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_2004.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2004-08-01 01:00:00	NaN	0.66	NaN	NaN	NaN	89.550003	118.900002	NaN	40.020000	39.9900
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22.9500
2	2004-08-01 01:00:00	NaN	1.02	NaN	NaN	NaN	93.389999	138.600006	NaN	20.860001	49.4800
3	2004-08-01 01:00:00	NaN	0.53	NaN	NaN	NaN	87.290001	105.000000	NaN	36.730000	31.0700
4	2004-08-01 01:00:00	NaN	0.17	NaN	NaN	NaN	34.910000	35.349998	NaN	86.269997	54.0800
...	...	...	...	...	...	...	...	...	...	...	
245491	2004-06-01 00:00:00	0.75	0.21	0.85	1.55	0.07	59.580002	64.389999	0.66	33.029999	30.9000
245492	2004-06-01 00:00:00	2.49	0.75	2.44	4.57	NaN	97.139999	146.899994	2.34	7.740000	37.6899
245493	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.13	102.699997	132.600006	NaN	17.809999	22.8400
245494	2004-06-01 00:00:00	NaN	NaN	NaN	NaN	0.09	82.599998	102.599998	NaN	NaN	45.6300
245495	2004-06-01 00:00:00	3.01	0.67	2.78	5.12	0.20	92.550003	141.000000	2.60	11.460000	24.3899

245496 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 245496 entries, 0 to 245495
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     245496 non-null  object 
 1   BEN      65158 non-null   float64
 2   CO       226043 non-null  float64
 3   EBE      56781 non-null   float64
 4   MXY      39867 non-null   float64
 5   NMHC     107630 non-null  float64
 6   NO_2     243280 non-null  float64
 7   NOx      243283 non-null  float64
 8   OXY      39882 non-null   float64
 9   O_3      233811 non-null  float64
 10  PM10     234655 non-null  float64
 11  PM25     58145 non-null   float64
 12  PXY      39891 non-null   float64
 13  SO_2     243402 non-null  float64
 14  TCH      107650 non-null  float64
 15  TOL      64914 non-null   float64
 16  station  245496 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 31.8+ MB
```

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

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

Out[5]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2004-08-01 01:00:00	0.00	0.66	0.00	0.00	0.00	89.550003	118.900002	0.00	40.020000	39.990002
1	2004-08-01 01:00:00	2.66	0.54	2.99	6.08	0.18	51.799999	53.860001	3.28	51.689999	22.950001
2	2004-08-01 01:00:00	0.00	1.02	0.00	0.00	0.00	93.389999	138.600006	0.00	20.860001	49.480000
3	2004-08-01 01:00:00	0.00	0.53	0.00	0.00	0.00	87.290001	105.000000	0.00	36.730000	31.070000
4	2004-08-01 01:00:00	0.00	0.17	0.00	0.00	0.00	34.910000	35.349998	0.00	86.269997	54.080002
...	...	...	...	...	...	...	...	...	...	...	...
49995	2004-03-14 13:00:00	0.00	0.42	0.00	0.00	0.00	38.070000	50.389999	0.00	60.299999	9.540000
49996	2004-03-14 13:00:00	0.00	0.12	0.00	0.00	0.00	14.940000	20.059999	0.00	60.220001	6.460000
49997	2004-03-14 13:00:00	1.75	0.56	1.38	2.86	0.08	47.490002	89.339996	1.46	47.070000	12.980000
49998	2004-03-14 13:00:00	0.00	0.43	0.00	0.00	0.12	32.970001	44.410000	0.00	59.610001	7.740000
49999	2004-03-14 13:00:00	0.11	0.47	1.00	0.00	0.04	40.349998	56.369999	0.00	66.879997	8.750000

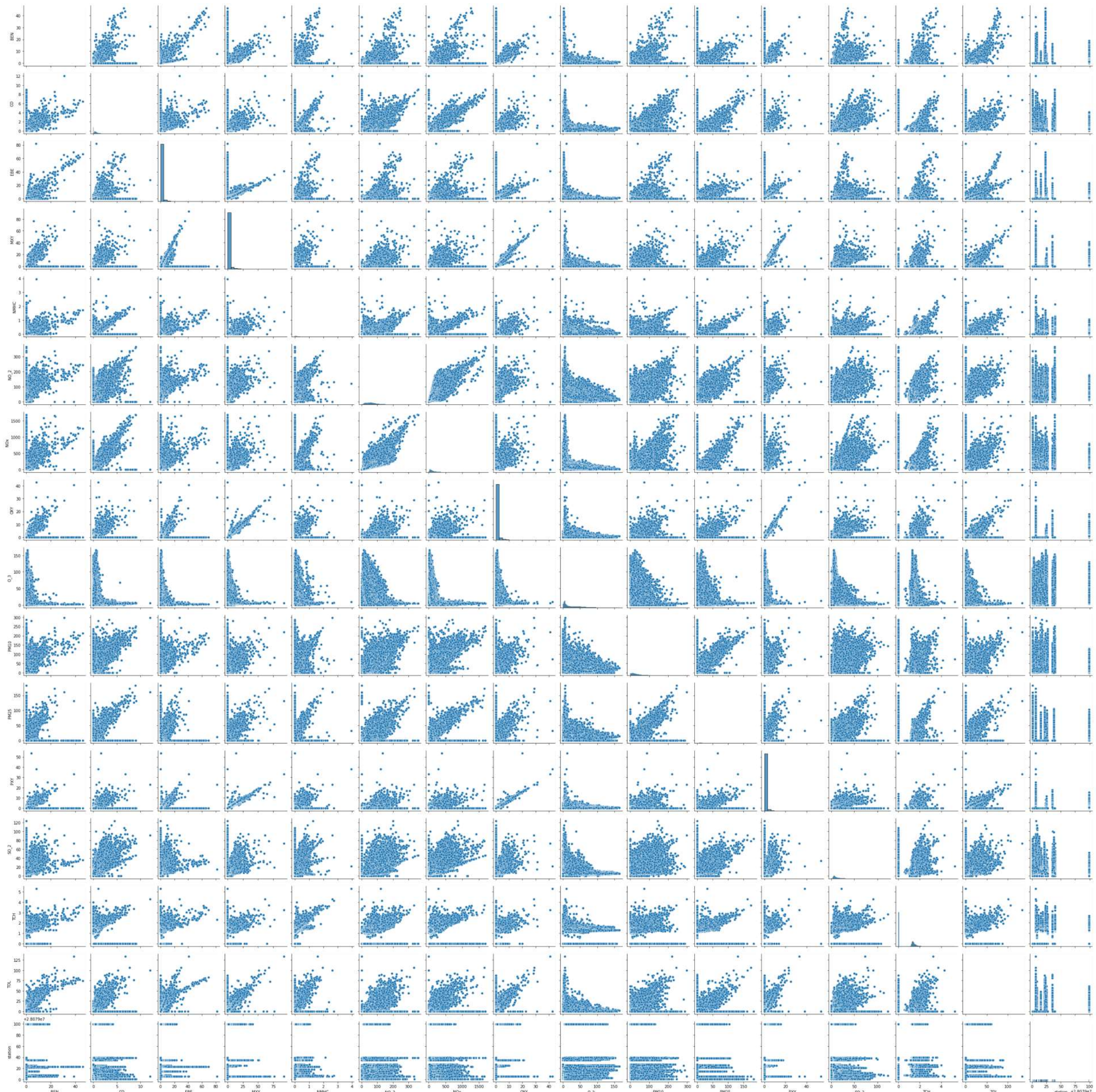
50000 rows × 17 columns

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

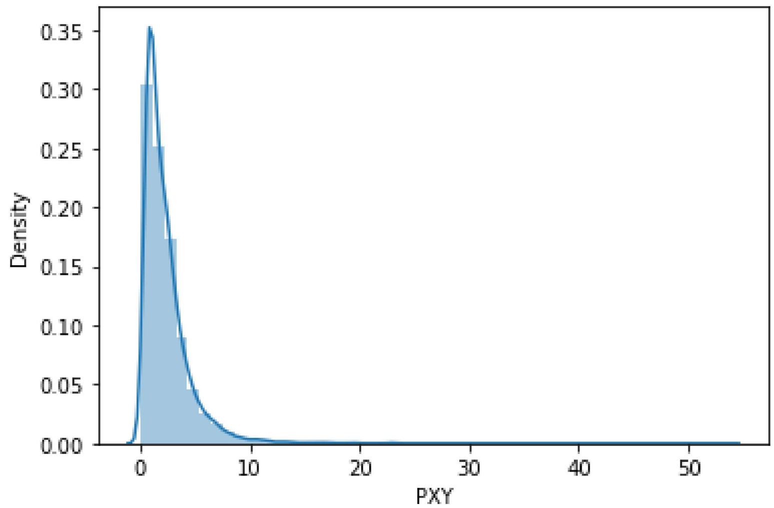
Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x22244fa5400>
```



In [8]:
```
sns.distplot(data["PXY"])
```

```
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='PXY', 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', 'SO_2', 'TCH', 'T
y=df1[['PXY']]
```

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)
```

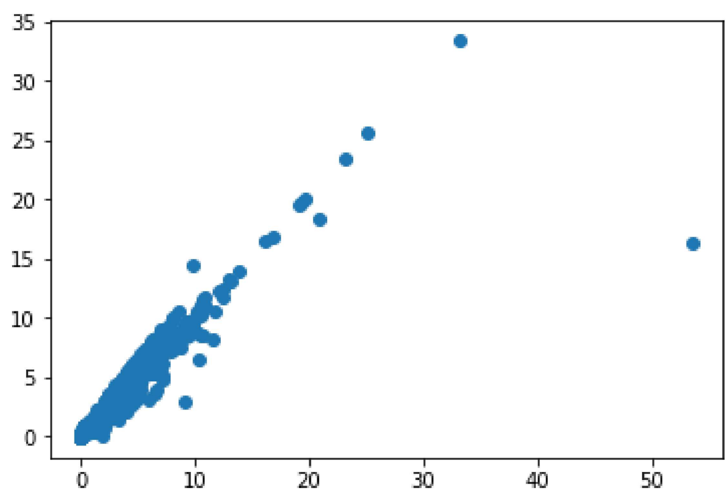
Out[12]: LinearRegression()

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

```
[0.02406518]
```

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

Out[14]: <matplotlib.collections.PathCollection at 0x2226a68b640>



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

```
0.9264209694914435
```

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]:
```
0.926412979280104
```

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]:
```
0.04087321348158257
```

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_)
```

```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  6.20477781e-04  8.79739361e-05  3.20671656e-01 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  5.55930879e-02  9.85827459e-05]
```

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

```
[0.08691586 0.04410173 0.69683633 ... 1.58716533 0.076783   0.66184054]
```

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

```
0.7271915365221034
```

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,
       28079017, 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]:
```
0.88602
```

6.Random Forest

In [37]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2',  'station']]
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2']]
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]:
```
0.3548793242733912
```

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)
```

```
\nvalue = [0, 0, 0, 319, 0, 784, 0, 0, 0, 0, 484, 0, 0\n0, 0, 0, 0, 206, 132,
0, 0, 0, 111, 0, 0, 0\n0, 88]'),
 Text(3236.3999999999996, 181.19999999999982, 'gini = 0.739\nsamples = 1197\n
value = [0, 0, 0, 274, 0, 778, 0, 0, 0, 0, 467, 0, 0\n0, 0, 0, 0, 56, 131, 0,
0, 0, 102, 0, 0, 0\n0, 88]'),
 Text(3459.6, 181.19999999999982, 'gini = 0.52\nsamples = 139\nvalue = [0, 0,
0, 45, 0, 6, 0, 0, 0, 0, 17, 0, 0, 0\n0, 0, 0, 150, 1, 0, 0, 0, 9, 0, 0, 0,
0, 0]'),
 Text(4017.6, 906.0, 'NO_2 <= 73.35\ngini = 0.807\nsamples = 3853\nvalue =
[0, 0, 0, 1230, 0, 818, 0, 0, 0, 0, 301, 0, 0\n0, 0, 0, 0, 455, 230, 0, 0, 0,
1538, 0, 0, 0\n0, 1471]'),
 Text(3794.3999999999996, 543.5999999999999, 'EBE <= 5.315\ngini = 0.716\nsam
ples = 1657\nvalue = [0, 0, 0, 217, 0, 97, 0, 0, 0, 0, 32, 0, 0, 0\n0, 0,
178, 177, 0, 0, 0, 893, 0, 0, 0, 0\n991]'),
 Text(3682.7999999999997, 181.19999999999982, 'gini = 0.695\nsamples = 1560\n
value = [0, 0, 0, 205, 0, 97, 0, 0, 0, 0, 29, 0, 0, 0\n0, 0, 76, 177, 0,
0, 0, 875, 0, 0, 0, 0\n978]'),
 Text(3906.0, 181.19999999999982, 'gini = 0.496\nsamples = 97\nvalue = [0, 0,
0, 12, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0\n0, 0, 102, 0, 0, 0, 0, 18, 0, 0, 0,
0, 13]'\
```

Results

1.Linear regression : 0.9264209694914435

2.Ridge regression : 0.926412979280104

3.Lasso regression : 0.04087321348158257

4.Elasticnet regression : 0.7271915365221034

5.Logistic regresssion : 0.88602

6.Random forest regression : 0.3548793242733912

Hence Linear regression gives high accuracy for the madrid_2004 model.