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

In [2]:
```
data=pd.read_csv(r"C:\Users\user\Downloads\madrid_2003.csv")
data
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.209
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.389
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.240
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.839
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.779
...	...	...	...	...	...	...	...	...	...	...	
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	7.380
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.400
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.830
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.570
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	12.350

243984 rows × 16 columns

In [3]:
```
data.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      69745 non-null   float64
 2   CO       225340 non-null  float64
 3   EBE      61244 non-null   float64
 4   MXY      42045 non-null   float64
 5   NMHC     111951 non-null  float64
 6   NO_2     242625 non-null  float64
 7   NOx      242629 non-null  float64
 8   OXY      42072 non-null   float64
 9   O_3      234131 non-null  float64
 10  PM10     240896 non-null  float64
 11  PXY      42063 non-null   float64
 12  SO_2     242729 non-null  float64
 13  TCH      111991 non-null  float64
 14  TOL      69439 non-null   float64
 15  station  243984 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 29.8+ MB
```

In [4]:
```
df=data.fillna(value=0)
df
```

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
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	55.209
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	52.389
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	63.240
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	67.839
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	95.779
...	...	...	...	...	...	...	...	...	...	...	
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	7.380
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	7.400
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	16.830
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	13.570
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	12.350

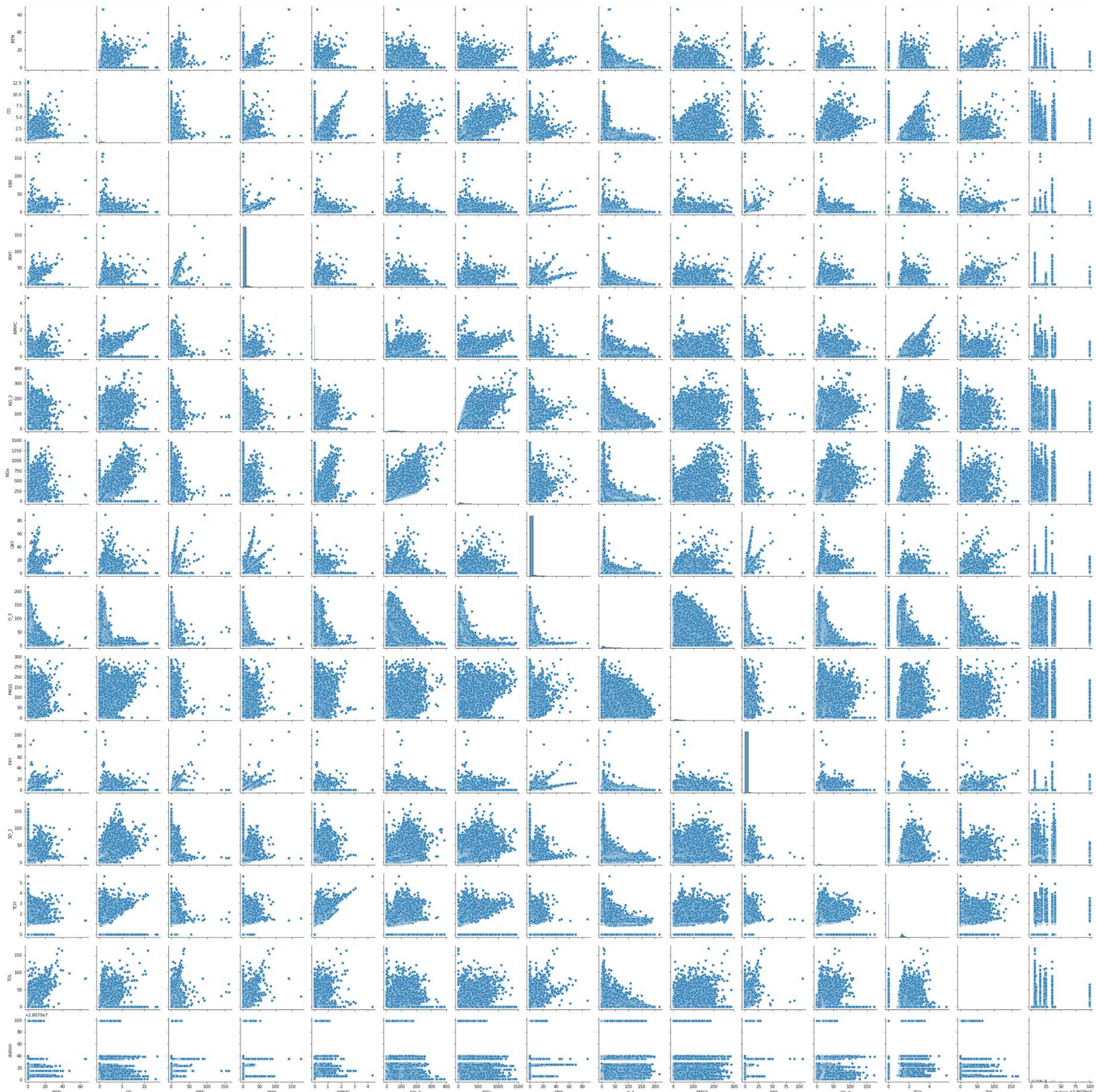
243984 rows × 16 columns

In [5]:
```
df.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]:
```
sns.pairplot(df)
```

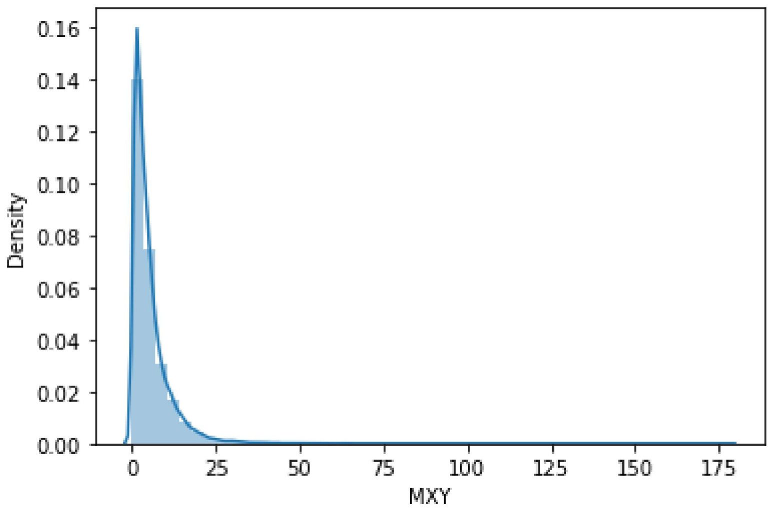
Out[6]: <seaborn.axisgrid.PairGrid at 0x161a78f4df0>



In [7]:
```
sns.distplot(data["MXY"])
```

```
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[7]: <AxesSubplot:xlabel='MXY', ylabel='Density'>



MODEL BUILDING

Linear Regression

In [8]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [9]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
y=df1[['MXY']]
```

In [10]:
```
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)
```

In [11]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

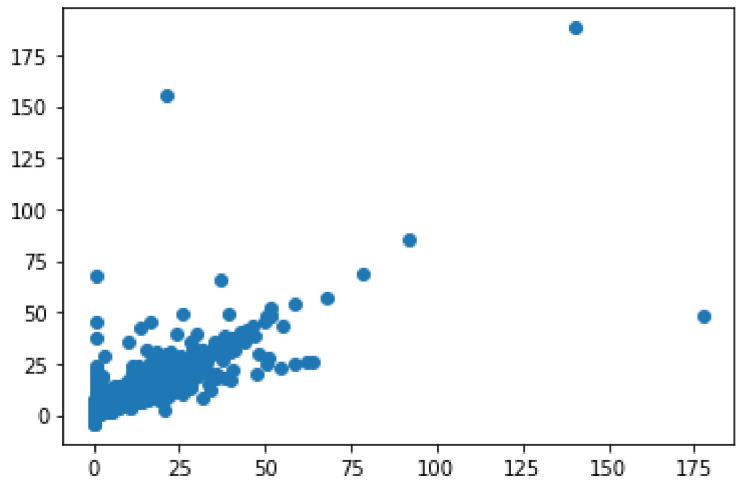
Out[11]: LinearRegression()

In [12]:
```
print(lr.intercept_)
```

```
[-29432.32092764]
```

In [13]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[13]: <matplotlib.collections.PathCollection at 0x161d16c7550>



In [14]:
```
print(lr.score(x_test,y_test))
```

```
0.869466955365737
```

Ridge Regression

In [15]:
```
from sklearn.linear_model import Ridge
```

In [16]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[16]: Ridge(alpha=10)

In [17]:
```
rr.score(x_test,y_test)
```

Out[17]: 0.8694728537229582

Lasso Regression

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

In [19]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```
Out[19]:
```
Lasso(alpha=10)
```

In [20]:
```
la.score(x_test,y_test)
```
Out[20]:
```
0.30547107421562425
```

Elastic Regression

In [21]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```
Out[21]:
```
ElasticNet()
```

In [22]:
```
print(en.coef_)
```
```
[-0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  4.50842333e-04  3.94048154e-01 -0.00000000e+00
  0.00000000e+00  7.05229617e-01 -0.00000000e+00  0.00000000e+00
  1.70269794e-01  9.29585752e-03]
```

In [23]:
```
print(en.predict(x_test))
```
```
[ 0.04897889  0.08734813  0.40241648 ... -0.0910591  -0.06200938
 -0.0707467 ]
```

In [24]:
```
print(en.score(x_test,y_test))
```
```
0.7981828312334192
```

Logistic Regression

In [25]:
```
from sklearn.linear_model import LogisticRegression
```

In [26]:
```
feature_matrix=df1.iloc[:,0:15]
target_vector=df1.iloc[:,-1]
```

In [27]:
```
feature_matrix.shape
```
Out[27]:
```
(243984, 15)
```

In [28]:
```
target_vector.shape
```
Out[28]:
```
(243984,)
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

In [30]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [31]:
```
logr=LogisticRegression()
logr.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[31]:
```
LogisticRegression()
```

In [32]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [33]:
```
prediction=logr.predict(observation)
print(observation)
```

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]]
```

In [34]:
```
logr.classes_
```

Out[34]:
```
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 [35]:
```
logr.score(fs,target_vector)
```

Out[35]:
```
0.9361966358449735
```

Random Forest

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [39]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
y=df1['station']
```

In [40]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.70)
```

In [41]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[41]:
```
RandomForestClassifier()
```

In [42]:
```
parameters={'max_depth':[1,2,3,4,5],
           'min_samples_leaf':[5,10,15,20,25],
           'n_estimators':[10,20,30,40,50]}
```

In [*]:
```
from sklearn.model_selection import GridSearchCV
grid_search=GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring='acc
grid_search.fit(x_train,y_train)
```

In [48]:
```
grid_search.best_score_
```

Out[48]:
```
0.7832223177646518
```

In [49]:
```
rfc_best=grid_search.best_estimator_
```

In [46]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

```
[0, 0, 0, 311, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 1108, 1442, 0, 0,
873, 0, 0, 0, 0\n919]'),
 Text(3224.0, 543.5999999999999, 'SO_2 <= 3.21\ngini = 0.26\nsamples = 106
6\nvalue = [0, 0, 0, 150, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 13, 14
42, 0, 0, 83, 0, 0, 0, 0\n0]'),
 Text(3100.0, 181.19999999999982, 'gini = 0.085\nsamples = 29\nvalue = [0,
0, 0, 43, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
0, 0, 0]'),
 Text(3348.0, 181.19999999999982, 'gini = 0.225\nsamples = 1037\nvalue =
[0, 0, 0, 107, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 13, 1440, 0, 0, 8
3, 0, 0, 0, 0\n0]'),
 Text(3720.0, 543.5999999999999, 'PXY <= 0.835\ngini = 0.694\nsamples = 18
66\nvalue = [0, 0, 0, 161, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 1095,
0, 0, 0, 790, 0, 0, 0, 0\n919]'),
 Text(3596.0, 181.19999999999982, 'gini = 0.566\nsamples = 706\nvalue =
[0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 690, 0, 0, 0, 269,
0, 0, 0, 0\n171]'),
 Text(3844.0, 181.19999999999982, 'gini = 0.691\nsamples = 1160\nvalue =
[0, 0, 0, 136, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 405, 0, 0, 0, 52
1, 0, 0, 0, 0\n748]'),
```

Results

The best model is Logistic Regression 0.9361966358449735