Loan Prediction

This is a practice hackathon. We have dataset with information about customers and the goal is to predict whether the company should give them loans or not.

At first I do some quick modelling to see what features are important. Then I do data exploration to get some insights and fill missing values. The prediction is done using RandomForest.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('whitegrid')
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from scipy.stats import skew
```

In [2]:
```python
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
```

These are the customers' details available in the dataset.

- Variable - Description
- Loan_ID - Unique Loan ID
- Gender - Male/ Female
- Married - Applicant married (Y/N)
- Dependents - Number of dependents
- Education - Applicant Education (Graduate/ Under Graduate)
- Self_Employed - Self employed (Y/N)
- ApplicantIncome - Applicant income
- CoapplicantIncome - Coapplicant income
- LoanAmount - Loan amount in thousands
- Loan_Amount_Term - Term of loan in months
- Credit_History - credit history meets guidelines
- Property_Area - Urban/ Semi Urban/ Rural
- Loan_Status - Loan approved (Y/N)

Quick modelling

The idea is to get a basic benchmark and to see which features are important while spending no time on data analysis. This will give a rough estimate, but it is useful.

In [3]:
```python
train = train.fillna(train.mean())
test = test.fillna(test.mean())
```

In [4]:
```python
#LoanID is just an index, so it isn't useful. LoanID in test data is necessa
train.drop(['Loan_ID'], axis=1, inplace=True)
test_id = test.Loan_ID
test.drop(['Loan_ID'], axis=1, inplace=True)
```

In [5]:
```python
for col in train.columns.drop('Loan_Status'):
    if train[col].dtype != 'object':
        if skew(train[col]) > 0.75:
            train[col] = np.log1p(train[col])
        pass
    else:
        dummies = pd.get_dummies(train[col], drop_first=False)
        dummies = dummies.add_prefix("{}_".format(col))
        train.drop(col, axis=1, inplace=True)
        train = train.join(dummies)
```

```
for col in test.columns:
    if test[col].dtype != 'object':
        if skew(test[col]) > 0.75:
            test[col] = np.log1p(test[col])
        pass
    else:
        dummies = pd.get_dummies(test[col], drop_first=False)
        dummies = dummies.add_prefix("{}_".format(col))
        test.drop(col, axis=1, inplace=True)
        test = test.join(dummies)
```

In [6]:
```
from sklearn.preprocessing import LabelEncoder
X_train = train.drop('Loan_Status', axis=1)
le = LabelEncoder()
Y_train = le.fit_transform(train.Loan_Status.values)
X_test = test
```

In [7]:
```
#Estimating feature importance.
clf = RandomForestClassifier(n_estimators=200)
clf = clf.fit(X_train, Y_train)
indices = np.argsort(clf.feature_importances_)[::-1]

print('Feature ranking:')
for f in range(X_train.shape[1]):
    print('%d. feature %d %s (%f)' % (f + 1, indices[f], X_train.columns[inc
                                      clf.feature_importances_[indices[f]]))
```

```
Feature ranking:
1. feature 4 Credit_History (0.259262)
2. feature 0 ApplicantIncome (0.180829)
3. feature 2 LoanAmount (0.170103)
4. feature 1 CoapplicantIncome (0.109906)
5. feature 3 Loan_Amount_Term (0.044797)
6. feature 18 Property_Area_Semiurban (0.022159)
7. feature 9 Dependents_0 (0.019059)
8. feature 17 Property_Area_Rural (0.018539)
9. feature 15 Self_Employed_No (0.017493)
10. feature 10 Dependents_1 (0.016999)
11. feature 19 Property_Area_Urban (0.015641)
12. feature 5 Gender_Female (0.015006)
13. feature 6 Gender_Male (0.014921)
14. feature 7 Married_No (0.014761)
15. feature 8 Married_Yes (0.014405)
16. feature 14 Education_Not Graduate (0.014094)
17. feature 13 Education_Graduate (0.014000)
18. feature 16 Self_Employed_Yes (0.013595)
19. feature 11 Dependents_2 (0.012738)
20. feature 12 Dependents_3+ (0.011692)
```

Obviously credit history, income, loan amount and loan amount term are important. Other variables have less importance and may be ignored for now.

In [8]:
```
#I'll use top-5 most important features.
best_features=X_train.columns[indices[0:5]]
```

```
X = X_train[best_features]
Xt = X_test[best_features]
```

In [9]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, Y_train, test_size=0.20,
```

RandomForest is a suitable choice here.

In [10]:
```
clf = RandomForestClassifier(n_estimators=300, n_jobs=-1, criterion = 'gini'
#CalibratedClassifierCV - probability calibration with cross-validation.
calibrated_clf = CalibratedClassifierCV(clf, method='isotonic', cv=5)
calibrated_clf.fit(Xtrain, ytrain)
y_val = calibrated_clf.predict_proba(Xtest)
y_f = [1 if y_val[i][0] < 0.5 else 0 for i in range(len(ytest))]
print("Validation accuracy: ", sum(y_f == ytest) / len(ytest))
```

```
Validation accuracy:  0.780487804878
```

In [11]:
```
clf = RandomForestClassifier(n_estimators=300, n_jobs=-1, criterion = 'gini'
calibrated_clf = CalibratedClassifierCV(clf, method='isotonic', cv=5)
calibrated_clf.fit(X, Y_train)
y_submit = calibrated_clf.predict_proba(Xt)
submission = pd.DataFrame({'Loan_ID':test_id,
                           'Loan_Status':le.inverse_transform([1 if y_submit
submission.to_csv('Loan.csv', index=False)
```

This submission had 0.75 accuracy when submitted, which is a good result. Let's see how it can be improved after paying more attention to data.

In [12]:
```
Input the path to the files instead of "../input".
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
```

Data exploration

In [13]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

In [14]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
Loan_ID              367 non-null object
Gender               356 non-null object
Married              367 non-null object
Dependents           357 non-null object
Education            367 non-null object
Self_Employed        344 non-null object
ApplicantIncome      367 non-null int64
CoapplicantIncome    367 non-null int64
LoanAmount           362 non-null float64
Loan_Amount_Term     361 non-null float64
Credit_History       338 non-null float64
Property_Area        367 non-null object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

In [15]:
```
train.describe(include='all')
```

Out[15]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	A
count	614	601	611	599	614	582	
unique	614	2	2	4	2	2	
top	LP002740	Male	Yes	0	Graduate	No	
freq	1	489	398	345	480	500	
mean	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	

In [16]:

```
train.head()
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applica
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	

A lot of missing values. I think that the score could be improved by careful imputation of missing values for important features.

Loan_ID

In [17]:

```
rain.drop(['Loan_ID'], axis=1, inplace=True)
test_id = test.Loan_ID
test.drop(['Loan_ID'], axis=1, inplace=True)
```

Gender

In [18]:

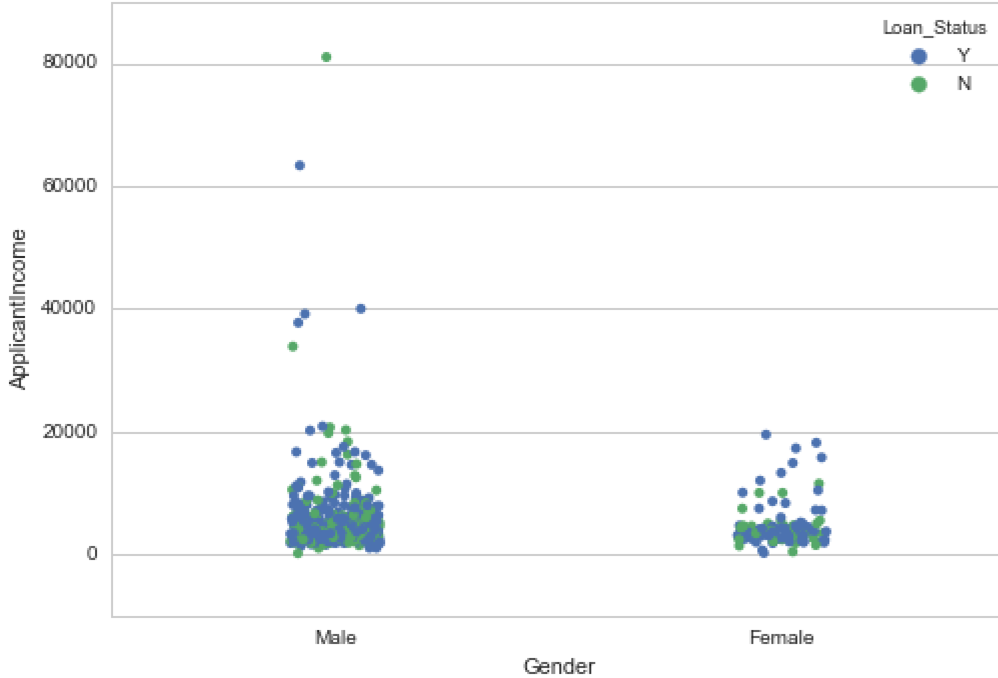
```
train.Gender.value_counts()
```

Out[18]:

```
Male      489
Female    112
Name: Gender, dtype: int64
```

```
In [19]: sns.stripplot(x="Gender", y="ApplicantIncome", data=train, hue='Loan_Status'
```

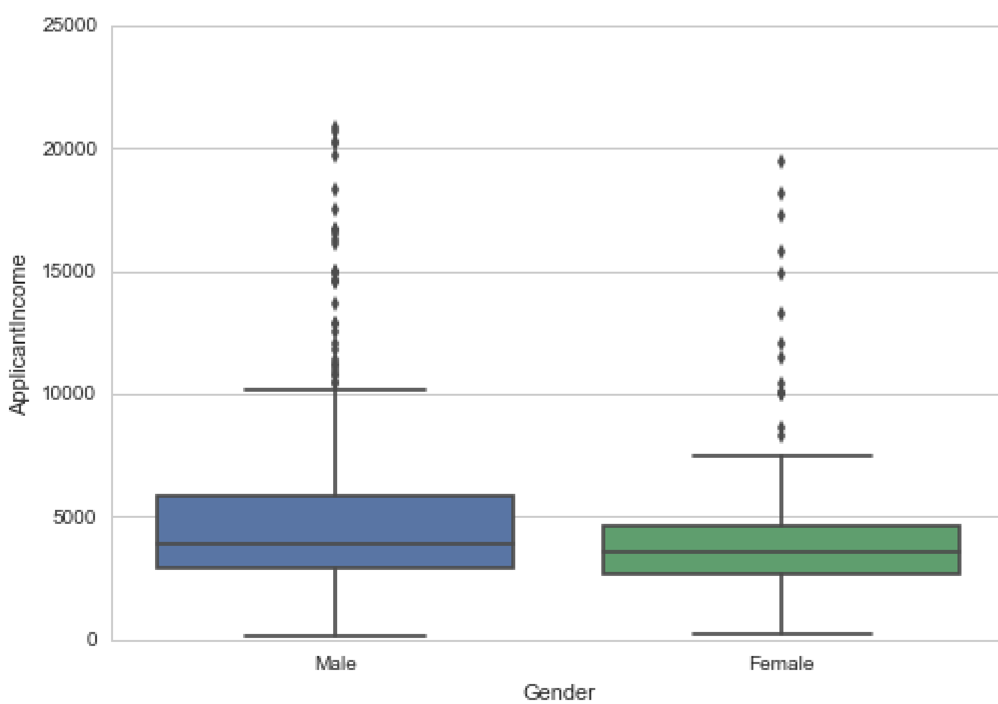
```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1c49ea22e48>
```



Much more men than women in the dataset.

```
In [20]: sns.boxplot(x='Gender', y='ApplicantIncome',  data=train.loc[train.Applicant
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1c49facb940>
```



In this boxplot I showed distribution of income between genders with income < 25000, as only men have higher income. The difference of income isn't high.

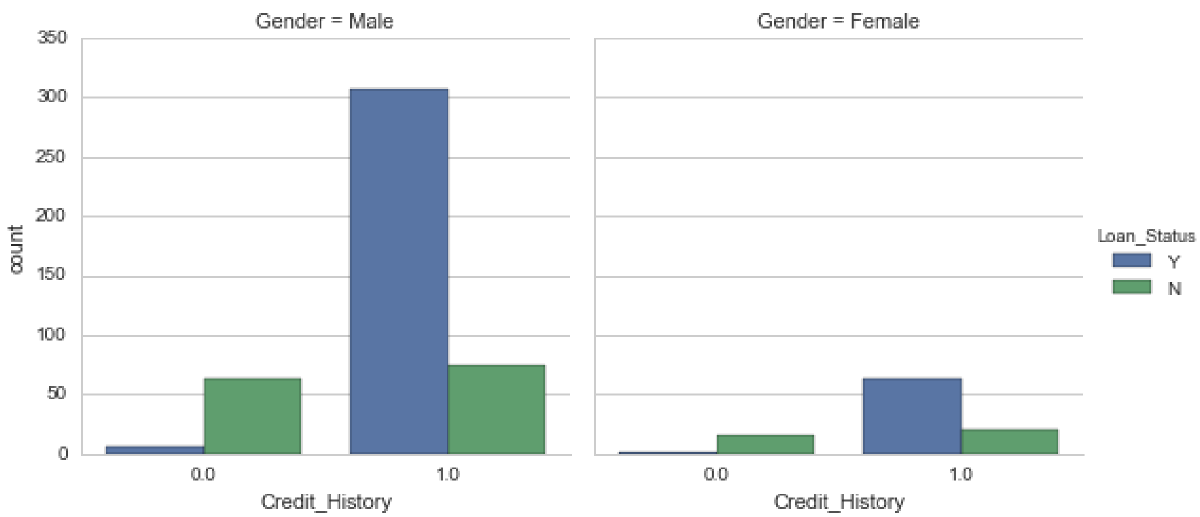
In [21]:
```
train.groupby(['Gender'])['Loan_Status'].value_counts(normalize=True)
```

Out[21]:
```
Gender  Loan_Status
Female  Y              0.669643
        N              0.330357
Male    Y              0.693252
        N              0.306748
Name: Loan_Status, dtype: float64
```

And little impact on Loan Status.

In [22]:
```
sns.factorplot(x="Credit_History", hue="Loan_Status", col="Gender", data=tra
```

Out[22]:
```
<seaborn.axisgrid.FacetGrid at 0x1c49e9bf198>
```



Cosidering all this information I'll fill nan with the most common value.

In [23]:
```
train['Gender'].fillna('Male', inplace=True)
test['Gender'].fillna('Male', inplace=True)
```

Married

In [24]:
```
train.Married.value_counts()
```

Out[24]:
```
Yes    398
No     213
Name: Married, dtype: int64
```

In [25]:
```
pd.crosstab(train.Married, train.Loan_Status)
```

Out[25]:

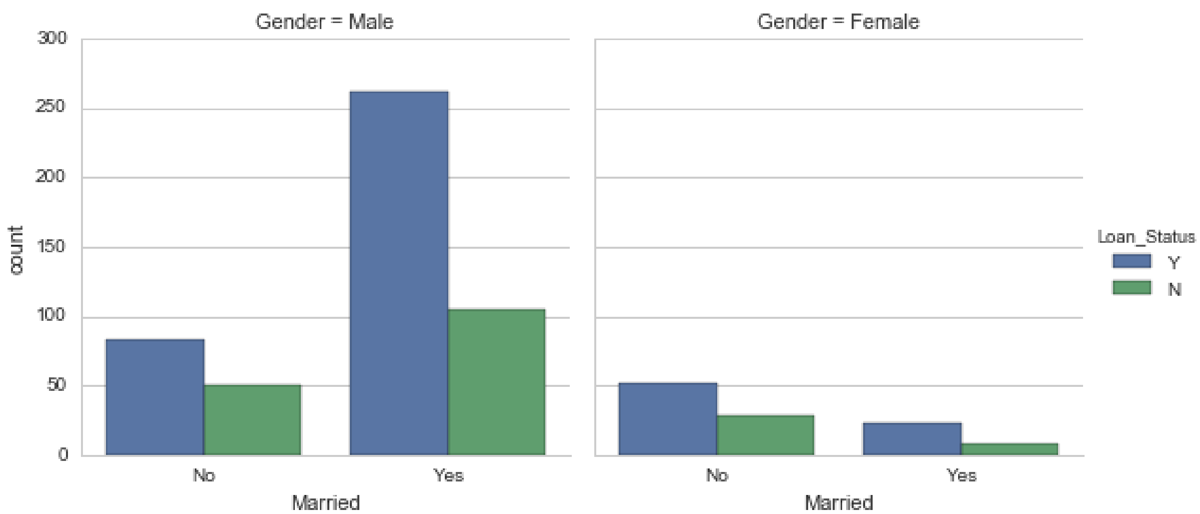
Loan_Status	N	Y
Married		
No	79	134
Yes	113	285

In [26]:
```
train.groupby(['Gender'])['Married'].value_counts(normalize=True)
```

Out[26]:
```
Gender  Married
Female  No         0.720721
        Yes        0.279279
Male    Yes        0.734000
        No         0.266000
Name: Married, dtype: float64
```

In [27]:
```
sns.factorplot(x="Married", hue="Loan_Status", col="Gender", data=train, kir
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x1c49ffa3908>
```



Women are less likely to be married than men.

In [28]:
```
train.loc[train.Married.isnull() == True]
```

Out[28]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncon
104	Male	NaN	NaN	Graduate	No	38:
228	Male	NaN	NaN	Graduate	No	47!
435	Female	NaN	NaN	Graduate	No	100‹

Two men and one woman. Fillna with most common value for gender.

In [29]:
```
train.loc[(train.Gender == 'Male') & (train.Married.isnull() == True), 'Marr
train.loc[(train.Gender == 'Female') & (train.Married.isnull() == True), 'Ma
```

Dependents

In [30]:
```
train.Dependents.value_counts()
```

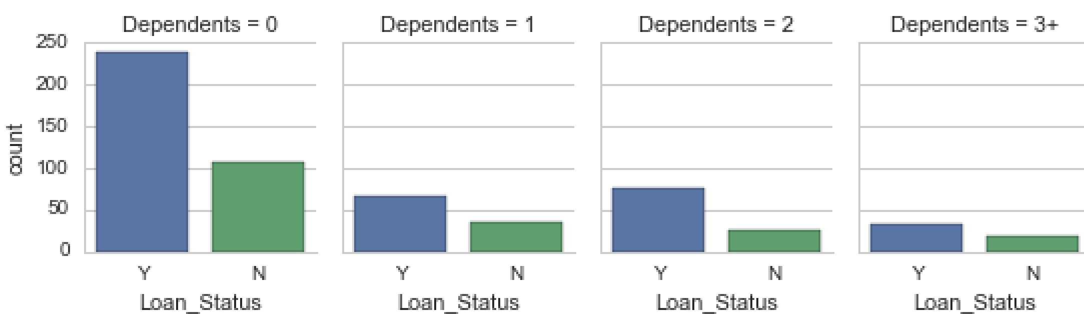
Out[30]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [31]:
```
train.groupby(['Dependents'])['Loan_Status'].value_counts(normalize=True)
```

Out[31]:
```
Dependents  Loan_Status
0           Y              0.689855
            N              0.310145
1           Y              0.647059
            N              0.352941
2           Y              0.752475
            N              0.247525
3+          Y              0.647059
            N              0.352941
Name: Loan_Status, dtype: float64
```

In [32]:
```
sns.factorplot("Loan_Status", col="Dependents", col_wrap=4, data=train, kinc
```

Out[32]:
```
<seaborn.axisgrid.FacetGrid at 0x1c4a1073a90>
```



Most common number of Dependents is zero. And people having 2 Dependents are more likely to get the loan.

In [33]:
```
train.groupby(['Gender', 'Married', 'Property_Area'])['Dependents'].value_cc
```

Out[33]:
```
Gender  Married  Property_Area  Dependents
Female  No       Rural          0             0.842105
                                1             0.105263
                                3+            0.052632
                 Semiurban      0             0.735294
                                1             0.235294
                                2             0.029412
                 Urban          0             0.760000
                                1             0.120000
                                3+            0.080000
                                2             0.040000
        Yes      Rural          0             1.000000
                 Semiurban      0             0.650000
                                1             0.200000
                                2             0.150000
                 Urban          0             0.333333
                                1             0.333333
                                2             0.333333
Male    No       Rural          0             0.840909
                                2             0.090909
                                3+            0.045455
                                1             0.022727
                 Semiurban      0             0.822222
                                1             0.088889
                                2             0.044444
                                3+            0.044444
                 Urban          0             0.880952
                                1             0.119048
        Yes      Rural          0             0.467890
                                2             0.229358
                                1             0.165138
                                3+            0.137615
                 Semiurban      0             0.429688
                                2             0.242188
                                1             0.187500
                                3+            0.140625
                 Urban          0             0.393443
                                2             0.262295
                                1             0.254098
                                3+            0.090164
Name: Dependents, dtype: float64
```

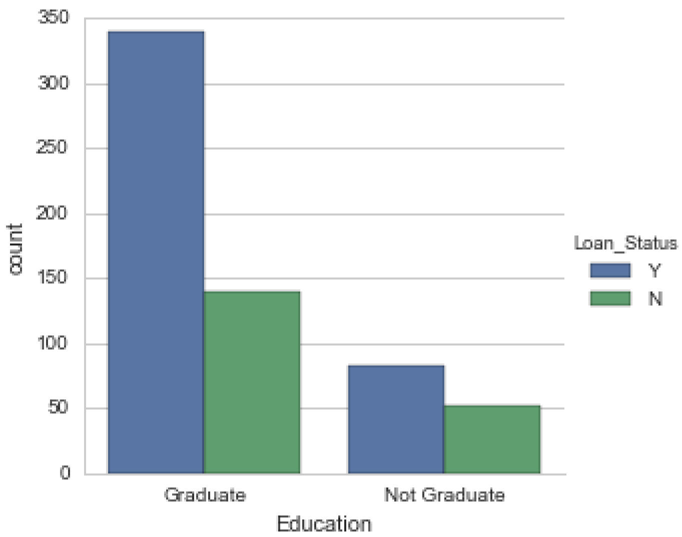
But even with grouping zero dependents is the most common value, so I'll use it to fill nan.

In [34]:
```
train['Dependents'] = train['Dependents'].fillna(train['Dependents'].mode().
test['Dependents'] = test['Dependents'].fillna(test['Dependents'].mode().ilc
```

Education

In [35]:
```
sns.factorplot(x="Education", hue="Loan_Status", data=train, kind="count")
```

Out[35]: <seaborn.axisgrid.FacetGrid at 0x1c4a108d3c8>



It isn't surprising that graduates have more chances to get the loan.

Self_Employed

In [36]:
```
train.groupby(['Self_Employed'])['Loan_Status'].value_counts(normalize=True)
```

Out[36]:
```
Self_Employed  Loan_Status
No             Y              0.686000
               N              0.314000
Yes            Y              0.682927
               N              0.317073
Name: Loan_Status, dtype: float64
```

In [37]:
```
sns.factorplot("Loan_Status", col="Self_Employed", col_wrap=4, data=train, k
```

Out[37]:
```
<seaborn.axisgrid.FacetGrid at 0x1c4a12054a8>
```

It seems that it doesn't really matter whether the customer is Self Employed or not.

In [38]:
```
train.groupby(['Education', 'Married', 'Dependents', 'Gender', 'Property_Are
```

```
Out[38]: Education     Married  Dependents  Gender  Property_Area
         Graduate      No       0           Female  Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                            Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                1           Female  Semiurban        0     No
                                                    Urban            0     No
                                            Male    Semiurban        0     No
                                                    Urban            0     No
                                2           Male    Rural            0     No
                                                    Semiurban        0     No
                                3+          Male    Rural            0     No
                       Yes      0           Female  Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                            Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                1           Female  Semiurban        0     No
                                            Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                2           Female  Semiurban        0     No
                                                    Urban            0     No
                                            Male    Rural            0    Yes
                                                    Semiurban        0     No
                                                    Urban            0     No
                                3+          Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
         Not Graduate  No       0           Female  Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                            Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                       Yes      0           Female  Semiurban        0     No
                                            Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                1           Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                2           Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
                                3+          Male    Rural            0     No
                                                    Semiurban        0     No
                                                    Urban            0     No
         Name: Self_Employed, dtype: object
```

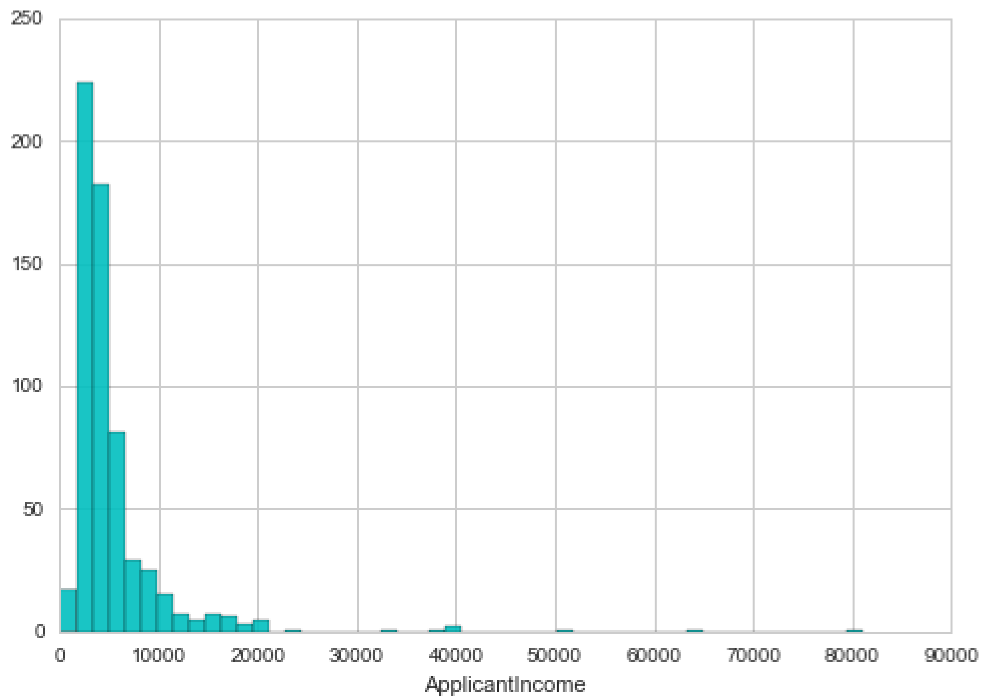
I thought that this grouping makes sense, but there is only one case when most common value is "Yes". In other cases 'Not' is more common.

In [39]:
```
train.loc[(train.Education == 'Graduate') & (train.Married == 'Yes')
          & (train.Dependents == '2') & (train.Gender == 'Male') & (train.Pr
          & (train.Self_Employed.isnull() == True), 'Self_Employed'] = 'Yes'
test.loc[(test.Education == 'Graduate') & (test.Married == 'Yes')
         & (test.Dependents == '2') & (test.Gender == 'Male') & (test.Proper
         & (test.Self_Employed.isnull() == True), 'Self_Employed'] = 'Yes'
train['Self_Employed'].fillna('No', inplace=True)
test['Self_Employed'].fillna('No', inplace=True)
```

ApplicantIncome

In [40]:
```
sns.distplot(train['ApplicantIncome'], kde=False, color='c', hist_kws={'alph
```

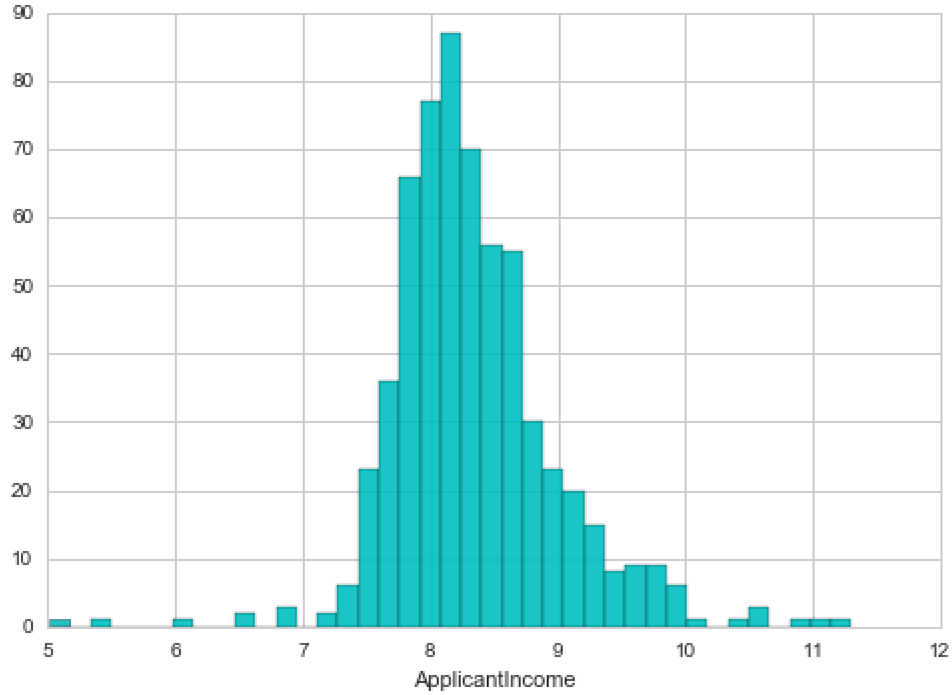
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4a1311438>



The values are highly skewered. Logarithm of data looks better.

In [41]:
```
sns.distplot(np.log1p(train['ApplicantIncome']), kde=False, color='c', hist_
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4a1374c18>



I think that maybe income could be divided in several groups, and there groups could have various rates of getting loan. I begin with 10 groups and if some groups have much higher/lower rate, then groups could be combined.

In [42]:
```
train['Income_group'] = pd.qcut(train.ApplicantIncome, 10, labels=[0,1,2,3,4
test['Income_group'] = pd.qcut(test.ApplicantIncome, 10, labels=[0,1,2,3,4,5
```

In [43]:
```
train['Income_group'] = train['Income_group'].astype(str)
test['Income_group'] = test['Income_group'].astype(str)
```

In [44]:
```
train.groupby(['Income_group'])['Loan_Status'].value_counts(normalize=True)
```

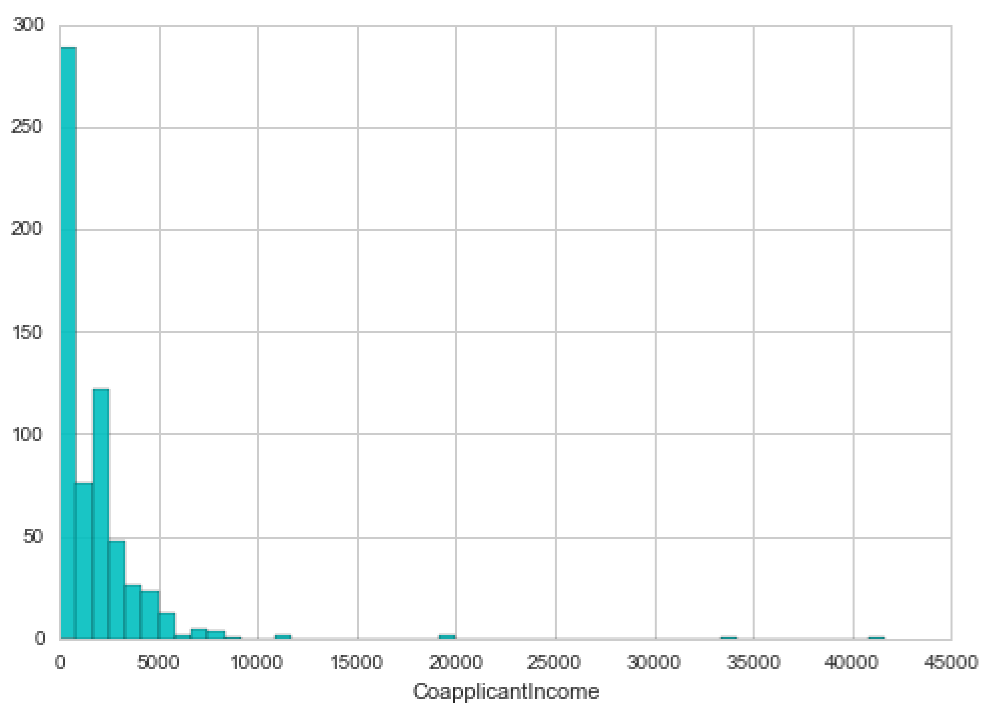
```
Out[44]: Income_group  Loan_Status
         0             Y              0.661290
                       N              0.338710
         1             Y              0.721311
                       N              0.278689
         2             Y              0.704918
                       N              0.295082
         3             Y              0.709677
                       N              0.290323
         4             Y              0.639344
                       N              0.360656
         5             Y              0.737705
                       N              0.262295
         6             Y              0.612903
                       N              0.387097
         7             Y              0.721311
                       N              0.278689
         8             Y              0.688525
                       N              0.311475
         9             Y              0.677419
                       N              0.322581
         Name: Loan_Status, dtype: float64
```

This doesn't seem to be a good feature sadly. We'll see later.

CoapplicantIncome

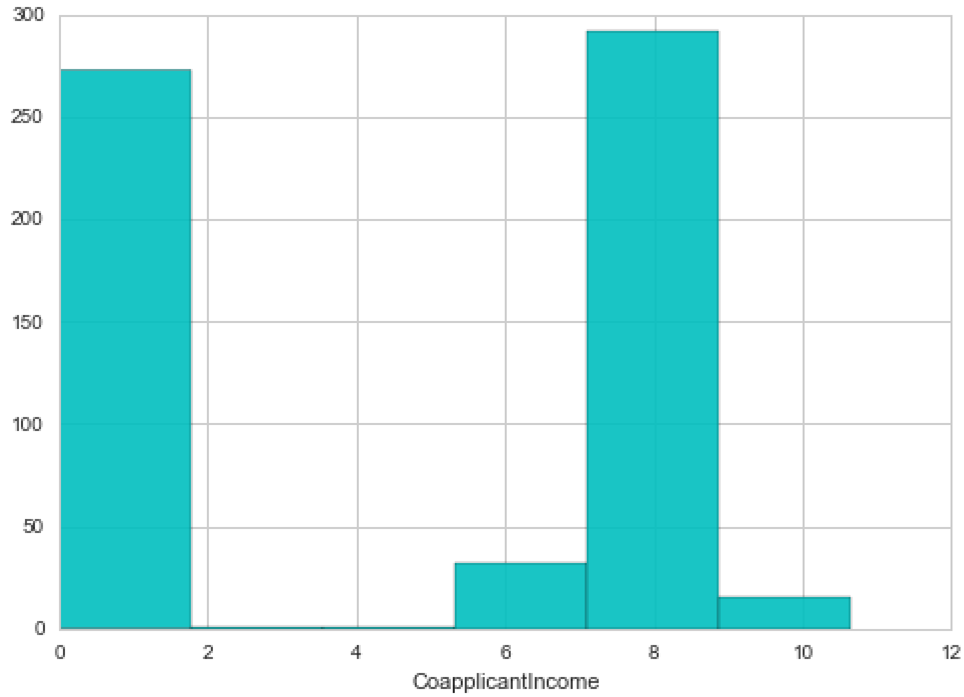
```
In [45]: sns.distplot(train['CoapplicantIncome'], kde=False, color='c', hist_kws={'al
```

```
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4a14f10f0>
```




```
In [46]: sns.distplot(np.log1p(train['CoapplicantIncome']), kde=False, color='c', his
```

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4a15cf1d0>



This variable is also skewered, but logarithm isn't much better. The data has bimodal distribution, so let's divide it into two groups.

In [47]:
```
train['Coap_group'] =  pd.qcut(train.CoapplicantIncome, 2, labels=[0,1])
test['Coap_group'] =  pd.qcut(test.CoapplicantIncome, 2, labels=[0,1])
```

In [48]:
```
train['Coap_group'] = train['Coap_group'].astype(str)
test['Coap_group'] = test['Coap_group'].astype(str)
```

In [49]:
```
train.groupby(['Coap_group'])['Loan_Status'].value_counts(normalize=True)
```

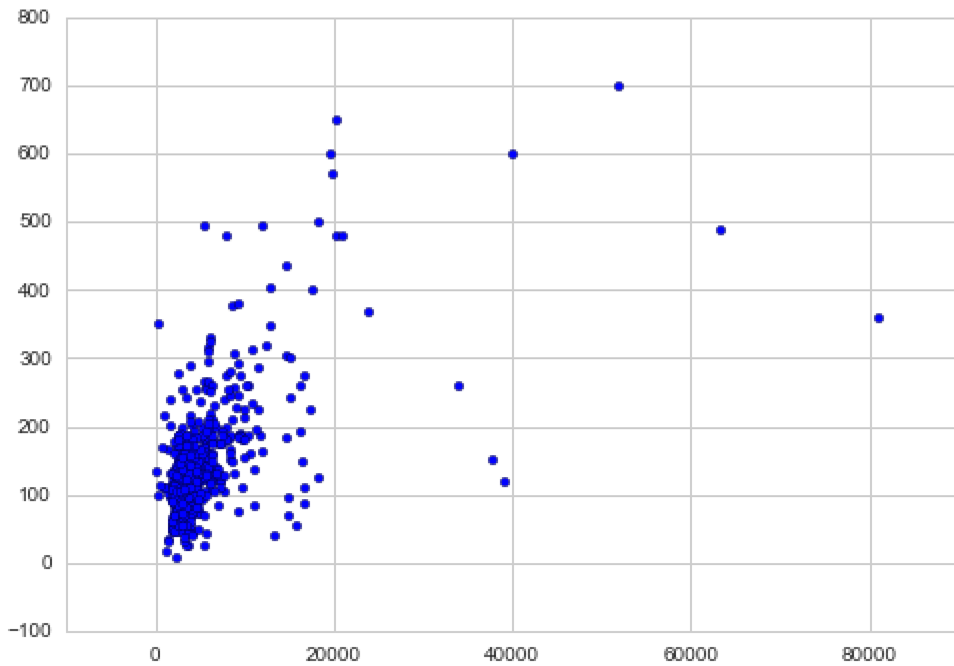
Out[49]:
```
Coap_group  Loan_Status
0           Y              0.677524
            N              0.322476
1           Y              0.697068
            N              0.302932
Name: Loan_Status, dtype: float64
```

Also not good.

LoanAmount

In [50]:
```
plt.scatter(train['ApplicantIncome'], train['LoanAmount'])
```

Out[50]: <matplotlib.collections.PathCollection at 0x1c4a26e0b70>



People with higher income want higher loans. Well, this is reasonable.

```
In [51]: train.groupby(['Education', 'Gender', 'Income_group', 'Self_Employed'])['Loa
```

```
Out[51]: Education     Gender  Income_group  Self_Employed
         Graduate      Female  0             No               113.0
                               1             No               100.0
                                             Yes               96.0
                               2             No                87.0
                               3             No               102.5
                               4             No               112.5
                                             Yes              122.0
                               5             No               115.5
                               6             No               115.0
                                             Yes              133.0
                               7             No               149.5
                                             Yes              105.0
                               8             No               200.0
                                             Yes              172.0
                               9             No               219.5
                                             Yes              286.0
                       Male    0             No                96.0
                                             Yes              160.0
                               1             No               104.0
                                             Yes              164.0
                               2             No               120.5
                                             Yes               95.0
                               3             No               131.0
                                             Yes               88.0
                               4             No               119.5
                                             Yes              130.0
                               5             No               130.0
                               6             No               129.0
                                             Yes              128.0
                               7             No               172.5
                                                              ...
                               8             No               182.5
                                             Yes              220.0
                               9             No               275.0
                                             Yes              182.0
         Not Graduate  Female  0             No                98.0
                               2             No                91.0
                               3             No                95.0
                               5             No               124.0
                                             Yes               62.0
                               6             No               120.0
                               7             No               132.0
                               8             Yes              138.0
                               9             Yes              175.0
                       Male    0             No                95.0
                                             Yes               97.0
                               1             No               118.0
                                             Yes              104.0
                               2             No                98.0
                                             Yes              177.5
                               3             No               113.0
                                             Yes              130.0
                               4             No               109.0
                               5             No               124.0
                                             Yes              158.0
```

```
                    6           No              124.0
                                Yes              96.0
                    7           No              161.0
                                Yes             131.0
                    8           No              130.0
                                Yes             156.0
Name: LoanAmount, dtype: float64
```

In [52]:
```
train.groupby(['Education', 'Gender', 'Self_Employed'])['LoanAmount'].mediar
```

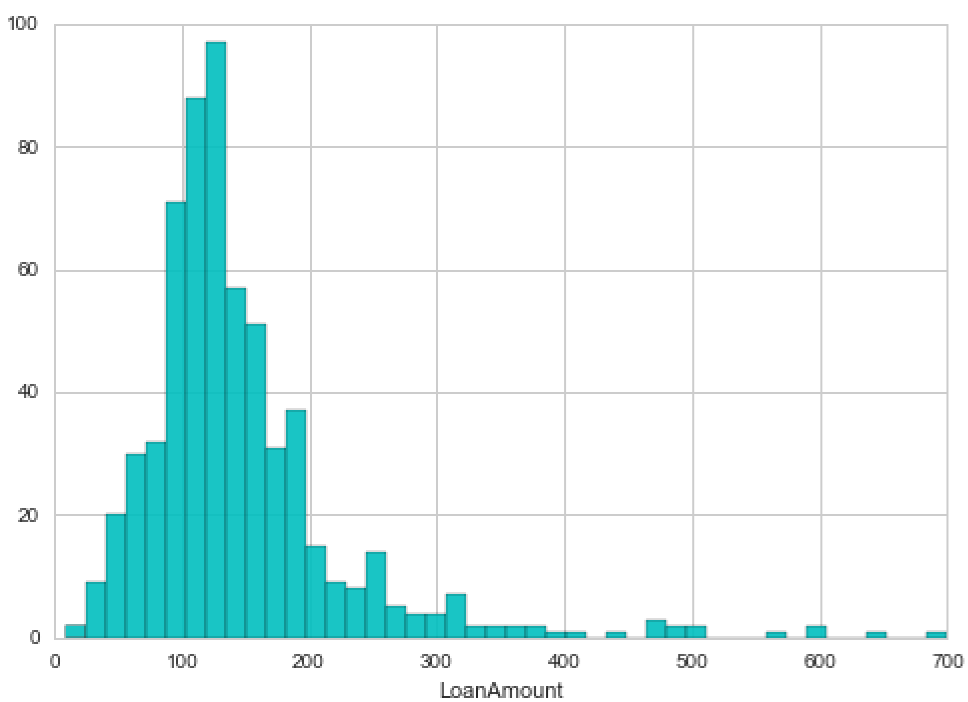
Out[52]:
```
Education     Gender  Self_Employed
Graduate      Female  No                 113.0
                      Yes                127.5
              Male    No                 134.0
                      Yes                160.0
Not Graduate  Female  No                 100.0
                      Yes                131.5
              Male    No                 113.0
                      Yes                130.0
Name: LoanAmount, dtype: float64
```

At first I fillna with mean by Education, Gender, Income Group and Self Employement, but not for all data exists, so second imputation is necesary.

In [53]:
```
train['LoanAmount'] = train.groupby(['Education', 'Gender', 'Income_group',
test['LoanAmount'] = test.groupby(['Education', 'Gender', 'Income_group', 'S
train['LoanAmount'] = train.groupby(['Education', 'Gender', 'Self_Employed']
test['LoanAmount'] = test.groupby(['Education', 'Gender', 'Self_Employed'])[
```

In [54]:
```
sns.distplot(train['LoanAmount'], kde=False, color='c', hist_kws={'alpha': 0
```

Out[54]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c4a26b0240>
```



Loan Amount seems to be more normal than previous variables.

In [55]:
```
train['Loan_group'] =  pd.qcut(train.LoanAmount, 10, labels=[0,1,2,3,4,5,6,7
test['Loan_group'] =  pd.qcut(test.LoanAmount, 10, labels=[0,1,2,3,4,5,6,7,8
train['Loan_group'] = train['Loan_group'].astype(str)
test['Loan_group'] = test['Loan_group'].astype(str)
```

Loan_Amount_Term

In [56]:
```
train.Loan_Amount_Term.value_counts()
```

Out[56]:
```
360.0    512
180.0     44
480.0     15
300.0     13
84.0       4
240.0      4
120.0      3
36.0       2
60.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

It seems than this feature is in fact categorical and not continuous.

In [57]:
```
sns.factorplot("Loan_Status", col="Loan_Amount_Term", col_wrap=3,
               data=train.loc[train.Loan_Amount_Term !=360.], kind="count",
```

Out[57]:
```
<seaborn.axisgrid.FacetGrid at 0x1c4a27a5eb8>
```

And various loan terms have different rates of getting loan.

In [58]:
```
train.groupby(['Education', 'Income_group', 'Loan_group'])['Loan_Amount_Term
```

```
Out[58]: Education     Income_group  Loan_group
         Graduate      0             0             0    360.0
                                     1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                                     5             0    360.0
                                     7             0    360.0
                       1             0             0    360.0
                                     1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                                     5             0    360.0
                                     6             0    360.0
                                     7             0    360.0
                       2             0             0    360.0
                                     1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                                     5             0    360.0
                                     6             0    360.0
                       3             0             0    360.0
                                     1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                                     5             0    360.0
                                     6             0    360.0
                                     7             0    360.0
                       4             1             0    360.0
                                                          ...
         Not Graduate  0             0             0    360.0
                                     1             0    360.0
                                     2             0    360.0
                       1             1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                       2             0             0    180.0
                                     1             0    360.0
                                     2             0    360.0
                                     3             0    360.0
                       3             1             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                       4             2             0    360.0
                                     3             0    360.0
                       5             0             0    360.0
                                     3             0    360.0
                                     4             0    360.0
                                     5             0    360.0
                                     6             0    360.0
                       6             1             0    360.0
                                     4             0    360.0
                                     5             0    360.0
```

```
                              6            0    360.0
              7               5            0    360.0
                              7            0    360.0
                              8            0    360.0
              8               5            0    360.0
                              8            0    360.0
Name: Loan_Amount_Term, dtype: float64
```

But 360 is truly the most common one.

In [59]:
```
train['Loan_Amount_Term'].fillna(360.0, inplace=True)
test['Loan_Amount_Term'].fillna(360.0, inplace=True)
train['Loan_Amount_Term'] = train['Loan_Amount_Term'].astype(str)
test['Loan_Amount_Term'] = test['Loan_Amount_Term'].astype(str)
```

Credit_History

In [60]:
```
train.Credit_History.value_counts()
```

Out[60]:
```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [61]:
```
train.groupby(['Education', 'Self_Employed', 'Property_Area', 'Income_group'
```

Out[61]:

Education	Self_Employed	Property_Area	Income_group		
Graduate	No	Rural	0	0	1.0
			1	0	1.0
			2	0	1.0
			3	0	1.0
			4	0	1.0
			5	0	1.0
			6	0	1.0
			7	0	1.0
			8	0	1.0
			9	0	1.0
		Semiurban	0	0	1.0
			1	0	1.0
			2	0	1.0
			3	0	1.0
			4	0	1.0
			5	0	1.0
			6	0	1.0
			7	0	1.0
			8	0	1.0
			9	0	1.0
		Urban	0	0	1.0
			1	0	1.0
			2	0	1.0
			3	0	1.0
			4	0	1.0
			5	0	1.0
			6	0	1.0
			7	0	1.0
			8	0	1.0
			9	0	1.0
					...
	Yes	Semiurban	8	0	1.0
			9	0	1.0
		Urban	0	0	1.0
			1	0	1.0
			7	0	1.0
			8	0	1.0
			9	0	0.0
Not Graduate	No	Rural	0	0	1.0
			1	0	1.0
			2	0	1.0
			3	0	1.0
			4	0	1.0
			5	0	1.0
			6	0	1.0
			7	0	0.0
		Semiurban	0	0	1.0
			1	0	1.0
			2	0	1.0
			3	0	1.0
			4	0	1.0
			5	0	1.0
			6	0	1.0
			7	0	1.0
		Urban	0	0	1.0

```
                                              1             0    1.0
                                              2             0    0.0
                                              3             0    1.0
                                              5             0    1.0
                 Yes           Rural          8             0    1.0
                               Urban          6             0    1.0
Name: Credit_History, dtype: float64
```

This is one of key variables. Filling missing values is an important decision. So I'll fill them with mode values based on the grouping higher.

In [62]:
```
train.loc[(train.Education == 'Graduate') & (train.Self_Employed == 'Yes')
          & (train.Property_Area == 'Urban') & (train.Income_group == '9') &
          'Self_Employed'] = 0.0
train.loc[(train.Education == 'Not Graduate') & (train.Self_Employed == 'No'
          & (train.Property_Area == 'Rural') & (train.Income_group == '7') &
          'Self_Employed'] = 0.0
train.loc[(train.Education == 'Not Graduate') & (train.Self_Employed == 'No'
          & (train.Property_Area == 'Urban') & (train.Income_group == '2') &
          'Self_Employed'] = 0.0
test.loc[(test.Education == 'Graduate') & (test.Self_Employed == 'Yes')
          & (test.Property_Area == 'Urban') & (test.Income_group == '9') & (
          'Self_Employed'] = 0.0
test.loc[(test.Education == 'Not Graduate') & (test.Self_Employed == 'No')
          & (test.Property_Area == 'Rural') & (test.Income_group == '7') & (
          'Self_Employed'] = 0.0
test.loc[(test.Education == 'Not Graduate') & (test.Self_Employed == 'No')
          & (test.Property_Area == 'Urban') & (test.Income_group == '2') & (
          'Self_Employed'] = 0.0
train['Credit_History'].fillna(1.0, inplace=True)
test['Credit_History'].fillna(1.0, inplace=True)
train['Credit_History'] = train['Credit_History'].astype(str)
test['Credit_History'] = test['Credit_History'].astype(str)
```

Property_Area

In [63]:
```
sns.factorplot('Loan_Status', col='Property_Area', col_wrap=3, data=train, k
```

Out[63]: <seaborn.axisgrid.FacetGrid at 0x1c4a2da96a0>

It seems that people living in Semiurban area have more chances to get loans.

Data Preparation

In [64]:
```
train.dtypes
```

Out[64]:
```
Gender                 object
Married                object
Dependents             object
Education              object
Self_Employed          object
ApplicantIncome         int64
CoapplicantIncome     float64
LoanAmount            float64
Loan_Amount_Term       object
Credit_History         object
Property_Area          object
Loan_Status            object
Income_group           object
Coap_group             object
Loan_group             object
dtype: object
```

In [65]:
```
for col in train.columns.drop('Loan_Status'):
    if train[col].dtype != 'object':
        if skew(train[col]) > 0.75:
            train[col] = np.log1p(train[col])
        pass
    else:
        dummies = pd.get_dummies(train[col], drop_first=False)
        dummies = dummies.add_prefix("{}_".format(col))
        if col == 'Credit_History' or col == 'Loan_Amount_Term':
            pass
        else:
            train.drop(col, axis=1, inplace=True)
        train = train.join(dummies)
for col in test.columns:
    if test[col].dtype != 'object':
        if skew(test[col]) > 0.75:
            test[col] = np.log1p(test[col])
        pass
    else:
        dummies = pd.get_dummies(test[col], drop_first=False)
        dummies = dummies.add_prefix("{}_".format(col))
        if col == 'Credit_History' or col == 'Loan_Amount_Term':
            pass
        else:
            test.drop(col, axis=1, inplace=True)
        test = test.join(dummies)
```

In [66]:
```
#I leave these two variables as they seem to be important by themselves.
train['Credit_History'] = train['Credit_History'].astype(float)
train['Loan_Amount_Term'] = train['Loan_Amount_Term'].astype(float)
test['Credit_History'] = test['Credit_History'].astype(float)
test['Loan_Amount_Term'] = test['Loan_Amount_Term'].astype(float)
```

In [67]:
```
X_train = train.drop('Loan_Status', axis=1)
le = LabelEncoder()
Y_train = le.fit_transform(train.Loan_Status.values)
X_test = test
```

In [68]:
```
clf = RandomForestClassifier(n_estimators=200)
clf = clf.fit(X_train, Y_train)
indices = np.argsort(clf.feature_importances_)[::-1]

print('Feature ranking:')
for f in range(X_train.shape[1]):
    print('%d. feature %d %s (%f)' % (f + 1, indices[f], X_train.columns[inc
                                     clf.feature_importances_[indices[f]]))
```

```
Feature ranking:
1. feature 0 ApplicantIncome (0.105550)
2. feature 28 Credit_History_0.0 (0.104609)
3. feature 2 LoanAmount (0.097904)
4. feature 4 Credit_History (0.094418)
5. feature 29 Credit_History_1.0 (0.079768)
6. feature 1 CoapplicantIncome (0.064432)
7. feature 31 Property_Area_Semiurban (0.019144)
8. feature 3 Loan_Amount_Term (0.017731)
9. feature 30 Property_Area_Rural (0.017615)
10. feature 39 Income_group_6 (0.016972)
11. feature 9 Dependents_0 (0.016816)
12. feature 10 Dependents_1 (0.015914)
13. feature 52 Loan_group_7 (0.014838)
14. feature 7 Married_No (0.014513)
15. feature 8 Married_Yes (0.014420)
16. feature 32 Property_Area_Urban (0.014341)
17. feature 46 Loan_group_1 (0.011999)
18. feature 5 Gender_Female (0.011917)
19. feature 14 Education_Not Graduate (0.011763)
20. feature 38 Income_group_5 (0.011430)
21. feature 43 Coap_group_0 (0.011129)
22. feature 50 Loan_group_5 (0.011068)
23. feature 44 Coap_group_1 (0.011062)
24. feature 24 Loan_Amount_Term_360.0 (0.010687)
25. feature 37 Income_group_4 (0.010678)
26. feature 6 Gender_Male (0.010525)
27. feature 13 Education_Graduate (0.010444)
28. feature 11 Dependents_2 (0.010369)
29. feature 16 Self_Employed_No (0.010063)
30. feature 40 Income_group_7 (0.009910)
31. feature 17 Self_Employed_Yes (0.009498)
32. feature 54 Loan_group_9 (0.009333)
33. feature 34 Income_group_1 (0.008874)
34. feature 41 Income_group_8 (0.008807)
35. feature 12 Dependents_3+ (0.008768)
36. feature 53 Loan_group_8 (0.008255)
37. feature 45 Loan_group_0 (0.007432)
38. feature 33 Income_group_0 (0.007203)
39. feature 42 Income_group_9 (0.007195)
40. feature 49 Loan_group_4 (0.006973)
41. feature 36 Income_group_3 (0.006803)
42. feature 48 Loan_group_3 (0.006733)
43. feature 51 Loan_group_6 (0.006564)
44. feature 25 Loan_Amount_Term_480.0 (0.006474)
45. feature 20 Loan_Amount_Term_180.0 (0.006219)
46. feature 35 Income_group_2 (0.005746)
47. feature 47 Loan_group_2 (0.005574)
48. feature 23 Loan_Amount_Term_36.0 (0.003702)
49. feature 22 Loan_Amount_Term_300.0 (0.003218)
50. feature 21 Loan_Amount_Term_240.0 (0.002481)
51. feature 27 Loan_Amount_Term_84.0 (0.001312)
52. feature 26 Loan_Amount_Term_60.0 (0.000306)
53. feature 15 Self_Employed_0.0 (0.000221)
54. feature 19 Loan_Amount_Term_120.0 (0.000187)
55. feature 18 Loan_Amount_Term_12.0 (0.000091)
```

Well, little changed. The most important variables are the same. Also Credit History is really important.

In [69]:
```
best_features = X_train.columns[indices[0:6]]
X = X_train[best_features]
Xt = X_test[best_features]
```

Model

In [70]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, Y_train, test_size=0.20,
```

In [71]:
```
clf = RandomForestClassifier(n_estimators=300, n_jobs=-1, criterion = 'gini'

calibrated_clf = CalibratedClassifierCV(clf, method='isotonic', cv=5)
calibrated_clf.fit(Xtrain, ytrain)
y_val = calibrated_clf.predict_proba(Xtest)
y_f = [1 if y_val[i][0] < 0.5 else 0 for i in range(len(ytest))]
sum(y_f == ytest) / len(ytest)
```

Out[71]: 0.77235772357723576

I tried using other algorithms, but they had worse results. Also I tried tuning RandomForest parameters, but it led to overfitting.

In [72]:
```
clf = RandomForestClassifier(n_estimators=300, n_jobs=-1, criterion = 'gini'
calibrated_clf = CalibratedClassifierCV(clf, method='isotonic', cv=5)
calibrated_clf.fit(X, Y_train)
y_submit = calibrated_clf.predict_proba(Xt)
```

In [73]:
```
y_pred = le.inverse_transform([1 if y_submit[i][0] < 0.5 else 0 for i in rar
submission = pd.DataFrame({'Loan_ID':test_id, 'Loan_Status':y_pred})
submission.to_csv('Loan.csv', index=False)
```

This solution had an accuracy of 0.784722222222. I couldn't improve it. Then suddenly I made a mistake and made a prediction using estimator fitted not on the whole dataset, but only on the training part(splitted from main train data) and reached a new best accuracy of 0.798611. This is fifth best score. Not sure what caused the increase in the score. I suppose the reason is small amount of data. Adding or substracting some samples could lead to changes is weights, assigned by the estimator. So while the score is higher, there could be overfitting. And on bigger datasets training model on the whole training data is better and more adequate.