Project - Income Qualification

Description

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:

1. Identify the output variable.
2. Understand the type of data.
3. Check if there are any biases in your dataset.
4. Check whether all members of the house have the same poverty level.
5. Check if there is a house without a family head.
6. Set poverty level of the members and the head of the house within a family.
7. Count how many null values are existing in columns.

8. Remove null value rows of the target variable.
9. Predict the accuracy using random forest classifier.
10. Check the accuracy using random forest with cross validation.

Understand the Data

In [100…

```
# Import necessary Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [101…

```
# Load Data

train_df=pd.read_csv('train.csv') # Loading Training csv file
test_df=pd.read_csv('test.csv') # Loading Test csv file
```

In [102…

```
# Explore Train dataset

train_df.shape # printing No of rows and columns
```

Out[102… (9557, 143)

In [103…

```
train_df.head() # Check top 5 record
```

Out[103…

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SC
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	4489	1	144	

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SC
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	289	16	121	
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	1369	16	121	

5 rows × 143 columns

In [104...
```
train_df.info() # Check 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 [105...
```
# Explore Test dataset

test_df.shape # printing No of rows and columns
```

Out[105... (23856, 142)

In [106...
```
test_df.head()  # Check top 5 record
```

Out[106...

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQBescolari	SQBage	SQBhogar_total	SQBedjef
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	...	4	0	16	9	
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	...	41	256	1681	9	
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	...	41	289	1681	9	
3	ID_a8db26a79	NaN	0	14	0	1	1	1	1.0	0	...	59	256	3481	1	25
4	ID_a62966799	175000.0	0	4	0	1	1	1	1.0	0	...	18	121	324	1	

5 rows × 142 columns

In [107…

```
test_df.info() # Check 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 [108…

```
# info summary:
# Train dataset has Rows: 9557 entries, Columns: 143 entries, Id to Target, dtypes: float64(8), int64(130), object(5)
# Test dataset has Rows: 23856 entries, Columns: 142 entries, Id to agesq, dtypes: float64(8), int64(129), object(5)

# Note: We don't have 'Target' feature in Test Dataset.
# There are 5 object type, 130(Train set)/ 129 (test set) integer type and 8 float type features.
```

Understand the type of data.

In [109…

```
# List the column for different datatypes:

print("Integer Type: ")
print(train_df.select_dtypes(np.int64).columns)
print("\n")

print("Float Type: ")
print(train_df.select_dtypes(np.float64).columns)
print("\n")

print("Object Type: ")
print(train_df.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 [110…
```
train_df.select_dtypes('int64').head()
```

Out[110…

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	...	area1	area2	age	SQBescolari	SQBage	SQBhogar_total	SQBed
0	0	3	0	1	1	0	0	1	1	0	...	1	0	43	100	1849	1	
1	0	4	0	1	1	1	0	1	1	0	...	1	0	67	144	4489	1	
2	0	8	0	1	1	0	0	0	0	0	...	1	0	92	121	8464	1	
3	0	5	0	1	1	1	0	2	2	1	...	1	0	17	81	289	16	
4	0	5	0	1	1	1	0	2	2	1	...	1	0	37	121	1369	16	

5 rows × 130 columns

Count how many null values are existing in columns.

In [112…
```
null_count=train_df.select_dtypes('int64').isnull().sum()
null_count[null_count>0]
```

Out[112…
```
Series([], dtype: int64)
```

In [113…
```
train_df.select_dtypes('float64').head()
```

Out[113…

	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 [114…
```
null_count=train_df.select_dtypes('float64').isnull().sum()
null_count[null_count>0]
```

Out[114…
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

In [115…
```
train_df.select_dtypes('object').head()
```

Out[115…

	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 [116…
```
null_count=train_df.select_dtypes('object').isnull().sum()
null_count[null_count>0]
```

Out[116…
```
Series([], dtype: int64)
```

In [117…
```
# 1. No null values for Integer type features.
# 2. No null values for Object type features.
# 3. For float type: v2a1: 6860, v18q1:7342, rez_esc:7928, meaneduc:5, SQBmeaned:5
# 4. We also noticed that object type features dependency, edjefe, edjefa have mixed values.
# 5.Lets fix the data for features with null values and features with mixed values
```

In [188…
```
## Lets fix the column with mixed values.
```

In [119…
```
# According to the documentation for these columns:
```

```
# dependency, Dependency rate, calculated = (number of members of the household younger than 19 or older than 64)/(number
# edjefe, years of education of male head of household, based on the interaction of escolari (years of education), head o
# edjefa, years of education of female head of household, based on the interaction of escolari (years of education), heaa
```

In [120…

```
# For these three variables, it seems “yes” = 1 and “no” = 0. We can correct the variables using a mapping and convert to
```

In [121…

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

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

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

Out[121…

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

In [122…

```
# 2.2 Lets fix the column with null values
```

In [123…

```
# Null values are in these variable:
# v2a1: 6860, v18q1:7342, rez_esc:7928, meaneduc:5, SQBmeaned:5

# According to the documentation for these columns:
# 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
```

In [124…

```
# 1. Lets look at v2a1 (total nulls: 6860) : Monthly rent payment
# why the null values, Lets look at few rows with nulls in v2a1
# Columns related to  Monthly rent payment
# tipovivi1, =1 own and fully paid house
# tipovivi3, =1 rented
# tipovivi4, =1 precarious
# tipovivi5, "=1 other(assigned,  borrowed)"
```

In [125…

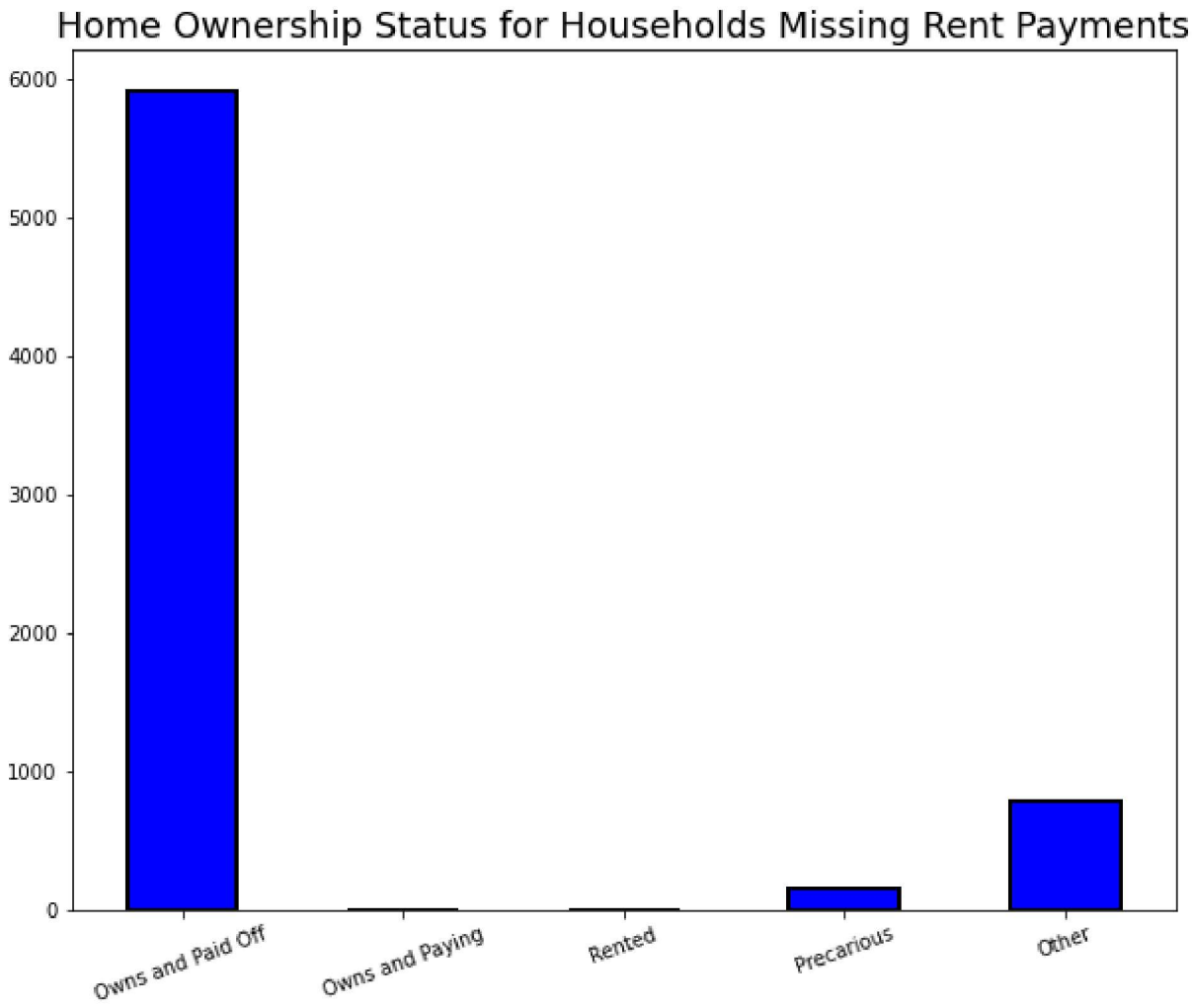
```
data = train_df[train_df['v2a1'].isnull()].head()
columns=['tipovivi1','tipovivi2','tipovivi3','tipovivi4','tipovivi5']
data[columns]
```

Out[125…

	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 [126…

```
# Variables indicating home ownership
own_variables = [x for x in train_df if x.startswith('tipo')]

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



In [127…

```
#Looking at the above data it makes sense that when the house is fully paid, there will be no monthly rent payment.
#Lets add 0 for all the null values.

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

In [128…

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

Out[128…

```
v2a1    0
dtype: int64
```

In [129…

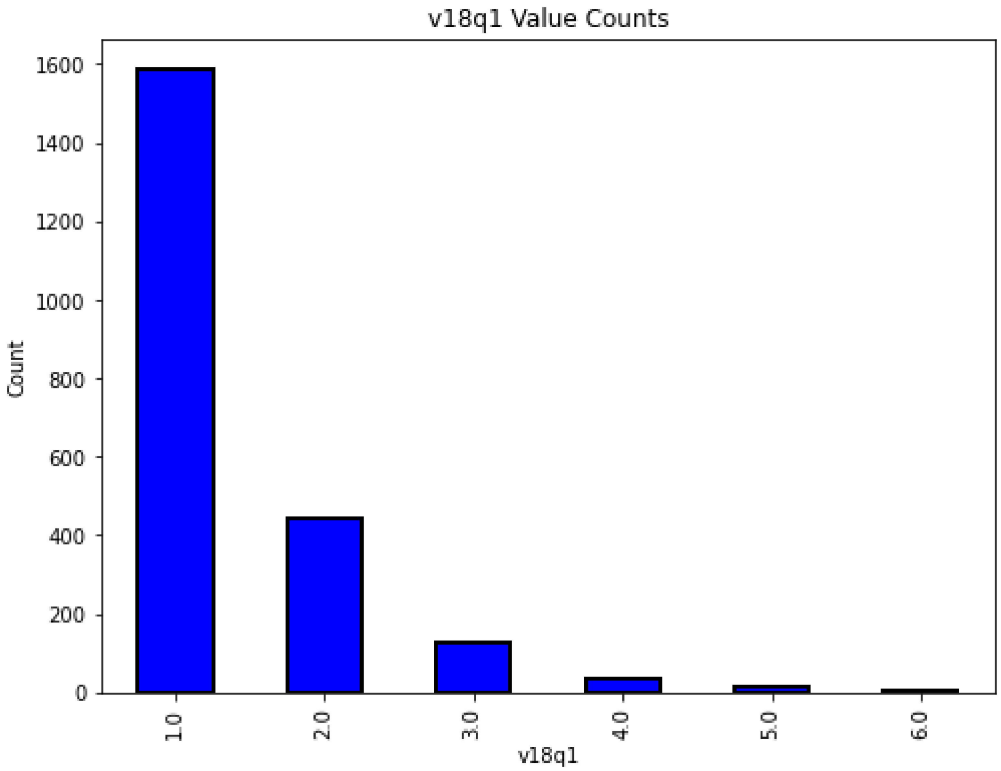
```
# 2. Lets look at v18q1 (total nulls: 7342) : number of tablets household owns
# why the null values, Lets look at few rows with nulls in v18q1
# Columns related to  number of tablets household owns
# v18q, owns a tablet

# Since this is a household variable, it only makes sense to look at it on a household level,
# so we'll only select the rows for the head of household.

# Heads of household

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

Out[129…

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

In [130…

```
plt.figure(figsize = (8, 6))
col='v18q1'
train_df[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();
```



In [131…

```
#Looking at the above data it makes sense that when owns a tablet column is 0, there will be no number of tablets household
#Lets add 0 for all the null values.

for df in [train_df,test_df]:
    df['v18q1'].fillna(value=0,inplace=True)

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

Out[131…

```
v18q1    0
dtype: int64
```

In [132…

```
# 3. Lets look at rez_esc (total nulls: 7928) : Years behind in school
# why the null values, Lets look at few rows with nulls in rez_esc
# Columns related to Years behind in school
# Age in years

# Lets look at the data with not null values first.

train_df[train_df['rez_esc'].notnull()]['age'].describe()
```

Out[132...
```
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
```

In [133...
```
#From the above , we see that when min age is 7 and max age is 17 for Years, then the 'behind in school' column has a val
#Lets confirm

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

Out[133...
```
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 [134...
```
train_df.loc[(train_df['rez_esc'].isnull() & ((train_df['age'] > 7) & (train_df['age'] < 17)))]['age'].describe()
#There is one value that has Null for the 'behind in school' column with age between 7 and 17
```

Out[134...
```
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 [135...
```
train_df[(train_df['age'] ==10) & train_df['rez_esc'].isnull()].head()
train_df[(train_df['Id'] =='ID_f012e4242')].head()
```

Out[135...

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe
2514	ID_f012e4242	160000.0	0	6	0	1	1	1	1.0	0	...	0	100	9	121

1 rows × 143 columns

In [136...

```
#from above we see that  the 'behind in school' column has null values
# Lets use the above to fix the data

for df in [train_df, test_df]:
    df['rez_esc'].fillna(value=0, inplace=True)
train_df[['rez_esc']].isnull().sum()
```

Out[136...

```
rez_esc    0
dtype: int64
```

In [137...

```
# Lets look at meaneduc (total nulls: 5) : average years of education for adults (18+)
# why the null values, Lets look at few rows with nulls in meaneduc
# Columns related to average years of education for adults (18+)
# 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
# instlevel1, =1 no level of education
# instlevel2, =1 incomplete primary
```

In [138...

```
data = train_df[train_df['meaneduc'].isnull()].head()

columns=['edjefe','edjefa','instlevel1','instlevel2']
data[columns][data[columns]['instlevel1']>0].describe()
```

Out[138...

	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

	edjefe	edjefa	instlevel1	instlevel2
25%	NaN	NaN	NaN	NaN
50%	NaN	NaN	NaN	NaN
75%	NaN	NaN	NaN	NaN
max	NaN	NaN	NaN	NaN

In [139…

```
#from the above, we find that meaneduc is null when no level of education is 0
#Lets fix the data

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

Out[139…

```
meaneduc    0
dtype: int64
```

In [140…

```
# Lets look at SQBmeaned  (total nulls: 5) : square of the mean years of education of adults (>=18) in the household 142
# why the null values, Lets look at few rows with nulls in SQBmeaned
# Columns related to average years of education for adults (18+)
# 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
# instlevel1, =1 no level of education
# instlevel2, =1 incomplete primary
```

In [141…

```
data = train_df[train_df['SQBmeaned'].isnull()].head()

columns=['edjefe','edjefa','instlevel1','instlevel2']
data[columns][data[columns]['instlevel1']>0].describe()
```

Out[141…

	edjefe	edjefa	instlevel1	instlevel2
count	0.0	0.0	0.0	0.0
mean	NaN	NaN	NaN	NaN
std	NaN	NaN	NaN	NaN

	edjefe	edjefa	instlevel1	instlevel2
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 [142…

```
#from the above, we find that SQBmeaned is null when no level of education is 0
#Lets fix the data
for df in [train_df, test_df]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
train_df[['SQBmeaned']].isnull().sum()
```

Out[142…

```
SQBmeaned    0
dtype: int64
```

In [143…

```
#Lets look at the overall data

null_counts = train_df.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

Out[143…

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

Identify the output variable.

In [144…

```
# Groupby the household and figure out the number of unique values

all_equal = train_df.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 the family members do not all have the same target.'.format(len(not_equal)))
```

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

In [145…

```
#Lets check one household
```

```
train_df[train_df['idhogar'] == not_equal.index[0]][['idhogar', 'parentesco1', 'Target']]
```

Out[145...

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

Check if there is a house without a family head.

In [146...

```
#Lets use Target value of the parent record (head of the household) and update rest. But before that lets check
# if all families has a head.

