

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_2018.csv")
df
```

Out[2]:

| | | date | BEN | CH4 | CO | EBE | NMHC | NO | NO_2 | NOx | O_3 | PM10 | PM25 | SO_2 | T |
|-------|--|---------------------|-----|------|-----|-----|------|-------|-------|-------|------|------|------|------|-----|
| 0 | | 2018-03-01 01:00:00 | NaN | NaN | 0.3 | NaN | NaN | 1.0 | 29.0 | 31.0 | NaN | NaN | NaN | 2.0 | N |
| 1 | | 2018-03-01 01:00:00 | 0.5 | 1.39 | 0.3 | 0.2 | 0.02 | 6.0 | 40.0 | 49.0 | 52.0 | 5.0 | 4.0 | 3.0 | 1 |
| 2 | | 2018-03-01 01:00:00 | 0.4 | NaN | NaN | 0.2 | NaN | 4.0 | 41.0 | 47.0 | NaN | NaN | NaN | NaN | N |
| 3 | | 2018-03-01 01:00:00 | NaN | NaN | 0.3 | NaN | NaN | 1.0 | 35.0 | 37.0 | 54.0 | NaN | NaN | NaN | N |
| 4 | | 2018-03-01 01:00:00 | NaN | NaN | NaN | NaN | NaN | 1.0 | 27.0 | 29.0 | 49.0 | NaN | NaN | 3.0 | N |
| ... | | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 69091 | | 2018-02-01 00:00:00 | NaN | NaN | 0.5 | NaN | NaN | 66.0 | 91.0 | 192.0 | 1.0 | 35.0 | 22.0 | NaN | N |
| 69092 | | 2018-02-01 00:00:00 | NaN | NaN | 0.7 | NaN | NaN | 87.0 | 107.0 | 241.0 | NaN | 29.0 | NaN | 15.0 | N |
| 69093 | | 2018-02-01 00:00:00 | NaN | NaN | NaN | NaN | NaN | 28.0 | 48.0 | 91.0 | 2.0 | NaN | NaN | NaN | N |
| 69094 | | 2018-02-01 00:00:00 | NaN | NaN | NaN | NaN | NaN | 141.0 | 103.0 | 320.0 | 2.0 | NaN | NaN | NaN | N |
| 69095 | | 2018-02-01 00:00:00 | NaN | NaN | NaN | NaN | NaN | 69.0 | 96.0 | 202.0 | 3.0 | 26.0 | NaN | NaN | N |

69096 rows × 16 columns



Data Cleaning and Data Preprocessing

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

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

