In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
data=pd.read_csv(r"C:\Users\user\Downloads\madrid_2011.csv")
data
```

Out[2]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL
0	2011-11-01 01:00:00	NaN	1.0	NaN	NaN	154.0	84.0	NaN	NaN	NaN	6.0	NaN	NaN
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7
2	2011-11-01 01:00:00	2.9	NaN	3.8	NaN	96.0	99.0	NaN	NaN	NaN	NaN	NaN	7.2
3	2011-11-01 01:00:00	NaN	0.6	NaN	NaN	60.0	83.0	2.0	NaN	NaN	NaN	NaN	NaN
4	2011-11-01 01:00:00	NaN	NaN	NaN	NaN	44.0	62.0	3.0	NaN	NaN	3.0	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
209923	2011-09-01 00:00:00	NaN	0.2	NaN	NaN	5.0	19.0	44.0	NaN	NaN	NaN	NaN	NaN
209924	2011-09-01 00:00:00	NaN	0.1	NaN	NaN	6.0	29.0	NaN	11.0	NaN	7.0	NaN	NaN
209925	2011-09-01 00:00:00	NaN	NaN	NaN	0.23	1.0	21.0	28.0	NaN	NaN	NaN	1.44	NaN
209926	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	3.0	15.0	48.0	NaN	NaN	NaN	NaN	NaN
209927	2011-09-01 00:00:00	NaN	NaN	NaN	NaN	4.0	33.0	38.0	13.0	NaN	NaN	NaN	NaN

209928 rows × 14 columns

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209928 entries, 0 to 209927
Data columns (total 14 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     209928 non-null  object 
 1   BEN      51393 non-null   float64
 2   CO       87127 non-null   float64
 3   EBE      51350 non-null   float64
 4   NMHC     43517 non-null   float64
 5   NO       208954 non-null  float64
 6   NO_2     208973 non-null  float64
 7   O_3      122049 non-null  float64
 8   PM10     103743 non-null  float64
 9   PM25     51079 non-null   float64
 10  SO_2     87131 non-null   float64
 11  TCH      43519 non-null   float64
 12  TOL      51175 non-null   float64
 13  station  209928 non-null  int64  
dtypes: float64(12), int64(1), object(1)
memory usage: 22.4+ MB
```

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

Out[4]:

	date	BEN	CO	EBE	NMHC	NO	NO_2	O_3	PM10	PM25	SO_2	TCH	TOL	
0	2011-11-01 01:00:00	0.0	1.0	0.0	0.00	154.0	84.0	0.0	0.0	0.0	6.0	0.00	0.0	2
1	2011-11-01 01:00:00	2.5	0.4	3.5	0.26	68.0	92.0	3.0	40.0	24.0	9.0	1.54	8.7	2
2	2011-11-01 01:00:00	2.9	0.0	3.8	0.00	96.0	99.0	0.0	0.0	0.0	0.0	0.00	7.2	2
3	2011-11-01 01:00:00	0.0	0.6	0.0	0.00	60.0	83.0	2.0	0.0	0.0	0.0	0.00	0.0	2
4	2011-11-01 01:00:00	0.0	0.0	0.0	0.00	44.0	62.0	3.0	0.0	0.0	3.0	0.00	0.0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
209923	2011-09-01 00:00:00	0.0	0.2	0.0	0.00	5.0	19.0	44.0	0.0	0.0	0.0	0.00	0.0	2
209924	2011-09-01 00:00:00	0.0	0.1	0.0	0.00	6.0	29.0	0.0	11.0	0.0	7.0	0.00	0.0	2
209925	2011-09-01 00:00:00	0.0	0.0	0.0	0.23	1.0	21.0	28.0	0.0	0.0	0.0	1.44	0.0	2
209926	2011-09-01 00:00:00	0.0	0.0	0.0	0.00	3.0	15.0	48.0	0.0	0.0	0.0	0.00	0.0	2
209927	2011-09-01 00:00:00	0.0	0.0	0.0	0.00	4.0	33.0	38.0	13.0	0.0	0.0	0.00	0.0	2

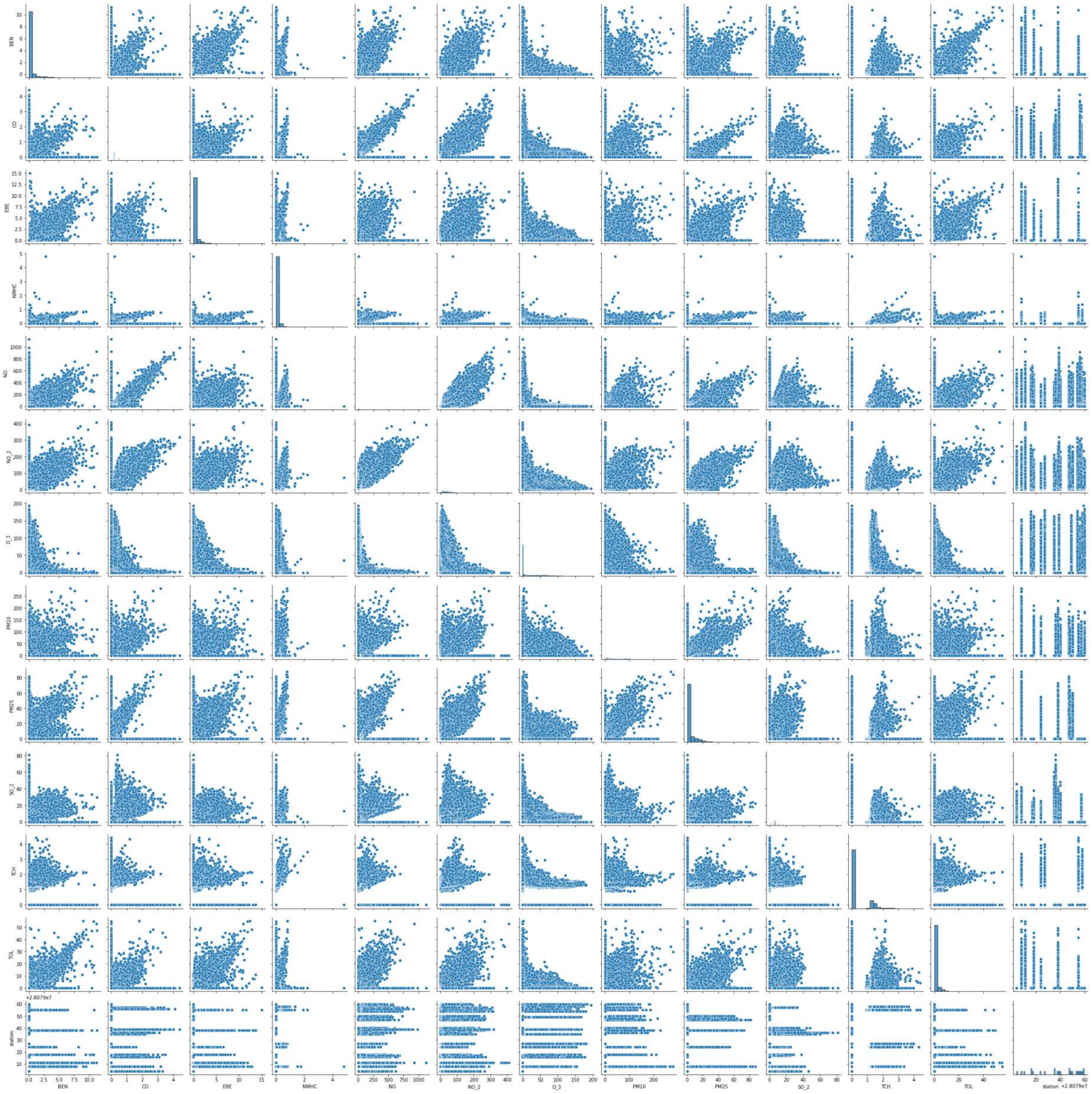
209928 rows × 14 columns

In [5]:
```
df.columns
```

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

```
In [6]: sns.pairplot(df)
```

```
Out[6]: <seaborn.axisgrid.PairGrid at 0x1f809544850>
```



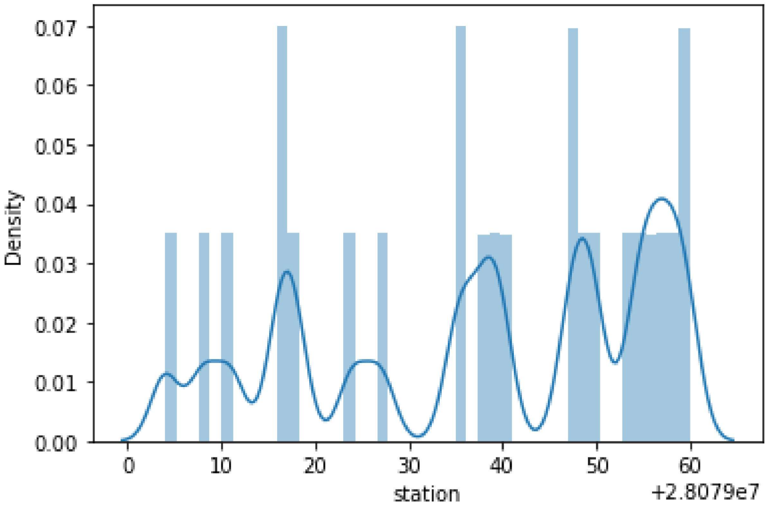
In [8]:

```
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[8]:

```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



MODEL BUILDING

Linear Regression

In [9]:

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

In [10]:

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

In [13]:

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[13]:

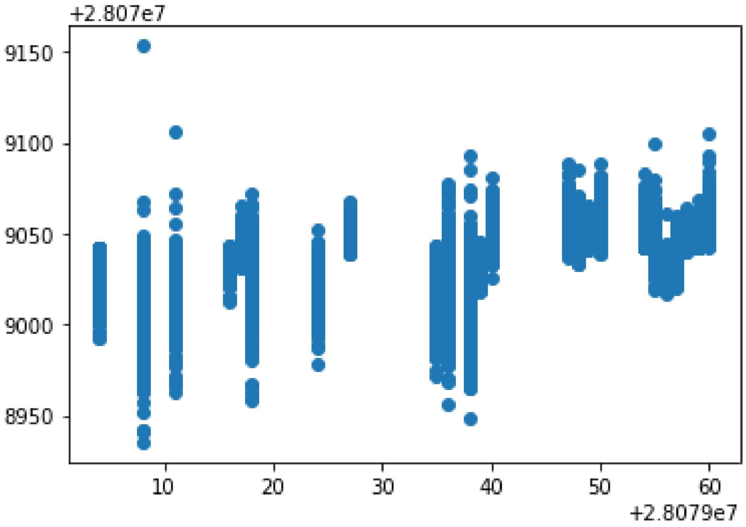
```
LinearRegression()
```

```
In [14]: print(lr.intercept_)

         28079042.027264953
```

```
In [15]: prediction=lr.predict(x_test)
         plt.scatter(y_test,prediction)

Out[15]: <matplotlib.collections.PathCollection at 0x1f81a00a580>
```



```
In [16]: print(lr.score(x_test,y_test))

         0.298928872212475
```

Ridge Regression

```
In [17]: from sklearn.linear_model import Ridge
```

```
In [18]: rr=Ridge(alpha=10)
         rr.fit(x_train,y_train)

Out[18]: Ridge(alpha=10)
```

```
In [19]: rr.score(x_test,y_test)

Out[19]: 0.2990637039599836
```

