In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import collections
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
from sklearn.model_selection import cross_val_score


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

In [5]:
```
# Loading the dataset
df_income_train = pd.read_csv(r"C:\Users\prate\Desktop\train.csv")
df_income_test =  pd.read_csv(r"C:\Users\prate\Desktop\test.csv")
```

In [6]:
```
df_income_train.head()
```

Out[6]:

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

5 rows × 143 columns

In [7]:
```
df_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 [8]:
```
df_income_test.head()
```

Out[8]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQI
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 [9]:
```
df_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 [10]:
```
#List the columns for different datatypes:
print('Integer Type: ')
print(df_income_train.select_dtypes(np.int64).columns)
print('\n')
print('Float Type: ')
print(df_income_train.select_dtypes(np.float64).columns)
print('\n')
print('Object Type: ')
print(df_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 [11]:
```
df_income_train.select_dtypes('int64').head()
```

Out[11]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	...	area1	area2	age	SQ
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 [12]:
```
for column in df_income_train:
    if column not in df_income_test:
        print('The output variable is', column)
```

```
The output variable is Target
```

In [13]:
```
print(df_income_train['Target'].value_counts())
```

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

In [14]:
```
# Finding columns with null values
null_counts=df_income_train.select_dtypes('int64').isnull().sum()
null_counts[null_counts > 0]
```

Out[14]: `Series([], dtype: int64)`

In [15]:
```
df_income_train.select_dtypes('float64').head()
```

Out[15]:

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

In [16]:
```
# Finding columns with null values
null_counts=df_income_train.select_dtypes('float64').isnull().sum()
null_counts[null_counts > 0]
```

Out[16]:
```
v2a1        6860
v18q1       7342
rez_esc     7928
meaneduc       5
SQBmeaned      5
dtype: int64
```

In [17]:
```
df_income_train.select_dtypes('object').head()
```

Out[17]:

	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 [18]:
```
#Find columns with null values
null_counts=df_income_train.select_dtypes('object').isnull().sum()
null_counts[null_counts > 0]
```

Out[18]:
```
Series([], dtype: int64)
```

In [19]:
```
# With out Null values treatment we cannot get the correct answers
```

Define Variable Categories There are several different categories of variables:

Squared Variables: derived from squaring variables in the data

Id variables: identifies the data and should not be used as features

Household variables

Boolean: Yes or No Ordered Discrete: Integers with an ordering Continuous numeric Individual Variables: these are characteristics of each individual rather than the household

Boolean: Yes or No (0 or 1) Ordered Discrete: Integers with an ordering

In [20]:
```
# dependency column
```

In [21]:
```
mapping={'yes':1,'no':0}

for df in [df_income_train, df_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)
```

In [22]:
```
df_income_train[['dependency','edjefe','edjefa']]
```

Out[22]:

	dependency	edjefe	edjefa
0	0.00	10.0	0.0
1	8.00	12.0	0.0
2	8.00	0.0	11.0
3	1.00	11.0	0.0
4	1.00	11.0	0.0
...	...	...	...
9552	0.25	9.0	0.0
9553	0.25	9.0	0.0
9554	0.25	9.0	0.0
9555	0.25	9.0	0.0
9556	0.25	9.0	0.0

9557 rows × 3 columns

In [23]:
```
df_income_train[['dependency','edjefe','edjefa']].describe()
```

Out[23]:

	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

In [24]:
```
df_income_test[['dependency','edjefe','edjefa']].describe()
```

Out[24]:

	dependency	edjefe	edjefa
count	23856.000000	23856.000000	23856.000000
mean	1.181327	5.199824	2.800176
std	1.666209	5.200980	4.603592
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

In [25]:
```

# v2a1 column
```

In [26]:
```
data = df_income_train[df_income_train['v2a1'].isnull()].head()

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

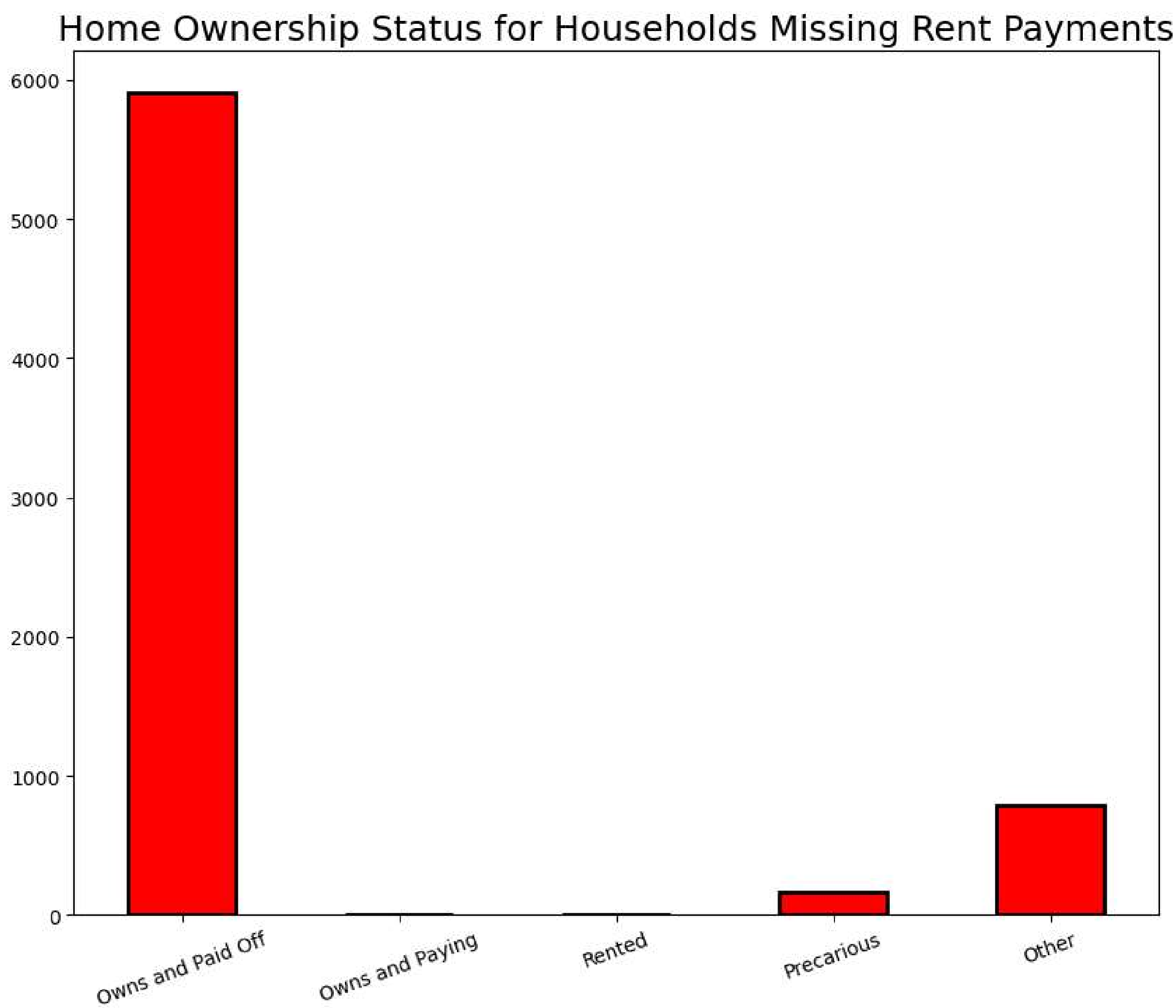
Out[26]:

	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 [27]:

