Project 2 - Income Qualification

ADITHYA M.N.

Problem Statement:

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

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 need to 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.
```

Solution:

In [1]:

```
# Importing
import os
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
import collections
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import KFold,cross_val_score,cross_validate
```

In [2]:

```
# Import the dataset
df=pd.read_csv('train.csv')
print('Shape of the data',df.shape)
print()
print(df.head())
```

```
Shape of the data (9557, 143)

             Id      v2a1  hacdor  rooms  hacapo  v14a  refrig  v18q  v18q1  \
0  ID_279628684  190000.0       0      3       0     1       1     0    NaN
1  ID_f29eb3ddd  135000.0       0      4       0     1       1     1    1.0
2  ID_68de51c94       NaN       0      8       0     1       1     0    NaN
3  ID_d671db89c  180000.0       0      5       0     1       1     1    1.0
4  ID_d56d6f5f5  180000.0       0      5       0     1       1     1    1.0

   r4h1  ...  SQBescolari  SQBage  SQBhogar_total  SQBedjefe  SQBhogar_nin  \
0     0  ...          100    1849               1        100             0
1     0  ...          144    4489               1        144             0
2     0  ...          121    8464               1          0             0
3     0  ...           81     289              16        121             4
4     0  ...          121    1369              16        121             4

   SQBovercrowding  SQBdependency  SQBmeaned  agesq  Target
0         1.000000            0.0      100.0   1849       4
1         1.000000           64.0      144.0   4489       4
2         0.250000           64.0      121.0   8464       4
3         1.777778            1.0      121.0    289       4
4         1.777778            1.0      121.0   1369       4

[5 rows x 143 columns]
```

Understanding the Data given.

In [3]:

```
#Understanding the Data given to us at hand.
datatyp_df = df.dtypes.reset_index()
datatyp_df.columns = ['col_name','col_type']
datatyp_df.groupby('col_type').size()
# We can infer that there are 3 type of Data in the given dataframe from above.
```

Out[3]:

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

Checking For Biases

In [4]:

```
#We can understand the biases by looking at the targets if the given dataset.
df.Target.value_counts()
#From this we can undertand the difference in cases which suggests BIAS
```

Out[4]:

```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```

In [5]:

```
df.columns
#From this we can infer that the total columns in this dataset is 143
```

Out[5]:

```
Index(['Id', 'v2a1', 'hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q',
       'v18q1', 'r4h1',
       ...
       'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe', 'SQBhogar_nin',
       'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq', 'Target'],
      dtype='object', length=143)
```

Pre - Processing

In [6]:

```
#Finding all the columns which have null values in them.
#The Goal is to remove them because sklearn does not accept null values to train any dataset.
null_columns=df.columns[df.isnull().any()]
df[null_columns].isnull().sum()
```

Out[6]:

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

In [7]:

```
print ('Percentage of null values in v2a1 : ', df['v2a1'].isnull().sum()/df.shape[0]*100)
print ('Percentage of null values in v18q1 : ', df['v18q1'].isnull().sum()/df.shape[0]*100)
print ('Percentage of null values in rez_esc : ', df['rez_esc'].isnull().sum()/df.shape[0]*100)
print ('Percentage of null values in meaneduc : ', df['meaneduc'].isnull().sum()/df.shape[0]*100)
print ('Percentage of null values in SQBmeaned : ', df['SQBmeaned'].isnull().sum()/df.shape[0]*100)
```

```
Percentage of null values in v2a1 :  71.7798472323951
Percentage of null values in v18q1 :  76.82327090091033
Percentage of null values in rez_esc :  82.95490216595167
Percentage of null values in meaneduc :  0.05231767290990897
Percentage of null values in SQBmeaned :  0.05231767290990897
```

In [8]:

```
#Removing the Columns which have more than 50% null values
#We can say that these are not immportant to the data.
df= df.drop(['v2a1','v18q1','rez_esc'],axis=1)
print(df.shape)
#Notice the shape has become 140 from 143
#This is because we removed the columnns with more than 50% Null values
```

```
(9557, 140)
```

In [9]:

```
#Imputing the meaneduc & SQBmeaned coumns
#We are imputing the data because as already mentioned we have to change the null values  to zero.
imp = SimpleImputer(missing_values=np.nan, strategy='median')
imp.fit(df[['meaneduc','SQBmeaned']])
df[['meaneduc','SQBmeaned']]=imp.transform(df[['meaneduc','SQBmeaned']])
df[['meaneduc','SQBmeaned']].isnull().sum()
```

Out[9]:

```
meaneduc     0
SQBmeaned    0
dtype: int64
```

In [10]:

```
df= df.drop(['Id'],axis=1)
df.describe(include='O')
```

Out[10]:

	idhogar	dependency	edjefe	edjefa
count	9557	9557	9557	9557
unique	2988	31	22	22
top	fd8a6d014	yes	no	no
freq	13	2192	3762	6230

In [11]:

```
df.dependency = df.dependency.replace(to_replace=['yes','no'],value=[0.5,0]).astype('float')
median1=np.median(df.edjefe[df.edjefe.isin(['yes','no'])==False].astype('float'))
df.edjefe = df.edjefe.replace(to_replace=['yes','no'],value=[median1,0]).astype('float')
median2 = np.median(df.edjefa[df.edjefa.isin(['yes','no'])==False].astype('float'))
df.edjefa = df.edjefa.replace(to_replace=['yes','no'],value=[median2,0]).astype('float')
```

In [12]:

```
df.describe(include='O')
```

Out[12]:

	idhogar
count	9557
unique	2988
top	fd8a6d014
freq	13

In [13]:

```
print(df.idhogar.nunique()) #Returns the number of unique values in the colum idhogar
```

```
2988
```

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

In [14]:

```
df['idhogar']
```

Out[14]:

```
0       21eb7fcc1
1       0e5d7a658
2       2c7317ea8
3       2b58d945f
4       2b58d945f
          ...
9552    d6c086aa3
9553    d6c086aa3
9554    d6c086aa3
9555    d6c086aa3
9556    d6c086aa3
Name: idhogar, Length: 9557, dtype: object
```

In [15]:

```
len(df['idhogar'].unique())
```

Out[15]:

```
2988
```

In [16]:

```
#Grouping by idhogar we can find the the number of families
poverty_level=(df.groupby('idhogar')['Target'].nunique()>1).index
print(poverty_level)
```

```
Index(['001ff74ca', '003123ec2', '004616164', '004983866', '005905417',
       '006031de3', '006555fe2', '00693f597', '006b64543', '00941f1f4',
       ...
       'ff250fd6c', 'ff31b984b', 'ff38ddef1', 'ff6d16fd0', 'ff703eed4',
       'ff9343a35', 'ff9d5ab17', 'ffae4a097', 'ffe90d46f', 'fff7d6be1'],
      dtype='object', name='idhogar', length=2988)
```

Checking if there is a house without a family head.

In [17]:

```
no_head=(df.groupby('idhogar')['parentesco1'].sum()==0).index;
print('Number of families without a house head = {} .'.format(no_head))
```

```
Number of families without a house head = Index(['001ff74ca', '003123ec2', '004616164', '004983866',
'005905417',
       '006031de3', '006555fe2', '00693f597', '006b64543', '00941f1f4',
       ...
       'ff250fd6c', 'ff31b984b', 'ff38ddef1', 'ff6d16fd0', 'ff703eed4',
       'ff9343a35', 'ff9d5ab17', 'ffae4a097', 'ffe90d46f', 'fff7d6be1'],
      dtype='object', name='idhogar', length=2988) .
```

Setting the poverty level of the members and the head of the house as same in a family.

In [18]:

```
target_mean=df.groupby('idhogar')['Target'].mean().astype('int64').reset_index().rename(columns={'Target':'Target
_mean'})
df = df.merge(target_mean,how='left',on='idhogar')
df.Target=df.Target_mean
df.drop('Target_mean',axis=1,inplace=True)
df.head()
```

Out[18]:

	hacdor	rooms	hacapo	v14a	refrig	v18q	r4h1	r4h2	r4h3	r4m1	...	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	SQBhoga
0	0	3	0	1	1	0	0	1	1	0	...	100	1849	1	100	
1	0	4	0	1	1	1	0	1	1	0	...	144	4489	1	144	
2	0	8	0	1	1	0	0	0	0	0	...	121	8464	1	0	
3	0	5	0	1	1	1	0	2	2	1	...	81	289	16	121	
4	0	5	0	1	1	1	0	2	2	1	...	121	1369	16	121	

5 rows × 139 columns

In [19]:

```
df = df.drop(['idhogar'],axis=1)
df.shape
```

Out[19]:

```
(9557, 138)
```

Initialising

In [20]:

```
X = df.drop(['Target'],axis=1)
print('shape of the x',X.shape)
y = df.Target
print('shape of the y',y.shape)
```

```
shape of the x (9557, 137)
shape of the y (9557,)
```

Deploying Random Forest Classifier.

In [21]:

```
#A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of th
e dataset
#and uses averaging to improve the predictive accuracy and control over-fitting.
#This Classifier is highly recommended due to its high accuracy and ability to not overfit the data.
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=10)
Rand_Forest_Class = RandomForestClassifier(n_estimators=10)
Rand_Forest_Class.fit(X_train,y_train)
Prediction = Rand_Forest_Class.predict(X_test)
```

Check the accuracy using Random Forest Classifier

In [22]:

```
print('Accuracy score: ', accuracy_score(Prediction,y_test))
print('Confusion Matrix')
print(confusion_matrix(Prediction,y_test))
print('Classification Report')
print(classification_report(Prediction,y_test))
```

```
Accuracy score:  0.9048117154811716
Confusion Matrix
[[ 131    4    3    8]
 [   8  257   10   14]
 [   1    6  164   10]
 [  29   50   39 1178]]
Classification Report
              precision    recall  f1-score   support

           1       0.78      0.90      0.83       146
           2       0.81      0.89      0.85       289
           3       0.76      0.91      0.83       181
           4       0.97      0.91      0.94      1296

    accuracy                           0.90      1912
   macro avg       0.83      0.90      0.86      1912
weighted avg       0.91      0.90      0.91      1912
```

Using KFold Crossvalidation to validate the performance of the designed classifier.

In [23]:

```
Kfold = KFold(n_splits=10 , random_state=1 , shuffle =True)
Result =  cross_val_score(estimator=Rand_Forest_Class,
                              X=X,
                              y=y,
                              cv=Kfold,
                              scoring='accuracy')


print('K-Fold accuracy scores : \n', Result)
print('Mean score : \n', Result.mean())
```

```
K-Fold accuracy scores :
 [0.91945607 0.94979079 0.90899582 0.92887029 0.93096234 0.93410042
 0.94769874 0.92041885 0.92984293 0.93612565]
Mean score :
 0.9306261911542422
```

Hence we have verified with KFold CrossValidation that the classifier classifies the data with an accuracy of 93%