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.

Following actions should be performed:

-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.

Importing necessary packages

In [1]:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import seaborn as sns
seed=5000

import warnings
warnings.filterwarnings('ignore')
```

Identifying the output variable

In [2]:
```
# loading the dataframe
df_inc =pd.read_csv('train_IQ.csv')
```

In [3]:
```
df_inc.head()
```

Out[3]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBescolari	S
0	ID_279628684	190000.0	0	3	0	1	1	0	NaN	0	...	100	
1	ID_f29eb3ddd	135000.0	0	4	0	1	1	1	1.0	0	...	144	
2	ID_68de51c94	NaN	0	8	0	1	1	0	NaN	0	...	121	
3	ID_d671db89c	180000.0	0	5	0	1	1	1	1.0	0	...	81	
4	ID_d56d6f5f5	180000.0	0	5	0	1	1	1	1.0	0	...	121	

5 rows × 143 columns

In [4]:
```
df_inc['Target'].unique()
```

Out[4]:
```
array([4, 2, 3, 1], dtype=int64)
```

Understanding the type of data

In [5]:
```
df_inc.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 [6]:
```
df_inc.select_dtypes('object').head()
```

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

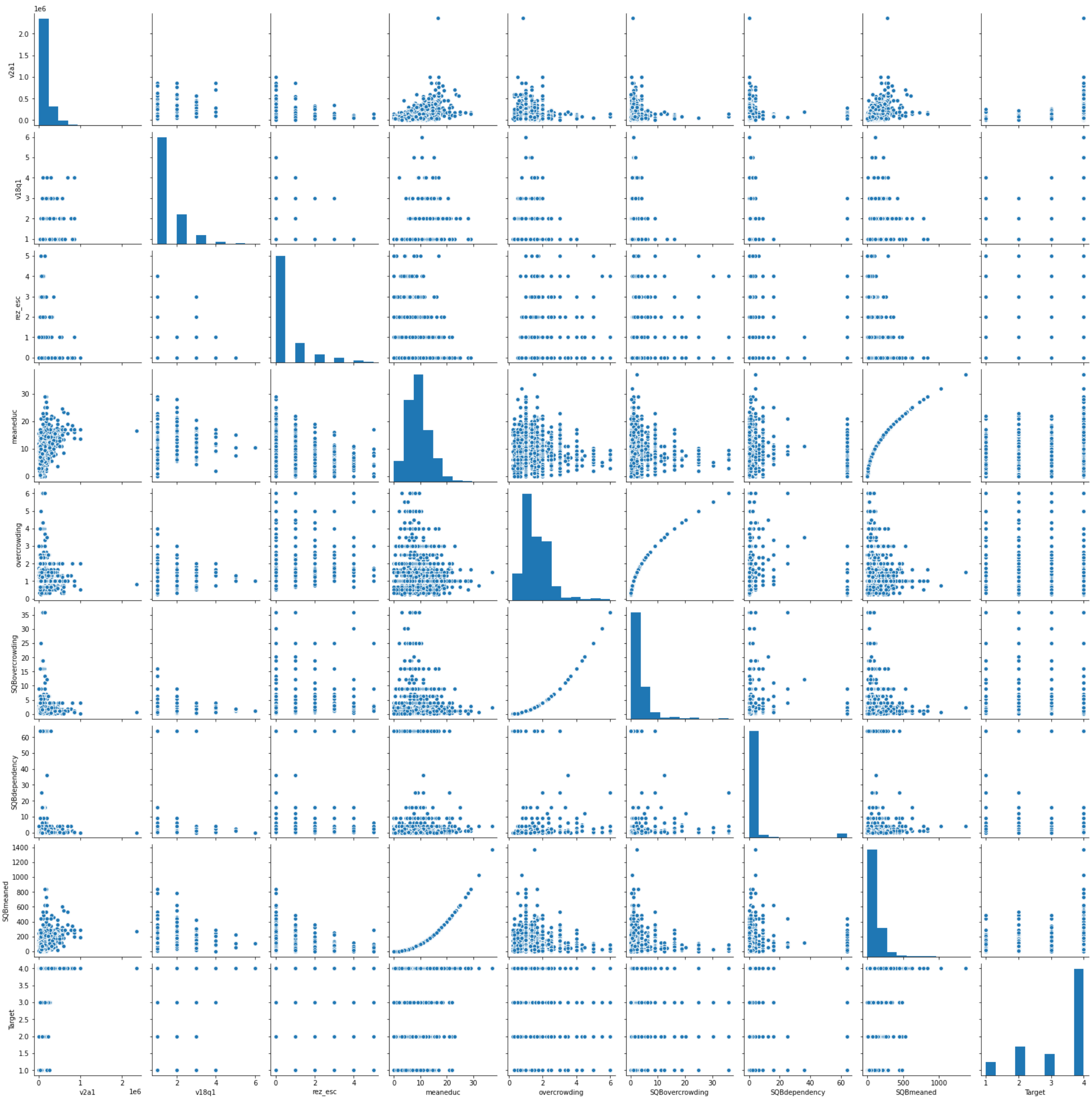
In [7]:
```
allcolumns=df_inc.columns
binary_columns=[]
non_binary=[]
for item in allcolumns:
    if df_inc[item].nunique() ==2:
        binary_columns.append(item)
    else:
        non_binary.append(item)
```

In [8]:

```
df_inc_float=df_inc.select_dtypes('float64')
df_inc_float['Target']=df_inc['Target']
sns.pairplot(df_inc_float)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x193aa168be0>
```



In [9]:

```python
df_inc['dependency'].value_counts()
```

Out[9]:

```
yes          2192
no           1747
.5           1497
2             730
1.5           713
.33333334     598
.66666669     487
8             378
.25           260
3             236
4             100
.75            98
.2             90
.40000001      84
1.3333334      84
2.5            77
5              24
3.5            18
.80000001      18
1.25           18
2.25           13
.71428573      12
1.75           11
.22222222      11
.83333331      11
1.2            11
.2857143        9
.60000002       8
1.6666666       8
6               7
.16666667       7
Name: dependency, dtype: int64
```

In [10]:

```python
def mapping_yes_no(val):
    if val=='yes':
        return 1
    elif val=='no':
        return 0
    else:
        return val
df_inc['dependency']=df_inc['dependency'].map(mapping_yes_no)
```

In [11]:
```
df_inc['dependency'].value_counts()
```

Out[11]:
```
1             2192
0             1747
.5            1497
2              730
1.5            713
.33333334      598
.66666669      487
8              378
.25            260
3              236
4              100
.75             98
.2              90
.40000001       84
1.3333334       84
2.5             77
5               24
.80000001       18
3.5             18
1.25            18
2.25            13
.71428573       12
.83333331       11
1.75            11
.22222222       11
1.2             11
.2857143         9
.60000002        8
1.6666666        8
6                7
.16666667        7
Name: dependency, dtype: int64
```

In [12]:
```
df_inc['edjefe'].value_counts()
```

Out[12]:
```
no      3762
6       1845
11       751
9        486
3        307
15       285
8        257
7        234
5        222
14       208
17       202
2        194
4        137
16       134
yes      123
12       113
10       111
13       103
21        43
18        19
19        14
20         7
Name: edjefe, dtype: int64
```

```
In [13]: df_inc['edjefe']=df_inc['edjefe'].map(mapping_yes_no)
```

```
In [14]: df_inc['edjefa'].value_counts()
```

```
Out[14]: no     6230
         6       947
         11      399
         9       237
         8       217
         15      188
         7       179
         5       176
         3       152
         4       136
         14      120
         16      113
         10       96
         2        84
         17       76
         12       72
         yes      69
         13       52
         21        5
         19        4
         18        3
         20        2
         Name: edjefa, dtype: int64
```

```
In [15]: df_inc['edjefa']=df_inc['edjefa'].map(mapping_yes_no)
         df_inc['edjefa'].value_counts()
```

```
Out[15]: 0     6230
         6      947
         11     399
         9      237
         8      217
         15     188
         7      179
         5      176
         3      152
         4      136
         14     120
         16     113
         10      96
         2       84
         17      76
         12      72
         1       69
         13      52
         21       5
         19       4
         18       3
         20       2
         Name: edjefa, dtype: int64
```

```
In [16]: df_inc['edjefa']=df_inc['edjefa'].astype('int64')
         df_inc['edjefe']=df_inc['edjefe'].astype('int64')
         df_inc['dependency']=df_inc['dependency'].astype('float64')
```

Counting how many null values are existing in columns

In [17]:

```
temp_null_check =df_inc.isnull().sum().reset_index()
nullcolummns =temp_null_check[temp_null_check[0] >0]
nullcolummns
```

Out[17]:

	index	0
1	v2a1	6860
8	v18q1	7342
21	rez_esc	7928
103	meaneduc	5
140	SQBmeaned	5

In [18]:

```
df_inc["v2a1"] = df_inc.groupby("Target").transform(lambda x: x.fillna(x.mean()))
df_inc['v18q1'].fillna(0,inplace=True)
df_inc['rez_esc'].fillna(0,inplace=True)
df_inc['meaneduc'].fillna(0,inplace=True)
df_inc['SQBmeaned'].fillna(0,inplace=True)
mean_val=df_inc['v2a1'].mean()
df_inc['v2a1'].fillna(mean_val,inplace=True)
```

In [19]:

```
temp_null_check =df_inc.isnull().sum().reset_index()
nullcolummns =temp_null_check[temp_null_check[0] >0]
nullcolummns
```

Out[19]:

	index	0

Predicting the accuracy using Random Forest classifier.

In [20]:

```
X=df_inc.select_dtypes(exclude='object')
y=df_inc['Target']
X=X.drop(columns=['Target'],axis=1)
```

In [21]:

```
train_data,test_data, train_label, test_label = train_test_split( X, y, test_size=0.3
, random_state=seed)
classifier = RandomForestClassifier()
classifier.fit(train_data,train_label)
score = classifier.score( test_data,test_label)
print('Random Forest Classifier : ',   score   )
```

```
Random Forest Classifier :  0.9682705718270572
```

In [22]:

```
classifier.get_params
```

Out[22]:

```
<bound method BaseEstimator.get_params of RandomForestClassifier()>
```

In [25]:

```
pd.set_option('display.max_colwidth', -1)
```

Checking the accuracy using Random Forest with Cross Validation.

```
In [24]: %%time
         model_params = {
             'random_forest': {
                 'model': RandomForestClassifier(),
                 'params' : {
                     'n_estimators': [1,5,10],
                     'max_depth': [1,5,10,20],
                     'min_samples_leaf': [1,10,15],
                     'criterion' :['gini','entropy'],
                     'max_features' :['sqrt','log2']
                 }
             }
         }

         import pandas as pd
         scores = []
         for model_name, mp in model_params.items():
             clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
             clf.fit(train_data,train_label)
             scores.append({
                 'model': model_name,
                 'best_score': clf.best_score_,
                 'best_params': clf.best_params_
             })
         df = pd.DataFrame(scores,columns=['model','best_score','best_params'])

         print(df.best_params,df.best_score)
```

```
0    {'criterion': 'gini', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_lea
f': 1, 'n_estimators': 10}
Name: best_params, dtype: object 0    0.936461
Name: best_score, dtype: float64
Wall time: 44.5 s
```