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_2009.csv")
data1
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM1
0	2009-10-01 01:00:00	NaN	0.27	NaN	NaN	NaN	39.889999	48.150002	NaN	50.680000	18.26000
1	2009-10-01 01:00:00	NaN	0.22	NaN	NaN	NaN	21.230000	24.260000	NaN	55.880001	10.58000
2	2009-10-01 01:00:00	NaN	0.18	NaN	NaN	NaN	31.230000	34.880001	NaN	49.060001	25.19000
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.53000
4	2009-10-01 01:00:00	NaN	0.41	NaN	NaN	0.12	61.349998	76.260002	NaN	38.090000	23.76000
...	...	...	...	...	...	...	...	...	...	...	.
215683	2009-06-01 00:00:00	0.50	0.22	0.39	0.75	0.09	22.000000	24.510000	1.00	82.239998	10.83000
215684	2009-06-01 00:00:00	NaN	0.31	NaN	NaN	NaN	76.110001	101.099998	NaN	41.220001	9.92000
215685	2009-06-01 00:00:00	0.13	NaN	0.86	NaN	0.23	81.050003	99.849998	NaN	24.830000	12.46000
215686	2009-06-01 00:00:00	0.21	NaN	2.96	NaN	0.10	72.419998	82.959999	NaN	NaN	13.03000
215687	2009-06-01 00:00:00	0.37	0.32	0.99	1.36	0.14	54.290001	64.480003	1.06	56.919998	15.36000

215688 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215688 entries, 0 to 215687
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     215688 non-null  object 
 1   BEN      60082 non-null   float64
 2   CO       190801 non-null  float64
 3   EBE      60081 non-null   float64
 4   MXY      24846 non-null   float64
 5   NMHC     74748 non-null   float64
 6   NO_2     214562 non-null  float64
 7   NOx      214565 non-null  float64
 8   OXY      24854 non-null   float64
 9   O_3      204482 non-null  float64
 10  PM10     196331 non-null  float64
 11  PM25     55822 non-null   float64
 12  PXY      24854 non-null   float64
 13  SO_2     212671 non-null  float64
 14  TCH      75213 non-null   float64
 15  TOL      59920 non-null   float64
 16  station  215688 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 28.0+ 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	PM1
0	2009-10-01 01:00:00	0.00	0.27	0.00	0.00	0.00	39.889999	48.150002	0.00	50.680000	18.26000
1	2009-10-01 01:00:00	0.00	0.22	0.00	0.00	0.00	21.230000	24.260000	0.00	55.880001	10.58000
2	2009-10-01 01:00:00	0.00	0.18	0.00	0.00	0.00	31.230000	34.880001	0.00	49.060001	25.19000
3	2009-10-01 01:00:00	0.95	0.33	1.43	2.68	0.25	55.180000	81.360001	1.57	36.669998	26.53000
4	2009-10-01 01:00:00	0.00	0.41	0.00	0.00	0.12	61.349998	76.260002	0.00	38.090000	23.76000
...	...	...	...	...	...	...	...	...	...	...	.
49995	2009-09-22 09:00:00	0.49	0.45	0.43	0.00	0.08	80.260002	159.300003	0.00	16.520000	0.00000
49996	2009-09-22 09:00:00	0.43	0.65	0.52	1.00	0.75	49.860001	57.209999	1.00	24.760000	13.15000
49997	2009-09-22 09:00:00	0.00	0.57	0.00	0.00	0.00	132.899994	291.299988	0.00	9.780000	25.92000
49998	2009-09-22 09:00:00	0.28	0.00	0.47	0.00	0.26	80.089996	111.500000	0.00	10.670000	8.49000
49999	2009-09-22 09:00:00	1.15	0.00	0.60	0.00	0.18	95.199997	150.000000	0.00	0.000000	13.17000

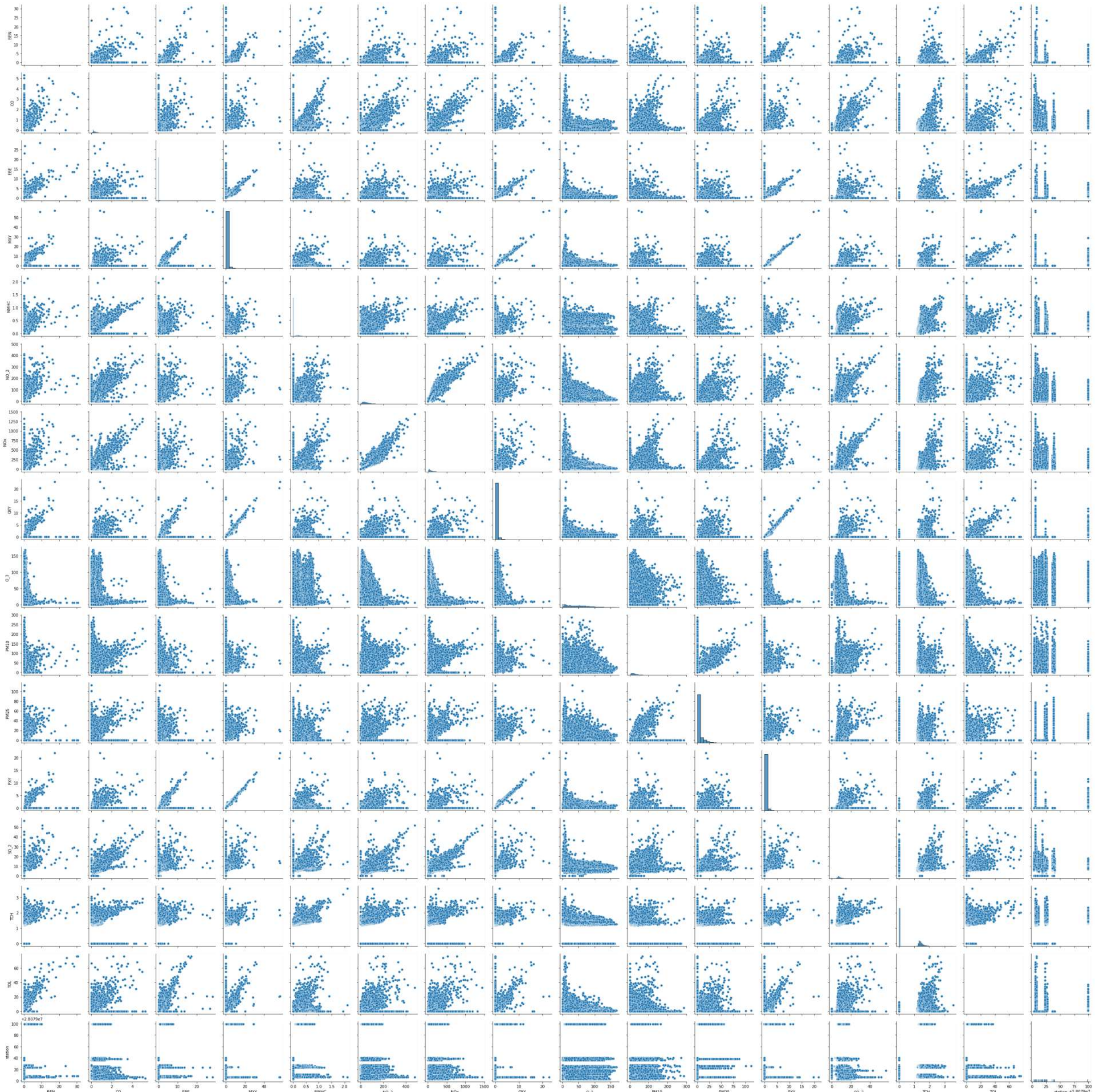
50000 rows × 17 columns

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)
```

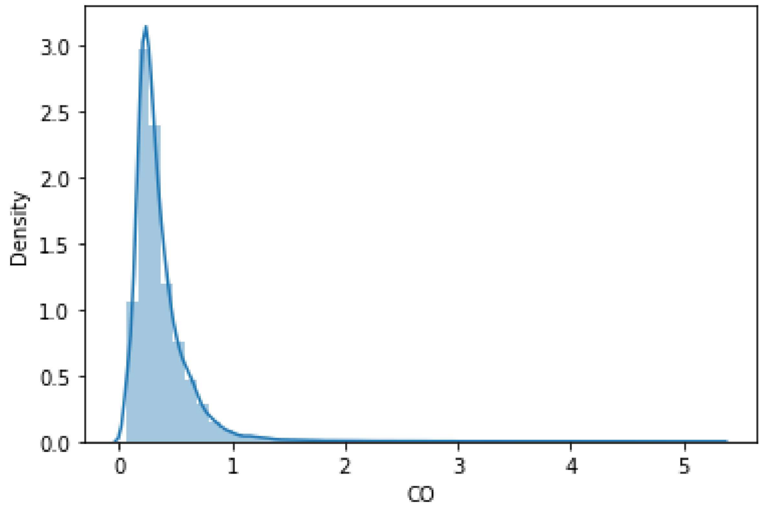
Out[7]: <seaborn.axisgrid.PairGrid at 0x2791f338d30>



In [11]:
```
sns.distplot(data["CO"])
```

```
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[11]:
```
<AxesSubplot:xlabel='CO', ylabel='Density'>
```



MODEL BUILDING

1.Linear Regression

In [12]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [18]:
```
x=df1[['BEN','EBE', 'NMHC', 'NO_2', 'NOx', 'OXY', 'PXY', 'SO_2', 'TCH', 'TOL','O_
y=df1[['CO']]
```

In [19]:
```
#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 [20]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

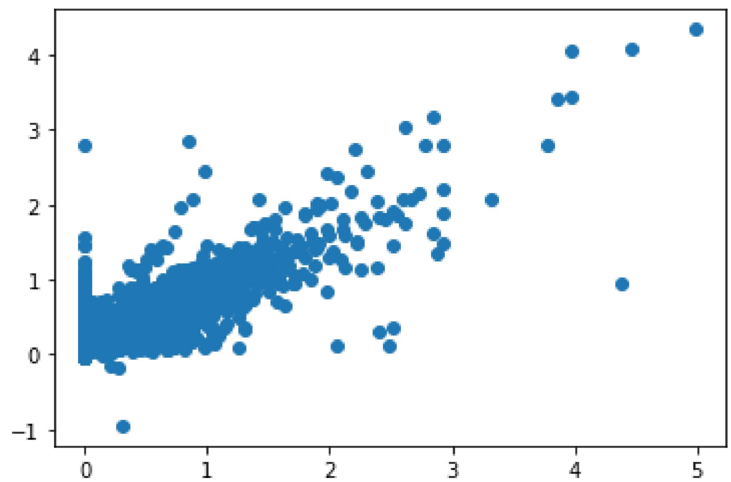
Out[20]:
```
LinearRegression()
```

In [21]:
```
print(lr.intercept_)
```

```
[0.04373171]
```

In [22]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[22]:
```
<matplotlib.collections.PathCollection at 0x2794688a6a0>
```



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

```
0.6227533529560285
```

2.Ridge Regression

In [24]:
```
from sklearn.linear_model import Ridge
```

In [25]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[25]:
```
Ridge(alpha=10)
```

In [26]:
```
rr.score(x_test,y_test)
```

Out[26]:
```
0.6228324506367963
```

3.Lasso Regression

In [27]:
```
from sklearn.linear_model import Lasso
```

In [28]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[28]:
```
Lasso(alpha=10)
```

In [29]:
```
la.score(x_test,y_test)
```

Out[29]:
```
0.34904580452002887
```

4.ElasticNet Regression

In [30]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[30]:
```
ElasticNet()
```

In [31]:
```
print(en.coef_)
```

```
[ 0.          0.          0.          0.          0.00252639  0.
  0.          0.         -0.          0.          0.00061329]
```

In [32]:
```
print(en.predict(x_test))
```

```
[0.72692391 0.2298091  0.28704567 ... 0.16254928 0.21697574 0.45624914]
```

In [33]:
```
print(en.score(x_test,y_test))
```

```
0.5766883556957916
```

5.Logistic Regression

In [34]:
```
from sklearn.linear_model import LogisticRegression
```

In [35]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

In [36]:
```
feature_matrix.shape
```

Out[36]:
```
(50000, 15)
```

In [37]:
```
target_vector.shape
```

Out[37]:
```
(50000,)
```

In [38]:
```
from sklearn.preprocessing import StandardScaler
```

In [39]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [40]:
```
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[40]:
```
LogisticRegression()
```

In [41]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [42]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [43]:
```
logr.classes_
```

Out[43]:
```
array([28079003, 28079004, 28079006, 28079007, 28079008, 28079009,
       28079011, 28079012, 28079014, 28079016, 28079017, 28079018,
       28079019, 28079021, 28079022, 28079023, 28079024, 28079025,
       28079026, 28079027, 28079036, 28079038, 28079039, 28079040,
       28079099], dtype=int64)
```

In [44]:
```
logr.score(fs,target_vector)
```

Out[44]:
```
0.89362
```

6.Random Forest

In [60]:
```
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3','PM10', 'P
x=df1[['BEN','EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'TCH', 'T
y=df['station']
```

In [61]:
```
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 [62]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[62]:
```
RandomForestClassifier()
```

In [63]:
```
parameters = {'max_depth':[1,2,3,4,5],
    'min_samples_leaf':[5,10,15,20,25],
    'n_estimators':[10,20,30,40,50]}
```

In [64]:
```
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[64]:
```
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 [65]:
```
grid_search.best_score_
```

Out[65]:
```
0.5333200507153746
```

In [66]:
```
rfc_best = grid_search.best_estimator_
```

In [67]:
```
from sklearn.tree import plot_tree

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

```
0]'),
 Text(3124.8, 906.0, 'EBE <= 2.87\ngini = 0.002\nsamples = 1074\nvalue = [0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 1701, 0, 0, 0, 1, 0, 0, 0, 0, 0]'),
 Text(3035.52, 543.5999999999999, 'TOL <= 0.315\ngini = 0.001\nsamples = 1069
\nvalue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 1695, 0, 0, 0, 1, 0,
0, 0, 0, 0]'),
 Text(2946.2400000000002, 181.19999999999982, 'gini = 0.32\nsamples = 5\nvalu
e = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 4, 0, 0, 0, 1, 0, 0, 0, 0,
0]'),
 Text(3124.8, 181.19999999999982, 'gini = 0.0\nsamples = 1064\nvalue = [0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 1691, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(3214.08, 543.5999999999999, 'gini = 0.245\nsamples = 5\nvalue = [0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),

 Text(3481.92, 906.0, 'BEN <= 0.215\ngini = 0.045\nsamples = 167\nvalue = [0,
0, 1, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 258, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(3392.64, 543.5999999999999, 'gini = 0.473\nsamples = 7\nvalue = [0, 0,
0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(3571.2, 543.5999999999999, 'TOL <= 4.435\ngini = 0.008\nsamples = 160\n
value = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 250, 0, 0, 0, 0, 0, 0,
```

Results

1.Linear regression :0.6227533529560285

2.Ridge regression : 0.6228324506367963

3.Lasso regression : 0.34904580452002887

4.Elastic net regression : 0.5766883556957916

5.Logistic regresssion : 0.89362

6.Random forest regression : 0.5333200507153746

Hence Logistic regression gives high accuracy for the madrid_2009 model.