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	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.2099
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.3899
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.2400
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.8399
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.7799
...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.3800
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.4000
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.8300
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.5700
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.3500

243984 rows × 16 columns

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

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	0.00	1.72	0.00	0.00	0.00	73.900002	316.299988	0.00	10.550000	55.2099
1	2003-03-01 01:00:00	0.00	1.45	0.00	0.00	0.26	72.110001	250.000000	0.73	6.720000	52.3899
2	2003-03-01 01:00:00	0.00	1.57	0.00	0.00	0.00	80.559998	224.199997	0.00	21.049999	63.2400
3	2003-03-01 01:00:00	0.00	2.45	0.00	0.00	0.00	78.370003	450.399994	0.00	4.220000	67.8399
4	2003-03-01 01:00:00	0.00	3.26	0.00	0.00	0.00	96.250000	479.100006	0.00	8.460000	95.7799
...	...	...	...	...	...	...	...	...	...	...	
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.3800
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	0.00	10.450000	14.760000	1.00	34.610001	7.4000
243981	2003-10-01 00:00:00	0.00	0.00	0.00	0.00	0.07	34.639999	50.810001	0.00	32.160000	16.8300
243982	2003-10-01 00:00:00	0.00	0.00	0.00	0.00	0.07	32.580002	41.020000	0.00	0.000000	13.5700
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.3500

243984 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243984 entries, 0 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     243984 non-null  object 
 1   BEN      243984 non-null  float64
 2   CO       243984 non-null  float64
 3   EBE      243984 non-null  float64
 4   MXY      243984 non-null  float64
 5   NMHC     243984 non-null  float64
 6   NO_2     243984 non-null  float64
 7   NOx      243984 non-null  float64
 8   OXY      243984 non-null  float64
 9   O_3      243984 non-null  float64
 10  PM10     243984 non-null  float64
 11  PXY      243984 non-null  float64
 12  SO_2     243984 non-null  float64
 13  TCH      243984 non-null  float64
 14  TOL      243984 non-null  float64
 15  station  243984 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 29.8+ MB
```

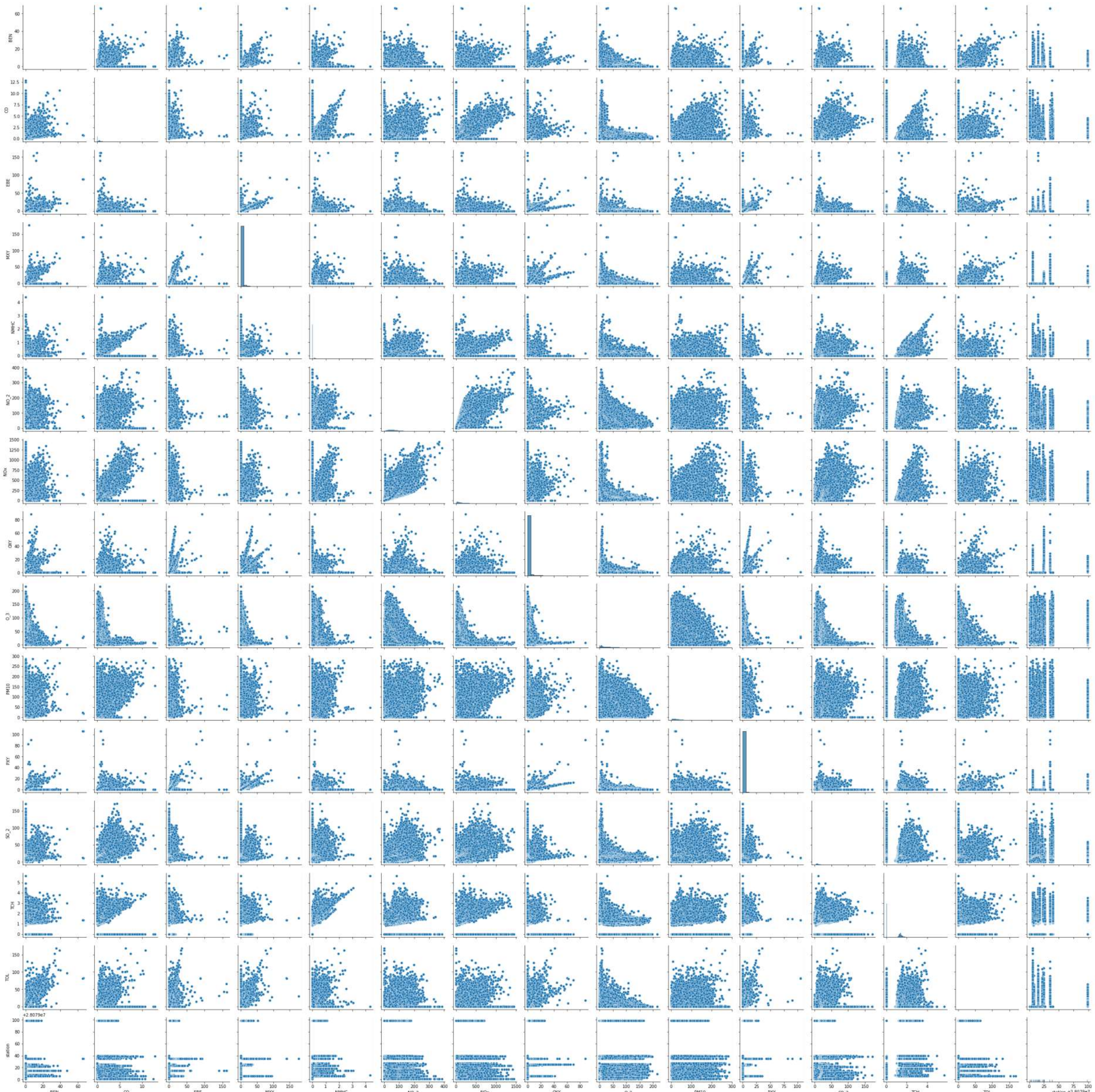
In [5]:
```
df1.columns
```

Out[5]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', '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)
```

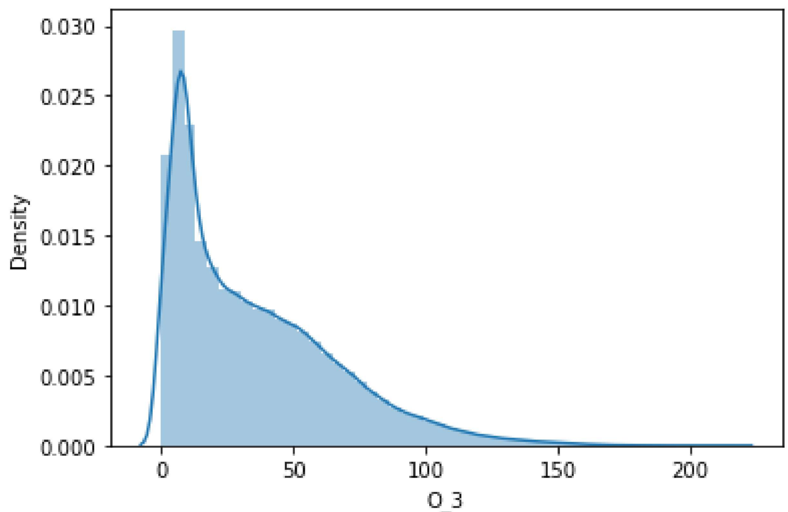
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x1d8c13962e0>
```



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

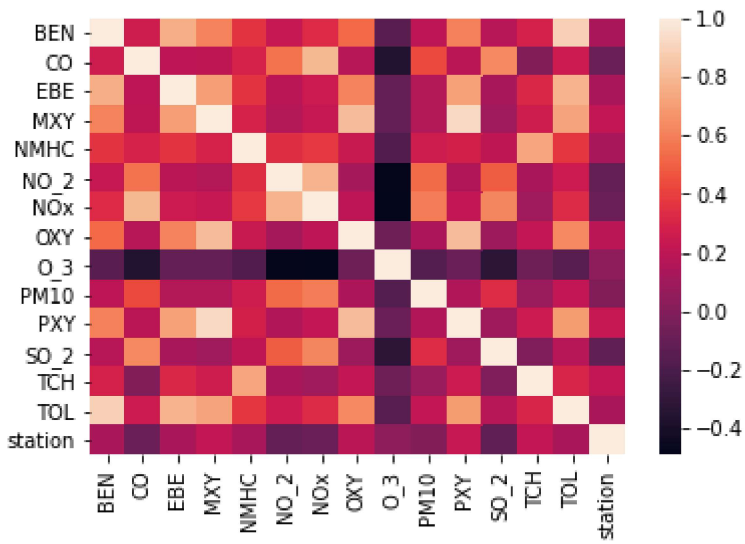
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function 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 [40]:
```
x = df2[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY','O_3',
       'PM10', 'PXY', 'SO_2', 'TCH']]
y = df2['TOL']
```

In [41]:
```
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 [42]:
```
from sklearn.linear_model import LinearRegression

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

Out[42]:
```
LinearRegression()
```

In [43]:
```
print(lr.intercept_)
```

```
0.05153717878677311
```

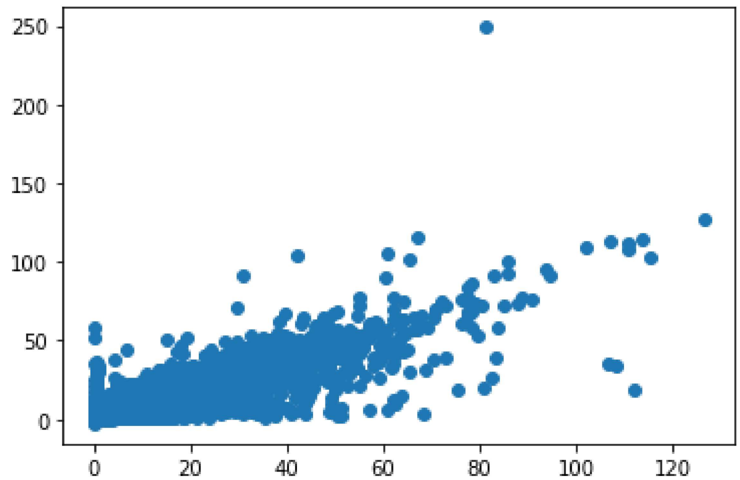
In [44]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[44]:

	Co-efficient
BEN	2.751783
CO	-0.413755
EBE	0.409060
MXY	0.615866
NMHC	0.462169
NO_2	0.000508
NOx	0.003124
OXY	0.261635
O_3	-0.001993
PM10	0.002020
PXY	-0.517661
SO_2	0.006842
TCH	0.080668

In [45]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[45]: <matplotlib.collections.PathCollection at 0x1d8ee99c130>



In [46]:
```
print(lr.score(x_test,y_test))
```

0.8588823576390503

In [47]:
```
lr.score(x_train,y_train)
```

Out[47]: 0.8522993489926665

Ridge and Lasso

In [48]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [49]:
```
rr = Ridge(alpha=10)
rr.fit(x_train,y_train)
rr.score(x_train,y_train)
```

Out[49]: 0.852299345539862

In [50]:
```
rr.score(x_test,y_test)
```

Out[50]: 0.8588808675901978

In [51]:
```
ls = Lasso(alpha=10)
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
```

Out[51]: 0.3420746807446454

In [52]:
```
ls.score(x_test,y_test)
```

Out[52]: 0.3523943809006804

ElacticNET regression

In [53]:
```
from sklearn.linear_model import ElasticNet
es = ElasticNet()
es.fit(x_train,y_train)
```

Out[53]:
```
ElasticNet()
```

In [54]:
```
print(es.coef_)
```

```
[ 1.83576883e+00 -0.00000000e+00  6.35732366e-01  6.11793843e-01
  0.00000000e+00  0.00000000e+00  4.90226757e-03  2.25159780e-02
 -1.24296046e-03  1.78610812e-03  0.00000000e+00  0.00000000e+00
  0.00000000e+00]
```

In [55]:
```
print(es.intercept_)
```

```
0.08538665806495027
```

In [56]:
```
print(es.score(x_test,y_test))
```

```
0.8307640903754144
```

In [57]:
```
print(es.score(x_train,y_train))
```

```
0.827585164995586
```

LogisticRegression

In [58]:
```
from sklearn.linear_model import LogisticRegression
```

In [59]:
```
feature_matrix = df2.iloc[:,0:15]
target_vector = df2.iloc[:,-1]
```

In [60]:
```
feature_matrix.shape
```

Out[60]:
```
(243984, 15)
```

In [61]:
```
from sklearn.preprocessing import StandardScaler
```

In [62]:
```
fs = StandardScaler().fit_transform(feature_matrix)
```

In [63]:
```
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[63]:
```
LogisticRegression()
```

In [64]:
```
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 [65]:
```
print(prediction)
```

```
[28079035]
```

In [66]:
```
logs.classes_
```

Out[66]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079017, 28079018, 28079019, 28079021, 28079022, 28079023,
       28079024, 28079025, 28079026, 28079027, 28079035, 28079036,
       28079038, 28079039, 28079040, 28079099], dtype=int64)
```

In [67]:
```
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 [68]:
```
print(logs.score(x_test,y_test))
```

```
0.035807967648505384
```

In [69]:
```
print(logs.score(x_train,y_train))
```

```
0.035945148371079934
```

Conclusion

Linear regression is bestfit model

Linear regression is best fit model for dataset madrid_2001. The score of x_train,y_train is 0.8588823576390503 and x_test and y_test score is 0.8522993489926665.

In [ ]:

```

```