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_2003.csv")
data1
```

Out[3]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM1
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.20999
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.38999
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.24000
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.83999
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.77999
...	...	...	...	...	...	...	...	...	...	...	.
243979	2003-10-01 00:00:00	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.38000
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.40000
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.83000
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.57000
243983	2003-10-01 00:00:00	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.35000

243984 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243984 entries, 0 to 243983
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     243984 non-null  object 
 1   BEN      69745 non-null   float64
 2   CO       225340 non-null  float64
 3   EBE      61244 non-null   float64
 4   MXY      42045 non-null   float64
 5   NMHC     111951 non-null  float64
 6   NO_2     242625 non-null  float64
 7   NOx      242629 non-null  float64
 8   OXY      42072 non-null   float64
 9   O_3      234131 non-null  float64
 10  PM10     240896 non-null  float64
 11  PXY      42063 non-null   float64
 12  SO_2     242729 non-null  float64
 13  TCH      111991 non-null  float64
 14  TOL      69439 non-null   float64
 15  station  243984 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 29.8+ MB
```

In [5]:
```
data=data1.head(50000)
```

In [6]:
```
#filling null values
df=data.fillna(0)
df
```

Out[6]:

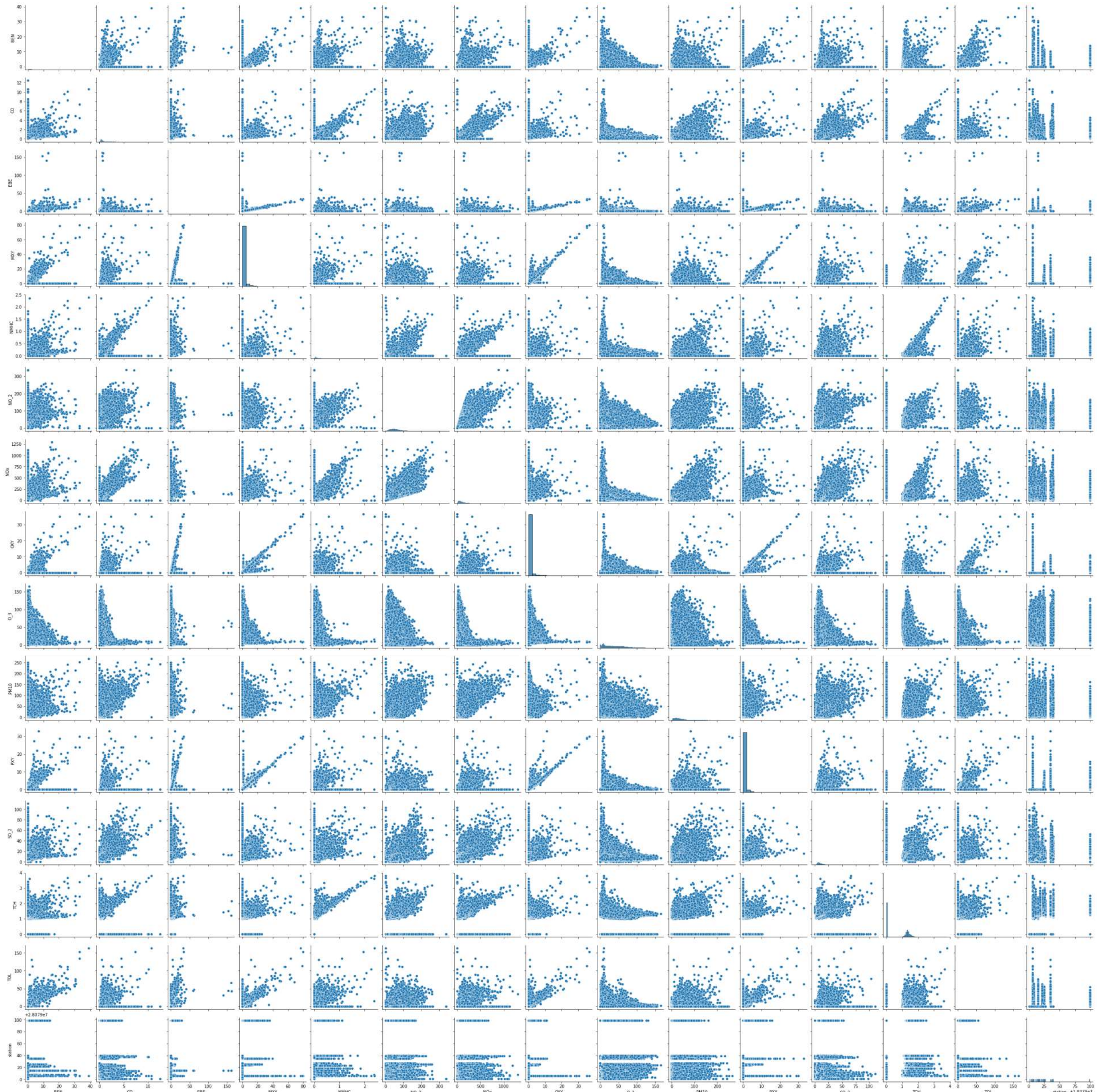
	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
0	2003-03-01 01:00:00	0.00	1.72	0.00	0.0	0.00	73.900002	316.299988	0.00	10.550000	55.209999
1	2003-03-01 01:00:00	0.00	1.45	0.00	0.0	0.26	72.110001	250.000000	0.73	6.720000	52.389999
2	2003-03-01 01:00:00	0.00	1.57	0.00	0.0	0.00	80.559998	224.199997	0.00	21.049999	63.240002
3	2003-03-01 01:00:00	0.00	2.45	0.00	0.0	0.00	78.370003	450.399994	0.00	4.220000	67.839996
4	2003-03-01 01:00:00	0.00	3.26	0.00	0.0	0.00	96.250000	479.100006	0.00	8.460000	95.779999
...	...	...	...	...	...	...	...	...	...	...	...
49995	2003-10-13 15:00:00	0.00	0.59	0.00	0.0	0.00	47.070000	95.970001	0.00	17.620001	33.619999
49996	2003-10-13 15:00:00	0.00	0.30	0.00	0.0	0.00	64.330002	153.399994	0.00	5.250000	45.980000
49997	2003-10-13 15:00:00	2.92	0.71	0.00	0.0	0.00	62.869999	184.600006	0.00	3.840000	57.740002
49998	2003-10-13 15:00:00	0.00	0.47	0.00	0.0	0.09	59.220001	108.900002	0.00	22.260000	36.439999
49999	2003-10-13 15:00:00	0.68	0.24	2.19	3.8	0.07	61.720001	94.680000	1.78	17.500000	31.440001

50000 rows × 16 columns

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

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

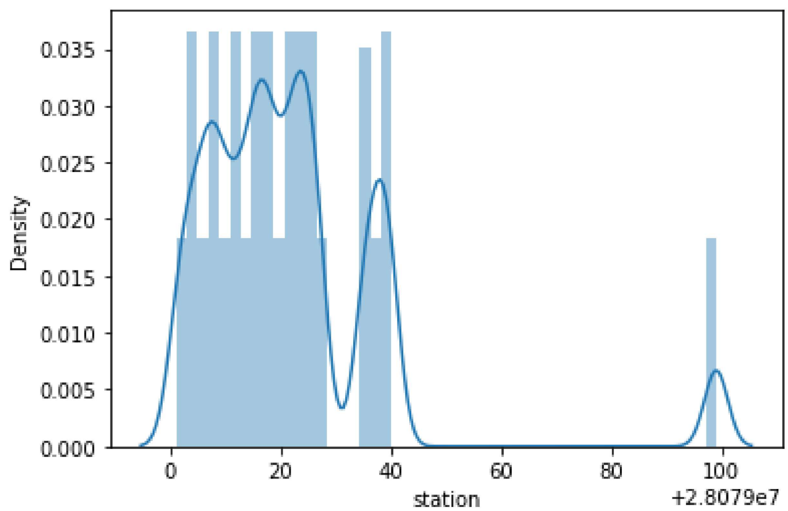
In [8]:
```
sns.pairplot(df)
```
Out[8]: <seaborn.axisgrid.PairGrid at 0x1cce78efbe0>



In [9]:
```python
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 [12]:
```python
df1=df[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
```

In [13]:
```python
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO_2', 'NOx', 'OXY','PM10', 'PXY', 'SO_2', 'T
y=df1[['station']]
```

In [14]:
```
#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 [15]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

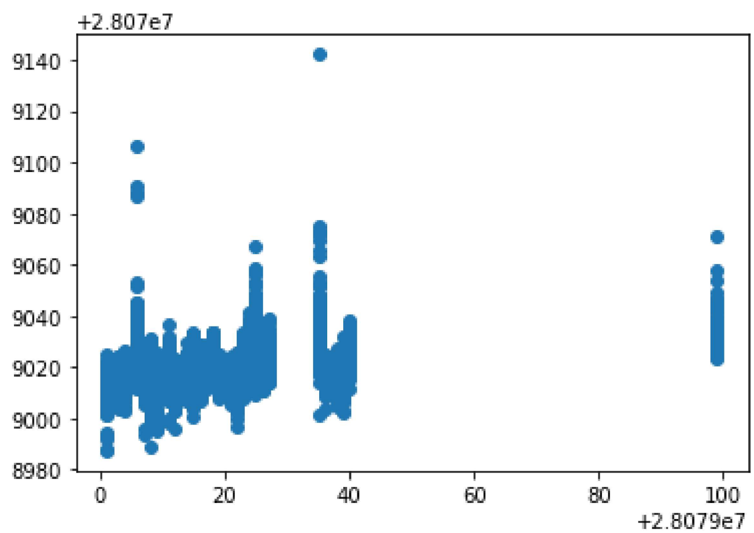
Out[15]: LinearRegression()

In [16]:
```
print(lr.intercept_)
```

```
[28079022.55462194]
```

In [17]:
```
prediction = lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[17]: <matplotlib.collections.PathCollection at 0x1ccff84a400>



In [18]:
```
print(lr.score(x_test,y_test))
```

```
0.11520628844535274
```

2.Ridge Regression

In [19]:
```
from sklearn.linear_model import Ridge
```

In [20]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[20]: Ridge(alpha=10)

In [21]:
```
rr.score(x_test,y_test)
```

Out[21]:
```
0.11519355940988374
```

3.Lasso Regression

In [22]:
```
from sklearn.linear_model import Lasso
```

In [23]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[23]:
```
Lasso(alpha=10)
```

In [24]:
```
la.score(x_test,y_test)
```

Out[24]:
```
0.03857460583675565
```

4.ElasticNet Regression

In [25]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[25]:
```
ElasticNet()
```

In [26]:
```
print(en.coef_)
```

```
[ 0.03038465 -0.          0.          0.         -0.04082635 -0.01512202
  0.04592795  0.07932249  1.75823636 -0.33005828  1.74290323  0.19960044
 -0.00718733]
```

In [27]:
```
print(en.predict(x_test))
```

```
[28079020.24532942 28079031.53919696 28079015.95681932 ...
 28079012.16383981 28079021.83197869 28079008.79068541]
```

In [28]:
```
print(en.score(x_test,y_test))
```

```
0.08515302878922548
```

5.Logistic Regression

In [29]:
```
from sklearn.linear_model import LogisticRegression
```

In [30]:
```
feature_matrix = df1.iloc[:,0:16]
target_vector = df1.iloc[:,-1]
```

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

Out[31]:
```
(50000, 15)
```

In [32]:
```
target_vector.shape
```

Out[32]:
```
(50000,)
```

In [33]:
```
from sklearn.preprocessing import StandardScaler
```

In [34]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [35]:
```
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[35]:
```
LogisticRegression()
```

In [36]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]]
```

In [37]:
```
prediction=logr.predict(observation)
print(prediction)
```

```
[28079099]
```

In [38]:
```
logr.classes_
```

Out[38]:
```
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 [39]:
```
logr.score(fs,target_vector)
```

Out[39]:
```
0.9097
```

6.Random Forest

In [38]:
```
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 [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]:
```
0.5368232977237609
```

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]'),
 Text(2418.0, 543.5999999999999, 'TCH <= 1.405\ngini = 0.41\nsamples = 723\nv
alue = [0, 0, 0, 0, 0, 0, 0, 0, 0, 323, 0, 0, 0, 0\n0, 817, 0, 0, 4, 0, 0,
0, 0, 0]'),
 Text(2325.0, 181.19999999999982, 'gini = 0.174\nsamples = 362\nvalue = [0,
0, 0, 0, 0, 0, 0, 0, 0, 52, 0, 0, 0, 0\n0, 0, 529, 0, 0, 4, 0, 0, 0, 0, 0]'),
 Text(2511.0, 181.19999999999982, 'gini = 0.5\nsamples = 361\nvalue = [0, 0,
0, 0, 0, 0, 0, 0, 0, 271, 0, 0, 0, 0\n0, 0, 288, 0, 0, 0, 0, 0, 0, 0, 0]'),
 Text(3557.25, 1630.8000000000002, 'TOL <= 16.145\ngini = 0.8\nsamples = 6161
\nvalue = [0, 0, 0, 2051, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 1868, 1959,
1971, 0, 0, 0, 0, 1933]'),
 Text(3115.5, 1268.4, 'TCH <= 0.53\ngini = 0.79\nsamples = 5022\nvalue = [0,
0, 0, 928, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 1849, 1816, 1567, 0, 0, 0,
0, 1820]'),
 Text(2883.0, 906.0, 'SO_2 <= 9.765\ngini = 0.002\nsamples = 1128\nvalue =
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 1, 1816, 1, 0, 0, 0, 0,
0]'),
 Text(2790.0, 543.5999999999999, 'SO_2 <= 9.52\ngini = 0.007\nsamples = 375\n
value = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 1, 604, 1, 0, 0,
0, 0, 0]'),
```

Results

1.Linear regression : 0.11520628844535274

2.Ridge regression : 0.11519355940988374

3.Lasso regression : 0.03857460583675565

4.Elasticnet regression : 0.08515302878922548

5.Logistic regresssion : 0.9097

6.Random forest regression : 0.5368232977237609

Hence Logistic regression gives high accuracy for the madrid_2004 model.