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
data1=pd.read_csv(r"C:\Users\user\Downloads\madrid_2010.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2010-03-01 01:00:00	NaN	0.29	NaN	NaN	NaN	25.090000	29.219999	NaN	68.930000	Na
1	2010-03-01 01:00:00	NaN	0.27	NaN	NaN	NaN	24.879999	30.040001	NaN	NaN	Na
2	2010-03-01 01:00:00	NaN	0.28	NaN	NaN	NaN	17.410000	20.540001	NaN	72.120003	Na
3	2010-03-01 01:00:00	0.38	0.24	1.74	NaN	0.05	15.610000	21.080000	NaN	72.970001	19.4100
4	2010-03-01 01:00:00	0.79	NaN	1.32	NaN	NaN	21.430000	26.070000	NaN	NaN	24.6700
...	...	...	...	...	...	...	...	...	...	...	
209443	2010-08-01 00:00:00	NaN	0.55	NaN	NaN	NaN	125.000000	219.899994	NaN	25.379999	Na
209444	2010-08-01 00:00:00	NaN	0.27	NaN	NaN	NaN	45.709999	47.410000	NaN	NaN	51.2599
209445	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	0.24	46.560001	49.040001	NaN	46.250000	Na
209446	2010-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	46.770000	50.119999	NaN	77.709999	Na
209447	2010-08-01 00:00:00	0.92	0.43	0.71	NaN	0.25	76.330002	88.190002	NaN	52.259998	47.1500

209448 rows × 17 columns

In [3]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209448 entries, 0 to 209447
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209448 non-null  object 
 1   BEN      60268 non-null   float64
 2   CO       94982 non-null   float64
 3   EBE      60253 non-null   float64
 4   MXY      6750 non-null    float64
 5   NMHC     51727 non-null   float64
 6   NO_2     208219 non-null  float64
 7   NOx      208210 non-null  float64
 8   OXY      6750 non-null    float64
 9   O_3      126684 non-null  float64
 10  PM10     106186 non-null  float64
 11  PM25     55514 non-null   float64
 12  PXY      6740 non-null    float64
 13  SO_2     93184 non-null   float64
 14  TCH      51730 non-null   float64
 15  TOL      60171 non-null   float64
 16  station  209448 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 27.2+ MB
```

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

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

Out[5]:

N	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PM25	PXY	SO_
00	0.29	0.00	0.0	0.00	25.090000	29.219999	0.0	68.930000	0.000000	0.000000	0.0	10.1
00	0.27	0.00	0.0	0.00	24.879999	30.040001	0.0	0.000000	0.000000	0.000000	0.0	12.2
00	0.28	0.00	0.0	0.00	17.410000	20.540001	0.0	72.120003	0.000000	0.000000	0.0	0.0
38	0.24	1.74	0.0	0.05	15.610000	21.080000	0.0	72.970001	19.410000	7.870000	0.0	10.0
79	0.00	1.32	0.0	0.00	21.430000	26.070000	0.0	0.000000	24.670000	22.030001	0.0	10.6
...	...	...	...	...	...	...	...	...	...	...	...	.
51	0.23	1.02	0.0	0.12	32.910000	38.000000	0.0	57.400002	24.389999	13.210000	0.0	5.5
20	0.00	0.23	0.0	0.00	21.629999	25.700001	0.0	0.000000	17.719999	10.100000	0.0	8.1
3	0.00	0.25	0.0	0.00	17.030001	21.040001	0.0	0.000000	0.000000	0.000000	0.0	0.0
00	0.00	0.00	0.0	0.00	28.639999	30.980000	0.0	0.000000	20.400000	0.000000	0.0	6.5
00	0.22	0.00	0.0	0.00	55.360001	63.799999	0.0	54.169998	0.000000	0.000000	0.0	0.0

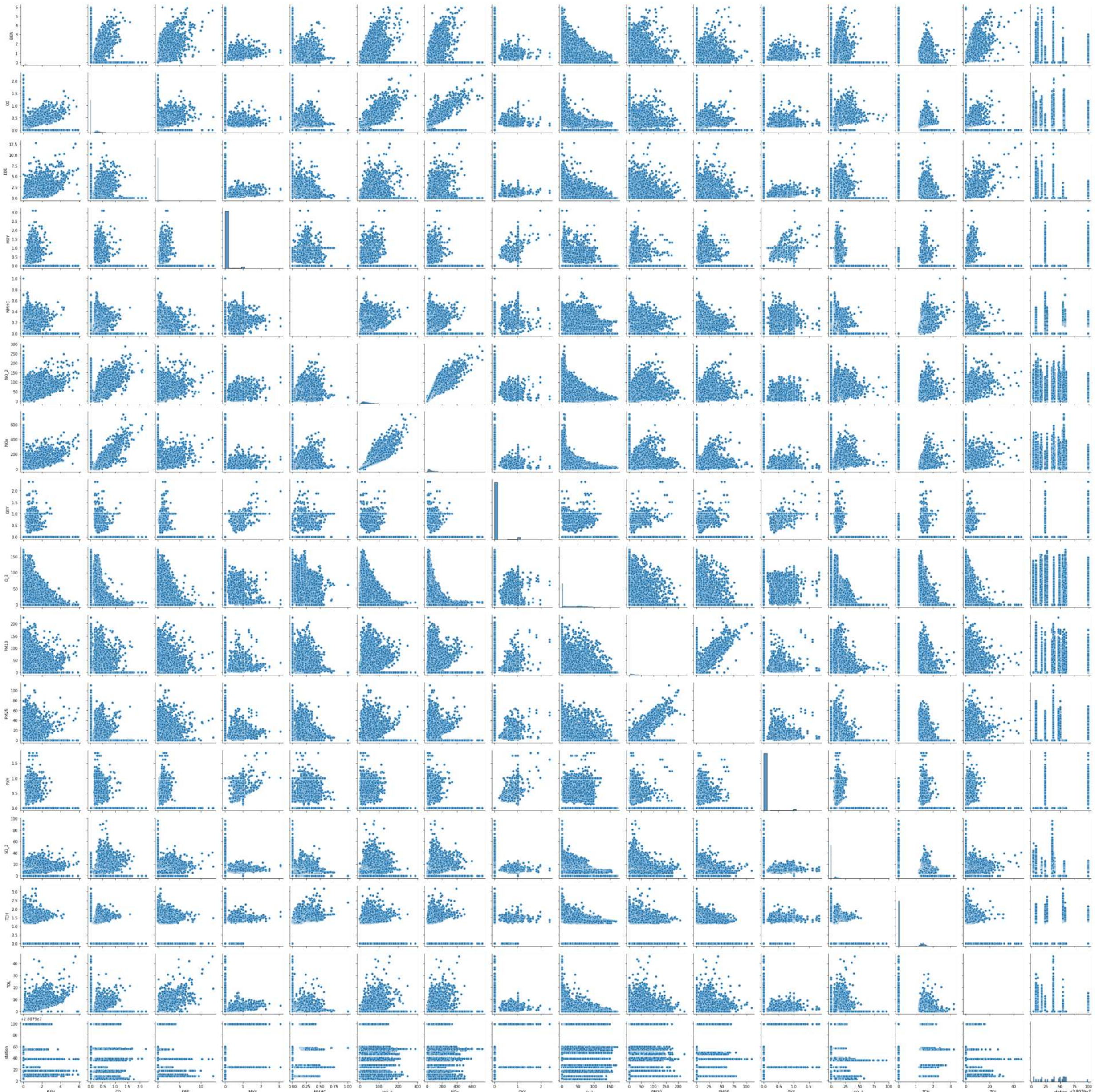
umns

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

Out[6]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

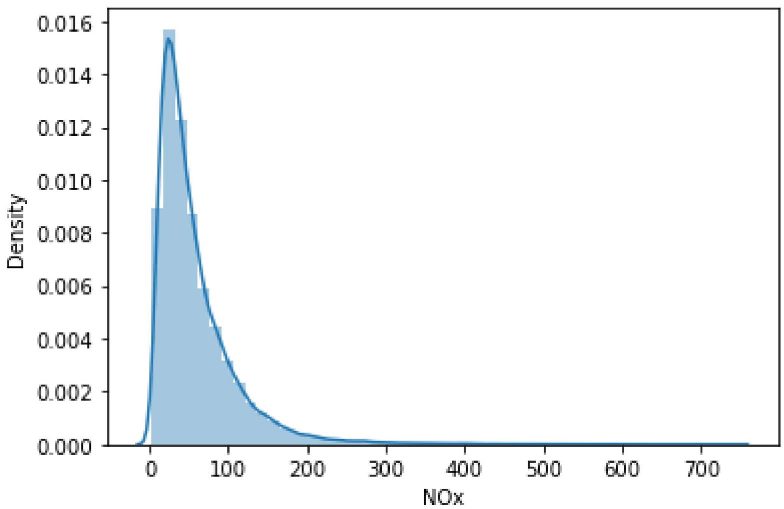
Out[7]: <seaborn.axisgrid.PairGrid at 0x1b1f57f78b0>



In [9]:
```
sns.distplot(data["NOx"])
```

```
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='NOx', ylabel='Density'>
```



MODEL BUILDING

1.Linear Regression

In [14]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [15]:
```
x=df1[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'OXY', 'O_3','PM10', 'PXY', 'SO
y=df1[['NOx']]
```

In [16]:
```
#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 [17]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

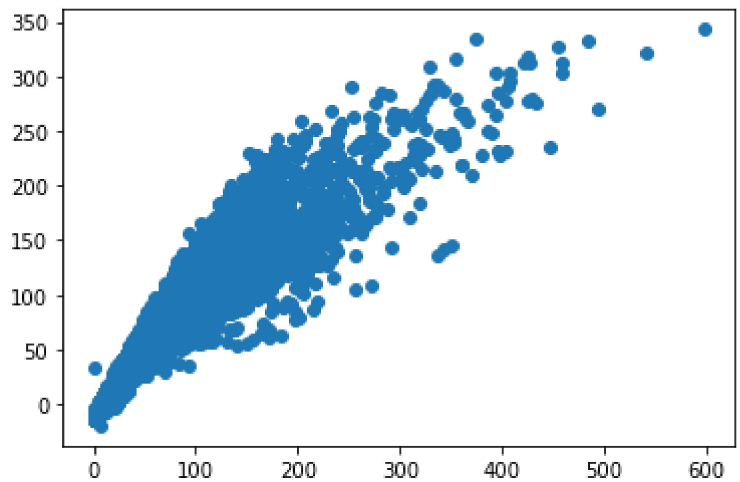
Out[17]:
```
LinearRegression()
```

In [18]:
```
print(lr.intercept_)
```

```
[-1365705.55708257]
```

In [19]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1b19d063910>
```



In [20]:
```
print(lr.score(x_test,y_test))
```

```
0.8692089018915892
```

2.Ridge Regression

In [21]:
```
from sklearn.linear_model import Ridge
```

In [22]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[22]:
```
Ridge(alpha=10)
```

In [23]:
```
rr.score(x_test,y_test)
```

Out[23]:
```
0.8691959334056143
```

3.Lasso Regression

In [24]:
```
from sklearn.linear_model import Lasso
```

In [25]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[25]:
```
Lasso(alpha=10)
```

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

Out[26]:
```
0.8604972643160442
```

4.ElasticNet Regression

In [27]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[27]:
```
ElasticNet()
```

In [28]:
```
print(en.coef_)
```

```
[ 0.00000000e+00  4.78422177e-02  2.22102267e-01 -0.00000000e+00
  0.00000000e+00  1.70494634e+00 -0.00000000e+00  8.61202032e-03
 -4.04713928e-02 -0.00000000e+00  5.35160842e-01  0.00000000e+00
  1.58818980e-01  1.57322777e-03]
```

In [29]:
```
print(en.predict(x_test))
```

```
[ 42.53741104 124.67896895 -13.2848226  ...  37.29012965  20.50131458
  99.52221428]
```

In [30]:
```
print(en.score(x_test,y_test))
```

```
0.8619496362026571
```

5.Logistic Regression

In [31]:
```
from sklearn.linear_model import LogisticRegression
```

In [32]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

In [33]:
```
feature_matrix.shape
```

Out[33]:
```
(50000, 15)
```

In [34]:
```
target_vector.shape
```

Out[34]: (50000,)

In [35]:
```
from sklearn.preprocessing import StandardScaler
```

In [36]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [37]:
```
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[37]: LogisticRegression()

In [38]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [39]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [40]:
```
logr.classes_
```

Out[40]:
```
array([28079003, 28079004, 28079008, 28079011, 28079016, 28079017,
       28079018, 28079024, 28079027, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060,
       28079099], dtype=int64)
```

In [41]:
```
logr.score(fs,target_vector)
```

Out[41]: 0.98068

6.Random Forest

In [56]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3','PM10', 'P
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'OXY','PM10', 'PXY', 'SO_2', 'TCH', 'T
y=df['station']
```

In [57]:
```
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 [58]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[58]:
```
RandomForestClassifier()
```

In [59]:
```
parameters = {'max_depth':[1,2,3,4,5],
    'min_samples_leaf':[5,10,15,20,25],
    'n_estimators':[10,20,30,40,50]}
```

In [60]:
```
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[60]:
```
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 [61]:
```
grid_search.best_score_
```

Out[61]:
```
0.9550593720536289
```

In [62]:
```
rfc_best = grid_search.best_estimator_
```

In [63]:
```
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]'),
 Text(3048.5853658536585, 181.19999999999982, 'gini = 0.0\nsamples = 1289\nva
lue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 2055, 0, 0, 0,
0, 0, 0]'),
 Text(3375.2195121951218, 543.5999999999999, 'station <= 28079013.0\ngini =
0.698\nsamples = 3494\nvalue = [0, 0, 2042, 0, 0, 0, 1997, 685, 0, 0, 0, 0, 0
\n0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 833]'),
 Text(3266.341463414634, 181.19999999999982, 'gini = 0.0\nsamples = 1285\nval
ue = [0, 0, 2042, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0,
0, 0]'),
 Text(3484.0975609756097, 181.19999999999982, 'gini = 0.583\nsamples = 2209\n
value = [0, 0, 0, 0, 0, 0, 1997, 685, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 833]'),
 Text(4028.487804878049, 906.0, 'SO_2 <= 9.525\ngini = 0.5\nsamples = 1671\nv
alue = [0, 0, 0, 0, 0, 0, 0, 1352, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0,
0, 0, 1294]'),
 Text(3810.731707317073, 543.5999999999999, 'TCH <= 1.405\ngini = 0.267\nsamp
les = 765\nvalue = [0, 0, 0, 0, 0, 0, 0, 1036, 0, 0, 0, 0, 0, 0\n0, 0, 0,
0, 0, 0, 0, 0, 0, 195]'),
 Text(3701.8536585365855, 181.19999999999982, 'gini = 0.113\nsamples = 450\nv
```

Results

```
1.Linear regression : 0.8692089018915892

2.Ridge regression : 0.8691959334056143

3.Lasso regression : 0.8604972643160442

4.Elasticnet regression : 0.8619496362026571

5.Logistic regresssion : 0.98068

6.Random forest regression : 0.9550593720536289

Hence Logistic regression gives high accuracy for the madrid_2010 model.
```

In [ ]: