

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_2009.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.2600
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.5800
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.1900
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.5300
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.7600
...
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.8300
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.9200
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.4600
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.0300
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.3600

215688 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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      24717 non-null   object 
 1   BEN        24717 non-null   float64
 2   CO         24717 non-null   float64
 3   EBE        24717 non-null   float64
 4   MXY        24717 non-null   float64
 5   NMHC       24717 non-null   float64
 6   NO_2       24717 non-null   float64
 7   NOx        24717 non-null   float64
 8   OXY        24717 non-null   float64
 9   O_3         24717 non-null   float64
 10  PM10       24717 non-null   float64
 11  PM25       24717 non-null   float64
 12  PXY        24717 non-null   float64
 13  SO_2       24717 non-null   float64
 14  TCH         24717 non-null   float64
 15  TOL         24717 non-null   float64
 16  station    24717 non-null   int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 3.4+ MB
```

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

Out[6]:

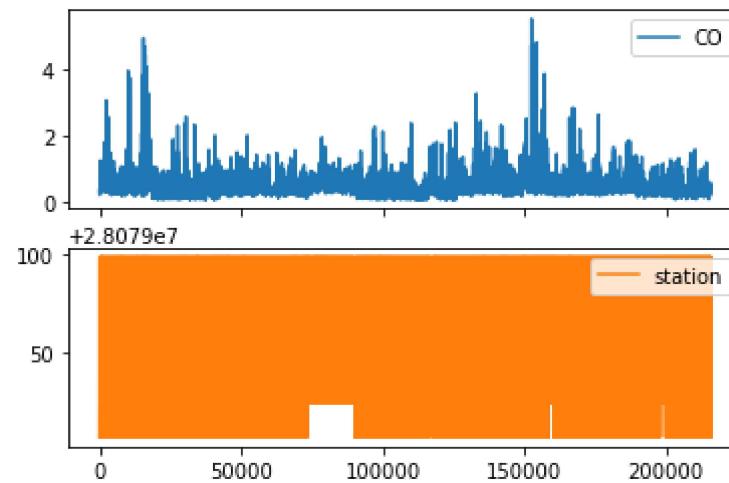
	CO	station
3	0.33	28079006
20	0.32	28079024
24	0.24	28079099
28	0.21	28079006
45	0.30	28079024
...
215659	0.27	28079024
215663	0.35	28079099
215667	0.29	28079006
215683	0.22	28079024
215687	0.32	28079099

24717 rows × 2 columns

Line chart

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

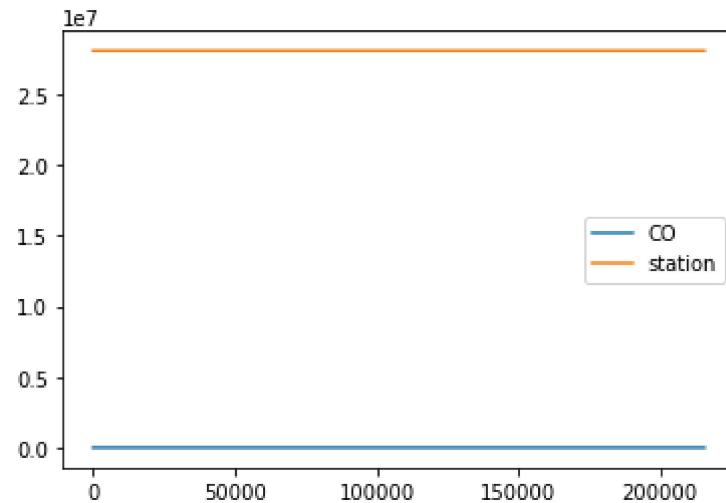
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

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

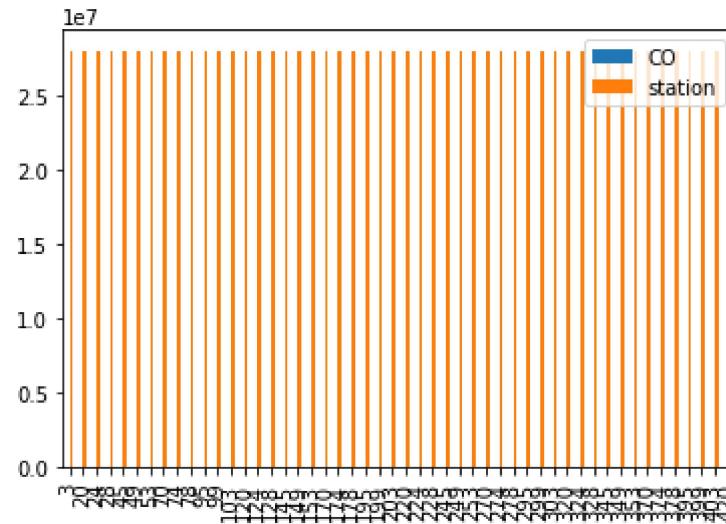


Bar chart

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

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

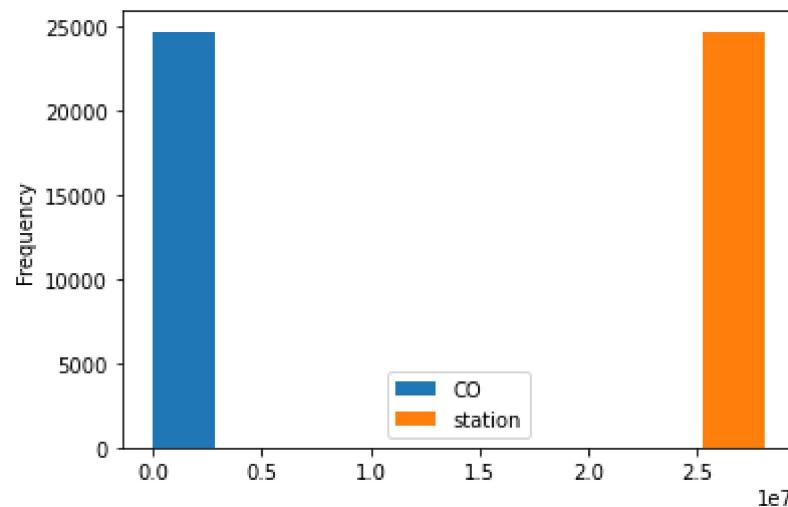
```
Out[10]: <AxesSubplot:>
```



Histogram

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

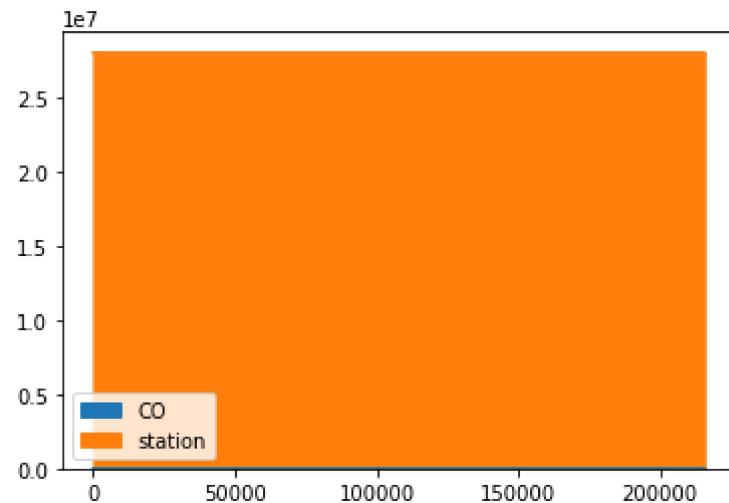
```
Out[11]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

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

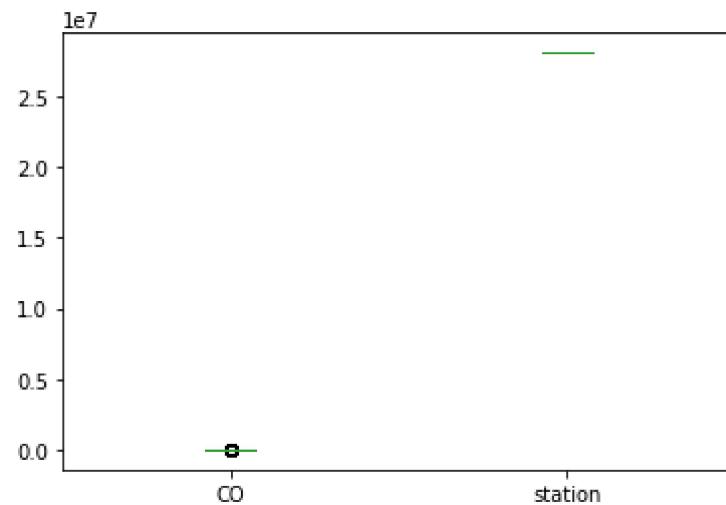
```
Out[12]: <AxesSubplot:>
```



Box chart

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

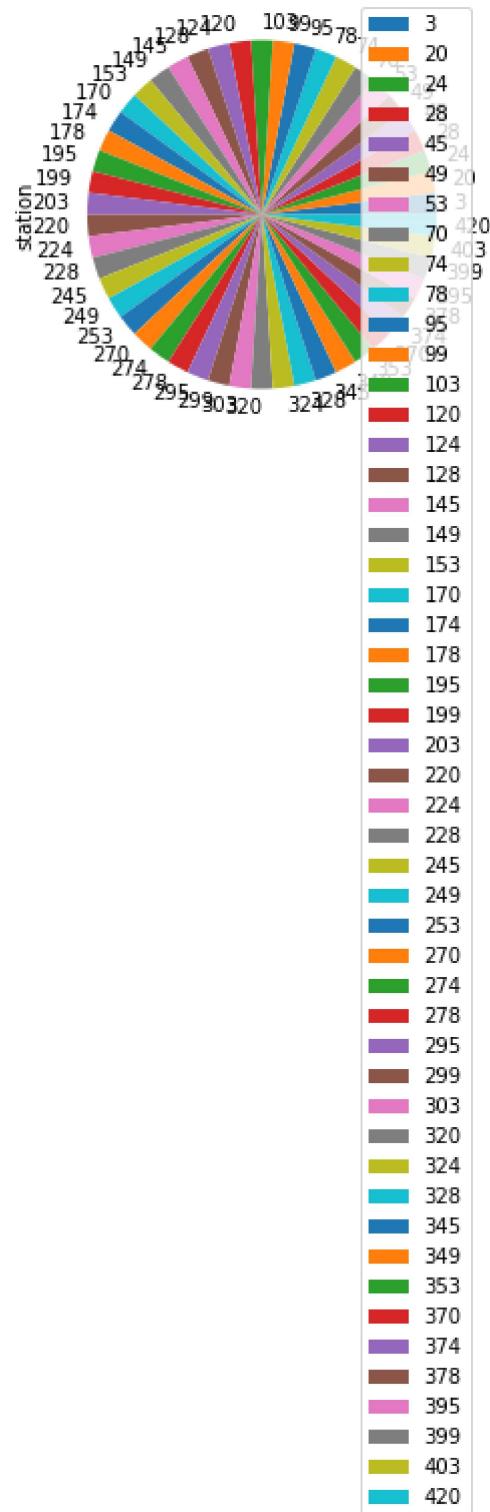
Out[13]: <AxesSubplot:>



Pie chart

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

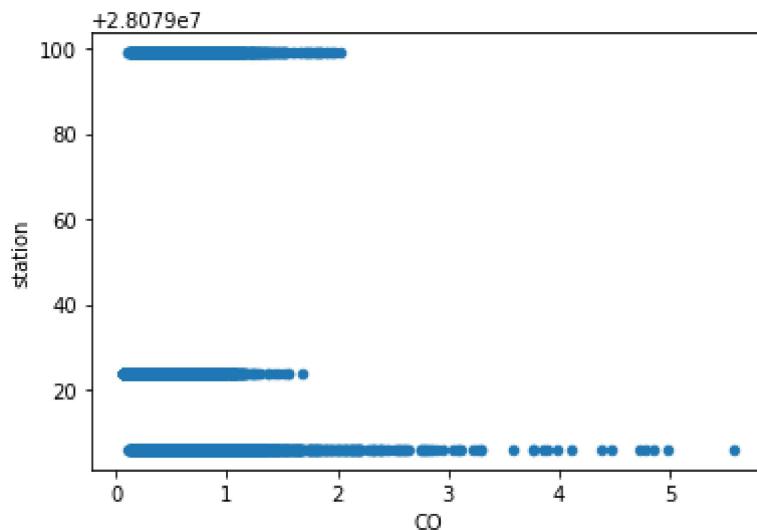
```
Out[14]: <AxesSubplot:ylabel='station'>
```



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: 24717 entries, 3 to 215687
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      24717 non-null   object 
 1   BEN       24717 non-null   float64
 2   CO        24717 non-null   float64
 3   EBE       24717 non-null   float64
 4   MXY       24717 non-null   float64
 5   NMHC      24717 non-null   float64
 6   NO_2      24717 non-null   float64
 7   NOx       24717 non-null   float64
 8   OXY       24717 non-null   float64
 9   O_3        24717 non-null   float64
 10  PM10      24717 non-null   float64
 11  PM25      24717 non-null   float64
 12  PXY       24717 non-null   float64
 13  SO_2      24717 non-null   float64
 14  TCU       24717 non-null   float64
```

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

Out[17]:

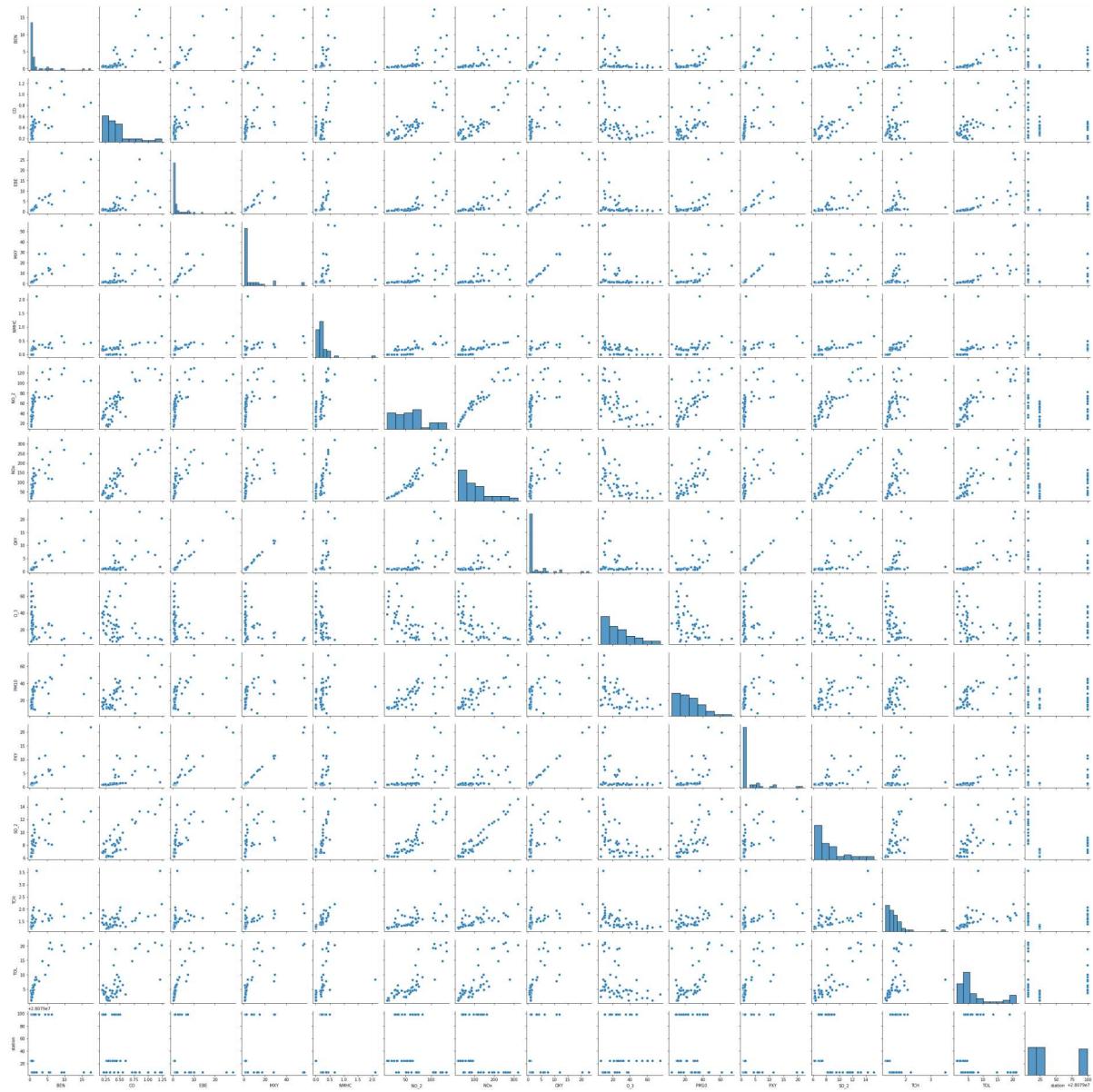
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	24717.000000	247
mean	1.010583	0.448056	1.262430	2.244469	0.219582	55.563929	
std	1.007345	0.291706	1.074768	2.242214	0.141661	38.911677	
min	0.170000	0.060000	0.250000	0.240000	0.000000	0.600000	
25%	0.460000	0.270000	0.720000	0.990000	0.140000	26.510000	
50%	0.670000	0.370000	1.000000	1.490000	0.190000	47.930000	
75%	1.180000	0.570000	1.430000	2.820000	0.260000	76.269997	1
max	22.379999	5.570000	47.669998	56.500000	2.580000	477.399994	14

```
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])
```

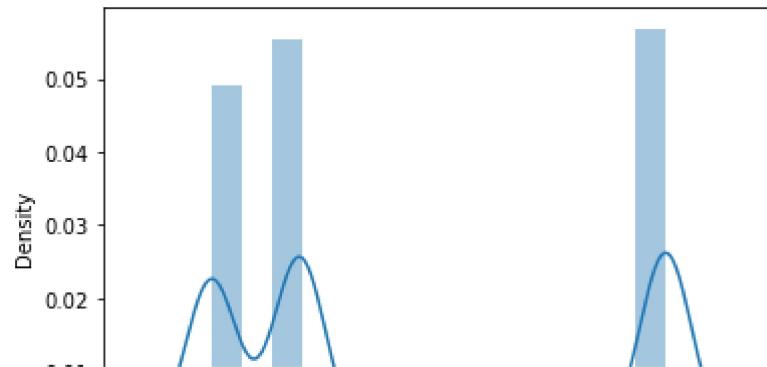
```
Out[19]: <seaborn.axisgrid.PairGrid at 0x1a1f7d40040>
```



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

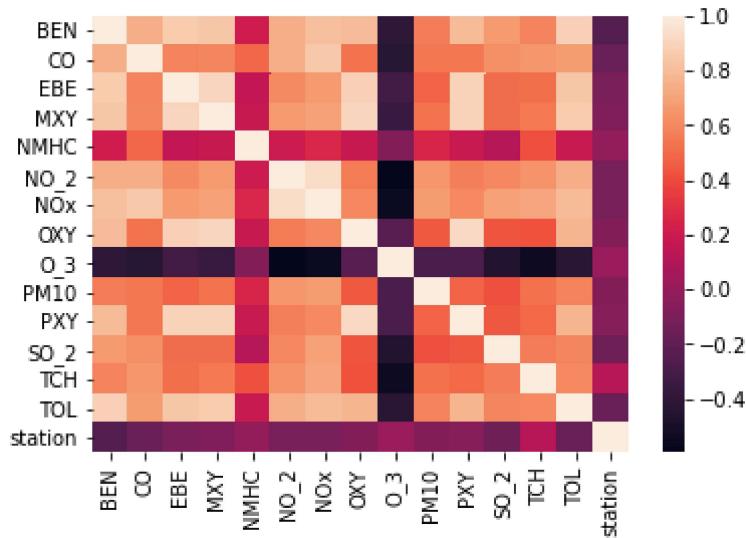
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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]: 28078904.852659103
```

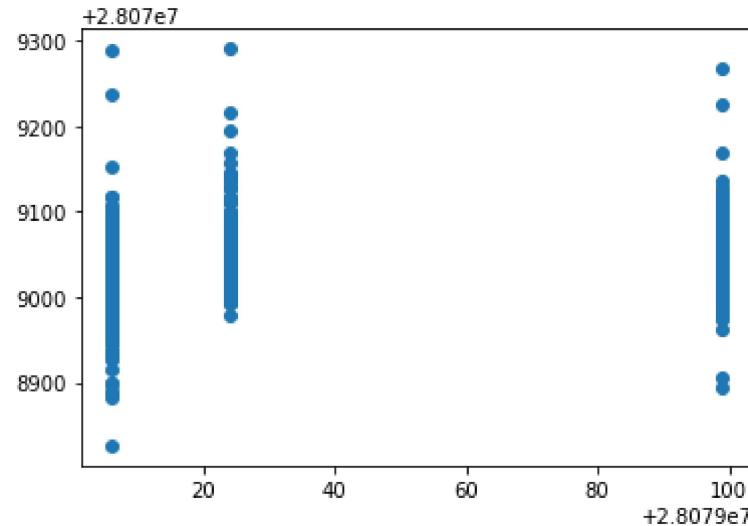
```
In [26]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[26]:
```

	Co-efficient
BEN	-35.587107
CO	-28.413224
EBE	5.831866
MXY	0.212538
NMHC	-17.336057
NO_2	-0.182930
NOx	0.194997
OXY	13.312325
O_3	0.014328
PM10	-0.056213
PXY	2.110024
SO_2	-0.422879
TCH	116.346794
TOL	-1.131840

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

```
Out[27]: <matplotlib.collections.PathCollection at 0x1a187e31b50>
```



ACCURACY

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

```
Out[28]: 0.2906184326381347
```

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

```
Out[29]: 0.2850937428589063
```

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]: 0.2897207247743636
```

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

```
Out[33]: 0.28480514743807195
```

```
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]: 0.035095419882119616
```

Accuracy(Lasso)

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

```
Out[36]: 0.032496528275414605
```

```
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([-6.93968226, -0.66695477,  0.20411009,  2.56648335, -0.
                 -0.23388424,  0.1261577 ,  1.33183377, -0.1394248 ,  0.0668451 ,
                 1.91209004, -0.85409164,  1.48692704, -1.98736381])
```

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

```
Out[39]: 28079063.935898777
```

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

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

```
Out[41]: 0.10446131845923845
```

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)))
```

35.97246041991261
1482.7298419160634
38.50623120893634

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]: (24717, 14)
```

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

```
Out[46]: (24717,)
```

```
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]: 0.8951733624630821
```

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

```
Out[54]: 5.447205522232353e-13
```

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

```
Out[55]: array([[5.44720552e-13, 8.28692830e-44, 1.00000000e+00]])
```

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]: 0.8987343038293545
```

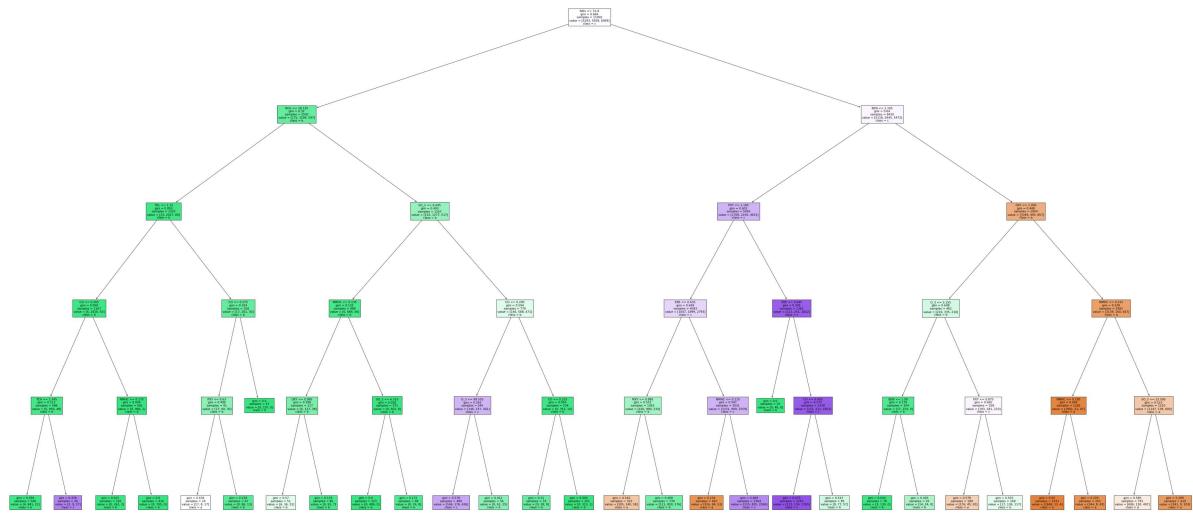
```
In [61]: rfc_best=grid_search.best_estimator_
```

```
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(2192.142857142857, 1993.2, 'NOx <= 31.8\nngini = 0.666\nsamples = 11002\nvalue = [5293, 5939, 6069]\nclass = c'),  
 Text(1096.0714285714284, 1630.8000000000002, 'NOx <= 18.135\nngini = 0.32\nsamples = 2592\nvalue = [175, 3294, 597]\nclass = b'),  
 Text(597.8571428571428, 1268.4, 'TOL <= 1.12\nngini = 0.093\nsamples = 1325\nvalue = [23, 2017, 80]\nclass = b'),  
 Text(318.85714285714283, 906.0, 'CO <= 0.265\nngini = 0.058\nsamples = 1167\nvalue = [6, 1816, 50]\nclass = b'),  
 Text(159.42857142857142, 543.5999999999999, 'TCH <= 1.345\nngini = 0.113\nsamples = 586\nvalue = [6, 850, 48]\nclass = b'),  
 Text(79.71428571428571, 181.1999999999982, 'gini = 0.056\nsamples = 560\nvalue = [4, 841, 21]\nclass = b'),  
 Text(239.1428571428571, 181.1999999999982, 'gini = 0.436\nsamples = 26\nvalue = [2, 9, 27]\nclass = c'),  
 Text(478.2857142857142, 543.5999999999999, 'NMHC <= 0.175\nngini = 0.004\nsamples = 581\nvalue = [0, 966, 2]\nclass = b'),  
 Text(398.57142857142856, 181.1999999999982, 'gini = 0.015\nsamples = 165\nvalue = [0, 261, 2]\nclass = b'),  
 Text(558.0, 181.1999999999982, 'gini = 0.0\nsamples = 416\nvalue = [0, 705, 0]\nclass = b'),  
 Text(876.8571428571428, 906.0, 'CO <= 0.275\nngini = 0.324\nsamples = 158\nvalue = [17, 201, 30]\nclass = b'),  
 Text(797.1428571428571, 543.5999999999999, 'PXY <= 0.63\nngini = 0.496\nsamples = 91\nvalue = [17, 94, 30]\nclass = b'),  
 Text(717.4285714285713, 181.1999999999982, 'gini = 0.636\nsamples = 24\nvalue = [17, 8, 17]\nclass = a'),  
 Text(876.8571428571428, 181.1999999999982, 'gini = 0.228\nsamples = 67\nvalue = [0, 86, 13]\nclass = b'),  
 Text(956.5714285714284, 543.5999999999999, 'gini = 0.0\nsamples = 67\nvalue = [0, 107, 0]\nclass = b'),  
 Text(1594.2857142857142, 1268.4, 'SO_2 <= 6.645\nngini = 0.493\nsamples = 1267\nvalue = [152, 1277, 517]\nclass = b'),  
 Text(1275.4285714285713, 906.0, 'NMHC <= 0.135\nngini = 0.132\nsamples = 488\nvalue = [6, 689, 46]\nclass = b'),  
 Text(1116.0, 543.5999999999999, 'OXY <= 0.995\nngini = 0.398\nsamples = 117\nvalue = [6, 127, 38]\nclass = b'),  
 Text(1036.2857142857142, 181.1999999999982, 'gini = 0.57\nsamples = 51\nvalue = [6, 36, 31]\nclass = b'),  
 Text(1195.7142857142856, 181.1999999999982, 'gini = 0.133\nsamples = 66\nvalue = [0, 91, 7]\nclass = b'),  
 Text(1434.8571428571427, 543.5999999999999, 'SO_2 <= 6.515\nngini = 0.028\nsamples = 371\nvalue = [0, 562, 8]\nclass = b'),  
 Text(1355.142857142857, 181.1999999999982, 'gini = 0.0\nsamples = 323\nvalue = [0, 486, 0]\nclass = b'),  
 Text(1514.5714285714284, 181.1999999999982, 'gini = 0.172\nsamples = 48\nvalue = [0, 76, 8]\nclass = b'),  
 Text(1913.1428571428569, 906.0, 'CO <= 0.295\nngini = 0.594\nsamples = 779\nvalue = [146, 588, 471]\nclass = b'),  
 Text(1753.7142857142856, 543.5999999999999, 'O_3 <= 99.555\nngini = 0.593\nsamples = 545\nvalue = [146, 237, 461]\nclass = c'),  
 Text(1673.999999999998, 181.1999999999982, 'gini = 0.578\nsamples = 489\nvalue = [146, 176, 436]\nclass = c'),  
 Text(1833.4285714285713, 181.1999999999982, 'gini = 0.412\nsamples = 56\nvalue = [0, 61, 25]\nclass = b'),  
 Text(2072.5714285714284, 543.5999999999999, 'CO <= 0.315\nngini = 0.054\nsamples = 234\nvalue = [0, 351, 10]\nclass = b'),  
 Text(1992.8571428571427, 181.1999999999982, 'gini = 0.31\nsamples = 33\nvalue = [0, 100, 10]\nclass = b')]
```

```
ue = [0, 38, 9]\nclass = b'),  
Text(2152.285714285714, 181.19999999999982, 'gini = 0.006\nsamples = 201\nva  
lue = [0, 313, 1]\nclass = b'),  
Text(3288.2142857142853, 1630.8000000000002, 'BEN <= 1.105\ngini = 0.64\nsam  
ples = 8410\nvalue = [5118, 2645, 5472]\nclass = c'),  
Text(2750.142857142857, 1268.4, 'PXY <= 1.185\ngini = 0.601\nsamples = 5456  
\nvalue = [1769, 2150, 4615]\nclass = c'),  
Text(2550.8571428571427, 906.0, 'EBE <= 0.635\ngini = 0.649\nsamples = 4093  
\nvalue = [1647, 1899, 2793]\nclass = c'),  
Text(2391.428571428571, 543.5999999999999, 'MXY <= 0.885\ngini = 0.551\nsam  
ples = 1062\nvalue = [416, 990, 234]\nclass = b'),  
Text(2311.7142857142853, 181.19999999999982, 'gini = 0.542\nsamples = 323\nv  
alue = [305, 140, 58]\nclass = a'),  
Text(2471.142857142857, 181.19999999999982, 'gini = 0.408\nsamples = 739\nv  
alue = [111, 850, 176]\nclass = b'),  
Text(2710.285714285714, 543.5999999999999, 'NMHC <= 0.115\ngini = 0.597\nsam  
ples = 3031\nvalue = [1231, 909, 2559]\nclass = c'),  
Text(2630.5714285714284, 181.19999999999982, 'gini = 0.234\nsamples = 667\nv  
alue = [916, 84, 53]\nclass = a'),  
Text(2790.0, 181.19999999999982, 'gini = 0.469\nsamples = 2364\nvalue = [31  
5, 825, 2506]\nclass = c'),  
Text(2949.428571428571, 906.0, 'EBE <= 0.645\ngini = 0.295\nsamples = 1363\nv  
alue = [122, 251, 1822]\nclass = c'),  
Text(2869.7142857142853, 543.5999999999999, 'gini = 0.0\nsamples = 25\nvalue  
= [0, 40, 0]\nclass = b'),  
Text(3029.142857142857, 543.5999999999999, 'CO <= 0.605\ngini = 0.272\nsam  
ples = 1338\nvalue = [122, 211, 1822]\nclass = c'),  
Text(2949.428571428571, 181.19999999999982, 'gini = 0.223\nsamples = 1253\nv  
alue = [113, 134, 1765]\nclass = c'),  
Text(3108.8571428571427, 181.19999999999982, 'gini = 0.547\nsamples = 85\nv  
alue = [9, 77, 57]\nclass = b'),  
Text(3826.2857142857138, 1268.4, 'OXY <= 1.005\ngini = 0.448\nsamples = 2954  
\nvalue = [3349, 495, 857]\nclass = a'),  
Text(3507.428571428571, 906.0, 'O_3 <= 5.155\ngini = 0.648\nsamples = 462\nv  
alue = [210, 335, 210]\nclass = b'),  
Text(3347.999999999995, 543.5999999999999, 'BEN <= 1.58\ngini = 0.179\nsam  
ples = 104\nvalue = [17, 154, 0]\nclass = b'),  
Text(3268.285714285714, 181.19999999999982, 'gini = 0.044\nsamples = 78\nv  
alue = [3, 130, 0]\nclass = b'),  
Text(3427.7142857142853, 181.19999999999982, 'gini = 0.465\nsamples = 26\nv  
alue = [14, 24, 0]\nclass = b'),  
Text(3666.8571428571427, 543.5999999999999, 'PXY <= 0.875\ngini = 0.665\nsam  
ples = 358\nvalue = [193, 181, 210]\nclass = c'),  
Text(3587.142857142857, 181.19999999999982, 'gini = 0.578\nsamples = 189\nv  
alue = [176, 45, 93]\nclass = a'),  
Text(3746.5714285714284, 181.19999999999982, 'gini = 0.555\nsamples = 169\nv  
alue = [17, 136, 117]\nclass = b'),  
Text(4145.142857142857, 906.0, 'NMHC <= 0.215\ngini = 0.339\nsamples = 2492  
\nvalue = [3139, 160, 647]\nclass = a'),  
Text(3985.7142857142853, 543.5999999999999, 'NMHC <= 0.185\ngini = 0.064\nsa  
mples = 1282\nvalue = [1992, 21, 47]\nclass = a'),  
Text(3905.999999999995, 181.19999999999982, 'gini = 0.02\nsamples = 1031\nv  
alue = [1648, 13, 4]\nclass = a'),  
Text(4065.428571428571, 181.19999999999982, 'gini = 0.229\nsamples = 251\nv  
alue = [344, 8, 43]\nclass = a'),  
Text(4304.571428571428, 543.5999999999999, 'SO_2 <= 22.585\ngini = 0.523\nsa  
mples = 1210\nvalue = [1147, 139, 600]\nclass = a'),
```

```
Text(4224.857142857142, 181.19999999999982, 'gini = 0.585\nsamples = 791\nvalue = [606, 130, 497]\nclass = a'),
Text(4384.285714285714, 181.19999999999982, 'gini = 0.289\nsamples = 419\nvalue = [541, 9, 103]\nclass = a'))]
```



Conclusion

Accuracy

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

```
Out[63]: 0.2850937428589063
```

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

```
Out[64]: 0.28480514743807195
```

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

```
Out[65]: 0.035095419882119616
```

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

```
Out[66]: 0.10446131845923845
```

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

```
Out[67]: 0.8951733624630821
```

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

```
Out[68]: 0.8987343038293545
```

Random Forest is suitable for this dataset

In []: