Final Assesement 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 [3]:
```
#importing dataset
data1=pd.read_csv(r"C:\Users\user\Downloads\madrid_2002.csv")
data1
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2002-04-01 01:00:00	NaN	1.39	NaN	NaN	NaN	145.100006	352.100006	NaN	6.54	41.990002
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	20.980000
2	2002-04-01 01:00:00	NaN	0.80	NaN	NaN	NaN	103.699997	134.000000	NaN	13.01	28.440001
3	2002-04-01 01:00:00	NaN	1.61	NaN	NaN	NaN	97.599998	268.000000	NaN	5.12	42.180000
4	2002-04-01 01:00:00	NaN	1.90	NaN	NaN	NaN	92.089996	237.199997	NaN	7.28	76.330002
...	...	...	...	...	...	...	...	...	...	...	...
217291	2002-11-01 00:00:00	4.16	1.14	NaN	NaN	NaN	81.080002	265.700012	NaN	7.21	36.750000
217292	2002-11-01 00:00:00	3.67	1.73	2.89	NaN	0.38	113.900002	373.100006	NaN	5.66	63.389999
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	9.640000
217294	2002-11-01 00:00:00	4.51	0.91	4.83	10.99	NaN	149.800003	202.199997	1.00	5.75	NaN
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	29.240000

217296 rows × 16 columns

In [4]:
```
data1.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 [5]:
```
data=data1.head(50000)
```

In [6]:
```
#filling null values
df=data.fillna(0)
df
```

Out[6]:

	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	0.0	0.00	145.100006	352.100006	0.00	6.540000	41.99000
1	2002-04-01 01:00:00	1.93	0.71	2.33	6.2	0.15	98.150002	153.399994	2.67	6.850000	20.98000
2	2002-04-01 01:00:00	0.00	0.80	0.00	0.0	0.00	103.699997	134.000000	0.00	13.010000	28.44000
3	2002-04-01 01:00:00	0.00	1.61	0.00	0.0	0.00	97.599998	268.000000	0.00	5.120000	42.18000
4	2002-04-01 01:00:00	0.00	1.90	0.00	0.0	0.00	92.089996	237.199997	0.00	7.280000	76.33000
...	...	...	...	...	...	...	...	...	...	...	..
49995	2002-07-24 13:00:00	0.00	0.36	0.00	0.0	0.00	90.639999	113.900002	0.00	43.169998	35.59999
49996	2002-07-24 13:00:00	0.00	0.34	0.00	0.0	0.18	100.000000	131.699997	0.00	42.020000	54.15000
49997	2002-07-24 13:00:00	0.00	0.61	0.00	0.0	0.00	105.000000	155.100006	0.00	32.860001	45.13000
49998	2002-07-24 13:00:00	0.00	0.32	0.00	0.0	0.00	43.130001	64.269997	0.00	42.849998	20.24000
49999	2002-07-24 13:00:00	0.00	0.47	0.00	0.0	0.00	81.580002	140.600006	0.00	33.439999	39.70999

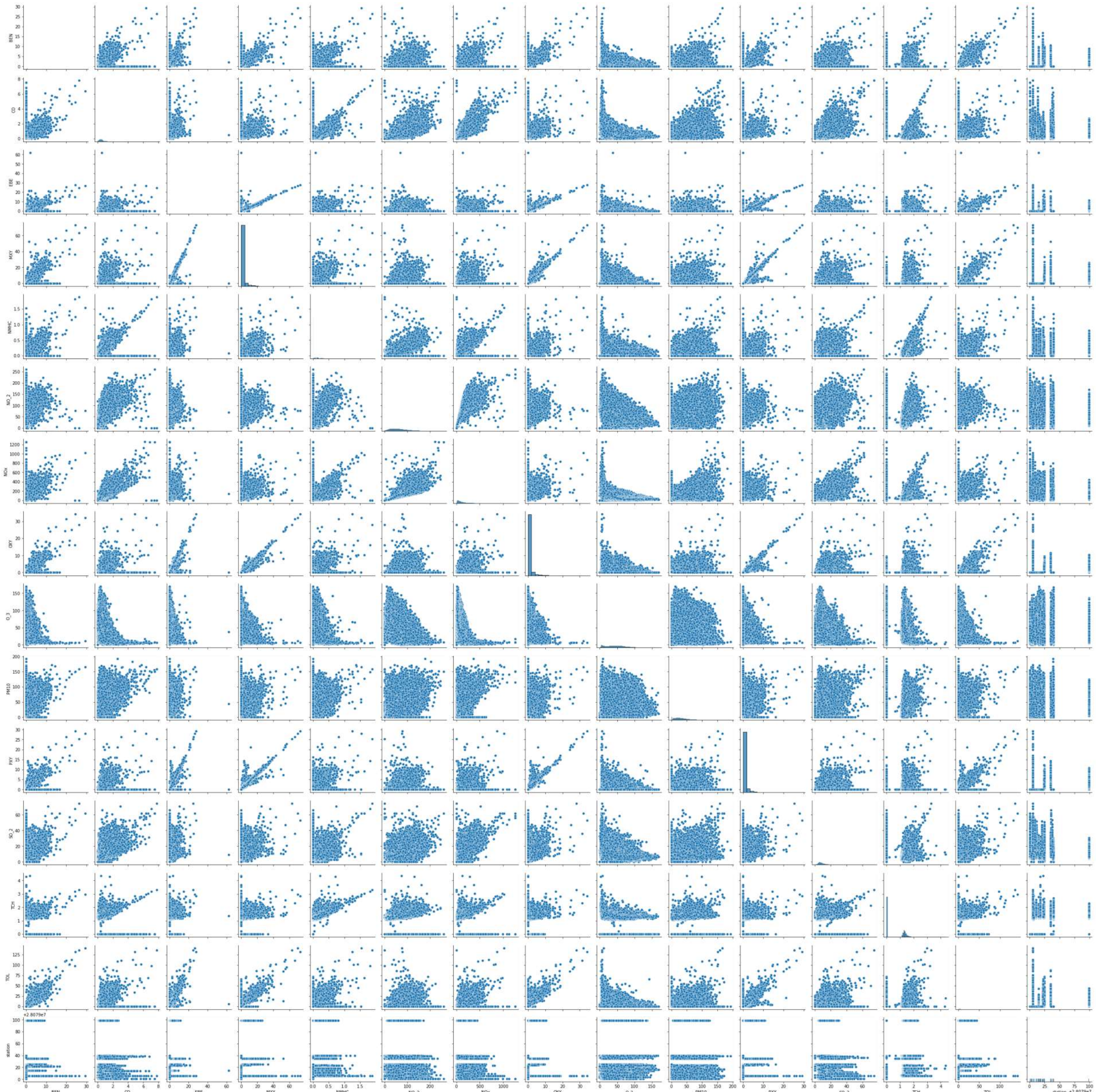
50000 rows × 16 columns

In [7]:
```
df.columns
```

Out[7]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [8]:
```
sns.pairplot(df)
```

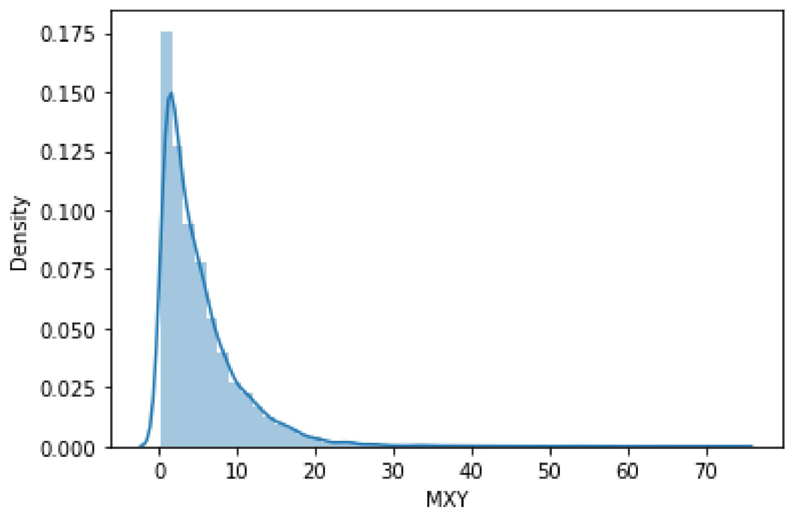
Out[8]: <seaborn.axisgrid.PairGrid at 0x20eb1f67e80>



In [9]:
```
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[9]:
```
<AxesSubplot:xlabel='MXY', ylabel='Density'>
```



MODEL BUILDING

1.Linear Regression

In [10]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [11]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df1[['MXY']]
```

In [12]:
```
#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 [13]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

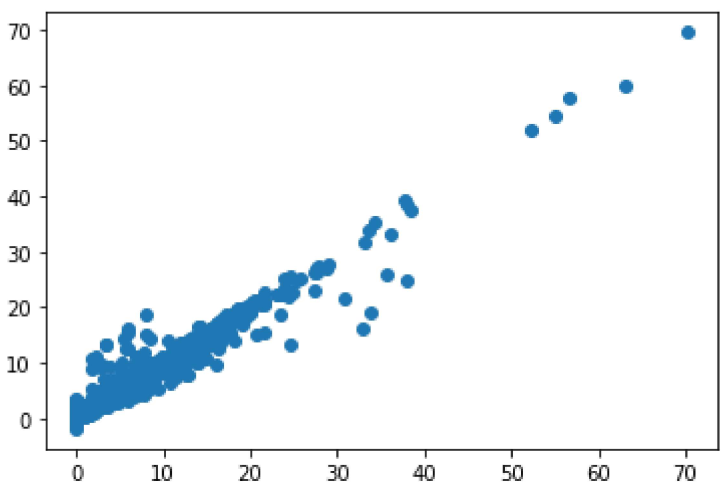
Out[13]:
```
LinearRegression()
```

In [14]:
```
print(lr.intercept_)
```

```
[-0.09181589]
```

In [15]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x20ed37edca0>
```



In [16]:
```
print(lr.score(x_test,y_test))
```

```
0.9723403212108328
```

2.Ridge Regression

In [17]:
```
from sklearn.linear_model import Ridge
```

In [18]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[18]:
```
Ridge(alpha=10)
```

In [19]:
```
rr.score(x_test,y_test)
```

Out[19]:
```
0.9723443614233309
```

3.Lasso Regression

In [20]:
```
from sklearn.linear_model import Lasso
```

In [21]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[21]:
```
Lasso(alpha=10)
```

In [22]:
```
la.score(x_test,y_test)
```

Out[22]:
```
0.45834434643546607
```

4.ElasticNet Regression

In [23]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[23]:
```
ElasticNet()
```

In [24]:
```
print(en.coef_)
```

```
[ 0.          0.          0.          0.         -0.          0.
  0.51356962 -0.00072824  0.36800219  0.00656326  0.          0.2431628
 -0.        ]
```

In [25]:
```
print(en.predict(x_test))
```

```
[-0.02362556  4.86771722 -0.04724334 ... -0.06656103 -0.04094948
  1.02838864]
```

In [26]:
```
print(en.score(x_test,y_test))
```

```
0.8974690482213497
```

5.Logistic Regression

In [27]:
```
from sklearn.linear_model import LogisticRegression
```

In [28]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

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

Out[29]:
```
(50000, 15)
```

In [30]:
```
target_vector.shape
```

Out[30]:
```
(50000,)
```

In [31]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [33]:
```
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[33]:
```
LogisticRegression()
```

In [34]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [35]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [36]:
```
logr.classes_
```

Out[36]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079009,
       28079011, 28079012, 28079014, 28079015, 28079016, 28079017,
       28079018, 28079019, 28079021, 28079022, 28079023, 28079024,
       28079025, 28079035, 28079036, 28079038, 28079039, 28079040,
       28079099], dtype=int64)
```

In [37]:
```
logr.score(fs,target_vector)
```

Out[37]:
```
0.91776
```

6.Random Forest

In [38]:
```
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=df['station']
```

In [39]:
```
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 [40]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[40]:
```
RandomForestClassifier()
```

In [41]:
```
parameters = {'max_depth':[1,2,3,4,5],
    'min_samples_leaf':[5,10,15,20,25],
    'n_estimators':[10,20,30,40,50]}
```

In [42]:
```
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[42]:
```
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 [43]:
```
grid_search.best_score_
```

Out[43]:
```
0.5368232977237609
```

In [44]:
```
rfc_best = grid_search.best_estimator_
```

In [45]:
```
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, 1168, 0]'),
 Text(465.0, 181.19999999999982, 'gini = 0.363\nsamples = 383\nvalue = [0, 0,
0, 0, 440, 0, 3, 0, 0, 4, 0, 0, 22, 0\n0, 0, 0, 0, 0, 0, 0, 0, 96, 0]'),
 Text(651.0, 181.19999999999982, 'gini = 0.696\nsamples = 1964\nvalue = [0,
0, 0, 0, 426, 0, 296, 0, 0, 28, 0, 0, 1266\n0, 0, 0, 0, 69, 0, 0, 0, 0, 0, 10
72, 0]'),
 Text(1116.0, 906.0, 'TCH <= 0.325\ngini = 0.933\nsamples = 15118\nvalue = [1
929, 1415, 1019, 0, 1151, 1902, 1684, 1932, 1914\n57, 1382, 392, 690, 1178, 1
999, 45, 0, 0, 8, 6\n1934, 1901, 554, 820, 16]'),
 Text(930.0, 543.5999999999999, 'PM10 <= 23.395\ngini = 0.914\nsamples = 1237
2\nvalue = [1929, 1415, 1019, 0, 9, 1902, 2, 1932, 1914, 32\n1382, 392, 0, 11
78, 1999, 45, 0, 0, 8, 6, 1934\n1901, 554, 10, 7]'),
 Text(837.0, 181.19999999999982, 'gini = 0.903\nsamples = 3730\nvalue = [443,
349, 325, 0, 2, 508, 0, 648, 439, 28, 451\n24, 0, 294, 884, 14, 0, 0, 8, 0, 7
58, 599, 76\n0, 0]'),
 Text(1023.0, 181.19999999999982, 'gini = 0.915\nsamples = 8642\nvalue = [148
6, 1066, 694, 0, 7, 1394, 2, 1284, 1475, 4, 931\n368, 0, 884, 1115, 31, 0, 0,
0, 6, 1176, 1302\n478, 10, 7]'),
 Text(1302.0, 543.5999999999999, 'CO <= 0.685\ngini = 0.723\nsamples = 2746\n
```

Results

1.linear regression :0.9723403212108328

2.ridge regression : 0.9723443614233309

3.lasso regression : 0.45834434643546607

4.Elasticnet regression : 0.8629615244493962

5.Logistic regresssion : 0.91776

6.Random forest regression : 0.5368232977237609

Hence Ridge regression gives high accuracy this model.

In [ ]: