USE CASE

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.

Task:
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.

IMPORT LIBRARIES

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,classification_report


%matplotlib inline

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

Data Analysis (EDA)

In [2]:
```
data_train = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Machine Learning/Assessments/Income Qualification/train.csv")
data_train.head()
```

Out[2]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQB
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 [3]:
```
data_test = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Machine Learning/Assessments/Income Qualification/test.csv")
data_test.head()
```

Out[3]:

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

5 rows × 142 columns

In [4]:
```
print("Total records present in train data: ", data_train.shape, "\n")
print("Total records present in test data: ", data_test.shape, "\n")
```

```
Total records present in train data:  (9557, 143)

Total records present in test data:  (23856, 142)
```

Observation: We can analyze that the train data has **143 records**, however test data has only **142 records**. There is a missing column in the Test data. Lets verify the missing data in the Test Data

TASK 1: Identify the Output Variable

In [5]:
```
for i in data_train.columns:
    if i in data_test.columns:
        continue
    else:
        print("The column not present in Test Data is: ", i)
```

```
The column not present in Test Data is:  Target
```

Analysis: Since **Target** is not present in the Test Data, Hence it is the **Output Variable**

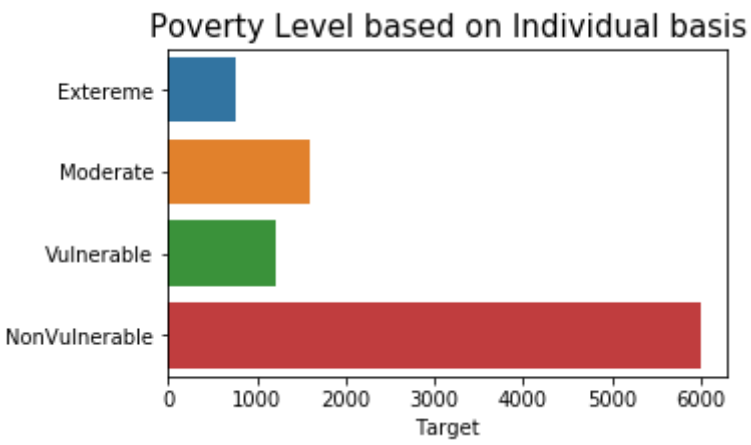
In [6]:
```
labels = ["Extereme", "Moderate", "Vulnerable", "NonVulnerable"]
povertyLevels = data_train['Target'].value_counts().to_frame().sort_index(ascending=True)
povertyLevels
```

Out[6]:

	Target
1	755
2	1597
3	1209
4	5996

In [7]:
```
plt.figure(figsize=(5,3))
ax = sns.barplot(y=labels,x=povertyLevels['Target'])
plt.tick_params(axis="x", labelsize = 10)
plt.title("Poverty Level based on Individual basis", fontdict = {'fontsize' : 15})
```

Out[7]: `Text(0.5, 1.0, 'Poverty Level based on Individual basis')`



TASK 2: Understand the type of data

In [8]:
```
data_train.info()
```

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

Train data has 3 different data type features:
Float : 8 Features
Int : 130 features
Object: 5 features

In [9]:
```
numericfeatures = [col for col in data_train.columns if (data_train[col].dtype == 'float64')
                   | (data_train[col].dtype == 'int64') & (col != 'Target')]
```

In [10]:
```
objectfeatures = [obj for obj in data_train.columns if data_train[obj].dtype == 'object']
objectfeatures
```

Out[10]: `['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa']`

Id: Unique ID
idhogar: Household level identifier
dependency: Dependency rate, calculated = (number of members of the household younger than 19 or older than 64)/(number of member of household between 19 and 64)
edjefe: years of education of male head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0
edjefa: years of education of female head of household, based on the interaction

Created array containing the Object features: **objectfeatures** and Numeric features: **numericfeatures**(except output variable: **Target**)

In [11]:
```
objectfeatures.remove("Id")
objectfeatures.remove("idhogar")
objectfeatures
```

Out[11]: `['dependency', 'edjefe', 'edjefa']`

In [12]:
```
# Finding Unique Values for Object Features

for col in data_train[objectfeatures]:
    print("Unique values for: ", col)
    print(data_train[col].unique())
    print("\n")
```

```
Unique values for:  dependency
['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']


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


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

Create and apply function to YES and NO with 1 and 0 in **object features**

In [13]:
```
def map(i):
    if i=='yes':
        return(float(1))
    elif i=='no':
        return(float(0))
    else:
        return(float(i))
```

In [14]:
```
for column in objectfeatures:
    data_train[column] = data_train[column].apply(map)
```

Verify if **YES & NO** is replaced with **1 & 0**

In [15]:
```
print("Unique values for edjefe: ", data_train['edjefe'].unique())
```

```
Unique values for edjefe:  [10. 12.  0. 11.  9. 15.  4.  6.  8. 17.  7. 16. 14.  5. 21.  2. 19.  1.
  3. 18. 13. 20.]
```

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

In [16]:
```
# Groupby the household and figure out the number of unique values
all_equal = data_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)

# Households where targets are not all equal
not_equal = all_equal[all_equal != True]
print('There are {} households where the family members do not all have the same target.'.format(len(not_equal)))
```

```
There are 85 households where the family members do not all have the same target.
```

In [17]:
```
data_train[data_train['idhogar'] == not_equal.index[0]][['idhogar', 'parentesco1', 'Target']]
```

Out[17]:

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

Analysis: We can see that the member of same family has different **Target labels.**
Since this cannot be true, hence we can say that few data are misrepresented. We will fix the Target Label, for misrepresented data's.

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

In [18]:
```
# Iterate through each household
for household in not_equal.index:
    # Find correct label for head of household
    true_target = int(data_train[(data_train['idhogar'] == household) & (data_train['parentesco1'] == 1.0)]['Target'])

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


# Groupby the household and figure out the number of unique values
all_equal = data_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)

# Households where targets are not all equal
not_equal = all_equal[all_equal != True]
print('There are {} households where the family members do not have the same target.'.format(len(not_equal)))
```

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

In [19]:
```
data_train.loc[data_train['idhogar'] == '0172ab1d9',['idhogar','Target']]
```

Out[19]:

	idhogar	Target
7651	0172ab1d9	3
7652	0172ab1d9	3
7653	0172ab1d9	3
7654	0172ab1d9	3
7655	0172ab1d9	3

We haved fixed the Taget Labels for the household now

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

Feature which tells us about the Family Head: parentesco1
parentesco1 = 1 (if household head)

In [20]:
```
data_train['parentesco1'].value_counts()
```

Out[20]:
```
0    6584
1    2973
Name: parentesco1, dtype: int64
```

Feature idhogar describes the unique id for each household. The unique count will give us information on how many households are present in the dataset.

In [21]:
```
data_train['idhogar'].value_counts()
```

Out[21]:
```
fd8a6d014    13
0c7436de6    12
ae6cf0558    12
4476ccd4c    11
6b35cdcf0    11
             ..
7c1cfa65c     1
188e498f3     1
223dab3ac     1
1bb8a3f9c     1
2e2acffa3     1
Name: idhogar, Length: 2988, dtype: int64
```

There are 2988 families/household discussed about in the data

In [22]:
```
household_head = data_train.groupby('idhogar')['parentesco1'].sum()
household_head.value_counts()
```

Out[22]:
```
1    2973
0      15
Name: parentesco1, dtype: int64
```

Analysis: There are 15 familes/househlds without a family head

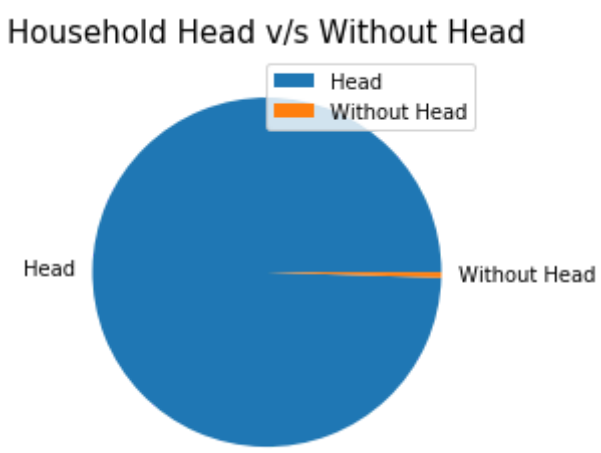
In [23]:
```
household_head_frame = household_head.value_counts().to_frame()
household_head_frame
```

Out[23]:

	parentesco1
1	2973
0	15

In [24]:
```
labels = ["Head", "Without Head"]
plt.tick_params(axis="y", labelsize = 10)
plt.pie(household_head_frame, labels=labels)
plt.title("Household Head v/s Without Head", fontdict = {'fontsize' : 15})
plt.legend()
```

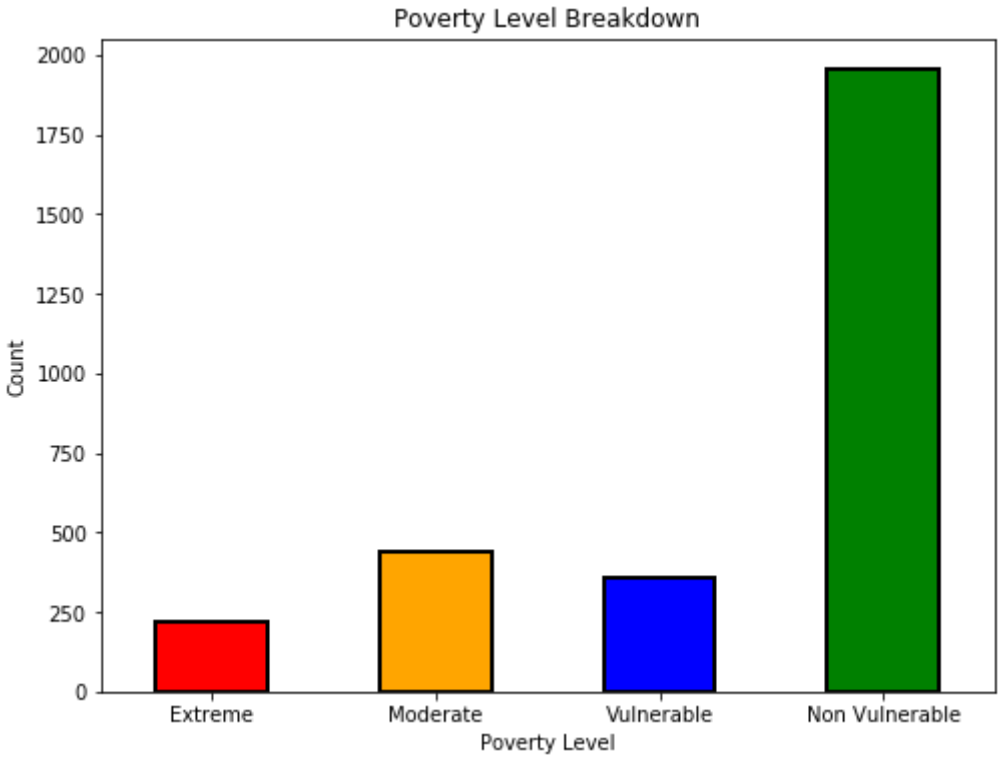
Out[24]: <matplotlib.legend.Legend at 0x20e08f8fac8>



TASK 3: Check if there are any biases in your dataset.

In [25]:
```
targetLabels = data_train.loc[(data_train['Target'].notnull()) & (data_train['parentesco1'] == 1), ['Target', 'idhoga
r']]
label_counts = targetLabels['Target'].value_counts().sort_index()

# Bar plot of occurrences of each label
from collections import OrderedDict
colors = OrderedDict({1: 'red', 2: 'orange', 3: 'blue', 4: 'green'})
poverty_mapping = OrderedDict({1: 'Extreme', 2: 'Moderate', 3: 'Vulnerable', 4: 'Non Vulnerable'})
label_counts.plot.bar(figsize = (8, 6),
                      color = colors.values(),
                      edgecolor = 'k', linewidth = 2)

# labels and titles
plt.xlabel('Poverty Level'); plt.ylabel('Count');
plt.xticks([x - 1 for x in poverty_mapping.keys()],
           list(poverty_mapping.values()), rotation = 360)
plt.title('Poverty Level Breakdown');

label_counts
```

Out[25]:
```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```



Analysis:We can see the data is Unbalanced. Hence we can confirm that there is biasness in the dataset

TASK 7: Count how many null values are existing in columns.

In [26]:
```
data_train[objectfeatures].isnull().any()
```

Out[26]:
```
dependency    False
edjefe        False
edjefa        False
dtype: bool
```

In [27]:
```
nullvalues = data_train[numericfeatures].isnull().sum()
nullvalues[nullvalues > 0]
```

Out[27]:
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

v2a1: Monthly Rent Payment
v18q1: Number of tablets household owns
rez esc: Years behind in school

In [28]:
```
data_train['Target'].isnull().any()
```

Out[28]:
```
False
```

TASK 8: Remove null value rows of the target variable.

Analysis: There is no null value present in the Target variable

Remove null value present in the v2a1feature variable

v2a1 is corelated with features like
tipovivi1 =1 own and fully paid house
tipovivi2 =1 own, paying in installments
tipovivi3 =1 rented
tipovivi4 =1 precarious
tipovivi5 =1 other(assigned, borrowed)

In [29]:
```
nullRentValue = data_train[data_train['v2a1'].isnull()]
nullRentValue.head()
```

Out[29]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBho
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	
13	ID_064b57869	NaN	0	4	0	1	1	1	1.0	0	...	16	6241	4	0	
14	ID_5c837d8a4	NaN	0	4	0	1	1	1	1.0	0	...	225	1521	4	0	
26	ID_e5cdba865	NaN	0	5	0	1	1	0	NaN	0	...	225	1936	1	225	
32	ID_e24d9c3c9	NaN	0	5	0	1	1	0	NaN	0	...	1	784	25	121	

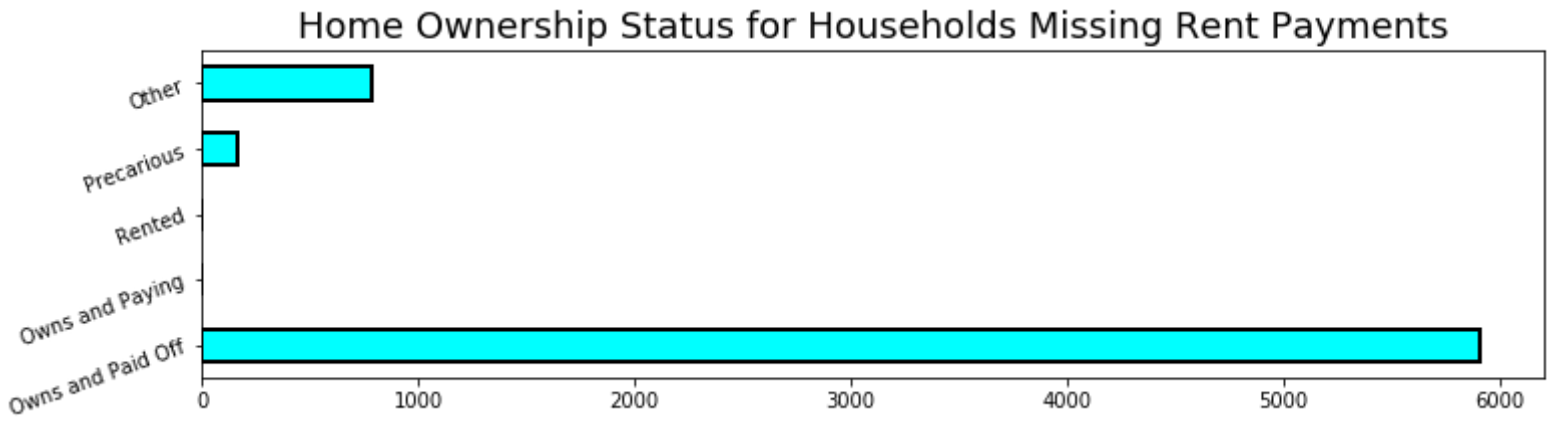
5 rows × 143 columns

In [30]:
```
rentDetailColumns = ['tipovivi1', 'tipovivi2', 'tipovivi3' , 'tipovivi4', 'tipovivi5']
nullRentValue[rentDetailColumns].head(3)
```

Out[30]:

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

In [31]:
```
rentDetails= [x for x in rentDetailColumns]

# Plot of the home ownership variables for home missing rent payments
data_train.loc[data_train['v2a1'].isnull(), rentDetails].sum().plot.barh(figsize = (12, 3),
                                                                         color = 'cyan',
                                                                       edgecolor = 'k', linewidth = 2);
plt.yticks([0, 1, 2, 3, 4],
           ['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precarious', 'Other'],
          rotation = 20)
plt.title('Home Ownership Status for Households Missing Rent Payments', size = 18);
```



Analysis: Looking at the above data it makes sense that when the house is fully paid, there will be no monthly rent payment.

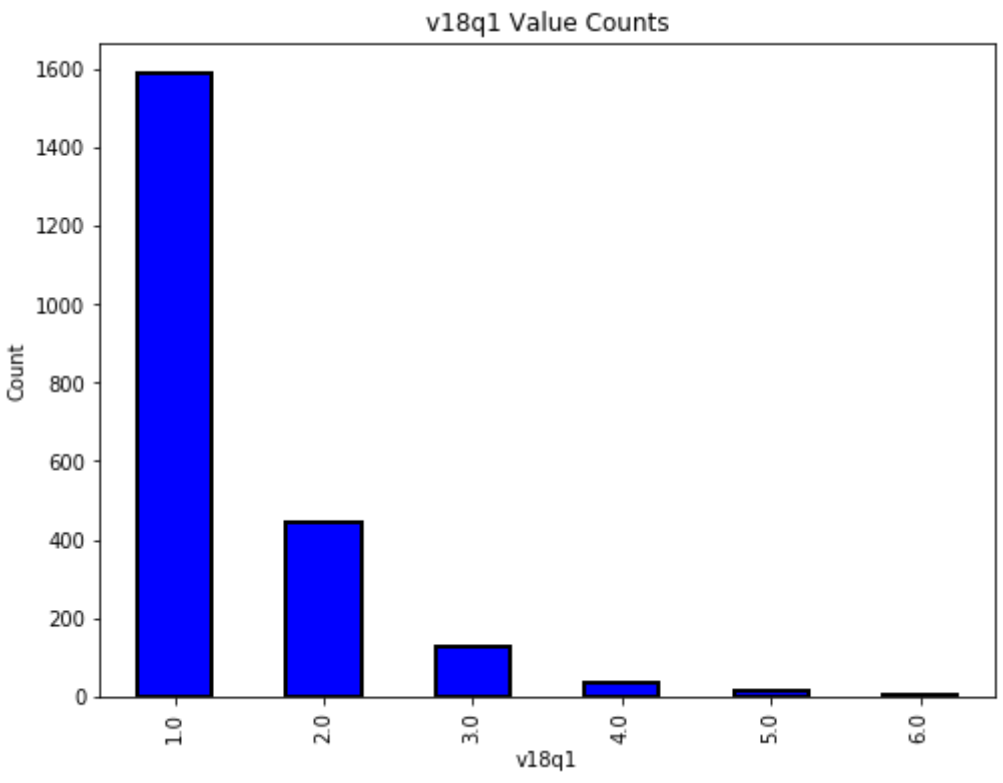
Imputing the missing values with 0

In [32]:
```
data_train['v2a1'].fillna(value=0, inplace=True)
data_train['v2a1'].isnull().sum()
```

Out[32]: 0

Remove null value present in the v18q1feature variable

In [33]:
```
plt.figure(figsize = (8, 6))
col='v18q1'
data_train[col].value_counts().sort_index().plot.bar(color = 'blue',
                                                     edgecolor = 'k',
                                                     linewidth = 2)
plt.xlabel(f'{col}'); plt.title(f'{col} Value Counts'); plt.ylabel('Count')
plt.show();
```



In [34]:
```
nullTablets = data_train[data_train['v18q1'].isnull()]
nullTablets.head(3)
```

Out[34]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQB
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	1849	1	100	
2	ID_68de51c94	0.0	0	8	0	1	1	0	NaN	0	...	121	8464	1	0	
7	ID_3e04e571e	130000.0	1	2	0	1	1	0	NaN	0	...	0	49	16	81	

3 rows × 143 columns

In [35]:
```
nullTablesIndividual = nullTablets['v18q']
print("Check for Table count when there is > 1 tablet with individual: ", nullTablesIndividual[nullTablesIndividual > 0].count(), "\n")
print("Check for Table count when there is 0 tablet with individual: ", nullTablesIndividual[nullTablesIndividual == 0].count())
```

```
Check for Table count when there is > 1 tablet with individual:  0

Check for Table count when there is 0 tablet with individual:  7342
```

