Home Credit Capstone Project

</> Code

AUTHOR
Kleyton Polzonoff

PUBLISHED
December 8, 2024

```
library(lubridate); library(ggplot2); library(tidyverse); library(rmarkdown); library(rpact); libr
#seed for the entire document
seed<-500
```

1. Introduction

Access to credit is crucial for development globally, enabling individuals and businesses to leverage financial resources for growth. However, many people face barriers to credit access due to inadequate information, such as risk agency scores, making them susceptible to predatory lending and severe financial consequences. Home Credit seeks to address this issue by providing secure credit access to underserved populations. This project aims to develop a default prediction model using advanced analytics, aligning with Home Credit's goal of fostering financial inclusion while ensuring the company's financial stability.

---

2. Data set

Data set structure and description

```
# Set the working directory
mydir <- getwd()
setwd(mydir)

# Loading train data
application_train <- read.csv(file = "C:\\Home_Credit_Project\\Home-Credit-Project\\home-credit-de

# Loading test data
application_test <- read.csv(file = "C:\\Home_Credit_Project\\Home-Credit-Project\\home-credit-def

# Display the first 6 columns of the dataset
application_train %>%
  head() %>%
  knitr::kable()
```

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHIL
100002	1	Cash loans	M	N	Y	
100003	0	Cash loans	F	N	N	
100004	0	Revolving loans	M	Y	Y	

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHIL
100006	0	Cash loans	F	N	Y	
100007	0	Cash loans	M	N	Y	
100008	0	Cash loans	M	N	Y	

```
# Number of variables and types
variable_types <- sapply(application_train, class)
type_counts <- table(variable_types)

knitr::kable(type_counts)
```

variable_types	Freq
character	16
integer	41
numeric	65

This data set contains information about consumers who have obtained credit from Home Credit. It includes demographic and business details to support the credit granting process.

The application_train table will serve as the starting point for creating predictive models. It comprises 122 columns and 307,511 rows, originally categorized into three types: character, numeric, and integer.

The descriptions of each variable are detailed in a CSV file, which is quite extensive and, therefore, not included in this document. To access these descriptions, please visit: https://www.kaggle.com/c/home-credit-default-risk/data.

---

3. Exploratory Data Analysis (EDA)

To understand the data in the application_train table and to identify any potential errors or anomalies, a series of analyses will be conducted. Additionally, this phase aims to explore the characteristics and relationships between the target variable and other variables, which will help in developing a predictive model aligned with the company's principles, strategies, and objectives.

To make this document more understandable and straightforward, only the key analyses will be presented here. However, some code used for these analyses may be included even if their results are not displayed.

3.1 Target variable: Default

The main objective of this project is to predict the binary outcome of the TARGET variable, which indicates loan default. The data set has a notable imbalance, with only about 8% of the observations corresponding to clients who are in default.

```
# Percentages for the target variable
application_train %>%
  group_by(TARGET) %>%
  summarise(customers = n(),
            avg_loan = mean(AMT_CREDIT),
            min = min(AMT_CREDIT),
            max = max(AMT_CREDIT)) %>%
  mutate(Percent. = round(((customers/sum(customers))*100),1))%>%
  knitr::kable()
```

TARGET	customers	avg_loan	min	max	Percent.
0	282686	602648.3	45000	4050000	91.9
1	24825	557778.5	45000	4027680	8.1

3.2 Outliers and Anomalies

Numeric Values

In this analysis, extreme values will be examined to identify and address outliers that may clearly represent errors. Additionally, the logical integrity of the numeric values will be assessed to ensure consistency and accuracy within the data set.

After reviewing the numeric and integer variables (results not shown), it was determined which variables are relevant to show their summary at this time due to anomalies in their values.

```
# Select specified columns
selected_columns <- application_train[, c("AMT_INCOME_TOTAL",
                                          "AMT_CREDIT",
                                          "AMT_ANNUITY",
                                          "AMT_GOODS_PRICE",
                                          "DAYS_BIRTH",
                                          "DAYS_EMPLOYED")]

# Generate statistical summary
summary(selected_columns)
```

```
AMT_INCOME_TOTAL      AMT_CREDIT       AMT_ANNUITY     AMT_GOODS_PRICE
Min.   :    25650   Min.   :  45000   Min.   :  1616   Min.   :  40500
1st Qu.:   112500   1st Qu.: 270000   1st Qu.: 16524   1st Qu.: 238500
Median :   147150   Median : 513531   Median : 24903   Median : 450000
Mean   :   168798   Mean   : 599026   Mean   : 27109   Mean   : 538396
3rd Qu.:   202500   3rd Qu.: 808650   3rd Qu.: 34596   3rd Qu.: 679500
```

```
Max.   :117000000   Max.   :4050000   Max.   :258026   Max.   :4050000
                                      NA's   :12       NA's   :278
  DAYS_BIRTH     DAYS_EMPLOYED
Min.   :-25229   Min.   :-17912
1st Qu.:-19682   1st Qu.: -2760
Median :-15750   Median : -1213
Mean   :-16037   Mean   : 63815
3rd Qu.:-12413   3rd Qu.:  -289
Max.   : -7489   Max.   :365243
```

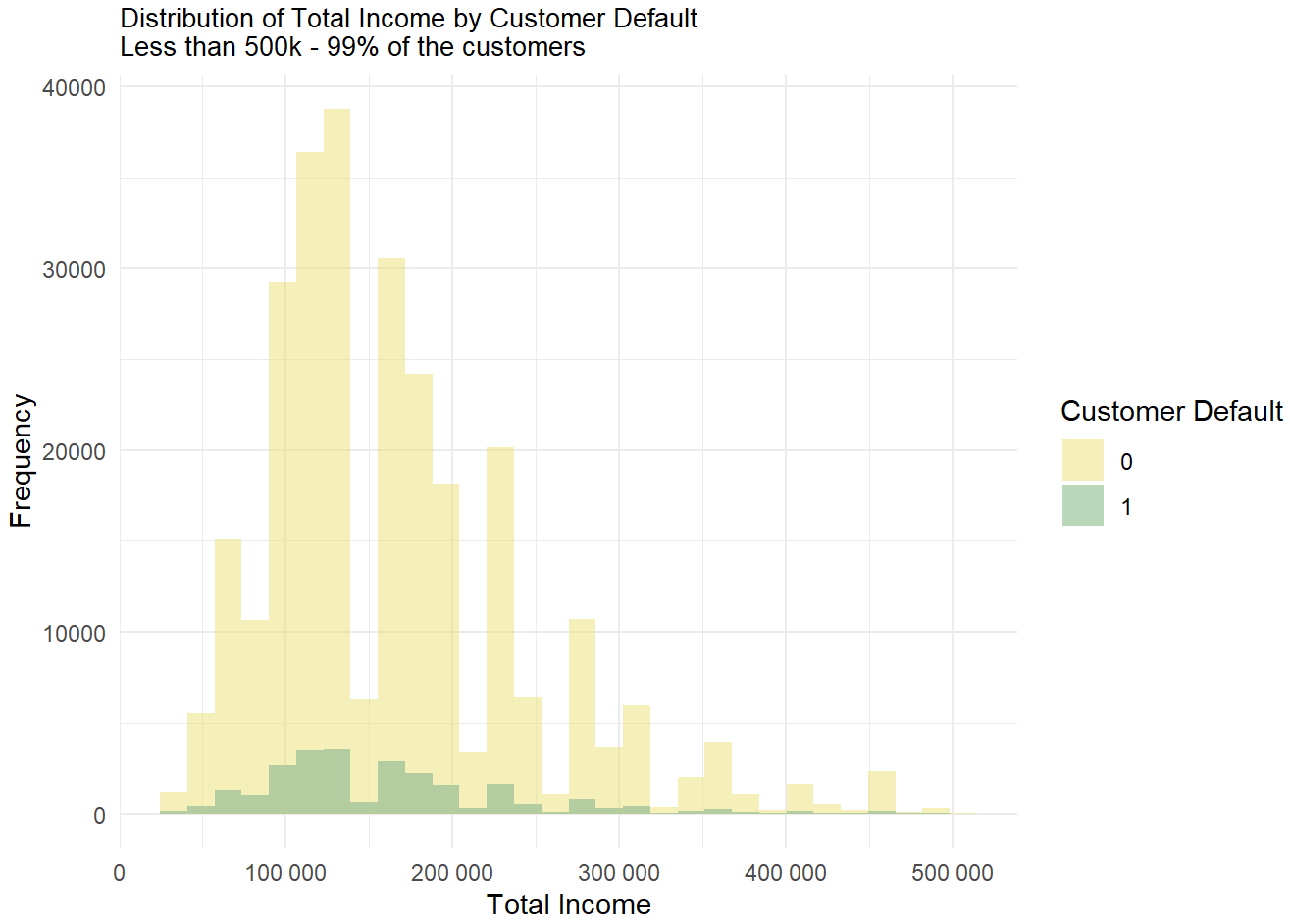
It is possible to observe extreme values and missing data (NAs) in the data set. The date-related variables are counted retroactively from the application date, so to accurately view the applicants' ages, these variables will be transformed for better understanding.

For AMT_ANNUITY, none of the 12 missing values are in the default group (target = 1), so I will impute them using the median of the observed values. Similarly, for AMT_GOODS_PRICE, with 278 missing values (21 in the default group), I will apply the same imputation method.

Income

For AMT_INCOME_TOTAL, five individuals reported incomes exceeding 5 million, with one reporting 117 million. These extreme outliers are far from the variable's median of $147,150 and are not the main focus of the company's business plan. Additionally, 99% of the individuals have incomes lower than $500,000. Therefore, these five observations will be removed, and the transformation process will also be applied to the application_test data frame later.

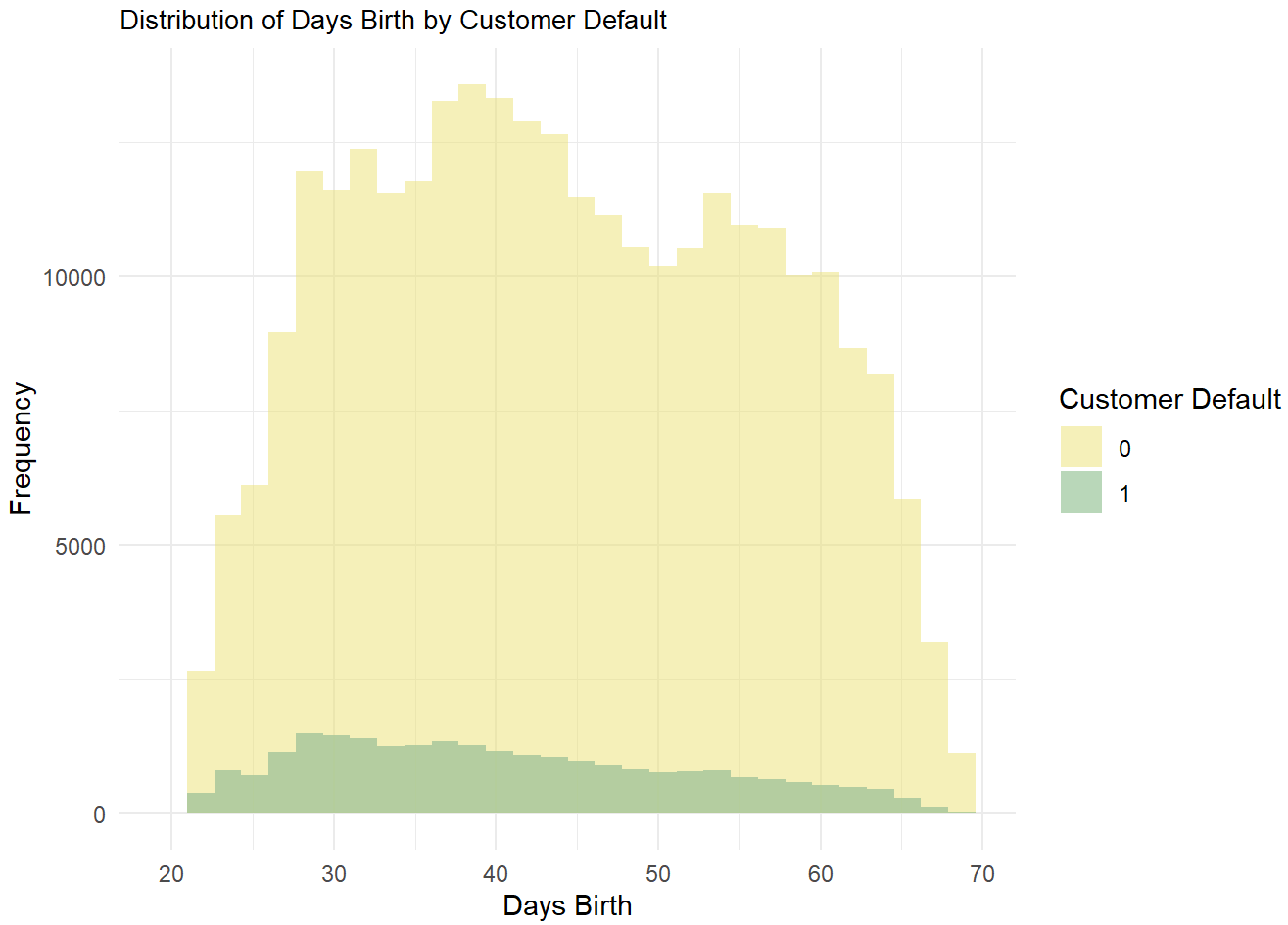
```r
# Create the clean data set
app_train_clean <- application_train

# Clean and transform
app_train_clean <- app_train_clean %>%
  filter(AMT_INCOME_TOTAL <= 5000000)

# Income plot under 500K by default
ggplot(subset(app_train_clean, AMT_INCOME_TOTAL <= 500000), aes(x = AMT_INCOME_TOTAL, fill = as.fa
  geom_histogram(alpha = 0.6, bins = 30, position = "identity") +
  labs(title = "Distribution of Total Income by Customer Default\nLess than 500k - 99% of the cust
       x = "Total Income",
       y = "Frequency") +
  scale_x_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal()+
  theme(plot.title = element_text(size = 10))
```



Customer age

Below, the DAYS_BIRTH variable will be transformed to provide a clearer understanding of the consumers' ages at the time of the loan application. The data set app_train_clean will be created to store these transformations.

```
# Transformation for better understanding ages
app_train_clean$DAYS_BIRTH <- app_train_clean$DAYS_BIRTH / -365

# Histogram of Ages by default
ggplot(app_train_clean, aes(x = DAYS_BIRTH, fill = as.factor(TARGET))) +
  geom_histogram(alpha = 0.6, bins = 30, position = "identity") +
  labs(title = "Distribution of Days Birth by Customer Default",
       x = "Days Birth",
       y = "Frequency") +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal()+
  theme(plot.title = element_text(size = 10))
```



Days Employed (Years)

The DAYS_EMPLOYED variable shows positive values, which should not occur.

```
# Visualize a table about the anomalies
app_train_clean %>%
  filter(DAYS_EMPLOYED > 0) %>%
  group_by(TARGET) %>%
  summarise(customers = n(),
            min = min(DAYS_EMPLOYED),
            max= max(DAYS_EMPLOYED)) %>%
  mutate(Percent. = round((customers / sum(customers)) * 100, 1))
```

```
# A tibble: 2 × 5
  TARGET customers    min    max Percent.
   <int>     <int>  <int>  <int>    <dbl>
1      0     52384 365243 365243     94.6
2      1      2990 365243 365243      5.4
```

Since anomalous values account for a significant portion of the data set (55,374 entries), the strategy is to mark these entries as non-numeric and create a new column, DAYS_EMPLOYED_ANOM, to flag the presence of these anomalies for each customer.

Below is the histogram of the corrected DAYS_EMPLOYED distribution. In this case, the distribution, transformed into years, is right-skewed. Most defaults are found among consumers with up to 5 years of employment.

```
# Create a new column
app_train_clean <- application_train %>%
  mutate(DAYS_EMPLOYED_ANOM = ifelse(DAYS_EMPLOYED == 365243, TRUE, FALSE))

# Replace the anomalies for NA
app_train_clean$DAYS_EMPLOYED[app_train_clean$DAYS_EMPLOYED == 365243] <- NA

# Convert DAYS_EMPLOYED to years
app_train_clean <- app_train_clean %>%
  mutate(DAYS_EMPLOYED_YEARS = DAYS_EMPLOYED / -365)

# Remove DAYS_EMPLOYED
app_train_clean <- app_train_clean %>%
  select(-DAYS_EMPLOYED)

# Histogram of Days employed by default
ggplot(app_train_clean, aes(x = DAYS_EMPLOYED_YEARS, fill = as.factor(TARGET))) +
  geom_histogram(alpha = 0.6, bins = 30, position = "identity") +
  labs(title = "Distribution of Days Employed (in Years) by Customer Default",
       x = "Years Employed",
       y = "Frequency") +
  scale_x_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal()+
  theme(plot.title = element_text(size = 10))
```

Character Variables

As seen earlier, there are 16 character variables in the data set. The values of these variables will be converted to factors at this stage. However, if the machine learning process requires other types of adjustments, these will be made later. The code below was used to inspect these variables, but its results will not be shown as the key points will be discussed in the following sections.

For the variable CODE_GENDER, men represent 34% of the loans and have a default rate of 10%, which is 42% higher compared to women, who have a default rate of 7%.

```
# Get the total number of observations
total_observations <- nrow(app_train_clean)

# Percentage of customers by gender and default
app_train_clean %>%
  group_by(CODE_GENDER, TARGET) %>%
  summarise(customers = n(), .groups = 'drop') %>%
  group_by(CODE_GENDER) %>%
  mutate(
    Percent_of_total = round((customers / total_observations) * 100, 0),
    Total_by_gender = sum(customers),
    Percent_by_gender = round((customers / Total_by_gender) * 100, 0)
  ) %>%
```

```
  ungroup() %>%
  knitr::kable()
```

CODE_GENDER	TARGET	customers	Percent_of_total	Total_by_gender	Percent_by_gender
F	0	188278	61	202448	93
F	1	14170	5	202448	7
M	0	94404	31	105059	90
M	1	10655	3	105059	10
XNA	0	4	0	4	100

The difference in default rates between consumers who own a car (FLAG_OWN_CAR) and those who do not is only 1.3% higher for those without a car. There is no difference in default rates between those who own property (FLAG_OWN_REALTY) and those who do not.

While several variables can be converted to factors to differentiate between groups, they are not necessarily ranked by importance. Many of these variables contain blank values, which will be replaced with 'unknown'.

Contract types

The vast majority (93.5%) of defaults are associated with cash loans, compared to just 6.5% for revolving loans. Additionally, the average loan amount for cash loans is more than double at $578,598, with a maximum limit three times higher at $4,027,680.

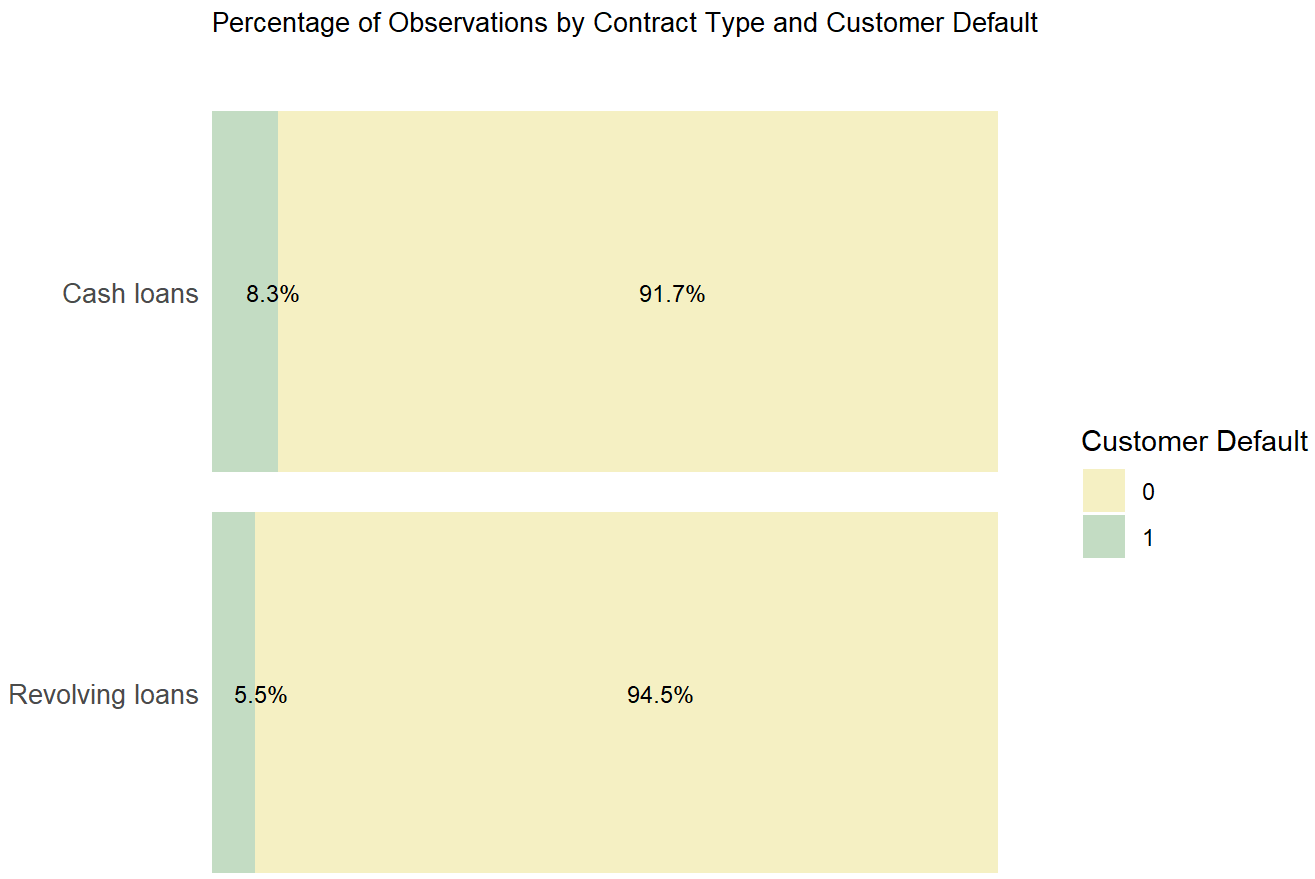
```
# Loan types summary for default
app_train_clean %>%
  filter(TARGET == "1") %>%
  group_by(NAME_CONTRACT_TYPE) %>%
  summarise(customers = n(),
            avg_loan = mean(AMT_CREDIT),
            min = min(AMT_CREDIT),
            max= max(AMT_CREDIT)) %>%
  mutate(Percent. = round((customers / sum(customers)) * 100, 1)) %>%
  knitr::kable()
```

NAME_CONTRACT_TYPE	customers	avg_loan	min	max	Percent.
Cash loans	23221	578598.8	45000	4027680	93.5
Revolving loans	1604	256365.3	135000	1350000	6.5

Relatively, cash loans have a 50% higher default rate compared to revolving loans.

```
# Summarize data by contract type and default
data_summary <- app_train_clean %>%
  group_by(NAME_CONTRACT_TYPE, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
```

```r
  group_by(NAME_CONTRACT_TYPE) %>%
  mutate(total = sum(count),
         percentage = (count / total) * 100) %>%
  ungroup()

# Extract percentage for TARGET == 0
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(NAME_CONTRACT_TYPE, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "NAME_CONTRACT_TYPE") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
  arrange(desc(order)) %>%
  mutate(NAME_CONTRACT_TYPE = factor(NAME_CONTRACT_TYPE, levels = unique(NAME_CONTRACT_TYPE)))

# Contract type by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(NAME_CONTRACT_TYPE, -order), fill = a
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Contract Type and Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

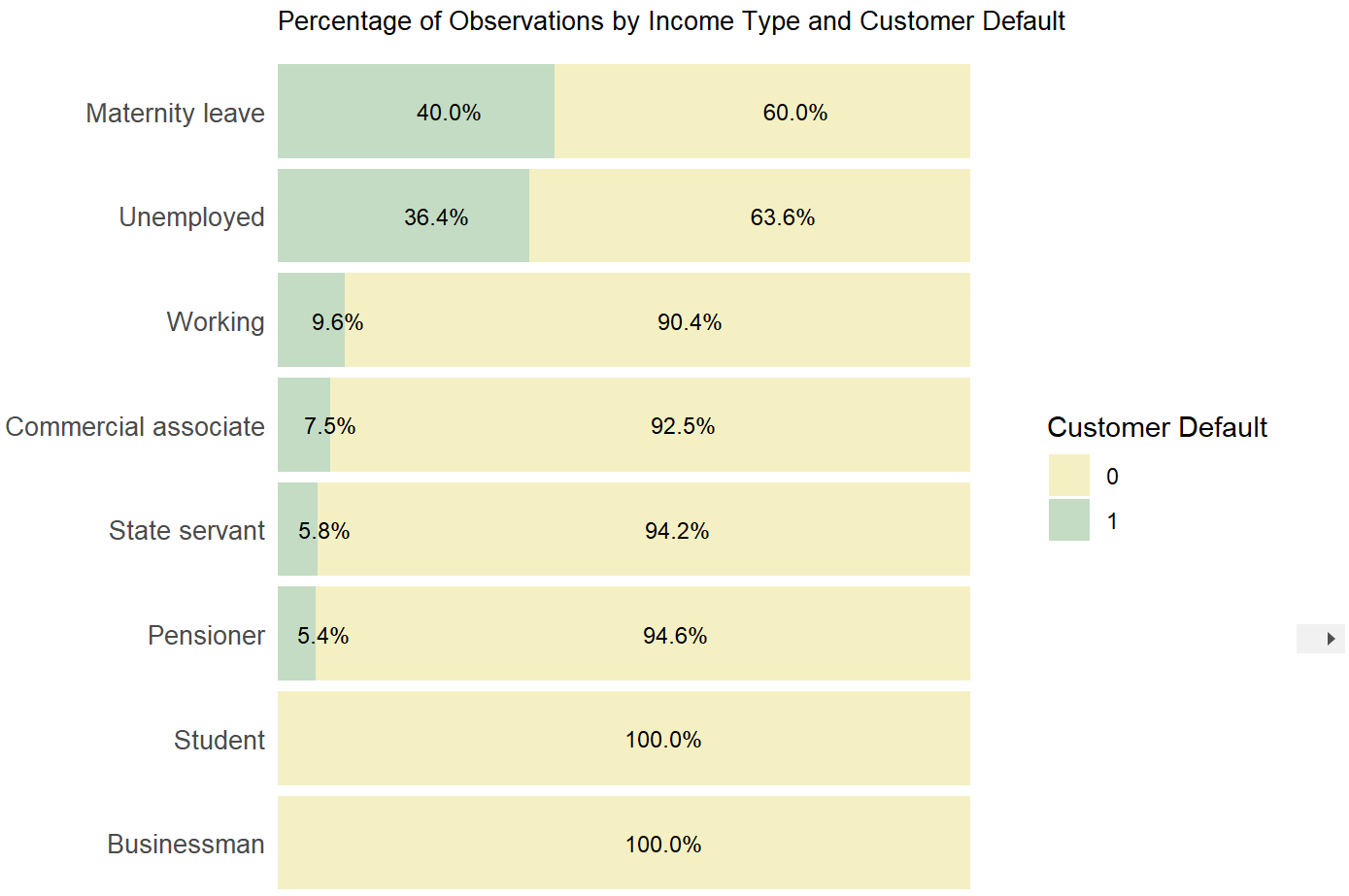


Income type

Among the income type groups, those on maternity leave and unemployed have the highest default rates, with more than 35% of loans in default for these groups. The groups of students and business people do not have any individuals with defaults.

```
# Summarize data by income type and default
data_summary <- app_train_clean %>%
  group_by(NAME_INCOME_TYPE, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
  group_by(NAME_INCOME_TYPE) %>%
  mutate(total = sum(count),  # Total count per income type
         percentage = (count / total) * 100) %>%  # Percentage per income type
  ungroup()

# Extract percentage for TARGET == 0
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(NAME_INCOME_TYPE, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "NAME_INCOME_TYPE") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
```

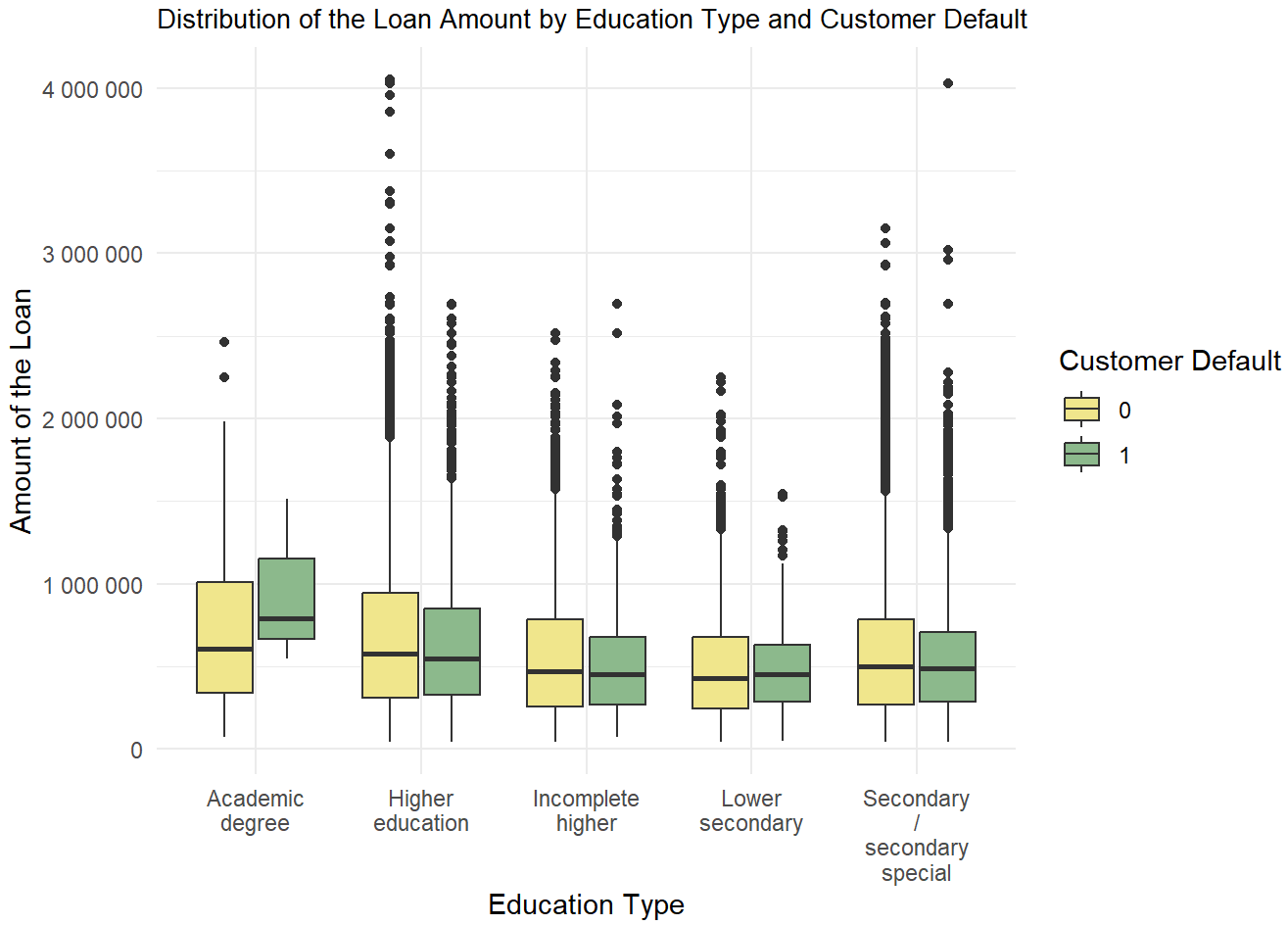
```r
  arrange(desc(order)) %>%
  mutate(NAME_INCOME_TYPE = factor(NAME_INCOME_TYPE, levels = unique(NAME_INCOME_TYPE)))

# Income by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(NAME_INCOME_TYPE, -order), fill = as.
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Income Type and Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



Education

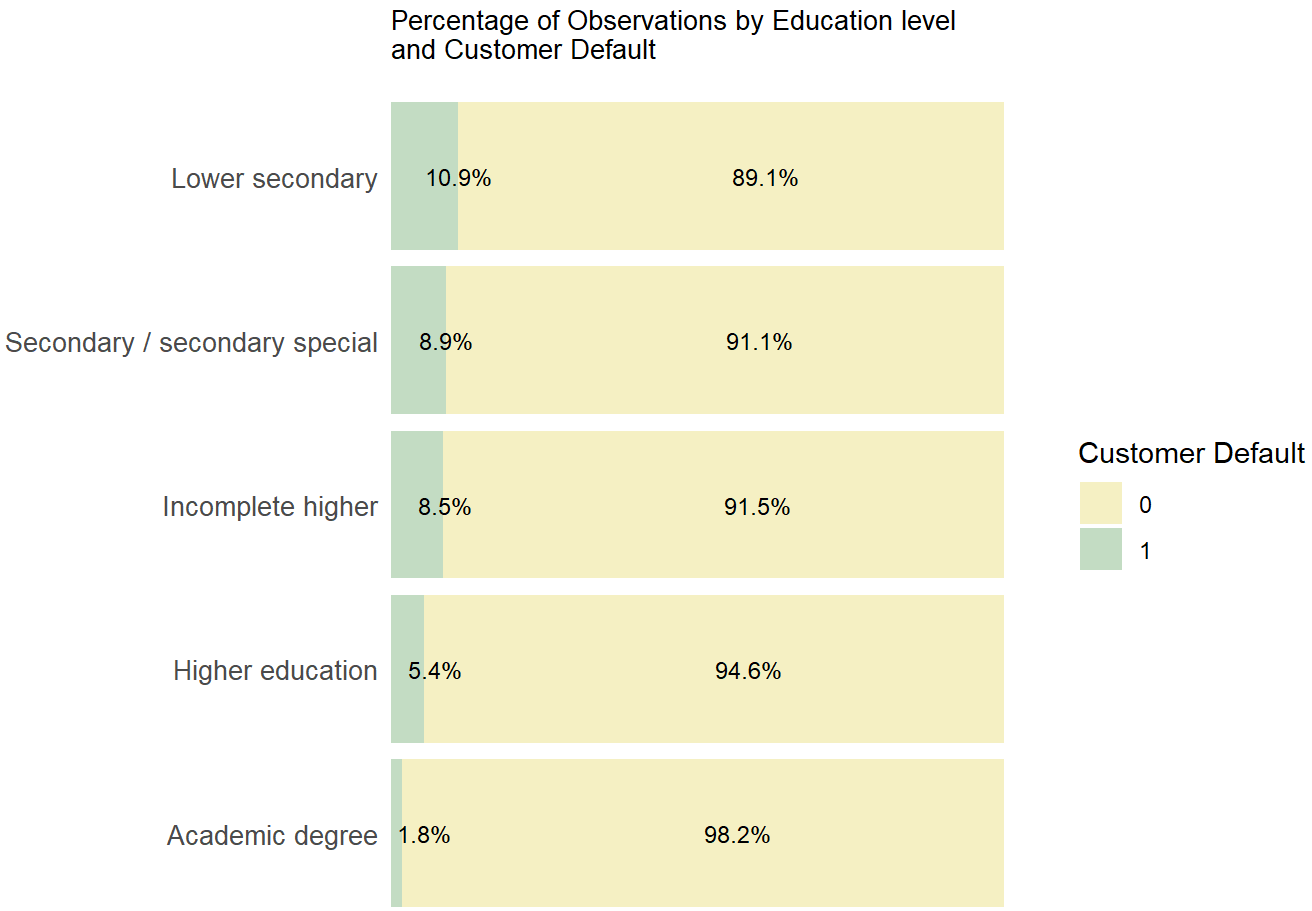
Among the educational levels, it is notable that individuals with an academic degree who are in default have higher loan amounts compared to those who are not in default. This pattern is not significantly observed in other educational levels.

```
# Loan amount by education and default
ggplot(app_train_clean, aes(x = as.factor(NAME_EDUCATION_TYPE), y = AMT_CREDIT, fill = as.factor(T
  geom_boxplot() +
  labs(title = "Distribution of the Loan Amount by Education Type and Customer Default",
       x = "Education Type",
       y = "Amount of the Loan") +
  scale_y_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10)) +
  theme(axis.text.x = element_text(hjust = 0.5)) +  # Center align the x-axis labels
  scale_x_discrete(labels = function(x) gsub(" ", "\n", x))  # Replace spaces with newline charact
```



Regarding education, all levels have default cases. However, the default rate for the “Lower Secondary” education group is six times higher compared to the “Academic Degree” group.

```r
# Summarize data by education and default
data_summary <- app_train_clean %>%
  group_by(NAME_EDUCATION_TYPE, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
  group_by(NAME_EDUCATION_TYPE) %>%
  mutate(total = sum(count),
         percentage = (count / total) * 100) %>%
  ungroup()

# Extract percentage for TARGET == 0g
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(NAME_EDUCATION_TYPE, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "NAME_EDUCATION_TYPE") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
  arrange(desc(order)) %>%
  mutate(NAME_EDUCATION_TYPE = factor(NAME_EDUCATION_TYPE, levels = unique(NAME_EDUCATION_TYPE)))

# Education by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(NAME_EDUCATION_TYPE, -order), fill =
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Education level \nand Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

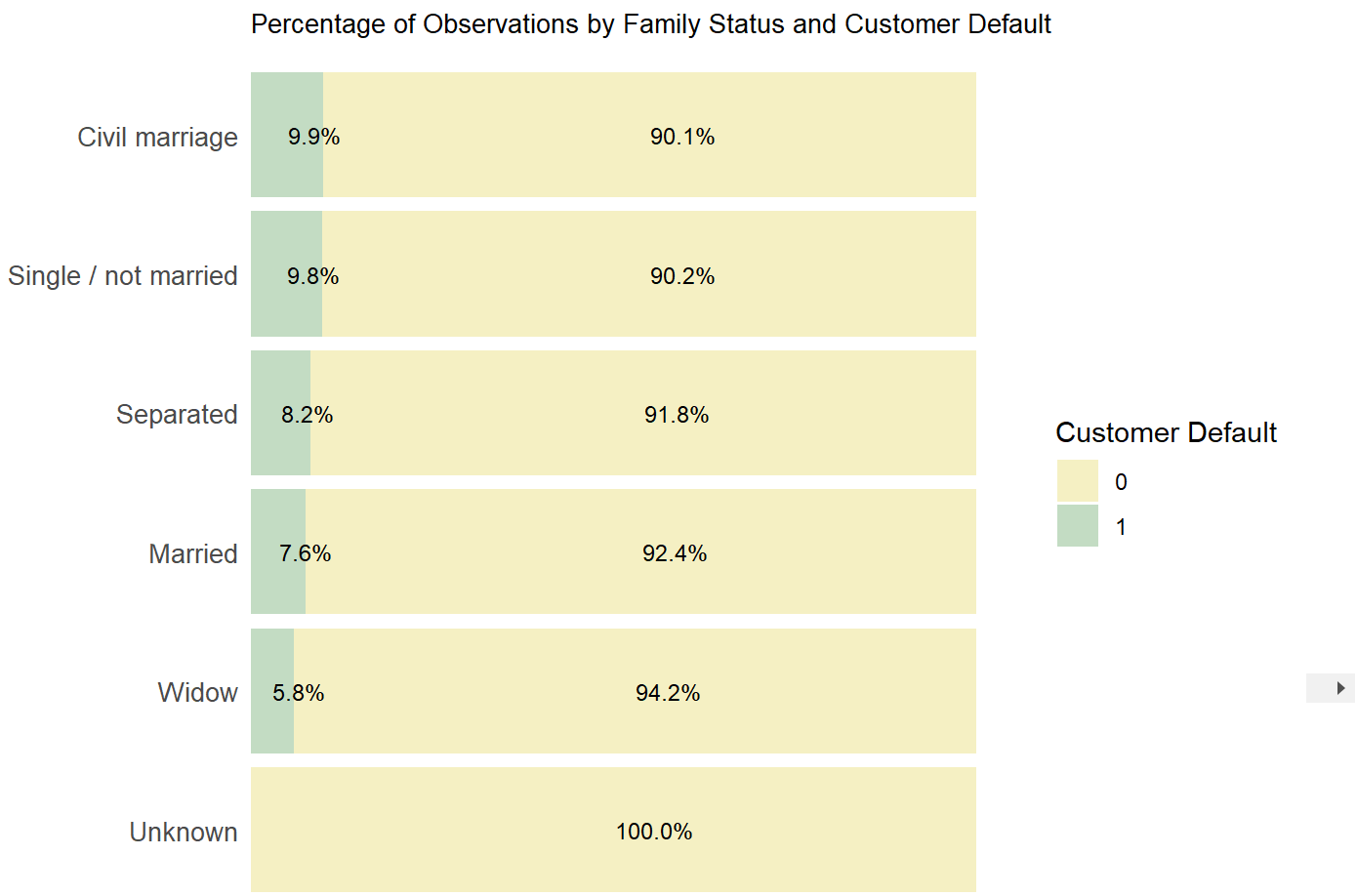


Family Status

The default rates by family status are quite similar, with a notable exception for the lower percentage of defaults among widows. There are no defaults reported among those with unknown family status.

```
# Summarize data by family and default
data_summary <- app_train_clean %>%
  group_by(NAME_FAMILY_STATUS, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
  group_by(NAME_FAMILY_STATUS) %>%
  mutate(total = sum(count),
         percentage = (count / total) * 100) %>%
  ungroup()

# Extract percentage for TARGET == 0
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(NAME_FAMILY_STATUS, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "NAME_FAMILY_STATUS") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
  arrange(desc(order)) %>%
```

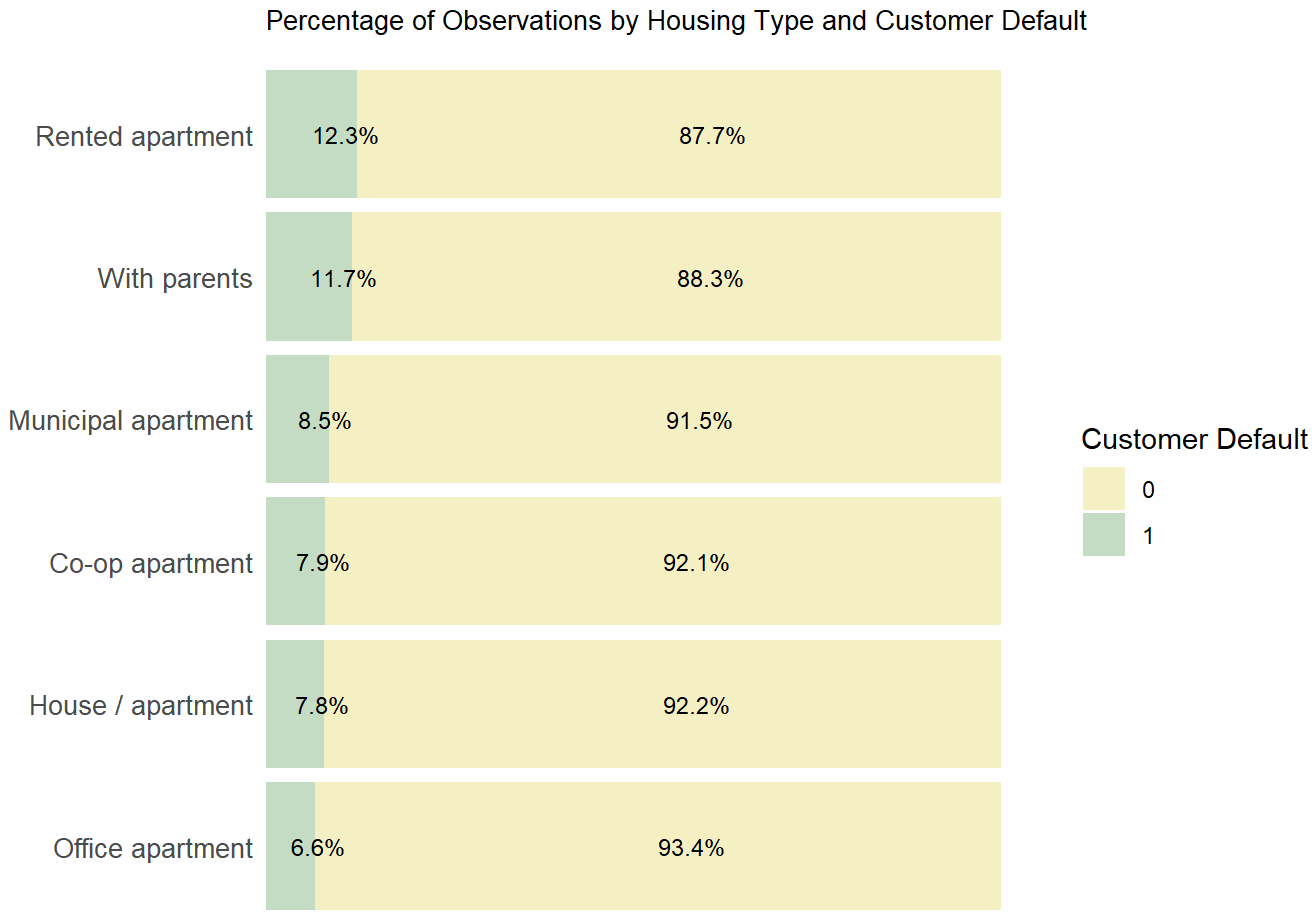
```r
  mutate(NAME_FAMILY_STATUS = factor(NAME_FAMILY_STATUS, levels = unique(NAME_FAMILY_STATUS)))

# Family status by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(NAME_FAMILY_STATUS, -order), fill = a
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Family Status and Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



Housing type

Consumers with different housing types exhibit varying default rates. Notably, those living in rented apartments or with their parents have default rates close to 12%. The lowest default rate is observed among those living in office apartments, at 6.6%.

```
# Summarize data by housing type and default
data_summary <- app_train_clean %>%
  group_by(NAME_HOUSING_TYPE, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
  group_by(NAME_HOUSING_TYPE) %>%
  mutate(total = sum(count),
         percentage = (count / total) * 100) %>%
  ungroup()

# Extract percentage for TARGET == 0
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(NAME_HOUSING_TYPE, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "NAME_HOUSING_TYPE") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
  arrange(desc(order)) %>%
  mutate(NAME_HOUSING_TYPE = factor(NAME_HOUSING_TYPE, levels = unique(NAME_HOUSING_TYPE)))

# Housing by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(NAME_HOUSING_TYPE, -order), fill = as
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Housing Type and Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



Occupation type

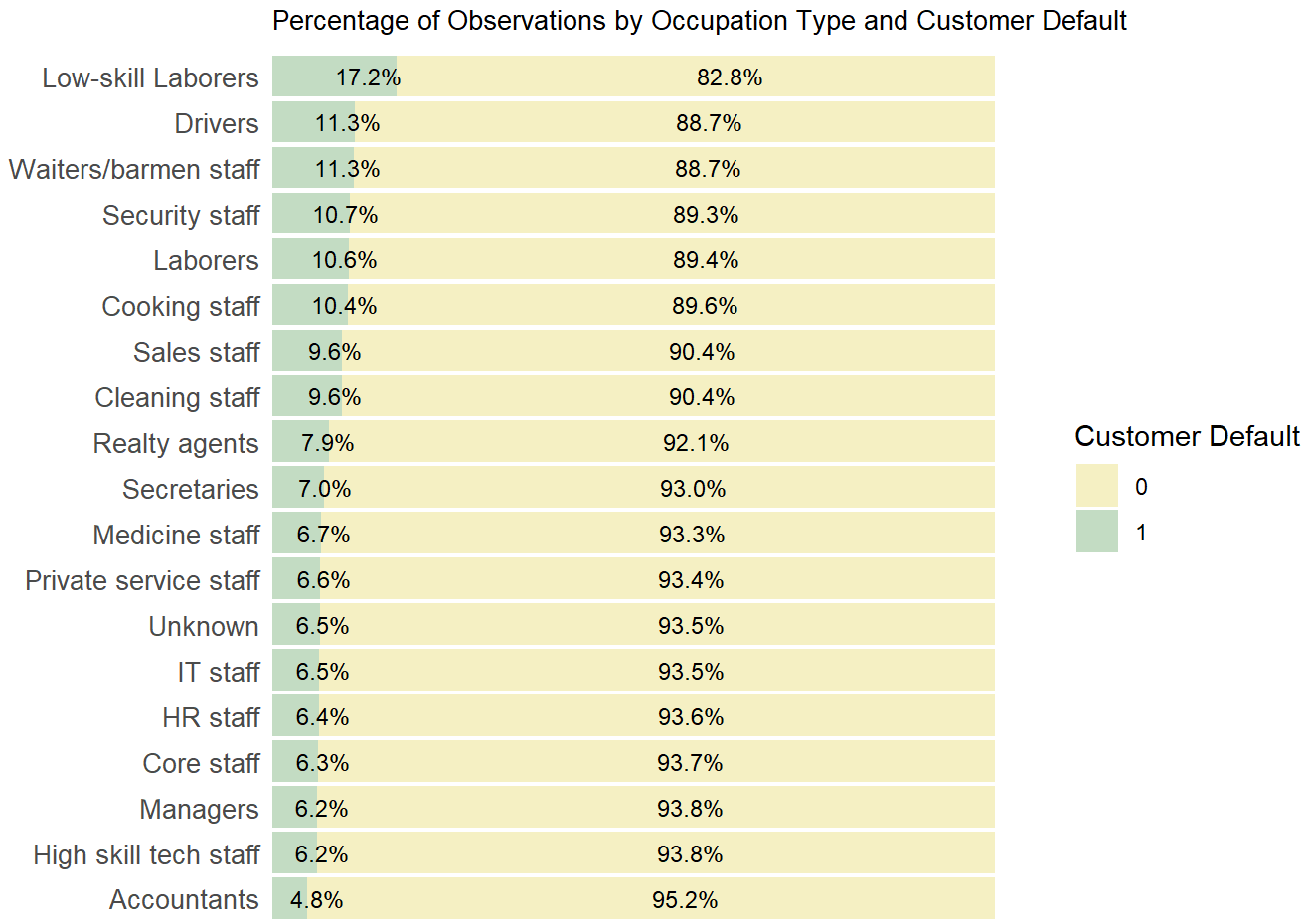
Among the occupation types, 96,391 observations were blank and have been transformed to “Unknown”.

```
# Convert OCCUPATION_TYPE to factor if not already done
app_train_clean$OCCUPATION_TYPE <- as.factor(app_train_clean$OCCUPATION_TYPE)

# Replace empty strings with "Unknown"
app_train_clean$OCCUPATION_TYPE <- as.character(app_train_clean$OCCUPATION_TYPE)  # Convert to cha
app_train_clean$OCCUPATION_TYPE[app_train_clean$OCCUPATION_TYPE == ""] <- "Unknown"  # Replace emp
app_train_clean$OCCUPATION_TYPE <- as.factor(app_train_clean$OCCUPATION_TYPE)  # Convert back to f
```

Among the occupation types, low-skill laborers have the highest default rate at 17.2%.

```
# Summarize data by occupation type and default
data_summary <- app_train_clean %>%
  group_by(OCCUPATION_TYPE, TARGET) %>%
  summarize(count = n(), .groups = 'drop') %>%
  group_by(OCCUPATION_TYPE) %>%
  mutate(total = sum(count),
         percentage = (count / total) * 100) %>%
```

```r
  ungroup()

# Extract percentage for TARGET == 0
percentage_target_0 <- data_summary %>%
  filter(TARGET == 0) %>%
  select(OCCUPATION_TYPE, percentage_target_0 = percentage)

# Order and adjust data for plotting
data_summary_ordered <- data_summary %>%
  left_join(percentage_target_0, by = "OCCUPATION_TYPE") %>%
  mutate(order = if_else(is.na(percentage_target_0), 0, percentage_target_0)) %>%
  arrange(desc(order)) %>%
  mutate(OCCUPATION_TYPE = factor(OCCUPATION_TYPE, levels = unique(OCCUPATION_TYPE)))

# Occupation by default
ggplot(data_summary_ordered, aes(x = percentage, y = reorder(OCCUPATION_TYPE, -order), fill = as.f
  geom_bar(stat = "identity", position = "stack", alpha = 0.5) +  # Add transparency
  geom_text(aes(label = scales::percent(percentage / 100, accuracy = 0.1)),
            position = position_stack(vjust = 0.5),
            hjust = -0.01,
            color = "black", size = 3.2) +
  labs(title = "Percentage of Observations by Occupation Type and Customer Default",
       x = NULL,  # Remove x-axis label
       y = NULL) +
  scale_x_continuous(labels = NULL, expand = expansion(c(0, 0.05))) +  # Remove x-axis values
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.y = element_text(size = 10),
        axis.title.x = element_blank(),  # Remove x-axis title
        legend.position = "right",
        legend.direction = "vertical",
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

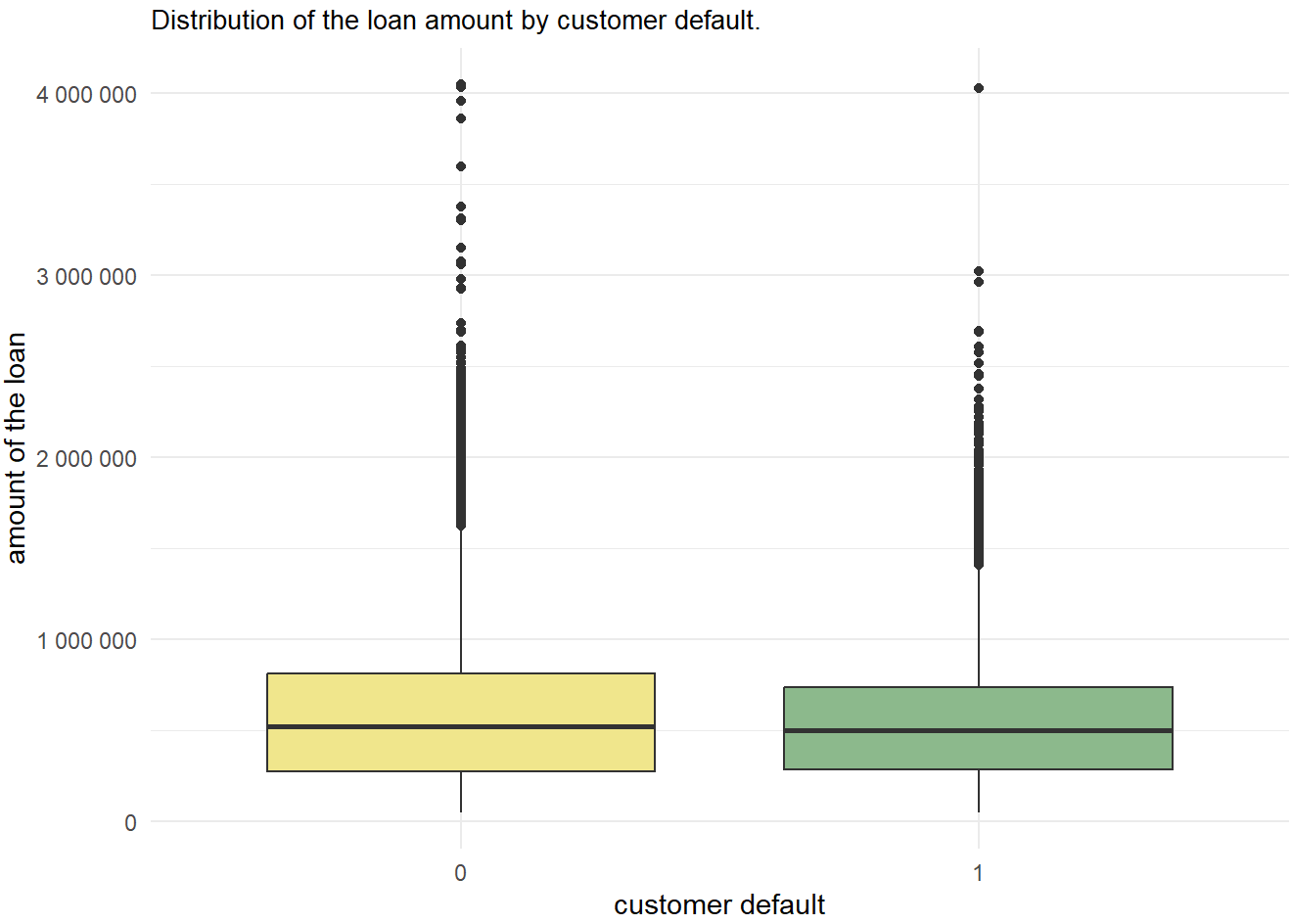


3.3 Additional Observations

Loan amount and default

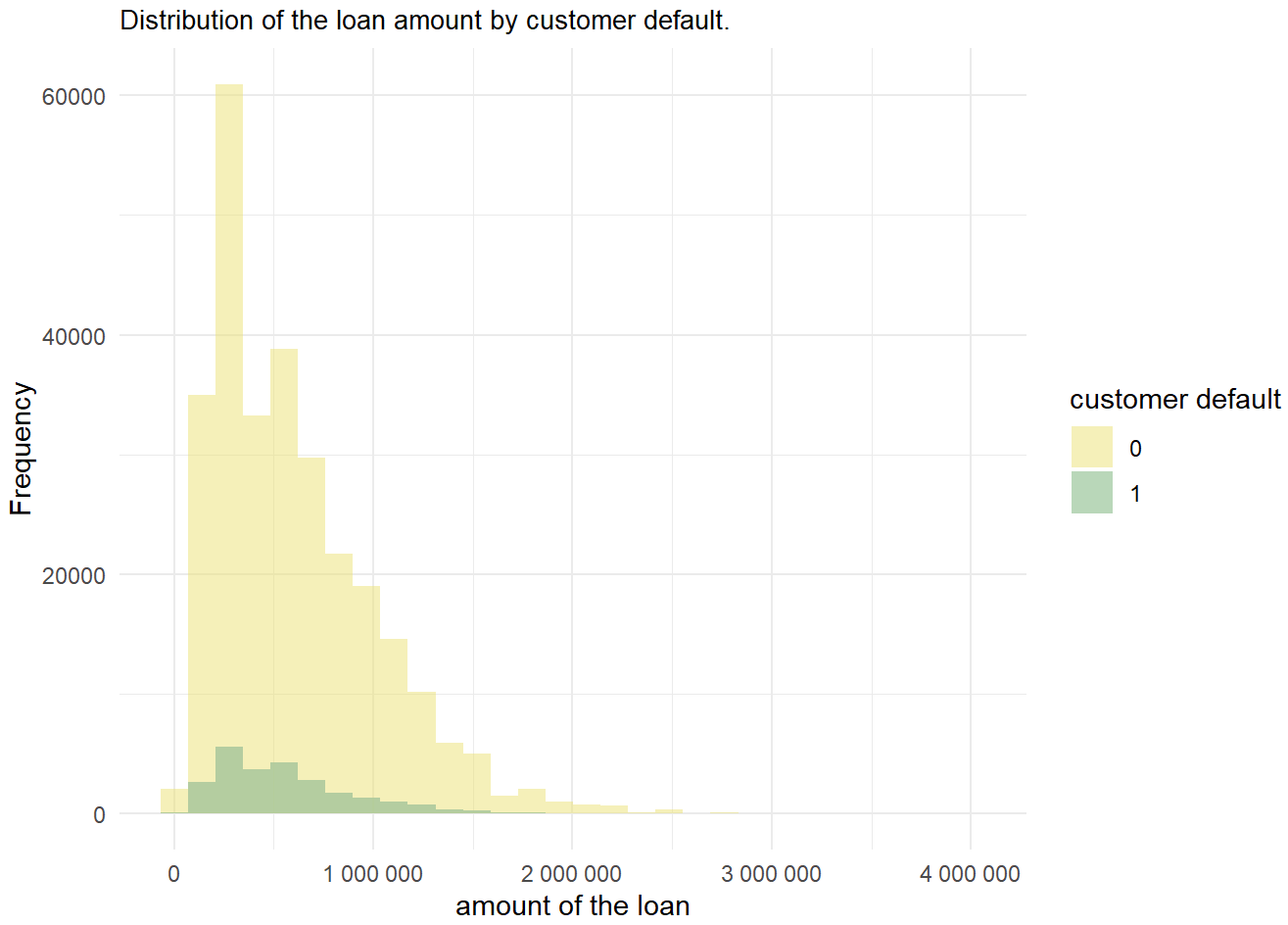
The average loan amounts are similar between the groups that default ($557,778) and those that do not ($602,648).

```
# Distribution of loan amount by default
ggplot(app_train_clean, aes(x = as.factor(TARGET), y = AMT_CREDIT, fill = as.factor(TARGET))) +
  geom_boxplot() +
  labs(title = "Distribution of the loan amount by customer default.",
       x = "customer default",
       y = "amount of the loan") +
  scale_y_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen")) +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(legend.position = "none")
```



The distribution of borrowers in relation to the loan amounts is left-skewed, with the same pattern of skewness at different intensities across the target variable groups.

```
# Histogram - distribution of loan amount by default
ggplot(app_train_clean, aes(x = AMT_CREDIT, fill = as.factor(TARGET))) +
  geom_histogram(alpha = 0.6, bins = 30, position = "identity") +
  labs(title = "Distribution of the loan amount by customer default.",
       x = "amount of the loan",
       y = "Frequency") +
  scale_x_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "customer default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(legend.position = "right")
```

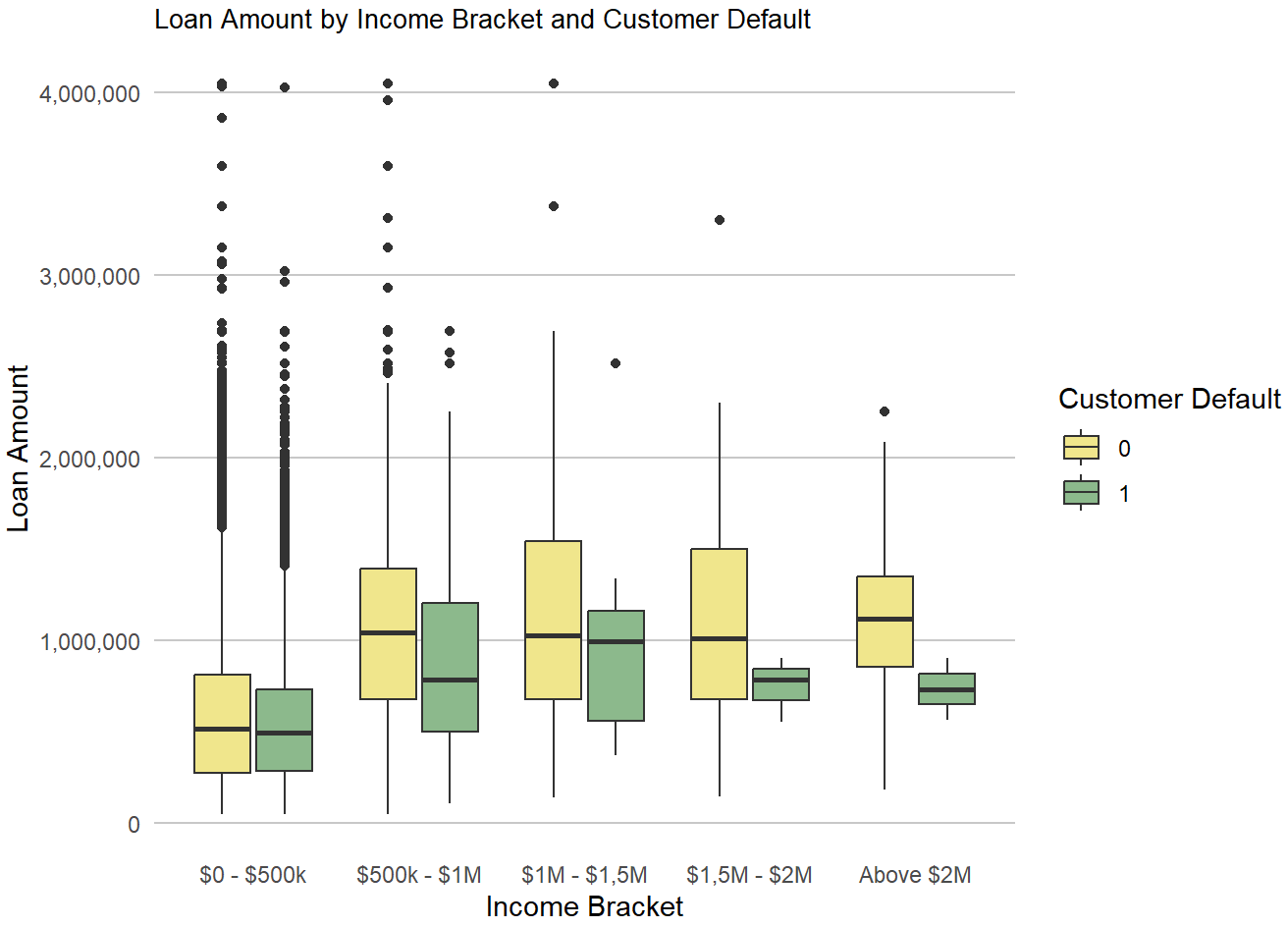


Loan amount, income and default

The graph below aims to explore the variation in loan amounts across income groups and default status. The income groups were defined based on the distribution of the majority of the data set.

```
# Income levels
income_b <- app_train_clean %>%
  mutate(income_bracket = cut(AMT_INCOME_TOTAL,
                              breaks = c(seq(0, 2000000, by = 500000), Inf),
                              labels = c(
                                "$0 - $500k",
                                "$500k - $1M",
                                "$1M - $1,5M",
                                "$1,5M - $2M",
                                "Above $2M")))

# Plot for loan by income and default
ggplot(income_b, aes(x = income_bracket, y = AMT_CREDIT, fill = as.factor(TARGET))) +
  geom_boxplot() +
  labs(title = "Loan Amount by Income Bracket and Customer Default",
       x = "Income Bracket",
       y = "Loan Amount") +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  scale_y_continuous(labels = scales::label_number(big.mark = ",")) +
```

```
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(axis.text.x = element_text(),
        axis.text.y = element_text(),
        legend.position = "right",
        panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.major.y = element_line(colour = "grey80"),
        panel.grid.minor.y = element_blank())
```



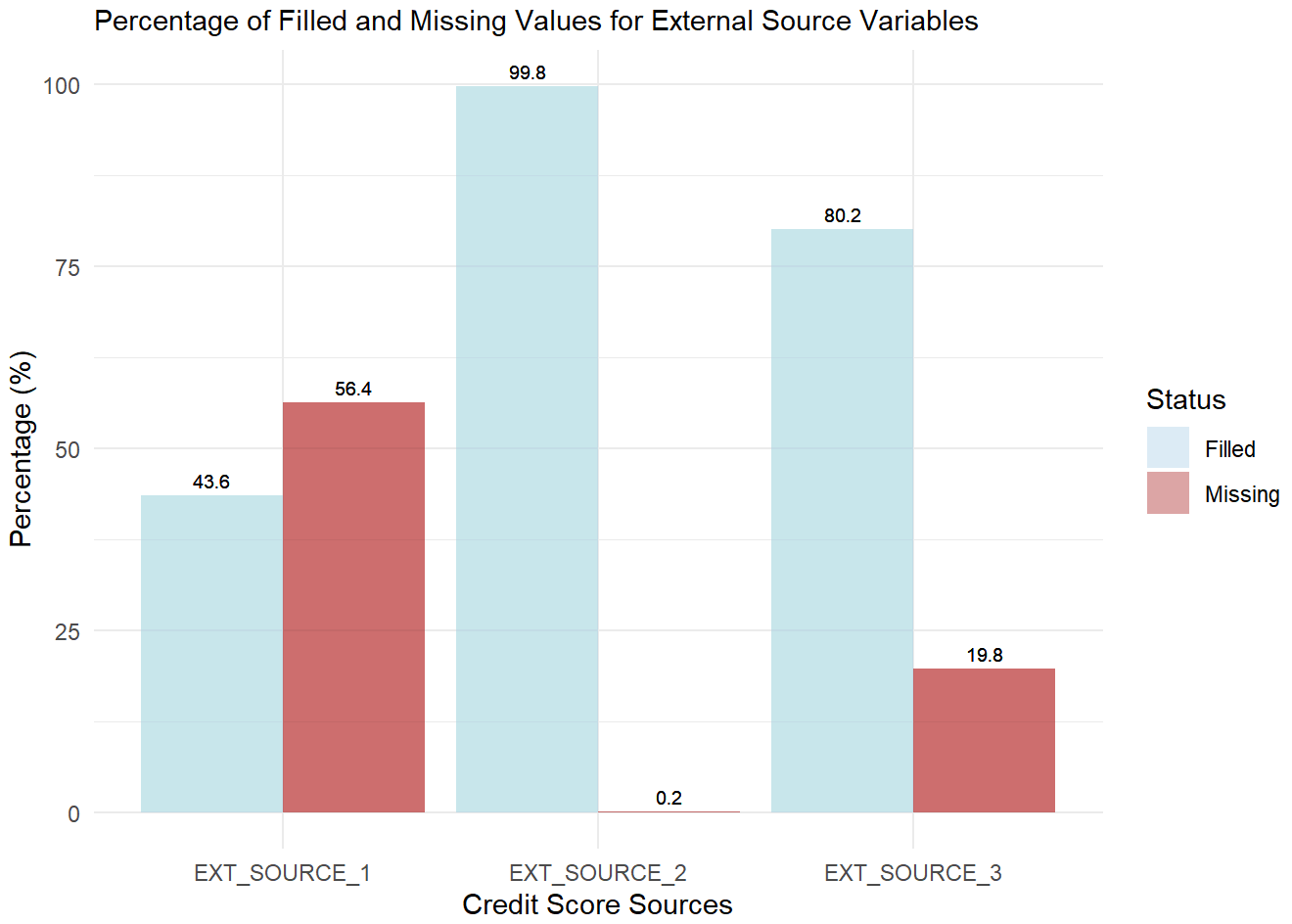
Among consumers with incomes above $500,000, the average loan amounts tend to be lower for those who default compared to those who pay on time. The exception is the income range between $1 million and $1.5 million, where the average loan amounts are very similar for both groups.

External sources

Credit scores generally serve as a good indicator of whether consumers are likely to delay payments or not. Among the variables used by Home Credit, there are three sources providing such information. These scores are normalized to be comparable, with values ranging from 0 to 1, where lower scores indicate higher risk of default and higher scores represent lower risk of default.

```
# Function to calculate count and percentage of non-missing and missing values for each EXT_SOURCE
calculate_percentages <- function(data, source) {
  data %>%
```

```
    summarise(
      Filled = sum(!is.na(.data[[source]])),
      Missing = sum(is.na(.data[[source]])),
      Total = n()
    ) %>%
    mutate(
      Percent_Filled = (Filled / Total) * 100,
      Percent_Missing = (Missing / Total) * 100,
      Source = source
    ) %>%
    pivot_longer(cols = c(Filled, Missing), names_to = "Status", values_to = "Count") %>%
    pivot_longer(cols = c(Percent_Filled, Percent_Missing), names_to = "Percentage_Status", values
    separate(Percentage_Status, into = c("Type", "Status"), sep = "_") %>%
    filter(Type == "Percent")
}

# Apply the function to each EXT_SOURCE
ext_source_1 <- calculate_percentages(app_train_clean, "EXT_SOURCE_1")
ext_source_2 <- calculate_percentages(app_train_clean, "EXT_SOURCE_2")
ext_source_3 <- calculate_percentages(app_train_clean, "EXT_SOURCE_3")

# Combining the sources
combined_data <- bind_rows(ext_source_1, ext_source_2, ext_source_3)

# Plot of external source percentages
ggplot(combined_data, aes(x = Source, y = Percentage, fill = Status)) +
  geom_col(position = "dodge", alpha = 0.4) +
  geom_text(aes(label = round(Percentage, 1)), position = position_dodge(width = 0.9), vjust = -0.
  labs(title = "Percentage of Filled and Missing Values for External Source Variables",
       x = "Credit Score Sources",
       y = "Percentage (%)") +
  scale_fill_manual(values = c("lightblue", "firebrick"), labels = c("Filled", "Missing")) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 0, hjust = 0.5),
        plot.title = element_text(size = 11))
```



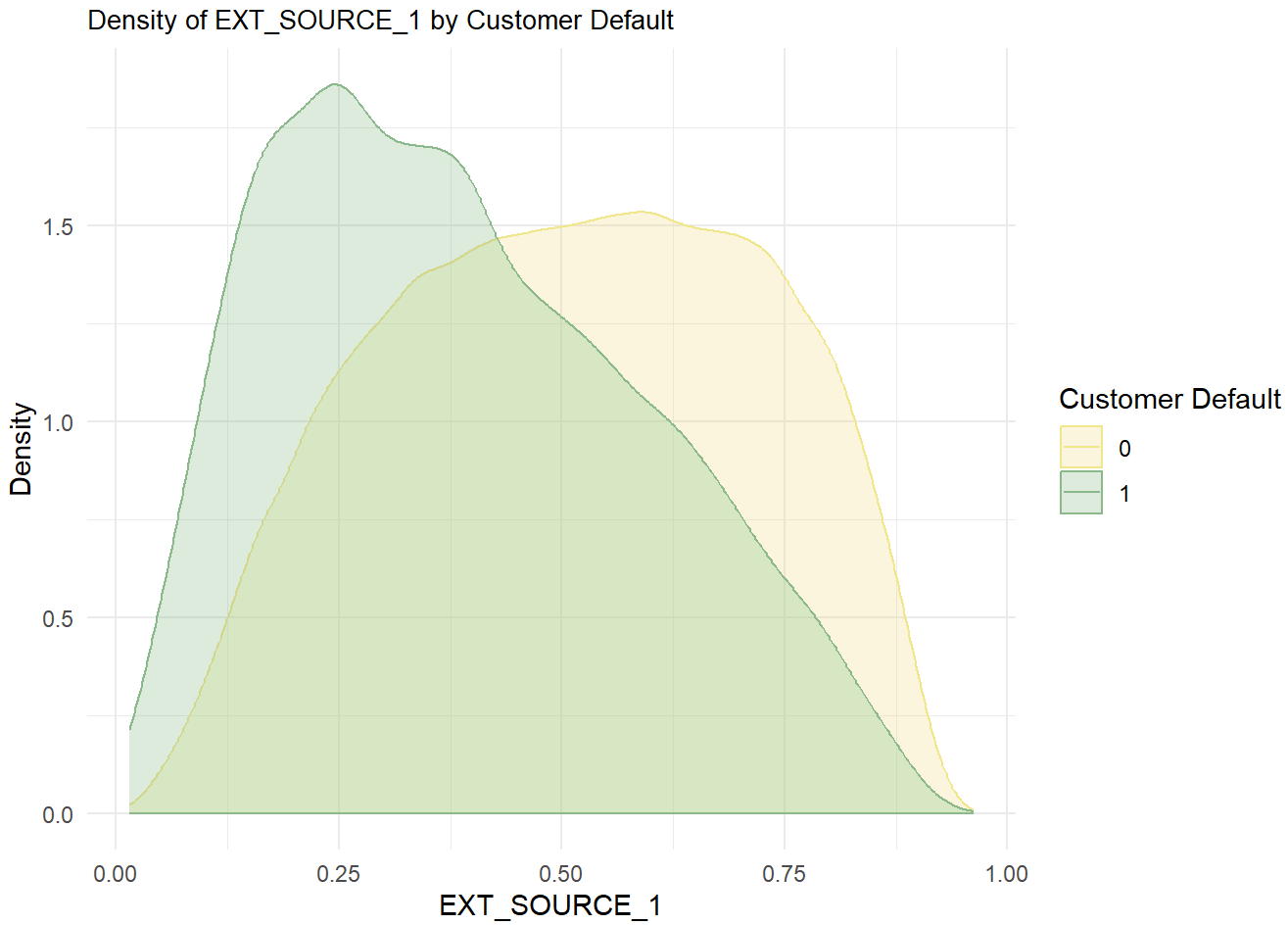
For almost all individuals observed, there is information available from SOURCE 2. For SOURCE 3, more than 70% of individuals have a score, while for SOURCE 1, less than 50% have available data.

Density plots

The graphs below illustrate the density distributions between consumers who are in default and those who are not.

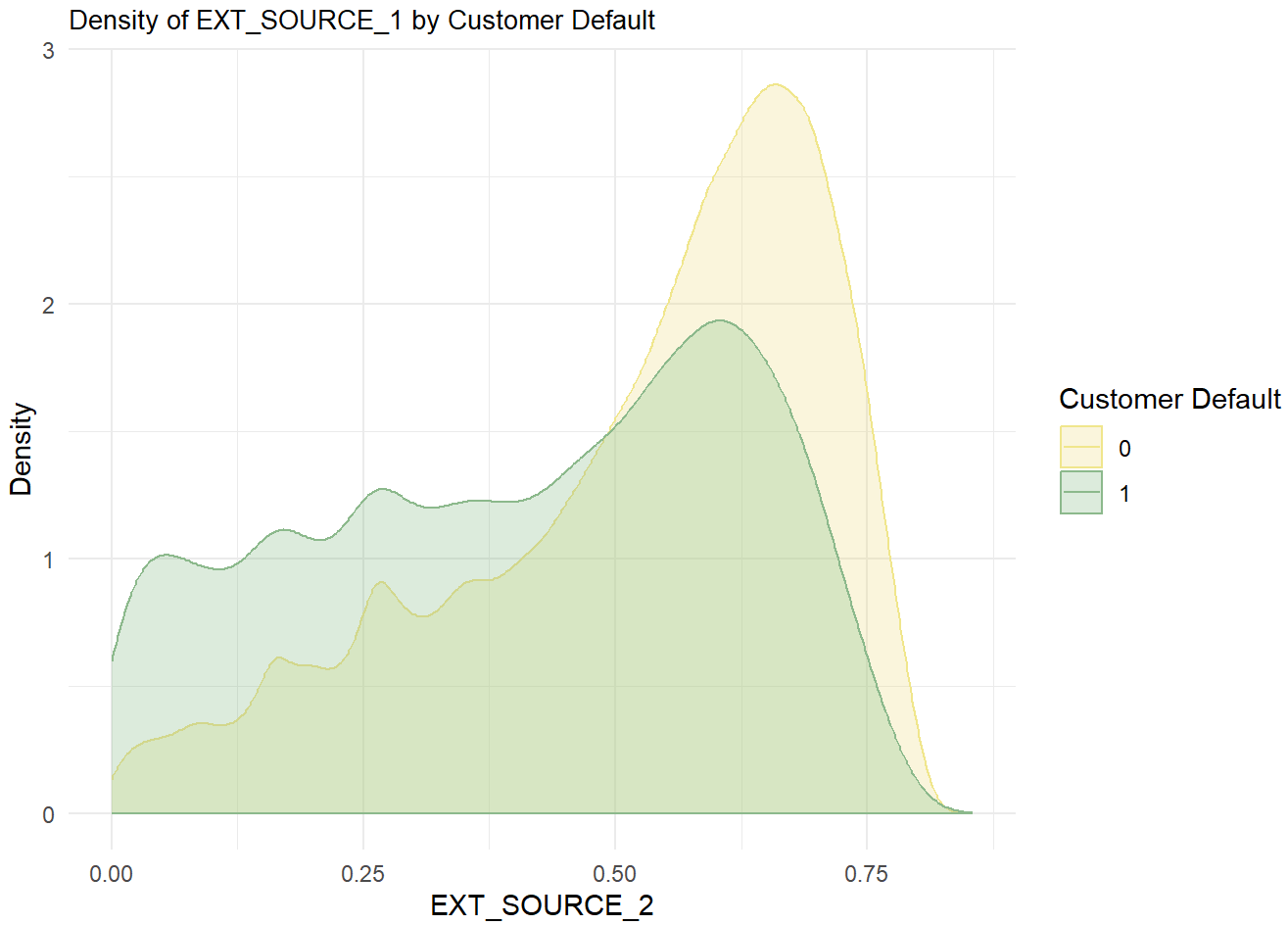
Source 1

```
# Density of ext_source_1 scores by default
ggplot(app_train_clean, aes(x = EXT_SOURCE_1, color = as.factor(TARGET), fill = as.factor(TARGET))
  stat_density(geom = "line", position = "identity") +
  stat_density(geom = "area", position = "identity", alpha = 0.3) +
  labs(title = "Density of EXT_SOURCE_1 by Customer Default",
       x = "EXT_SOURCE_1",
       y = "Density") +
  scale_color_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal()+
  theme(plot.title = element_text(size = 10))
```



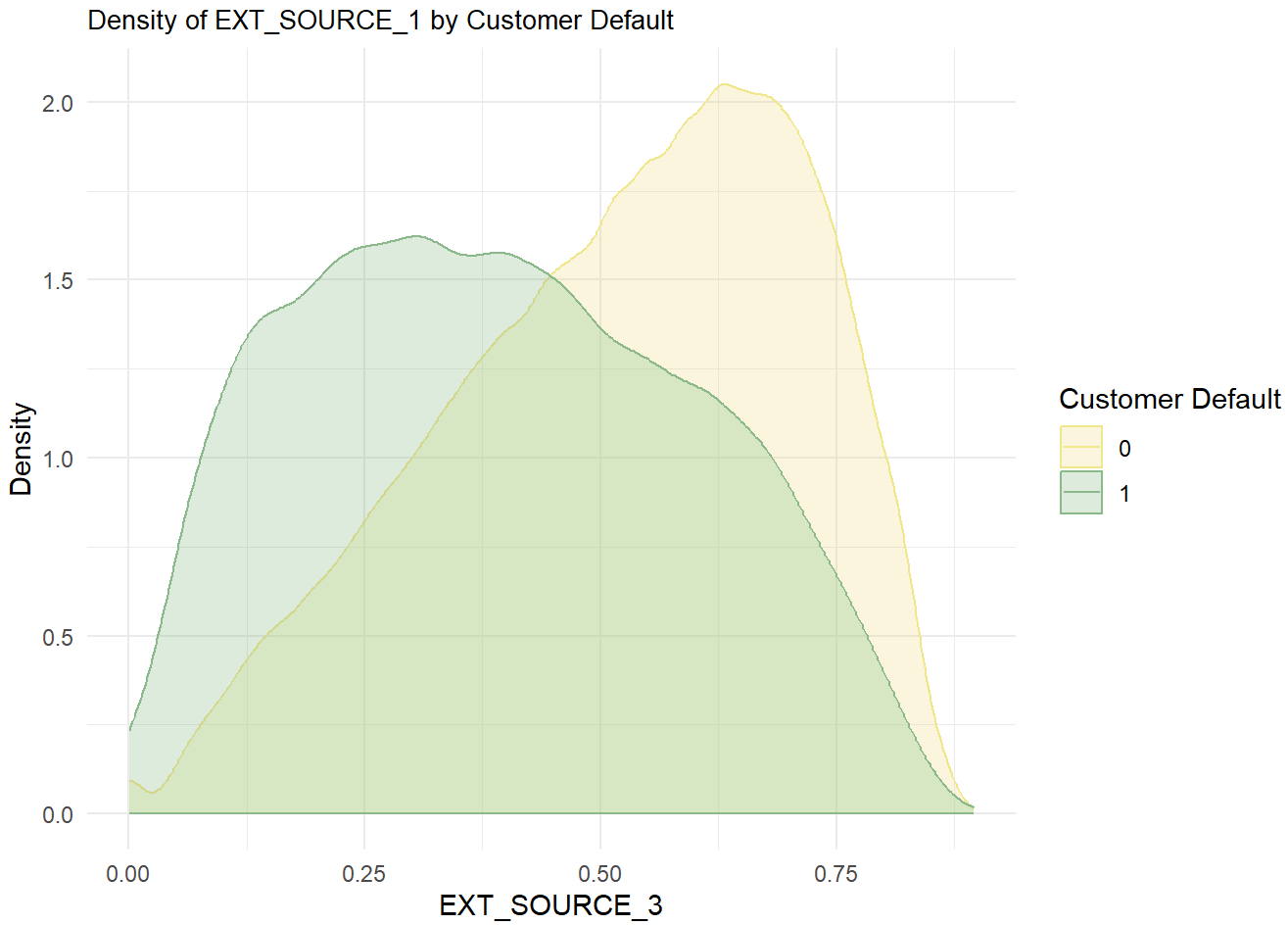
Source 2

```
# Density of ext_source_2 scores by default
ggplot(app_train_clean, aes(x = EXT_SOURCE_2, color = as.factor(TARGET), fill = as.factor(TARGET))
  stat_density(geom = "line", position = "identity") +
  stat_density(geom = "area", position = "identity", alpha = 0.3) +
  labs(title = "Density of EXT_SOURCE_1 by Customer Default",
       x = "EXT_SOURCE_2",
       y = "Density") +
  scale_color_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal()+
  theme(plot.title = element_text(size = 10))
```



Source 3

```
# Density of ext_source_3 scores by default
ggplot(app_train_clean, aes(x = EXT_SOURCE_3, color = as.factor(TARGET), fill = as.factor(TARGET))
  stat_density(geom = "line", position = "identity") +
  stat_density(geom = "area", position = "identity", alpha = 0.3) +
  labs(title = "Density of EXT_SOURCE_1 by Customer Default",
       x = "EXT_SOURCE_3",
       y = "Density") +
  scale_color_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  scale_fill_manual(values = c("khaki", "darkseagreen"), name = "Customer Default") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))
```



Below is the numeric distribution of scores by default status for each source.

```
# Select variables EXT_SOURCE_1, EXT_SOURCE_2, e EXT_SOURCE_3
app_train_filtered <- app_train_clean %>%
  select(TARGET, EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3) %>%
  pivot_longer(cols = starts_with("EXT_SOURCE"),
               names_to = "Source",
               values_to = "Score") %>%
  mutate(Source = factor(Source, levels = c("EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE_3")))

# Plot the three variables together
ggplot(app_train_filtered, aes(x = as.factor(TARGET), y = Score, fill = as.factor(TARGET))) +
  geom_boxplot() +
  labs(title = "Distribution of EXT_SOURCE Scores by Customer Default",
       x = "Customer Default",
       y = "Score") +
  scale_y_continuous(labels = scales::label_number(scale = 1)) +
  scale_fill_manual(values = c("khaki", "darkseagreen")) +
  theme_minimal() +
  theme(plot.title = element_text(size = 10))+
  theme(legend.position = "none",
        axis.text.x = element_text(size = 8),
        axis.text.y = element_text(size = 8)) +
  facet_wrap(~Source, scales = "free_y")
```

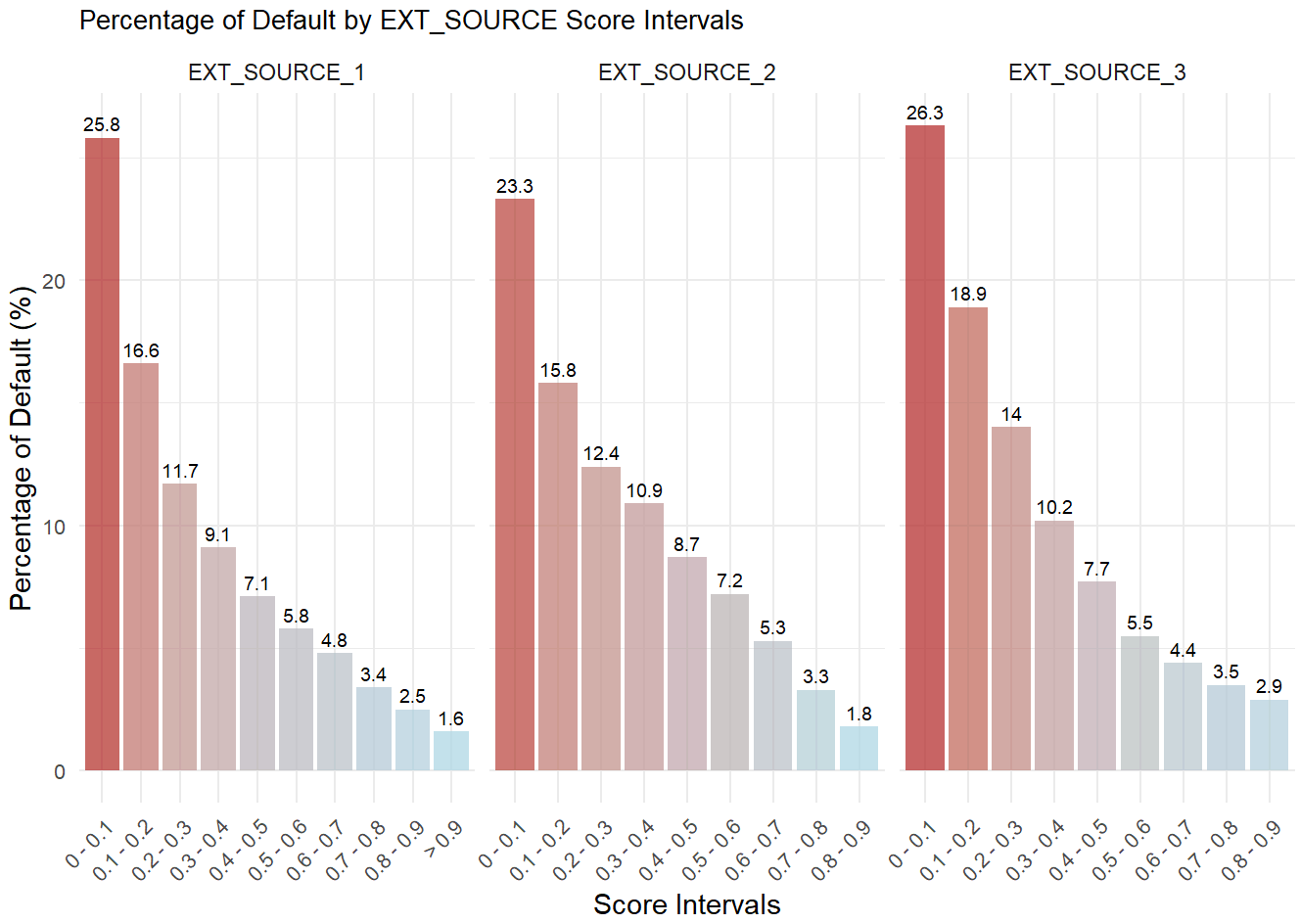
In the box plot above, comparing the three sources, it is clear that, on average, lower scores are observed among consumers who default. However, there are individuals with low scores who make their payments regularly, as well as those with high scores who end up defaulting.

The chart below clearly shows the percentages of defaults based on the scores.

```
# Filter dataset to include only rows where EXT_SOURCE variables are not missing
app_train_clean_filtered <- app_train_clean %>%
  filter(!is.na(EXT_SOURCE_1) | !is.na(EXT_SOURCE_2) | !is.na(EXT_SOURCE_3))

# Define intervals for EXT_SOURCE variables with "> 0.9" at the end
app_train_clean_filtered <- app_train_clean_filtered %>%
  mutate(score_levels_1 = case_when(
    EXT_SOURCE_1 <= 0.1 ~ "0 - 0.1",
    EXT_SOURCE_1 <= 0.2 ~ "0.1 - 0.2",
    EXT_SOURCE_1 <= 0.3 ~ "0.2 - 0.3",
    EXT_SOURCE_1 <= 0.4 ~ "0.3 - 0.4",
    EXT_SOURCE_1 <= 0.5 ~ "0.4 - 0.5",
    EXT_SOURCE_1 <= 0.6 ~ "0.5 - 0.6",
    EXT_SOURCE_1 <= 0.7 ~ "0.6 - 0.7",
    EXT_SOURCE_1 <= 0.8 ~ "0.7 - 0.8",
    EXT_SOURCE_1 <= 0.9 ~ "0.8 - 0.9",
    TRUE ~ "> 0.9"
  )) %>%
```

```r
  mutate(score_levels_2 = case_when(
    EXT_SOURCE_2 <= 0.1 ~ "0 - 0.1",
    EXT_SOURCE_2 <= 0.2 ~ "0.1 - 0.2",
    EXT_SOURCE_2 <= 0.3 ~ "0.2 - 0.3",
    EXT_SOURCE_2 <= 0.4 ~ "0.3 - 0.4",
    EXT_SOURCE_2 <= 0.5 ~ "0.4 - 0.5",
    EXT_SOURCE_2 <= 0.6 ~ "0.5 - 0.6",
    EXT_SOURCE_2 <= 0.7 ~ "0.6 - 0.7",
    EXT_SOURCE_2 <= 0.8 ~ "0.7 - 0.8",
    EXT_SOURCE_2 <= 0.9 ~ "0.8 - 0.9",
    TRUE ~ "> 0.9"
  )) %>%
  mutate(score_levels_3 = case_when(
    EXT_SOURCE_3 <= 0.1 ~ "0 - 0.1",
    EXT_SOURCE_3 <= 0.2 ~ "0.1 - 0.2",
    EXT_SOURCE_3 <= 0.3 ~ "0.2 - 0.3",
    EXT_SOURCE_3 <= 0.4 ~ "0.3 - 0.4",
    EXT_SOURCE_3 <= 0.5 ~ "0.4 - 0.5",
    EXT_SOURCE_3 <= 0.6 ~ "0.5 - 0.6",
    EXT_SOURCE_3 <= 0.7 ~ "0.6 - 0.7",
    EXT_SOURCE_3 <= 0.8 ~ "0.7 - 0.8",
    EXT_SOURCE_3 <= 0.9 ~ "0.8 - 0.9",
    TRUE ~ "> 0.9"
  ))

# Combine the data into a long format for faceting
counts_df <- bind_rows(
  app_train_clean_filtered %>%
    filter(!is.na(EXT_SOURCE_1)) %>%
    group_by(score_levels_1, TARGET) %>%
    summarise(n = n(), .groups = 'drop') %>%
    pivot_wider(names_from = TARGET, values_from = n, names_prefix = "n_TARGET_") %>%
    mutate(percent_default = round((n_TARGET_1 / (n_TARGET_1 + n_TARGET_0)) * 100, 1)) %>%
    rename(`Default_no` = `n_TARGET_0`, `Default_yes` = `n_TARGET_1`) %>%
    mutate(Source = "EXT_SOURCE_1", score_levels = factor(score_levels_1, levels = c(
      "0 - 0.1", "0.1 - 0.2", "0.2 - 0.3", "0.3 - 0.4", "0.4 - 0.5",
      "0.5 - 0.6", "0.6 - 0.7", "0.7 - 0.8", "0.8 - 0.9", "> 0.9"
    ))),

  app_train_clean_filtered %>%
    filter(!is.na(EXT_SOURCE_2)) %>%
    group_by(score_levels_2, TARGET) %>%
    summarise(n = n(), .groups = 'drop') %>%
    pivot_wider(names_from = TARGET, values_from = n, names_prefix = "n_TARGET_") %>%
    mutate(percent_default = round((n_TARGET_1 / (n_TARGET_1 + n_TARGET_0)) * 100, 1)) %>%
    rename(`Default_no` = `n_TARGET_0`, `Default_yes` = `n_TARGET_1`) %>%
    mutate(Source = "EXT_SOURCE_2", score_levels = factor(score_levels_2, levels = c(
      "0 - 0.1", "0.1 - 0.2", "0.2 - 0.3", "0.3 - 0.4", "0.4 - 0.5",
      "0.5 - 0.6", "0.6 - 0.7", "0.7 - 0.8", "0.8 - 0.9", "> 0.9"
    ))),
```

```r
  app_train_clean_filtered %>%
    filter(!is.na(EXT_SOURCE_3)) %>%
    group_by(score_levels_3, TARGET) %>%
    summarise(n = n(), .groups = 'drop') %>%
    pivot_wider(names_from = TARGET, values_from = n, names_prefix = "n_TARGET_") %>%
    mutate(percent_default = round((n_TARGET_1 / (n_TARGET_1 + n_TARGET_0)) * 100, 1)) %>%
    rename(`Default_no` = `n_TARGET_0`, `Default_yes` = `n_TARGET_1`) %>%
    mutate(Source = "EXT_SOURCE_3", score_levels = factor(score_levels_3, levels = c(
      "0 - 0.1", "0.1 - 0.2", "0.2 - 0.3", "0.3 - 0.4", "0.4 - 0.5",
      "0.5 - 0.6", "0.6 - 0.7", "0.7 - 0.8", "0.8 - 0.9", "> 0.9"
    )))
) %>%
  select(Source, score_levels, percent_default)

# Create the facet wrap plot with gradient colors and no legend
ggplot(counts_df, aes(x = score_levels, y = percent_default, fill = percent_default)) +
  geom_bar(stat = "identity", alpha = 0.7) +
  geom_text(aes(label = round(percent_default, 1)), position = position_dodge(width = 0.9), vjust
  labs(title = "Percentage of Default by EXT_SOURCE Score Intervals",
       x = "Score Intervals",
       y = "Percentage of Default (%)") +
  scale_fill_gradient(low = "lightblue", high = "firebrick") +
  theme_minimal() +
  theme(plot.title = element_text(size = 10)) +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 8),
    axis.text.y = element_text(size = 8),
    legend.position = "none"  # Remove the legend
  ) +
  facet_wrap(~Source, scales = "free_x")
```



As expected, lower scores tend to show a higher percentage of defaults.

3.4 Correlations

The tables below display the main correlations between numerical variables and default.

```
# Create the correlation matrix
cor_matrix <- cor(app_train_clean %>% select_if(is.numeric), use = "complete.obs")

# Extract the target correlations
target_correlations <- cor_matrix["TARGET", ]

# Remove the correlation of TARGET with itself (correlation of 1)
target_correlations <- target_correlations[names(target_correlations) != "TARGET"]

# Sort the correlations
sorted_correlations <- sort(target_correlations, decreasing = TRUE)

# Ten most positive correlations
knitr::kable(head(sorted_correlations, 10), caption = "Ten most Positive correlations")
```

Ten most Positive correlations

	x
REGION_RATING_CLIENT_W_CITY	0.0625843
REGION_RATING_CLIENT	0.0545622
FLAG_DOCUMENT_3	0.0540813
DAYS_BIRTH	0.0443884
OWN_CAR_AGE	0.0386917
AMT_REQ_CREDIT_BUREAU_YEAR	0.0336506
OBS_30_CNT_SOCIAL_CIRCLE	0.0299685
OBS_60_CNT_SOCIAL_CIRCLE	0.0295146
DAYS_ID_PUBLISH	0.0292651
DEF_60_CNT_SOCIAL_CIRCLE	0.0254981

In addition to the data exploration conducted so far, the table above shows the ten highest positive correlations between numerical variables and default.

It is noteworthy that variables related to consumers' social circles and the regions where they live are present. The submission of document 3, the age of the car, and the time since a new ID issuance are also among the strongest correlations.

```
# Sort the correlations
sorted_correlations <- sort(target_correlations)

# Ten most negative correlations
knitr::kable(head(sorted_correlations, 10), caption = "Ten most Negative correlations")
```

Ten most Negative correlations

	x
EXT_SOURCE_3	-0.1570209
EXT_SOURCE_2	-0.1370899
EXT_SOURCE_1	-0.1333277
DAYS_EMPLOYED_YEARS	-0.0596796
FLOORSMAX_AVG	-0.0477564
FLOORSMAX_MODE	-0.0474792
FLOORSMAX_MEDI	-0.0471926
AMT_INCOME_TOTAL	-0.0460543
FLOORSMIN_MEDI	-0.0323371
FLOORSMIN_AVG	-0.0322657

Among the negative correlations—where higher values are associated with a lower occurrence of default—external credit information stands out, with values approximately two to three times larger than the fourth variable. An increase in years of employment, age, and income also contributes to a lower likelihood of default. Additionally, variables related to individuals' housing show that the larger these values are, the lower the probability of default.

3.5 Evaluation of Prediction Baselines

Naive baseline and majority classifier

The naive baseline and the Majority Rule Classifier both predict that no customer defaults, as 91.9% of customers do not default (TARGET=0). This results in an accuracy of 91.9% for the model; however, it fails to identify any defaulters, missing the 8.1% who actually do default. Thus, even though the accuracy appears high, this approach is not effective for identifying customers at risk of default. A more advanced model is needed to accurately classify both defaulters and non-defaulters.

Random classifier

The Random Classifier makes predictions randomly based on the distribution of classes in the data set. With 92% of the TARGET being 0 and 8% being 1, the classifier will predict TARGET = 0 with a 92% probability and TARGET = 1 with an 8% probability.

The expected accuracy reflects the class distribution, resulting in an expected accuracy of about 92% for TARGET = 0 and 8% for TARGET = 1, similar to the majority rule classifier.

