In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as py
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
df=pd.read_csv(r"D:\New folder\madrid_2001.csv")
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	I
0	2001-08-01 01:00:00	NaN	0.37	NaN	NaN	NaN	58.400002	87.150002	NaN	34.529999	105.0(
1	2001-08-01 01:00:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.5(
2	2001-08-01 01:00:00	NaN	0.28	NaN	NaN	NaN	50.660000	61.380001	NaN	46.310001	100.0(
3	2001-08-01 01:00:00	NaN	0.47	NaN	NaN	NaN	69.790001	73.449997	NaN	40.650002	69.7
4	2001-08-01 01:00:00	NaN	0.39	NaN	NaN	NaN	22.830000	24.799999	NaN	66.309998	75.1
...	...	...	...	...	...	...	...	...	...	...	
217867	2001-04-01 00:00:00	10.45	1.81	NaN	NaN	NaN	73.000000	264.399994	NaN	5.200000	47.8
217868	2001-04-01 00:00:00	5.20	0.69	4.56	NaN	0.13	71.080002	129.300003	NaN	13.460000	26.8(
217869	2001-04-01 00:00:00	0.49	1.09	NaN	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.7
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	NaN	80.019997	197.000000	2.58	5.840000	37.8
217871	2001-04-01 00:00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.3

217872 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217872 entries, 0 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     217872 non-null  object 
 1   BEN      70389 non-null   float64
 2   CO       216341 non-null  float64
 3   EBE      57752 non-null   float64
 4   MXY      42753 non-null   float64
 5   NMHC     85719 non-null   float64
 6   NO_2     216331 non-null  float64
 7   NOx      216318 non-null  float64
 8   OXY      42856 non-null   float64
 9   O_3      216514 non-null  float64
 10  PM10     207776 non-null  float64
 11  PXY      42845 non-null   float64
 12  SO_2     216403 non-null  float64
 13  TCH      85797 non-null   float64
 14  TOL      70196 non-null   float64
 15  station  217872 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 26.6+ MB
```

```
In [5]: df1=df.fillna(value=0)
        df1
```

Out[5]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2001-08-01 01:00:00	0.00	0.37	0.00	0.00	0.00	58.400002	87.150002	0.00	34.529999	105.0
1	2001-08-01 01:00:00	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.5
2	2001-08-01 01:00:00	0.00	0.28	0.00	0.00	0.00	50.660000	61.380001	0.00	46.310001	100.0
3	2001-08-01 01:00:00	0.00	0.47	0.00	0.00	0.00	69.790001	73.449997	0.00	40.650002	69.7
4	2001-08-01 01:00:00	0.00	0.39	0.00	0.00	0.00	22.830000	24.799999	0.00	66.309998	75.1
...	...	...	...	...	...	...	...	...	...	...	
217867	2001-04-01 00:00:00	10.45	1.81	0.00	0.00	0.00	73.000000	264.399994	0.00	5.200000	47.8
217868	2001-04-01 00:00:00	5.20	0.69	4.56	0.00	0.13	71.080002	129.300003	0.00	13.460000	26.8
217869	2001-04-01 00:00:00	0.49	1.09	0.00	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.7
217870	2001-04-01 00:00:00	5.62	1.01	5.04	11.38	0.00	80.019997	197.000000	2.58	5.840000	37.8
217871	2001-04-01 00:00:00	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.3

217872 rows × 16 columns

In [6]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217872 entries, 0 to 217871
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     217872 non-null  object 
 1   BEN      217872 non-null  float64
 2   CO       217872 non-null  float64
 3   EBE      217872 non-null  float64
 4   MXY      217872 non-null  float64
 5   NMHC     217872 non-null  float64
 6   NO_2     217872 non-null  float64
 7   NOx      217872 non-null  float64
 8   OXY      217872 non-null  float64
 9   O_3      217872 non-null  float64
 10  PM10     217872 non-null  float64
 11  PXY      217872 non-null  float64
 12  SO_2     217872 non-null  float64
 13  TCH      217872 non-null  float64
 14  TOL      217872 non-null  float64
 15  station  217872 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 26.6+ MB
```

In [7]:
```
df1.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]:

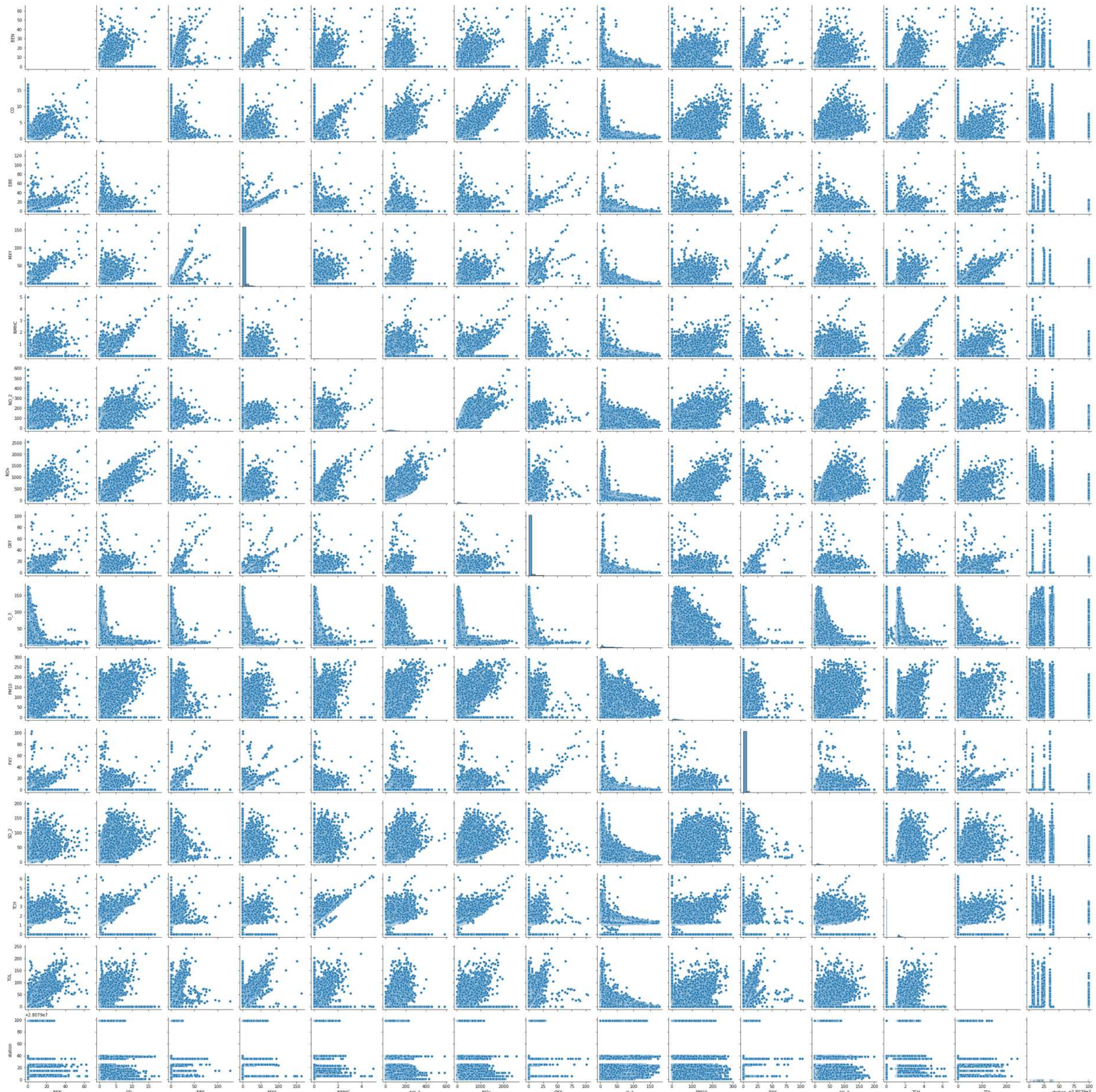
```
df2=df1[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
df2
```

0	0.00	0.37	0.00	0.00	0.00	58.400002	87.150002	0.00	34.529999	105.000000
1	1.50	0.34	1.49	4.10	0.07	56.250000	75.169998	2.11	42.160000	100.599998
2	0.00	0.28	0.00	0.00	0.00	50.660000	61.380001	0.00	46.310001	100.099998
3	0.00	0.47	0.00	0.00	0.00	69.790001	73.449997	0.00	40.650002	69.779999
4	0.00	0.39	0.00	0.00	0.00	22.830000	24.799999	0.00	66.309998	75.180000
...	...	...	...	...	...	...	...	...	...	...
217867	10.45	1.81	0.00	0.00	0.00	73.000000	264.399994	0.00	5.200000	47.880001
217868	5.20	0.69	4.56	0.00	0.13	71.080002	129.300003	0.00	13.460000	26.809999
217869	0.49	1.09	0.00	1.00	0.19	76.279999	128.399994	0.35	5.020000	40.770000
217870	5.62	1.01	5.04	11.38	0.00	80.019997	197.000000	2.58	5.840000	37.889999
217871	8.09	1.62	6.66	13.04	0.18	76.809998	206.300003	5.20	8.340000	35.369999

217872 rows × 15 columns

In [8]:
```
sns.pairplot(df2)
```

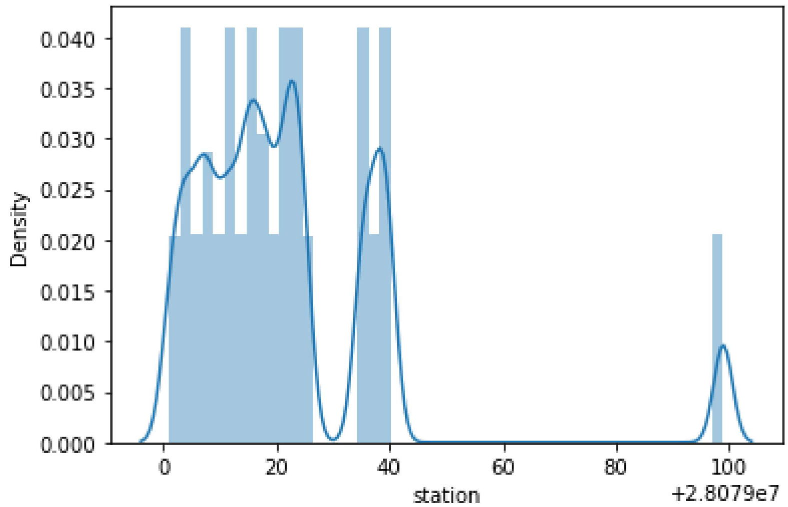
Out[8]: <seaborn.axisgrid.PairGrid at 0x284000690d0>



In [9]:
```
sns.distplot(df2['station'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]:
```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



In [10]:
```
x=df2[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'O_3','PM10', 'PXY', '
y=df2['station']
```

In [13]:
```
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)
```

linear

In [12]:
```
from sklearn.linear_model import LinearRegression
```

In [13]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

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

In [14]:
```
coeff =pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[14]:

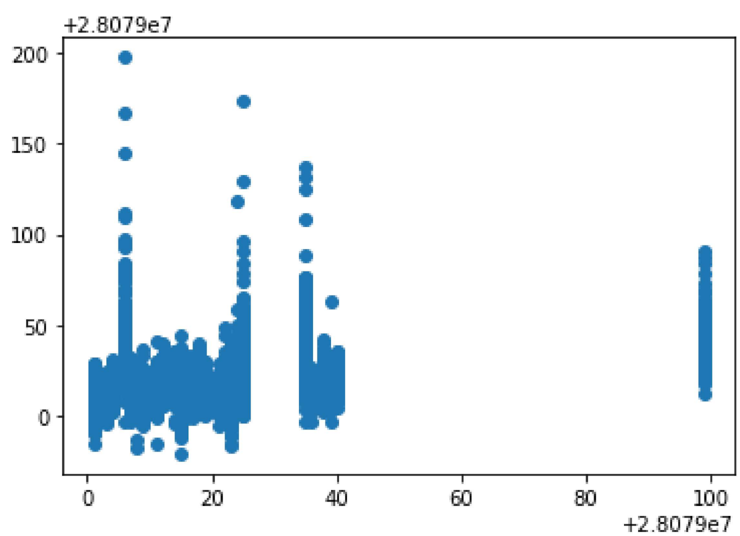
	Co-efficient
BEN	1.178125
CO	4.729986
EBE	-0.482808
MXY	1.204062
NMHC	-12.278957
NO_2	-0.014289
NOx	-0.037306
O_3	0.016504
PM10	0.017617
PXY	1.238319
SO_2	-0.093601
TCH	5.805968
TOL	-0.209488
OXY	-1.115316

In [15]:
```
print(lr.intercept_)
```

```
28079020.607324712
```

In [16]:
```
prediction =lr.predict(x_test)
py.scatter(y_test,prediction)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x284135414c0>
```



In [17]:
```
print(lr.score(x_test,y_test))
```

```
0.09946538993934295
```

In [18]:
```
print(lr.score(x_train,y_train))
```

```
0.10697649345555793
```

Ridge

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

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]:
```
0.09945090633562192
```

Lasso

In [22]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[22]:
```
Lasso(alpha=10)
```

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

Out[23]:
```
0.04613897441016113
```

elasticnet

In [24]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[24]:
```
ElasticNet()
```

In [25]:
```
print(en.coef_)
```

```
[ 0.26761416  0.4274753  -0.          0.80447773  0.         -0.03452848
 -0.01596133  0.01056008  0.02986053  0.41286935 -0.06180386  1.35445825
 -0.01676271  0.        ]
```

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

```
28079023.58901875
```

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

```
[28079020.87200194 28079020.24760915 28079018.99952377 ...
 28079019.76714293 28079024.00669158 28079035.30050029]
```

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

```
0.07787911689213145
```

logistic

In [29]:
```
feature_matrix=df2.iloc[:,0:14]
target_vector=df2.iloc[:,-1]
```

In [30]:
```
feature_matrix=df2[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'O_3','PM
y=df2['station']
```

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

Out[31]:
```
(217872, 14)
```

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

Out[32]:
```
(217872,)
```

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:
763: 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://sciki
t-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-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

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

In [36]:
```
observation=[[1.4,2.3,5.0,11,12,13,14,15,4,5,7,6,7,13]]
```

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, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079099], dtype=int64)
```

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

Out[39]:
```
0.9102362855254461
```

In [40]:
```
logr.predict_proba(observation)[0][0]
```

Out[40]:
```
0.0
```

In [41]:
```
logr.predict_proba(observation)[0][1]
```

Out[41]:
```
0.0
```

random forest

In [9]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [10]:
```
x=df2.drop('station',axis=1)
y=df2['station']
```

In [14]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.70)
```

In [15]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[15]:
```
RandomForestClassifier()
```

In [16]:
```
parameters = {'max_depth':[1,2,3,4,5],
             'min_samples_leaf':[5,10,15,20,25],
              'n_estimators':[10,20,30,40,50]}
```

In [17]:
```
from sklearn.model_selection import GridSearchCV
```

In [18]:
```
grid_search = GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring='ac
grid_search.fit(x_train,y_train)
```

Out[18]:
```
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 [19]:
```
grid_search.best_score_
```

Out[19]:
```
0.4717951482482813
```

In [20]:
```
rfc_best =grid_search.best_estimator_
```

In [21]:
```
py.figure(figsize=(80,50))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

Out[21]:
```
[Text(2317.846153846154, 2491.5, 'PXY <= 0.08\ngini = 0.961\nsamples = 4142
1\nvalue = [2590, 2643, 2599, 2601, 2382, 1384, 2596, 2545, 2662\n2590, 259
6, 2544, 1437, 2440, 2627, 2565, 2575, 2706\n2633, 2618, 2740, 2668, 2544,
2752, 2685, 2639]'),
 Text(1373.5384615384614, 2038.5, 'BEN <= 0.095\ngini = 0.952\nsamples = 33
205\nvalue = [2590, 2643, 2599, 28, 2382, 1384, 2596, 2545, 2662\n2590, 259
6, 2544, 1437, 2440, 2627, 2565, 2575, 2706\n66, 72, 61, 2668, 2544, 2752,
2685, 17]'),
 Text(686.7692307692307, 1585.5, 'TCH <= 0.1\ngini = 0.943\nsamples = 28006
\nvalue = [2590, 2643, 2599, 24, 2382, 44, 2596, 2545, 2662\n2590, 37, 254
4, 1437, 2440, 2627, 2565, 105, 822, 66\n72, 61, 2668, 2544, 2752, 2685, 1
0]'),
 Text(343.38461538461536, 1132.5, 'NOx <= 144.45\ngini = 0.926\nsamples = 2
0920\nvalue = [2590, 2643, 2599, 9, 99, 44, 2596, 1361, 75, 2590\n26, 2544,
1437, 18, 2627, 2565, 105, 1, 25, 72, 1\n2668, 2544, 2752, 882, 5]'),
 Text(171.69230769230768, 679.5, 'CO <= 0.905\ngini = 0.924\nsamples = 1423
2\nvalue = [834, 2027, 1682, 9, 97, 28, 1165, 1124, 66, 1619\n22, 2173, 73
8, 18, 1730, 1928, 48, 1, 25, 67, 1\n2155, 1778, 2179, 807, 5]'),
 Text(85.84615384615384, 226.5, 'gini = 0.923\nsamples = 12929\nvalue = [81
```

conclusion

The bestfit model is Logistic Regression with score of 0.9102362855254461

In [ ]: