Part I - (Exploring the ProsperLoanData Dataset)

by (Olatunji Jola)

Introduction

> The ProsperLoanData Dataset is a large dataset containing 113937 observations and 81 variables. It was put together by Prosper Funding LLC, The first peer to peer online loan company. The dataset contains information about several information about different loan listing from 2005 to 2014. The listing provides different aspect of the listing that can be broadly divided into three observational units, a concise summary of the loan listing, a detailed profile of the borrower and historical data of previous loan by the same borrower and information about the current loan. a more detailed description of the dataset variables is available in the dataset variable defination here.

Table of Content

Preliminary Wrangling

This section is divided into three segmets according to the identified observational unit

- listing
- borrower_profile
- loan
- Dataset Structure

- Home

In [109…
```
# 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
import warnings

%matplotlib inline
```

In [110…
```
# suppress warnings from final output
warnings.filterwarnings('ignore')
```

In [111…
```python
# loading in the dataset
prosper = pd.read_csv('prosperLoanData.csv')
```

> Load in your dataset and describe its properties through the questions below. Try and motivate your exploration goals through this section.

In [112…
```python
print(prosper.shape)
prosper.info()
```

```
(113937, 81)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 81 columns):
 #   Column                               Non-Null Count   Dtype  
---  ------                               --------------   -----  
 0   ListingKey                           113937 non-null  object 
 1   ListingNumber                        113937 non-null  int64  
 2   ListingCreationDate                  113937 non-null  object 
 3   CreditGrade                          28953 non-null   object 
 4   Term                                 113937 non-null  int64  
 5   LoanStatus                           113937 non-null  object 
 6   ClosedDate                           55089 non-null   object 
 7   BorrowerAPR                          113912 non-null  float64
 8   BorrowerRate                         113937 non-null  float64
 9   LenderYield                          113937 non-null  float64
 10  EstimatedEffectiveYield              84853 non-null   float64
 11  EstimatedLoss                        84853 non-null   float64
 12  EstimatedReturn                      84853 non-null   float64
 13  ProsperRating (numeric)              84853 non-null   float64
 14  ProsperRating (Alpha)                84853 non-null   object 
 15  ProsperScore                         84853 non-null   float64
 16  ListingCategory (numeric)            113937 non-null  int64  
 17  BorrowerState                        108422 non-null  object 
 18  Occupation                           110349 non-null  object 
 19  EmploymentStatus                     111682 non-null  object 
 20  EmploymentStatusDuration             106312 non-null  float64
 21  IsBorrowerHomeowner                  113937 non-null  bool   
 22  CurrentlyInGroup                     113937 non-null  bool   
 23  GroupKey                             13341 non-null   object 
 24  DateCreditPulled                     113937 non-null  object 
 25  CreditScoreRangeLower                113346 non-null  float64
 26  CreditScoreRangeUpper                113346 non-null  float64
 27  FirstRecordedCreditLine              113240 non-null  object 
 28  CurrentCreditLines                   106333 non-null  float64
 29  OpenCreditLines                      106333 non-null  float64
 30  TotalCreditLinespast7years           113240 non-null  float64
 31  OpenRevolvingAccounts                113937 non-null  int64  
 32  OpenRevolvingMonthlyPayment          113937 non-null  float64
 33  InquiriesLast6Months                 113240 non-null  float64
 34  TotalInquiries                       112778 non-null  float64
 35  CurrentDelinquencies                 113240 non-null  float64
 36  AmountDelinquent                     106315 non-null  float64
 37  DelinquenciesLast7Years              112947 non-null  float64
 38  PublicRecordsLast10Years             113240 non-null  float64
 39  PublicRecordsLast12Months            106333 non-null  float64
 40  RevolvingCreditBalance               106333 non-null  float64
 41  BankcardUtilization                  106333 non-null  float64
 42  AvailableBankcardCredit              106393 non-null  float64
 43  TotalTrades                          106393 non-null  float64
 44  TradesNeverDelinquent (percentage)   106393 non-null  float64
 45  TradesOpenedLast6Months              106393 non-null  float64
 46  DebtToIncomeRatio                    105383 non-null  float64
 47  IncomeRange                          113937 non-null  object 
```

```
 48  IncomeVerifiable                     113937 non-null  bool
 49  StatedMonthlyIncome                  113937 non-null  float64
 50  LoanKey                              113937 non-null  object
 51  TotalProsperLoans                    22085 non-null   float64
 52  TotalProsperPaymentsBilled           22085 non-null   float64
 53  OnTimeProsperPayments                22085 non-null   float64
 54  ProsperPaymentsLessThanOneMonthLate  22085 non-null   float64
 55  ProsperPaymentsOneMonthPlusLate      22085 non-null   float64
 56  ProsperPrincipalBorrowed             22085 non-null   float64
 57  ProsperPrincipalOutstanding          22085 non-null   float64
 58  ScorexChangeAtTimeOfListing          18928 non-null   float64
 59  LoanCurrentDaysDelinquent            113937 non-null  int64
 60  LoanFirstDefaultedCycleNumber        16952 non-null   float64
 61  LoanMonthsSinceOrigination           113937 non-null  int64
 62  LoanNumber                           113937 non-null  int64
 63  LoanOriginalAmount                   113937 non-null  int64
 64  LoanOriginationDate                  113937 non-null  object
 65  LoanOriginationQuarter               113937 non-null  object
 66  MemberKey                            113937 non-null  object
 67  MonthlyLoanPayment                   113937 non-null  float64
 68  LP_CustomerPayments                  113937 non-null  float64
 69  LP_CustomerPrincipalPayments         113937 non-null  float64
 70  LP_InterestandFees                   113937 non-null  float64
 71  LP_ServiceFees                       113937 non-null  float64
 72  LP_CollectionFees                    113937 non-null  float64
 73  LP_GrossPrincipalLoss                113937 non-null  float64
 74  LP_NetPrincipalLoss                  113937 non-null  float64
 75  LP_NonPrincipalRecoverypayments      113937 non-null  float64
 76  PercentFunded                        113937 non-null  float64
 77  Recommendations                      113937 non-null  int64
 78  InvestmentFromFriendsCount           113937 non-null  int64
 79  InvestmentFromFriendsAmount          113937 non-null  float64
 80  Investors                            113937 non-null  int64
dtypes: bool(3), float64(50), int64(11), object(17)
memory usage: 68.1+ MB
```

Split into three observational units

A quick observation of the dataset reveals that there are three observational units

- Listing details
- Borrower's profile
- Loan data

I will separate these data into these observational units while using the Listingkey as the primary key

In [113…

```
# Splitting the data set into three observational unit with ListingKey as primary key
listing = prosper.iloc[:,0:17].copy()
borrowers_profile = prosper.iloc[:,np.r_[0, 17:50, 66]].copy()
loan = prosper.iloc[:, np.r_[0, 50:80]].drop('MemberKey', axis = 1).copy()
```

In [114…

```
print(listing.shape)
print(listing.info())
```

```
(113937, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 17 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   ListingKey              113937 non-null  object
 1   ListingNumber           113937 non-null  int64
 2   ListingCreationDate     113937 non-null  object
```

```
 3   CreditGrade                28953 non-null   object
 4   Term                       113937 non-null  int64
 5   LoanStatus                 113937 non-null  object
 6   ClosedDate                 55089 non-null   object
 7   BorrowerAPR                113912 non-null  float64
 8   BorrowerRate               113937 non-null  float64
 9   LenderYield                113937 non-null  float64
 10  EstimatedEffectiveYield    84853 non-null   float64
 11  EstimatedLoss              84853 non-null   float64
 12  EstimatedReturn            84853 non-null   float64
 13  ProsperRating (numeric)    84853 non-null   float64
 14  ProsperRating (Alpha)      84853 non-null   object
 15  ProsperScore               84853 non-null   float64
 16  ListingCategory (numeric)  113937 non-null  int64
dtypes: float64(8), int64(3), object(6)
memory usage: 14.8+ MB
None
```

listing

I will focus on cleaning the listing DataFrame in this section.

In [115…

```
listing.head()
```

Out[115]:

	ListingKey	ListingNumber	ListingCreationDate	CreditGrade	Term	LoanStatus	ClosedDate	Bo
0	1021339766868145413AB3B	193129	2007-08-26 19:09:29.263000000	C	36	Completed	2009-08-14 00:00:00	
1	10273602499503308B223C1	1209647	2014-02-27 08:28:07.900000000	NaN	36	Current	NaN	
2	0EE9337825851032864889A	81716	2007-01-05 15:00:47.090000000	HR	36	Completed	2009-12-17 00:00:00	
3	0EF5356002482715299901A	658116	2012-10-22 11:02:35.010000000	NaN	36	Current	NaN	
4	0F023589499656230C5E3E2	909464	2013-09-14 18:38:39.097000000	NaN	36	Current	NaN	

In [116…

```
# Summary statistics of numerical variables
listing.describe()
```

Out[116]:

	ListingNumber	Term	BorrowerAPR	BorrowerRate	LenderYield	EstimatedEffectiveYield	Estim
count	1.139370e+05	113937.000000	113912.000000	113937.000000	113937.000000	84853.000000	848!
mean	6.278857e+05	40.830248	0.218828	0.192764	0.182701	0.168661	
std	3.280762e+05	10.436212	0.080364	0.074818	0.074516	0.068467	
min	4.000000e+00	12.000000	0.006530	0.000000	-0.010000	-0.182700	
25%	4.009190e+05	36.000000	0.156290	0.134000	0.124200	0.115670	
50%	6.005540e+05	36.000000	0.209760	0.184000	0.173000	0.161500	
75%	8.926340e+05	36.000000	0.283810	0.250000	0.240000	0.224300	
max	1.255725e+06	60.000000	0.512290	0.497500	0.492500	0.319900	

Observation

- ListingCreationDate and ClosedDate column should be a datetime object
- ProsperRating (numeric) not neccessary
- ProsperRating (alpha) should be an ordered categorical variable from best to worst
- ProsperScore, CreditGrade and LoanStatus are supposed to be categorical variables
- ListingCategory variable should be more descriptive and should be a categorical variable

DateTime object

- ListingCreationDate and ClosedDate column should be a datetime object

  Convert ListingCreationDate and ClosedDate to Date time object

Code

In [117…
```
listing['ListingCreationDate'] = pd.to_datetime(listing['ListingCreationDate'])
listing['ClosedDate'] = pd.to_datetime(listing['ClosedDate'])
```

Test

In [118…
```
listing[['ListingCreationDate','ClosedDate']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 2 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ListingCreationDate  113937 non-null  datetime64[ns]
 1   ClosedDate           55089 non-null   datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 1.7 MB
```

ProsperRating (numeric)

```
- ProsperRating (numeric) not neccessary
```

Code

In [119…
```
listing.drop('ProsperRating (numeric)', axis = 1, inplace = True)
```

Test

In [120…
```
listing.columns
```

Out[120]:
```
Index(['ListingKey', 'ListingNumber', 'ListingCreationDate', 'CreditGrade',
       'Term', 'LoanStatus', 'ClosedDate', 'BorrowerAPR', 'BorrowerRate',
       'LenderYield', 'EstimatedEffectiveYield', 'EstimatedLoss',
       'EstimatedReturn', 'ProsperRating (Alpha)', 'ProsperScore',
       'ListingCategory (numeric)'],
      dtype='object')
```

Ordered Categorical variable

`Term`, `ProsperRating`, `CreditGrade`, `ProsperScore`, `LoanStatus` are converted to ordered categorical variables

In [121…
```
def ordered_class(list_,dataframe,col, order):


    '''
    creates an ordered class of a categorical variable

    Args:
    list_ (list): Ordered list of the class
    dataframe (DataFrame): The DataFrame on which the categorical variable exist
    col (string): the c column of interest
    order (boolean) a True or False value indicating whether the category should be orde


    returns:
    dataframe[col]:The column that is now converted to categorical variable

    '''
    # creating an ordered category of c class
    class_ = pd.api.types.CategoricalDtype(ordered = order, categories = list_)

    #apply to c_col
    dataframe[col] = dataframe[col].astype(class_)
    return (dataframe[col])
```

In [122…
```
# rename the column ProsperRating (Alpha)
listing.rename(columns = {'ProsperRating (Alpha)': 'ProsperRating'}, inplace = True)
```

In [123…
```
# Converting prosperscore to integer
listing['ProsperScore'].apply(lambda x: x if np.isnan(x) else int(x))

# creating an ordered list of ProsperRating, CreditGrade, ProsperScore, and LoanStatus l

Term = [12,36,60]
ProsperRating = ['HR', 'E', 'D', 'C', 'B', 'A', 'AA']
CreditGrade = ['NC', 'HR', 'E', 'D', 'C', 'B', 'A', 'AA']
ProsperScore= [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0]
LoanStatus = ['Completed', 'FinalPaymentInProgress', 'Current',
              'Cancelled', '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)', 'Defaulted', 'Chargedoff']
```

In [124…
```
categorical_column = [Term, ProsperRating, CreditGrade, ProsperScore, LoanStatus]
column_name = ['Term', 'ProsperRating','CreditGrade','ProsperScore','LoanStatus']

a = 0

for value in categorical_column:

    listing[column_name[a]] = ordered_class(value, listing, column_name[a], True)
    a+=1
```

Test

In [125…
```
listing[column_name].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 5 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Term           113937 non-null  category
 1   ProsperRating  84853 non-null   category
 2   CreditGrade    28953 non-null   category
```

```
 3   ProsperScore   84853 non-null   category
 4   LoanStatus     113937 non-null  category
dtypes: category(5)
memory usage: 558.1 KB
```

In [ ]:

ListingCategory

- rename the `ListingCategory (numeric)` column as `ListingCategory`
- ListingCategory variable should be more descriptive each of the numerical values have a particular meaning

  > 0 - NaN, 1 - Debt Consolidation, 2 - Home Improvement, 3 - Business, 4 - Personal Loan, 5 - Student Use, 6 - Auto, 7- Other, 8 - Baby&Adoption, 9 - Boat, 10 - Cosmetic Procedure, 11 - Engagement Ring, 12 - Green Loans, 13 - Household Expenses, 14 - Large Purchases, 15 - Medical/Dental, 16 - Motorcycle, 17 - RV, 18 - Taxes, 19 - Vacation, 20 - WeddingLoans

- convert the type of the column to a categorical variable

Code

In [126…

```
# rename 'ListingCategory (numeric)' with 'ListingCategory'
listing.rename(columns = {'ListingCategory (numeric)': 'ListingCategory'}, inplace = Tru

# Creating a list of category_number and category_value
category_number = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,


category_value = ['Not Available','Debt Consolidation', 'Home Improvement', 'Business',
            'Personal Loan', 'Student Use', 'Auto', 'Other', 'Baby&Adoption',
            'Boat', 'Cosmetic Procedure', 'Engagement Ring', 'Green Loans',
            'Household Expenses', 'Large Purchases', 'Medical/Dental', 'Motorcycle',
            'RV', 'Taxes', 'Vacation', 'Wedding Loans']
```

In [127…

```
# replace the ListingCategory numeric value with their meanings.
listing['ListingCategory'].replace(category_number, category_value, inplace = True)
```

In [128…

```
# convert to categorical variable using the ordered_class function
listing['ListingCategory'] = ordered_class(category_value, listing,'ListingCategory', Fa
```

Test

In [129…

```
listing['ListingCategory'].head()
```

Out[129]:

```
0       Not Available
1    Home Improvement
2       Not Available
3          Motorcycle
4    Home Improvement
Name: ListingCategory, dtype: category
Categories (21, object): ['Not Available', 'Debt Consolidation', 'Home Improvement', 'Bu
siness', ..., 'RV', 'Taxes', 'Vacation', 'Wedding Loans']
```

In [130…

```
listing['ListingCategory'].dtypes
```

Out[130]:

```
CategoricalDtype(categories=['Not Available', 'Debt Consolidation', 'Home Improvement',
                  'Business', 'Personal Loan', 'Student Use', 'Auto', 'Other',
                  'Baby&Adoption', 'Boat', 'Cosmetic Procedure',
                  'Engagement Ring', 'Green Loans', 'Household Expenses',
```

```
                       'Large Purchases', 'Medical/Dental', 'Motorcycle', 'RV',
                       'Taxes', 'Vacation', 'Wedding Loans'],
, ordered=False)
```

Return

borrowers_profile

I will focus on cleaning the borrowers_profile DataFrame in this section.

In [131…

```
print(borrowers_profile.shape)
borrowers_profile.info()
```

```
(113937, 35)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 35 columns):
 #   Column                              Non-Null Count   Dtype  
---  ------                              --------------   -----  
 0   ListingKey                          113937 non-null  object 
 1   BorrowerState                       108422 non-null  object 
 2   Occupation                          110349 non-null  object 
 3   EmploymentStatus                    111682 non-null  object 
 4   EmploymentStatusDuration            106312 non-null  float64
 5   IsBorrowerHomeowner                 113937 non-null  bool   
 6   CurrentlyInGroup                    113937 non-null  bool   
 7   GroupKey                            13341 non-null   object 
 8   DateCreditPulled                    113937 non-null  object 
 9   CreditScoreRangeLower               113346 non-null  float64
 10  CreditScoreRangeUpper               113346 non-null  float64
 11  FirstRecordedCreditLine             113240 non-null  object 
 12  CurrentCreditLines                  106333 non-null  float64
 13  OpenCreditLines                     106333 non-null  float64
 14  TotalCreditLinespast7years          113240 non-null  float64
 15  OpenRevolvingAccounts               113937 non-null  int64  
 16  OpenRevolvingMonthlyPayment         113937 non-null  float64
 17  InquiriesLast6Months                113240 non-null  float64
 18  TotalInquiries                      112778 non-null  float64
 19  CurrentDelinquencies                113240 non-null  float64
 20  AmountDelinquent                    106315 non-null  float64
 21  DelinquenciesLast7Years             112947 non-null  float64
 22  PublicRecordsLast10Years            113240 non-null  float64
 23  PublicRecordsLast12Months           106333 non-null  float64
 24  RevolvingCreditBalance              106333 non-null  float64
 25  BankcardUtilization                 106333 non-null  float64
 26  AvailableBankcardCredit             106393 non-null  float64
 27  TotalTrades                         106393 non-null  float64
 28  TradesNeverDelinquent (percentage)  106393 non-null  float64
 29  TradesOpenedLast6Months             106393 non-null  float64
 30  DebtToIncomeRatio                   105383 non-null  float64
 31  IncomeRange                         113937 non-null  object 
 32  IncomeVerifiable                    113937 non-null  bool   
 33  StatedMonthlyIncome                 113937 non-null  float64
 34  MemberKey                           113937 non-null  object 
dtypes: bool(3), float64(22), int64(1), object(9)
memory usage: 28.1+ MB
```

In [132…

```
borrowers_profile.head()
```

Out[132]:

	ListingKey	BorrowerState	Occupation	EmploymentStatus	EmploymentStatusDuration	IsBorrov
0	1021339766868145413AB3B	CO	Other	Self-employed	2.0	

1	10273602499503308B223C1	CO	Professional	Employed	44.0
2	0EE9337825851032864889A	GA	Other	Not available	NaN
3	0EF5356002482715299901A	GA	Skilled Labor	Employed	113.0
4	0F023589499656230C5E3E2	MN	Executive	Employed	44.0

5 rows × 35 columns

In [133…

```
# summary statistics of numerical variables
borrowers_profile.describe()
```

Out[133]:

	EmploymentStatusDuration	CreditScoreRangeLower	CreditScoreRangeUpper	CurrentCreditLines	OpenCredi
count	106312.000000	113346.000000	113346.000000	106333.000000	106333.0
mean	96.071582	685.567731	704.567731	10.317192	9.2
std	94.480605	66.458275	66.458275	5.457866	5.0
min	0.000000	0.000000	19.000000	0.000000	0.0
25%	26.000000	660.000000	679.000000	7.000000	6.0
50%	67.000000	680.000000	699.000000	10.000000	9.0
75%	137.000000	720.000000	739.000000	13.000000	12.0
max	755.000000	880.000000	899.000000	59.000000	54.0

8 rows × 23 columns

Observation

- The IncomeRange column has 2 variables the lower and upper bound of income
- DateCreditPulled and FirstRecordedcreditLine columns are DateTime objects

IncomRange

The IncomeRange column has 2 variables the lower and upper bound of income

- extract the income lower bound into a new column `IncomeLowerBound` and
- upper bound into a new column `IncomeUpperBound`
- drop the `IncomeRange` column

Code

In [134…

```
# Extract lowerbound of income
borrowers_profile['IncomeLowerBound'] = borrowers_profile['IncomeRange'].str.extract(r'\

# Remove the middle comma and convert to float
borrowers_profile['IncomeLowerBound'] = borrowers_profile['IncomeLowerBound'].str.replac
```

In [135…
```
# Extract upperbound of income
borrowers_profile['IncomeUpperBound'] = borrowers_profile['IncomeRange'].str.extract(r'\

# Remove the middle comma and convert to float
borrowers_profile['IncomeUpperBound'] = borrowers_profile['IncomeUpperBound'].str.replac
```

In [136…
```
# drop IncomeRange column
borrowers_profile.drop('IncomeRange', axis =1, inplace = True )
```

Test

In [137…
```
borrowers_profile[['IncomeLowerBound','IncomeUpperBound']].head(1)
```

Out[137]:

	IncomeLowerBound	IncomeUpperBound
0	25000.0	49999.0

DateTime Objects

- convert the `DateCreditPulled` column and the `FirstRecordedCreditLine` column to datetime objects.

Code

In [138…
```
borrowers_profile['DateCreditPulled'] = pd.to_datetime(borrowers_profile['DateCreditPull
```

In [139…
```
borrowers_profile['FirstRecordedCreditLine'] = pd.to_datetime(borrowers_profile['FirstRe
```

Test

In [140…
```
borrowers_profile[['DateCreditPulled', 'FirstRecordedCreditLine']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 2 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   DateCreditPulled         113937 non-null  datetime64[ns]
 1   FirstRecordedCreditLine  113240 non-null  datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 1.7 MB
```

Return

loan

Here is just a brief overview of the loan DataFrame

In [141…
```
print(loan.shape)
loan.info()
```

```
(113937, 30)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 30 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
```

```
 0   ListingKey                           113937 non-null  object
 1   LoanKey                              113937 non-null  object
 2   TotalProsperLoans                    22085 non-null   float64
 3   TotalProsperPaymentsBilled           22085 non-null   float64
 4   OnTimeProsperPayments                22085 non-null   float64
 5   ProsperPaymentsLessThanOneMonthLate  22085 non-null   float64
 6   ProsperPaymentsOneMonthPlusLate      22085 non-null   float64
 7   ProsperPrincipalBorrowed             22085 non-null   float64
 8   ProsperPrincipalOutstanding          22085 non-null   float64
 9   ScorexChangeAtTimeOfListing          18928 non-null   float64
 10  LoanCurrentDaysDelinquent            113937 non-null  int64
 11  LoanFirstDefaultedCycleNumber        16952 non-null   float64
 12  LoanMonthsSinceOrigination           113937 non-null  int64
 13  LoanNumber                           113937 non-null  int64
 14  LoanOriginalAmount                   113937 non-null  int64
 15  LoanOriginationDate                  113937 non-null  object
 16  LoanOriginationQuarter               113937 non-null  object
 17  MonthlyLoanPayment                   113937 non-null  float64
 18  LP_CustomerPayments                  113937 non-null  float64
 19  LP_CustomerPrincipalPayments         113937 non-null  float64
 20  LP_InterestandFees                   113937 non-null  float64
 21  LP_ServiceFees                       113937 non-null  float64
 22  LP_CollectionFees                    113937 non-null  float64
 23  LP_GrossPrincipalLoss                113937 non-null  float64
 24  LP_NetPrincipalLoss                  113937 non-null  float64
 25  LP_NonPrincipalRecoverypayments      113937 non-null  float64
 26  PercentFunded                        113937 non-null  float64
 27  Recommendations                      113937 non-null  int64
 28  InvestmentFromFriendsCount           113937 non-null  int64
 29  InvestmentFromFriendsAmount          113937 non-null  float64
dtypes: float64(20), int64(6), object(4)
memory usage: 26.1+ MB
```

In [142…

```
loan.head()
```

Out[142]:

	ListingKey	LoanKey	TotalProsperLoans	TotalProsperPaymentsBilled	OnTimePr
0	1021339766868145413AB3B	E33A3400205839220442E84	NaN	NaN	
1	10273602499503308B223C1	9E3B37071505919926B1D82	NaN	NaN	
2	0EE9337825851032864889A	6954337960046817851BCB2	NaN	NaN	
3	0EF5356002482715299901A	A0393664465886295619C51	NaN	NaN	
4	0F023589499656230C5E3E2	A180369302188889200689E	1.0	11.0	

5 rows × 30 columns

In [143…

```
loan.describe()
```

Out[143]:

	TotalProsperLoans	TotalProsperPaymentsBilled	OnTimeProsperPayments	ProsperPaymentsLessThanOneMont
count	22085.000000	22085.000000	22085.000000	22085.0
mean	1.421100	22.934345	22.271949	0.6
std	0.764042	19.249584	18.830425	2.4
min	0.000000	0.000000	0.000000	0.0
25%	1.000000	9.000000	9.000000	0.0
50%	1.000000	16.000000	15.000000	0.0
75%	2.000000	33.000000	32.000000	0.0

max	8.000000	141.000000	141.000000	42.0

8 rows × 26 columns

Observation

- `LoanOriginationDate` column should be a datetime object

convert `LoanOriginationDate` column to a datetime object

Code

In [144…
```
loan['LoanOriginationDate'] = pd.to_datetime(loan['LoanOriginationDate'])
```

Test

In [145…
```
loan['LoanOriginationDate'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 113937 entries, 0 to 113936
Series name: LoanOriginationDate
Non-Null Count   Dtype
--------------   -----
113937 non-null  datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 890.3 KB
```

In [146…
```
listing.to_csv('Prosper_Listing.csv', index = False)
borrowers_profile.to_csv('Prosper_Borrowers_profile.csv', index = False)
loan.to_csv('Prosper_Loan.csv', index = False)
```

What is the structure of your dataset?

> There are 113937 loan listing in the data set with 81 variables. These variables can be divided into three main observational units. I have divided the dataset into these three DataFrames namely, **listing**, **borrowers_profile** and **loan** DataFrame in keeping with a tidy data condition of keeping each observation in the appropriate observational units.
>
> - The `listing` dataframe has 16 features which provides a very concise information about the loan listing.
>   - There are 6 categorical variables (Term, LoanStatus, CreditGrade, ProsperRating, ProsperScore and ListingCategory) providing different metrics by which the loans could be classified. 5 of them are ordered while one is not ordered i.e, ListingCategory.
>   - There are 6 numerical variable quantifying the listing under various headings like (BorrowerAPR, BorrowerRate, LenderYield, EstimatedEffectiveYield, EstimatedLoss and EstimatedReturn)
>   - There are also two datetime objects the ListingCreationDate (The date the listing was created) and the Closed date (The closing date for listings that are no longer active)
>   - There are also two identification features. The ListingKey (This is the primary key for the three dataframes and is unique for each observaion in the dataset) and the ListingNumber (also unique for each observation in the dataset).

- The `borrowers_profile` dataframe contains 34 features apart from the primary key, the ListingKey. Each of these features provide important background information about the borrower. These information will be very vital for the decision making of the lender and will be a great asset in uncovering patterns across the entire dataset. Some important features in these dataframe are Occupation, EmploymentStatus, IsBorrowerHomeowner, CreditScoreRangeLower, CreditScoreRangeUpper, CurrentDelinquencies, DelinquenciesLast7Years, IncomeRange, DebtToIncomeRatio, StatedMonthlyIncome and a host of others. Most of them will be strong predictor of major target features like BorrowerAPR, LoanStatus and ProsperRating.

- The `loan` dataframe contains the third observational unit. It has information about the loan itself like LoanOriginalAmount, LoanOriginationDate, MonthlyLoanPayment, investors. Also, It shows a lot of historical information about previous loans on Prosper platform by borrowers like ScorexChangeAtTimeofListing, TotalProsperLoans, TotalProsperPaymentsBilled, OnTimeProsperPayments, ProsperPaymentsLessThanOneMonthLate, ProsperPaymentsOneMonthPlusLate. These features will be a good indicator of the ProsperRating of the borrower and therfore, a good indicator of the BorrowerAPR.

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

I am most interested in figuring out what are the best features for predicting the LoanStatus and the Borrower Annual Percentage Rate (BorrowerAPR.)

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

I expect that the CreditScore will have the strongest effect on BorrowerAPR and LoanStatus. I also think that other factors that will have significant effect are EmploymentStatus, ProsperRating, ProsperScore, DebtToIncomeRatio and IsBorrowerHomeowner.

Return

Univariate Exploration

In the section the focus is on observing the distribution of individual variable and to look for interesting pattern that will direct further investigatios.

I have divided the section into three based on the observational unit division.

- listing
- borrowers_profile
- loan
- Discussion

- Home

I will start by looking at the distribution of the main variable of interest: LoanStatus and BorrowerAPR

listing

I will explore this listing DataFrame under four broad heading

- Target Features
- Other Numerical features
- Ordered Categorical variables
- Norminal categorical Variable

- Return

In [147…

```
listing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 16 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ListingKey               113937 non-null  object
 1   ListingNumber            113937 non-null  int64
 2   ListingCreationDate      113937 non-null  datetime64[ns]
 3   CreditGrade              28953 non-null   category
 4   Term                     113937 non-null  category
 5   LoanStatus               113937 non-null  category
 6   ClosedDate               55089 non-null   datetime64[ns]
 7   BorrowerAPR              113912 non-null  float64
 8   BorrowerRate             113937 non-null  float64
 9   LenderYield              113937 non-null  float64
 10  EstimatedEffectiveYield  84853 non-null   float64
 11  EstimatedLoss            84853 non-null   float64
 12  EstimatedReturn          84853 non-null   float64
 13  ProsperRating            84853 non-null   category
 14  ProsperScore             84853 non-null   category
 15  ListingCategory          113937 non-null  category
dtypes: category(6), datetime64[ns](2), float64(6), int64(1), object(1)
memory usage: 9.3+ MB
```

Target Features

LoanStatus and BorrowerAPR

In [148…

```
sb.color_palette()[0]
```

Out[148]:

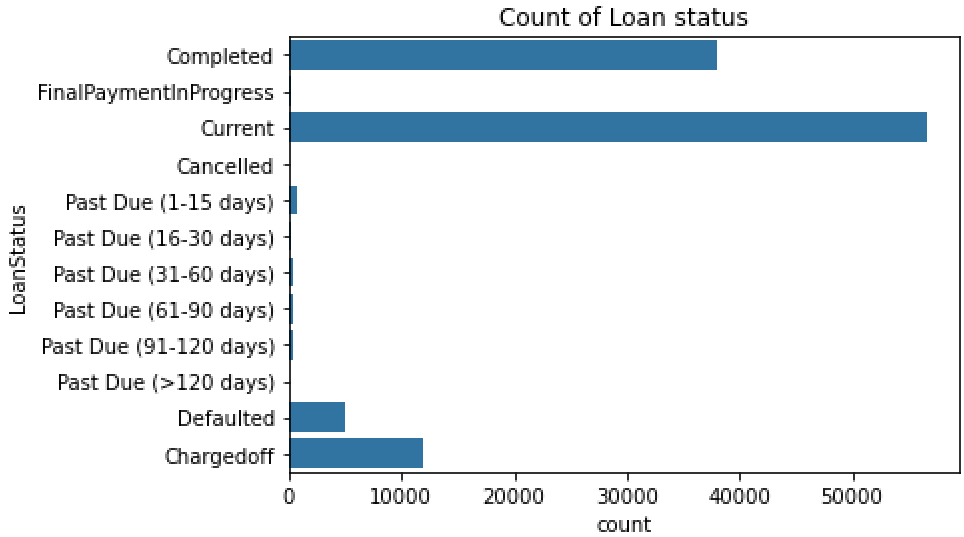
```
(0.12156862745098039, 0.4666666666666667, 0.7058823529411765)
```

LoanStatus

- how is the loan Status distributed ?

In [149…

```
# The count of Loanstatus in base scale
color = sb.color_palette()[0]
sb.countplot(data = listing, y = 'LoanStatus', color = color)
plt.title('Count of Loan status');
```



Observation

- It can be observed that the count range for the loan status levels is really wide and cannot be properly displayed on a linear scale.
- The lowest count value is 5 for cancelled while the largest count is 56576 for current

In [150…

```
status_counts = listing.LoanStatus.value_counts(sort = False)
status_counts
```

Out[150]:

```
Completed                 38074
FinalPaymentInProgress      205
Current                   56576
Cancelled                     5
Past Due (1-15 days)        806
Past Due (16-30 days)       265
Past Due (31-60 days)       363
Past Due (61-90 days)       313
Past Due (91-120 days)      304
Past Due (>120 days)         16
Defaulted                  5018
Chargedoff                11992
Name: LoanStatus, dtype: int64
```

LoanStatus on a log scale

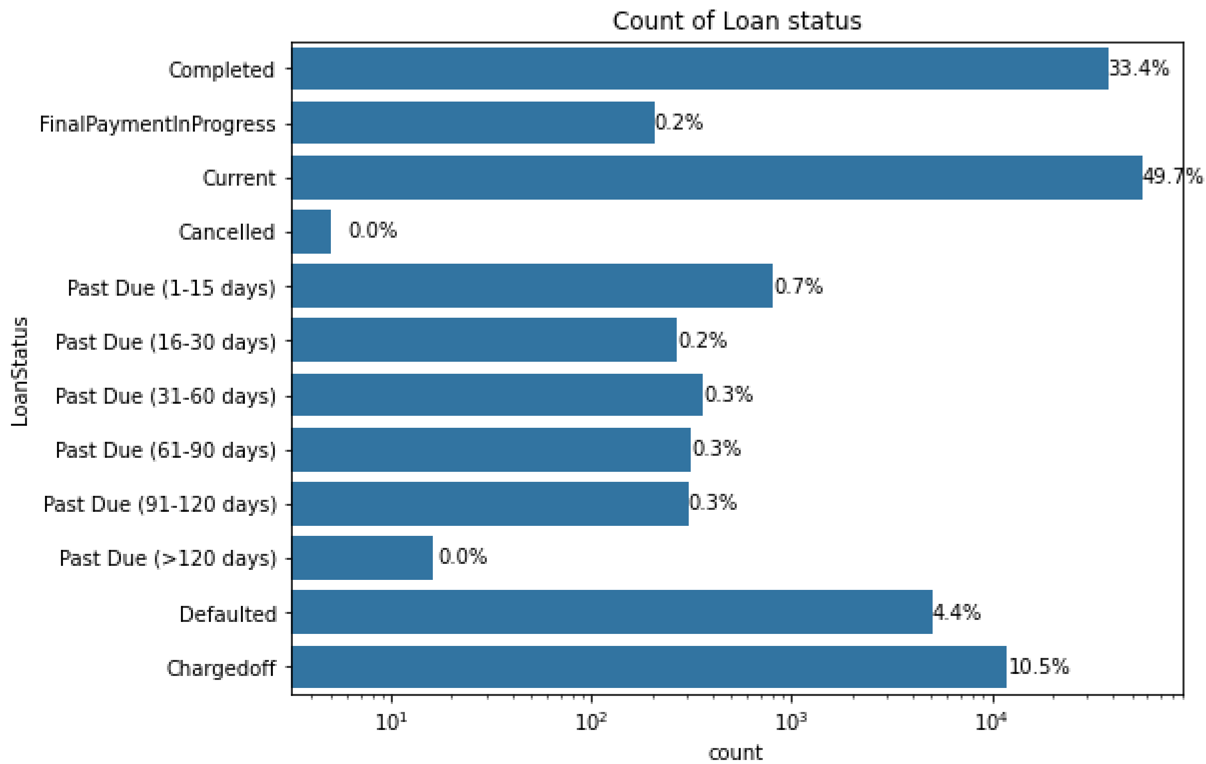
- How does the LoanStatus look like on a log scale ?