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

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



In [28]:
```
for df in [df_income_train, df_income_test]:
    df['v2a1'].fillna(value=0, inplace=True)

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

Out[28]:
```
v2a1    0
dtype: int64
```

In [29]:
```
df_income_test[['v2a1']].isnull().sum()
```

Out[29]:
```
v2a1    0
dtype: int64
```

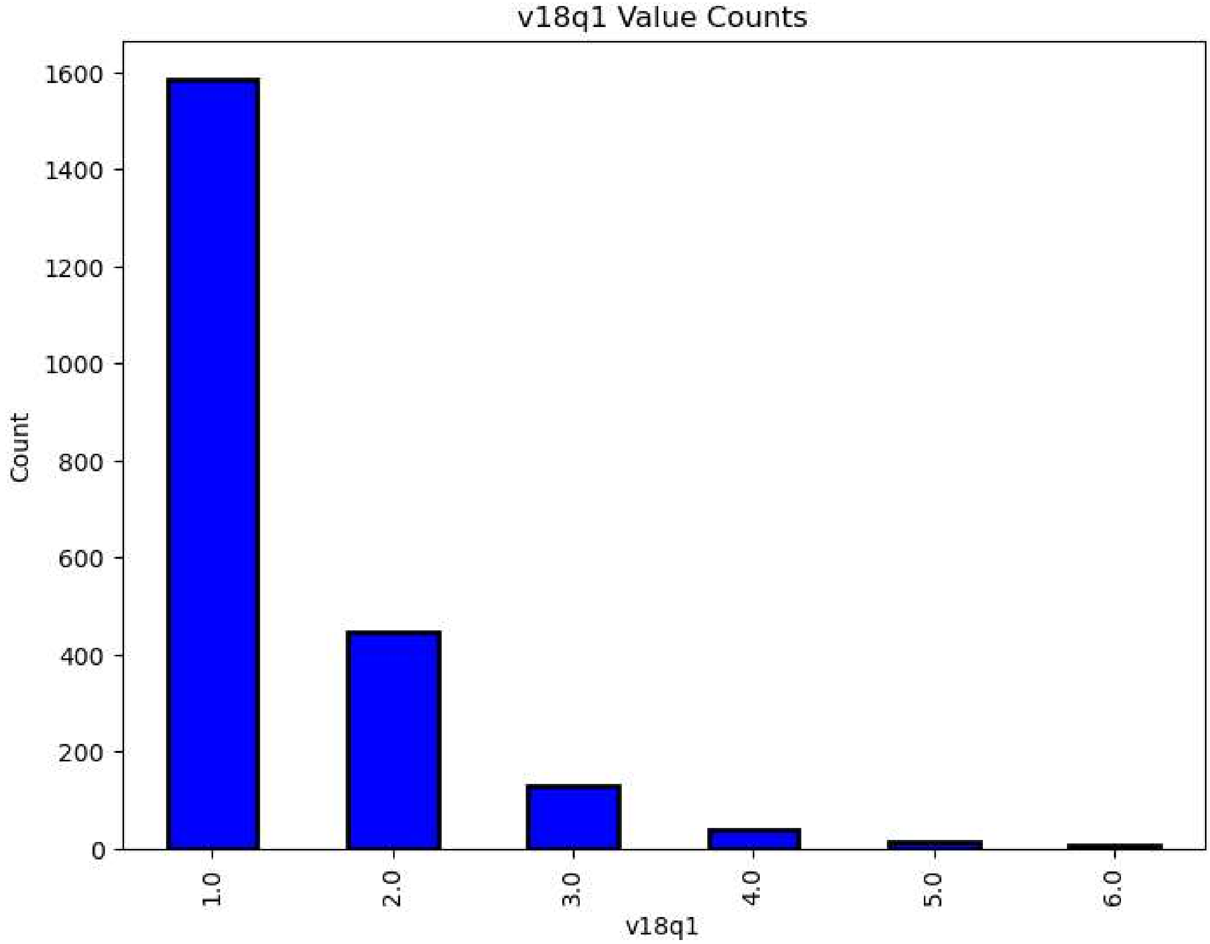
In [30]:
```

# v18q1 Column
```

In [31]:
```
heads = df_income_train.loc[df_income_train['parentesco1'] == 1].copy()
heads.groupby('v18q')['v18q1'].apply(lambda x: x.isnull().sum())
```

Out[31]:
```
v18q
0    2318
1       0
Name: v18q1, dtype: int64
```

In [32]:
```
plt.figure(figsize = (8, 6))
col='v18q1'
df_income_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();
```



In [33]:
```
for df in [df_income_train, df_income_test]:
    df['v18q1'].fillna(value=0, inplace=True)

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

Out[33]:
```
v18q1    0
dtype: int64
```

In [34]:
```
df_income_test[['v18q1']].isnull().sum()
```

Out[34]:
```
v18q1    0
dtype: int64
```

In [35]:
```
# rez_esc column
```

In [36]:
```
# Checking for no null values
df_income_train[df_income_train['rez_esc'].notnull()]['age'].describe()
```

Out[36]:
```
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 [37]:
```
df_income_train.loc[df_income_train['rez_esc'].isnull()]['age'].describe()
```

Out[37]:
```
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 [38]:
```
.loc[(df_income_train['rez_esc'].isnull() & ((df_income_train['age'] > 7) & (df_in
```

Out[38]:
```
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 [39]:
```
df_income_train[(df_income_train['age'] ==10) & df_income_train['rez_esc'].isnull
df_income_train[(df_income_train['Id'] =='ID_f012e4242')].head()
```

Out[39]:

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

1 rows × 143 columns

In [40]:
```
for df in [df_income_train, df_income_test]:
    df['rez_esc'].fillna(value=0, inplace=True)
df_income_train[['rez_esc']].isnull().sum()
```

Out[40]:
```
rez_esc    0
dtype: int64
```

In [41]:
```
# meaneduc column
```

In [42]:
```
data = df_income_train[df_income_train['meaneduc'].isnull()].head()

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

Out[42]:

	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 [43]:
```
for df in [df_income_train, df_income_test]:
    df['meaneduc'].fillna(value=0, inplace=True)
df_income_train[['meaneduc']].isnull().sum()
```

Out[43]:
```
meaneduc    0
dtype: int64
```

In [44]:
```
df_income_test[['meaneduc']].isnull().sum()
```

Out[44]:
```
meaneduc    0
dtype: int64
```

In [45]:
```
# SQBmeaned Column
```

In [46]:
```
data = df_income_train[df_income_train['SQBmeaned'].isnull()].head()

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

Out[46]:

	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 [47]:
```
for df in [df_income_train, df_income_test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
df_income_train[['SQBmeaned']].isnull().sum()
```

Out[47]:
```
SQBmeaned    0
dtype: int64
```

In [48]:
```
df_income_test[['SQBmeaned']].isnull().sum()
```

Out[48]:
```
SQBmeaned    0
dtype: int64
```

In [49]:
```
null_counts = df_income_train.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

Out[49]:
```
Series([], dtype: int64)
```

In [50]:
```
null_counts = df_income_test.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

Out[50]:
```
Series([], dtype: int64)
```

In [51]:
```
# Groupby the household and figure out the number of unique values
all_equal = df_income_train.groupby('idhogar')['Target'].apply(lambda x: x.nuniqu

# 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
```

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

In [52]:
```
df_income_train[df_income_train['idhogar'] == not_equal.index[0]][['idhogar', 'pa
```

Out[52]:

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

In [53]:
```
households_head = df_income_train.groupby('idhogar')['parentesco1'].sum()

# Find households without a head
households_no_head = df_income_train.loc[df_income_train['idhogar'].isin(househol

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

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

In [54]:
```
# Find households without a head and where Target value are different
households_no_head_equal = households_no_head.groupby('idhogar')['Target'].apply(
print('{} Households with no head have different Target value.'.format(sum(househ
```

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

In [55]:
```
# Iterating through each household
for household in not_equal.index:
    # Find the correct label (for the head of household)
    true_target = int(df_income_train[(df_income_train['idhogar'] == household) &

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


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

# 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
```

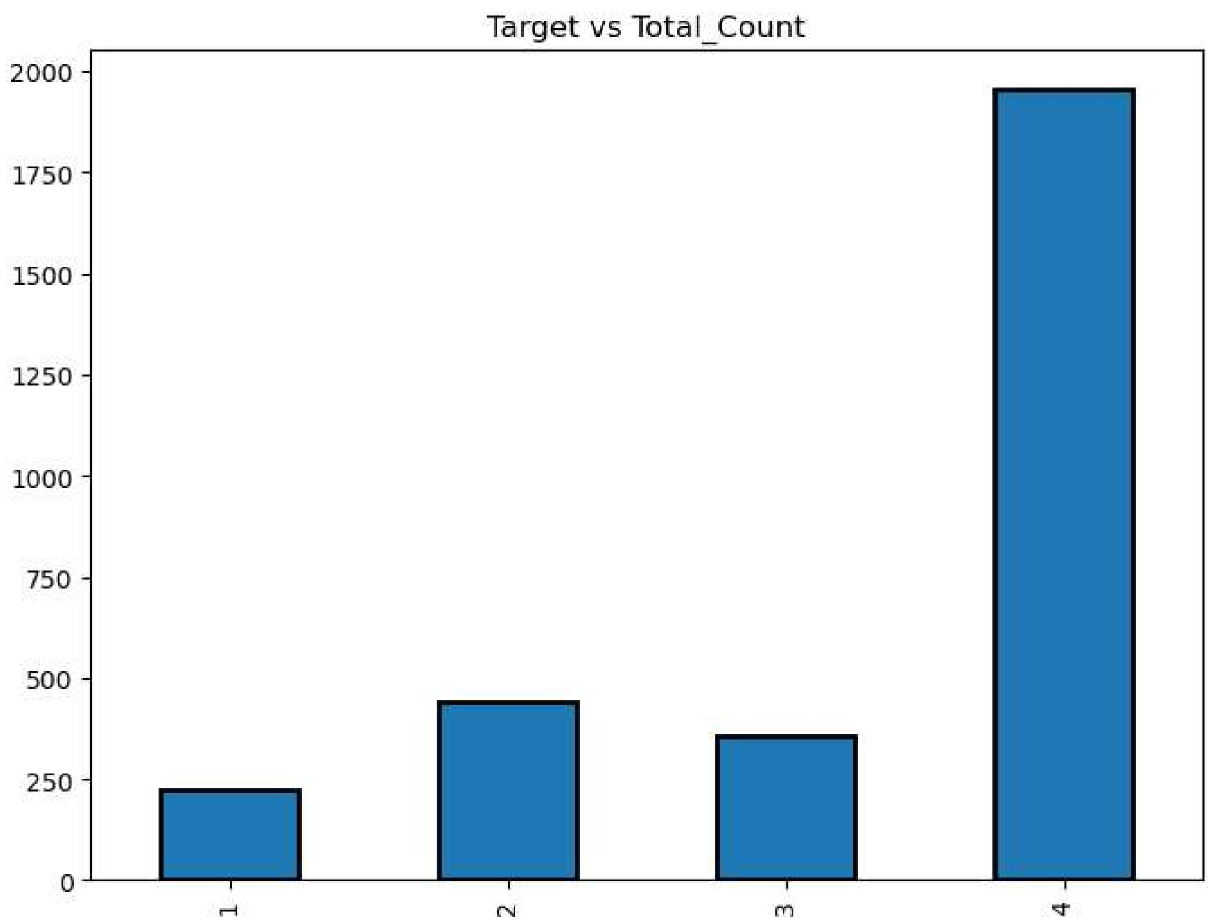
```
There are 0 households where the family members do not all have the same targe
t.
```

In [56]:
```
# 1 = extreme poverty 2 = moderate poverty 3 = vulnerable households 4 = non vulr
target_counts = heads['Target'].value_counts().sort_index()
target_counts
```

Out[56]:
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

In [57]:
```
target_counts.plot.bar(figsize = (8, 6),linewidth = 2,edgecolor = 'k',title="Targ
```

Out[57]:
```
<AxesSubplot:title={'center':'Target vs Total_Count'}>
```



In [58]:
```
# extreme poverty is the smallest count in the train dataset. The dataset is bias
print(df_income_train.shape)
cols=['SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
        'SQBhogar_nin', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq']


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

print(df_income_train.shape)
```

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

In [59]:
```
id_ = ['Id', 'idhogar', 'Target']

ind_bool = ['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil2', 'est
            'estadocivil4', 'estadocivil5', 'estadocivil6', 'estadocivil7',
            'parentesco1', 'parentesco2',  'parentesco3', 'parentesco4', 'parente
            'parentesco6', 'parentesco7', 'parentesco8',  'parentesco9', 'parente
            'parentesco11', 'parentesco12', 'instlevel1', 'instlevel2', 'instleve
            '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',  'r
              'r4t3', 'v18q1', 'tamhog','tamviv','hhsize','hogar_nin',
              'hogar_adul','hogar_mayor','hogar_total',  'bedrooms', 'qmobilephon

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

In [60]:
```
#Check for redundant household variables
heads = df_income_train.loc[df_income_train['parentesco1'] == 1, :]
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

Out[60]: (2973, 98)

In [61]:
```
# 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
```

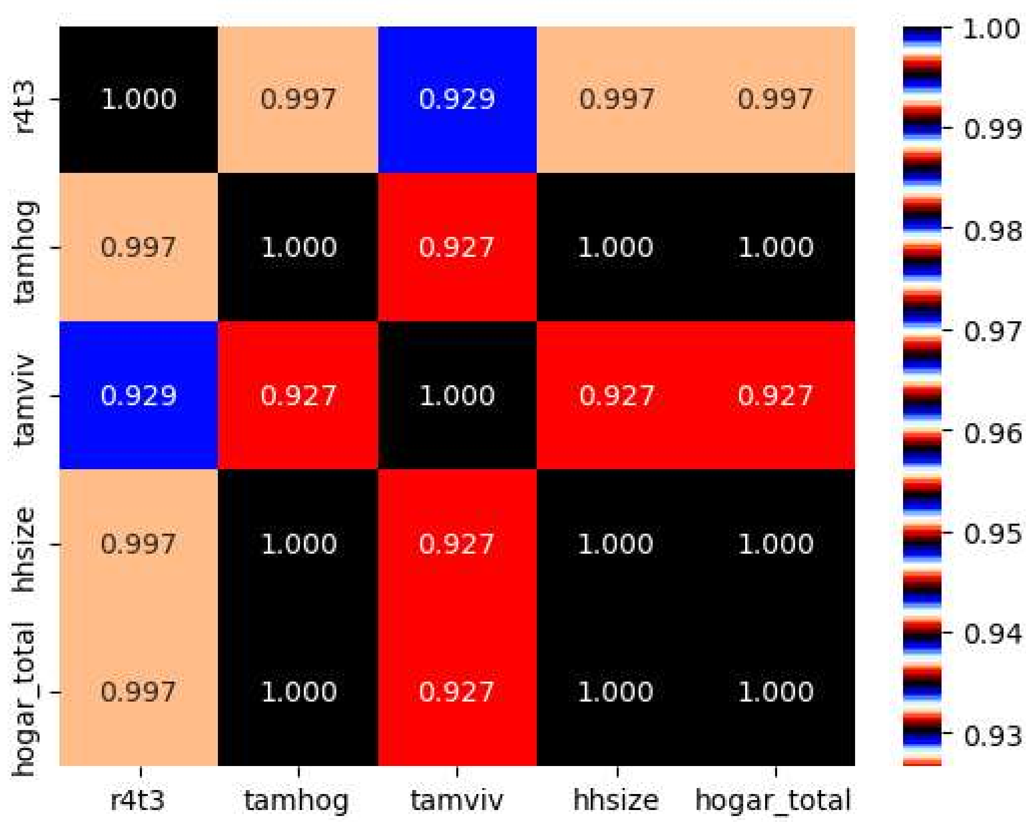
Out[61]:
```
['coopele', 'area2', 'tamhog', 'hhsize', 'hogar_total']
```

In [62]:
```
corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].abs() >
```

Out[62]:

	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 [63]:
```
sns.heatmap(corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamho
            annot=True, cmap = plt.cm.flag, fmt='.3f');
```



In [64]:
```
cols=['tamhog', 'hogar_total', 'r4t3']
for df in [df_income_train, df_income_test]:
    df.drop(columns = cols,inplace=True)

df_income_train.shape
```

Out[64]: (9557, 131)

In [65]:
```
#Check for redundant Individual variables
ind = df_income_train[id_ + ind_bool + ind_ordered]
ind.shape
```

Out[65]: (9557, 39)

In [66]:
```
# 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[66]: ['female']

In [67]:
```
# This is simply the opposite of male! We can remove the male flag.
for df in [df_income_train, df_income_test]:
    df.drop(columns = 'male',inplace=True)

df_income_train.shape
```

Out[67]: (9557, 130)

In [68]:
```
#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 z

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

df_income_train.shape
```

Out[68]: (9557, 129)

In [69]:
```
cols=['Id','idhogar']
for df in [df_income_train, df_income_test]:
    df.drop(columns = cols,inplace=True)

df_income_train.shape
```

Out[69]: (9557, 127)

In [70]:
```
df_income_train['Target'].isnull().any().sum()
```

Out[70]: 0

In [71]:
```
x_features=df_income_train.iloc[:,0:-1]
y_features=df_income_train.iloc[:,-1]
print(x_features.shape)
print(y_features.shape)
```

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

In [72]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classificati

x_train,x_test,y_train,y_test=train_test_split(x_features,y_features,test_size=0.
rmclassifier = RandomForestClassifier()
```

In [73]:
```
rmclassifier.fit(x_train,y_train)
```

Out[73]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [74]:
```
y_predict = rmclassifier.predict(x_test)
```

In [75]:
```
print(accuracy_score(y_test,y_predict))
```

```
0.9476987447698745
```

In [76]:
```
print(confusion_matrix(y_test,y_predict))
```

```
[[ 137    1    0   19]
 [    1  282    1   33]
 [    0    0  190   43]
 [    0    1    1 1203]]
```

In [77]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           1       0.99      0.87      0.93       157
           2       0.99      0.89      0.94       317
           3       0.99      0.82      0.89       233
           4       0.93      1.00      0.96      1205

    accuracy                           0.95      1912
   macro avg       0.98      0.89      0.93      1912
weighted avg       0.95      0.95      0.95      1912
```

In [78]:
```
y_predict_testdata = rmclassifier.predict(df_income_test)
```

In [80]:
```
y_predict_testdata
```

Out[80]:
```
array([4, 4, 4, ..., 4, 4, 4], dtype=int64)
```

In [81]:
```
# Predict the accuracy using random forest classifier.
# Check the accuracy using random forest with cross validation
from sklearn.model_selection import KFold,cross_val_score
```

In [82]:
```
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='accura
results=cross_val_score(rmclassifier,x_features,y_features,cv=kfold,scoring='accu
print(results.mean()*100)
```

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

In [83]:
```
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='accura
results=cross_val_score(rmclassifier,x_features,y_features,cv=kfold,scoring='accu
print(results.mean()*100)
```

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

In [84]:
```
rmclassifier.fit(x_features,y_features)
labels = list(x_features)
feature_importances = pd.DataFrame({'feature': labels, 'importance': rmclassifier
feature_importances=feature_importances[feature_importances.importance>0.015]
feature_importances.head()
```

Out[84]:

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

In [ ]: