

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	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL
0	2011-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN 2
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7 2
2	2011-11-01 01:00:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2 2
3	2011-11-01 01:00:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN 2
4	2011-11-01 01:00:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN 2
...
209923	2011-09-01 00:00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN 2
209924	2011-09-01 00:00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN 2
209925	2011-09-01 00:00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN 2
209926	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN 2
209927	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN 2

209928 rows × 14 columns

Data Cleaning and Data Preprocessing

```
In [3]: df=df.dropna()
```

```
In [4]: df.columns
```

```
Out[4]: Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

```
In [5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      16460 non-null   object 
 1   BEN        16460 non-null   float64
 2   CO         16460 non-null   float64
 3   EBE        16460 non-null   float64
 4   NMHC       16460 non-null   float64
 5   NO         16460 non-null   float64
 6   NO_2       16460 non-null   float64
 7   O_3        16460 non-null   float64
 8   PM10       16460 non-null   float64
 9   PM25       16460 non-null   float64
 10  SO_2       16460 non-null   float64
 11  TCH        16460 non-null   float64
 12  TOL        16460 non-null   float64
 13  station    16460 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

```
In [6]: data=df[['BEN', 'TOL', 'TCH']]  
data
```

Out[6]:

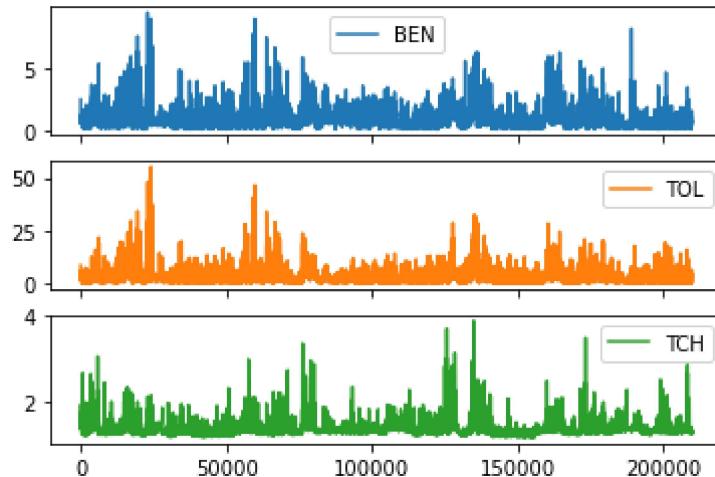
	BEN	TOL	TCH
1	2.5	8.7	1.54
6	0.7	1.7	1.36
25	1.8	7.4	1.71
30	1.0	2.9	1.40
49	1.3	6.2	1.75
...
209862	0.4	0.7	1.26
209881	0.9	4.9	1.34
209886	0.6	0.9	1.26
209905	0.6	3.8	1.32
209910	0.7	0.9	1.25

16460 rows × 3 columns

Line chart

```
In [7]: data.plot.line(subplots=True)
```

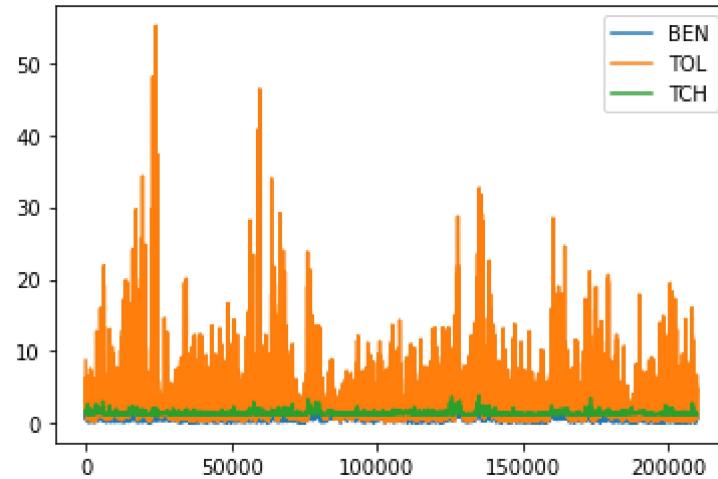
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

```
In [8]: data.plot.line()
```

```
Out[8]: <AxesSubplot:>
```

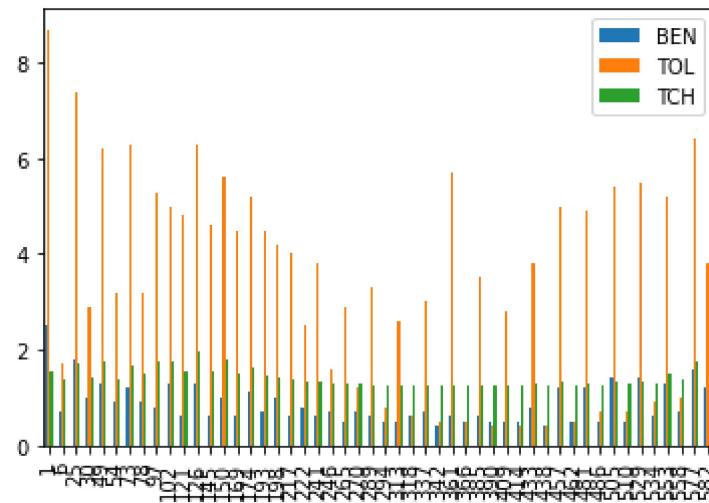


Bar chart

```
In [9]: b=data[0:50]
```

```
In [10]: b.plot.bar()
```

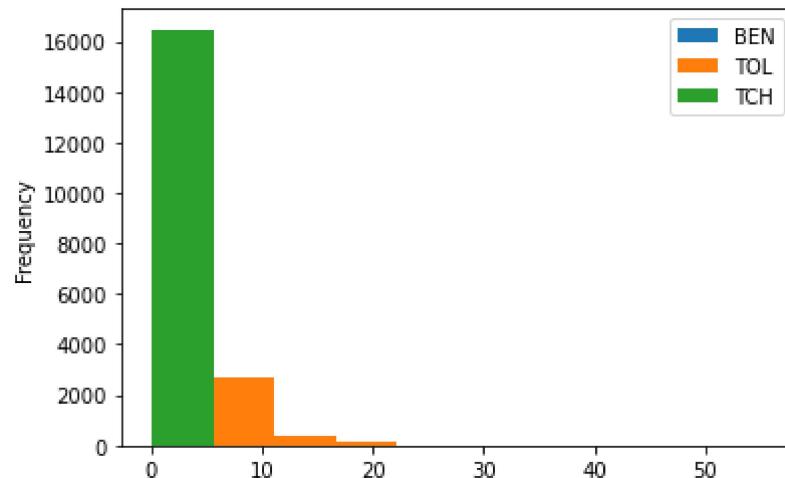
```
Out[10]: <AxesSubplot:>
```



Histogram

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

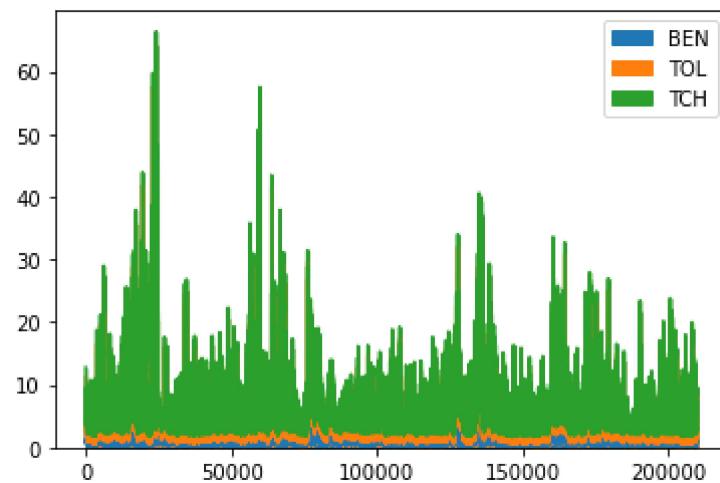
```
Out[11]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

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

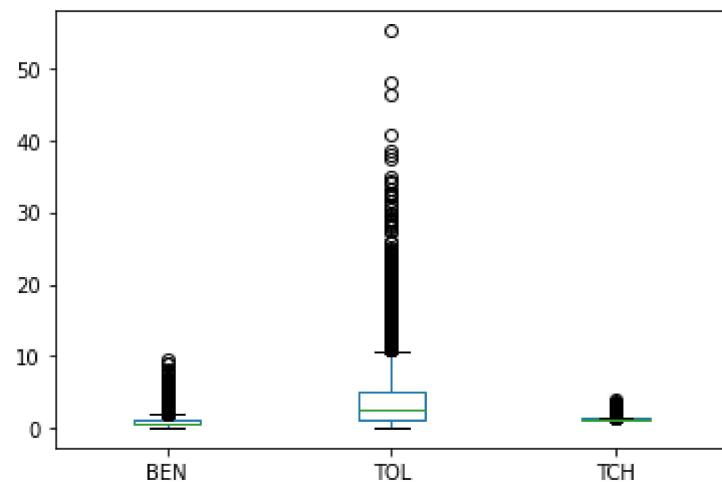
```
Out[12]: <AxesSubplot:>
```



Box chart

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

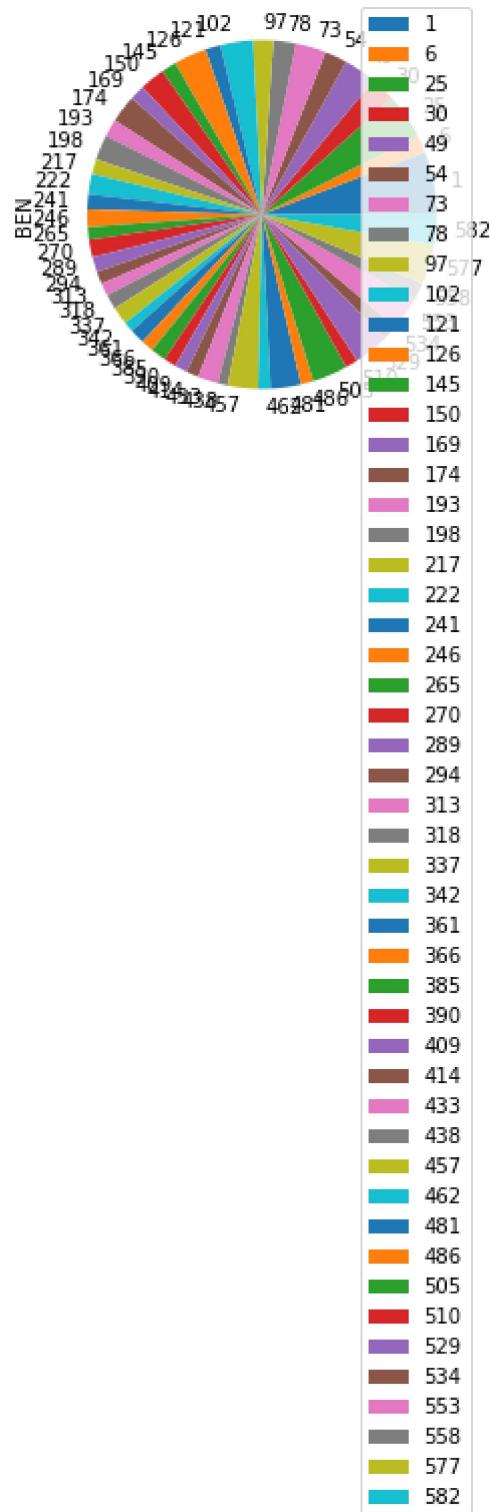
Out[13]: <AxesSubplot:>



Pie chart

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

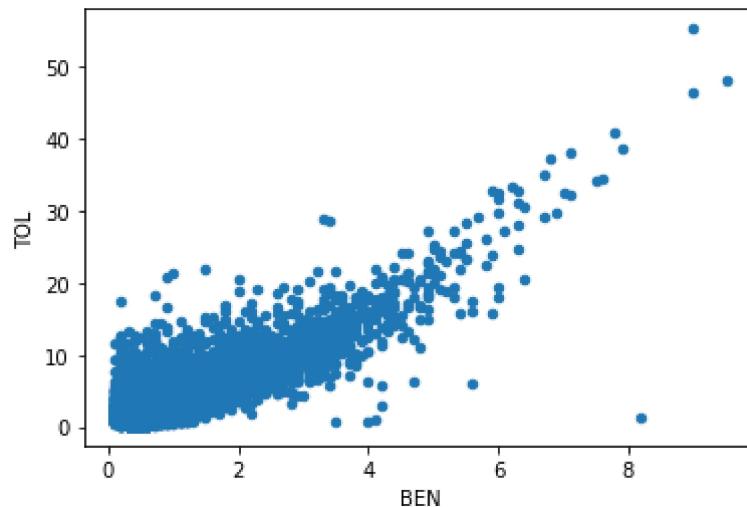
```
Out[14]: <AxesSubplot:ylabel='BEN'>
```



Scatter chart

```
In [15]: data.plot.scatter(x='BEN' ,y='TOL')
```

```
Out[15]: <AxesSubplot:xlabel='BEN', ylabel='TOL'>
```



```
In [16]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16460 entries, 1 to 209910
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      16460 non-null   object 
 1   BEN       16460 non-null   float64
 2   CO        16460 non-null   float64
 3   EBE       16460 non-null   float64
 4   NMHC      16460 non-null   float64
 5   NO        16460 non-null   float64
 6   NO_2      16460 non-null   float64
 7   O_3        16460 non-null   float64
 8   PM10      16460 non-null   float64
 9   PM25      16460 non-null   float64
 10  SO_2      16460 non-null   float64
 11  TCH       16460 non-null   float64
 12  TOL       16460 non-null   float64
 13  station    16460 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.9+ MB
```

In [17]: df.describe()

Out[17]:

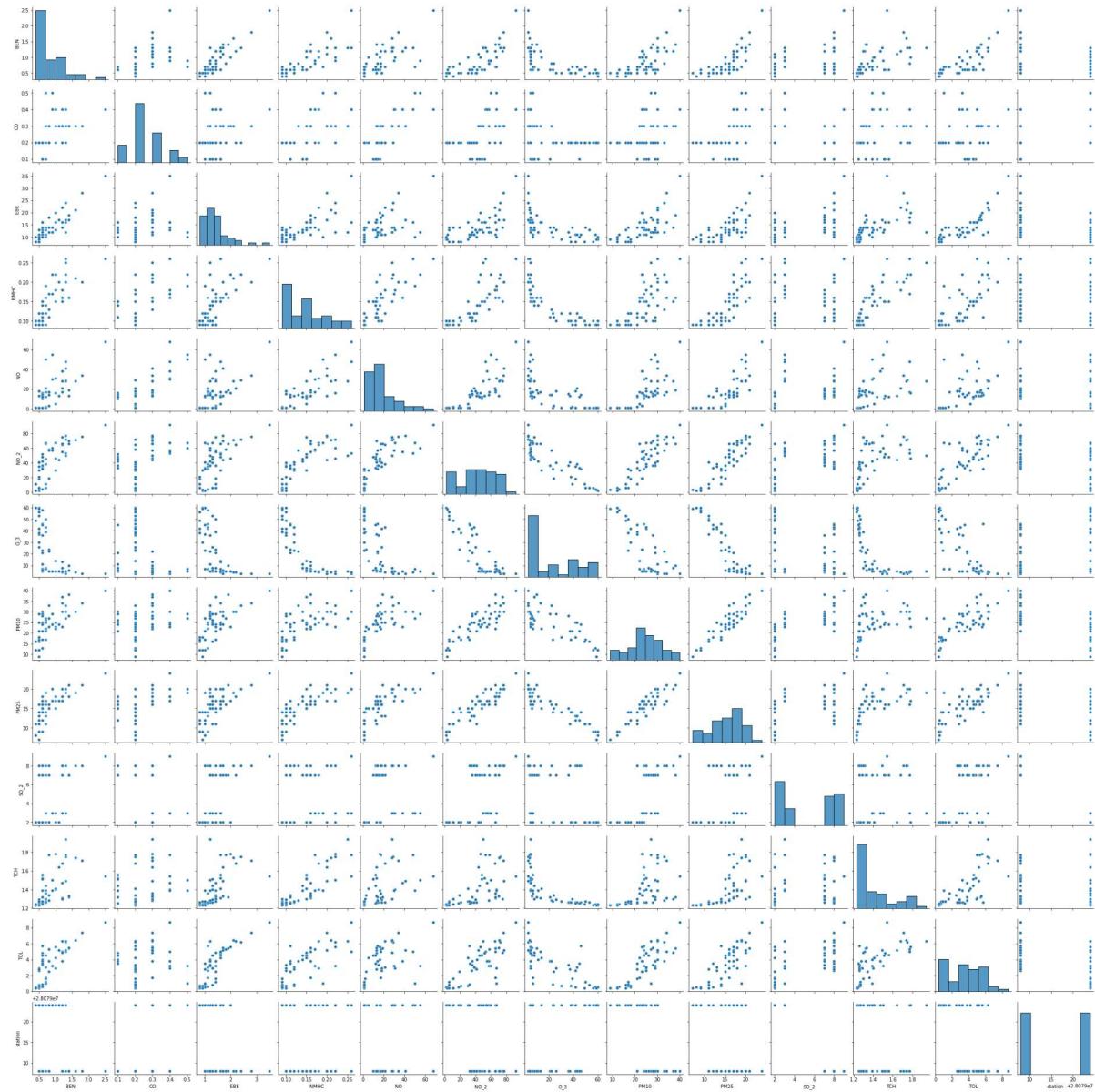
	BEN	CO	EBE	NMHC	NO	NO_2	
count	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	16460.000000	164
mean	0.900680	0.277758	1.471871	0.167043	23.671810	44.583961	
std	0.768892	0.206143	1.051004	0.075068	44.362859	31.569185	
min	0.100000	0.100000	0.200000	0.010000	1.000000	1.000000	
25%	0.500000	0.200000	0.800000	0.120000	2.000000	19.000000	
50%	0.700000	0.200000	1.200000	0.160000	7.000000	40.000000	
75%	1.100000	0.300000	1.700000	0.200000	25.000000	63.000000	
max	9.500000	3.200000	12.800000	0.840000	615.000000	289.000000	1

In [18]: df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station']]

EDA AND VISUALIZATION

```
In [19]: sns.pairplot(df1[0:50])
```

```
Out[19]: <seaborn.axisgrid.PairGrid at 0x1bb2a70d7f0>
```

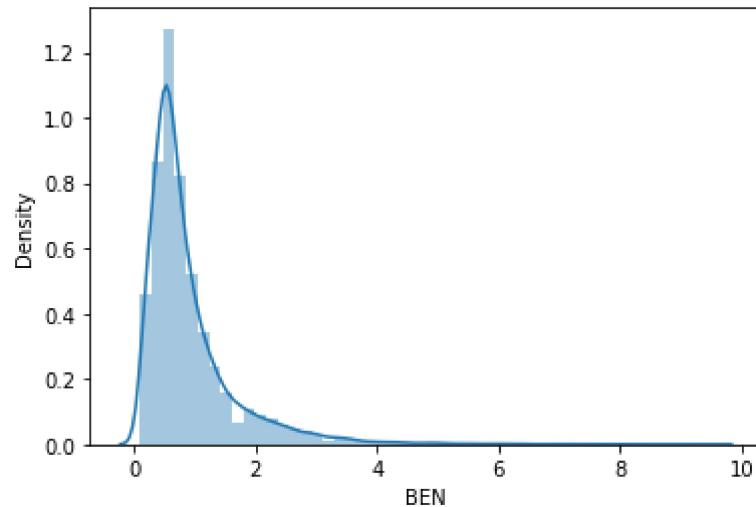


```
In [20]: sns.distplot(df1['BEN'])
```

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).

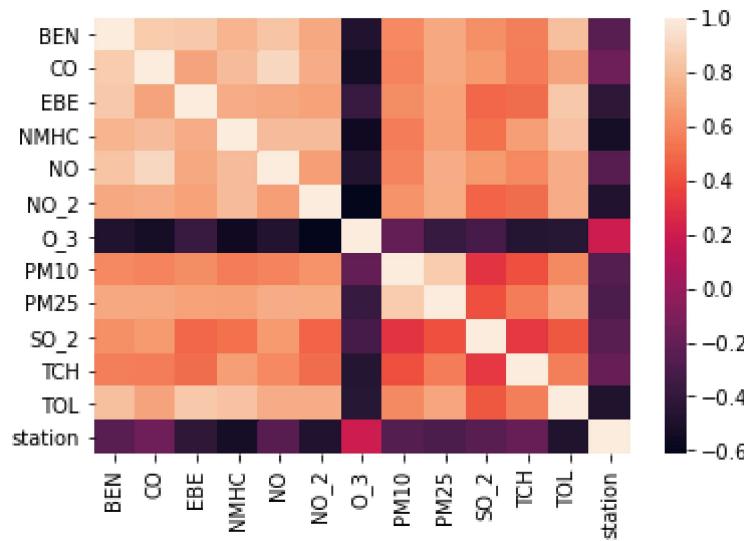
```
warnings.warn(msg, FutureWarning)
```

```
Out[20]: <AxesSubplot:xlabel='BEN', ylabel='Density'>
```



```
In [21]: sns.heatmap(df1.corr())
```

```
Out[21]: <AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BUILDING

```
In [22]: x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',  
           '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]: 28079015.223056305
```

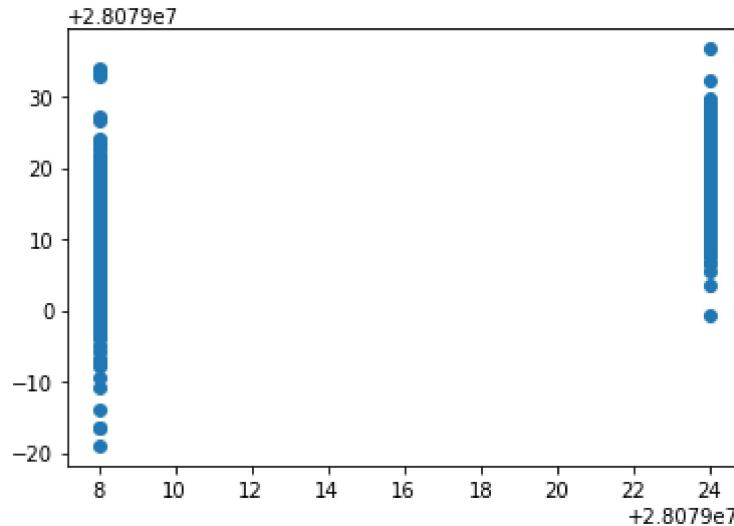
```
In [26]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[26]:
```

	Co-efficient
BEN	3.805170
CO	37.603729
EBE	-1.858668
NMHC	-93.094409
NO	-0.036306
NO_2	-0.089968
O_3	-0.015230
PM10	0.014540
PM25	-0.035677
SO_2	-0.474032
TCH	11.074567
TOL	-0.415248

```
In [27]: prediction = lr.predict(x_test)  
plt.scatter(y_test,prediction)
```

```
Out[27]: <matplotlib.collections.PathCollection at 0x1bb36eccac0>
```



ACCURACY

```
In [28]: lr.score(x_test,y_test)
```

```
Out[28]: 0.6176242622614179
```

```
In [29]: lr.score(x_train,y_train)
```

```
Out[29]: 0.6312376538837546
```

Ridge and Lasso

```
In [30]: from sklearn.linear_model import Ridge,Lasso
```

```
In [31]: rr=Ridge(alpha=10)  
rr.fit(x_train,y_train)
```

```
Out[31]: Ridge(alpha=10)
```

Accuracy(Ridge)

```
In [32]: rr.score(x_test,y_test)
```

```
Out[32]: 0.580321441574621
```

```
In [33]: rr.score(x_train,y_train)
```

```
Out[33]: 0.5979243288685172
```

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

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

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

```
Out[35]: 0.24603868232736525
```

Accuracy(Lasso)

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

```
Out[36]: 0.23241032291592945
```

Accuracy(Elastic Net)

```
In [37]: from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

```
Out[37]: ElasticNet()
```

```
In [38]: en.coef_
```

```
Out[38]: array([ 0.30437437,  0.          , -0.          , -0.          ,
   -0.13281545, -0.04260736,  0.02915675,  0.09360197, -0.1808079 ,
    0.          , -1.00493552])
```

```
In [39]: en.intercept_
```

```
Out[39]: 28079025.017157324
```

```
In [40]: prediction=en.predict(x_test)
```

```
In [41]: en.score(x_test,y_test)
```

```
Out[41]: 0.3502416888161278
```

Evaluation Metrics

```
In [42]: 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)))
```

```
5.604138864118284  
41.58278557254639  
6.448471568716605
```

Logistic Regression

```
In [43]: from sklearn.linear_model import LogisticRegression
```

```
In [44]: feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',  
    'PM10', 'SO_2', 'TCH', 'TOL']]  
target_vector=df['station']
```

```
In [45]: feature_matrix.shape
```

```
Out[45]: (16460, 10)
```

```
In [46]: target_vector.shape
```

```
Out[46]: (16460,)
```

```
In [47]: from sklearn.preprocessing import StandardScaler
```

```
In [48]: fs=StandardScaler().fit_transform(feature_matrix)
```

```
In [49]: logr=LogisticRegression(max_iter=10000)  
logr.fit(fs,target_vector)
```

```
Out[49]: LogisticRegression(max_iter=10000)
```

```
In [50]: observation=[[1,2,3,4,5,6,7,8,9,10]]
```

```
In [51]: prediction=logr.predict(observation)  
print(prediction)
```

```
[28079008]
```

```
In [52]: logr.classes_
```

```
Out[52]: array([28079008, 28079024], dtype=int64)
```

```
In [53]: logr.score(fs,target_vector)
```

```
Out[53]: 0.9237545565006076
```

```
In [54]: logr.predict_proba(observation)[0][0]
```

```
Out[54]: 0.9999999999999966
```

```
In [55]: logr.predict_proba(observation)
```

```
Out[55]: array([[1.0000000e+00, 3.47334507e-15]])
```

Random Forest

```
In [56]: from sklearn.ensemble import RandomForestClassifier
```

```
In [57]: rfc=RandomForestClassifier()  
rfc.fit(x_train,y_train)
```

```
Out[57]: RandomForestClassifier()
```

```
In [58]: parameters={'max_depth':[1,2,3,4,5],  
                  'min_samples_leaf':[5,10,15,20,25],  
                  'n_estimators':[10,20,30,40,50]  
}
```

```
In [59]: 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[59]: 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 [60]: grid_search.best_score_
```

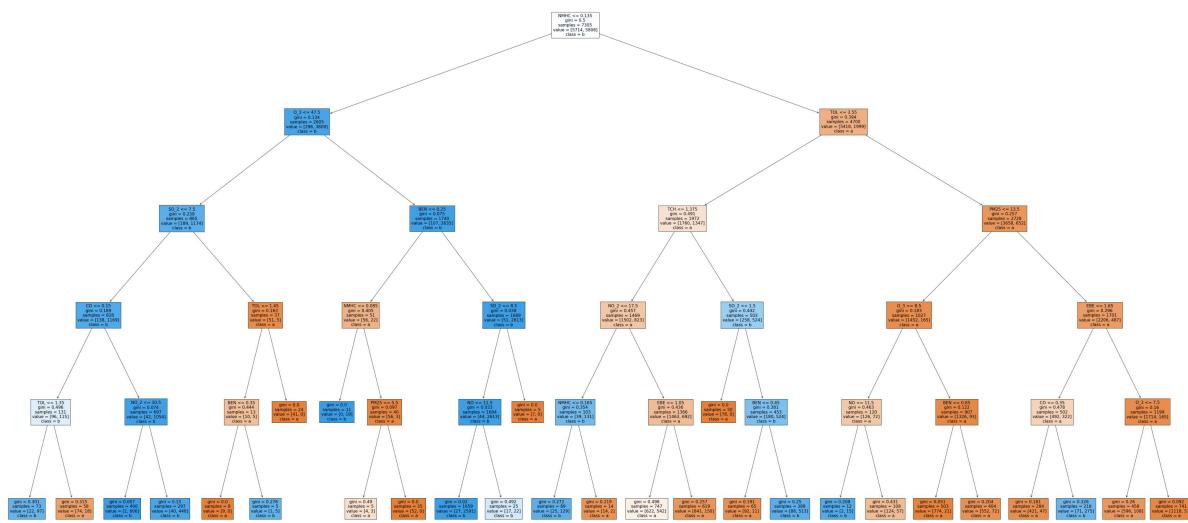
```
Out[60]: 0.9395938205172714
```

```
In [61]: rfc_best=grid_search.best_estimator_
```

```
In [62]: 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[62]: [Text(2142.7200000000003, 1993.2, 'NMHC <= 0.135\ngini = 0.5\nsamples = 7305\nvalue = [5714, 5808]\nclass = b'),  
Text(1138.32, 1630.8000000000002, 'O_3 <= 47.5\ngini = 0.134\nsamples = 2605\nvalue = [296, 3809]\nclass = b'),  
Text(669.6, 1268.4, 'SO_2 <= 7.5\ngini = 0.239\nsamples = 865\nvalue = [189, 1174]\nclass = b'),  
Text(357.12, 906.0, 'CO <= 0.15\ngini = 0.189\nsamples = 828\nvalue = [138, 1169]\nclass = b'),  
Text(178.56, 543.5999999999999, 'TOL <= 1.35\ngini = 0.496\nsamples = 131\nvalue = [96, 115]\nclass = b'),  
Text(89.28, 181.1999999999982, 'gini = 0.301\nsamples = 73\nvalue = [22, 97]\nclass = b'),  
Text(267.8400000000003, 181.1999999999982, 'gini = 0.315\nsamples = 58\nvalue = [74, 18]\nclass = a'),  
Text(535.6800000000001, 543.5999999999999, 'NO_2 <= 30.5\ngini = 0.074\nsamples = 697\nvalue = [42, 1054]\nclass = b'),  
Text(446.4, 181.1999999999982, 'gini = 0.007\nsamples = 400\nvalue = [2, 606]\nclass = b'),  
Text(624.96, 181.1999999999982, 'gini = 0.15\nsamples = 297\nvalue = [40, 448]\nclass = b'),  
Text(982.08, 906.0, 'TOL <= 1.45\ngini = 0.163\nsamples = 37\nvalue = [51, 55]\nclass = a'),  
Text(892.8, 543.5999999999999, 'BEN <= 0.35\ngini = 0.444\nsamples = 13\nvalue = [10, 5]\nclass = a'),  
Text(803.52, 181.1999999999982, 'gini = 0.0\nsamples = 8\nvalue = [9, 0]\nclass = a'),  
Text(982.08, 181.1999999999982, 'gini = 0.278\nsamples = 5\nvalue = [1, 5]\nclass = b'),  
Text(1071.3600000000001, 543.5999999999999, 'gini = 0.0\nsamples = 24\nvalue = [41, 0]\nclass = a'),  
Text(1607.04, 1268.4, 'BEN <= 0.25\ngini = 0.075\nsamples = 1740\nvalue = [107, 2635]\nclass = b'),  
Text(1339.2, 906.0, 'NMHC <= 0.085\ngini = 0.405\nsamples = 51\nvalue = [56, 22]\nclass = a'),  
Text(1249.92, 543.5999999999999, 'gini = 0.0\nsamples = 11\nvalue = [0, 19]\nclass = b'),  
Text(1428.48, 543.5999999999999, 'PM25 <= 5.5\ngini = 0.097\nsamples = 40\nvalue = [56, 3]\nclass = a'),  
Text(1339.2, 181.1999999999982, 'gini = 0.49\nsamples = 5\nvalue = [4, 3]\nclass = a'),  
Text(1517.76, 181.1999999999982, 'gini = 0.0\nsamples = 35\nvalue = [52, 0]\nclass = a'),  
Text(1874.88, 906.0, 'SO_2 <= 8.5\ngini = 0.038\nsamples = 1689\nvalue = [51, 2613]\nclass = b'),  
Text(1785.6, 543.5999999999999, 'NO <= 11.5\ngini = 0.033\nsamples = 1684\nvalue = [44, 2613]\nclass = b'),  
Text(1696.32, 181.1999999999982, 'gini = 0.02\nsamples = 1659\nvalue = [27, 2591]\nclass = b'),  
Text(1874.88, 181.1999999999982, 'gini = 0.492\nsamples = 25\nvalue = [17, 22]\nclass = b'),  
Text(1964.16, 543.5999999999999, 'gini = 0.0\nsamples = 5\nvalue = [7, 0]\nclass = a'),  
Text(3147.12, 1630.8000000000002, 'TOL <= 3.55\ngini = 0.394\nsamples = 4700\nvalue = [5418, 1999]\nclass = a'),  
Text(2544.48, 1268.4, 'TCH <= 1.375\ngini = 0.491\nsamples = 1972\nvalue = [1760, 1347]\nclass = a'),  
Text(2321.28, 906.0, 'NO_2 <= 17.5\ngini = 0.457\nsamples = 1469\nvalue = [1
```

```
502, 823]\nclass = a'),  
    Text(2142.720000000003, 543.599999999999, 'NMHC <= 0.165\ngini = 0.354\nsamples = 103\nvalue = [39, 131]\nclass = b'),  
    Text(2053.44, 181.1999999999982, 'gini = 0.272\nsamples = 89\nvalue = [25, 129]\nclass = b'),  
    Text(2232.0, 181.1999999999982, 'gini = 0.219\nsamples = 14\nvalue = [14, 2]\nclass = a'),  
    Text(2499.84, 543.599999999999, 'EBE <= 1.05\ngini = 0.436\nsamples = 1366\nvalue = [1463, 692]\nclass = a'),  
    Text(2410.56, 181.1999999999982, 'gini = 0.498\nsamples = 747\nvalue = [62 2, 542]\nclass = a'),  
    Text(2589.12, 181.1999999999982, 'gini = 0.257\nsamples = 619\nvalue = [84 1, 150]\nclass = a'),  
    Text(2767.68, 906.0, 'SO_2 <= 1.5\ngini = 0.442\nsamples = 503\nvalue = [25 8, 524]\nclass = b'),  
    Text(2678.4, 543.599999999999, 'gini = 0.0\nsamples = 50\nvalue = [78, 0]\nclass = a'),  
    Text(2856.96, 543.599999999999, 'BEN <= 0.45\ngini = 0.381\nsamples = 453\nvalue = [180, 524]\nclass = b'),  
    Text(2767.68, 181.1999999999982, 'gini = 0.191\nsamples = 65\nvalue = [92, 11]\nclass = a'),  
    Text(2946.240000000002, 181.1999999999982, 'gini = 0.25\nsamples = 388\nvalue = [88, 513]\nclass = b'),  
    Text(3749.76, 1268.4, 'PM25 <= 13.5\ngini = 0.257\nsamples = 2728\nvalue = [3658, 652]\nclass = a'),  
    Text(3392.64, 906.0, 'O_3 <= 8.5\ngini = 0.183\nsamples = 1027\nvalue = [145 2, 165]\nclass = a'),  
    Text(3214.08, 543.599999999999, 'NO <= 11.5\ngini = 0.463\nsamples = 120\nvalue = [126, 72]\nclass = a'),  
    Text(3124.8, 181.1999999999982, 'gini = 0.208\nsamples = 12\nvalue = [2, 1 5]\nclass = b'),  
    Text(3303.36, 181.1999999999982, 'gini = 0.431\nsamples = 108\nvalue = [12 4, 57]\nclass = a'),  
    Text(3571.2, 543.599999999999, 'BEN <= 0.85\ngini = 0.122\nsamples = 907\nvalue = [1326, 93]\nclass = a'),  
    Text(3481.92, 181.1999999999982, 'gini = 0.051\nsamples = 503\nvalue = [77 4, 21]\nclass = a'),  
    Text(3660.48, 181.1999999999982, 'gini = 0.204\nsamples = 404\nvalue = [55 2, 72]\nclass = a'),  
    Text(4106.88, 906.0, 'EBE <= 1.65\ngini = 0.296\nsamples = 1701\nvalue = [22 06, 487]\nclass = a'),  
    Text(3928.32, 543.599999999999, 'CO <= 0.35\ngini = 0.478\nsamples = 502\nvalue = [492, 322]\nclass = a'),  
    Text(3839.04, 181.1999999999982, 'gini = 0.181\nsamples = 284\nvalue = [42 1, 47]\nclass = a'),  
    Text(4017.6, 181.1999999999982, 'gini = 0.326\nsamples = 218\nvalue = [71, 275]\nclass = b'),  
    Text(4285.440000000005, 543.599999999999, 'O_3 <= 7.5\ngini = 0.16\nsamples = 1199\nvalue = [1714, 165]\nclass = a'),  
    Text(4196.16, 181.1999999999982, 'gini = 0.26\nsamples = 458\nvalue = [596, 108]\nclass = a'),  
    Text(4374.72, 181.1999999999982, 'gini = 0.092\nsamples = 741\nvalue = [111 8, 57]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.6312376538837546

Ridge Regression:0.5979243288685172

Lasso Regression:0.23241032291592945

ElasticNet Regression:0.3502416888161278

Logistic Regression:0.9999999999999966

Random Forest:0.9395938205172714

Logistic Regression is suitable for this dataset