Income Qualification Project

Loading required libraries

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy.stats import zscore,itemfreq
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classification_report
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import KFold,cross_val_score

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

Loading data

In [2]:
```
df_income_train = pd.read_csv("D:/AI Engineer Masters/Project 02 - Income Qualification/train.csv")
df_income_train.head()
```

Out[2]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBhogar_nin
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	0
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	4489	1	144	0
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	0
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	289	16	121	4
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	1369	16	121	4

5 rows × 143 columns

In [3]:
```
df_income_test =  pd.read_csv("D:/AI Engineer Masters/Project 02 - Income Qualification/test.csv")
df_income_test.head()
```

Out[3]:

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

5 rows × 142 columns

Identify the output variable.

In [4]:
```
# Ideally speakign the output variable should be the one which is availble in train dataset and not availble in test dataset.
output_var=[x for x in df_income_train.columns if x not in df_income_test]
output_var
```

Out[4]:
```
['Target']
```

Understand the type of data.

In [5]:
```
# Listing the types of columns collectively
df_income_train.dtypes.value_counts()
```

Out[5]:
```
int64      130
float64      8
object       5
dtype: int64
```

In [6]:
```
# Listing more info
df_income_train.info()
```

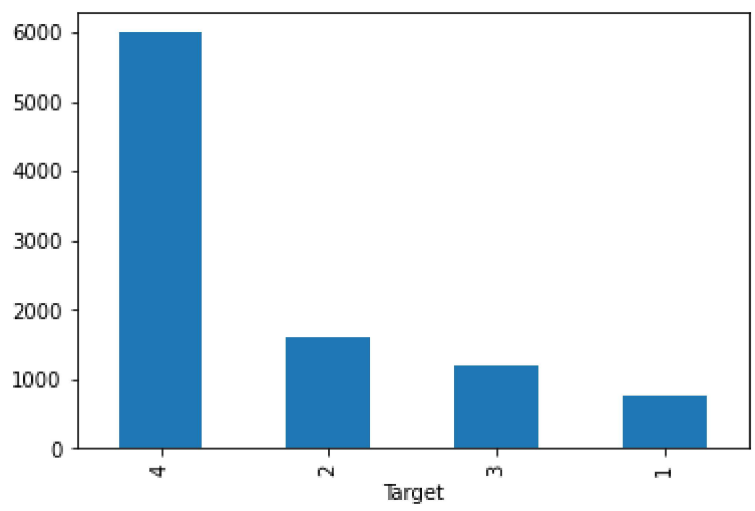
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9557 entries, 0 to 9556
Columns: 143 entries, Id to Target
dtypes: float64(8), int64(130), object(5)
memory usage: 10.4+ MB
```

Check if there are any biases in your dataset.

In [7]:
```
# Checking the distribution of the output variable
target_counts=df_income_train.value_counts('Target')
print(target_counts)
target_counts.plot.bar()
```

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

Out[7]: <AxesSubplot:xlabel='Target'>



Target variable is distributed among values 1,2,3,4. It has way more 4s than others. Means the dataset is biased.

Check whether all members of the house have the same poverty level.

In [8]:
```
# Finding non-unique values on Target (poverty level) on grouping house hold level id
df_temp1=df_income_train.groupby(['idhogar']).agg('nunique')[['Target']].reset_index()
# Occurance 1 means unique. So lets pick the > 1 items
df_temp2=df_temp1[df_temp1.Target>1]
#Printing a sample
print(df_temp2.head())
# Let see how many non-unique
print('Total house holds with different poverty levels are : ',len(df_temp2.index))
```

```
       idhogar  Target
17   0172ab1d9       2
42   03f4e5f4d       2
54   0511912b6       2
92   078a0b6e2       2
122  09e25d616       2
Total house holds with different poverty levels are :  85
```

In [9]:
```
# Print how many are unique
print(df_temp1.Target.value_counts())
```

```
1    2903
2      84
3       1
Name: Target, dtype: int64
```

Means 2903 house holds have unique proverty level and 84 has 2 different poverty level and 1 has 3 different poverty levels.

Check if there is a house without a family head.

In [10]:
```
#Print number of house holds with head
print('Number of house holds with head:',len(df_income_train[df_income_train.parentesco1==1]['idhogar'].unique()))
#Print unique number of house holds
print('Number of unique house holds:',len(df_income_train['idhogar'].unique()))
```

```
Number of house holds with head: 2973
Number of unique house holds: 2988
```

Means there are 15 house holds with out head

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

In [11]:
```
# Preapring a df with unique house hold and its target for the head of the house
df_household_true_target=df_income_train[df_income_train.parentesco1==1][['idhogar','Target']].drop_duplicates()
# Dropping the exisiting Target column from the original df
df_income_train=df_income_train.drop(columns=['Target'])
# Merging it with the true tareget df so that it will get the Target of the head of the house hold
df_income_train=df_income_train.merge(df_household_true_target,on=['idhogar'])
```

In [12]:
```
# Finding non-unique values on Target (poverty level) on grouping house hold level id
df_temp1=df_income_train.groupby(['idhogar']).agg('nunique')[['Target']].reset_index()
# Occurance 1 means unique. So lets pick the > 1 items
df_temp2=df_temp1[df_temp1.Target>1]
# Let see how many non-unique
print('Total house holds with different poverty levels are : ',len(df_temp2.index))
```

```
Total house holds with different poverty levels are :  0
```

Ths the poverty level of the members are set to the ehad of the house within the family

Count how many null values are existing in columns.

In [13]:
```
# List of columns wiht null values
```

```
df_income_train.columns[df_income_train.isnull().any()].to_list()
```

Out[13]:
```
['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'SQBmeaned']
```

v2a1, Monthly rent payment
v18q1, number of tablets household owns rez_esc,
rez_esc, Years behind in school
meaneduc,average years of education for adults (18+)
SQBmeaned, square of the mean years of education of adults (>=18) in the household

In [14]:
```
# Lisiting the 5 columns wiht their count of null values
df_income_train.isnull().sum().sort_values(ascending = False).head(5)
```

Out[14]:
```
rez_esc      7921
v18q1        7319
v2a1         6843
meaneduc        5
SQBmeaned       5
dtype: int64
```

Remove null value rows of the target variable.

In [15]:
```
# Counting null values in the target variable
print('Number of null values in Target:',df_income_train['Target'].isnull().sum())
```

```
Number of null values in Target: 0
```

There are no null values in the target variable. So, nothing to remove

Predict the accuracy using random forest classifier.

In order to create a prediction model, we will need to do a few steps before it.

Step 1. Data Pre-processing

a) Null handling

In [16]:
```
# Lets first take care of the null values
df_train_null = pd.DataFrame(df_income_train.isnull().sum()).rename(columns={0:'Null_Total'})
df_train_null['Null_Percent'] = round(100*(df_train_null['Null_Total']/df_income_train.shape[0]),2)
df_train_null.sort_values('Null_Percent',ascending= False).head(6)
```

Out[16]:

	Null_Total	Null_Percent
rez_esc	7921	83.08
v18q1	7319	76.77
v2a1	6843	71.77
SQBmeaned	5	0.05
meaneduc	5	0.05
Id	0	0.00

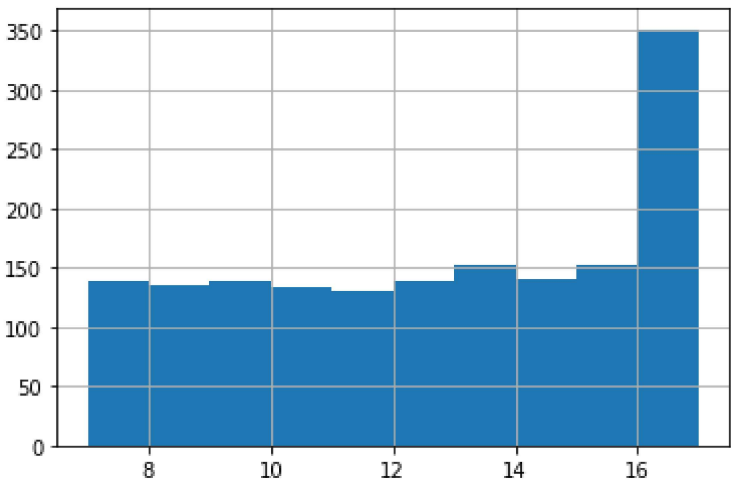
#1) rez_esc, Years behind in school

In [17]:
```
# The age of the person (age) could be a good aspect for analysiing this column's distribution
# Lets first see for what age group we have the values are filled for this column
df_income_train[df_income_train['rez_esc'].notnull()]['age'].describe()
```

Out[17]:
```
count    1613.000000
mean       12.262244
std         3.218671
min         7.000000
25%         9.000000
50%        12.000000
75%        15.000000
max        17.000000
Name: age, dtype: float64
```

In [18]:
```
df_income_train[df_income_train['rez_esc'].notnull()]['age'].hist()
```

Out[18]:
```
<AxesSubplot:>
```



In [19]:
```
# This means that the rez_esc has values filled for the age group 7 to 17
# and it makes sense as it is relevant for only the school going children.
# in other words, for the non-filled age groups , this value s irrelavant and we can safely fill 0 for them.
df_income_train['rez_esc'].fillna(0,inplace=True) # filling zero for missing values
print('Number of null values for rez_esc:', df_income_train['rez_esc'].isnull().sum())
```

```
Number of null values for rez_esc: 0
```

#2) v18q1, number of tablets household owns

In [20]:
```
# v18q (owns a tablet) could be a column which can give insights on this field.
# Let see how the nulls are distributed against the v18q values.
df_income_train.groupby('v18q').agg({'v18q1': lambda x: x.isnull().sum()}).reset_index().rename(columns={'v18q1':'V18q1_Null_Total'}
```

Out[20]:

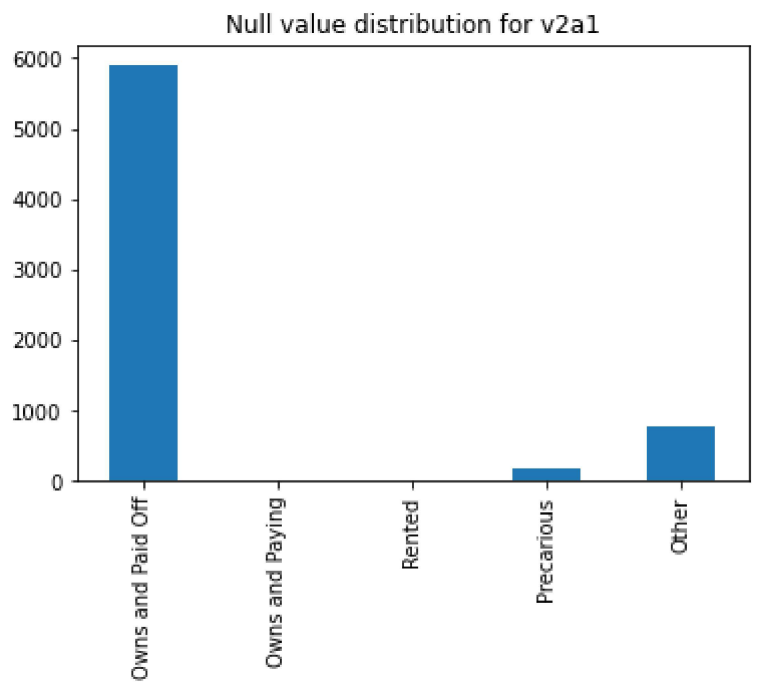
	v18q	V18q1_Null_Total
0	0	7319.0
1	1	0.0

In [21]:
```
# This means that the value is missing for the house holds who do not own a tablet at all
# And it completely makes sense.
# So, we can safely fill the missing values with zero.
df_income_train['v18q1'].fillna(0,inplace=True) # filling zero for missing values
print('Number of null values for v18q1:', df_income_train['v18q1'].isnull().sum())
```

```
Number of null values for v18q1: 0
```

#3) v2a1, Monthly rent payment

In [22]:
```
# feel like this could be related to the ownership columns as the house holds who own ahouse won't be paying any rent.
# tipovivi1, =1 own and fully paid house
# tipovivi2, =1 own, paying in installments
# tipovivi3, =1 rented
# tipovivi4, =1 precarious
# tipovivi5, =1 other(assigned, borrowed)
# So lets see the distibution on them.
ownership_columns = ['tipovivi1','tipovivi2','tipovivi3','tipovivi4','tipovivi5']
df_income_train.loc[df_income_train['v2a1'].isnull(), ownership_columns].sum().plot.bar()
plt.xticks([0, 1, 2, 3, 4],['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precarious', 'Other'])
plt.title('Null value distribution for v2a1')
plt.show()
```

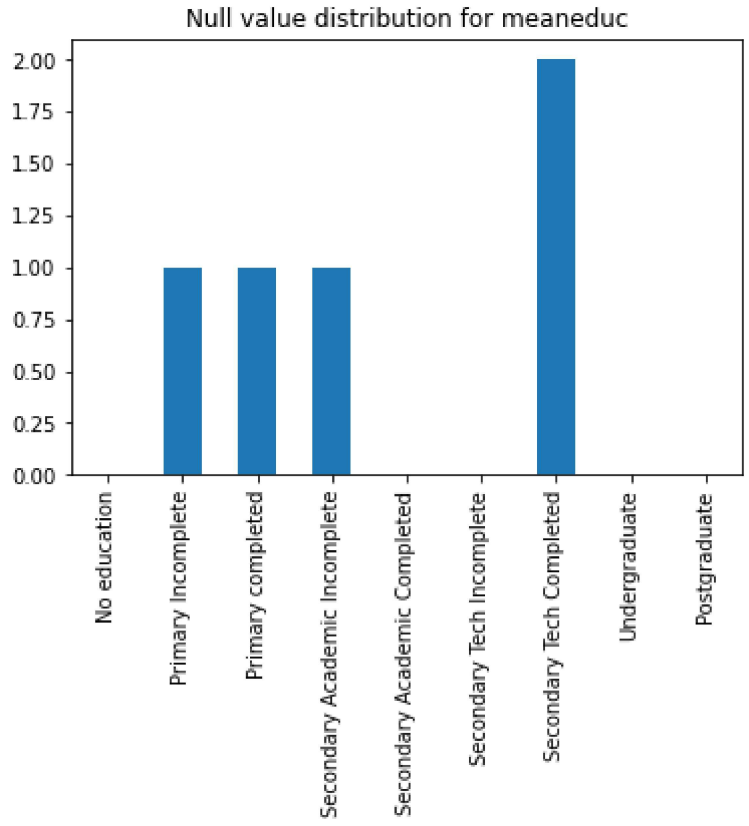


In [23]:
```
# This makes complete sense again.
# There are no missing values for monthly rent payements, if the house hold is either rented or owns & paying
# So it will be safe to fill the missign values with 0
df_income_train['v2a1'].fillna(0,inplace=True) # filling zero for missing values
print('Number of null values for v2a1:', df_income_train['v2a1'].isnull().sum())
```

```
Number of null values for v2a1: 0
```

#4) meaneduc,average years of education for adults (18+)

In [24]:
```
# feel like this could be related to the level of education columns.
# instlevel1, =1 no level of education
# instlevel2, =1 incomplete primary
# instlevel3, =1 complete primary
# instlevel4, =1 incomplete academic secondary level
# instlevel5, =1 complete academic secondary level
# instlevel6, =1 incomplete technical secondary level
# instlevel7, =1 complete technical secondary level
# instlevel8, =1 undergraduate and higher education
# instlevel9, =1 postgraduate higher education
edu_level_columns = [x for x in df_income_train if x.startswith('instlevel')]
df_income_train.loc[df_income_train['meaneduc'].isnull(), edu_level_columns].sum().plot.bar()
plt.xticks([0,1,2,3,4,5,6,7,8],['No education', 'Primary Incomplete', 'Primary completed', 'Secondary Academic Incomplete', 'Seconda
plt.title('Null value distribution for meaneduc')
plt.show()
```



In [25]:
```
# This shows that the missign values are definitely not for the people who do not have any education.
# Meaning to say, it may not be apt to fill them with zero
# and as it is a mean education column, it would be more meaningful to fill with mean
df_income_train['meaneduc'].fillna(np.mean(df_income_train['meaneduc']),inplace=True) # filling averge for missing values
print('Number of null values for meaneduc:', df_income_train['meaneduc'].isnull().sum())
```

```
Number of null values for meaneduc: 0
```

#5) SQBmeaned, square of the mean years of education of adults (>=18) in the household

In [26]:
```
# As we took mean to fill the missing values for the meaneduc, we can do the same here as this column is a square of the same.
df_income_train['SQBmeaned'].fillna(np.mean(df_income_train['SQBmeaned']),inplace=True) # filling averge for missing values
print('Number of null values for SQBmeaned:', df_income_train['SQBmeaned'].isnull().sum())
```

```
Number of null values for SQBmeaned: 0
```

In [27]:
```
# Now we can see that there are no more null values
df_train_null = pd.DataFrame(df_income_train.isnull().sum()).rename(columns={0:'Null_Total'})
df_train_null['Null_Percent'] = round(100*(df_train_null['Null_Total']/df_income_train.shape[0]),2)
df_train_null.sort_values('Null_Percent',ascending= False).head(6)
```

Out[27]:

	Null_Total	Null_Percent
Id	0	0.0
hogar_mayor	0	0.0
parentesco10	0	0.0
parentesco11	0	0.0
parentesco12	0	0.0
idhogar	0	0.0

b) Conversion of object columns

In [28]:
```
# Lets see how many object columns are there
object_df=df_income_train.select_dtypes(include=['object'])
print(list(object_df.columns))
#Lets analysis each field and take care of them now
```

```
['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa']
```

#1) Id = Unique ID

In [29]:
```
print(object_df.Id.head())
# This has nothing to do with the prediction model. So, we can safely remove it.
# We will add them to a list and drop them together. This will be useful as the same drop list can be used to clean test dataset as
columns_to_drop=['Id']
# means we can remove this column.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['Id'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
0    ID_279628684
1    ID_f29eb3ddd
2    ID_68de51c94
3    ID_d671db89c
4    ID_d56d6f5f5
Name: Id, dtype: object
Shape before dropping columns (9534, 143)
Shape after dropping columns (9534, 142)
```

#2) idhogar = Household level identifier

In [30]:
```
print(object_df.idhogar.head())
# This also has nothing to do with the prediction model. So, we can safely remove it.
columns_to_drop+=['idhogar']
# means we can remove this column.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['idhogar'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
0    21eb7fcc1
1    0e5d7a658
2    2c7317ea8
3    2b58d945f
4    2b58d945f
Name: idhogar, dtype: object
Shape before dropping columns (9534, 142)
Shape after dropping columns (9534, 141)
```

#3) dependency, Dependency rate, calculated = (number of members of the household younger than 19 or older than 64)/(number of member of household between 19 and 64)

In [31]:
```
# Lets the value distirubtion first
object_df.dependency.value_counts().head()
```

Out[31]:
```
yes    2187
no     1746
.5     1495
2       727
1.5     704
Name: dependency, dtype: int64
```

In [32]:
```
# It looks like yes can be converted to a numeric 1 and no to 0
mapping = {'no':0,'yes':1}
# Lets replace them accordingly
df_income_train['dependency']=df_income_train['dependency'].replace(mapping)
# It would be good to convert the column to float as most of the values are float
df_income_train['dependency']=df_income_train['dependency'].astype('float64')
df_income_train['dependency'].value_counts().head()
```

Out[32]:
```
1.0    2187
0.0    1746
0.5    1495
2.0     727
1.5     704
Name: dependency, dtype: int64
```

#4) 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

In [33]:
```
# Lets the value distirubtion first
object_df.edjefe.value_counts()
```

Out[33]:
```
no     3757
6      1838
11      748
9       484
3       305
15      285
8       257
7       234
5       220
14      208
17      202
2       194
4       136
16      133
yes     123
12      113
10      111
13      103
21       43
18       19
19       14
20        7
Name: edjefe, dtype: int64
```

In [34]:
```
# It is evident that yes can be converted to a numeric 1 and no to 0
# Lets replace them accordingly
df_income_train['edjefe']=df_income_train['edjefe'].replace(mapping)
# It would be good to convert the column to int as most of the values are int
df_income_train['edjefe']=df_income_train['edjefe'].astype('int64')
df_income_train['edjefe'].value_counts().head()
```

Out[34]:
```
0     3757
6     1838
11     748
9      484
3      305
Name: edjefe, dtype: int64
```

#5) 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

In [35]:
```
# Lets the value distirubtion first
object_df.edjefa.value_counts()
```

Out[35]:
```
no     6212
6       943
11      399
9       237
8       217
15      188
7       179
5       176
3       151
4       136
14      120
16      113
10       96
2        84
17       76
12       72
```

```
yes      69
13       52
21        5
19        4
18        3
20        2
Name: edjefa, dtype: int64
```

In [36]:
```
# It is evident that yes can be converted to a numeric 1 and no to 0
# Lets replace them accordingly
df_income_train['edjefa']=df_income_train['edjefa'].replace(mapping)
# It would be good to convert the column to int as most of the values are int
df_income_train['edjefa']=df_income_train['edjefa'].astype('int64')
df_income_train['edjefa'].value_counts().head()
```

Out[36]:
```
0     6212
6      943
11     399
9      237
8      217
Name: edjefa, dtype: int64
```

In [37]:
```
# Lets check if there is any more object columns
print('Number of object columns:',len(df_income_train.select_dtypes(include=['object']).columns))
```

```
Number of object columns: 0
```

c) IRemove Redundent Columns

In [38]:
```
# we see a lot of squared columns and they are actually redundent information.
# So, it may be ideal to remove all of them
# SQBescolari= escolari squared
# SQBage, age squared
# SQBhogar_total, hogar_total squared
# SQBedjefe, edjefe squared
# SQBhogar_nin, hogar_nin squared
# SQBovercrowding, overcrowding squared
# SQBdependency, dependency squared
# SQBmeaned, square of the mean years of education of adults (>=18) in the household
# agesq= Age squared
sq_cols=['SQBescolari','SQBage','SQBhogar_total','SQBedjefe','SQBhogar_nin','SQBovercrowding','SQBdependency','SQBmeaned','agesq']
columns_to_drop+=sq_cols
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=sq_cols, inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 141)
Shape after dropping columns (9534, 132)
```

d) Remove zero variance columns

In [39]:
```
# Lets find if there are any zero variance columns in our training dataset
df_income_train.loc[:,df_income_train.var()==0.0].head()
```

Out[39]:

	elimbasu5
0	0
1	0
2	0
3	0
4	0

elimbasu5 : 1 if rubbish disposal mainly by throwing in river, creek or sea.

In [40]:
```
# Means all the values of elimbasu5 column is same and do not have a value in prediction model
columns_to_drop+=['elimbasu5']
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['elimbasu5'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 132)
Shape after dropping columns (9534, 131)
```

e) Further Dimensionality Reduction

In [41]:
```
# Now lets seggregate the columns according to thier background and behaviour

# Target columns
Target_cols = ['Target']

# Individual boolean variables
ind_boolean = ['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']

# Individual Ordered variables
ind_ordered = ['rez_esc', 'escolari', 'age']

# Household boolean
```

```
hh_boolean = ['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', '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']

# Household ordered
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']

# Household continuous
hh_continuous = ['v2a1', 'dependency', 'edjefe', 'edjefa', 'meaneduc', 'overcrowding']
```

In [42]:
```
# Lets check for redundant Individual variables
df_ind = df_income_train[ind_boolean + ind_ordered]           # df with only the individual variable columns
df_ind_corr = df_ind.corr()                                   # correlation matrix for individual df
ind_corr_shape = df_ind_corr.shape                            # Stroign shape to a variable
ind_true_bool = np.ones(ind_corr_shape).astype(bool)          # creating an identity boolean matrix for the ind df shape
ind_upper_tri_bool = np.triu(ind_true_bool, k=1)              # Taking only the upper triangle boolean matrix
df_ind_upper_tri_corr = df_ind_corr.where(ind_upper_tri_bool) # Getting the upper traingle correlation df
# Now lets find if there is any column which has correlation > 0.95 or < -0.95
ind_strong_corr_cols = [col for col in df_ind_upper_tri_corr.columns if any(abs(df_ind_upper_tri_corr[col])>0.95) ]
ind_strong_corr_cols
```

Out[42]:
```
['female']
```

In [43]:
```
# Means that female column has a strong correlation with another variable
# Lets find which one that is.
df_ind_corr[df_ind_corr['female'].abs() > 0.95]['female']
```

Out[43]:
```
male     -1.0
female    1.0
Name: female, dtype: float64
```

male, =1 if male
female, =1 if female

In [44]:
```
# As evident above, the female column is strongly correlated with the male variable. We can choose to keep only one
columns_to_drop+=['female']
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['female'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 131)
Shape after dropping columns (9534, 130)
```

In [45]:
```
# Now Lets check for redundant Household variables
df_hh = df_income_train[hh_boolean + hh_ordered + hh_continuous] # df with only the household variable columns
df_hh_corr = df_hh.corr()                                       # correlation matrix for individual df
hh_corr_shape = df_hh_corr.shape                                # Storing shape to a variable
hh_true_bool = np.ones(hh_corr_shape).astype(bool)              # creating an identity boolean matrix for the ind df shape
hh_upper_tri_bool = np.triu(hh_true_bool, k=1)                  # Taking only the upper triangle boolean matrix
df_hh_upper_tri_corr = df_hh_corr.where(hh_upper_tri_bool)      # Getting the upper traingle correlation df
# Now lets find if there is any column which has correlation > 0.95 or < -0.95
hh_strong_corr_cols = [col for col in df_hh_upper_tri_corr.columns if any(abs(df_hh_upper_tri_corr[col])>0.95) ]
hh_strong_corr_cols
```

Out[45]:
```
['coopele', 'area2', 'tamhog', 'hhsize', 'hogar_total']
```

In [46]:
```
# Lets see coopele first
# Lets find which is the other column that is highly correlated with coopele
df_hh_corr[df_hh_corr['coopele'].abs() > 0.95]['coopele']
```

Out[46]:
```
public    -0.979769
coopele    1.000000
Name: coopele, dtype: float64
```

coopele, =1 electricity from cooperative
public, =1 electricity from CNFL, ICE, ESPH/JASEC

In [47]:
```
# As evident from above, the coopele column is strongly correlated with the public variable. We can choose to keep only one
columns_to_drop+=['public']
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['public'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 130)
Shape after dropping columns (9534, 129)
```

In [48]:
```
# Lets see area2
# Lets find which is the other column that is highly correlated with area2
df_hh_corr[df_hh_corr['area2'].abs() > 0.95]['area2']
```

Out[48]:
```
area1   -1.0
area2    1.0
Name: area2, dtype: float64
```

area1, =1 zona urbana
area2, =2 zona rural

In [49]:
```
# As evident from above, the area2 column is strongly correlated with the area1 variable. We can choose to keep only one
columns_to_drop+=['area2']
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['area2'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 129)
Shape after dropping columns (9534, 128)
```

In [50]:
```
# Lets see tamhog
# Lets find which is the other column that is highly correlated with tamhog
df_hh_corr[df_hh_corr['tamhog'].abs() > 0.95]['tamhog']
```

Out[50]:
```
r4t3           0.998106
tamhog         1.000000
hhsize         1.000000
hogar_total    1.000000
Name: tamhog, dtype: float64
```

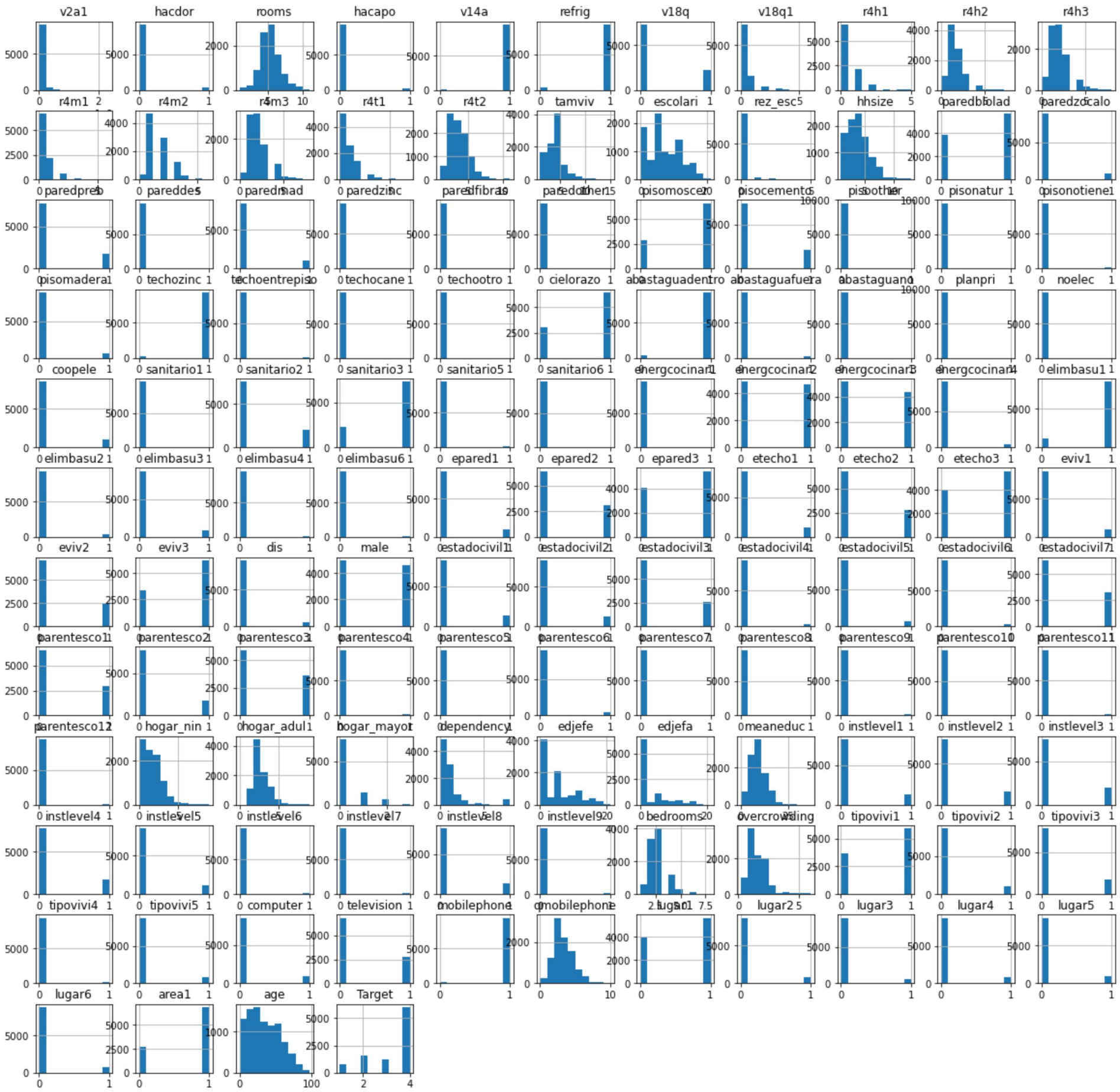
r4t3, Total persons in the household
tamhog, size of the household
hhsize, household size
hogar_total, # of total individuals in the household

In [51]:
```
# As evident from above, all the 4 variables talk about the size of the house hold. We can keep only one
columns_to_drop+=['r4t3','tamhog','hogar_total']
# means we can remove this column as well.
print('Shape before dropping columns',df_income_train.shape)
df_income_train.drop(columns=['r4t3','tamhog','hogar_total'], inplace=True)
print('Shape after dropping columns',df_income_train.shape)
```

```
Shape before dropping columns (9534, 128)
Shape after dropping columns (9534, 125)
```

f) Remove outliers

In [52]:
```
# 1st lets how the data distribution in our dataset.
df_income_train.hist(figsize=(20,20))
plt.show()
```



In [53]:
```
# It is evident that majority of the columns do not have a normal distirbution.
# So, we can't use the standard deviation method to identify and remove outliers.
# Then, the best method is to use Interquartile Range (IQR) mehtod
# We can use IQR to identify outliers in an input feature that does not follow a normal or normal-like distribution.
# Values that are outside a specified threshold
# (usually, 1.5 times IQR above the 75th percentile and below the 25th percentile) are filtered out for further analysis.
```

In [54]:
```
# Calculate 1st and 3rd percentiles, and IQR
Q1 = df_income_train.quantile(0.25)
Q3 = df_income_train.quantile(0.75)
IQR = Q3 - Q1
# Filter out the rows that fall outside the 1.5 threshold in each column
df_income_train[~((df_income_train < (Q1 - 1.5 * IQR)) | (df_income_train > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Out[54]:

v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...	qmobilephone	lugar1	lugar2	lugar3	lugar4	lugar5	lugar6	area1	age	T

0 rows × 125 columns

Good news is that we do not have any outliers which need to be removed.

g) Split datasets to source(X) & target(y)

In [55]:
```
# Source is with all columns except our target poverty levels
X_data = df_income_train.drop(columns=['Target'])
print('Number of columns in X:',len(X_data.columns))
y_data = df_income_train[['Target']]
print('Number of columns in y:',len(y_data.columns))
```

```
Number of columns in X: 124
Number of columns in y: 1
```

Step 2. Model Building

a) Applying Standard Scalling

In [56]:
```
# Its always good to do a scaling or normalization on the data for better results.
```

```
# Lets use sklearn standard scaler for the purpose
SS=StandardScaler()
# Scaling source dataset
X=SS.fit_transform(X_data)
X=pd.DataFrame(X,columns=X_data.columns)
print(X.head())
# As it is classification levels, no need to scale the target dataset
y=y_data
print(y.head())
```

```
       v2a1   hacdor     rooms    hacapo      v14a    refrig      v18q  \
0  1.311819 -0.19838 -1.333317 -0.155821  0.070386  0.210356 -0.550125
1  0.808428 -0.19838 -0.652301 -0.155821  0.070386  0.210356  1.817770
2 -0.427170 -0.19838  2.071762 -0.155821  0.070386  0.210356 -0.550125
3  1.220293 -0.19838  0.028715 -0.155821  0.070386  0.210356  1.817770
4  1.220293 -0.19838  0.028715 -0.155821  0.070386  0.210356  1.817770

      v18q1     r4h1      r4h2  ...  mobilephone  qmobilephone    lugar1  \
0 -0.467512 -0.56578 -0.538696  ...     0.159317     -1.226974  0.835978
1  0.965691 -0.56578 -0.538696  ...     0.159317     -1.226974  0.835978
2 -0.467512 -0.56578 -1.503009  ...    -6.276807     -1.900703  0.835978
3  0.965691 -0.56578  0.425617  ...     0.159317      0.120486  0.835978
4  0.965691 -0.56578  0.425617  ...     0.159317      0.120486  0.835978

     lugar2    lugar3    lugar4    lugar5    lugar6     area1       age
0 -0.319682 -0.257997 -0.298708 -0.321472 -0.296201  0.629369  0.400397
1 -0.319682 -0.257997 -0.298708 -0.321472 -0.296201  0.629369  1.510706
2 -0.319682 -0.257997 -0.298708 -0.321472 -0.296201  0.629369  2.667278
3 -0.319682 -0.257997 -0.298708 -0.321472 -0.296201  0.629369 -0.802438
4 -0.319682 -0.257997 -0.298708 -0.321472 -0.296201  0.629369  0.122819

[5 rows x 124 columns]
   Target
0       4
1       4
2       4
3       4
4       4
```

b) Splitting dataset for train & test

In [57]:
```
# Lets spli the dataset with an 80:20 spllit and stratify use to keep the target proportions.
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,stratify=y,random_state=0)
print('X_train shape:',X_train.shape)
print('y_train shape:',y_train.shape)
print('X_test shape:',X_test.shape)
print('y_test shape:',y_test.shape)
```

```
X_train shape: (7150, 124)
y_train shape: (7150, 1)
X_test shape: (2384, 124)
y_test shape: (2384, 1)
```

c) Hyper parameter tuning for Random Forest Classifier

In [58]:
```
# Lets concsider the below hyper parameters for tuning as they are best for the bucks
# n_estimators = number of trees in the foreset
# max_features = max number of features considered for splitting a node
# max_depth = max number of levels in each decision tree
# min_samples_split = min number of data points placed in a node before the node is split
# min_samples_leaf = min number of data points allowed in a leaf node
# bootstrap = method for sampling data points (with or without replacement)
```

RandomizedSearchCV

In [59]:
```
# To use RandomizedSearchCV, we first need to create a parameter grid to sample from, during fitting

# Number of trees in random forest
n_estimators = list(range(200,2001,200))   # 10 values between 200 & 2001 with 200 step
print('n_estimators:',n_estimators)
# Number of features to consider at every split
max_features = ['auto','log2', 'sqrt']
print('max_features:',max_features)
# Maximum number of levels in tree
max_depth = list(range(10,111,10))+[None]  # 10 values between 10 & 111 with 10 step. Also added 'None'
print('max_depth:',max_depth)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
print('min_samples_split:',min_samples_split)
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
print('min_samples_leaf:',min_samples_leaf)
# Method of selecting samples for training each tree
bootstrap = [True, False]
print('bootstrap:',bootstrap)
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

```
n_estimators: [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
max_features: ['auto', 'log2', 'sqrt']
max_depth: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None]
min_samples_split: [2, 5, 10]
min_samples_leaf: [1, 2, 4]
bootstrap: [True, False]
```

In [60]:
```
# Now lets use the random grid to search for best hyperparameters
```

```
# First create the base model to tune
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation, search across 100 different combinations
# n_iter : controls the number of different combinations to try
# cv : the number of folds to use for cross validation
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=1, random_state=9, r
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

Out[60]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'log2',
                                                         'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=9, verbose=1)
```

In [61]:
```
print('Best Score:', round(rf_random.best_score_*100,2),'%')
print('Best Params:\n', rf_random.best_params_)
```

```
Best Score: 91.89 %
Best Params:
 {'n_estimators': 1600, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 50, 'bootstrap': False}
```

GridSearchCV

In [62]:
```
# Random search allowed us to narrow down the range for each hyperparameter.
# Now that we know where to concentrate our search, we can explicitly specify every combination of settings to try.
```

In [63]:
```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [40,45,50,55,60],
    'max_features': ['auto'],
    'min_samples_leaf': [1, 2, 3],
    'min_samples_split': [1, 2, 3],
    'n_estimators': [1500,1550,1600,1650,1700]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
rf_grid = GridSearchCV(estimator = rf, param_grid = param_grid, cv = 3, n_jobs = -1, verbose = 1)
# Fit the grid search to the data
rf_grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 225 candidates, totalling 675 fits
```

Out[63]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [False],
                         'max_depth': [40, 45, 50, 55, 60],
                         'max_features': ['auto'],
                         'min_samples_leaf': [1, 2, 3],
                         'min_samples_split': [1, 2, 3],
                         'n_estimators': [1500, 1550, 1600, 1650, 1700]},
             verbose=1)
```

In [64]:
```
print('Best Score:', round(rf_grid.best_score_*100,2),'%')
print('Best Params:\n', rf_grid.best_params_)
```

```
Best Score: 91.93 %
Best Params:
 {'bootstrap': False, 'max_depth': 55, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 1600}
```

d) Fitting the model with Random Forest Classifier

In [65]:
```
# USing the best one we got
rfc=rf_grid.best_estimator_
#Fititng the model
rfc_model=rfc.fit(X_train,y_train)
```

Step 3. Prediction and Metrics evaluation

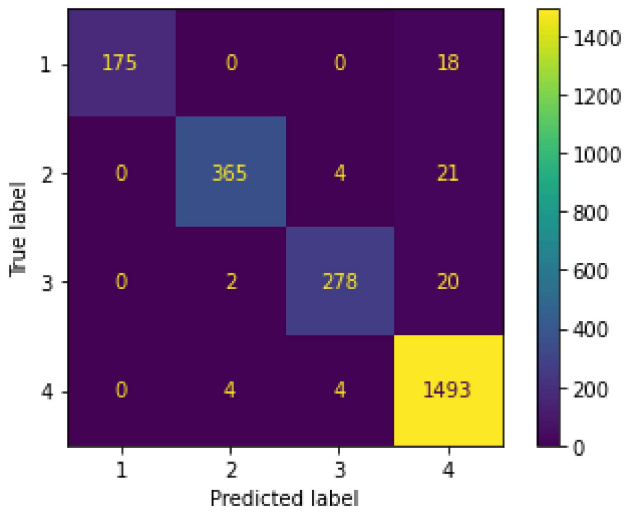
In [66]:
```
#predicting with the model
y_pred=rfc_model.predict(X_test)
```

In [67]:
```
print('Model Score of train data : {} %'.format(round(rfc_model.score(X_train,y_train)*100,2)))
print('Model Score of test data : {} %'.format(round(rfc_model.score(X_test,y_test)*100,2)))
```

```
Model Score of train data : 100.0 %
Model Score of test data : 96.94 %
```

Accuracy of the model for test data : 96.94 %

In [68]:
```
# plotting the confsion matrix
plot_confusion_matrix(rfc_model,X_test, y_test)
plt.show()
```



In [69]:
```
#Printing the classification report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       1.00      0.91      0.95       193
           2       0.98      0.94      0.96       390
           3       0.97      0.93      0.95       300
           4       0.96      0.99      0.98      1501

    accuracy                           0.97      2384
   macro avg       0.98      0.94      0.96      2384
weighted avg       0.97      0.97      0.97      2384
```

Predicting for the test dataset

In [70]:
```
# Lets remove unwanted columsn from the test dataset similar to what we did for train dataset
print('Columns to drop:',columns_to_drop)
print('Shape before dropping columns',df_income_test.shape)
df_income_test.drop(columns=columns_to_drop, inplace=True)
print('Shape after dropping columns',df_income_test.shape)
```

```
Columns to drop: ['Id', 'idhogar', 'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe', 'SQBhogar_nin', 'SQBovercrowding', 'SQBde
pendency', 'SQBmeaned', 'agesq', 'elimbasu5', 'female', 'public', 'area2', 'r4t3', 'tamhog', 'hogar_total']
Shape before dropping columns (23856, 142)
Shape after dropping columns (23856, 124)
```

In [71]:
```
# Lets take care of the null value columns for the test dataset similar to what we did for train dataset
print('Columns with null values:',df_income_test.columns[df_income_test.isnull().any()].to_list())
df_income_test['v2a1'].fillna(0,inplace=True) # filling zero for missing values
df_income_test['v18q1'].fillna(0,inplace=True) # filling zero for missing values
df_income_test['rez_esc'].fillna(0,inplace=True) # filling zero for missing values
df_income_test['meaneduc'].fillna(np.mean(df_income_test['meaneduc']),inplace=True) # filling averge for missing values
print('Columns with null values:',df_income_test.columns[df_income_test.isnull().any()].to_list())
```

```
Columns with null values: ['v2a1', 'v18q1', 'rez_esc', 'meaneduc']
Columns with null values: []
```

In [72]:
```
# Lets see how many object columns are there, if any lets concert them to numeric
print('Non-numeric columns:',list(df_income_test.select_dtypes(include=['object']).columns))
mapping = {'no':0,'yes':1}
df_income_test['dependency']=df_income_test['dependency'].replace(mapping)  # replacing yes & no
df_income_test['dependency']=df_income_test['dependency'].astype('float64') # converting to numeric
df_income_test['edjefe']=df_income_test['edjefe'].replace(mapping)          # replacing yes & no
df_income_test['edjefe']=df_income_test['edjefe'].astype('int64')           # converting to numeric
df_income_test['edjefa']=df_income_test['edjefa'].replace(mapping)          # replacing yes & no
df_income_test['edjefa']=df_income_test['edjefa'].astype('int64')           # converting to numeric
print('Non-numeric columns:',list(df_income_test.select_dtypes(include=['object']).columns))
```

```
Non-numeric columns: ['dependency', 'edjefe', 'edjefa']
Non-numeric columns: []
```

In [73]:
```
# Lets use scaling similar to what we did for training dataset
SS=StandardScaler()
test_X=SS.fit_transform(df_income_test) # Scaling source dataset
test_X=pd.DataFrame(test_X,columns=df_income_test.columns) # preparing dataframe
```

In [74]:
```
#predicting with the model
test_y_pred=rfc_model.predict(test_X)
```

In [75]:
```
print("Predictions for test dataset",test_y_pred)
```

```
Predictions for test dataset [4 4 4 ... 4 4 4]
```

In [76]:
```
# Frequency distribution
itemfreq(test_y_pred)
```

Out[76]:
```
array([[    1,   718],
       [    2,  3062],
       [    3,   511],
       [    4, 19565]], dtype=int64)
```

Check the accuracy using random forest with cross validation.

In [77]:
```
# Lets do a 10 fold cross validation
kfold=KFold(n_splits=10,random_state=9,shuffle=True)
accuracy_values=cross_val_score(rfc_model, X , y, cv=kfold, scoring='accuracy')
print('Accuracy values for 10-fold Cross Validation:\n',accuracy_values)
print('Final Average Accuracy of the model:', round(accuracy_values.mean()*100,2),'%')
```

```
Accuracy values for 10-fold Cross Validation:
 [0.96540881 0.9769392  0.96855346 0.9769392  0.97691501 0.96327387
```

```
 0.97376705 0.98426023 0.98111228 0.97271773]
Final Average Accuracy of the model: 97.4 %
```

Average Accuracy of the model using K Fold: 97.4 %

Check the feature importance of our model

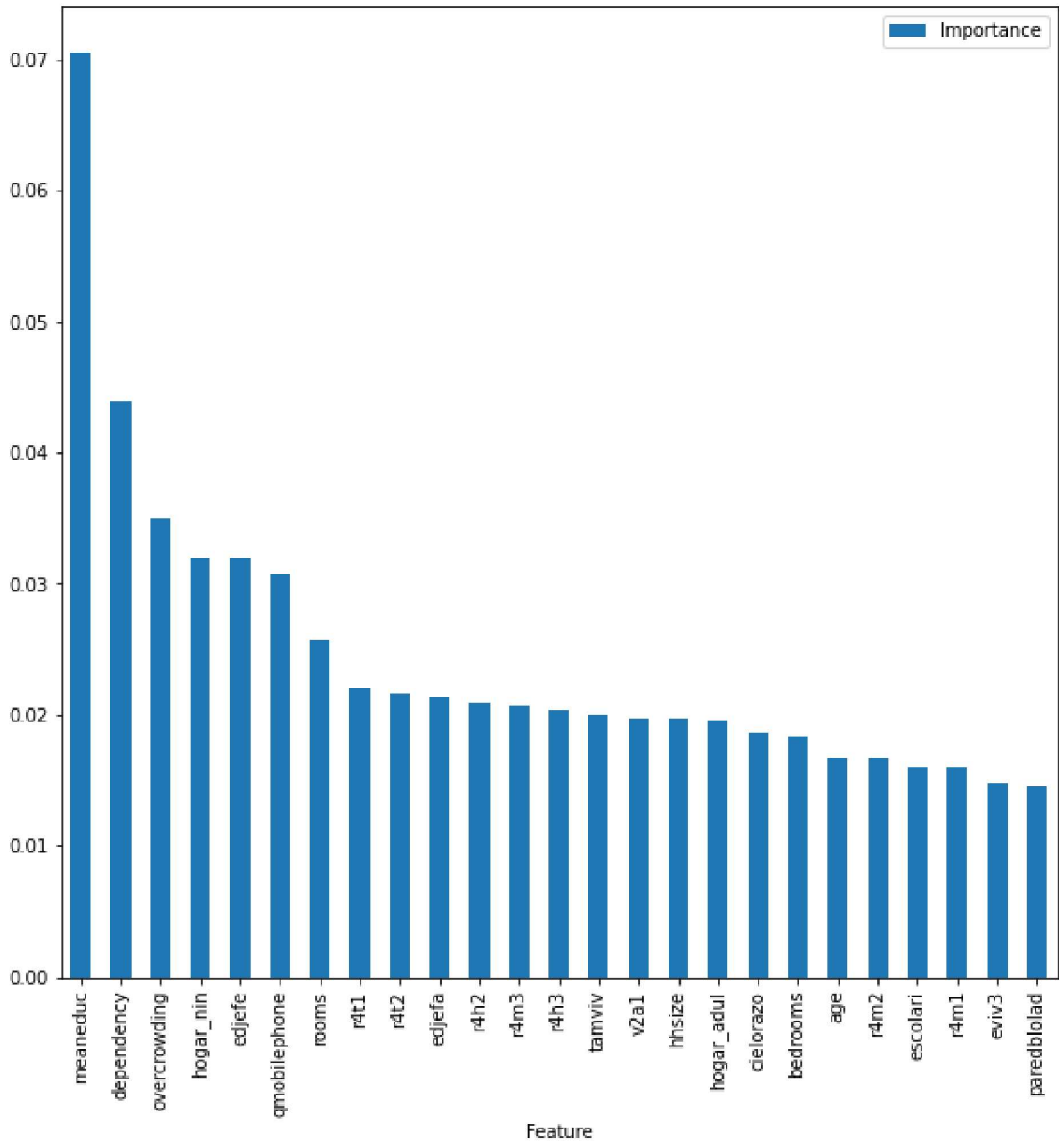
In [78]:
```
# To get a better sense of what is going on inside the RandomForestClassifier model,
# lets visualize how our model uses the different features and which features have greater effect.
df_feature_importance=pd.DataFrame({'Feature': X.columns, 'Importance': rfc_model.feature_importances_})
df_top25_features=df_feature_importance.sort_values(by=['Importance'],ascending=False).head(25).set_index('Feature')
df_top25_features.head()
```

Out[78]:

Feature	Importance
meaneduc	0.070539
dependency	0.043987
overcrowding	0.034977
hogar_nin	0.032004
edjefe	0.031998

In [79]:
```
df_top25_features.plot(kind='bar',figsize=(10,10))
```

Out[79]:
```
<AxesSubplot:xlabel='Feature'>
```



In [80]:
```
# From this it is evident that meaneduc,dependency,overcrowding has significant influence on the model.
```