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_2006.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2006-02-01 01:00:00	NaN	1.84	NaN	NaN	NaN	155.100006	490.100006	NaN	4.880000	97.5700
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.8200
2	2006-02-01 01:00:00	NaN	1.25	NaN	NaN	NaN	66.800003	192.000000	NaN	4.430000	34.4199
3	2006-02-01 01:00:00	NaN	1.68	NaN	NaN	NaN	103.000000	407.799988	NaN	4.830000	28.2600
4	2006-02-01 01:00:00	NaN	1.31	NaN	NaN	NaN	105.400002	269.200012	NaN	6.990000	54.1800
...	...	...	...	...	...	...	...	...	...	...	
230563	2006-05-01 00:00:00	5.88	0.83	6.23	NaN	0.20	112.500000	218.000000	NaN	24.389999	93.1200
230564	2006-05-01 00:00:00	0.76	0.32	0.48	1.09	0.08	51.900002	54.820000	0.61	48.410000	29.4699
230565	2006-05-01 00:00:00	0.96	NaN	0.69	NaN	0.19	135.100006	179.199997	NaN	11.460000	64.6800
230566	2006-05-01 00:00:00	0.50	NaN	0.67	NaN	0.10	82.599998	105.599998	NaN	NaN	94.3600
230567	2006-05-01 00:00:00	1.95	0.74	1.99	4.00	0.24	107.300003	160.199997	2.01	17.730000	52.4900

230568 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 230568 entries, 0 to 230567
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     230568 non-null  object 
 1   BEN      73979 non-null   float64
 2   CO       211665 non-null  float64
 3   EBE      73948 non-null   float64
 4   MXY      33422 non-null   float64
 5   NMHC     90829 non-null   float64
 6   NO_2     228855 non-null  float64
 7   NOx      228855 non-null  float64
 8   OXY      33472 non-null   float64
 9   O_3      216511 non-null  float64
 10  PM10     227469 non-null  float64
 11  PM25     61758 non-null   float64
 12  PXY      33447 non-null   float64
 13  SO_2     229125 non-null  float64
 14  TCH      90887 non-null   float64
 15  TOL      73840 non-null   float64
 16  station  230568 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.9+ MB
```

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

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

Out[5]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2006-02-01 01:00:00	0.00	1.84	0.00	0.00	0.00	155.100006	490.100006	0.00	4.880000	97.5700
1	2006-02-01 01:00:00	1.68	1.01	2.38	6.36	0.32	94.339996	229.699997	3.04	7.100000	25.8200
2	2006-02-01 01:00:00	0.00	1.25	0.00	0.00	0.00	66.800003	192.000000	0.00	4.430000	34.4199
3	2006-02-01 01:00:00	0.00	1.68	0.00	0.00	0.00	103.000000	407.799988	0.00	4.830000	28.2600
4	2006-02-01 01:00:00	0.00	1.31	0.00	0.00	0.00	105.400002	269.200012	0.00	6.990000	54.1800
...	...	...	...	...	...	...	...	...	...	...	
49995	2006-06-21 23:00:00	0.00	0.74	0.00	0.00	0.00	110.599998	125.300003	0.00	47.700001	52.1199
49996	2006-06-21 23:00:00	0.00	0.67	0.00	0.00	0.43	81.430000	85.779999	0.00	50.410000	46.6800
49997	2006-06-21 23:00:00	0.00	0.90	0.00	0.00	0.00	86.559998	192.100006	0.00	8.380000	63.5700
49998	2006-06-21 23:00:00	0.00	0.41	0.00	0.00	0.00	78.629997	88.400002	0.00	30.959999	55.2999
49999	2006-06-21 23:00:00	0.00	0.51	0.00	0.00	0.00	115.000000	140.500000	0.00	19.129999	102.3000

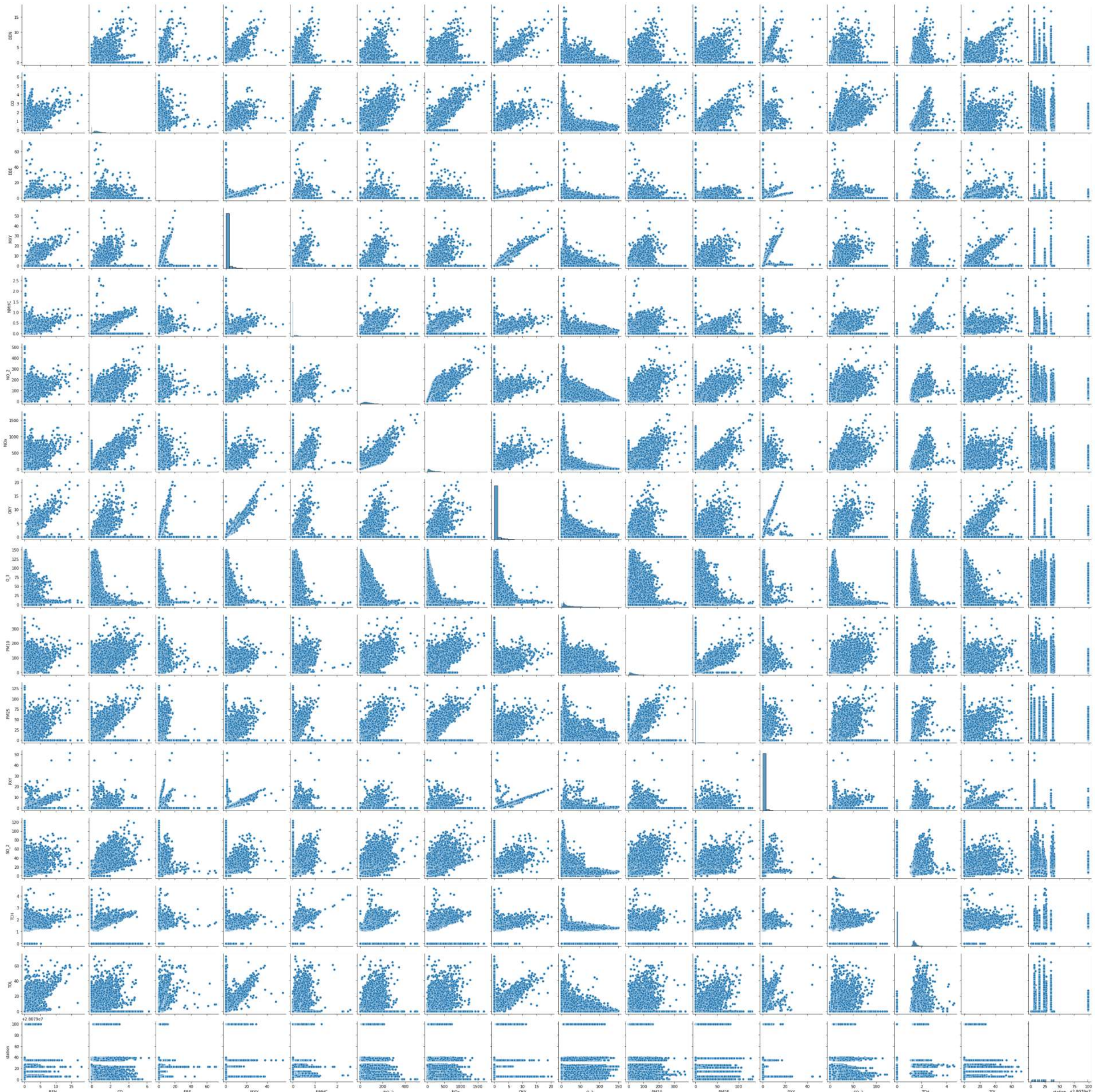
50000 rows × 17 columns

In [6]:
```python
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)
```

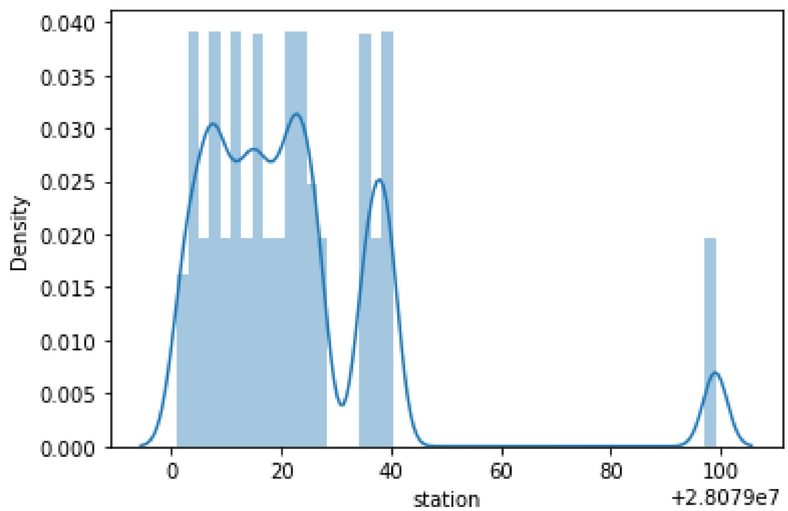
Out[7]: <seaborn.axisgrid.PairGrid at 0x25430cf61f0>



In [8]:
```
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[8]:
```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



MODEL BUILDING

1.Linear Regression

In [9]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [10]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df1[['station']]
```

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)
```

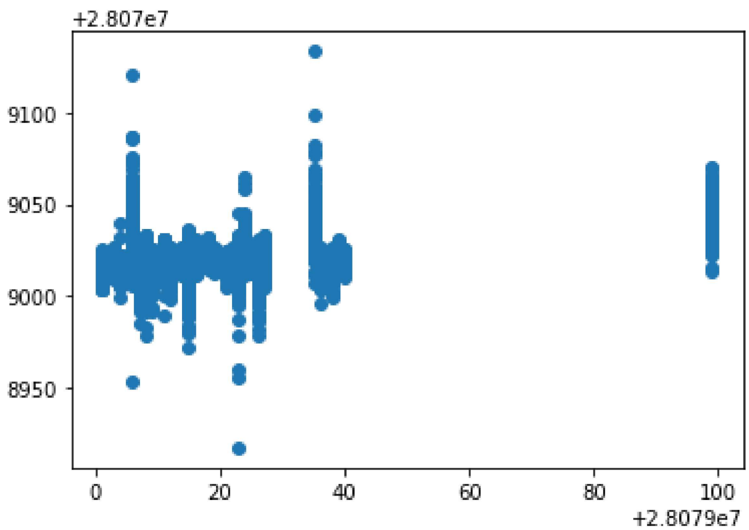
Out[12]:
```
LinearRegression()
```

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

```
[28079021.55268988]
```

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

Out[14]:
```
<matplotlib.collections.PathCollection at 0x25455ce8cd0>
```



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

```
0.12268730249242776
```

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]:
```
0.12271730012488524
```

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]:
```
0.01792187654069788
```

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.          -0.01441676 -0.00838733
  2.54635135  0.0352772   1.00609046 -0.24127562  0.52822346  0.01561784
 -0.00987587]
```

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

```
[28079022.93875691 28079017.94964714 28079042.03127455 ...
 28079020.94663141 28079020.88617244 28079020.86766885]
```

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

```
0.08357986438258 38
```

5.Logistic Regression

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

In [27]:
```
feature_matrix = df1.iloc[:,0:16]
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 [33]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

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

```
[28079099]
```

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

Out[35]:
```
array([28079001, 28079003, 28079004, 28079006, 28079007, 28079008,
       28079009, 28079011, 28079012, 28079014, 28079015, 28079016,
       28079018, 28079019, 28079021, 28079022, 28079023, 28079024,
       28079025, 28079026, 28079027, 28079035, 28079036, 28079038,
       28079039, 28079040, 28079099], dtype=int64)
```

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

Out[36]:
```
0.89522
```

6.Random Forest

In [37]:
```
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 [38]:
```
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 [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]:
```
0.47881107549043606
```

In [43]:
```
rfc_best = grid_search.best_estimator_
```

In [44]:
```python
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

Out[44]:
```
[Text(1755.375, 1993.2, 'CO <= 0.005\ngini = 0.962\nsamples = 31614\nvalue =
[1590, 1872, 1981, 1916, 1867, 1943, 1946, 1827, 1983\n1916, 1924, 1901, 192
1, 1890, 1933, 1880, 2002, 1833\n489, 1903, 1893, 1904, 1908, 1968, 1924, 195
0, 1891]'),
 Text(534.75, 1630.8000000000002, 'NOx <= 7.005\ngini = 0.604\nsamples = 2715
\nvalue = [14, 30, 0, 29, 29, 7, 32, 1, 5, 28, 5, 82, 4\n17, 3, 24, 21, 74, 1
4, 1903, 1893, 0, 26, 0, 27\n1, 0]'),
 Text(186.0, 1268.4, 'PM10 <= 1.905\ngini = 0.929\nsamples = 206\nvalue = [1
4, 30, 0, 27, 29, 7, 32, 0, 0, 28, 2, 16, 4\n17, 1, 10, 20, 5, 14, 12, 1, 0,
25, 0, 27, 0\n0]'),
 Text(93.0, 906.0, 'gini = 0.924\nsamples = 195\nvalue = [10, 30, 0, 27, 29,
7, 32, 0, 0, 28, 2, 15, 2\n17, 1, 2, 18, 5, 14, 12, 0, 0, 25, 0, 27, 0\n0]'),
 Text(279.0, 906.0, 'PM10 <= 14.315\ngini = 0.722\nsamples = 11\nvalue = [4,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0\n0, 8, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0]'),
 Text(186.0, 543.5999999999999, 'gini = 0.0\nsamples = 6\nvalue = [0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(372.0, 543.5999999999999, 'gini = 0.74\nsamples = 5\nvalue = [4, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0\n0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]'),
```

Results

```
1.Linear regression : 0.12271730012488524

2.Ridge regression : 0.12271730012488524

3.Lasso regression :0.01792187654069788

4.Elasticnet regression : 0.0835798643825838

5.Logistic regresssion : 0.89522

6.Random forest regression : 0.47881107549043606

Hence Logistic regression gives high accuracy for the madrid_2006 model.
```