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_2003.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PX
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209999	Na
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389999	Na
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240002	Na
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839996	Na
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779999	Na
...	...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380000	1.2
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400000	0.5
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830000	Na

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PX
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570000	Na
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350000	2.4

243984 rows × 16 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', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

Out[6]:

	CO	station
5	1.94	28079006

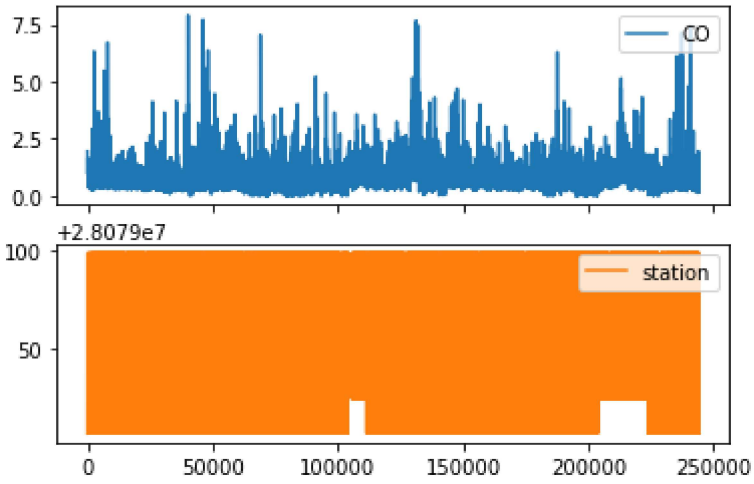
	CO	station
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

Line chart

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

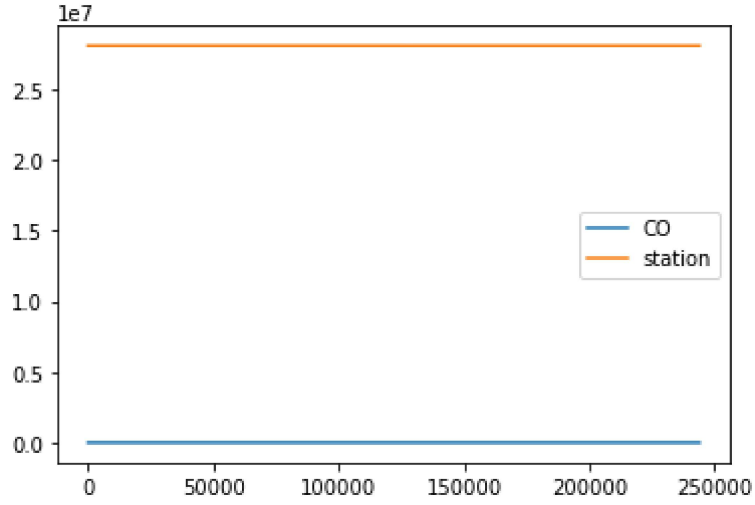
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

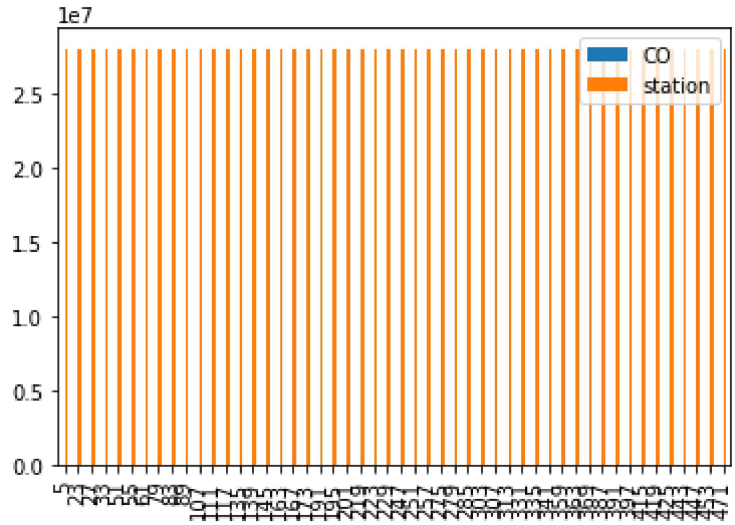


Bar chart

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

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

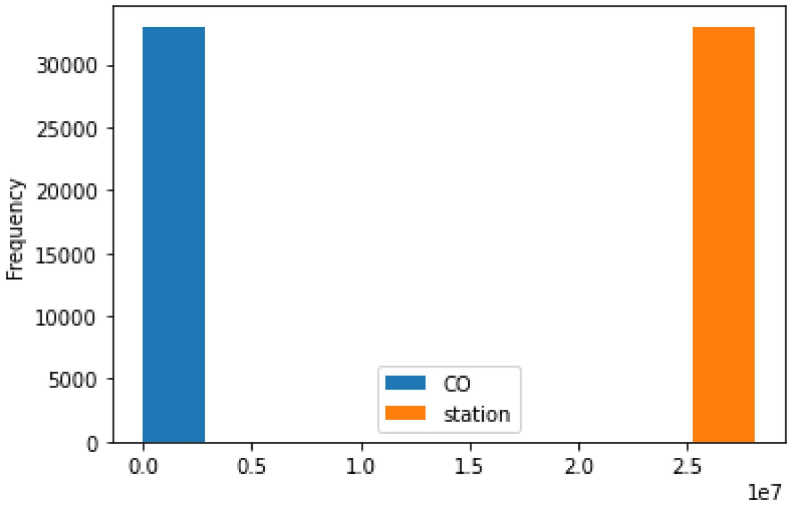
Out[10]: <AxesSubplot:>



Histogram

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

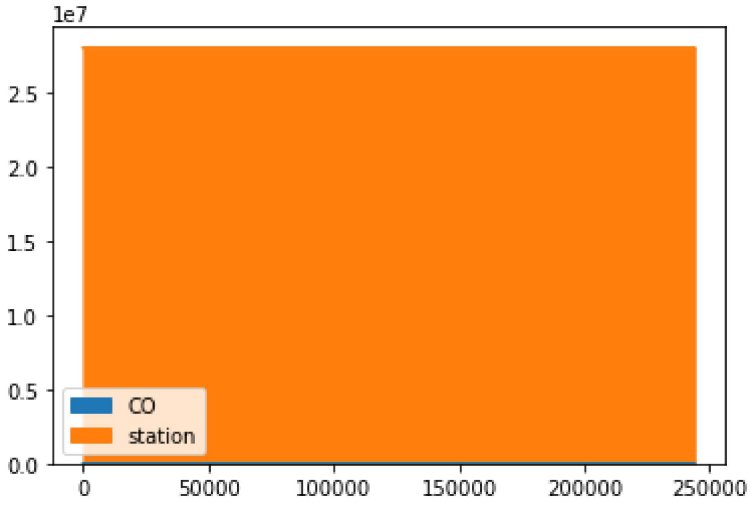
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

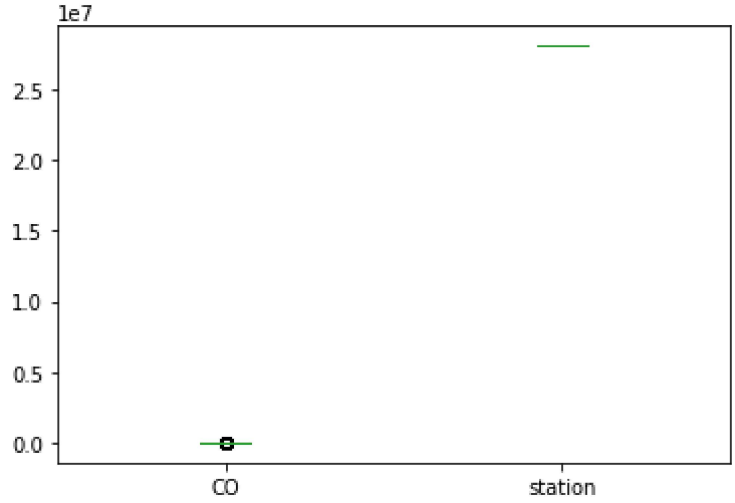
Out[12]: <AxesSubplot:>



Box chart

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

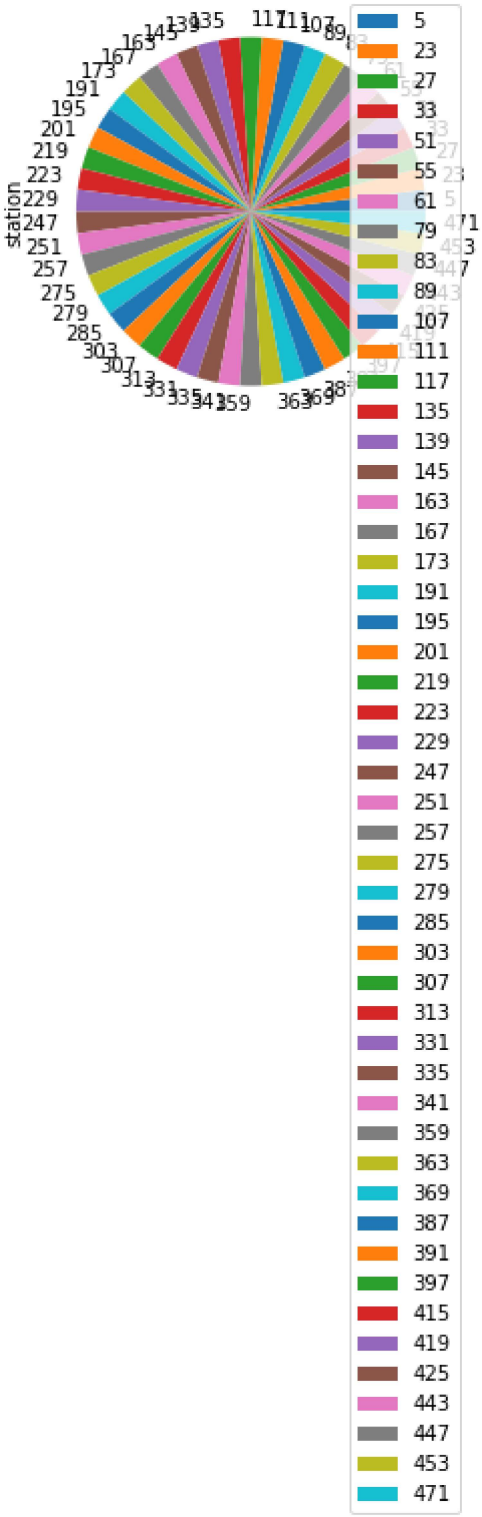
Out[13]: <AxesSubplot:>



Pie chart

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

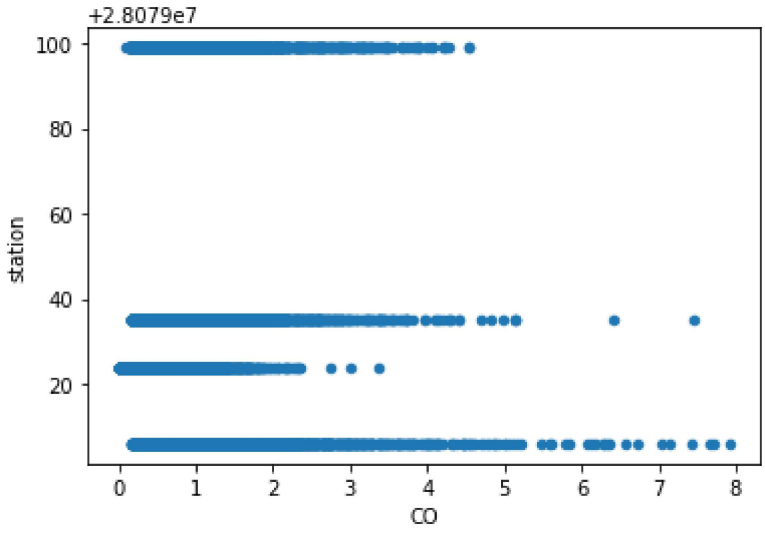
Out[14]: <AxesSubplot:ylabel='station'>



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: 33010 entries, 5 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     33010 non-null  object 
 1   BEN      33010 non-null  float64
 2   CO       33010 non-null  float64
 3   EBE      33010 non-null  float64
 4   MXY      33010 non-null  float64
 5   NMHC     33010 non-null  float64
 6   NO_2     33010 non-null  float64
 7   NOx      33010 non-null  float64
 8   OXY      33010 non-null  float64
 9   O_3      33010 non-null  float64
 10  PM10     33010 non-null  float64
 11  PXY      33010 non-null  float64
 12  SO_2     33010 non-null  float64
 13  TCH      33010 non-null  float64
 14  TOL      33010 non-null  float64
 15  station  33010 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.3+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000	33010.000000
mean	2.192633	0.759868	2.639726	5.838414	0.137177	57.328049	120.15367
std	2.064160	0.545999	2.825194	6.267296	0.127863	31.811082	104.52170
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.900000	0.430000	1.010000	1.880000	0.060000	34.529999	49.07000
50%	1.610000	0.620000	1.890000	4.070000	0.110000	55.105000	92.77999
75%	2.810000	0.930000	3.300000	7.530000	0.170000	76.160004	160.10000

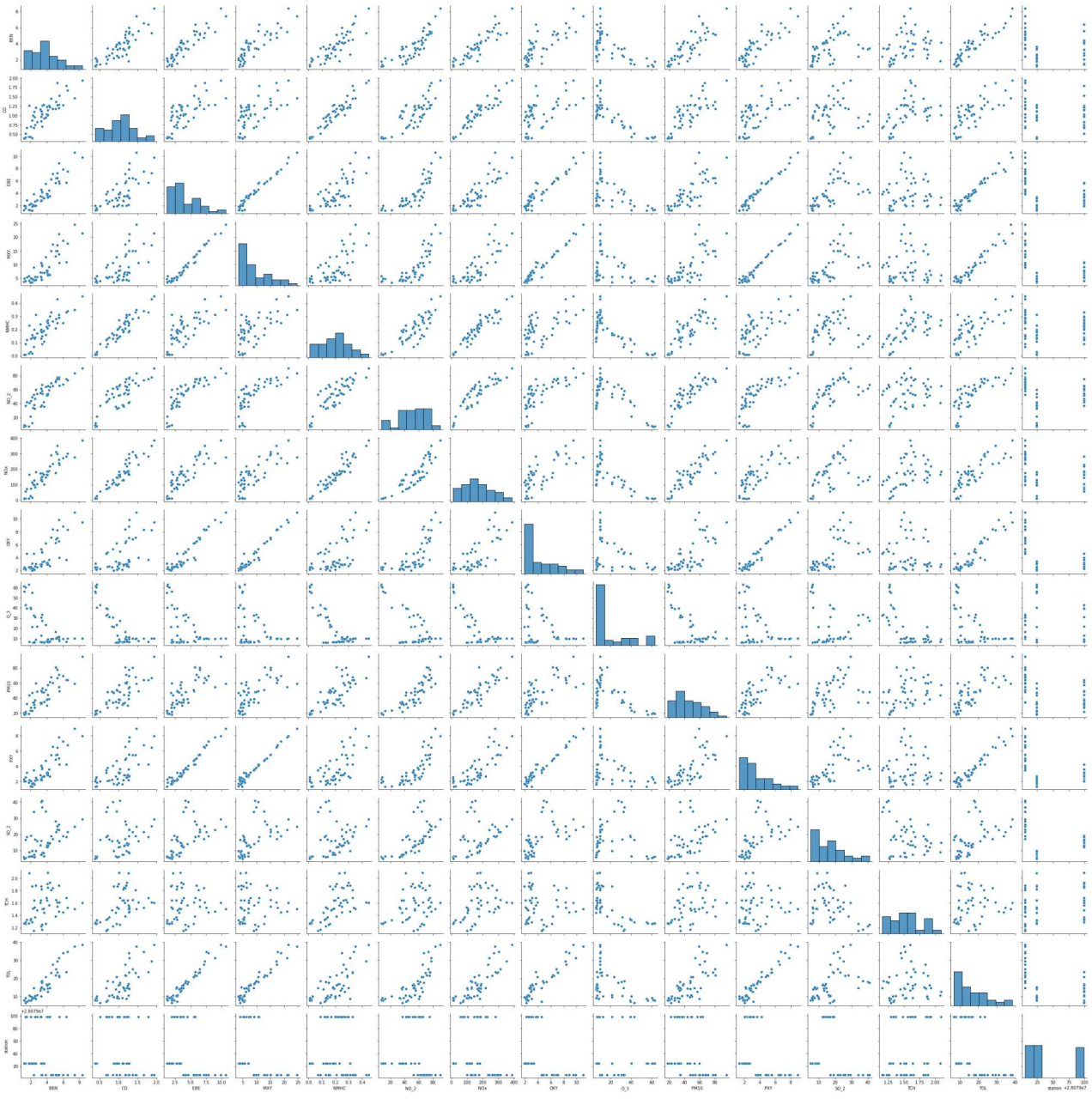
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
max	66.389999	7.920000	92.589996	177.600006	2.180000	342.700012	1246.00000

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])
```

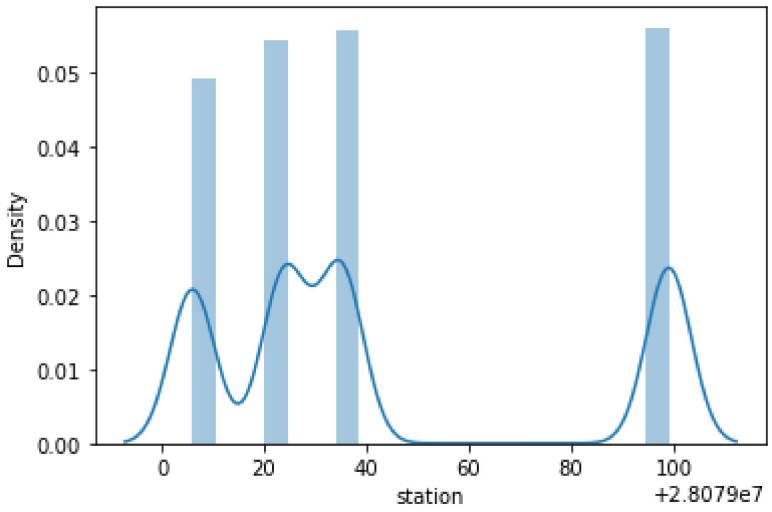
Out[19]: <seaborn.axisgrid.PairGrid at 0x19fe9c5f8b0>



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

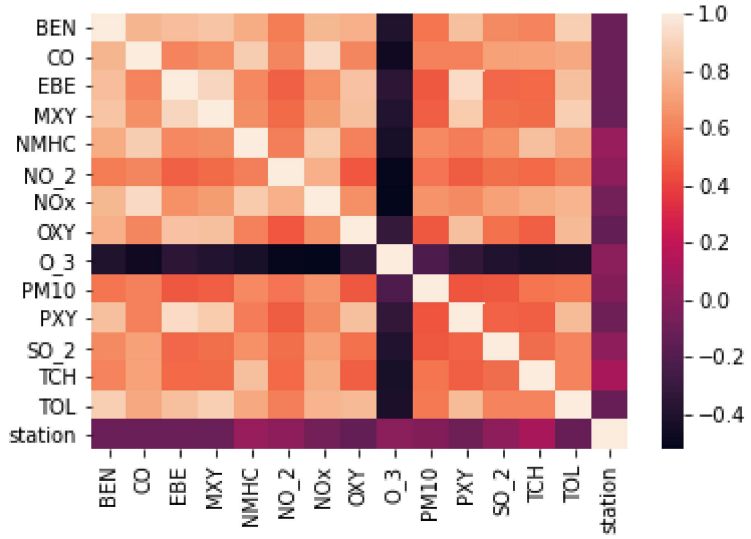
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28079002.996943984
```

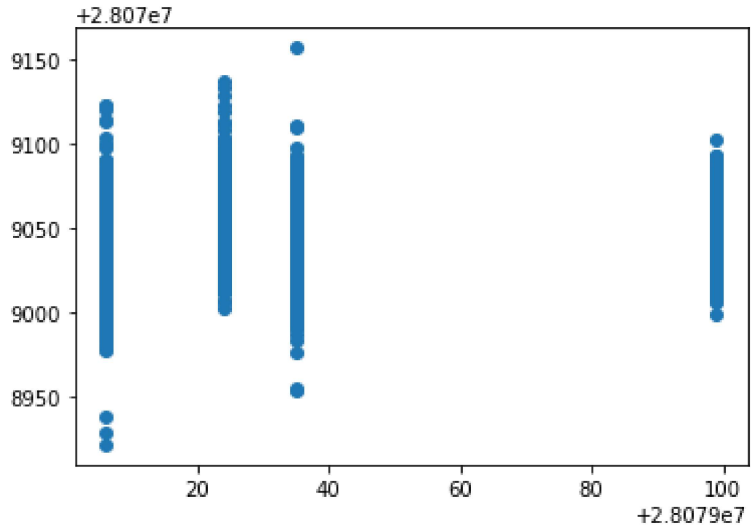
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	1.276738
CO	-38.419943
EBE	-1.652659
MXY	0.120003
NMHC	155.388222
NO_2	0.155343
NOx	-0.067647
OXY	-1.316740
O_3	-0.013520
PM10	-0.057980
PXY	1.662017
SO_2	0.895247
TCH	34.063839
TOL	-0.836322

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x19ff9941790>
```



ACCURACY

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

Out[28]: 0.16958450932868763

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

Out[29]: 0.1787910184192012

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]: 0.16875553306978774

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

Out[33]: 0.17770868147324226

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]:
```
0.03709957832024513
```

Accuracy(Lasso)

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

Out[36]:
```
0.03394912089509239
```

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.        , -0.32574759,  0.02119408, -0.02987462,  0.13340555,
        0.15031014, -0.0702183 , -1.30891183, -0.0421444 ,  0.070547  ,
        0.23795802,  0.78604522,  1.58155984, -0.41258578])
```

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

Out[39]:
```
28079037.5312244
```

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

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

Out[41]:
```
0.04461573097242899
```

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)))
```

```
29.276741085597166
1191.7744062400225
34.52208577476197
```

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]: (33010, 14)

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

Out[46]: (33010,)

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)
```

```
[28079035]
```

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

Out[52]: array([28079006, 28079024, 28079035, 28079099], dtype=int64)

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

Out[53]: 0.7584974250227204

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

Out[54]:
```
2.3306153265290618e-23
```

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

Out[55]:
```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457491e-16]])
```

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]:
```
0.7247150023086772
```

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

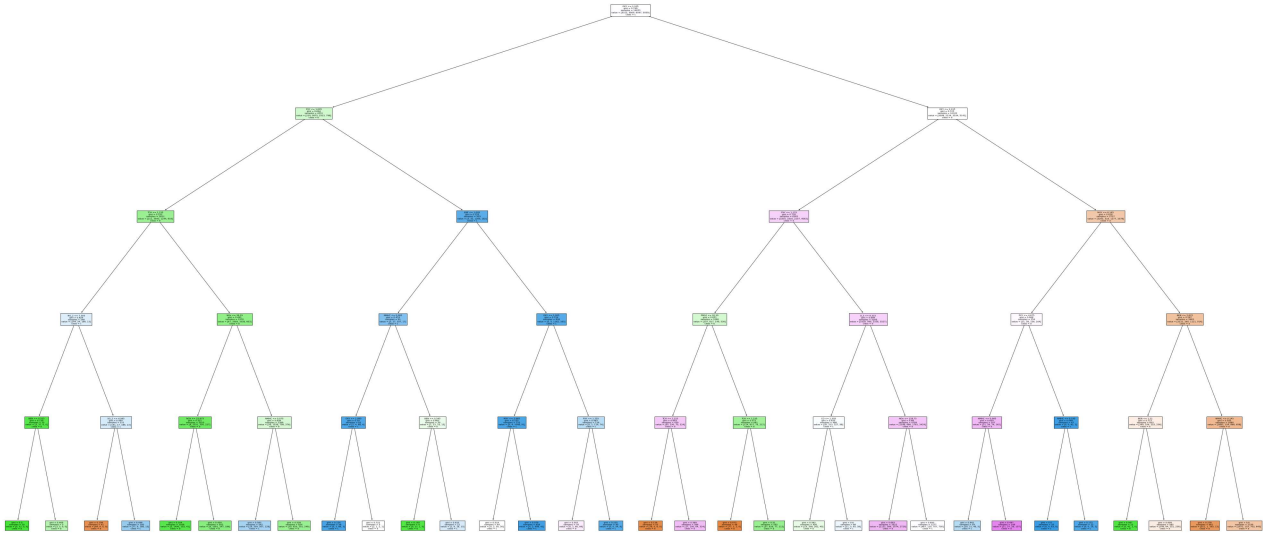
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2232.0, 1993.2, 'OXY <= 1.105\ngini = 0.749\nsamples = 14653\nvalue = [5312, 5809,
6047, 5939]\nclass = c'),
 Text(1116.0, 1630.8000000000002, 'PXY <= 1.005\ngini = 0.604\nsamples = 4551\nvalue =
[214, 3675, 2513, 798]\nclass = b'),
```

```
 Text(558.0, 1268.4, 'TCH <= 1.235\ngini = 0.533\nsamples = 3622\nvalue = [211, 3643, 12
44, 616]\nclass = b'),
 Text(279.0, 906.0, 'NO_2 <= 7.305\ngini = 0.629\nsamples = 246\nvalue = [144, 54, 186,
13]\nclass = c'),
 Text(139.5, 543.5999999999999, 'BEN <= 0.525\ngini = 0.139\nsamples = 25\nvalue = [3, 3
7, 0, 0]\nclass = b'),
 Text(69.75, 181.19999999999982, 'gini = 0.0\nsamples = 20\nvalue = [0, 32, 0, 0]\nclass
= b'),
 Text(209.25, 181.19999999999982, 'gini = 0.469\nsamples = 5\nvalue = [3, 5, 0, 0]\nclas
s = b'),
 Text(418.5, 543.5999999999999, 'SO_2 <= 4.995\ngini = 0.569\nsamples = 221\nvalue = [14
1, 17, 186, 13]\nclass = c'),
 Text(348.75, 181.19999999999982, 'gini = 0.198\nsamples = 41\nvalue = [64, 8, 0, 0]\ncl
ass = a'),
 Text(488.25, 181.19999999999982, 'gini = 0.498\nsamples = 180\nvalue = [77, 9, 186, 13]
\nclass = c'),
 Text(837.0, 906.0, 'NOx <= 39.29\ngini = 0.492\nsamples = 3376\nvalue = [67, 3589, 105
8, 603]\nclass = b'),
 Text(697.5, 543.5999999999999, 'NOx <= 22.675\ngini = 0.321\nsamples = 2010\nvalue =
[8, 2551, 350, 227]\nclass = b'),
 Text(627.75, 181.19999999999982, 'gini = 0.164\nsamples = 1058\nvalue = [0, 1490, 103,
41]\nclass = b'),
 Text(767.25, 181.19999999999982, 'gini = 0.459\nsamples = 952\nvalue = [8, 1061, 247, 1
86]\nclass = b'),
 Text(976.5, 543.5999999999999, 'NMHC <= 0.075\ngini = 0.638\nsamples = 1366\nvalue = [5
9, 1038, 708, 376]\nclass = b'),
 Text(906.75, 181.19999999999982, 'gini = 0.583\nsamples = 542\nvalue = [29, 209, 507, 1
28]\nclass = c'),
 Text(1046.25, 181.19999999999982, 'gini = 0.538\nsamples = 824\nvalue = [30, 829, 201,
248]\nclass = b'),
 Text(1674.0, 1268.4, 'EBE <= 1.005\ngini = 0.255\nsamples = 929\nvalue = [3, 32, 1269,
182]\nclass = c'),
 Text(1395.0, 906.0, 'NMHC <= 0.085\ngini = 0.474\nsamples = 91\nvalue = [3, 27, 107, 1
7]\nclass = c'),
 Text(1255.5, 543.5999999999999, 'OXY <= 1.065\ngini = 0.19\nsamples = 61\nvalue = [2,
4, 88, 4]\nclass = c'),
 Text(1185.75, 181.19999999999982, 'gini = 0.124\nsamples = 56\nvalue = [0, 3, 86, 3]\nc
lass = c'),
 Text(1325.25, 181.19999999999982, 'gini = 0.722\nsamples = 5\nvalue = [2, 1, 2, 1]\ncla
ss = a'),
 Text(1534.5, 543.5999999999999, 'BEN <= 0.545\ngini = 0.662\nsamples = 30\nvalue = [1,
23, 19, 13]\nclass = b'),
 Text(1464.75, 181.19999999999982, 'gini = 0.105\nsamples = 9\nvalue = [0, 17, 1, 0]\ncl
ass = b'),
 Text(1604.25, 181.19999999999982, 'gini = 0.633\nsamples = 21\nvalue = [1, 6, 18, 13]\n
class = c'),
 Text(1953.0, 906.0, 'OXY <= 1.005\ngini = 0.224\nsamples = 838\nvalue = [0, 5, 1162, 16
5]\nclass = c'),
 Text(1813.5, 543.5999999999999, 'MXY <= 2.685\ngini = 0.153\nsamples = 722\nvalue = [0,
4, 1048, 91]\nclass = c'),
 Text(1743.75, 181.19999999999982, 'gini = 0.519\nsamples = 64\nvalue = [0, 2, 50, 50]\n
class = c'),
 Text(1883.25, 181.19999999999982, 'gini = 0.079\nsamples = 658\nvalue = [0, 2, 998, 41]
\nclass = c'),
 Text(2092.5, 543.5999999999999, 'PXY <= 1.755\ngini = 0.483\nsamples = 116\nvalue = [0,
1, 114, 74]\nclass = c'),
 Text(2022.75, 181.19999999999982, 'gini = 0.507\nsamples = 76\nvalue = [0, 1, 60, 66]\n
class = d'),
 Text(2162.25, 181.19999999999982, 'gini = 0.225\nsamples = 40\nvalue = [0, 0, 54, 8]\nc
lass = c'),
 Text(3348.0, 1630.8000000000002, 'OXY <= 3.315\ngini = 0.725\nsamples = 10102\nvalue =
[5098, 2134, 3534, 5141]\nclass = d'),
 Text(2790.0, 1268.4, 'PXY <= 1.125\ngini = 0.716\nsamples = 6365\nvalue = [1835, 1816,
2257, 4063]\nclass = d'),
 Text(2511.0, 906.0, 'PM10 <= 20.39\ngini = 0.651\nsamples = 1080\nvalue = [207, 817, 14
```

```
9, 536]\nclass = b'),
 Text(2371.5, 543.5999999999999, 'TCH <= 1.225\ngini = 0.663\nsamples = 439\nvalue = [9
3, 190, 70, 324]\nclass = d'),
 Text(2301.75, 181.19999999999982, 'gini = 0.136\nsamples = 53\nvalue = [76, 0, 6, 0]\nc
lass = a'),
 Text(2441.25, 181.19999999999982, 'gini = 0.589\nsamples = 386\nvalue = [17, 190, 64, 3
24]\nclass = d'),
 Text(2650.5, 543.5999999999999, 'TCH <= 1.235\ngini = 0.571\nsamples = 641\nvalue = [11
4, 627, 79, 212]\nclass = b'),
 Text(2580.75, 181.19999999999982, 'gini = 0.073\nsamples = 48\nvalue = [77, 1, 2, 0]\nc
lass = a'),
 Text(2720.25, 181.19999999999982, 'gini = 0.51\nsamples = 593\nvalue = [37, 626, 77, 21
2]\nclass = b'),
 Text(3069.0, 906.0, 'O_3 <= 6.315\ngini = 0.699\nsamples = 5285\nvalue = [1628, 999, 21
08, 3527]\nclass = d'),
 Text(2929.5, 543.5999999999999, 'CO <= 1.305\ngini = 0.635\nsamples = 486\nvalue = [30,
313, 327, 98]\nclass = c'),
 Text(2859.75, 181.19999999999982, 'gini = 0.582\nsamples = 400\nvalue = [19, 308, 262,
40]\nclass = b'),
 Text(2999.25, 181.19999999999982, 'gini = 0.6\nsamples = 86\nvalue = [11, 5, 65, 58]\nc
lass = c'),
 Text(3208.5, 543.5999999999999, 'NOx <= 128.75\ngini = 0.68\nsamples = 4799\nvalue = [1
598, 686, 1781, 3429]\nclass = d'),
 Text(3138.75, 181.19999999999982, 'gini = 0.663\nsamples = 3522\nvalue = [1131, 566, 10
74, 2729]\nclass = d'),
 Text(3278.25, 181.19999999999982, 'gini = 0.693\nsamples = 1277\nvalue = [467, 120, 70
7, 700]\nclass = c'),
 Text(3906.0, 1268.4, 'MXY <= 6.165\ngini = 0.616\nsamples = 3737\nvalue = [3263, 318, 1
277, 1078]\nclass = a'),
 Text(3627.0, 906.0, 'PXY <= 4.675\ngini = 0.668\nsamples = 254\nvalue = [31, 58, 156, 1
64]\nclass = d'),
 Text(3487.5, 543.5999999999999, 'NMHC <= 0.085\ngini = 0.659\nsamples = 207\nvalue = [3
1, 58, 74, 162]\nclass = d'),
 Text(3417.75, 181.19999999999982, 'gini = 0.641\nsamples = 59\nvalue = [20, 21, 48, 5]
\nclass = c'),
 Text(3557.25, 181.19999999999982, 'gini = 0.497\nsamples = 148\nvalue = [11, 37, 26, 15
7]\nclass = d'),
 Text(3766.5, 543.5999999999999, 'NMHC <= 0.195\ngini = 0.046\nsamples = 47\nvalue = [0,
0, 82, 2]\nclass = c'),
 Text(3696.75, 181.19999999999982, 'gini = 0.0\nsamples = 35\nvalue = [0, 0, 63, 0]\ncla
ss = c'),
 Text(3836.25, 181.19999999999982, 'gini = 0.172\nsamples = 12\nvalue = [0, 0, 19, 2]\nc
lass = c'),
 Text(4185.0, 906.0, 'BEN <= 2.625\ngini = 0.587\nsamples = 3483\nvalue = [3232, 260, 11
21, 914]\nclass = a'),
 Text(4045.5, 543.5999999999999, 'BEN <= 1.21\ngini = 0.717\nsamples = 603\nvalue = [34
9, 106, 253, 256]\nclass = a'),
 Text(3975.75, 181.19999999999982, 'gini = 0.087\nsamples = 23\nvalue = [0, 42, 2, 0]\nc
lass = b'),
 Text(4115.25, 181.19999999999982, 'gini = 0.699\nsamples = 580\nvalue = [349, 64, 251,
256]\nclass = a'),
 Text(4324.5, 543.5999999999999, 'NMHC <= 0.145\ngini = 0.543\nsamples = 2880\nvalue =
[2883, 154, 868, 658]\nclass = a'),
 Text(4254.75, 181.19999999999982, 'gini = 0.278\nsamples = 681\nvalue = [911, 1, 165, 1
3]\nclass = a'),
 Text(4394.25, 181.19999999999982, 'gini = 0.6\nsamples = 2199\nvalue = [1972, 153, 703,
645]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.16958450932868763

Ridge Regression:0.16875553306978774

Lasso Regression:0.033949120895092 39

ElasticNet Regression:0.044615730972428 99

Logistic Regression:0.7584974250227204

Random Forest:0.7247150023086772

Logistic Regression is suitable for this dataset