In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge,Lasso
from sklearn.linear_model import ElasticNet
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree
```

In [2]:
```
df=pd.read_csv(r"C:\Users\user\Downloads\csvs_per_year\csvs_per_year\madrid_20
df
```

Out[2]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
0	2007-12-01 01:00:00	NaN	2.86	NaN	NaN	NaN	282.200012	1054.000000	NaN	4.030000	156.
1	2007-12-01 01:00:00	NaN	1.82	NaN	NaN	NaN	86.419998	354.600006	NaN	3.260000	80.
2	2007-12-01 01:00:00	NaN	1.47	NaN	NaN	NaN	94.639999	319.000000	NaN	5.310000	53.
3	2007-12-01 01:00:00	NaN	1.64	NaN	NaN	NaN	127.900002	476.700012	NaN	4.500000	105.
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.
...	...	...	...	...	...	...	...	...	...	...	
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.
225116	2007-03-01 00:00:00	NaN	0.16	NaN	NaN	NaN	46.820000	51.480000	NaN	22.150000	5.
225117	2007-03-01 00:00:00	0.24	NaN	0.20	NaN	0.09	51.259998	66.809998	NaN	18.540001	13.
225118	2007-03-01 00:00:00	0.11	NaN	1.00	NaN	0.05	24.240000	36.930000	NaN	NaN	6.
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.

225120 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 225120 entries, 0 to 225119
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     225120 non-null  object 
 1   BEN      68885 non-null   float64
 2   CO       206748 non-null  float64
 3   EBE      68883 non-null   float64
 4   MXY      26061 non-null   float64
 5   NMHC     86883 non-null   float64
 6   NO_2     223985 non-null  float64
 7   NOx      223972 non-null  float64
 8   OXY      26062 non-null   float64
 9   O_3      211850 non-null  float64
 10  PM10     222588 non-null  float64
 11  PM25     68870 non-null   float64
 12  PXY      26062 non-null   float64
 13  SO_2     224372 non-null  float64
 14  TCH      87026 non-null   float64
 15  TOL      68845 non-null   float64
 16  station  225120 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 29.2+ MB
```

In [4]:
```
df=df.dropna()
df
```

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	
4	2007-12-01 01:00:00	4.64	1.86	4.26	7.98	0.57	145.100006	573.900024	3.49	52.689999	106.5(
21	2007-12-01 01:00:00	1.98	0.31	2.56	6.06	0.35	76.059998	208.899994	1.70	1.000000	37.7(
25	2007-12-01 01:00:00	2.82	1.42	3.15	7.02	0.49	123.099998	402.399994	2.60	7.160000	70.8(
30	2007-12-01 02:00:00	4.65	1.89	4.41	8.21	0.65	151.000000	622.700012	3.55	58.080002	117.0(
47	2007-12-01 02:00:00	1.97	0.30	2.15	5.08	0.33	78.760002	189.800003	1.62	1.000000	34.7(
...	...	...	...	...	...	...	...	...	...	...	
225073	2007-02-28 23:00:00	2.12	0.47	2.51	4.99	0.05	43.560001	83.889999	2.57	13.090000	21.8(
225094	2007-02-28 23:00:00	0.87	0.45	1.19	2.66	0.13	40.000000	61.959999	1.79	20.440001	15.0(
225098	2007-03-01 00:00:00	0.95	0.41	1.55	3.11	0.05	36.090000	63.349998	1.74	17.160000	9.2(
225115	2007-03-01 00:00:00	0.30	0.45	1.00	0.30	0.26	8.690000	11.690000	1.00	42.209999	6.7(
225119	2007-03-01 00:00:00	0.53	0.40	1.00	1.70	0.12	32.360001	47.860001	1.37	24.150000	10.2(

25443 rows × 17 columns

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
date       0
BEN        0
CO         0
EBE        0
MXY        0
NMHC       0
NO_2       0
NOx        0
OXY        0
O_3        0
PM10       0
PM25       0
PXY        0
SO_2       0
TCH        0
TOL        0
station    0
dtype: int64
```

In [6]:
```
df.describe()
```

Out[6]:

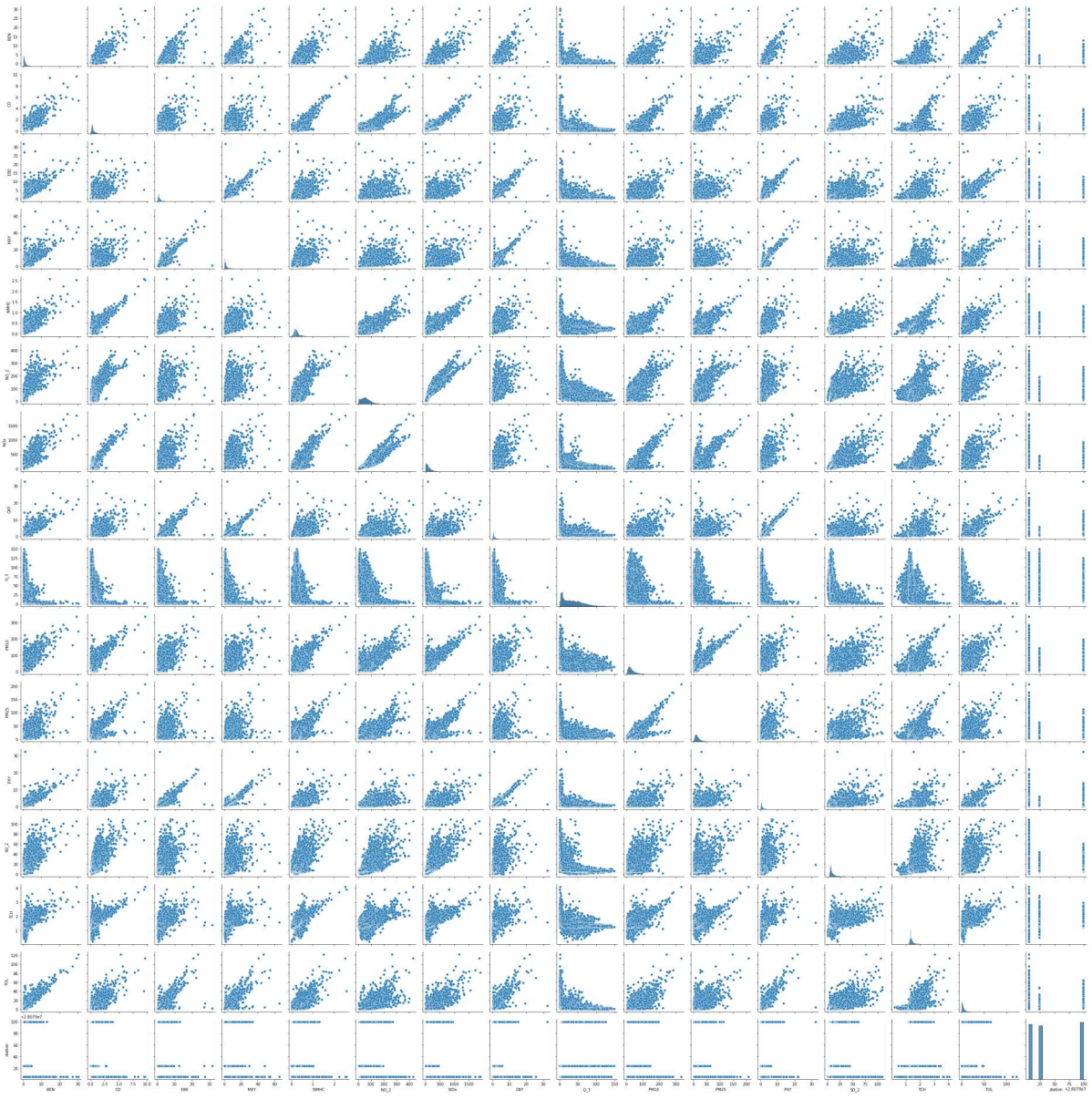
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	25443.000000	254
mean	1.146744	0.505120	1.394071	2.392008	0.249967	58.532683	1
std	1.278733	0.423231	1.268265	2.784302	0.142627	37.755029	1
min	0.130000	0.000000	0.120000	0.150000	0.000000	1.690000	
25%	0.450000	0.260000	0.780000	0.960000	0.160000	31.285001	
50%	0.770000	0.400000	1.000000	1.500000	0.220000	54.080002	
75%	1.390000	0.640000	1.580000	2.855000	0.300000	79.230003	1
max	30.139999	9.660000	31.680000	65.480003	2.570000	430.299988	18

In [7]:
```
df.columns
```

Out[7]:
```
Index(['date', 'BEN', 'CO', 'EBE', 'MXY', 'NMHC', 'NO_2', 'NOx', 'OXY', 'O_
3',
       'PM10', 'PM25', 'PXY', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

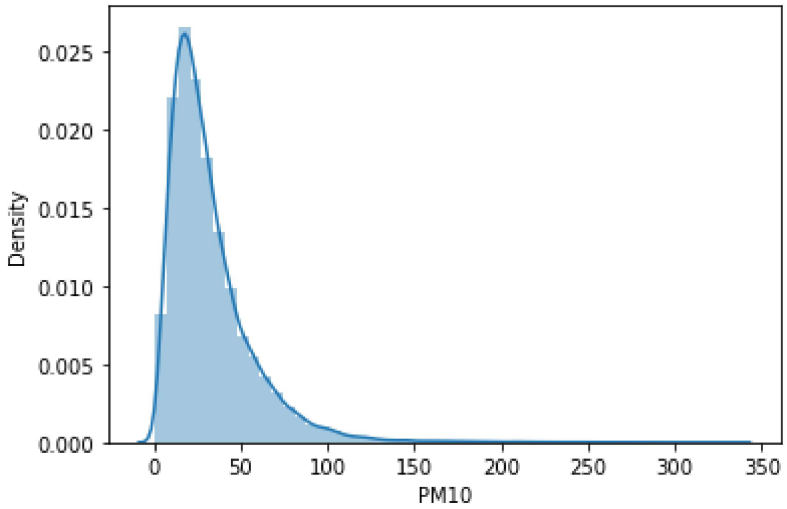
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1c881327fa0>
```



In [9]:
```
sns.distplot(df['PM10'])
```

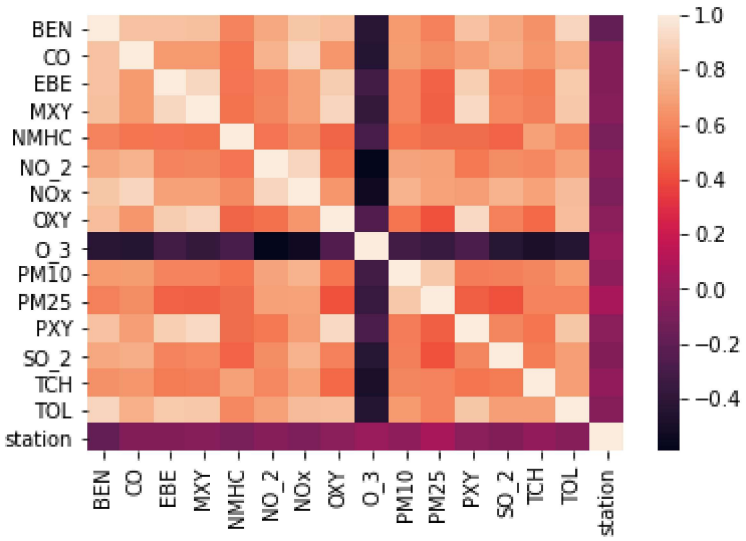
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function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

Out[9]: <AxesSubplot:xlabel='PM10', ylabel='Density'>



In [10]:
```
sns.heatmap(df.corr())
```

Out[10]: <AxesSubplot:>



In [11]:
```
df.loc[df['NMHC']<1,'NMHC']=0
df.loc[df['NMHC']>1,'NMHC']=1
df['NMHC']=df['NMHC'].astype(int)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1720: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  self._setitem_single_column(loc, value, pi)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1720: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  self._setitem_single_column(loc, value, pi)
<ipython-input-11-c5145d14383f>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  df['NMHC']=df['NMHC'].astype(int)
```

LogisticRegression

In [12]:
```
x=df[[ 'BEN', 'CO', 'EBE', 'MXY', 'NO_2', 'NOx', 'OXY', 'O_3',
       'PM10', 'PXY', 'SO_2', 'TCH', 'TOL', 'station']]
y=df['NMHC']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
lgr=LogisticRegression()
lgr.fit(x_train,y_train)
```

Out[12]:
```
LogisticRegression()
```

In [13]:
```
lgr.predict(x_test)
```

Out[13]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

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

Out[14]:
```
0.997248788156688
```

In [15]:
```
fs=StandardScaler().fit_transform(x)
logr=LogisticRegression()
logr.fit(fs,y)
```

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

In [16]:
```
o=[[1,2,3,4,5,6,7,8,9,10,11,12,13,14]]
prediction=logr.predict(o)
print(prediction)
```

```
[1]
```

In [17]:
```
logr.classes_
```

Out[17]:
```
array([0, 1])
```

In [18]:
```
logr.predict_proba(o)[0][0]
```

Out[18]:
```
0.0004090635017235389
```

In [19]:
```
logr.predict_proba(o)[0][1]
```

Out[19]:
```
0.9995909364982765
```

LinearRegression

In [20]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[20]:
```
LinearRegression()
```

In [21]:
```
print(lr.intercept_)
```

```
-1961.4137390217
```

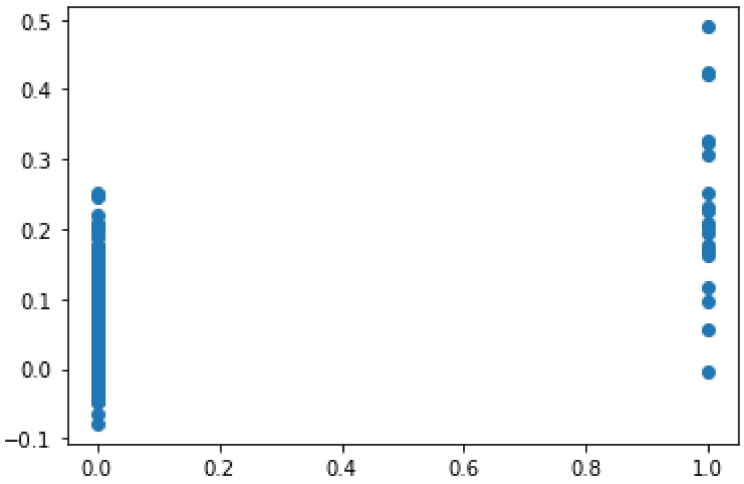
In [22]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[22]:

	Co-efficient
BEN	0.015548
CO	0.034187
EBE	-0.002571
MXY	-0.000945
NO_2	-0.000404
NOx	0.000161
OXY	-0.002479
O_3	0.000295
PM10	-0.000042
PXY	0.004161
SO_2	-0.000861
TCH	0.010441
TOL	-0.000225
station	0.000070

In [23]:
```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[23]:
```
<matplotlib.collections.PathCollection at 0x1c89982b580>
```



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

```
0.21746168204756788
```

Ridge,Lasso

In [25]:
```
rr=Ridge(alpha=10)
rr.fit(x_train,y_train)
```

Out[25]:
```
Ridge(alpha=10)
```

In [26]:
```
rr.score(x_test,y_test)
```

Out[26]:
```
0.21758399930163264
```

In [27]:
```
la=Lasso(alpha=10)
la.fit(x_train,y_train)
```

Out[27]:
```
Lasso(alpha=10)
```

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

Out[28]:
```
-4.592046030538199e-06
```

ElasticNet

In [29]:
```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[29]:
```
ElasticNet()
```

In [30]:
```
print(en.coef_)
```

```
[ 0.          0.          0.          0.         -0.          0.00015192
  0.          0.          0.          0.          0.          0.
  0.         -0.        ]
```

In [31]:
```
print(en.intercept_)
```

```
-0.014580707229938499
```

In [32]:
```
print(en.predict(x_train))
```

```
[-0.00379866 -0.00383361 -0.01143588 ... -0.00030897 -0.00225663
  0.00266266]
```

In [33]:
```
print(en.score(x_train,y_train))
```

```
0.17677487225058175
```

In [34]:
```
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolytre Error: 0.014885388003927746
```

In [35]:
```
print("Mean Square Error:",metrics.mean_squared_error(y_test,prediction))
```

```
Mean Square Error: 0.0021470055252913616
```

In [36]:
```
print("Root Mean Square Error:",np.sqrt(metrics.mean_absolute_error(y_test,pre
```

```
Root Mean Square Error: 0.12200568840807278
```

RandomForest

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

Out[37]:
```
RandomForestClassifier()
```

In [38]:
```
parameters={'max_depth':[1,2,3,4,5],
            'min_samples_leaf':[5,10,15,20,25],
            'n_estimators':[10,20,30,40,50]}
```

In [39]:
```
grid_search=GridSearchCV(estimator=rfc,param_grid=parameters,cv=2,scoring="acc
grid_search.fit(x_train,y_train)
```

Out[39]:
```
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')
```

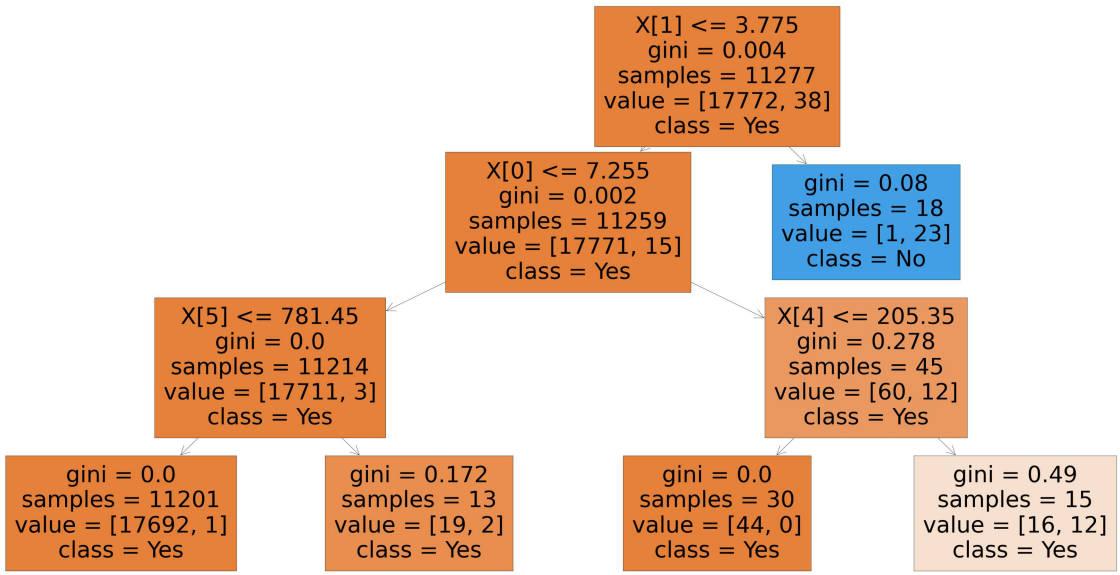
In [40]:
```
grid_search.best_score_
```

Out[40]:
```
0.9988770353733858
```

In [41]:
```
rfc_best=grid_search.best_estimator_
```

In [42]:
```
plt.figure(figsize=(80,40))
plot_tree(rfc_best.estimators_[5],class_names=['Yes','No','Yes','No'],filled=T
```

Out[42]:
```
[Text(2790.0, 1902.6000000000001, 'X[1] <= 3.775\ngini = 0.004\nsamples = 112
77\nvalue = [17772, 38]\nclass = Yes'),
 Text(2232.0, 1359.0, 'X[0] <= 7.255\ngini = 0.002\nsamples = 11259\nvalue =
[17771, 15]\nclass = Yes'),
 Text(1116.0, 815.4000000000001, 'X[5] <= 781.45\ngini = 0.0\nsamples = 11214
\nvalue = [17711, 3]\nclass = Yes'),
 Text(558.0, 271.79999999999995, 'gini = 0.0\nsamples = 11201\nvalue = [1769
2, 1]\nclass = Yes'),
 Text(1674.0, 271.79999999999995, 'gini = 0.172\nsamples = 13\nvalue = [19,
2]\nclass = Yes'),
 Text(3348.0, 815.4000000000001, 'X[4] <= 205.35\ngini = 0.278\nsamples = 45
\nvalue = [60, 12]\nclass = Yes'),
 Text(2790.0, 271.79999999999995, 'gini = 0.0\nsamples = 30\nvalue = [44, 0]
\nclass = Yes'),
 Text(3906.0, 271.79999999999995, 'gini = 0.49\nsamples = 15\nvalue = [16, 1
2]\nclass = Yes'),
 Text(3348.0, 1359.0, 'gini = 0.08\nsamples = 18\nvalue = [1, 23]\nclass = N
o')]
```



Best model:RandomForest

In [ ]:
```
```