Final Assessment 1

Kaviyadevi(20106064)

In [2]:
```
#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_2007.csv")
data1
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	F
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.19
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.80
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.09
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.30
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.50
...	...	...	...	...	...	...	...	...	...	...	
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.76
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	5.70
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	13.01
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	6.61
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.26

225120 rows × 17 columns

In [4]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225120 entries, 0 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     225120 non-null  object 
 1   BEN      68885 non-null   float64
 2   CO       206748 non-null  float64
 3   EBE      68883 non-null   float64
 4   MXY      26061 non-null   float64
 5   NMHC     86883 non-null   float64
 6   NO_2     223985 non-null  float64
 7   NOx      223972 non-null  float64
 8   OXY      26062 non-null   float64
 9   O_3      211850 non-null  float64
 10  PM10     222588 non-null  float64
 11  PM25     68870 non-null   float64
 12  PXY      26062 non-null   float64
 13  SO_2     224372 non-null  float64
 14  TCH      87026 non-null   float64
 15  TOL      68845 non-null   float64
 16  station  225120 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.2+ 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	PM
0	2007-12-01 01:00:00	0.00	2.86	0.00	0.00	0.00	282.200012	1054.000000	0.00	4.030000	156.199
1	2007-12-01 01:00:00	0.00	1.82	0.00	0.00	0.00	86.419998	354.600006	0.00	3.260000	80.809
2	2007-12-01 01:00:00	0.00	1.47	0.00	0.00	0.00	94.639999	319.000000	0.00	5.310000	53.099
3	2007-12-01 01:00:00	0.00	1.64	0.00	0.00	0.00	127.900002	476.700012	0.00	4.500000	105.300
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.500
...	...	...	...	...	...	...	...	...	...	...	
49995	2007-10-20 12:00:00	0.00	0.45	0.00	0.00	0.29	54.110001	89.779999	0.00	26.580000	39.150
49996	2007-10-20 12:00:00	0.40	0.47	1.38	0.00	0.27	65.879997	98.970001	0.00	12.550000	36.340
49997	2007-10-20 12:00:00	0.00	0.96	0.00	0.00	0.00	89.959999	226.100006	0.00	7.120000	0.000
49998	2007-10-20 12:00:00	0.00	0.46	0.00	0.00	0.00	70.089996	111.400002	0.00	28.370001	34.590
49999	2007-10-20 12:00:00	0.00	0.29	0.00	0.00	0.29	53.610001	78.440002	0.00	31.700001	30.280

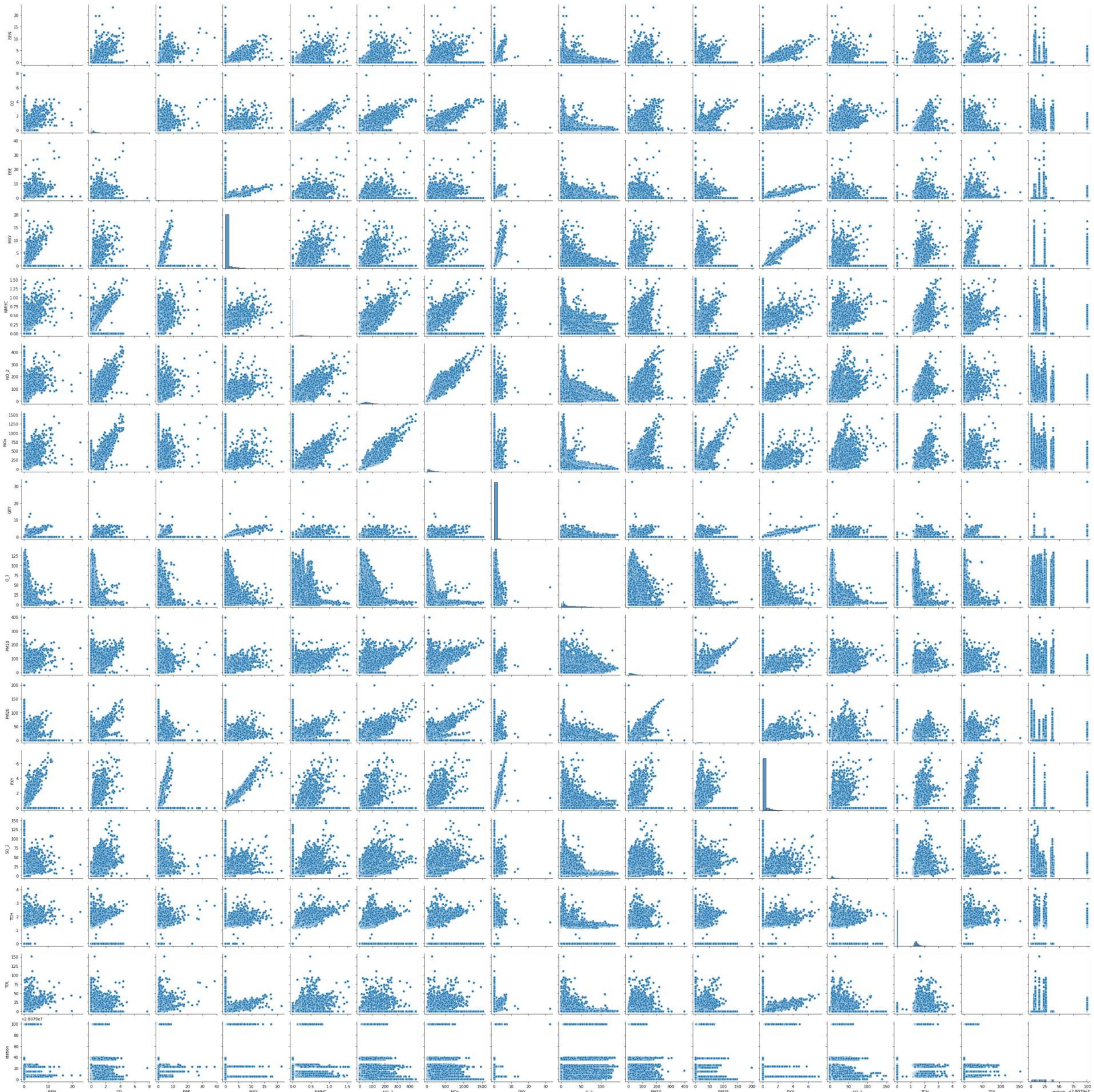
50000 rows × 17 columns

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

Out[7]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

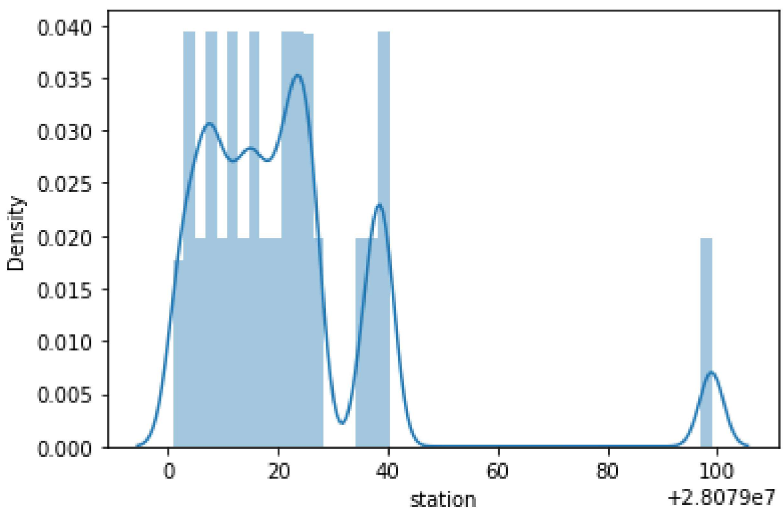
Out[8]: <seaborn.axisgrid.PairGrid at 0x1cbb96e2b80>



In [10]:
```
sns.distplot(data["station"])
```

