Final Assessment 1

Kaviyadevi(20106064)

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

In [2]:
```
#importing dataset
data=pd.read_csv(r"C:\Users\user\Downloads\madrid_2001.csv")
data
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	8/1/2001 1:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	105.000
1	8/1/2001 1:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.599
2	8/1/2001 1:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.099
3	8/1/2001 1:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.779
4	8/1/2001 1:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.180
...	...	...	...	...	...	...	...	...	...	...	
217867	4/1/2001 0:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	47.880
217868	4/1/2001 0:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	26.809
217869	4/1/2001 0:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.770
217870	4/1/2001 0:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	37.889
217871	4/1/2001 0:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.369

217872 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217872 entries, 0 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     217872 non-null  object 
 1   BEN      70389 non-null   float64
 2   CO       216341 non-null  float64
 3   EBE      57752 non-null   float64
 4   MXY      42753 non-null   float64
 5   NMHC     85719 non-null   float64
 6   NO_2     216331 non-null  float64
 7   NOx      216318 non-null  float64
 8   OXY      42856 non-null   float64
 9   O_3      216514 non-null  float64
 10  PM10     207776 non-null  float64
 11  PXY      42845 non-null   float64
 12  SO_2     216403 non-null  float64
 13  TCH      85797 non-null   float64
 14  TOL      70196 non-null   float64
 15  station  217872 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 26.6+ MB
```

In [5]:
```
#filling null values
df=data.fillna(0)
df
```

Out[5]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PI
0	8/1/2001 1:00	0.00	0.37	0.00	0.00	0.00	58.400002	87.150002	0.00	34.529999	105.000
1	8/1/2001 1:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.599
2	8/1/2001 1:00	0.00	0.28	0.00	0.00	0.00	50.660000	61.380001	0.00	46.310001	100.099
3	8/1/2001 1:00	0.00	0.47	0.00	0.00	0.00	69.790001	73.449997	0.00	40.650002	69.779
4	8/1/2001 1:00	0.00	0.39	0.00	0.00	0.00	22.830000	24.799999	0.00	66.309998	75.180
...	...	...	...	...	...	...	...	...	...	...	
217867	4/1/2001 0:00	10.45	1.81	0.00	0.00	0.00	73.000000	264.399994	0.00	5.200000	47.880
217868	4/1/2001 0:00	5.20	0.69	4.56	0.00	0.13	71.080002	129.300003	0.00	13.460000	26.809
217869	4/1/2001 0:00	0.49	1.09	0.00	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.770
217870	4/1/2001 0:00	5.62	1.01	5.04	11.38	0.00	80.019997	197.000000	2.58	5.840000	37.889
217871	4/1/2001 0:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.369

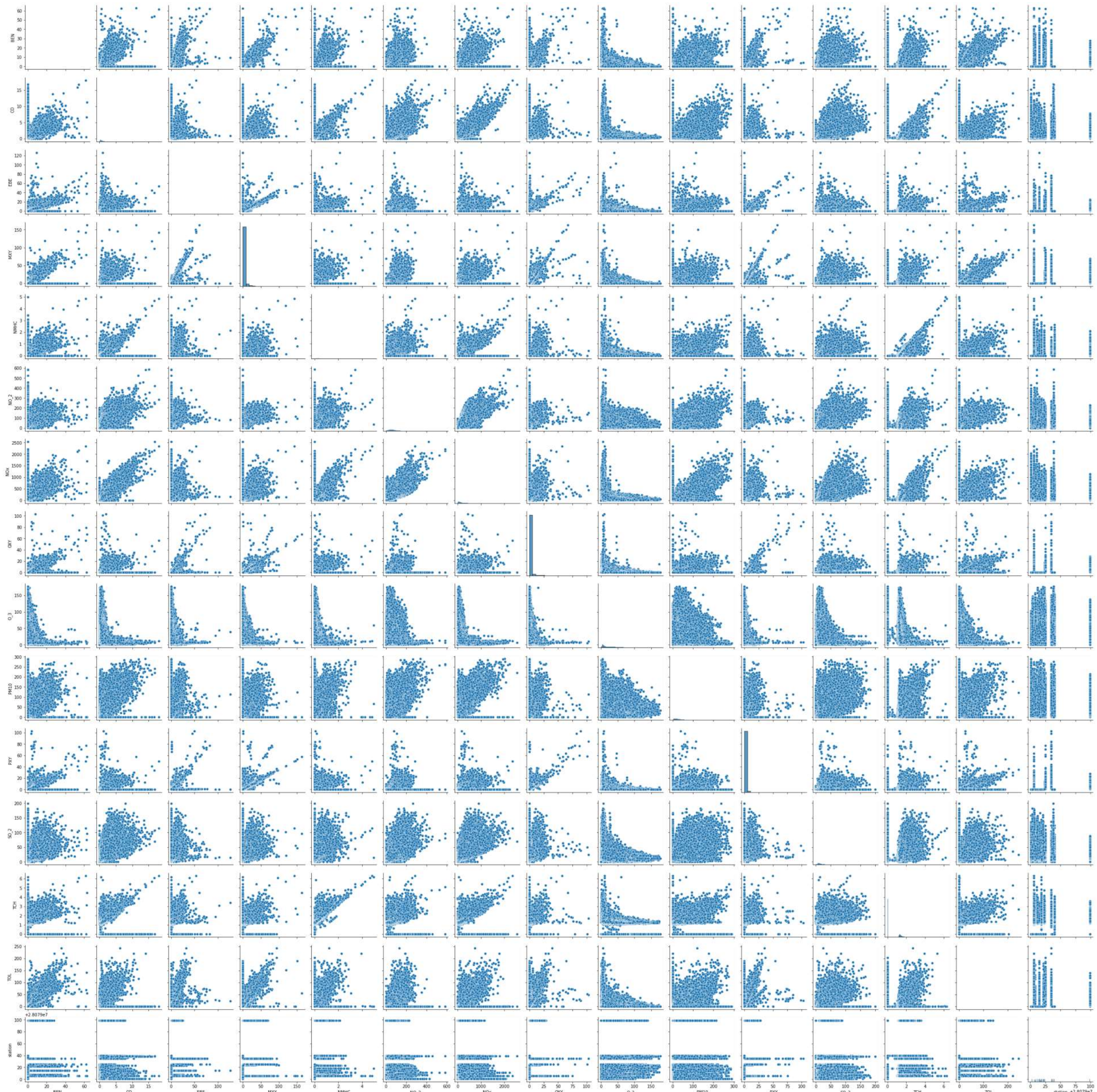
217872 rows × 16 columns

In [6]:
```
df.columns
```

Out[6]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

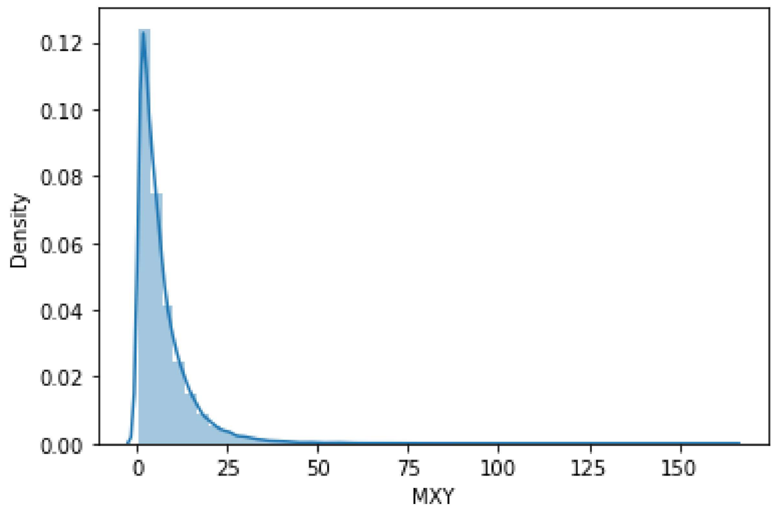
Out[7]: <seaborn.axisgrid.PairGrid at 0x269e6d76970>



```
In [8]: sns.distplot(data["MXY"])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Futur
eWarning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

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



MODEL BUILDING

1.Linear Regression

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

```
In [10]: x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
         y=df1[['MXY']]
```

```
In [11]: #split the dataset into trainning and test
         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 [12]: from sklearn.linear_model import LinearRegression
         lr=LinearRegression()
         lr.fit(x_train,y_train)
```

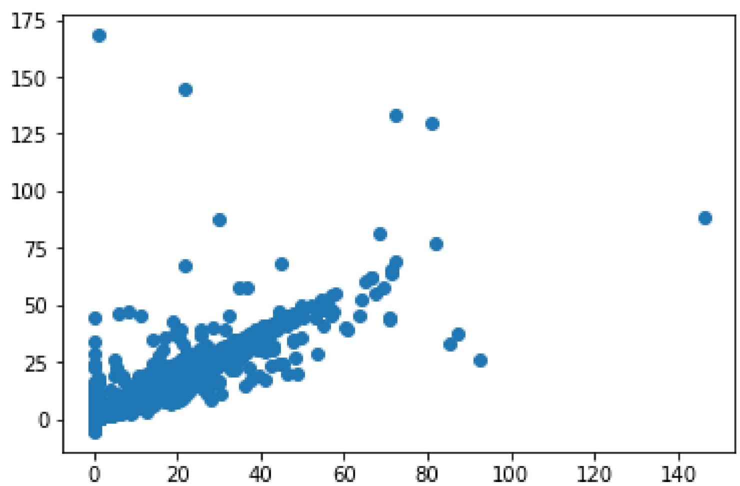
```
Out[12]: LinearRegression()
```

In [13]:
```
print(lr.intercept_)
```

```
[0.01226058]
```

In [14]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[14]:
```
<matplotlib.collections.PathCollection at 0x269f39bea60>
```



In [15]:
```
print(lr.score(x_test,y_test))
```

```
0.8870068844960968
```

2.Ridge Regression

In [16]:
```
from sklearn.linear_model import Ridge
```

In [17]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[17]:
```
Ridge(alpha=10)
```

In [18]:
```
rr.score(x_test,y_test)
```

Out[18]:
```
0.8870086109906489
```

3.Lasso Regression

In [19]:
```
from sklearn.linear_model import Lasso
```

In [20]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[20]:
```
Lasso(alpha=10)
```

In [21]:
```
la.score(x_test,y_test)
```

Out[21]:
```
0.5104330787352147
```

4.ElasticNet Regression

In [22]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[22]:
```
ElasticNet()
```

In [23]:
```
print(en.coef_)
```

```
[-0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
 -5.65114828e-04  5.87169395e-04  8.91091911e-01 -2.97518722e-03
  5.13046364e-01  4.40821204e-03  0.00000000e+00  1.17194898e-01
  1.31778107e-04]
```

In [24]:
```
print(en.predict(x_test))
```

```
[ 2.98494975e+00  3.53444909e+00 -6.48697821e-03 ...  1.37137025e-01
  6.95848272e+00  5.87737634e-01]
```

In [25]:
```
print(en.score(x_test,y_test))
```

```
0.8629615244493962
```

5.Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
```

In [27]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

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

Out[28]:
```
(217872, 15)
```

In [29]:
```
target_vector.shape
```

Out[29]:
```
(217872,)
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
```

In [31]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [32]:
```
logr = LogisticRegression()
logr.fit(fs,target_vector)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:76
3: 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-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[32]:
```
LogisticRegression()
```

In [33]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [34]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [35]:
```
logr.classes_
```

Out[35]:
```
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 [36]:
```
logr.score(fs,target_vector)
```

Out[36]:
```
0.9029889109201733
```

6.Random Forest

In [44]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3','PM10', 'P
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df1['station']
```

In [45]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=45)
```

In [46]:
```python
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[46]:
```
RandomForestClassifier()
```

In [47]:
```python
parameters = {'max_depth':[1,2,3,4,5],
    'min_samples_leaf':[5,10,15,20,25],
    'n_estimators':[10,20,30,40,50]}
```

In [48]:
```python
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[48]:
```
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 [49]:
```python
grid_search.best_score_
```

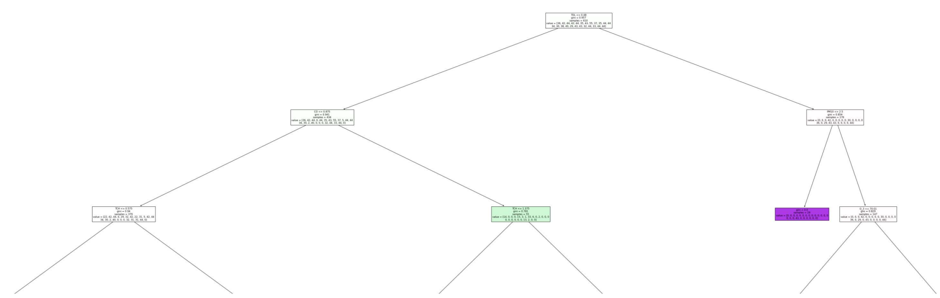
Out[49]:
```
0.5780396129926406
```

In [50]:
```python
rfc_best = grid_search.best_estimator_
```

In [51]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

```
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 5]')]
```



Results

1.linear regression : 0.8870068844960968

2.lasso regression : 0.5104330787352147

3.ridge regression : 0.8870086109906489

4.Elasticnet regression : 0.8629615244493962

5.Logistic regresssion : 0.9029889109201733

6.Random forest regression : 0.5780396129926406

    Hence Logistic regression gives high accuarcy for the madrid_2001 model.