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	PM10
0	2005-11-01 01:00:00	NaN	0.77	NaN	NaN	NaN	57.130001	128.699997	NaN	14.720000	14.91
1	2005-11-01 01:00:00	1.52	0.65	1.49	4.57	0.25	86.559998	181.699997	1.27	11.680000	30.93
2	2005-11-01 01:00:00	NaN	0.40	NaN	NaN	NaN	46.119999	53.000000	NaN	30.469999	14.60
3	2005-11-01 01:00:00	NaN	0.42	NaN	NaN	NaN	37.220001	52.009998	NaN	21.379999	15.16
4	2005-11-01 01:00:00	NaN	0.57	NaN	NaN	NaN	32.160000	36.680000	NaN	33.410000	5.00
...	...	...	...	...	...	...	...	...	...	...	...
236995	2006-01-01 00:00:00	1.08	0.36	1.01	NaN	0.11	21.990000	23.610001	NaN	43.349998	5.00
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95
236997	2006-01-01 00:00:00	0.19	NaN	0.26	NaN	0.08	26.730000	30.809999	NaN	43.840000	4.31
236998	2006-01-01 00:00:00	0.14	NaN	1.00	NaN	0.06	13.770000	17.770000	NaN	NaN	5.00
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67

237000 rows × 17 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237000 entries, 0 to 236999
Data columns (total 17 columns):
 #   Column   Non-Null Count   Dtype  
---  ------   --------------   -----  
 0   date     237000 non-null  object 
 1   BEN      70370 non-null   float64
 2   CO       217656 non-null  float64
 3   EBE      68955 non-null   float64
 4   MXY      32549 non-null   float64
 5   NMHC     92854 non-null   float64
 6   NO_2     235022 non-null  float64
 7   NOx      235049 non-null  float64
 8   OXY      32555 non-null   float64
 9   O_3      223162 non-null  float64
 10  PM10     232142 non-null  float64
 11  PM25     69407 non-null   float64
 12  PXY      32549 non-null   float64
 13  SO_2     235277 non-null  float64
 14  TCH      93076 non-null   float64
 15  TOL      70255 non-null   float64
 16  station  237000 non-null  int64  
dtypes: float64(15), int64(1), object(1)
memory usage: 30.7+ MB
```

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

Out[4]:

	date	BEN	CO	EBE	MXY	NMHC	NO_2	NOx	OXY	O_3	PM10
5	2005-11-01 01:00:00	1.92	0.88	2.44	5.14	0.22	90.309998	207.699997	2.78	13.760000	18.07
22	2005-11-01 01:00:00	0.30	0.22	0.25	0.59	0.11	18.540001	19.020000	0.67	46.799999	9.88
25	2005-11-01 01:00:00	0.67	0.49	0.94	3.44	0.17	48.740002	74.349998	1.57	23.430000	13.88
31	2005-11-01 02:00:00	3.10	0.84	3.21	6.82	0.22	89.919998	224.199997	3.72	12.390000	28.74
48	2005-11-01 02:00:00	0.39	0.20	0.29	0.68	0.11	16.639999	17.080000	0.40	47.689999	8.78
...	...	...	...	...	...	...	...	...	...	...	...
236970	2005-12-31 23:00:00	0.37	0.39	1.00	1.00	0.10	4.500000	5.550000	1.00	57.779999	8.26
236973	2005-12-31 23:00:00	0.92	0.45	1.26	3.42	0.14	37.250000	49.060001	2.57	31.889999	19.73
236979	2006-01-01 00:00:00	1.00	0.38	1.11	2.35	0.04	35.919998	59.480000	1.39	35.810001	4.22
236996	2006-01-01 00:00:00	0.39	0.54	1.00	1.00	0.11	2.200000	4.220000	1.00	69.639999	4.95
236999	2006-01-01 00:00:00	0.50	0.40	0.73	1.84	0.13	20.940001	26.950001	1.49	48.259998	5.67

20070 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]:

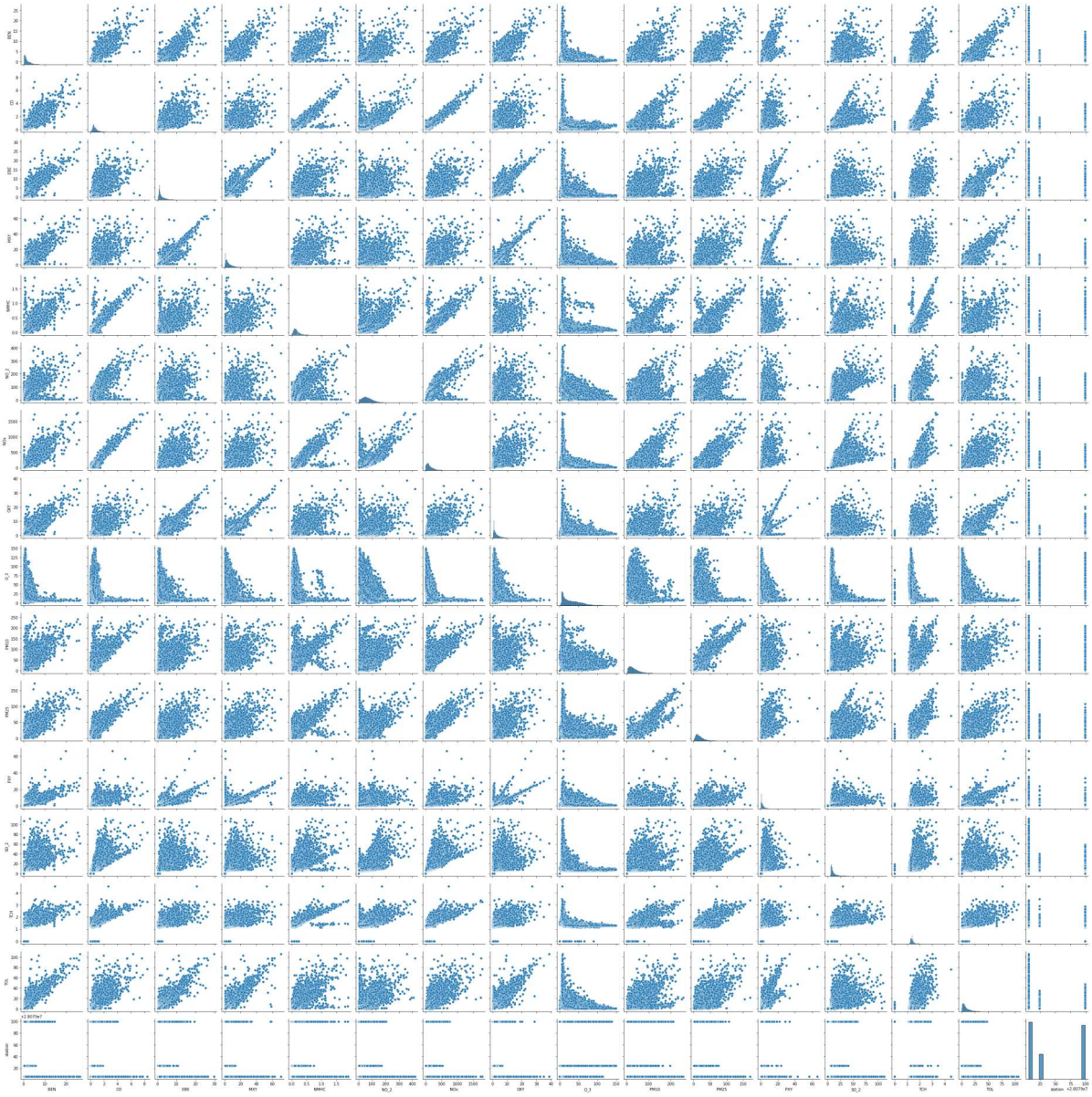
	BEN	CO	EBE	MXY	NMHC	NO_2	
count	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	20070.000000	200
mean	1.923656	0.720657	2.345423	5.457855	0.179282	66.226924	1
std	2.019061	0.549723	2.379219	5.495147	0.152783	40.568197	1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.690000	0.400000	0.950000	1.930000	0.090000	36.602499	
50%	1.260000	0.580000	1.480000	3.800000	0.150000	60.525000	1
75%	2.510000	0.880000	2.950000	7.210000	0.220000	89.317499	1
max	26.570000	8.380000	29.870001	71.050003	1.880000	419.500000	17

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)
```

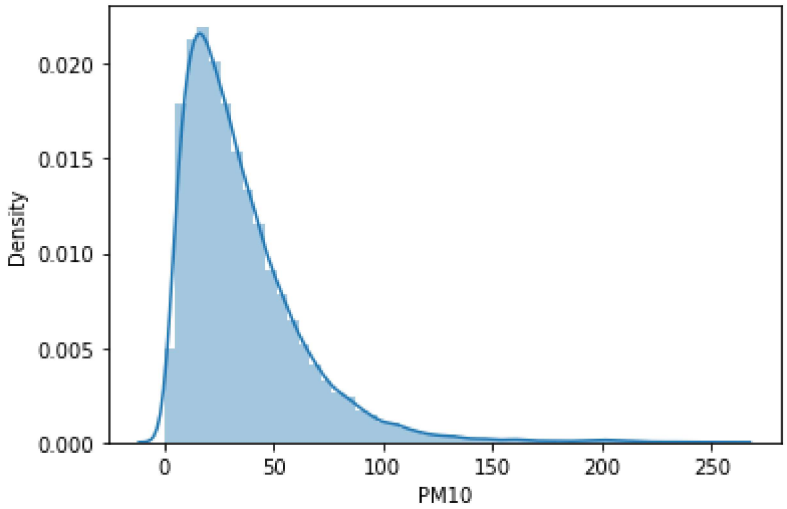
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1edb709c190>
```



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

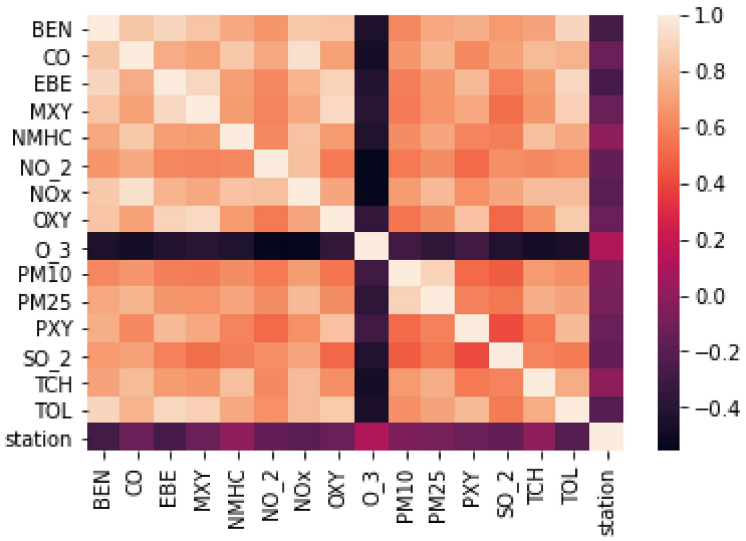
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function 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]:
```
0.9951835243315064
```

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]:
```
2.892727901659953e-07
```

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

Out[19]:
```
0.9999997107272098
```

LinearRegression

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

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

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

```
-304.51660357621955
```

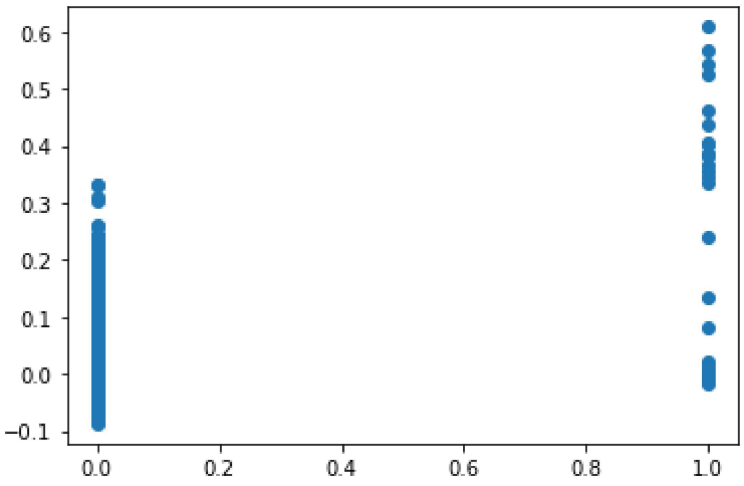
In [22]:
```
coeff = pd.DataFrame(lr.coef_,x.columns,columns=['Co-efficient'])
coeff
```

Out[22]:

	Co-efficient
BEN	0.004289
CO	0.036282
EBE	-0.008472
MXY	-0.000160
NO_2	-0.000656
NOx	0.000347
OXY	0.002620
O_3	0.000413
PM10	-0.000097
PXY	-0.000186
SO_2	-0.000915
TCH	0.018613
TOL	-0.000031
station	0.000011

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

Out[23]:
```
<matplotlib.collections.PathCollection at 0x1edbd21d910>
```



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

```
0.2128129695633354
```

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]:
```
0.2129180042166411
```

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]:
```
-0.0002171555975640782
```

ElasticNet

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

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

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

```
[ 0.          0.         -0.          0.         -0.          0.00019286
  0.          0.          0.          0.         -0.          0.
  0.          0.        ]
```

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

```
-0.02194357386618345
```

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

```
[ 0.00181618 -0.01605956  0.00557686 ...  0.01277035 -0.01969874
  0.00773683]
```

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

```
0.15763576705512827
```

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

```
Mean Absolytre Error: 0.02076448595288982
```

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

```
Mean Square Error: 0.0037732056692380625
```

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

```
Root Mean Square Error: 0.14409887561285764
```

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')
```

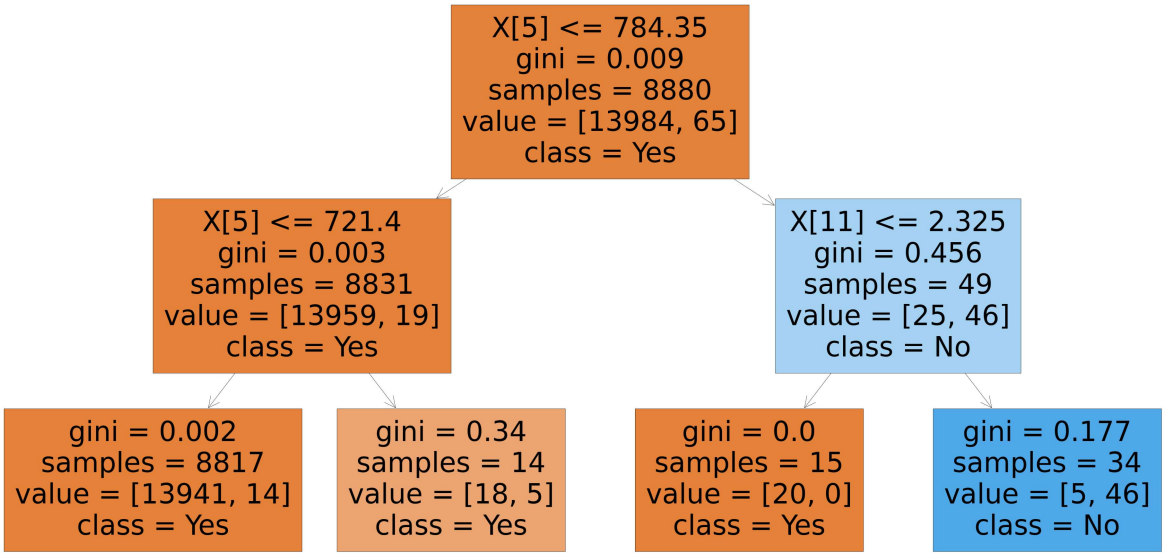
In [40]:
```
grid_search.best_score_
```

Out[40]:
```
0.9973663453821773
```

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(2232.0, 1812.0, 'X[5] <= 784.35\ngini = 0.009\nsamples = 8880\nvalue =
[13984, 65]\nclass = Yes'),
 Text(1116.0, 1087.2, 'X[5] <= 721.4\ngini = 0.003\nsamples = 8831\nvalue =
[13959, 19]\nclass = Yes'),
 Text(558.0, 362.39999999999986, 'gini = 0.002\nsamples = 8817\nvalue = [1394
1, 14]\nclass = Yes'),
 Text(1674.0, 362.39999999999986, 'gini = 0.34\nsamples = 14\nvalue = [18, 5]
\nclass = Yes'),
 Text(3348.0, 1087.2, 'X[11] <= 2.325\ngini = 0.456\nsamples = 49\nvalue = [2
5, 46]\nclass = No'),
 Text(2790.0, 362.39999999999986, 'gini = 0.0\nsamples = 15\nvalue = [20, 0]
\nclass = Yes'),
 Text(3906.0, 362.39999999999986, 'gini = 0.177\nsamples = 34\nvalue = [5, 4
6]\nclass = No')]
```



Best model:RandomForest

In [ ]:
```
```