income-qualification

September 8, 2023

[4]:
```
import os
os.chdir('/home/nemesis/DatasetsIQ')
```

[5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[6]:
```
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
```

1 Exploratory Data Analysis

[7]:
```
print('Shape of train dataset is {}'.format(train.shape))
print('Shape of test dataset is {}'.format(test.shape))
```

```
Shape of train dataset is (9557, 143)
Shape of test dataset is (23856, 142)
```

[8]:
```
#Finding the target variable
for i in train.columns:
    if i not in test.columns:
        print("Our Target variable is {}".format(i))
```

```
Our Target variable is Target
```

[9]:
```
#Understanding the type of data preent in the dataset
print(train.dtypes.value_counts())
```

```
int64      130
float64      8
object       5
dtype: int64
```

[10]:
```
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 143 entries, Id to Target
dtypes: float64(8), int64(130), object(5)
memory usage: 10.4+ MB
None
```

From the above information we can see that the dataset contain mixed datatype.

[11]:
```python
for i in train.columns:
    a=train[i].dtype
    if a == 'object':
        print(i)
```

```
Id
idhogar
dependency
edjefe
edjefa
```

Data dictionary :

```
1. ID = Unique ID
2. idhogar, Household level identifier
3. dependency, Dependency rate, calculated = (number of members of the household younger than 1
4. edjefe, years of education of male head of household, based on the interaction of escolari
5. edjefa, years of education of female head of household, based on the interaction of escolari
```

[12]:
```python
#We can now drop the id variables
train.drop(['Id','idhogar'],axis=1,inplace=True)
```

[13]:
```python
train['dependency'].value_counts()
```

[13]:
```
yes          2192
no           1747
.5           1497
2             730
1.5           713
.33333334     598
.66666669     487
8             378
.25           260
3             236
4             100
.75            98
.2             90
.40000001      84
1.3333334      84
2.5            77
```

```
5               24
1.25            18
3.5             18
.80000001       18
2.25            13
.71428573       12
1.75            11
1.2             11
.83333331       11
.22222222       11
.2857143         9
1.6666666        8
.60000002        8
6                7
.16666667        7
Name: dependency, dtype: int64
```

2 Now we need to convert object variables

[14]:
```python
def map(i):

    if i=='yes':
        return(float(1))
    elif i=='no':
        return(float(0))
    else:
        return(float(i))
```

[15]:
```python
train['dependency']=train['dependency'].apply(map)
```

[16]:
```python
for i in train.columns:
    a=train[i].dtype
    if a == 'object':
        print(i)
```

```
edjefe
edjefa
```

[17]:
```python
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 141 entries, v2a1 to Target
dtypes: float64(9), int64(130), object(2)
memory usage: 10.3+ MB
```

[18]:
```
train['edjefe']=train['edjefe'].apply(map)
train['edjefa']=train['edjefa'].apply(map)
```

[19]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 141 entries, v2a1 to Target
dtypes: float64(11), int64(130)
memory usage: 10.3 MB
```

Now all the data is in numerical form.

3 Lets identify the variable with 0 variance

[20]:
```
var_df=pd.DataFrame(np.var(train,0),columns=['variance'])
var_df.sort_values(by='variance').head(15)
print('Below are columns with variance 0.')
col=list((var_df[var_df['variance']==0]).index)
print(col)
```

```
Below are columns with variance 0.
['elimbasu5']
```

elimbasu5 : 1 if rubbish disposal mainly by throwing in river, creek or sea.

Interpretation : *From above it is shown that all values of elimbasu5 is same so there is no variablity in dataset therefor we will drop this variable*

4 Checking for biases in the given dataset :

[21]:
```
contingency_tab=pd.crosstab(train['r4t3'],train['hogar_total'])
Observed_Values=contingency_tab.values
import scipy.stats
b=scipy.stats.chi2_contingency(contingency_tab)
Expected_Values = b[3]
no_of_rows=len(contingency_tab.iloc[0:2,0])
no_of_columns=len(contingency_tab.iloc[0,0:2])
df=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",df)
from scipy.stats import chi2
chi_square=sum([(o-e)**2./e for o,e in zip(Observed_Values,Expected_Values)])
chi_square_statistic=chi_square[0]+chi_square[1]
print("chi-square statistic:-",chi_square_statistic)
alpha=0.05
critical_value=chi2.ppf(q=1-alpha,df=df)
print('critical_value:',critical_value)
```

```
p_value=1-chi2.cdf(x=chi_square_statistic,df=df)
print('p-value:',p_value)
print('Significance level: ',alpha)
print('Degree of Freedom: ',df)
print('chi-square statistic:',chi_square_statistic)
print('critical_value:',critical_value)
print('p-value:',p_value)
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")
```

```
Degree of Freedom:- 1
chi-square statistic:- 17022.072400560897
critical_value: 3.841458820694124
p-value: 0.0
Significance level:  0.05
Degree of Freedom:  1
chi-square statistic: 17022.072400560897
critical_value: 3.841458820694124
p-value: 0.0
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
```

The variables, r4t3, hogar_total have relationship between them and hence we can use any one of them for good results.

[22]:
```
contingency_tab=pd.crosstab(train['tipovivi3'],train['v2a1'])
Observed_Values=contingency_tab.values
import scipy.stats
b=scipy.stats.chi2_contingency(contingency_tab)
Expected_Values = b[3]
no_of_rows=len(contingency_tab.iloc[0:2,0])
no_of_columns=len(contingency_tab.iloc[0,0:2])
df=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",df)
from scipy.stats import chi2
chi_square=sum([(o-e)**2./e for o,e in zip(Observed_Values,Expected_Values)])
chi_square_statistic=chi_square[0]+chi_square[1]
print("chi-square statistic:-",chi_square_statistic)
alpha=0.05
critical_value=chi2.ppf(q=1-alpha,df=df)
print('critical_value:',critical_value)
```

```
p_value=1-chi2.cdf(x=chi_square_statistic,df=df)
print('p-value:',p_value)
print('Significance level: ',alpha)
print('Degree of Freedom: ',df)
print('chi-square statistic:',chi_square_statistic)
print('critical_value:',critical_value)
print('p-value:',p_value)
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")
```

```
Degree of Freedom:- 1
chi-square statistic:- 54.04781105990782
critical_value: 3.841458820694124
p-value: 1.9562129693895258e-13
Significance level:  0.05
Degree of Freedom:  1
chi-square statistic: 54.04781105990782
critical_value: 3.841458820694124
p-value: 1.9562129693895258e-13
Reject H0,There is a relationship between 2 categorical variables
Reject H0,There is a relationship between 2 categorical variables
```

The variables, tipovivi3, v2a1 have relationship between them and hence we can use any one of them for good results.

[23]:
```
contingency_tab=pd.crosstab(train['v18q'],train['v18q1'])
Observed_Values=contingency_tab.values
import scipy.stats
b=scipy.stats.chi2_contingency(contingency_tab)
Expected_Values = b[3]
no_of_rows=len(contingency_tab.iloc[0:2,0])
no_of_columns=len(contingency_tab.iloc[0,0:2])
df=(no_of_rows-1)*(no_of_columns-1)
print("Degree of Freedom:-",df)
from scipy.stats import chi2
chi_square=sum([(o-e)**2./e for o,e in zip(Observed_Values,Expected_Values)])
chi_square_statistic=chi_square[0]+chi_square[1]
print("chi-square statistic:-",chi_square_statistic)
alpha=0.05
critical_value=chi2.ppf(q=1-alpha,df=df)
print('critical_value:',critical_value)
```

```
p_value=1-chi2.cdf(x=chi_square_statistic,df=df)
print('p-value:',p_value)
print('Significance level: ',alpha)
print('Degree of Freedom: ',df)
print('chi-square statistic:',chi_square_statistic)
print('critical_value:',critical_value)
print('p-value:',p_value)
if chi_square_statistic>=critical_value:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")

if p_value<=alpha:
    print("Reject H0,There is a relationship between 2 categorical variables")
else:
    print("Retain H0,There is no relationship between 2 categorical variables")
```

```
Degree of Freedom:- 0
chi-square statistic:- 0.0
critical_value: nan
p-value: nan
Significance level:  0.05
Degree of Freedom:  0
chi-square statistic: 0.0
critical_value: nan
p-value: nan
Retain H0,There is no relationship between 2 categorical variables
Retain H0,There is no relationship between 2 categorical variables
```

The variables, v18q ,v18q1 have relationship between them and hence we can use any one of them for good results.

Conclusion : Therefore biases exist.

[26]:
```
train.drop('r4t3',axis=1,inplace=True)
```

#Now we check if there is a house with no family head

"parentesco1" =1 if household head

[28]:
```
train.parentesco1.value_counts()
```

[28]:
```
0    6584
1    2973
Name: parentesco1, dtype: int64
```

[29]:
```
pd.crosstab(train['edjefa'],train['edjefe'])
```

```
[29]: edjefe   0.0   1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  …  12.0  \
      edjefa                                                             …
      0.0       435   123   194   307   137   222  1845   234   257   486  …   113
      1.0        69     0     0     0     0     0     0     0     0     0  …     0
      2.0        84     0     0     0     0     0     0     0     0     0  …     0
      3.0       152     0     0     0     0     0     0     0     0     0  …     0
      4.0       136     0     0     0     0     0     0     0     0     0  …     0
      5.0       176     0     0     0     0     0     0     0     0     0  …     0
      6.0       947     0     0     0     0     0     0     0     0     0  …     0
      7.0       179     0     0     0     0     0     0     0     0     0  …     0
      8.0       217     0     0     0     0     0     0     0     0     0  …     0
      9.0       237     0     0     0     0     0     0     0     0     0  …     0
      10.0       96     0     0     0     0     0     0     0     0     0  …     0
      11.0      399     0     0     0     0     0     0     0     0     0  …     0
      12.0       72     0     0     0     0     0     0     0     0     0  …     0
      13.0       52     0     0     0     0     0     0     0     0     0  …     0
      14.0      120     0     0     0     0     0     0     0     0     0  …     0
      15.0      188     0     0     0     0     0     0     0     0     0  …     0
      16.0      113     0     0     0     0     0     0     0     0     0  …     0
      17.0       76     0     0     0     0     0     0     0     0     0  …     0
      18.0        3     0     0     0     0     0     0     0     0     0  …     0
      19.0        4     0     0     0     0     0     0     0     0     0  …     0
      20.0        2     0     0     0     0     0     0     0     0     0  …     0
      21.0        5     0     0     0     0     0     0     0     0     0  …     0

      edjefe  13.0  14.0  15.0  16.0  17.0  18.0  19.0  20.0  21.0
      edjefa
      0.0      103   208   285   134   202    19    14     7    43
      1.0        0     0     0     0     0     0     0     0     0
      2.0        0     0     0     0     0     0     0     0     0
      3.0        0     0     0     0     0     0     0     0     0
      4.0        0     0     0     0     0     0     0     0     0
      5.0        0     0     0     0     0     0     0     0     0
      6.0        0     0     0     0     0     0     0     0     0
      7.0        0     0     0     0     0     0     0     0     0
      8.0        0     0     0     0     0     0     0     0     0
      9.0        0     0     0     0     0     0     0     0     0
      10.0       0     0     0     0     0     0     0     0     0
      11.0       0     0     0     0     0     0     0     0     0
      12.0       0     0     0     0     0     0     0     0     0
      13.0       0     0     0     0     0     0     0     0     0
      14.0       0     0     0     0     0     0     0     0     0
      15.0       0     0     0     0     0     0     0     0     0
      16.0       0     0     0     0     0     0     0     0     0
      17.0       0     0     0     0     0     0     0     0     0
      18.0       0     0     0     0     0     0     0     0     0
      19.0       0     0     0     0     0     0     0     0     0
```

```
20.0        0       0       0       0       0       0       0       0       0
21.0        0       0       0       0       0       0       0       0       0

[22 rows x 22 columns]
```

Conclusion : From the above table we can see that there are 0 male heads and 0 female heads, therefore there are 435 families with 0 heads.

[30]:
```
#Checking for null values
train.isna().sum().value_counts()
```

[30]:
```
0       135
5         2
6860      1
7342      1
7928      1
dtype: int64
```

[31]:
```
train['Target'].isna().sum()
```

[31]:
```
0
```

Conclusion : There is no null values present in target and henceforth we can now fill null values of other variables if they are present.

[33]:
```
float_col=[]
for i in train.columns:
    a=train[i].dtype
    if a == 'float64':
        float_col.append(i)
print(float_col)
```

```
['v2a1', 'v18q1', 'rez_esc', 'dependency', 'edjefe', 'edjefa', 'meaneduc',
'overcrowding', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned']
```

[34]:
```
train[float_col].isna().sum()
```

[34]:
```
v2a1               6860
v18q1              7342
rez_esc            7928
dependency            0
edjefe                0
edjefa                0
meaneduc              5
overcrowding          0
SQBovercrowding       0
SQBdependency         0
SQBmeaned             5
```

```
dtype: int64
```

[35]:
```
train['v18q1'].value_counts()
```

[35]:
```
1.0    1586
2.0     444
3.0     129
4.0      37
5.0      13
6.0       6
Name: v18q1, dtype: int64
```

[36]:
```
pd.crosstab(train['tipovivi1'],train['v2a1'])
```

[36]:
```
v2a1       0.0        12000.0    13000.0    14000.0    15000.0    16000.0    \
tipovivi1
0                 29          3          4          3          3          2

v2a1       17000.0    20000.0    23000.0    25000.0    …  570540.0   \
tipovivi1                                              …
0                  4         22          5         21  …         25

v2a1       600000.0   620000.0   684648.0   700000.0   770229.0   800000.0   \
tipovivi1
0                 11          3          3          7          3          4

v2a1       855810.0   1000000.0  2353477.0
tipovivi1
0                 11          7          2

[1 rows x 157 columns]
```

[37]:
```
pd.crosstab(train['v18q1'],train['v18q'])
```

[37]:
```
v18q      1
v18q1
1.0    1586
2.0     444
3.0     129
4.0      37
5.0      13
6.0       6
```

Conclusion and the action that should follow : 'v2a1', 'v18q1', and'rez_esc' all have greater than 50% null values because, in the case of 'v18q1', some families may own their own homes and, in such case, would not be required to pay rent; similarly, some families may own '0' tablets.

We can also drop tipovivi3,v18q 1. tipovivi3, =1 rented 2. v18q, owns a tablet as v2a1 is enough to show both as the variable v18q1 can show that if respondent owns a tablet or not

[38]:
```
train['v2a1'].fillna(0,inplace=True)
train['v18q1'].fillna(0,inplace=True)
```

[39]:
```
train.drop(['tipovivi3', 'v18q','rez_esc','elimbasu5'],axis=1,inplace=True)
```

[40]:
```
train['meaneduc'].fillna(np.mean(train['meaneduc']),inplace=True)
train['SQBmeaned'].fillna(np.mean(train['SQBmeaned']),inplace=True)
print(train.isna().sum().value_counts())
```

```
0    136
dtype: int64
```

[41]:
```
int_col=[]
for i in train.columns:
    a=train[i].dtype
    if a == 'int64':
        int_col.append(i)
print(int_col)
```

```
['hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'r4h1', 'r4h2', 'r4h3', 'r4m1',
'r4m2', 'r4m3', 'r4t1', 'r4t2', 'tamhog', 'tamviv', 'escolari', 'hhsize',
'paredblolad', 'paredzocalo', 'paredpreb', 'pareddes', 'paredmad', 'paredzinc',
'paredfibras', 'paredother', 'pisomoscer', 'pisocemento', 'pisoother',
'pisonatur', 'pisonotiene', 'pisomadera', 'techozinc', 'techoentrepiso',
'techocane', 'techootro', 'cielorazo', 'abastaguadentro', 'abastaguafuera',
'abastaguano', 'public', 'planpri', 'noelec', 'coopele', 'sanitario1',
'sanitario2', 'sanitario3', 'sanitario5', 'sanitario6', 'energcocinar1',
'energcocinar2', 'energcocinar3', 'energcocinar4', 'elimbasu1', 'elimbasu2',
'elimbasu3', 'elimbasu4', 'elimbasu6', 'epared1', 'epared2', 'epared3',
'etecho1', 'etecho2', 'etecho3', 'eviv1', 'eviv2', 'eviv3', 'dis', 'male',
'female', 'estadocivil1', 'estadocivil2', 'estadocivil3', 'estadocivil4',
'estadocivil5', 'estadocivil6', 'estadocivil7', 'parentesco1', 'parentesco2',
'parentesco3', 'parentesco4', 'parentesco5', 'parentesco6', 'parentesco7',
'parentesco8', 'parentesco9', 'parentesco10', 'parentesco11', 'parentesco12',
'hogar_nin', 'hogar_adul', 'hogar_mayor', 'hogar_total', 'instlevel1',
'instlevel2', 'instlevel3', 'instlevel4', 'instlevel5', 'instlevel6',
'instlevel7', 'instlevel8', 'instlevel9', 'bedrooms', 'tipovivi1', 'tipovivi2',
'tipovivi4', 'tipovivi5', 'computer', 'television', 'mobilephone',
'qmobilephone', 'lugar1', 'lugar2', 'lugar3', 'lugar4', 'lugar5', 'lugar6',
'area1', 'area2', 'age', 'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
'SQBhogar_nin', 'agesq', 'Target']
```

[42]:
```
train[int_col].isna().sum().value_counts()
```

[42]:
```
0    126
dtype: int64
```

Conclusion : No null values present.

[43]:
```
train.Target.value_counts()
```

[43]:
```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```

Setting poverty level for the members as well as the head same.

Now, those living below the poverty line may pay a lower rent and not buy a home. Additionally, whether a house is in an urban or rural area affects the answer.

[44]:
```
Poverty_level=train[train['v2a1'] !=0]
```

[45]:
```
Poverty_level.shape
```

[45]:
```
(2668, 136)
```

[46]:
```
poverty_level=Poverty_level.groupby('area1')['v2a1'].apply(np.median)
```

[47]:
```
poverty_level
```

[47]:
```
area1
0     80000.0
1    140000.0
Name: v2a1, dtype: float64
```

Note : 1. If renters in rural areas pay less than 80000 per month, they are considered to be living in poverty. 2. If renters in urban areas pay less than 140000 per month, they are considered to be living in poverty.

[48]:
```
def povert(x):
    if x<80000:
        return('Below poverty level')

    elif x>140000:
        return('Above poverty level')
    elif x<140000:
        return('Below poverty level: Ur-ban ; Above poverty level : Rural ')
```

[49]:
```
c=Poverty_level['v2a1'].apply(povert)
```

[50]:
```
c.shape
```

[50]:
```
(2668,)
```

[51]:
```python
pd.crosstab(c,Poverty_level['area1'])
```

[51]:
```
area1                                                    0     1
v2a1
Above poverty level                                    139  1103
Below poverty level                                    208   418
Below poverty level: Ur-ban ; Above poverty lev…   98   663
```

1. Rural : Above poverty level : 139 Below poverty level : 208
2. Urban : Above poverty level : 1103 Below poverty level : 663

[52]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

[53]:
```python
X_data=train.drop('Target',axis=1)
Y_data=train.Target
```

[54]:
```python
X_data_col=X_data.columns
```

Applying standard scaling

[55]:
```python
from sklearn.preprocessing import StandardScaler
SS=StandardScaler()
X_data_1=SS.fit_transform(X_data)
X_data_1=pd.DataFrame(X_data_1,columns=X_data_col)
```

Model fitting phase

[57]:
```python
X_train,X_test,Y_train,Y_test=train_test_split(X_data_1,Y_data,test_size=0.
 ↪25,stratify=Y_data,random_state=0)
```

Idfentification of best parameters using GridSearchCV

[59]:
```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

rfc=RandomForestClassifier(random_state=0)
parameters={'n_estimators':[10,50,100,300],'max_depth':[3,5,10,15]}
grid=zip([rfc],[parameters])

best_=None

for i, j in grid:
    a=GridSearchCV(i,param_grid=j,cv=3,n_jobs=1)
    a.fit(X_train,Y_train)
    if best_ is None:
        best_=a
    elif a.best_score_>best_.best_score_:
```

```
        best_=a


print ("Best CV Score",best_.best_score_)
print ("Model Parameters",best_.best_params_)
print("Best Estimator",best_.best_estimator_)
```

```
Best CV Score 0.8507046183898423
Model Parameters {'max_depth': 15, 'n_estimators': 300}
Best Estimator RandomForestClassifier(max_depth=15, n_estimators=300,
random_state=0)
```

[60]:

```
RFC=best_.best_estimator_
Model=RFC.fit(X_train,Y_train)
pred=Model.predict(X_test)
```

[61]:

```
print('Model Score of train data : {}'.format(Model.score(X_train,Y_train)))
print('Model Score of test data : {}'.format(Model.score(X_test,Y_test)))
```

```
Model Score of train data : 0.9831170643225896
Model Score of test data : 0.8824267782426778
```

[62]:

```
Important_features=pd.DataFrame(Model.
 ↪feature_importances_,X_data_col,columns=['feature_importance'])
```

[63]:

```
Top50Features=Important_features.
 ↪sort_values(by='feature_importance',ascending=False).head(50).index
```

[64]:

```
Top50Features
```

[64]:

```
Index(['SQBmeaned', 'meaneduc', 'SQBdependency', 'dependency', 'overcrowding',
       'SQBovercrowding', 'qmobilephone', 'SQBhogar_nin', 'SQBedjefe',
       'edjefe', 'hogar_nin', 'rooms', 'cielorazo', 'r4t1', 'v2a1', 'edjefa',
       'agesq', 'r4m3', 'r4h2', 'SQBage', 'age', 'escolari', 'r4t2', 'r4h3',
       'hogar_adul', 'SQBescolari', 'eviv3', 'bedrooms', 'r4m1', 'epared3',
       'r4m2', 'tamviv', 'paredblolad', 'v18q1', 'SQBhogar_total', 'tamhog',
       'hhsize', 'hogar_total', 'pisomoscer', 'etecho3', 'r4h1', 'lugar1',
       'eviv2', 'tipovivi1', 'energcocinar2', 'energcocinar3', 'epared2',
       'television', 'area2', 'area1'],
      dtype='object')
```

[65]:

```
for i in Top50Features:
    if i not in X_data_col:
        print(i)
```

[66]:

```
X_data_Top50=X_data[Top50Features]
```

[67]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X_data_Top50,Y_data,test_size=0.
    ↪25,stratify=Y_data,random_state=0)
```

[68]:
```
Model_1=RFC.fit(X_train,Y_train)
pred=Model_1.predict(X_test)
```

[70]:
```
from sklearn.metrics import confusion_matrix,f1_score,accuracy_score
confusion_matrix(Y_test,pred)
```

[70]:
```
array([[ 143,   17,    0,   29],
       [   8,  324,    4,   63],
       [   1,   12,  214,   75],
       [   2,   10,    3, 1485]])
```

[71]:
```
f1_score(Y_test,pred,average='weighted')
```

[71]:
```
0.9026906492316511
```

[72]:
```
accuracy_score(Y_test,pred)
```

[72]:
```
0.906276150627615
```

[73]:
```
#Now we will clean test data and apply prediction after that and we will also␣
    ↪drop the Id variables
test.drop('r4t3',axis=1,inplace=True)
test.drop(['Id','idhogar'],axis=1,inplace=True)
test['dependency']=test['dependency'].apply(map)
test['edjefe']=test['edjefe'].apply(map)
test['edjefa']=test['edjefa'].apply(map)
```

[74]:
```
test['v2a1'].fillna(0,inplace=True)
test['v18q1'].fillna(0,inplace=True)
```

[75]:
```
test.drop(['tipovivi3', 'v18q','rez_esc','elimbasu5'],axis=1,inplace=True)
```

[76]:
```
train['meaneduc'].fillna(np.mean(train['meaneduc']),inplace=True)
train['SQBmeaned'].fillna(np.mean(train['SQBmeaned']),inplace=True)
```

[77]:
```
test_data=test[Top50Features]
```

[78]:
```
test_data.isna().sum().value_counts()
```

[78]:
```
0     48
31     2
dtype: int64
```

[79]:
```
test_data.SQBmeaned.fillna(np.mean(test_data['SQBmeaned']),inplace=True)
```

```
/tmp/ipykernel_11915/1933955761.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_data.SQBmeaned.fillna(np.mean(test_data['SQBmeaned']),inplace=True)
```

[80]:
```
test_data.meaneduc.fillna(np.mean(test_data['meaneduc']),inplace=True)
```

```
/tmp/ipykernel_11915/1212364859.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_data.meaneduc.fillna(np.mean(test_data['meaneduc']),inplace=True)
```

[81]:
```
Test_data_1=SS.fit_transform(test_data)
X_data_1=pd.DataFrame(Test_data_1)
```

[82]:
```
test_prediction=Model_1.predict(test_data)
```

[83]:
```
test_prediction
```

[83]:
```
array([4, 4, 4, …, 4, 4, 4])
```

Conclusion : Above is the prediction for the test data.

4.0.1 Conclusion : With random forest we can predict the test data with an accuracy of 90%(approx.)

[ ]:
```
```