In [151…

```
status_order = status_counts.index
```

In [152…

```
# The count of LoanStatus in logscale
plt.figure(figsize = (8,6))
color = sb.color_palette()[0]
sb.countplot(data = listing, y = 'LoanStatus', color = color)
plt.xscale('log')
plt.title('Count of Loan status')

# defining the rate
for i in range(status_counts.shape[0]):
    count = status_counts[i]
    n_status = sum(status_counts)
    pct_string = '{:0.1f}%'.format(100*count/n_status)

    # placing the text
    plt.text(count+1, i, pct_string, va = 'center')
```



Observation

- It can be observed that 49.7% of the listings are **current** while 33.4% are **completed**
- 4.4% are defaulted and 10.5% are chargedoff
- **Cancelled** and **Past Due (>120days)** showed 0.0%. This is due to the level of precision set in the text formatting. they are actually, 0.004% and 0.014% respectively. They are the least occuring status.

BorrowerAPR distribution

- How is the BorrowerAPR distributed ?

In [153…

```
listing[['BorrowerAPR', 'BorrowerRate']].describe()
```

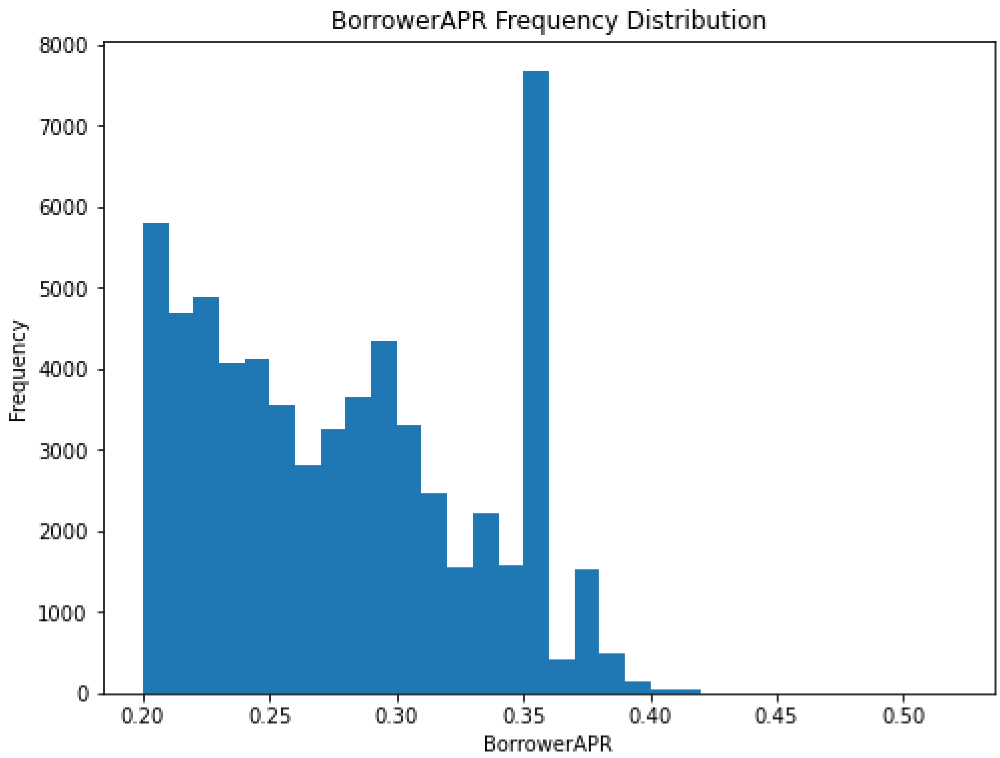
Out[153]:

	BorrowerAPR	BorrowerRate
count	113912.000000	113937.000000
mean	0.218828	0.192764
std	0.080364	0.074818
min	0.006530	0.000000
25%	0.156290	0.134000
50%	0.209760	0.184000
75%	0.283810	0.250000
max	0.512290	0.497500

In [154…

```
# Plotting the distribution of the BorrowerAPR
plt.figure(figsize = (8,6))
bin = np.arange(0.2, listing['BorrowerAPR'].max()+0.01, 0.01)
plt.hist(data = listing, x = 'BorrowerAPR', bins = bin);
plt.xlabel('BorrowerAPR')
```

```python
plt.ylabel('Frequency')
plt.title('BorrowerAPR Frequency Distribution');
```



Observation

- The overall trend of the distribution is that as the Borrower Annual Percentage Rate increases the count of Loans in the dataset reduces. This is quite reasonable since most people will rather go for cheaper loans than more expensive ones. Therefore people will always device means to ensure that they pay less.
- The trend has spikes at interval as it trends downwards. The most notable spikes are at 20%, 29%, 33%, 35% and 37%.
- The spike at the 35% APR forms the highest peak of the distribution and it is very much against the trend. This is quite an interesting point and require further investigation.
- The lower boundary of the BorrowerAPR seem to be clipped at 0.20. indicating that the least annual percentage rate on the dataset is 20%

Other Numerical features

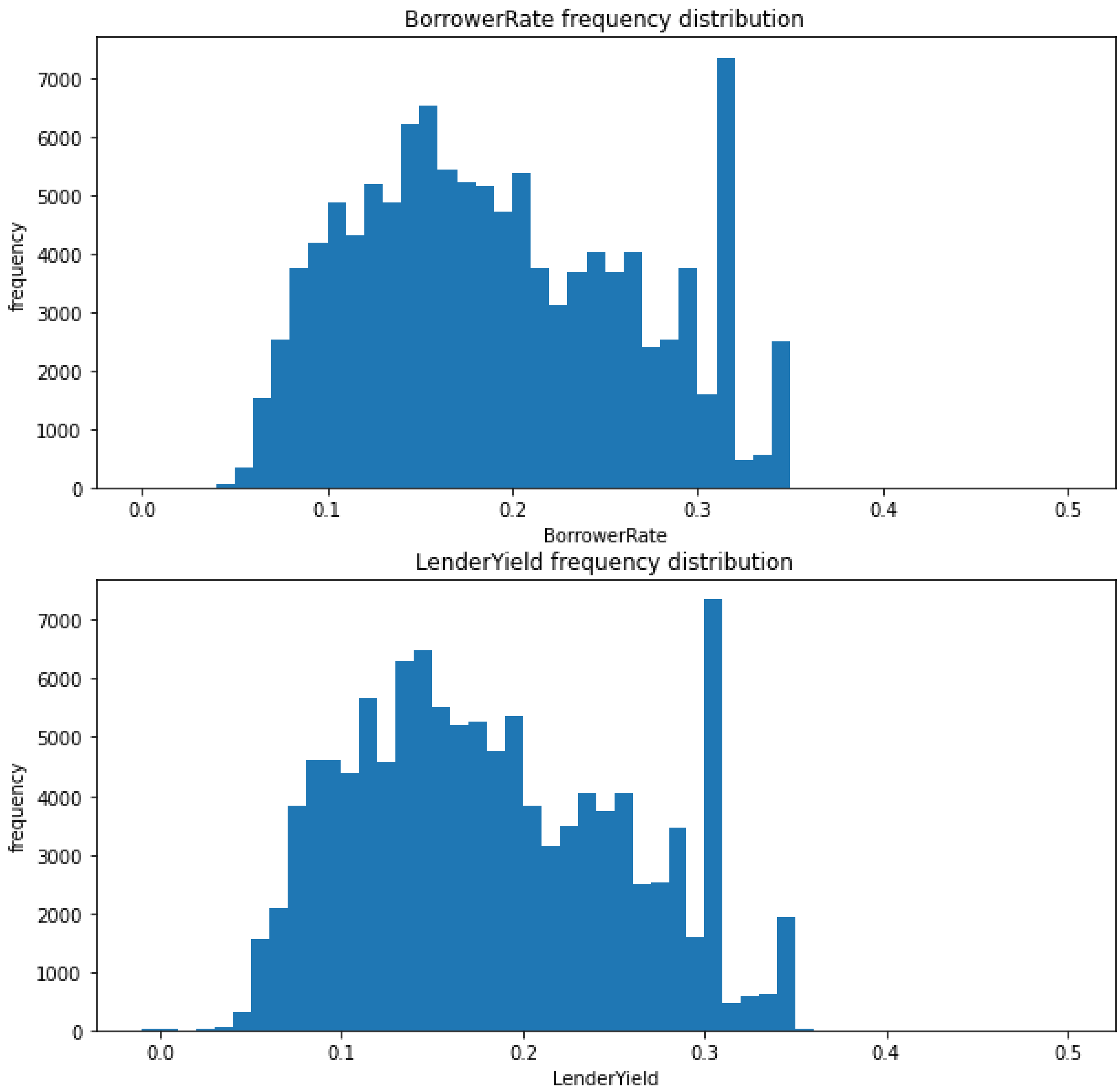
Let us consider the distribution of other numerical features in the listing dataframe i.e BorrowerRate, LenderYield, EstimatedEffectiveYield, EstimatedLoss and EstimatedReturn

BorrowerRate and LenderYield

In [155…

```python
# Plotting the BorrowerRate and LenderYield features distribution
fig, ax = plt.subplots(nrows = 2, figsize = [10,10])

features = ['BorrowerRate', 'LenderYield']

for i in range(len(features)):
    var = features[i]
```

```
    bins = np.arange(min(listing[var]), max(listing[var])+0.01, 0.01)
    ax[i].hist(data = listing, x = var, bins = bins)
    ax[i].set_xlabel('{}'.format(var))
    ax[i].set_title('{} frequency distribution'.format(var))
    ax[i].set_ylabel('frequency')
```



Observation

- LenderYield and BorrorwerRate has the same distribution as BorrowerAPR.
- It can be observed that the BorrowerRate and LenderYield are slightly shifted to the left with respect to the BorrowerAPR. This indicates that the three variable have the same base value. Looking through the variable definitions, it was confirmed that the Borrower's Annual Percentage Rate (BorrowerAPR) is the annualized value of the borrower's rate plus all other fee the borrower will pay for obtaining the Loan.

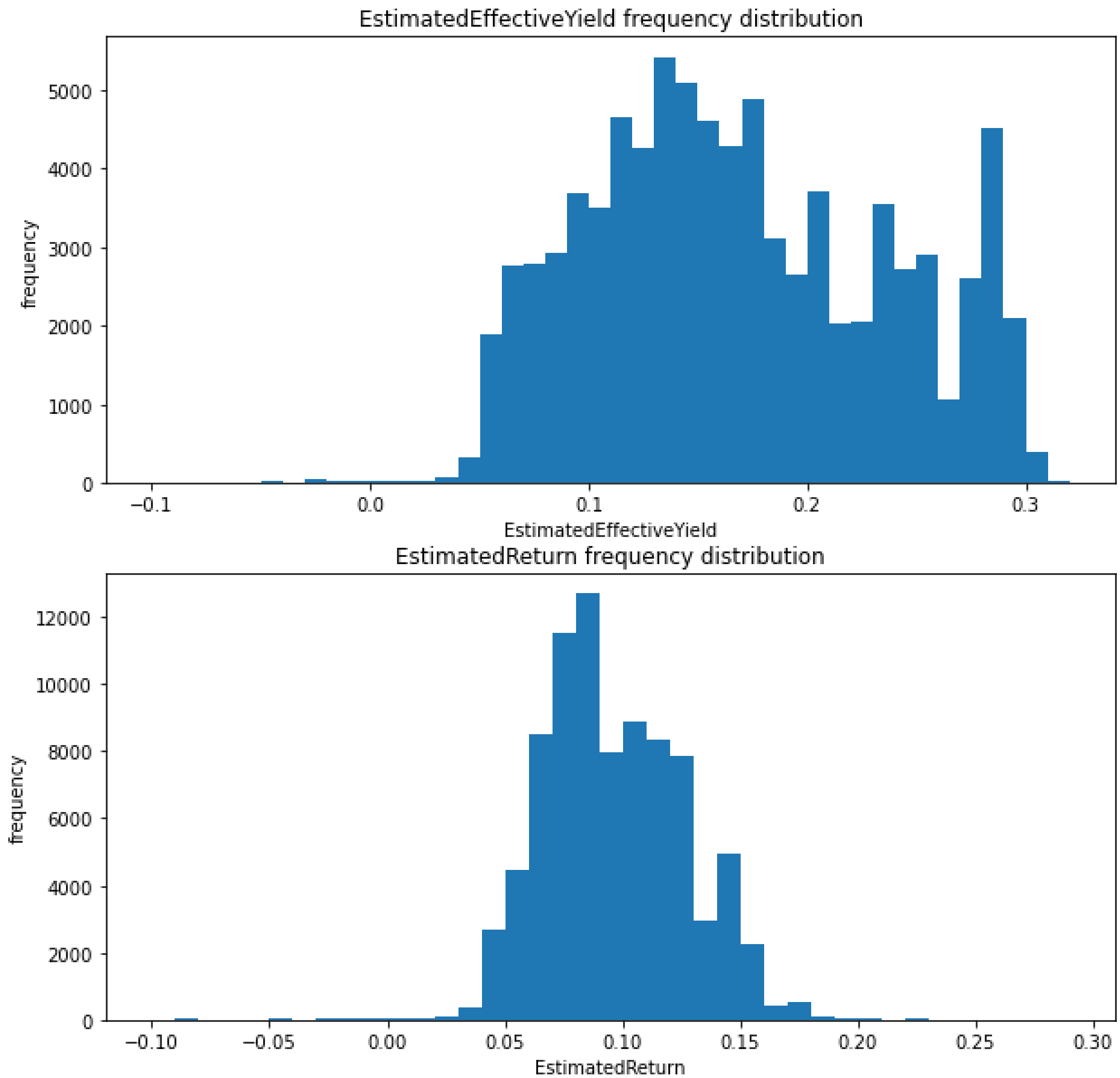
EstimatedEffectiveYield and EstimatedReturn

will the EstimatedEffectiveYield and EstimatedReturn show the same pattern as the previous numerical fields ?

In [156…

```
fig, ax = plt.subplots(nrows = 2, figsize = (10,10))

features = ['EstimatedEffectiveYield', 'EstimatedReturn']
```

```
for i in range(len(features)):
    var = features[i]
    bins = np.arange(-0.1, listing[var].max()+0.01, 0.01)
    ax[i].hist(data = listing, x = var, bins = bins)
    ax[i].set_xlabel('{}'.format(var))
    ax[i].set_ylabel('frequency')
    ax[i].set_title('{} frequency distribution'.format(var))
```



Observation

- Shows the same over all pattern as BorrowerAPR since they all have the same base value in the BorrowerRate.
- We can also observe that the distribution is further shifted to the left since it represents further deduction from the BorrowerRate.

EstimatedLoss

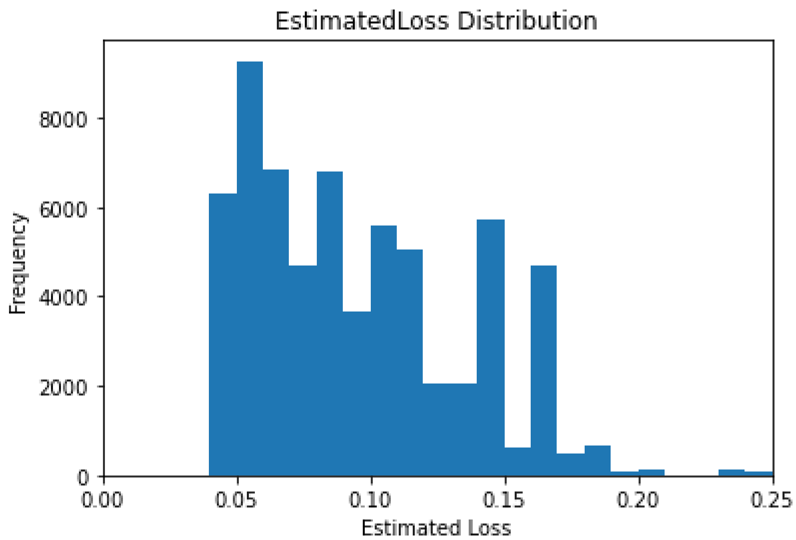
- What about the EstimatedLoss ?

In [157…
```
bin = np.arange(0.04, listing.EstimatedLoss.max()+0.01, 0.01)
plt.hist(data = listing, x = 'EstimatedLoss', bins = bin)
```

```
plt.xlabel('Estimated Loss');
plt.ylabel('Frequency')
plt.title('EstimatedLoss Distribution')
plt.xlim(0,0.25)
```

Out[157]:

```
(0.0, 0.25)
```



- This shows a similar pattern as BorrowerAPR.
- Trending downwards and suggesting that higher estimated loss value occurences are less in the distribution. Although, there are spikes at various estimated loss value ranges like around 0.16%, 0.14%, 0.11%, 0.8% and a highest peak at around 0.05%

Ordered Categorical variables.

here is the list of all ordered categorical features in our listing dataframe `Term` , `ProsperRating` , `CreditGrade` , `ProsperScore` and `LoanStatus` . How are they distributed ?

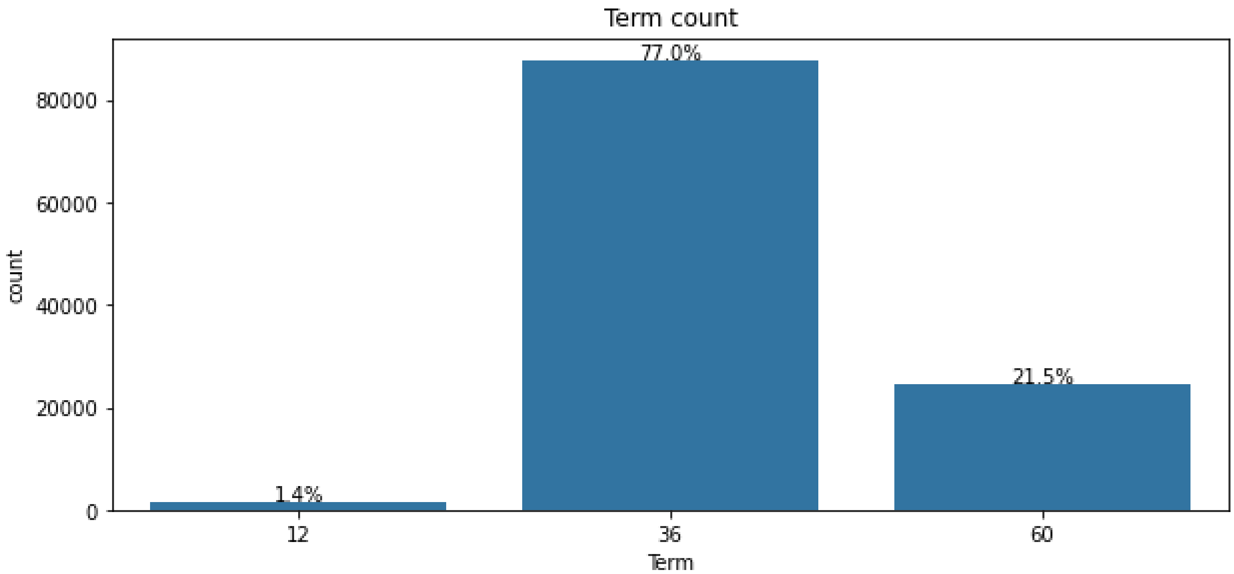
In [158…

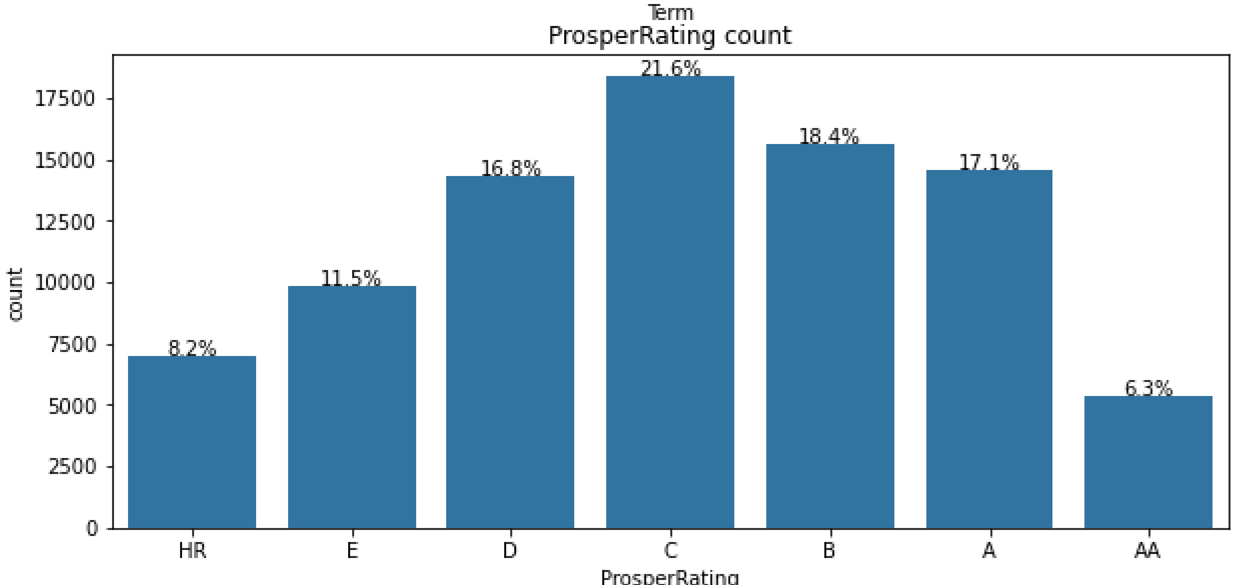
```
# defining the rate
for i in range(status_counts.shape[0]):
    count = status_counts[i]

    n_status = sum(status_counts)
    pct_string = '{:0.1f}%'.format(100*count/n_status)

    # placing the text
 #   plt.text(count+1, i, pct_string, va = 'center')
```

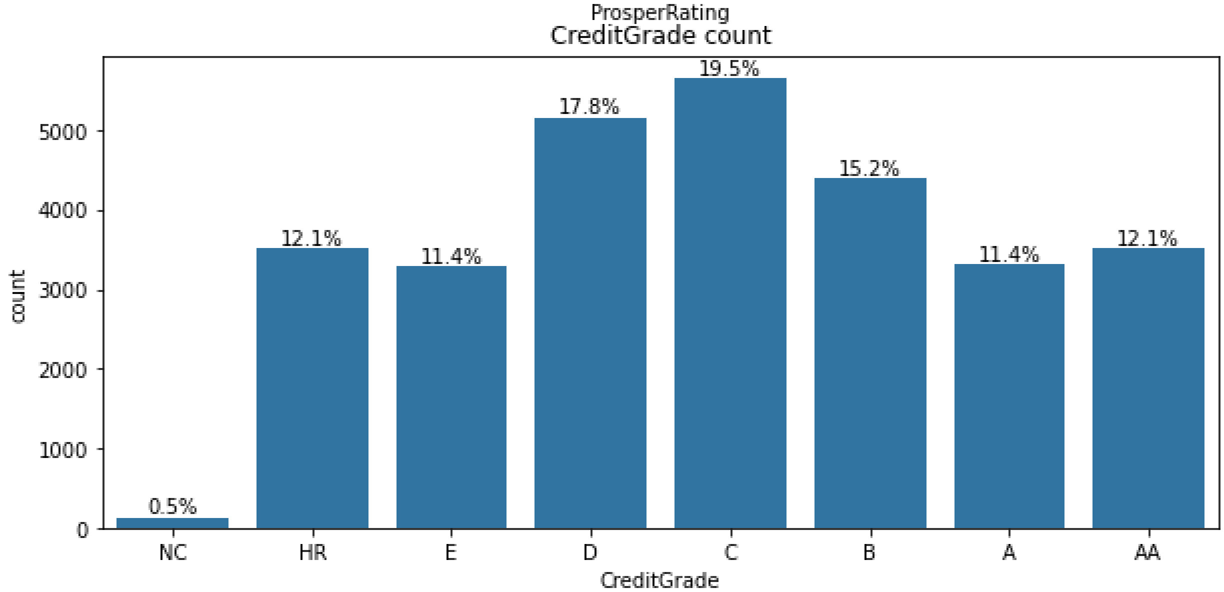
In [159…

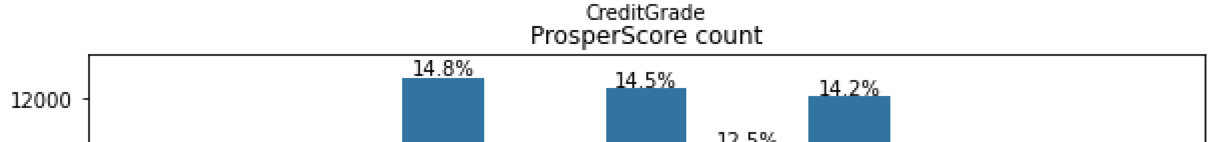
```
fig, ax = plt.subplots(nrows = 4, figsize = (10,20))


feature = ['Term', 'ProsperRating', 'CreditGrade', 'ProsperScore']
color = sb.color_palette()[0]

# plot for each features
for i in range(len(feature)):
    var = feature[i]
    g = sb.countplot(data = listing, x = var, ax=ax[i], color = color)
    ax[i].set_xlabel('{}'.format(var))
    ax[i].set_title('{} count'.format(var))

    var_level_count = listing[var].value_counts(sort = False)
    n_var = sum(var_level_count)
```

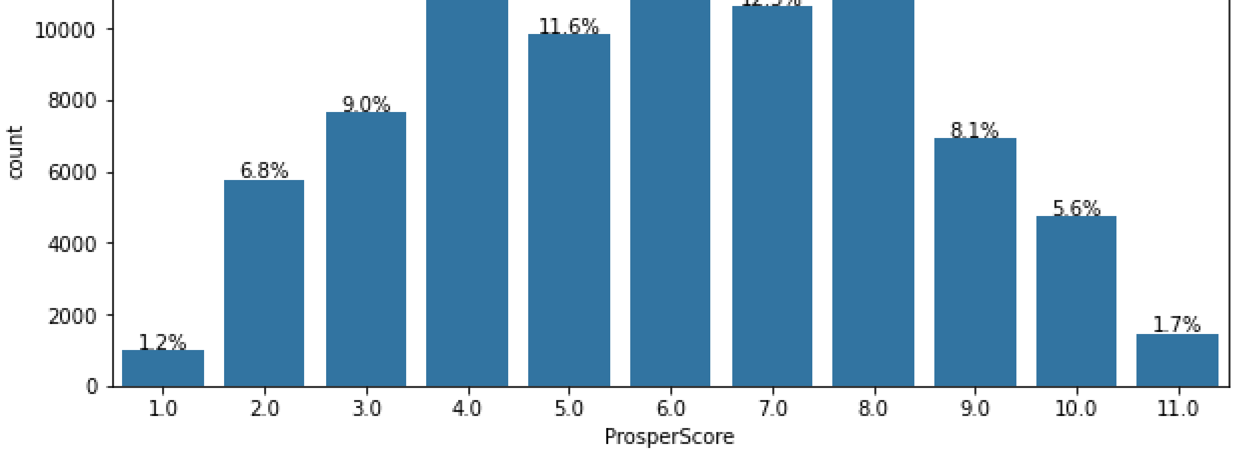
```
# annotate the bars
    for p in g.patches:

        g.annotate('{:0.1f}%'.format(100*p.get_height()/n_var),
                   (p.get_x()+p.get_width()/2, p.get_height()+50),
                   horizontalalignment = 'center')
```











Observation

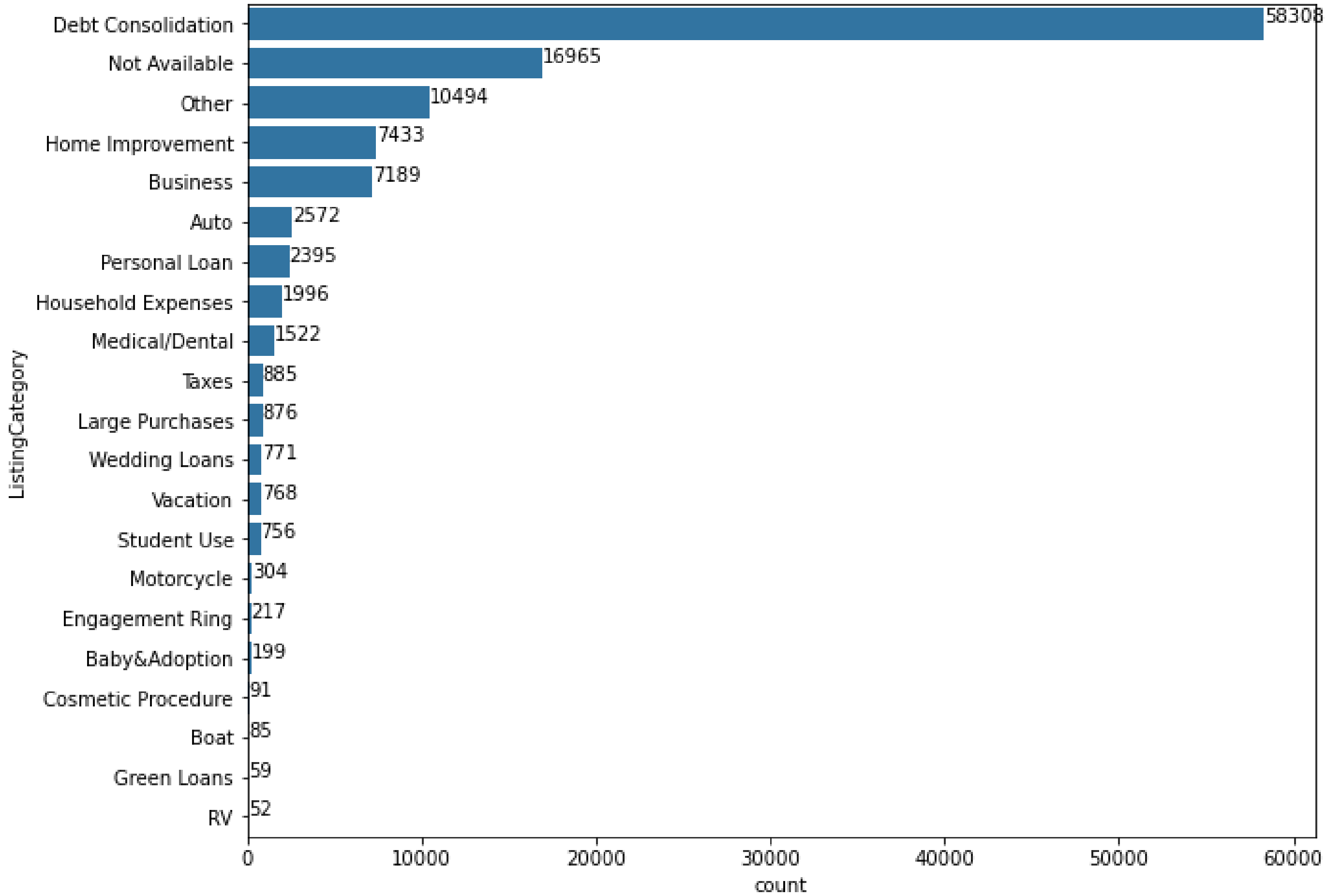
- Most of the categorical plots approximate to a normal distribution. where the modal class is in the center of the distribution
- ProsperScore is trimodal with almost equal peaks at prosperscore level, 4, 6 and 8 they each have around 14% occurrence in the dataset
- ProsperRating and CreditGrade are slightly skewed to the left with more data points on the left side of the modal class. This suggest that it gets more and more difficult to obtain higher CreditGrade and ProsperScore value especially, beyond the modal class value.

Norminal categorical Variables

-The only norminal categorical variable we have is ListingCategory. how does it look ?

In [160…

```
# value count and value count index for order
value_count = listing['ListingCategory'].value_counts()
value_count_index = value_count.index
sum_value = value_count.sum()

# plotting on  linear scale
plt. figure(figsize = (10,8))
g = sb.countplot(data = listing, y = 'ListingCategory', color = color, order = value_cou

# annotate the bars
for p in g.patches:
    g.annotate('{}'.format(p.get_width()),
                ((p.get_x()+p.get_width()), p.get_y()),horizontalalignment = 'left',
                 verticalalignment = 'top')
```



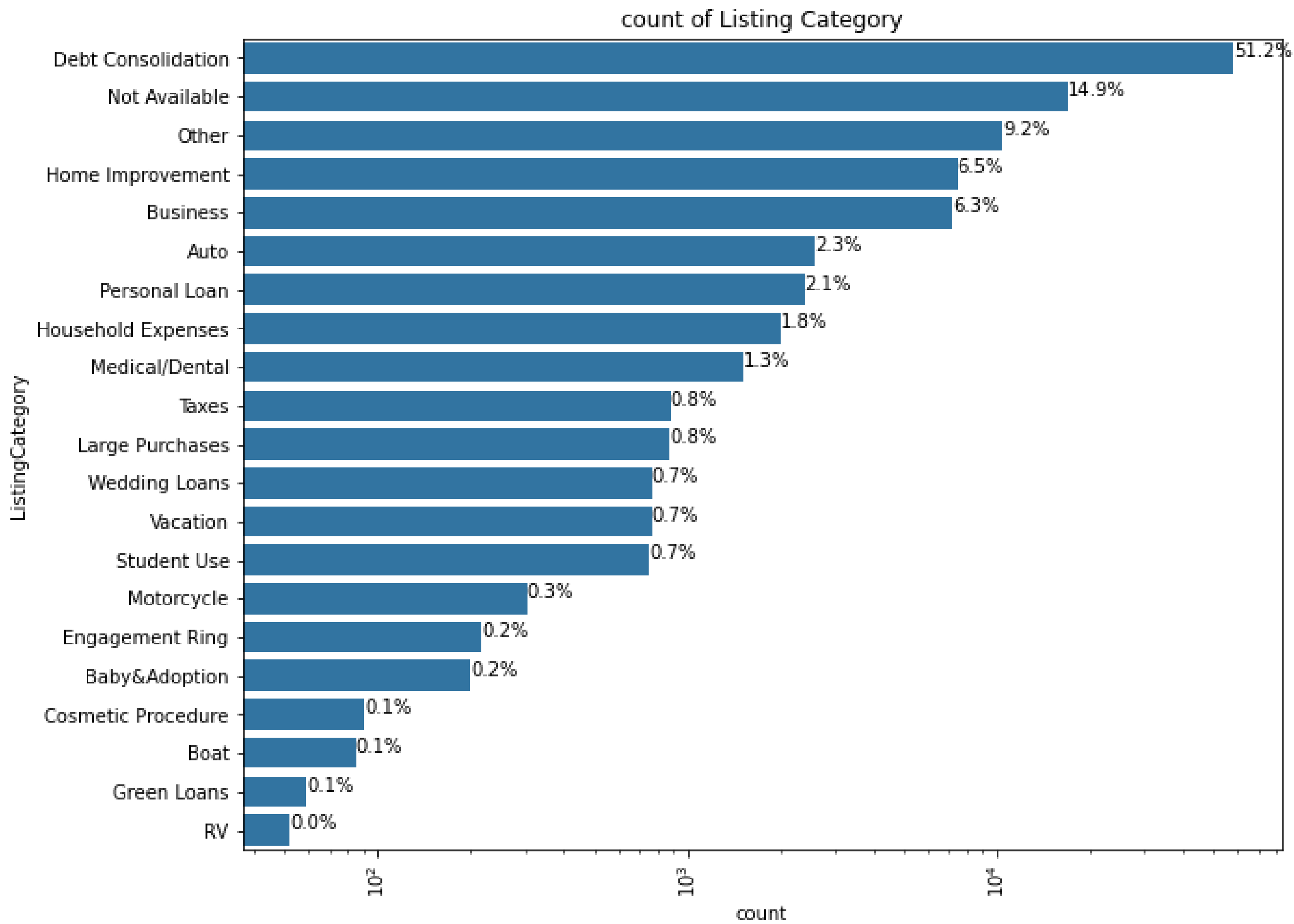
Observation

- This plot shows that the major reason people use Prosper Loan srvice is for Debt Consolidation. This suggest that they have a very low BorrowerAPR as compared to other Loan service outlet. It will be a great idea to explore this further if sufficient information are available.
- The count value is spread widely form close to zero to over 58,000. It will be better to use a logarithm scale to visualize the distribution n bettter.

Plotting ListingCategory on a log scale

In [161…

```
# plotting on  logarithmic scale
plt. figure(figsize = (10,8))


g = sb.countplot(data = listing, y = 'ListingCategory', color = color, order = value_cou
plt.xticks(rotation = 90);
plt.title('count of Listing Category')
plt.xscale('log');


# annotate the bars
for p in g.patches:
    g.annotate('{:.1f}%'.format(100*p.get_width()/sum_value),
                    ((p.get_x()+p.get_width()), p.get_y()),horizontalalignment = 'left',
                     verticalalignment = 'top')
```



Observation

- Debt consolidation is the single most important reason people take out loans at prosper loan and it accounts for over 50% of all loans taken between 2005 and 2014.

In [162…

```
listing.ListingCategory.info()
```

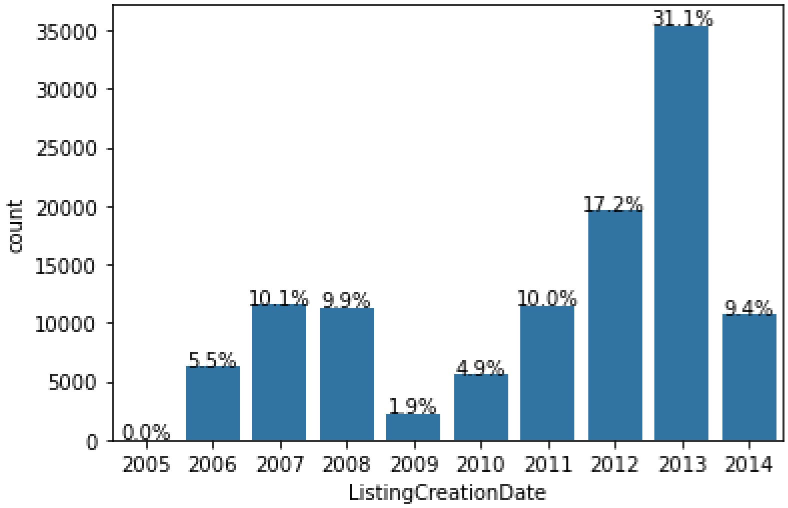
```
<class 'pandas.core.series.Series'>
RangeIndex: 113937 entries, 0 to 113936
Series name: ListingCategory
Non-Null Count   Dtype
--------------   -----
113937 non-null  category
dtypes: category(1)
memory usage: 112.1 KB
```

ListingCreationDate

- Can any significant pattern be observed in the ListingCreationDate distribution ?

In [163…

```
g = sb.countplot(data = listing, x = listing.ListingCreationDate.apply(lambda x: x.year)

total = listing.ListingCreationDate.value_counts().sum()
for p in g.patches:
    g.annotate('{:0.1f}%'.format(100*p.get_height()/total),
                       (p.get_x()+p.get_width()/2, p.get_height()+50),
                       horizontalalignment = 'center')
```



Observation

- Two significant bar stands out.
    - In 2009, they processed the least number of loans at 1.9% of the total listing in the dataset and just around 3000 listings
    - In 2013, they processed 31.1% of entire loans in this dataset and over 35000 loans were listed.
- There seem to have been an overall increase per year in the number of loans processed although, there was a significant dip in 2014

In [ ]:

borrowers_profile

We will consider some selected features from this table that are percieved to have strong implication on BorrowersAPR. They are chosen based on their importance in the loan application process itself.

Some of these features are as followers;

- BorrowerState
- Occupation
- EmploymentStatus
- IsBorrowerHomeowner
- CreditScore
- DebtToIncomeRatio
- StatedMonthlyIncome
- CurrentCreditLines and OpenCurrentCreditLines
- Return

Borrower's Categorical Classification

We will consider these set of 4 qualities, `Occupation` , `EmploymentStatus` , `IsBorrowerHomeowner` and `BorrowerState` . The first three are critical considration in the application process while the State charcteristics is also important because it might reveal the geographic distribution of borrowers across the US states.

Four Functions

We will visually explore these set of categorical plot using these set of four functions

The four functions are:

- `count_plotterv` will create a vertical plot which will be annotated with `annotate_vertical`.
- `count_plotterh` will create a horizontal plot which will be annotated with `annotate_horizontal`.

In [164…

```
def count_plotterv(df,var,title):

    '''
    Plots and title a vertical countplot

    args:
    df (DataFrame): The dataframe containing the qualitative
                    variable of interest
    var (string): The name of the column of the qualtative
                  feature of interest
    title ('string'); The title of the plot


    return:
    g(seaborn plot object): returns the seaborn plot object for further
                            customization of the plot
    '''




    # value count and value count index for order
    value_count = df[var].value_counts()
    value_count_index = value_count.index
    sum_value = value_count.sum()
    # plotting the count of the variable levels
    plt.figure(figsize = (10,5))
    color = sb.color_palette()[0]
    g = sb.countplot(data = df, x = var,
                     color = color, order =value_count_index )
    plt.title(title)
    return(g)
```

In [165…

```
def count_plotterh(df,var,title):

    '''
    Plots and title a horizontal countplot

    args:
    df (DataFrame): The dataframe containing the qualitative
                    variable of interest
    var (string): The name of the column of the qualtative
                  feature of interest
    title ('string'); The title of the plot


    return:
    g(seaborn plot object): returns the seaborn plot object for further
                            customization of the plot
    '''




    # value count and value count index for order
```

```
    value_count = df[var].value_counts()
    value_count_index = value_count.index
    sum_value = value_count.sum()
    # plotting the count of the variable levels
    plt.figure(figsize = (10,20))
    color = sb.color_palette()[0]
    g = sb.countplot(data = df, y = var,
                 color = color, order =value_count_index )
    plt.title(title)
    return(g)
```

In [166…

```
# annotate the bars

def annotate_vertical(g):
    '''
    annotate the plot with the percentage value of each bar.

    args:
    g(seaborn plot object):The plot object returned by the plotter function
    '''
    for p in g.patches:

        g.annotate('{:0.1f}%'.format(100*p.get_height()/sum_value),
                       (p.get_x()+p.get_width()/2, p.get_height()+50),
                       horizontalalignment = 'center')
```

In [167…

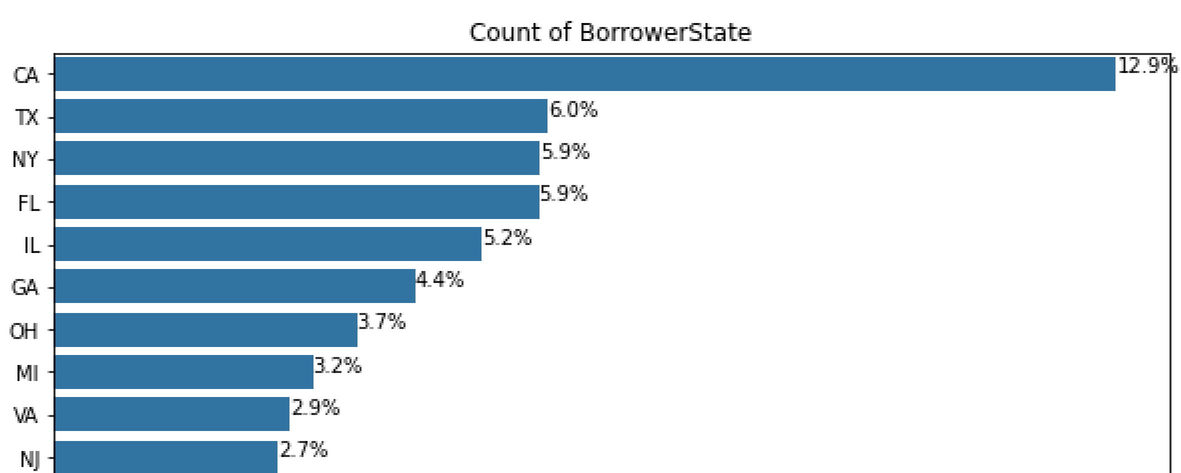
```
# annotate the bars
def annotate_horizontal(g):
    '''
    annotate the plot with the percentage valu of each bar.

    args:
    g(seaborn plot object):The plot object returned by the plotter function
    '''
    for p in g.patches:
        g.annotate('{:.1f}%'.format(100*p.get_width()/total),
                ((p.get_x()+p.get_width()), p.get_y()),horizontalalignment = 'left',
                 verticalalignment = 'top')
```

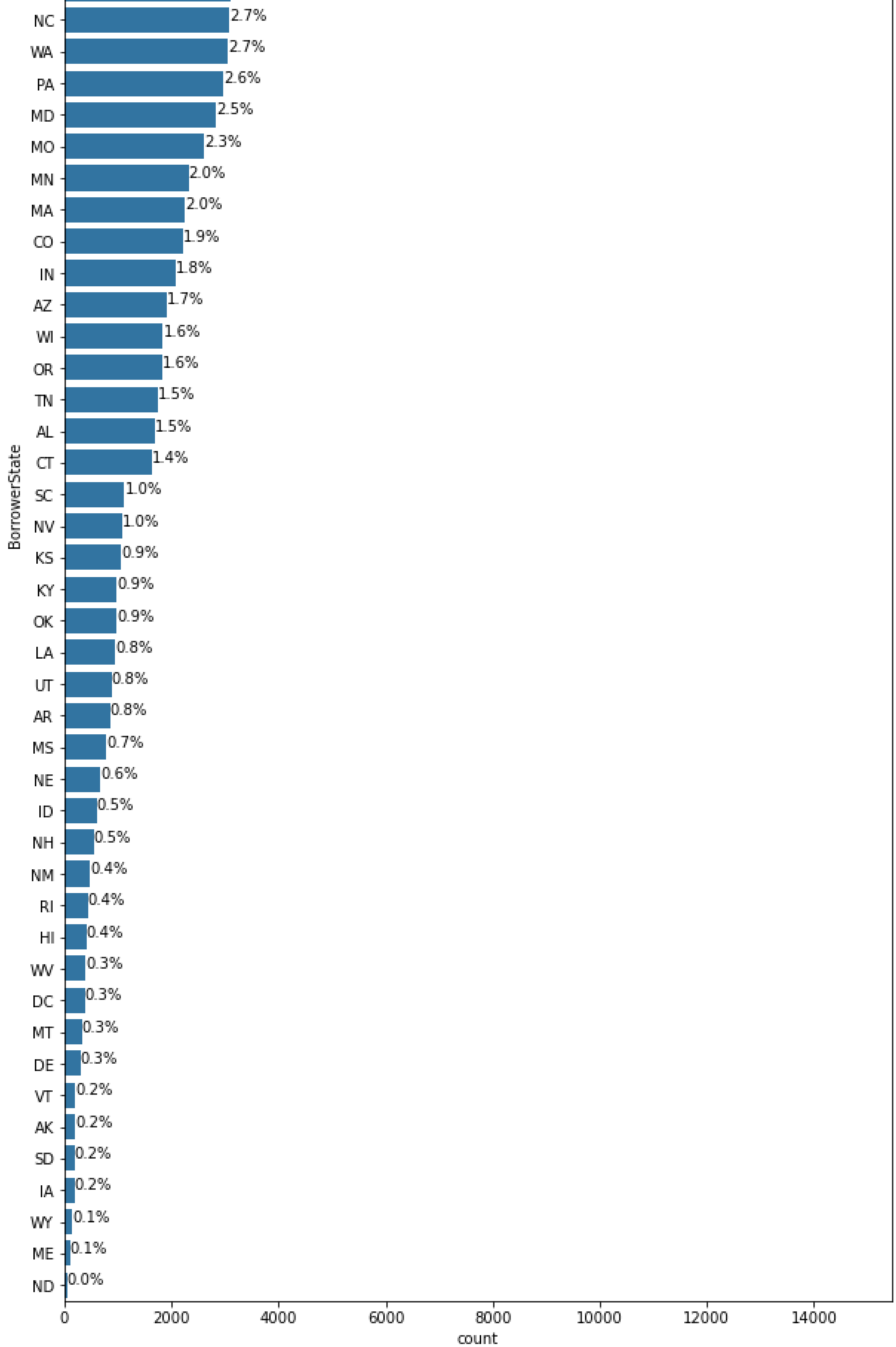
Visually Exploring the BorrowerState column

Which state has the largest count in the dataset ?

In [168…

```
# plot the  count of BorrowerState.
state = count_plotterh(borrowers_profile, 'BorrowerState', 'Count of BorrowerState')
annotate_horizontal(state)
```





Observation

- California state is the state with the highest count in the BorrowerState column by a wide margin.
- North Dakorta is the state with least count.

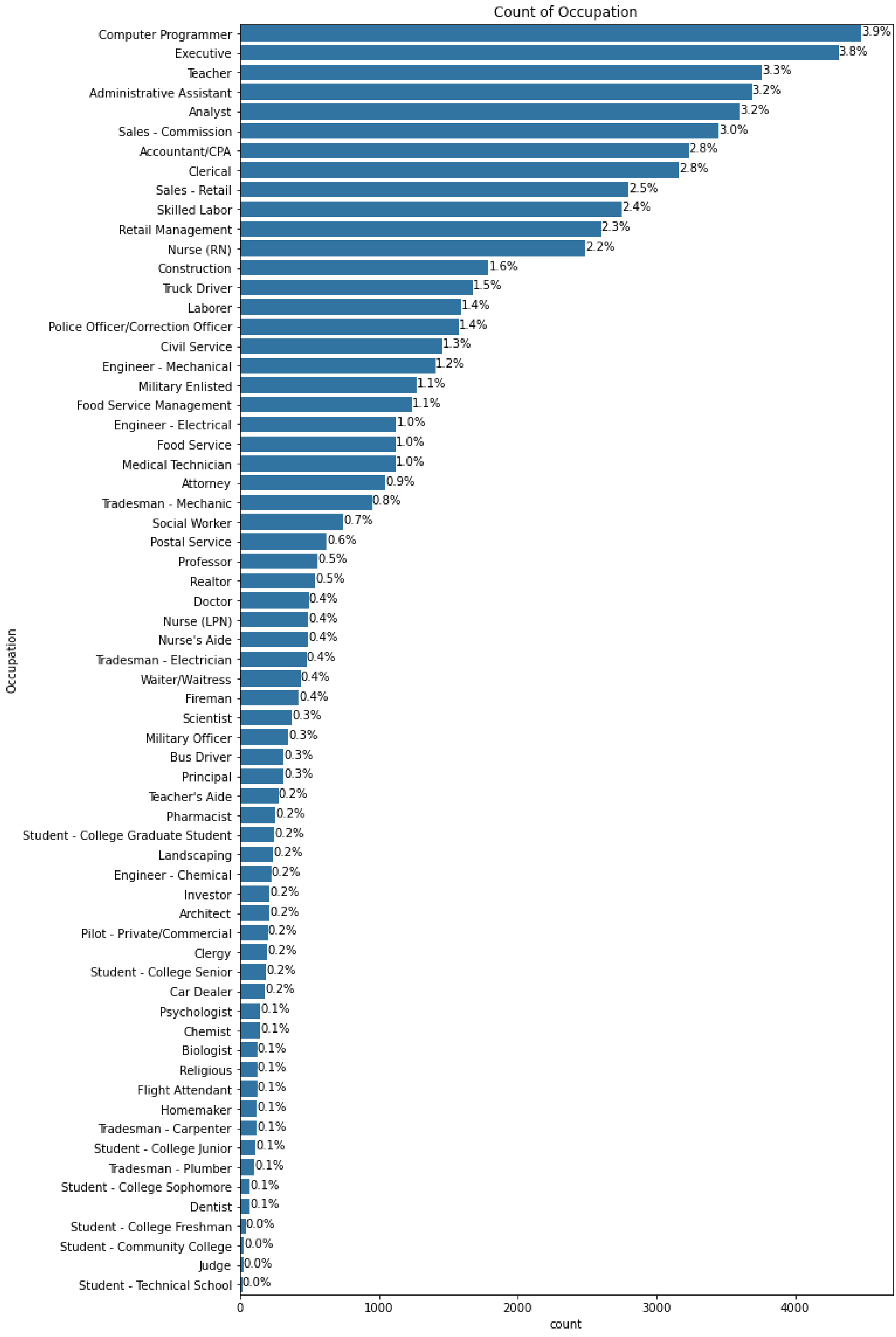
Visually Exploring the Occupation column

Which group of professional use the prosper funding service the most ?

There are two outlier levels in the Occupation field. Other and Professional. They will be filtered out as they do not represent any particular occupation.

In [169…

```
# filtering out observations with Occupation value, 'Other' and 'Professional'
occupation_list = ['Other', 'Professional']
borrowers_profile_subset = borrowers_profile[~borrowers_profile['Occupation'].isin(occup

# plot the  count of Occupation.
state = count_plotterh(borrowers_profile_subset, 'Occupation', 'Count of Occupation')
annotate_horizontal(state)
```


Count of Occupation
Computer Programmer 3.9%
Executive 3.8%
Teacher 3.3%
Administrative Assistant 3.2%
Analyst 3.2%
Sales - Commission 3.0%
Accountant/CPA 2.8%
Clerical 2.8%
Sales - Retail 2.5%
Skilled Labor 2.4%
Retail Management 2.3%
Nurse (RN) 2.2%
Construction 1.6%
Truck Driver 1.5%
Laborer 1.4%
Police Officer/Correction Officer 1.4%
Civil Service 1.3%
Engineer - Mechanical 1.2%
Military Enlisted 1.1%
Food Service Management 1.1%
Engineer - Electrical 1.0%
Food Service 1.0%
Medical Technician 1.0%
Attorney 0.9%
Tradesman - Mechanic 0.8%
Social Worker 0.7%
Postal Service 0.6%
Professor 0.5%
Realtor 0.5%
Doctor 0.4%
Nurse (LPN) 0.4%
Nurse's Aide 0.4%
Tradesman - Electrician 0.4%
Waiter/Waitress 0.4%
Fireman 0.4%
Scientist 0.3%
Military Officer 0.3%
Bus Driver 0.3%
Principal 0.3%
Teacher's Aide 0.2%
Pharmacist 0.2%
Student - College Graduate Student 0.2%
Landscaping 0.2%
Engineer - Chemical 0.2%
Investor 0.2%
Architect 0.2%
Pilot - Private/Commercial 0.2%
Clergy 0.2%
Student - College Senior 0.2%
Car Dealer 0.2%
Psychologist 0.1%
Chemist 0.1%
Biologist 0.1%
Religious 0.1%
Flight Attendant 0.1%
Homemaker 0.1%
Tradesman - Carpenter 0.1%
Student - College Junior 0.1%
Tradesman - Plumber 0.1%
Student - College Sophomore 0.1%
Dentist 0.1%
Student - College Freshman 0.0%
Student - Community College 0.0%
Judge 0.0%
Student - Technical School 0.0%
Occupation
0
1000
2000
3000
4000
count

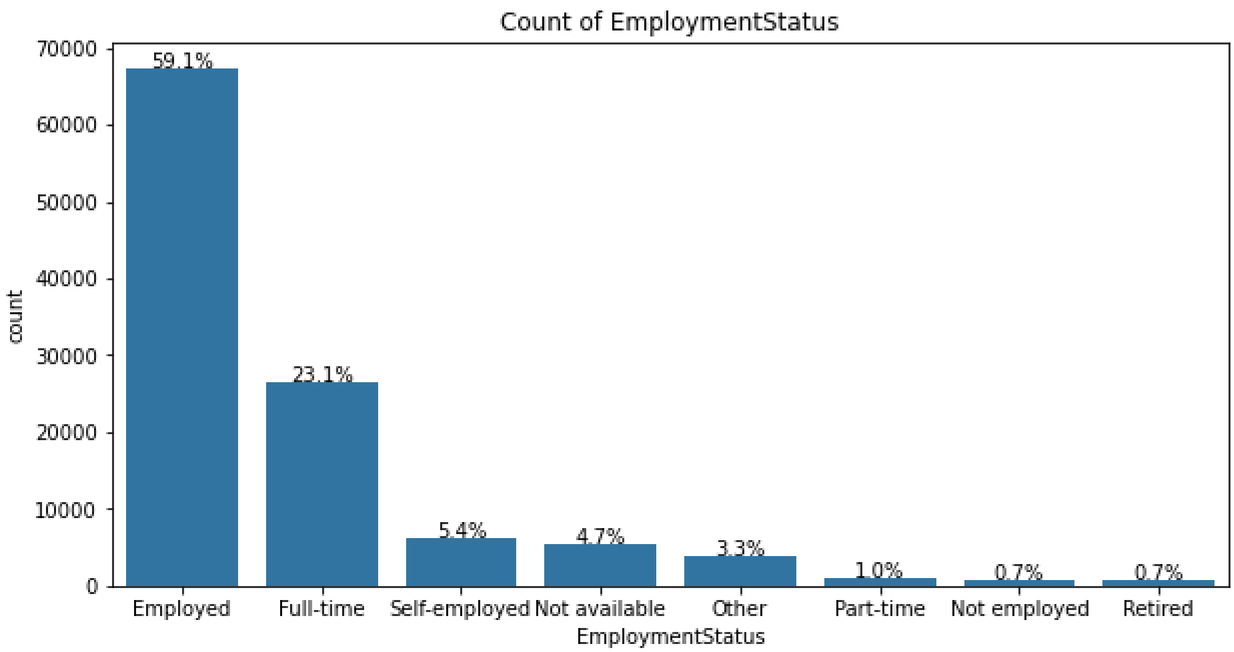
Observation

- Computer Programmers are the most people using the Prosper loan service. Followed by executives. This might explain why California state is the state with highest count in the BorrowerState column.

Visually Exploring the EmploymentStatus

In [170…

```
# plot the  count of EmploymentStatus.
state = count_plotterv(borrowers_profile, 'EmploymentStatus', 'Count of EmploymentStatus
annotate_vertical(state)
```



Observation

People who are employed and actively in service tend to get more access to loans as compared to those who are unemployed or retired.

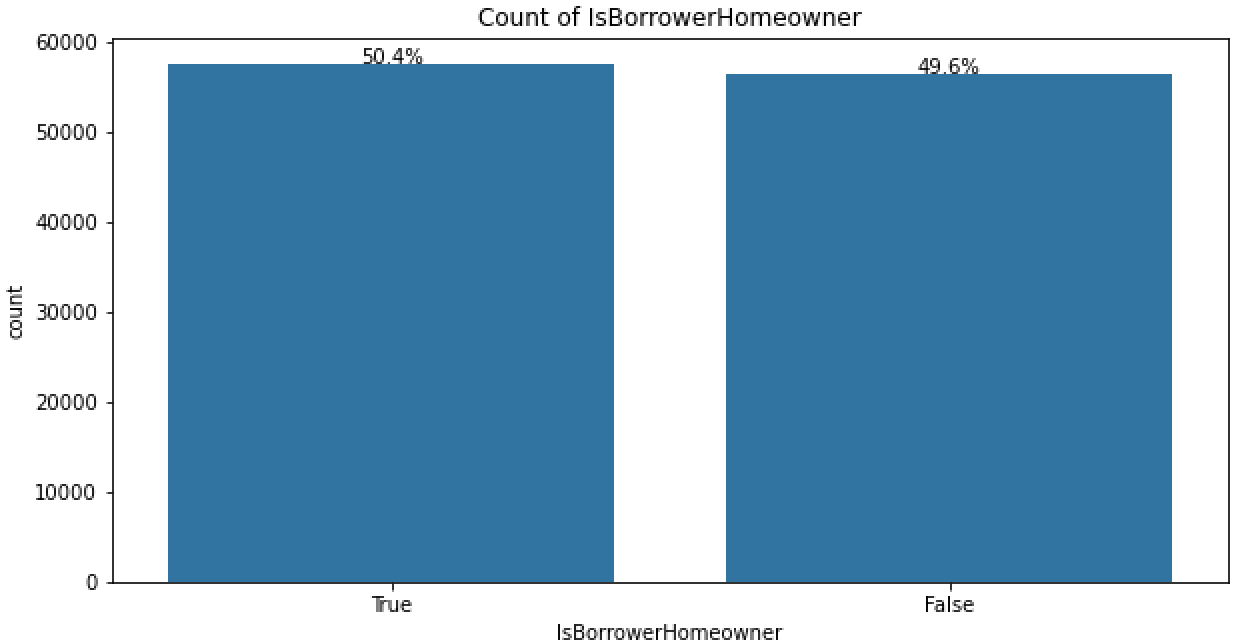
Also, those employed in formal establishment gets more access to loan as compared to those who are self employed

Visually Exploring the IsBorrowerHomeowner

What is the distribution of homeowners ?

In [171…

```
# plot the  count of IsBorrowerHomeowner.
state = count_plotterv(borrowers_profile, 'IsBorrowerHomeowner', 'Count of IsBorrowerHom
annotate_vertical(state)
```



Observation

There are just as much homeowners as those who don't own a home in the dataset

Visually exploring the CreditScore.

The credit score value typically range between 300 and 850 and represents the credit risk of an individual and how likely is an individual to pay bills on time.

- It was observed that there were 133 rows with CreditscoreRangeUpper values at 19 and CreditscoreRangeLower values at 0 These observations will be filtered out, since credit score value should naturally range between 300 and 850.

In [172…
```
borrowers_profile[['CreditScoreRangeLower','CreditScoreRangeUpper']][(borrowers_profile[
```

Out[172]:

	CreditScoreRangeLower	CreditScoreRangeUpper
count	133.0	133.0
mean	0.0	19.0
std	0.0	0.0
min	0.0	19.0
25%	0.0	19.0
50%	0.0	19.0
75%	0.0	19.0
max	0.0	19.0

In [173…
```
# filtering out credit score values less than 300
borrowers_profile= borrowers_profile[~(borrowers_profile['CreditScoreRangeUpper'] < 300)

# summary statistics of Differences between the CreditScoreRangUpper and CreditScoreRang
(borrowers_profile['CreditScoreRangeUpper']-borrowers_profile['CreditScoreRangeLower']).
```

Out[173]:
```
count    113213.0
mean         19.0
```

```
std            0.0
min           19.0
25%           19.0
50%           19.0
75%           19.0
max           19.0
dtype: float64
```

In [174…

```python
# Summary statistics of the CreditScoreRangeLower
borrowers_profile['CreditScoreRangeLower'].describe()
```

Out[174]:

```
count    113213.000000
mean        686.373120
std          62.201999
min         360.000000
25%         660.000000
50%         680.000000
75%         720.000000
max         880.000000
Name: CreditScoreRangeLower, dtype: float64
```

In [175…

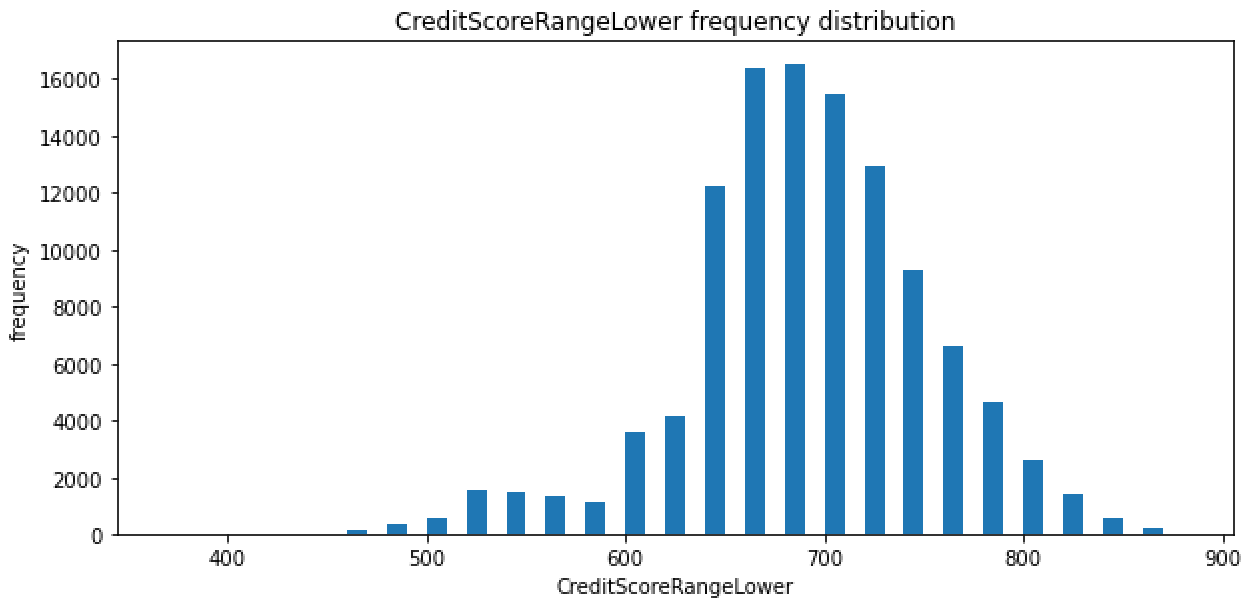
```python
# Summary statistics of the CreditScoreRangeUpper
borrowers_profile['CreditScoreRangeUpper'].describe()
```

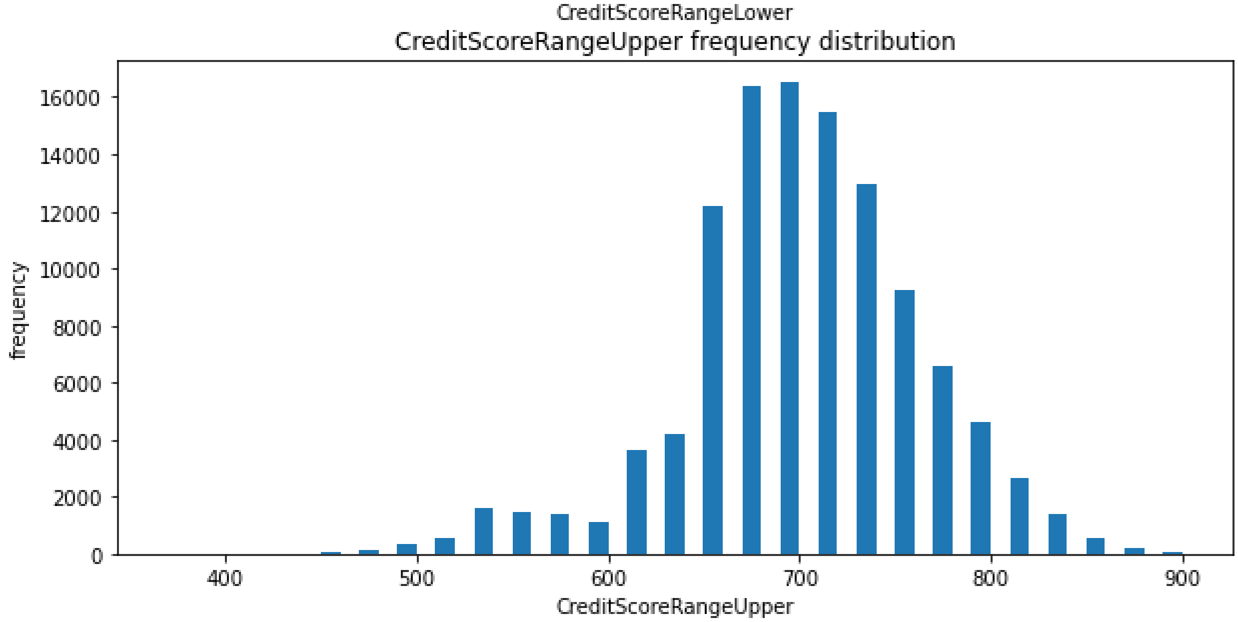
Out[175]:

```
count    113213.000000
mean        705.373120
std          62.201999
min         379.000000
25%         679.000000
50%         699.000000
75%         739.000000
max         899.000000
Name: CreditScoreRangeUpper, dtype: float64
```

In [176…

```python
# Plotting the distribution of CreditScoreRangeUpper and CreditScoreRangeLower
fig, ax = plt.subplots(nrows = 2, figsize = (10,10))

features = ['CreditScoreRangeLower', 'CreditScoreRangeUpper']

for i in range(len(features)):
    var = features[i]
    bins = np.arange(370, borrowers_profile[var].max()+10, 10)
    ax[i].hist(data = borrowers_profile, x = var, bins = bins)
    ax[i].set_xlabel('{}'.format(var))
    ax[i].set_ylabel('frequency')
    ax[i].set_title('{} frequency distribution'.format(var))
```





Observation

- It can be observed that the modal credit score value ranges between 660 and 720.
- It can also be observed that there is a slightly longer tale to the left of the modal class. However, more of the population in the dataset are on the right side of the distributon.

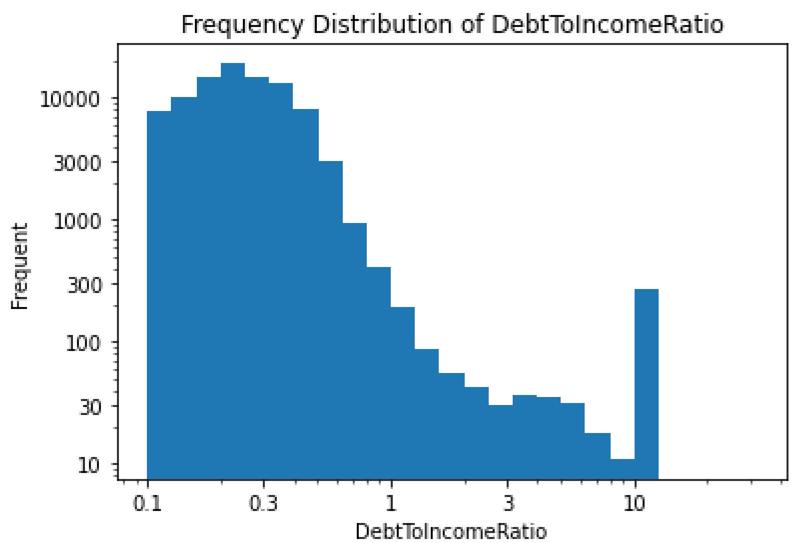
DebtToIncomeRatio

what is the destribution of the DebtToIncomeRatio ?

In [177…
```
borrowers_profile.DebtToIncomeRatio.describe()
```

Out[177]:
```
count    105311.000000
mean          0.276075
std           0.551883
min           0.000000
25%           0.140000
50%           0.220000
75%           0.320000
max          10.010000
Name: DebtToIncomeRatio, dtype: float64
```

In [178…
```
#Plotting the DebtToIncomeRatioof DTI_normal
bin = 10**np.arange(-1, 1.5+0.1, 0.1)
plt.hist(data = borrowers_profile, x ='DebtToIncomeRatio', bins = bin)
plt.xlabel('DebtToIncomeRatio')
plt.ylabel('Frequent')
plt.title('Frequency Distribution of DebtToIncomeRatio')
xticks = [0.1, 0.3, 1, 3, 10]
yticks = [10, 30, 100, 300,1000,3000,10000]
plt.xscale('log')
plt.yscale('log')
plt.xticks(xticks, xticks)
plt.yticks(yticks, yticks);
```



Observations

- According to the variable defination document, the DebtToIncomeRatio has been been capped at 10.01. i.e. 1001% we can oberve this by the sharp spike at that point.
- Also, the visualisation has been truncated at the lower end, on the left.
- The distribution is skewed to the right, with the modal cass around 0.3 1.e 30% Debt to income ratio.

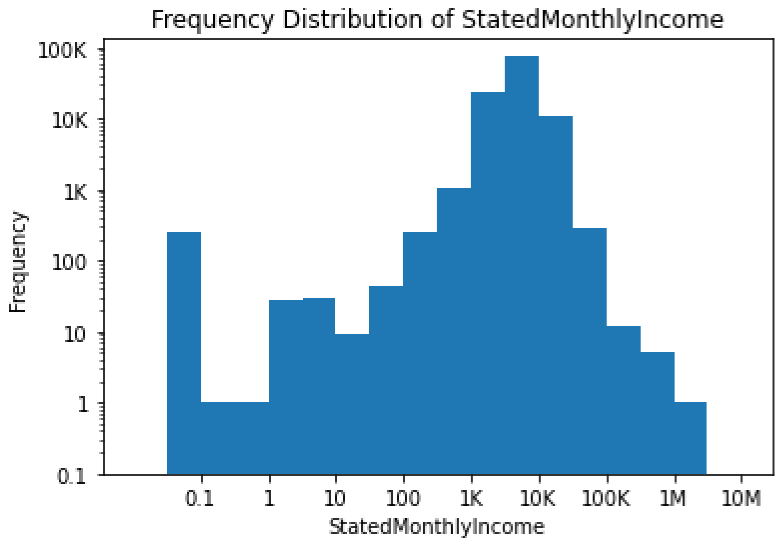
StatedMonthlyIncome

In [179…
```
borrowers_profile['StatedMonthlyIncome'].describe()
```

Out[179]:
```
count    1.138040e+05
mean     5.611754e+03
std      7.481310e+03
min      0.000000e+00
25%      3.208333e+03
50%      4.666667e+03
75%      6.833333e+03
max      1.750003e+06
Name: StatedMonthlyIncome, dtype: float64
```

In [180…
```
#Plotting the StatedMonthlyIncome DTI_normal
bin = 10**np.arange(-2, 7+0.5, 0.5)
plt.hist(data = borrowers_profile, x ='StatedMonthlyIncome', bins = bin)
plt.xlabel('StatedMonthlyIncome')
plt.ylabel('Frequency')
plt.title('Frequency Distribution of StatedMonthlyIncome')
xticks = [0.1,1,10,100,1000,10000, 100000,1000000,10000000]
x_labels = ['0.1', '1', '10', '100', '1K', '10K', '100K', '1M', '10M']
```

```
yticks = [0.1,1,10,100,1000,10000, 100000]
y_labels = ['0.1', '1', '10', '100', '1K', '10K', '100K']
plt.xscale('log')
plt.yscale('log');
plt.xticks(xticks, x_labels)
plt.yticks(yticks,y_labels);
```



Observation

- The distribution is slightly skewed to the left
- with a modal class around 3000 dollars stated income.

CurrentCreditLines and OpenCreditLines

CurrentCreditLines

In [181…

```
borrowers_profile[['CurrentCreditLines', 'OpenCreditLines']].describe()
```

Out[181]:

	CurrentCreditLines	OpenCreditLines
count	106333.000000	106333.000000
mean	10.317192	9.260164
std	5.457866	5.022644
min	0.000000	0.000000
25%	7.000000	6.000000
50%	10.000000	9.000000
75%	13.000000	12.000000
max	59.000000	54.000000

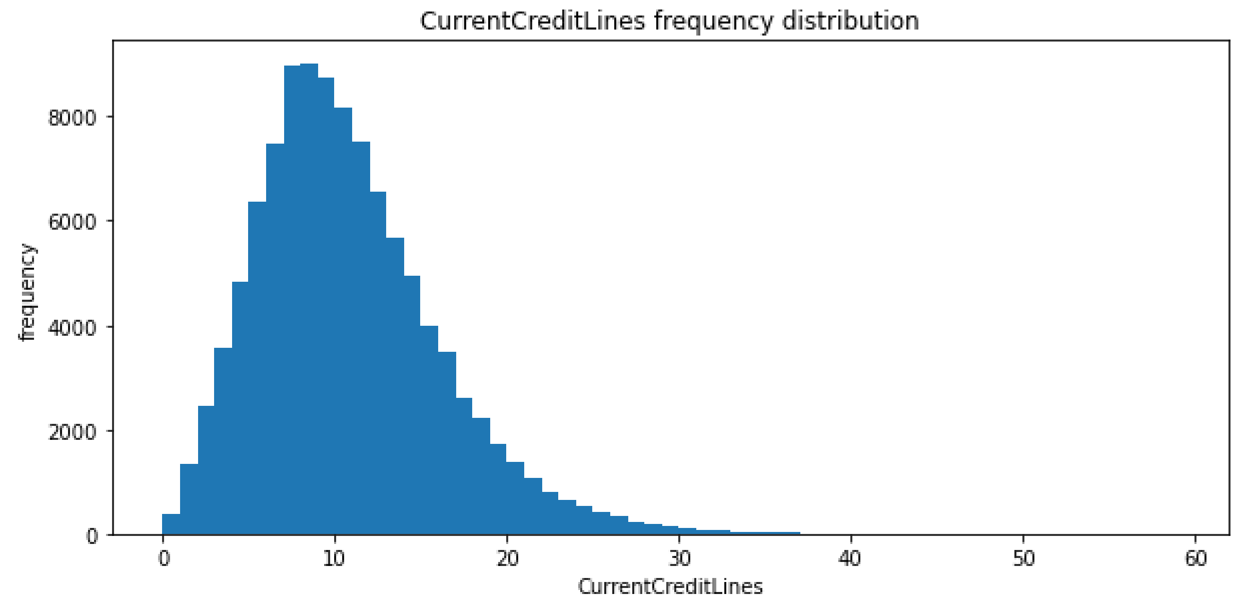
In [182…

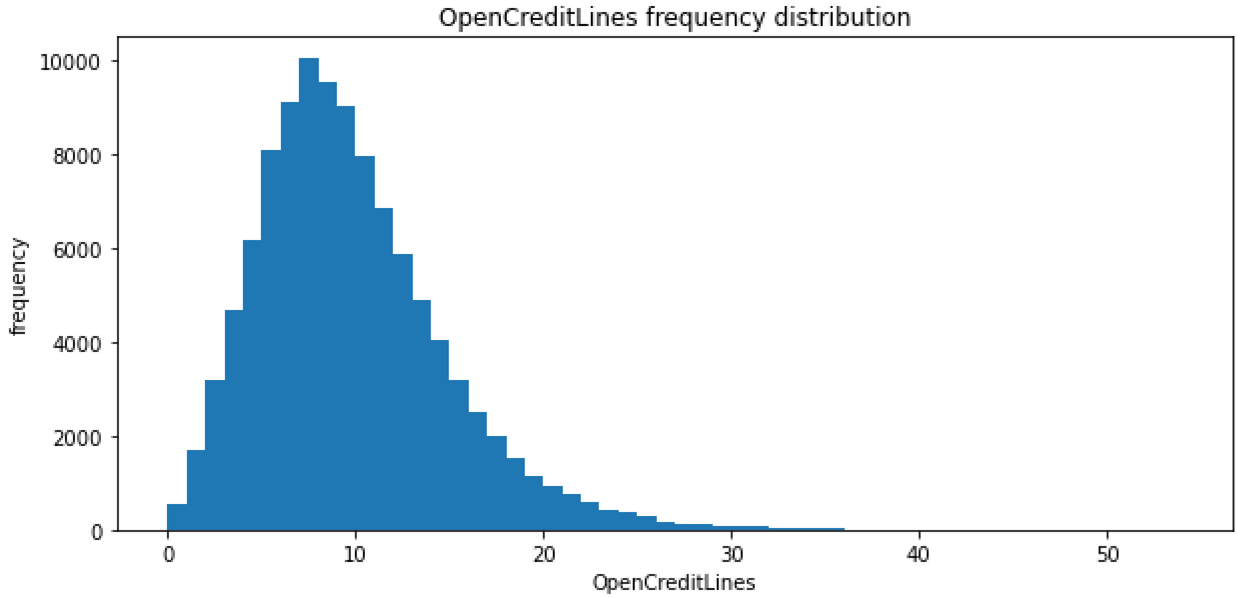
```
def hist_plotter(df, feature_list, nrows, bin_lower, binsize):
    '''
     Plots the distribution of a list of features

     Args:
     df(DataFrame): The dataFrame containing the feature of interest.
     feature_list(list): A list of feature whose distribution is to be plotted
     nrows(int): The number of rows of the plot based on the number of variables in the
     bin_lower(int or float): The lower boundary of the bin
```

```
    binsize(int or float): The size of each bin in the plot


    returns:
    ret(list): a list of plot object


    '''
    features = feature_list

    if nrows == 1:
        bins = np.arange(bin_lower, df[features[0]].max()+binsize, binsize)
        g = plt.hist(data = df, x = features[0], bins = bins)
        plt.xlabel('{}'.format(features[0]))
        plt.ylabel('frequency')
        plt.title('{} frequency distribution'.format(features[0]))

    else:

        fig, ax = plt.subplots(nrows = nrows, figsize = (10,10))




        for i in range(len(features)):
            ret = []
            var = features[i]
            bins = np.arange(bin_lower, df[var].max()+binsize, binsize)
            g = ax[i].hist(data = df, x = var, bins = bins)
            ax[i].set_xlabel('{}'.format(var))
            ax[i].set_ylabel('frequency')
            ax[i].set_title('{} frequency distribution'.format(var))


    return(g)
```

In [183…

```
# Plotting the CurrentCreditLines and OpenCreditLines.
feature_list = ['CurrentCreditLines', 'OpenCreditLines']
hist_plotter(borrowers_profile, feature_list, 2, 0, 1);
```





Observations

- Modal class is between 10 and 11 with the distribution slightly skewed to the right.
- The OpenCreditLines distribution is exactly the same as the CurrentCreditLines but it is shifted to the left.

Loan

The loan dataframe contains information about the loan and some historical information about the borrower's loan activity on the prosper platform.

Some of the features of interest here can be categorized into three:

- The borrowers loan history with Prosper e.g. TotalProperLoans, TotalProsperPaymentsBilled, OnTimeProsperPayments, ProsperPaymentsLessThanOneMonthLate, ProsperPaymentsOneMonthPlusLate,ScorexChangeAtTimeOfListing
- The current loan status e.g. LoanCurrentDaysDelinquent, LoanOriginalAmount, MonthlyLoanPayment
- The current loan charges e.g. LP_CustomerPayments, LP_ServiceFees, LP_CollectionFees, LP_GrossPrincipalLoss

- Return

The borrowers loan history with Prosper

TotalProsperLoans, TotalProsperPaymentsBilled and OnTimeProsperPayments

In [184...
```
loan[['TotalProsperLoans', 'TotalProsperPaymentsBilled', 'OnTimeProsperPayments']].descr
```

Out[184]:

	TotalProsperLoans	TotalProsperPaymentsBilled	OnTimeProsperPayments
count	22085.000000	22085.000000	22085.000000
mean	1.421100	22.934345	22.271949
std	0.764042	19.249584	18.830425
min	0.000000	0.000000	0.000000
25%	1.000000	9.000000	9.000000
50%	1.000000	16.000000	15.000000
75%	2.000000	33.000000	32.000000
max	8.000000	141.000000	141.000000

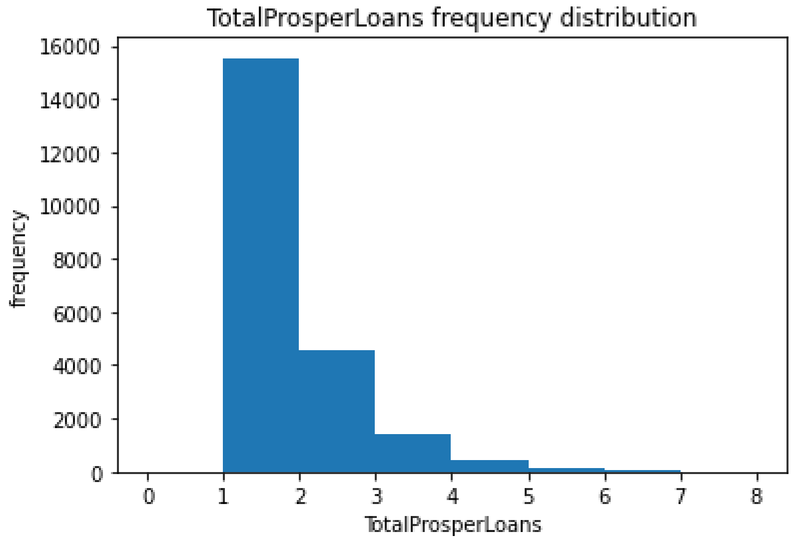
TotalProsperLoans

In [185...
```
loan['TotalProsperLoans'].describe()
```

Out[185]:
```
count    22085.000000
mean         1.421100
std          0.764042
min          0.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          8.000000
Name: TotalProsperLoans, dtype: float64
```

In [186...
```
feature_list = ['TotalProsperLoans']
hist_plotter(loan, feature_list, 1, 0, 1)
```

Out[186]:
```
(array([1.0000e+00, 1.5538e+04, 4.5400e+03, 1.4470e+03, 4.1700e+02,
        1.0400e+02, 2.9000e+01, 9.0000e+00]),
 array([0., 1., 2., 3., 4., 5., 6., 7., 8.]),
 <BarContainer object of 8 artists>)
```



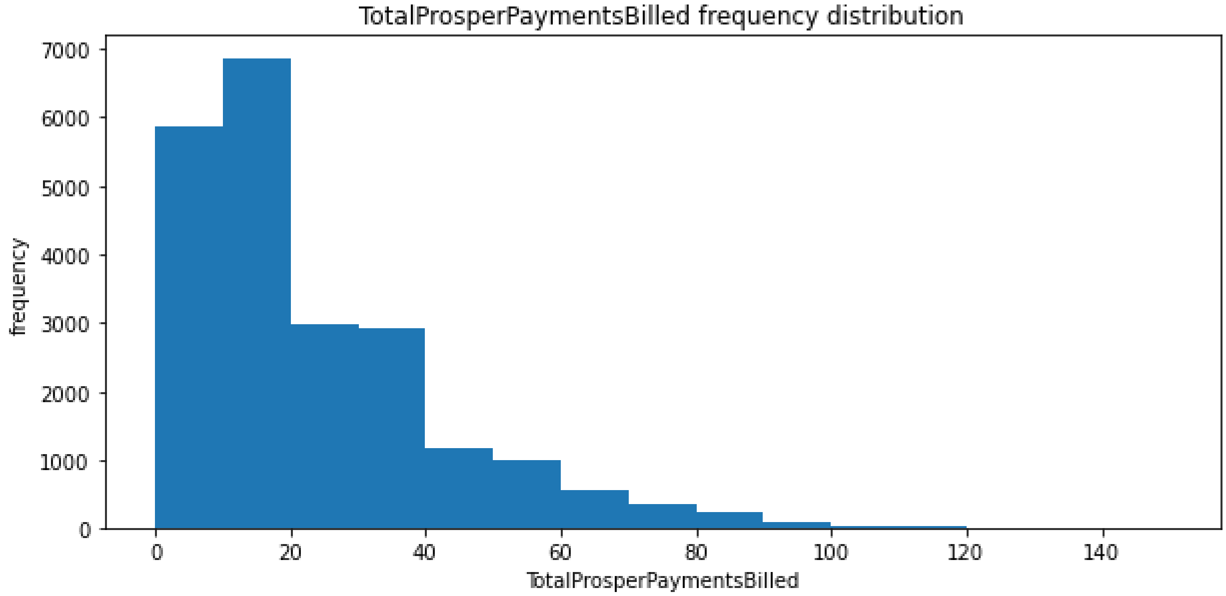
Observations

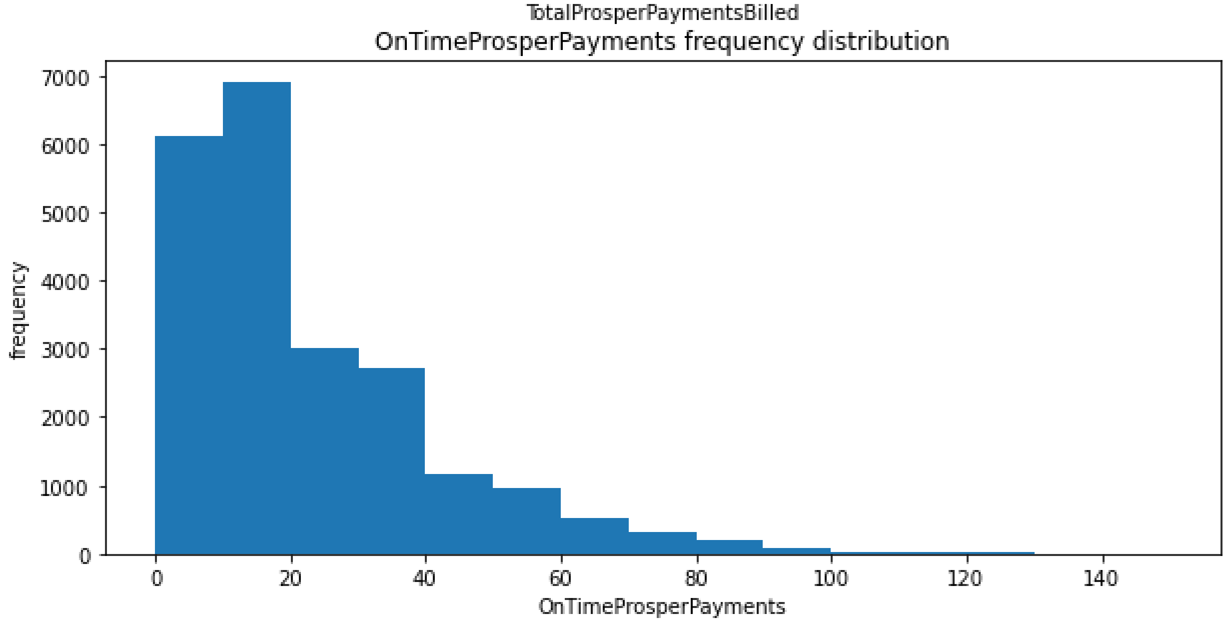
- The modal class is between 1 and 2. Indicating that, most people have 1 to 2 loans on prosper partform before this application
- The distribution is skewed to the right indicating that there are less number of borrowers with higher number of prior loans.

In [187…
```
# plotting TotalProsperPaymentsBilled, OnTimeProsperPayments
```

TotalProsperPaymentsBilled and OnTimeProsperPayments

In [188…
```
feature_list = ['TotalProsperPaymentsBilled', 'OnTimeProsperPayments']
hist_plotter(loan, feature_list, 2, 0, 10);
```





Observations

- The distribution is skewed to the left with the modal class between 10 and 20 i.e most borrowers have made upto 10 to 20 on time payments.

ProsperPaymentsLessThanOneMonthLate, ProsperPaymentsOneMonthPlusLate

In [189…

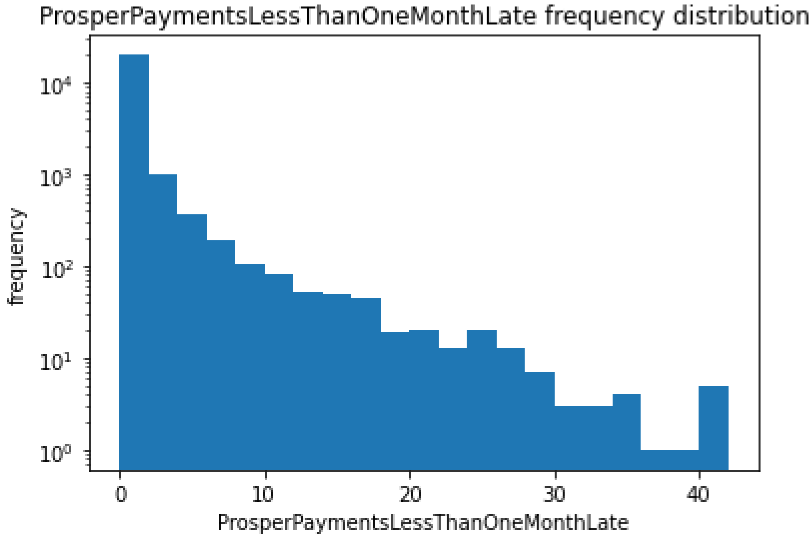
```
loan['ProsperPaymentsLessThanOneMonthLate'].describe()
```

Out[189]:

```
count    22085.000000
mean         0.613629
std          2.446827
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max         42.000000
Name: ProsperPaymentsLessThanOneMonthLate, dtype: float64
```

In [190…

```
feature_list = ['ProsperPaymentsLessThanOneMonthLate']
hist_plotter(loan, feature_list, 1, 0, 2)
plt.yscale('log')
```



Observation

- The modal class is 0 to 2 which indicates that most people make their payments on time.
- The distribution is skewed to right. also showing that the number of people who make their payments late reduces as the lateness duration increases.

In [191…

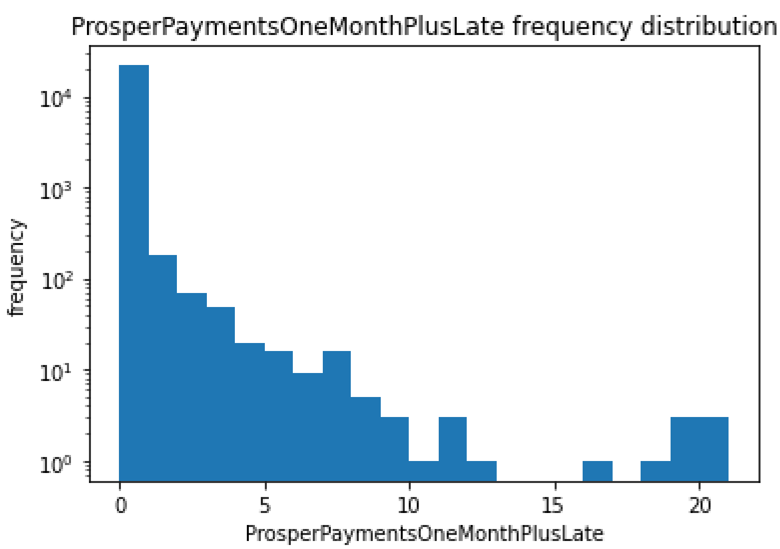
```
loan['ProsperPaymentsOneMonthPlusLate'].describe()
```

Out[191]:

```
count    22085.000000
mean         0.048540
std          0.556285
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max         21.000000
Name: ProsperPaymentsOneMonthPlusLate, dtype: float64
```

In [192…

```
feature_list = ['ProsperPaymentsOneMonthPlusLate']
hist_plotter(loan, feature_list, 1, 0, 1)
plt.yscale('log')
```



Observation

- The modal class is 0 to 2 which indicates that most people make their payments on time.

- The distribution is skewed to right. also showing that the number of people who make their payments late reduces as the lateness duration increases.

ScorexChangeAtTimeOfListing

In [193…

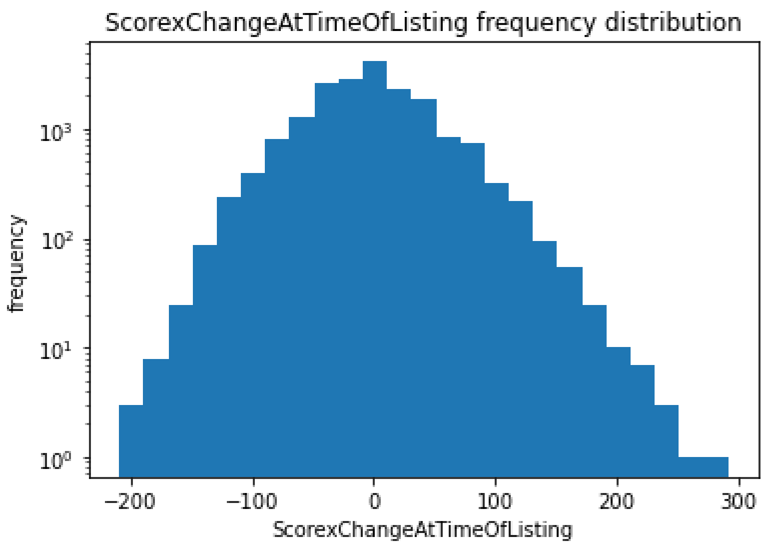
```
loan['ScorexChangeAtTimeOfListing'].describe()
```

Out[193]:

```
count     18928.000000
mean         -3.223214
std          50.063567
min        -209.000000
25%         -35.000000
50%          -3.000000
75%          25.000000
max         286.000000
Name: ScorexChangeAtTimeOfListing, dtype: float64
```

In [194…

```
feature_list = ['ScorexChangeAtTimeOfListing']
hist_plotter(loan, feature_list, 1, -209, 20)
plt.yscale('log')
```



Observations

-This almost approximates a normal distribution with the mean at zero.

The current loan status

In [195…

```
loan[['LoanCurrentDaysDelinquent', 'LoanOriginalAmount', 'MonthlyLoanPayment']].describe
```

Out[195]:

	LoanCurrentDaysDelinquent	LoanOriginalAmount	MonthlyLoanPayment
count	113937.000000	113937.00000	113937.000000
mean	152.816539	8337.01385	272.475783
std	466.320254	6245.80058	192.697812
min	0.000000	1000.00000	0.000000
25%	0.000000	4000.00000	131.620000
50%	0.000000	6500.00000	217.740000
75%	0.000000	12000.00000	371.580000

max	2704.000000	35000.00000	2251.510000

LoanOriginalAmount

In [196…

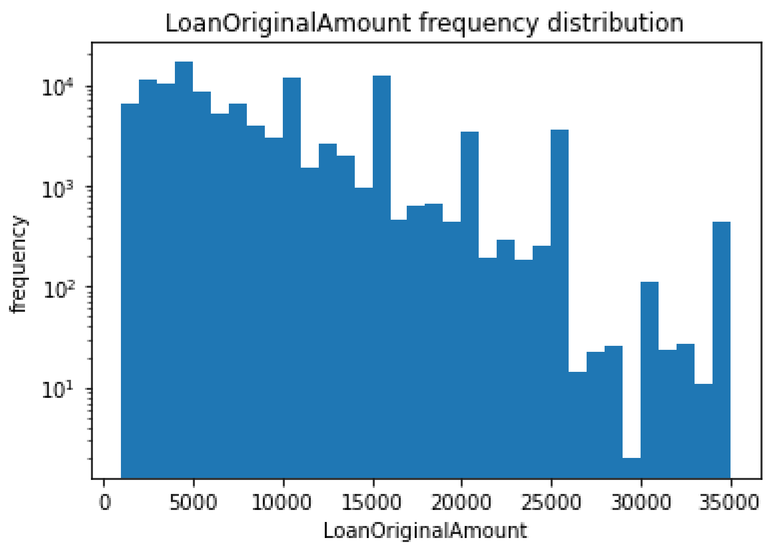
```
loan['LoanOriginalAmount'].describe()
```

Out[196]:

```
count    113937.00000
mean       8337.01385
std        6245.80058
min        1000.00000
25%        4000.00000
50%        6500.00000
75%       12000.00000
max       35000.00000
Name: LoanOriginalAmount, dtype: float64
```

In [197…

```
feature_list = ['LoanOriginalAmount']
hist_plotter(loan, feature_list, 1, 1000, 1000);
plt.yscale('log')
```



Observations

- The modal class is between 5000 to 6000 dollars.
- The distribution is skewed to the right. and the overall trend is that lesser amount of people take up larger loans.
- There is a spike at every 5000 dollar value. It seems more people obtain loans in rounded figure values.

MonthlyLoanPayment

In [198…

```
loan[['MonthlyLoanPayment']].describe()
```

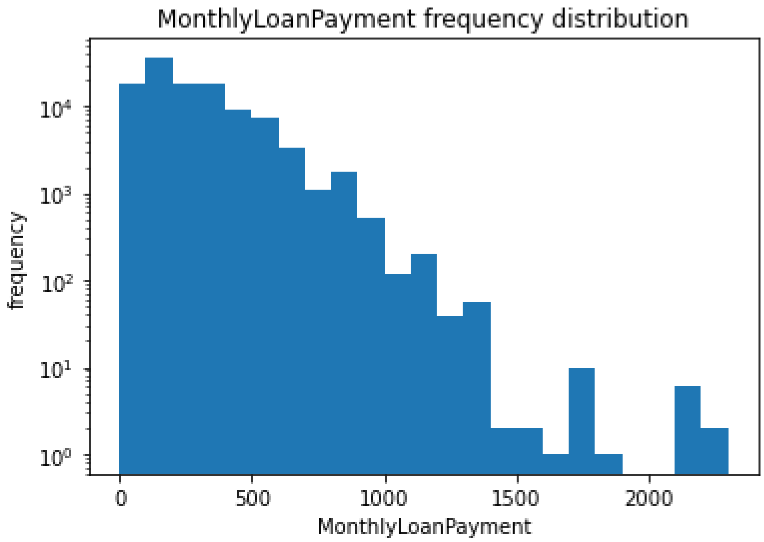
Out[198]:

	MonthlyLoanPayment
count	113937.000000
mean	272.475783
std	192.697812
min	0.000000
25%	131.620000
50%	217.740000

75%	371.580000
max	2251.510000

In [199…

```
feature_list = ['MonthlyLoanPayment']
hist_plotter(loan, feature_list, 1, 0, 100)
plt.yscale('log')
```



Observations

- It can be observed that most MonthlyLoanPayment are in 100s of dollars. with modal class being the class between 100 and 200 dollars.
- The distribution is skewed to the right. indicating that lesser amount of the population make high monthly loan payment.

In [200…

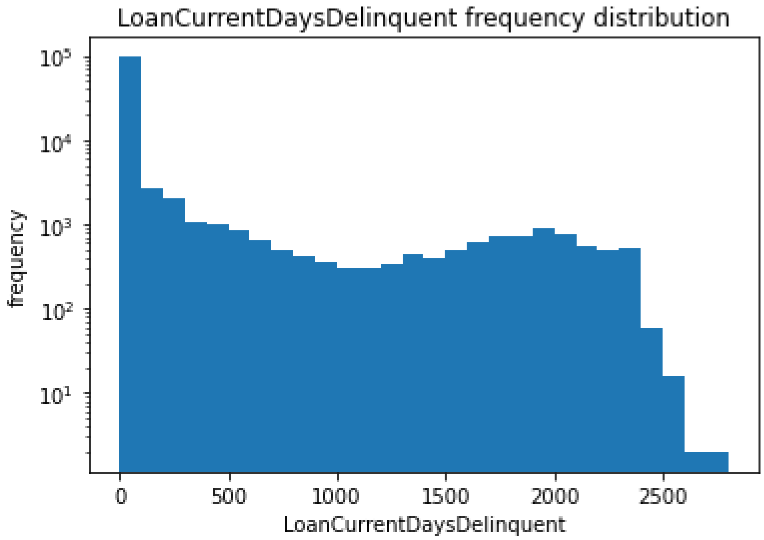
```
loan['LoanCurrentDaysDelinquent'].describe()
```

Out[200]:

```
count    113937.000000
mean        152.816539
std         466.320254
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max        2704.000000
Name: LoanCurrentDaysDelinquent, dtype: float64
```

In [201…

```
feature_list = ['LoanCurrentDaysDelinquent']
hist_plotter(loan, feature_list, 1, 0, 100)
plt.yscale('log')
```



Observation

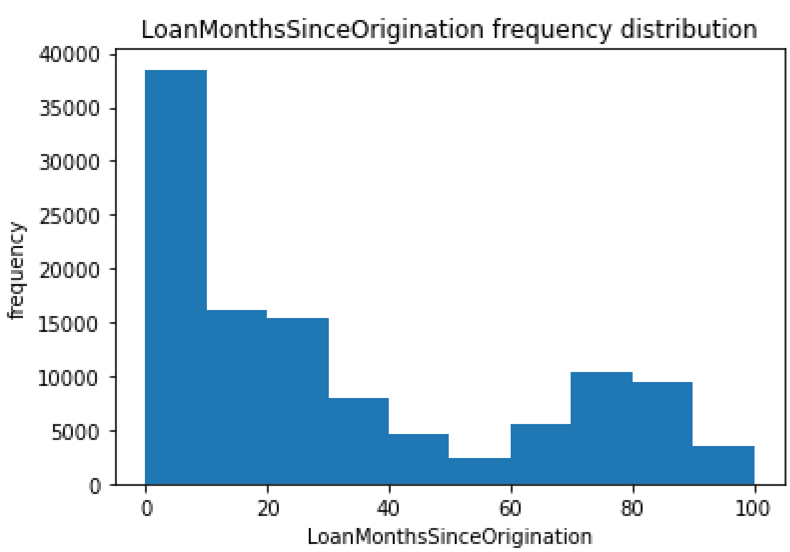
- The majority of the population make thier payment promptly, and are never delinquent.
- The distribution is rather flat, with a deep in the centre around 1000days.

LoanMonthsSinceOrigination

In [202…
```
loan['LoanMonthsSinceOrigination'].describe()
```

Out[202]:
```
count    113937.000000
mean         31.896882
std          29.974184
min           0.000000
25%           6.000000
50%          21.000000
75%          65.000000
max         100.000000
Name: LoanMonthsSinceOrigination, dtype: float64
```

In [203…
```
feature_list = ['LoanMonthsSinceOrigination']
hist_plotter(loan, feature_list, 1, 0, 10);
```



Observations

- Most of the loans captured in this dataset are in their first ten months.

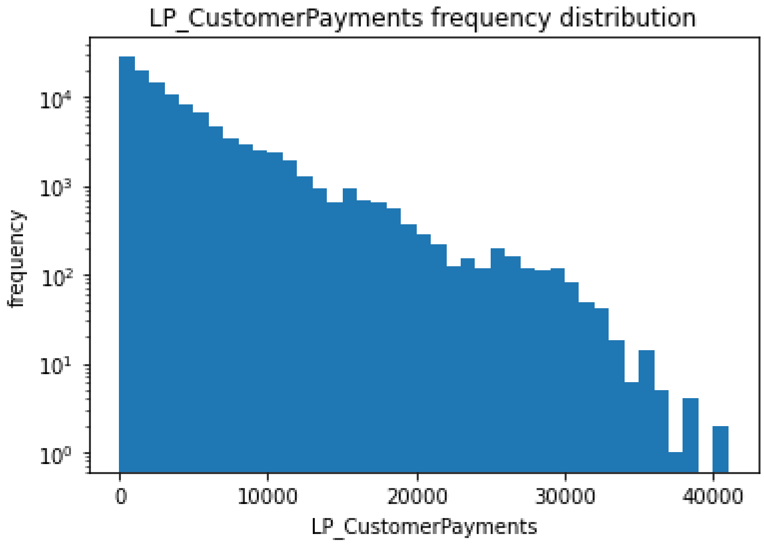
The current loan charges

LP_CustomerPayments

In [204…
```
loan['LP_CustomerPayments'].describe()
```

Out[204]:
```
count    113937.000000
mean       4183.079489
std        4790.907234
min          -2.349900
25%        1005.760000
50%        2583.830000
75%        5548.400000
max       40702.390000
Name: LP_CustomerPayments, dtype: float64
```

In [205…
```
feature_list = ['LP_CustomerPayments']
hist_plotter(loan, feature_list, 1, 0, 1000);
plt.yscale('log')
```



Observation

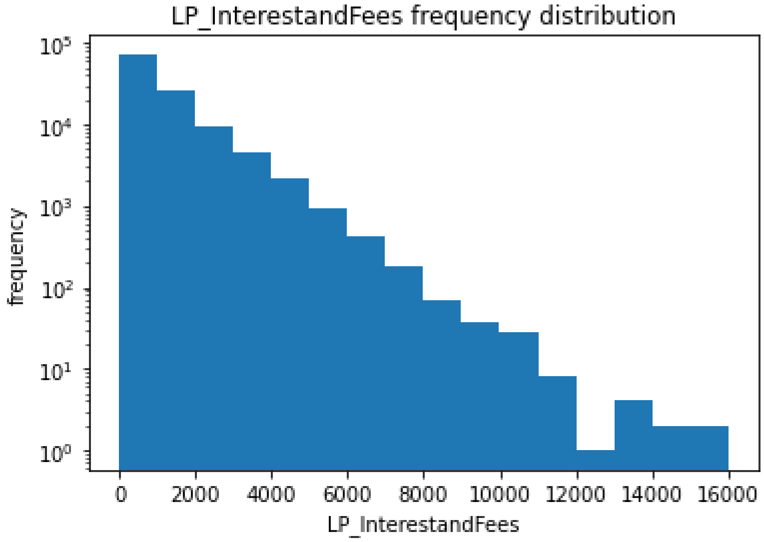
-The modal class is between 0 and 1000. This is reasonable since most of the loans are in their first ten months and most monthly payments are in hundreds of dollars.

LP_InterestandFees

In [206…
```
loan['LP_InterestandFees'].describe()
```

Out[206]:
```
count    113937.000000
mean       1077.542901
std        1183.414168
min          -2.349900
25%         274.870000
50%         700.840100
75%        1458.540000
max       15617.030000
Name: LP_InterestandFees, dtype: float64
```

In [207…
```
feature_list = ['LP_InterestandFees']
hist_plotter(loan, feature_list, 1, -2, 1000);
plt.yscale('log')
```



Observations

- The modal class is again between 0 and 1000. since most of the customer payments

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

> The LoanStatus feature has a large difference in the count of the categories. Therefore, I looked at the data using log transform. It was observed that 49.7% of the loans are current, 33.4% of the Loans are completed, 10.5% are charged off.
>
> The BorrowerAPR feature has an overall trend that is skewed to the right. This shows that the population is higher at lower BorrowerAPR and lower at higher BorrowerAPR. However, there are spikes along these trends that are higher than the surrounding regions. Also, it is interesting to note that the modal class is between 0.35 and 0.36 which is very muchagainst the trend.

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?

> The CreditScoreRangeLower and CreditScoreRangeUpper typically should range between 300 and 850. However, there were some CreditScoreRangeLower values of 19 and CreditScoreRangeUpper values of 0. This must have been due to some entry errors. These set of values has been removed going forward.

- Return

Bivariate Exploration

We will take the same approach as before by observing the pairs in their observational units. However, most of the emphasis will be on the **listing** DataFrame.

Then we will observe the pair of other features in the other two data frame (i.e) with respect to the target variables.

Major subheadings

In [208…

```
listing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 16 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ListingKey                113937 non-null  object
 1   ListingNumber             113937 non-null  int64
 2   ListingCreationDate       113937 non-null  datetime64[ns]
 3   CreditGrade               28953 non-null   category
 4   Term                      113937 non-null  category
 5   LoanStatus                113937 non-null  category
 6   ClosedDate                55089 non-null   datetime64[ns]
 7   BorrowerAPR               113912 non-null  float64
 8   BorrowerRate              113937 non-null  float64
 9   LenderYield               113937 non-null  float64
 10  EstimatedEffectiveYield  84853 non-null   float64
 11  EstimatedLoss             84853 non-null   float64
 12  EstimatedReturn           84853 non-null   float64
 13  ProsperRating             84853 non-null   category
 14  ProsperScore              84853 non-null   category
 15  ListingCategory           113937 non-null  category
dtypes: category(6), datetime64[ns](2), float64(6), int64(1), object(1)
memory usage: 9.3+ MB
```

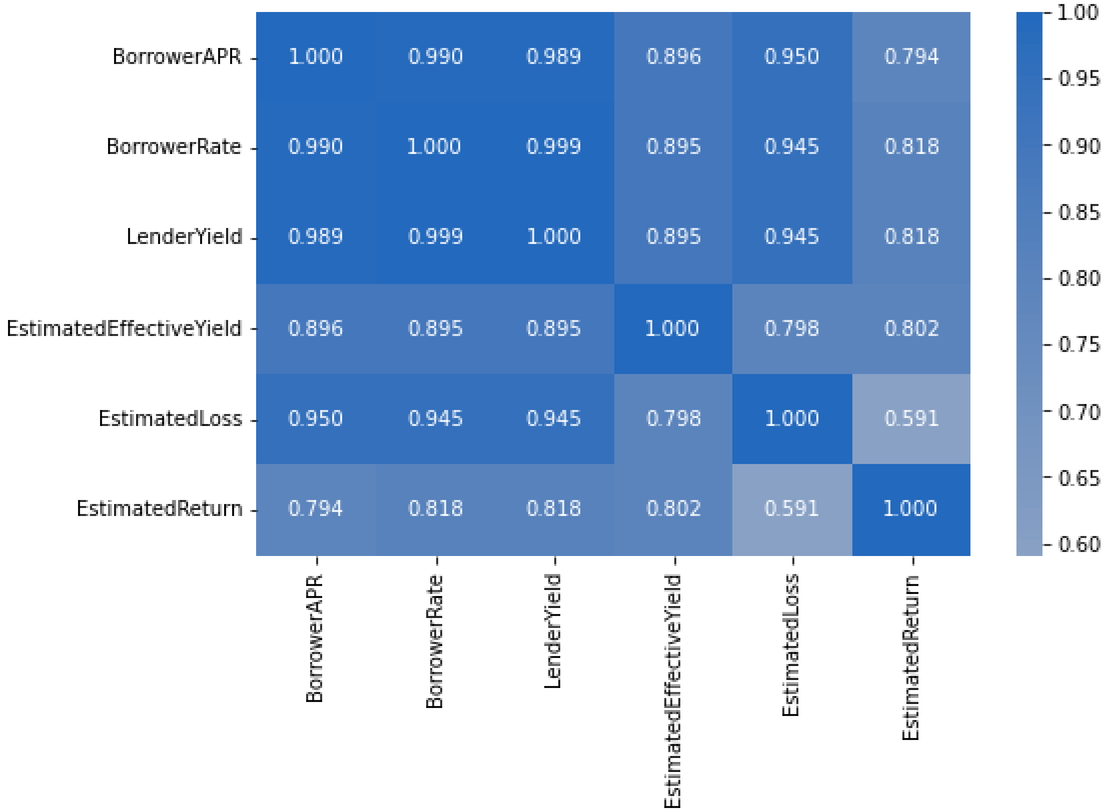
Numerical Variables in the listing DataFrame

In [209…

```
# Dividing the features in the listing DataFrame into numerical and categorical variable
numeric_vars = ['BorrowerAPR', 'BorrowerRate', 'LenderYield',
                'EstimatedEffectiveYield', 'EstimatedLoss', 'EstimatedReturn']
categoric_vars = ['CreditGrade', 'Term', 'LoanStatus',
                  'ProsperRating', 'ProsperScore', 'ListingCategory' ]
```

In [210…

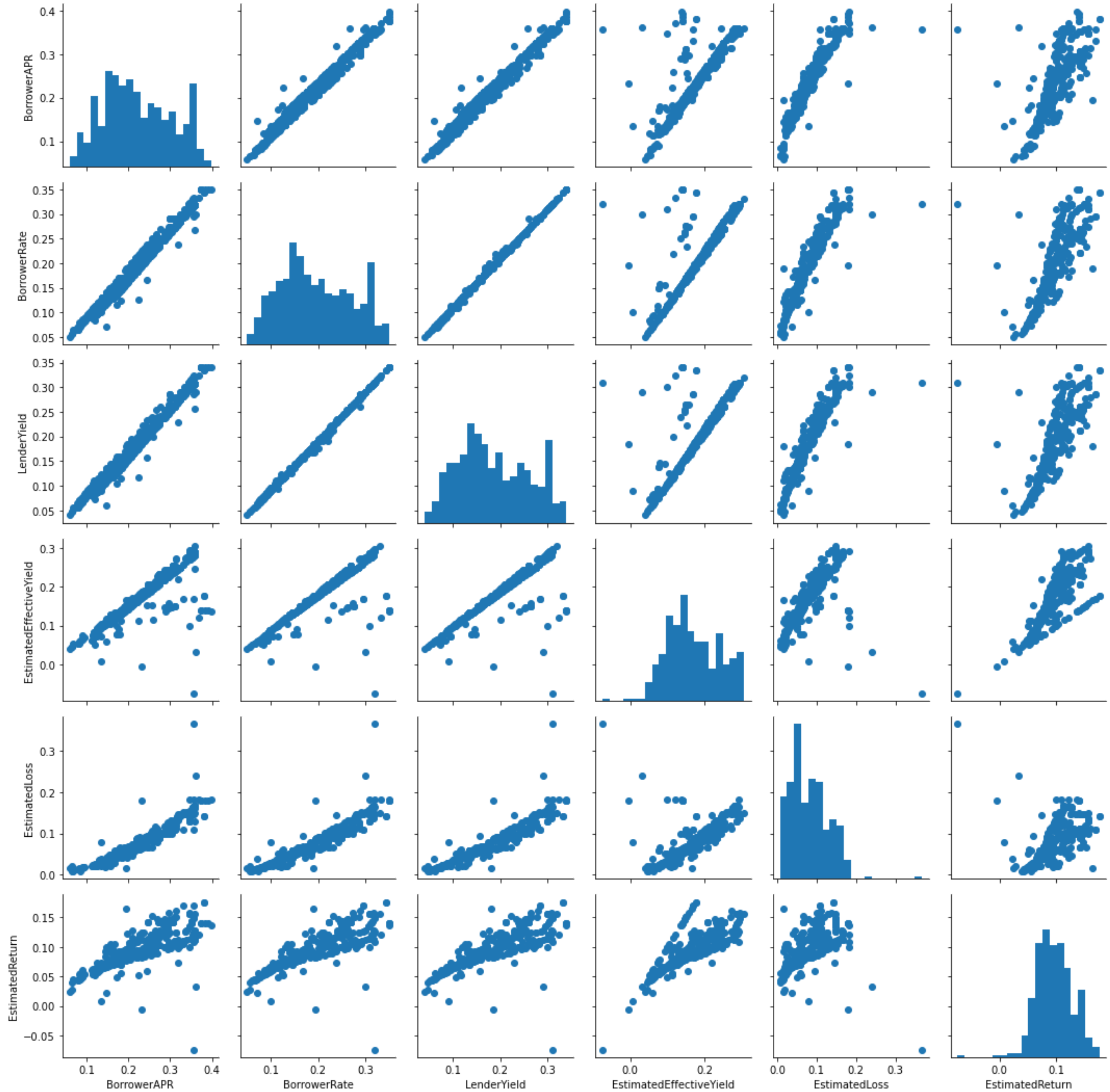
```
# Corrlation plot
plt.figure(figsize = [8,5])
sb.heatmap(listing[numeric_vars].corr(), annot = True, fmt = '.3f', cmap = 'vlag_r', cen
```



In [211…

```
# plot matrix: sample 500 data point so that the plots are clearer and they render faste
print('listing.shape',listing.shape)
listing_samp = listing.sample(n = 500, replace = False)
print('listing_samp.shape =', listing_samp.shape)

g = sb.PairGrid(data = listing_samp, vars = numeric_vars)
g= g.map_diag(plt.hist, bins = 20);
g.map_offdiag(plt.scatter);
```

```
listing.shape (113937, 16)
listing_samp.shape = (500, 16)
```



Observation

As expected, BorrowersRate, LendersYield, and EstimatedEffectiveYield all show strong correlation with BorrowersAPR. The correlation is not as perfect for EstimatedEffectiveYield that has several outliers. However, the outliers all show the same pattern by being on the same side of the more regular plot points.

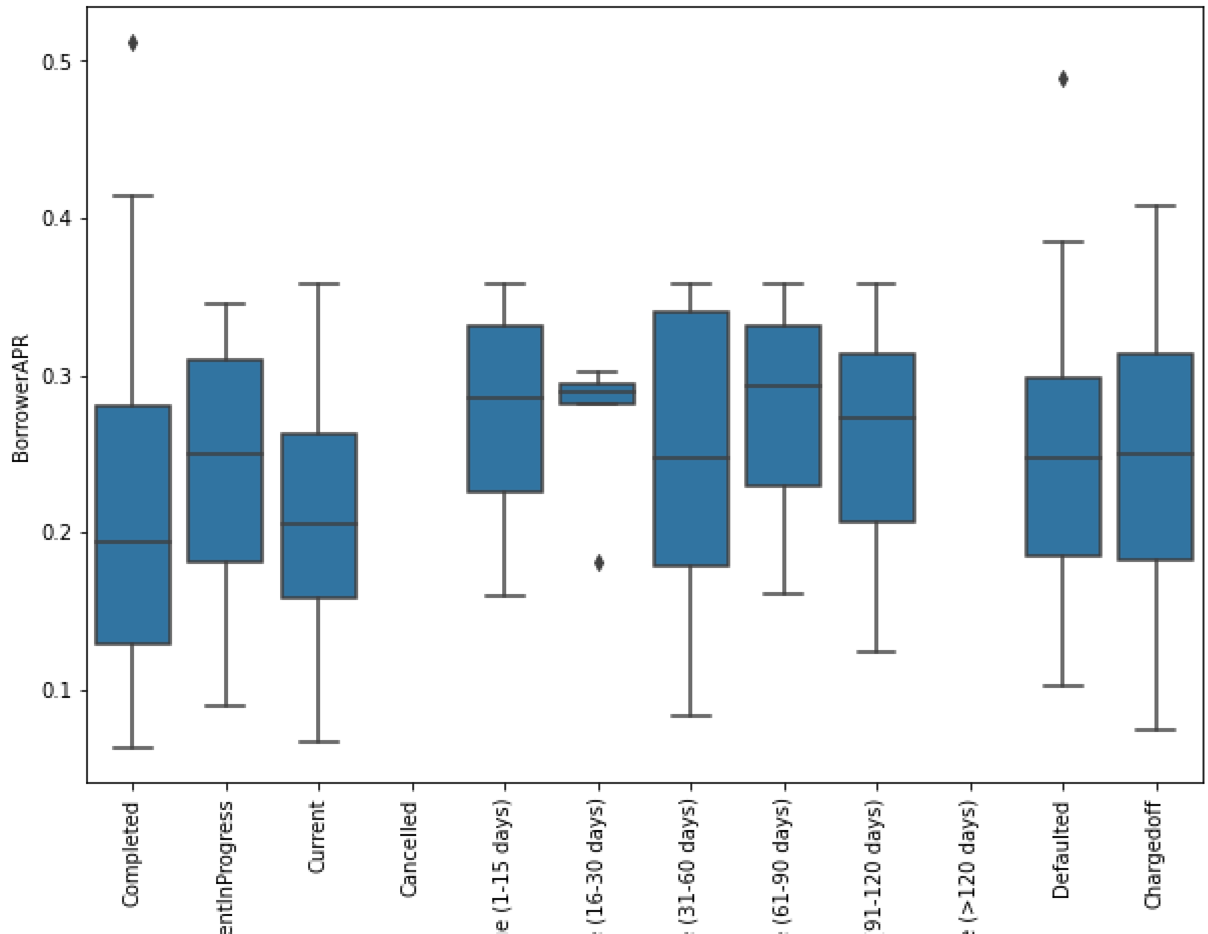
EstimatedReturn and Estimated loss show lesser correlation with BorrowersAPR as compared with the previous set of variables. Although, they are also positively correlated.

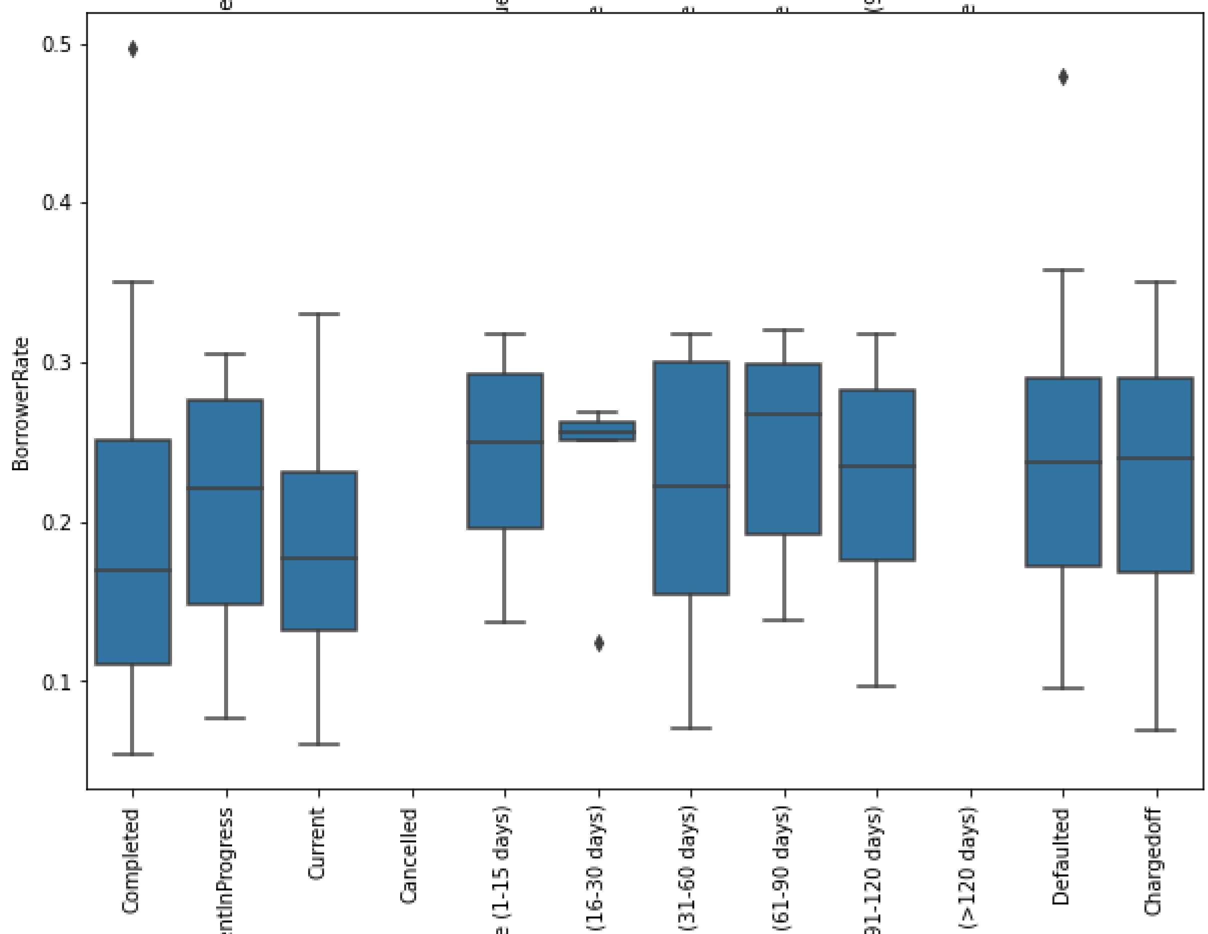
LoanStatus and numerical variables.

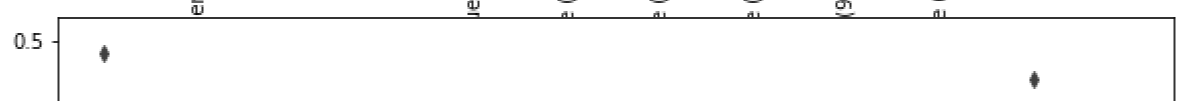
In [212…

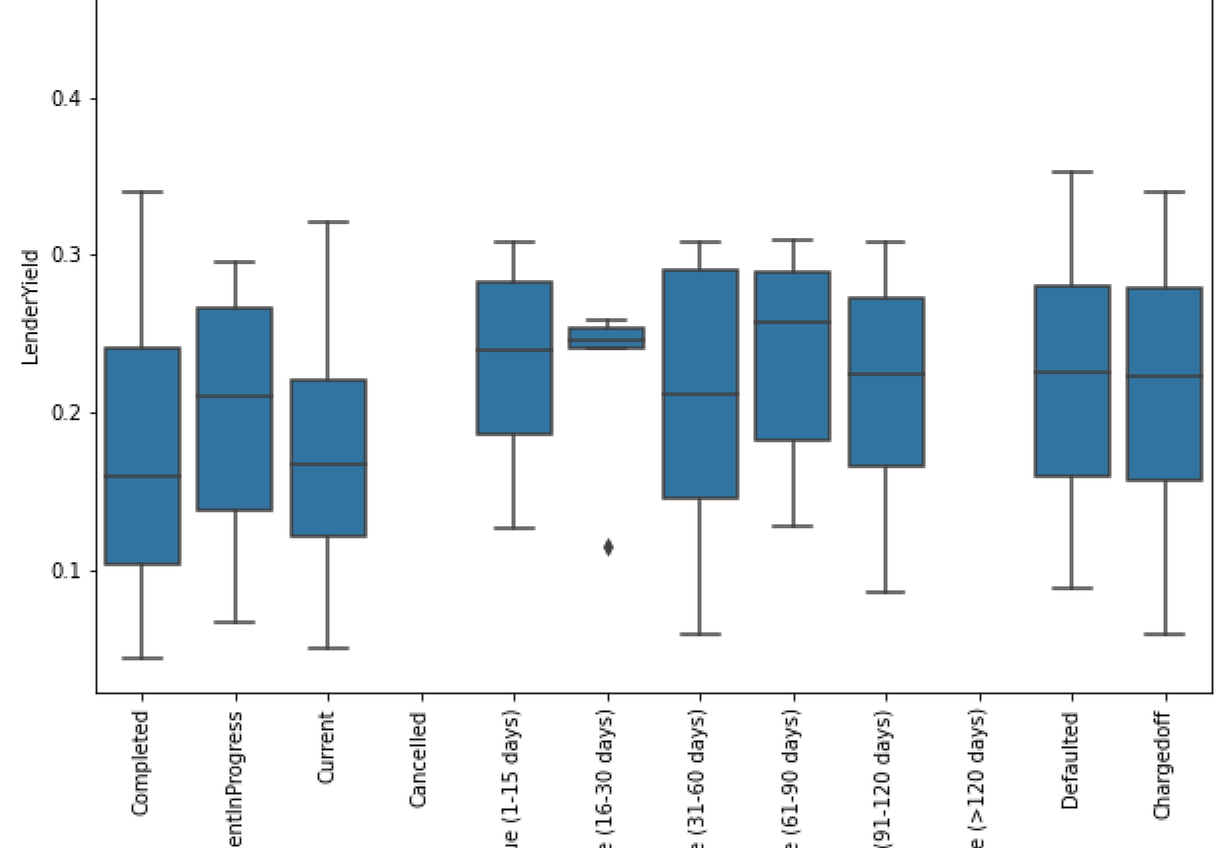
```
# plot the LoanStatus against the numerical variables.
# Using a sample size of 2000
fig, ax = plt.subplots(nrows = 6, figsize = [10,50])
listing_sample = listing.sample(n = 2000, replace = False)
default_color = sb.color_palette()[0]
for i in range(len(numeric_vars)):
    var = numeric_vars[i]
```

```
    g = sb.boxplot(data = listing_sample, x = 'LoanStatus', y = var, ax =ax[i],
                   color = default_color)
    g.set_xticklabels(g.get_xticklabels(), rotation = 90)
```

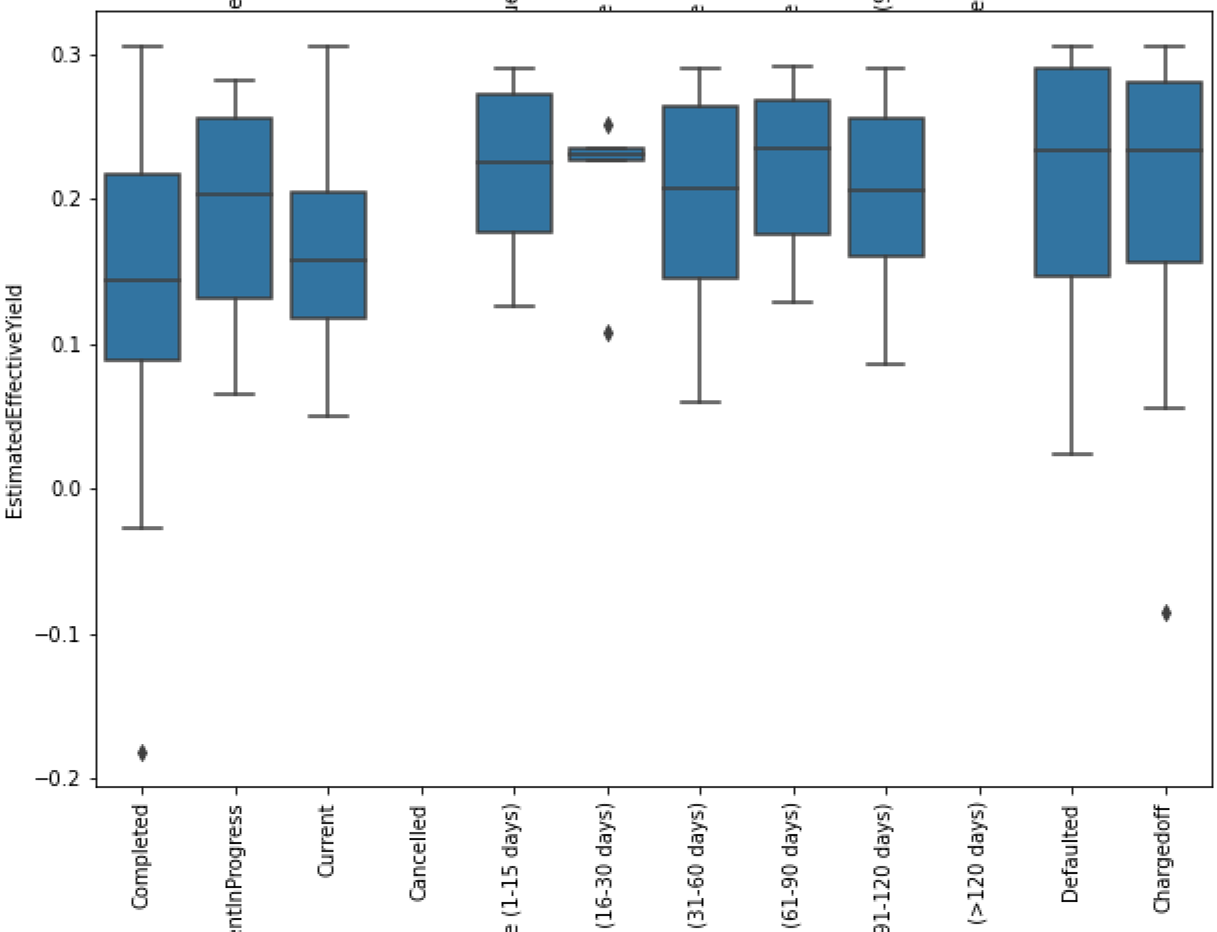




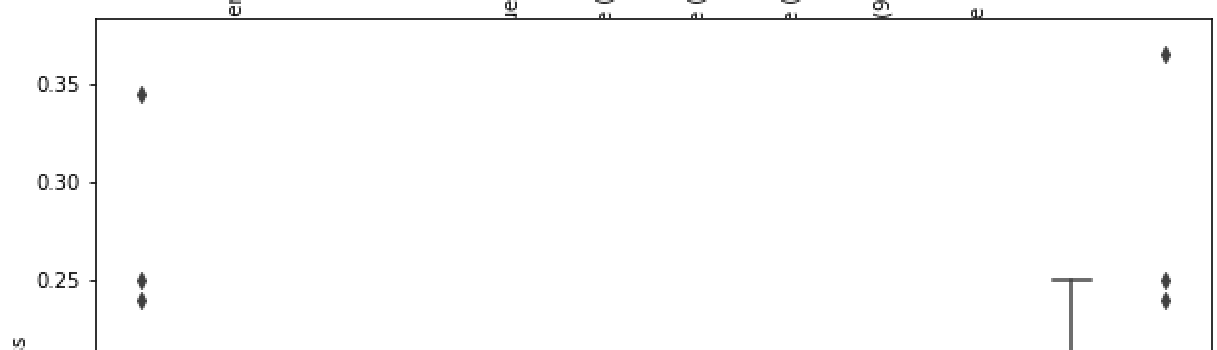




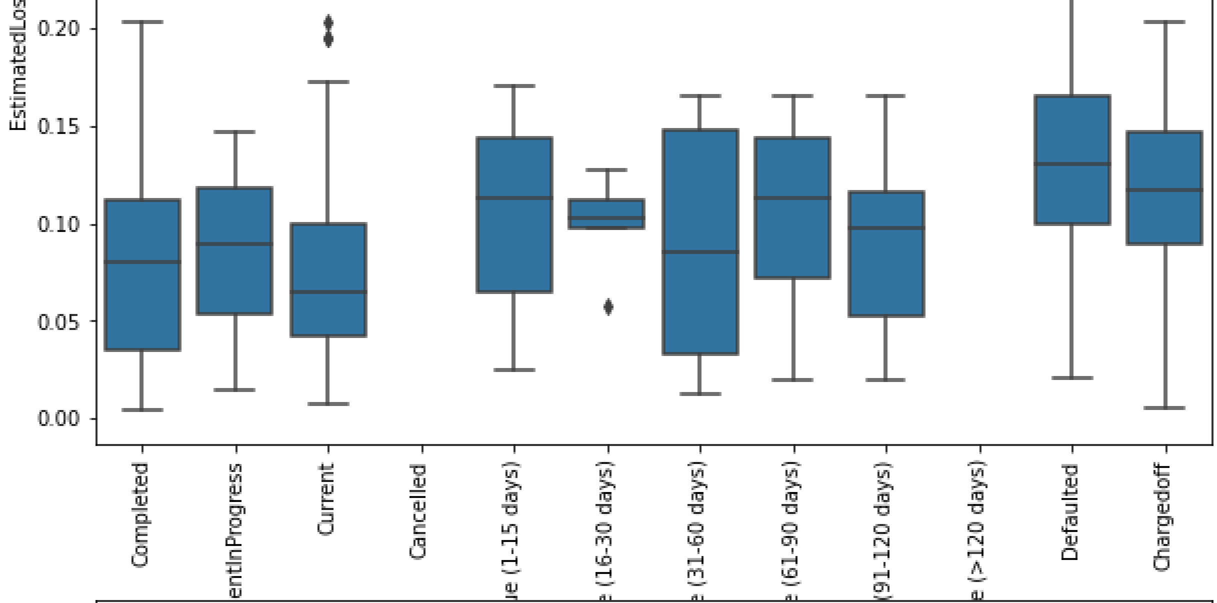
LenderYield
0.4
0.3
0.2
0.1
Completed
entInProgress
Current
Cancelled
ue (1-15 days)
e (16-30 days)
e (31-60 days)
e (61-90 days)
(91-120 days)
e (>120 days)
Defaulted
Chargedoff

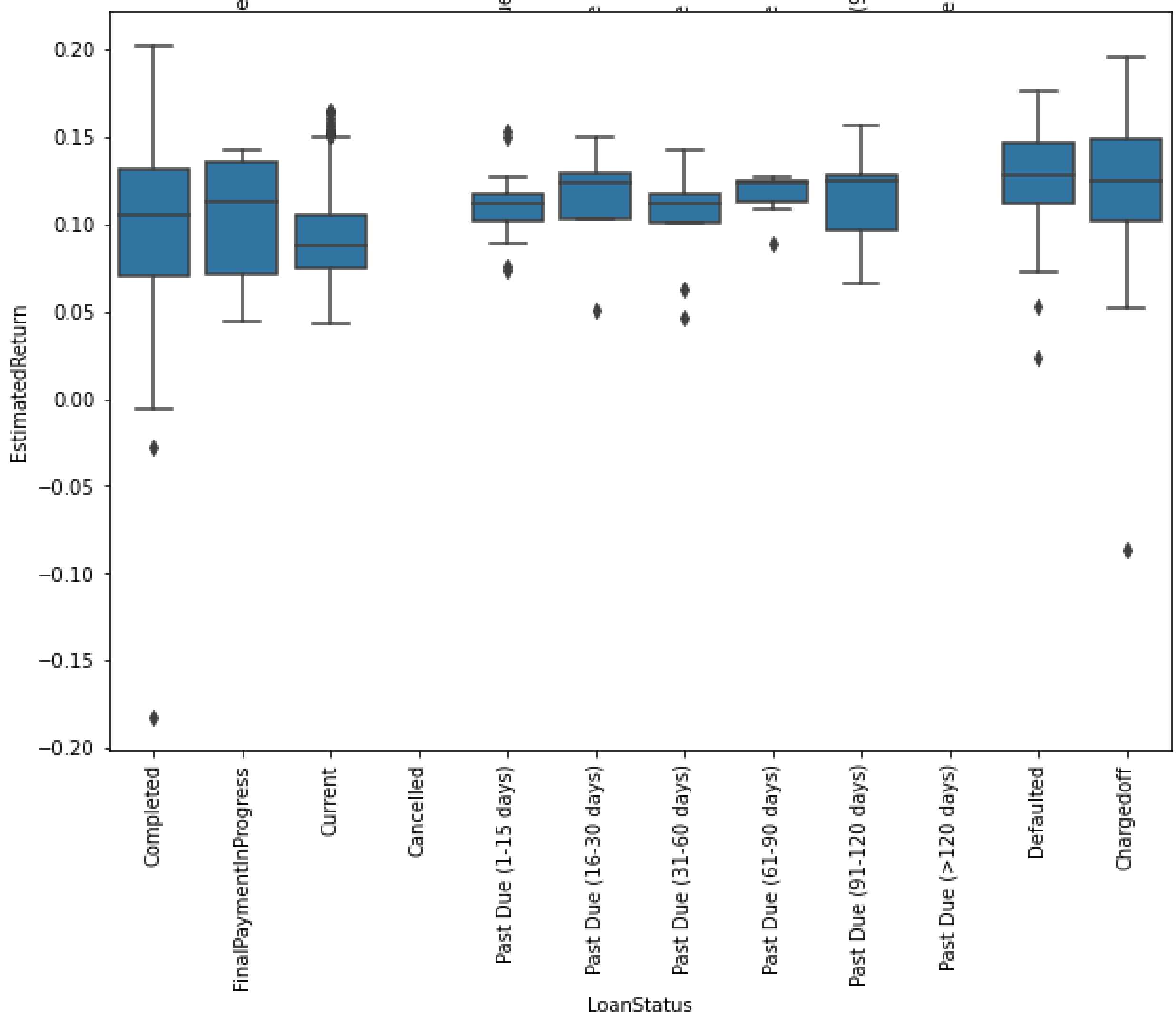


EstimatedEffectiveYield
0.3
0.2
0.1
0.0
−0.1
−0.2
Completed
entInProgress
Current
Cancelled
ue (1-15 days)
e (16-30 days)
e (31-60 days)
e (61-90 days)
(91-120 days)
e (>120 days)
Defaulted
Chargedoff



0.35
0.30
0.25





Observation:

- generally, the EstimatedReturn and EstimatedLoss have an average lower median value as compared to the other 4 variables.
- The LoanStatus, clearly has an influence on the numerical variables. LoanStatus completed, tend to have the lowest median value for each of the numerical variable and the medians tends to increase as you go across the level.
- These is actually reasonable since a higher APR will be associated with increased risk of default.

BorrowersAPR and Categorical variables

How does the BorrowersAPR vary with the categorical variables ?

In [213…
```
listing[categoric_vars].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   CreditGrade      28953 non-null   category
 1   Term             113937 non-null  category
 2   LoanStatus       113937 non-null  category
 3   ProsperRating    84853 non-null   category
 4   ProsperScore     84853 non-null   category
 5   ListingCategory  113937 non-null  category
dtypes: category(6)
memory usage: 670.0 KB
```

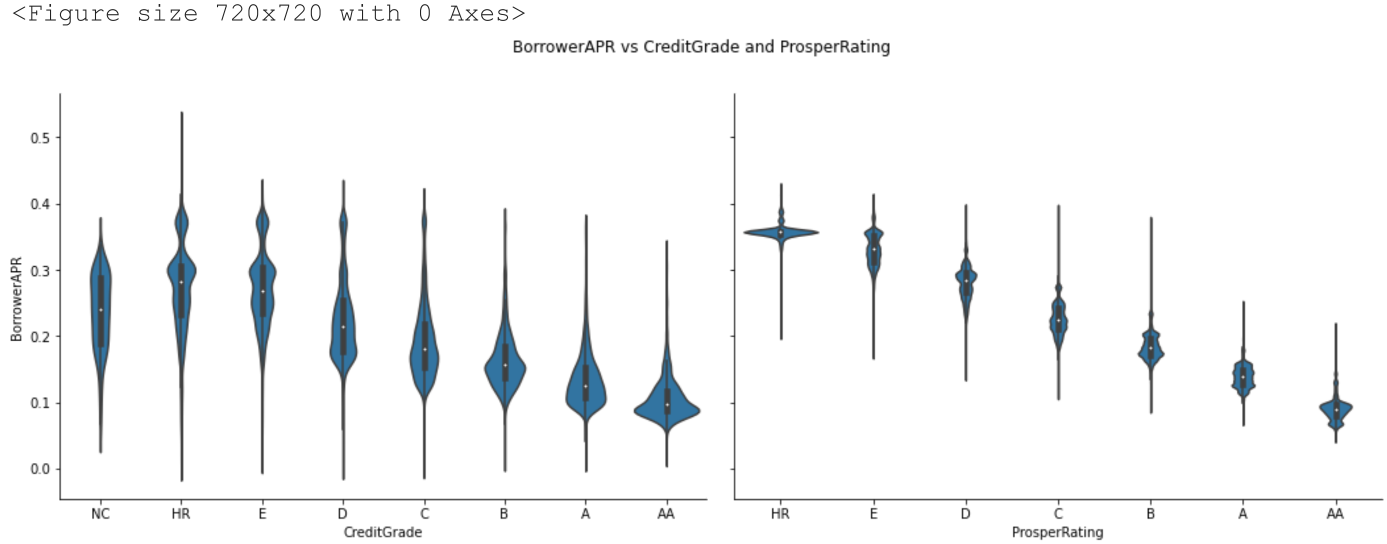
BorrowerAPR vs CreditGrade and ProsperRating

In [214…
```
category_1 = ['CreditGrade', 'ProsperRating']
```

In [215…
```
def violingrid(x, y, **kwargs):
    ''' Quick hack for creating violin plots with seaborn's PairGrid.'''

    default_color = sb.color_palette()[0]
    sb.violinplot(x = x, y = y, color = default_color)

plt.figure(figsize = [10, 10])
g = sb.PairGrid(data = listing, y_vars = 'BorrowerAPR', x_vars = category_1,
                height = 5, aspect = 1.5)
g.map(violingrid)
g.fig.suptitle ('BorrowerAPR vs CreditGrade and ProsperRating', y = 1.08)
```

Out[215]:
```
Text(0.5, 1.08, 'BorrowerAPR vs CreditGrade and ProsperRating')
```

```
<Figure size 720x720 with 0 Axes>
```



Observation:

- The CreditGrade rating is the rating employed by prosperloans before 2009. Its a way to classify the loan listing by level of risk. It can be observed that the rating approach is not as linear as the more

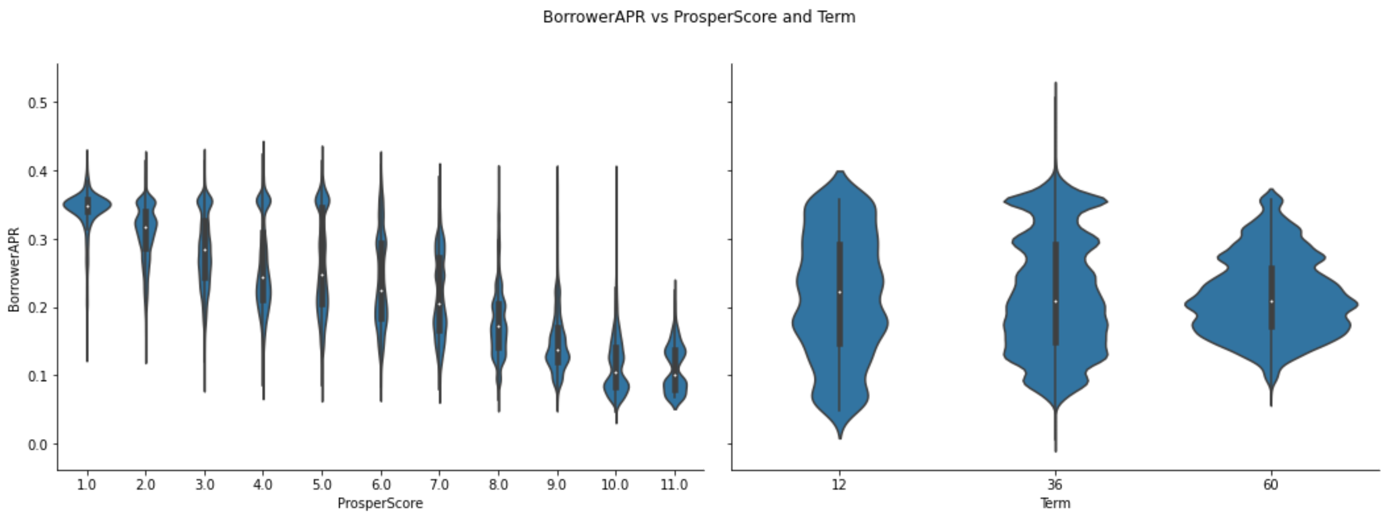
recent ProsperRating which was introduced post 2009.
- They both show that the more risky the loan the higher the borrowerAPR. HR being the most risky while AA being the least risky.

ProsperScore and Term

In [216…

```
category_2 = ['ProsperScore', 'Term']
plt.figure(figsize = [5, 5])
g = sb.PairGrid(data = listing, y_vars = 'BorrowerAPR', x_vars = category_2,
                height = 5, aspect = 1.5)
g.map(violingrid)
g.fig.suptitle ('BorrowerAPR vs ProsperScore and Term', y = 1.08);
```

```
<Figure size 360x360 with 0 Axes>
```



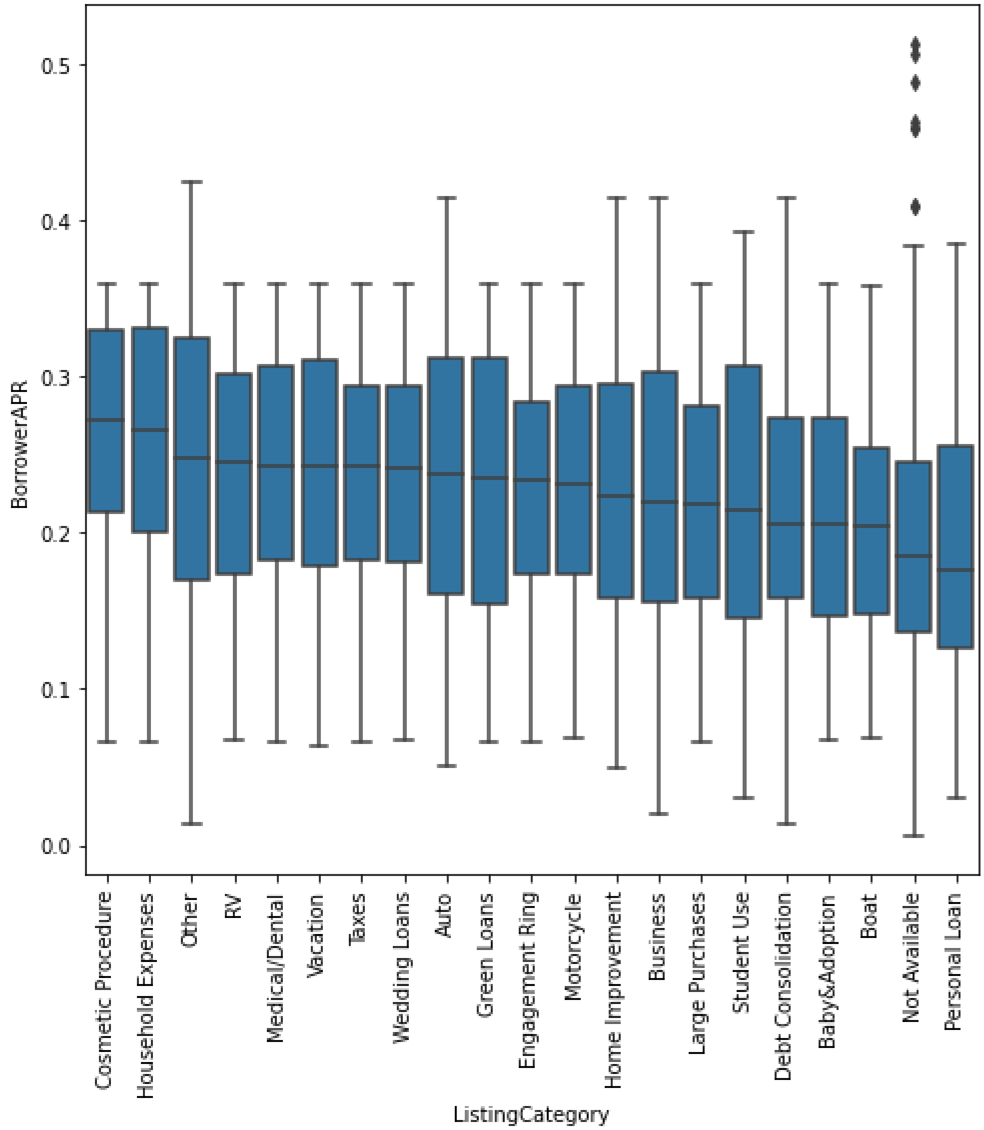
Observation

- ProsperScore is the customer's risk score based on their previous Loans on Prosper platform with 1 being highly risky and 10 being least risky according to the variable definition. There is another level of 11 in the ProsperScore column that doesn't fit in properly with the trend.
    - It can be observed that the BorrowerAPR reduces as the risk value for an individual reduces.
    - The borrowerAPR is not affected by the term of the loan. Although the 60 months term seem a little wider on the lower end suggesting that longer term loans get a little lower APR.

BorrowerAPR vs ListingCategory

In [217…

```
# Defining the order
listing_category_median = listing.groupby('ListingCategory')['BorrowerAPR'].median().sor

# Plotting the boxplot
plt.figure(figsize = [8, 8])
g = sb.boxplot(data = listing, y = 'BorrowerAPR', x = 'ListingCategory', color = default

plt.xticks(rotation = 90);
```



Observation

- It can be observed that Personal loan has lowest median APR while Cosmetic Procedure has the highest median APR

Loan Status and Categorical Variable

LoanStatus and

In [218…

```
categoric_vars
```

Out[218]:

```
['CreditGrade',
 'Term',
 'LoanStatus',
 'ProsperRating',
 'ProsperScore',
 'ListingCategory']
```

In [219…

```
def clustered_barchart(df, x ,hue, ncol = 1, rot = 90, scale = 'log',color = 'Blues'):
    ''' Plots a clustered bar chart using seaborn countplot and anotate it countplot  ''


    sb.countplot(data = df, x = x, hue = hue, palette = color)
    plt.legend(title = hue, bbox_to_anchor = (1.04,1), ncol =ncol)
    plt.yscale(scale)
    plt.tight_layout()
```

```
    plt.title('Count of {} for different levels of {}'.format(x,hue))
    plt.xticks(rotation = rot);
```

In [220…

```
def FacetGrid_Plotter(df, target_feature, var, var2 = None, scale = 'log', col_wrap = 3,
                      height = 3,  aspect = 1.5, order = None, kind = sb.countplot):
    ''' plots a Countplot FacetGrid '''
    if var2 == None:
        g= sb.FacetGrid(data = df, col = target_feature, col_wrap=col_wrap, height = hei
        g.map(kind, var, order = order)
        g.set(yscale = scale)
        g.fig.suptitle('{} vs {}'.format(target_feature, var), y = 1.04)

        return(g)
    else:
        g= sb.FacetGrid(data = df, col = target_feature, col_wrap=col_wrap, height = hei
        g.map(kind, var,var2)
        g.set(yscale = scale)
        g.fig.suptitle('{} vs {}'.format(target_feature, var), y = 1.04)
```

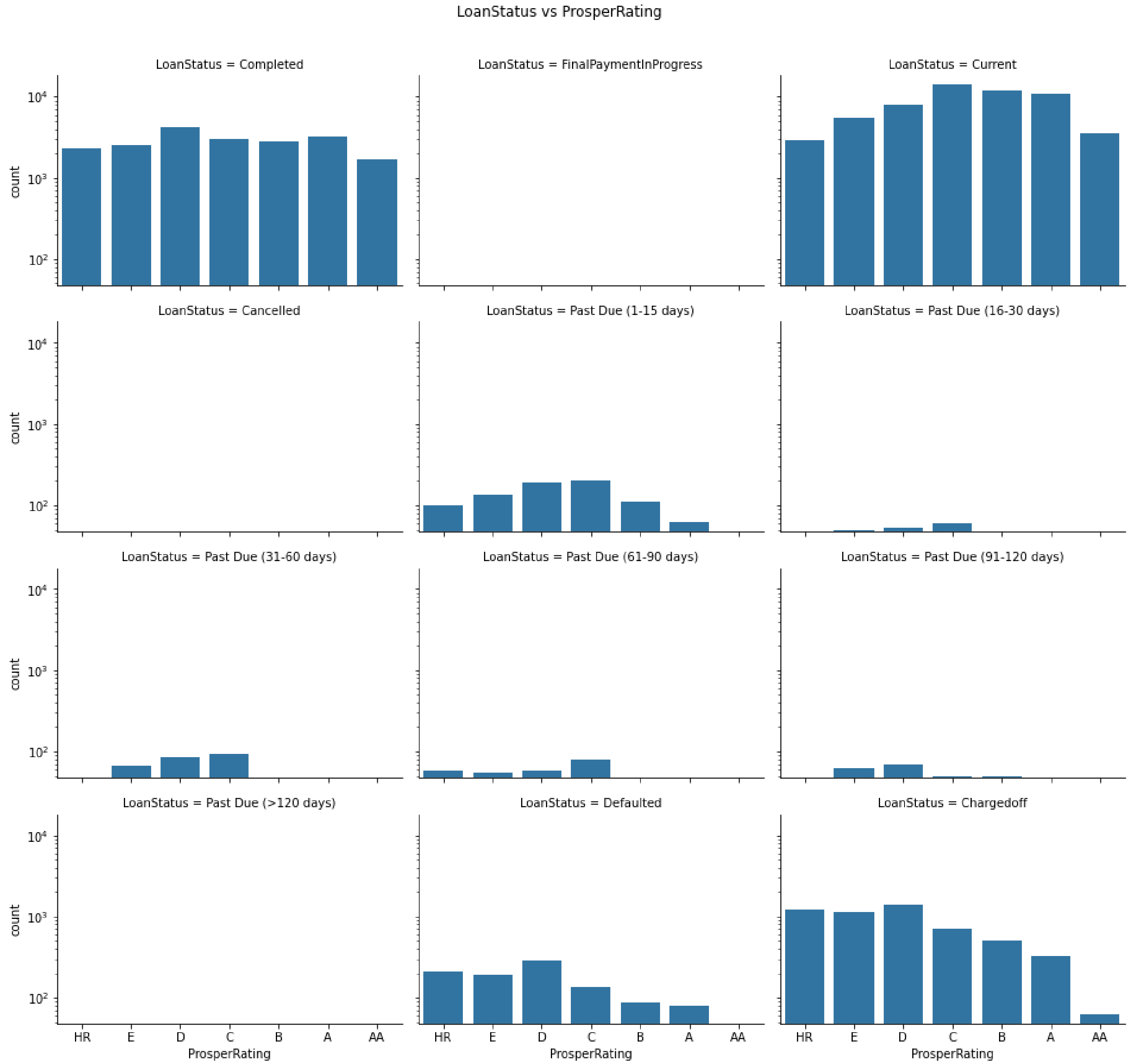
LoanStatus vs ProsperRating

In [221…

```
# Using the FacetGrid_Plotter Function
FacetGrid_Plotter(listing, 'LoanStatus', 'ProsperRating')
```

Out[221]:

```
<seaborn.axisgrid.FacetGrid at 0x2555510fbe0>
```



Observation

- The D level of the ProsperRating is always the most frequent across the different levels of LoanStatus.
- The completed level of the LoanStatus shows a rather flat relationship with prosperRating.
- The Defaulted and Chargedoff level of the LoanStatus shows a negative relationship with improving prosperRating. This is more obvious from the D to AA rating.

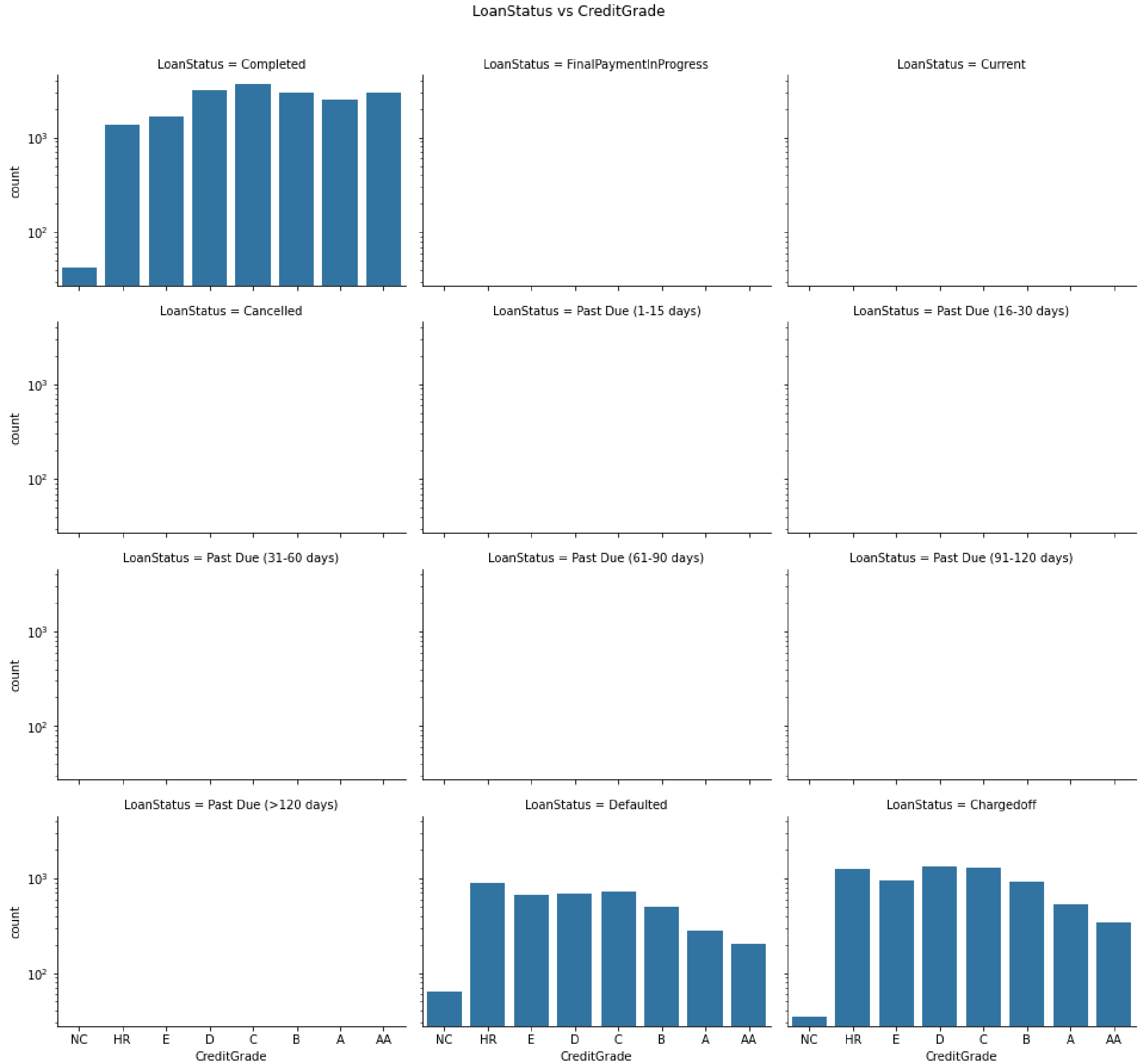
LoanStatus vs CreditGrade

In [222…

```
# Using the FacetGrid_Plotter Function
FacetGrid_Plotter(listing, 'LoanStatus', 'CreditGrade')
```

Out[222]:

```
<seaborn.axisgrid.FacetGrid at 0x25553f8dd60>
```



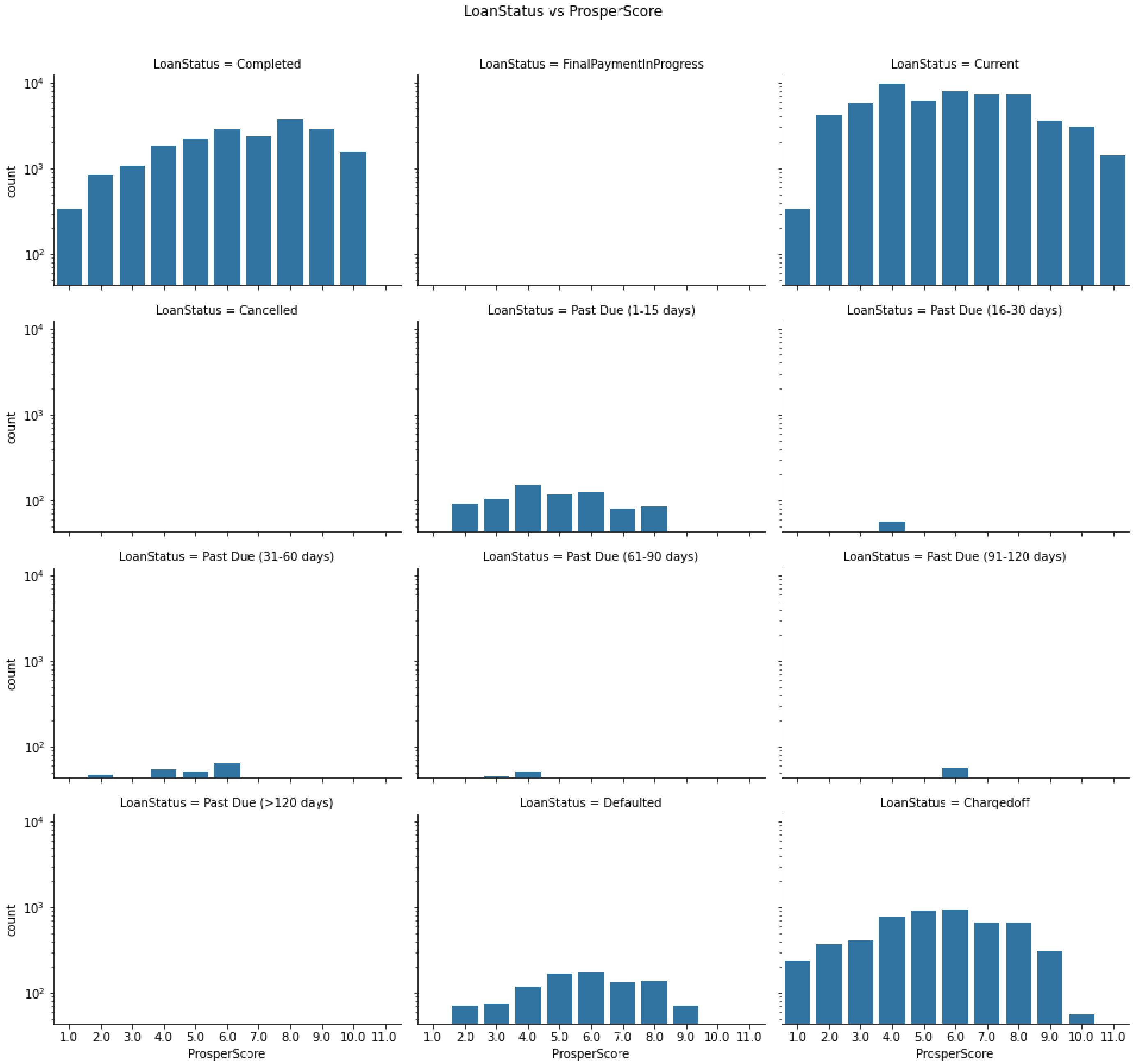
- CreditGrade was the prosper listing rating system Pre 2009. this plot reveals that there were less categorisation of the loanStatus as at then. Pre 2009,Its either the Loan was completed, Defaulted, or Chargedoff.
- A lot of HighRisk Loans got Chargedoff or defaulted
- The rate of Default is also higher as compared to what is obtained in the ProsperRating era(i.e post 2009)
- The **prosperRating** seem an overall better predictor of the **loanStatus** than **CreditGrade**, as it has lesser count for defaulted and chargedoff loans for higher level rating like **B**, **A** and **AA**.

In [223…

```
# Using the FacetGrid_Plotter Function

FacetGrid_Plotter(listing, 'LoanStatus', 'ProsperScore', col_wrap = 3)
```

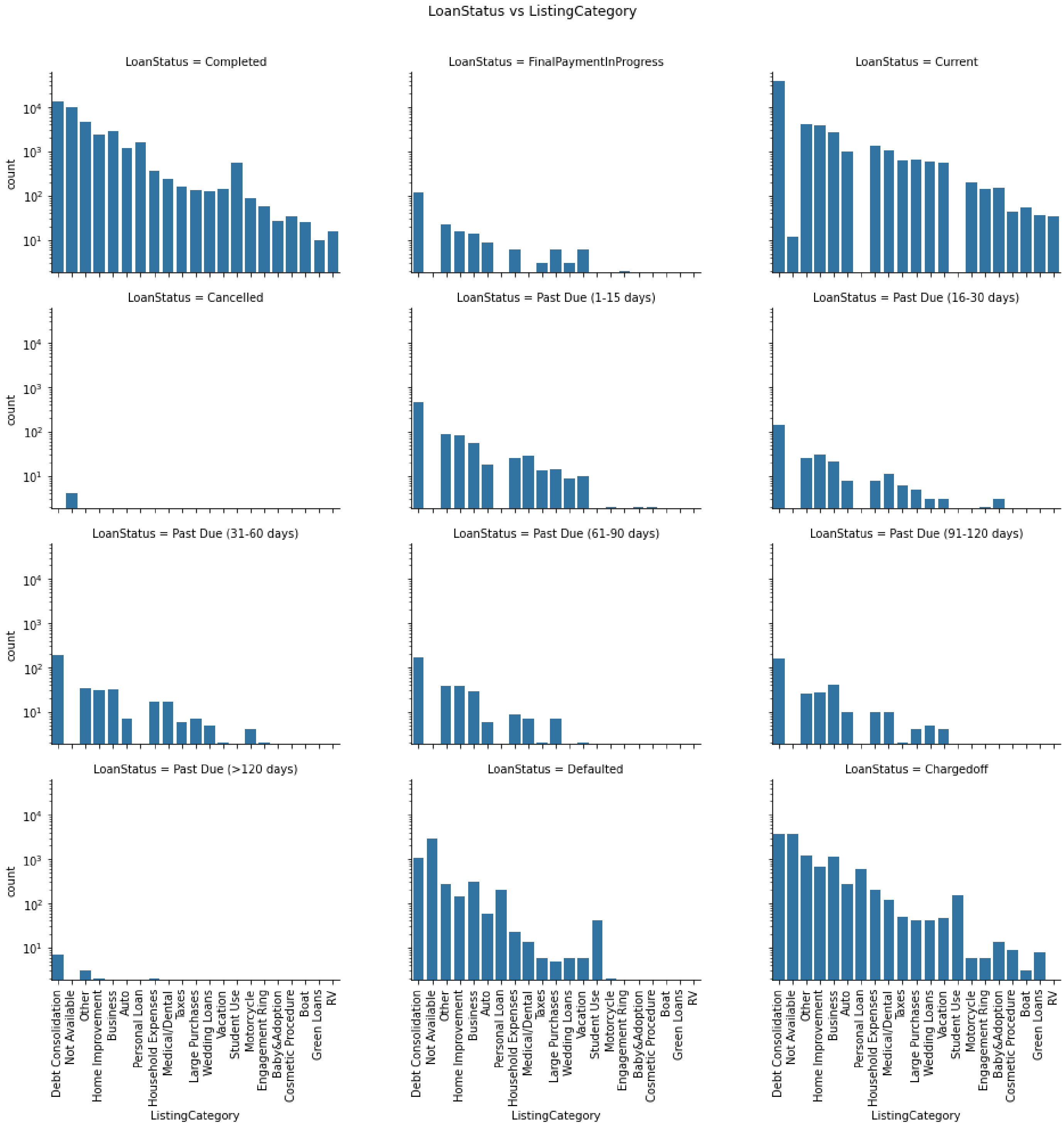
Out[223]:

```
<seaborn.axisgrid.FacetGrid at 0x25553fd1bb0>
```



In [224…

```
# Using the FacetGrid_Plotter Function
ax = FacetGrid_Plotter(listing, 'LoanStatus', 'ListingCategory', col_wrap = 3, order = l
#ax.set_xticklabels(xticks = ax.get_xticklabels(), rotation = 90)
for axes in ax.axes.flat:
    axes.set_xticklabels(axes.get_xticklabels(),
                         rotation=90)
```

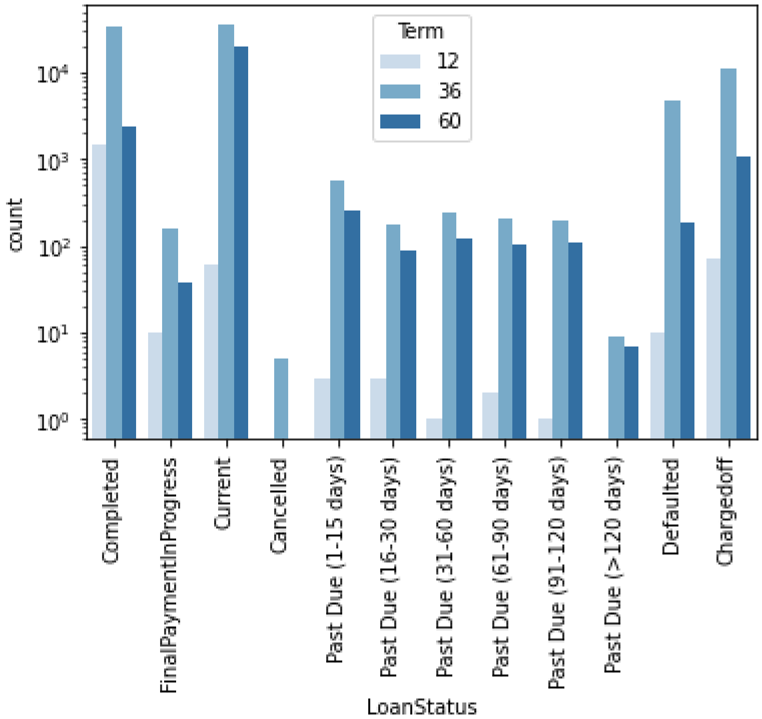


Time and Target Features

Count of Terms and LoanStatus

In [225…

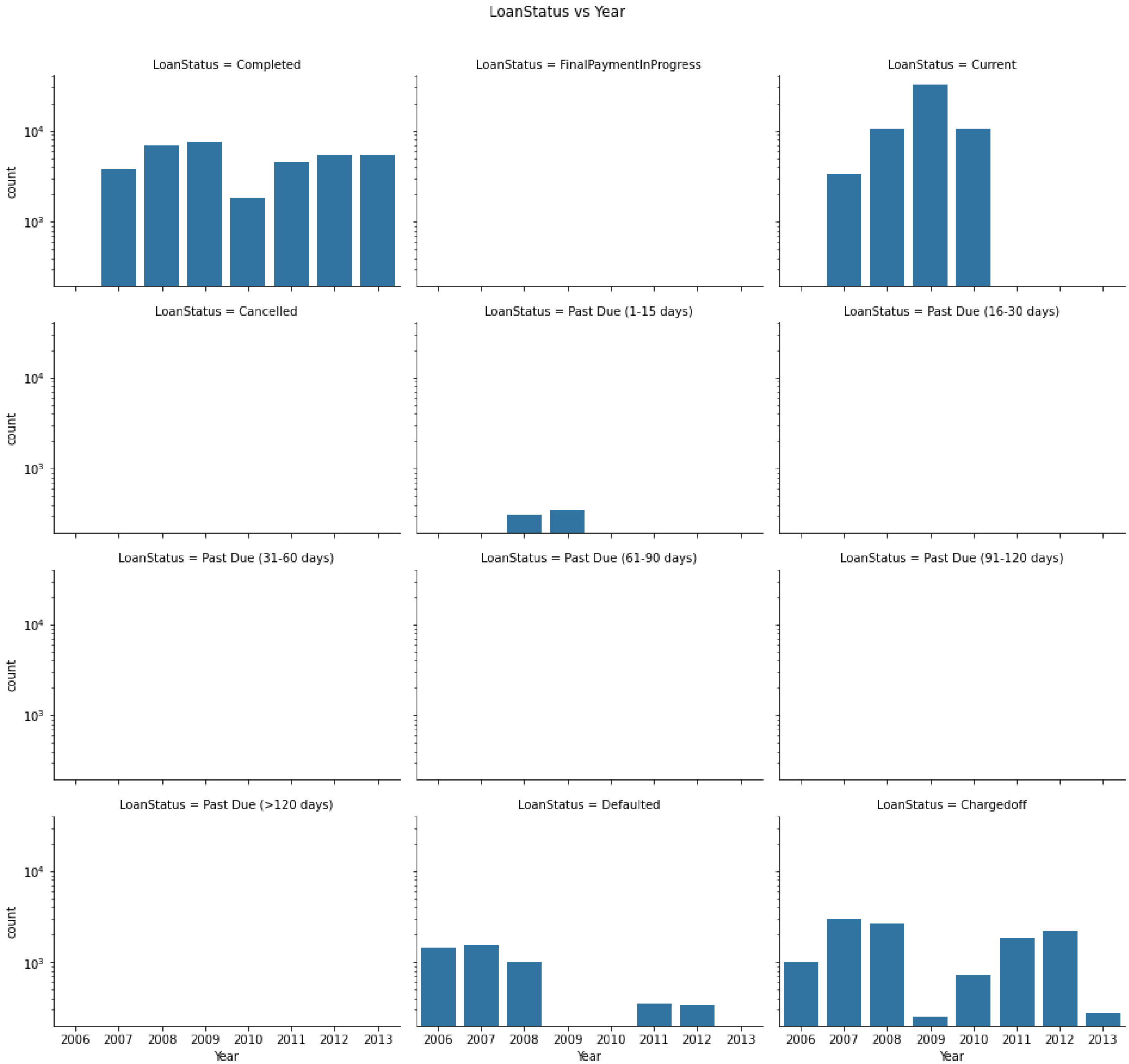
```
# Plot of count of terms and LoanStatus
g = sb.countplot(data = listing, x = 'LoanStatus', hue= 'Term', palette = 'Blues')
g.set(yscale = 'log');
g.set_xticklabels(g.get_xticklabels(), rotation = 90);
```



ListingCreationDate and LoneStatus

In [226…

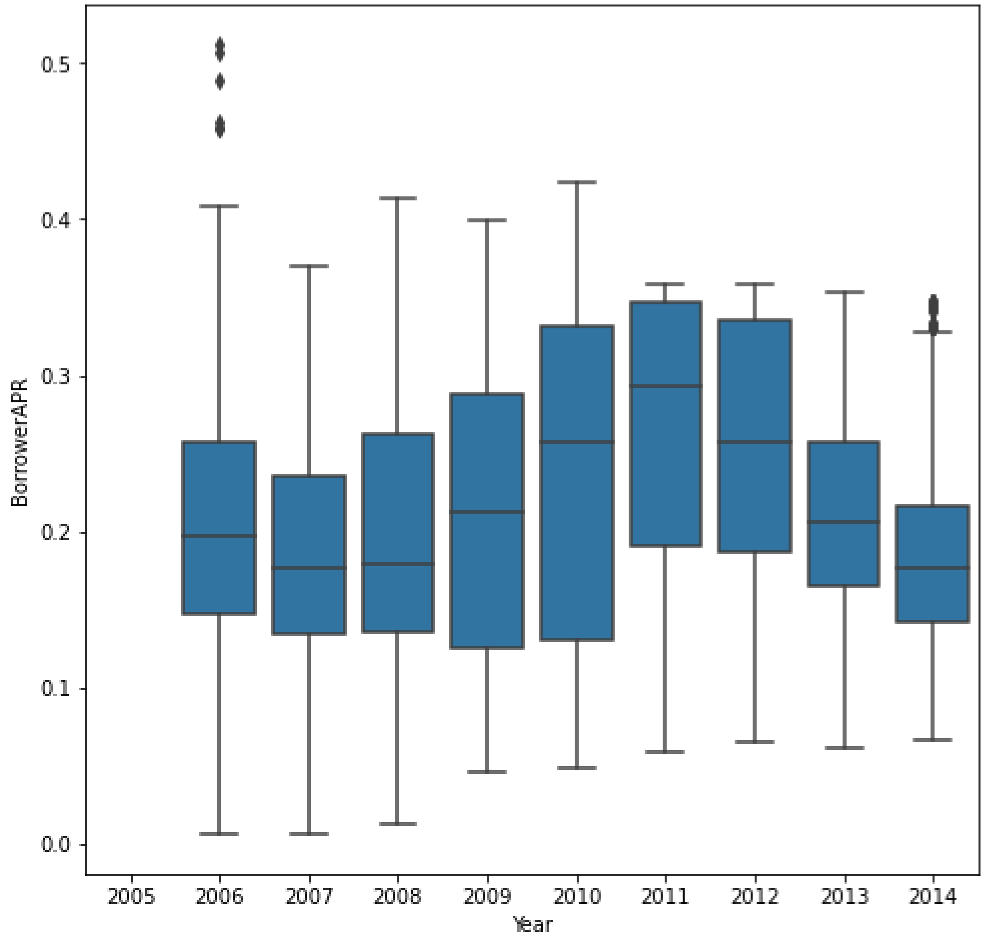
```
listing['Year'] = listing.ListingCreationDate.apply(lambda x: x.year)
ax = FacetGrid_Plotter(listing, 'LoanStatus', 'Year', col_wrap = 3)
```



BorrowerAPR vs ListingCategory(Year)

In [227…

```
# Plotting the boxplot for
plt.figure(figsize = [8, 8])
g = sb.boxplot(data = listing, y = 'BorrowerAPR', x = 'Year', color = default_color)
```



In [ ]:

In [ ]:

In [ ]:

borrowers_profile and Target Features.

In [228…

```
# Merge selected column from the borrowers_profile DataFrame to listing DataFrame
borrowers_profile_select = ['ListingKey','BorrowerState', 'Occupation','EmploymentStatus
listing_borrower = listing.merge(borrowers_profile[borrowers_profile_select], on = 'List
```

In [229…

```
listing_borrower.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115654 entries, 0 to 115653
Data columns (total 23 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   ListingKey               115654 non-null  object
 1   ListingNumber            115654 non-null  int64
 2   ListingCreationDate      115654 non-null  datetime64[ns]
 3   CreditGrade              28820 non-null   category
 4   Term                     115654 non-null  category
 5   LoanStatus               115654 non-null  category
 6   ClosedDate               54982 non-null   datetime64[ns]
 7   BorrowerAPR              115629 non-null  float64
 8   BorrowerRate             115654 non-null  float64
 9   LenderYield              115654 non-null  float64
 10  EstimatedEffectiveYield  86703 non-null   float64
 11  EstimatedLoss            86703 non-null   float64
```

```
 12  EstimatedReturn          86703 non-null   float64
 13  ProsperRating            86703 non-null   category
 14  ProsperScore             86703 non-null   category
 15  ListingCategory          115654 non-null  category
 16  Year                     115654 non-null  int64
 17  BorrowerState            110183 non-null  object
 18  Occupation               111974 non-null  object
 19  EmploymentStatus         113431 non-null  object
 20  IsBorrowerHomeowner      115654 non-null  bool
 21  CreditScoreRangeLower    115063 non-null  float64
 22  DebtToIncomeRatio        106991 non-null  float64
dtypes: bool(1), category(6), datetime64[ns](2), float64(8), int64(2), object(4)
memory usage: 15.8+ MB
```

In [230…

```python
# rename 'CreditScoreRangeLower' as CreditScore
listing_borrower.rename(columns = {'CreditScoreRangeLower' : 'CreditScore'}, inplace = T
```

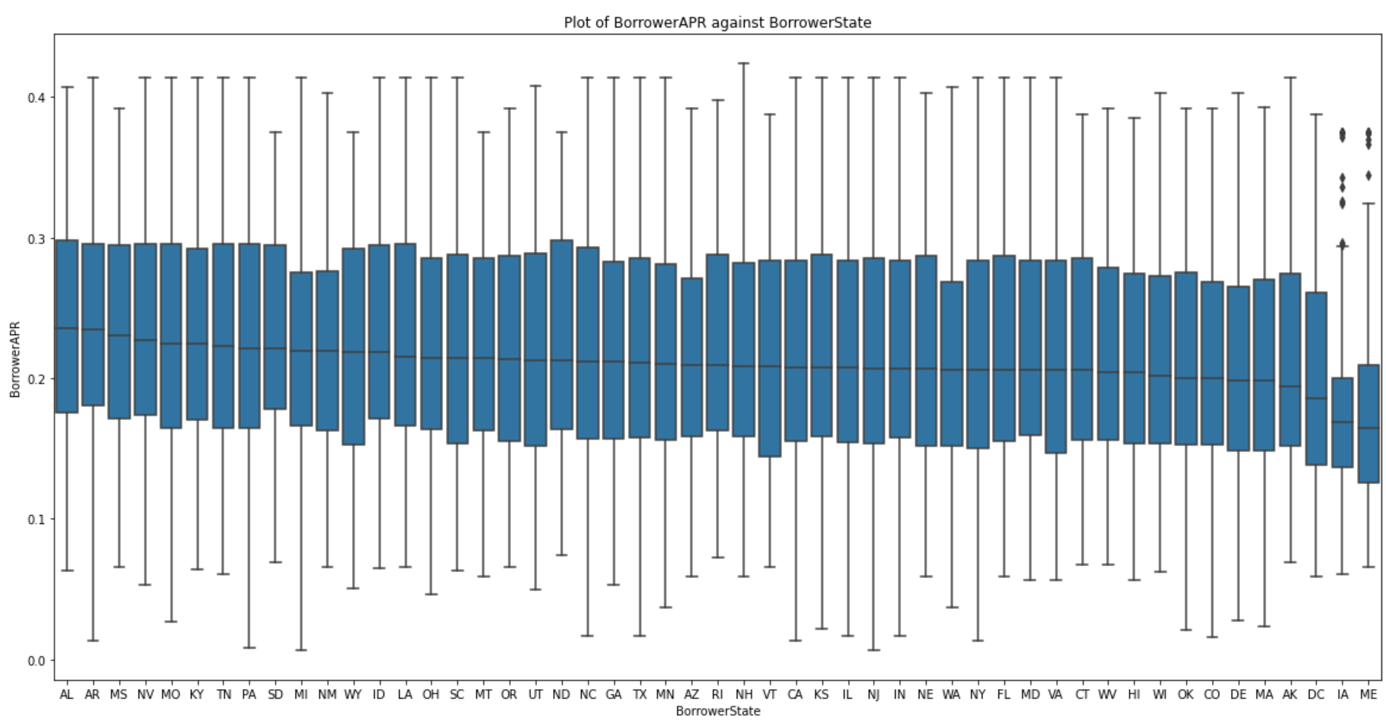
In [231…

```python
def quant_qualplotter(df, target_feature, variable, rotation =90, width = 10, height = 1

    '''Plots the bivariate plts of 1 quantitative and 1 qulitative variable '''

    # Plotting the boxplot
    plt.figure(figsize = [width, height])
    borrower_state_order = df.groupby(variable)[target_feature].median().sort_values(asc
    g = kind(data = df, y = target_feature, x = variable, color = default_color, order =
    plt.xticks(rotation = rotation)
    plt.title('Plot of {} against {}'.format(target_feature, variable));
```

BorrowerAPR VS BorrowerState

In [232…

```python
quant_qualplotter(listing_borrower, 'BorrowerAPR', 'BorrowerState', width =20, rotation
```

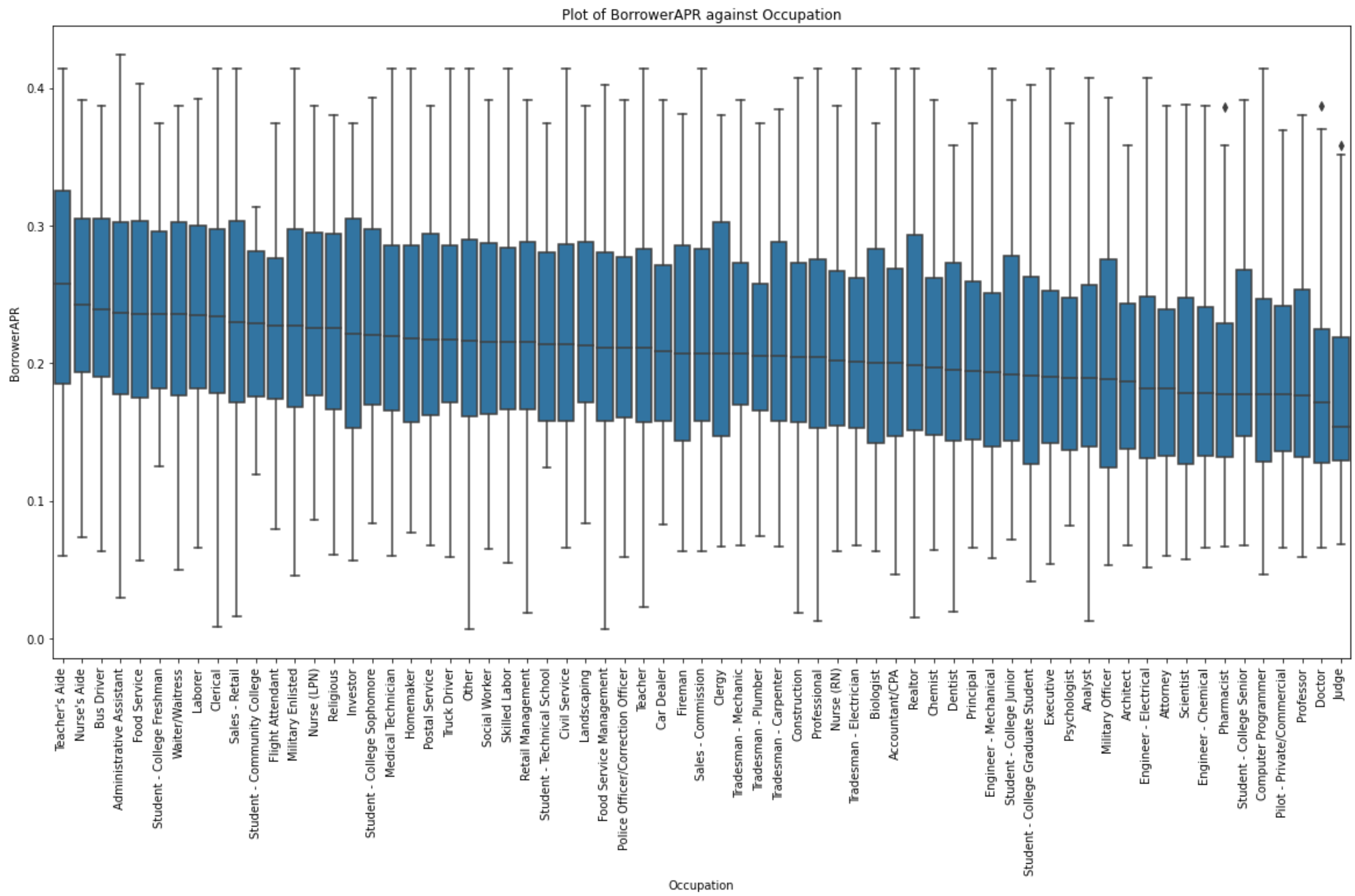


In [ ]:

BorrowerAPR and Occupation

In [233…

```python
quant_qualplotter(listing_borrower, 'BorrowerAPR', 'Occupation', width =20)
```

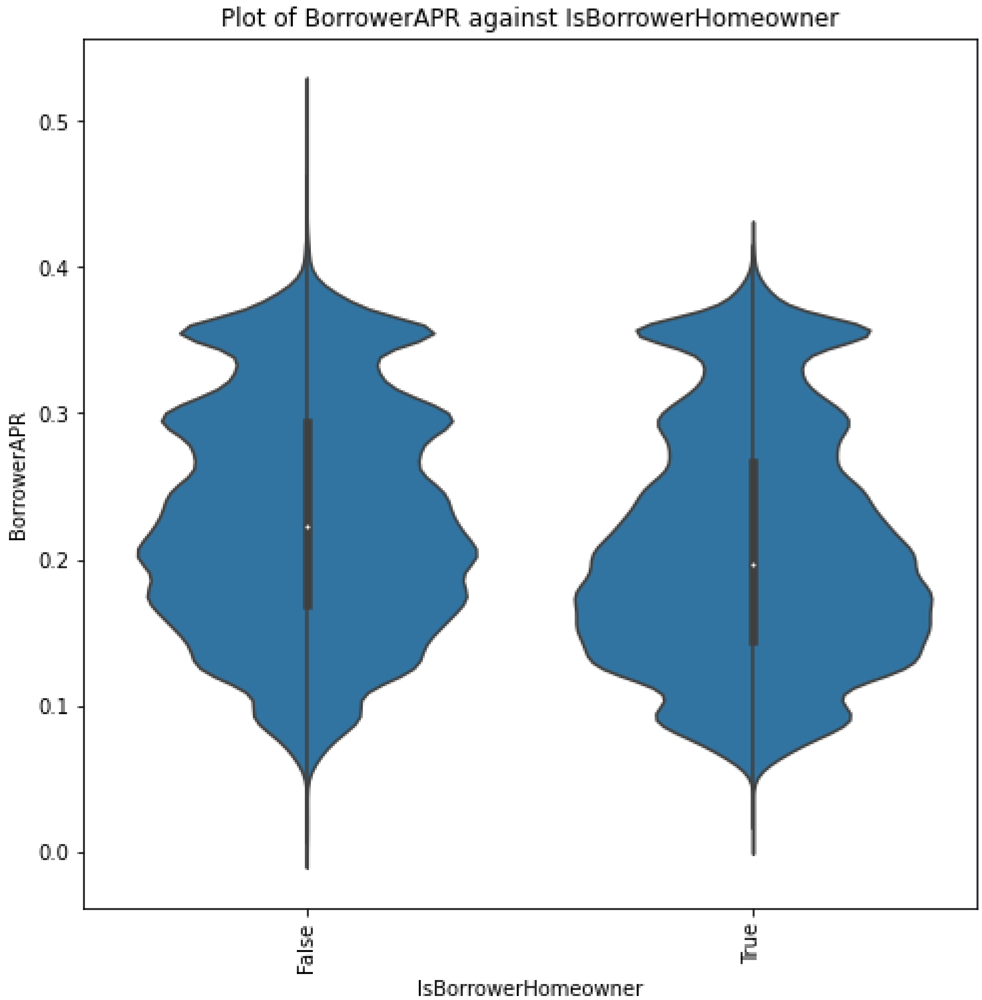


Observation

- This is as expected, Those with lower or unskilled labour occupation types tends to pay a higher APR as compared to semi-skilled and skilled labor
- The higher the quality of your Skill the lesser you tend to pay in servicing your loan.

BorrowerAPR and IsBorrowerHomeowner

In [234…
```
quant_qualplotter(listing_borrower, 'BorrowerAPR', 'IsBorrowerHomeowner', width =8, heig
```



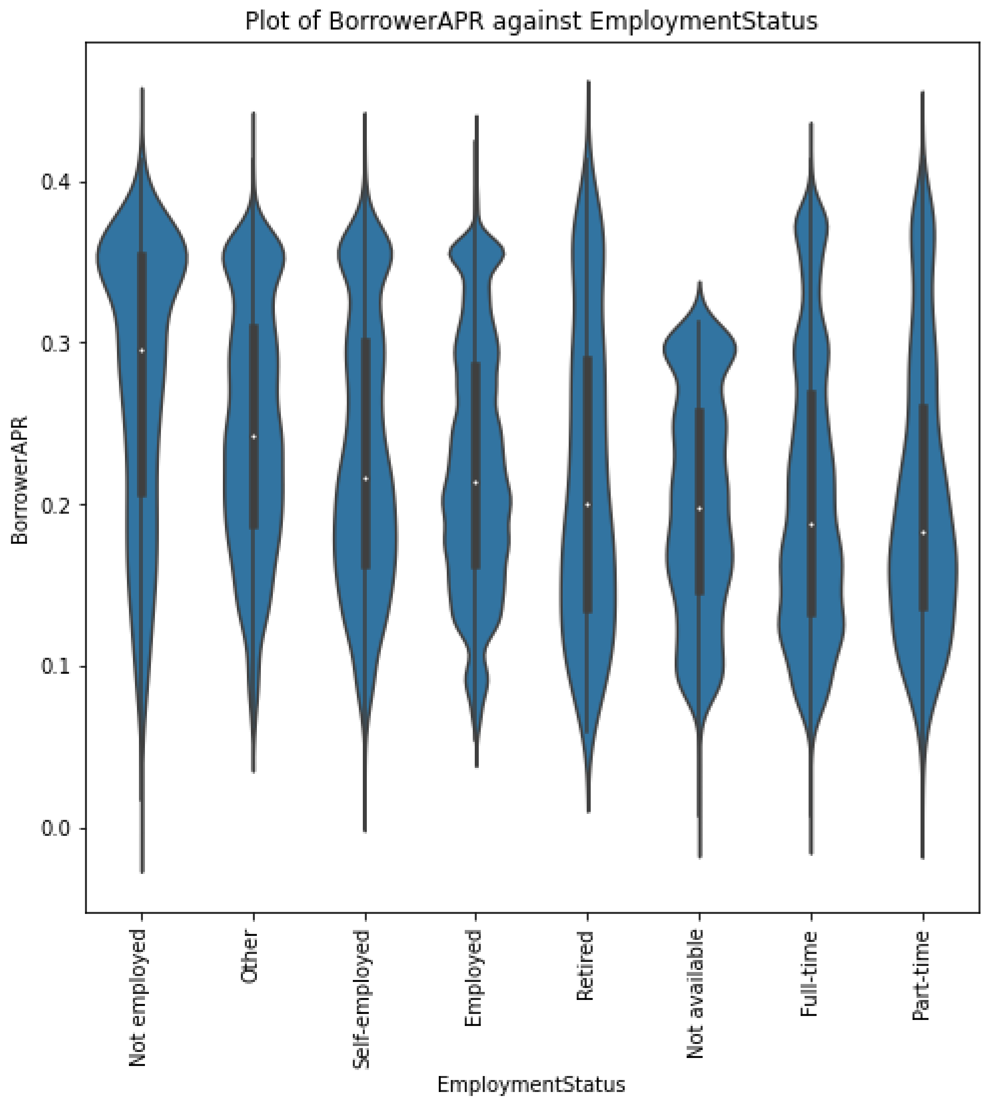
Observation

- This is as expected. A secured loan tend to be cheaper than an unsecured loan due to the reduced risk exposure.
- It can also be observed that the distribution is wider on the lower end of the violinplot.

BorrowerAPR and EmploymentStatus

In [235…

```
quant_qualplotter(listing_borrower, 'BorrowerAPR', 'EmploymentStatus', width =8, height
```



Observation

- This plot shows that those who are unemployed tend to have higher APR
- Retired people tend to have moderately lower APR. I suspect that this is because they tend to have better CreditScore value since they are more experienced in Loan taking.
- we can also observe large wiskers showing that the borrower APR distribution for each level inthe EmploymentStatus cover large ranges. This will be due to several variables acting to decide what APR rate a Borrower will get.

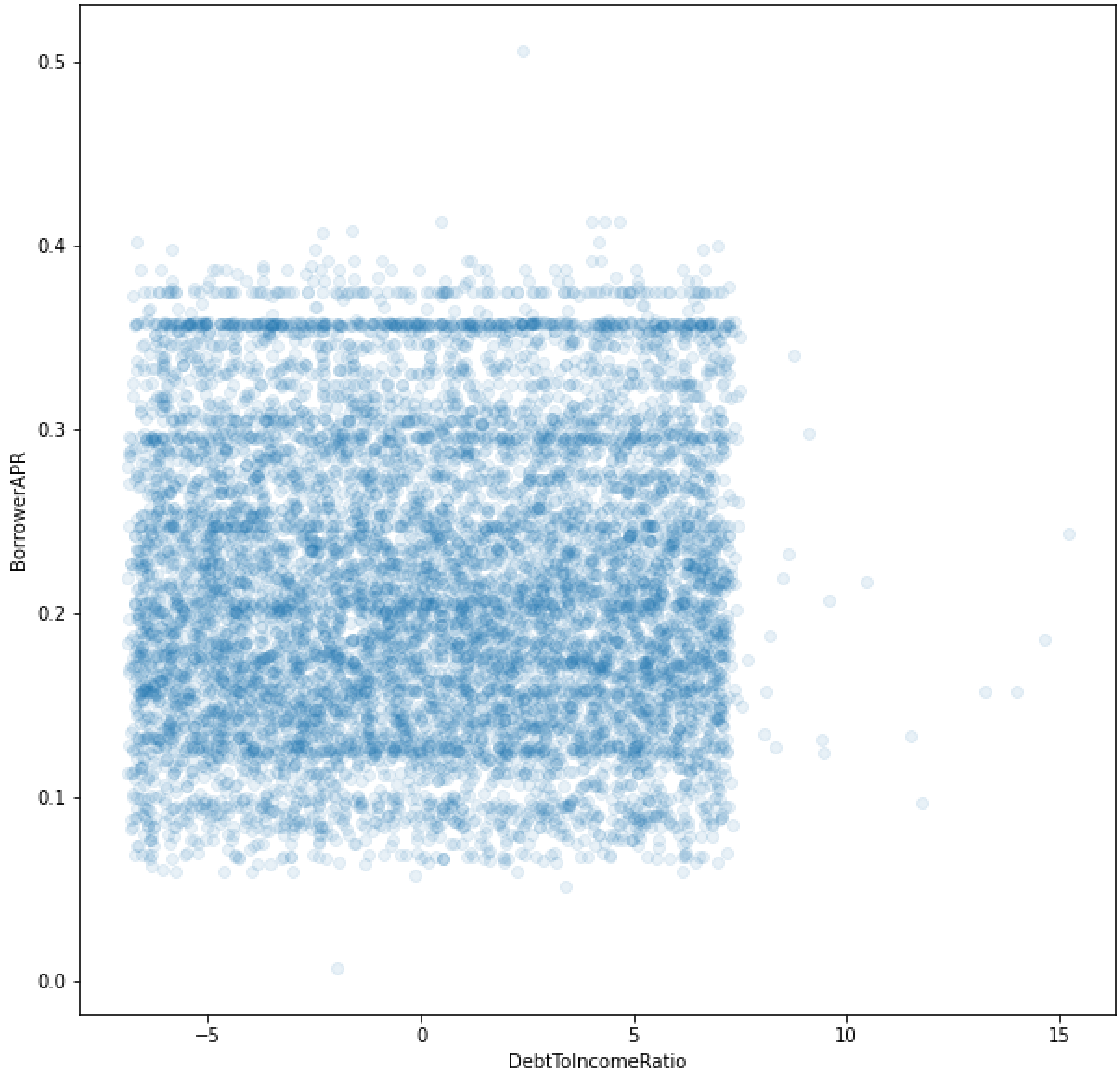
BorrowerAPR and DebtToIncomeRatio

In [236…

```
plt.figure(figsize = (10,10))
listing_borrower_sample = listing_borrower.sample(10000, replace = False)
sb.regplot(data = listing_borrower_sample, x = 'DebtToIncomeRatio', y = 'BorrowerAPR', x
```

Out[236]:

```
<AxesSubplot:xlabel='DebtToIncomeRatio', ylabel='BorrowerAPR'>
```



Observation

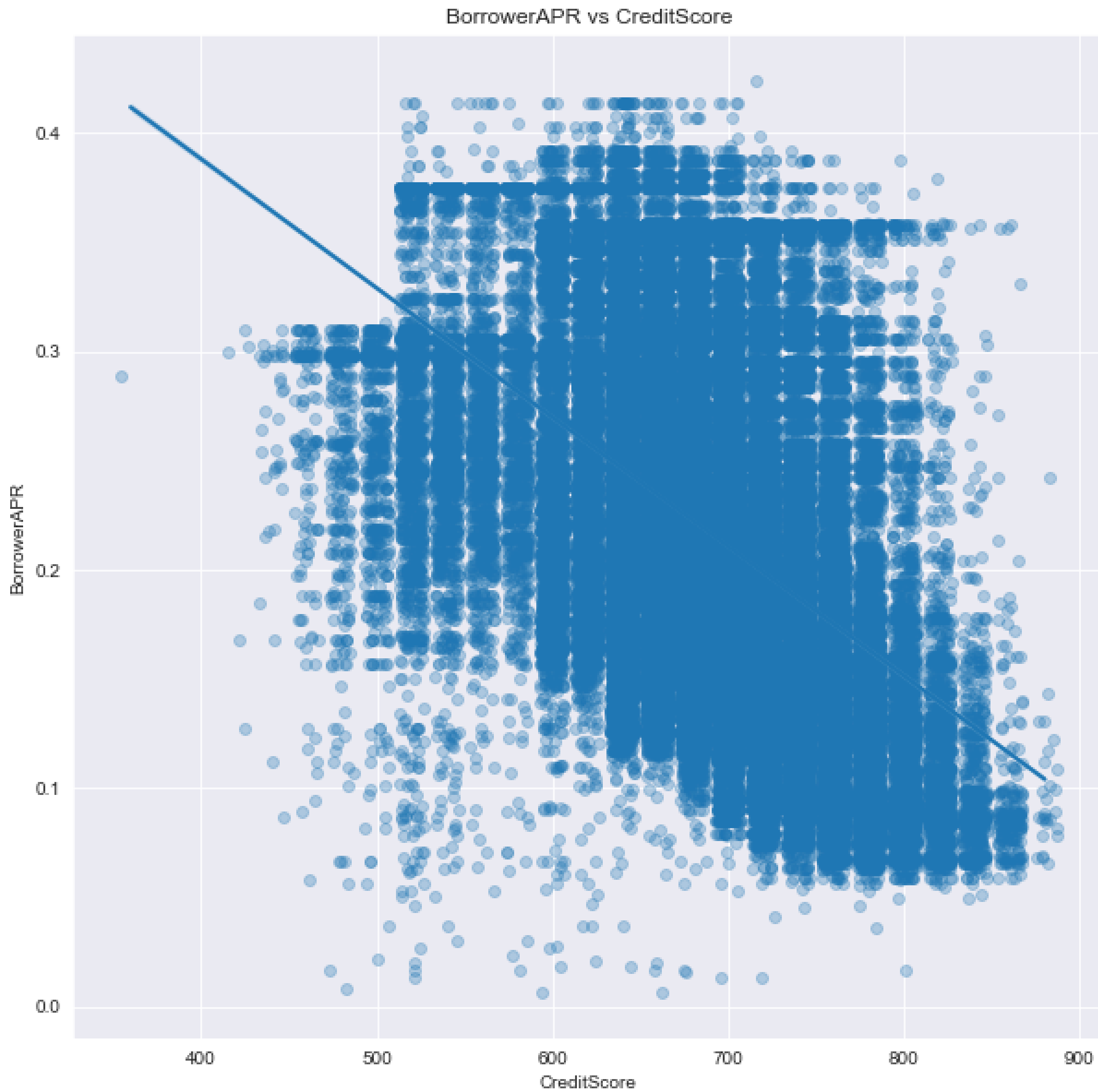
- No clear pattern can be observed

CreditScore and other features

BorrowerAPR and CreditScore

In [255…

```
plt.figure(figsize = (10,10))
sb.regplot(data = listing_borrower, x = 'CreditScore', y = 'BorrowerAPR',
           x_jitter = 7, scatter_kws = {'alpha':0.3});
plt.title('BorrowerAPR vs CreditScore');
```



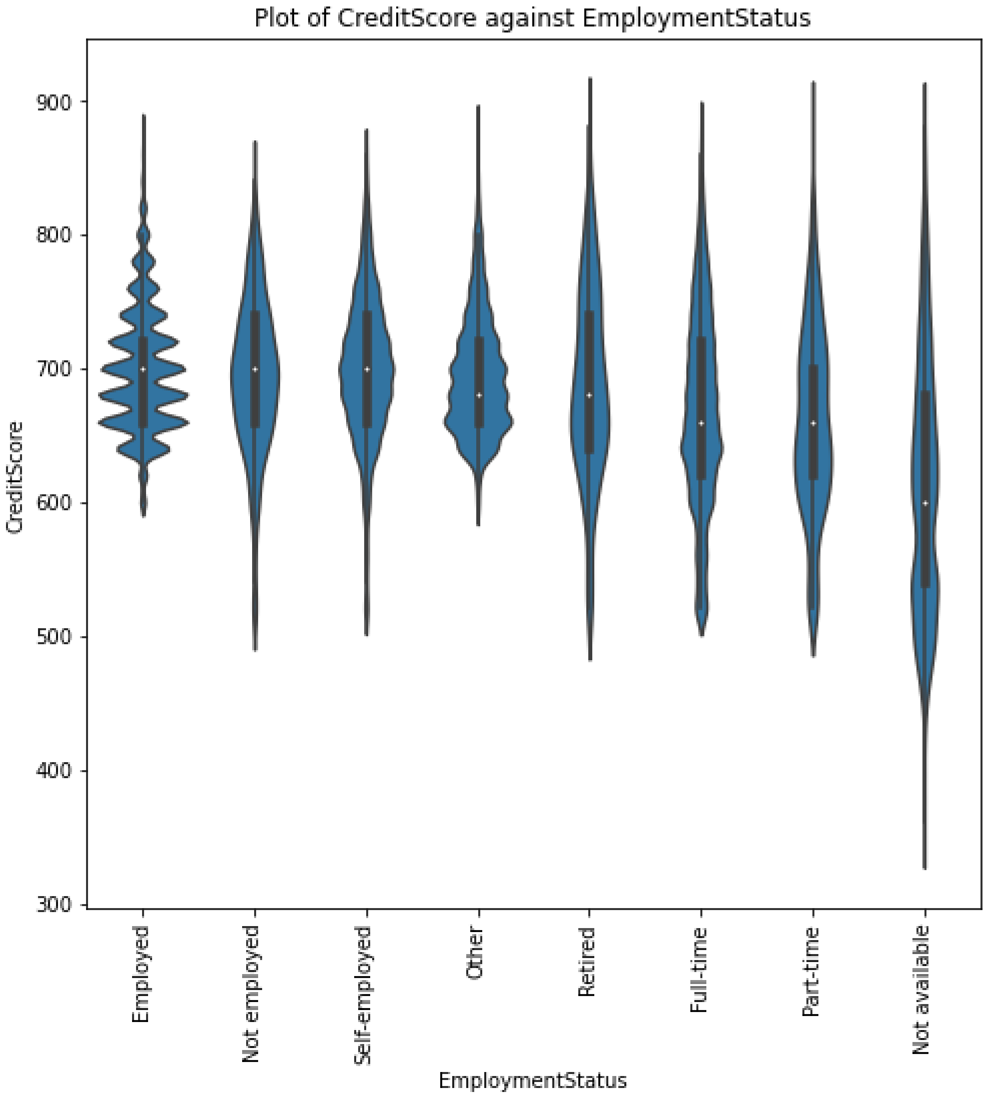
Observation

- The trend is that the BorrowerAPR tend to reduce with incresing CreditScore value.This is as expected.

CreditScore vs EmploymentStatus

In [238…

```
quant_qualplotter(listing_borrower, 'CreditScore', 'EmploymentStatus', width =8, height
```

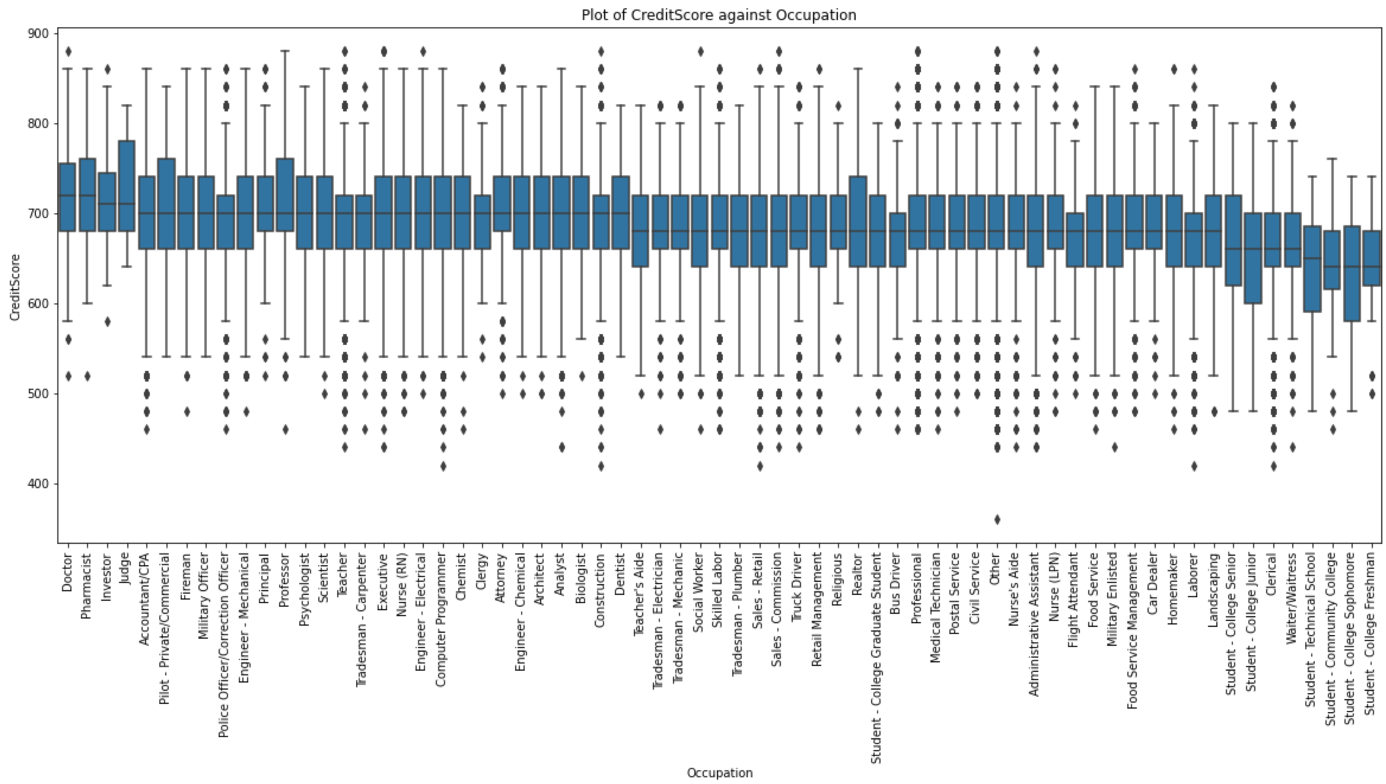


Observation

- The median credit score typically ranges between 600 and 700 across the EmploymentStatus Level

In [239…

```
quant_qualplotter(listing_borrower, 'CreditScore', 'Occupation', width =20, height =8)
```



In [ ]:

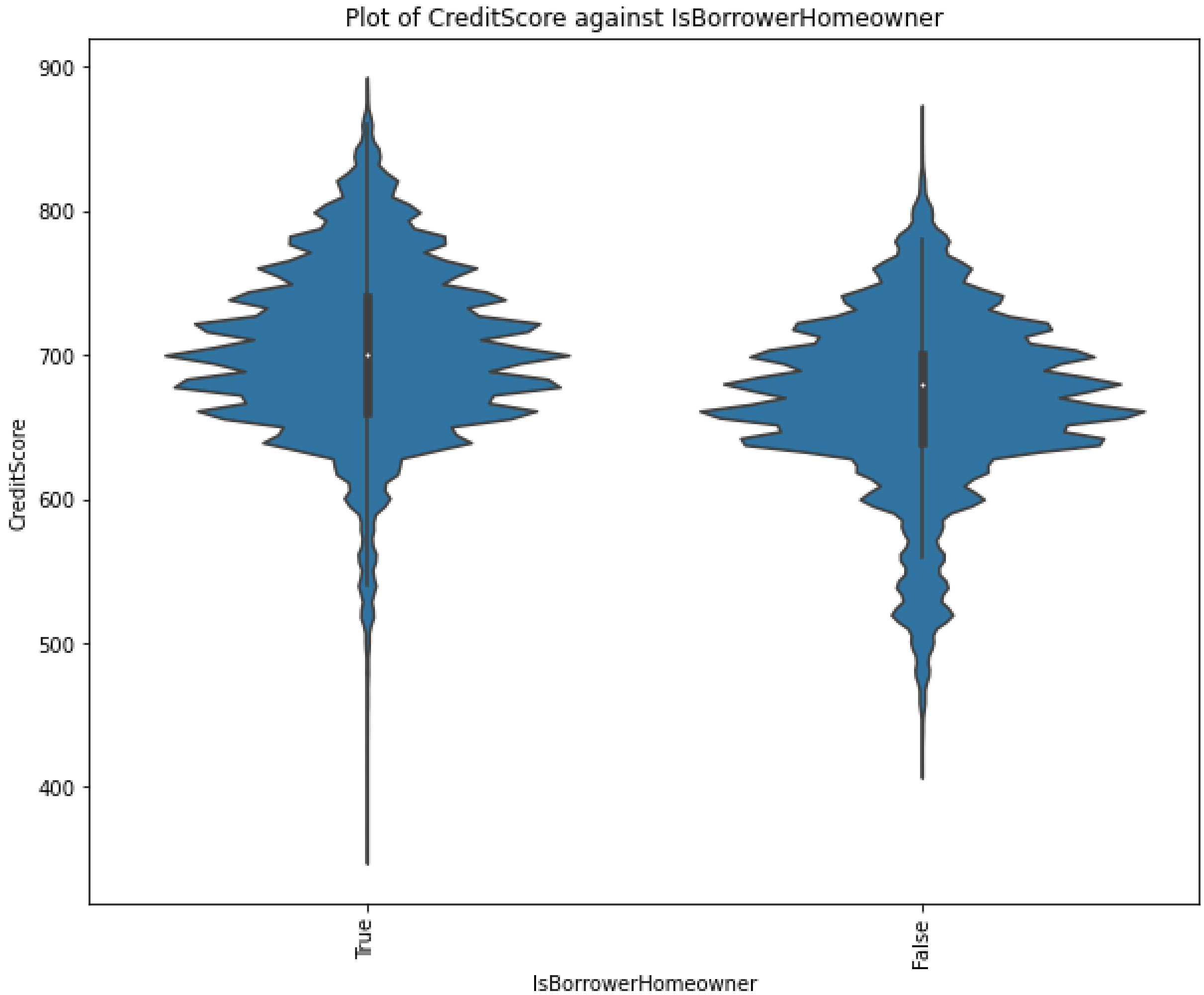
Observation

- The CreditScore shows a pattern where certain group of occupation have similar credit score median value.
```
- Doctors and pharmacist
- Investors and Judges
- Accountant Through to dentist
- Teachers aid to Landscaping
- Senior college students to Waiter/Waitress
- Technical school students to  College Freshman.
```
- The credit score median value shows a reducing trend in this order.

In [ ]:

CreditScore vs IsBorrowerHomeowner

In [240…
```
quant_qualplotter(listing_borrower, 'CreditScore', 'IsBorrowerHomeowner', width =10, hei
```



In [ ]:

Observation

- The CreditScore has the same distribution for the two levels of IsBorrowerHomeowner column
- The Creditscore value for those who are homeowners is generaly higher than for those who are not homeowner

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?

> Overall, most of the Relationships are as expected. The highlight of the Investigation so far, are as follows:

- As the BorrowerAPR (the feature of interest) increases, the Lender's Yield increases. i.e The more overall payment the borrower pays on the loan, the more interest the lender makes on his money.

- As the BorrowerAPR increases, the EstimatedLoss also increases. i.e The higher the APR, the higher the amount of the lender's money at risk of being lost in the event of charge-offs.

- With improving CreditGrade, ProsperRating and ProsperScore ratings, the BorrowerAPR reduces. This means that the BorrowerAPR is associated with higher risk.

- Also, The LoanStatus (another feature of interest), shows that there are more occurences of default and chargeoffs amongst loans of lower or undesirable risk rating.

- Finally, The relationship between several borrowers_profile variables were observed with respect to BorrowerAPR.

- Credit score is a major borrower profile for consideration for loans. It can be observed that with increased credit score There is a reduction in BorrowerAPR.

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

> A little more focus on the Creditscore and other borrowers_profile variables reveals the following information:
>
> - CreditScore shows a clear relationship with employment status.
> ```
>   - The credit score median value is rather the same for employed,
> unemployed and self employed. Although, the outliers are all towards
> higher creditscore value for employed While for unemployed and self
> employed, they are more distributed around the median value
>   - The creditscore also shows a strong relationship with occupation as
> it groups them into different category,mostly based on skill level.
> ```
> - Return

Multivariate Exploration

Here, I will be exploring the data set further, with the aim to see how various borrowers profile variable interact to affect BorrowerAPR. The major question on my mind is, What are the combination of `borrowers_profile` status that indicates **lower risk** and **higher BorrowerAPR** ?

major headings

- BorrowerAPR, CreditScore and EmploymentStatus
- BorrowerAPR, CreditScore, EmploymentStatus and IsBorrowerHomeowner status
- BorrowerAPR, CreditScore and Occupation
- BorrowerAPR, CreditScore, Occupation and IsBorrowerHomeowner
- BorrowerAPR, CreditScore, Occupation, IsBorrowerHomeowner and Employmentstatus
- Discussion
    - Home

BorrowerAPR, CreditScore and EmploymentStatus

In [241…

```
# Filtering out non descriptive values from the EmploymentStatus i.e. Other and Not avai
listing_borrower_filtered = listing_borrower[~(listing_borrower['EmploymentStatus'].isin
```

In [242…

```
listing_borrower[['EmploymentStatus', 'Occupation']][listing_borrower['EmploymentStatus'
```

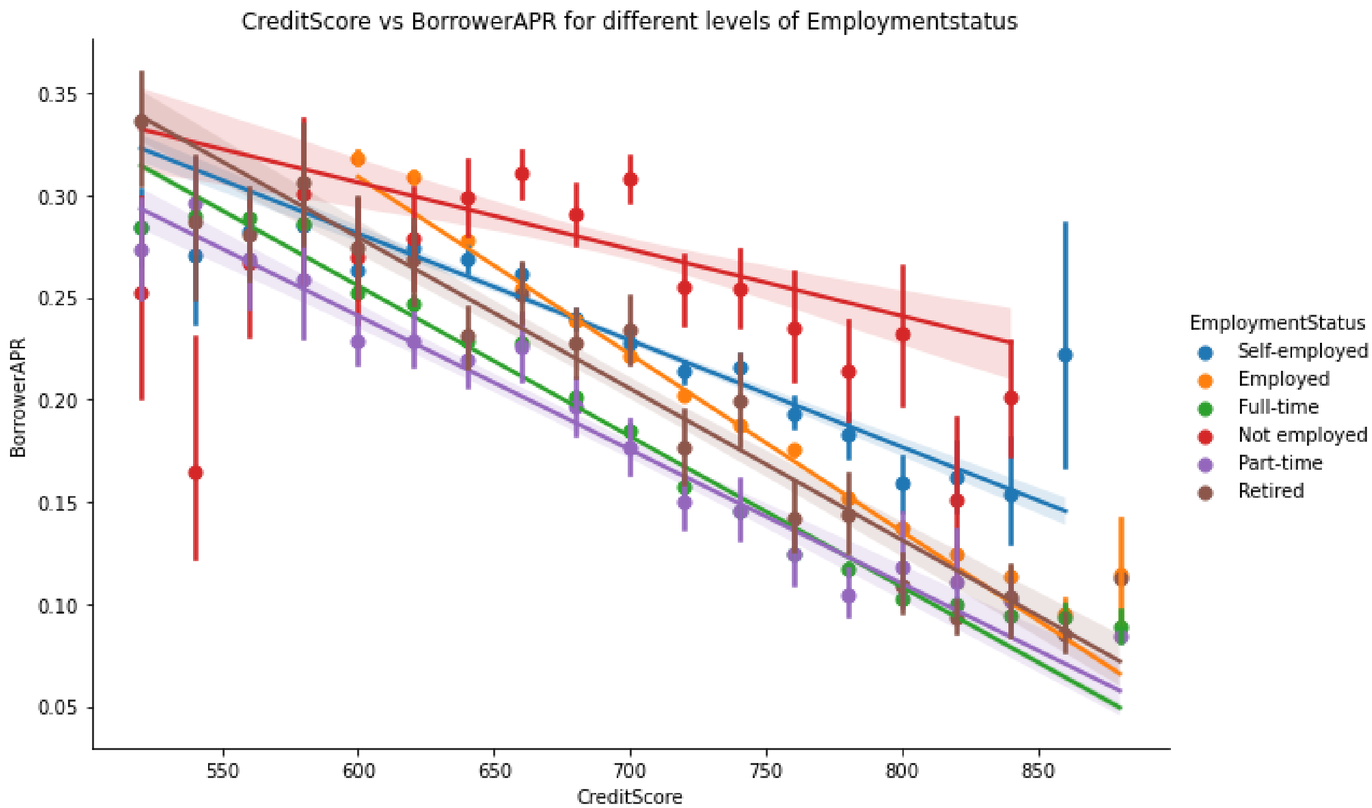
Out[242]:

```
EmploymentStatus  Occupation
Other             Other                 2479
Not available     Other                 1503
                  Professional           589
                  Clerical               293
```

```
                     Computer Programmer          258
                                                 ...
                     Biologist                      3
Other                Computer Programmer            2
                     Accountant/CPA                 1
                     Administrative Assistant       1
                     Judge                          1
Length: 69, dtype: int64
```

In [243…

```
# The multivariate plot of BorrowerAPR, CreditScore and EmploymentStatus
ax = sb.lmplot(data= listing_borrower_filtered, x ='CreditScore', y = 'BorrowerAPR', hue
               x_estimator = np.mean, height = 6, aspect = 1.5)
plt.title('CreditScore vs BorrowerAPR for different levels of Employmentstatus');
```



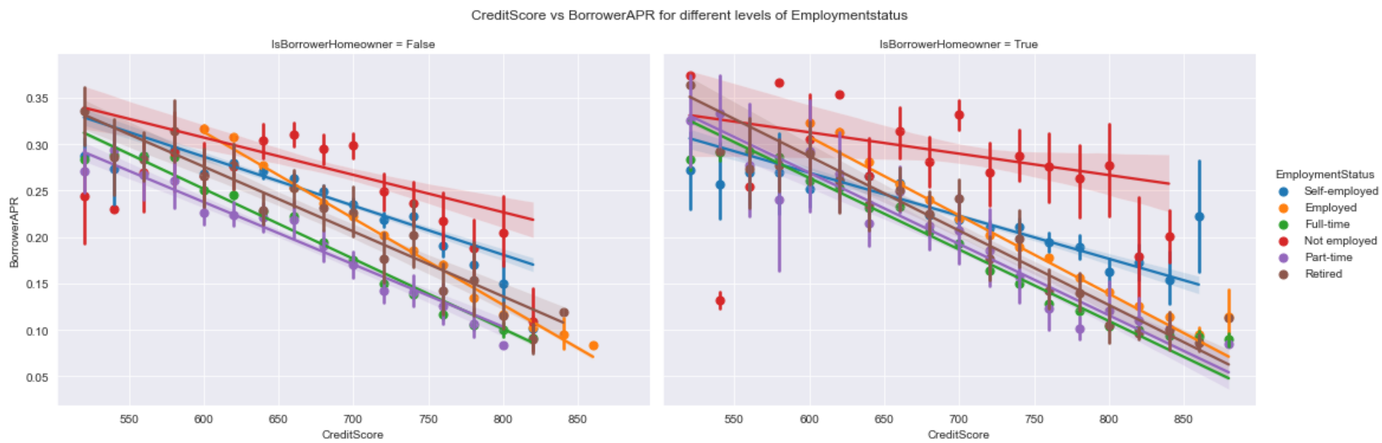
Observations

- Overall, **Not employed** showed the highest BorrowerAPR across the entire CreditScore values. It is obvious that the outliers at lower CreditScore values had the effect of pulling the regression line downwards else, the borrowerAPR would have been much higher at lower CreditScore ranges.
- Employed, Not employed and retired at lower credit score ranges (< 600) have the highest BorrowerAPR. this is indicative of higher risk although it also means higher potential yield.
- At the intermediate CreditScore values (600< CreditScore > 750), Employed and Self-employed have the highest BorrowerAPR.This is without respect to the Not employed category that has the highest BorrowerAPR across board.
- At the highest CreditScore values (> 750), the self employed and the retired has the highest BorrowerAPR. Again, the Not employed category is not being considered.

BorrowerAPR, CreditScore and EmploymentStatus further categorized by their IsBorrowerHomeowner status

In [244…

```
# The multivariate plot of BorrowerAPR, CreditScore and EmploymentStatus
#further categorized by IsBorrowerHomeowner
sb.set_style('darkgrid')
ax = sb.lmplot(data= listing_borrower_filtered, x ='CreditScore', y = 'BorrowerAPR',
               hue ='EmploymentStatus',col = 'IsBorrowerHomeowner',
               x_estimator = np.mean, height = 5, aspect = 1.5)
ax.fig.suptitle('CreditScore vs BorrowerAPR for different levels of Employmentstatus', y
```



Observation

- Generally, at lower creditscore (< 500) homeowners tend to have higher BorrowerAPR as compared to those with similar creditscore who are not homeowners. This might be because of the way prosper loan defines homeowner. You are a homeowner if you have a mortgage on your credit profile or provide a documentation that proves you own a house. However a mortgage can greatly increase your debt burden.

- Amongst the homeowner category, there is less intersection across the range of creditscore. especially amongst the Employed, Retired, Part-time and Full-time group.

- For borrowers who are homeowners, Employed have the highest BorrowerAPR across the range of creditscore value followed by Retired, then Part-Time and then Full-time.

- The **Not employed** shows the greatest variablity. This is more pronounced at lower CreditScore values especially, when they have a home.

- The Self employed with a home have the lowest BorrowerAPR value at lower creditscore ranges in comparison with other category with home. At the higher end of the Creditscore range their APR value moves upward slightly, making their regression line flatter overall.

- The employed have a lower cap for their creditscore value at 600. This is way better than other categories that extend well below 500.

- overall, the group of borrowers who are homeowners have a wider range of credit score, extending well into the 900 range.

BorrowerAPR, CreditScore and Occupation.

- It will be too cumbersome to look at the occupation directly due to the large amount of occupation listed in the occupation column.

- However, in the bivariate section of these analysis, I have already proven that certain groups of occupation have similar Creditscore value. I will take advantage of this information to further categorise the occupation variable.
- A close look at the clusteres formed already inform that the credit score is with respect to their skill levels.Therefore, I have categorised them into 5 groups namely, **Highly Skilled**, **Skilled**, **Semiskilled**, **Unskilled** and **Students**

Code

In [245…

```
# Creating lists of each group of occupation
occupation_skill_median = listing_borrower.groupby('Occupation')['CreditScore'].median()

Highly_Skilled_A = ['Doctor', 'Pharmacist']
Highly_Skilled_B = ['Investor','Judge']
Skilled = occupation_skill_median['Accountant/CPA':'Dentist'].index.tolist()
Semi_skilled =  occupation_skill_median["Teacher's Aide":'Landscaping'].index.tolist()
Unskilled = occupation_skill_median["Student - College Senior":'Waiter/Waitress'].index.
Students = occupation_skill_median["Student - Technical School":'Student - College Fresh
```

In [246…

```
# Creating a new column and filling with appropriate values by occupation status.
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Highly_Skille
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Highly_Skille
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Skilled), 'Oc
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Semi_skilled)
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Unskilled), '
listing_borrower_filtered.loc[listing_borrower_filtered['Occupation'].isin(Students), 'O
```

In [247…

```
# convert to categorical variable
OccupationStatus_list = ['Highly_Skilled_A', 'Highly_Skilled_B', 'Skilled', 'Semi_skille
listing_borrower_filtered['Occupation_status'] = ordered_class(OccupationStatus_list, li
```

In [248…

```
Semi_skilled
```

Out[248]:

```
["Teacher's Aide",
 'Tradesman - Electrician',
 'Tradesman - Mechanic',
 'Social Worker',
 'Skilled Labor',
 'Tradesman - Plumber',
 'Sales - Retail',
 'Sales - Commission',
 'Truck Driver',
 'Retail Management',
 'Religious',
 'Realtor',
 'Student - College Graduate Student',
 'Bus Driver',
 'Professional',
 'Medical Technician',
 'Postal Service',
 'Civil Service',
 'Other',
 "Nurse's Aide",
 'Administrative Assistant',
 'Nurse (LPN)',
 'Flight Attendant',
 'Food Service',
 'Military Enlisted',
 'Food Service Management',
 'Car Dealer',
```

```
 'Homemaker',
 'Laborer',
 'Landscaping']
```

Test

In [249…

```
# Check to see that all observations has been captured
listing_borrower_filtered['Occupation_status'].isnull().sum()
```

Out[249]:

```
2224
```

In [250…

```
# confirmed that all available observations has been captured
listing_borrower_filtered['Occupation'].isnull().sum()
```

Out[250]:

```
2224
```

In [251…

```
#Multivariate plot of BorrowerAPR, CreditScore and Occupation
ax = sb.lmplot(data= listing_borrower_filtered, x ='CreditScore', y = 'BorrowerAPR', hue
                x_estimator = np.mean, height = 6, aspect = 1.5)
plt.title('CreditScore vs BorrowerAPR for different levels of Occupation_status');
```

Observation

1. At lower CreditScore values(< 500), skilled and semiskilled tend to have very high BorrowerAPR values. meaning that they are very risky.

2. At lower CreditScore values(< 500), Unskilled and students tend to have lower APR, however,the confidence interval is wider, indicating a lot of variability and uncertainty.

3. At intermediate CreditScore values(600>CreditScore<720), there is a strong intersection amongst all the group. suggesting that they are all within range for relatively similar BorrowerAPR values. Although, the Highly_Skilled_B group tend to have a higher BorrowerAPR of all the groups. This is clearly due to the outlier point at the top right corner of the plot page.

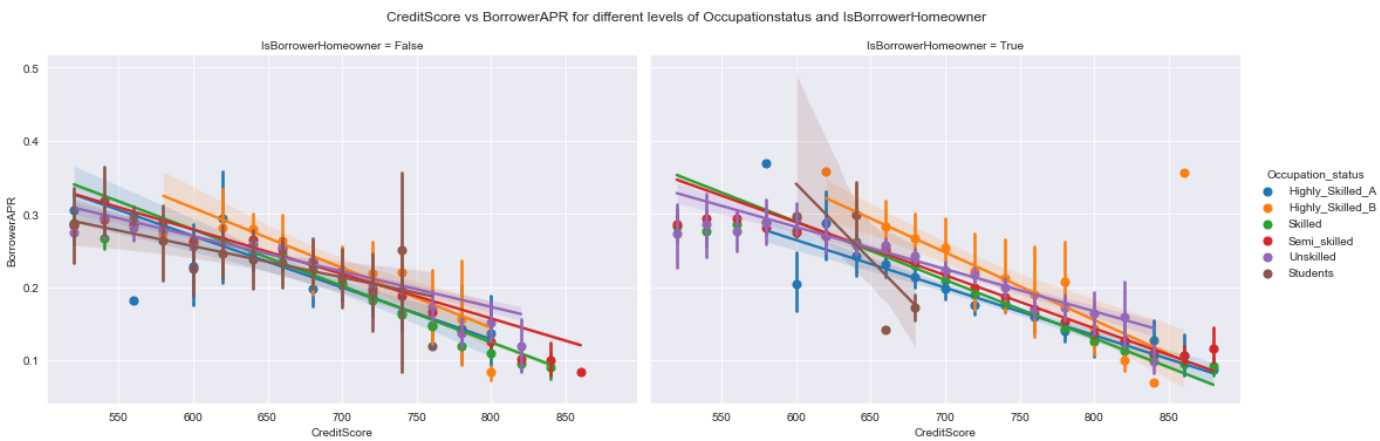
4. At higher CreditScore value (> 700), Unskilled and Semi_skilled tend to have higher BorrowerAPR values, Highly_Skilled_B now have lower APR than the two previous groups while, Skilled and Highly_Skilled_A now have the lowest APR of the two groups.

5. The intermediate CreditScore region will make for the best APR mix with moderate risk.

BorrowerAPR, CreditScore, Occupation and IsBorrowerHomeowner

In [252…

```
#Multivariate plot of BorrowerAPR, CreditScore and Occupation
# Further Categorised by IsBorrowerHomeowner
ax = sb.lmplot(data= listing_borrower_filtered, x ='CreditScore', y = 'BorrowerAPR',
               hue ='Occupation_status',col = 'IsBorrowerHomeowner',
               x_estimator = np.mean, height = 5, aspect = 1.5, hue_order = OccupationSt
ax.fig.suptitle('CreditScore vs BorrowerAPR for different levels of Occupationstatus and
```



Observation

1. At lower CreditScore ranges (<650), the HighlySkilled_B group have the highest BorrowerAPR. This is even more pronounced especially for those who own homes. It seem like there might be an explanation apart from the outlier.

```
 - One explanation could be that they are supposed to have a higher CreditScore
value, and a low CreditScore is indicative of some other underlining personal
finance issues.
```

2. At lower CreditScore ranges (<650), Students who are homeowners,have very wide confidence interval indicative of uncertainty.

3. At lower CreditScore ranges (<650), Students and Unskilled who are not homeoweners have lower BorrowerAPR. This is consistent with logic given above.

```
 - having a lower CreditScore is expected for this group so a lower creditscore is
not indicative of risk hence, a lower APR. However, if they are homeowners
(Particularly, the Students), The confidence interval widens significantly.
Indicating a lot of uncartainty.
```

4. At intermediate CreditScore value (650 > CreditScore < 750) there is a convergence indicating that here you have a good mix of high BorrowerAPR with moderate risk for all groups.

5. At higher CreditScore Value (> 750) unskilled and Semi_skilled have the highest BorrowerAPR. However, if Semi_skilled have a home, the BorrowerAPR goes down significantly.

BorrowerAPR, CreditScore, Occupation, IsBorrowerHomeowner and Employmentstatus.

- Here I will be filtering out employment status for '**Employed**' and '**Not employed**' to focus only on those who are employed and stated more specific employment status.i.e, **Self-employed**, **Full-time**, **Part-time** and **Retired**.

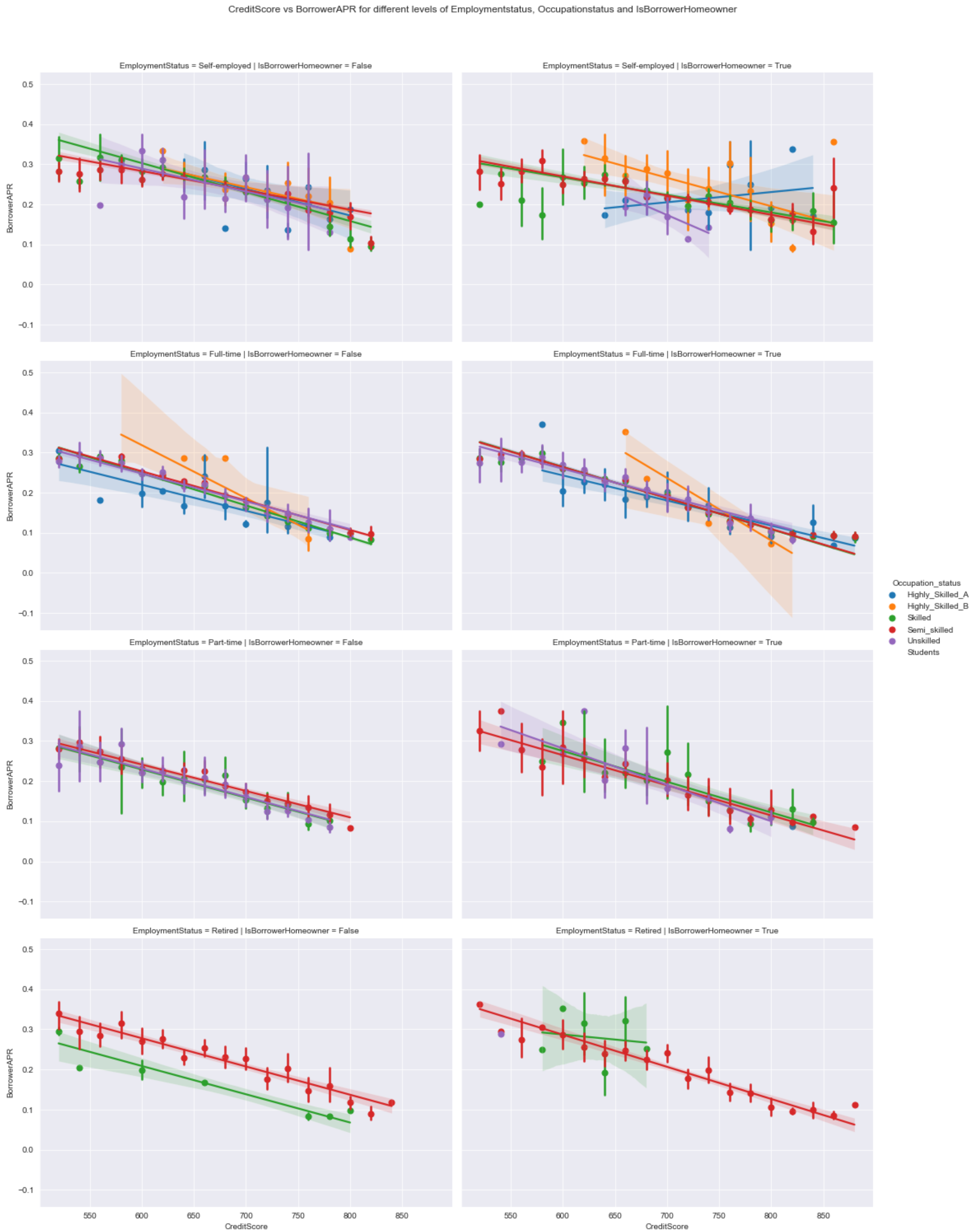
- I will also filter out students from the OccupationStatus because they will not fit into most of the EmploymentStatus category.

In [253…

```
# filtering out Employed and Not-employed from the EmploymentStatus
listing_borrower_subset = listing_borrower_filtered[~(listing_borrower_filtered.Employme

# Filtering out student from the Occupation_Status
listing_borrower_subset = listing_borrower_subset[~(listing_borrower_subset.Occupation_s
```

In [254…

```
#Multivariate plot of BorrowerAPR, CreditScore and Occupation
# Further Categorised by IsBorrowerHomeowner col
# even Further categorised by employmentstatus row
ax = sb.lmplot(data= listing_borrower_subset, x ='CreditScore', y = 'BorrowerAPR',
               hue ='Occupation_status',col = 'IsBorrowerHomeowner', row = 'EmploymentSt
               x_estimator = np.mean, height = 5, aspect = 1.5, hue_order = OccupationSt
ax.fig.suptitle('CreditScore vs BorrowerAPR for different levels of Employmentstatus, Oc
```



Observation

- One striking note is that Those in High_Skilled_A group i.e Doctors and Pharmacist, and those in High_Skilled_B group i.e Investors and Judges, only work Fulltime or are Self-employed.
- The relationship between Skilled, Semi_skilled and Unskilled labor group can also be observed across the EmploymentStatus axis.

    - It can be observed that Skilled people tend to have lower borrowerAPR as compared to Semi_skilled and Unskilled especially at higher Creditscore ranges across the different levels of EmploymentStatus. The difference becomes more pronounced for the retired group.

    - When the extra condition of being homeowners are added (The axis on the right), the diferences between the Skilled and Semi_skilled group seems to close out. with the Semi_skilled sometimes having lower Borrower APR at higher creditScore ranges.

- There are several plots with wide confidence intervals, especially for High_Skilled_A and High_Skilled_B most likely due to not enough datapoint or no clear pattern formation to confidently produce a regression line.

Discussion

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 the previous Section (Bivariate Analysis section), I have clearly established the relationship between BorrowerAPR, LenderYield and Risk. I have also established the relationship between BorrowerAPR and CreditScore.

This section follows from these established relationships to understand the variablity of several borrower_profile variables like EmploymentStatus, Homeownership status (IsBorrowerHomeowner) and Occupation_status in a bid to find an ideal borrowers_profile with high BorrowerAPR and moderate risk.

- The observations can be generally divided into three regions based on CreditScore value.

    - The lower CreditScore region (averagely CreditScore < 600): Associated with higher BorrowerAPR values and higher risk. At this region most levels in the EmploymentStatus and OccupationStatus features are well differentiated into their particular BorrowerAPR score.

        - When the homeownership criteria is introduced the differentiation generally reduces sometimes significantly.
    - The intermediate region (ranging between 600 and 750): This region is associated with a convergence of most of the levels in the EmploymentStatus and Occupation_status. This region is also associated with moderately high BorrowerAPR and relatively lowerRisk.

    - The higher CreditScore region (> 750): This region is associated with differentiated BorrowerAPR value for different levels of the EmploymentStatus and OccupationStatus. The region is always associated with lower BorrowerAPR and low risk.

        - When the homeownership status is added the creditscore range for the different levels of EmploymentStatus and OccupatonStatus increases significantly towards the right.

Were there any interesting or surprising interactions between features?

> The High_Skilled_A group containing Doctor and Pharmacist shows increased BorrowerAPR with increasing CreditScore value when Self_employed. This is is the only group showing such trend and its worthy of further investigation.

- Return

Conclusions

> In this Analysis, I have disected this very large dataset of Prosperloan data containing three differnt observational units into three dataframes and named them as follows to reflect their observation
>
> ```
>     - listing: a concise summary of the loan request
>     - borrower_profile: A profile of the borrower
>     - loan: The credit history of the borrow with the Prosper platform and
> information about the current loan.
> ```
>
> I carried out three levels of analysis on the datasets to reveal more and more details in my quest to answer the question, ''What is the combination of borrower's profile that will yield the most interest for an investor with minimal risk?'.

Univariate analysis: reveals the distribution of various variables in the dataset. **Bivariate analysis**: reveals the relationship that exist between several variables in the dataset. At this level of the analysis, I was able to make the following deductions.

```
     - As the BorrowerAPR increases, the Lender's Yield increases.
     - As the BorrowerAPR increases, the EstimatedLoss also increases
     - With improving CreditGrade, ProsperRating and ProsperScore ratings, the
BorrowerAPR reduces
     - The LoanStatus, shows that there are more occurences of default and
chargeoffs amongst loans of lower or undesirable risk rating.
     - It can be observed that with increased credit score, there is a
reduction in BorrowerAPR.
```

In summary, I reached the conclusion that with increased BorrowerAPR comes increased Yield for the lender which comes at greater risk as well. and also that creditscore of the borrower is a base criteria for determining the level of risk exposure a lender will have in entering into a deal with any borrower.

Multivariate analysis: On the basis of these conclusion I have explored more specific attributes of a lender like EmploymentStatus, OccupationStatus and homeownership status with respect to his CreditScore and BorrowerAPR. The following conclusion were reached at the end of the multivariate analysis.

```
        - CreditScore ranges can be divided into three different regions
showing consistent pattern across the different levels of OccupationStatus and
EmploymentStatus.

        - The lower CreditScore region (averagely CreditScore < 600):
Associated with higher risk and higher BorrowerAPR. Here the OCcupationStatus
and EmploymentStatus show differetiated borrowerAPR

        - The intermediate CreditScore region (ranging between 600 and 750):
Associated with moderate Risk and Yield. The different levels of
OccupationStatus and EmploymentStatus tend to converge at this region.

        - The higher CreditScore region (> 750): Associated with Lower
```

```
BorrowerAPR, lower LenderYield and lower risk. Here also, the different levels
of Employment status and OccupationStatus are differentiated by BorrowerAPR.

        - The homeownership status has the tendencies to reduce the
differentiation between differnt level's BorrowerAPR while also extending the
range of the CreditScore for each level more to the right.
```

Key Take Away: Different Investors have different strategies on the level of risk exposure they are willing to take. This analysis has explored and distilled a systematic approach to profiling a borrower even before delving into further details by first knowing the credit score, then the borrower's EmploymentStatus, Occupation_tatus and home ownership status. This will allow the investor filter through several listing before making further research into the once that passes the profiling test.

- Home

> Remove all Tips mentioned above, before you convert this notebook to PDF/HTML
>
> At the end of your report, make sure that you export the notebook as an html file from the `File > Download as... > HTML or PDF` menu. Make sure you keep track of where the exported file goes, so you can put it in the same folder as this notebook for project submission. Also, make sure you remove all of the quote-formatted guide notes like this one before you finish your report!

In [ ]: