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_2017.csv")
data1
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH
0	2017-06-01 01:00:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	NaN
1	2017-06-01 01:00:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1.4
2	2017-06-01 01:00:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN
3	2017-06-01 01:00:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	NaN
4	2017-06-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
210115	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	NaN
210116	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	NaN
210117	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	NaN
210118	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	NaN
210119	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	NaN

210120 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210120 entries, 0 to 210119
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210120 non-null  object 
 1   BEN      50201 non-null   float64
 2   CH4      6410 non-null    float64
 3   CO       87001 non-null   float64
 4   EBE      49973 non-null   float64
 5   NMHC     25472 non-null   float64
 6   NO       209065 non-null  float64
 7   NO_2     209065 non-null  float64
 8   NOx      52818 non-null   float64
 9   O_3      121398 non-null  float64
 10  PM10     104141 non-null  float64
 11  PM25     52023 non-null   float64
 12  SO_2     86803 non-null   float64
 13  TCH      25472 non-null   float64
 14  TOL      50117 non-null   float64
 15  station  210120 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 25.6+ MB
```

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

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

Out[5]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH	T
0	2017-06-01 01:00:00	0.0	0.0	0.3	0.0	0.00	4.0	38.0	0.0	0.0	0.0	0.0	5.0	0.00	
1	2017-06-01 01:00:00	0.6	0.0	0.3	0.4	0.08	3.0	39.0	0.0	71.0	22.0	9.0	7.0	1.40	
2	2017-06-01 01:00:00	0.2	0.0	0.0	0.1	0.00	1.0	14.0	0.0	0.0	0.0	0.0	0.0	0.00	
3	2017-06-01 01:00:00	0.0	0.0	0.2	0.0	0.00	1.0	9.0	0.0	91.0	0.0	0.0	0.0	0.00	
4	2017-06-01 01:00:00	0.0	0.0	0.0	0.0	0.00	1.0	19.0	0.0	69.0	0.0	0.0	2.0	0.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
49995	2017-04-27 23:00:00	0.0	0.0	0.2	0.0	0.00	3.0	18.0	0.0	85.0	0.0	0.0	0.0	0.00	
49996	2017-04-27 23:00:00	0.0	0.0	0.0	0.0	0.00	1.0	22.0	0.0	84.0	0.0	0.0	2.0	0.00	
49997	2017-04-27 23:00:00	0.2	0.0	0.5	0.1	0.00	2.0	24.0	0.0	85.0	14.0	0.0	8.0	0.00	
49998	2017-04-27 23:00:00	0.2	0.0	0.2	0.1	0.11	1.0	15.0	0.0	91.0	7.0	4.0	3.0	1.17	
49999	2017-04-27 23:00:00	0.0	0.0	0.0	0.0	0.00	1.0	12.0	0.0	89.0	0.0	0.0	0.0	0.00	

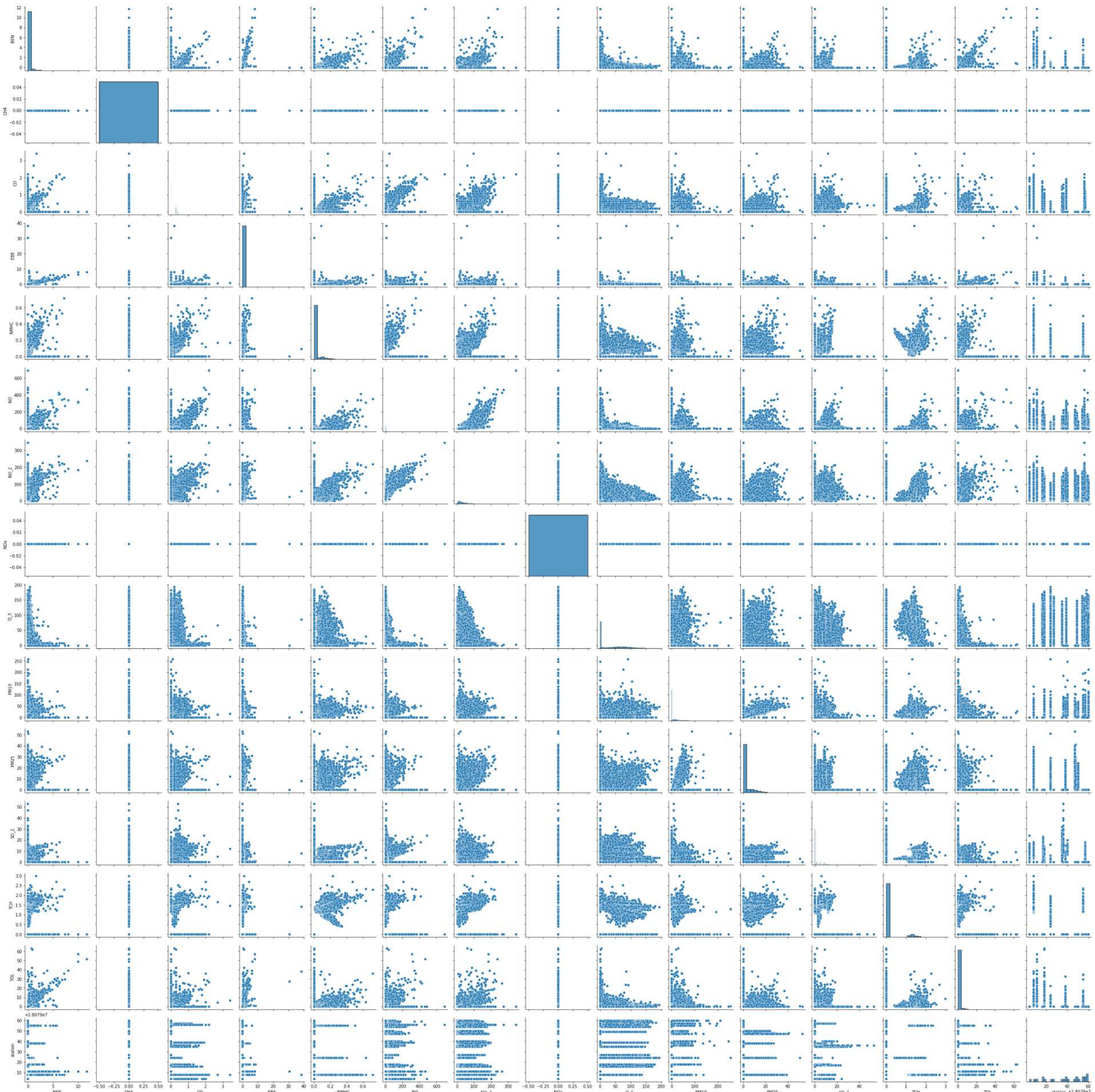
50000 rows × 16 columns

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

Out[6]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

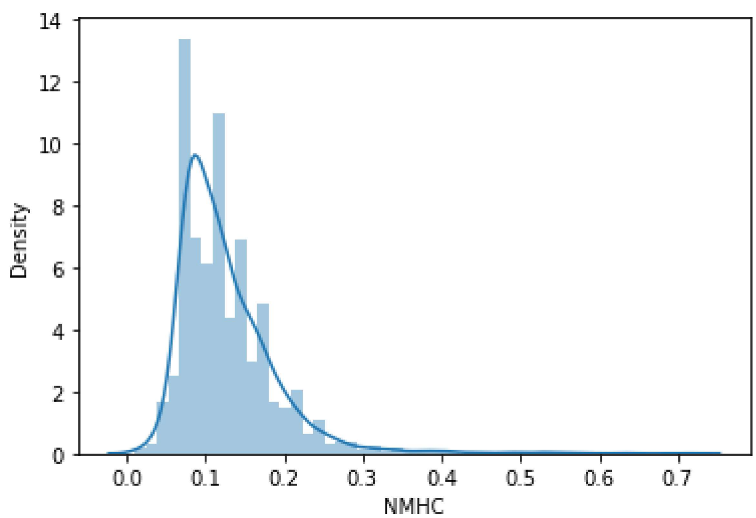
Out[7]: <seaborn.axisgrid.PairGrid at 0x1ab93c53250>



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

```
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='NMHC', ylabel='Density'>



MODEL BUILDING

1.Linear Regression

In [9]:
```
df1=df[['BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [10]:
```
x=df1[['BEN', 'CH4', 'CO', 'EBE','NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station']]
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)
```

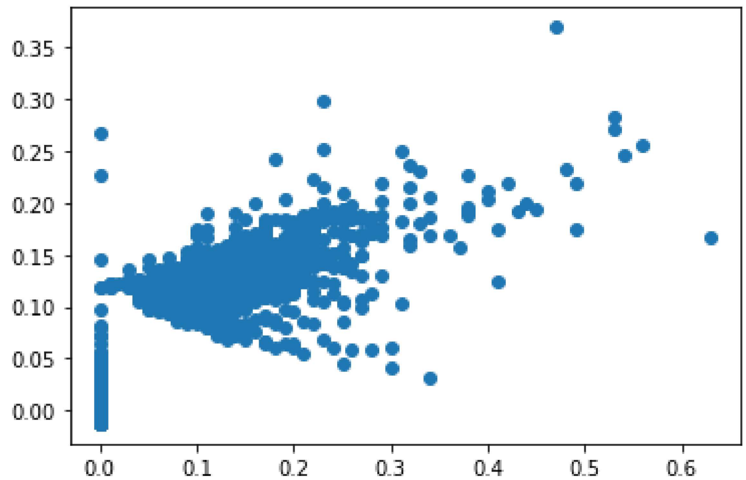
Out[12]:
```
LinearRegression()
```

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

```
[-5011.21826019]
```

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

Out[14]:
```
<matplotlib.collections.PathCollection at 0x1abb13d5730>
```



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

```
0.8256634613132217
```

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]: 0.8256361719111386

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]: -0.000182872072800766

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.  0. -0.]
```

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

```
[0.01478057 0.01478057 0.01478057 ... 0.01478057 0.01478057 0.01478057]
```

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

```
-0.000182872072800766
```

5.Logistic Regression

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

In [27]:
```
feature_matrix = df1.iloc[:,0:15]
target_vector = df1.iloc[:,-1]
```

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

Out[28]:
```
(50000, 15)
```

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: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 [35]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [36]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079059]
```

In [37]:
```
logr.classes_
```

Out[37]:
```
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 [38]:
```
logr.score(fs,target_vector)
```

Out[38]:
```
0.95514
```

6.Random Forest

In [39]:
```
df1=df[['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25', '
x=df1[['CO','NMHC', 'NO_2', 'O_3', 'PM10','SO_2', 'TCH', 'TOL']]
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=0.3)
```

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
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]:
```
0.6889428571428571
```

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)
```

Out[46]:
```
[Text(1953.0, 1993.2, 'CO <= 0.05\ngini = 0.958\nsamples = 22197\nvalue = [14
10, 1464, 1446, 1491, 1482, 1472, 1446, 1510, 1397\n1457, 1409, 1402, 1427, 1
441, 1485, 1484, 1413, 1444\n1531, 1441, 1445, 1453, 1563, 1487]'),
 Text(870.48, 1630.8000000000002, 'PM10 <= 0.5\ngini = 0.929\nsamples = 13171
\nvalue = [2, 19, 1446, 1, 1482, 2, 28, 1510, 20, 3, 1409\n16, 1427, 1441, 14
85, 1484, 1413, 1444, 1531, 47, 3\n1453, 1563, 1487]'),
 Text(446.4, 1268.4, 'TOL <= 0.05\ngini = 0.863\nsamples = 6748\nvalue = [2,
17, 1446, 1, 1482, 0, 24, 1510, 20, 0, 11, 16\n26, 4, 18, 1484, 16, 1444, 18,
47, 0, 1453, 1563\n7]'),
 Text(357.12, 906.0, 'SO_2 <= 0.5\ngini = 0.841\nsamples = 5817\nvalue = [2,
17, 11, 1, 1482, 0, 24, 1510, 20, 0, 0, 16\n26, 4, 18, 1484, 16, 1444, 6, 47,
0, 1453, 1563\n7]'),
 Text(178.56, 543.5999999999999, 'O_3 <= 0.5\ngini = 0.809\nsamples = 4855\nv
alue = [2, 17, 11, 1, 3, 0, 24, 1510, 4, 0, 0, 16, 1\n4, 18, 1484, 16, 1444,
6, 47, 0, 1453, 1563, 7]'),
 Text(89.28, 181.19999999999982, 'gini = 0.907\nsamples = 100\nvalue = [2, 1
7, 11, 1, 0, 0, 24, 3, 2, 0, 0, 16, 1, 4\n18, 1, 16, 13, 6, 14, 0, 2, 7,
0]'),
 Text(267.84000000000003, 181.19999999999982, 'gini = 0.802\nsamples = 4755\n
```

Results

```
1.Linear regression : 0.8256634613132217

2.Ridge regression :  0.8256361719111386

3.Lasso regression : -0.000182872072800766

4.Elasticnet regression : -0.000182872072800766

5.Logistic regresssion : 0.95514

6.Random forest regression : 0.6889428571428571

Hence Logistic regression gives high accuracy for the madrid_2017 model.
```