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_2008.csv")
data1
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2008-06-01 01:00:00	NaN	0.47	NaN	NaN	NaN	83.089996	120.699997	NaN	16.990000	16.8899
1	2008-06-01 01:00:00	NaN	0.59	NaN	NaN	NaN	94.820000	130.399994	NaN	17.469999	19.0400
2	2008-06-01 01:00:00	NaN	0.55	NaN	NaN	NaN	75.919998	104.599998	NaN	13.470000	20.2700
3	2008-06-01 01:00:00	NaN	0.36	NaN	NaN	NaN	61.029999	66.559998	NaN	23.110001	10.8500
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.1600
...	...	...	...	...	...	...	...	...	...	...	
226387	2008-11-01 00:00:00	0.48	0.30	0.57	1.00	0.31	13.050000	14.160000	0.91	57.400002	5.4500
226388	2008-11-01 00:00:00	NaN	0.30	NaN	NaN	NaN	41.880001	48.500000	NaN	35.830002	15.0200
226389	2008-11-01 00:00:00	0.25	NaN	0.56	NaN	0.11	83.610001	102.199997	NaN	14.130000	17.5400
226390	2008-11-01 00:00:00	0.54	NaN	2.70	NaN	0.18	70.639999	81.860001	NaN	NaN	11.9100
226391	2008-11-01 00:00:00	0.75	0.36	1.20	2.75	0.16	58.240002	74.239998	1.64	31.910000	12.6900

226392 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 226392 entries, 0 to 226391
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     226392 non-null  object 
 1   BEN      67047 non-null   float64
 2   CO       208109 non-null  float64
 3   EBE      67044 non-null   float64
 4   MXY      25867 non-null   float64
 5   NMHC     85079 non-null   float64
 6   NO_2     225315 non-null  float64
 7   NOx      225311 non-null  float64
 8   OXY      25878 non-null   float64
 9   O_3      215716 non-null  float64
 10  PM10     220179 non-null  float64
 11  PM25     67833 non-null   float64
 12  PXY      25877 non-null   float64
 13  SO_2     225405 non-null  float64
 14  TCH      85107 non-null   float64
 15  TOL      66940 non-null   float64
 16  station  226392 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.4+ 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	2008-06-01 01:00:00	0.00	0.47	0.00	0.00	0.00	83.089996	120.699997	0.00	16.990000	16.88999
1	2008-06-01 01:00:00	0.00	0.59	0.00	0.00	0.00	94.820000	130.399994	0.00	17.469999	19.04000
2	2008-06-01 01:00:00	0.00	0.55	0.00	0.00	0.00	75.919998	104.599998	0.00	13.470000	20.27000
3	2008-06-01 01:00:00	0.00	0.36	0.00	0.00	0.00	61.029999	66.559998	0.00	23.110001	10.85000
4	2008-06-01 01:00:00	1.68	0.80	1.70	3.01	0.30	105.199997	214.899994	1.61	12.120000	37.16000
...	...	...	...	...	...	...	...	...	...	...	..
49995	2008-05-20 04:00:00	0.29	0.33	0.45	1.01	0.32	35.790001	36.389999	1.00	34.290001	11.75000
49996	2008-05-20 04:00:00	0.00	0.51	0.00	0.00	0.00	76.500000	160.000000	0.00	11.980000	32.18000
49997	2008-05-20 04:00:00	0.20	0.00	0.42	0.00	0.13	56.650002	57.250000	0.00	28.360001	4.57000
49998	2008-05-20 04:00:00	0.20	0.00	0.29	0.00	0.09	35.740002	45.279999	0.00	0.000000	8.52000
49999	2008-05-20 04:00:00	0.47	0.26	1.68	3.65	0.22	47.939999	64.250000	1.29	25.040001	13.32000

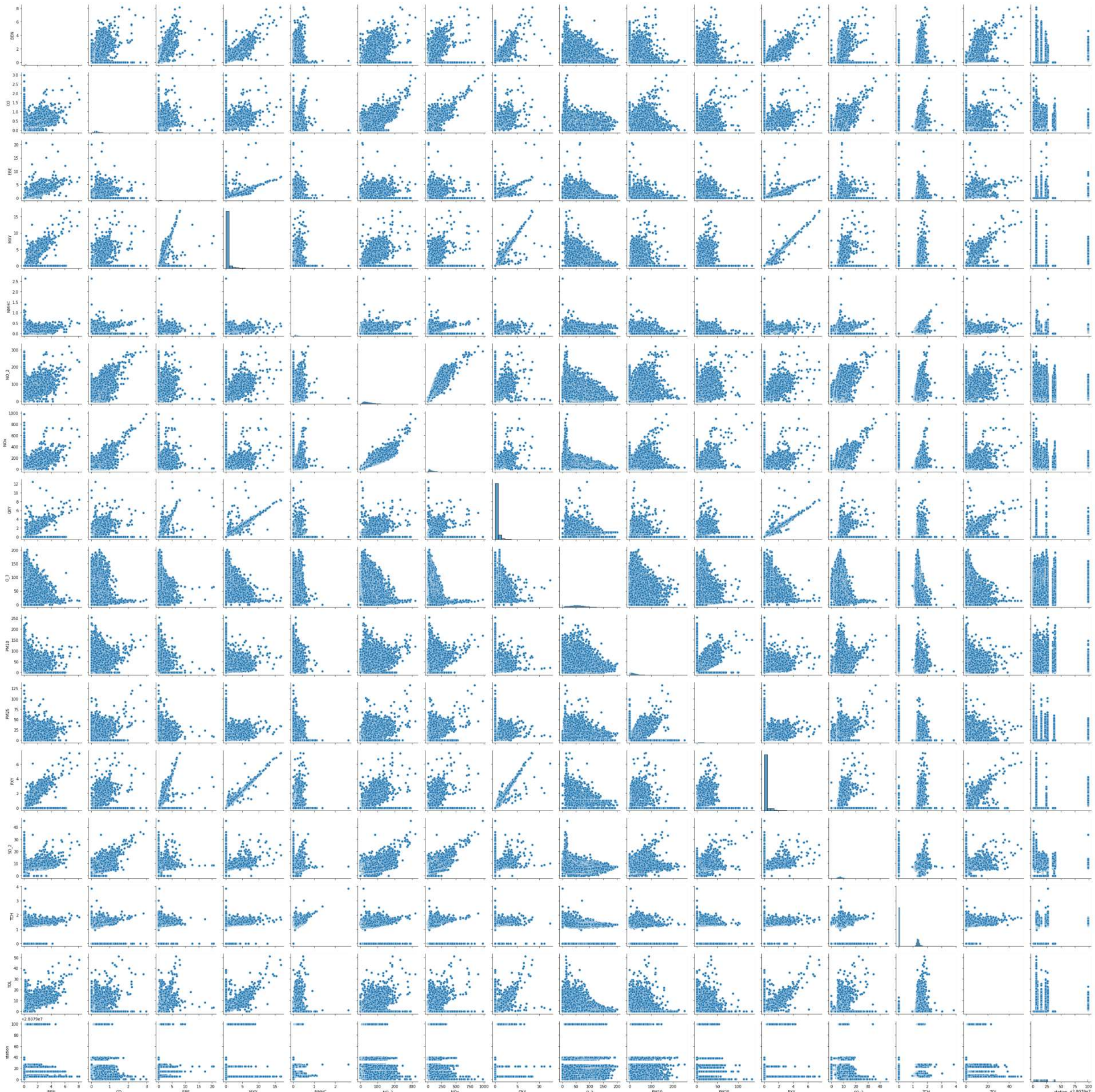
50000 rows × 17 columns

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)
```

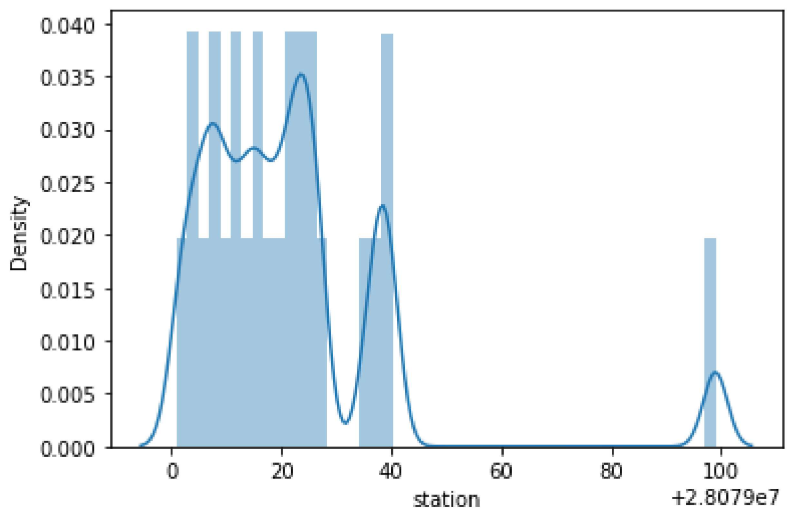
Out[8]: <seaborn.axisgrid.PairGrid at 0x277e40def70>



In [9]:
```
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[9]:
```
<AxesSubplot:xlabel='station', 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[['station']]
```

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)
```

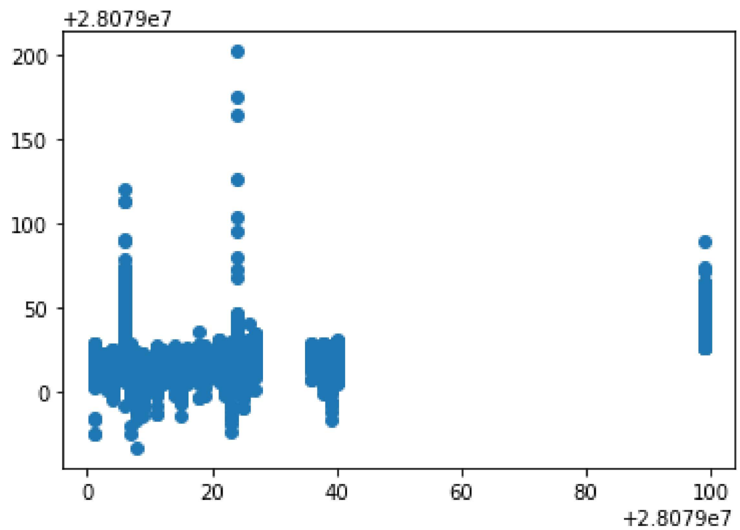
Out[13]:
```
LinearRegression()
```

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

```
[28079021.77127241]
```

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

Out[15]:
```
<matplotlib.collections.PathCollection at 0x27787f4f4c0>
```



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

```
0.17398490424114188
```

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]:
```python
rr.score(x_test,y_test)
```

Out[19]:
```
0.17407543842870543
```

3.Lasso Regression

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

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

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

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

Out[22]:
```
0.02160682046655149
```

4.ElasticNet Regression

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

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

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

```
[ 0.         -0.          0.64567519  0.          0.03324613 -0.11230503
  2.86502372  0.05947635  2.20038189  0.81758234  0.12316062  0.19045421
 -0.03519829]
```

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

```
[28079022.14724725 28079020.04075697 28079023.17655486 ...
 28079021.32527933 28079017.82509347 28079021.59910372]
```

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

```
0.10799922293971642
```

5.Logistic Regression

In [27]:
```python
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, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079018, 28079019, 28079021, 28079022, 28079023, 28079024,
       28079025, 28079026, 28079027, 28079036, 28079038, 28079039,
       28079040, 28079099], dtype=int64)
```

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

Out[37]:
```
0.89506
```

6.Random Forest

In [38]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3','PM10', 'F
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]:
```
0.527394713104829
```

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]'),
 Text(313.2631578947368, 181.19999999999982, 'gini = 0.804\nsamples = 699\nva
lue = [112, 0, 6, 0, 5, 1, 0, 0, 0, 26, 10, 16, 1\n13, 261, 3, 0, 0, 0, 0, 0,
43, 193, 90, 347, 0]'),
 Text(704.8421052631578, 906.0, 'SO_2 <= 6.685\ngini = 0.845\nsamples = 1766
\nvalue = [80, 8, 9, 0, 7, 0, 1, 0, 0, 53, 9, 595, 88\n19, 72, 390, 2, 6, 0,
1, 0, 40, 580, 439, 444\n0]'),
 Text(548.2105263157895, 543.5999999999999, 'O_3 <= 83.995\ngini = 0.63\nsamp
les = 679\nvalue = [1, 8, 0, 0, 4, 0, 1, 0, 0, 14, 0, 541, 84, 9\n0, 388, 2,
0, 0, 1, 0, 0, 1, 15, 36, 0]'),
 Text(469.8947368421052, 181.19999999999982, 'gini = 0.652\nsamples = 398\nva
lue = [1, 6, 0, 0, 4, 0, 1, 0, 0, 11, 0, 325, 49, 9\n0, 202, 2, 0, 0, 1, 0,
0, 1, 15, 30, 0]'),
 Text(626.5263157894736, 181.19999999999982, 'gini = 0.589\nsamples = 281\nva
lue = [0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 0, 216, 35, 0\n0, 186, 0, 0, 0, 0, 0, 0,
0, 0, 6, 0]'),
 Text(861.4736842105262, 543.5999999999999, 'NO_2 <= 9.605\ngini = 0.769\nsam
ples = 1087\nvalue = [79, 0, 9, 0, 3, 0, 0, 0, 0, 39, 9, 54, 4, 10\n72, 2, 0,
6, 0, 0, 0, 40, 579, 424, 408, 0]'),
```

Results

1.Linear regression : 0.17398490424114188

2.Ridge regression : 0.17407543842870543

3.Lasso regression : 0.02160682046655149

4.Elasticnet regression : 0.10799922293971642

5.Logistic regresssion : 0.89506

6.Random forest regression : 0.527394713104829

Hence Logistic regression gives high accuracy for the madrid_2008 model.