20104028

Heamnath N

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("madrid_2004.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
0	2004-08-01 01:00:00	NaN	0.66	NaN	NaN	NaN	89.550003	118.900002	NaN	40.020000	39.990002	2
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22.950001	
2	2004-08-01 01:00:00	NaN	1.02	NaN	NaN	NaN	93.389999	138.600006	NaN	20.860001	49.480000	
3	2004-08-01 01:00:00	NaN	0.53	NaN	NaN	NaN	87.290001	105.000000	NaN	36.730000	31.070000	
4	2004-08-01 01:00:00	NaN	0.17	NaN	NaN	NaN	34.910000	35.349998	NaN	86.269997	54.080002	
...	...	...	...	...	...	...	...	...	...	...	...	...
245491	2004-06-01 00:00:00	0.75	0.21	0.85	1.55	0.07	59.580002	64.389999	0.66	33.029999	30.900000	1
245492	2004-06-01 00:00:00	2.49	0.75	2.44	4.57	NaN	97.139999	146.899994	2.34	7.740000	37.689999	
245493	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.13	102.699997	132.600006	NaN	17.809999	22.840000	1

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	
245494	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.09	82.599998	102.599998	NaN	NaN	45.630001	
245495	2004-06-01 00:00:00	3.01	0.67	2.78	5.12	0.20	92.550003	141.000000	2.60	11.460000	24.389999	1

245496 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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     19397 non-null  object 
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [6]:
```
data=df[['CO' ,'station']]
data
```

Out[6]:

	CO	station
5	0.63	28079006

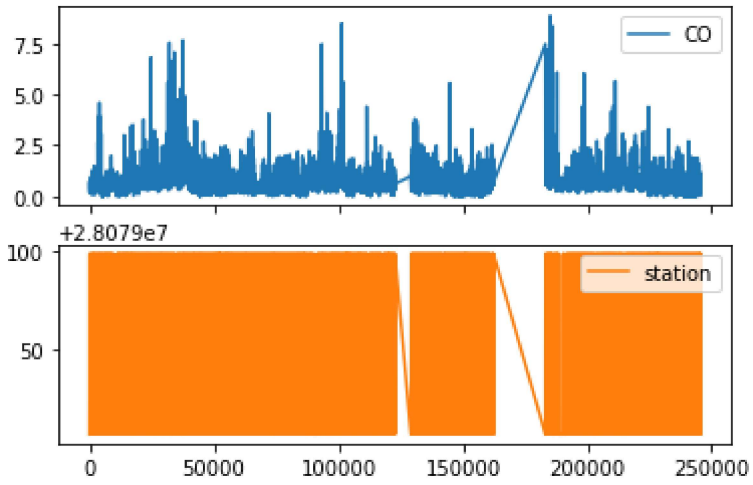
	CO	station
22	0.30	28079024
26	0.46	28079099
32	0.67	28079006
49	0.30	28079024
...	...	...
245463	0.08	28079024
245467	0.67	28079099
245473	1.12	28079006
245491	0.21	28079024
245495	0.67	28079099

19397 rows × 2 columns

Line chart

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

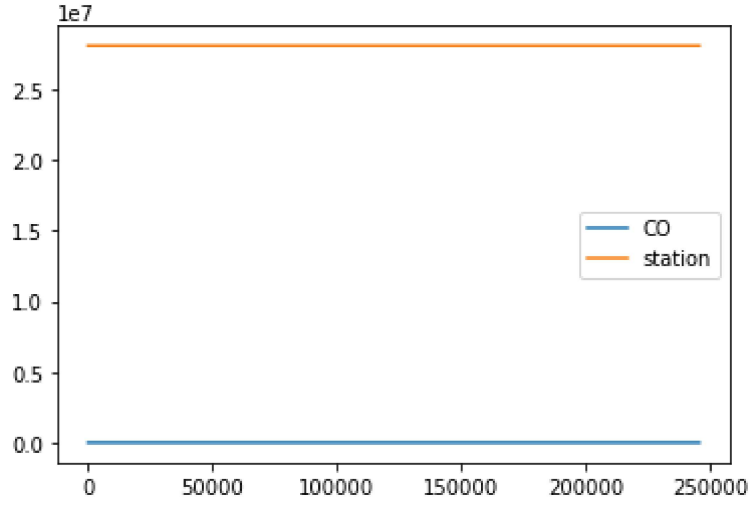
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

