Income Qualification Project

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.

In [1]:
```
# Importing required libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor
```

In [2]:
```
# Loading test and train data
train = pd.read_csv('income_qualification_data/train.csv')
test = pd.read_csv('income_qualification_data/test.csv')
print(train.shape, test.shape)
```

```
(9557, 143) (23856, 142)
```

Identifying the output variable

In [3]:
```
print("Output variable is:")
for col in train.columns:
    if col not in test.columns:
        print(col)
```

```
Output variable is:
Target
```

Understanding the type of data

In [4]:
```
train.dtypes.unique()
```

Out[4]:
```
array([dtype('O'), dtype('float64'), dtype('int64')], dtype=object)
```

In [5]:
```python
obj_cols = []
for col in train.columns:
    if train[col].dtype == 'O':
        obj_cols.append(col)

obj_cols
```

Out[5]:
```
['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa']
```

In [6]:
```python
train[obj_cols]
```

Out[6]:

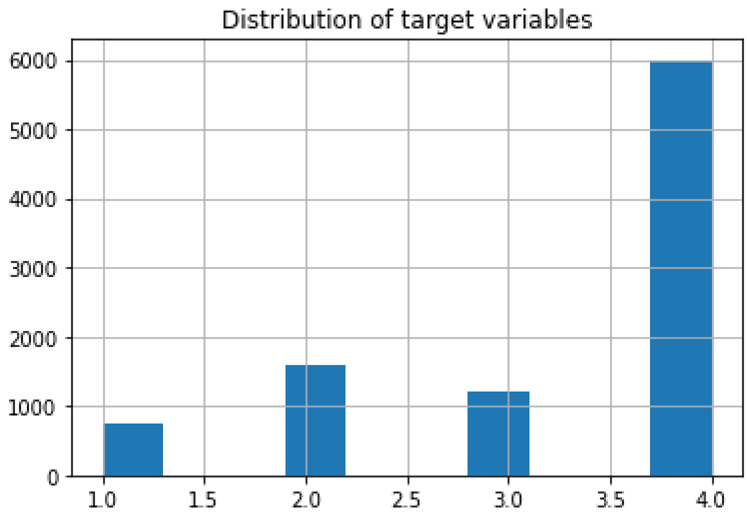
	Id	idhogar	dependency	edjefe	edjefa
0	ID_279628684	21eb7fcc1	no	10	no
1	ID_f29eb3ddd	0e5d7a658	8	12	no
2	ID_68de51c94	2c7317ea8	8	no	11
3	ID_d671db89c	2b58d945f	yes	11	no
4	ID_d56d6f5f5	2b58d945f	yes	11	no
...	...	...	...	...	...
9552	ID_d45ae367d	d6c086aa3	.25	9	no
9553	ID_c94744e07	d6c086aa3	.25	9	no
9554	ID_85fc658f8	d6c086aa3	.25	9	no
9555	ID_ced540c61	d6c086aa3	.25	9	no
9556	ID_a38c64491	d6c086aa3	.25	9	no

9557 rows × 5 columns

Checking if there are any biases in the dataset

In [7]:
```python
plt.title("Distribution of target variables")
train['Target'].hist()
# Viewing the value counts in the target data
train['Target'].value_counts()
```

Out[7]:
```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```



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

In [8]:
```
poverty_lvl = train[((train['idhogar'] == train['idhogar'].shift()) & (train['Targe
poverty_lvl
```

Out[8]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQI
283	ID_17d9dcd44	60000.0	0	3	0	1	0	0	NaN	0	...	
290	ID_606ed140f	NaN	0	3	0	1	1	0	NaN	0	...	
322	ID_dab0d86a2	160000.0	0	6	0	1	1	0	NaN	0	...	
323	ID_6e8c57bc1	160000.0	0	6	0	1	1	0	NaN	0	...	
410	ID_0f28c8bfa	180000.0	0	7	0	1	1	0	NaN	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
9371	ID_938d043f0	NaN	0	4	0	1	1	0	NaN	1	...	
9372	ID_3f735f7fe	NaN	0	4	0	1	1	0	NaN	1	...	
9475	ID_d4f56f88a	0.0	0	4	0	1	1	0	NaN	1	...	
9476	ID_2aa25ef1f	0.0	0	4	0	1	1	0	NaN	1	...	
9536	ID_d20bd7576	NaN	0	4	0	1	0	0	NaN	1	...	

137 rows × 143 columns

Checking if there is a house without a family head

In [9]:
```
train['parentesco1'].value_counts()
```

Out[9]:
```
0    6584
1    2973
Name: parentesco1, dtype: int64
```

In [10]:
```
with_family_head = train['idhogar'][train['parentesco1']==1]
without_family_head = train['idhogar'][train['parentesco1']!=1]
hhold_nohead = train['idhogar'][train['idhogar'].isin(without_family_head) & ~trai
hhold_nohead
```

Out[10]:
```
4935    09b195e7a
4975    896fe6d3e
5391    61c10e099
5396    374ca5a19
6443    bfd5067c2
6444    bfd5067c2
7086    1367ab31d
7438    6b1b2405f
7439    6b1b2405f
7440    6b1b2405f
7461    f2bfa75c4
7462    f2bfa75c4
7463    f2bfa75c4
7705    03c6bdf85
7706    03c6bdf85
7756    ad687ad89
7757    b1f4d89d7
8431    c0c8a5013
8432    c0c8a5013
8433    c0c8a5013
8636    a0812ef17
9489    d363d9183
9497    1bc617b23
Name: idhogar, dtype: object
```

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

In [11]:
```
poverty_lvl_head = poverty_lvl[['idhogar', 'Target']][poverty_lvl['parentesco1']==1
poverty_lvl_head
```

Out[11]:

	idhogar	Target
412	5c3f7725d	3
604	daafc1281	3
1374	bcaa2e2f5	4
1599	efd3aec61	2
1692	3c6973219	4
3250	a20ff33ba	2
3498	6bcf799cf	1
3989	d9b1558b5	1
4510	8bb6da3c1	3
4813	2cb443214	3
5068	694a0cbf4	2
5511	15a891635	1
5538	6a389f3de	1
6067	bd82509d1	2
6332	46af47063	1
6427	6c543442a	2
6549	17fb04a62	2
6592	513adb616	3
6635	dfb966eec	1
6693	30a70901d	2
6825	7c57f8237	1
7314	54118d5d9	3
7346	0f3e65c83	1
7481	03f4e5f4d	1
7523	309fb7246	3
7612	564eab113	1
7627	8242a51ec	2
7670	a94a45642	2
8260	55a662731	2
8322	e17b252ed	3
8551	d64524b6b	4
8634	2c9872b82	1
8795	f94589d38	1
9026	654ef7612	2
9104	cc971b690	2

In [12]:
```python
for i in poverty_lvl_head.index:
    for ix in train.index:
        if train.at[ix, 'idhogar'] == poverty_lvl_head.at[i, 'idhogar']:
            train.at[ix,'Target'] = poverty_lvl_head.at[i, 'Target']

train
```

Out[12]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQI
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
9552	ID_d45ae367d	80000.0	0	6	0	1	1	0	NaN	0	...	
9553	ID_c94744e07	80000.0	0	6	0	1	1	0	NaN	0	...	
9554	ID_85fc658f8	80000.0	0	6	0	1	1	0	NaN	0	...	
9555	ID_ced540c61	80000.0	0	6	0	1	1	0	NaN	0	...	
9556	ID_a38c64491	80000.0	0	6	0	1	1	0	NaN	0	...	

9557 rows × 143 columns

Count how many null values are existing in columns

In [13]:
```python
train.isnull().sum().sort_values(ascending=False)[:10]
```

Out[13]:
```
rez_esc          7928
v18q1            7342
v2a1             6860
SQBmeaned           5
meaneduc            5
Id                  0
hogar_adul          0
parentesco10        0
parentesco11        0
parentesco12        0
dtype: int64
```

In [14]:
```python
# Dropping columns with more null values
train_new = train.drop(['rez_esc', 'v18q1', 'v2a1'], axis=1)
train_new.dropna(inplace=True)
train_new.isnull().sum().sort_values(ascending=False)[:10]
```

Out[14]:
```
Id              0
hogar_total     0
parentesco11    0
parentesco12    0
idhogar         0
hogar_nin       0
hogar_adul      0
hogar_mayor     0
dependency      0
parentesco9     0
dtype: int64
```

In [15]:
```
train_new['Target'].isna().sum().any()
```

Out[15]:
```
False
```

In [16]:
```
# Drop unnecessary columns
train_new = train_new.drop(columns=['Id', 'idhogar', 'dependency', 'edjefe', 'edje
train_new.shape
```

Out[16]:
```
(9552, 135)
```

There are no null values in the dataset

Predict the accuracy using random forest classifier

In [17]:
```
X = train_new.drop('Target', axis=1)
y = train_new['Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print(f'Train X {X_train.shape}, Train y {y_train.shape}')
print(f'Test X {X_test.shape}, Test y {y_test.shape}')
```

```
Train X (7641, 134), Train y (7641,)
Test X (1911, 134), Test y (1911,)
```

In [18]:
```
rfc_model = RandomForestClassifier(n_estimators=15, criterion='entropy', max_depth
rfc_model.fit(X_train, y_train)
```

Out[18]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=9, n_estimators=15,
                       random_state=10)
```

In [19]:
```
print("Training results:\n")
y_pred = rfc_model.predict(X_train)
accuracy = accuracy_score(y_train, y_pred)
print("Model accuracy", accuracy)

print(classification_report(y_train, y_pred))
```

```
Training results:

Model accuracy 0.806831566548881
              precision    recall  f1-score   support

           1       0.99      0.52      0.68       624
           2       0.90      0.55      0.69      1250
           3       0.99      0.37      0.54       958
           4       0.77      1.00      0.87      4809

    accuracy                           0.81      7641
   macro avg       0.91      0.61      0.69      7641
weighted avg       0.84      0.81      0.78      7641
```

In [20]:

```
print("Testing results:\n")
y_pred = rfc_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Model accuracy", accuracy)

print(classification_report(y_test, y_pred))
```

```
Testing results:

Model accuracy 0.7514390371533228
              precision    recall  f1-score   support

           1       0.97      0.39      0.55       150
           2       0.82      0.42      0.55       327
           3       0.97      0.25      0.40       248
           4       0.73      0.99      0.84      1186

    accuracy                           0.75      1911
   macro avg       0.87      0.51      0.59      1911
weighted avg       0.79      0.75      0.71      1911
```

Check the accuracy using random forest with cross-validation

In [21]:

```
cv_score = cross_val_score(rfc_model, X, y,scoring='accuracy', cv=5)
print(cv_score)
print("Scores mean", cv_score.mean())
```

```
[0.66248038 0.64050235 0.64450262 0.58376963 0.60052356]
Scores mean 0.6263557086144969
```