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.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from numext import number_extractor as numext
from scipy.stats import skew
```

In [2]:
```
income=pd.read_csv("../../Dataset/ML/Assignment/Income Qualification/train.csv"
```

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

Out[3]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBesc
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	...	

5 rows × 143 columns

In [4]:
```
income.shape
```

Out[4]:
```
(9557, 143)
```

Identify the output variable.

In [5]:
```
income['Target']
```

Out[5]:
```
0       4
1       4
2       4
3       4
4       4
       ..
9552    2
9553    2
9554    2
9555    2
9556    2
Name: Target, Length: 9557, dtype: int64
```

Understand the type of data.

In [6]:
```
object_type=[x for x in income.columns if income[x].dtypes=='O']
numerical_type=[x for x in income.columns if x not in object_type]
```

In [7]:
```
print('Object_type Column:\n------------------\n',object_type)
print('\nNumerical_type Column:\n----------------------\n',numerical_type)
```

```
Object_type Column:
------------------
 ['Id', 'idhogar', 'dependency', 'edjefe', 'edjefa']

Numerical_type Column:
----------------------
 ['v2a1', 'hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q', 'v18q1', 'r4
h1', 'r4h2', 'r4h3', 'r4m1', 'r4m2', 'r4m3', 'r4t1', 'r4t2', 'r4t3', 'tamho
g', 'tamviv', 'escolari', 'rez_esc', 'hhsize', 'paredblolad', 'paredzocalo',
'paredpreb', 'pareddes', 'paredmad', 'paredzinc', 'paredfibras', 'paredothe
r', 'pisomoscer', 'pisocemento', 'pisoother', 'pisonatur', 'pisonotiene', 'pi
somadera', 'techozinc', 'techoentrepiso', 'techocane', 'techootro', 'cieloraz
o', 'abastaguadentro', 'abastaguafuera', 'abastaguano', 'public', 'planpri',
'noelec', 'coopele', 'sanitario1', 'sanitario2', 'sanitario3', 'sanitario5',
'sanitario6', 'energcocinar1', 'energcocinar2', 'energcocinar3', 'energcocina
r4', 'elimbasu1', 'elimbasu2', 'elimbasu3', 'elimbasu4', 'elimbasu5', 'elimba
su6', 'epared1', 'epared2', 'epared3', 'etecho1', 'etecho2', 'etecho3', 'eviv
1', 'eviv2', 'eviv3', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil
2', 'estadocivil3', 'estadocivil4', 'estadocivil5', 'estadocivil6', 'estadoci
vil7', 'parentesco1', 'parentesco2', 'parentesco3', 'parentesco4', 'parentesc
o5', 'parentesco6', 'parentesco7', 'parentesco8', 'parentesco9', 'parentesco1
0', 'parentesco11', 'parentesco12', 'hogar_nin', 'hogar_adul', 'hogar_mayor',
'hogar_total', 'meaneduc', 'instlevel1', 'instlevel2', 'instlevel3', 'instlev
el4', 'instlevel5', 'instlevel6', 'instlevel7', 'instlevel8', 'instlevel9',
'bedrooms', 'overcrowding', 'tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi
4', 'tipovivi5', 'computer', 'television', 'mobilephone', 'qmobilephone', 'lu
gar1', 'lugar2', 'lugar3', 'lugar4', 'lugar5', 'lugar6', 'area1', 'area2', 'a
ge', 'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe', 'SQBhogar_nin',
'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq', 'Target']
```

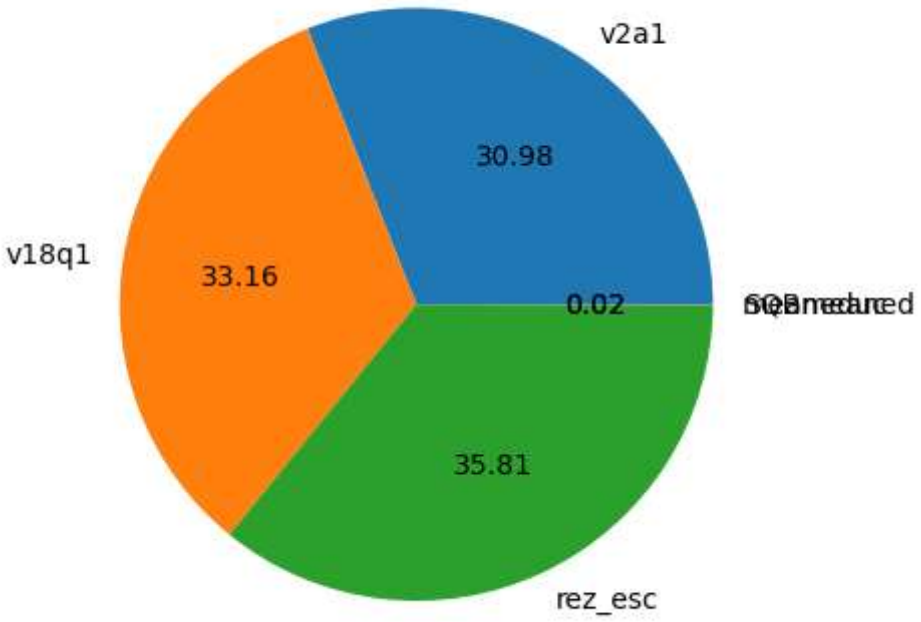
Count how many null values are existing in columns

In [8]:
```
def nan_check(type,cols):
    cols_null=[]
    for col in cols:
        if(income[col].isna().any()):
            cols_null.append(col)
    if(len(cols_null)>0):
        print("The null column of "+type+" are")
        print(cols_null)
    else:
        print("No null column for ",type)
```

In [9]:
```
nan_check("Object",object_type)
print("-------------------------")
nan_check("Numerical",numerical_type)
```

```
No null column for  Object
-------------------------
The null column of Numerical are
['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'SQBmeaned']
```

In [10]:
```
nan_col=income[['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'SQBmeaned']].isna().su
plt.pie(x=nan_col.values,labels=nan_col.index,autopct="%1.2f")
plt.show()
```



Remove null value rows of the target variable

In [11]:
```
# The null value are present only in the numerical column so we can use median
null_col=[x for x in income.columns if income[x].isna().any()==True]
null_col
```

Out[11]:
```
['v2a1', 'v18q1', 'rez_esc', 'meaneduc', 'SQBmeaned']
```

In [12]:
```
impute=SimpleImputer(strategy='median')
```

In [13]:
```
def imputer(data):
    for col in data:
        income[col]=impute.fit_transform(income[[col]])
```

In [14]:
```
imputer(null_col)
```

In [15]:
```
income.isna().sum().any()
```

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

Label Encoding

In [16]:
```
from sklearn.preprocessing import LabelEncoder
```

In [17]:
```
encode=LabelEncoder()
```

In [18]:
```
def encoder(features):
    for data in features:
        income[data]=encode.fit_transform(income[data])
```

In [19]:
```
encoder(object_type)
```

Treating skewness

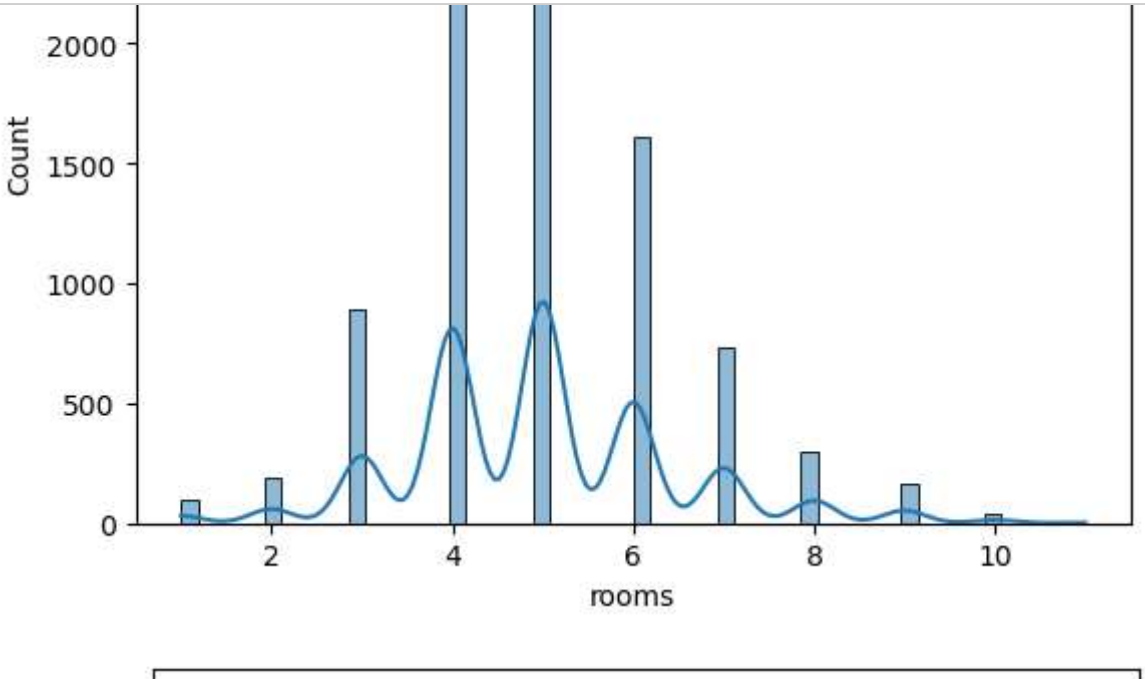
In [20]:
```
income.columns
```

Out[20]:
```
Index(['Id', 'v2a1', 'hacdor', 'rooms', 'hacapo', 'v14a', 'refrig', 'v18q',
       'v18q1', 'r4h1',
       ...
       'SQBescolari', 'SQBage', 'SQBhogar_total', 'SQBedjefe', 'SQBhogar_ni
n',
       'SQBovercrowding', 'SQBdependency', 'SQBmeaned', 'agesq', 'Target'],
      dtype='object', length=143)
```

In [21]:
```
income.dtypes
```

Out[21]:
```
Id                  int32
v2a1              float64
hacdor              int64
rooms               int64
hacapo              int64
                    ...
SQBovercrowding   float64
SQBdependency     float64
SQBmeaned         float64
agesq               int64
Target              int64
Length: 143, dtype: object
```

In [22]:
```
for data in income.columns[1:]:
    sns.histplot(income[data],kde=True)
    plt.show()
```

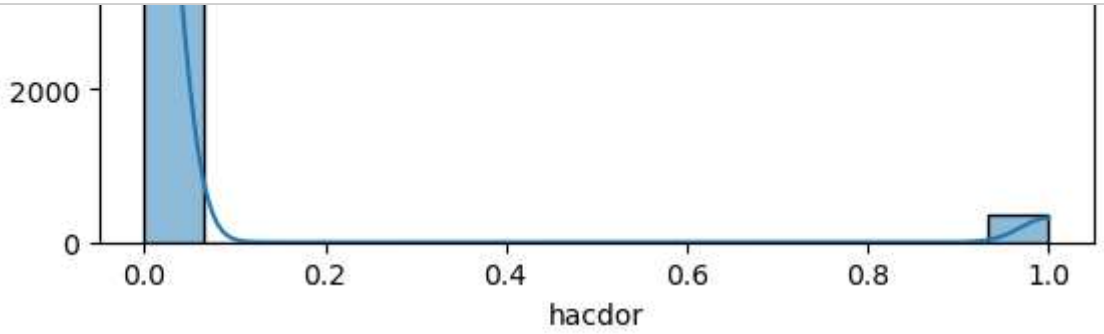


In [23]:
```
positive_skew=[]
negative_skew=[]
for data in income.columns[1:]:
    skw=skew(income[data])
    if(skw<-0.5):
        negative_skew.append(data)
    elif(skw>0.5):
        positive_skew.append(data)
```

In [24]:
```
# Checking continuous data
positive_continuous=[]
for col in positive_skew:
    if(len(income[col].unique())>2):
        positive_continuous.append(col)
```

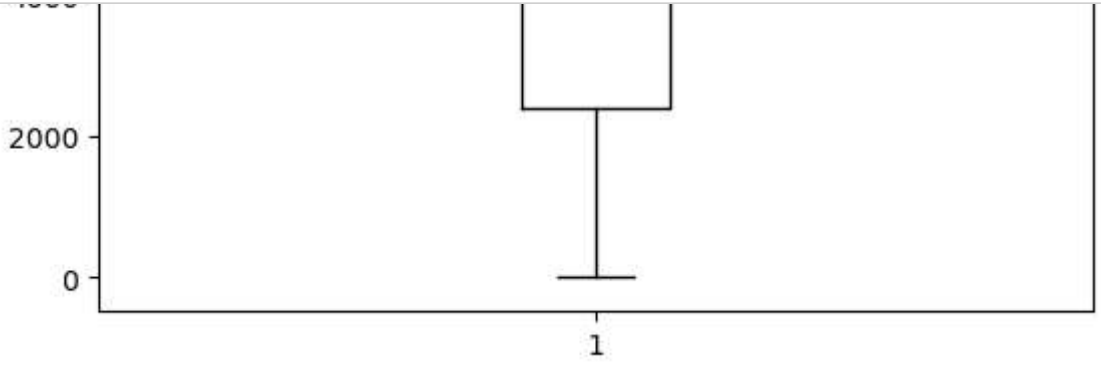
In [25]:
```
# for col in positive_continuous:
#     if(skew(np.log(income[col]))!=np.nan):
#         income[col]=np.log(income[col])
#     else:
#         income[col]=np.sqrt(income[col])
```

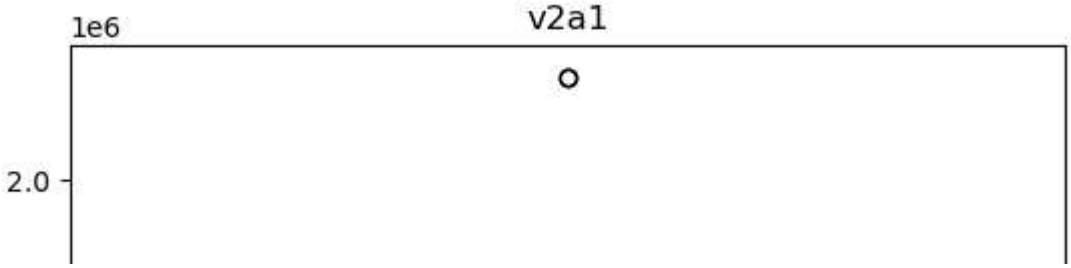
In [26]:
```
for col in positive_skew:
    sns.histplot(income[col],kde=True)
    plt.show()
```



In [27]:
```
# for col in positive_skew_ref:
#     income[col]=np.log(income[col])
```

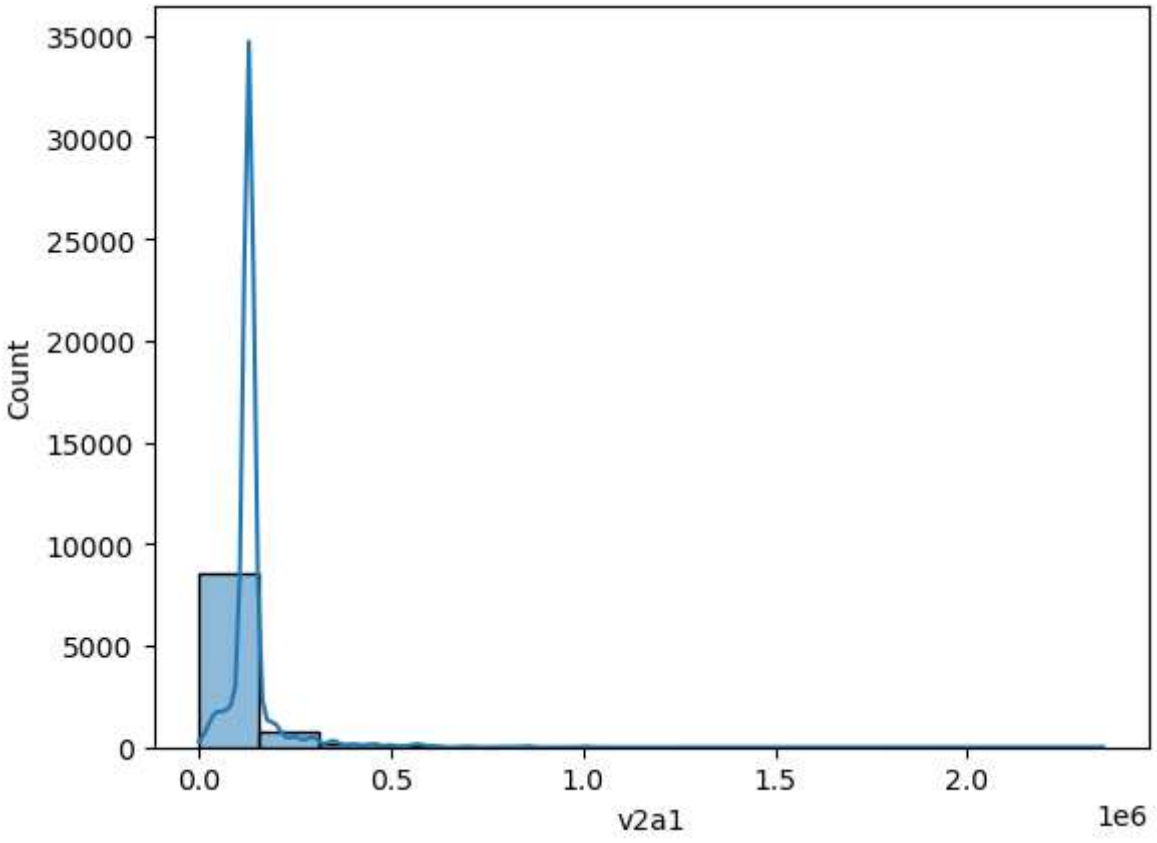
In [28]:
```
for col in income.columns:
    plt.title(col)
    plt.boxplot(income[col])
    plt.show()
```





In [29]:
```
sns.histplot(income['v2a1'],kde=True)
```

Out[29]:
```
<AxesSubplot:xlabel='v2a1', ylabel='Count'>
```



In [30]:
```
income['idhogar'].value_counts()
```

Out[30]:
```
2946    13
2034    12
150     12
2132    11
816     11
        ..
1850     1
1379     1
275      1
18       1
401      1
Name: idhogar, Length: 2988, dtype: int64
```

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

In [31]:
```
# v18q,tipovivi1,tipovivi2,mobilephone
```

In [32]:
```
poverty_label=[]

