Machine Learning Project - Income Qualification¶

Submitted By: Shahroo Akhtar

Problem Statement Scenario¶

Many social programs have a hard time ensuring that the right people are given enough aid. Itâ□□s tricky when a program focuses on the poorest segment of the population. This segment of the population canâ□□t provide the necessary income and expense records to prove that they qualify.

In Latin America, a popular method called Proxy Means Test (PMT) uses an algorithm to verify income qualification. With PMT, agencies use a model that considers a familyâ□□s observable household attributes like the material of their walls and ceiling or the assets found in their homes to classify them and predict their level of need.

While this is an improvement, accuracy remains a problem as the regionâ□□s population grows and poverty declines.

The Inter-American Development Bank (IDB)believes that new methods beyond traditional econometrics, based on a dataset of Costa Rican household characteristics, might help improve PMTâ□□s performance.

Task to be performed on the data provided:¶

- Identify the output variable.
- Understand the type of data.
- Check if there are any biases in your dataset.
- Check whether all members of the house have the same poverty level.
- Check if there is a house without a family head.
- Set poverty level of the members and the head of the house within a family.
- Count how many null values are existing in columns.
- Remove null value rows of the target variable.
- Predict the accuracy using random forest classifier.
- Check the accuracy using random forest with cross validation.

Explanation:¶

- The data provided, is already divided into two parts:**Train & Test**
- Each row represents one **individual**
- Each column is a **feature** and have unique values pretaining to the particular individual or household.
- The training set has an additional column called "Target"
- The target column contains poverty level on a scale of 1-4
- Level value **1** represents extreme poverty.

Problem Type:¶

- This is a **Supervised Multi-Class Classification** problem as we are provided with labels and the labels are classified into 4 classes.

Import necessary Libraries¶

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
```

Importing the Datasets¶

In [2]:

```
df_train = pd.read_csv('/Users/apple/Desktop/train.csv')
df_test = pd.read_csv('/Users/apple/Desktop/test.csv')
```

Explore the dataset (Train)¶

In [3]:

```
df_train.head()
```

Out[3]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBhogar_nin	SQBovercr
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	0	1.000000
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	4489	1	144	0	1.000000
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	0	0.250000
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	289	16	121	4	1.777778
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	1369	16	121	4	1.777778

5 rows Ã□ 143 columns

In [4]:

```
df_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
```

Explore the dataset(Test)¶

In [5]:

```
df_test.head()
```

Out[5]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBhogar_nin	SQBov
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	...	4	0	16	9	0	1	2.25
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	...	41	256	1681	9	0	1	2.25
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	...	41	289	1681	9	0	1	2.25
3	ID_a8db26a79	NaN	0	14	0	1	1	1	1.0	0	...	59	256	3481	1	256	0	1.00
4	ID_a62966799	175000.0	0	4	0	1	1	1	1.0	0	...	18	121	324	1	0	1	0.25

5 rows Ã□ 142 columns

Output Variable¶

As mentioned above too that the output variable in this scenario is the "**Target**" Feature column that is only present in the train dataset.

In [6]:

```
for i in df_train.columns:
    if i not in df_test.columns:
        print ('Output Variable is {}'.format(i))
```

```
Output Variable is Target
```

In [7]:

```
df_train.Target.value_counts()
```

Out[7]:

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

Understanding Data Types¶

In [8]:

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23856 entries, 0 to 23855
Columns: 142 entries, Id to agesq
dtypes: float64(8), int64(129), object(5)
memory usage: 25.8+ MB
```

Important: There is no "Target" feature column in the test dataset.

In [9]:

```
df_train.dtypes
```

Out[9]:

```
Id                  object
v2a1               float64
hacdor               int64
rooms                int64
hacapo               int64
                    ...
SQBovercrowding    float64
SQBdependency      float64
SQBmeaned          float64
agesq                int64
Target               int64
Length: 143, dtype: object
```

In [10]:

```
print (df_train.dtypes.value_counts())
```

```
int64      130
float64      8
```

```
object       5
dtype: int64
```

In [11]:

```
df_test.dtypes
```

Out[11]:

```
Id                  object
v2a1               float64
hacdor               int64
rooms                int64
hacapo               int64
                    ...
SQBhogar_nin         int64
SQBovercrowding    float64
SQBdependency      float64
SQBmeaned          float64
agesq                int64
Length: 142, dtype: object
```

In [12]:

```
print (df_test.dtypes.value_counts())
```

```
int64      129
float64      8
object       5
dtype: int64
```

We have mixed data types.

- float64 : 8 variables
- int64 : 129(Test) & 130(Train)
- object : 5 variables

Data Cleaning¶

Lets check our data for Null values

In [13]:

```
df_train.isnull().sum()
```

Out[13]:

```
Id                    0
v2a1               6860
hacdor                0
rooms                 0
hacapo                0
                   ...
SQBovercrowding       0
SQBdependency         0
SQBmeaned             5
agesq                 0
Target                0
Length: 143, dtype: int64
```

In [14]:

```
df_train.columns[df_train.isnull().any()]
```

Out[14]:

```
Index(['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'SQBmeaned'], dtype='object')
```

In [15]:

```
df_train.select_dtypes(include=['object']).head()
```

Out[15]:

	Id	idhogar	dependency	edjefe	edjefa
0	ID_279628684	21eb7fcc1	no	10	no
1	ID_f29eb3ddd	0e5d7a658	8	12	no
2	ID_68de51c94	2c7317ea8	8	no	11
3	ID_d671db89c	2b58d945f	yes	11	no
4	ID_d56d6f5f5	2b58d945f	yes	11	no

Result: We noticed that we have no null values for float64 and int64 dtypes, but we have a lot for object datatype. Also we see that there is alot mixed values for data features dependancy, edijefe and edjefa

The documentation provided for the above columns is as follows:

- The '**Id**' and '**idhogar**' are identifying variables.
- dependency: Dependency rate, calculated = (number of members of the household younger than 19 or older than 64)/(number of member of household between 19 and 64)
- edjefe: years of education of male head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0
- edjefa: years of education of female head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0

'Yes' = 1 and No = '2' for these 3 variables. We will use the **map** function

In [16]:

```
mapping = {"yes": 1, "no": 0}

for df in [df_train, df_test]:
    df['dependency'] = df['dependency'].replace(mapping).astype(np.float64)
    df['edjefa'] = df['edjefa'].replace(mapping).astype(np.float64)
    df['edjefe'] = df['edjefe'].replace(mapping).astype(np.float64)

df_train[['dependency', 'edjefa', 'edjefe']].describe()
```

Out[16]:

	dependency	edjefa	edjefe
count	9557.000000	9557.000000	9557.000000
mean	1.149550	2.896830	5.096788
std	1.605993	4.612056	5.246513
min	0.000000	0.000000	0.000000
25%	0.333333	0.000000	0.000000
50%	0.666667	0.000000	6.000000
75%	1.333333	6.000000	9.000000
max	8.000000	21.000000	21.000000

In [17]:

```
df_train['dependency'].head()
```

Out[17]:

```
0    0.0
1    8.0
2    8.0
3    1.0
4    1.0
Name: dependency, dtype: float64
```

Join Datasets (Train & Test)¶

Lets join the two datasets together so we can perform the operations on both the data sets at the same time. Later we can separate. We will also add a null column named "Target" to the test dataset

In [18]:

```
##df_test['Target'] = np.nan
##df_data = pd.concat(objs = [df_train,df_test], axis = 0).reset_index(drop = True)
##df_data.info()
```

Fix Null Values:¶

Now we will fix the columns with null values

As previously seen there are 5 columns with Null values and as per the documentation provided, they are as follows :

- v2a1 (total nulls: 6860) : Monthly rent payment
- v18q1 (total nulls: 7342) : number of tablets household owns
- rez_esc (total nulls: 7928) : Years behind in school
- meaneduc (total nulls: 5) : average years of education for adults (18+)
- SQBmeaned (total nulls: 5) : square of the mean years of education of adults (>=18) in the household 142

v2a1: Monthly Rent payment¶

In [19]:

```
new = df_train[df_train['v2a1'].isnull()].head()

columns=['tipovivi1','tipovivi2','tipovivi3','tipovivi4','tipovivi5']
new[columns]
```

Out[19]:

	tipovivi1	tipovivi2	tipovivi3	tipovivi4	tipovivi5
2	1	0	0	0	0
13	1	0	0	0	0
14	1	0	0	0	0
26	1	0	0	0	0
32	1	0	0	0	0

In order to understand the missing values from this column, we have to consider the distribution of tipovivi_.This column shows the ownership/renting status of the home.