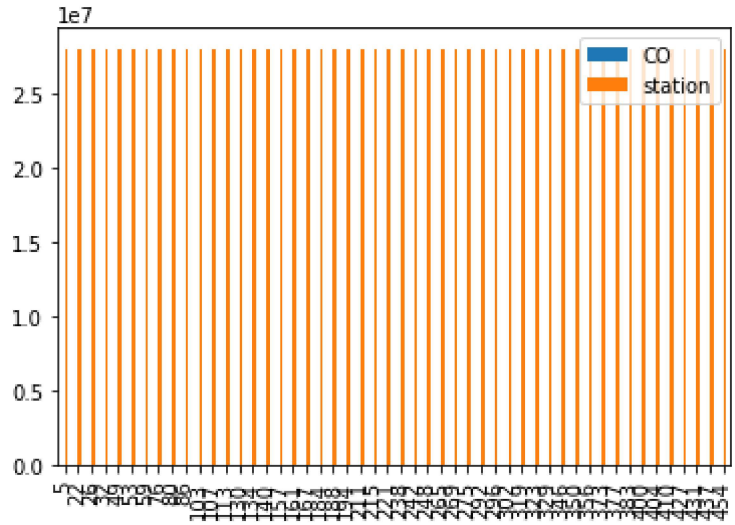


Bar chart

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

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

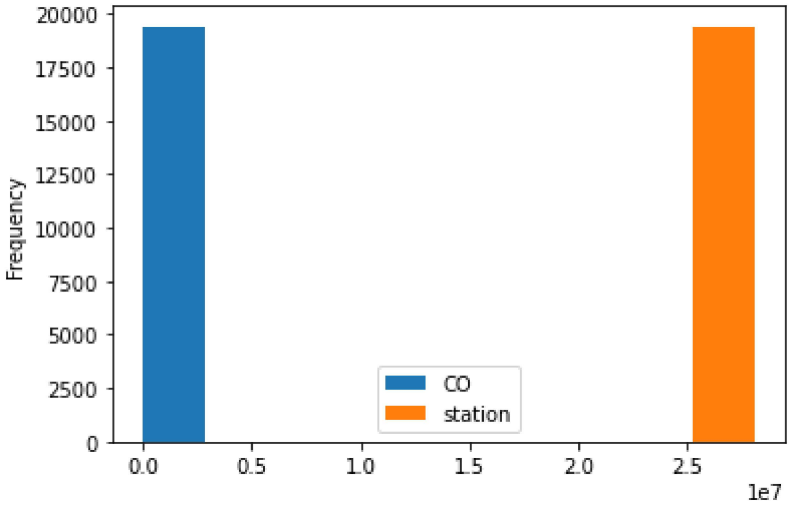
Out[10]: <AxesSubplot:>



Histogram

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

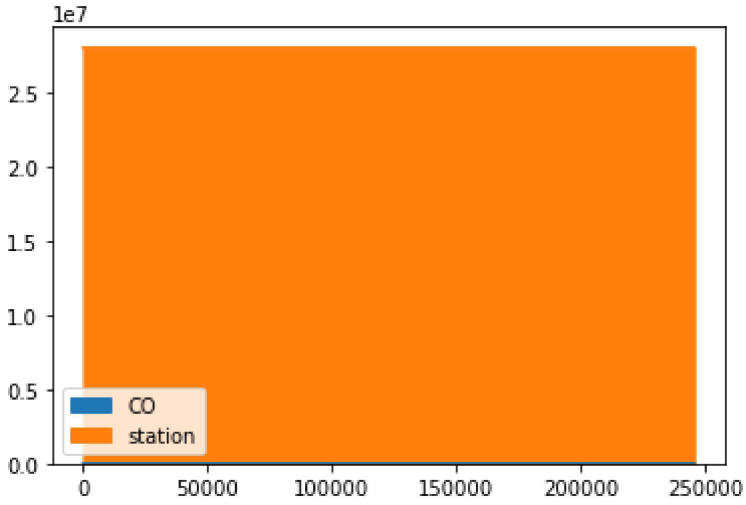
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

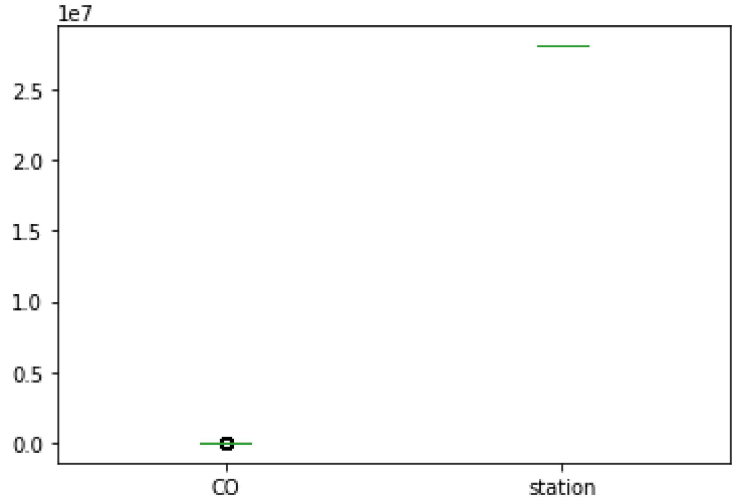
Out[12]: <AxesSubplot:>



Box chart

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

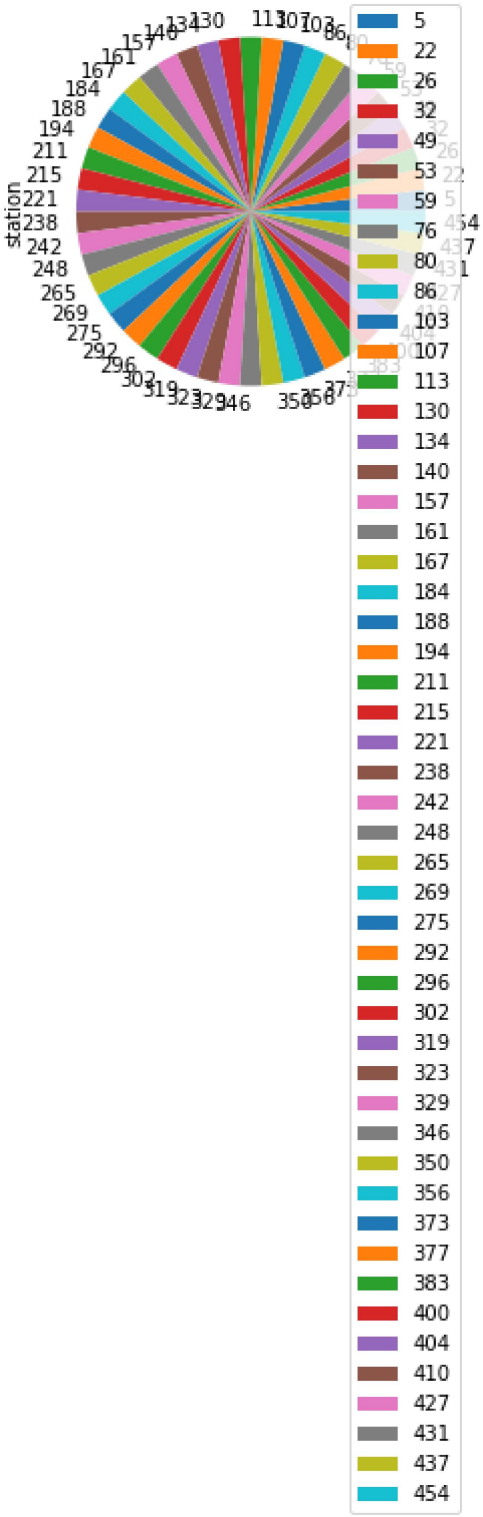
Out[13]: <AxesSubplot:>



Pie chart

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

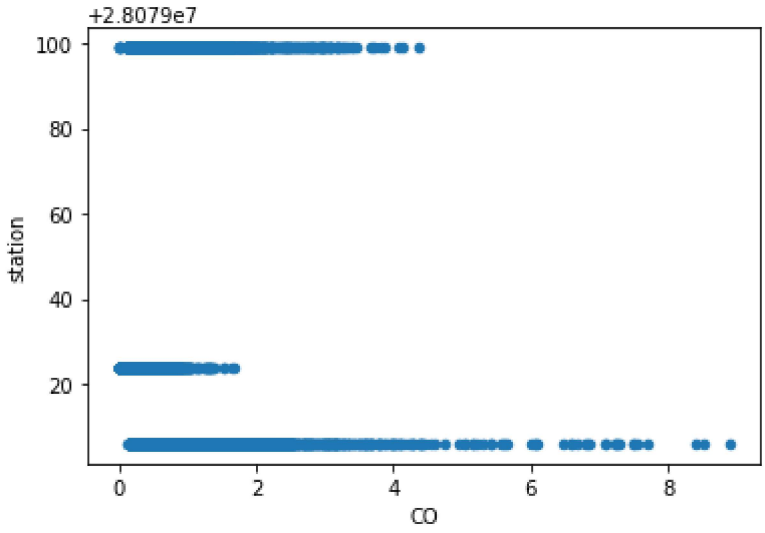
Out[14]: <AxesSubplot:ylabel='station'>



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: 19397 entries, 5 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     19397 non-null  object 
 1   BEN      19397 non-null  float64
 2   CO       19397 non-null  float64
 3   EBE      19397 non-null  float64
 4   MXY      19397 non-null  float64
 5   NMHC     19397 non-null  float64
 6   NO_2     19397 non-null  float64
 7   NOx      19397 non-null  float64
 8   OXY      19397 non-null  float64
 9   O_3      19397 non-null  float64
 10  PM10     19397 non-null  float64
 11  PM25     19397 non-null  float64
 12  PXY      19397 non-null  float64
 13  SO_2     19397 non-null  float64
 14  TCH      19397 non-null  float64
 15  TOL      19397 non-null  float64
 16  station  19397 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.7+ MB
```

In [17]:
```
df.describe()
```

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.000000	19397.00000
mean	2.250781	0.675347	2.775913	5.424809	0.151024	62.887023	128.55402
std	2.184724	0.591026	2.729622	5.554358	0.158603	37.952255	127.91241
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.090000	2.41000
25%	0.870000	0.320000	1.020000	1.780000	0.060000	35.150002	45.20999
50%	1.620000	0.520000	1.970000	3.800000	0.110000	58.310001	93.22000

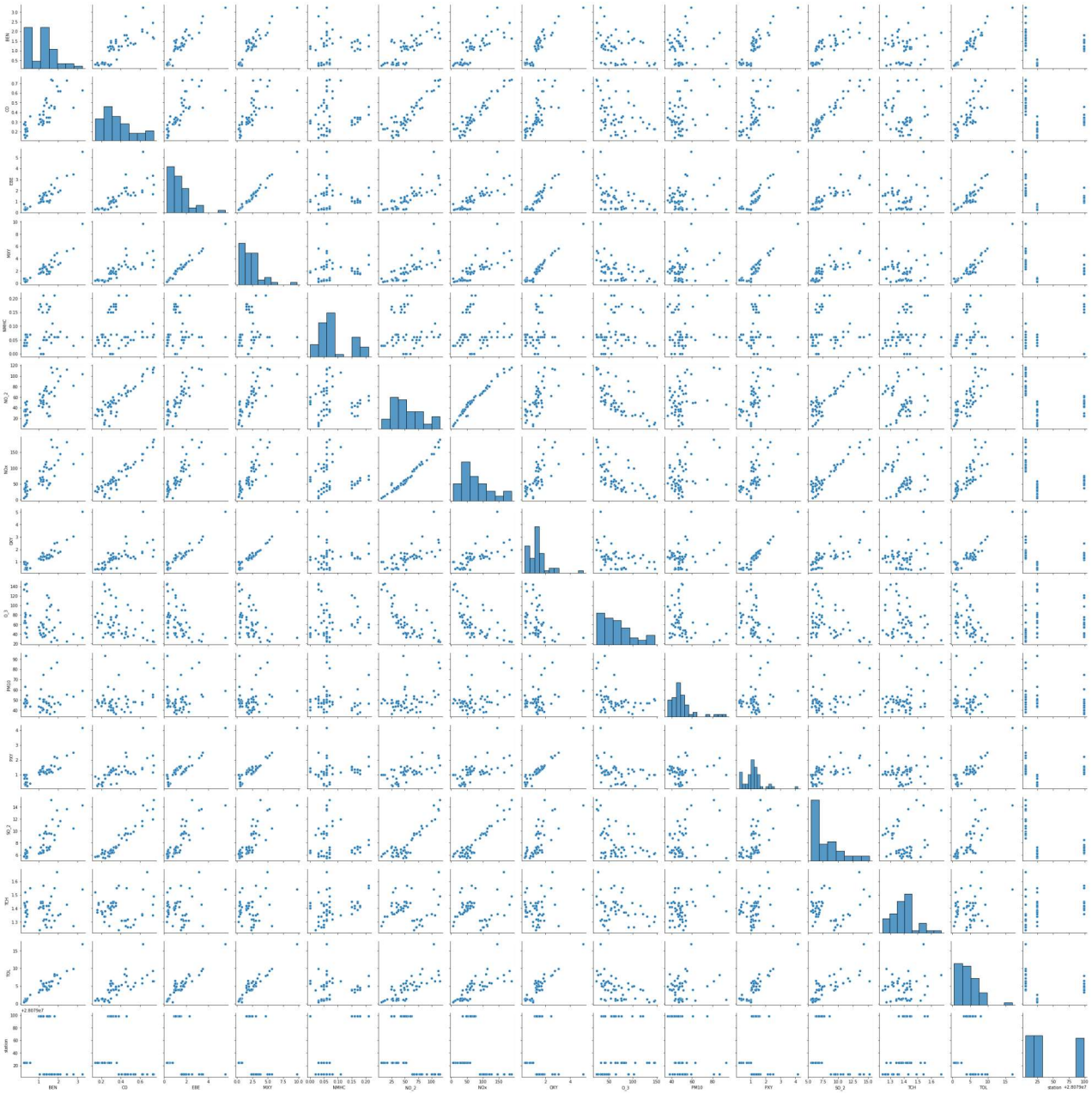
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	2.910000	0.860000	3.580000	7.260000	0.200000	85.730003	174.30000
max	34.180000	8.900000	41.880001	91.599998	4.810000	355.100006	1700.00000

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])
```

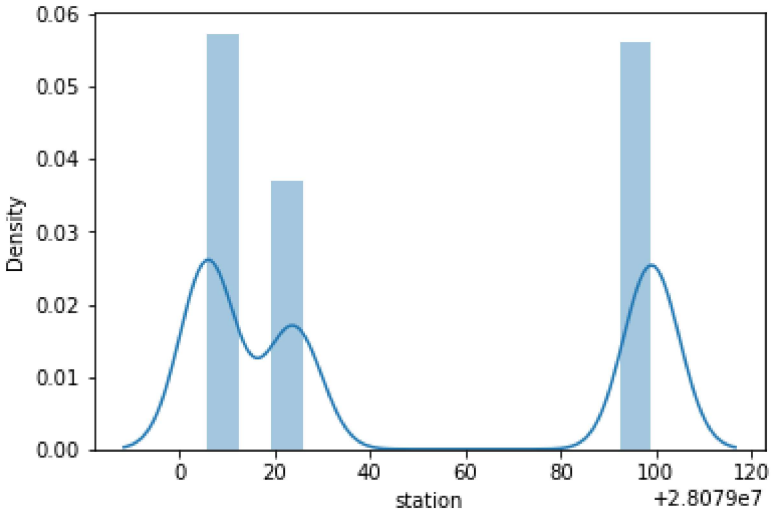
Out[19]: <seaborn.axisgrid.PairGrid at 0x1f7b59cd5e0>



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

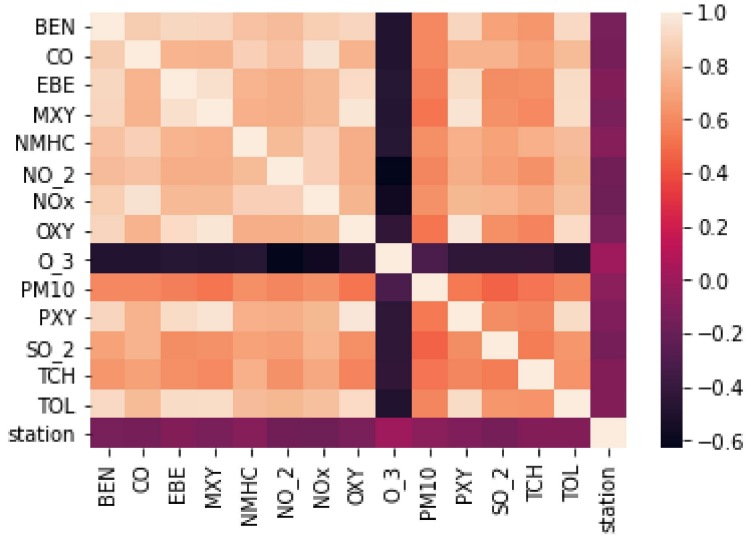
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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]:
```
28079076.91181206
```

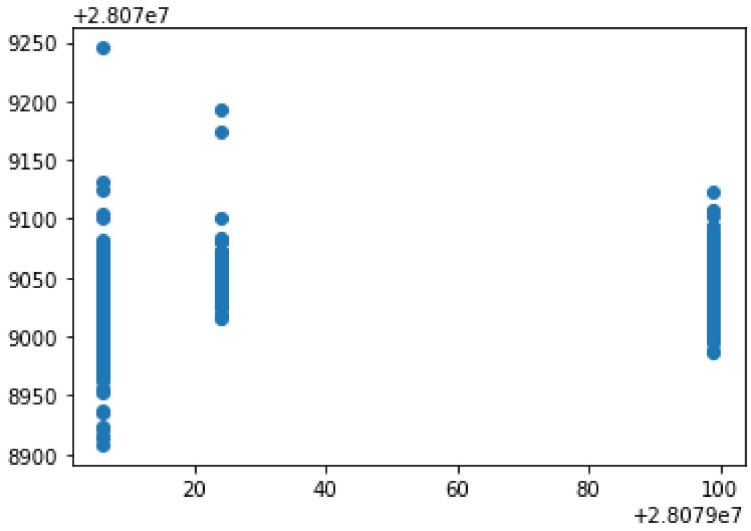
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-4.355807
CO	24.496752
EBE	4.849014
MXY	-3.750852
NMHC	88.871130
NO_2	-0.136418
NOx	-0.253092
OXY	-1.458679
O_3	-0.285837
PM10	0.065431
PXY	5.200924
SO_2	-0.147580
TCH	-8.703978
TOL	1.060865

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1f7c4494910>
```



ACCURACY

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

Out[28]: 0.10767927161857482

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

Out[29]: 0.10569628546834742

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]: 0.10680483588399026

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

Out[33]: 0.10527715630198753

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]:
```
0.05064114678724907
```

Accuracy(Lasso)

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

Out[36]:
```
0.05619445047275928
```

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([-0.0306825 ,  0.38253883,  1.72845239, -2.01509147,  0.        ,
       -0.14507463, -0.09330498, -0.        , -0.21236303,  0.09655968,
        0.43776566, -0.08927506,  0.        ,  1.14836656])
```

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

Out[39]:
```
28079065.323923618
```

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

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

Out[41]:
```
0.07137531484264747
```

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)))
```

```
38.587961140293565
1665.3084756072194
40.808191280761505
```

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]: (19397, 14)

In [46]:
```
target_vector.shape
```

Out[46]: (19397,)

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)
```

```
[28079006]
```

In [52]:
```
logr.classes_
```

Out[52]: array([28079006, 28079024, 28079099], dtype=int64)

In [53]:
```
logr.score(fs,target_vector)
```

Out[53]: 0.7360416559261741

In [54]:
```
logr.predict_proba(observation)[0][0]
```

Out[54]:
```
0.9999978255573396
```

In [55]:
```
logr.predict_proba(observation)
```

Out[55]:
```
array([[9.99997826e-01, 7.75018107e-20, 2.17444266e-06]])
```

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]:
```
0.7724092961915497
```

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

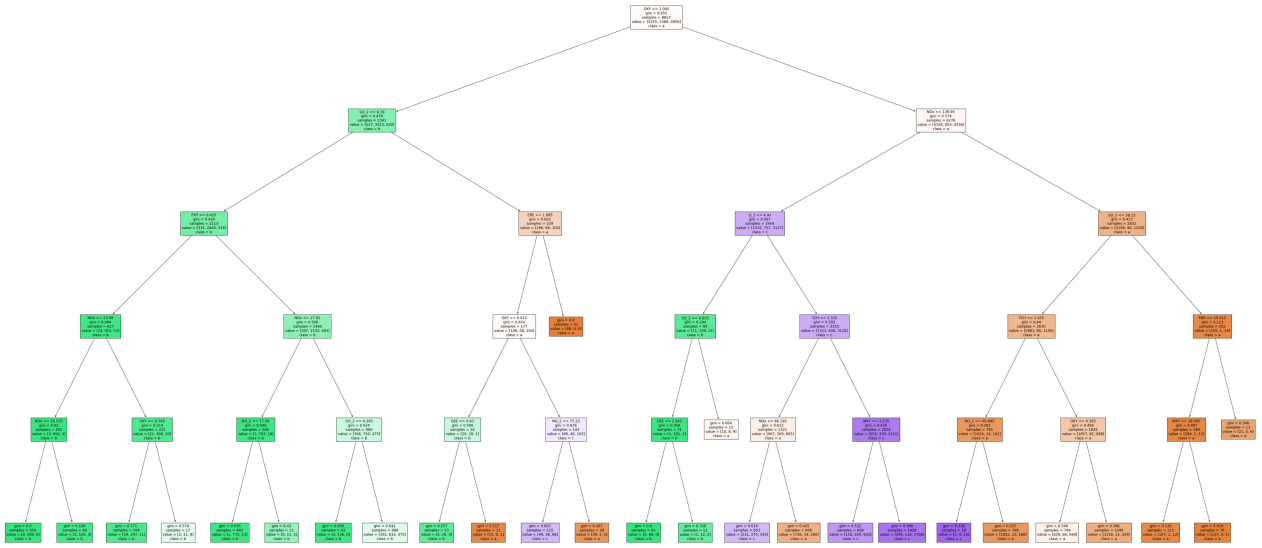
In [62]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,class_names=['a','b','c','d']
```

Out[62]:
```
[Text(2323.1020408163267, 1993.2, 'OXY <= 1.045\ngini = 0.655\nsamples = 8617\nvalue =
[5255, 3366, 4956]\nclass = a'),
 Text(1320.9795918367347, 1630.8000000000002, 'SO_2 <= 9.78\ngini = 0.479\nsamples = 234
1\nvalue = [527, 2513, 620]\nclass = b'),
```

```
 Text(728.8163265306123, 1268.4, 'PXY <= 0.425\ngini = 0.414\nsamples = 2113\nvalue = [3
31, 2445, 516]\nclass = b'),
 Text(364.40816326530614, 906.0, 'NOx <= 23.99\ngini = 0.094\nsamples = 627\nvalue = [2
4, 912, 23]\nclass = b'),
 Text(182.20408163265307, 543.5999999999999, 'NOx <= 20.535\ngini = 0.02\nsamples = 402
\nvalue = [3, 604, 3]\nclass = b'),
 Text(91.10204081632654, 181.19999999999982, 'gini = 0.0\nsamples = 334\nvalue = [0, 50
4, 0]\nclass = b'),
 Text(273.30612244897964, 181.19999999999982, 'gini = 0.108\nsamples = 68\nvalue = [3, 1
00, 3]\nclass = b'),
 Text(546.6122448979593, 543.5999999999999, 'OXY <= 0.765\ngini = 0.214\nsamples = 225\n
value = [21, 308, 20]\nclass = b'),
 Text(455.51020408163265, 181.19999999999982, 'gini = 0.171\nsamples = 208\nvalue = [19,
297, 11]\nclass = b'),
 Text(637.7142857142858, 181.19999999999982, 'gini = 0.574\nsamples = 17\nvalue = [2, 1
1, 9]\nclass = b'),
 Text(1093.2244897959185, 906.0, 'NOx <= 17.91\ngini = 0.506\nsamples = 1486\nvalue = [3
07, 1533, 493]\nclass = b'),
 Text(911.0204081632653, 543.5999999999999, 'NO_2 <= 17.09\ngini = 0.046\nsamples = 506
\nvalue = [1, 783, 18]\nclass = b'),
 Text(819.9183673469388, 181.19999999999982, 'gini = 0.035\nsamples = 495\nvalue = [1, 7
72, 13]\nclass = b'),
 Text(1002.1224489795919, 181.19999999999982, 'gini = 0.43\nsamples = 11\nvalue = [0, 1
1, 5]\nclass = b'),
 Text(1275.4285714285716, 543.5999999999999, 'SO_2 <= 4.385\ngini = 0.624\nsamples = 980
\nvalue = [306, 750, 475]\nclass = b'),
 Text(1184.326530612245, 181.19999999999982, 'gini = 0.056\nsamples = 92\nvalue = [4, 13
6, 0]\nclass = b'),
 Text(1366.530612244898, 181.19999999999982, 'gini = 0.641\nsamples = 888\nvalue = [302,
614, 475]\nclass = b'),
 Text(1913.1428571428573, 1268.4, 'EBE <= 1.885\ngini = 0.602\nsamples = 228\nvalue = [1
96, 68, 104]\nclass = a'),
 Text(1822.0408163265306, 906.0, 'OXY <= 0.615\ngini = 0.654\nsamples = 177\nvalue = [10
8, 68, 104]\nclass = a'),
 Text(1639.8367346938776, 543.5999999999999, 'EBE <= 0.63\ngini = 0.506\nsamples = 34\nv
alue = [20, 28, 1]\nclass = b'),
 Text(1548.734693877551, 181.19999999999982, 'gini = 0.257\nsamples = 23\nvalue = [5, 2
8, 0]\nclass = b'),
 Text(1730.938775510204, 181.19999999999982, 'gini = 0.117\nsamples = 11\nvalue = [15,
0, 1]\nclass = a'),
 Text(2004.2448979591838, 543.5999999999999, 'NO_2 <= 77.23\ngini = 0.626\nsamples = 143
\nvalue = [88, 40, 103]\nclass = c'),
 Text(1913.1428571428573, 181.19999999999982, 'gini = 0.607\nsamples = 115\nvalue = [49,
38, 98]\nclass = c'),
 Text(2095.3469387755104, 181.19999999999982, 'gini = 0.267\nsamples = 28\nvalue = [39,
2, 5]\nclass = a'),
 Text(2004.2448979591838, 906.0, 'gini = 0.0\nsamples = 51\nvalue = [88, 0, 0]\nclass =
a'),
 Text(3325.2244897959185, 1630.8000000000002, 'NOx <= 139.95\ngini = 0.574\nsamples = 62
76\nvalue = [4728, 853, 4336]\nclass = a'),
 Text(2687.5102040816328, 1268.4, 'O_3 <= 4.97\ngini = 0.567\nsamples = 3444\nvalue = [1
532, 757, 3127]\nclass = c'),
 Text(2459.7551020408164, 906.0, 'SO_2 <= 9.855\ngini = 0.294\nsamples = 89\nvalue = [1
1, 109, 11]\nclass = b'),
 Text(2368.65306122449, 543.5999999999999, 'EBE <= 2.945\ngini = 0.056\nsamples = 74\nva
lue = [1, 101, 2]\nclass = b'),
 Text(2277.5510204081634, 181.19999999999982, 'gini = 0.0\nsamples = 62\nvalue = [0, 89,
0]\nclass = b'),
 Text(2459.7551020408164, 181.19999999999982, 'gini = 0.338\nsamples = 12\nvalue = [1, 1
2, 2]\nclass = b'),
 Text(2550.857142857143, 543.5999999999999, 'gini = 0.664\nsamples = 15\nvalue = [10, 8,
9]\nclass = a'),
 Text(2915.265306122449, 906.0, 'TCH <= 1.335\ngini = 0.555\nsamples = 3355\nvalue = [15
21, 648, 3116]\nclass = c'),
 Text(2733.061224489796, 543.5999999999999, 'NOx <= 66.195\ngini = 0.612\nsamples = 1321
```

```
\nvalue = [967, 309, 805]\nclass = a'),
 Text(2641.9591836734694, 181.19999999999982, 'gini = 0.614\nsamples = 653\nvalue = [23
1, 275, 545]\nclass = c'),
 Text(2824.1632653061224, 181.19999999999982, 'gini = 0.425\nsamples = 668\nvalue = [73
6, 34, 260]\nclass = a'),
 Text(3097.469387755102, 543.5999999999999, 'MXY <= 3.235\ngini = 0.439\nsamples = 2034
\nvalue = [554, 339, 2311]\nclass = c'),
 Text(3006.367346938776, 181.19999999999982, 'gini = 0.522\nsamples = 606\nvalue = [118,
225, 603]\nclass = c'),
 Text(3188.571428571429, 181.19999999999982, 'gini = 0.388\nsamples = 1428\nvalue = [43
6, 114, 1708]\nclass = c'),
 Text(3962.9387755102043, 1268.4, 'SO_2 <= 38.15\ngini = 0.423\nsamples = 2832\nvalue =
[3196, 96, 1209]\nclass = a'),
 Text(3644.081632653061, 906.0, 'TCH <= 1.455\ngini = 0.44\nsamples = 2630\nvalue = [288
1, 95, 1190]\nclass = a'),
 Text(3461.877551020408, 543.5999999999999, 'NO_2 <= 60.895\ngini = 0.282\nsamples = 785
\nvalue = [1024, 14, 192]\nclass = a'),
 Text(3370.775510204082, 181.19999999999982, 'gini = 0.338\nsamples = 19\nvalue = [2, 4,
24]\nclass = c'),
 Text(3552.979591836735, 181.19999999999982, 'gini = 0.255\nsamples = 766\nvalue = [102
2, 10, 168]\nclass = a'),
 Text(3826.2857142857147, 543.5999999999999, 'OXY <= 4.365\ngini = 0.484\nsamples = 1845
\nvalue = [1857, 81, 998]\nclass = a'),
 Text(3735.183673469388, 181.19999999999982, 'gini = 0.546\nsamples = 799\nvalue = [639,
66, 569]\nclass = a'),
 Text(3917.387755102041, 181.19999999999982, 'gini = 0.396\nsamples = 1046\nvalue = [121
8, 15, 429]\nclass = a'),
 Text(4281.795918367347, 906.0, 'EBE <= 19.015\ngini = 0.113\nsamples = 202\nvalue = [31
5, 1, 19]\nclass = a'),
 Text(4190.693877551021, 543.5999999999999, 'MXY <= 18.085\ngini = 0.087\nsamples = 189
\nvalue = [294, 1, 13]\nclass = a'),
 Text(4099.591836734694, 181.19999999999982, 'gini = 0.135\nsamples = 113\nvalue = [167,
1, 12]\nclass = a'),
 Text(4281.795918367347, 181.19999999999982, 'gini = 0.016\nsamples = 76\nvalue = [127,
0, 1]\nclass = a'),
 Text(4372.897959183674, 543.5999999999999, 'gini = 0.346\nsamples = 13\nvalue = [21, 0,
6]\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: 0.10767927161857482
Ridge Regression: 0.10680483588399026
Lasso Regression 0.05619445047275928
ElasticNet Regression: 0.07137531484264747
Logistic Regression: 0.7360416559261741
Random Forest: 0.7724092961915497
```

Random Forest is suitable for this dataset