Lasso Regression

```
In [20]: from sklearn.linear_model import Lasso
```

In [21]:

```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[21]:

```
Lasso(alpha=10)
```

In [22]:

```
la.score(x_test,y_test)
```

Out[22]:

```
0.14425099333511227
```

Elastic Regression

In [23]:

```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[23]:

```
ElasticNet()
```

In [24]:

```
print(en.coef_)
```

```
[-6.55959003e-01 -4.26230045e-01 -7.95802910e-01 -0.00000000e+00
  2.91478073e-02 -3.41943221e-02 -1.14131981e-03  2.78342122e-01
 -2.46922131e-01 -1.31701512e+00 -0.00000000e+00 -1.69251706e+00]
```

In [25]:

```
print(en.predict(x_test))
```

```
[28079035.77703039 28079035.11076136 28079040.3825461  ...
 28079032.62400804 28079038.38271381 28079041.36218826]
```

In [26]:

```
print(en.score(x_test,y_test))
```

```
0.2418437372373553
```

Logistic Regression

In [27]:

```
from sklearn.linear_model import LogisticRegression
```

In [28]:

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

In [29]:

```
feature_matrix.shape
```

Out[29]:

```
(209928, 13)
```

In [30]:

```
target_vector.shape
```

Out[30]:

```
(209928,)
```

In [31]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [33]:
```
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[33]:
```
LogisticRegression()
```

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

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

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

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

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

Out[39]:
```
0.9833323806257384
```

Random Forest

In [40]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [41]:
```
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 [42]:
```
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 [43]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[43]:
```
RandomForestClassifier()
```

In [44]:
```
parameters={'max_depth':[1,2,3,4,5],
           'min_samples_leaf':[5,10,15,20,25],
           'n_estimators':[10,20,30,40,50]}
```

In [45]:
```
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[45]:
```
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 [46]:
```
grid_search.best_score_
```

Out[46]:
```
0.7565975419987933
```

In [47]:
```
rfc_best=grid_search.best_estimator_
```

In [48]:

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

```
1, 13, 27, 3]'),
 Text(148.8, 543.5999999999999, 'NO <= 5.5\ngini = 0.682\nsamples = 5156\n
value = [8, 1, 54, 4, 8, 0, 8, 1, 2, 2, 16, 2, 14\n2724, 2550, 8, 2714, 5,
0, 20, 1, 12, 27, 3]'),
 Text(74.4, 181.19999999999982, 'gini = 0.666\nsamples = 2304\nvalue = [5,
1, 31, 1, 5, 0, 8, 0, 1, 0, 16, 1, 13\n1550, 1198, 8, 728, 5, 0, 17, 0, 8,
20, 1]'),
 Text(223.20000000000002, 181.19999999999982, 'gini = 0.657\nsamples = 285
2\nvalue = [3, 0, 23, 3, 3, 0, 0, 1, 1, 2, 0, 1, 1, 1174\n1352, 0, 1986,
0, 0, 3, 1, 4, 7, 2]'),
 Text(446.40000000000003, 543.5999999999999, 'NO <= 2.5\ngini = 0.292\nsam
ples = 28\nvalue = [0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0,
30, 0, 0, 3, 0, 0]'),
 Text(372.0, 181.19999999999982, 'gini = 0.64\nsamples = 5\nvalue = [0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 2, 0, 0, 2, 0, 0]'),
 Text(520.8000000000001, 181.19999999999982, 'gini = 0.179\nsamples = 23\n
value = [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 28, 0, 0,
1, 0, 0]'),
 Text(892.8000000000001, 906.0, 'BEN <= 0.65\ngini = 0.502\nsamples = 3243
\nvalue = [0, 3, 2553, 0, 0, 0, 5, 0, 0, 0, 1, 0, 0, 0\n0, 0, 0, 0, 2549,
0, 0, 0, 0, 0]'),
```

Results

The best model is Logistic Regression 0.9833323806257384

In [ ]: