Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [63]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2003.
df
```

Out[63]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.2099
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.3899
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.2400
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.8399
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.7799
...	...	...	...	...	...	...	...	...	...	...	
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	7.3800
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.4000
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.8300
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.5700
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	12.3500

243984 rows × 16 columns

Data Cleaning and Data Preprocessing

In [64]:
```
df=df.dropna()
```

In [65]:
```
df.columns
```

Out[65]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [66]:
```
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 [67]:
```
data=df[['CO' ,'station']]
data
```

Out[67]:

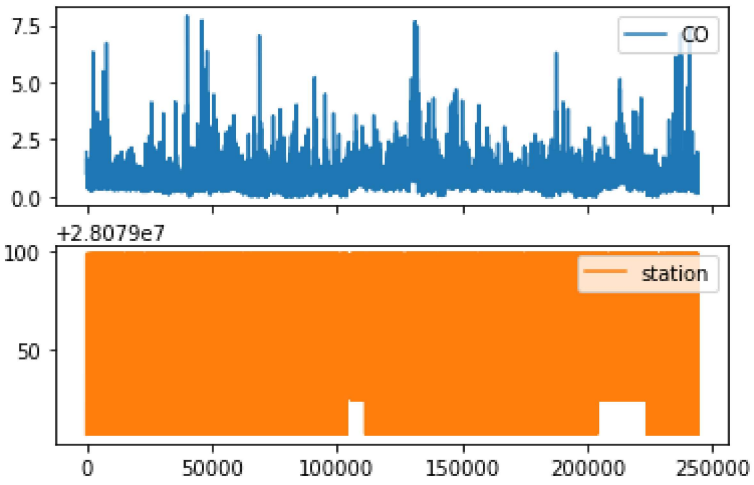
	CO	station
5	1.94	28079006
23	1.27	28079024
27	1.79	28079099
33	1.47	28079006
51	1.29	28079024
...	...	...
243955	0.41	28079099
243957	0.60	28079035
243961	0.82	28079006
243979	0.16	28079024
243983	0.29	28079099

33010 rows × 2 columns

Line chart

In [68]:
```
data.plot.line(subplots=True)
```

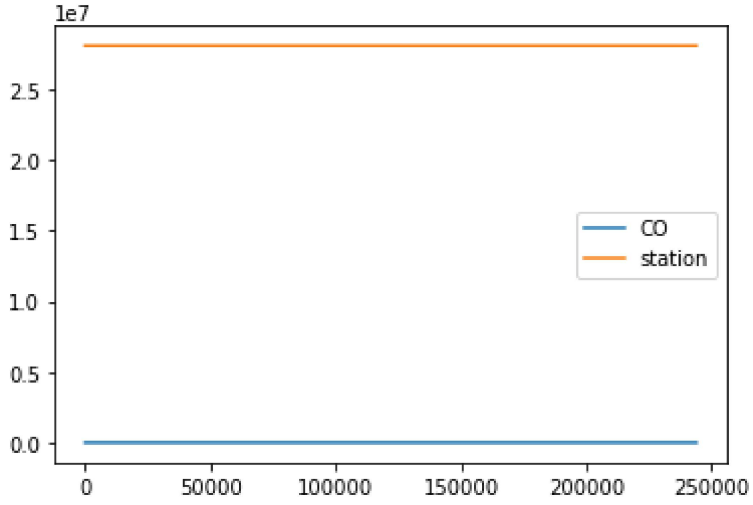
Out[68]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [69]:
```
data.plot.line()
```

Out[69]: <AxesSubplot:>

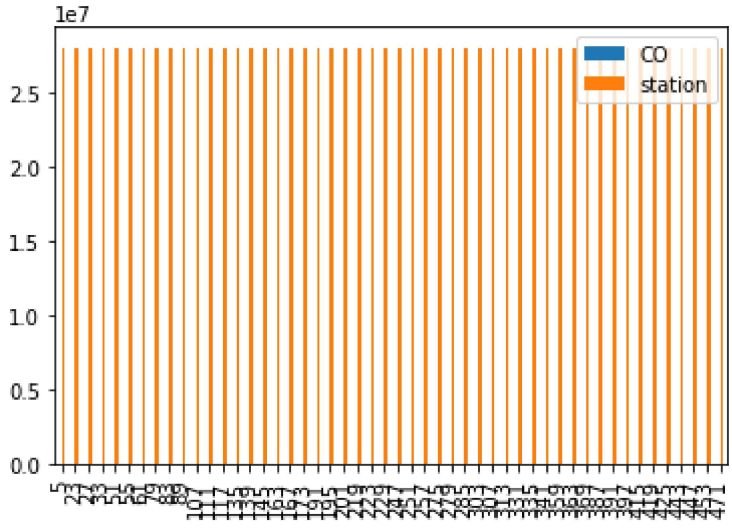


Bar chart

In [70]:
```
b=data[0:50]
```

In [71]:
```
b.plot.bar()
```

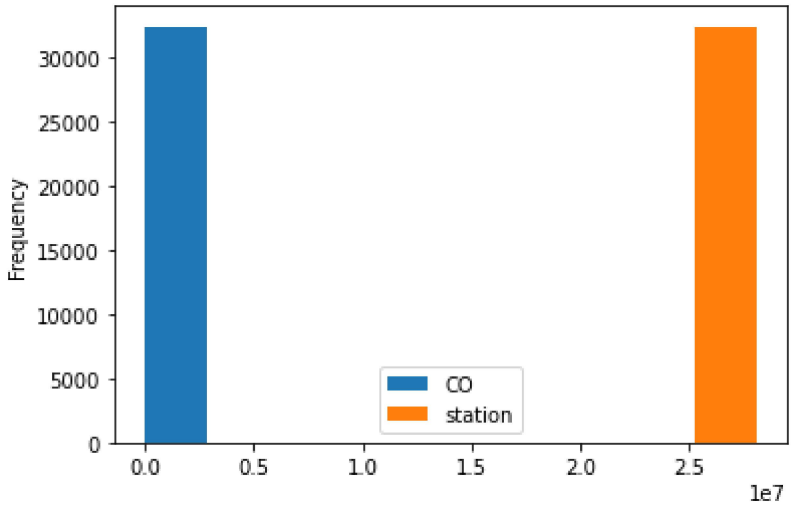
Out[71]: <AxesSubplot:>



Histogram

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

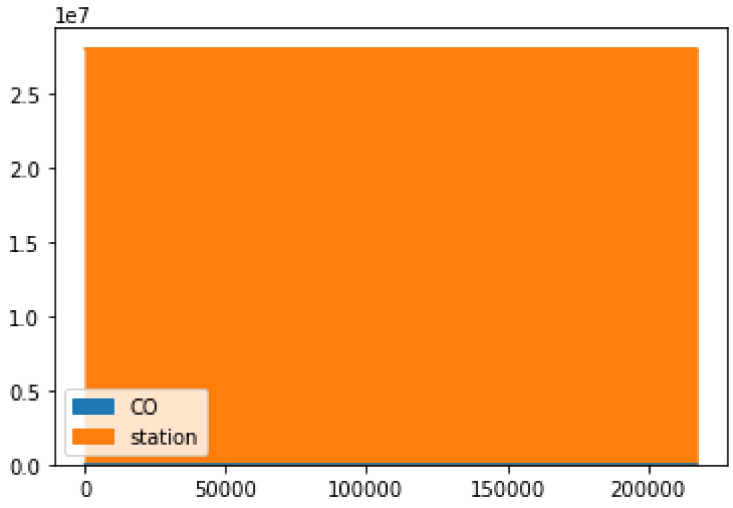
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

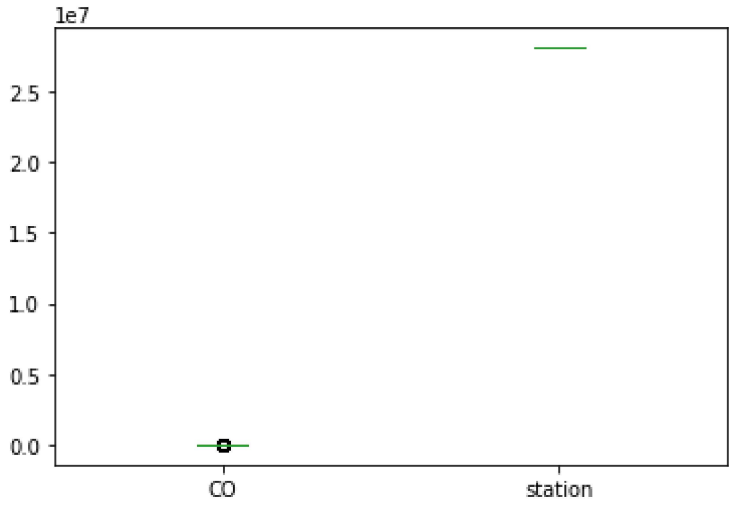
Out[12]: <AxesSubplot:>



Box chart

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

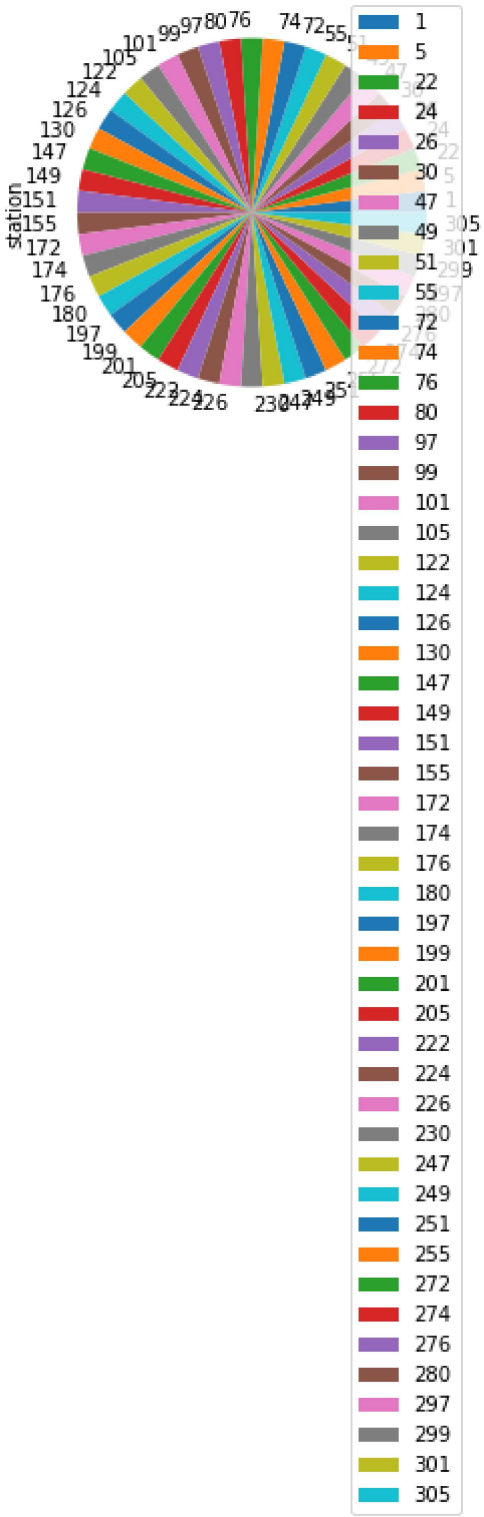
Out[13]: <AxesSubplot:>



Pie chart

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

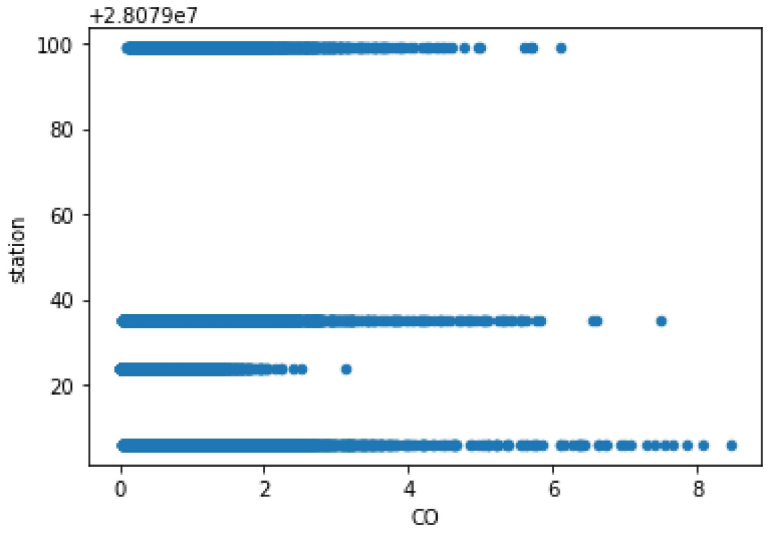
Out[14]: <AxesSubplot:ylabel='station'>



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: 32381 entries, 1 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     32381 non-null  object 
 1   BEN      32381 non-null  float64
 2   CO       32381 non-null  float64
 3   EBE      32381 non-null  float64
 4   MXY      32381 non-null  float64
 5   NMHC     32381 non-null  float64
 6   NO_2     32381 non-null  float64
 7   NOx      32381 non-null  float64
 8   OXY      32381 non-null  float64
 9   O_3      32381 non-null  float64
 10  PM10     32381 non-null  float64
 11  PXY      32381 non-null  float64
 12  SO_2     32381 non-null  float64
 13  TCH      32381 non-null  float64
 14  TOL      32381 non-null  float64
 15  station  32381 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 4.2+ MB
```

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

Out[17]:

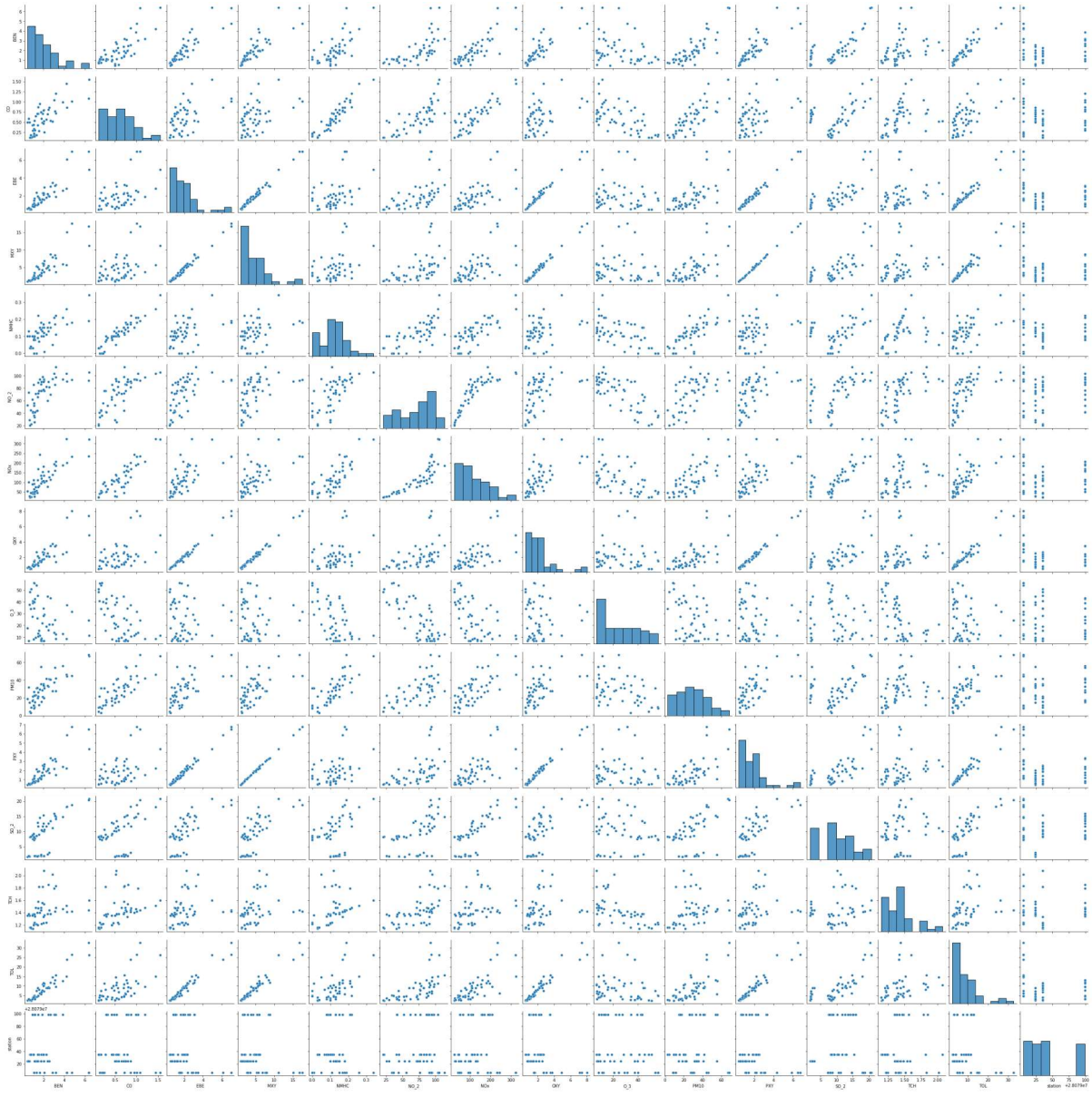
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000	32381.000000	3238
mean	2.479155	0.787323	2.914004	7.013636	0.155827	58.936796	1:
std	2.280959	0.610810	2.667881	6.774365	0.135731	31.472733	1
min	0.180000	0.000000	0.180000	0.190000	0.000000	0.890000	
25%	0.970000	0.420000	1.140000	2.420000	0.080000	35.660000	4
50%	1.840000	0.620000	2.130000	5.140000	0.130000	57.160000	9
75%	3.250000	0.980000	3.830000	9.420000	0.200000	78.769997	1(
max	32.660000	8.460000	41.740002	99.879997	2.700000	263.600006	13:

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])
```

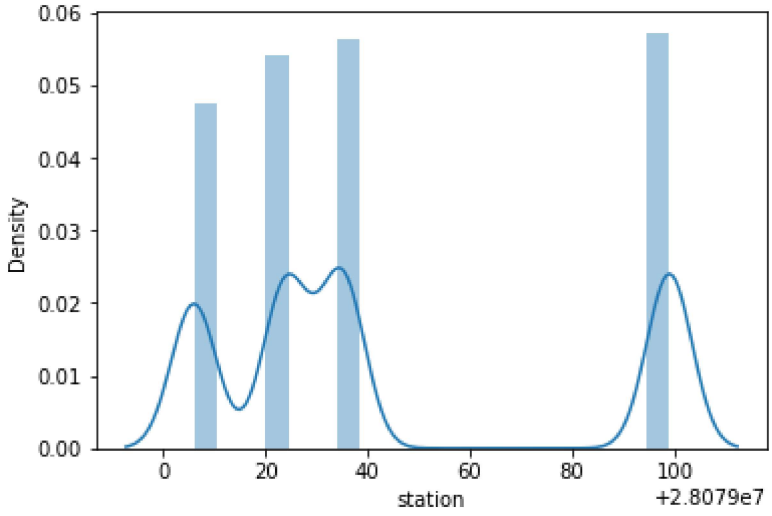
Out[19]: <seaborn.axisgrid.PairGrid at 0x1cd706f87c0>



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

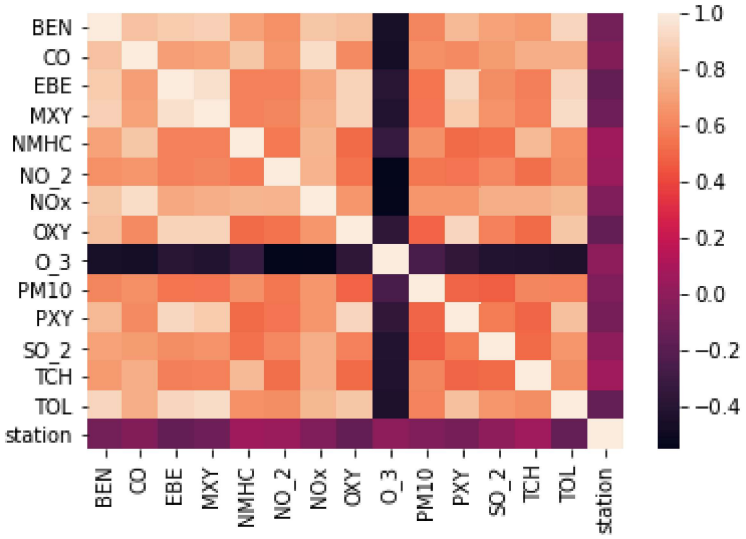
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  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]:
```
28078986.118767593
```

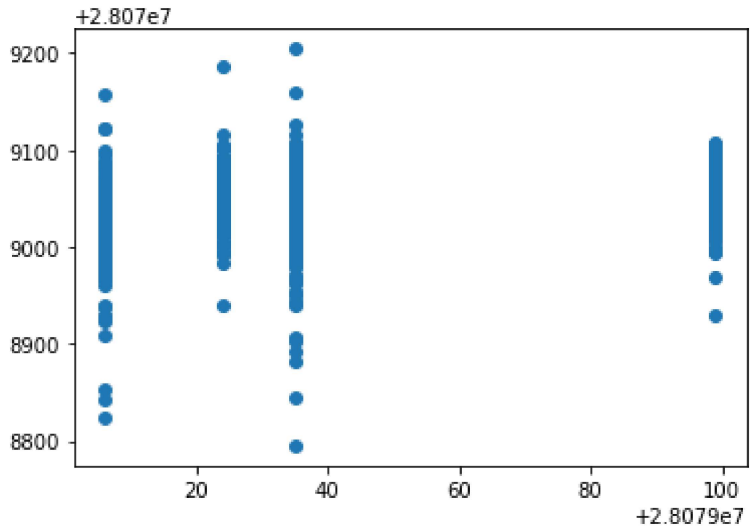
In [72]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[72]:

	Co-efficient
BEN	2.274450
CO	-13.582455
EBE	-11.456936
MXY	4.132700
NMHC	78.599697
NO_2	0.248235
NOx	-0.094317
OXY	-5.283710
O_3	-0.025774
PM10	-0.108382
PXY	7.735897
SO_2	0.600407
TCH	45.166445
TOL	-1.564016

In [73]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[73]: <matplotlib.collections.PathCollection at 0x1cd02222d30>



ACCURACY

In [74]:
```
lr.score(x_test,y_test)
```

Out[74]: 0.1937170619900085

In [75]:
```
lr.score(x_train,y_train)
```

Out[75]: 0.20059181248043334

Ridge and Lasso

In [76]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [77]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[77]: Ridge(alpha=10)

Accuracy(Ridge)

In [78]:
```
rr.score(x_test,y_test)
```

Out[78]:
```
0.19266041562106995
```

In [79]:
```
rr.score(x_train,y_train)
```

Out[79]:
```
0.2004088944667991
```

In [80]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[80]:
```
Lasso(alpha=10)
```

In [81]:
```
la.score(x_train,y_train)
```

Out[81]:
```
0.057740736606634924
```

Accuracy(Lasso)

In [82]:
```
la.score(x_test,y_test)
```

Out[82]:
```
0.05834962560339074
```

In [83]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[83]:
```
ElasticNet()
```

In [84]:
```
en.coef_
```

Out[84]:
```
array([ 1.14363723,  0.        , -2.8545531 ,  1.62939227,  0.21486699,
        0.22571418, -0.02632462, -2.47441929, -0.02159756,  0.01963492,
        2.25749049,  0.38928751,  1.11558424, -1.26974741])
```

In [85]:
```
en.intercept_
```

Out[85]:
```
28079038.361825775
```

In [86]:
```
prediction=en.predict(x_test)
```

In [87]:
```
en.score(x_test,y_test)
```

Out[87]:
```
0.09734806326924794
```

Evaluation Metrics

In [88]:
```
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)))
```

```
28.44943271497292
1112.1725165149585
33.34925061399369
```

Logistic Regression

In [89]:
```
from sklearn.linear_model import LogisticRegression
```

In [90]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [91]:
```
feature_matrix.shape
```

Out[91]:
```
(33010, 14)
```

In [92]:
```
target_vector.shape
```

Out[92]:
```
(33010,)
```

In [93]:
```
from sklearn.preprocessing import StandardScaler
```

In [94]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [95]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[95]:
```
LogisticRegression(max_iter=10000)
```

In [96]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [97]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079035]
```

In [98]:
```
logr.classes_
```

Out[98]:
```
array([28079006, 28079024, 28079035, 28079099], dtype=int64)
```

In [99]:
```
logr.score(fs,target_vector)
```

Out[99]:
```
0.7584974250227204
```

In [100]:
```
logr.predict_proba(observation)[0][0]
```

Out[100]:
```
2.3306153265290618e-23
```

In [101]:
```
logr.predict_proba(observation)
```

Out[101]:
```
array([[2.33061533e-23, 1.44436075e-55, 1.00000000e+00, 6.68457491e-16]])
```

Random Forest

In [102]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [103]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[103]:
```
RandomForestClassifier()
```

In [104]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [105]:
```
from sklearn.model_selection import GridSearchCV
grid_search =GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="acc
grid_search.fit(x_train,y_train)
```

Out[105]:
```
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 [106]:
```
grid_search.best_score_
```

Out[106]:
```
0.769875584576017
```

In [107]:
```
rfc_best=grid_search.best_estimator_
```

In [108]:
```
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',
```

Out[108]:
```
[Text(2111.1000000000004, 1993.2, 'PXY <= 1.005\ngini = 0.749\nsamples = 14
333\nvalue = [4982, 5625, 5945, 6114]\nclass = d'),
 Text(948.6, 1630.8000000000002, 'NO_2 <= 16.285\ngini = 0.43\nsamples = 33
24\nvalue = [138, 3849, 693, 556]\nclass = b'),
 Text(409.20000000000005, 1268.4, 'NOx <= 14.58\ngini = 0.138\nsamples = 10
85\nvalue = [23, 1550, 52, 46]\nclass = b'),
 Text(223.20000000000002, 906.0, 'CO <= 0.245\ngini = 0.055\nsamples = 729
\nvalue = [4, 1108, 21, 7]\nclass = b'),
 Text(148.8, 543.5999999999999, 'PM10 <= 19.795\ngini = 0.351\nsamples = 10
4\nvalue = [4, 122, 21, 7]\nclass = b'),
 Text(74.4, 181.19999999999982, 'gini = 0.481\nsamples = 66\nvalue = [4, 6
7, 21, 6]\nclass = b'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.035\nsamples = 38\n
value = [0, 55, 0, 1]\nclass = b'),
 Text(297.6, 543.5999999999999, 'gini = 0.0\nsamples = 625\nvalue = [0, 98
6, 0, 0]\nclass = b'),
 Text(595.2, 906.0, 'OXY <= 0.785\ngini = 0.297\nsamples = 356\nvalue = [1
9, 442, 31, 39]\nclass = b'),
 Text(446.40000000000003, 543.5999999999999, 'BEN <= 0.475\ngini = 0.188\ns
```

Conclusion

Accuracy

Linear Regression:0.18001211504197578

Ridge Regression:0.03581429332066366

Lasso Regression:0.03428948704282486

ElasticNet Regression:0.04941857713580211

Logistic Regression::0.8480899292795158

Random Forest:0.7293464034866008

From the above data, we can conclude that random forest regression is preferrable to other regression types