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

# Find households without a head
households_no_head = train_df.loc[train_df['idhogar'].isin(households_head[households_head == 0].index), :]

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

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

In [147...

```
# Find households without a head and where Target value are different

households_no_head_equal = households_no_head.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)
print('{} Households with no head have different Target value.'.format(sum(households_no_head_equal == False)))
```

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

Set poverty level of the members and the head of the house within a family.

In [148...

```
for household in not_equal.index:
    # Find the correct label (for the head of household)
    true_target = int(train_df[(train_df['idhogar'] == household) & (train_df['parentesco1'] == 1.0)]['Target'])
```

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


# Groupby the household and figure out the number of unique values
all_equal = train_df.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 the family members do not all have the same target.'.format(len(not_equal)))
```

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

Lets check for any bias in the dataset

In [149…

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

Out[149…

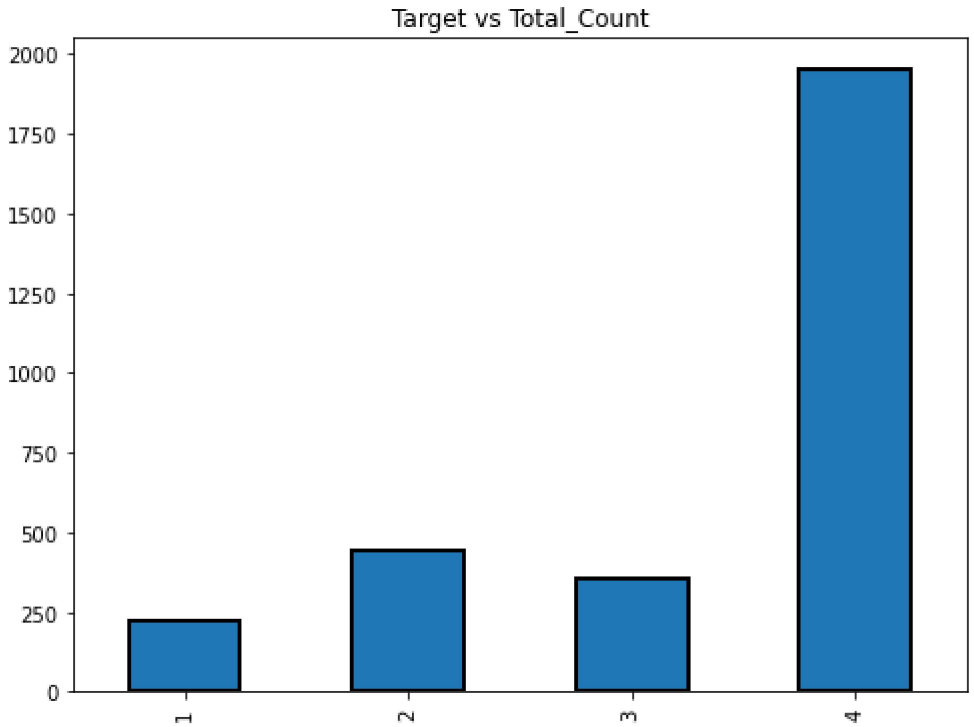
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

In [150…

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

Out[150…

```
<AxesSubplot:title={'center':'Target vs Total_Count'}>
```



In [151…

```
# extreme poverty is the smallest count in the train dataset. The dataset is biased.
```

Lets look at the Squared Variables

In [152…

```
#‘SQBescolari’
#‘SQBage’
#‘SQBhogar_total’
#‘SQBedjefe’
#‘SQBhogar_nin’
#‘SQBovercrowding’
#‘SQBdependency’
#‘SQBmeaned’
#‘agesq’
```

In [153…

```
# Lets remove them

print(train_df.shape)
```

```
cols=['SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
        'SQBhogar_nin', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq']

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

print(train_df.shape)
```

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

In [154…

```
id_ = ['Id', 'idhogar', 'Target']
```

In [155…

```
    print(train_df.select_dtypes(np.int64).columns)
```

```
Index(['hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q', 'r4h1', 'r4h2',
       'r4h3', 'r4m1',
       ...
       'lugar1', 'lugar2', 'lugar3', 'lugar4', 'lugar5', 'lugar6', 'area1',
       'area2', 'age', 'Target'],
      dtype='object', length=124)
```

In [156…

```
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 [157…

```
# Check for redundant household variables

heads = train_df.loc[train_df['parentesco1'] == 1, :]
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

Out[157… (2973, 98)

In [158…

```
# 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[158… ['coopele', 'area2', 'edjefa', 'tamhog', 'hhsize', 'hogar_total']

In [159…

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

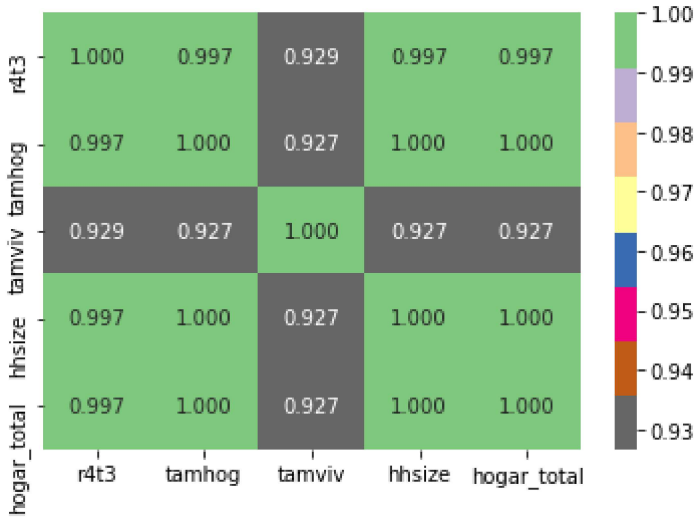
Out[159…

	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

	r4t3	tamhog	tamviv	hhsize	hogar_total
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 [160…

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



In [161…

```
# 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 [162…

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

train_df.shape
```

Out[162… (9557, 131)

In [163…
```
# Check for redundant Individual variables

ind = train_df[id_ + ind_bool + ind_ordered]
ind.shape
```

Out[163… (9557, 39)

In [164…
```
# 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[164… ['female']

In [165…
```
# This is simply the opposite of male! We can remove the male flag.
for df in [train_df, test_df]:
    df.drop(columns = 'male',inplace=True)

train_df.shape
```

Out[165… (9557, 130)

In [166…
```
#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 [train_df, test_df]:
    df.drop(columns = 'area2',inplace=True)

train_df.shape
```

Out[166… (9557, 129)

In [167…
```
#Finally lets delete 'Id', 'idhogar'
cols=['Id','idhogar']
for df in [train_df, test_df]:
    df.drop(columns = cols,inplace=True)

train_df.shape
```

Out[167… (9557, 127)

Predict the accuracy using random forest classifier.

In [168…
```
x_features=train_df.iloc[:,0:-1]
y_features=train_df.iloc[:,-1]
print(x_features.shape)
print(y_features.shape)
```

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

In [169…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classification_report

x_train,x_test,y_train,y_test=train_test_split(x_features,y_features,test_size=0.2,random_state=1)
rmclassifier = RandomForestClassifier()
```

In [170…
```
rmclassifier.fit(x_train,y_train)
```

Out[170… RandomForestClassifier()

In [171…
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=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=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Out[171… RandomForestClassifier(n_estimators=10)

In [172…
```
y_predict = rmclassifier.predict(x_test)
```

In [173…
```
print(accuracy_score(y_test,y_predict))
print(confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
```

```
0.9518828451882845
[[ 133    1    0   23]
 [   1  287    1   28]
 [   0    1  197   35]
 [   0    1    1 1203]]
              precision    recall  f1-score   support

           1       0.99      0.85      0.91       157
           2       0.99      0.91      0.95       317
           3       0.99      0.85      0.91       233
           4       0.93      1.00      0.96      1205

    accuracy                           0.95      1912
   macro avg       0.98      0.90      0.93      1912
weighted avg       0.95      0.95      0.95      1912
```

In [175…
```
y_predict_testdata = rmclassifier.predict(test_df)
```

In [176…
```
y_predict_testdata
```

Out[176… array([4, 4, 4, ..., 4, 4, 4], dtype=int64)

Check the accuracy using random forest with cross validation.

In [178…
```
from sklearn.model_selection import KFold,cross_val_score
```

Checking the score using default 10 trees

In [179…
```
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_features,cv=kfold,scoring='accuracy'))
results=cross_val_score(rmclassifier,x_features,y_features,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.94508368 0.94874477 0.94243851 0.9429618  0.95133438]
94.61126291520792
```

Checking the score using 100 trees

In [180…
```
num_trees= 100

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

```
[0.94508368 0.94874477 0.94243851 0.9429618  0.95133438]
94.61126291520792
```

In [183…
```
#y_predict_testdata = rmclassifier.predict(test_df)
#y_predict_testdata
```

In [184…
```
# Looking at the accuracy score, RandomForestClassifier with cross validation has the highest accuracy score of 94.60%
```

In [185…
```
rmclassifier.fit(x_features,y_features)
labels = list(x_features)
feature_importances = pd.DataFrame({'feature': labels, 'importance': rmclassifier.feature_importances_})
feature_importances=feature_importances[feature_importances.importance>0.015]
feature_importances.head()
```

Out[185…

	feature	importance
0	v2a1	0.018506
2	rooms	0.024329
9	r4h2	0.020550

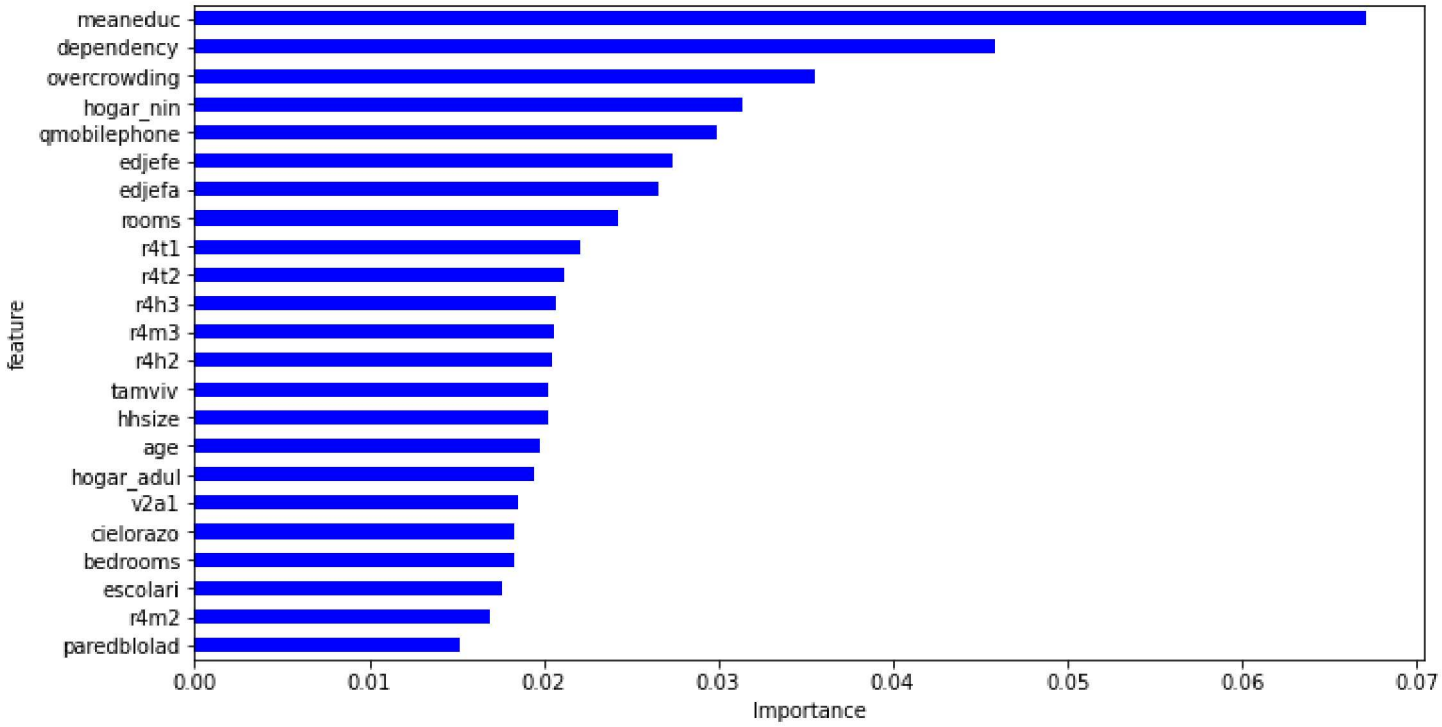
	feature	importance
10	r4h3	0.020674
12	r4m2	0.016913

In [186…

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

Out[186…

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



In [187…

```
# From the above figure, meaneduc,dependency,overcrowding has significant influence on the model.
```

In [ ]: