Loan Default Classification

How to handle imbalanced class data in R programming

Kieu Van Dang Email: dang.v@northeastern.edu

LinkedIn: https://www.linkedin.com/in/kirudang/ Phone: +1 647 782 4558

Data science enthusiast with a great passion for data pre-processing and prediction with a strong background in business. Relevant skills include machine learning, statistics, problem-solving, programming (including SQL, Python, and R), and critical & creative thinking.

INDEX

I. INTRODUCTION

1. Topic of interest

Lending loans is an important source of revenue for the banking and financial business. However, the industry is frequently associated with risks. Due to insufficient or non-existent credit history, lending firms, particularly smaller ones, have had difficulty selecting whether or not to approve loans (Sundaramahadevan, 2021). To begin with, a potentially good-paying borrower is refused, resulting in a financial loss to the loan company. Second, a high-risk applicant who is unlikely to repay the loan is granted, resulting in yet another possible loss for the company. Therefore, with the right decision-making tools including data mining and analytics, this could be mitigated.

2. Research questions and project objectives

The main objectives while authors propose this dataset are:

- **Master the data munging** through four components to facilitate machine learning works and model development including Exploration, Transformation, Enrichment (if any), and Validation.
- **Deploy various methods of data mining and analytics for classification problems** by applying knowledges and techniques acquired through course. These could be Logistics Regression, Support Vector Machines, Radom Forest, Gradient Boosting, and other advanced methods.
- **Handle unbalanced class of target variable**. One approach that authors think of is to compare the measure metrics across methods to select the most optimal technique.
- **Familiarize ourselves with the target industry**. It is no doubt that Banking and Finance industry is one of the most developed field in Canada; thus, initial knowledge and experience in this industry could help us with more potential opportunities to land a prominent job.

To fulfill those targets, we propose several research questions as follow: **Given current customer information, should the lender let the customer borrow?** And **How to reduce false negative rate?**

II. ANALYSIS

1. Data Exploration and Preparation

Data exploration is a crucial step to know about dataset purpose, its structure, variables and fields within the set, data types, outliers, missing values, incorrect observations, as well as relationships between data.

a. Dataset introduction

The dataset contains fundamental information about historical data of borrowing customers along with the approval status as target variable. The set comprises a diverse set of predictors, spanning across 34 variables through 148,670 observations, in multiple data types ranging from time data, categorical data to continuous and discrete data. This broad information would provide a solid foundation for further investigation and optimize the performance of classification problems.

Before delving into the dataset, here are the full definitions of the attributes listed in the table:

Variables	Description	Expected data type	Sample value
ID	The unique identification number	Integer or Character	24890
Year	The year in which loan was taken	Time	2019
loan_limit	Cash flow type	Categorical	**cf** – Cash flow **ncf** – Net cash flow
Gender	Gender of the borrowers	Categorical	Male, Female
approv_in_adv	Whether loan is approved prior	Categorical	**nopre**: No **pre**: Yes
loan_type	Type of loan	Categorical	type1, type2
loan_purpose	Purpose of loan	Categorical	p1, p2, p3, p4
Credit_Worthiness	Credit worthiness of the applicant	Categorical	l1, l2
open_credit	Type of Credit	Categorical	**opc** – Open credit **nopc** – Not open credit
business_or_commercial	Usage type of the loan amount	Categorical	**nob/c**: not for business or commercial

			b/c: business or commercial
loan_amount	Defines how much the loan amount is	Number	76000
rate_of_interest	Defines the interest rate	Number	3.2
Interest_rate_spread	Spread of interest rate by banks	Number	0.18
Upfront_charges	Initial charges before taking the loan	Number	1150.2
Term	Term of payment	Integer	360
Neg_ammortization	Negotiated amortization	Categorical	**not_neg**: No **neg_amm**: Yes
interest_only	interest-only payment method?	Categorical	**not_int**: No **int_only**: Yes
lump_sum_payment	Allow to pay back at once or not	Categorical	not_lpsm: No lpsm: Yes
property_value	Collateral value	Number	708000
construction_type	Collateral construction type	Categorical	sb, mh
occupancy_type	Occupancy of borrowers	Categorical	pr, ir
Security_by	Type of collateral	Categorical	home, land
Total_units	Unit of collaterals	Number	1U, 2U
Age	Age of applicant	Number	45-54
income	income of borrowers	Number	6720
Credit_type	Type of credit	Categorical	CIB, CRIF
Credit_Score	Credit score	Number	683
co.applicant_credit_type	Co-applicant of credit type	Categorical	CIB, CRIF
submission_of_application	Submission of application	Categorical	Not_inst: No installment To_inst: Installment
LTV	Loan to value ratio, the relative difference between the loan amount and the current market value of a home.	Number	92.2
Region	Region	Categorical	North, south
Security_Type	Security type	Categorical	Direct, indirect
Status	Target variable: Loan status	Binary	0: Non-default 1: Default

dtir1	Debt to income rate, the percentage of gross monthly income that is used to pay monthly debt and determines borrowing risk	Number	32.1

***Table 1**: Description of variables*

b. Dimension reduction – Feature selection

For dataset with too many variables, keeping them all for EDA as well as for modeling would result in heavy computation, or so called **the curse of dimension**. Furthermore, there are several variables that do not have explanatory power to the outcome, retaining them is not a common practice. In this session, authors try to eliminate insignificant variables by multiple methods including: skewness analysis, using domain knowledge, justifying the missing value proportion, and so on.

```
> # Dimension reduction
> describe(df)
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
ID	1	148670	99224.50	42917.48	99224.50	99224.50	55104.54	24890.00	173559.00	148669.00	0.00	-1.20	111.31
year	2	148670	2019.00	0.00	2019.00	2019.00	0.00	2019.00	2019.00	0.00	NaN	NaN	0.00
loan_limit*	3	148670	2.04	0.30	2.00	2.00	0.00	1.00	3.00	2.00	1.26	7.77	0.00
Gender*	4	148670	2.61	1.05	3.00	2.64	1.48	1.00	4.00	3.00	-0.11	-1.20	0.00
approv_in_adv*	5	148670	2.15	0.37	2.00	2.07	0.00	1.00	3.00	2.00	1.61	1.76	0.00
loan_type*	6	148670	1.34	0.65	1.00	1.17	0.00	1.00	3.00	2.00	1.70	1.46	0.00
loan_purpose*	7	148670	3.88	1.15	4.00	3.98	1.48	1.00	5.00	4.00	-0.70	-0.94	0.00
Credit_Worthiness*	8	148670	1.04	0.20	1.00	1.00	0.00	1.00	2.00	1.00	4.53	18.55	0.00
open_credit*	9	148670	1.00	0.06	1.00	1.00	0.00	1.00	2.00	1.00	16.26	262.39	0.00
business_or_commercial*	10	148670	1.86	0.35	2.00	1.95	0.00	1.00	2.00	1.00	-2.08	2.32	0.00
loan_amount	11	148670	331117.74	183909.31	296500.00	313493.00	177912.00	16500.00	3576500.00	3560000.00	1.67	9.13	476.97
rate_of_interest	12	112231	4.05	0.56	3.99	4.03	0.54	0.00	8.00	8.00	0.39	0.34	0.00
Interest_rate_spread	13	112031	0.44	0.51	0.39	0.42	0.51	-3.64	3.36	7.00	0.28	-0.18	0.00
Upfront_charges	14	109028	3225.00	3251.12	2596.45	2745.90	3126.30	0.00	60000.00	60000.00	1.75	6.37	9.85
term	15	148629	335.14	58.41	360.00	351.78	0.00	96.00	360.00	264.00	-2.17	3.17	0.15
Neg_ammortization*	16	148670	2.90	0.31	3.00	3.00	0.00	1.00	3.00	2.00	-2.69	5.59	0.00
interest_only*	17	148670	1.95	0.21	2.00	2.00	0.00	1.00	2.00	1.00	-4.24	15.96	0.00
lump_sum_payment*	18	148670	1.98	0.15	2.00	2.00	0.00	1.00	2.00	1.00	-6.40	38.96	0.00
property_value	19	133572	497893.47	359935.32	418000.00	447440.75	252042.00	8000.00	16508000.00	16500000.00	4.59	73.22	984.84
construction_type*	20	148670	2.00	0.01	2.00	2.00	0.00	1.00	2.00	1.00	-67.10	4500.09	0.00
occupancy_type*	21	148670	1.97	0.26	2.00	2.00	0.00	1.00	3.00	2.00	-1.22	10.94	0.00
Secured_by*	22	148670	1.00	0.01	1.00	1.00	0.00	1.00	2.00	1.00	67.10	4500.09	0.00
total_units*	23	148670	1.02	0.20	1.00	1.00	0.00	1.00	4.00	3.00	11.07	137.97	0.00
income	24	139520	6957.34	6496.59	5760.00	6128.95	3380.33	0.00	578580.00	578580.00	17.31	885.25	17.39
credit_type*	25	148670	2.33	1.19	2.00	2.29	1.48	1.00	4.00	3.00	0.31	-1.43	0.00
Credit_Score	26	148670	699.79	115.88	699.00	699.73	148.26	500.00	900.00	400.00	0.00	-1.20	0.30
co.applicant_credit_type*	27	148670	1.50	0.50	1.00	1.50	0.00	1.00	2.00	1.00	0.00	-2.00	0.00
age*	28	148670	5.83	1.45	6.00	5.88	1.48	1.00	8.00	7.00	-0.30	-0.54	0.00
submission_of_application*	29	148470	1.65	0.48	2.00	1.68	0.00	1.00	2.00	1.00	-0.61	-1.63	0.00
LTV	30	133572	72.75	39.97	75.14	74.00	18.55	0.97	7831.25	7830.28	120.61	19978.00	0.11
Region*	31	148670	2.81	1.06	2.00	2.84	0.00	1.00	4.00	3.00	0.09	-1.67	0.00
Security_Type*	32	148670	1.00	0.01	1.00	1.00	0.00	1.00	2.00	1.00	67.10	4500.09	0.00
Status	33	148670	0.25	0.43	0.00	0.18	0.00	0.00	1.00	1.00	1.18	-0.62	0.00
dtir1	34	124549	37.73	10.55	39.00	38.33	10.38	5.00	61.00	56.00	-0.55	0.38	0.03

***Figure 1.1**: Summary of statistic*

From the summary statistics in **Figure 1.1**, we have noticed four variables that may be unnecessary.

ID: The skewness is equal to 0; that means its distribution is uniform and all values are unique without any duplication, which indicates that all borrowers in the set only apply the loan for once. To validate this assumption, we check the duplication; the result of all FALSEs in **Figure 1.2** supports our hypothesis and hence we can drop this variable.

```
> # ID investigation
> table(duplicated(df$ID))

 FALSE
148670
> df$ID <- NULL # Drop ID column
```

***Figure 1.2**: Validation for ID*

Year: Min and max are identical, equal 2019, with range of 0. This signifies that there is only one **year of 2019** in our dataset, so we could drop this column.

Construction type: This variable contains only two classes with a highly skewed distribution. Therefore, we investigate further the significance and contribution of minority class to the set.

```
> # Investigate construction type
> prop.table(table(df$construction_type))

          mh           sb
0.0002219681 0.9997780319
> df$construction_type <- NULL # Drop constuction type column
```

***Figure 1.3**: Validation for construction type and drop column*

As seen in **Figure 1.3**, 99.97% observations of construction type are into **sb** group. This signifies that the column does not have much explanatory power to the outcome as almost 100% classes are sb. As a result, we drop this column as well.

Secured_by and **Security_type**: The same insight is applied for two variables about collateral security. Consequently, we drop these two as well.

```
> # Investigate Security
> prop.table(table(df$Secured_by))

        home         land
0.9997780319 0.0002219681
> df$Secured_by <- NULL # Drop the column
```

```
> # Investigate Security
> prop.table(table(df$Security_Type))

      direct     Indriect
0.9997780319 0.0002219681
> df$Security_Type <- NULL # Drop the column
```

***Figure 1.4:** Drop unnecessary columns*

Domain Knowledge Method: Among 29 variables left, which are still quite many, authors decide to use domain knowledge to conduct feature selection and trim down the data.

Normally, to define if a client could repay the loan or not, it is advised to consider these following important factors: Demographic, financial power, loan amount, financial score (historical data) and other factors. Consequently, authors map these factors with current variables of data set to point out effective variables for our model as table below.

No	Group factor	Description	Potential variables to keep
1	**Demographic**	Gender, age, living location, education level.	Gender, age, Region
2	**Financial power**	Mostly about income of clients	property_value, occupancy_type, total_units, income
3	**Loan amount**	Total loan amount to see if clients are able to repay	loan_limit, loan_type, loan_purpose, open_credit, loan_amount, business_or_commercial, LTV, interest_only
4	**Financial score and historical data**	The credit score for clients from financial agencies and historical data.	Credit_Worthiness, Credit_Score, approv_in_adv, dtir1
5	**Other**	Other information	rate_of_interest, Interest_rate_spread

***Table 2**: Domain knowledge for feature selection*

From insignificant variables, which are not listed on the above table, we investigate each of them before removing:

Upfront_charges: This is processing fee for the loan as an admin procedure, we consider this non-significant variable and drop it. Another reason to drop the column is the variable contains too many missing values, which may violate the assumption: an important factor in evaluating the risk will not be missing dramatically.

Neg_ammortization: Amortization is just a method of paying off an amount owed over time by making planned, incremental payments of principal and interest. A loan with or without this method is considered optional and hence could be neglectable. Authors remove the column to reduce data dimension.

c. Duplication

There are two types of duplication that may exist in our dataset: Partly duplicated data and fully duplicated data. However, within the context of this report, the author only investigates the fully duplicated observations.

```
> # Duplication
> summary(duplicated(df))
   Mode    FALSE
logical  148670
```

***Figure 1.5**: Duplication and clean data*

Referring to **Figure 1.5**, there are no duplicated observations in the data set, so we keep the current data frame and move forward to data transformation.

d. Data transformation

In order to convert data in the correct format, we first take a look at data structure as below.

```
> str(df)
'data.frame':   148670 obs. of  29 variables:
 $ loan_limit               : chr  "cf" "cf" "cf" "cf" ...
 $ Gender                   : chr  "Sex Not Available" "Male" "Male" "Male" ...
 $ approv_in_adv            : chr  "nopre" "nopre" "pre" "nopre" ...
 $ loan_type                : chr  "type1" "type2" "type1" "type1" ...
 $ loan_purpose             : chr  "p1" "p1" "p1" "p4" ...
 $ Credit_Worthiness        : chr  "l1" "l1" "l1" "l1" ...
 $ open_credit              : chr  "nopc" "nopc" "nopc" "nopc" ...
 $ business_or_commercial   : chr  "nob/c" "b/c" "nob/c" "nob/c" ...
 $ loan_amount              : int  116500 206500 406500 456500 696500 706500 3465
 $ rate_of_interest         : num  NA NA 4.56 4.25 4 ...
 $ Interest_rate_spread     : num  NA NA 0.2 0.681 0.304 ...
 $ Upfront_charges          : num  NA NA 595 NA 0 ...
 $ term                     : num  360 360 360 360 360 360 360 360 360 360 ...
 $ Neg_ammortization        : chr  "not_neg" "not_neg" "neg_amm" "not_neg" ...
 $ interest_only            : chr  "not_int" "not_int" "not_int" "not_int" ...
 $ lump_sum_payment         : chr  "not_lpsm" "lpsm" "not_lpsm" "not_lpsm" ...
 $ property_value           : num  118000 NA 508000 658000 758000 ...
 $ occupancy_type           : chr  "pr" "pr" "pr" "pr" ...
 $ total_units              : chr  "1U" "1U" "1U" "1U" ...
 $ income                   : num  1740 4980 9480 11880 10440 ...
 $ credit_type              : chr  "EXP" "EQUI" "EXP" "EXP" ...
 $ Credit_Score             : int  758 552 834 587 602 864 860 863 580 788 ...
 $ co.applicant_credit_type : chr  "CIB" "EXP" "CIB" "CIB" ...
 $ age                      : chr  "25-34" "55-64" "35-44" "45-54" ...
 $ submission_of_application: chr  "to_inst" "to_inst" "to_inst" "not_inst" ...
 $ LTV                      : num  98.7 NA 80 69.4 91.9 ...
 $ Region                   : chr  "south" "North" "south" "North" ...
 $ Status                   : int  1 1 0 0 0 0 0 0 0 0 ...
 $ dtir1                    : num  45 NA 46 42 39 40 44 42 44 30 ...
```

***Figure 1.6**: Data structure and data types*

Red highlighted variables are in undesirable data types. As they are categorical data, they should be converted to factor. While a **Blue highlighted variable** needs to be in number type, not in character type. Therefore, authors transform all of them with the following code.

```
# Factor some features
ranfac <- c("loan_limit","Gender","approv_in_adv","loan_type","loan_purpose","Credit_Worthiness","open_credit",
            "business_or_commercial","interest_only","lump_sum_payment","occupancy_type",
            "credit_type","co.applicant_credit_type","submission_of_application","Region","total_units")
df[ranfac] <- lapply(df[ranfac], factor)
```

***Figure 1.7**: Factor several variables*

For **total_units**, since we expect its type should be number, but currently it is in character. Thus, we use substring to extract the number and convert to numerical data as below.

```
> summary(df$total_units)
    1U      2U      3U      4U
146480    1477     393     320
> df$total_units<-as.numeric(substr(df$total_units,1,1))
```

***Figure 1.8**: Transform total_units variable*

e. Missing value

In the previous part, we noticed that our data may contain missing values. To have a double check, we first make a map to locate the available missing values by deploying the *missmap()* function from ***Amelia*** package, and then the authors also check by computing total missing values, using *sum(is.na(x))* of *sapply*.

```
> sapply(df,function(x) sum(is.na(x)))
               loan_limit                    Gender              approv_in_adv                 loan_type
                        0                         0                         0                         0
            loan_purpose         Credit_Worthiness              open_credit business_or_commercial
                        0                         0                         0                         0
             loan_amount          rate_of_interest     Interest_rate_spread                      term
                        0                     36439                     36639                        41
           interest_only         lump_sum_payment             property_value               occupancy_type
                        0                         0                     15098                         0
            total_units                    income               credit_type              Credit_Score
                        0                      9150                         0                         0
co.applicant_credit_type                       age submission_of_application                      LTV
                        0                         0                       200                     15098
                 Region                    Status                      dtir1
                        0                         0                     24121
```

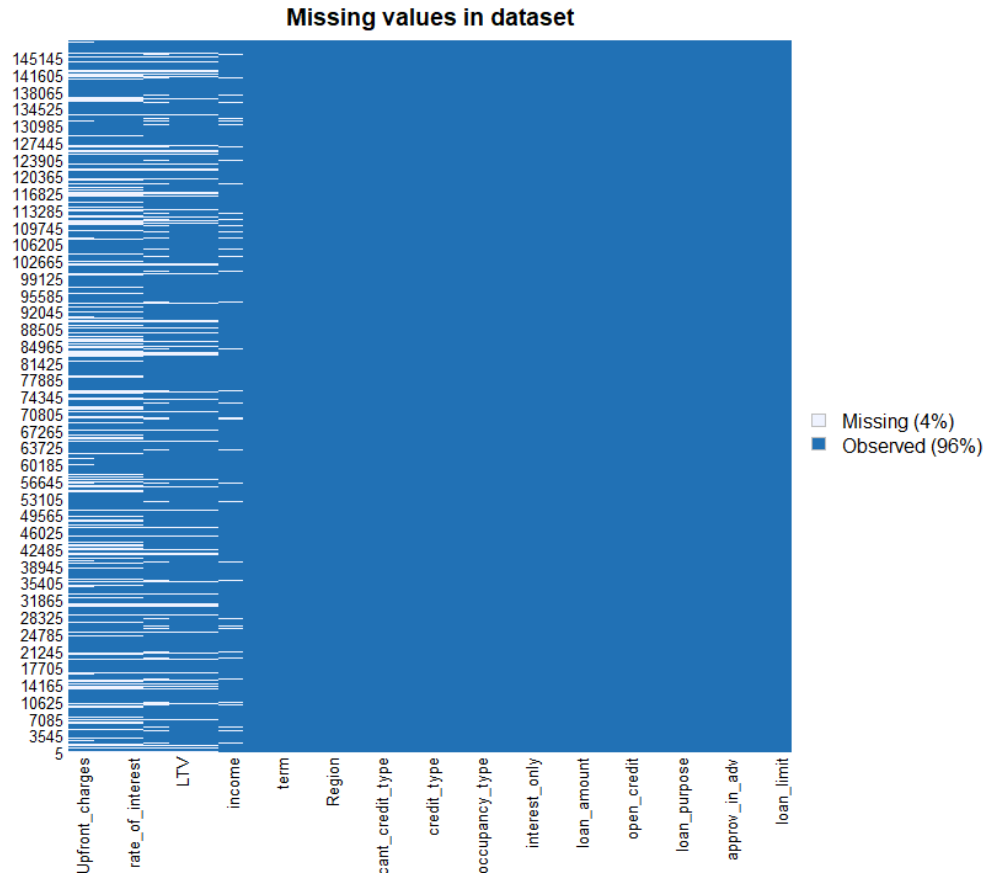


***Figure 1.9**: Missing value details and map*

There are 8 variables with missing values ranging from 41 to 39642 observations. Please be mindful that the default setting of model in R will automatically remove entire rows with any missing values in such observations; in other words, we actually can run the model without missing data imputation. However, doing so will scale down our dataset dimension and somewhat create bias. Therefore, in this part, authors are going to approach each variable to handle its missing data for modeling.

Missing data pattern assumption:

Since the number of variables with missing data are quite significant (4% in total), we should be mindful of our missing pattern, whether it is Missing Completely at Random (MCAR), Missing at Random (MAR) or Missing not at Random (MNAR). To validate this pattern, data scientist could use Little's (1988) test statistic in the R-package BaylorPsychEd with the null hypothesis is that data is not missing

completely at random to assess missing pattern. However, this method has elicited controversy in the field, hence it should be careful to employ.

As shown in **Figure 1.10** below, p value is 0, we can reject the null hypothesis, revealing that our missing data are in either MCAR and MAR.

```
> NA_result
# A tibble: 1 x 4
  statistic    df p.value missing.patterns
      <dbl> <dbl>   <dbl>            <int>
1   164044.    76       0               10
```

***Figure 1.10**: Missing pattern testing*

Being said that, authors further deploy *md.pattern()* from ***Mice*** package to check the pattern of missing data visually. The pattern is as shown in **Figure 1.11**.

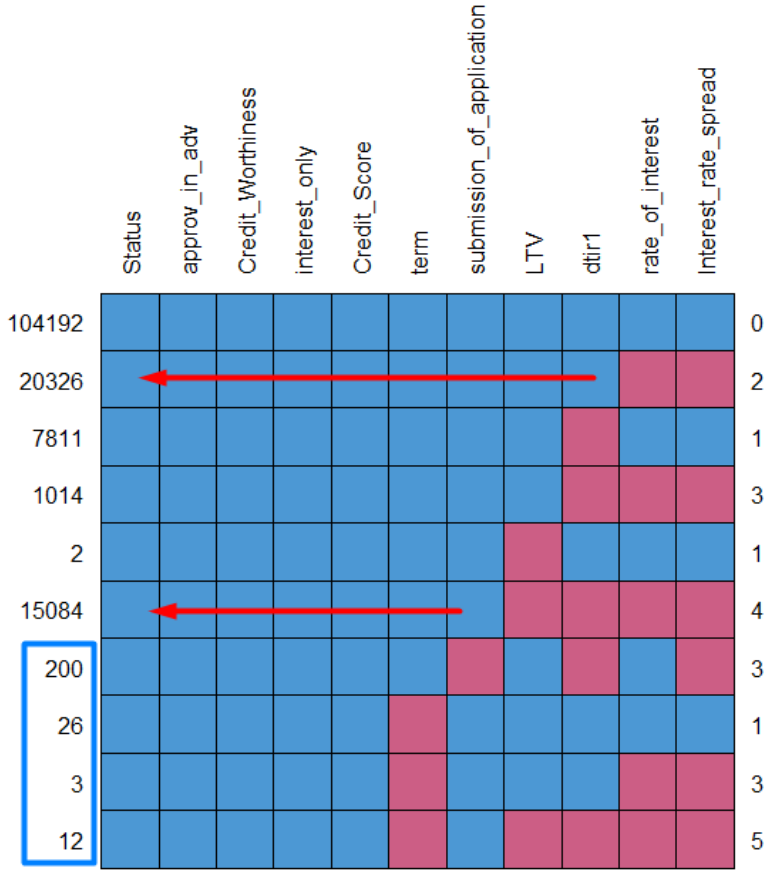


***Figure 1.11**: Missing pattern plot*

For the plot above, left figures are the number of missing observations, right figures illustrate the number of variables contain missing values and the top display names of the associated variables.

Among all patterns, it is noticeable that 20,326 observations from 2 variables and 15,084 observations from 4 variables are missing together. These two patterns comprise the majority of missing data, therefore we cannot delete but have to impute them.

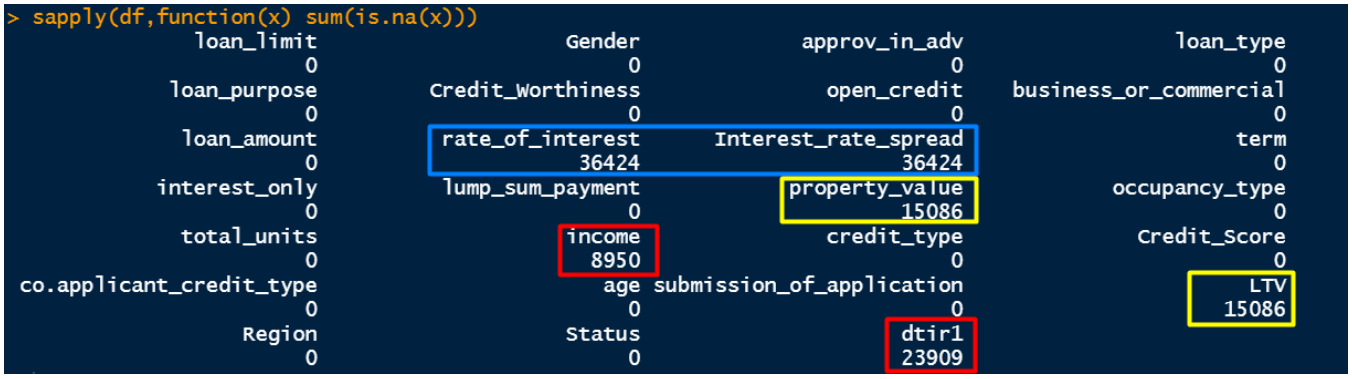


***Figure 1.12**: Missing together pattern*

If we take a look at details of missing map details again, we may notice that there are three pairs of variables that are mutually affect each other, or in other words they are highly correlated (multicollinearity) and will missing together.

rate_of_interest and **Interest_rate_spread** are a pair.

property_value and **LTV**, which is the relative difference between the loan amount and the current market value of a home (the property_value) are another pair.

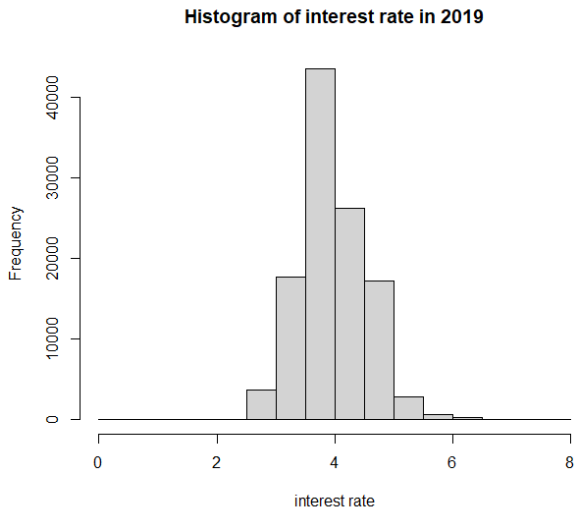
Income and **dtir1**, which is ratio is the percentage of gross monthly income that is used to pay monthly debt and determines borrowing risk, are another pair.

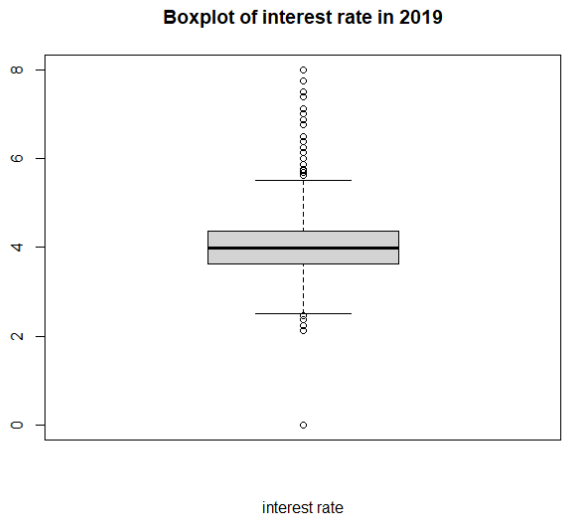
This means we do not need to impute all these 6 variables (3 pairs) because it will cause severe multicollinearity issues when we run models but just fill out 3 variables and drop 3 remaining ones of each pair.

List-wise (Complete-case analysis — CCA) deletion: term and submission_of_application

The last 4 rows with total 241 missing values in the plot are from **term** (41 missing rows) and **submission_of_application** (200 rows), which account only marginal proportion. In addition, since we have large enough dataset, the listwise deletion could be the most reasonable strategy where the assumption of MCAR is satisfied. Applied CCA, authors drop these observations of missing data.

Median imputation: Interest rate.






```
> describe(df$rate_of_interest)
   vars      n mean   sd median trimmed  mad min max range skew kurtosis se
X1    1 112005 4.04 0.56   3.99    4.03 0.54   0   8     8 0.39     0.35  0
```

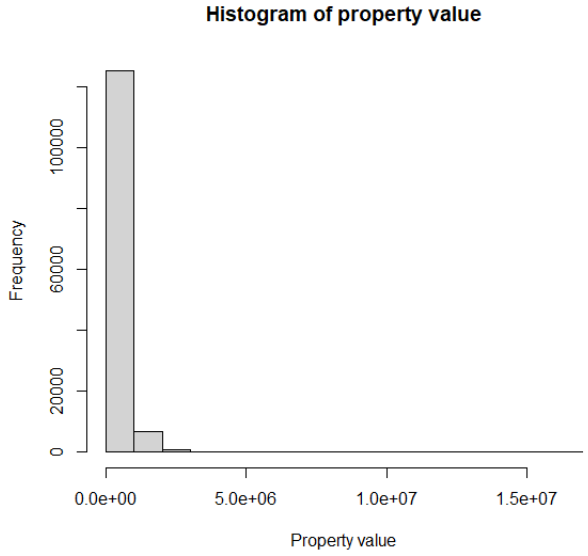
```
> quantile(df$rate_of_interest, na.rm=TRUE)
   0%   25%   50%   75%  100%
0.000 3.625 3.990 4.375 8.000
```

***Figure 1.13**: Mixed plots (histogram, boxplot) and statistics for Interest rate*

For **interest_rate**, it is a fact that within a year, the rate does not vary much. The descriptive analysis gives us the normal distribution of the interest rate with very low skewness of 0.39. In addition, the mean is quite equivalent to the median, around 4% and when taking account into quantile analysis, we could see that the first to the third quantile, the value is ranging from 3.625% to 4.375%. Combining with two plots, we could confidently choose the median for filling out missing observations.

Subset analysis and imputation: property_value

Since the property value distributes in highly positive skewed shape as shown in **Figure 1.14**, the mean and the median will differ substantially, hence the imputation may show high variance. We attempt another method of replacement, subset analysis and then replacement.




```
> describe(df$property_value)
   vars      n     mean       sd median  trimmed    mad  min      max    range skew kurtosis    se
X1    1 133343 498276.3 360013.8 418000 447807.2 252042 8000 16508000 16500000 4.59    73.26 985.9
```

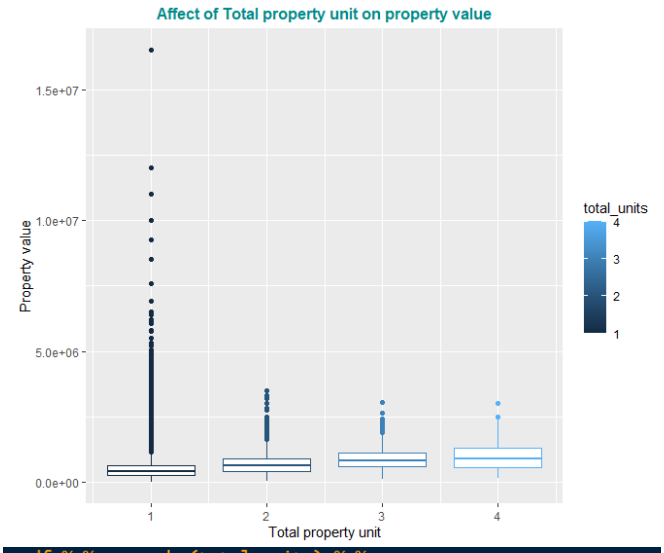
***Figure 1.14**: Histogram and statistics for Property value*

In order to conduct subset analysis, we first find out which variable could explain the missing values of property_value well. From the rest variables, we may find that two variables possessing the most explanatory power for property_value are **Region** (where borrowers live) and **total_units** (the number of units of property). Therefore, we will leverage of these variables for imputation.

Creating boxplots by each level of associated variable and statistical summary in **Figures 1.15** and **1.16** below, we could observe the higher impact of unit to property value than the impact of region to property value. With this finding, we subset property value by each number of unit and replace missing values by group's median, as displayed in **Figure 1.17**.


```
> df %>% group_by(Region) %>%
+   summarise(Mean_value=mean(property_value,na.rm=TRUE),
+             Median_value=median(property_value,na.rm=TRUE))
# A tibble: 4 x 3
  Region     Mean_value Median_value
  <fct>           <dbl>        <dbl>
1 central       452565.       388000
2 North         492762.       418000
3 North-East    444544.       388000
4 south         512140.       428000
```

***Figure 1.15**: Property value by region*




```
> df %>% group_by(total_units) %>%
+   summarise(Mean_value=mean(property_value,na.rm=TRUE),
+             Median_value=median(property_value,na.rm=TRUE))
# A tibble: 4 x 3
  total_units Mean_value Median_value
        <dbl>      <dbl>        <dbl>
1           1    493790.       418000
2           2    741816.       638000
3           3    899170.       813000
4           4    970491.       878000
```

***Figure 1.16**: Property value by unit*

```r
# replace by median
df <- df %>% group_by(total_units) %>%
             mutate(property_value = ifelse(is.na(property_value),
                                median(property_value, na.rm = TRUE),
                                property_value))
df$LTV <- NULL # Drop the column
```

***Figure 1.17**: Subset property value by unit and replace by group median.*

Multiple Imputation - MICE: Income

The same observation from property value can be seen in income variable while data are highly skewed. In this case, authors come up with MICE for imputation. The key concept of Multiple Imputation (MI) is to use the distribution of the observed data to estimate a set of plausible values for the missing data. To elaborate, we could simulate and replicate the missing data based on current distribution of observed data as seen in **Figure 1.18**.

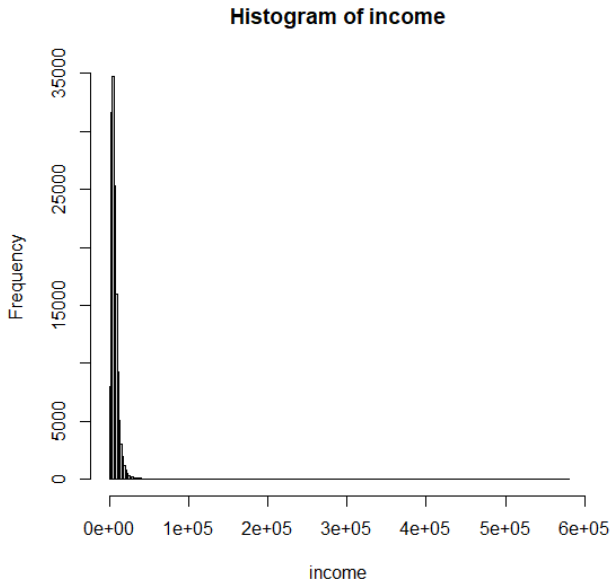


***Figure 1.18**: Distribution of Income*

Within MICE, there are many methods used to predict the missing values. In our case, authors choose Predictive Mean Matching (PMM), which is a semi-parametric imputation approach. The PMM method ensures that imputed values are plausible; it might be more appropriate than the regression method (which assumes a joint multivariate normal distribution) if the normality assumption is violated (Horton and Lipsitz 2001, p. 246).

Coding and Execution:

First, authors choose the most significant variables for explaining the target variable (income). Second, mice with "pmm" is run to create a temporary dataset. Third, authors check the distributions of observed data versus imputed data, as illustrated in **Figure 1.19**. Next, authors fill missing values with predicted values by *complete()* function as well as replace these complete columns in original dataset. Finally, we have a complete column for income.

```
# Use MICE to impute data
to_income <- c("income","Gender","business_or_commercial","loan_amount","property_value",
               "occupancy_type","Credit_Score") # Choose most important variables to income
# Running MIC
tempData <- mice(df2[to_income],m=5,method = "pmm",seed=500) # impute for missing data
#Summary and visualization
summary(tempData)
densityplot(tempData)
# Fill data in
completedData <- complete(tempData,1)
# Replace in the original data frame
df$income <- completedData$income


df$dtir1 <- NULL # Drop the column
```

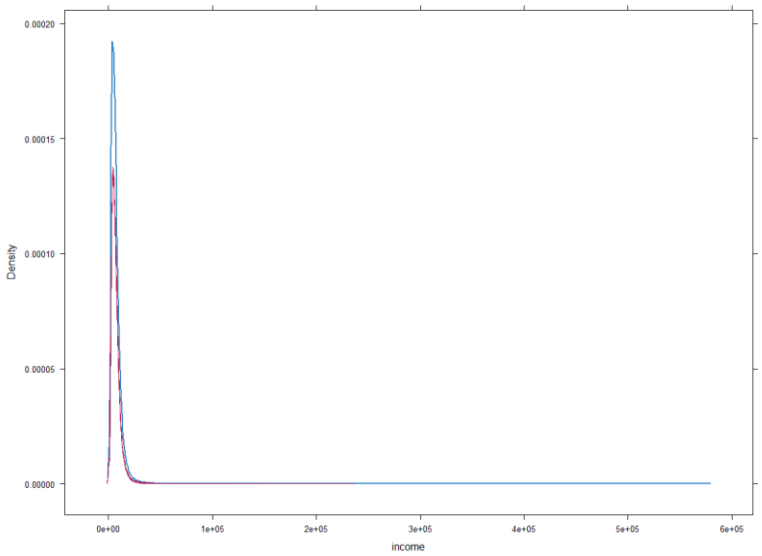


***Figure 1.19**: Distribution of observed values (Blue) and simulated values (Pink)*

Up to this step, we basically fulfill all missing data for the set.

f. Outlier detection

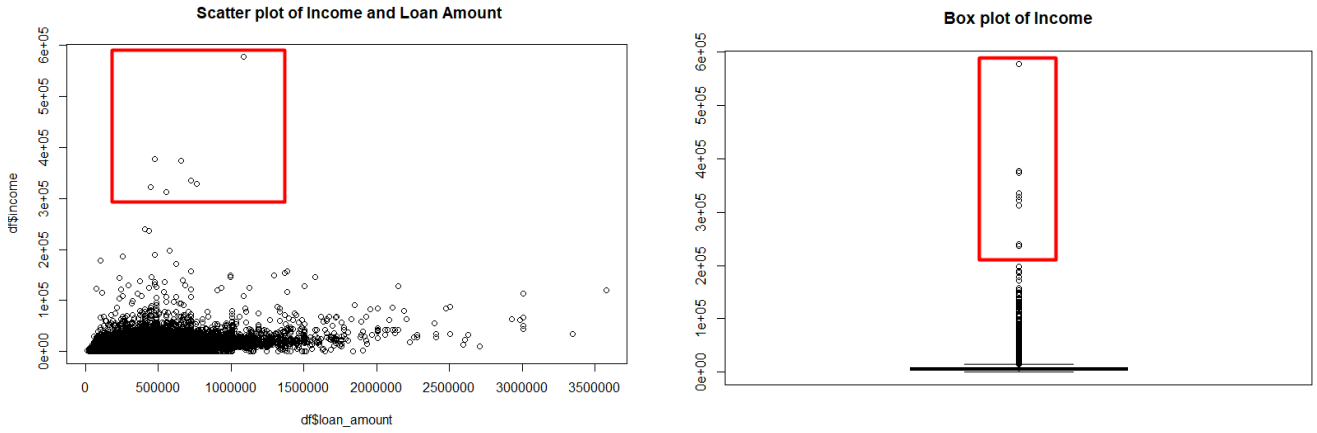
In order to validate the outliers, there are many approaches such as visualizing by boxplot, histogram, or quantile analysis, summary statistic, and so on. One effective method is to look at their descriptive summary, specifically the combination of range and skew.

To elaborate, if both range and skew of variable are high, this is the sign of outlier existence. Referring to **Figure 1.20** below, outliers are expected to exist in **property_value** and **income**.

