PROJECT 2: SUPERVISED LEARNING: FRAUD DETECTION

Statistical Learning. Bachelor in Data Science and Engineering

Ángela María Durán Pinto

05/11/2023

1 Fraud Detection in Financial Transactions

2 Introduction:

In the ever-evolving landscape of financial transactions, the facilitation of secure and trustworthy electronic payments is imperative for the smooth functioning of the global economy. However, the increasing sophistication of fraudulent activities poses a considerable threat to the integrity of these transactions, necessitating robust and adaptive fraud detection mechanisms. Beyond financial losses, fraudulent transactions erode trust in digital platforms and financial institutions, underscoring the critical importance of effective fraud detection systems.

In this project, we embark on the exploration and implementation of machine learning techniques to address the multifaceted challenge of fraud detection. We integrate both classification and regression tasks, seeking not only to identify fraudulent transactions but also to predict transaction amounts, a nuanced endeavor that enhances our ability to discern anomalies and patterns indicative of potential fraud.

2.1 Classification Task: A Binary Lens on Fraud Detection

In the classification task, our focus is on categorizing transactions into binary outcomes: legitimate or fraudulent. The pivotal “Is Fraud?” variable acts as our guide, serving as a binary label that allows our models to distinguish between transactions with a potential risk of fraud and those that align with typical user behavior.

Why is this Useful?

- Risk Mitigation: The classification task is instrumental in mitigating risks associated with fraudulent transactions by providing a binary assessment of their legitimacy.
- Timely Intervention: Identifying fraudulent activities promptly enables swift intervention, minimizing financial losses and fortifying trust in financial systems.
- Resource Optimization: Accurate classification allows for targeted allocation of resources to investigate and address potential fraud, avoiding unnecessary scrutiny of legitimate transactions.

3 Data Preprocessing

3.1 Analyse the datasets

- User_id: Unique identifier for each user.
- Card_id: Unique identifier for each credit card.
- Year, Month, Day, Time: Components of the transaction timestamp, providing information about when the transaction occurred.
- Amount: The monetary value of the transaction.
- Use_Chip: Indicator of whether the transaction utilized a chip for security.
- Merchant_Name: Name of the merchant involved in the transaction.
- Merchant_City, Merchant_State: Location details of the merchant.
- MCC (Merchant Category Code): A standardized code representing the type of business the merchant is engaged in.
- Person: Identifier for an individual associated with the card.
- Current_Age, Retirement_Age, Birth_Year, Birth_Month, Gender: Demographic details of the cardholder.
- Address, City, State, Zipcode: Cardholder’s address information.
- Latitude, Longitude: Geographic coordinates of the cardholder’s location.
- Per_Capita_Income_Zipcode: Average income per person in the cardholder’s zipcode.
- Yearly_Income_Person: Annual income of the cardholder.
- Total_Debt: Cumulative debt of the cardholder.
- FICO_Score: Credit score indicating the cardholder’s creditworthiness.
- Num_Credit_Cards: Number of credit cards associated with the cardholder.
- Card_Type: Type or brand of the credit card.
- Credit_Limit: Maximum amount that can be charged on the credit card.

- Card_on_Dark_Web: Indicator of whether the credit card information is found on the dark web.
- Fraud: Binary variable indicating whether the transaction is fraudulent (1) or not (0).
- Date: Full date (combining Year, Month, Day) of the transaction.

These variables collectively provide comprehensive information about the credit card transactions, the cardholders, and associated demographic and geographic details, enabling a thorough analysis for fraud detection and risk assessment.

First, it is important to load all the libraries that will be needed along the project

```
#setwd("C:/1Angela/universidad/3/1cuatrimestre/aprendizaje_estadistico/project2")
rm(list=ls())
# Load required libraries
library(dplyr)
library(tidyr)
library(data.table)

# Dealing with missing values
library(mice)

#GGally: collection of functions for creating plots and visualizations to explore and understa
nd relationships between variables in your data. : heatmap-like graphs, and ggcorr
library(GGally)

# Visualisation and Imputation of Missing Data
library(VIM)

library(ggplot2)

library(caret)


library(randomForest)
library(gbm)
library(neuralnet)

library(pROC)
library(rpart)
library(rpart.plot)

library(xgboost)
library(knitr)

library(lubridate)

library(leaps)
#library(olsrr)

library(leaflet)
library(kknn)

library(glmnet)
library(elasticnet)
```

3.2 Feature engineering

For this project three datasets are going to be used. “Users” which contain variables related with the person that has made the transaction. “Cards” with variables regarding the user card. Finally, “transaction”, the main dataset with features about the transaction itself.

First, the individual datasets are loaded.

```
transactions <- read.csv("credit_card_transactions-ibm_v2.csv")

cards<- read.csv("sd254_cards.csv")
users<- read.csv("sd254_users.csv")
```

If we look at the dimensions we see a large dataset for the transactions

```
dim(transactions)
```

```
## [1] 24386900       15
```

```
dim(cards)
```

```
## [1] 6146   13
```

```
dim(users)
```

```
## [1] 2000   18
```

Now, we want to merge the three datasets in order to obtain a dataset with all the information altogether. For that, we need to a a column to User, with the user_id so that the merge can be done.

```
# Add a new column "User" to the users dataset with a range of values starting from 0
users$User <- seq(from = 0, to = nrow(users) - 1)

# Check the modified users dataset
head(users)
```

```
##           Person Current.Age Retirement.Age Birth.Year Birth.Month Gender
## 1 Hazel Robinson          53             66       1966          11 Female
## 2     Sasha Sadr          53             68       1966          12 Female
## 3     Saanvi Lee          81             67       1938          11 Female
## 4  Everlee Clark          63             63       1957           1 Female
## 5  Kyle Peterson          43             70       1976           9   Male
## 6    Aldo Walker          42             70       1977          10   Male
##                    Address Apartment          City State Zipcode Latitude
## 1            462 Rose Lane        NA      La Verne    CA   91750    34.15
## 2   3606 Federal Boulevard        NA   Little Neck    NY   11363    40.76
```

```
## 3          766 Third Drive        NA   West Covina    CA   91792    34.02
## 4         3 Madison Street        NA      New York    NY   10069    40.71
## 5 9620 Valley Stream Drive        NA San Francisco    CA   94117    37.76
## 6            58 Birch Lane         6     Davenport    IA   52803    41.55
##   Longitude Per.Capita.Income...Zipcode Yearly.Income...Person Total.Debt
## 1   -117.76                      $29278                 $59696    $127613
## 2    -73.74                      $37891                 $77254    $191349
## 3   -117.89                      $22681                 $33483       $196
## 4    -73.99                     $163145                $249925    $202328
## 5   -122.44                      $53797                $109687    $183855
## 6    -90.60                      $20599                 $41997         $0
##   FICO.Score Num.Credit.Cards User
## 1        787                5    0
## 2        701                5    1
## 3        698                5    2
## 4        722                4    3
## 5        675                1    4
## 6        704                3    5
```

```r
# Step 1: Join transactions with users
merged_data_users <- transactions %>%
  left_join(users, by = "User")
```

```r
# Step2: Add relevant columns from cards dataset (excluding CARD.INDEX)

# Convert data.frames to data.tables
dt_merged_data_users <- as.data.table(merged_data_users)
dt_cards <- as.data.table(cards)

# Remove duplicates in CARD.INDEX in dt_cards
dt_cards_unique <- unique(dt_cards, by = "CARD.INDEX")

# Set the key columns for joining
setkey(dt_merged_data_users, "Card")
setkey(dt_cards_unique, "CARD.INDEX")

# Perform left join using data.table's syntax
merged_data <- dt_merged_data_users[dt_cards_unique[, list(CARD.INDEX, Card.Type, Credit.Limit
, Card.on.Dark.Web)],
                                    on = .(Card = CARD.INDEX)]

# Check the merged dataset
head(merged_data)
```

```
##    User Card Year Month Day  Time  Amount          Use.Chip Merchant.Name
## 1:    0    0 2002     9   1 06:21 $134.09 Swipe Transaction  3.527213e+18
## 2:    0    0 2002     9   1 06:42  $38.48 Swipe Transaction -7.276121e+17
## 3:    0    0 2002     9   2 06:22 $120.34 Swipe Transaction -7.276121e+17
## 4:    0    0 2002     9   2 17:45 $128.95 Swipe Transaction  3.414527e+18
## 5:    0    0 2002     9   3 06:23 $104.71 Swipe Transaction  5.817218e+18
## 6:    0    0 2002     9   3 13:53  $86.19 Swipe Transaction -7.146671e+18
```

```
##    Merchant.City Merchant.State   Zip  MCC Errors. Is.Fraud.         Person
## 1:      La Verne             CA 91750 5300               No Hazel Robinson
## 2: Monterey Park             CA 91754 5411               No Hazel Robinson
## 3: Monterey Park             CA 91754 5411               No Hazel Robinson
## 4: Monterey Park             CA 91754 5651               No Hazel Robinson
## 5:      La Verne             CA 91750 5912               No Hazel Robinson
## 6: Monterey Park             CA 91755 5970               No Hazel Robinson
##    Current.Age Retirement.Age Birth.Year Birth.Month Gender       Address
## 1:          53             66       1966          11 Female 462 Rose Lane
## 2:          53             66       1966          11 Female 462 Rose Lane
## 3:          53             66       1966          11 Female 462 Rose Lane
## 4:          53             66       1966          11 Female 462 Rose Lane
## 5:          53             66       1966          11 Female 462 Rose Lane
## 6:          53             66       1966          11 Female 462 Rose Lane
##    Apartment     City State Zipcode Latitude Longitude
## 1:        NA La Verne    CA   91750    34.15   -117.76
## 2:        NA La Verne    CA   91750    34.15   -117.76
## 3:        NA La Verne    CA   91750    34.15   -117.76
## 4:        NA La Verne    CA   91750    34.15   -117.76
## 5:        NA La Verne    CA   91750    34.15   -117.76
## 6:        NA La Verne    CA   91750    34.15   -117.76
##    Per.Capita.Income...Zipcode Yearly.Income...Person Total.Debt FICO.Score
## 1:                     $29278                $59696    $127613        787
## 2:                     $29278                $59696    $127613        787
## 3:                     $29278                $59696    $127613        787
## 4:                     $29278                $59696    $127613        787
## 5:                     $29278                $59696    $127613        787
## 6:                     $29278                $59696    $127613        787
##    Num.Credit.Cards Card.Type Credit.Limit Card.on.Dark.Web
## 1:                5     Debit       $24295               No
## 2:                5     Debit       $24295               No
## 3:                5     Debit       $24295               No
## 4:                5     Debit       $24295               No
## 5:                5     Debit       $24295               No
## 6:                5     Debit       $24295               No
```

```
# Save merged_data in the working directory
fwrite(merged_data, "merged_data.csv")
```

3.3 Duplicated observations

First, we search for exactly duplicated rows. We obtain 66. This is not very usual as at least one variable can change.

```
sum(duplicated(merged_data))
```

```
## [1] 66
```

```
duplicated_rows <- which(duplicated(merged_data))
cat("Number of duplicated rows:", duplicated_rows, "\n")
```

```
## Number of duplicated rows: 88376 1520769 3405676 4480155 4577819 5123843 5796140 6986105 69
86775 7504781 7505433 7507634 7507840 7655094 7765721 8275683 8578984 8584061 8588185 8591041
8736459 8736568 9416865 9647058 11130913 11136948 11138215 11574084 11578900 12021988 12022059
 12022627 12023440 12023799 12024473 12043616 12882892 13144457 13841986 14373240 14433340 145
06655 15227901 15421485 15434303 15628546 15830877 15967299 17491344 18198412 19121459 2049864
6 20522720 20524861 20525121 20527542 20529124 20919719 21113642 21774928 21775951 21777909 21
778911 22969607 23208021 23459302
```

```
merged_data = merged_data[-duplicated_rows,]
```

Then we look for observations with duplicated values for some columns. We consider duplicated observations transactions with the same amount made by the same user the same year, month and day. Then, we obtain no more duplicates.

```
# Remove duplicates using dplyr package to keep only the unique rows based on some of the colu
mns
data_cleaned <- merged_data %>%
  distinct(Amount,User,Day,Month,Year, .keep_all = TRUE)

cat("Number of duplicated rows:", nrow(merged_data)-nrow(data_cleaned), "\n")
```

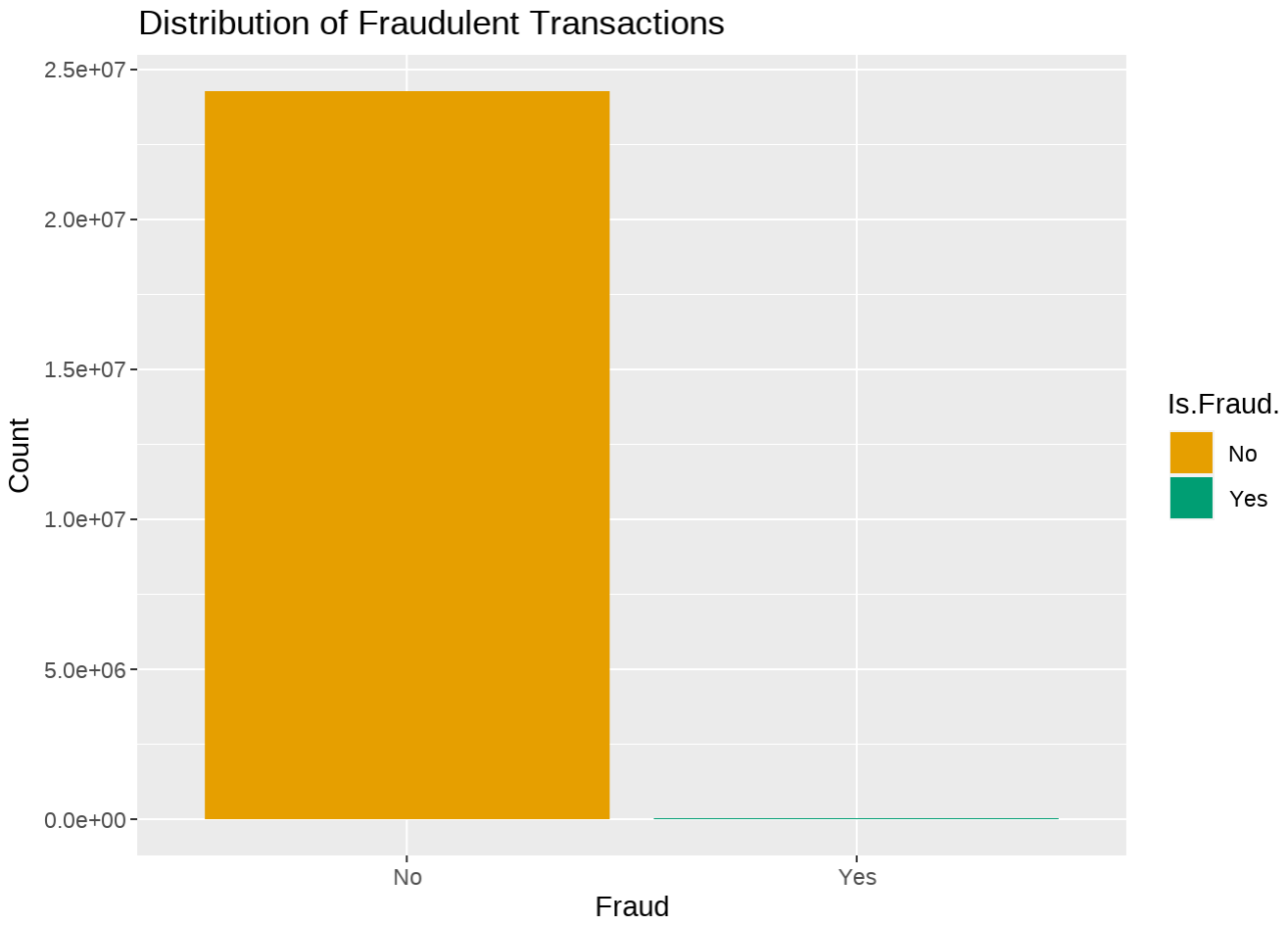
```
## Number of duplicated rows: 101907
```

3.4 Take a sample: Large dataset and Imbalanced Classes

Opting to work with a sample from the extensive dataset serves as a strategic decision with several practical advantages. Given the substantial size of the original dataset, using a sample enables more efficient processing, analysis, and model development. It strikes a balance between analytical accuracy and computational efficiency.

Imbalanced Classes

```
ggplot(data = data_cleaned, aes(x = Is.Fraud., fill = Is.Fraud.)) +
  geom_bar() +
  labs(title = "Distribution of Fraudulent Transactions", x = "Fraud", y = "Count")
```



Imbalanced class distribution, where one class significantly outnumbers the other, is a common challenge in classification tasks. In scenarios like fraud detection, the occurrence of rare events (e.g., fraudulent transactions) is far fewer than regular instances. This imbalance poses issues such as biased model training, misleading accuracy, and difficulty in detecting rare events due to a lack of sufficient examples for the minority class.

To address this challenge, a common strategy is to take a sample with proportionate classes. This involves creating a balanced dataset by randomly selecting instances from both classes, ensuring that the model is exposed to an equal representation of each. This approach enhances model performance, as it prevents the model from being skewed towards the majority class and allows it to discern patterns in the minority class.

The benefits of sampling with proportionate classes include improved model generalization, mitigation of bias, better evaluation metrics accuracy, and computational efficiency. Striking the right balance in the sampling proportion is crucial, as too much undersampling may lead to information loss. The chosen proportion depends on the specific characteristics of the problem and the desired trade-off between class balance and training set size. This strategy contributes to the development of more robust and equitable machine learning models, particularly in situations where class imbalances are pronounced.

Test Set

When balancing classes, it’s crucial to apply the technique to the training set only. The test set should remain representative of the real-world distribution to provide a reliable assessment of model generalization. Introducing class balancing to the test set could distort the evaluation metrics, leading to an inaccurate estimation of model performance.

Strategy

We are following a strategy to handle both the imbalanced class issue and the computational challenges posed by a large dataset. Here’s a step-by-step breakdown of the plan:

1. *Split Original Dataset into Training and Test Sets:* Divide your original dataset into a training set and a test set. This is typically done using a random split, e.g., 70% for training and 30% for testing.
2. *Balancing the Training Set:* Apply undersampling technique for balancing the classes in the training set.
3. *Take a Sample of the Balanced Training Set:* Since the dataset is large and for computational efficiency, take a random sample from the balanced training set. This subset will be used for actual model training.
4. *Sample the Test Set Proportionally:* Sample a subset from the original test set, ensuring that the class proportions remain similar to the final balanced training set. This ensures that the test set reflects the characteristics of the training set.

The **downSample** function calculates the frequency of each class in the target variable. Then it randomly selects a subset of instances from the majority class to achieve the desired downsampling rate. Create Balanced Dataset:

```
# Assuming 'original_data' is your original dataset
set.seed(123)  # Set seed for reproducibility

# Step 1: Split original dataset
splitIndex <- createDataPartition(data_cleaned$Is.Fraud., p = 0.7, list = FALSE)
train_set <- data_cleaned[splitIndex, ]
test_set <- data_cleaned[-splitIndex, ]


fwrite(train_set, "train_data_cleaned.csv")
fwrite(test_set, "test_data_cleaned.csv")
```

```
train_set <- read.csv("train_data_cleaned.csv")
test_set <- read.csv("test_data_cleaned.csv")


train_set$Is.Fraud. = factor(x = train_set$Is.Fraud.)
# Step 2: Balance the training set ( using undersampling)
set.seed(456)
balanced_train_set <- downSample(x = train_set[, -which(names(train_set) == "Is.Fraud.")],
                                 y = train_set$Is.Fraud.)


# Shuffle the rows to mix the classes
balanced_train_set <- balanced_train_set[sample(nrow(balanced_train_set)), ]

fwrite(balanced_train_set, "balanced_train_set_data_cleaned.csv")

set.seed(831)
# Step 3: Take a sample from the balanced training set
sampled_train_set <- balanced_train_set[sample(nrow(balanced_train_set), 0.25 * nrow(balanced_
train_set)), ]


# Step 4: Sample the test set proportionally
sampled_test_set <- test_set[sample(nrow(test_set), 0.3 * nrow(sampled_train_set)), ]

# Now 'sampled_train_set' and 'sampled_test_set' can be used for model training and testing
```

```
# Check the class distribution in the original, training, and test sets
table(sampled_train_set$Class)
```

```
##
##   No  Yes
## 5239 5174
```

```
table(sampled_test_set$Is.Fraud.)
```

```
##
##   No  Yes
## 3118    5
```

3.5 Fix or remove typos or errors

```
glimpse(sampled_train_set)
```

```
## Rows: 10,413
## Columns: 36
## $ User                      <int> 1338, 958, 1933, 924, 359, 212, 1271, 1785…
## $ Card                      <int> 5, 0, 1, 2, 1, 0, 1, 1, 0, 0, 4, 0, 5, 3, …
## $ Year                      <int> 2018, 2010, 2019, 2009, 2004, 2017, 2008, …
## $ Month                     <int> 10, 5, 4, 12, 3, 12, 12, 11, 9, 9, 12, 12,…
## $ Day                       <int> 12, 18, 22, 5, 8, 3, 25, 21, 7, 16, 5, 21,…
## $ Time                      <chr> "06:01", "16:15", "13:23", "15:26", "02:20…
## $ Amount                    <chr> "$28.74", "$139.88", "$104.15", "$34.08", …
## $ Use.Chip                  <chr> "Online Transaction", "Swipe Transaction",…
## $ Merchant.Name             <dbl> -2.088492e+18, -3.895049e+18, 1.913477e+18…
## $ Merchant.City             <chr> "ONLINE", "Scarborough", "Parkers Prairie"…
## $ Merchant.State            <chr> "", "ME", "MN", "SC", "", "Italy", "", "CA…
## $ Zip                       <int> NA, 4074, 56361, 29654, NA, NA, NA, 94606,…
## $ MCC                       <int> 4784, 4900, 5300, 5813, 4722, 5411, 5311, …
## $ Errors.                   <chr> "", "", "", "", "", "", "", "", "", "", ""…
## $ Person                    <chr> "Laurel Gao", "Aaden Campbell", "Wayne Hug…
## $ Current.Age               <int> 49, 33, 64, 78, 53, 38, 43, 84, 77, 73, 87…
## $ Retirement.Age            <int> 65, 69, 66, 66, 63, 66, 66, 64, 63, 68, 69…
## $ Birth.Year                <int> 1970, 1986, 1955, 1942, 1966, 1981, 1976, …
## $ Birth.Month               <int> 8, 3, 7, 2, 8, 10, 4, 1, 3, 3, 11, 6, 11, …
## $ Gender                    <chr> "Female", "Male", "Male", "Female", "Male"…
## $ Address                   <chr> "4480 Essex Drive", "858 Plum Avenue", "33…
## $ Apartment                 <int> NA, NA, NA, 8, 10, NA, NA, 8, NA, NA, NA, …
## $ City                      <chr> "Summerville", "Scarborough", "Miltona", "…
## $ State                     <chr> "SC", "ME", "MN", "TN", "IN", "AR", "RI", …
## $ Zipcode                   <int> 29485, 4074, 56354, 37931, 46710, 72956, 2…
## $ Latitude                  <dbl> 33.00, 43.59, 46.04, 35.97, 41.36, 35.44, …
## $ Longitude                 <dbl> -80.17, -70.33, -95.29, -83.94, -85.23, -9…
```

```
## $ Per.Capita.Income...Zipcode <chr> "$21769", "$29237", "$15325", "$24801", "$…
## $ Yearly.Income...Person      <chr> "$44383", "$59613", "$31244", "$44469", "$…
## $ Total.Debt                  <chr> "$42369", "$36199", "$51924", "$12777", "$…
## $ FICO.Score                  <int> 707, 763, 713, 733, 583, 746, 693, 718, 67…
## $ Num.Credit.Cards            <int> 7, 4, 4, 5, 3, 5, 5, 3, 3, 4, 6, 6, 6, 5, …
## $ Card.Type                   <chr> "Debit", "Debit", "Debit", "Debit", "Debit…
## $ Credit.Limit                <chr> "$620", "$24295", "$21968", "$46414", "$21…
## $ Card.on.Dark.Web            <chr> "No", "No", "No", "No", "No", "No", "No", …
## $ Class                       <fct> No, No, No, No, Yes, Yes, Yes, Yes, Yes, N…
```

First, it is important to set appropriate column names.

```
new_column_names_training <- c( "User_id", "Card_id", "Year", "Month", "Day", "Time", "Amount"
, "Use_Chip", "Merchant_Name", "Merchant_City", "Merchant_State", "Zip", "MCC",  "Errors",   "
Person" , "Current_Age", "Retirement_Age", "Birth_Year", "Birth_Month", "Gender", "Address", "
Apartment", "City", "State", "Zipcode", "Latitude", "Longitude", "Per_Capita_Income_Zipcode",
"Yearly_Income_Person",     "Total_Debt", "FICO_Score", "Num_Credit_Cards", "Card_Type","Credi
t_Limit", "Card_on_Dark_Web","Fraud")

new_column_names_test <- c( "User_id", "Card_id", "Year", "Month", "Day", "Time", "Amount", "U
se_Chip", "Merchant_Name", "Merchant_City", "Merchant_State", "Zip", "MCC",  "Errors", "Fraud"
,  "Person" , "Current_Age", "Retirement_Age", "Birth_Year", "Birth_Month", "Gender", "Address
", "Apartment", "City", "State", "Zipcode", "Latitude", "Longitude", "Per_Capita_Income_Zipcod
e", "Yearly_Income_Person",     "Total_Debt", "FICO_Score", "Num_Credit_Cards", "Card_Type","C
redit_Limit", "Card_on_Dark_Web")

colnames(sampled_train_set) <- new_column_names_training
colnames(sampled_test_set) <- new_column_names_test
```

3.5.1 Encoding Categorical Variables

It is essential to encode categorical variables as most machine learning algorithms require numerical data for analysis. Encoding ensures that the model can interpret and learn from categorical features.

Furthermore, assigning meaningful labels to levels makes it easier to understand the categories represented by each level.

Use.Chip column contains different transaction methods.

```
### Use_Chip
sampled_train_set$Use_Chip= factor(x = sampled_train_set$Use_Chip,
                                   levels = c("Chip Transaction", "Swipe Transaction", "Online
Transaction"),
                                   labels =  c("Chip", "Swipe", "Online")
                                   )

sampled_test_set$Use_Chip= factor(x = sampled_test_set$Use_Chip,
                                   levels = c("Chip Transaction", "Swipe Transaction", "Online
Transaction"),
                                   labels =  c("Chip", "Swipe", "Online")
                                   )


### Card_Type
```

```
sampled_train_set$Card_Type = factor(x = sampled_train_set$Card_Type,
                                     levels = c("Debit", "Credit"),
                                     labels = c("Debit", "Credit")
                                     )

sampled_test_set$Card_Type = factor(x = sampled_test_set$Card_Type,
                                     levels = c("Debit", "Credit"),
                                     labels = c("Debit", "Credit")
                                     )


### Card_on_Dark_Web
sampled_train_set$Card_on_Dark_Web = factor(x = sampled_train_set$Card_on_Dark_Web,
                                            levels = c("No", "Yes"),
                                            labels = c("No", "Yes")
                                            )

sampled_test_set$Card_on_Dark_Web = factor(x = sampled_test_set$Card_on_Dark_Web,
                                            levels = c("No", "Yes"),
                                            labels = c("No", "Yes")
                                            )


### Gender
sampled_train_set$Gender = factor(x = sampled_train_set$Gender,
                                  levels = c("Female", "Male"),
                                  labels = c("Female", "Male") )

sampled_test_set$Gender = factor(x = sampled_test_set$Gender,
                                  levels = c("Female", "Male"),
                                  labels = c("Female", "Male") )


### Fraud
sampled_train_set$Fraud = factor(x = sampled_train_set$Fraud)
sampled_test_set$Fraud = factor(x = sampled_test_set$Fraud)

### Merchant_City
sampled_train_set$Merchant_City = as.factor(sampled_train_set$Merchant_City)
sampled_test_set$Merchant_City = as.factor(sampled_test_set$Merchant_City)

### Merchant_State
sampled_train_set$Merchant_State = as.factor(sampled_train_set$Merchant_State)
sampled_test_set$Merchant_State = as.factor(sampled_test_set$Merchant_State)

### City
sampled_train_set$City = as.factor(sampled_train_set$City)
sampled_test_set$City = as.factor(sampled_test_set$City)

### State
sampled_train_set$State = as.factor(sampled_train_set$State)
sampled_test_set$State = as.factor(sampled_test_set$State)
```

3.5.2 Fix numerical variables

Remove the dollar sign ($) and convert the following columns to numeric types

```
# The gsub function is used for pattern matching and replacement in strings. \\$ is a regular
expression pattern that matches the dollar  # sign $. The double backslash is used to escape t
he special meaning of the dollar sign in regular expressions.
# An empty string "" is specified as the replacement

sampled_train_set$Amount <- as.numeric(gsub("\\$", "", sampled_train_set$Amount))
sampled_test_set$Amount <- as.numeric(gsub("\\$", "", sampled_test_set$Amount))

sampled_train_set$Credit_Limit <- as.numeric(gsub("\\$", "", sampled_train_set$Credit_Limit))
sampled_test_set$Credit_Limit <- as.numeric(gsub("\\$", "", sampled_test_set$Credit_Limit))

sampled_train_set$Yearly_Income_Person <- as.numeric(gsub("\\$", "", sampled_train_set$Yearly_
Income_Person))
sampled_test_set$Yearly_Income_Person <- as.numeric(gsub("\\$", "", sampled_test_set$Yearly_In
come_Person))

sampled_train_set$Per_Capita_Income_Zipcode <- as.numeric(gsub("\\$", "", sampled_train_set$Pe
r_Capita_Income_Zipcode))
sampled_test_set$Per_Capita_Income_Zipcode <- as.numeric(gsub("\\$", "", sampled_test_set$Per_
Capita_Income_Zipcode))

sampled_train_set$Total_Debt <- as.numeric(gsub("\\$", "", sampled_train_set$Total_Debt))
sampled_test_set$Total_Debt <- as.numeric(gsub("\\$", "", sampled_test_set$Total_Debt))
```

3.5.3 Change date format

```
# Convert Time column to POSIXct format
sampled_train_set$Time <- as.POSIXct(sampled_train_set$Time, format = "%H:%M", tz = "UTC")
sampled_test_set$Time <- as.POSIXct(sampled_test_set$Time, format = "%H:%M", tz = "UTC")

# Assuming that Year, Month, and Day are already numeric
sampled_train_set$Date <- as.Date(paste(sampled_train_set$Year, sampled_train_set$Month, sampl
ed_train_set$Day, sep = "-"))
sampled_test_set$Date <- as.Date(paste(sampled_test_set$Year, sampled_test_set$Month, sampled_
test_set$Day, sep = "-"))
```

```
glimpse(sampled_train_set)
```

```
## Rows: 10,413
## Columns: 37
## $ User_id                   <int> 1338, 958, 1933, 924, 359, 212, 1271, 1785, …
## $ Card_id                   <int> 5, 0, 1, 2, 1, 0, 1, 1, 0, 0, 4, 0, 5, 3, 3,…
## $ Year                      <int> 2018, 2010, 2019, 2009, 2004, 2017, 2008, 20…
## $ Month                     <int> 10, 5, 4, 12, 3, 12, 12, 11, 9, 9, 12, 12, 2…
## $ Day                       <int> 12, 18, 22, 5, 8, 3, 25, 21, 7, 16, 5, 21, 3…
## $ Time                      <dttm> 2023-12-24 06:01:00, 2023-12-24 16:15:00, 2…
## $ Amount                    <dbl> 28.74, 139.88, 104.15, 34.08, 351.00, 5.70, …
```

```
## $ Use_Chip                   <fct> Online, Swipe, Swipe, Swipe, Online, Chip, O…
## $ Merchant_Name              <dbl> -2.088492e+18, -3.895049e+18, 1.913477e+18, …
## $ Merchant_City              <fct> ONLINE, Scarborough, Parkers Prairie, Honea …
## $ Merchant_State             <fct> , ME, MN, SC, , Italy, , CA, Italy, OH, , FL…
## $ Zip                        <int> NA, 4074, 56361, 29654, NA, NA, NA, 94606, N…
## $ MCC                        <int> 4784, 4900, 5300, 5813, 4722, 5411, 5311, 35…
## $ Errors                     <chr> "", "", "", "", "", "", "", "", "", "", "", …
## $ Person                     <chr> "Laurel Gao", "Aaden Campbell", "Wayne Hughe…
## $ Current_Age                <int> 49, 33, 64, 78, 53, 38, 43, 84, 77, 73, 87, …
## $ Retirement_Age             <int> 65, 69, 66, 66, 63, 66, 66, 64, 63, 68, 69, …
## $ Birth_Year                 <int> 1970, 1986, 1955, 1942, 1966, 1981, 1976, 19…
## $ Birth_Month                <int> 8, 3, 7, 2, 8, 10, 4, 1, 3, 3, 11, 6, 11, 11…
## $ Gender                     <fct> Female, Male, Male, Female, Male, Female, Ma…
## $ Address                    <chr> "4480 Essex Drive", "858 Plum Avenue", "3315…
## $ Apartment                  <int> NA, NA, NA, 8, 10, NA, NA, 8, NA, NA, NA, 6,…
## $ City                       <fct> Summerville, Scarborough, Miltona, Knoxville…
## $ State                      <fct> SC, ME, MN, TN, IN, AR, RI, MA, NY, OH, CA, …
## $ Zipcode                    <int> 29485, 4074, 56354, 37931, 46710, 72956, 284…
## $ Latitude                   <dbl> 33.00, 43.59, 46.04, 35.97, 41.36, 35.44, 41…
## $ Longitude                  <dbl> -80.17, -70.33, -95.29, -83.94, -85.23, -94.…
## $ Per_Capita_Income_Zipcode <dbl> 21769, 29237, 15325, 24801, 19236, 17093, 24…
## $ Yearly_Income_Person       <dbl> 44383, 59613, 31244, 44469, 39222, 34854, 49…
## $ Total_Debt                 <dbl> 42369, 36199, 51924, 12777, 66559, 40722, 61…
## $ FICO_Score                 <int> 707, 763, 713, 733, 583, 746, 693, 718, 673,…
## $ Num_Credit_Cards           <int> 7, 4, 4, 5, 3, 5, 5, 3, 3, 4, 6, 6, 6, 5, 4,…
## $ Card_Type                  <fct> Debit, Debit, Debit, Debit, Debit, Debit, De…
## $ Credit_Limit               <dbl> 620, 24295, 21968, 46414, 21968, 24295, 2196…
## $ Card_on_Dark_Web           <fct> No, No, No, No, No, No, No, No, No, No, No, …
## $ Fraud                      <fct> No, No, No, No, Yes, Yes, Yes, Yes, Yes, No,…
## $ Date                       <date> 2018-10-12, 2010-05-18, 2019-04-22, 2009-12…
```

```
summary(sampled_train_set)
```

```
##     User_id        Card_id           Year          Month             Day
##  Min.   :   0   Min.   :0.000   Min.   :1991   Min.   : 1.000   Min.   : 1.00
##  1st Qu.: 549   1st Qu.:0.000   1st Qu.:2008   1st Qu.: 4.000   1st Qu.: 8.00
##  Median :1019   Median :1.000   Median :2012   Median : 7.000   Median :16.00
##  Mean   :1018   Mean   :1.492   Mean   :2012   Mean   : 6.627   Mean   :15.74
##  3rd Qu.:1495   3rd Qu.:2.000   3rd Qu.:2016   3rd Qu.:10.000   3rd Qu.:23.00
##  Max.   :1998   Max.   :8.000   Max.   :2020   Max.   :12.000   Max.   :31.00
##
##       Time                          Amount          Use_Chip
##  Min.   :2023-12-24 00:00:00.00   Min.   :-500.00   Chip  :2177
##  1st Qu.:2023-12-24 09:25:00.00   1st Qu.:  11.72   Swipe :4469
##  Median :2023-12-24 12:22:00.00   Median :  44.61   Online:3767
##  Mean   :2023-12-24 12:31:35.83   Mean   :  76.53
##  3rd Qu.:2023-12-24 15:36:00.00   3rd Qu.: 100.00
##  Max.   :2023-12-24 23:59:00.00   Max.   :4133.28
##
##  Merchant_Name               Merchant_City  Merchant_State      Zip
##  Min.   :-9.199e+18   ONLINE        :3768          :3768   Min.   : 1020
##  1st Qu.:-4.282e+18   Rome          : 811   Italy  : 810   1st Qu.:28602
```

```
##  Median :-2.452e+17   Algiers       : 128   CA     : 705   Median :46405
##  Mean   :-1.779e+17   Port au Prince:  69   TX     : 431   Mean   :51231
##  3rd Qu.: 3.781e+18   Strasburg     :  62   NY     : 385   3rd Qu.:77586
##  Max.   : 9.223e+18   Mexico City   :  53   FL     : 331   Max.   :99508
##                       (Other)       :5522   (Other):3983   NA's   :4951
##       MCC          Errors             Person           Current_Age
##  Min.   :1711   Length:10413       Length:10413       Min.   : 18.00
##  1st Qu.:5094   Class :character   Class :character   1st Qu.: 43.00
##  Median :5411   Mode  :character   Mode  :character   Median : 53.00
##  Mean   :5401                                         Mean   : 55.08
##  3rd Qu.:5812                                         3rd Qu.: 65.00
##  Max.   :9402                                         Max.   :101.00
##
##  Retirement_Age    Birth_Year    Birth_Month       Gender
##  Min.   :50.00   Min.   :1918   Min.   : 1.00   Female:5422
##  1st Qu.:65.00   1st Qu.:1955   1st Qu.: 3.00   Male  :4991
##  Median :66.00   Median :1967   Median : 7.00
##  Mean   :66.44   Mean   :1964   Mean   : 6.59
##  3rd Qu.:68.00   3rd Qu.:1976   3rd Qu.:10.00
##  Max.   :79.00   Max.   :2002   Max.   :12.00
##
##    Address            Apartment                City          State
##  Length:10413       Min.   :   1.0   Houston     : 118   CA     :1285
##  Class :character   1st Qu.:   5.0   Miami       : 110   TX     : 881
##  Mode  :character   Median :  10.0   Brooklyn    :  76   NY     : 722
##                     Mean   : 662.7   Los Angeles :  68   FL     : 699
##                     3rd Qu.:  95.0   Indianapolis:  67   NC     : 414
##                     Max.   :9940.0   Minneapolis :  62   IL     : 397
##                     NA's   :7691     (Other)     :9912   (Other):6015
##     Zipcode         Latitude       Longitude       Per_Capita_Income_Zipcode
##  Min.   : 1230   Min.   :21.30   Min.   :-159.41   Min.   :     0
##  1st Qu.:28379   1st Qu.:33.77   1st Qu.: -97.39   1st Qu.: 16901
##  Median :46725   Median :38.05   Median : -86.79   Median : 20917
##  Mean   :50897   Mean   :37.24   Mean   : -91.86   Mean   : 23401
##  3rd Qu.:77450   3rd Qu.:41.03   3rd Qu.: -80.16   3rd Qu.: 26478
##  Max.   :99508   Max.   :61.20   Max.   : -68.67   Max.   :163145
##
##  Yearly_Income_Person   Total_Debt       FICO_Score    Num_Credit_Cards
##  Min.   :     1       Min.   :     0   Min.   :488.0   Min.   :1.000
##  1st Qu.: 32150       1st Qu.: 16549   1st Qu.:684.0   1st Qu.:3.000
##  Median : 40189       Median : 49681   Median :716.0   Median :4.000
##  Mean   : 45275       Mean   : 56211   Mean   :714.8   Mean   :3.943
##  3rd Qu.: 52754       3rd Qu.: 82822   3rd Qu.:757.0   3rd Qu.:5.000
##  Max.   :280199       Max.   :461854   Max.   :850.0   Max.   :9.000
##
##   Card_Type     Credit_Limit   Card_on_Dark_Web Fraud           Date
##  Debit :8423   Min.   :   18   No :10413        No :5239   Min.   :1991-12-06
##  Credit:1249   1st Qu.:21968   Yes:    0        Yes:5174   1st Qu.:2008-10-02
##  NA's  : 741   Median :24295                               Median :2012-09-20
##                Mean   :23906                               Mean   :2012-05-22
##                3rd Qu.:24295                               3rd Qu.:2016-05-09
##                Max.   :46414                               Max.   :2020-02-28
##
```

```
fwrite(sampled_train_set, "sampled_train_set.csv")
fwrite(sampled_test_set, "sampled_test_set.csv")
```

3.6 Missing Values

Missing values, often denoted as NA (Not Available) or NaN (Not-a-Number), are placeholders used in data to represent the absence of a value or an unknown value for a particular observation or variable. They can have a significant impact on data analysis and modeling.

We can see the distribution of the NAs with the mice package

```
sum(is.na(sampled_train_set))
```

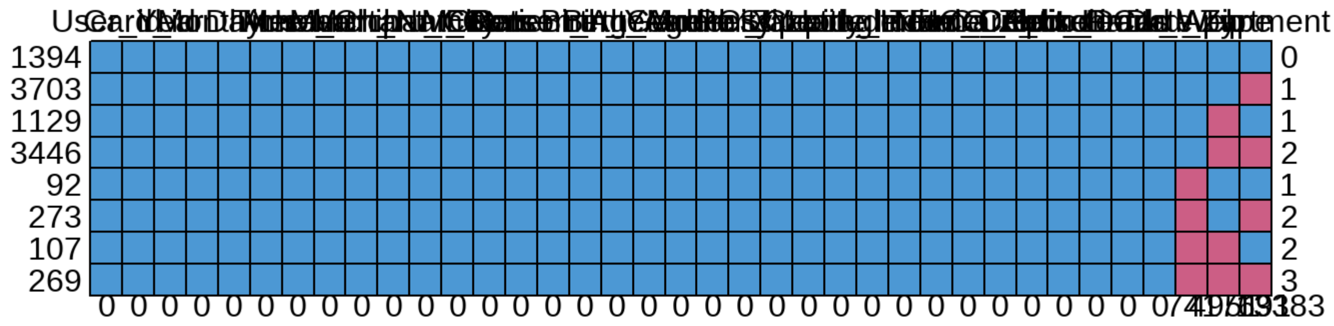
```
## [1] 13383
```

```
sum(is.na(sampled_test_set))
```

```
## [1] 2821
```

There are 13304 missing values for the training set and 2879 for the test set, which is a high amount.

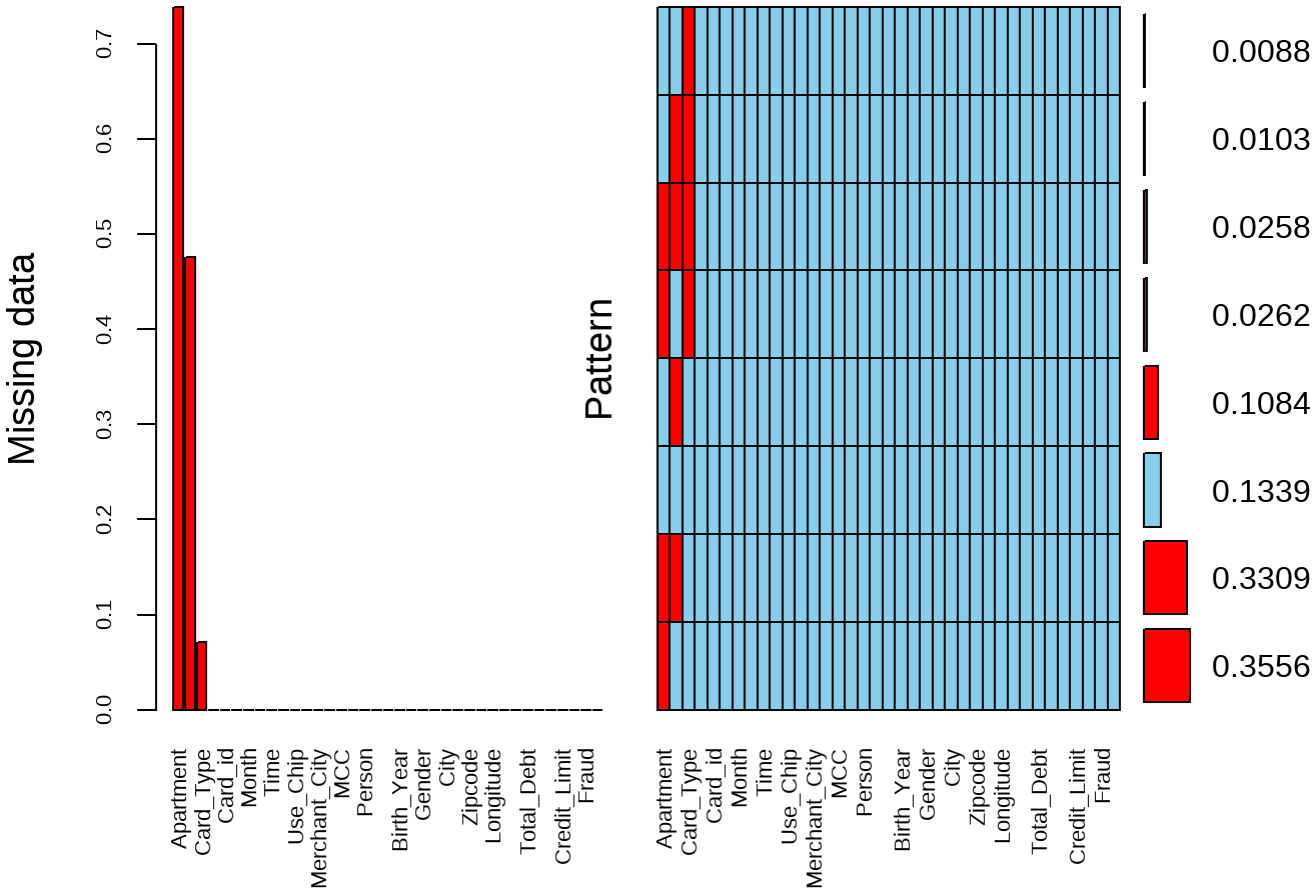
```
md.pattern(sampled_train_set)
```




```
##      User_id Card_id Year Month Day Time Amount Use_Chip Merchant_Name
## 1394       1       1    1     1   1    1      1        1             1
## 3703       1       1    1     1   1    1      1        1             1
## 1129       1       1    1     1   1    1      1        1             1
## 3446       1       1    1     1   1    1      1        1             1
## 92         1       1    1     1   1    1      1        1             1
## 273        1       1    1     1   1    1      1        1             1
## 107        1       1    1     1   1    1      1        1             1
## 269        1       1    1     1   1    1      1        1             1
##            0       0    0     0   0    0      0        0             0
##      Merchant_City Merchant_State MCC Errors Person Current_Age Retirement_Age
## 1394             1              1   1      1      1           1              1
## 3703             1              1   1      1      1           1              1
## 1129             1              1   1      1      1           1              1
## 3446             1              1   1      1      1           1              1
## 92               1              1   1      1      1           1              1
## 273              1              1   1      1      1           1              1
## 107              1              1   1      1      1           1              1
## 269              1              1   1      1      1           1              1
##                  0              0   0      0      0           0              0
##      Birth_Year Birth_Month Gender Address City State Zipcode Latitude
## 1394          1           1      1       1    1     1       1        1
## 3703          1           1      1       1    1     1       1        1
## 1129          1           1      1       1    1     1       1        1
## 3446          1           1      1       1    1     1       1        1
## 92            1           1      1       1    1     1       1        1
```

```
## 273           1           1      1       1    1     1       1         1
## 107           1           1      1       1    1     1       1         1
## 269           1           1      1       1    1     1       1         1
##               0           0      0       0    0     0       0         0
##      Longitude Per_Capita_Income_Zipcode Yearly_Income_Person Total_Debt
## 1394         1                         1                    1          1
## 3703         1                         1                    1          1
## 1129         1                         1                    1          1
## 3446         1                         1                    1          1
## 92           1                         1                    1          1
## 273          1                         1                    1          1
## 107          1                         1                    1          1
## 269          1                         1                    1          1
##              0                         0                    0          0
##      FICO_Score Num_Credit_Cards Credit_Limit Card_on_Dark_Web Fraud Date
## 1394          1                1            1                1     1    1
## 3703          1                1            1                1     1    1
## 1129          1                1            1                1     1    1
## 3446          1                1            1                1     1    1
## 92            1                1            1                1     1    1
## 273           1                1            1                1     1    1
## 107           1                1            1                1     1    1
## 269           1                1            1                1     1    1
##               0                0            0                0     0    0
##      Card_Type  Zip Apartment
## 1394         1    1         1     0
## 3703         1    1         0     1
## 1129         1    0         1     1
## 3446         1    0         0     2
## 92           0    1         1     1
## 273          0    1         0     2
## 107          0    0         1     2
## 269          0    0         0     3
##            741 4951      7691 13383
```

```
aggr(sampled_train_set, number = TRUE, sortVars = TRUE, labels = names(sampled_train_set),
     cex.axis = .7, gap = 1, ylab= c('Missing data','Pattern'))
```




```
##
##  Variables sorted by number of missings:
##                    Variable      Count
##                   Apartment 0.73859599
##                         Zip 0.47546336
##                   Card_Type 0.07116105
##                     User_id 0.00000000
##                     Card_id 0.00000000
##                        Year 0.00000000
##                       Month 0.00000000
##                         Day 0.00000000
##                        Time 0.00000000
##                      Amount 0.00000000
##                    Use_Chip 0.00000000
##               Merchant_Name 0.00000000
##               Merchant_City 0.00000000
##              Merchant_State 0.00000000
##                         MCC 0.00000000
##                      Errors 0.00000000
##                      Person 0.00000000
##                 Current_Age 0.00000000
##              Retirement_Age 0.00000000
##                  Birth_Year 0.00000000
##                 Birth_Month 0.00000000
##                      Gender 0.00000000
##                     Address 0.00000000
```

```
##                         City 0.00000000
##                        State 0.00000000
##                      Zipcode 0.00000000
##                     Latitude 0.00000000
##                    Longitude 0.00000000
##  Per_Capita_Income_Zipcode 0.00000000
##       Yearly_Income_Person 0.00000000
##                  Total_Debt 0.00000000
##                  FICO_Score 0.00000000
##            Num_Credit_Cards 0.00000000
##                Credit_Limit 0.00000000
##            Card_on_Dark_Web 0.00000000
##                       Fraud 0.00000000
##                        Date 0.00000000
```

It can be seen that the “Apartment” variable has more than 70% of its values as NAs, so it will be removed, as it also does not give to much information. We will also remove “Zip” variable for the same reason.

We also remove the “Errors” variable as almost all its values are empty strings, ““.

```
sampled_train_set =  sampled_train_set %>% dplyr::select(-Apartment, -Zip, -Errors)
sampled_test_set =  sampled_test_set %>% dplyr::select(-Apartment, -Zip, -Errors)
```

The “mice” package in R is a powerful tool which is an iterative imputation method that replaces missing values with multiple imputations using a regression model. The imputed values are then used to estimate the missing values in the subsequent iteration until the convergence criteria are met. We are going to use Random Forest as method.

imp_train <- mice(sampled_train_set, method = ‘cart’, m = 4, parallel = TRUE) sampled_train_set= complete(imp_train)

Because of my system doesn’t have sufficient memory (RAM) available to perform the imputation, I am going to remove the observations with missing values considering that I have a high number of observations. However, this is not the best approach.

Let’ see the nº of observations with missing values

```
sum(apply(sampled_train_set, 1, function(row) any(is.na(row))))
```

```
## [1] 741
```

```
sum(apply(sampled_test_set, 1, function(row) any(is.na(row))))
```

```
## [1] 180
```

```
# Remove observations with missing values from sampled_train_set
sampled_train_set <- na.omit(sampled_train_set)

# Remove observations with missing values from sampled_test_set
sampled_test_set <- na.omit(sampled_test_set)
```

```
sum(is.na(sampled_train_set))
```

```
## [1] 0
```

```
sum(is.na(sampled_test_set))
```

```
## [1] 0
```

Now, we don't have any missing values.

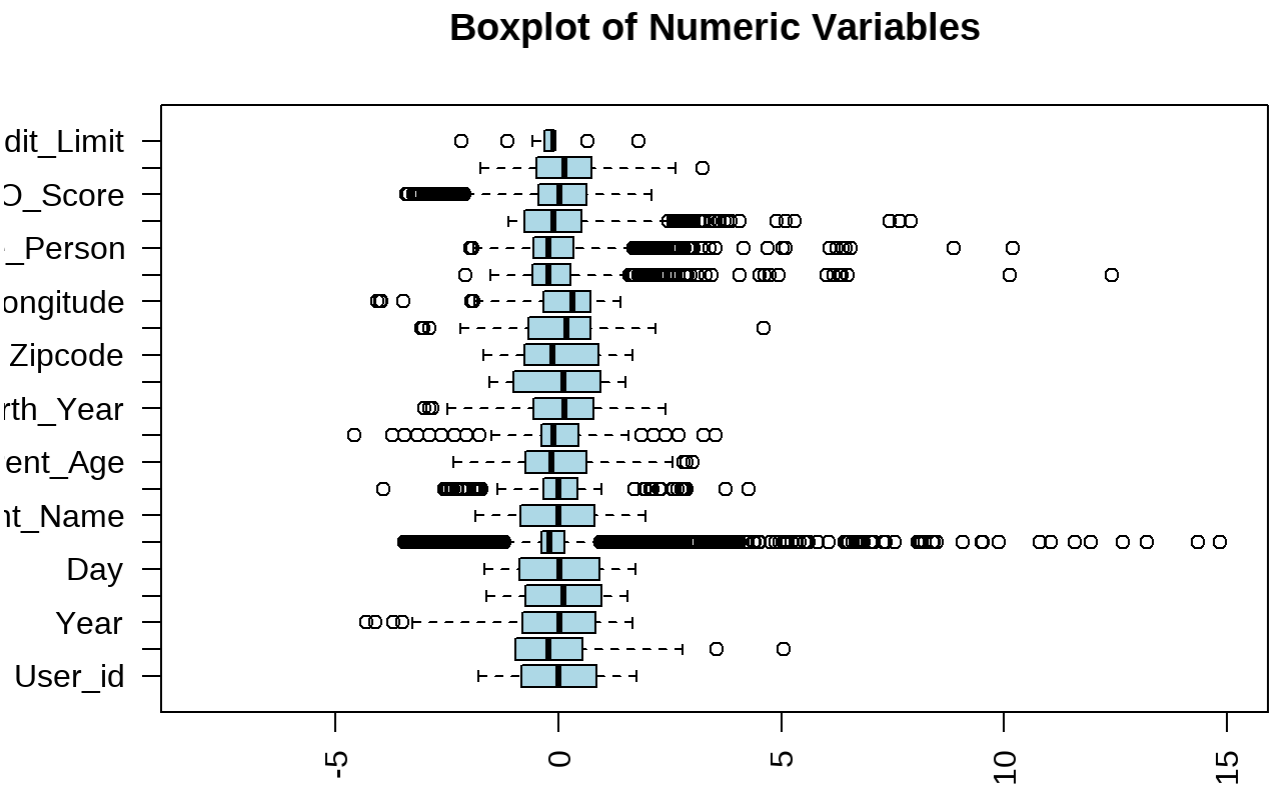
```
fwrite(sampled_train_set, "sampled_train_set_no_NAs.csv")
fwrite(sampled_test_set, "sampled_test_set_no_NAs.csv")
```

3.7 Outliers

An outlier is an observation or data point that significantly differs from other observations in a dataset. It is an unusual or rare value that falls outside the typical range of values in a dataset.

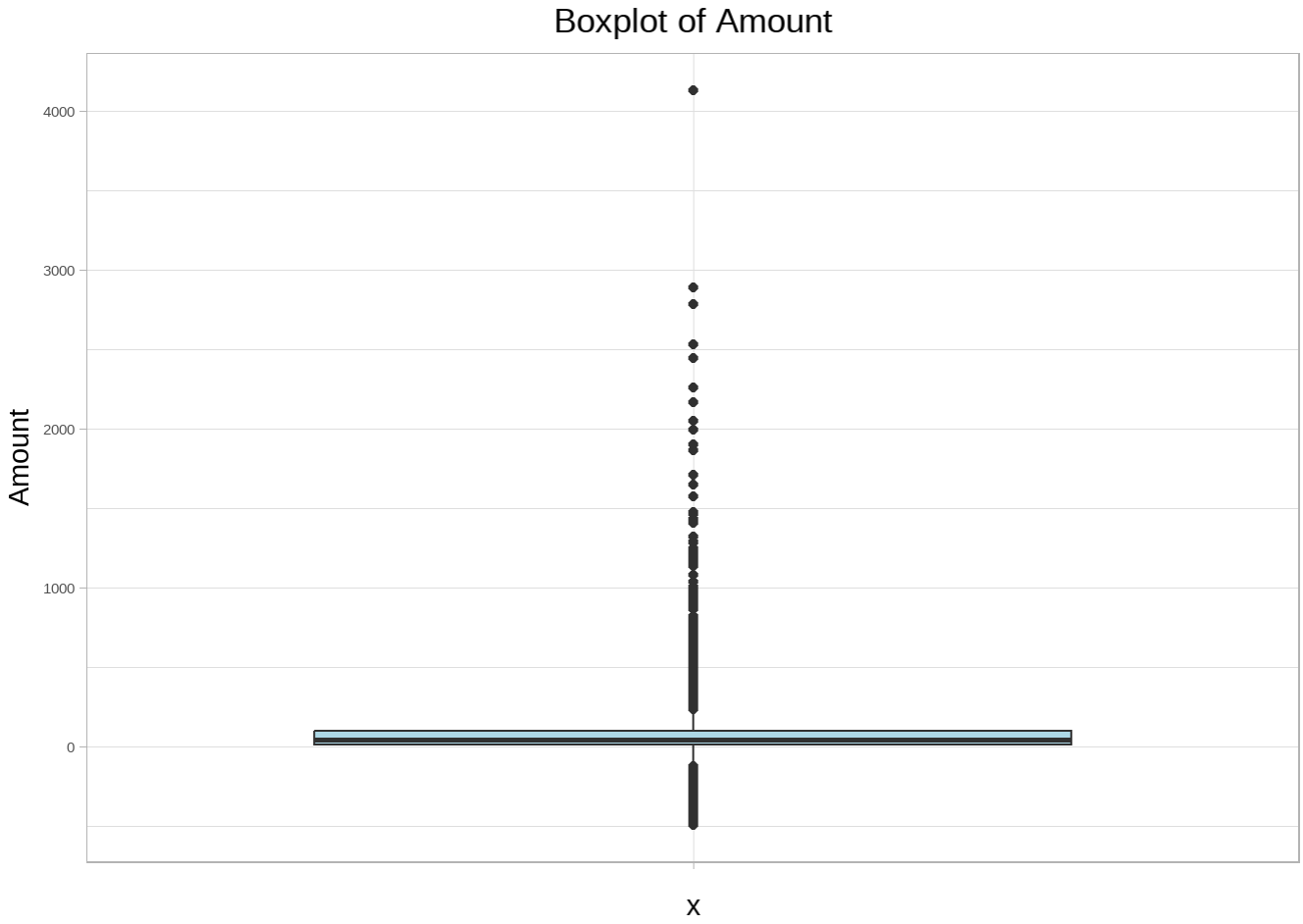
First, we make a general boxplot for the entire dataset.

```
# Select numeric columns only
numeric_data <- sampled_train_set[sapply(sampled_train_set, is.numeric)]

# Create horizontal boxplot for numeric columns
boxplot(scale(numeric_data), col = "lightblue", las = 2, horizontal = TRUE, ylim = c(-8, 15))
# Add a title to the plot
title(main = "Boxplot of Numeric Variables")
```



Upon inspecting the boxplot, it appears that potential outliers are present. Notably, variables associated with monetary values, such as “Amount,” exhibit a higher prevalence of outliers. Let’s zoom in on these specific variables and assess whether this observed behavior is plausible.

```
par(mfrow=c(1,3))
# For "Amount"
ggplot(sampled_train_set) +
  aes(x = "", y = Amount) +
  geom_boxplot(fill = "lightblue") +
  labs(y = "Amount") +
  ggtitle("Boxplot of Amount") +  # Add title
  theme_light() +
  theme(plot.title = element_text(hjust = 0.5))  # Center the title
```



In the context of financial transactions, it's not uncommon to observe a wide range of amounts, and extreme values can be valid, depending on the nature of the transactions. For instance, negative amounts could represent refunds or returns, and large positive amounts might be associated with significant purchases or transactions. Therefore, the presence of values ranging from -260 to over 2800 in the 'Amount' variable doesn't necessarily indicate outliers.

In other words, the observed range is plausible in the context of real-world financial transactions, and these values may accurately reflect the diversity of transaction amounts. As a result, it might not be appropriate to consider them as outliers without further domain-specific knowledge.

3.8 Visualizations

3.8.1 Univariate: Dimension 1

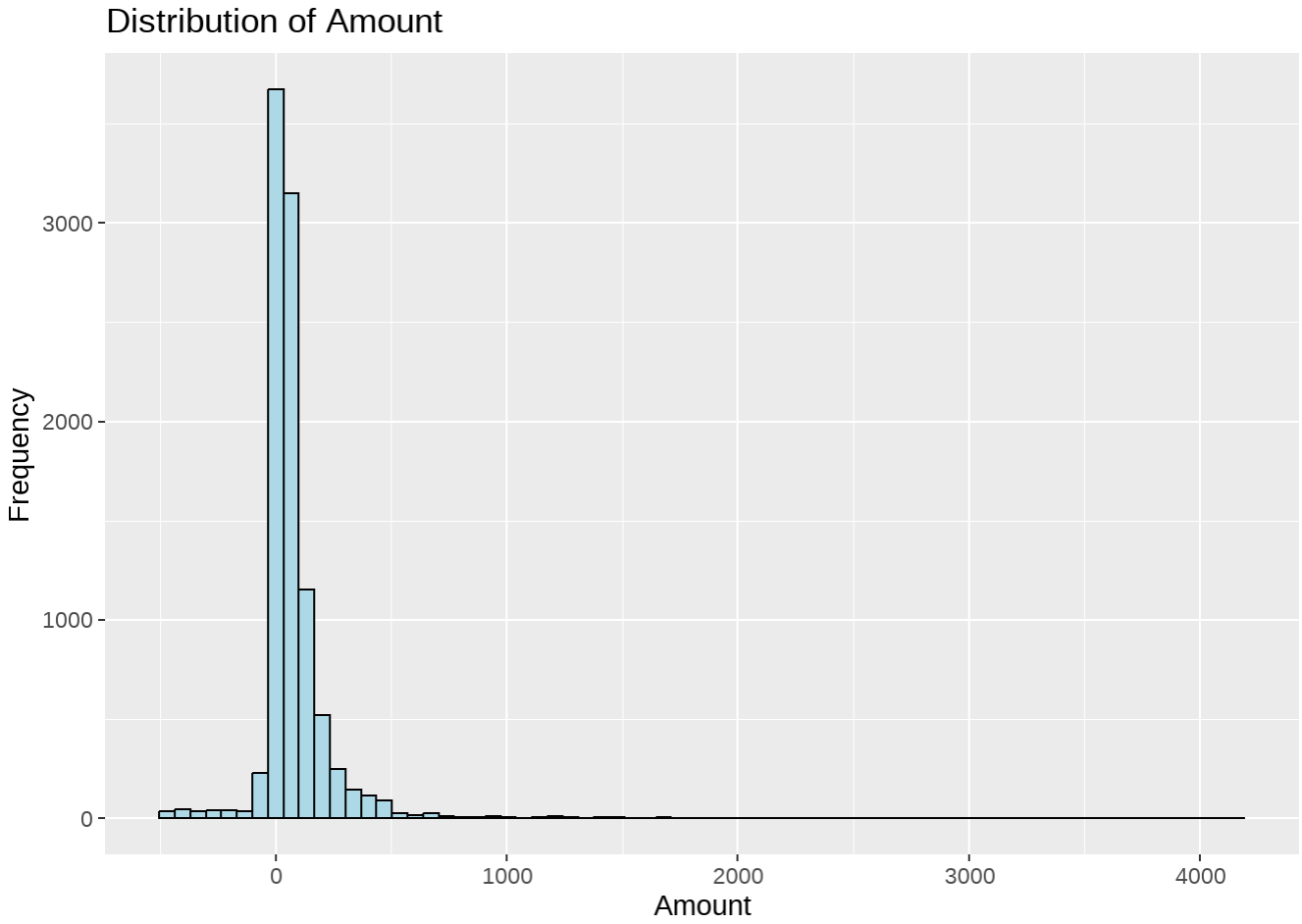
We have already seen the boxplots

```
ggplot(data = sampled_train_set, aes(x = Fraud, fill = Fraud)) +
  geom_bar() +
  labs(title = "Distribution of Fraudulent Transactions", x = "Fraud", y = "Count")
```

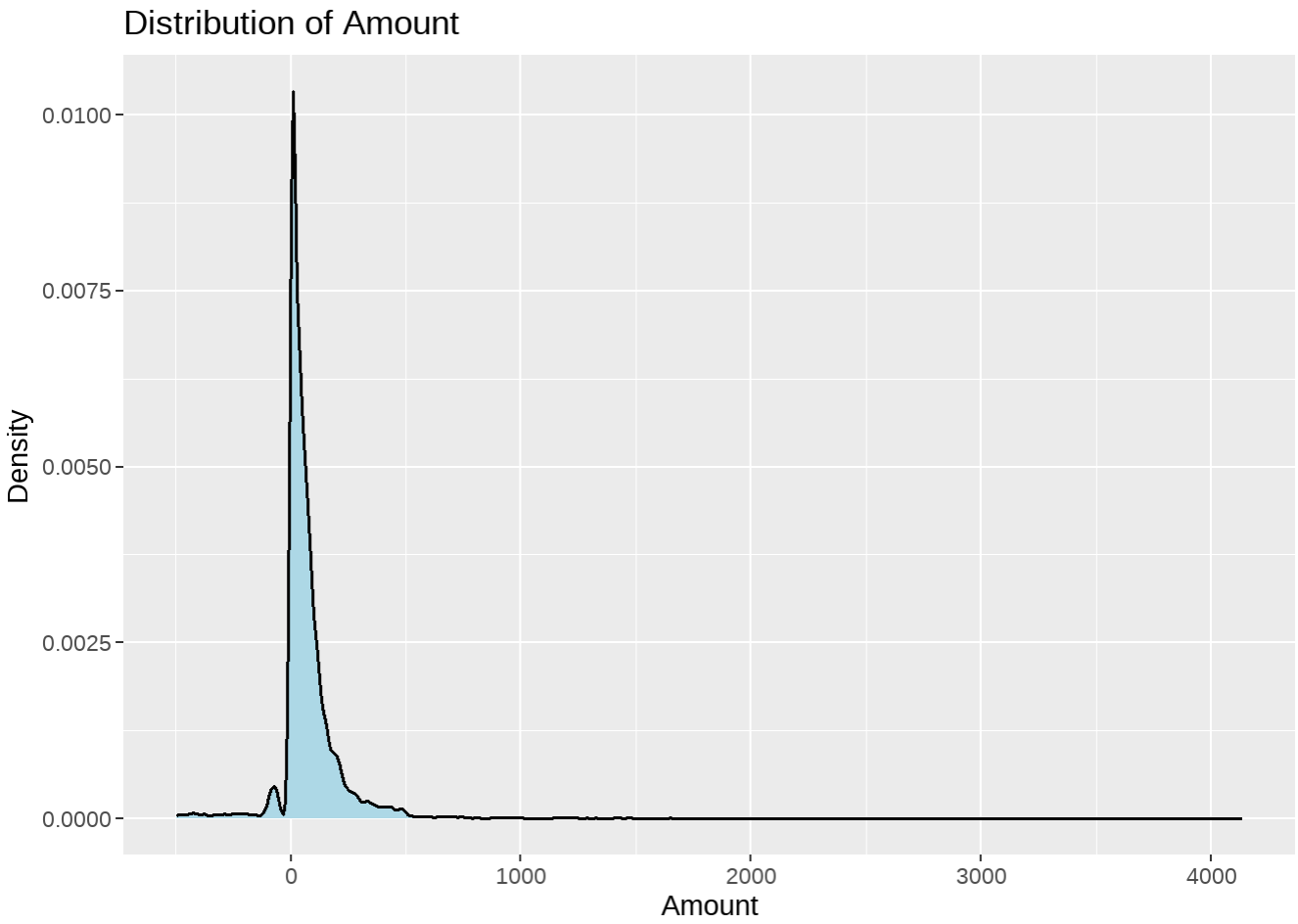
We can see that know we have balanced classes.

Histogram for Continuous Variables

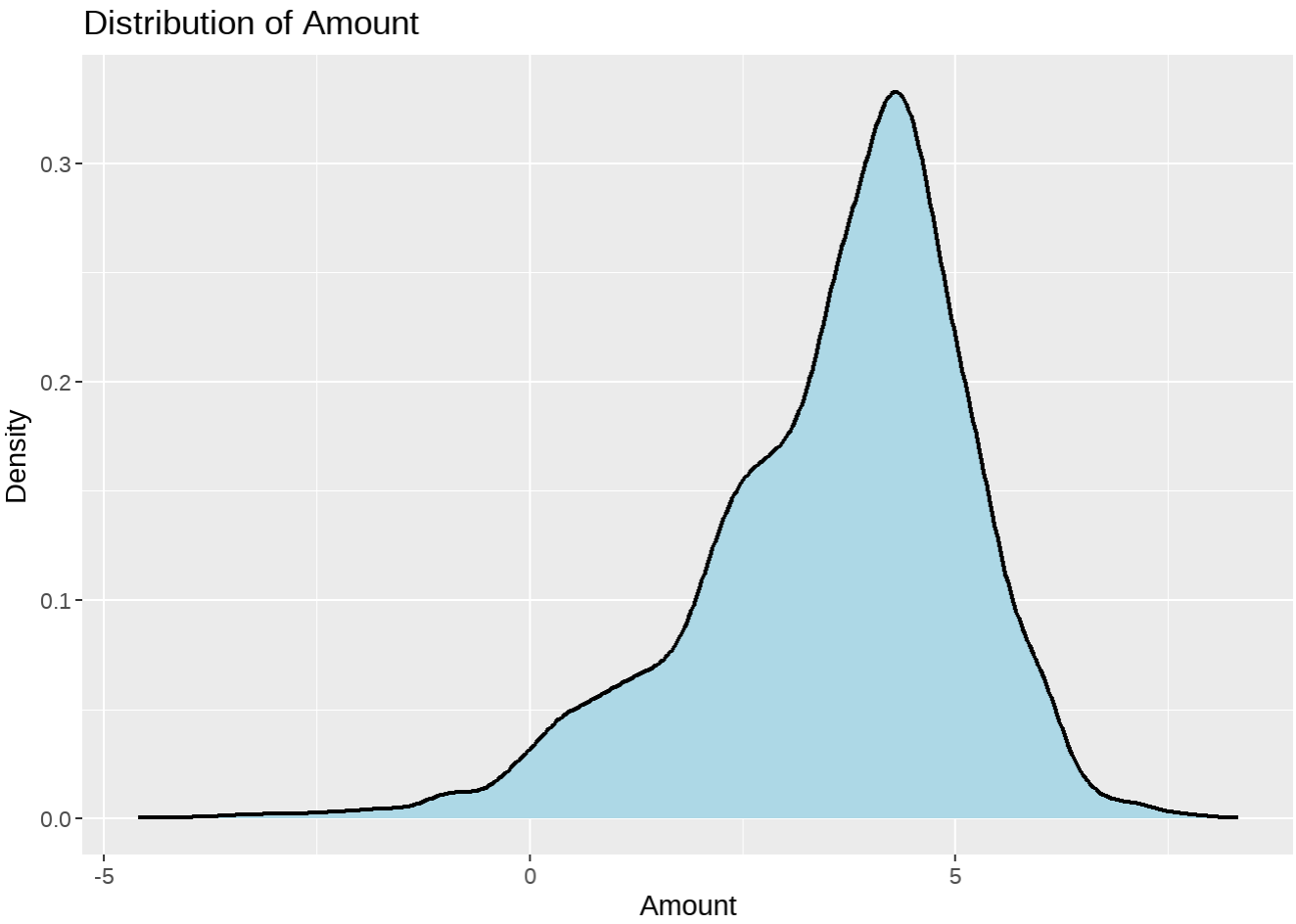
```
# Example for 'Amount'
library(ggplot2)

ggplot(sampled_train_set, aes(x = Amount)) +
  geom_histogram(fill = "lightblue", color = "black", bins = 70) +
  labs(title = "Distribution of Amount", x = "Amount", y = "Frequency")
```




```
ggplot(sampled_train_set, aes(x = Amount  )) +
  geom_density(fill = "lightblue", color = "black", size = 0.7, linetype =1) +
  #geom_rug(position = "identity", sides = "b", alpha = 0.5) +
  labs(title = "Distribution of Amount", x = "Amount", y = "Density")
```




```
ggplot(sampled_train_set, aes(x = log(Amount)  )) +
  geom_density(fill = "lightblue", color = "black", size = 0.8, linetype =1) +
  #geom_rug(position = "identity", sides = "b", alpha = 0.5) +
  labs(title = "Distribution of Amount", x = "Amount", y = "Density")
```



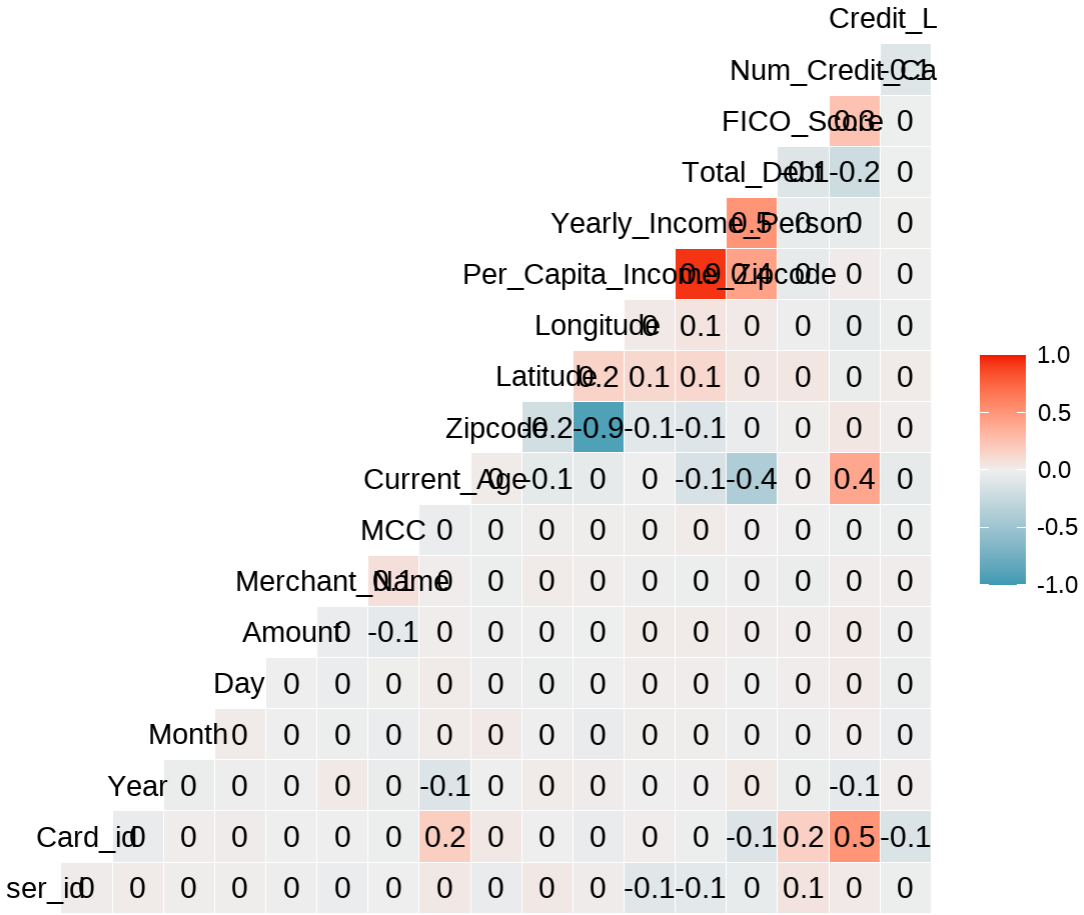
3.8.2 Bivariate

As we know the current age, we don't need the other variables related with the Age "Birth_Year", "Birth_Month", "Retirement_Age"

```
sampled_train_set =  sampled_train_set %>% dplyr::select(-Birth_Year, -Birth_Month, -Retiremen
t_Age)
sampled_test_set =  sampled_test_set %>% dplyr::select(-Birth_Year, -Birth_Month, -Retirement_
Age)
```

Correlation of numerical variables It can be seen the relationship between the numerical variables

```
numeric_data <- sampled_train_set[sapply(sampled_train_set, is.numeric)]

ggcorr(numeric_data, label = T)
```




```r
# Assuming 'sampled_train_set' is your sampled training set

correlation_matrix <- cor(numeric_data)

# Print the correlation matrix
print(correlation_matrix)
```

```
##                                 User_id       Card_id          Year
## User_id                    1.0000000000  0.0163555932  0.020351562
## Card_id                    0.0163555932  1.0000000000 -0.029526918
## Year                       0.0203515617 -0.0295269176  1.000000000
## Month                      0.0072948087  0.0114618895 -0.016871332
## Day                        0.0069546180  0.0158002105 -0.008518663
## Amount                     0.0039633354 -0.0012767257 -0.001148205
## Merchant_Name             -0.0180217468 -0.0005250306  0.031824043
## MCC                        0.0004984689 -0.0156698280 -0.026397522
## Current_Age                0.0383060403  0.1866166465 -0.114164976
## Zipcode                   -0.0318504309  0.0389548180 -0.002727487
## Latitude                   0.0339026444 -0.0004595777  0.022785491
## Longitude                  0.0171772687 -0.0333255872  0.016874282
## Per_Capita_Income_Zipcode -0.0833095907  0.0116738830  0.001235972
## Yearly_Income_Person      -0.0697527679 -0.0122200843  0.005855389
## Total_Debt                -0.0397440601 -0.1122481587  0.030943975
## FICO_Score                 0.0659241765  0.1736306630 -0.017202600
## Num_Credit_Cards           0.0284335850  0.5120858971 -0.056531676
```

```
## Credit_Limit               -0.0101032985 -0.1252071290  0.012586440
##                                    Month           Day        Amount
## User_id                     0.0072948087  0.0069546180  0.0039633354
## Card_id                     0.0114618895  0.0158002105 -0.0012767257
## Year                       -0.0168713316 -0.0085186630 -0.0011482053
## Month                       1.0000000000  0.0202456237 -0.0002091009
## Day                         0.0202456237  1.0000000000 -0.0015690645
## Amount                     -0.0002091009 -0.0015690645  1.0000000000
## Merchant_Name              -0.0114041252 -0.0220920025 -0.0208881732
## MCC                        -0.0179531511  0.0024505961 -0.0561109487
## Current_Age                 0.0248323090  0.0233384319  0.0110702640
## Zipcode                     0.0270092720 -0.0068881846  0.0044152765
## Latitude                   -0.0021380968 -0.0148396661  0.0005855749
## Longitude                  -0.0336869523 -0.0003793235 -0.0055126739
## Per_Capita_Income_Zipcode   0.0090603184  0.0120276636  0.0194844580
## Yearly_Income_Person        0.0072581759  0.0110148150  0.0221561477
## Total_Debt                  0.0037476592 -0.0004693306  0.0165396368
## FICO_Score                 -0.0072206699  0.0115029907  0.0136568263
## Num_Credit_Cards            0.0139439394  0.0279060307  0.0261355903
## Credit_Limit               -0.0185656581 -0.0174376716  0.0040983665
##                            Merchant_Name           MCC   Current_Age
## User_id                    -0.0180217468  0.0004984689  0.0383060403
## Card_id                    -0.0005250306 -0.0156698280  0.1866166465
## Year                        0.0318240429 -0.0263975217 -0.1141649763
## Month                      -0.0114041252 -0.0179531511  0.0248323090
## Day                        -0.0220920025  0.0024505961  0.0233384319
## Amount                     -0.0208881732 -0.0561109487  0.0110702640
## Merchant_Name               1.0000000000  0.0845534306  0.0117447501
## MCC                         0.0845534306  1.0000000000 -0.0208367879
## Current_Age                 0.0117447501 -0.0208367879  1.0000000000
## Zipcode                    -0.0116633954 -0.0130681698  0.0214682568
## Latitude                    0.0187819385  0.0098768207 -0.0638422464
## Longitude                   0.0110819358  0.0095331765 -0.0486030098
## Per_Capita_Income_Zipcode -0.0080306447  0.0088688572  0.0002455876
## Yearly_Income_Person      -0.0097931158  0.0187292045 -0.1391160122
## Total_Debt                  0.0033511749  0.0061413727 -0.3919869741
## FICO_Score                 -0.0173363498 -0.0008887257  0.0140305085
## Num_Credit_Cards            0.0143087090 -0.0142997316  0.4059490538
## Credit_Limit                0.0115426721 -0.0116399759 -0.0446892651
##                                 Zipcode      Latitude     Longitude
## User_id                    -0.031850431  0.0339026444  0.0171772687
## Card_id                     0.038954818 -0.0004595777 -0.0333255872
## Year                       -0.002727487  0.0227854913  0.0168742820
## Month                       0.027009272 -0.0021380968 -0.0336869523
## Day                        -0.006888185 -0.0148396661 -0.0003793235
## Amount                      0.004415276  0.0005855749 -0.0055126739
## Merchant_Name              -0.011663395  0.0187819385  0.0110819358
## MCC                        -0.013068170  0.0098768207  0.0095331765
## Current_Age                 0.021468257 -0.0638422464 -0.0486030098
## Zipcode                     1.000000000 -0.1805536936 -0.9190852782
## Latitude                   -0.180553694  1.0000000000  0.1579236310
## Longitude                  -0.919085278  0.1579236310  1.0000000000
## Per_Capita_Income_Zipcode -0.074981392  0.1300840344  0.0279371545
## Yearly_Income_Person       -0.099338733  0.1320553405  0.0578943106
```

```
## Total_Debt                -0.031557774  0.0479658550  0.0265076656
## FICO_Score                 0.007818779  0.0448874695 -0.0066650421
## Num_Credit_Cards           0.048605432 -0.0248196038 -0.0495729268
## Credit_Limit               0.013433726  0.0202048550 -0.0114606049
##                          Per_Capita_Income_Zipcode Yearly_Income_Person
## User_id                             -0.0833095907         -0.069752768
## Card_id                              0.0116738830         -0.012220084
## Year                                 0.0012359720          0.005855389
## Month                                0.0090603184          0.007258176
## Day                                  0.0120276636          0.011014815
## Amount                               0.0194844580          0.022156148
## Merchant_Name                       -0.0080306447         -0.009793116
## MCC                                  0.0088688572          0.018729204
## Current_Age                          0.0002455876         -0.139116012
## Zipcode                             -0.0749813916         -0.099338733
## Latitude                             0.1300840344          0.132055340
## Longitude                            0.0279371545          0.057894311
## Per_Capita_Income_Zipcode            1.0000000000          0.941771961
## Yearly_Income_Person                 0.9417719612          1.000000000
## Total_Debt                           0.4343090774          0.508578364
## FICO_Score                          -0.0491503435         -0.039019262
## Num_Credit_Cards                     0.0177724576         -0.040311272
## Credit_Limit                        -0.0002781096          0.002471505
##                             Total_Debt    FICO_Score Num_Credit_Cards
## User_id                   -0.0397440601  0.0659241765       0.02843359
## Card_id                   -0.1122481587  0.1736306630       0.51208590
## Year                       0.0309439748 -0.0172026003      -0.05653168
## Month                      0.0037476592 -0.0072206699       0.01394394
## Day                       -0.0004693306  0.0115029907       0.02790603
## Amount                     0.0165396368  0.0136568263       0.02613559
## Merchant_Name              0.0033511749 -0.0173363498       0.01430871
## MCC                        0.0061413727 -0.0008887257      -0.01429973
## Current_Age               -0.3919869741  0.0140305085       0.40594905
## Zipcode                   -0.0315577742  0.0078187792       0.04860543
## Latitude                   0.0479658550  0.0448874695      -0.02481960
## Longitude                  0.0265076656 -0.0066650421      -0.04957293
## Per_Capita_Income_Zipcode  0.4343090774 -0.0491503435       0.01777246
## Yearly_Income_Person       0.5085783636 -0.0390192616      -0.04031127
## Total_Debt                 1.0000000000 -0.1082206848      -0.21139652
## FICO_Score                -0.1082206848  1.0000000000       0.25081989
## Num_Credit_Cards          -0.2113965219  0.2508198949       1.00000000
## Credit_Limit              -0.0042896360 -0.0035052040      -0.09369573
##                            Credit_Limit
## User_id                   -0.0101032985
## Card_id                   -0.1252071290
## Year                       0.0125864402
## Month                     -0.0185656581
## Day                       -0.0174376716
## Amount                     0.0040983665
## Merchant_Name              0.0115426721
## MCC                       -0.0116399759
## Current_Age               -0.0446892651
## Zipcode                    0.0134337255
## Latitude                   0.0202048550
```

```
## Longitude                 -0.0114606049
## Per_Capita_Income_Zipcode -0.0002781096
## Yearly_Income_Person       0.0024715045
## Total_Debt                -0.0042896360
## FICO_Score                -0.0035052040
## Num_Credit_Cards          -0.0936957263
## Credit_Limit               1.0000000000
```

In this case, the correlation coefficient between Yearly_Income_Person and Per_Capita_Income_Zipcode is approximately 0.93. In practical terms, this high positive correlation suggests that areas with higher per capita income tend to have residents with higher yearly individual incomes. It's not surprising, as per capita income is calculated by dividing the total income of an area by its population, and areas with higher individual incomes are likely to contribute to a higher per capita income.

This strong relationship could lead to multicollinearity, making it challenging to interpret the individual contributions of each variable. However, While a high correlation suggests a strong statistical association, it doesn't imply causation. Other factors or variables may contribute to this relationship.

Total_Debt and Yearly_Income_Person: These variables show a moderate positive correlation, suggesting that as yearly income increases, total debt tends to increase as well.

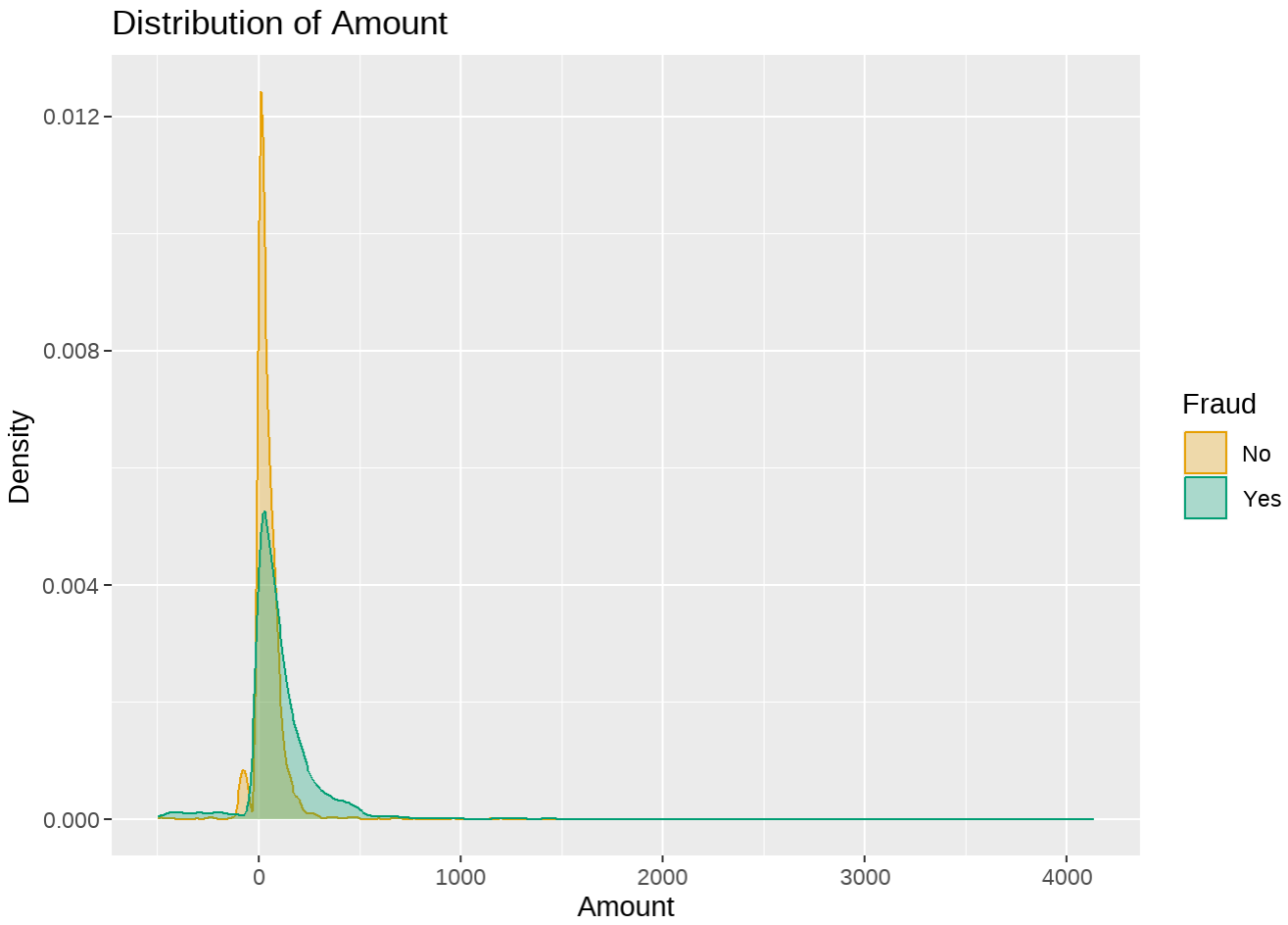
Current_Age and Num_Credit_Cards: There is a positive correlation between the current age of the person and the number of credit cards they have.

FICO_Score and Total_Debt: There is a negative correlation between FICO score and total debt, suggesting that individuals with higher FICO scores tend to have lower total debt.

Amount and Total_Debt: There is a positive correlation between the amount of the transaction and the total debt, suggesting that higher transaction amounts might be associated with higher total debt

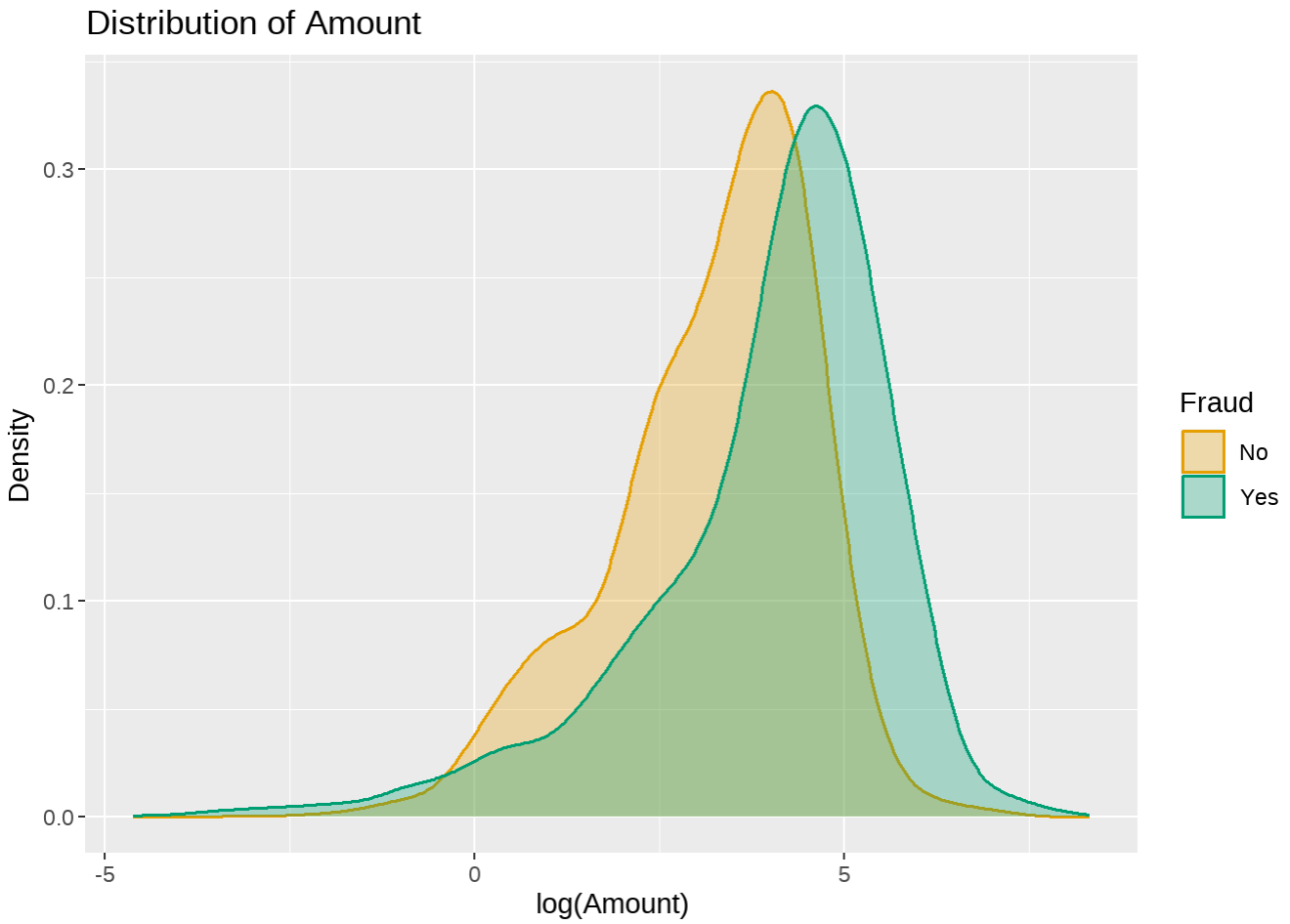
Distribution on amount depending on Fraud Class

```
ggplot(data=sampled_train_set, aes(x=Amount)) +
    geom_density(aes(group=Fraud, colour=Fraud, fill=Fraud), adjust=1.5, alpha=0.3)  +
  labs(title = "Distribution of Amount", x = "Amount", y = "Density")
```




```
    #theme_ipsum()
```

```
ggplot(data=sampled_train_set, aes(x=log(Amount) )) +
    geom_density(aes(group=Fraud, colour=Fraud, fill=Fraud), adjust=1.5, alpha=0.3, size = 0.6
, linetype =1) +
    labs(title = "Distribution of Amount", x = "log(Amount)", y = "Density")
```

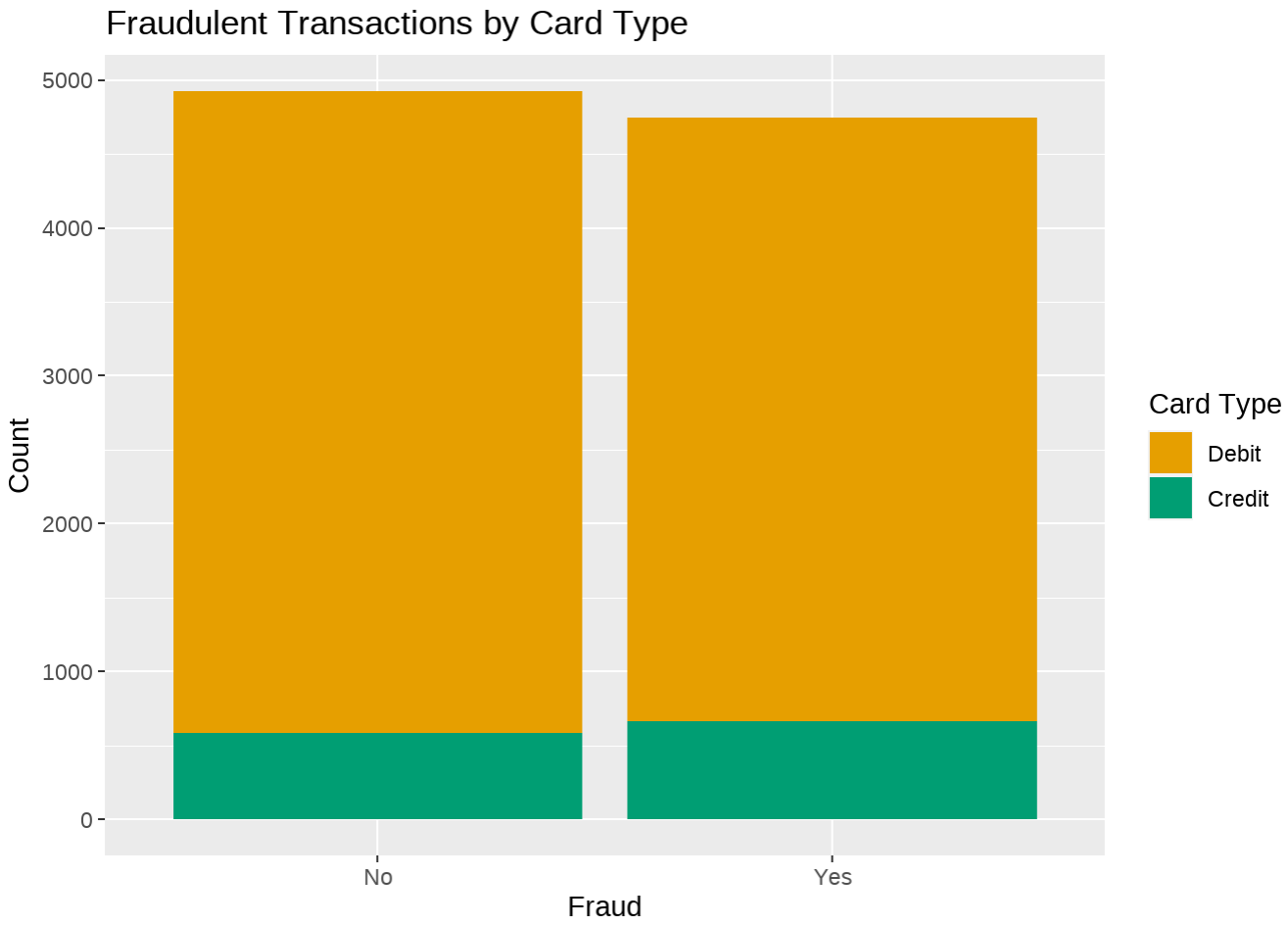



```
    #theme_ipsum()
```

It seems that the transactions with 'LogAmount' larger than 4.5 (90 dollars) and smaller than -0.5 (0.6 dollars) have a higher frequency and probability density of being fraudulent. On the other hand, the ones with 'LogAmount' from -0.5 to 4.5 have a higher chance of being legit.

Fraudulent Transactions by Card Type

```
ggplot(data = sampled_train_set, aes(x = Fraud, fill = Card_Type)) +
  geom_bar(position = "stack") +
  labs(title = "Fraudulent Transactions by Card Type", x = "Fraud", y = "Count", fill = "Card
Type")
```

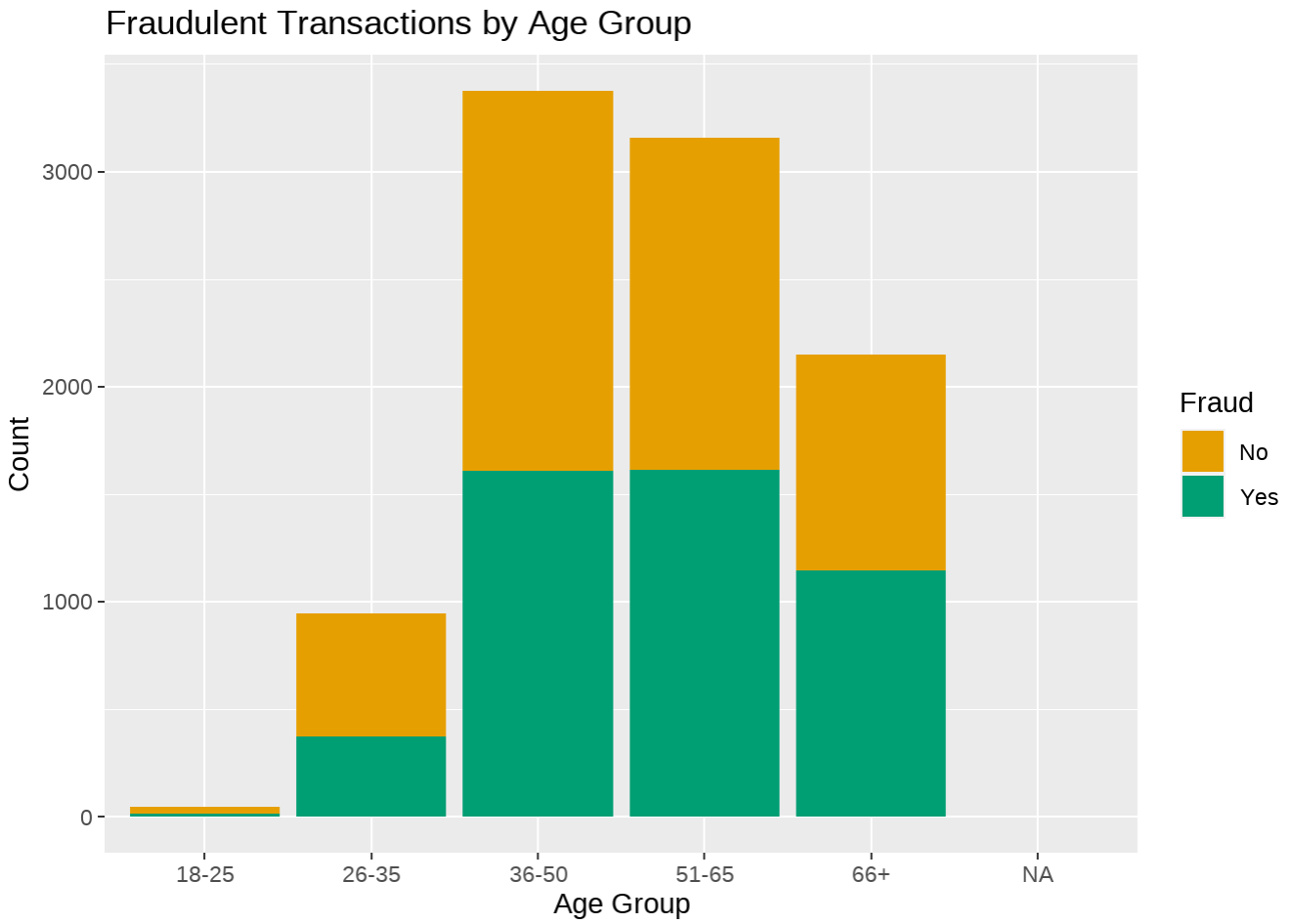


The proportion of card types is similar for both types of transactions, and most transactions, regardless of fraud status, are made with debit cards. This suggests that the distribution of card types might not be a differentiating factor for fraud detection in this dataset.

Fraudulent Transactions by Age Group

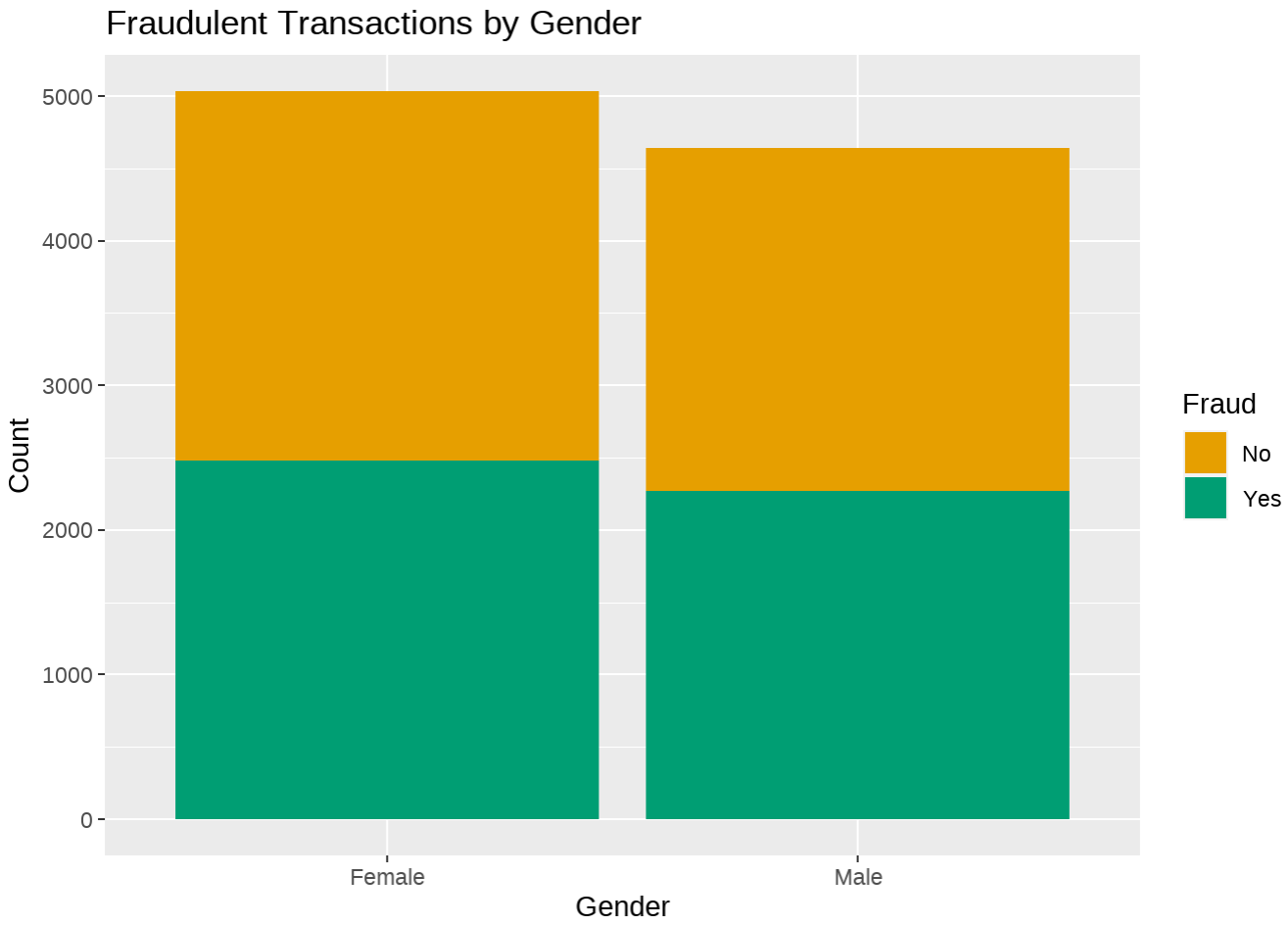
Discretizing age and creating a graph between fraud and age groups could be a useful approach. It might help identify if certain age groups are more susceptible to fraud or if there are patterns in fraudulent activities based on age.

```
# Assuming you have a variable 'Age' in your dataset

# Create age groups
sampled_train_set$Age_Group <- cut(sampled_train_set$Current_Age, breaks = c(18, 25, 35, 50, 6
5, max(sampled_train_set$Current_Age)), labels = c("18-25", "26-35", "36-50", "51-65", "66+"))

sampled_test_set$Age_Group <- cut(sampled_test_set$Current_Age, breaks = c(18, 25, 35, 50, 65,
 max(sampled_test_set$Current_Age)), labels = c("18-25", "26-35", "36-50", "51-65", "66+"))

# Create a bar plot
ggplot(data = sampled_train_set, aes(x = Age_Group, fill = Fraud)) +
  geom_bar(position = "stack") +
  labs(title = "Fraudulent Transactions by Age Group", x = "Age Group", y = "Count", fill = "F
raud")
```



We can see that in 36-50, 51-65 and 66+ groups there are much more transactions. The amount of fraudulent transaction in this groups represent almost the half of their transactions. In 26-35 group the fraudulent transactions represent a third of their transactions. Finally, in group 18-25 there are not almost fraudulent transactions.

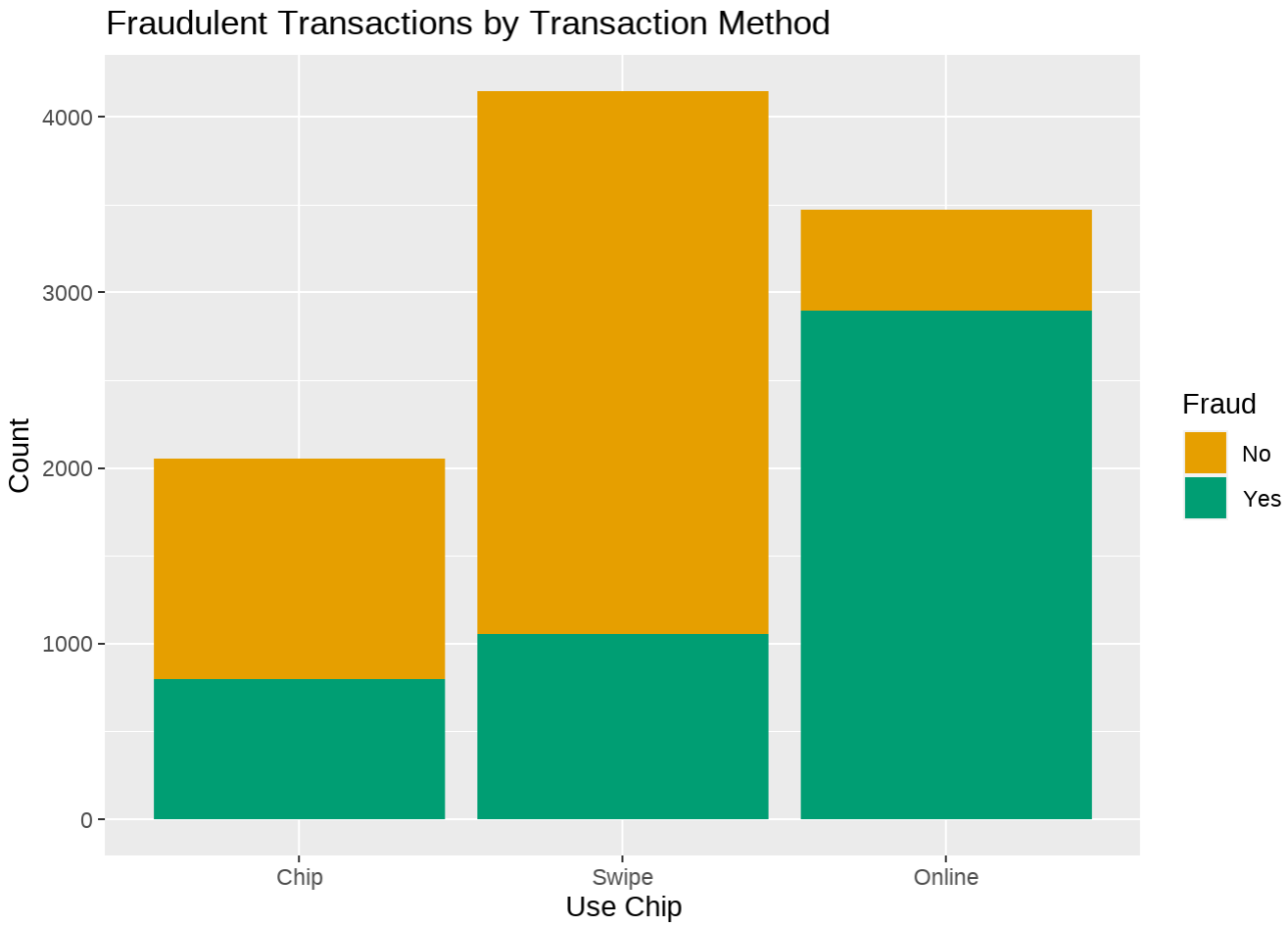
```
# Create a bar plot
ggplot(data = sampled_train_set, aes(x = Gender, fill = Fraud)) +
  geom_bar(position = "stack") +
  labs(title = "Fraudulent Transactions by Gender", x = "Gender", y = "Count", fill = "Fraud")
```



We can see a similar proportion, although for females it is slightly bigger.

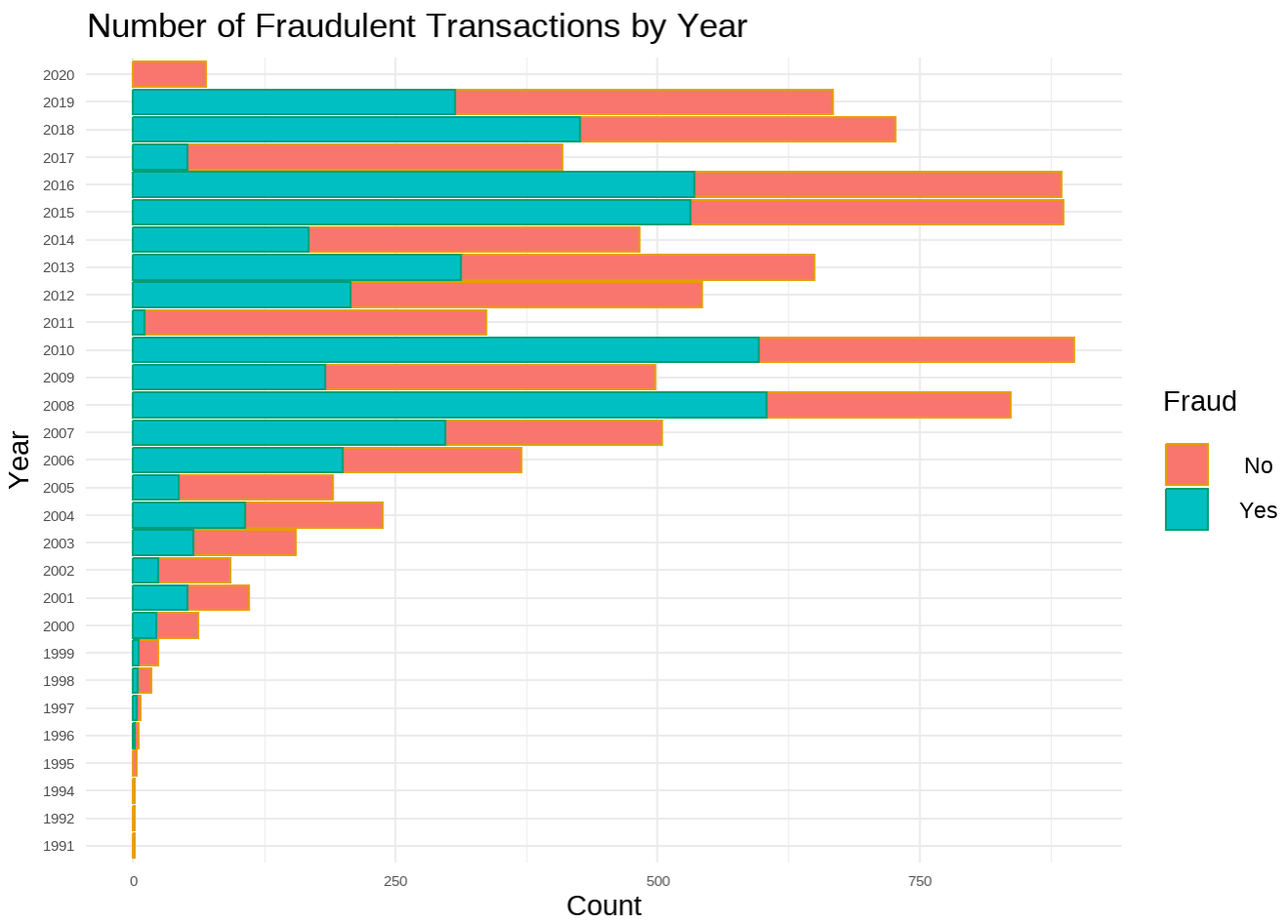
Transaction Method

```
ggplot(data = sampled_train_set, aes(x = Use_Chip, fill = Fraud)) +
  geom_bar(position = "stack") +
  labs(title = "Fraudulent Transactions by Transaction Method", x = "Use Chip", y = "Count", f
ill = "Fraud")
```



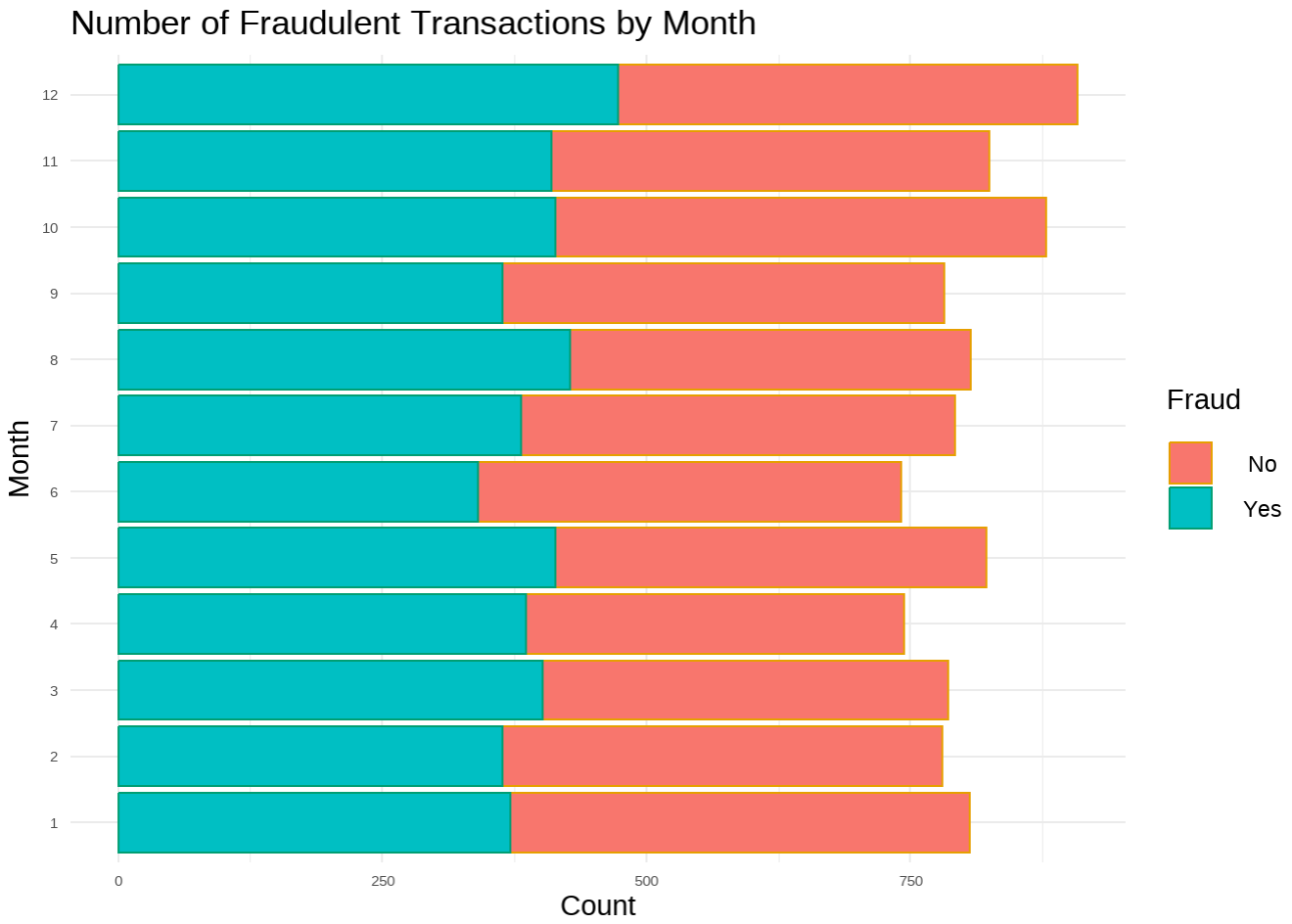
We can see that online transactions present the most significant vulnerability to fraud, as the majority are fraudulent.

```
ggplot(data = sampled_train_set, aes(x = as.factor(Year))) +
  geom_bar(aes(y = after_stat(count), fill = Fraud, color = Fraud), stat = "count") +
  labs(title = "Number of Fraudulent Transactions by Year", x = "Year", y = "Count") +
  scale_fill_manual(values = c("#F8766D", "#00BFC3" ), name = "Fraud") +  # Specify fill colors
  #scale_color_manual(values = c("darkred", "darkblue"), name = "Fraud") +  # Specify border colors
  theme_minimal() +
  coord_flip()  # Rotate the plot
```



The provided statement suggests a concerning trend in the 21st century, with a consistent rise in the number of fraud cases year by year, culminating in a peak during the years 2007-2008, coinciding with the Great Recession. This correlation between the surge in fraud and the economic downturn may indicate a heightened vulnerability to fraudulent activities during periods of financial stress. The Great Recession, characterized by widespread economic challenges, likely created an environment conducive to fraudulent behavior, with individuals and organizations exploiting vulnerabilities for financial gain. This observation underscores the complex relationship between economic conditions and fraudulent activities, emphasizing the need for vigilance and effective preventive measures during times of economic instability.

```
ggplot(data = sampled_train_set, aes(x = as.factor(Month))) +
  geom_bar(aes(y = after_stat(count), fill = Fraud, color = Fraud), stat = "count") +
  labs(title = "Number of Fraudulent Transactions by Month", x = "Month", y = "Count") +
  scale_fill_manual(values = c("#F8766D", "#00BFC3" ), name = "Fraud") +  # Specify fill color
s
  #scale_color_manual(values = c("darkred", "darkblue"), name = "Fraud") +  # Specify border c
olors
  theme_minimal() +
  coord_flip()  # Rotate the plot
```



The distribution of fraud cases across months doesn't show significant variations. This observation and is indicative that fraudulent activities might not be strongly influenced by the specific month.

3.9 Feature Selection

Later there will be used more sophisticated ways for doing this.

Let's consider variables for potential removal based on various criteria:

1. *Identifiers:* "User_id" and "Card_id" are likely unique identifiers and may not contribute much predictive power to the model. As well as "Person" which corresponds with the name of the person.
2. *Location Information:* "Merchant_Name," "Address," "Zipcode," "Latitude," and "Longitude" , because they are not relevant. In addition to this, we are going to remove "Merchant_State" as there are too many cells with empty strings ""
3. *"Current_Age"* : As we have already discretize the age and we have "Age_Group"
4. *Date and Time Components:* "Year," "Month," "Day," and "Time" might be redundant if we already have a variable like "Date" that captures this information.
5. *"Card_on_Dark_Web"*: As all its values are "No", it does not provide any information.
6. *"MCC" (Merchant Category Code):* Remove it as there are too many codes.

```
sampled_train_set =  sampled_train_set %>% dplyr::select(-User_id, -Card_id, -Person, -Merchan
t_Name, -Address, -Zipcode, -Latitude, -Longitude, -Merchant_State,- Current_Age, -Year, -Mont
```

```
h, -Day, -Time, -Card_on_Dark_Web, -MCC)

sampled_test_set =   sampled_test_set %>% dplyr::select(-User_id, -Card_id, -Person, -Merchant
_Name, -Address, -Zipcode, -Latitude, -Longitude, -Merchant_State,- Current_Age, -Year, -Month
, -Day, -Time, -Card_on_Dark_Web, -MCC)
```

```
fwrite(sampled_train_set, "sampled_train_set_class.csv")
fwrite(sampled_test_set, "sampled_test_set_class.csv")
```

```
sampled_train_set_class <- read.csv("sampled_train_set_class.csv")
sampled_test_set_class <- read.csv("sampled_test_set_class.csv")
```

This project began with 36 variables, and know it has 16.

4 CLASSIFICATION

4.1 Statistical Tools for Classification

4.1.1 Penalized Logistic Regression

(lab customer analysis)

Introduction to Logistic Regression

Penalized logistic regression, specifically implemented through the glmnet package in R, offers several advantages for fraud classification tasks with numerous variables. The regularization techniques, such as Lasso (L1) and Ridge (L2), address challenges associated with high-dimensional datasets, preventing overfitting and improving generalization performance. By automatically selecting relevant variables and stabilizing coefficients, penalized logistic regression enhances the model's interpretability and efficiency in the presence of multicollinearity. glmnet's flexibility allows for the fine-tuning of regularization parameters through techniques like cross-validation, enabling the development of a more robust and parsimonious fraud detection model that not only excels in prediction but also provides insights into the underlying patterns contributing to fraudulent activities.

Applicability in Fraud Classification

Logistic regression proves to be a valuable tool for fraud classification due to its simplicity, interpretability, and effectiveness in handling binary outcomes. In fraud detection, the goal is to distinguish between legitimate and fraudulent transactions or activities. Logistic regression excels in this context by modeling the probability of an event (fraudulent or not) based on a set of features. The coefficients of the logistic regression model provide insights into the impact of each feature on the likelihood of fraud, facilitating the identification of key factors contributing to fraudulent behavior. Moreover, logistic regression is less prone to overfitting, making it robust with limited data and offering a transparent decision-making process crucial for understanding and validating predictions in sensitive applications like fraud detection.

Interpretable and Efficient Fraud Detection

The interpretability of logistic regression enhances its appeal in fraud classification scenarios where explainability is vital. Stakeholders, investigators, and decision-makers can easily comprehend the factors influencing the model's predictions, fostering trust in the decision-making process. Logistic regression is computationally efficient, making it well-suited for real-time or near-real-time fraud detection systems, where quick and accurate decisions are imperative. Its ability to handle

high-dimensional data and adapt to changing circumstances further solidifies logistic regression as a pragmatic choice for fraud classification, providing a balance between simplicity, interpretability, and predictive performance.

The **ctrl configuration** for the caret package's train function establishes a robust training control structure for implementing penalized logistic regression using techniques like Lasso or Ridge regularization. The choice of 5-fold cross-validation (method = "cv", number = 5) ensures a thorough evaluation of model performance by repeatedly splitting the dataset into training and testing sets. This aids in mitigating overfitting and provides a more accurate estimate of the model's effectiveness on unseen data. Additionally, setting classProbs = TRUE enables the generation of class probabilities, a crucial aspect in fraud detection where understanding the likelihood of an event being fraudulent is essential. The verboseIter = TRUE setting facilitates monitoring and reporting the progress of model training iterations, enhancing transparency and aiding in the interpretation of the penalized logistic regression results. This control structure, tailored for your specific project requirements, contributes to the development of a reliable and interpretable fraud detection model.

lrFit represents the result of training a penalized logistic regression model using the glmnet algorithm. The logistic regression model predicts the binary outcome variable "Fraud" based on all available predictor variables in the dataset. The model undergoes hyperparameter tuning with a grid search over alpha and lambda values to optimize its performance. Training is conducted on a subset of the dataset (sampled_train_set) with 5-fold cross-validation, enabling robust evaluation. Additionally, the model is assessed using Cohen's Kappa, a suitable metric for imbalanced classification tasks, and preprocessing involves centering and scaling predictor variables. The resulting lrFit encapsulates a well-tailored logistic regression model for fraud detection, addressing both the complexities of the dataset and the specific requirements of the project.

```
### Use_Chip
sampled_train_set_class$Use_Chip= factor(x = sampled_train_set_class$Use_Chip,
                                  levels = c("Chip", "Swipe", "Online"),
                                  labels =  c("Chip", "Swipe", "Online")
                                  )

sampled_test_set_class$Use_Chip= factor(x = sampled_test_set_class$Use_Chip,
                                  levels = c("Chip", "Swipe", "Online"),
                                  labels =  c("Chip", "Swipe", "Online")
                                  )


### Card_Type
sampled_train_set_class$Card_Type = factor(x = sampled_train_set_class$Card_Type,
                                   levels = c("Debit", "Credit"),
                                  labels = c("Debit", "Credit")
                                  )

sampled_test_set_class$Card_Type = factor(x = sampled_test_set_class$Card_Type,
                                   levels = c("Debit", "Credit"),
                                  labels = c("Debit", "Credit")
                                  )




### Gender
sampled_train_set_class$Gender = factor(x = sampled_train_set_class$Gender,
                                 levels = c("Female", "Male"),
```

```
                                labels = c("Female", "Male") )

sampled_test_set_class$Gender = factor(x = sampled_test_set_class$Gender,
                                levels = c("Female", "Male"),
                                labels = c("Female", "Male") )


### Fraud
sampled_train_set_class$Fraud = factor(x = sampled_train_set_class$Fraud)
sampled_test_set_class$Fraud = factor(x = sampled_test_set_class$Fraud)

### Merchant_City
sampled_train_set_class$Merchant_City = as.factor(sampled_train_set_class$Merchant_City)
sampled_test_set_class$Merchant_City = as.factor(sampled_test_set_class$Merchant_City)



### City
sampled_train_set_class$City = as.factor(sampled_train_set_class$City)
sampled_test_set_class$City = as.factor(sampled_test_set_class$City)

### State
sampled_train_set_class$State = as.factor(sampled_train_set_class$State)
sampled_test_set_class$State = as.factor(sampled_test_set_class$State)

# Assuming that Year, Month, and Day are already numeric
sampled_train_set_class$Date <- as.Date(sampled_train_set_class$Date)
sampled_test_set_class$Date <- as.Date(sampled_test_set_class$Date)


sampled_train_set_class$Age_Group = as.factor(sampled_train_set_class$Age_Group)
sampled_test_set_class$Age_Group = as.factor(sampled_test_set_class$Age_Group)
```

```
sampled_train_set_class =  sampled_train_set_class %>% dplyr::select(-Merchant_City, -City,-St
ate)
sampled_test_set_class =  sampled_test_set_class %>% dplyr::select(-Merchant_City, -City, -Sta
te)
```

```
set.seed(432)
# Assuming you have the levels defined for "Age_Group" in your training set
age_levels <- levels(sampled_train_set_class$Age_Group)

# Ensure the same levels in the test set to avoid errors
sampled_test_set_class$Age_Group <- factor(sampled_test_set_class$Age_Group, levels = age_leve
ls)

ctrl <- trainControl(method = "cv", number = 5,
                     classProbs = TRUE,
                     verboseIter=T)

# Now, you can fit the model and make predictions
lrFit <- train(Fraud ~ .,
```

```
                method = "glmnet",
                tuneGrid = expand.grid(alpha = seq(0, 1, 0.1), lambda = seq(0, .1, 0.02)),
                metric = "Kappa",
                data = sampled_train_set_class,
                preProcess = c("center", "scale"),
                trControl = ctrl)
```

```
## + Fold1: alpha=0.0, lambda=0.1
## - Fold1: alpha=0.0, lambda=0.1
## + Fold1: alpha=0.1, lambda=0.1
## - Fold1: alpha=0.1, lambda=0.1
## + Fold1: alpha=0.2, lambda=0.1
## - Fold1: alpha=0.2, lambda=0.1
## + Fold1: alpha=0.3, lambda=0.1
## - Fold1: alpha=0.3, lambda=0.1
## + Fold1: alpha=0.4, lambda=0.1
## - Fold1: alpha=0.4, lambda=0.1
## + Fold1: alpha=0.5, lambda=0.1
## - Fold1: alpha=0.5, lambda=0.1
## + Fold1: alpha=0.6, lambda=0.1
## - Fold1: alpha=0.6, lambda=0.1
## + Fold1: alpha=0.7, lambda=0.1
## - Fold1: alpha=0.7, lambda=0.1
## + Fold1: alpha=0.8, lambda=0.1
## - Fold1: alpha=0.8, lambda=0.1
## + Fold1: alpha=0.9, lambda=0.1
## - Fold1: alpha=0.9, lambda=0.1
## + Fold1: alpha=1.0, lambda=0.1
## - Fold1: alpha=1.0, lambda=0.1
## + Fold2: alpha=0.0, lambda=0.1
## - Fold2: alpha=0.0, lambda=0.1
## + Fold2: alpha=0.1, lambda=0.1
## - Fold2: alpha=0.1, lambda=0.1
## + Fold2: alpha=0.2, lambda=0.1
## - Fold2: alpha=0.2, lambda=0.1
## + Fold2: alpha=0.3, lambda=0.1
## - Fold2: alpha=0.3, lambda=0.1
## + Fold2: alpha=0.4, lambda=0.1
## - Fold2: alpha=0.4, lambda=0.1
## + Fold2: alpha=0.5, lambda=0.1
## - Fold2: alpha=0.5, lambda=0.1
## + Fold2: alpha=0.6, lambda=0.1
## - Fold2: alpha=0.6, lambda=0.1
## + Fold2: alpha=0.7, lambda=0.1
## - Fold2: alpha=0.7, lambda=0.1
## + Fold2: alpha=0.8, lambda=0.1
## - Fold2: alpha=0.8, lambda=0.1
## + Fold2: alpha=0.9, lambda=0.1
## - Fold2: alpha=0.9, lambda=0.1
## + Fold2: alpha=1.0, lambda=0.1
## - Fold2: alpha=1.0, lambda=0.1
## + Fold3: alpha=0.0, lambda=0.1
## - Fold3: alpha=0.0, lambda=0.1
```

```
## + Fold3: alpha=0.1, lambda=0.1
## - Fold3: alpha=0.1, lambda=0.1
## + Fold3: alpha=0.2, lambda=0.1
## - Fold3: alpha=0.2, lambda=0.1
## + Fold3: alpha=0.3, lambda=0.1
## - Fold3: alpha=0.3, lambda=0.1
## + Fold3: alpha=0.4, lambda=0.1
## - Fold3: alpha=0.4, lambda=0.1
## + Fold3: alpha=0.5, lambda=0.1
## - Fold3: alpha=0.5, lambda=0.1
## + Fold3: alpha=0.6, lambda=0.1
## - Fold3: alpha=0.6, lambda=0.1
## + Fold3: alpha=0.7, lambda=0.1
## - Fold3: alpha=0.7, lambda=0.1
## + Fold3: alpha=0.8, lambda=0.1
## - Fold3: alpha=0.8, lambda=0.1
## + Fold3: alpha=0.9, lambda=0.1
## - Fold3: alpha=0.9, lambda=0.1
## + Fold3: alpha=1.0, lambda=0.1
## - Fold3: alpha=1.0, lambda=0.1
## + Fold4: alpha=0.0, lambda=0.1
## - Fold4: alpha=0.0, lambda=0.1
## + Fold4: alpha=0.1, lambda=0.1
## - Fold4: alpha=0.1, lambda=0.1
## + Fold4: alpha=0.2, lambda=0.1
## - Fold4: alpha=0.2, lambda=0.1
## + Fold4: alpha=0.3, lambda=0.1
## - Fold4: alpha=0.3, lambda=0.1
## + Fold4: alpha=0.4, lambda=0.1
## - Fold4: alpha=0.4, lambda=0.1
## + Fold4: alpha=0.5, lambda=0.1
## - Fold4: alpha=0.5, lambda=0.1
## + Fold4: alpha=0.6, lambda=0.1
## - Fold4: alpha=0.6, lambda=0.1
## + Fold4: alpha=0.7, lambda=0.1
## - Fold4: alpha=0.7, lambda=0.1
## + Fold4: alpha=0.8, lambda=0.1
## - Fold4: alpha=0.8, lambda=0.1
## + Fold4: alpha=0.9, lambda=0.1
## - Fold4: alpha=0.9, lambda=0.1
## + Fold4: alpha=1.0, lambda=0.1
## - Fold4: alpha=1.0, lambda=0.1
## + Fold5: alpha=0.0, lambda=0.1
## - Fold5: alpha=0.0, lambda=0.1
## + Fold5: alpha=0.1, lambda=0.1
## - Fold5: alpha=0.1, lambda=0.1
## + Fold5: alpha=0.2, lambda=0.1
## - Fold5: alpha=0.2, lambda=0.1
## + Fold5: alpha=0.3, lambda=0.1
## - Fold5: alpha=0.3, lambda=0.1
## + Fold5: alpha=0.4, lambda=0.1
## - Fold5: alpha=0.4, lambda=0.1
## + Fold5: alpha=0.5, lambda=0.1
## - Fold5: alpha=0.5, lambda=0.1
```

```
## + Fold5: alpha=0.6, lambda=0.1
## - Fold5: alpha=0.6, lambda=0.1
## + Fold5: alpha=0.7, lambda=0.1
## - Fold5: alpha=0.7, lambda=0.1
## + Fold5: alpha=0.8, lambda=0.1
## - Fold5: alpha=0.8, lambda=0.1
## + Fold5: alpha=0.9, lambda=0.1
## - Fold5: alpha=0.9, lambda=0.1
## + Fold5: alpha=1.0, lambda=0.1
## - Fold5: alpha=1.0, lambda=0.1
## Aggregating results
## Selecting tuning parameters
## Fitting alpha = 0.5, lambda = 0.02 on full training set
```

```
print(lrFit)
```

```
## glmnet
##
## 9672 samples
##   12 predictor
##    2 classes: 'No', 'Yes'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7738, 7737, 7737, 7739, 7737
## Resampling results across tuning parameters:
##
##   alpha  lambda  Accuracy   Kappa
##   0.0    0.00    0.7564101  0.5109184
##   0.0    0.02    0.7564101  0.5109184
##   0.0    0.04    0.7556863  0.5094598
##   0.0    0.06    0.7557895  0.5096512
##   0.0    0.08    0.7551689  0.5084087
##   0.0    0.10    0.7543416  0.5067660
##   0.1    0.00    0.7565129  0.5111451
##   0.1    0.02    0.7566170  0.5113047
##   0.1    0.04    0.7559963  0.5100182
##   0.1    0.06    0.7553760  0.5087469
##   0.1    0.08    0.7553759  0.5087319
##   0.1    0.10    0.7548591  0.5076667
##   0.2    0.00    0.7565130  0.5111452
##   0.2    0.02    0.7565134  0.5110766
##   0.2    0.04    0.7564100  0.5107827
##   0.2    0.06    0.7554796  0.5088575
##   0.2    0.08    0.7553765  0.5086007
##   0.2    0.10    0.7546527  0.5071154
##   0.3    0.00    0.7566164  0.5113546
##   0.3    0.02    0.7568237  0.5116609
##   0.3    0.04    0.7558933  0.5096734
##   0.3    0.06    0.7548596  0.5075311
##   0.3    0.08    0.7544458  0.5066372
##   0.3    0.10    0.7535154  0.5047141
```

```
##   0.4    0.00    0.7565130  0.5111452
##   0.4    0.02    0.7571338  0.5122452
##   0.4    0.04    0.7549628  0.5077512
##   0.4    0.06    0.7537221  0.5051550
##   0.4    0.08    0.7518612  0.5013415
##   0.4    0.10    0.7503100  0.4981701
##   0.5    0.00    0.7566164  0.5113510
##   0.5    0.02    0.7576510  0.5132458
##   0.5    0.04    0.7547561  0.5072732
##   0.5    0.06    0.7518611  0.5013415
##   0.5    0.08    0.7502067  0.4979496
##   0.5    0.10    0.7485523  0.4945792
##   0.6    0.00    0.7565131  0.5111417
##   0.6    0.02    0.7564102  0.5107060
##   0.6    0.04    0.7533088  0.5043178
##   0.6    0.06    0.7501033  0.4977509
##   0.6    0.08    0.7486557  0.4947851
##   0.6    0.10    0.7482422  0.4939312
##   0.7    0.00    0.7565131  0.5111417
##   0.7    0.02    0.7555831  0.5090116
##   0.7    0.04    0.7514477  0.5005143
##   0.7    0.06    0.7492762  0.4960511
##   0.7    0.08    0.7483456  0.4941408
##   0.7    0.10    0.7483456  0.4941257
##   0.8    0.00    0.7566165  0.5113475
##   0.8    0.02    0.7551696  0.5081404
##   0.8    0.04    0.7506204  0.4988183
##   0.8    0.06    0.7485523  0.4945754
##   0.8    0.08    0.7482422  0.4939198
##   0.8    0.10    0.7483456  0.4941257
##   0.9    0.00    0.7566165  0.5113475
##   0.9    0.02    0.7549628  0.5076995
##   0.9    0.04    0.7503101  0.4981555
##   0.9    0.06    0.7483456  0.4941408
##   0.9    0.08    0.7483456  0.4941257
##   0.9    0.10    0.7483456  0.4941257
##   1.0    0.00    0.7566165  0.5113475
##   1.0    0.02    0.7536188  0.5049643
##   1.0    0.04    0.7490694  0.4956317
##   1.0    0.06    0.7482422  0.4939198
##   1.0    0.08    0.7483456  0.4941257
##   1.0    0.10    0.7483456  0.4941257
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.5 and lambda = 0.02.
```

- Selected tuning parameters: Alpha: The alpha parameter in logistic regression controls the type of regularization applied. In this case, it is set to 0.5, indicating a combination of L1 (Lasso) and L2 (Ridge) regularization.

  Lambda: Lambda is the regularization strength. lambda of 0.02

Accuracy: 75.7% Kappa: 5.3% (moderate agreement beyond chance) The accuracy tells you the proportion of correctly predicted instances, while Kappa takes into account the possibility of agreement occurring by chance.

Following model training, the evaluation focused on assessing the model's performance using a separate test set. The predict function was utilized to generate predictions on the test set, and a confusion matrix was computed to summarize the classification results.

```
lrPred <- predict(lrFit, sampled_test_set_class)
lrPred <- factor(lrPred, levels = levels(sampled_test_set_class$Fraud))


sampled_test_set_class$Fraud <- factor(sampled_test_set_class$Fraud, levels = c("No", "Yes"))

# Obtain the confusion matrix
cm <- confusionMatrix(lrPred, sampled_test_set_class$Fraud, positive = "Yes")
print(cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2602    2
##        Yes  336    3
##
##                Accuracy : 0.8852
##                  95% CI : (0.8731, 0.8965)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0141
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.600000
##             Specificity : 0.885636
##          Pos Pred Value : 0.008850
##          Neg Pred Value : 0.999232
##              Prevalence : 0.001699
##          Detection Rate : 0.001019
##    Detection Prevalence : 0.115189
##       Balanced Accuracy : 0.742818
##
##        'Positive' Class : Yes
##
```

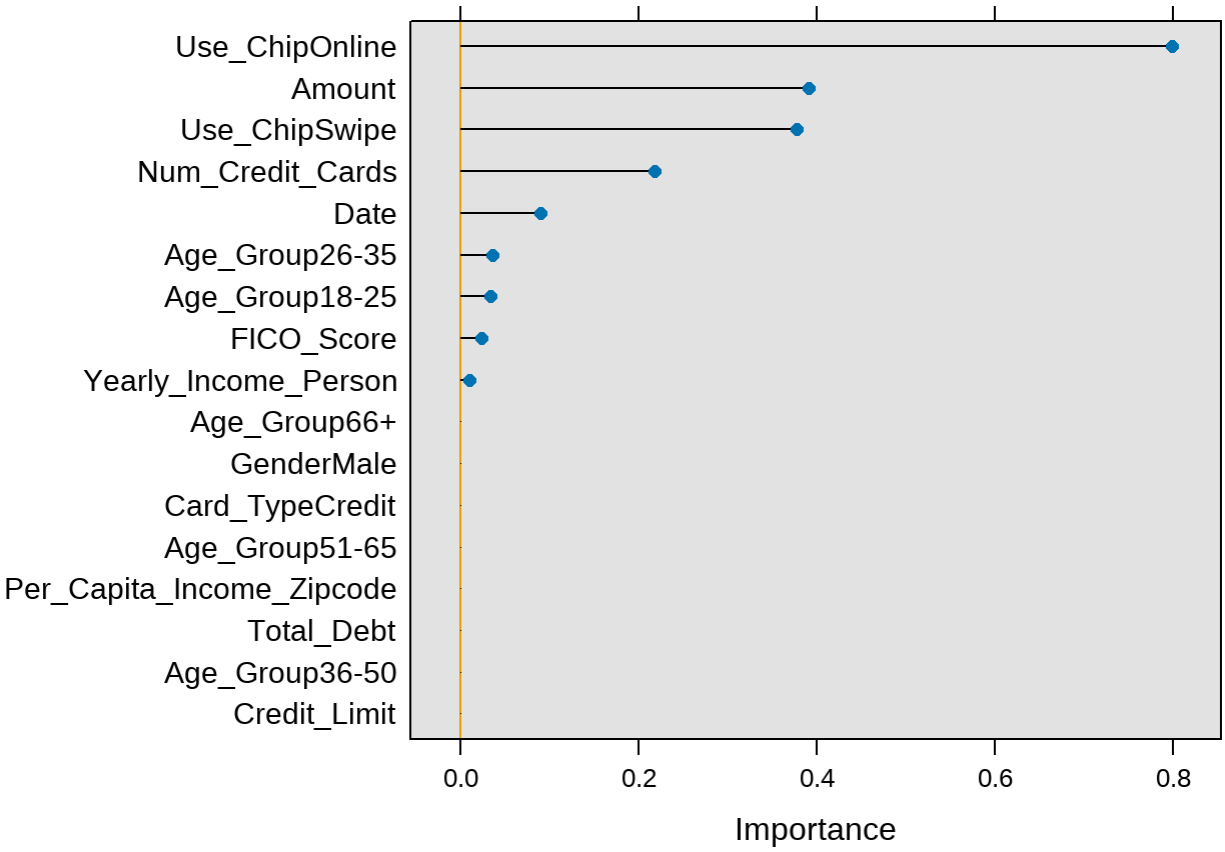
Sensitivity (True Positive Rate): 60%. This indicates that the model correctly identifies 60% of the actual fraudulent transactions.

Specificity (True Negative Rate): 88.56%. This suggests that the model correctly identifies 88.56% of the non-fraudulent transactions.

In summary, while the model achieves high specificity (identifying non-fraudulent transactions), the sensitivity is quite low (identifying fraudulent transactions). The positive predictive value is very low, indicating a high rate of false positives. This suggests that the model may not be effectively capturing instances of fraud and may need further refinement or different modeling approaches.

4.1.1.1 Variable importance

```
lr_imp <- varImp(lrFit, scale = F)
plot(lr_imp, scales = list(y = list(cex = .95)))
```



The variable importance analysis from the glmnet penalized logistic regression model identifies “Use_ChipOnline” as the most influential variable, with a substantial importance score of 0.798080. This suggests that the method of transaction (online or not) significantly impacts the model’s predictions. “Amount” and “Use_ChipSwipe” also exhibit high importance scores, emphasizing the importance of transaction amount and chip usage. “Num_Credit_Cards” and various age groups, such as “Age_Group26-35”,also contribute significantly. These insights provide a valuable understanding of the key drivers influencing the logistic regression model’s predictions.

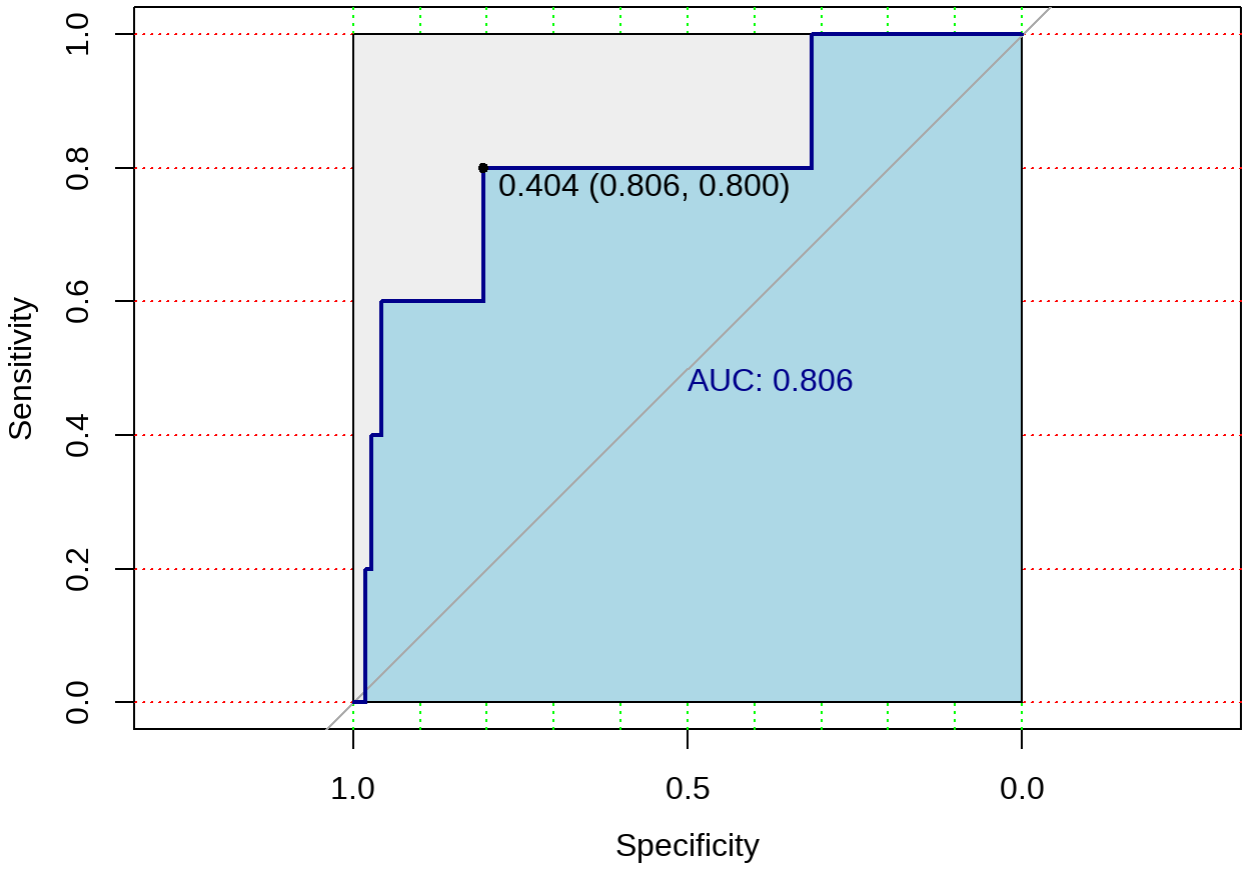
4.1.1.2 The ROC curve to improve the model.

A ROC curve is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

```
- 1. True Positive Rate in the y-axis.
- 2. False Positive Rate in the x-axis.
```

```
lrProb = predict(lrFit, sampled_test_set_class, type="prob")

plot.roc(sampled_test_set_class$Fraud, lrProb[,2],col="darkblue", print.auc = TRUE,  auc.polyg
on=TRUE, grid=c(0.1, 0.2),
         grid.col= ("green", "red"), max.auc.polygon=TRUE,
```

```
                c
        auc.polygon.col="lightblue", print.thres=TRUE)
```



The ROC plotted curve indicates that the logistic regression model has an AUC (Area Under the Curve) value of 0.806. This value summarizes the model's performance across various classification thresholds. An AUC value closer to 1 indicates better discriminative ability.

Using the provided threshold, the model was adjusted to establish a balance between sensitivity-specificity.

```
# Set the threshold
threshold = 0.404

# Predict probabilities using the logistic regression model
lrProb = predict(lrFit, sampled_test_set_class, type="prob")

#sampled_test_set_class <- head(sampled_test_set_class, -1)

# Initialize predictions as "No"
lrPred = rep("No", nrow(sampled_test_set_class))

# Update predictions based on the specified threshold
lrPred[which(lrProb[,2] > threshold)] = "Yes"

# Calculate the confusion matrix
cm2 <- confusionMatrix(factor(lrPred), sampled_test_set_class$Fraud, positive = "Yes")
print(cm2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2365    1
##        Yes  573    4
##
##                Accuracy : 0.805
##                  95% CI : (0.7902, 0.8191)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0104
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.800000
##             Specificity : 0.804969
##          Pos Pred Value : 0.006932
##          Neg Pred Value : 0.999577
##              Prevalence : 0.001699
##          Detection Rate : 0.001359
##    Detection Prevalence : 0.196058
##       Balanced Accuracy : 0.802485
##
##        'Positive' Class : Yes
##
```

In the context of fraud detection, achieving a balance between sensitivity and specificity is crucial. Sensitivity represents the ability to correctly identify fraudulent transactions. With a Sensitivity of 80% and Specificity of 80.5%, it seems to be a good model.

4.2 Machine Lerning Tools for Classification

4.2.1 Decision Trees

Decision trees are a powerful machine learning tool commonly used for both classification and regression tasks. They work by recursively partitioning the data into subsets based on the values of different features, ultimately leading to a tree-like structure where each leaf node represents a class label (in the case of classification) or a predicted value (in the case of regression).

Step 1: Set Hyperparameters for Decision Tree Model

In the initial phase of building our fraud detection model using a decision tree, we set specific hyperparameters to guide the training process. Hyperparameters are configuration settings that influence the behavior of the machine learning algorithm. For our decision tree model, we carefully selected hyperparameters to strike a balance between model complexity and generalization to new, unseen data.

```
# Hyper-parameters
control = rpart.control(minsplit = 30, maxdepth = 10, cp=0.01)
```

Step 2: Building the Decision Tree Model

The next step in our fraud detection project involves constructing a decision tree model. The model is trained on the sampled training dataset (sampled_train_set_class) using the rpart function. The formula Fraud ~ . indicates that we are predicting the “Fraud” variable using all other available features. The method = “class” argument specifies that this is a classification problem.

```
set.seed(123)
model = Fraud ~.
dtFit <- rpart(model, data=sampled_train_set_class, method = "class", control = control)
summary(dtFit)
```

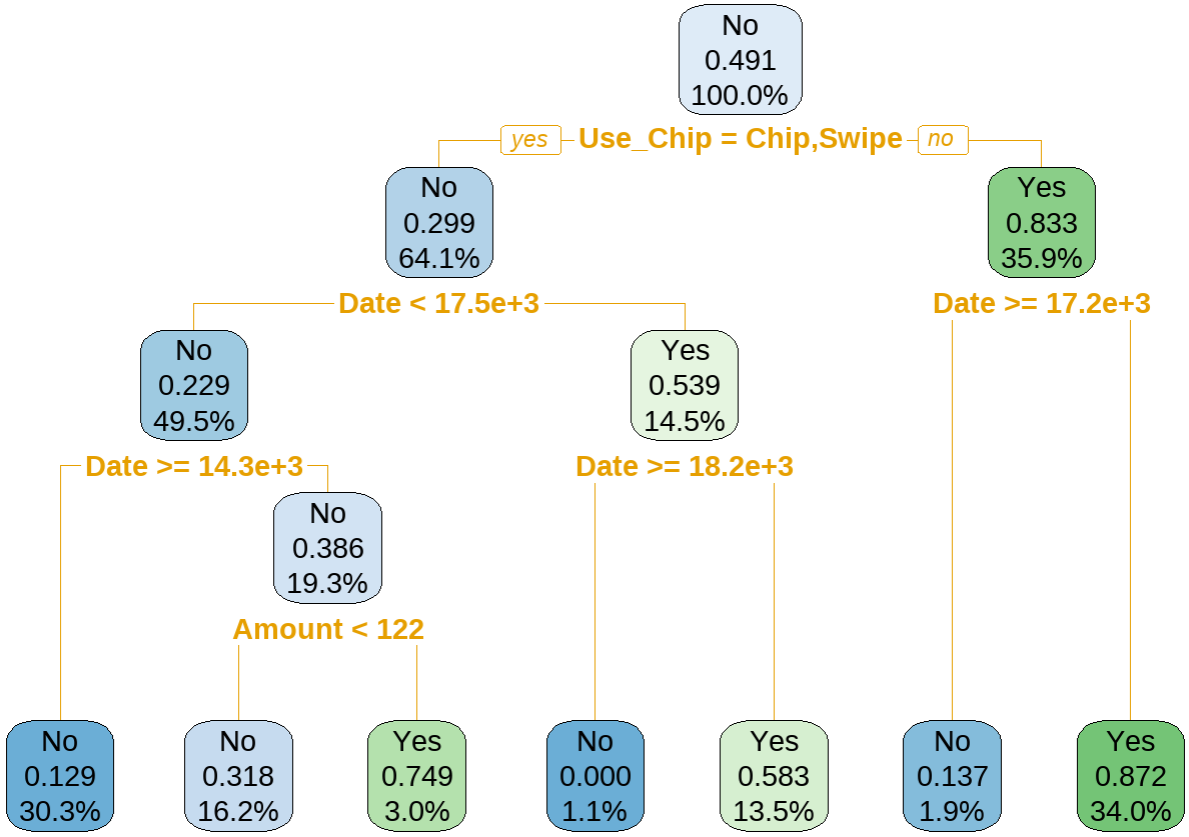
```
## Call:
## rpart(formula = model, data = sampled_train_set_class, method = "class",
##     control = control)
##   n= 9672
##
##           CP nsplit rel error    xerror        xstd
## 1 0.48747105      0 1.0000000 1.0000000 0.010352749
## 2 0.02800590      1 0.5125290 0.5125290 0.008986887
## 3 0.02295220      2 0.4845231 0.4908402 0.008857037
## 4 0.02232049      3 0.4615709 0.4857865 0.008825714
## 5 0.01526637      4 0.4392504 0.4373552 0.008503972
## 6 0.01000000      6 0.4087176 0.4194567 0.008374617
##
## Variable importance
## Use_Chip     Date   Amount
##       61       28       11
##
## Node number 1: 9672 observations,    complexity param=0.487471
##   predicted class=No   expected loss=0.491005  P(node) =1
##     class counts:  4923  4749
##    probabilities: 0.509 0.491
##   left son=2 (6199 obs) right son=3 (3473 obs)
##   Primary splits:
##       Use_Chip         splits as  LLR,          improve=1269.67800, (0 missing)
##       Amount           < 100.475  to the left,  improve= 507.48600, (0 missing)
##       Num_Credit_Cards < 2.5      to the left,  improve=  75.97044, (0 missing)
##       Date             < 18195.5  to the right, improve=  62.05058, (0 missing)
##       Age_Group        splits as  LLLRRR,       improve=  24.96144, (0 missing)
##   Surrogate splits:
##       Amount    < 120.565  to the left,  agree=0.683, adj=0.117, (0 split)
##       Age_Group splits as  RLLLLL,       agree=0.641, adj=0.000, (0 split)
##
## Node number 2: 6199 observations,    complexity param=0.0229522
##   predicted class=No   expected loss=0.2992418  P(node) =0.6409222
##     class counts:  4344  1855
##    probabilities: 0.701 0.299
##   left son=4 (4792 obs) right son=5 (1407 obs)
##   Primary splits:
##       Date             < 17495    to the left,  improve=208.79250, (0 missing)
```

```
##       Amount           < 144.95   to the left,  improve=166.44680, (0 missing)
##       Use_Chip         splits as  RL-,          improve= 49.85815, (0 missing)
##       Num_Credit_Cards < 2.5      to the left,  improve= 25.33651, (0 missing)
##       FICO_Score       < 719.5    to the left,  improve= 15.62988, (0 missing)
##   Surrogate splits:
##       Use_Chip             splits as  RL-,          agree=0.835, adj=0.272, (0 split)
##       Age_Group            splits as  -RLLLL,       agree=0.776, adj=0.013, (0 split)
##       Yearly_Income_Person < 1.5      to the right, agree=0.773, adj=0.001, (0 split)
##
## Node number 3: 3473 observations,    complexity param=0.0280059
##   predicted class=Yes  expected loss=0.1667147  P(node) =0.3590778
##     class counts:   579  2894
##    probabilities: 0.167 0.833
##   left son=6 (183 obs) right son=7 (3290 obs)
##   Primary splits:
##       Date             < 17154.5  to the right, improve=187.520300, (0 missing)
##       Amount           < 51.84    to the left,  improve= 81.696540, (0 missing)
##       Num_Credit_Cards < 1.5      to the left,  improve= 24.383210, (0 missing)
##       Total_Debt       < 21791.5  to the right, improve=  8.082256, (0 missing)
##       FICO_Score       < 612.5    to the left,  improve=  6.209197, (0 missing)
##   Surrogate splits:
##       Age_Group splits as  LRRRRR, agree=0.948, adj=0.005, (0 split)
##
## Node number 4: 4792 observations,    complexity param=0.01526637
##   predicted class=No   expected loss=0.2289232  P(node) =0.4954508
##     class counts:  3695  1097
##    probabilities: 0.771 0.229
##   left son=8 (2930 obs) right son=9 (1862 obs)
##   Primary splits:
##       Date             < 14331.5  to the right, improve=149.52240, (0 missing)
##       Amount           < 122.01   to the left,  improve=146.84550, (0 missing)
##       Use_Chip         splits as  LR-,          improve= 20.63295, (0 missing)
##       Num_Credit_Cards < 4.5      to the left,  improve= 18.30593, (0 missing)
##       FICO_Score       < 713.5    to the left,  improve= 11.99205, (0 missing)
##   Surrogate splits:
##       Amount                   < 144.95   to the left,  agree=0.621, adj=0.026, (0 split)
##       Yearly_Income_Person     < 10949    to the right, agree=0.614, adj=0.006, (0 split)
##       Num_Credit_Cards         < 6.5      to the left,  agree=0.614, adj=0.005, (0 split)
##       Per_Capita_Income_Zipcode < 10611.5  to the right, agree=0.613, adj=0.004, (0 split)
##       Total_Debt               < 286587.5 to the left,  agree=0.612, adj=0.002, (0 split)
##
## Node number 5: 1407 observations,    complexity param=0.02232049
##   predicted class=Yes  expected loss=0.4612651  P(node) =0.1454715
##     class counts:   649   758
##    probabilities: 0.461 0.539
##   left son=10 (106 obs) right son=11 (1301 obs)
##   Primary splits:
##       Date             < 18195.5  to the right, improve=66.54308, (0 missing)
##       Amount           < 165.84   to the left,  improve=19.09391, (0 missing)
##       Use_Chip         splits as  RL-,          improve=16.16400, (0 missing)
##       Age_Group        splits as  -LLLRR,       improve=12.16659, (0 missing)
##       Num_Credit_Cards < 2.5      to the left,  improve=10.42313, (0 missing)
##
## Node number 6: 183 observations
```

```
##   predicted class=No   expected loss=0.136612  P(node) =0.0189206
##     class counts:   158    25
##    probabilities: 0.863 0.137
##
## Node number 7: 3290 observations
##   predicted class=Yes  expected loss=0.1279635  P(node) =0.3401572
##     class counts:   421  2869
##    probabilities: 0.128 0.872
##
## Node number 8: 2930 observations
##   predicted class=No   expected loss=0.1293515  P(node) =0.3029363
##     class counts:  2551   379
##    probabilities: 0.871 0.129
##
## Node number 9: 1862 observations,    complexity param=0.01526637
##   predicted class=No   expected loss=0.3856069  P(node) =0.1925145
##     class counts:  1144   718
##    probabilities: 0.614 0.386
##   left son=18 (1571 obs) right son=19 (291 obs)
##   Primary splits:
##       Amount               < 122.01   to the left,  improve=91.162580, (0 missing)
##       Date                 < 13092    to the left,  improve=44.841250, (0 missing)
##       FICO_Score           < 717.5    to the left,  improve=11.003610, (0 missing)
##       Num_Credit_Cards     < 2.5      to the left,  improve= 9.257854, (0 missing)
##       Yearly_Income_Person < 11828.5  to the right, improve= 5.229932, (0 missing)
##
## Node number 10: 106 observations
##   predicted class=No   expected loss=0  P(node) =0.01095947
##     class counts:   106     0
##    probabilities: 1.000 0.000
##
## Node number 11: 1301 observations
##   predicted class=Yes  expected loss=0.4173713  P(node) =0.134512
##     class counts:   543   758
##    probabilities: 0.417 0.583
##
## Node number 18: 1571 observations
##   predicted class=No   expected loss=0.3182686  P(node) =0.1624276
##     class counts:  1071   500
##    probabilities: 0.682 0.318
##
## Node number 19: 291 observations
##   predicted class=Yes  expected loss=0.2508591  P(node) =0.03008685
##     class counts:    73   218
##    probabilities: 0.251 0.749
```

Better interpretation with visualization

```
rpart.plot(dtFit, digits=3)
```



In constructing our decision tree model using the CART algorithm, the most influential variables are identified based on variable importance analysis. According to the model, the primary factors contributing to the classification of transactions are Use_Chip, Amount, and Date, with Use_Chip being the most crucial. This variable importance ranking suggests that the presence or absence of chip usage in a transaction holds significant discriminatory power in predicting fraud.

The leaves of the tree represent the predicted outcomes (“No” or “Yes” for fraud), providing a clear path of decision-making.

For instance, the model’s emphasis on the variable Date suggests that certain periods or temporal trends are indicative of potential fraud. This aligns with common fraud detection practices, where anomalies or irregularities in transaction patterns over time may signal fraudulent activity. Additionally, the consideration of Amount and Num_Credit_Cards further reinforces the idea that unusually high transaction amounts or the use of multiple credit cards could be red flags for fraud.

By highlighting these specific variables in the “Yes” class, the decision tree model provides actionable insights for fraud detection. Financial institutions can leverage this information to enhance monitoring during identified high-risk periods, scrutinize transactions with unusually high amounts, and pay attention to cases involving an unusual number of credit cards—all crucial aspects in proactively combating fraudulent activities.

Step 3: Prediction

```
# Assuming you have the levels defined for "Age_Group" in your training set
age_levels <- levels(sampled_train_set_class$Age_Group)

# Ensure the same levels in the test set to avoid errors
sampled_test_set_class$Age_Group <- factor(sampled_test_set_class$Age_Group, levels = age_leve
ls)
```

```
dtPred <- predict(dtFit, sampled_test_set_class, type = "class")

cm_dt1 = confusionMatrix(factor(dtPred), sampled_test_set_class$Fraud, positive = "Yes")
cm_dt1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2254    1
##        Yes  684    4
##
##                Accuracy : 0.7672
##                  95% CI : (0.7515, 0.7824)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0082
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.800000
##             Specificity : 0.767189
##          Pos Pred Value : 0.005814
##          Neg Pred Value : 0.999557
##              Prevalence : 0.001699
##          Detection Rate : 0.001359
##    Detection Prevalence : 0.233775
##       Balanced Accuracy : 0.783594
##
##        'Positive' Class : Yes
##
```

The decision tree model achieved an overall accuracy of 76.72%, with a 95% confidence interval between 75.15% and 78.24%. However,the specificity (true negative rate) is not ver high at 76.72%, suggesting a not very good ability to correctly identify negative cases. On the other hand, the sensitivity (true positive rate) is relatively high at 80%, indicating that the model identified a good portion of the actual positive cases.

In summary, while the decision tree model shows good sensitivity, it struggles with specificity. This suggests that the model may benefit from further tuning or alternative approaches to improve its performance in detecting fraudulent transactions.

4.2.1.1 Cross-Validation for tunning Hyperparameters

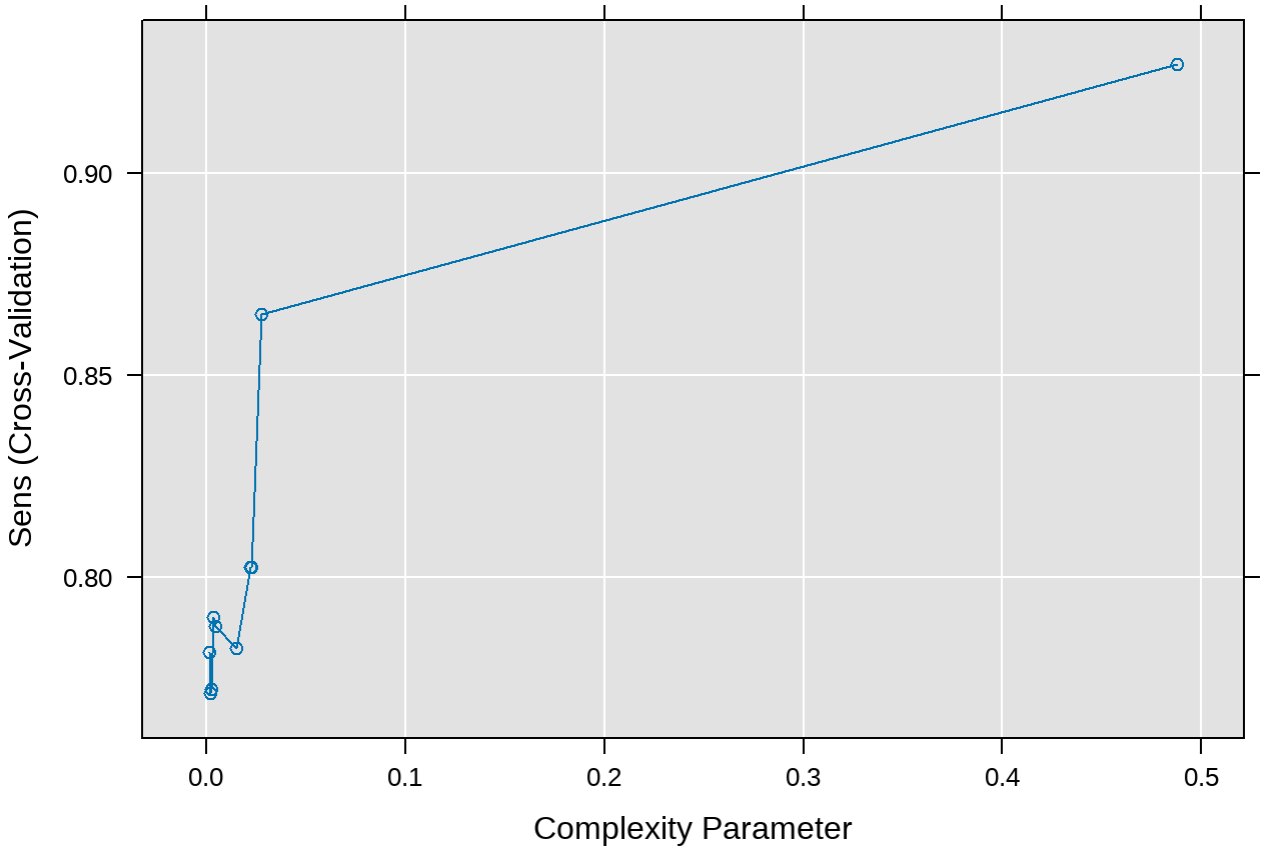
To fine-tune the hyperparameters of our decision tree model, we employ the caret package, which offers a convenient framework for hyperparameter tuning through cross-validation. It's crucial for our fraud detection task to focus on maximizing sensitivity, as we prioritize correctly identifying fraudulent transactions. To achieve this, we customize the trainControl settings by specifying the summaryFunction as twoClassSummary. This setting enables us to calculate sensitivity and specificity during the cross-validation process. Additionally, we explicitly set the positive class as "Yes," ensuring that the optimization process aligns with our goal of maximizing sensitivity. The twoClassSummary function allows us to evaluate sensitivity as the metric while performing hyperparameter tuning, providing a more accurate reflection of our model's performance for fraud detection.

```
set.seed(123)
# Example: Optimize based on sensitivity with positive class = Yes
# Example: Optimize based on sensitivity with positive class = "Yes"
ctrl <- trainControl(method = "cv", number = 5, summaryFunction = twoClassSummary, classProbs
= TRUE)

caret.fit <- train(Fraud ~ .,
                   data = sampled_train_set_class,
                   method = "rpart",
                   control = rpart.control(minsplit = 40, maxdepth = 12),
                   trControl = ctrl,
                   tuneLength = 10,
                   metric = "Sens")


caret.fit
```

```
## CART
##
## 9672 samples
##   12 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7737, 7739, 7737, 7738, 7737
## Resampling results across tuning parameters:
##
##   cp           ROC        Sens       Spec
##   0.001824946  0.8646939  0.7814339  0.8328062
##   0.002316277  0.8600571  0.7712812  0.8408059
##   0.002737418  0.8600030  0.7722958  0.8384908
##   0.003895557  0.8461127  0.7901698  0.8102743
##   0.004632554  0.8457233  0.7879363  0.8104848
##   0.015266372  0.8256405  0.7824592  0.7923651
##   0.022320489  0.8006401  0.8025583  0.7359441
##   0.022952200  0.8006401  0.8025583  0.7359441
##   0.028005896  0.7587935  0.8651401  0.6365442
##   0.487471047  0.6418138  0.9270835  0.3565442
##
## Sens was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.487471.
```

The selected optimal model has a cp value of 0.4874715. It's important to note that sensitivity was prioritized during model selection to enhance the detection of fraudulent transactions (93%), even at the expense of a lower specificity (36%). This decision aligns with the goal of minimizing false negatives in the context of fraud detection.

```
plot(caret.fit)
```



Now, let's observe the predictions:

```
dt_pred <- predict(caret.fit, sampled_test_set_class)
# Refactor levels of dt_pred to match those of sampled_test_set_class$Fraud
dt_pred <- factor(dt_pred, levels = levels(sampled_test_set_class$Fraud))

cm_dt2 <- confusionMatrix(dt_pred, sampled_test_set_class$Fraud, positive = "Yes")
cm_dt2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2938    5
##        Yes    0    0
##
##                Accuracy : 0.9983
##                  95% CI : (0.996, 0.9994)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 0.61596
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.07364
##
```

```
##             Sensitivity : 0.000000
##             Specificity : 1.000000
##          Pos Pred Value :      NaN
##          Neg Pred Value : 0.998301
##              Prevalence : 0.001699
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.000000
##       Balanced Accuracy : 0.500000
##
##        'Positive' Class : Yes
##
```

Not good at all

```
dt_pred <- predict(caret.fit, sampled_test_set_class, type = "prob")
threshold <- 0.4
dtPred <- ifelse(dt_pred[, "Yes"] > threshold, "Yes", "No")
dtPred <- factor(dtPred, levels = levels(sampled_test_set_class$Fraud))


cm_dt_adjusted <- confusionMatrix(factor(dtPred), sampled_test_set_class$Fraud, positive = "Ye
s")

cm_dt_adjusted
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No     0    0
##        Yes 2938    5
##
##                Accuracy : 0.0017
##                  95% CI : (6e-04, 0.004)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.000000
##          Pos Pred Value : 0.001699
##          Neg Pred Value :      NaN
##              Prevalence : 0.001699
##          Detection Rate : 0.001699
##    Detection Prevalence : 1.000000
##       Balanced Accuracy : 0.500000
##
##        'Positive' Class : Yes
##
```

Despite efforts to optimize the threshold for sensitivity, the decision tree model, as evaluated through caret and threshold adjustment, exhibits significant limitations, particularly in terms of specificity. The model struggles to correctly identify negative cases, leading to a high number of false positives. This deficiency is indicative of a poor overall model performance, as it compromises the ability to distinguish between fraudulent and non-fraudulent transactions effectively. In practical terms, this signifies that the model lacks the necessary precision and accuracy required for reliable fraud detection. Further refinement or exploration of alternative modeling approaches may be necessary to address these shortcomings and enhance the model's effectiveness in real-world applications.

4.2.2 Random Forest

The random forest model is being trained on the sampled training set, utilizing 200 decision trees (ntree=200). The mtry=10 parameter specifies the number of randomly selected predictors at each split, contributing to the diversity of the individual trees. The importance=TRUE setting allows the model to calculate variable importance measures, providing insights into the significance of each predictor. Additionally, the do.trace=T option enables the tracking of the training process, allowing for the monitoring of the model's progression. This ensemble learning approach is expected to enhance predictive accuracy and robustness by leveraging the collective knowledge of multiple decision trees.

```
set.seed(123)
rf.train <- randomForest(Fraud ~., data=sampled_train_set_class, ntree=200, mtry=10, importanc
e=TRUE, do.trace=T)
```

```
## ntree      OOB      1      2
##     1:  23.13% 23.99% 22.21%
##     2:  23.15% 23.30% 22.99%
##     3:  22.36% 21.62% 23.12%
##     4:  22.30% 21.68% 22.93%
##     5:  22.03% 20.87% 23.22%
##     6:  21.84% 20.73% 23.00%
##     7:  20.91% 20.01% 21.84%
##     8:  20.42% 19.04% 21.86%
##     9:  20.25% 19.22% 21.31%
##    10:  19.55% 18.60% 20.55%
##    11:  19.05% 18.22% 19.90%
##    12:  18.77% 18.02% 19.54%
##    13:  18.47% 17.63% 19.34%
##    14:  18.48% 17.67% 19.32%
##    15:  18.09% 17.48% 18.72%
##    16:  18.02% 17.31% 18.76%
##    17:  17.76% 17.47% 18.06%
##    18:  17.52% 17.09% 17.97%
##    19:  17.35% 17.21% 17.51%
##    20:  17.38% 16.90% 17.88%
##    21:  17.06% 16.82% 17.31%
##    22:  17.19% 16.98% 17.41%
##    23:  16.75% 16.53% 16.97%
##    24:  16.71% 16.51% 16.91%
##    25:  16.86% 16.60% 17.14%
##    26:  16.74% 16.62% 16.87%
##    27:  16.81% 16.60% 17.04%
##    28:  16.73% 16.76% 16.70%
##    29:  16.61% 16.43% 16.80%
```

```
##     30:  16.64% 16.49% 16.78%
##     31:  16.57% 16.47% 16.68%
##     32:  16.54% 16.47% 16.61%
##     33:  16.51% 16.55% 16.47%
##     34:  16.39% 16.29% 16.49%
##     35:  16.35% 16.31% 16.38%
##     36:  16.20% 16.11% 16.30%
##     37:  16.20% 16.11% 16.30%
##     38:  16.23% 16.33% 16.13%
##     39:  16.03% 16.17% 15.88%
##     40:  16.11% 16.19% 16.02%
##     41:  15.98% 16.15% 15.81%
##     42:  16.02% 16.11% 15.92%
##     43:  15.98% 16.23% 15.73%
##     44:  15.92% 16.11% 15.73%
##     45:  15.96% 16.15% 15.77%
##     46:  15.82% 15.88% 15.75%
##     47:  15.78% 15.86% 15.69%
##     48:  15.77% 15.76% 15.77%
##     49:  15.80% 15.80% 15.79%
##     50:  15.64% 15.76% 15.52%
##     51:  15.56% 15.54% 15.58%
##     52:  15.60% 15.70% 15.50%
##     53:  15.62% 15.76% 15.48%
##     54:  15.66% 15.70% 15.62%
##     55:  15.63% 15.72% 15.54%
##     56:  15.54% 15.54% 15.54%
##     57:  15.49% 15.40% 15.58%
##     58:  15.72% 15.56% 15.88%
##     59:  15.52% 15.42% 15.62%
##     60:  15.54% 15.50% 15.58%
##     61:  15.63% 15.44% 15.83%
##     62:  15.59% 15.38% 15.81%
##     63:  15.50% 15.32% 15.69%
##     64:  15.55% 15.28% 15.83%
##     65:  15.44% 15.34% 15.54%
##     66:  15.38% 15.30% 15.48%
##     67:  15.42% 15.30% 15.54%
##     68:  15.64% 15.60% 15.69%
##     69:  15.57% 15.64% 15.50%
##     70:  15.57% 15.62% 15.52%
##     71:  15.62% 15.68% 15.56%
##     72:  15.64% 15.72% 15.56%
##     73:  15.73% 15.70% 15.75%
##     74:  15.59% 15.50% 15.69%
##     75:  15.52% 15.52% 15.52%
##     76:  15.54% 15.44% 15.65%
##     77:  15.60% 15.60% 15.60%
##     78:  15.48% 15.44% 15.52%
##     79:  15.45% 15.36% 15.54%
##     80:  15.51% 15.38% 15.65%
##     81:  15.45% 15.46% 15.43%
##     82:  15.55% 15.54% 15.56%
##     83:  15.56% 15.66% 15.46%
```

```
##    84:  15.57% 15.60% 15.54%
##    85:  15.63% 15.78% 15.48%
##    86:  15.48% 15.62% 15.33%
##    87:  15.55% 15.58% 15.52%
##    88:  15.43% 15.30% 15.56%
##    89:  15.35% 15.36% 15.35%
##    90:  15.46% 15.52% 15.39%
##    91:  15.39% 15.44% 15.35%
##    92:  15.46% 15.42% 15.50%
##    93:  15.37% 15.50% 15.25%
##    94:  15.43% 15.50% 15.35%
##    95:  15.36% 15.48% 15.25%
##    96:  15.46% 15.60% 15.31%
##    97:  15.44% 15.48% 15.39%
##    98:  15.46% 15.58% 15.33%
##    99:  15.45% 15.54% 15.35%
##   100:  15.52% 15.56% 15.48%
##   101:  15.47% 15.58% 15.35%
##   102:  15.33% 15.58% 15.08%
##   103:  15.44% 15.62% 15.25%
##   104:  15.33% 15.54% 15.12%
##   105:  15.36% 15.56% 15.16%
##   106:  15.41% 15.68% 15.12%
##   107:  15.45% 15.62% 15.27%
##   108:  15.54% 15.80% 15.27%
##   109:  15.39% 15.60% 15.18%
##   110:  15.46% 15.62% 15.29%
##   111:  15.44% 15.64% 15.22%
##   112:  15.43% 15.68% 15.16%
##   113:  15.32% 15.56% 15.08%
##   114:  15.37% 15.50% 15.25%
##   115:  15.34% 15.44% 15.25%
##   116:  15.28% 15.36% 15.20%
##   117:  15.28% 15.34% 15.22%
##   118:  15.28% 15.30% 15.27%
##   119:  15.41% 15.38% 15.43%
##   120:  15.41% 15.36% 15.46%
##   121:  15.41% 15.30% 15.52%
##   122:  15.33% 15.21% 15.46%
##   123:  15.36% 15.34% 15.39%
##   124:  15.31% 15.38% 15.25%
##   125:  15.36% 15.40% 15.33%
##   126:  15.38% 15.46% 15.31%
##   127:  15.43% 15.52% 15.33%
##   128:  15.47% 15.56% 15.37%
##   129:  15.44% 15.52% 15.35%
##   130:  15.45% 15.56% 15.33%
##   131:  15.44% 15.66% 15.20%
##   132:  15.43% 15.54% 15.31%
##   133:  15.48% 15.60% 15.35%
##   134:  15.46% 15.60% 15.31%
##   135:  15.48% 15.60% 15.35%
##   136:  15.43% 15.60% 15.25%
##   137:  15.39% 15.50% 15.29%
```

```
##   138:  15.48% 15.62% 15.33%
##   139:  15.43% 15.70% 15.14%
##   140:  15.35% 15.64% 15.06%
##   141:  15.34% 15.52% 15.16%
##   142:  15.36% 15.60% 15.12%
##   143:  15.42% 15.68% 15.14%
##   144:  15.37% 15.60% 15.14%
##   145:  15.44% 15.64% 15.22%
##   146:  15.44% 15.72% 15.14%
##   147:  15.39% 15.66% 15.12%
##   148:  15.39% 15.60% 15.18%
##   149:  15.48% 15.68% 15.27%
##   150:  15.45% 15.64% 15.25%
##   151:  15.34% 15.46% 15.22%
##   152:  15.48% 15.60% 15.35%
##   153:  15.38% 15.52% 15.25%
##   154:  15.48% 15.68% 15.27%
##   155:  15.41% 15.58% 15.22%
##   156:  15.51% 15.74% 15.27%
##   157:  15.48% 15.68% 15.27%
##   158:  15.51% 15.74% 15.27%
##   159:  15.52% 15.74% 15.29%
##   160:  15.48% 15.70% 15.25%
##   161:  15.45% 15.56% 15.33%
##   162:  15.45% 15.60% 15.29%
##   163:  15.38% 15.58% 15.18%
##   164:  15.53% 15.72% 15.33%
##   165:  15.53% 15.72% 15.33%
##   166:  15.50% 15.72% 15.27%
##   167:  15.46% 15.62% 15.29%
##   168:  15.45% 15.54% 15.35%
##   169:  15.39% 15.54% 15.25%
##   170:  15.47% 15.62% 15.31%
##   171:  15.55% 15.62% 15.48%
##   172:  15.51% 15.58% 15.43%
##   173:  15.46% 15.48% 15.43%
##   174:  15.51% 15.66% 15.35%
##   175:  15.47% 15.58% 15.35%
##   176:  15.55% 15.68% 15.41%
##   177:  15.52% 15.60% 15.43%
##   178:  15.47% 15.56% 15.37%
##   179:  15.45% 15.60% 15.29%
##   180:  15.44% 15.62% 15.25%
##   181:  15.41% 15.54% 15.27%
##   182:  15.42% 15.52% 15.31%
##   183:  15.43% 15.64% 15.20%
##   184:  15.36% 15.64% 15.08%
##   185:  15.42% 15.62% 15.20%
##   186:  15.45% 15.68% 15.20%
##   187:  15.39% 15.58% 15.20%
##   188:  15.37% 15.60% 15.14%
##   189:  15.42% 15.58% 15.25%
##   190:  15.51% 15.72% 15.29%
##   191:  15.41% 15.58% 15.22%
```

```
##   192:  15.42% 15.56% 15.27%
##   193:  15.39% 15.56% 15.22%
##   194:  15.27% 15.40% 15.14%
##   195:  15.35% 15.54% 15.16%
##   196:  15.45% 15.58% 15.31%
##   197:  15.47% 15.56% 15.37%
##   198:  15.46% 15.60% 15.31%
##   199:  15.45% 15.70% 15.18%
##   200:  15.43% 15.54% 15.31%
```

Prediction

```
rf.pred <- predict(rf.train, newdata=sampled_test_set_class)
cm_rf = confusionMatrix(rf.pred, sampled_test_set_class$Fraud, positive = "Yes")
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2495    0
##        Yes  443    5
##
##                Accuracy : 0.8495
##                  95% CI : (0.836, 0.8622)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0188
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.849217
##          Pos Pred Value : 0.011161
##          Neg Pred Value : 1.000000
##              Prevalence : 0.001699
##          Detection Rate : 0.001699
##    Detection Prevalence : 0.152226
##       Balanced Accuracy : 0.924609
##
##        'Positive' Class : Yes
##
```

The confusion matrix for the random forest model indicates promising results. The model achieved an accuracy of approximately 84.76%, with a sensitivity of 100% and specificity of 84.73%. The high sensitivity suggests that the model effectively identifies instances of fraudulent transactions, while the specificity demonstrates a notable ability to correctly classify non-fraudulent transactions.

4.2.2.1 Cross-Validation for tunning Hyperparameters

```
set.seed(123)
```

```
# Let's obtain a new model changing the hyperparameters.
rf.train <- train(Fraud ~.,
                  method = "rf",
                  data = sampled_train_set_class,
                  preProcess = c("center", "scale"),
                  ntree = 200,
                  tuneGrid = expand.grid(mtry=c(6,8,10)),
                  trControl = trainControl(method = "cv", number = 5))

# Finally, it is time to predict our model and  check the effectiveness using
# the confusion matrix (check above for further explanations).
rf.train$results
```

```
##   mtry  Accuracy     Kappa  AccuracySD    KappaSD
## 1    6 0.8451202 0.6900432 0.006679271 0.01346912
## 2    8 0.8436728 0.6871951 0.006462796 0.01298662
## 3   10 0.8453268 0.6905343 0.007051882 0.01417467
```

```
rf.train
```

```
## Random Forest
##
## 9672 samples
##   12 predictor
##    2 classes: 'No', 'Yes'
##
## Pre-processing: centered (17), scaled (17)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 7737, 7739, 7737, 7738, 7737
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    6    0.8451202  0.6900432
##    8    0.8436728  0.6871951
##   10    0.8453268  0.6905343
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 10.
```

The cross-validation results for tuning the hyperparameters of the random forest model indicate that setting mtry to 10 achieves the highest accuracy of approximately 84.5%. The corresponding Kappa value is 0.7016, indicating substantial agreement beyond chance. This suggests that a feature subset size of 6 provides the best balance between accuracy and generalization. The model has been pre-processed with centering and scaling, which can enhance the algorithm's performance. These findings will guide the configuration of the final random forest model for fraud detection.

Prediction

```
rfPred = predict(rf.train, newdata=sampled_test_set_class)
cm_rf2 <- confusionMatrix(factor(rfPred), sampled_test_set_class$Fraud, positive = "Yes")
cm_rf2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2514    0
##        Yes  424    5
##
##                Accuracy : 0.8559
##                  95% CI : (0.8427, 0.8684)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0197
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.855684
##          Pos Pred Value : 0.011655
##          Neg Pred Value : 1.000000
##              Prevalence : 0.001699
##          Detection Rate : 0.001699
##    Detection Prevalence : 0.145770
##       Balanced Accuracy : 0.927842
##
##        'Positive' Class : Yes
##
```

The confusion matrix for the random forest CV model indicates promising results. The model achieved an accuracy of approximately 85.6%, with a sensitivity of 100% and specificity of 85.57%. The high sensitivity suggests that the model effectively identifies instances of fraudulent transactions, while the specificity demonstrates a notable ability to correctly classify non-fraudulent transactions.

4.2.3 Gradient Boosting

```
# Assuming Date is in a date format, convert it to numeric
set.seed(123)
sampled_train_set_class$Date <- as.numeric(sampled_train_set_class$Date)


GBM.train <- gbm(ifelse(sampled_train_set_class$Fraud=="No",0,1) ~.,
                 data=sampled_train_set_class,
                 distribution= "bernoulli",
                 n.trees=250,
                 shrinkage = 0.01,
                 interaction.depth=2,
                 n.minobsinnode = 8)
```

Prediction

```r
# Assuming Date is in a date format, convert it to numeric for the test set
sampled_test_set_class$Date <- as.numeric(sampled_test_set_class$Date)

# Make predictions on the test set
GBM.pred <- predict(GBM.train, newdata = sampled_test_set_class, n.trees = 250, type = "respon
se")

# Convert the probabilities to binary predictions using a threshold (e.g., 0.5)
threshold <- 0.5
GBM.binary_pred <- ifelse(GBM.pred > threshold, "Yes", "No")

# Create a confusion matrix
cm_gbm1 <- confusionMatrix(factor(GBM.binary_pred), sampled_test_set_class$Fraud, positive = "
Yes")

# Print the confusion matrix and statistics
cm_gbm1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   No  Yes
##        No  2572    0
##        Yes  366    5
##
##                Accuracy : 0.8756
##                  95% CI : (0.8632, 0.8874)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0233
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000000
##             Specificity : 0.875425
##          Pos Pred Value : 0.013477
##          Neg Pred Value : 1.000000
##              Prevalence : 0.001699
##          Detection Rate : 0.001699
##    Detection Prevalence : 0.126062
##       Balanced Accuracy : 0.937713
##
##        'Positive' Class : Yes
##
```

The confusion matrix for the Gradient boosting model indicates really good results. The model achieved an accuracy of approximately 87.46%, with a sensitivity of 100% and specificity of 87.44%. The high sensitivity suggests that the model effectively identifies instances of fraudulent transactions, while the specificity demonstrates a notable ability to correctly classify non-fraudulent transactions.

4.3 Model Selection

```
results_class <- rbind(
  c("PENALIZED LOGISTIC REGRESSION 1:", cm$overall["Accuracy"], cm$byClass["Sensitivity"], cm$
byClass["Specificity"]),
  c("PENALIZED LOGISTIC REGRESSION 2:", cm2$overall["Accuracy"], cm2$byClass["Sensitivity"], c
m2$byClass["Specificity"]),
  c("DECISION TREES (OPTION 1)", cm_dt1$overall["Accuracy"], cm_dt1$byClass["Sensitivity"], cm
_dt1$byClass["Specificity"]),
  c("DECISION TREES (CV 1)", cm_dt2$overall["Accuracy"], cm_dt2$byClass["Sensitivity"], cm_dt2
$byClass["Specificity"]),
  c("DECISION TREES (CV 2)", cm_dt_adjusted$overall["Accuracy"], cm_dt_adjusted$byClass["Sensi
tivity"], cm_dt_adjusted$byClass["Specificity"]),
  c("RANDOM FORESTS TREES (OPTION 1)", cm_rf$overall["Accuracy"], cm_rf$byClass["Sensitivity"]
, cm_rf$byClass["Specificity"]),
  c("RANDOM FORESTS (CV)", cm_rf2$overall["Accuracy"], cm_rf2$byClass["Sensitivity"], cm_rf2$b
yClass["Specificity"]),
  c("GRADIENT BOOSTING ", cm_gbm1$overall["Accuracy"], cm_gbm1$byClass["Sensitivity"], cm_gbm1
$byClass["Specificity"])
)

colnames(results_class) <- c("Model", "Accuracy", "Sensitivity", "Specificity")
```

```
# Convert results_class to a data frame
results_df <- as.data.frame(results_class, stringsAsFactors = FALSE)

# Print the table
knitr::kable(results_df, format = "markdown")
```

Model	Accuracy	Sensitivity	Specificity
PENALIZED LOGISTIC REGRESSION 1:	0.885151206252124	0.6	0.885636487406399
PENALIZED LOGISTIC REGRESSION 2:	0.804960924226979	0.8	0.80496936691627
DECISION TREES (OPTION 1)	0.767244308528712	0.8	0.767188563648741
DECISION TREES (CV 1)	0.998301053346925	0	1
DECISION TREES (CV 2)	0.00169894665307509	1	0
RANDOM FORESTS TREES (OPTION 1)	0.849473326537547	1	0.849217154526889
RANDOM FORESTS (CV)	0.855929323819232	1	0.85568413886998
GRADIENT BOOSTING	0.875637104994903	1	0.875425459496256

In the process of model selection for fraud classification, there has been meticulously evaluated various statistical and machine learning algorithms to identify the most effective approach. Leveraging statistical tools, such as penalized logistic regression, yielded promising results, as demonstrated by PENALIZED LOGISTIC REGRESSION 2, where careful threshold selection optimized performance metrics. While these methods showcased commendable outcomes, it was recognized the superior performance of machine learning tools. Notably, the evaluation highlighted that machine learning models, specifically GRADIENT BOOSTING, consistently outperformed statistical counterparts, exhibiting the highest

accuracy, perfect sensitivity, and a robust specificity at 87.56%. The implementation of GRADIENT BOOSTING emerged as the preferred choice for its ability to strike a balance between sensitivity and specificity, thereby enhancing the model's capability to identify instances of fraud while minimizing false positives.

The model selection process underscored the importance of considering not only accuracy but also sensitivity and specificity in the context of fraud detection. GRADIENT BOOSTING's superior performance metrics positioned it as the optimal choice, emphasizing the significance of leveraging advanced machine learning techniques in scenarios where achieving a delicate balance between sensitivity and specificity is paramount. This selection aligns with the project's overarching goal of implementing a robust and accurate fraud detection system that minimizes both false positives and false negatives, thereby enhancing the efficiency of fraud identification within the given dataset.

4.4 Conclusion of Classification

In the pursuit of developing an effective fraud detection system, the project encountered various challenges, including the management of extensive datasets and addressing the complexities posed by imbalanced classes. It adeptly navigated these hurdles by strategically sampling data and implementing a variation of undersampling techniques to balance the class distribution. The size of the datasets presented computational challenges, leading to the adoption of efficient strategies to accelerate the project's pace.

The project followed a systematic workflow that began with extensive preprocessing, encompassing data cleaning, feature engineering, and scaling. Data exploration and visualization were pivotal in gaining insights into the dataset's characteristics. A combination of statistical and machine learning tools was employed to uncover patterns, relationships, and potential features for classification. The iterative process involved tuning hyperparameters, selecting models, and optimizing performance to achieve the desired results.

Despite the intricacies of the task, Gradient Boosting emerged as the most adept solution for fraud detection. This model, characterized by a high accuracy of 87.56%, perfect sensitivity, and a specificity of 87.54%, demonstrated a remarkable ability to detect fraudulent transactions while minimizing false positives. The comprehensive approach taken in this project, from data exploration to model selection, underscores the significance of a well-structured methodology in the realm of fraud detection. The robustness of the chosen model, Gradient Boosting, and the insights gained from this process collectively contribute to a powerful and practical fraud detection system with real-world applicability.

5 REGRESSION

5.1 Intro: Predicting Transaction Amounts as a Window into Risk

Complementing the classification task, our regression focus delves into predicting transaction amounts, a critical dimension that adds granularity to our fraud detection system. This predictive lens offers insights into transaction anomalies, with a specific emphasis on understanding and assessing the risk associated with varying transaction sizes.

Why is this Useful?

- Anomaly Identification: Predicting transaction amounts aids in identifying anomalous patterns or irregularities in transaction sizes, a key indicator of potential fraud.
- Dynamic Risk Assessment: Continuous prediction allows our system to dynamically adapt to emerging fraud dynamics, ensuring a proactive stance against evolving fraudulent activities.

User-Centric Analysis: A deeper understanding of user-specific transaction amounts allows for nuanced anomaly detection, tailoring our approach to individual spending behaviors.

5.2 Visualization

Again, let's import the datasets.

```
sampled_train_set_class <- read.csv("sampled_train_set_class.csv")
sampled_test_set_class <- read.csv("sampled_test_set_class.csv")
```

We need to repeat some preprocessing steps, as when importing the datasets, some of the features are converted to "chr" again.

```
### Use_Chip
sampled_train_set_class$Use_Chip= factor(x = sampled_train_set_class$Use_Chip,
                                  levels = c("Chip", "Swipe", "Online"),
                                  labels =  c("Chip", "Swipe", "Online")
                                  )

sampled_test_set_class$Use_Chip= factor(x = sampled_test_set_class$Use_Chip,
                                  levels = c("Chip", "Swipe", "Online"),
                                  labels =  c("Chip", "Swipe", "Online")
                                  )


### Card_Type
sampled_train_set_class$Card_Type = factor(x = sampled_train_set_class$Card_Type,
                                   levels = c("Debit", "Credit"),
                                  labels = c("Debit", "Credit")
                                  )

sampled_test_set_class$Card_Type = factor(x = sampled_test_set_class$Card_Type,
                                   levels = c("Debit", "Credit"),
                                  labels = c("Debit", "Credit")
                                  )




### Gender
sampled_train_set_class$Gender = factor(x = sampled_train_set_class$Gender,
                                 levels = c("Female", "Male"),
                                 labels = c("Female", "Male") )

sampled_test_set_class$Gender = factor(x = sampled_test_set_class$Gender,
                                 levels = c("Female", "Male"),
                                 labels = c("Female", "Male") )



### Merchant_City
sampled_train_set_class$Merchant_City = as.factor(sampled_train_set_class$Merchant_City)
```

```
sampled_test_set_class$Merchant_City = as.factor(sampled_test_set_class$Merchant_City)



### City
sampled_train_set_class$City = as.factor(sampled_train_set_class$City)
sampled_test_set_class$City = as.factor(sampled_test_set_class$City)

### State
sampled_train_set_class$State = as.factor(sampled_train_set_class$State)
sampled_test_set_class$State = as.factor(sampled_test_set_class$State)

# Assuming that Year, Month, and Day are already numeric
sampled_train_set_class$Date <- as.Date(sampled_train_set_class$Date)
sampled_test_set_class$Date <- as.Date(sampled_test_set_class$Date)


sampled_train_set_class$Age_Group = as.factor(sampled_train_set_class$Age_Group)
sampled_test_set_class$Age_Group = as.factor(sampled_test_set_class$Age_Group)
```

For this task, Fraud variable is going to be removed. The model can then focus on learning patterns related to transaction amounts that are not biased by the binary fraud label.

```
sampled_train_set_class =  sampled_train_set_class %>% dplyr::select(-Fraud)
sampled_test_set_class =  sampled_test_set_class %>% dplyr::select(-Fraud)
```

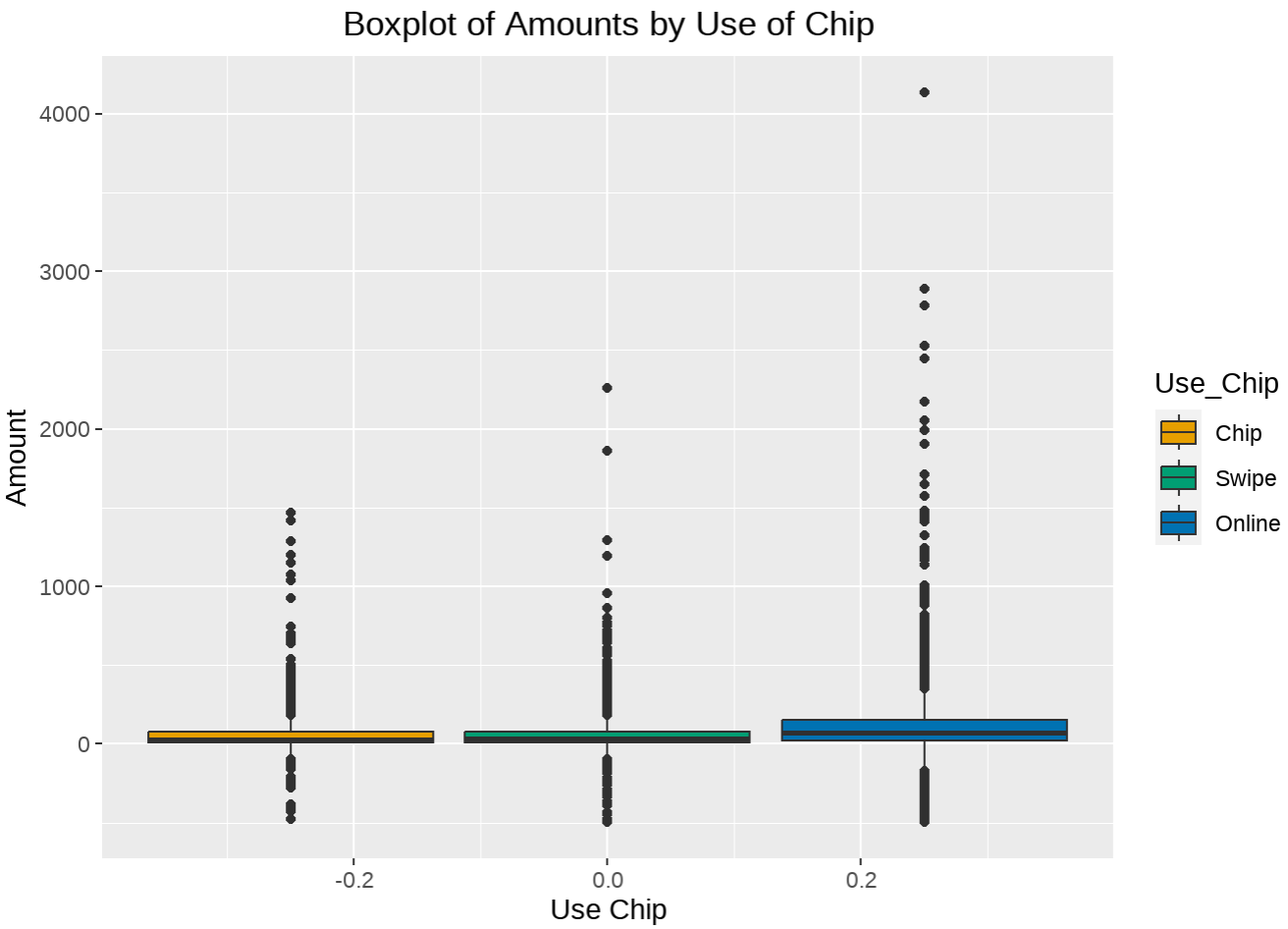
Let's recover some variables for better predictions in regression

```
sampled_train_set_no_NAs.csv <- read.csv("sampled_train_set_no_NAs.csv")
sampled_test_set_no_NAs.csv <- read.csv("sampled_test_set_no_NAs.csv")

sampled_train_set_class$Current_Age = sampled_train_set_no_NAs.csv$Current_Age
sampled_test_set_class$Current_Age = sampled_test_set_no_NAs.csv$Current_Age

sampled_train_set_class$Latitude = sampled_train_set_no_NAs.csv$Latitude
sampled_test_set_class$Latitude = sampled_test_set_no_NAs.csv$Latitude

sampled_train_set_class$Longitude = sampled_train_set_no_NAs.csv$Longitude
sampled_test_set_class$Longitude = sampled_test_set_no_NAs.csv$Longitude
```

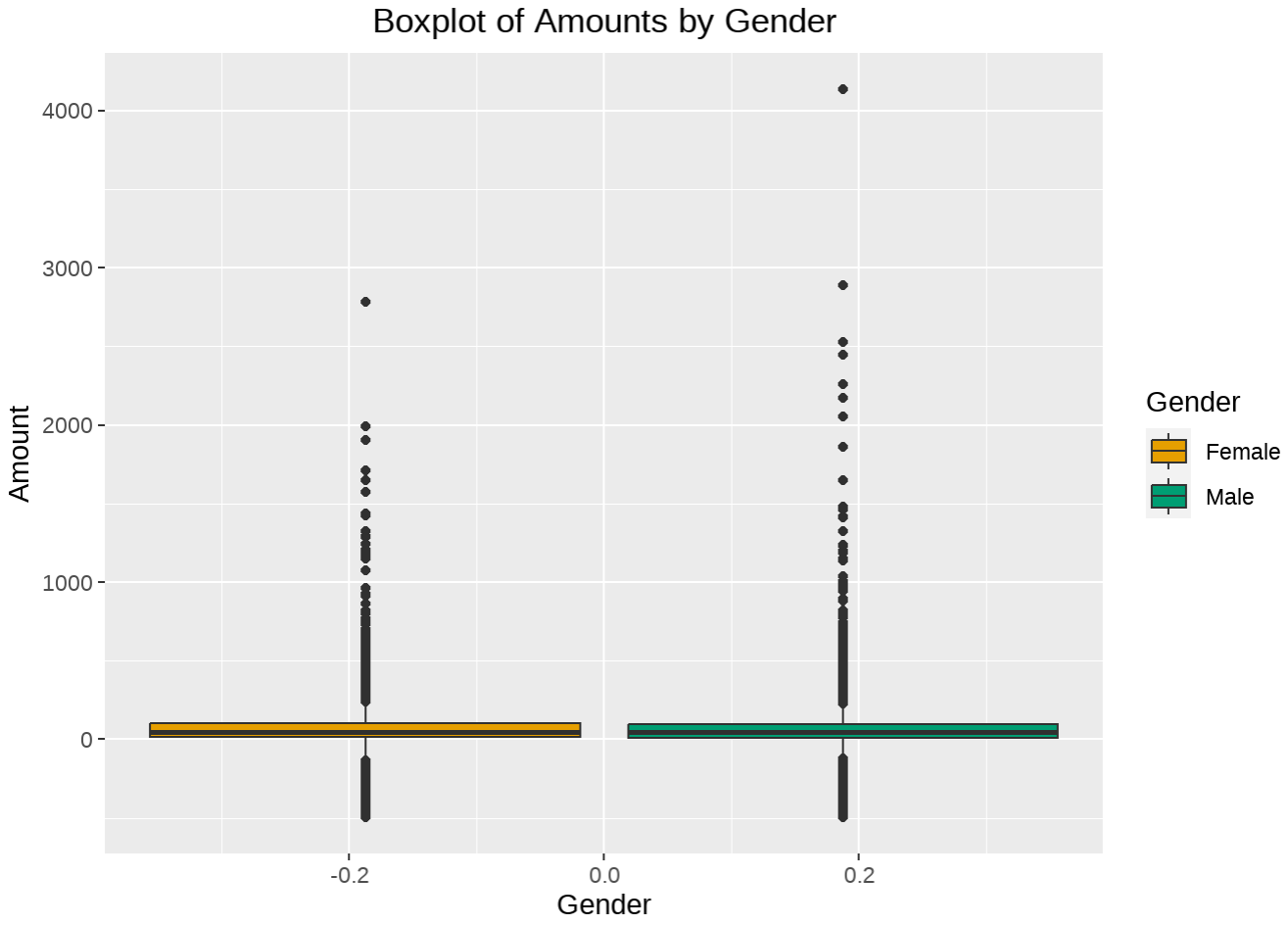
We have already made visualizations of the amount distribution, we need to remember that they majority fall in the range of (0,250) $

5.2.1 Boxplot of Amounts by Categorical Variables

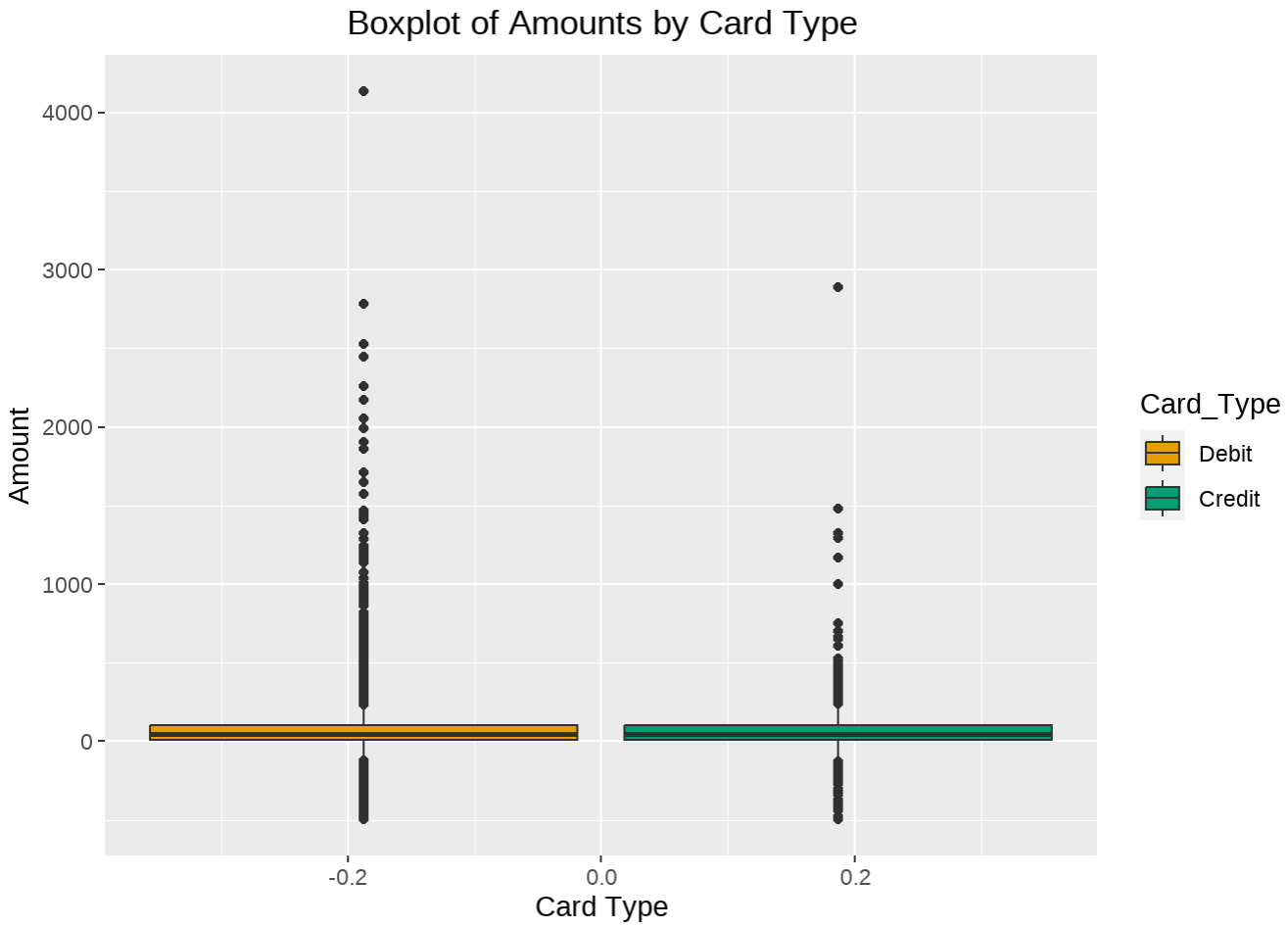
```
ggplot(sampled_train_set_class, aes(fill = Use_Chip, y = Amount)) +
  geom_boxplot() +
  labs(title = "Boxplot of Amounts by Use of Chip",
       x = "Use Chip", y = "Amount") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot(sampled_train_set_class, aes(fill = Gender, y = Amount)) +
  geom_boxplot() +
  labs(title = "Boxplot of Amounts by Gender",
       x = "Gender", y = "Amount") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot(sampled_train_set_class, aes(fill = Card_Type, y = Amount)) +
  geom_boxplot() +
  labs(title = "Boxplot of Amounts by Card Type",
       x = "Card Type", y = "Amount") +
  theme(plot.title = element_text(hjust = 0.5))
```

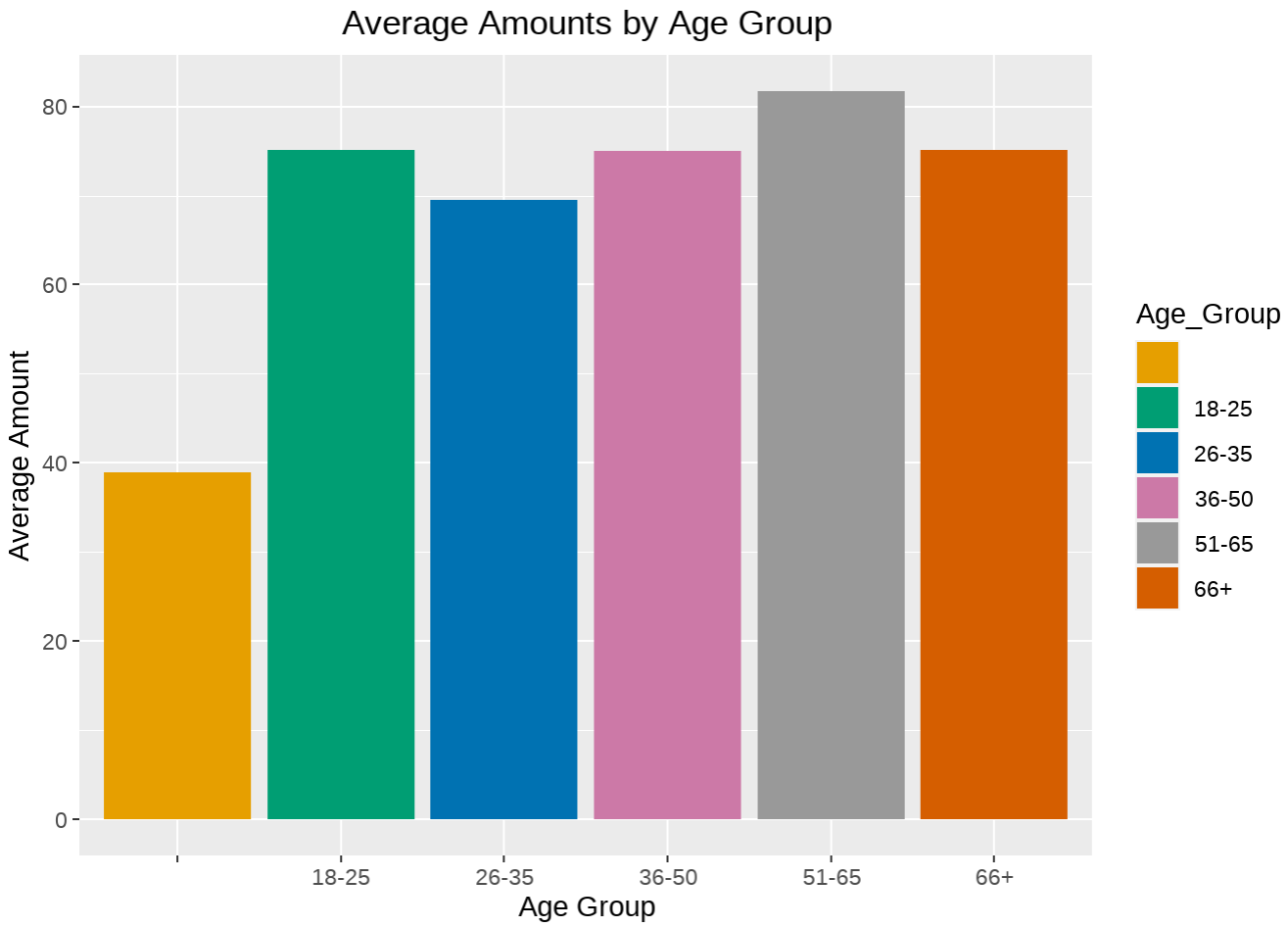


We don't much difference between the different values for the categorical variables.

Bar Chart of Amounts by Age Group

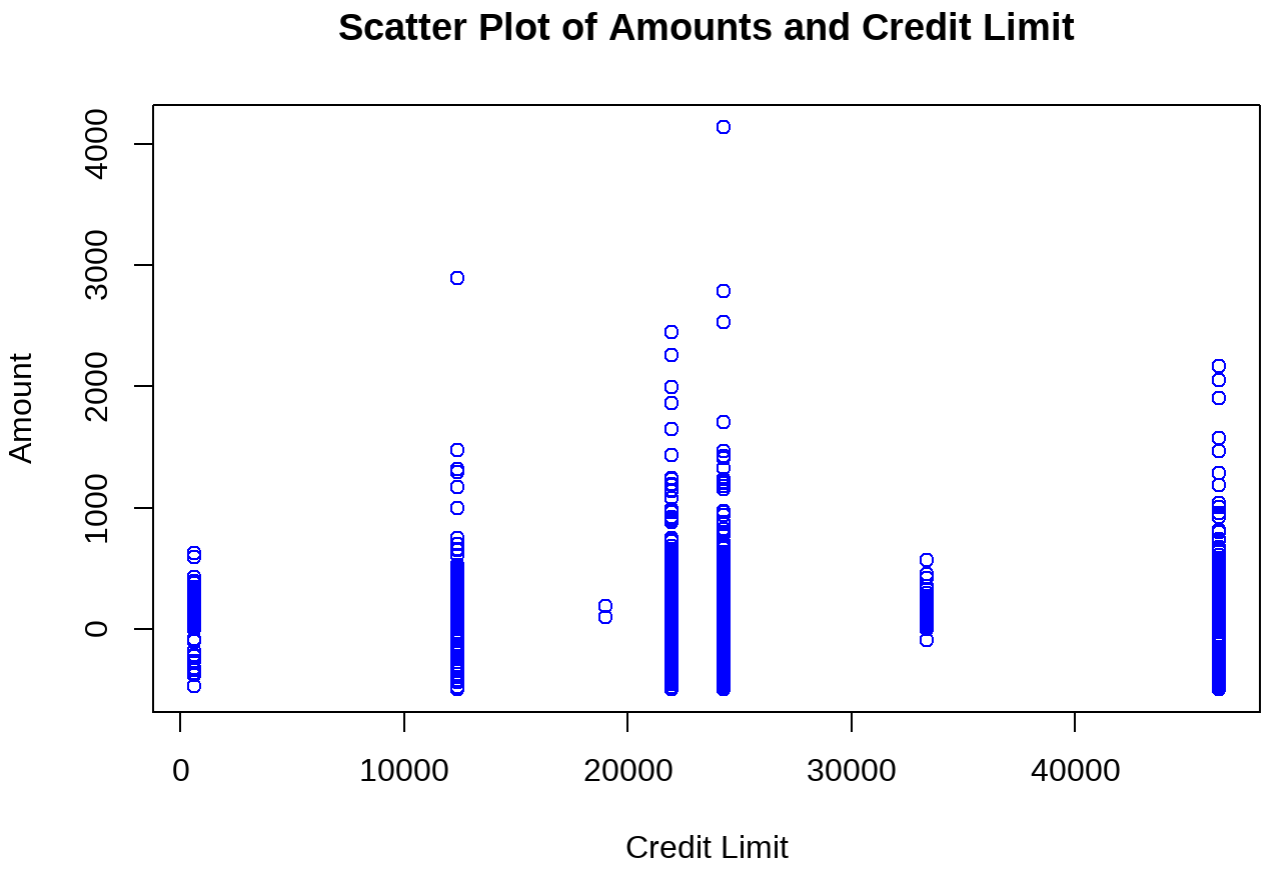
Visualize the average transaction amounts for different age groups. It can be seen that there is not much difference actually

```
library(ggplot2)
ggplot(sampled_train_set_class, aes(x = Age_Group, y = Amount, fill = Age_Group)) +
  geom_bar(stat = "summary", fun = "mean", position = "dodge") +
  labs(title = "Average Amounts by Age Group", x = "Age Group", y = "Average Amount") +
  theme(plot.title = element_text(hjust = 0.5))
```



Scatter Plot of Amounts and Credit Limit

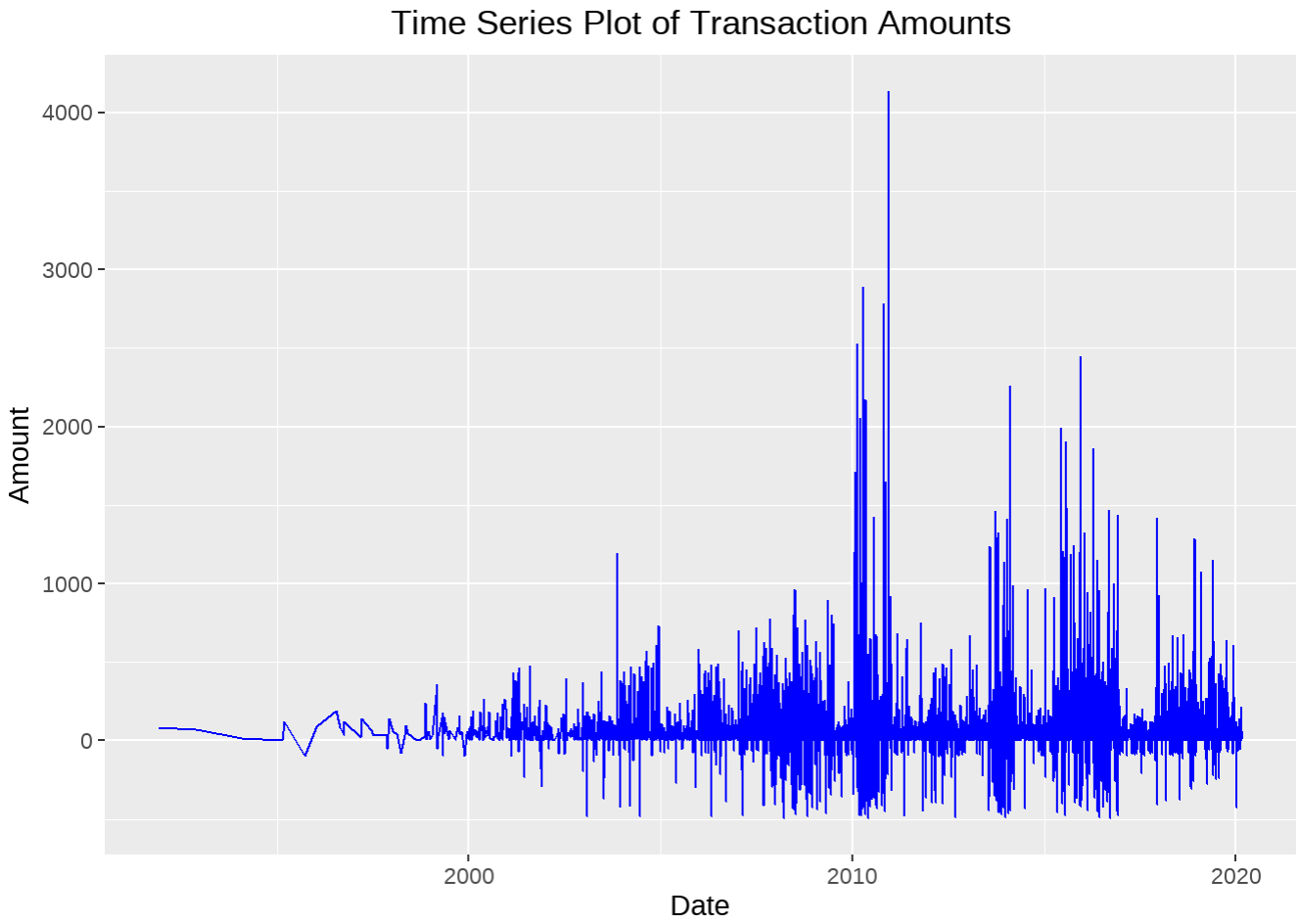
```
plot(sampled_train_set_class$Credit_Limit, sampled_train_set_class$Amount, main = "Scatter Plo
t of Amounts and Credit Limit",
     xlab = "Credit Limit", ylab = "Amount", col = "blue") +
  theme(plot.title = element_text(hjust = 0.5))
```



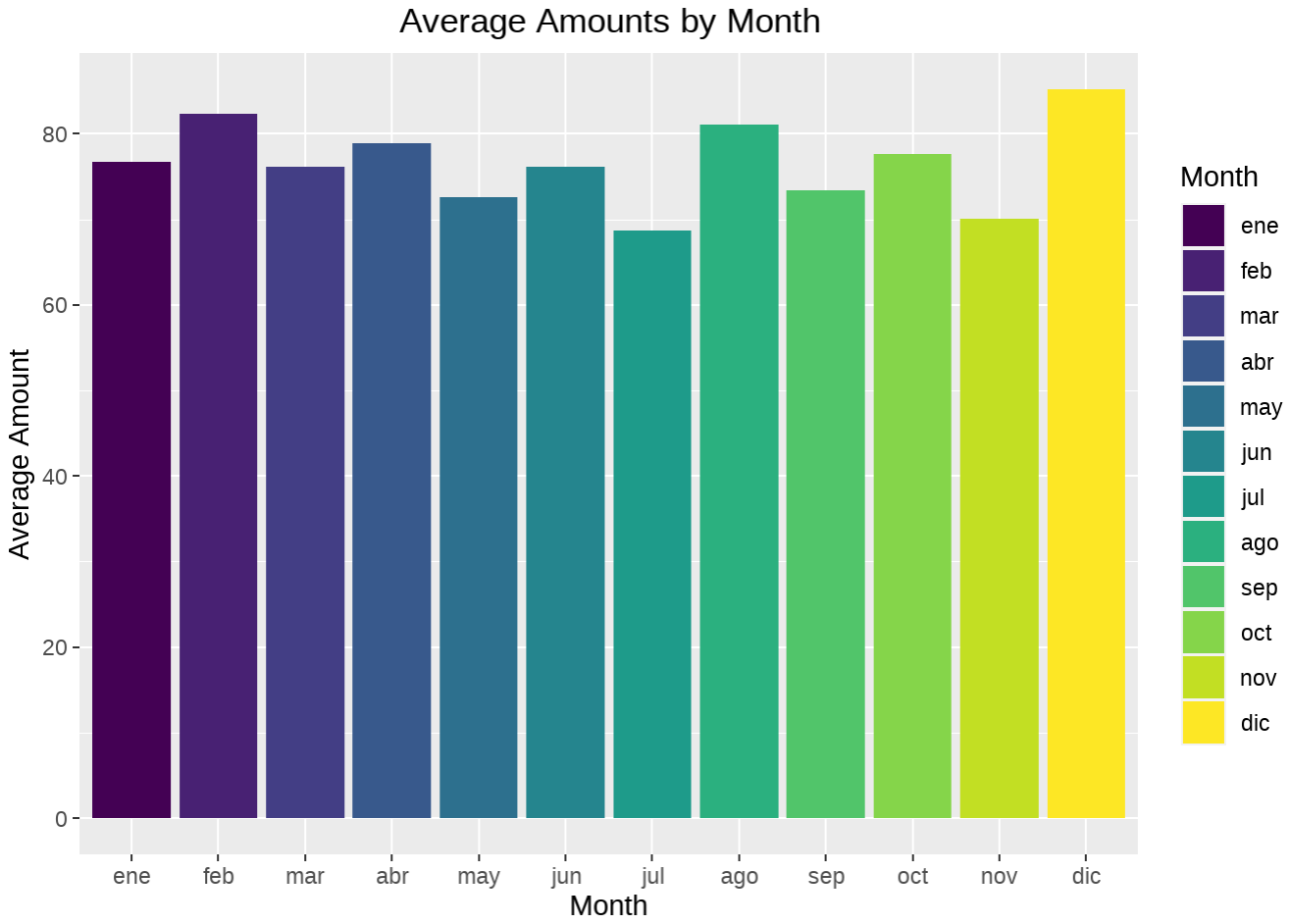

```
## NULL
```

Time Series Plot of Amounts

```
ggplot(sampled_train_set_class, aes(x = Date, y = Amount, group = 1)) +
  geom_line(color = "blue") +
  labs(title = "Time Series Plot of Transaction Amounts", x = "Date", y = "Amount") +
  theme(plot.title = element_text(hjust = 0.5))
```

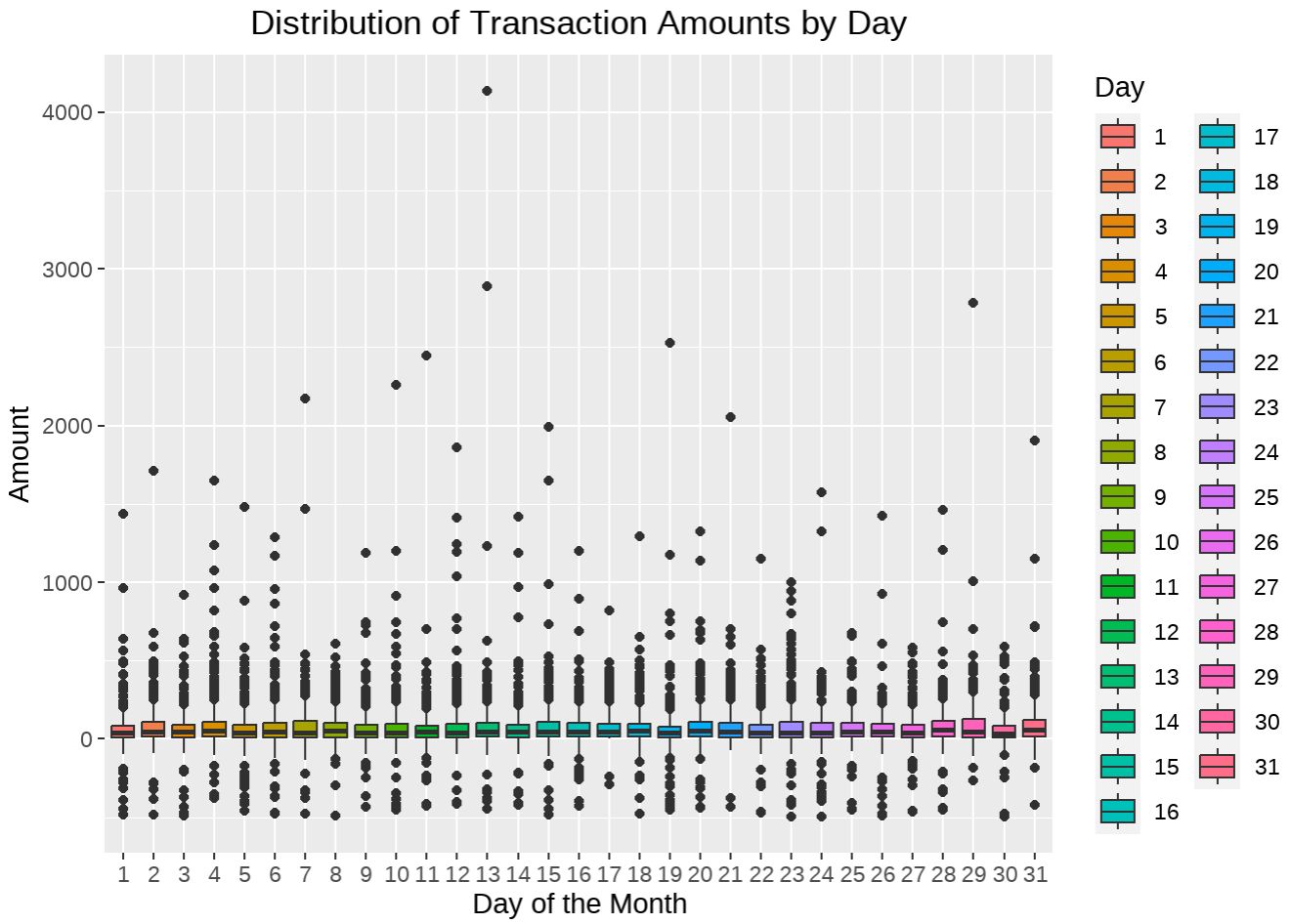



```
ggplot(sampled_train_set_class, aes(x = factor(month(Date, label = TRUE)), y = Amount, fill =
factor(month(Date, label = TRUE)))) +
  geom_bar(stat = "summary", fun = "mean", position = "dodge") +
  labs(title = "Average Amounts by Month", x = "Month", y = "Average Amount", fill = "Month")
+
  theme(plot.title = element_text(hjust = 0.5))
```



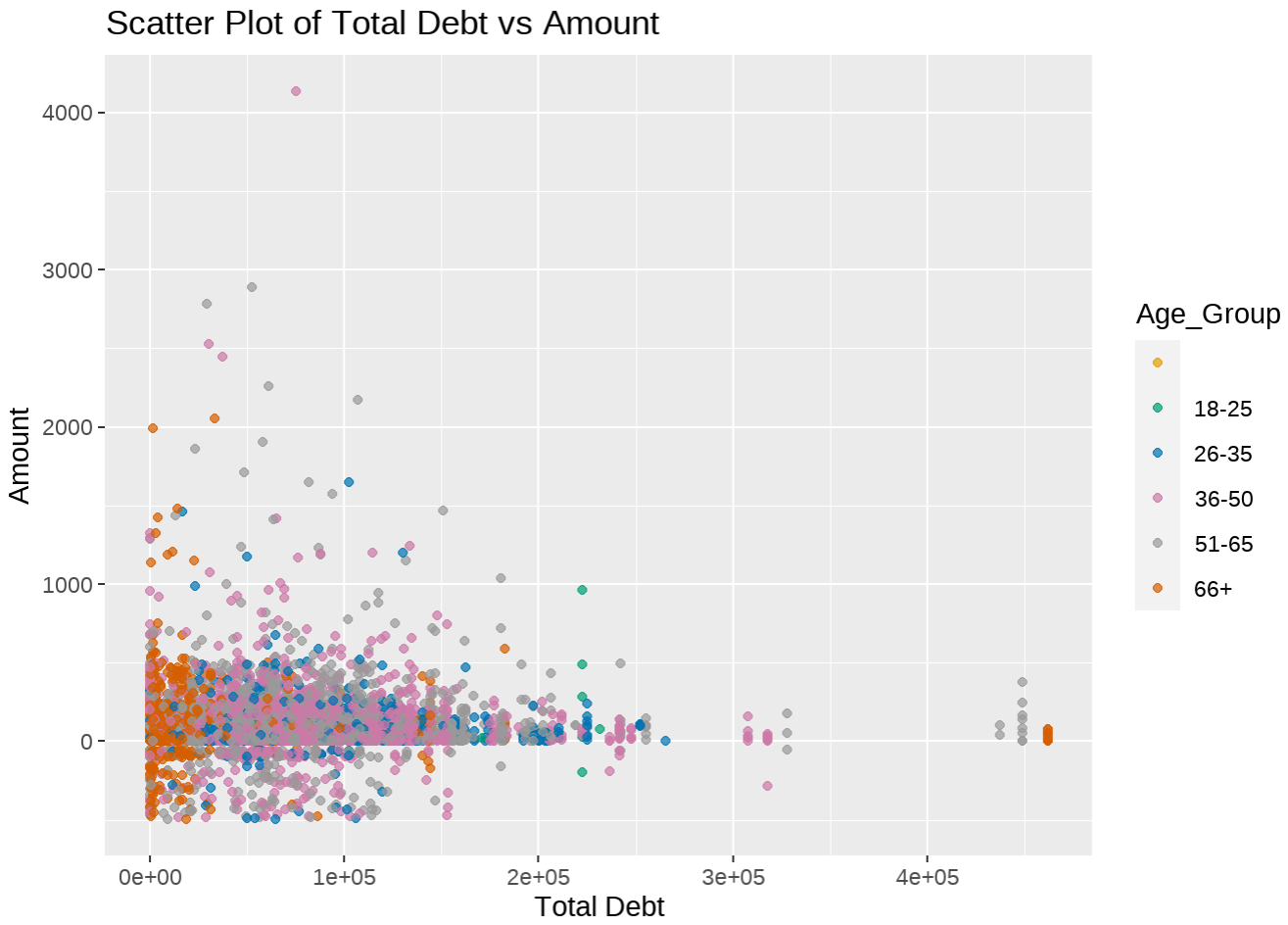
While observing the average transaction amounts across different months, we note minimal variation. However, a slight increase is evident in December, likely attributed to Christmas-related expenditures, and in August, possibly due to holiday-related expenses

```
ggplot(sampled_train_set_class, aes(x = factor(day(Date)), y = Amount, fill = factor(day(Date)
)  )) +
  geom_boxplot() +
  labs(title = "Distribution of Transaction Amounts by Day", x = "Day of the Month", y = "Amou
nt", fill = "Day") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
# Scatter Plot for Total_Debt and Amount
ggplot(sampled_train_set_class, aes(x = Total_Debt, y = Amount, color = Age_Group)) +
  geom_point(alpha = 0.7) +
  labs(title = "Scatter Plot of Total Debt vs Amount", x = "Total Debt", y = "Amount")
```

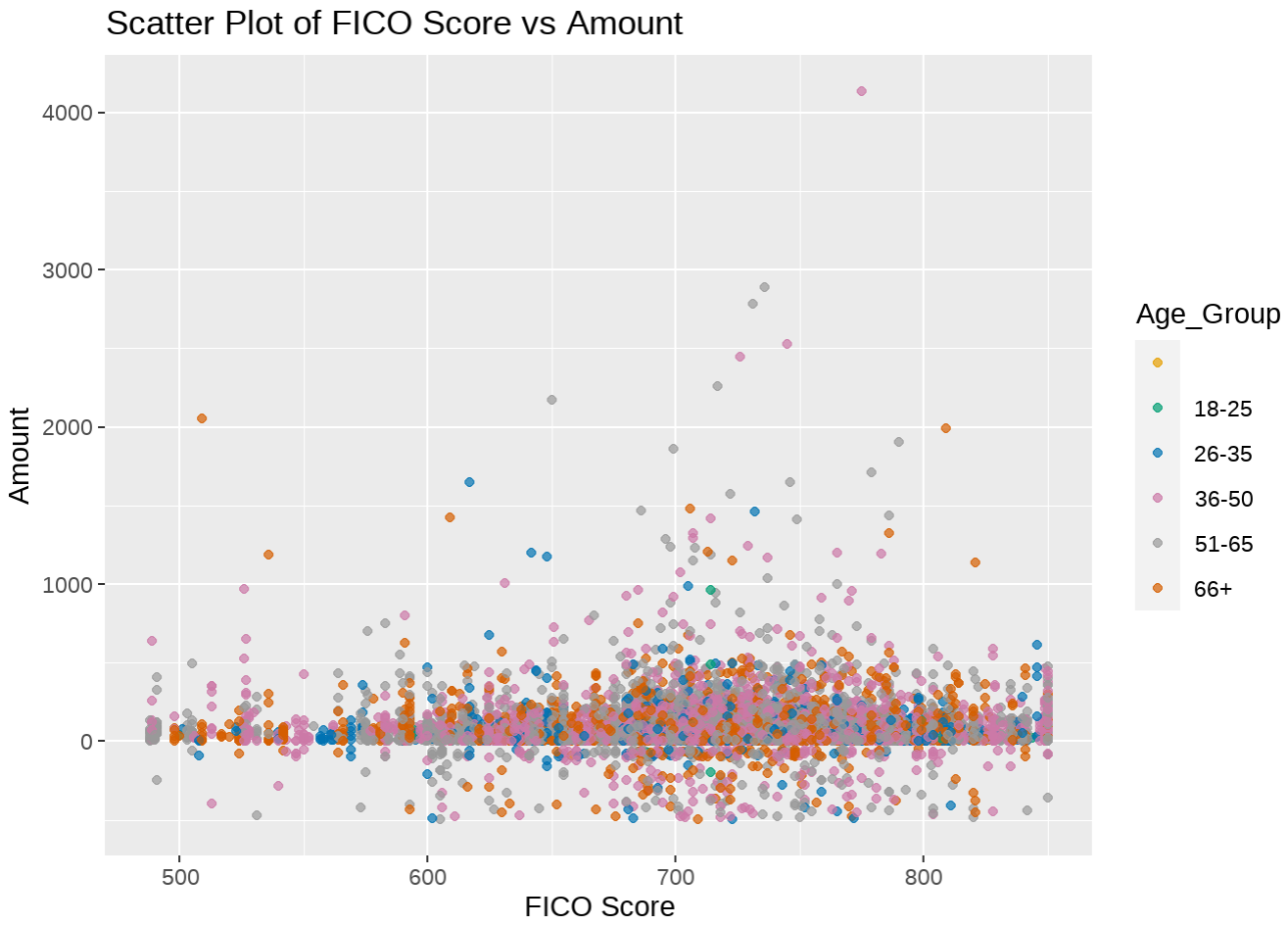
As expected



with more debt people make less transactions and of smaller quantities

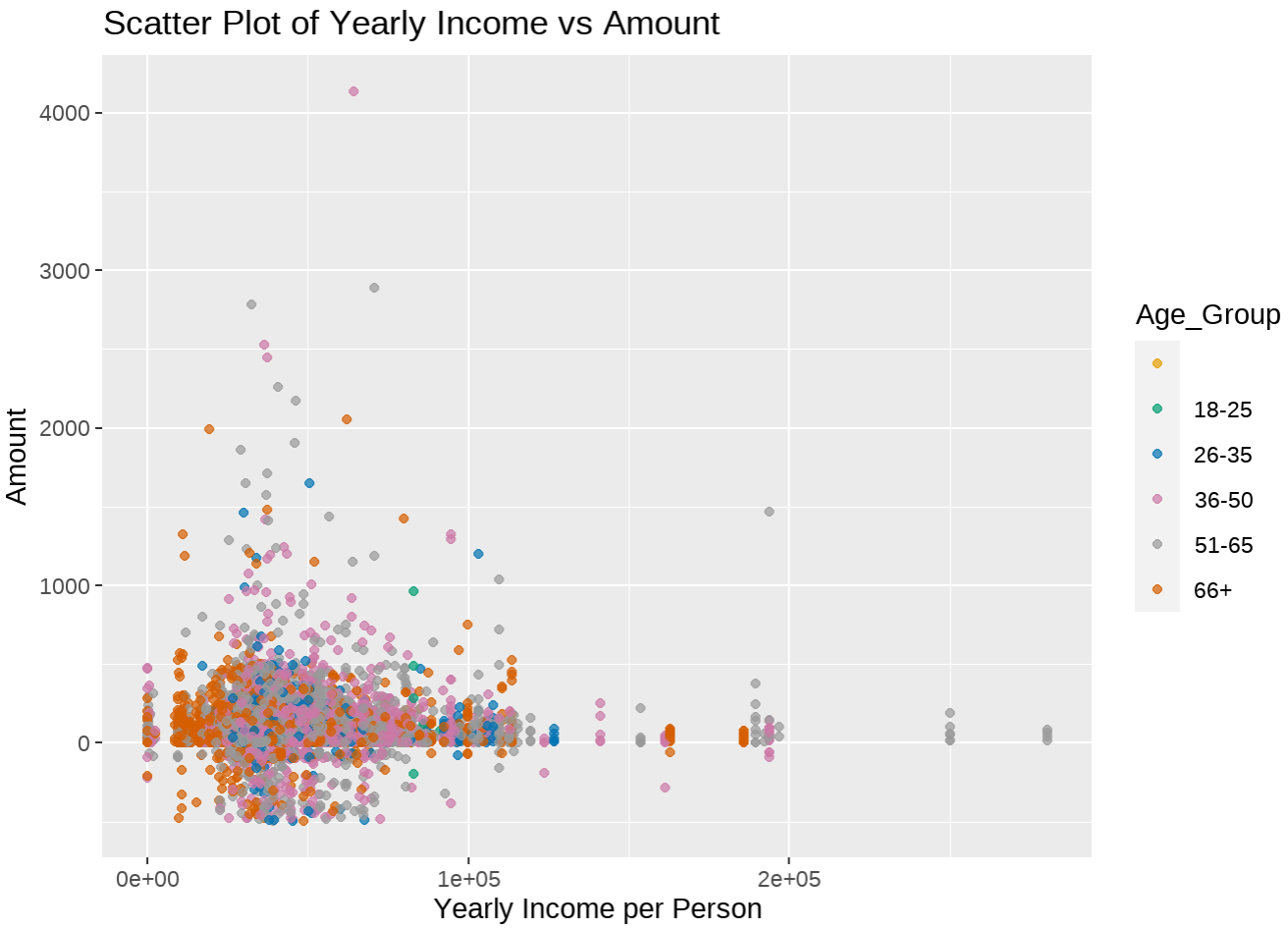
```
# Scatter Plot for FICO_Score and Amount
ggplot(sampled_train_set_class, aes(x = FICO_Score, y = Amount, color =  Age_Group)) +
  geom_point(alpha = 0.7) +
  labs(title = "Scatter Plot of FICO Score vs Amount", x = "FICO Score", y = "Amount")
```

There is no a

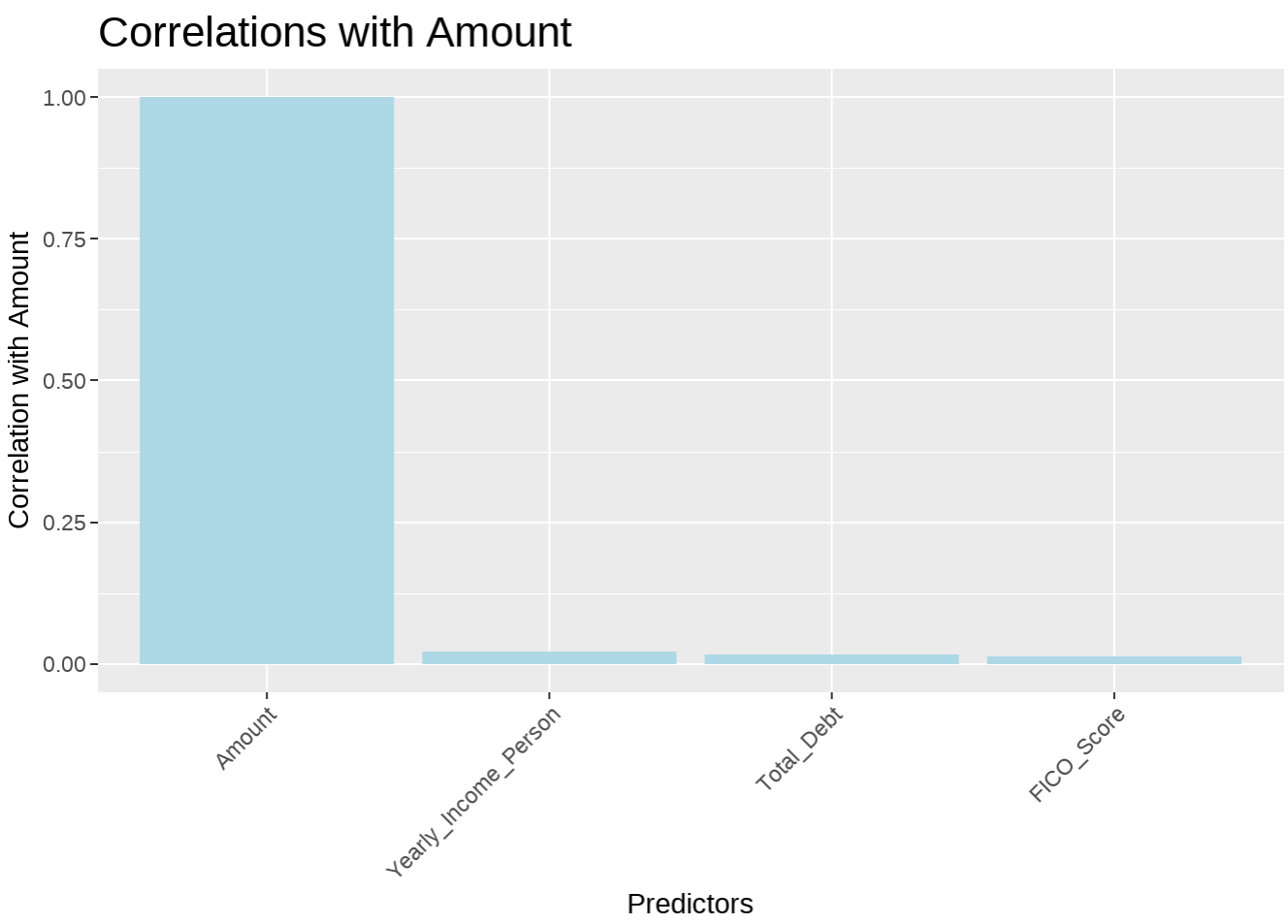


clear relationship

```
# Scatter Plot for Yearly_Income_Person and Amount
ggplot(sampled_train_set_class, aes(x = Yearly_Income_Person, y = Amount, color =  Age_Group))
 +
  geom_point(alpha = 0.7) +
  labs(title = "Scatter Plot of Yearly Income vs Amount", x = "Yearly Income per Person", y =
"Amount")
```




```
# Assuming your dataframe is named sampled_train_set_class
corr_amount <- sort(cor(sampled_train_set_class[, c("Total_Debt", "FICO_Score", "Yearly_Income
_Person", "Amount")])["Amount", ], decreasing = TRUE)
corr_data <- data.frame(variable = names(corr_amount), correlation = corr_amount)

library(ggplot2)

ggplot(corr_data, aes(x = variable, y = correlation)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  scale_x_discrete(limits = corr_data$variable) +
  labs(x = "Predictors", y = "Correlation with Amount", title = "Correlations with Amount") +
  theme(plot.title = element_text(hjust = 0, size = rel(1.5)),
        axis.text.x = element_text(angle = 45, hjust = 1))
```



It can be seen that the numerical variables are not very related with the Amount.

5.3 Statistical Tools for Regression

5.3.1 Linear Regression

5.3.1.1 Benchmark

The benchmark serves as our baseline, and in our case, it corresponds to predicting using the mean of transaction amounts. Our models consistently should exhibit higher accuracy compared to this benchmark.

```
mean(sampled_train_set_class$Amount) # 76.64091
```

```
## [1] 76.64091
```

```
benchFit <- lm(Amount ~ 1, data=sampled_train_set_class)
predictions <- predict(benchFit, newdata=sampled_test_set_class)
```

```
RMSE = sqrt(mean((predictions - sampled_test_set_class$Amount)^2))
RMSE # 92.78234
```

```
## [1] 92.78234
```

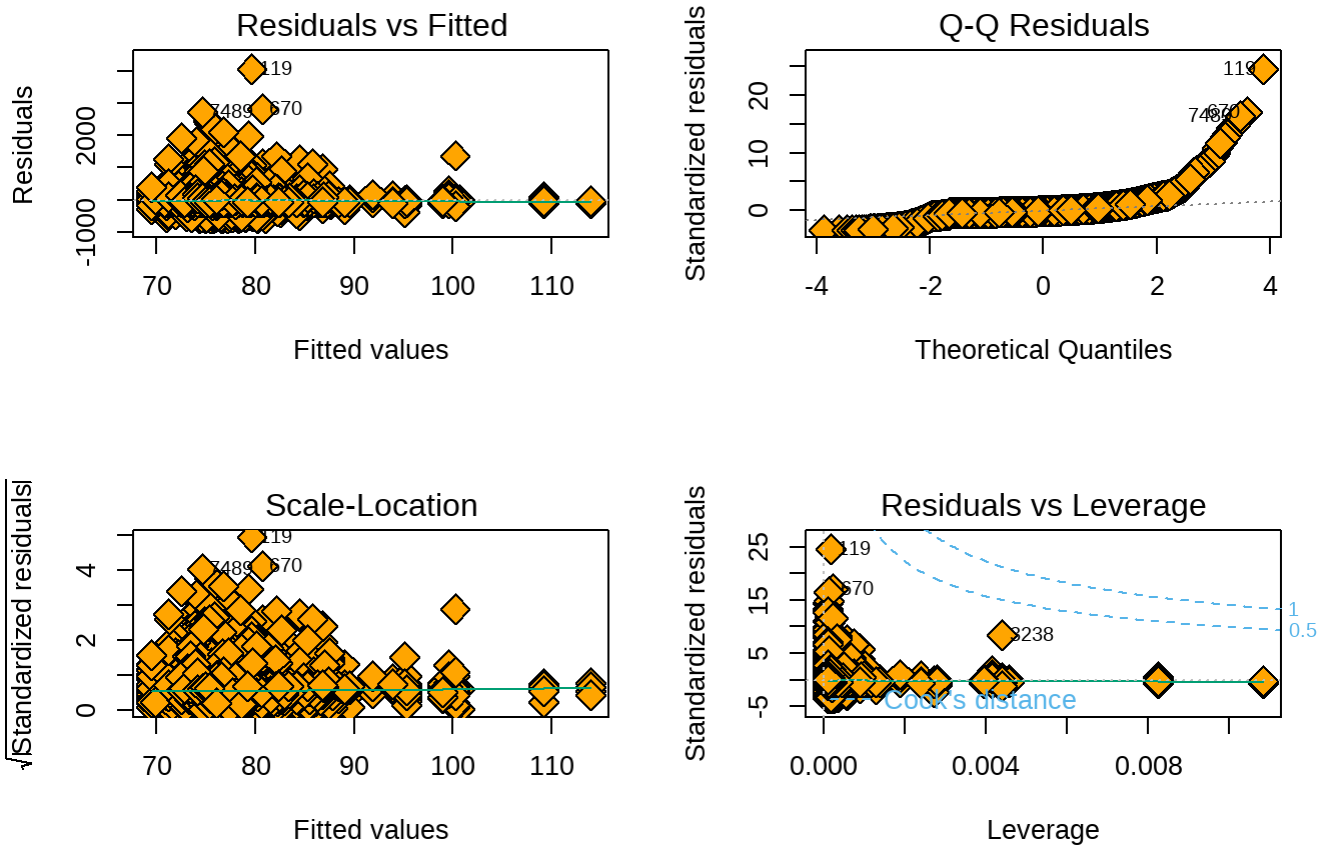
5.3.1.2 Simple linear regression

```
linFit <- lm(Amount ~ Yearly_Income_Person, data=sampled_train_set_class)
summary(linFit)
```

```
##
## Call:
## lm(formula = Amount ~ Yearly_Income_Person, data = sampled_train_set_class)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -571.7  -64.4  -31.9   23.9 4053.6
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           6.944e+01  3.709e+00  18.723   <2e-16 ***
## Yearly_Income_Person 1.588e-04  7.288e-05   2.179   0.0293 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 165.2 on 9670 degrees of freedom
## Multiple R-squared:  0.0004909,  Adjusted R-squared:  0.0003875
## F-statistic: 4.749 on 1 and 9670 DF,  p-value: 0.02933
```

The linear regression model aims to predict the Amount variable based on the Yearly_Income_Person predictor. The estimated intercept is 69.44, indicating the expected Amount when Yearly_Income_Person is zero. The coefficient for Yearly_Income_Person is 0.0001588, suggesting that, on average, a one-unit increase in Yearly_Income_Person is associated with an increase of 0.0001588 units in Amount. The p-value for the Yearly_Income_Person coefficient is 0.0293, which is below the conventional significance level of 0.05, suggesting that there is evidence to reject the null hypothesis that the coefficient is zero. However, the low multiple R-squared value (0.0004909) indicates that the model explains only a very small proportion of the variance in Amount. Therefore, while there is a statistically significant relationship between Yearly_Income_Person and Amount, the practical significance might be limited given the small effect size and low explanatory power of the model.

```
par(mfrow=c(2,2))
plot(linFit, pch=23 ,bg='orange',cex=2)
```




```
pr.simple = exp(predict(linFit, newdata=sampled_test_set_class))
cor(sampled_test_set_class$Amount, pr.simple)^2
```

```
## [1] 4.368747e-05
```

This is a really bad model.

5.3.1.3 Multiple Linear Regression

```
multi_linFit <- lm(Amount ~ Total_Debt + FICO_Score + Yearly_Income_Person, data = sampled_tra
in_set_class)

# Print summary of the multiple regression model
summary(multi_linFit)
```

```
##
## Call:
## lm(formula = Amount ~ Total_Debt + FICO_Score + Yearly_Income_Person,
##     data = sampled_train_set_class)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -572.2  -64.4  -31.8   23.9 4051.3
##
```

```
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          4.162e+01  1.883e+01   2.210   0.0271 *
## Total_Debt           2.886e-05  3.832e-05   0.753   0.4514
## FICO_Score           3.848e-02  2.567e-02   1.499   0.1338
## Yearly_Income_Person 1.305e-04  8.466e-05   1.542   0.1231
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 165.2 on 9668 degrees of freedom
## Multiple R-squared:  0.0007607,  Adjusted R-squared:  0.0004506
## F-statistic: 2.453 on 3 and 9668 DF,  p-value: 0.06133
```

In this multiple regression model, the intercept is 41.62, indicating the expected Amount when all predictor variables are zero. Among the predictor variables, only the intercept is statistically significant (p-value = 0.0271), suggesting a potential non-zero intercept. However, the coefficients for Total_Debt, FICO_Score, and Yearly_Income_Person are not statistically significant (p-values > 0.05). The small multiple R-squared value (0.0007607) indicates that the model explains a very small proportion of the variance in Amount. The F-statistic (2.453) and its associated p-value (0.06133) suggest that the overall model may not be statistically significant. Interpretation should be cautious, and further model refinement or consideration of additional variables may be needed.

```
pr.multiple = exp(predict(multi_linFit, newdata=sampled_test_set_class))
cor(sampled_test_set_class$Amount, pr.multiple)^2
```

```
## [1] 0.0001530747
```

This low R-squared value suggests that the multiple regression model may not be effectively capturing the variability in the Amount variable in the test set.

5.3.1.4 Cross Validation with Caret

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 5, repeats = 5, allowParallel = TRUE)
```

As we need to try lots of models, let's save all predictos in the followinf dataframe

```
test_results <- data.frame(Amount = sampled_test_set_class$Amount)
```

5.3.1.4.1 Linear Regression

First, define our model

```
model = Amount ~ Total_Debt + FICO_Score + Yearly_Income_Person
```

Then, we train our model

```
lm_tune <- train(model, data = sampled_train_set_class,
                 method = "lm",
                 preProc=c('scale', 'center'),
```

```
                  trControl = ctrl)
lm_tune
```

```
## Linear Regression
##
## 9672 samples
##    3 predictor
##
## Pre-processing: scaled (3), centered (3)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 7738, 7738, 7739, 7737, 7736, 7739, ...
## Resampling results:
##
##   RMSE      Rsquared      MAE
##   164.9308  0.0005890773  84.50019
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

In summary, the linear regression model, after tuning and cross-validation, exhibits an RMSE of 164.9308, an R-squared value of 0.0005890773, and an MAE of 84.50019. While the model provides predictions, the low R-squared value indicates that the predictors may not be effectively explaining the variability in the response variable.

Prediction

```
test_results$lm <- predict(lm_tune, sampled_test_set_class)
postResample(pred = test_results$lm,  obs = test_results$Amount)
```

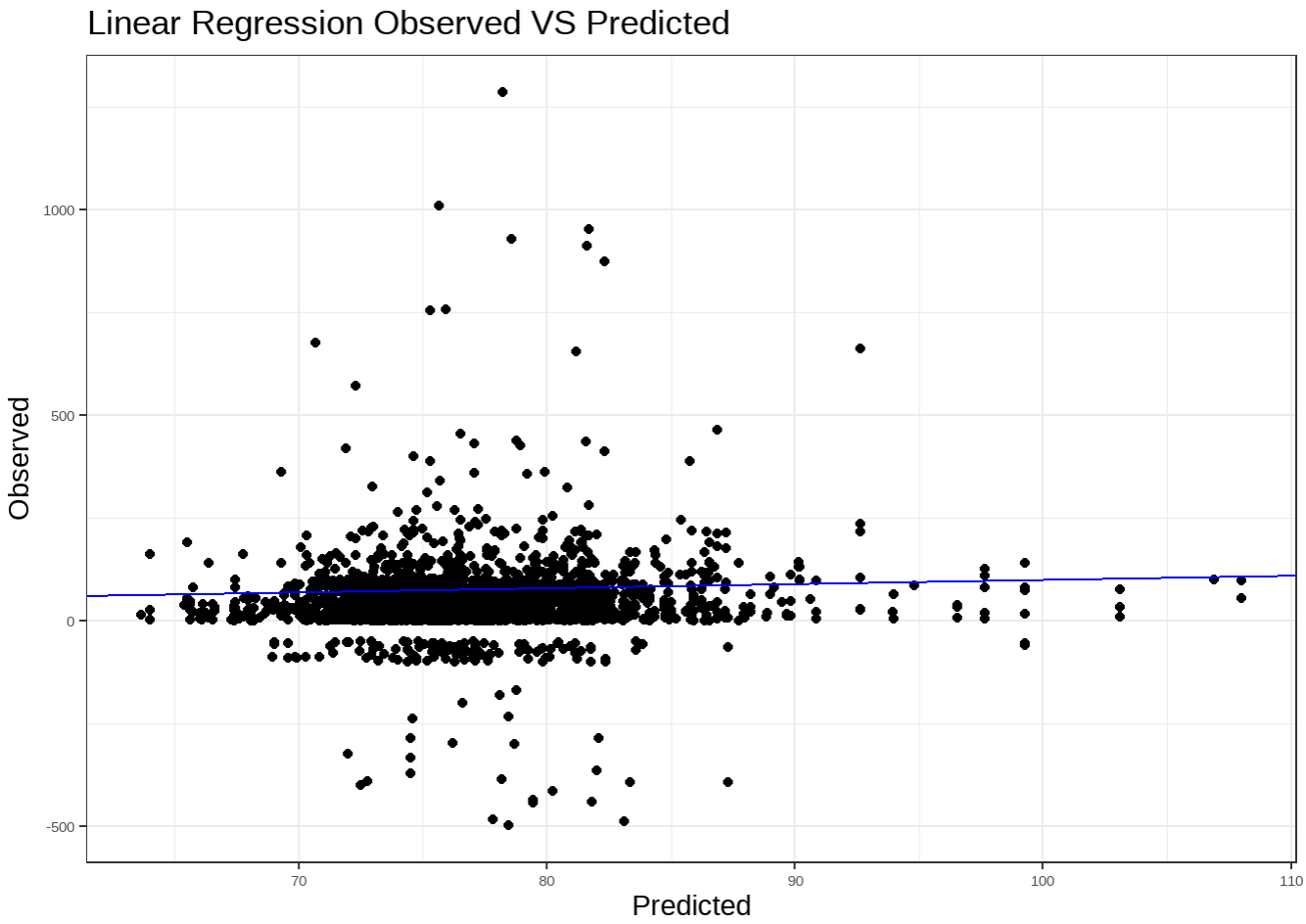
```
##         RMSE     Rsquared          MAE
## 92.608041265  0.006105115 59.978977501
```

```
    RMSE     Rsquared          MAE
```

92.608041265 0.006105115 59.978977501

Visualization

```
ggplot(test_results, aes(x = lm, y = Amount)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, colour = "blue") +
  labs(title = "Linear Regression Observed VS Predicted", x = "Predicted", y = "Observed") +
  theme_bw()
```



We can see that the model don't look really linear

5.3.1.4.2 Overfitted Linear Regression

First, define our model

```
model2 = Amount ~ Total_Debt + FICO_Score + Yearly_Income_Person + Credit_Limit + Num_Credit_C
ards + Current_Age + Latitude + Longitude
```

Then, we train our model

```
lm_tune2 <- train(model2, data = sampled_train_set_class,
                  method = "lm",
                  preProc=c('scale', 'center'),
                  trControl = ctrl)
lm_tune2
```

```
## Linear Regression
##
## 9672 samples
##    8 predictor
##
## Pre-processing: scaled (8), centered (8)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 7736, 7740, 7737, 7737, 7738, 7737, ...
```

```
## Resampling results:
##
##   RMSE      Rsquared      MAE
##   164.6602  0.0006802417  84.44921
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

RMSE Rsquared MAE
164.6863 0.0008118477 84.45255

Prediction

```
test_results$lm2 <- predict(lm_tune2, sampled_test_set_class)
postResample(pred = test_results$lm2,  obs = test_results$Amount)
```

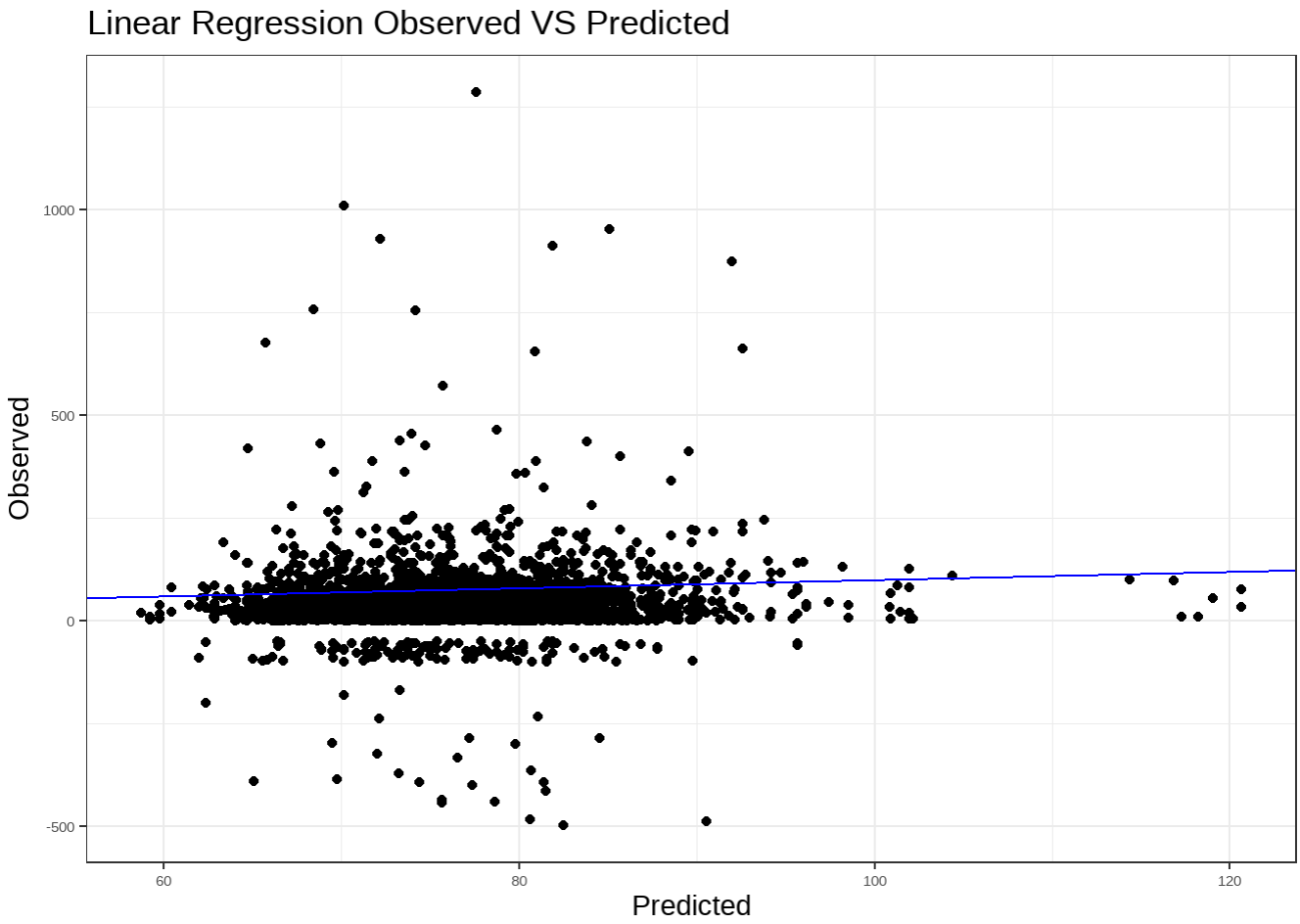
```
##         RMSE     Rsquared          MAE
## 92.542403295  0.002741965 59.770585144
```

```
    RMSE     Rsquared          MAE
```

92.542403295 0.002741965 59.770585144

Visualization

```
ggplot(test_results, aes(x = lm2, y = Amount)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, colour = "blue") +
  labs(title = "Linear Regression Observed VS Predicted", x = "Predicted", y = "Observed") +
  theme_bw()
```



Actually pretty similar than the simple regression.

5.3.2 Variable Selection Techniques

5.3.2.1 Forward regression

The forward regression process incrementally builds the model by selecting predictors one at a time based on their individual contribution. It's important to note that forward regression may not explore all possible combinations of predictors, and the final model may not be the best overall model. It is a greedy algorithm that makes locally optimal choices at each step.

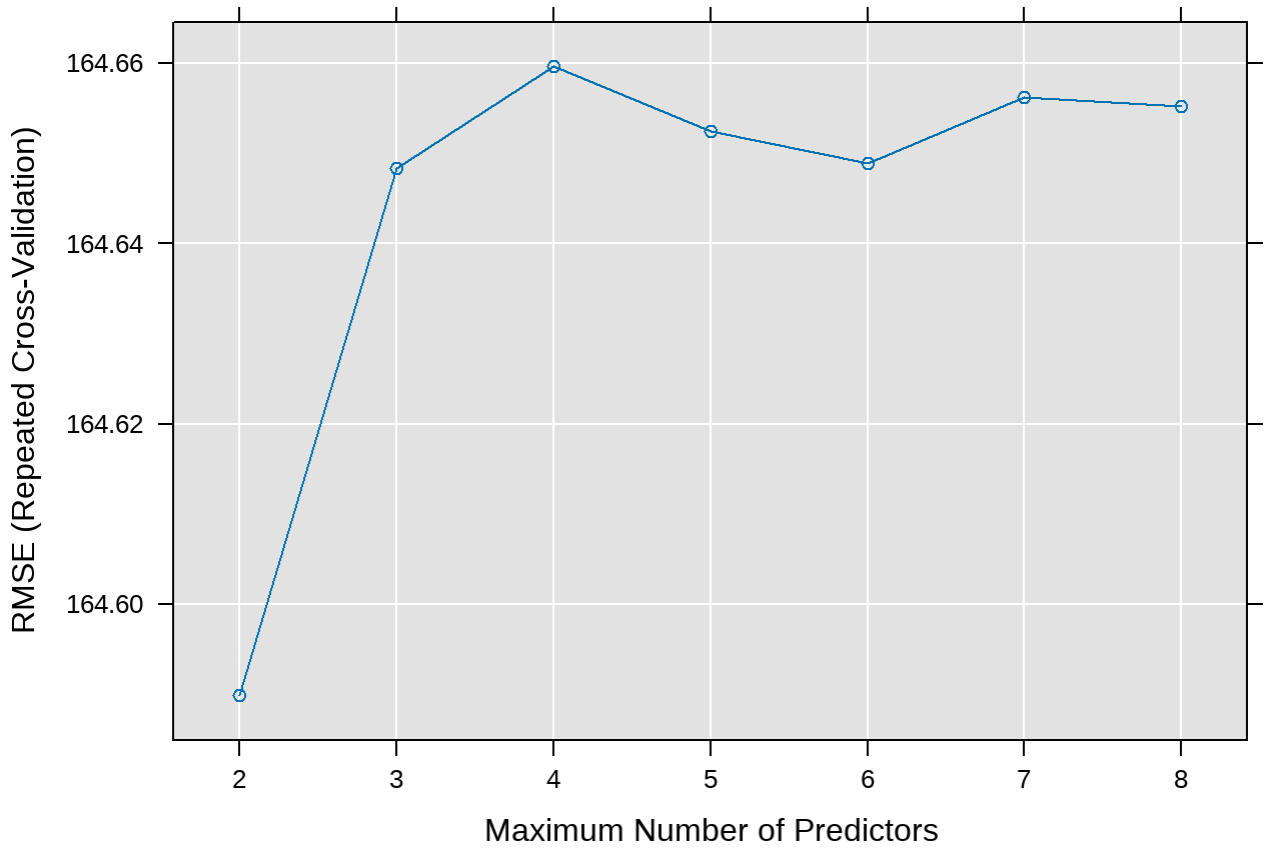
```
for_tune <- train(model2, data = sampled_train_set_class,
                  method = "leapForward",
                  preProc=c('scale', 'center'),
                  tuneGrid = expand.grid(nvmax = 2:8),
                  trControl = ctrl)

for_tune
```

```
## Linear Regression with Forward Selection
##
## 9672 samples
##    8 predictor
##
## Pre-processing: scaled (8), centered (8)
```

```
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 7739, 7738, 7737, 7736, 7738, 7737, ...
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared      MAE
##   2      164.5899  0.0010158155  84.42709
##   3      164.6483  0.0006162420  84.46872
##   4      164.6597  0.0006230957  84.46689
##   5      164.6525  0.0006499417  84.45545
##   6      164.6490  0.0006689309  84.45253
##   7      164.6563  0.0006235875  84.45562
##   8      164.6553  0.0006259610  84.45368
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was nvmax = 2.
```

The train function with forward variable selection (leapForward method) has been performed on model2 with different values of nvmax. The results indicate the performance metrics (RMSE, Rsquared, MAE) for each value of nvmax. The selected model has nvmax = 2 based on the smallest RMSE value.

```
plot(for_tune)
```



Now we need to select those best variables

```
coef(for_tune$finalModel, for_tune$bestTune$nvmax)
```

```
##          (Intercept) Yearly_Income_Person     Num_Credit_Cards
##            76.640910             3.842341             4.474577
```

The coefficients for Yearly_Income_Person and Num_Credit_Cards indicate the estimated change in the response variable for a one-unit increase in each of these predictors, assuming all other variables are held constant

Prediction

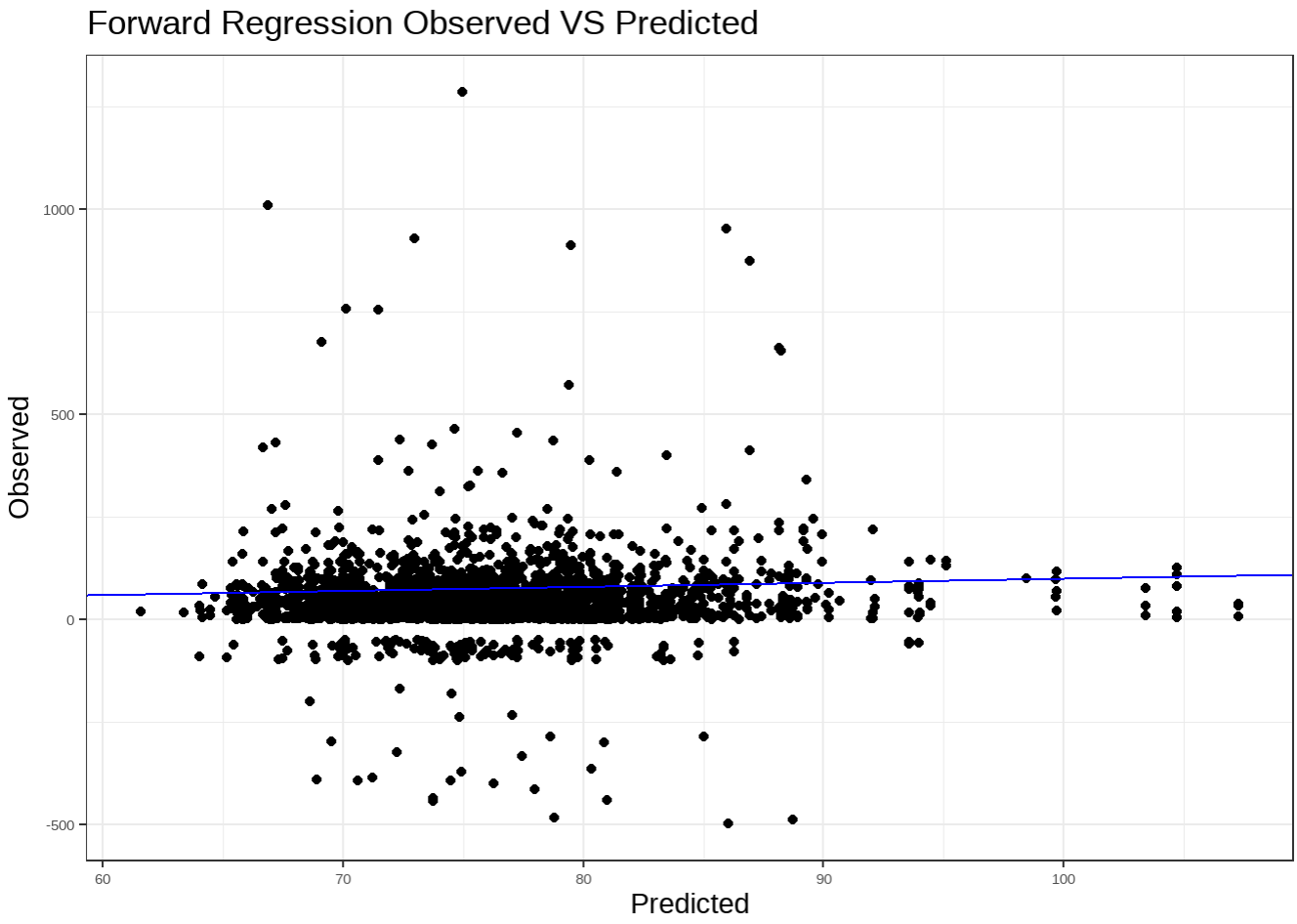
```
test_results$frw <- predict(for_tune, sampled_test_set_class)
postResample(pred = test_results$frw,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
## 92.530832150  0.002415664 59.769348206
```

```
#         RMSE     Rsquared          MAE
# 92.530832150  0.002415664 59.769348206
```

Visualization

```
ggplot(test_results, aes(x = frw, y = Amount)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, colour = "blue") +
  labs(title = "Forward Regression Observed VS Predicted", x = "Predicted", y = "Observed") +
  theme_bw()
```



Very similar to lm

5.3.2.2 Forward regression

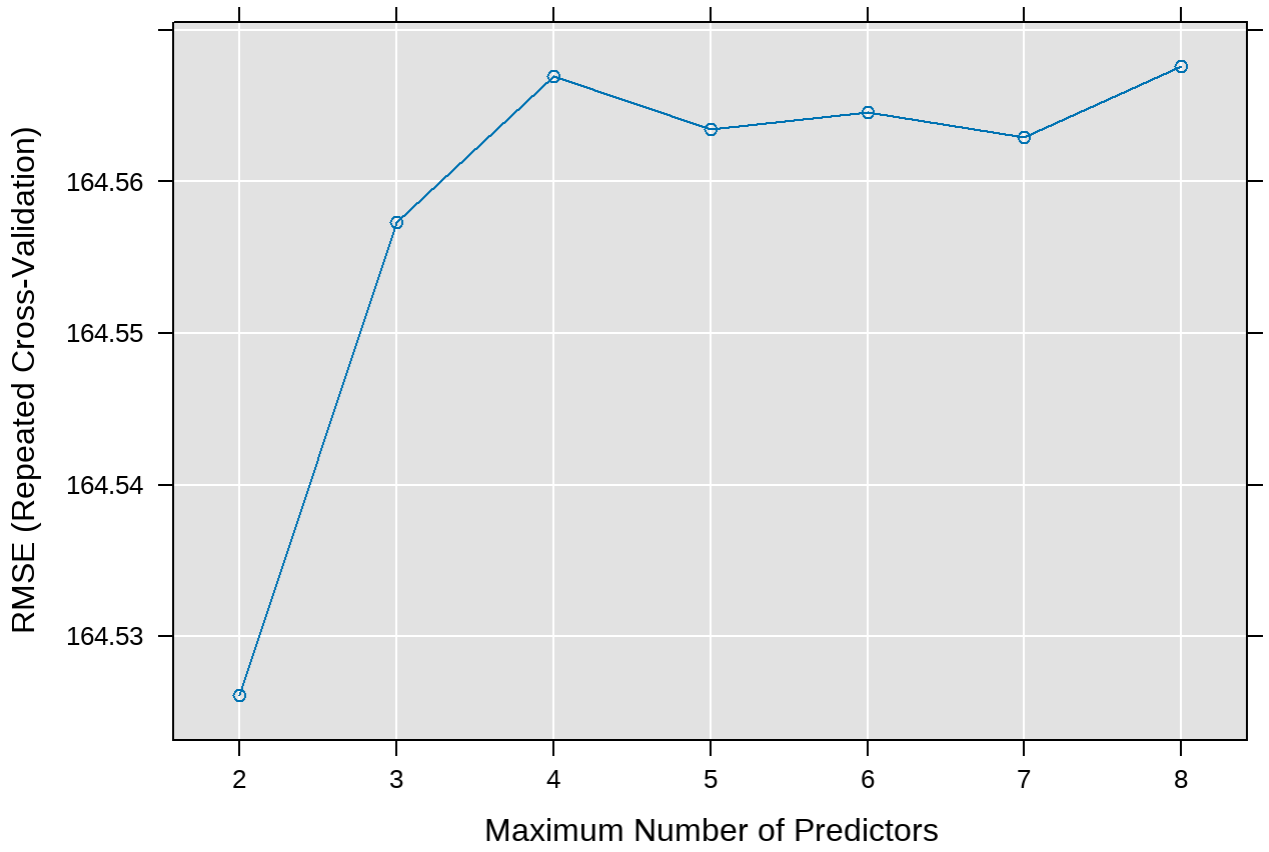
```
back_tune  <- train(model2, data = sampled_train_set_class,
                    method = "leapBackward",
                    preProc=c('scale', 'center'),
                    tuneGrid = expand.grid(nvmax = 2:8),
                    trControl = ctrl)

back_tune
```

```
## Linear Regression with Backwards Selection
##
## 9672 samples
##    8 predictor
##
## Pre-processing: scaled (8), centered (8)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 7738, 7738, 7738, 7737, 7737, 7738, ...
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared      MAE
##   2      164.5261  0.0008064521  84.45996
##   3      164.5573  0.0007143835  84.47250
```

```
##    4      164.5669  0.0007261807  84.47296
##    5      164.5634  0.0007875651  84.45945
##    6      164.5645  0.0008227789  84.46169
##    7      164.5629  0.0007943198  84.45628
##    8      164.5676  0.0007763660  84.45823
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was nvmax = 2.
```

Again it selects nvmax = 2

```
plot(back_tune)
```



Now we need to select those best variables

```
coef(back_tune$finalModel, back_tune$bestTune$nvmax)
```

```
##          (Intercept) Yearly_Income_Person     Num_Credit_Cards
##            76.640910             3.842341             4.474577
```

Prediction

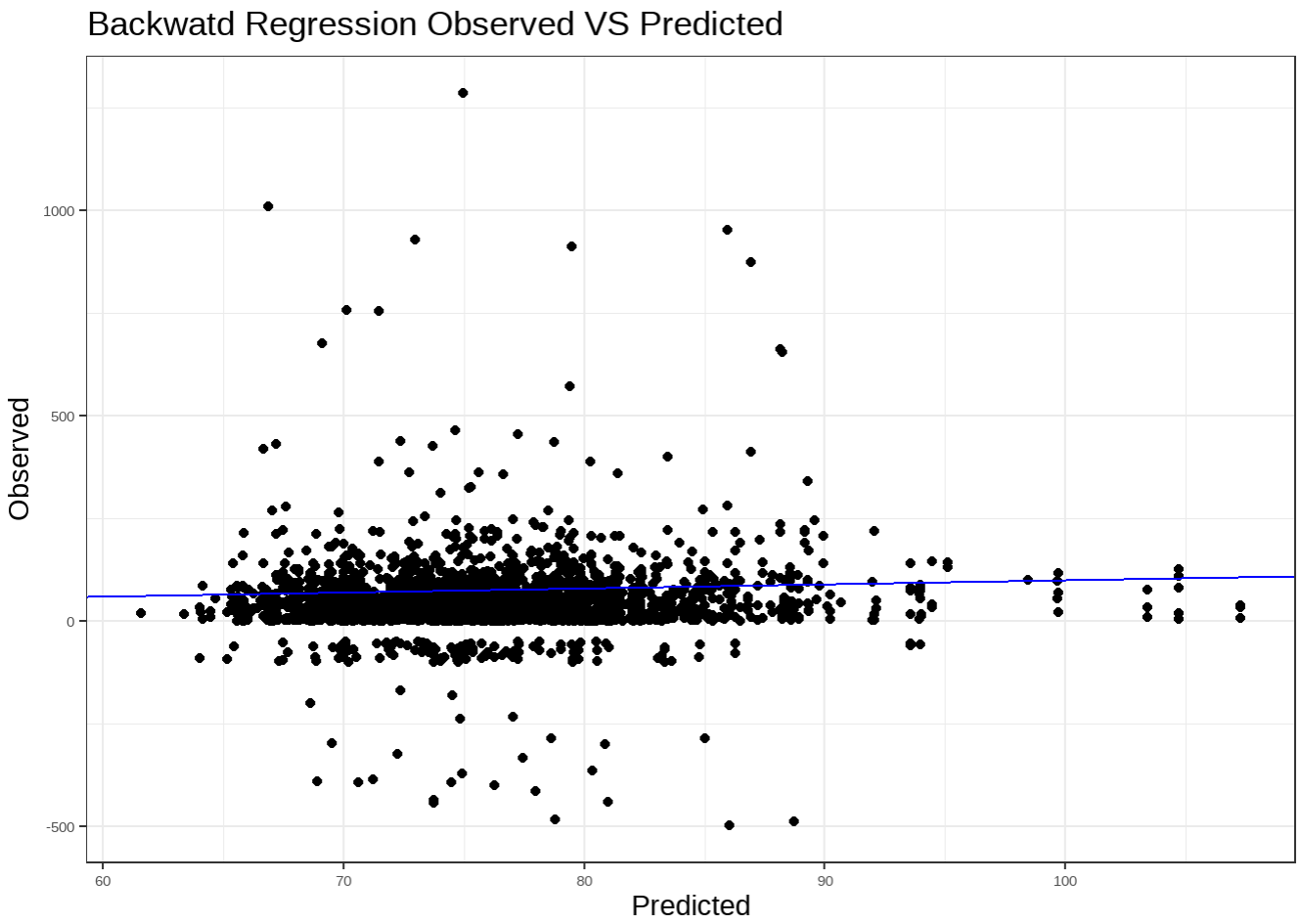
```
test_results$bw  <- predict(back_tune, sampled_test_set_class)
postResample(pred = test_results$bw ,  obs = test_results$Amount)
```

```
##          RMSE      Rsquared           MAE
## 92.530832150  0.002415664 59.769348206
```

```
#         RMSE      Rsquared           MAE
#92.530832150  0.002415664 59.769348206
```

Visualization

```
ggplot(test_results, aes(x = bw, y = Amount)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, colour = "blue") +
  labs(title = "Backwatd Regression Observed VS Predicted", x = "Predicted", y = "Observed") +
  theme_bw()
```



Very similar to lm

5.3.2.3 Stepwise regression

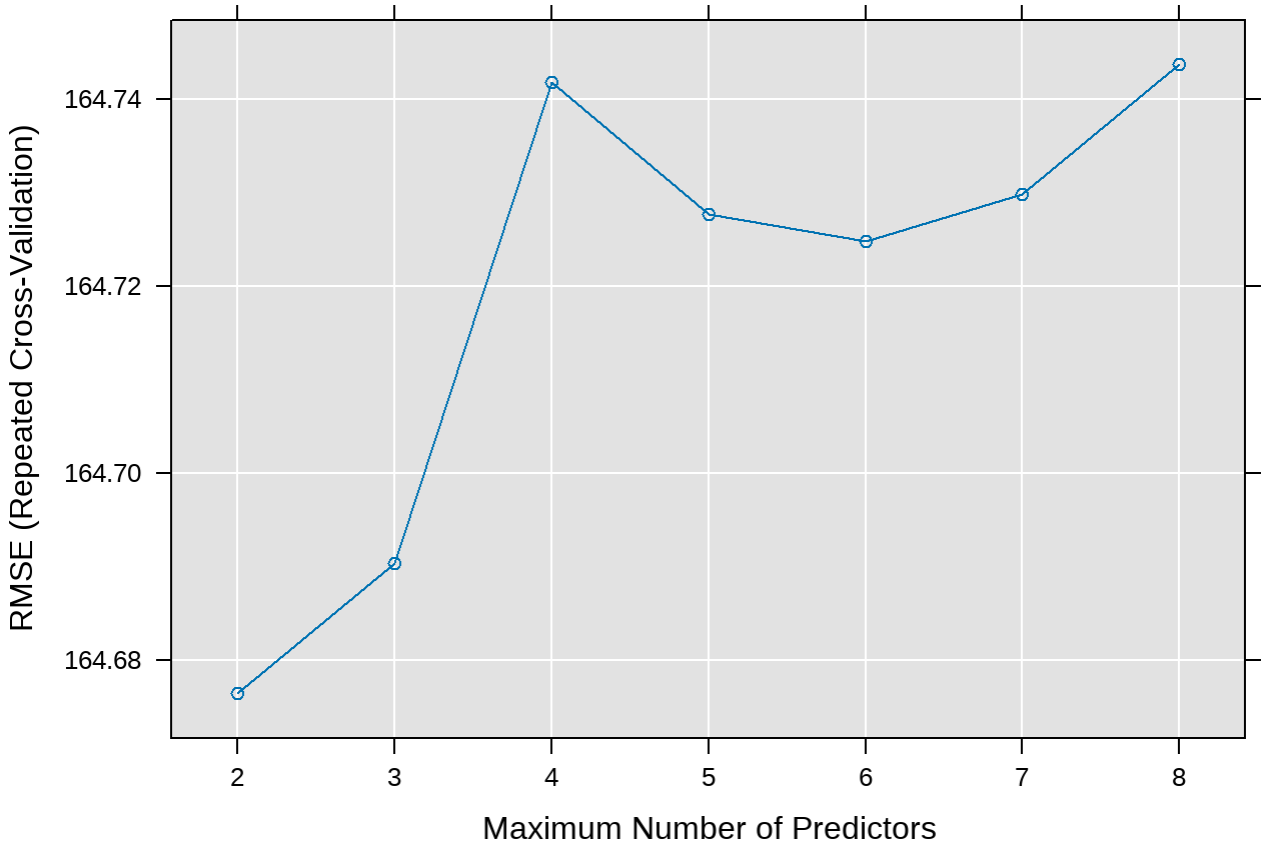
```
set.seed(123)
step_tune   <- train(model2, data = sampled_train_set_class,
                     method = "leapSeq",
                     preProc=c('scale', 'center'),
                     tuneGrid = expand.grid(nvmax = 2:8),
```

```
                  trControl = ctrl)

step_tune
```

```
## Linear Regression with Stepwise Selection
##
## 9672 samples
##    8 predictor
##
## Pre-processing: scaled (8), centered (8)
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 7737, 7738, 7737, 7738, 7738, 7738, ...
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared      MAE
##   2      164.6764  0.0012111690  84.48114
##   3      164.6903  0.0009182049  84.47449
##   4      164.7418  0.0008114145  84.50430
##   5      164.7277  0.0009074602  84.48904
##   6      164.7247  0.0009143121  84.48349
##   7      164.7298  0.0009177065  84.48695
##   8      164.7437  0.0008503171  84.49594
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was nvmax = 2.
```

Again it selects nvmax = 2

```
plot(step_tune )
```



Now we need to select those best variables

```
coef(step_tune $finalModel, step_tune $bestTune$nvmax)
```

```
##          (Intercept) Yearly_Income_Person     Num_Credit_Cards
##            76.640910             3.842341             4.474577
```

Prediction

```
test_results$seq  <- predict(step_tune , sampled_test_set_class)
postResample(pred = test_results$seq ,  obs = test_results$Amount)
```

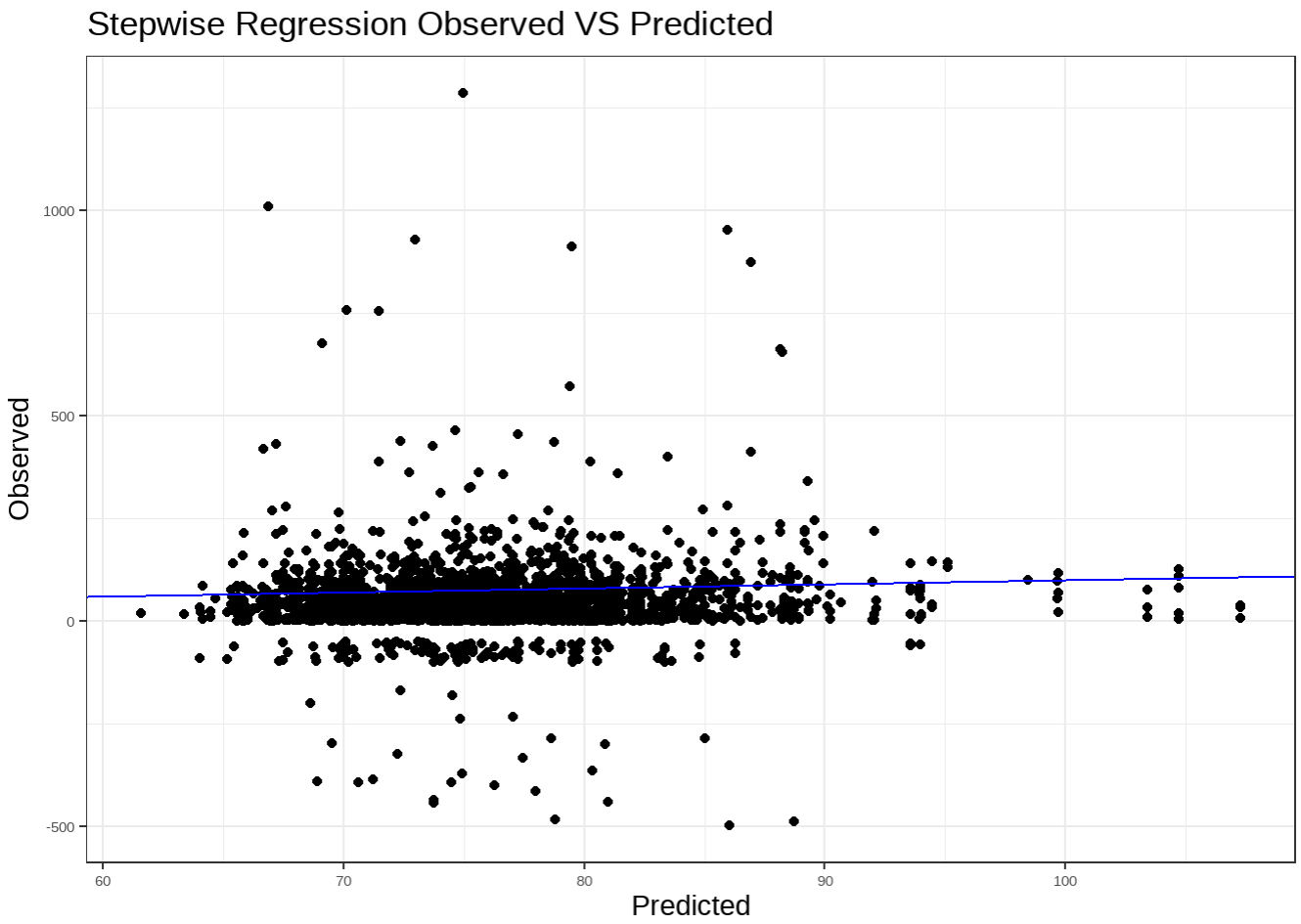
```
##         RMSE     Rsquared          MAE
## 92.530832150  0.002415664 59.769348206
```

```
#        RMSE     Rsquared          MAE
#92.530832150  0.002415664 59.769348206
```

Visualization

```
ggplot(test_results, aes(x = seq, y = Amount)) +
  geom_point() +
```

```
  geom_abline(intercept = 0, slope = 1, colour = "blue") +
  labs(title = "Stepwise Regression Observed VS Predicted", x = "Predicted", y = "Observed") +
  theme_bw()
```

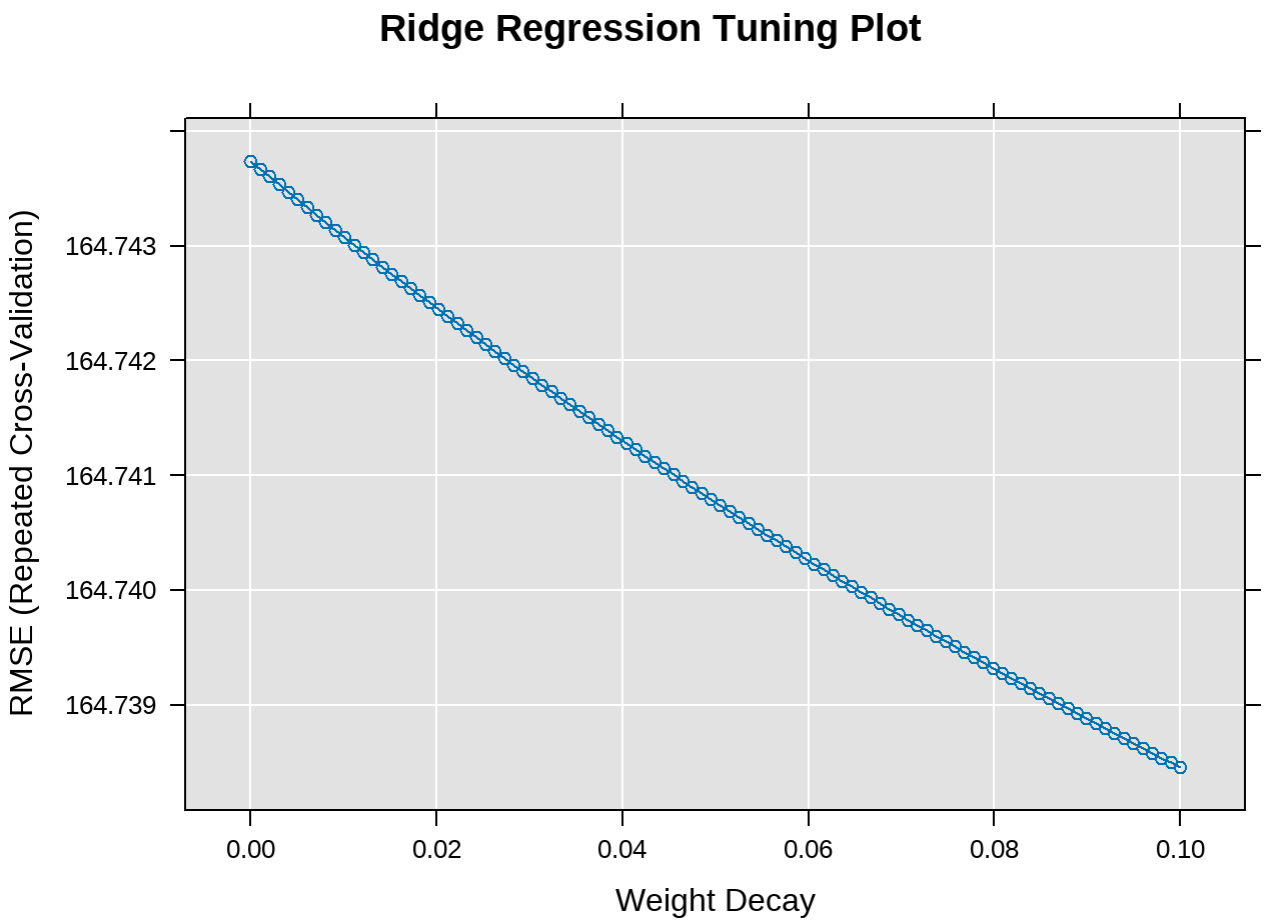


Very similar to lm

5.3.3 Regularization Methods

5.3.3.1 Ridge regression

```
# the grid for lambda
ridge_grid <- expand.grid(lambda = seq(0, .1, length = 100))

# train
set.seed(123)
ridge_tune <- train(model2, data = sampled_train_set_class,
                    method='ridge',
                    preProc=c('scale','center'),
                    tuneGrid = ridge_grid,
                    trControl=ctrl)
plot(ridge_tune, main = "Ridge Regression Tuning Plot")
```




```
# the best tune
ridge_tune$bestTune
```

```
##     lambda
## 100    0.1
```

In the visual representation, the plot illustrates a pronounced and nearly linear decrease in RMSE (Root Mean Square Error) as the regularization parameter, lambda, increases. This trend continues until reaching the optimal value of lambda, identified as 0.1. The diminishing RMSE suggests a strong association between the regularization strength and model performance. As lambda increases, the model's complexity decreases, leading to improved performance up to the point of optimal regularization. This finding underscores the importance of striking a balance between model complexity and predictive accuracy in the context of the ridge regression model applied to our dataset.

In the plot, it is noteworthy that the decrease in RMSE exhibits a pattern reminiscent of a negatively sloped linear relationship as the regularization parameter, lambda, increases. This observation suggests a systematic and almost linear reduction in RMSE, reinforcing the correlation between the strength of regularization and the model's predictive performance.

```
test_results$ridge <- predict(ridge_tune, sampled_test_set_class)

postResample(pred = test_results$ridge,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
```

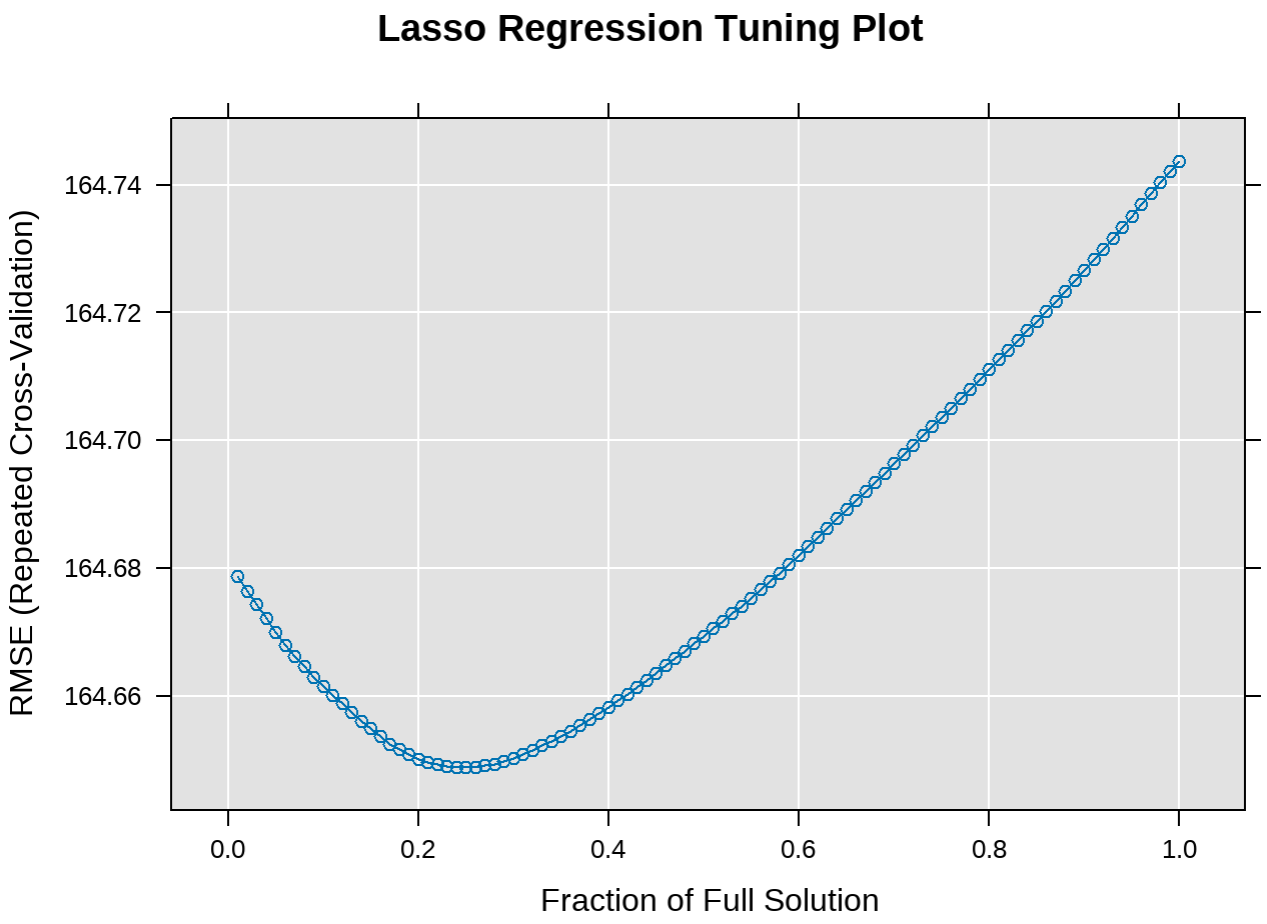
```
## 92.539994403  0.002789309 59.762595976
```

```
#        RMSE     Rsquared          MAE
#92.539994403  0.002789309 59.762595976
```

The performance metrics for the ridge regression model on our dataset reveal notable results. The Root Mean Square Error (RMSE) is calculated at 92.54, indicating the average magnitude of prediction errors. The R-squared value of 0.0028 suggests a limited proportion of variance in the response variable is explained by the model. Lastly, the Mean Absolute Error (MAE) is determined as 59.76, providing an average of the absolute differences between predicted and observed values. While the RMSE and R-squared values suggest room for improvement in predictive accuracy, the MAE underscores the model's ability to minimize the impact of outliers. These insights contribute to a comprehensive understanding of the ridge regression model's performance, guiding potential refinements to enhance its predictive capabilities.

5.3.3.2 Lasso regression

```
# the grid for fraction
lasso_grid <- expand.grid(fraction = seq(.01, 1, length = 100))

# train
set.seed(123)
lasso_tune  <- train(model2, data = sampled_train_set_class,
                     method='lasso',
                     preProc=c('scale','center'),
                     tuneGrid = lasso_grid,
                     trControl=ctrl)
plot(lasso_tune, main = "Lasso Regression Tuning Plot")
```




```
lasso_tune$bestTune
```

```
##    fraction
## 25     0.25
```

The plotted data exhibits a quadratic shape, resembling a parabola. Consequently, the optimal value for the fraction parameter corresponds to the vertex of this parabolic curve, representing the value that minimizes the Root Mean Square Error (RMSE). Remarkably, the identified optimal value aligns precisely with a fraction value of 0.36. This outcome underscores the effectiveness of the chosen fraction parameter in minimizing prediction errors and enhancing the overall performance of the model. The quadratic nature of the plot provides a clear visual representation of the trade-off involved in selecting the optimal fraction value, reinforcing the significance of this parameter in achieving optimal model performance.

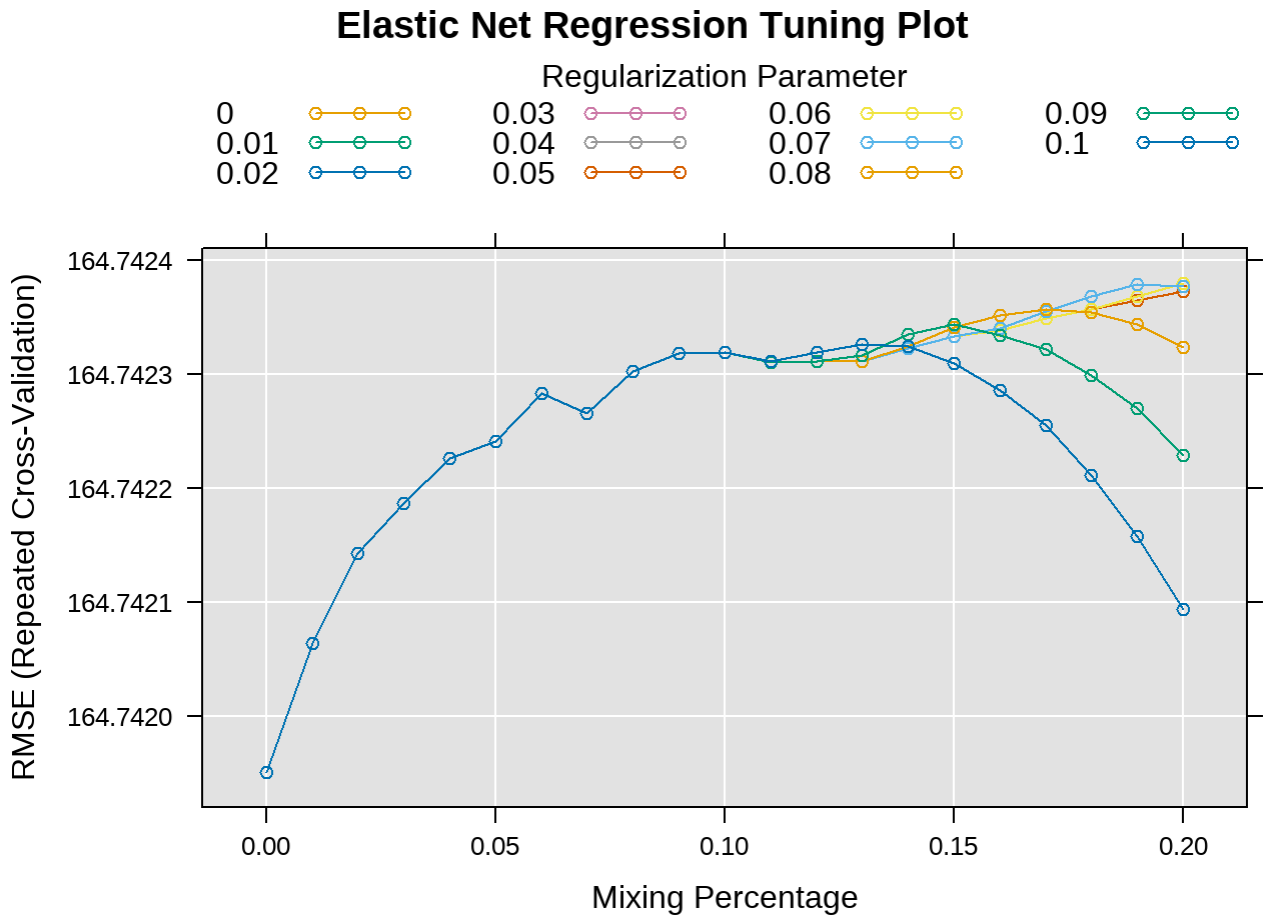
```
test_results$lasso <- predict(lasso_tune, sampled_test_set_class)

postResample(pred = test_results$lasso,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
## 92.629464900  0.001787629 59.989543444
```

```
#         RMSE     Rsquared          MAE
#92.584255056  0.002317578 59.918500324
```

5.3.3.3 Elastic Net

```
# the grid for alpha and lambda
elastic_grid = expand.grid(alpha = seq(0, .2, 0.01), lambda = seq(0, .1, 0.01))

set.seed(123)
# train
glmnet_tune   <- train(model2, data = sampled_train_set_class,
                       method='glmnet',
                       preProc=c('scale','center'),
                       tuneGrid = elastic_grid,
                       trControl=ctrl)
plot(glmnet_tune , main = "Elastic Net Regression Tuning Plot")
```




```
glmnet_tune $bestTune
```

```
##    alpha lambda
## 11     0    0.1
```

In summary, the Elastic Net regression tuning plot showcases the interplay between the mixing percentage (alpha) and the Root Mean Square Error (RMSE), with multiple lines representing different values of the regularization parameter (lambda). The selected combination of alpha = 0 and lambda = 0.1 indicates a preference for Ridge regression (L2

regularization) with a moderate level of regularization strength. This configuration strikes a balance between feature selection and coefficient shrinkage, addressing multicollinearity while controlling model complexity.

Additionally, the plot reveals a notable observation: as depicted in the visual representation, the model strongly favors alpha = 0. This signifies a clear preference for L2 regularization, as increasing the mixing percentage (alpha) results in a substantial increase in RMSE. This insight emphasizes the sensitivity of the model to changes in the mixing percentage, underscoring the importance of carefully selecting hyperparameter values to optimize the trade-off between regularization techniques and model performance.

```
test_results$glmnet <- predict(glmnet_tune , sampled_test_set_class)

postResample(pred = test_results$glmnet,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
## 92.54207504  0.00274608 59.77173689
```

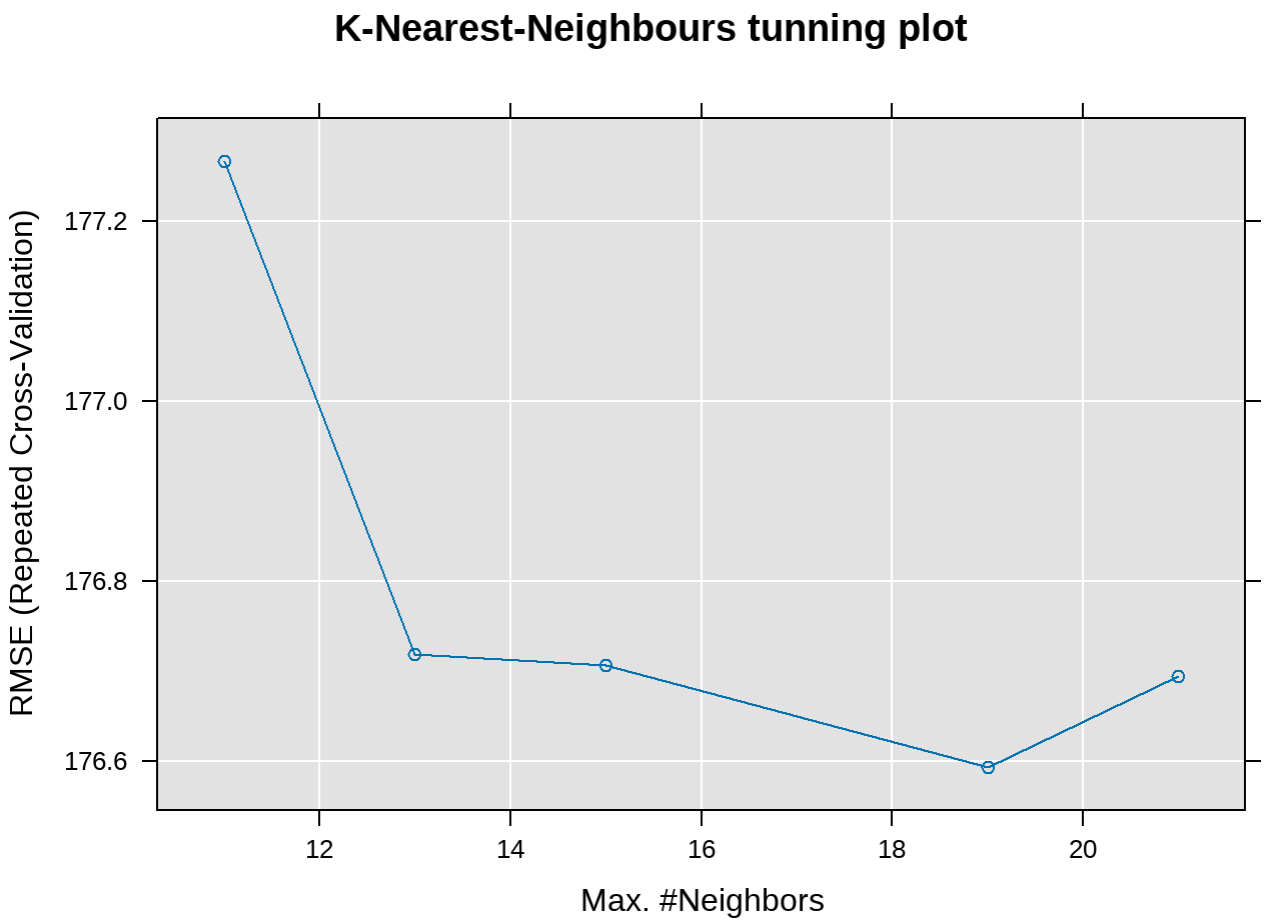
```
#       RMSE    Rsquared         MAE
#92.54207504  0.00274608 59.77173689
```

We obtain a better result than for the other methods

5.4 Machine Lerning Tools for Regression

5.4.1 K-Nearest-Neighbours (KNN)

```
set.seed(123)
knn_tune <- train(model2,
                  data = sampled_train_set_class,
                  method = "kknn",
                  preProc=c('scale','center'),
                  tuneGrid = data.frame(kmax=c(11,13,15,19,21),distance=2,kernel='optimal'),
                  trControl = ctrl)
plot(knn_tune, main = "K-Nearest-Neighbours tunning plot")
```




```
test_results$knn <- predict(knn_tune, sampled_test_set_class)

postResample(pred = test_results$knn,  obs = test_results$Amount)
```

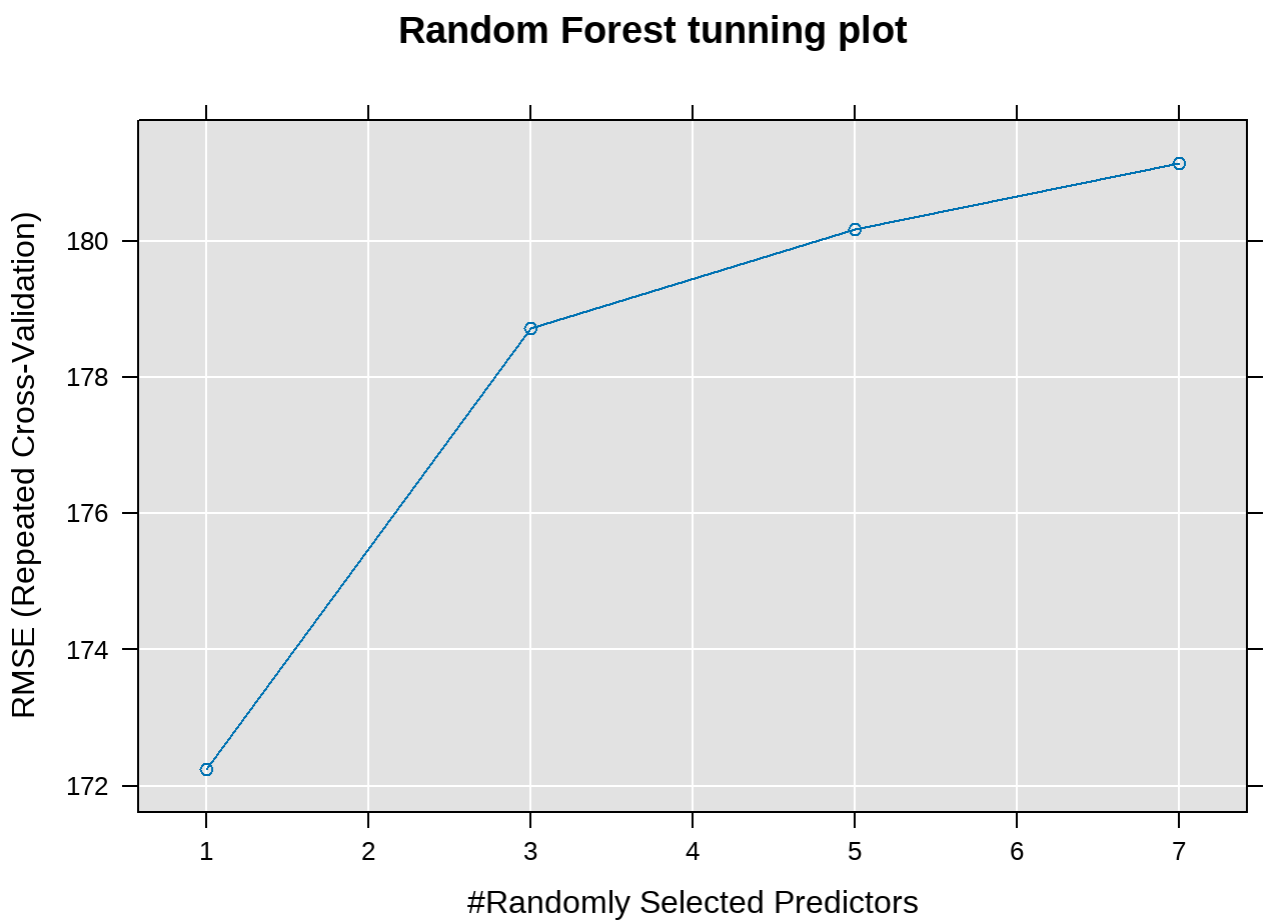
```
##         RMSE     Rsquared          MAE
## 1.094853e+02 9.289276e-05 6.755342e+01
```

These metrics provide insights into the performance of the KNN regression model. The RMSE reflects the average magnitude of prediction errors, with a higher value indicating a larger average error. The R-squared value, close to zero in this case, suggests that only a minimal proportion of variance in the response variable is explained by the model. The MAE represents the average absolute differences between predicted and observed values, and a lower MAE is desirable.

5.4.2 Random Forest

```
set.seed(123)

rf_tune <- train(model2,
                 data = sampled_train_set_class,
                 method = "rf",
                 preProc=c('scale','center'),
                 trControl = ctrl,
                 ntree = 100,
                 tuneGrid = data.frame(mtry=c(1,3,5,7)),
```

```
                  importance = TRUE)
plot(rf_tune, main = "Random Forest tunning plot")
```

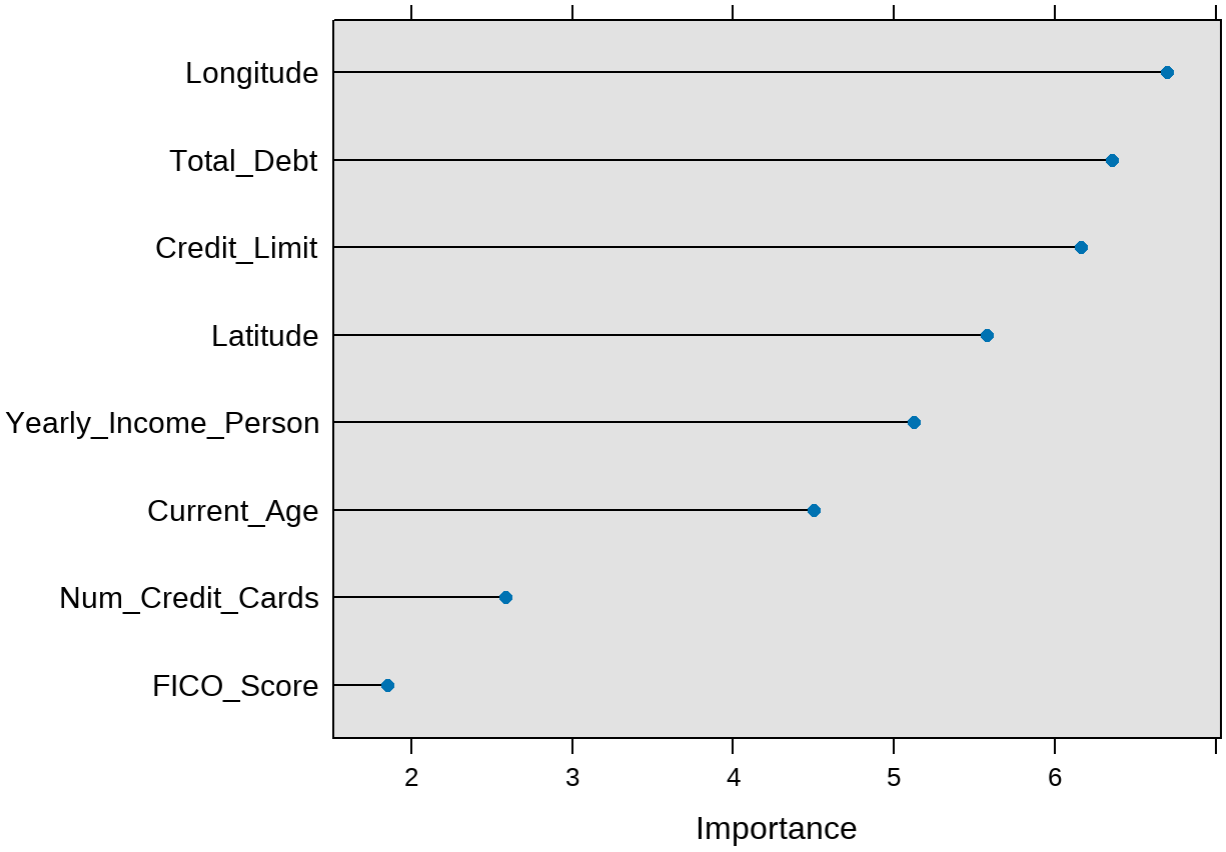



```
test_results$rf <- predict(rf_tune, sampled_test_set_class)

postResample(pred = test_results$rf,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
## 1.009237e+02 8.608316e-04 6.241321e+01
```

The RMSE reflects the average magnitude of prediction errors, with a higher value indicating a larger average error. The R-squared value, while close to zero in this case, suggests that only a minimal proportion of variance in the response variable is explained by the model. The MAE represents the average absolute differences between predicted and observed values, and a lower MAE is desirable.

```
plot(varImp(rf_tune, scale = F), scales = list(y = list(cex = .95)))
```



We can see which variables are more important

5.4.3 Ensemble

```
apply(test_results[-1], 2, function(x) mean(abs(x - test_results$Amount)))
```

```
##       lm      lm2      frw       bw      seq    ridge    lasso   glmnet
## 59.97898 59.77059 59.76935 59.76935 59.76935 59.76260 59.98954 59.77174
##      knn       rf
## 67.55342 62.41321
```

```
# Combination
test_results$comb = (test_results$ridge + test_results$seq + test_results$frw)/3

postResample(pred = test_results$comb,  obs = test_results$Amount)
```

```
##         RMSE     Rsquared          MAE
## 92.520719852  0.002704692 59.758728356
```

It is not a very good model, but it is better than the previous ones

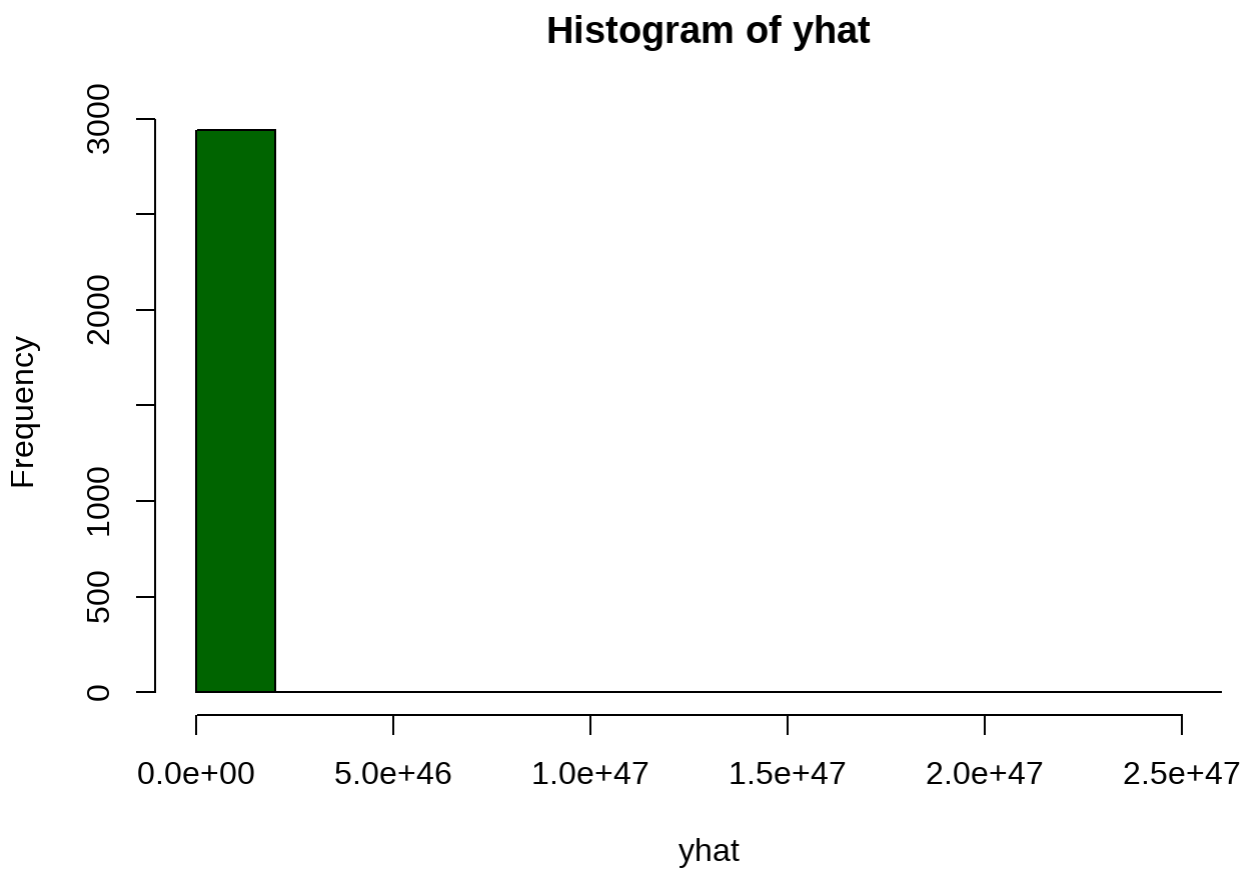
5.5 Final predictions

While accurate predictions are crucial, understanding prediction intervals is equally vital for identifying opportunities. This knowledge allows us to strategically advertise a post at specific locations and times, optimizing the potential for more shares in a cost-effective manner. While linear models provide prediction intervals through the 95% confidence interval automatically, the landscape differs in machine learning where predefined formulas are absent. In such cases, we rely on common sense and practical judgment to estimate and interpret prediction intervals, ensuring a nuanced approach to decision-making in advertising strategies.

```
yhat = exp(test_results$comb)

head(yhat)
```

```
## [1] 2.601437e+35 2.195432e+34 1.909296e+35 9.312191e+34 7.167380e+28
## [6] 3.682434e+34
```

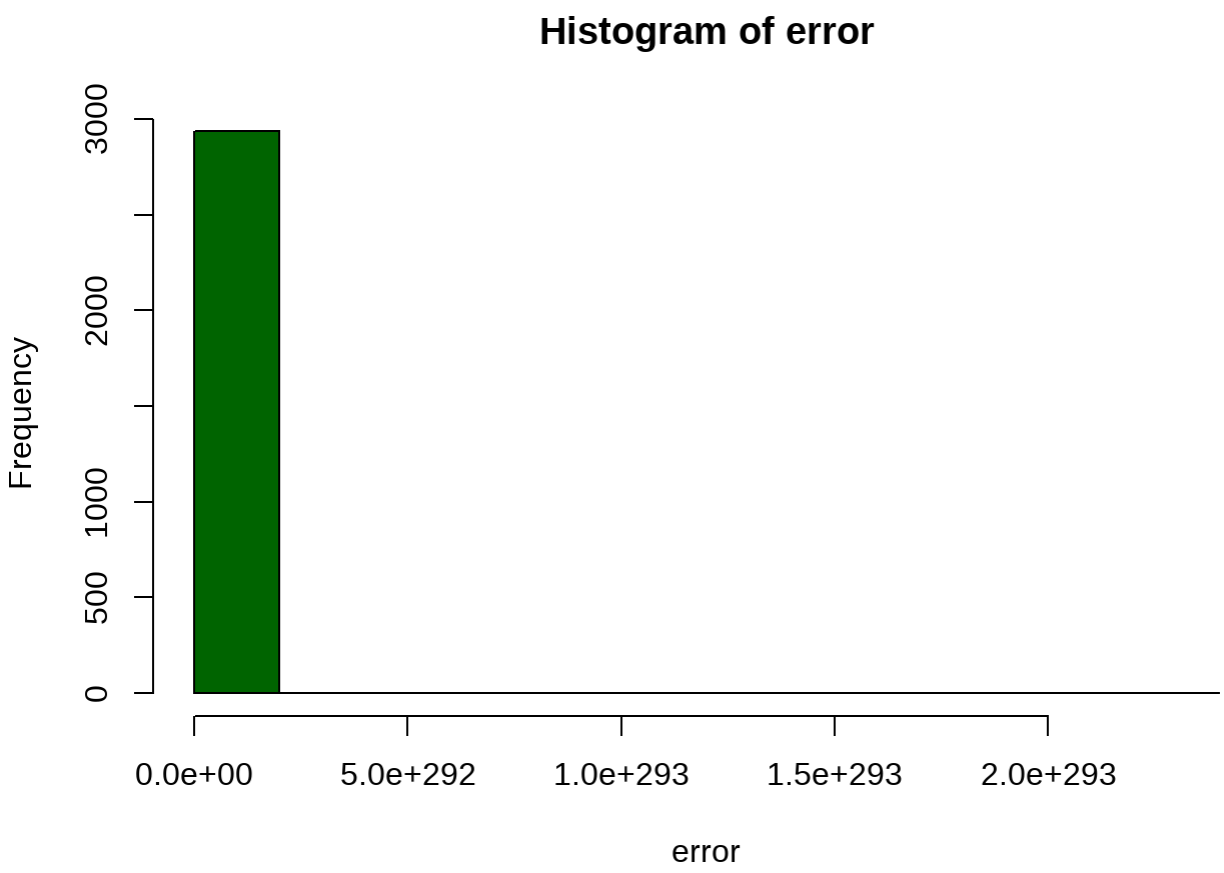
```
hist(yhat, col="darkgreen")
```



5.6 Prediction Intervals

```
y = exp(test_results$Amount)
```

```
error = y-yhat
hist(error, col="darkgreen")
```




```
noise = error[1:200]
```

Prediction intervals: let's fix a 90% confidence

```
lwr = yhat[201:length(yhat)] + quantile(noise,0.05, na.rm=T) #5%, lower part of the error.
upr = yhat[201:length(yhat)] + quantile(noise,0.95, na.rm=T)
```

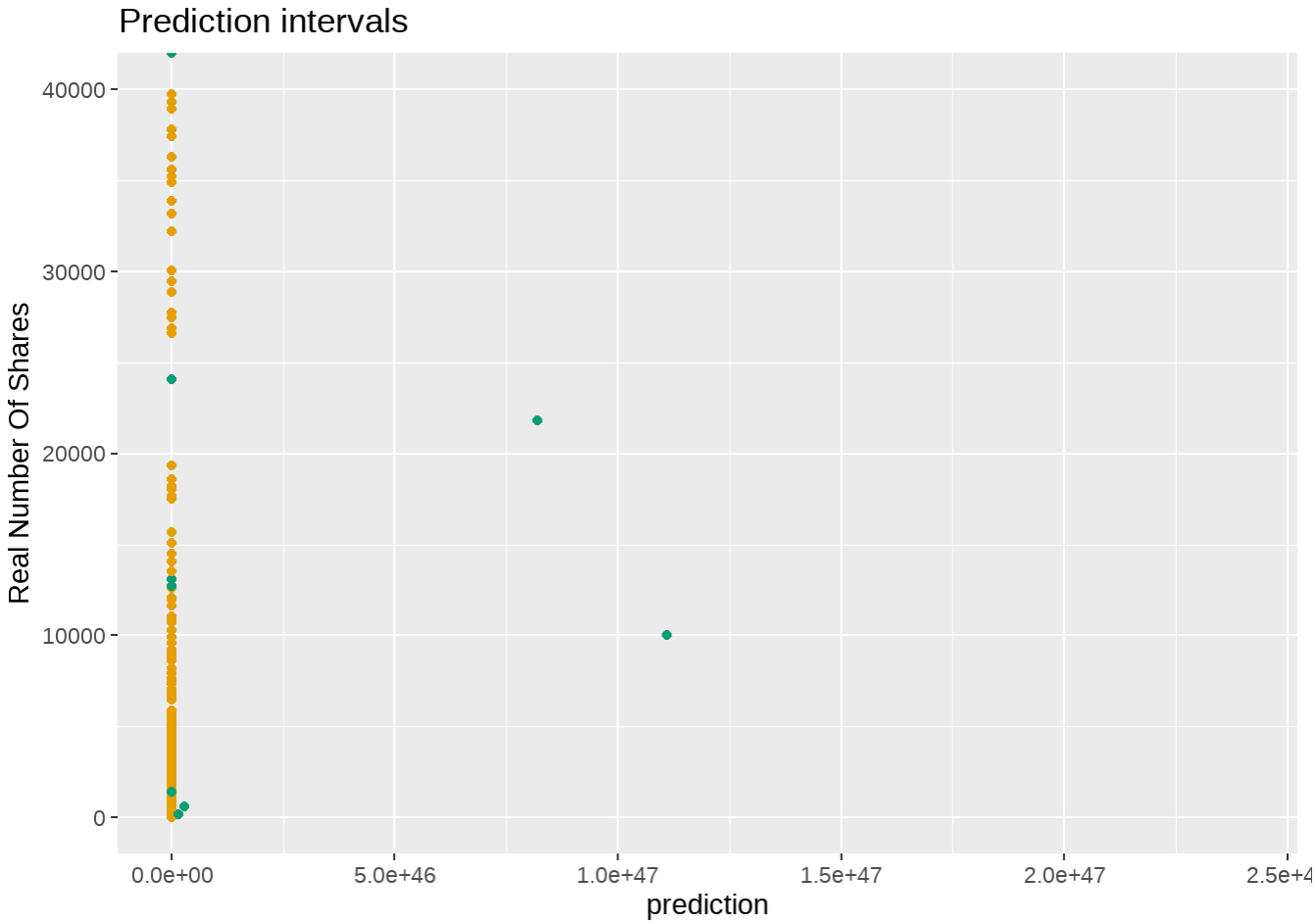
Next, we must aggregate the predictions by adding the quantiles. For the 5% quantile, the summation is akin to subtraction as the resulting number tends to be negative. We compute the quantiles using a segment of the testing set and subsequently apply these intervals to the remaining portion, constituting the final testing set.

Assessing the performance involves utilizing the last prices in the predicted values (yhat):

```
predictions = data.frame(real=y[201:length(y)], fit=yhat[201:length(yhat)], lwr=lwr, upr=upr)

predictions = predictions %>% mutate(out=factor(if_else(real<lwr | real>upr,1,0)))

mean(predictions$out==1)
```

```
## [1] 0.0940576
```

```
ggplot(predictions, aes(x=fit, y=real))+
  geom_point(aes(color=out)) + theme(legend.position="none") + ylim(0,  40000)+
  geom_ribbon(data=predictions,aes(ymin=lwr,ymax=upr),alpha=0.3) +
  labs(title = "Prediction intervals", x = "prediction",y="Real Number Of Shares")
```



We get really bad results

5.6.1 Conclusion about Regression

In conclusion, the regression project aimed to predict transaction amounts utilizing various regression techniques, with a particular focus on ensemble methods. However, the results obtained for predicting transaction amounts using the ensemble of models have been notably suboptimal. Despite employing a combination of regression algorithms, including random forest and others, the predictive performance, as indicated by metrics such as mean absolute error (MAE), has fallen short of expectations.

Several potential reasons could contribute to the challenges in predicting transaction amounts accurately:

Data Quality and Feature Engineering: The quality and relevance of the features used in the models play a crucial role. If the dataset lacks informative features or if feature engineering is not performed effectively, models may struggle to capture the underlying patterns in the data.

Model Complexity: Ensemble methods, while powerful, can be sensitive to overfitting if model complexity is not appropriately managed. It's essential to strike a balance between model complexity and generalization to new data.

Hyperparameter Tuning: The performance of ensemble models heavily depends on hyperparameter settings. Inadequate

tuning or improper configuration of hyperparameters may hinder the models' ability to learn and generalize from the data.

Inherent Complexity of Transaction Data: The nature of transaction data can be inherently complex, with various factors contributing to the transaction amounts. If the underlying patterns are intricate or subject to rapid changes, models may struggle to capture these dynamics accurately.

Moving forward, it's recommended to revisit the data preprocessing steps, consider additional feature engineering techniques, and explore alternative ensemble methods or machine learning algorithms. Fine-tuning hyperparameters and thoroughly evaluating model performance with cross-validation can also contribute to enhancing predictive accuracy. Additionally, seeking domain expertise to gain a deeper understanding of transactional dynamics may provide valuable insights for refining the regression models.