In [52]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as py
import seaborn as sns
from sklearn.linear_model import LogisticRegression
```

In [53]:
```python
df=pd.read_csv(r"D:\New folder\madrid_2003.csv")
df
```

Out[53]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	NaN	1.72	NaN	NaN	NaN	73.900002	316.299988	NaN	10.550000	55.2099
1	2003-03-01 01:00:00	NaN	1.45	NaN	NaN	0.26	72.110001	250.000000	0.73	6.720000	52.3899
2	2003-03-01 01:00:00	NaN	1.57	NaN	NaN	NaN	80.559998	224.199997	NaN	21.049999	63.2400
3	2003-03-01 01:00:00	NaN	2.45	NaN	NaN	NaN	78.370003	450.399994	NaN	4.220000	67.8399
4	2003-03-01 01:00:00	NaN	3.26	NaN	NaN	NaN	96.250000	479.100006	NaN	8.460000	95.7799
...	...	...	...	...	...	...	...	...	...	...	
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	7.3800
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	NaN	10.450000	14.760000	1.00	34.610001	7.4000
243981	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	34.639999	50.810001	NaN	32.160000	16.8300
243982	2003-10-01 00:00:00	NaN	NaN	NaN	NaN	0.07	32.580002	41.020000	NaN	NaN	13.5700
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	12.3500

243984 rows × 16 columns

In [3]:
```
df.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 [4]:
```
df1=df.fillna(value=0)
df1
```

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM
0	2003-03-01 01:00:00	0.00	1.72	0.00	0.00	0.00	73.900002	316.299988	0.00	10.550000	55.2099
1	2003-03-01 01:00:00	0.00	1.45	0.00	0.00	0.26	72.110001	250.000000	0.73	6.720000	52.3899
2	2003-03-01 01:00:00	0.00	1.57	0.00	0.00	0.00	80.559998	224.199997	0.00	21.049999	63.2400
3	2003-03-01 01:00:00	0.00	2.45	0.00	0.00	0.00	78.370003	450.399994	0.00	4.220000	67.8399
4	2003-03-01 01:00:00	0.00	3.26	0.00	0.00	0.00	96.250000	479.100006	0.00	8.460000	95.7799
...	...	...	...	...	...	...	...	...	...	...	
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	7.3800
243980	2003-10-01 00:00:00	0.32	0.08	0.36	0.72	0.00	10.450000	14.760000	1.00	34.610001	7.4000
243981	2003-10-01 00:00:00	0.00	0.00	0.00	0.00	0.07	34.639999	50.810001	0.00	32.160000	16.8300
243982	2003-10-01 00:00:00	0.00	0.00	0.00	0.00	0.07	32.580002	41.020000	0.00	0.000000	13.5700
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	12.3500

243984 rows × 16 columns

In [5]:
```
df1.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      243984 non-null  float64
 2   CO       243984 non-null  float64
 3   EBE      243984 non-null  float64
 4   MXY      243984 non-null  float64
 5   NMHC     243984 non-null  float64
 6   NO_2     243984 non-null  float64
 7   NOx      243984 non-null  float64
 8   OXY      243984 non-null  float64
 9   O_3      243984 non-null  float64
 10  PM10     243984 non-null  float64
 11  PXY      243984 non-null  float64
 12  SO_2     243984 non-null  float64
 13  TCH      243984 non-null  float64
 14  TOL      243984 non-null  float64
 15  station  243984 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 29.8+ MB
```

In [6]:
```
df1.columns
```

Out[6]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

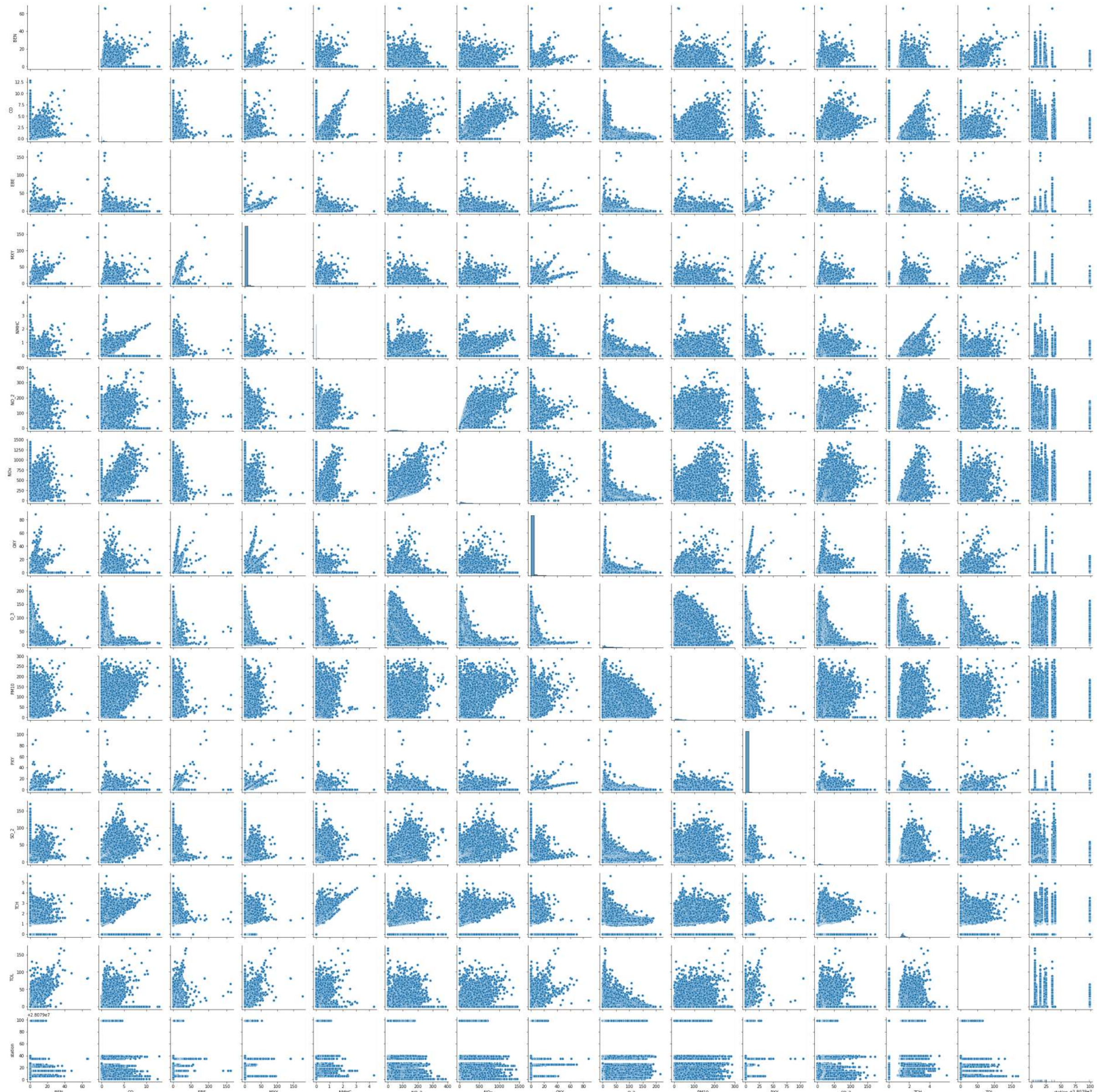
In [7]:
```
df2=df1[['BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
df2
```

Out[7]:

	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10	PX
0	0.00	1.72	0.00	0.00	0.00	73.900002	316.299988	0.00	10.550000	55.209999	0.0
1	0.00	1.45	0.00	0.00	0.26	72.110001	250.000000	0.73	6.720000	52.389999	0.0
2	0.00	1.57	0.00	0.00	0.00	80.559998	224.199997	0.00	21.049999	63.240002	0.0
3	0.00	2.45	0.00	0.00	0.00	78.370003	450.399994	0.00	4.220000	67.839996	0.0
4	0.00	3.26	0.00	0.00	0.00	96.250000	479.100006	0.00	8.460000	95.779999	0.0
...	...	...	...	...	...	...	...	...	...	...	.
243979	0.20	0.16	2.01	3.17	0.02	31.799999	32.299999	1.68	34.049999	7.380000	1.2
243980	0.32	0.08	0.36	0.72	0.00	10.450000	14.760000	1.00	34.610001	7.400000	0.5
243981	0.00	0.00	0.00	0.00	0.07	34.639999	50.810001	0.00	32.160000	16.830000	0.0
243982	0.00	0.00	0.00	0.00	0.07	32.580002	41.020000	0.00	0.000000	13.570000	0.0
243983	1.00	0.29	2.15	6.41	0.07	37.150002	56.849998	2.28	21.480000	12.350000	2.4

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

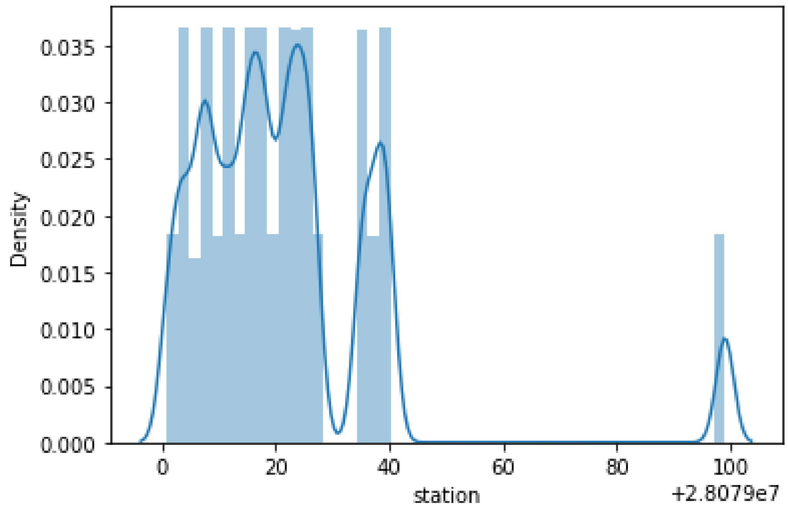
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x26b0892beb0>
```



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 [11]:
```
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]:
```python
coeff =pd.DataFrame(lr.coef_,x.columns,columns=["Co-efficient"])
coeff
```

Out[14]:

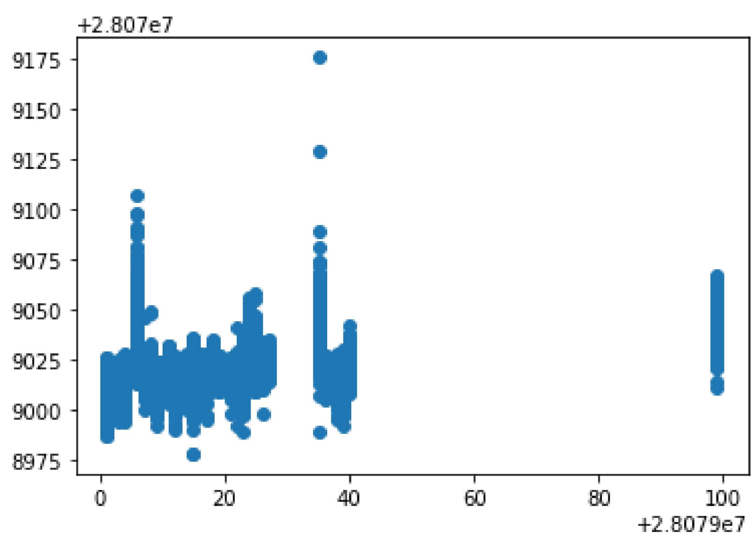
	Co-efficient
BEN	1.329314
CO	0.078055
EBE	-1.103965
MXY	0.395101
NMHC	0.744723
NO_2	-0.079070
NOx	-0.008685
O_3	-0.019030
PM10	0.043117
PXY	2.733720
SO_2	-0.156103
TCH	4.654519
TOL	-0.231877
OXY	0.262578

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

```
28079024.492372576
```

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

Out[16]:
```
<matplotlib.collections.PathCollection at 0x26b318a5280>
```



In [17]:
```python
print(lr.score(x_test,y_test))
```
```
0.12427344825624764
```

In [18]:
```python
print(lr.score(x_train,y_train))
```
```
0.11829138820457574
```

Ridge

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

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

In [21]:
```python
rr.score(x_test,y_test)
```
Out[21]:
```
0.1242719430594611
```

Lasso

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

In [23]:
```python
la.score(x_test,y_test)
```
Out[23]:
```
0.040051613031118816
```

elasticnet

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

In [25]:
```python
print(en.coef_)
```
```
[ 0.13012942 -0.         -0.          0.89170003  0.         -0.07099582
 -0.00816406 -0.01816097  0.04245185  0.56079423 -0.17328773  1.78537876
  0.          0.17031933]
```

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

```
28079025.995923337
```

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

```
[28079027.46757036 28079021.1167562  28079022.89446069 ...
 28079025.9103636  28079025.96325697 28079027.48770221]
```

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

```
0.10087740966978953
```

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]:
```
(243984, 14)
```

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

Out[32]:
```
(243984,)
```

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, 28079026, 28079027, 28079035, 28079036,
       28079038, 28079039, 28079040, 28079099], dtype=int64)
```

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

Out[39]:
```
0.9353441209259623
```

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 [42]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [43]:
```
x=df2.drop('station',axis=1)
y=df2['station']
```

In [44]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.70)
```

In [45]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[45]:
```
RandomForestClassifier()
```

In [46]:
```
parameters = {'max_depth':[1,2,3,4,5],
             'min_samples_leaf':[5,10,15,20,25],
              'n_estimators':[10,20,30,40,50]}
```

In [47]:
```
from sklearn.model_selection import GridSearchCV
```

In [48]:
```
grid_search = GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring='ac
grid_search.fit(x_train,y_train)
```
Out[48]:
```
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 [49]:
```
grid_search.best_score_
```
Out[49]:
```
0.48788852882546796
```
In [50]:
```
rfc_best =grid_search.best_estimator_
```
In [51]:
```
py.figure(figsize=(80,50))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```
Out[51]:
```
[Text(1985.1923076923076, 2491.5, 'TCH <= 0.44\ngini = 0.964\nsamples = 463
66\nvalue = [2648, 2630, 2721, 2473, 2671, 2621, 2667, 2643, 2630\n2476, 26
63, 2622, 2598, 2642, 2588, 2604, 2765, 2565\n2608, 2524, 2684, 2434, 2668,
2494, 2678, 2617, 2641\n2620]'),
 Text(815.5384615384614, 2038.5, 'BEN <= 0.1\ngini = 0.934\nsamples = 25103
\nvalue = [2648, 2630, 2721, 41, 16, 6, 2667, 1, 2630, 2476\n25, 2622, 259
8, 5, 2588, 2604, 2765, 14, 32, 2524\n17, 19, 5, 2494, 2678, 2617, 24,
4]'),
 Text(257.53846153846155, 1585.5, 'CO <= 0.665\ngini = 0.924\nsamples = 217
88\nvalue = [2648, 2630, 2721, 37, 16, 1, 2667, 1, 2630, 2476\n9, 2622, 259
8, 5, 2588, 2604, 78, 9, 17, 19, 17\n19, 3, 2494, 2678, 2617, 24, 4]'),
 Text(171.69230769230768, 1132.5, 'gini = 0.919\nsamples = 13298\nvalue =
[619, 1694, 1820, 37, 16, 0, 717, 1, 1526, 1270, 9\n2120, 1909, 2, 1681, 19
68, 47, 9, 17, 18, 17, 19\n3, 1865, 1585, 1939, 15, 4]'),
 Text(343.38461538461536, 1132.5, 'PM10 <= 0.855\ngini = 0.908\nsamples = 8
490\nvalue = [2029, 936, 901, 0, 0, 1, 1950, 0, 1104, 1206, 0\n502, 689, 3,
907, 636, 31, 0, 0, 1, 0, 0, 0\n629, 1093, 678, 9, 0]'),
 Text(171.69230769230768, 679.5, 'NO_2 <= 88.885\ngini = 0.137\nsamples = 1
51\nvalue = [5, 2, 0, 0, 0, 0, 4, 0, 0, 3, 0, 0, 1, 1\n0, 0, 0, 0, 0, 0, 0,
```
conclusion ¶

The bestfit model is Logistic Regression with score of 0.9353441209259623

In [ ]: