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_2005.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91	10.65
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.93	NaN
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60	NaN
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16	NaN
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00	NaN
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95	1.49
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31	2.93

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00	NaN
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67	2.11

237000 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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

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

Out[6]:

	CO	station
5	0.88	28079006

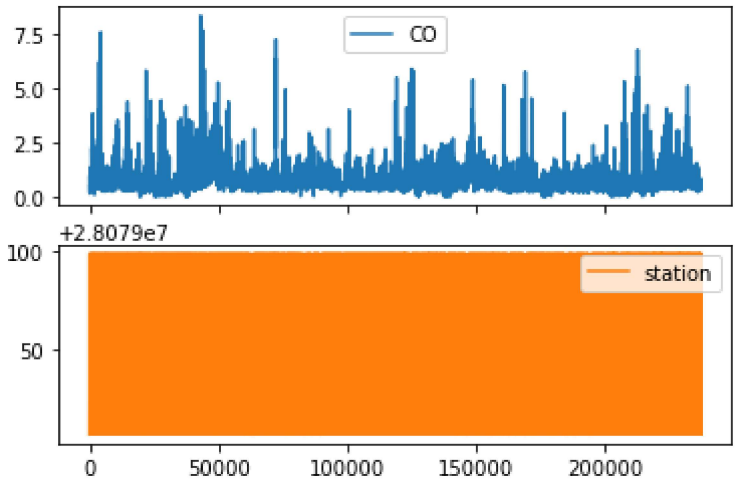
	CO	station
22	0.22	28079024
25	0.49	28079099
31	0.84	28079006
48	0.20	28079024
...	...	...
236970	0.39	28079024
236973	0.45	28079099
236979	0.38	28079006
236996	0.54	28079024
236999	0.40	28079099

20070 rows × 2 columns

Line chart

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

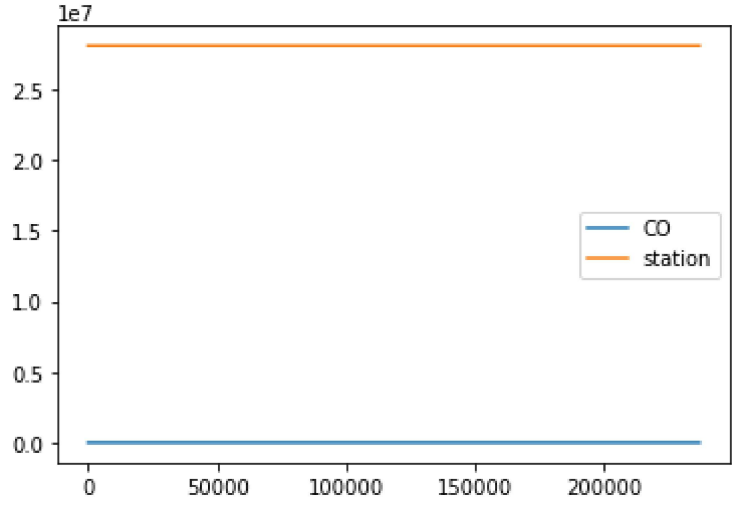
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

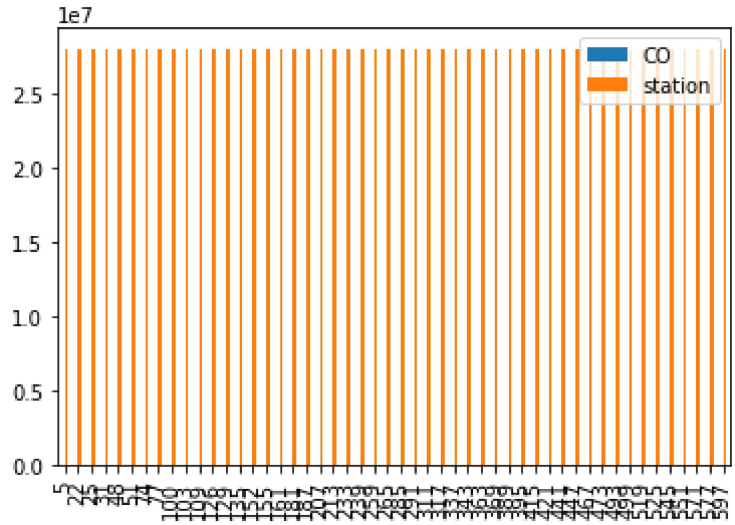


Bar chart

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

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

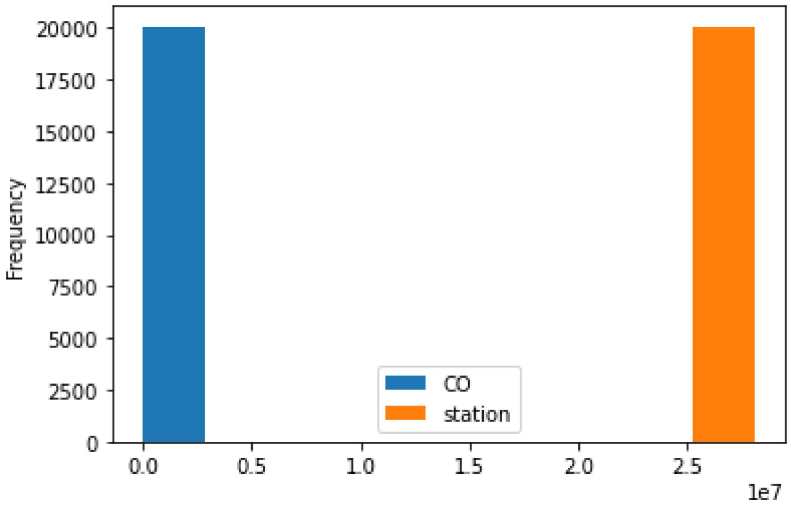
Out[10]: <AxesSubplot:>



Histogram

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

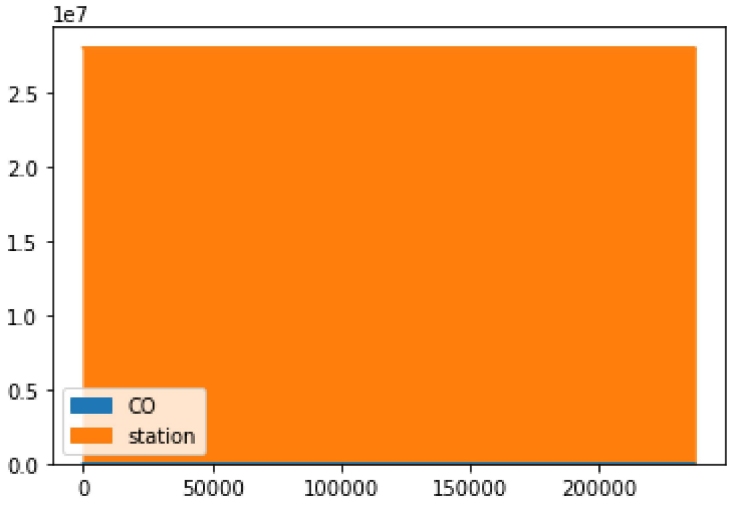
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

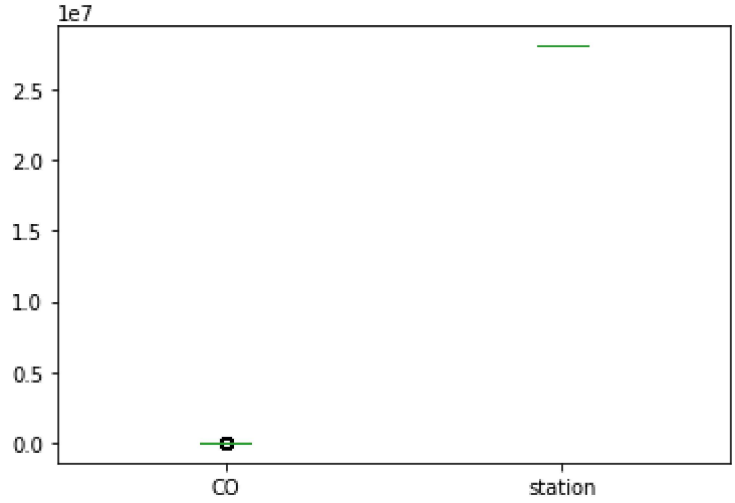
Out[12]: <AxesSubplot:>



Box chart

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

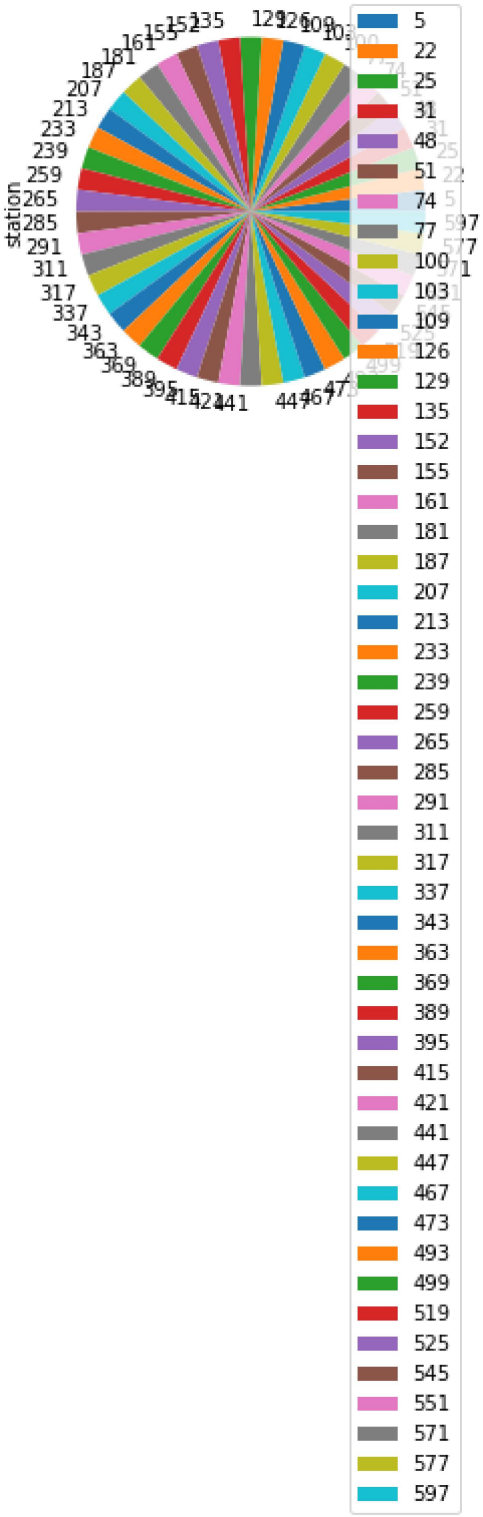
Out[13]: <AxesSubplot:>



Pie chart

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

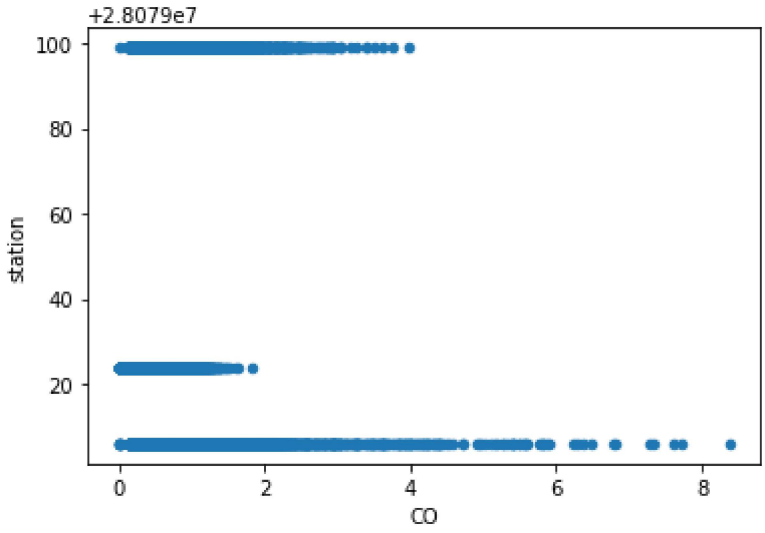
Out[14]: <AxesSubplot:ylabel='station'>



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: 20070 entries, 5 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     20070 non-null  object 
 1   BEN      20070 non-null  float64
 2   CO       20070 non-null  float64
 3   EBE      20070 non-null  float64
 4   MXY      20070 non-null  float64
 5   NMHC     20070 non-null  float64
 6   NO_2     20070 non-null  float64
 7   NOx      20070 non-null  float64
 8   OXY      20070 non-null  float64
 9   O_3      20070 non-null  float64
 10  PM10     20070 non-null  float64
 11  PM25     20070 non-null  float64
 12  PXY      20070 non-null  float64
 13  SO_2     20070 non-null  float64
 14  TCH      20070 non-null  float64
 15  TOL      20070 non-null  float64
 16  station  20070 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 2.8+ MB
```

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

Out[17]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NO
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.00000
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	143.04653
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	136.58252
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499	56.10249
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	105.69999

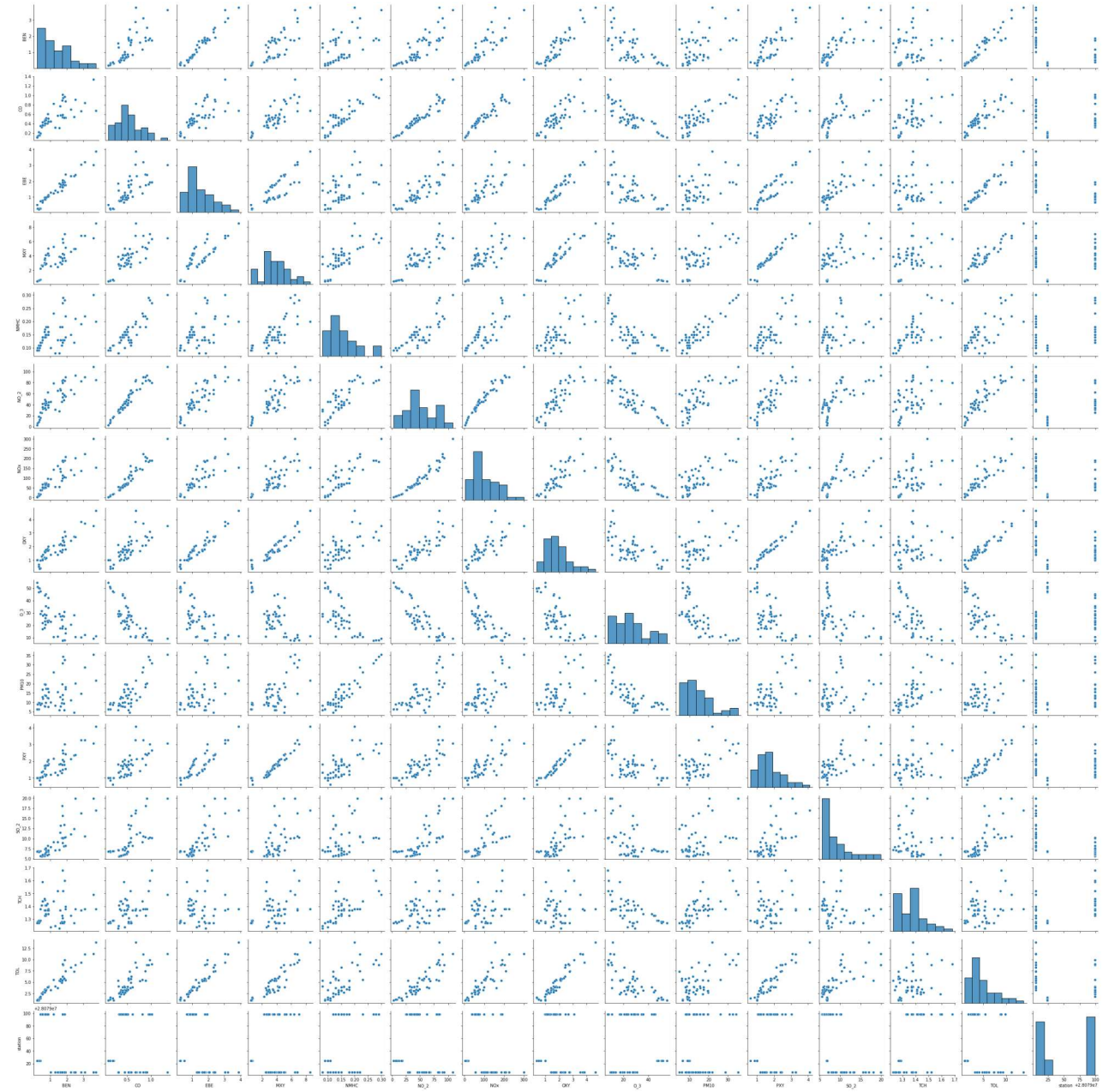
	BEN	CO	EBE	MXY	NMHC	NO_2	NO
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	190.10000
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	1774.00000

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])
```

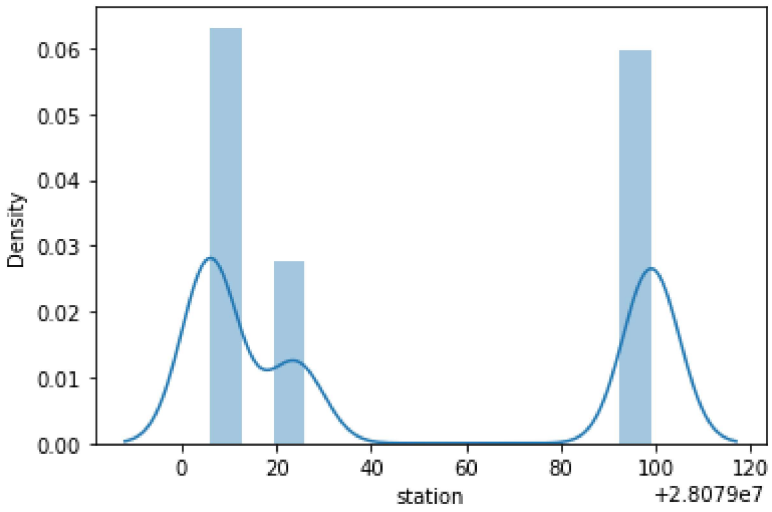
Out[19]: <seaborn.axisgrid.PairGrid at 0x28921f01730>



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

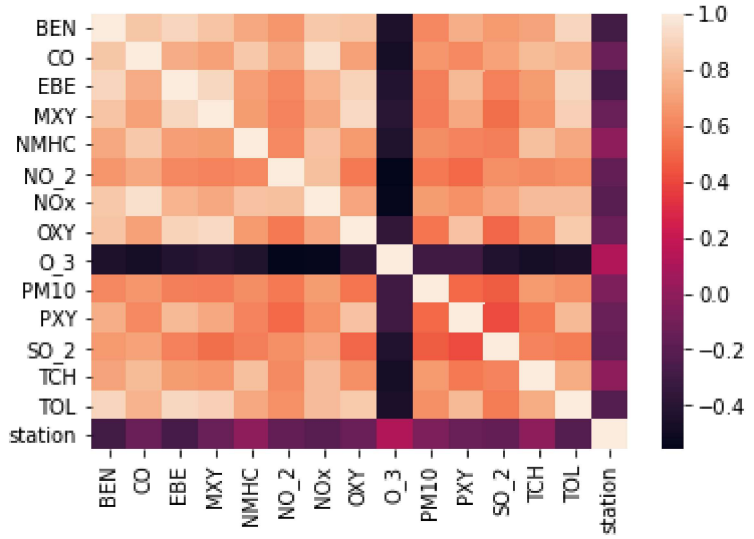
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28078961.660010003
```

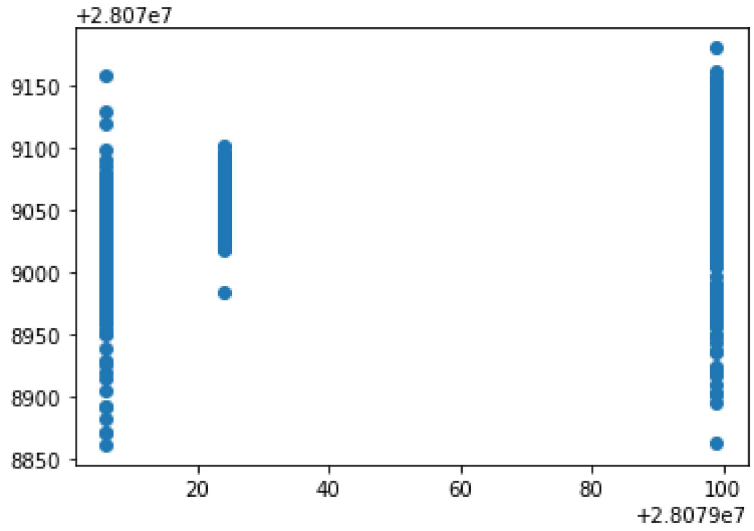
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-9.897882
CO	37.778584
EBE	-13.325203
MXY	3.644205
NMHC	80.521440
NO_2	0.115536
NOx	-0.260250
OXY	3.393619
O_3	0.011446
PM10	0.037372
PXY	2.801847
SO_2	0.219537
TCH	61.121715
TOL	-0.605742

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x28930869700>
```



ACCURACY

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

Out[28]: 0.3036475687503324

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

Out[29]: 0.304191474067494

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]: 0.30313654420191183

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

Out[33]: 0.30395217205028113

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]:
```
0.06639646707548374
```

Accuracy(Lasso)

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

Out[36]:
```
0.059641260552344866
```

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([-5.70028500e+00,  1.41914170e+00, -7.37448881e+00,  2.57581706e+00,
        8.86991954e-01, -5.77923438e-02, -6.86244551e-03,  1.91964269e+00,
       -2.17554242e-02,  2.33471133e-01,  1.51130702e+00,  1.37817392e-01,
        1.52831078e+00, -8.13429571e-01])
```

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

Out[39]:
```
28079050.67954227
```

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

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

Out[41]:
```
0.17395085484374662
```

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)))
```

```
36.864103421434606
1539.017310135056
39.230311114431096
```

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]: (20070, 14)

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

Out[46]: (20070,)

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]: 0.879023418036871

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

Out[54]:
```
0.9998967601812779
```

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

Out[55]:
```
array([[9.99896760e-01, 3.21124597e-30, 1.03239819e-04]])
```

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]:
```
0.8646161305620181
```

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

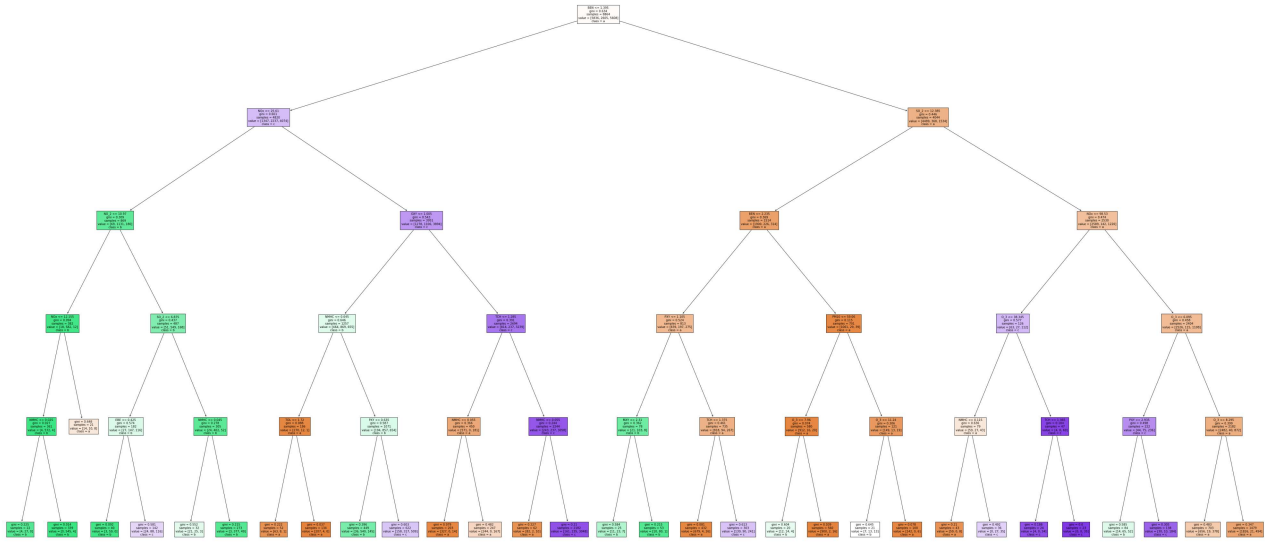
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2111.1000000000004, 1993.2, 'BEN <= 1.395\ngini = 0.634\nsamples = 8864\nvalue =
[5836, 2605, 5608]\nclass = a'),
 Text(948.6, 1630.8000000000002, 'NOx <= 25.61\ngini = 0.601\nsamples = 4820\nvalue = [1
347, 2237, 4074]\nclass = c'),
```

```
 Text(409.20000000000005, 1268.4, 'NO_2 <= 10.97\ngini = 0.309\nsamples = 869\nvalue =
[69, 1131, 180]\nclass = b'),
 Text(223.20000000000002, 906.0, 'NOx <= 12.155\ngini = 0.094\nsamples = 382\nvalue = [1
8, 582, 12]\nclass = b'),
 Text(148.8, 543.5999999999999, 'NMHC <= 0.025\ngini = 0.027\nsamples = 361\nvalue = [4,
572, 4]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.225\nsamples = 22\nvalue = [4, 27, 0]\nclass =
b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.014\nsamples = 339\nvalue = [0,
545, 4]\nclass = b'),
 Text(297.6, 543.5999999999999, 'gini = 0.648\nsamples = 21\nvalue = [14, 10, 8]\nclass
= a'),
 Text(595.2, 906.0, 'SO_2 <= 6.875\ngini = 0.437\nsamples = 487\nvalue = [51, 549, 168]
\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'EBE <= 0.425\ngini = 0.574\nsamples = 182
\nvalue = [27, 147, 116]\nclass = b'),
 Text(372.0, 181.19999999999982, 'gini = 0.092\nsamples = 40\nvalue = [3, 59, 0]\nclass
= b'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.581\nsamples = 142\nvalue = [24,
88, 116]\nclass = c'),
 Text(744.0, 543.5999999999999, 'NMHC <= 0.045\ngini = 0.278\nsamples = 305\nvalue = [2
4, 402, 52]\nclass = b'),
 Text(669.6, 181.19999999999982, 'gini = 0.552\nsamples = 32\nvalue = [21, 25, 3]\nclass
= b'),
 Text(818.4000000000001, 181.19999999999982, 'gini = 0.215\nsamples = 273\nvalue = [3, 3
77, 49]\nclass = b'),
 Text(1488.0, 1268.4, 'OXY <= 1.005\ngini = 0.543\nsamples = 3951\nvalue = [1278, 1106,
3894]\nclass = c'),
 Text(1190.4, 906.0, 'NMHC <= 0.045\ngini = 0.646\nsamples = 1257\nvalue = [464, 869, 65
5]\nclass = b'),
 Text(1041.6000000000001, 543.5999999999999, 'TOL <= 1.72\ngini = 0.088\nsamples = 186\n
value = [270, 12, 1]\nclass = a'),
 Text(967.2, 181.19999999999982, 'gini = 0.222\nsamples = 52\nvalue = [63, 8, 1]\nclass
= a'),
 Text(1116.0, 181.19999999999982, 'gini = 0.037\nsamples = 134\nvalue = [207, 4, 0]\ncla
ss = a'),
 Text(1339.2, 543.5999999999999, 'PXY <= 0.635\ngini = 0.587\nsamples = 1071\nvalue = [1
94, 857, 654]\nclass = b'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.396\nsamples = 449\nvalue = [36,
540, 145]\nclass = b'),
 Text(1413.6000000000001, 181.19999999999982, 'gini = 0.603\nsamples = 622\nvalue = [15
8, 317, 509]\nclass = c'),
 Text(1785.6000000000001, 906.0, 'TCH <= 1.285\ngini = 0.391\nsamples = 2694\nvalue = [8
14, 237, 3239]\nclass = c'),
 Text(1636.8000000000002, 543.5999999999999, 'NMHC <= 0.055\ngini = 0.366\nsamples = 450
\nvalue = [571, 0, 181]\nclass = a'),
 Text(1562.4, 181.19999999999982, 'gini = 0.079\nsamples = 203\nvalue = [327, 0, 14]\ncl
ass = a'),
 Text(1711.2, 181.19999999999982, 'gini = 0.482\nsamples = 247\nvalue = [244, 0, 167]\nc
lass = a'),
 Text(1934.4, 543.5999999999999, 'NMHC <= 0.055\ngini = 0.244\nsamples = 2244\nvalue =
[243, 237, 3058]\nclass = c'),
 Text(1860.0000000000002, 181.19999999999982, 'gini = 0.227\nsamples = 62\nvalue = [82,
2, 10]\nclass = a'),
 Text(2008.8000000000002, 181.19999999999982, 'gini = 0.21\nsamples = 2182\nvalue = [16
1, 235, 3048]\nclass = c'),
 Text(3273.6000000000004, 1630.8000000000002, 'SO_2 <= 12.385\ngini = 0.446\nsamples = 4
044\nvalue = [4489, 368, 1534]\nclass = a'),
 Text(2678.4, 1268.4, 'BEN <= 2.235\ngini = 0.369\nsamples = 1514\nvalue = [1900, 226, 3
14]\nclass = a'),
 Text(2380.8, 906.0, 'PXY <= 1.105\ngini = 0.524\nsamples = 813\nvalue = [839, 197, 275]
\nclass = a'),
 Text(2232.0, 543.5999999999999, 'MXY <= 2.31\ngini = 0.362\nsamples = 78\nvalue = [21,
103, 8]\nclass = b'),
 Text(2157.6000000000004, 181.19999999999982, 'gini = 0.584\nsamples = 25\nvalue = [11,
```

```
23, 7]\nclass = b'),
 Text(2306.4, 181.19999999999982, 'gini = 0.215\nsamples = 53\nvalue = [10, 80, 1]\nclas
s = b'),
 Text(2529.6000000000004, 543.5999999999999, 'TCH <= 1.375\ngini = 0.461\nsamples = 735
\nvalue = [818, 94, 267]\nclass = a'),
 Text(2455.2000000000003, 181.19999999999982, 'gini = 0.081\nsamples = 432\nvalue = [67
9, 4, 26]\nclass = a'),
 Text(2604.0, 181.19999999999982, 'gini = 0.613\nsamples = 303\nvalue = [139, 90, 241]\n
class = c'),
 Text(2976.0, 906.0, 'PM10 <= 59.06\ngini = 0.115\nsamples = 701\nvalue = [1061, 29, 39]
\nclass = a'),
 Text(2827.2000000000003, 543.5999999999999, 'O_3 <= 7.96\ngini = 0.074\nsamples = 580\n
value = [912, 16, 20]\nclass = a'),
 Text(2752.8, 181.19999999999982, 'gini = 0.604\nsamples = 20\nvalue = [12, 14, 4]\nclas
s = b'),
 Text(2901.6000000000004, 181.19999999999982, 'gini = 0.039\nsamples = 560\nvalue = [90
0, 2, 16]\nclass = a'),
 Text(3124.8, 543.5999999999999, 'O_3 <= 11.24\ngini = 0.306\nsamples = 121\nvalue = [14
9, 13, 19]\nclass = a'),
 Text(3050.4, 181.19999999999982, 'gini = 0.645\nsamples = 21\nvalue = [7, 13, 13]\nclas
s = b'),
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.078\nsamples = 100\nvalue = [14
2, 0, 6]\nclass = a'),
 Text(3868.8, 1268.4, 'NOx <= 98.53\ngini = 0.474\nsamples = 2530\nvalue = [2589, 142, 1
220]\nclass = a'),
 Text(3571.2000000000003, 906.0, 'O_3 <= 38.345\ngini = 0.577\nsamples = 126\nvalue = [6
3, 27, 112]\nclass = c'),
 Text(3422.4, 543.5999999999999, 'NMHC <= 0.115\ngini = 0.636\nsamples = 79\nvalue = [5
9, 27, 43]\nclass = a'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.21\nsamples = 43\nvalue = [59,
0, 8]\nclass = a'),
 Text(3496.8, 181.19999999999982, 'gini = 0.492\nsamples = 36\nvalue = [0, 27, 35]\nclas
s = c'),
 Text(3720.0000000000005, 543.5999999999999, 'TCH <= 1.365\ngini = 0.104\nsamples = 47\n
value = [4, 0, 69]\nclass = c'),
 Text(3645.6000000000004, 181.19999999999982, 'gini = 0.188\nsamples = 24\nvalue = [4,
0, 34]\nclass = c'),
 Text(3794.4, 181.19999999999982, 'gini = 0.0\nsamples = 23\nvalue = [0, 0, 35]\nclass =
c'),
 Text(4166.400000000001, 906.0, 'O_3 <= 6.095\ngini = 0.458\nsamples = 2404\nvalue = [25
26, 115, 1108]\nclass = a'),
 Text(4017.6000000000004, 543.5999999999999, 'PXY <= 2.935\ngini = 0.498\nsamples = 222
\nvalue = [44, 75, 236]\nclass = c'),
 Text(3943.2000000000003, 181.19999999999982, 'gini = 0.585\nsamples = 84\nvalue = [14,
65, 52]\nclass = b'),
 Text(4092.0000000000005, 181.19999999999982, 'gini = 0.305\nsamples = 138\nvalue = [30,
10, 184]\nclass = c'),
 Text(4315.200000000001, 543.5999999999999, 'O_3 <= 8.295\ngini = 0.399\nsamples = 2182
\nvalue = [2482, 40, 872]\nclass = a'),
 Text(4240.8, 181.19999999999982, 'gini = 0.483\nsamples = 703\nvalue = [656, 19, 378]\n
class = a'),
 Text(4389.6, 181.19999999999982, 'gini = 0.347\nsamples = 1479\nvalue = [1826, 21, 494]
\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: 0.3036475687503324
Ridge Regression: 0.30313654420191183
Lasso Regression 0.059641260552344866
ElasticNet Regression: 0.17395085484374662
Logistic Regression: 0.879023418036871
Random Forest: 0.8646161305620181
```

Logistic Regression is suitable for this dataset