Submitted by:Krishna Aryal
Trainer : Abhishek Tiwari

In [315]:
```
!jt -t chesterish
```

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

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

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

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

The Inter-American Development Bank (IDB)believes that new methods beyond traditional econometrics, based on a dataset of Costa Rican household characteristics, might help improve PMT's performance. Following actions should be performed:

1. Identify the output variable.

Load Libraries

In [316]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load data

In [317]:
```
df_train = pd.read_csv("train.csv")
df_test  =pd.read_csv("test.csv")
```

In [318]:
```
df_train.head()
```

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

5 rows × 143 columns

In [319]:
```
df_test.head()
```

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQBescolari	SQBage	SQBhogar_total	SQBec
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

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	age	SQBescolari	SQBage	SQBhogar_total	SQBed
4	ID_a62966799	175000.0	0	4	0	1	1	1	1	0	...	18	121	324	1	9

5 rows × 142 columns

Ans: Output variable is Target . By observing Target column output is in form 4,3,2,1.

Train data has 143 columns and test data has 142 columns. Let's check which is not matching that will be our output variable. That we will find by checking column name

In [320]:
```
for i in df_train.columns:
    if i not in df_test.columns:
        print("Output variable is {}".format(i))
```

```
Output variable is Target
```

2. Understand the type of data.

In [321]:
```
df_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
```

In [322]:
```
df_train.dtypes.value_counts()
```

```
int64      130
float64      8
object       5
dtype: int64
```

There are 130 columns integer types, 8 float64 type, that won't create problem. Let's figure out object type column and what are they.

In [323]:
```
df_train.select_dtypes(np.object).columns
```

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

Below is Data dictionary for above object variables

(1) ID = Unique ID

(2) idhogar, Household level identifier

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

(4) edjefe, years of education of male head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0

(5) edjefa, years of education of female head of household, based on the interaction of escolari (years of education), head of household and gender, yes=1 and no=0

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

In [324]:
```
heads=df_train.loc[df_train['parentesco1'] == 1].copy()
target_counts = heads['Target'].value_counts().sort_index()
target_counts
```

```
1     222
2     442
3     355
4    1954
Name: Target, dtype: int64
```

1 = extreme poverty 2 = moderate poverty 3 = vulnerable households 4 = non vulnerable households very small value of small proverty in comparison with others shows there is biasness in the data.

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

idhogar, Household level identifier,if idhogar is matching with Target, all member will have same poverty level otherwise not.

In [325]:
```
all_same = df_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)
```

In [326]:
```
not_same = all_same[all_same != True]
```

In [327]:
```
not_same
```

```
idhogar
0172ab1d9    False
03f4e5f4d    False
0511912b6    False
078a0b6e2    False
09e25d616    False
             ...
e65d4b943    False
efd3aec61    False
f006348ed    False
f7b421c2c    False
f94589d38    False
Name: Target, Length: 85, dtype: bool
```

85 families do not have same poverty level

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

"parentesco1" =1 if household head

In [328]:
```
df_train.parentesco1.value_counts()
```

```
0    6584
1    2973
Name: parentesco1, dtype: int64
```

Out of 9557 rows 2973 are head of families. idhogar, Household level identifier helps to figure out families without head.

In [329]:
```
df_train.groupby('idhogar')['parentesco1'].sum()
```

```
idhogar
001ff74ca    1
003123ec2    1
004616164    1
004983866    1
005905417    1
            ..
ff9343a35    1
ff9d5ab17    1
ffae4a097    1
```

```
ffe90d46f    1
fff7d6be1    1
Name: parentesco1, Length: 2988, dtype: int64
```

Out of 9557 and 2973 head. so 2988 - 2973 =15 families are without head.

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

In [330]:
```
for household in not_same.index:
    # Find the correct label
    true_target = int(df_train[(df_train['idhogar'] == household) & (df_train['parentesco1'] == 1.0)]['Target'])

    # Set the correct label
    df_train.loc[df_train['idhogar'] == household, 'Target'] = true_target

    all_same = df_train.groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)
     #check it
    not_same = all_same[all_same!= True]
```

In [331]:
```
len(not_same)
```

```
0
```

Now poverty level is matched

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

In [332]:
```
df_train.select_dtypes('float64').isnull().sum()
```

```
v2a1               6860
v18q1              7342
rez_esc            7928
meaneduc              5
overcrowding          0
SQBovercrowding       0
SQBdependency         0
SQBmeaned             5
dtype: int64
```

Ans: v2a1 has 6860 null values,
v18q1 has 7342 null values,
meaneduc has 5 null values,
SQBmeaned has 5 null values

8. Remove null value rows of the target variable.

(a) v2a1 Monthly rent payment has 6860 missing values. let's fill by 0 in place of null

In [333]:
```
for df in [df_train, df_test]:
    df['v2a1'].fillna(value=0, inplace=True)

df_train[['v2a1']].isnull().sum()
```

```
v2a1    0
dtype: int64
```

(b) v18q1, number of tablets household owns 7342 missing values

In [334]:
```
for df in [df_train, df_test]:
    df['v18q1'].fillna(value=0, inplace=True)

df_train[['v18q1']].isnull().sum()
```

```
v18q1    0
dtype: int64
```

(c) rez_esc, Years behind in school 7928 missing values

In [335]:
```
for df in [df_train, df_test]:
    df['rez_esc'].fillna(value=0, inplace=True)
df_train[['rez_esc']].isnull().sum()
```

```
rez_esc    0
dtype: int64
```

(d) meaneduc average years of education for adults (18+) has 5 missing values

In [336]:
```
for df in [df_train, df_test]:
    df['meaneduc'].fillna(value=0, inplace=True)
df_train[['meaneduc']].isnull().sum()
```

```
meaneduc    0
dtype: int64
```

(e) square of the mean years of education of adults (>=18) in the household 142 has 5 missing values

In [337]:
```
for df in [df_train, df_test]:
    df['SQBmeaned'].fillna(value=0, inplace=True)
df_train[['SQBmeaned']].isnull().sum()
```

```
SQBmeaned    0
dtype: int64
```

9. Predict the accuracy using random forest classifier.

In [338]:
```
df_train.select_dtypes(np.object).columns
```

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

In [339]:
```
df_train_1= df_train.drop(['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa'],axis=1)
```

In [340]:
```
df_train_1
```

	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQ
0	190000.0	0	3	0	1	1	0	0.0	0	1	...	100	1849	1	100	0
1	135000.0	0	4	0	1	1	1	1.0	0	1	...	144	4489	1	144	0
2	0.0	0	8	0	1	1	0	0.0	0	0	...	121	8464	1	0	0
3	180000.0	0	5	0	1	1	1	1.0	0	2	...	81	289	16	121	4
4	180000.0	0	5	0	1	1	1	1.0	0	2	...	121	1369	16	121	4

...	...	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	r4h2	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQ
9552	80000.0	0	6	0	1	1	0	0.0	0	2	...	81	2116	25	81	1	
9553	80000.0	0	6	0	1	1	0	0.0	0	2	...	0	4	25	81	1	
9554	80000.0	0	6	0	1	1	0	0.0	0	2	...	25	2500	25	81	1	
9555	80000.0	0	6	0	1	1	0	0.0	0	2	...	121	676	25	81	1	
9556	80000.0	0	6	0	1	1	0	0.0	0	2	...	64	441	25	81	1	

9557 rows × 138 columns

In [341]:
```
X = df_train_1.iloc[:,0:136]
X.shape
```

```
(9557, 136)
```

In [342]:
```
y = df_train_1.iloc[:,137]
```

In [343]:
```
y.shape
```

```
(9557,)
```

In [344]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state = 100 )
```

In [345]:
```
X_train.shape
```

```
(7645, 136)
```

In [346]:
```
y_train.shape
```

```
(7645,)
```

In [347]:
```
X_test.shape
```

```
(1912, 136)
```

In [348]:
```
y_test.shape
```

```
(1912,)
```

In [349]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100, criterion='entropy')
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy')
```

In [350]:
```
rfc_pred = rfc.predict(X_test)
```

In [351]:
```
rfc_pred
```

```
array([4, 3, 4, ..., 4, 4, 4])
```

In [352]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
accuracy=accuracy_score(y_test,rfc_pred)
print(accuracy)
```

```
0.9513598326359832
```

In [353]:
```
rfc_pred_train = rfc.predict(X_train)
```

In [354]:
```
print(confusion_matrix(y_train,rfc_pred_train))
```

```
[[ 612    0    0    0]
 [   0 1238    0    0]
 [   0    0 1004    0]
 [   0    0    0 4791]]
```

In [355]:
```
accuracy_train=accuracy_score(y_train,rfc_pred_train)
```

In [356]:
```
accuracy_train
```

```
1.0
```

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

In [357]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

In [358]:
```
rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='accuracy')
```

In [359]:
```
rfc_cv_score
```

```
array([0.65899582, 0.66841004, 0.66736402, 0.66527197, 0.67050209,
       0.67573222, 0.60460251, 0.56439791, 0.56649215, 0.63560209])
```

In [360]:
```
confusion_matrix(y_test, rfc_pred)
```

```
array([[ 141,    5,    0,   16],
       [   3,  277,    3,   37],
       [   0,    2,  190,   25],
       [   0,    1,    1, 1211]])
```

In [361]:
```
classification_report(y_test, rfc_pred)
```

```
'              precision    recall  f1-score   support\n\n           1       0.98      0.87      0.92       162\n           2       0.97
0.92       320\n           3       0.98      0.88      0.92       217\n           4       0.94      1.00      0.97      1213\n\n    accuracy
0.95      1912\n   macro avg       0.97      0.90      0.93      1912\nweighted avg       0.95      0.95      0.95      1912\n'
```

In [362]:
```
rfc_cv_score.mean()
```

```
0.6377370807684725
```

In [ ]: