Simplilearn Post Graduate Program - Data Science - In Partnership With Purdue University

Project Report - Income Qualification Prediction

Organization: **Simplilearn - Purdue University**
Batch: **PGP DS Mar 2022 COHORT 2**
Course: **PG DS - Machine Learning**
Project: **Income Qualification Prediction**
Programming Language: **Python**
Submitted by: **Lavkush Singh**

Problem Statement

The project aims at building a model to predict Income Qualification, given various attributes collected across the household and the people of Latin America, using Machine Learning.

Analysis Tasks to be performed

- Identify the output variable.
- Understand the type of data.
- Check if there are any biases in your dataset.
- Check whether all members of the house have the same poverty level.
- Check if there is a house without a family head.
- Set poverty level of the members and the head of the house within a family.
- Count how many null values are existing in columns.
- Remove null value rows of the target variable.
- Predict the accuracy using random forest classifier.
- Check the accuracy using random forest with cross validation.

In [1]:
```
# Importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_m
atrix, ConfusionMatrixDisplay, multilabel_confusion_matrix
```

In [2]:
```
# settings to display all columns

pd.set_option("display.max_columns", None)
pd.options.display.max_rows = None
```

In [3]:
```
# reading the data

train_data = pd.read_csv('Dataset/train.csv')
test_data = pd.read_csv('Dataset/test.csv')
```

In [4]:
```
train_data.head() # viewing first few observations of train dataset
```

Out[4]:

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

In [5]:
```
test_data.head() # viewing first few observations of test dataset
```

Out[5]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	1	2
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	1	2
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	1	2
3	ID_a8db26a79	NaN	0	14	0	1	1	1	1.0	0	1	1
4	ID_a62966799	175000.0	0	4	0	1	1	1	1.0	0	0	0

Task 1: Identify the output variable.

In [6]:
```
print("The output variable is 'Target' of train_data (level of income qualification)")
```

```
The output variable is 'Target' of train_data (level of income qualification)
```

Task 2: Understand the type of data.

In [7]:
```
train_data.shape # checking rows and cols of the train dataset
```

Out[7]:
```
(9557, 143)
```

In [8]:
```
train_data.dtypes.value_counts()  # count of distinct datatype of columns present in the dataset
```

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

In [9]:
```
test_data.shape  # checking rows and cols of the test dataset
```

Out[9]:
```
(23856, 142)
```

In [10]:
```
test_data.dtypes.value_counts()
```

Out[10]:
```
int64      129
float64      8
object       5
dtype: int64
```

In [11]:
```
print('The datatypes varies across int, float and object. Further inspection is needed to ensure the data is consistant with the datatype of the columns')
```

```
The datatypes varies across int, float and object. Further inspection is needed to ensure the data is consistant with the datatype of the columns
```

Task 3, 7, 8: Check if there are any biases and Null Values in your dataset and remove the Null Values

In [12]:
```
train_data.head(n = 3)
```

Out[12]:

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

In [13]:
```
test_data.head(n = 3)
```

Out[13]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	r4h3	r4
0	ID_2f6873615	NaN	0	5	0	1	1	0	NaN	1	1	2	
1	ID_1c78846d2	NaN	0	5	0	1	1	0	NaN	1	1	2	
2	ID_e5442cf6a	NaN	0	5	0	1	1	0	NaN	1	1	2	

Sub-Task: Checking if the datatype is consistant across the columns

In [14]:
```
train_data.select_dtypes(include='object').columns  # checking columns of only object datatype
```

Out[14]:
```
Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'], dtype='object')
```

In [15]:
```
test_data.select_dtypes(include='object').columns
```

Out[15]:
```
Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'], dtype='object')
```

In [16]:
```
train_data.dtypes[train_data.dtypes == 'object']
```

Out[16]:
```
Id            object
idhogar       object
dependency    object
edjefe        object
edjefa        object
dtype: object
```

In [17]:
```
test_data.dtypes[train_data.dtypes == 'object']
```

Out[17]:
```
Id            object
idhogar       object
dependency    object
edjefe        object
edjefa        object
dtype: object
```

In [18]:
```
# checking if 'dependency' column has any missing values in train dataset and
 test dataset

train_data['dependency'].isnull().any(), test_data['dependency'].isnull().any
()
```

Out[18]:
```
(False, False)
```

In [19]:
```
train_data['dependency'].unique()   # checking unique values to identify why n
umerical column is of object type
```

Out[19]:
```
array(['no', '8', 'yes', '3', '.5', '.25', '2', '.66666669', '.33333334',
       '1.5', '.40000001', '.75', '1.25', '.2', '2.5', '1.2', '4',
       '1.3333334', '2.25', '.22222222', '5', '.83333331', '.80000001',
       '6', '3.5', '1.6666666', '.2857143', '1.75', '.71428573',
       '.16666667', '.60000002'], dtype=object)
```

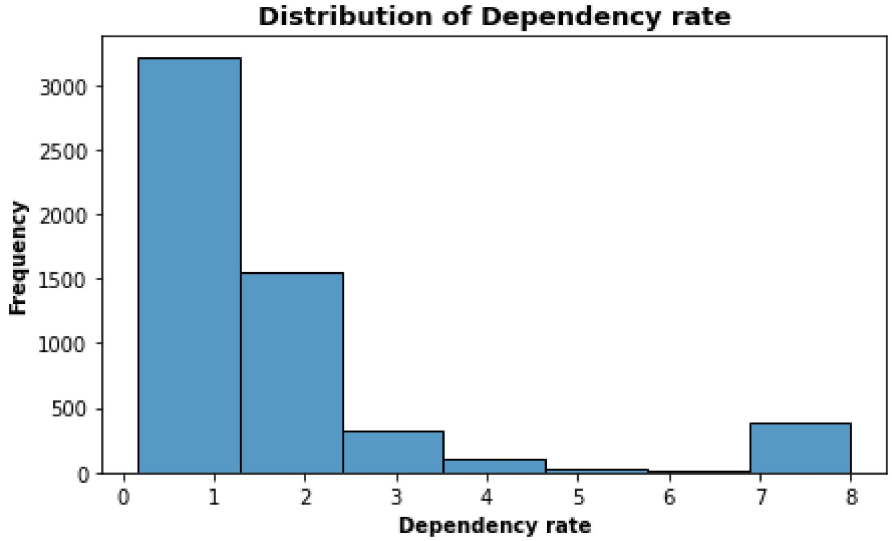
In [20]:
```
test_data['dependency'].unique()
```

Out[20]:
```
array(['.5', 'no', '8', 'yes', '.25', '2', '.33333334', '.375',
       '.60000002', '1.5', '.2', '.75', '.66666669', '3', '.14285715',
       '.40000001', '.80000001', '1.6666666', '.2857143', '1.25', '2.5',
       '5', '.85714287', '1.3333334', '.16666667', '4', '.125',
       '.83333331', '2.3333333', '7', '1.2', '3.5', '2.25', '3.3333333',
       '6'], dtype=object)
```

In [21]:
```
# getting all the values of dependency column from train dataset, except for
 'yes' and 'no'

dependency_values_train = train_data[train_data['dependency'].isin(['yes', 'n
o']) == False]['dependency']
```

In [22]:
```
# converting the obtained numerical values of dependency column from train dat
aset to float, to plot its distribution

dependency_values_train = dependency_values_train.astype('float64')
```

In [23]:
```
plt.figure(figsize=(7,4))
sns.histplot( x = dependency_values_train, bins = 7)
plt.title('Distribution of Dependency rate', fontsize = 13, fontweight="bold")
plt.xlabel('Dependency rate', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



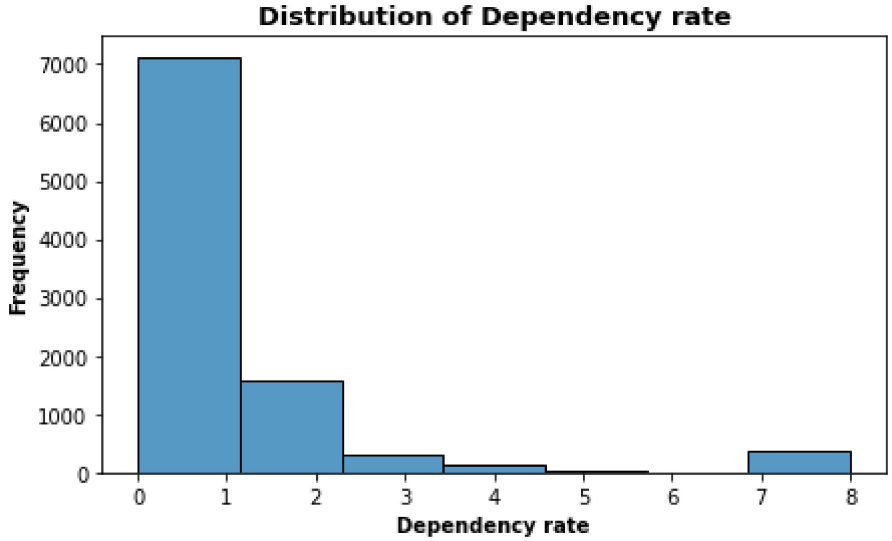
Replacing 'no' with 0 and 'yes' with median of 'dependency' column

In [24]:
```
train_data['dependency'] = train_data['dependency'].str.replace('no', '0')
test_data['dependency'] = test_data['dependency'].str.replace('no', '0')
```

In [25]:
```
median_dependency_train = train_data[train_data['dependency'] != 'yes']['depen
dency'].astype('float64').median()
```

In [26]:
```
train_data['dependency'] = train_data['dependency'].str.replace('yes', str(med
ian_dependency_train)).astype('float64')
test_data['dependency'] = test_data['dependency'].str.replace('yes', str(media
n_dependency_train)).astype('float64')
```

In [27]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'dependency', data = train_data, bins = 7);
plt.title('Distribution of Dependency rate', fontsize = 13, fontweight="bold")
plt.xlabel('Dependency rate', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



Checking remaining object types column and validating them for null values and consistency.

In [28]:
```
train_data.dtypes[train_data.dtypes == 'object']
```

Out[28]:
```
Id          object
idhogar     object
edjefe      object
edjefa      object
dtype: object
```

In [29]:
```
test_data.dtypes[train_data.dtypes == 'object']
```

Out[29]:
```
Id          object
idhogar     object
edjefe      object
edjefa      object
dtype: object
```

In [30]:
```
train_data[['edjefe', 'edjefa']].isnull().sum()
```

Out[30]:
```
edjefe    0
edjefa    0
dtype: int64
```

In [31]:
```
test_data[['edjefe', 'edjefa']].isnull().sum()
```

Out[31]:
```
edjefe    0
edjefa    0
dtype: int64
```

In [32]:
```
train_data['edjefe'].unique()
```

Out[32]:
```
array(['10', '12', 'no', '11', '9', '15', '4', '6', '8', '17', '7', '16',
       '14', '5', '21', '2', '19', 'yes', '3', '18', '13', '20'],
      dtype=object)
```

In [33]:
```
train_data['edjefa'].unique()
```

Out[33]:
```
array(['no', '11', '4', '10', '9', '15', '7', '14', '13', '8', '17', '6',
       '5', '3', '16', '19', 'yes', '21', '12', '2', '20', '18'],
      dtype=object)
```

In [34]:
```
test_data['edjefe'].unique()
```

Out[34]:
```
array(['no', '16', '10', '6', '11', '8', '13', '14', '5', '3', '9', '17',
       '15', '7', '21', '4', '12', '2', '20', 'yes', '19', '18'],
      dtype=object)
```

In [35]:
```
test_data['edjefa'].unique()
```

Out[35]:
```
array(['17', 'no', '11', '14', '10', '15', '9', '6', '8', '3', '2', '5',
       '16', '12', 'yes', '7', '13', '21', '4', '19', '18', '20'],
      dtype=object)
```

In [36]:
```
# Replaced 'no'values of 'edjefe', 'edjefa' columns of train and test dataset
 with 0

train_data[['edjefe', 'edjefa']] = train_data[['edjefe', 'edjefa']].replace('n
o', '0')
test_data[['edjefe', 'edjefa']] = test_data[['edjefe', 'edjefa']].replace('no'
, '0')
```

In [37]:
```
# calculated the median value of 'edjefe', 'edjefa' columns excluding the 'ye
s' value. This median wil be used for imputation

edjefe_median_train = train_data[train_data['edjefe'] != 'yes']['edjefe'].asty
pe('float64').median()
edjefa_median_train = train_data[train_data['edjefa'] != 'yes']['edjefa'].asty
pe('float64').median()
```

In [38]:
```
# imputing the median value of 'edjefe' from train data, into train and test d
ataset's 'edjefe' column in place of 'yes'

train_data['edjefe'] = train_data['edjefe'].str.replace('yes', str(edjefe_medi
an_train)).astype('float64')
test_data['edjefe'] = test_data['edjefe'].str.replace('yes', str(edjefe_median
_train)).astype('float64')
```

In [39]:
```
# imputing the median value of 'edjefa' from train data, into train and test d
ataset's 'edjefe' column in place of 'yes'

train_data['edjefa'] = train_data['edjefa'].str.replace('yes', str(edjefa_medi
an_train)).astype('float64')
test_data['edjefa'] = test_data['edjefa'].str.replace('yes', str(edjefa_median
_train)).astype('float64')
```

In [40]:
```
train_data.dtypes.value_counts()
```

Out[40]:
```
int64      130
float64     11
object       2
dtype: int64
```

In [41]:
```
test_data.dtypes.value_counts()
```

Out[41]:
```
int64      129
float64     11
object       2
dtype: int64
```

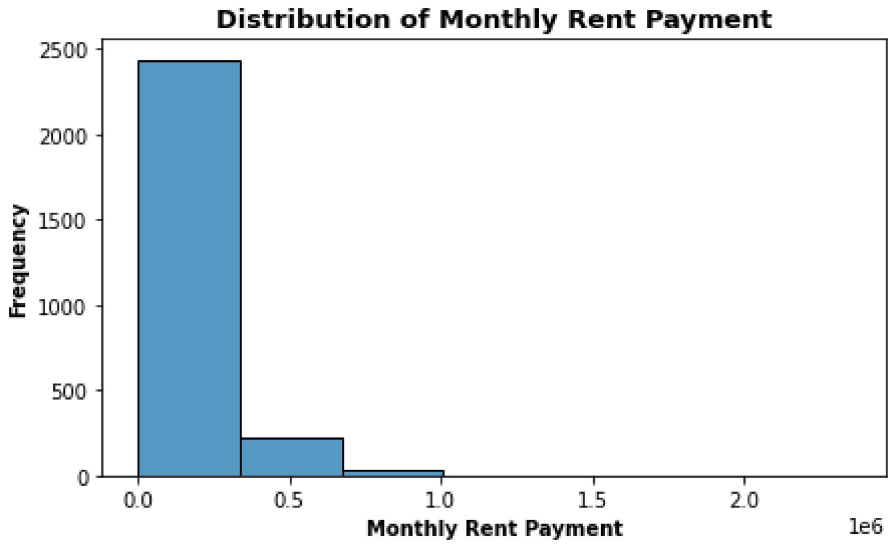
There are only 2 columns of object type, 'ID' and 'idhogar'. They will be dropped before splitting the dataset into training and testing set

Sub-Task: In order to check bias, checking the distribution of the numerical columns via plots.

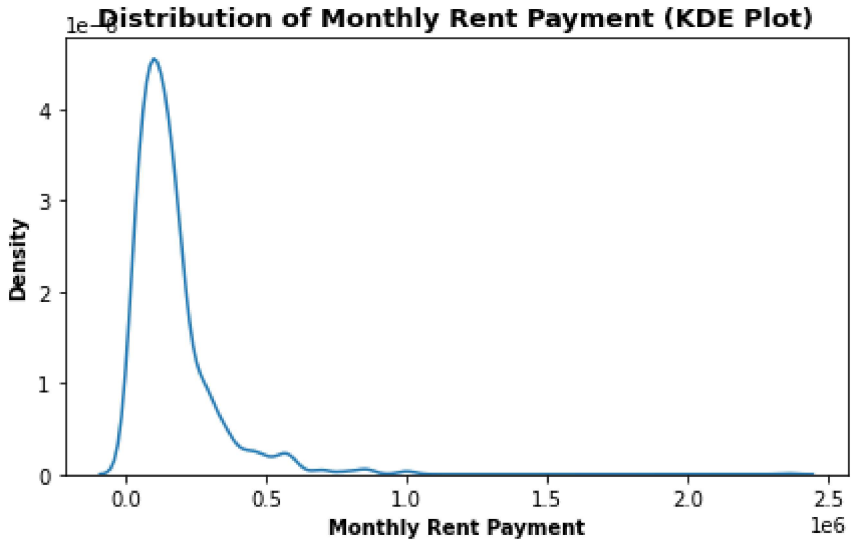
In [42]:
```
non_categ_cols = [col for col in train_data.columns if ~train_data[col].isin([
0,1]).all()]
non_categ_cols
```

Out[42]:
```
['Id',
 'v2a1',
 'rooms',
 'v18q1',
 'r4h1',
 'r4h2',
 'r4h3',
 'r4m1',
 'r4m2',
 'r4m3',
 'r4t1',
 'r4t2',
 'r4t3',
 'tamhog',
 'tamviv',
 'escolari',
 'rez_esc',
 'hhsize',
 'idhogar',
 'hogar_nin',
 'hogar_adul',
 'hogar_mayor',
 'hogar_total',
 'dependency',
 'edjefe',
 'edjefa',
 'meaneduc',
 'bedrooms',
 'overcrowding',
 'qmobilephone',
 'age',
 'SQBescolari',
 'SQBage',
 'SQBhogar_total',
 'SQBedjefe',
 'SQBhogar_nin',
 'SQBovercrowding',
 'SQBdependency',
 'SQBmeaned',
 'agesq',
 'Target']
```

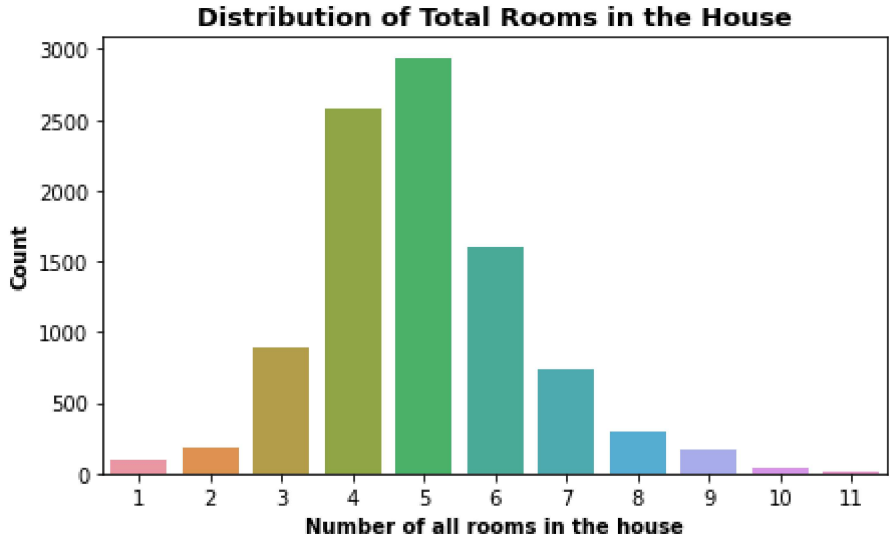
In [43]:
```
plt.figure(figsize=(7,4))
sns.histplot(train_data['v2a1'], bins=7)  # Monthly Rent Payment
plt.title('Distribution of Monthly Rent Payment', fontsize = 13, fontweight="b
old")
plt.xlabel('Monthly Rent Payment', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



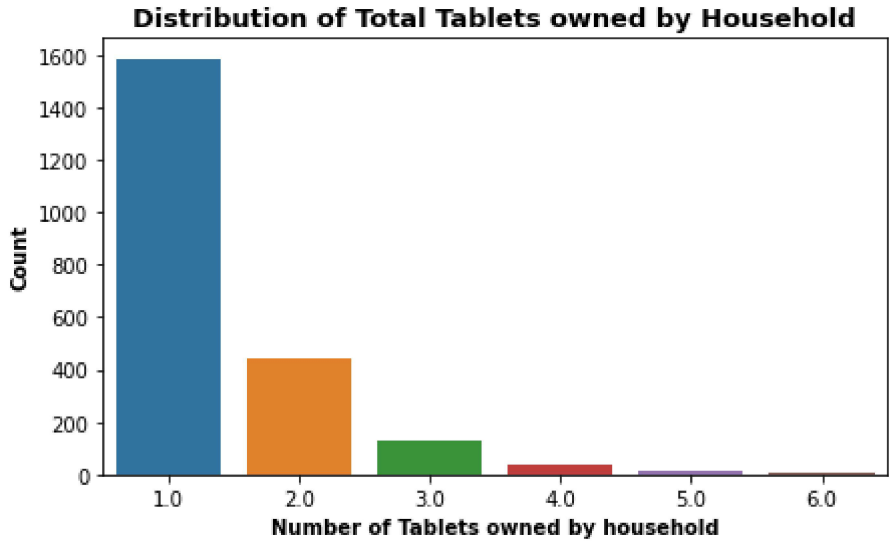
In [44]:
```
plt.figure(figsize=(7,4))
sns.kdeplot(train_data['v2a1'])  # Monthly Rent Payment
plt.title('Distribution of Monthly Rent Payment (KDE Plot)', fontsize = 13, fo
ntweight="bold")
plt.xlabel('Monthly Rent Payment', fontweight="bold")
plt.ylabel('Density', fontweight="bold")
plt.show()
```



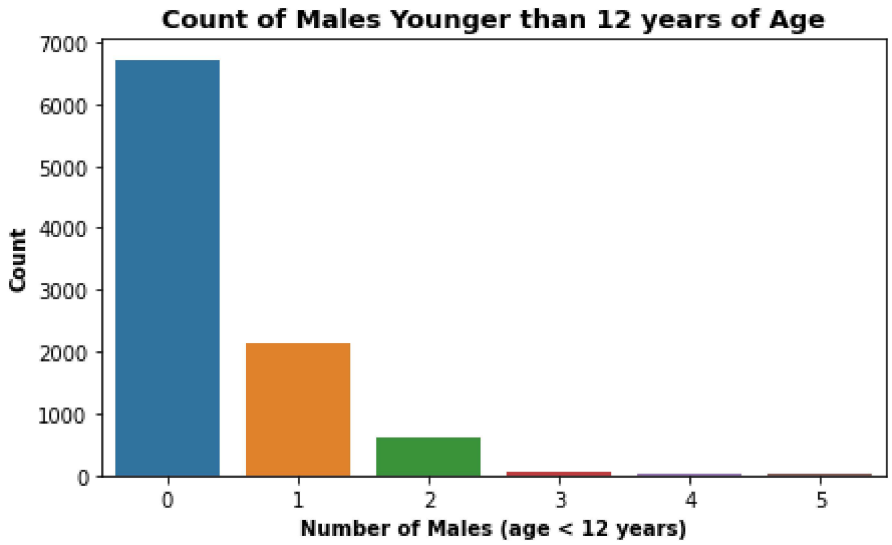
In [45]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'rooms', data = train_data)
plt.title('Distribution of Total Rooms in the House', fontsize = 13, fontweigh
t="bold")
plt.xlabel('Number of all rooms in the house', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



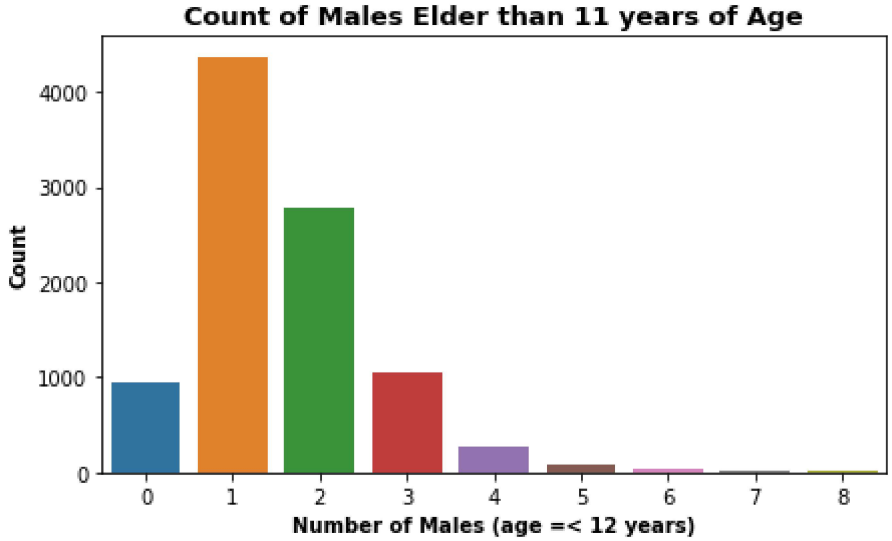
In [46]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'v18q1', data = train_data) #number of tablets household own
s
plt.title('Distribution of Total Tablets owned by Household', fontsize = 13, f
ontweight="bold")
plt.xlabel('Number of Tablets owned by household', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



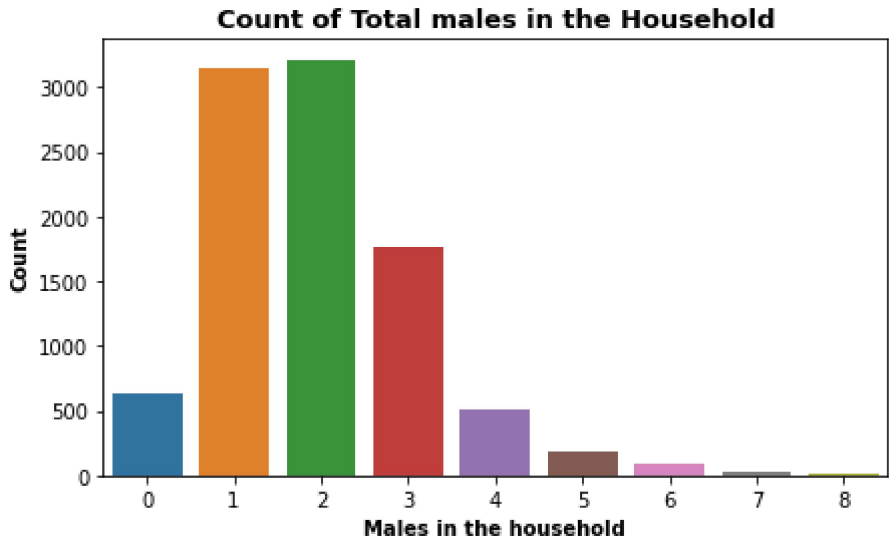
In [47]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4h1', data = train_data) # Males younger than 12 years of
 age
plt.title('Count of Males Younger than 12 years of Age', fontsize = 13, fontwe
ight="bold")
plt.xlabel('Number of Males (age < 12 years)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



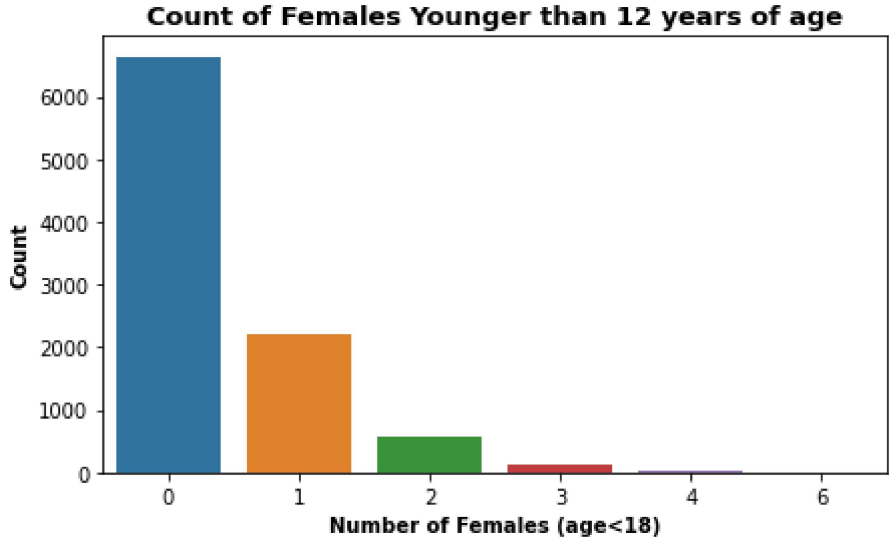
In [48]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4h2', data = train_data )  # Males 12 years of age and old
er
plt.title('Count of Males Elder than 11 years of Age', fontsize = 13, fontweig
ht="bold")
plt.xlabel('Number of Males (age =< 12 years)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



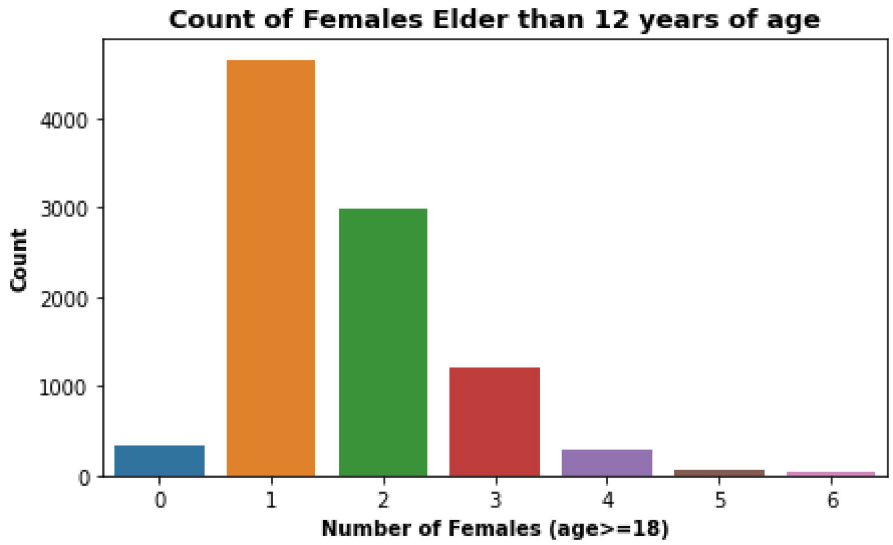
In [49]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4h3', data = train_data )  # Total males in the household
plt.title('Count of Total males in the Household', fontsize = 13, fontweight=
"bold")
plt.xlabel('Males in the household', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



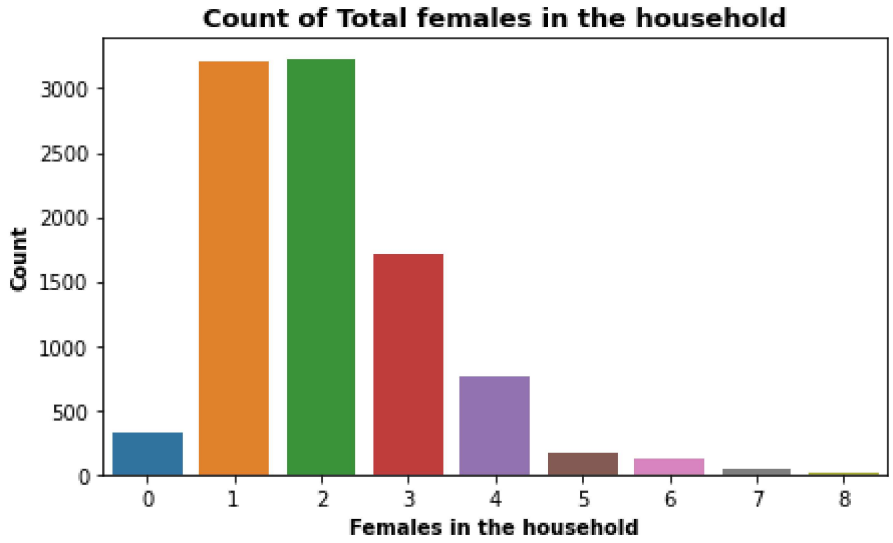
In [50]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4m1' , data = train_data) # Females younger than 12 years
 of age
plt.title('Count of Females Younger than 12 years of age', fontsize = 13, font
weight="bold")
plt.xlabel('Number of Females (age<18)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



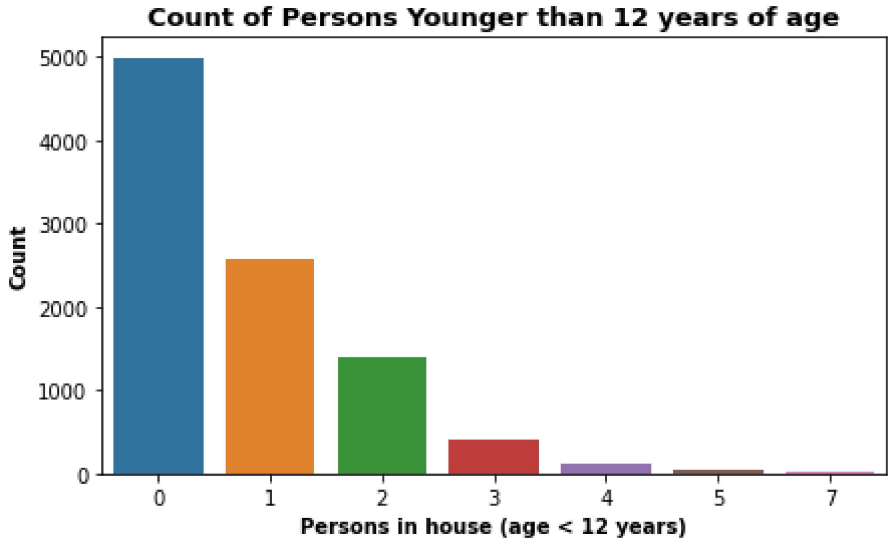
In [51]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4m2' , data = train_data) # Females 12 years of age and ol
der
plt.title('Count of Females Elder than 12 years of age', fontsize = 13, fontwe
ight="bold")
plt.xlabel('Number of Females (age>=18)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



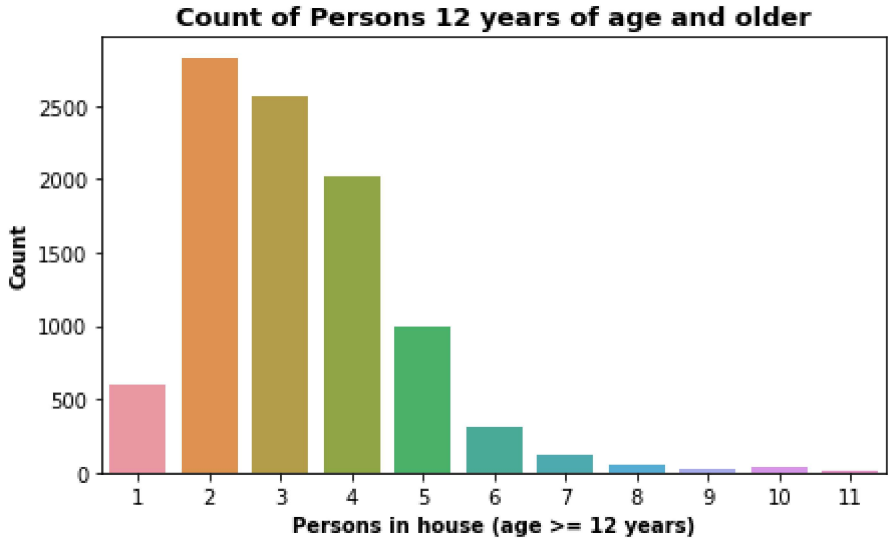
In [52]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4m3' , data = train_data) # Total females in the household
plt.title('Count of Total females in the household', fontsize = 13, fontweight
="bold")
plt.xlabel('Females in the household', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



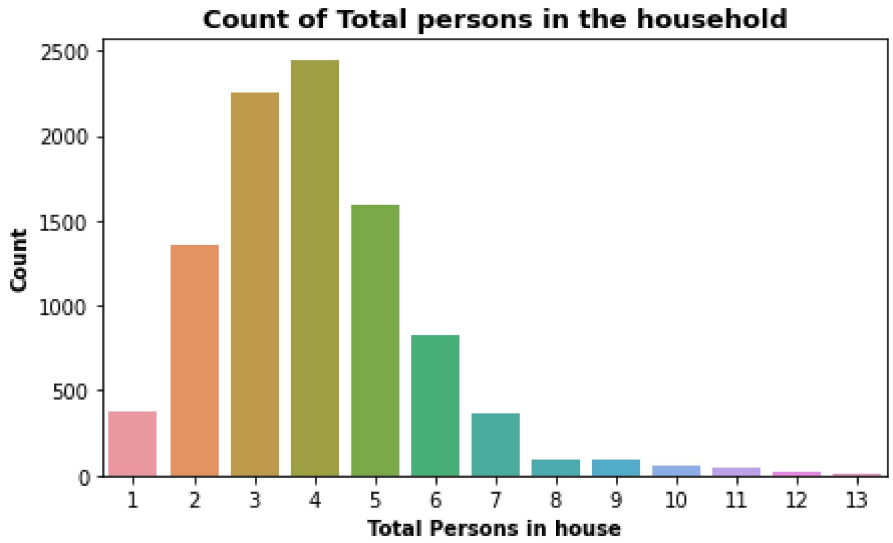
In [53]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4t1', data = train_data) # persons younger than 12 years of age
plt.title('Count of Persons Younger than 12 years of age', fontsize = 13, fontweight="bold")
plt.xlabel('Persons in house (age < 12 years)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



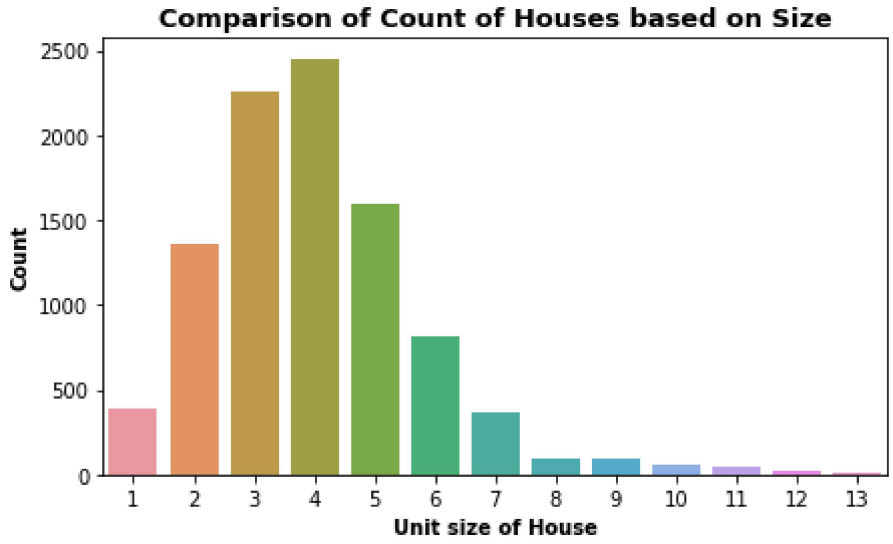
In [54]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4t2', data = train_data) #persons 12 years of age and older
plt.title('Count of Persons 12 years of age and older', fontsize = 13, fontweight="bold")
plt.xlabel('Persons in house (age >= 12 years)', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



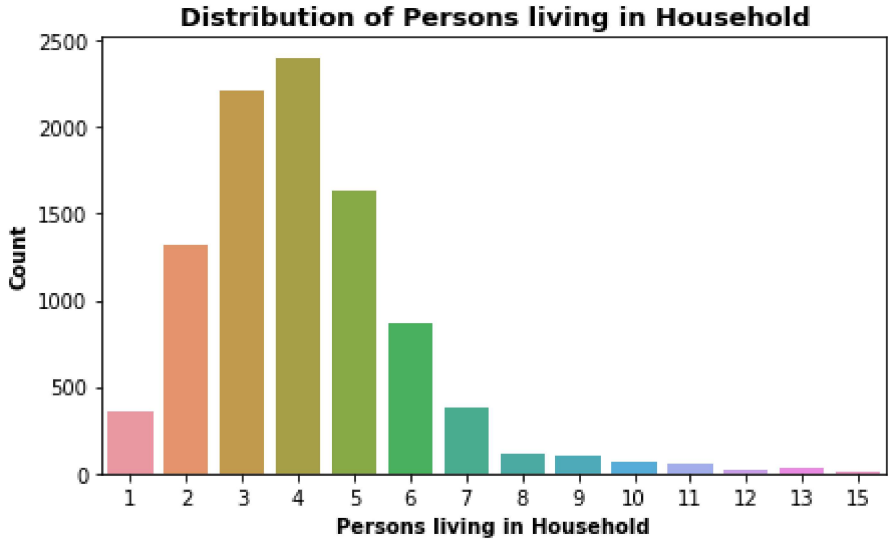
In [55]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'r4t3' , data = train_data) # Total persons in the household
plt.title('Count of Total persons in the household', fontsize = 13, fontweight
="bold")
plt.xlabel('Total Persons in house', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



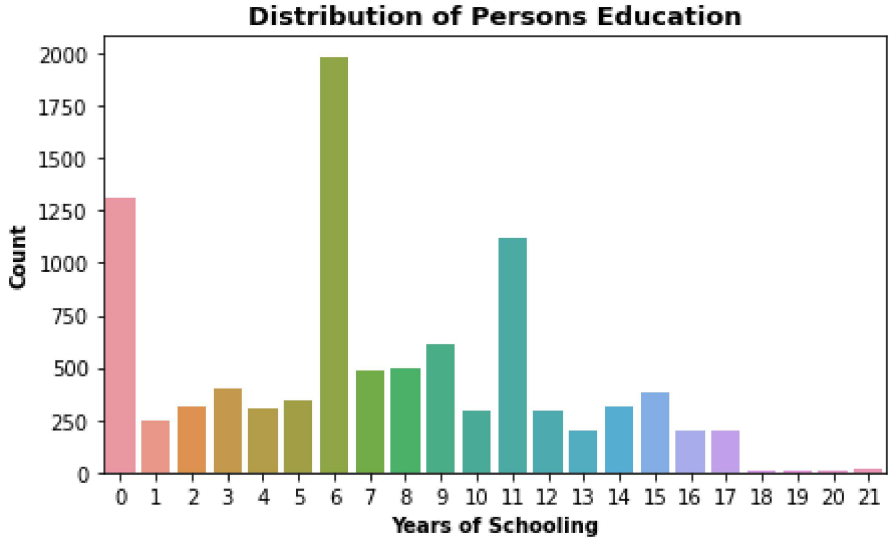
In [56]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'tamhog', data = train_data) # size of the household
plt.title('Comparison of Count of Houses based on Size', fontsize = 13, fontwe
ight="bold")
plt.xlabel('Unit size of House', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



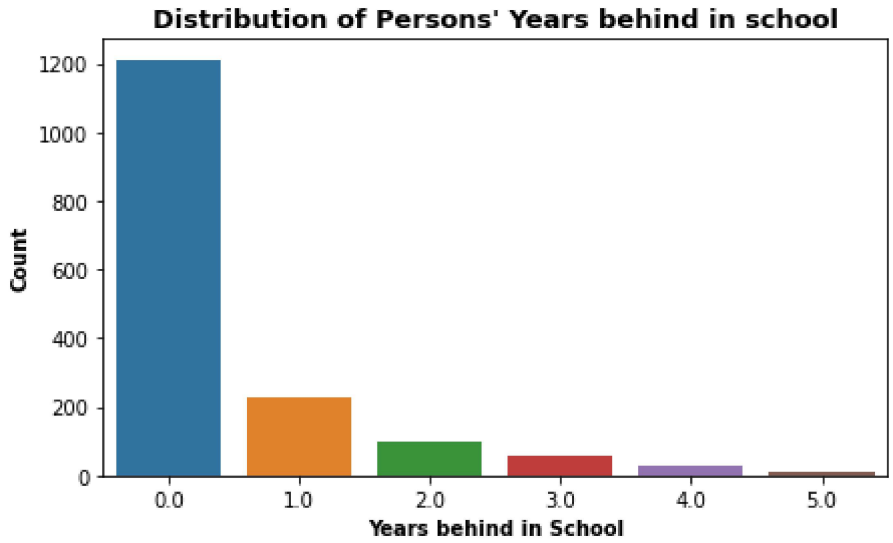
In [57]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'tamviv', data = train_data) # number of persons living in t
he household
plt.title('Distribution of Persons living in Household', fontsize = 13, fontwe
ight="bold")
plt.xlabel('Persons living in Household', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



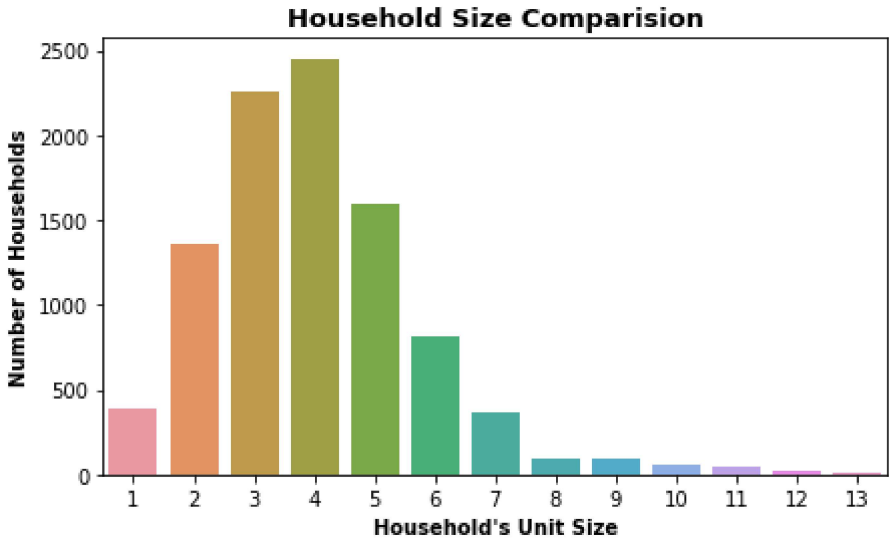
In [58]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'escolari', data = train_data ) # years of schooling
plt.title('Distribution of Persons Education', fontsize = 13, fontweight="bol
d")
plt.xlabel('Years of Schooling', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



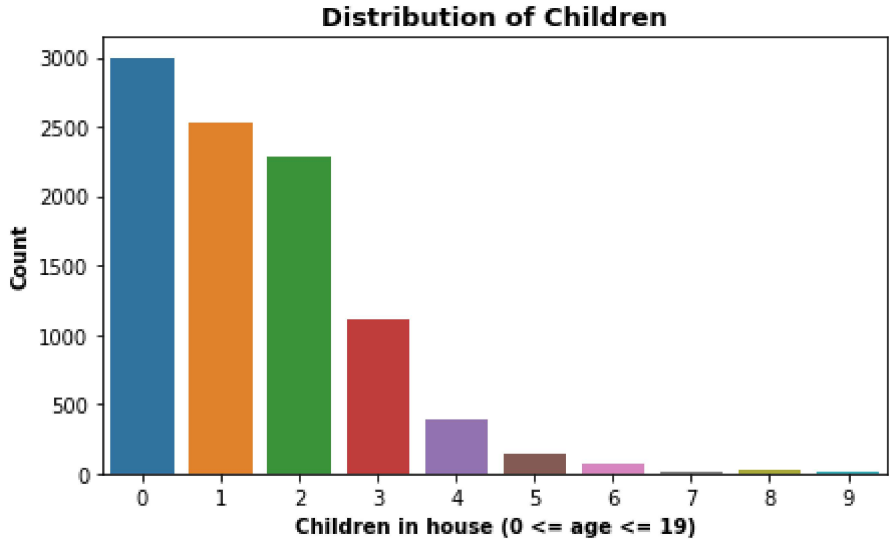
In [59]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'rez_esc', data = train_data) # Years behind in school
plt.title("Distribution of Persons' Years behind in school", fontsize = 13, fo
ntweight="bold")
plt.xlabel('Years behind in School', fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



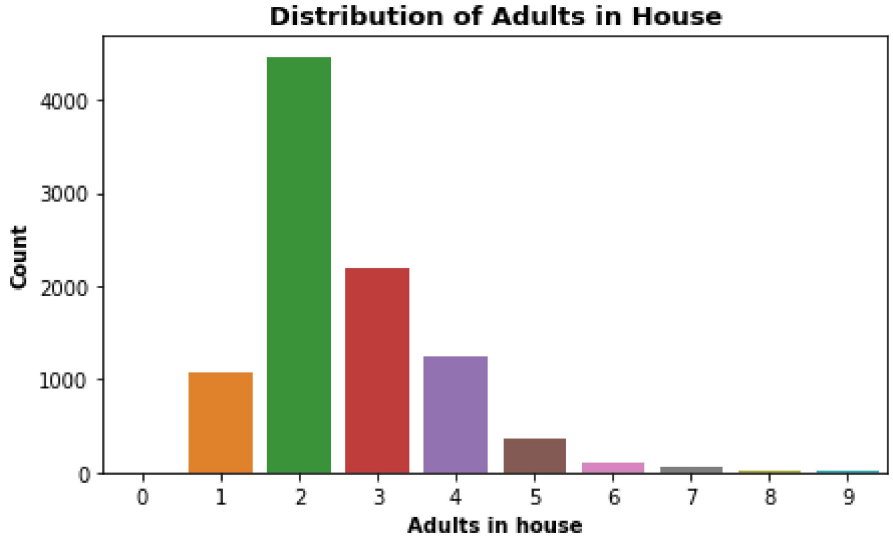
In [60]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'hhsize', data = train_data) # household size
plt.title("Household Size Comparision", fontsize = 13, fontweight="bold")
plt.xlabel("Household's Unit Size", fontweight="bold")
plt.ylabel('Number of Households', fontweight="bold")
plt.show()
```



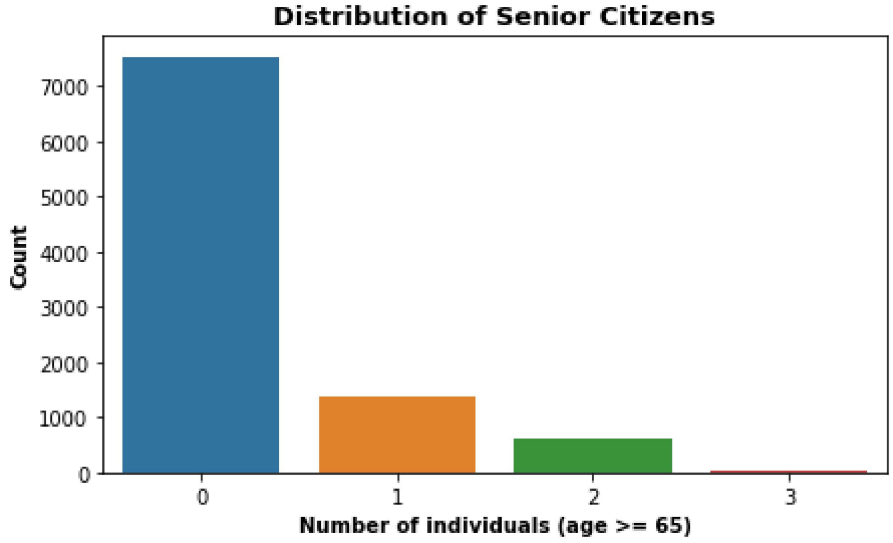
In [61]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'hogar_nin', data = train_data) # Number of children 0 to 19 in household
plt.title("Distribution of Children", fontsize = 13, fontweight="bold")
plt.xlabel("Children in house (0 <= age <= 19)", fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



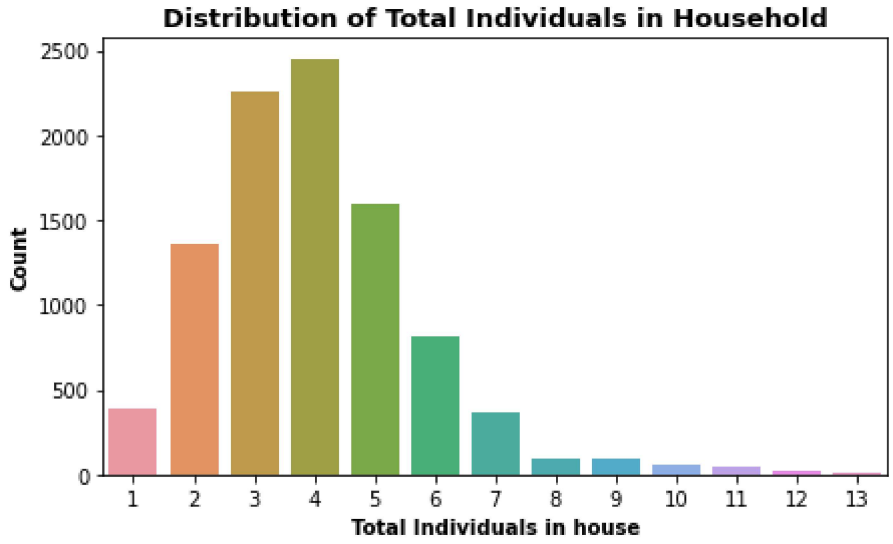
In [62]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'hogar_adul', data = train_data) # Number of adults in house hold
plt.title("Distribution of Adults in House", fontsize = 13, fontweight="bold")
plt.xlabel("Adults in house", fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



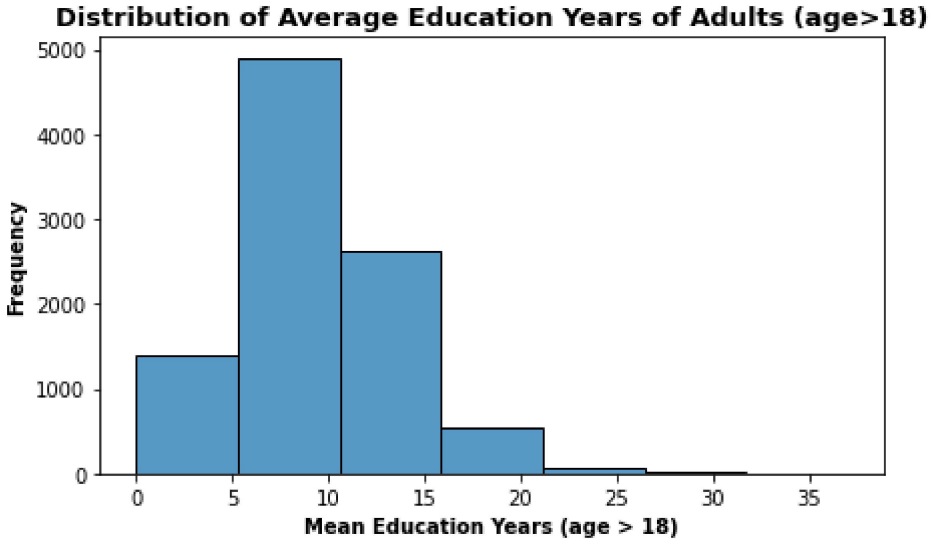
In [63]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'hogar_mayor', data = train_data)
plt.title("Distribution of Senior Citizens", fontsize = 13, fontweight="bold")
plt.xlabel("Number of individuals (age >= 65)", fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



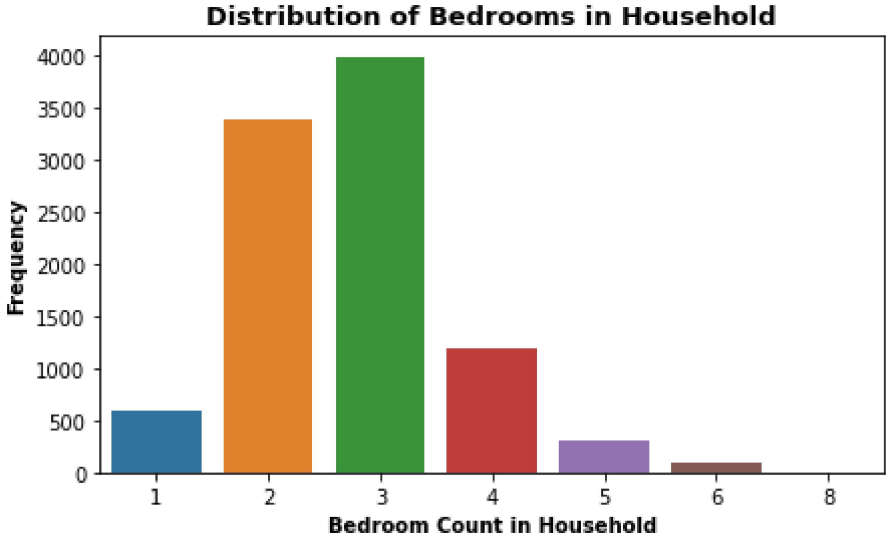
In [64]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'hogar_total', data = train_data ) # Number of total individ
uals in the household
plt.title("Distribution of Total Individuals in Household", fontsize = 13, fon
tweight="bold")
plt.xlabel("Total Individuals in house", fontweight="bold")
plt.ylabel('Count', fontweight="bold")
plt.show()
```



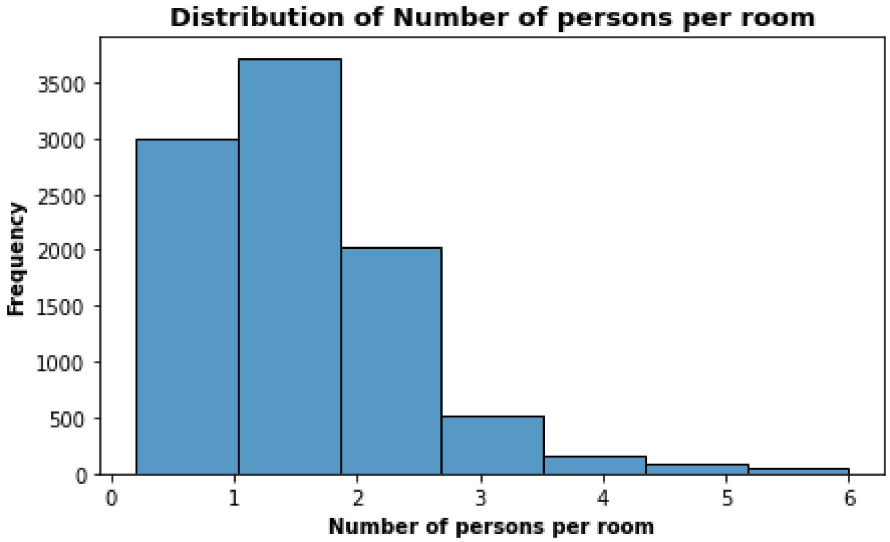
In [65]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'meaneduc', data = train_data, bins=7) # average years of edu
cation for adults (18+)
plt.title("Distribution of Average Education Years of Adults (age>18)", fontsi
ze = 13, fontweight="bold")
plt.xlabel("Mean Education Years (age > 18)", fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



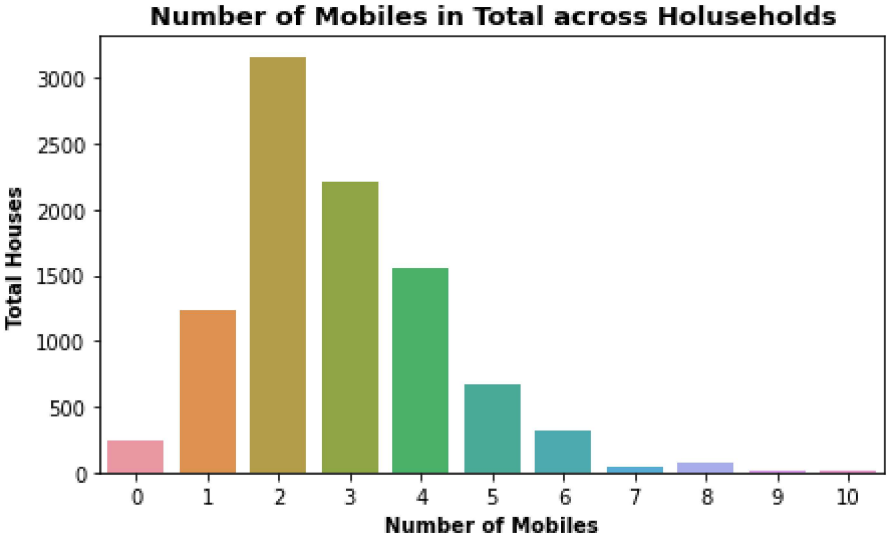
In [66]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'bedrooms', data = train_data) # number of bedrooms
plt.title("Distribution of Bedrooms in Household", fontsize = 13, fontweight=
"bold")
plt.xlabel("Bedroom Count in Household", fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



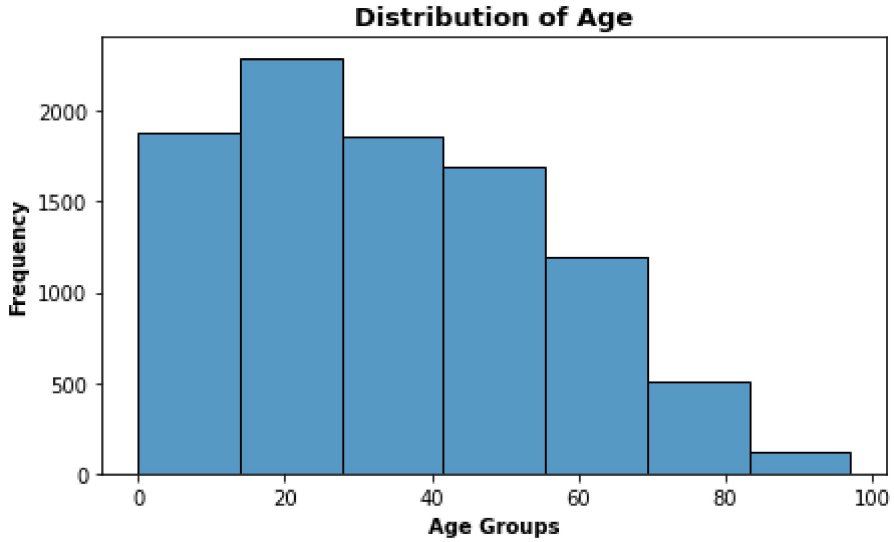
In [67]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'overcrowding' , data = train_data, bins = 7) # Number of persons per room
plt.title("Distribution of Number of persons per room", fontsize = 13, fontweight="bold")
plt.xlabel("Number of persons per room", fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



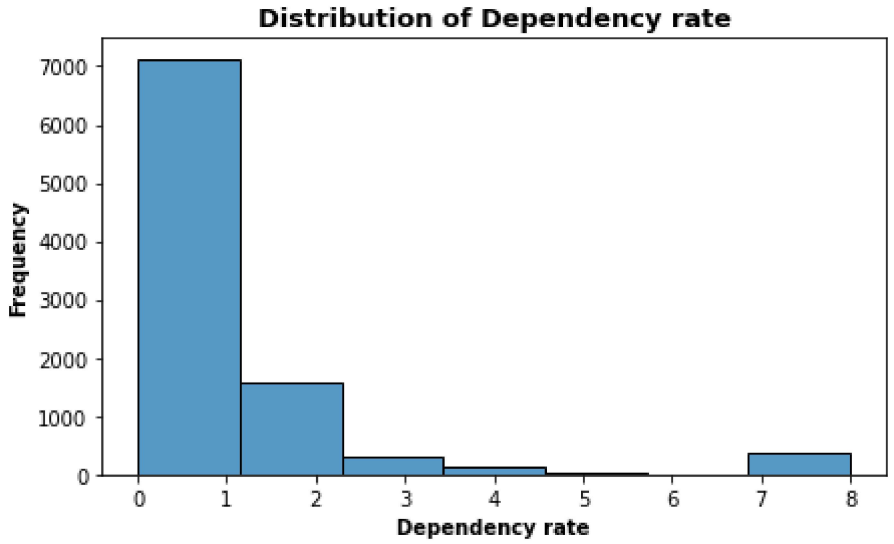
In [68]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'qmobilephone' , data = train_data)
plt.title("Number of Mobiles in Total across Holuseholds", fontsize = 13, fontweight="bold")
plt.xlabel("Number of Mobiles", fontweight="bold")
plt.ylabel('Total Houses', fontweight="bold")
plt.show()
```



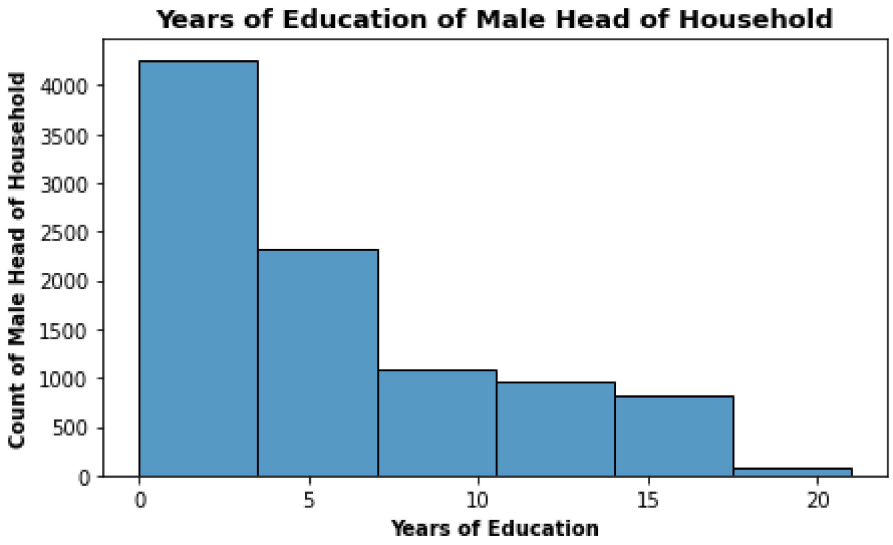
In [69]:
```
plt.figure(figsize=(7,4))
sns.histplot(x =  'age', data = train_data, bins = 7) # Age in years
plt.title("Distribution of Age", fontsize = 13, fontweight="bold")
plt.xlabel("Age Groups", fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



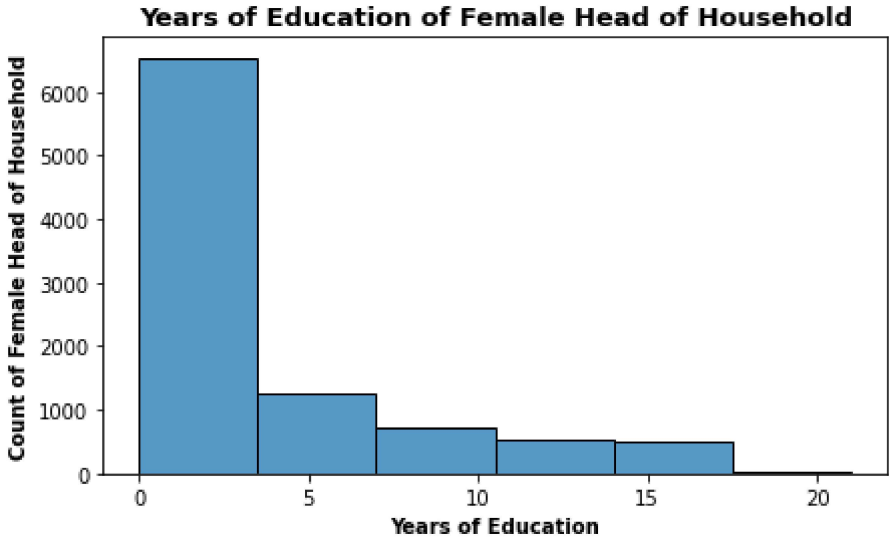
In [70]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'dependency', data = train_data, bins = 7);
plt.title('Distribution of Dependency rate', fontsize = 13, fontweight="bold")
plt.xlabel('Dependency rate', fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [71]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'edjefe', data = train_data, bins = 6)  # years of education
 of male head of household
plt.title('Years of Education of Male Head of Household', fontsize = 13, fontw
eight="bold")
plt.xlabel('Years of Education', fontweight="bold")
plt.ylabel('Count of Male Head of Household', fontweight="bold")
plt.show()
```



In [72]:
```
plt.figure(figsize=(7,4))
sns.histplot(x = 'edjefa', data = train_data, bins = 6) # years of education o
f female head of household
plt.title('Years of Education of Female Head of Household', fontsize = 13, fon
tweight="bold")
plt.xlabel('Years of Education', fontweight="bold")
plt.ylabel('Count of Female Head of Household', fontweight="bold")
plt.show()
```

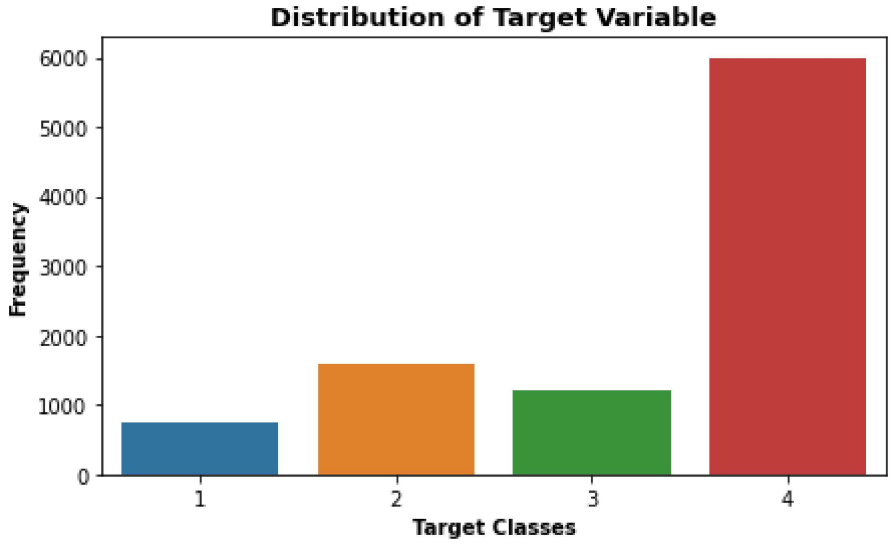


Sub-Task: Analysing the target column

In [73]:
```
train_data['Target'].value_counts() #
```

Out[73]:
```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```

In [74]:
```
plt.figure(figsize=(7,4))
sns.countplot(x = 'Target', data = train_data) # Target Variable
plt.title("Distribution of Target Variable", fontsize = 13, fontweight="bold")
plt.xlabel("Target Classes", fontweight="bold")
plt.ylabel('Frequency', fontweight="bold")
plt.show()
```



In [75]:
```
round(train_data['Target'].value_counts()/train_data.shape[0]*100, 2) # percen
t of target classes present in dataset
```

Out[75]:
```
4    62.74
2    16.71
3    12.65
1     7.90
Name: Target, dtype: float64
```

In [76]:
```
print('Based on the above output, we see that class 4 dominates and is visible
for more then half the number of \
      other classes. Classes 2 and 3 have relatively similar percent presence,
however class 1 has lowest presence across \
      the dataset')
```

```
Based on the above output, we see that class 4 dominates and is visible for m
ore then half the number of       other classes. Classes 2 and 3 have relativ
ely similar percent presence, however class 1 has lowest presence across
the dataset
```

SubTask: Analysing Null values and Treatment

In [77]:
```
train_data.isnull().sum()[train_data.isnull().sum() > 0]
```

Out[77]:
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

In [78]:
```
test_data.isnull().sum()[test_data.isnull().sum() > 0]
```

Out[78]:
```
v2a1         17403
v18q1        18126
rez_esc      19653
meaneduc        31
SQBmeaned       31
dtype: int64
```

In [79]:
```
train_data.isnull().sum()[train_data.isnull().sum() > 0]/train_data.shape[0]*1
00 # % of null values across columns (train_data)
```

Out[79]:
```
v2a1         71.779847
v18q1        76.823271
rez_esc      82.954902
meaneduc      0.052318
SQBmeaned     0.052318
dtype: float64
```

In [80]:
```
test_data.isnull().sum()[test_data.isnull().sum() > 0]/test_data.shape[0]*100
# % of null values across columns (test_data)
```

Out[80]:
```
v2a1         72.950201
v18q1        75.980885
rez_esc      82.381791
meaneduc      0.129946
SQBmeaned     0.129946
dtype: float64
```

In [81]:
```
# dropping the null values column, as they have more then 70% datapoints missi
ng

train_data.drop(['v2a1','v18q1','rez_esc'], axis=1, inplace=True)
test_data.drop(['v2a1','v18q1','rez_esc'], axis=1, inplace=True)
```

In [82]:
```
train_data[['meaneduc', 'SQBmeaned']].describe()
```

Out[82]:

	meaneduc	SQBmeaned
count	9552.000000	9552.000000
mean	9.231523	102.588867
std	4.167694	93.516890
min	0.000000	0.000000
25%	6.000000	36.000000
50%	9.000000	81.000000
75%	11.600000	134.560010
max	37.000000	1369.000000

In [83]:
```
test_data[['meaneduc', 'SQBmeaned']].describe()
```

Out[83]:

	meaneduc	SQBmeaned
count	23825.000000	23825.000000
mean	9.157474	100.509220
std	4.080513	89.211063
min	0.000000	0.000000
25%	6.000000	36.000000
50%	8.666667	75.111115
75%	11.500000	132.250000
max	36.000000	1296.000000

In [84]:
```
# Imputing the median values of 'meaneduc', 'SQBmeaned' from train data, into
 respective columns missing points of
# train and test dataset

train_data[['meaneduc', 'SQBmeaned']] = train_data[['meaneduc', 'SQBmeaned']].
fillna(
                                                                        tr
ain_data[['meaneduc', 'SQBmeaned']].median())

test_data[['meaneduc', 'SQBmeaned']] = test_data[['meaneduc', 'SQBmeaned']].fi
llna(
                                                                        tr
ain_data[['meaneduc', 'SQBmeaned']].median())
```

In [85]:
```
train_data.isnull().sum()[train_data.isnull().sum() > 0]
```

Out[85]:
```
Series([], dtype: int64)
```

In [86]:
```
test_data.isnull().sum()[test_data.isnull().sum() > 0]
```

Out[86]:
```
Series([], dtype: int64)
```

In [87]:
```
train_data.duplicated().sum()
```

Out[87]:
```
0
```

In [88]:
```
test_data.duplicated().sum()
```

Out[88]:
```
0
```

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

In [89]:
```
# checking if the household identifier is unique across all the rows

train_data['idhogar'].nunique() == train_data.shape[0]
```

Out[89]:
```
False
```

In [90]:
```
train_data['idhogar'].head()
```

Out[90]:
```
0    21eb7fcc1
1    0e5d7a658
2    2c7317ea8
3    2b58d945f
4    2b58d945f
Name: idhogar, dtype: object
```

In [91]:
```
train_data.groupby('idhogar').get_group('2b58d945f') # getting the members of
 the group '2b58d945f'
```

Out[91]:

	Id	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	r4
3	ID_d671db89c	0	5	0	1	1	1	0	2	2	1	1	
4	ID_d56d6f5f5	0	5	0	1	1	1	0	2	2	1	1	
5	ID_ec05b1a7b	0	5	0	1	1	1	0	2	2	1	1	
6	ID_e9e0c1100	0	5	0	1	1	1	0	2	2	1	1	

In [92]:
```
print(f"There are { train_data.shape[0] - train_data['idhogar'].nunique()} hou
ses which has more then one entry in the dataset")
```

```
There are 6569 houses which has more then one entry in the dataset
```

In [93]:
```
idhogar_freq = train_data['idhogar'].value_counts()
items = idhogar_freq[idhogar_freq>1].index # items that appear more than once
items
#train_data[train_data['idhogar'].isin(items)]
```

Out[93]:
```
Index(['fd8a6d014', 'ae6cf0558', '0c7436de6', '3fe29a56b', '6b35cdcf0',
       'b7a0b59d7', '4476ccd4c', '0fc6c05f7', '63f11d6ea', '7cad2d6c4',
       ...
       'c3526ec6b', '17e26c0f0', 'bd82509d1', '47636186d', 'b2654c83b',
       'd2dfde10d', '749b787e4', '7d184f883', '9b2fb9b5c', '5fee0c9fa'],
      dtype='object', length=2590)
```

In [94]:
```
train_data[['idhogar', 'Target']].groupby('idhogar').filter(lambda x: (x['Target'].nunique()>1)).drop_duplicates().head(7)
```

Out[94]:

	idhogar	Target
282	4b6077882	1
283	4b6077882	2
285	6833ac5dc	2
290	6833ac5dc	1
318	43b9c83e5	2
322	43b9c83e5	1
406	5c3f7725d	3

In [95]:
```
houses_with_diff_poverty_levels = train_data[['idhogar', 'Target']].groupby('idhogar').filter(lambda x:

(x['Target'].nunique()>1))
houses_with_diff_poverty_levels = houses_with_diff_poverty_levels.drop_duplicates()
houses_with_diff_poverty_levels.shape
```

Out[95]:
```
(171, 2)
```

In [96]:
```
print(f'All the members of the house do not have same poverty level. \
      There are {houses_with_diff_poverty_levels.shape[0]} such houses which has different poverty level for same house ID.')
```

```
All the members of the house do not have same poverty level.       There are
171 such houses which has different poverty level for same house ID.
```

Task 5: Check if there is a house without a family head.

In [97]:
```
# parentesco1, =1 if household head

# the below code removes the duplicate entries of the house idetifier and than
checks for the presence of head or not.

house_without_head = train_data[['idhogar', 'parentesco1']].drop_duplicates()[
'parentesco1'].value_counts()
house_without_head
```

Out[97]:
```
1    2973
0    2600
Name: parentesco1, dtype: int64
```

In [98]:
```
print(f"There are {house_without_head[0]} houses without a head.")
```

```
There are 2600 houses without a head.
```

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

In [99]:
```
# Below is the code where data is grouped by house identifier and the mean is
 calculated for different values of target across
# same house identifiers

poverty_level_data = train_data[['idhogar', 'Target']].groupby('idhogar')['Tar
get'].mean()
poverty_level_data.head()
```

Out[99]:
```
idhogar
001ff74ca    4.0
003123ec2    2.0
004616164    2.0
004983866    3.0
005905417    2.0
Name: Target, dtype: float64
```

In [100]:
```
poverty_level_data.dtype
```

Out[100]:
```
dtype('float64')
```

In [101]:
```
poverty_level_data = pd.DataFrame(poverty_level_data)
poverty_level_data.head()
```

Out[101]:

	Target
idhogar	
001ff74ca	4.0
003123ec2	2.0
004616164	2.0
004983866	3.0
005905417	2.0

In [102]:
```
poverty_level_data = poverty_level_data.reset_index().rename(columns={"Target"
: "Target_Mean"})
poverty_level_data.head()
```

Out[102]:

	idhogar	Target_Mean
0	001ff74ca	4.0
1	003123ec2	2.0
2	004616164	2.0
3	004983866	3.0
4	005905417	2.0

In [103]:
```
poverty_level_data['Target_Mean'].unique()
```

Out[103]:
```
array([4.        , 2.        , 3.        , 1.        , 2.6       ,
       1.33333333, 3.75      , 1.5       , 1.75      , 1.66666667,
       2.2       , 3.33333333, 2.75      , 1.83333333, 2.5       ,
       2.33333333, 3.5       , 2.63636364, 1.8       , 2.66666667,
       2.8       , 3.66666667, 2.71428571, 1.6       , 2.25      ,
       3.8       , 1.25      , 2.4       ])
```

In [104]:
```
copy_train_data = train_data.copy(deep=True)
copy_train_data.head()
```

Out[104]:

	Id	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	r4
0	ID_279628684	0	3	0	1	1	0	0	1	1	0	0	
1	ID_f29eb3ddd	0	4	0	1	1	1	0	1	1	0	0	
2	ID_68de51c94	0	8	0	1	1	0	0	0	0	0	1	
3	ID_d671db89c	0	5	0	1	1	1	0	2	2	1	1	
4	ID_d56d6f5f5	0	5	0	1	1	1	0	2	2	1	1	

In [105]:
```
copy_train_data = copy_train_data.merge(poverty_level_data, how = 'left', on =
'idhogar')
```

In [106]:
```
copy_train_data[['idhogar', 'Target', 'Target_Mean']].head()
```

Out[106]:

	idhogar	Target	Target_Mean
0	21eb7fcc1	4	4.0
1	0e5d7a658	4	4.0
2	2c7317ea8	4	4.0
3	2b58d945f	4	4.0
4	2b58d945f	4	4.0

In [107]:
```
print('Above is dataset obtained after the requested operation of setting the
 poverty level across same houses \
       with different poverty levels')
```

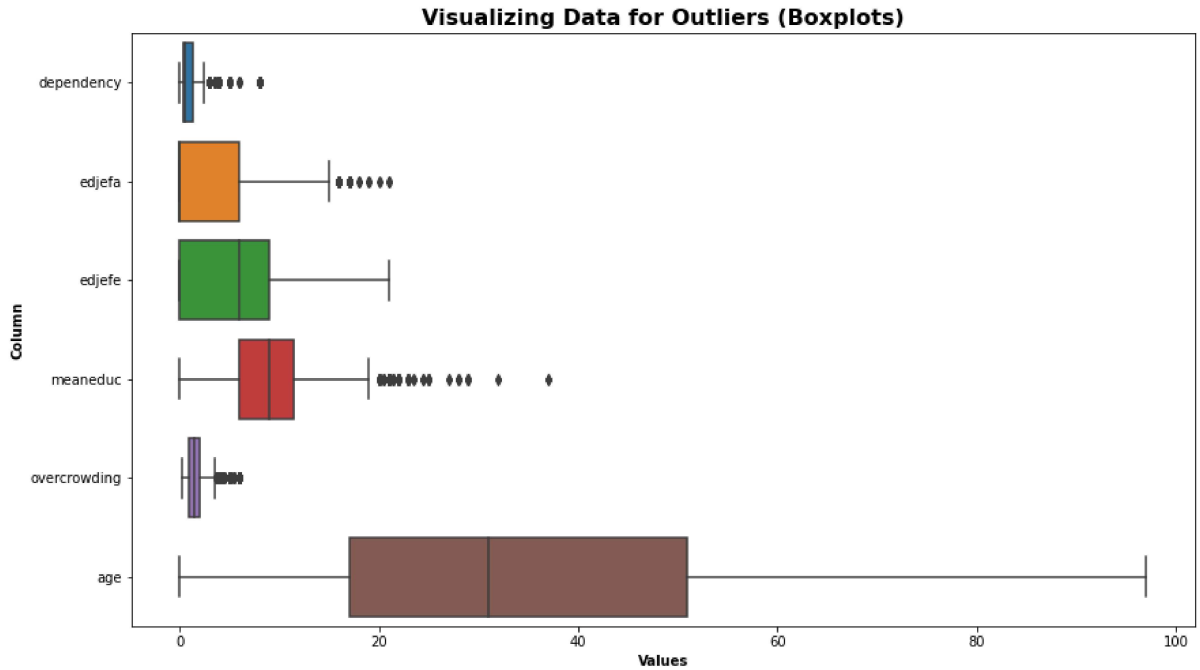
```
Above is dataset obtained after the requested operation of setting the povert
y level across same houses         with different poverty levels
```

In [108]:
```
# Final validation if the nulls were created during the join.

copy_train_data[['idhogar', 'Target', 'Target_Mean']].isnull().any()
```

Out[108]:
```
idhogar        False
Target         False
Target_Mean    False
dtype: bool
```

Sub-Task: Outliar Detection

In [109]:
```
plt.figure(figsize=(14,8))

sns.boxplot(data = train_data[['dependency', 'edjefa', 'edjefe', 'meaneduc','o
vercrowding', 'age' ]], orient='h');
plt.title('Visualizing Data for Outliers (Boxplots)', fontsize = 16, fontweigh
t="bold")
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



In [110]:
```
def outliar_removal(col, df):

    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)

#     print(f'For column {col}:\n')
#     print(f'25th Percentile: {percentile25}')
#     print(f'75th Percentile: {percentile75}')

    iqr = percentile75 - percentile25

#     print(f'IQR: {iqr}\n')

    upper_limit = percentile75 + (1.5 * iqr)
    lower_limit = percentile25 - (1.5 * iqr)

#     print('Before: {df.shape}')

    df = df[df[col] < upper_limit]
    df = df[df[col] > lower_limit]
#     print('After: {df.shape}')

    return df
```

In [111]:
```
# shape before outlier removal

before_outliar_removal = train_data.shape
before_outliar_removal
```

Out[111]: (9557, 140)

In [112]:
```
for col in ['dependency', 'edjefa', 'meaneduc','overcrowding']:
    train_data = outliar_removal(col, train_data)
```

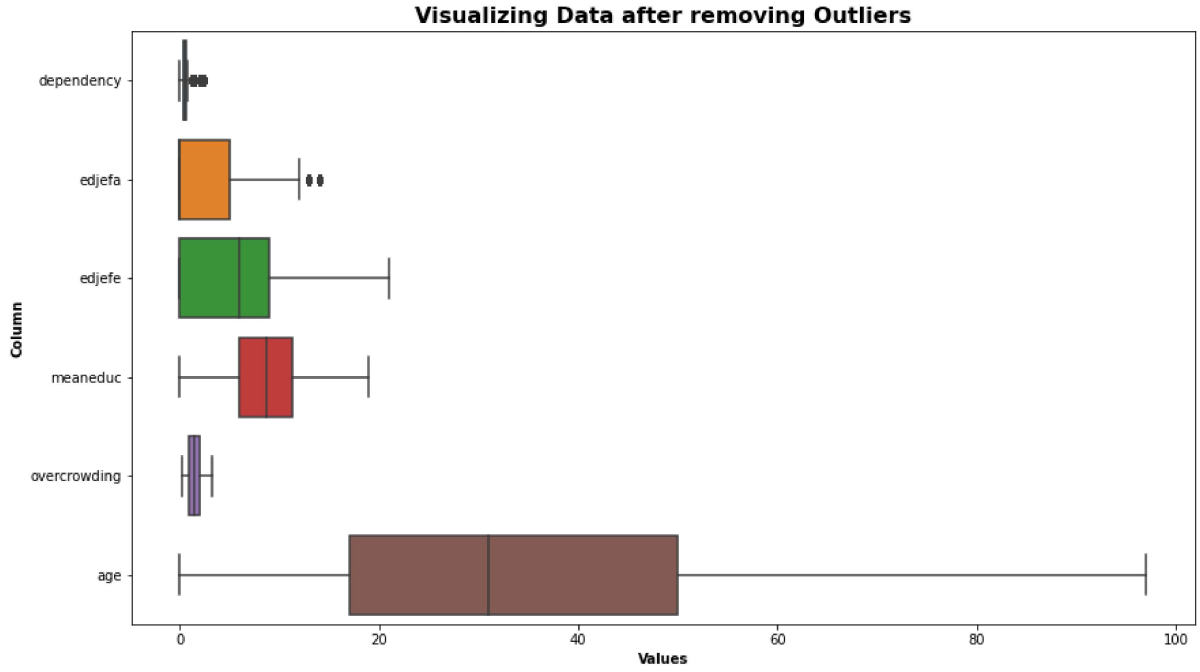
In [113]:
```
# shape after outlier removal

after_outliar_removal = train_data.shape
after_outliar_removal
```

Out[113]: (8057, 140)

In [114]:
```
# percent value of outliers removed

percent_outliers_removed = ((before_outliar_removal[0] - after_outliar_removal
[0])/before_outliar_removal[0])*100
print(f'Percent of Outliers removed from train dataset: {round(percent_outlier
s_removed,2)}%')
```

```
Percent of Outliers removed from train dataset: 15.7%
```

In [115]:
```
plt.figure(figsize=(14,8))

sns.boxplot(data = train_data[['dependency', 'edjefa', 'edjefe', 'meaneduc','o
vercrowding', 'age' ]], orient='h');
plt.title('Visualizing Data after removing Outliers', fontsize = 16, fontweigh
t="bold")
plt.xlabel('Values', fontweight="bold")
plt.ylabel('Column', fontweight="bold")
plt.show()
```



Sub-Task: Splitting and Standardization of the dataset

In [116]:
```
# removing 'ID' and 'idhogar' columns as they are unique identifiers and

train_data.drop(['Id', 'idhogar'], axis = 1, inplace=True)
test_data.drop(['Id', 'idhogar'], axis = 1, inplace=True)
```

In [117]:
```
X = train_data.drop(['Target'], axis = 1)
y = train_data['Target']
```

Sub-Task: Splitting the data into training and testing set

In [118]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [119]:
```
non_categ_cols = [col for col in X_train.columns if ~X_train[col].isin([0,1]).
all()]
non_categ_cols
```

Out[119]:
```
['rooms',
 'r4h1',
 'r4h2',
 'r4h3',
 'r4m1',
 'r4m2',
 'r4m3',
 'r4t1',
 'r4t2',
 'r4t3',
 'tamhog',
 'tamviv',
 'escolari',
 'hhsize',
 'hogar_nin',
 'hogar_adul',
 'hogar_mayor',
 'hogar_total',
 'dependency',
 'edjefe',
 'edjefa',
 'meaneduc',
 'bedrooms',
 'overcrowding',
 'qmobilephone',
 'age',
 'SQBescolari',
 'SQBage',
 'SQBhogar_total',
 'SQBedjefe',
 'SQBhogar_nin',
 'SQBovercrowding',
 'SQBdependency',
 'SQBmeaned',
 'agesq']
```

Sub-Task: Scaling the data

In [120]:
```
scaler = MinMaxScaler()
X_train[non_categ_cols] = scaler.fit_transform(X_train[non_categ_cols])
X_test[non_categ_cols] = scaler.transform(X_test[non_categ_cols])
```

In [121]:
```
X_train.head()
```

Out[121]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	
6104	0	0.5	0	1	1	1	0.00	0.142857	0.142857	0.00	0.333333	0.
9303	0	0.2	0	1	1	0	0.25	0.142857	0.285714	0.00	0.166667	0.
8365	0	0.3	0	1	1	0	0.25	0.285714	0.428571	0.00	0.166667	0.
4156	0	0.2	0	1	1	0	0.00	0.142857	0.142857	0.00	0.333333	0.
8142	0	0.2	0	1	1	0	0.00	0.142857	0.142857	0.25	0.166667	0.

In [122]:
```
X_test.head()
```

Out[122]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	
9246	0	0.5	0	1	1	0	0.25	0.285714	0.428571	0.25	0.166667	0.
9101	0	0.4	0	1	0	0	0.00	0.285714	0.285714	0.00	0.500000	0.
188	0	0.3	1	1	1	0	0.75	0.285714	0.714286	0.25	0.500000	0.
7235	0	0.4	0	1	1	0	0.25	0.142857	0.285714	0.00	0.166667	0.
383	0	0.7	0	1	1	1	0.00	0.142857	0.142857	0.25	0.500000	0.

In [123]:
```
test_data.head()
```

Out[123]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	r4m3	r4t1	r4t2
0	0	5	0	1	1	0	1	1	2	0	1	1	1	2
1	0	5	0	1	1	0	1	1	2	0	1	1	1	2
2	0	5	0	1	1	0	1	1	2	0	1	1	1	2
3	0	14	0	1	1	1	0	1	1	0	0	0	0	1
4	0	4	0	1	1	1	0	0	0	0	1	1	0	1

In [124]:
```
# scaling the test data provided

test_data[non_categ_cols] = scaler.transform(test_data[non_categ_cols])
test_data.head()
```

Out[124]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	r4m2	r4
0	0	0.4	0	1	1	0	0.25	0.142857	0.285714	0.0	0.166667	0.142
1	0	0.4	0	1	1	0	0.25	0.142857	0.285714	0.0	0.166667	0.142
2	0	0.4	0	1	1	0	0.25	0.142857	0.285714	0.0	0.166667	0.142
3	0	1.3	0	1	1	1	0.00	0.142857	0.142857	0.0	0.000000	0.000
4	0	0.3	0	1	1	1	0.00	0.000000	0.000000	0.0	0.166667	0.142

Sub-Task: Finding Principle Components to reduce dataset Dimention (PCA)

In [125]:
```
X_train_covariance_matrix = np.cov(X_train.T)
```

In [126]:
```
X_train_covariance_matrix.shape
```

Out[126]:
```
(137, 137)
```

In [127]:
```
eig_vals, eig_vecs = np.linalg.eig(X_train_covariance_matrix)
```

In [128]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals
))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)
```

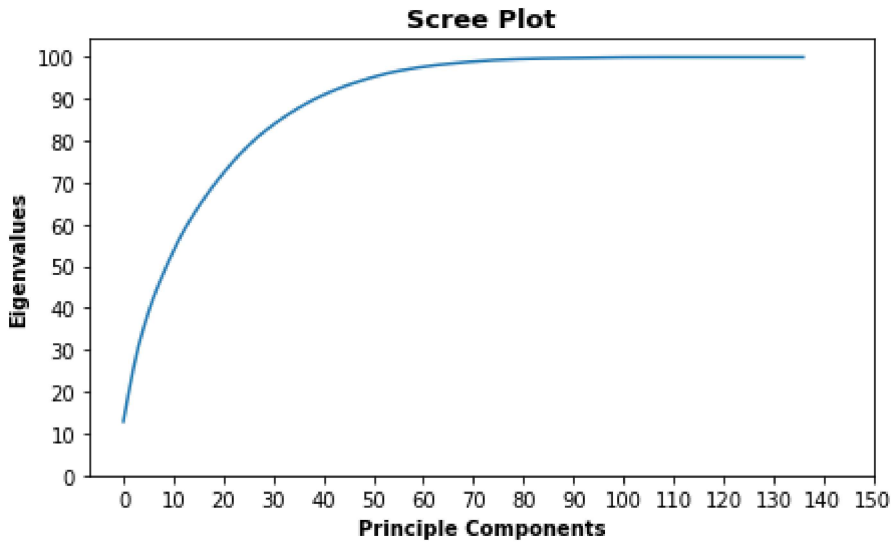
In [129]:
```
# Visually confirm that the list is correctly sorted by decreasing eigenvalues
print('Eigenvalues in descending order: \n')
for i in eig_pairs[:5]:   # first 5 values
    print(i[0])
```

```
Eigenvalues in descending order:

1.3369442663318214
0.6840499321486111
0.6284526439732685
0.5549560771994041
0.43858305816690824
```

In [130]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]  # Variance
 captured by each component
cum_var_exp = np.cumsum(var_exp)

# Since the cumulative variance is in complex form,
cum_var_exp = np.round(np.real(cum_var_exp),2)

print("Cumulative variance (first 5 values) \n\n", cum_var_exp[:5])
```

```
Cumulative variance (first 5 values)

 [12.88 19.47 25.52 30.87 35.1 ]
```

In [131]:
```
plt.figure(figsize=(7,4))
sns.lineplot(x = range(X_train.shape[1]), y = cum_var_exp);
plt.title('Scree Plot', fontsize = 13, fontweight="bold")
plt.xlabel('Principle Components', fontweight="bold")
plt.ylabel('Eigenvalues', fontweight="bold")

plt.xticks(range(0, 151, 10))
plt.yticks(range(0, 101, 10))

plt.show()
```



In [132]:
```
np.where(cum_var_exp>98)[0][0]  # First index of Principle component which has
captured upto 98% variance
```

Out[132]: 62

In [133]:
```
# defining PCA with 62 components (62 value is obtained from scree plot and ab
ove code) to capture 98% variance
pca = PCA(n_components=62)
```

In [134]:
```
X_train_pc = pca.fit_transform(X_train)
```

In [135]:
```
X_test_pc = pca.transform(X_test)
```

In [136]:
```
test_data_pc = pca.transform(test_data)
```

In [137]:
```
X_train.shape, X_train_pc.shape
```

Out[137]: ((6445, 137), (6445, 62))

In [138]:
```
X_test.shape, X_test_pc.shape
```

Out[138]: ((1612, 137), (1612, 62))

In [139]:
```
test_data.shape, test_data_pc.shape
```

Out[139]: ((23856, 137), (23856, 62))

Task 9: Predict the accuracy using random forest classifier.

In [140]:
```
random_forest_clf = RandomForestClassifier(n_estimators=100, max_depth=7, oob_
score=True)
```

In [141]:
```
random_forest_clf.fit(X_train_pc, y_train)
```

Out[141]: RandomForestClassifier(max_depth=7, oob_score=True)

In [142]:
```
random_forest_clf.oob_score_
```

Out[142]: 0.6637703646237393

In [143]:
```
y_pred_train = random_forest_clf.predict(X_train_pc)
y_pred_test = random_forest_clf.predict(X_test_pc)
```

In [144]:
```
print(f'Training Accuracy: {round((accuracy_score(y_train, y_pred_train)*100),
2)}%')
print(f'Test Accuracy: {round((accuracy_score(y_test, y_pred_test)*100),2)}%')
```

```
Training Accuracy: 69.7%
Test Accuracy: 66.94%
```

Sub-Task: Hyperparameter tuning for Random Forest using GridSearchCV

In [145]:
```
rf = RandomForestClassifier(n_jobs=-1)

params = {
    'max_depth': [10, 20, 25, 30, 35, 40],
    'min_samples_leaf': [2,5,7,9,15],
    'n_estimators': [150, 200, 250, 300, 350]
}
```

In [146]:
```
# Instantiate the grid search model

grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 5,
                           n_jobs=-1, verbose=2, scoring="accuracy", return_tr
ain_score=True)
```

In [147]:
```
grid_search.fit(X_train_pc, y_train)
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
```

Out[147]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(n_jobs=-1), n_jobs=-1,
             param_grid={'max_depth': [10, 20, 25, 30, 35, 40],
                         'min_samples_leaf': [2, 5, 7, 9, 15],
                         'n_estimators': [150, 200, 250, 300, 350]},
             return_train_score=True, scoring='accuracy', verbose=2)
```

In [148]:
```
grid_search.cv_results_.keys()
```

Out[148]:
```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
e', 'param_max_depth', 'param_min_samples_leaf', 'param_n_estimators', 'param
s', 'split0_test_score', 'split1_test_score', 'split2_test_score', 'split3_te
st_score', 'split4_test_score', 'mean_test_score', 'std_test_score', 'rank_te
st_score', 'split0_train_score', 'split1_train_score', 'split2_train_score',
'split3_train_score', 'split4_train_score', 'mean_train_score', 'std_train_sc
ore'])
```

In [149]:
```
grid_search.cv_results_['mean_train_score'][:10]
```

Out[149]:
```
array([0.80570209, 0.80810706, 0.80713732, 0.80771916, 0.80671063,
       0.76978278, 0.7704422 , 0.77067494, 0.77079131, 0.76982157])
```

In [150]:
```
grid_cv_scores = pd.DataFrame([grid_search.cv_results_['mean_train_score'], gr
id_search.cv_results_['mean_test_score']]).T
grid_cv_scores.columns = ['mean_train_score', 'mean_test_score']
grid_cv_scores.head(7)
```

Out[150]:

	mean_train_score	mean_test_score
0	0.805702	0.691699
1	0.808107	0.692320
2	0.807137	0.693716
3	0.807719	0.692940
4	0.806711	0.692785
5	0.769783	0.682545
6	0.770442	0.681458

In [151]:
```
grid_search.best_score_
```

Out[151]:
```
0.7478665632273079
```

In [152]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'max_depth': 30, 'min_samples_leaf': 2, 'n_estimators': 300}
```

In [153]:
```
rf_best = grid_search.best_estimator_
rf_best
```

Out[153]:
```
RandomForestClassifier(max_depth=30, min_samples_leaf=2, n_estimators=300,
                       n_jobs=-1)
```

Sub-Task: Using best model obtained from Gridsearch for predictions

In [154]:
```
rf_best.fit(X_train_pc, y_train)

y_pred_train = rf_best.predict(X_train_pc)
y_pred_test = rf_best.predict(X_test_pc)
```

Sub-Task: Analysing Test Results

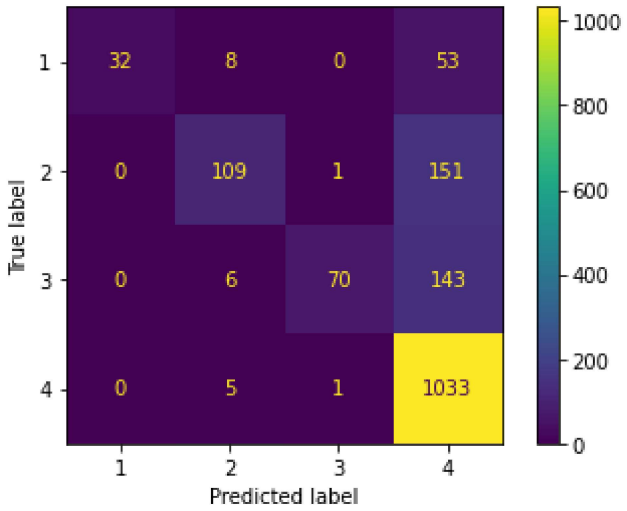
In [155]:
```
print(f'Training Accuracy: {round((accuracy_score(y_train, y_pred_train)*100),
2)}%')
print(f'Test Accuracy: {round((accuracy_score(y_test, y_pred_test)*100),2)}%')
```

```
Training Accuracy: 99.52%
Test Accuracy: 77.17%
```

In [156]:
```
print(confusion_matrix(y_test, y_pred_test))
```

```
[[  32    8    0   53]
 [   0  109    1  151]
 [   0    6   70  143]
 [   0    5    1 1033]]
```

In [157]:
```
cm = confusion_matrix(y_test, y_pred_test, labels=rf_best.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_best.clas
ses_)
disp.plot()
plt.show()
```



In [158]:
```
print(multilabel_confusion_matrix(y_test, y_pred_test))
```

```
[[[1519    0]
  [  61   32]]

 [[1332   19]
  [ 152  109]]

 [[1391    2]
  [ 149   70]]

 [[ 226  347]
  [   6 1033]]]
```

In [159]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           1       1.00      0.34      0.51        93
           2       0.85      0.42      0.56       261
           3       0.97      0.32      0.48       219
           4       0.75      0.99      0.85      1039

    accuracy                           0.77      1612
   macro avg       0.89      0.52      0.60      1612
weighted avg       0.81      0.77      0.74      1612
```

Task 10: Check the accuracy using random forest with cross validation.

In [160]:
```
%%time

# rf_best = RandomForestClassifier(max_depth=30, min_samples_leaf=2, n_estimat
ors=250, n_jobs=-1)

# Not defining model again because we already have it in 'rf_best' which was o
btained by gridsearch

k_folds = KFold(n_splits = 10)

scores = cross_val_score(rf_best, X, y, cv = k_folds)
```

```
CPU times: user 4.53 s, sys: 700 ms, total: 5.23 s
Wall time: 18 s
```

In [161]:
```
print("Cross Validation Scores: ", scores)
print("\nAverage CV Score: ", scores.mean())
print("\nNumber of CV Scores used in Average: ", len(scores))
```

```
Cross Validation Scores:  [0.79528536 0.73200993 0.74441687 0.68610422 0.6699
7519 0.64019851
 0.57444169 0.54782609 0.54409938 0.62236025]

Average CV Score:  0.6556717476072313

Number of CV Scores used in Average:  10
```

Sub-Task: Predicition of datapoints given in test.csv file

In [162]:
```
test_csv_pred = rf_best.predict(test_data_pc)
```

In [163]:
```
test_csv_pred = pd.DataFrame(test_csv_pred, columns=['predicted_values'])
test_csv_pred.head(7)
```

Out[163]:

	predicted_values
0	4
1	4
2	4
3	4
4	2
5	4
6	4

In [164]:
```
# exporting the test.csv predicted values as csv file

test_csv_pred.to_csv('test_data_pred.csv', index = False)
```

End of the Project