Part_I_exploration_template

November 17, 2022

1 Part I - Prosper Loan Data

1.1 by RANDRIANIRINA Ghislain Brice

1.2 IINTRODUCTION

1.3 Preliminary Wrangling

```
In [1]: # import all packages and set plots to be embedded inline
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sb
        %matplotlib inline
```

1.3.1 load data into dataframe

```
In [2]: # load data into dataframe

        loan = pd.read_csv('prosperLoanData.csv')
```

1.3.2 How many features and observations of them do we have?

```
In [3]: # overiview of data shape

        print(f'The loan data contain {loan.shape[0]} observations and {loan.shape[1]} features'

The loan data contain 113937 observations and 81 features
```

1.3.3 How the composition of the data looks like?

```
In [4]: # overiview of composition

        loan.head()

Out[4]:                 ListingKey  ListingNumber            ListingCreationDate  \
        0  1021339766868145413AB3B         193129  2007-08-26 19:09:29.263000000
        1  10273602499503308B223C1        1209647  2014-02-27 08:28:07.900000000
```

```
2  0EE9337825851032864889A          81716  2007-01-05 15:00:47.090000000
3  0EF5356002482715299901A         658116  2012-10-22 11:02:35.010000000
4  0F023589499656230C5E3E2         909464  2013-09-14 18:38:39.097000000

  CreditGrade  Term LoanStatus           ClosedDate  BorrowerAPR  \
0           C    36  Completed  2009-08-14 00:00:00      0.16516
1         NaN    36    Current                  NaN      0.12016
2          HR    36  Completed  2009-12-17 00:00:00      0.28269
3         NaN    36    Current                  NaN      0.12528
4         NaN    36    Current                  NaN      0.24614

   BorrowerRate  LenderYield    ...     LP_ServiceFees  LP_CollectionFees  \
0        0.1580       0.1380    ...            -133.18                0.0
1        0.0920       0.0820    ...               0.00                0.0
2        0.2750       0.2400    ...             -24.20                0.0
3        0.0974       0.0874    ...            -108.01                0.0
4        0.2085       0.1985    ...             -60.27                0.0

   LP_GrossPrincipalLoss  LP_NetPrincipalLoss LP_NonPrincipalRecoverypayments  \
0                    0.0                  0.0                             0.0
1                    0.0                  0.0                             0.0
2                    0.0                  0.0                             0.0
3                    0.0                  0.0                             0.0
4                    0.0                  0.0                             0.0

   PercentFunded  Recommendations InvestmentFromFriendsCount  \
0            1.0                0                          0
1            1.0                0                          0
2            1.0                0                          0
3            1.0                0                          0
4            1.0                0                          0

  InvestmentFromFriendsAmount Investors
0                         0.0       258
1                         0.0         1
2                         0.0        41
3                         0.0       158
4                         0.0        20

[5 rows x 81 columns]
```

1.3.4 How many misssing values per attribute the data have ?

In [5]:
```
# Let's count the missing values

loan.isna().sum()
```

Out[5]:
```
ListingKey                              0
```

```
ListingNumber                                0
ListingCreationDate                          0
CreditGrade                              84984
Term                                         0
LoanStatus                                   0
ClosedDate                               58848
BorrowerAPR                                 25
BorrowerRate                                 0
LenderYield                                  0
EstimatedEffectiveYield                  29084
EstimatedLoss                            29084
EstimatedReturn                          29084
ProsperRating (numeric)                  29084
ProsperRating (Alpha)                    29084
ProsperScore                             29084
ListingCategory (numeric)                    0
BorrowerState                             5515
Occupation                                3588
EmploymentStatus                          2255
EmploymentStatusDuration                  7625
IsBorrowerHomeowner                          0
CurrentlyInGroup                             0
GroupKey                                100596
DateCreditPulled                             0
CreditScoreRangeLower                      591
CreditScoreRangeUpper                      591
FirstRecordedCreditLine                    697
CurrentCreditLines                        7604
OpenCreditLines                           7604
                                         ...
TotalProsperLoans                        91852
TotalProsperPaymentsBilled               91852
OnTimeProsperPayments                    91852
ProsperPaymentsLessThanOneMonthLate      91852
ProsperPaymentsOneMonthPlusLate          91852
ProsperPrincipalBorrowed                 91852
ProsperPrincipalOutstanding              91852
ScorexChangeAtTimeOfListing              95009
LoanCurrentDaysDelinquent                    0
LoanFirstDefaultedCycleNumber            96985
LoanMonthsSinceOrigination                   0
LoanNumber                                   0
LoanOriginalAmount                           0
LoanOriginationDate                          0
LoanOriginationQuarter                       0
MemberKey                                    0
MonthlyLoanPayment                           0
LP_CustomerPayments                          0
```

```
LP_CustomerPrincipalPayments          0
LP_InterestandFees                    0
LP_ServiceFees                        0
LP_CollectionFees                     0
LP_GrossPrincipalLoss                 0
LP_NetPrincipalLoss                   0
LP_NonPrincipalRecoverypayments       0
PercentFunded                         0
Recommendations                       0
InvestmentFromFriendsCount            0
InvestmentFromFriendsAmount           0
Investors                             0
Length: 81, dtype: int64
```

1.3.5 save the count of missing values in a dataframe

In [6]:
```
# save the missing value counts in dataframe

loan_missing_values = loan.isna().sum().reset_index(name = 'count')
loan_missing_values.rename(columns = {'index':'features'}, inplace = True)
loan_missing_values
```

Out[6]:

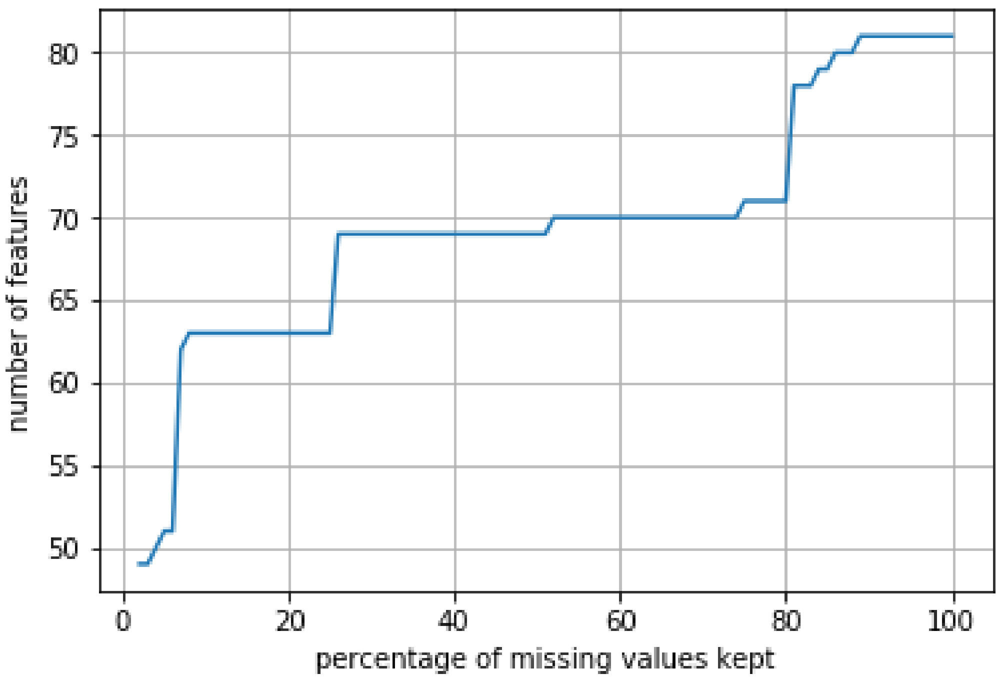
	features	count
0	ListingKey	0
1	ListingNumber	0
2	ListingCreationDate	0
3	CreditGrade	84984
4	Term	0
5	LoanStatus	0
6	ClosedDate	58848
7	BorrowerAPR	25
8	BorrowerRate	0
9	LenderYield	0
10	EstimatedEffectiveYield	29084
11	EstimatedLoss	29084
12	EstimatedReturn	29084
13	ProsperRating (numeric)	29084
14	ProsperRating (Alpha)	29084
15	ProsperScore	29084
16	ListingCategory (numeric)	0
17	BorrowerState	5515
18	Occupation	3588
19	EmploymentStatus	2255
20	EmploymentStatusDuration	7625
21	IsBorrowerHomeowner	0
22	CurrentlyInGroup	0
23	GroupKey	100596
24	DateCreditPulled	0

```
25                CreditScoreRangeLower      591
26                CreditScoreRangeUpper      591
27               FirstRecordedCreditLine     697
28                   CurrentCreditLines     7604
29                      OpenCreditLines     7604
..                                   ...     ...
51                    TotalProsperLoans    91852
52           TotalProsperPaymentsBilled    91852
53                OnTimeProsperPayments    91852
54   ProsperPaymentsLessThanOneMonthLate   91852
55       ProsperPaymentsOneMonthPlusLate   91852
56             ProsperPrincipalBorrowed    91852
57          ProsperPrincipalOutstanding    91852
58          ScorexChangeAtTimeOfListing    95009
59            LoanCurrentDaysDelinquent        0
60          LoanFirstDefaultedCycleNumber  96985
61           LoanMonthsSinceOrigination        0
62                           LoanNumber        0
63                   LoanOriginalAmount        0
64                  LoanOriginationDate        0
65               LoanOriginationQuarter        0
66                            MemberKey        0
67                   MonthlyLoanPayment        0
68                  LP_CustomerPayments        0
69         LP_CustomerPrincipalPayments        0
70                   LP_InterestandFees        0
71                       LP_ServiceFees        0
72                    LP_CollectionFees        0
73               LP_GrossPrincipalLoss         0
74                  LP_NetPrincipalLoss        0
75       LP_NonPrincipalRecoverypayments       0
76                        PercentFunded        0
77                      Recommendations        0
78           InvestmentFromFriendsCount        0
79          InvestmentFromFriendsAmount        0
80                            Investors        0

[81 rows x 2 columns]
```

1.3.6 How many attributes left if we keep data having some percentage of missing values?

In [7]:
```
# Let's plot the number of attributes vs percentage of missing values kept

N = [] # nb of features
n = [] # percentage of observations

for i in range(100, 1,-1):
    df = loan_missing_values[loan_missing_values['count']<=(i)*len(loan)/100]
```

```
        N.append(len(df))
        n.append(i)

    plt.plot(n,N)
    plt.grid()
    plt.xlabel('percentage of missing values kept')
    plt.ylabel('number of features')
    plt.show()

    accepted_missing = 5
    df = loan_missing_values[loan_missing_values['count']<=accepted_missing*len(loan)/100]
    print(f'So there are {len(df)} attributes left if we keep data having missing values les
```




```
So there are 51 attributes left if we keep data having missing values less than or equal to 5 pe
```

1.3.7 List of features where missing values are less than or equal to 5 percent of observations

```
In [8]: # All features where missing values count are less than or equal to 5 percent of observa

        list_5 = list(df['features'].unique())
        list_5

Out[8]: ['ListingKey',
         'ListingNumber',
```

```
'ListingCreationDate',
'Term',
'LoanStatus',
'BorrowerAPR',
'BorrowerRate',
'LenderYield',
'ListingCategory (numeric)',
'BorrowerState',
'Occupation',
'EmploymentStatus',
'IsBorrowerHomeowner',
'CurrentlyInGroup',
'DateCreditPulled',
'CreditScoreRangeLower',
'CreditScoreRangeUpper',
'FirstRecordedCreditLine',
'TotalCreditLinespast7years',
'OpenRevolvingAccounts',
'OpenRevolvingMonthlyPayment',
'InquiriesLast6Months',
'TotalInquiries',
'CurrentDelinquencies',
'DelinquenciesLast7Years',
'PublicRecordsLast10Years',
'IncomeRange',
'IncomeVerifiable',
'StatedMonthlyIncome',
'LoanKey',
'LoanCurrentDaysDelinquent',
'LoanMonthsSinceOrigination',
'LoanNumber',
'LoanOriginalAmount',
'LoanOriginationDate',
'LoanOriginationQuarter',
'MemberKey',
'MonthlyLoanPayment',
'LP_CustomerPayments',
'LP_CustomerPrincipalPayments',
'LP_InterestandFees',
'LP_ServiceFees',
'LP_CollectionFees',
'LP_GrossPrincipalLoss',
'LP_NetPrincipalLoss',
'LP_NonPrincipalRecoverypayments',
'PercentFunded',
'Recommendations',
'InvestmentFromFriendsCount',
'InvestmentFromFriendsAmount',
```

```
        'Investors']
```

1.3.8 Loan data with features where missing values are less than or equal to 5 percent of observations

```
In [9]: # we will keep only loan data where features have missing values less than or equal to 5

        loan_5 = loan[list_5]
        loan_5.shape

Out[9]: (113937, 51)
```

1.3.9 Let's remove all rows having missing values

```
In [10]: # removing rows having missing values

         loan_5 = loan_5.dropna()
         loan_5.shape

Out[10]: (106159, 51)
```

1.3.10 No more missing values ?

```
In [11]: # check if there are still missing values

         loan_5.isnull().sum()

Out[11]: ListingKey                        0
         ListingNumber                     0
         ListingCreationDate               0
         Term                              0
         LoanStatus                        0
         BorrowerAPR                       0
         BorrowerRate                      0
         LenderYield                       0
         ListingCategory (numeric)         0
         BorrowerState                     0
         Occupation                        0
         EmploymentStatus                  0
         IsBorrowerHomeowner               0
         CurrentlyInGroup                  0
         DateCreditPulled                  0
         CreditScoreRangeLower             0
         CreditScoreRangeUpper             0
         FirstRecordedCreditLine           0
         TotalCreditLinespast7years        0
         OpenRevolvingAccounts             0
         OpenRevolvingMonthlyPayment       0
         InquiriesLast6Months              0
```

```
TotalInquiries                        0
CurrentDelinquencies                  0
DelinquenciesLast7Years               0
PublicRecordsLast10Years              0
IncomeRange                           0
IncomeVerifiable                      0
StatedMonthlyIncome                   0
LoanKey                               0
LoanCurrentDaysDelinquent             0
LoanMonthsSinceOrigination            0
LoanNumber                            0
LoanOriginalAmount                    0
LoanOriginationDate                   0
LoanOriginationQuarter                0
MemberKey                             0
MonthlyLoanPayment                    0
LP_CustomerPayments                   0
LP_CustomerPrincipalPayments          0
LP_InterestandFees                    0
LP_ServiceFees                        0
LP_CollectionFees                     0
LP_GrossPrincipalLoss                 0
LP_NetPrincipalLoss                   0
LP_NonPrincipalRecoverypayments       0
PercentFunded                         0
Recommendations                       0
InvestmentFromFriendsCount            0
InvestmentFromFriendsAmount           0
Investors                             0
dtype: int64
```

1.3.11 Attributes choice for analysis

In [12]:
```
# we choose few attributes among the list above for the analysis

last_features = [
 'Term',
 'LoanStatus',
 'BorrowerAPR',
 'BorrowerRate',
 'ListingCategory (numeric)',
 'Occupation',
    'LoanOriginalAmount',
 'EmploymentStatus',
 'IsBorrowerHomeowner',
 'StatedMonthlyIncome',
 'MonthlyLoanPayment',
 'Investors'
```

```
                    ]
loan_clean = loan_5[last_features]
loan_clean.head()
```

```
Out[12]:    Term LoanStatus  BorrowerAPR  BorrowerRate  ListingCategory (numeric)  \
        0     36  Completed      0.16516        0.1580                          0
        1     36    Current      0.12016        0.0920                          2
        2     36  Completed      0.28269        0.2750                          0
        3     36    Current      0.12528        0.0974                         16
        4     36    Current      0.24614        0.2085                          2

             Occupation  LoanOriginalAmount EmploymentStatus  IsBorrowerHomeowner  \
        0         Other                9425    Self-employed                 True
        1  Professional               10000         Employed                False
        2         Other                3001    Not available                False
        3 Skilled Labor               10000         Employed                 True
        4     Executive               15000         Employed                 True

           StatedMonthlyIncome  MonthlyLoanPayment  Investors
        0          3083.333333              330.43        258
        1          6125.000000              318.93          1
        2          2083.333333              123.32         41
        3          2875.000000              321.45        158
        4          9583.333333              563.97         20
```

1.3.12 check if data some datatypes need to be converted

```
In [13]: # overiview of  datatype

         loan_clean.info()
         print('\nSome categorical variables type need to be converted from "object/numeric" typ
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106159 entries, 0 to 113936
Data columns (total 12 columns):
Term                         106159 non-null int64
LoanStatus                   106159 non-null object
BorrowerAPR                  106159 non-null float64
BorrowerRate                 106159 non-null float64
ListingCategory (numeric)    106159 non-null int64
Occupation                   106159 non-null object
LoanOriginalAmount           106159 non-null int64
EmploymentStatus             106159 non-null object
IsBorrowerHomeowner          106159 non-null bool
StatedMonthlyIncome          106159 non-null float64
MonthlyLoanPayment           106159 non-null float64
Investors                    106159 non-null int64
dtypes: bool(1), float64(4), int64(4), object(3)
```

```
memory usage: 9.8+ MB
```

Some categorical variables type need to be converted from "object/numeric" type to "categorical"

1.3.13 Data type conversion

```
In [14]: # change 'LoanStatus', 'Occupation', 'EmploymentStatus' type from 'object' to 'category

         categories = ['LoanStatus', 'Occupation', 'EmploymentStatus', 'ListingCategory (numeric
         loan_clean[categories] = loan_clean[categories].astype('category')
```

```
/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py:3140: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  self[k1] = value[k2]
```

```
In [15]: # check if conversion is done
         loan_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 106159 entries, 0 to 113936
Data columns (total 12 columns):
Term                         106159 non-null int64
LoanStatus                   106159 non-null category
BorrowerAPR                  106159 non-null float64
BorrowerRate                 106159 non-null float64
ListingCategory (numeric)    106159 non-null category
Occupation                   106159 non-null category
LoanOriginalAmount           106159 non-null int64
EmploymentStatus             106159 non-null category
IsBorrowerHomeowner          106159 non-null bool
StatedMonthlyIncome          106159 non-null float64
MonthlyLoanPayment           106159 non-null float64
Investors                    106159 non-null int64
dtypes: bool(1), category(4), float64(4), int64(3)
memory usage: 7.0 MB
```

1.3.14 descriptive statistic

```
In [16]: # descriptive statistic for numeric variables

         numeric_count = -1
         numeric_features = []
         for col in loan_clean.columns:
```

```
            if loan_clean[col].dtypes in ['float64', 'int64']:
                numeric_count +=1
                numeric_features.append(loan_clean[col].describe())
                print(numeric_features[numeric_count], '\n')
        print(f'There are {numeric_count+1} numeric attributes')
```

```
count    106159.000000
mean         41.125595
std          10.689162
min          12.000000
25%          36.000000
50%          36.000000
75%          36.000000
max          60.000000
Name: Term, dtype: float64

count    106159.000000
mean          0.220429
std           0.081007
min           0.006530
25%           0.157130
50%           0.211320
75%           0.287040
max           0.423950
Name: BorrowerAPR, dtype: float64

count    106159.000000
mean          0.193462
std           0.075397
min           0.000000
25%           0.134000
50%           0.184000
75%           0.253700
max           0.360000
Name: BorrowerRate, dtype: float64

count    106159.000000
mean       8530.504008
std        6283.458769
min        1000.000000
25%        4000.000000
50%        7000.000000
75%       12000.000000
max       35000.000000
Name: LoanOriginalAmount, dtype: float64

count    1.061590e+05
mean     5.683045e+03
```

```
std      7.658363e+03
min      0.000000e+00
25%      3.331500e+03
50%      4.750000e+03
75%      6.916667e+03
max      1.750003e+06
Name: StatedMonthlyIncome, dtype: float64

count    106159.000000
mean        278.026517
std         192.885945
min           0.000000
25%         136.980000
50%         227.030000
75%         377.000000
max        2251.510000
Name: MonthlyLoanPayment, dtype: float64

count    106159.000000
mean         81.348477
std         104.510748
min           1.000000
25%           1.000000
50%          44.000000
75%         117.000000
max        1189.000000
Name: Investors, dtype: float64

There are 7 numeric attributes
```

1.3.15 What is the structure of your dataset?

The Prosper loan dataset contain 113937 loans with 81 features where 60 are numerics and 21 non-numerics (categorical, date).

1.3.16 What is/are the main feature(s) of interest in your dataset?

I will focus on the features that affect the most the **Borrower's Annual Percentage Rate** in the dataset.

1.3.17 What features in the dataset do you think will help support your investigation into your feature(s) of interest?

I await the **borrower interest** and **loan amount** have a strongest impact on the loan rate. However, I will inspect other features during the exploration.

1.4 Univariate Exploration

1.4.1 Define univariate/bivariate functions

In [17]:
```
def dynamic_subplot(n):
    v=np.sqrt(n)
    m=round(np.sqrt(n))
    p=round(n/m)
    if m*p < n:
        p += 1
    return (p,m)

def plot_numeric(data, numeric, size_label, size_ticks, percent):
    span = data[numeric].max() - data[numeric].min()
    bins = np.arange(0, data[numeric].max()+span*percent/100, span*percent/100)
    plt.hist(data = data, x = numeric, bins = bins)
    plt.xlabel(numeric, size = size_label)
    plt.ylabel('count', size = size_label)
    plt.xticks(size = size_ticks);
    plt.yticks(size = size_ticks)

def plot_categorical_ord(data, categorical, size_label, size_ticks):
    base_color = sb.color_palette()[0]
    sb.countplot(data=data, y=categorical, color=base_color);
    plt.ylabel(categorical, size = size_label)
    plt.xlabel('count', size = size_label)
    plt.xticks(size = size_ticks)
    plt.yticks(size = size_ticks);

def plot_categorical_nom(data, categorical, size_label, size_ticks):
    base_color = sb.color_palette()[0]
    freq = data[categorical].value_counts()
    cat_order = freq.index
    sb.countplot(data=data, y=categorical, color=base_color, order = cat_order);
    plt.ylabel(categorical, size = size_label)
    plt.xlabel('count', size = size_label)
    plt.xticks(size = size_ticks)
    plt.yticks(size = size_ticks);

def plot_log(data, numeric, size_label, size_ticks, percent):

    mini = np.log10(data[numeric].min())
    maxi = np.log10(data[numeric].max())
    span = maxi - mini

    # Bin size
    bins = 10 ** np.arange(mini, maxi+percent*span/100, percent*span/100)
    plt.hist(data=data, x=numeric, bins=bins);
```

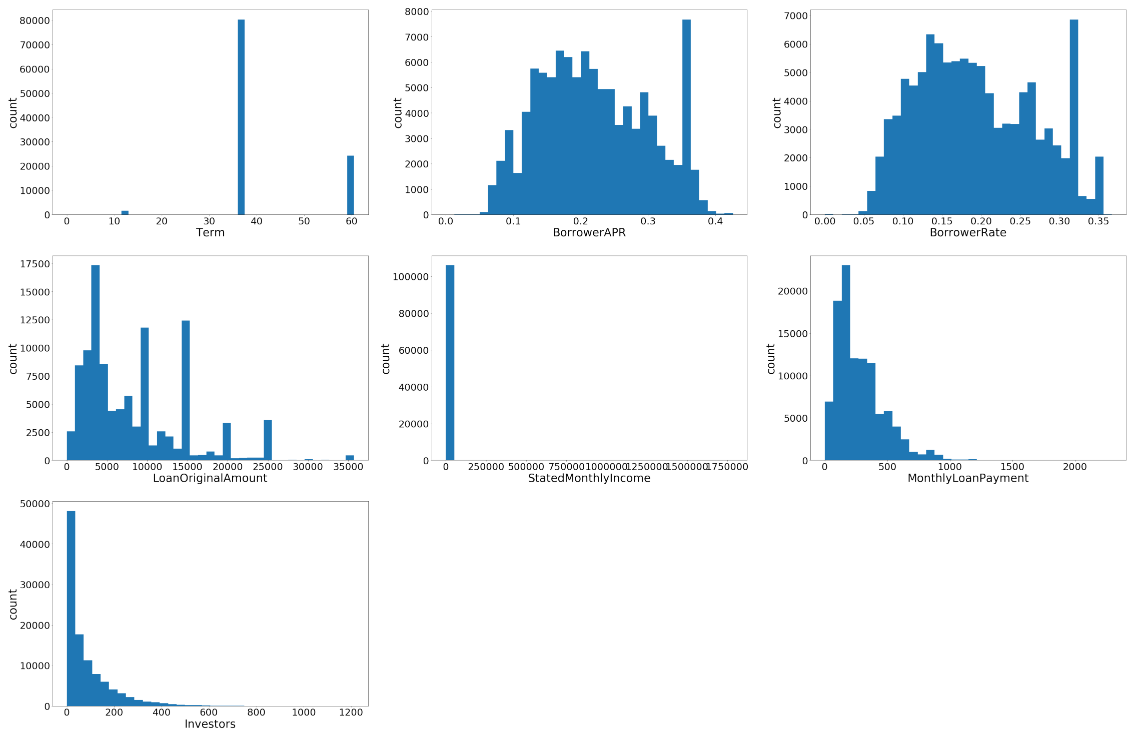
```
    plt.xscale('log')
    # Apply x-axis label
    plt.xlabel(numeric, size = size_label)
    plt.ylabel('count', size = size_label)
    plt.xticks(size = size_ticks)
    plt.yticks(size = size_ticks);

def plot_num_vs_cat(data, categorical, numeric, size_label, size_ticks):
    base_color = sb.color_palette()[0]
    sb.boxplot(data=data, y=categorical, x=numeric, color = base_color)
    plt.xticks(rotation = 45);
    plt.xlabel(numeric, size = size_label)
    plt.ylabel(categorical, size = size_label)
    plt.xticks(size = size_ticks)
    plt.yticks(size = size_ticks);
```

1.4.2 Nnumerical attributes

```
In [18]: # collect all numerical attributes
         numeric_attr = []
         for col in loan_clean.columns:
             if loan_clean[col].dtypes in ['float64', 'int64']:
                 numeric_attr.append(col)
```

1.4.3 Histogram for all numerical attributes

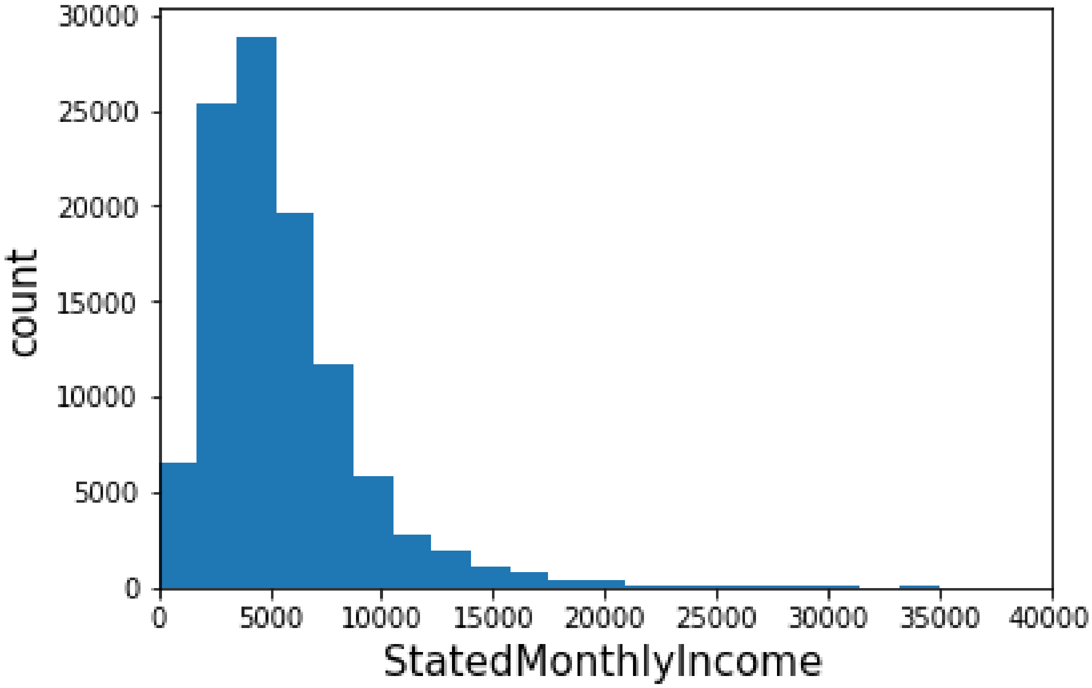
```
In [19]: # plot histogram for all numerical attributes
         plt.figure(figsize = [60,40])

         grids = dynamic_subplot(len(numeric_attr))

         for i in np.arange(len(numeric_attr)):

             plt.subplot(grids[0],grids[1], i+1)
             plot_numeric(loan_clean, numeric_attr[i], 35, 30,3)
```



- Investors, Monthly Loan Payment, Stated Monthly Income and Loan Original Amount are right skweed.
- Loan Original Amount have pattern such as spike then skween righ, then a spike skween righ, and so on.
- Most of the borrowers choose 36 months term.
- There is an unexpected high count between 0.3 and 0.4 for BorrowerAPR and BorrowerRate distributions even if they are looked slightly righ skweed .

In [20]:
```
plot_numeric(loan_clean, 'StatedMonthlyIncome', 15, 10, 0.1)
plt.xlim(0, 40000)
```

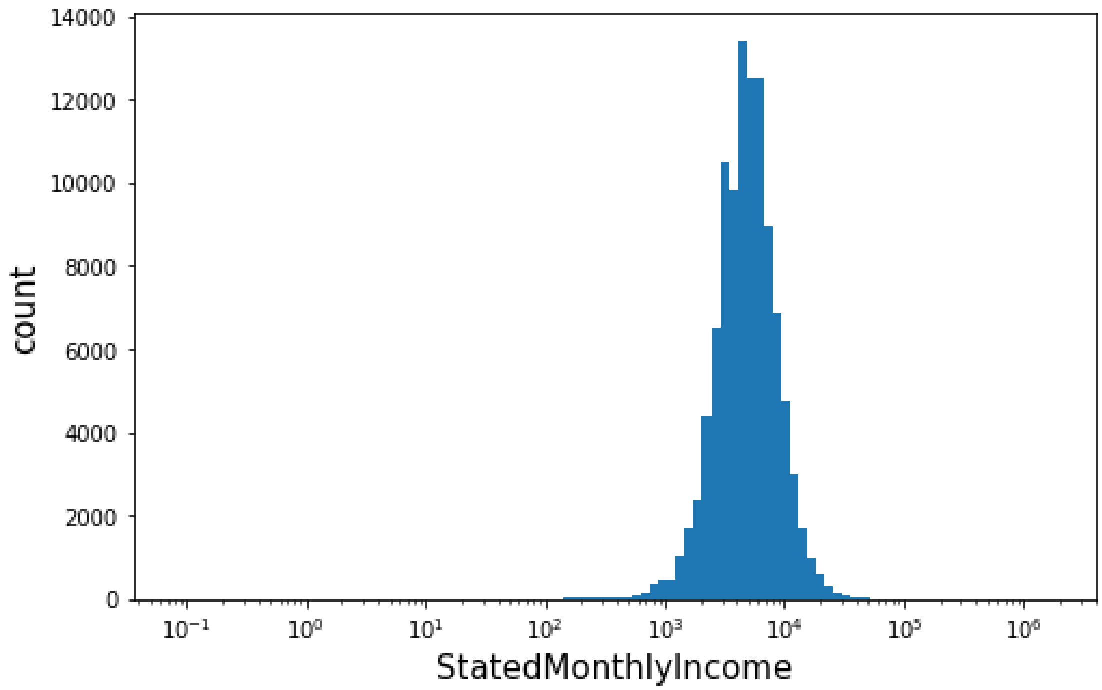
Out[20]:
```
(0, 40000)
```




```
In [ ]:
```

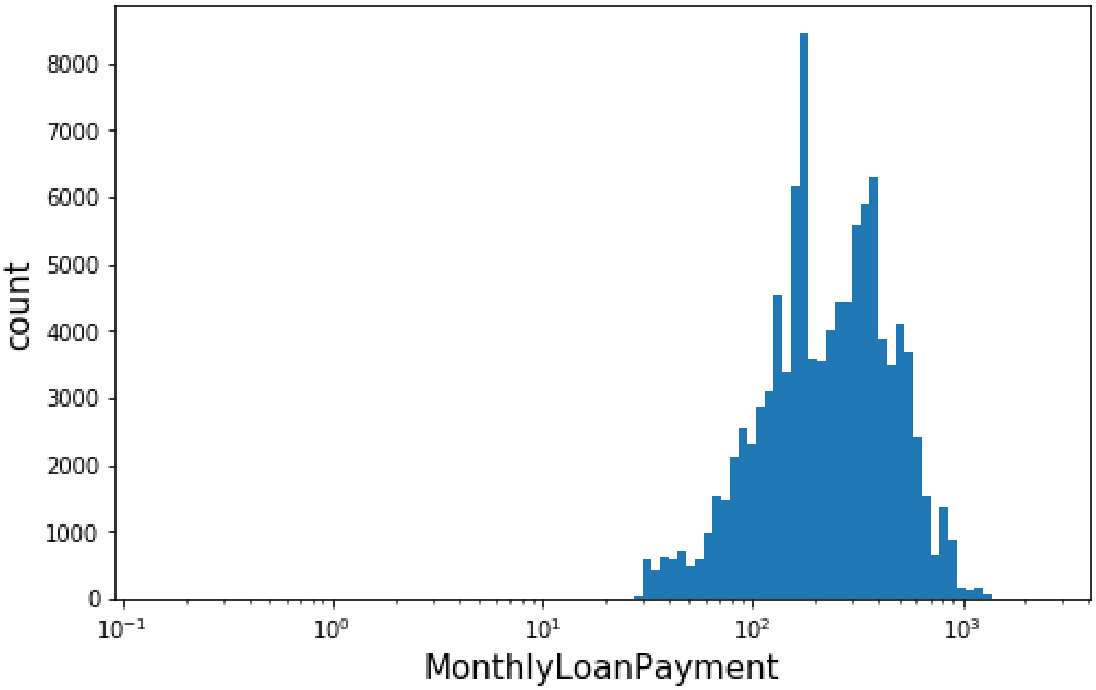
1.4.4 Plot 'stated Monthly Income' more than 0 with log transformation, may be they are student or record mistake

```
In [21]: df = loan_clean[loan_clean['StatedMonthlyIncome'] > loan_clean['StatedMonthlyIncome'].m
         plt.figure(figsize = [8,5])
         plot_log(df, 'StatedMonthlyIncome', 15, 10, 1)
```



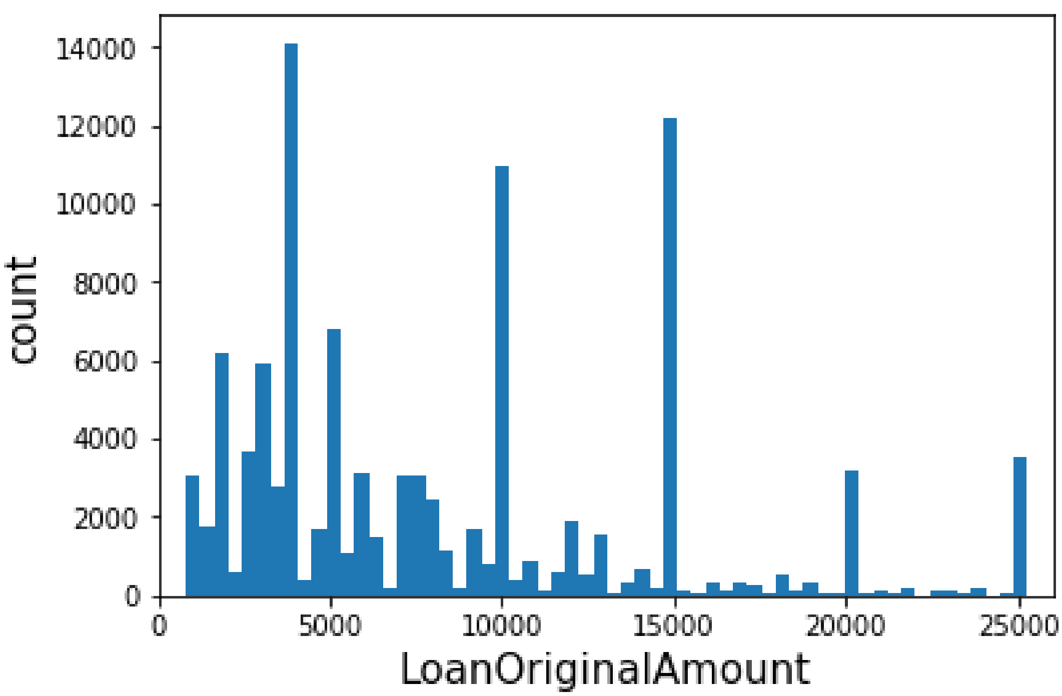
1.4.5 Plot ' Monthly Loan Payment' more than 0 with log transformation, may be they are completed their payment

```
In [22]: df = df[df['MonthlyLoanPayment'] > df['MonthlyLoanPayment'].min()]
         plt.figure(figsize = [8,5])
         plot_log(df, 'MonthlyLoanPayment', 15, 10, 1)
```



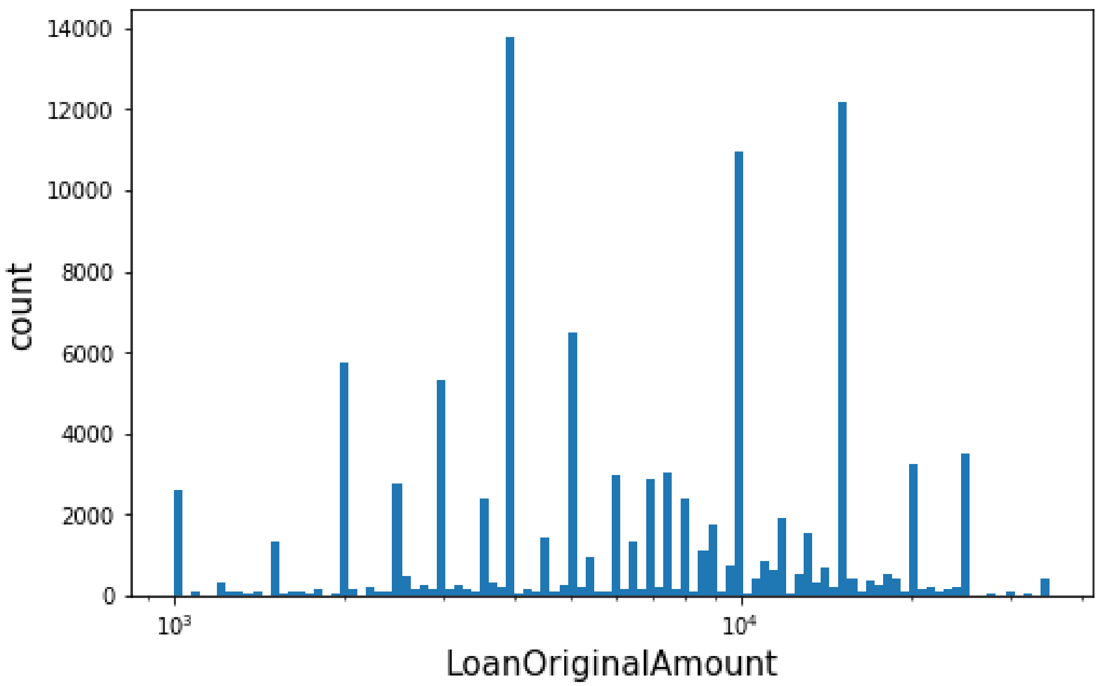

```
In [23]: plot_numeric(loan_clean, 'LoanOriginalAmount', 15, 10, 1.2)
         plt.xlim(0, 26000)
```

```
Out[23]: (0, 26000)
```



1.4.6 Plot 'Loan Original Amount' with log transformation

```
In [24]: plt.figure(figsize = [8,5])
         plot_log(loan_clean, 'LoanOriginalAmount', 15, 10, 1)
```




```
In [ ]:
```

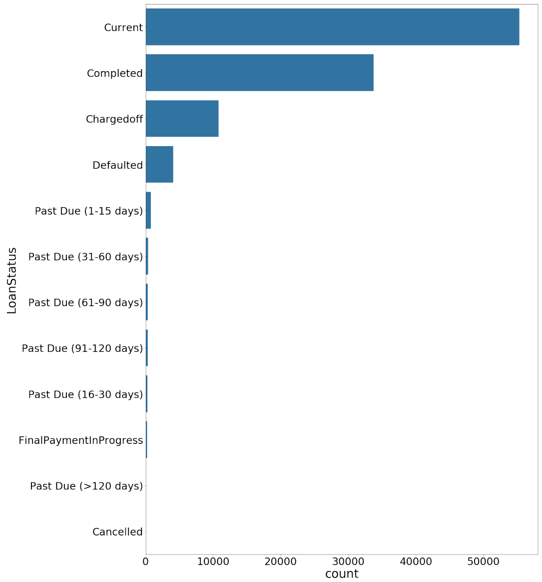
1.4.7 categorical attributes

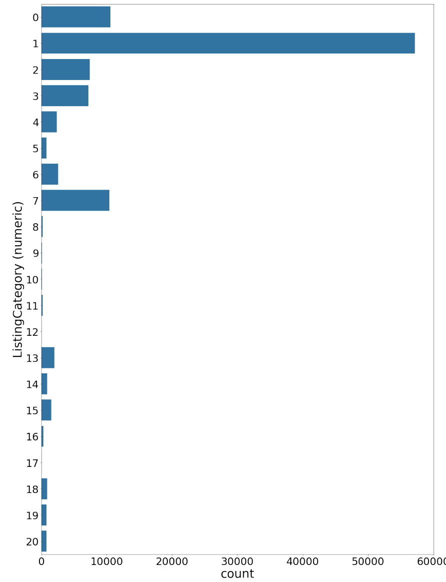
```
In [25]: # collect all categorical attributes
         categoric_attr = []
         for col in loan_clean.columns:
             if loan_clean[col].dtypes not in ['float64', 'int64']:
                 categoric_attr.append(col)
         categoric_attr

Out[25]: ['LoanStatus',
          'ListingCategory (numeric)',
          'Occupation',
          'EmploymentStatus',
          'IsBorrowerHomeowner']
```

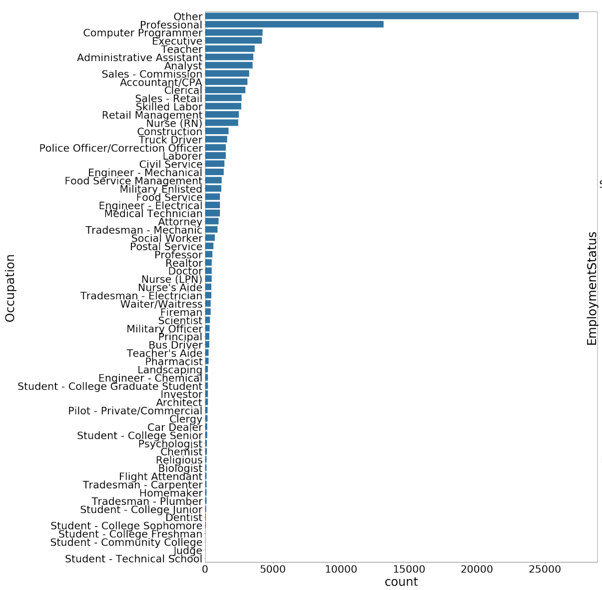
```
In [26]: # separte nominal and ordinal attributes
         nominal = ['LoanStatus',
          'Occupation',
          'EmploymentStatus',
          'IsBorrowerHomeowner']

         ordinal = [
          'ListingCategory (numeric)',
         ]
```

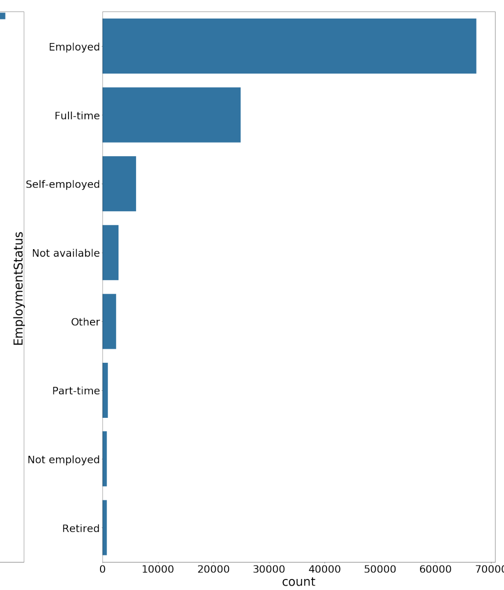
1.4.8 bar graph for all categorical attributes

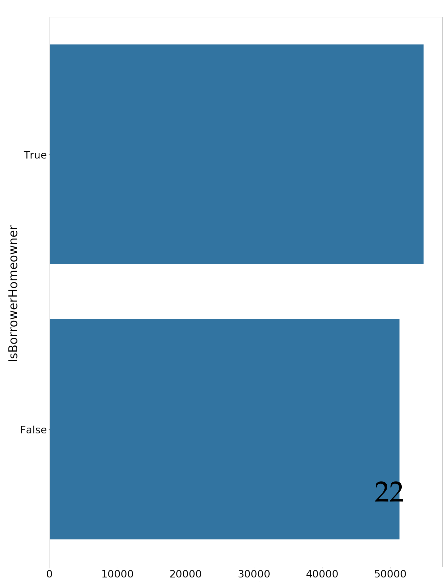
```
In [27]: # plot bar graph for all categorical attributes

         plt.figure(figsize = [45,100])

         grids = dynamic_subplot(len(categoric_attr))

         for i in np.arange(len(categoric_attr)):


             plt.subplot(grids[0],grids[1], i+1)
             if categoric_attr[i] in nominal:
                 plot_categorical_nom(loan_clean, categoric_attr[i],35,30)
             else:
                 plot_categorical_ord(loan_clean, categoric_attr[i],35,30)
```


LoanStatus
Current
Completed
Chargedoff
Defaulted
Past Due (1-15 days)
Past Due (31-60 days)
Past Due (61-90 days)
Past Due (91-120 days)
Past Due (16-30 days)
FinalPaymentInProgress
Past Due (>120 days)
Cancelled
0
10000
20000
30000
40000
50000
count


ListingCategory (numeric)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
0
10000
20000
30000
40000
50000
60000
count


Occupation
Other
Professional
Computer Programmer
Executive
Teacher
Administrative Assistant
Analyst
Sales - Commission
Accountant/CPA
Clerical
Sales - Retail
Skilled Labor
Retail Management
Nurse (RN)
Construction
Truck Driver
Police Officer/Correction Officer
Laborer
Civil Service
Engineer - Mechanical
Food Service Management
Military Enlisted
Food Service
Engineer - Electrical
Medical Technician
Attorney
Tradesman - Mechanic
Social Worker
Postal Service
Professor
Realtor
Doctor
Nurse (LPN)
Nurse's Aide
Tradesman - Electrician
Waiter/Waitress
Fireman
Scientist
Military Officer
Principal
Bus Driver
Teacher's Aide
Pharmacist
Landscaping
Engineer - Chemical
Student - College Graduate Student
Investor
Architect
Pilot - Private/Commercial
Clergy
Car Dealer
Student - College Senior
Psychologist
Chemist
Religious
Biologist
Flight Attendant
Tradesman - Carpenter
Homemaker
Tradesman - Plumber
Student - College Junior
Dentist
Student - College Sophomore
Student - College Freshman
Student - Community College
Judge
Student - Technical School
0
5000
10000
15000
20000
25000
count


EmploymentStatus
Employed
Full-time
Self-employed
Not available
Other
Part-time
Not employed
Retired
0
10000
20000
30000
40000
50000
60000
70000
count


IsBorrowerHomeowner
True
False
0
10000
20000
30000
40000
50000
count

In [ ]:

1.4.9 Discuss the distribution(s) of your variable(s) of interest. Were there any unusual points? Did you need to perform any transformations?

- There is an unexpected high count between 0.3 and 0.4 for BorrowerAPR and BorrowerRate distributions even if they are looked slightly righ skweed.

1.4.10 Of the features you investigated, were there any unusual distributions? Did you perform any operations on the data to tidy, adjust, or change the form of the data? If so, why did you do this?

- Investors, Monthly Loan Payment, Stated Monthly Income and Loan Original Amount are right skweed, so we use logartithm transformation.
- Loan Original Amount have pattern such as spike then skween righ, then a spike skween righ, and so on.
- Most of the borrowers choose 36 months term.

1.5 Bivariate Exploration

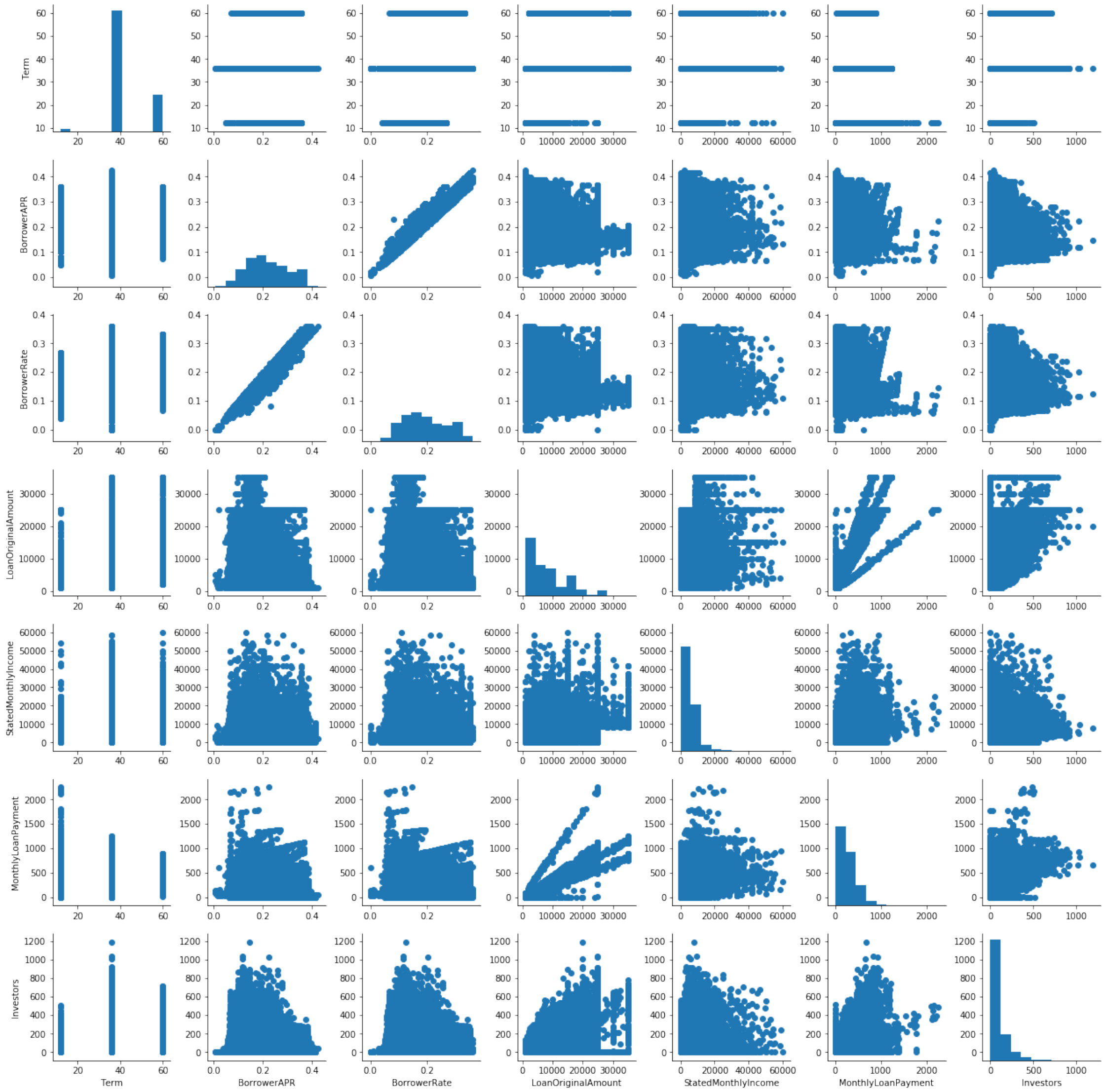
In this section, investigate relationships between pairs of variables in your data. Make sure the variables that you cover here have been introduced in some fashion in the previous section (univariate exploration).

In [28]:
```
numeric_attr
```

Out[28]:
```
['Term',
 'BorrowerAPR',
 'BorrowerRate',
 'LoanOriginalAmount',
 'StatedMonthlyIncome',
 'MonthlyLoanPayment',
 'Investors']
```

In [29]:
```
df = loan_clean[loan_clean['StatedMonthlyIncome'] < 60000]
g = sb.PairGrid(data = df, vars = numeric_attr)
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
```

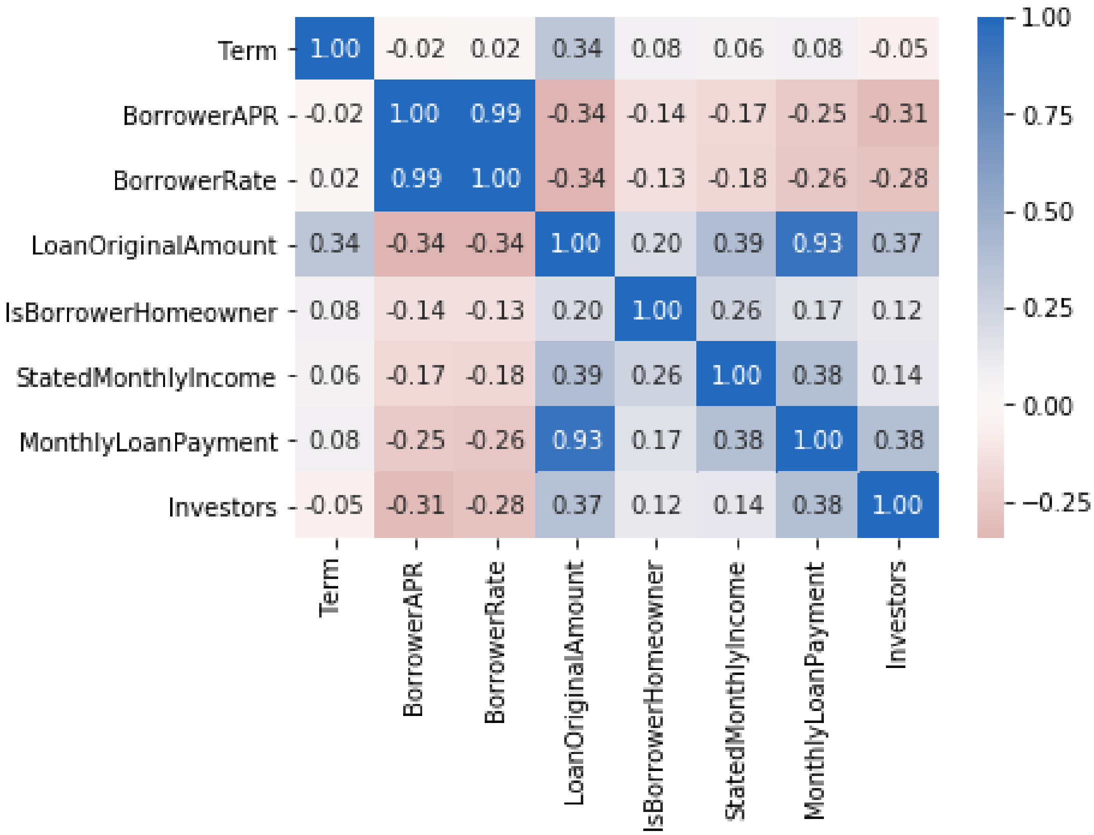
Out[29]:
```
<seaborn.axisgrid.PairGrid at 0x7fa615aa0b70>
```



In [ ]:

```
In [30]: sb.heatmap(df.corr(), annot = True, fmt = '.2f', cmap = 'vlag_r', center = 0)
```

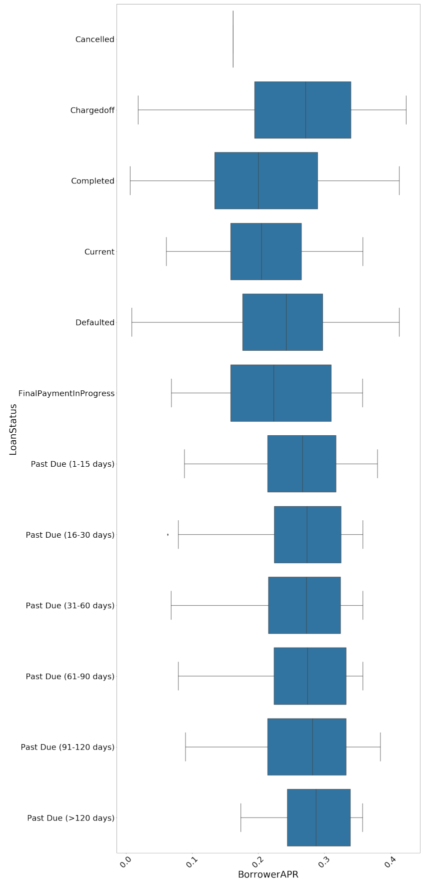
```
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa60a0064a8>
```

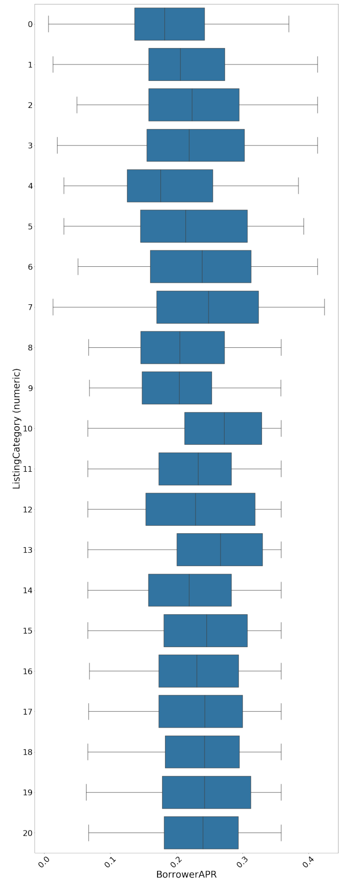


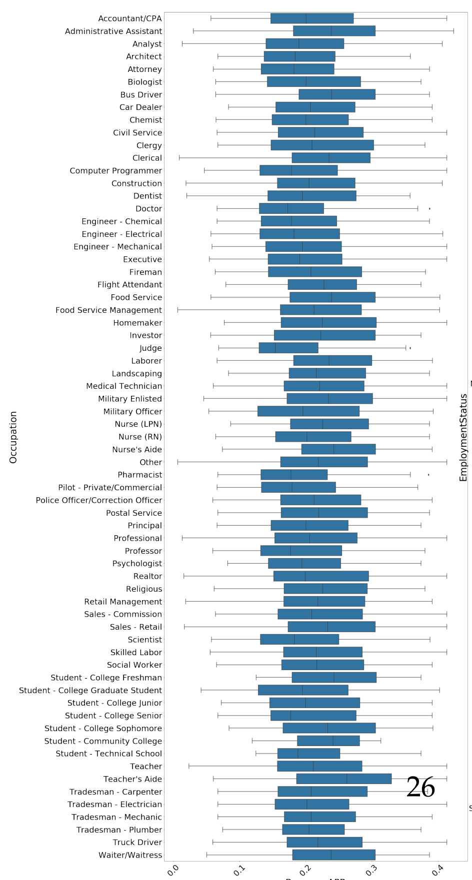
- There is **high correlation** between **BorrowerAPR** and **BorrowerRate**, also between **MonthlyLoanPayment** and **LoanOriginalAmount**

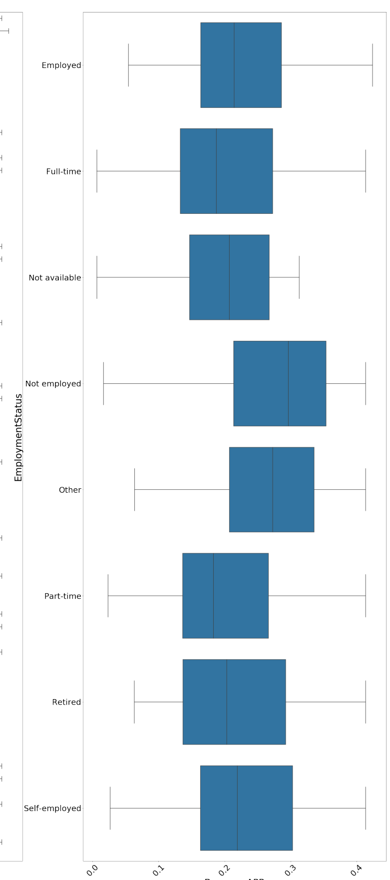
In [ ]:

In [31]:
```
plt.figure(figsize = [45,200])
grids = dynamic_subplot(len(categoric_attr))

for categ in categoric_attr:
    i = categoric_attr.index(categ)
    if i < len(categoric_attr)-1:
        plt.subplot(grids[0],grids[1], i+1)
        plot_num_vs_cat(df, categ, 'BorrowerAPR', 35, 30)
```


LoanStatus
Cancelled
Chargedoff
Completed
Current
Defaulted
FinalPaymentInProgress
Past Due (1-15 days)
Past Due (16-30 days)
Past Due (31-60 days)
Past Due (61-90 days)
Past Due (91-120 days)
Past Due (>120 days)
0.0
0.1
0.2
0.3
0.4
BorrowerAPR


ListingCategory (numeric)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
0.0
0.1
0.2
0.3
0.4
BorrowerAPR

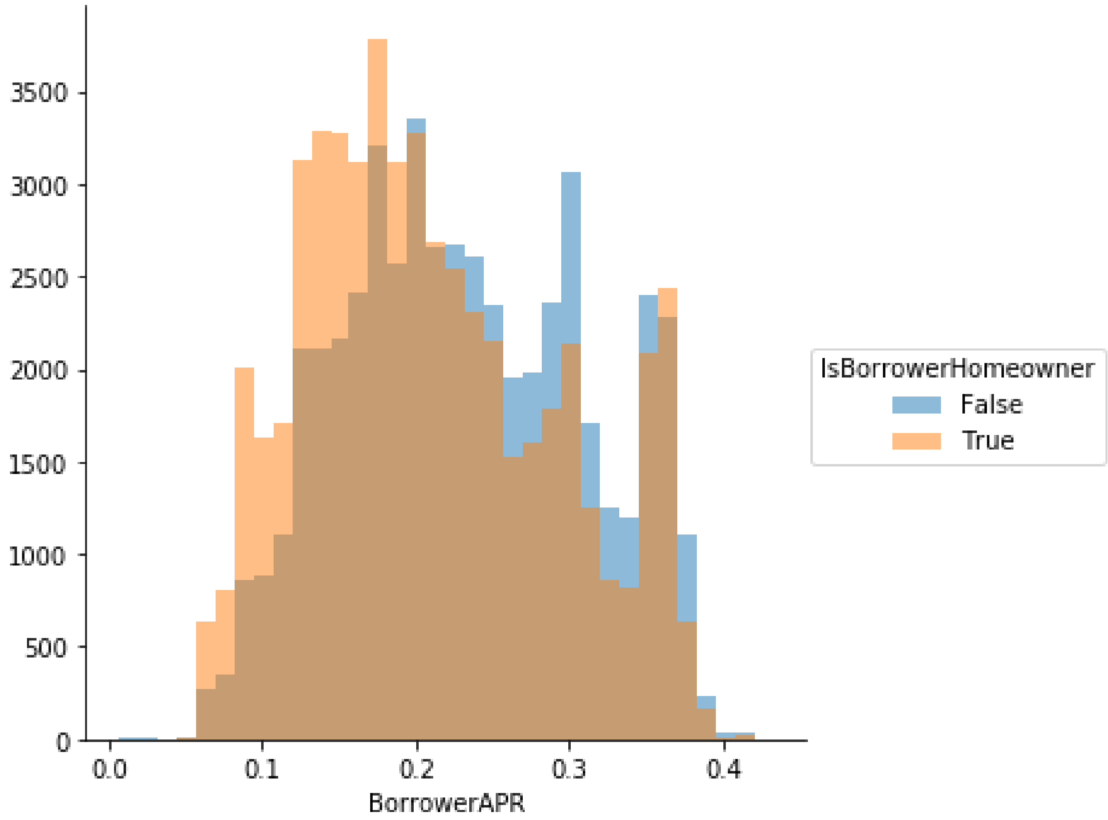

Occupation
Accountant/CPA
Administrative Assistant
Analyst
Architect
Attorney
Biologist
Bus Driver
Car Dealer
Chemist
Civil Service
Clergy
Clerical
Computer Programmer
Construction
Dentist
Doctor
Engineer - Chemical
Engineer - Electrical
Engineer - Mechanical
Executive
Fireman
Flight Attendant
Food Service
Food Service Management
Homemaker
Investor
Judge
Laborer
Landscaping
Medical Technician
Military Enlisted
Military Officer
Nurse (LPN)
Nurse (RN)
Nurse's Aide
Other
Pharmacist
Pilot - Private/Commercial
Police Officer/Correction Officer
Postal Service
Principal
Professional
Professor
Psychologist
Realtor
Religious
Retail Management
Sales - Commission
Sales - Retail
Scientist
Skilled Labor
Social Worker
Student - College Freshman
Student - College Graduate Student
Student - College Junior
Student - College Senior
Student - College Sophomore
Student - Community College
Student - Technical School
Teacher
Teacher's Aide
Tradesman - Carpenter
Tradesman - Electrician
Tradesman - Mechanic
Tradesman - Plumber
Truck Driver
Waiter/Waitress
0.0
0.1
0.2
0.3
0.4
BorrowerAPR


EmploymentStatus
Employed
Full-time
Not available
Not employed
Other
Part-time
Retired
Self-employed
0.0
0.1
0.2
0.3
0.4
BorrowerAPR

- **charged off** for loan status, **10-Cosmetic Procedure** for listing category, **teacher's aide** for occupation and **not employed status** have higher borrower rate.

In [32]:
```
span = df['BorrowerAPR'].max() - df['BorrowerAPR'].min()

bin_edges = np.arange(df['BorrowerAPR'].min(), df['BorrowerAPR'].max()+3*span/100, 3*sp
g = sb.FacetGrid(data = df, hue = categoric_attr[-1], size = 5)
g.map(plt.hist, "BorrowerAPR", bins = bin_edges, alpha = 0.5)
g.add_legend()
```

Out[32]: <seaborn.axisgrid.FacetGrid at 0x7fa606c502b0>



surprise : Client having a house do not have better borrower rate

In [ ]:

In [ ]:

1.5.1 Talk about some of the relationships you observed in this part of the investigation. How did the feature(s) of interest vary with other features in the dataset?

- There is high correlation between BorrowerAPR and BorrowerRate.
- charged off for loan status, 10-Cosmetic Procedure for listing category, teacher's aide for occupation and not employed status have higher borrower rate.

1.5.2 Did you observe any interesting relationships between the other features (not the main feature(s) of interest)?

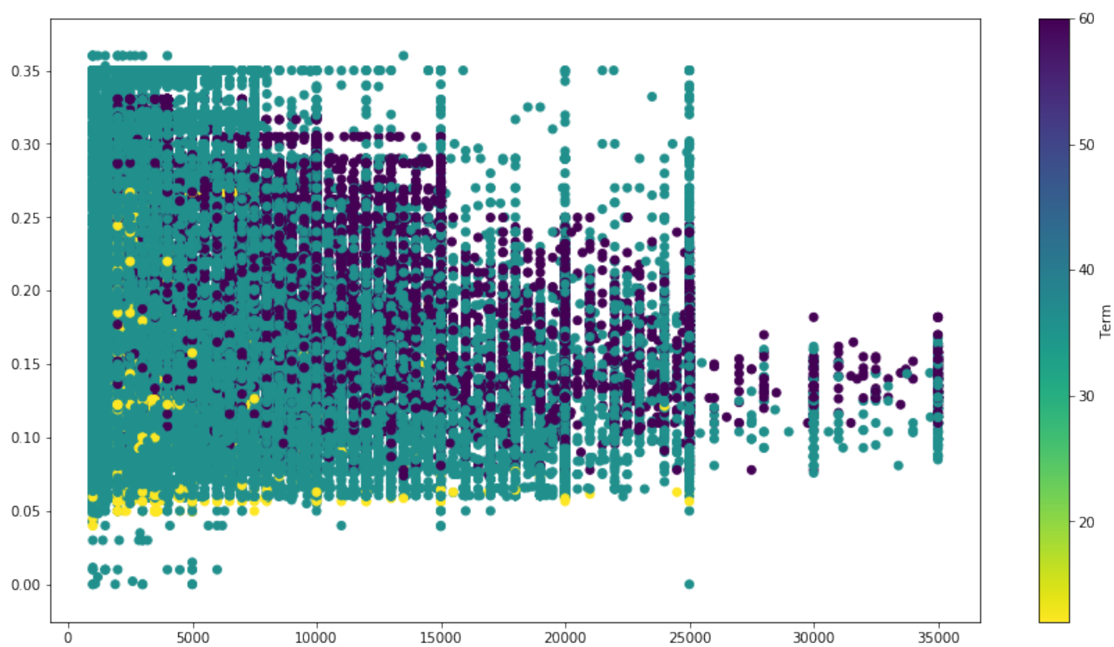
- There is high correlation between MonthlyLoanPayment and LoanOriginalAmount

1.6 Multivariate Exploration

Create plots of three or more variables to investigate your data even further. Make sure that your investigations are justified, and follow from your work in the previous sections.

In [ ]:

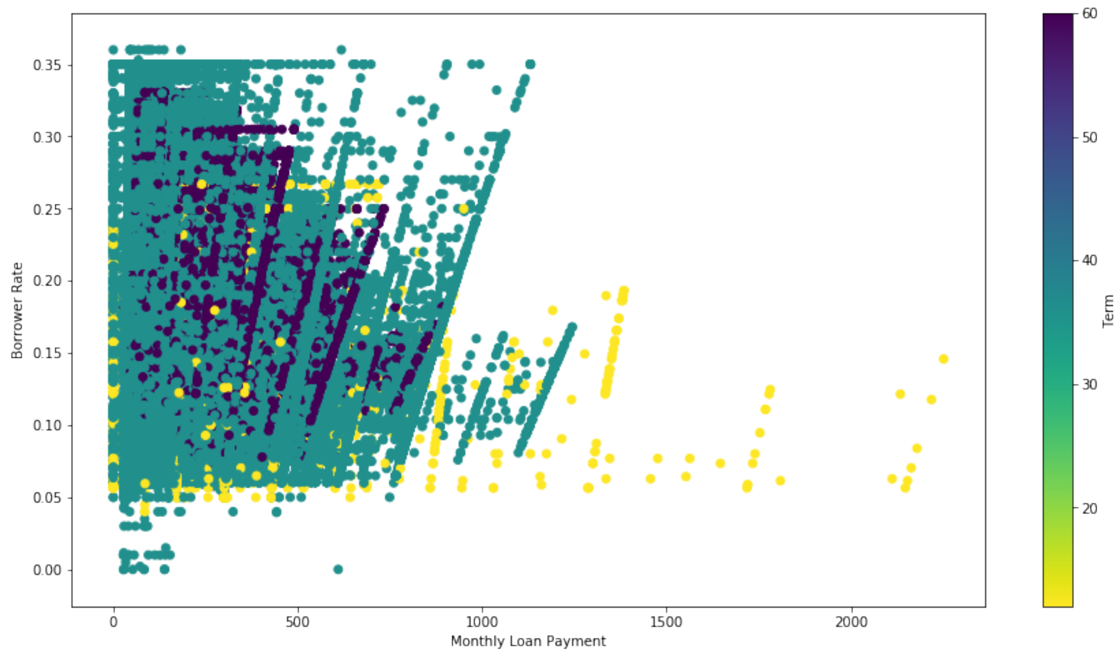
In [34]:
```
plt.figure(figsize =[15,8])
plt.scatter(data = df, x = 'LoanOriginalAmount', y = 'BorrowerRate', c = 'Term',
            cmap = 'viridis_r')
plt.colorbar(label = 'Term');
```



- In general, more the amount of loan increase, lower is the borrower rate.

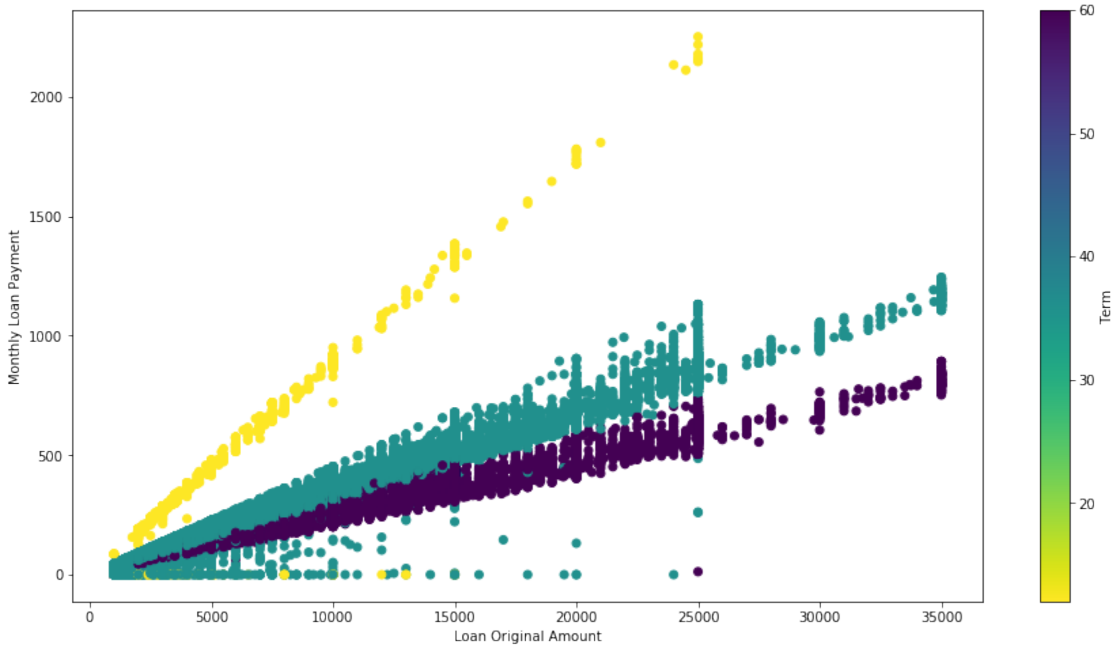
- Borrower using 12 monthts Term borrow lower amount of loan.

In [51]:
```
plt.figure(figsize =[15,8])
plt.scatter(data = df, x = 'MonthlyLoanPayment', y = 'BorrowerRate', c = 'Term',
            cmap = 'viridis_r')
plt.xlabel('Monthly Loan Payment')
plt.ylabel('Borrower Rate')
plt.colorbar(label = 'Term');
```



- more than 95% of borrower pay loan monthly less than 1000.
- for borrower using 12 and 60 months Term, more the monthly loan payment increase, the borrower rate decrase.
- the few borrowers having monthly loan payment more than around 1200 use only 12 month Term and they have low borrower rate.

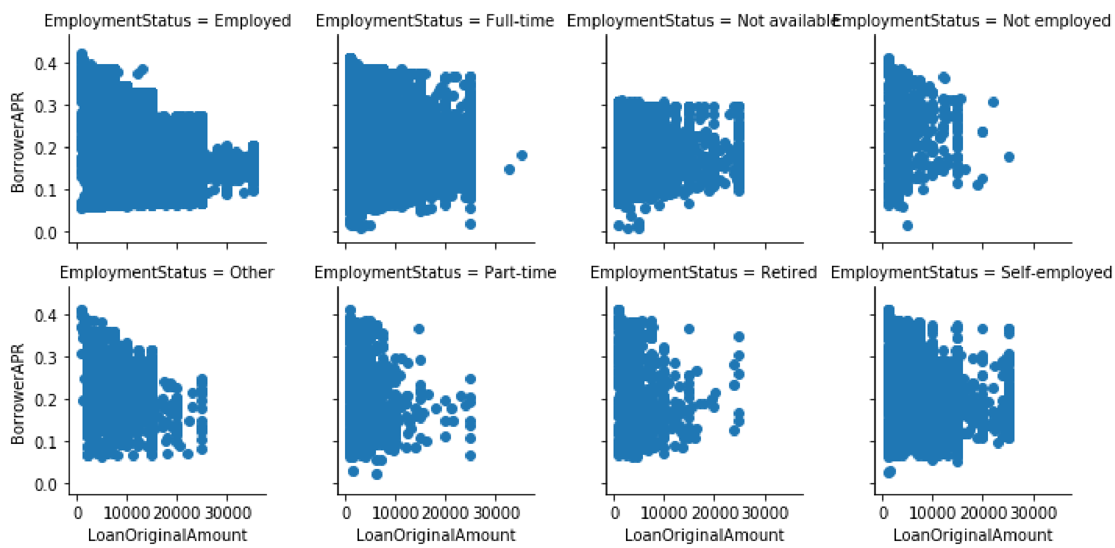
In [50]:
```
plt.figure(figsize =[15,8])
plt.scatter(data = df, x = 'LoanOriginalAmount', y = 'MonthlyLoanPayment', c = 'Term',
            cmap = 'viridis_r')
plt.xlabel('Loan Original Amount')
plt.ylabel('Monthly Loan Payment')
plt.colorbar(label = 'Term');
```



- 25000 is the highest Loan Original Amount for 12 months Term.
- 35000 is the highest Loan Original Amount for 36 and 60 months Terms.
- Lower is the Term of payment, higher is the Monthly Loan Payment.

In [36]:
```
g = sb.FacetGrid(data = df, col = 'EmploymentStatus', size = 2.5, margin_titles = True,
g.map(plt.scatter, 'LoanOriginalAmount', 'BorrowerAPR')
```

Out[36]: <seaborn.axisgrid.FacetGrid at 0x7fa6015bb2e8>

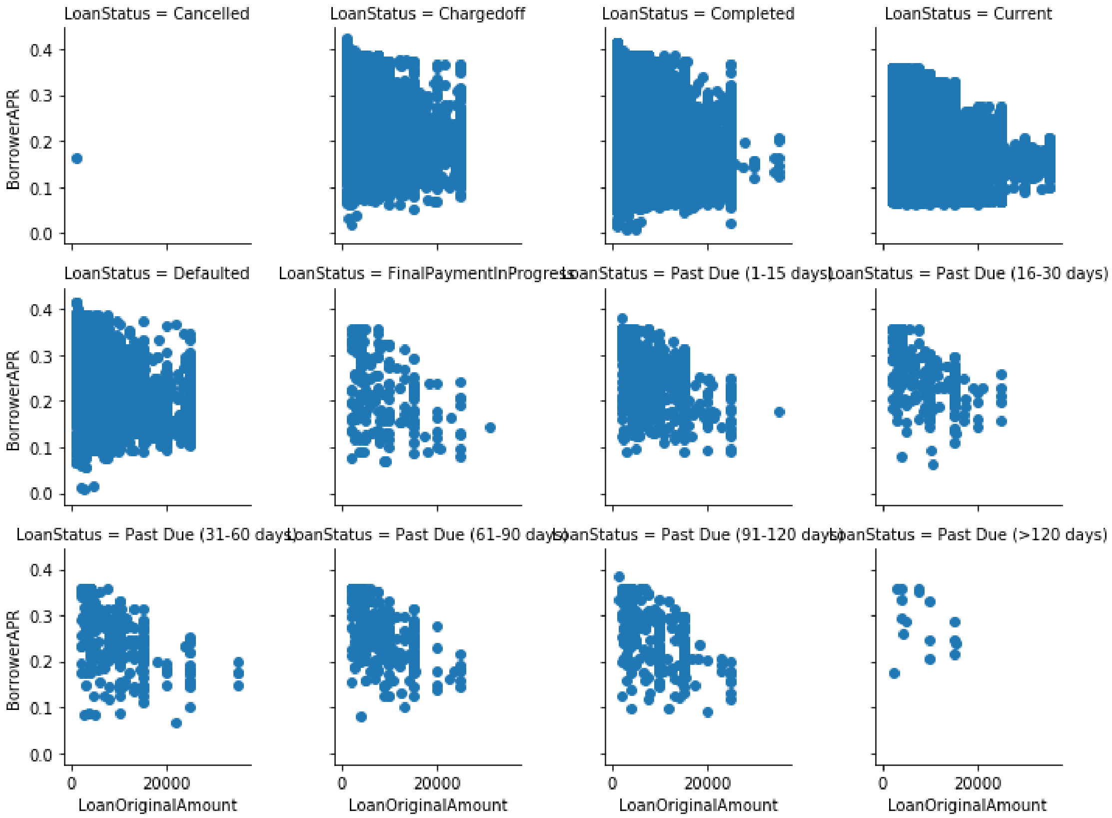


- Not available empoyement status have the lower borrower rate.

In [ ]:

In [37]:
```
g = sb.FacetGrid(data = df, col = 'LoanStatus', size = 2.5, margin_titles = True, col_w
g.map(plt.scatter, 'LoanOriginalAmount', 'BorrowerAPR')
```

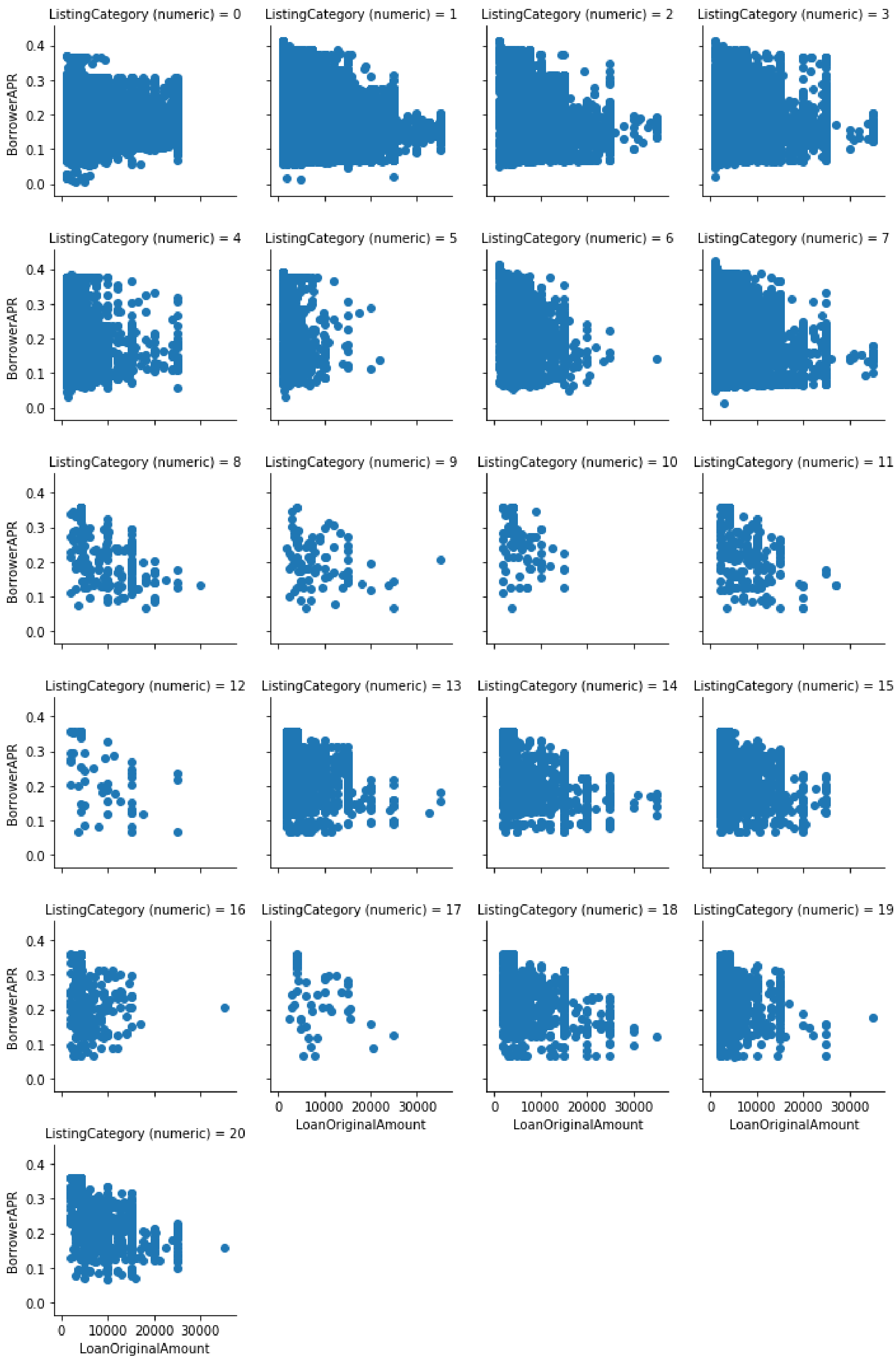
Out[37]: <seaborn.axisgrid.FacetGrid at 0x7fa60137e198>



In [ ]:

In [38]:
```
g = sb.FacetGrid(data = df, col = 'ListingCategory (numeric)', size = 2.5, margin_title
g.map(plt.scatter, 'LoanOriginalAmount', 'BorrowerAPR')
```

Out[38]: <seaborn.axisgrid.FacetGrid at 0x7fa600e4e940>


ListingCategory (numeric) = 0
ListingCategory (numeric) = 1
ListingCategory (numeric) = 2
ListingCategory (numeric) = 3
ListingCategory (numeric) = 4
ListingCategory (numeric) = 5
ListingCategory (numeric) = 6
ListingCategory (numeric) = 7
ListingCategory (numeric) = 8
ListingCategory (numeric) = 9
ListingCategory (numeric) = 10
ListingCategory (numeric) = 11
ListingCategory (numeric) = 12
ListingCategory (numeric) = 13
ListingCategory (numeric) = 14
ListingCategory (numeric) = 15
ListingCategory (numeric) = 16
ListingCategory (numeric) = 17
ListingCategory (numeric) = 18
ListingCategory (numeric) = 19
ListingCategory (numeric) = 20
BorrowerAPR
0.0
0.1
0.2
0.3
0.4
LoanOriginalAmount
0
10000
20000
30000

```
In [ ]:
```

1.6.1 Talk about some of the relationships you observed in this part of the investigation. Were there features that strengthened each other in terms of looking at your feature(s) of interest?

- In general, more the amount of loan increase, lower is the borrower rate.
- Borrower using 12 monthts Term borrow lower amount of loan but have higher monthly loan payement.
- more than 95% of borrower pay monthly loan less than 1000.
- for borrower using 12 and 60 months Term, more the monthly loan payment increase, the borrower rate decrase.
- the few borrowers having monthly loan payment more than around 1200 use only 12 month Term and they have low borrower rate.
- 25000 is the highest Loan Original Amount for 12 months Term.
- 35000 is the highest Loan Original Amount for 36 and 60 months Terms.
- Lower is the Term of payment, higher is the Monthly Loan Payment.
- Not available empoyement status have the lower borrower rate.

1.6.2 Were there any interesting or surprising interactions between features?

- In general, more the amount of loan increase, lower is the borrower rate.
- Borrower using 12 monthts Term borrow lower amount of loan but have higher monthly loan payement.
- the few borrowers having monthly loan payment more than around 1200 use only 12 month Term and they have low borrower rate.
- more than 95% of borrower pay monthly loan less than 1000.
- Not available empoyement status have the lower borrower rate.
- for borrower using 12 and 60 months Term, more the monthly loan payment increase, the borrower rate decrase.

1.7 Conclusions

- There is high correlation between BorrowerAPR and BorrowerRate, also between MonthlyLoanPayment and LoanOriginalAmount.
- **charged off** for loan status, **10-Cosmetic Procedure** for listing category, **teacher's aide** for occupation and **not employed status** have higher borrower rate.
- In general, more the amount of loan increase, lower is the borrower rate.
- Borrower using 12 monthts Term borrow lower amount of loan but have higher monthly loan payement.
- the few borrowers having monthly loan payment more than around 1200 use only 12 month Term and they have low borrower rate.
- more than 95% of borrower pay monthly loan less than 1000.
- Not available empoyement status have the lower borrower rate.
- for borrower using 12 and 60 months Term, more the monthly loan payment increase, the borrower rate decrase.

- **surprise** : Client having a house do not have better borrower rate