Income_Qualification

May 4, 2021

[1]:
```python
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')
```

[2]:
```python
# Loading the dataset
df_income_train = pd.read_csv("train.csv")
df_income_test =  pd.read_csv("test.csv")
```

[3]:
```python
df_income_train.head()
```

[3]:
```
              Id      v2a1  hacdor  rooms  hacapo  v14a  refrig  v18q  v18q1  \
0  ID_279628684  190000.0       0      3       0     1       1     0    NaN
1  ID_f29eb3ddd  135000.0       0      4       0     1       1     1    1.0
2  ID_68de51c94       NaN       0      8       0     1       1     0    NaN
3  ID_d671db89c  180000.0       0      5       0     1       1     1    1.0
4  ID_d56d6f5f5  180000.0       0      5       0     1       1     1    1.0

   r4h1  …  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  SQBhogar_nin  \
0     0  …          100    1849               1        100             0
1     0  …          144    4489               1        144             0
2     0  …          121    8464               1          0             0
3     0  …           81     289              16        121             4
4     0  …          121    1369              16        121             4
```

```
   SQBovercrowding  SQBdependency  SQBmeaned  agesq  Target
0         1.000000            0.0      100.0   1849       4
1         1.000000           64.0      144.0   4489       4
2         0.250000           64.0      121.0   8464       4
3         1.777778            1.0      121.0    289       4
4         1.777778            1.0      121.0   1369       4

[5 rows x 143 columns]
```

[4]:
```
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
```

[5]:
```
df_income_test.head()
```

[5]:
```
             Id      v2a1  hacdor  rooms  hacapo  v14a  refrig  v18q  v18q1  \
0  ID_2f6873615       NaN       0      5       0     1       1     0    NaN
1  ID_1c78846d2       NaN       0      5       0     1       1     0    NaN
2  ID_e5442cf6a       NaN       0      5       0     1       1     0    NaN
3  ID_a8db26a79       NaN       0     14       0     1       1     1    1.0
4  ID_a62966799  175000.0       0      4       0     1       1     1    1.0

   r4h1  …  age  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  \
0     1  …    4            0      16               9          0
1     1  …   41          256    1681               9          0
2     1  …   41          289    1681               9          0
3     0  …   59          256    3481               1        256
4     0  …   18          121     324               1          0

   SQBhogar_nin  SQBovercrowding  SQBdependency  SQBmeaned  agesq
0             1             2.25           0.25     272.25     16
1             1             2.25           0.25     272.25   1681
2             1             2.25           0.25     272.25   1681
3             0             1.00           0.00     256.00   3481
4             1             0.25          64.00        NaN    324

[5 rows x 142 columns]
```

[6]:
```
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
```

[7]:

```
#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')
```

[8]:

```
df_income_train.select_dtypes('int64').head()
```

[8]:

```
   hacdor  rooms  hacapo  v14a  refrig  v18q  r4h1  r4h2  r4h3  r4m1  …  \
0       0      3       0     1       1     0     0     1     1     0  …
1       0      4       0     1       1     1     0     1     1     0  …
2       0      8       0     1       1     0     0     0     0     0  …
3       0      5       0     1       1     1     0     2     2     1  …
4       0      5       0     1       1     1     0     2     2     1  …

   area1  area2  age  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  \
0      1      0   43          100    1849               1        100
1      1      0   67          144    4489               1        144
2      1      0   92          121    8464               1          0
3      1      0   17           81     289              16        121
4      1      0   37          121    1369              16        121

   SQBhogar_nin  agesq  Target
```

```
0             0   1849       4
1             0   4489       4
2             0   8464       4
3             4    289       4
4             4   1369       4

[5 rows x 130 columns]
```

[9]:
```
for column in df_income_train:
    if column not in df_income_test:
        print('The output variable is', column)
```

```
The output variable is Target
```

[10]:
```
print(df_income_train['Target'].value_counts())
```

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

[11]:
```
# Finding columns with null values
null_counts=df_income_train.select_dtypes('int64').isnull().sum()
null_counts[null_counts > 0]
```

