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(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs(Dataset)\madrid_201
df
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	NaN	NaN	2
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	1.41	0.8	2
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	NaN	1.1	1
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	NaN	NaN	2
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	NaN	NaN	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	NaN	NaN	2
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	NaN	NaN	2
69093	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	28.0	48.0	91.0	2.0	NaN	NaN	NaN	NaN	NaN	2
69094	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	141.0	103.0	320.0	2.0	NaN	NaN	NaN	NaN	NaN	2
69095	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	69.0	96.0	202.0	3.0	26.0	NaN	NaN	NaN	NaN	2

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[['BEN', 'TOL', 'TCH']]
data
```

Out[6]:

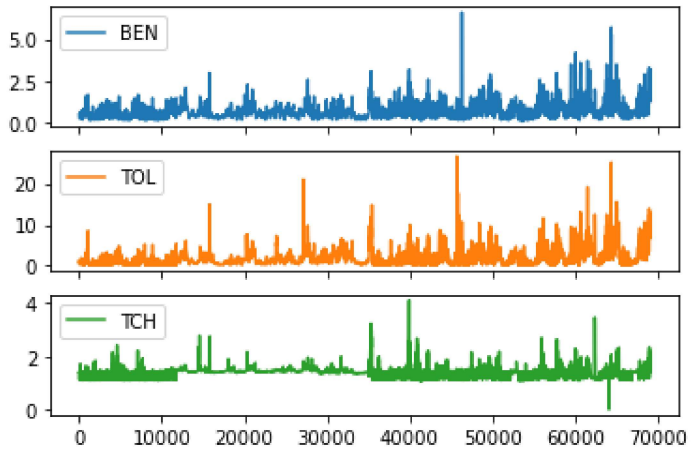
	BEN	TOL	TCH
1	0.5	0.8	1.41
6	0.4	1.4	1.16
25	0.4	0.7	1.44
30	0.3	0.8	1.14
49	0.3	0.4	1.42
...	...	...	...
69030	1.8	11.9	1.40
69049	3.1	12.5	2.22
69054	1.6	10.3	1.32
69073	3.2	13.0	1.72
69078	1.3	6.8	1.24

4562 rows × 3 columns

Line chart

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

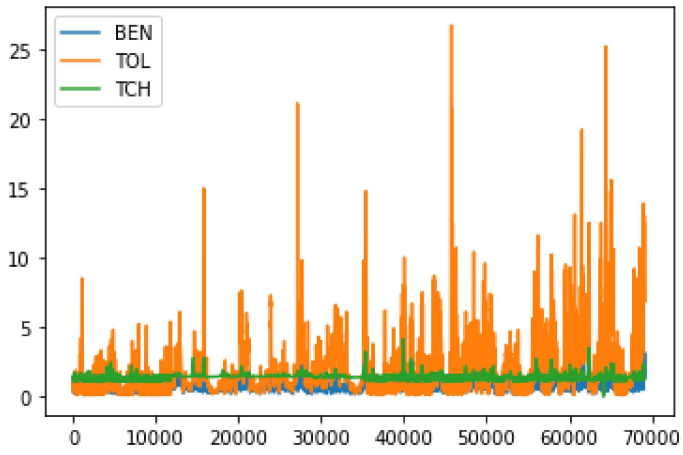
Out[7]:
```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Line chart

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

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

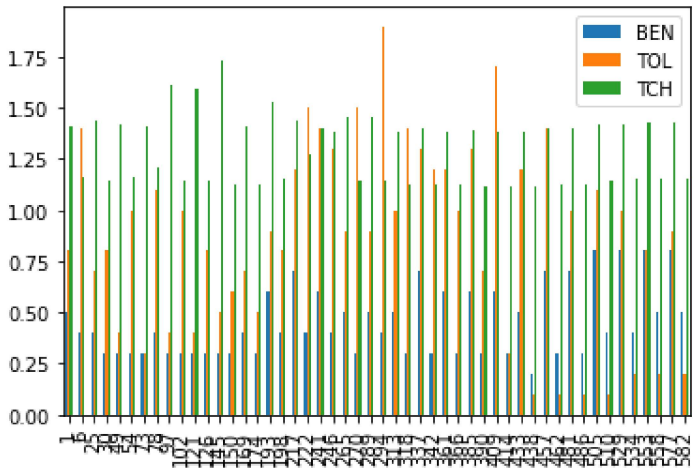


Bar chart

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

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

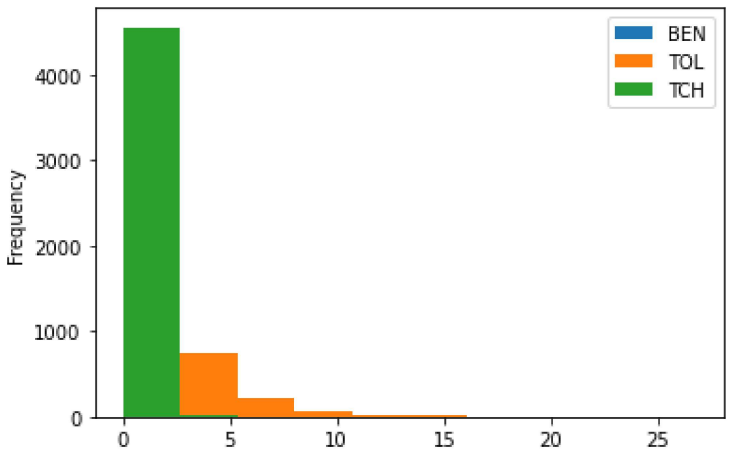
Out[10]: <AxesSubplot:>



Histogram

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

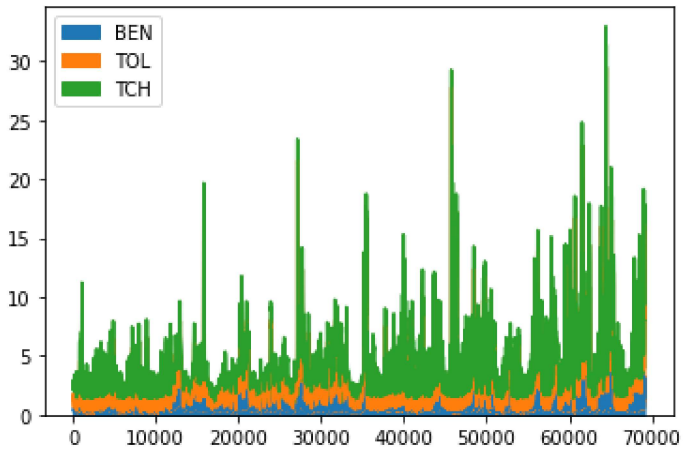
Out[11]: <AxesSubplot:ylabel='Frequency'>



Area chart

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

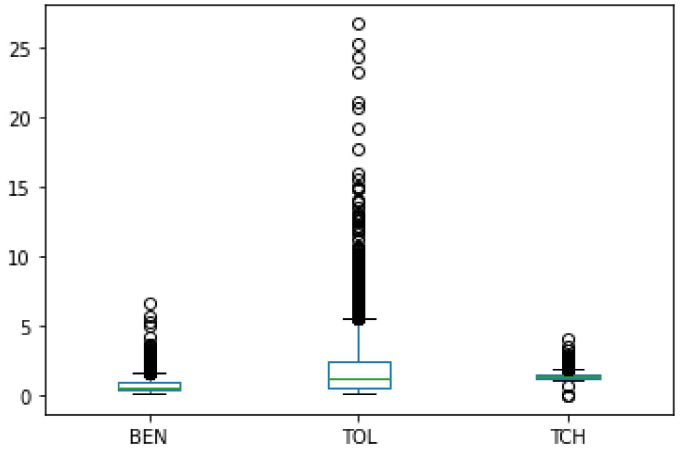
Out[12]: <AxesSubplot:>



Box chart

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

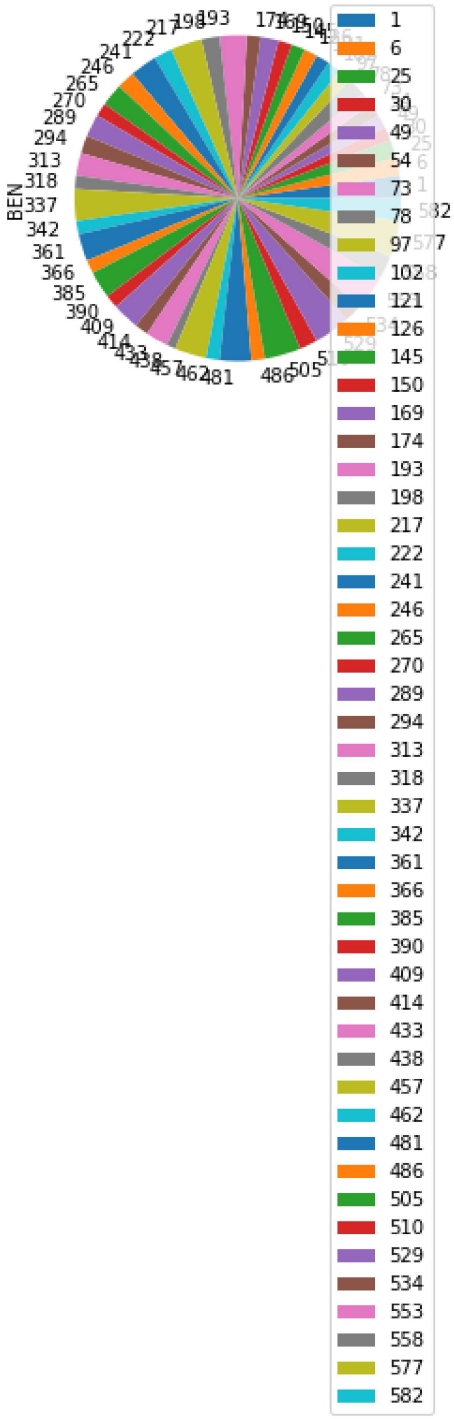
Out[13]: <AxesSubplot:>



Pie chart

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

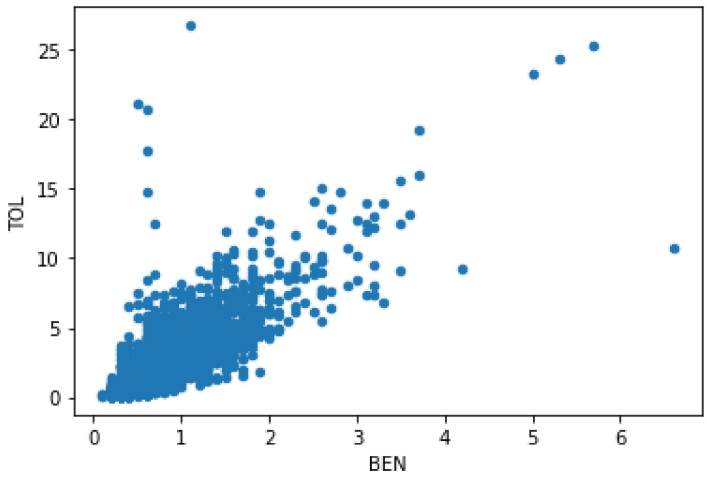
Out[14]: <AxesSubplot:ylabel='BEN'>



Scatter chart

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

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



In [16]:
```
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 [17]:
```
df.describe()
```

Out[17]:

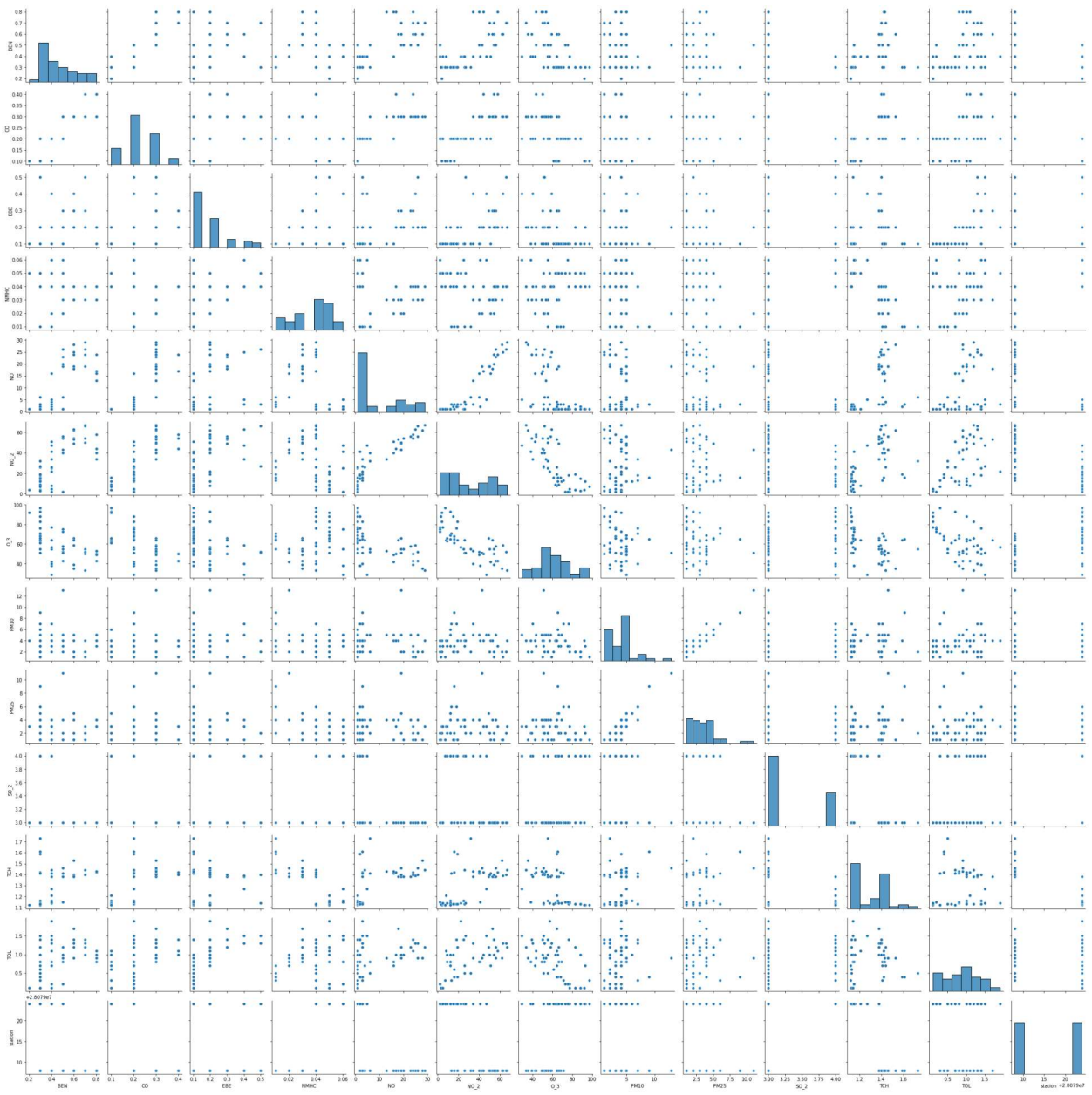
	BEN	CH4	CO	EBE	NMHC	NO	NO_2	N
count	4562.00000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.0000
mean	0.69349	1.329163	0.330579	0.286782	0.056773	21.742218	44.152126	77.4947
std	0.46832	0.214399	0.161489	0.354442	0.037711	35.539531	30.234015	79.2185
min	0.10000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000	2.0000
25%	0.40000	1.120000	0.200000	0.100000	0.030000	1.000000	20.000000	24.0000
50%	0.60000	1.390000	0.300000	0.200000	0.050000	9.000000	41.000000	56.0000
75%	0.90000	1.420000	0.400000	0.300000	0.070000	27.000000	64.000000	106.0000
max	6.60000	3.920000	2.000000	7.400000	0.490000	431.000000	184.000000	844.0000

In [18]:
```
df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
```

EDA AND VISUALIZATION

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

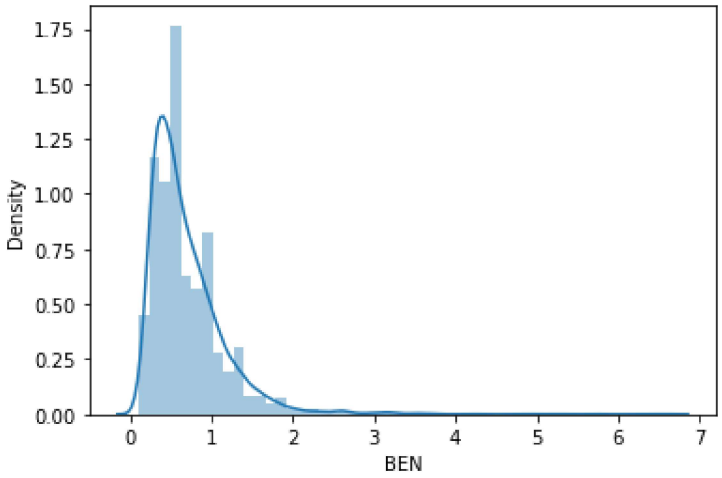
Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x27c84cd3910>
```



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

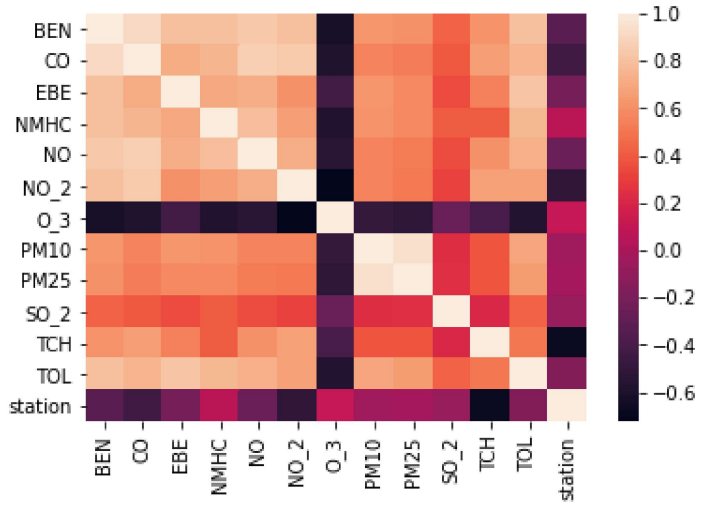
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Please
adapt your code to use either `displot` (a figure-level function with similar flexibil
ity) or `histplot` (an axes-level function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

Out[20]: <AxesSubplot:xlabel='BEN', ylabel='Density'>



In [21]:
```
sns.heatmap(df1.corr())
```

Out[21]: <AxesSubplot:>



TO TRAIN THE MODEL AND MODEL BULDING

In [22]:
```
x=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       '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]:
```
28079046.570636477
```

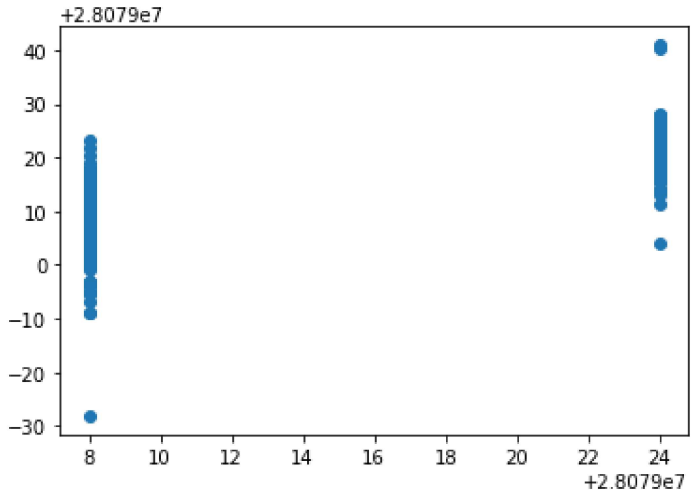
In [26]:
```
coeff=pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[26]:

	Co-efficient
BEN	-0.988819
CO	-21.859751
EBE	0.864948
NMHC	143.943341
NO	0.034024
NO_2	-0.146479
O_3	-0.082675
PM10	0.038415
PM25	0.107486
SO_2	-0.023119
TCH	-17.542887
TOL	-0.188026

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

Out[27]:
```
<matplotlib.collections.PathCollection at 0x27c8faf3f40>
```



ACCURACY

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

Out[28]:
```
0.793861847017371
```

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

Out[29]:
```
0.8165979998859799
```

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]:
```
0.6888725016180086
```

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

Out[33]:
```
0.7163387032712538
```

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

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

In [35]:
```
la.score(x_test,y_test)
```

Out[35]:
```
0.4072231243638361
```

Accuracy(Lasso)

In [36]:
```
la.score(x_train,y_train)
```

Out[36]:
```
0.4234731805172326
```

Accuracy(Elastic Net)

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.        ,  0.03511094,
       -0.29197906, -0.14910184,  0.24485607, -0.05401081,  0.07035688,
       -0.09640885,  0.        ])
```

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

Out[39]:
```
28079030.29771935
```

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

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

Out[41]:
```
0.46420112917354417
```

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)))
```

```
4.7996836569761605
32.42922822200433
5.694666647136102
```

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([[1.00000000e+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_
```

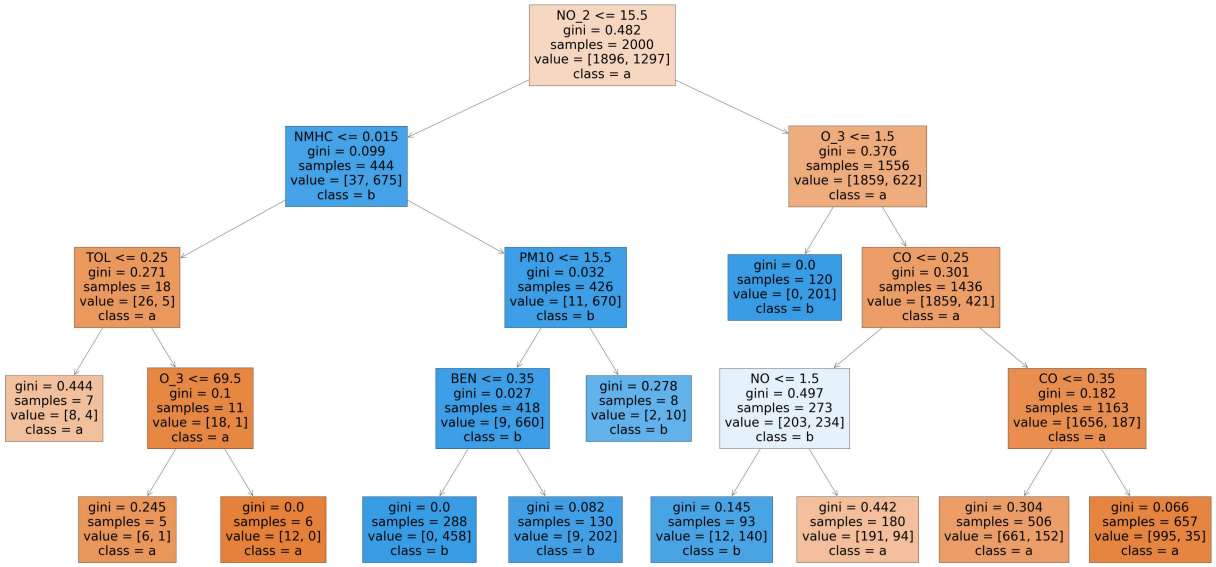
Out[60]:
```
0.9921706661770269
```

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','c','d']
```

Out[62]:
```
[Text(2232.0, 1956.96, 'NO_2 <= 15.5\ngini = 0.482\nsamples = 2000\nvalue = [1896, 129
7]\nclass = a'),
 Text(1312.941176470588, 1522.0800000000002, 'NMHC <= 0.015\ngini = 0.099\nsamples = 4
44\nvalue = [37, 675]\nclass = b'),
 Text(525.1764705882352, 1087.2, 'TOL <= 0.25\ngini = 0.271\nsamples = 18\nvalue = [2
6, 5]\nclass = a'),
 Text(262.5882352941176, 652.3200000000002, 'gini = 0.444\nsamples = 7\nvalue = [8, 4]
\nclass = a'),
 Text(787.7647058823529, 652.3200000000002, 'O_3 <= 69.5\ngini = 0.1\nsamples = 11\nva
lue = [18, 1]\nclass = a'),
 Text(525.1764705882352, 217.44000000000005, 'gini = 0.245\nsamples = 5\nvalue = [6,
1]\nclass = a'),
 Text(1050.3529411764705, 217.44000000000005, 'gini = 0.0\nsamples = 6\nvalue = [12,
0]\nclass = a'),
 Text(2100.705882352941, 1087.2, 'PM10 <= 15.5\ngini = 0.032\nsamples = 426\nvalue =
[11, 670]\nclass = b'),
 Text(1838.1176470588234, 652.3200000000002, 'BEN <= 0.35\ngini = 0.027\nsamples = 418
\nvalue = [9, 660]\nclass = b'),
 Text(1575.5294117647059, 217.44000000000005, 'gini = 0.0\nsamples = 288\nvalue = [0,
458]\nclass = b'),
 Text(2100.705882352941, 217.44000000000005, 'gini = 0.082\nsamples = 130\nvalue = [9,
202]\nclass = b'),
 Text(2363.2941176470586, 652.3200000000002, 'gini = 0.278\nsamples = 8\nvalue = [2, 1
0]\nclass = b'),
 Text(3151.0588235294117, 1522.0800000000002, 'O_3 <= 1.5\ngini = 0.376\nsamples = 155
6\nvalue = [1859, 622]\nclass = a'),
 Text(2888.4705882352937, 1087.2, 'gini = 0.0\nsamples = 120\nvalue = [0, 201]\nclass
= b'),
 Text(3413.6470588235293, 1087.2, 'CO <= 0.25\ngini = 0.301\nsamples = 1436\nvalue =
[1859, 421]\nclass = a'),
 Text(2888.4705882352937, 652.3200000000002, 'NO <= 1.5\ngini = 0.497\nsamples = 273\n
value = [203, 234]\nclass = b'),
 Text(2625.882352941176, 217.44000000000005, 'gini = 0.145\nsamples = 93\nvalue = [12,
140]\nclass = b'),
 Text(3151.0588235294117, 217.44000000000005, 'gini = 0.442\nsamples = 180\nvalue = [1
91, 94]\nclass = a'),
 Text(3938.8235294117644, 652.3200000000002, 'CO <= 0.35\ngini = 0.182\nsamples = 1163
\nvalue = [1656, 187]\nclass = a'),
 Text(3676.235294117647, 217.44000000000005, 'gini = 0.304\nsamples = 506\nvalue = [66
1, 152]\nclass = a'),
 Text(4201.411764705882, 217.44000000000005, 'gini = 0.066\nsamples = 657\nvalue = [99
5, 35]\nclass = a')]
```



Conclusion

Accuracy

Linear Regression:0.8165979998859799

Ridge Regression:0.7163387032712538

Lasso Regression:0.4234731805172326

ElasticNet Regression:0.46420112917354417

Logistic Regression:0.9888206926786497

Random Forest:0.9921706661770269

Random Forest is suitable for this dataset