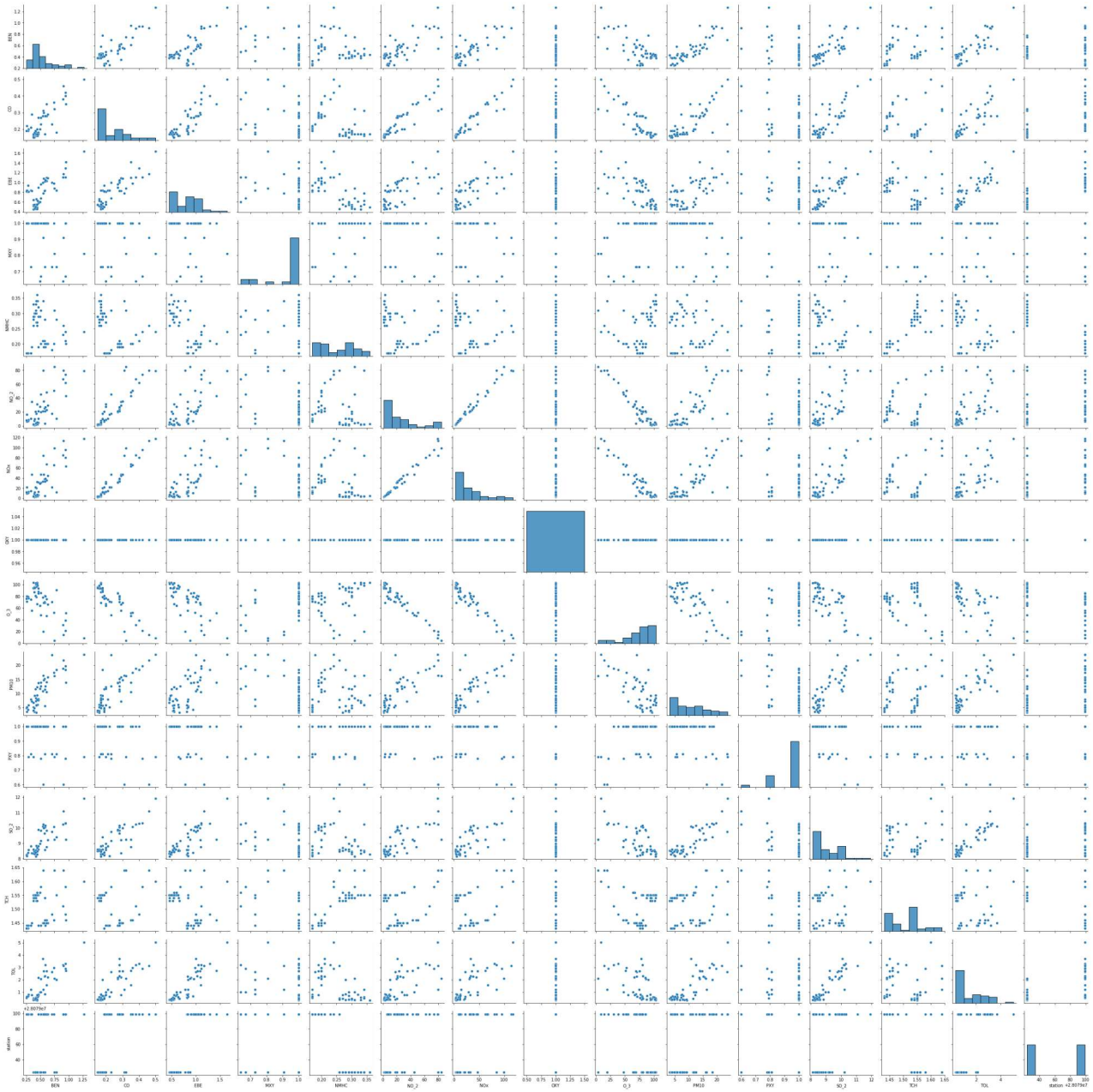
	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	
75%	0.810000	0.340000	1.000000	1.000000	0.280000	47.657500	62.102501	1.
max	5.110000	1.590000	5.190000	6.810000	0.930000	133.399994	409.299988	2.

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])
```

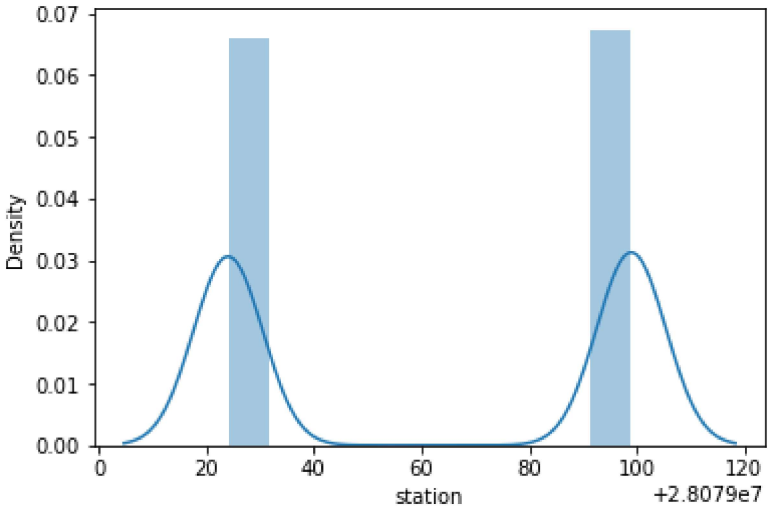
Out[19]: <seaborn.axisgrid.PairGrid at 0x264f3ca8430>



In [20]:
```
sns.distplot(df1['station'])
```

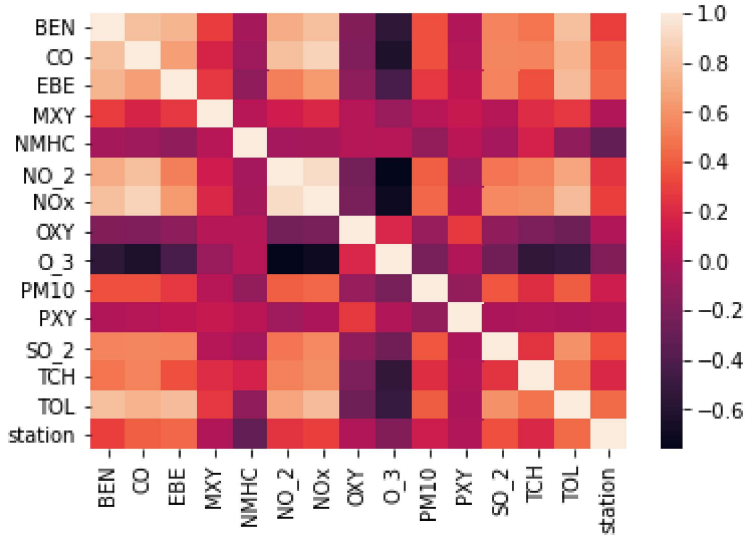
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28078937.64139527
```

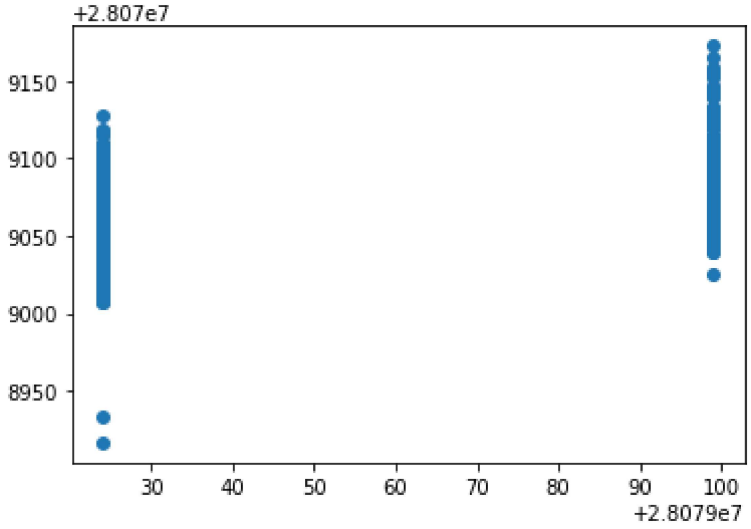
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-38.386991
CO	159.509076
EBE	16.424092
MXY	-9.333602
NMHC	-75.670402
NO_2	0.267954
NOx	-0.556646
OXY	28.437369
O_3	0.075757
PM10	-0.172624
PXY	-5.544565
SO_2	1.842959
TCH	48.171538
TOL	9.655533

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x2648334b9d0>
```



ACCURACY

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

Out[28]: 0.4446045055136888

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

Out[29]: 0.41970559594901846

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]: 0.42816955450560223

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

Out[33]: 0.40852124529603007

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]:
```
0.17849488279818082
```

Accuracy(Lasso)

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

Out[36]:
```
0.17584035546046928
```

Elastic Net regression

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.        ,  0.15547032,  2.48079023, -1.09064889, -1.28596159,
       -0.03883731, -0.06105292,  0.29512578, -0.02893951, -0.08836445,
       -0.        ,  2.44154204,  0.        ,  6.87050073])
```

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

Out[39]:
```
28079028.100049313
```

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

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

Out[41]:
```
0.24011088810927483
```

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)))
```

```
30.778920363623648
1068.555594210043
32.68876862486629
```

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]:
```
0.9292756108015431
```

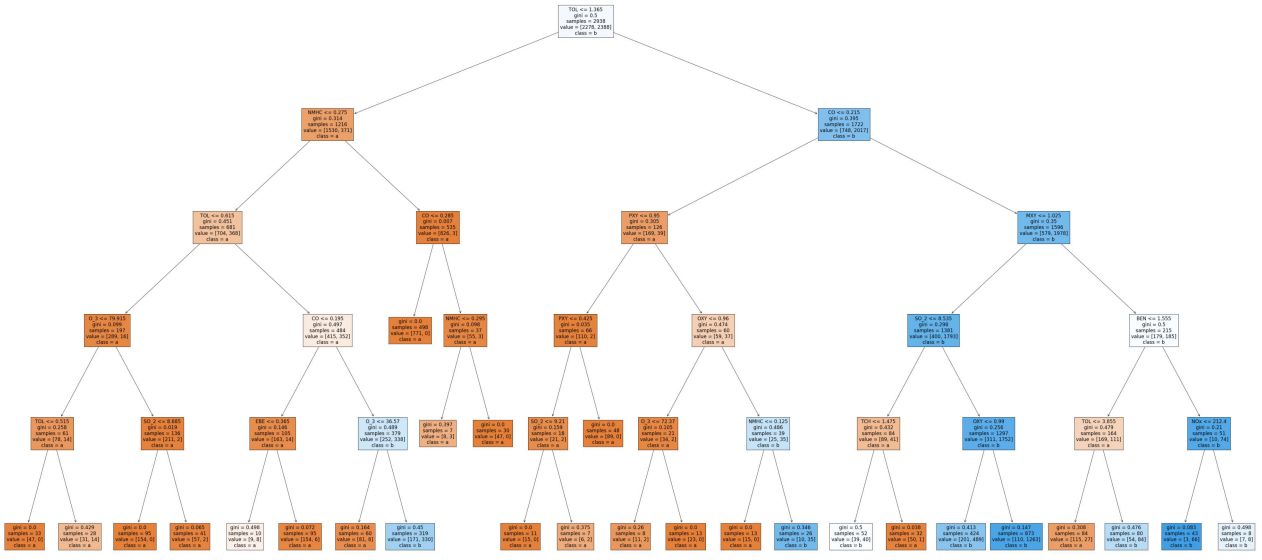
In [61]:
```
rfc_best=grid_search.best_estimator_
```

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']
```

Out[62]:
```
[Text(2074.304347826087, 1993.2, 'TOL <= 1.365\ngini = 0.5\nsamples = 2938\nvalue = [227
8, 2388]\nclass = b'),
 Text(1164.5217391304348, 1630.8000000000002, 'NMHC <= 0.275\ngini = 0.314\nsamples = 12
16\nvalue = [1530, 371]\nclass = a'),
 Text(776.3478260869565, 1268.4, 'TOL <= 0.615\ngini = 0.451\nsamples = 681\nvalue = [70
4, 368]\nclass = a'),
 Text(388.17391304347825, 906.0, 'O_3 <= 79.915\ngini = 0.099\nsamples = 197\nvalue = [2
89, 16]\nclass = a'),
 Text(194.08695652173913, 543.5999999999999, 'TOL <= 0.515\ngini = 0.258\nsamples = 61\n
value = [78, 14]\nclass = a'),
 Text(97.04347826086956, 181.19999999999982, 'gini = 0.0\nsamples = 33\nvalue = [47, 0]
\nclass = a'),
 Text(291.1304347826087, 181.19999999999982, 'gini = 0.429\nsamples = 28\nvalue = [31, 1
4]\nclass = a'),
 Text(582.2608695652174, 543.5999999999999, 'SO_2 <= 8.685\ngini = 0.019\nsamples = 136
\nvalue = [211, 2]\nclass = a'),
 Text(485.2173913043478, 181.19999999999982, 'gini = 0.0\nsamples = 95\nvalue = [154, 0]
\nclass = a'),
 Text(679.304347826087, 181.19999999999982, 'gini = 0.065\nsamples = 41\nvalue = [57, 2]
\nclass = a'),
 Text(1164.5217391304348, 906.0, 'CO <= 0.195\ngini = 0.497\nsamples = 484\nvalue = [41
5, 352]\nclass = a'),
 Text(970.4347826086956, 543.5999999999999, 'EBE <= 0.365\ngini = 0.146\nsamples = 105\n
value = [163, 14]\nclass = a'),
 Text(873.391304347826, 181.19999999999982, 'gini = 0.498\nsamples = 10\nvalue = [9, 8]
\nclass = a'),
 Text(1067.4782608695652, 181.19999999999982, 'gini = 0.072\nsamples = 95\nvalue = [154,
6]\nclass = a'),
 Text(1358.608695652174, 543.5999999999999, 'O_3 <= 36.57\ngini = 0.489\nsamples = 379\n
value = [252, 338]\nclass = b'),
 Text(1261.5652173913043, 181.19999999999982, 'gini = 0.164\nsamples = 60\nvalue = [81,
8]\nclass = a'),
 Text(1455.6521739130435, 181.19999999999982, 'gini = 0.45\nsamples = 319\nvalue = [171,
330]\nclass = b'),
 Text(1552.695652173913, 1268.4, 'CO <= 0.285\ngini = 0.007\nsamples = 535\nvalue = [82
6, 3]\nclass = a'),
 Text(1455.6521739130435, 906.0, 'gini = 0.0\nsamples = 498\nvalue = [771, 0]\nclass =
a'),
 Text(1649.7391304347825, 906.0, 'NMHC <= 0.295\ngini = 0.098\nsamples = 37\nvalue = [5
5, 3]\nclass = a'),
 Text(1552.695652173913, 543.5999999999999, 'gini = 0.397\nsamples = 7\nvalue = [8, 3]\n
class = a'),
 Text(1746.782608695652, 543.5999999999999, 'gini = 0.0\nsamples = 30\nvalue = [47, 0]\n
class = a'),
 Text(2984.086956521739, 1630.8000000000002, 'CO <= 0.215\ngini = 0.395\nsamples = 1722
\nvalue = [748, 2017]\nclass = b'),
 Text(2280.5217391304345, 1268.4, 'PXY <= 0.95\ngini = 0.305\nsamples = 126\nvalue = [16
9, 39]\nclass = a'),
 Text(2037.9130434782608, 906.0, 'PXY <= 0.425\ngini = 0.035\nsamples = 66\nvalue = [11
0, 2]\nclass = a'),
 Text(1940.8695652173913, 543.5999999999999, 'SO_2 <= 9.21\ngini = 0.159\nsamples = 18\n
value = [21, 2]\nclass = a'),
 Text(1843.8260869565217, 181.19999999999982, 'gini = 0.0\nsamples = 11\nvalue = [15, 0]
\nclass = a'),
 Text(2037.9130434782608, 181.19999999999982, 'gini = 0.375\nsamples = 7\nvalue = [6, 2]
\nclass = a'),
 Text(2134.9565217391305, 543.5999999999999, 'gini = 0.0\nsamples = 48\nvalue = [89, 0]
\nclass = a'),
 Text(2523.1304347826085, 906.0, 'OXY <= 0.96\ngini = 0.474\nsamples = 60\nvalue = [59,
37]\nclass = a'),
 Text(2329.0434782608695, 543.5999999999999, 'O_3 <= 72.37\ngini = 0.105\nsamples = 21\n
value = [34, 2]\nclass = a'),
 Text(2232.0, 181.19999999999982, 'gini = 0.26\nsamples = 8\nvalue = [11, 2]\nclass =
a'),
 Text(2426.086956521739, 181.19999999999982, 'gini = 0.0\nsamples = 13\nvalue = [23, 0]
```

```
\nclass = a'),
 Text(2717.217391304348, 543.5999999999999, 'NMHC <= 0.125\ngini = 0.486\nsamples = 39\n
value = [25, 35]\nclass = b'),
 Text(2620.173913043478, 181.19999999999982, 'gini = 0.0\nsamples = 13\nvalue = [15, 0]
\nclass = a'),
 Text(2814.2608695652175, 181.19999999999982, 'gini = 0.346\nsamples = 26\nvalue = [10,
35]\nclass = b'),
 Text(3687.6521739130435, 1268.4, 'MXY <= 1.025\ngini = 0.35\nsamples = 1596\nvalue = [5
79, 1978]\nclass = b'),
 Text(3299.478260869565, 906.0, 'SO_2 <= 8.535\ngini = 0.298\nsamples = 1381\nvalue = [4
00, 1793]\nclass = b'),
 Text(3105.391304347826, 543.5999999999999, 'TCH <= 1.475\ngini = 0.432\nsamples = 84\nv
alue = [89, 41]\nclass = a'),
 Text(3008.3478260869565, 181.19999999999982, 'gini = 0.5\nsamples = 52\nvalue = [39, 4
0]\nclass = b'),
 Text(3202.4347826086955, 181.19999999999982, 'gini = 0.038\nsamples = 32\nvalue = [50,
1]\nclass = a'),
 Text(3493.565217391304, 543.5999999999999, 'OXY <= 0.99\ngini = 0.256\nsamples = 1297\n
value = [311, 1752]\nclass = b'),
 Text(3396.5217391304345, 181.19999999999982, 'gini = 0.413\nsamples = 424\nvalue = [20
1, 489]\nclass = b'),
 Text(3590.608695652174, 181.19999999999982, 'gini = 0.147\nsamples = 873\nvalue = [110,
1263]\nclass = b'),
 Text(4075.8260869565215, 906.0, 'BEN <= 1.555\ngini = 0.5\nsamples = 215\nvalue = [179,
185]\nclass = b'),
 Text(3881.7391304347825, 543.5999999999999, 'TOL <= 3.855\ngini = 0.479\nsamples = 164
\nvalue = [169, 111]\nclass = a'),
 Text(3784.695652173913, 181.19999999999982, 'gini = 0.308\nsamples = 84\nvalue = [115,
27]\nclass = a'),
 Text(3978.782608695652, 181.19999999999982, 'gini = 0.476\nsamples = 80\nvalue = [54, 8
4]\nclass = b'),
 Text(4269.913043478261, 543.5999999999999, 'NOx <= 212.4\ngini = 0.21\nsamples = 51\nva
lue = [10, 74]\nclass = b'),
 Text(4172.869565217391, 181.19999999999982, 'gini = 0.083\nsamples = 43\nvalue = [3, 6
6]\nclass = b'),
 Text(4366.95652173913, 181.19999999999982, 'gini = 0.498\nsamples = 8\nvalue = [7, 8]\n
class = b')]
```



Conclusion

Accuracy

linear regression

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

Out[63]: 0.4446045055136888

Ridge regression

In [64]:
```
rr.score(x_test,y_test)
```

Out[64]: 0.42816955450560223

Lasso regression

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

Out[65]: 0.17584035546046928

Elastic net regression

In [66]:
```
en.score(x_test,y_test)
```

Out[66]: 0.24011088810927483

Logistic regression

In [67]:
```
logr.score(fs,target_vector)
```

Out[67]: 0.8660366036603661

Random forest

In [68]:
```
grid_search.best_score_
```

Out[68]: 0.9292756108015431

Accuracy for random forest is higher so it is the best fit model