In [20]:

```
# Declare a variable for home ownership
```

```
owns = [x for x in df_train if x.startswith('tipo')]

# Plot of the home ownership variables for home missing rent payments
df_train.loc[df_train['v2a1'].isnull(), owns].sum().plot.bar(figsize = (10, 8),
                                                              color = 'Blue',
                                                              edgecolor = 'k',
                                                               linewidth = 2);
plt.xticks([0, 1, 2, 3, 4],
           ['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precarious', 'Other'],
          rotation = 20)
plt.title('Households with Missing Rent Payments', size = 18);
```

The meaning of home ownership variable :

- tipovivi1, =1 own and fully paid house
- tipovivi2, "=1 own, paying in installments"
- tipovivi3, =1 rented
- tipovivi4, =1 precarious
- tipovivi5, "=1 other(assigned, borrowed)"

Looking at the plot we can notice that if the house is fully paid, there will be no monthly payments. We will add 0 for all the null values.

In [21]:

```
for df in [df_train,df_test]:
    df['v2a1'].fillna(value = 0, inplace = True)

df_train[['v2a1']].isnull().sum()
```

Out[21]:

```
v2a1    0
dtype: int64
```

v18q1: Number of tablets household owns¶

It indicates the number of tablets a family owns. Since this is a household variable, it makes sense to look at it on a household level.We will only select the rows for the head of the household.

In [22]:

```
heads = df_train.loc[df_train['parentesco1'] == 1].copy()
heads.groupby('v18q')['v18q1'].apply(lambda x: x.isnull().sum())
```

Out[22]:

```
v18q
0    2318
1       0
Name: v18q1, dtype: int64
```

In [23]:

```
plt.figure(figsize = (8, 6))
col='v18q1'
df_train[col].value_counts().sort_index().plot.bar(color = 'blue',
                                                   edgecolor = 'k',
                                                   linewidth = 2)
plt.xlabel(f'{col}'); plt.title(f'{col} Value Counts'); plt.ylabel('Count')
plt.show();
```

If the tablet column is 0, it means that the household owns no tablets We will add 0 for all the null values

In [24]:

```
for df in [df_train,df_test]:
    df['v18q1'].fillna(value=0, inplace=True)

df_train[['v18q1']].isnull().sum()
```

Out[24]:

```
v18q1    0
dtype: int64
```

rez_esc : Years behind in school¶

Another colomn with considerable missing values is rez_esc, that is years behind school. If the family has a null value, it is a possibilty that the family doesnt have any children in school.We will make a comparison between the ages of who do not have a missing value with ones who have missing values.

In [25]:

```
df_train.loc[df_train['rez_esc'].notnull()]['age'].describe()
```

Out[25]:

```
count    1629.000000
mean       12.258441
std         3.218325
min         7.000000
25%         9.000000
```

```
50%        12.000000
75%        15.000000
max        17.000000
Name: age, dtype: float64
```

It shows that the oldest age with missing value is 17. Lets look at the ages of those who have a missing value.

In [26]:

```
df_train.loc[df_train['rez_esc'].isnull()]['age'].describe()
```

Out[26]:

```
count    7928.000000
mean       38.833249
std        20.989486
min         0.000000
25%        24.000000
50%        38.000000
75%        54.000000
max        97.000000
Name: age, dtype: float64
```

If the person is over 19 and they have a missing value or if they are younger than 7 and have a missing value, we can set it to 0.

In [27]:

```
df_train['rez_esc'].fillna(0, inplace = True)
df_train[['rez_esc']].isnull().sum()
```

Out[27]:

```
rez_esc    0
dtype: int64
```

meaneduc : Average years of education for adults (18+)¶

For these missing values, we have to consider the following columns:

- **edjefe**: years of education of male head of household, based on another column named **escolari** i.e. years of education. If head of the household and gender, yes=1 and no=0
- **edjefa**: years of education of female head of household, based on another column named **escolari** i.e. years of education. If head of the household and gender, yes=1 and no=0
- instlevel1= 1, no level of education
- instlevel2= 2, incomplete primary

meaneduc is null when no level of education is 0. We will fix the data.

In [28]:

```
for df in [df_train,df_test]:
    df['meaneduc'].fillna(value=0, inplace=True)
df_train[['meaneduc']].isnull().sum()
```

Out[28]:

```
meaneduc    0
dtype: int64
```

SQBmeaned : square of the mean years of education of adults (>=18) in the household 142¶

For these missing values, we have to consider the following columns:

- **edjefe**: years of education of male head of household, based on another column named **escolari** i.e. years of education. If head of the household and gender, yes=1 and no=0
- **edjefa**: years of education of female head of household, based on another column named **escolari** i.e. years of education. If head of the household and gender, yes=1 and no=0
- instlevel1= 1, no level of education
- instlevel2= 2, incomplete primary

SQBmeaned is null when no level of education is 0. We will fix the data.

In [29]:

```
for df in [df_train,df_test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
df_train[['SQBmeaned']].isnull().sum()
```

Out[29]:

```
SQBmeaned    0
dtype: int64
```

House without a family head¶

For this we will check the 'idhogar', i.e. household level identifier, column against the target column. We do this in order to check if household records match with a corresponding score in the target. We will use the train data for this and now onwards.

In [30]:

```
##For unique records
similar_records = df_train.groupby('idhogar')['Target'].apply(lambda x:x.nunique() == 1)
#for different target values
not_similar_records = similar_records[similar_records != True]
print ('{} Households are with different records in target for family members'.format(len(not_similar_records)))
```

```
85 Households are with different records in target for family members
```

We will use the target value from parentesco1, that is if household has a head. We will use the value and we will update the rest of the missing values.

In [31]:

```
heads = df_train.groupby('idhogar')['parentesco1'].sum()
#Now lets find if we still have missing values
no_heads = df_train.loc[df_train['idhogar'].isin(heads[heads == 0].index), :]
print ('{} Households without a head.'.format(no_heads['idhogar'].nunique()))
```

```
15 Households without a head.
```

Now we will find the different target value of households without a head.

In [32]:

```
no_similar_heads = no_heads.groupby('idhogar')['Target'].apply(lambda x: x.nunique()== 1)
print ('{} Households with no head and different target values.'.format(sum(no_similar_heads == False)))
```

```
0 Households with no head and different target values.
```

Lets check Bias in the dataset¶

Lets check the target value

1- Extremely poverty 2- Moderate poverty 3- Vulnerable Households 4- Non Valnerable Households

In [33]:

```
target_counts = df_train['Target'].value_counts()
target_counts
```

Out[33]:

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

In [34]:

```
target_counts.plot.bar(figsize = (8, 6),linewidth = 2,edgecolor = 'k',title="Target vs Total_Count")
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1bdcd510>
```

It shows that the dataset is baised as extreme poverty is the smallest count in the dataset.

Set the poverty level of the members and head of the house in a family¶

People below poverty level can be paying less rent and dont own a house. It also depends on whether a house is in urban or rural areas.

In [35]:

```
df_train['v2a1'].isna().value_counts()
```

Out[35]:

```
False    9557
Name: v2a1, dtype: int64
```

Feature Selection¶

Lets look at the Squared Variables and remove them

- â□□SQBescolariâ□□
- â□□SQBageâ□□
- â□□SQBhogar_totalâ□□
- â□□SQBedjefeâ□□
- â□□SQBhogar_ninâ□□
- â□□SQBovercrowdingâ□□
- â□□SQBdependencyâ□□
- â□□SQBmeanedâ□□
- â□□agesqâ□□

In [36]:

```
print(df_train.shape)
cols=['SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
        'SQBhogar_nin', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq']


for df in [df_train, df_test]:
    df.drop(columns = cols,inplace=True)

print(df_train.shape)
```

```
(9557, 143)
(9557, 134)
```

In [37]:

```
#Lets define the variable categories
id_ = ['Id', 'idhogar', 'Target']

ind_bool = ['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil2', 'estadocivil3',
            'estadocivil4', 'estadocivil5', 'estadocivil6', 'estadocivil7',
            'parentesco1', 'parentesco2',  'parentesco3', 'parentesco4', 'parentesco5',
            'parentesco6', 'parentesco7', 'parentesco8',  'parentesco9', 'parentesco10',
            'parentesco11', 'parentesco12', 'instlevel1', 'instlevel2', 'instlevel3',
            'instlevel4', 'instlevel5', 'instlevel6', 'instlevel7', 'instlevel8',
            'instlevel9', 'mobilephone']

ind_ordered = ['rez_esc', 'escolari', 'age']

hh_bool = ['hacdor', 'hacapo', 'v14a', 'refrig', 'paredblolad', 'paredzocalo',
           'paredpreb','pisocemento', 'pareddes', 'paredmad',
           'paredzinc', 'paredfibras', 'paredother', 'pisomoscer', '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',
           'elimbasu5', 'elimbasu6', 'epared1', 'epared2', 'epared3',
           'etecho1', 'etecho2', 'etecho3', 'eviv1', 'eviv2', 'eviv3',
           'tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi4', 'tipovivi5',
           'computer', 'television', 'lugar1', 'lugar2', 'lugar3',
           'lugar4', 'lugar5', 'lugar6', 'area1', 'area2']

hh_ordered = [ 'rooms', 'r4h1', 'r4h2', 'r4h3', 'r4m1','r4m2','r4m3', 'r4t1',  'r4t2',
              'r4t3', 'v18q1', 'tamhog','tamviv','hhsize','hogar_nin',
              'hogar_adul','hogar_mayor','hogar_total',  'bedrooms', 'qmobilephone']

hh_cont = ['v2a1', 'dependency', 'edjefe', 'edjefa', 'meaneduc', 'overcrowding']
```

In [38]:

```
#Check for redundant household variables
heads = df_train.loc[df_train['parentesco1'] == 1, :]
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

Out[38]:

```
(2973, 98)
```

In [39]:

```
# Create correlation matrix
corr_matrix = heads.corr()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(abs(upper[column]) > 0.95)]
to_drop
```

Out[39]:

```
['coopele', 'area2', 'tamhog', 'hhsize', 'hogar_total']
```

In [40]:

```
corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].abs() > 0.9]
```

Out[40]:

	r4t3	tamhog	tamviv	hhsize	hogar_total
r4t3	1.000000	0.996884	0.929237	0.996884	0.996884
tamhog	0.996884	1.000000	0.926667	1.000000	1.000000
tamviv	0.929237	0.926667	1.000000	0.926667	0.926667
hhsize	0.996884	1.000000	0.926667	1.000000	1.000000
hogar_total	0.996884	1.000000	0.926667	1.000000	1.000000

In [41]:

```
sns.heatmap(corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].abs() > 0.9],
                    annot=True, cmap = plt.cm.YlGnBu_r, fmt='.3f');
```

Variables elated to households

- r4t3, Total persons in the household

- tamhog, size of the household
- tamviv, number of persons living in the household
- hhsize, household size
- hogar_total, # of total individuals in the household

These variables are all highly correlated with one another.

In [42]:

```
cols=['tamhog', 'hogar_total', 'r4t3']
for df in [df_train, df_test]:
    df.drop(columns= cols,inplace=True)

df_train.shape
```

Out[42]:

```
(9557, 131)
```

In [43]:

```
#Check for redundant Individual variables
individual = df_train[id_ + ind_bool + ind_ordered]
individual.shape
```

Out[43]:

```
(9557, 39)
```

In [44]:

```
# Create correlation matrix
corr_matrix = individual.corr()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(abs(upper[column]) > 0.95)]
to_drop
```

Out[44]:

```
['female']
```

In [45]:

```
# This is simply the opposite of male! We can remove the male flag.
for df in [df_train, df_test]:
    df.drop(columns = 'male',inplace=True)

df_train.shape
```

Out[45]:

```
(9557, 130)
```

Now will check the area1 and area2 where

- area1 = zona urbana
- area2 = zona rural We will remove area2 as it is redundant too

In [46]:

```
for df in [df_train, df_test]:
    df.drop(columns = 'area2',inplace=True)

df_train.shape
```

Out[46]:

```
(9557, 129)
```

Finally lets delete 'Id','idhogar'

In [47]:

```
cols=['Id','idhogar']
for df in [df_train, df_test]:
    df.drop(columns = cols,inplace=True)

df_train.shape
```

Out[47]:

```
(9557, 127)
```

In [48]:

```
for df in [df_train, df_test]:
    df['v2a1'].fillna(value=0, inplace=True)

df_train[['v2a1']].isnull().sum()
df_test[['v2a1']].isnull().sum()
```

Out[48]:

```
v2a1    0
dtype: int64
```

In [49]:

```
df_train[~df_train.isin([np.nan, np.inf, -np.inf]).any(1)]
df_test[~df_test.isin([np.nan, np.inf, -np.inf]).any(1)]
```

Out[49]:

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...	mobilephone	qmobilephone	lugar1	lugar2	lugar3	lugar4	lugar5	lugar6	area1	age
0	0.0	0	5	0	1	1	0	0.0	1	1	...	1	2	1	0	0	0	0	0	1	4
1	0.0	0	5	0	1	1	0	0.0	1	1	...	1	2	1	0	0	0	0	0	1	41
2	0.0	0	5	0	1	1	0	0.0	1	1	...	1	2	1	0	0	0	0	0	1	41
3	0.0	0	14	0	1	1	1	1.0	0	1	...	1	2	1	0	0	0	0	0	1	59
4	175000.0	0	4	0	1	1	1	1.0	0	0	...	1	1	1	0	0	0	0	0	1	18
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23851	0.0	1	2	1	1	1	0	0.0	0	2	...	1	1	0	0	0	0	0	1	0	10
23852	0.0	0	3	0	1	1	0	0.0	0	1	...	1	2	0	0	0	0	0	1	0	54
23853	0.0	0	3	0	1	1	0	0.0	0	1	...	1	2	0	0	0	0	0	1	0	12
23854	0.0	0	3	0	1	1	0	0.0	0	1	...	1	2	0	0	0	0	0	1	0	12
23855	0.0	0	3	0	1	1	0	0.0	0	1	...	1	2	0	0	0	0	0	1	0	51

23856 rows Ã□ 126 columns

In [50]:

```
df_test.count()
```

Out[50]:

```
v2a1      23856
hacdor    23856
rooms     23856
hacapo    23856
v14a      23856
          ...
lugar4    23856
lugar5    23856
lugar6    23856
area1     23856
age       23856
Length: 126, dtype: int64
```

In [51]:

```
np.isnan(df_test)
```

Out[51]:

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...	mobilephone	qmobilephone	lugar1	lugar2	lugar3	lugar4	lugar5	lugar6	area1	age
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23851	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
23852	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
23853	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
23854	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
23855	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

23856 rows Ã□ 126 columns

In [52]:

```
df_test.isnull().sum()
```

Out[52]:

```
v2a1      0
hacdor    0
rooms     0
hacapo    0
v14a      0
         ..
lugar4    0
lugar5    0
lugar6    0
area1     0
age       0
Length: 126, dtype: int64
```

Implementing the Model¶

We will use the **Random forest classifier** to predict the accuracy.

Lets import all the necessary libraries

In [53]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
```

In [54]:

```
X_data=df_train.drop('Target',axis=1)
Y_data=df_train.Target
```

In [55]:

```
X_data_col=X_data.columns
```

Applying Standard Scalling to dataset¶

In [56]:

```
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¶

In [57]:

```
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)
```

Identify best parameters for our model using GridSearchCv¶

In [58]:

```
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 0.8575415096972234
Model Parameters {'max_depth': 15, 'n_estimators': 300}
Best Estimator RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=15, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=300,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [59]:

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

In [60]:

```
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 : 0.9753034742570112
Model Score of test data : 0.8828451882845189
```

In [61]:

```
Important_features=pd.DataFrame(Model.feature_importances_,X_data_col,columns=['feature_importance'])
Top50Features=Important_features.sort_values(by='feature_importance',ascending=False).head(50).index
Top50Features
```

Out[61]:

```
Index(['meaneduc', 'dependency', 'overcrowding', 'hogar_nin', 'edjefe',
       'qmobilephone', 'rooms', 'r4t1', 'cielorazo', 'escolari', 'age', 'r4t2',
```

```
       'r4m3', 'hhsize', 'v2a1', 'edjefa', 'tamviv', 'r4h2', 'hogar_adul',
       'r4h3', 'eviv3', 'r4m1', 'bedrooms', 'r4m2', 'pisomoscer', 'r4h1',
       'etecho3', 'epared3', 'paredblolad', 'v18q', 'v18q1', 'lugar1',
       'energcocinar3', 'energcocinar2', 'area1', 'tipovivi1', 'pisocemento',
       'hogar_mayor', 'paredpreb', 'etecho2', 'epared2', 'eviv2', 'television',
       'etecho1', 'lugar3', 'sanitario3', 'eviv1', 'epared1', 'paredmad',
       'elimbasu1'],
      dtype='object')
```

In [62]:

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

X_data_Top50=X_data[Top50Features]
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)
Model_1=RFC.fit(X_train,Y_train)
pred=Model_1.predict(X_test)
```

In [63]:

```
from sklearn.metrics import confusion_matrix,f1_score,accuracy_score
confusion_matrix(Y_test,pred)
```

Out[63]:

```
array([[ 147,   17,    0,   25],
       [   6,  335,    2,   56],
       [   0,   12,  225,   65],
       [   2,    8,    5, 1485]])
```

In [64]:

```
f1_score(Y_test,pred,average='weighted')
```

Out[64]:

```
0.9144866370546362
```

In [65]:

```
accuracy_score(Y_test,pred)
```

Out[65]:

```
0.9171548117154812
```

Lets clean the test dataset and then predict¶

In [66]:

```
test_data=df_test[Top50Features]
```

In [67]:

```
test_data.isna().sum().value_counts()
```

Out[67]:

```
0    50
dtype: int64
```

In [68]:

```
Test_data_1=SS.fit_transform(test_data)
X_data_1=pd.DataFrame(Test_data_1)
```

In [69]:

```
test_prediction=Model_1.predict(test_data)
```

In [70]:

```
test_prediction
```

Out[70]:

```
array([4, 4, 4, ..., 4, 4, 4])
```

Using Cross validation to check accuracy¶

In [71]:

```
from sklearn.model_selection import KFold,cross_val_score
```

In [72]:

```
seed=7
kfold=KFold(n_splits=5,random_state=seed,shuffle=True)

rmclassifier=RandomForestClassifier(random_state=10,n_jobs = -1)
print(cross_val_score(rmclassifier,X_data,Y_data,cv=kfold,scoring='accuracy'))
results=cross_val_score(rmclassifier,X_data,Y_data,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.92834728 0.93357741 0.92883307 0.92203035 0.92935636]
92.84288932824498
```

In [73]:

```
#Now for 100 trees
num_trees= 100

rmclassifier=RandomForestClassifier(n_estimators=100, random_state=10,n_jobs = -1)
print(cross_val_score(rmclassifier,X_data,Y_data,cv=kfold,scoring='accuracy'))
results=cross_val_score(rmclassifier,X_data,Y_data,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.92834728 0.93357741 0.92883307 0.92203035 0.92935636]
92.84288932824498
```

y_predict_test = rmclassifier.predict(X_test) y_predict_test

In [75]:

```
rmclassifier.fit(X_data,Y_data)
labels = list(X_data)
feature_importances = pd.DataFrame({'feature': labels, 'importance': rmclassifier.feature_importances_})
feature_importances=feature_importances[feature_importances.importance>0.015]
feature_importances.head()
```

Out[75]:

	feature	importance
0	v2a1	0.019105
2	rooms	0.023847
9	r4h2	0.018925
10	r4h3	0.018820
12	r4m2	0.016106

In [76]:

```
feature_importances.sort_values(by=['importance'], ascending=True, inplace=True)
feature_importances['positive'] = feature_importances['importance'] > 0
feature_importances.set_index('feature',inplace=True)
feature_importances.head()

feature_importances.importance.plot(kind='barh', figsize=(11, 6),color = feature_importances.positive.map({True: 'blue', False: 'red'}))
plt.xlabel('Importance')
```

Out[76]:

```
Text(0.5, 0, 'Importance')
```