The simulation below is intended to illustrate the accuracy of using a random classifier for this data set.

```
# Seed for reproducibility
set.seed(seed)

# Number of predictions
num_predictions <- nrow(app_train_clean)

# Probabilities for each class
prob_0 <- 0.92
prob_1 <- 0.08

# Generate random predictions based on the probabilities
random_predictions <- sample(c(0, 1), size = num_predictions, replace = TRUE, prob = c(prob_0, pro

# Calculate accuracy
accuracy_random <- round(mean(random_predictions == app_train_clean$TARGET),2)
accuracy_random
```

```
[1] 0.85
```

Given that a Random Classifier achieves an accuracy of 85%, I expect the developed model to outperform this benchmark, showing improvements in both precision and recall, thereby enhancing its ability to accurately identify true positives and minimize false negatives.

---

4. Pre-Modeling

4.1 Dummy Encoding

Before proceeding with the dummy encoding process, we will remove instances with missing CODE_Gender values. Additionally, the variable created to indicate individuals with an infinite number of employment days will be converted into a factor.

In addition, all non-numeric variables will be selected to perform dummy encoding, ensuring they are properly transformed for analysis.

```
# Filter out instances with 'XNA' in CODE_GENDER and remove unused levels
app_train_clean <- app_train_clean[app_train_clean$CODE_GENDER != "XNA", ]
app_train_clean$CODE_GENDER <- droplevels(app_train_clean$CODE_GENDER)

# Convert the DAYS_EMPLOYED_ANOM variable to a factor
app_train_clean$DAYS_EMPLOYED_ANOM <- as.factor(app_train_clean$DAYS_EMPLOYED_ANOM)

# Identify all non-numeric columns in the dataset (factors)
non_numeric_list <- unlist(lapply(app_train_clean, is.factor))
# Create a data table with only non-numeric columns
data_non_num <- setDT(app_train_clean)[,..non_numeric_list]

# Combine the target variable with the non-numeric data
data_non_num <- cbind(data_non_num, app_train_clean[,'TARGET'])
```

For clarity and standardization, the variables will be renamed to remove special characters.

```
# Function to clean the levels of each factor variable
clean_factor_levels <- function(factor_var) {
  # Replace spaces and punctuation with underscores and convert to lowercase
  levels(factor_var) <- tolower(gsub("[[:space:][:punct:]]", "_", levels(factor_var)))
  return(factor_var)
}

# Applying the function to each variable in the data.table
data_non_num[] <- lapply(data_non_num, function(x) {
  # Check if the variable is a factor
  if (is.factor(x)) {
    # Clean the factor levels
    clean_factor_levels(x)
  } else {
    # Return the variable unchanged if it is not a factor
    return(x)
```

```
  }
})
```

Dummy encoding is a common method for transforming categorical variables into a numerical format, making them usable for machine learning models. This approach creates binary (0 or 1) columns for each category in a categorical variable, allowing the models to interpret these categories without implying any order. Our intention in using dummy encoding is to avoid biases that can arise when treating categorical variables as ordered. Additionally, dummy encoding helps the model better understand the relationships between the input features and the target variable.

```
# Create dummy variables for the non-numeric data, dropping the second level to avoid multicolline
dummies <- dummyVars(TARGET ~ ., data = data_non_num, drop2nd = TRUE)

# Apply the dummy variable transformation to the non-numeric data
data_non_num_dum <- predict(dummies, newdata = data_non_num)
```

The code below combines the variables back into a single data frame after performing dummy encoding. The process will create new columns in the dataset, increasing the total to 253.

```
#Function to change index to column
index_to_col <- function(data, Column_Name){
          data <- cbind(newColName = rownames(data), data)
          rownames(data) <- 1:nrow(data)
          colnames(data)[1] <- Column_Name
          return (data)
        }

# Recreate the list that includes numeric and integer columns
numeric_integer_list <- unlist(lapply(app_train_clean, function(x) is.numeric(x) || is.integer(x))

# Create a new data frame with the numeric and integer columns
data_num <- setDT(app_train_clean)[, ..numeric_integer_list]  # Select only numeric and integer co

# Combine one-hot encoded data with numeric and integer data
data_pre <- cbind(data_non_num_dum, data_num)
```

4.2 Handling with Missing Data

The process of identifying missing values in the dataset is described. The percentage of missing data for each column is calculated and visualized in the following table. Then, the missing values are replaced with the mean for each variable using the aggregate function.

```
# Calculate the percentage of missing values for each column
mv <- as.data.frame(apply(data_pre, 2, function(col) sum(is.na(col)) / length(col)))
colnames(mv)[1] <- "missing_values"  # Rename the first column to "missing_values"

# Add a column with the index as the first column
```

```r
mv <- index_to_col(mv, 'Column')
# Order the missing values in descending order
mv <- setDT(mv)[order(missing_values, decreasing = TRUE)]

# Create an interactive table to display all columns with their missing values percentage
datatable(mv, options = list(pageLength = 10),
          caption = "Percentage of Missing Values for Each Column") %>%
  formatPercentage('missing_values', 1)  # Format the missing values as percentage with 2 decimal
```

Show 10 entries Search:

Percentage of Missing Values for Each Column

	Column	missing_values
1	COMMONAREA_AVG	69.9%
2	COMMONAREA_MODE	69.9%
3	COMMONAREA_MEDI	69.9%
4	NONLIVINGAPARTMENTS_AVG	69.4%
5	NONLIVINGAPARTMENTS_MODE	69.4%
6	NONLIVINGAPARTMENTS_MEDI	69.4%
7	LIVINGAPARTMENTS_AVG	68.4%
8	LIVINGAPARTMENTS_MODE	68.4%
9	LIVINGAPARTMENTS_MEDI	68.4%
10	FLOORSMIN_AVG	67.8%

Showing 1 to 10 of 253 entries Previous 1 2 3 4 5 ... 26 Next

```r
# Fill in the missing values using aggregation
data_pre <- na.aggregate(data_pre)

# Replace dots with underscores
colnames(data_pre) <- gsub("\\.", "_", colnames(data_pre))
```

4.3 Sampling and Balancing

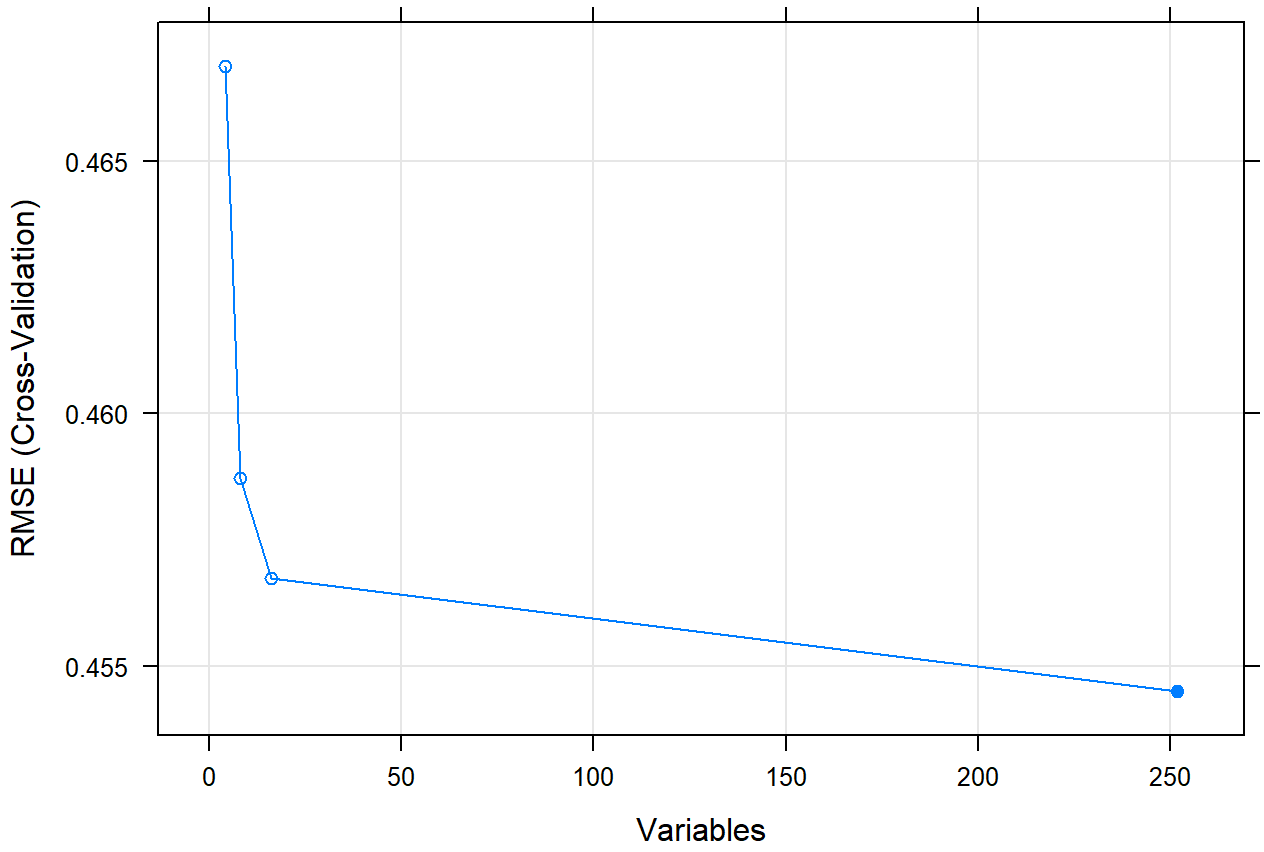
Due to the high computational cost of processing the entire dataset, we opted to perform downsampling by using nearly all instances that represent TARGET == 1. To balance the dataset and avoid bias in the modeling process, we randomly sampled the same number of instances with

TARGET == 0. As a result, the dataset is balanced at 50% for each class, using approximately 23% of the total dataset (72,000 instances) for training and testing. Additionally, We removed the variable SK_ID_CURR because it is not necessary for the modeling process, as it is just a registration number.

Recursive Feature Elimination for Variable Selection

This code uses the Recursive Feature Elimination (RFE) method to select important variables from the dataset, aiming to reduce computational time for subsequent analyses. This process takes many hours to complete, so I chose to save the vector and load it in its final version.

```
# Register parallel backend for faster computation using available cores
#registerDoParallel(cores = parallel::detectCores() - 1)

#tic()
# Set control parameters for RFE, using random forest functions and cross-validation with 3 folds
#control <- rfeControl(functions=rfFuncs, method="cv", number=3)

# Set training control to compute class probabilities and use AUC as the summary function
#trainctrl <- trainControl(classProbs= TRUE, summaryFunction = twoClassSummary)

# Perform Recursive Feature Elimination (RFE)
#results <- rfe(
#  as.data.frame(data_pre_sample)[, -target_column_index],  # All columns except the target
#  as.data.frame(data_pre_sample)[, target_column_index],   # Only the target
#  rfeControl = control,
#  method = "rf",
#  metric = "AUC",
#  trControl = trainctrl)

#toc()

# Save the RFE results to an .RData file
#save(results, file = "D:/mymodels/rfe_results.RData")
load("D:/mymodels/rfe_results.RData")

# Print the RFE results to see the variable selection performance
#print(results, digits = 2)

# Plot the results showing variable importance and selection performance
plot(results, type=c("g", "o"))
```



The results indicate that among the 253 variables in this dataset, fewer than 25 correspond to the lowest root mean squared error. Below is the list of the 10 most important predictors.

```
# Obtain the list of predictors ordered by importance
selected_predictors <- predictors(results)

# Limit to the first 10 predictors
top_predictors <- selected_predictors[1:10]

# Create a data frame from the top predictors
predictors_df <- data.frame(Predictors = top_predictors)

# Display the table of ordered predictors
knitr::kable(predictors_df, caption = "Top 10 Predictors")
```

Top 10 Predictors

Predictors
EXT_SOURCE_3
EXT_SOURCE_2
EXT_SOURCE_1
AMT_CREDIT

Predictors
DAYS_EMPLOYED_YEARS
AMT_GOODS_PRICE
DAYS_BIRTH
AMT_ANNUITY
DAYS_ID_PUBLISH
NAME_EDUCATION_TYPE_higher_education

To simplify the dataset, we removed the variables NAME_EDUCATION_TYPE_secondary__secondary_special, NAME_EDUCATION_TYPE_higher_education, NAME_CONTRACT_TYPE_revolving_loans, and CODE_GENDER_f, as they were redundant or exhibited collinearity, ensuring a cleaner and more effective set of predictors. Additionally, to include a broader range of predictors and improve the model's quality, we chose to focus on the 50 most impactful variables, going beyond the standard Recursive Feature Elimination approach to refine the selection further.

```
# Define the number of predictors to keep
n_predictors_to_keep <- 54

# Select the first predictors
cols_to_keep <- c(predictors(results)[1:n_predictors_to_keep], "TARGET")

# Create a new data frame that only includes the selected columns
data_pre_sample <- as.data.frame(data_pre_sample)[, (colnames(data_pre_sample) %in% cols_to_keep)]

# Remove the redundant binary variables from the data frame `data_pre_sample`
# Removing the "NAME_EDUCATION_TYPE_secondary___secondary_special" column and keeping only "NAME_E
data_pre_sample$NAME_EDUCATION_TYPE_secondary___secondary_special <- NULL

# Removing the "NAME_EDUCATION_TYPE_higher_education" column and keeping only "NAME_EDUCATION_TYPE
data_pre_sample$NAME_EDUCATION_TYPE_higher_education <- NULL

# Removing the "NAME_CONTRACT_TYPE_revolving_loans" column and keeping only "NAME_CONTRACT_TYPE_re
data_pre_sample$NAME_CONTRACT_TYPE_revolving_loans <- NULL

# Removing the "CODE_GENDER_f" column and keeping only "CODE_GENDER_m"
data_pre_sample$CODE_GENDER_f <- NULL

# Remove the variable from data_pre_sample
#data_pre_sample <- data_pre_sample[, !colnames(data_pre_sample) %in% "NAME_CONTRACT_TYPE_Revolvin

# Remove the variables 'NAME_CONTRACT_TYPE_Revolving loans' and 'CODE_GENDER_M' from data_pre_samp
#data_pre_sample <- data_pre_sample[, !colnames(data_pre_sample) %in% c("NAME_CONTRACT_TYPE_Revolv
```

```
# Save data_pre_sample to a file
save(data_pre_sample, file = "D:/mymodels/data_pre_sample.RData")
```

4.4 Data partition

Partition

The dataset will be partitioned into training and testing sets. The TARGET variable will be transformed into a binary format to facilitate better model interpretation, and 70% of the data will be allocated for training while the remaining 30% will be reserved for testing, ensuring effective evaluation on unseen data.

```
# Set the seed for reproducibility
set.seed(seed)

# Transform TARGET variable into a binary factor
data_pre_sample <- mutate(data_pre_sample, TARGET = ifelse(TARGET == 0, 'Class0', 'Class1'))
data_pre_sample$TARGET <- as.factor(data_pre_sample$TARGET)

# Create a partition for training and testing sets; 70% for training
inTrain <- createDataPartition(data_pre_sample$TARGET, p = .7)[[1]]
dt_train <- data_pre_sample[inTrain, ]  # Training data
dt_test  <- data_pre_sample[-inTrain, ]  # Testing data
```

The training control is configured to use repeated cross-validation, enhancing the reliability of the model evaluation. This setup incorporates 10 folds with 2 repeats and enables probability calculations for each class.

```
# train control
traincntrl <- trainControl(
  method = 'repeatedcv',          # Use repeated cross-validation as the method
  number = 10,                     # Number of folds for cross-validation
  repeats = 2,                    # Number of times to repeat the cross-validation process
  classProbs = TRUE,              # Calculate class probabilities to evaluate performance
  summaryFunction = twoClassSummary, # Use the twoClassSummary to evaluate performance metrics
  verboseIter = FALSE,             # Show progress during training
  allowParallel = TRUE)           # Allow parallel processing for faster training
```

---

5. Modeling

Due to the computational cost of the modeling process, we opted to save the final versions of the models, making the knitting process easier.

All the metrics displayed within the model blocks refer to their performance on the test set (30%, 21,600 instances).

5.1 Logistic Regression

Since we have already performed the processes of variable elimination and standardization, I decided not to use techniques like Lasso and Ridge at this moment. (Note: we tested them and still opted to remove it.)

```
# Set time measure
tic()

# Train the logistic regression model with centering and scaling
logistic_model <- train(TARGET ~ .,              # Train the model to predict TARGET using all pre
                        data = dt_train,           # Use the training dataset
                        method = 'glm',           # Specify the method as generalized linear model
                        preProcess = c("center", "scale"),  # Apply centering and scaling
                        trControl = traincntrl)   # Apply the training control settings
# End time measure
toc()
```

```
24.67 sec elapsed
```

```
# Save the logistic regression model to an .RData file
#save(logistic_model, file = "D:/mymodels/logistic_model.RData")
#load("D:/mymodels/logistic_model.RData")

#Summary
print(summary(logistic_model), digits = 2)
```

```
Call:
NULL

Deviance Residuals:
   Min      1Q  Median      3Q     Max
 -3.20   -0.84   -0.58    1.04    2.96

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                    -0.8202     0.0107   -76.9   <2e-16 ***
NAME_CONTRACT_TYPE_cash_loans   0.0923     0.0118     7.8    7e-15 ***
CODE_GENDER_m                   0.1516     0.0106    14.2   <2e-16 ***
AMT_INCOME_TOTAL               -0.1158     0.0175    -6.6    4e-11 ***
AMT_CREDIT                      0.8704     0.0645    13.5   <2e-16 ***
AMT_ANNUITY                     0.1518     0.0176     8.6   <2e-16 ***
AMT_GOODS_PRICE                -0.9871     0.0661   -14.9   <2e-16 ***
DAYS_BIRTH                     -0.0217     0.0134    -1.6    0.105
DAYS_REGISTRATION               0.0291     0.0112     2.6    0.009 **
DAYS_ID_PUBLISH                 0.0744     0.0108     6.9    6e-12 ***
OWN_CAR_AGE                     0.0401     0.0101     4.0    7e-05 ***
REGION_RATING_CLIENT           -0.0472     0.0342    -1.4    0.168
REGION_RATING_CLIENT_W_CITY     0.1240     0.0343     3.6    3e-04 ***
```

```
REG_CITY_NOT_LIVE_CITY          0.0506     0.0100     5.0    5e-07 ***
EXT_SOURCE_1                   -0.1998     0.0117   -17.1   <2e-16 ***
EXT_SOURCE_2                   -0.4460     0.0110   -40.5   <2e-16 ***
EXT_SOURCE_3                   -0.5050     0.0108   -46.7   <2e-16 ***
APARTMENTS_AVG                 -0.1179     0.1974    -0.6    0.550
BASEMENTAREA_AVG               -0.2099     0.1747    -1.2    0.230
YEARS_BEGINEXPLUATATION_AVG    -0.1244     0.0997    -1.2    0.212
YEARS_BUILD_AVG                 0.5551     0.2218     2.5    0.012 *
COMMONAREA_AVG                 -0.0357     0.0703    -0.5    0.612
FLOORSMAX_AVG                   0.0817     0.1624     0.5    0.615
LANDAREA_AVG                    0.0175     0.0549     0.3    0.750
LIVINGAPARTMENTS_AVG           -0.0761     0.1420    -0.5    0.592
LIVINGAREA_AVG                  0.0231     0.1210     0.2    0.849
NONLIVINGAREA_AVG               0.0597     0.0786     0.8    0.448
APARTMENTS_MODE                 0.0034     0.0938     0.0    0.971
BASEMENTAREA_MODE               0.1103     0.0797     1.4    0.166
YEARS_BEGINEXPLUATATION_MODE    0.0514     0.0487     1.1    0.291
YEARS_BUILD_MODE               -0.0781     0.0772    -1.0    0.312
COMMONAREA_MODE                 0.0260     0.0699     0.4    0.710
FLOORSMAX_MODE                 -0.0165     0.0812    -0.2    0.839
LANDAREA_MODE                   0.0131     0.0688     0.2    0.849
LIVINGAPARTMENTS_MODE          -0.0617     0.0703    -0.9    0.380
LIVINGAREA_MODE                -0.1091     0.0898    -1.2    0.224
NONLIVINGAREA_MODE             -0.0405     0.0519    -0.8    0.435
APARTMENTS_MEDI                 0.1292     0.2074     0.6    0.533
BASEMENTAREA_MEDI               0.0877     0.1742     0.5    0.614
YEARS_BEGINEXPLUATATION_MEDI    0.0528     0.1003     0.5    0.599
YEARS_BUILD_MEDI               -0.5002     0.2148    -2.3    0.020 *
FLOORSMAX_MEDI                 -0.1083     0.1764    -0.6    0.539
LANDAREA_MEDI                  -0.0352     0.0845    -0.4    0.677
LIVINGAPARTMENTS_MEDI           0.1582     0.1536     1.0    0.303
LIVINGAREA_MEDI                 0.0677     0.1460     0.5    0.643
NONLIVINGAREA_MEDI             -0.0117     0.0909    -0.1    0.898
TOTALAREA_MODE                 -0.0041     0.0318    -0.1    0.897
DEF_30_CNT_SOCIAL_CIRCLE        0.0744     0.0104     7.1    9e-13 ***
DAYS_LAST_PHONE_CHANGE          0.0412     0.0110     3.7    2e-04 ***
FLAG_DOCUMENT_6                -0.0471     0.0120    -3.9    8e-05 ***
DAYS_EMPLOYED_YEARS            -0.1446     0.0121   -11.9   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 64161  on 50399  degrees of freedom
Residual deviance: 55827  on 50349  degrees of freedom
AIC: 55929

Number of Fisher Scoring iterations: 4
```

The coefficient values in the logistic regression are expressed as probabilities of occurrence, indicating the likelihood of an event happening for each predictor. However, interpreting these

coefficients can be complex, as they are influenced by the scale and type of variables used in the model, with only 20 out of 50 predictors meeting the 5% significance level or lower.

These include: NAME_CONTRACT_TYPE_cash_loans, CODE_GENDER_m, AMT_INCOME_TOTAL, AMT_CREDIT, AMT_ANNUITY, AMT_GOODS_PRICE, DAYS_ID_PUBLISH, OWN_CAR_AGE, REGION_RATING_CLIENT_W_CITY, REG_CITY_NOT_LIVE_CITY, EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3, DEF_30_CNT_SOCIAL_CIRCLE, DAYS_LAST_PHONE_CHANGE, FLAG_DOCUMENT_6, DAYS_EMPLOYED_YEARS, DAYS_REGISTRATION, YEARS_BUILD_AVG, and YEARS_BUILD_MEDI.

Among these 20 predictors, 8 are listed among the top 10 most important predictors identified through the RFE process.

Train set Metrics

```
# Predict probabilities on the train dataset
pred_train_logistic <- predict(logistic_model, newdata = dt_train, type = "prob")  # Generate pred

# ROC and AUC for the train set
roc_train_logistic <- roc(dt_train$TARGET, pred_train_logistic$Class1, levels = c("Class0", "Class
auc_train_logistic <- auc(roc_train_logistic)  # Calculate AUC
cat("AUC Train set - Logistic:", round(auc_train_logistic, 2), "\n")
```

```
AUC Train set - Logistic: 0.74
```

```
# Predict class labels for the train set using the logistic regression model
predictClassesTrainLogistic <- predict(logistic_model, dt_train)

# Calculate Accuracy for the train set
accuracy_train_logistic <- mean(predictClassesTrainLogistic == dt_train$TARGET)
cat("Accuracy Train set - Logistic:", round(accuracy_train_logistic, 2), "\n")
```

```
Accuracy Train set - Logistic: 0.72
```

```
# Generate confusion matrix for the logistic regression model on the train set
conf_matrix_train_logistic <- confusionMatrix(data = predictClassesTrainLogistic,
                                              reference = dt_train$TARGET,
                                              positive = "Class1")  # "Class1" as the positive cla
```

The model showed the above performance on the train set.

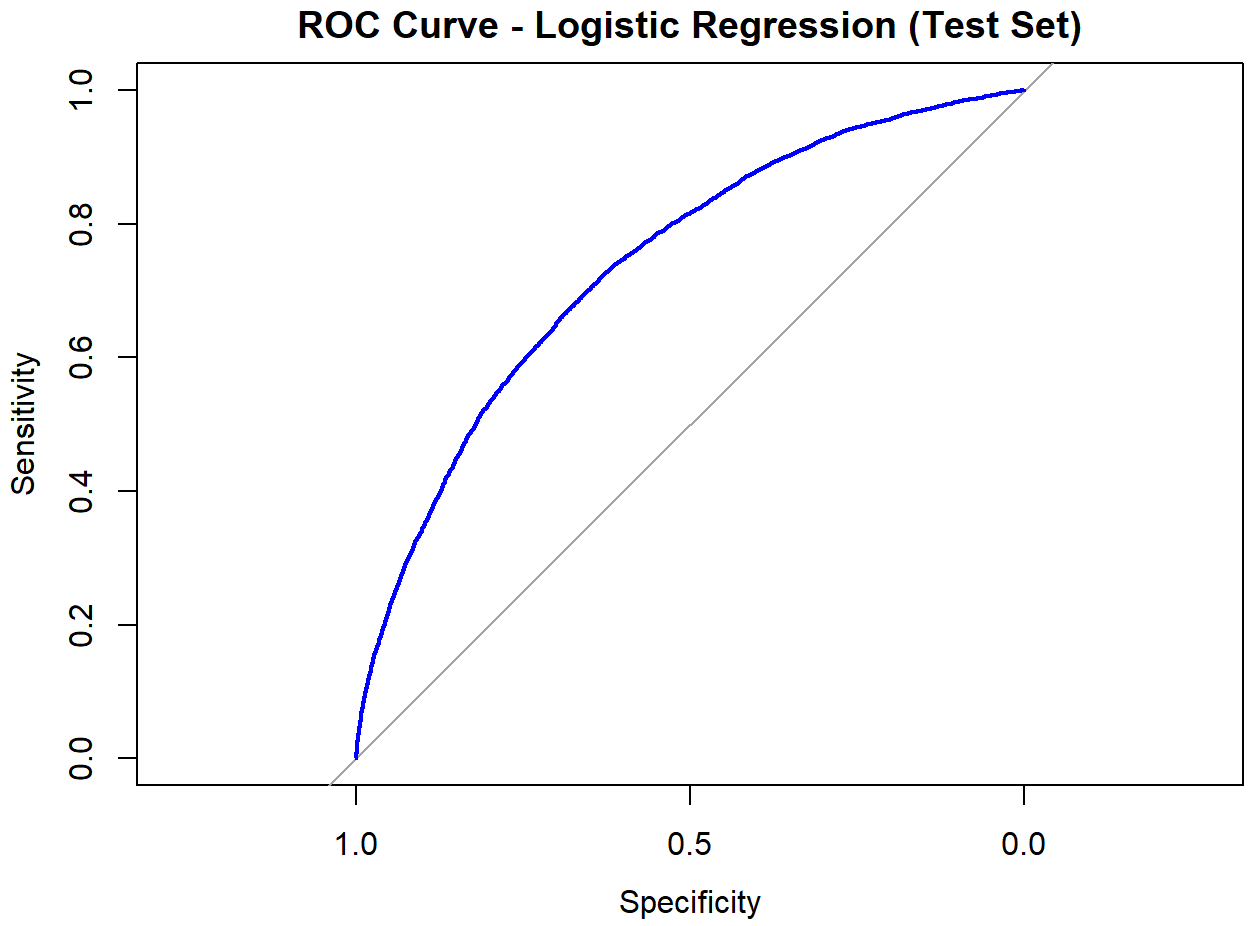
AUC - Logistic Model - Test set

```
# Predict probabilities on the test dataset
pred_test_logistic <- predict(logistic_model, newdata = dt_test, type = "prob")  # Generate predic

# Calculate the ROC and AUC for the test set (with "Class1" as the positive class)
roc_test_logistic <- roc(dt_test$TARGET, pred_test_logistic$Class1, levels = c("Class0", "Class1")
```

```r
# Calculate AUC
auc_test_logistic <- auc(roc_test_logistic)  # Calculate AUC
cat("AUC Test set - Logistic:", round(auc_test_logistic, 2), "\n")  # Print the AUC value for the
```

```
AUC Test set - Logistic: 0.74
```

```r
# Plot the ROC curve for the test set
plot(roc_test_logistic, col = "blue", lwd = 2, main = "ROC Curve - Logistic Regression (Test Set)"
```



Confusion Matrix - Logistic Regression Test Set

```r
# Predict class labels for the test set using the logistic regression model
predictClassesTestLogistic <- predict(logistic_model, dt_test)

# Calculate the confusion matrix for the logistic regression model on the test set
conf_matrix_test_logistic <- confusionMatrix(data = predictClassesTestLogistic,
                                             reference = dt_test$TARGET,
                                             positive = "Class1")  # Specify "Class1" as the posit
```

```
# Display the confusion matrix in the console
print(conf_matrix_test_logistic)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Class0 Class1
    Class0  12793   4541
    Class1   1607   2659

               Accuracy : 0.7154
                 95% CI : (0.7093, 0.7214)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2869

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

            Sensitivity : 0.3693
            Specificity : 0.8884
         Pos Pred Value : 0.6233
         Neg Pred Value : 0.7380
             Prevalence : 0.3333
         Detection Rate : 0.1231
   Detection Prevalence : 0.1975
      Balanced Accuracy : 0.6289

       'Positive' Class : Class1
```

Results for the logistic model (test set) include an AUC of 0.74, accuracy of 71%, sensitivity (true positive for Class 1) of 37%, and specificity (true negative for Class 0) of 89%. The logistic regression model provides a reasonable baseline for distinguishing between defaulters and non-defaulters, being better than a classification based on the no-information rate.

5.2 Random Forest

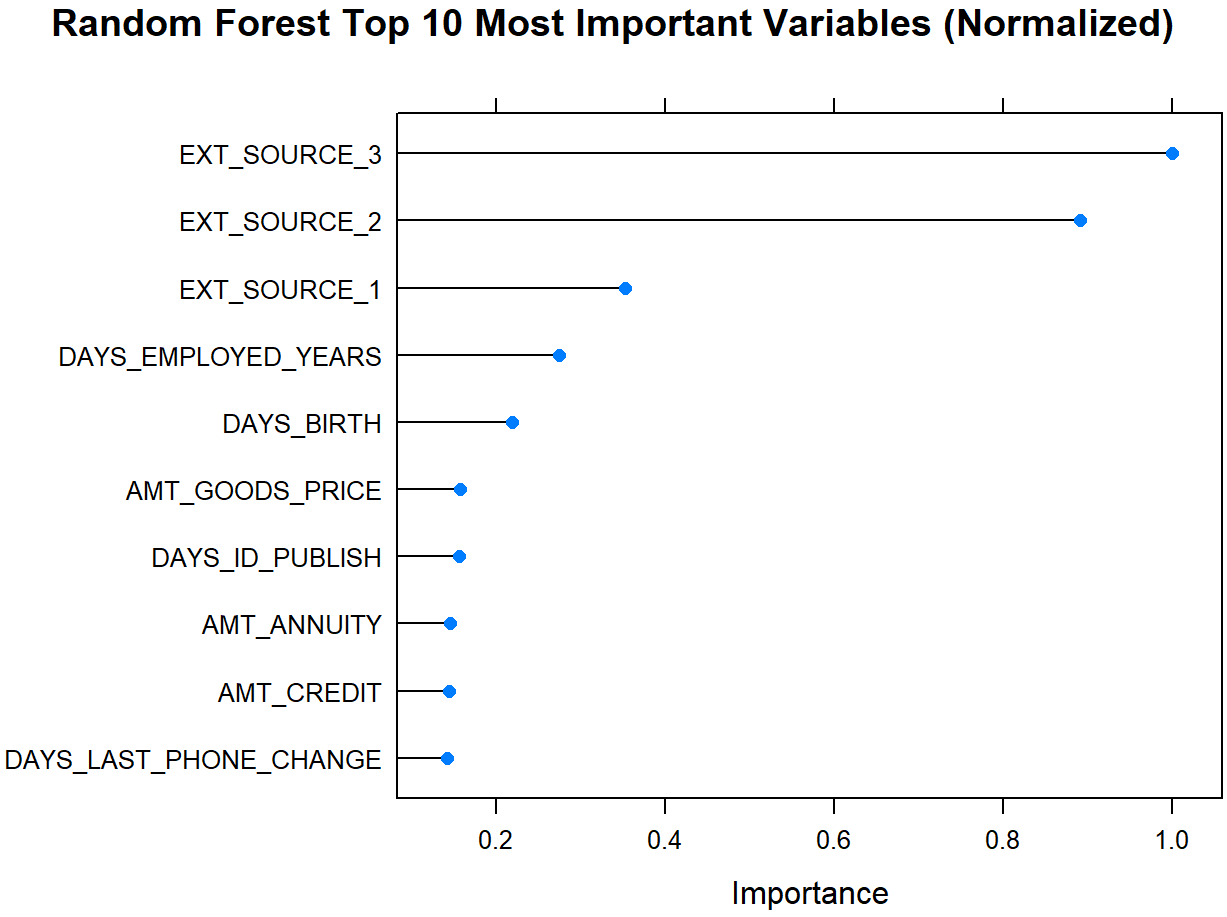
In the Random Forest model setup, I had to make several adjustments as the model was showing signs of overfitting to the training set. Key hyperparameters included experimenting with different values for the number of features considered at each split to balance model complexity and accuracy. I also explored different split criteria and modified the minimum node size to control tree depth and reduce overfitting. The model was trained using repeated cross-validation with 10 folds and 2 repeats. Class probabilities were calculated for performance evaluation, and parallel processing was enabled to speed up training. Also I used the “ranger” method for Random Forest modeling because it’s efficient with large datasets and faster than traditional implementations, plus it supports parallel processing.

```
# Set time measure
#tic()

#random_forest_model <- train(
#  x = setDT(dt_train)[, -c('TARGET')],  # Remove the target variable from the features
#  y = dt_train$TARGET,  # Define the target variable
#  method = 'ranger',  # Using the ranger package for Random Forest
#  preProc = c('center', 'scale'),  # Standardize the data by centering and scaling
#  tuneGrid = expand.grid(
#    .mtry = c(2, 4, 6),  # Number of features considered at each split
#    .splitrule = c("gini", "extratrees"),  # Experiment with different split criteria
#    .min.node.size = c(50, 150, 250)),  # Adjust to reduce tree complexity (minimum node size)

#  metric = "ROC",  # Evaluate the model using the ROC metric
#  trControl = traincntrl,  # Use cross-validation for model evaluation
#  num.trees = 120,  # Number of trees
#  importance = "impurity")  # Request calculation of variable importance

# End time measure
#toc()

# Save the random forest model to an .RData file
#save(random_forest_model, file = "D:/mymodels/random_forest_model.RData")
load("D:/mymodels/random_forest_model.RData")

# View the best hyperparameters
best_hyperparameters <- random_forest_model$bestTune
print(best_hyperparameters)
```

```
   mtry splitrule min.node.size
15    6      gini           250
```

The best hyperparameters obtained for the model are: mtry = 15, which means that 15 variables were selected for each split; splitrule = “gini”, indicating that the splits were made based on the Gini index; and min.node.size = 250, which sets the minimum number of observations required in each node before it can be split.

```
# Extract the variable importance values from the trained model
importance_rf <- varImp(random_forest_model, scale = FALSE)

# Normalize importance values between 0 and 1
importance_rf$importance <- importance_rf$importance / max(importance_rf$importance)

# Plot normalized importance of the top 10 most important variables
plot(importance_rf, top = 10, main = "Random Forest Top 10 Most Important Variables (Normalized)")
```



These are the top 10 most important variables in the model, with EXT_SOURCE_3 being the most important, followed by EXT_SOURCE_2 and EXT_SOURCE_1. The other variables, such as DAYS_EMPLOYED_YEARS and DAYS_BIRTH, have progressively lower importance in comparison.

Among these 10 predictors, 9 are listed among the top 10 most important predictors identified through the RFE process.

Train set Metrics

```
# Predict probabilities on the train dataset using the random forest model
pred_train_rf <- predict(random_forest_model, newdata = dt_train, type = "prob")  # Generate predi

# ROC and AUC for the train set using the random forest model
roc_train_rf <- roc(dt_train$TARGET, pred_train_rf$Class1, levels = c("Class0", "Class1"), directi
auc_train_rf <- auc(roc_train_rf)  # Calculate AUC
cat("AUC Train set - Random Forest:", round(auc_train_rf, 2), "\n")
```

```
AUC Train set - Random Forest: 0.84
```

```
# Predict class labels for the train set using the random forest model
predictClassesTrainRF <- predict(random_forest_model, dt_train)

# Calculate Accuracy for the train set using the random forest model
```

```r
accuracy_train_rf <- mean(predictClassesTrainRF == dt_train$TARGET)
cat("Accuracy Train set - Random Forest:", round(accuracy_train_rf, 2), "\n")
```

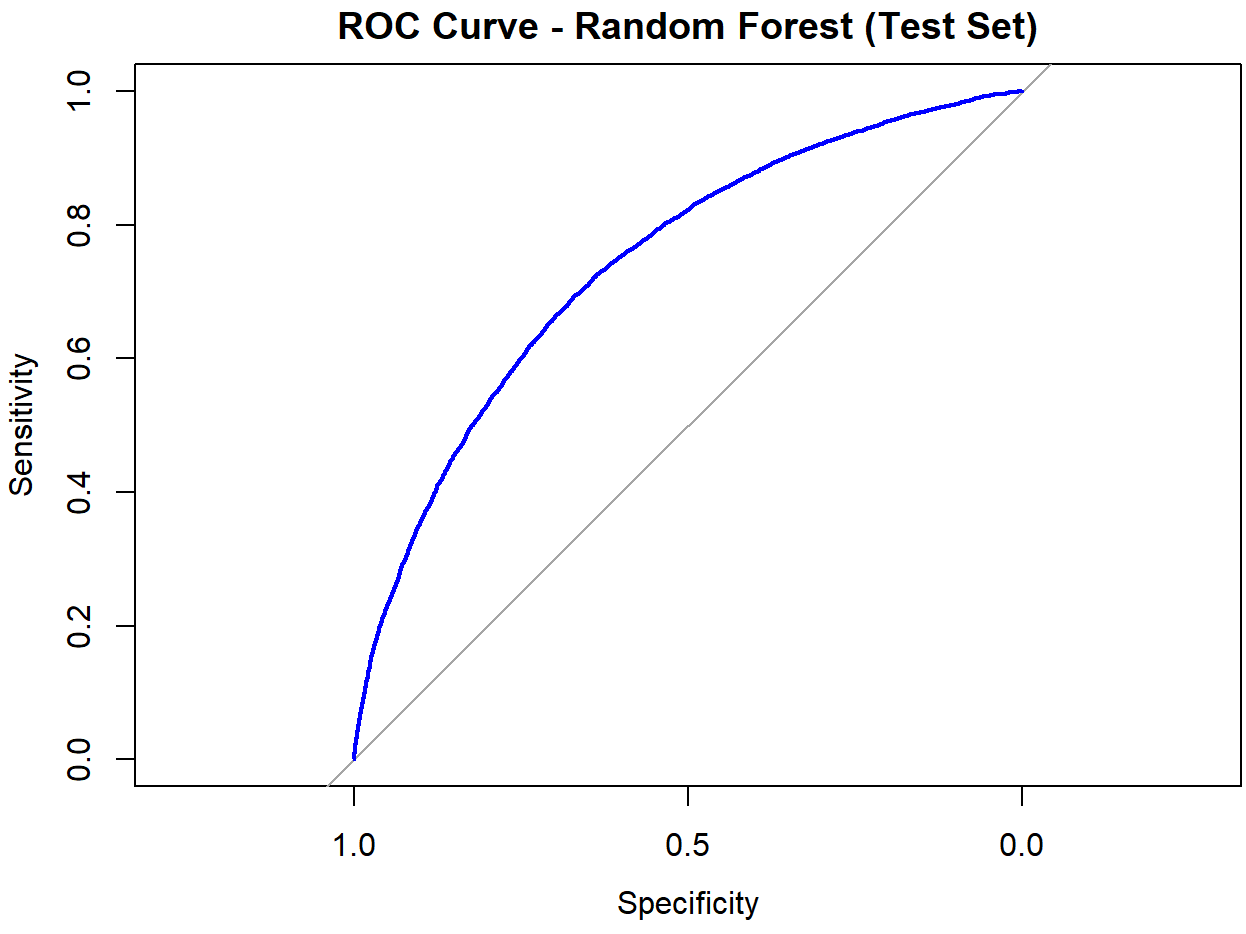
```
Accuracy Train set - Random Forest: 0.76
```

```r
# Generate confusion matrix for the random forest model on the train set
conf_matrix_train_rf <- confusionMatrix(data = predictClassesTrainRF,
                                        reference = dt_train$TARGET,
                                        positive = "Class1")  # "Class1" as the positive class
```

Initially with maximum overfitting to the train set, the model now has better balance, aiming for greater generalization.

AUC - Random Forest - Test Set

```r
# Predict probabilities on the test set using the random forest model
pred_test_rf <- predict(random_forest_model, newdata = dt_test, type = "prob")  # Generate predict

# Calculate the ROC and AUC for the test set using the random forest model (with "Class1" as the p
roc_test_rf <- roc(dt_test$TARGET, pred_test_rf$Class1, levels = c("Class0", "Class1"), direction

# Plot the ROC curve for the test set
plot(roc_test_rf, col = "blue", lwd = 2, main = "ROC Curve - Random Forest (Test Set)")  # Plot RO
```



```r
# Calculate AUC for the test set
auc_test_rf <- auc(roc_test_rf)

# Display AUC in the console
cat("AUC Test set - Random Forest:", round(auc_test_rf, 2), "\n")  # Print the AUC value
```

```
AUC Test set - Random Forest: 0.74
```

Confusion Matrix - Random Forest - Test Set

```r
# Predict class labels for the test set using the random forest model
predictClassesTestRF <- predict(random_forest_model, dt_test)

# Calculate the confusion matrix for the random forest model on the test set
conf_matrix_test_rf <- confusionMatrix(data = predictClassesTestRF,
                                       reference = dt_test$TARGET,
                                       positive = "Class1")  # Specify "Class1" as the positive cl

# Display the confusion matrix in the console
print(conf_matrix_test_rf)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Class0 Class1
    Class0  13197   4912
    Class1   1203   2288

               Accuracy : 0.7169
                 95% CI : (0.7108, 0.7229)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2689

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

            Sensitivity : 0.3178
            Specificity : 0.9165
         Pos Pred Value : 0.6554
         Neg Pred Value : 0.7288
             Prevalence : 0.3333
         Detection Rate : 0.1059
   Detection Prevalence : 0.1616
      Balanced Accuracy : 0.6171

       'Positive' Class : Class1
```

Results for the Random Forest model (test set) include an AUC of 0.74, accuracy of 72%, sensitivity (true positive for Class 1) of 32%, and specificity (true negative for Class 0) of 92%. The Random Forest model offers a slight improvement in specificity compared to logistic regression, making it more effective at identifying non-defaulters. However, its lower sensitivity means it may miss more defaulters. While less interpretable than logistic regression, Random Forest can capture non-linear relationships and interactions among variables, which is useful for complex datasets, but it does not show a significant performance gain in this case.

5.3 XGBoost Model

The XGBoost model was optimized in multiple stages to improve computational performance. The dataset was split into training and test sets, then transformed into the xgb.DMatrix format. Hyperparameter tuning was performed in rounds, adjusting key parameters like max_depth, min_child_weight, and eta to maximize the AUC score. The second round fine-tuned sampling parameters (subsample, colsample_bytree) to reduce overfitting, followed by regularization parameters (gamma, lambda, alpha) in the final round.

Finally, the number of boosting iterations (nrounds) was optimized, with cross-validation used throughout to identify the best parameter combination. This process improved performance while minimizing overfitting.

```
# Prepare data for XGBoost (separated from training and test datasets)
dtrain_xgb <- xgb.DMatrix(
  data = as.matrix(select(dt_train, -TARGET)),
  label = as.numeric(dt_train$TARGET) - 1)  # Convert the target to binary (0 and 1)

dtest_xgb <- xgb.DMatrix(
  data = as.matrix(select(dt_test, -TARGET)),
  label = as.numeric(dt_test$TARGET) - 1)    # Same conversion for the test set

# Set time measure
#tic()

# Function to evaluate model parameters using cross-validation
evaluate_params <- function(params, dtrain, max_trees = 150) {
  cv_results <- xgb.cv(
    params = params,
    data = dtrain,
    nrounds = max_trees,
    nfold = 10,
    early_stopping_rounds = 50,
    verbose = 0)

  best_auc <- max(cv_results$evaluation_log$test_auc_mean)  # Use AUC for evaluation
  best_nround <- which.max(cv_results$evaluation_log$test_auc_mean)
  return(list(auc = best_auc, nrounds = best_nround)) }

# Initial parameters regularization to avoid overfitting
base_params <- list(
```

```r
  objective = "binary:logistic",  # Binary classification
  eval_metric = "auc",  # AUC is more robust for binary classification, especially with imbalanced
  tree_method = "hist",  # Efficient tree building for large datasets
  max_depth = 5,  # Limited tree depth to prevent overfitting
  min_child_weight = 5,  # Minimum number of data points required in a child node
  eta = 0.05,  # Learning rate to prevent large steps in training
  gamma = 0.1,  # Regularization parameter for pruning
  lambda = 0.1,  # L2 regularization (shrinkage of leaf weights)
  alpha = 0.1,  # L1 regularization (lasso)
  subsample = 0.8,  # Random sample of training data for each tree (helps reduce overfitting)
  colsample_bytree = 0.8)  # Random sampling of features for each tree

# Hyperparameter tuning for basic tree parameters (max_depth, min_child_weight, eta)
tune_round1 <- function(dtrain) {
  tune_grid <- expand.grid(
    max_depth = c(3, 5, 7),  # Trying different tree depths
    min_child_weight = c(20, 40, 60),
    eta = c(0.05, 0.1, 0.5))

  best_params <- list()
  best_auc <- 0
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    params <- list(
      objective = "binary:logistic",
      eval_metric = "auc",
      max_depth = tune_grid$max_depth[i],
      min_child_weight = tune_grid$min_child_weight[i],
      eta = tune_grid$eta[i])

    result <- evaluate_params(params, dtrain)
    if (result$auc > best_auc) {
      best_auc <- result$auc
      best_params <- params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, auc = best_auc)) }

# Tuning for sampling parameters (subsample and colsample_bytree)
tune_round2 <- function(base_params, dtrain) {
  tune_grid <- expand.grid(
    subsample = seq(0.6, 1, 0.1),
    colsample_bytree = seq(0.6, 1, 0.1))

  best_params <- base_params
  best_auc <- 0
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    current_params <- base_params
```

```
    current_params$subsample <- tune_grid$subsample[i]
    current_params$colsample_bytree <- tune_grid$colsample_bytree[i]
    result <- evaluate_params(current_params, dtrain)
    if (result$auc > best_auc) {
      best_auc <- result$auc
      best_params <- current_params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, auc = best_auc)) }

# Fine-tune the regularization parameters (gamma, lambda, alpha)
tune_round3 <- function(base_params, dtrain) {
  tune_grid <- expand.grid(
    gamma = c(0, 0.1, 0.3, 0.5, 1),
    lambda = c(0.01, 0.1, 1),
    alpha = c(0.01, 0.1, 1))

  best_params <- base_params
  best_auc <- 0
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    current_params <- base_params
    current_params$gamma <- tune_grid$gamma[i]
    current_params$lambda <- tune_grid$lambda[i]
    current_params$alpha <- tune_grid$alpha[i]
    result <- evaluate_params(current_params, dtrain)
    if (result$auc > best_auc) {
      best_auc <- result$auc
      best_params <- current_params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, auc = best_auc)) }

# Fine-tune the number of trees by testing around the optimal number of rounds
fine_tune_trees <- function(params, dtrain, current_nround) {
  test_rounds <- seq(max(1, current_nround - 100), current_nround + 100, by = 50)
  best_auc <- 0
  best_nround <- current_nround

  for (n in test_rounds) {
    cv_results <- xgb.cv(
      params = params,
      data = dtrain,
      nrounds = n,
      nfold = 5,  # Reduced folds for faster execution
      early_stopping_rounds = 50,
      verbose = 0)

    current_auc <- max(cv_results$evaluation_log$test_auc_mean)  # Use AUC for evaluation
    if (current_auc > best_auc) {
```

```
      best_auc <- current_auc
      best_nround <- n } }

  return(list(params = params, nrounds = best_nround, auc = best_auc))}

# Train the final model with the best parameters
final_results <- fine_tune_trees(params = base_params, dtrain = dtrain_xgb, current_nround = 150)

xgboost_model <- xgb.train(
  params = final_results$params,  # Best parameters from tuning
  data = dtrain_xgb,
  nrounds = final_results$nrounds,
  watchlist = list(train = dtrain_xgb, test = dtest_xgb),
  verbose = 0)

# End time measure
toc()

# Save the XGBoost Faster model in a single .RData file in D: drive
save(xgboost_model, file = "D:/mymodels/xgboost_model.RData")
#load("D:/mymodels/xgboost_model_faster.RData")
```

Best hyperparameters and Predictors

```
# Displaying Final Model Details
#cat("XGBoost Model Summary:\n")
#cat("Number of Trees:", final_results$nrounds, "\n")  # Retrieve the number of trees from the mod

# Variable Importance
importance <- xgb.importance(
  feature_names = colnames(select(dt_train, -TARGET)),  # Exclude the TARGET variable from the fea
  model = xgboost_model  # Use the trained XGBoost model to extract feature importance
)

# Format the importance values to two decimal places for better readability
importance[, `:=`(Gain = round(Gain, 2), Cover = round(Cover, 2), Frequency = round(Frequency, 2))

# Create an interactive table to display all the predictors and their importance values
datatable(importance, options = list(pageLength = 10),
          caption = "XGBoost Feature Importance") %>%
  formatPercentage('Gain', 2) %>%  # Format the 'Gain' values as percentages with 2 decimal places
  formatPercentage('Cover', 2) %>%  # Format the 'Cover' values as percentages with 2 decimal plac
  formatPercentage('Frequency', 2)  # Format the 'Frequency' values as percentages with 2 decimal
```

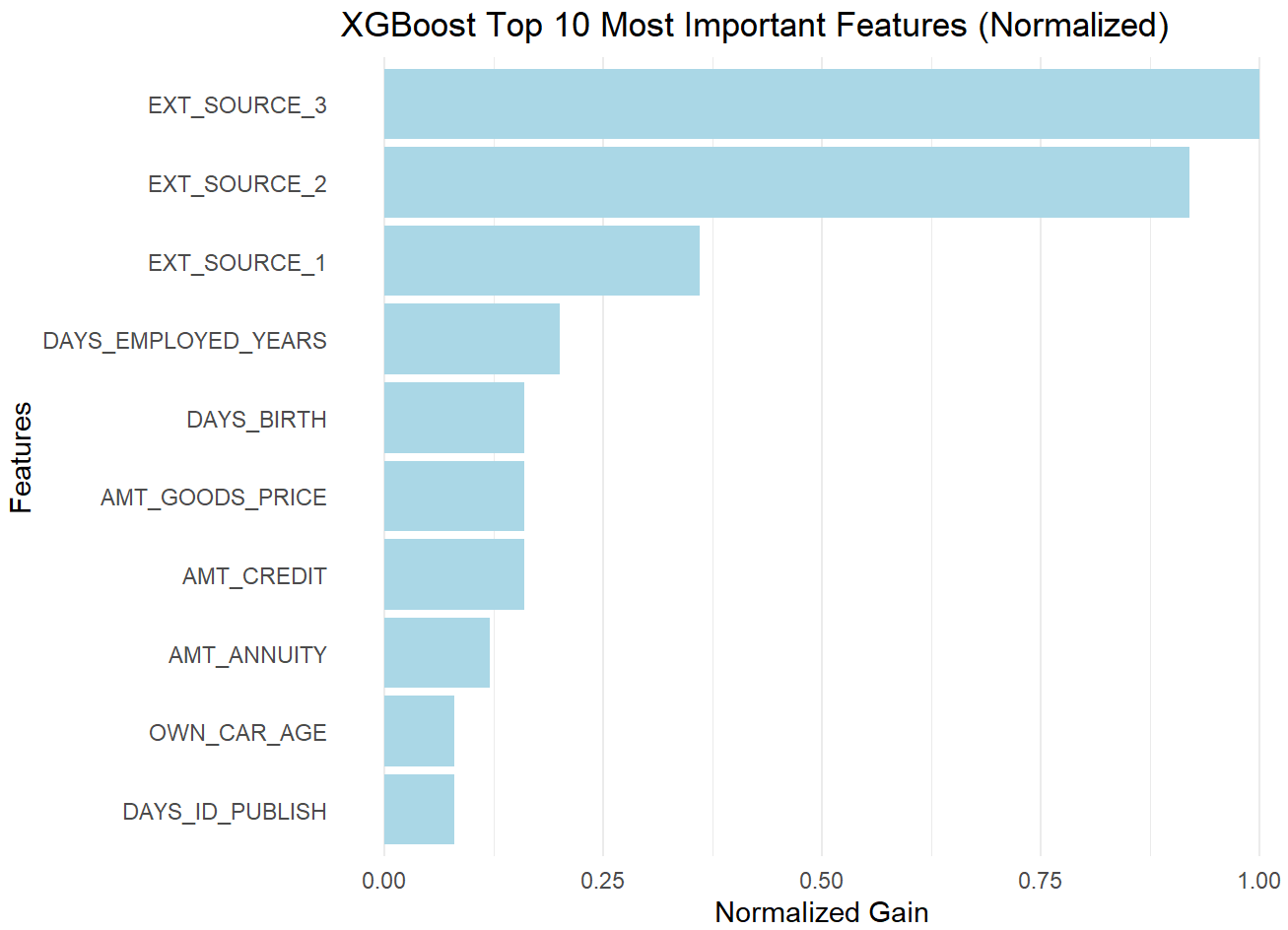
Show 10 entries Search:

XGBoost Feature Importance

	Feature	Gain	Cover	Frequency
1	EXT_SOURCE_3	25.00%	11.00%	8.00%
2	EXT_SOURCE_2	23.00%	11.00%	7.00%
3	EXT_SOURCE_1	9.00%	9.00%	7.00%
4	DAYS_EMPLOYED_YEARS	5.00%	5.00%	5.00%
5	AMT_CREDIT	4.00%	6.00%	6.00%
6	DAYS_BIRTH	4.00%	7.00%	7.00%
7	AMT_GOODS_PRICE	4.00%	7.00%	5.00%
8	AMT_ANNUITY	3.00%	5.00%	6.00%
9	OWN_CAR_AGE	2.00%	5.00%	3.00%
10	DAYS_ID_PUBLISH	2.00%	4.00%	6.00%

Showing 1 to 10 of 50 entries Previous 1 2 3 4 5 Next

```
# Normalize the feature importances to a scale of 0 to 1 for the Gain column
importance$NormalizedGain <- importance$Gain / max(importance$Gain)

# Select the top 10 most important features based on the normalized Gain
top_10_importance <- importance[order(-NormalizedGain)][1:10]

# Plot Variable Importance for the top 10 features
ggplot(top_10_importance, aes(x = reorder(Feature, NormalizedGain), y = NormalizedGain)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  coord_flip() +
  labs(title = "XGBoost Top 10 Most Important Features (Normalized)",
       x = "Features",
       y = "Normalized Gain") +
  theme_minimal() +
  theme(panel.grid.major.y = element_blank(),  # Remove horizontal grid lines
        panel.grid.minor.y = element_blank())  # Remove minor horizontal grid lines
```



The final model was trained with 250 trees, and the AUC metric was used to evaluate its performance. The feature importance analysis revealed that EXT_SOURCE_3 and EXT_SOURCE_2 are the most relevant variables, with contributions of 24% and 23%, respectively.

Additionally, “Cover” and “Frequency” metrics provide insight into how these features contribute to the model. “Cover” measures the proportion of samples that each feature influences, indicating how broadly the feature affects the data. “Frequency” shows how often each feature is used to split the data in the decision trees. EXT_SOURCE_3 and EXT_SOURCE_2 not only have high contributions to the model but also exhibit higher cover and frequency, meaning they are both widely and frequently utilized in the model’s decision-making process.

Other important variables include EXT_SOURCE_1, DAYS_EMPLOYED_YEARS, and DAYS_BIRTH, which also play significant roles in the model’s predictions. These variables help model the relationship between customer characteristics and the likelihood of success in classifying the TARGET variable.

Only OWN_CAR_AGE and NAME_EDUCATION_TYPE_higher_education are not among the top 10 most important variables from the RFE.

Train Set Metrics

```
# Predict probabilities on the train dataset using the XGBoost model
pred_train_xgb <- predict(xgboost_model, newdata = dtrain_xgb)  # Generate predicted probabilities
```

```
# ROC and AUC for the train set using the XGBoost model
roc_train_xgb <- roc(dt_train$TARGET, pred_train_xgb, levels = c("Class0", "Class1"), direction =
auc_train_xgb <- auc(roc_train_xgb)  # Calculate AUC
cat("AUC Train set - XGBoost:", round(auc_train_xgb, 2), "\n")
```

```
AUC Train set - XGBoost: 0.8
```

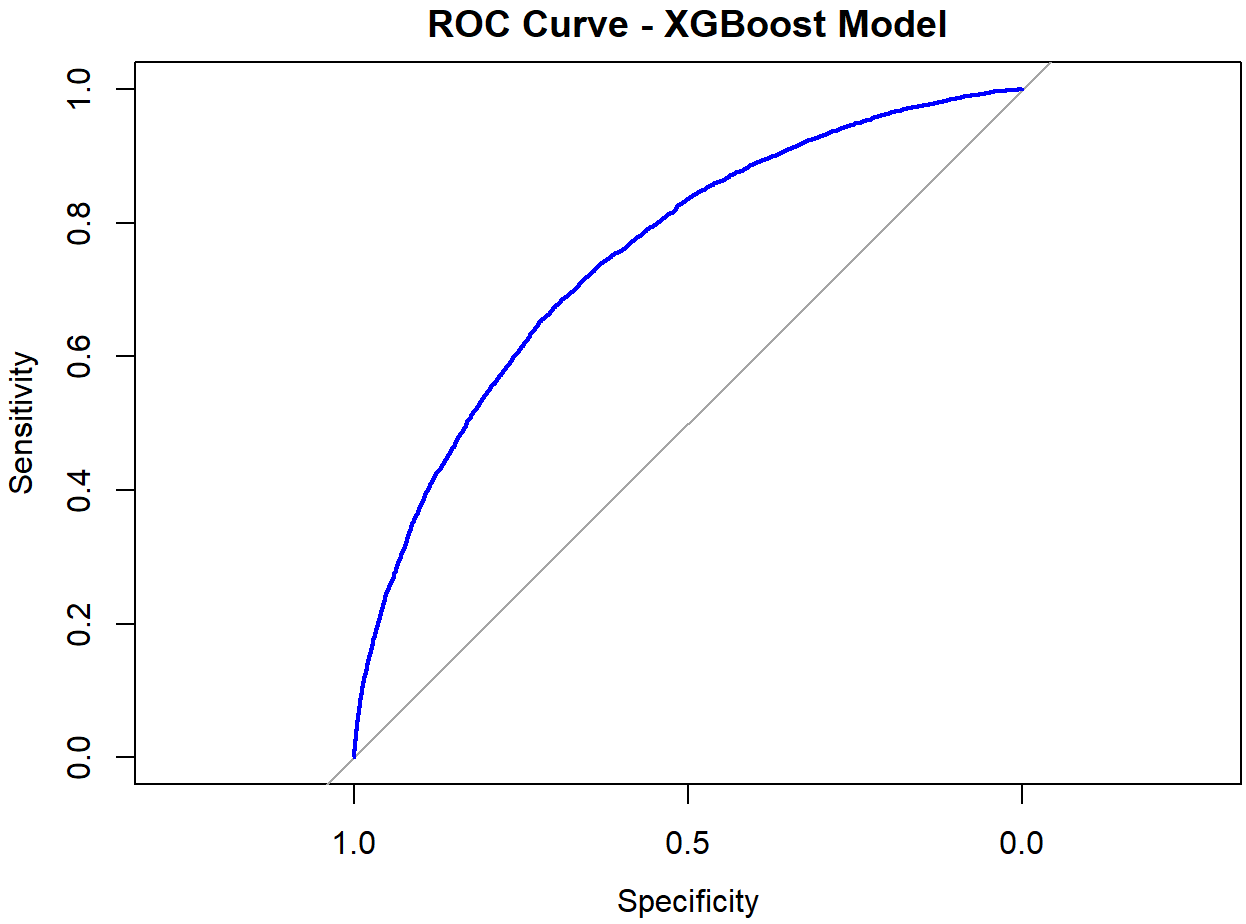
```
# Predict class labels for the train set using the XGBoost model
predictClassesTrainXGB <- ifelse(pred_train_xgb > 0.5, "Class1", "Class0")  # Convert probabilitie

# Calculate Accuracy for the train set using the XGBoost model
accuracy_train_xgb <- mean(predictClassesTrainXGB == dt_train$TARGET)
cat("Accuracy Train set - XGBoost:", round(accuracy_train_xgb, 2), "\n")
```

```
Accuracy Train set - XGBoost: 0.75
```

```
# Generate confusion matrix for the XGBoost model on the train set
conf_matrix_train_xgb <- confusionMatrix(data = factor(predictClassesTrainXGB, levels = c("Class0"
                                         reference = dt_train$TARGET,
                                         positive = "Class1")  # "Class1" as the positive class
```

AUC - XGBoost Model

```
# Predict class probabilities on the test dataset using the XGBoost model
pred_test_xgb <- predict(xgboost_model, newdata = dtest_xgb)  # Generate predicted probabilities f

# For binary classification, probabilities for Class1
pred_test_xgb_class1 <- pred_test_xgb  # Probability for Class 1
pred_test_xgb_class0 <- 1 - pred_test_xgb_class1  # Probability for Class 0

# Calculate ROC and AUC for the test set using the XGBoost model
roc_test_xgb <- roc(dt_test$TARGET, pred_test_xgb_class1, levels = c("Class0", "Class1"), directio

# Plot the ROC curve
plot(roc_test_xgb, col = "blue", lwd = 2, main = "ROC Curve - XGBoost Model")  # Plot ROC curve
```




```
# Calculate and display the AUC value in the console
auc_test_xgb <- auc(roc_test_xgb)
cat("AUC Test set - XGBoost:", auc_test_xgb, "\n")  # Print the AUC value
```

```
AUC Test set - XGBoost: 0.7518127
```

Confusion Matrix - XGBoost Model

```
# Predict class labels on the test dataset using the trained XGBoost model
predictClassesTestXGB <- ifelse(pred_test_xgb_class1 > 0.5, "Class1", "Class0")  # Convert probabi

# Calculate the confusion matrix to evaluate the XGBoost model's performance
conf_matrix_test_xgb <- confusionMatrix(data = factor(predictClassesTestXGB, levels = c("Class0",
                                        reference = dt_test$TARGET,        # True class labels fr
                                        positive = "Class1")               # Specify the positive

# Display the confusion matrix
print(conf_matrix_test_xgb)
```

```
Confusion Matrix and Statistics

          Reference
Prediction Class0 Class1
```

```
   Class0  12724   4239
   Class1   1676   2961

               Accuracy : 0.7262
                 95% CI : (0.7202, 0.7321)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3237

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

            Sensitivity : 0.4113
            Specificity : 0.8836
         Pos Pred Value : 0.6386
         Neg Pred Value : 0.7501
             Prevalence : 0.3333
         Detection Rate : 0.1371
   Detection Prevalence : 0.2147
      Balanced Accuracy : 0.6474

       'Positive' Class : Class1
```

Results for the XGBoost model (test set) include an AUC of 0.75, accuracy of 73%, sensitivity of 42%, and specificity of 88%. The XGBoost model outperforms both logistic regression and Random Forest, offering the highest AUC and accuracy with a balanced performance in sensitivity and specificity. XGBoost's ability to capture complex patterns and interactions likely contributed to this improved performance. Additionally, the modeling process was much faster than with Random Forest, given the way it was implemented.

5.4 LightGBM Model

The LightGBM model was optimized by tuning key parameters in multiple stages. First, the dataset was split into training and test sets and converted to the lgb.Dataset format. The model was then fine-tuned by adjusting tree parameters (max_depth, min_data_in_leaf, learning_rate), followed by sampling and regularization parameters to minimize overfitting. Cross-validation was used to determine the optimal number of boosting iterations. The training process was faster than Random Forest, showcasing LightGBM's efficiency while maintaining strong performance.

```r
# Load the data_pre_sample dataset from the specified file
load("D:/mymodels/data_pre_sample.RData")

# Set the seed for reproducibility
set.seed(seed)

# Create a partition for training and testing sets; 70% for training
inTrain <- createDataPartition(data_pre_sample$TARGET, p = 0.7, list = FALSE)

# Create the training and testing subsets
```

```r
dt_train <- data_pre_sample[inTrain, ]
dt_test <- data_pre_sample[-inTrain, ]

# Convert data to lgb.Dataset
dtrain_lgb <- lgb.Dataset(data = as.matrix(dt_train[, -which(names(dt_train) == "TARGET")]),
                          label = dt_train$TARGET)

dtest_lgb <- lgb.Dataset(data = as.matrix(dt_test[, -which(names(dt_test) == "TARGET")]),
                         label = dt_test$TARGET, free_raw_data = FALSE)

# Set time measure
tic()

# Base parameters for lightgbm
base_params_lgb <- list(
  objective = "binary",
  metric = "binary_error",
  boosting_type = "gbdt",
  num_threads = 4)


# Initial model evaluation: cross-validation to find best nrounds
evaluate_params_lgb <- function(params, dtrain, max_trees = 2000) {
  cv_results <- lgb.cv(
    params = params,
    data = dtrain,
    nrounds = max_trees,
    nfold = 10,
    early_stopping_rounds = 50,
    verbose = -1)

  best_rmse <- min(cv_results$evaluation_log$valid_rmse)
  best_nround <- which.min(cv_results$evaluation_log$valid_rmse)
  return(list(rmse = best_rmse, nrounds = best_nround)) }

# Tune basic tree parameters
tune_round1_lgb <- function(dtrain) {
  tune_grid <- expand.grid(
    max_depth = c(3, 5, 7),
    min_data_in_leaf = c(10, 20, 30),
    learning_rate = c(0.01, 0.05, 0.1))

  best_params <- list()
  best_rmse <- Inf
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    params <- list(
      objective = "binary",
      metric = "binary_error",
      max_depth = tune_grid$max_depth[i],
```

```
      min_data_in_leaf = tune_grid$min_data_in_leaf[i],
      learning_rate = tune_grid$learning_rate[i])

    result <- evaluate_params_lgb(params, dtrain)
    if (result$rmse < best_rmse) {
      best_rmse <- result$rmse
      best_params <- params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, rmse = best_rmse))}

# Tune sampling parameters
tune_round2_lgb <- function(base_params, dtrain) {
  tune_grid <- expand.grid(
    subsample = seq(0.6, 1, 0.1),
    colsample_bytree = seq(0.6, 1, 0.1))

  best_params <- base_params
  best_rmse <- Inf
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    current_params <- base_params
    current_params$subsample <- tune_grid$subsample[i]
    current_params$colsample_bytree <- tune_grid$colsample_bytree[i]
    result <- evaluate_params_lgb(current_params, dtrain)
    if (result$rmse < best_rmse) {
      best_rmse <- result$rmse
      best_params <- current_params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, rmse = best_rmse)) }

# Fine-Tune regularization parameters
tune_round3_lgb <- function(base_params, dtrain) {
  tune_grid <- expand.grid(
    lambda_l1 = c(0.01, 0.1, 1),
    lambda_l2 = c(0.01, 0.1, 1),
    min_gain_to_split = c(0.01, 0.1, 0.5))

  best_params <- base_params
  best_rmse <- Inf
  best_nround <- 0

  for (i in 1:nrow(tune_grid)) {
    current_params <- base_params
    current_params$lambda_l1 <- tune_grid$lambda_l1[i]
    current_params$lambda_l2 <- tune_grid$lambda_l2[i]
    current_params$min_gain_to_split <- tune_grid$min_gain_to_split[i]
    result <- evaluate_params_lgb(current_params, dtrain)
    if (result$rmse < best_rmse) {
```

```
      best_rmse <- result$rmse
      best_params <- current_params
      best_nround <- result$nrounds } }

  return(list(params = best_params, nrounds = best_nround, rmse = best_rmse)) }

# Fine-tune the number of trees
fine_tune_trees_lgb <- function(params, dtrain, current_nround) {
  test_rounds <- seq(max(1, current_nround - 200), current_nround + 200, by = 50)
  best_rmse <- Inf
  best_nround <- current_nround

  for (n in test_rounds) {
    cv_results <- lgb.cv(
      params = params,
      data = dtrain,
      nrounds = n,
      nfold = 5,
      early_stopping_rounds = 50,
      verbose = -1)

    current_rmse <- min(cv_results$evaluation_log$valid_rmse)
    if (current_rmse < best_rmse) {
      best_rmse <- current_rmse
      best_nround <- n } }

# Ensure final result is in the expected list format
  return(list(params = params, nrounds = best_nround, rmse = best_rmse)) }

# Training the final model with the best parameters
final_results_lgb <- fine_tune_trees_lgb(params = base_params_lgb, dtrain = dtrain_lgb, current_nr

lightgbm_model <- lgb.train(
  params = final_results_lgb$params,  # Using best params from tuning
  data = dtrain_lgb,
  nrounds = final_results_lgb$nrounds,
  valids = list(test = dtest_lgb),
  verbose = -1)

# End time measure
toc()
```

```
66.7 sec elapsed
```

```
# Save the LightGBM model
save(lightgbm_model, file = "D:/mymodels/lightgbm_model.RData")
# Load the saved LightGBM model
#load("D:/mymodels/lightgbm_model.RData")
```

Best hyperparameters and Predictors

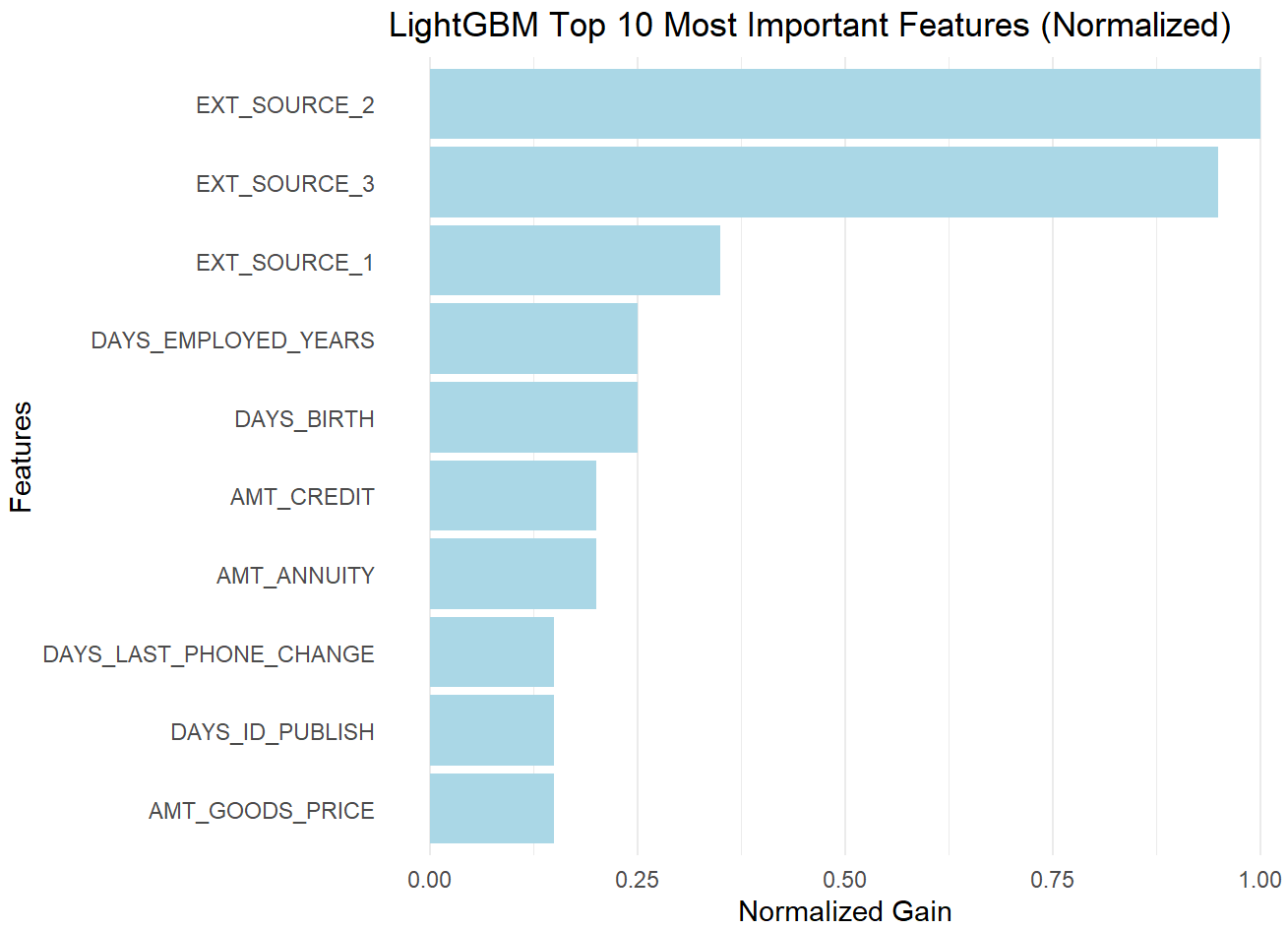
```r
# Displaying Final Model Details for LightGBM
#cat("LightGBM Model Summary:\n")
#cat("Best Number of Trees:", final_results_lgb$nrounds, "\n")  # Best number of trees
#cat("Best Parameters:\n")

# Limiting the number of decimal places to 2 for the parameters
#print(lapply(final_results_lgb$params, function(x) if (is.numeric(x)) round(x, 2) else x))  # Dis

# Variable Importance
importance <- lgb.importance(lightgbm_model)  # Get feature importance from the LightGBM model

# Limiting the importance values to 2 decimal places
importance$Gain <- round(importance$Gain, 2)
importance$Cover <- round(importance$Cover, 2)
importance$Frequency <- round(importance$Frequency, 2)

# Create an interactive table to display all the predictors and their importance values
datatable(importance, options = list(pageLength = 10),
          caption = "LightGBM Feature Importance") %>%
  formatPercentage('Gain', 2) %>%  # Format the 'Gain' values as percentages with 2 decimal places
  formatPercentage('Cover', 2) %>%  # Format the 'Cover' values as percentages with 2 decimal plac
  formatPercentage('Frequency', 2)  # Format the 'Frequency' values as percentages with 2 decimal
```

Show 10 entries Search:

LightGBM Feature Importance

	Feature	Gain	Cover	Frequency
1	EXT_SOURCE_2	20.00%	7.00%	7.00%
2	EXT_SOURCE_3	19.00%	10.00%	6.00%
3	EXT_SOURCE_1	7.00%	7.00%	6.00%
4	DAYS_EMPLOYED_YEARS	5.00%	5.00%	6.00%
5	DAYS_BIRTH	5.00%	7.00%	7.00%
6	AMT_CREDIT	4.00%	7.00%	6.00%
7	AMT_ANNUITY	4.00%	5.00%	6.00%
8	DAYS_LAST_PHONE_CHANGE	3.00%	3.00%	6.00%
9	DAYS_ID_PUBLISH	3.00%	3.00%	6.00%
10	AMT_GOODS_PRICE	3.00%	8.00%	4.00%

```r
# Normalize the feature importances to a scale of 0 to 1 for the Gain column
importance$NormalizedGain <- importance$Gain / max(importance$Gain)

# Select the top 10 most important features based on the normalized Gain
top_10_importance <- importance[order(-NormalizedGain)][1:10]

# Plot Variable Importance for the top 10 features
ggplot(top_10_importance, aes(x = reorder(Feature, NormalizedGain), y = NormalizedGain)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  coord_flip() +
  labs(title = "LightGBM Top 10 Most Important Features (Normalized)",
       x = "Features",
       y = "Normalized Gain") +
  theme_minimal() +
  theme(panel.grid.major.y = element_blank(),  # Remove horizontal grid lines
        panel.grid.minor.y = element_blank(),  # Remove minor horizontal grid lines
        axis.text.x = element_text(angle = 0))  # Ensure x-axis labels are horizontal
```



The LightGBM model was trained with 2000 boosting iterations, with binary classification as the objective and binary error as the evaluation metric. The best hyperparameters include boosting type “gbdt” and 4 threads for parallelization.

Feature importance analysis shows that EXT_SOURCE_2 and EXT_SOURCE_3 are the most influential variables, with contributions of 20% and 19%, respectively. These features also have the highest cover, indicating that they affect a significant portion of the data, and are frequently used in decision tree splits. Other important features, such as EXT_SOURCE_1, DAYS_EMPLOYED_YEARS, and DAYS_BIRTH, also contribute to the model's predictions but to a lesser extent.

For this model, only AMT_ANNUITY, DAYS_LAST_PHONE_CHANGE, and NAME_EDUCATION_TYPE_higher_education are not among the top 10 most important features in the RFE.

Train Set Metrics

```
# Predict probabilities on the train dataset
pred_train_lgb <- predict(lightgbm_model, newdata = as.matrix(dt_train[, -which(names(dt_train) ==

# ROC and AUC for the train set
roc_train_lgb <- roc(dt_train$TARGET, pred_train_lgb, levels = c(0, 1), direction = "<")
auc_train_lgb <- auc(roc_train_lgb)  # Calculate AUC
cat("AUC Train set - LightGBM:", round(auc_train_lgb, 2), "\n")
```

```
AUC Train set - LightGBM: 0.87
```

```
# Predict class labels for the train set using LightGBM model
predictClassesTrainLGB <- ifelse(pred_train_lgb > 0.5, 1, 0)

# Calculate Accuracy for the train set
accuracy_train_lgb <- mean(predictClassesTrainLGB == dt_train$TARGET)
cat("Accuracy Train set - LightGBM:", round(accuracy_train_lgb, 2), "\n")
```

```
Accuracy Train set - LightGBM: 0.8
```

```
# Generate confusion matrix for the LightGBM model on the train set
conf_matrix_train_lgb <- confusionMatrix(data = factor(predictClassesTrainLGB, levels = c(0, 1)),
                                         reference = factor(dt_train$TARGET, levels = c(0, 1)),
                                         positive = "1")  # "1" as the positive class
```

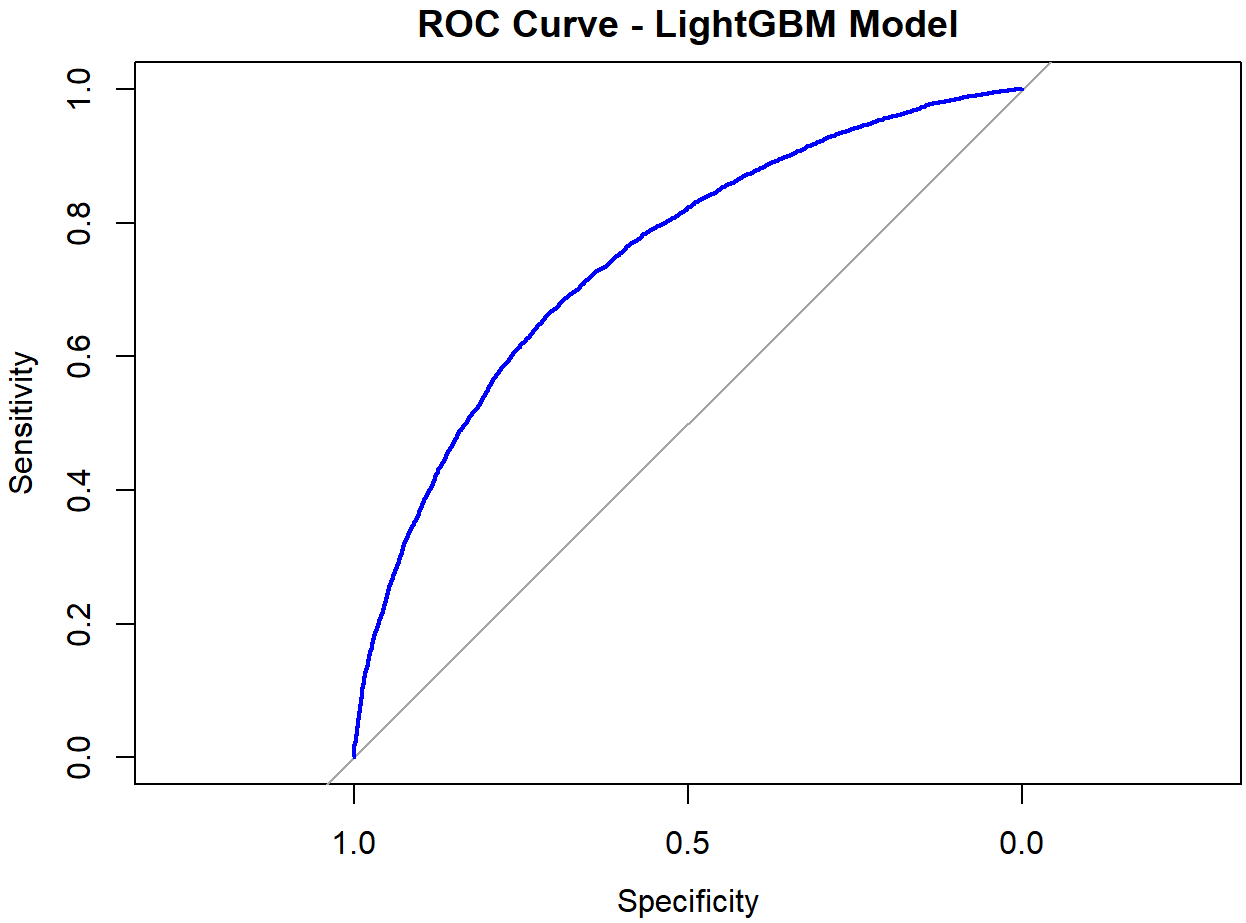
AUC - LightGBM Model

```
# Predict class probabilities on the test dataset using LightGBM
pred_test_lgb <- predict(lightgbm_model, newdata = as.matrix(dt_test[, -which(names(dt_test) == "

# For binary classification, probabilities for Class 1
pred_test_lgb_class1 <- pred_test_lgb  # Probability for Class 1
pred_test_lgb_class0 <- 1 - pred_test_lgb_class1  # Probability for Class 0

# Calculate ROC and AUC for the test set
roc_test_lgb <- roc(dt_test$TARGET, pred_test_lgb_class1, levels = c(0, 1), direction = "<")  # RO
```

```
auc_test_lgb <- auc(roc_test_lgb)  # Calculate AUC
cat("AUC Test set - LightGBM:", round(auc_test_lgb, 2), "\n")
```

```
AUC Test set - LightGBM: 0.75
```

```
# Plot the ROC curve
plot(roc_test_lgb, col = "blue", lwd = 2, main = "ROC Curve - LightGBM Model")
```



Confusion Matrix - LightGBM Model

```
# Convert probabilities to class labels (threshold 0.5 for binary classification)
pred_test_lgb_class_labels <- ifelse(pred_test_lgb_class1 > 0.5, 1, 0)

# Convert predicted labels and true labels to factors with the same levels
pred_test_lgb_class_labels <- factor(pred_test_lgb_class_labels, levels = c(0, 1))
true_labels <- factor(dt_test$TARGET, levels = c(0, 1))

# Calculate the confusion matrix to evaluate the LightGBM model's performance
conf_matrix_test_lgb <- confusionMatrix(data = pred_test_lgb_class_labels,   # Predicted class lab
                                        reference = true_labels,           # True class labels fro
                                        positive = "1")                    # Specify the positive
```

```
# Display the confusion matrix
print(conf_matrix_test_lgb)
```

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 12636  4035
         1  1843  3086

               Accuracy : 0.7279
                 95% CI : (0.7219, 0.7338)
    No Information Rate : 0.6703
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.332

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

            Sensitivity : 0.4334
            Specificity : 0.8727
         Pos Pred Value : 0.6261
         Neg Pred Value : 0.7580
             Prevalence : 0.3297
         Detection Rate : 0.1429
   Detection Prevalence : 0.2282
      Balanced Accuracy : 0.6530

       'Positive' Class : 1
```

Results for the LightGBM model (test set) include an AUC of 0.75, accuracy of 73%, sensitivity (true positive for Class 1) of 43%, and specificity (true negative for Class 0) of 88%. The LightGBM model performs almost identically to the XGBoost model, with both models achieving the same AUC and similar accuracy and specificity. While the sensitivity of LightGBM is slightly lower than that of XGBoost, both models effectively balance the identification of defaulters and non-defaulters.

---

6. Comparison Metrics

Final Metrics

The metrics below are simply to summarize what has already been interpreted and provide a unique comparative visualization of the models' performance on both the training and test sets.

```
# Metrics summary with updated column names and descriptions
metrics_df <- data.frame(
  Model = c("Logistic Regression\n(Train)", "Logistic Regression\n(Test)",
            "Random Forest\n(Train)", "Random Forest\n(Test)",
            "XGBoost\n(Train)", "XGBoost\n(Test)",
```

```
            "LightGBM\n(Train)", "LightGBM\n(Test)"),
  AUC = c(auc_train_logistic, auc_test_logistic,
          auc_train_rf, auc_test_rf,
          auc_train_xgb, auc_test_xgb,
          auc_train_lgb, auc_test_lgb),
  Accuracy = c(conf_matrix_train_logistic$overall["Accuracy"], conf_matrix_test_logistic$overall["
               conf_matrix_train_rf$overall["Accuracy"], conf_matrix_test_rf$overall["Accuracy"],
               conf_matrix_train_xgb$overall["Accuracy"], conf_matrix_test_xgb$overall["Accuracy"]
               conf_matrix_train_lgb$overall["Accuracy"], conf_matrix_test_lgb$overall["Accuracy"]
  Sensitivity_True_Positive_Rate_for_Class_1 = c(conf_matrix_train_logistic$byClass["Sensitivity"]
                                                 conf_matrix_test_logistic$byClass["Sensitivity"],
                                                 conf_matrix_train_rf$byClass["Sensitivity"],
                                                 conf_matrix_test_rf$byClass["Sensitivity"],
                                                 conf_matrix_train_xgb$byClass["Sensitivity"],
                                                 conf_matrix_test_xgb$byClass["Sensitivity"],
                                                 conf_matrix_train_lgb$byClass["Sensitivity"],
                                                 conf_matrix_test_lgb$byClass["Sensitivity"]),
  Specificity_True_Negative_Rate_for_Class_0 = c(conf_matrix_train_logistic$byClass["Specificity"]
                                                 conf_matrix_test_logistic$byClass["Specificity"],
                                                 conf_matrix_train_rf$byClass["Specificity"],
                                                 conf_matrix_test_rf$byClass["Specificity"],
                                                 conf_matrix_train_xgb$byClass["Specificity"],
                                                 conf_matrix_test_xgb$byClass["Specificity"],
                                                 conf_matrix_train_lgb$byClass["Specificity"],
                                                 conf_matrix_test_lgb$byClass["Specificity"]))

# Display the table with kable, enabling LaTeX to interpret the line breaks
kable(metrics_df,
      caption = "Model Performance Metrics",
      digits = 2,
      col.names = c("Model", "AUC", "Accuracy",
                    "Sensitivity\nTPR_1", "Specificity\nTNR_0"),
      escape = FALSE)
```

Model Performance Metrics

Model	AUC	Accuracy	Sensitivity TPR_1	Specificity TNR_0
Logistic Regression (Train)	0.74	0.72	0.38	0.89
Logistic Regression (Test)	0.74	0.72	0.37	0.89
Random Forest (Train)	0.84	0.76	0.40	0.94
Random Forest (Test)	0.74	0.72	0.32	0.92
XGBoost (Train)	0.80	0.75	0.46	0.90
XGBoost (Test)	0.75	0.73	0.41	0.88
LightGBM (Train)	0.87	0.80	0.56	0.92
LightGBM (Test)	0.75	0.73	0.43	0.87

The AUC represents the overall performance of the model in distinguishing between classes, measuring its ability to rank positive instances higher than negative ones across all possible thresholds. Accuracy indicates the proportion of correctly predicted instances among all instances.

Sensitivity (TPR_1) represents the percentage of correct predictions among all actual class 1 instances, which refers to individuals who defaulted on their loans (TARGET=1). Specificity (TNR_0) represents the percentage of correct predictions among all actual class 0 instances, which refers to individuals who paid their loans on time (TARGET=0).

6.1 Bussiness relevance of each model

Logistic Regression
It was straightforward but performed poorly in predicting defaulters, capturing only 37% of them, which was insufficient for credit prediction.

Random Forest
It was effective at identifying non-defaulters (92% specificity), reducing the risk of lending to them. However, it missed 32% of defaulters, leading to more false negatives. Key predictors, such as EXT_SOURCE_1, EXT_SOURCE_2, and EXT_SOURCE_3, were significant, but its inability to handle complex relationships limited its effectiveness compared to other models.

XGBoost
It offered the best performance with an AUC of 0.75. It excelled at capturing complex relationships, particularly between EXT_SOURCE_3 and EXT_SOURCE_2, which were crucial for predicting defaults. XGBoost effectively balanced sensitivity and specificity, improving the accuracy of credit decisions.

LightGBM
It showed similar results to XGBoost, with key predictors like EXT_SOURCE_2 and EXT_SOURCE_3 playing a significant role in the decision process.

6.2 Ensemble Model for Better Credit Allocation

Given this scenario, we propose adopting a combined model (Ensemble) that integrates predictions from all four models. Averaging the probabilities from each model has shown slightly better performance compared to individual models. The ensemble will balance the strengths of each technique—for example, the interpretability of logistic regression, the ability of Random Forest to capture non-linear patterns, and the accuracy of XGBoost and LightGBM. This approach provides a more robust and reliable solution for credit allocation, crucial for achieving financial inclusion without compromising Home Credit's financial stability.

---

7. Predicting the Kaggle Test Set

7.1 Application Test Transformations

The code chunks below aim to apply the same feature engineering to the target prediction file (application_test) as was performed on the application_train file. This ensures that the models created can make accurate predictions.

```r
# Create the clean data set
app_test_clean <- application_test


# Clean and transform
app_test_clean <- app_test_clean %>%
  filter(AMT_INCOME_TOTAL <= 5000000)
```

```r
# Transformation for better understanding ages
app_test_clean$DAYS_BIRTH <- app_test_clean$DAYS_BIRTH / -365
```

```r
# Create a new column
app_test_clean <- application_test %>%
  mutate(DAYS_EMPLOYED_ANOM = ifelse(DAYS_EMPLOYED == 365243, TRUE, FALSE))

# Replace the anomalies for NA
app_test_clean$DAYS_EMPLOYED[app_test_clean$DAYS_EMPLOYED == 365243] <- NA

# Convert DAYS_EMPLOYED to years
app_test_clean <- app_test_clean %>%
  mutate(DAYS_EMPLOYED_YEARS = DAYS_EMPLOYED / -365)

# Remove DAYS_EMPLOYED
app_test_clean <- app_test_clean %>%
  select(-DAYS_EMPLOYED)
```

```r
# Loop through each column of the dataset
for (col in names(app_test_clean)) {
  # Check if the column is of character type
  if (is.character(app_test_clean[[col]])) {
    # Replace empty strings with "Unknown"
    app_test_clean[[col]] <- as.character(app_test_clean[[col]])  # Ensure column is character
    app_test_clean[[col]][app_test_clean[[col]] == ""] <- "Unknown"  # Replace empty strings
    app_test_clean[[col]] <- as.factor(app_test_clean[[col]]) }}  # Convert back to factor if neec
```

```r
# Convert OCCUPATION_TYPE to factor if not already done
app_test_clean$OCCUPATION_TYPE <- as.factor(app_test_clean$OCCUPATION_TYPE)

# Replace empty strings with "Unknown"
app_test_clean$OCCUPATION_TYPE <- as.character(app_test_clean$OCCUPATION_TYPE)  # Convert to chara
app_test_clean$OCCUPATION_TYPE[app_test_clean$OCCUPATION_TYPE == ""] <- "Unknown"  # Replace empty
app_test_clean$OCCUPATION_TYPE <- as.factor(app_test_clean$OCCUPATION_TYPE)  # Convert back to fac
```

```r
#Function to change index to column
index_to_col <- function(data, Column_Name){
          data <- cbind(newColName = rownames(data), data)
```

```
          rownames(data) <- 1:nrow(data)
          colnames(data)[1] <- Column_Name
          return (data)
        }

# Recreate the list that includes numeric and integer columns
numeric_integer_list <- unlist(lapply(app_test_clean, function(x) is.numeric(x) || is.integer(x)))

# Create a new data frame with the numeric and integer columns
data_num <- setDT(app_test_clean)[, ..numeric_integer_list]  # Select only numeric and integer col

# Combine one-hot encoded data with numeric and integer data
test_pre <- cbind(data_non_num_dum, data_num)
```

```
# Calculate the percentage of missing values for each column
mv <- as.data.frame(apply(test_pre, 2, function(col) sum(is.na(col)) / length(col)))
colnames(mv)[1] <- "missing_values"  # Rename the first column to "missing_values"

# Add a column with the index as the first column
mv <- index_to_col(mv, 'Column')

# Order the missing values in descending order
mv <- setDT(mv)[order(missing_values, decreasing = TRUE)]

# Fill in the missing values using aggregation
test_pre <- na.aggregate(test_pre)

# Replace dots with underscores
colnames(test_pre) <- gsub("\\.", "_", colnames(test_pre))

# Replace spaces, special characters, or dashes in column names with underscores
#colnames(test_pre) <- gsub("[[:punct:]\\s-]+", "_", colnames(test_pre))
```

```
# Get the names of all columns in data_pre_sample except for TARGET
cols_to_keep_test <- colnames(data_pre_sample)[colnames(data_pre_sample) != "TARGET"]

# Create a new data frame that only includes the selected columns
test_pre <- as.data.frame(test_pre)[, (colnames(test_pre) %in% cols_to_keep_test)]

# Using the same test set name
dt_test_kaggle <-test_pre
```

7.2 Predicting Application Test

Below, we will make probability predictions for each model for submission.

Logistic Regression

```
# Predict probabilities on the test dataset using Logistic Regression
pred_logistic <- predict(logistic_model, newdata = dt_test_kaggle, type = "prob")
```

Random Forest

```
# Predict probabilities on the test dataset using Random Forest
pred_random_forest <- predict(random_forest_model, newdata = dt_test_kaggle, type = "prob")
```

XGBoost Model

```
# Convert to matrix format (excluding any columns that are not features)
dt_test_kaggle_matrix <- as.matrix(dt_test_kaggle)

# Predict class probabilities for the test dataset using XGBoost
pred_xgboost <- predict(xgboost_model, newdata = dt_test_kaggle_matrix, type = "prob")

# Extract probabilities
pred_xgboost_class1 <- pred_xgboost  # Probabilities for Class 1
pred_xgboost_class0 <- 1 - pred_xgboost_class1  # Probabilities for Class 0 (1 - prob_class1)
```

LightGBM Model

```
# Predict class probabilities for the test dataset using LightGBM
pred_lgb <- predict(lightgbm_model, newdata = dt_test_kaggle_matrix)

# Extract probabilities for Class 1
pred_lgb_class1 <- pred_lgb  # Probabilities for Class 1
pred_lgb_class0 <- 1 - pred_lgb_class1  # Probabilities for Class 0 (1 - prob_class1)
```

---

8. Submission and Kaggle Score

Average of the models

This submission combines the predictions using the ensemble method, calculating the average of all predictions for submission to Kaggle leveraging the strengths of each individual model to enhance overall performance.

```
# Initialize the data frame with the SK_ID_CURR column from application_test
results_kaggle <- data.frame(SK_ID_CURR = application_test$SK_ID_CURR)

# Logistic Regression returns probabilities in a data frame with columns Class0 and Class1
results_kaggle$TARGET_Logistic <- pred_logistic$Class1  # Use Class1 probabilities for TARGET

# Random Forest returns probabilities in a data frame with columns Class0 and Class1
results_kaggle$TARGET_RF <- pred_random_forest$Class1  # Add Class1 probabilities for RF

# XGBoost returns a numeric vector of probabilities for Class1
```

```
results_kaggle$TARGET_XGBoost <- pred_xgboost  # Directly use probabilities for Class1

# LightGBM returns a named numeric vector, where "1" corresponds to Class1 probabilities
results_kaggle$TARGET_LGB <- pred_lgb  # Use Class1 probabilities for LGB (probabilities for class

# Calculate the average probabilities for "Class1"
results_kaggle$TARGET_Average <- rowMeans(results_kaggle[, c("TARGET_Logistic", "TARGET_RF", "TARG

# Create the final submission data frame with only SK_ID_CURR and average TARGET
submission_average <- results_kaggle[, c("SK_ID_CURR", "TARGET_Average")]
colnames(submission_average) <- c("SK_ID_CURR", "TARGET")  # Rename columns to match submission fo

# Save the final submission data frame as CSV
write.csv(submission_average, file = "D:/mymodels/submission_average.csv", row.names = FALSE, quot
```

Kaggle Score

The score achieved by averaging the results of all the models was 0.739 only 8% lower than the competition winner. The individual scores for each model were as follows:

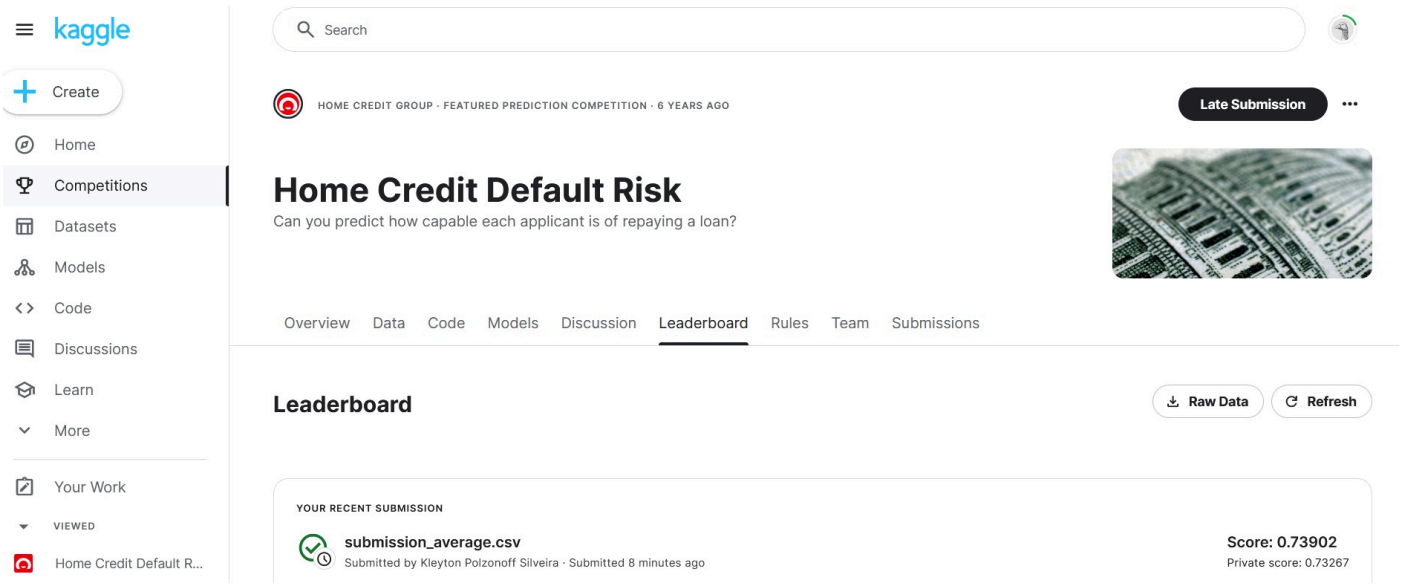
Logistic Regression: 0.730

Random Forest: 0.729

XGBoost: 0.733

LightGBM: 0.733

```
# Kaggle Score Image
knitr::include_graphics("D:/Analytics Projects/Home_Credit_Project/images/Kaggle Kleyton final.jpg
```



Conclusion

Predicting defaults is inherently challenging, as all models rely heavily on external credit scoring data. However, integrating additional variables such as AMT_CREDIT, AMT_GOODS_PRICE, DAYS_EMPLOYED_YEARS, DAYS_BIRTH, DAYS_ID_PUBLISH, OWN_CAR_AGE, DAYS_LAST_PHONE_CHANGE, and NAME_EDUCATION_TYPE_higher_education can significantly enhance model performance. These insights enable the company to focus on the most relevant factors, reducing the need for unnecessary customer inputs.

By adopting an ensemble model that combines predictions from all four approaches, Home Credit can leverage the strengths of each technique to create a more balanced and accurate system. This approach ensures better credit allocation decisions, reduces the risk of defaults, and fosters financial inclusion while safeguarding the company’s financial stability.