LendingClub Loans Pricing

HarvardX - PH125.9x Data Science Capstone

Emmanuel Rialland - https://github.com/Emmanuel_R8

November 12, 2019

Contents

List of Tables

List of Figures

Introduction

Lending Club (LC) is an American company listed on the New York stock exchange that provides a platform for peer-to-peer lending. Unlike banks, it does not take deposits and invest them. It is purely a matching system. Each loan is split into $25 that multiple investors can invest in. LC is remunerated by fees received from both sides. LC states that they have intermediated more than $50bln since they started operations. Further description of the company is easily available online numerous sources.

In order for investors to make the best investment decisions, LC make a historical dataset publicly available to all registered investors. This dataset is the subject of this report. It was downloaded from the Kaggle data science website[1].

The size of the dataset is rich enough that it could be used to answer many different questions. We decided for a focused approach. Following Chapter 5 of (Peng, 2012), we will first formulate the question we want to answer to guide our analysis.

The business model of LC is to match borrowers and investors. Naturally, more people want to receive money than part with it. An important limiting factor to LC's growth is the ability to attract investors, build a trusting relationship where, as a minimum first step, investors trust LC to provide accurate, transparent and reliable information of the borrowers. For this purpose, LC decided not only to provide extensive information about potential borrowers' profile, but also historical information about past borrowers' performance. This is, as we understand, one of the key purposes of this dataset. We decided to use the dataset for this very purpose. Essentially, the questions are: **given a borrower profile, is his/her rating appropriate in terms of risk of default? And if a default occurs, what is the expected recovery? The summary question is: given a borrower profile, is the risk/reward balance appropriate to commit funds?** In answering this question, we understand that LC allows investment of very granular amounts. Therefore, even an individual investor can diversify his/her loan and risk portfolio. It is not necessary to 'gamble' funds on a single borrower. This is exactly what institutional investors achieve through syndication (although on a very different scale, typically $10-25mln for a medium-size bank).

For this exercise, we made two simplifying (hopefully not simplistic) assumptions:

- In determining the risk/return balance, we have not accounted for LC's cost of intermediation. By ignoring fees paid by both sides, we obviously overestimate the returns to the investors. But in first approximation, **we will assume that the risk/reward balance, from the investors' point of view, across ratings is independent from fees.** This is a simplification. Real-world fees are higher the lower the investment grade and push the investors to receive, and the borrowers to pay, higher interest margin.
- All-in interest rates paid by borrowers are fixed. This is highly desirable for borrowers to be able to manage their cashflow. However, an investor should always consider an investment

[1] https://www.kaggle.com/wendykan/lending-club-loan-data/data

return as a margin above a risk-free return. Banks would look at LIBOR; bond investors (e.g. life insurers) would look at government bonds. Those risk-free rates can change very quickly, whereas we understand that LC sets those rates on a less frequent basis. In other word, the risk premium will vary rapidly. **We assume that individual investors are 'in-elastic' to change in implied risk premia.** But we recognise this as a limitation of our work.

This report is organised as follows:

- [XXXX]

Chapter 1

Dataset

The data is sourced as a *SQLite* database that downloaded from (Kan, 2019) and imported as a `tibble dataframe` with the `RSQLite` package. The variables were reformatted according to their respective types.

[We also sourced US zip and FIPS codes, and macroeconomical data for possible geographical statistics. The source code for the data import and reformatting is given in appendix.]

1.1 Preamble

The LendingClub dataset, although rich, is difficult to interpret. The only explanation of what the variables mean comes from a spreadsheet attached to the dataset. The explanations are not precise and/or subject to conflicting interpretation. Despite serching the LendingClub website, no further original information was found. We collected a number of reasonable assumptions in Appendix (see (List of Assumptions)[list-assumptions]).

The dataset has been used a number of times in the past by various people. One paper (Kim and Cho, 2019) mentions they used a dataset that included 110 variables, which is less than ours with 145 variables. The dataset has changed over time in ways we do not know. For example, have loans been excluded because the full 145 veriables were not available?

1.2 General presentation

The original dataset is large: it includes 2260668 loan samples, each containing 145 variables (after the identification variables filled with null values). The loans were issued from 2007-06-01 to 2018-12-01.

1.2.1 Business volume

The dataset represents a total of ca.$34bln in loan principals, which is a substantial share of the total amount stated to have been intermediated to date by LC (publicly reported to be $50bln+). About 55%/60% of the portfolio is fully repaid. See Table 1.1.

Figure 1.1 plots the number, volume (cumulative principal amount) and average principal per loan. It shows that the business grew exponentially (in the common sense of the word) from inception until 2016. At this point, according to Wikipedia [1]:

[1] source: https://en.wikipedia.org/wiki/LendingClub - Retrieval date 15 September 2019

Table 1.1: Number of loans per status

Loan status	Count	Proportion (%)
Charged Off	261655	11.574
Current	919695	40.682
Default	31	0.001
Does not meet the credit policy. Status:Charged Off	761	0.034
Does not meet the credit policy. Status:Fully Paid	1988	0.088
Fully Paid	1041952	46.090
In Grace Period	8952	0.396
Late (16-30 days)	3737	0.165
Late (31-120 days)	21897	0.969

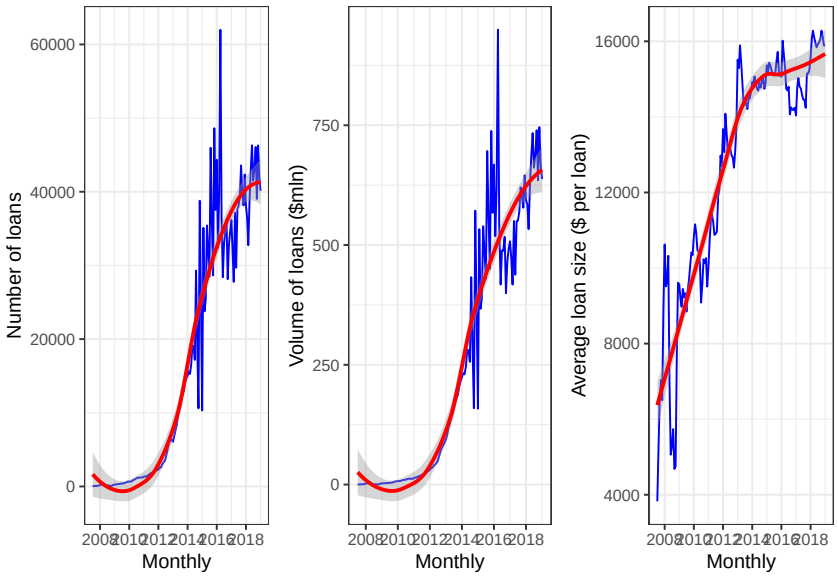

Figure 1.1: Business volume written per month

" *Like other peer-to-peer lenders including Prosper, Sofi and Khutzpa.com, LendingClub experienced increasing difficulty attracting investors during early 2016. This led the firm to increase the interest rate it charges borrowers on three occasions during the first months of the year. The increase in interest rates and concerns over the impact of the slowing United States economy caused a large drop in LendingClub's share price.*"

The number and volume of loans plotted have been aggregated by month. The growth is very smooth in the early years, and suddenly very volatile. As far as the first part of the dataset is concerned, a starting business could expect to be volatile and could witness a yearly cycle (expected from economic consumption figures) superimposed on the growth trend. This is not the case.

An interesting metric is that the average principal of loans has increased (see RHS Figure 1.1, on a sample of 100,000 loans). Partly, the increase in the early years could be interpreted success in improving marketing, distribution capabilities and confidence building. This metric plateau-ed in 2016 and decreased afterwards, but to a much lesser extent than the gross volume metrics. However, it is more volatile than the two previous metrics in the early years.

By the end of the dataset, those metrics have essentially recovered to their 2016 level.

1.2.2 Loan lifecyle and status

In the dataset, less loans are still outstanding than matured or "*charged off*" (term that LC use to mean partially or fully written off, i.e. there are no possibilty for LC and/or the investors to receive further payments). The share of outstanding loans is:

```
## Share of current loans =  42.214 %
```

The dataset describes the life cycle of a loan. In the typical (ideal) case, we understand it to be:

Loan is approved → Full amount funded by investors → Loan marked as Current → Fully Paid

In the worst case, it is:

Loan is approved → Full amount funded by investors → Loan marked as Current →

→ Grace period (missed payments under 2 weeks) → Late 15 to 31 days →

→ Late 31 to 120 days → Default → Charged Off

Note that *Default* precedes and is distinct from *Charged Off* [2]. A couple of things could happen to a loan in default:

- LC and the borrower restructure the loan with a new repayment schedule, where the borrower may repay a lesser amount over a longer period; or,
- the claim could be sold to a debt recovery company that would buy the claim from LC/investors. This would be the final payment (if any) received by LC and the investors.

[2] See LendingClub FAQ at and help page

The dataset also describes situations where a borrower negotiated a restructuring of the repayment schedule in case of unexpected hardship (e.g. disaster, sudden unemployment).

Note that this progression of distinguishing default (event in time) from actual financial loss mirrors what banks and rating agencies do. The former is called the *Probability of Default* (PD), the latter *Loss Given Default* (LGD). Ratings change over time (in a process resembling Markov Chains). LGD show some correlations with ratings. The dataset, although detailed, does not include the full life of each loan to conduct this sort of analysis (change of loan quality over time). This is an important reason why we decided to focus on the loan approval and expected return.

> CHANGE: Debt-Settlement companies [3] that explains that debt settlement companies can step in, buy the debt and pay to LC.
>
> CHANGE: What is the difference between a loan that is in "default" and a loan that has been "charged off"? [4]

1.2.3 Loan application

Before a loan is approved, the borrower undergoes a review process that assess his/her capacity to repay. This includes:

- employment situation and income, as well whether this income and possibly its source has been independently verified;
- whether the application is made jointly (likely with a partner or a spouse, but there are no details);
- housing situation (owner, owner with current mortgage, rental) and in which county he/she lives (that piece of information is partially anonymised by removing the last 2 digits of the borrower's zipcode);
- the amount sought, its tenor and the purpose of the loan; and,
- what seems to be previous credit history (number of previous deliquencies). The dataset is very confusing in that regard: it is clear that such information relates to before the loan is approved in the case of the joint applicant. In the case of the principal borrower however, the variable descriptions could be read as pre-approval information, or information gathered during the life of the loan. We have assumed that the information related to the principal borrower is also pre-approval. We also used *Sales Supplements* from the LC website[5] that describe some of the information provided to investors. LendingClub also provides a summary description of its approval process in its regulatory filings with the Securities Exchange Commission (California, 2019).

1.2.4 Interest rates

Based on this information, the loan is approved or not. Approval includes the final amount (which could be lower than the amount requested), tenor (3 or 5 years) and a rating similar to those given to corporate borrowers. Unlike corporate borrowers however, the rating mechanically determines the rate of interest according to a grid known to the borrower in advance[6]. The rates have changed over time. Those changes where not as frequent as market conditions (e.g. changes in Federal Reserve Bank's rates)[7].

[3] https://help.lendingclub.com/hc/en-us/articles/115011819087-Debt-settlement-companies
[4] https://help.lendingclub.com/hc/en-us/articles/216127747
[5] See https://www.lendingclub.com/legal/prospectus
[6] https://www.lendingclub.com/investing/investor-education/interest-rates-and-fees
[7] Corporate borrowers would negociate interest margins on a case-by-case basis despite similar risk profiles.

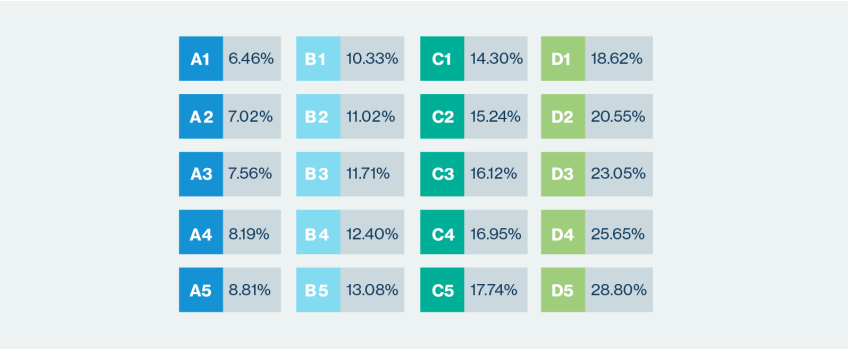

Figure 1.2: Interest rates given rating

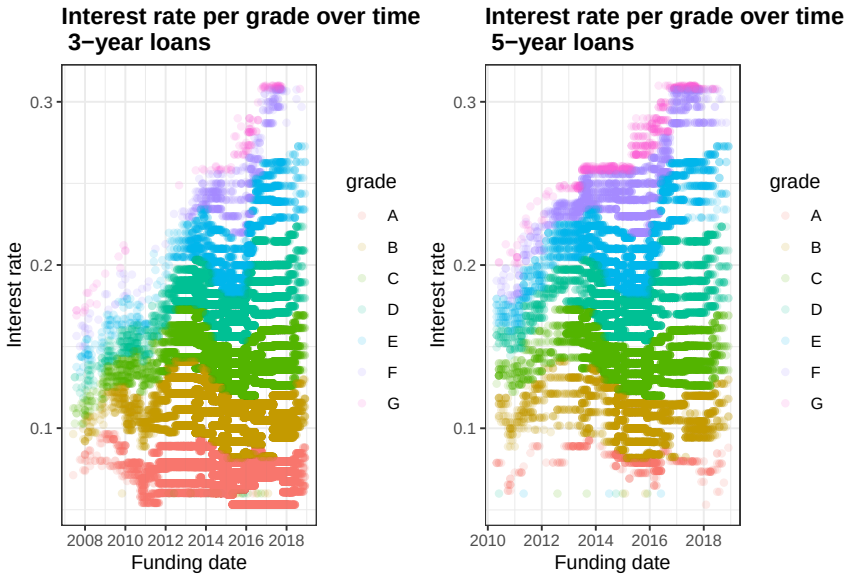

Figure 1.3: Interest rate per grade over time

Figure **??** [8] shows the predetermined interest rate depending on the initial rating as of July 2019.

At the date of this report, the ratings range from `A` (the best) down to `D`, each split in 5 sub-ratings. However, LC previously also intermediated loans rated F or G (until 6 November 2017) and E (until 30 June 2019) [9]. This explains that such ratings are in the dataset. We will assume that the ratings in the dataset are the rating at the time of approval and that, even if loans are re-rated by LC, the dataset does not reflect it.

Figures 1.3 shows the change in interest rate over time for different ratings and separated for each tenor. (Each figure is on a sample of 100,000 loans.) For each rating, we can see several parallel lines which correspond to the 5 sub-rating of each rating. We note that the range of interest rates has substantial widened over time. That is, the risk premium necessary to attract potential investors has had to substantially increase. In the most recent years, the highest rates exceed 30% which is higher than many credit cards.3-year loans are naturally considered safer (more A-rated, less G-rated). Identical ratings attract identical rates of interest.

By comparison, we plot the 3-year (in red) and 5-year (in blue) bank swap rates in Figure 1.4. We see that the swap curve has flattened in recent times (3-year and 5-y rates are almost identical). We also can see that in broad terms the interest rates charged reflect those underlying swap

[8] source: https://www.lendingclub.com/investing/investor-education/interest-rates-and-fees
[9] See https://www.lendingclub.com/info/demand-and-credit-profile.action

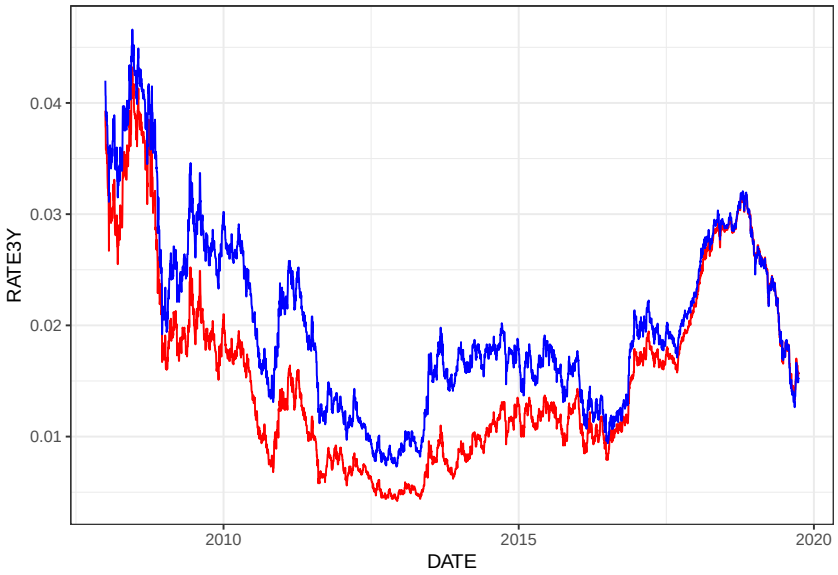

Figure 1.4: Historical Swap Rates

rates. It is therefore most relevant to examine the credit margins added to the swap rates.

Figures 1.5 shows the change in credit margin over time for different ratings and separated for each tenor. (Each figure is on a sample of 100,000 loans.) As above, for each rating, we can see several parallel lines which correspond to the 5 sub-rating of each rating. We note that the range of credit margins has widened over time but less than the interest rates. Identical ratings attract identical credit margins.

[TODO: DTI, amount... by grade]

1.2.5 Payments

The loans are approved for only two tenors, 3 and 5 years, with monthly repayments. Installments are calculated easily with the standard formula:

$$Installment = Principal \times rate \times \frac{1}{1 - \frac{1}{(1+rate)^N}}$$

Where $Principal$ is the amount borrowed, $rate = \frac{\text{Quoted Interest Rate}}{12}$ is the monthly interest rate, and N is the number of installments (36 or 60 monthly payments). The following piece of code shows that the average error between this formula and the dataset value is about 2 cents. We therefore precisely understand this variable.

```
local({
  installmentError <- loans %>%
    mutate(
      PMT = round(funded_amnt * int_rate / 12 / (1 - 1 / (1 + int_rate / 12) ^
                                                   term), 2),
      PMT_delta = abs(installment - PMT)
    ) %>%
    select(PMT_delta)

  round(mean(100 * installmentError$PMT_delta), digits = 2)
})
```

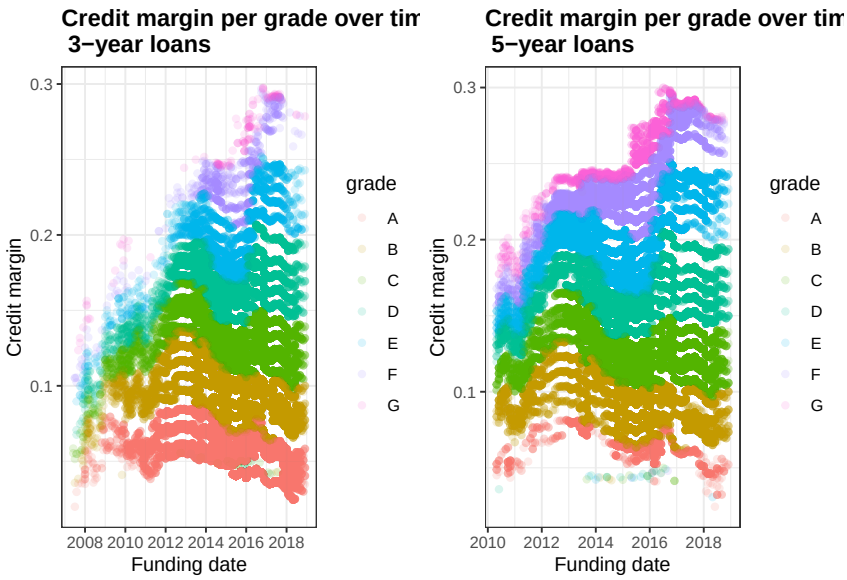

Figure 1.5: Credit margins per grade over time

1.3 Variables

We here present the dataset in a bit more details The full list of variable is given in appendix (see Table 5.1). This dataset will be reduced as we focused on our core question: *Are LC's loans priced appropriately?*.

1.3.1 General

1.3.2 Identification

The dataset is anonymised (all identifying ID numbers are deleted) and we therefore removed those columns from the dataset. Since the identification `IDs` have been removed to anonymise the dataset, we cannot see if a borrower borrowed several times.

1.4 Loan decision

As indicated in the introduction, our focus is on loans that have gone through their entire life cycle to consider their respective pricing, risk and profitability. To that effect, we will remove all loans which are still current (either performing or not). From here on, everything will be based on this reduced dataset.

In this reduced dataset, we focus on loans that have matured or been terminated. It contains 1306356 samples. Most of the loans (ca.80%) have been repaid in full. See Table 1.2.

When grouped by grade (Figure 1.6), we see a clear correlation between grade and default: the lower the grade the higher the portion defaults (all the way down to about 50%). In addition, most of the business is written in the B- or C-rating range.

1.5 Payment-related information

As mentioned previously, the descriptions of the dataset variables is at times incomplete or confusing. For the purpose of determining the cash flow of each individual loans, we have attempted to reconstruct the variables.

Table 1.2: Matured loans per status

Loan status	Count	Proportion (%)
Fully Paid	1041952	26048800
Charged Off	261655	6541375
Does not meet the credit policy. Status:Fully Paid	1988	49700
Does not meet the credit policy. Status:Charged Off	761	19025

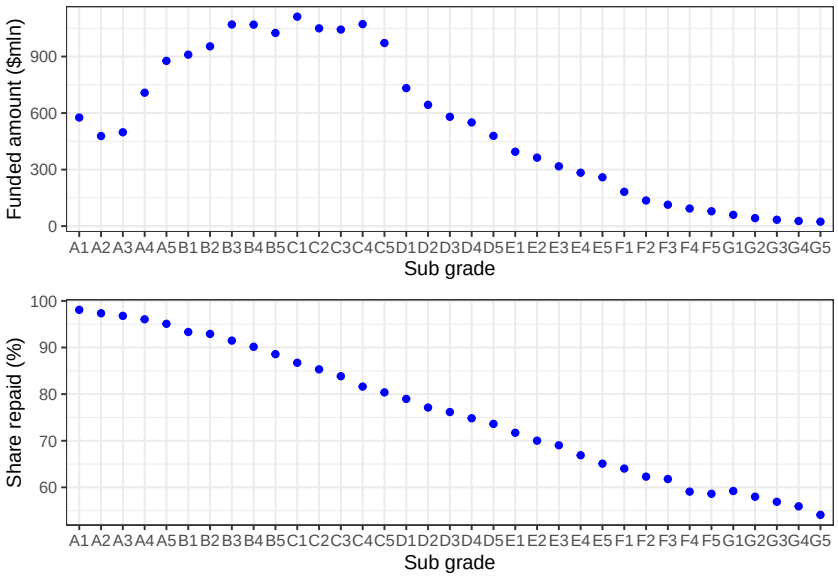

Figure 1.6: Funding and Write-offs by Sub-grades

We have verfied that:

- installments are calculated as per the formula shown above, rounded to the next cent.
- total_pymnt = total_rec_prncp + total_rec_int + total_rec_late_fee + recoveries

Chapter 2

Internal Rate of Return and Credit Margins

In this section, we describe the response variables that we will generate for each loan and that will be used in the modeling section.

As indicated in the introduction, our purpose is to test a model that predicts the financial risk of a loan.

2.1 Important warning

The calculations presented here are simplistic. Although they bear some resemblance to what financial institutions (*FIs*) do. The literature on credit assessment and pricing is very rich and very complex. Finding the optimal capital allocation to particular risks while at the same time satisfying internal risk policies and regulatory requirements is a problem that financial institutions have yet to solve in full. Investing in a loan is not only a matter of assessing the risk of a particular borrower, but also assessing systemic risks (which exist across all borrowers), risks associated with funding the loan (interest, currency and liquidity markets), each requiring a risk assessment and pricing.

In other words, nobody would, let alone should, make any investment decision based on the calculations below.

2.2 Background

This subsection can be skipped by anybody with basic financial knowledge.

A bird in hand or two in the bush; a penny today or a pound tomorrow. What is the price of delaying obtaining and owning something? This is what pricing a loan is about. A lender could keep his/her cash in hand, or lend it and have it in hand later. He/she would accept this in exchange for receiving a bit more: this is the rate of interest.

There are borrowers that one can see as completely safe (risk-free) such as central banks or governments of strong economies. A lender always has the possibility to lend to them instead of a more risky borrower. Therefore, a lender would require a higher interest rate than risk-free. The additional interest that a lender requires is commensurate with the risk of the borrower not repaying (called *credit worthiness*) and is called the *credit margin*.

For each individual borrower, an FI would assess information provided by the borrower and

massive amounts of historical data to answer the question: considering historical borrowers with a profile similar to the applicant's, what is the probability of not getting principal and interest back (PD)? And, in case the borrower stops paying and using additional courses of action (such as seizing and selling assets), what is the total loss that could be expected on average (LGD)?

Making that assessment, the FI would require an interest rate which would roughly be the sum of:

- the risk-free rate;
- a margin to cover the average loss of similar borrowers (see the important footnote[1]);
- a margin to cover all the operational costs of running their operations; and,
- a margin to remunerate the capital allocated by the FI (banking regulations require all banks to allocate an amount of capital against any risk taken; those are stipulated in a number of complex rules).

Said crudely, this total is the amount for the FI to get out of bed and look at a loan. Although this sounds like an exact science (for some definition of the word), it is not. At the end of the day, the FI will also have to contend with the competition, market liquidity (if there is a lot of money available to be lent, it brings prices down) and, critically, whether the borrower would at all be interested in accepting that cost.

Note that the dataset is distorted by this additional survival effect: the application information of many loans does not appear merely because the rate of interest was considered too high (this is not dissimilar to *survival effects* where some data did not survive through the history of a dataset[2]).

2.3 Internal Rate of Return

For the purpose of this report, we will simplify things enormously: we will only consider the first two components of the interest rate. The risk-free rate and the credit margin that would cover the cost of default/losses of individual borrowers.

The modeling exercise will focus on trying to approximate *on average* the credit margin given the information provided by the borrower.

With respect to a given loan and its cash flow, two calculations are important here: the Net Present Value (NPV) and the Internal Rate of Return (IRR). If we remember that an FI is indifferent to holding a principal P today or receiving it with an annual interest a year later (i.e. $P \times (1+r)$ where r is the annual rate of interest), we can say that any amount CF_1 received in a year is equivalent to $CF_0 = \frac{CF_1}{1+r}$ today. More generally, a steam of future cash receipts is worth:

$$NPV(r) = \sum_{Year i=1}^{Year=n} \frac{CF_i}{(1+r)^i}$$

The amount $NPV(r)$ is called the *Net Present Value* of the cash flow discounted at the rate r. Given that the LendingClub repayments are monthly, the formula becomes:

[1] It is important to realise that the average margin only brings the borrower back to having made no money at all on average: the additional income from the credit margin will be spent to cover average losses. In addition, we present the credit margin as income against borrower-specific losses. It does not address a lot of other risks such as correlation risks: a borrower might default because the economy as a whole gets worse, in which case many borrowers will default. This is a cyclical systemic risk similar to the 2007 US real estate crisis.

[2] A well-known example is historical stock prices which disappear when companies are de-listed or go bankrupt.

$$NPV(r) = \sum_{Monthi=1}^{Month=12\times n} \frac{CF_i}{(1+\frac{r}{12})^i}$$

If we now have a day 1 cash flow CF_0, we can calculate:

$$CF_0 - \sum_{Yeari=1}^{Year=n} \frac{CF_i}{(1+r)^i}$$

However, for any given CF_0, there is no reason that it would equal the NPV of the future cash flow (i.e no reason why the difference would be equal to zero). But this is an equation depending on r. If we can find a value of r that zeroes this formula, it is called the internal rate of return of the cash flow:

$$CF_0 - \sum_{Yeari=1}^{Year=n} \frac{CF_i}{(1+IRR(CF))^i} = 0$$

or for monthly cash flow:

$$CF_0 - \sum_{Monthi=1}^{Month=12\times n} \frac{CF_i}{(1+\frac{IRR(r)}{12})^i} = 0$$

2.4 Dataset

2.4.1 Calculation

We used the dataset to calculate the IRR of each loan. We used the following information for the dataset: `funded_amnt`(loan amount funded), `int_rate` (all-in interest rate), `term` (tenor of the loan in months), `total_pymnt` (total cumulative amount received from the borrower), `total_rec_prncp` (amount repaid allocated to principal repayment), `total_rec_int` (amount repaid allocated to interest payment), `recoveries` (any amount recovered later from the borrower) and `total_rec_late_fee` (any late payments fees paid by the borrower).

From that information, we recreated a cash flow for each loan. The R code is presented in appendix. Unfortunately, this code takes close to a full day to run on the entire dataset of completed loans (i.e. excluding all ongoing loans). This is just impractical for anybody to run to check this report and the resulting IRR results dataset is included in the Github repository. To make things practical, the dataset was actually created using code in Julia[3]. It is a direct translation of the R code, with a similar syntax (therefore very easy to follow). The Julia code runs about 500 times quicker (this is not a typo), or about 3 minutes. We appreciate that this is the departure from the assignment description.

Similarly, we calculate the credit margin required by each loan noting that:

$$\text{Risk-free} + \text{Credit Margin} = IRR(loan)$$

As noted in the previous section, risk-free rates change over time. When solving for the credit margin, we use the relevant risk-free rate.

[3] https://julialang.org/

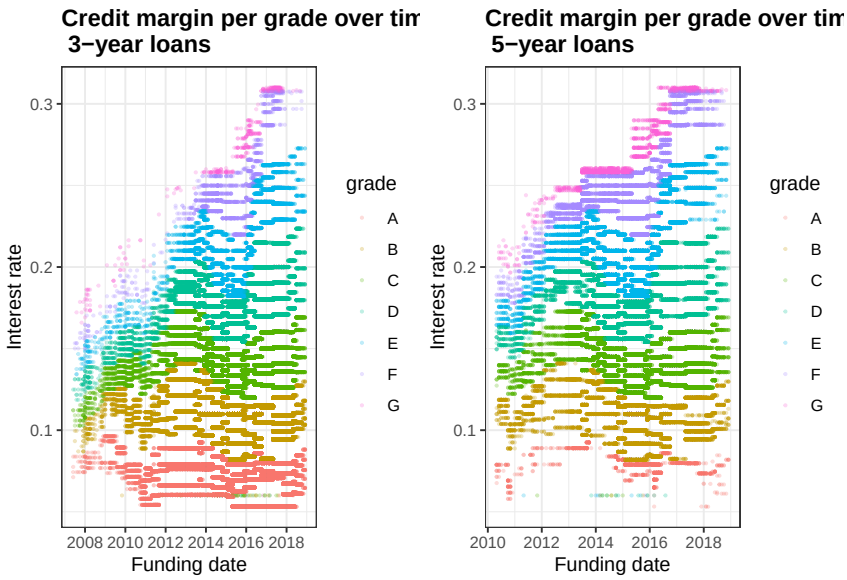

Figure 2.1: Credit margins per grade over time

Again, this was coded in Julia. The Julia code here takes about 1h20min to run. On the assumptions that the equivalent R code would take therefore almost 30 days to run through the full dataset, we did not write any R code for this calculation. The code is in appendix, and we believe easy to follow.

Figure 2.1 shows the evolution of credit margins over time grouped by ratings. The plots are made with a random sample of 300,000 loans.

We notice long periods where certain margins remain very stable which indicate that *both* the initial pricing was constant *and* that the proportion of default remains very low.

The graphs show considerations that are relevant to the modeling":

- The margins clearly change over time. Predictions will require to account for time [probably in a non-linear fashion].
- For a given rating, it widens and narrows over time. The changes happen in multiples that depends on the ratings:
  - For high quality / low margin loans: the changes are multiples of the margin, for example going from roughly 3% to 6/7%.
  - Although the range of change is wide, those changes do not happen very often, especially in the later years.
  - By comparison, for low quality / high margin loans, the range of change is smaller, but more frequent and volatile.
- In other words, the relation between loan quality (its rating) and its pricing (the credit margin) will significantly non-linear.

Chapter 3

Modelling

At the outset, the dataset presents a number of challenges:

- There is a mix of continuous and categorical data.
- The number of observations is very large.
- The possible number of predictors is large (partially due to one-hot encoding of categorical values).
- As shown in the previous section, credit margins have changed over time. This is clearly related to the wider US economic environment. Financial hardship is a key driver for some of the loans. Availability of disposable income is important to assess the ability to repay. Therefore, the cost of living, which varies from state to state, seems relevant.

3.1 Introduction

The dataset was randomly split into training and validation sets (80/20 ratio).

We initiated the exploration of potential models with Principal Component Analysis, Linear Regression, Extreme Boosting and Random Forest. No model could be trained on the full set. We therefore came to limit the training set on a random sample of 0.1% (1 thousandth) of the initial full set.

Any model will require training in batch (e.g. stochastic gradient descent) or online. Our untested intuition is that online methods are not adapted to an unbalanced dataset: the number of defaults/write-offs is fairly low for high quality ratings. However, the dataset is evidently a time series which points to online training. We will not explore that line of investigation, although exploring that tension would be an interesting subject.

Chapter 4

Stochastic Gradient Descent

The diagram 4.1[1] shows a useful decision tree.

Following the disappointing results of the previous section, we will explore a simpler model, but iteratively trained on a wider set of random samples.

4.1 Gradient descent

Gradient descent is a generic numerical optimisation algorithm to iteratively converge towards a (sometimes local) minumum of a given function. It is extensively used in statistical learning to minimise error functions.

In the case of a simple linear regression model, the model training error J (the **cost function**) as a function of θ is:

$$J_{(\theta)} = \frac{1}{2} \sum_{i=1}^{N} \epsilon(y_i, \theta X_i)$$

where the model parameters are denoted θ_i, $X_i \in \mathbb{R}^{\kappa}$ are the predictors, $Y_i \in \mathbb{R}^{\kappa}$ are the responses and ϵ is a distance function. Typically, ϵ will be the Manhattan error or the Euclidian norm ($A = (a_1, \cdots, a_n), B = (b_1, \cdots, b_n)$).

Manhattan: $\epsilon(A, B) = \sum_{i=1}^{n} |a_i - b_i|)\$$

Euclidian norm: $\epsilon(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$

The gradient descent algorithm uses the gradient of the error function, $\nabla J_{(\theta)}$, defined as:

$$\nabla J_{(\theta)} = \left(\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}, \cdots, \frac{\partial J}{\partial \theta_p} \right)$$

And in the case of linear regression is in a matrix form that can be computed efficiently:

$$\nabla J_{(\theta)} = \left(y^T - \theta X^T \right) X$$

The gradient decent algorithm finds parameters in the following manner iterating over the training samples:

[1] Source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

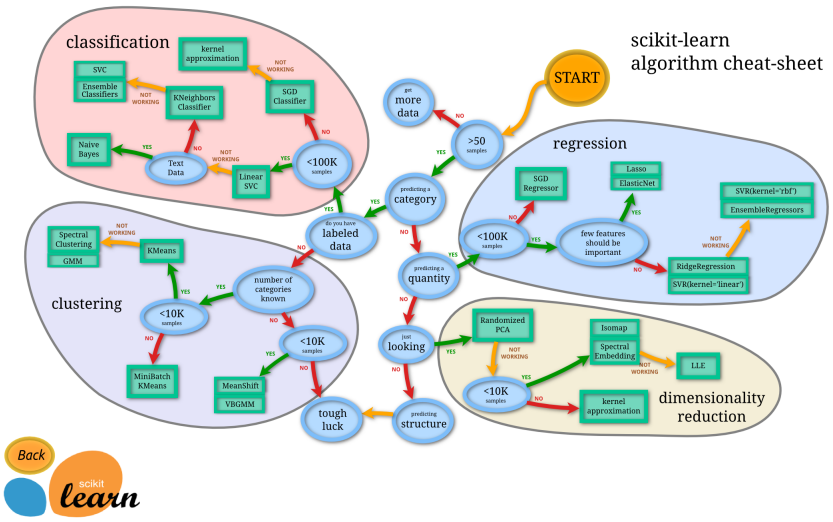

Figure 4.1: Scikit Learn algorithm cheat-sheet

While $||\alpha \nabla J_{(\theta)}|| > \eta$, $\theta := \theta - \alpha \nabla J_{(\theta)}$

In practice, the cost function will add a penalty term to regularise the model parameters (see below).

4.2 Stochastic Gradient Descent

With realistic datasets, gradient descent can experience slow convergence because (1) each iteration requires calculation of the gradient for every single training example, and (2) since each individual sample is potentially very different from another, the calculated gradient may not be optimal. In such case, the gradient descent can be done using a batch of several training samples and use the average of the cost function (**batch gradient descent**). This addresses those two sources of inefficiency.

This method however still requires iterating over the entire dataset. We can instead iterate over batched of random training samples drawn from the entire dataset, instead of being drawn sequentially. This is the **stochastic gradient descent**.

Aside from the choice of the initial choice of samples, and the averaging of the cost function, the update of θ remains identical.

4.2.0.1 Cost function regularisation and derivative

Regularisation is a way to decrease the complexity of models by narrowing the range that parameters can take. It has long been established that it assists the stability and robustness of learning algorithms. See Chapter 13 of (Shalev-Shwartz and Ben-David, 2014).

Our regularised cost function is written:

[TODO]

$$J_{P,Q} = \sum_{(i,j)\in\Omega} \left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k} q_{j,k} \right)^2 + \frac{\lambda}{2} \left(\sum_{i,k} p_{i,k}^2 + \sum_{j,k} q_{j,k}^2 \right)$$

The gradient descent algorithm seeks to minimise the $J_{(\theta)}$ cost function by step-wise update of each model parameter x as follows:

$$\theta_{t+1}^{i} \leftarrow \theta_{t}^{i} - \alpha \frac{\partial J_{(\theta)}}{\partial \theta_i}$$

The parameters are the matrix coefficients $p_{i,k}$ $q_{j,k}$. α is the learning parameter that needs to be adjusted.

4.2.0.2 Cost function partial derivatives

The partial derivatives of the cost function is:

$$\frac{\partial J_{P,Q}}{\partial x} = \frac{\partial}{\partial x}\left(\sum_{(i,j)\in\Omega}\left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}\right)^2 + \frac{\lambda}{2}\left(\sum_{i,k} p_{i,k}^2 + \sum_{j,k} q_{j,k}^2\right)\right)$$

$$\frac{\partial J_{P,Q}}{\partial x} = \sum_{(i,j)\in\Omega} 2\frac{\partial r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}}{\partial x}\left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}\right) + \frac{\lambda}{2}\left(\sum_{i,k} 2\frac{\partial p_{i,k}}{\partial x}p_{i,k} + \sum_{j,k} 2\frac{\partial p_{i,k}}{\partial x}q_{j,k}\right)$$

We note that $r_{i,j}$ are constants

$$\frac{\partial J_{P,Q}}{\partial x} = 2\sum_{(i,j)\in\Omega}\sum_{k=1}^{N_{features}}\left(\frac{\partial - p_{i,k}q_{j,k}}{\partial x}\left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}\right)\right) + \lambda\left(\sum_{i,k}\frac{\partial p_{i,k}}{\partial x}p_{i,k} + \sum_{j,k}\frac{\partial p_{i,k}}{\partial x}q_{j,k}\right)$$

If x is a coefficient of P (resp. Q), say $p_{a,b}$ (resp. $q_{a,b}$), all partial derivatives will be nil unless for $(i,j) = (a,b)$.

Therefore:

$$\frac{\partial J_{P,Q}}{\partial p_{a,b}} = -2\sum_{(i,j)\in\Omega} q_{j,b}\left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}\right) + \lambda p_{a,b}$$

and,

$$\frac{\partial J_{P,Q}}{\partial q_{a,b}} = -2\sum_{(i,j)\in\Omega} p_{i,b}\left(r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}\right) + \lambda q_{a,b}$$

Since $\epsilon i,j = r_{i,j} - \sum_{k=1}^{N_{features}} p_{i,k}q_{j,k}$ is the rating prediction error, this becomes:

$$\frac{\partial J_{P,Q}}{\partial p_{a,b}} = -2\sum_{(i,j)\in\Omega} q_{j,b}\epsilon_{i,j} + \lambda p_{a,b}$$

and,

$$\frac{\partial J_{P,Q}}{\partial q_{a,b}} = -2\sum_{(i,j)\in\Omega} p_{i,b}\epsilon_{i,j} + \lambda q_{a,b}$$

We note that the recent implementations of autodifferentiation algorithm (key to deep learning implementations) would avoid making those calculations by hand. We did not attempt to use the R `madness` package [2].

4.2.1 Stochastic Gradient Descent (SGD)

The size of the datasets is prohibitive to do those calculations across the entire training set.

Instead, we will repeatedly update the model parameters on small random samples of the training set.

Chapter 14 of (Shalev-Shwartz and Ben-David, 2014) gives an extensive introduction to various SGD algorithms.

We implemented a simple version of the algorithm and present the code in more detail.

```
# The algorithm is implemented from scratch and relies on nothing but the `Tidyverse` libraries.
library(tidyverse)

# The quality of the training and predictions is measured by the _root mean squared error_
# (RMSE), for which we define a few helper functions (the global variables are defined
# later):
rmse_training <- function() {
  prediction_Z <- rowSums(LF_Model$P[tri_train$userN, ] *
                            LF_Model$Q[tri_train$movieN, ])
  prediction <- prediction_Z * r_sd + r_m
  sqrt(sum((tri_train$rating - prediction) ^ 2 / nSamples))
}

rmse_validation <- function() {
  prediction_Z <- rowSums(LF_Model$P[tri_test$userN, ] *
                            LF_Model$Q[tri_test$movieN, ])
  prediction <- prediction_Z * r_sd + r_m
  sqrt(sum((tri_test$rating - prediction) ^ 2) / nTest)
}

sum_square <- function(v) {
  return (sqrt(sum(v ^ 2) / nrow(v)))
}

# The key function updates the model coefficients. Its inputs are:
#
# + a list that contains the $P$ an $Q$ matrices, the training RMSE of those matrices, and
# a logical value indicating whether this RMSE is worse than what it was before the update
# (i.e. did the update diverge).
#
# + a `batch_size` that defines the number of samples to be drawn from the training set. A
# normal gradient descent would use the full training set; by default we only use 10,000
# samples out of 10 million (one tenth of a percent).
#
# + The cost regularisation `lambda` and gradient descent learning parameter `alpha`.
#
# + A number of `times` to run the descent before recalculating the RMSE and exiting the
# function (calculating the RMSE is computationally expensive).
#
#
# The training set used is less rich than the original set. As discussed, it only uses the
```

[2] https://github.com/shabbychef/madness

```r
# rating (more exactly on the z_score of the rating). Genres, timestamps,... are
# discarded.


# Iterate gradient descent
stochastic_grad_descent <- function(model,
                                    times = 1,
                                    batch_size = 10000,
                                    lambda = 0.1,
                                    alpha = 0.01,
                                    verbose = TRUE) {
  # Run the descent `times` times.
  for (i in 1:times) {
    # Extract a sample of size `batch_size` from the training set.
    spl <- sample(1:nSamples, size = batch_size, replace = FALSE)
    spl_training_values <- tri_train[spl, ]


    # Take a subset of `P` and `Q` matching the users and
    # movies in the training sample.
    spl_P <- model$P[spl_training_values$userN, ]
    spl_Q <- model$Q[spl_training_values$movieN, ]

    # rowSums returns the cross-product for a given user and movie.
    # err is the term inside brackets in the partial derivatives
    # calculation above.
    err <- spl_training_values$rating_z - rowSums(spl_P * spl_Q)

    # Partial derivatives wrt p and q
    delta_P <- -err * spl_Q + lambda * spl_P
    delta_Q <- -err * spl_P + lambda * spl_Q

    model$P[spl_training_values$userN, ]  <- spl_P - alpha * delta_P
    model$Q[spl_training_values$movieN, ] <- spl_Q - alpha * delta_Q

  }

  # RMSE against the training set
  error <- sqrt(sum((
    tri_train$rating_z - rowSums(model$P[tri_train$userN, ] *
                                   model$Q[tri_train$movieN, ])
  ) ^ 2)
  / nSamples)

  # Compares to RMSE before update
  model$WORSE_RMSE <- (model$RMSE < error)
  model$RMSE <- error

  # Print some information to keep track of success
  if (verbose) {
    cat(
      "  # features=",
      ncol(model$P),
      "  J=",
      nSamples * error ^ 2 +
        lambda / 2 * (sum(model$P ^ 2) + sum(model$Q ^ 2)),
      "  Z-scores RMSE=",
      model$RMSE,
```

```
      "\n"
    )
    flush.console()
  }

  return(model)
}


rm(list_results)
list_results <- tibble(
  "alpha" = numeric(),
  "lambda" = numeric(),
  "nFeatures" = numeric(),
  "rmse_training_z_score" = numeric(),
  "rmse_training" = numeric(),
  "rmse_validation" = numeric()
)

# The main training loop runs as follows:
#
# + We start with 3 features.
#
# + The model is updated in batches of 100 updates. This is done up to 250 times. At each
# time, if the model starts diverging, the learning parameter ($\alpha$) is reduced.
#
# + Once the 250 times have passed, or if $\alpha$ has become incredibly small, or if the
# RMSE doesn't really improve anymoe (by less than 1 millionth), we add another features
# and start again.

initial_alpha <- 0.1
for (n in 1:100) {
  # Current number of features
  number_features <- ncol(LF_Model$P)

  # lambda = 0.01 for 25 features, i.e. for about 2,000,000 parameters.
  # We keep lambda proportional to the number of features
  lambda <- 0.1 * (nUsers + nMovies) * number_features / 2000000

  alpha <- initial_alpha

  cat(
    "CURRENT FEATURES: ",
    number_features,
    "---- Pre-training validation RMSE = ",
    rmse_validation(),
    "\n"
  )

  list_results <- list_results %>% add_row(
    alpha = alpha,
    lambda = lambda,
    nFeatures = number_features,
    rmse_training_z_score = LF_Model$RMSE,
    rmse_training = rmse_training(),
    rmse_validation = rmse_validation()
  )

```

```r
for (i in 1:250) {
  pre_RMSE <- LF_Model$RMSE
  LF_Model <- stochastic_grad_descent(
    model = LF_Model,
    times = 100,
    batch_size = 1000 * number_features,
    alpha = alpha,
    lambda = lambda
  )

  list_results <- list_results %>% add_row(
    alpha = alpha,
    lambda = lambda,
    nFeatures = number_features,
    rmse_training_z_score = LF_Model$RMSE,
    rmse_training = rmse_training(),
    rmse_validation = rmse_validation()
  )

  if (LF_Model$WORSE_RMSE) {
    alpha <- alpha / 2
    cat("Decreasing gradient parameter to: ", alpha, "\n")
  }

  if (initial_alpha / alpha > 1000 |
      abs((LF_Model$RMSE - pre_RMSE) / pre_RMSE) < 1e-6) {
    break()
  }
}


# RMSE against validation set:
rmse_validation_post <- rmse_validation()
cat(
  "CURRENT FEATURES: ",
  number_features,
  "---- POST-training validation RMSE = ",
  rmse_validation_post,
  "\n"
)

# if (number_features == 12){
#   break()
# }


# Add k features
k_features <- 1
LF_Model$P <- cbind(LF_Model$P,
                    matrix(
                      rnorm(
                        nrow(LF_Model$P) * k_features,
                        mean = 0,
                        sd = sd(LF_Model$P) / 100
                      ),
                      nrow = nrow(LF_Model$P),
                      ncol = k_features
                    ))
```

```
216
217   LF_Model$Q <- cbind(LF_Model$Q,
218                       matrix(
219                         rnorm(
220                           nrow(LF_Model$Q) * k_features,
221                           mean = 0,
222                           sd = sd(LF_Model$Q) / 100
223                         ),
224                         nrow = nrow(LF_Model$Q),
225                         ncol = k_features
226                       ))
227
228 }
229
230 saveRDS(list_results, "datasets/LRMF_results.rds")
```

Chapter 5

Conclusion

Is the interest rate proposed by LC high enough?

We use the entire dataset to estimate the impact of time as a polynomial curve. To assess future loans, this would not be acceptable. Only an online algorithm should be used. For example ARMA/ARIMA, Kalman filtering of the time-trend trajectory.

Appendix

5.1 List of assumptions / limitations regarding the dataset

As mentioned during this report, we had to make numerous assumptions given the lack of clarity of the variable descriptions.

- *Dataset quality*: Aside from cents rounding issues, the dataset does not contain any flagrant errors that we could see (e.g. minor error of amount or rate, zipcode). Quality of the variable description is a different matter altogether.
- *Ratings*: The day-1 rating is between A1 and (and no lower than) G5. No note is rated lower than E5 after 6 November 2017, and lower than D5 after 30 June 2019.
- *Credit history*: Credit history information for the principal borrower relates to pre-approval and not post-funding. This is clear for the joint applicants, but simply an assumption for the principal borrower.
- *Recoveries*: Recoveries (if any) are assumed to be paid 3 months after the last scheduled payment date (variable `last_pymnt_d`)
- *Survival effect*: The dataset does not include applications that were rejected by the lender (for whatever reason) or by the borrower (for example because the interest rate quote is too high). It may also be the case that some actual loans were excluded as and when the dataset changed over the years.
- LIBOR funding rate: we use the 3-year and 5-year swap rates. In reality, we should have used average tenor-weighted swap rates (i.e. ca. 1.5 Y and 2.5 Y). This requires a full swap curve and more calculation than necessary for our purpose. The principles of this report should not be significantly affted by this approximation.

We do hope that LendingClub investors receive information of much better quality!

5.2 Data preparation and formatting

We used different sources of information:

- The LendingClub dataset made available on Kaggle;
- US geographical data about zip and FIPS codes;
- Market interest rates from the Saint Louis Federal Reserve Bank; and,
- Macro data from the same source.

We here show the code used to prepare the data. It was automatically formatted by *RStudio*.

5.2.1 LendinClub dataset

```
local({
  #
  # STEP 1: Download the dataset
  #
  #   Got to https://www.kaggle.com/wendykan/lending-club-loan-data
  #
  #   Download into the 'datasets' subdirectory
  #   Unzip the file.
  #   WARNING: The unzipping will be about 2.4GB
  #
  #   Name the sql database "datasets/lending_club.sqlite"
  #

  #
  # STEP 2: Prepare the dabase as a tibble
  #

  ##
  ## WARNING: THIS ASSUMES A 'datasets' DIRECTORY WAS CREATED
  ##
  library(RSQLite)
  db_conn <-
    dbConnect(RSQLite::SQLite(), "datasets/lending_club.sqlite")
  dbListTables(db_conn)

  # Returns a 2.96GB data frame
  lending_club <- dbGetQuery(db_conn, "SELECT * FROM loan")
  lending_club <- as_tibble(lending_club)

  # Close the database
  dbDisconnect(db_conn)

  # Compressed to ca.285MB on disk
  saveRDS(lending_club, "datasets/lending_club.rds")


  library(tidyverse)
  library(lubridate)
  library(hablar)

  # Before reformat in case the previous step was already done
  # lending_club <- readRDS("datasets/lending_club.rds")
  #
  # str(lending_club)
  #

  # Leave the original dataset untouched and work with a copy.
  lc <- lending_club

  lc <- lc %>%

    # Add loan identification number to track the loans across calculations
    mutate(loanID = row_number()) %>%

    # Remove useless strings
    mutate(
```

```
    term       = str_remove(term, " months"),
    emp_length = str_replace(emp_length, "<1", "0"),
    emp_length = str_replace(emp_length, "10+", "10"),
    emp_length = str_remove(emp_length, "years")
  ) %>%

  # Creates dates out of strings - Parse errors will be raised when no dates.
  mutate(
    debt_settlement_flag_date = as_date(dmy(
      str_c("1-", debt_settlement_flag_date)
    )),
    earliest_cr_line          = as_date(dmy(str_c(
      "1-", earliest_cr_line
    ))),
    hardship_start_date       = as_date(dmy(str_c(
      "1-", hardship_start_date
    ))),
    hardship_end_date         = as_date(dmy(str_c(
      "1-", hardship_end_date
    ))),
    issue_d                   = as_date(dmy(str_c("1-", issue_d))),
    last_credit_pull_d        = as_date(dmy(str_c(
      "1-", last_credit_pull_d
    ))),
    last_pymnt_d              = as_date(dmy(str_c(
      "1-", last_pymnt_d
    ))),
    next_pymnt_d              = as_date(dmy(str_c(
      "1-", next_pymnt_d
    ))),
    payment_plan_start_date   = as_date(dmy(str_c(
      "1-", payment_plan_start_date
    ))),
    sec_app_earliest_cr_line  = as_date(dmy(str_c(
      "1-", sec_app_earliest_cr_line
    ))),
    settlement_date           = as_date(dmy(str_c(
      "1-", settlement_date
    )))
  ) %>%

  # Bulk type conversion with convert from the `hablar` package
  convert(
    # Strings
    chr(emp_title, title, url, zip_code),

    # Factors
    fct(
      addr_state,
      application_type,
      debt_settlement_flag,
      desc,
      disbursement_method,
      grade,
      hardship_flag,
      hardship_loan_status,
      hardship_reason,
      hardship_status,
```

```
        hardship_type,
        home_ownership,
        id,
        initial_list_status,
        loan_status,
        member_id,
        policy_code,
        purpose,
        pymnt_plan,
        settlement_status,
        sub_grade,
        verification_status,
        verification_status_joint
      ),

      # Integers
      int(
        acc_now_delinq,
        acc_open_past_24mths,
        chargeoff_within_12_mths,
        collections_12_mths_ex_med,
        deferral_term,
        delinq_2yrs,
        emp_length,
        hardship_dpd,
        hardship_length,
        inq_fi,
        inq_last_12m,
        inq_last_6mths,
        mo_sin_old_il_acct,
        mo_sin_old_rev_tl_op,
        mo_sin_rcnt_rev_tl_op,
        mo_sin_rcnt_tl,
        mort_acc,
        mths_since_last_delinq,
        mths_since_last_major_derog,
        mths_since_last_record,
        mths_since_rcnt_il,
        mths_since_recent_bc,
        mths_since_recent_bc_dlq,
        mths_since_recent_inq,
        mths_since_recent_revol_delinq,
        num_accts_ever_120_pd,
        num_actv_bc_tl,
        num_actv_rev_tl,
        num_bc_sats,
        num_bc_tl,
        num_il_tl,
        num_op_rev_tl,
        num_rev_accts,
        num_rev_tl_bal_gt_0,
        num_sats,
        num_tl_120dpd_2m,
        num_tl_30dpd,
        num_tl_90g_dpd_24m,
        num_tl_op_past_12m,
        open_acc,
        open_acc_6m,
```

```
        open_act_il,
        open_il_12m,
        open_il_24m,
        open_rv_12m,
        open_rv_24m,
        sec_app_chargeoff_within_12_mths,
        sec_app_collections_12_mths_ex_med,
        sec_app_inq_last_6mths,
        sec_app_mort_acc,
        sec_app_mths_since_last_major_derog,
        sec_app_num_rev_accts,
        sec_app_open_acc,
        sec_app_open_act_il,
        term
      ),

      # Floating point
      dbl(
        all_util,
        annual_inc,
        annual_inc_joint,
        avg_cur_bal,
        bc_open_to_buy,
        bc_util,
        collection_recovery_fee,
        delinq_amnt,
        dti,
        dti_joint,
        funded_amnt,
        funded_amnt_inv,
        hardship_amount,
        hardship_last_payment_amount,
        hardship_payoff_balance_amount,
        il_util,
        installment,
        int_rate,
        last_pymnt_amnt,
        loan_amnt,
        max_bal_bc,
        orig_projected_additional_accrued_interest,
        out_prncp,
        out_prncp_inv,
        pct_tl_nvr_dlq,
        percent_bc_gt_75,
        pub_rec,
        pub_rec_bankruptcies,
        recoveries,
        revol_bal,
        revol_bal_joint,
        revol_util,
        sec_app_revol_util,
        settlement_amount,
        settlement_percentage,
        tax_liens,
        tot_coll_amt,
        tot_cur_bal,
        tot_hi_cred_lim,
        total_acc,
```

```
        total_bal_ex_mort,
        total_bal_il,
        total_bc_limit,
        total_cu_tl,
        total_il_high_credit_limit,
        total_pymnt,
        total_pymnt_inv,
        total_rec_int,
        total_rec_late_fee,
        total_rec_prncp,
        total_rev_hi_lim
      )
    ) %>%

    # Converts some values to 1/-1 (instead of Boolean)
    mutate(
      pymnt_plan =           if_else(pymnt_plan == "y",           1, -1),
      hardship_flag =        if_else(hardship_flag == "Y",        1, -1),
      debt_settlement_flag = if_else(debt_settlement_flag == "Y", 1, -1)
    ) %>%

    # Some values are percentages
    mutate(
      int_rate = int_rate / 100,
      dti = dti / 100,
      dti_joint = dti_joint / 100,
      revol_util = revol_util / 100,
      il_util = il_util / 100,
      all_util = all_util / 100,
      bc_open_to_buy = bc_util / 100,
      pct_tl_nvr_dlq = pct_tl_nvr_dlq / 100,
      percent_bc_gt_75 = percent_bc_gt_75 / 100,
      sec_app_revol_util = sec_app_revol_util / 100
    ) %>%

    # Create quasi-centered numerical grades out of grade factors with "A" = 3 down to "G" = -3
    mutate(grade_num = 4 - as.integer(grade)) %>%

    # Ditto with sub_grades. "A1" = +3.4, "A3" = +3.0, down to "G3" = -3.0, "G5" = -3.4
    mutate(sub_grade_num = 3.6 - as.integer(sub_grade) / 5) %>%

    # Keep the first 3 digits of the zipcode as numbers
    mutate(zip_code = as.integer(str_sub(zip_code, 1, 3))) %>%

    # order by date
    arrange(issue_d) %>%

    # Remove empty columns
    select(-id, -member_id, -url)

  saveRDS(lc, "datasets/lending_club_reformatted.rds")


  # Select loans which have matured or been terminated
  past_loans <- lc %>%
    filter(
      loan_status %in% c(
        "Charged Off",
```

```
        "Does not meet the credit policy. Status:Charged Off",
        "Does not meet the credit policy. Status:Fully Paid",
        "Fully Paid"
      )
    )

  saveRDS(past_loans, "datasets/lending_club_reformatted_paid.rds")
})```
```

5.2.2 Zip codes and FIPS codes

The R package `zipcode` was installed.

```
#
# ZIPCodes dataset.
#

library(zipcode)
data(zipcode)
zips <- zipcode %>%
  as_tibble() %>%
  mutate(zip = as.integer(str_sub(zip, 1, 3)))

saveRDS(zips, "datasets/zips.rds")
```

A csv file containing zip codes, FIPS codes and population information was downloaded from the *Simple Maps* [1] website.

```
local({
  kaggleCodes <- read.csv("datasets/csv/ZIP-COUNTY-FIPS_2017-06.csv")

  kaggleCodes <-
    kaggleCodes %>%
    as_tibble() %>%
    mutate(zip = floor(ZIP/100),
           FIPS = STCOUNTYFP,
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "County", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Borough", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Municipio", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Parish", replacement = ""),
           COUNTYNAME = str_replace(COUNTYNAME, pattern = "Census Area", replacement = "")) %>%
    rename(county = COUNTYNAME) %>%
    select(zip, county, FIPS) %>%
    arrange(zip)

  saveRDS(zipfips, "datasets/kaggleCodes.rds")
})
```

5.2.3 Market interest rates

Market interest rates (3-year and 5-year swap rates) were download from the Saint Louis Federal Reserve Bank. Datasets are split between before and after the LIBOR fixing scandal. The datasets are merged with disctinct dates.

Download sources are:

- Pre-LIBOR 3-y swap https://fred.stlouisfed.org/series/DSWP3

[1] https://simplemaps.com/data/us-zips

- Post-LIBOR 3-y swap https://fred.stlouisfed.org/series/ICERATES1100USD3Y
- Pre-LIBOR 5-y swap https://fred.stlouisfed.org/series/MSWP5
- Post-LIBOR 5-y swap https://fred.stlouisfed.org/series/ICERATES1100USD5Y

```
local({
  LIBOR3Y <- read.csv("datasets/csv/DSWP3.csv") %>%
    as_tibble() %>%
    filter(DSWP3 != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE3Y = as.numeric(as.character(DSWP3)) / 100) %>%
    select(DATE, RATE3Y)

  ICE3Y <- read.csv("datasets/csv/ICERATES1100USD3Y.csv") %>%
    as_tibble() %>%
    filter(ICERATES1100USD3Y != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE3Y = as.numeric(as.character(ICERATES1100USD3Y)) / 100) %>%
    select(DATE, RATE3Y)


  LIBOR5Y <- read.csv("datasets/csv/DSWP5.csv") %>%
    as_tibble() %>%
    filter(DSWP5 != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE5Y = as.numeric(as.character(DSWP5)) / 100) %>%
    select(DATE, RATE5Y)

  ICE5Y <- read.csv("datasets/csv/ICERATES1100USD5Y.csv") %>%
    as_tibble() %>%
    filter(ICERATES1100USD5Y != ".") %>%
    mutate(DATE = as_date(DATE),
           RATE5Y = as.numeric(as.character(ICERATES1100USD5Y)) / 100) %>%
    select(DATE, RATE5Y)

  RATES3Y <- LIBOR3Y %>% rbind(ICE3Y) %>%
    arrange(DATE) %>% distinct(DATE, .keep_all = TRUE)

  RATES5Y <- LIBOR5Y %>% rbind(ICE5Y) %>%
    arrange(DATE) %>% distinct(DATE, .keep_all = TRUE)

  saveRDS(RATES3Y, "datasets/rates3Y.rds")
  saveRDS(RATES5Y, "datasets/rates5Y.rds")


  # Note there are 7212 days from 1 Jan 2000 to 30 Sep 2019
  #
  # (ymd("2000-01-01") %--% ymd("2019-09-30")) %/% days(1)
  RATES <- tibble(n = seq(0, 7212)) %>%

    # Create a column with all dates
    mutate(DATE = ymd("2000-01-01") + days(n)) %>%
    select(-n) %>%

    # Add all daily 3- then 5-year rates and fill missing down
    left_join(RATES3Y) %>%
    fill(RATE3Y, .direction = "down") %>%

```

```
    left_join(RATES5Y) %>%
    fill(RATE5Y, .direction = "down")

  saveRDS(RATES, "datasets/rates.rds")
})
```

5.2.4 Macro-economical data

Macro-economical datasets were sourced from the same website as Microsoft Excel files. They were converted as-is to tab-separated csv files with LibreOffice.

- Median income per household: https://geofred.stlouisfed.org/map/?th=pubugn&cc=5&rc=false&im=fractile&sb&lng=-112.41&lat=44.31&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Annual,%20Dollars&sti=2022&fq=Annual&rt=county&un=lin&dt=2017-01-01
- Per capita personal income: https://geofred.stlouisfed.org/map/?th=pubugn&cc=5&rc=false&im=fractile&sb&lng=-112.41&lat=44.31&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Annual,%20Dollars&sti=882&fq=Annual&rt=county&un=lin&dt=2017-01-01
- Unemployment: https://geofred.stlouisfed.org/map/?th=rdpu&cc=5&rc=false&im=fractile&sb&lng=-90&lat=40&zm=4&sl&sv&am=Average&at=Not%20Seasonally%20Adjusted,%20Monthly,%20Percent&sti=1224&fq=Monthly&rt=county&un=lin&dt=2019-08-01

```
local({
  ##############################################################################################################
  ##
  ## Median income per household by FIPS from 2002 to 2017
  ##
  # Prepare median income
  medianIncome <-
    # Load the dataset after dropping the first line
    read.csv(
      "datasets/csv/GeoFRED_Estimate_of_Median_Household_Income_by_County_Dollars.csv",
      sep = "\t",
      skip = 1,
      stringsAsFactors = FALSE
    ) %>%

    # Drops columnsn containing a unique identifier and the FIPS name
    select(-"Series.ID", -"Region.Name") %>%

    # Rename the relevant column to 'FIPS'
    rename(FIPS = "Region.Code") %>%

    # Order by FIPS
    arrange(FIPS) %>%

    # Convert to a 'long' table, i.e. one column for FIPS, one for date, one for income
    pivot_longer(cols = starts_with("X"),
                 names_to = "Date",
                 values_to = "medianIncome") %>%

    # Create actual dates
    mutate(Date = str_replace(Date, "[X]", ""),
```

```
          Date = ymd(str_c(Date, "-12-31")))

saveRDS(medianIncome, "datasets/medianincome.rds")



################################################################################################################
##
## Per capita income by FIPS from 2002 to 2017
##
personalIncome <-
  # Load the dataset after dropping the first line
  read.csv(
    "datasets/csv/GeoFRED_Per_Capita_Personal_Income_by_County_Dollars.csv",
    sep = "\t",
    skip = 1,
    stringsAsFactors = FALSE
  ) %>%

  # Drops columnsn containing a unique identifier and the FIPS name
  select(-"Series.ID", -"Region.Name") %>%

  # Rename the relevant column to 'FIPS'
  rename(FIPS = "Region.Code") %>%

  # Order by FIPS
  arrange(FIPS) %>%

  # Convert to a 'long' table, i.e. one column for FIPS, one for date, one for income
  pivot_longer(cols = starts_with("X"),
               names_to = "Date",
               values_to = "personalIncome") %>%

  # Create actual dates
  mutate(Date = str_replace(Date, "[X]", ""),
         Date = ymd(str_c(Date, "-12-31")))

saveRDS(personalIncome, "datasets/personalincome.rds")


################################################################################################################
##
## Unemplyment rate monthly by FIPS from January 2000 to August 2019
##
unemploymentRate <-
  # Load the dataset after dropping the first line
  read.csv(
    "datasets/csv/GeoFRED_Unemployment_Rate_by_County_Percent.csv",
    sep = "\t",
    skip = 1,
    stringsAsFactors = FALSE
  ) %>%

  # Drops columnsn containing a unique identifier and the FIPS name
  select(-"Series.ID", -"Region.Name") %>%

  # Rename the relevant column to 'FIPS'
  rename(FIPS = "Region.Code") %>%
```

```
    # Order by FIPS
    arrange(FIPS) %>%

    mutate_all(as.double) %>%

    # Convert to a 'long' table, i.e. one column for FIPS, one for date, one for income
    pivot_longer(cols = starts_with("X"),
                 names_to = "Date",
                 values_to = "unemploymentRate",
                 values_ptypes = c("unemploymentRate", numeric)) %>%

    # Converts the content of the Year column to an actual date
    mutate(
      Date = str_replace(Date, "[X]", ""),
      Date = str_replace(Date, "[.]", "-"),
      Date = ymd(str_c(Date, "-1"))
    )

  saveRDS(unemploymentRate, "datasets/unemployment.rds")
})
```

5.3 List of variables

This table presents the list of variables provided in the original dataset. The descriptions come from a spreadsheet attached with the dataset and, unfortunately, are not extremely precise and subject to interpretation. We added comments and/or particular interpretations in *CAPITAL LETTERS*.

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle

Variable Name	Description
loanID	NOTE THIS IS NOT AN ORIGINAL VARIABLE. IT WAS ADDED FOR THE PURPOSE OF TRACKING LOANS INDIVIDUALLY AS AND WHEN NEEDED.
loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
funded_amnt	The total amount committed to that loan at that point in time.
funded_amnt_inv	The total amount committed by investors for that loan at that point in time.
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
int_rate	Interest Rate on the loan
installment	The monthly payment owed by the borrower if the loan originates.
grade	LC assigned loan grade
sub_grade	LC assigned loan subgrade
emp_title	The job title supplied by the Borrower when applying for the loan.

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER, NONE
annual_inc	The self-reported annual income provided by the borrower during registration. NOT USED AS A VARIABLE SINCE JOINT INCOME ALREADY INCLUDES IT.
verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
issue_d	The month which the loan was funded
loan_status	Current status of the loan
pymnt_plan	Indicates if a payment plan has been put in place for the loan
url	URL for the LC page with listing data.
desc	Loan description provided by the borrower
purpose	A category provided by the borrower for the loan request.
title	The loan title provided by the borrower
zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.
addr_state	The state provided by the borrower in the loan application
dti	A ratio calculated using the borrower s total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower s self-reported monthly income. NOT USED AS A VARIABLE. ONLY USE JOINT DTI.
delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower s credit file for the past 2 years
earliest_cr_line	The month the borrower s earliest reported credit line was opened
inq_last_6mths	The number of inquiries in past 6 months (excluding auto and mortgage inquiries)
mths_since_last_delinq	The number of months since the borrower s last delinquency.
mths_since_last_record	The number of months since the last public record.
open_acc	The number of open credit lines in the borrower s credit file.
pub_rec	Number of derogatory public records
revol_bal	Total credit revolving balance
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
total_acc	The total number of credit lines currently in the borrower s credit file
initial_list_status	The initial listing status of the loan. Possible values are – W, F
out_prncp	Remaining outstanding principal for total amount funded. NOTE ONCE A LOAN IS REPAID OR CHARGED OFF, THIS AMOUNT BECOMES 0.
out_prncp_inv	Remaining outstanding principal for portion of total amount funded by investors. NOTE ONCE A LOAN IS REPAID OR CHARGED OFF, THIS AMOUNT BECOMES 0.
total_pymnt	Payments received to date for total amount funded
total_pymnt_inv	Payments received to date for portion of total amount funded by investors
total_rec_prncp	Principal received to date. NOTE THIS AMOUNT WILL SHOW WHETHER A BORROWER DID NOT REPAY IN FULL
total_rec_int	Interest received to date
total_rec_late_fee	Late fees received to date
recoveries	Post charge off gross recovery. NOTE IF A LOAN IS REPAID, THIS AMOUNT IS 0.
collection_recovery_fee	Post charge off collection fee
last_pymnt_d	Last month payment was received
last_pymnt_amnt	Last total payment amount received
next_pymnt_d	Next scheduled payment date
last_credit_pull_d	The most recent month LC pulled credit for this loan
collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections
mths_since_last_major_derog	Months since most recent 90-day or worse rating
policy_code	Publicly available policy_code=1 / New products not publicly available policy_code=2
application_type	Indicates whether the loan is an individual application or a joint application with two coborrowers
annual_inc_joint	The combined self-reported annual income provided by the coborrowers during registration
dti_joint	A ratio calculated using the coborrowers total monthly payments on the total debt obligations, excluding mortgages and the requested LC loan, divided by the coborrowers combined self-reported monthly income
verification_status_joint	Indicates if income was verified by LC, not verified, or if the income source was verified
acc_now_delinq	The number of accounts on which the borrower is now delinquent.
tot_coll_amt	Total collection amounts ever owed

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
tot_cur_bal	Total current balance of all accounts
open_acc_6m	Number of open trades in last 6 months
open_act_il	Number of currently active installment trades
open_il_12m	Number of installment accounts opened in past 12 months
open_il_24m	Number of installment accounts opened in past 24 months
mths_since_rcnt_il	Months since most recent instalment accounts opened
total_bal_il	Total current balance of all installment accounts
il_util	Ratio of total current balance to high credit/credit limit on all install acct
open_rv_12m	Number of revolving trades opened in past 12 months
open_rv_24m	Number of revolving trades opened in past 24 months
max_bal_bc	Maximum current balance owed on all revolving accounts
all_util	Balance to credit limit on all trades
total_rev_hi_lim	Total revolving high credit/credit limit
inq_fi	Number of personal finance inquiries
total_cu_tl	Number of finance trades
inq_last_12m	Number of credit inquiries in past 12 months
acc_open_past_24mths	Number of trades opened in past 24 months.
avg_cur_bal	Average current balance of all accounts
bc_open_to_buy	Total open to buy on revolving bankcards.
bc_util	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
chargeoff_within_12_mths	Number of charge-offs within 12 months
delinq_amnt	The past-due amount owed for the accounts on which the borrower is now delinquent.
mo_sin_old_il_acct	Months since oldest bank instalment account opened
mo_sin_old_rev_tl_op	Months since oldest revolving account opened
mo_sin_rcnt_rev_tl_op	Months since most recent revolving account opened
mo_sin_rcnt_tl	Months since most recent account opened
mort_acc	Number of mortgage accounts.
mths_since_recent_bc	Months since most recent bankcard account opened.
mths_since_recent_bc_dlq	Months since most recent bankcard delinquency
mths_since_recent_inq	Months since most recent inquiry.
mths_since_recent_revol_delinq	Months since most recent revolving delinquency.
num_accts_ever_120_pd	Number of accounts ever 120 or more days past due
num_actv_bc_tl	Number of currently active bankcard accounts
num_actv_rev_tl	Number of currently active revolving trades

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
num_bc_sats	Number of satisfactory bankcard accounts
num_bc_tl	Number of bankcard accounts
num_il_tl	Number of installment accounts
num_op_rev_tl	Number of open revolving accounts
num_rev_accts	Number of revolving accounts
num_rev_tl_bal_gt_0	Number of revolving trades with balance >0
num_sats	Number of satisfactory accounts
num_tl_120dpd_2m	Number of accounts currently 120 days past due (updated in past 2 months)
num_tl_30dpd	Number of accounts currently 30 days past due (updated in past 2 months)
num_tl_90g_dpd_24m	Number of accounts 90 or more days past due in last 24 months
num_tl_op_past_12m	Number of accounts opened in past 12 months
pct_tl_nvr_dlq	Percent of trades never delinquent
percent_bc_gt_75	Percentage of all bankcard accounts > 75% of limit.
pub_rec_bankruptcies	Number of public record bankruptcies
tax_liens	Number of tax liens
tot_hi_cred_lim	Total high credit/credit limit
total_bal_ex_mort	Total credit balance excluding mortgage
total_bc_limit	Total bankcard high credit/credit limit
total_il_high_credit_limit	Total installment high credit/credit limit
revol_bal_joint	Total credit revolving balance
sec_app_earliest_cr_line	Earliest credit line at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_inq_last_6mths	Credit inquiries in the last 6 months at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_mort_acc	Number of mortgage accounts at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_open_acc	Number of open trades at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_revol_util	Ratio of total current balance to high credit/credit limit for all revolving accounts. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
sec_app_open_act_il	Number of currently active installment trades at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_num_rev_accts	Number of revolving accounts at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_chargeoff_within_12_mths	Number of charge-offs within last 12 months at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_collections_12_mths_ex_med	Number of collections within last 12 months excluding medical collections at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
sec_app_mths_since_last_major_derog	Months since most recent 90-day or worse rating at time of application for the secondary applicant. VARIABLE NOT USED. WE RELY ON THE MAIN BORROWER IN THE FIRST INSTANCE.
hardship_flag	Flags whether or not the borrower is on a hardship plan
hardship_type	Describes the hardship plan offering
hardship_reason	Describes the reason the hardship plan was offered
hardship_status	Describes if the hardship plan is active, pending, cancelled, completed, or broken
deferral_term	Amount of months that the borrower is expected to pay less than the contractual monthly payment amount due to a hardship plan
hardship_amount	The interest payment that the borrower has committed to make each month while they are on a hardship plan
hardship_start_date	The start date of the hardship plan period
hardship_end_date	The end date of the hardship plan period
payment_plan_start_date	The day the first hardship plan payment is due. For example, if a borrower has a hardship plan period of 3 months, the start date is the start of the three-month period in which the borrower is allowed to make interest-only payments.
hardship_length	The number of months the borrower will make smaller payments than normally obligated due to a hardship plan

Table 5.1: Description of the dataset variables as provided in the dataset downloaded from Kaggle *(continued)*

Variable Name	Description
hardship_dpd	Account days past due as of the hardship plan start date
hardship_loan_status	Loan Status as of the hardship plan start date
orig_projected_additional_accrued_interest	The original projected additional interest amount that will accrue for the given hardship payment plan as of the Hardship Start Date. This field will be null if the borrower has broken their hardship payment plan.
hardship_payoff_balance_amount	The payoff balance amount as of the hardship plan start date
hardship_last_payment_amount	The last payment amount as of the hardship plan start date
disbursement_method	The method by which the borrower receives their loan. Possible values are: CASH, DIRECT_PAY
debt_settlement_flag	Flags whether or not the borrower, who has charged-off, is working with a debt-settlement company.
debt_settlement_flag_date	The most recent date that the Debt_Settlement_Flag has been set
settlement_status	The status of the borrower's settlement plan. Possible values are: COMPLETE, ACTIVE, BROKEN, CANCELLED, DENIED, DRAFT
settlement_date	The date that the borrower agrees to the settlement plan
settlement_amount	The loan amount that the borrower has agreed to settle for
settlement_percentage	The settlement amount as a percentage of the payoff balance amount on the loan
settlement_term	The number of months that the borrower will be on the settlement plan

5.4 Calculations of the internal rate of returns and month-to-default

The following shows two versions of the same code. The R version is provided because of the description of the assignment. However, the R version takes just under a full day to run. A Julia version, which is a direct translation of the R code, runs in about 150s (ca. 500x faster).

5.4.1 R code

```
#############################################################################################################
#
# Given some numerical parameters describing a loan in the dataset, returns its Internal Rate
# of Return.
#
# In the first instance, the function creates a schedule of payments.
# In many cases, the schedule will be extremely simple: a series of 36 or 60 equal instalements.
#
# But in some cases, a loan repayment are accelerated. Therefore the total amount of interest will
```

```r
# be lower than expected (but this is good for the investor because highe interest rate over
# shorter tenor.).
#
# In other cases, the borrower defaults. Overall payments are less than expected.
#
# Based on the limited information of the dataset, the function makes educated guesses on the exact
#
# WARNING: THIS IS NOT OPTIMISED. RUNNING THIS FOR ALL LOANS (1.3 MLN OF THEM) TAKES CA.20 HOURS !!!
#
calculateIRR <- function(loanNumber = 1,
                         loan,
                         intRate,
                         term = 36,
                         totalPaid,
                         totalPrincipalPaid,
                         totalInterestPaid,
                         recoveries = 0,
                         lateFees = 0,
                         showSchedule = FALSE) {
  require(tidyverse)

  # number of monthly payments.
  # It exceeds 60 months in case recoveries on a 60-month loan takes the schedule after 60 months.
  nMonths <- 90

  # Months after which a loan defaults (normal tenor if no default or early prepayment)
  monthDefault = term

  # Note: *100 /100 to calculate in cent because ceiling cannot specify significant digits.
  installment <-
    ceiling(100 * loan * intRate / 12 / (1 - 1 / (1 + intRate / 12) ^ term)) / 100

  # We create a schedule
  schedule <- tibble(
    month = 0:nMonths,
    monthlyPayment = 0.0,
    totalPandI = 0.0,
    totalI = 0.0,
    totalP = 0.0
  )

  for (i in 2:(nMonths + 2)) {
    # Get situation at the end of previous month
    previousTotalPandI <- as.numeric(schedule[i - 1, "totalPandI"])
    previousTotalP     <- as.numeric(schedule[i - 1, "totalP"])
    previousTotalI     <- as.numeric(schedule[i - 1, "totalI"])

    # This is the beginning of a new month. First and foremost, the borrower is expected to pay the
    # accrued interest on amount of principal outstanding.
    # ceiling doesn't seem accept to accept significative digits.
    accruedInterest <-
      ceiling(100 * (loan - previousTotalP) * intRate / 12) / 100

    # If that amount takes the schedule above the total amount of interest shown in the data set,
    # we should stop the schedule at this point
    if (previousTotalI + accruedInterest > totalInterestPaid) {
      # We stop the normal schedule at this date.
      # Interest is paid (although less than scheduled)
```

```r
      schedule[i, "monthlyPayment"] <-
        totalInterestPaid - previousTotalI

      # As well as whatever principal is left as per the dataset
      schedule[i, "monthlyPayment"] <-
        schedule[i, "monthlyPayment"] + totalPrincipalPaid - previousTotalP

      # Then 3-month after the last payment date, recoveries and and later fees are paid
      schedule[i + 3, "monthlyPayment"] <-
        schedule[i + 3, "monthlyPayment"] + recoveries + lateFees

      # Not really useful, but for completeness
      schedule[i, "totalPandI"] <- totalPaid
      schedule[i, "totalI"]     <- totalInterestPaid
      schedule[i, "totalP"]     <- totalPrincipalPaid

      # If total principal paid is less than borrower, then it is a default, and the monthDefault
      # is adjusted.
      if (totalPrincipalPaid < loan) {
        monthDefault = i
      }

      # No more payments to add to the schedule
      break()

    } else {
      # Deal with normal schedule
      schedule[i, "monthlyPayment"] <- installment
      schedule[i, "totalPandI"] <-
        schedule[i - 1, "totalPandI"] + installment
      schedule[i, "totalI"]     <-
        schedule[i - 1, "totalI"]   + accruedInterest
      schedule[i, "totalP"]     <-
        schedule[i - 1, "totalP"]   + installment - accruedInterest
    }
  }

  # At this point schedule[, "monthlyPayment"] contains the schedule of all payments, but needs to
  # include the initial loan.
  schedule[1, "monthlyPayment"] <- -loan

  if (showSchedule) {
    schedule %>% view()
  }

  NPV <- function(interest, cashFlow) {
    t = 0:(length(cashFlow) - 1)
    sum(cashFlow / (1 + interest) ^ t)
  }

  IRR <- function(CF) {
    res <- NA
    try({
      res <- uniroot(NPV, c(-0.9, 1), cashFlow = CF)$root
    },
    silent = TRUE)
    return(res)
  }
```

```
126
127   return(tibble(
128     loanID = loanNumber,
129     IRR = round(as.numeric(IRR(
130       schedule$monthlyPayment
131     ) * 12), digits = 4),
132     monthDefault = monthDefault
133   ))
134 }
135
136 loanNumberIRR <- function(loanNumber) {
137   require(tidyverse)
138
139   l <- loans %>% filter(loanID == loanNumber)
140   calculateIRR(
141     loanNumber = l$loanID,
142     loan = l$funded_amnt, intRate = l$int_rate, term = l$term,
143     totalPaid = l$total_pymnt, totalPrincipalPaid = l$total_rec_prncp, totalInterestPaid = l$total_r
144     recoveries = l$recoveries, lateFees = l$total_rec_late_fee,
145     showSchedule = TRUE
146   )
147 }
```

```
1  #
2  # Calculate all the IRRs and month of default for all the loans.
3  # WARNING: This takes around a full day to run!!!!
4  #
5  # The actual data was generated by the Julia version, with cross-checks.
6  # Julia version takes about 150 sec on the same unoptimised code.
7  #
8  local({
9    loansIRR <-
10     loans %>%
11     rowwise() %>%
12     do(calculateIRR(loanNumber = .$loanID,
13                     loan = .$funded_amnt, intRate = .$int_rate, term = .$term,
14                     totalPaid = .$total_pymnt, totalPrincipalPaid = .$total_rec_prncp,
15                     totalInterestPaid = .$total_rec_int,
16                     recoveries = .$recoveries, lateFees = .$total_rec_late_fee))
17
18   saveRDS(loansIRR, "datasets/lending_club_IRRs.rds")
19
20   })
```

5.4.2 Julia code

5.4.2.1 Internal Rate of Return

```
1  ##############################################################################################################
2  ##
3  ## Prepare datasets
4  ##
5
6  ## Previously saved from R with:
7  ##     lending_club <- readRDS("lending_club.rds"); write.csv(lending_club, "lending_club.rds")
8  ##
9  ## WARNING: 1.7GB on disk
10 ##
```

```
using CSV
lendingClub = CSV.read("datasets/lending_club.csv"; delim = ",")



##############################################################################################################
##
## IRR calculations
##
## Given some numerical parameters describing a loan in the dataset, returns its Internal Rate
## of Return.
##
## In the first instance, the function creates a schedule of payments.
## In many cases, the schedule will be extremely simple: a series of 36 or 60 equal instalements.
##
## But in some cases, a loan repayment are accelerated. Therefore the total amount of interest will
## be lower than expected (but this is good for the investor because highe interest rate over
## shorter tenor.).
##
## In other cases, the borrower defaults. Overall payments are less than expected.
##
## Based on the limited information of the dataset, the function makes educated guesses on the exact
## schedule.
##
using DataFrames, Roots

function calculateIRR(; loanNumber = 1, loan = 0.0, intRate = 0.0, term = 36,
  totalPaid = 0.0, totalPrincipalPaid = 0.0, totalInterestPaid = 0.0,
  recoveries = 0.0, lateFees = 0.0,
  showSchedule = false)

  # number of monthly payments.
  # It exceeds 60 months in case recoveries on a 60-month loan takes the schedule after 60 months.
  nMonths = 90

  # Months after which a loan defaults (normal tenor if no default or early prepayment)
  monthDefault = term

  # Note: *100 /100 to calculate in cent because ceiling cannot specify significant digits.
  installment = ceil(loan * intRate / 12 / (1 - 1 / (1 + intRate / 12) ^ term), digits = 2)

  # We create a schedule
  schedule = DataFrame(month = 0:nMonths, monthlyPayment = 0.0,
                       totalPandI = 0.0, totalI = 0.0, totalP = 0.0)

  for i in 2:(nMonths + 1)
    # Get situation at the end of previous month
    previousTotalPandI = schedule[i - 1, :totalPandI]
    previousTotalP     = schedule[i - 1, :totalP]
    previousTotalI     = schedule[i - 1, :totalI]

    # This is the beginning of a new month. First and foremost, the borrower is expected to pay the
    # accrued interest on amount of principal outstanding.
    # The installment is expected to cover that amount of interest and the rest goes to
    # reducing the principal due outstanding.
    accruedInterest = ceil((loan - previousTotalP) * intRate / 12; digits = 2)
    decreasePrincipal = installment - accruedInterest

```

```
    # If that amount takes the schedule above the total amount of interest shown in the data set,
    # we should stop the schedule at this point
    # This is a shortcut since we could have a payment higher than the interest due, but not enough
    # to cover the expected principal repayment. However, it works well in practice.
    if previousTotalI + accruedInterest > totalInterestPaid

      # We stop the normal schedule at this date.
      # Interest is paid (although less than scheduled)
      schedule[i, :monthlyPayment] = totalInterestPaid - previousTotalI

      # As well as whatever principal is left as per the dataset
      schedule[i, :monthlyPayment] = schedule[i, :monthlyPayment] + totalPrincipalPaid - previousTota

      # Then 3-month after the last payment date, recoveries and and later fees are paid
      schedule[i + 3, :monthlyPayment] = schedule[i + 3, :monthlyPayment] + recoveries + lateFees

      # Not really useful, but for completeness
      schedule[i, :totalPandI] = totalPaid
      schedule[i, :totalI]     = totalInterestPaid
      schedule[i, :totalP]     = totalPrincipalPaid

      # If total principal paid is less than borrower, then it is a default, and the monthDefault
      # is adjusted.
      if (totalPrincipalPaid < loan)
        monthDefault = i
      end

      # No more payments to add to the schedule
      break

    else
      # Deal with normal schedule
      schedule[i, :monthlyPayment] = installment
      schedule[i, :totalPandI]     = schedule[i - 1, :totalPandI] + installment
      schedule[i, :totalI]         = schedule[i - 1, :totalI]     + accruedInterest
      schedule[i, :totalP]         = schedule[i - 1, :totalP]     + installment - accruedInterest
    end
  end

  # At this point schedule[, :monthlyPayment] contains the schedule of all payments, but needs to
  # include the initial loan.
  schedule[1, :monthlyPayment] = -loan

  if (showSchedule)
    print(schedule)
  end

  cashFlow = schedule[:,:monthlyPayment]

  ##
  ## Finding the IRR is equivalent to finding the root such that the NPV of the cash flow is zero.
  ## Julia has a function (see below) called `find_zero` to do that which requires a function to
  ## be zeroed. This helper function is defined as NPV.
  ##
  function NPV(interest)
    t = 0:(length(cashFlow) - 1)

    ## If you are new to Julia, note the dot before the operation. This indicates that the
```

```
    ## operation has to be done element-wise (called `broadcasting` in Julia-ese).
    ## Otherwise, Julia would try to divide one vector by another vector, which makes no sense.
    ## This is also exactly the approach taken in Matlab/Octave.
    return sum(cashFlow ./ (1 + interest) .^ t)
  end

  ## Finds the root, catching any problems which would instead return the R equivalent of NA
  rootInterest = try
                   round(12 * find_zero(NPV, (-0.9, 1.0), Bisection(); xatol = 0.000001); digits = 4)
                 catch e
                   NaN
                 end

  return((
    loanID = loanNumber,
    IRR = rootInterest,
    monthDefault = monthDefault
  ))
end


##
## Calculate the IRR and repayment schedule of a particular loan identified by its loanID
function loanNumberIRR(loanNumber)
  l = lc[ lc[:, :Column1] .== loanNumber, :]
  global lc
  calculateIRR(loanNumber = l[1, :Column1],
               loan = l[1, :funded_amnt], intRate = l[1, :int_rate], term =l[1, :tenor],
               totalPaid = l[1, :total_pymnt], totalPrincipalPaid = l[1, :total_rec_prncp],
               totalInterestPaid = l[1, :total_rec_int],
               recoveries = l[1, :recoveries], lateFees = l[1, :total_rec_late_fee],
               showSchedule = true)
end


##############################################################################################
##
## Quick check
##
calculateIRR(loanNumber = 1, loan = 5600, intRate = 0.1299, term = 36,
             totalPaid = 6791.72, totalPrincipalPaid = 5600, totalInterestPaid = 1191.72,
             recoveries = 0, lateFees = 0,
             showSchedule = true)

calculateIRR(loanNumber = 1, loan = 35000, intRate = 0.1820, term = 60,
             totalPaid = 26600.1, totalPrincipalPaid = 3874.72, totalInterestPaid = 5225.38,
             recoveries = 17500, lateFees = 0.0,
             showSchedule = false)

calculateIRR(loanNumber = 1734666, loan = 35000, intRate = 0.0797, term = 36,
             totalPaid = 1057.04, totalPrincipalPaid = 863.83, totalInterestPaid = 193.72,
             recoveries = 0, lateFees = 0,
             showSchedule = false)




##
```

```
## Look for the loans which have gone to their end
##
indextmp = (lendingClub.loan_status .== "Fully Paid") .|
           (lendingClub.loan_status .== "Charged Off") .|
           (lendingClub.loan_status .== "Does not meet the credit policy. Status:Charged Off") .|
           (lendingClub.loan_status .== "Does not meet the credit policy. Status:Fully Paid")

## Create the dataset we will use - Should be the same as lending_club_reformatted_paid.rds
lc = lendingClub[indextmp, :]

## Select relevant variables to calculate profitability
## Column1 contains the loanID's
cols = [:Column1, :funded_amnt, :int_rate, :term,
        :total_pymnt, :total_rec_prncp, :total_rec_int,
        :recoveries, :total_rec_late_fee]

lc = select(lc, cols)

## Interest rates as percentage
lc[:, :int_rate] = lc[:, :int_rate] ./ 100

## Create a new column
lc[:tenor] = 0

## that will record the official loan tenor as a number (instead of string)
lc[startswith.( lc[:, :term], " 36"), :tenor] .= 36
lc[startswith.( lc[:, :term], " 60"), :tenor] .= 60

## New data frame to store the results
IRR_Result = DataFrame(loanID = zeros(Int64, nrow(lc)),
                       IRR = zeros(Float64, nrow(lc)),
                       monthDefault = zeros(Int64, nrow(lc)))


# ~150 sec. to do the whole dataset
@time for i in 1:nrow(lc)
  global IRR_Result

  # Use multiple-return-value
  (IRR_Result[i, :loanID], IRR_Result[i, :IRR], IRR_Result[i, :monthDefault]) =
      calculateIRR(
          loanNumber = lc[i, :Column1],
          loan = lc[i, :funded_amnt], intRate = lc[i, :int_rate], term =lc[i, :tenor],
          totalPaid = lc[i, :total_pymnt], totalPrincipalPaid = lc[i, :total_rec_prncp],
          totalInterestPaid = lc[i, :total_rec_int],
          recoveries = lc[i, :recoveries], lateFees = lc[i, :total_rec_late_fee],
          showSchedule = false)
end


IRR_Result[1:10,:]
# Check
loanNumberIRR(171)

CSV.write("datasets/loanIRR.csv", IRR_Result)
```

5.4.2.2 Credit margins

```
################################################################################################################
##
## Prepare datasets
##

## Previously saved from R with:
##     lending_club <- readRDS("lending_club.rds"); write.csv(lending_club, "lending_club.rds")
##
## WARNING: 1.7GB on disk
##
using CSV
lendingClub = CSV.read("datasets/lending_club.csv"; delim = ",")



################################################################################################################
##
## CREDIT MARGIN
##
## This method of approximating the credit margin is far less sophisticated than what FI's do.
##
## We need to calculate what the credit margin should be on a defaulted loan to get a nil NPV.
##
## On a risk free loan the CF will be P+I at risk-free on _both_ borrowing and lending sides.
##
##  On a defaulted loan, the CF will be:
##      borrowing unchanged = P&I at risk-free
##    and
##      lending P&I at (risk-free + credit margin) until before default, then recoveries+fees.
##
## The principal amortisation profile depends on the credit margin used. We will arbitrarily use
## 20% which is a conservative assumption.
##
## credit risk on that CF should be nil with the right margin when discounted at risk-free.
##
## The function is very similar to the IRR calculation.
##

using DataFrames, Roots

# number of monthly payments to model
# It exceeds 60 months in case recoveries on a 60-month loan takes the schedule after 60 months.
const nMonths = 90



# Sculpt credit foncier profiles over 36 and 60 months at 20% per annum.
# The profile is expressed as percentage of loan amount
function CreateCreditFoncier(;n = 36, riskFree = 0.0)

  instalment = 1 * riskFree/12 * 1 / (1 - 1 / (1 + riskFree/12) ^ n)

  # We create a schedule
  schedule = DataFrame(month = 0:nMonths, payment = 0.0)

  # Add the day 1 principal outlay
```

```
  schedule[1,       :payment] = -1
  schedule[2:(n+1), :payment] = instalment

  return(schedule)
end


# Solve for the credit margin
function CreditMargin(; loanNumber = 1, loan = 1000.0, intRate = 0.05, term = 36,
  totalPaid = 1000.0, totalPrincipalPaid = 700.0, totalInterestPaid = 50.0,
  recoveries = 0.0, lateFees = 0.0,
  riskFree = 0.01,
  showSchedule = false)

  # Months after which a loan defaults (normal tenor if no default or early prepayment)
  monthDefault = term

  # Monthly instlment
  instalment = ceil(loan * intRate/12 / (1 - 1 / (1 + intRate/12) ^ term), digits = 2)

  # Create a blank schedule
  schedule = DataFrame(month = 0:nMonths, monthlyPayment = 0.0,
                       principalPayment = 0.0,
                       totalPandI = 0.0, totalI = 0.0, totalP = 0.0)

  for i in 2:(nMonths + 1)
    # Get situation at the end of previous month
    previousTotalPandI = schedule[i - 1, :totalPandI]
    previousTotalP     = schedule[i - 1, :totalP]
    previousTotalI     = schedule[i - 1, :totalI]

    # This is the beginning of a new month. First and foremost, the borrower is expected to pay the
    # accrued interest on amount of principal outstanding.
    # The instalment is expected to cover that amount of interest and the rest goes to
    # reducing the principal due outstanding.
    accruedInterest = ceil((loan - previousTotalP) * intRate/12; digits = 2)
    decreasePrincipal = instalment - accruedInterest

    # If that amount takes the schedule above the total amount of interest shown in the data set,
    # we should stop the schedule at this point
    # This is a shortcut since we could have a payment higher than the interest due, but not enough
    # to cover the expected principal repayment. However, it works well in practice.
    if previousTotalI + accruedInterest > totalInterestPaid

      # We stop the normal schedule at this date.
      # Interest is paid (although less than scheduled)
      schedule[i, :monthlyPayment] = totalInterestPaid - previousTotalI

      # Whatever principal is left as per the dataset
      schedule[i, :monthlyPayment] += totalPrincipalPaid - previousTotalP

      # Then 3-month after the last payment date, recoveries and and later fees are paid
      schedule[i + 3, :principalPayment] += recoveries + lateFees

      # Not really useful, but for completeness
      schedule[i, :totalPandI] = totalPaid
      schedule[i, :totalI]     = totalInterestPaid
      schedule[i, :totalP]     = totalPrincipalPaid
```

```
115
116       # If total principal paid is less than borrower, then it is a default, and the monthDefault
117       # is adjusted.
118       if (totalPrincipalPaid < loan)
119         monthDefault = i
120       end
121
122       # No more payments to add to the schedule
123       break
124
125     else
126       # Deal with normal schedule
127       schedule[i, :monthlyPayment]   = instalment
128       schedule[i, :principalPayment] = decreasePrincipal
129       schedule[i, :totalPandI]       = schedule[i-1, :totalPandI] + instalment
130       schedule[i, :totalI]           = schedule[i-1, :totalI]     + accruedInterest
131       schedule[i, :totalP]           = schedule[i-1, :totalP]     + decreasePrincipal
132     end
133   end
134
135   # At this point schedule[, :monthlyPayment] contains the schedule of all payments, but needs to
136   # include the initial loan.
137   schedule[1, :principalPayment] = -loan
138
139   if (showSchedule)
140     println("Payments")
141     println(schedule)
142   end
143
144   # Principal profile on the borrowing side
145   creditFoncier = round.(CreateCreditFoncier(n = term, riskFree = riskFree)[:, :payment] .* loan;
146                                               digits = 2)
147
148   # For a given margin, calculate the _net_ NPV between the what is borrowed at the risk-free rate
149   # and what is earned on the loan principal profile (possibly shortned because of default)
150   # carrying an interest of risk-free + credit margin
151   function NetNPV(margin)
152     # We need to store the calculated interest
153     interestSchedule = DataFrame(month = 0:nMonths, interestPayment = 0.0)
154
155     # For each month, calculate the amount of interest with the credit margin
156     for i in 2:(monthDefault + 1)
157       outstandingPrincipal = sum(schedule[1:(i - 1), :principalPayment])
158       interestSchedule[i, :interestPayment] =
159             -round((riskFree + margin)/12 * outstandingPrincipal; digits = 2)
160     end
161
162     # Net final cashflow is:
163     #     total principal and interest cashflow on the lending side
164     # less
165     #     borrowing profile
166     cashFlow = schedule[:, :principalPayment] .+ interestSchedule[:, :interestPayment]
167     cashFlow = cashFlow .- creditFoncier
168
169     return sum(cashFlow ./ (1 + riskFree/12) .^ (0:nMonths))
170   end
171
172   # rootInterest = round(find_zero(NetNPV, (-0.5, 10), Bisection()); digits = 6)
```

```
  rootInterest = try
                   rootInterest = round(find_zero(NetNPV, (-0.5, 10), Bisection()); digits = 6)
                 catch e
                   NaN
                 end


  return((
    loanID = loanNumber,
    creditMargin = rootInterest,
    monthDefault = monthDefault
  ))
end


#############################################################################################################
##
## Quick check
##
CreditMargin(loanNumber = 1, loan = 5600, intRate = 0.1299, term = 36,
             totalPaid = 6791.72, totalPrincipalPaid = 5600, totalInterestPaid = 1191.72,
             recoveries = 0, lateFees = 0,
             riskFree = 0.02, showSchedule = false)

CreditMargin(loanNumber = 1, loan = 35000, intRate = 0.1820, term = 60,
             totalPaid = 26600.1, totalPrincipalPaid = 3874.72, totalInterestPaid = 5225.38,
             recoveries = 17500, lateFees = 0.0,
             riskFree = 0.02, showSchedule = true)

CreditMargin(loanNumber = 1734666, loan = 35000, intRate = 0.0797, term = 36,
             totalPaid = 1057.04, totalPrincipalPaid = 863.83, totalInterestPaid = 193.72,
             recoveries = 0, lateFees = 0,
             riskFree = 0.02, showSchedule = true)


##
## Look for the loans which have gone to their end
##
indextmp = (lendingClub.loan_status .== "Fully Paid") .|
           (lendingClub.loan_status .== "Charged Off") .|
           (lendingClub.loan_status .== "Does not meet the credit policy. Status:Charged Off") .|
           (lendingClub.loan_status .== "Does not meet the credit policy. Status:Fully Paid")

## Create the dataset we will use - Should be the same as lending_club_reformatted_paid.rds
lc = lendingClub[indextmp, :]

## Select relevant variables to calculate profitability
## Column1 contains the loanID's
cols = [:Column1, :funded_amnt, :int_rate, :term,
        :total_pymnt, :total_rec_prncp, :total_rec_int,
        :recoveries, :total_rec_late_fee]

lc = select(lc, cols)

## Interest rates as percentage
lc[!, :int_rate] = lc[!, :int_rate] ./ 100

## Create a new column
```

```
lc[:tenor] = 0

## that will record the official loan tenor as a number (instead of string)
lc[startswith.( lc[!, :term], " 36"), :tenor] .= 36
lc[startswith.( lc[!, :term], " 60"), :tenor] .= 60

## New data frame to store the results
creditMargin_Result = DataFrame(loanID = zeros(Int64, nrow(lc)),
                                creditMargin = zeros(Float64, nrow(lc)),
                                monthDefault = zeros(Int64, nrow(lc)))


# ~4,700 sec. to do the whole dataset
@time for i in 1:nrow(lc)
  global creditMargin_Result

  # Use multiple-return-value
  # 1h17m runtime
  (creditMargin_Result[i, :loanID],
   creditMargin_Result[i, :creditMargin],
   creditMargin_Result[i, :monthDefault]) =
      CreditMargin(
          loanNumber = lc[i, :Column1],
          loan = lc[i, :funded_amnt], intRate = lc[i, :int_rate], term =lc[i, :tenor],
          totalPaid = lc[i, :total_pymnt], totalPrincipalPaid = lc[i, :total_rec_prncp],
          totalInterestPaid = lc[i, :total_rec_int],
          recoveries = lc[i, :recoveries], lateFees = lc[i, :total_rec_late_fee],
          riskFree = 0.02,
          showSchedule = false)
end


creditMargin_Result[1:10,:]

CSV.write("datasets/CreditMargins.csv", creditMargin_Result)
```

5.5 System version

```
##                                       sysname
##                                       "Linux"
##                                       release
##                               "5.3.0-21-generic"
##                                       version
## "#22-Ubuntu SMP Tue Oct 29 22:55:51 UTC 2019"
##                                      nodename
##                                        "x260"
##                                       machine
##                                      "x86_64"
##                                         login
##                                     "unknown"
##                                          user
##                                    "emmanuel"
##                                 effective_user
##                                    "emmanuel"
```

Bibliography

California, L. C. S. F. (2019). Prospectus Regulatory Filing S3-ASR for Member Payment Dependent Notes. https://www.sec.gov/Archives/edgar/data/1409970/000140997019000988/0001409970-19-000988-index.htm. [Note: Accessed 31 October 2019].

Kan, L. C. P. W. (2019). Kaggle - LendingClub dataset. https://www.kaggle.com/wendykan/lending-club-loan-data. [Note: Accessed 31 August 2019].

Kim, A. and Cho, S.-B. (2019). An ensemble semi-supervised learning method for predicting defaults in social lending. *Engineering Applications of Artificial Intelligence*, 81:193–199.

Peng, R. (2012). *Exploratory data analysis with R*. Lulu. com.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.