Analysis: v18q1 is NAN only when the value of v18q is 0

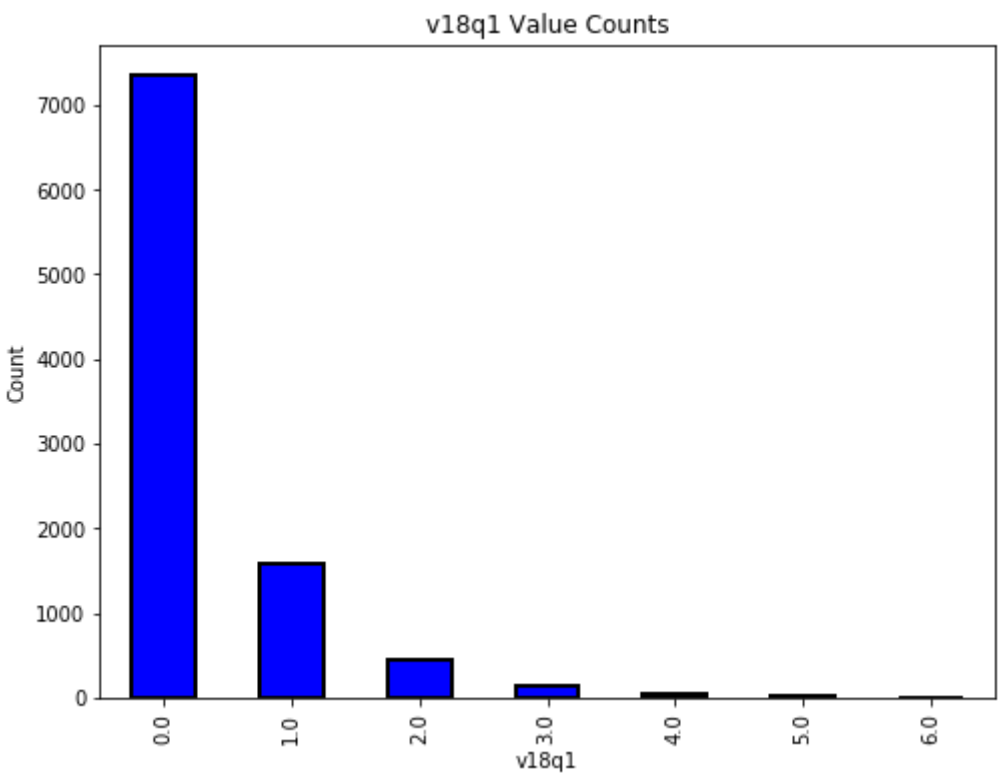
Replacing the NAN values with 0

In [36]:
```
data_train['v18q1'].fillna(value=0, inplace=True)
data_train['v18q1'].isnull().sum()
```

Out[36]: 0

In [37]:
```
plt.figure(figsize = (8, 6))
col='v18q1'
data_train[col].value_counts().sort_index().plot.bar(color = 'blue',
                                                     edgecolor = 'k',
                                                     linewidth = 2)
plt.xlabel(f'{col}'); plt.title(f'{col} Value Counts'); plt.ylabel('Count')
```

Out[37]: Text(0, 0.5, 'Count')



Remove null value present in the meaneduc & SQBmeaned feature variable

In [38]:
```
nullMeanEduc = data_train[data_train['meaneduc'].isnull()]
nullMeanEduc.head()
```

Out[38]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	S
1291	ID_bd8e11b0f	0.0	0	7	0	1	1	0	0.0	0	...	100	324	1	0	
1840	ID_46ff87316	110000.0	0	1	0	1	1	0	0.0	0	...	36	324	4	16	
1841	ID_69f50bf3e	110000.0	0	1	0	1	1	0	0.0	0	...	16	324	4	16	
2049	ID_db3168f9f	180000.0	0	3	0	1	1	0	0.0	0	...	144	361	4	144	
2050	ID_2a7615902	180000.0	0	3	0	1	1	0	0.0	0	...	144	361	4	144	

5 rows × 143 columns

In [39]:
```
nullMeanEduc = nullMeanEduc.loc[:,['age','meaneduc','idhogar','instlevel1','hogar_adul']]
nullMeanEduc
```

Out[39]:

	age	meaneduc	idhogar	instlevel1	hogar_adul
1291	18	NaN	1b31fd159	0	0
1840	18	NaN	a874b7ce7	0	0
1841	18	NaN	a874b7ce7	0	0
2049	19	NaN	faaebf71a	0	0
2050	19	NaN	faaebf71a	0	0

Analysis:According to above data, last 2 row seems to belong from same household and 19+ is considered as adult, hence hogar_adul is misrepresnting the data. Looking at instlevel1 we can say NaN values are presnt for all members whose level of education is 0. Hence, we can replace the value with 0

In [40]:
```
data_train['meaneduc'].fillna(0,inplace=True)
data_train['SQBmeaned'].fillna(0,inplace=True)
print(data_train.isna().sum().value_counts())
```

```
0       142
7928      1
dtype: int64
```

Remove null value present in the rez_esc feature variable

In [41]:
```
data_train.drop(columns=['rez_esc'], inplace=True)
```

In [42]:
```
data_train.shape
```

Out[42]:
```
(9557, 142)
```

Find if there exists any feature with 0 variance

In [43]:
```
dataVariance = data_train.var()
toBeDroppedRows = dataVariance[dataVariance == 0]
toBeDroppedRows
```

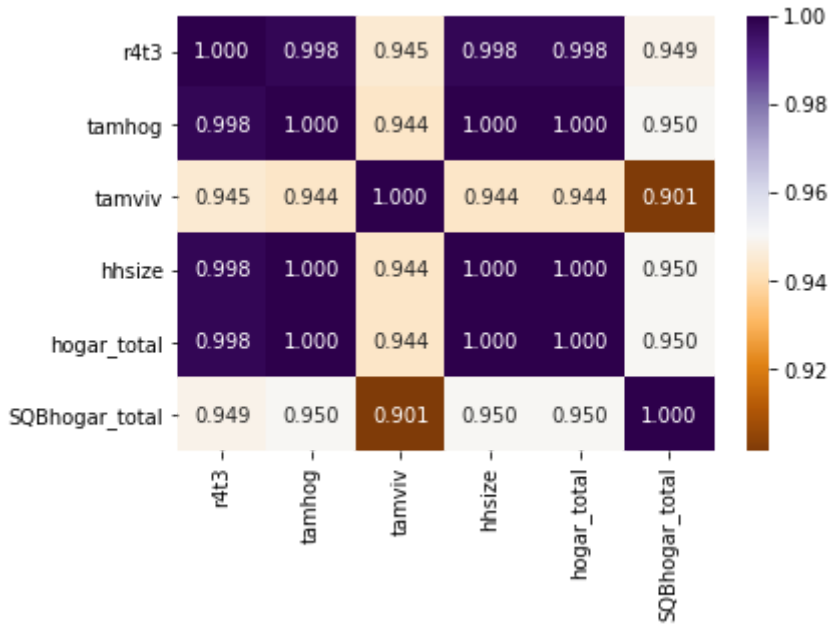
Out[43]:
```
elimbasu5    0.0
dtype: float64
```

Check for Highly Correlated Data

In [44]:
```
correlation = data_train.corr()

# Select upper triangle of correlation matrix
upper = correlation.where(np.triu(np.ones(correlation.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.95
highVarianceColumns = [column for column in upper.columns if any(abs(upper[column]) > 0.95)]


highVarianceColumns
```

Out[44]:
```
['tamhog',
 'hhsize',
 'coopele',
 'female',
 'hogar_total',
 'area2',
 'SQBage',
 'SQBhogar_total',
 'agesq']
```

In [45]:
```
sns.heatmap(correlation.loc[correlation['tamhog'].abs() > 0.9, correlation['tamhog'].abs() > 0.9],
            annot=True, cmap = plt.cm.PuOr, fmt='.3f');
```



Remove the features which has higer correlation

In [46]:
```
data_train.drop(columns=highVarianceColumns,inplace=True)
data_train.shape
```

Out[46]: (9557, 133)

Remove Unwanted Features

In [47]:
```
featuresToRemove = ['Id','elimbasu5','idhogar','tamviv','SQBescolari','SQBedjefe','SQBhogar_nin','SQBmeaned',
                    'SQBdependency','SQBovercrowding']

data_train.drop(columns=featuresToRemove,inplace=True)
data_train.shape
```

Out[47]: (9557, 123)

Calculate VIF Scores for features left

In [48]:
```
vif = pd.DataFrame()

def calc_vif(X):

    # Calculating VIF
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif.head())

X = data_train.iloc[:,:-1]
calc_vif(X)
```

Out[48]:

	variables	VIF
0	v2a1	2.314023
1	hacdor	2.848814
2	rooms	3.661095
3	hacapo	2.069508
4	v14a	1.661559

In [49]:
```
vifFeatures = vif[vif['VIF'] < 5.0]
vifFeatures
```

Out[49]:

	variables	VIF
0	v2a1	2.314023
1	hacdor	2.848814
2	rooms	3.661095
3	hacapo	2.069508
4	v14a	1.661559
5	refrig	1.238010
6	v18q	3.822567
7	v18q1	3.847648
34	techocane	3.517894
35	techootro	2.835830
36	cielorazo	1.911436
40	public	1.556971
41	planpri	1.014696
42	noelec	1.196070
66	dis	1.133267
67	male	1.537065
89	hogar_mayor	2.833492
90	dependency	2.413497
91	edjefe	4.790400
92	edjefa	4.076021
93	meaneduc	3.421691
110	computer	1.251761
111	television	1.155111
112	mobilephone	1.276683
113	qmobilephone	2.619380
120	area1	1.581404

Feature Engineering

Separating Feature and Label columns

In [50]:
```
featureNames = data_train.columns.to_list()
featureNames.remove('Target')
print("No of features taken into consideration: ", len(featureNames))
```

```
No of features taken into consideration:  122
```

In [51]:
```
features = data_train.iloc[:,:-1]
label = data_train.loc[:,['Target']]

print("Features shape: ",features.shape)
print("Label shape: ",label.shape)
```

```
Features shape:  (9557, 122)
Label shape:  (9557, 1)
```

In [52]:
```
features = features.values
label = label.values
```

Finding Important Features using Select By Model

In [53]:
```
modelRFC = RandomForestClassifier()
selectFeaturesFromSFM = SelectFromModel(modelRFC)
selectFeaturesFromSFM.fit(features,label)
print(selectFeaturesFromSFM.get_support())
```

```
[ True False  True False False False  True  True  True  True  True  True
  True  True  True  True  True  True  True False  True False False False
 False False  True  True False False False False False False False False
  True False False False False False False False False False False False
 False  True  True False False False False False False False  True  True
 False  True  True False  True  True False False False False False False
 False False False False False False False False False False False False
 False False False  True  True  True  True  True  True  True False False
 False False False False False False False  True  True  True False False
 False False False  True False  True  True False False False False False
  True  True]
```

In [54]:
```
selectedFeatures = []
model = RandomForestClassifier()
selectFeaturesFromSFM = SelectFromModel(model)
selectFeaturesFromSFM.fit(features,label)
for feature_list_index in selectFeaturesFromSFM.get_support(indices=True):
    selectedFeatures.append(featureNames[feature_list_index])

print(selectedFeatures,"\n")
print("Total features selected from SFM is: ", len(selectedFeatures))
```

```
['v2a1', 'rooms', 'v18q1', 'r4h1', 'r4h2', 'r4h3', 'r4m1', 'r4m2', 'r4m3', 'r4t1', 'r4t2', 'r4t3', 'escolari', 'pared
blolad', 'paredpreb', 'pisomoscer', 'cielorazo', 'energcocinar2', 'energcocinar3', 'epared2', 'epared3', 'etecho2',
'etecho3', 'eviv3', 'hogar_nin', 'hogar_adul', 'hogar_mayor', 'dependency', 'edjefe', 'edjefa', 'meaneduc', 'bedroom
s', 'overcrowding', 'tipovivi1', 'television', 'qmobilephone', 'lugar1', 'area1', 'age']

Total features selected from SFM is:  39
```

Building initial model

Applying Output of Feature Engineering

In [55]:
```
finalFeatures = selectFeaturesFromSFM.transform(features)
selectedFeaturesFrame= pd.DataFrame(finalFeatures, columns=selectedFeatures)
print("Shape of selected feature is: ", selectedFeaturesFrame.shape)
selectedFeaturesFrame.head()
```

```
Shape of selected feature is:  (9557, 39)
```

Out[55]:

	v2a1	rooms	v18q1	r4h1	r4h2	r4h3	r4m1	r4m2	r4m3	r4t1	...	edjefa	meaneduc	bedrooms	overcrowding	tipovivi1	television	qmc
0	190000.0	3.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	10.0	1.0	1.000000	0.0	0.0	
1	135000.0	4.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	...	0.0	12.0	1.0	1.000000	0.0	0.0	
2	0.0	8.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	...	11.0	11.0	2.0	0.500000	1.0	0.0	
3	180000.0	5.0	1.0	0.0	2.0	2.0	1.0	1.0	2.0	1.0	...	0.0	11.0	3.0	1.333333	0.0	0.0	
4	180000.0	5.0	1.0	0.0	2.0	2.0	1.0	1.0	2.0	1.0	...	0.0	11.0	3.0	1.333333	0.0	0.0	

5 rows × 39 columns

TASK 9: Predict the accuracy using random forest classifier.

In [56]:
```
for i in range(1,10):
    X_train,X_test,y_train,y_test = train_test_split(finalFeatures, label, test_size=0.2, random_state = i)
    model1 = RandomForestClassifier()
    model1.fit(X_train,y_train)

    train_score = model1.score(X_train,y_train)
    test_score = model1.score(X_test,y_test)

    if (test_score > 0.95):
        print("Test: {} , Train: {} , RS : {}".format(test_score,train_score,i))
```

```
Test: 0.9665271966527197 , Train: 1.0 , RS : 1
Test: 0.9555439330543933 , Train: 1.0 , RS : 2
Test: 0.9612970711297071 , Train: 1.0 , RS : 3
Test: 0.9748953974895398 , Train: 1.0 , RS : 4
Test: 0.9607740585774058 , Train: 1.0 , RS : 5
Test: 0.9691422594142259 , Train: 1.0 , RS : 6
Test: 0.9602510460251046 , Train: 1.0 , RS : 7
Test: 0.9701882845188284 , Train: 1.0 , RS : 8
Test: 0.9576359832635983 , Train: 1.0 , RS : 9
```

Analysis: Maxinum Accuracy is obtained when RS is 4

Train Test Data Split

In [57]:
```
X_train,X_test,y_train,y_test = train_test_split(finalFeatures, label, test_size=0.2, random_state = 4)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[57]:
```
((7645, 39), (1912, 39), (7645, 1), (1912, 1))
```

Standard Scalar

In [58]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Cross Validation

Using Cross Validation technique to find the best accuracy result

In [59]:
```
crossValAlgo = RandomForestClassifier()
scores = cross_val_score(crossValAlgo,
                         finalFeatures,
                         label,
                         cv = 10) #5 or 10

scores
```

Out[59]:
```
array([0.65167364, 0.67573222, 0.68200837, 0.64539749, 0.68096234,
       0.69142259, 0.60460251, 0.66282723, 0.60314136, 0.59790576])
```

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

Applying K-fold Validation to check the best train-test split

In [60]:
```
kfoldModel = RandomForestClassifier()

from sklearn.model_selection import KFold

kfold = KFold(n_splits=10, #Use the same CV values that was applied in cross_val_score
              shuffle=True,
              random_state = 8) # To ensure the data is not randomized at every iteration

counter = 0

for train,test in kfold.split(finalFeatures):

    #Counter will help you track the sample split
    counter += 1

    #Extract the training set and testing set
    X_train,X_test = finalFeatures[train],finalFeatures[test]
    y_train,y_test = label[train] , label[test]

    #Fit the model
    kfoldModel.fit(X_train,y_train)

    if kfoldModel.score(X_test,y_test) > 0.96:
        print("Test Score {} Train Score {} for Sample Split {}".format(kfoldModel.score(X_test,y_test),kfoldModel.sc
ore(X_train,y_train),counter))
```

```
Test Score 0.9864016736401674 Train Score 1.0 for Sample Split 1
Test Score 0.9665271966527197 Train Score 1.0 for Sample Split 2
Test Score 0.9717573221757322 Train Score 1.0 for Sample Split 3
Test Score 0.9623430962343096 Train Score 1.0 for Sample Split 4
Test Score 0.9696652719665272 Train Score 1.0 for Sample Split 5
Test Score 0.9780334728033473 Train Score 1.0 for Sample Split 6
Test Score 0.9633891213389121 Train Score 1.0 for Sample Split 7
Test Score 0.9675392670157068 Train Score 1.0 for Sample Split 8
Test Score 0.9664921465968587 Train Score 1.0 for Sample Split 9
Test Score 0.9602094240837696 Train Score 1.0 for Sample Split 10
```

Find the Parameters of Random Forest Classifier for Hyper parameter tuning

RandomForestRegressor(	
n_estimators=100	Positive Int Value
criterion='gini',	"gini", "entropy"
max_depth=None,	Positive Int Value
min_samples_split=2,	Positive Value
min_samples_leaf=1,	Positive Value
min_weight_fraction_leaf=0.0,	Positive Float Value
max_features='auto',	"auto", "sqrt", "log2"
max_leaf_nodes=None,	Positive Int Value
min_impurity_decrease=0.0,	Positive Float Value
min_impurity_split=None,	Positive Float Value (to be depreceated soon)
bootstrap=True,	True, False
oob_score=False,	True, False
n_jobs=None,	1, -1 or -2
random_state=None,	Positive Int Value
verbose=0,	Positive Int Value
warm_start=False,	True, False
class_weight=None	"balanced", "balanced_subsample"
ccp_alpha=0.0	Positive Float Value
max_samples=None	Positive Value
)	

In [61]:
```
#Step1: Design the parameter grid
criterionParameter = ['gini', 'entropy']
maxFeaturesParameter = ['auto', 'sqrt', 'log2']
nJobsParameter = [-2,-1,1]

paramGrid = dict(criterion=criterionParameter,
                 max_features=maxFeaturesParameter,
                 n_jobs=nJobsParameter)

# Step2: Initialize the algo
modelGridSearch = RandomForestClassifier()

#Step3: Search the best parameter for the data
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(modelGridSearch,
                    param_grid=paramGrid,
                    cv = 10) #Same as cross_val_score
```

In [62]:
```
# Step4: Extract Results
grid.fit(finalFeatures,label)
```

Out[62]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_jobs': [-2, -1, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [63]:
```
grid.best_score_
```

Out[63]:
```
0.6528125479199982
```

In [64]:
```
grid.best_estimator_
```

Out[64]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-2, oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

Building Final Model

In [65]:
```
kfold = KFold(n_splits=10, #Use the same CV values that was applied in cross_val_score
              shuffle=True,
              random_state = 8) # To ensure the data is not randomized at every iteration
counter = 0
for train,test in kfold.split(finalFeatures):

    #Counter will help you track the sample split
    counter += 1

    if counter == 1:
        X_train,X_test,y_train,y_test = finalFeatures[train],finalFeatures[test],label[train] ,label[test]
```

In [66]:
```
finalModel = RandomForestClassifier()
finalModel.fit(X_train,y_train)
finalModel.score(X_test,y_test)
```

Out[66]:
```
0.9822175732217573
```

In [67]:
```
predictedValue = finalModel.predict(X_test)
```

In [68]:
```
print(confusion_matrix(y_test,predictedValue))
```

```
[[ 71   0   0   3]
 [  1 159   0   4]
 [  1   0 121   5]
 [  0   1   2 588]]
```

In [69]:
```
print(classification_report(y_test,predictedValue))
```

```
              precision    recall  f1-score   support

           1       0.97      0.96      0.97        74
           2       0.99      0.97      0.98       164
           3       0.98      0.95      0.97       127
           4       0.98      0.99      0.99       591

    accuracy                           0.98       956
   macro avg       0.98      0.97      0.98       956
weighted avg       0.98      0.98      0.98       956
```

Visualizing the Target and Predicted Values

In [70]:
```
# Converting Dimensions of y_test from 2 to 1 dimension
y_test = y_test.ravel()
```

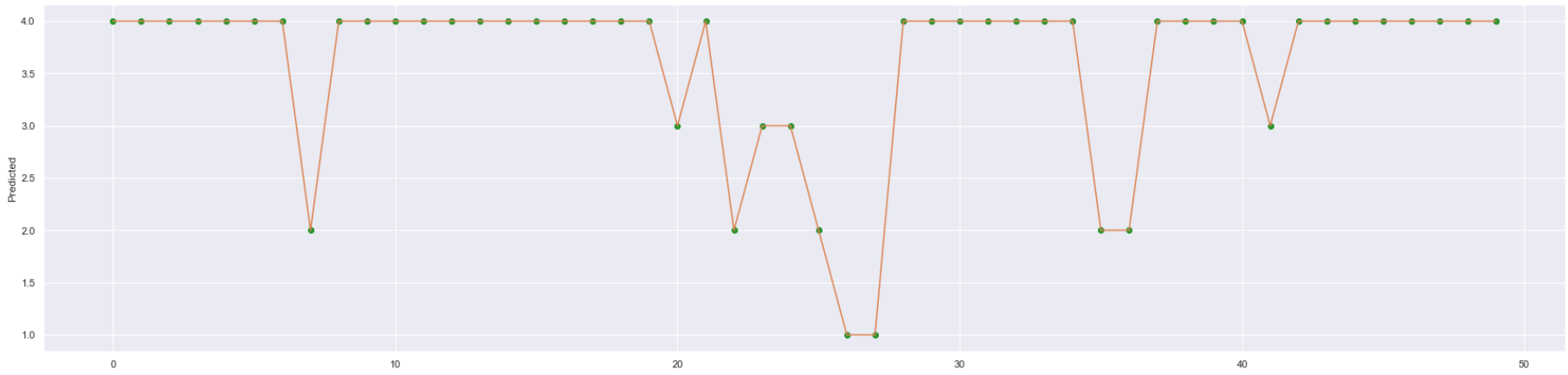
In [71]:
```
predictedFrame = pd.DataFrame(predictedValue, columns=['Predicted'])
targetFrame = pd.DataFrame(y_test, columns=['Target'])
Observed_Predicted_Table =pd.concat([targetFrame.reset_index(drop=True),predictedFrame],axis=1)
Observed_Predicted_Table.head()
```

Out[71]:

	Target	Predicted
0	4	4
1	4	4
2	4	4
3	4	4
4	4	4

In [74]:
```
Observed_Predicted_Table_Split = Observed_Predicted_Table.head(50)
fig, ax = plt.subplots()
sns.set(color_codes=True)
sns.set(rc={'figure.figsize':(30, 7)})
sns.regplot(Observed_Predicted_Table_Split.index, Observed_Predicted_Table_Split.Target, fit_reg=False, ax=ax,scatter
_kws={"color": "green"});
sns.lineplot(Observed_Predicted_Table_Split.index, Observed_Predicted_Table_Split.Predicted)
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x20e0b5c3ac8>



Deploy the final model

In [73]:
```
import pickle
pickle.dump(finalModel , open('modelIncomeQualification.model' , 'wb') )
```