

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_2008.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.889
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.040
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.270
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.850
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.160
...
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5.450
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15.020
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17.540
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11.910
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12.690

226392 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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      25631 non-null   object 
 1   BEN        25631 non-null   float64
 2   CO         25631 non-null   float64
 3   EBE        25631 non-null   float64
 4   MXY        25631 non-null   float64
 5   NMHC       25631 non-null   float64
 6   NO_2       25631 non-null   float64
 7   NOx        25631 non-null   float64
 8   OXY        25631 non-null   float64
 9   O_3         25631 non-null   float64
 10  PM10       25631 non-null   float64
 11  PM25       25631 non-null   float64
 12  PXY        25631 non-null   float64
 13  SO_2       25631 non-null   float64
 14  TCH         25631 non-null   float64
 15  TOL         25631 non-null   float64
 16  station    25631 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.5+ MB
```

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

Out[6]:

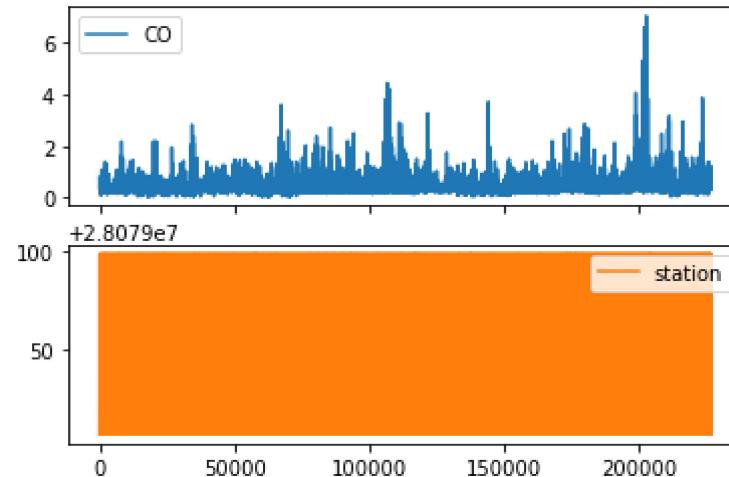
	CO	station
4	0.80	28079006
21	0.37	28079024
25	0.39	28079099
30	0.51	28079006
47	0.39	28079024
...
226362	0.35	28079024
226366	0.46	28079099
226371	0.53	28079006
226387	0.30	28079024
226391	0.36	28079099

25631 rows × 2 columns

Line chart

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

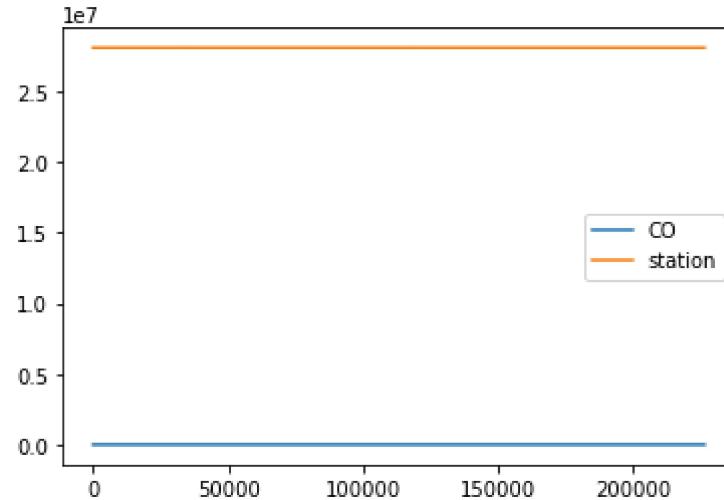
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

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

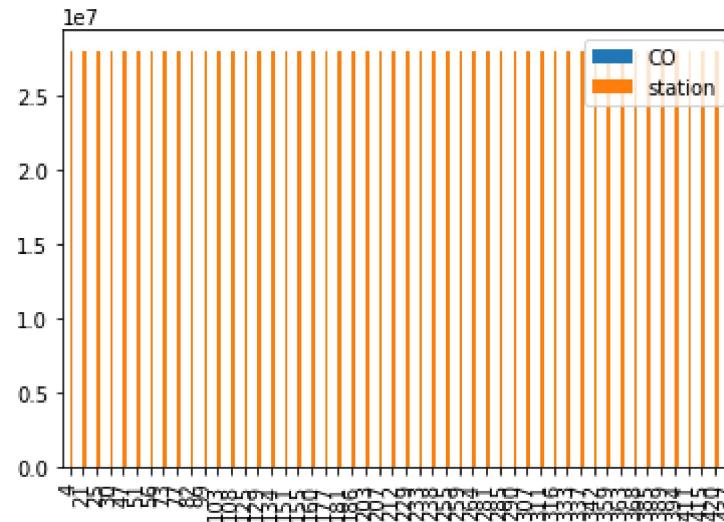


Bar chart

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

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

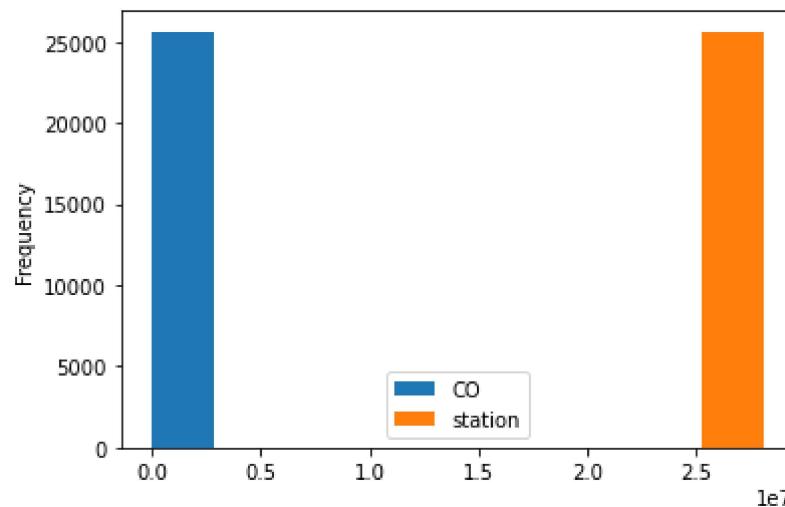
```
Out[10]: <AxesSubplot:>
```



Histogram

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

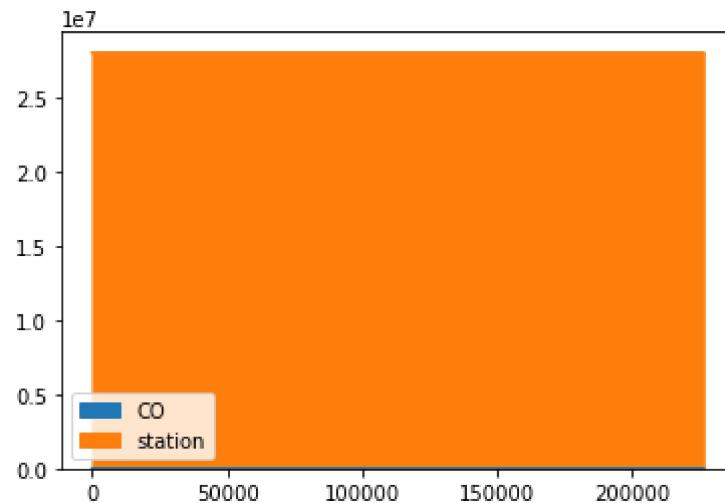
```
Out[11]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

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

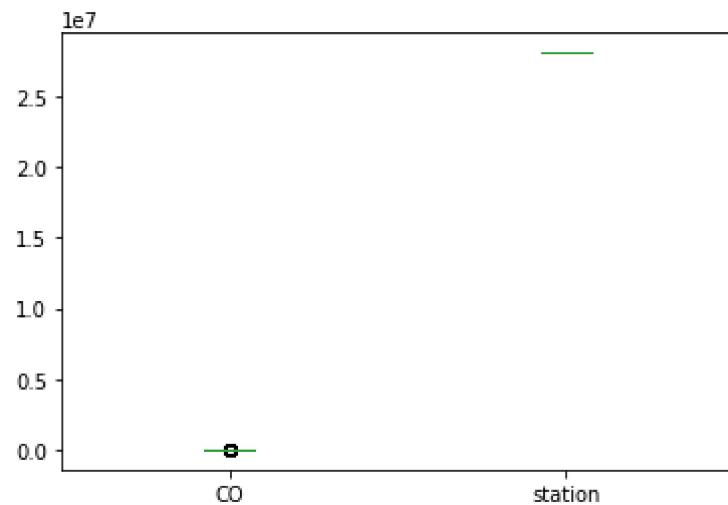
```
Out[12]: <AxesSubplot:>
```



Box chart

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

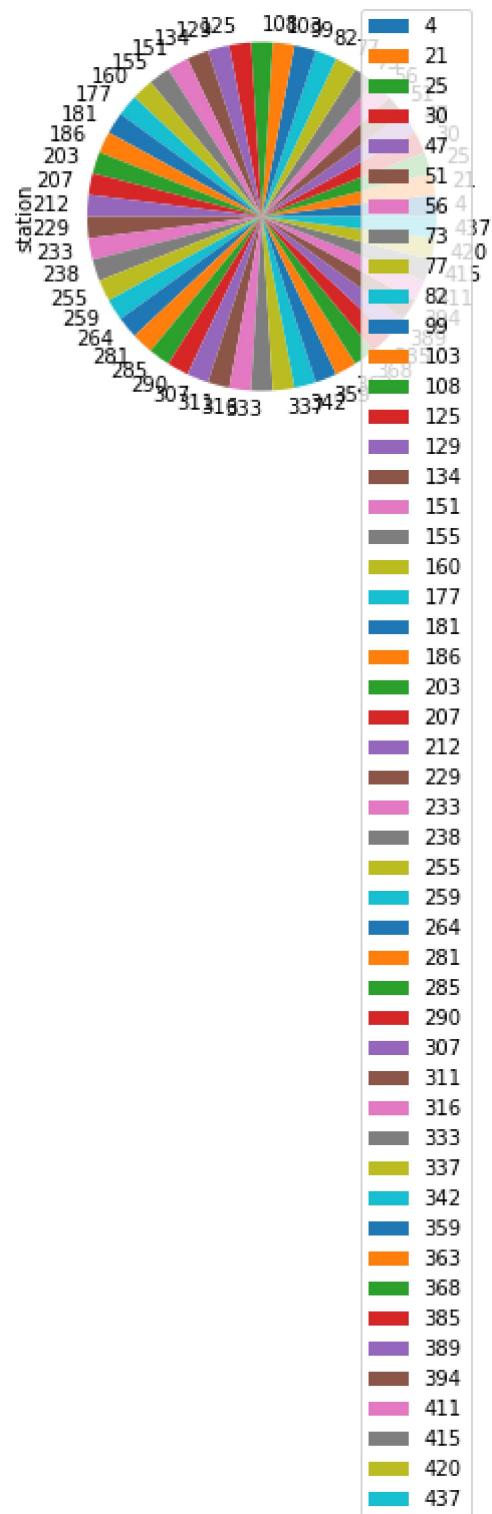
Out[13]: <AxesSubplot:>



Pie chart

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

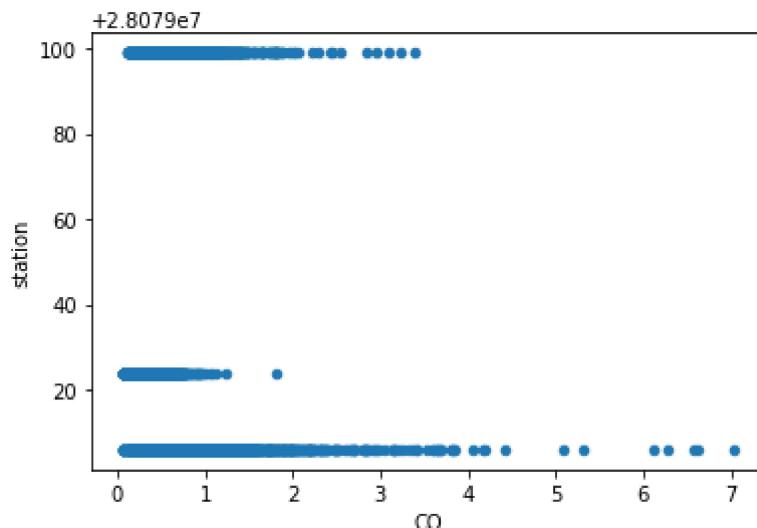
```
Out[14]: <AxesSubplot:ylabel='station'>
```



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: 25631 entries, 4 to 226391
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      25631 non-null   object 
 1   BEN       25631 non-null   float64
 2   CO        25631 non-null   float64
 3   EBE       25631 non-null   float64
 4   MXY       25631 non-null   float64
 5   NMHC      25631 non-null   float64
 6   NO_2      25631 non-null   float64
 7   NOx       25631 non-null   float64
 8   OXY       25631 non-null   float64
 9   O_3        25631 non-null   float64
 10  PM10      25631 non-null   float64
 11  PM25      25631 non-null   float64
 12  PXY       25631 non-null   float64
 13  SO_2      25631 non-null   float64
 14  TCU       25631 non-null   float64
```

In [17]: df.describe()

Out[17]:

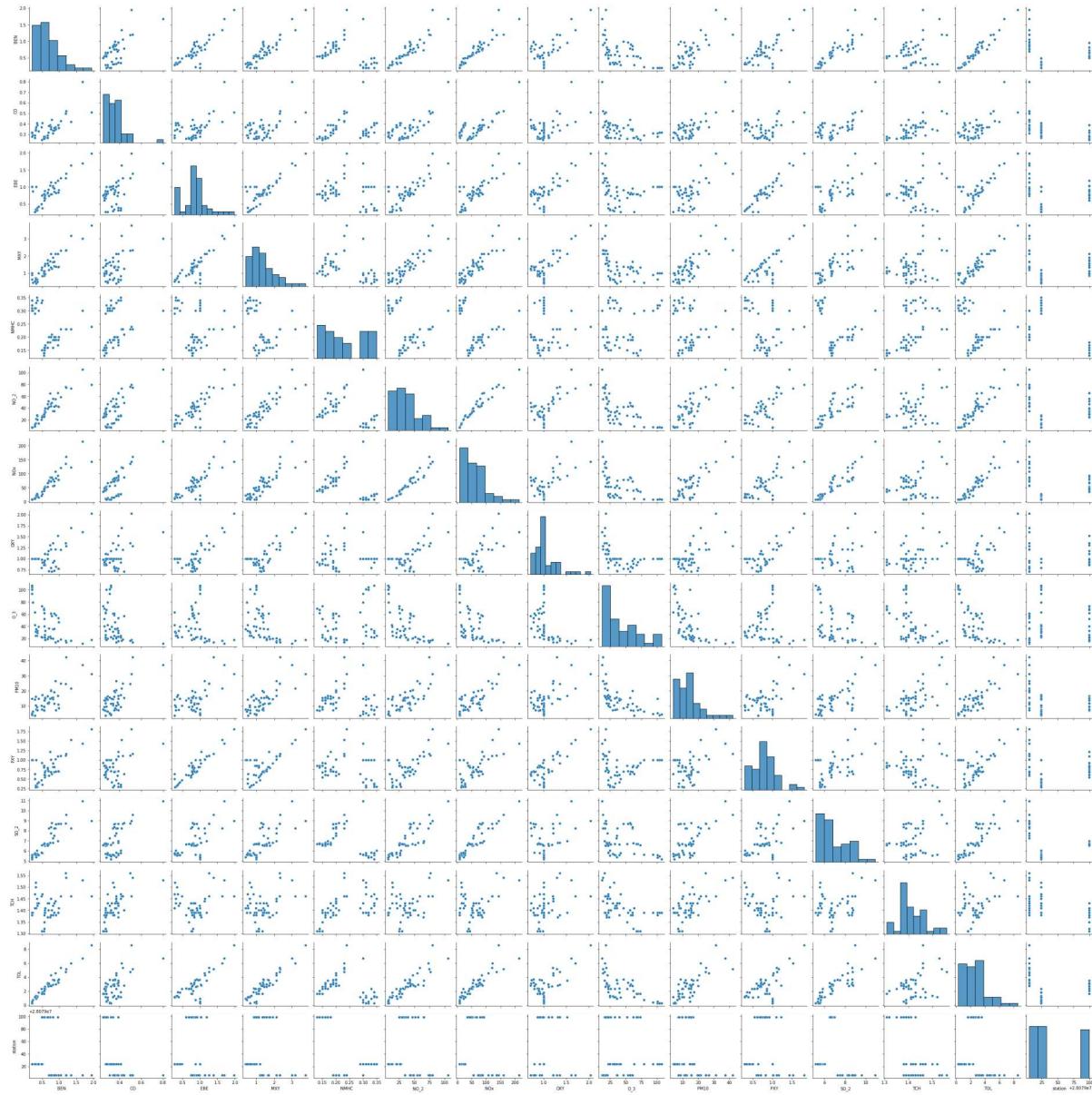
	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	O3	PM10	PXY	SO_2	TCH	TOL	station
count	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000	25631.000000
mean	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261	1.090541	0.440632	1.352355	2.446045	0.213323	54.225261	1.090541	0.440632
std	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647	1.146461	0.317853	1.118191	2.390023	0.123409	38.164647	1.146461	0.317853
min	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000	0.100000	0.060000	0.170000	0.240000	0.000000	0.240000	0.100000	0.060000
25%	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999	0.430000	0.260000	0.740000	1.000000	0.130000	25.719999	0.430000	0.260000
50%	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000	0.750000	0.350000	1.000000	1.620000	0.190000	48.000000	0.750000	0.350000
75%	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999	1.320000	0.510000	1.580000	3.105000	0.270000	74.924999	1.320000	0.510000
max	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024	27.230000	7.030000	26.740000	55.889999	1.760000	554.900024	27.230000	7.030000

In [18]: df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'O3', 'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]

EDA AND VISUALIZATION

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

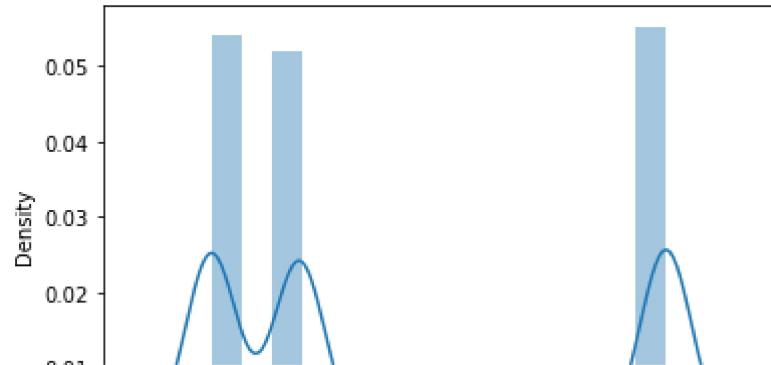
```
Out[19]: <seaborn.axisgrid.PairGrid at 0x1d11d294a90>
```



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

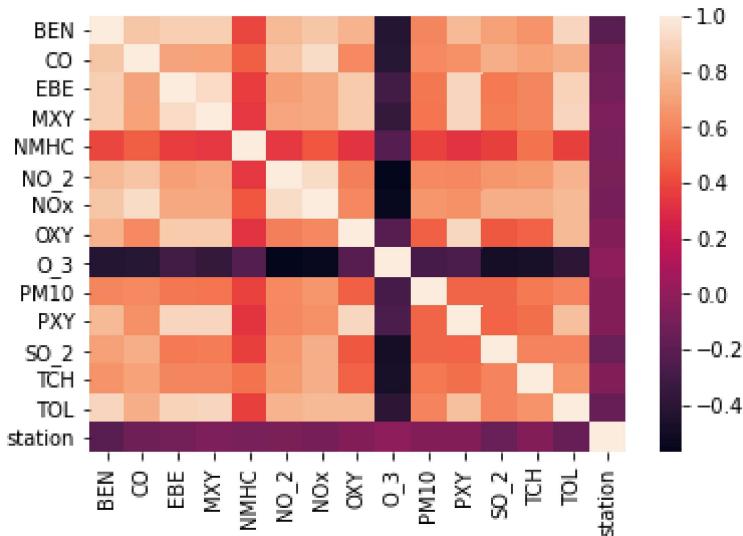
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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]: 28079031.065470487
```

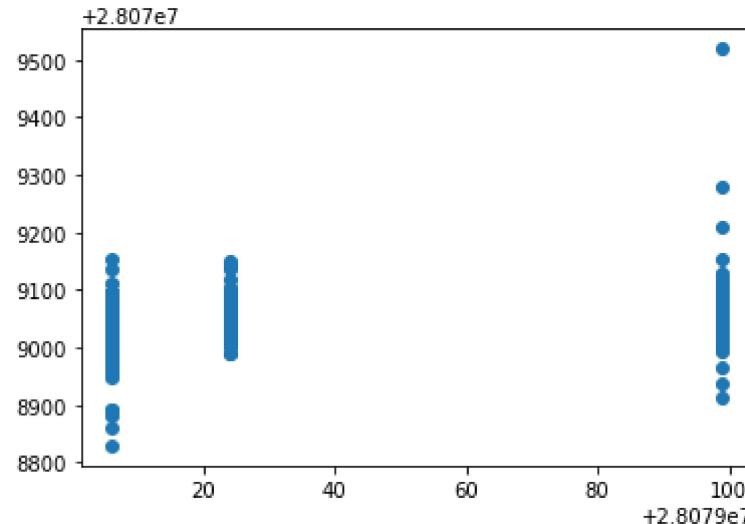
```
In [26]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[26]:
```

	Co-efficient
BEN	-26.132920
CO	-4.208156
EBE	-2.753468
MXY	6.926832
NMHC	-26.959804
NO_2	-0.065479
NOx	0.146133
OXY	5.443002
O_3	-0.149331
PM10	0.135665
PXY	4.763087
SO_2	-0.583942
TCH	20.769396
TOL	-2.006514

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

```
Out[27]: <matplotlib.collections.PathCollection at 0x1d12bccf340>
```



ACCURACY

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

```
Out[28]: 0.1220761472050651
```

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

```
Out[29]: 0.15051204275392893
```

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]: 0.12232859123145778
```

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

```
Out[33]: 0.15048388945549218
```

Accuracy(Lasso)

```
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]: 0.042848623977528866
```

ElasticNet

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

```
Out[36]: 0.04025127243121873
```

```
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([-4.76204435, -0.          , -0.          ,  3.25343583, -0.          ,
   0.02995354,  0.03565052,  1.85135368, -0.1679361 ,  0.13831068,
  1.74051678, -0.95548433,  0.          , -2.61027311])
```

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

```
Out[39]: 28079058.183745507
```

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

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

```
Out[41]: 0.0917524325732435
```

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)))
```

```
35.73226767703649  
1490.2690657398275  
38.60400323463653
```

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]: (25631, 14)
```

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

```
Out[46]: (25631,)
```

```
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([28079006, 28079024, 28079099], dtype=int64)
```

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

```
Out[53]: 0.794194530061254
```

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

```
Out[54]: 8.321803242555043e-09
```

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

```
Out[55]: array([[8.32180324e-09, 1.19114634e-13, 9.99999992e-01]])
```

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="acc  
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]: 0.8502868527064851
```

```
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',
```

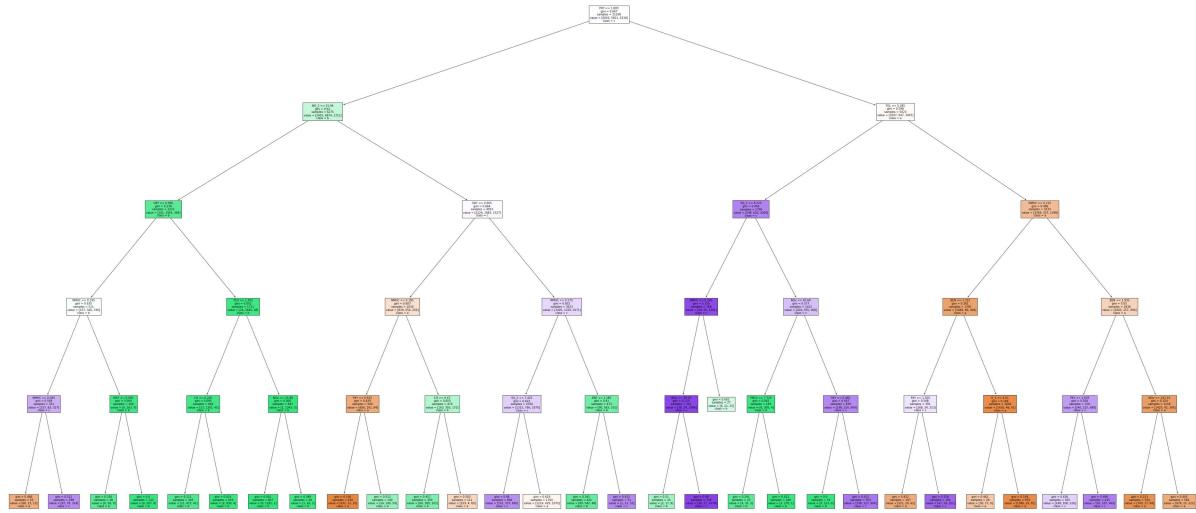
```
Out[62]: [Text(2259.9, 1993.2, 'PXY <= 1.005\ngini = 0.667\nsamples = 11298\nvalue = [6002, 5821, 6118]\nnclass = c'),  
Text(1190.4, 1630.800000000002, 'NO_2 <= 21.94\ngini = 0.63\nsamples = 6275\nvalue = [2405, 4874, 2711]\nnclass = b'),  
Text(595.2, 1268.4, 'OXY <= 0.965\ngini = 0.278\nsamples = 2222\nvalue = [181, 2991, 384]\nnclass = b'),  
Text(297.6, 906.0, 'NMHC <= 0.195\ngini = 0.635\nsamples = 511\nvalue = [157, 346, 336]\nnclass = b'),  
Text(148.8, 543.599999999999, 'NMHC <= 0.065\ngini = 0.569\nsamples = 351\nvalue = [157, 83, 327]\nnclass = c'),  
Text(74.4, 181.1999999999982, 'gini = 0.468\nsamples = 53\nvalue = [60, 13, 13]\nnclass = a'),  
Text(223.200000000002, 181.1999999999982, 'gini = 0.512\nsamples = 298\nvalue = [97, 70, 314]\nnclass = c'),  
Text(446.400000000003, 543.599999999999, 'MXY <= 0.995\ngini = 0.064\nsamples = 160\nvalue = [0, 263, 9]\nnclass = b'),  
Text(372.0, 181.1999999999982, 'gini = 0.239\nsamples = 38\nvalue = [0, 56, 9]\nnclass = b'),  
Text(520.800000000001, 181.1999999999982, 'gini = 0.0\nsamples = 122\nvalue = [0, 207, 0]\nnclass = b'),  
Text(892.800000000001, 906.0, 'TCH <= 1.355\ngini = 0.052\nsamples = 1711\nvalue = [24, 2645, 48]\nnclass = b'),  
Text(744.0, 543.599999999999, 'CO <= 0.245\ngini = 0.095\nsamples = 864\nvalue = [23, 1302, 45]\nnclass = b'),  
Text(669.6, 181.1999999999982, 'gini = 0.211\nsamples = 345\nvalue = [21, 472, 41]\nnclass = b'),  
Text(818.400000000001, 181.1999999999982, 'gini = 0.014\nsamples = 519\nvalue = [2, 830, 4]\nnclass = b'),  
Text(1041.600000000001, 543.599999999999, 'NOx <= 25.88\ngini = 0.006\nsamples = 847\nvalue = [1, 1343, 3]\nnclass = b'),  
Text(967.2, 181.1999999999982, 'gini = 0.002\nsamples = 807\nvalue = [0, 1281, 1]\nnclass = b'),  
Text(1116.0, 181.1999999999982, 'gini = 0.089\nsamples = 40\nvalue = [1, 62, 2]\nnclass = b'),  
Text(1785.600000000001, 1268.4, 'OXY <= 0.665\ngini = 0.664\nsamples = 4053\nvalue = [2224, 1883, 2327]\nnclass = c'),  
Text(1488.0, 906.0, 'NMHC <= 0.185\ngini = 0.607\nsamples = 1030\nvalue = [819, 554, 256]\nnclass = a'),  
Text(1339.2, 543.599999999999, 'PXY <= 0.615\ngini = 0.429\nsamples = 560\nvalue = [656, 161, 84]\nnclass = a'),  
Text(1264.800000000002, 181.1999999999982, 'gini = 0.106\nsamples = 420\nvalue = [632, 12, 25]\nnclass = a'),  
Text(1413.600000000001, 181.1999999999982, 'gini = 0.512\nsamples = 140\nvalue = [24, 149, 59]\nnclass = b'),  
Text(1636.800000000002, 543.599999999999, 'CO <= 0.47\ngini = 0.603\nsamples = 470\nvalue = [163, 393, 172]\nnclass = b'),  
Text(1562.4, 181.1999999999982, 'gini = 0.457\nsamples = 359\nvalue = [60, 389, 103]\nnclass = b'),  
Text(1711.2, 181.1999999999982, 'gini = 0.503\nsamples = 111\nvalue = [103, 4, 69]\nnclass = a'),  
Text(2083.200000000003, 906.0, 'NMHC <= 0.275\ngini = 0.652\nsamples = 3023\nvalue = [1405, 1329, 2071]\nnclass = c'),  
Text(1934.4, 543.599999999999, 'SO_2 <= 7.425\ngini = 0.622\nsamples = 2550\nvalue = [1315, 766, 1970]\nnclass = c'),  
Text(1860.000000000002, 181.1999999999982, 'gini = 0.48\nsamples = 846\nvalue = [101, 337, 895]\nnclass = c'),  
Text(2008.800000000002, 181.1999999999982, 'gini = 0.619\nsamples = 1704\nvalue = [101, 337, 895]\nnclass = c')]
```

```

value = [1214, 429, 1075]\nclass = a'),
Text(2232.0, 543.5999999999999, 'EBE <= 1.185\ngini = 0.41\nsamples = 473\nvalue = [90, 563, 101]\nclass = b'),
Text(2157.600000000004, 181.1999999999982, 'gini = 0.341\nsamples = 422\nvalue = [89, 542, 48]\nclass = b'),
Text(2306.4, 181.1999999999982, 'gini = 0.422\nsamples = 51\nvalue = [1, 21, 53]\nclass = c'),
Text(3329.4, 1630.800000000002, 'TOL <= 5.285\ngini = 0.598\nsamples = 5023\nvalue = [3597, 947, 3407]\nclass = a'),
Text(2790.0, 1268.4, 'SO_2 <= 8.225\ngini = 0.456\nsamples = 1790\nvalue = [248, 610, 2009]\nclass = c'),
Text(2604.0, 906.0, 'NMHC <= 0.245\ngini = 0.155\nsamples = 768\nvalue = [44, 55, 1101]\nclass = c'),
Text(2529.600000000004, 543.5999999999999, 'NOx <= 29.03\ngini = 0.119\nsamples = 741\nvalue = [38, 34, 1086]\nclass = c'),
Text(2455.200000000003, 181.1999999999982, 'gini = 0.51\nsamples = 15\nvalue = [2, 17, 8]\nclass = b'),
Text(2604.0, 181.1999999999982, 'gini = 0.09\nsamples = 726\nvalue = [36, 17, 1078]\nclass = c'),
Text(2678.4, 543.5999999999999, 'gini = 0.602\nsamples = 27\nvalue = [6, 21, 15]\nclass = b'),
Text(2976.0, 906.0, 'NOx <= 45.64\ngini = 0.577\nsamples = 1022\nvalue = [204, 555, 908]\nclass = c'),
Text(2827.200000000003, 543.5999999999999, 'PM10 <= 7.725\ngini = 0.062\nsamples = 183\nvalue = [6, 305, 4]\nclass = b'),
Text(2752.8, 181.1999999999982, 'gini = 0.291\nsamples = 23\nvalue = [4, 35, 3]\nclass = b'),
Text(2901.600000000004, 181.1999999999982, 'gini = 0.022\nsamples = 160\nvalue = [2, 270, 1]\nclass = b'),
Text(3124.8, 543.5999999999999, 'OXY <= 0.485\ngini = 0.497\nsamples = 839\nvalue = [198, 250, 904]\nclass = c'),
Text(3050.4, 181.1999999999982, 'gini = 0.0\nsamples = 76\nvalue = [0, 123, 0]\nclass = b'),
Text(3199.200000000003, 181.1999999999982, 'gini = 0.422\nsamples = 763\nvalue = [198, 127, 904]\nclass = c'),
Text(3868.8, 1268.4, 'NMHC <= 0.215\ngini = 0.486\nsamples = 3233\nvalue = [3349, 337, 1398]\nclass = a'),
Text(3571.200000000003, 906.0, 'BEN <= 1.315\ngini = 0.361\nsamples = 1395\nvalue = [1684, 80, 404]\nclass = a'),
Text(3422.4, 543.5999999999999, 'PXY <= 1.505\ngini = 0.548\nsamples = 391\nvalue = [268, 34, 313]\nclass = c'),
Text(3348.000000000005, 181.1999999999982, 'gini = 0.422\nsamples = 197\nvalue = [221, 20, 62]\nclass = a'),
Text(3496.8, 181.1999999999982, 'gini = 0.328\nsamples = 194\nvalue = [47, 14, 251]\nclass = c'),
Text(3720.000000000005, 543.5999999999999, 'O_3 <= 4.61\ngini = 0.164\ncounts = 1004\nvalue = [1416, 46, 91]\nclass = a'),
Text(3645.600000000004, 181.1999999999982, 'gini = 0.462\ncounts = 28\nvalue = [30, 17, 0]\nclass = a'),
Text(3794.4, 181.1999999999982, 'gini = 0.149\ncounts = 976\nvalue = [1386, 29, 91]\nclass = a'),
Text(4166.400000000001, 906.0, 'BEN <= 1.935\ngini = 0.55\ncounts = 1838\nvalue = [1665, 257, 994]\nclass = a'),
Text(4017.600000000004, 543.5999999999999, 'PXY <= 1.635\ngini = 0.558\ncounts = 720\nvalue = [240, 215, 688]\nclass = c'),
Text(3943.200000000003, 181.1999999999982, 'gini = 0.636\ncounts = 305\nvalue = [148, 108, 226]\nclass = c'),

```

```
Text(4092.000000000005, 181.19999999999982, 'gini = 0.466\nsamples = 415\nvalue = [92, 107, 462]\nclass = c'),
Text(4315.200000000001, 543.5999999999999, 'NOx <= 243.15\ngini = 0.324\nsamples = 1118\nvalue = [1425, 42, 306]\nclass = a'),
Text(4240.8, 181.19999999999982, 'gini = 0.213\nsamples = 534\nvalue = [749, 17, 84]\nclass = a'),
Text(4389.6, 181.19999999999982, 'gini = 0.405\nsamples = 584\nvalue = [676, 25, 222]\nclass = a')]
```



Conclusion

Accuracy

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

```
Out[63]: 0.15051204275392893
```

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

```
Out[64]: 0.15048388945549218
```

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

```
Out[65]: 0.042848623977528866
```

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

```
Out[66]: 0.0917524325732435
```

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

```
Out[67]: 0.794194530061254
```

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

```
Out[68]: 0.8502868527064851
```

Random is suitable for this dataset