Part I - Loan Data from Prosper Exploration

by Agustin Barto

In [1]:
```
import calendar

from pathlib import Path
from urllib import request
from zipfile import ZipFile

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns  # We've used the traditional namespace instead of
from IPython.display import display, Markdown, HTML

%matplotlib inline
```

Table of Contents

Introduction

I've chosen the "Loan Data from Prosper" dataset as it strikes a good balance between size, complexity and richness. It has a lot of variables to chose from, both numerical and categorical and enough samples to provided meaningful results.

As we'll show in the following section, the data set contains quite a lot of columns to chose from. During the wrangling process we'll briefly explore the dataset to decide which columns are going to be the focus of the analysis. Whenever needed, the column definition will be expanded with external sources.

Preliminary Wrangling

Data dictionary

The following table (converted to CSV from the original Google sheet) contains a brief description of each column:

In [2]:
```
variable_definitions_df = pd.read_csv(
    "https://docs.google.com/spreadsheets/d/1gDyi_L4UvIrLTEC6Wri5nbaMmkGm
)
```

In [3]:
```
styler = variable_definitions_df\
    .style\
    .set_properties(
        **{'text-align': 'left'}
    )\
    .hide(axis="index")
display(HTML(styler.to_html()))
```

Variable	Description
ListingKey	Unique key for each listing, same value as the 'key' used in the listing object in the API.
ListingNumber	The number that uniquely identifies the listing to the public as displayed on the website.
ListingCreationDate	The date the listing was created.
CreditGrade	The Credit rating that was assigned at the time the listing went live. Applicable for listings pre-2009 period and will only be populated for those listings.
Term	The length of the loan expressed in months.
LoanStatus	The current status of the loan: Cancelled, Chargedoff, Completed, Current, Defaulted, FinalPaymentInProgress, PastDue. The PastDue status will be accompanied by a delinquency bucket.
ClosedDate	Closed date is applicable for Cancelled, Completed, Chargedoff and Defaulted loan statuses.
BorrowerAPR	The Borrower's Annual Percentage Rate (APR) for the loan.
BorrowerRate	The Borrower's interest rate for this loan.
LenderYield	The Lender yield on the loan. Lender yield is equal to the interest rate on the loan less the servicing fee.
EstimatedEffectiveYield	Effective yield is equal to the borrower interest rate (i) minus the servicing fee rate, (ii) minus estimated uncollected interest on charge-offs, (iii) plus estimated collected late fees. Applicable for loans originated after July 2009.
EstimatedLoss	Estimated loss is the estimated principal loss on charge-offs. Applicable for loans originated after July 2009.
EstimatedReturn	The estimated return assigned to the listing at the time it was created. Estimated return is the difference between the Estimated Effective Yield and the Estimated Loss Rate. Applicable for loans originated after July 2009.
ProsperRating (numeric)	The Prosper Rating assigned at the time the listing was created: 0 - N/A, 1 - HR, 2 - E, 3 - D, 4 - C, 5 - B, 6 - A, 7 - AA. Applicable for loans originated after July 2009.
ProsperRating (Alpha)	The Prosper Rating assigned at the time the listing was created between AA - HR. Applicable for loans originated after July 2009.
ProsperScore	A custom risk score built using historical Prosper data. The score ranges from 1-10, with 10 being the best, or lowest risk score. Applicable for loans originated after July 2009.
ListingCategory	The category of the listing that the borrower selected when posting their listing: 0 - Not Available, 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 -

Variable	Description
	Motorcycle, 17 - RV, 18 - Taxes, 19 - Vacation, 20 - Wedding Loans
BorrowerState	The two letter abbreviation of the state of the address of the borrower at the time the Listing was created.
Occupation	The Occupation selected by the Borrower at the time they created the listing.
EmploymentStatus	The employment status of the borrower at the time they posted the listing.
EmploymentStatusDuration	The length in months of the employment status at the time the listing was created.
IsBorrowerHomeowner	A Borrower will be classified as a homowner if they have a mortgage on their credit profile or provide documentation confirming they are a homeowner.
CurrentlyInGroup	Specifies whether or not the Borrower was in a group at the time the listing was created.
GroupKey	The Key of the group in which the Borrower is a member of. Value will be null if the borrower does not have a group affiliation.
DateCreditPulled	The date the credit profile was pulled.
CreditScoreRangeLower	The lower value representing the range of the borrower's credit score as provided by a consumer credit rating agency.
CreditScoreRangeUpper	The upper value representing the range of the borrower's credit score as provided by a consumer credit rating agency.
FirstRecordedCreditLine	The date the first credit line was opened.
CurrentCreditLines	Number of current credit lines at the time the credit profile was pulled.
OpenCreditLines	Number of open credit lines at the time the credit profile was pulled.
TotalCreditLinespast7years	Number of credit lines in the past seven years at the time the credit profile was pulled.
OpenRevolvingAccounts	Number of open revolving accounts at the time the credit profile was pulled.
OpenRevolvingMonthlyPayment	Monthly payment on revolving accounts at the time the credit profile was pulled.
InquiriesLast6Months	Number of inquiries in the past six months at the time the credit profile was pulled.
TotalInquiries	Total number of inquiries at the time the credit profile was pulled.
CurrentDelinquencies	Number of accounts delinquent at the time the credit profile was pulled.
AmountDelinquent	Dollars delinquent at the time the credit profile was pulled.
DelinquenciesLast7Years	Number of delinquencies in the past 7 years at the time the credit profile was pulled.
PublicRecordsLast10Years	Number of public records in the past 10 years at the time the credit profile was pulled.

Variable	Description
PublicRecordsLast12Months	Number of public records in the past 12 months at the time the credit profile was pulled.
RevolvingCreditBalance	Dollars of revolving credit at the time the credit profile was pulled.
BankcardUtilization	The percentage of available revolving credit that is utilized at the time the credit profile was pulled.
AvailableBankcardCredit	The total available credit via bank card at the time the credit profile was pulled.
TotalTrades	Number of trade lines ever opened at the time the credit profile was pulled.
TradesNeverDelinquent	Number of trades that have never been delinquent at the time the credit profile was pulled.
TradesOpenedLast6Months	Number of trades opened in the last 6 months at the time the credit profile was pulled.
DebtToIncomeRatio	The debt to income ratio of the borrower at the time the credit profile was pulled. This value is Null if the debt to income ratio is not available. This value is capped at 10.01 (any debt to income ratio larger than 1000% will be returned as 1001%).
IncomeRange	The income range of the borrower at the time the listing was created.
IncomeVerifiable	The borrower indicated they have the required documentation to support their income.
StatedMonthlyIncome	The monthly income the borrower stated at the time the listing was created.
LoanKey	Unique key for each loan. This is the same key that is used in the API.
TotalProsperLoans	Number of Prosper loans the borrower at the time they created this listing. This value will be null if the borrower had no prior loans.
TotalProsperPaymentsBilled	Number of on time payments the borrower made on Prosper loans at the time they created this listing. This value will be null if the borrower had no prior loans.
OnTimeProsperPayments	Number of on time payments the borrower had made on Prosper loans at the time they created this listing. This value will be null if the borrower has no prior loans.
ProsperPaymentsLessThanOneMonthLate	Number of payments the borrower made on Prosper loans that were less than one month late at the time they created this listing. This value will be null if the borrower had no prior loans.
ProsperPaymentsOneMonthPlusLate	Number of payments the borrower made on Prosper loans that were greater than one month late at the time they created this listing. This value will be null if the borrower had no prior loans.
ProsperPrincipalBorrowed	Total principal borrowed on Prosper loans at the time the listing was created. This value will be null if the borrower had no prior loans.
ProsperPrincipalOutstanding	Principal outstanding on Prosper loans at the time the listing was created. This value will be null if the borrower had no prior loans.

Variable	Description
ScorexChangeAtTimeOfListing	Borrower's credit score change at the time the credit profile was pulled. This will be the change relative to the borrower's last Prosper loan. This value will be null if the borrower had no prior loans.
LoanCurrentDaysDelinquent	The number of days delinquent.
LoanFirstDefaultedCycleNumber	The cycle the loan was charged off. If the loan has not charged off the value will be null.
LoanMonthsSinceOrigination	Number of months since the loan originated.
LoanNumber	Unique numeric value associated with the loan.
LoanOriginalAmount	The origination amount of the loan.
LoanOriginationDate	The date the loan was originated.
LoanOriginationQuarter	The quarter in which the loan was originated.
MemberKey	The unique key that is associated with the borrower. This is the same identifier that is used in the API member object.
MonthlyLoanPayment	The scheduled monthly loan payment.
LP_CustomerPayments	Pre charge-off cumulative gross payments made by the borrower on the loan. If the loan has charged off, this value will exclude any recoveries.
LP_CustomerPrincipalPayments	Pre charge-off cumulative principal payments made by the borrower on the loan. If the loan has charged off, this value will exclude any recoveries.
LP_InterestandFees	Pre charge-off cumulative interest and fees paid by the borrower. If the loan has charged off, this value will exclude any recoveries.
LP_ServiceFees	Cumulative service fees paid by the investors who have invested in the loan.
LP_CollectionFees	Cumulative collection fees paid by the investors who have invested in the loan.
LP_GrossPrincipalLoss	The gross charged off amount of the loan.
LP_NetPrincipalLoss	The principal that remains uncollected after any recoveries.
LP_NonPrincipalRecoverypayments	The interest and fee component of any recovery payments. The current payment policy applies payments in the following order: Fees, interest, principal.
PercentFunded	Percent the listing was funded.
Recommendations	Number of recommendations the borrower had at the time the listing was created.
InvestmentFromFriendsCount	Number of friends that made an investment in the loan.
InvestmentFromFriendsAmount	Dollar amount of investments that were made by friends.
Investors	The number of investors that funded the loan.

Downloading the dataset

We assume the dataset has been included with the submission, but in case it had to be removed due to size constraints, the following cell checks if the data is available and downloads it if it is not.

In [4]:
```
prosper_loan_data_csv_zip = Path("./prosperLoanData.csv.zip")
```

In [5]:
```
def download_compress_dataset():
    try:
        with request.urlopen("https://s3.amazonaws.com/udacity-hosted-dow
            with ZipFile(prosper_loan_data_csv_zip, "w") as zf:
                with zf.open("prosperLoanData.csv", "w") as g:
                    g.write(f.read())
    except Exception as e:
        display(Markdown(f"> <span style='color: red;'>**Exception raised
        prosper_loan_data_csv_zip.unlink()
```

In [6]:
```
if not prosper_loan_data_csv_zip.exists():
    download_compress_dataset()
```

Loading the dataset

Once the dataset has been downloaded (and compressed), pandas can just read the CSV file straight out of the zip file:

In [7]:
```
prosper_loan_data_df = pd.read_csv(prosper_loan_data_csv_zip)
```

What is the structure of your dataset?

In [8]:
```
prosper_loan_data_df.sample(5)
```

Out[8]:

	ListingKey	ListingNumber	ListingCreationDate	CreditGrade	Te
101507	6D50337052833432199ED7F	46587	2006-10-10 07:58:21.570000000	E	
69591	07883597241291851674006	1091465	2013-12-23 09:08:48.293000000	NaN	
11016	045B3579177052451874F53	781918	2013-05-17 12:55:48.233000000	NaN	
51071	911B3589572874426E4D562	904919	2013-09-13 19:35:18.360000000	NaN	
47734	44CB359794630620 13C4190	1105842	2013-12-23 09:12:27.017000000	NaN	

5 rows × 81 columns

We've used `sample()` instead of `head()` as some features of the dataset might be hidden by a specific ordering of rowsm.

In [9]:
```
prosper_loan_data_df.shape
```

Out[9]:
```
(113937, 81)
```

In [10]:
```
prosper_loan_data_df.info()
```

```
<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
```

The dataset is comprised of 113937 rows (loans) and 81 columns. The purpose of each column is described in the data dictionary. Before we delve into the contents, we need to make sure certain basic data hygiene criteria are met. Are there any duplicate rows?

In [11]:
```
prosper_loan_data_df.duplicated().sum()
```

Out[11]:
```
0
```

Are there multiple records with the same `ListingKey` values?

In [12]:
```
prosper_loan_data_df.ListingKey.duplicated().sum()
```

Out[12]:
```
871
```

There are entries with the same `ListingKey`. Let's explore this records to determine whether we need to drop the duplicates or not.

In [13]:
```
prosper_loan_data_df[
    prosper_loan_data_df.ListingKey.duplicated(keep=False)  # Keep duplic
].sort_values("ListingKey").head(10)
```

Out[13]:

	ListingKey	ListingNumber	ListingCreationDate	CreditGrade	Te
32680	00223594917038064A7C947	998257	2013-11-15 16:58:37.167000000	NaN	
32681	00223594917038064A7C947	998257	2013-11-15 16:58:37.167000000	NaN	
32964	00473590513960687DD308F	941296	2013-10-07 15:47:36.023000000	NaN	
17274	00473590513960687DD308F	941296	2013-10-07 15:47:36.023000000	NaN	
7478	009836046l900952056DB93	1190614	2014-03-02 14:21:39.583000000	NaN	
33220	009836046l900952056DB93	1190614	2014-03-02 14:21:39.583000000	NaN	
27677	01163604029146842E28D9C	1233732	2014-02-25 14:33:39.830000000	NaN	
788	01163604029146842E28D9C	1233732	2014-02-25 14:33:39.830000000	NaN	
27360	014F35910923350802E1B29	930618	2013-09-26 16:44:24.163000000	NaN	
18324	014F35910923350802E1B29	930618	2013-09-26 16:44:24.163000000	NaN	

10 rows × 81 columns

From these sample, we can assume that there are all duplicated entries which can be safely removed. Notice that we've copied the original dataframe onto a new variable `prosper_loan_data_clean_df` to keep the original data as reference.

In [14]:
```
prosper_loan_data_clean_df = prosper_loan_data_df.copy()
```

In [15]:
```
prosper_loan_data_clean_df = prosper_loan_data_clean_df[~prosper_loan_dat
```

In [16]:
```
prosper_loan_data_clean_df.shape
```

Out[16]:
```
(113066, 81)
```

Now we're left with 113066 rows. Additional clean-up might be performed on the next section where we have to analyze the contents of the chosen columns.

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

The initial set of columns of our interest is the following (you can check the data dictionary for an explanation of each column):

- `ListingCreationDate`
- `CreditGrade`
- `Term`
- `LoanStatus`
- `BorrowerAPR`
- `BorrowerRate`
- `LenderYield`
- `ProsperRating (Alpha)`
- `ListingCategory (numeric)`
- `Occupation`
- `EmploymentStatus`
- `EmploymentStatusDuration`
- `IsBorrowerHomeowner`
- `DebtToIncomeRatio`
- `IncomeRange`
- `LoanOriginalAmount`
- `LoanOriginationDate`

This set of columns was chosen with the idea of analyzing the influence of socio-economic and financial factors on multiple aspects of the loans:

- Which are the most popular loan categories?
- How many borrowers are home owners?
- How are the borrower APR, borrower rate and lender yield related to each other?
- How does the employment status, occupation, home ownership, credit grades, income range and debt to income ratio affect the APR?
- Is there a relationship between the loan amount and the income range?
- How did the loans fared over the years. Is the 2008 global crisis represented?
- Are there specific period with higher activity than others?
- Does the income range influence the chances of defaulting?

As expected, other questions might arise during the analysis and limitationos of the dataset might force us to drop some of these.

Before we can proceed with the analysis, we need to explore each column to determine whether its contents or structure are appropriate for our needs, and if they aren't, can they be adapted or do we need to exclude them?

`ListingCreationDate`, `LoanOriginationDate`

In [17]:
```
prosper_loan_data_clean_df[[
    "ListingCreationDate",
    "LoanOriginationDate"
]].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113066 entries, 0 to 113936
Data columns (total 2 columns):
 #   Column               Non-Null Count   Dtype 
---  ------               --------------   ----- 
 0   ListingCreationDate  113066 non-null  object
 1   LoanOriginationDate  113066 non-null  object
dtypes: object(2)
memory usage: 2.6+ MB
```

In [18]:
```
prosper_loan_data_clean_df[[
    "ListingCreationDate",
    "LoanOriginationDate"
]].sample(5)
```

Out[18]:

	ListingCreationDate	LoanOriginationDate
27432	2013-06-22 06:17:47.677000000	2013-07-05 00:00:00
48431	2013-12-30 11:19:56.820000000	2014-01-08 00:00:00
79716	2010-10-01 11:37:47.930000000	2010-10-26 00:00:00
37792	2012-10-27 08:45:42.060000000	2012-11-02 00:00:00
94675	2012-09-14 12:57:44.910000000	2012-09-25 00:00:00

These two columns represent dates, with `ListingCreationDate` in particular also containing time information, but are stored as strings. Before we can continue, we should convert these columns to `datetime64` using pandas' `to_datetime`.

In [19]:
```
prosper_loan_data_clean_df["ListingCreationDate"] = pd.to_datetime(
    prosper_loan_data_clean_df.ListingCreationDate
)
prosper_loan_data_clean_df["LoanOriginationDate"] = pd.to_datetime(
    prosper_loan_data_clean_df.LoanOriginationDate
)
```

In [20]:
```
prosper_loan_data_clean_df[[
    "ListingCreationDate",
    "LoanOriginationDate"
]].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113066 entries, 0 to 113936
Data columns (total 2 columns):
 #   Column               Non-Null Count   Dtype         
---  ------               --------------   -----         
 0   ListingCreationDate  113066 non-null  datetime64[ns]
 1   LoanOriginationDate  113066 non-null  datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 2.6 MB
```

Let's now focus on the contents of these columns.

In [21]:
```
display(HTML(f"""\
<table>
    <thead>
        <tr>
            <th>Column</th>
            <th style="text-align:center">Min</th>
            <th style="text-align:center">Max</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>ListingCreationDate</td>
            <td style="text-align:center">{prosper_loan_data_clean_df.Lis
            <td style="text-align:center">{prosper_loan_data_clean_df.Lis
        </tr>
        <tr>
            <td>LoanOriginationDate</td>
            <td style="text-align:center">{prosper_loan_data_clean_df.Loa
            <td style="text-align:center">{prosper_loan_data_clean_df.Loa
        </tr>
    </tbody>
</table>
"""))
```

Column	Min	Max
ListingCreationDate	2005-11-09 20:44:28.847000	2014-03-10 12:20:53.760000
LoanOriginationDate	2005-11-15 00:00:00	2014-03-12 00:00:00

This means that our dataset goes from November 2005 until March 2014.

`CreditGrade`, `ProsperRating (Alpha)`

According to the data dictionary, `CreditGrade` will only be populated for loans prior to 2009, while `ProsperRating (Alpha)` will be available for deals from 2009 onwards. This situation will make any global analysis on these columns impossible, it might be interesting to analyse each period in isolation or to perform a comparison between the two variables.

In [22]:
```
prosper_loan_data_clean_df.CreditGrade.isna().sum()
```

Out[22]:
```
84113
```

In [23]:
```
prosper_loan_data_clean_df.CreditGrade.value_counts()
```

Out[23]:
```
C     5649
D     5153
B     4389
AA    3509
HR    3508
A     3315
E     3289
NC     141
Name: CreditGrade, dtype: int64
```

According to Prosper's API documentation this is a categorial column with the following scale:

1. NC
2. HR
3. E
4. D
5. C
6. B
7. A
8. AA

In [24]:
```
prosper_loan_data_clean_df["ProsperRating (Alpha)"].isna().sum()
```

Out[24]:
```
29084
```

In [25]:
```
prosper_loan_data_clean_df["ProsperRating (Alpha)"].value_counts()
```

Out[25]:
```
C     18096
B     15368
A     14390
D     14170
E      9716
HR     6917
AA     5325
Name: ProsperRating (Alpha), dtype: int64
```

According to Prosper's API documentation and the following article this is a categorial column with the following scale:

1. NA
2. HR
3. E
4. D
5. C
6. B
7. A
8. AA

Although there seems to be a correspondence between values, we cannot assume these columns are compatible, so they will be kept separate. Given that these are ordinal categorical values, we should convert them accordingly.

In [26]:
```
prosper_loan_data_clean_df["CreditGrade"] = prosper_loan_data_clean_df.Cr
    pd.CategoricalDtype(ordered=True, categories=[
        "NC",
        "HR",
        "E",
        "D",
        "C",
        "B",
        "A",
        "AA"
    ])
)
prosper_loan_data_clean_df["ProsperRating (Alpha)"] = prosper_loan_data_c
    "ProsperRating (Alpha)"
].astype(
    pd.CategoricalDtype(ordered=True, categories=[
        "NA",
        "HR",
        "E",
        "D",
        "C",
        "B",
        "A",
        "AA"
    ])
)
```

In [27]:
```
prosper_loan_data_clean_df[[
    "CreditGrade",
    "ProsperRating (Alpha)"
]].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 113066 entries, 0 to 113936
Data columns (total 2 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   CreditGrade            28953 non-null  category
 1   ProsperRating (Alpha)  83982 non-null  category
dtypes: category(2)
memory usage: 1.1 MB
```

Are there any loands without values for either of these columns?

In [28]:
```
(
    prosper_loan_data_clean_df.CreditGrade.isna()
    & prosper_loan_data_clean_df["ProsperRating (Alpha)"].isna()
).sum()
```

Out[28]: 131

We're to drop these rows as we think these variables are essential for our analysis.

In [29]:
```
prosper_loan_data_clean_df = prosper_loan_data_clean_df[
    (~prosper_loan_data_clean_df.CreditGrade.isna())
    | (~prosper_loan_data_clean_df["ProsperRating (Alpha)"].isna())
]
```

In [30]:
```
(
    prosper_loan_data_clean_df.CreditGrade.isna()
    & prosper_loan_data_clean_df["ProsperRating (Alpha)"].isna()
).sum()
```

Out[30]:
```
0
```

Term

In [31]:
```
prosper_loan_data_clean_df.Term.nunique()
```

Out[31]:
```
3
```

In [32]:
```
prosper_loan_data_clean_df.Term.value_counts()
```

Out[32]:
```
36    87094
60    24228
12     1613
Name: Term, dtype: int64
```

Given that we're only seeing three possible values for the `Term` column, we could be tempted to convert it to a categorical type, but since the documentation clearly states that these are to be interpreted as months, we shouldn't convert it.

In [33]:
```
prosper_loan_data_clean_df.Term.isna().sum()
```

Out[33]:
```
0
```

There are no loans without a `Term` value.

LoanStatus

In [34]:
```
prosper_loan_data_clean_df.LoanStatus.nunique()
```

Out[34]:
```
12
```

In [35]:
```
prosper_loan_data_clean_df.LoanStatus.value_counts()
```

Out[35]:
```
Current                   55730
Completed                 37939
Chargedoff                11986
Defaulted                  5015
Past Due (1-15 days)        800
Past Due (31-60 days)       361
Past Due (61-90 days)       311
Past Due (91-120 days)      304
Past Due (16-30 days)       265
FinalPaymentInProgress      203
Past Due (>120 days)         16
Cancelled                     5
Name: LoanStatus, dtype: int64
```

Are there any loans without a `LoanStatus` value?

In [36]:
```
prosper_loan_data_clean_df.LoanStatus.isna().sum()
```

Out[36]: 0

According to the documentation, this column violates the Tidy Data principle of one variable per column is it contains both the load status and the delinquency bucket:

> The current status of the loan: Cancelled, Chargedoff, Completed, Current, Defaulted, FinalPaymentInProgress, PastDue. The PastDue status will be accompanied by a delinquency bucket.

We're going to convert `LoanStatus` to a categorical type and move the delinquency bucket into its own categorical column named `DelinquencyBucket`.

In [37]:
```
(
    prosper_loan_data_clean_df[
        # Match loans where the LoanStatus column starts with "Past Due"
        prosper_loan_data_clean_df.LoanStatus.str.match(r"^Past Due.*$")
    ]
).LoanStatus.str.extract(
    # Extract the past due period with a capturing group
    r"^Past Due \((.*) days\)$"
).value_counts()
```

Out[37]:
```
1-15      800
31-60     361
61-90     311
91-120    304
16-30     265
>120       16
dtype: int64
```

In [38]:
```
# Create a new DelinquencyBucket string column
prosper_loan_data_clean_df["DelinquencyBucket"] = (
    prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanStatus.str.
).LoanStatus.str.extract(r"^Past Due \((.*) days\)$")
```

In [39]:
```
# Convert the DelinquencyBucket to an ordinal category
prosper_loan_data_clean_df["DelinquencyBucket"] = prosper_loan_data_clean
    pd.CategoricalDtype(
        ordered=True,
        categories=[
            "1-15",
            "16-30",
            "31-60",
            "61-90",
            "91-120",
            ">120"
        ]
    )
)
```

In [40]:
```
prosper_loan_data_clean_df.DelinquencyBucket.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112935 entries, 0 to 113936
Series name: DelinquencyBucket
Non-Null Count  Dtype
--------------  -----
2057 non-null   category
dtypes: category(1)
memory usage: 992.8 KB
```

Now we can remove the delinquency bucket info from the past due loans.

In [41]:
```
# Create a new LoanStatus column extracting the status while dropping the
prosper_loan_data_clean_df["LoanStatus"] = prosper_loan_data_clean_df.Loa
    r"^(Current|Completed|Chargedoff|Defaulted|Past Due|FinalPaymentInPro
)
```

In [42]:
```
prosper_loan_data_clean_df["LoanStatus"].value_counts()
```

Out[42]:
```
Current                   55730
Completed                 37939
Chargedoff                11986
Defaulted                  5015
Past Due                   2057
FinalPaymentInProgress      203
Cancelled                     5
Name: LoanStatus, dtype: int64
```

Notice that the value counts match those shown above. Now we can safely convert the column to a nominal categorical type.

In [43]:
```
prosper_loan_data_clean_df["LoanStatus"] = prosper_loan_data_clean_df.Loa
    pd.CategoricalDtype(
        ordered=False,
        categories=[
            "Current",
            "Completed",
            "Chargedoff",
            "Defaulted",
            "FinalPaymentInProgress",
            "Past Due",
            "Cancelled"
        ]
    )
)
```

In [44]:
```
prosper_loan_data_clean_df.LoanStatus.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112935 entries, 0 to 113936
Series name: LoanStatus
Non-Null Count   Dtype
--------------   -----
112935 non-null  category
dtypes: category(1)
memory usage: 992.9 KB
```

In [45]:
```
prosper_loan_data_clean_df.LoanStatus.isna().sum()
```

Out[45]:
```
0
```

BorrowerAPR, BorrowerRate

According to the data dictionary, `BorrowerAPR` is the Borrower's Annual Percentage Rate (APR) for the loan. while `BorrowerRate` is the borrower's interest rate for this loan. According to this article, there is a clear relationship between this variables, but we'll explore it in detail. Let's see how much data is available for these variables.

In [46]:
```
prosper_loan_data_clean_df.BorrowerAPR.isna().sum()
```

Out[46]:
```
25
```

In [47]:
```
prosper_loan_data_clean_df.BorrowerRate.isna().sum()
```

Out[47]:
```
0
```

We're going to drop rows without a `BorrowerAPR` or `BorrowerRate` values:

In [48]:
```
prosper_loan_data_clean_df = prosper_loan_data_clean_df[
    ~(
        prosper_loan_data_clean_df.BorrowerAPR.isna()
        & prosper_loan_data_clean_df.BorrowerRate.isna()
    )
]
```

In [49]:
```
prosper_loan_data_clean_df.BorrowerAPR.describe()
```

Out[49]:
```
count    112910.000000
mean          0.219018
std           0.080464
min           0.006530
25%           0.156290
50%           0.209860
75%           0.283860
max           0.512290
Name: BorrowerAPR, dtype: float64
```

In [50]:
```
prosper_loan_data_clean_df.BorrowerRate.describe()
```

Out[50]:
```
count    112935.000000
mean          0.192980
std           0.074899
min           0.000000
25%           0.134000
50%           0.184000
75%           0.250600
max           0.497500
Name: BorrowerRate, dtype: float64
```

LenderYield

`LenderYield` is the other side of the coin of `BorrowerAPR` and `BorrowerRate`. Let's see if there's anything to do with this variable.

In [51]:
```
prosper_loan_data_clean_df.LenderYield.isna().sum()
```

Out[51]:
```
0
```

In [52]:
```
prosper_loan_data_clean_df.LenderYield.describe()
```

Out[52]:
```
count    112935.000000
mean          0.182917
std           0.074595
min          -0.010000
25%           0.124600
50%           0.174000
75%           0.240600
max           0.492500
Name: LenderYield, dtype: float64
```

`ListingCategory (numeric)`

According to the dictionary, `ListingCategory (numeric)` is a categorial variable:

> The category of the listing that the borrower selected when posting their listing: 0 - Not Available, 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 - Wedding Loans

In [53]:
```
sorted(prosper_loan_data_clean_df["ListingCategory (numeric)"].unique())
```

Out[53]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0]
```

We're going to convert these numeric categories into their string counterparts to make it easier to perform analysis and visualizations. We'll create a new categorical typed column named `ListingCategory`.

In [54]:
```
listing_categories = [
    "Not Available",       # 0
    "Debt Consolidation",  # 1
    "Home Improvement",    # 2
    "Business",            # 3
    "Personal Loan",       # 4
    "Student Use",         # 5
    "Auto",                # 6
    "Other",               # 7
    "Baby&Adoption",       # 8
    "Boat",                # 9
    "Cosmetic Procedure",  # 10
    "Engagement Ring",     # 11
    "Green Loans",         # 12
    "Household Expenses",  # 13
    "Large Purchases",     # 14
    "Medical/Dental",      # 15
    "Motorcycle",          # 16
    "RV",                  # 17
    "Taxes",               # 18
    "Vacation",            # 19
    "Wedding Loans"        # 20
]

listing_category_value_map = dict(enumerate(listing_categories))
listing_category_value_map
```

Out[54]:
```
{0: 'Not Available',
 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: 'Wedding Loans'}
```

In [55]:
```
# Create a new ListingCategory column using the listing_category_value_ma
prosper_loan_data_clean_df["ListingCategory"] =\
    prosper_loan_data_clean_df["ListingCategory (numeric)"].apply(
        lambda v: listing_category_value_map[v]
    )
```

In [56]:
```
# Convert ListingCategory to a categorical column
prosper_loan_data_clean_df["ListingCategory"] = prosper_loan_data_clean_d
    pd.CategoricalDtype(ordered=False, categories=listing_categories)
)
```

In [57]:
```
prosper_loan_data_clean_df.ListingCategory.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112935 entries, 0 to 113936
Series name: ListingCategory
Non-Null Count   Dtype
--------------   -----
112935 non-null  category
dtypes: category(1)
memory usage: 993.3 KB
```

Occupation

In [58]:
```
prosper_loan_data_clean_df.Occupation.nunique()
```

Out[58]:
```
67
```

In [59]:
```
prosper_loan_data_clean_df.Occupation.value_counts()
```

Out[59]:
```
Other                          28392
Professional                   13508
Computer Programmer             4433
Executive                       4276
Teacher                         3724
                               ...
Dentist                           67
Student - College Freshman        41
Student - Community College       28
Judge                             22
Student - Technical School        16
Name: Occupation, Length: 67, dtype: int64
```

In [60]:
```
prosper_loan_data_clean_df.Occupation.isna().sum()
```

Out[60]:
```
3529
```

There are 3529 rows without an `Occupation` value.

`Ocupation` is a categorical variable and has to be converted as such, but given the number of categories, we'll have to be careful when presenting charts that includes it.

In [61]:
```
prosper_loan_data_clean_df["Occupation"] = prosper_loan_data_clean_df.Occ
    pd.CategoricalDtype(
        ordered=False,
        # Categories sorted alphabetically (but we're not using a sorted
        categories=sorted(
            prosper_loan_data_clean_df.Occupation.dropna().unique()
        )
    )
)
```

EmploymentStatus , EmploymentStatusDuration

In [62]:
```
prosper_loan_data_clean_df.EmploymentStatus.unique()
```

Out[62]:
```
array(['Self-employed', 'Employed', 'Not available', 'Full-time', 'Other
',
       nan, 'Not employed', 'Part-time', 'Retired'], dtype=object)
```

In [63]:
```
prosper_loan_data_clean_df.EmploymentStatus.isna().sum()  # Has missing v
```

Out[63]:
```
2255
```

There are 2255 rows without an `EmploymentStatus` value. Once again, we'll convert this column into a categorical type.

In [64]:
```
prosper_loan_data_clean_df["EmploymentStatus"] = prosper_loan_data_clean_
    pd.CategoricalDtype(
        ordered=False,
        # Categories sorted alphabetically (but we're not using a sorted
        categories=sorted(
            prosper_loan_data_clean_df.EmploymentStatus.dropna().unique()
        )
    )
)
```

Since `EmploymentStatusDuration` is expressed in number of months, we'll convert the column to `Int64`.

In [65]:
```
prosper_loan_data_clean_df["EmploymentStatusDuration"] =\
prosper_loan_data_clean_df.EmploymentStatusDuration.astype("Int64")
```

In [66]:
```
prosper_loan_data_clean_df.EmploymentStatusDuration.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112935 entries, 0 to 113936
Series name: EmploymentStatusDuration
Non-Null Count   Dtype
--------------   -----
105310 non-null  Int64
dtypes: Int64(1)
memory usage: 1.8 MB
```

IsBorrowerHomeowner

In [67]:
```
prosper_loan_data_clean_df.IsBorrowerHomeowner.nunique()
```

Out[67]:
```
2
```

In [68]:
```
prosper_loan_data_clean_df.IsBorrowerHomeowner.value_counts()
```

Out[68]:
```
True     56987
False    55948
Name: IsBorrowerHomeowner, dtype: int64
```

In [69]:
```
prosper_loan_data_clean_df.IsBorrowerHomeowner.isna().sum()
```

Out[69]:
```
0
```

Since this is clearly a boolean column without any missing values, we'll convert it accordingly.

In [70]:
```
prosper_loan_data_clean_df["IsBorrowerHomeowner"] =\
prosper_loan_data_clean_df.IsBorrowerHomeowner.astype("bool")
```

In [71]:
```
prosper_loan_data_clean_df.IsBorrowerHomeowner.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112935 entries, 0 to 113936
Series name: IsBorrowerHomeowner
Non-Null Count   Dtype
--------------   -----
112935 non-null  bool
dtypes: bool(1)
memory usage: 992.6 KB
```

DebtToIncomeRatio

In [72]:
```
prosper_loan_data_clean_df.DebtToIncomeRatio.sample(5)
```

Out[72]:
```
9676     0.47
37737    0.26
84056    0.32
11882    0.19
57107    0.27
Name: DebtToIncomeRatio, dtype: float64
```

In [73]:
```
prosper_loan_data_clean_df.DebtToIncomeRatio.describe()
```

Out[73]:
```
count    104474.000000
mean          0.276062
std           0.553793
min           0.000000
25%           0.140000
50%           0.220000
75%           0.320000
max          10.010000
Name: DebtToIncomeRatio, dtype: float64
```

In [74]:
```
prosper_loan_data_clean_df.DebtToIncomeRatio.isna().sum()
```

Out[74]:
```
8461
```

There are 8461 rows without a `DebtToIncomeRatio` value.

IncomeRange

In [75]:
```
prosper_loan_data_clean_df.IncomeRange.nunique()
```

Out[75]:
```
8
```

In [76]:
```
prosper_loan_data_clean_df.IncomeRange.value_counts()
```

Out[76]:
```
$25,000-49,999    31900
$50,000-74,999    30704
$100,000+         17172
$75,000-99,999    16763
Not displayed      7741
$1-24,999          7228
Not employed        806
$0                  621
Name: IncomeRange, dtype: int64
```

We certainly need to convert this column into a categorical type, but there's also the matter of a minor violation of the Tidy Data rules as this column is also representing the employment status for some rows. We've chosen to ignore this violation to simplify the analysis. Given that the number of rows with a value of `$0` is so low, we'll drop them from our analysis.

In [77]:
```
prosper_loan_data_clean_df = prosper_loan_data_clean_df[prosper_loan_data
```

In [78]:
```
prosper_loan_data_clean_df["IncomeRange"] = prosper_loan_data_clean_df.In
    pd.CategoricalDtype(
        ordered=True,
        categories=[
            "Not employed",
            "Not displayed",
            "$1-24,999",
            "$25,000-49,999",
            "$50,000-74,999",
            "$75,000-99,999",
            "$100,000+"
        ]
    )
)
```

In [79]:
```
prosper_loan_data_clean_df.IncomeRange.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 112314 entries, 0 to 113936
Series name: IncomeRange
Non-Null Count   Dtype
--------------   -----
112314 non-null  category
dtypes: category(1)
memory usage: 987.5 KB
```

LoanOriginalAmount

In [80]:
```
prosper_loan_data_clean_df.LoanOriginalAmount.describe()
```

Out[80]:
```
count    112314.000000
mean       8324.573784
std        6234.488919
min        1000.000000
25%        4000.000000
50%        6404.500000
75%       12000.000000
max       35000.000000
Name: LoanOriginalAmount, dtype: float64
```

In [81]:
```
prosper_loan_data_clean_df.LoanOriginalAmount.isna().sum()
```

Out[81]:
```
0
```

Nothing needs to be done with this column.

Final steps

The last thing to do is to drop the unnecessary columns and to save the dataframe on to a format that allows us to keep the type information so we don't have to reprocess the original dataset each time we want to perform the rest of the exploration steps.

In [82]:
```
prosper_loan_data_clean_df = prosper_loan_data_clean_df[[
    "ListingKey",  # Needed to individualize loans
    "ListingCreationDate",
    "CreditGrade",
    "Term",
    "LoanStatus",
    "DelinquencyBucket",
    "ClosedDate",
    "BorrowerAPR",
    "BorrowerRate",
    "LenderYield",
    "ProsperRating (Alpha)",
    "ListingCategory",
    "Occupation",
    "EmploymentStatus",
    "EmploymentStatusDuration",
    "IsBorrowerHomeowner",
    "DebtToIncomeRatio",
    "IncomeRange",
    "LoanOriginalAmount",
    "LoanOriginationDate"
]]
```

In [83]:
```
prosper_loan_data_clean_df.sample(5)
```

Out[83]:

	ListingKey	ListingCreationDate	CreditGrade	Term	LoanStatus
51935	21E435146560950839AF4B6	2011-05-05 20:28:20.117	NaN	36	Completed
64959	7B763594337841350F24660	2013-11-18 14:31:37.433	NaN	60	Current
46003	E0653415403276527E7B45F	2008-03-11 16:57:54.150	AA	36	Completed
78931	F8823542677633797162518	2012-03-12 14:40:34.520	NaN	60	Completed
102700	C5C234160705673717C15F8	2008-03-09 19:35:01.500	A	36	Completed

In [84]:
```
prosper_loan_data_clean_pickle = Path("prosper_loan_data_clean.pickle")
```

In [85]:
```
prosper_loan_data_clean_df.to_pickle(prosper_loan_data_clean_pickle)
```

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

After a few steps of cleanup, this is the final set of columns that will support the rest of our analysis.

- `ListingCreationDate`
- `CreditGrade`
- `Term`
- `LoanStatus`
- `DelinquencyBucket` (added)
- `ClosedDate`
- `BorrowerAPR`
- `BorrowerRate`
- `LenderYield`
- `ProsperRating (Alpha)`
- `ListingCategory` (added)
- `Occupation`
- `EmploymentStatus`
- `EmploymentStatusDuration`
- `IsBorrowerHomeowner`
- `DebtToIncomeRatio`
- `IncomeRange`
- `LoanOriginalAmount`
- `LoanOriginationDate`

As part of the wrangling process, we've added two columns:

- `DelinquencyBucket` : Extracted from `LoanStatus` , it contains the delinquency bucket for past due loans.
- `ListingCategory` : Converted from `ListingCategory (numeric)` with the information from the data dictionary and the API documentation.

The list of columns/variables can be categorized as follows:

Variable	Quantitative		Categorical	
	Continuous	Discrete	Nominal	Ordinal
ListingCreationDate	✓			
CreditGrade				✓
Term		✓		
LoanStatus			✓	
DelinquencyBucket				✓
BorrowerAPR	✓			
BorrowerRate	✓			
LenderYield	✓			
ProsperRating (Alpha)				✓
ListingCategory			✓	
Occupation			✓	
EmploymentStatus			✓	
EmploymentStatusDuration		✓		
IsBorrowerHomeowner			✓	
DebtToIncomeRatio	✓			
IncomeRange				✓
LoanOriginalAmount	✓			
LoanOriginationDate		✓		

With this information in mind, we can begin our exploration.

Univariate Exploration

At the beginning of the previous question we had asked two questions related to time:

- How did the loans fared over the years. Is the 2008 global crisis represented?
- Are there specific period with higher activity than others?

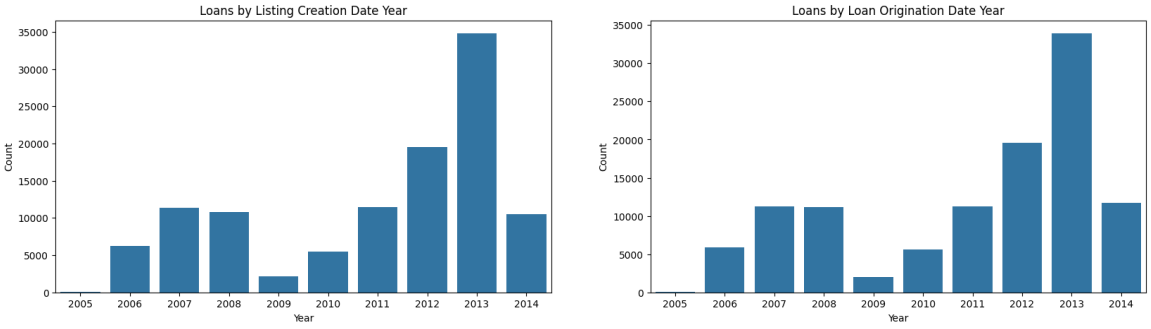
We have two variables that might help answer these questions:

- `ListingCreationDate` , and
- `LoanOriginationDate`

Let's analyze each of these to see if we can provide an answer to these questions. First, let's look number of loans per year:

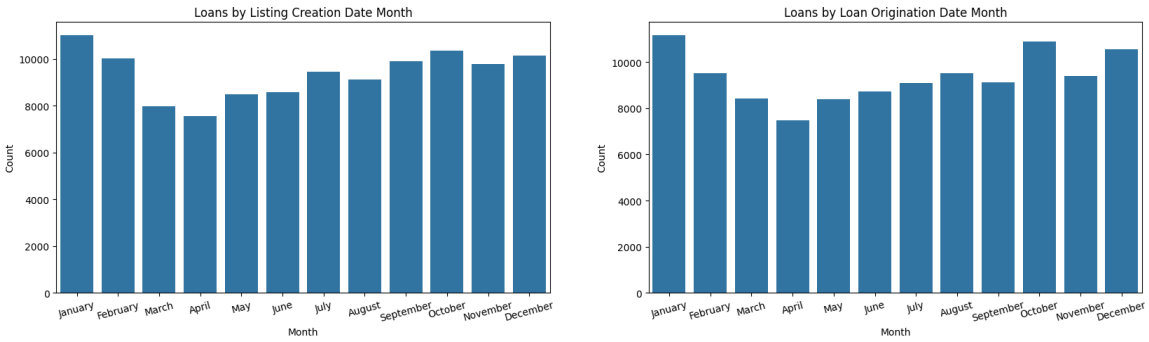
In [86]:
```
base_color = sns.color_palette()[0]
```

In [87]:
```
plt.figure(figsize = [20, 5])

# ListingCreationDate subplot
plt.subplot(1, 2, 1)
sns.countplot(
    x=prosper_loan_data_clean_df.ListingCreationDate.dt.year,
    color=base_color
);
plt.xlabel("Year")
plt.ylabel("Count")
plt.title("Loans by Listing Creation Date Year")

# LoanOriginationDate subplot
plt.subplot(1, 2, 2)
sns.countplot(
    x=prosper_loan_data_clean_df.LoanOriginationDate.dt.year,
    color=base_color
);
plt.xlabel("Year")
plt.ylabel("Count")
plt.title("Loans by Loan Origination Date Year");
```



As you can see, the count plots are pretty comparable with a mode at year 2013. There's a huge dip in 2009 which might be due to the 2008 global financial crisis. We can't really comment on the value of 2014 as it is only a partial sample for that year. Let's focus now on the month of the year to see if there's higher activity on any of them.

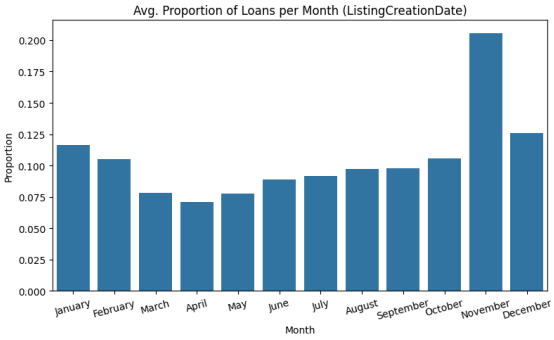
In [88]:
```
plt.figure(figsize = [20, 5])

month_names = list(map(lambda m: calendar.month_name[m], range(1, 13)))

# ListingCreationDate subplot
plt.subplot(1, 2, 1)
sns.countplot(
    x=prosper_loan_data_clean_df.ListingCreationDate.dt.month_name(),
    color=base_color,
    order=month_names
);
plt.xlabel("Month")
plt.xticks(rotation=15)
plt.ylabel("Count")
plt.title("Loans by Listing Creation Date Month")

# LoanOriginationDate subplot
plt.subplot(1, 2, 2)
sns.countplot(
    x=prosper_loan_data_clean_df.LoanOriginationDate.dt.month_name(),
    color=base_color,
    order=month_names
);
plt.xlabel("Month")
plt.xticks(rotation=15)
plt.ylabel("Count")
plt.title("Loans by Loan Origination Date Month");
```

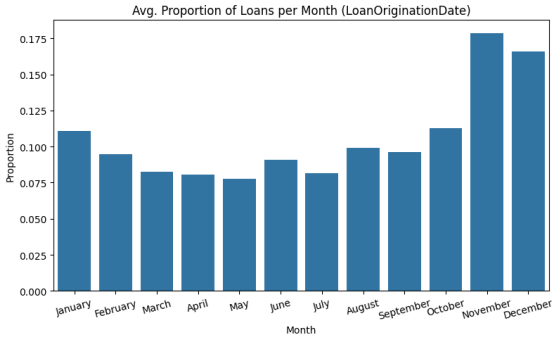


Once again we see a similar distribution in both variables, but given the huge differences in the number of deals per year, it will be better if we explore the proportion of deals for each month for any given year.

In [89]:
```
def get_monthly_proportion_mean(df: pd.DataFrame, id_column: str, dt_colu
    """
    Computes the mean proportion for each month of every year
    """
    monthly_proportion_mean =  (
        df[id_column].groupby([                                          # G
            prosper_loan_data_clean_df[dt_column].dt.year,
            prosper_loan_data_clean_df[dt_column].dt.month,              # W
        ])
        .count()                                                         # C
        .groupby(
            level=0                                                      # G
        ).apply(
            lambda x: x / x.sum()                                        # C
        ).groupby(
            level=1                                                      # G
        ).mean()                                                         # T
    )
    monthly_proportion_mean.index = monthly_proportion_mean.index.map(lam
    return monthly_proportion_mean
```

In [90]:
```
plt.figure(figsize=(20, 5))

monthly_proportion_mean_lcd = get_monthly_proportion_mean(
    prosper_loan_data_clean_df,
    "ListingKey",
    "ListingCreationDate"
)

monthly_proportion_mean_lod = get_monthly_proportion_mean(
    prosper_loan_data_clean_df,
    "ListingKey",
    "LoanOriginationDate"
)

# ListingCreationDate subplot
plt.subplot(1, 2, 1)
sns.barplot(
    x=monthly_proportion_mean_lcd.index,
    y=monthly_proportion_mean_lcd.values,
    color=base_color
);
plt.xlabel("Month")
plt.xticks(rotation=15)
plt.ylabel("Proportion")
plt.title("Avg. Proportion of Loans per Month (ListingCreationDate)");

# LoanOriginationDate subplot
plt.subplot(1, 2, 2)
sns.barplot(
    x=monthly_proportion_mean_lod.index,
    y=monthly_proportion_mean_lod.values,
    color=base_color
);
plt.xlabel("Month")
plt.xticks(rotation=15)
plt.ylabel("Proportion")
plt.title("Avg. Proportion of Loans per Month (LoanOriginationDate)");
```





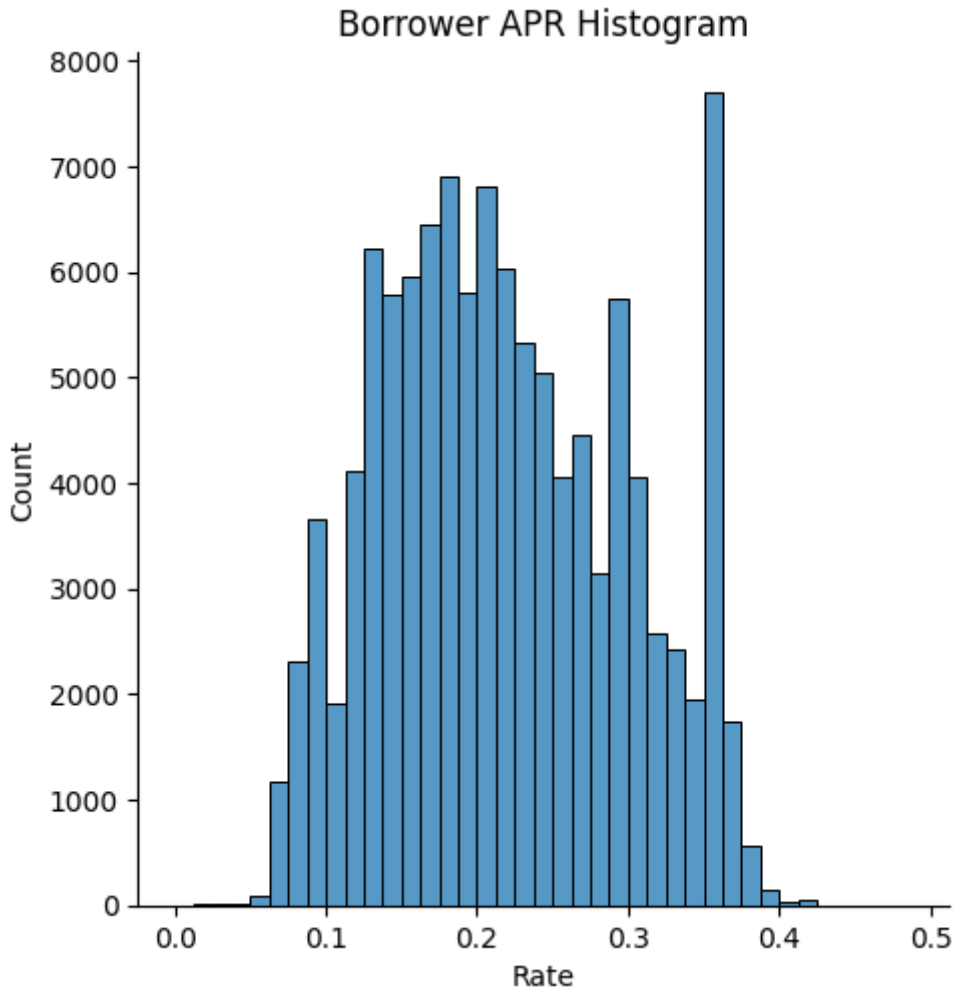
Now the two charts do show some differences, but the dip around April remains.

Let's focus now on

- How are the borrower APR, borrower rate and lender yield related to each other?

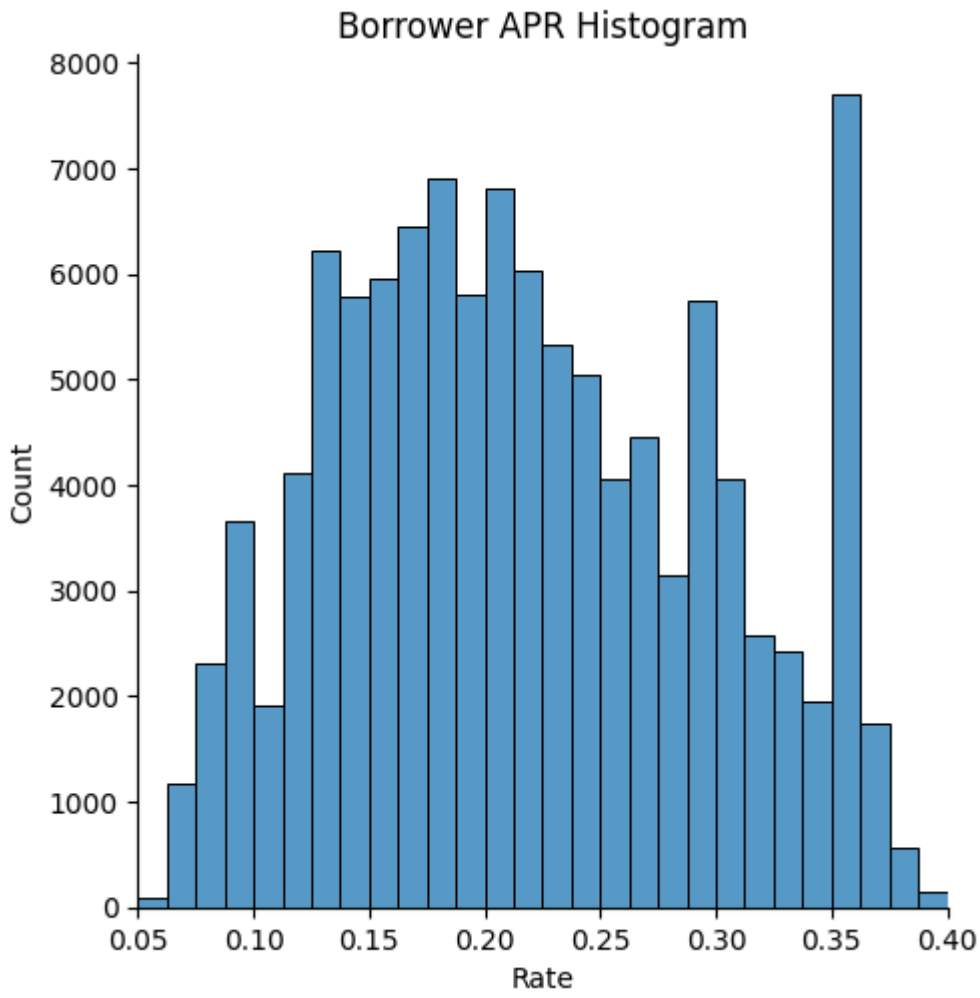
Before we can attempt to answer this question, we need to look at the specific distribution of each variable. Being all numeric continuos variables, histograms are going to provide the answers we need.

In [91]:
```
sns.displot(
    prosper_loan_data_clean_df.BorrowerAPR,
    kind="hist",
    bins=np.arange(0, 0.5, 0.0125)
)
plt.xlabel("Rate")
plt.title("Borrower APR Histogram");
```



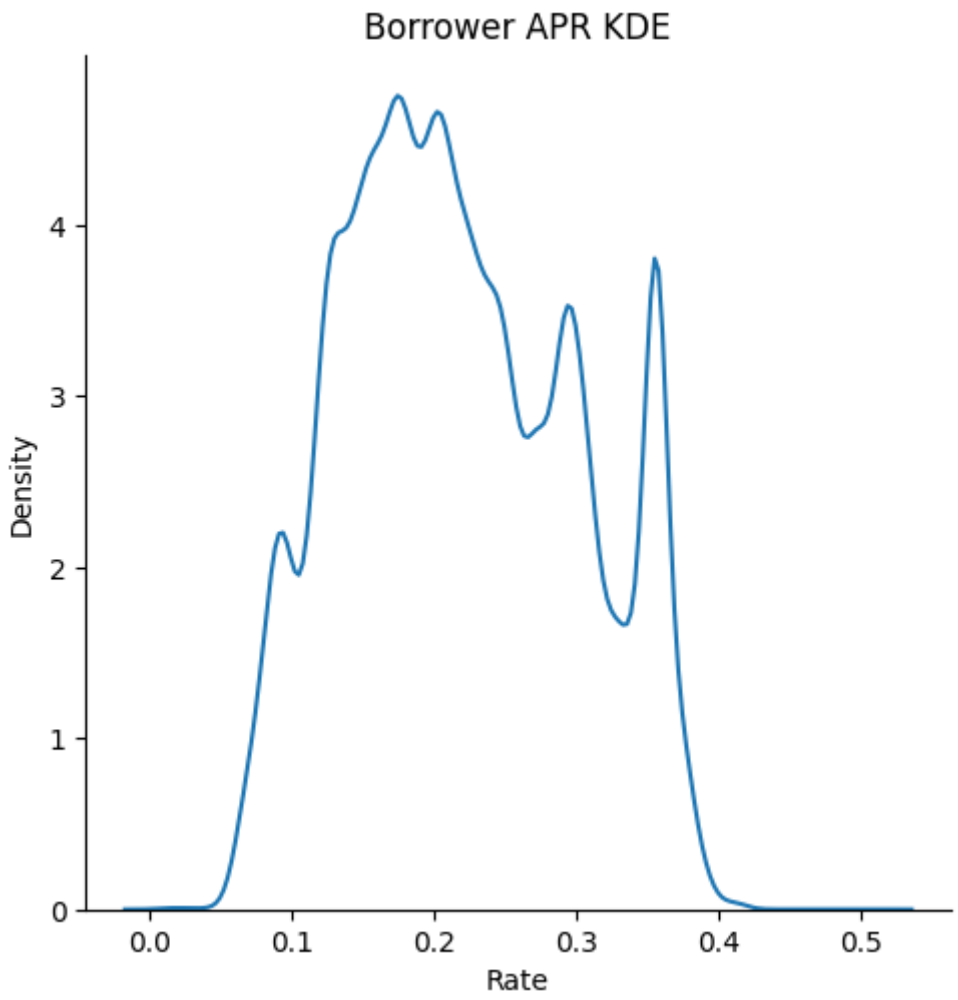
The chart clearly shows a multimodal distribution. Let's tighten the limits a little bit.

In [92]:
```
sns.displot(
    prosper_loan_data_clean_df.BorrowerAPR,
    kind="hist",
    bins=np.arange(0, 0.5, 0.0125)
)
plt.xlabel("Rate")
plt.xlim(0.05, 0.4)
plt.title("Borrower APR Histogram");
```



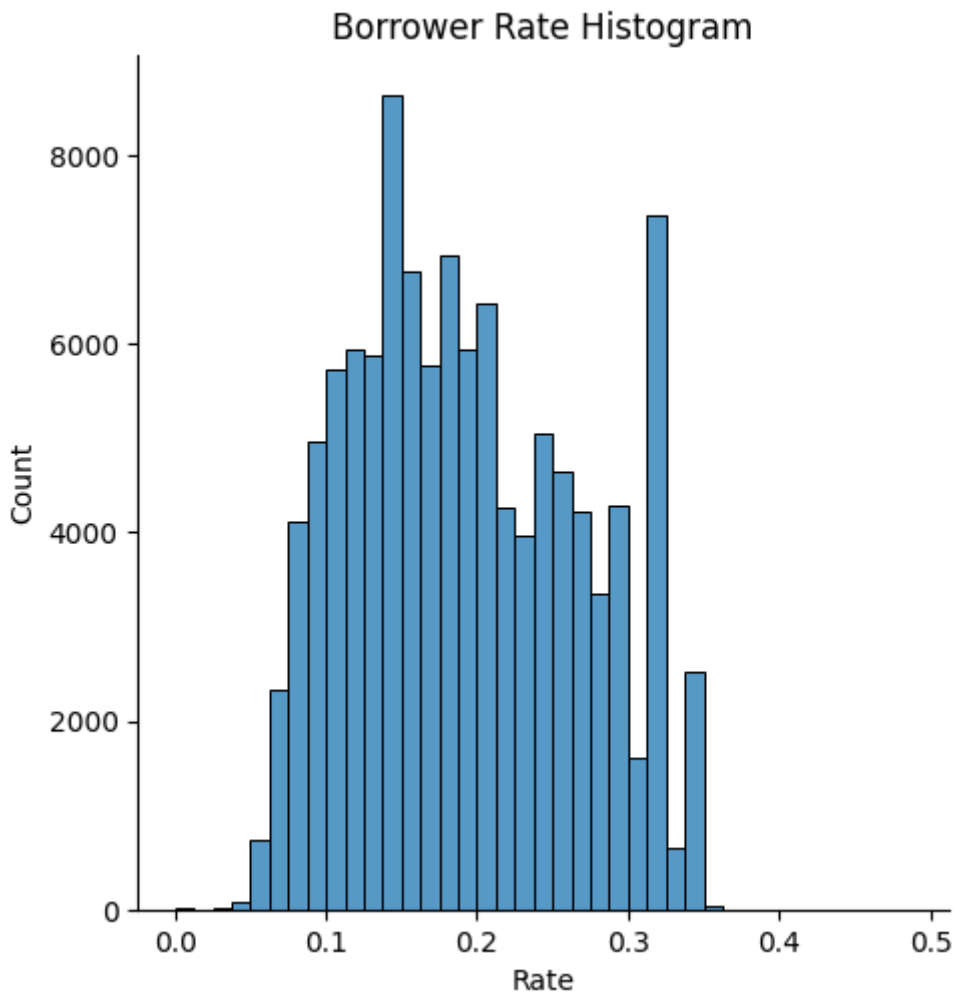
There are peaks between 0.05, and 0.1, between 0.1 and 0.15, around 0.2, around 0.3 and a huge spike aorund 0.35. Let's focus on those segments. Let's try plotting the Kernel Density Estimation:

In [93]:
```
sns.displot(
    prosper_loan_data_clean_df.BorrowerAPR,
    kind="kde"
);
plt.xlabel("Rate")
plt.title("Borrower APR KDE");
```



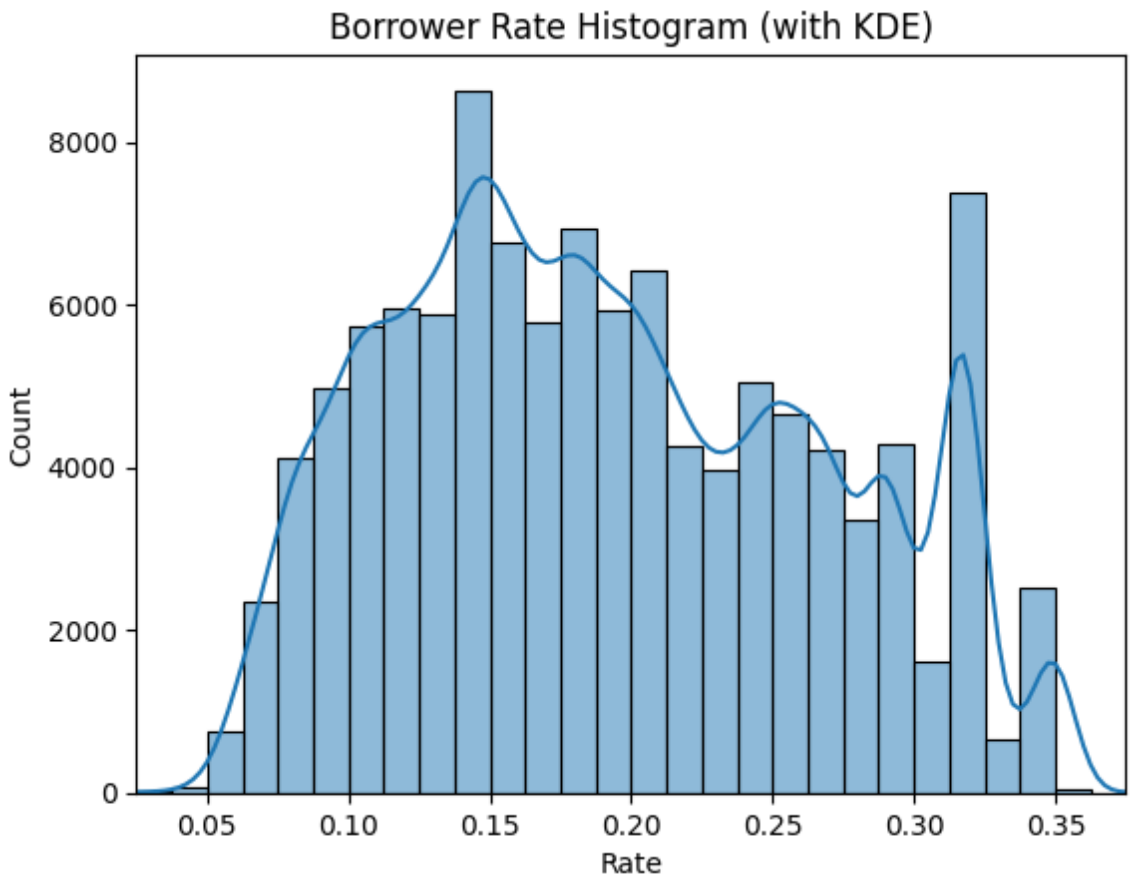
Now the peaks are more clearly shown. Maybe we're seing the influence of a categorical value the lumps interest rates around these peaks? Let's analyze the rest of the variables to see if there's something that might help explain this distribution.

In [94]:
```
sns.displot(
    prosper_loan_data_clean_df.BorrowerRate,
    kind="hist",
    bins=np.arange(0, 0.5, 0.0125)
)
plt.xlabel("Rate")
plt.title("Borrower Rate Histogram");
```



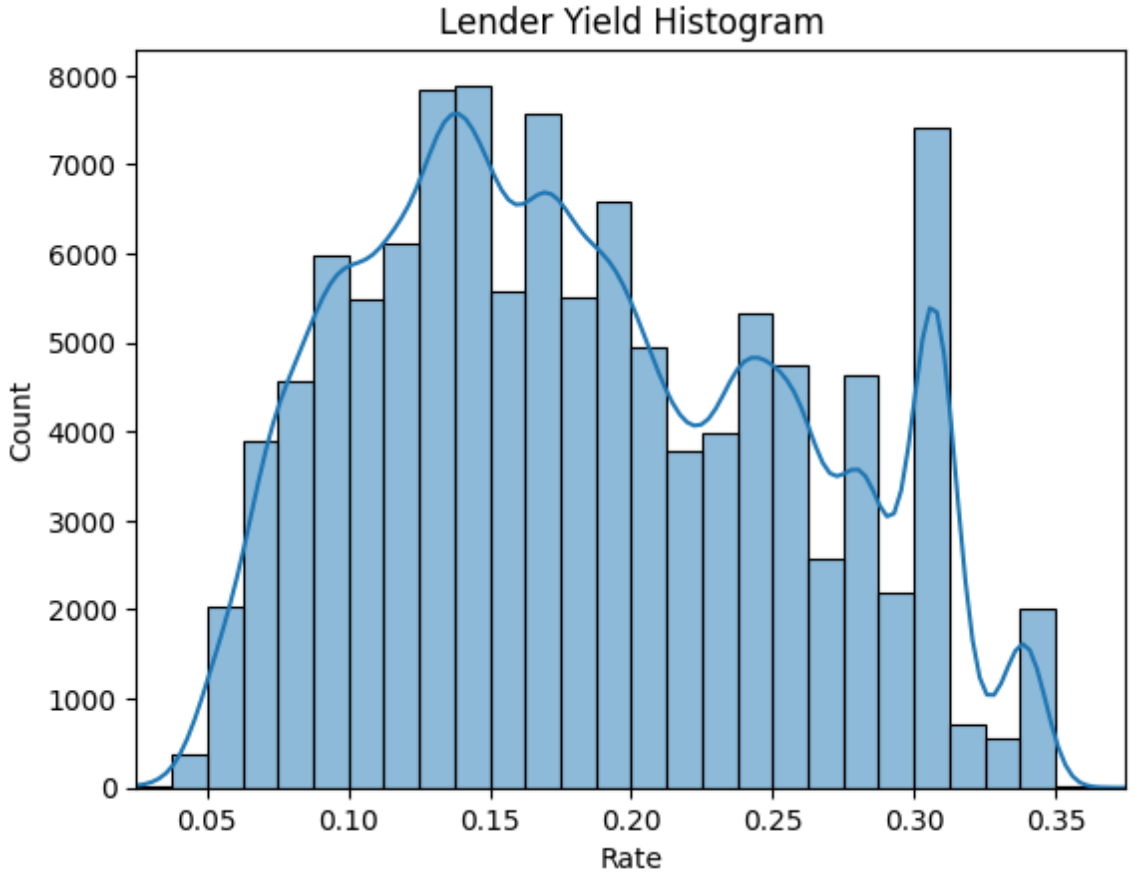
This looks an awful lot like the `BorrowerAPR` histogram, which would be consistent with the definition of APR which includes the rate itself.

In [95]:
```
sns.histplot(
    prosper_loan_data_clean_df.BorrowerRate,
    bins=np.arange(0, 0.5, 0.0125),
    kde=True
)
plt.title("Borrower Rate Histogram (with KDE)")
plt.xlabel('Rate')
plt.xlim(0.025, 0.375);
```



The plots are similar (multimodal) with peaks at different values (as expected).
Let's look at `LenderYield` as well:

In [96]:
```
sns.histplot(
    prosper_loan_data_clean_df.LenderYield,
    bins=np.arange(0, 0.5, 0.0125),
    kde=True
)
plt.title("Lender Yield Histogram");
plt.xlabel('Rate')
plt.xlim(0.025, 0.375);
```



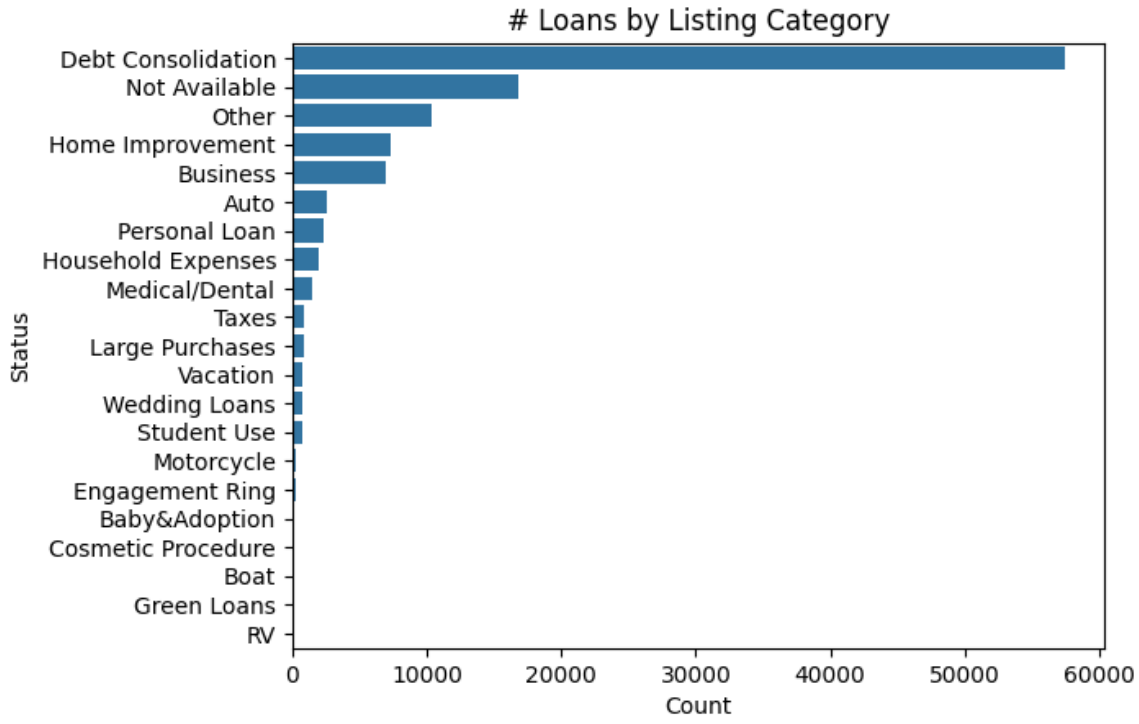
We can see that the distributions of `BorrowerRate` and `LenderYield` are pretty similar. We'll come back to these variables when we do multivariate analysis.

The answer to

- Which are the most popular loan categories?

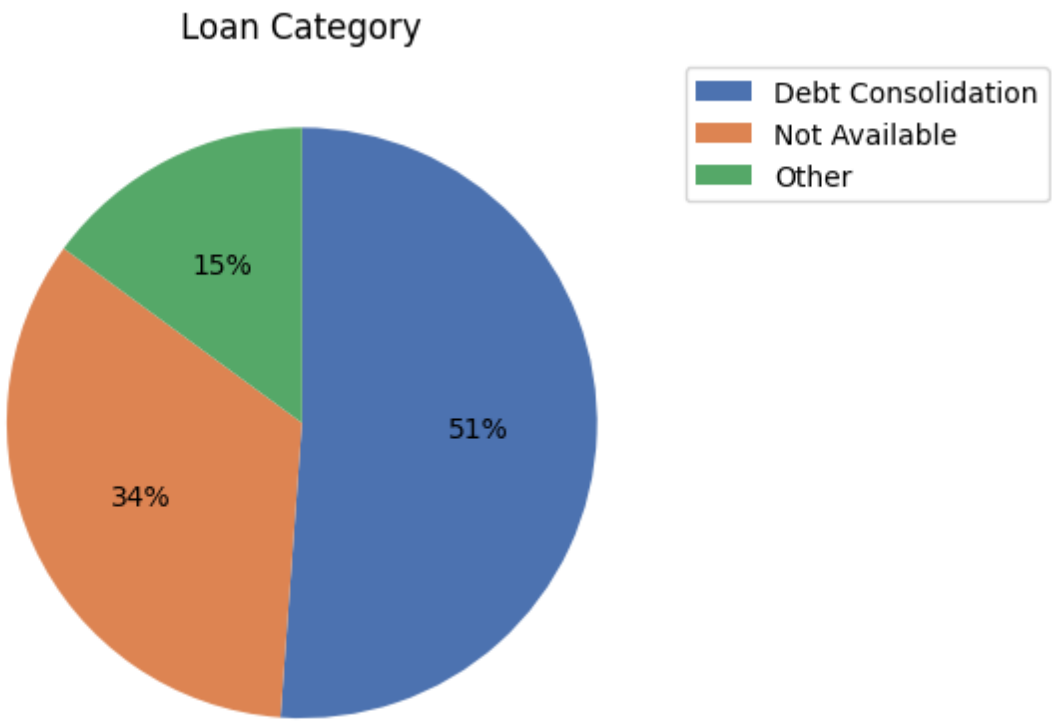
can be easily shown with a simple bar chart:

In [97]:
```
sns.countplot(
    y=prosper_loan_data_clean_df.ListingCategory,
    color=base_color,
    order=prosper_loan_data_clean_df.ListingCategory.value_counts().index
);
plt.xlabel("Count")
plt.ylabel("Status")
plt.title("# Loans by Listing Category");
```



The vast majority of deals are destined towards debt consolidation. The next two categories are "Not Available" and "Other" so any analysis of the influence of this variable on the APR is going to be limited. That being said, just the fact of showing that most of the loans are for debt consolidation is worth noting, but we need a chart that better reflects this fact:

In [98]:
```
plt.pie(
    prosper_loan_data_clean_df.ListingCategory.apply(
        lambda x: x if x in ("Debt Consolidation", "Not Available") else
    ).value_counts(),
    colors=sns.color_palette('deep'),
    startangle=90,
    counterclock=False,
    autopct='%.0f%%'
);
plt.legend(
    labels=("Debt Consolidation", "Not Available", "Other"),
    loc=2,
    bbox_to_anchor=(1, 1)
)
plt.title("Loan Category");
```

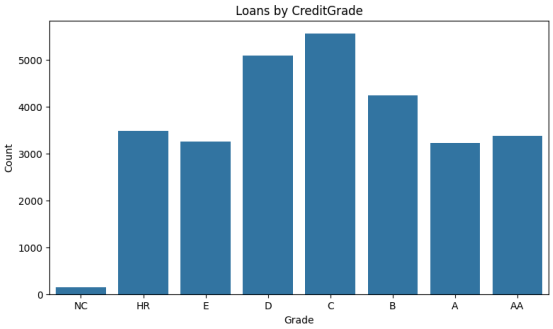


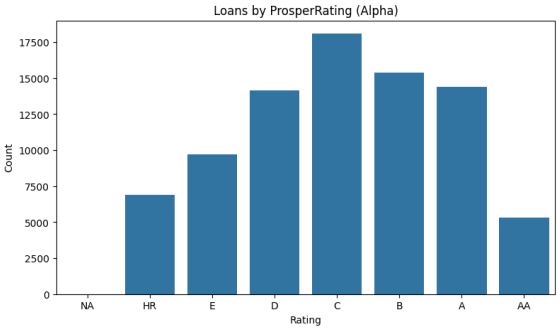
Before we can try to answer the question

- How does the employment status, occupation, home ownership, credit grades, income range and debt to income ratio affect the APR?

We need to analyse each variable in isolation to see if the available information allows us to do a significant analysis and if there are any notable patterns in the data. Let's start by analysis the credit grade variables.

In [99]:
```
plt.figure(figsize=(20, 5))

# CreditGrade subplot
plt.subplot(1, 2, 1)
sns.countplot(
    x=prosper_loan_data_clean_df.CreditGrade,
    color=base_color
);
plt.xlabel("Grade")
plt.ylabel("Count")
plt.title("Loans by CreditGrade")

# ProsperRating (Alpha) subplot
plt.subplot(1, 2, 2)
sns.countplot(
    x=prosper_loan_data_clean_df["ProsperRating (Alpha)"],
    color=base_color
);
plt.xlabel("Rating")
plt.ylabel("Count")
plt.title("Loans by ProsperRating (Alpha)");
```

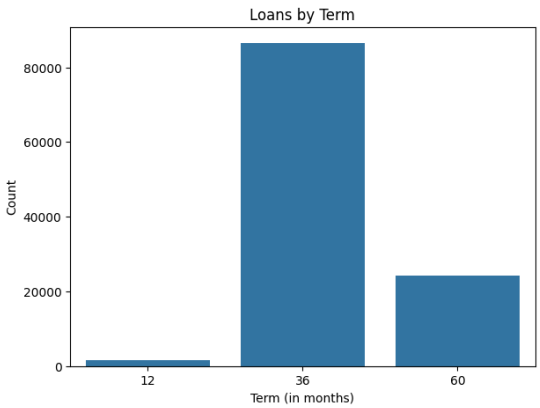


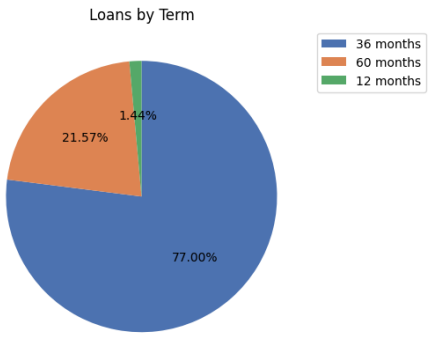


Notice that for both grading variables, the middling ranges are preferred. Usually people with a better credit rating are those that don't need to take loans at all. It's interesting that the number of loans with "AA" is comparable with those with "HR". Let's look a the `Term` now:

In [100…

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
sns.countplot(
    x=prosper_loan_data_clean_df.Term,
    color=base_color
);
plt.xlabel("Term (in months)")
plt.ylabel("Count")
plt.title("Loans by Term");

term_value_counts = prosper_loan_data_clean_df.Term.value_counts()

plt.subplot(1, 2, 2)
plt.pie(
    term_value_counts,
    colors=sns.color_palette('deep'),
    startangle=90,
    counterclock=False,
    autopct='%.2f%%'
);
plt.legend(
    labels=term_value_counts.index.map(
        lambda x: f"{x} months"
    ),
    loc=2,
    bbox_to_anchor=(1, 1)
)
plt.title("Loans by Term");
```

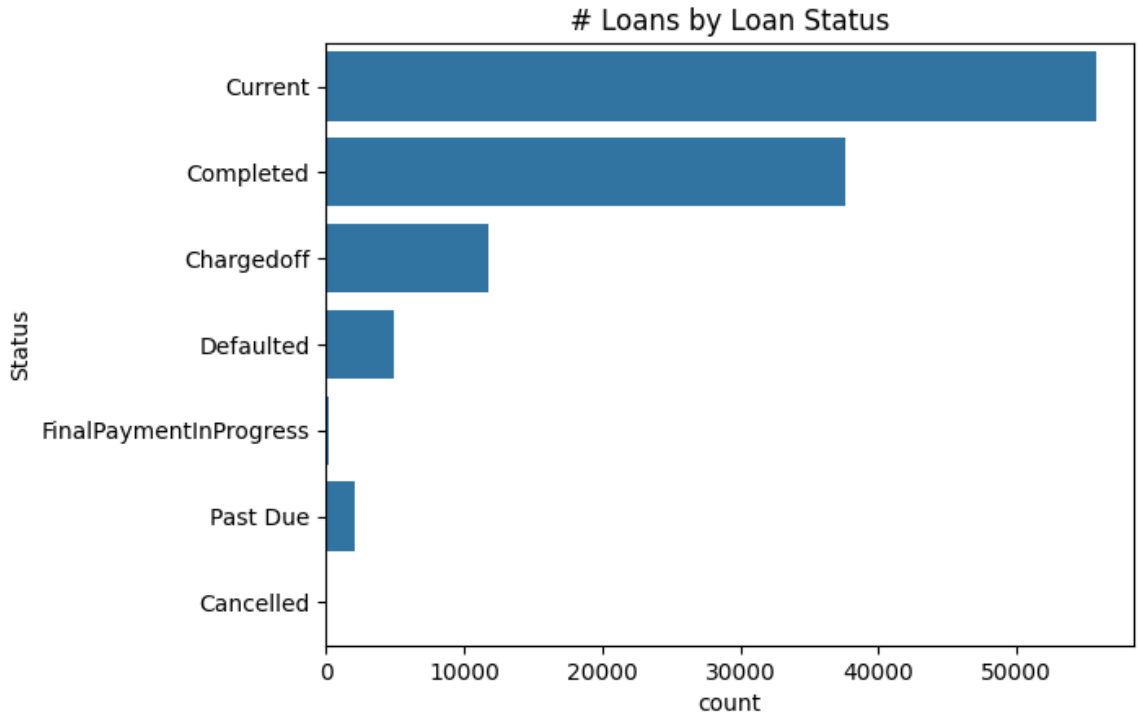




Although both charts are represent the same facts, the piechart clearly shows that the overwhelming majority of loans are taken with a 36 month term.

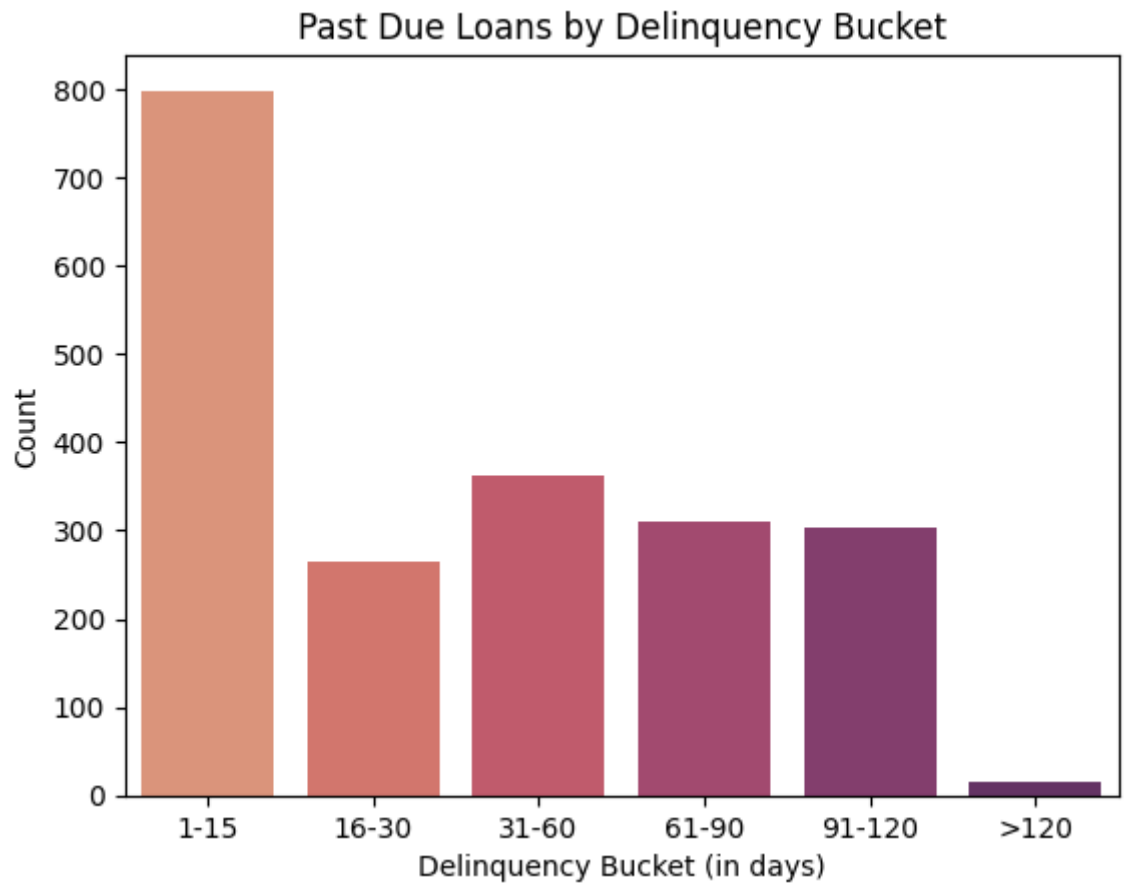
I don't intend on showing specific charts about `LoanStatus`, but the information will be useful when slicing the data set based on the categories.

In [101…
```
sns.countplot(
    y=prosper_loan_data_clean_df.LoanStatus,
    color=base_color
);
plt.ylabel("Status")
plt.title("# Loans by Loan Status");
```



The most important information about this chart is that the number of loans that are "Defaulted" or "Past Due" are low, so any analysis on those specific subsets of loans will be limited. For instance, if we wanted to plot the the `DelinquencyBucket` for "Past Due" loans:

In [102…
```
sns.countplot(
    x=prosper_loan_data_clean_df[
        prosper_loan_data_clean_df.LoanStatus == "Past Due"
    ].DelinquencyBucket,
    palette="flare"
);
plt.xlabel("Delinquency Bucket (in days)")
plt.ylabel("Count")
plt.title("Past Due Loans by Delinquency Bucket");
```



I chose a gradient palette to depict the increase in risk with each subsequent bucket. It's interesting to see that the 1-15 days period is predominant, but the rest are all comparable.

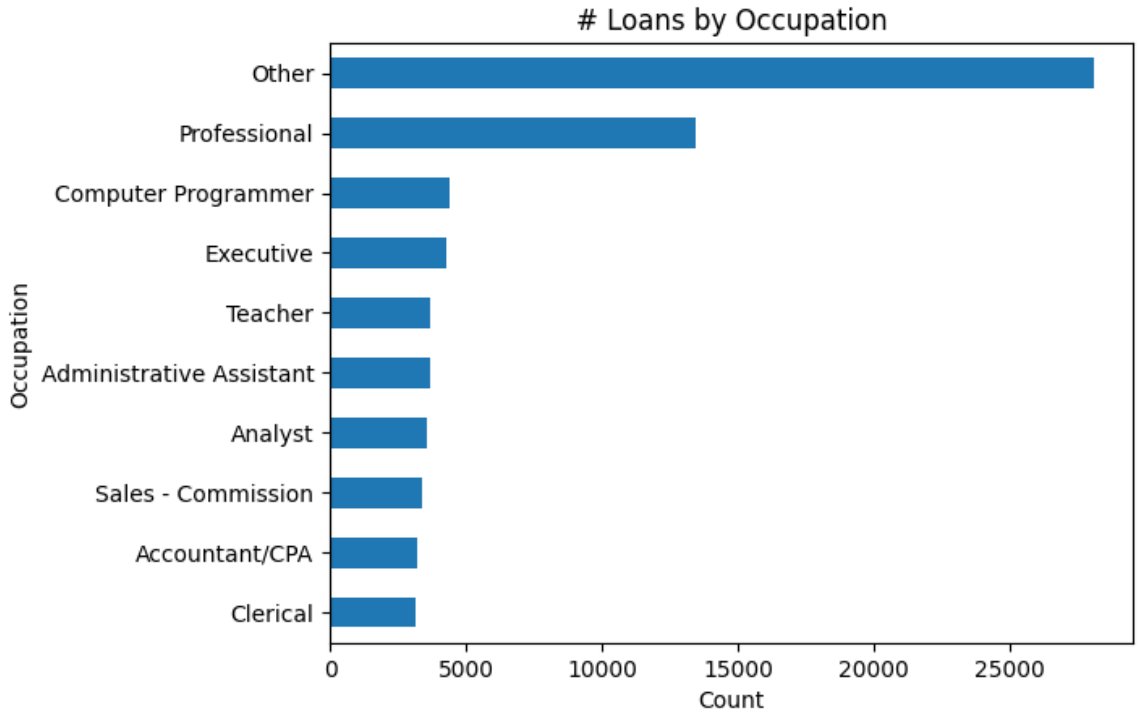
Now let's look at the employment related variables.

In [103…
```
prosper_loan_data_clean_df.Occupation.nunique()
```

Out[103]: 67

There are 67 possible values for `Occupation`, so any plot for categorical values will be hard to understand no matter the design. Let's look instead at the 10 most popular values for `Ocupation` instead:

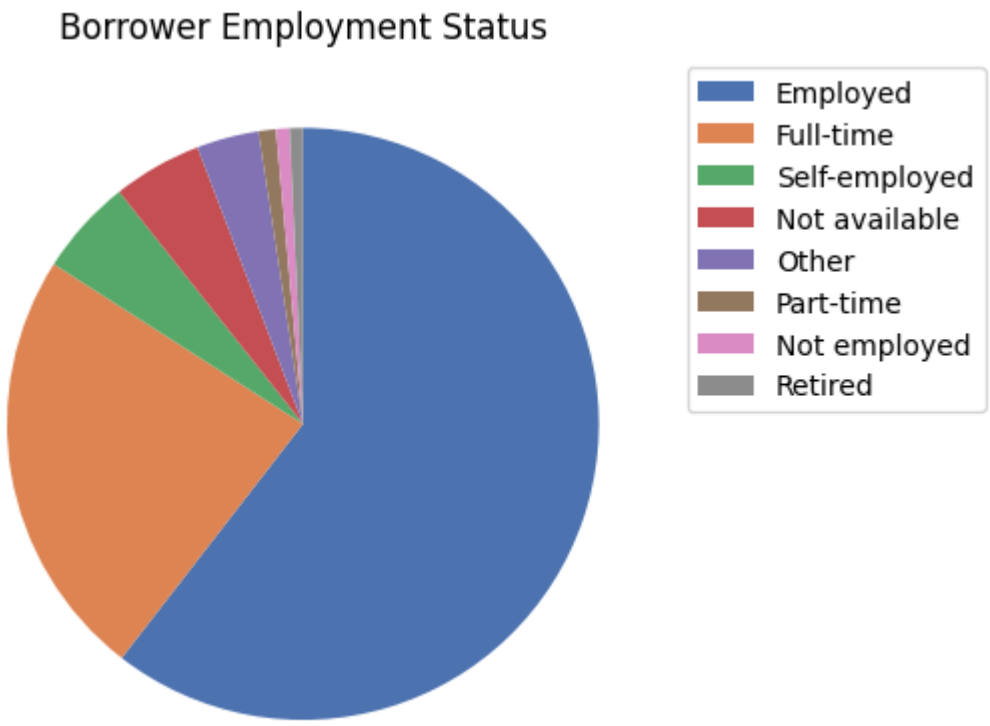
In [104…
```
prosper_loan_data_clean_df.Occupation.value_counts().head(10).sort_values
    kind="barh",
    color=base_color
);
plt.xlabel("Count")
plt.ylabel("Occupation")
plt.title("# Loans by Occupation");
```



Having "Other" as the most popular category is not goin to be very useful, so we'll drop this variable from our analysis. Let's check the `EmploymentStatus` instead:

In [105…

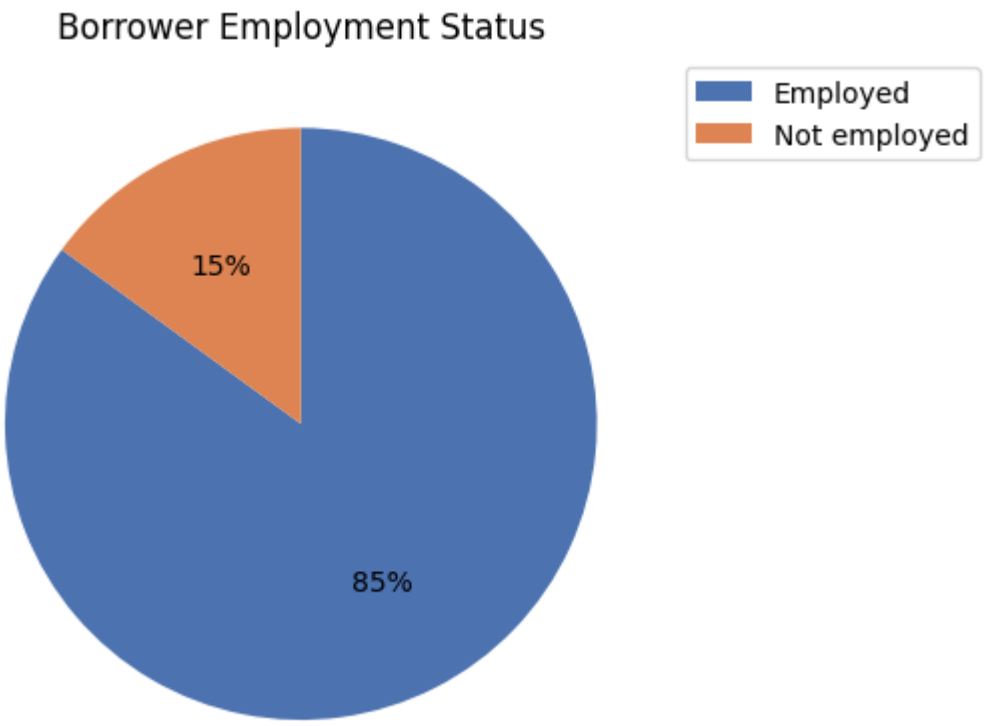
```
plt.pie(
    prosper_loan_data_clean_df.EmploymentStatus.value_counts(),
    colors=sns.color_palette('deep'),
    startangle=90,
    counterclock=False
);
plt.legend(
    labels=prosper_loan_data_clean_df.EmploymentStatus.value_counts().ind
    loc=2,
    bbox_to_anchor=(1, 1)
)
plt.title("Borrower Employment Status");
```



The sheer number of categories and the bias towards the "Other" value would complicate subsequent analysis as with the `Ocupation` variable. Also, there's no clear definition of the relationship between the `Employed`, `Full-time` and `Part-time` category. Maybe we can consolidate all of these into a single `Employed` category to show that most borrowers are in fact employed. The chart is also a bit hard to parse, so we'll improve the readability as well.

In [106…

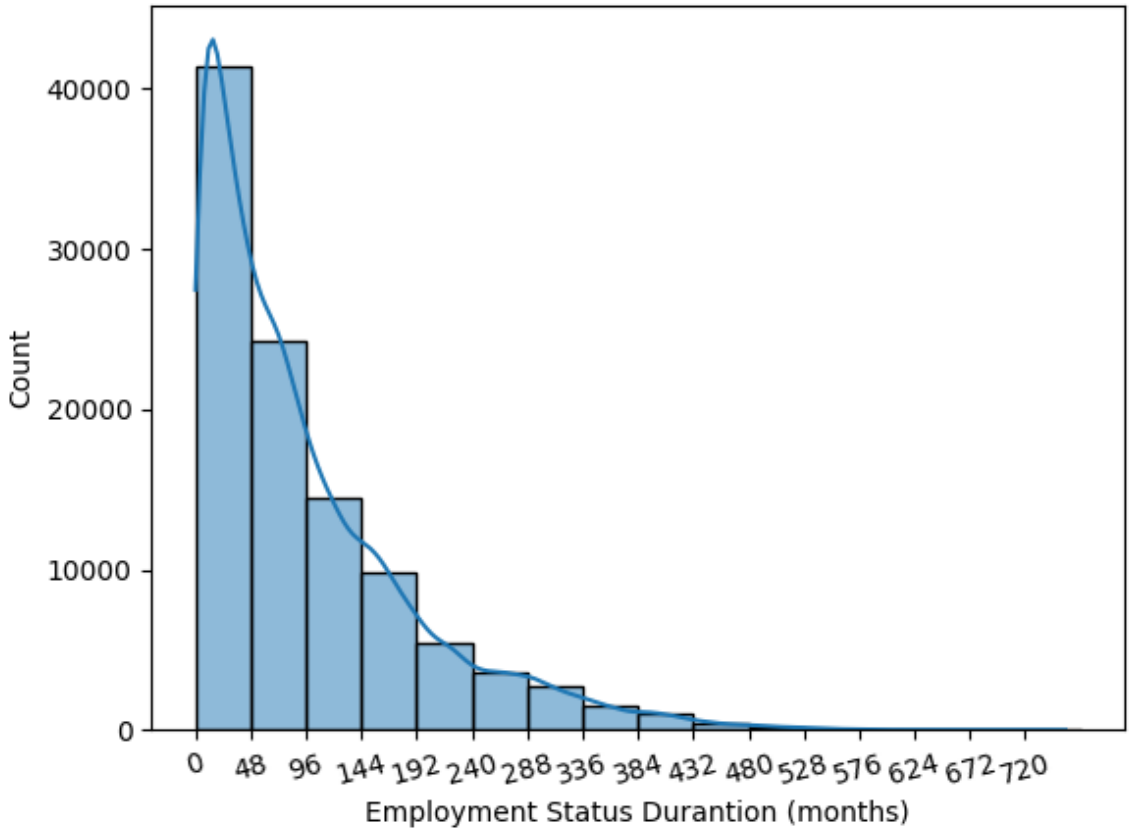
```
plt.pie(
    prosper_loan_data_clean_df.EmploymentStatus.apply(
        lambda x: "Employed" if x in ("Employed", "Full-time", "Part-time
    ).value_counts(),
    colors=sns.color_palette('deep'),
    startangle=90,
    counterclock=False,
    autopct='%.0f%%'
);
plt.legend(
    labels=("Employed", "Not employed"),
    loc=2,
    bbox_to_anchor=(1, 1)
)
plt.title("Borrower Employment Status");
```



I think this chart more clearly shows the relative proportion of employed against non employed borrowers.

In [107…

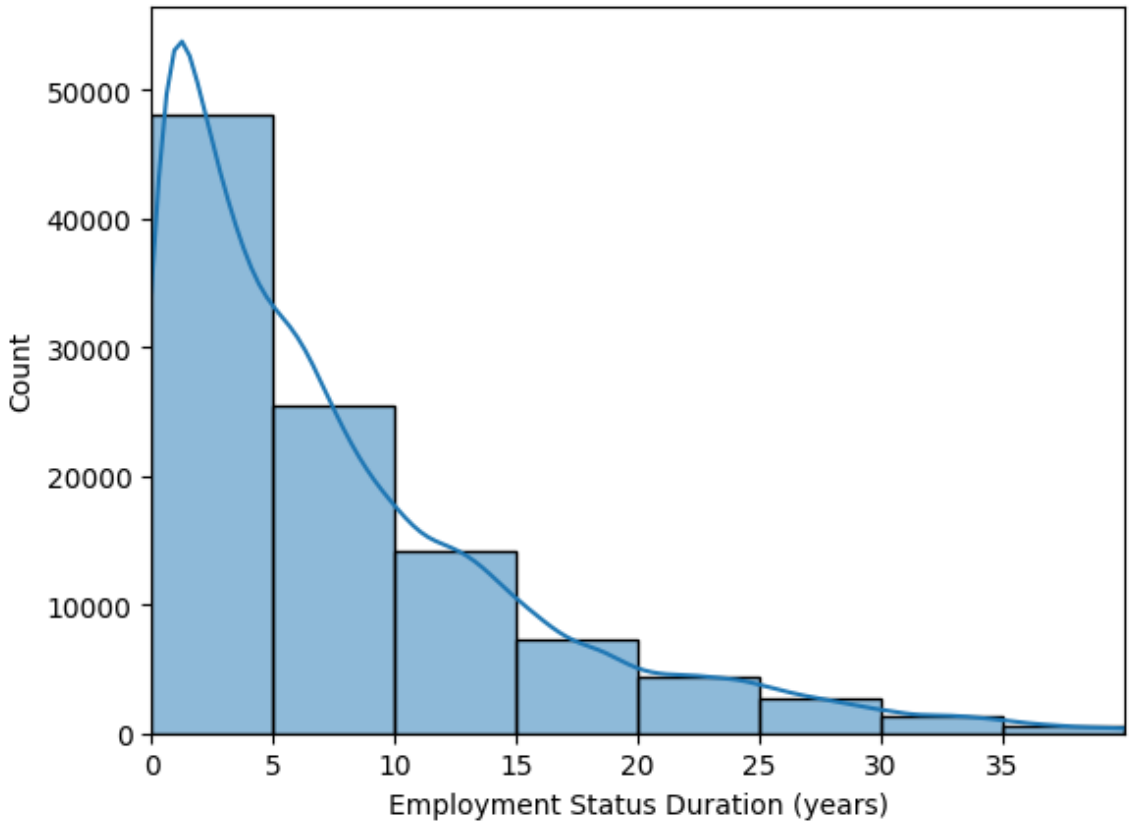
```
sns.histplot(
    prosper_loan_data_clean_df.EmploymentStatusDuration,
    binwidth=48,
    kde=True
);
plt.xlabel("Employment Status Durantion (months)");
plt.xticks(np.arange(0, 768, 48))
plt.xticks(rotation=15);
```



The chart shows a clearly right skewed distribution, which means that loans are primarily taken by people with fewer months on jobs. The issue with this chart is that it's hard for people to think in terms of months, so we'll plot the number of years instead:

In [108…

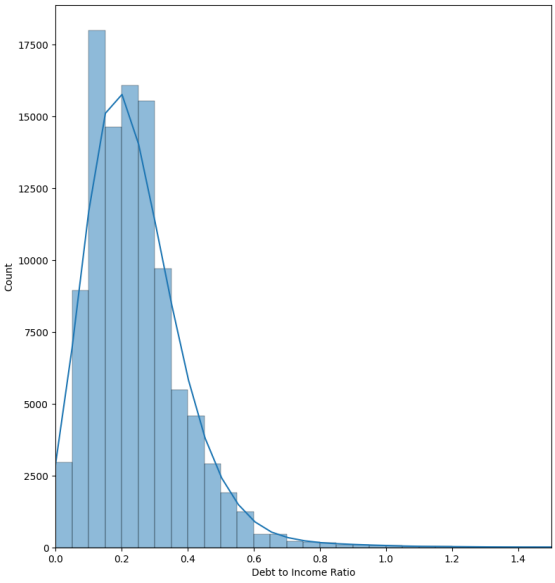
```
sns.histplot(
    np.round(prosper_loan_data_clean_df.EmploymentStatusDuration / 12.0,
    binwidth=5,
    kde=True
);
plt.xlabel("Employment Status Duration (years)")
plt.xticks(np.arange(0, 40, 5))
plt.xlim(0, 40);
```

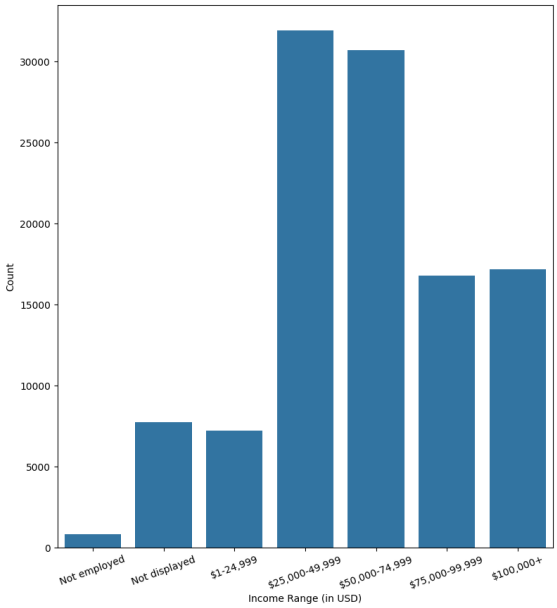


This chart shows the same distribution, but it's a bit easier to parse. Finally let's analyze the income related variables:

In [109…

```
plt.figure(figsize=(20, 10))

plt.subplot(1, 2, 1)
sns.histplot(
    prosper_loan_data_clean_df.DebtToIncomeRatio,
    binwidth=0.05,
    kde=True
);
plt.xlim(0,1.5);
plt.xlabel("Debt to Income Ratio")

plt.subplot(1, 2, 2)
sns.countplot(
    x=prosper_loan_data_clean_df.IncomeRange,
    color=base_color
);
plt.xlabel("Income Range (in USD)")
plt.xticks(rotation=20);
plt.ylabel("Count");
```





The debt to income ratio seems to have a right skewed unimodal distribution with a mean around 0.28 (which means the debt is around 30% of the income) while mid-level income ranges are the most populous, but the highest income ranges also represent a hugh proportion of the population.

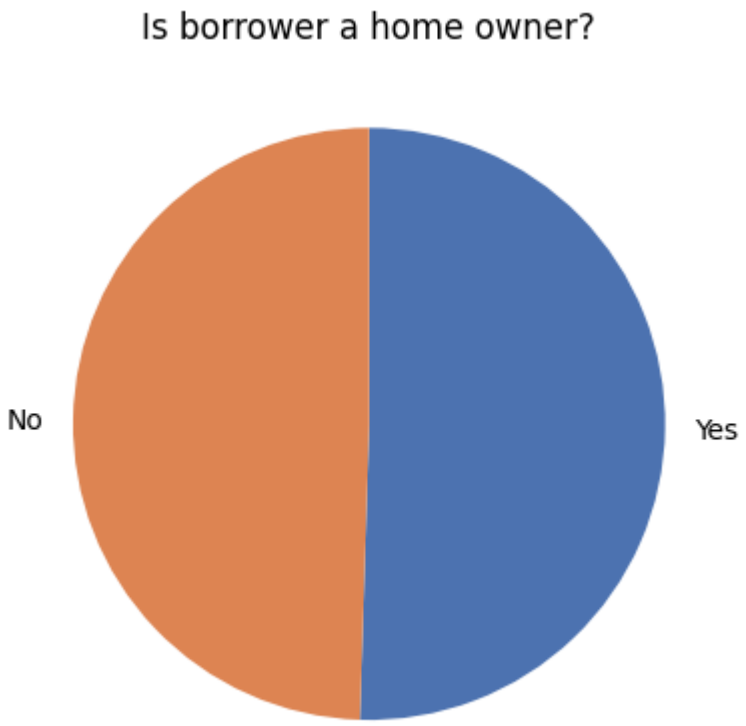
The answer to

- How many borrowers are home owners?

can be shown with a simple pie chart:

In [110…

```
plt.pie(
    prosper_loan_data_clean_df.IsBorrowerHomeowner.apply(
        lambda x: "Yes" if x else "No"
    ).value_counts(),
    colors=sns.color_palette('deep'),
    labels=("Yes", "No"),
    startangle=90,
    counterclock=False
);
plt.title("Is borrower a home owner?");
```



Half the borrowers are homeowners! We only had to transform the column slightly as "True", "False" are not very user friendly.

Finally, in order to answer

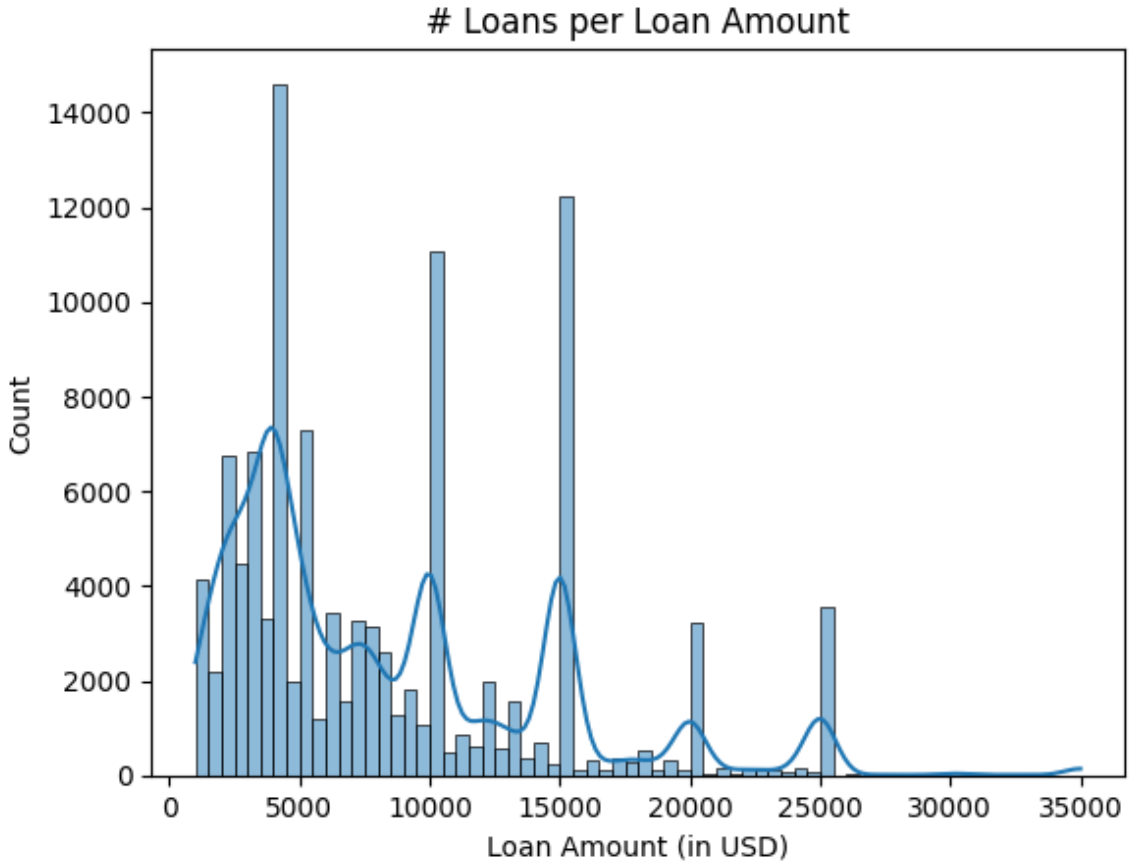
- Is there a relationship between the loan amount and the income range?

We first need to analyze the income range (which we've done before), and the loan amount.

In [111…
```
prosper_loan_data_clean_df.LoanOriginalAmount.describe()
```

Out[111]:
```
count    112314.000000
mean       8324.573784
std        6234.488919
min        1000.000000
25%        4000.000000
50%        6404.500000
75%       12000.000000
max       35000.000000
Name: LoanOriginalAmount, dtype: float64
```

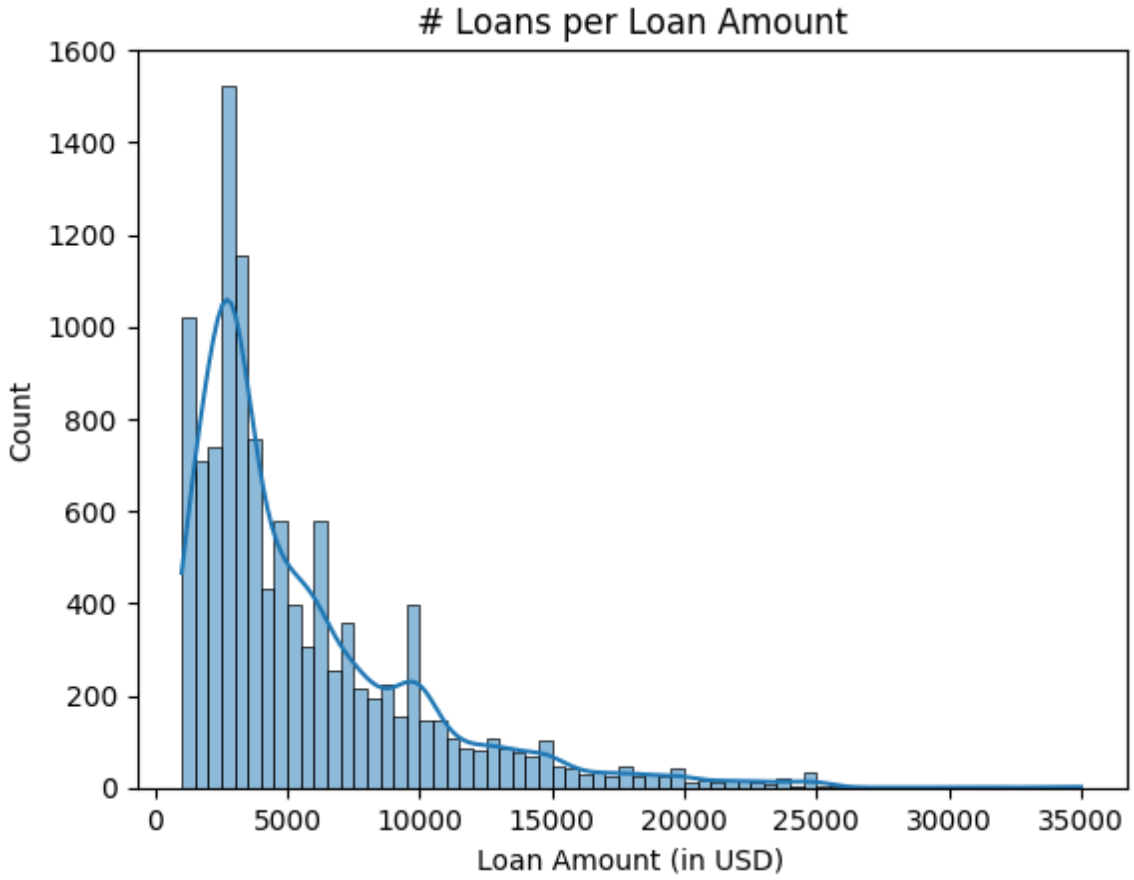
In [112…
```
sns.histplot(
    prosper_loan_data_clean_df.LoanOriginalAmount,
    bins=np.arange(1000, 30000, 500),
    kde=True
);
plt.xlabel("Loan Amount (in USD)")
plt.title("# Loans per Loan Amount");
```



It seems that the round numbers (500, 1000, 1500, 5000, 10000, 15000, etc.) are more popular that the number in between. Let's analyse the histogram without these values:

In [113…

```
sns.histplot(
    prosper_loan_data_clean_df.LoanOriginalAmount[
        prosper_loan_data_clean_df.LoanOriginalAmount % 500 != 0  # No "r
    ],
    bins=np.arange(1000, 35000, 500),
    kde=True
);
plt.xlabel("Loan Amount (in USD)")
plt.title("# Loans per Loan Amount");
```



This time the chart shows a right skewed distribution.

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

The following is a list of conclusions obtained from analysing the plotted charts:

- Dip in number of loans on 2009
- Most loans are taken at the end of the year
- BorrowerRate, BorrowerAPR and LenderYield have similar multi-modal distributions
- Most loans are taken for debt consolidation
- CreditGrade and ProsperRating have similar distributions with a higher number of loans on the mid-range grades
- The vast majority of loans have 36 month terms
- Most past due deals are in the 1 to 15 days bucket
- Unsurprisingly the employment status duration is a right-skewed unimodal distibution
- Half the borrowers are home owners
- Loans are taken mostly on "round" amounts

Most of the transformations were just type casting like converting strings to dates, and creating categorical dtypes, but most variables had the appropritate type.

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?

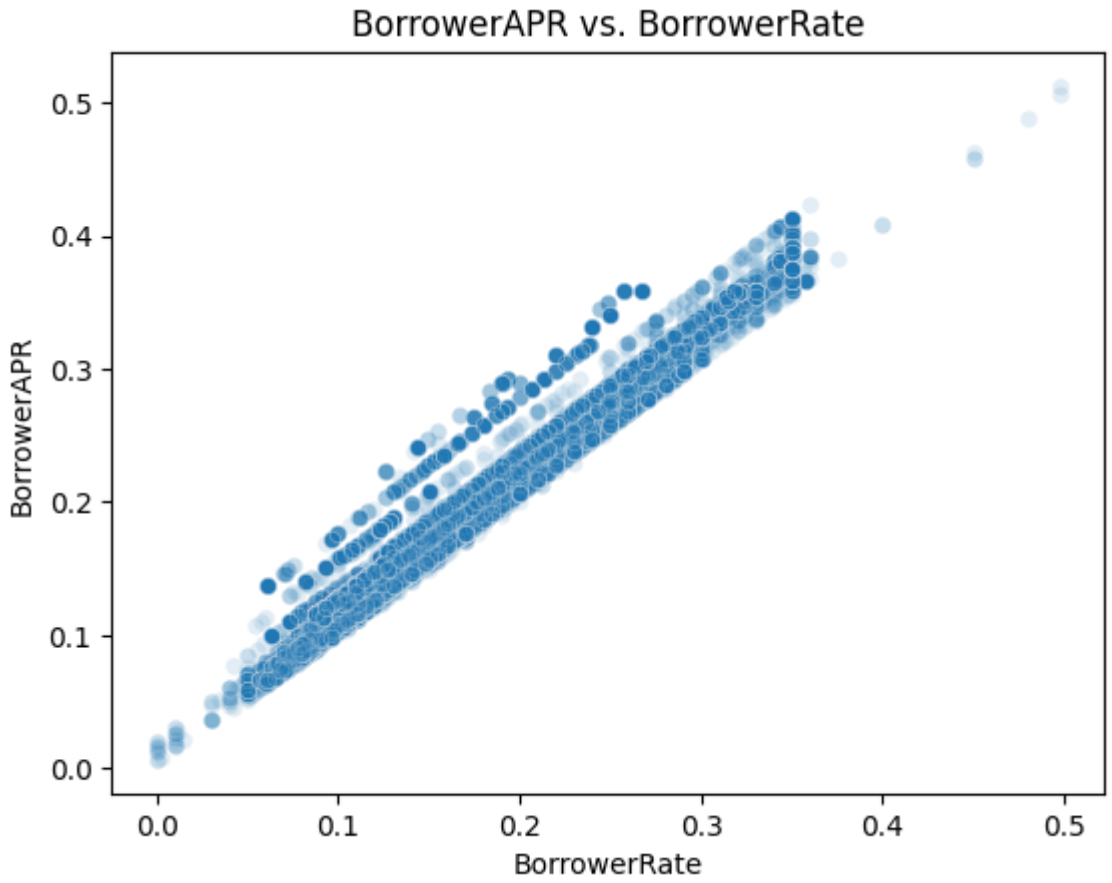
I think the most surprising facts from the features was that most loans are taken for debt consolidation and that loans are taken in round amounts. The rest of the conclusions are pretty consistent with the expected model for loans.

The most notable change to the data was the extraction of the delinquency bucket form the `LoanStatus` columns. They were clearly two different variables, and we managed to extract some conclusions from the the `DelinquencyBucket` variable itself.

The rest of the changes were the typical wrangling tasks of dropping duplicates and rows with missing values, etc.

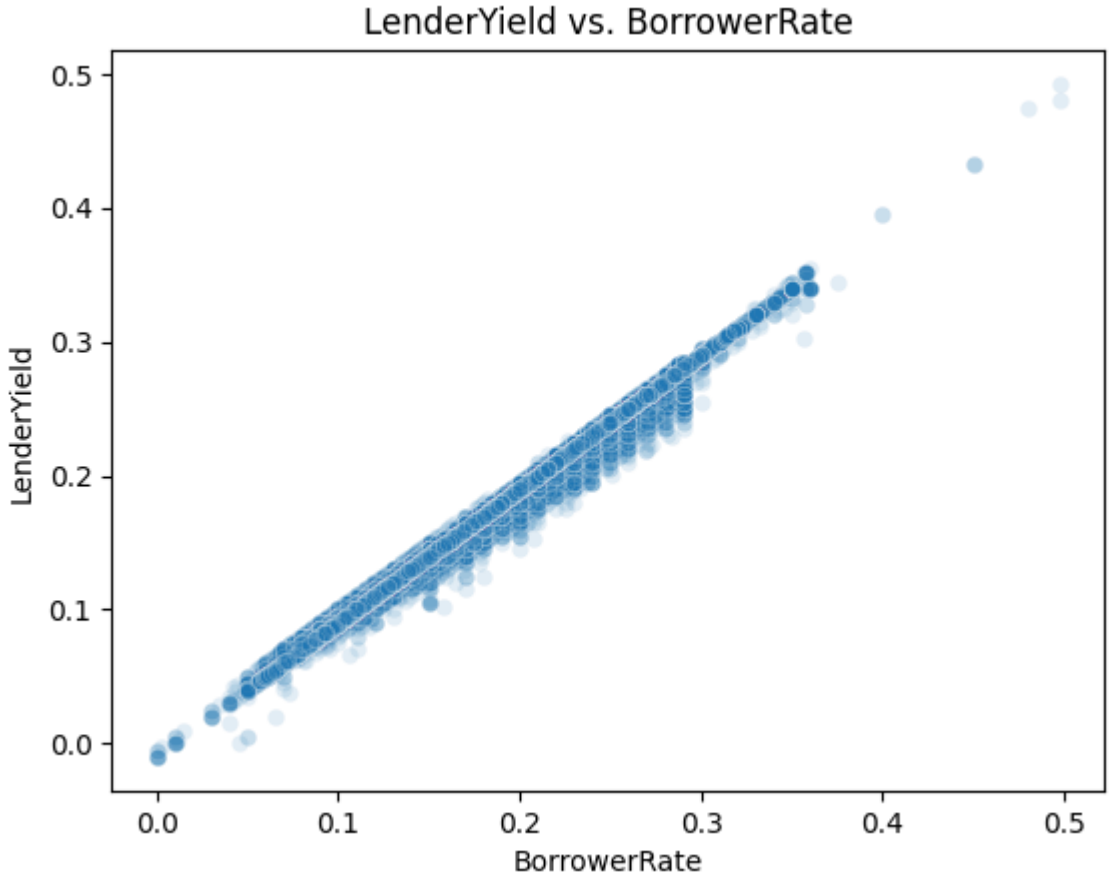
Bivariate Exploration

In [114…
```
sns.scatterplot(
    data=prosper_loan_data_clean_df,
    x="BorrowerRate",
    y="BorrowerAPR",
    alpha=0.125
)
plt.title("BorrowerAPR vs. BorrowerRate");
```



As expected, there's a linear correlation between the `BorrowerRate` and `BorrowerAPR` variables. What's notable is that there seems to be stratification where groups of loans share the same slope.

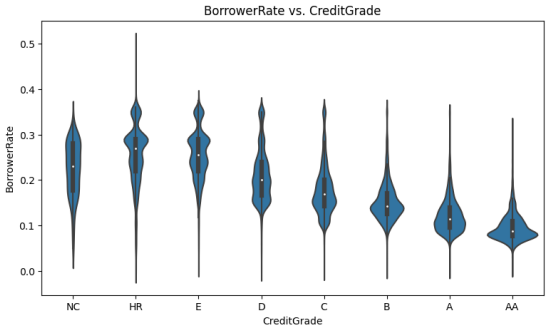
In [115…
```
sns.scatterplot(
    data=prosper_loan_data_clean_df,
    x="BorrowerRate",
    y="LenderYield",
    alpha=0.125
)
plt.title("LenderYield vs. BorrowerRate");
```

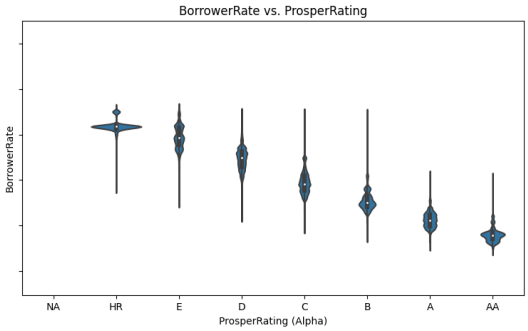


Unsuprisingly, the `LenderYield` also has a linear relationship with the `BorrowerRate`. Let's look at how both grading variables affect the borrower rate.

In [116…

```
fig, (ax1, ax2) = plt.subplots(
    1, 2,
    figsize=(20, 5),
    sharey=True
)

sns.violinplot(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df.CreditGra
    x="CreditGrade",
    y="BorrowerRate",
    color=base_color,
    ax=ax1
)
ax1.set_title("BorrowerRate vs. CreditGrade")
sns.violinplot(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df["ProsperR
    x="ProsperRating (Alpha)",
    y="BorrowerRate",
    color=base_color,
    ax=ax2
)
ax2.set_title("BorrowerRate vs. ProsperRating");
```

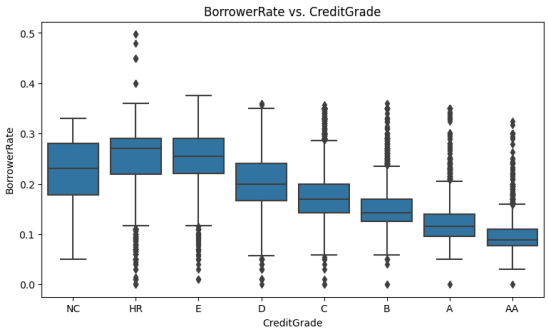


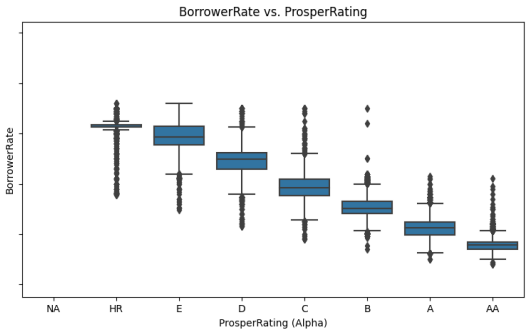


Both `CreditGrade` and `ProsperRating` show that in avergage, higher ratings get better rates, but the shape of the distributions is different as there's seems to be a lot more variability with the `CreditGrade` distribution. Let's use boxplots to make this a little clearer.

In [117…

```
fig, (ax1, ax2) = plt.subplots(
    1, 2,
    figsize=(20, 5),
    sharey=True
)

sns.boxplot(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df.CreditGra
    x="CreditGrade",
    y="BorrowerRate",
    color=base_color,
    ax=ax1
)
ax1.set_title("BorrowerRate vs. CreditGrade")
sns.boxplot(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df["ProsperR
    x="ProsperRating (Alpha)",
    y="BorrowerRate",
    color=base_color,
    ax=ax2
)
ax2.set_title("BorrowerRate vs. ProsperRating");
```

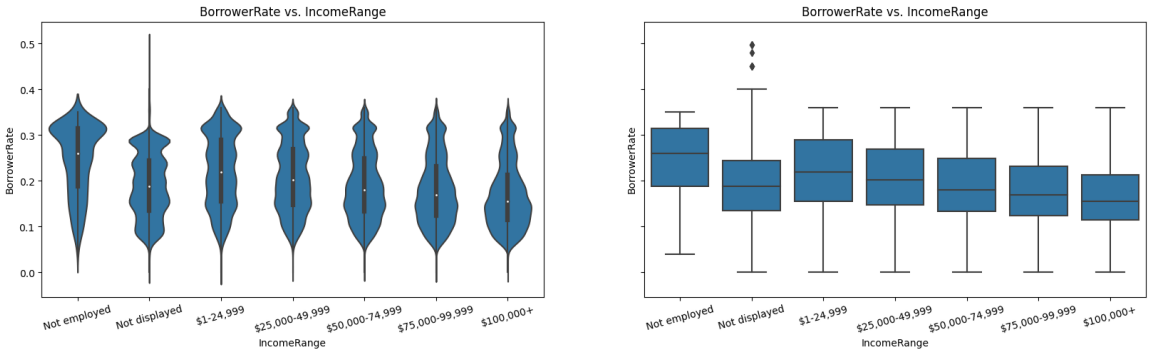




These two plots clearly show the higher grades get better rates (in average). Let's see if the same applies for the income range.

In [118…

```
fig, (ax1, ax2) = plt.subplots(
    1, 2,
    figsize=(20, 5),
    sharey=True
)

sns.violinplot(
    data=prosper_loan_data_clean_df,
    x="IncomeRange",
    y="BorrowerRate",
    color=base_color,
    ax=ax1
)
ax1.set_title("BorrowerRate vs. IncomeRange");
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=15)
sns.boxplot(
    data=prosper_loan_data_clean_df,
    x="IncomeRange",
    y="BorrowerRate",
    color=base_color,
    ax=ax2
)
ax2.set_title("BorrowerRate vs. IncomeRange");
ax2.set_xticklabels(ax2.get_xticklabels(), rotation=15);
```

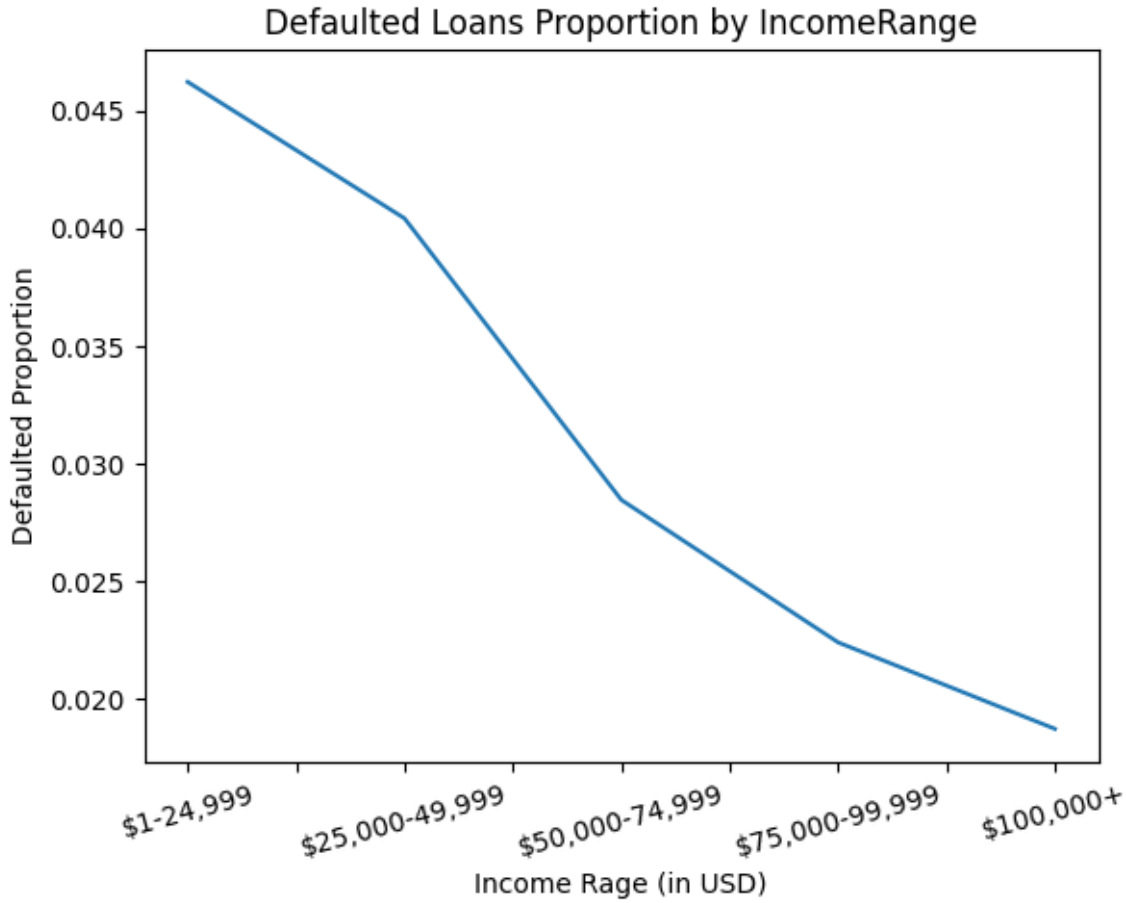


There is a slight inverse relation between the income range and the borrower rage (which is to be expected), but the wiskers of the box plot show that rates are pretty spread across the same ranges for all income ranges. Let's see how the income ranges fare in terms of defaulting ratios.

In [119…

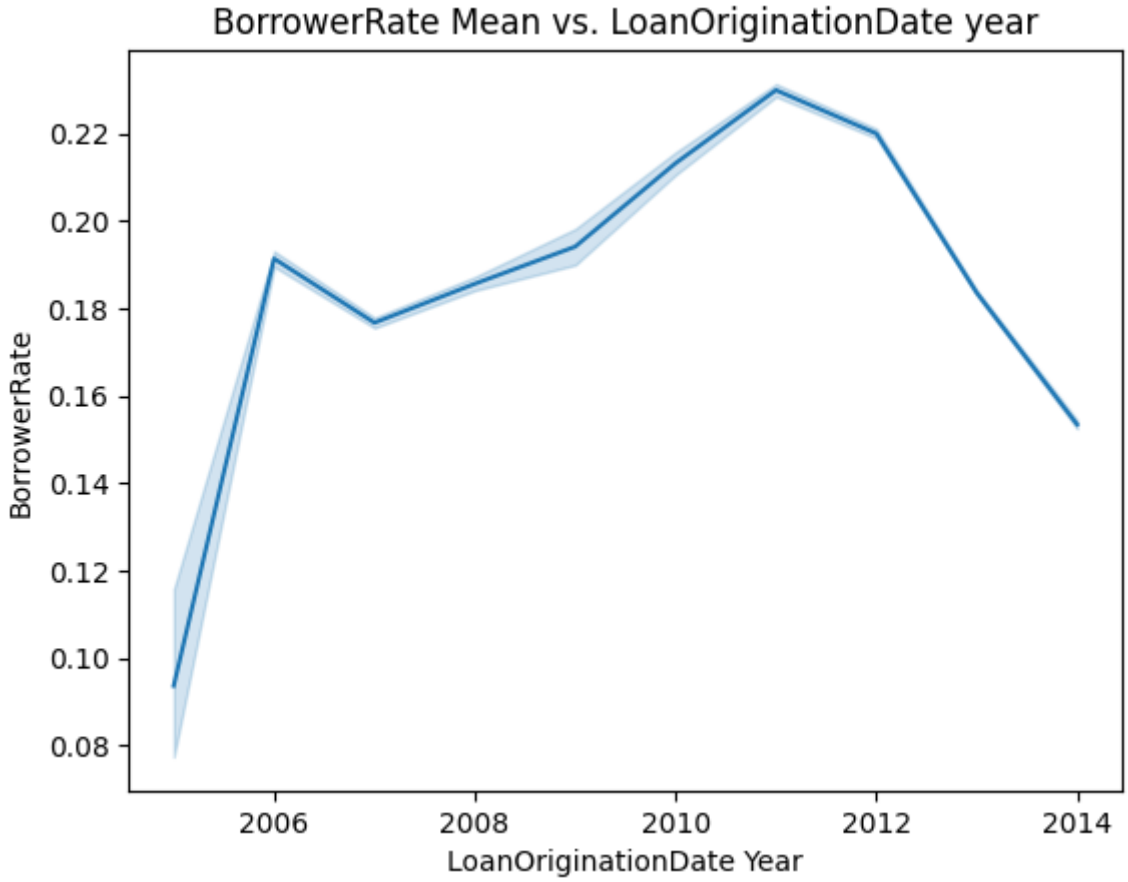
```
defaulted_prop_by_income_range = (
    prosper_loan_data_clean_df.ListingKey.groupby([
        # Group by income range and whether the deal was defaulted or not
        prosper_loan_data_clean_df.IncomeRange,
        prosper_loan_data_clean_df.LoanStatus == "Defaulted"
    ])
    .count()                        # Count the number of deals in each gr
    .groupby(level=0)               # Group by the income range
    .apply(lambda x: x / x.sum())  # Compute the defaulted ratio (summing
)
```

In [120…
```
defaulted_prop_by_income_range[
    ~defaulted_prop_by_income_range.index.get_level_values("IncomeRange")
][:, True].plot(kind="line");
plt.title("Defaulted Loans Proportion by IncomeRange")
plt.xlabel("Income Rage (in USD)")
plt.ylabel("Defaulted Proportion")
plt.xticks(rotation=15);
```



The chart shows that people with higher income ranges default less than those lower income ranges. Let's continue looking at what could affect the borrowing rate. Let's see if the average remained consistent throughout the years.

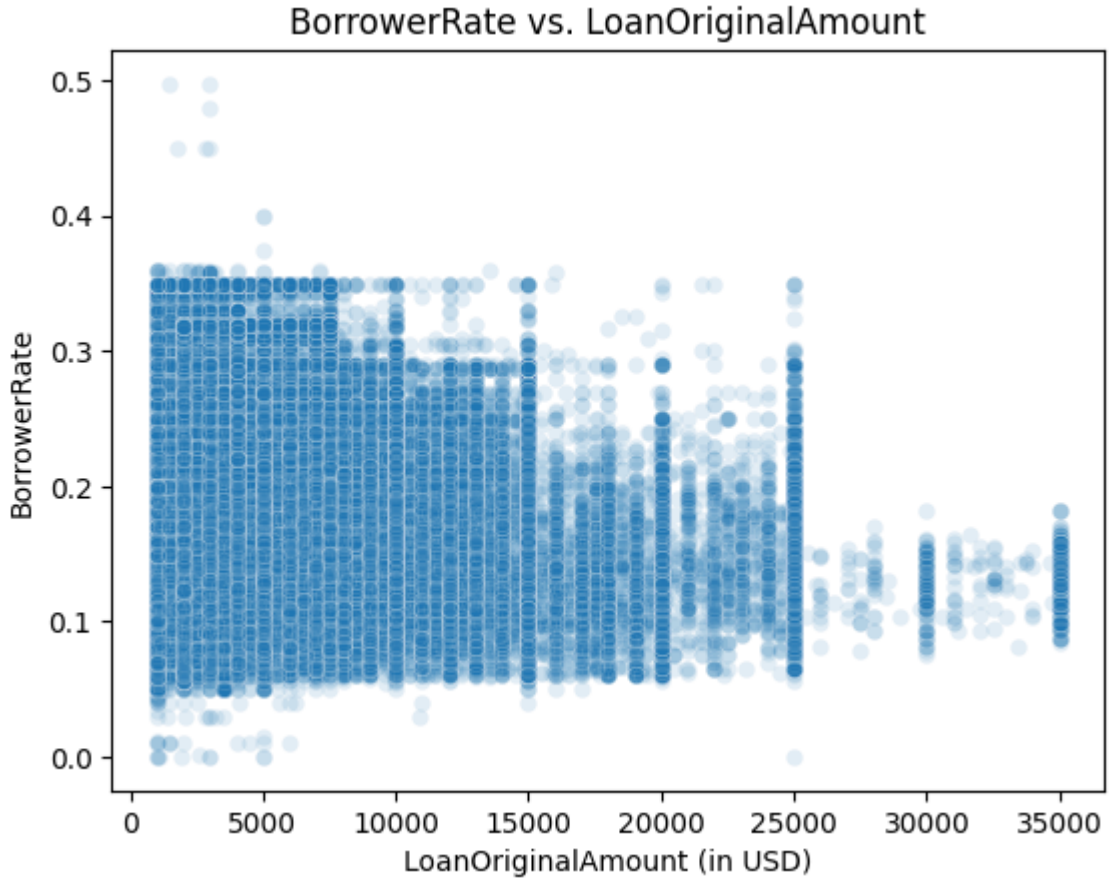
In [121…
```
sns.lineplot(
    x=prosper_loan_data_clean_df.LoanOriginationDate.dt.year,
    y=prosper_loan_data_clean_df.BorrowerRate,
    color=base_color
)
plt.xlabel("LoanOriginationDate Year")
plt.title("BorrowerRate Mean vs. LoanOriginationDate year");
```



The average borrower rate changed quite a lot over the observed period. There seems to be an external factor that affected the rate. Let's see if the loan amount affected the borrower rate.

In [122…

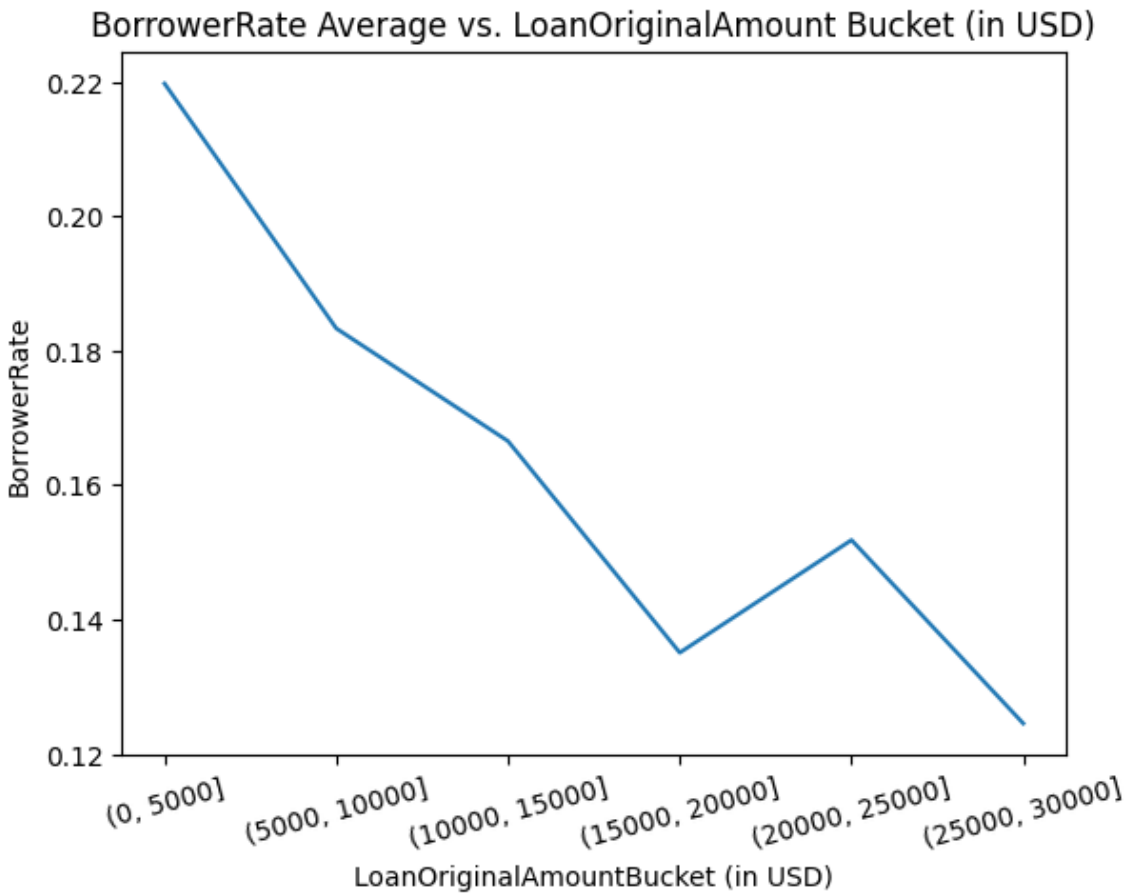
```
sns.scatterplot(
    data=prosper_loan_data_clean_df,
    x="LoanOriginalAmount",
    y="BorrowerRate",
    alpha=0.125
)
plt.xlabel("LoanOriginalAmount (in USD)")
plt.title("BorrowerRate vs. LoanOriginalAmount");
```



From this chart is hard to see a clear correlation. Let's group the loans by buckets of `LoanOriginalAmount` of 5000 USD, and take the average of each group.

In [123…
```
prosper_loan_data_clean_df["LoanOriginalAmountBucket"] =\
    pd.cut(prosper_loan_data_clean_df.LoanOriginalAmount, bins=np.arange(
```

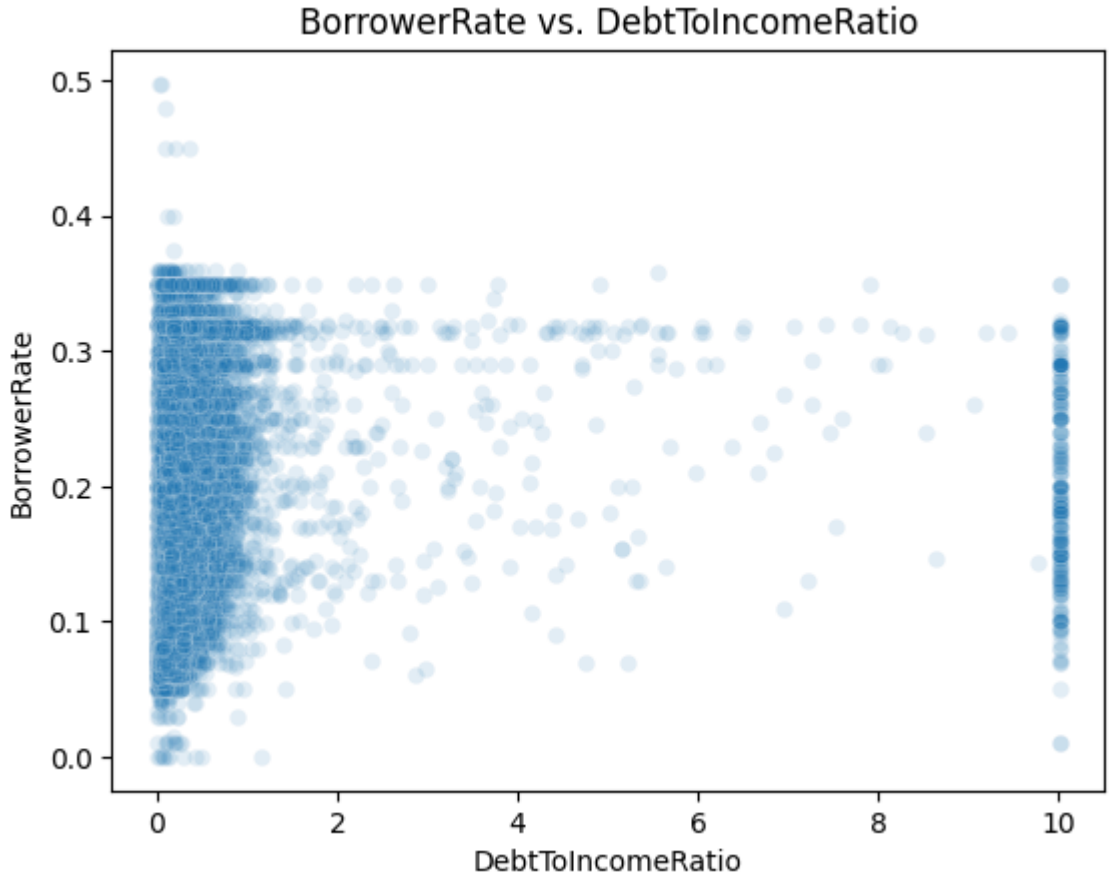
In [124…
```
prosper_loan_data_clean_df.groupby(
    "LoanOriginalAmountBucket").BorrowerRate.mean().plot(kind="line")
plt.title("BorrowerRate Average vs. LoanOriginalAmount Bucket (in USD)")
plt.xlabel("LoanOriginalAmountBucket (in USD)")
plt.xticks(rotation=15)
plt.ylabel("BorrowerRate");
```



Now the decrease in rate (on average) as the amount increases is more apparent. Let's focus on `DebtToIncomeRatio` now:

In [125…

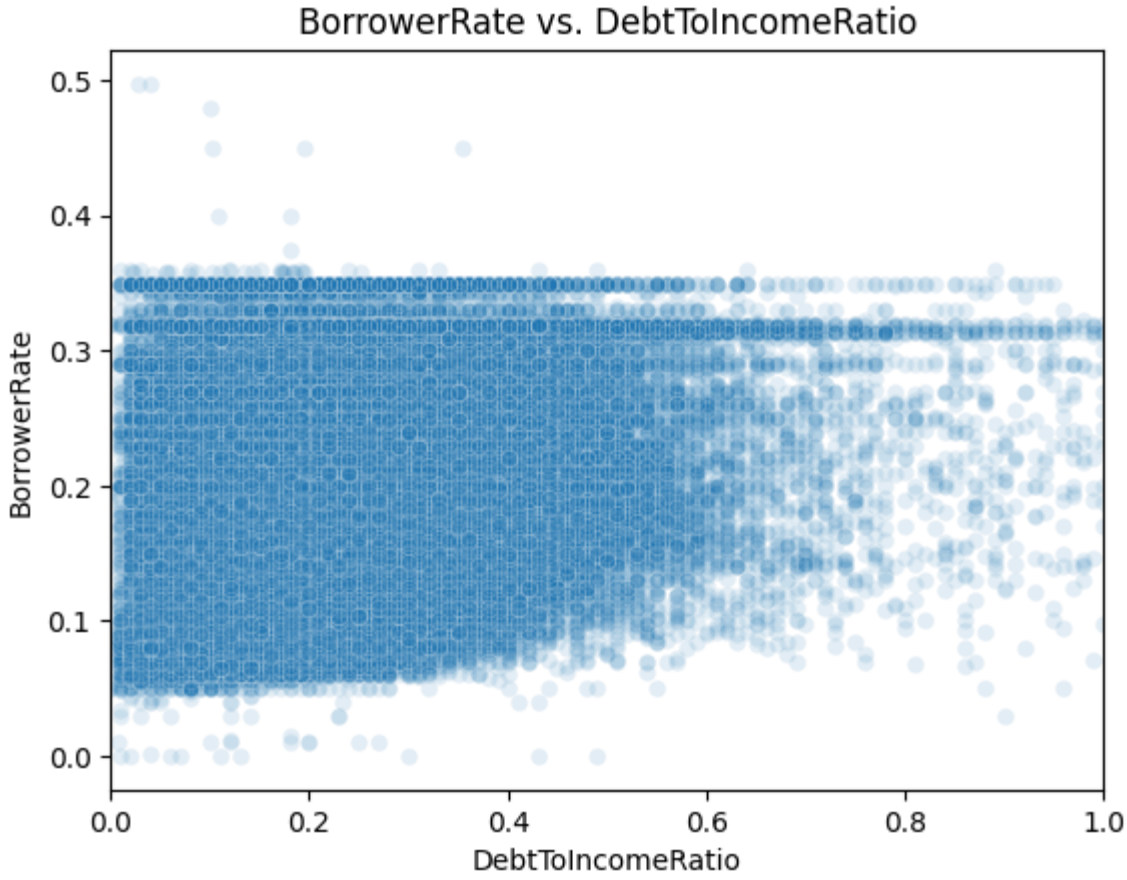
```
sns.scatterplot(
    data=prosper_loan_data_clean_df,
    x="DebtToIncomeRatio",
    y="BorrowerRate",
    alpha=0.125
)
plt.title("BorrowerRate vs. DebtToIncomeRatio");
```



Once again, there's no clear relation between the two variables. Let's focus on the ratios between 0 and 1 to see if there's a pattern there.

In [126…

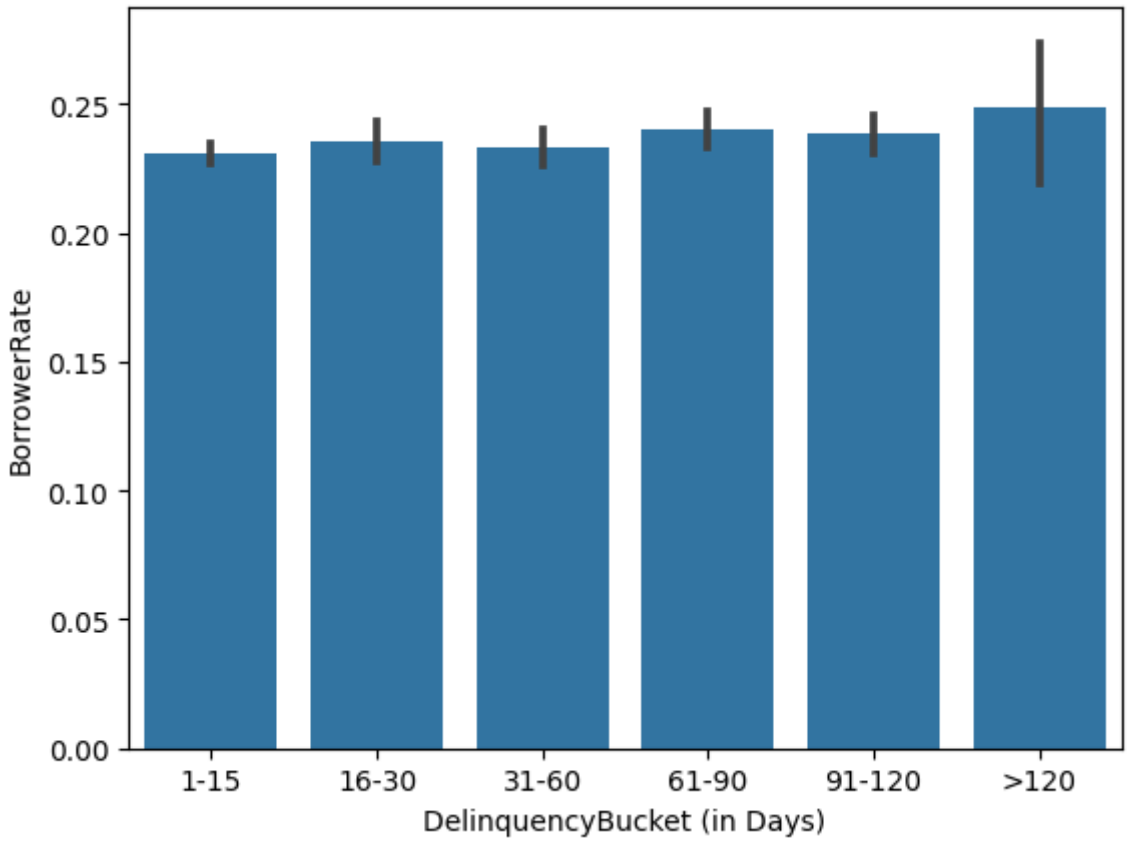
```
sns.scatterplot(
    data=prosper_loan_data_clean_df,
    x="DebtToIncomeRatio",
    y="BorrowerRate",
    alpha=0.125
)
plt.xlim((0, 1))
plt.title("BorrowerRate vs. DebtToIncomeRatio");
```



We can't really tell much from this chart either. Let's now see if there's a correlation between the number of days delinquent and the borrower rate.

In [127…

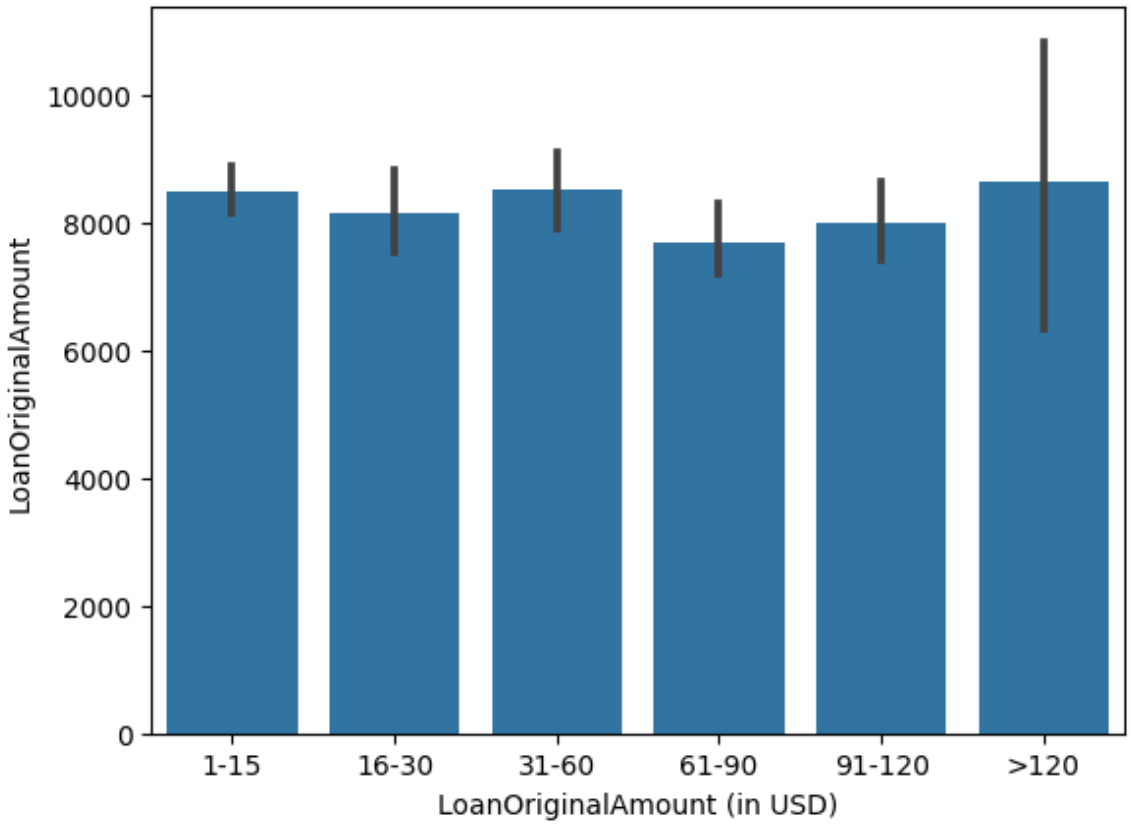
```
sns.barplot(
    data=prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanStatus
    x="DelinquencyBucket",
    y="BorrowerRate",
    color=base_color
)
plt.xlabel("DelinquencyBucket (in Days)");
```



The average rate remained consistent throughout the groups, with a slight increase in variability for the last one. Maybe there's a correlation with the amount?

In [128…

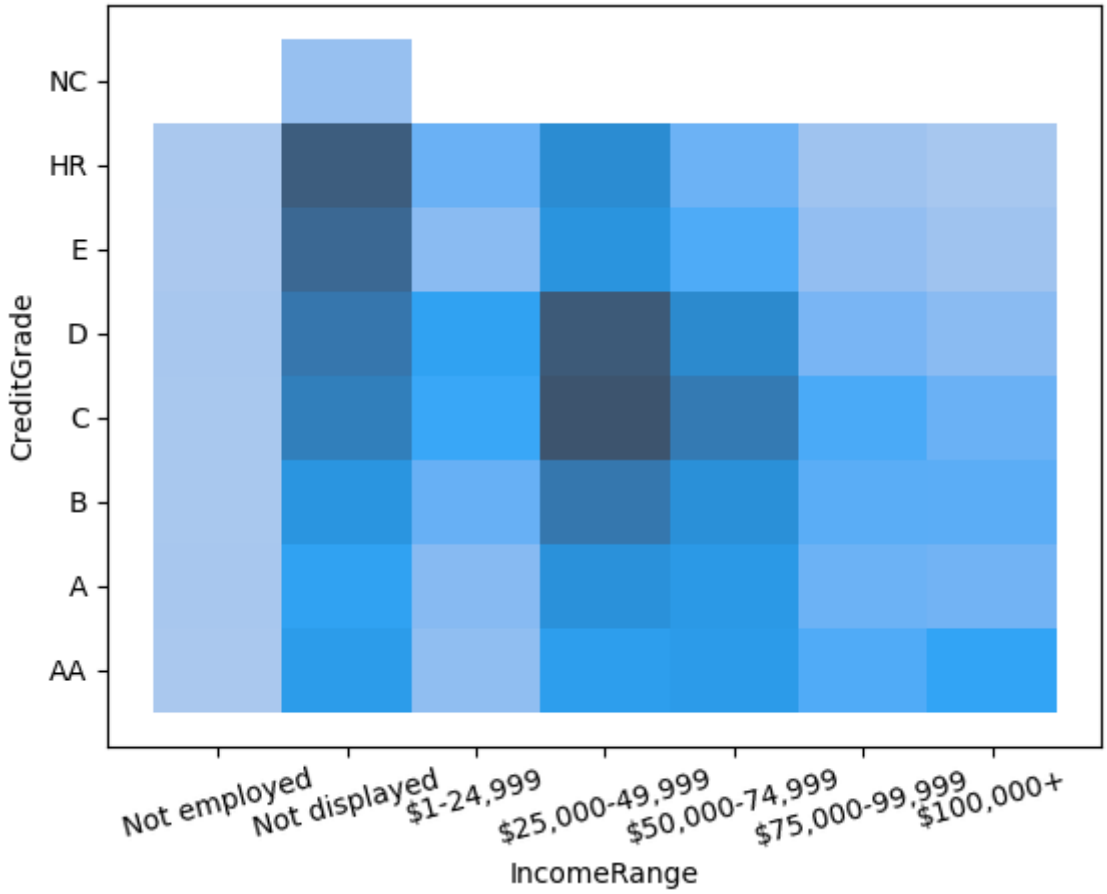
```
sns.barplot(
    data=prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanStatus
    x="DelinquencyBucket",
    y="LoanOriginalAmount",
    color=base_color
);
plt.xlabel("DelinquencyBucket (in Days)");
plt.xlabel("LoanOriginalAmount (in USD)");
```



Once again, the averages are comparable across all groups. Finally, let's see how the income range and credit grades are related.

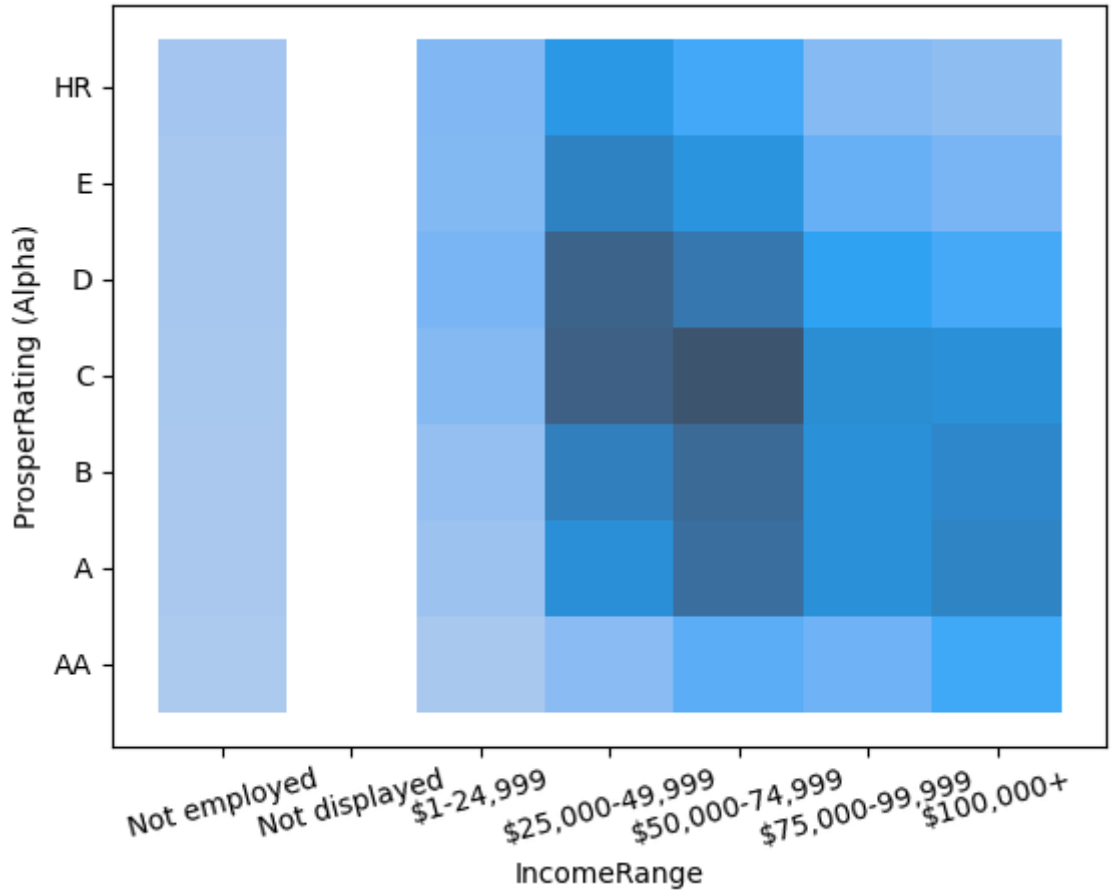
In [129…

```
sns.histplot(
    prosper_loan_data_clean_df[~prosper_loan_data_clean_df.CreditGrade.is
    x="IncomeRange",
    y="CreditGrade"
);
plt.xticks(rotation=15);
```



In [130…

```
sns.histplot(
    prosper_loan_data_clean_df[~prosper_loan_data_clean_df["ProsperRating
    x="IncomeRange",
    y="ProsperRating (Alpha)"
);
plt.xticks(rotation=15);
```



It seems the loans are lumped mostly around the mid-level ranges of the grade and income variables. On the `CreditGrade` chart we can see that loans with income range as "Not Displayed" are mostly of "High Risk".

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?

These are some of the observations from the multivariate analysis:

- BorrowerRage and BorrowerRate are highly correlated, but there's another variable that's grouping the loans
- Higher credit grades (for both variables) lead to lower borrower rates
- Higher income ranges also lead to lower rates, but only slightly
- Borrower rate average have not remained contant throughout the years

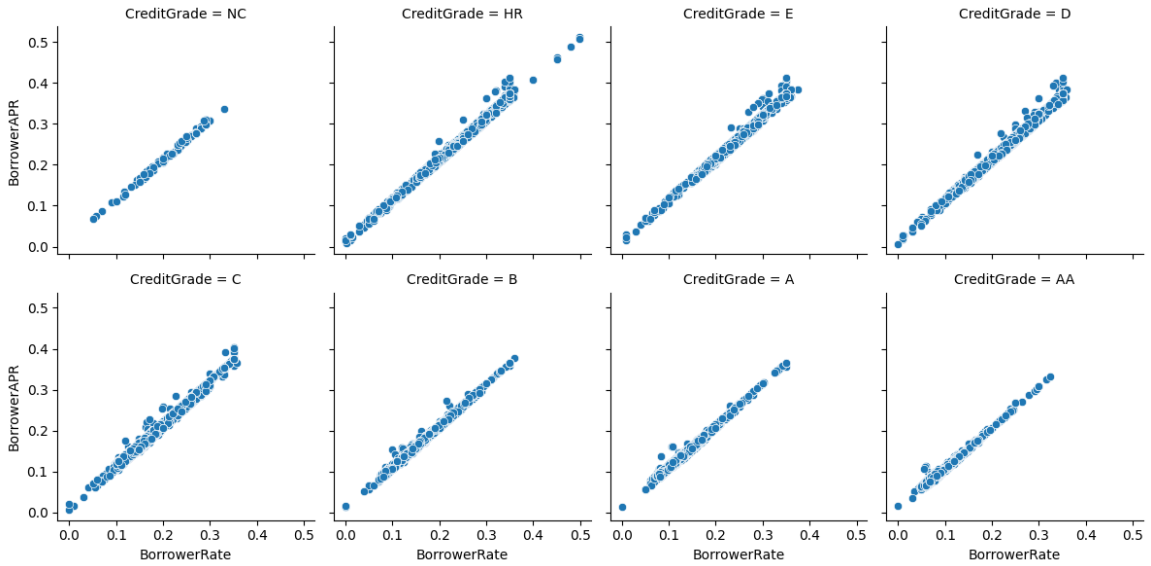
Did you observe any interesting relationships between the other features (not the main feature(s) of interest)?

The only thing of interest was that most loans are taken by people with middle credit grades and income ranges. I would have guessed that people with higher income ranges would have better grades.

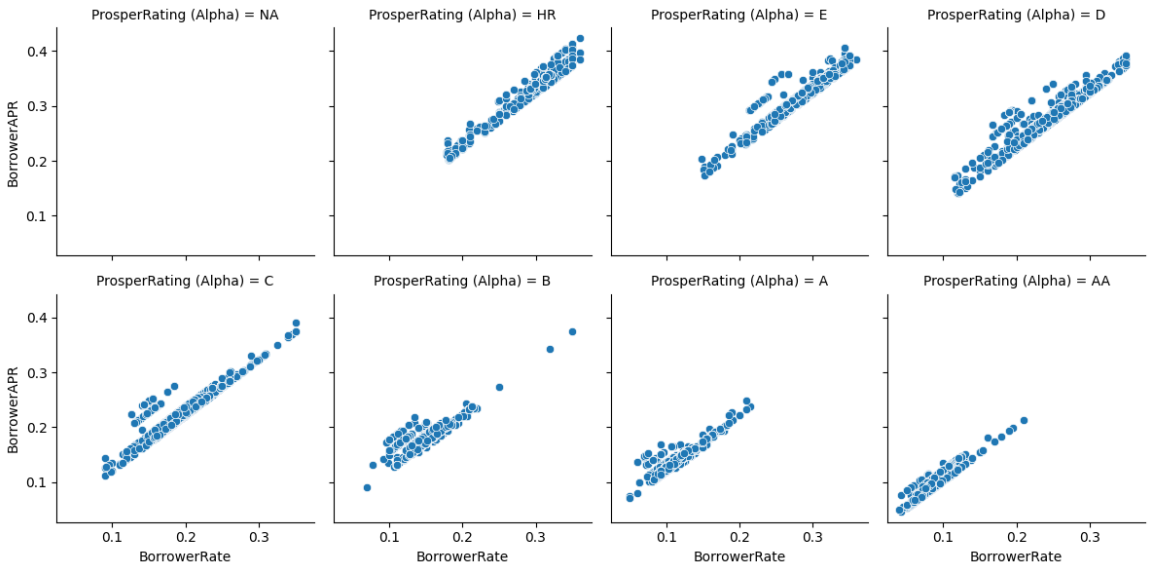
Multivariate Exploration

The first thing we need to try to explain is the grouping in the scatter plot for `BorrowerRate` and `BorrowerAPR`.

In [131…
```
g = sns.FacetGrid(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df.CreditGra
    col="CreditGrade",
    col_wrap = 4
)
g.map(sns.scatterplot, "BorrowerRate", "BorrowerAPR");
```

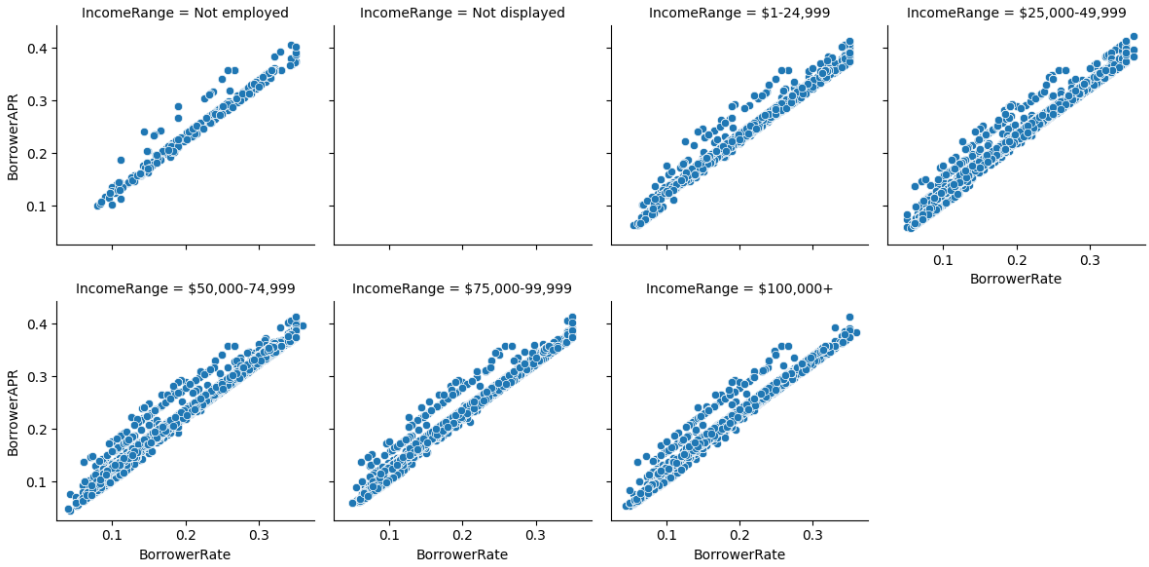


In [132…
```
g = sns.FacetGrid(
    data=prosper_loan_data_clean_df[~prosper_loan_data_clean_df["ProsperR
    col="ProsperRating (Alpha)",
    col_wrap = 4
)
g.map(sns.scatterplot, "BorrowerRate", "BorrowerAPR");
```



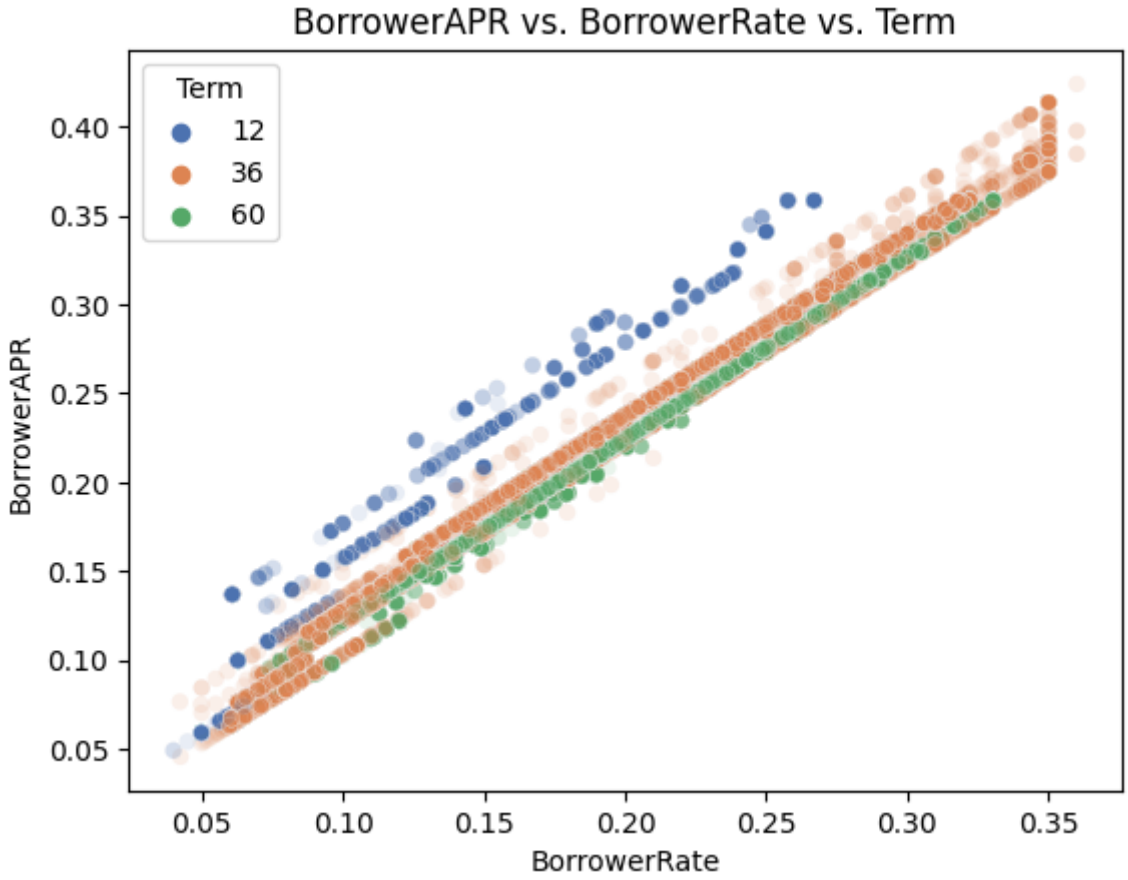
Interstingly enough, the stratification is only visible on deals with `ProsperRating (Alpha)` which are all from 2009 and after. This means that the credit grade is not creating this effect. Let's focus on loands from 2009 and after.

In [133…
```
g = sns.FacetGrid(
    data=prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanOrigin
    col="IncomeRange",
    col_wrap = 4
)
g.map(sns.scatterplot, "BorrowerRate", "BorrowerAPR");
```



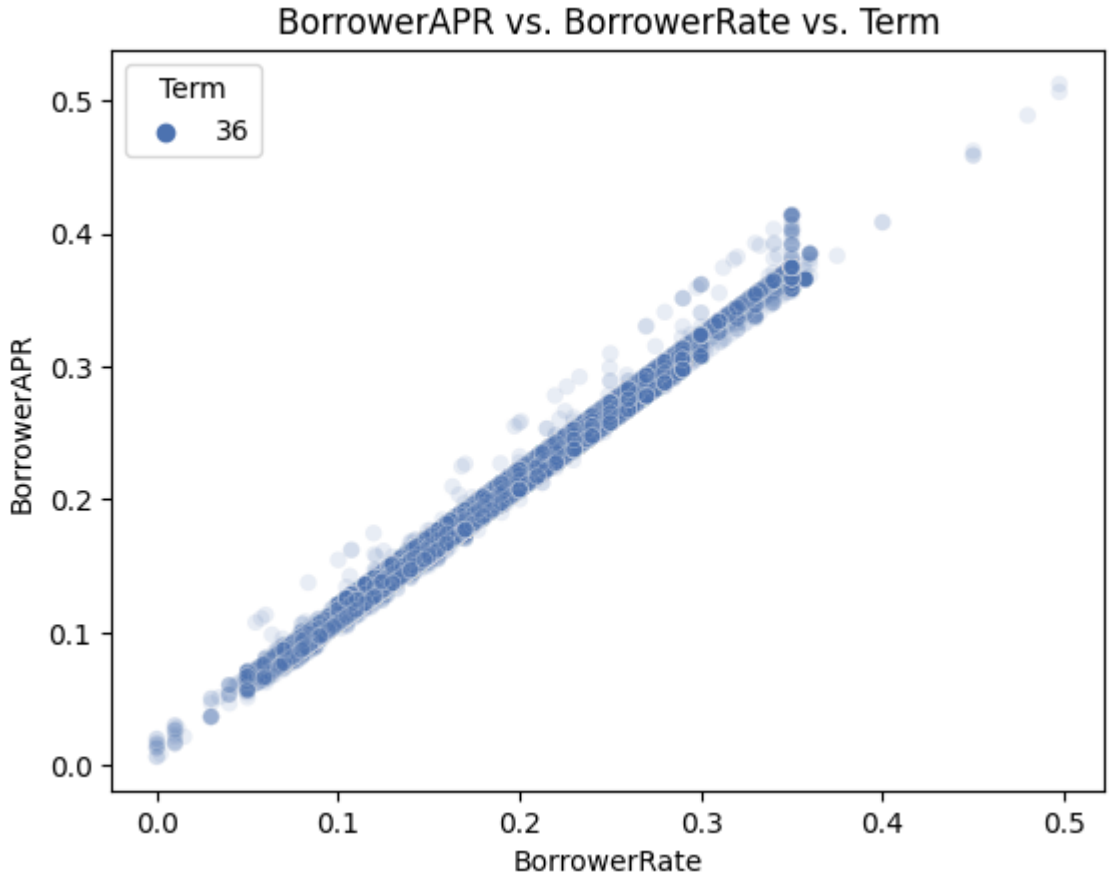
The income range does not seem to be the cause either.

In [134…
```
sns.scatterplot(
    data=prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanOrigin
    x="BorrowerRate",
    y="BorrowerAPR",
    hue="Term",
    alpha=0.125,
    palette="deep"  # Changing the palette to avoid the gradient associat
)
plt.title("BorrowerAPR vs. BorrowerRate vs. Term");
```



In [135…

```
sns.scatterplot(
    data=prosper_loan_data_clean_df[prosper_loan_data_clean_df.LoanOrigin
    x="BorrowerRate",
    y="BorrowerAPR",
    hue="Term",
    alpha=0.125,
    palette="deep"  # Changing the palette to avoid the gradient associat
)
plt.title("BorrowerAPR vs. BorrowerRate vs. Term");
```



We can now see one possible explanation of the stratification effect, where loans with shorter terms getting higher rates, but there's clearly some other variable affecting this relationship as there's even grouping within the same categories. What this means is that the `Term` is one of the factors that increases the APR over the rate. Sadly, all the deals prior to 2009 have a single `Term` value so the same analysis cannot be be performed for the full dataset.

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?

Initially we were trying to analyze the effect observed between `BorrowerAPR` and `BorrowerRate` based on what we saw during the bivarate analysis. By faceting over the credit rating variables we could observe the effect only occuring for loans after 2009. We would have never found this if we didn't perform a multi-variate analysis.

Were there any interesting or surprising interactions between features?

Although the `Term` certainly has an effect on the relationship between `BorrowerAPR` and `BorrowerRate`, it doesn't fully explain whet we're observing. We hadn't originally considered the term as having an effect, but thinking about our experience with the financial world, it is quite obvious that it should have.

Conclusions

Here's a summary of all the conclusions we reached during the exploratory analysis (We've **highlighted** the key ones):

- **There was a dip in number of loans on 2009**
- **Most loans are taken at the end of the year**
- **BorrowerRate, BorrowerAPR and LenderYield have similar multi-modal distributions**
- **Most loans are taken for debt consolidation**
- CreditGrade and ProsperRating have similar distributions with a higher number of loans on the mid-range grades
- **The vast majority of loans have 36 month terms**
- Most past due deals are in the 1 to 15 days bucket
- Unsurprisingly the employment status duration is a right-skewed unimodal distibution
- **Half the borrowers are home owners**
- **Loans are taken mostly on "round" amounts**
- **BorrowerRage and BorrowerRate are highly correlated, but there's another variable that's grouping the loans**
- **Higher credit grades (for both variables) lead to lower borrower rates**
- Higher income ranges also lead to lower rates, but only slightly
- Borrower rate average have not remained contant throughout the years
- **At least for deals after 2009, the term is a factor that increases the APR over the rate**

Although this dataset was in really good shape, requiring only minimal alterations, our analysis wasn't as extensive as we would have hoped for due to multiple reasons:

- The subject matter is quite complex. The financial world (and loans in particular) are notoriously obscure when it comes to the modelling. Exploring the details behind all the variables was beyond the scope of this project.
- Some variables, like the credit grades, split the dataset in two, so this limited global explorations and it's an indicator that perhaps the rest of the variables should be segmented as well.
- All the information is mostly about the loans, and not so much about the borrowers. Some information, like the age, is a factor that can affect some of the variables.

Another surprising aspect of the project, was the amount of work required. Exploring a few variables and their interactions took many hours. That being said, it was worth the effort as we learned a lot and had a lot of fun during the process.

In [ ]: