My Income Qualification Project

October 9, 2021

[278]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[279]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[280]:
```
train.head()
```

[280]:

	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]

[281]:
```
train.shape
```

[281]: (9557, 143)

[282]:
```
# Output variable
train.Target.value_counts()
```

[282]:
```
4    5996
2    1597
3    1209
1     755
Name: Target, dtype: int64
```

[283]:
```
# Columns containing NA
train.isna().sum()[train.isna().sum() != 0]
```

[283]:
```
v2a1          6860
v18q1         7342
rez_esc       7928
meaneduc         5
SQBmeaned        5
dtype: int64
```

[284]:
```
train.shape
```

[284]:
```
(9557, 143)
```

[287]:
```
train.dtypes[train.dtypes == 'object']
```

[287]:
```
Id             object
idhogar        object
dependency     object
edjefe         object
edjefa         object
dtype: object
```

[288]:
```
mapping = {'yes':1,'no':0}

for df in [train, test]:
    df['dependency'] =df['dependency'].replace(mapping).astype(np.float64)
    df['edjefe'] =df['edjefe'].replace(mapping).astype(np.float64)
    df['edjefa'] =df['edjefa'].replace(mapping).astype(np.float64)

train[['dependency','edjefe','edjefa']].describe()
```

[288]:

	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
```

[289]:
```
train.isna().sum()[train.isna().sum() != 0]/train.shape[0]
```

[289]:
```
v2a1          0.717798
v18q1         0.768233
rez_esc       0.829549
meaneduc      0.000523
SQBmeaned     0.000523
dtype: float64
```

Columns related to Monthly rent payment

- tipovivi1, =1 own and fully paid house
- tipovivi2, "=1 own, paying in installments"
- tipovivi3, =1 rented
- tipovivi4, =1 precarious
- tipovivi5, "=1 other(assigned, borrowed)"

[290]:
```
data = train[train.v2a1.isnull()].head(10)

cols = ['tipovivi1','tipovivi2','tipovivi3','tipovivi4','tipovivi5']
data[cols]
```

[290]:

	tipovivi1	tipovivi2	tipovivi3	tipovivi4	tipovivi5
2	1	0	0	0	0
13	1	0	0	0	0
14	1	0	0	0	0
26	1	0	0	0	0
32	1	0	0	0	0
33	1	0	0	0	0
34	1	0	0	0	0
35	1	0	0	0	0
36	1	0	0	0	0
42	1	0	0	0	0

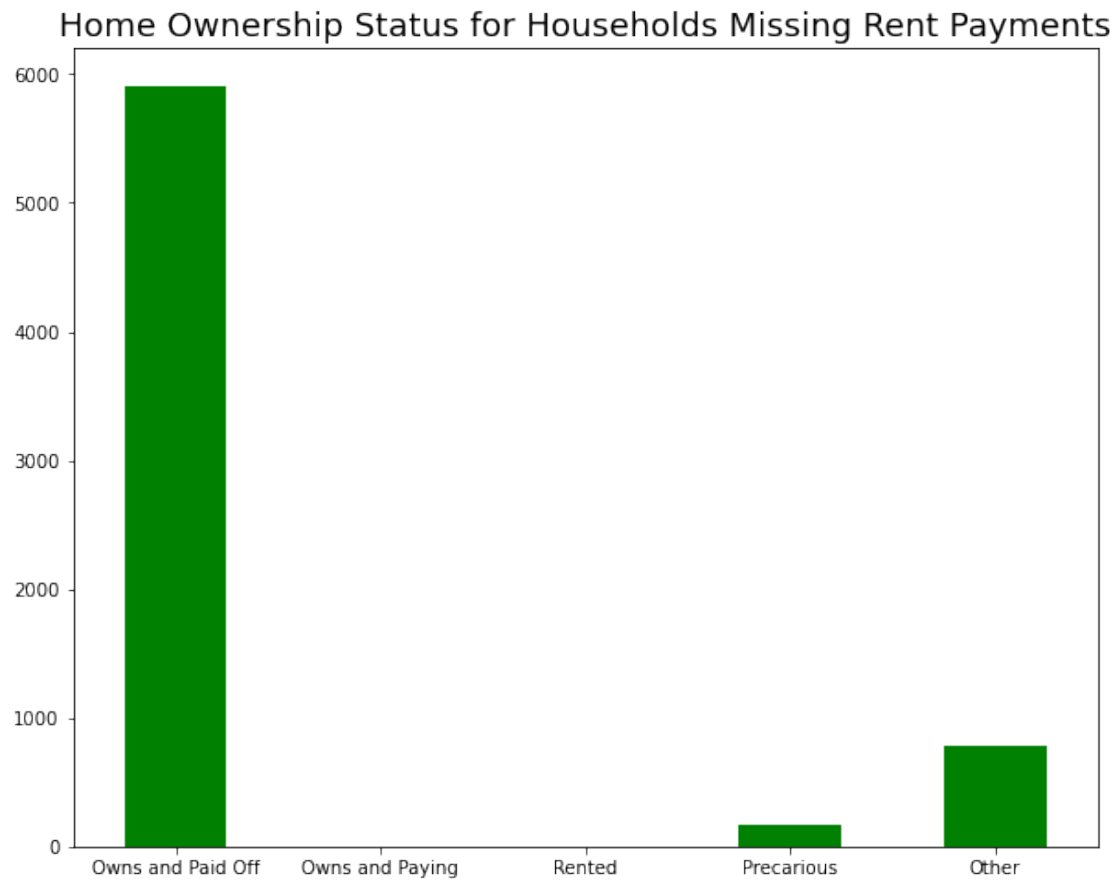
[291]:
```
own_vars = [x for x in train if x.startswith('tipovivi')]

# Plot of the home ownership variables for home missing rent payments.
train.loc[train.v2a1.isnull(),own_vars].sum().plot.bar(figsize = (10,8),color =␣
↪'green')

plt.xticks([0, 1, 2, 3, 4],
           ['Owns and Paid Off', 'Owns and Paying', 'Rented', 'Precarious',␣
↪'Other'], rotation = 0)
```

```
plt.title('Home Ownership Status for Households Missing Rent Payments', size =␣
↪18)
```

[291]:
```
Text(0.5, 1.0, 'Home Ownership Status for Households Missing Rent Payments')
```



[292]:
```
#Looking at the above data it makes sense that when the house is fully paid,␣
↪there will be no monthly rent payment.
#Lets add 0 for all the null values.
for df in [train,test]:
    df.v2a1.fillna(0,inplace = True)

df.v2a1.isna().sum()
```

[292]:
```
0
```

[293]:
```
train.v18q1.value_counts()
```

[293]:
```
1.0    1586
2.0     444
3.0     129
4.0      37
5.0      13
6.0       6
Name: v18q1, dtype: int64
```

[294]:
```
# v18q is owns a tablet and v18q1 is number of tablets household owns. So if␣
↪v18q1 is null means v18q1 owns zero tablet.
for df in [train,test]:
    df.v18q1.fillna(0,inplace = True)

df.v18q1.isna().sum()
```

[294]:
```
0
```

[295]:
```
train.rez_esc.value_counts()
```

[295]:
```
0.0    1211
1.0     227
2.0      98
3.0      55
4.0      29
5.0       9
Name: rez_esc, dtype: int64
```

[296]:
```
train[train['rez_esc'].notnull()]['age'].describe()
```

[296]:
```
count    1629.000000
mean       12.258441
std         3.218325
min         7.000000
25%         9.000000
50%        12.000000
75%        15.000000
max        17.000000
Name: age, dtype: float64
```

[297]:
```
train.loc[train['rez_esc'].isnull()]['age'].describe()
```

[297]:
```
count    7928.000000
mean       38.833249
std        20.989486
min         0.000000
25%        24.000000
50%        38.000000
```

```
75%        54.000000
max        97.000000
Name: age, dtype: float64
```

[298]:
```
train.loc[(train.rez_esc.isnull() & ((train.age > 7) & (train.age <
↪17)))]['age'].describe()
```

[298]:
```
count     1.0
mean     10.0
std       NaN
min      10.0
25%      10.0
50%      10.0
75%      10.0
max      10.0
Name: age, dtype: float64
```

[299]:
```
# there is only 1 null value of rez_esc between the age 7 and 17.

train.loc[(train.rez_esc.isnull() & (train.age == 10))]
```

[299]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	\
2514	ID_f012e4242	160000.0	0	6	0	1	1	1	

	v18q1	r4h1	…	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	\
2514	1.0	0	…	0	100	9	121	

	SQBhogar_nin	SQBovercrowding	SQBdependency	SQBmeaned	agesq	Target
2514	1	2.25	0.25	182.25	100	4

[1 rows x 143 columns]

[300]:
```
train[train['Id'] == 'ID_f012e4242'].head()
```

[300]:

	Id	v2a1	hacdor	rooms	hacapo	v14a	refrig	v18q	\
2514	ID_f012e4242	160000.0	0	6	0	1	1	1	

	v18q1	r4h1	…	SQBescolari	SQBage	SQBhogar_total	SQBedjefe	\
2514	1.0	0	…	0	100	9	121	

	SQBhogar_nin	SQBovercrowding	SQBdependency	SQBmeaned	agesq	Target
2514	1	2.25	0.25	182.25	100	4

[1 rows x 143 columns]

[301]:
```
for df in [train,test]:
    df['rez_esc'].fillna(0,inplace = True)
```

```
df.rez_esc.isnull().sum()
```

[301]:
```
0
```

[302]:
```
train.isna().sum()[train.isna().sum() != 0]
```

[302]:
```
meaneduc     5
SQBmeaned    5
dtype: int64
```

[358]:
```
train.dropna(inplace = True)
test.dropna(inplace = True)
```

[304]:
```
all_equal = train.groupby('idhogar')['Target'].apply(lambda x: x.nunique()==1)
not_equal = all_equal[all_equal == False]

print('There are {} households where all family members do not have the same␣
 ↪target. And this shows that for some households not all members of the house␣
 ↪have the same poverty level'.format(len(not_equal)))
```

```
There are 85 households where all family members do not have the same target.
And this shows that for some households not all members of the house have the
same poverty level
```

[305]:
```
train[train.idhogar == not_equal.index[0]][['idhogar', 'parentesco1', 'Target']]
```

[305]:

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

[306]:
```
household_head = train.groupby('idhogar')['parentesco1'].sum()
```

[307]:
```
len(household_head[household_head == 0])

print('There are {} household without any head.'.
 ↪format(len(household_head[household_head == 0])))
```

```
There are 15 household without any head.
```

[308]:
```
# Find households without a head and where Target value are different
household_without_head = household_head[(household_head == 0) & (all_equal ==␣
 ↪False)]
```

[309]:
```
household_without_head

print('{} Households with no head have different Target value.'.
 ↪format(len(household_without_head)))
```

```
0 Households with no head have different Target value.
```

[310]:
```
# Lets fix the data.
# Set poverty level of the members and the head of the house within a family.
# Iterate through each household.

for household in not_equal.index:
    true_target = int(train[(train['idhogar'] == household) &␣
 ↪(train['parentesco1'] == 1.0)]['Target'])

# Set the correct label for all members in the household
train.loc[train.idhogar == household, 'Target'] = true_target
```

[311]:
```
heads = train.loc[train.parentesco1 == 1.0]
```

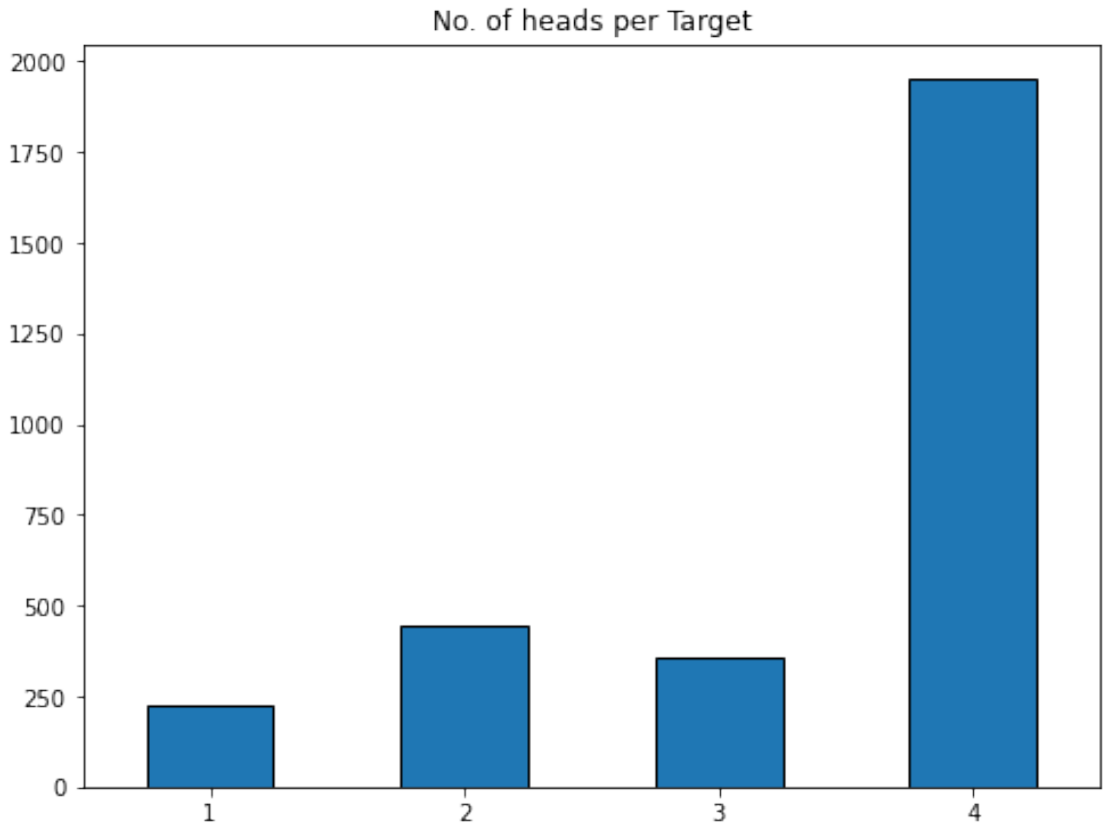
[312]:
```
Target_counts = heads.Target.value_counts().sort_index()
```

[313]:
```
Target_counts
```

[313]:
```
1     222
2     442
3     355
4    1951
Name: Target, dtype: int64
```

[314]:
```
Target_counts.plot.bar(figsize = (8,6), linewidth = 1,edgecolor = 'k', title =␣
 ↪'No. of heads per Target', rot = 0)
```

[314]:
```
<AxesSubplot:title={'center':'No. of heads per Target'}>
```



1 NOTE-

1.0.1 Extreme Poverty is smallest count in the dataset, means data is biased.

[315]:
```
cols = [x for x in train if x.startswith('SQB')]
```

[316]:
```
for df in [train,test]:
    df.drop(columns = cols ,inplace = True)
```

[317]:
```
train.shape
```

[317]: (9552, 135)

[318]:
```
for df in [train,test]:
    df.drop(columns = 'agesq' ,inplace = True)
```

[319]:
```
train.shape
```

[319]:
```
(9552, 134)
```

[320]:
```
id_ = ['Id','idhogar','Target']

ind_bool = ['v18q', 'dis', 'male', 'female', 'estadocivil1', 'estadocivil2',␣
 ↪'estadocivil3',
            'estadocivil4', 'estadocivil5', 'estadocivil6', 'estadocivil7',
            'parentesco1', 'parentesco2',  'parentesco3', 'parentesco4',␣
 ↪'parentesco5',
            'parentesco6', 'parentesco7', 'parentesco8',  'parentesco9',␣
 ↪'parentesco10',
            'parentesco11', 'parentesco12', 'instlevel1', 'instlevel2',␣
 ↪'instlevel3',
            'instlevel4', 'instlevel5', 'instlevel6', 'instlevel7',␣
 ↪'instlevel8',
            'instlevel9', 'mobilephone']

ind_ordered = ['rez_esc', 'escolari', 'age']

hh_bool = ['hacdor', 'hacapo', 'v14a', 'refrig', 'paredblolad', 'paredzocalo',
           'paredpreb','pisocemento', 'pareddes', 'paredmad',
           'paredzinc', 'paredfibras', 'paredother', 'pisomoscer', 'pisoother',
           'pisonatur', 'pisonotiene', 'pisomadera',
           'techozinc', 'techoentrepiso', 'techocane', 'techootro', 'cielorazo',
           'abastaguadentro', 'abastaguafuera', 'abastaguano',
            'public', 'planpri', 'noelec', 'coopele', 'sanitario1',
           'sanitario2', 'sanitario3', 'sanitario5',   'sanitario6',
           'energcocinar1', 'energcocinar2', 'energcocinar3', 'energcocinar4',
           'elimbasu1', 'elimbasu2', 'elimbasu3', 'elimbasu4',
           'elimbasu5', 'elimbasu6', 'epared1', 'epared2', 'epared3',
           'etecho1', 'etecho2', 'etecho3', 'eviv1', 'eviv2', 'eviv3',
           'tipovivi1', 'tipovivi2', 'tipovivi3', 'tipovivi4', 'tipovivi5',
           'computer', 'television', 'lugar1', 'lugar2', 'lugar3',
           'lugar4', 'lugar5', 'lugar6', 'area1', 'area2']

hh_ordered = [ 'rooms', 'r4h1', 'r4h2', 'r4h3', 'r4m1','r4m2','r4m3', 'r4t1', ␣
 ↪'r4t2',
              'r4t3', 'v18q1', 'tamhog','tamviv','hhsize','hogar_nin',
              'hogar_adul','hogar_mayor','hogar_total',  'bedrooms',␣
 ↪'qmobilephone']

hh_cont = ['v2a1', 'dependency', 'edjefe', 'edjefa', 'meaneduc', 'overcrowding']
```

[321]:
```
#Check for redundant household variables
heads = heads[id_ + hh_bool + hh_cont + hh_ordered]
heads.shape
```

[321]: (2970, 98)

[322]:
```
def correlation(dataset,threshold):
    col_corr = set() # set containing all the columns which needs to drop.
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

[323]:
```
cols = correlation(heads,0.9)
```

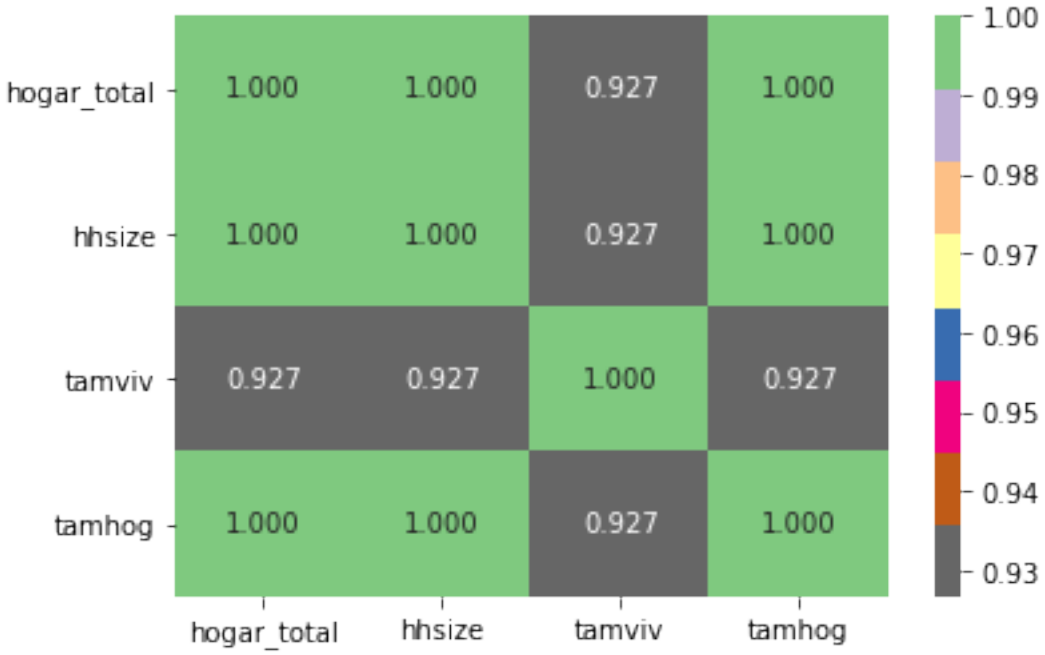
[324]:
```
corr_mat = heads[cols].corr()
corr_mat
```

[324]:

	hogar_total	hhsize	tamviv	tamhog
hogar_total	1.000000	1.000000	0.926591	1.000000
hhsize	1.000000	1.000000	0.926591	1.000000
tamviv	0.926591	0.926591	1.000000	0.926591
tamhog	1.000000	1.000000	0.926591	1.000000

[325]:
```
sns.heatmap(corr_mat, annot=True, cmap = plt.cm.Accent_r, fmt='.3f')
```

[325]: <AxesSubplot:>



[326]:
```
cols = ['hogar_total','tamviv','tamhog']

for df in [train,test]:
    df.drop(columns = cols, inplace = True)
```

[327]:
```
train.shape
```

[327]: (9552, 131)

[328]:
```
for df in [train,test]:
    df.drop(columns = 'male',inplace = True)

train.shape
```

[328]: (9552, 130)

[329]:
```
## area1, =1 zona urbana
## area2, =2 zona rural

for df in [train,test]:
    df.drop(columns = 'area1', inplace = True)

train.shape
```

[329]: (9552, 129)

[330]:
```
cols=['Id','idhogar']
for df in [train, test]:
    df.drop(columns = cols,inplace=True)
```

[331]:
```
train.shape
```

[331]: (9552, 127)

1.0.2 Predict the accuracy using random forest classifier.

[334]:
```
X = train.iloc[:,0:-1]
X.shape
```

[334]: (9552, 126)

[338]:
```
y = train.iloc[:,-1]
y.shape
```

[338]:
```
(9552,)
```

[343]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,f1_score,classification_report

rfc = RandomForestClassifier()
```

[350]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.
 ↪2,random_state = 1)
```

[351]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(7641, 126)
(1911, 126)
(7641,)
(1911,)
```

[352]:
```
rfc.fit(X_train,y_train)
```

[352]:
```
RandomForestClassifier()
```

[353]:
```
y_pred = rfc.predict(X_test)
```

[354]:
```
print(accuracy_score(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
0.9309262166405023
[[ 124   11    0   15]
 [   5  261    5   34]
 [   0    5  176   49]
 [   0    2    6 1218]]
              precision    recall  f1-score   support

           1       0.96      0.83      0.89       150
           2       0.94      0.86      0.89       305
           3       0.94      0.77      0.84       230
           4       0.93      0.99      0.96      1226

    accuracy                           0.93      1911
   macro avg       0.94      0.86      0.90      1911
weighted avg       0.93      0.93      0.93      1911
```

[360]:
```
y_pred_test = rfc.predict(test)
print(y_pred_test)
```

```
[4 4 4 … 4 4 4]
```

1.0.3 Check the accuracy using random forest with cross validation.

[363]:
```
from sklearn.model_selection import KFold,cross_val_score
```

[364]:
```
seed=7
kfold=KFold(n_splits=5,random_state=seed,shuffle=True)
```

[365]:
```
print(cross_val_score(rfc,X,y,cv=kfold,scoring='accuracy'))
results=cross_val_score(rfc,X,y,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.93563579 0.92150706 0.93717277 0.92408377 0.92774869]
92.99622740759614
```

[366]:
```
num_trees= 100

rfc=RandomForestClassifier(n_estimators=100, random_state=10,n_jobs = -1)
```

[367]:
```
print(cross_val_score(rfc,X,y,cv=kfold,scoring='accuracy'))
results=cross_val_score(rfc,X,y,cv=kfold,scoring='accuracy')
print(results.mean()*100)
```

```
[0.93354265 0.92203035 0.93612565 0.93036649 0.92827225]
93.00674792671802
```

[368]:
```
rfc.fit(X_train,y_train)
labels = list(X_train)
feature_imp = pd.DataFrame({'feature': labels, 'importance': rfc.
 ↪feature_importances_})
feature_imp = feature_imp[feature_imp.importance>0.015]
feature_imp.head()
```

[368]:

	feature	importance
0	v2a1	0.018566
2	rooms	0.024148
9	r4h2	0.018782
10	r4h3	0.018442
12	r4m2	0.016706

[369]:
```
y_pred_test = rfc.predict(test)
y_pred_test
```

[369]:
```
array([4, 4, 4, …, 4, 2, 4], dtype=int64)
```

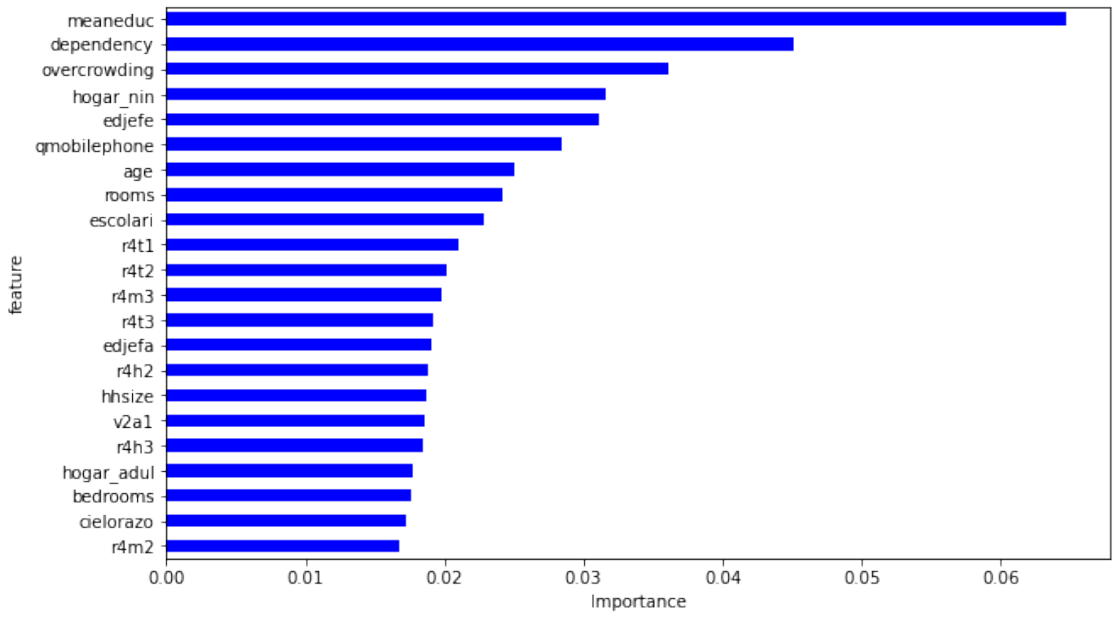
[370]:
```
feature_imp.sort_values(by = ['importance'], ascending = True, inplace = True)
feature_imp['positive'] = feature_imp['importance'] > 0
feature_imp.set_index('feature',inplace = True)
feature_imp.head()
```

[370]:

feature	importance	positive
r4m2	0.016706	True
cielorazo	0.017178	True
bedrooms	0.017633	True
hogar_adul	0.017770	True
r4h3	0.018442	True

[371]:
```
feature_imp.importance.plot(kind = 'barh', figsize=(10,6), color = feature_imp.
↪positive.map({True: 'blue', False: 'red'}))
plt.xlabel('Importance')
```

[371]:
```
Text(0.5, 0, 'Importance')
```



[ ]:
```

```