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_2016.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2016-11-01 01:00:00	NaN	0.7	NaN	NaN	153.0	77.0	NaN	NaN	NaN	7.0	NaN	NaN	28
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	28
2	2016-11-01 01:00:00	5.9	NaN	7.5	NaN	297.0	139.0	NaN	NaN	NaN	NaN	NaN	26.0	28
3	2016-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	113.0	2.0	NaN	NaN	NaN	NaN	NaN	28
4	2016-11-01 01:00:00	NaN	NaN	NaN	NaN	275.0	127.0	2.0	NaN	NaN	18.0	NaN	NaN	28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
209491	2016-07-01 00:00:00	NaN	0.2	NaN	NaN	2.0	29.0	73.0	NaN	NaN	NaN	NaN	NaN	28
209492	2016-07-01 00:00:00	NaN	0.3	NaN	NaN	1.0	29.0	NaN	36.0	NaN	5.0	NaN	NaN	28
209493	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	1.0	19.0	71.0	NaN	NaN	NaN	NaN	NaN	28
209494	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	6.0	17.0	85.0	NaN	NaN	NaN	NaN	NaN	28
209495	2016-07-01 00:00:00	NaN	NaN	NaN	NaN	2.0	46.0	61.0	34.0	NaN	NaN	NaN	NaN	28

209496 rows × 14 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209496 entries, 0 to 209495
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209496 non-null  object 
 1   BEN      50755 non-null   float64
 2   CO       85999 non-null   float64
 3   EBE      50335 non-null   float64
 4   NMHC     25970 non-null   float64
 5   NO       208614 non-null  float64
 6   NO_2     208614 non-null  float64
 7   O_3      121197 non-null  float64
 8   PM10     102892 non-null  float64
 9   PM25     52165 non-null   float64
 10  SO_2     86023 non-null   float64
 11  TCH      25970 non-null   float64
 12  TOL      50662 non-null   float64
 13  station  209496 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

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

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

Out[5]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	sta
0	2016-11-01 01:00:00	0.0	0.7	0.0	0.00	153.0	77.0	0.0	0.0	0.0	7.0	0.00	0.0	2807
1	2016-11-01 01:00:00	3.1	1.1	2.0	0.53	260.0	144.0	4.0	46.0	24.0	18.0	2.44	14.4	2807
2	2016-11-01 01:00:00	5.9	0.0	7.5	0.00	297.0	139.0	0.0	0.0	0.0	0.0	0.00	26.0	2807
3	2016-11-01 01:00:00	0.0	1.0	0.0	0.00	154.0	113.0	2.0	0.0	0.0	0.0	0.00	0.0	2807
4	2016-11-01 01:00:00	0.0	0.0	0.0	0.00	275.0	127.0	2.0	0.0	0.0	18.0	0.00	0.0	2807
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
49995	2016-05-27 07:00:00	0.0	0.3	0.0	0.00	32.0	68.0	6.0	0.0	0.0	0.0	0.00	0.0	2807
49996	2016-05-27 07:00:00	0.0	0.0	0.0	0.00	40.0	57.0	2.0	0.0	0.0	16.0	0.00	0.0	2807
49997	2016-05-27 07:00:00	0.3	0.4	0.2	0.00	35.0	46.0	2.0	13.0	0.0	5.0	0.00	2.4	2807
49998	2016-05-27 07:00:00	0.1	0.2	0.1	0.05	1.0	14.0	31.0	17.0	10.0	2.0	1.19	1.4	2807
49999	2016-05-27 07:00:00	0.0	0.0	0.0	0.00	24.0	50.0	5.0	0.0	0.0	0.0	0.00	0.0	2807

