

# 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("madrid_2010.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.410000
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.670000
...	...	...	...	...	...	...	...	...	...	...	...
209443	2010-08-01 00:00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	
209444	2010-08-01 00:00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.250000
209445	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	
209446	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	
209447	2010-08-01 00:00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.150000

209448 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: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      6666 non-null   object 
 1   BEN        6666 non-null   float64
 2   CO         6666 non-null   float64
 3   EBE        6666 non-null   float64
 4   MXY        6666 non-null   float64
 5   NMHC       6666 non-null   float64
 6   NO_2       6666 non-null   float64
 7   NOx        6666 non-null   float64
 8   OXY        6666 non-null   float64
 9   O_3         6666 non-null   float64
 10  PM10       6666 non-null   float64
 11  PM25       6666 non-null   float64
 12  PXY        6666 non-null   float64
 13  SO_2       6666 non-null   float64
 14  TCH         6666 non-null   float64
 15  TOL         6666 non-null   float64
 16  station    6666 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 937.4+ KB
```

```
In [6]: data=df[['CO' , 'station']]  
data
```

```
Out[6]:
```

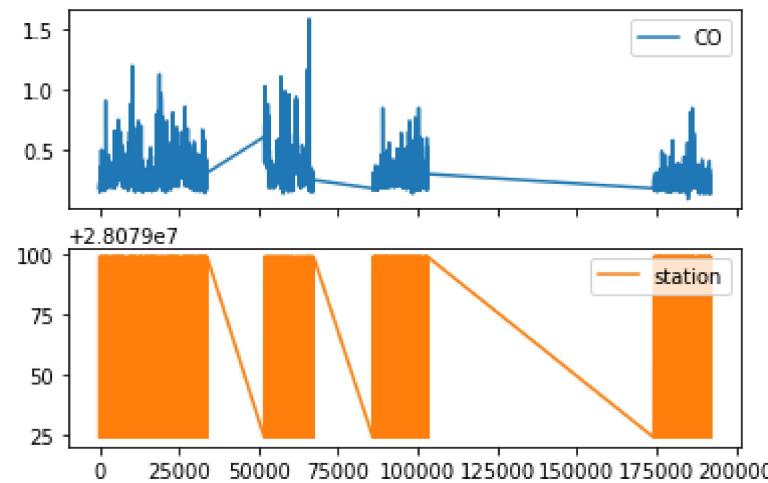
	CO	station
11	0.18	28079024
23	0.23	28079099
35	0.17	28079024
47	0.21	28079099
59	0.16	28079024
...	...	...
191879	0.26	28079099
191891	0.16	28079024
191903	0.28	28079099
191915	0.16	28079024
191927	0.25	28079099

6666 rows × 2 columns

## Line chart

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

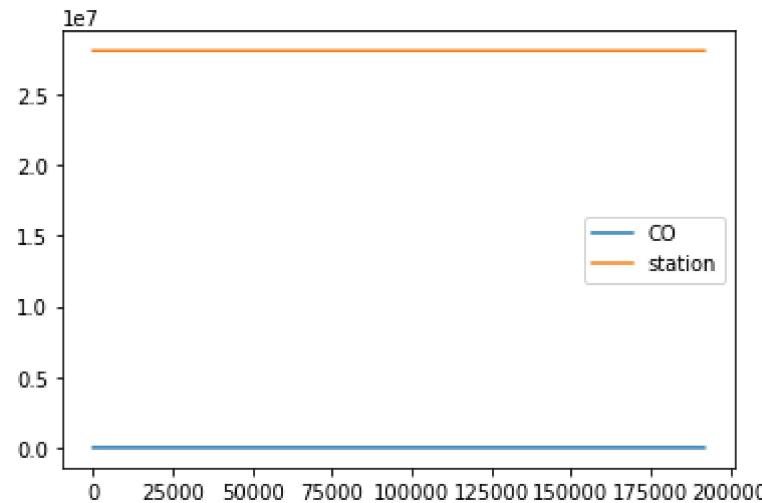
```
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



## Line chart

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

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

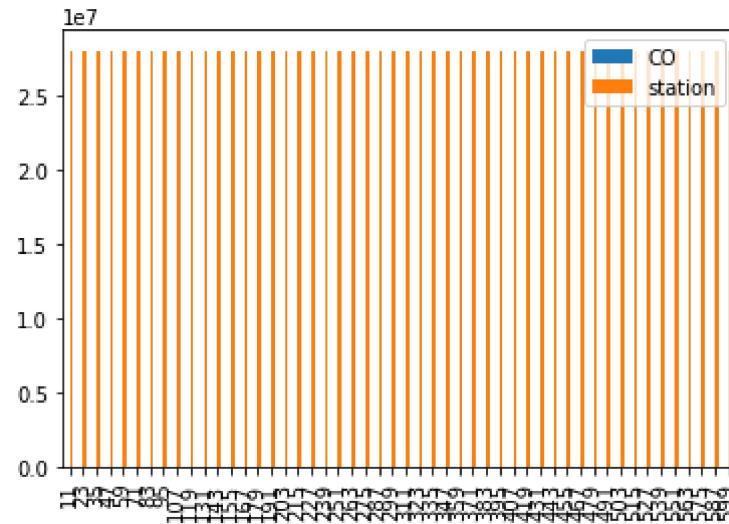


## Bar chart

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

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

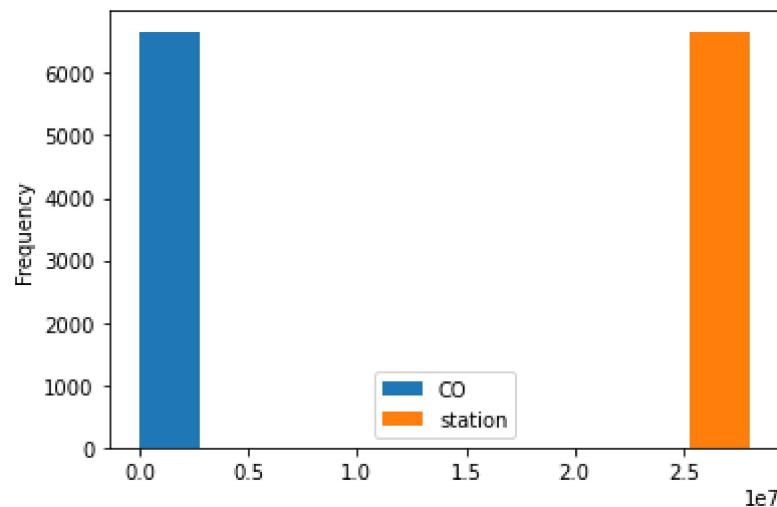
```
Out[10]: <AxesSubplot:>
```



## Histogram

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

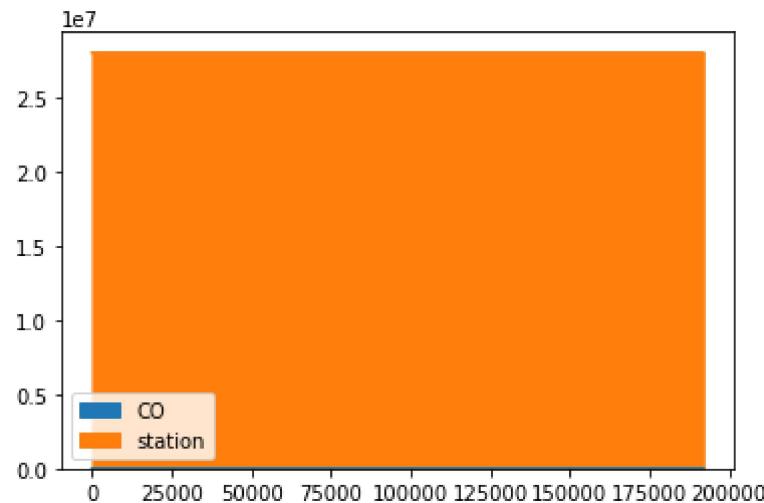
```
Out[11]: <AxesSubplot:ylabel='Frequency'>
```



## Area chart

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

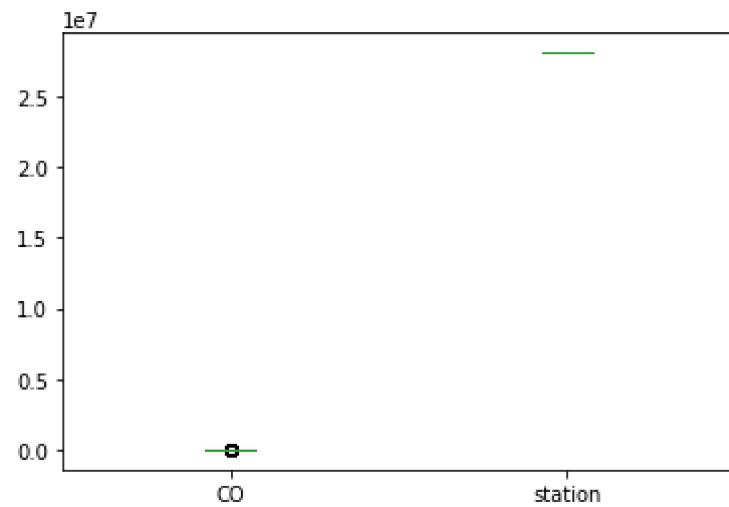
```
Out[12]: <AxesSubplot:>
```



## Box chart

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

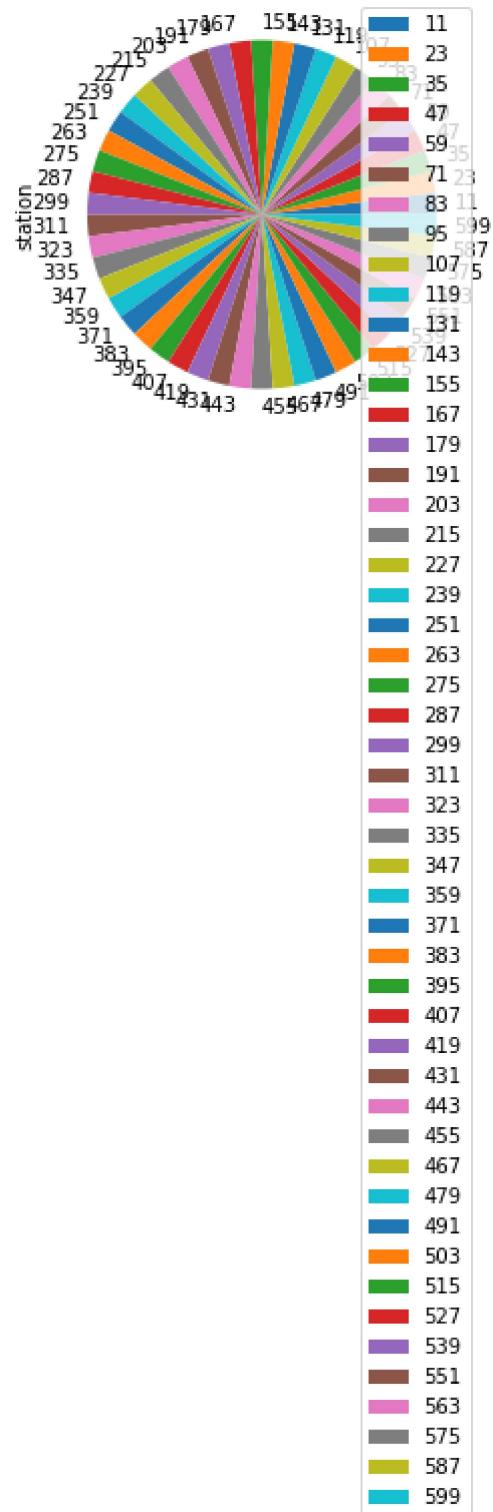
Out[13]: <AxesSubplot:>



## Pie chart

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

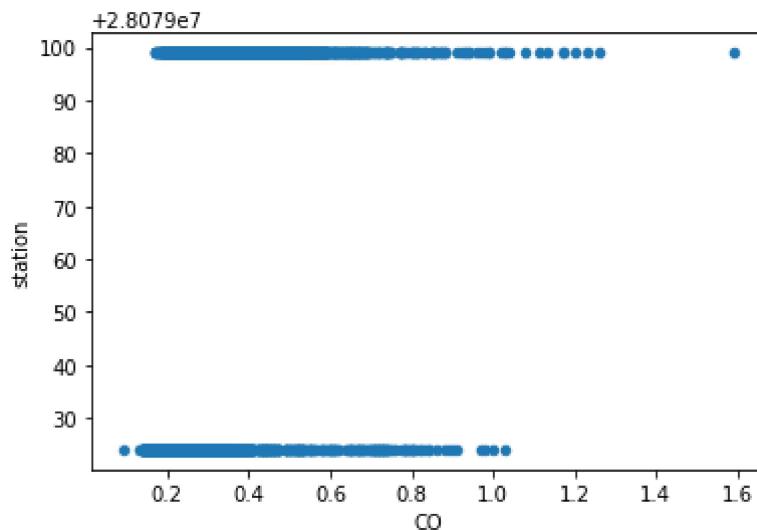
```
Out[14]: <AxesSubplot:ylabel='station'>
```



## Scatter chart

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

```
Out[15]: <AxesSubplot:xlabel='CO', ylabel='station'>
```



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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6666 entries, 11 to 191927
Data columns (total 17 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   date      6666 non-null   object 
 1   BEN       6666 non-null   float64
 2   CO        6666 non-null   float64
 3   EBE       6666 non-null   float64
 4   MXY       6666 non-null   float64
 5   NMHC      6666 non-null   float64
 6   NO_2      6666 non-null   float64
 7   NOx       6666 non-null   float64
 8   OXY       6666 non-null   float64
 9   O_3        6666 non-null   float64
 10  PM10      6666 non-null   float64
 11  PM25      6666 non-null   float64
 12  PXY       6666 non-null   float64
 13  SO_2      6666 non-null   float64
 14  TCU       6666 non-null   float64
```

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

Out[17]:

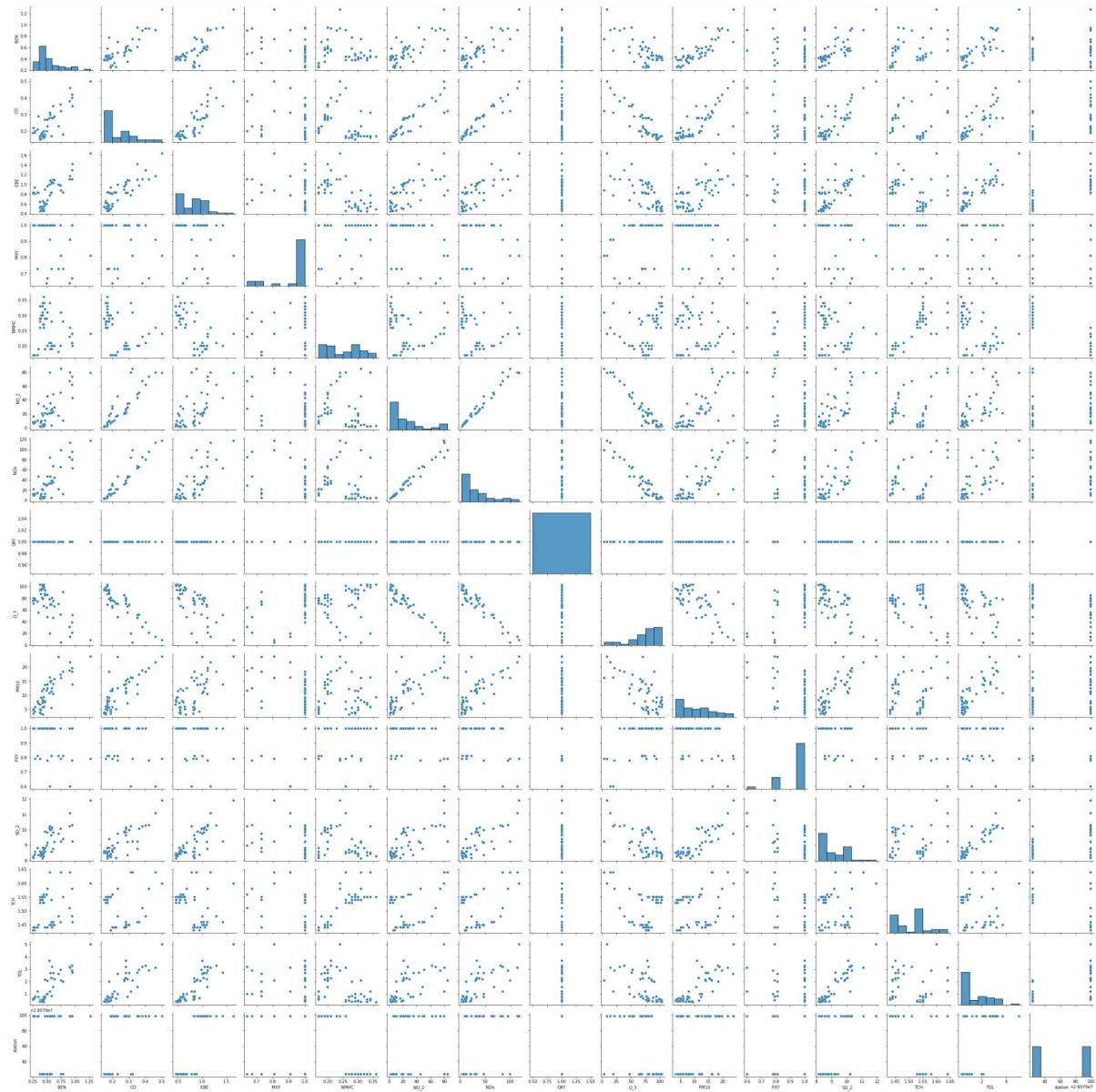
	BEN	CO	EBE	MXY	NMHC	NO_2	NC
count	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000	6666.000000
mean	0.648425	0.296280	0.840585	0.839959	0.243378	33.888744	47.5406
std	0.395346	0.133296	0.508031	0.382263	0.115730	23.465169	41.2305
min	0.170000	0.090000	0.140000	0.110000	0.000000	1.290000	2.7600
25%	0.380000	0.200000	0.470000	0.590000	0.180000	15.752500	19.4425
50%	0.540000	0.260000	0.755000	1.000000	0.220000	29.320000	36.7700
75%	0.810000	0.340000	1.000000	1.000000	0.280000	47.657500	62.1025
max	5.110000	1.590000	5.190000	6.810000	0.930000	133.399994	409.29998

```
In [18]: df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',  
'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

## EDA AND VISUALIZATION

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

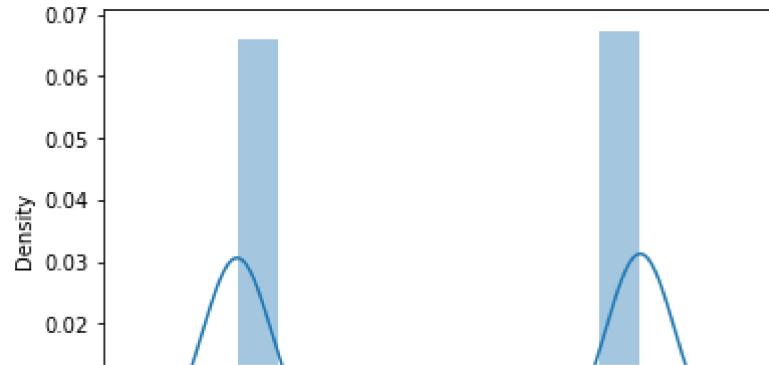
```
Out[19]: <seaborn.axisgrid.PairGrid at 0x25149bf4940>
```



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

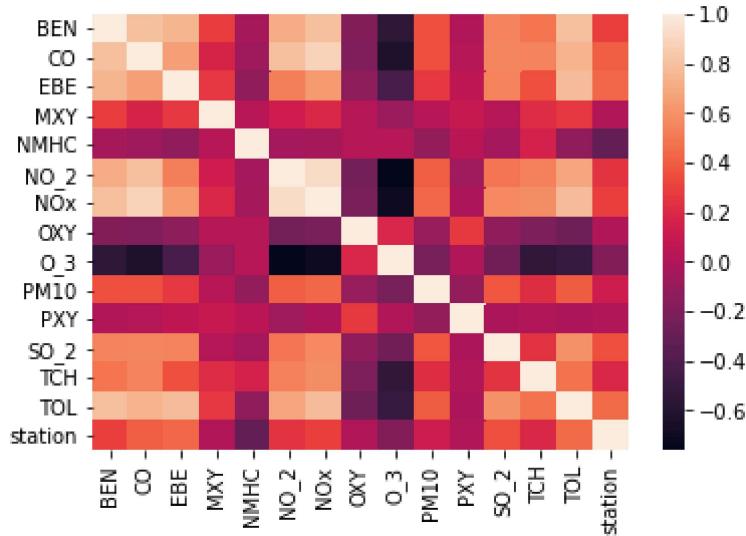
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function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

Out[20]: <AxesSubplot:xlabel='station', 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', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3', 'PM10', 'PXY', '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]: 28078936.56162695
```

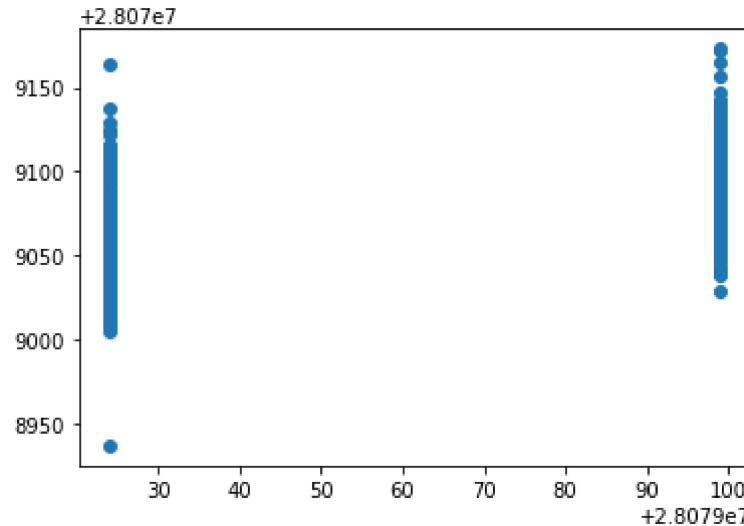
```
In [26]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[26]:
```

	Co-efficient
BEN	-34.578844
CO	161.384133
EBE	16.160963
MXY	-9.338061
NMHC	-71.158044
NO_2	0.272834
NOx	-0.637946
OXY	30.111022
O_3	0.049036
PM10	-0.188619
PXY	-5.029782
SO_2	1.796968
TCH	47.937807
TOL	10.449652

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

```
Out[27]: <matplotlib.collections.PathCollection at 0x25157c0b4c0>
```



## ACCURACY

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

```
Out[28]: 0.4293249170386153
```

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

```
Out[29]: 0.4260852500438306
```

## 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]: 0.4091480644636627
```

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

```
Out[33]: 0.4147353743176887
```

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

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

```
In [35]: la.score(x_train,y_train)
```

```
Out[35]: 0.1911033062588675
```

## Accuracy(Lasso)

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

```
Out[36]: 0.18055107373375134
```

```
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.          ,  0.15189022,  2.75606498, -0.88422296, -1.16143342,
   -0.01459679, -0.12115579,  0.49450224, -0.05568998, -0.12653083,
    0.          ,  2.3506107 ,  0.          ,  7.70470858])
```

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

```
Out[39]: 28079030.89293577
```

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

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

```
Out[41]: 0.22267594277421798
```

## 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)))
```

30.98206406256929  
1093.058392987938  
33.0614336196714

## Logistic Regression

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

```
In [44]: feature_matrix=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_P10', 'PXY', 'SO_2', 'TCH', 'TOL']]
target_vector=df[ 'station']
```

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

```
Out[45]: (6666, 14)
```

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

```
Out[46]: (6666,)
```

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

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

```
print(prediction)
```

```
[28079099]
```

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

```
Out[52]: array([28079024, 28079099], dtype=int64)
```

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

```
Out[53]: 0.8660366036603661
```

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

```
Out[54]: 0.0
```

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

```
Out[55]: array([[0., 1.]])
```

## 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]: 0.9339905700814402
```

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

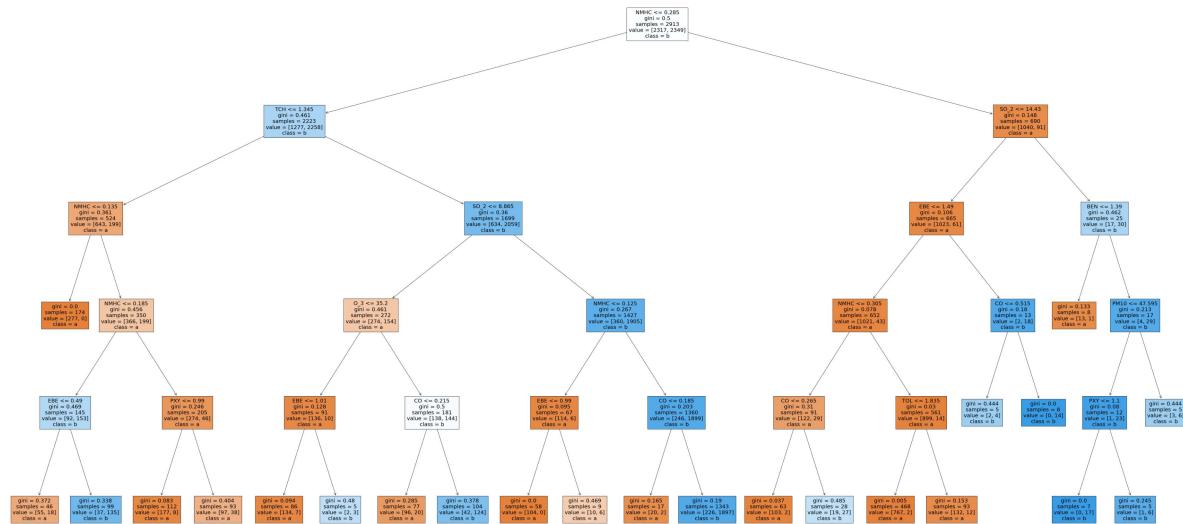
```
In [62]: from sklearn.tree import plot_tree  
  
plt.figure(figsize=(80,40))  
plot_tree(rfc_best.estimators_[5], feature_names=x.columns, class_names=['a', 'b'])
```

```
Out[62]: [Text(2446.6153846153848, 1993.2, 'NMHC <= 0.285\nngini = 0.5\nsamples = 2913\nvalue = [2317, 2349]\nclass = b'),  
Text(1087.3846153846155, 1630.8000000000002, 'TCH <= 1.345\nngini = 0.461\nsamples = 2223\nvalue = [1277, 2258]\nclass = b'),  
Text(343.3846153846154, 1268.4, 'NMHC <= 0.135\nngini = 0.361\nsamples = 524\nvalue = [643, 199]\nclass = a'),  
Text(228.92307692307693, 906.0, 'gini = 0.0\nsamples = 174\nvalue = [277, 0]\nclass = a'),  
Text(457.84615384615387, 906.0, 'NMHC <= 0.185\nngini = 0.456\nsamples = 350\nvalue = [366, 199]\nclass = a'),  
Text(228.92307692307693, 543.5999999999999, 'EBE <= 0.49\nngini = 0.469\nsamples = 145\nvalue = [92, 153]\nclass = b'),  
Text(114.46153846153847, 181.1999999999982, 'gini = 0.372\nsamples = 46\nvalue = [55, 18]\nclass = a'),  
Text(343.3846153846154, 181.1999999999982, 'gini = 0.338\nsamples = 99\nvalue = [37, 135]\nclass = b'),  
Text(686.7692307692308, 543.5999999999999, 'PXY <= 0.99\nngini = 0.246\nsamples = 205\nvalue = [274, 46]\nclass = a'),  
Text(572.3076923076924, 181.1999999999982, 'gini = 0.083\nsamples = 112\nvalue = [177, 8]\nclass = a'),  
Text(801.2307692307693, 181.1999999999982, 'gini = 0.404\nsamples = 93\nvalue = [97, 38]\nclass = a'),  
Text(1831.3846153846155, 1268.4, 'SO_2 <= 8.865\nngini = 0.36\nsamples = 1699\nvalue = [634, 2059]\nclass = b'),  
Text(1373.5384615384617, 906.0, 'O_3 <= 35.2\nngini = 0.461\nsamples = 272\nvalue = [274, 154]\nclass = a'),  
Text(1144.6153846153848, 543.5999999999999, 'EBE <= 1.01\nngini = 0.128\nsamples = 91\nvalue = [136, 10]\nclass = a'),  
Text(1030.1538461538462, 181.1999999999982, 'gini = 0.094\nsamples = 86\nvalue = [134, 7]\nclass = a'),  
Text(1259.076923076923, 181.1999999999982, 'gini = 0.48\nsamples = 5\nvalue = [2, 3]\nclass = b'),  
Text(1602.4615384615386, 543.5999999999999, 'CO <= 0.215\nngini = 0.5\nsamples = 181\nvalue = [138, 144]\nclass = b'),  
Text(1488.0, 181.1999999999982, 'gini = 0.285\nsamples = 77\nvalue = [96, 20]\nclass = a'),  
Text(1716.923076923077, 181.1999999999982, 'gini = 0.378\nsamples = 104\nvalue = [42, 124]\nclass = b'),  
Text(2289.2307692307695, 906.0, 'NMHC <= 0.125\nngini = 0.267\nsamples = 1427\nvalue = [360, 1905]\nclass = b'),  
Text(2060.3076923076924, 543.5999999999999, 'EBE <= 0.99\nngini = 0.095\nsamples = 67\nvalue = [114, 6]\nclass = a'),  
Text(1945.846153846154, 181.1999999999982, 'gini = 0.0\nsamples = 58\nvalue = [104, 0]\nclass = a'),  
Text(2174.769230769231, 181.1999999999982, 'gini = 0.469\nsamples = 9\nvalue = [10, 6]\nclass = a'),  
Text(2518.153846153846, 543.5999999999999, 'CO <= 0.185\nngini = 0.203\nsamples = 1360\nvalue = [246, 1899]\nclass = b'),  
Text(2403.6923076923076, 181.1999999999982, 'gini = 0.165\nsamples = 17\nvalue = [20, 2]\nclass = a'),  
Text(2632.6153846153848, 181.1999999999982, 'gini = 0.19\nsamples = 1343\nvalue = [226, 1897]\nclass = b'),  
Text(3805.846153846154, 1630.8000000000002, 'SO_2 <= 14.43\nngini = 0.148\nsamples = 690\nvalue = [1040, 91]\nclass = a'),  
Text(3491.0769230769233, 1268.4, 'EBE <= 1.49\nngini = 0.106\nsamples = 665\nvalue = [1023, 61]\nclass = a'),  
Text(3204.923076923077, 906.0, 'NMHC <= 0.305\nngini = 0.078\nsamples = 652\nvalue = [1023, 61]\nclass = a')]
```

```

value = [1021, 43]\nclass = a'),
Text(2976.0, 543.5999999999999, 'CO <= 0.265\ngini = 0.31\ncount = 91\nclass = [122, 29]\nclass = a'),
Text(2861.538461538462, 181.19999999999982, 'gini = 0.037\ncount = 63\nclass = [103, 2]\nclass = a'),
Text(3090.4615384615386, 181.19999999999982, 'gini = 0.485\ncount = 28\nclass = [19, 27]\nclass = b'),
Text(3433.846153846154, 543.5999999999999, 'TOL <= 1.835\ngini = 0.03\ncount = 561\nclass = [899, 14]\nclass = a'),
Text(3319.3846153846157, 181.19999999999982, 'gini = 0.005\ncount = 468\nclass = [767, 2]\nclass = a'),
Text(3548.3076923076924, 181.19999999999982, 'gini = 0.153\ncount = 93\nclass = [132, 12]\nclass = a'),
Text(3777.2307692307695, 906.0, 'CO <= 0.515\ngini = 0.18\ncount = 13\nclass = [2, 18]\nclass = b'),
Text(3662.769230769231, 543.5999999999999, 'gini = 0.444\ncount = 5\nclass = [2, 4]\nclass = b'),
Text(3891.692307692308, 543.5999999999999, 'gini = 0.0\ncount = 8\nclass = [0, 14]\nclass = b'),
Text(4120.615384615385, 1268.4, 'BEN <= 1.39\ngini = 0.462\ncount = 25\nclass = [17, 30]\nclass = b'),
Text(4006.153846153846, 906.0, 'gini = 0.133\ncount = 8\nclass = [13, 1]\nclass = a'),
Text(4235.076923076923, 906.0, 'PM10 <= 47.595\ngini = 0.213\ncount = 17\nclass = [4, 29]\nclass = b'),
Text(4120.615384615385, 543.5999999999999, 'PXY <= 1.1\ngini = 0.08\ncount = 12\nclass = [1, 23]\nclass = b'),
Text(4006.153846153846, 181.19999999999982, 'gini = 0.0\ncount = 7\nclass = [0, 17]\nclass = b'),
Text(4235.076923076923, 181.19999999999982, 'gini = 0.245\ncount = 5\nclass = [1, 6]\nclass = b'),
Text(4349.538461538462, 543.5999999999999, 'gini = 0.444\ncount = 5\nclass = [3, 6]\nclass = b')]

```



## Conclusion

## Accuracy

```
In [63]: lr.score(x_train,y_train)
```

```
Out[63]: 0.4260852500438306
```

```
In [64]: rr.score(x_train,y_train)
```

```
Out[64]: 0.4147353743176887
```

```
In [65]: la.score(x_train,y_train)
```

```
Out[65]: 0.1911033062588675
```

```
In [66]: en.score(x_test,y_test)
```

```
Out[66]: 0.22267594277421798
```

```
In [67]: logr.score(fs,target_vector)
```

```
Out[67]: 0.8660366036603661
```

```
In [68]: grid_search.best_score_
```

```
Out[68]: 0.9339905700814402
```

**Random Forest is suitable for this dataset**

```
In [ ]:
```