Final Assessment 1

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_2002.csv")
data
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM1(
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.99000:
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.85	20.98000(
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.44000
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.18000(
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.33000:
...	...	...	...	...	...	...	...	...	...	...	..
217291	2002-11-01 00:00:00	4.16	1.14	NaN	NaN	NaN	81.080002	265.700012	NaN	7.21	36.75000(
217292	2002-11-01 00:00:00	3.67	1.73	2.89	NaN	0.38	113.900002	373.100006	NaN	5.66	63.38999!
217293	2002-11-01 00:00:00	1.37	0.58	1.17	2.37	0.15	65.389999	107.699997	1.30	9.11	9.64000(
217294	2002-11-01 00:00:00	4.51	0.91	4.83	10.99	NaN	149.800003	202.199997	1.00	5.75	Nal
217295	2002-11-01 00:00:00	3.11	1.17	3.00	7.77	0.26	80.110001	180.300003	2.25	7.38	29.24000(

217296 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217296 entries, 0 to 217295
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     217296 non-null  object 
 1   BEN      66747 non-null   float64
 2   CO       216637 non-null  float64
 3   EBE      58547 non-null   float64
 4   MXY      41255 non-null   float64
 5   NMHC     87045 non-null   float64
 6   NO_2     216439 non-null  float64
 7   NOx      216439 non-null  float64
 8   OXY      41314 non-null   float64
 9   O_3      216726 non-null  float64
 10  PM10     209113 non-null  float64
 11  PXY      41256 non-null   float64
 12  SO_2     216507 non-null  float64
 13  TCH      87115 non-null   float64
 14  TOL      66619 non-null   float64
 15  station  217296 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 26.5+ MB
```

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

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM1
0	2002-04-01 01:00:00	0.00	1.39	0.00	0.00	0.00	145.100006	352.100006	0.00	6.54	41.99000
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.20	0.15	98.150002	153.399994	2.67	6.85	20.98000
2	2002-04-01 01:00:00	0.00	0.80	0.00	0.00	0.00	103.699997	134.000000	0.00	13.01	28.44000
3	2002-04-01 01:00:00	0.00	1.61	0.00	0.00	0.00	97.599998	268.000000	0.00	5.12	42.18000
4	2002-04-01 01:00:00	0.00	1.90	0.00	0.00	0.00	92.089996	237.199997	0.00	7.28	76.33000
...	...	...	...	...	...	...	...	...	...	...	..
217291	2002-11-01 00:00:00	4.16	1.14	0.00	0.00	0.00	81.080002	265.700012	0.00	7.21	36.75000
217292	2002-11-01 00:00:00	3.67	1.73	2.89	0.00	0.38	113.900002	373.100006	0.00	5.66	63.38999
217293	2002-11-01 00:00:00	1.37	0.58	1.17	2.37	0.15	65.389999	107.699997	1.30	9.11	9.64000
217294	2002-11-01 00:00:00	4.51	0.91	4.83	10.99	0.00	149.800003	202.199997	1.00	5.75	0.00000
217295	2002-11-01 00:00:00	3.11	1.17	3.00	7.77	0.26	80.110001	180.300003	2.25	7.38	29.24000

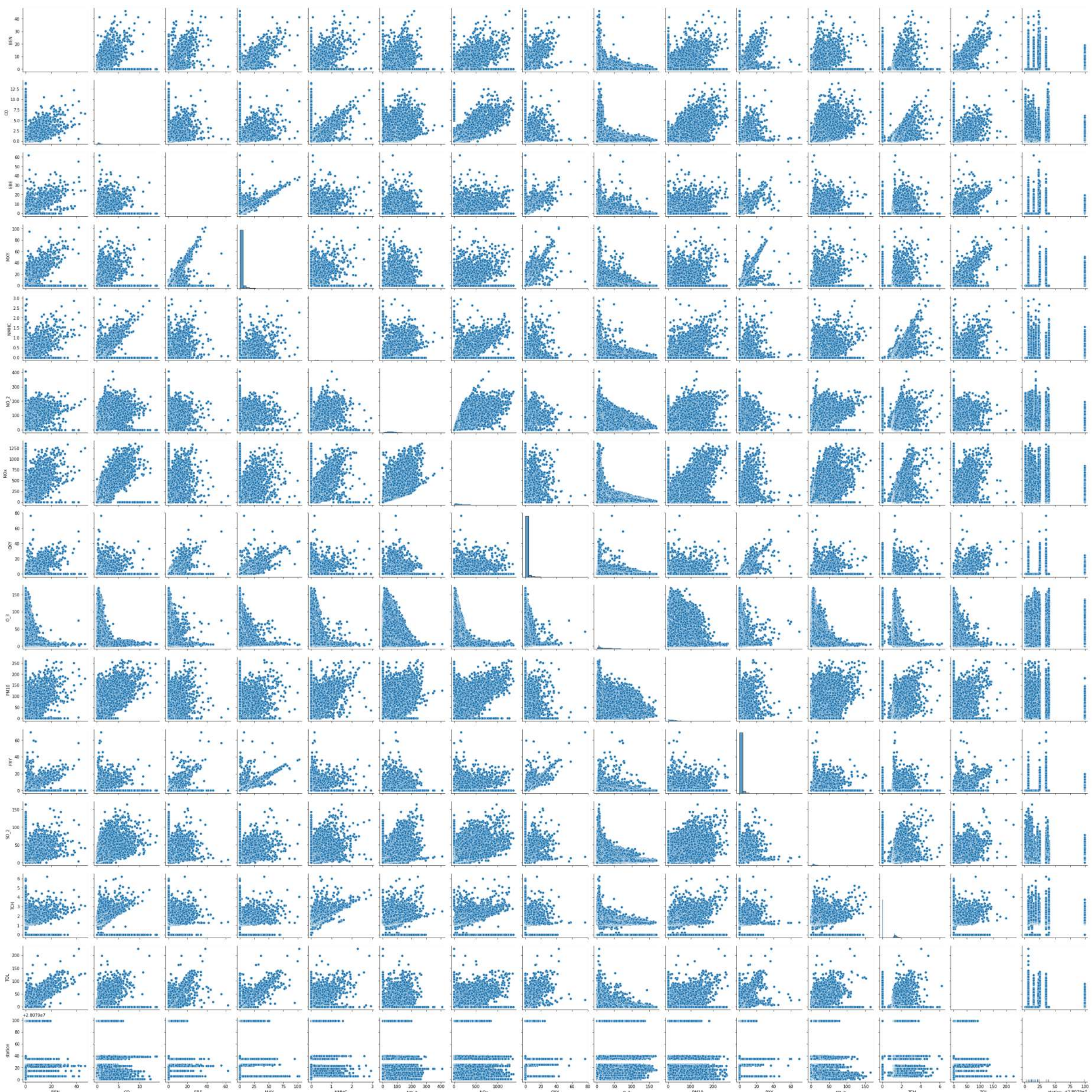
217296 rows × 16 columns

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)
```

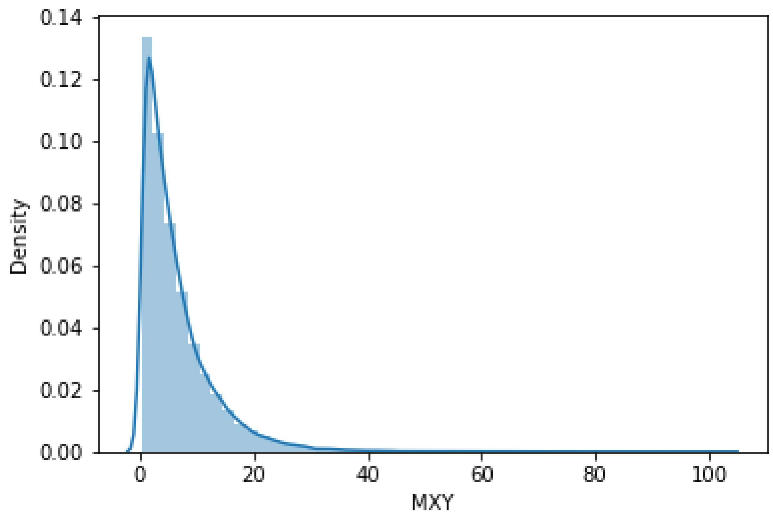
Out[6]: <seaborn.axisgrid.PairGrid at 0x2094fdf3a90>



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)
```

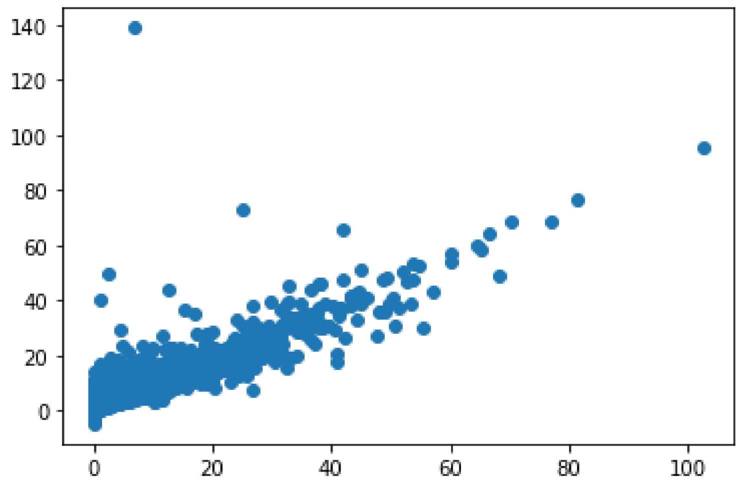
Out[11]:
```
LinearRegression()
```

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

```
[-114854.33120557]
```

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

Out[13]: <matplotlib.collections.PathCollection at 0x2090dff6100>



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

```
0.9238917627616073
```

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]: 0.9238912222984711

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]:
```
0.5518320220956707
```

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.          0.00184403 -0.          0.          0.
  0.64019889  0.         -0.00089491  0.58239666  0.00587193  0.
  0.17912466  0.00796413]
```

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

```
[-0.0193248  -0.23557702  0.20837586 ... -0.10215646 -0.04900807
 -0.15602359]
```

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

```
0.8857186530810414
```

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]:
```
(217296, 15)
```

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

Out[28]:
```
(217296,)
```

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, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079099], dtype=int64)
```

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

Out[35]:
```
0.923532876813195
```

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 [43]:
```
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[43]:
```
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 [44]:
```
grid_search.best_score_
```

Out[44]:
```
0.8196140393937534
```

In [45]:
```
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)
```

```
 Text(2999.25, 181.19999999999982, 'gini = 0.029\nsamples = 669\nvalue =
[0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 11, 1025, 0, 0, 0,
0, 0, 0]'),
 Text(3208.5, 543.5999999999999, 'NOx <= 110.4\ngini = 0.351\nsamples = 72
\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 92,
0, 0, 0, 0, 27]'),
 Text(3138.75, 181.19999999999982, 'gini = 0.198\nsamples = 52\nvalue =
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 72, 0, 0, 0,
0, 9]'),
 Text(3278.25, 181.19999999999982, 'gini = 0.499\nsamples = 20\nvalue =
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 20, 0, 0, 0,
0, 18]'),
 Text(3906.0, 1268.4, 'NMHC <= 0.005\ngini = 0.764\nsamples = 5226\nvalue
= [0, 0, 0, 2146, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 1012, 631, 216
6, 0, 0, 0, 0, 2366]'),
 Text(3627.0, 906.0, 'SO_2 <= 14.885\ngini = 0.549\nsamples = 674\nvalue =
[0, 0, 0, 349, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 6, 631, 98, 0, 0,
0, 0, 0]'),
 Text(3487.5, 543.5999999999999, 'station <= 28079015.0\ngini = 0.529\nsam
ples = 295\nvalue = [0, 0, 0, 315, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0,
```

Results

```
The best model is Linear Regression 0.9238917627616073
```