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	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	T
0	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	29.0	31.0	NaN	NaN	NaN	2.0	N
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1
2	2018-03-01 01:00:00	0.4	NaN	NaN	0.2	NaN	4.0	41.0	47.0	NaN	NaN	NaN	NaN	N
3	2018-03-01 01:00:00	NaN	NaN	0.3	NaN	NaN	1.0	35.0	37.0	54.0	NaN	NaN	NaN	N
4	2018-03-01 01:00:00	NaN	NaN	NaN	NaN	NaN	1.0	27.0	29.0	49.0	NaN	NaN	3.0	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69091	2018-02-01 00:00:00	NaN	NaN	0.5	NaN	NaN	66.0	91.0	192.0	1.0	35.0	22.0	NaN	N
69092	2018-02-01 00:00:00	NaN	NaN	0.7	NaN	NaN	87.0	107.0	241.0	NaN	29.0	NaN	15.0	N
69093	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	28.0	48.0	91.0	2.0	NaN	NaN	NaN	N
69094	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	141.0	103.0	320.0	2.0	NaN	NaN	NaN	N
69095	2018-02-01 00:00:00	NaN	NaN	NaN	NaN	NaN	69.0	96.0	202.0	3.0	26.0	NaN	NaN	N

69096 rows × 16 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69096 entries, 0 to 69095
Data columns (total 16 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   date     69096 non-null  object 
 1   BEN      16950 non-null  float64
 2   CH4      8440 non-null   float64
 3   CO       28598 non-null  float64
 4   EBE      16949 non-null  float64
 5   NMHC     8440 non-null   float64
 6   NO       68826 non-null  float64
 7   NO_2     68826 non-null  float64
 8   NOx      68826 non-null  float64
 9   O_3      40049 non-null  float64
 10  PM10     36911 non-null  float64
 11  PM25     18912 non-null  float64
 12  SO_2     28586 non-null  float64
 13  TCH      8440 non-null   float64
 14  TOL      16950 non-null  float64
 15  station  69096 non-null  int64  
dtypes: float64(14), int64(1), object(1)
memory usage: 8.4+ MB
```

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

Out[4]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TC
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	1.
6	2018-03-01 01:00:00	0.4	1.11	0.2	0.1	0.06	1.0	25.0	27.0	55.0	5.0	4.0	4.0	1.
25	2018-03-01 02:00:00	0.4	1.42	0.2	0.1	0.01	4.0	26.0	32.0	64.0	4.0	4.0	3.0	1.
30	2018-03-01 02:00:00	0.3	1.10	0.2	0.1	0.05	1.0	12.0	13.0	69.0	5.0	4.0	4.0	1.
49	2018-03-01 03:00:00	0.3	1.41	0.2	0.1	0.01	3.0	16.0	20.0	68.0	3.0	2.0	3.0	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69030	2018-01-31 22:00:00	1.8	1.21	0.7	1.7	0.19	151.0	129.0	361.0	1.0	45.0	26.0	11.0	1.
69049	2018-01-31 23:00:00	3.1	1.87	1.2	2.0	0.35	296.0	162.0	615.0	3.0	39.0	23.0	8.0	2.
69054	2018-01-31 23:00:00	1.6	1.17	0.6	1.4	0.15	127.0	106.0	301.0	1.0	43.0	25.0	8.0	1.
69073	2018-02-01 00:00:00	3.2	1.53	1.0	2.1	0.19	125.0	117.0	309.0	3.0	37.0	24.0	6.0	1.
69078	2018-02-01 00:00:00	1.3	1.14	0.4	0.8	0.10	54.0	73.0	155.0	1.0	27.0	16.0	5.0	1.

4562 rows × 16 columns

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

Out[5]:
```
date       0
BEN        0
CH4        0
CO         0
EBE        0
NMHC       0
NO         0
NO_2       0
NOx        0
O_3        0
PM10       0
PM25       0
SO_2       0
TCH        0
TOL        0
station    0
dtype: int64
```

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

Out[6]:

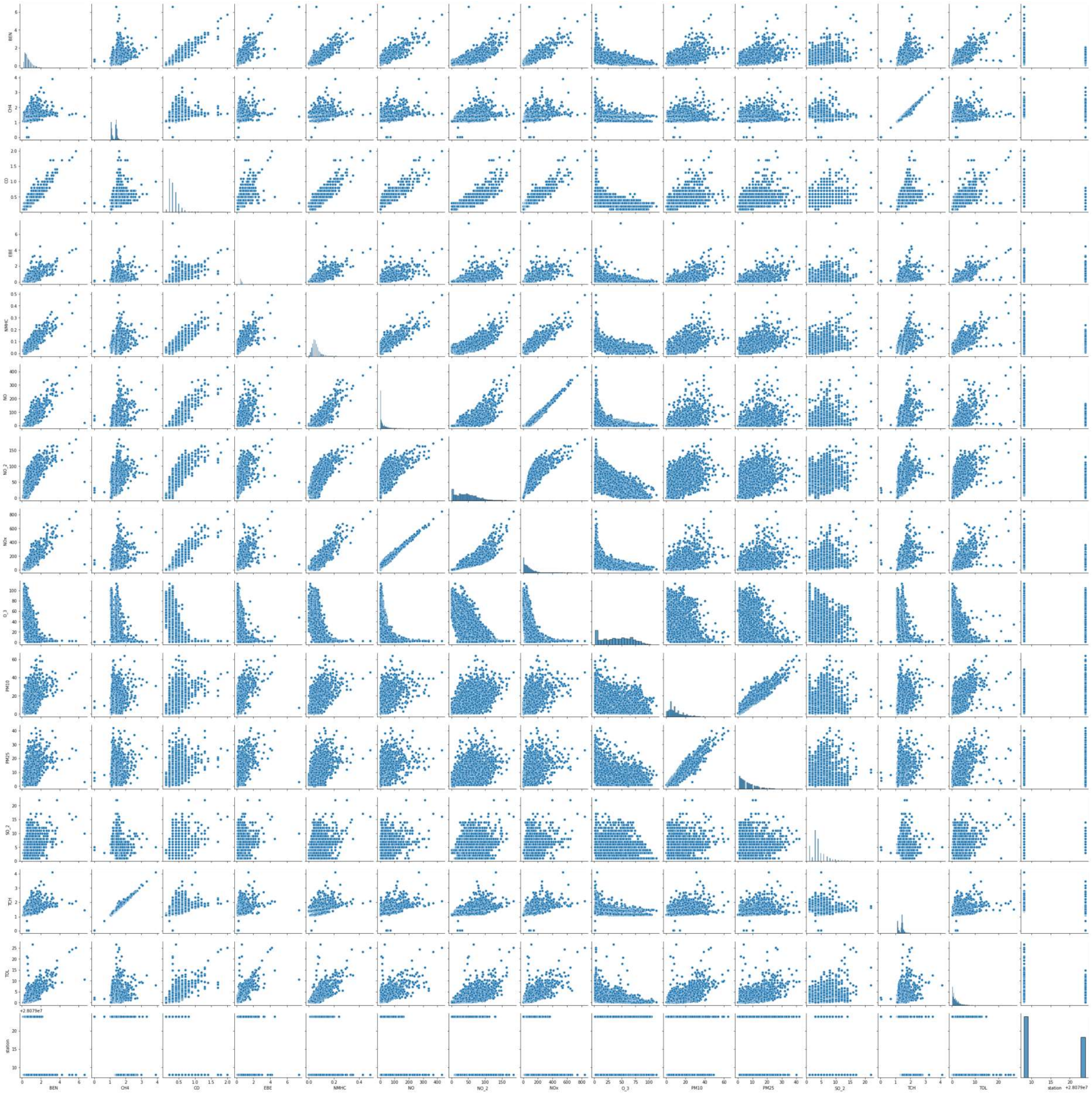
	BEN	CH4	CO	EBE	NMHC	NO	NO_2
count	4562.00000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000	4562.000000
mean	0.69349	1.329163	0.330579	0.286782	0.056773	21.742218	44.152126
std	0.46832	0.214399	0.161489	0.354442	0.037711	35.539531	30.234015
min	0.10000	0.020000	0.100000	0.100000	0.000000	1.000000	1.000000
25%	0.40000	1.120000	0.200000	0.100000	0.030000	1.000000	20.000000
50%	0.60000	1.390000	0.300000	0.200000	0.050000	9.000000	41.000000
75%	0.90000	1.420000	0.400000	0.300000	0.070000	27.000000	64.000000
max	6.60000	3.920000	2.000000	7.400000	0.490000	431.000000	184.000000

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

Out[7]:
```
Index(['date', 'BEN', 'CH4', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'NOx', 'O_3',
       'PM10', 'PM25', 'SO_2', 'TCH', 'TOL', 'station'],
      dtype='object')
```

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

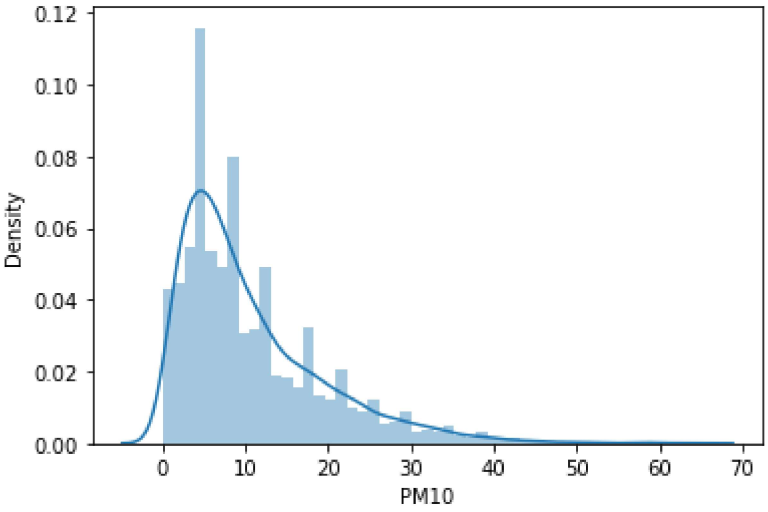
Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x21ddf7180a0>
```



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

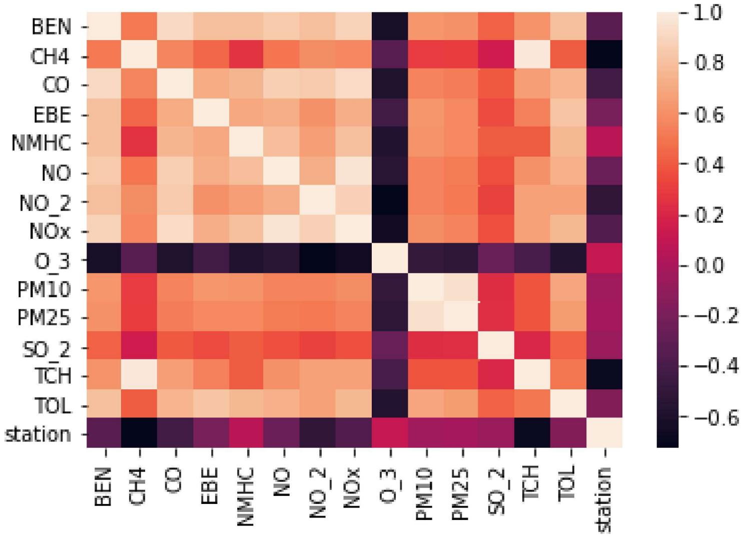
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `histplot` (an axes-level function 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['TCH']<2,'TCH']=0
df.loc[df['TCH']>2,'TCH']=1
df['TCH']=df['TCH'].astype(int)
df
```

```
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-e3d36a273982>: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['TCH']=df['TCH'].astype(int)
```

Out[11]:

	date	BEN	CH4	CO	EBE	NMHC	NO	NO_2	NOx	O_3	PM10	PM25	SO_2	TC
1	2018-03-01 01:00:00	0.5	1.39	0.3	0.2	0.02	6.0	40.0	49.0	52.0	5.0	4.0	3.0	
6	2018-03-01 01:00:00	0.4	1.11	0.2	0.1	0.06	1.0	25.0	27.0	55.0	5.0	4.0	4.0	
25	2018-03-01 02:00:00	0.4	1.42	0.2	0.1	0.01	4.0	26.0	32.0	64.0	4.0	4.0	3.0	
30	2018-03-01 02:00:00	0.3	1.10	0.2	0.1	0.05	1.0	12.0	13.0	69.0	5.0	4.0	4.0	
49	2018-03-01 03:00:00	0.3	1.41	0.2	0.1	0.01	3.0	16.0	20.0	68.0	3.0	2.0	3.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
69030	2018-01-31 22:00:00	1.8	1.21	0.7	1.7	0.19	151.0	129.0	361.0	1.0	45.0	26.0	11.0	
69049	2018-01-31 23:00:00	3.1	1.87	1.2	2.0	0.35	296.0	162.0	615.0	3.0	39.0	23.0	8.0	
69054	2018-01-31 23:00:00	1.6	1.17	0.6	1.4	0.15	127.0	106.0	301.0	1.0	43.0	25.0	8.0	
69073	2018-02-01 00:00:00	3.2	1.53	1.0	2.1	0.19	125.0	117.0	309.0	3.0	37.0	24.0	6.0	
69078	2018-02-01 00:00:00	1.3	1.14	0.4	0.8	0.10	54.0	73.0	155.0	1.0	27.0	16.0	5.0	

4562 rows × 16 columns

LogisticRegression

In [12]:

```
x=df[[ 'BEN', 'CO', 'EBE', 'NMHC', 'NO', 'NO_2', 'O_3', 'PM10', 'PM25',
       'SO_2', 'TOL', 'station']]
y=df['TCH']
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]:
```
0.9824689554419284
```

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

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

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

```
[0]
```

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

Out[17]:
```
array([0, 1, 2])
```

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

Out[18]:
```
0.9999999975037841
```

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

Out[19]:
```
5.085510314304287e-16
```

LinearRegression

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

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

In [21]:

```
print(lr.intercept_)
```

```
104579.50711627983
```

In [22]:

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

Out[22]:

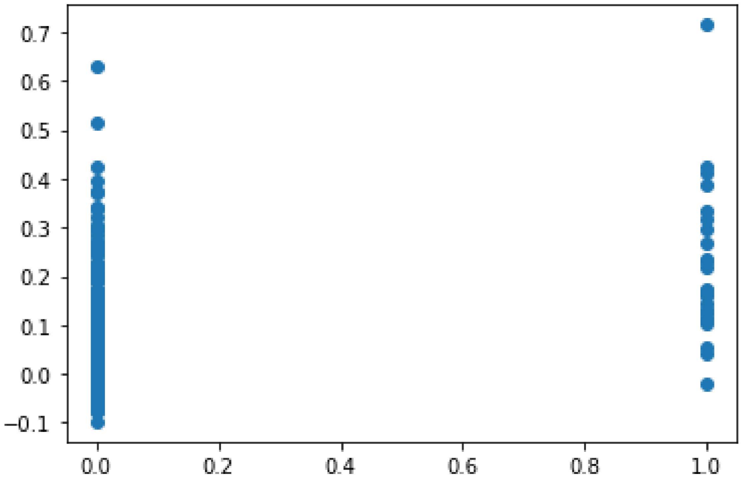
	Co-efficient
BEN	-0.016616
CO	-0.396203
EBE	0.061732
NMHC	1.052382
NO	0.002328
NO_2	-0.000613
O_3	-0.000199
PM10	-0.001645
PM25	0.003811
SO_2	-0.002612
TOL	-0.005893
station	-0.003724

In [23]:

```
prediction=lr.predict(x_test)
plt.scatter(y_test,prediction)
```

Out[23]:

```
<matplotlib.collections.PathCollection at 0x21df29904f0>
```



In [24]:

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

```
0.16185028565714
```

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]:

```
0.15748597817418575
```

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]:

```
-0.00020661188397363972
```

ElasticNet

In [29]:

```
en=ElasticNet()
en.fit(x_train,y_train)
```

Out[29]:

```
ElasticNet()
```

In [30]:

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

```
[ 0.          0.          0.          0.          0.00122064  0.
 -0.          0.          0.         -0.          0.          0.        ]
```

In [31]:

```
print(en.intercept_)
```

```
-0.007178926900212195
```

In [32]:

```
print(en.predict(x_train))
```

```
[-0.00595829 -0.00595829  0.0807071  ...  0.02211641  0.00258619
  0.05873559]
```

In [33]:

```
print(en.score(x_train,y_train))
```

```
0.14800264044790312
```

In [34]:

```
print("Mean Absolytre Error:",metrics.mean_absolute_error(y_test,prediction))
```

```
Mean Absolytre Error: 0.04467923237090176
```

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

```
Mean Square Error: 0.014436045131121353
```

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

```
Root Mean Square Error: 0.21137462565526108
```

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')
```

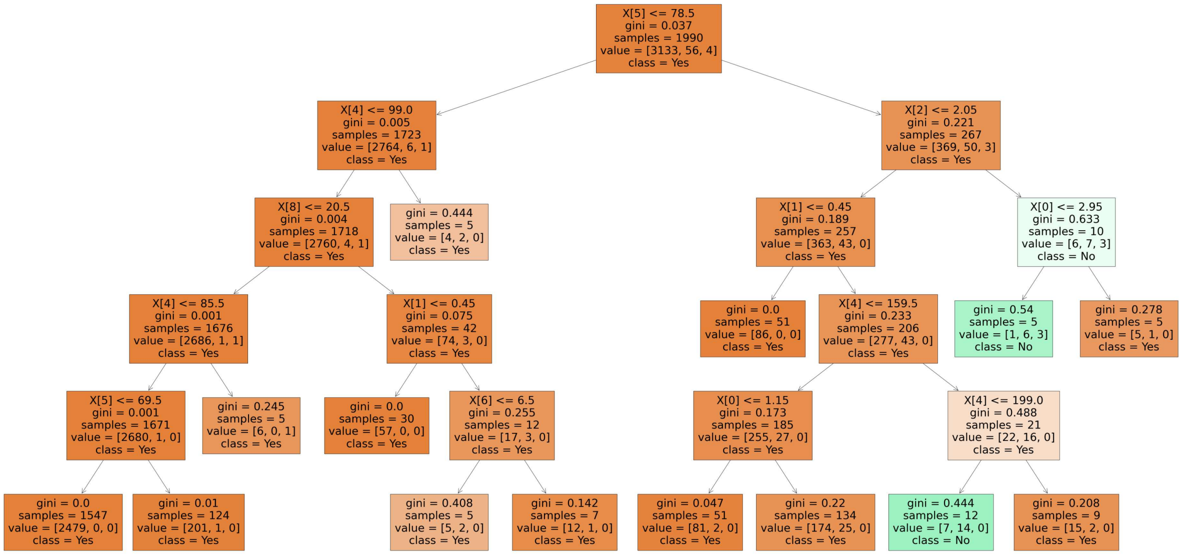
In [40]:
```
grid_search.best_score_
```

Out[40]:
```
0.9821483498979132
```

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=Tr
Out[42]: [Text(2466.9473684210525, 1993.2, 'X[5] <= 78.5\ngini = 0.037\nsamples = 1990
         \nvalue = [3133, 56, 4]\nclass = Yes'),
          Text(1409.6842105263158, 1630.8000000000002, 'X[4] <= 99.0\ngini = 0.005\nsa
         mples = 1723\nvalue = [2764, 6, 1]\nclass = Yes'),
          Text(1174.7368421052631, 1268.4, 'X[8] <= 20.5\ngini = 0.004\nsamples = 1718
         \nvalue = [2760, 4, 1]\nclass = Yes'),
          Text(704.8421052631579, 906.0, 'X[4] <= 85.5\ngini = 0.001\nsamples = 1676\n
         value = [2686, 1, 1]\nclass = Yes'),
          Text(469.89473684210526, 543.5999999999999, 'X[5] <= 69.5\ngini = 0.001\nsam
         ples = 1671\nvalue = [2680, 1, 0]\nclass = Yes'),
          Text(234.94736842105263, 181.19999999999982, 'gini = 0.0\nsamples = 1547\nva
         lue = [2479, 0, 0]\nclass = Yes'),
          Text(704.8421052631579, 181.19999999999982, 'gini = 0.01\nsamples = 124\nval
         ue = [201, 1, 0]\nclass = Yes'),
          Text(939.7894736842105, 543.5999999999999, 'gini = 0.245\nsamples = 5\nvalue
         = [6, 0, 1]\nclass = Yes'),
          Text(1644.6315789473683, 906.0, 'X[1] <= 0.45\ngini = 0.075\nsamples = 42\nv
         alue = [74, 3, 0]\nclass = Yes'),
          Text(1409.6842105263158, 543.5999999999999, 'gini = 0.0\nsamples = 30\nvalue
         = [57, 0, 0]\nclass = Yes'),
          Text(1879.578947368421, 543.5999999999999, 'X[6] <= 6.5\ngini = 0.255\nsampl
         es = 12\nvalue = [17, 3, 0]\nclass = Yes'),
          Text(1644.6315789473683, 181.19999999999982, 'gini = 0.408\nsamples = 5\nval
         ue = [5, 2, 0]\nclass = Yes'),
          Text(2114.5263157894738, 181.19999999999982, 'gini = 0.142\nsamples = 7\nval
         ue = [12, 1, 0]\nclass = Yes'),
          Text(1644.6315789473683, 1268.4, 'gini = 0.444\nsamples = 5\nvalue = [4, 2,
         0]\nclass = Yes'),
          Text(3524.2105263157896, 1630.8000000000002, 'X[2] <= 2.05\ngini = 0.221\nsa
         mples = 267\nvalue = [369, 50, 3]\nclass = Yes'),
          Text(3054.315789473684, 1268.4, 'X[1] <= 0.45\ngini = 0.189\nsamples = 257\n
         value = [363, 43, 0]\nclass = Yes'),
          Text(2819.3684210526317, 906.0, 'gini = 0.0\nsamples = 51\nvalue = [86, 0,
         0]\nclass = Yes'),
          Text(3289.2631578947367, 906.0, 'X[4] <= 159.5\ngini = 0.233\nsamples = 206
         \nvalue = [277, 43, 0]\nclass = Yes'),
          Text(2819.3684210526317, 543.5999999999999, 'X[0] <= 1.15\ngini = 0.173\nsam
         ples = 185\nvalue = [255, 27, 0]\nclass = Yes'),
          Text(2584.4210526315787, 181.19999999999982, 'gini = 0.047\nsamples = 51\nva
         lue = [81, 2, 0]\nclass = Yes'),
          Text(3054.315789473684, 181.19999999999982, 'gini = 0.22\nsamples = 134\nval
         ue = [174, 25, 0]\nclass = Yes'),
          Text(3759.157894736842, 543.5999999999999, 'X[4] <= 199.0\ngini = 0.488\nsam
         ples = 21\nvalue = [22, 16, 0]\nclass = Yes'),
          Text(3524.2105263157896, 181.19999999999982, 'gini = 0.444\nsamples = 12\nva
         lue = [7, 14, 0]\nclass = No'),
          Text(3994.1052631578946, 181.19999999999982, 'gini = 0.208\nsamples = 9\nval
         ue = [15, 2, 0]\nclass = Yes'),
          Text(3994.1052631578946, 1268.4, 'X[0] <= 2.95\ngini = 0.633\nsamples = 10\n
         value = [6, 7, 3]\nclass = No'),
          Text(3759.157894736842, 906.0, 'gini = 0.54\nsamples = 5\nvalue = [1, 6, 3]
         \nclass = No'),
          Text(4229.0526315789475, 906.0, 'gini = 0.278\nsamples = 5\nvalue = [5, 1,
         0]\nclass = Yes')]
```



```
Best model:LogisticRegression
```

In [ ]: