University of Applied Sciences and Arts Northwestern Switzerland
School of Business

Assignment1: Regression Task

Group 1:
Erëza Abdullahu
Piero Jean Pier Hierro Canchari
Daniele Porumboiu
Piermichele Rosati

Supervisors:
Prof. Dr. Holger Wache
Dr. Gwendolin Wilke

Master of Science in Business Information Systems
Data Science
Autumn Semester 2023

Abstract

Real life cases most of the time can be complicated. Taking into consideration having to deal with huge datasets, can be really tricky. Firstly, you need to gain knowledge of what the task is about and what the company is working for. In one word, a general visualization should be investigated before starting to dive deep into the solution. Hence, before starting to explain the steps of achieving the solution in our case, firstly the Lending Club (LC) website has been investigated, how the company works, their data, and how they proceed with the loan credits given to the customers, have been understood. Afterward, as in every company that contains huge data, some preprocessing steps have been taken into consideration, such as deleting unnecessary rows, and columns, and transforming NA values. Of course, it is not just that, in most prediction tasks, 80% of the time is spent on preprocessing and 20% for the learning algorithms. But, let's leave some space for certain paragraphs for some more detailed information. Therefore, since the task is predicting the interest rate that should be given to a customer based on their history, five learning algorithms have been tried, to choose and analyze which predicts in the best way. The initial examination encompassed six learning algorithms such as: Linear Regression, Lasso Regression, Ridge Regression, Decision Trees, Random Forest and XGBoost. To leave space for a better-detailed explanation further on, based on the Root Mean Squared Error (RMSE), Boosting algorithms showed the best results in predicting the interest rate with an RMSE of 2.774481 in the testing phase.

Contents

Chapter 1

Introduction

LC is a Peer-to-Peer (P2P) lending industry. How LC works is that customers need credits, they fill out a credit application and the company decides based on their historical data on the interest rate that should be given to the clients. The process of giving a loan in the LC company is quite fast, specifically in a few hours. Nevertheless, the exact turnaround time will depend on the client's unique information. Therefore, the main objective of this assignment is reaching to build a prediction system that helps this company predict automatically the interest rate given to a specific person using their historical data within a short period of time. From the fact that every dataset taken from companies at the beginning is a bit complex to understand and have a clear visualisation what should be done, a proper understanding of business needs and feature semantics should be taken into consideration. A Data Science project without these two is doomed to failure. Exploring and focusing on the output desired, a numerous preprocessing steps were taken at the beginning, in particular 2 different versions of preprocessing were applied. Afterwards, since not everything is clear from the scratch of a prediction task, in each step that has been taken during this prediction journey, new removals or data transformations have been applied which led the task to better outcomes. Specifically, several Data Cleaning, Variable Transformations, Outlier Removals has been made, as well as handling of missing data. In conclusion, after all the preprocessing steps, the application of several learning algorithms were applied. To get a better understanding of how we came to this conclusion, let's dive into the detailed chapters that will explain every step taken in detail and giving reason of every decision made. Chapter 2 highlights the importance of business and data understanding in general and then goes into detail for our dataset to show an initial data exploration; Chapter 3 illustrates the 2 versions of preprocessing applied to the LC dataset, explaining all the reasons and choices made, pieces of R code and graphs generated are also described. Chapter 4 describes the models used for the prediction of the interest rate and shows the results of applying these models. Finally, Chapter 5 presents the conclusions for this task and Chapter 6 illustrates some reflections of the course assignment from our point of view.

Chapter 2

The LC Dataset

This chapter presents an overview of the importance of business and data understanding, also with regard to our domain. It then proceeds with a brief initial exploration of the data and finally a classification of the variables into categorical and numerical is shown.

2.1 Business and Data Understanding

In Data Science, the significance of understanding data cannot be overstated. At its core, data understanding involves diving deep into the available data, grasping its nuances, its structure, and its quirks. This process is not merely about seeing numbers in rows and columns; it's about comprehending the story they tell, the gaps they may have, and the potential they hold. The importance of data understanding emerges from several key aspects. First, it enables data scientists to form a clear picture of what the data can and cannot provide. This clarity is crucial for setting realistic expectations and achievable goals. It's like understanding the rules of a game before trying to play it; knowing what is possible within the boundaries of your data set paves the way for effective analysis and modeling.
Moreover, a thorough understanding of the data aids in identifying the quality issues that are almost always present in any real-world data set. Issues such as missing values, outliers, or inconsistent formatting are best addressed early in the process. Recognizing and rectifying these issues ensures that the subsequent stages of data processing, analysis, and modeling are built on a firm, reliable foundation.

In our context, we needed to do a lot of research to better understand the business behind LC and its dataset. One of these was reading the the generic FAQs from the LC website (https://www.lendingclub.com/personal-loan). By looking at the most frequently asked questions, we were able to better understand the meaning of terminologies such as interest rate on

personal loans. Since the objective of this task is to predict the interest rate itself, it was of crucial importance to be able to understand the factors that most influence our target variable. In according to Experian, some of these factors are as follows: lender, market conditions, credit score, credit report information, loan amount, repayment term, debt-to-income ratio and collateral. Having gained a sufficient understanding of this business, Descriptive Data Synthesis helped us to acquire Understanding of LC data.
In the next section, part of the data exploration carried out is briefly illustrated.

2.2 Data Exploration

An initial exploration to observe the LC dataset from a statistical point of view was carried out using the R command **summary**. It provides valuable statistical information including mean, fashion, median, minimum, maximum, first and third quartile of each variable in the dataset. It is important to note that the dataset contains 72 attributes but that not all of these in the historical dataset are available for predicting the interest rate of new loan applications. For example, *issue_d* (the month in which the loan was funded) will not be available in new applications, because the funding decision will still be open. This means that it is not possible to use these attributes for learning and for this reason, these attributes were immediately excluded. Fig. 2.1 and Fig. 2.2 show the remaining variables we considered to preprocess the data.

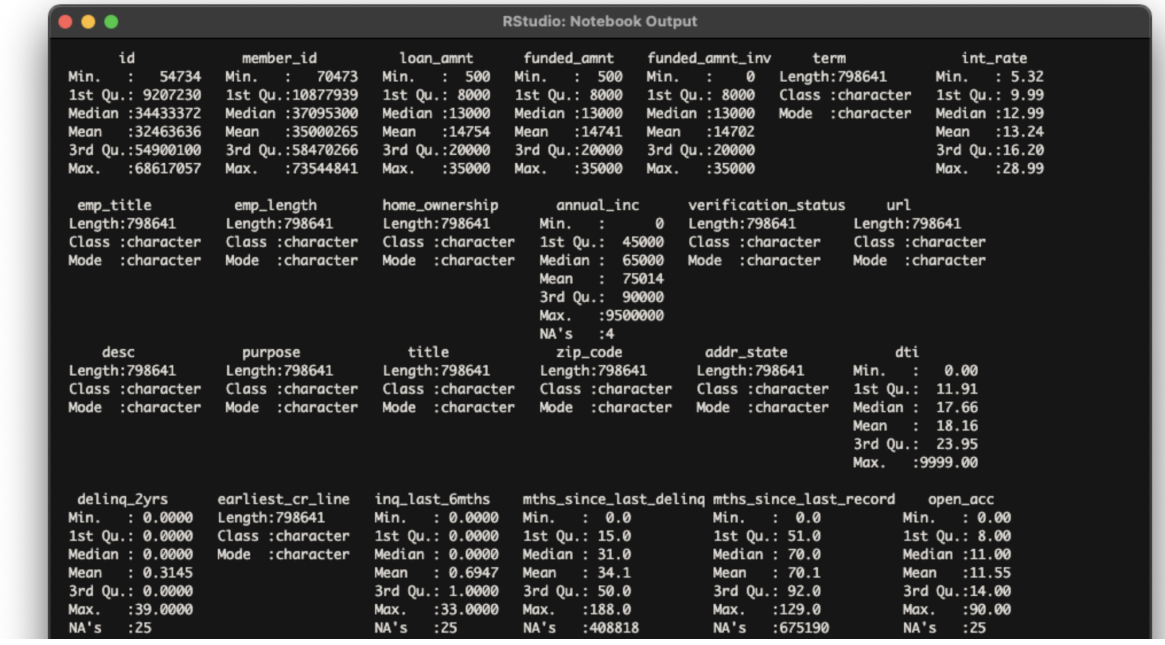

FIGURE 2.1: First part of statistical summary of LC dataset

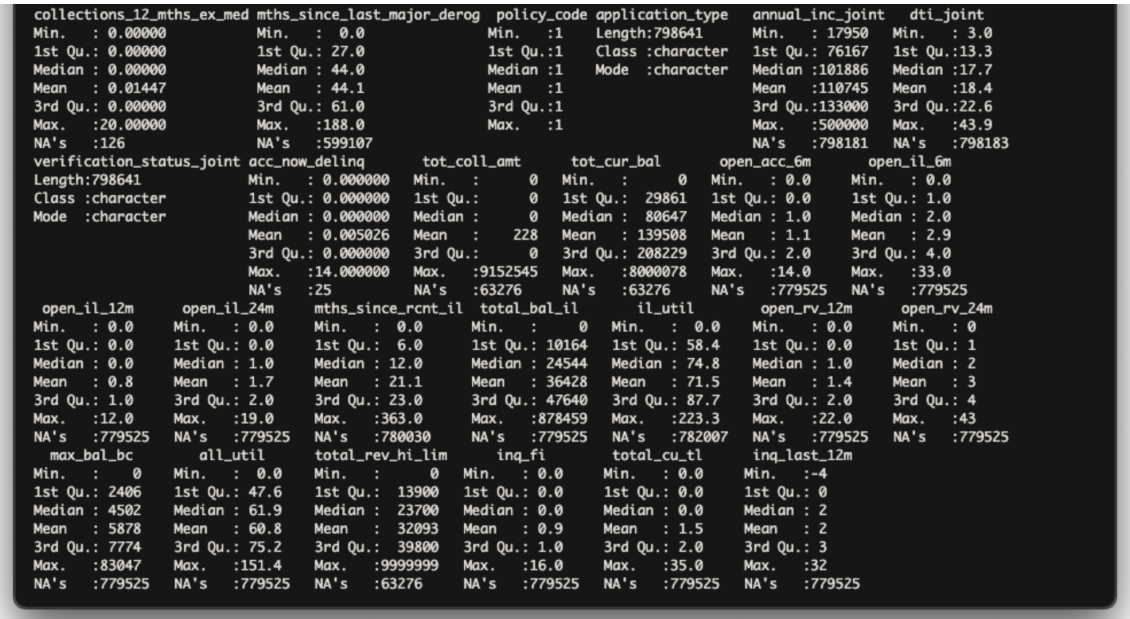

FIGURE 2.2: Second part of statistical summary of LC dataset

Based on the figures presented above, we are dealing with several problems, including missing values for many variables. As explained in the next chapter, it is also important to analyse each variable to see whether it is numeric or categorical and act accordingly (e.g. in the case of a variable that has numeric type but its values are categories, a variable transformation should be applied). At first glance, we can classify the variables into numerical (Table 2.1) and categorical (Table 2.2). Obviously, the content of the following tables is only a first visual analysis of the variables. Again, some of them are classified as categorical, but should actually be numerical, e.g. *emp_lenght*, *term*. On the other hand, there are some variables that have alphanumeric values, such as *initial_list_status* ('W' or 'F'). Variables of this type require a variable transformation from categorical to numerical. The next chapter explains in detail how each variable was preprocessed.

Variable	Description
acc_now_delinq	The number of accounts on which the borrower is now delinquent.
all_util	Balance to credit limit on all trades.
annual_inc	The self-reported annual income provided by the borrower during registration.
annual_inc_joint	The combined self-reported annual income provided by the co-borrowers during registration.
collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections.
delinq_2yrs	The number of 30 days past-due incidences of delinquency in the borrower's credit file for the past 2 years.
dti	A ratio derived by dividing the borrower's total monthly debt payments, excluding mortgage and the requested loan, by their self-reported monthly income.
dti_joint	A ratio derived from the co-borrowers' total monthly debt payments, excluding mortgages and the requested LC loan, divided by their combined monthly income.
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.
id	A unique LC assigned ID for the loan listing.
il_util	Ratio of total current balance to high credit/credit limit on all install acct
inq_last_6mths	The number of inquiries in past 6 months (excluding auto and mortgage inquiries).
inq_last_12m	Number of credit inquiries in past 12 months.
inq_fi	Number of personal finance inquiries.
int_rate	Interest rate on the loan.
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.
max_bal_bc	Maximum current balance owed on all revolving accounts.
member_id	A unique LC assigned ID for the borrower member.
mths_since_last_delinq	The number of months since the borrower's last delinquency.
mths_since_last_major_derog	Months since most recent 90-day or worse rating.
mths_since_last_record	The number of months since the last public record.
mths_since_rcnt_il	Months since most recent installment accounts opened.
open_acc	The number of open credit lines in the borrower's credit file.
open_acc_6m	Number of open trades in last 6 months.
open_il_6m	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.
open_rv_12m	Number of revolving trades opened in past 12 months.
open_rv_24m	Number of revolving trades opened in past 24 months.
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.
total_acc	The total number of credit lines currently in the borrower's credit file.
total_bal_il	Total current balance of all installment accounts.
tot_coll_amt	Total collection amounts ever owed.
tot_cur_bal	Total current balance of all accounts.
total_rev_hi_lim	Total revolving high credit/credit limit.
total_cu_tl	Number of finance trades.

TABLE 2.1: Numerical variables

Variable	Description
addr_state	The state provided by the borrower in the loan application
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers.
desc	Loan description provided by the borrower.
earliest_cr_line	The month the borrower's earliest reported credit line was opened.
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.
emp_title	The job title supplied by the Borrower when applying for the loan.
home_ownership	The home ownership status provided by the borrower during registration. Our values are: RENT, OWN, MORTGAGE, OTHER.
initial_list_status	The initial listing status of the loan. Possible values are: W, F.
last_credit_pull_d	The most recent month LC pulled credit for this loan.
loan_status	Current status of the loan.
policy_code	publicly available policy_code=1 new products not publicly available policy_code=2
purpose	A category provided by the borrower for the loan request.
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
title	The loan title provided by the borrower.
url	URL for the LC page with listing data.
verified_status_joint	Indicates if the co-borrowers' joint income was verified by LC, not verified, or if the income source was verified.
zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.

Table 2.2: Categorical variables

Chapter 3

Data Preprocessing

Prediction without data preprocessing is useless. It was mentioned also before that 80% of the prediction task's work is preprocessing. Data preprocessing is a pivotal phase in any data analysis or Machine Learning (ML) undertaking, playing a fundamental role in shaping the quality and efficacy of subsequent analyses. This critical process involves cleaning and transforming raw data into a format that is conducive to meaningful insights and model development. Therefore, let's just dive into the detailed steps taken for the LC dataset preprocessing.

