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_2006.csv")
data
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.880000	97.57
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.82
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.430000	34.41
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.830000	28.26
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.990000	54.18
...	...	...	...	...	...	...	...	...	...	...	
230563	2006-05-01 00:00:00	5.88	0.83	6.23	NaN	0.20	112.500000	218.000000	NaN	24.389999	93.12
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29.46
230565	2006-05-01 00:00:00	0.96	NaN	0.69	NaN	0.19	135.100006	179.199997	NaN	11.460000	64.68
230566	2006-05-01 00:00:00	0.50	NaN	0.67	NaN	0.10	82.599998	105.599998	NaN	NaN	94.36
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52.49

230568 rows × 17 columns

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230568 entries, 0 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     230568 non-null  object 
 1   BEN      73979 non-null   float64
 2   CO       211665 non-null  float64
 3   EBE      73948 non-null   float64
 4   MXY      33422 non-null   float64
 5   NMHC     90829 non-null   float64
 6   NO_2     228855 non-null  float64
 7   NOx      228855 non-null  float64
 8   OXY      33472 non-null   float64
 9   O_3      216511 non-null  float64
 10  PM10     227469 non-null  float64
 11  PM25     61758 non-null   float64
 12  PXY      33447 non-null   float64
 13  SO_2     229125 non-null  float64
 14  TCH      90887 non-null   float64
 15  TOL      73840 non-null   float64
 16  station  230568 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.9+ MB
```

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

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	P
0	2006-02-01 01:00:00	0.00	1.84	0.00	0.00	0.00	155.100006	490.100006	0.00	4.880000	97.57
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.82
2	2006-02-01 01:00:00	0.00	1.25	0.00	0.00	0.00	66.800003	192.000000	0.00	4.430000	34.41
3	2006-02-01 01:00:00	0.00	1.68	0.00	0.00	0.00	103.000000	407.799988	0.00	4.830000	28.26
4	2006-02-01 01:00:00	0.00	1.31	0.00	0.00	0.00	105.400002	269.200012	0.00	6.990000	54.18
...	...	...	...	...	...	...	...	...	...	...	
230563	2006-05-01 00:00:00	5.88	0.83	6.23	0.00	0.20	112.500000	218.000000	0.00	24.389999	93.12
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29.46
230565	2006-05-01 00:00:00	0.96	0.00	0.69	0.00	0.19	135.100006	179.199997	0.00	11.460000	64.68
230566	2006-05-01 00:00:00	0.50	0.00	0.67	0.00	0.10	82.599998	105.599998	0.00	0.000000	94.36
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52.49

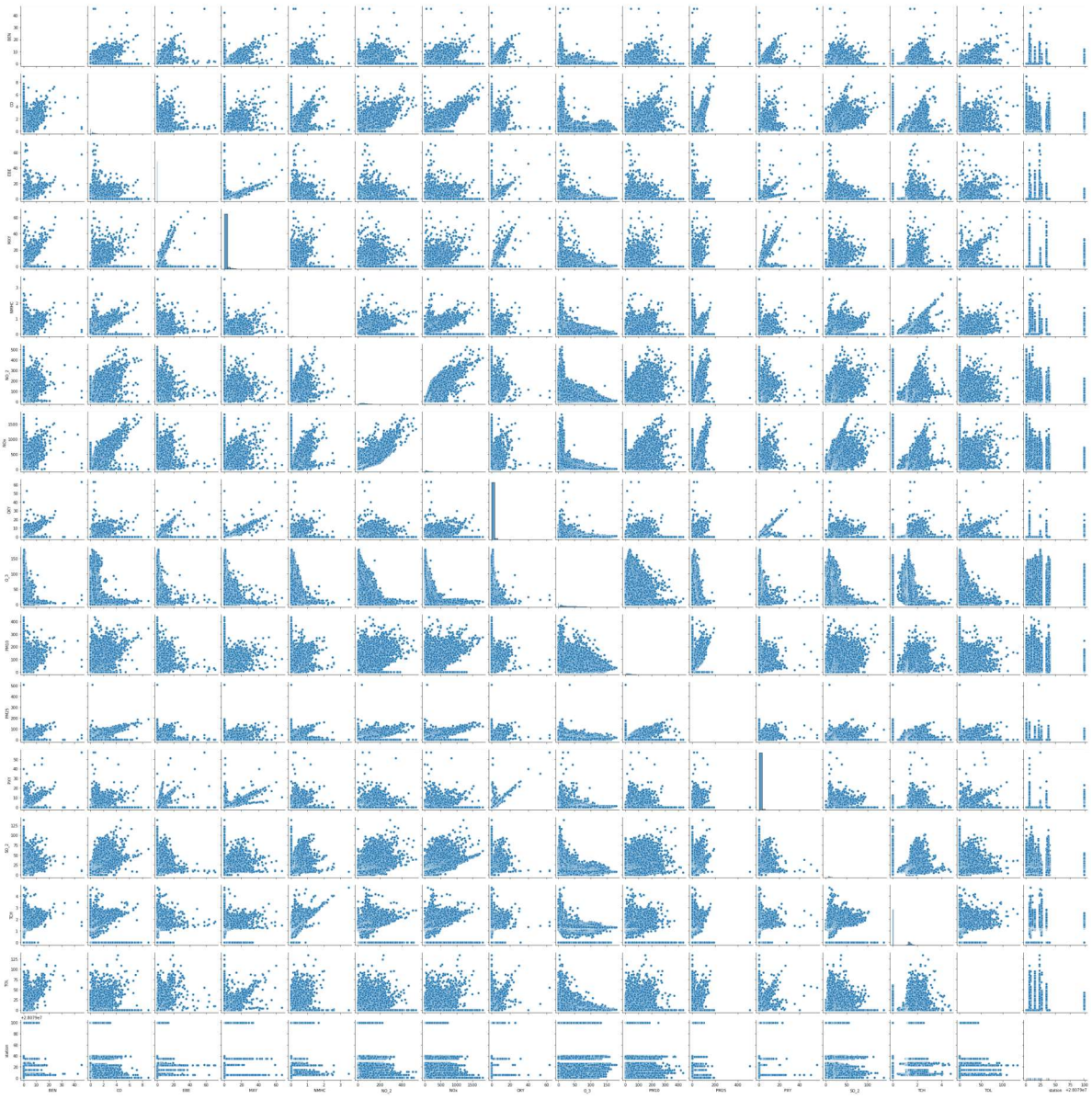
230568 rows × 17 columns

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

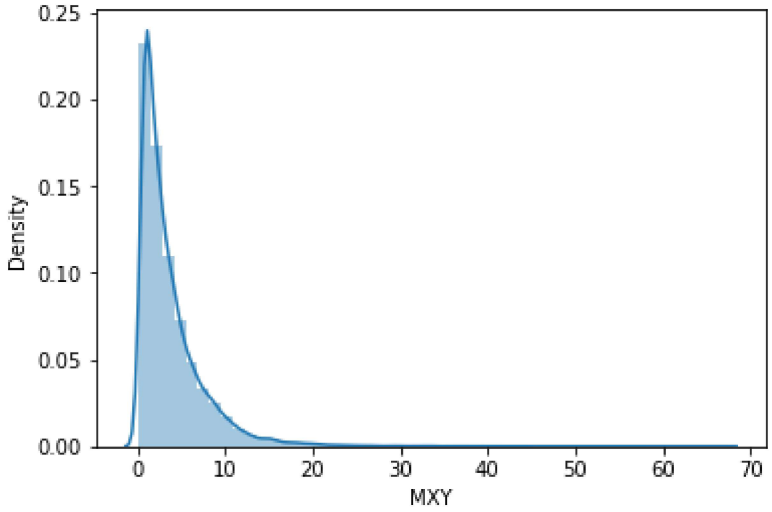
Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x2d5f9741c70>
```



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

```
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[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)
```

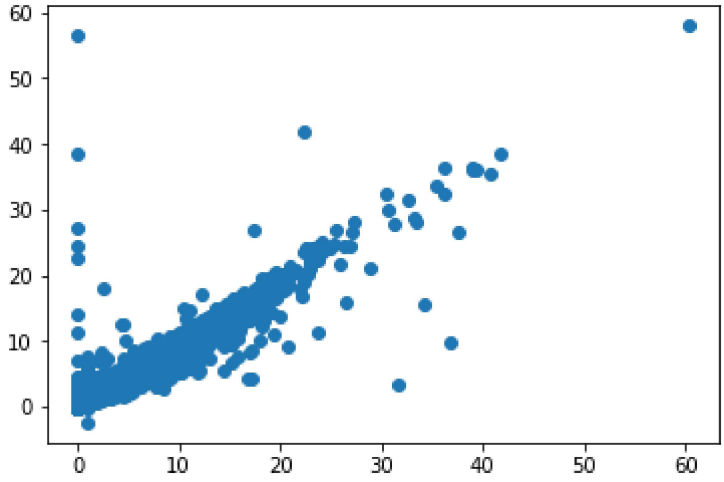
Out[11]:
```
LinearRegression()
```

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

```
[-21421.9865118]
```

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

Out[13]:
```
<matplotlib.collections.PathCollection at 0x2d59894d0d0>
```



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

```
0.9267060020053357
```

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]:
```
0.9267070700237199
```

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]:
```
0.04809089726963345
```

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.          0.          0.          0.          0.          0.00131568
  0.52948898  0.         -0.0006754   0.19728964  0.          0.
  0.13238778  0.01146552]
```

In [23]:
```
print(en.predict(x_test))
```

```
[ 1.92201814  2.76911128  1.05761045 ...  0.5440165  -0.16558609
 -0.05781555]
```

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

```
0.7196134103451814
```

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]:
```
(230568, 15)
```

In [28]:
```
target_vector.shape
```

Out[28]:
```
(230568,)
```

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://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[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,
       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]:
```
0.9029136740571111
```

Random Forest

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

In [37]:
```
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 [38]:
```
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 [39]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[39]:
```
RandomForestClassifier()
```

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

In [41]:
```
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)
```

Out[41]:
```
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 [42]:
```
grid_search.best_score_
```

Out[42]:
```
0.8118548503686569
```

In [43]:
```
rfc_best=grid_search.best_estimator_
```

In [44]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

```
\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 54, 0, 0, 0\n0, 0, 893, 0, 0, 0,
9, 0, 0, 0, 0, 0, 0]'),
 Text(1974.4615384615383, 181.19999999999982, 'gini = 0.215\nsamples = 306
\nvalue = [0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 436, 0, 0, 0\n0, 0, 36, 0, 0, 7,
8, 0, 0, 0, 0, 0, 0]'),
 Text(2232.0, 543.5999999999999, 'O_3 <= 5.89\ngini = 0.739\nsamples = 191
2\nvalue = [0, 0, 0, 2, 0, 282, 0, 0, 0, 0, 1248, 0, 0\n0, 0, 0, 670, 0,
0, 493, 375, 0, 0, 0, 0, 0\n6]'),
 Text(2146.153846153846, 181.19999999999982, 'gini = 0.731\nsamples = 691
\nvalue = [0, 0, 0, 0, 0, 123, 0, 0, 0, 0, 43, 0, 0, 0\n0, 0, 204, 0, 0, 3
53, 375, 0, 0, 0, 0, 0, 0]'),
 Text(2317.846153846154, 181.19999999999982, 'gini = 0.562\nsamples = 1221
\nvalue = [0, 0, 0, 2, 0, 159, 0, 0, 0, 0, 1205, 0, 0\n0, 0, 0, 466, 0, 0,
140, 0, 0, 0, 0, 0, 0, 6]'),
 Text(3219.230769230769, 1630.8000000000002, 'TCH <= 0.34\ngini = 0.749\ns
amples = 6268\nvalue = [0, 0, 0, 2553, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0,
0, 2444, 0, 0, 0, 2298, 0, 0, 0, 0, 2653]'),
 Text(2661.230769230769, 1268.4, 'NO_2 <= 13.22\ngini = 0.097\nsamples = 6
54\nvalue = [0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 17, 0, 0,
0, 965, 0, 0, 0, 0, 25]'),
```

Results

The best model is Ridge Regression 0.9267070700237199

In [ ]: