

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(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\ma
df
```

Out[2]:

	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
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000
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
...
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
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN
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

225120 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: 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 [8]: data=df[['station', 'TCH', 'TOL']]  
data
```

Out[8]:

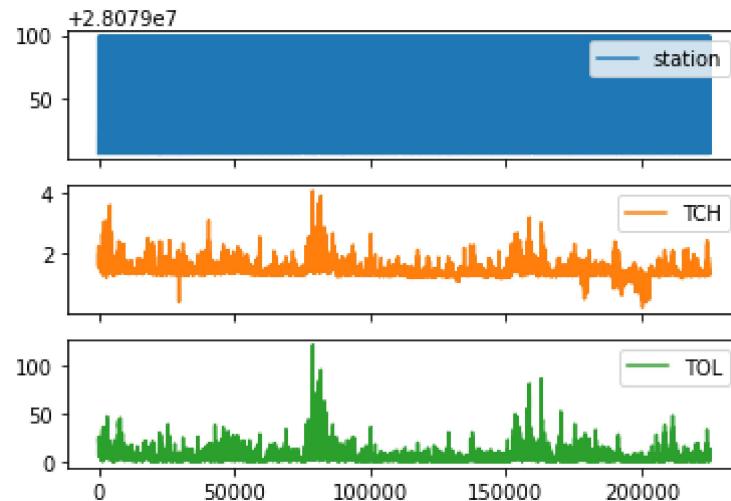
	station	TCH	TOL
4	28079006	1.94	21.200001
21	28079024	1.54	8.440000
25	28079099	1.84	15.010000
30	28079006	2.23	21.330000
47	28079024	1.53	8.400000
...
225073	28079006	1.28	7.850000
225094	28079099	1.33	3.340000
225098	28079006	1.28	4.560000
225115	28079024	1.44	0.510000
225119	28079099	1.32	2.410000

25443 rows × 3 columns

Line chart

```
In [9]: data.plot.line(subplots=True)
```

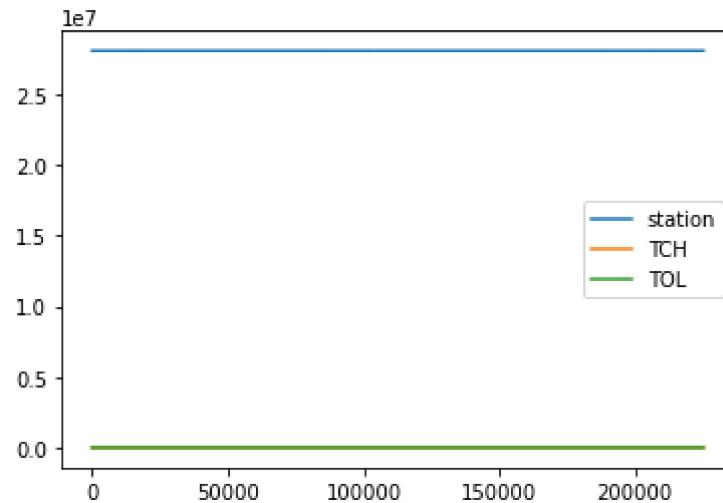
Out[9]: array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

In [10]: `data.plot.line()`

Out[10]: <AxesSubplot:>

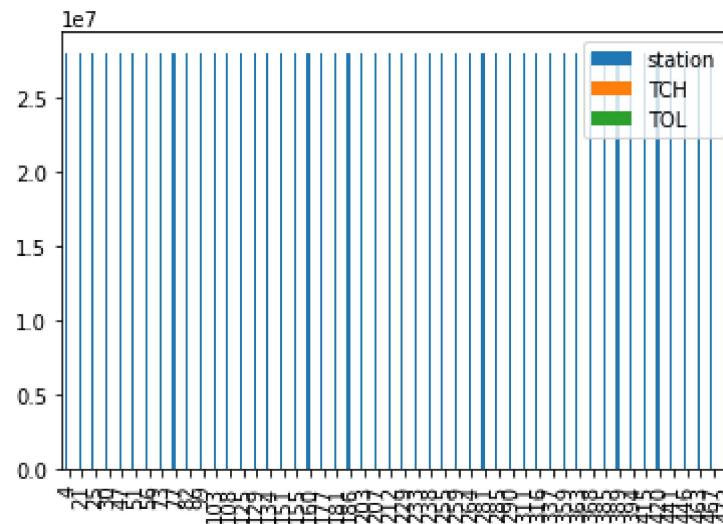


Bar chart

In [11]: `b=data[0:50]`

In [12]: `b.plot.bar()`

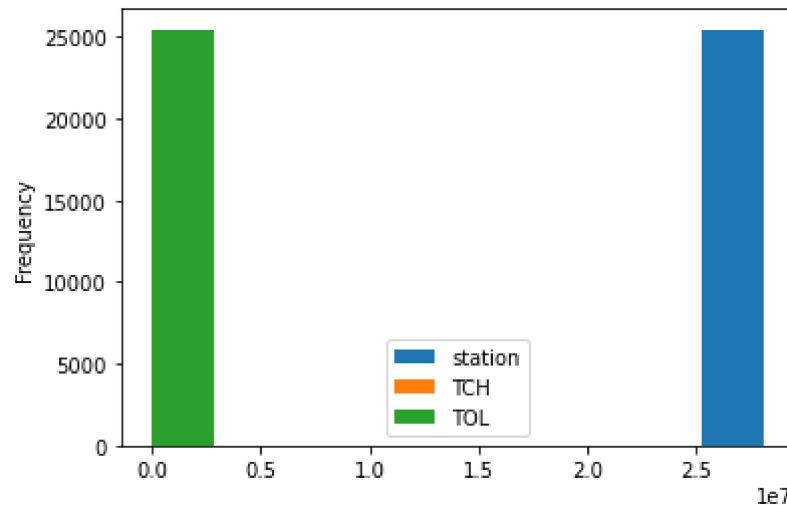
Out[12]: <AxesSubplot:>



Histogram

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

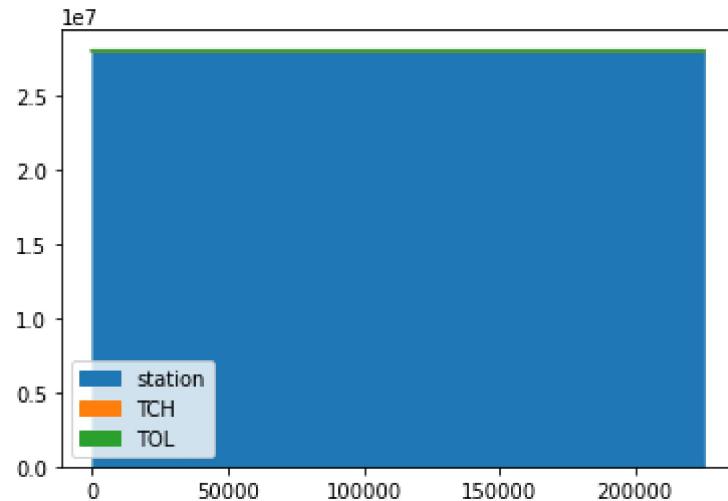
```
Out[13]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

```
In [14]: data.plot.area()
```

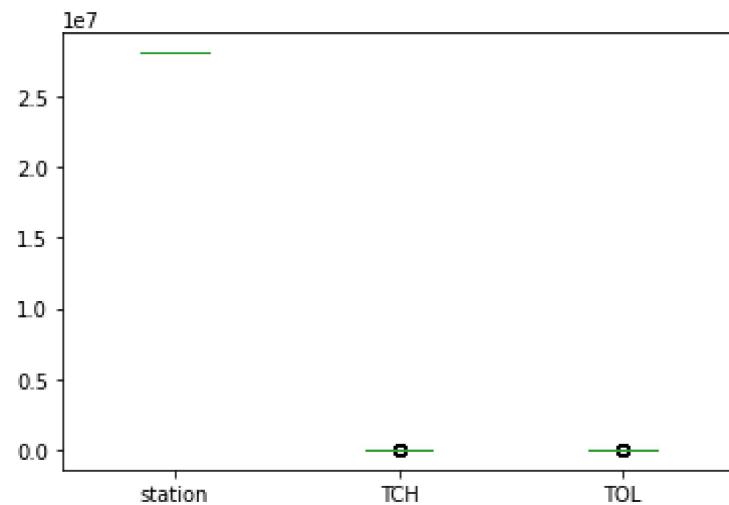
```
Out[14]: <AxesSubplot:>
```



Box chart

In [15]: `data.plot.box()`

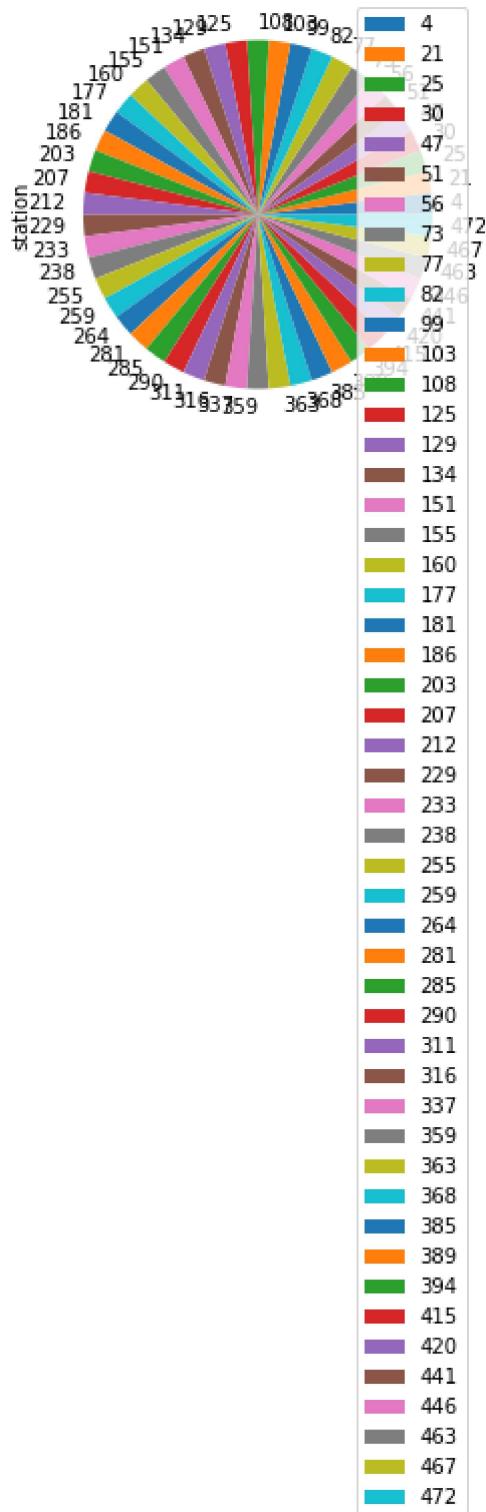
Out[15]: <AxesSubplot:>



Pie chart

```
In [17]: b.plot.pie(y='station' )
```

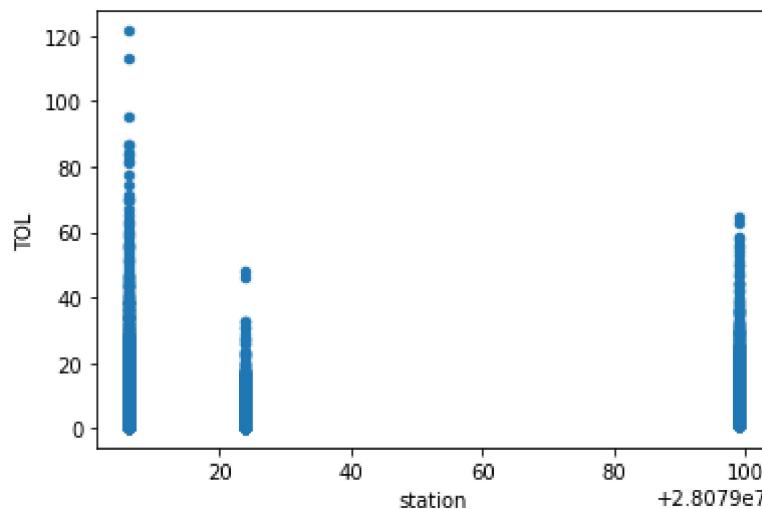
```
Out[17]: <AxesSubplot:ylabel='station'>
```



Scatter chart

```
In [18]: data.plot.scatter(x='station' ,y='TOL')
```

```
Out[18]: <AxesSubplot:xlabel='station', ylabel='TOL'>
```



```
In [19]: 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 [20]: df.describe()

Out[20]:

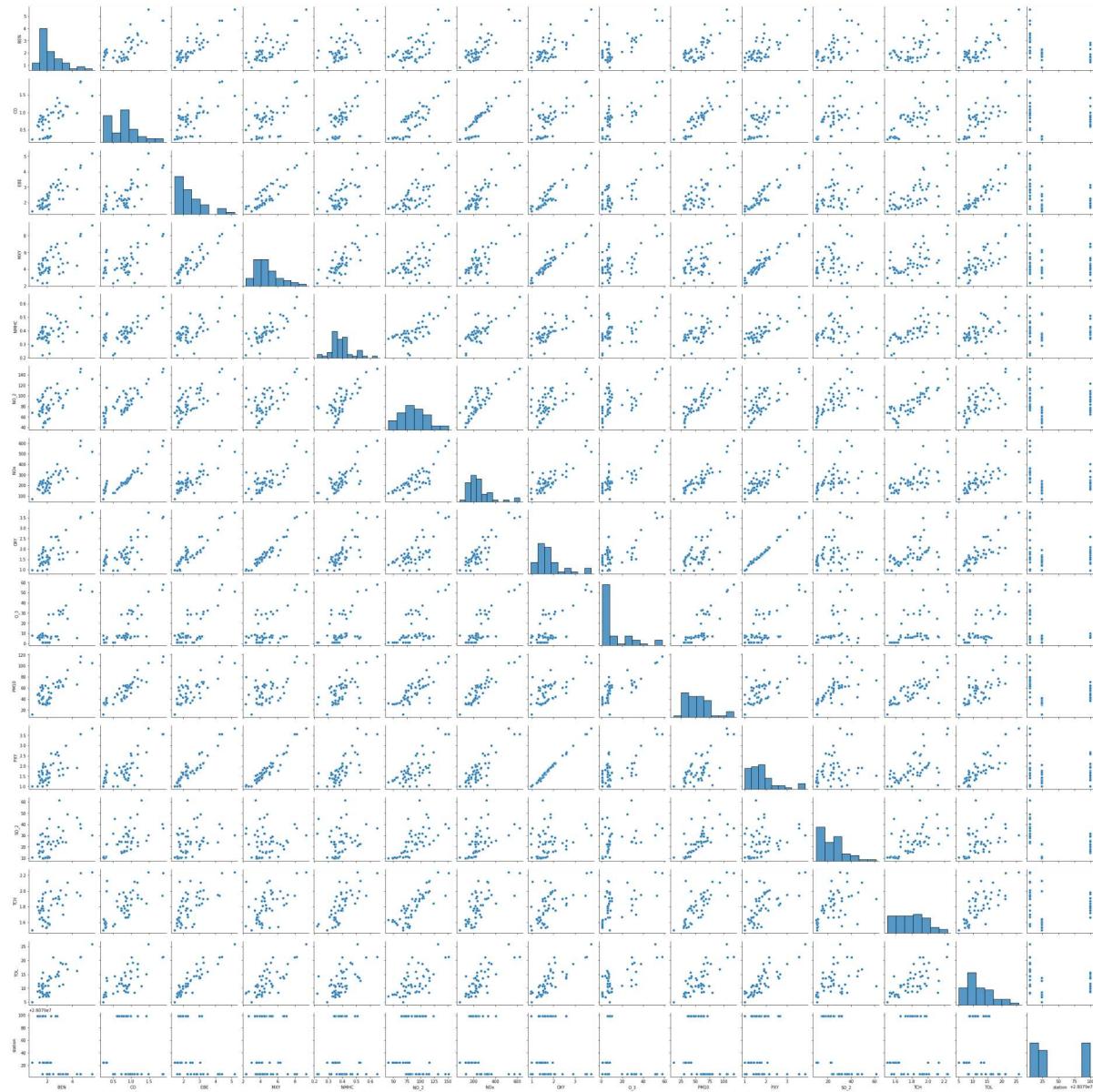
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	254
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683	1
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029	1
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000	
25%	0.450000	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 [21]: df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3', 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]

EDA AND VISUALIZATION

```
In [22]: sns.pairplot(df1[0:50])
```

Out[22]: <seaborn.axisgrid.PairGrid at 0x220851bf4f0>

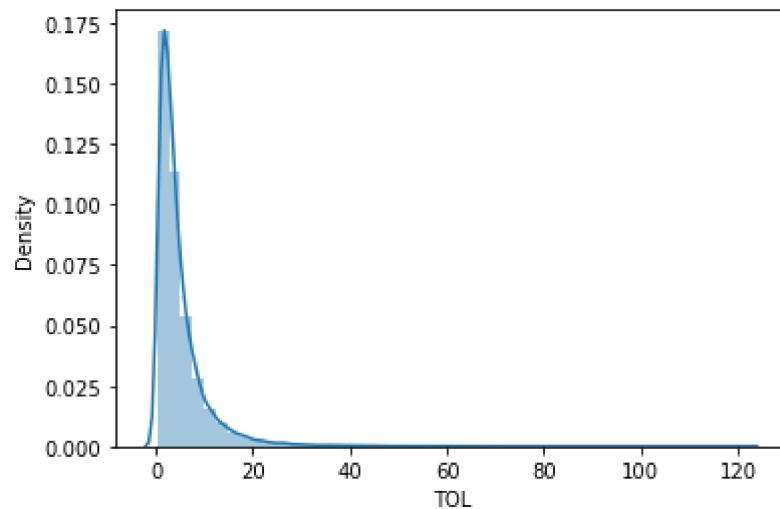


In [23]: `sns.distplot(df1['TOL'])`

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).

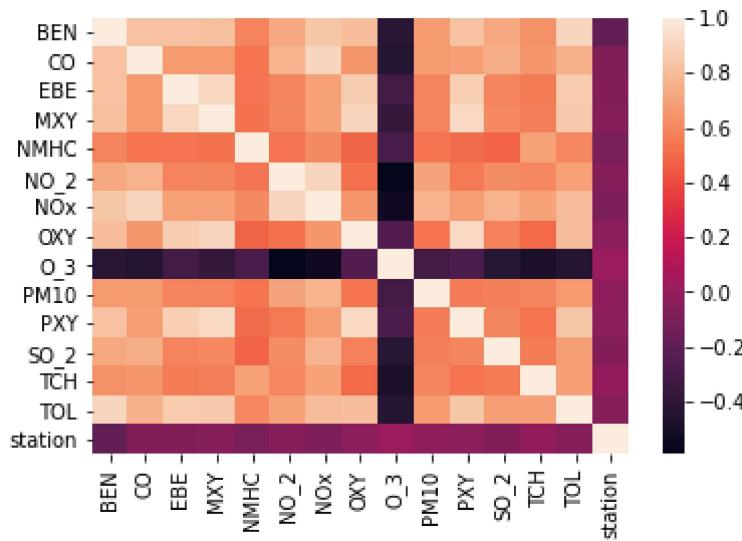
```
warnings.warn(msg, FutureWarning)
```

Out[23]: <AxesSubplot:xlabel='TOL', ylabel='Density'>



In [24]: `sns.heatmap(df1.corr())`

Out[24]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BUILDING

```
In [25]: x=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
           'PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]
y=df['station']
```

```
In [26]: 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 [27]: from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[27]: LinearRegression()

```
In [28]: lr.intercept_
```

Out[28]: 28079009.445346206

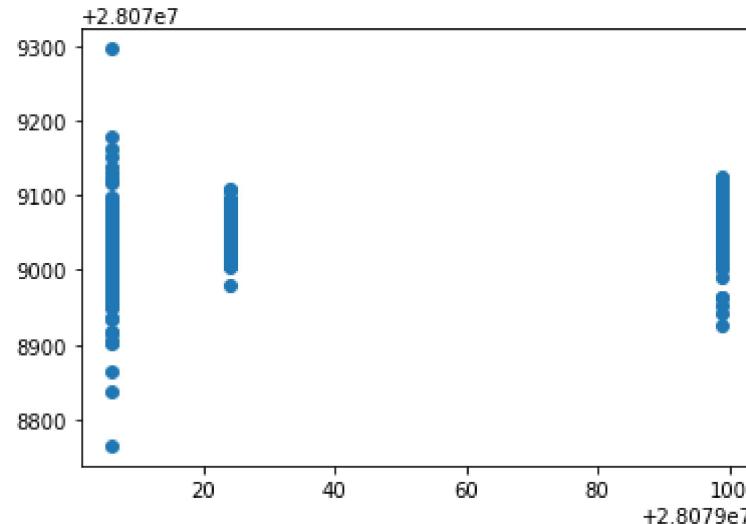
```
In [29]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[29]:

	Co-efficient
BEN	-33.295002
CO	18.460482
EBE	0.804154
MXY	-1.028983
NMHC	-39.893319
NO_2	0.107691
NOx	-0.033843
OXY	5.073755
O_3	-0.026383
PM10	0.148810
PXY	7.137987
SO_2	0.184490
TCH	25.579256
TOL	3.013003

```
In [30]: prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

```
Out[30]: <matplotlib.collections.PathCollection at 0x22094aff3a0>
```



ACCURACY

```
In [31]: lr.score(x_test,y_test)
```

```
Out[31]: 0.15658034817295796
```

```
In [32]: lr.score(x_train,y_train)
```

```
Out[32]: 0.16040101066333845
```

Ridge and Lasso

```
In [33]: from sklearn.linear_model import Ridge,Lasso
```

```
In [34]: rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
Out[34]: Ridge(alpha=10)
```

Accuracy(Ridge)

```
In [35]: rr.score(x_test,y_test)
```

```
Out[35]: 0.15651166203018474
```

```
In [36]: rr.score(x_train,y_train)
```

```
Out[36]: 0.1603509695403328
```

```
In [37]: la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

```
Out[37]: Lasso(alpha=10)
```

```
In [38]: la.score(x_test,y_test)
```

```
Out[38]: 0.011348336385610946
```

Accuracy(Lasso)

```
In [39]: la.score(x_train,y_train)
```

```
Out[39]: 0.014633893408507292
```

Accuracy(Elastic Net)

```
In [40]: from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

```
Out[40]: ElasticNet()
```

```
In [41]: en.coef_
```

```
Out[41]: array([-7.99120661,  0.          , -0.          ,  0.09725735, -0.          ,
   0.05781774, -0.05434256,  0.61204634, -0.05206503,  0.17822584,
   0.7630782 , -0.          ,  0.          ,  0.92763063])
```

```
In [42]: en.intercept_
```

```
Out[42]: 28079045.32997434
```

```
In [43]: prediction=en.predict(x_test)
```

```
In [44]: en.score(x_test,y_test)
```

```
Out[44]: 0.06558933566008651
```

Evaluation Metrics

```
In [45]: 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)))
```

```
36.73147362190291  
1543.4894917284475  
39.28726882500803
```

Logistic Regression

```
In [46]: from sklearn.linear_model import LogisticRegression
```

```
In [47]: feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O  
PM10', 'PXY', 'SO_2', 'TCH', 'TOL']]  
target_vector=df[ 'station']
```

```
In [48]: feature_matrix.shape
```

```
Out[48]: (25443, 14)
```

```
In [49]: target_vector.shape
```

```
Out[49]: (25443,)
```

```
In [50]: from sklearn.preprocessing import StandardScaler
```

```
In [51]: fs=StandardScaler().fit_transform(feature_matrix)
```

```
In [52]: logr=LogisticRegression(max_iter=10000)  
logr.fit(fs,target_vector)
```

```
Out[52]: LogisticRegression(max_iter=10000)
```

```
In [53]: observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
```

```
In [54]: prediction=logr.predict(observation)
```

```
print(prediction)
```

```
[28079099]
```

```
In [55]: logr.classes_
```

```
Out[55]: array([28079006, 28079024, 28079099], dtype=int64)
```

```
In [56]: logr.score(fs,target_vector)
```

```
Out[56]: 0.8146838030106512
```

```
In [57]: logr.predict_proba(observation)[0][0]
```

```
Out[57]: 1.082753977181323e-19
```

```
In [58]: logr.predict_proba(observation)
```

```
Out[58]: array([[1.08275398e-19, 1.80383815e-19, 1.00000000e+00]])
```

Random Forest

```
In [59]: from sklearn.ensemble import RandomForestClassifier
```

```
In [60]: rfc=RandomForestClassifier()  
rfc.fit(x_train,y_train)
```

```
Out[60]: RandomForestClassifier()
```

```
In [61]: parameters={'max_depth':[1,2,3,4,5],  
                   'min_samples_leaf':[5,10,15,20,25],  
                   'n_estimators':[10,20,30,40,50]  
}
```

```
In [62]: 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[62]: 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 [63]: grid_search.best_score_
```

```
Out[63]: 0.8256597417181359
```

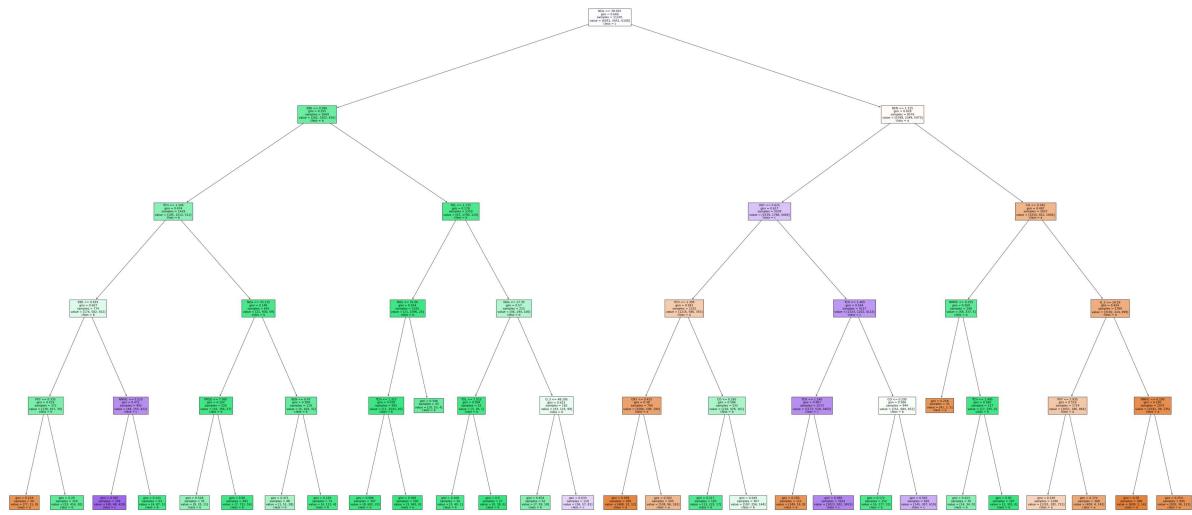
```
In [64]: rfc_best=grid_search.best_estimator_
```

```
In [65]: 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[65]: [Text(2271.8571428571427, 1993.2, 'NOx <= 38.025\ngini = 0.666\nsamples = 112  
45\nvalue = [6051, 5651, 6108]\nclass = c'),  
 Text(1175.7857142857142, 1630.8000000000002, 'EBE <= 0.995\ngini = 0.355\nsamples = 2669\nvalue = [262, 3302, 635]\nclass = b'),  
 Text(637.7142857142857, 1268.4, 'TCH <= 1.345\ngini = 0.474\nsamples = 1419\nvalue = [195, 1512, 511]\nclass = b'),  
 Text(318.85714285714283, 906.0, 'EBE <= 0.655\ngini = 0.607\nsamples = 774\nvalue = [174, 582, 452]\nclass = b'),  
 Text(159.42857142857142, 543.5999999999999, 'PXY <= 0.335\ngini = 0.419\nsamples = 372\nvalue = [130, 427, 30]\nclass = b'),  
 Text(79.71428571428571, 181.1999999999982, 'gini = 0.219\nsamples = 56\nvalue = [77, 11, 0]\nclass = a'),  
 Text(239.1428571428571, 181.1999999999982, 'gini = 0.29\nsamples = 316\nvalue = [53, 416, 30]\nclass = b'),  
 Text(478.2857142857142, 543.5999999999999, 'NMHC <= 0.215\ngini = 0.471\nsamples = 402\nvalue = [44, 155, 422]\nclass = c'),  
 Text(398.57142857142856, 181.1999999999982, 'gini = 0.345\nsamples = 339\nvalue = [40, 68, 419]\nclass = c'),  
 Text(558.0, 181.1999999999982, 'gini = 0.141\nsamples = 63\nvalue = [4, 87, 3]\nclass = b'),  
 Text(956.5714285714284, 906.0, 'NOx <= 33.115\ngini = 0.148\nsamples = 645\nvalue = [21, 930, 59]\nclass = b'),  
 Text(797.1428571428571, 543.5999999999999, 'PM10 <= 7.065\ngini = 0.102\nsamples = 526\nvalue = [16, 766, 27]\nclass = b'),  
 Text(717.4285714285713, 181.1999999999982, 'gini = 0.528\nsamples = 35\nvalue = [9, 35, 11]\nclass = b'),  
 Text(876.8571428571428, 181.1999999999982, 'gini = 0.06\nsamples = 491\nvalue = [7, 731, 16]\nclass = b'),  
 Text(1116.0, 543.5999999999999, 'BEN <= 0.47\ngini = 0.308\nsamples = 119\nvalue = [5, 164, 32]\nclass = b'),  
 Text(1036.2857142857142, 181.1999999999982, 'gini = 0.471\nsamples = 48\nvalue = [1, 51, 28]\nclass = b'),  
 Text(1195.7142857142856, 181.1999999999982, 'gini = 0.126\nsamples = 71\nvalue = [4, 113, 4]\nclass = b'),  
 Text(1713.8571428571427, 1268.4, 'TOL <= 1.175\ngini = 0.178\nsamples = 1250\nvalue = [67, 1790, 124]\nclass = b'),  
 Text(1514.5714285714284, 906.0, 'NOx <= 31.66\ngini = 0.054\nsamples = 1035\nvalue = [21, 1596, 24]\nclass = b'),  
 Text(1434.8571428571427, 543.5999999999999, 'TCH <= 1.315\ngini = 0.039\nsamples = 992\nvalue = [11, 1543, 20]\nclass = b'),  
 Text(1355.142857142857, 181.1999999999982, 'gini = 0.086\nsamples = 397\nvalue = [8, 602, 20]\nclass = b'),  
 Text(1514.5714285714284, 181.1999999999982, 'gini = 0.006\nsamples = 595\nvalue = [3, 941, 0]\nclass = b'),  
 Text(1594.2857142857142, 543.5999999999999, 'gini = 0.348\nsamples = 43\nvalue = [10, 53, 4]\nclass = b'),  
 Text(1913.1428571428569, 906.0, 'NOx <= 17.35\ngini = 0.57\nsamples = 215\nvalue = [46, 194, 100]\nclass = b'),  
 Text(1753.7142857142856, 543.5999999999999, 'TOL <= 1.515\ngini = 0.094\nsamples = 53\nvalue = [3, 78, 1]\nclass = b'),  
 Text(1673.999999999998, 181.1999999999982, 'gini = 0.168\nsamples = 26\nvalue = [3, 40, 1]\nclass = b'),  
 Text(1833.4285714285713, 181.1999999999982, 'gini = 0.0\nsamples = 27\nvalue = [0, 38, 0]\nclass = b'),  
 Text(2072.5714285714284, 543.5999999999999, 'O_3 <= 48.295\ngini = 0.623\nsamples = 162\nvalue = [43, 116, 99]\nclass = b'),  
 Text(1992.8571428571427, 181.1999999999982, 'gini = 0.454\nsamples = 52\nvalue = [11, 111, 11]\nclass = b')]
```

```
lue = [7, 59, 18]\nclass = b'),  
Text(2152.285714285714, 181.19999999999982, 'gini = 0.633\nclass = 110\nva  
lue = [36, 57, 81]\nclass = c'),  
Text(3367.928571428571, 1630.8000000000002, 'BEN <= 1.315\ngini = 0.628\nclass  
= 8576\nvalue = [5789, 2349, 5473]\nclass = a'),  
Text(2869.7142857142853, 1268.4, 'OXY <= 0.625\ngini = 0.617\nclass = 5539  
\nvalue = [2539, 1788, 4469]\nclass = c'),  
Text(2550.8571428571427, 906.0, 'TCH <= 1.395\ngini = 0.581\nclass = 1322  
\nvalue = [1216, 586, 355]\nclass = a'),  
Text(2391.428571428571, 543.5999999999999, 'OXY <= 0.415\ngini = 0.38\nclass  
= 790\nvalue = [1006, 108, 194]\nclass = a'),  
Text(2311.7142857142853, 181.19999999999982, 'gini = 0.094\nclass = 285\nva  
lue = [468, 12, 12]\nclass = a'),  
Text(2471.142857142857, 181.19999999999982, 'gini = 0.502\nclass = 505\nva  
lue = [538, 96, 182]\nclass = a'),  
Text(2710.285714285714, 543.5999999999999, 'CO <= 0.265\ngini = 0.586\nclass  
= 532\nvalue = [210, 478, 161]\nclass = b'),  
Text(2630.5714285714284, 181.19999999999982, 'gini = 0.217\nclass = 165\nva  
lue = [13, 222, 17]\nclass = b'),  
Text(2790.0, 181.19999999999982, 'gini = 0.649\nclass = 367\nvalue = [197,  
256, 144]\nclass = b'),  
Text(3188.5714285714284, 906.0, 'TCH <= 1.465\ngini = 0.544\nclass = 4217  
\nvalue = [1323, 1202, 4114]\nclass = c'),  
Text(3029.142857142857, 543.5999999999999, 'TCH <= 1.245\ngini = 0.487\nclass  
= 3273\nvalue = [1172, 518, 3462]\nclass = c'),  
Text(2949.428571428571, 181.19999999999982, 'gini = 0.256\nclass = 111\nva  
lue = [149, 16, 9]\nclass = a'),  
Text(3108.8571428571427, 181.19999999999982, 'gini = 0.466\nclass = 3162\nva  
lue = [1023, 502, 3453]\nclass = c'),  
Text(3347.999999999995, 543.5999999999999, 'CO <= 0.295\ngini = 0.586\nclass  
= 944\nvalue = [151, 684, 652]\nclass = b'),  
Text(3268.285714285714, 181.19999999999982, 'gini = 0.172\nclass = 252\nva  
lue = [6, 377, 33]\nclass = b'),  
Text(3427.7142857142853, 181.19999999999982, 'gini = 0.565\nclass = 692\nva  
lue = [145, 307, 619]\nclass = c'),  
Text(3866.142857142857, 1268.4, 'CO <= 0.345\ngini = 0.487\nclass = 3037\nva  
lue = [3250, 561, 1004]\nclass = a'),  
Text(3587.142857142857, 906.0, 'NMHC <= 0.255\ngini = 0.269\nclass = 248\nva  
lue = [58, 337, 5]\nclass = b'),  
Text(3507.428571428571, 543.5999999999999, 'gini = 0.258\nclass = 31\nva  
lue = [41, 2, 5]\nclass = a'),  
Text(3666.8571428571427, 543.5999999999999, 'TCH <= 1.495\ngini = 0.092\nclass  
= 217\nvalue = [17, 335, 0]\nclass = b'),  
Text(3587.142857142857, 181.19999999999982, 'gini = 0.413\nclass = 30\nva  
lue = [14, 34, 0]\nclass = b'),  
Text(3746.5714285714284, 181.19999999999982, 'gini = 0.02\nclass = 187\nva  
lue = [3, 301, 0]\nclass = b'),  
Text(4145.142857142857, 906.0, 'O_3 <= 14.59\ngini = 0.424\nclass = 2789\nva  
lue = [3192, 224, 999]\nclass = a'),  
Text(3985.7142857142853, 543.5999999999999, 'PXY <= 2.935\ngini = 0.519\nclass  
= 1714\nvalue = [1651, 186, 864]\nclass = a'),  
Text(3905.999999999995, 181.19999999999982, 'gini = 0.549\nclass = 1346\nva  
lue = [1201, 182, 721]\nclass = a'),  
Text(4065.428571428571, 181.19999999999982, 'gini = 0.374\nclass = 368\nva  
lue = [450, 4, 143]\nclass = a'),  
Text(4304.571428571428, 543.5999999999999, 'NMHC <= 0.195\ngini = 0.185\nclass  
= 1075\nvalue = [1541, 38, 135]\nclass = a'),
```

```
Text(4224.857142857142, 181.19999999999982, 'gini = 0.05\nsamples = 380\\nvalue = [609, 2, 14]\\nclass = a'),  
Text(4384.285714285714, 181.19999999999982, 'gini = 0.254\\nsamples = 695\\nvalue = [932, 36, 121]\\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.16040101066333845

Ridge Regression:0.1603509695403328

Lasso Regression:0.014633893408507292

ElasticNet Regression:0.06558933566008651

Logistic Regression:0.8146838030106512

Random Forest:0.8256597417181359

Random Forest is suitable for this dataset