[11]:
```
Series([], dtype: int64)
```

[12]:
```
df_income_train.select_dtypes('float64').head()
```

[12]:
```
       v2a1  v18q1  rez_esc  meaneduc  overcrowding  SQBovercrowding  \
0  190000.0    NaN      NaN      10.0      1.000000         1.000000
1  135000.0    1.0      NaN      12.0      1.000000         1.000000
2       NaN    NaN      NaN      11.0      0.500000         0.250000
3  180000.0    1.0      1.0      11.0      1.333333         1.777778
4  180000.0    1.0      NaN      11.0      1.333333         1.777778

   SQBdependency  SQBmeaned
0            0.0      100.0
1           64.0      144.0
2           64.0      121.0
3            1.0      121.0
4            1.0      121.0
```

[13]:
```
# Finding columns with null values
null_counts=df_income_train.select_dtypes('float64').isnull().sum()
null_counts[null_counts > 0]
```

[13]:
```
v2a1         6860
v18q1        7342
rez_esc      7928
meaneduc        5
SQBmeaned       5
dtype: int64
```

[14]:
```
df_income_train.select_dtypes('object').head()
```

[14]:

	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

[15]:
```
#Find columns with null values
null_counts=df_income_train.select_dtypes('object').isnull().sum()
null_counts[null_counts > 0]
```

[15]:
```
Series([], dtype: int64)
```

[16]:
```
# With out Null values treatment we cannot get the correct answers
```

0.1 Define Variable Categories

There are several different categories of variables:

1. `Squared Variables`: derived from squaring variables in the data
2. `Id variables`: identifies the data and should not be used as features
3. `Household variables`
   - Boolean: Yes or No
   - Ordered Discrete: Integers with an ordering
   - Continuous numeric
4. `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

[19]:
```
# dependency column
```

[17]:
```
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)
```

[18]:
```
df_income_train[['dependency','edjefe','edjefa']]
```

[18]:

	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 x 3 columns]

[19]:
```
df_income_train[['dependency','edjefe','edjefa']].describe()
```

[19]:

	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

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

[23]:

	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

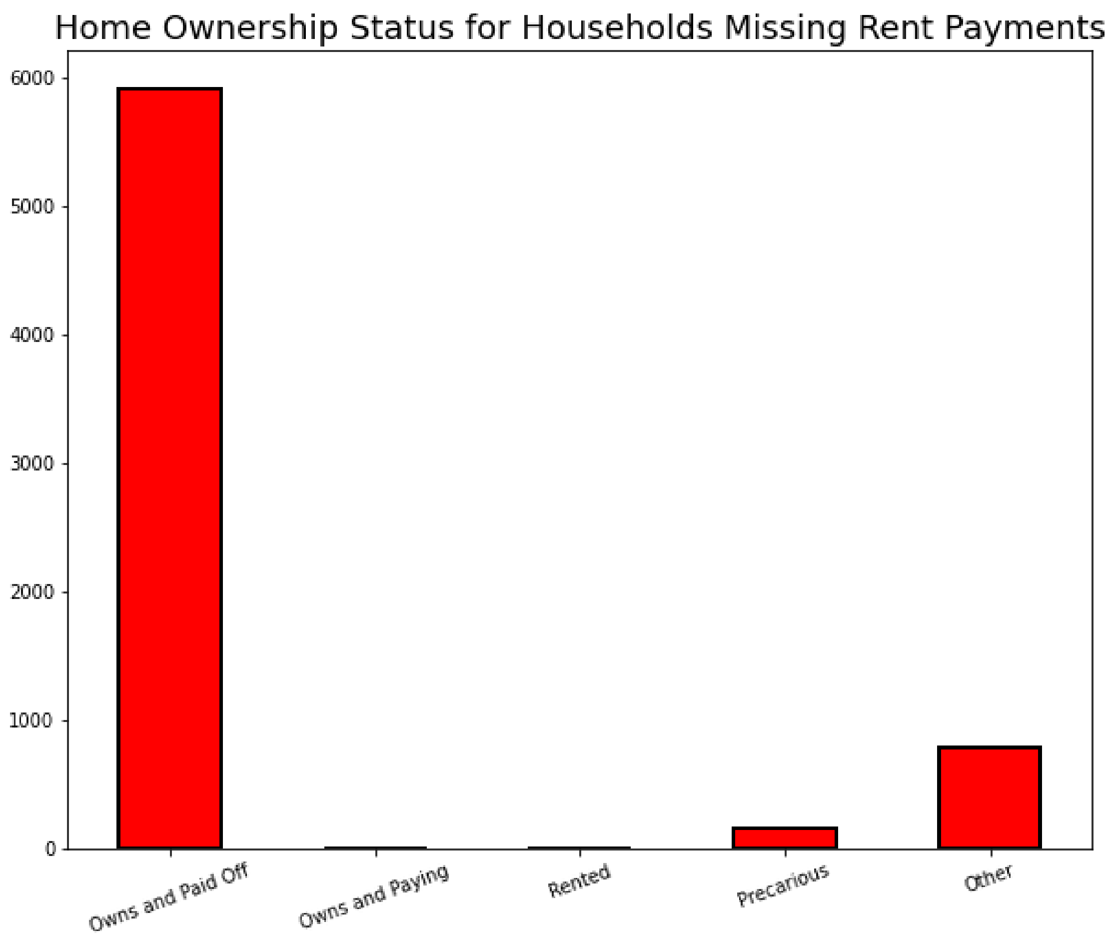
[24]:
```
# v2a1 column
```

[25]:
```
data = df_income_train[df_income_train['v2a1'].isnull()].head()

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

[25]:

	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

[26]:
```
# 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.
 ↪bar(figsize = (10, 8),
                                                                          color =␣
 ↪'red',
                                                              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('Home Ownership Status for Households Missing Rent Payments', size =␣
 ↪18);
```



[27]:
```
for df in [df_income_train, df_income_test]:
    df['v2a1'].fillna(value=0, inplace=True)

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

[27]:
```
v2a1    0
dtype: int64
```

[28]:
```
df_income_test[['v2a1']].isnull().sum()
```

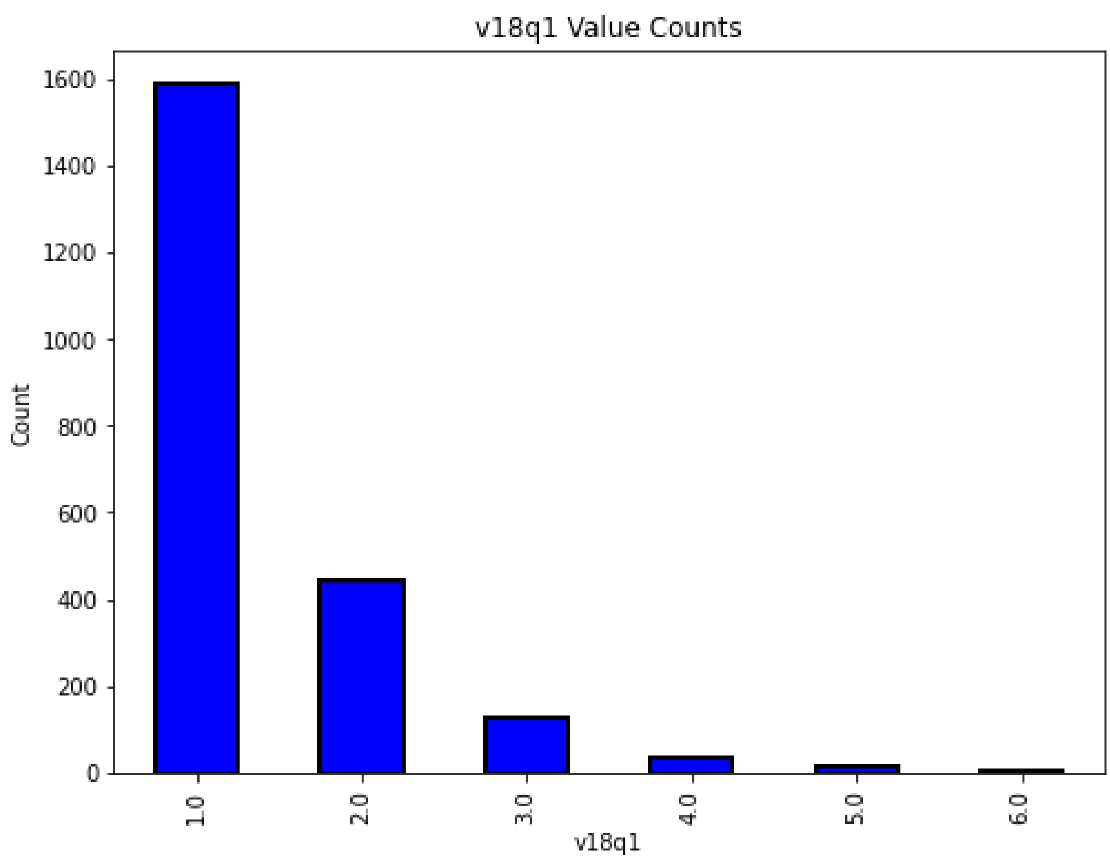
[28]:
```
v2a1    0
dtype: int64
```

[29]:
```
# v18q1 Column
```

[30]:
```
heads = df_income_train.loc[df_income_train['parentesco1'] == 1].copy()
heads.groupby('v18q')['v18q1'].apply(lambda x: x.isnull().sum())
```

[30]: v18q
0   2318
1   0
Name: v18q1, dtype: int64

[31]:
```
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();
```



[32]:
```
for df in [df_income_train, df_income_test]:
    df['v18q1'].fillna(value=0, inplace=True)

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

[32]: v18q1   0
dtype: int64

[33]:
```
df_income_test[['v18q1']].isnull().sum()
```

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

[34]:
```
# rez_esc column
```

[35]:
```
# Checking for no null values
df_income_train[df_income_train['rez_esc'].notnull()]['age'].describe()
```

[35]:
```
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
```

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

[36]:
```
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
```

[37]:
```
df_income_train.loc[(df_income_train['rez_esc'].isnull() &␣
↪((df_income_train['age'] > 7) & (df_income_train['age'] < 17)))]['age'].
↪describe()
```

[37]:
```
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
```

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

[38]:
```
                Id      v2a1  hacdor  rooms  hacapo  v14a  refrig  v18q  \
2514  ID_f012e4242  160000.0       0      6       0     1       1     1

      v18q1  r4h1  …  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  \
2514    1.0     0  …            0     100               9        121

      SQBhogar_nin  SQBovercrowding  SQBdependency  SQBmeaned  agesq  Target
2514             1             2.25           0.25     182.25    100       4

[1 rows x 143 columns]
```

[39]:
```
for df in [df_income_train, df_income_test]:
    df['rez_esc'].fillna(value=0, inplace=True)
df_income_train[['rez_esc']].isnull().sum()
```

[39]:
```
rez_esc    0
dtype: int64
```

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

[40]:
```
data = df_income_train[df_income_train['meaneduc'].isnull()].head()

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

[40]:
```
       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
```

[41]:
```
for df in [df_income_train, df_income_test]:
    df['meaneduc'].fillna(value=0, inplace=True)
df_income_train[['meaneduc']].isnull().sum()
```

[41]:
```
meaneduc    0
dtype: int64
```

[42]:
```
df_income_test[['meaneduc']].isnull().sum()
```

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

[43]:
```
# SQBmeaned Column
```

[44]:
```
data = df_income_train[df_income_train['SQBmeaned'].isnull()].head()

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

[44]:

	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

[45]:
```
for df in [df_income_train, df_income_test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
df_income_train[['SQBmeaned']].isnull().sum()
```

[45]:
```
SQBmeaned    0
dtype: int64
```

[46]:
```
df_income_test[['SQBmeaned']].isnull().sum()
```

[46]:
```
SQBmeaned    0
dtype: int64
```

[47]:
```
null_counts = df_income_train.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

[47]:
```
Series([], dtype: int64)
```

[48]:
```
null_counts = df_income_test.isnull().sum()
null_counts[null_counts > 0].sort_values(ascending=False)
```

[48]:
```
Series([], dtype: int64)
```

[49]:
```
# Groupby the household and figure out the number of unique values
all_equal = df_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 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.
```

[50]:
```
df_income_train[df_income_train['idhogar'] == not_equal.index[0]][['idhogar',␣
 ↪'parentesco1', 'Target']]
```

[50]:

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

[51]:
```
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(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.
```

[52]:
```
# 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.
```

[53]:
```
# 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)␣
 ↪& (df_income_train['parentesco1'] == 1.0)]['Target'])

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

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

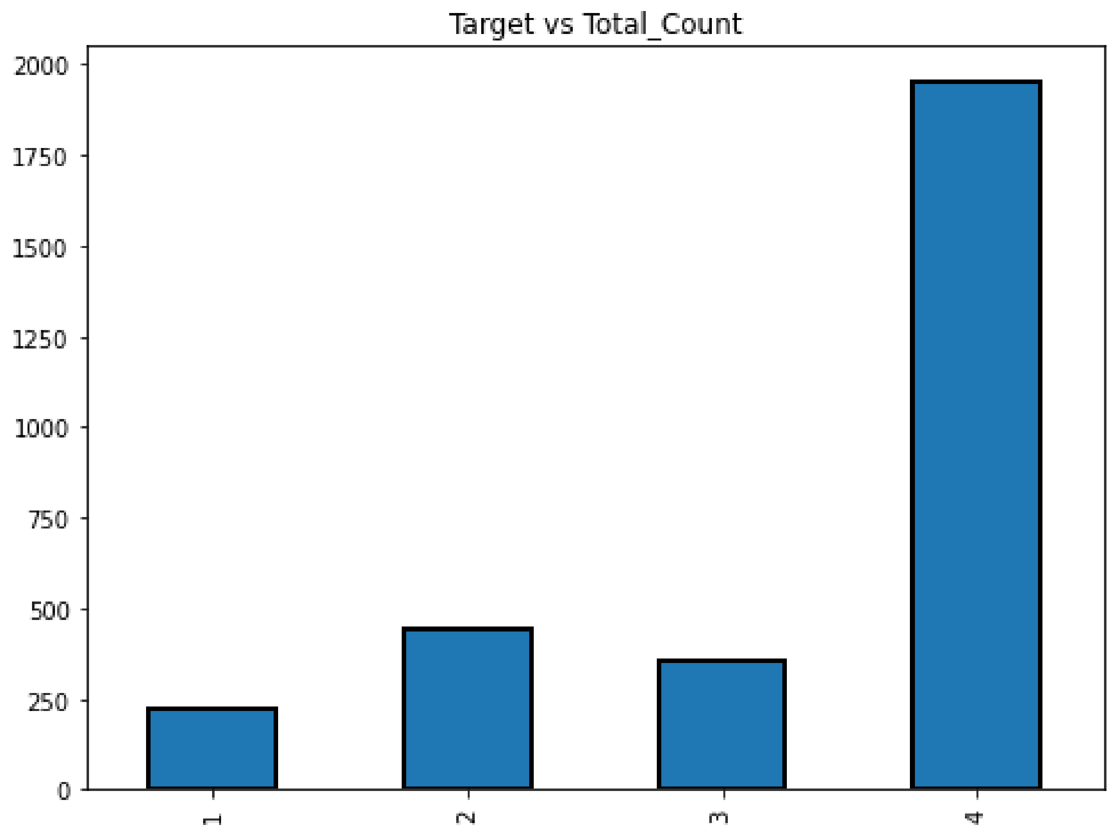
```
There are 0 households where the family members do not all have the same target.
```

[54]:
```
# 1 = extreme poverty 2 = moderate poverty 3 = vulnerable households 4 = non␣
 ↪vulnerable households
target_counts = heads['Target'].value_counts().sort_index()
target_counts
```

[54]:
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

[55]:
```
target_counts.plot.bar(figsize = (8, 6),linewidth = 2,edgecolor =␣
 ↪'k',title="Target vs Total_Count")
```

[55]:
```
<AxesSubplot:title={'center':'Target vs Total_Count'}>
```




```
[56]: # extreme poverty is the smallest count in the train dataset. The dataset is␣
      ↪biased.
      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)
```

```
[57]: 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']
```

[58]:
```
#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
```

[58]: (2973, 98)

[59]:
```
# Create correlation matrix
corr_matrix = heads.corr()
```

```python
# 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
```

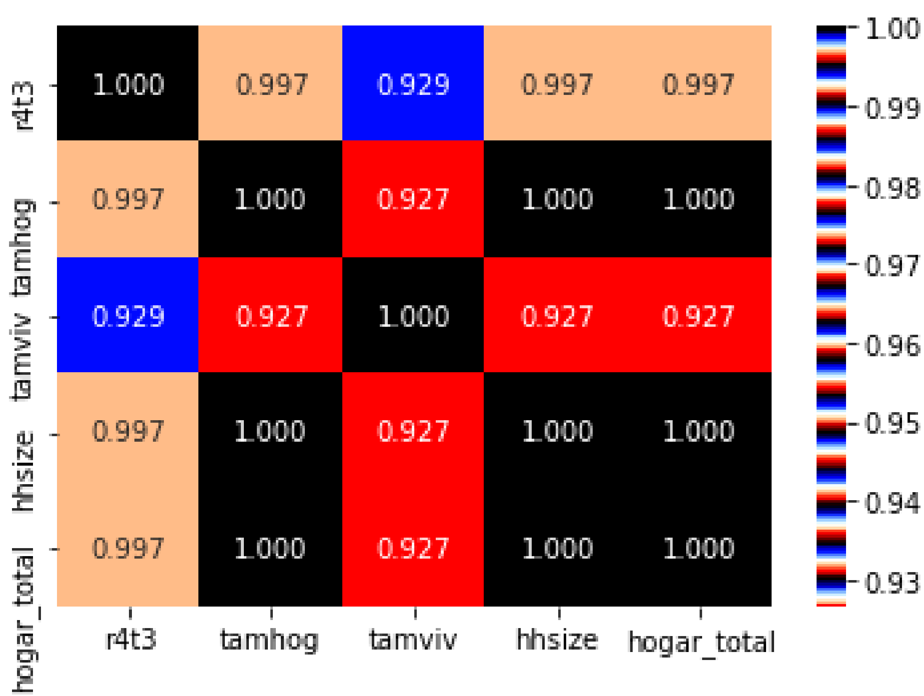
[59]: ['coopele', 'area2', 'tamhog', 'hhsize', 'hogar_total']

[60]:
```python
corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_matrix['tamhog'].abs()␣
 ↪> 0.9]
```

[60]:

	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

[61]:
```python
sns.heatmap(corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9,␣
 ↪corr_matrix['tamhog'].abs() > 0.9],
            annot=True, cmap = plt.cm.flag, fmt='.3f');
```



[62]:
```
cols=['tamhog', 'hogar_total', 'r4t3']
for df in [df_income_train, df_income_test]:
    df.drop(columns = cols,inplace=True)

df_income_train.shape
```

[62]: (9557, 131)

[63]:
```
#Check for redundant Individual variables
ind = df_income_train[id_ + ind_bool + ind_ordered]
ind.shape
```

[63]: (9557, 39)

[64]:
```
# 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
```

[64]: ['female']

[65]:
```
# 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
```

[65]: (9557, 130)

[66]:
```
#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 [df_income_train, df_income_test]:
    df.drop(columns = 'area2',inplace=True)

df_income_train.shape
```

[66]: (9557, 129)

[67]:
```
cols=['Id','idhogar']
for df in [df_income_train, df_income_test]:
    df.drop(columns = cols,inplace=True)

df_income_train.shape
```

[67]: (9557, 127)

[68]:
```
df_income_train['Target'].isnull().any().sum()
```

[68]: 0

[69]:
```
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,)
```

[70]:
```
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()
```

[71]:
```
rmclassifier.fit(x_train,y_train)
```

[71]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, 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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[72]:
```
y_predict = rmclassifier.predict(x_test)
```

[73]:
```
print(accuracy_score(y_test,y_predict))
```

```
0.946652719665272
```

[74]:
```
print(confusion_matrix(y_test,y_predict))
```

```
[[ 135    0    0   22]
 [    0  285    1   31]
 [    0    2  188   43]
 [    0    2    1 1202]]
```

[75]:
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           1       1.00      0.86      0.92       157
           2       0.99      0.90      0.94       317
           3       0.99      0.81      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
```

[76]:
```
y_predict_testdata = rmclassifier.predict(df_income_test)
```

[77]:
```
y_predict_testdata
```

[77]:
```
array([4, 4, 4, …, 4, 4, 4])
```

[78]:
```
# 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
```

[79]:
```
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)
```

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

[80]:
```
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)
```

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

[81]:
```
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()
```

[81]:

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

[ ]:
```
# Output According to the actions need to be Performed



# Identify the output variable. - Target Variable

# Understand the type of data. - 3 types of Data - Int,Float,Object

# Check if there are any biases in your dataset. - Yes bias was there

# Check whether all members of the house have the same poverty level. - No, It␣
 ↪was differing

# Check if there is a house without a family head. - Yes there were some house␣
 ↪very less number compared to the total no.of houses

# Set poverty level of the members and the head of the house within a family.-␣
 ↪Yes,It has been set on the basis of target varaible

# Count how many null values are existing in columns.

# Remove null value rows of the target variable.

# Predict the accuracy using random forest classifier. - 0.946652719665272 = 0.
 ↪95 According to classification report

# Check the accuracy using random forest with cross validation. - 0.94246862
```