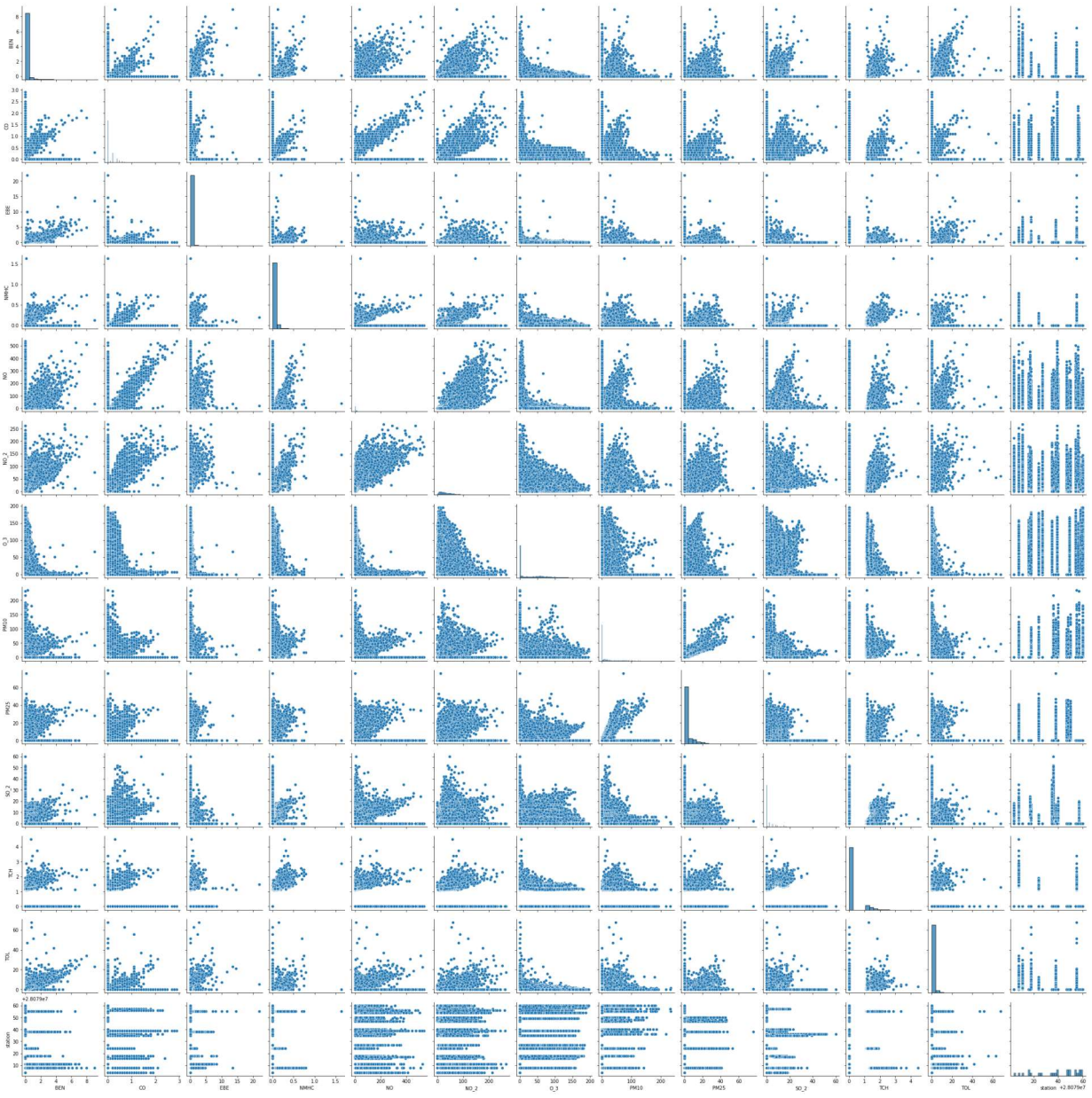
50000 rows × 14 columns

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

Out[6]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

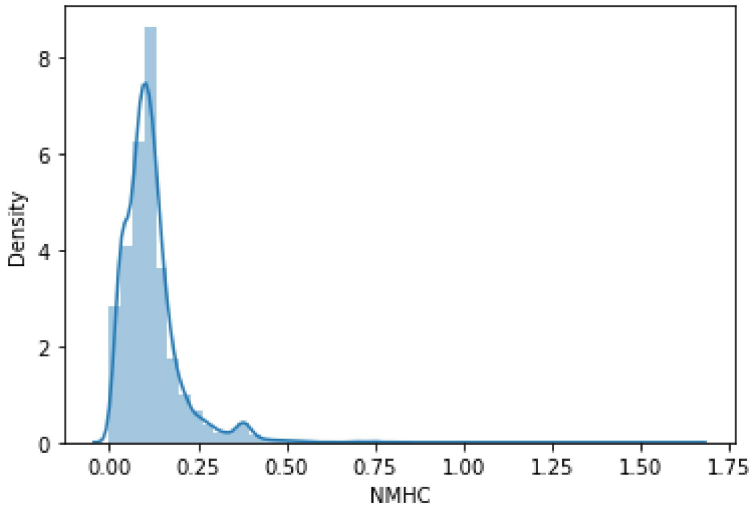
Out[7]: <seaborn.axisgrid.PairGrid at 0x1805bf94100>



In [8]:
```
sns.distplot(data['NMHC'])
```

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

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



MODEL BUILDING

1.Linear Regression

In [9]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25', 'SO_2',
```

In [10]:
```
x=df1[[ 'BEN', 'CO', 'EBE',  'NO', 'NO_2', 'O_3', 'PM10', 'PM25', 'SO_2', 'TCH',
y=df1[['NMHC']]
```

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)
```

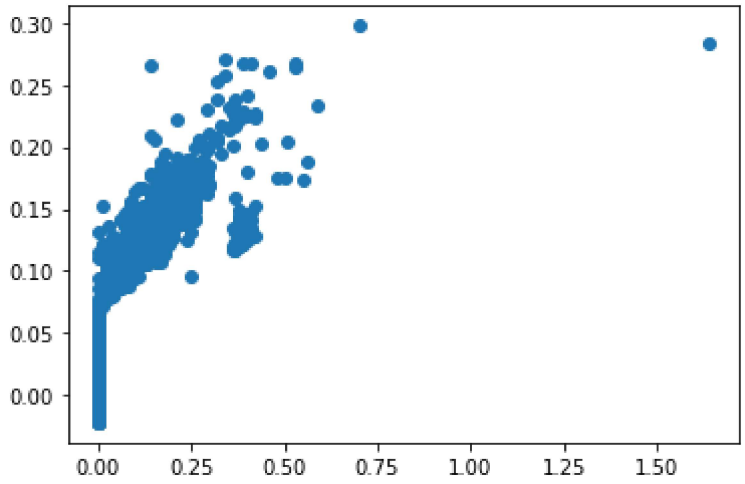
Out[12]:
```
LinearRegression()
```

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

```
[-8577.9945126]
```

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

Out[14]:
```
<matplotlib.collections.PathCollection at 0x18073636550>
```



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

```
0.7406233769938031
```

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]:
```
0.7405549029540026
```

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]:
```
-2.1867189305524448e-05
```

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.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0. -0.]
```

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

```
[0.01445943 0.01445943 0.01445943 ... 0.01445943 0.01445943 0.01445943]
```

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

```
-2.1867189305524448e-05
```

5.Logistic Regression

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

In [27]:
```
feature_matrix = df1.iloc[:,0:11]
target_vector = df1.iloc[:,-1]
```

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

Out[28]:
```
(50000, 11)
```

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

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

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

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

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

```
[28079008]
```

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

Out[35]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

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

Out[36]:
```
0.69488
```

6.Random Forest

In [37]:
```
', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25','SO_2', 'TCH', 'TOL', 'station']]
, 'PM10','SO_2', 'TCH', 'TOL']]
```

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=0.3)
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
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]:
```
0.6724857142857144
```

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)
```

```
 Text(2021.433962264151, 181.19999999999982, 'gini = 0.006\nsamples = 614\nva
lue = [0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0, 9
68]'),
 Text(2274.11320754717, 543.5999999999999, 'CO <= 0.1\ngini = 0.847\nsamples
= 2170\nvalue = [0, 2, 0, 375, 0, 0, 0, 320, 10, 0, 0, 540, 0\n0, 0, 294, 0,
364, 0, 972, 0, 90, 147, 394]'),
 Text(2189.8867924528304, 181.19999999999982, 'gini = 0.805\nsamples = 1017\n
value = [0, 0, 0, 0, 0, 0, 0, 320, 0, 0, 0, 0, 0, 0\n0, 294, 0, 364, 0, 2, 0,
90, 147, 394]'),
 Text(2358.33962264151, 181.19999999999982, 'gini = 0.618\nsamples = 1153\nva
lue = [0, 2, 0, 375, 0, 0, 0, 0, 10, 0, 0, 540, 0\n0, 0, 0, 0, 0, 0, 970, 0,
0, 0, 0]'),
 Text(3326.943396226415, 1630.8000000000002, 'O_3 <= 0.5\ngini = 0.9\nsamples
= 9125\nvalue = [1542, 1445, 0, 0, 1406, 1322, 1387, 0, 1425, 1507\n1447, 0,
1423, 0, 0, 0, 0, 0, 0, 0, 1453, 0, 0\n0]'),
 Text(2863.6981132075475, 1268.4, 'PM10 <= 0.5\ngini = 0.801\nsamples = 4686
\nvalue = [1542, 1, 0, 0, 8, 5, 3, 0, 0, 1507, 1447, 0\n1423, 0, 0, 0, 0, 0,
0, 0, 1453, 0, 0, 0]'),
 Text(2779.471698113208, 906.0, 'SO_2 <= 5.5\ngini = 0.072\nsamples = 1010\nv
alue = [1542, 0, 0, 0, 8, 0, 0, 0, 0, 1, 34, 0, 10\n0, 0, 0, 0, 0, 0, 6,
```

Results

1.Linear regression : 0.7406233769938031

2.Ridge regression : 0.7405549029540026

3.Lasso regression : -2.1867189305524448e-05

4.Elasticnet regression : -2.1867189305524448e-05

5.Logistic regresssion : 0.69488

6.Random forest regression : 0.6724857142857144

Hence Linear regression gives high accuracy for the madrid_2013 model.

In [ ]: