Identify the level of income qualification needed for the families in Latin America.

Problem Statement Scenario: Many social programs have a hard time ensuring that the right people are given enough aid. It's tricky when a program focuses on the poorest segment of the population. This segment of the population can'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 family'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 region'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 PMT's performance. Following actions should be performed:

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.

In [22]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
sns.set()
```

In [23]:
```
income_train = pd.read_csv("train.csv")
income_test = pd.read_csv("test.csv")
```

In [24]:
```
income_train.head()
```

Out[24]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121

5 rows × 143 columns

In [25]:
```
income_test.head()
```

Out[25]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQBescc
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	...	4	
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	...	41	
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	...	41	
3	ID_a8db26a79	NaN	0	14	0	1	1	1	1.0	0	...	59	
4	ID_a62966799	175000.0	0	4	0	1	1	1	1.0	0	...	18	

5 rows × 142 columns

In [26]:
```
income_train.info
```

Out[26]:
```
<bound method DataFrame.info of                Id      v2a1  hacdor  rooms  hacapo  v14
a  refrig  v18q  \
0     ID_279628684  190000.0       0      3       0     1       1     0
1     ID_f29eb3ddd  135000.0       0      4       0     1       1     1
2     ID_68de51c94       NaN       0      8       0     1       1     0
3     ID_d671db89c  180000.0       0      5       0     1       1     1
4     ID_d56d6f5f5  180000.0       0      5       0     1       1     1
...            ...       ...     ...    ...     ...   ...     ...   ...
9552  ID_d45ae367d   80000.0       0      6       0     1       1     0
9553  ID_c94744e07   80000.0       0      6       0     1       1     0
9554  ID_85fc658f8   80000.0       0      6       0     1       1     0
9555  ID_ced540c61   80000.0       0      6       0     1       1     0
9556  ID_a38c64491   80000.0       0      6       0     1       1     0

      v18q1  r4h1  ...  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  \
0       NaN     0  ...          100    1849               1        100
1       1.0     0  ...          144    4489               1        144
2       NaN     0  ...          121    8464               1          0
3       1.0     0  ...           81     289              16        121
4       1.0     0  ...          121    1369              16        121
...     ...   ...  ...          ...     ...             ...        ...
9552    NaN     0  ...           81    2116              25         81
9553    NaN     0  ...            0       4              25         81
9554    NaN     0  ...           25    2500              25         81
9555    NaN     0  ...          121     676              25         81
9556    NaN     0  ...           64     441              25         81

      SQBhogar_nin  SQBovercrowding  SQBdependency  SQBmeaned  agesq  Target
0                0         1.000000         0.0000   100.0000   1849       4
1                0         1.000000        64.0000   144.0000   4489       4
2                0         0.250000        64.0000   121.0000   8464       4
3                4         1.777778         1.0000   121.0000    289       4
4                4         1.777778         1.0000   121.0000   1369       4
...            ...              ...            ...        ...    ...     ...
9552             1         1.562500         0.0625    68.0625   2116       2
9553             1         1.562500         0.0625    68.0625      4       2
9554             1         1.562500         0.0625    68.0625   2500       2
9555             1         1.562500         0.0625    68.0625    676       2
9556             1         1.562500         0.0625    68.0625    441       2
```

```
[9557 rows x 143 columns]>
```

In [27]:
```
income_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
```

In [28]:
```
income_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
```

In [29]:
```
#datatypes-column map in train set
print('Integer Type: ')
print(income_train.select_dtypes(np.int64).columns)
print('\n')
print('Float Type: ')
print(income_train.select_dtypes(np.float64).columns)
print('\n')
print('Object Type: ')
print(income_train.select_dtypes(np.object).columns)
```

```
Integer Type:
Index(['hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q', 'r4h1', 'r4h2',
       'r4h3', 'r4m1',
       ...
       'area1', 'area2', 'age', 'SQBescolari', 'SQBage', 'SQBhogar_total',
       'SQBedjefe', 'SQBhogar_nin', 'agesq', 'Target'],
      dtype='object', length=130)


Float Type:
Index(['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'overcrowding',
       'SQBovercrowding', 'SQBdependency', 'SQBmeaned'],
      dtype='object')


Object Type:
Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'], dtype='object')
```

In [30]:
```
income_train.select_dtypes('int64').head()
```

Out[30]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	...	area1	area2	age	SQBes
0	0	3	0	1	1	0	0	1	1	0	...	1	0	43	
1	0	4	0	1	1	1	0	1	1	0	...	1	0	67	
2	0	8	0	1	1	0	0	0	0	0	...	1	0	92	
3	0	5	0	1	1	1	0	2	2	1	...	1	0	17	
4	0	5	0	1	1	1	0	2	2	1	...	1	0	37	

5 rows × 130 columns

In [31]:
```
income_train.select_dtypes('float64').head()
```

Out[31]:

	v2a1	v18q1	rez_esc	meaneduc	overcrowding	SQBovercrowding	SQBdependency	SQBmeaned
0	190000.0	NaN	NaN	10.0	1.000000	1.000000	0.0	100.0
1	135000.0	1.0	NaN	12.0	1.000000	1.000000	64.0	144.0
2	NaN	NaN	NaN	11.0	0.500000	0.250000	64.0	121.0
3	180000.0	1.0	1.0	11.0	1.333333	1.777778	1.0	121.0
4	180000.0	1.0	NaN	11.0	1.333333	1.777778	1.0	121.0

In [33]:
```
income_train.select_dtypes('object').head()
```

Out[33]:

	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

In [34]:
```
# null values in columns
null_counts=income_train.select_dtypes('int64').isnull().sum()
null_counts[null_counts > 0]
```

Out[34]:
```
Series([], dtype: int64)
```

In [35]:
```
null_counts=income_train.select_dtypes('float64').isnull().sum()
null_counts[null_counts > 0]
```

Out[35]:
```
v2a1         6860
v18q1        7342
rez_esc      7928
meaneduc        5
SQBmeaned       5
dtype: int64
```

In [36]:
```
null_counts=income_train.select_dtypes('object').isnull().sum()
null_counts[null_counts > 0]
```

Out[36]:
```
Series([], dtype: int64)
```

1. The columns object data types are having both numeric and text values 2. Null values found only for float type as mentioned below v2a1 6860 v18q1 7342 rez_esc 7928 meaneduc 5 SQBmeaned 5 hence Data cleaning should be performed on these values

In [37]:
```
#Data cleaning
#converting the columns with both numeric and text values to numeric values

mapping={'yes':1,'no':0}

for df in [income_train, income_test]:
```

```
    df['dependency'] =df['dependency'].replace(mapping).astype(np.float64)
    df['edjefe'] =df['edjefe'].replace(mapping).astype(np.float64)
    df['edjefa'] =df['edjefa'].replace(mapping).astype(np.float64)

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

Out[37]:

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

#cleaning null values As per Data dictionary: v2a1 : Monthly rent payment v18q1 : number of tablets household owns rez_esc : Years behind in school meaneduc : average years of education for adults (18+) SQBmeaned : square of the mean years of education of adults (>=18) in the householdColumns linked to v2a1 : Monthly rent payment tipovivi1 = own and fully paid house tipovivi2 = own, paying in installments" tipovivi3 = rented tipovivi4 = precarious tipovivi5 = other(assigned, borrowed) (for all the above columns Values are 1 if true or 0 if not true)

In [38]:
```
data = income_train[income_train['v2a1'].isnull()].head()

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

Out[38]:

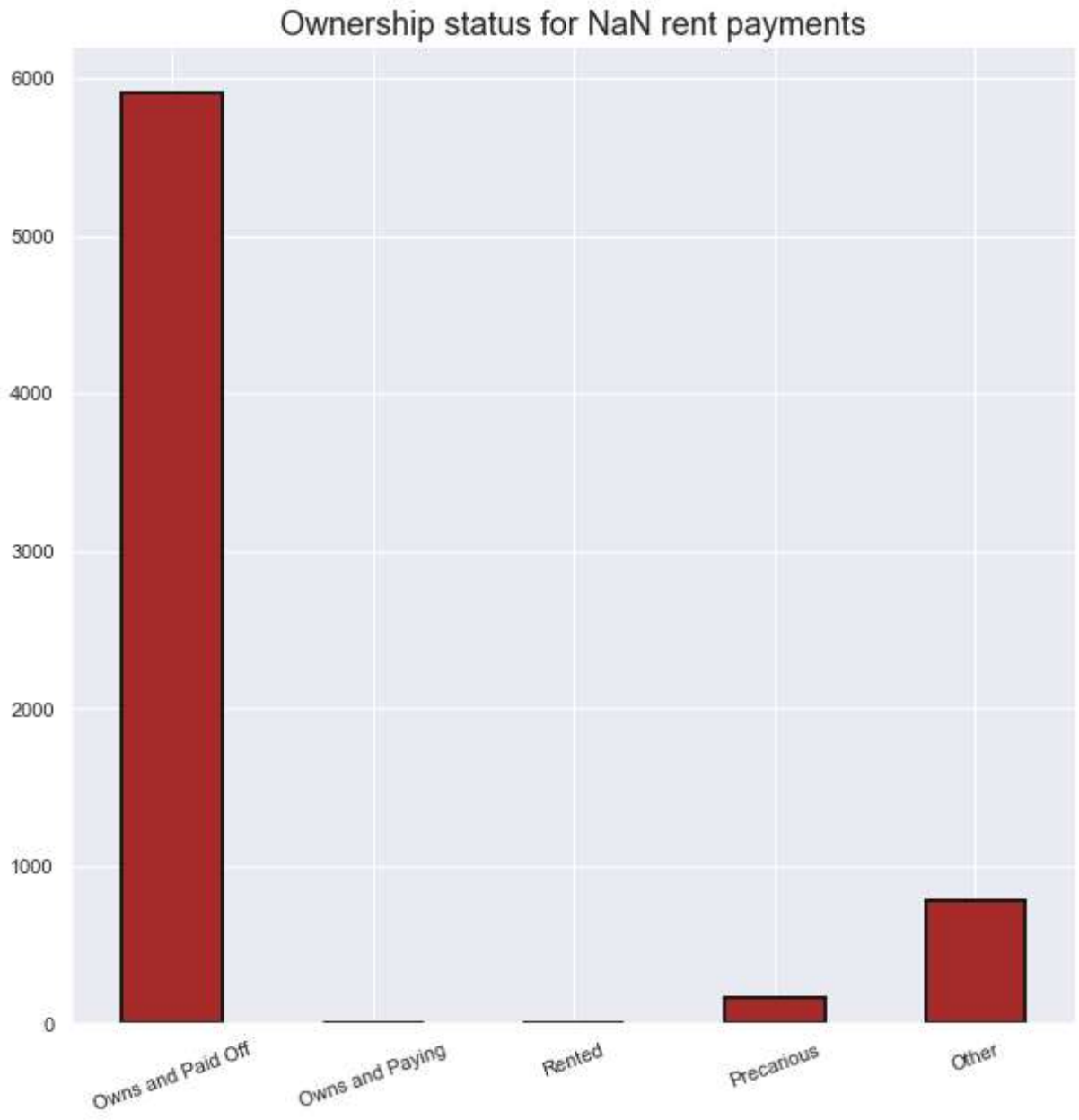
	tipovivi1	tipovivi2	tipovivi3	tipovivi4	tipovivi5
2	1	0	0	0	0
13	1	0	0	0	0
14	1	0	0	0	0
26	1	0	0	0	0
32	1	0	0	0	0

In [39]:
```
# Variables indicating home ownership
ownership = [x for x in income_train if x.startswith('tipo')]
```

In [40]:
```
ownership
```

Out[40]:
```
['tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi4', 'tipovivi5']
```

In [41]:
```
# Plot of the home ownership variables for home missing rent payments
income_train.loc[income_train['v2a1'].isnull(), ownership].sum().plot.bar(figsize = (10
plt.xticks([0, 1, 2, 3, 4], ['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precari
plt.title('Ownership status for NaN rent payments', size = 18);
```



In [42]:
```
#since house is not rented -  rent need not be paid hence changing values to '0'
for df in [income_train, income_test]:
    df['v2a1'].fillna(value=0, inplace=True)

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

Out[42]:
```
v2a1    0
dtype: int64
```

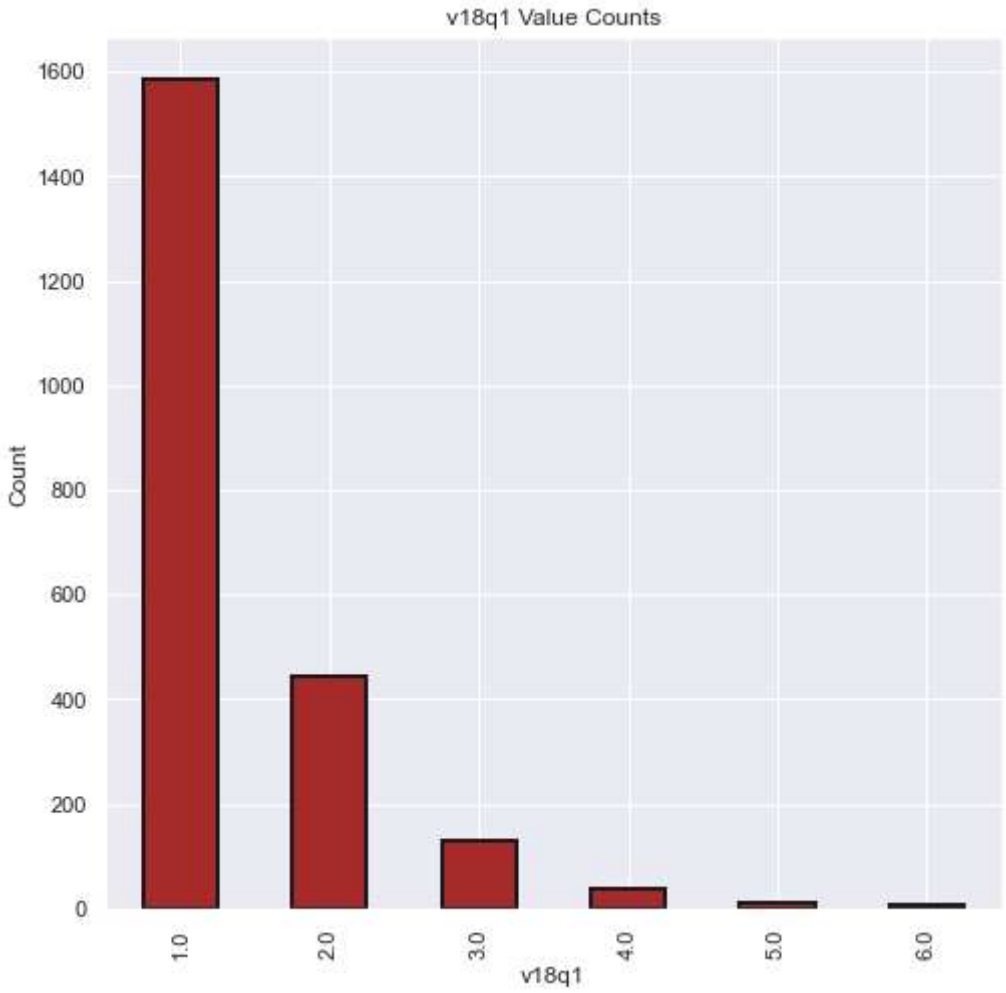
v18q1 : number of tablets household owns v18q : owns a tablet verifying if NaN in v18q1 holds any owns a tablet or not

In [43]:
```
heads = income_train.loc[income_train['parentesco1'] == 1].copy()
heads.groupby('v18q')['v18q1'].apply(lambda x: x.isnull().sum())
```

Out[43]:
```
v18q
0    2318
1       0
Name: v18q1, dtype: int64
```

In [44]:
```
plt.figure(figsize = (8, 8))
```

```
income_train['v18q1'].value_counts().sort_index().plot.bar(color = 'brown', edgecolor =
plt.xlabel('v18q1')
plt.title('v18q1 Value Counts')
plt.ylabel('Count')
plt.show()
```



When atleast tablet is owned v18q value is 1, when no tablet is owned v18q is 0 for all the NaN values v18q is 0 hence v18q1 is also 0.

In [45]:
```
for df in [income_train, income_test]:
    df['v18q1'].fillna(value=0, inplace=True)

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

Out[45]:
```
v18q1    0
dtype: int64
```

rez_esc : Years behind in school identify if there is any minimum age for school and if NaN is rez_esc is falling under that category

In [46]:
```
income_train[income_train['rez_esc'].notnull()]['age'].describe()
```

Out[46]:
```
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
```

Minimum age for school is 7 years

In [47]:
```
income_train.loc[income_train['rez_esc'].isnull()]['age'].describe()
```

Out[47]:
```
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
```

In [48]:
```
income_train.loc[(income_train['rez_esc'].isnull() & ((income_train['age'] > 7) & (inco
```

Out[48]:
```
count     1.0
mean     10.0
std       NaN
min      10.0
25%      10.0
50%      10.0
75%      10.0
max      10.0
Name: age, dtype: float64
```

In [49]:
```
income_train[(income_train['age'] ==10) & income_train['rez_esc'].isnull()].head()
```

Out[49]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolar
2514	ID_f012e4242	160000.0	0	6	0	1	1	1	1.0	0	...	

1 rows × 143 columns

In [50]:
```
income_train[(income_train['Id'] =='ID_f012e4242')].head()
```

Out[50]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolar
2514	ID_f012e4242	160000.0	0	6	0	1	1	1	1.0	0	...	

1 rows × 143 columns

In [51]:
```
school = income_train[income_train['Id'] =='ID_f012e4242']
```

In [52]:
```
school['instlevel1']
```

Out[52]:
```
2514    1
Name: instlevel1, dtype: int64
```

Since the instlevel1 = 1 - no level of education for the id = ID_f012e4242. rez_esc should be 0. for rest all rez_esc with NaN value, Since age is not in the bound of 7 to 17 the values can be changed to 0.

In [53]:
```
for df in [income_train, income_test]:
    df['rez_esc'].fillna(value=0, inplace=True)
```

In [54]:
```
income_train[['rez_esc']].isnull().sum()
```

Out[54]:
```
rez_esc    0
dtype: int64
```

meaneduc : average years of education for adults (18+) number of null values are 5

In [55]:
```
income_train[income_train['meaneduc'].isnull()].head()
```

Out[55]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescola
1291	ID_bd8e11b0f	0.0	0	7	0	1	1	0	0.0	0	...	10
1840	ID_46ff87316	110000.0	0	1	0	1	1	0	0.0	0	...	3
1841	ID_69f50bf3e	110000.0	0	1	0	1	1	0	0.0	0	...	1
2049	ID_db3168f9f	180000.0	0	3	0	1	1	0	0.0	0	...	14
2050	ID_2a7615902	180000.0	0	3	0	1	1	0	0.0	0	...	14

5 rows × 143 columns

In [56]:
```
#edjefe, years of education of male head of household, based on the interaction of esco
# edjefa, years of education of female head of household, based on the interaction of e
# instlevel1 =1 no level of education
# instlevel2 =1 incomplete primary
data = income_train[income_train['meaneduc'].isnull()].head()
columns=['edjefe','edjefa','instlevel1','instlevel2']
data[columns][data[columns]['instlevel1']>0].describe()
```

Out[56]:

	edjefe	edjefa	instlevel1	instlevel2
count	0.0	0.0	0.0	0.0
mean	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN

In [57]:
```
#fixing NaN with 0 since no education is found
for df in [income_train, income_test]:
    df['meaneduc'].fillna(value=0, inplace=True)
income_train[['meaneduc']].isnull().sum()
```

Out[57]:
```
meaneduc    0
dtype: int64
```

In [58]:
```
#SQBmeaned  (total nulls: 5) : square of the mean years of education of adults (>=18) i
#following the previous steps
data = income_train[income_train['SQBmeaned'].isnull()].head()
columns=['edjefe','edjefa','instlevel1','instlevel2']
data[columns][data[columns]['instlevel1']>0].describe()
```

Out[58]:

	edjefe	edjefa	instlevel1	instlevel2
count	0.0	0.0	0.0	0.0
mean	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN
min	NaN	NaN	NaN	NaN
25%	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN

In [59]:
```
for df in [income_train, income_test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
income_train[['SQBmeaned']].isnull().sum()
```

Out[59]:
```
SQBmeaned    0
dtype: int64
```

In [60]:
```
#final check for nulls
null_counts = income_train.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

Out[60]:
```
Series([], dtype: int64)
```

In [62]:
```
# 1.Identify the output variable as per the target score
# Groupby the household and figure out the number of unique values
all_equal = income_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)

# Households where targets are not all equal
not_equal = all_equal[all_equal != True]
print('There are {} households where all the family members do not have the same target
```

```
There are 85 households where all the family members do not have the same target.
```

In [64]:
```
#Lets check one household
income_train[income_train['idhogar'] == not_equal.index[0]][['idhogar', 'parentesco1',
```

Out[64]:

	idhogar	parentesco1	Target
7651	0172ab1d9	0	3
7652	0172ab1d9	0	2
7653	0172ab1d9	0	3
7654	0172ab1d9	1	3
7655	0172ab1d9	0	2

In [65]:
```
# 1. check if all families has a head.
# 2. use Target value of the parent record (head of the household) and update rest.

households_head = income_train.groupby('idhogar')['parentesco1'].sum()
```

```
#  check households without a head
households_no_head = income_train.loc[income_train['idhogar'].isin(households_head[hous

print('There are {} households without a head.'.format(households_no_head['idhogar'].nu
```

```
There are 15 households without a head.
```

In [66]:
```
# Find households without a head and where Target value are different
households_no_head_equal = households_no_head.groupby('idhogar')['Target'].apply(lambda
print('{} Households with no head have different Target value.'.format(sum(households_n
```

```
0 Households with no head have different Target value.
```

In [67]:
```
#Set poverty level of the members and the head of the house within a family.
# Iterate through each household
for household in not_equal.index:
    # Find the correct label (for the head of household)
    true_target = int(income_train[(income_train['idhogar'] == household) & (income_tra

    # Set the correct label for all members in the household
    income_train.loc[income_train['idhogar'] == household, 'Target'] = true_target


# Groupby the household and figure out the number of unique values
all_equal = income_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)

# Households where targets are not all equal
not_equal = all_equal[all_equal != True]
print('There are {} households where all the family members do not have the same target
```

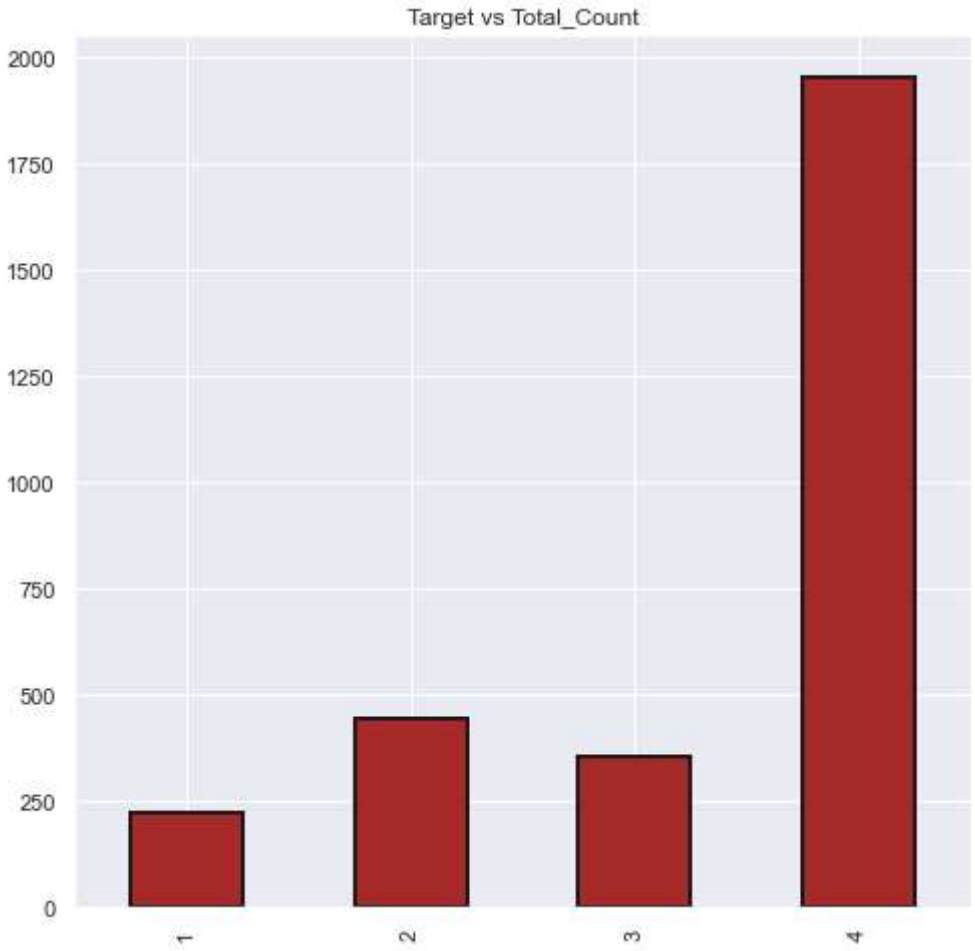
```
There are 0 households where all the family members do not have the same target.
```

In [68]:
```
# dataset and plot head of household and Target
# 1 = extreme poverty 2 = moderate poverty 3 = vulnerable households 4 = non vulnerable
target_counts = heads['Target'].value_counts().sort_index()
target_counts
```

Out[68]:
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

In [72]:
```
target_counts.plot.bar(color = 'brown', figsize = (8, 8),linewidth = 2,edgecolor = 'k',
```

Out[72]:
```
<AxesSubplot:title={'center':'Target vs Total_Count'}>
```



The dataset is biased since extreme poverty is the smallest count in the train dataset. Removing Squared Variables: SQBescolari SQBage SQBhogar_total SQBedjefe SQBhogar_nin SQBovercrowding SQBdependency SQBmeaned agesq

In [73]:
```
print(income_train.shape)
cols=['SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
       'SQBhogar_nin', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq']


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

print(income_train.shape)
```

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

In [74]:
```
id_ = ['Id', 'idhogar', 'Target']

ind_bool = ['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil2', 'estadociv
            '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 [75]:
```
#Check for redundant household variables
heads = income_train.loc[income_train['parentesco1'] == 1, :]
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

Out[75]: (2973, 98)

In [76]:
```
# 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[76]: ['coopele', 'area2', 'tamhog', 'hhsize', 'hogar_total']

In [77]:
```
corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].abs() > 0.9]
```

Out[77]:

	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 [78]:
```
sns.heatmap(corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].ab
            annot=True, cmap = plt.cm.Accent_r, fmt='.3f');
```

In [79]:
```
# There are several variables here having to do with the size of the house:
# 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 [80]:
```
cols=['tamhog', 'hogar_total', 'r4t3']
for df in [income_train, income_test]:
    df.drop(columns = cols,inplace=True)
income_train.shape
```

Out[80]: (9557, 131)

In [82]:
```
#Check for redundant Individual variables
ind = income_train[id_ + ind_bool + ind_ordered]
ind.shape
```

Out[82]: (9557, 39)

In [83]:
```
# Create correlation matrix
corr_matrix = ind.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[83]: ['female']

In [84]:
```
# This is simply the opposite of male! We can remove the male flag.
for df in [income_train, income_test]:
    df.drop(columns = 'male',inplace=True)

income_train.shape
```

Out[84]: (9557, 130)

In [85]:
```
#lets check area1 and area2 also
# area1, =1 zona urbana
# area2, =2 zona rural
#area2 redundant because we have a column indicating if the house is in a urban zone

for df in [income_train, income_test]:
    df.drop(columns = 'area2',inplace=True)

income_train.shape
```

Out[85]: (9557, 129)

In [86]:
```
#Finally lets delete 'Id', 'idhogar'
cols=['Id','idhogar']
for df in [income_train, income_test]:
    df.drop(columns = cols,inplace=True)

income_train.shape
```

Out[86]: (9557, 127)

Predict the accuracy using random forest classifier

In [87]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classification_rep
```

In [88]:
```
x_features=income_train.iloc[:,0:-1]
y_target=income_train.iloc[:,-1]
print(x_features.shape)
print(y_target.shape)
```

```
(9557, 126)
(9557,)
```

In [90]:
```
x_train,x_test,y_train,y_test=train_test_split(x_features,y_target,test_size=0.2,random
rmclassifier = RandomForestClassifier()
rmclassifier.fit(x_train,y_train)
```

Out[90]: RandomForestClassifier()

In [91]:
```
rmclassifier.fit(x_train,y_train)
```

Out[91]: RandomForestClassifier()

In [93]:
```
y_predict = rmclassifier.predict(x_test)
```

In [95]:
```
print(accuracy_score(y_test,y_predict))
```

```
0.9518828451882845
```

In [96]:
```
print(confusion_matrix(y_test,y_predict))
```

```
[[ 136    0    0   21]
```

```
[   2  288    0   27]
[   0    3  194   36]
[   0    2    1 1202]]
```

In [97]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           1       0.99      0.87      0.92       157
           2       0.98      0.91      0.94       317
           3       0.99      0.83      0.91       233
           4       0.93      1.00      0.97      1205

    accuracy                           0.95      1912
   macro avg       0.97      0.90      0.93      1912
weighted avg       0.95      0.95      0.95      1912
```

In [99]:
```
y_predict_testdata = rmclassifier.predict(income_test)
```

In [100…
```
y_predict_testdata
```

Out[100…
```
array([4, 4, 4, ..., 2, 2, 2], dtype=int64)
```

Accuracy using random forest with cross validation.

In [101…
```
from sklearn.model_selection import KFold,cross_val_score
```

In [103…
```
#score for default 10 trees
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_features,y_target,cv=kfold,scoring='accuracy'))
results=cross_val_score(rmclassifier,x_features,y_target,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.94246862 0.94979079 0.94557823 0.94243851 0.94976452]
94.60081361157272
```

In [104…
```
#score for 100 trees
num_trees= 100
rmclassifier=RandomForestClassifier(n_estimators=100, random_state=10,n_jobs = -1)
print(cross_val_score(rmclassifier,x_features,y_target,cv=kfold,scoring='accuracy'))
results=cross_val_score(rmclassifier,x_features,y_target,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.94246862 0.94979079 0.94557823 0.94243851 0.94976452]
94.60081361157272
```

RandomForestClassifier with cross validation has the highest accuracy score of 94.60%

In [106…
```
rmclassifier.fit(x_features,y_target)
```

Out[106…
```
RandomForestClassifier(n_jobs=-1, random_state=10)
```

In [108…
```
y_predict_testdata = rmclassifier.predict(income_test)
y_predict_testdata
```

Out[108…
```
array([4, 4, 4, ..., 4, 4, 4], dtype=int64)
```

visualize how our model uses the different features and which features have greater effect.

In [114…
```
labels = list(x_features)
```

```
feature_importances = pd.DataFrame({'feature': labels, 'importance': rmclassifier.featu
feature_importances=feature_importances[feature_importances.importance>0.015]
feature_importances.head()
```

Out[114…

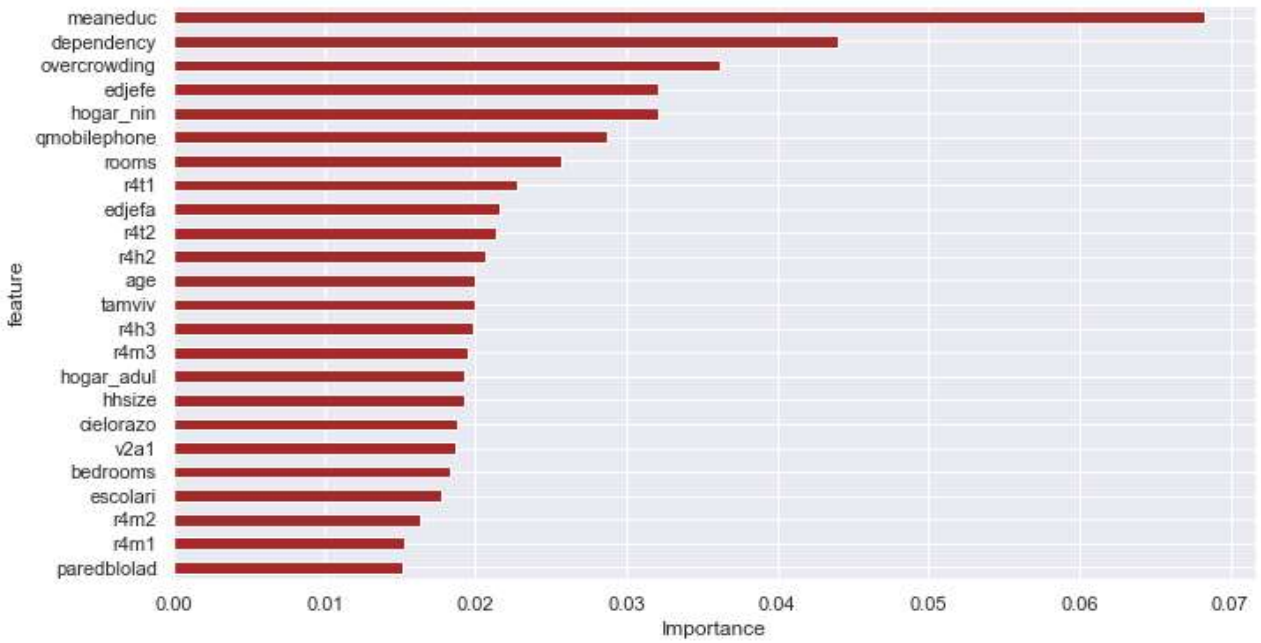
	feature	importance
0	v2a1	0.018653
2	rooms	0.025719
9	r4h2	0.020706
10	r4h3	0.019808
11	r4m1	0.015271

In [115…

```
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_import
plt.xlabel('Importance')
```

Out[115… `Text(0.5, 0, 'Importance')`



meaneduc has the highest influence on the model followed by dependency, overcrowding, edjefe, hogar_nin

In [ ]: