

Importing Libraries

```
In [2]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing Datasets

```
In [3]: df=pd.read_csv(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\ma
df
```

Out[3]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL
0	2015-10-01 01:00:00	NaN	0.8	NaN	NaN	90.0	82.0	NaN	NaN	NaN	10.0	NaN	NaN 28
1	2015-10-01 01:00:00	2.0	0.8	1.6	0.33	40.0	95.0	4.0	37.0	24.0	12.0	1.83	8.3 28
2	2015-10-01 01:00:00	3.1	NaN	1.8	NaN	29.0	97.0	NaN	NaN	NaN	NaN	NaN	7.1 28
3	2015-10-01 01:00:00	NaN	0.6	NaN	NaN	30.0	103.0	2.0	NaN	NaN	NaN	NaN	NaN 28
4	2015-10-01 01:00:00	NaN	NaN	NaN	NaN	95.0	96.0	2.0	NaN	NaN	9.0	NaN	NaN 28
...
210091	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	11.0	33.0	53.0	NaN	NaN	NaN	NaN	NaN 28
210092	2015-08-01 00:00:00	NaN	0.2	NaN	NaN	1.0	5.0	NaN	26.0	NaN	10.0	NaN	NaN 28
210093	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	7.0	74.0	NaN	NaN	NaN	NaN	NaN 28
210094	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	3.0	7.0	65.0	NaN	NaN	NaN	NaN	NaN 28
210095	2015-08-01 00:00:00	NaN	NaN	NaN	NaN	1.0	9.0	54.0	29.0	NaN	NaN	NaN	NaN 28

210096 rows × 14 columns

Data Cleaning and Data Preprocessing

```
In [4]: df=df.dropna()
```

```
In [5]: df.columns
```

```
Out[5]: Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

```
In [6]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      16026 non-null   object 
 1   BEN        16026 non-null   float64
 2   CO         16026 non-null   float64
 3   EBE        16026 non-null   float64
 4   NMHC       16026 non-null   float64
 5   NO         16026 non-null   float64
 6   NO_2       16026 non-null   float64
 7   O_3        16026 non-null   float64
 8   PM10       16026 non-null   float64
 9   PM25       16026 non-null   float64
 10  SO_2       16026 non-null   float64
 11  TCH        16026 non-null   float64
 12  TOL        16026 non-null   float64
 13  station    16026 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

```
In [7]: data=df[['BEN', 'TOL', 'TCH']]  
data
```

Out[7]:

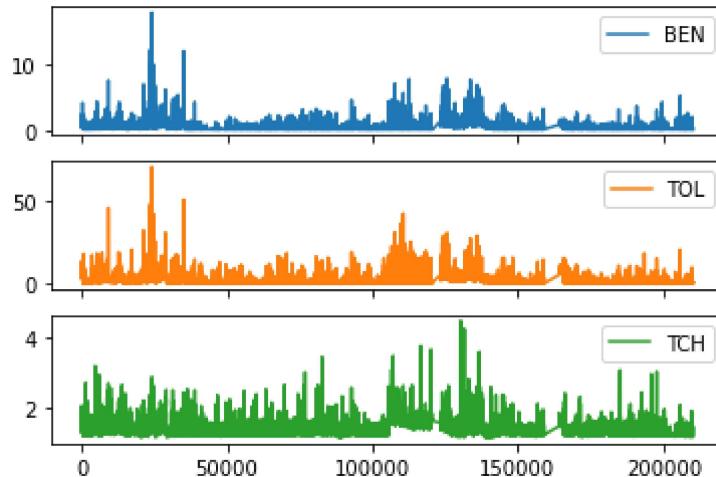
	BEN	TOL	TCH
1	2.0	8.3	1.83
6	0.5	4.8	1.29
25	1.6	6.9	1.93
30	0.4	7.8	1.27
49	2.2	13.9	2.05
...
210030	0.1	0.2	1.18
210049	0.4	1.2	1.45
210054	0.1	0.2	1.18
210073	0.1	0.6	1.44
210078	0.1	0.4	1.18

16026 rows × 3 columns

Line chart

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

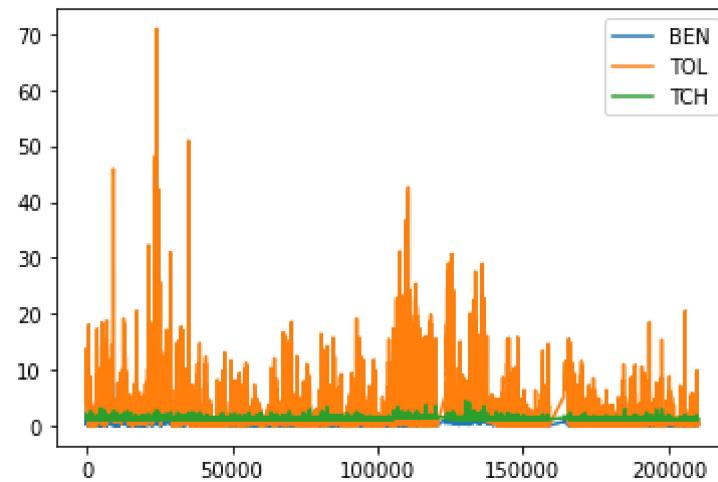
Out[8]: array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

```
Out[9]: <AxesSubplot:>
```

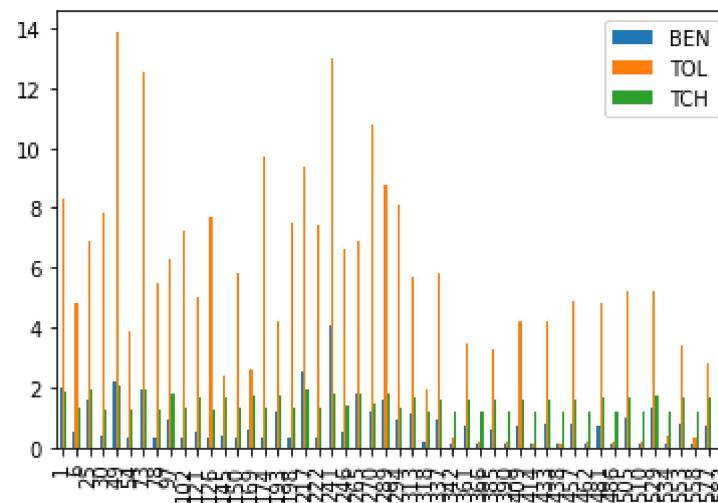


Bar chart

```
In [10]: b=data[0:50]
```

```
In [11]: b.plot.bar()
```

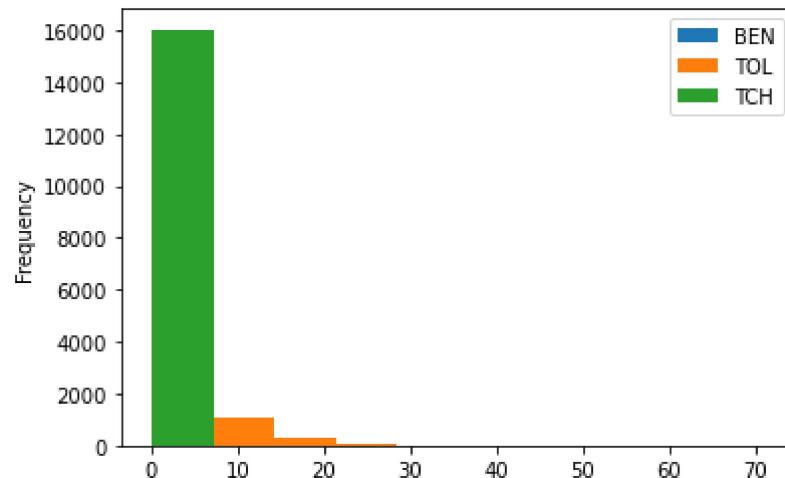
```
Out[11]: <AxesSubplot:>
```



Histogram

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

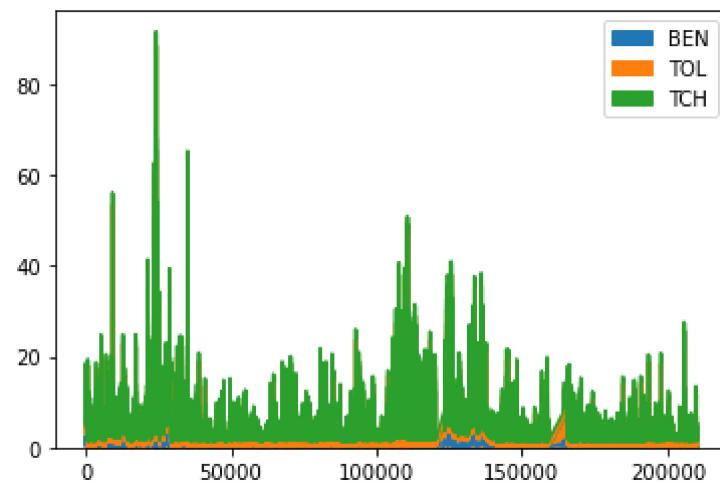
```
Out[12]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

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

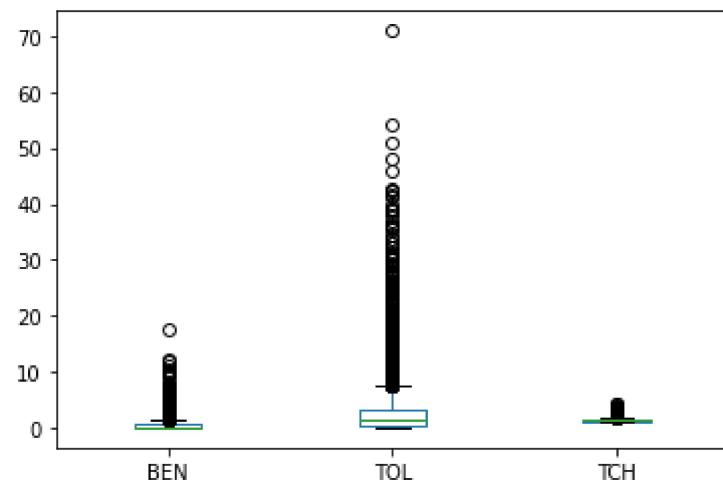
```
Out[13]: <AxesSubplot:>
```



Box chart

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

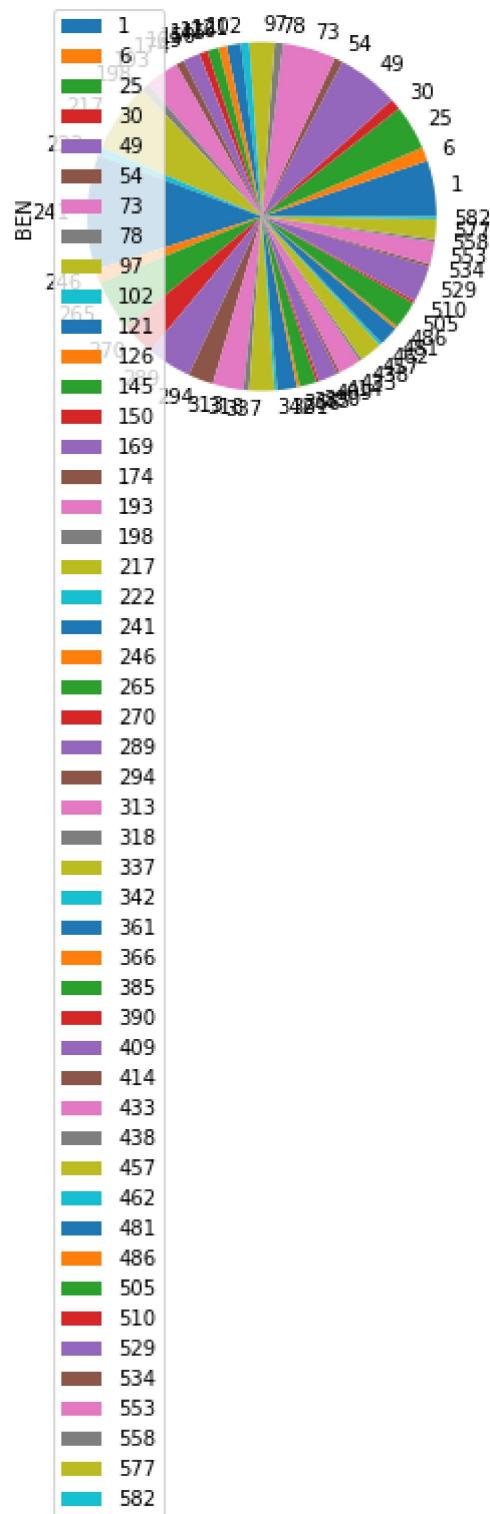
Out[14]: <AxesSubplot:>



Pie chart

```
In [15]: b.plot.pie(y='BEN' )
```

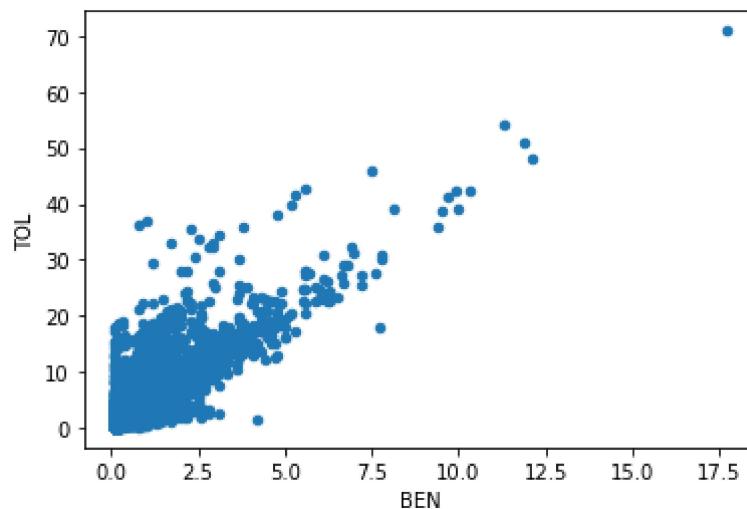
```
Out[15]: <AxesSubplot:ylabel='BEN'>
```



Scatter chart

```
In [16]: data.plot.scatter(x='BEN' ,y='TOL')
```

```
Out[16]: <AxesSubplot:xlabel='BEN', ylabel='TOL'>
```



```
In [17]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16026 entries, 1 to 210078
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      16026 non-null   object 
 1   BEN       16026 non-null   float64
 2   CO        16026 non-null   float64
 3   EBE       16026 non-null   float64
 4   NMHC      16026 non-null   float64
 5   NO        16026 non-null   float64
 6   NO_2      16026 non-null   float64
 7   O_3        16026 non-null   float64
 8   PM10      16026 non-null   float64
 9   PM25      16026 non-null   float64
 10  SO_2      16026 non-null   float64
 11  TCH       16026 non-null   float64
 12  TOL       16026 non-null   float64
 13  station    16026 non-null   int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 1.8+ MB
```

In [18]: df.describe()

Out[18]:

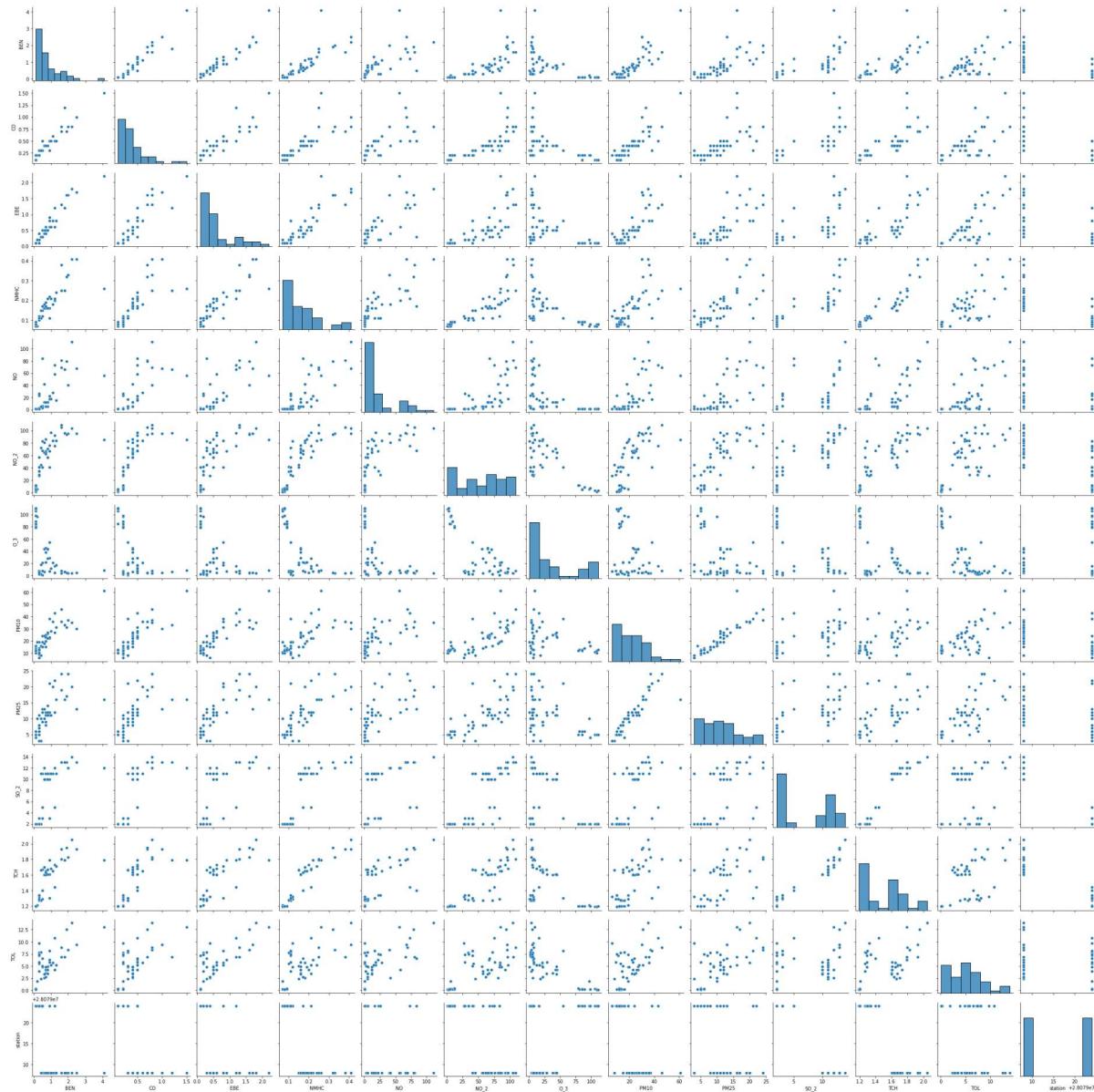
	BEN	CO	EBE	NMHC	NO	NO_2	
count	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000	16026.000000
mean	0.504823	0.380594	0.394247	0.123099	23.842256	40.948771	
std	0.716896	0.260805	0.678592	0.092368	51.255660	33.236098	
min	0.100000	0.100000	0.100000	0.000000	1.000000	1.000000	
25%	0.100000	0.200000	0.100000	0.070000	1.000000	14.000000	
50%	0.200000	0.300000	0.100000	0.100000	6.000000	35.000000	
75%	0.700000	0.500000	0.400000	0.140000	24.000000	60.000000	
max	17.700001	4.500000	12.100000	1.090000	960.000000	369.000000	2

In [19]: df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station']]

EDA AND VISUALIZATION

```
In [20]: sns.pairplot(df1[0:50])
```

```
Out[20]: <seaborn.axisgrid.PairGrid at 0x1ee9e9f0c40>
```

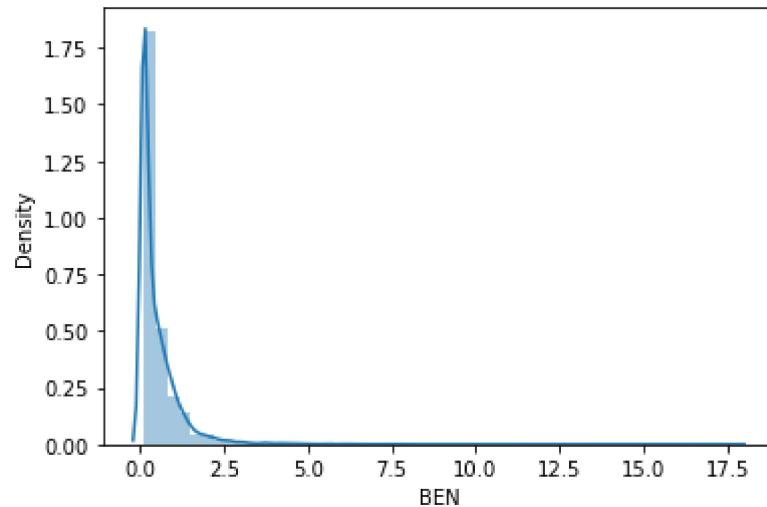


```
In [21]: sns.distplot(df1['BEN'])
```

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).

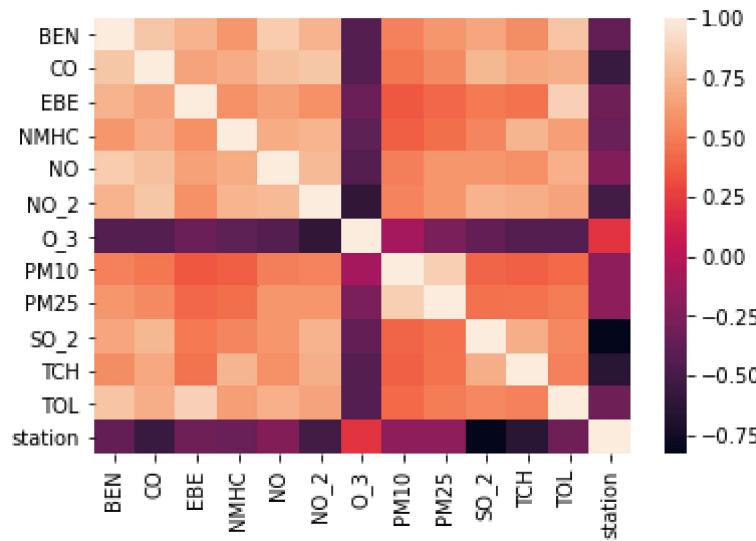
```
warnings.warn(msg, FutureWarning)
```

```
Out[21]: <AxesSubplot:xlabel='BEN', ylabel='Density'>
```



```
In [22]: sns.heatmap(df1.corr())
```

```
Out[22]: <AxesSubplot:>
```



TO TRAIN THE MODEL AND MODEL BUILDING

```
In [23]: x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',  
           'SO_2', 'TCH', 'TOL']]  
y=df['station']
```

```
In [24]: 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 [25]: from sklearn.linear_model import LinearRegression  
lr=LinearRegression()  
lr.fit(x_train,y_train)
```

```
Out[25]: LinearRegression()
```

```
In [26]: lr.intercept_
```

```
Out[26]: 28079038.123703483
```

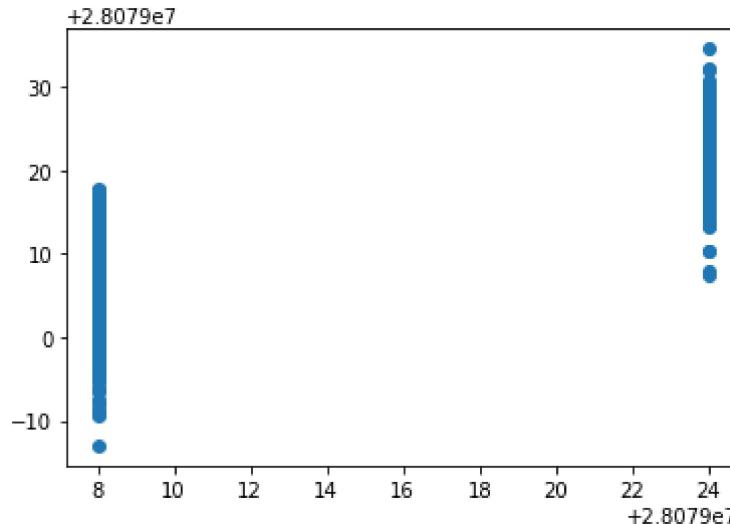
```
In [27]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[27]:
```

	Co-efficient
BEN	1.150938
CO	-9.490519
EBE	-0.493180
NMHC	13.263580
NO	0.079809
NO_2	-0.018493
O_3	-0.014573
PM10	0.010000
PM25	0.096769
SO_2	-1.127079
TCH	-9.503718
TOL	-0.115914

```
In [28]: prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

```
Out[28]: <matplotlib.collections.PathCollection at 0x1eeabbee0d0>
```



ACCURACY

```
In [29]: lr.score(x_test,y_test)
```

```
Out[29]: 0.8712230144436421
```

```
In [30]: lr.score(x_train,y_train)
```

```
Out[30]: 0.871779860524847
```

Ridge and Lasso

```
In [31]: from sklearn.linear_model import Ridge,Lasso
```

```
In [32]: rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

```
Out[32]: Ridge(alpha=10)
```

Accuracy(Ridge)

```
In [33]: rr.score(x_test,y_test)
```

```
Out[33]: 0.8699290510221751
```

```
In [34]: rr.score(x_train,y_train)
```

```
Out[34]: 0.8710049657009941
```

```
In [35]: la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

```
Out[35]: Lasso(alpha=10)
```

```
In [36]: la.score(x_test,y_test)
```

```
Out[36]: 0.7297414144040693
```

Accuracy(Lasso)

```
In [37]: la.score(x_train,y_train)
```

```
Out[37]: 0.7320870681548695
```

Accuracy(Elastic Net)

```
In [38]: from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

```
Out[38]: ElasticNet()
```

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

```
Out[39]: array([-0.          , -0.          , -0.          , -0.          ,
 -0.05194118, -0.01167171,  0.02182083,  0.05253284, -1.31766686,
 -0.          , -0.08100712])
```

```
In [40]: en.intercept_
```

```
Out[40]: 28079025.950274505
```

```
In [41]: prediction=en.predict(x_test)
```

```
In [42]: en.score(x_test,y_test)
```

```
Out[42]: 0.818679035054062
```

Evaluation Metrics

```
In [43]: 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)))
```

2.5316843413887313
11.604509628883251
3.406539245169979

Logistic Regression

```
In [44]: from sklearn.linear_model import LogisticRegression
```

```
In [45]: feature_matrix=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',  
    'PM10', 'SO_2', 'TCH', 'TOL']]  
target_vector=df['station']
```

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

```
Out[46]: (16026, 10)
```

```
In [47]: target_vector.shape
```

```
Out[47]: (16026,)
```

```
In [48]: from sklearn.preprocessing import StandardScaler
```

```
In [49]: fs=StandardScaler().fit_transform(feature_matrix)
```

```
In [50]: logr=LogisticRegression(max_iter=10000)  
logr.fit(fs,target_vector)
```

```
Out[50]: LogisticRegression(max_iter=10000)
```

```
In [51]: observation=[[1,2,3,4,5,6,7,8,9,10]]
```

```
In [52]: prediction=logr.predict(observation)  
print(prediction)
```

[28079008]

```
In [53]: logr.classes_
```

```
Out[53]: array([28079008, 28079024], dtype=int64)
```

```
In [54]: logr.score(fs,target_vector)
```

```
Out[54]: 0.9947585174092101
```

```
In [55]: logr.predict_proba(observation)[0][0]
```

```
Out[55]: 1.0
```

```
In [56]: logr.predict_proba(observation)
```

```
Out[56]: array([[1.0, 5.6979311e-39]])
```

Random Forest

```
In [57]: from sklearn.ensemble import RandomForestClassifier
```

```
In [58]: rfc=RandomForestClassifier()  
rfc.fit(x_train,y_train)
```

```
Out[58]: RandomForestClassifier()
```

```
In [59]: parameters={'max_depth':[1,2,3,4,5],  
                  'min_samples_leaf':[5,10,15,20,25],  
                  'n_estimators':[10,20,30,40,50]  
}
```

```
In [60]: 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[60]: 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 [61]: grid_search.best_score_
```

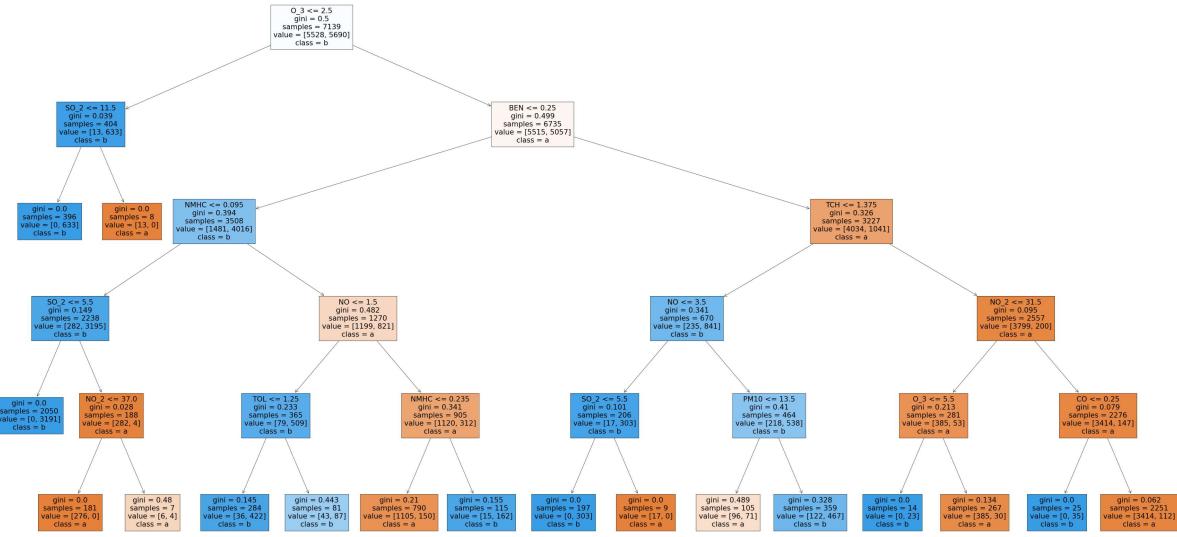
```
Out[61]: 0.9940274558744875
```

```
In [62]: rfc_best=grid_search.best_estimator_
```

```
In [63]: 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[63]: [Text(1212.2068965517242, 1993.2, 'O_3 <= 2.5\nngini = 0.5\nsamples = 7139\nvalue = [5528, 5690]\nclass = b'),  
Text(384.82758620689657, 1630.8000000000002, 'SO_2 <= 11.5\nngini = 0.039\nsamples = 404\nvalue = [13, 633]\nclass = b'),  
Text(230.89655172413796, 1268.4, 'gini = 0.0\nsamples = 396\nvalue = [0, 633]\nclass = b'),  
Text(538.7586206896552, 1268.4, 'gini = 0.0\nsamples = 8\nvalue = [13, 0]\nclass = a'),  
Text(2039.5862068965519, 1630.8000000000002, 'BEN <= 0.25\nngini = 0.499\nsamples = 6735\nvalue = [5515, 5057]\nclass = a'),  
Text(846.6206896551724, 1268.4, 'NMHC <= 0.095\nngini = 0.394\nsamples = 3508\nvalue = [1481, 4016]\nclass = b'),  
Text(307.86206896551727, 906.0, 'SO_2 <= 5.5\nngini = 0.149\nsamples = 2238\nvalue = [282, 3195]\nclass = b'),  
Text(153.93103448275863, 543.5999999999999, 'gini = 0.0\nsamples = 2050\nvalue = [0, 3191]\nclass = b'),  
Text(461.79310344827593, 543.5999999999999, 'NO_2 <= 37.0\nngini = 0.028\nsamples = 188\nvalue = [282, 4]\nclass = a'),  
Text(307.86206896551727, 181.1999999999982, 'gini = 0.0\nsamples = 181\nvalue = [276, 0]\nclass = a'),  
Text(615.7241379310345, 181.1999999999982, 'gini = 0.48\nsamples = 7\nvalue = [6, 4]\nclass = a'),  
Text(1385.3793103448277, 906.0, 'NO <= 1.5\nngini = 0.482\nsamples = 1270\nvalue = [1199, 821]\nclass = a'),  
Text(1077.5172413793105, 543.5999999999999, 'TOL <= 1.25\nngini = 0.233\nsamples = 365\nvalue = [79, 509]\nclass = b'),  
Text(923.5862068965519, 181.1999999999982, 'gini = 0.145\nsamples = 284\nvalue = [36, 422]\nclass = b'),  
Text(1231.448275862069, 181.1999999999982, 'gini = 0.443\nsamples = 81\nvalue = [43, 87]\nclass = b'),  
Text(1693.2413793103449, 543.5999999999999, 'NMHC <= 0.235\nngini = 0.341\nsamples = 905\nvalue = [1120, 312]\nclass = a'),  
Text(1539.3103448275863, 181.1999999999982, 'gini = 0.21\nsamples = 790\nvalue = [1105, 150]\nclass = a'),  
Text(1847.1724137931037, 181.1999999999982, 'gini = 0.155\nsamples = 115\nvalue = [15, 162]\nclass = b'),  
Text(3232.551724137931, 1268.4, 'TCH <= 1.375\nngini = 0.326\nsamples = 3227\nvalue = [4034, 1041]\nclass = a'),  
Text(2616.8275862068967, 906.0, 'NO <= 3.5\nngini = 0.341\nsamples = 670\nvalue = [235, 841]\nclass = b'),  
Text(2308.9655172413795, 543.5999999999999, 'SO_2 <= 5.5\nngini = 0.101\nsamples = 206\nvalue = [17, 303]\nclass = b'),  
Text(2155.034482758621, 181.1999999999982, 'gini = 0.0\nsamples = 197\nvalue = [0, 303]\nclass = b'),  
Text(2462.896551724138, 181.1999999999982, 'gini = 0.0\nsamples = 9\nvalue = [17, 0]\nclass = a'),  
Text(2924.689655172414, 543.5999999999999, 'PM10 <= 13.5\nngini = 0.41\nsamples = 464\nvalue = [218, 538]\nclass = b'),  
Text(2770.7586206896553, 181.1999999999982, 'gini = 0.489\nsamples = 105\nvalue = [96, 71]\nclass = a'),  
Text(3078.6206896551726, 181.1999999999982, 'gini = 0.328\nsamples = 359\nvalue = [122, 467]\nclass = b'),  
Text(3848.275862068966, 906.0, 'NO_2 <= 31.5\nngini = 0.095\nsamples = 2557\nvalue = [3799, 200]\nclass = a'),  
Text(3540.4137931034484, 543.5999999999999, 'O_3 <= 5.5\nngini = 0.213\nsamples = 281\nvalue = [385, 53]\nclass = a'),  
Text(3386.4827586206898, 181.1999999999982, 'gini = 0.0\nsamples = 14\nvalue = [18, 16]\nclass = a')]
```

```
e = [0, 23]\nclass = b'),
Text(3694.3448275862074, 181.19999999999982, 'gini = 0.134\ncount = 267\nvalue = [385, 30]\nclass = a'),
Text(4156.137931034483, 543.5999999999999, 'CO <= 0.25\ngini = 0.079\ncount = 2276\nvalue = [3414, 147]\nclass = a'),
Text(4002.2068965517246, 181.19999999999982, 'gini = 0.0\ncount = 25\nvalue = [0, 35]\nclass = b'),
Text(4310.068965517242, 181.19999999999982, 'gini = 0.062\ncount = 2251\nvalue = [3414, 112]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression: 0.871779860524847

Ridge Regression: 0.8710049657009941

Lasso Regression: 0.7320870681548695

ElasticNet Regression: 0.818679035054062

Logistic Regression: 0.9947585174092101

Random Forest: 0.9940274558744875

Logistic Regression is suitable for this dataset

