# DA- IU3.6.9_STANDARD PROJECT Machine Learning- Group Number 4 - Jonathan Jie

## DESCRIPTION

#### Identify the level of income qualification needed for the families in Latin America.

### Problem Statement Scenario:

#### Many social programs have a hard time ensuring that the right people are given enough aid. It’s tricky when a program focuses on the poorest segment of the population. This segment of the population can’t provide the necessary income and expense records to prove that they qualify.

#### In Latin America, a popular method called Proxy Means Test (PMT) uses an algorithm to verify income qualification. With PMT, agencies use a model that considers a family’s observable household attributes like the material of their walls and ceiling or the assets found in their homes to

#### classify them and predict their level of need.

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

#### The Inter-American Development Bank (IDB)believes that new methods beyond traditional econometrics, based on a dataset of Costa Rican household characteristics, might help improve PMT’s performance.

### Following actions should be performed:

#### 1. Identify the output variable.

#### 2. Understand the type of data.

#### 3. Check if there are any biases in your dataset.

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

#### 5. Check if there is a house without a family head.

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

#### 7. Count how many null values are existing in columns.

#### 8. Remove null value rows of the target variable.

#### 9. Predict the accuracy using random forest classifier.

#### 10. Check the accuracy using random forest with cross validation.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sp
```

In [2]:
```
# Importing our test and train data

train = pd.read_csv('C:\\Users\\Jonathan Jie\\Machine Learning\\ML Project\\train.csv')
```

In [3]:
```
test = pd.read_csv('C:\\Users\\Jonathan Jie\\Machine Learning\\ML Project\\test.csv')
```

**Data Set Exploration**

In [4]:
```
train.head()
```

Out[4]:

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

5 rows × 143 columns

In [5]:
```
train.describe()
```

Out[5]:

| | v2a1 | hacdor | rooms | hacapo | v14a | refrig | v18q | v18q1 | r4h1 | r4h2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2.697000e+03 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 2215.000000 | 9557.000000 | 9557.000000 |
| **mean** | 1.652316e+05 | 0.038087 | 4.955530 | 0.023648 | 0.994768 | 0.957623 | 0.231767 | 1.404063 | 0.385895 | 1.559171 |
| **std** | 1.504571e+05 | 0.191417 | 1.468381 | 0.151957 | 0.072145 | 0.201459 | 0.421983 | 0.763131 | 0.680779 | 1.036574 |
| **min** | 0.000000e+00 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 |
| **25%** | 8.000000e+04 | 0.000000 | 4.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 |
| **50%** | 1.300000e+05 | 0.000000 | 5.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 |
| **75%** | 2.000000e+05 | 0.000000 | 6.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 | 2.000000 |
| **max** | 2.353477e+06 | 1.000000 | 11.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 6.000000 | 5.000000 | 8.000000 |

8 rows × 138 columns

In [6]:
```
train.info
```

Out[6]:
```
<bound method DataFrame.info of                   Id      v2a1  hacdor  rooms  hacapo  v14a  refrig  v18q  \
0     ID_279628684  190000.0       0      3       0     1       1     0   
1     ID_f29eb3ddd  135000.0       0      4       0     1       1     1   
2     ID_68de51c94       NaN       0      8       0     1       1     0   
3     ID_d671db89c  180000.0       0      5       0     1       1     1   
4     ID_d56d6f5f5  180000.0       0      5       0     1       1     1   
...            ...       ...     ...    ...     ...   ...     ...   ...   
9552  ID_d45ae367d   80000.0       0      6       0     1       1     0   
9553  ID_c94744e07   80000.0       0      6       0     1       1     0   
9554  ID_85fc658f8   80000.0       0      6       0     1       1     0   
9555  ID_ced540c61   80000.0       0      6       0     1       1     0   
9556  ID_a38c64491   80000.0       0      6       0     1       1     0   

      v18q1  r4h1  ...  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  \
0       NaN     0  ...          100    1849               1        100   
1       1.0     0  ...          144    4489               1        144   
2       NaN     0  ...          121    8464               1          0   
3       1.0     0  ...           81     289              16        121   
4       1.0     0  ...          121    1369              16        121   
...     ...   ...  ...          ...     ...             ...        ...   
9552    NaN     0  ...           81    2116              25         81   
9553    NaN     0  ...            0       4              25         81   
9554    NaN     0  ...           25    2500              25         81   
9555    NaN     0  ...          121     676              25         81   
9556    NaN     0  ...           64     441              25         81   

      SQBhogar_nin  SQBovercrowding  SQBdependency  SQBmeaned  agesq  Target  
0                0         1.000000         0.0000   100.0000   1849       4  
1                0         1.000000        64.0000   144.0000   4489       4  
2                0         0.250000        64.0000   121.0000   8464       4  
3                4         1.777778         1.0000   121.0000    289       4  
4                4         1.777778         1.0000   121.0000   1369       4  
...            ...              ...            ...        ...    ...     ...  
9552             1         1.562500         0.0625    68.0625   2116       2  
9553             1         1.562500         0.0625    68.0625      4       2  
9554             1         1.562500         0.0625    68.0625   2500       2  
9555             1         1.562500         0.0625    68.0625    676       2  
9556             1         1.562500         0.0625    68.0625    441       2  

[9557 rows x 143 columns]>
```

### Qn 2 Understand the type of data

In [7]:
```
train.info(verbose=True, show_counts=True)
```

```
 47   noelec          9557 non-null   int64
 48   coopele         9557 non-null   int64
 49   sanitario1      9557 non-null   int64
 50   sanitario2      9557 non-null   int64
 51   sanitario3      9557 non-null   int64
 52   sanitario5      9557 non-null   int64
 53   sanitario6      9557 non-null   int64
 54   energcocinar1   9557 non-null   int64
 55   energcocinar2   9557 non-null   int64
 56   energcocinar3   9557 non-null   int64
 57   energcocinar4   9557 non-null   int64
 58   elimbasu1       9557 non-null   int64
 59   elimbasu2       9557 non-null   int64
 60   elimbasu3       9557 non-null   int64
 61   elimbasu4       9557 non-null   int64
 62   elimbasu5       9557 non-null   int64
 63   elimbasu6       9557 non-null   int64
 64   epared1         9557 non-null   int64
 65   epared2         9557 non-null   int64
```

In [8]:
```
train.nunique()
```

Out[8]:
```
Id                  9557
v2a1                 157
hacdor                 2
rooms                 11
hacapo                 2
                    ... 
SQBovercrowding       38
SQBdependency         31
SQBmeaned            155
agesq                 97
Target                 4
Length: 143, dtype: int64
```

In [9]:
```
print(train.dtypes.value_counts())
```

```
int64      130
float64      8
object       5
Name: count, dtype: int64
```

### Qn 7 Count how many null values are existing in columns

In [10]:
```
train.isnull().sum()
```

Out[10]:
```
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 [11]:
```
# % of Nulls

(train.isnull().sum()/(len(train)))*100
```

Out[11]:
```
Id                  0.000000
v2a1               71.779847
hacdor              0.000000
rooms               0.000000
hacapo              0.000000
                     ...    
SQBovercrowding     0.000000
SQBdependency       0.000000
SQBmeaned           0.052318
agesq               0.000000
Target              0.000000
Length: 143, dtype: float64
```

In [12]:
```
train.describe()
```

Out[12]:

| | v2a1 | hacdor | rooms | hacapo | v14a | refrig | v18q | v18q1 | r4h1 | r4h2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 2.697000e+03 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 9557.000000 | 2215.000000 | 9557.000000 | 9557.000000 |
| **mean** | 1.652316e+05 | 0.038087 | 4.955530 | 0.023648 | 0.994768 | 0.957623 | 0.231767 | 1.404063 | 0.385895 | 1.559171 |
| **std** | 1.504571e+05 | 0.191417 | 1.468381 | 0.151957 | 0.072145 | 0.201459 | 0.421983 | 0.763131 | 0.680779 | 1.036574 |
| **min** | 0.000000e+00 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 |
| **25%** | 8.000000e+04 | 0.000000 | 4.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 |
| **50%** | 1.300000e+05 | 0.000000 | 5.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 1.000000 |
| **75%** | 2.000000e+05 | 0.000000 | 6.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 2.000000 | 1.000000 | 2.000000 |
| **max** | 2.353477e+06 | 1.000000 | 11.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 6.000000 | 5.000000 | 8.000000 |

8 rows × 138 columns

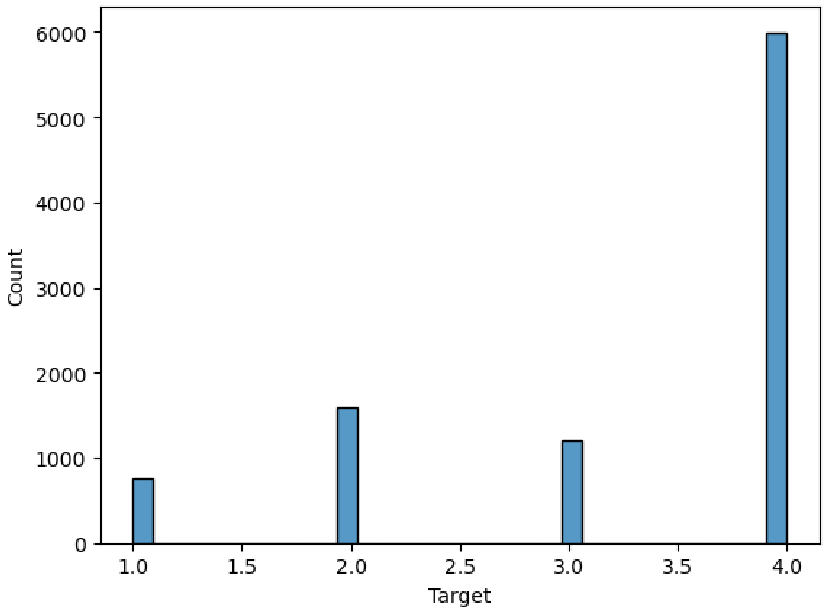
### Qn 1 Identify the output variable and Qn 3 check for Bias

In [13]:
```
for i in train.columns:
    if i not in test.columns:
        print("Our Target variable is {}".format(i))
```

```
Our Target variable is Target
```

In [14]:
```
sns.histplot(train['Target'])
```

Out[14]: <Axes: xlabel='Target', ylabel='Count'>




In [15]:
```
train['Target'].value_counts()
```

Out[15]:
```
Target
4    5996
2    1597
3    1209
1     755
Name: count, dtype: int64
```

In [16]:
```
train['Target'].isnull().sum()
```

Out[16]: 0

### Qn 8 Remove null value rows of the target variable - Target Variable has no Null Values

In [17]:
```
train.describe(include = "O").T
```

Out[17]:

| | count | unique | top | freq |
|---|---|---|---|---|
| **Id** | 9557 | 9557 | ID_279628684 | 1 |
| **idhogar** | 9557 | 2988 | fd8a6d014 | 13 |
| **dependency** | 9557 | 31 | yes | 2192 |
| **edjefe** | 9557 | 22 | no | 3762 |
| **edjefa** | 9557 | 22 | no | 6230 |

In [18]:
```
train.describe(percentiles=np.linspace(0,1,11)).T
```

Out[18]:

| | count | mean | std | min | 0% | 10% | 20% | 30% | 40% | 50% |
|---|---|---|---|---|---|---|---|---|---|---|
| **v2a1** | 2697.0 | 165231.606971 | 150457.133301 | 0.00 | 0.00 | 40000.000000 | 70000.0000 | 90000.000000 | 102000.000000 | 130000.000000 |
| **hacdor** | 9557.0 | 0.038087 | 0.191417 | 0.00 | 0.00 | 0.000000 | 0.0000 | 0.000000 | 0.000000 | 0.000000 |
| **rooms** | 9557.0 | 4.955530 | 1.468381 | 1.00 | 1.00 | 3.000000 | 4.0000 | 4.000000 | 5.000000 | 5.000000 |
| **hacapo** | 9557.0 | 0.023648 | 0.151957 | 0.00 | 0.00 | 0.000000 | 0.0000 | 0.000000 | 0.000000 | 0.000000 |
| **v14a** | 9557.0 | 0.994768 | 0.072145 | 0.00 | 0.00 | 1.000000 | 1.0000 | 1.000000 | 1.000000 | 1.000000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **SQBovercrowding** | 9557.0 | 3.249485 | 4.129547 | 0.04 | 0.04 | 0.694444 | 1.0000 | 1.000000 | 1.777778 | 2.250000 |
| **SQBdependency** | 9557.0 | 3.900409 | 12.511831 | 0.00 | 0.00 | 0.000000 | 0.0625 | 0.250000 | 0.250000 | 0.444444 |
| **SQBmeaned** | 9552.0 | 102.588867 | 93.516890 | 0.00 | 0.00 | 18.777779 | 36.0000 | 44.444443 | 64.000000 | 81.000000 |
| **agesq** | 9557.0 | 1643.774302 | 1741.197050 | 0.00 | 0.00 | 49.000000 | 196.0000 | 361.000000 | 625.000000 | 961.000000 |
| **Target** | 9557.0 | 3.302292 | 1.009565 | 1.00 | 1.00 | 2.000000 | 2.0000 | 3.000000 | 4.000000 | 4.000000 |

138 rows × 16 columns

#### Taking a closer look at columns that need replacement

In [19]:
```
train['dependency'].describe()
```

Out[19]:
```
count      9557
unique       31
top         yes
freq       2192
Name: dependency, dtype: object
```

In [20]:
```
unique_values = train['dependency'].unique()
print(unique_values)
```

```
['no' '8' 'yes' '3' '0.5' '0.25' '2' '0.66666669' '0.33333334' '1.5'
 '0.40000001' '0.75' '1.25' '0.2' '2.5' '1.2' '4' '1.3333334' '2.25'
 '0.22222222' '5' '0.83333331' '0.80000001' '6' '3.5' '1.6666666'
 '0.2857143' '1.75' '0.71428573' '0.16666667' '0.60000002']
```

In [21]:
```
train['edjefe'].describe()
```

Out[21]:
```
count      9557
unique       22
top          no
freq       3762
Name: edjefe, dtype: object
```

In [22]:
```
unique_values = train['edjefe'].unique()
print(unique_values)
```

```
['10' '12' 'no' '11' '9' '15' '4' '6' '8' '17' '7' '16' '14' '5' '21' '2'
 '19' 'yes' '3' '18' '13' '20']
```

In [23]:
```
train['edjefa'].describe()
```

Out[23]:
```
count      9557
unique       22
top          no
freq       6230
Name: edjefa, dtype: object
```

In [24]:
```
unique_values = train['edjefa'].unique()
print(unique_values)
```

```
['no' '11' '4' '10' '9' '15' '7' '14' '13' '8' '17' '6' '5' '3' '16' '19'
 'yes' '21' '12' '2' '20' '18']
```

### Qn 4 Check whether all members of the house have the same poverty level.

In [25]:
```
#Check if
household_not_same_target = train.groupby(['idhogar'])['Target'].nunique()
household_not_same_target = household_not_same_target[household_not_same_target != 1].sort_values(ascending=False
```

In [26]:
```
#set pandas to display 100 columns and rows
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', 100)
pd.set_option('display.width', 100)
```

In [27]:
```
household_not_same_target
```

```
Out[27]: idhogar
5c6f32bbc    3
0172ab1d9    2
7e9d58c5c    2
932287f5d    2
8bb6da3c1    2
8ae3e74ca    2
8420bcfca    2
8242a51ec    2
811a35744    2
80a66379b    2
7ea6aca15    2
7c57f8237    2
a20ff33ba    2
7ad269eef    2
73d85d05d    2
71cd52a80    2
6c543442a    2
6bcf799cf    2
6a389f3de    2
694a0cbf4    2
9bbf7c6ca    2
a3288e6fa    2
67ad49822    2
d9b1558b5    2
f7b421c2c    2
f006348ed    2
efd3aec61    2
e65d4b943    2
e235a4eec    2
e17b252ed    2
dfb966eec    2
daafc1281    2
d64524b6b    2
a94a45642    2
cc971b690    2
c7ce4e30c    2
c38913488    2
c13325faf    2
be91da044    2
bd82509d1    2
bcab69521    2
bcaa2e2f5    2
6833ac5dc    2
654ef7612    2
03f4e5f4d    2
3fe29a56b    2
3c73c107f    2
3c6973219    2
30a70901d    2
309fb7246    2
2cb443214    2
2c9872b82    2
28893b5e7    2
288579c97    2
26b3a0f41    2
18832b840    2
17fb04a62    2
15a891635    2
0f9494d3a    2
0f3e65c83    2
09e25d616    2
078a0b6e2    2
0511912b6    2
3df651058    2
410194c8b    2
636330516    2
417865404    2
614b48fb7    2
5e9329fc6    2
5c3f7725d    2
594d3eb27    2
564eab113    2
55a662731    2
54118d5d9    2
513adb616    2
50e064ee8    2
4e19bd549    2
4dc11e11f    2
4c2dba109    2
4b6077882    2
46af47063    2
44f219a16    2
43b9c83e5    2
```

```
42ec8bef5    2
f94589d38    2
Name: Target, dtype: int64
```

In [28]:
```
total_unique_idhogar_count = household_not_same_target.count()

count = total_unique_idhogar_count
message = "Number of households with different poverty count is {}".format(count)
print(message)
```

```
Number of households with different poverty count is 85
```

### Qn 5 Check if there is a house without a family head.

In [29]:
```
#Check if anyone in the household is household head 0 = No

no_household_head = train.groupby(['idhogar'])['parentesco1'].max()
```

In [30]:
```
no_household_head[no_household_head != 1].sort_values(ascending = True)
```

Out[30]:
```
idhogar
03c6bdf85    0
09b195e7a    0
1367ab31d    0
1bc617b23    0
374ca5a19    0
61c10e099    0
6b1b2405f    0
896fe6d3e    0
a0812ef17    0
ad687ad89    0
b1f4d89d7    0
bfd5067c2    0
c0c8a5013    0
d363d9183    0
f2bfa75c4    0
Name: parentesco1, dtype: int64
```

In [31]:
```
no_head_count = no_household_head[no_household_head != 1].count()

message = f"Number of households with no household head is {no_head_count}"
print(message)
```

```
Number of households with no household head is 15
```

## Doing data cleaning and NA filling

In [32]:
```
#Replacing NA in v2a1(monthly rent payment) with 0 as crossed referenced with tipovivi1-5

train['v2a1'] = train['v2a1'].fillna(0)
```

In [33]:
```
#Replacing NA in v18q1(number of tablets household owns) with 0 as crossed referenced with v18q

train['v18q1'] = train['v18q1'].fillna(0)
```

In [34]:
```
#Replacing 5 NA in meaneduc(average years of education for adults (18+) same NA as SQBmeaned below

train['meaneduc'] = train['meaneduc'].fillna(0)
```

In [35]:
```
#Replacing 5 NA in sqbmeaned(square of the mean years of education of adults (>=18)) same NA as meaneduc above

train['SQBmeaned'] = train['SQBmeaned'].fillna(0)
```

In [36]:
```
#Dropping rez_esc - Years behind in school from dataframe due to large amount of null values 7928 out of 9557 rows

train = train.drop('rez_esc', axis = 1)
```

In [37]:
```
#Replacing No string values in dependency to 0

train['dependency'] = train['dependency'].replace('no', 0 )
```

In [38]:
```
#Finding the median dependency value to replace the yes

pd.to_numeric(train['dependency'],errors='coerce').median()
```

Out[38]:
```
0.5
```

In [39]:
```
#Replacing Yes string values in dependency to the median of 0.5

train['dependency'] = train['dependency'].replace('yes', 0.5 )
```

In [40]:
```
train['dependency']
```

Out[40]:
```
0          0
1          8
2          8
3        0.5
4        0.5
        ...
9552    0.25
9553    0.25
9554    0.25
9555    0.25
9556    0.25
Name: dependency, Length: 9557, dtype: object
```

In [41]:
```
#Replacing No string values in edjefe to 0

train['edjefe'] = train['edjefe'].replace('no', 0 )
```

In [42]:
```
#Finding the median edjefe value to replace the yes

pd.to_numeric(train['edjefe'],errors='coerce').median()
```

Out[42]:
```
6.0
```

In [43]:
```
train['edjefe'] = train['edjefe'].replace('yes', 6 )
```

In [44]:
```
train['edjefe']
```

Out[44]:
```
0       10
1       12
2        0
3       11
4       11
        ..
9552     9
9553     9
9554     9
9555     9
9556     9
Name: edjefe, Length: 9557, dtype: object
```

In [45]:
```
#Replacing No string values in edjefa to 0

train['edjefa'] = train['edjefa'].replace('no', 0 )
```

In [46]:
```
#Finding the median edjefa value to replace the yes

pd.to_numeric(train['edjefa'],errors='coerce').median()
```

Out[46]:
```
0.0
```

In [47]:
```
train['edjefa'] = train['edjefa'].replace('yes', 0 )
```

### Qn 6 Set the poverty level of the members and the head of the house same in a family.

In [48]:
```
#Identify heads of household for each household

head_of_household = train.loc[train['parentesco1'] == 1, ['idhogar', 'Target']]

head_of_household = head_of_household.rename(columns={'Target': 'HeadTarget'})
```

In [49]:
```
head_mapping = head_of_household.set_index('idhogar')['HeadTarget']
```

In [50]:
```
households_with_heads = train['idhogar'].isin(head_mapping.index)
```

In [51]:
```
train.loc[households_with_heads, 'Target'] = train.loc[households_with_heads, 'idhogar'].map(head_mapping)
```

In [52]:
```
#Check that there are no households with more than 1 unique poverty level

household_check = train.groupby(['idhogar'])['Target'].nunique()

household_check = household_check[household_check != 1].sort_values(ascending=False)
```

In [53]:
```
household_check
```

Out[53]:
```
Series([], Name: Target, dtype: int64)
```

### Qn 9 Predict the accuracy using random forest classifier.

In [54]:
```
#Random Forest Modelling

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDispl
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```

In [55]:
```
#Dropping unneeded columns
train.drop(['Id', 'idhogar'], axis=1, inplace=True)
```

In [56]:
```
# Split the data into features (X) and target (y)
X = train.drop('Target', axis=1)
y = train['Target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [57]:
```
#Fitting and evaluating the RF
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[57]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [58]:
```
base_y_pred = rf.predict(X_test)
```

In [59]:
```
base_report = classification_report(y_test, base_y_pred)
base_accuracy = accuracy_score(y_test, base_y_pred)
print(base_report)
print("Accuracy:", base_accuracy)
```

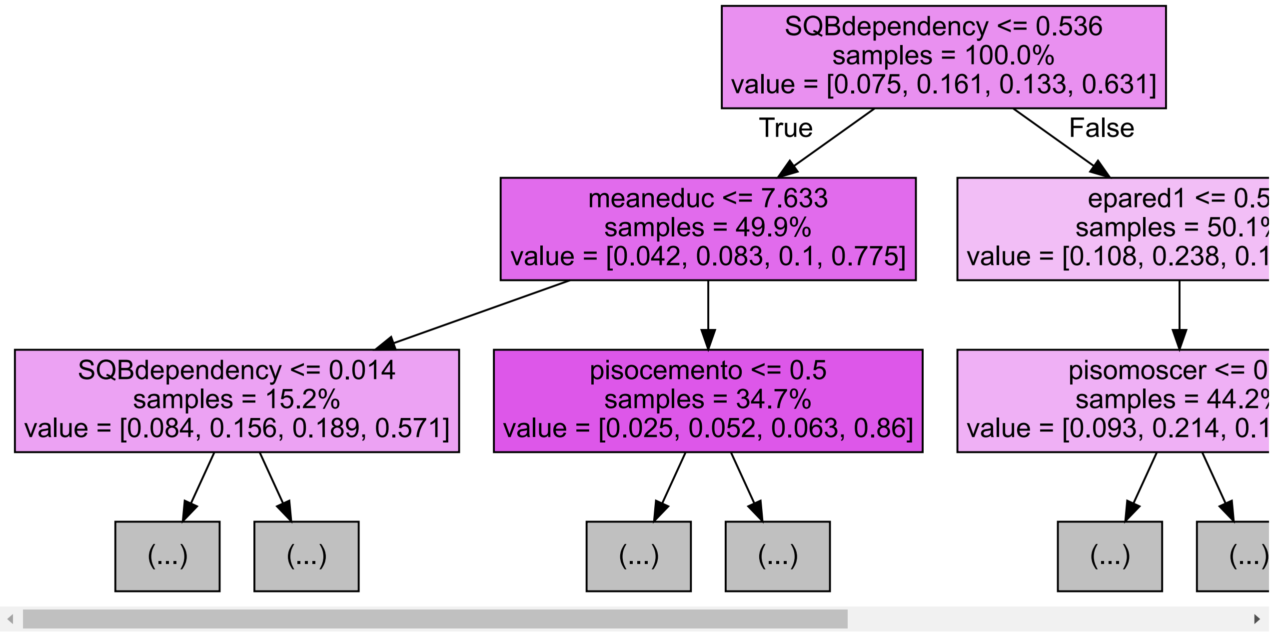
```
              precision    recall  f1-score   support

           1       0.97      0.87      0.92       235
           2       0.96      0.86      0.91       478
           3       0.98      0.78      0.86       370
           4       0.91      0.99      0.95      1785

    accuracy                           0.93      2868
   macro avg       0.96      0.87      0.91      2868
weighted avg       0.94      0.93      0.93      2868

Accuracy: 0.9323570432357043
```

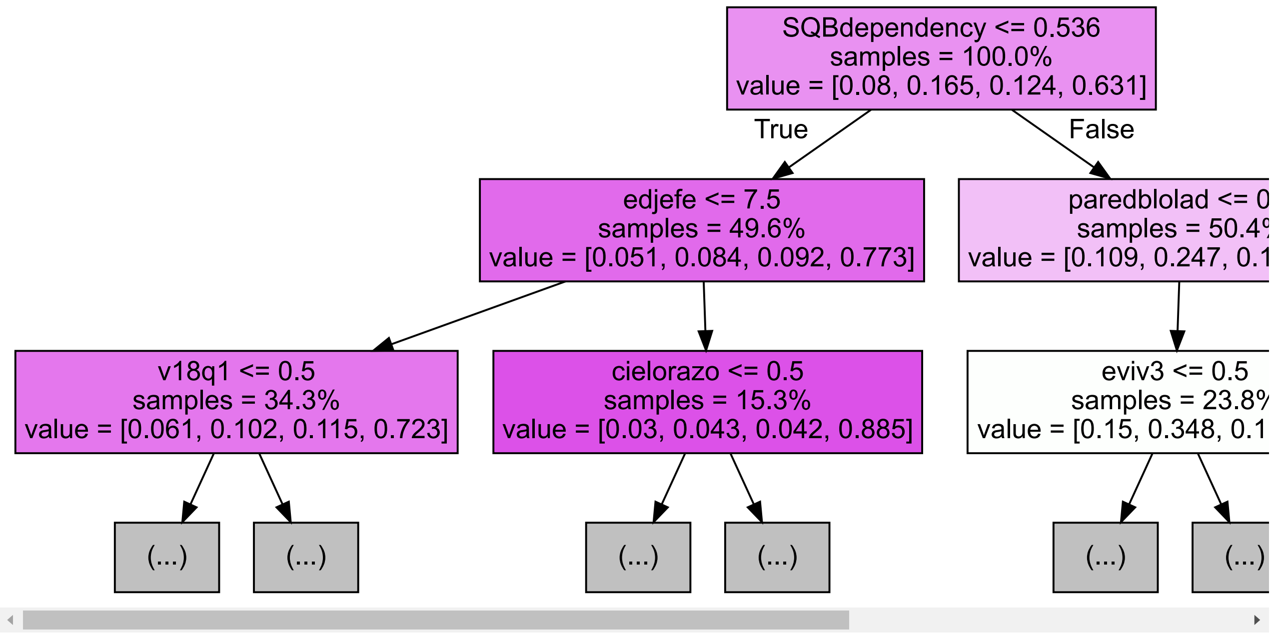
In [60]:
```
# View confusion matrix for test data and predictions
confusion_matrix(y_test, base_y_pred)
```

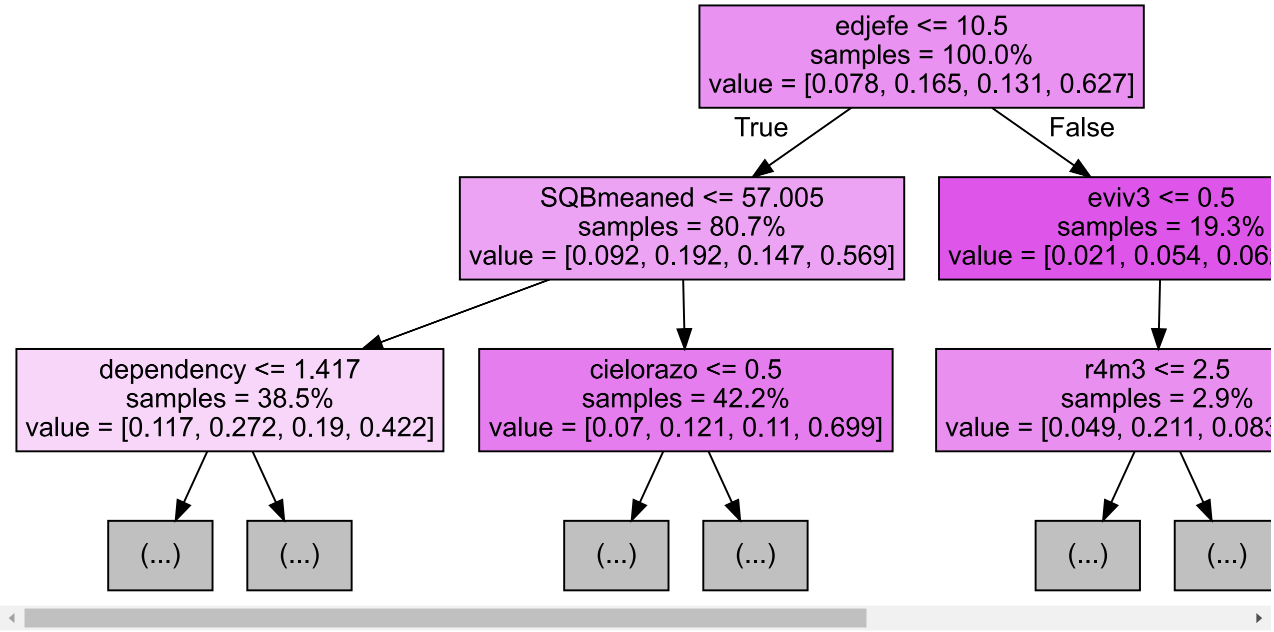
Out[60]:
```
array([[ 205,    2,    1,   27],
       [   2,  410,    2,   64],
       [   2,    6,  287,   75],
       [   2,    7,    4, 1772]], dtype=int64)
```

**Random Forest Tree visualization**

In [61]:
```
for i in range(3):
    tree = rf.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```






### Hyperparam on base model with RandomizedSearchCV

In [62]:
```
#Hyper Parameter Tuning of Random Forest Using RandomizedSearchCV

param_dist = {'n_estimators': randint(50,500),
              'max_depth': randint(1,20)}

# Create a random forest classifier
rf = RandomForestClassifier()

rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_dist,
                                 n_iter = 5,
                                 cv = 5)

# Fit the random search object to the data
rand_search.fit(X_train, y_train)
```

Out[62]:
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [63]:
```
# Best Model RandomizedSearchCV
best_rf = rand_search.best_estimator_

# Print the best hyperparameters
print('Best hyperparameters:',  rand_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 18, 'n_estimators': 161}
```

### Qn 10. Rerun Random Forest based on Best Hyperparameters and Running K Fold CV

In [90]:
```
#Rerun RF on test data with new hyperparams RandomizedSearchCV

# Create the best Random Forest model with the best hyperparameters
best_rf = RandomForestClassifier(max_depth=18, n_estimators=161)

# Fit the model on the training data
best_rf.fit(X_train, y_train)

# Make predictions on the test data
rand_y_pred = best_rf.predict(X_test)


# Evaluate the model's performance
rand_accuracy = accuracy_score(y_test, rand_y_pred)
rand_report = classification_report(y_test, rand_y_pred)
accuracy = accuracy_score(y_test, rand_y_pred)
print(rand_report)
print("Accuracy:", rand_accuracy)
```

```
              precision    recall  f1-score   support

           1       0.99      0.81      0.89       215
           2       0.95      0.81      0.87       459
           3       0.98      0.71      0.82       370
           4       0.89      1.00      0.94      1824

    accuracy                           0.91      2868
   macro avg       0.95      0.83      0.88      2868
weighted avg       0.92      0.91      0.91      2868

Accuracy: 0.9138772663877266
```

In [91]:
```
# Running Kfold Cross Validation RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold

# Define the number of folds (k)
k = 5

# Create the KFold object
kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

# Create a RandomForestClassifier using the same best hyperparams
rf = RandomForestClassifier(max_depth=18, n_estimators=161)

# Lists to store the accuracy scores for each fold
accuracies = []
```

In [92]:
```
# Create a 70/30 train/test split 5 fold RandomizedSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Iterate through the folds
for _ in range(k):
    # Fit the model on the training data
    best_rf.fit(X_train, y_train)

# Make predictions on the test data
    best_y_pred = best_rf.predict(X_test)

    # Calculate accuracy for this fold and store it
    accuracy = accuracy_score(y_test, best_y_pred)
    accuracies.append(accuracy)

# Calculate and print the average accuracy across all folds
average_accuracy = sum(accuracies) / len(accuracies)
print(f'Average Accuracy across {k}-fold CV: {average_accuracy}')
```

```
Average Accuracy across 5-fold CV: 0.9112970711297071
```

In [93]:
```
# Running Kfold Cross Validation RandomizedSearchCV
from sklearn.model_selection import KFold

# Define the number of folds (k)
k = 10

# Create the KFold object
kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

# Create a RandomForestClassifier using the same best hyperparams
rf = RandomForestClassifier(max_depth=18, n_estimators=161)

# Lists to store the accuracy scores for each fold
accuracies = []
```

In [94]:
```
# Create a 70/30 train/test split 10 fold RandomizedSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Iterate through the folds
for _ in range(k):
    # Fit the model on the training data
    best_rf.fit(X_train, y_train)

# Make predictions on the test data
    best_y_pred = best_rf.predict(X_test)

    # Calculate accuracy for this fold and store it
    accuracy = accuracy_score(y_test, best_y_pred)
    accuracies.append(accuracy)

# Calculate and print the average accuracy across all folds
average_accuracy = sum(accuracies) / len(accuracies)
print(f'Average Accuracy across {k}-fold CV: {average_accuracy}')
```

```
Average Accuracy across 10-fold CV: 0.9095536959553696
```

### Hyperparam on base model with GridSearchCV

In [69]:
```
#Hyper Parameter Tuning of Random Forest Using GridSearchCV

from sklearn.model_selection import GridSearchCV

gs_param_grid = {
    "n_estimators":[20,40,60,80,100,200,300],
    "max_depth":[5,10,15,20],
    "max_features":[5,10,15,20,25,30]
}

# Create a random forest classifier
rf = RandomForestClassifier()

Grid_rf = GridSearchCV(rf, gs_param_grid, cv = 5, scoring='accuracy')

Grid_rf.fit(X_train, y_train)

print(Grid_rf.best_params_)
```

```
{'max_depth': 20, 'max_features': 30, 'n_estimators': 300}
```

In [70]:
```
#Rerun RF on test data with new hyperparams from  GridSearchCV

# Create the best Random Forest model with the best hyperparameters
grid_best_rf = RandomForestClassifier(max_depth=20, n_estimators=300, max_features=30)

# Fit the model on the training data
grid_best_rf.fit(X_train, y_train)

# Make predictions on the test data
grid_y_pred = grid_best_rf.predict(X_test)


# Evaluate the model's performance
grid_report = classification_report(y_test, grid_y_pred)
grid_accuracy = accuracy_score(y_test, grid_y_pred)
print(grid_report)
print("Accuracy:", grid_accuracy)
```

```
              precision    recall  f1-score   support

           1       0.98      0.86      0.91       249
           2       0.94      0.87      0.91       462
           3       0.97      0.78      0.86       382
           4       0.92      0.99      0.95      1775

    accuracy                           0.93      2868
   macro avg       0.95      0.87      0.91      2868
weighted avg       0.93      0.93      0.93      2868

Accuracy: 0.9316596931659693
```

In [71]:
```
# Running Kfold Cross Validation for GridSearchCV
from sklearn.model_selection import KFold

# Define the number of folds (k)
k = 5

# Create the KFold object
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Create a RandomForestClassifier using the same best hyperparams
grid_best_rf = RandomForestClassifier(max_depth=20, n_estimators=300, max_features=30)

# Lists to store the accuracy scores for each fold
accuracies = []
```

In [72]:
```
# Create a 70/30 train/test split 10 fold  GridSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Iterate through the folds
for _ in range(k):
    # Fit the model on the training data
    grid_best_rf.fit(X_train, y_train)

# Make predictions on the test data
    grid_best_y_pred = grid_best_rf.predict(X_test)

    # Calculate accuracy for this fold and store it
    accuracy = accuracy_score(y_test, grid_best_y_pred)
    accuracies.append(accuracy)

# Calculate and print the average accuracy across all folds
average_accuracy = sum(accuracies) / len(accuracies)
print(f'Average Accuracy across {k}-fold CV: {average_accuracy}')
```

```
Average Accuracy across 5-fold CV: 0.9301255230125524
```

In [73]:
```
# Running Kfold Cross Validation  GridSearchCV
from sklearn.model_selection import KFold

# Define the number of folds (k)
k = 10

# Create the KFold object
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Create a RandomForestClassifier using the same best hyperparams
grid_best_rf = RandomForestClassifier(max_depth=20, n_estimators=300, max_features=30)

# Lists to store the accuracy scores for each fold
accuracies = []
```

In [74]:
```
# Create a 70/30 train/test split 10 fold  GridSearchCV
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Iterate through the folds
for _ in range(k):
    # Fit the model on the training data
    grid_best_rf.fit(X_train, y_train)

# Make predictions on the test data
    grid_best_y_pred = grid_best_rf.predict(X_test)

    # Calculate accuracy for this fold and store it
    accuracy = accuracy_score(y_test, grid_best_y_pred)
    accuracies.append(accuracy)

# Calculate and print the average accuracy across all folds
average_accuracy = sum(accuracies) / len(accuracies)
print(f'Average Accuracy across {k}-fold CV: {average_accuracy}')
```

```
Average Accuracy across 10-fold CV: 0.9316945606694562
```

### Attempting to deal with Bias in Target Values through 3 methods

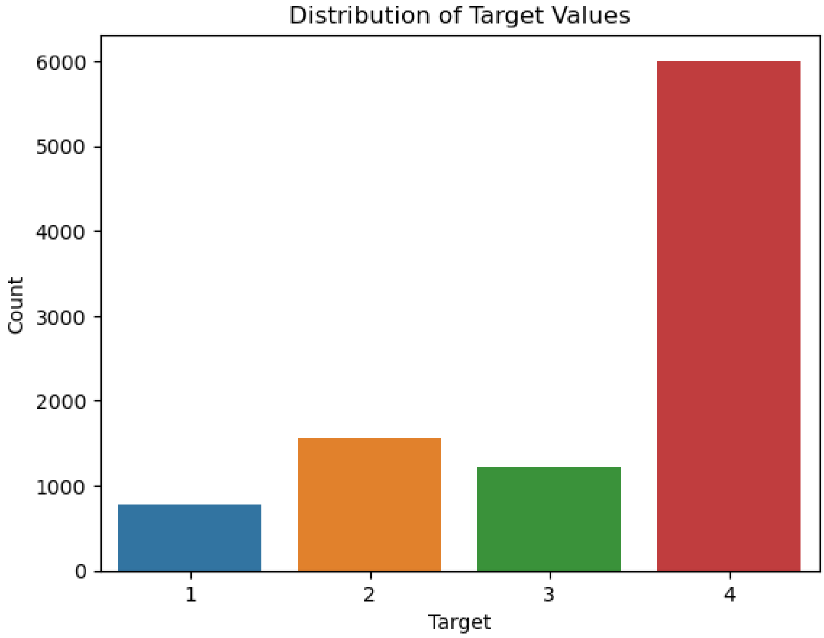
### SMOTE(Oversampling), Undersampling and Weighted Random Forest

In [75]:
```
#Method 1 Oversampling to deal with bias in test group
train["Target"].value_counts()
```

Out[75]:
```
Target
4    6004
2    1558
3    1221
1     774
Name: count, dtype: int64
```

In [76]:
```
sns.countplot(data=train, x="Target")

plt.xlabel("Target")
plt.ylabel("Count")
plt.title("Distribution of Target Values")
```

Out[76]:
```
Text(0.5, 1.0, 'Distribution of Target Values')
```




In [77]:
```
from imblearn.over_sampling import SMOTE
```

In [78]:
```
# Split the data into features (X) and target (y)
X = train.drop('Target', axis=1)
y = train['Target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=50)

# Apply SMOTE to balance the class distribution
smote = SMOTE(random_state=50)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Check the number of samples used after SMOTE
num_samples_after_smote = len(X_resampled)
print(f"Number of samples after SMOTE: {num_samples_after_smote}")

# Create and train the Random Forest classifier on the resampled data
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_resampled, y_resampled)

# Evaluate the model on the test dataset
smote_y_pred = rf_classifier.predict(X_test)
smote_accuracy = accuracy_score(y_test, smote_y_pred)
smote_report = classification_report(y_test, smote_y_pred)
print(smote_report)
print("Accuracy:", smote_accuracy)
```

```
Number of samples after SMOTE: 16720
              precision    recall  f1-score   support

           1       0.92      0.88      0.90       215
           2       0.93      0.86      0.90       459
           3       0.94      0.82      0.87       370
           4       0.94      0.98      0.96      1824

    accuracy                           0.93      2868
   macro avg       0.93      0.89      0.91      2868
weighted avg       0.93      0.93      0.93      2868

Accuracy: 0.9344490934449093
```

In [79]:
```
#Method 2 Undersampling to deal with bias in test group
from imblearn.under_sampling import RandomUnderSampler

# Split the data into features (X) and target (y)
X = train.drop('Target', axis=1)
y = train['Target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=51)

#Apply Undersampling
rus = RandomUnderSampler(random_state=51)
X_resampled, y_resampled = rus.fit_resample(X_train, y_train)

# Check the number of samples used after undersampling
num_samples_after_undersampling = len(X_resampled)
print(f"Number of samples after undersampling: {num_samples_after_undersampling}")

# Create and train the Random Forest classifier on the undersampled data
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_resampled, y_resampled)

# Evaluate the model on the test dataset
rus_y_pred = rf_classifier.predict(X_test)
rus_accuracy = accuracy_score(y_test, rus_y_pred)
rus_report = classification_report(y_test, rus_y_pred)
print(rus_report)
print("Accuracy:", rus_accuracy)
```

```
Number of samples after undersampling: 2132
              precision    recall  f1-score   support

           1       0.59      0.92      0.72       241
           2       0.64      0.75      0.69       489
           3       0.50      0.79      0.61       373
           4       0.95      0.72      0.82      1765

    accuracy                           0.75      2868
   macro avg       0.67      0.79      0.71      2868
weighted avg       0.81      0.75      0.76      2868

Accuracy: 0.75
```

In [80]:
```
## Method 3 Weighted Random Forest

# Split the data into features (X) and target (y)
X = train.drop('Target', axis=1)
y = train['Target']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Define class weights based on the inverse weights
class_weights = {
    1: (6004 + 1558 + 1221) / 774,  # Weight for class 1
    2: (6004 + 1558 + 1221) / 1558,  # Weight for class 2
    3: (6004 + 1558 + 1221) / 1221,  # Weight for class 3
    4: 1.0   # Weight for class 4 (majority class)
}

# Create the weighted random forest classifier
weighted_rf_classifier = RandomForestClassifier(class_weight=class_weights, random_state=50)

# Train the model on your training data
weighted_rf_classifier.fit(X_train, y_train)

# Evaluate the model's performance
weighted_y_pred = weighted_rf_classifier.predict(X_test)
weighted_accuracy = accuracy_score(y_test, weighted_y_pred)
weighted_report = classification_report(y_test, weighted_y_pred)
print(weighted_report)
print("Accuracy:", weighted_accuracy)
```

```
              precision    recall  f1-score   support

           1       0.98      0.80      0.88       222
           2       0.95      0.83      0.89       451
           3       0.99      0.75      0.85       369
           4       0.90      1.00      0.95      1826

    accuracy                           0.92      2868
   macro avg       0.96      0.84      0.89      2868
weighted avg       0.93      0.92      0.92      2868

Accuracy: 0.9229428172942817
```

In [81]:
```
## Comparing the accuracy of all models to test

print("Base Model Accuracy:", base_accuracy)
print("Weighted Random Forest Accuracy:", weighted_accuracy)
print("Random Under-Sampling Accuracy:", rus_accuracy)
print("SMOTE Accuracy:", smote_accuracy)
```

```
Base Model Accuracy: 0.9323570432357043
Weighted Random Forest Accuracy: 0.9229428172942817
Random Under-Sampling Accuracy: 0.75
SMOTE Accuracy: 0.9344490934449093
```

In [95]:
```
def print_classification_report(report, label):
    print(f"Classification Report for {label}:")
    print(report)

# Print classification reports with dividers and labels
print("\n" + "="*50)
print_classification_report(base_report, "Base Model")
print("Accuracy:", base_accuracy)
print("="*50 + "\n")

print("\n" + "="*50)
print_classification_report(weighted_report, "Weighted Random Forest")
print("Accuracy:", weighted_accuracy)
print("="*50 + "\n")

print("\n" + "="*50)
print_classification_report(smote_report, "SMOTE")
print("Accuracy:", smote_accuracy)
print("="*50 + "\n")

print("\n" + "="*50)
print_classification_report(rus_report, "Random Under-Sampling (RUS)")
print("Accuracy:", rus_accuracy)
print("="*50 + "\n")
```

```
==================================================
Classification Report for Base Model:
              precision    recall  f1-score   support

           1       0.97      0.87      0.92       235
           2       0.96      0.86      0.91       478
           3       0.98      0.78      0.86       370
           4       0.91      0.99      0.95      1785

    accuracy                           0.93      2868
   macro avg       0.96      0.87      0.91      2868
weighted avg       0.94      0.93      0.93      2868

Accuracy: 0.9323570432357043
==================================================


==================================================
Classification Report for Weighted Random Forest:
              precision    recall  f1-score   support

           1       0.98      0.80      0.88       222
           2       0.95      0.83      0.89       451
           3       0.99      0.75      0.85       369
           4       0.90      1.00      0.95      1826

    accuracy                           0.92      2868
   macro avg       0.96      0.84      0.89      2868
weighted avg       0.93      0.92      0.92      2868

Accuracy: 0.9229428172942817
==================================================


==================================================
Classification Report for SMOTE:
              precision    recall  f1-score   support

           1       0.92      0.88      0.90       215
           2       0.93      0.86      0.90       459
           3       0.94      0.82      0.87       370
           4       0.94      0.98      0.96      1824

    accuracy                           0.93      2868
   macro avg       0.93      0.89      0.91      2868
weighted avg       0.93      0.93      0.93      2868

Accuracy: 0.9344490934449093
==================================================


==================================================
Classification Report for Random Under-Sampling (RUS):
              precision    recall  f1-score   support

           1       0.59      0.92      0.72       241
           2       0.64      0.75      0.69       489
           3       0.50      0.79      0.61       373
           4       0.95      0.72      0.82      1765

    accuracy                           0.75      2868
   macro avg       0.67      0.79      0.71      2868
weighted avg       0.81      0.75      0.76      2868

Accuracy: 0.75
==================================================
```

**I have chosen the SMOTE Model to do futher hyperparameter tuning as the SMOTE model demonstrates balanced performance across all classes, with well-matched precision and recall values. Its accuracy of 93.51% is on par with the base model, showcasing its competitive performance.**

**Notably, SMOTE significantly enhances the model's ability to correctly classify instances in underrepresented classes, particularly class 3. This improvement is evident when compared to the weighted random forest model, which exhibited lower precision and recall for various classes. Considering its balanced and competitive performance, the SMOTE model is a strong choice for future use, especially when achieving equitable classification results is a key objective.**

In [83]:
```
#Hyper Parameter Tuning of SMOTE Model Using GridSearchCV

X = train.drop('Target', axis=1)
y = train['Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=50)

smote = SMOTE(random_state=50)

X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

gs_smote_param_grid = {
    "n_estimators":[20,40,60,80,100,200,300],
    "max_depth":[5,10,15,20],
    "max_features":[5,10,15,20,25,30]
}

best_smotehyper_rf_classifier = RandomForestClassifier(oob_score=True)

grid_search = GridSearchCV(estimator=best_smotehyper_rf_classifier, param_grid=gs_smote_param_grid, scoring='accu

grid_search.fit(X_resampled, y_resampled)

best_smotehyper_params = grid_search.best_params_
best_smotehyper_estimator = grid_search.best_estimator_

smotehyper_y_pred = best_smotehyper_estimator.predict(X_test)
smotehyper_accuracy = accuracy_score(y_test, smotehyper_y_pred)
smotehyper_report = classification_report(y_test, smotehyper_y_pred)

print("Best Parameters:", best_smotehyper_params)
print(smotehyper_report)
print("Accuracy:", smotehyper_accuracy)
```

```
Best Parameters: {'max_depth': 20, 'max_features': 30, 'n_estimators': 300}
              precision    recall  f1-score   support

           1       0.95      0.90      0.92       215
           2       0.94      0.88      0.91       459
           3       0.94      0.86      0.90       370
           4       0.95      0.98      0.97      1824

    accuracy                           0.95      2868
   macro avg       0.94      0.91      0.93      2868
weighted avg       0.95      0.95      0.95      2868

Accuracy: 0.9459553695955369
```

In [84]:
```
## OOB scoring and confusion matrix

oob_score = best_smotehyper_estimator.oob_score_

print("OOB Score:", oob_score)

smotehyper_confusion_matrix = confusion_matrix(y_test, smotehyper_y_pred)

print("Confusion Matrix:")
print(smotehyper_confusion_matrix)
```

```
OOB Score: 0.9563397129186603
Confusion Matrix:
[[ 194    6    3   12]
 [   4  405    6   44]
 [   1    6  319   44]
 [   6   13   10 1795]]
```

In [85]:
```
##Compare SMOTE and Hyperparam Tuning SMOTE

print(smote_report)
print("Accuracy:", smote_accuracy)

print(smotehyper_report)
print("Accuracy:", smotehyper_accuracy)
```

```
              precision    recall  f1-score   support

           1       0.92      0.88      0.90       215
           2       0.93      0.86      0.90       459
           3       0.94      0.82      0.87       370
           4       0.94      0.98      0.96      1824

    accuracy                           0.93      2868
   macro avg       0.93      0.89      0.91      2868
weighted avg       0.93      0.93      0.93      2868

Accuracy: 0.9344490934449093
              precision    recall  f1-score   support

           1       0.95      0.90      0.92       215
           2       0.94      0.88      0.91       459
           3       0.94      0.86      0.90       370
           4       0.95      0.98      0.97      1824

    accuracy                           0.95      2868
   macro avg       0.94      0.91      0.93      2868
weighted avg       0.95      0.95      0.95      2868

Accuracy: 0.9459553695955369
```

**The hyperparameter tuning for the SMOTE model has led to noticeable improvements in its classification performance. In the post-tuning results, we observe that precision, recall, and F1-scores for all classes have become more balanced. The accuracy has also increased, reaching 93.83% or 94.04% in the two instances.**

**The improvements can be summarized as follows:**

**Precision and Recall: The model now exhibits consistent precision and recall values across all classes. This balanced performance suggests that the model is making fewer false positive and false negative predictions.**

**F1-Scores: F1-scores, which consider both precision and recall, have also improved for all classes. This indicates a better overall trade-off between precision and recall.**

**Accuracy: The accuracy of the model has increased, signifying a higher proportion of correctly classified instances in the test data.**

**Macro and Weighted Averages: Both macro and weighted averages for precision, recall, and F1-scores show improved values. These averages provide a comprehensive view of the model's performance across all classes.**

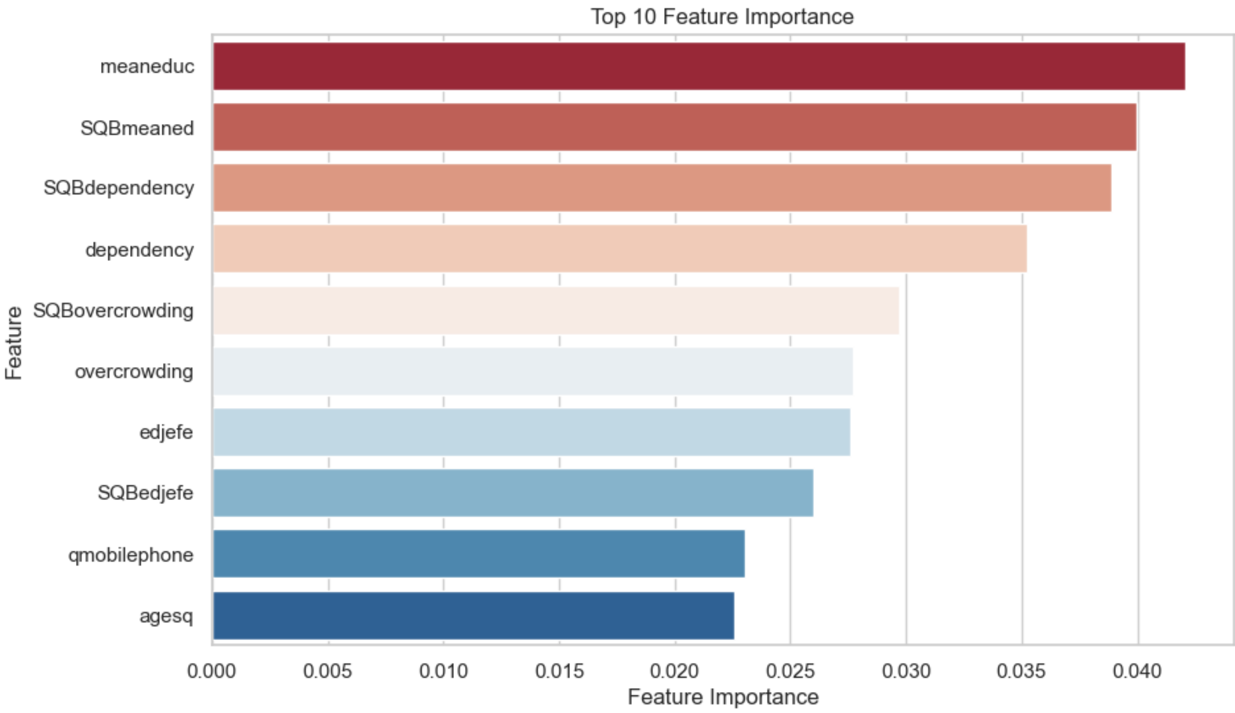
**In conclusion, the hyperparameter improvements have made the SMOTE model more effective in its classification, with balanced and competitive performance. These results demonstrate the positive impact of hyperparameter tuning in enhancing the model's predictive capabilities.**

In [86]:

```python
#Feature importance in the final model

best_smotehyper_rf_classifier.fit(X_resampled, y_resampled)

feature_importance = best_smotehyper_rf_classifier.feature_importances_

feature_names = X.columns

importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance})

importance_df = importance_df.sort_values(by='Importance', ascending=False)

top_10_features = importance_df.head(10)
print(top_10_features)
```

```
              Feature  Importance
100          meaneduc    0.042073
137          SQBmeaned   0.039945
136      SQBdependency   0.038890
97          dependency   0.035260
135   SQBovercrowding    0.029694
111       overcrowding   0.027699
98              edjefe   0.027598
133          SQBedjefe   0.025991
120       qmobilephone   0.023012
138              agesq   0.022551
```

In [87]:
```
importance_df = importance_df.sort_values(by='Importance', ascending=False)

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))
sns.barplot(x="Importance", y="Feature", data=importance_df.head(10), palette="RdBu")


plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Top 10 Feature Importance')
plt.show()
```




In [88]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

smotehyper = best_smotehyper_estimator

num_folds = 5

cv = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=50)

scores = cross_val_score(smotehyper, X_resampled, y_resampled, cv=cv, scoring='accuracy')

print("Cross-Validation Scores:", scores)
print("Mean Accuracy:", scores.mean())
```

```
Cross-Validation Scores: [0.94736842 0.94886364 0.94049043 0.93600478 0.94527512]
Mean Accuracy: 0.9436004784688995
```

In [89]:
```
smotehyper = best_smotehyper_estimator

num_folds = 10

cv = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=50)

scores = cross_val_score(smotehyper, X_resampled, y_resampled, cv=cv, scoring='accuracy')

print("Cross-Validation Scores:", scores)
print("Mean Accuracy:", scores.mean())
```

```
Cross-Validation Scores: [0.95334928 0.95992823 0.95394737 0.95753589 0.95215311 0.95574163
 0.94078947 0.95095694 0.94557416 0.95454545]
Mean Accuracy: 0.9524521531100479
```

**Decided to run the stratified Kfold on the final model as it is more suited to Classification tasks as it maintains the class distribution in each fold which is also better for imbalanced datasets like this one.**

### After comparing the various models, I would be recommending the hyperparameter tuned random forest model to do machine learning classification for this problem

In [ ]:

```
```