In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

In [186]:
```
#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 a
re given enough aid. It’s tricky when a program focuses on the poorest
 segment of the population. This segment of the population can’t provid
e 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 t
he material of their walls and ceiling or the assets found in their hom
es to
#classify them and predict their level of need.

#While this is an improvement, accuracy remains a problem as the regio
n’s population grows and poverty declines.

#The Inter-American Development Bank (IDB)believes that new methods bey
ond traditional econometrics, based on a dataset of Costa Rican househo
ld 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 fa
mily.
```

```
#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.
# Data manipulation
import pandas as pd
import numpy as np
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Set a few plotting defaults
%matplotlib inline
```

In [187]:
```
# Read in data
train = pd.read_csv('/content/train.csv')
test = pd.read_csv('/content/test.csv')
train.head()
```

Out[187]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

In [188]:
```
test.head()
```

Out[188]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	1	2
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	1	2

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	1	2
3	ID_a8db26a79	NaN	0	14	0	1	1	1	1.0	0	1	1
4	ID_a62966799	175000.0	0	4	0	1	1	1	1.0	0	0	0

5 rows × 142 columns

In [189]:
```
train.columns
```

Out[189]:
```
Index(['Id', 'v2a1', 'hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v1
8q',
       'v18q1', 'r4h1',
       ...
       'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe', 'SQBhoga
r_nin',
       'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq', 'Targe
t'],
      dtype='object', length=143)
```

In [190]:
```
train.head()
```

Out[190]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

In [191]:
```
#Identify the output variable.
```

```
target = train['Target']
```

In [192]:
```
target
```

Out[192]:
```
0       4
1       4
2       4
3       4
4       4
       ..
9552    2
9553    2
9554    2
9555    2
9556    2
Name: Target, Length: 9557, dtype: int64
```

In [193]:
```
target.unique()
# here target variable is the ordinal varibale having 4 diffrent catego
ry
```

Out[193]:
```
array([4, 2, 3, 1])
```

In [194]:
```
train.describe
```

Out[194]:
```
<bound method NDFrame.describe of                 Id      v2a1  hacdor
...  SQBmeaned  agesq  Target
0     ID_279628684  190000.0       0  ...   100.0000   1849       4
1     ID_f29eb3ddd  135000.0       0  ...   144.0000   4489       4
2     ID_68de51c94       NaN       0  ...   121.0000   8464       4
3     ID_d671db89c  180000.0       0  ...   121.0000    289       4
4     ID_d56d6f5f5  180000.0       0  ...   121.0000   1369       4
...            ...       ...     ...  ...        ...    ...     ...
9552  ID_d45ae367d   80000.0       0  ...    68.0625   2116       2
9553  ID_c94744e07   80000.0       0  ...    68.0625      4       2
9554  ID_85fc658f8   80000.0       0  ...    68.0625   2500       2
9555  ID_ced540c61   80000.0       0  ...    68.0625    676       2
9556  ID_a38c64491   80000.0       0  ...    68.0625    441       2
```

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

In [195]:
```
# Shape of train and test data
print("Train:",train.shape)
print("Test:",test.shape)
```

```
Train: (9557, 143)
Test: (23856, 142)
```

In [196]:
```
test['Target'] = np.nan
test = train.append(test, ignore_index = True)
test.head()
```

Out[196]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

In [197]:
```
test['Target'].unique()
```

Out[197]:
```
array([ 4.,  2.,  3.,  1., nan])
```

In [198]:
```
# Check for Null values
train.isnull().sum()
```

Out[198]:
```
Id                   0
v2a1              6860
hacdor               0
rooms                0
```

```
hacapo                0
                    ...
SQBovercrowding       0
SQBdependency         0
SQBmeaned             5
agesq                 0
Target                0
Length: 143, dtype: int64
```

In [199]:
```
train.isnull().sum(axis=0).sort_values(ascending = False)
```

Out[199]:
```
rez_esc        7928
v18q1          7342
v2a1           6860
meaneduc          5
SQBmeaned         5
                ...
hogar_total       0
dependency        0
edjefe            0
edjefa            0
Id                0
Length: 143, dtype: int64
```

In [200]:
```
test['Target'] = np.nan
data = train.append(test, ignore_index = True)
data.head()
```

Out[200]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

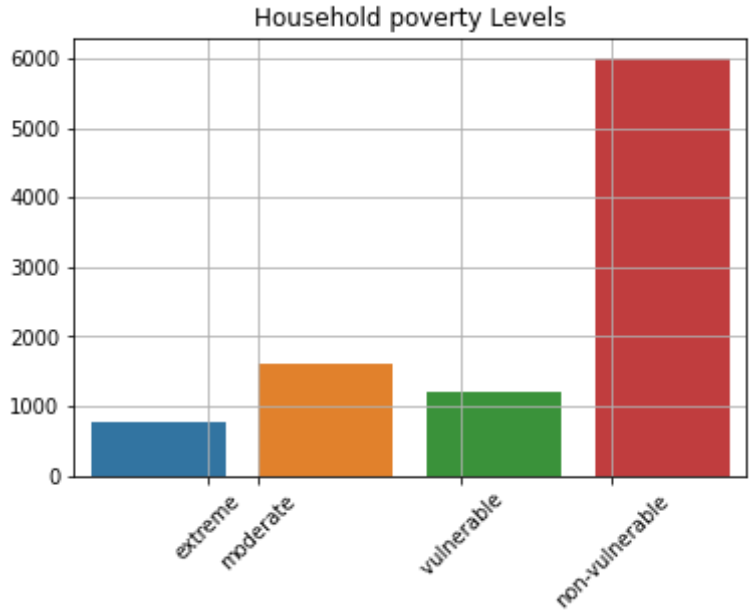
In [201]:
```
# Heads of household
heads = data.loc[data['parentesco1'] == 1].copy()
heads.head()
```

Out[201]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
5	ID_ec05b1a7b	180000.0	0	5	0	1	1	1	1.0	0	2	2
8	ID_1284f8aad	130000.0	1	2	0	1	1	0	NaN	0	1	1

5 rows × 143 columns

In [202]:
```
#different types of poverty levels sns.set(style="white",font_scale=1.2
5) #setting style and the palette sns.set_palette("rocket")
ax = sns.countplot(data= train, x = 'Target') #use seaborn library in p
ython for plotting
#setting the labels on x-axis
plt.xticks([0.3,0.6,1.8,2.7],['extreme', 'moderate', ' vulnerable', 'no
n-vulnerable'], rotation = 45)
plt.xlabel("")
plt.ylabel("")
plt.grid('True')
#set the title
plt.title('Household poverty Levels');
```



In [203]:
```
train_labels = data.loc[(data['Target'].notnull())]
```

In [204]:
```
train_labels
```

Out[204]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	
...	...	...	...	...	...	...	...	...	...	...	...	
9552	ID_d45ae367d	80000.0	0	6	0	1	1	0	NaN	0	2	
9553	ID_c94744e07	80000.0	0	6	0	1	1	0	NaN	0	2	

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4
9554	ID_85fc658f8	80000.0	0	6	0	1	1	0	NaN	0	2	
9555	ID_ced540c61	80000.0	0	6	0	1	1	0	NaN	0	2	
9556	ID_a38c64491	80000.0	0	6	0	1	1	0	NaN	0	2	

9557 rows × 143 columns

In [205]:
```
data['Target'].isnull()
```

Out[205]:
```
0         False
1         False
2         False
3         False
4         False
          ...
42965      True
42966      True
42967      True
42968      True
42969      True
Name: Target, Length: 42970, dtype: bool
```

In [206]:
```
train_labels = data.loc[(data['Target'].notnull()) & (data['parentesco1'] == 1), ['Target', 'idhogar']]
```

In [207]:
```
train_labels
```

Out[207]:

	Target	idhogar
0	4.0	21eb7fcc1
1	4.0	0e5d7a658
2	4.0	2c7317ea8
5	4.0	2b58d945f

	Target	idhogar
8	4.0	d6dae86b7
...	...	...
9535	1.0	9bbf7c6ca
9541	2.0	e87e70c06
9545	4.0	a8eeafc29
9551	2.0	212db6f6c
9552	2.0	d6c086aa3

2973 rows × 2 columns

In [208]:
```
# Value counts of target
label_counts = train_labels['Target'].value_counts().sort_index()
label_counts
```

Out[208]:
```
1.0     222
2.0     442
3.0     355
4.0    1954
Name: Target, dtype: int64
```

In [209]:
```
unique_values = train.groupby('idhogar')['Target']
```

In [210]:
```
pd.DataFrame(unique_values).head(10)
```

Out[210]:

	0	1
0	001ff74ca	7471 4 7472 4 Name: Target, dtype: int64
1	003123ec2	8159 2 8160 2 8161 2 8162 2 Name: ...
2	004616164	6472 2 6473 2 Name: Target, dtype: int64
3	004983866	6606 3 6607 3 Name: Target, dtype: int64

	0	1
4	005905417	7790 2 7791 2 7792 2 Name: Target, dt...
5	006031de3	3775 4 3776 4 3777 4 3778 4 Name: ...
6	006555fe2	2977 4 2978 4 2979 4 2980 4 2981 ...
7	00693f597	4706 4 4707 4 4708 4 4709 4 Name: ...
8	006b64543	7208 4 7209 4 Name: Target, dtype: int64
9	00941f1f4	7144 1 7145 1 7146 1 7147 1 Name: ...

In [211]:
```
unique_values = train.groupby('idhogar')['Target'].apply(lambda x: x.nu
nique() == 1)
```

In [212]:
```
unique_values.value_counts()
```

Out[212]:
```
True     2903
False      85
Name: Target, dtype: int64
```

In [213]:
```
train.head()
```

Out[213]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

In [214]:
```
numeric_data = train.select_dtypes(include=[np.number])
categorical_data = train.select_dtypes(exclude=[np.number])
```

In [215]:
```
numeric_data.head()
```

Out[215]:

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3	r4m1	r4m2
0	190000.0	0	3	0	1	1	0	NaN	0	1	1	0	0
1	135000.0	0	4	0	1	1	1	1.0	0	1	1	0	0
2	NaN	0	8	0	1	1	0	NaN	0	0	0	0	1
3	180000.0	0	5	0	1	1	1	1.0	0	2	2	1	1
4	180000.0	0	5	0	1	1	1	1.0	0	2	2	1	1

5 rows × 138 columns

In [216]:
```
categorical_data.head()
```

Out[216]:

	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 [217]:
```
#for i in numeric_data.columns:
#    plt.hist(numeric_data[i], bins = 10)
#    plt.title("Histogram for: " + i)
#    plt.show()
```

In [218]:
```
train.head()
```

Out[218]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	1	1
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	1	1
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	2	2
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	2	2

5 rows × 143 columns

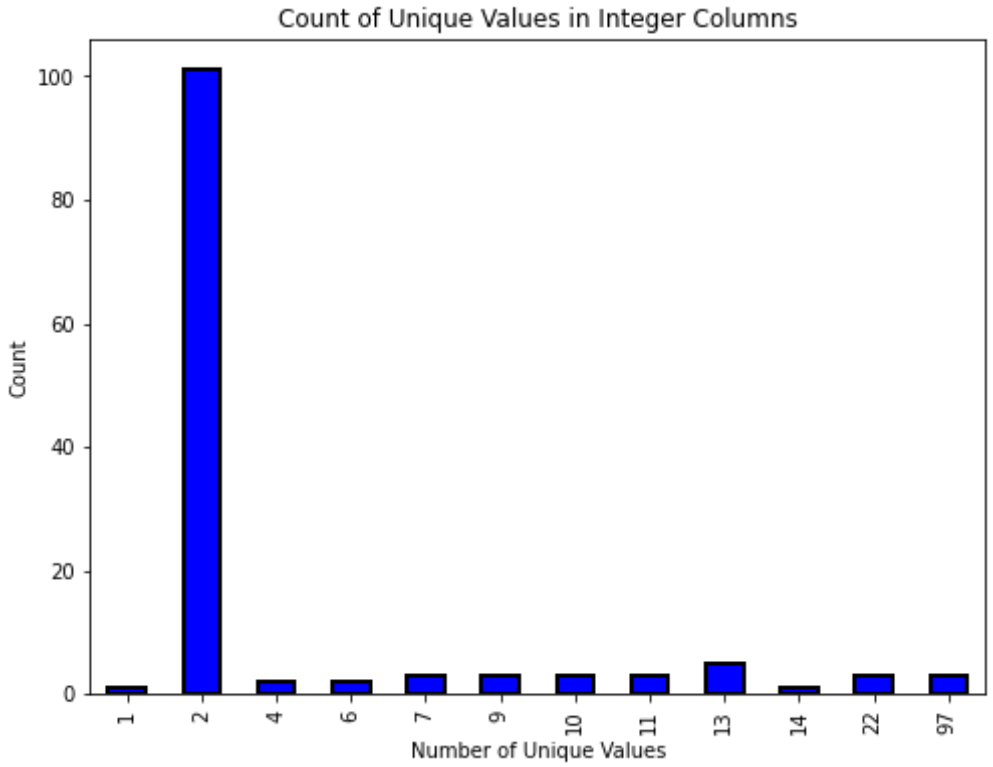
In [219]:
```
train.isnull().sum()
```

Out[219]:
```
Id                     0
v2a1                6860
hacdor                 0
rooms                  0
hacapo                 0
                    ...
SQBovercrowding        0
SQBdependency          0
SQBmeaned              5
agesq                  0
Target                 0
Length: 143, dtype: int64
```

In [220]:
```
##Number of tablets household owns(v18q1) , Monthly rent payment(v2a1),
 Years behind in school(rez_esc)
##have most missing values.
```

In [221]:
```
train.select_dtypes(np.int64).nunique().value_counts().sort_index().plo
t.bar(color = 'blue',

      figsize = (8, 6),

     edgecolor = 'k', linewidth = 2);
```

```
plt.xlabel('Number of Unique Values'); plt.ylabel('Count');
plt.title('Count of Unique Values in Integer Columns');
```



In [222]:
```
train.select_dtypes('object').head()
```

Out[222]:

	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 [223]:
```
print('Integer Type: ')
print(train.select_dtypes(np.int64).columns)
print('\n')
print('Float Type: ')
print(train.select_dtypes(np.float64).columns)
print('\n')
print('Object Type: ')
print(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_tota
l',
       '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='objec
t')
```

In [224]:
```
null_counts=train.select_dtypes('int64').isnull().sum()
null_counts[null_counts > 0]
```

Out[224]:
```
Series([], dtype: int64)
```

In [225]:
```
##we observe that there are no missing values present in the integer co
lumn
```

In [226]:
```
train.select_dtypes('float64').head()
```

Out[226]:

	v2a1	v18q1	rez_esc	meaneduc	overcrowding	SQBovercrowding	SQBdependency	SQBi
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 [227]:
```
null_counts=train.select_dtypes('float64').isnull().sum()
null_counts[null_counts > 0]
```

Out[227]:
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

In [228]:
```
##we observe that there are missing values in most of the float column
```

In [229]:
```
train.select_dtypes('object').head()
```

Out[229]:

	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 [230]:
```
null_counts=train.select_dtypes('object').isnull().sum()
null_counts[null_counts > 0]
```

Out[230]:
```
Series([], dtype: int64)
```

In [231]:
```
##we observe that there are no missing values present in the object col
umn
```

In [232]:
```
##We also noticed that object type features dependency, edjefe, edjefa
 have mixed values.
##Lets fix the data for features with null values and features with mix
ed values
```

In [233]:
```
##dependency, Dependency rate, calculated = (number of members of the h
ousehold
##younger than 19 or older than 64)/(number of member of household betw
een 19 and 64) 102.
##edjefe, years of education of male head of household, based on the in
teraction of
##escolari (years of education), head of household and gender, yes=1 an
d 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 an
d no=0
```

In [234]:
```
##For these three variables, it seems “yes” = 1 and “no” = 0. We can co
rrect the variables
##using a mapping and convert to floats.
mapping={'yes':1,'no':0}

for df in [train, test]:
    df['dependency'] =df['dependency'].replace(mapping).astype(np.float
64)
    df['edjefe'] =df['edjefe'].replace(mapping).astype(np.float64)
    df['edjefa'] =df['edjefa'].replace(mapping).astype(np.float64)

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

Out[234]:

	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 [235]:
```
#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 [236]:
```
data = train[train['v2a1'].isnull()].head()

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

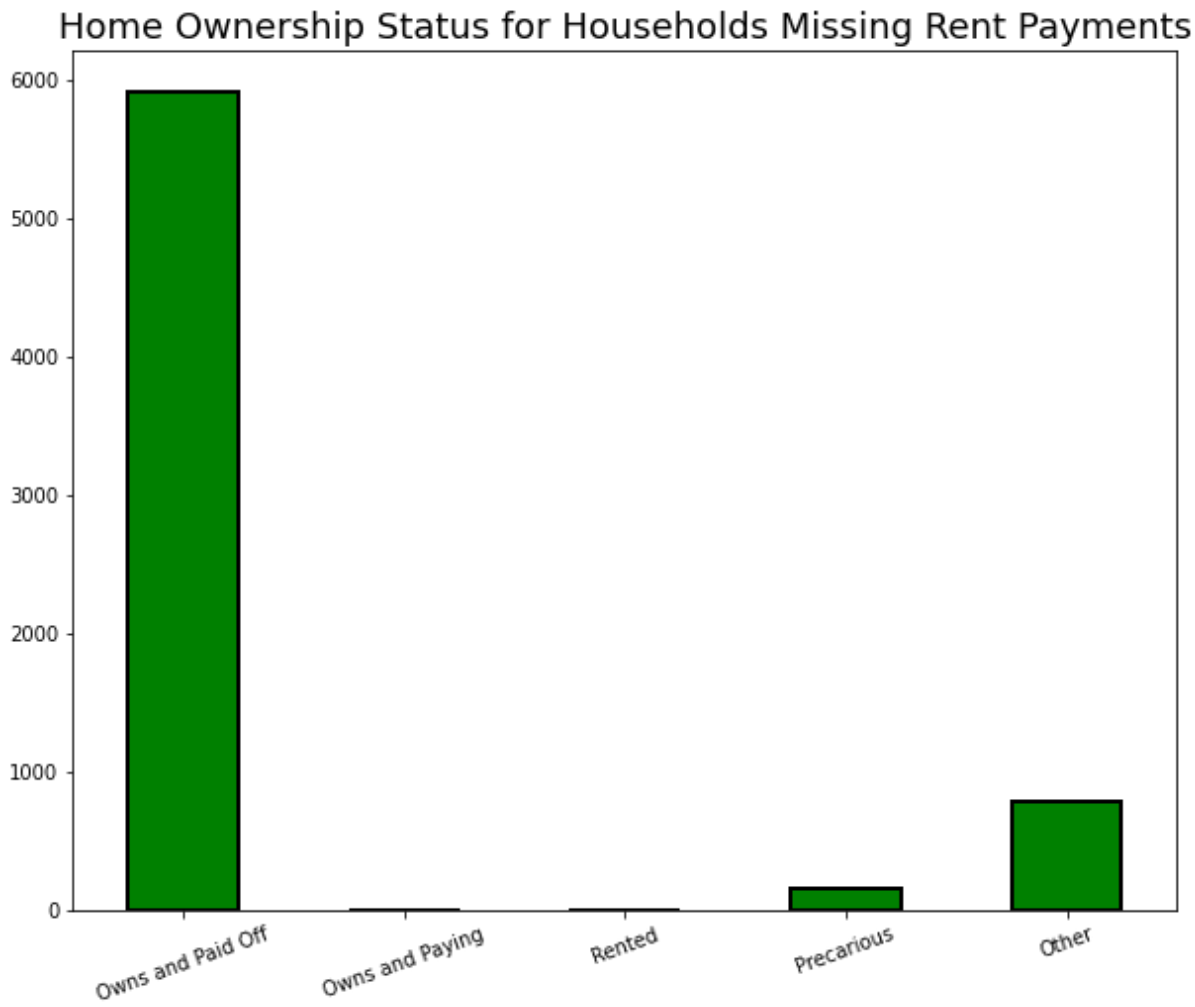
Out[236]:

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

	tipovivi1	tipovivi2	tipovivi3	tipovivi4	tipovivi5
26	1	0	0	0	0
32	1	0	0	0	0

In [237]:
```
#we observe that when the payment is done then there is no need of mont
hly
##payment hence we can impute the null values by 0
```

In [238]:
```
own_variables = [x for x in train if x.startswith('tipo')]

# Plot of the home ownership variables for home missing rent payments
train.loc[train['v2a1'].isnull(), own_variables].sum().plot.bar(figsize
 = (10, 8),

 color = 'green',
                                                                edgecolor
 = 'k', linewidth = 2);
plt.xticks([0, 1, 2, 3, 4],
           ['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precario
us', 'Other'],
           rotation = 20)
plt.title('Home Ownership Status for Households Missing Rent Payments',
 size = 18);
```



In [239]:
```
##we observe that most of the null values present in the first
##category where the payment is already done hence no need of monthly p
ayment.
```

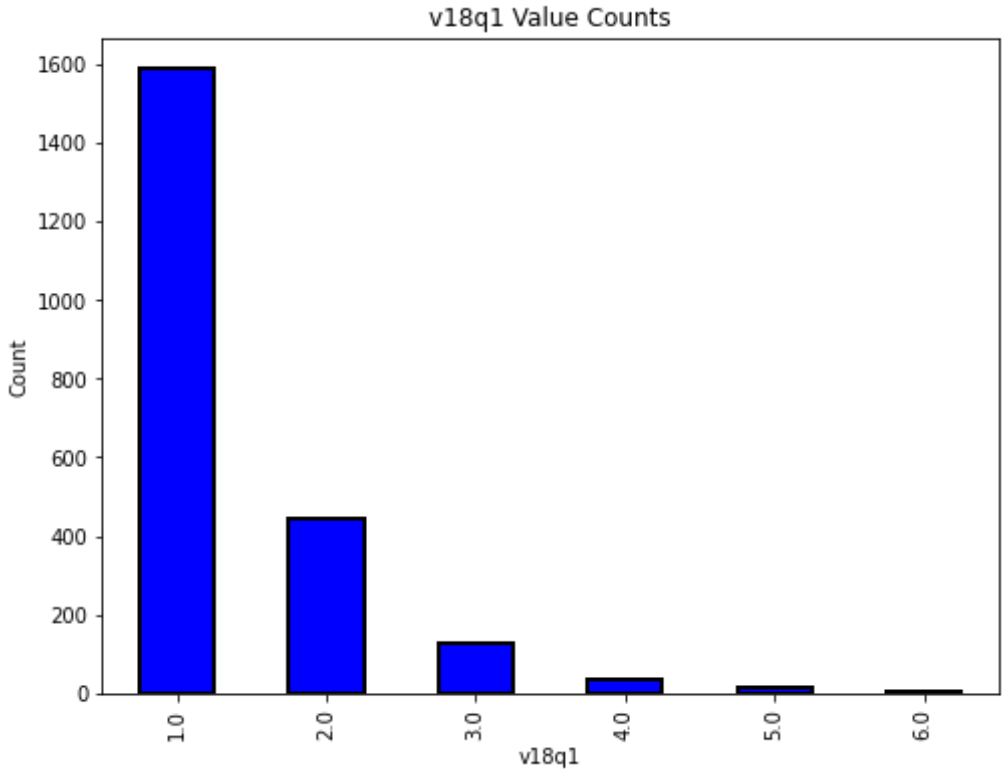
In [240]:
```
for df in [train, test]:
    df['v2a1'].fillna(value=0, inplace=True)

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

Out[240]:
```
v2a1    0
dtype: int64
```

In [241]:
```
# 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.loc[train['parentesco1'] == 1].copy()
heads.groupby('v18q')['v18q1'].apply(lambda x: x.isnull().sum())
```

Out[241]:
```
v18q
0    2318
1       0
Name: v18q1, dtype: int64
```

In [242]:
```
plt.figure(figsize = (8, 6))
col='v18q1'
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('Cou
nt')
plt.show();
```



In [243]:
```
# imputation of the null value with 0
for df in [train, test]:
    df['v18q1'].fillna(value=0, inplace=True)

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

Out[243]:
```
v18q1    0
dtype: int64
```

In [244]:
```
# Lets look at the data with not null values first.
train[train['rez_esc'].notnull()]['age'].describe()
```

Out[244]:
```
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 [245]:
```
train.loc[(train['rez_esc'].isnull() & ((train['age'] > 7) & (train['age'] < 17)))]['age'].describe()
```

Out[245]:
```
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 [246]:
```
# from the above we can observe that there is one null value lets imput e it
for df in [train, test]:
    df['rez_esc'].fillna(value=0, inplace=True)

train[['rez_esc']].isnull().sum()
```

Out[246]:
```
rez_esc    0
dtype: int64
```

In [247]:
```
data = train[train['SQBmeaned'].isnull()].head()

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

Out[247]:

	edjefe	edjefa	instlevel1	instlevel2

	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 [248]:
```
# imputation of the null value for SQBmeaned column
for df in [train, test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)

train[['SQBmeaned']].isnull().sum()
```

Out[248]:
```
SQBmeaned    0
dtype: int64
```

In [249]:
```
# lest check if there is any null value present overall in the data
null_counts = train.isnull().sum()
```

In [250]:
```
null_counts
```

Out[250]:
```
Id                 0
v2a1               0
hacdor             0
rooms              0
hacapo             0
                  ..
SQBovercrowding    0
SQBdependency      0
SQBmeaned          0
agesq              0
```

```
Target              0
Length: 143, dtype: int64
```

In [251]:
```
# Lets see if records belonging to same household has same target/scor
e.
all_equal = train.groupby('idhogar')['Target'].apply(lambda x: x.nuniqu
e() == 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 sa
me target.
```

In [252]:
```
#Lets check one household
train[train['idhogar'] == not_equal.index[0]][['idhogar', 'parentesco1'
, 'Target']]
```

Out[252]:

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

In [253]:
```
# check if all families has a head.

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

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

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

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

In [254]:
```
# Find households without a head and where Target value are different
households_no_head_equal = households_no_head.groupby('idhogar')['Targe
t'].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.
```

In [255]:
```
#Set poverty level of the members and the head of the house within a fa
mily.
# Iterate through each household
for household in not_equal.index:
    # Find the correct label (for the head of household)
    true_target = int(train[(train['idhogar'] == household) & (train['p
arentesco1'] == 1.0)]['Target'])

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


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

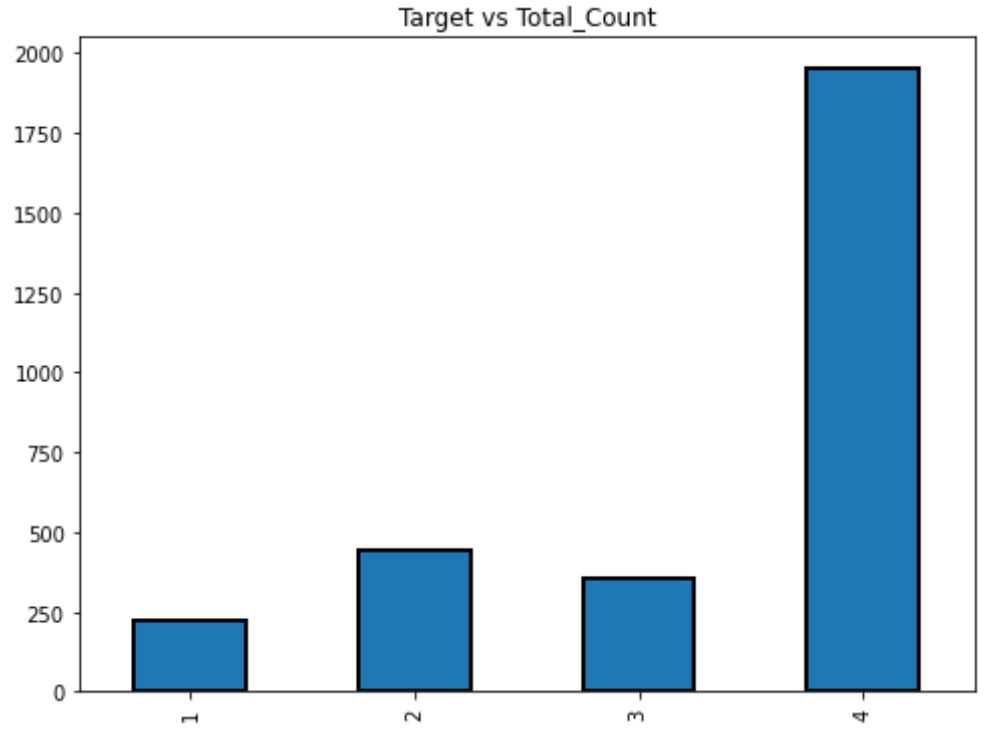
In [256]:
```
#lets look for bias in the dataset
#Lets look at the dataset and plot head of household and Target
```

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

Out[256]:
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

In [257]:
```
target_counts.plot.bar(figsize = (8, 6),linewidth = 2,edgecolor = 'k',t
itle="Target vs Total_Count")
```

Out[257]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd43289cac8>
```



In [258]:
```
# checking for multicolienity
```

```
# removing squared variable as it can lead to polynomial problem
print(train.shape)
cols=['SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe',
        'SQBhogar_nin', 'SQBovercrowding', 'SQBdependency', 'SQBmeaned'
, 'agesq']


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

print(train.shape)
```

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

In [259]:

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

ind_bool = ['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadociv
il2', 'estadocivil3',
            'estadocivil4', 'estadocivil5', 'estadocivil6', 'estadocivi
l7',
            'parentesco1', 'parentesco2',  'parentesco3', 'parentesco4'
, 'parentesco5',
            'parentesco6', 'parentesco7', 'parentesco8',  'parentesco9'
, 'parentesco10',
            'parentesco11', 'parentesco12', 'instlevel1', 'instlevel2',
 'instlevel3',
            'instlevel4', 'instlevel5', 'instlevel6', 'instlevel7', 'in
stlevel8',
            'instlevel9', 'mobilephone']

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

hh_bool = ['hacdor', 'hacapo', 'v14a', 'refrig', 'paredblolad', 'paredz
ocalo',
           'paredpreb','pisocemento', 'pareddes', 'paredmad',
           'paredzinc', 'paredfibras', 'paredother', 'pisomoscer', 'pis
oother',
           'pisonatur', 'pisonotiene', 'pisomadera',
```

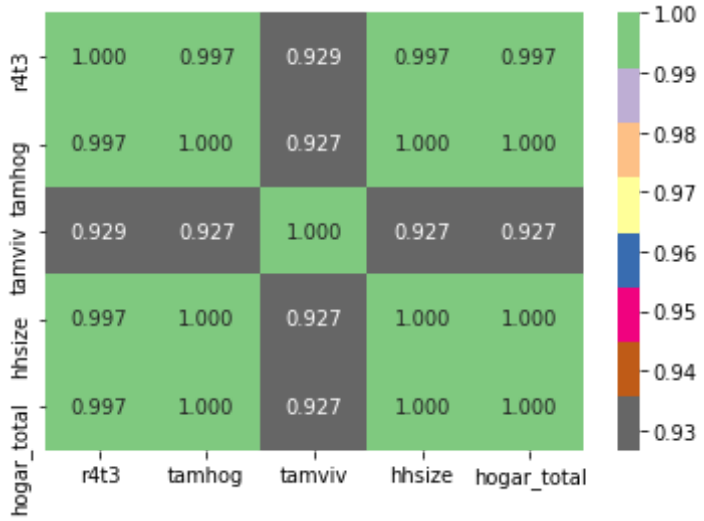
```
            'techozinc', 'techoentrepiso', 'techocane', 'techootro', 'ci
elorazo',
            'abastaguadentro', 'abastaguafuera', 'abastaguano',
             'public', 'planpri', 'noelec', 'coopele', 'sanitario1',
            'sanitario2', 'sanitario3', 'sanitario5',   'sanitario6',
            'energcocinar1', 'energcocinar2', 'energcocinar3', 'energcoc
inar4',
            'elimbasu1', 'elimbasu2', 'elimbasu3', 'elimbasu4',
            'elimbasu5', 'elimbasu6', 'epared1', 'epared2', 'epared3',
            'etecho1', 'etecho2', 'etecho3', 'eviv1', 'eviv2', 'eviv3',
            'tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi4', 'tipoviv
i5',
            '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', 'q
mobilephone']

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

In [260]:
```
#Check for redundant household variables
heads = train.loc[train['parentesco1'] == 1, :]
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

Out[260]:
```
(2973, 98)
```

In [261]:
```
# Create correlation matrix
corr_matrix = heads.corr()
```

In [262]:
```
sns.heatmap(corr_matrix.loc[corr_matrix['tamhog'].abs() > 0.9, corr_mat
rix['tamhog'].abs() > 0.9],
            annot=True, cmap = plt.cm.Accent_r, fmt='.3f');
```



In [263]:
```
# There are several variables here having to do with the size of the ho
use:
# 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.
# lets remove them
cols=['tamhog', 'hogar_total', 'r4t3']
for df in [train, test]:
    df.drop(columns = cols,inplace=True)

train.shape
```

Out[263]: (9557, 131)

In [264]:
```
#Check for redundant Individual variables
ind = train[id_ + ind_bool + ind_ordered]
ind.shape
```

Out[264]: (9557, 39)

In [265]:
```
# we can remove the male column as we already have the female column wh
ich can specify the male and female category
for df in [train, test]:
    df.drop(columns = 'male',inplace=True)

train.shape
```

Out[265]:
```
(9557, 130)
```

In [266]:
```
# 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, test]:
    df.drop(columns = 'area2',inplace=True)

train.shape
```

Out[266]:
```
(9557, 129)
```

In [267]:
```
#Finally lets delete 'Id', 'idhogar' for model building
cols=['Id','idhogar']
for df in [train, test]:
    df.drop(columns = cols,inplace=True)

train.shape
```

Out[267]:
```
(9557, 127)
```

In [268]:
```
# Predict the accuracy using random forest classifier with cross valida
tion
x_features=train.iloc[:,0:-1]
y_features=train.iloc[:,-1]

print(x_features.shape)
print(y_features.shape)
```

```
(9557, 126)
```

```
(9557,)
```

In [269]:
```
test.shape
```

Out[269]:
```
(33413, 127)
```

In [270]:
```
y_features.head(5)
```

Out[270]:
```
0    4
1    4
2    4
3    4
4    4
Name: Target, dtype: int64
```

In [271]:
```
x_features.isnull().values.any()
```

Out[271]:
```
True
```

In [272]:
```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
x_features = imp.fit_transform(x_features)
```

In [273]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,cl
assification_report

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

In [275]:
```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
```

```
x_train = imp.fit_transform(x_train)
x_test = imp.fit_transform(x_test)
```

In [276]:
```
rmclassifier.fit(x_train,y_train)
```

Out[276]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       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)
```

In [277]:
```
y_predict = rmclassifier.predict(x_test)
```

In [278]:
```
print(accuracy_score(y_test,y_predict))
print(confusion_matrix(y_test,y_predict))
print(classification_report(y_test,y_predict))
```

```
0.948744769874477
[[ 132    0    0   25]
 [   1  288    1   27]
 [   0    1  191   41]
 [   0    1    1 1203]]
              precision    recall  f1-score   support

           1       0.99      0.84      0.91       157
           2       0.99      0.91      0.95       317
           3       0.99      0.82      0.90       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 [279]:
```
test['Target'] = np.nan
test = train.append(test, ignore_index = True)
test
```

Out[279]:

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3	r4m1	r4
0	190000.0	0	3	0	1	1	0	0.0	0	1	1	0	
1	135000.0	0	4	0	1	1	1	1.0	0	1	1	0	
2	0.0	0	8	0	1	1	0	0.0	0	0	0	0	
3	180000.0	0	5	0	1	1	1	1.0	0	2	2	1	
4	180000.0	0	5	0	1	1	1	1.0	0	2	2	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
42965	0.0	1	2	1	1	1	0	0.0	0	2	2	1	
42966	0.0	0	3	0	1	1	0	0.0	0	1	1	0	
42967	0.0	0	3	0	1	1	0	0.0	0	1	1	0	
42968	0.0	0	3	0	1	1	0	0.0	0	1	1	0	
42969	0.0	0	3	0	1	1	0	0.0	0	1	1	0	

42970 rows × 127 columns

In [280]:
```
#Imputing for NaN values
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
test1 = imp.fit_transform(test)
```

In [281]:
```
test.columns
```

Out[281]:
```
Index(['v2a1', 'hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q',
```

```
'v18q1',
       'r4h1', 'r4h2',
       ...
       'qmobilephone', 'lugar1', 'lugar2', 'lugar3', 'lugar4', 'lugar
5',
       'lugar6', 'area1', 'age', 'Target'],
      dtype='object', length=127)
```

In [282]:
```
test_fill = test
```

In [283]:
```
for column in test_fill.columns:
    test_fill[column].fillna(test_fill[column].mode()[0], inplace=True)
```

In [284]:
```
test_fill = test_fill.drop('Target',axis = 1)
```

In [296]:
```
y_predict_testdata = rmclassifier.predict(test_fill)
y_predict_testdata
```

Out[296]:
```
array([4, 4, 4, ..., 4, 4, 4])
```

In [286]:
```
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan,strategy="most_frequent")
x_features1 = imp.fit_transform(x_features)
```

In [287]:
```
from sklearn.model_selection import KFold,cross_val_score
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_features1,y_features,cv=kfold,scor
ing='accuracy'))
results=cross_val_score(rmclassifier,x_features1,y_features,cv=kfold,sc
oring='accuracy')
print(results.mean()*100)
```

```
[0.94299163 0.95031381 0.94767138 0.94034537 0.95028781]
94.63219983841623
```

In [288]:
```
num_trees= 100

rmclassifier=RandomForestClassifier(n_estimators=100, random_state=10,n
_jobs = -1)
print(cross_val_score(rmclassifier,x_features1,y_features,cv=kfold,scor
ing='accuracy'))
results=cross_val_score(rmclassifier,x_features1,y_features,cv=kfold,sc
oring='accuracy')
print(results.mean()*100)
```

```
[0.94299163 0.95031381 0.94767138 0.94034537 0.95028781]
94.63219983841623
```

In [300]:
```
pd.DataFrame(test1).head()
```

Out[300]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	190000.0	0.0	3.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	10.0	0.0
1	135000.0	0.0	4.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	12.0	0.0
2	0.0	0.0	8.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	1.0	11.0	0.0
3	180000.0	0.0	5.0	0.0	1.0	1.0	1.0	1.0	0.0	2.0	2.0	1.0	1.0	2.0	1.0	3.0	4.0	9.0	1.0
4	180000.0	0.0	5.0	0.0	1.0	1.0	1.0	1.0	0.0	2.0	2.0	1.0	1.0	2.0	1.0	3.0	4.0	11.0	0.0

5 rows × 127 columns

In [304]:
```
test_fill = test
```

In [305]:
```
for column in test_fill.columns:
    test_fill[column].fillna(test_fill[column].mode()[0], inplace=True)
```

In [306]:
```
test_fill = test_fill.drop('Target',axis = 1)
```

In [308]:
```
rmclassifier.fit(x_train,y_train)
```

```
y_predict_testdata = rmclassifier.predict(test_fill)
y_predict_testdata
```

Out[308]:
```
array([4, 4, 4, ..., 4, 4, 4])
```