**Project: Income Qualification**

**Objective:**

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.

**Data set names:**

Train.csv

Test.csv

**Source code:**

In [30]:

*# Importing libraries*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **sklearn**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

%**matplotlib** inline

In [2]:

*# Importing the dataset*

train\_df = pd.read\_csv("train.csv")

test\_df = pd.read\_csv("test.csv")

In [3]:

train\_df.shape

Out[3]:

Output: (9557, 143)

In [4]:

test\_df.shape

Out[4]:

**Output**: (23856, 142)

In [5]:

train\_df.info()

**Output**: <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

In [6]:

test\_df.info()

**Output**: <class 'pandas.core.frame.DataFrame'>

RangeIndex: 23856 entries, 0 to 23855

Columns: 142 entries, Id to agesq

dtypes: float64(8), int64(129), object(5)

memory usage: 25.8+ MB

In [7]:

*# Identify the output variable.*

train\_df['Target']

Out[7]:

**Output:**

0 4

1 4

2 4

3 4

4 4

..

9552 2

9553 2

9554 2

9555 2

9556 2

Name: Target, Length: 9557, dtype: int64

In [8]:

*# Understand the type of the data*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

train\_df.dtypes

Out[8]:

**Output:**

Id object

v2a1 float64

hacdor int64

rooms int64

hacapo int64

...

SQBovercrowding float64

SQBdependency float64

SQBmeaned float64

agesq int64

Target int64

Length: 143, dtype: object

In [9]:

test\_df.dtypes

Out[9]:

**Output:**

Id object

v2a1 float64

hacdor int64

rooms int64

hacapo int64

...

SQBhogar\_nin int64

SQBovercrowding float64

SQBdependency float64

SQBmeaned float64

agesq int64

Length: 142, dtype: object

In [10]:

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

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

train\_df['Target'].value\_counts()

*# Insight: There is bias with Target - 4*

Out[10]:

**Output:**

4 5996

2 1597

3 1209

1 755

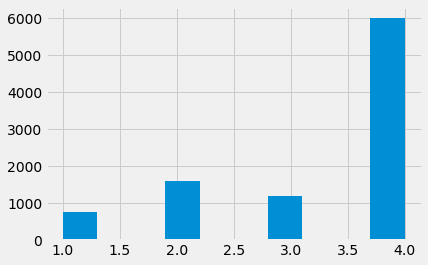
Name: Target, dtype: int64

In [58]:

train\_df['Target'].hist()

Out[58]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x1c77c122d08>



In [120]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*# 1 = extreme poverty*

*# 2 = moderate poverty*

*# 3 = vulnerable households*

*# 4 = non vulnerable households*

colors = OrderedDict({1: 'red', 2: 'orange', 3: 'blue', 4: 'green'})

poverty\_mapping = OrderedDict({1: 'extreme', 2: 'moderate', 3: 'vulnerable', 4: 'non vulnerable'})

mem\_p\_level = train\_df.groupby('idhogar')['r4t3'].unique().sum()

print('Houses with same poverty level of all the members : ',mem\_p\_level)

**Output:**

Houses with same poverty level of all the members : [9643]

In [71]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*#d1 = train\_df.loc[train\_df['parentesco1']==1]*

*#print("Count of Houses without Family head is: ",len(train\_df)-len(d1))*

households\_head = train\_df.groupby('idhogar')['parentesco1'].sum()

*# Find households without a head*

households\_no\_head = train\_df.loc[train\_df['idhogar'].isin(households\_head[households\_head == 0].index), :]

print('There are **{}** households without a head.'.format(households\_no\_head['idhogar'].nunique()))

**Output:**

There are 15 households without a head.

In [115]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

p\_level = int(train\_df[(train\_df['idhogar'] == 1) & (train\_df['parentesco1'] == 1.0)]['Target'])

train\_df.loc[train\_df['idhogar'] == 1, 'Target'] = p\_level

print('There are **{}** houses with different poverty level.'.format(p\_level))

**Output:**

There are 2 houses with different poverty level.

In [14]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

train\_df.isnull().sum()

*# Insight: Gets Count of null values in each column respetively*

Out[14]:

**Output:**

Id 0

v2a1 6860

hacdor 0

rooms 0

hacapo 0

...

SQBovercrowding 0

SQBdependency 0

SQBmeaned 5

agesq 0

Target 0

Length: 143, dtype: int64

In [15]:

train\_df.columns[train\_df.isnull().sum()!=0]

*#Insight: Result is the column names that have null values*

Out[15]:

**Output:**

Index(['v2a1', 'v18q1', 'rez\_esc', 'meaneduc', 'SQBmeaned'], dtype='object')

In [16]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

train\_df.loc[train\_df['v2a1'].isna()]

train\_df['v2a1'].describe()

train\_df['v2a1'] = train\_df['v2a1'].fillna(0)

*# Insight: 6860 rows is with NaN values in "v2a1" column (Monthly rent payment)*

*# As the min value equals to "0",*

*# so the best choice is NaN values are filled with 0*

In [17]:

train\_df.loc[train\_df['v18q1'].isna()]

train\_df['v18q1'].describe()

train\_df['v18q1'].unique()

train\_df['v18q1'] = train\_df['v18q1'].fillna(0)

*#Insight: 7342 rows is with NaN values in "v18q1" column (number of tablets household owns)*

*# As the column values are integers (count of tablets) without 0*

*# thus filling NaN values as 0*

In [18]:

train\_df.loc[train\_df['rez\_esc'].isna()]

train\_df['rez\_esc'].describe()

train\_df['rez\_esc'] = train\_df['rez\_esc'].fillna(0)

*#Insight: 7928 rows is with NaN values in rez\_esc column*

*# Filling values with "0"*

In [19]:

train\_df.loc[train\_df['meaneduc'].isna()]

train\_df['meaneduc'].describe()

train\_df['meaneduc'] = train\_df['meaneduc'].fillna(0)

*#Insight: 5 rows is with NaN values in meaneduc column*

In [20]:

train\_df.loc[train\_df['SQBmeaned'].isna()]

train\_df['SQBmeaned'].describe()

train\_df['SQBmeaned'] = train\_df['SQBmeaned'].fillna(0)

*#Insight: 5 rows is with NaN values in SQBmeaned column*

In [48]:

*# Removing NaN values in the test data and replacing them with "0"*

test\_df.isnull().sum()

test\_df = test\_df.fillna(0)

In [22]:

*# Checking for mixed data columns*

train\_df.columns[train\_df.dtypes == object]

Out[22]:

**Output:**

Index(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'], dtype='object')

In [50]:

*# Applying Label Encoding on the columns - 'idhogar', 'dependency', 'edjefe', 'edjefa'*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**from** **sklearn.preprocessing** **import** LabelEncoder

*#train\_df['dependency'] = train\_df['dependency',inplace = True].replace(yes\_or\_no).astype(np.float32)*

**def** encode\_data(df):

yes\_or\_no = {'no': 0, 'yes': 1}

df['dependency'] = df['dependency'].replace(yes\_or\_no).astype(np.float32)

df['edjefe'] = df['edjefe'].replace(yes\_or\_no).astype(np.float32)

df['edjefa'] = df['edjefa'].replace(yes\_or\_no).astype(np.float32)

df['idhogar'] = LabelEncoder().fit\_transform(df['idhogar'])

encode\_data(train\_df)

encode\_data(test\_df)

**---------------------------------------------------------------------------**

**Output:**

Columns are encoded.

In [99]:

*#Train and Test Split the data*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**from** **sklearn.model\_selection** **import** train\_test\_split

*#Splitting the data*

X = train\_df[['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil2', 'estadocivil3','estadocivil4', 'estadocivil5', 'estadocivil6', 'estadocivil7','parentesco1', 'parentesco2', 'parentesco3', 'parentesco4', 'parentesco5','parentesco6', 'parentesco7', 'parentesco8', 'parentesco9', 'parentesco10','parentesco11', 'parentesco12', 'instlevel1', 'instlevel2', 'instlevel3','instlevel4', 'instlevel5', 'instlevel6', 'instlevel7', 'instlevel8','instlevel9', 'mobilephone','rez\_esc', 'escolari', 'age','hacdor', 'hacapo', 'v14a', 'refrig', 'paredblolad', 'paredzocalo','paredpreb','pisocemento', 'pareddes', 'paredmad','paredzinc', 'paredfibras', 'paredother', 'pisomoscer', 'pisoother','pisonatur', 'pisonotiene', 'pisomadera','techozinc', 'techoentrepiso', 'techocane', 'techootro', 'cielorazo','abastaguadentro', 'abastaguafuera', 'abastaguano','public', 'planpri', 'noelec', 'coopele', 'sanitario1','sanitario2', 'sanitario3', 'sanitario5', 'sanitario6','energcocinar1', 'energcocinar2', 'energcocinar3', 'energcocinar4','elimbasu1', 'elimbasu2', 'elimbasu3', 'elimbasu4','elimbasu5', 'elimbasu6', 'epared1', 'epared2', 'epared3','etecho1', 'etecho2', 'etecho3', 'eviv1', 'eviv2', 'eviv3','tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi4', 'tipovivi5','computer', 'television', 'lugar1', 'lugar2', 'lugar3','lugar4', 'lugar5', 'lugar6', 'area1', 'area2','rooms','r4h1','r4h2','r4h3','r4m1','r4m2','r4m3', 'r4t1', 'r4t2','r4t3', 'v18q1', 'tamhog','tamviv','hhsize','hogar\_nin','hogar\_adul','hogar\_mayor','hogar\_total', 'bedrooms', 'qmobilephone', 'v2a1', 'dependency', 'edjefe', 'edjefa', 'meaneduc', 'overcrowding']]

*#X = train\_df[['v2a1','rooms','tamhog','r4t3','hhsize','v18q1','tamviv','r4t2','r4h2','r4t1','parentesco1','mobilephone']]*

y = train\_df['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X , y, test\_size=0.2)

In [106]:

*# Predict the accuracy using random forest classifier.*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**from** **sklearn.ensemble** **import** RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=100)

rfc.fit(X\_train,y\_train)

y\_pred = rfc.predict(X\_test)

*# Imported scikit-learn metrics module for accuracy calculation*

**from** **sklearn** **import** metrics

print("Accuracy: **%.2f%%** " % ((metrics.accuracy\_score(y\_test, y\_pred))\*100))

**Output:**

Accuracy: 92.10%

In [108]:

*# train-test split evaluation of xgboost model*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** **xgboost**

**from** **numpy** **import** loadtxt

**from** **xgboost** **import** XGBClassifier

**from** **sklearn.metrics** **import** accuracy\_score

model = XGBClassifier()

model.fit(X\_train, y\_train)

*# make predictions for test data*

y\_pred = model.predict(X\_test)

*# evaluate predictions*

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy: **%.2f%%**" % (accuracy \* 100.0))

**Output:**

Accuracy: 91.32%

In [101]:

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*# k-fold cross validation evaluation of xgboost model*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**from** **sklearn.model\_selection** **import** KFold

**from** **sklearn.model\_selection** **import** cross\_val\_score

*# CV model*

model = xgboost.XGBClassifier()

kfold = KFold(n\_splits=10, random\_state=7)

results = cross\_val\_score(model, X, y, cv=kfold)

print("Accuracy: **%.2f%%** (**%.2f%%**)" % (results.mean()\*100, results.std()\*100))

C:\Users\Tejasri5\anaconda3\lib\site-packages\sklearn\model\_selection\\_split.py:296: FutureWarning: Setting a random\_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave random\_state to its default (None), or set shuffle=True.

FutureWarning

**Output:**

Accuracy: 62.50% (9.06%)

**Insights:**

1. The Output variable is "Target" variable.
2. The type of the data is float64, int64, object
3. There are biases in the dataset.
4. Count of the members of the houses have the same poverty level - 9643
5. There are 15 houses without a family head.
6. There are 2 households with different poverty level of the members and the head of the house within a family.
7. Null values are existing in columns. (v2a1 – 6860, v18q1 - 7342, rez\_esc – 7928, meaneduc – 5, SQBmeaned – 5)
8. Removed null value rows of the target variable.
9. 92.10% accuracy using random forest classifier.
10. 91.32% accuracy using xgboost model.
11. 62.50% accuracy with cross validation (k-fold cross validation evaluation of xgboost model).