# 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_2010.csv")
df
```

Out[2]:

| | date | BEN | CO | EBE | MXY | NMHC | NO_2 | NOx | OXY | O_3 | PM10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2010-03-01 01:00:00 | NaN | 0.29 | NaN | NaN | NaN | 25.090000 | 29.219999 | NaN | 68.930000 | NaN | |
| **1** | 2010-03-01 01:00:00 | NaN | 0.27 | NaN | NaN | NaN | 24.879999 | 30.040001 | NaN | NaN | NaN | |
| **2** | 2010-03-01 01:00:00 | NaN | 0.28 | NaN | NaN | NaN | 17.410000 | 20.540001 | NaN | 72.120003 | NaN | |
| **3** | 2010-03-01 01:00:00 | 0.38 | 0.24 | 1.74 | NaN | 0.05 | 15.610000 | 21.080000 | NaN | 72.970001 | 19.410000 | |
| **4** | 2010-03-01 01:00:00 | 0.79 | NaN | 1.32 | NaN | NaN | 21.430000 | 26.070000 | NaN | NaN | 24.670000 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **209443** | 2010-08-01 00:00:00 | NaN | 0.55 | NaN | NaN | NaN | 125.000000 | 219.899994 | NaN | 25.379999 | NaN | |
| **209444** | 2010-08-01 00:00:00 | NaN | 0.27 | NaN | NaN | NaN | 45.709999 | 47.410000 | NaN | NaN | 51.259998 | |
| **209445** | 2010-08-01 00:00:00 | 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** | 2010-08-01 00:00:00 | NaN | NaN | NaN | NaN | NaN | 46.770000 | 50.119999 | NaN | 77.709999 | NaN | |
| **209447** | 2010-08-01 00:00:00 | 0.92 | 0.43 | 0.71 | NaN | 0.25 | 76.330002 | 88.190002 | NaN | 52.259998 | 47.150002 | 2 |

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 |

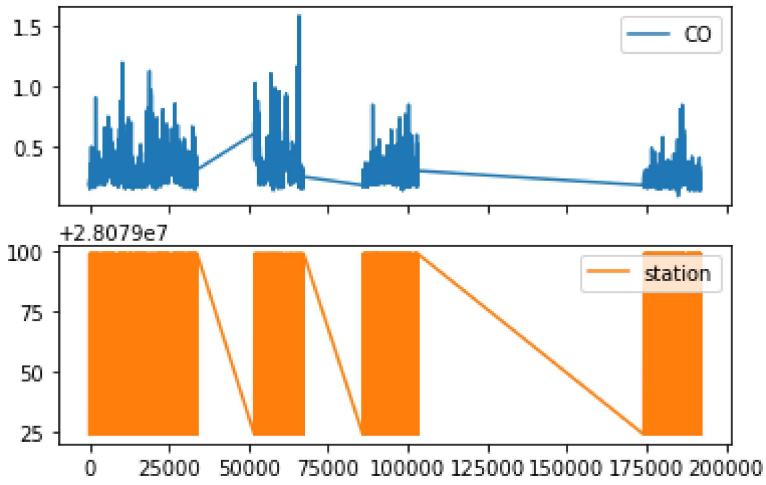
| | CO | station |
|---|---|---|
| **23** | 0.29 | 28079099 |
| **35** | 0.17 | 28079024 |
| **47** | 0.21 | 28079099 |
| **59** | 0.16 | 28079024 |
| **...** | ... | ... |
| **191879** | 0.26 | 28079099 |
| **191891** | 0.16 | 28079024 |
| **191903** | 0.28 | 28079099 |
| **191915** | 0.16 | 28079024 |
| **191927** | 0.25 | 28079099 |

6666 rows × 2 columns

# Line chart

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

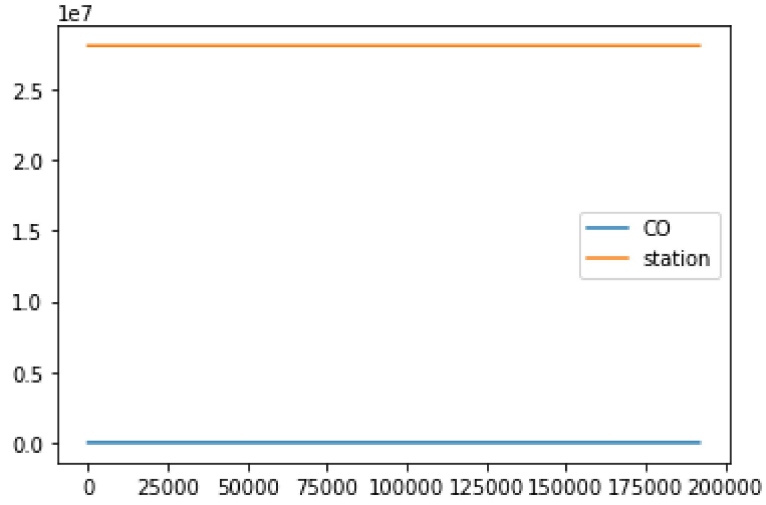
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```




# Line chart

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

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

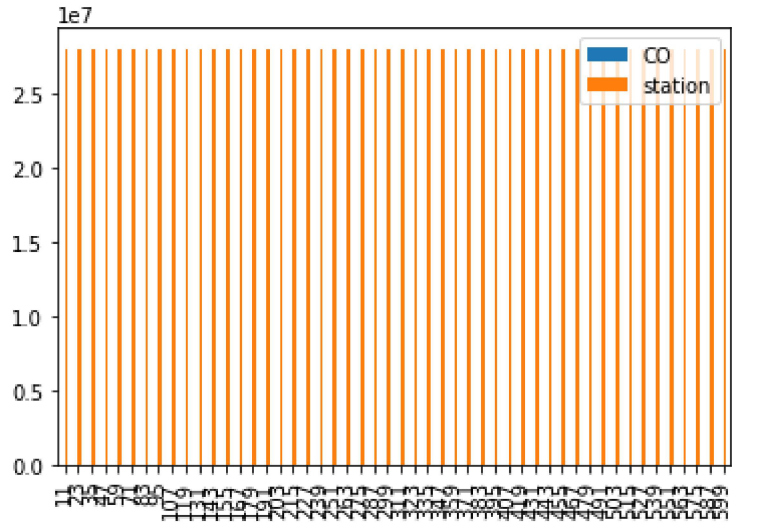



# Bar chart

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

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

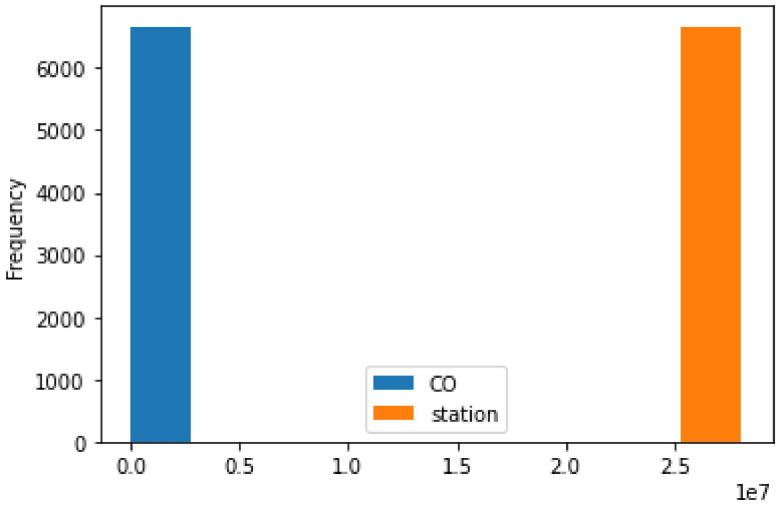
Out[10]: <AxesSubplot:>




# Histogram

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

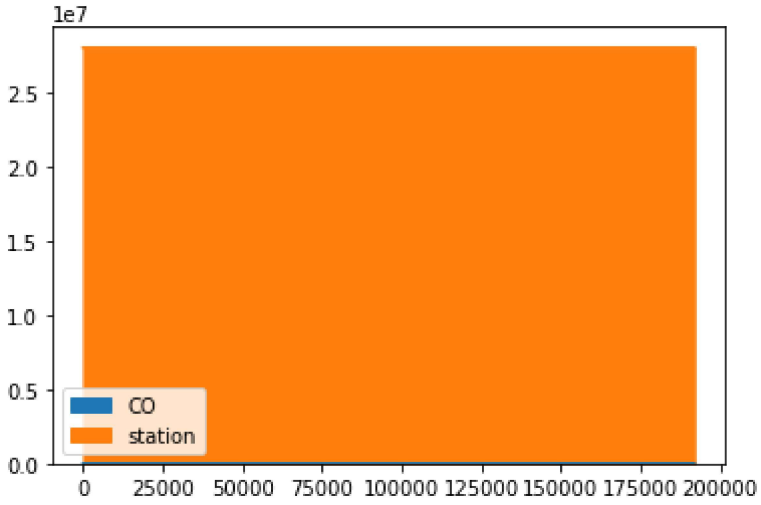
Out[11]: <AxesSubplot:ylabel='Frequency'>




# Area chart

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

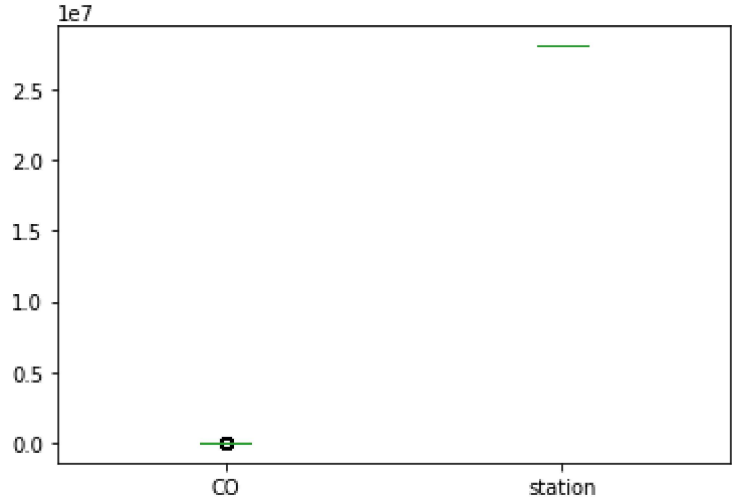
Out[12]: <AxesSubplot:>




# Box chart

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

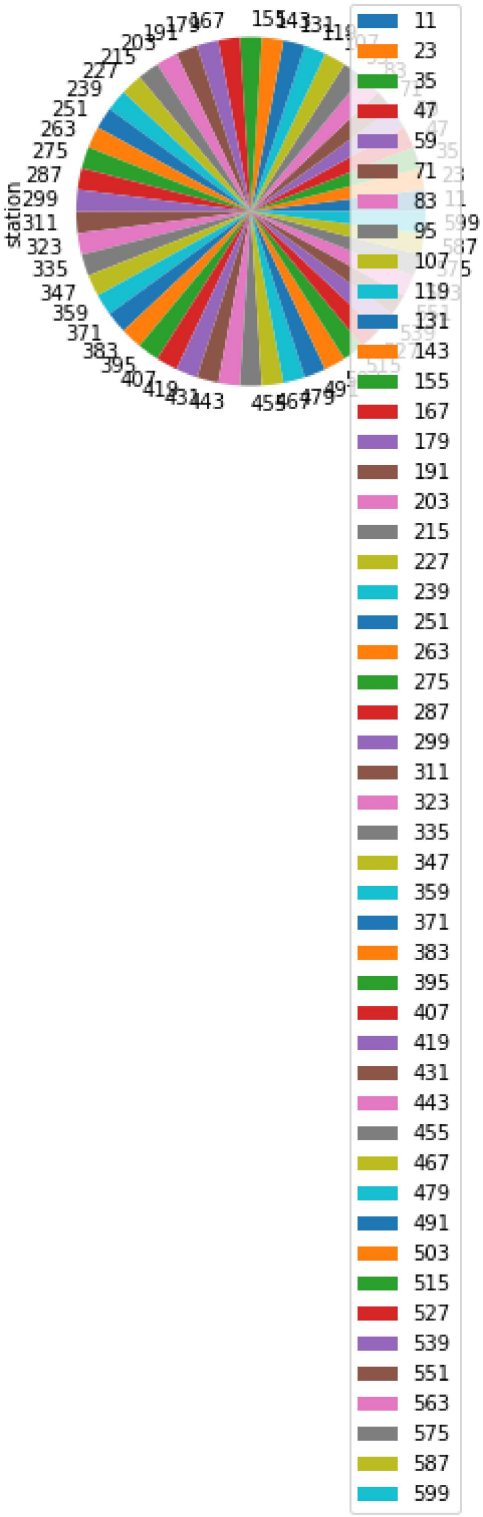
Out[13]: <AxesSubplot:>




# Pie chart

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

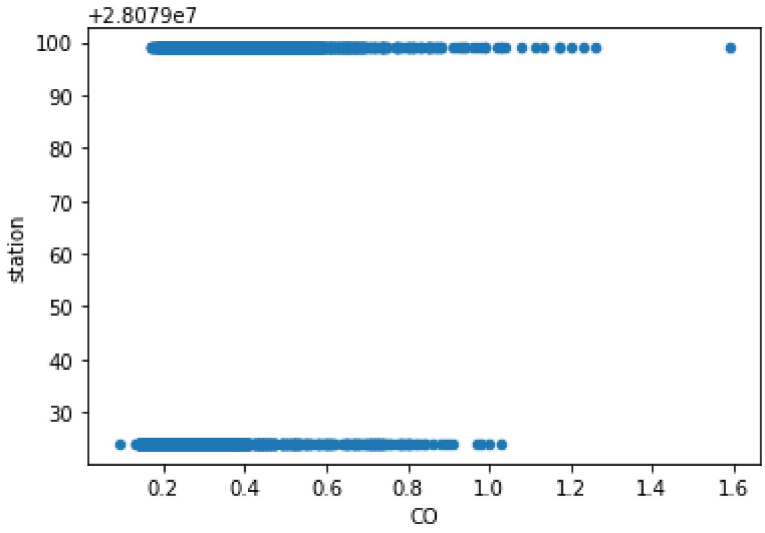
Out[14]:
```
<AxesSubplot:ylabel='station'>
```




# 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. |

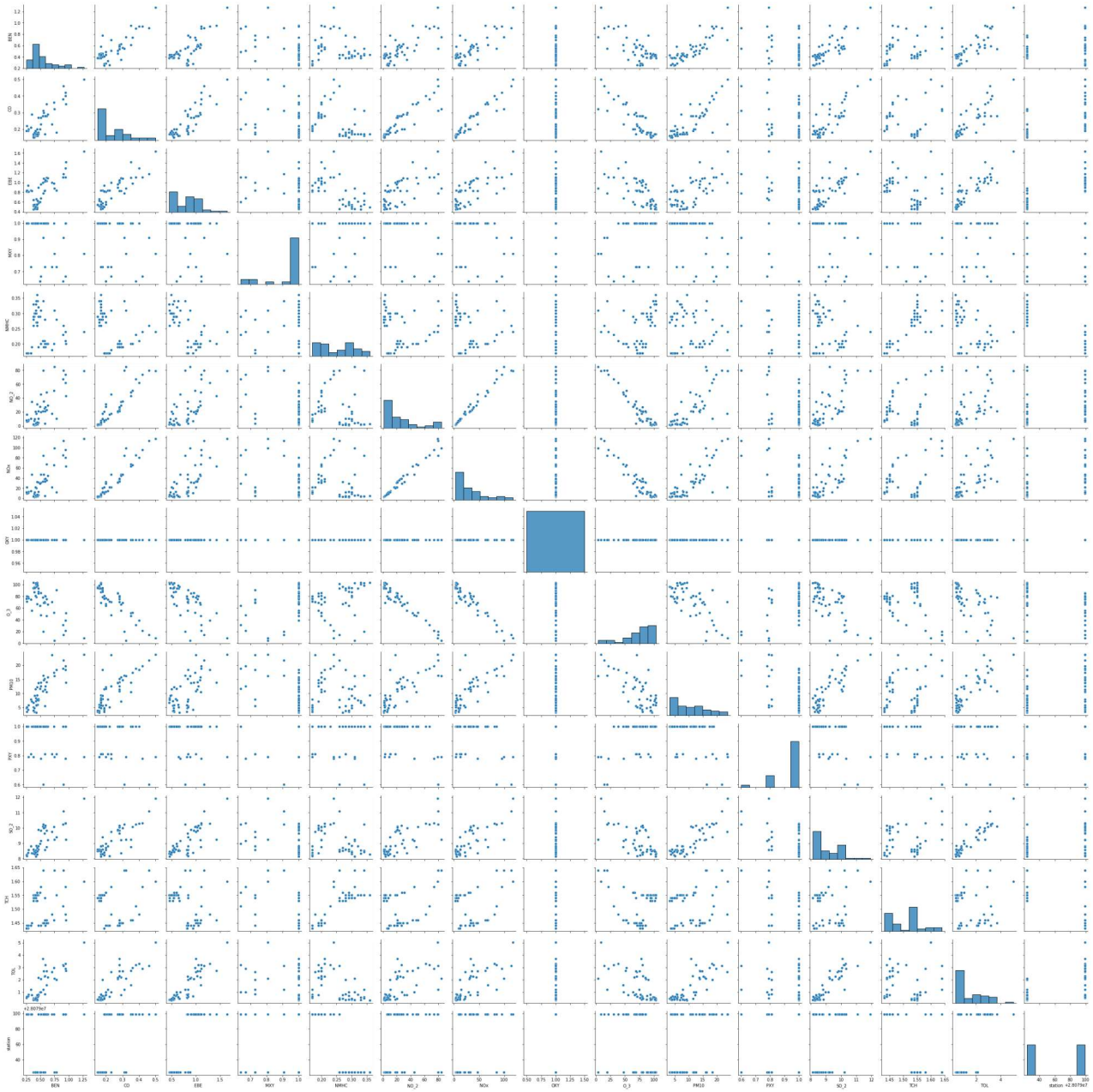
| | BEN | CO | EBE | MXY | NMHC | NO_2 | NOx | |
|---|---|---|---|---|---|---|---|---|
| **75%** | 0.810000 | 0.340000 | 1.000000 | 1.000000 | 0.280000 | 47.657500 | 62.102501 | 1. |
| **max** | 5.110000 | 1.590000 | 5.190000 | 6.810000 | 0.930000 | 133.399994 | 409.299988 | 2. |

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])
```

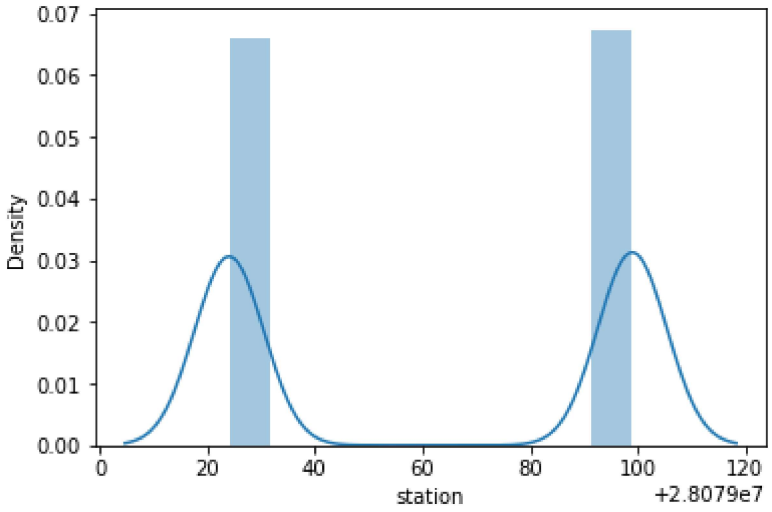
Out[19]: <seaborn.axisgrid.PairGrid at 0x1d273bb29a0>



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

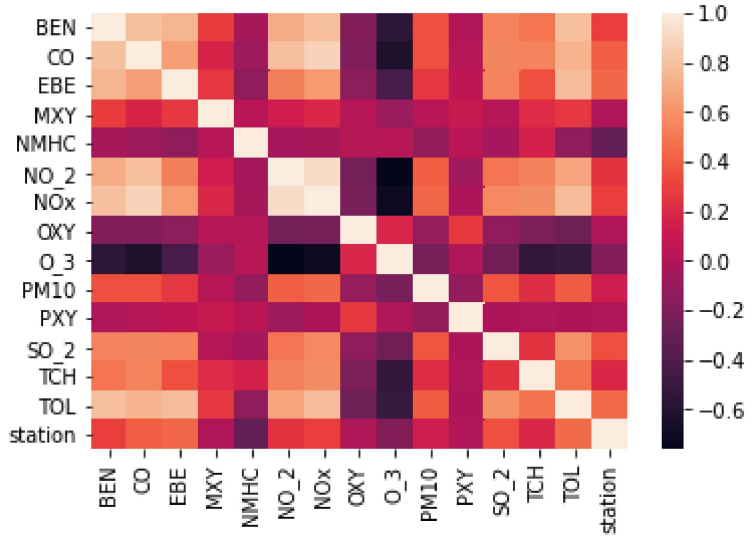
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function 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]:
```
28078938.64318518
```

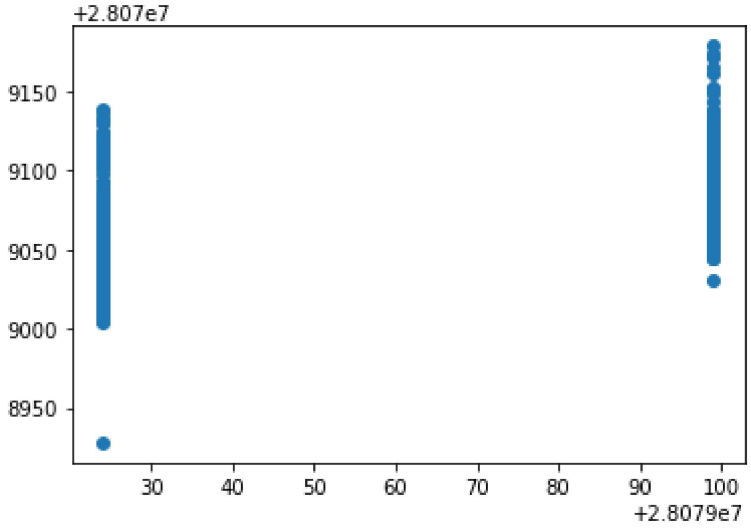
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

| | Co-efficient |
|---|---|
| **BEN** | -37.033725 |
| **CO** | 179.006515 |
| **EBE** | 17.499675 |
| **MXY** | -9.068156 |
| **NMHC** | -75.033110 |
| **NO_2** | 0.297505 |
| **NOx** | -0.640456 |
| **OXY** | 28.988090 |
| **O_3** | 0.045980 |
| **PM10** | -0.143751 |
| **PXY** | -4.746798 |
| **SO_2** | 1.884593 |
| **TCH** | 45.179861 |
| **TOL** | 8.932138 |

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x1d2033029d0>
```




# ACCURACY

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

Out[28]: 0.42636209059174646

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

Out[29]: 0.4269764306886642

# 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]: 0.4232594416267147

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

Out[33]: 0.41331394362829144

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]:
```
0.1716061804846113
```

# Accuracy(Lasso)

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

Out[36]:
```
0.1855608532882559
```

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([-0.00000000e+00,  2.31547529e-01,  2.82783475e+00, -1.44499197e+00,
       -1.24109383e+00, -6.60189260e-03, -1.09859804e-01,  3.74495431e-01,
       -6.29269500e-02, -1.13552062e-01,  0.00000000e+00,  2.66893845e+00,
        0.00000000e+00,  6.62846880e+00])
```

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

Out[39]:
```
28079029.885198087
```

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

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

Out[41]:
```
0.2482316620660071
```

# 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)))
```

```
30.74620879883878
1056.9670190715344
32.51102919120732
```

# 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]:
```
0.928204029147021
```

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

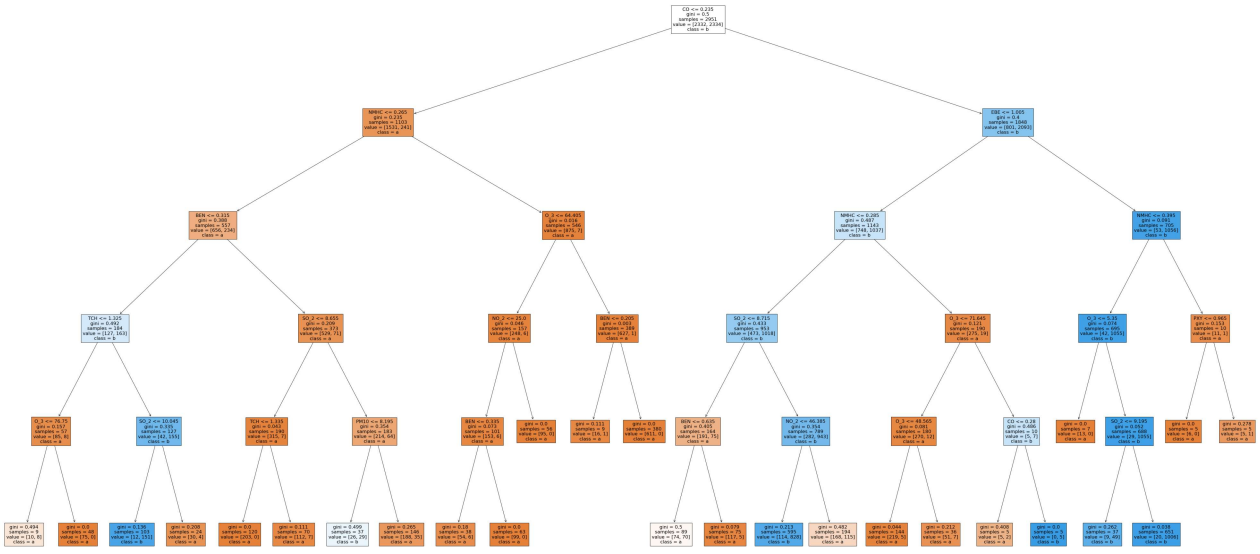
In [62]:
```
from sklearn.tree import plot_tree

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

Out[62]:
```
[Text(2469.446808510638, 1993.2, 'CO <= 0.235\ngini = 0.5\nsamples = 2951\nvalue = [233
2, 2334]\nclass = b'),
 Text(1377.1914893617022, 1630.8000000000002, 'NMHC <= 0.265\ngini = 0.235\nsamples = 11
03\nvalue = [1531, 241]\nclass = a'),
```

```
 Text(759.8297872340426, 1268.4, 'BEN <= 0.315\ngini = 0.388\nsamples = 557\nvalue = [65
6, 234]\nclass = a'),
 Text(379.9148936170213, 906.0, 'TCH <= 1.325\ngini = 0.492\nsamples = 184\nvalue = [12
7, 163]\nclass = b'),
 Text(189.95744680851064, 543.5999999999999, 'O_3 <= 76.75\ngini = 0.157\nsamples = 57\n
value = [85, 8]\nclass = a'),
 Text(94.97872340425532, 181.19999999999982, 'gini = 0.494\nsamples = 9\nvalue = [10, 8]
\nclass = a'),
 Text(284.93617021276594, 181.19999999999982, 'gini = 0.0\nsamples = 48\nvalue = [75, 0]
\nclass = a'),
 Text(569.8723404255319, 543.5999999999999, 'SO_2 <= 10.045\ngini = 0.335\nsamples = 127
\nvalue = [42, 155]\nclass = b'),
 Text(474.8936170212766, 181.19999999999982, 'gini = 0.136\nsamples = 103\nvalue = [12,
151]\nclass = b'),
 Text(664.8510638297872, 181.19999999999982, 'gini = 0.208\nsamples = 24\nvalue = [30,
4]\nclass = a'),
 Text(1139.7446808510638, 906.0, 'SO_2 <= 8.655\ngini = 0.209\nsamples = 373\nvalue = [5
29, 71]\nclass = a'),
 Text(949.7872340425532, 543.5999999999999, 'TCH <= 1.335\ngini = 0.043\nsamples = 190\n
value = [315, 7]\nclass = a'),
 Text(854.8085106382979, 181.19999999999982, 'gini = 0.0\nsamples = 120\nvalue = [203,
0]\nclass = a'),
 Text(1044.7659574468084, 181.19999999999982, 'gini = 0.111\nsamples = 70\nvalue = [112,
7]\nclass = a'),
 Text(1329.7021276595744, 543.5999999999999, 'PM10 <= 8.195\ngini = 0.354\nsamples = 183
\nvalue = [214, 64]\nclass = a'),
 Text(1234.723404255319, 181.19999999999982, 'gini = 0.499\nsamples = 37\nvalue = [26, 2
9]\nclass = b'),
 Text(1424.6808510638298, 181.19999999999982, 'gini = 0.265\nsamples = 146\nvalue = [18
8, 35]\nclass = a'),
 Text(1994.5531914893618, 1268.4, 'O_3 <= 64.405\ngini = 0.016\nsamples = 546\nvalue =
[875, 7]\nclass = a'),
 Text(1804.595744680851, 906.0, 'NO_2 <= 25.0\ngini = 0.046\nsamples = 157\nvalue = [24
8, 6]\nclass = a'),
 Text(1709.6170212765958, 543.5999999999999, 'BEN <= 0.335\ngini = 0.073\nsamples = 101
\nvalue = [153, 6]\nclass = a'),
 Text(1614.6382978723404, 181.19999999999982, 'gini = 0.18\nsamples = 38\nvalue = [54,
6]\nclass = a'),
 Text(1804.595744680851, 181.19999999999982, 'gini = 0.0\nsamples = 63\nvalue = [99, 0]
\nclass = a'),
 Text(1899.5744680851064, 543.5999999999999, 'gini = 0.0\nsamples = 56\nvalue = [95, 0]
\nclass = a'),
 Text(2184.5106382978724, 906.0, 'BEN <= 0.205\ngini = 0.003\nsamples = 389\nvalue = [62
7, 1]\nclass = a'),
 Text(2089.531914893617, 543.5999999999999, 'gini = 0.111\nsamples = 9\nvalue = [16, 1]
\nclass = a'),
 Text(2279.4893617021276, 543.5999999999999, 'gini = 0.0\nsamples = 380\nvalue = [611,
0]\nclass = a'),
 Text(3561.7021276595747, 1630.8000000000002, 'EBE <= 1.005\ngini = 0.4\nsamples = 1848
\nvalue = [801, 2093]\nclass = b'),
 Text(3039.31914893617, 1268.4, 'NMHC <= 0.285\ngini = 0.487\nsamples = 1143\nvalue = [7
48, 1037]\nclass = b'),
 Text(2659.404255319149, 906.0, 'SO_2 <= 8.715\ngini = 0.433\nsamples = 953\nvalue = [47
3, 1018]\nclass = b'),
 Text(2469.446808510638, 543.5999999999999, 'BEN <= 0.635\ngini = 0.405\nsamples = 164\n
value = [191, 75]\nclass = a'),
 Text(2374.468085106383, 181.19999999999982, 'gini = 0.5\nsamples = 89\nvalue = [74, 70]
\nclass = a'),
 Text(2564.425531914894, 181.19999999999982, 'gini = 0.079\nsamples = 75\nvalue = [117,
5]\nclass = a'),
 Text(2849.3617021276596, 543.5999999999999, 'NO_2 <= 46.385\ngini = 0.354\nsamples = 78
9\nvalue = [282, 943]\nclass = b'),
 Text(2754.3829787234044, 181.19999999999982, 'gini = 0.213\nsamples = 595\nvalue = [11
4, 828]\nclass = b'),
 Text(2944.340425531915, 181.19999999999982, 'gini = 0.482\nsamples = 194\nvalue = [168,
```

```
115]\nclass = a'),
 Text(3419.2340425531916, 906.0, 'O_3 <= 71.645\ngini = 0.121\nsamples = 190\nvalue = [2
75, 19]\nclass = a'),
 Text(3229.276595744681, 543.5999999999999, 'O_3 <= 48.565\ngini = 0.081\nsamples = 180
\nvalue = [270, 12]\nclass = a'),
 Text(3134.2978723404253, 181.19999999999982, 'gini = 0.044\nsamples = 144\nvalue = [21
9, 5]\nclass = a'),
 Text(3324.255319148936, 181.19999999999982, 'gini = 0.212\nsamples = 36\nvalue = [51,
7]\nclass = a'),
 Text(3609.191489361702, 543.5999999999999, 'CO <= 0.28\ngini = 0.486\nsamples = 10\nval
ue = [5, 7]\nclass = b'),
 Text(3514.2127659574467, 181.19999999999982, 'gini = 0.408\nsamples = 5\nvalue = [5, 2]
\nclass = a'),
 Text(3704.1702127659573, 181.19999999999982, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]\n
class = b'),
 Text(4084.0851063829787, 1268.4, 'NMHC <= 0.395\ngini = 0.091\nsamples = 705\nvalue =
[53, 1056]\nclass = b'),
 Text(3894.127659574468, 906.0, 'O_3 <= 5.35\ngini = 0.074\nsamples = 695\nvalue = [42,
1055]\nclass = b'),
 Text(3799.148936170213, 543.5999999999999, 'gini = 0.0\nsamples = 7\nvalue = [13, 0]\nc
lass = a'),
 Text(3989.1063829787236, 543.5999999999999, 'SO_2 <= 9.195\ngini = 0.052\nsamples = 688
\nvalue = [29, 1055]\nclass = b'),
 Text(3894.127659574468, 181.19999999999982, 'gini = 0.262\nsamples = 37\nvalue = [9, 4
9]\nclass = b'),
 Text(4084.0851063829787, 181.19999999999982, 'gini = 0.038\nsamples = 651\nvalue = [20,
1006]\nclass = b'),
 Text(4274.04255319149, 906.0, 'PXY <= 0.965\ngini = 0.153\nsamples = 10\nvalue = [11,
1]\nclass = a'),
 Text(4179.063829787234, 543.5999999999999, 'gini = 0.0\nsamples = 5\nvalue = [6, 0]\ncl
ass = a'),
 Text(4369.021276595745, 543.5999999999999, 'gini = 0.278\nsamples = 5\nvalue = [5, 1]\n
class = 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: 0.42636209059174646
Ridge Regression: 0.4232594416267147
Lasso Regression 0.1855608532882559
ElasticNet Regression: 0.2482316620660071
Logistic Regression: 0.8660366036603661
Random Forest: 0.928204029147021
```

# Random Forest is suitable for this dataset