```
Out[4]: Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4562 entries, 1 to 69078
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   date      4562 non-null   object 
 1   BEN        4562 non-null   float64
 2   CH4        4562 non-null   float64
 3   CO         4562 non-null   float64
 4   EBE        4562 non-null   float64
 5   NMHC       4562 non-null   float64
 6   NO         4562 non-null   float64
 7   NO_2       4562 non-null   float64
 8   NOx        4562 non-null   float64
 9   O_3         4562 non-null   float64
 10  PM10       4562 non-null   float64
 11  PM25       4562 non-null   float64
 12  SO_2       4562 non-null   float64
 13  TCH         4562 non-null   float64
 14  TOL         4562 non-null   float64
 15  station    4562 non-null   int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 605.9+ KB
```

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

Out[6]:

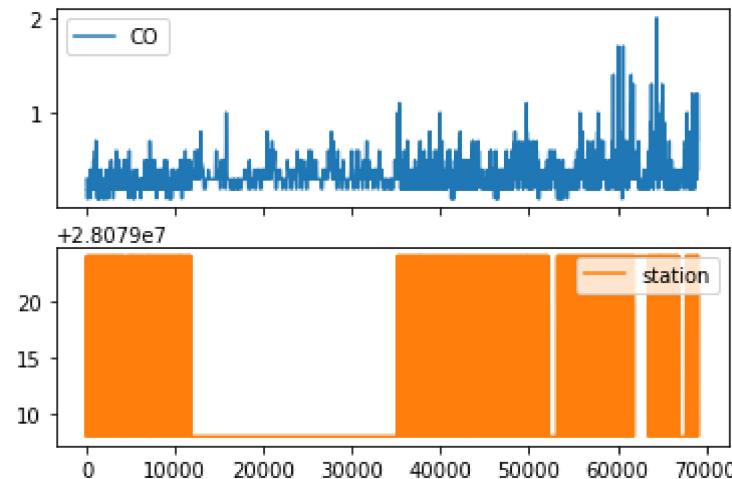
| | CO | station |
|-------|-----|----------|
| 1 | 0.3 | 28079008 |
| 6 | 0.2 | 28079024 |
| 25 | 0.2 | 28079008 |
| 30 | 0.2 | 28079024 |
| 49 | 0.2 | 28079008 |
| ... | ... | ... |
| 69030 | 0.7 | 28079024 |
| 69049 | 1.2 | 28079008 |
| 69054 | 0.6 | 28079024 |
| 69073 | 1.0 | 28079008 |
| 69078 | 0.4 | 28079024 |

4562 rows × 2 columns

Line chart

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

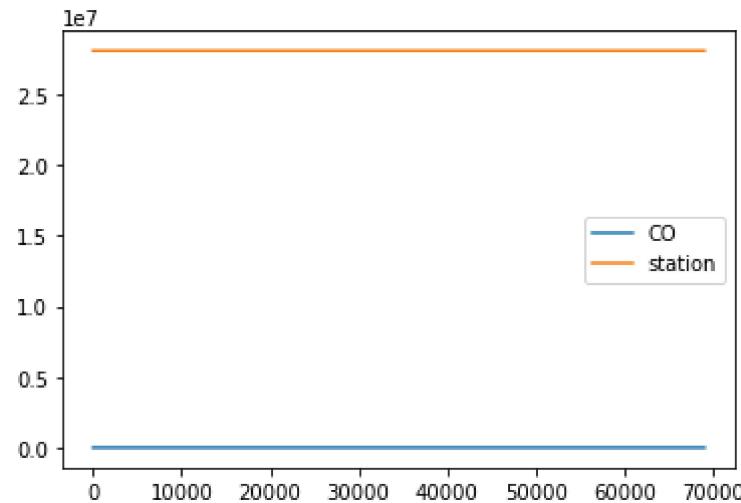
Out[7]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)



Line chart

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

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

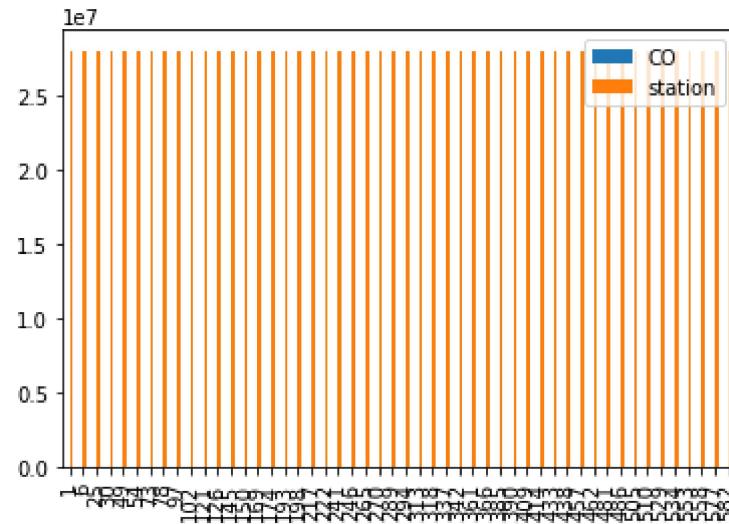


Bar chart

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

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

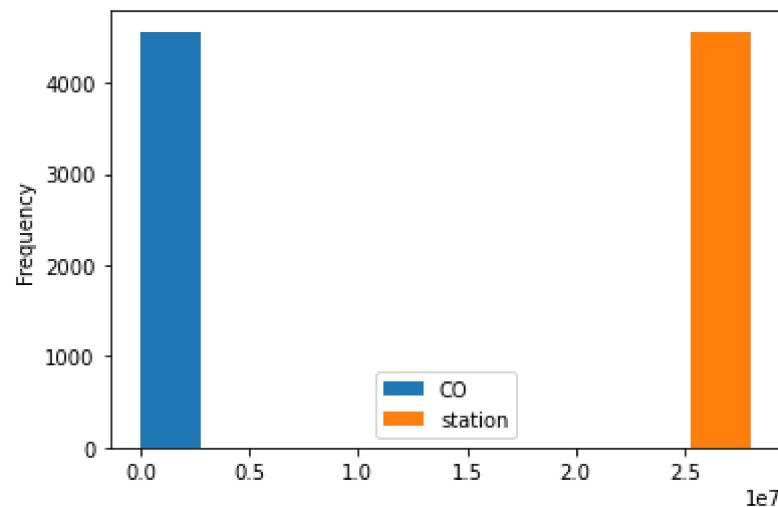
```
Out[10]: <AxesSubplot:>
```



Histogram

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

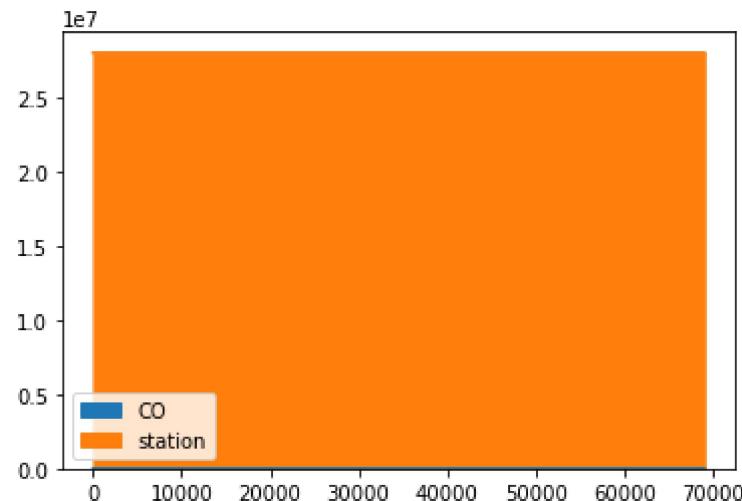
```
Out[11]: <AxesSubplot:ylabel='Frequency'>
```



Area chart

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

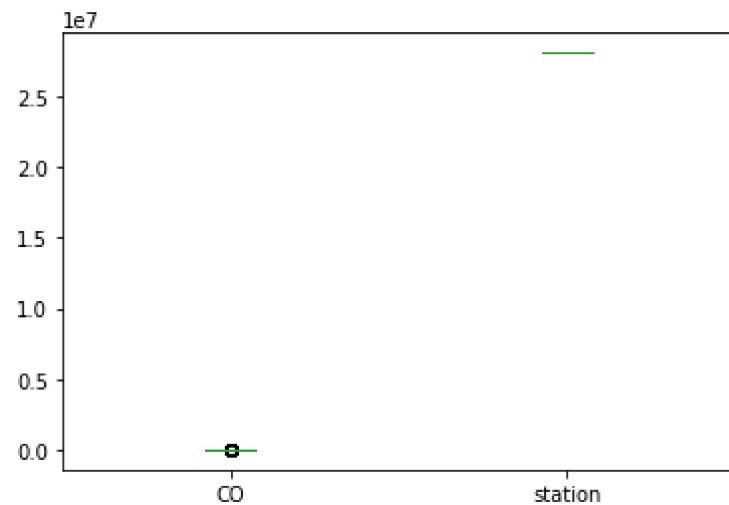
```
Out[12]: <AxesSubplot:>
```



Box chart

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

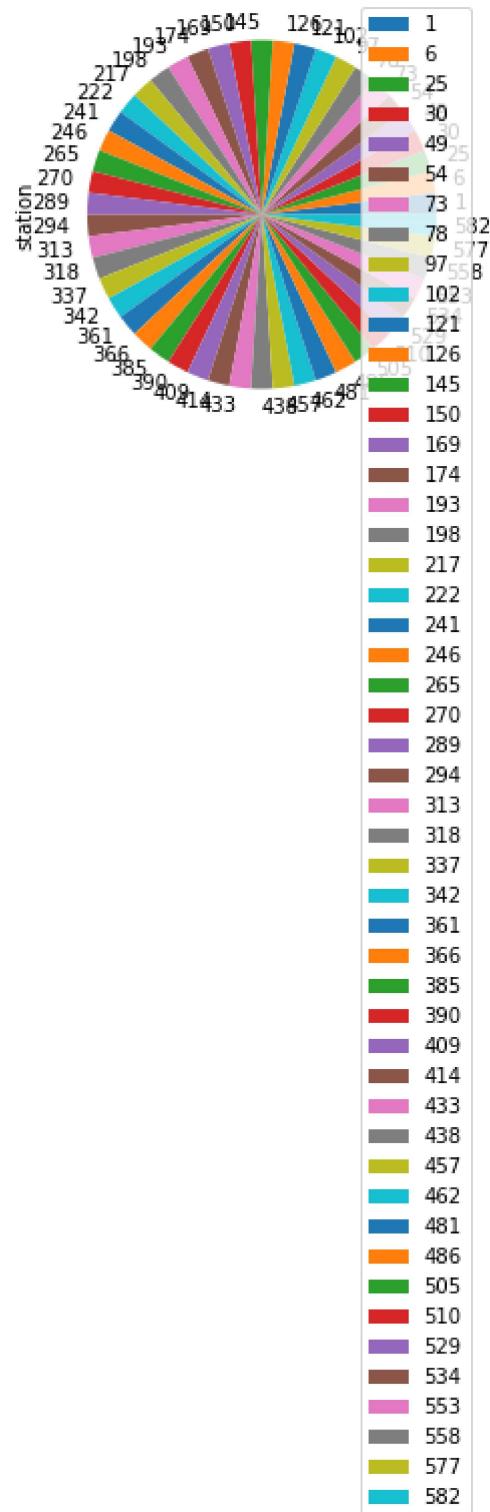
Out[13]: <AxesSubplot:>



Pie chart

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

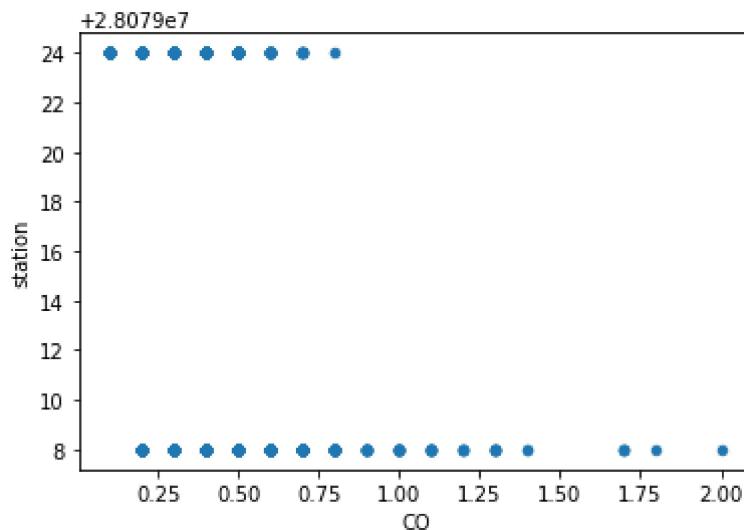
```
Out[14]: <AxesSubplot:ylabel='station'>
```



Scatter chart

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

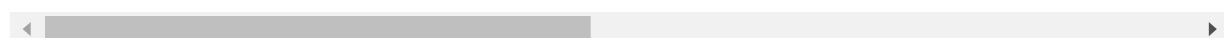
```
Out[15]: <AxesSubplot:xlabel='CO', ylabel='station'>
```



In [17]: df.describe()

Out[17]:

| | BEN | CH4 | CO | EBE | NMHC | NO | NO_2 |
|-------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| count | 4562.000000 | 4562.000000 | 4562.000000 | 4562.000000 | 4562.000000 | 4562.000000 | 4562.000000 |
| mean | 0.69349 | 1.329163 | 0.330579 | 0.286782 | 0.056773 | 21.742218 | 44.152126 |
| std | 0.46832 | 0.214399 | 0.161489 | 0.354442 | 0.037711 | 35.539531 | 30.234015 |
| min | 0.10000 | 0.020000 | 0.100000 | 0.100000 | 0.000000 | 1.000000 | 1.000000 |
| 25% | 0.40000 | 1.120000 | 0.200000 | 0.100000 | 0.030000 | 1.000000 | 20.000000 |
| 50% | 0.60000 | 1.390000 | 0.300000 | 0.200000 | 0.050000 | 9.000000 | 41.000000 |
| 75% | 0.90000 | 1.420000 | 0.400000 | 0.300000 | 0.070000 | 27.000000 | 64.000000 |
| max | 6.60000 | 3.920000 | 2.000000 | 7.400000 | 0.490000 | 431.000000 | 184.000000 |

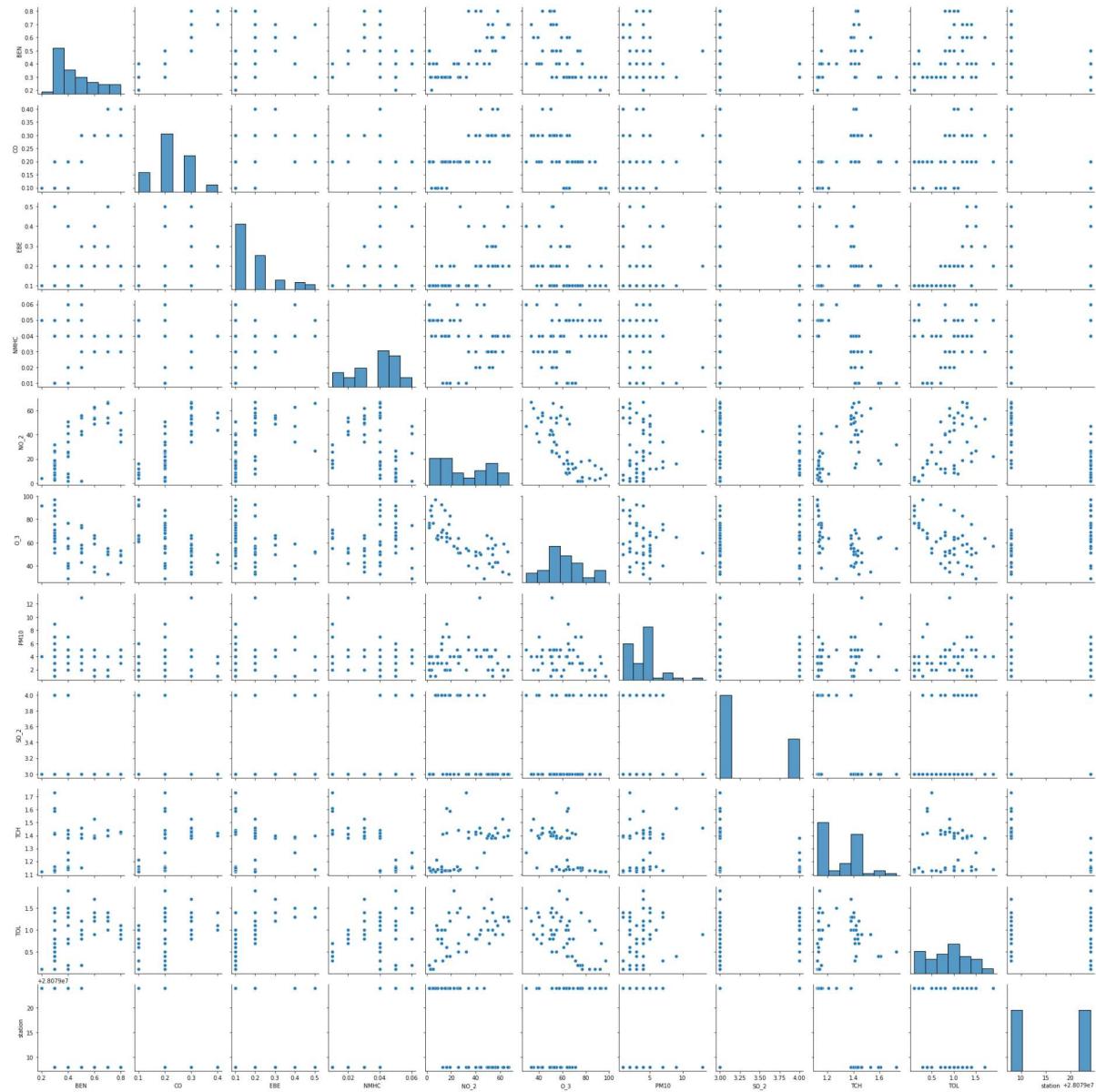


In [18]: df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
'PM10', 'SO_2', 'TCH', 'TOL', 'station']]

EDA AND VISUALIZATION

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

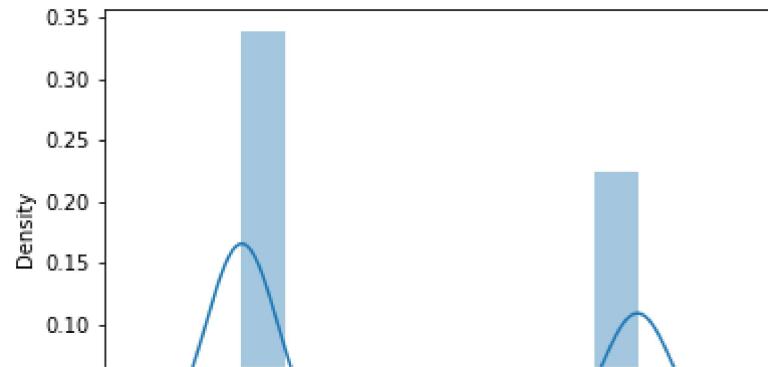
```
Out[19]: <seaborn.axisgrid.PairGrid at 0x244efe7f0a0>
```



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

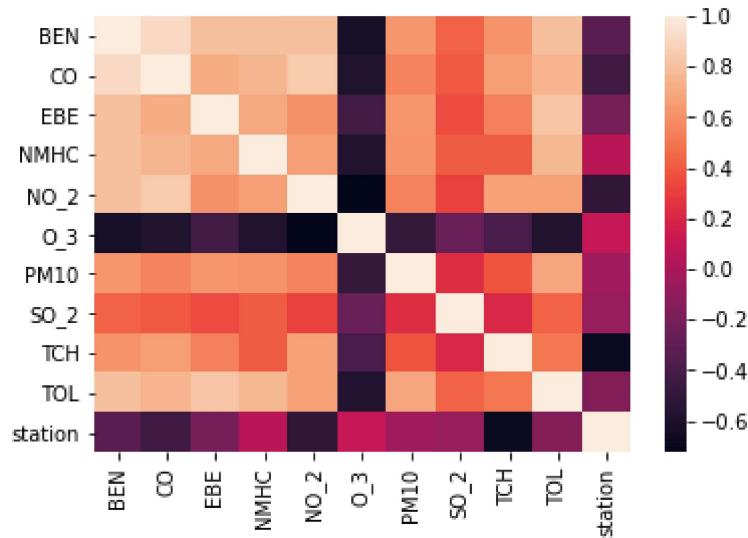
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function 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', 'NMHC', 'NO_2', 'O_3', 'PM10', '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]: 28079045.78354072
```

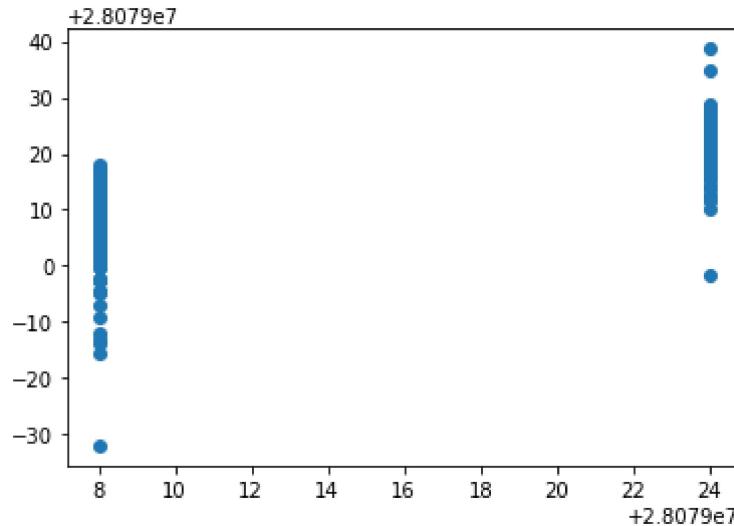
```
In [26]: coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])  
coeff
```

```
Out[26]:
```

| | Co-efficient |
|------|--------------|
| BEN | -0.717629 |
| CO | -19.047929 |
| EBE | 1.541832 |
| NMHC | 150.115720 |
| NO_2 | -0.147298 |
| O_3 | -0.087138 |
| PM10 | 0.098585 |
| SO_2 | -0.041842 |
| TCH | -17.404539 |
| TOL | -0.175334 |

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

```
Out[27]: <matplotlib.collections.PathCollection at 0x244f722ca00>
```



ACCURACY

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

```
Out[28]: 0.7851030761894053
```

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

```
Out[29]: 0.8116451643581055
```

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]: 0.6371597168844237
```

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

Out[33]: 0.6991419461780721

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]: 0.4031332358533595

Accuracy(Lasso)

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

Out[36]: 0.4216167158797801

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. , 0. , 0. , -0.26539256,
 -0.14616584, 0.22544676, 0.07568376, -0.09117967, 0.09846418])

In [39]: `en.intercept_`

Out[39]: 28079029.238879517

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

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

Out[41]: 0.4810089589514748

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)))`

4.82801817357268
31.989556574121774
5.655931097009738

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]: (4562, 10)
```

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

```
Out[46]: (4562,)
```

```
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]: 0.9888206926786497
```

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

```
Out[54]: 1.0
```

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

```
Out[55]: array([[1.0000000e+00, 1.42669593e-19]])
```

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_
```

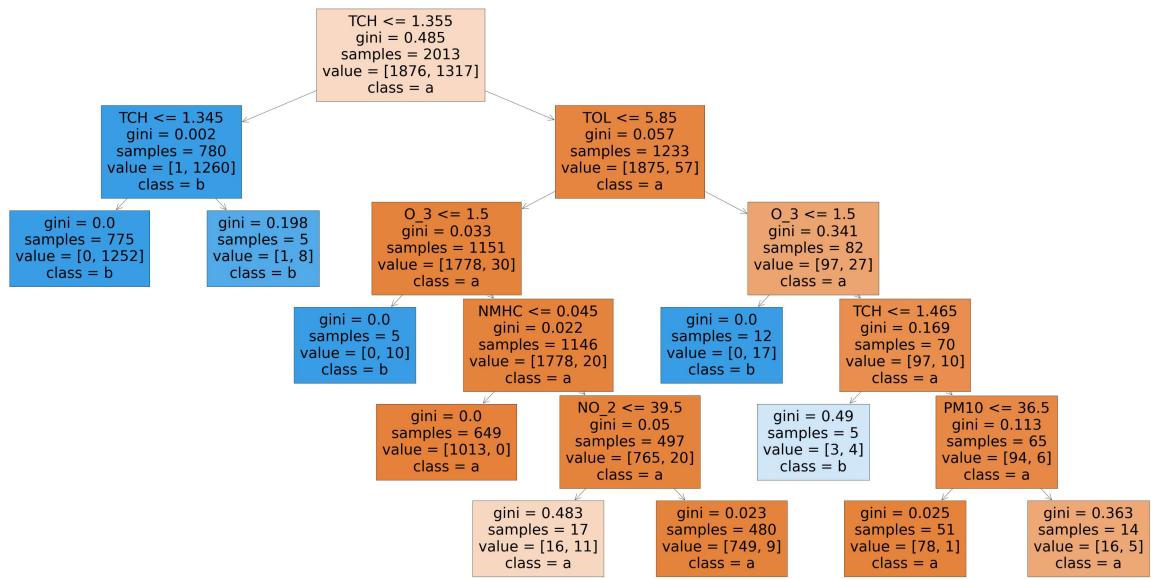
```
Out[60]: 0.9931099272916166
```

```
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(1545.230769230769, 1993.2, 'TCH <= 1.355\ngini = 0.485\nsamples = 2013
\nvalue = [1876, 1317]\nnclass = a'),
Text(686.7692307692307, 1630.8000000000002, 'TCH <= 1.345\ngini = 0.002\nsamples = 780\nvalue = [1, 1260]\nnclass = b'),
Text(343.38461538461536, 1268.4, 'gini = 0.0\nsamples = 775\nvalue = [0, 125
2]\nnclass = b'),
Text(1030.1538461538462, 1268.4, 'gini = 0.198\nsamples = 5\nvalue = [1, 8]
\nnclass = b'),
Text(2403.6923076923076, 1630.8000000000002, 'TOL <= 5.85\ngini = 0.057\nsamples = 1233\nvalue = [1875, 57]\nnclass = a'),
Text(1716.9230769230767, 1268.4, 'O_3 <= 1.5\ngini = 0.033\nsamples = 1151\nvalue = [1778, 30]\nnclass = a'),
Text(1373.5384615384614, 906.0, 'gini = 0.0\nsamples = 5\nvalue = [0, 10]\nnclass = b'),
Text(2060.3076923076924, 906.0, 'NMHC <= 0.045\ngini = 0.022\nsamples = 1146
\nvalue = [1778, 20]\nnclass = a'),
Text(1716.9230769230767, 543.5999999999999, 'gini = 0.0\nsamples = 649\nvalue = [1013, 0]\nnclass = a'),
Text(2403.6923076923076, 543.5999999999999, 'NO_2 <= 39.5\ngini = 0.05\nsamples = 497\nvalue = [765, 20]\nnclass = a'),
Text(2060.3076923076924, 181.1999999999982, 'gini = 0.483\nsamples = 17\nvalue = [16, 11]\nnclass = a'),
Text(2747.076923076923, 181.1999999999982, 'gini = 0.023\nsamples = 480\nvalue = [749, 9]\nnclass = a'),
Text(3090.461538461538, 1268.4, 'O_3 <= 1.5\ngini = 0.341\nsamples = 82\nvalue = [97, 27]\nnclass = a'),
Text(2747.076923076923, 906.0, 'gini = 0.0\nsamples = 12\nvalue = [0, 17]\nnclass = b'),
Text(3433.8461538461534, 906.0, 'TCH <= 1.465\ngini = 0.169\nsamples = 70\nvalue = [97, 10]\nnclass = a'),
Text(3090.461538461538, 543.5999999999999, 'gini = 0.49\nsamples = 5\nvalue = [3, 4]\nnclass = b'),
Text(3777.230769230769, 543.5999999999999, 'PM10 <= 36.5\ngini = 0.113\nsamples = 65\nvalue = [94, 6]\nnclass = a'),
Text(3433.8461538461534, 181.19999999999982, 'gini = 0.025\nsamples = 51\nvalue = [78, 1]\nnclass = a'),
Text(4120.615384615385, 181.19999999999982, 'gini = 0.363\nsamples = 14\nvalue = [16, 5]\nnclass = a)]]
```



Conclusion

Accuracy

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

```
Out[63]: 0.8116451643581055
```

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

```
Out[64]: 0.6991419461780721
```

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

```
Out[65]: 0.4031332358533595
```

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

```
Out[66]: 0.4810089589514748
```

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

```
Out[67]: 0.9888206926786497
```

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

```
Out[68]: 0.9931099272916166
```

Random Forest is suitable for this dataset

In []: