# Assessment
# Income Qualification

**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.

**Actions to Perform**

- 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.

## Problem Approach

1. **EDA Understanding the data.**

   File train.csv contains 9557 data points and 143 features, and test.csv contains 23856 data points and 142 features.

[2]:
```
train_data = pd.read_csv('train.csv')
train_data.head()
```

[2]:

| | Id | v2a1 | hacdor | rooms | hacapo | v14a | refrig | v18q | v18q1 | r4h1 | ... | SQBescolari | SQBage | SQBhogar_total | SQBedjefe | SQBhogar_nin | SQBovercrowding | SQBdependency | SQBmeaned | agesq | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ID_279628684 | 190000.0 | 0 | 3 | 0 | 1 | 1 | 0 | NaN | 0 | ... | 100 | 1849 | 1 | 100 | 0 | 1.000000 | 0.0 | 100.0 | 1849 | 4 |
| 1 | ID_f29eb3ddd | 135000.0 | 0 | 4 | 0 | 1 | 1 | 1 | 1.0 | 0 | ... | 144 | 4489 | 1 | 144 | 0 | 1.000000 | 64.0 | 144.0 | 4489 | 4 |
| 2 | ID_68de51c94 | NaN | 0 | 8 | 0 | 1 | 1 | 0 | NaN | 0 | ... | 121 | 8464 | 1 | 0 | 0 | 0.250000 | 64.0 | 121.0 | 8464 | 4 |
| 3 | ID_d671db89c | 180000.0 | 0 | 5 | 0 | 1 | 1 | 1 | 1.0 | 0 | ... | 81 | 289 | 16 | 121 | 4 | 1.777778 | 1.0 | 121.0 | 289 | 4 |
| 4 | ID_d56d6f5f5 | 180000.0 | 0 | 5 | 0 | 1 | 1 | 1 | 1.0 | 0 | ... | 121 | 1369 | 16 | 121 | 4 | 1.777778 | 1.0 | 121.0 | 1369 | 4 |

5 rows × 143 columns

[6]:
```
train_data.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
```

[7]:
```
test_data.info()
```

```
<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
```

There is only float values null in the dataset.

[9]:
```
floatdata_nullcount = train_data.select_dtypes('float64').isnull().sum()
floatdata_nullcount[floatdata_nullcount > 0]
```

[9]:
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

[10]:
```
intdata_nullcount = train_data.select_dtypes('int64').isnull().sum()
intdata_nullcount[intdata_nullcount > 0]
```

[10]:
```
Series([], dtype: int64)
```

[11]:
```
objdata_nullcount = train_data.select_dtypes('object').isnull().sum()
objdata_nullcount[objdata_nullcount > 0]
```

[11]:
```
Series([], dtype: int64)
```

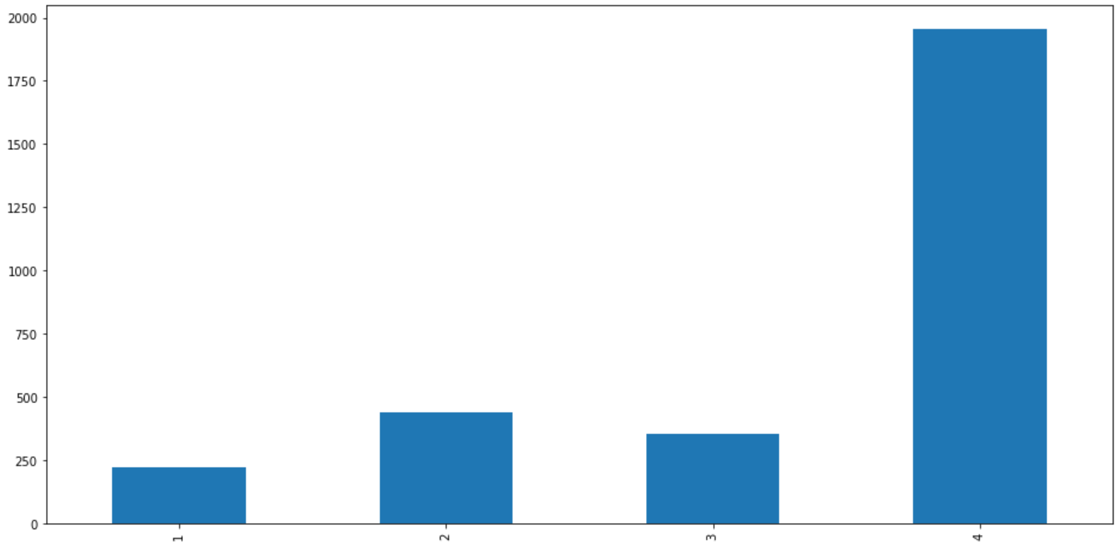
## 2. Target and Feature Analysis

- Object values are converted into np.float.

```
[13]: 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)

[14]: train_data['edjefe'].unique()

[14]: 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)

[15]: train_data['edjefa'].unique()

[15]: 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)

[16]: mapping = {'yes':1, 'no':0}
      train_data['dependency'] = train_data['dependency'].replace(mapping).astype(np.
       ↪float64)
      train_data['edjefa'] = train_data['edjefa'].replace(mapping).astype(np.float64)
      train_data['edjefe'] = train_data['edjefe'].replace(mapping).astype(np.float64)

[17]: test_data['dependency'] = test_data['dependency'].replace(mapping).astype(np.
       ↪float64)
      test_data['edjefa'] = test_data['edjefa'].replace(mapping).astype(np.float64)
      test_data['edjefe'] = test_data['edjefe'].replace(mapping).astype(np.float64)

[18]: train_data[['dependency', 'edjefe', 'edjefa']].describe()

[18]:        dependency       edjefe       edjefa
      count  9557.000000  9557.000000  9557.000000
      mean      1.149550     5.096788     2.896830
      std       1.605993     5.246513     4.612056
      min       0.000000     0.000000     0.000000
      25%       0.333333     0.000000     0.000000
      50%       0.666667     6.000000     0.000000
      75%       1.333333     9.000000     6.000000
      max       8.000000    21.000000    21.000000
```

- Dataset is more biased to level4.

Datapoints vs Target



- Same Households with different Poverty level.

[28]:
```
same_household_target = train_data.groupby('idhogar')['Target'].apply(lambda x:␣
↪x.nunique() == 1)

household_traget_not_same = same_household_target[same_household_target != True]
len(household_traget_not_same)
```

[28]: 85

[29]:
```
train_data[train_data['idhogar'] == household_traget_not_same.
↪index[0]][['idhogar', 'parentesco1', 'Target']]
```

[29]:

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

- Households without Family Head.

[30]:
```
#Household with the family head
households_head = train_data.groupby('idhogar')['parentesco1'].sum()

#Households without family head
households_without_head = train_data.loc[train_data['idhogar'].
↪isin(households_head[households_head == 0].index), :]

print(households_without_head['idhogar'].nunique())
```

```
15
```

[31]:
```
households_without_head_diff_target = households_without_head.
↪groupby('idhogar')['Target'].apply(lambda x: x.nunique() == 1)
sum(households_without_head_diff_target == False)
```

[31]: 0

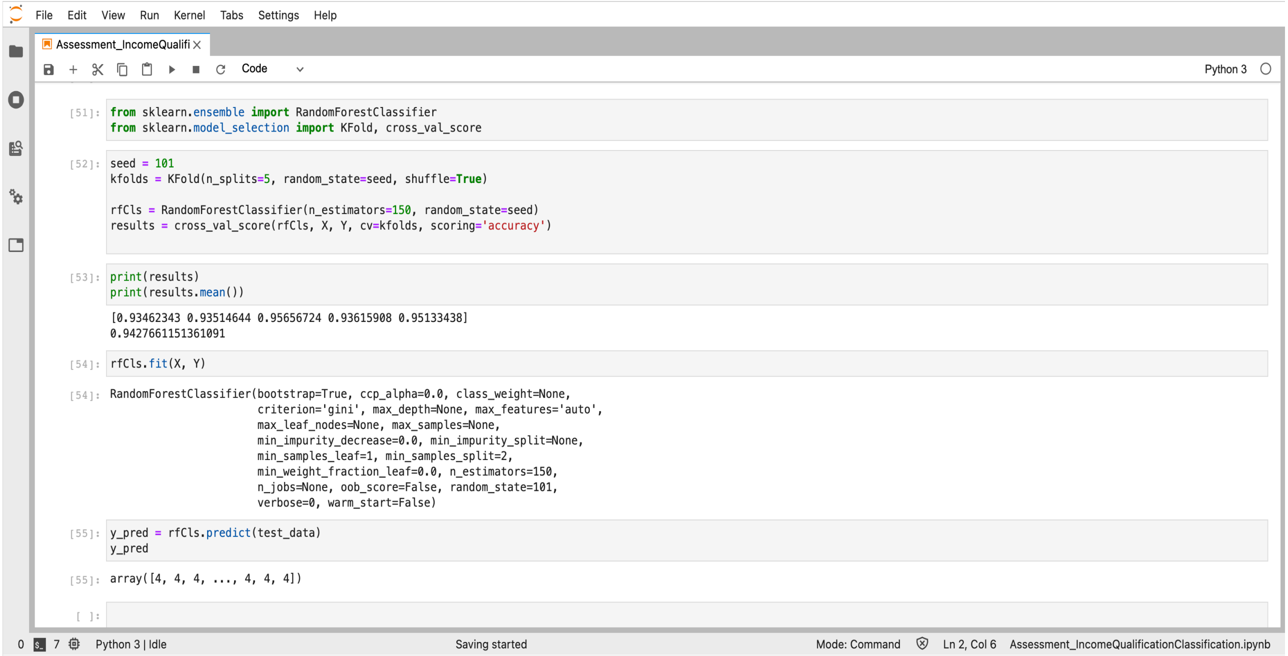
- Setting poverty level of the members and the head of the house within a family.

[33]:
```
#Set poverty level of the members and the head of the house within a family.

for household in household_traget_not_same.index:
    target_hoh = int(train_data[(train_data['idhogar'] == household) &␣
↪(train_data['parentesco1'] == 1.0)]['Target'])
#     print(target_hoh)
    train_data.loc[train_data['idhogar'] == household, 'Target'] = target_hoh

same_household_target = train_data.groupby('idhogar')['Target'].apply(lambda x:␣
↪x.nunique() == 1)

household_traget_not_same = same_household_target[same_household_target != True]
len(household_traget_not_same)
```

[33]: 0

## 3. Model Training

- RandomForestClassifier with Cross Validation.



```
[51]: from sklearn.ensemble import RandomForestClassifier
      from sklearn.model_selection import KFold, cross_val_score

[52]: seed = 101
      kfolds = KFold(n_splits=5, random_state=seed, shuffle=True)

      rfCls = RandomForestClassifier(n_estimators=150, random_state=seed)
      results = cross_val_score(rfCls, X, Y, cv=kfolds, scoring='accuracy')

[53]: print(results)
      print(results.mean())

      [0.93462343 0.93514644 0.95656724 0.93615908 0.95133438]
      0.9427661151361091

[54]: rfCls.fit(X, Y)

[54]: 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=150,
                             n_jobs=None, oob_score=False, random_state=101,
                             verbose=0, warm_start=False)

[55]: y_pred = rfCls.predict(test_data)
      y_pred

[55]: array([4, 4, 4, ..., 4, 4, 4])

[ ]:
```

Average accuracy: 0.9427661151361091

## Conclusion

RandomForestClassifier, the accuracy is 0. 9427661151361091

The above model is used for predicting test data as well

```
[55]: y_pred = rfCls.predict(test_data)
      y_pred

[55]: array([4, 4, 4, …, 4, 4, 4])

[ ]:
```