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	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

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	2008-06-01 01:00:00	0.00	0.47	0.00	0.00	0.00	83.089996	120.699997	0.00	16.990000	16.889
1	2008-06-01 01:00:00	0.00	0.59	0.00	0.00	0.00	94.820000	130.399994	0.00	17.469999	19.040
2	2008-06-01 01:00:00	0.00	0.55	0.00	0.00	0.00	75.919998	104.599998	0.00	13.470000	20.270
3	2008-06-01 01:00:00	0.00	0.36	0.00	0.00	0.00	61.029999	66.559998	0.00	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	0.00	0.30	0.00	0.00	0.00	41.880001	48.500000	0.00	35.830002	15.020
226389	2008-11-01 00:00:00	0.25	0.00	0.56	0.00	0.11	83.610001	102.199997	0.00	14.130000	17.540
226390	2008-11-01 00:00:00	0.54	0.00	2.70	0.00	0.18	70.639999	81.860001	0.00	0.000000	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

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 226392 entries, 0 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     226392 non-null  object 
 1   BEN      226392 non-null  float64
 2   CO       226392 non-null  float64
 3   EBE      226392 non-null  float64
 4   MXY      226392 non-null  float64
 5   NMHC     226392 non-null  float64
 6   NO_2     226392 non-null  float64
 7   NOx      226392 non-null  float64
 8   OXY      226392 non-null  float64
 9   O_3      226392 non-null  float64
 10  PM10     226392 non-null  float64
 11  PM25     226392 non-null  float64
 12  PXY      226392 non-null  float64
 13  SO_2     226392 non-null  float64
 14  TCH      226392 non-null  float64
 15  TOL      226392 non-null  float64
 16  station  226392 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.4+ MB
```

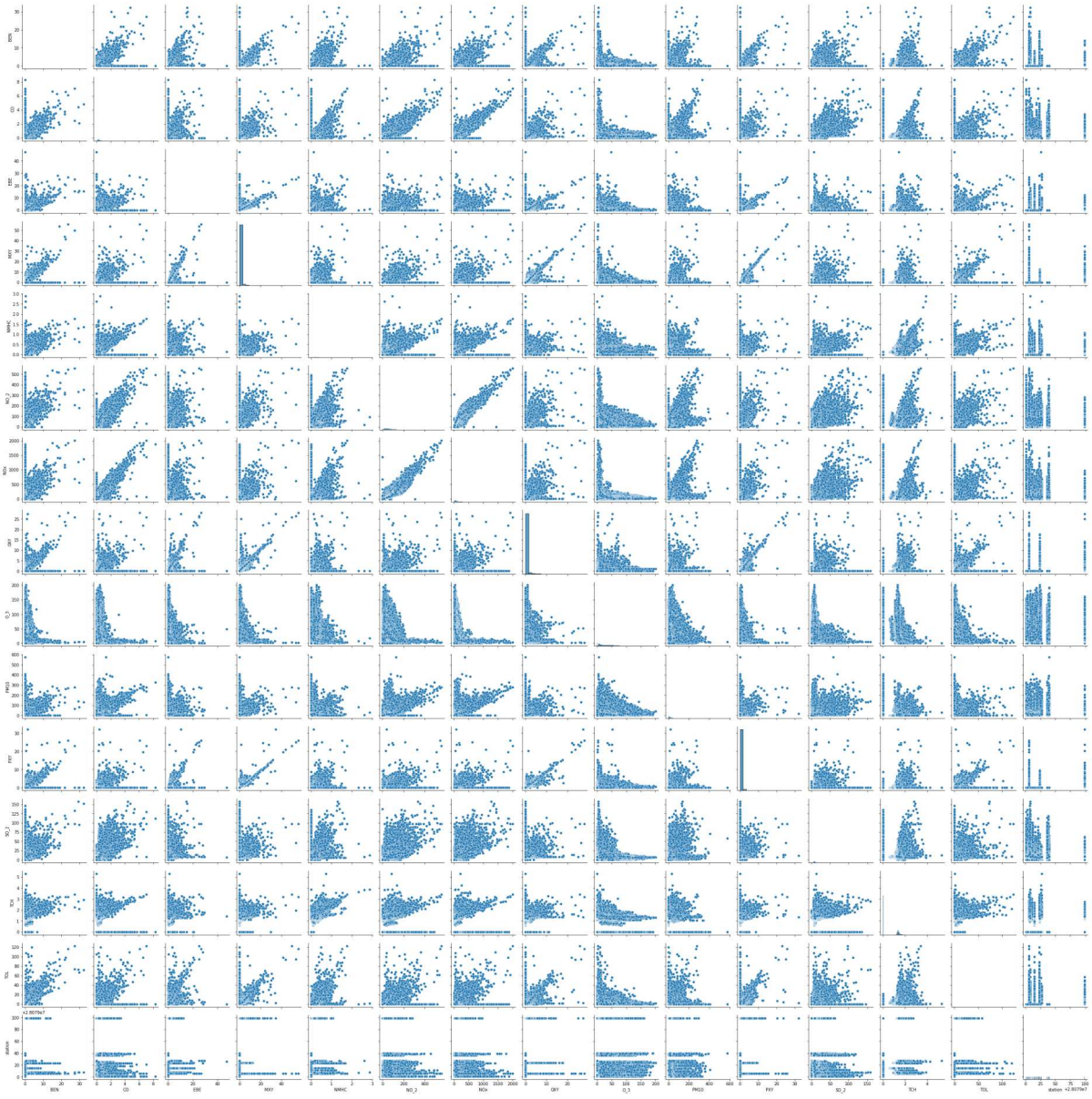
In [5]:
```
df1.columns
```

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [6]:
```
df2 = df1[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

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

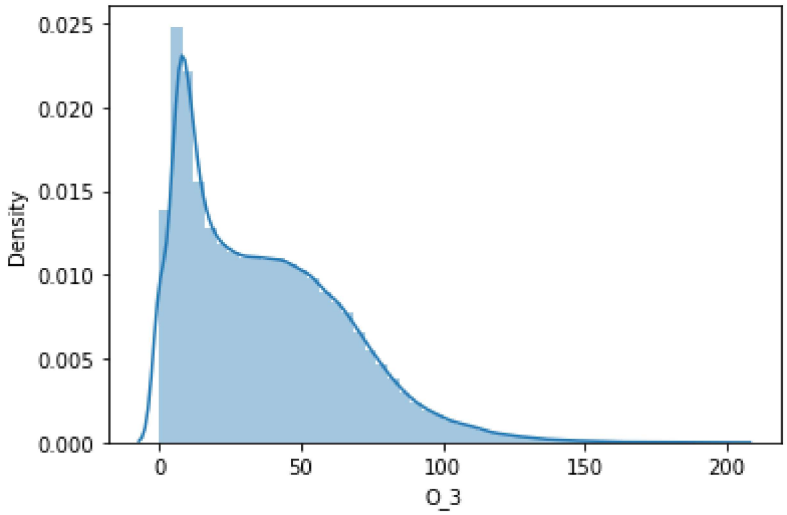
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x130a32253d0>
```



In [8]:
```
sns.distplot(df2['O_3'])
```

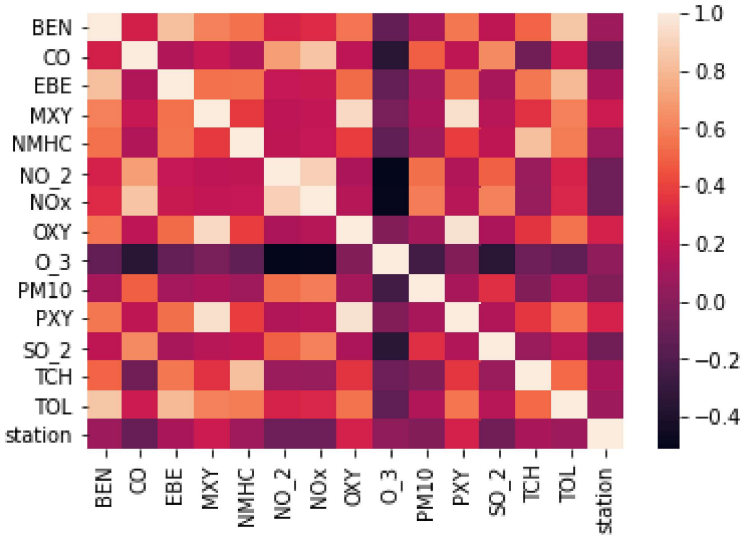
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function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

Out[8]: <AxesSubplot:xlabel='O_3', ylabel='Density'>



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

Out[9]: <AxesSubplot:>



Linear Regression

In [10]:
```
x = df2[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY','O_3',
       'PM10', 'SO_2','PXY', '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_)
```

```
-0.15790525774946507
```

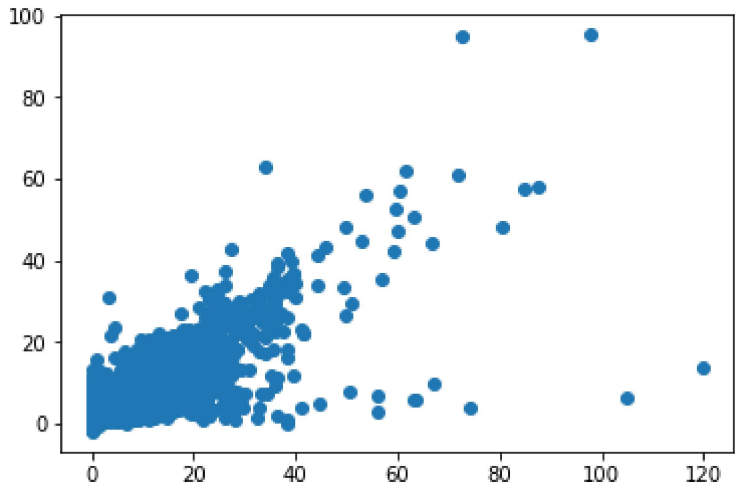
In [14]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[14]:

	Co-efficient
BEN	2.290059
CO	0.400020
EBE	1.035683
MXY	0.823381
NMHC	4.476706
NO_2	0.004692
NOx	-0.000773
OXY	-0.095826
O_3	-0.000748
PM10	0.001484
SO_2	-0.020287
PXY	-0.910980
TCH	-0.109260

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

Out[15]: <matplotlib.collections.PathCollection at 0x130bf6da0a0>



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

```
0.7718979516505728
```

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

Out[17]: 0.7772643081000624

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]: 0.7772627691767515

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

Out[20]: 0.7718947646020118

Lasso Regression

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

Out[21]: 0.0939728549590042

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

Out[22]:
```
0.09418802859837905
```

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_)
```

```
[ 0.77622095 -0.          0.9126162   0.46491494  0.          0.00434494
  0.00559674  0.          0.         -0.0035862  -0.          0.
  0.11956222]
```

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

```
-0.11045473847242904
```

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

```
0.6332365209822374
```

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

```
0.6391083273349589
```

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]:
```
(226392, 15)
```

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://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

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

In [34]:
```
observation = [[1.4,1.5,1.6,2.7,2.3,3.3,2.3,4.1,2.3,4.2,1.2,2.1,4.3,6,2.2]]
prediction = logs.predict(observation)
```

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

```
[28079099]
```

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

Out[36]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079018, 28079019, 28079021, 28079022, 28079023, 28079024,
       28079025, 28079026, 28079027, 28079036, 28079038, 28079039,
       28079040, 28079099], 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))
```

```
0.03900291528019082
```

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

```
0.03871297499905347
```

Conclusion

Ridge regression is bestfit model

Ridge regression is best fit model for dataset madrid_2001. The score of x_train,y_train is 0.6997214702742314 and x_test and y_test score is 0.7029800822222487.