Importing Libraries

In [366]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [367]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2008.
df
```

Out[367]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.889
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.040
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.270
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.850
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.160
...	...	...	...	...	...	...	...	...	...	...	
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5.450
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15.020
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17.540
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11.910
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12.690

226392 rows × 17 columns

Data Cleaning and Data Preprocessing

In [368]:
```
df=df.dropna()
```

In [369]:
```
df.columns
```

Out[369]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [370]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [371]:
```
data=df[['CO' ,'station']]
data
```

Out[371]:

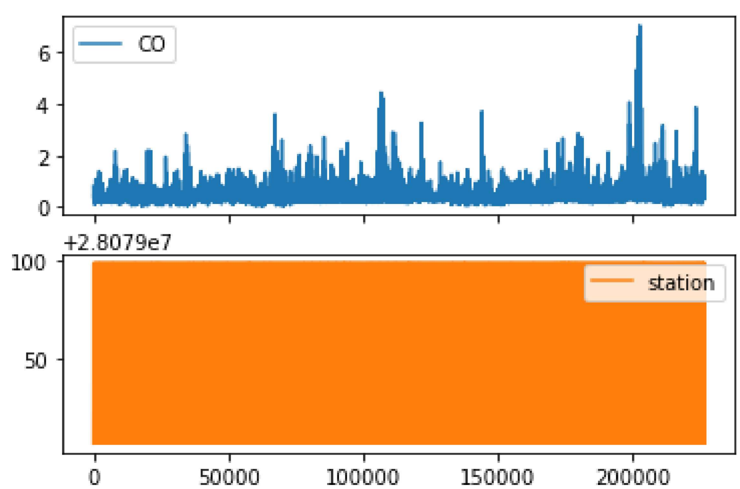
	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...	...	...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

Line chart

In [372]:
```
data.plot.line(subplots=True)
```

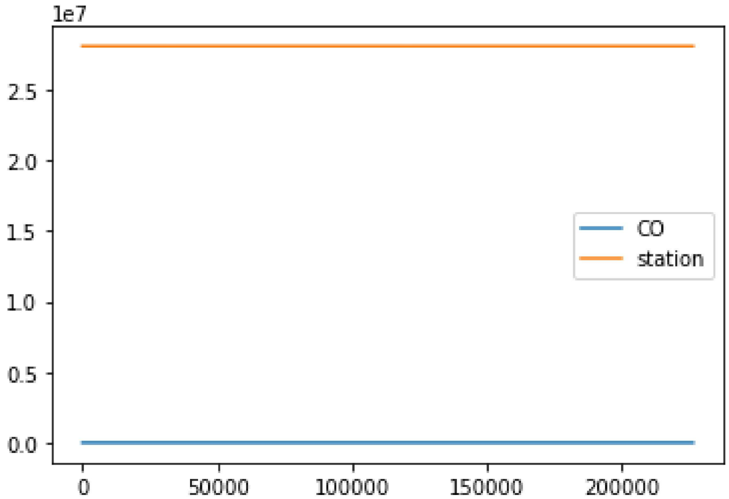
Out[372]:
```
array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

In [373]:
```
data.plot.line()
```

Out[373]: <AxesSubplot:>

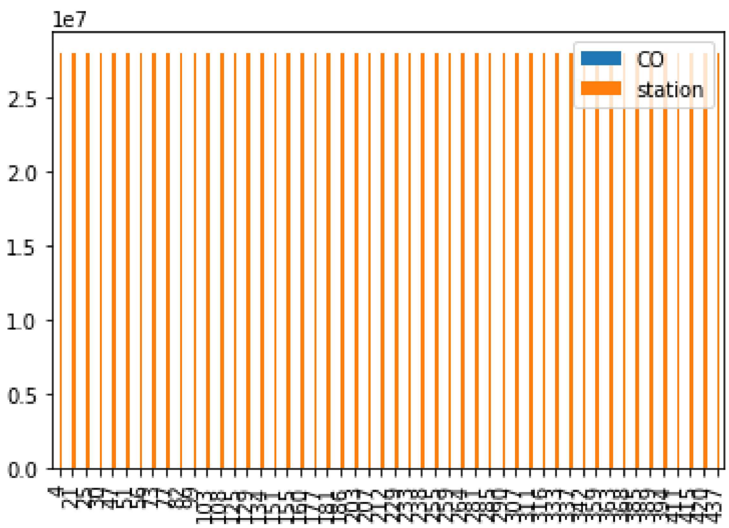


Bar chart

In [374]:
```
b=data[0:50]
```

In [375]:
```
b.plot.bar()
```

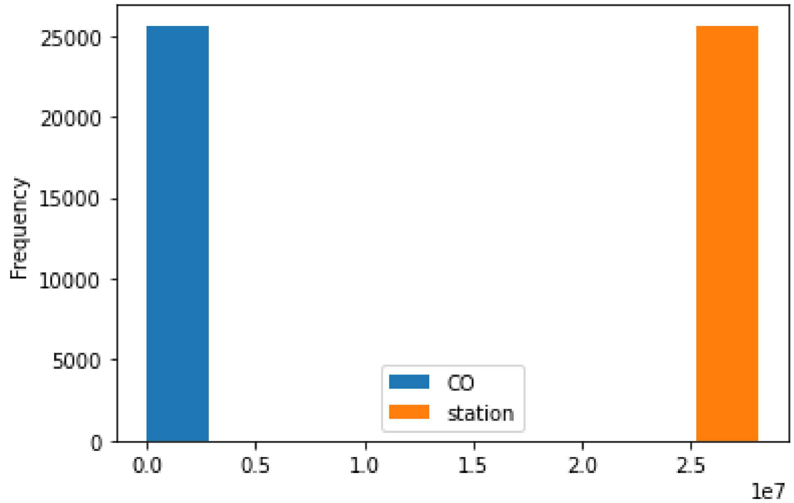
Out[375]: <AxesSubplot:>



Histogram

In [376]:
```
data.plot.hist()
```

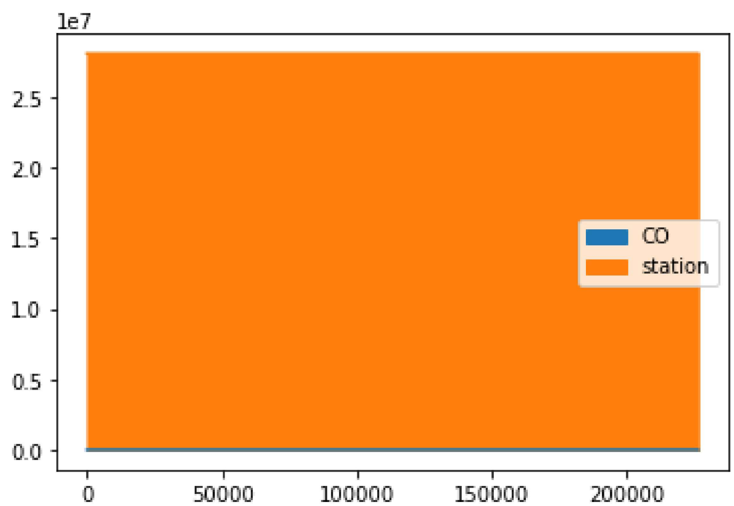
Out[376]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [377]:
```
data.plot.area()
```

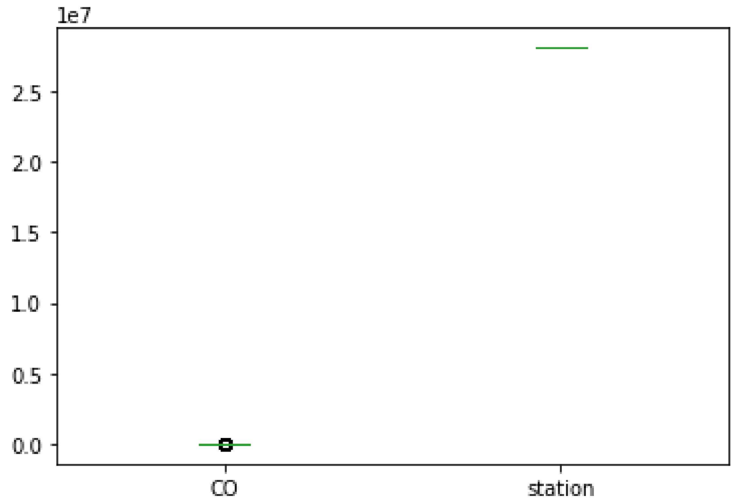
Out[377]: <AxesSubplot:>



Box chart

In [378]:
```
data.plot.box()
```

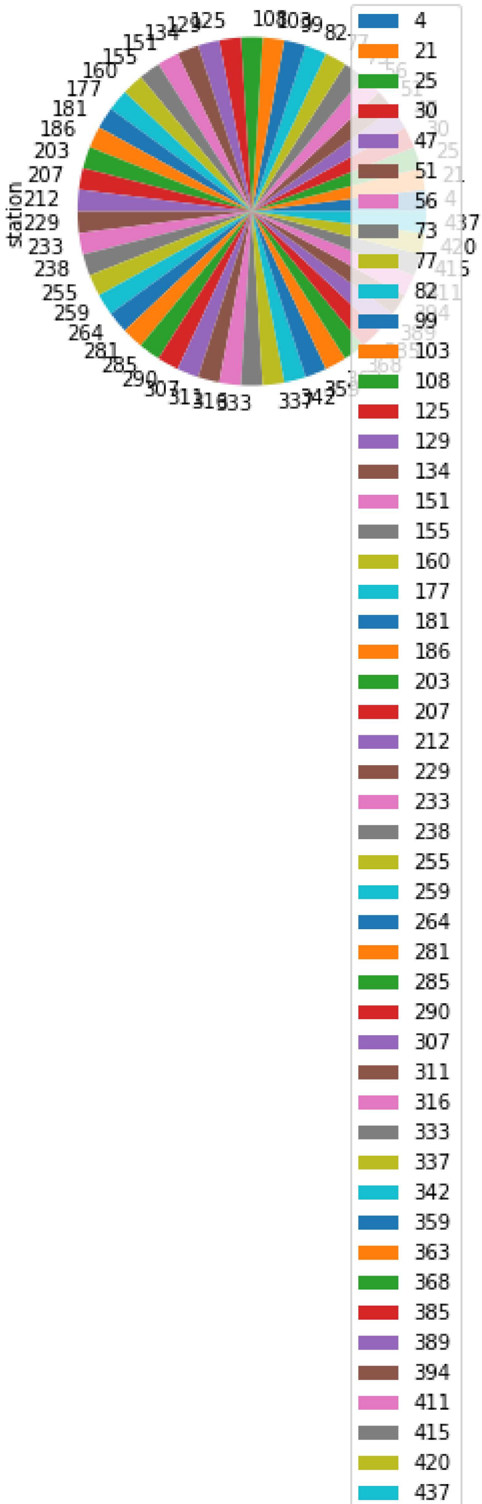
Out[378]: <AxesSubplot:>



Pie chart

In [379]:
```
b.plot.pie(y='station' )
```

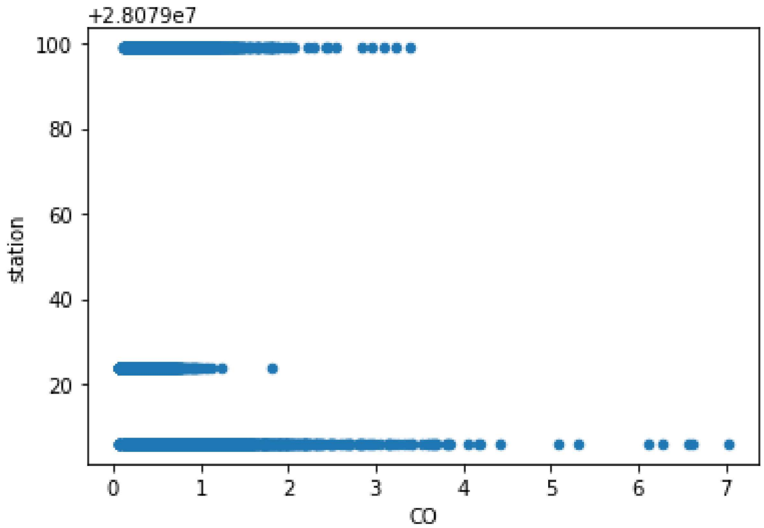
Out[379]: <AxesSubplot:ylabel='station'>



Scatter chart

In [380]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[380]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [381]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     25631 non-null  object 
 1   BEN      25631 non-null  float64
 2   CO       25631 non-null  float64
 3   EBE      25631 non-null  float64
 4   MXY      25631 non-null  float64
 5   NMHC     25631 non-null  float64
 6   NO_2     25631 non-null  float64
 7   NOx      25631 non-null  float64
 8   OXY      25631 non-null  float64
 9   O_3      25631 non-null  float64
 10  PM10     25631 non-null  float64
 11  PM25     25631 non-null  float64
 12  PXY      25631 non-null  float64
 13  SO_2     25631 non-null  float64
 14  TCH      25631 non-null  float64
 15  TOL      25631 non-null  float64
 16  station  25631 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

In [382]:
```
df.describe()
```

Out[382]:

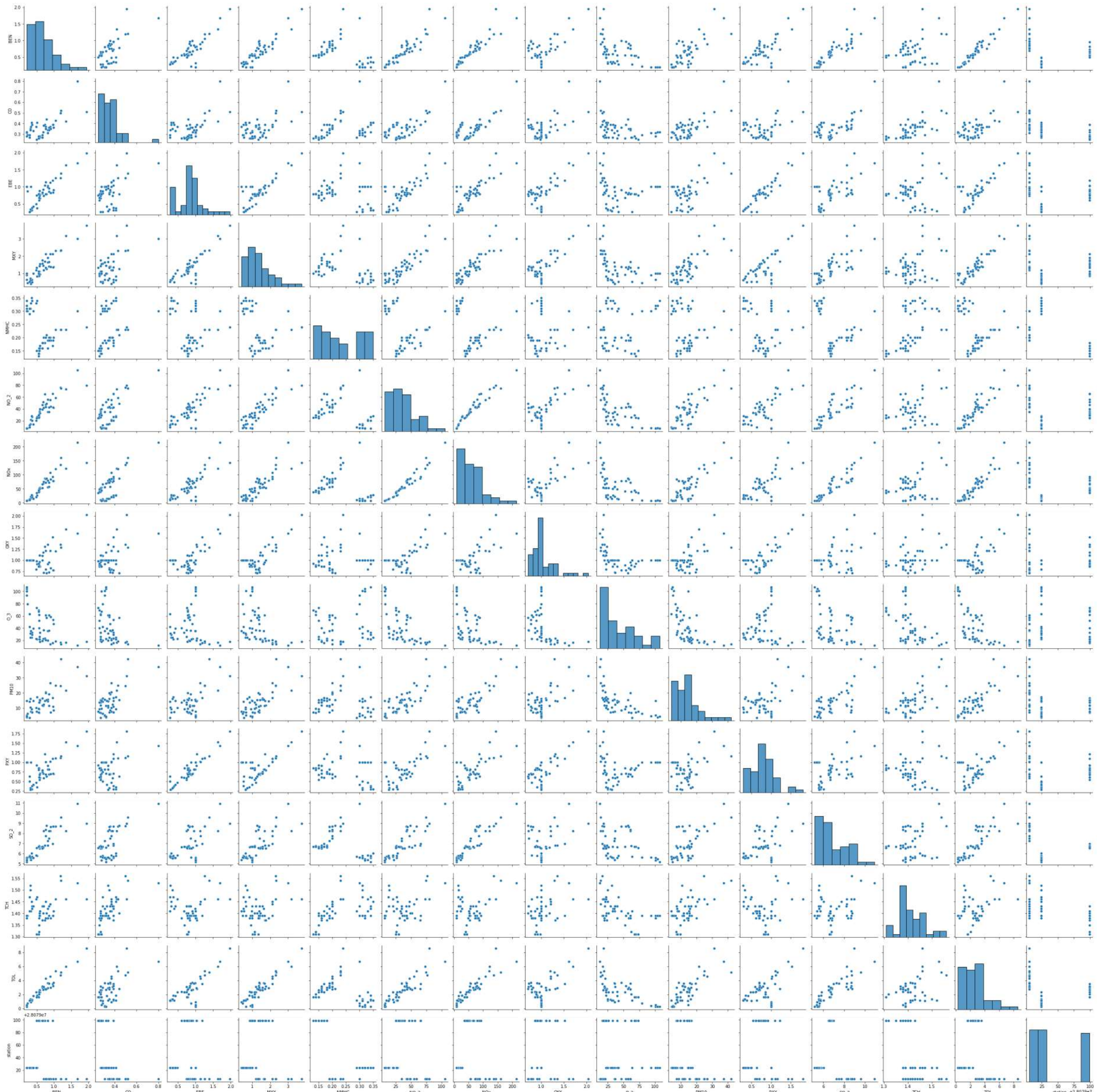
	BEN	CO	EBE	MXY	NMHC	NO_2
count	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024

In [383]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [384]:
```
sns.pairplot(df1[0:50])
```

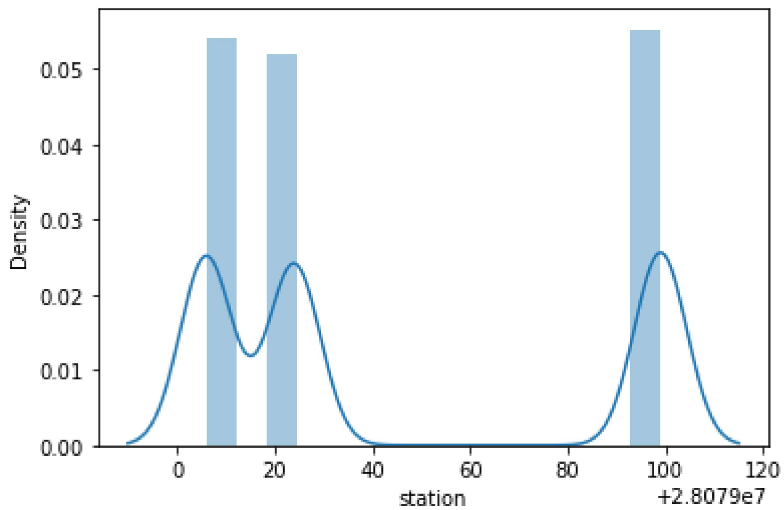
Out[384]: <seaborn.axisgrid.PairGrid at 0x1cd52e96c40>



In [385]:
```
sns.distplot(df1['station'])
```

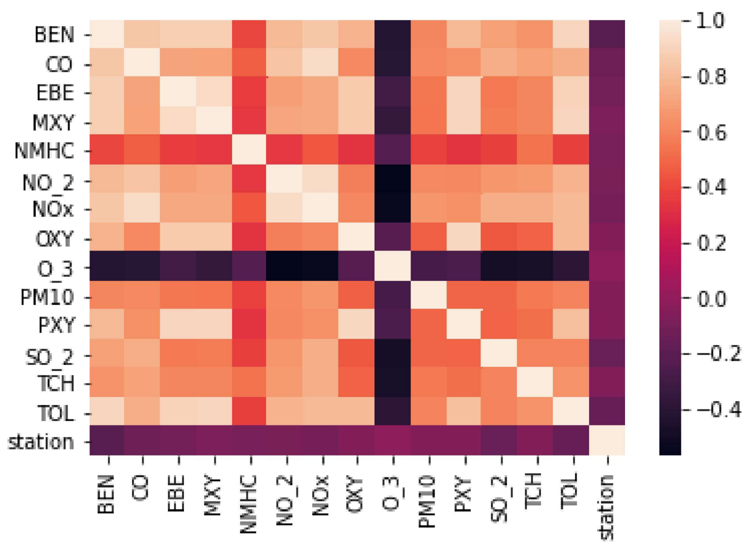
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[385]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [386]:
```
sns.heatmap(df1.corr())
```

Out[386]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [387]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [388]:
```
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 [389]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[389]:
```
LinearRegression()
```

In [390]:
```
lr.intercept_
```

Out[390]:
```
28079031.39809577
```

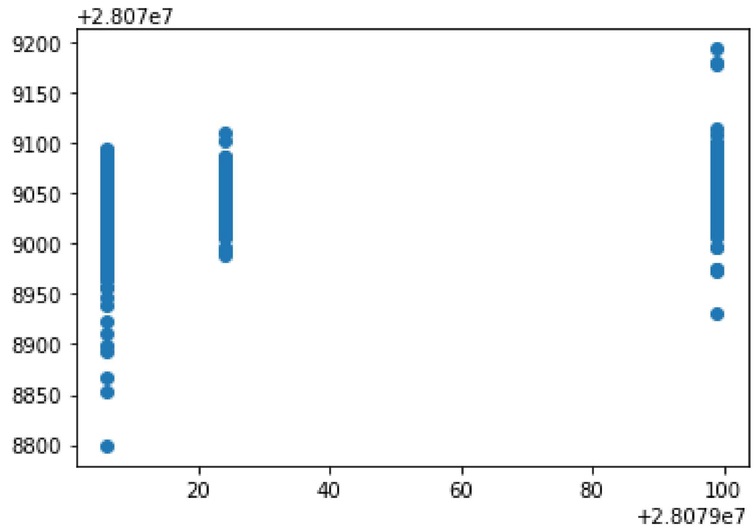
In [391]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[391]:

	Co-efficient
BEN	-25.793483
CO	-0.705893
EBE	-2.014298
MXY	7.261478
NMHC	-27.411242
NO_2	-0.042552
NOx	0.130355
OXY	3.685991
O_3	-0.135068
PM10	0.118036
PXY	2.888176
SO_2	-0.623197
TCH	21.034469
TOL	-1.681766

In [392]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[392]: <matplotlib.collections.PathCollection at 0x1cd50f69ac0>



ACCURACY

In [393]:
```
lr.score(x_test,y_test)
```

Out[393]: 0.1435527227143537

In [394]:
```
lr.score(x_train,y_train)
```

Out[394]: 0.1428576695160011

Ridge and Lasso

In [395]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [396]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[396]: Ridge(alpha=10)

Accuracy(Ridge)

In [397]:
```
rr.score(x_test,y_test)
```

Out[397]: 0.14367650207423788

In [398]:
```
rr.score(x_train,y_train)
```

Out[398]: 0.14282962972482038

In [399]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[399]: Lasso(alpha=10)

In [400]:
```
la.score(x_train,y_train)
```

Out[400]: 0.04189346138551475

Accuracy(Lasso)

In [401]:
```
la.score(x_test,y_test)
```

Out[401]: 0.03848991470257501

In [402]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[402]: ElasticNet()

In [403]:
```
en.coef_
```

Out[403]:
```
array([-4.71133639, -0.        , -0.        ,  3.22226583, -0.        ,
        0.05134131,  0.03069928,  1.35211976, -0.15695137,  0.11110857,
        1.48807403, -0.98618759,  0.        , -2.42383951])
```

In [404]:
```
en.intercept_
```

Out[404]: 28079058.635058858

In [405]:
```
prediction=en.predict(x_test)
```

In [406]:
```
en.score(x_test,y_test)
```

Out[406]: 0.09425292630981419

Evaluation Metrics

In [407]:
```
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)))
```

```
35.588080879383185
1467.3729550091664
38.306304376814616
```

Logistic Regression

In [408]:
```
from sklearn.linear_model import LogisticRegression
```

In [409]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [410]:
```
feature_matrix.shape
```

Out[410]:
```
(25631, 14)
```

In [411]:
```
target_vector.shape
```

Out[411]:
```
(25631,)
```

In [412]:
```
from sklearn.preprocessing import StandardScaler
```

In [413]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [414]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[414]:
```
LogisticRegression(max_iter=10000)
```

In [415]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [416]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [417]:
```
logr.classes_
```

Out[417]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [418]:
```
logr.score(fs,target_vector)
```

Out[418]:
```
0.794194530061254
```

In [419]:
```
logr.predict_proba(observation)[0][0]
```

Out[419]:
```
8.321803242555043e-09
```

In [420]:
```
logr.predict_proba(observation)
```

Out[420]:
```
array([[8.32180324e-09, 1.19114634e-13, 9.99999992e-01]])
```

Random Forest

In [421]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [422]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[422]:
```
RandomForestClassifier()
```

In [423]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [*]:
```
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)
```

In [*]:
```
grid_search.best_score_
```

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

In [*]:
```
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',
```

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