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_2017.csv")
data
```

Out[2]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TC
0	2017-06-01 01:00:00	NaN	NaN	0.3	NaN	NaN	4.0	38.0	NaN	NaN	NaN	NaN	5.0	Na
1	2017-06-01 01:00:00	0.6	NaN	0.3	0.4	0.08	3.0	39.0	NaN	71.0	22.0	9.0	7.0	1
2	2017-06-01 01:00:00	0.2	NaN	NaN	0.1	NaN	1.0	14.0	NaN	NaN	NaN	NaN	NaN	Na
3	2017-06-01 01:00:00	NaN	NaN	0.2	NaN	NaN	1.0	9.0	NaN	91.0	NaN	NaN	NaN	Na
4	2017-06-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	19.0	NaN	69.0	NaN	NaN	2.0	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
210115	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	27.0	NaN	65.0	NaN	NaN	NaN	Na
210116	2017-08-01 00:00:00	NaN	NaN	0.2	NaN	NaN	1.0	14.0	NaN	NaN	73.0	NaN	7.0	Na
210117	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	4.0	NaN	83.0	NaN	NaN	NaN	Na
210118	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	11.0	NaN	78.0	NaN	NaN	NaN	Na
210119	2017-08-01 00:00:00	NaN	NaN	NaN	NaN	NaN	1.0	14.0	NaN	77.0	60.0	NaN	NaN	Na

210120 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210120 entries, 0 to 210119
Data columns (total 16 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     210120 non-null  object 
 1   BEN      50201 non-null   float64
 2   CH4      6410 non-null    float64
 3   CO       87001 non-null   float64
 4   EBE      49973 non-null   float64
 5   NMHC     25472 non-null   float64
 6   NO       209065 non-null  float64
 7   NO_2     209065 non-null  float64
 8   NOx      52818 non-null   float64
 9   O_3      121398 non-null  float64
 10  PM10     104141 non-null  float64
 11  PM25     52023 non-null   float64
 12  SO_2     86803 non-null   float64
 13  TCH      25472 non-null   float64
 14  TOL      50117 non-null   float64
 15  station  210120 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 25.6+ MB
```

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

Out[4]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TCH
0	2017-06-01 01:00:00	0.0	0.0	0.3	0.0	0.00	4.0	38.0	0.0	0.0	0.0	0.0	5.0	0.
1	2017-06-01 01:00:00	0.6	0.0	0.3	0.4	0.08	3.0	39.0	0.0	71.0	22.0	9.0	7.0	1.
2	2017-06-01 01:00:00	0.2	0.0	0.0	0.1	0.00	1.0	14.0	0.0	0.0	0.0	0.0	0.0	0.
3	2017-06-01 01:00:00	0.0	0.0	0.2	0.0	0.00	1.0	9.0	0.0	91.0	0.0	0.0	0.0	0.
4	2017-06-01 01:00:00	0.0	0.0	0.0	0.0	0.00	1.0	19.0	0.0	69.0	0.0	0.0	2.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
210115	2017-08-01 00:00:00	0.0	0.0	0.2	0.0	0.00	1.0	27.0	0.0	65.0	0.0	0.0	0.0	0.
210116	2017-08-01 00:00:00	0.0	0.0	0.2	0.0	0.00	1.0	14.0	0.0	0.0	73.0	0.0	7.0	0.
210117	2017-08-01 00:00:00	0.0	0.0	0.0	0.0	0.00	1.0	4.0	0.0	83.0	0.0	0.0	0.0	0.
210118	2017-08-01 00:00:00	0.0	0.0	0.0	0.0	0.00	1.0	11.0	0.0	78.0	0.0	0.0	0.0	0.
210119	2017-08-01 00:00:00	0.0	0.0	0.0	0.0	0.00	1.0	14.0	0.0	77.0	60.0	0.0	0.0	0.

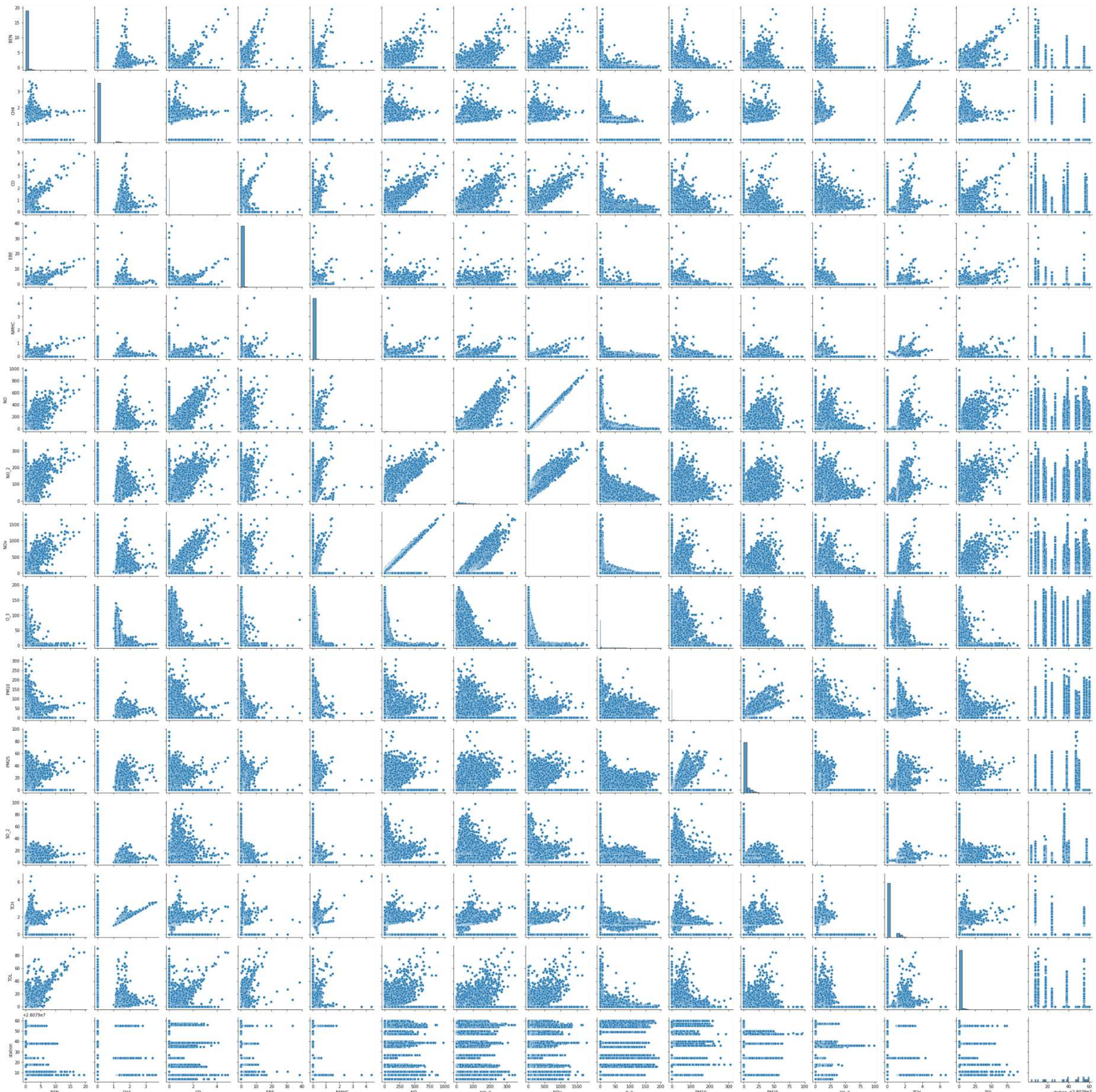
210120 rows × 16 columns

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

Out[5]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

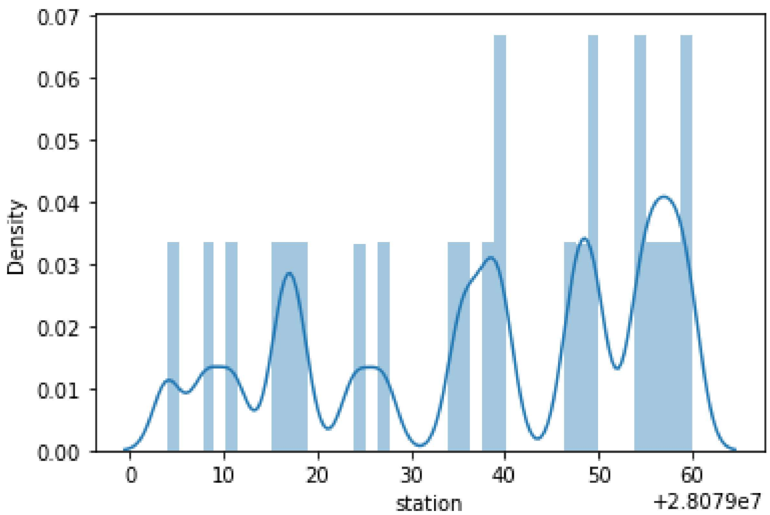
Out[6]: <seaborn.axisgrid.PairGrid at 0x26cbd622df0>



In [7]:
```
sns.distplot(data["station"])
```

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

Out[7]:
```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



MODEL BUILDING

Linear Regression

In [8]:
```
df1=df[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL', 'station']]
```

In [9]:
```
x=df1[['BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TCH', 'TOL']]
y=df1[['station']]
```

In [10]:
```
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 [11]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

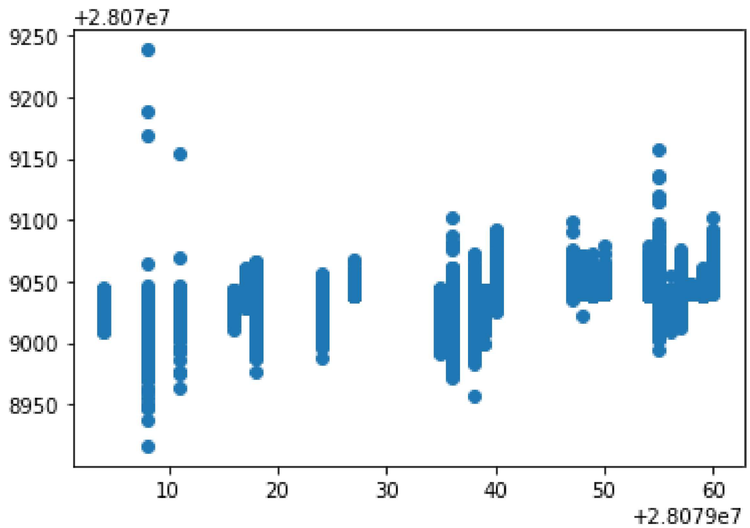
Out[11]:
```
LinearRegression()
```

In [12]:
```
print(lr.intercept_)
```

```
[28079042.81026006]
```

In [13]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[13]: <matplotlib.collections.PathCollection at 0x26c81a63190>



In [14]:
```
print(lr.score(x_test,y_test))
```

```
0.2359153103049333
```

Ridge Regression

In [15]:
```
from sklearn.linear_model import Ridge
```

In [16]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[16]: Ridge(alpha=10)

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

Out[17]: 0.2363491537898943

Lasso Regression

In [18]:
```
from sklearn.linear_model import Lasso
```

In [19]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[19]:
```
Lasso(alpha=10)
```

In [20]:
```
la.score(x_test,y_test)
```

Out[20]:
```
0.08092999365827225
```

Elastic Regression

In [21]:
```
from sklearn.linear_model import ElasticNet
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[21]:
```
ElasticNet()
```

In [22]:
```
print(en.coef_)
```

```
[-0.29327956 -0.68408609 -0.         -0.          0.03832697 -0.07593624
 -0.02483141  0.26352825 -0.30977209 -1.0163488  -0.99777073 -1.12735484]
```

In [23]:
```
print(en.predict(x_test))
```

```
[28079039.1149954  28079042.41788843 28079035.52163452 ...
 28079022.34529698 28079038.61424688 28079035.4925634 ]
```

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

```
0.1591707626137785
```

Logistic Regression

In [25]:
```
from sklearn.linear_model import LogisticRegression
```

In [26]:
```
feature_matrix=df1.iloc[:,0:14]
target_vector=df1.iloc[:,-1]
```

In [27]:
```
feature_matrix.shape
```

Out[27]:
```
(210120, 13)
```

In [28]:
```
target_vector.shape
```

Out[28]:
```
(210120,)
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

In [30]:
```
fs=StandardScaler().fit_transform(feature_matrix)
```

In [31]:
```
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[31]:
```
LogisticRegression()
```

In [32]:
```
observation=[[1,2,3,4,5,6,7,8,9,10,11,12,13]]
```

In [33]:
```
prediction=logr.predict(observation)
print(observation)
```

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

In [34]:
```
logr.classes_
```

Out[34]:
```
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 [35]:
```
logr.score(fs,target_vector)
```

Out[35]:
```
0.9765610127546164
```

Random Forest

In [36]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
```

In [37]:
```
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 [38]:
```
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 [39]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[39]:
```
RandomForestClassifier()
```

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

In [41]:
```
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[41]:
```
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 [42]:
```
grid_search.best_score_
```

Out[42]:
```
0.6964908940922647
```

In [43]:
```
rfc_best=grid_search.best_estimator_
```

In [44]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],feature_names=x.columns,filled=True)
```

```
 Text(3199.2000000000003, 181.19999999999982, 'gini = 0.001\nsamples = 106
9\nvalue = [0, 1, 0, 0, 0, 1702, 0, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0,
0, 0, 0, 0]'),
 Text(3868.8, 1268.4, 'CO <= 0.25\ngini = 0.5\nsamples = 3217\nvalue = [0,
2558, 0, 0, 0, 0, 2505, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0,
0]'),
 Text(3571.2000000000003, 906.0, 'NO <= 1.5\ngini = 0.384\nsamples = 1464
\nvalue = [0, 595, 0, 0, 0, 0, 1701, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0]'),
 Text(3422.4, 543.5999999999999, 'TCH <= 1.295\ngini = 0.058\nsamples = 82
5\nvalue = [0, 39, 0, 0, 0, 0, 1269, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0,
0, 0, 0, 0, 0]'),
 Text(3348.0000000000005, 181.19999999999982, 'gini = 0.005\nsamples = 725
\nvalue = [0, 3, 0, 0, 0, 0, 1140, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0,
0, 0, 0, 0]'),
 Text(3496.8, 181.19999999999982, 'gini = 0.341\nsamples = 100\nvalue =
[0, 36, 0, 0, 0, 0, 129, 0, 0, 0, 0, 0, 0, 0\n0, 0, 0, 0, 0, 0, 0, 0, 0,
0]'),
 Text(3720.0000000000005, 543.5999999999999, 'TCH <= 1.325\ngini = 0.492\n
samples = 639\nvalue = [0, 556, 0, 0, 0, 0, 432, 0, 0, 0, 0, 0, 0, 0\n0,
```

Results

The best model is Logistic Regression 0.9765610127546164

In [ ]: