Importing Libraries

In [182]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [305]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2007.
df
```

Out[305]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.1
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.8
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.0
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.3
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.5
...	...	...	...	...	...	...	...	...	...	...	
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.7
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	5.7
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	13.0
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	6.6
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.2

225120 rows × 17 columns

Data Cleaning and Data Preprocessing

In [306]:
```
df=df.dropna()
```

In [307]:
```
df.columns
```

Out[307]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [308]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [309]:
```
data=df[['CO' ,'station']]
data
```

Out[309]:

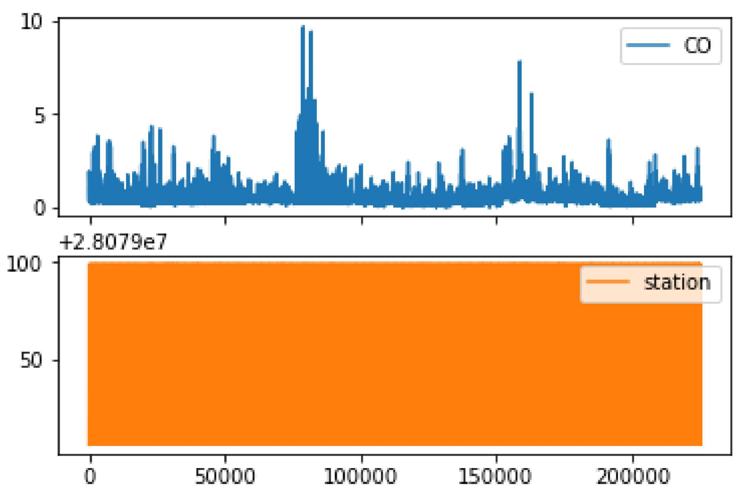
	CO	station
4	1.86	28079006
21	0.31	28079024
25	1.42	28079099
30	1.89	28079006
47	0.30	28079024
...	...	...
225073	0.47	28079006
225094	0.45	28079099
225098	0.41	28079006
225115	0.45	28079024
225119	0.40	28079099

25443 rows × 2 columns

Line chart

In [310]:
```
data.plot.line(subplots=True)
```

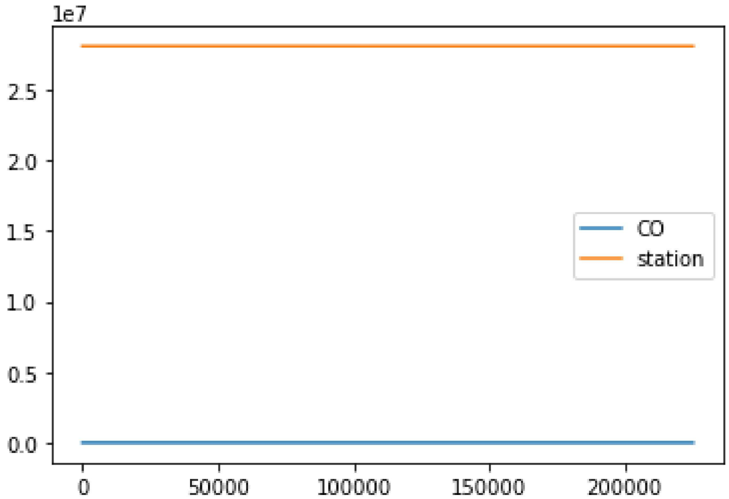
Out[310]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [311]:
```
data.plot.line()
```

Out[311]: <AxesSubplot:>

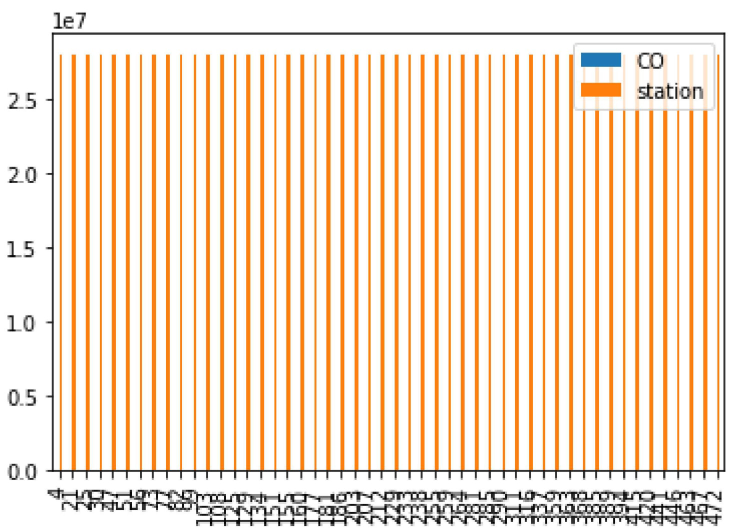


Bar chart

In [312]:
```
b=data[0:50]
```

In [313]:
```
b.plot.bar()
```

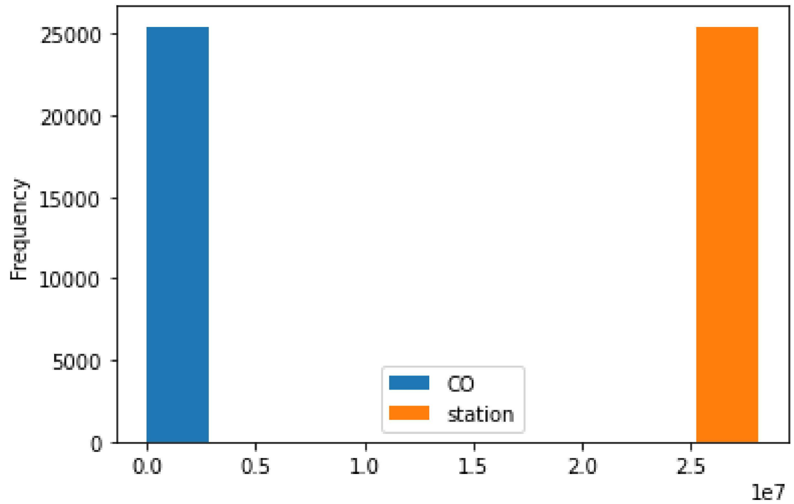
Out[313]: <AxesSubplot:>



Histogram

In [314]:
```
data.plot.hist()
```

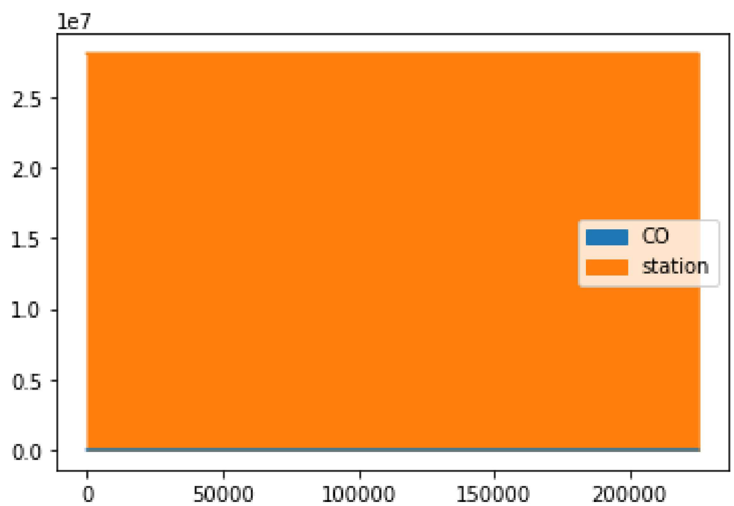
Out[314]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [315]:
```
data.plot.area()
```

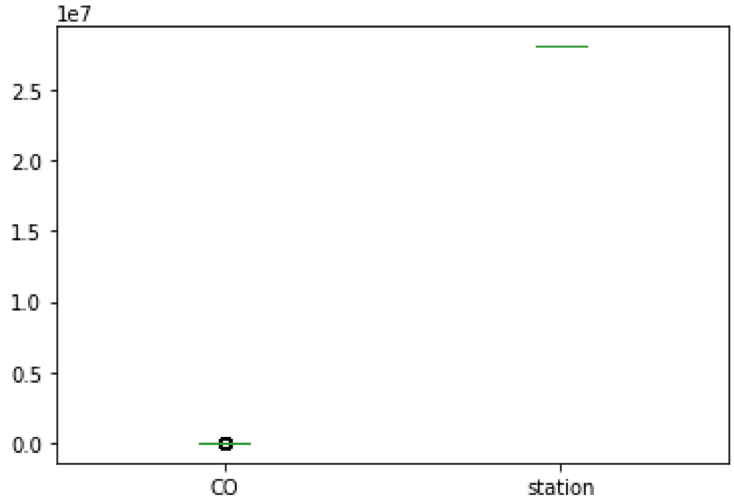
Out[315]: <AxesSubplot:>



Box chart

In [316]:
```
data.plot.box()
```

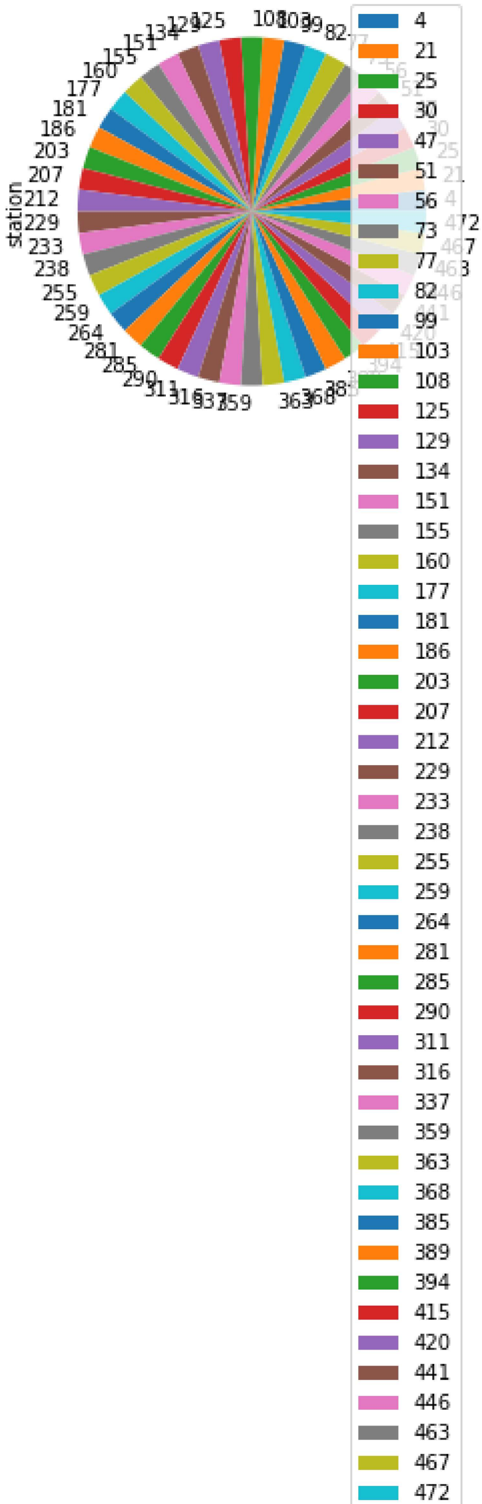
Out[316]: <AxesSubplot:>



Pie chart

In [317]:
```
b.plot.pie(y='station' )
```

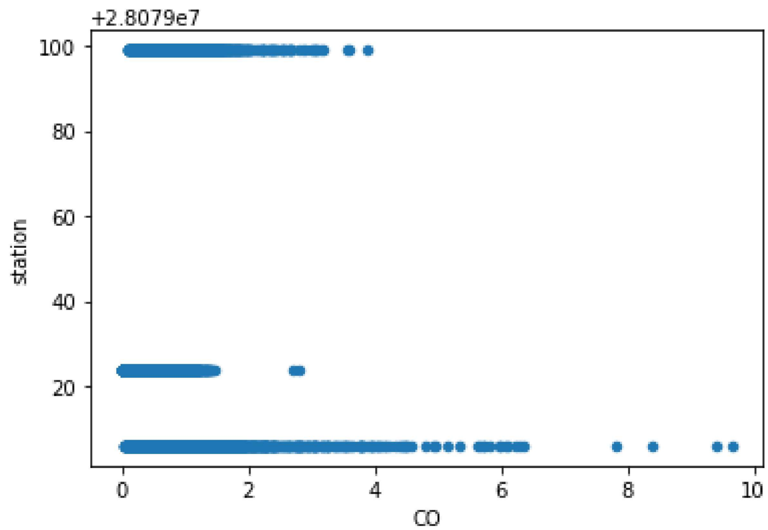
Out[317]: <AxesSubplot:ylabel='station'>



Scatter chart

In [318]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[318]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [319]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25443 entries, 4 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25443 non-null  object 
 1   BEN      25443 non-null  float64
 2   CO       25443 non-null  float64
 3   EBE      25443 non-null  float64
 4   MXY      25443 non-null  float64
 5   NMHC     25443 non-null  float64
 6   NO_2     25443 non-null  float64
 7   NOx      25443 non-null  float64
 8   OXY      25443 non-null  float64
 9   O_3      25443 non-null  float64
 10  PM10     25443 non-null  float64
 11  PM25     25443 non-null  float64
 12  PXY      25443 non-null  float64
 13  SO_2     25443 non-null  float64
 14  TCH      25443 non-null  float64
 15  TOL      25443 non-null  float64
 16  station  25443 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [320]:
```
df.describe()
```

Out[320]:

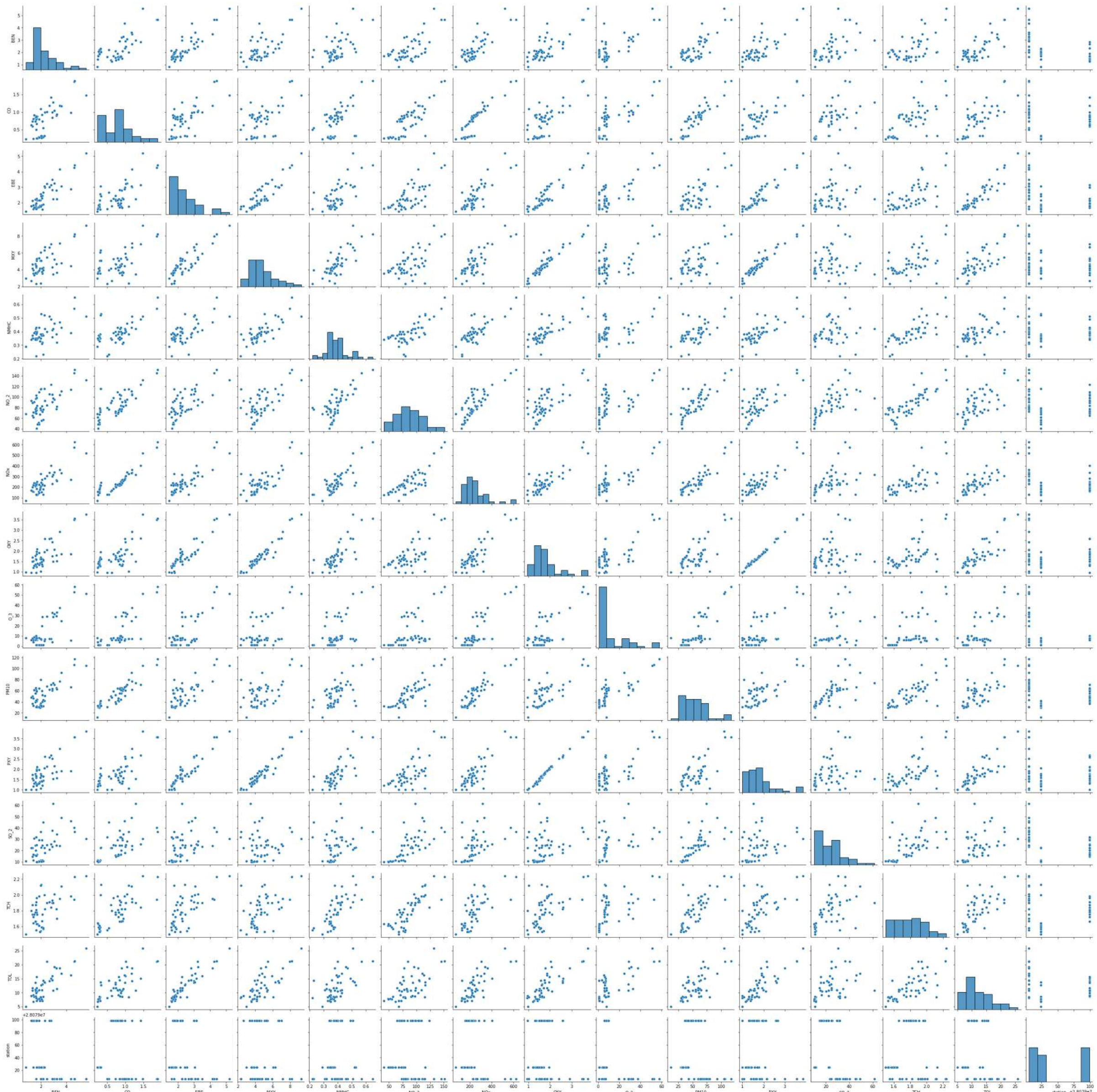
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	254
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683	1
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029	1
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000	
25%	0.450000	0.260000	0.780000	0.960000	0.160000	31.285001	
50%	0.770000	0.400000	1.000000	1.500000	0.220000	54.080002	
75%	1.390000	0.640000	1.580000	2.855000	0.300000	79.230003	1
max	30.139999	9.660000	31.680000	65.480003	2.570000	430.299988	18

In [321]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [322]:
```
sns.pairplot(df1[0:50])
```

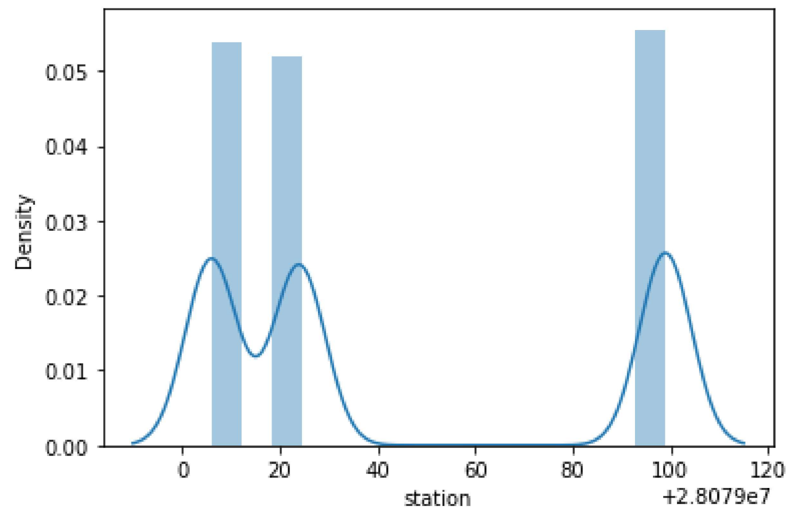
Out[322]: <seaborn.axisgrid.PairGrid at 0x1cd3f6a0370>



In [323]:
```
sns.distplot(df1['station'])
```

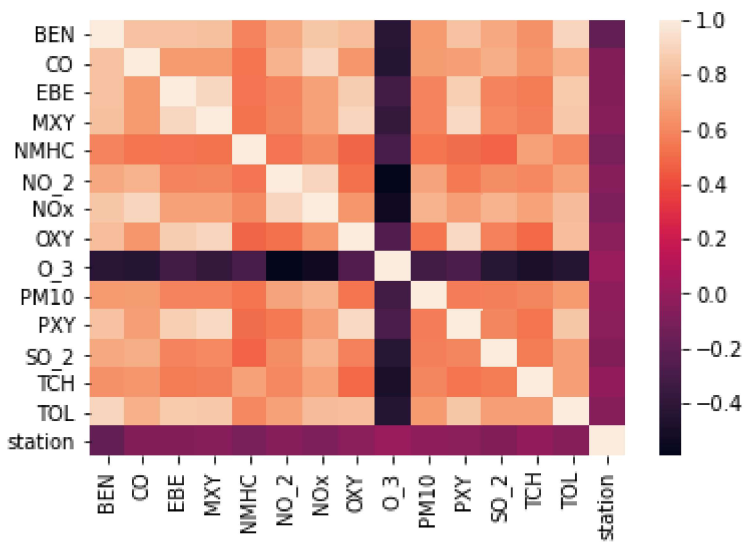
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[323]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [324]:
```
sns.heatmap(df1.corr())
```

Out[324]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [325]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [326]:
```
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 [327]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[327]:
```
LinearRegression()
```

In [328]:
```
lr.intercept_
```

Out[328]:
```
28079008.864636354
```

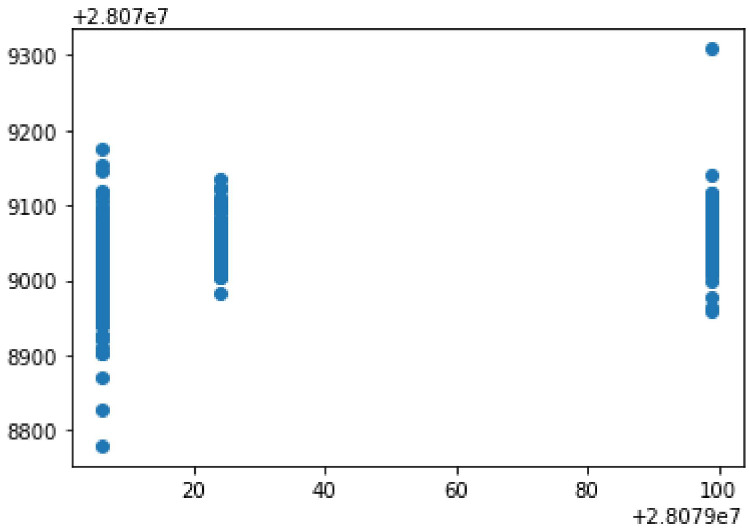
In [329]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[329]:

	Co-efficient
BEN	-32.236540
CO	18.829784
EBE	0.554598
MXY	-1.207037
NMHC	-41.467572
NO_2	0.132824
NOx	-0.054933
OXY	3.444702
O_3	-0.030696
PM10	0.148554
PXY	9.242893
SO_2	0.194962
TCH	26.206990
TOL	3.044746

In [330]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[330]: <matplotlib.collections.PathCollection at 0x1cd630c5490>



ACCURACY

In [331]:
```
lr.score(x_test,y_test)
```

Out[331]: 0.1656836298520762

In [332]:
```
lr.score(x_train,y_train)
```

Out[332]: 0.15576734872658937

Ridge and Lasso

In [333]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [334]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[334]: Ridge(alpha=10)

Accuracy(Ridge)

In [335]:
```
rr.score(x_test,y_test)
```
Out[335]: 0.16576377535882858

In [336]:
```
rr.score(x_train,y_train)
```
Out[336]: 0.15571427620227551

In [337]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[337]: Lasso(alpha=10)

In [338]:
```
la.score(x_train,y_train)
```
Out[338]: 0.013396609870542253

Accuracy(Lasso)

In [339]:
```
la.score(x_test,y_test)
```
Out[339]: 0.013655657610385008

In [340]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```
Out[340]: ElasticNet()

In [341]:
```
en.coef_
```
Out[341]:
```
array([-7.88215711,  0.        ,  0.        ,  0.        , -0.        ,
        0.06218596, -0.05698424,  0.51989087, -0.05151554,  0.16619749,
        0.58618021,  0.        ,  0.        ,  1.058272  ])
```

In [342]:
```
en.intercept_
```
Out[342]: 28079045.65718712

In [343]:
```
prediction=en.predict(x_test)
```

In [344]:
```
en.score(x_test,y_test)
```
Out[344]: 0.06828925225659888

Evaluation Metrics

In [345]:
```
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)))
```

```
36.59806709897106
1531.6968382280866
39.13689867922708
```

Logistic Regression

In [346]:
```
from sklearn.linear_model import LogisticRegression
```

In [347]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [348]:
```
feature_matrix.shape
```

Out[348]:
```
(25443, 14)
```

In [349]:
```
target_vector.shape
```

Out[349]:
```
(25443,)
```

In [350]:
```
from sklearn.preprocessing import StandardScaler
```

In [351]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [352]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[352]:
```
LogisticRegression(max_iter=10000)
```

In [353]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [354]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [355]:
```
logr.classes_
```

Out[355]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [356]:
```
logr.score(fs,target_vector)
```

Out[356]:
```
0.8146838030106512
```

In [357]:
```
logr.predict_proba(observation)[0][0]
```

Out[357]:
```
1.0827539771813 23e-19
```

In [358]:
```
logr.predict_proba(observation)
```

Out[358]:
```
array([[1.08275398e-19, 1.80383815e-19, 1.00000000e+00]])
```

Random Forest

In [359]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [360]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[360]:
```
RandomForestClassifier()
```

In [361]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [362]:
```
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[362]:
```
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 [363]:
```
grid_search.best_score_
```

Out[363]:
```
0.8251544076361594
```

In [364]:
```
rfc_best=grid_search.best_estimator_
```

In [365]:
```
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',
```

```
alue = [0, 41, 0]\nclass = b'),
 Text(1370.5263157894735, 1268.4, 'OXY <= 0.365\ngini = 0.573\nsamples = 23
89\nvalue = [606, 2140, 960]\nclass = b'),
 Text(1096.421052631579, 906.0, 'NOx <= 18.98\ngini = 0.172\nsamples = 127
\nvalue = [193, 18, 2]\nclass = a'),
 Text(1018.1052631578947, 543.5999999999999, 'gini = 0.5\nsamples = 7\nvalu
e = [5, 5, 0]\nclass = a'),
 Text(1174.7368421052631, 543.5999999999999, 'TCH <= 1.385\ngini = 0.138\ns
amples = 120\nvalue = [188, 13, 2]\nclass = a'),
 Text(1096.421052631579, 181.19999999999982, 'gini = 0.063\nsamples = 109\n
value = [179, 4, 2]\nclass = a'),
 Text(1253.0526315789473, 181.19999999999982, 'gini = 0.5\nsamples = 11\nva
lue = [9, 9, 0]\nclass = a'),
 Text(1644.6315789473683, 906.0, 'NMHC <= 0.225\ngini = 0.542\nsamples = 22
62\nvalue = [413, 2122, 958]\nclass = b'),
 Text(1488.0, 543.5999999999999, 'SO_2 <= 5.225\ngini = 0.613\nsamples = 10
71\nvalue = [331, 487, 869]\nclass = c'),
 Text(1409.6842105263156, 181.19999999999982, 'gini = 0.381\nsamples = 270
\nvalue = [63, 321, 34]\nclass = b'),
 Text(1566.3157894736842, 181.19999999999982, 'gini = 0.505\nsamples = 801
```

Conclusion

Accuracy

Linear Regression:0.16331457098631952

Ridge Regression:0.16317654437433604

Lasso Regression:0.013732764982463452

ElasticNet Regression:0.0693172677037851

Logistic Regression:0.8146838030106512

Random Forest:0.8748413156376227

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

In [ ]:

```

```