```
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[10]: <AxesSubplot:xlabel='station', ylabel='Density'>



MODEL BUILDING

1.Linear Regression

In [11]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [12]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY', 'PXY', 'SO_2', 'TCH', 'T
y=df1[['PM10']]
```

In [13]:
```
#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 [14]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

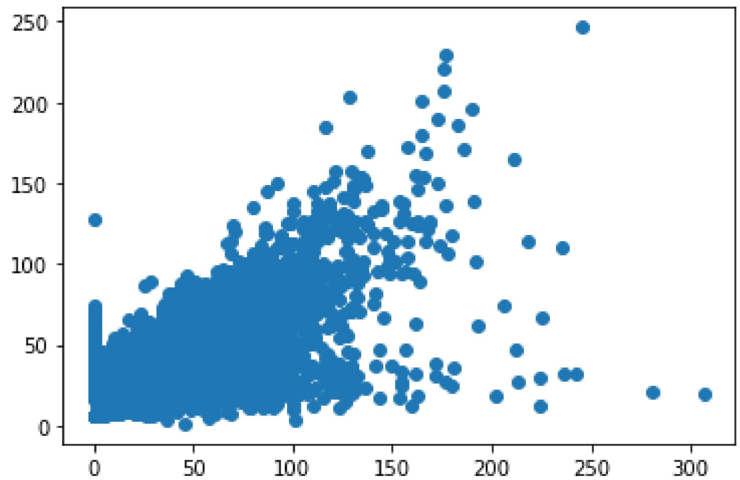
Out[14]:
```
LinearRegression()
```

In [15]:
```
print(lr.intercept_)
```

```
[6.3964739]
```

In [16]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x1cbc9e75400>
```



In [17]:
```
print(lr.score(x_test,y_test))
```

```
0.47252370822056
```

2.Ridge Regression

In [18]:
```
from sklearn.linear_model import Ridge
```

In [19]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[19]:
```
Ridge(alpha=10)
```

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

Out[20]:
```
0.47253459053957125
```

3.Lasso Regression

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

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

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

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

Out[23]:
```
0.4716378609310229
```

4.ElasticNet Regression

In [24]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[24]:
```
ElasticNet()
```

In [25]:
```
print(en.coef_)
```

```
[-0.          0.          -0.          -0.          0.11344146  0.14030236
  0.          0.          -0.05376687 -0.         -0.00297606  0.13695436]
```

In [26]:
```
print(en.predict(x_test))
```

```
[43.92928729 28.90939916 28.13026041 ... 24.8886037  27.5010261
 40.18690079]
```

In [27]:
```
print(en.score(x_test,y_test))
```

```
0.4721303407512927
```

5.Logistic Regression

In [28]:
```
from sklearn.linear_model import LogisticRegression
```

In [29]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

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

Out[30]:
```
(50000, 15)
```

In [31]:
```
target_vector.shape
```

Out[31]:
```
(50000,)
```

In [32]:
```
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [34]:
```
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[34]:
```
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)
```

```
[28079099]
```

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

Out[37]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079018, 28079019, 28079021, 28079022, 28079023, 28079024,
       28079025, 28079026, 28079027, 28079036, 28079038, 28079039,
       28079040, 28079099], dtype=int64)
```

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

Out[38]:
```
0.83154
```

6.Random Forest

In [39]:
```
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 [40]:
```
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 [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]:
```
0.47662896145180605
```

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)
```

```
0, 0]'),
 Text(515.0769230769231, 543.5999999999999, 'PM10 <= 15.135\ngini = 0.471\nsamples = 585\nvalue = [0, 0, 0, 0, 0, 0, 113, 3, 0, 0, 0, 0, 34, 0\n34, 0, 0, 0, 13, 0, 0, 67, 0, 0, 650, 0]'),
 Text(429.23076923076917, 181.19999999999982, 'gini = 0.383\nsamples = 278\nvalue = [0, 0, 0, 0, 0, 0, 26, 0, 0, 0, 0, 0, 28, 0\n20, 0, 0, 0, 0, 0, 0, 24, 0, 0, 342, 0]'),
 Text(600.9230769230769, 181.19999999999982, 'gini = 0.534\nsamples = 307\nvalue = [0, 0, 0, 0, 0, 0, 87, 3, 0, 0, 0, 0, 6, 0\n14, 0, 0, 0, 13, 0, 0, 43, 0, 0, 308, 0]'),
 Text(1030.1538461538462, 906.0, 'PM10 <= 14.83\ngini = 0.551\nsamples = 569\nvalue = [10, 11, 1, 0, 1, 0, 564, 2, 0, 2, 0, 2, 1, 1\n3, 0, 0, 0, 81, 0, 0, 11, 3, 0, 217, 0]'),
 Text(858.4615384615383, 543.5999999999999, 'NOx <= 81.55\ngini = 0.561\nsamples = 60\nvalue = [0, 4, 0, 0, 0, 0, 27, 0, 0, 2, 0, 0, 0, 0\n0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 55, 0]'),
 Text(772.6153846153845, 181.19999999999982, 'gini = 0.388\nsamples = 32\nvalue = [0, 3, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 40, 0]'),
```

Results

1.Linear regression : 0.47252370822056

2.Ridge regression : 0.47253459053957125

3.Lasso regression : 0.4716378609310229

4.Elastic net regression : 0.4721303407512927

5.Logistic regresssion : 0.83154

6.Random forest regression : 0.47662896145180605

Hence Logistic regression gives high accuracy for the madrid_2008 model.