```
> describe(df)
                          vars      n      mean        sd    median   trimmed       mad   min      max    range  skew kurtosis     se
loan_limit*                  1 148429      2.04      0.29      2.00      2.00      0.00     1        3        2  1.26     7.92   0.00
Gender*                      2 148429      2.61      1.05      3.00      2.63      1.48     1        4        3 -0.10    -1.20   0.00
approv_in_adv*               3 148429      2.15      0.37      2.00      2.07      0.00     1        3        2  1.61     1.75   0.00
loan_type*                   4 148429      1.34      0.65      1.00      1.17      0.00     1        3        2  1.70     1.46   0.00
loan_purpose*                5 148429      3.88      1.15      4.00      3.98      1.48     1        5        4 -0.70    -0.94   0.00
Credit_Worthiness*           6 148429      1.04      0.20      1.00      1.00      0.00     1        2        1  4.53    18.51   0.00
open_credit*                 7 148429      1.00      0.06      1.00      1.00      0.00     1        2        1 16.25   261.96   0.00
business_or_commercial*      8 148429      1.86      0.35      2.00      1.95      0.00     1        2        1 -2.09     2.37   0.00
loan_amount                  9 148429 331334.84 183901.99 296500.00 313721.95 177912.00 16500  3576500  3560000  1.67     9.14 477.34
rate_of_interest            10 148429      4.03      0.49      3.99      4.01      0.36     0        8        8  0.53     1.47   0.00
term                        11 148429    335.14     58.43    360.00    351.80      0.00    96      360      264 -2.17     3.17   0.15
interest_only*              12 148429      1.95      0.21      2.00      2.00      0.00     1        2        1 -4.23    15.93   0.00
lump_sum_payment*           13 148429      1.98      0.15      2.00      2.00      0.00     1        2        1 -6.39    38.88   0.00
property_value              14 148429 490603.06 342212.55 418000.00 442771.91 222390.00  8000 16508000 16500000  4.87    81.35 888.25
occupancy_type*             15 148429      1.97      0.26      2.00      2.00      0.00     1        3        2 -1.22    10.92   0.00
total_units                 16 148429      1.02      0.20      1.00      1.00      0.00     1        4        3 11.08   138.18   0.00
income                      17 148429   6948.84   6417.07   5760.00   6129.35   3380.33     0   578580   578580 17.05   876.45  16.66
credit_type*                18 148429      2.33      1.19      2.00      2.29      1.48     1        4        3  0.31    -1.43   0.00
Credit_Score                19 148429    699.79    115.88    699.00    699.73    148.26   500      900      400  0.00    -1.20   0.30
co.applicant_credit_type*   20 148429      1.50      0.50      1.00      1.50      0.00     1        2        1  0.00    -2.00   0.00
age*                        21 148429      4.84      1.44      5.00      4.88      1.48     1        7        6 -0.27    -0.63   0.00
submission_of_application*  22 148429      1.65      0.48      2.00      1.68      0.00     1        2        1 -0.61    -1.63   0.00
Region*                     23 148429      2.81      1.06      2.00      2.83      0.00     1        4        3  0.09    -1.67   0.00
Status                      24 148429      0.25      0.43      0.00      0.18      0.00     0        1        1  1.18    -0.60   0.00
```

***Figure 1.20**: Descriptive summary for data*

Investigate Income:




```
> quantile(df$income)
   0%   25%   50%   75%  100%
    0  3720  5760  8520 578580
> quantile(df$income,0.99)
  99%
26520
```

***Figure 1.21**: Mix figures to investigate outliers of Income*

Scatter plot of Income and Loan amount as well as the boxplot for this variable suggest a quite many outliers in the column. In the second step, 99% quantile is at 26,520 only, while the maximum is 578,580, which is 21 time higher, a huge gap for observations.

Therefore, we will remove outliers by 99% quantile for this variable.

Investigate Property: The same procedure is applied for this variable. However, the outliers are fewer as we can see from the plot. Therefore, authors choose quantile of 99.5% to cut off and remove highly unusual observations.

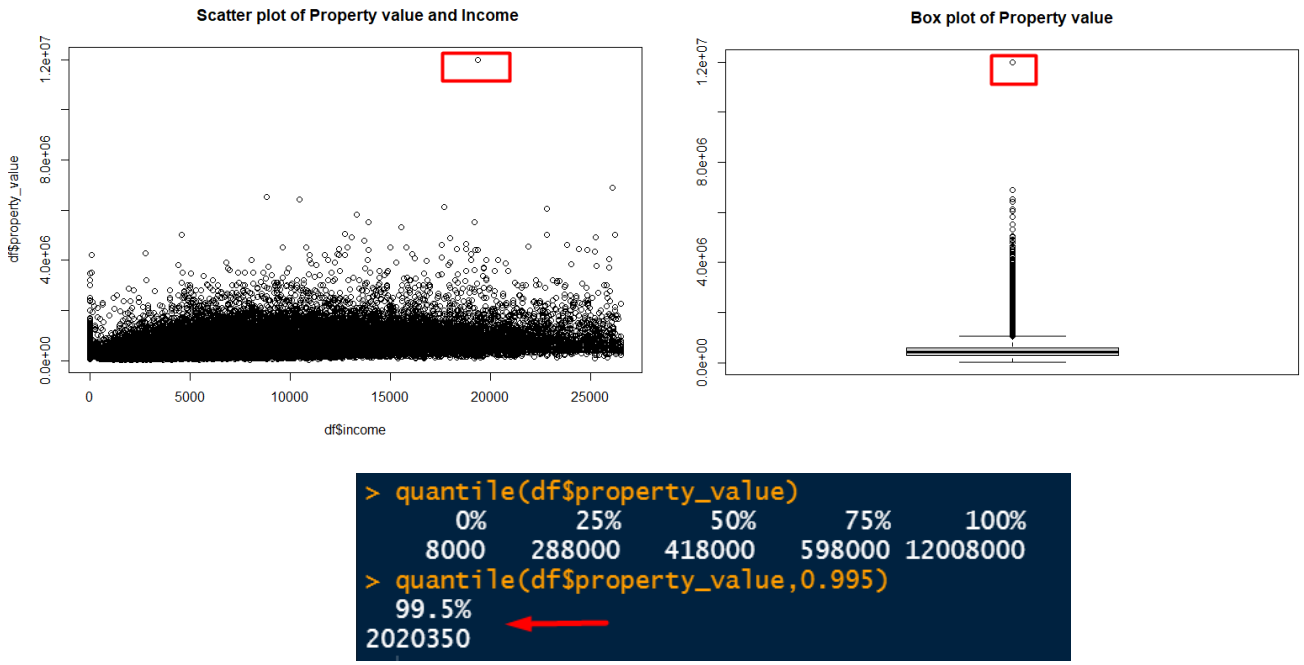


***Figure 1.22**: Mix figures to investigate outliers of Property value.*

The code to execute for removing outliers is displayed in Appendix

Final check for outliers: Up to this phase, authors use *str()* to scrutinize data once more. Unfortunately, there are three variables containing space in rows so that the computer classifies them as a blank value but not a missing data.

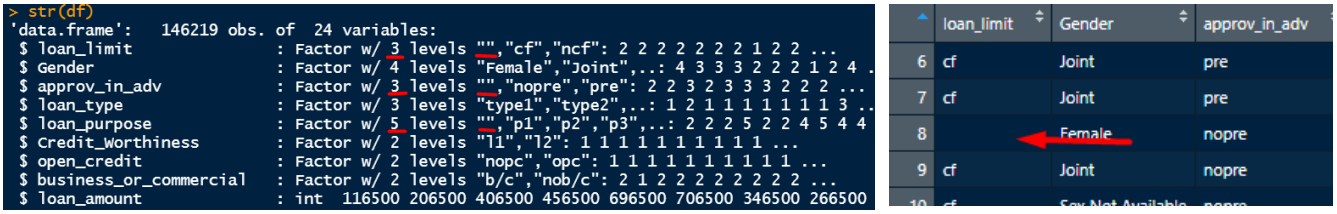


***Figure 1.23**: Structure double check*

Since all three variables are categorical, authors decide to keep missing data without imputation or deletion, assigning them by another bin called “Missing” to check the affect of them to the overall performance of classification problems later.

Final data are clean as below with 24 variables left with total 137336 observations.

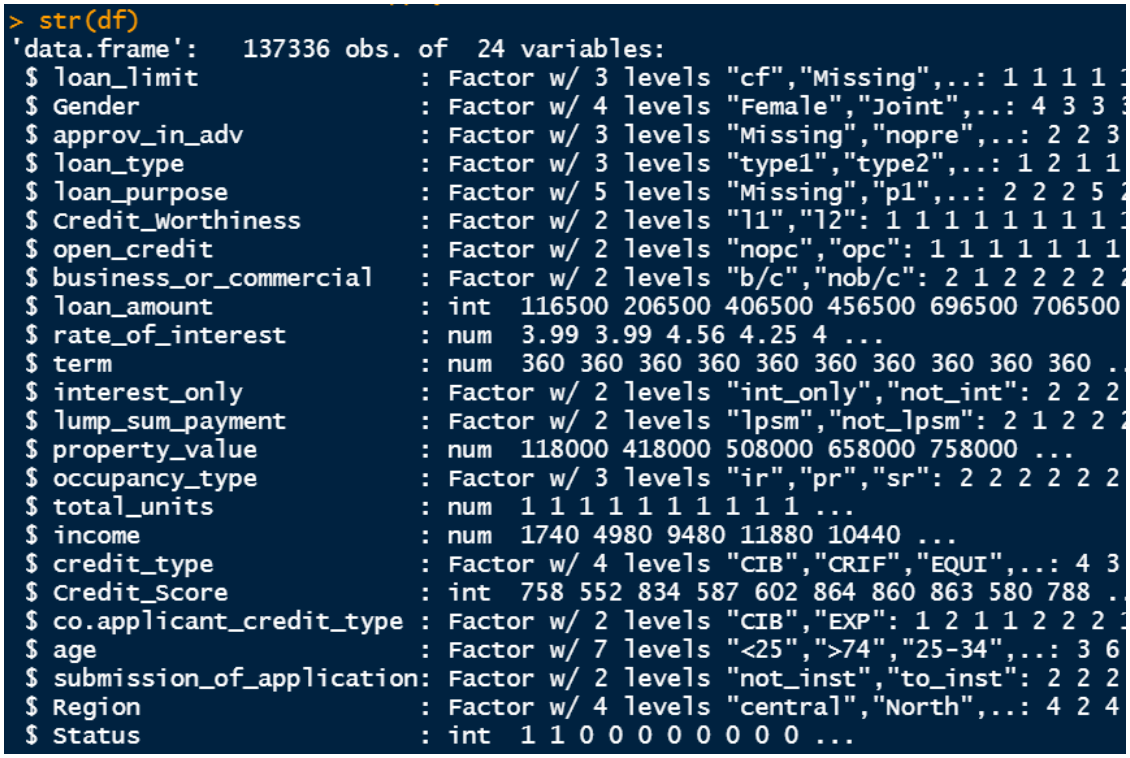
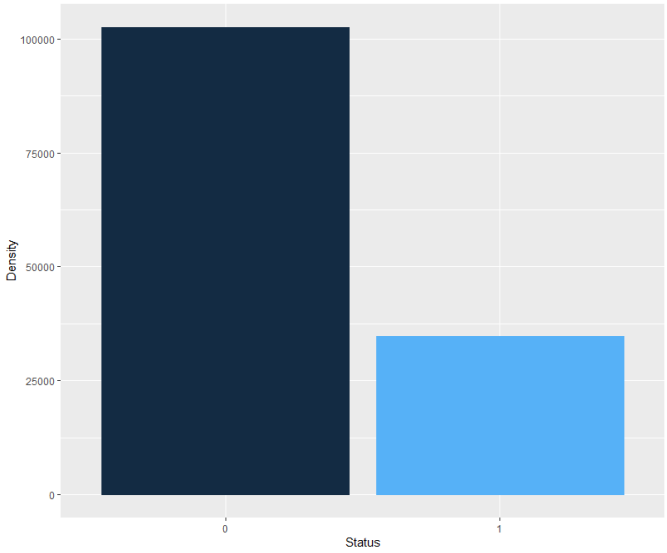


***Figure 1.24**: Final data structure*

2. Explanatory Data Analysis

a. Distribution validation

Since it is a loan default problem, we first take a look at our target variable. Our loan status is slightly skewed with the majority of non-default (75%), which is 3 times the count for default.



Total Observations in Table: 137336

0	1
102551	34785
0.747	0.253

***Figure 2.1**: Histogram and count for Loan Status.*

From the above analysis and after the cleaning of data there are multiple trends and patterns which could be observed and the same has been plotted below.

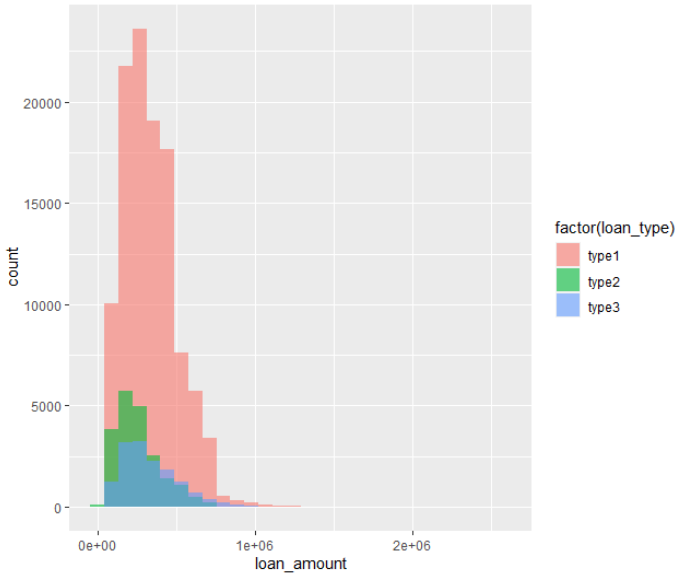


***Figure 2.2**: Histogram for Loan Amount on the basis of loan Type.*

The above chart indicates the count of loans taken and their types. The major contribution for the loans is the Type 1 loan having the highest amount as well for the same. The reason for the same could be that people might be using the same for business or commercial basis. The least amount of loan people prefer is the Type 3 loan.

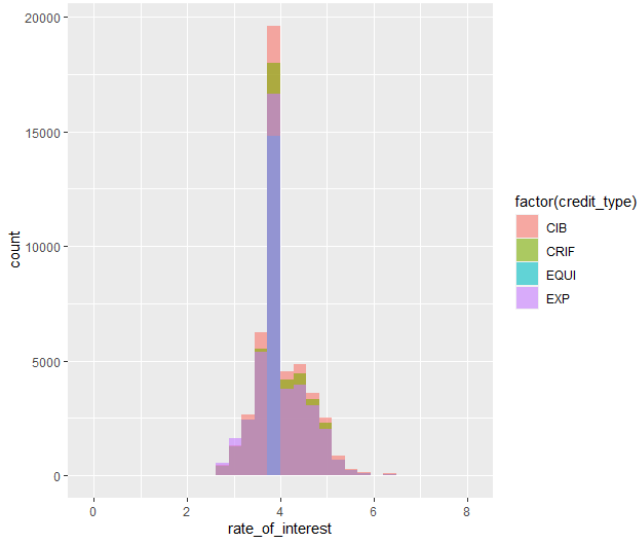


***Figure 2.3**: Histogram for Rate of Interest based on the Credit Type.*

The following histogram helps us in understanding the credit type and how much rate of interest is provided for the same and what is the frequency for the same. From the above chart it can be observed that CIB has the highest frequency with the rate of interest between 2% and 4% whereas the lowest includes the EXP credit type with less frequency and higher rate of interest of the same.

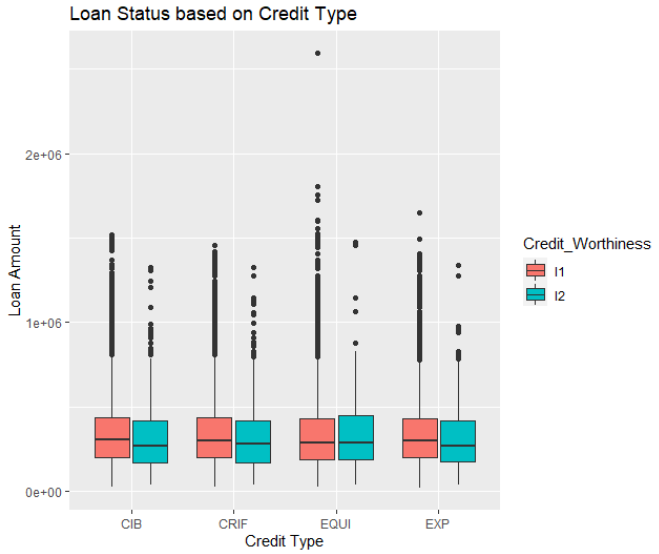


***Figure 2.4**: Boxplot for the multiple credit and the loan amount based on the credit worthiness.*

The beside plot explain the Loan amount sanctioned based on the credit type and it has been differentiated on the basis of the credit worthiness. From the above plot it could be observed that for each type of credit there is specific range of loan amount being allotted. Some values which are beyond the box plot needs to be investigated in regard to its importance and whether the same could be deleted as well.

***Figure 2.5**: Scatter Plot for the Advance approval of the loan based on credit score*

From the above chart it could be observed that the points are scattered equally, and some points could be observed to be scattered unevenly. Form the points scattered it could be observed that there is no advanced pre-approval given to multiple clients whereas very less advance approval could be observed from the scatter. This could be further taken up ahead in a correlation plot as well.

b. Correlation between the variables:

The Pearson correlation coefficient quantifies the magnitude of the relationship between the variables in the correlation matrix. The correlogram is a graphical representation of such association. The correlation coefficient value for loan amount and property value is 0.73, indicating that they are positively correlated, whereas the coefficient value for loan status and income is -0.12, indicating that they are negatively correlated.

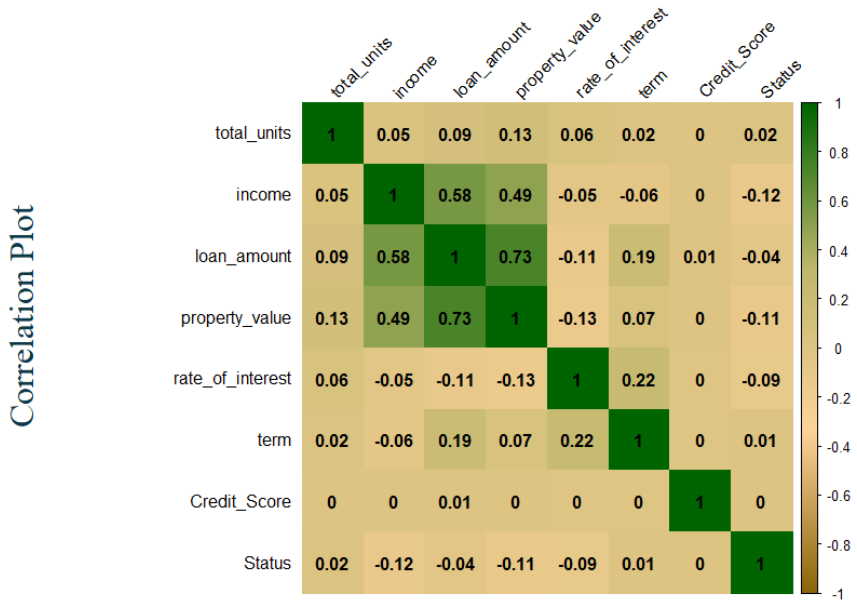
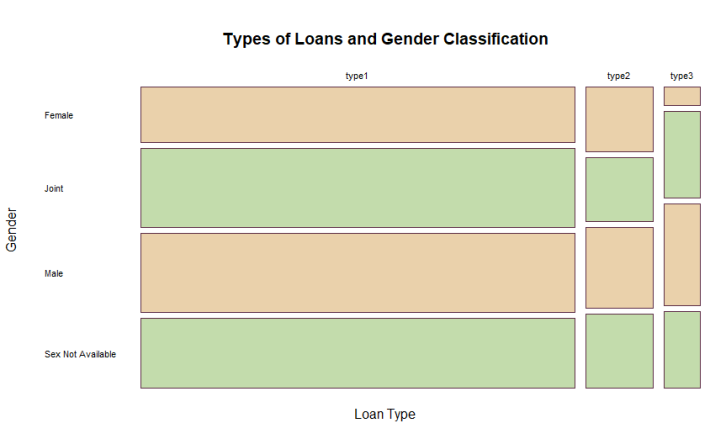


***Figure 2.6**: Correlation Plot*

Through Mosaic plot below, authors attempt to analyze the statistical association between the loan types and gender. The loan sanctioned to the borrowers are classified as Type 1 (80.7 percent), Type 2 (12.4 percent), and Type 3 (6.7 percent). According to the graph, males and joint (male and female) have received 22 percent, females have received 15 percent, and the population without gender disclosure has received 19 percent of Type 1 loans. We can also conclude that females received a lower percentage of loans across all loan categories compared to other gender classifications.




```
> prop.table(table(df$Gender,df$loan_type))

                       type1       type2       type3
  Female           0.157584284 0.028628373 0.004440439
  Joint            0.224791717 0.028049504 0.020700065
  Male             0.225055505 0.035457563 0.024290519
  Sex Not Available 0.199951639 0.032702439 0.018347952
>
```

***Figure 2.7**: Mosaic Plot: Statistical Association*

From the ***Figure 2.8***, we can infer that there exists a strong positive relationship between the property value and loan amount variables. In other words, when the property value rises the amount of money sanctioned for loan also increases.

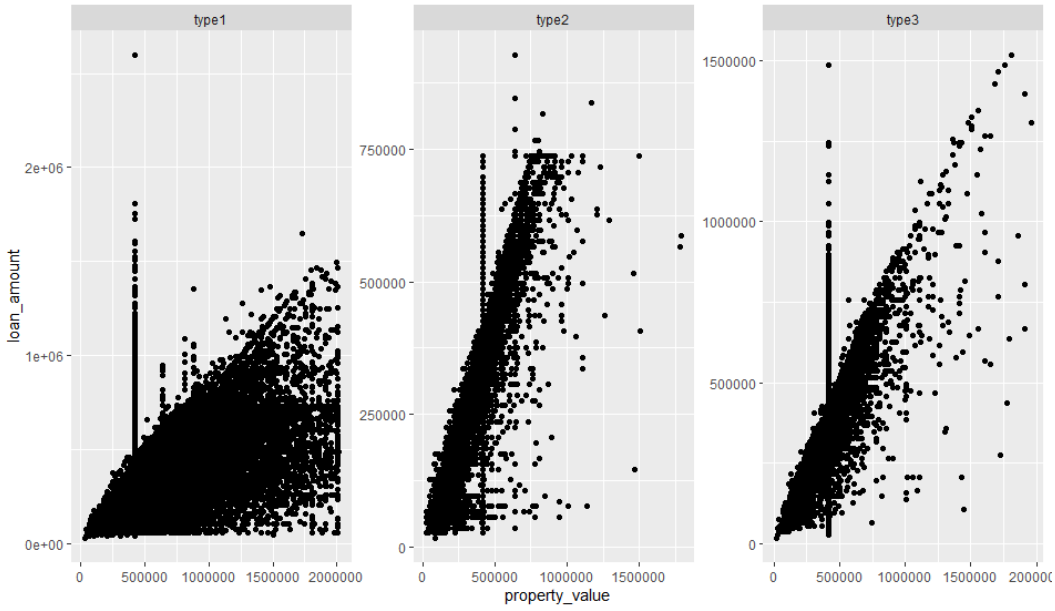


Figure 2.8*: Scatter Plot: Relation between Property value and Loan amount grants*

c. Explanatory power to the target

So far, we have looked at how the various variables are distributed and how they relate to one another. Now we will utilize cross-tabular analysis to see whether category variables have a positive impact on the Status variable's rate. For each region, we start by making a subset of the status variable's default value (1).

```
> table1

   central      North North-East      south
      2271      15991        362      15926
```

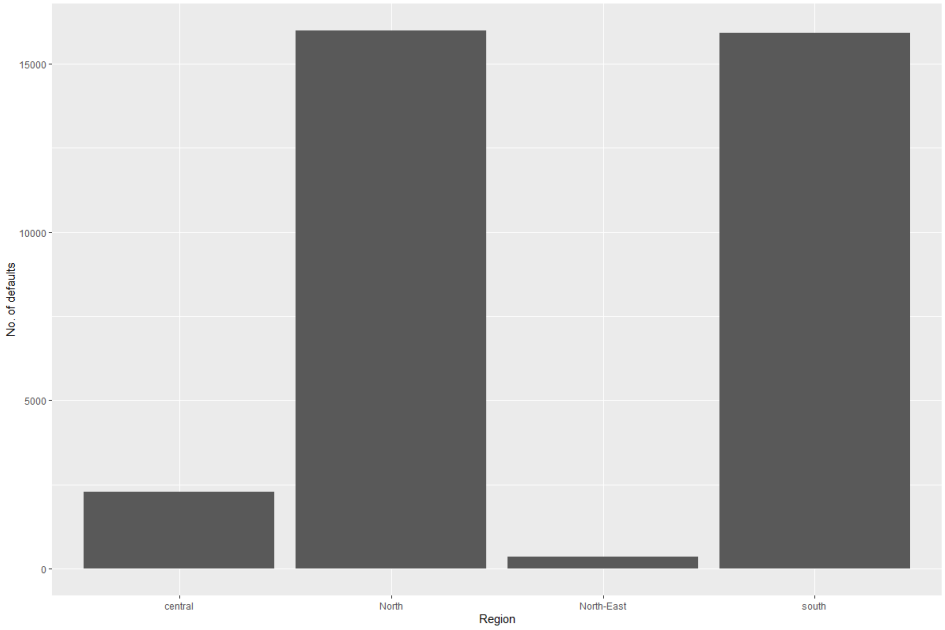


***Figure 2.9**: Region-wise default count*

The Northern and Southern Regions have a higher number of defaults than the other regions in the data, as seen in ***Figure 2.9***. As a result, we further divide our selection into higher and lower regions to examine the impact of other categorical variables.

```
> table2

       Female Joint Male Sex Not Available
  <25      56    58  160                71
  >74     349   399  468               666
  25-34   623   860 1407               937
  35-44  1091  1352 2124              1888
  45-54  1345  1482 2183              2220
  55-64  1376  1471 2031              2444
  65-74   939  1042 1148              1727
```

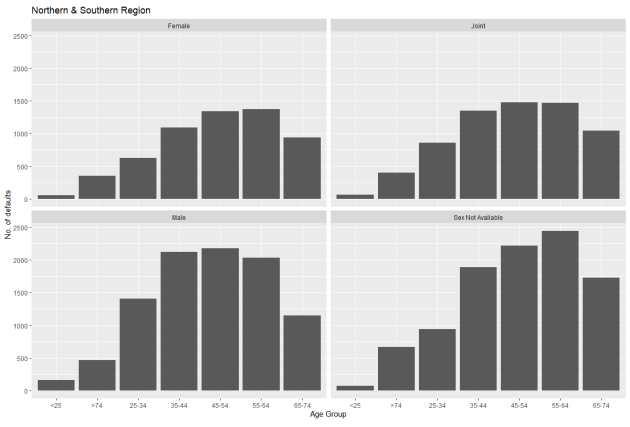


***Figure 2.10**: Northern and Southern Region Gender and Age-group wise default count*

```
> table3

      Female Joint Male Sex Not Available
<25        9     6    8                 0
>74       54    38   50                 0
25-34     61    98   85                 1
35-44    115   165  202                 3
45-54    204   190  247                 6
55-64    251   185  206                10
65-74    160   125  152                 2
```

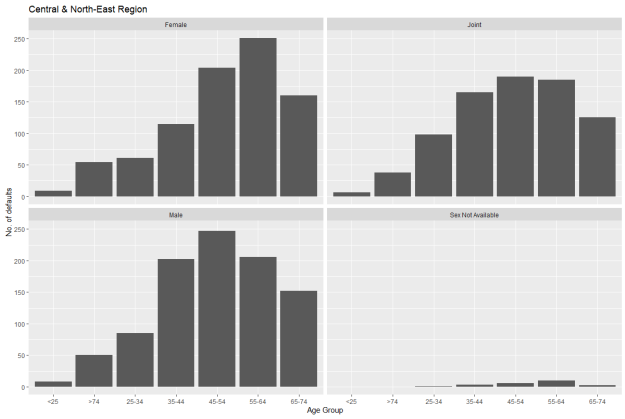


***Figure 2.11**: Central and North-East Region Gender and Age-group wise default count*

As indicated in the tables and graphs of **Figure 2.11**, the majority of defaults were in the 30-44, 45-54, and 55-64 age categories. Gender, on the other hand, has a significant impact in both geographical groups. As a result, we decided to dig deeper into those age groups to determine if the applicant's credit type and loan purpose (business or non-business) made a difference.

Finally, we discovered that non-business loans have a higher chance of defaulting based on the credit type and business or non-business crosstable study. Credit scores, on the other hand, have little bearing on it, even in different regions.

```
> table4

        CIB CRIF EQUI  EXP
b/c     930  815 1373  790
nob/c 3432 3226 7514 2927
```

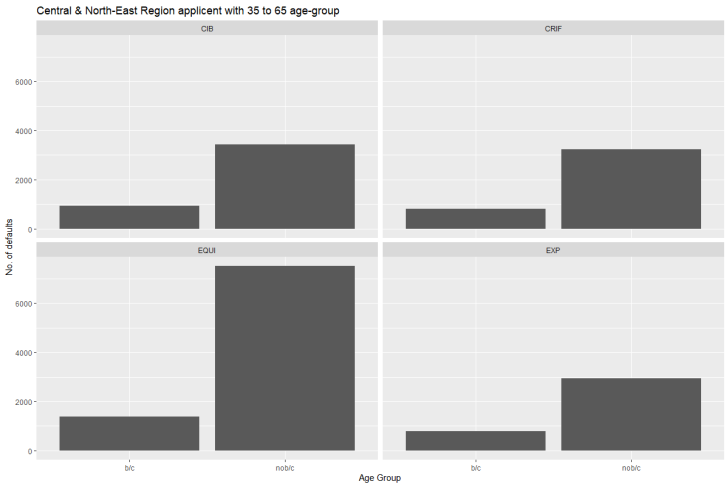


***Figure 2.12**: Credit type and business or non-business default count in major region*

```
> table5

       CIB CRIF EQUI EXP
b/c    121  134  158 105
nob/c 294  267  499 206
```

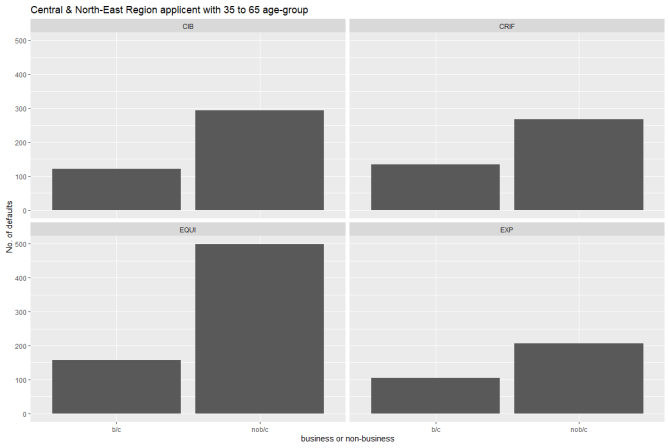


***Figure 2.13**: Credit type and business or non-business default count in major region*

3. Class bias and the importance of default classification

a. The problem of imbalanced dataset for Loan default

Highly imbalanced dataset: Even though authors dropped several observations in data manipulate and imputation, the class of target variable remains biased with the majority of non-default.

Why accuracy is less prioritized for imbalanced dataset in financial industry?

In practice, the minority class is usually the class of default in finance industry. Hence, attempts to predict default customer (Loan status = 1) is more crucial than for non-default customer (Loan status = 0). Or in other words, **False Negative (FN) – Error type 2 is a dangerous error** in model when it classifies customers as non-default, but they are actually unable to repay the loan, resulting to the company loss.

Class imbalance problems are a subset of problems relating to categorizing unbalanced data in which the majority class labels significantly more observations than the minority class. When trained on unbalanced data, however, most traditional statistics and machine learning models are significantly biased towards the majority class and substantially misclassify the minority class due to their assumptions of equal target class distribution and due to their target of maximizing the overall accuracy.

b. Solutions for class bias problem

To address these issues more effectively, data academics and professionals have used a variety of approaches, including:

No	Methods	Examples
1	Data sampling	Up-sampling, down-sampling, ROSE, SMOTE,…
2	Improved models	Weighted logistic regression, cost-sensitive logistic, decision tree upgrades (loss matrix, probability change, …), Random Forest,…
3	Threshold adjustment and hyperparameter tunning	Cut-off values, cp pruning, cost, gamma,…
4	Domain knowledge	Cost factor: Add cost for wrong prediction and choose the minimal cost model. Bad rate strategy: Associate cut-off with bad rate based on probability.

***Table 3**: Methods to deal with imbalanced data*

These methods are to reduce FN and somewhat increase Sensitivity rate, also known as Recall (TP/(TP+FN)) to predict more accurate default (TP) class.

In this report, the author will cover all above listed methods ranging from logistic regression to decision tree, XGBoost as well as Random Forest with improvement from sampling and hyperparameter tunning.

4. Feature Engineering and data split for modeling

a. Feature engineering to reduce data dimensionality

Even though authors dropped unnecessary columns from filter selection applying domain knowledge and missing data analysis, the dataset remains large horizontally with around 24 variables, in which some categorical variables with more than 4 levels, for prediction. To mitigate the curse of

dimension in modeling, authors continue to deploy another technique of feature engineering, which is weight of evidence (WOE) and information value (IV).

The WOE and IV could provide a great framework for exploratory analysis and variable screening for binary classifiers. Since one of benefits of these techniques is to evaluate the contribution of each independent variable to the outcome, we will leverage this to perform variable screening and select the most powerful explanatory predictors for our model when there are too many predictors. This step also helps us in easing the computation for a large data set.

```
> result_IV %>% arrange(desc(IV))
                    Variable           IV      PENALTY         AdjIV
1                credit_type 4.7099437758 4.4769492066  2.329946e-01
2           rate_of_interest 1.5606172795 0.0126053010  1.548012e+00
3             property_value 1.0860827809 0.0230526575  1.063030e+00
4           lump_sum_payment 0.1554766927 0.0035318315  1.519449e-01
5                     income 0.1368122802 0.0086186135  1.281937e-01
6   co.applicant_credit_type 0.1339541720 0.0014118757  1.325423e-01
7  submission_of_application 0.0821847134 0.0001745963  8.201012e-02
8                  loan_type 0.0771970271 0.0026632929  7.453373e-02
9     business_or_commercial 0.0555330342 0.0022667189  5.326632e-02
10                    Gender 0.0330305593 0.0054222958  2.760826e-02
11               loan_amount 0.0298489681 0.0044574576  2.539151e-02
12                    Region 0.0136776715 0.0011881164  1.248956e-02
13                       age 0.0130239666 0.0031790088  9.844958e-03
14               approv_in_adv 0.0101120412 0.0012723777  8.839664e-03
15                loan_limit 0.0096291974 0.0012855083  8.343689e-03
16         Credit_Worthiness 0.0046367265 0.0004170552  4.219671e-03
17              loan_purpose 0.0038234548 0.0002506684  3.572786e-03
18               total_units 0.0031402247 0.0016875386  1.452686e-03
19             occupancy_type 0.0031203891 0.0011781468  1.942242e-03
20                       term 0.0017637844 0.0008813795  8.824050e-04
21             interest_only 0.0007406019 0.0003482828  3.923191e-04
22               open_credit 0.0007230404 0.0003133342  4.097062e-04
23              Credit_Score 0.0004810035 0.0005129877 -3.198418e-05
```

***Figure 4.1**: Top 10 variables with highest contribution for target outcome*

Referring to **Figure 4.1** above, authors may pick top predictors with the most explanatory power to the outcome depending on the models.

Among chosen variables, we can see that **Credit type**, **rate of interest** and **property values** play the most important role for our target variable. One highlight we also notice is that **prior approved status** ranks only 14th, which we primarily assume must be high. And the same observation can be applied for **Credit Score**, which is the least important predictor.

b. Data split for modeling

In this session, the author simply splits data with 80% for training set and 20% for the test set. The split is conducted randomly without change the original proportion of class of target variable.

```
# Split data
# Set seed of 567
set.seed(567)
# Store row numbers for training set: index_train
index_train <- sample(1:nrow(df), 0.8* nrow(df))
# Create training set: training_set
training_set <- df[index_train, ]
# Create test set: test_set
test_set <- df[-index_train,]
```

Figure 4.2: *Data split*

5. Logistic regression models

a. Procedure of modeling

In this type of model, the author tries several phases of updating to either maximize model performance by improving overall accuracy or reduce False negative rates to better predict the default customers. More methods of bank strategy, ROC and AUC-based prunning to choose the best models will be illustrated in the end of this report.

Figure 5.1: *Working procedure to improve logistic models*

Below each model, we also denote model name for reference and easy recall when comparing.

b. Logistic full predictor model – log.full

This model is built from all predictors that we have from dataset to evaluate its performance. Model result and interpretation as below:

```
> log.full <- glm(Status ~ ., data=training_set, family = binomial(link = logit))
> summary(log.full)

Call:
glm(formula = Status ~ ., family = binomial(link = logit), data = training_set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.0741  -0.5980  -0.4350   0.0033   3.2848

Coefficients: (1 not defined because of singularities)
                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                    3.857e+00  4.574e-01   8.433  < 2e-16 ***
loan_limitMissing              2.171e-01  5.905e-02   3.677 0.000236 ***
loan_limitncf                  6.983e-01  3.647e-02  19.146  < 2e-16 ***
GenderJoint                    4.338e-02  3.833e-02   1.132 0.257798
GenderMale                     1.551e-01  2.702e-02   5.739 9.50e-09 ***
GenderSex Not Available        1.131e-02  3.458e-02   0.327 0.743716
approv_in_advnopre            -7.213e-02  1.105e-01  -0.653 0.514012
approv_in_advpre              -4.433e-01  1.131e-01  -3.920 8.86e-05 ***
loan_typetype2                 5.884e-01  2.645e-02  22.248  < 2e-16 ***
loan_typetype3                -1.647e-01  3.939e-02  -4.182 2.89e-05 ***
loan_purposep1                 4.424e-01  4.061e-01   1.090 0.275885
loan_purposep2                 1.167e+00  4.095e-01   2.849 0.004389 **
loan_purposep3                 5.122e-01  4.061e-01   1.261 0.207142
loan_purposep4                 5.630e-01  4.062e-01   1.386 0.165678
Credit_Worthinessl2            5.290e-01  4.299e-02  12.304  < 2e-16 ***
open_creditopc                 9.254e-01  2.160e-01   4.285 1.83e-05 ***
business_or_commercialnob/c           NA         NA      NA       NA
loan_amount                    3.348e-06  1.277e-07  26.208  < 2e-16 ***
rate_of_interest              -7.975e-01  2.276e-02 -35.041  < 2e-16 ***
term                           1.161e-03  1.734e-04   6.698 2.11e-11 ***
interest_onlynot_int          -1.175e-01  4.403e-02  -2.669 0.007597 **
lump_sum_paymentnot_lpsm      -2.395e+00  5.708e-02 -41.965  < 2e-16 ***
property_value                -1.317e-06  7.444e-08 -17.698  < 2e-16 ***
occupancy_typepr              -1.230e+00  4.253e-02 -28.913  < 2e-16 ***
age>74                        -1.195e-01  1.002e-01  -1.193 0.232694
age25-34                      -3.626e-01  9.448e-02  -3.837 0.000124 ***
age35-44                      -3.120e-01  9.344e-02  -3.339 0.000841 ***
age45-54                      -1.641e-01  9.343e-02  -1.756 0.079102 .
age55-64                      -1.278e-01  9.356e-02  -1.366 0.172079
age65-74                      -2.011e-01  9.487e-02  -2.120 0.033977 *
submission_of_applicationto_inst  9.301e-01  2.608e-02  35.667  < 2e-16 ***
RegionNorth                   -3.018e-01  3.798e-02  -7.948 1.90e-15 ***
RegionNorth-East               1.103e-02  9.759e-02   0.113 0.909995
Regionsouth                   -1.326e-01  4.194e-02  -3.162 0.001568 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 124402  on 109867  degrees of freedom
Residual deviance:  77867  on 109827  degrees of freedom
AIC: 77949

Number of Fisher Scoring iterations: 11
```

Since the model contains all predictors, it is not surprising that there are roughly 30% variables that are not statistically significant. In that, business_or_commercial (a categorical variable) has high multilinearity issue when its coefficient is NA. The deviance difference is 40 and AIC is extremely high, at 77949. Therefore, to improve this model, we move to the next model.

c. Stepwise logistic model – log.stepwise

```
> log.stepwise <- step(log.full, direction = "backward", trace = FALSE)
> summary(log.stepwise)

Call:
glm(formula = Status ~ loan_limit + Gender + approv_in_adv +
    loan_type + loan_purpose + Credit_Worthiness + open_credit +
    loan_amount + rate_of_interest + term + interest_only + lump_sum_payment +
    property_value + occupancy_type + total_units + income +
    credit_type + Credit_Score + co.applicant_credit_type + age +
    submission_of_application + Region, family = binomial(link = logit),
    data = training_set)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.0741  -0.5980  -0.4350   0.0033   3.2848

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 124402  on 109867  degrees of freedom
Residual deviance:  77867  on 109827  degrees of freedom
AIC: 77949

Number of Fisher Scoring iterations: 11
```

Apply backward selection method to reduce non-significant variables, we eliminate only two variables from the model, and it is left of total 22 variables in the model.

However, model performance measures remain unchanged for both AIC and deviance difference.

d. Model comparison

Compare log.full and log.stepwise:

The fundamental comparison methods for logistic regression are the Akaike Information Criterion (AIC) and the difference between null deviance and residual deviance. However, as we observed from the

model performance measures, they are consistent, which signifying that we do not know which one is better. Therefore, authors deploy ANOVA test for further model comparison and use method of Chisq. The *anova()* function is used to test whether a significant difference between the two models exists. To interpret the ANOVA test, compare the p-value with the significance level – typically 0.05. If the p-value is larger than the significance level, then there is no significant difference between the two models, and therefore the simpler model should be selected.

```
> # Compare two models
> anova(log.full,log.stepwise,test="LRT")
Analysis of Deviance Table

Model 1: Status ~ loan_limit + Gender + approv_in_adv + loan_type + loan_purpose +
    Credit_Worthiness + open_credit + business_or_commercial +
    loan_amount + rate_of_interest + term + interest_only + lump_sum_payment +
    property_value + occupancy_type + total_units + income +
    credit_type + Credit_Score + co.applicant_credit_type + age +
    submission_of_application + Region
Model 2: Status ~ loan_limit + Gender + approv_in_adv + loan_type + loan_purpose +
    Credit_Worthiness + open_credit + loan_amount + rate_of_interest +
    term + interest_only + lump_sum_payment + property_value +
    occupancy_type + total_units + income + credit_type + Credit_Score +
    co.applicant_credit_type + age + submission_of_application +
    Region
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1    109827      77867
2    109827      77867  0        0
```

Figure 5.2: *ANOVA test for model comparison.*

In the output above, since the p-value of 0 is smaller than 0.05, we reject the null hypothesis and can conclude that the nested model (log.stepwise) does not performs better than the original model (log.full).

However, since log.stepwise reduce multicollinearity from one variable, we adopt it for prediction.

Prediction on test set for log.stepwise:

Apply the cut-off value of 0.5, we test the performance of model on test set with following result:

```
> confusionMatrix(t(conf_matrix),positive = "1")
Confusion Matrix and Statistics

          Actual
Pred         0     1
  0      20368  3589
  1        166  3345

               Accuracy : 0.8633
                 95% CI : (0.8592, 0.8673)
    No Information Rate : 0.7476
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.567

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4824
            Specificity : 0.9919
         Pos Pred Value : 0.9527
         Neg Pred Value : 0.8502
             Prevalence : 0.2524
         Detection Rate : 0.1218
   Detection Prevalence : 0.1278
      Balanced Accuracy : 0.7372

       'Positive' Class : 1
```

Even though the overall accuracy is good with 86.33%, the Sensitivity is only 48.24%, significantly in comparison with Specificity.

It is noticeable that the number of FN is 3589, which is even higher than the TP of 3345. This means the chance to predict a true TP is only less than 50%, and mostly we predict wrongly.

To tackle the above issues, we could deploy **Cost-sensitive logistic regression** – assigning cost for wrong prediction or **Weighted logistic regression** – an updated version of Logistic regression with only a minor change in hyperparameter to classify FN more accurately. Within the scope of this report, authors decide to go with the second option since we can easily tune the model by changing hyperparameter.

e. Weighted Logistic Regression – log.weighted

In this model, we will add weight into regression that computer would recognize which factors (TARGET = 0 or TARGET = 1) are more important, and hence it will focus more on prediction of prioritized factor. To calculate the weight, please consult below formula:

$$w_j = \frac{n}{kn_j}$$

where w_j is the weight to class j, n is the number of observations, n_j is the number of observations in class j, and k is the total number of classes.

Here, as we can see: the target variable is not highly imbalanced with 75% of 0 (82017) and 25% of 1 (27851); therefore, the assigned values for each class a weight are not highly different, in which 0 (majority class) has weight of 0.669, while 1 (minority class) has weight of 1.97, roughly 3 times higher only. This technique is called balanced weight, but in reality, weighted values should be computed by domain knowledge.

```
> table(training_set$Status)

    0     1
82017 27851
> n<- length(training_set$Status)
> n0 <- table(training_set$Status)[1]
> n1 <- table(training_set$Status)[2]
> k<-2
> model_weights<- ifelse(training_set$Status == 0,n/(k*n0),n/(k*n1))
> table(model_weights)
model_weights
0.669787970786544  1.97242468852106
            82017             27851
```

***Figure 5.3**: Create new column of model weight.*

Simply adding weights into the regression, as displayed in **Figure 5.4**, we have the result as below.

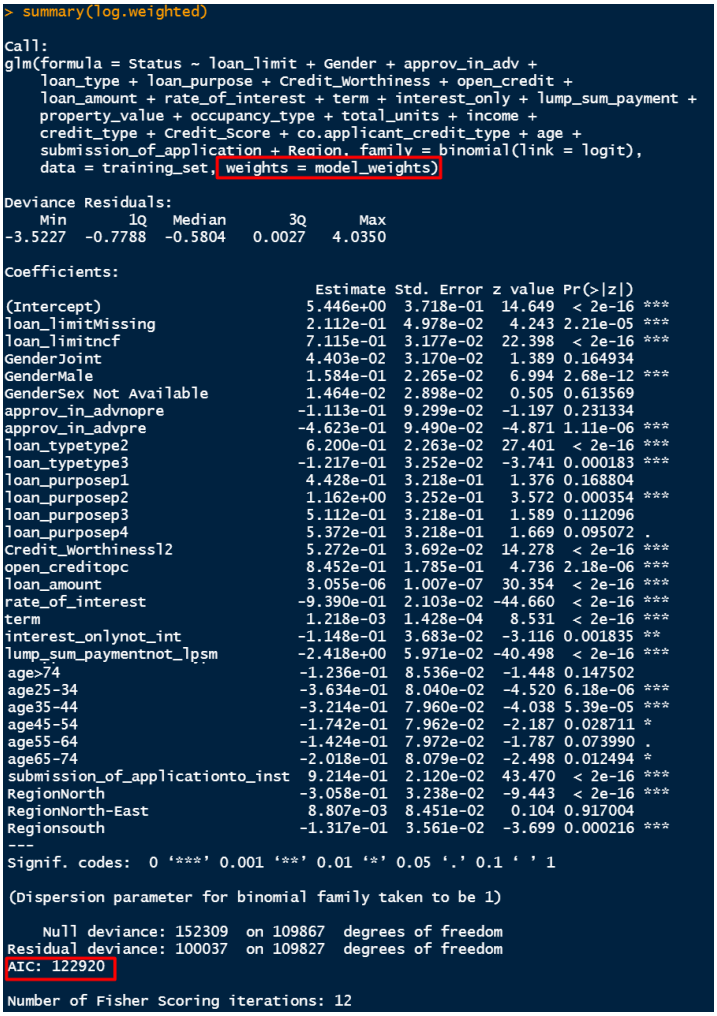


***Figure 5.4**: Weighted logistic regression performance*

The increment of AIC (now at 122,920 from 77,949) and decrease of accuracy (now at 81.7% from 86.33%) are expected when conducting weighted logistic regression because the model tries to predict the TP (minority) than TN (majority). But it is also noticeable that Sensitivity is now grown significantly, at 64.10% and the FN is now at 2489, which is smaller than TP of 4445. The model reduces FN to be at 1:2 ratio to TP.

f. Final Comparison

Please consult comparison table below for conclusion.

Model	Accuracy	Sensitivity	FN	F1	AUC
Log.stepwise	86.33%	48.24%	3589	0.6404	0.8453
Log.weighted	81.70%	**64.10%**	2489	0.6388	**0.8461**

***Table 4:** Model comparison*

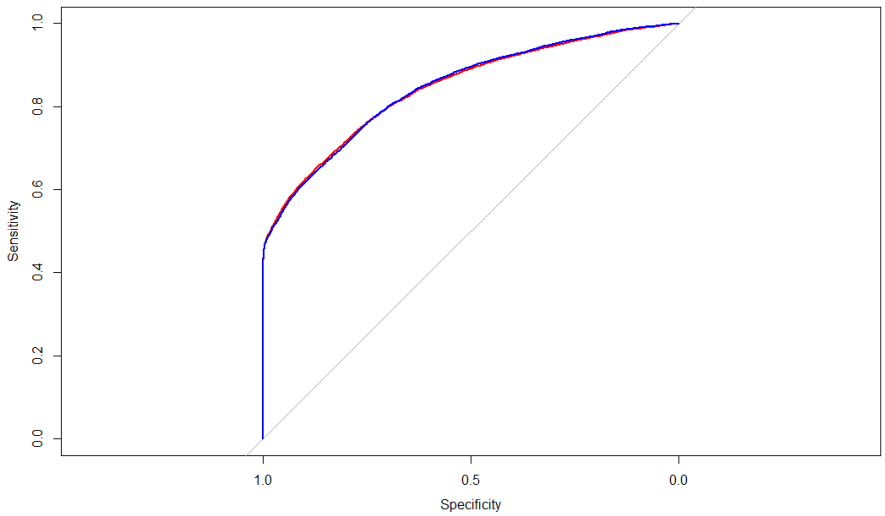


***Figure 5.5**: ROC performance of two logistics models*

In terms of accuracy, weighted logistic regression is surpassed by stepwise logistic model. However, when taking account to overall performance (AUC) and Sensitivity, it outperforms the stepwise counterpart.

6. Decision tree models

a. Procedure of modeling

In this type of model, the authors try several phases of updating to either maximize model performance by improving overall.

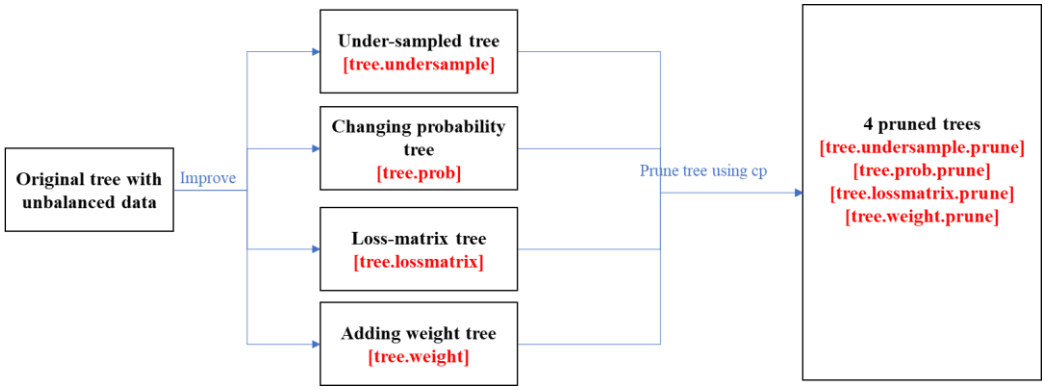
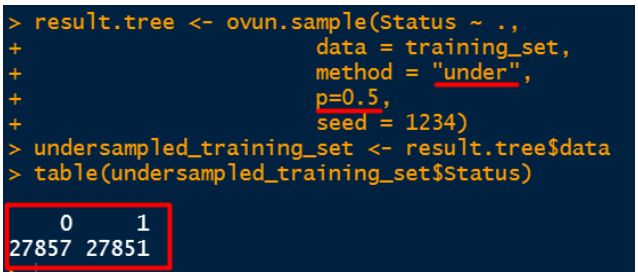
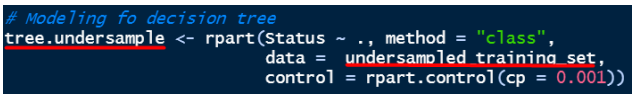
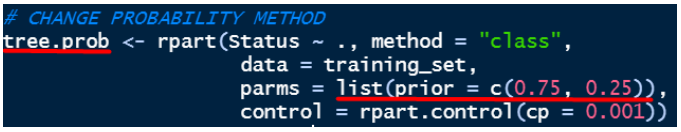
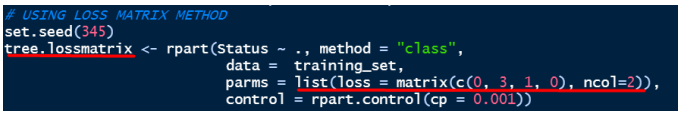


***Figure 6.1**: Procedure for decision tree algorithm*

b. Decision Tree modelling with the improvement methods

Since we cannot run the original model with unbalanced data, this part will showcase improvement from the original tree.

Improvement method	Explanation
Under-sampling method – tree.umdersample   	In order to have a better balance in class distribution, undersampling strategies eliminate samples from the training dataset that belong to the majority class. In this, we create a new sample with 50% for minority and 50% for majority by **ROSE** package, in which the number of observations for 0 is 27857 and for 1 is 27851. After that, we will apply a decision tree based on this training set to build a predicted model.
Changing probability method – tree.prob  	We change the prior probabilities to obtain a decision tree. This is an indirect way of adjusting the importance of misclassifications for each class. In the code, we change the proportion of non-defaults to 0.75, and of defaults to 0.25 according to the proportion of class in target variable.
Loss matrix method – tree.lossmatrix  	In this method, we stress that misclassifying a default as a non-default should be penalized more heavily by a loss matrix. In details, we penalize non-default at cost of 3 and default at cost of only 1.

Adding weight method – tree.weight	By specifying higher weights for default, the model will assign higher importance to classifying defaults correctly. This vector contains weights of 1 for the non-defaults and weights of 3 for defaults in the training set.

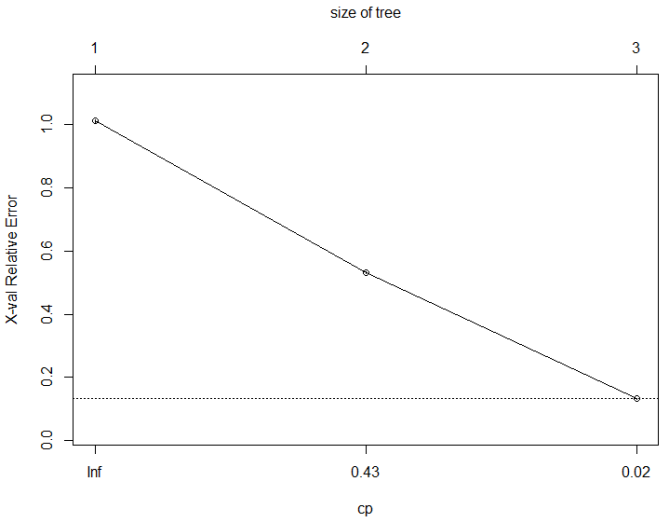
```
# USING WEIGHT
# Create weight
case_weights_tree <- ifelse(training_set$Status== 1,3,1) # 3 for default, 1 for non-default
# Run model
set.seed(345)
tree.weight <- rpart(Status ~ ., method = "class",
                     data = training_set,weights=case_weights_tree,
                     control = rpart.control(cp = 0.001))
```

c. Further improvement using optimal cp

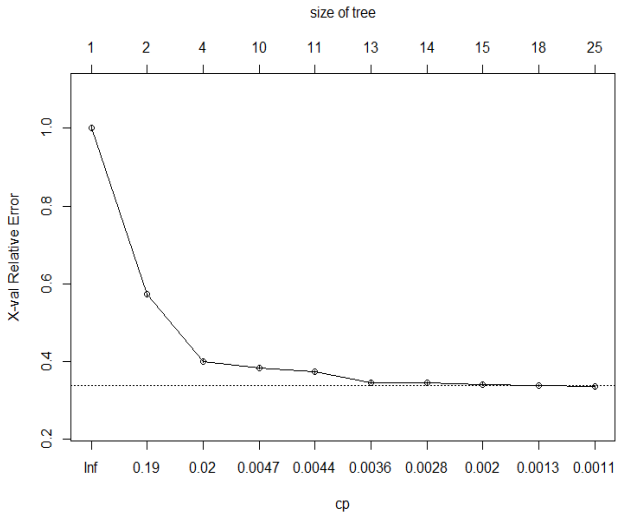
Up to this step, all models still have many splits that are challenging to vilusualize. And one major problem of trees is overfitting, thus we need to prune trees, in this case, using optimal cp.

cp is the complexity parameter, the value at which the overall lack of fit for any split decreases. Further splits will not be pursued if cp is not fulfilled. The default value of cp is 0.01, although it is recommended to relax cp for complex problems.

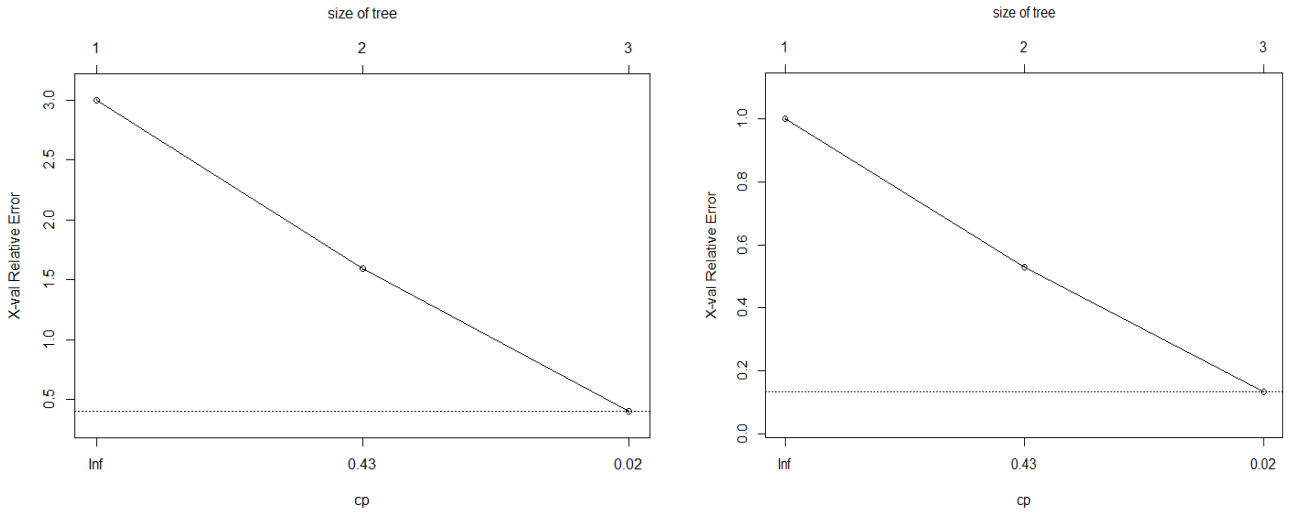
Below are the plots for cp along with errors of four models:



Under-sampling method



Changing probability method



Loss matrix method **Adding weight method**

***Figure 6.2: cp** plot for four trees*

Utilizing the cp tables as well as cp plots, we could pick out the most optimal cp values for each model. As tree structrures are different, the optimal values also vary as shown in **Figure 6.2**.

While undersample model, loss matrix model and adding weight model have the small size of tree in the plot , which is 3, we will use the optimal cp for these models. However, for the probability tree, we will pick cp = 0.0036 as this figure creates 13 trees, which may help us in avoiding overfitting, instead of cp of 0.0011 for 25 tree plits.

Prune model using optimal cp as stated above, we execute the following code.

```
# Prune the tree using optimal cp
tree.undersample.prune <- prune(tree.undersample, cp = tree_min_u)
tree.prob.prune <- prune(tree.prob, cp = 0.0036)
tree.lossmatrix.prune <- prune(tree.lossmatrix, cp = tree_min_l)
tree.weight.prune <- prune(tree.weight, cp = tree_min_w)
```

***Figure 6.3:** Pruning four trees*

Finally, we could use *prp()* or *rpart.plot()* to draw a tree for visualization. Below are 4 pruned trees plot.

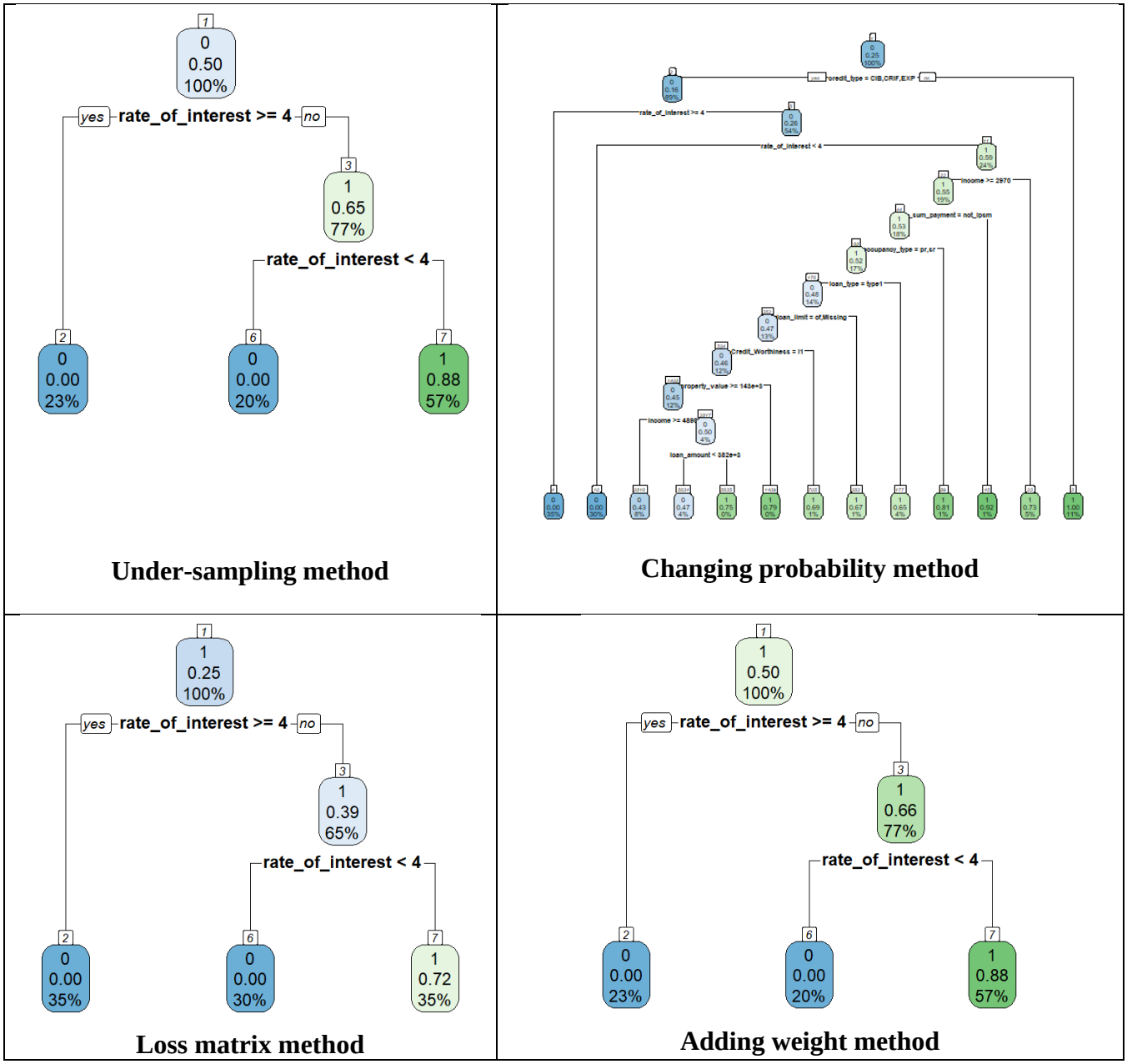
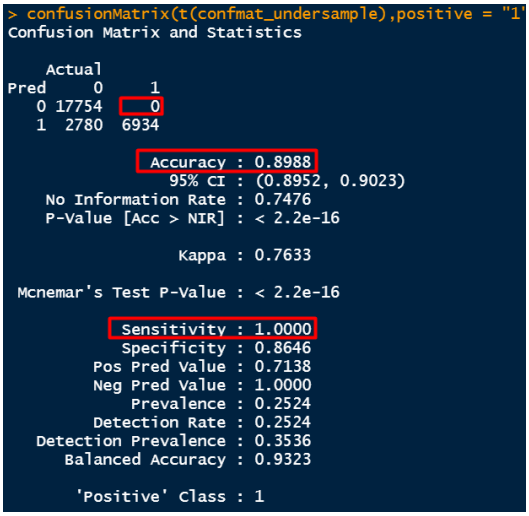


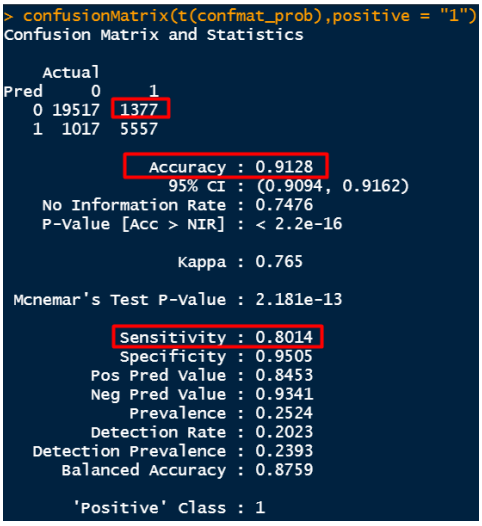
Figure 6.4: *Decision trees plot*

d. Prediction evaluation

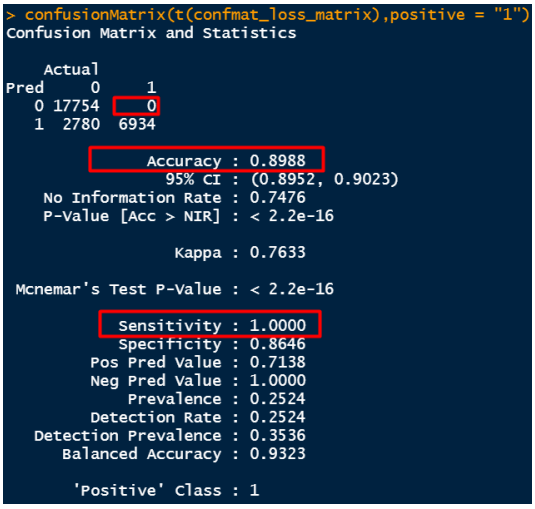
Using *predict()* function for four models on test set, and combining with the actual classification, we could easily generate a confusion matrix. At the same as the previous part, we also deploy ***caret*** package to compute tree performance metrics.



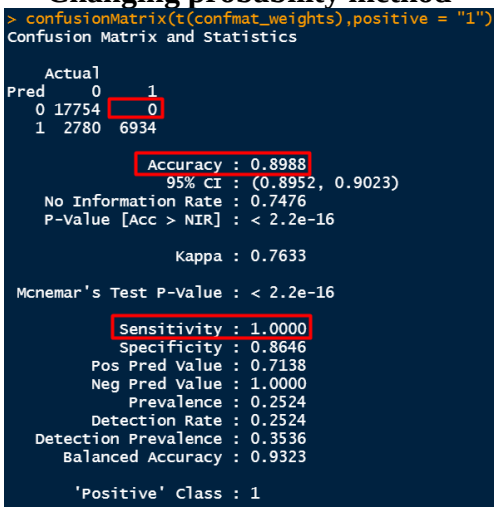
Under-sampling method



Changing probability method



Loss matrix method



Adding weight method

Figure 6.5: *Performance of four pruned trees*

For accuracy, **pruned Changing probability tree** perform the best with 91.28%. However, it misclassifies 1377 default borrowers with Sensitivity is 80.14%.

The rest of 3 trees have the same performance while their accuracies are 89.88% and Sensitivities are 100% without any FN.

e. Final Comparison

Please consult comparison table below for conclusion.

Model	Accuracy	Sensitivity	FN	F1	AUC
tree.undersample.prune	89.88%	100.00%	0	0.8330	0.9323
tree.prob.prune	**91.28%**	80.14%	1377	0.8223	0.8759

tree.lossmatrix.prune	89.88%	**100.00%**	**0**	**0.8330**	**0.9323**
tree.weight.prune	89.88%	100.00%	0	0.8330	0.9323

Table 5: *Model comparison*

As we can see, the 1st, 3rd and 4th are the same, while the 2nd model is somewhat different. Thus, in the final comparison, author may use only 2 models from the table to evaluate each method performance.

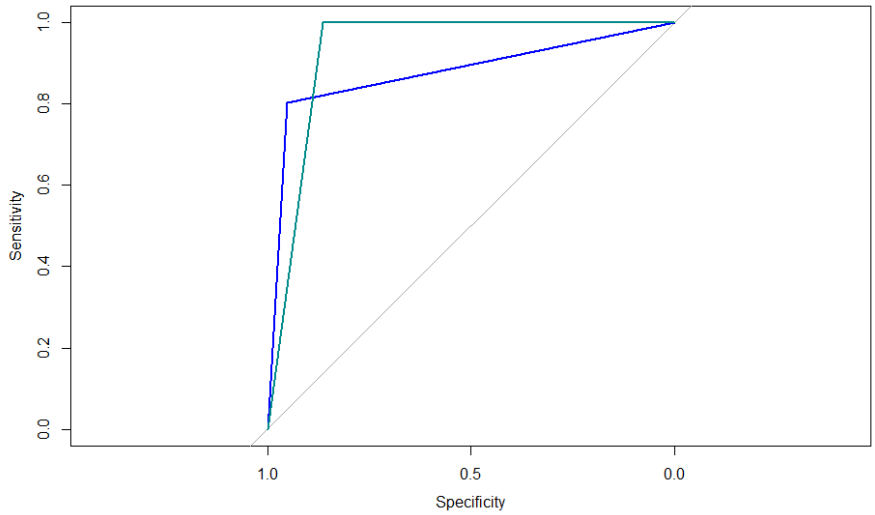


Figure 6.6*: ROC performance of two logistics models*

In terms of accuracy, tree.prob.prune is surpassed the rests. However, when taking account to overall performance (AUC) and Sensitivity, tree.lossmatrix.prune outperforms the its counterpart.

7. XGBoost models

a. Procedure of modeling

The extreme gradient boosting algorithm is a machine learning algorithm that employs the gradient boosting framework. It supports both linear model solvers and decision-tree learning algorithms.

In this part, we develop full XGBoost model with all predictors to check its original performance, but then we demonstrate how to use tunning hyperparameters and Feature Importance to develop a more robust XGBoost model in other cases as the current original model already works well in imbalanced data set without much improvement

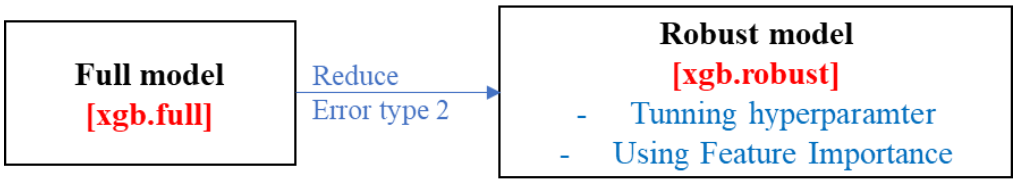


***Figure 7.1**: Improvement procedure of XGBoost*

b. XGBoost model – xgb.full

Initially, a training and testing subset is created from the data using 80:20 split. To improve the model's prediction and classification accuracy, all classes in categorical variables are converted to dummy variables using the One hot encoding process. The function *sparse.model.matrix()* from the **xgboost** package is used to convert all the categorical variables. Run the model with 100 iterations, we have the summary below:

```
> xgb_model
##### xgb.Booster
raw: 813.4 Kb
call:
  xgb.train(params = xgb_params, data = train_matrix, nrounds = 100,
    watchlist = watchlist)
params (as set within xgb.train):
  objective = "multi:softprob", eval_metric = "mlogloss", num_class = "2", v
alidate_parameters = "TRUE"
xgb.attributes:
  niter
callbacks:
  cb.print.evaluation(period = print_every_n)
  cb.evaluation.log()
# of features: 42
niter: 100
nfeatures : 42
evaluation_log:
    iter train_mlogloss test_mlogloss
       1       0.494340      0.494301
       2       0.384170      0.384138
---
      99       0.114238      0.135762
     100       0.113959      0.135785
>
```

***Figure 7.2:** XGBoost result summary*

Apply the model to evaluate its performance on test set, we have result as following:

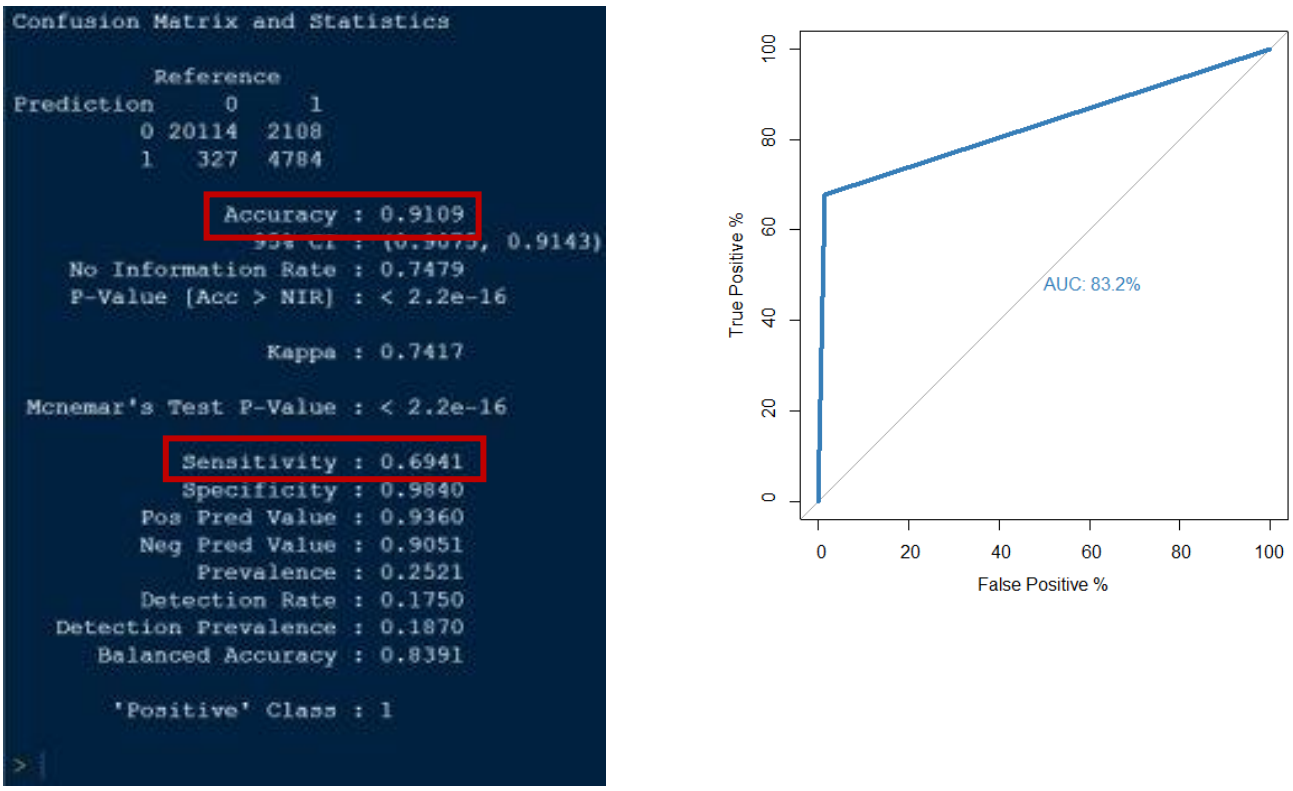


Figure 7.3: *Confusion matrix and AUC for XGBoost*

The model correctly classifies 4784 as loan defaulters (TP) but also has 2108 FN. The model's classification accuracy is 91% but the Sensitivity is only 69.4%. Besides that, F1 score is 0.831 and The value of the area under the curve is 83.2%.

High number of FN leaves a major problem for this model, and therefore we can improve further to acquire a better Sensitivity.

c. Apply boosting parameters and feature selection for model tunning

Parameter boosting: A boosting model is developed using the function xgb.train. In order to tune the model, boosting parameters such as eta, max_depth, and subsample are used.

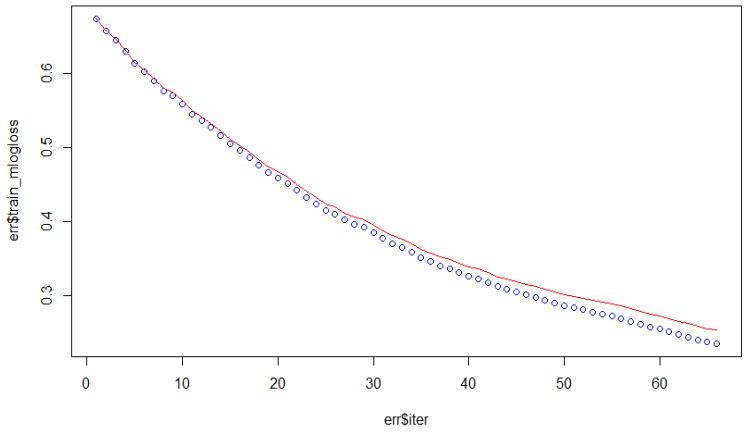


Figure 7.4: *Error difference in training and test set*

From the above graph in **Figure 7.4**, we can infer that the error difference between train set and testing set is minimal. The blue dotted lines indicate the trainset error, while the red line indicates the test set.

Feature importance application: The xgb.importance function provides the information of features importance. The rate of interest, credit type, property value, income, and credit score are some of the factors which contribute to the model's significance.

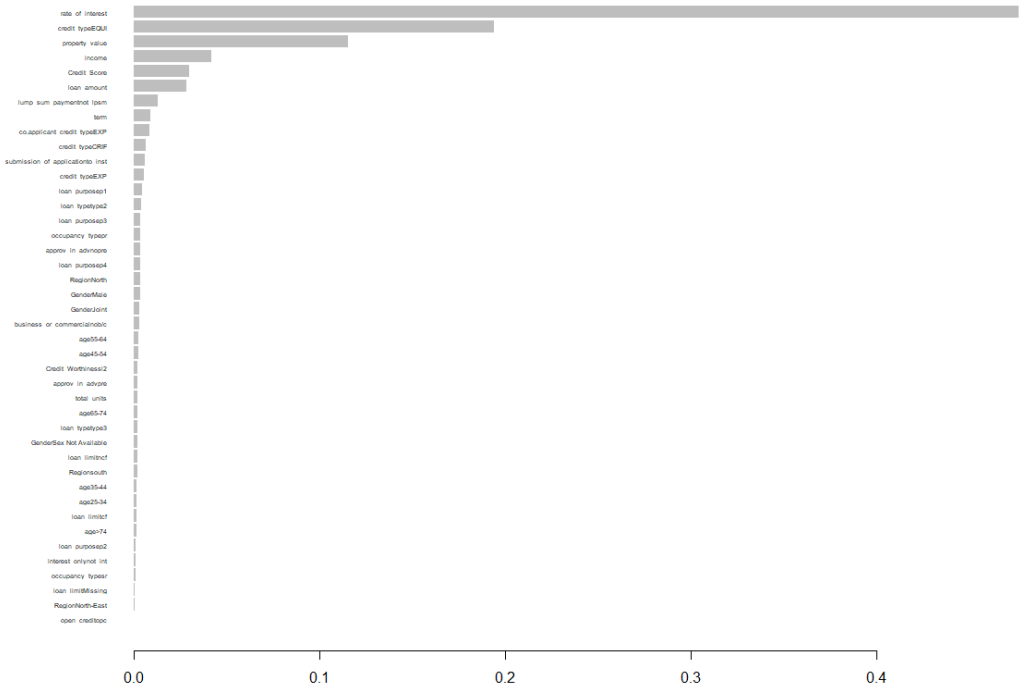


Figure 7.5: *Error difference in training and test set*

d. XGBoost model – xgb.robust

Using both the hyperparameter tuning and some important features for this model to optimize further, we test again on test set and acquire the below result.

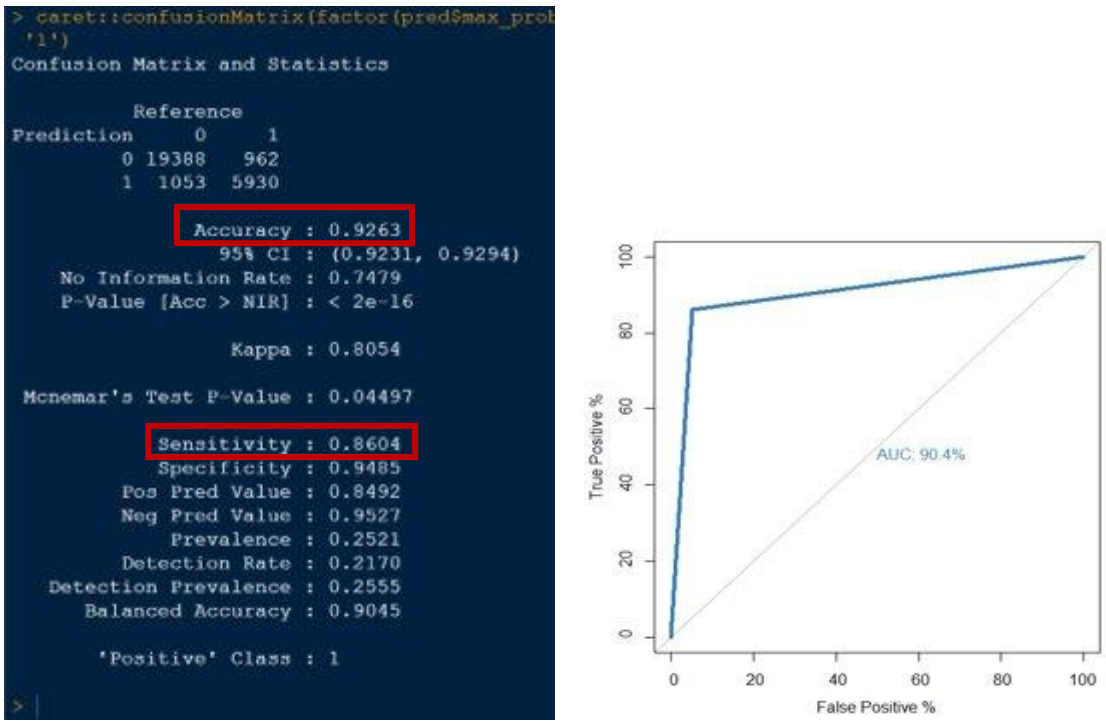


Figure 7.6: *Performance Result of tunning model (xgb.robust)*

We could notice that the performance of the model against the unseen data has increased. The False negative value is reduced to 962. The sensitivity has increased by 16% to 86.04%. The area under the curve value is also uplifted to 90.4%

e. Final Comparison

Please consult comparison table below for conclusion.

Model	Accuracy	Sensitivity	FN	F1	AUC
xgb.full	91.09%	69.41%	2108	0.850	0.839
xgb.robust	92.63%	86.04%	962	0.874	0.904

Table 6: *Model comparison*

From the table, we can conclude that the robust model using hyperparameter tunning and feature importance clearly outperformance the original XGBoost model in all metrics

8. Random Forest models

a. Procedure of modeling

A Random Forest is made up of a huge number of independent decision trees that work together as an ensemble. Each individual tree in the Random Forest produces a class prediction, and the class with the most votes becomes the prediction of our model. As it could work with classification as well as regression, it can be improved by many parameters for our problem. In this project, we will have several models as displayed below procedure:

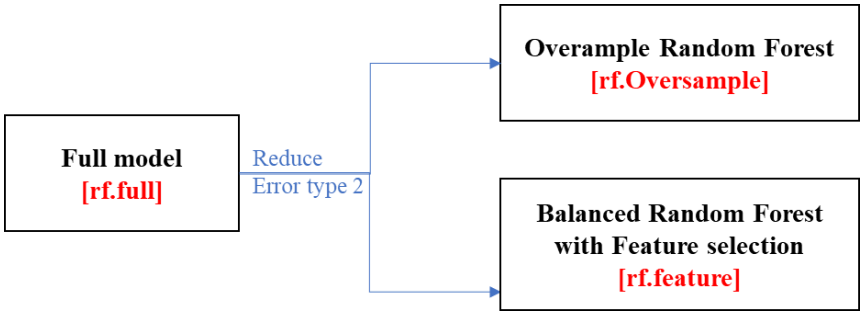


***Figure 8.1**: Random Forest process*

We will start with a Random Forest with default parameters first. In order to improve accuracy, we actually could build a grid search to find out the best **mtry** (number of selected variables) and **ntree** (number of trees). However, since we target to Sensitivity and reduce FN, we have two upgrades from the original models: **The oversampled Random Forest** and the **Balanced Random Forest with feature selection.**

b. Random Forest with full predictors and default parameters – rf.full

This model is built from all predictors that we have from dataset and set with default values to evaluate its performance. Model result and interpretation as below:

```
> rf.full <- randomForest(Status~.,
+                         data = training_set)
> rf.full

Call:
 randomForest(formula = Status ~ ., data = training_set)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 7.51%
Confusion matrix:
      0     1 class.error
0 78309  3708  0.04521014
1  4540 23311  0.16301030
```

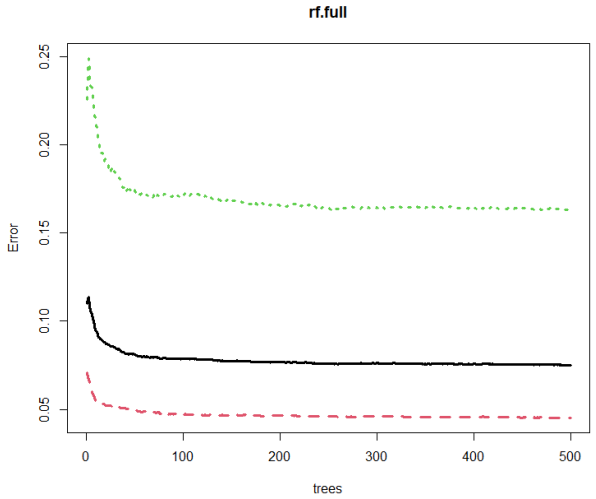


***Figure 8.2**: Model result and plot on training set*

```
> confusionMatrix(t(rf.full.matrix),positive = "1")
Confusion Matrix and Statistics

    Actual
Pred     0     1
   0 19538  1141
   1   996  5793

               Accuracy : 0.9222
                 95% CI : (0.919, 0.9253)
    No Information Rate : 0.7476
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7924

 Mcnemar's Test P-Value : 0.001839

            Sensitivity : 0.8354
            Specificity : 0.9515
         Pos Pred Value : 0.8533
         Neg Pred Value : 0.9448
             Prevalence : 0.2524
         Detection Rate : 0.2109
   Detection Prevalence : 0.2472
      Balanced Accuracy : 0.8935

       'Positive' Class : 1
```

***Figure 8.3**: Confusion matrix on test set*

Simply conduct the model with all default setting, the computer has run 500 trees with 4 variables each time.

The result is applied for Training set only. Out of bag error rate is great, at 7.51%. This indicates the power of Random Forest in prediction, even with a very first model. However, there is one warning that the class error for Positive (Default) is significantly higher than the Non-default, at 16.3%.

The plot of Error rate also demonstrates this observation, in which Default error is green line, while non-default is the red line.

Apply model to predict on test set, we have the confusion matrix as beside **Figure of 8.3**.

Overall accuracy is excellent, at 92.22%. While the Sensitivity is also great with 83.54%. The same insight could be drawn if we take a look at FN of only 1141, comparing to TP of 5793.

Conclusion: Even this is a default model, Random Forest performs effectively on both training and test sets. But let move forward to upgraded models to see if we could optimize further.

c. Oversampled Random Forest model – rf.Oversample

To run this model, the authors first deploy ROSE package to balance the class of distribution by oversampling method. The result balanced training set contains 82017 non-default observations and 81997 defaults, which is relatively equivalent to the another.

```
# Over sample method
oversamp <- ovun.sample(Status ~ .,
                        data = training_set,
                        method = "over",
                        p = 0.5,
                        seed = 1234)
oversampled_training_set <- oversamp$data
table(oversampled_training_set$Status)
```

```
> table(oversampled_training_set$Status)

    0     1
82017 81997
```

***Figure 8.4**: Over sampling for training set*

Then the authors apply the model on this newly created training set. Result on training and confusion matrix are shown below:

```
> rf.Oversample = randomForest(Status~., data = oversampled_training_set,
+                              ntree = 500,
+                              mtry = 5)
> rf.Oversample

Call:
 randomForest(formula = Status ~ ., data = oversampled_training_set,
 ntree = 500, mtry = 5)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 4.53%
Confusion matrix:
      0     1 class.error
0 74884  7133 0.086969775
1   304 81693 0.003707453
```

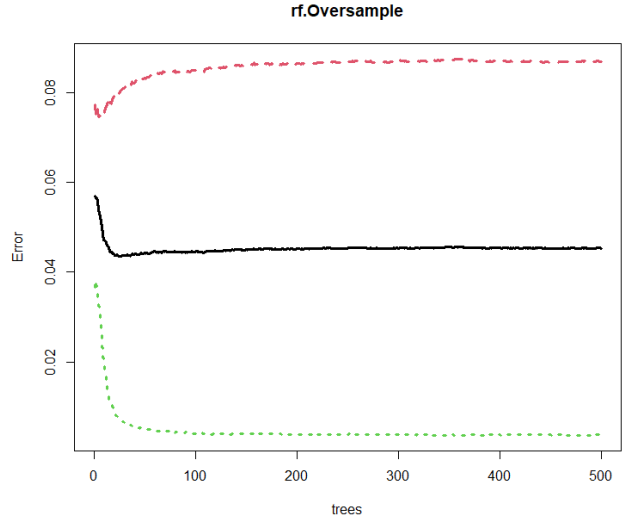


***Figure 8.5**: Model result and plot on training set*

For this model, the performance of model on training set has improved significantly on both OOB rate as well as error rate of Default.

The error plot illustrates a flip in error rates for default and non-default class, in which the green line is now minimized to 0.37%, a very marginal error rate for Default customers. This is a very robust performance of the model for imbalanced data set.

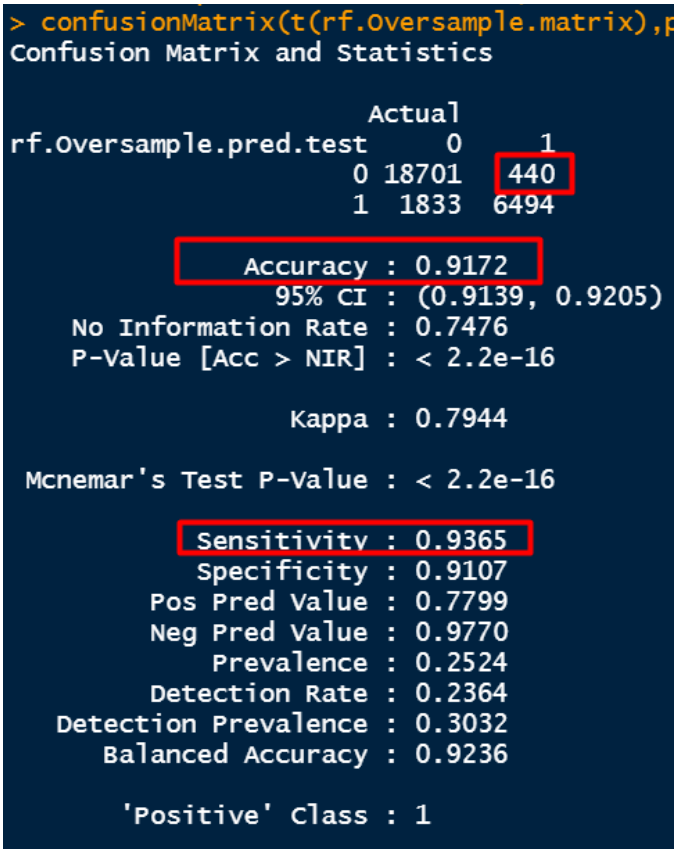


***Figure 8.6**: Confusion matrix on test set*

Apply model to predict on test set, we have the confusion matrix as beside **Figure of 8.6**.

Overall accuracy remains excellent, at 91.72% (only reduced 1% compared to the original). While the Sensitivity is uplifted to 93.65% (More 10% than the original one) with only 440 FN.

Conclusion: The oversampled Random Forest performs effectively on both training and test sets. It improves substantially the prediction for TP.

d. Balanced Random Forest model with Feature Selection – rf.feature

In this model, authors will run a Balanced Random Forest by adjusting 2 tunning parameters of the model and only run the model on variables with the most explanatory powers in Feature Selection session (Part 4 of this report).

Predictors are narrow down from 23 to 10: credit_type, rate_of_interest, property_value, lump_sum_payment, income, co.applicant_credit_type, submission_of_application, loan_type, business_or_commercial, and Gender.

Adjust of tunning parameters: The "randomForest" function in the "randomForest" R package supports the Balanced Random Forest by specifying the "strata" and the "sampsize" parameters to enable the balanced bootstrapping resampling.

Strata: A (factor) variable that is used for stratified sampling.

Sampsize: Size(s) of sample to draw. For classification, if sampsize is a vector of the length the number of strata, then sampling is stratified by strata, and the elements of sampsize indicate the numbers to be drawn from the strata.

The reason for this model is that: If we run with only powerful predictors, the chance that model selects those variables is high, hence the model is more precise. For tunning parameters, we change them to enable the balanced bootstrapping resampling, and hence model could overcome the imbalanced classes.

The code is displayed as below. We only change the number of predictors and add two tunning figures for the model.

```
# Run model
set.seed(300)
rf.feature = randomForest(Status~credit_type + rate_of_interest+property_value+
                            lump_sum_payment+income+co.applicant_credit_type+
                            submission_of_application+ loan_type +
                            business_or_commercial+Gender,
                          data = training_set,
                          ntree = 100,
                          mtry = 4,
                          strata = training_set$Status,
                          sampsize = rep(sum(training_set$Status==1),2))
```

***Figure 8.7**: Execution code for Balanced Random Forest with selected features*

Then the authors apply the model on this newly created training set. Result on training and confusion matrix are shown below:

```
> rf.feature

Call:
 randomForest(formula = Status ~ credit_type + rate_of_inter
est +      property_value + lump_sum_payment + income + co.a
pplicant_credit_type +      submission_of_application + loan
_type + business_or_commercial +      Gender, data = trainin
g_set, ntree = 100, mtry = 4, strata = training_set$Status,
     sampsize = rep(sum(training_set$Status == 1), 2))
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 4

        OOB estimate of  error rate: 10%
Confusion matrix:
      0     1  class.error
0 71055 10962 0.1336552178
1    26 27825 0.0009335392
```

For this model, the performance of model on training set has improved significantly on error rate of Default. The overall OOB rate is 10%, which is still better than the original mode.

Even though, authors only set 100 trees, the default rate drops significantly. Within the

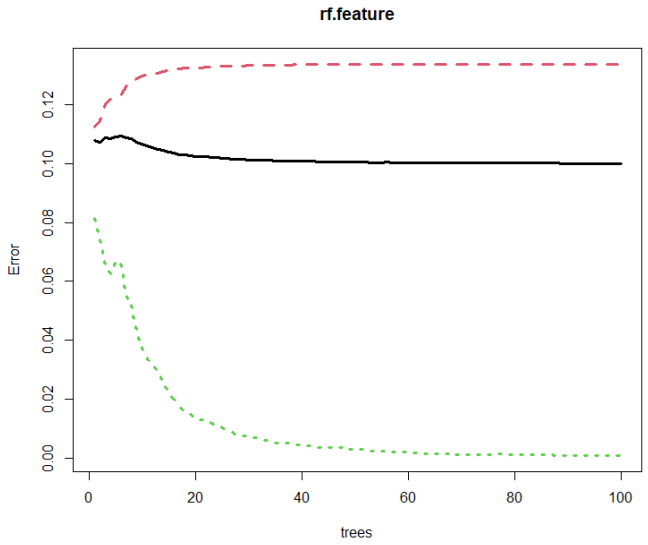


***Figure 8.8**: Model result and plot on training set*

```
> confusionMatrix(t(rf.feature.matrix),posi
Confusion Matrix and Statistics

                    Actual
rf.feature.pred.test     0     1
                   0 17756     0
                   1  2778  6934

               Accuracy : 0.8989
                 95% CI : (0.8952, 0.9024)
    No Information Rate : 0.7476
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7634

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 1.0000
            Specificity : 0.8647
         Pos Pred Value : 0.7140
         Neg Pred Value : 1.0000
             Prevalence : 0.2524
         Detection Rate : 0.2524
   Detection Prevalence : 0.3536
      Balanced Accuracy : 0.9324

       'Positive' Class : 1
```

***Figure 8.9**: Confusion matrix on test set*

training set, only 26 wrong predictions for Default class, which accounts for les than 0.1%.

Apply model to predict on test set, we have the confusion matrix as beside **Figure of 8.9**.

Surprisingly, there is no FN prediction in test set, which makes Sensitivity achieves 100%.

Overall accuracy is dropped but it is expected. However, it still remains excellent, at roughly 90%.

Conclusion: The Balanced Random Forest performs effectively on both training and test sets. It improves substantially the prediction for TP with 100% Sensitivity.

e. Final Comparison

Please consult comparison table below for conclusion.

Model	Accuracy	Sensitivity	FN	F1	AUC
Rf.full	**92.22%**	83.54%	1141	0.8442	0.8935
Rf.Oversample	91.72%	93.65%	440	**0.8510**	0.9236
Rf.feature	89.89%	**100.00%**	**0**	0.8331	**0.9324**

Table 7: *Model comparison*

In term of accuracy, the original model performs the best. Conversely, when taking into our target, the last model of Balanced Random Forest with feature selection outperforms the rest with the highest Sensitivity, the lowest FN, and the best overall AUC.

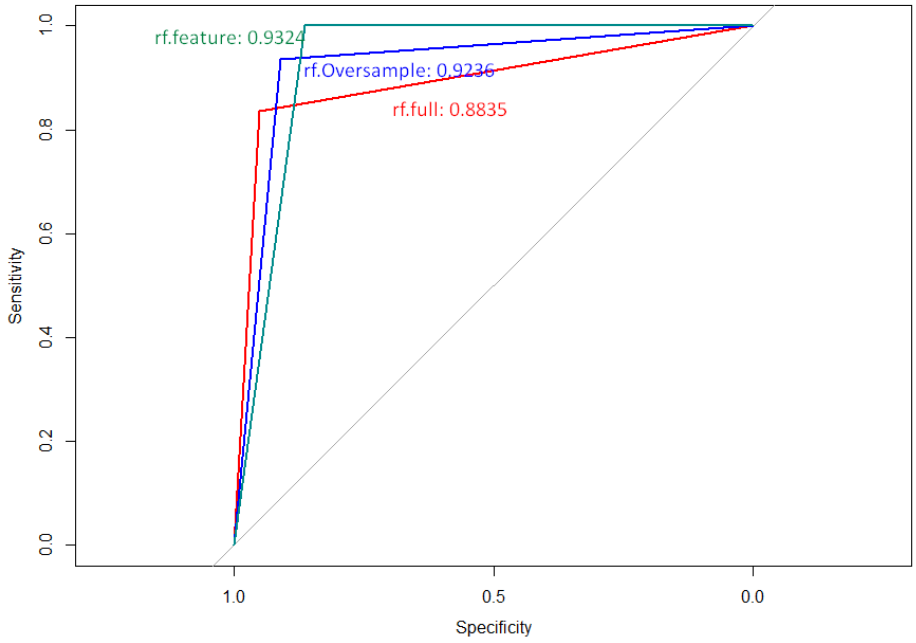


Figure 8.10*: ROC performance of three Random Forests*

Overall, Balanced Random Forest with Feature Selection is a great candidate for this data set with outstanding performance among three models.

9. All model comparison and selection

a. Model selection by performance metrics

The traditional approach of model selection is based on performance metrics, which we have acquired for classification such as ROC, AUC, F1, Accuracy, as well as Sensitivity. In this part, we will summarize all results for model comparison and selection. In the heat map of **Figure 9.1**, we highlight the measures by color, in which **Red** is the least desire, and **Green** is the expected desire.

Type	Model	Accuracy	Sensitivity	FN	F1	AUC
Logistic Regression	log.stepwise	86.33%	48.24%	3589	0.640	0.845
Logistic Regression	log.weighted	81.70%	64.10%	2489	0.639	0.846
Decision Tree	tree.prob.prune	91.28%	80.14%	1377	0.822	0.876
Decision Tree	tree.lossmatrix.prune	89.88%	**100.00%**	**0**	0.833	**0.932**
XGBoost	xgb.full	91.09%	69.41%	2108	0.850	0.839
XGBoost	xgb.robust	92.63%	86.04%	962	0.874	0.904
Random Forest	rf.full	**92.22%**	83.54%	1141	0.844	0.894
Random Forest	rf.Oversample	91.72%	93.65%	440	**0.851**	0.924
Random Forest	rf.feature	89.89%	**100.00%**	**0**	0.833	**0.932**

***Figure 9.1**: Heat map for model comparison*

In terms of performance metrics, ***overall best performance is Random Forest type***, while ***the worst performance is for Logistic Regression***.

The highest accuracy belongs to Random Forest default model while ***the most appropriate model in our case of imbalanced data (Highest Sensitivity and lowest FN) belongs to Balanced Random Forest with Feature Selection***.

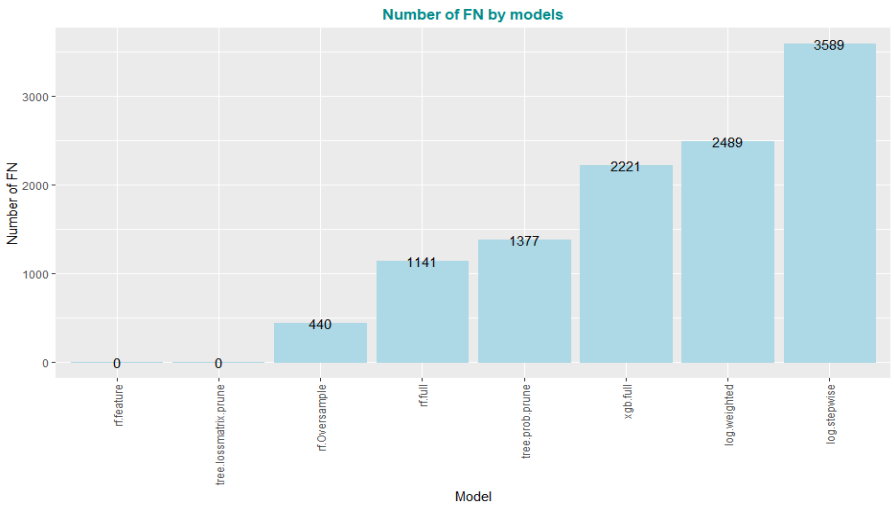


***Figure 9.2**: Number of FN by each model in test set*

We can also refer to **Figure 9.2** for model selection.

b. Model selection by Domain knowledge using Bad rate strategy

There is another alternative for model comparing and then selecting the most appropriate one, which relies on Domain knowledge.

Until now, we have set cutoffs based on best guesses or our gut feeling in order to make accuracy matrices for each of the models used. The choice of a cutoff is important, however, as this changes the validation metrics and hence affects the final model selection.

In order to select more appropriate models, base on the probability, we can construct a table with acceptance rates, corresponding cutoff values, and the percentage of "bad loans" that are accepted. This is a useful tool for a bank because banks can adapt their acceptance rate depending on the percentage of bad loans they can allow in their loan portfolio.

In this section, author develop a function to select the best model with the input is acceptance rate. **The chosen models are the ones with the lowest bad loans rate**. But first, we plot several strategy curves, which illustrate the bad rate associated with each acceptance rate.

Refering to **Figure 9.3,** we can see that ***if acceptance rate is low, we should choose Decision Tree and Random Forest model as their bad rate values are also low***. While Logistic regressions depict a steep line while acceptance rate increases.

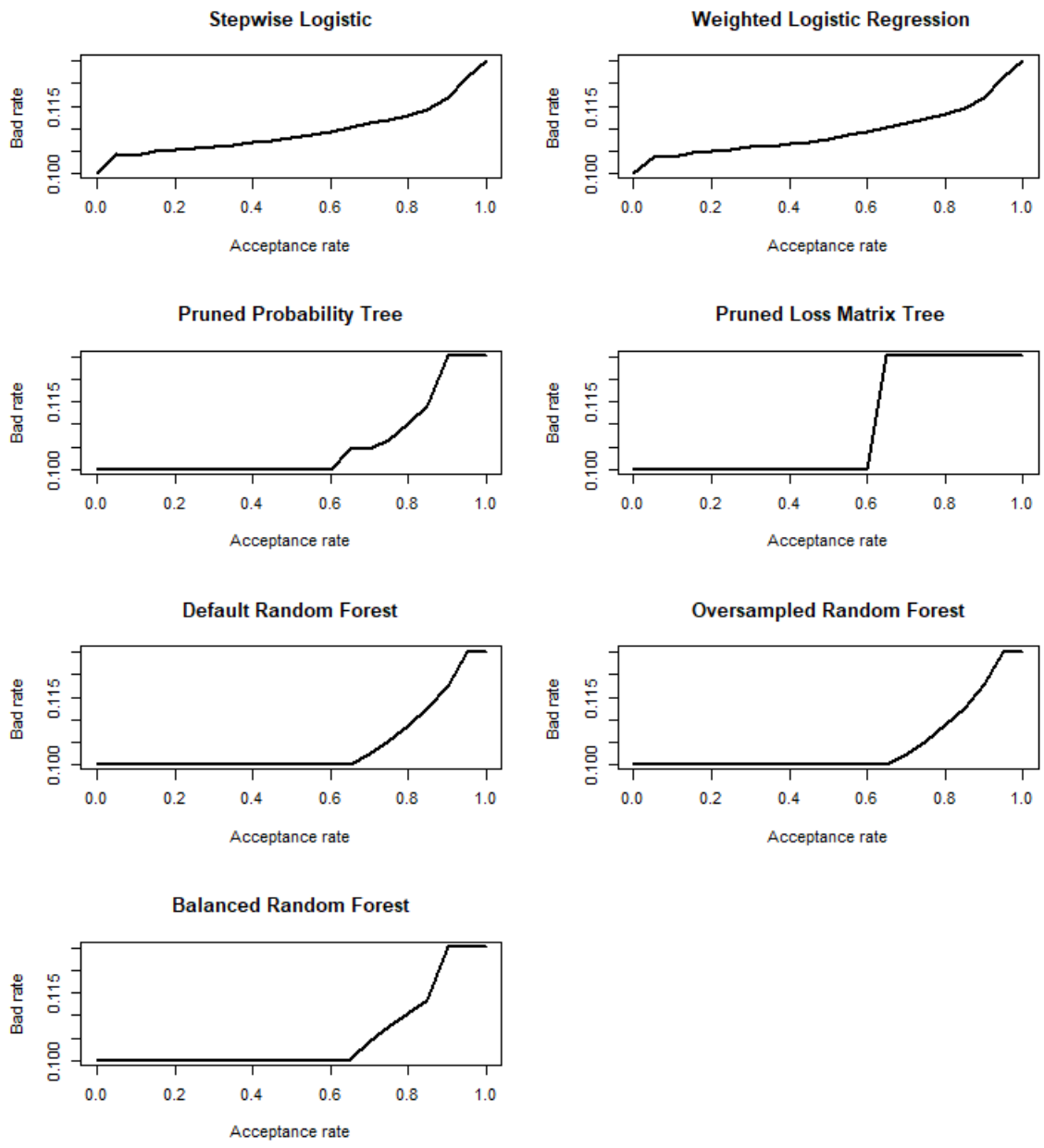


Figure 9.3: *Strategy curves for 7 models*

By this plot, we could test each scenario to choose which model we should use to minimize that bad rate. We develop a function to choose the lowest bad rate model with input of acceptance rate as below:

```r
Model_Selection <- function(rate){
  index = 2+ ((1-rate)/0.05)
result.table = rbind(strategy.log.stepwise$table[index,c(1,3)],
              strategy.log.weighted$table[index,c(1,3)],
              strategy.tree.prob.prune$table[index,c(1,3)],
              strategy.tree.lossmatrix.prune$table[index,c(1,3)],
              strategy.rf.Oversample$table[index,c(1,3)],
              strategy.rf.feature$table[index,c(1,3)],
              strategy.rf.full$table[index,c(1,3)])
result.table <- as.data.frame(result.table)
result.table <- result.table %>% mutate(Model = c("log.stepwise","log.weighted","tree.prob.prune",
                                                  "tree.lossmatrix.prune",
                                                  'rf.Oversample',"rf.feature","rf.full")) %>%
  arrange(bad_rate)

return(result.table)}
```

Figure 9.4: *Code execution for model selection*

Now, we test 3 scenarios to see which model we should choose:

```
> Model_Selection(0.9)
  accept_rate bad_rate                 Model
1         0.9   0.1169           log.stepwise
2         0.9   0.1169           log.weighted
3         0.9   0.1176                rf.full
4         0.9   0.1178          rf.Oversample
5         0.9   0.1252         tree.prob.prune
6         0.9   0.1252 tree.lossmatrix.prune
7         0.9   0.1252             rf.feature
>
> Model_Selection(0.8)
  accept_rate bad_rate                 Model
1         0.8   0.1087                rf.full
2         0.8   0.1089          rf.Oversample
3         0.8   0.1100         tree.prob.prune
4         0.8   0.1104             rf.feature
5         0.8   0.1130           log.stepwise
6         0.8   0.1132           log.weighted
7         0.8   0.1252 tree.lossmatrix.prune
>
> Model_Selection(0.5)
  accept_rate bad_rate                 Model
1        0.45   0.1000         tree.prob.prune
2        0.45   0.1000 tree.lossmatrix.prune
3        0.45   0.1000          rf.Oversample
4        0.45   0.1000             rf.feature
5        0.45   0.1000                rf.full
6        0.45   0.1071           log.weighted
7        0.45   0.1074           log.stepwise
>
```

If acceptance rate is **90%**, we should choose ***either two logistic regressions*** as they yield the lowest acceptance rate.

While in case of **80%**, we should pick the ***Original Random Forest***.

The last scenario of **50%** acceptance rate, we can choose ***any Decision Tree models or Random Forest models***.

Assuming we have the acceptance rate less than 80%, we the most appropriate model here is **Balanced Random Forest with Feature selection**.

10. Insight suggestions from the work

The ultimate goal of data science is to help businesses grow by translating models and engineer languages to business performance indexes. In this report, after building the best overall model for future prediction, we develop several insightful suggestions for business as well.

a. What to collect when bank institutions constraints in resources?

Sometimes, collecting all 34 variables in data set are time-consuming while some of them are not useful for prediction. Therefore, in case the businesses just need to collect insightful predictors with explanatory powers only. Here are top 5 for Loan default***: Credit type, Rate of interest, Property value, Lump sum payment*** *and* ***Income,*** based on Information value in **Figure 4.1** conducted earlier in this report.

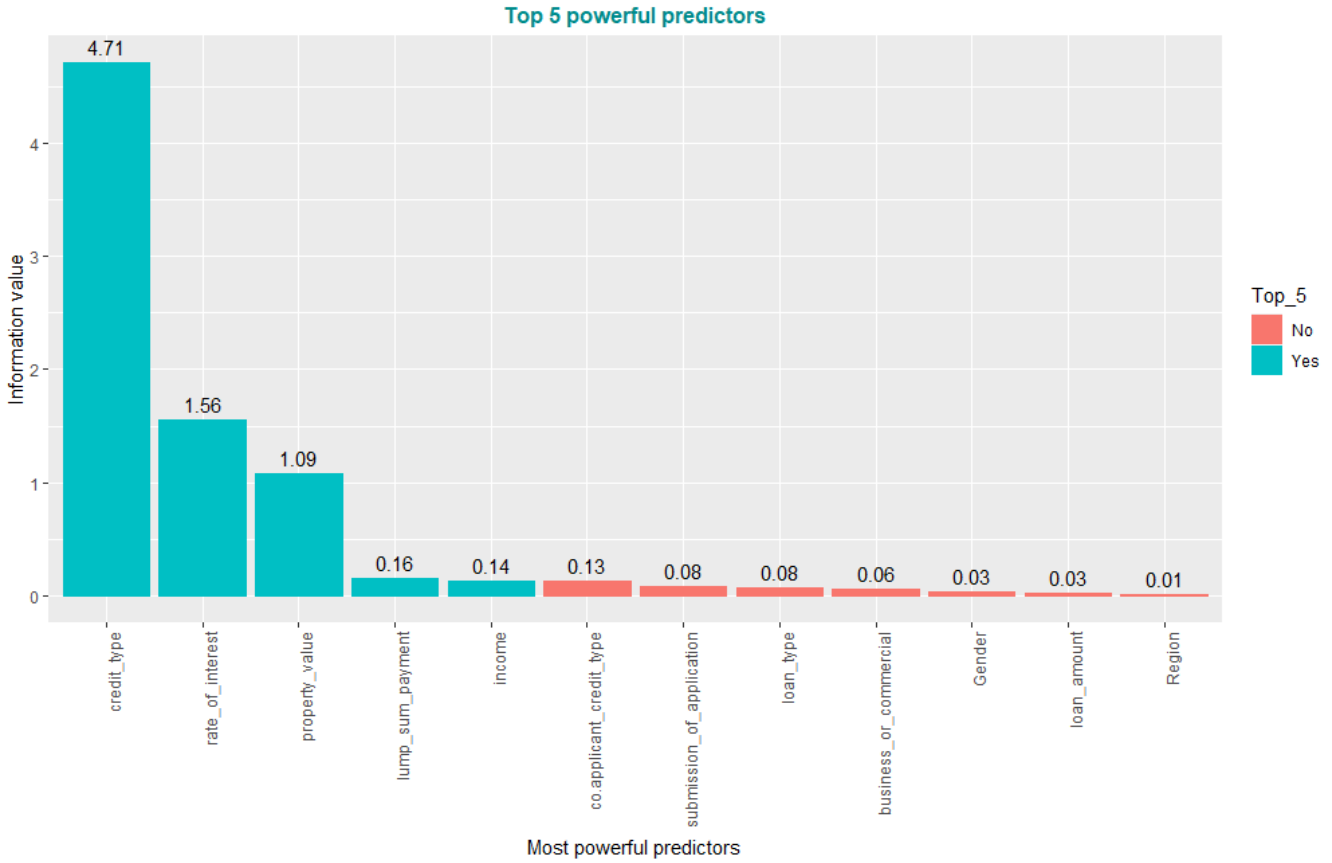


Figure 10.1*: Top 5 powerful predictors need to collect in case of resource constraints*

Therefore, in case banks do not have enough data and limit in resources, just collect these variables for prediction using Balanced Random Forest.

b. What the traits of default customer look like?

In order to draw insights to picture the characteristics of default customers, we will conduct **contingency table analysis and visualization for the most important predictors.** Please note that contingency analysis is more appropriate for categorical variables.

The trait of Default customer will have the following properties:

- **If credit type of customer is EQUI, they are most likely defaults:**

df$credit_type	df$Status 0	1	Row Total
CIB	40529	7610	48139
	0.842	0.158	0.324
CRIF	36769	7126	43895
	0.838	0.162	0.296
EQUI	1	15085	15086
	0.000	1.000	0.102
EXP	34706	6603	41309
	0.840	0.160	0.278
Column Total	112005	36424	148429

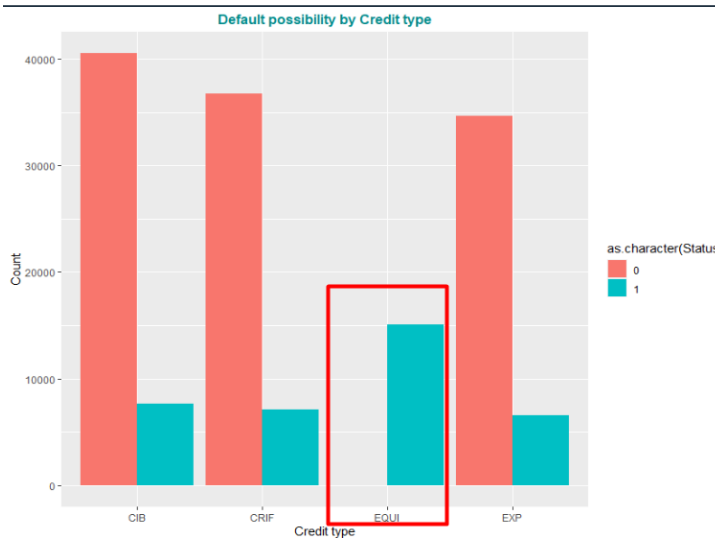


***Figure 10.2**: Default possibility by Credit type*

- **If property values of borrowers are below 425.000 USD, there is a high chance that they are defaults:** (Both median and mean property values of defaults are lower those of non-default customer, and the middle point is around 425.000 USD, therefore authors choose this figure for estimation and best guess. However, since this is numerical data, just take the insights for reference).

```
# A tibble: 2 x 3
  Status Mean_Property_Value Median_Property_Value
   <int>               <dbl>                 <dbl>
1      0             492688.                428000
2      1             425774.                418000
```

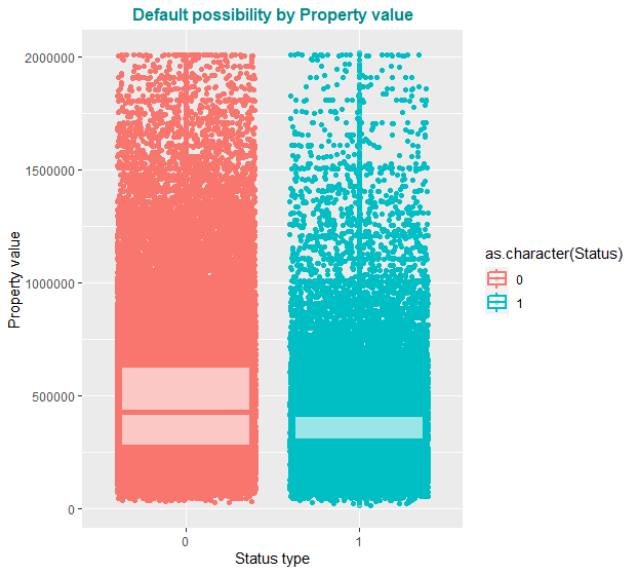


***Figure 10.3**: Default possibility by Property values*

- **If borrowers choose lump sum payment (This means they require banks allow them to pay back at once for the total amount), they are more likely to default:**

df$lump_sum_payment	df$Status 0	1	Row Total
lpsm	688	2491	3179
	0.216	0.784	0.023
not_lpsm	101863	32294	134157
	0.759	0.241	0.977
Column Total	102551	34785	137336

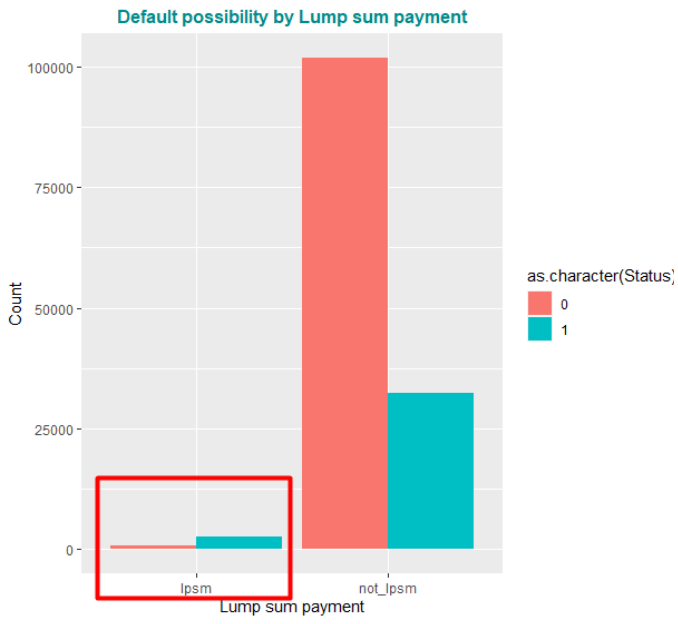


***Figure 10.4**: Default possibility by Lump sum payment option*

- **If borrower's incomes are less than 5000 USD, bankers should consider them high chance to be defaults:**

	Status	Mean_Income	Median_Income
	<int>	<dbl>	<dbl>
1	0	6816.	5940
2	1	5733.	4800

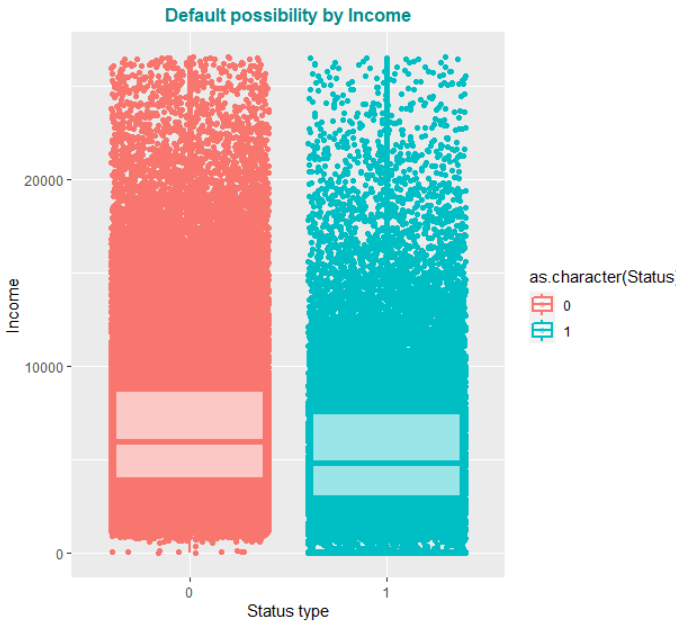


***Figure 10.5**: Default possibility by Income*

- **If customers borrow moneys for the purpose of Business or commercial related, they could be defaults more than for those who borrow for non-commercial purposes:**

df$business_or_commercial	df$Status 0	1	Row Total
b/c	10656	6491	17147
	0.621	0.379	0.125
nob/c	91895	28294	120189
	0.765	0.235	0.875
Column Total	102551	34785	137336

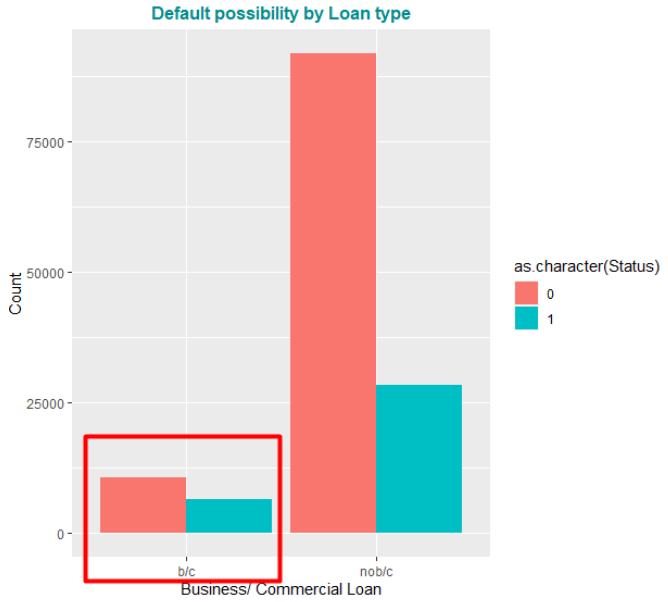


***Figure 10.6**: Default possibility by Loan Purpose*

- **If customers do not declare their genders, they could be defaults more than for those who are willing to declare:**

df$Gender	df$Status 0	1	Row Total
Female	19512	6667	26179
	0.745	0.255	0.191
Joint	30037	7521	37558
	0.800	0.200	0.273
Male	28575	10550	39125
	0.730	0.270	0.285
Sex Not Available	24427	10047	34474
	0.709	0.291	0.251
Column Total	102551	34785	137336

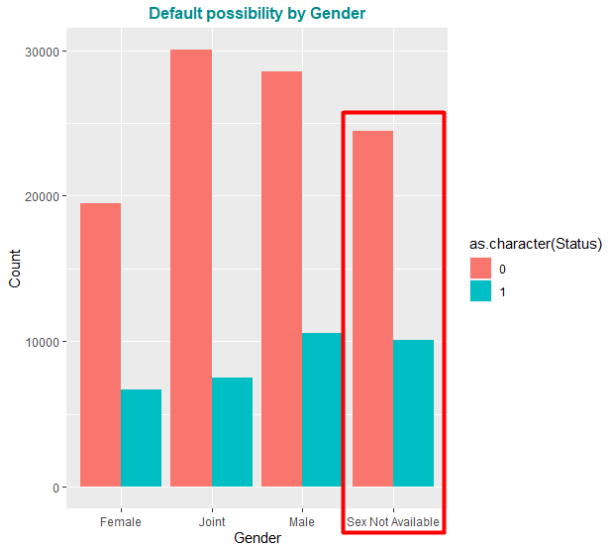


***Figure 10.7**: Default possibility by Gender*

- **If customers choose option of installment for their loan, they are more likely to default than those who do not choose this option:**

df$submission_of_application	df$Status 0	1	Row Total
not_inst	39308	8721	48029
	0.818	0.182	0.350
to_inst	63243	26064	89307
	0.708	0.292	0.650
Column Total	102551	34785	137336

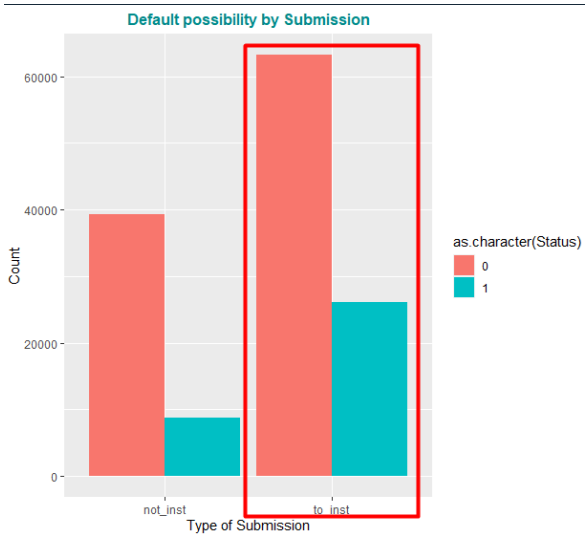


***Figure 10.8**: Default possibility by type of Submission*

In summary: **The traits of default customers are**: Who have credit type of EQUI, whose property values are below 425.000 USD, who require banks allow them to pay back at once for the total amount (choose lump sum payment), whose incomes are less than 5000 USD, who borrow moneys for the purpose of Business or commercial related, who are not willing to declare their genders and who choose option of installment.

C. CONCLUSION

In this project, we have demonstrated various methods for 4 types of algorithms to tackle the imbalanced data set. Below is the conclusion of project:

Final model:

Taking account 2 methods of Model selection, which are Performance metrics and Domain knowledge, we propose to choose **Balanced Random Forest with Feature Selection** for default prediction as this model yields 90% Accuracy, 100% Sensitivity, 0 FN and AUC up to 93.2%. In addition, when we test different scenarios of acceptance rates, it is also the model offering the lowest bad rates among all.

Business Suggestions:

What to do when lack of resources: In case the financial businesses want to predict default customers, but they are facing resource and constraints, they just need to collect data for several insightful predictors with the most explanatory powers only. Here are top 5 variables that business should consider: ***Credit type, Rate of interest, Property value, Lump sum payment*** *and* ***Income.*** And then using our chosen Random Forest, we can predict the target effectively.

The traits of default customers: If bankers want to detect the characteristics of default customer, these are the traits that we suggest: They are who have credit type of EQUI, whose property values are below 425.000 USD, who require banks allow them to pay back at once for the total amount (choose lump sum payment), whose incomes are less than 5000 USD, who borrow moneys for the purpose of Business or commercial related, who are not willing to declare their genders and who choose option of installment.

As the main purpose of this project is also about practicing, we want to highlight several key highlights as well.

1. For data set with many variables, we need to **conduct feature selection to reduce the dimensionality**. Some approaches could be skewness analysis, using domain knowledge, justifying the missing value proportion, and so on.

2. **Accuracy is not always the key metric** when evaluating model performance. Instead, we should take a look at the importance of classification. In our case, the objective is to reduce Error type 2, in which the number of FN needs to be minimized and we also try to uplift the Sensitivity (Recall).

3. **There are various solutions to tackle imbalanced data set**. Solutions are applied from sampling, modeling to tuning and changing thresholds. Mastering these techniques would help us for many problems with imbalanced class such as: Fraud Detection, Cancer prediction, Click rate,…

4. **Overall schemes and algorithms for Finance and Banking Industry**:

Feature selection: Among many methods of Feature selection such as Information Gain, XGBoost Feature Selection, Discriminant Analysis,…, we suggest to use Weight of Evidence (WOE) and Information Value (VI) for this problem as this method is highly applicable for binary classification and for finance industry. This method is developed by Claude Shannon in 1948, and Larsen (2015) also stated that "*WOE and IV have been used extensively in the credit risk world for several decades, and the underlying theory dates back to the 1950s*".

Prediction algorithm: Conversely, there is no specificly optimal algorithm for this industry, but instead we should test several algorithm, then try to improve them by mentioned methods and finally compare them by performance metrics as well as domain knowledge applications to choose the best models. That is why we have conduct many machine learning models in this research.

D . REFERENCES

Sundaramahadevan, V. (2021, August). *Credit EDA Case Study*. Retrieved from Kaggle: https://www.kaggle.com/venkatasubramanian/credit-eda-case-study

Loan Default Dataset. (2022). https://www.kaggle.com/datasets/yasserh/loan-default-dataset

Online Payments Fraud Detection Dataset. (2022). https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

KHARWAL, A. (2022 22 02). *Online Payments Fraud Detection with Machine Learning*. https://thecleverprogrammer.com/2022/02/22/online-payments-fraud-detection-with-machine-learning/

Petrova, N. *Detecting Online Fraud with Machine Learning*. Detecting Online Fraud with Machine Learning

Sidelov, P. (2021 Aug 20). *Detecting Payment Card Fraud with Machine Learning. H2O Driverless AI + Kaggle Dataset*. https://sdk.finance/detecting-payment-card-fraud-with-machine-learning-h2o-driverless-ai-kaggle-dataset/

Zoho. *Online Payment Fraud 101: What It Is, Types, and How to Prevent it*. https://www.zoho.com/books/articles/payment-fraud.html

R Notebook. https://rstudio-pubs-static.s3.amazonaws.com/409548_5dcab1f0a8b14069867428e231cfc1bb.html

DAVID, R. (2019). *Home Credit Default Risk - R*. https://www.kaggle.com/raenish/home-credit-default-risk-r

Gupta, A. (October 10, 2020). *Feature Selection Techniques in Machine Learning*. https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/

kim, H. w. (Jun 25 2018). *Simple EDA for beginners with R #1*. https://www.kaggle.com/chocozzz/simple-eda-for-beginners-with-r-1

Loss, A. (2019). *Home Credit Default Risk*. https://www.kaggle.com/c/home-credit-default-risk/discussion/58200

Matthys, C. (2018). *PREDICTING AN APPLICANT'S CAPABILITY OF REPAYING A BANK LOAN - A CASE STUDY IN DATA MINING* Universiteit Gent].

Sandy. (Nov 19 2019). *Sum by Rows in R*. https://stackoverflow.com/questions/29639343/sum-by-rows-in-r/58925819#58925819?newreg=f5d4f7efb50d473996389cbeea19e980

Statisticsglobe. *Get Column Index in Data Frame by Variable Name in R (2 Examples)*. https://statisticsglobe.com/get-column-index-in-data-frame-by-variable-name-in-r

Table Function In R – Frequency Table in R & Cross Table in R. https://www.datasciencemadesimple.com/table-function-in-r/

Contingency Table in R (5 Examples). https://statisticsglobe.com/contingency-table-r

Binning with decision trees | SQL Pete. (n.d.). Retrieved April 26, 2022, from https://sqlpete.wordpress.com/2017/01/23/binning-with-decision-trees/

Creating, Validating and Pruning the Decision Tree in R | Edureka Blog. (2015, January 29). Edureka. https://www.edureka.co/blog/implementation-of-decision-tree/

Decision Tree Algorithm, Explained. (n.d.). KDnuggets. Retrieved April 26, 2022, from https://www.kdnuggets.com/decision-tree-algorithm-explained.html/

Decision Trees and Pruning in R - DZone AI. (n.d.). Dzone.Com. Retrieved April 26, 2022, from https://dzone.com/articles/decision-trees-and-pruning-in-r

What is a Decision Tree Diagram. (n.d.). Lucidchart. Retrieved April 26, 2022, from https://www.lucidchart.com/pages/decision-tree

z_ai. (2021, September 26). Decision Trees Explained. Medium. https://towardsdatascience.com/decision-trees-explained-3ec41632ceb6

(n.d). (2020). *Down-Sampling Using Random Forests*. http://appliedpredictivemodeling.com/blog/2013/12/8/28rmc2lv96h8fw8700zm4nl50busep

Haldar, S. (2019 Jul 23). *How does class_weight work in Decision Tree*. https://datascience.stackexchange.com/questions/56250/how-does-class-weight-work-in-decision-tree/56260#56260

Hamalcikova, K. (2021 Jan 6). *Random Forest for imbalanced dataset: example with avalanches in French Alps*. https://towardsdatascience.com/random-forest-for-imbalanced-dataset-example-with-avalanches-in-french-alps-77ffa582f68b

Li, E. C. (2017 May 21). *Random Forrest Analysis for Bank Default*. https://rpubs.com/cheverne/randomforest_bankdefault

Maklin, C. (2019 Jul 30). *Random Forest In R*. https://towardsdatascience.com/random-forest-in-r-f66adf80ec9

mhdella. (2016 Oct 27). Is Random Forest a good option for unbalanced data Classification? In.

MiksL. (2015 Sep 2). *Implementing Balanced Random Forest (BRF) in R using RandomForests*. https://stats.stackexchange.com/questions/171380/implementing-balanced-random-forest-brf-in-r-using-randomforests

Sachdeva, J. (2020 Jan 31). *Minsplit and Minbucket*. https://medium.com/talking-with-data/minsplit-and-minbucket-a49ff56026c8

J. Brownlee. (Feb 2020). B*agging and Random Forest for Imbalanced Classification* https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/

J. Brownlee. (Jan 2020). *How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification* https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/

J. Huneycutt. (May 2018). *Implementing a Random Forest Classification Model in Python* https://medium.com/@hjhuney/implementing-a-random-forest-classification-model-in-python-583891c99652

Ch. Yau. (2017). *Growing RForest — 97% Recall and 100% Precision* https://www.kaggle.com/palmbook/growing-rforest-97-recall-and-100-precision

M. Vernay, and M. Lafaysse and P. Hagenmuller and R. Nheili and D. Verfaillie and S. Morin. (2019). *The S2M meteorological and snow cover reanalysis in the French mountainous areas (1958 — present)* https://en.aeris-data.fr/metadata/?865730e8-edeb-4c6b-ae58-80f95166509b

LARSEN, K. (2015 August 13). *Data Exploration with Weight of Evidence and Information Value in R.* https://multithreaded.stitchfix.com/blog/2015/08/13/weight-of-evidence/