for indx,col in income.iterrows():
    if(col['v18q']==1 and col['tipovivi1']==1 and col['mobilephone']==1):
        poverty_label.append("Rich")
    else:
        poverty_label.append("Poor")
```

In [33]:
```
income['Poverty_label']=poverty_label
```

In [34]:
```
member_poverty=[]
def count(col):
    print(len(set(col['Poverty_label'])))
    print("-------------")
```

In [35]:
```
income.groupby(['idhogar']).apply(count)
```

```
1
-------------
1
-------------
1
-------------
1
-------------
1
-------------
1
-------------
1
-------------
1
-------------
1
-------------
1
```

Check if there is a house without a family head

In [36]:
```
def check_head(col):
    hasHead=False
    for data in col['parentesco1'].unique():
        if(data==1):
            hasHead=True
    if(hasHead==False):
        print("This family with id ",set(col['idhogar']),"  has no family head")
```

In [37]:
```
income.groupby(['idhogar']).apply(check_head)
```

```
This family with id  {38}  has no family head
This family with id  {114}  has no family head
This family with id  {230}  has no family head
This family with id  {331}  has no family head
This family with id  {645}  has no family head
This family with id  {1143}  has no family head
This family with id  {1268}  has no family head
This family with id  {1613}  has no family head
This family with id  {1867}  has no family head
This family with id  {2027}  has no family head
This family with id  {2068}  has no family head
This family with id  {2241}  has no family head
This family with id  {2251}  has no family head
This family with id  {2453}  has no family head
This family with id  {2807}  has no family head
```

Out[37]: —

In [38]:
```
income
```

Out[38]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBage	SQ
0	1523	190000.0	0	3	0	1	1	0	1.0	0	...	1849	
1	9064	135000.0	0	4	0	1	1	1	1.0	0	...	4489	
2	3948	130000.0	0	8	0	1	1	0	1.0	0	...	8464	
3	8017	180000.0	0	5	0	1	1	1	1.0	0	...	289	
4	7984	180000.0	0	5	0	1	1	1	1.0	0	...	1369	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9552	7933	80000.0	0	6	0	1	1	0	1.0	0	...	2116	
9553	7539	80000.0	0	6	0	1	1	0	1.0	0	...	4	
9554	5023	80000.0	0	6	0	1	1	0	1.0	0	...	2500	
9555	7734	80000.0	0	6	0	1	1	0	1.0	0	...	676	
9556	6102	80000.0	0	6	0	1	1	0	1.0	0	...	441	

9557 rows × 144 columns

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

In [39]:
```
income['Poverty_label'].value_counts()
```

Out[39]:
```
Poor    8235
Rich    1322
Name: Poverty_label, dtype: int64
```

In [40]:
```
# 4 for rich and family head
# 3 for rich and family member
# 2 for poor and family head
# 1 for poor and family member
```

In [ ]:
```
```

In [41]:
```
for indx,col in income.iterrows():
    if((col['Poverty_label']=="Rich") and (col['parentesco1']==1)):
        income.loc[indx,'Poverty_level']=4
    elif((col['Poverty_label']=="Rich") and (col['parentesco1']==0)):
        income.loc[indx,'Poverty_level']=3
    elif((col['Poverty_label']=="Poor") and (col['parentesco1']==1)):
        income.loc[indx,'Poverty_level']=2
    elif((col['Poverty_label']=="Poor") and (col['parentesco1']==0)):
        income.loc[indx,'Poverty_level']=1
```

In [42]:
```
income
```

Out[42]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	v18q1	r4h1	...	SQBhogar_tot
0	1523	190000.0	0	3	0	1	1	0	1.0	0	...	
1	9064	135000.0	0	4	0	1	1	1	1.0	0	...	
2	3948	130000.0	0	8	0	1	1	0	1.0	0	...	
3	8017	180000.0	0	5	0	1	1	1	1.0	0	...	
4	7984	180000.0	0	5	0	1	1	1	1.0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
9552	7933	80000.0	0	6	0	1	1	0	1.0	0	...	
9553	7539	80000.0	0	6	0	1	1	0	1.0	0	...	
9554	5023	80000.0	0	6	0	1	1	0	1.0	0	...	
9555	7734	80000.0	0	6	0	1	1	0	1.0	0	...	
9556	6102	80000.0	0	6	0	1	1	0	1.0	0	...	

9557 rows × 145 columns

Predict the accuracy using random forest classifier

In [43]:
```
encoder=LabelEncoder()
```

In [44]:
```
income['Poverty_label']=encoder.fit_transform(income['Poverty_label'])
```

In [45]:
```
income.dtypes.value_counts()
```

Out[45]:
```
int64      130
float64      9
int32        6
dtype: int64
```

In [46]:
```
features=income.drop(['Id','Target'],axis=1)
target=income[['Target']]
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import accuracy_score
```

In [48]:
```
X_train,X_test,y_train,y_test=train_test_split(features,target,test_size=0.25,r
```

In [49]:
```
model=RandomForestClassifier()
```

In [50]:
```
best_params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
grid_search=GridSearchCV(model,param_grid=best_params,verbose=5)
```

In [51]:
```
grid_search.fit(X_train,y_train.values.ravel())
```

```
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
[CV 1/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=50;, score=0.886 total time=
0.4s
[CV 2/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=50;, score=0.894 total time=
0.4s
[CV 3/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=50;, score=0.881 total time=
0.4s
[CV 4/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=50;, score=0.885 total time=
0.4s
[CV 5/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=50;, score=0.899 total time=
0.4s
[CV 1/5] END criterion=gini, max_depth=None, max_features=auto, min_samples
_leaf=1, min_samples_split=2, n_estimators=100;, score=0.893 total time=
0.9s
```

In [52]:
```
grid_search.best_params_
```

Out[52]:
```
{'criterion': 'entropy',
 'max_depth': None,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

In [53]:
```
y_pred=grid_search.predict(X_test)
```

In [54]:
```
accuracy_score(y_test,y_pred)
```

Out[54]:
```
0.9196652719665271
```

In [55]:
```
from sklearn.model_selection import KFold,cross_val_score
```

In [56]:
```
Kf=KFold(n_splits=5,shuffle=False)
```

In [57]:
```
for train_set,test_set in Kf.split(features):
    print('TRAIN SET',train_set)
    print('TEST SET',test_set)
```

```
TRAIN SET [1912 1913 1914 ... 9554 9555 9556]
TEST SET [   0    1    2 ... 1909 1910 1911]
TRAIN SET [   0    1    2 ... 9554 9555 9556]
TEST SET [1912 1913 1914 ... 3821 3822 3823]
TRAIN SET [   0    1    2 ... 9554 9555 9556]
TEST SET [3824 3825 3826 ... 5732 5733 5734]
TRAIN SET [   0    1    2 ... 9554 9555 9556]
TEST SET [5735 5736 5737 ... 7643 7644 7645]
TRAIN SET [   0    1    2 ... 7643 7644 7645]
TEST SET [7646 7647 7648 ... 9554 9555 9556]
```

In [58]:
```
# rf_class=RandomForestClassifier(criterion='entropy',max_depth=None,max_featur
# print(cross_val_score(grid_search,features,target.values.ravel(),scoring='acc
```

In [59]:
```
rf_class=RandomForestClassifier(
  criterion='entropy',
  max_depth=None,
  max_features='sqrt',
  min_samples_leaf=1,
  min_samples_split=2,
  n_estimators=100
)
print(cross_val_score(rf_class,features,target.values.ravel(),scoring='accuracy
```

```
[0.65951883 0.64801255 0.65096808 0.55729984 0.57980115]
```

In [ ]: