In [45]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [46]:
```
data=pd.read_csv(r"C:\Users\user\Downloads\madrid_2012.csv")
data
```

Out[46]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	7.0	18.0	NaN	NaN	NaN	2.0	NaN	NaN	2
1	2012-09-01 01:00:00	0.3	0.3	0.7	NaN	3.0	18.0	55.0	10.0	9.0	1.0	NaN	2.4	2
2	2012-09-01 01:00:00	0.4	NaN	0.7	NaN	2.0	10.0	NaN	NaN	NaN	NaN	NaN	1.5	2
3	2012-09-01 01:00:00	NaN	0.2	NaN	NaN	1.0	6.0	50.0	NaN	NaN	NaN	NaN	NaN	2
4	2012-09-01 01:00:00	NaN	NaN	NaN	NaN	1.0	13.0	54.0	NaN	NaN	3.0	NaN	NaN	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210715	2012-03-01 00:00:00	NaN	0.6	NaN	NaN	37.0	84.0	14.0	NaN	NaN	NaN	NaN	NaN	2
210716	2012-03-01 00:00:00	NaN	0.4	NaN	NaN	5.0	76.0	NaN	17.0	NaN	7.0	NaN	NaN	2
210717	2012-03-01 00:00:00	NaN	NaN	NaN	0.34	3.0	41.0	24.0	NaN	NaN	NaN	1.34	NaN	2
210718	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	44.0	36.0	NaN	NaN	NaN	NaN	NaN	2
210719	2012-03-01 00:00:00	NaN	NaN	NaN	NaN	2.0	56.0	40.0	18.0	NaN	NaN	NaN	NaN	2

210720 rows × 14 columns

In [47]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210720 entries, 0 to 210719
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210720 non-null  object 
 1   BEN      51511 non-null   float64
 2   CO       87097 non-null   float64
 3   EBE      51482 non-null   float64
 4   NMHC     30736 non-null   float64
 5   NO       209871 non-null  float64
 6   NO_2     209872 non-null  float64
 7   O_3      122339 non-null  float64
 8   PM10     104838 non-null  float64
 9   PM25     52164 non-null   float64
 10  SO_2     87333 non-null   float64
 11  TCH      30736 non-null   float64
 12  TOL      51373 non-null   float64
 13  station  210720 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.5+ MB
```

In [48]:
```
df=data.fillna(value=0)
df
```

Out[48]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2012-09-01 01:00:00	0.0	0.2	0.0	0.00	7.0	18.0	0.0	0.0	0.0	2.0	0.00	0.0	28
1	2012-09-01 01:00:00	0.3	0.3	0.7	0.00	3.0	18.0	55.0	10.0	9.0	1.0	0.00	2.4	28
2	2012-09-01 01:00:00	0.4	0.0	0.7	0.00	2.0	10.0	0.0	0.0	0.0	0.0	0.00	1.5	28
3	2012-09-01 01:00:00	0.0	0.2	0.0	0.00	1.0	6.0	50.0	0.0	0.0	0.0	0.00	0.0	28
4	2012-09-01 01:00:00	0.0	0.0	0.0	0.00	1.0	13.0	54.0	0.0	0.0	3.0	0.00	0.0	28
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210715	2012-03-01 00:00:00	0.0	0.6	0.0	0.00	37.0	84.0	14.0	0.0	0.0	0.0	0.00	0.0	28
210716	2012-03-01 00:00:00	0.0	0.4	0.0	0.00	5.0	76.0	0.0	17.0	0.0	7.0	0.00	0.0	28
210717	2012-03-01 00:00:00	0.0	0.0	0.0	0.34	3.0	41.0	24.0	0.0	0.0	0.0	1.34	0.0	28
210718	2012-03-01 00:00:00	0.0	0.0	0.0	0.00	2.0	44.0	36.0	0.0	0.0	0.0	0.00	0.0	28
210719	2012-03-01 00:00:00	0.0	0.0	0.0	0.00	2.0	56.0	40.0	18.0	0.0	0.0	0.00	0.0	28

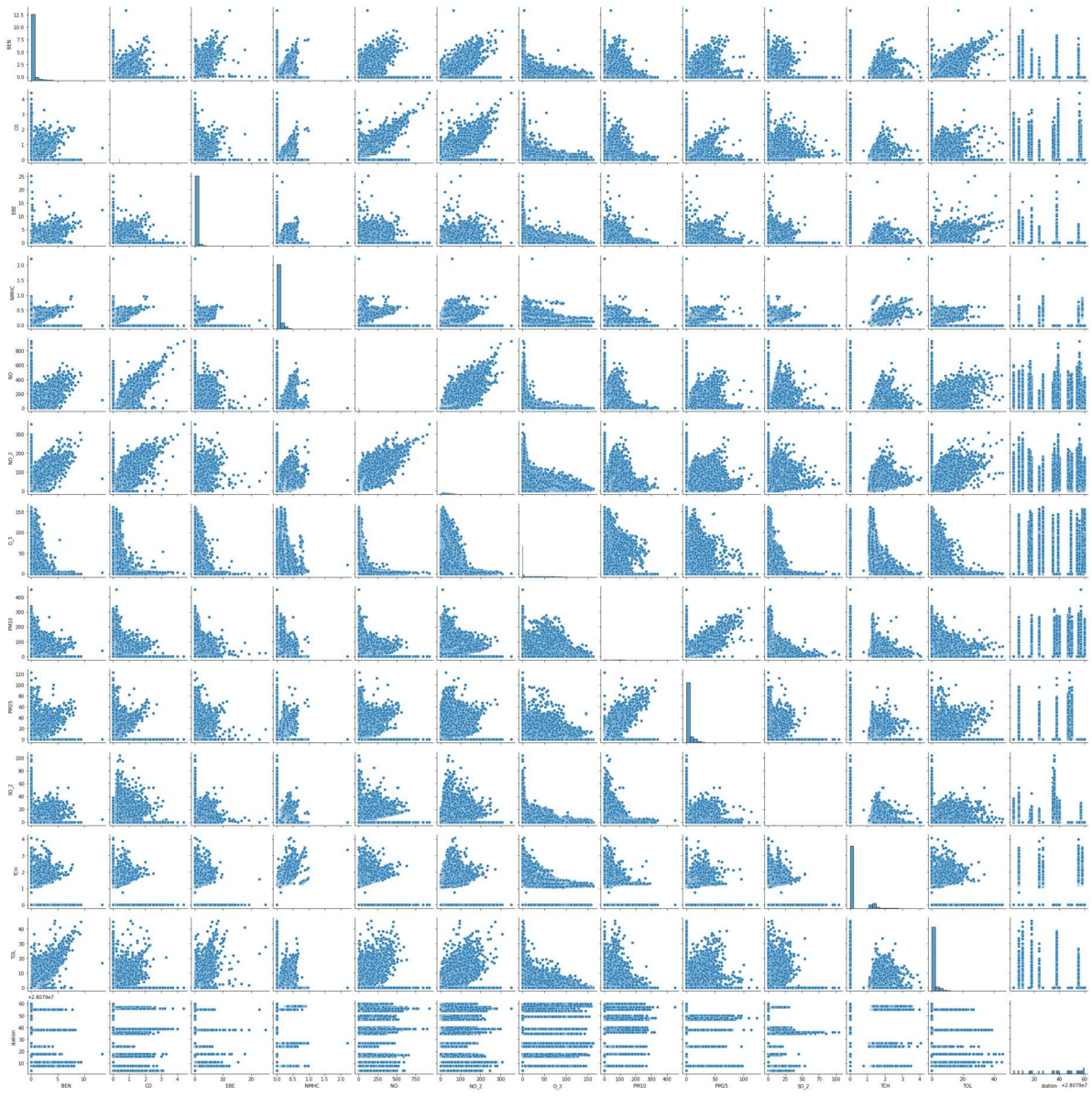
210720 rows × 14 columns

In [49]:
```
df.columns
```

Out[49]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM2
5',
       'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

In [50]:
```
sns.pairplot(df)
```

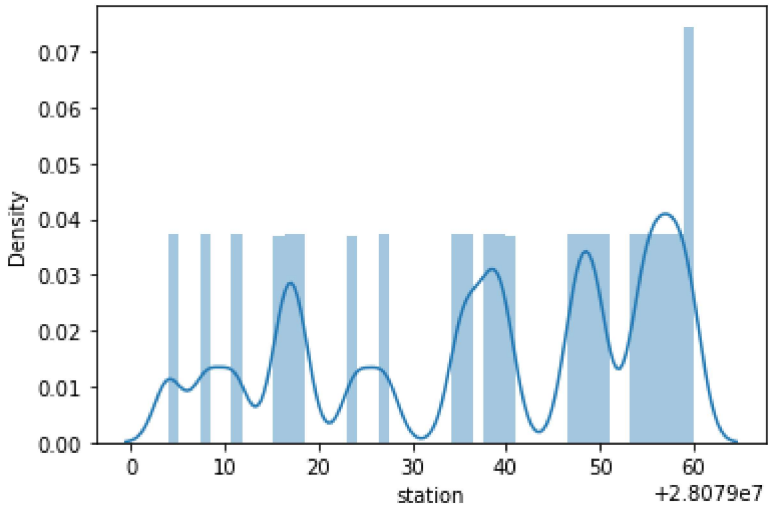
Out[50]:
```
<seaborn.axisgrid.PairGrid at 0x204c3c91f10>
```



In [51]:
```
sns.distplot(data["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[51]:
```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



MODEL BUILDING

Linear Regression

In [53]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
```

In [55]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
y=df1[['station']]
```

In [56]:
```
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 [57]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

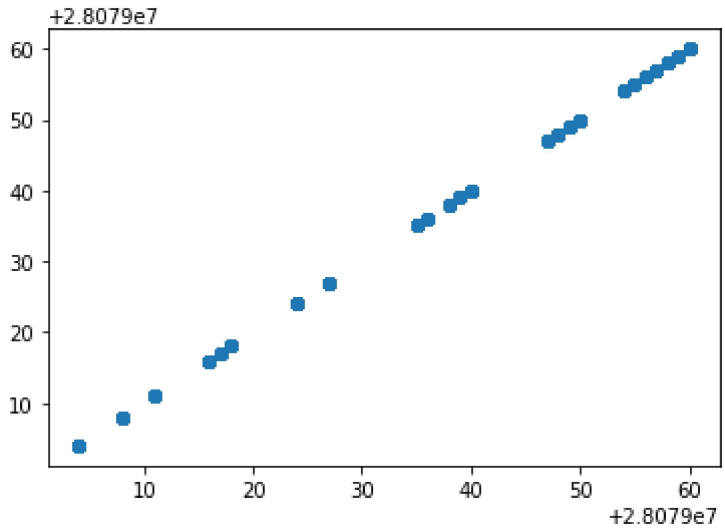
Out[57]:
```
LinearRegression()
```

In [58]:
```
print(lr.intercept_)
```

```
[-2.60770321e-08]
```

In [59]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[59]:
```
<matplotlib.collections.PathCollection at 0x2051e038fd0>
```



In [60]:
```
print(lr.score(x_test,y_test))
```

```
1.0
```

Ridge Regression

In [61]:
```
from sklearn.linear_model import Ridge
```

In [62]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[62]:
```
Ridge(alpha=10)
```

In [63]:
```
rr.score(x_test,y_test)
```

Out[63]:
```
0.9999999999999336
```

Lasso Regression

In [64]:
```
from sklearn.linear_model import Lasso
```

In [65]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[65]:
```
Lasso(alpha=10)
```

In [66]:
```
la.score(x_test,y_test)
```

Out[66]:
```
0.9989655602035696
```

Elastic Regression

In [67]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[67]:
```
ElasticNet()
```

In [68]:
```
print(en.coef_)
```

```
[-0.        -0.        -0.        -0.        -0.        -0.
  0.         0.        -0.        -0.        -0.        -0.
  0.9967889]
```

In [69]:
```
print(en.predict(x_test))
```

```
[28079017.06662801 28079053.94781713 28079039.9927726  ...
 28079048.96387266 28079038.9959837  28079039.9927726 ]
```

In [70]:
```
print(en.score(x_test,y_test))
```

```
0.9999896887923364
```

Logistic Regression

In [71]:
```
from sklearn.linear_model import LogisticRegression
```

In [72]:
```
feature_matrix=df1.iloc[:,0:14]
target_vector=df1.iloc[:,-1]
```

In [73]:
```
feature_matrix.shape
```

Out[73]:
```
(210720, 13)
```

In [74]:
```
target_vector.shape
```

Out[74]:
```
(210720,)
```

In [75]:
```
from sklearn.preprocessing import StandardScaler
```

In [76]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [77]:
```
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[77]:
```
LogisticRegression()
```

In [78]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13]]
```

In [79]:
```
prediction=logr.predict(observation)
print(observation)
```

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]]
```

In [80]:
```
logr.classes_
```

Out[80]:
```
array([28079004, 28079008, 28079011, 28079016, 28079017, 28079018,
       28079024, 28079027, 28079035, 28079036, 28079038, 28079039,
       28079040, 28079047, 28079048, 28079049, 28079050, 28079054,
       28079055, 28079056, 28079057, 28079058, 28079059, 28079060],
      dtype=int64)
```

In [81]:
```
logr.score(fs,target_vector)
```

Out[81]:
```
0.9869922171602126
```

Random Forest

In [82]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [84]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df1['station']
```

In [85]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.70)
```

In [86]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[86]:
```
RandomForestClassifier()
```

In [87]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]}
```

In [88]:
```
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[88]:
```
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 [89]:
```
grid_search.best_score_
```

Out[89]:
```
0.7466464186281954
```

In [90]:
```
rfc_best=grid_search.best_estimator_
```

In [91]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

```
samples = 5271\nvalue = [1767, 0, 0, 1695, 0, 7, 0, 0, 1568, 5, 0, 1138\n
0, 0, 0, 0, 0, 0, 0, 2152, 6, 0, 0, 0]'),
 Text(967.2, 181.19999999999982, 'gini = 0.743\nsamples = 1110\nvalue = [3
13, 0, 0, 616, 0, 4, 0, 0, 214, 0, 0, 96, 0\n0, 0, 0, 0, 0, 0, 492, 0, 0,
0, 0]'),
 Text(1116.0, 181.19999999999982, 'gini = 0.795\nsamples = 4161\nvalue =
[1454, 0, 0, 1079, 0, 3, 0, 0, 1354, 5, 0, 1042\n0, 0, 0, 0, 0, 0, 0, 166
0, 6, 0, 0, 0]'),
 Text(1674.0000000000002, 1268.4, 'NMHC <= 0.005\ngini = 0.835\nsamples =
10052\nvalue = [0, 11, 0, 0, 0, 2621, 3, 0, 0, 2628, 10, 0\n2571, 30, 13,
0, 21, 0, 2708, 0, 2618, 0, 0\n2683]'),
 Text(1488.0, 906.0, 'BEN <= 0.05\ngini = 0.803\nsamples = 8392\nvalue =
[0, 11, 0, 0, 0, 2621, 0, 0, 0, 2628, 10, 0\n2571, 30, 13, 0, 21, 0, 31,
0, 2618, 0, 0, 2683]'),
 Text(1339.2, 543.5999999999999, 'NO_2 <= 16.5\ngini = 0.755\nsamples = 67
34\nvalue = [0, 0, 0, 0, 0, 37, 0, 0, 0, 2628, 2, 0, 2571\n30, 13, 0, 21,
0, 12, 0, 2618, 0, 0, 2683]'),
 Text(1264.8000000000002, 181.19999999999982, 'gini = 0.742\nsamples = 174
7\nvalue = [0, 0, 0, 0, 0, 8, 0, 0, 0, 497, 0, 0, 607, 9\n5, 0, 8, 0, 7,
0, 694, 0, 0, 947]'),
```

Results

The best model is Elastic Regression 0.9999999999999336

In [ ]: