In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv(r"C:\Users\user\Downloads\C10_air\csvs_per_year\csvs_per_year\
df
```

Out[2]:

| | date | BEN | CO | EBE | NMHC | NO | NO_2 | O_3 | PM10 | PM25 | SO_2 | TCH | TOL | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2016-11-01 01:00:00 | NaN | 0.7 | NaN | NaN | 153.0 | 77.0 | NaN | NaN | NaN | 7.0 | NaN | NaN | 2 |
| **1** | 2016-11-01 01:00:00 | 3.1 | 1.1 | 2.0 | 0.53 | 260.0 | 144.0 | 4.0 | 46.0 | 24.0 | 18.0 | 2.44 | 14.4 | 2 |
| **2** | 2016-11-01 01:00:00 | 5.9 | NaN | 7.5 | NaN | 297.0 | 139.0 | NaN | NaN | NaN | NaN | NaN | 26.0 | 2 |
| **3** | 2016-11-01 01:00:00 | NaN | 1.0 | NaN | NaN | 154.0 | 113.0 | 2.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **4** | 2016-11-01 01:00:00 | NaN | NaN | NaN | NaN | 275.0 | 127.0 | 2.0 | NaN | NaN | 18.0 | NaN | NaN | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **209491** | 2016-07-01 00:00:00 | NaN | 0.2 | NaN | NaN | 2.0 | 29.0 | 73.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **209492** | 2016-07-01 00:00:00 | NaN | 0.3 | NaN | NaN | 1.0 | 29.0 | NaN | 36.0 | NaN | 5.0 | NaN | NaN | 2 |
| **209493** | 2016-07-01 00:00:00 | NaN | NaN | NaN | NaN | 1.0 | 19.0 | 71.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **209494** | 2016-07-01 00:00:00 | NaN | NaN | NaN | NaN | 6.0 | 17.0 | 85.0 | NaN | NaN | NaN | NaN | NaN | 2 |
| **209495** | 2016-07-01 00:00:00 | NaN | NaN | NaN | NaN | 2.0 | 46.0 | 61.0 | 34.0 | NaN | NaN | NaN | NaN | 2 |

209496 rows × 14 columns

In [3]:
```
df1 = df.fillna(0)
df1
```

Out[3]:

| | date | BEN | CO | EBE | NMHC | NO | NO_2 | O_3 | PM10 | PM25 | SO_2 | TCH | TOL | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2016-11-01 01:00:00 | 0.0 | 0.7 | 0.0 | 0.00 | 153.0 | 77.0 | 0.0 | 0.0 | 0.0 | 7.0 | 0.00 | 0.0 | 28 |
| **1** | 2016-11-01 01:00:00 | 3.1 | 1.1 | 2.0 | 0.53 | 260.0 | 144.0 | 4.0 | 46.0 | 24.0 | 18.0 | 2.44 | 14.4 | 28 |
| **2** | 2016-11-01 01:00:00 | 5.9 | 0.0 | 7.5 | 0.00 | 297.0 | 139.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.00 | 26.0 | 28 |
| **3** | 2016-11-01 01:00:00 | 0.0 | 1.0 | 0.0 | 0.00 | 154.0 | 113.0 | 2.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 28 |
| **4** | 2016-11-01 01:00:00 | 0.0 | 0.0 | 0.0 | 0.00 | 275.0 | 127.0 | 2.0 | 0.0 | 0.0 | 18.0 | 0.00 | 0.0 | 28 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **209491** | 2016-07-01 00:00:00 | 0.0 | 0.2 | 0.0 | 0.00 | 2.0 | 29.0 | 73.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 28 |
| **209492** | 2016-07-01 00:00:00 | 0.0 | 0.3 | 0.0 | 0.00 | 1.0 | 29.0 | 0.0 | 36.0 | 0.0 | 5.0 | 0.00 | 0.0 | 28 |
| **209493** | 2016-07-01 00:00:00 | 0.0 | 0.0 | 0.0 | 0.00 | 1.0 | 19.0 | 71.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 28 |
| **209494** | 2016-07-01 00:00:00 | 0.0 | 0.0 | 0.0 | 0.00 | 6.0 | 17.0 | 85.0 | 0.0 | 0.0 | 0.0 | 0.00 | 0.0 | 28 |
| **209495** | 2016-07-01 00:00:00 | 0.0 | 0.0 | 0.0 | 0.00 | 2.0 | 46.0 | 61.0 | 34.0 | 0.0 | 0.0 | 0.00 | 0.0 | 28 |

209496 rows × 14 columns

In [4]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209496 entries, 0 to 209495
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209496 non-null  object 
 1   BEN      209496 non-null  float64
 2   CO       209496 non-null  float64
 3   EBE      209496 non-null  float64
 4   NMHC     209496 non-null  float64
 5   NO       209496 non-null  float64
 6   NO_2     209496 non-null  float64
 7   O_3      209496 non-null  float64
 8   PM10     209496 non-null  float64
 9   PM25     209496 non-null  float64
 10  SO_2     209496 non-null  float64
 11  TCH      209496 non-null  float64
 12  TOL      209496 non-null  float64
 13  station  209496 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

In [5]:
```
df1.columns
```

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [6]:
```
df2 = df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
```

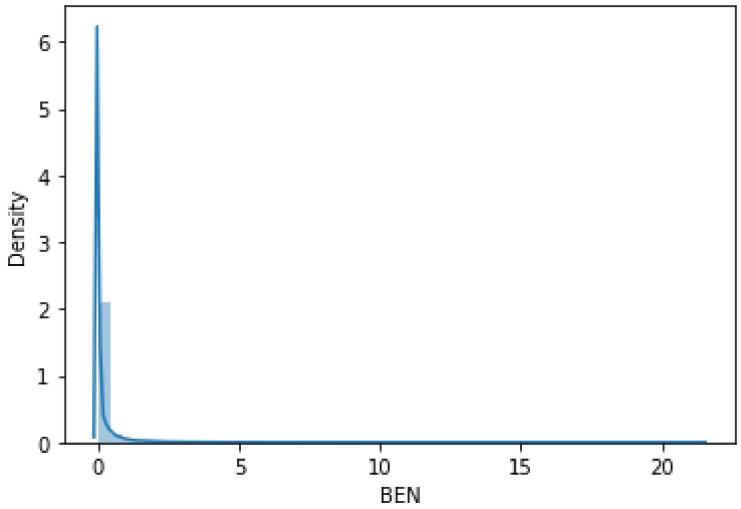
In [7]:
```
sns.pairplot(df2)
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x1ea327b8070>
```

In [8]:
```
sns.distplot(df2['BEN'])
```

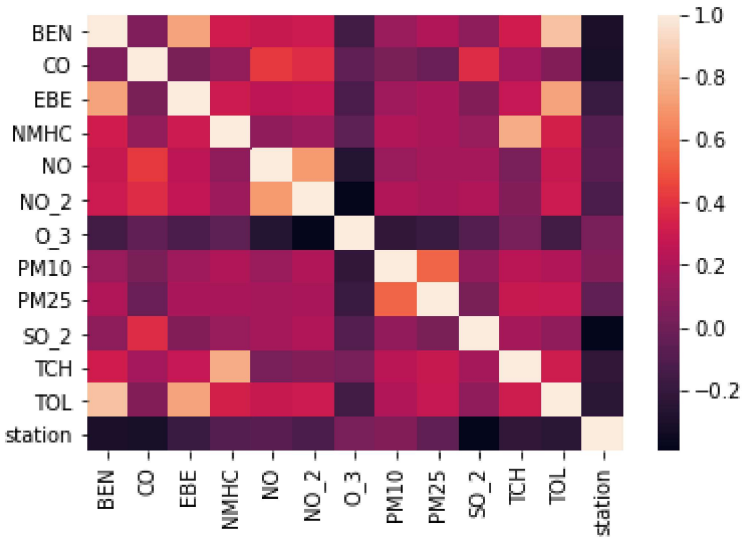
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function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

Out[8]: <AxesSubplot:xlabel='BEN', ylabel='Density'>




In [9]:
```
sns.heatmap(df2.corr())
```

Out[9]: <AxesSubplot:>




# Linear Regression

In [10]:
```
x = df2[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'O_3',
       'PM10', 'SO_2', 'TCH']]
y = df2['TOL']
```

In [11]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.30)
```

In [12]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(x_train,y_train)
```

Out[12]:
```
LinearRegression()
```

In [13]:
```
print(lr.intercept_)
```

```
-0.06639543167792183
```

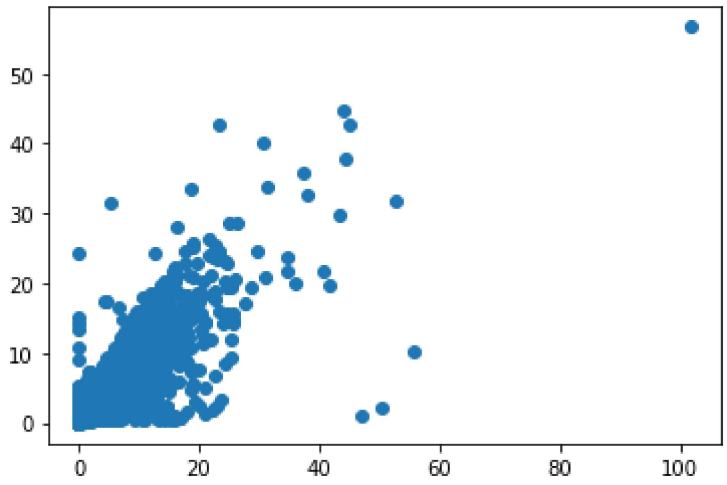
In [14]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[14]:

| | Co-efficient |
|---|---|
| **BEN** | 2.598907 |
| **CO** | 0.019456 |
| **EBE** | 1.400485 |
| **NMHC** | 1.016204 |
| **NO_2** | 0.001343 |
| **O_3** | -0.000184 |
| **PM10** | 0.008444 |
| **SO_2** | 0.007826 |
| **TCH** | -0.027463 |

In [15]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x1ea4172ce20>
```




In [16]:
```
print(lr.score(x_test,y_test))
```

```
0.777055654852674
```

In [17]:
```
lr.score(x_train,y_train)
```

Out[17]:
```
0.7555351183595433
```

# Ridge and Lasso

In [18]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [19]:
```
rr = Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_train,y_train)
```

Out[19]:
```
0.7555343808873061
```

In [20]:
```
rr.score(x_test,y_test)
```

Out[20]:
```
0.7770510973146894
```

# Lasso Regression

In [21]:
```
ls = Lasso(alpha=10)
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
```

Out[21]:
```
0.0630024658791849
```

In [22]:
```
ls.score(x_test,y_test)
```

Out[22]:
```
0.06595800371094274
```

# ElacticNET regression

In [23]:
```
from sklearn.linear_model import ElasticNet
es = ElasticNet()
es.fit(x_train,y_train)
```

Out[23]:
```
ElasticNet()
```

In [24]:
```
print(es.coef_)
```

```
[ 0.38428293 -0.          0.          0.          0.01652039 -0.00084011
  0.01565563  0.          0.        ]
```

In [25]:
```
print(es.intercept_)
```

```
-0.19412700172545483
```

In [26]:
```
print(es.score(x_test,y_test))
```

```
0.2535554399419412
```

In [27]:
```
print(es.score(x_train,y_train))
```

```
0.24439977120943712
```

# LogisticRegression

In [28]:
```
from sklearn.linear_model import LogisticRegression
```

In [29]:
```
feature_matrix = df2.iloc[:,0:15]
target_vector = df2.iloc[:,-1]
```

In [30]:
```
feature_matrix.shape
```

Out[30]:
```
(209496, 13)
```

In [31]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
fs = StandardScaler().fit_transform(feature_matrix)
```

In [33]:
```
logs = LogisticRegression()
logs.fit(fs,target_vector)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:
763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[33]:
```
LogisticRegression()
```

In [34]:
```
observation = [[1.4,1.5,1.6,2.7,2.3,3.3,2.3,4.1,2.3,4.2,1.2,12,2]]
prediction = logs.predict(observation)
```

In [35]:
```
print(prediction)
```

```
[28079059]
```

In [36]:
```
logs.classes_
```

Out[36]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [37]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(feature_matrix,target_vector,t
```

In [38]:
```
print(logs.score(x_test,y_test))
```

```
0.04144855128959888
```

In [39]:
```
print(logs.score(x_train,y_train))
```

```
0.042135195401201524
```

# Conclusion

linear regression is bestfit model

linear regression is best fit model for dataset madrid_2001. The score of x_train,y_train is 0.770048129954465 and x_test and y_test score is 0.7778147341001751.

In [ ]: