Importing Libraries

In [428]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

In [490]:
```
df=pd.read_csv(r"C:\Users\user\Desktop\csvs_per_year\csvs_per_year\madrid_2009.
df
```

Out[490]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.2600
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.5800
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.1900
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.5300
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.7600
...	...	...	...	...	...	...	...	...	...	...	
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.8300
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.9200
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.4600
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.0300
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.3600

215688 rows × 17 columns

Data Cleaning and Data Preprocessing

In [430]:
```
df=df.dropna()
```

In [431]:
```
df.columns
```

Out[431]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [432]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

In [433]:
```
data=df[['CO' ,'station']]
data
```

Out[433]:

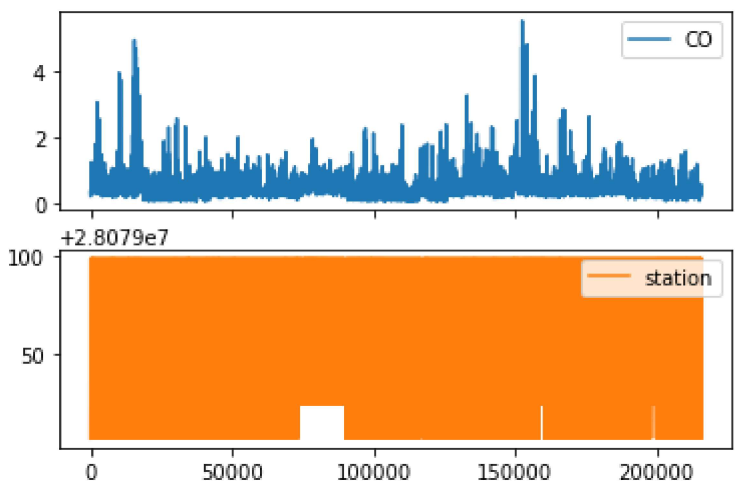
	CO	station
3	0.33	28079006
20	0.32	28079024
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...	...	...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

Line chart

In [434]:
```
data.plot.line(subplots=True)
```

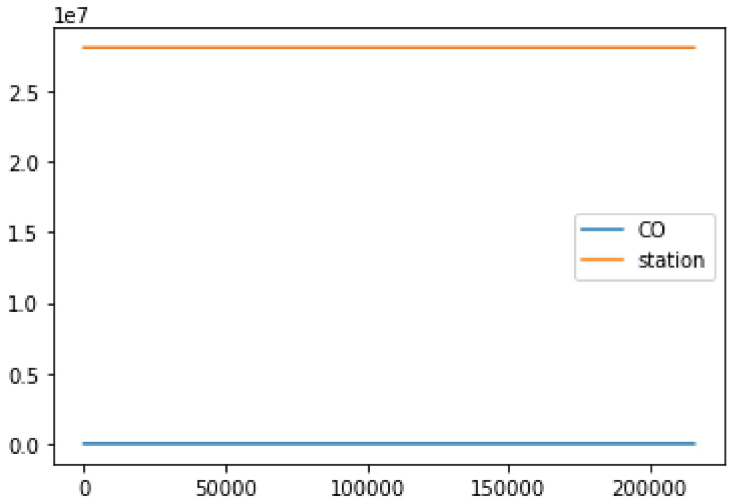
Out[434]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [435]:
```
data.plot.line()
```

Out[435]: <AxesSubplot:>

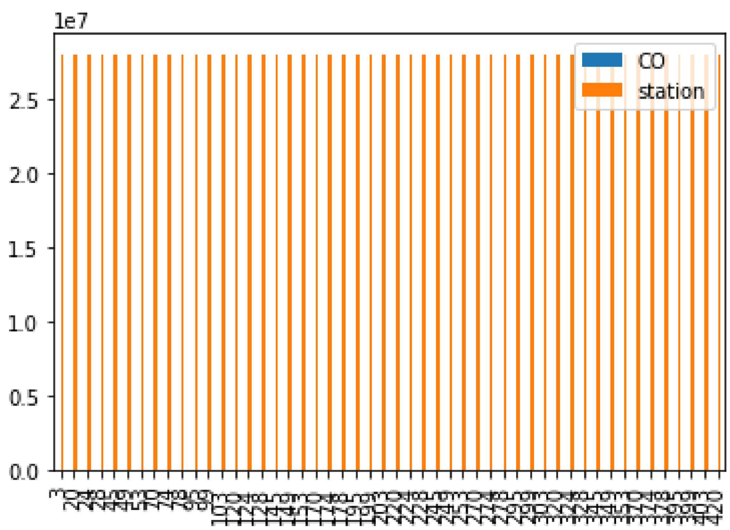


Bar chart

In [436]:
```
b=data[0:50]
```

In [437]:
```
b.plot.bar()
```

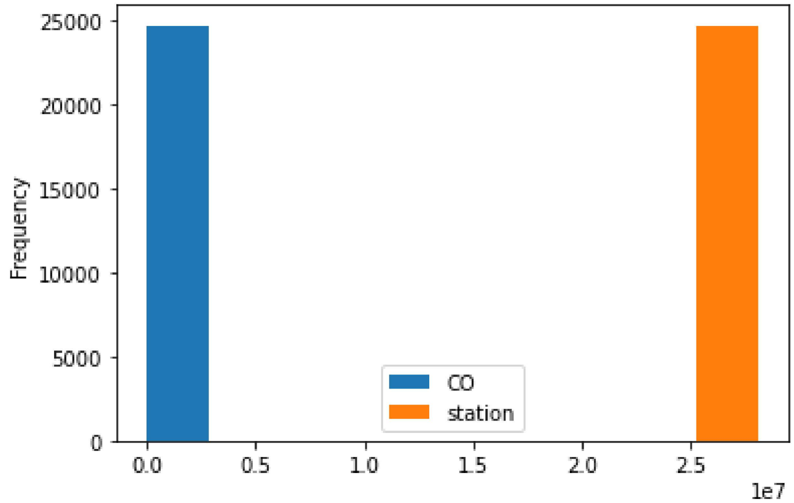
Out[437]: <AxesSubplot:>



Histogram

In [438]:
```
data.plot.hist()
```

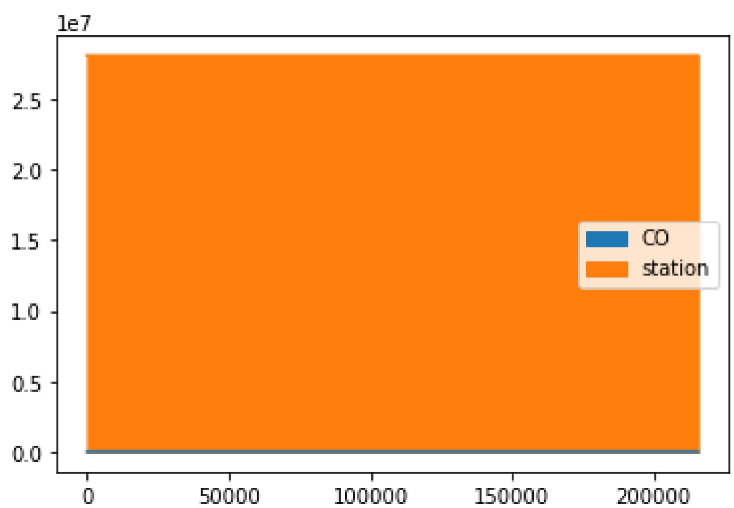
Out[438]: <AxesSubplot:ylabel='Frequency'>



Area chart

In [439]:
```
data.plot.area()
```

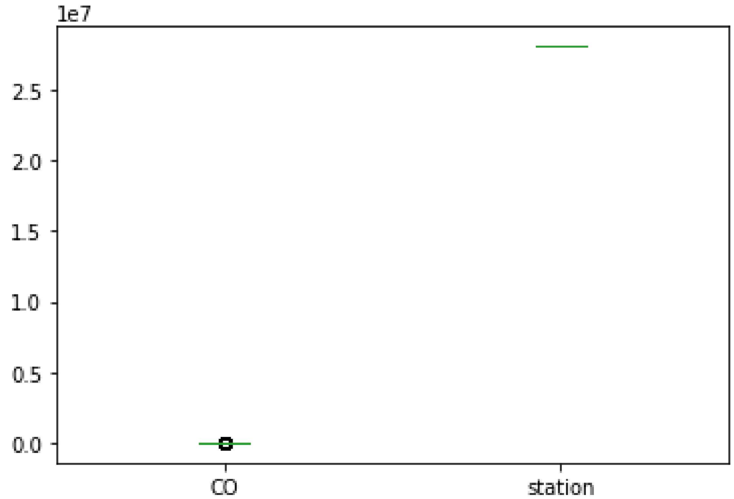
Out[439]: <AxesSubplot:>



Box chart

In [440]:
```
data.plot.box()
```

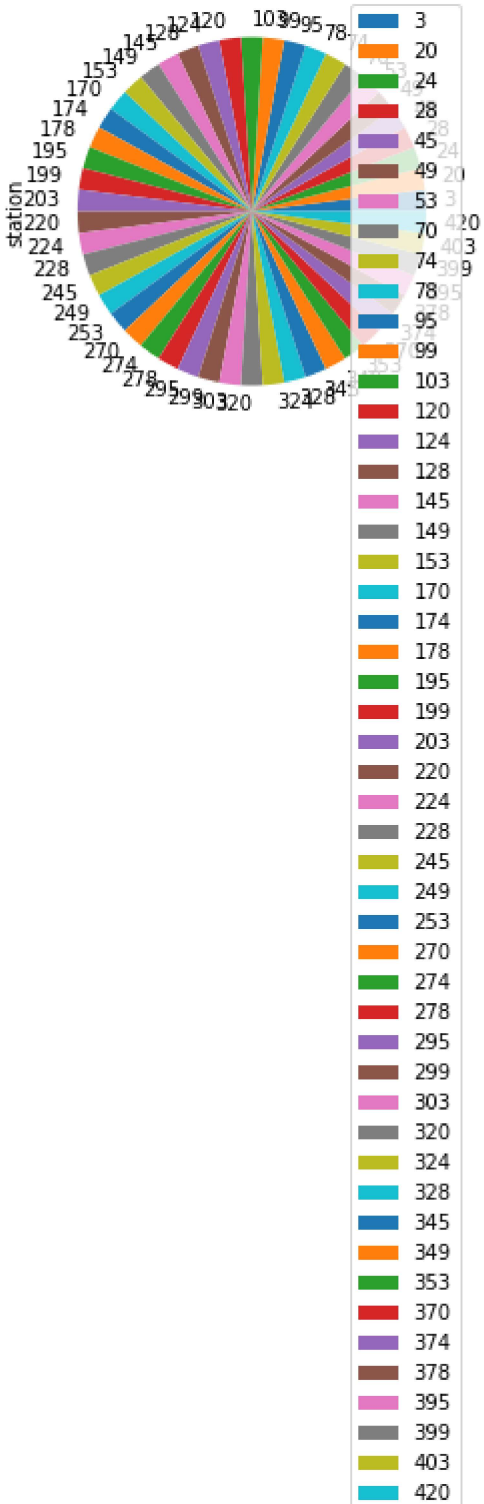
Out[440]: <AxesSubplot:>



Pie chart

In [441]:
```
b.plot.pie(y='station' )
```

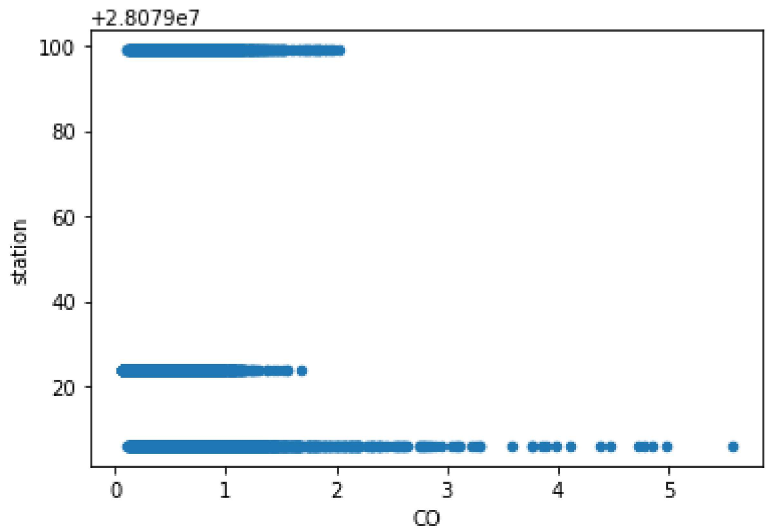
Out[441]: <AxesSubplot:ylabel='station'>



Scatter chart

In [442]:
```
data.plot.scatter(x='CO' ,y='station')
```

Out[442]: <AxesSubplot:xlabel='CO', ylabel='station'>



In [443]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     24717 non-null  object 
 1   BEN      24717 non-null  float64
 2   CO       24717 non-null  float64
 3   EBE      24717 non-null  float64
 4   MXY      24717 non-null  float64
 5   NMHC     24717 non-null  float64
 6   NO_2     24717 non-null  float64
 7   NOx      24717 non-null  float64
 8   OXY      24717 non-null  float64
 9   O_3      24717 non-null  float64
 10  PM10     24717 non-null  float64
 11  PM25     24717 non-null  float64
 12  PXY      24717 non-null  float64
 13  SO_2     24717 non-null  float64
 14  TCH      24717 non-null  float64
 15  TOL      24717 non-null  float64
 16  station  24717 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

In [444]:
```
df.describe()
```

Out[444]:

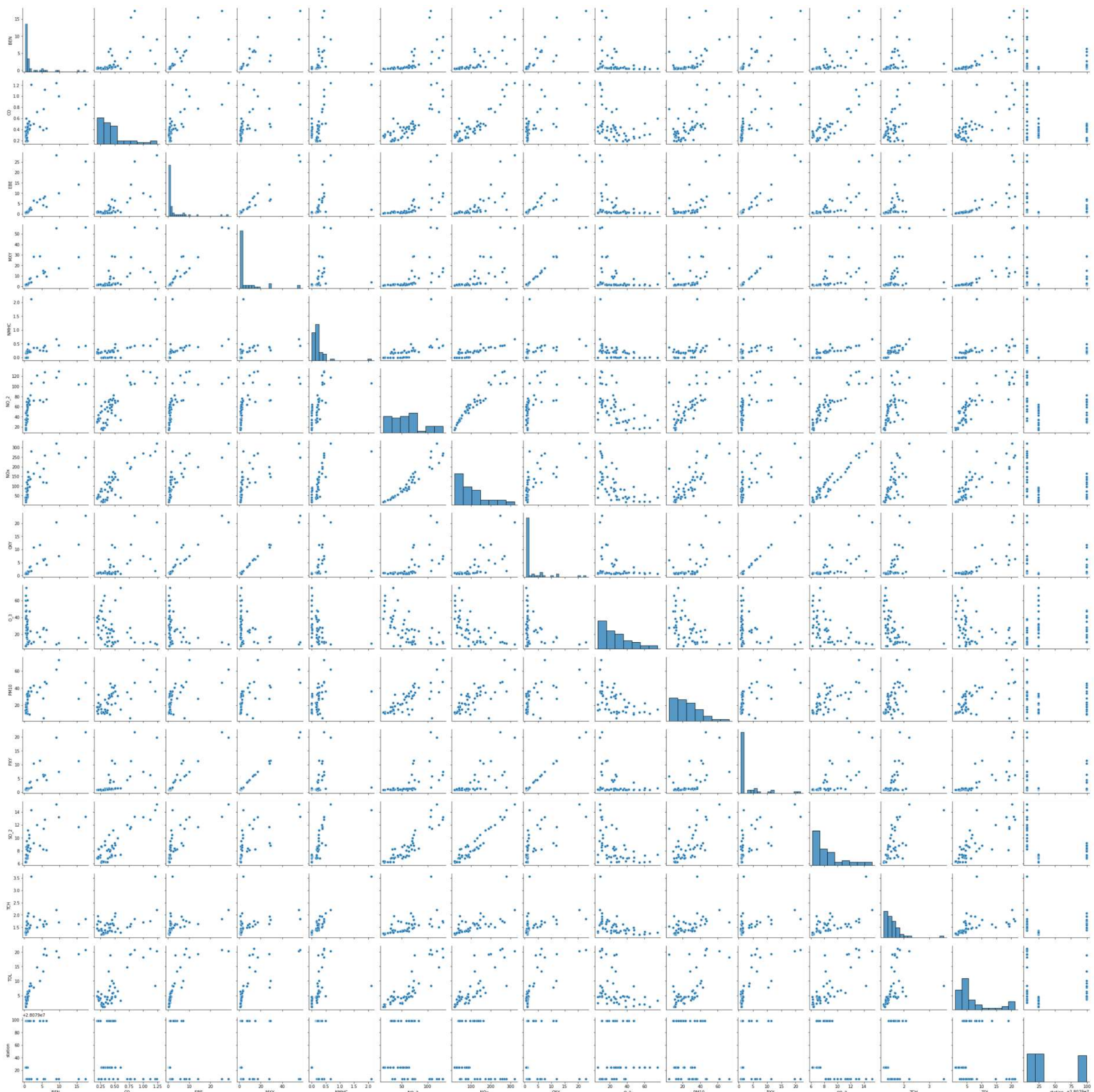
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	247
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929	[illegible]
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677	[illegible]
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000	
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000	[illegible]
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000	[illegible]
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997	1[illegible]
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994	14[illegible]

In [445]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

In [446]:
```
sns.pairplot(df1[0:50])
```

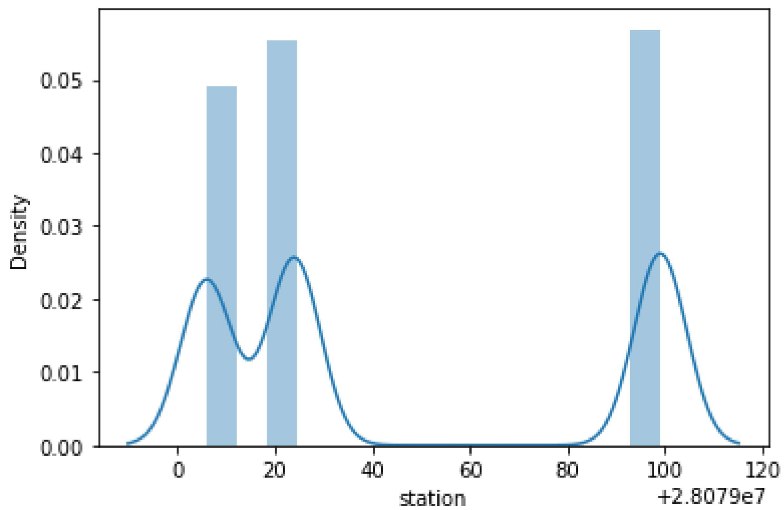
Out[446]: <seaborn.axisgrid.PairGrid at 0x1cd915c6730>



In [447]:
```
sns.distplot(df1['station'])
```

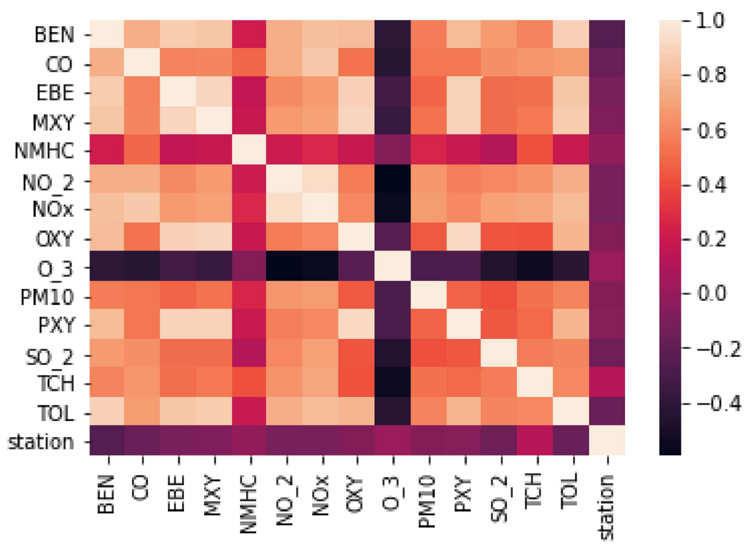
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[447]: <AxesSubplot:xlabel='station', ylabel='Density'>



In [448]:
```
sns.heatmap(df1.corr())
```

Out[448]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [449]:
```
x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

In [450]:
```
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 [451]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[451]: LinearRegression()

In [452]:
```
lr.intercept_
```

Out[452]: 28078891.75325078

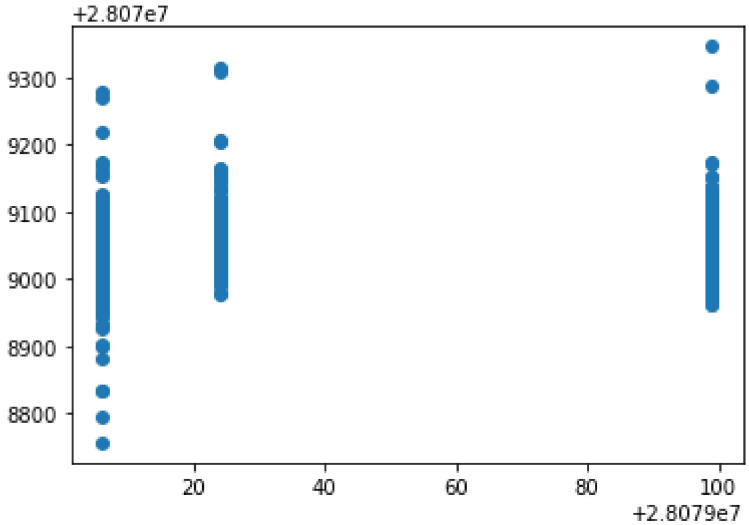
In [453]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[453]:

	Co-efficient
BEN	-37.039973
CO	-30.754126
EBE	7.531596
MXY	-1.633098
NMHC	-18.185915
NO_2	-0.183246
NOx	0.207947
OXY	14.764541
O_3	0.024889
PM10	-0.051958
PXY	2.828355
SO_2	-0.302665
TCH	124.896713
TOL	-1.148805

In [454]:
```
prediction =lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[454]: <matplotlib.collections.PathCollection at 0x1cd630e3070>



ACCURACY

In [455]:
```
lr.score(x_test,y_test)
```

Out[455]: 0.26828993596392614

In [456]:
```
lr.score(x_train,y_train)
```

Out[456]: 0.2946269921837963

Ridge and Lasso

In [457]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [458]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[458]: Ridge(alpha=10)

Accuracy(Ridge)

In [459]:
```
rr.score(x_test,y_test)
```
Out[459]: 0.2700143791476979

In [460]:
```
rr.score(x_train,y_train)
```
Out[460]: 0.2942743131240403

In [461]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[461]: Lasso(alpha=10)

In [462]:
```
la.score(x_train,y_train)
```
Out[462]: 0.033961539134792496

Accuracy(Lasso)

In [463]:
```
la.score(x_test,y_test)
```
Out[463]: 0.03863380766863844

In [464]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```
Out[464]: ElasticNet()

In [465]:
```
en.coef_
```
Out[465]:
```
array([-7.07079106, -0.68564137,  0.42393327,  2.12561565, -0.        ,
       -0.24862351,  0.13491165,  1.24030818, -0.13651085,  0.08671355,
        1.92554874, -0.71980373,  1.47185506, -1.99024297])
```

In [466]:
```
en.intercept_
```
Out[466]: 28079063.075263444

In [467]:
```
prediction=en.predict(x_test)
```

In [468]:
```
en.score(x_test,y_test)
```
Out[468]: 0.11209585600575955

Evaluation Metrics

In [469]:
```
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)))
```

```
35.84169726560099
1460.280829285163
38.21362099154126
```

Logistic Regression

In [470]:
```
from sklearn.linear_model import LogisticRegression
```

In [471]:
```
feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

In [472]:
```
feature_matrix.shape
```

Out[472]:
```
(24717, 14)
```

In [473]:
```
target_vector.shape
```

Out[473]:
```
(24717,)
```

In [474]:
```
from sklearn.preprocessing import StandardScaler
```

In [475]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [476]:
```
logr=LogisticRegression(max_iter=10000)
logr.fit(fs,target_vector)
```

Out[476]:
```
LogisticRegression(max_iter=10000)
```

In [477]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

In [478]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [479]:
```
logr.classes_
```

Out[479]:
```
array([28079006, 28079024, 28079099], dtype=int64)
```

In [480]:
```
logr.score(fs,target_vector)
```

Out[480]:
```
0.8951733624630821
```

In [481]:
```
logr.predict_proba(observation)[0][0]
```

Out[481]:
```
5.447205522232353e-13
```

In [482]:
```
logr.predict_proba(observation)
```

Out[482]:
```
array([[5.44720552e-13, 8.28692830e-44, 1.00000000e+00]])
```

Random Forest

In [483]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [484]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[484]:
```
RandomForestClassifier()
```

In [485]:
```
parameters={'max_depth':[1,2,3,4,5],
 'min_samples_leaf':[5,10,15,20,25],
 'n_estimators':[10,20,30,40,50]
}
```

In [486]:
```
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[486]:
```
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 [487]:
```
grid_search.best_score_
```

Out[487]:
```
0.9042250653638224
```

In [488]:
```
rfc_best=grid_search.best_estimator_
```

In [489]:
```
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',
```

```
 Text(692.6896551724138, 181.19999999999982, 'gini = 0.516\nsamples = 68\nv
alue = [12, 28, 71]\nclass = c'),
 Text(846.6206896551724, 181.19999999999982, 'gini = 0.069\nsamples = 58\nv
alue = [0, 81, 3]\nclass = b'),
 Text(1077.5172413793105, 543.5999999999999, 'NMHC <= 0.155\ngini = 0.102\n
samples = 72\nvalue = [1, 89, 4]\nclass = b'),
 Text(1000.5517241379312, 181.19999999999982, 'gini = 0.258\nsamples = 22\n
value = [1, 29, 4]\nclass = b'),
 Text(1154.4827586206898, 181.19999999999982, 'gini = 0.0\nsamples = 50\nva
lue = [0, 60, 0]\nclass = b'),
 Text(1731.7241379310346, 1268.4, 'MXY <= 2.145\ngini = 0.579\nsamples = 74
\nvalue = [11, 51, 56]\nclass = c'),
 Text(1539.3103448275863, 906.0, 'SO_2 <= 6.345\ngini = 0.492\nsamples = 47
\nvalue = [9, 15, 49]\nclass = c'),
 Text(1385.3793103448277, 543.5999999999999, 'TCH <= 1.285\ngini = 0.245\ns
amples = 10\nvalue = [2, 12, 0]\nclass = b'),
 Text(1308.4137931034484, 181.19999999999982, 'gini = 0.0\nsamples = 5\nval
ue = [0, 9, 0]\nclass = b'),
 Text(1462.344827586207, 181.19999999999982, 'gini = 0.48\nsamples = 5\nval
```

Conclusion

Accuracy

Linear Regression:0.16331457098631952

Ridge Regression:0.16317654437433604

Lasso Regression:0.013732764982463452

ElasticNet Regression:0.0693172677037851

Logistic Regression:0.8951733624630821

Random Forest:0.8911047204272553

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

In [ ]:

```

```