3.1 Data Summarization

As already briefly mentioned in the previous chapter (Sec. 2.2), we have a clear but not exact view of our categorical and numerical variables. In the following sections, two versions of preprocessing will be illustrated: the first consists of an initial approach that essentially removes several rows and several columns, thus reducing the size of the dataset considerably. But although this preprocessing lightened the size of our LC dataset, the results obtained were not the best, neither from a numerical nor a logical point of view. The opposite was true for the second and final version of our pre-processed data: here a huge amount of cleaning and transformation work was done, examining every single column in detail, in order to understand how to prepare the data for our models in the best way.

3.2 First Preprocessing Version

In this first version of the data preprocessing, the seed was set to 42. This number is often chosen somewhat extravagantly, in reference to its use in popular culture as the "answer to the ultimate question about life, the universe and everything" in Douglas Adams' "The Hitchhiker's Guide to

the Galaxy". This made it a popular and memorable choice among programmers. Obviously, if we had kept this version of preprocessing, the seed would have been set at 1 in the future, but not having kept it, we left it at 42.

3.2.1 Used Libraries

For this first version of Data Preprocessing, mainly the 3 libraries shown in Listing 3.1 were used. As will be explained below, for some categorical variables, it was necessary to apply the one-hot coding technique. A famous and powerful library that deals with this is **fastDummies**, which creates dummy columns from columns that have categorical variables (character or factor type). For reading the dataset, the **readr** library was used, which provides various functions, such as "read_delim", which allows us to simply specify the path or name of our file. . Finally, the **caret** library provides essential functions like "preProcess", useful for performing data standardization and also the "createDataPartition" function, which was used for splitting the dataset in training and testing.

```
library(readr)
library(caret)
```

Listing 3.1: Used libraries for Preprocessing V1

3.2.2 Handling Missing Values

Unnecessary Variables Removal

As will be explained into details, in the Second Preprocessing Version, there are some unnecessary columns to be removed because some of them are not available for the new loan applications and other ones are useless for this prediction task. Listing 3.2 shows the removal of such columns.

```
lc_data <- subset(lc_data, select = -c(collection_recovery_fee, installment, issue_d,
                                       last_pymnt_amnt, last_pymnt_d, loan_status,
                                       next_pymnt_d, out_prncp, out_prncp_inv,
                                       pymnt_plan, recoveries, term, total_pymnt,
                                       total_pymnt_inv, title, desc, url, member_id,
                                       policy_code, emp_title, pymnt_plan, id, zip_code))
```

Listing 3.2: Used libraries for Preprocessing V1

Missing Values' Columns Removal

Initially, what was planned to be done, was to remove all columns that contained more than 40% NA values. Hence, from 56 columns (after the removal of the useless ones) only 31 columns were achieved at the end. This resulted on the one hand in a significant reduction in the number of columns, but on the other hand a reduction without much meaning. After carrying out this data-cleaning operation, we thought it would be better to analyse each individual column, study its values well and operate as best as possible. Using such an approach would certainly have contributed in a positive way to achieving the objective of our task. But, this will be seen and better explained in the second preprocessing version.

3.2.3 Variable Transformation

Since we were dealing with time variables but stored in the dataset as strings, the function "to_unix_time" was defined to transform string values into Unix time. As we can see from Listing 3.3, the value returned from the function is the number of seconds that have elapsed since January 1, 1970 (UTC). The function takes a string date in the format of 'month-year' (e.g., "Feb-2020"), and it prepends "01" to this date to form a full date (assuming the first day of the month). It then converts this full date into a POSIXct date-time object, which represents the number of seconds since the Unix epoch. Finally, it returns this number as a numeric value, essentially converting the given 'month-year' date into Unix time.

```
# function to replace dates with unix time
to_unix_time <- function(date) {
  tmp <- paste("01", date, sep="-")
  return (as.numeric(as.POSIXct(tmp, format="%d-%b-%Y", tz="UTC")))
}
```

Listing 3.3: to_unix_time function V1

The function described above was very useful to transform the *earliest_cr_line* and *last_credit_pull_d* columns.

```
# map dates to unix time
lc_data$earliest_cr_line <- apply(lc_data, 1, function(row) to_unix_time(row["earliest_cr_line"]))
lc_data$last_credit_pull_d <- apply(lc_data, 1, function(row) to_unix_time(row["last_credit_pull_d"]))
```

Listing 3.4: Numerical Transformation for *earliest_cr_line* and *last_credit_pull_d* V1

3.2.4 Replacing Missing Values

After removing most of the columns containing NAs, there were exactly 734,972 out of 798,641 rows in total, that contained some missing values. What was done was first to classify the categorical columns from the numerical ones that contain NAs (Listing 3.5).

```
na_mean <- c("last_credit_pull_d","earliest_cr_line","annual_inc","delinq_2yrs","inq_last_6mths",
             "revol_bal", "revol_util", "tot_coll_amt","tot_cur_bal","total_rev_hi_lim")
na_mode <- c("open_acc", "pub_rec", "total_acc", "collections_12_mths_ex_med","acc_now_delinq")
```

Listing 3.5: Categorical and Numerical columns contain NAs V1

The next step was to replace the numeric variables by their mean and the categorical variables by their mode (Listing 3.6). In order to replace these values, the following 2 functions "replace_na_by_mean" and "replace_na_by_mode" have been defined, which, as can be guessed, take as input a column and replace the missing values of that column in mean or fashion respectively (Listing 3.7).

```
# apply function to replace each column value by mean
lc_data[na_mean] <- lapply(na_mean, replace_na_by_mean)
# apply function to replace each column value by mode
lc_data[na_mode] <- lapply(na_mode, replace_na_by_mode)
```

Listing 3.6: NAs replacing for Categorical and Numerical columns V1

```
# function to replace NAs by column mean
replace_na_by_mean <- function(col) {
  tmp_mean <- as.integer(mean(lc_data[[col]], na.rm = TRUE))
  lc_data[[col]] <- ifelse(is.na(lc_data[[col]]), tmp_mean, lc_data[[col]])
}

# function to replace NAs by column mode
replace_na_by_mode <- function(col) {
  tmp_mode <- names(sort(table(lc_data[[col]]), decreasing = TRUE))[1]
  lc_data[[col]] <- ifelse(is.na(lc_data[[col]]), tmp_mode, lc_data[[col]])
}
```

Listing 3.7: Functions for replacing NAs by mean and model V1

3.2.5 Data Standardization

Standardizing numerical values, is essential because it puts different variables on the same scale, allowing for meaningful comparison and analysis. This process, known as feature scaling, is crucial for models that are sensitive to the magnitude of variables because it ensures that each feature contributes equally to the result. It also improves the convergence speed of optimization algorithms and helps to avoid biases in the model due to the scale of the features. For these reason, we standardized our numerical columns (Listing 3.9)

```
# Continuous numerical variables have to be standardized
num_columns <- c("earliest_cr_line","last_credit_pull_d","loan_amnt", "annual_inc", "dti",
                 "revol_bal", "revol_util", "total_rec_prncp", "total_rec_int",
                 "total_rec_late_fee", "tot_coll_amt", "tot_cur_bal", "total_rev_hi_lim")
```

Listing 3.8: Numerical columns V1

```
# subset data for preprocessing (include numerical variables only)
lc_data_num <- lc_data[num_columns]
# standardize data
lc_data_num <- predict(preProcess(lc_data_num, method = c("center", "scale")),
                       newdata = lc_data_num)
```

Listing 3.9: Data Standardization for numerical columns V1

3.2.6 One-Hot Encoding

One-Hot encoding is a process used to convert categorical values into a binary vector representation. This is beneficial because it allows algorithms that expect numerical input, such as most machine learning algorithms, to utilize categorical data. By encoding categories as binary vectors, each category is given equal weight, avoiding any potential ordinal implications that numbers might introduce. This is crucial for models that are sensitive to numerical values and where the categorical order does not signify any inherent order. As we can see from Listing 3.11, One-Hot encoding was applied to every our categorical column.

```
# Categorical variables have to be mapped into numerical
cat_columns <- c("emp_length", "home_ownership", "verification_status", "purpose", "addr_state",
                 "initial_list_status", "application_type")
```

Listing 3.10: Categorical columns V1

```
# subset data for preprocessing (include categorical variables only)
lc_data_cat <- lc_data[cat_columns]
# one-hot encode all categorical columns
lc_data_cat <- dummy_cols(lc_data_cat, remove_selected_columns = TRUE)
```

Listing 3.11: One-Hot encoding for categorical columns V1

After applying these Data Preprocessing steps, running just the linear regression has been tried which obtained 3.415803 as RMSE on the training set, and 3.392856 on the test set. We also tested the same model on the whole dataset, obtaining an RMSE of 3.411226. Realizing that we had a poor preprocessing performance, it was decided to proceed with the second and final preprocessing version.

3.3 Second Preprocessing Version

Also in this version of the data preprocessing, the seed was set to 1 to allow reproducibility of the experiments performed. This is followed by all the preprocessing techniques we applied to clean and structure the data in the best possible way for the models that the next chapter presents. No standardisation or normalisation was applied, since Random forests are insensitive to the scale of features because their decision-making process is based on the ordinal nature of data, not on the absolute values. Hence, the model's performance is not affected by standardization or normalization, which are techniques typically used to scale data for algorithms that rely on distance calculations.

3.3.1 Used Libraries

For this second version of Data Preprocessing, mainly the 2 libraries shown in Listing 3.12 were used. As will be explained below, for some categorical variables, it was necessary to apply the one-hot coding technique. As for the first version of preprocessing, the **readr** library was used to read the dataset but we used the "read.csv" function which allows to specify also the separator. Finally, the **ggplot2** library was used to display all our graphs.

```
library(readr)
library(ggplot2)
```

Listing 3.12: Used libraries for Preprocessing V2

3.3.2 Unnecessary Variables Removal

It's crucial to recognize that some attributes in the historical dataset are not accessible for predicting the interest rate of new loan applications. For instance, the attribute *issue_d* won't be available for new applications as the funding decision is yet to be made. Consequently, these attributes should not be used for learning purposes. The attributes unavailable for new loan applications include:

- *collection_recovery_fee*;
- *installment*;
- *issue_d*;
- *last_pymnt_amnt*;
- *last_pymnt_d*;

- *loan_status*;
- *next_pymnt_d*;
- *out_prncp*;
- *out_prncp_inv*;
- *pymnt_plan*;
- *recoveries*;
- *term*;
- *total_pymnt*;
- *total_pymnt_inv*;
- *total_rec_int*;
- *total_rec_late_fee*;
- *total_rec_prncp*.

Listing 3.13 shows how all these attributes, except *term*, have been removed.

```
lc_data <- subset(lc_data, select = -c(collection_recovery_fee, installment, issue_d,
                                       last_pymnt_amnt, last_pymnt_d, loan_status,
                                       next_pymnt_d, out_prncp, out_prncp_inv,
                                       pymnt_plan, recoveries, total_pymnt,
                                       total_pymnt_inv,total_rec_int, total_rec_late_fee,
                                       total_rec_prncp))
```

Listing 3.13: Unnecessary attributes removal V2

3.3.3 Data Cleaning

Continuing with attributes removal that were not logically relevant for the prediction from the early steps, let's have a look at the table attached below:

id	emp_title	last_credit_pull_d	application_type
member_id	title	funded_amnt	verification_status_joint
url	desc	funded_amnt_inv	revol_bal
zip_code		collections_12_mths_ex_med	revol_util

Table 3.1: Removed Variables V2

Since nothing is completely clear from the first steps of a prediction task, analyzing and trying the data step by step during this study led us to more data cleaning which helps the output to be more accurate.

As can be seen, these new variables were removed after trying out what they actually contain and if they are valuable for the interest rate prediction.
Firstly, the elimination of the *id* and *member_id* columns (which are simply numeric IDs to distinguish one record from another) is necessary in every prediction task, since the IDs are unique values for every individual that cannot be useful in the prediction of an interest rate. The same applies to the *url*, which is a simple static string consisting of the url of LC and the ID of the record. The *zip_code* has also been removed because it is only the first 3 digits and there is the variable *addr_state* which represents the state in which the borrower lives, so it would be pointless to also keep it. If we look closely, the attribute *policy_code* is completely useless for prediction, as each record always has policy code equal to 1. Therefore, this was also removed.
The *emp_title* column is removed based on the fact that there are several entries for the same job title and there are too many different values for one-hot encoding. In addition, some titles are unclear and Natural Language Processing (NLP) may be required to categorize them.
The *title* and *desc* columns also require NLP and for the same reason were removed. Since *last_credit_pull_d* refers to the most recent month in which LC credited this loan, it is possible to remove this column because it is related to historical data.

From the graphs Fig. 3.1, Fig. 3.2 and Fig. 3.3, we can observe the distributions of the following 3 variables: *loan_amnt*, *funded_amnt* and *funded_amnt_inv*.
Since *funded_amnt* and *funded_amnt_inv* are probably closely related to *loan_amnt* (as the graphs suggest), their exclusion might not result in a significant loss of information. Given that *loan_amnt* is the amount requested by the borrower, and *funded_amnt* and *funded_amnt_inv* are the amounts actually financed by the lender and investors, they are naturally very similar. Especially if most loans are fully financed. It is therefore clear that the variables are highly correlated and thus pre prevent overfitting and multicollinearity, only *loan_amnt* has been retained.

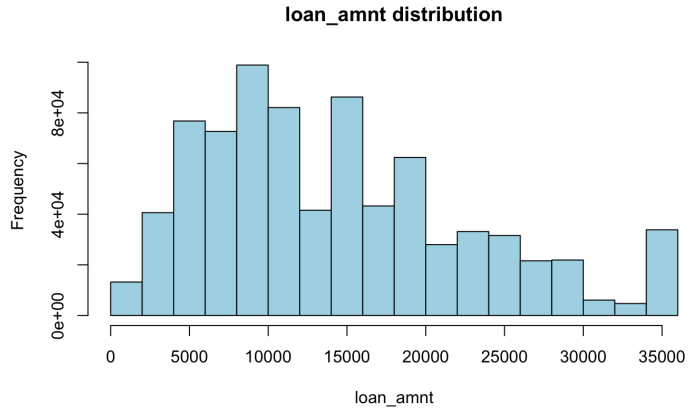

FIGURE 3.1: Loan amount distribution V2

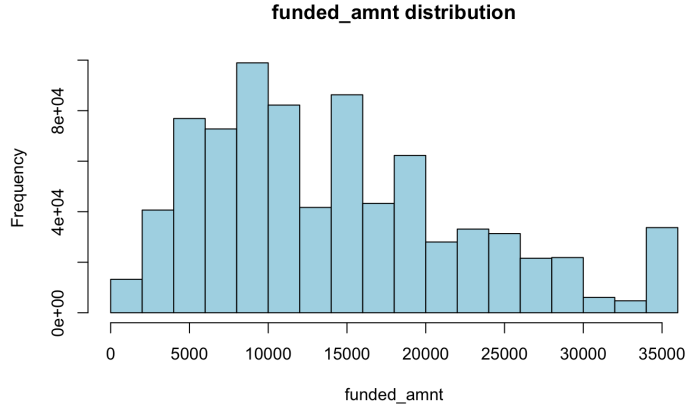

FIGURE 3.2: Distribution of funded amount committed to the loan V2

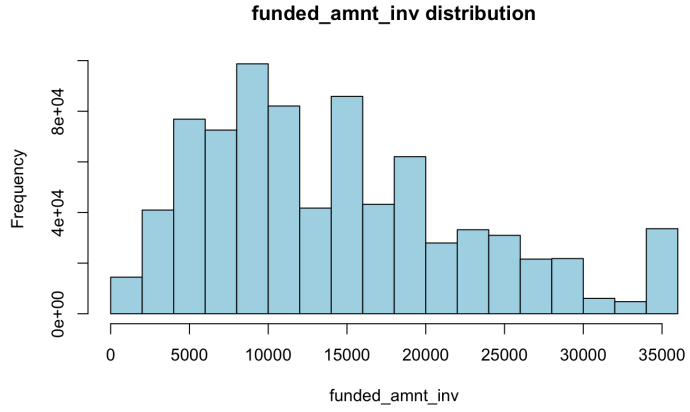

FIGURE 3.3: Distribution of funded amount committed by investors V2

3.3.4 Variable Transformation

Variable transformation in ML is a critical preprocessing step that can significantly impact the performance of a model. By transforming variables, you essentially reframe the data in a way that makes it easier for algorithms to detect patterns and learn from them.

Fig. 3.4 shows the distribution of employment length.

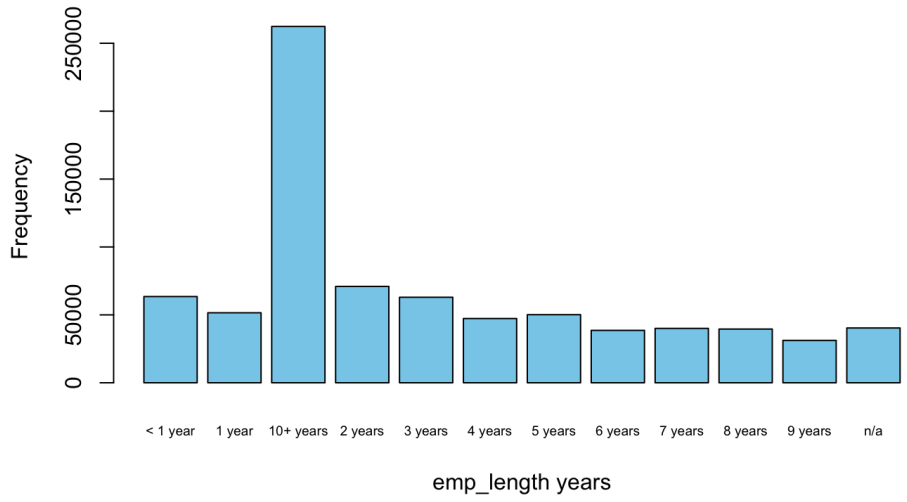

FIGURE 3.4: Distribution of numerical employment length V2

For example, as observed above, there are 40363 NAs. We can assume 40363 do not work.

Since *emp_length* seems categorical, we converted it into a factor and subsequently into a numeric variable (Listing 3.14). Fig. 3.5 shows. This conversion to numeric was crucial because, as will be explained in the next chapter, the xgboost model takes only numeric values as input. In addition, this is a perfect way to run Decision Trees, since they do not work with factor predictors that have at most 32 levels. Applying this transformation in factor and the numeric made them work.

```
lc_data$emp_length <- as.factor(lc_data$emp_length)
ggplot(data = lc_data, mapping = aes(x=int_rate,y=emp_length)) + geom_boxplot()
lc_data$emp_length <- as.numeric(lc_data$emp_length)
```

LISTING 3.14: From numerical to categorical emp_length V2

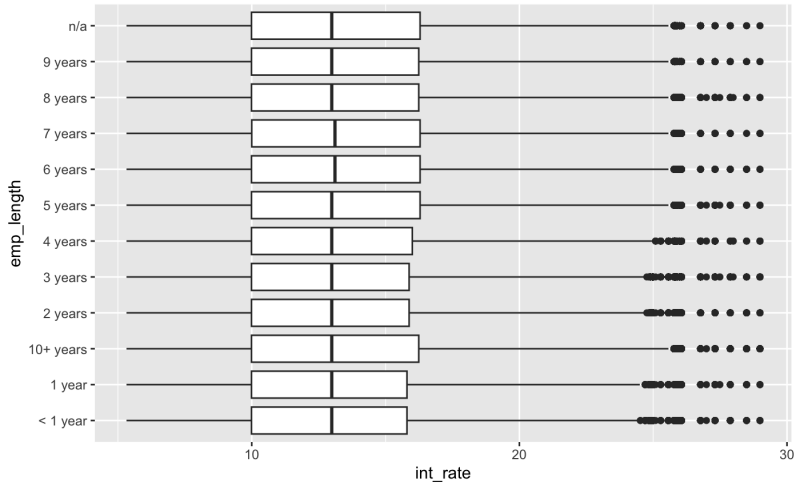

FIGURE 3.5: Distribution of categorical employment length V2

Since the number of payments on the loan plays an essential role in predicting the interest rate, *term* is very important. In summary, the boxplot of Fig 3.6, suggests that longer-term loans (60 months) not only tend to have higher median interest rates but also exhibit a greater variability in rates compared to the shorter-term loans (36 months). The presence of outliers, especially for the 60-month term, highlights the existence of loans with significantly higher interest rates, which could be a point of interest for further investigation regarding the risk assessment and pricing strategy for such loans. The *term* column was also changed to factor and then to numeric for the same reason of the boosting and decision trees explained above.

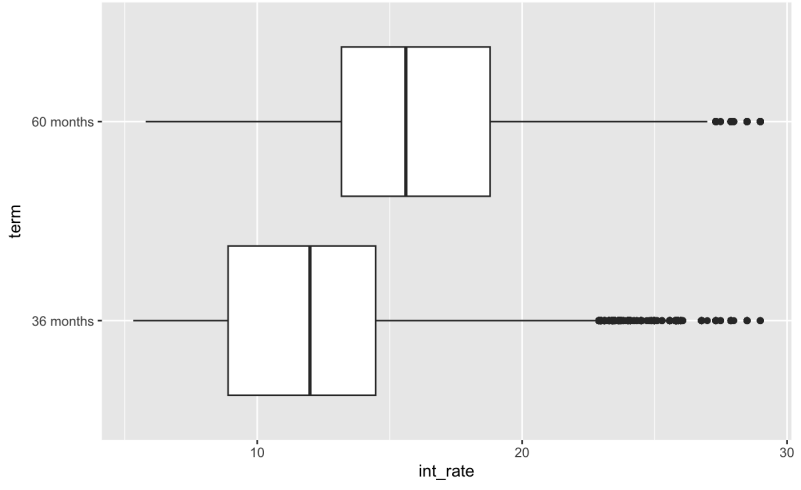

FIGURE 3.6: Distribution of loan term V2

During the data cleaning phase, our analysis revealed that the variable *home_ownership* does not show a distinct correlation with interest rates. Fig. 3.7 shows the boxplot for home ownership. Specifically, among the categories, "ANY" and "OTHER" contain 2 and 154 cases, respectively, while the "NONE" category comprises 39 cases. Although the "NONE" category appears to

demonstrate a higher interest rate compared to others, the limited sample size of 39 cases raises doubts about the reliability of this observation. Notably, the "NONE" category might pertain to individuals experiencing homelessness, prompting ethical concerns about loan provision to this demographic. We retain just mortgage, own and rent and then a factor with the subsequent numerical transformations were applied (Listing 3.15).

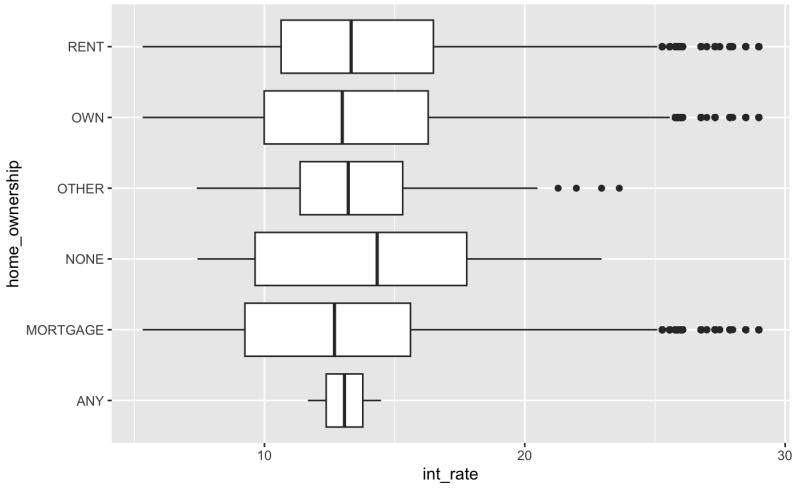

FIGURE 3.7: Distribution of home ownership V2

```
lc_data <- lc_data %>% filter(home_ownership %in% c("MORTGAGE","OWN","RENT"))
lc_data$home_ownership <- as.numeric(as.factor(lc_data$home_ownership))
```

LISTING 3.15: Retained variables for *home_ownership* V2

Most of the loan applications are Individual, this means that most of the values of the columns *dti_joint*, *annual_inc_joint* and *verification_status_joint* are Null. We would like to keep the information about Joint loans, this means that we can replace the Null values with 0 (Listing 3.16).

```
nav <- c('', ' ')
lc_data <- transform(lc_data, verification_status_joint=replace(verification_status_joint, verification_status_joint %in% nav, NA))
lc_data <-
  lc_data %>%
  mutate(dti_joint = ifelse(is.na(dti_joint) == TRUE, 0, dti_joint)) %>%
  mutate(annual_inc_joint = ifelse(is.na(annual_inc_joint) == TRUE, 0, annual_inc_joint)) %>%
  mutate(verification_status_joint = ifelse(is.na(verification_status_joint) == TRUE, 'NA', verification_status_joint))
```

LISTING 3.16: Application joint handling V2

Then *verification_status_joint* and *verification_status* columns are converted in categorical and then numerical value. The column *application_type* is obsolete, since the information about whether the loan is individual or joint is already contained in the previous variables (Listing 3.17).

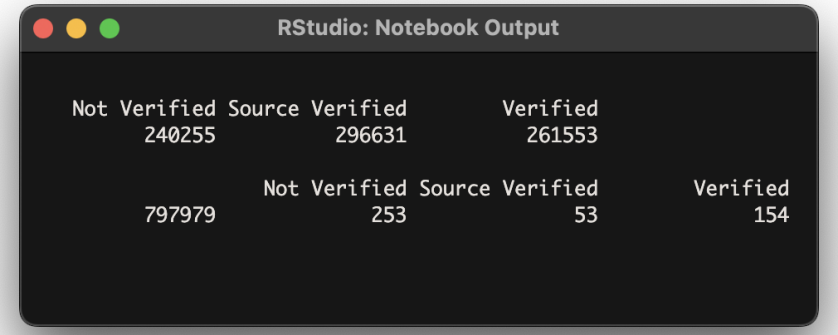

FIGURE 3.8: Verification status and joint values V2

```
lc_data$verification_status <- as.numeric(as.factor(lc_data$verification_status))
lc_data$verification_status_joint <- as.numeric(as.factor(lc_data$verification_status_joint))
lc_data <- lc_data %>% select(-application_type)
```

LISTING 3.17: *verifiction_status_joint* and *application_type* removal V2

About the *purpose* variable, we checked if "other" was a real value or an NA. Since it resulted as a real value we did not have to handle it. The only transformation that has been made is the usual factor and then numeric.

Regarding the *addr_state* variable, it can also be seen that we have no NAs (Fig. 3.10). THerefore, the usual factor numeric transformation was applied again(Listing 3.18).

The attribute *initial_list_status* denotes if a loan was first made available on the whole (W) or fractional (F) market. Given its potential usefulness, we shall retain this variable, convert it into a factor, and subsequently into a numerical format. This conversion is primarily for compatibility with the XGBoost algorithm, which requires numerical inputs.

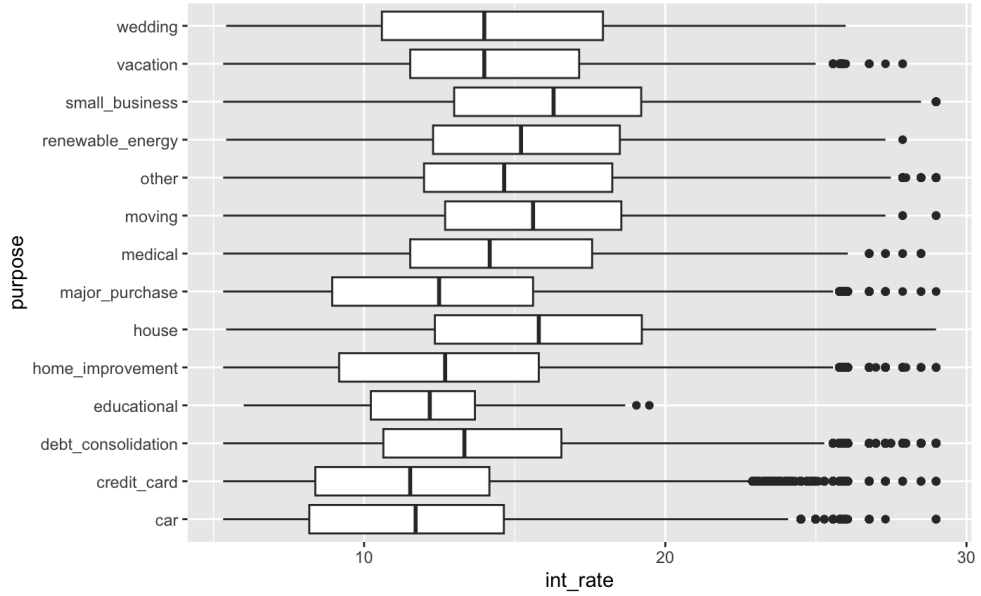

FIGURE 3.9: Distribution of *purpose* V2

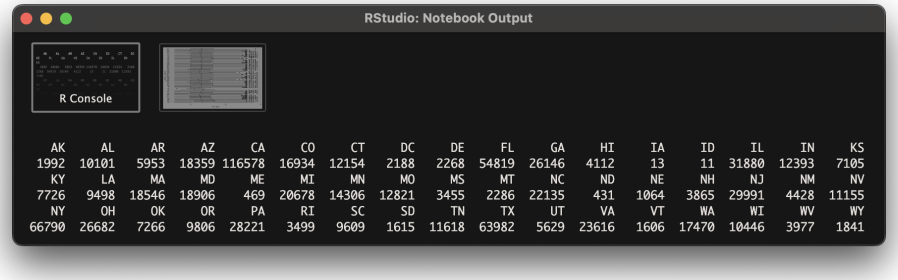

FIGURE 3.10: Values of *addr_state* V2

```
table(lc_data$addr_state)
lc_data$addr_state <- as.factor(lc_data$addr_state)
ggplot(data = lc_data, mapping = aes(x=int_rate,y=addr_state)) + geom_boxplot()
lc_data$addr_state <- as.numeric(lc_data$addr_state)
```

LISTING 3.18: *addr_state* transformation V2

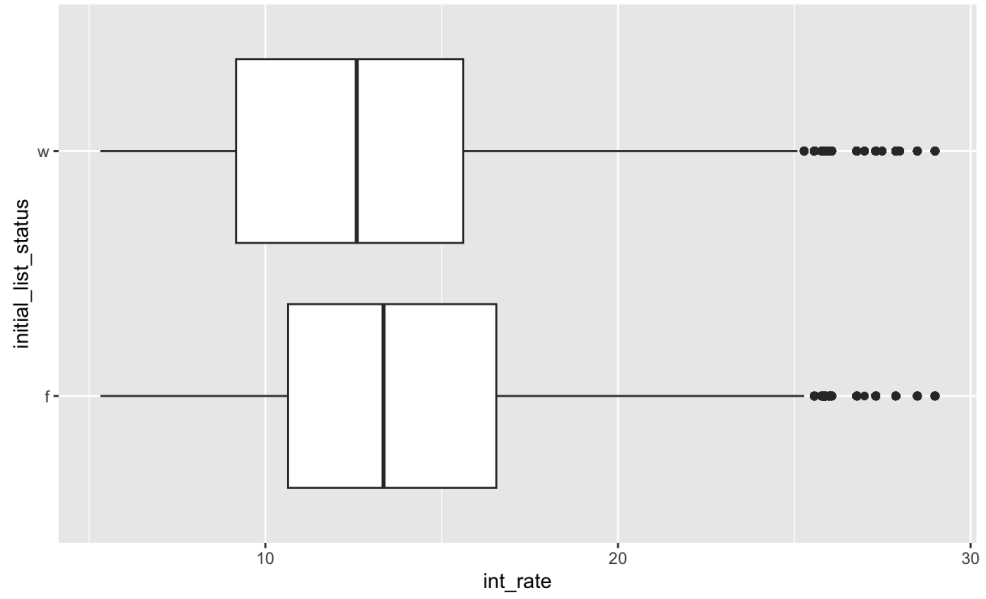

FIGURE 3.11: Distribution of *initial_list_status* V2

3.3.5 Missing Data

In the realm of predictive modeling, the presence of missing data poses a significant challenge that warrants careful consideration. As data-driven methodologies become increasingly integral to decision-making processes, the issue of missing information takes on added importance. In the context of prediction tasks, where the objective is to forecast outcomes based on historical patterns, the incomplete nature of datasets introduces complexities that can potentially compromise the accuracy and reliability of the models. Since our dataset had also some values where the data was NULL, it needed to be taken care of.

From the exploration of the data, we noticed that about the delinquency in the last 2 years, it has very few NA values and can therefore be removed (Listing 3.19).

```
lc_data <- lc_data %>%
    filter(!(is.na(delinq_2yrs)))
```

LISTING 3.19: NAs removal for delinquency in the last 2 years V2

There are some columns such as *mths_since_delinq_cat*, *mths_since_last_record*, *mths_since_rcnt_il*, and *mths_since_last_major_derog* that hold numerical data representing the elapsed months. These columns are rife with missing entries that are not accurately represented by zeros, thus discretization emerges as a suitable strategy. This process involves categorizing the numerical values into sequential bins that correspond to year-based intervals, providing a structured categorical representation of time periods. Concurrently, for any missing data, a distinct category is designated to

maintain data integrity and ensure proper analytical treatment. As we can see from Listing 3.20, we applied the discretization replacing the numerical year with the categorical one. The code is shown only for the *mths_since_delinq_cat* column since it is the same for the other ones. These 4 variables were also transformed first into factor and then into numeric.

```
lc_data <- lc_data %>%
  mutate(mths_since_delinq_cat = ifelse(
    is.na(mths_since_last_delinq) == TRUE,
    "NONE",
    ifelse(
      mths_since_last_delinq <= 12,
      "Less_1_Y",
      ifelse(
        mths_since_last_delinq <= 24,
        "Less_2_Y",
        ifelse(
          mths_since_last_delinq <= 36,
          "Less_3_Y",
          ifelse(mths_since_last_delinq <= 48, "Less_4_Y", "More_4_Y")
        )
      )
    )
  )) %>% select(-mths_since_last_delinq)

lc_data$mths_since_delinq_cat <- as.factor(lc_data$mths_since_delinq_cat)
ggplot(data = lc_data, mapping = aes(x=int_rate,y=mths_since_delinq_cat))+geom_boxplot()
lc_data$mths_since_delinq_cat <- as.numeric(lc_data$mths_since_delinq_cat)
```

LISTING 3.20: Discretization for months since last delinquency V2

Boxplots for the 4 variables to which discretization was applied are shown below.

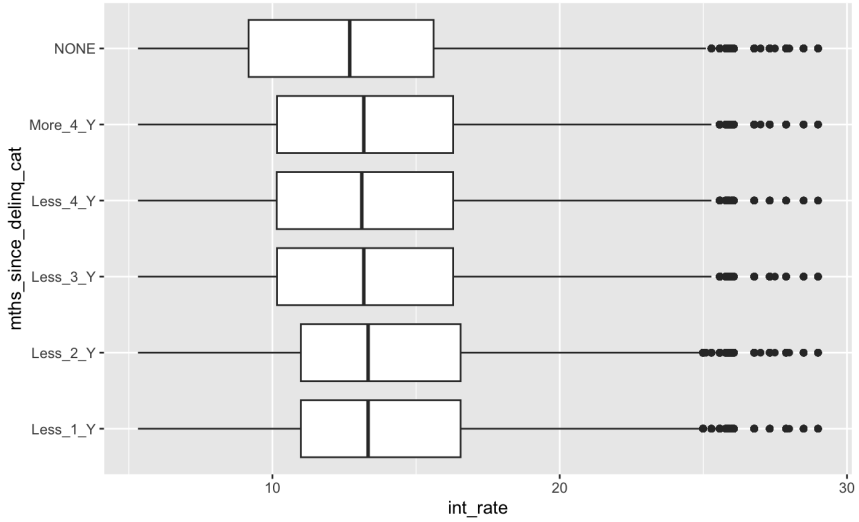

FIGURE 3.12: Distribution of *mths_since_delinq* after discretization V2

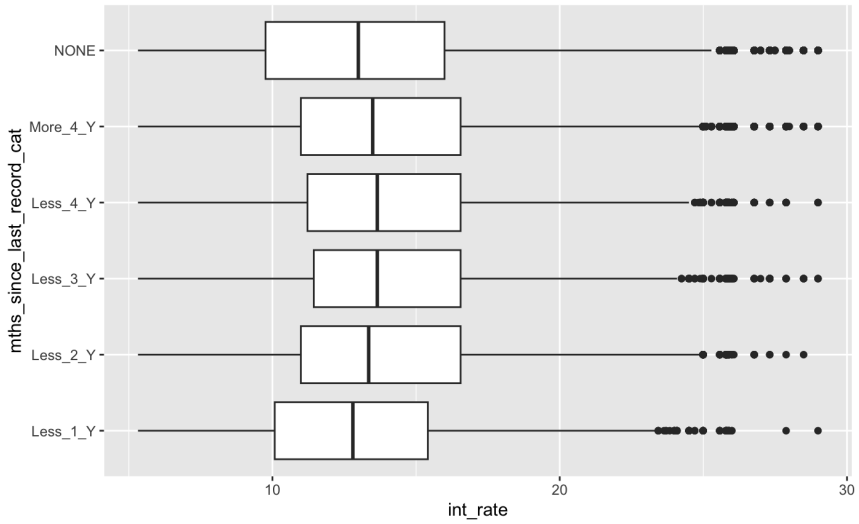

FIGURE 3.13: Distribution of *mths_since_last_record* after discretization V2

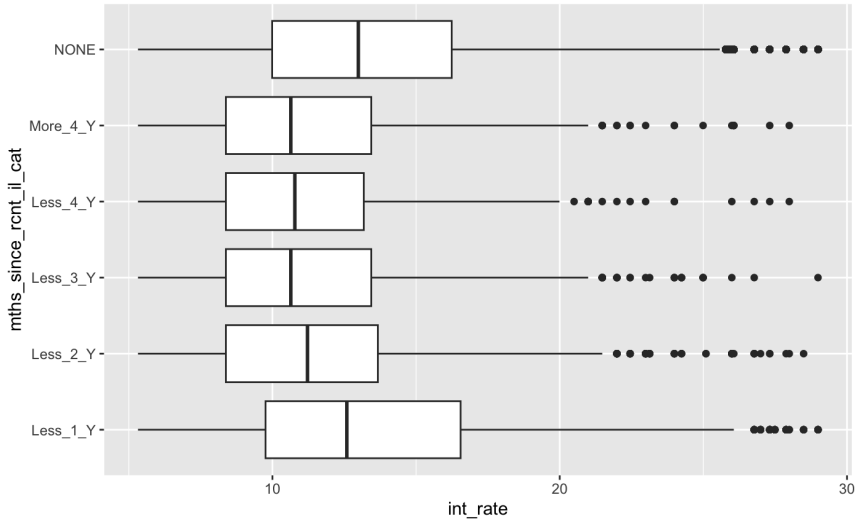

FIGURE 3.14: Distribution of *mths_since_rcnt_il* after discretization V2

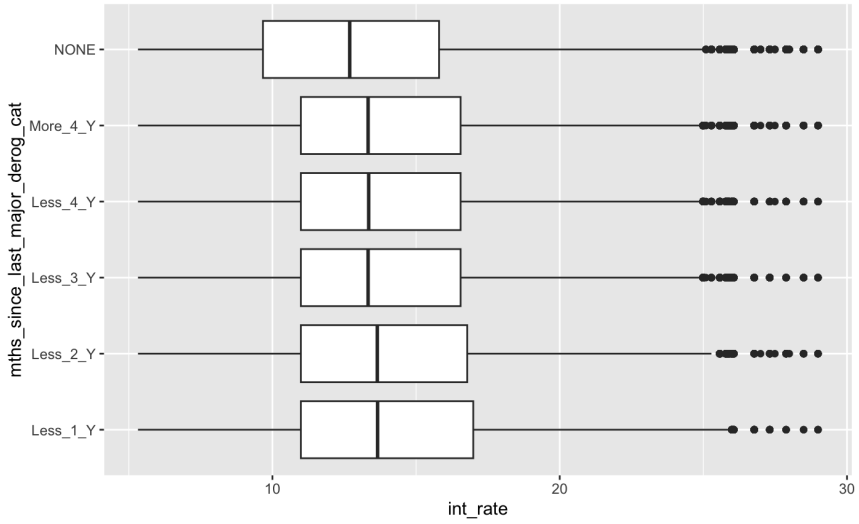

FIGURE 3.15: Distribution of *mths_since_major_derog* after discretization V2

The variable *earliest_cr_line* records the month and year when the borrower's first credit line was reported. Despite being comprised of only month and year, the diversity of unique entries is considerable. A practical solution would be to translate these dates into a numerical format, utilizing Unix Time as the standard (Listing 3.21). Unix Time quantifies time as the cumulative seconds since the epoch at 00:00:00 UTC on 1 January 1970. Owing to the absence of specific days in this column, we'll use the initial day of each month as our standard reference point for conversion.

```
lc_data <- lc_data %>%
    filter(!(is.na(earliest_cr_line)))

# function to replace dates with unix time
to_unix_time <- function(date) {
  tmp <- paste("01", date, sep="-")
  return (as.numeric(as.POSIXct(tmp, format="%d-%b-%Y", tz="UTC")))
}

# map dates to unix time
lc_data$earliest_cr_line <- apply(lc_data, 1, function(row) to_unix_time(row["earliest_cr_line"]))
```

LISTING 3.21: Numerical transformation for *earliest_cr_line* V2

After all these preprocessing steps, the following missing values are present:

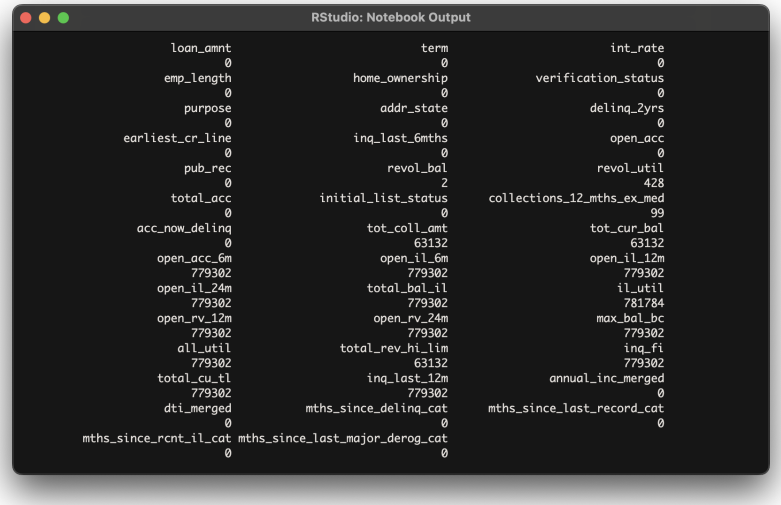

FIGURE 3.16: NAs after preprocessing steps V2

From the Fig. 3.16, we can notice that there are *revol_bal* and *revol_util* with only few NAs. Since we cannot replace them with 0, we proceeded to filter only the values that are not NA from these 2 columns.

Once the filter has been applied, the columns in Fig. 3.17 are the only ones that retain the missing values.

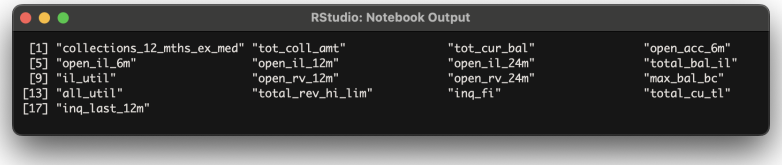

FIGURE 3.17: Filtered NAs after preprocessing steps V2

As a final variable transformation step, missing values in these columns were replaced by 0 where possible (Listing 3.22).

```
lc_data <-
  lc_data %>%
  mutate(open_acc_6m = ifelse(is.na(open_acc_6m) == TRUE, 0, open_acc_6m)) %>%
  mutate(tot_cur_bal = ifelse(is.na(tot_cur_bal) == TRUE, 0, tot_cur_bal)) %>%
  mutate(open_il_6m = ifelse(is.na(open_il_6m) == TRUE, 0, open_il_6m)) %>%
  mutate(open_il_12m = ifelse(is.na(open_il_12m) == TRUE, 0, open_il_12m)) %>%
  mutate(open_il_24m = ifelse(is.na(open_il_24m) == TRUE, 0, open_il_24m)) %>%
  mutate(total_bal_il = ifelse(is.na(total_bal_il) == TRUE, 0, total_bal_il)) %>%
  mutate(il_util = ifelse(is.na(il_util) == TRUE, 0, il_util)) %>%
  mutate(open_rv_12m = ifelse(is.na(open_rv_12m) == TRUE, 0, open_rv_12m)) %>%
  mutate(total_rev_hi_lim = ifelse(is.na(total_rev_hi_lim) == TRUE, 0, total_rev_hi_lim)) %>%
  mutate(max_bal_bc = ifelse(is.na(max_bal_bc) == TRUE, 0, max_bal_bc)) %>%
  mutate(all_util = ifelse(is.na(all_util) == TRUE, 0, all_util)) %>%
  mutate(inq_fi = ifelse(is.na(inq_fi) == TRUE, 0, inq_fi)) %>%
  mutate(total_cu_tl = ifelse(is.na(total_cu_tl) == TRUE, 0, total_cu_tl)) %>%
  mutate(inq_last_12m = ifelse(is.na(inq_last_12m) == TRUE, 0, inq_last_12m)) %>%
  mutate(open_rv_24m = ifelse(is.na(open_rv_24m) == TRUE, 0, open_rv_24m)) %>%
  mutate(tot_coll_amt = ifelse(is.na(tot_coll_amt)== TRUE,0, tot_coll_amt)) %>%
  mutate(collections_12_mths_ex_med = ifelse(is.na(collections_12_mths_ex_med)== TRUE,0, collections_12_mths_ex_med))
```

LISTING 3.22: Replacement of NAs by 0 where possible V2

3.3.6 Outliers Removal

The main box of Fig. 3.18 shows the middle 50% of the interest rate data, the line inside the box represents the median, and the whiskers extend to the data points that fall within a certain distance from the box. Points outside of this are considered **outliers**. Outliers might be removed from data analysis because they can skew statistical calculations and models, possibly leading to misleading conclusions. They are often excluded if they result from data entry errors, measurement errors, or if they are not representative of the population being studied. For these reasons, they were removed from our analysis.

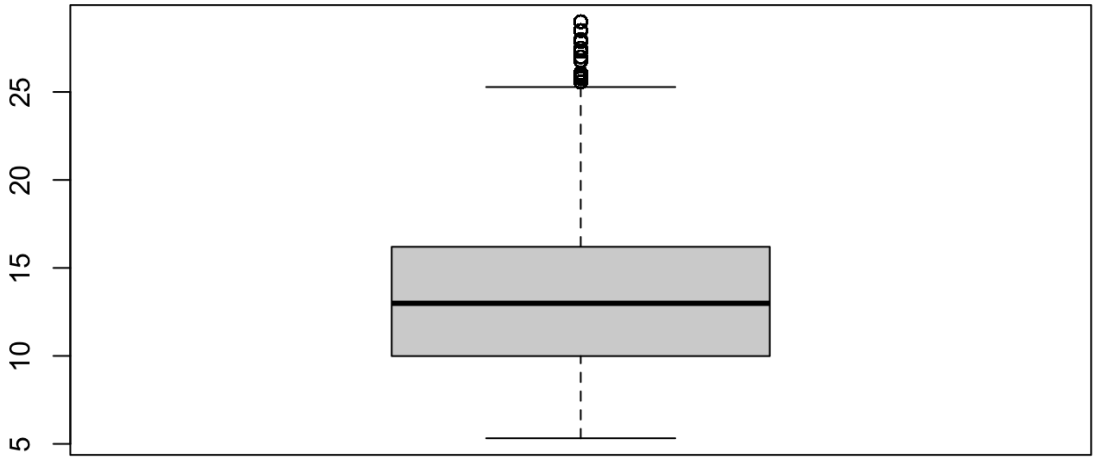

FIGURE 3.18: Outliers of interest rate V2

Chapter 4

Learning Algorithms

In this Data science tasks, several learning algorithms served as a backbone for the starting point of this journey. The algorithms chosen to be used from the course understandings gone through this semester are:

- Linear Regression;
- Lasso Regression;
- Ridge Regression;
- Decision Trees;
- Random Forest;
- XGBoost.

Linear regression, a fundamental method, seeked to establish a linear relationship between input features and a target variable, making it a cornerstone for predictive modeling. Expanding upon this foundation, regularization techniques like Lasso regression and Ridge regression came into play, introducing mechanisms to combat overfitting and enhance model generalization.

Decision Trees, a non-parametric supervised learning method was used to create a model that predicts the value of our target variable by learning simple decision rules inferred from the data features.

Random Forest harnessed the collective wisdom of decision trees to deliver robust and accurate predictions.

Meanwhile, XGBoost, an efficient and scalable gradient boosting framework, stands out for its prowess in handling complex relationships.

Furthermore, the K-Fold Cross Validation technique, which consist of dividing the dataset into subsets or folds, seemed to be helpful in the evaluation metrics which sent us to the conclusion of trying every algorithm with the help of the K-Fold mentioned above:

- Linear Regression 5-Fold;
- Random Forest 5-Fold ;
- XGBoost 5-Fold;
- Linear Regression 10-Fold;
- XGBoost 10-Fold.

Even though the K-Fold technique has shown good results, especially in XGBoosting, in this particular case the results seemed better without it.
In conclusion, together, these algorithms form a powerful arsenal, each with its unique strengths, contributing to the advancement of predictive analytics and data science applications where in our case XGBoost has shown its best practices of predicting the interest rate with the smallest RMSE. All these algorithms were evaluated on the data to which the second version of preprocessing was applied (Sec. 3.3). The more detailed information and comparative models will be seen in the further sections below.

4.1 Linear Regression

Multiple Linear Regression is an extension of simple linear regression, a statistical method used for predicting the relationship between a dependent variable and two or more independent variables. The goal of multiple linear regression is to estimate the coefficients that minimize the sum of squared differences between the predicted and actual values of the dependent variable.

Our Linear Regression model performs decently with a Testing Mean Squared Error (MSE) of 10.76845. As we can see from the residuals plot (Fig. 4.1), the spread of residuals is roughly constant and in the residual density plot (Fig. 4.2) we can see that the curve is slightly skewed towards the left, but it is close to zero. We can consider this model as a good model, but there is still some room of improvement.

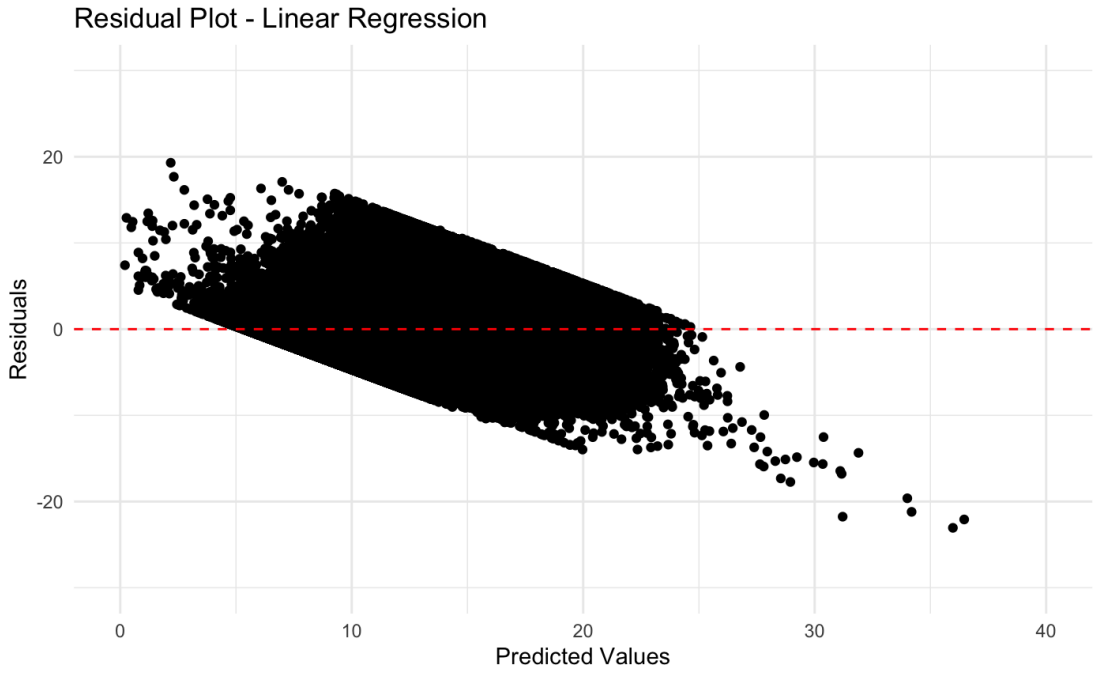

FIGURE 4.1: Residuals of Linear Regression

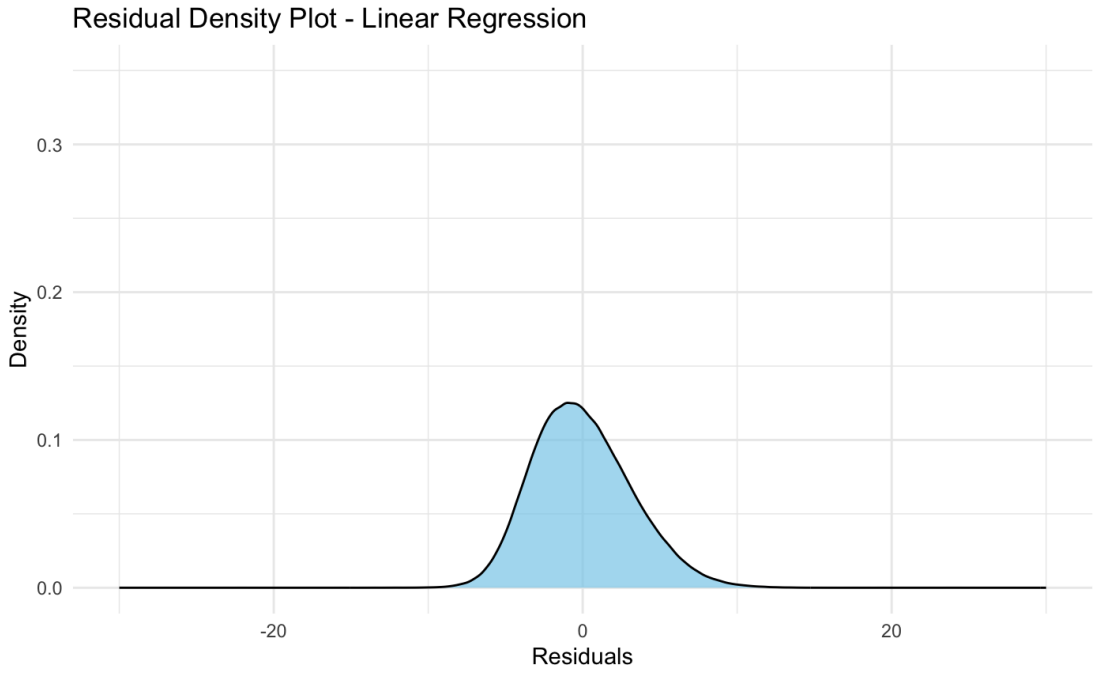

FIGURE 4.2: Residuals Density of Linear Regression

4.2 Lasso and Ridge Regression

Ridge regression and Lasso regression are both regularization techniques used in linear regression to prevent overfitting and improve the performance of a model. They add a penalty term to the standard linear regression objective function, which helps in controlling the complexity of the model by discouraging the coefficients from becoming too large. Ridge regression uses the L2 norm penalty, while lasso regression uses the L1 norm penalty. Ridge tends to shrink coefficients towards zero (Fig. 4.4), while lasso can shrink coefficients to zero (Fig. 4.3), effectively performing feature selection. Unfortunately in our case these two techniques weren't effective and they produced the worst results compared to all other models (Table 4.2 and Table 4.1).

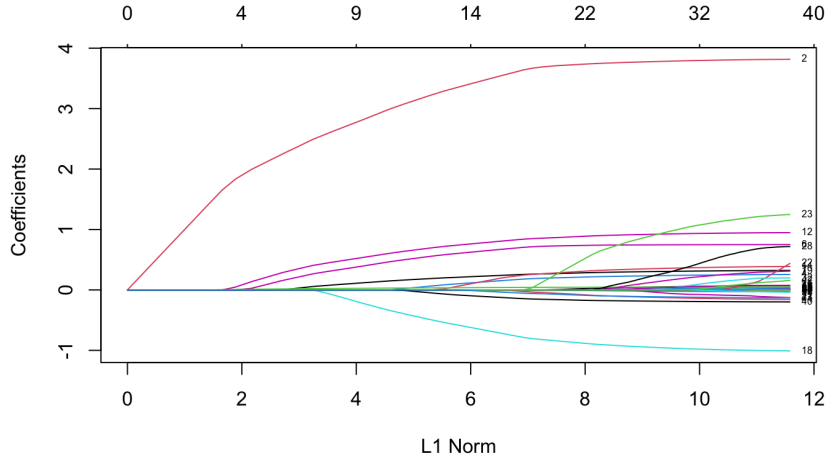

FIGURE 4.3: Lasso regression

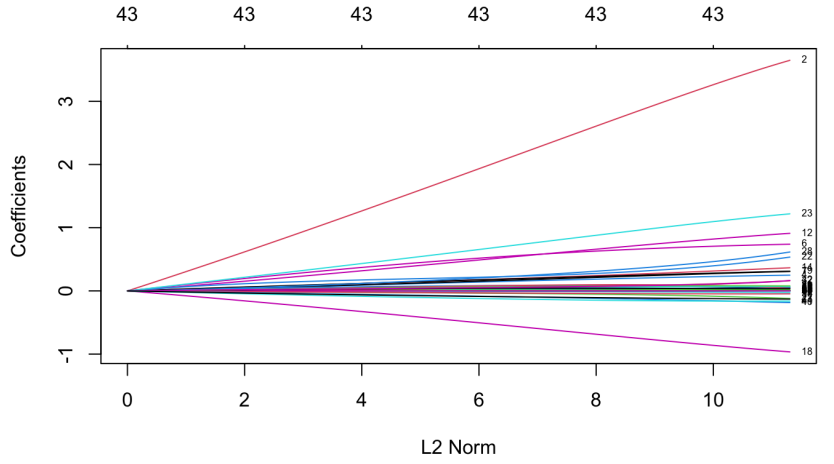

FIGURE 4.4: Ridge regression

4.3 Decision Trees

Decision trees are a type of ML algorithm that uses a set of binary rules to make decisions, much like the branches of a tree. They are easy to interpret and can handle both numerical and categorical data. Analyzing our decision tree results: Training MSE (13.38487) and Testing MSE (13.4105) are close, suggesting the model is consistent but with high error. The RMSE values (Training: 3.658534, Testing: 3.662034) confirm the average error magnitude is significant. The Mean Absolute Error (MAE) (Training: 2.953433, Testing: 2.953717) indicates average absolute errors are also high. Low R^2 values (Training: 0.2619011, Testing: 0.2614654) imply the model explains only a small fraction of the variance, reflecting its poor predictive power and inability to capture the complexity of the data. So, these results suggest the model has not performed very well. This could point to a decision tree model that is too simple or not well-tuned to capture the complexities of the dataset, a common critique when a decision tree over-simplifies the problem or fails to capture important patterns.

4.4 Random Forest

Random Forest is an ensemble learning method widely used for both classification and regression tasks. It belongs to the family of tree-based models and is known for its high accuracy, robustness, and ability to handle complex datasets. The "forest" in Random Forest is a collection of decision trees, where each tree is trained independently and contributes to the final prediction through a process of averaging or voting.

The evalutation of this model returned some interesting results: the Training MSE is very small with a value of 2.182288 while the Tesing MSE is more realistic and with a smaller value compared to the Linear Regression with 8.346162. Also in this case the Residual and Residual Density plots suggest us that the model has a good performance in predicting the variables, much better than the Linear Regression. But these results could also indicate that the model is overfitting.

Random Forest provides a measure of feature importance based on how much each feature contributes to the reduction of impurity (e.g., Gini impurity) across all the trees. This information can be valuable for feature selection (4.7). We also tried to run all the models using only the variables from this feature selection, but without improvements on the results.

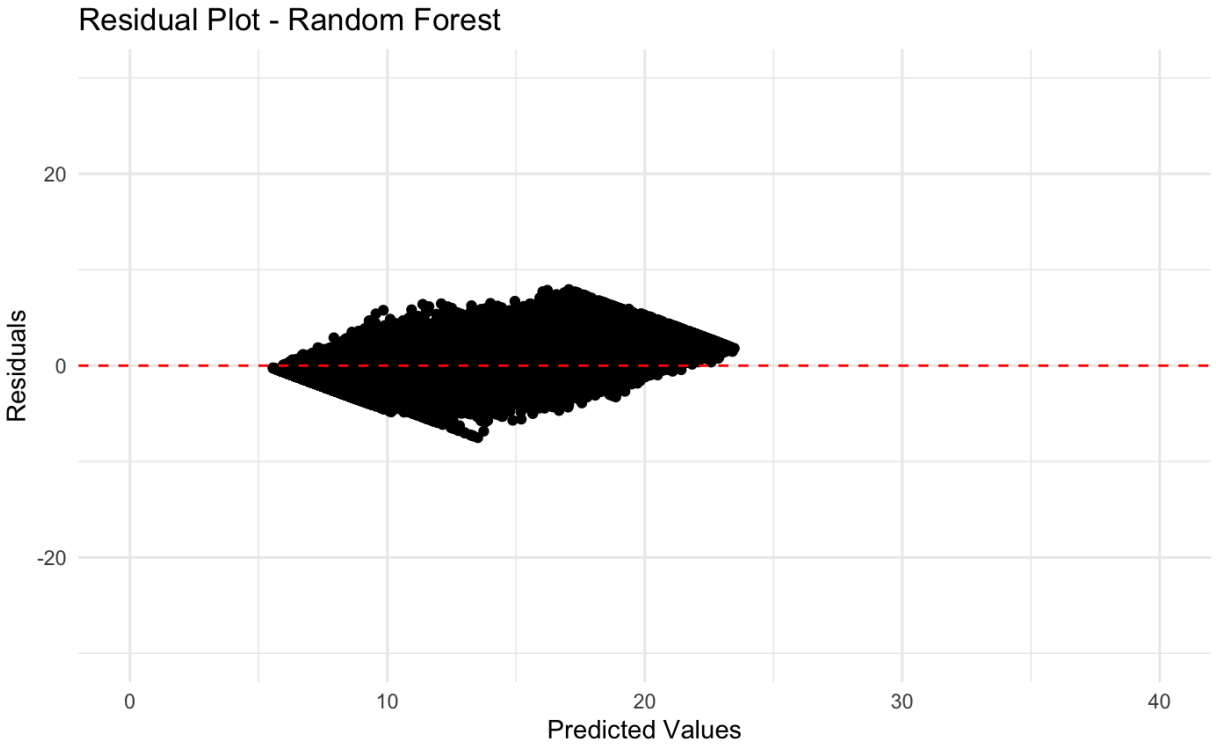

FIGURE 4.5: Residuals of Random Forest

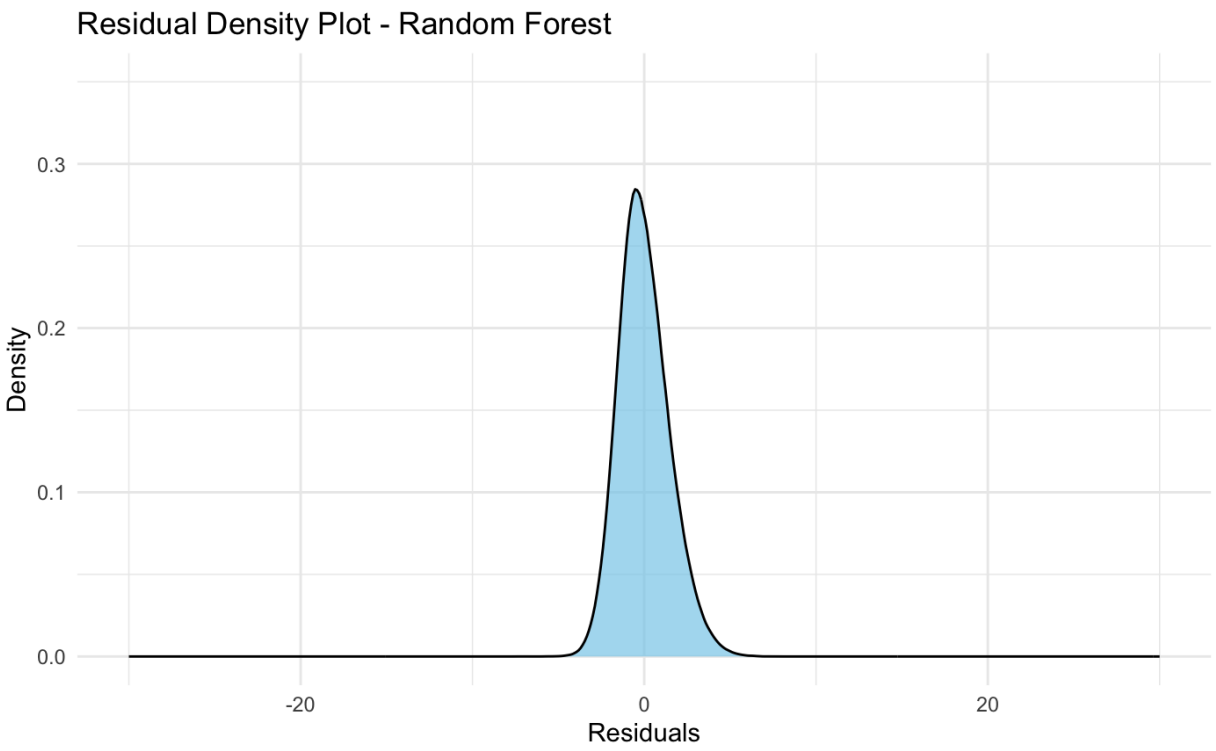

FIGURE 4.6: Residuals Density of Random Forest

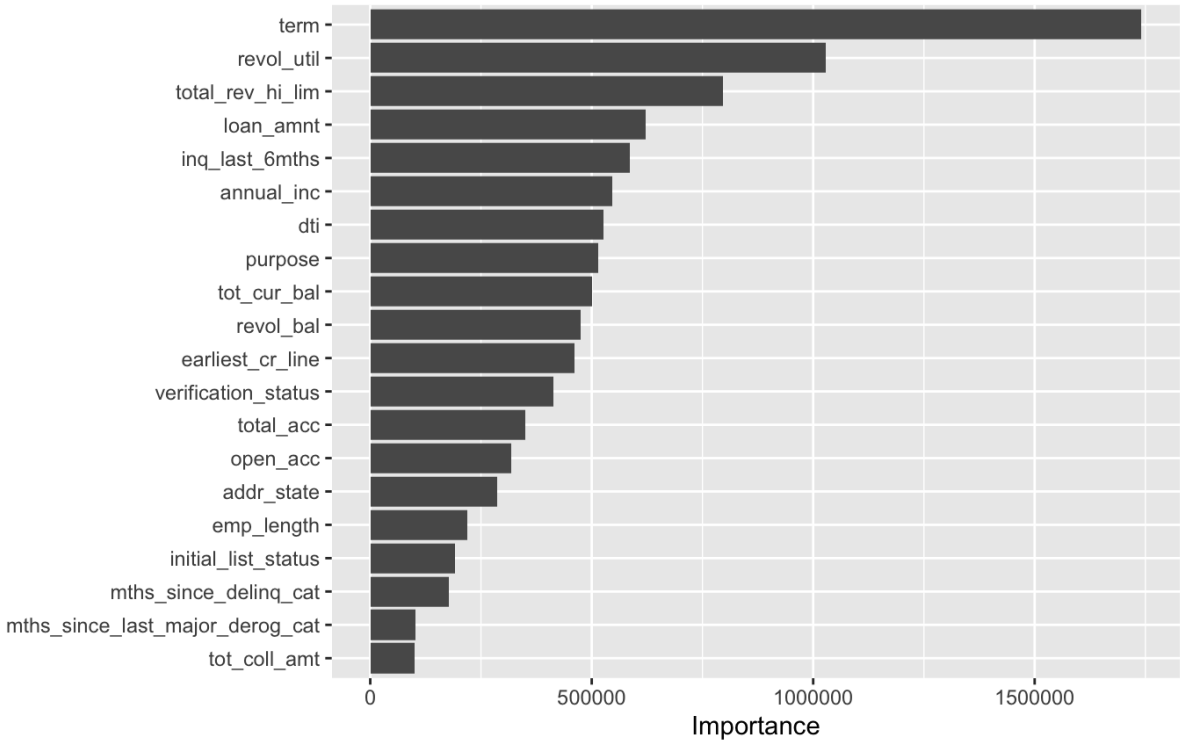

FIGURE 4.7: Random Forest Feature Importance Plot

4.5 Boosting

Gradient Boosting is a powerful ensemble learning technique used for both classification and regression tasks. It belongs to the family of boosting algorithms and is known for its high predictive accuracy and ability to handle complex relationships in data. The primary idea behind gradient boosting is to sequentially build a series of weak learners (typically decision trees) and combine their predictions, with each new learner focusing on correcting the errors made by the existing ensemble. The "gradient" in gradient boosting refers to the optimization of a loss function. The algorithm minimizes the residual errors (the differences between predictions and actual values) by adjusting the weights of the weak learners during each iteration.

In our case XGBoost was used to train the model and this model was the best performing one by having the best evaluation results and it is not showing any sign of overfitting. The Training and Testing MSE values are respectively 7.430606 and 7.697747.

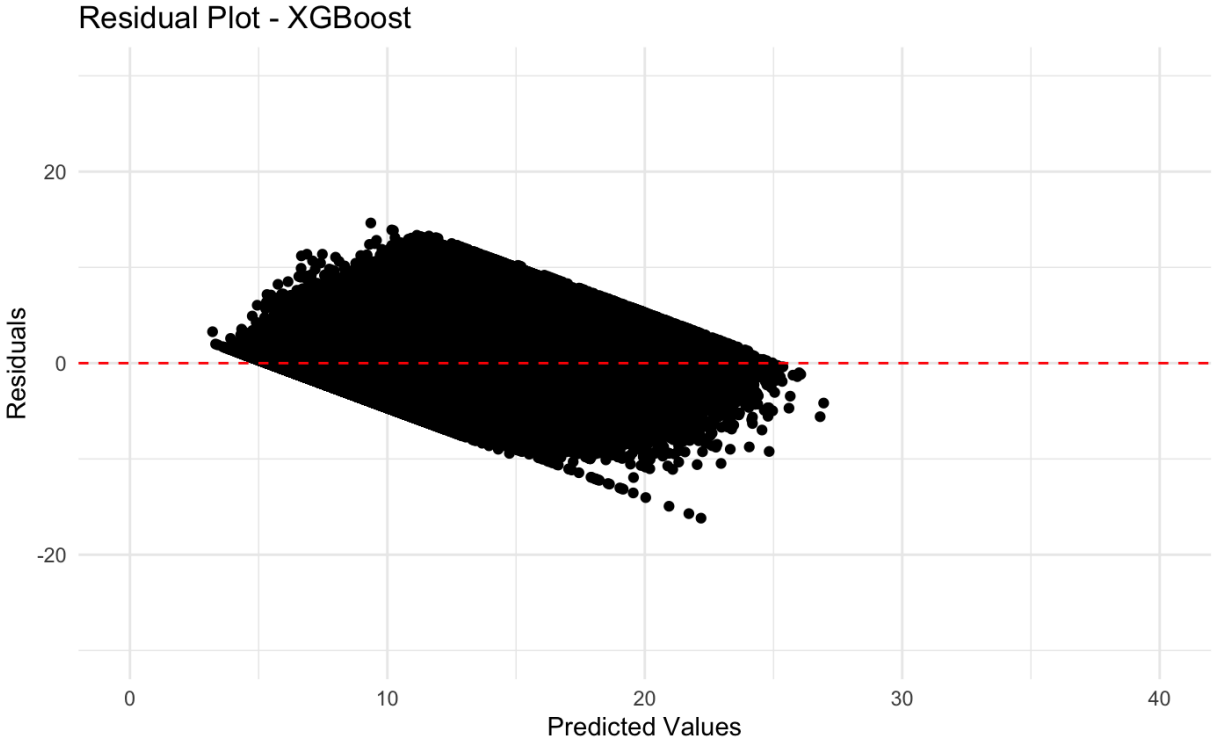

FIGURE 4.8: Residuals of XGBoost

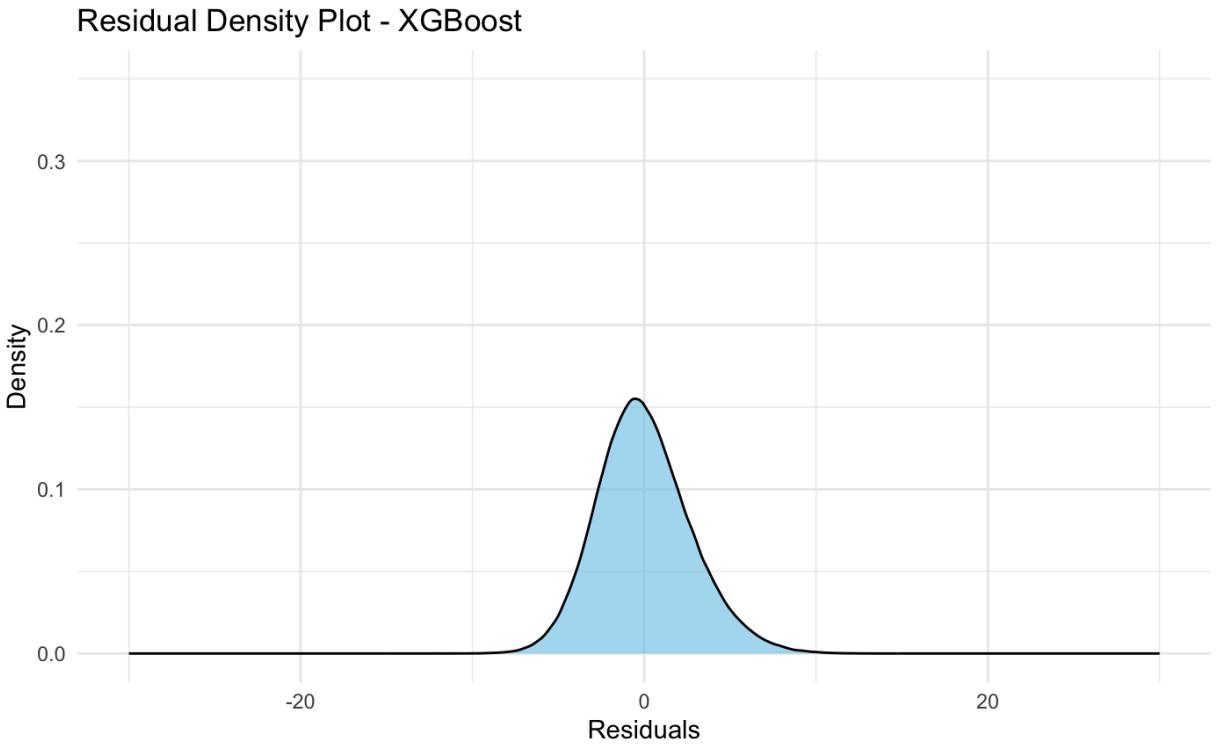

FIGURE 4.9: Residuals Density of XGBoost

4.6 K-Fold Cross-Validation

The Linear Regression, Random Forest and Boosting models were evaluated by Cross-Validation with k=5 and k=10. Testing K-Fold Cross-Validation with k=5 and then k=10 allows the impact of different levels of data segmentation on model performance to be assessed. With k=5, each model is trained and validated on larger segments of the data, which can be faster and less computationally demanding. Moving to k=10 offers greater granularity, with larger but smaller data segments for validation, often providing a more accurate estimate of the generalisation capability of the model, but at the cost of more computational resources and time. The Linear Regression and XGBoost models were evaluated across four metrics: MSE, RMSE, MAE and R^2. The random forest was evaluated with MSE, RMSE, MAE and Out-of-Bag (OOB).

4.6.1 5-Fold Cross Validation

Linear Regression with k=5

Figure 4.10 shows the performance of the Linear Regression model evaluated with 5-Fold Cross Validation. Each fold represents a unique subset of data used for testing and training. Variability across folds suggests model performance fluctuates with different data subsets. Applying 5-Fold Cross Validation ensures the model's robustness by training and testing it on different data segments, reducing bias and variance, and giving a more generalized performance indication.

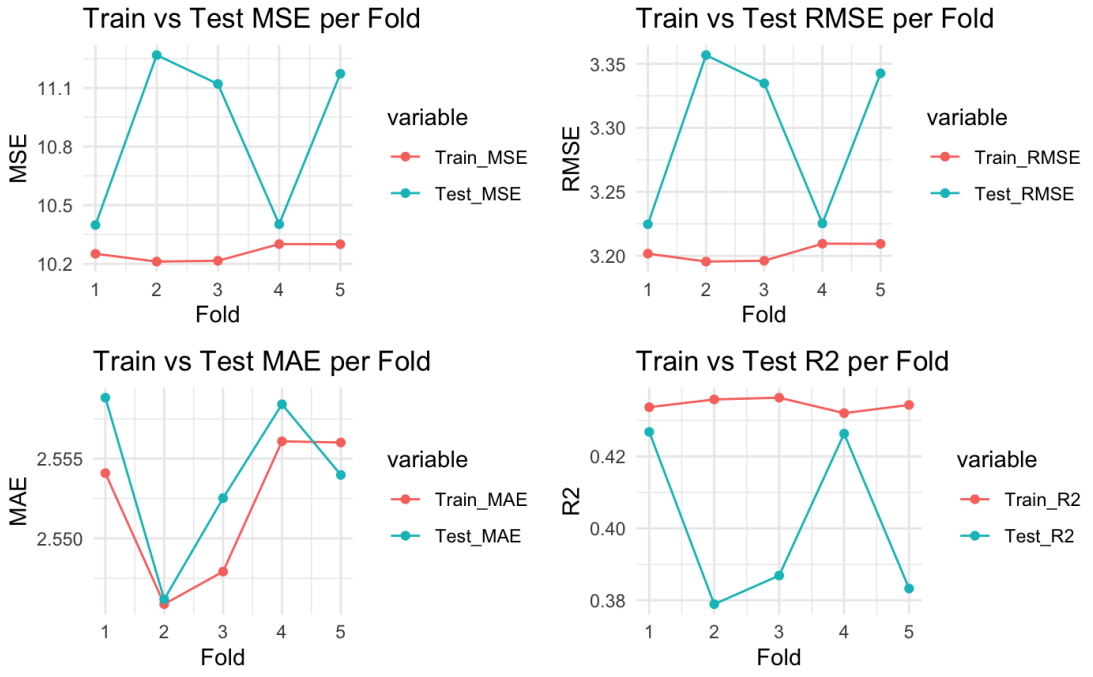

FIGURE 4.10: All metrics applying Linear Regression with 5-Fold Cross Validation

Random Forest with k=5

Figure 4.11 displays performance metrics for the Random Forest model evaluated with 5-Fold Cross Validation on both training and testing data. MSE, RMSE, and MAE are consistently lower for training data compared to testing data, indicating good fit but potential overfitting. The OOB error, unique to Random Forest, varies across folds, showing how the model generalizes on unseen data. Applying 5-Fold Cross Validation here helps to assess the model's stability and predictive performance across different subsets of the data.

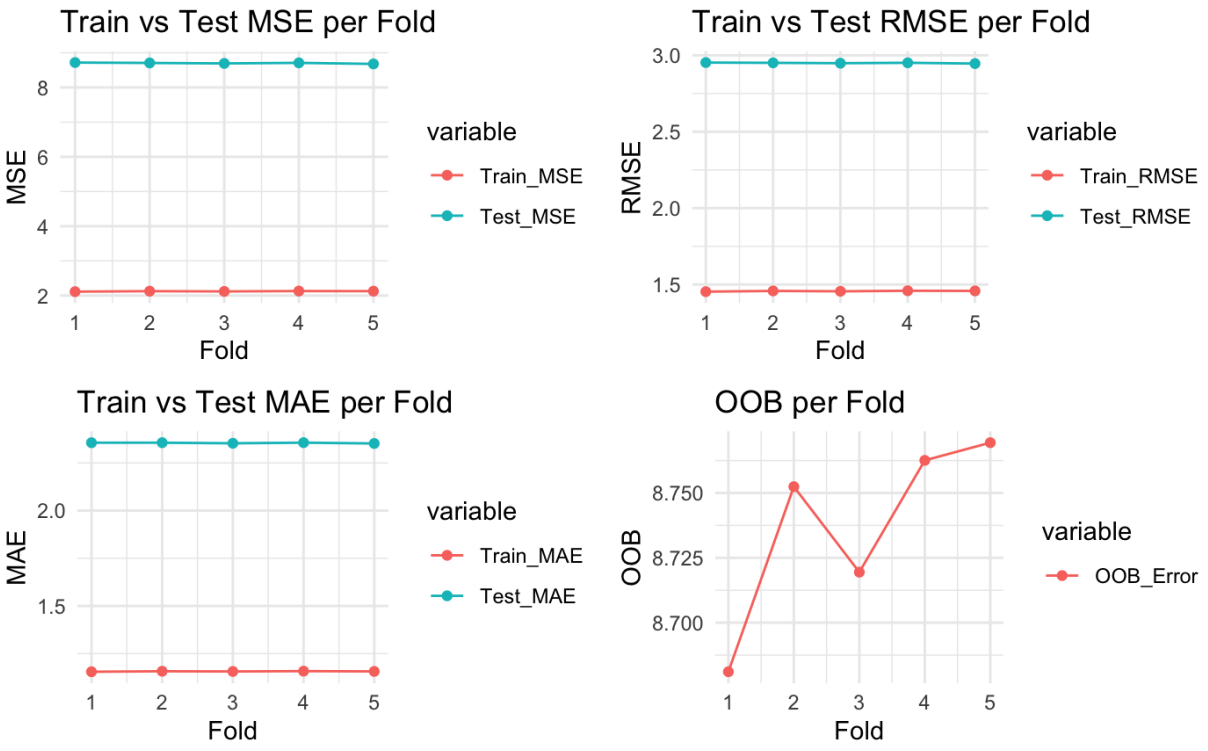

FIGURE 4.11: All metrics applying Random Forest with 5-Fold Cross Validation

XGBoost with k=5

Figure 4.12 illustrates the performance of the XGBoost model using 5-Fold Cross Validation. This shows MSE, RMSE, MAE and R^2 metrics for both training and testing sets across five folds. Consistently, the training data scores better than testing data, suggesting the model fits the training data well. The R^2 values, indicating the model's explanatory power, are relatively stable across the folds. Using 5-Fold Cross Validation provides a thorough assessment of the model's performance by ensuring that each data point gets to be in the test set once, minimizing the risk of overfitting and offering a more generalized performance metric.

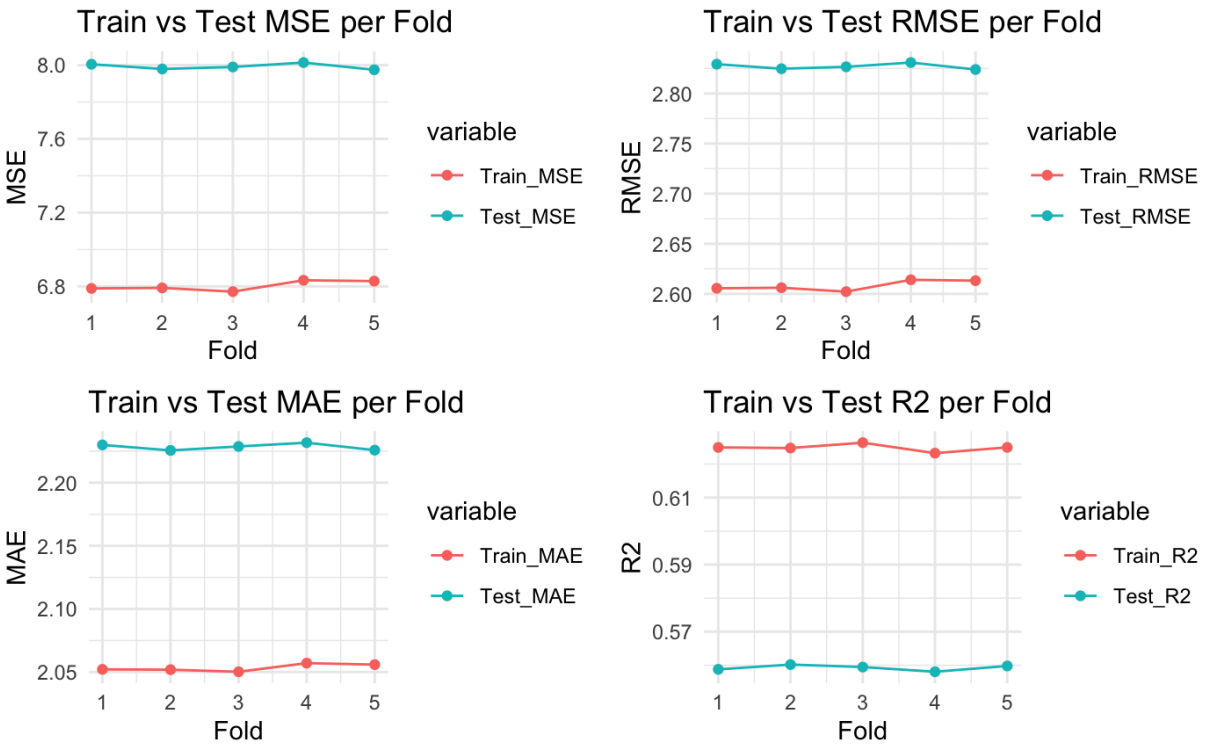

FIGURE 4.12: All metrics applying XGBoost with 5-Fold Cross Validation

4.6.2 10-Fold Cross Validation

Linear Regression with k=10

Figure 4.13 represents the Linear Regression model evaluated with 10-Fold Cross Validation. The performance of the model using 10-fold cross-validation shows consistent results across folds for MSE and RMSE, with slight variability. The MAE shows more fluctuation across different folds. R^2 values are stable, suggesting that the model has a consistent proportion of variance explained across the folds. Overall, the performance seems fairly reliable, but the variations in MAE could warrant a closer look to understand the model's consistency in terms of average error.

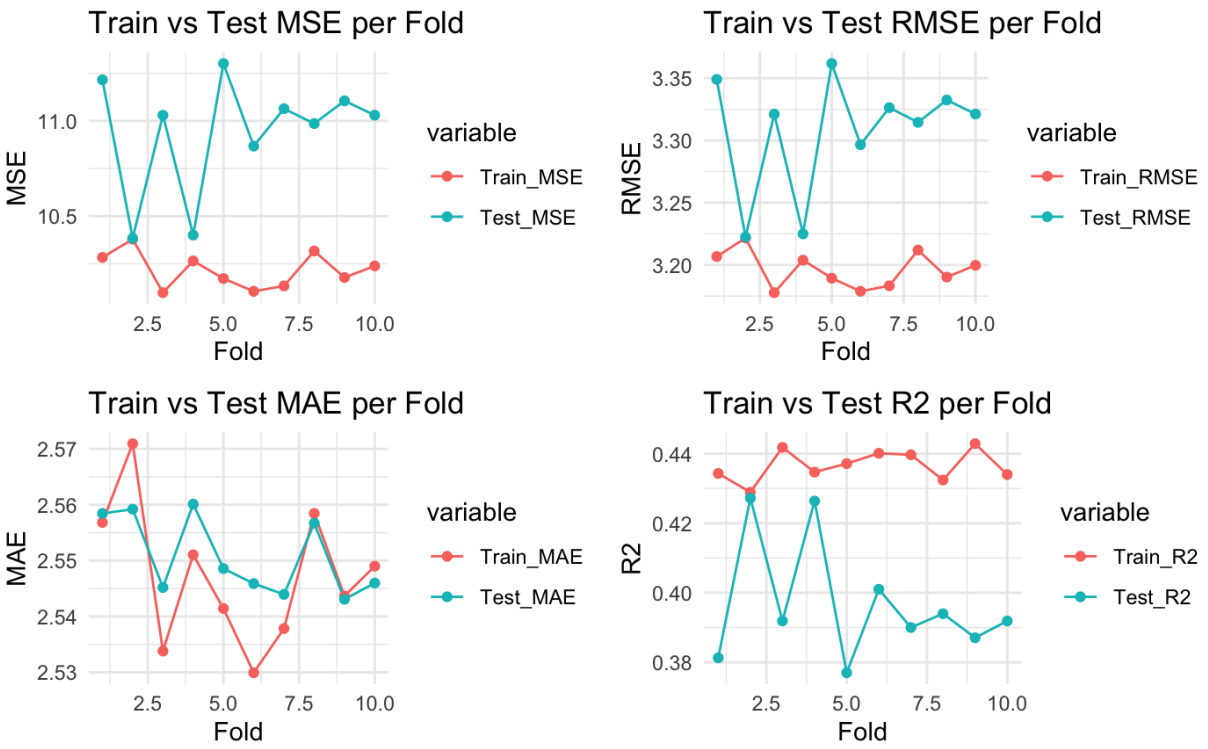

FIGURE 4.13: All metrics applying Linear Regression with 10-Fold Cross Validation

Random Forest with k=10

Attempting to use a Random Forest algorithm with 10-fold Cross-Validation on a personal computer can be computationally demanding due to the intrinsic nature of the method. Random Forest inherently involves building multiple decision trees, each requiring a subset of the data and features. When combined with 10-fold Cross-Validation, the computational load multiplies, as the algorithm needs to train and validate 10 separate times on different subsets of the data, increasing memory usage and processing time significantly. This could easily overwhelm the resources of a

standard PC. Due to these problems, it was not possible to run this model with 10-Fold Cross Validation.

XGBoosting with k=10

The Figure 4.14 illustrates the performance of XGBoost. It is evaluated with 10-fold cross-validation, shows very little variance between the training and testing phases for MSE, RMSE, and MAE, indicating a stable model with consistent performance across different data subsets. The R^2 metric also displays a stable performance, staying within a narrow range, which suggests the model's ability to explain the variance is uniform across the folds. Overall, the model's predictive accuracy and generalizability appear reliable based on these metrics.

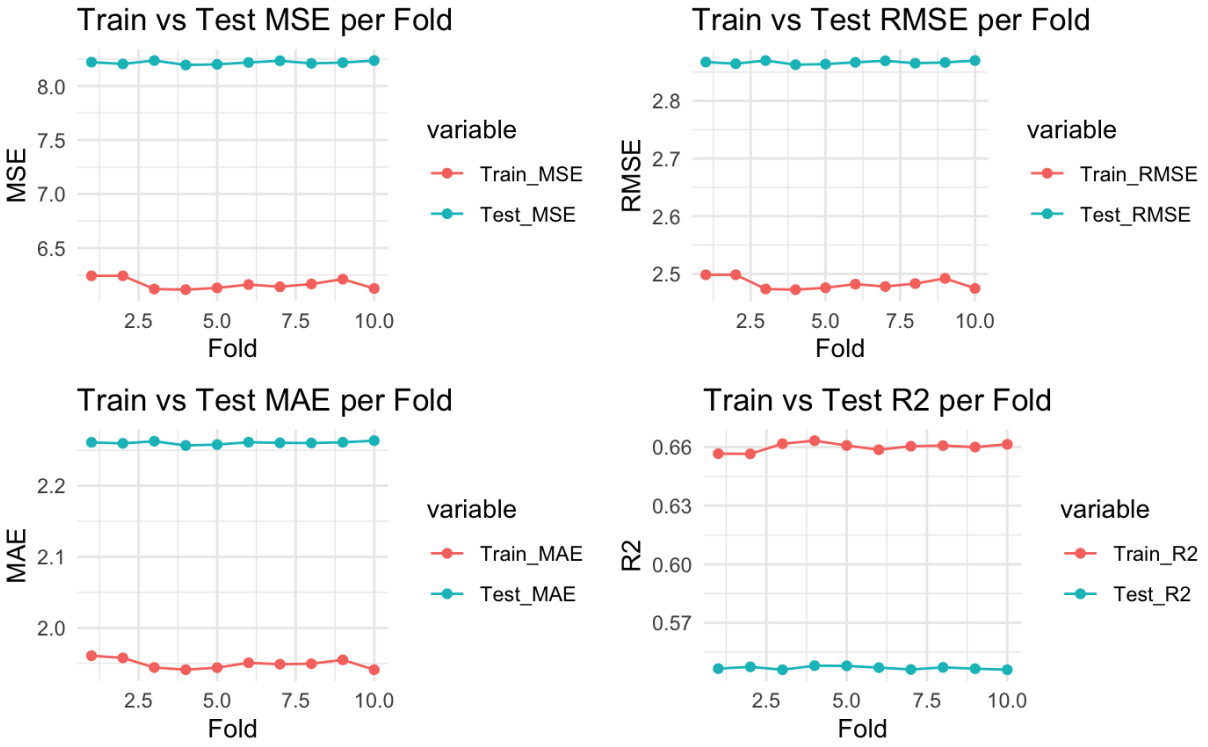

FIGURE 4.14: All metrics applying XGBoost with 10-Fold Cross Validation

4.7 Results

In comparing the models, the Random Forest and XGBoost algorithms, particularly with 10-fold cross-validation, exhibit robust performance for predicting interest rates. They achieve lower errors and higher R^2 values, indicating a strong predictive capacity and generalization. Conversely, Lasso and Ridge regressions, with their high MSE and negative R^2, are the least effective, likely due to underfitting the complexity of the data. Cross-validation results for both Linear Regression and ensemble methods highlight their efficacy, with ensemble methods outperforming simple Linear Regression, as evidenced by the significantly lower error metrics and higher R^2 values. The training and testing results are consistent, suggesting the models' stability. The use of cross-validation also demonstrates the importance of validating the model's ability to perform well on unseen data. Table 4.1 shows the results of the models on the training set and Table 4.2 shows those on the testing set. Both contain in bold the best model with its results.

Model	**MSE**	**RMSE**	**MAE**	$\boldsymbol{R^2}$
Linear Regression	10.33743	3.215187	2.560509	0.4299502
Lasso Regression	11.57067	3.401569	2.721616	-46.21615
Ridge Regression	14.52168	3.810732	3.053025	-79.07873
Linear Regression 5-Fold (Best fold k=2)	10.21160	3.195559	2.545874	0.4358379
Random Forest 5-Fold (Best fold k=1)	**2.112474**	**1.453435**	**1.154543**	**8.681077 (OOB Error)**
XGBoost 5-Fold (Best fold k=3)	6.771535	2.602217	2.050239	0.6263682
Linear Regression 10-Fold (Best fold k=3)	10.09809	3.177750	2.533809	0.4418171
XGBoost 10-Fold (Best fold k=4)	6.113943	2.472639	1.941072	0.6632966
Decision Trees	13.38487	3.658534	2.953433	0.2619011
Random Forest	2.182288	1.477257	1.17007	0.8796593
XGBoost	7.430606	2.725914	2.147623	0.5902447

Table 4.1: Training results for each model

Model	**MSE**	**RMSE**	**MAE**	$\boldsymbol{R^2}$
Linear Regression	10.76845	3.281531	2.563904	0.4069667
Lasso Regression	23.50268	4.847956	3.911493	-382.1256
Ridge Regression	20.06712	4.479634	3.612791	-441.0558
Linear Regression 5-Fold (Best fold k=1)	10.39869	3.224700	2.558819	0.4268264
Random Forest 5-Fold (Best fold k=5)	8.678974	2.946010	2.352115	8.769394 (OOB Error)
XGBoost 5-Fold (Best fold k=5)	7.974809	2.823970	2.225810	0.5597855
Linear Regression 10-Fold (Best fold k=2)	10.38403	3.222425	2.559209	0.4272398
XGBoost 10-Fold (Best fold k=4)	8.194665	2.862633	2.256622	0.5480447
Decision Trees	13.4105	3.662034	2.953717	0.2614654
Random Forest	8.346162	2.888972	2.300251	0.5403653
XGBoost	**7.697747**	**2.774481**	**2.185803**	**0.5760744**

TABLE 4.2: Testing results for each model

The best model according to the training table above, is the Random Forest with 5-Fold Cross Validation, specifically on the first fold. It shows the lowest MSE and MAE, along with the highest R^2 value, indicating it has the best fit among all models tested. It not only minimizes error but also maximizes the variance explained, which is particularly evident in the OOB error, a robust measure of prediction error in Random Forests. Although there are different best-folds on both tables, the Random Forest model with a significant discrepancy between the training MSE and the test MSE could be indicative of overfitting. The model appears to perform exceptionally well on training data, but does not generalise as effectively to test data (MSE: 8.681077).

The XGBoost model without K-Fold Cross-Validation shows the best performance on the testing set. It has the lowest RMSE, indicating a strong predictive accuracy, and the R^2 value, which means it explains the variance in the data very well compared to the other models. This suggests that the XGBoost model is likely the most effective for the interest rate prediction task.

Hyperparameter Tuning for XGBoost

Hyperparameter tuning is the process of finding the optimal combination of hyperparameters that define the architecture of a model. These hyperparameters are not learned from the data; instead, they are set prior to the training process and can significantly affect the performance of the model. Proper hyperparameter tuning can improve model accuracy by systematically searching for the best parameters, often using methods like grid search or random search.

In our case, since XGBoost had the best performance among all the other algorithms, we thought of doing hyper parameter tuning with a Grid Search on it (Listing 4.1). Unfortunately, the hyperparameter tuning process did not terminate even after a week, likely due to the extensive computational resources required for such an exhaustive search. This can happen when the search space is very large, the model is complex, or the computational power of the machine is limited. As a result, the process can become time-consuming and may not be feasible on a personal computer with limited processing capabilities.

```
# define the number of cores
numCores <- detectCores() - 1

# register doParallel as the backend for parallel execution
registerDoParallel(cores=numCores)

# define the control using a cross-validation approach
train_control <- trainControl(method = "cv", number = 5, verboseIter = TRUE)

# define the grid of hyperparameters to search over
xgb.grid <- expand.grid(
  nrounds = c(100, 200, 300),
  eta = c(0.01, 0.05, 0.1),
  max_depth = c(3, 6, 9),
  gamma = c(0, 0.1, 0.2),
  colsample_bytree = c(0.5, 0.8, 1),
  min_child_weight = c(1, 5, 10),
  subsample = c(0.5, 0.75, 1)
)

# train the model
xgb.tuned <- train(
  x = train_data, y = xgb.y_train,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = xgb.grid
)

# view the best tuning parameters
print(xgb.tuned$bestTune)

# stop the parallel backend
stopImplicitCluster()
```

Listing 4.1: Hyperparameter tuning on XGBoost

Chapter 5

Conclusion

Every algorithm in Data Science is good for a particular case. It can never be said that this algorithm is the best or this one is better than the other. Depending on the task and the data given, we can reach the conclusions of using a specific algorithm against the other. Nevertheless, sometimes for specific reasons of the type of data we contain or preprocessing steps, we achieve an unexpected output. There exist different reasons why we can never conclude without trying. Therefore, in this task of predicting the interest rate for the LC data, various algorithms have been tried. The visualization of these tries can be seen based on the metrics used to evaluate the performance of each of them. The mainly evaluation measures decided to be used in this particular task are MSE, RMSE, MAE, R^2. It has also been mentioned in the visualizations before that the testing phase needs the focus of this journey. Therefore, even though in two tables above, we can see the training and the testing measures, let us take a better look at the testing one. It is clear that the algorithm is going to give us a worse performance with unseen data than the training ones. Having this visualization in front, it is evident that XGBoost is giving the best evaluation metrics' outputs. As can be seen, it shows an MSE of 7.697747, RMSE of 2.774481 and MAE of 2.185803 on the test set. Hence it has been chosen as the best algorithm for this prediction task. As been said, trying and comparing algorithms is the solution to finding the best one. That is why we decided to try also the k-fold technique for partitioning the dataset into k-subsets or folds and evaluating it k times, each time using a different subset as the test set. Specifically, the XGBoost 5-Fold has shown the results given in the table above where the RMSE is nearly equal to the XGBoost without the k-fold technique. Nevertheless, it can be concluded that XGBoost has shown the best results compared to any other algorithm shown in the Table 4.2.

Chapter 6

Lessons Learnt

Reflecting on the course assignment centered around the exploration and application of data science techniques, it's clear that the journey from raw data to actionable insights is both intricate and nuanced. The assignment's focus on the LC dataset, which was be applied to the entire data science process, illustrating the fundamental importance of preprocessing and the careful application of various algorithms to unearth valuable insights.

Preprocessing, often regarded as the groundwork of data analysis, proved to be the base upon which predictive accuracy is built. The assignment emphasized that the majority of a data scientist's time is dedicated to this stage. Cleaning data, handling missing values, and feature selection were not just preliminary steps but pivotal actions that significantly influenced the outcomes of the models.

The assignment provided practical experience in applying various regression techniques, decision trees, and ensemble methods like Random Forest and XGBoost. The linear models, such as Linear Regression, Lasso, and Ridge, were foundational, offering a baseline for performance. However, they often fell short when grappling with the complexity of the data, highlighting their limitations in capturing non-linear relationships and interactions between features.

Decision trees provided a more intuitive approach to the modeling process, with the added benefit of interpretability. However, they also presented challenges, particularly in terms of their propensity to overfit the data.

The application into more advanced algorithms like Random Forest and XGBoost was particularly enlightening. These ensemble methods, which build upon the concept of decision trees, demonstrated superior performance, as evidenced by the lower RMSE and higher R^2 values. They illustrated the strengths of leveraging multiple models to improve prediction accuracy and handle overfitting more effectively.

XGBoost stood out, with its gradient boosting framework providing a fine balance between bias and variance, making it the most effective model for this particular task. This success highlighted the importance of hyperparameter tuning, even if it posed computational challenges. The experience underscored the need for computational power and the practical constraints one might encounter in real-world scenarios.

The assignment also brought us to use the critical role of Cross-Validation. By systematically dividing the data into k subsets and rotating the training and testing sets, it offered a robust method for assessing model performance. The contrast between 5-fold and 10-fold cross-validation results provided practical insights into how varying the number of folds could impact model evaluation, with higher k values offering a more granular assessment at the cost of increased computational demand.

In conclusion, the course assignment traversed the full spectrum of the data science workflow, from preprocessing to model evaluation, teaching us that successful data science projects are a blend of art and science. It reinforced the concept that while algorithms are vital, the ingenuity of the data scientist in preprocessing and choosing the right model is equally crucial.
The lessons learned from the assignment are manifold: the importance of preprocessing, the power and pitfalls of various algorithms, the delicate balance of bias and variance, and the practical considerations of computational resources. These experiences collectively enrich the understanding of the intricacies of ML and prepare one for the challenges of extracting knowledge from data in the real world.

Abbreviations

LC Lending Club. i, 1–3, 5–7, 14, 46, 47

MAE Mean Absolute Error. 32, 37–41, 44, 46

ML Machine Learning. 7, 16, 32, 48

MSE Mean Squared Error. 29, 32, 33, 35, 37–42, 44, 46

NLP Natural Language Processing. 14

OOB Out-of-Bag. 37, 38, 44

P2P Peer-to-Peer. 1

RMSE Root Mean Squared Error. i, 11, 29, 32, 37–41, 44, 46, 47

List of Figures

Listings

List of Tables