Credit card transactions fraud detection

Edyta Pietrucha & Anna Wieczorek

24.01.2022

Contents

Chapter 1

Introduction

The project will be run with R and we will be using the following libraries:

```r
main_path <- dirname(rstudioapi::getActiveDocumentContext()$path)
setwd(main_path)
library(ggplot2)
library(data.table)
library(dplyr)
library(tidyr)
library(tidyverse)
library(modelr)
library(klaR)
library(GGally)
library(randomForest)
library(gridExtra)
```

First, let us explain, what a credit card fraud is. We will call unauthorized credit card transactions as **frauds**. It is a serious problem, especially in the modern era, due to very easy to perform internet transactions. Although banks introduced two-step verification for shopping online, there are still people, who will find new ways to use someone's money.

In the project we will focus on finding a model, that suites the fraud detection best and it will be done with respect to two metrics - recall and precision.

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

We will try to maximize recall, but also try to keep precision on a reasonable level.

Chapter 2

Data description

Data input:

```r
data_train <- fread(paste0(main_path, "/Data/fraudTrain.csv"))
data_test <- fread(paste0(main_path, "Data/fraudTest.csv"))
```

The aim of this paper is to predict whether a transaction is fraudulent or not. Database which we will use is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. This was generated using Sparkov Data Generation tool.

Database is available on kaggle (see the link below):

https://www.kaggle.com/kartik2112/fraud-detection/version/1?select=fraudTrain.csv

Dataset has 1296675 observations and 23 columns.

Columns of the dataset:

- index - Unique Identifier for each row
- transdatetrans_time - Transaction DateTime
- cc_num - Credit Card Number of Customer
- merchant - Merchant Name
- category - Category of Merchant
- amt - Amount of Transaction
- first - First Name of Credit Card Holder
- last - Last Name of Credit Card Holder
- gender - Gender of Credit Card Holder
- street - Street Address of Credit Card Holder
- city - City of Credit Card Holder
- state - State of Credit Card Holder
- zip - Zip of Credit Card Holder
- lat - Latitude Location of Credit Card Holder
- long - Longitude Location of Credit Card Holder
- city_pop - Credit Card Holder's City Population
- job - Job of Credit Card Holder
- dob - Date of Birth of Credit Card Holder
- trans_num - Transaction Number
- unix_time - UNIX Time of transaction
- merch_lat - Latitude Location of Merchant
- merch_long - Longitude Location of Merchant
- is_fraud - Fraud Flag (1 - fraudulent transaction, 0 - non-fraudulent transaction)

We will predict the chances of a fraudulent transaction in function of the other variables.

Dataset contains 973 different holders and 983 different credit card numbers which means that there are Customers who have more than one credit card. We notice that each transaction number is unique.

Frequency of fraudulent transactions is approximately 0.5788652 %. The dataset has 0 missing values.

We notice that:

- X - id of observation, not helpful at all,
- first, last - name of each Holder will be individual and not helpful as such,
- street, zip - we have a location parameters so we do not use this variables,
- unix_time - we have transaction time which gives the same information,
- trans_num - is individual for each transaction and not helpful as such.

Therefore, we will remove those columns.

```
data_train <- data_train %>%
  .[,-c("V1", "first", "last", "street", "zip", "unix_time", "trans_num")]
```

Chapter 3

Data Wrangling

First, we will look at the structure of our data, to check if the datatypes are correct.

```r
data_train %>%
  str
```

```
## Classes 'data.table' and 'data.frame':   1296675 obs. of  16 variables:
##  $ trans_date_trans_time: POSIXct, format: "2019-01-01 00:00:18" "2019-01-01 00:00:44" ...
##  $ cc_num               :integer64 2703186189652095 630423337322 38859492057661 3534093764340240 375534
##  $ merchant             : chr  "fraud_Rippin, Kub and Mann" "fraud_Heller, Gutmann and Zieme" "fraud_Li
##  $ category             : chr  "misc_net" "grocery_pos" "entertainment" "gas_transport" ...
##  $ amt                  : num  4.97 107.23 220.11 45 41.96 ...
##  $ gender               : chr  "F" "F" "M" "M" ...
##  $ city                 : chr  "Moravian Falls" "Orient" "Malad City" "Boulder" ...
##  $ state                : chr  "NC" "WA" "ID" "MT" ...
##  $ lat                  : num  36.1 48.9 42.2 46.2 38.4 ...
##  $ long                 : num  -81.2 -118.2 -112.3 -112.1 -79.5 ...
##  $ city_pop             : int  3495 149 4154 1939 99 2158 2691 6018 1472 151785 ...
##  $ job                  : chr  "Psychologist, counselling" "Special educational needs teacher" "Nature
##  $ dob                  : IDate, format: "1988-03-09" "1978-06-21" ...
##  $ merch_lat            : num  36 49.2 43.2 47 38.7 ...
##  $ merch_long           : num  -82 -118.2 -112.2 -112.6 -78.6 ...
##  $ is_fraud             : int  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can see, that we will need to convert some variables to factors.

```r
dt <- c("cc_num", "merchant", "category", "gender", "city", "state", "job", "is_fraud")
data_train[,(dt):= lapply(.SD, as.factor), .SDcols = dt]
```

We need to convert `trans_date_trans_time` and `dob` to date class for further analysis.

```r
data_train[, dob := as.Date(dob)]
data_train[, trans_date_trans_time := strptime(trans_date_trans_time,
                                               format ="%Y-%m-%d %H:%M:%OS",
                                               tz = "EST")]
```

Now, let us look at the data structure again:

```
data_train %>%
  str
```

```
## Classes 'data.table' and 'data.frame':   1296675 obs. of  16 variables:
##  $ trans_date_trans_time: POSIXct, format: "2019-01-01 00:00:18" "2019-01-01 00:00:44" ...
##  $ cc_num               : Factor w/ 983 levels "60416207185",..: 445 43 238 510 369 755 158 821 777 454
##  $ merchant             : Factor w/ 693 levels "fraud_Abbott-Rogahn",..: 516 244 390 364 298 608 534 10
##  $ category             : Factor w/ 14 levels "entertainment",..: 9 5 1 3 10 3 4 3 10 5 ...
##  $ amt                  : num  4.97 107.23 220.11 45 41.96 ...
##  $ gender               : Factor w/ 2 levels "F","M": 1 1 2 2 2 1 1 2 1 1 ...
##  $ city                 : Factor w/ 894 levels "Achille","Acworth",..: 527 613 469 85 217 224 352 237 4
##  $ state                : Factor w/ 51 levels "AK","AL","AR",..: 28 48 14 27 46 39 17 46 39 43 ...
##  $ lat                  : num  36.1 48.9 42.2 46.2 38.4 ...
##  $ long                 : num  -81.2 -118.2 -112.3 -112.1 -79.5 ...
##  $ city_pop             : int  3495 149 4154 1939 99 2158 2691 6018 1472 151785 ...
##  $ job                  : Factor w/ 494 levels "Academic librarian",..: 371 429 308 329 117 480 30 128
##  $ dob                  : Date, format: "1988-03-09" "1978-06-21" ...
##  $ merch_lat            : num  36 49.2 43.2 47 38.7 ...
##  $ merch_long           : num  -82 -118.2 -112.2 -112.6 -78.6 ...
##  $ is_fraud             : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

And the types are correct now.

3.1 Explanatory Data Analysis

3.1.1 Age

We created a new variable which counts age of each Customer in a day of a transaction. We assume that age of a Holder can have an impact on detecting fraudulent transactions, especially combined with variables such as amount of money spent on a transaction or transaction time.

```
data_train[, age := round(as.numeric(difftime(trans_date_trans_time, dob, units = "days")/365), 0)]
```

To visualize the data we define one figures style:

```
# Custom figures style
text_size <- 12
# customize my_style settings
my_style <-  theme_light() +
  theme(
    legend.position = 'none',
    plot.subtitle = element_text(size = text_size - 4, colour='black'),
    plot.title = element_text(size = text_size, face = 'bold'),
    axis.text=element_text(size=text_size - 4),
    axis.title.y = element_text(size = text_size - 4),
    axis.title.x = element_text(size = text_size - 4),
    legend.background = element_rect(fill = "white")
  )
# define my palette colors
my_palette <- c("#D73027", "#FDAE61", "#FFFFBF",
                         "#A50026", '#ABD9E9', '#4575B4',
                         '#313695', 'black', "lawngreen")
```

```r
ggplot(data_train, aes(x = is_fraud, y = age)) +
  geom_boxplot() +
  my_style +
  labs(title = "Boxplot for age of Customer",
       subtitle="group by not-fraudulent transactions and fraudulent transactions")
```

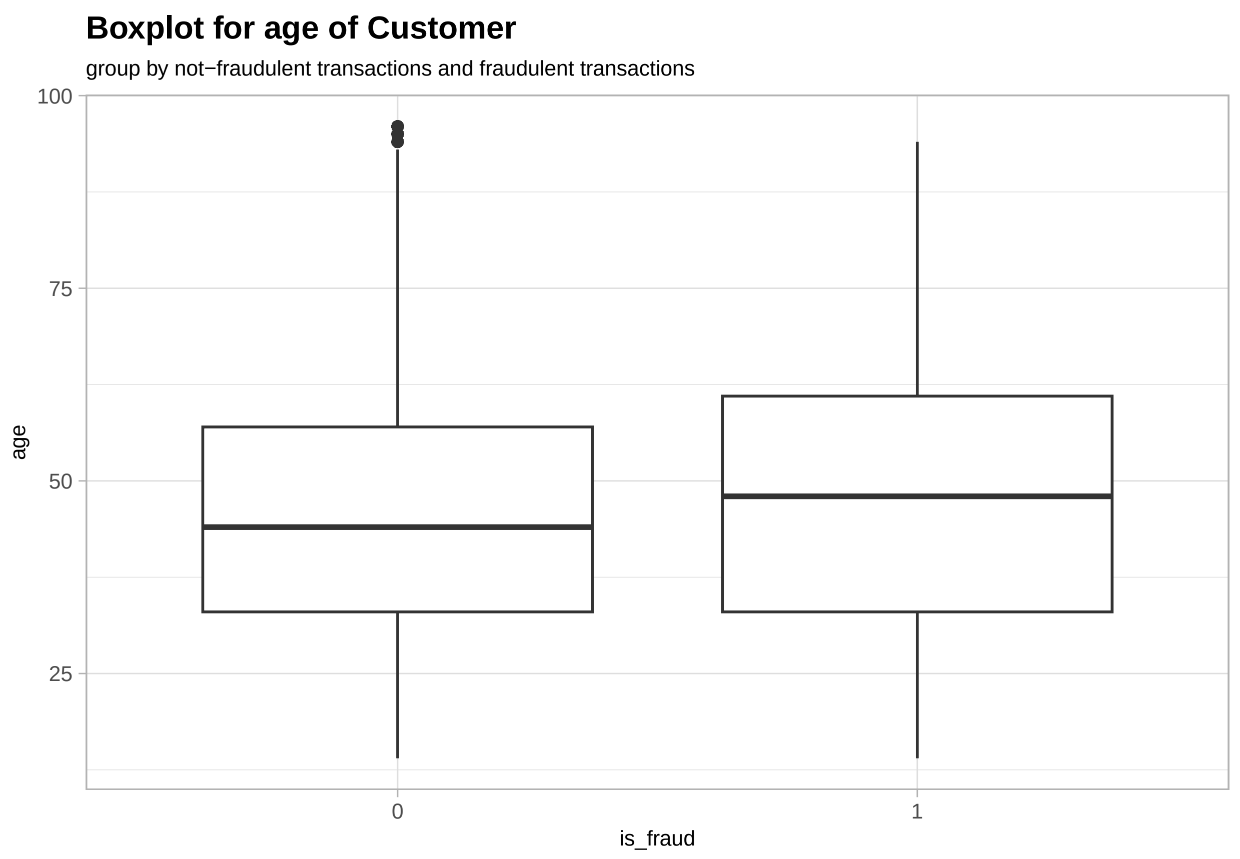


Figure ?? shows that mean for fraudulent transactions is a little bit higher than for not-fraudulent transactions.

3.1.2 Gender

Further analysis has led us to this conclusion that even though there are less men than women in our data, the number of frauds for men is slightly higher than for women.

```r
# checking a percentage of fraud transaction group by gender
gender_fraud <- data_train %>%
  .[, .(count = .N), by = c("gender", "is_fraud")] %>%
  .[, .(is_fraud, freq = count/sum(count)), by = gender] %>%
  .[is_fraud == 1]
gender_fraud
```

```
##    gender is_fraud        freq
## 1:      F        1 0.005261579
## 2:      M        1 0.006426249
```

Visualize the data:

```r
ggplot(gender_fraud, aes(x = gender, y = freq, fill = gender)) +
  geom_bar(stat = 'identity') +
  my_style +
  scale_fill_manual(values = c(my_palette[c(4,7)])) +
  labs(title = "Frequency of fraud group by gender",
       x = "",
       y = "")
```

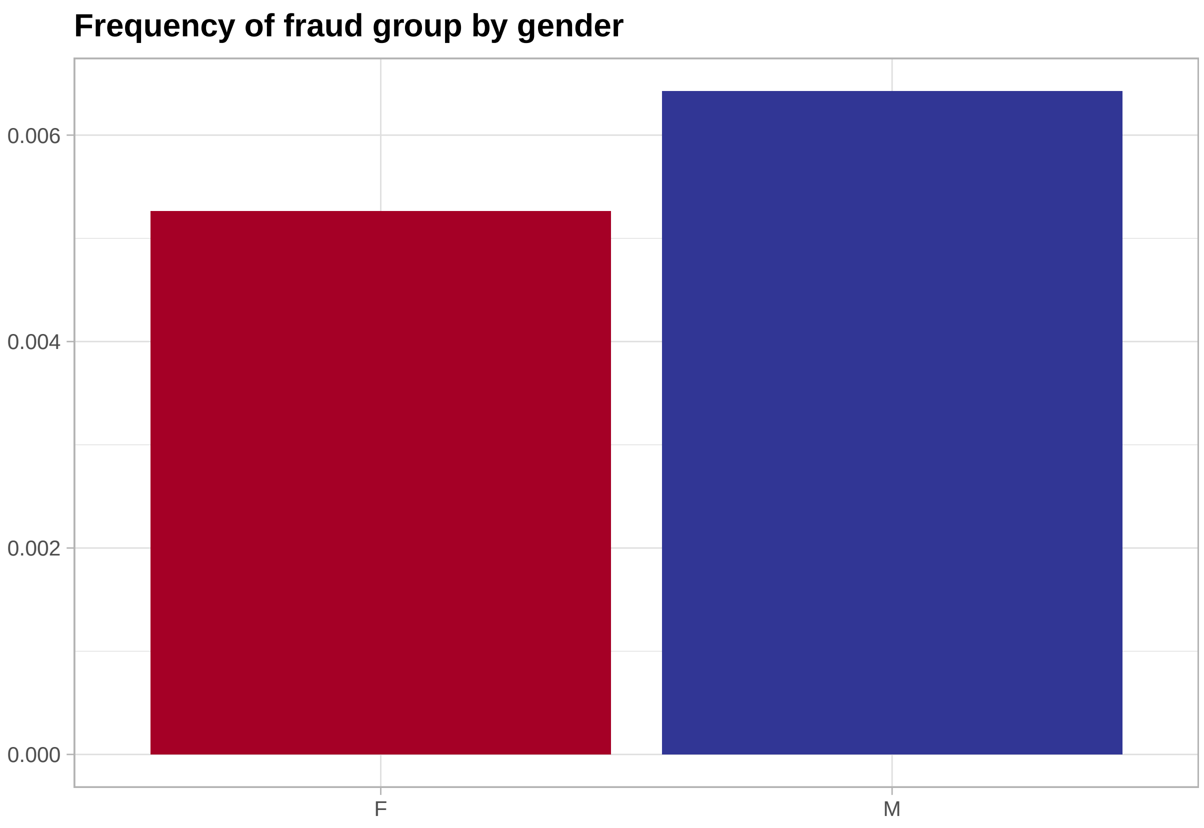


Figure 3.1: Frequency of fraudulent transactions group by gender.

3.1.3 Amt

Variable `amt` corresponds to the amount of money spent on each transaction. We assume that this information can have a huge impact to detect if a transaction is fraudulent. Let's see basic statistics for `amt`.

Mean and median amount of fraudulent transactions are higher than for not-fraudulent transactions.

We would like to create a boxplot to visualize datapoints, but we detected to much outliers in observations for not-fraudlent transactions. To deal with that problem we created our own function which removes outliers from a dataset.

```r
remove_outlier <- function(data, column) {

  # IQR= Q3 (75% quantile) - Q1 (25% quantile)
  IQR <- quantile(data[, get(column)], 0.75) - quantile(data[, get(column)], 0.25)
  # Outlier would be a point below [Q1- (1.5)IQR] or above [Q3+(1.5)IQR]
  l <- quantile(data[, get(column)], 0.25) - 1.5*IQR
  r <- quantile(data[, get(column)], 0.75) + 1.5*IQR

  data <- data %>%
    .[get(column) >= l & get(column) <= r]

  return(data)

}
```

```
ggplot(remove_outlier(data_train, "amt"),
       aes(x = is_fraud, y = amt)) +
  geom_boxplot() +
  my_style +
  labs(title = "Boxplot for amount spent on the transaction",
       subtitle = "group by not-fraudulent transactions and fraudulent transactions")
```

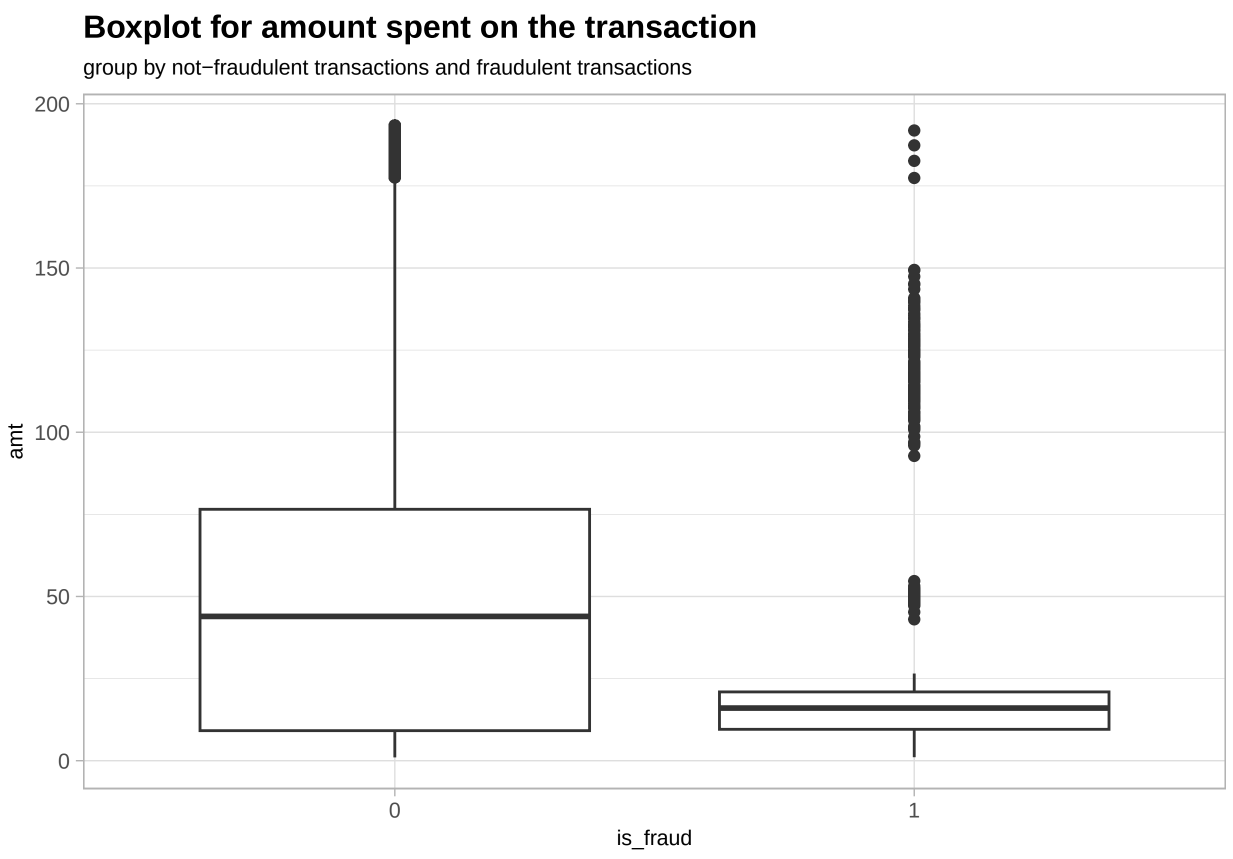


Figure 3.2: Boxplot for variable amt group by not-fraudulent transactions and fraudulent transactions.

Graphic analysis confirms our assumptions that the variable `amt` can be really important for aim of our project.

3.1.4 Weekday of transaction

We think, that the weekday of a transaction could be an important indicator of fraudulent transactions, especially when combined with other variables like the amount of money spent on purchase or the transaction hour. So we decided to create a variable `week_day_trans` to conduct the analysis.

```
data_train[, week_day_of_trans := as.factor(wday(trans_date_trans_time))]
```

```
week_day_fraud <- data_train %>%
  .[, .(count = .N), by = c("week_day_of_trans", "is_fraud")] %>%
  .[, .(is_fraud, freq = count/sum(count)), by = week_day_of_trans] %>%
  .[is_fraud == 1]
```

Once we have created the variable, we can look at the frequency of frauds on each day.

```
ggplot(week_day_fraud, aes(x = week_day_of_trans, y = freq, fill = week_day_of_trans)) +
  geom_bar(stat = 'identity') +
  my_style +
  scale_fill_manual(values = c(my_palette[1:7])) +
  labs(title = "Frequency of fraud group by weekday of the transaction",
       x = "",
       y = "") +
  scale_x_discrete(labels=c("Monday", "Tuesday", "Wednesday", "Thursday",
                            "Friday", "Saturday", "Sunday"))
```

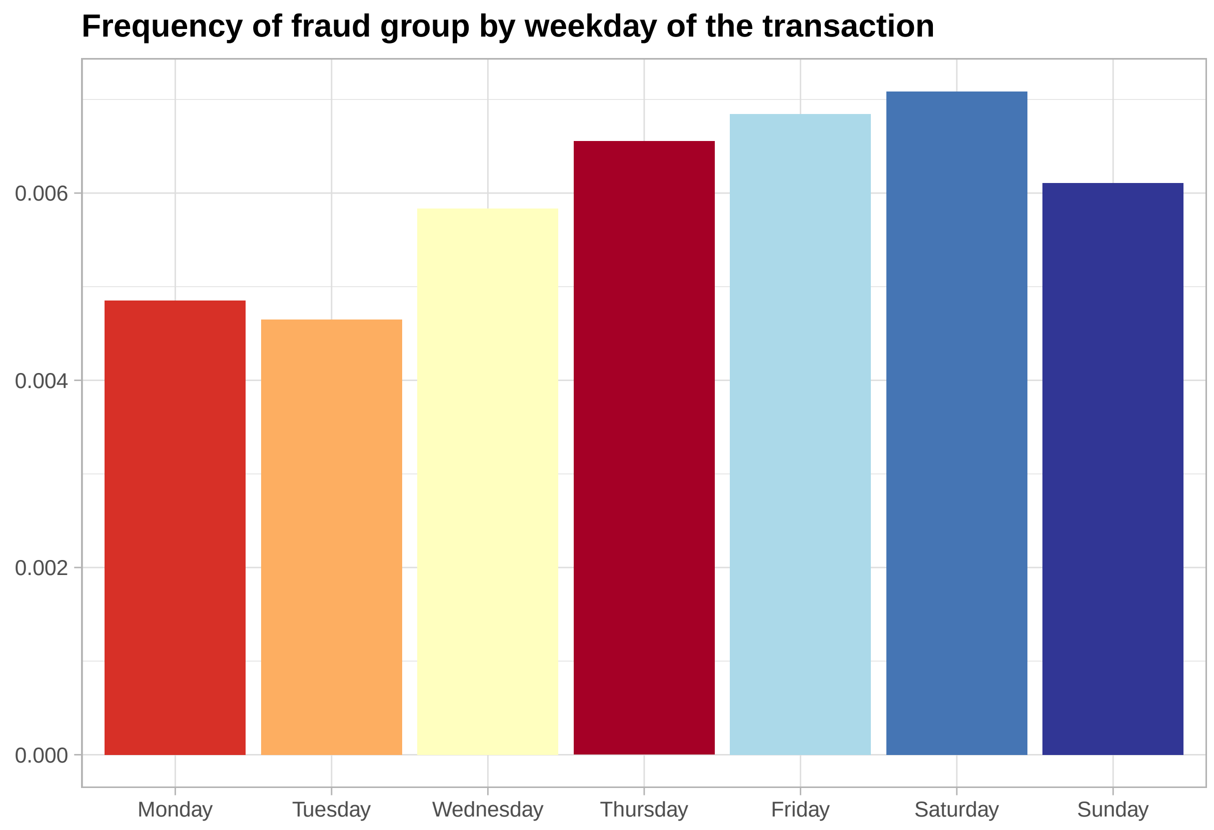


Figure 3.3: Frequancy of fraudulent transactions grouped by the weekday.

Based on Figure 3.3 on Mondays and Tuesdays the odds of a fraudulent transactions are slightly lower than on the other days.

3.1.5 Category of Purchase

Investigating the data, we see that some categories, that is: *grocery_pos*, *misc_net*, *shopping_net* and *shopping_pos*, seem to be more vulnerable to fraudulent transactions than others.

```
category_fraud <- data_train %>%
  .[, .(count = .N), by = c("category", "is_fraud")] %>%
  .[, .(is_fraud, freq = count/sum(count)), by = category] %>%
  .[is_fraud == 1]
```

```
ggplot(category_fraud, aes(x = category, y = freq, fill = category)) +
  geom_bar(stat = 'identity') +
  my_style +
  labs(title = "Frequency of fraud group by category of merchant",
       x = "",
       y = "") +
  coord_flip()
```

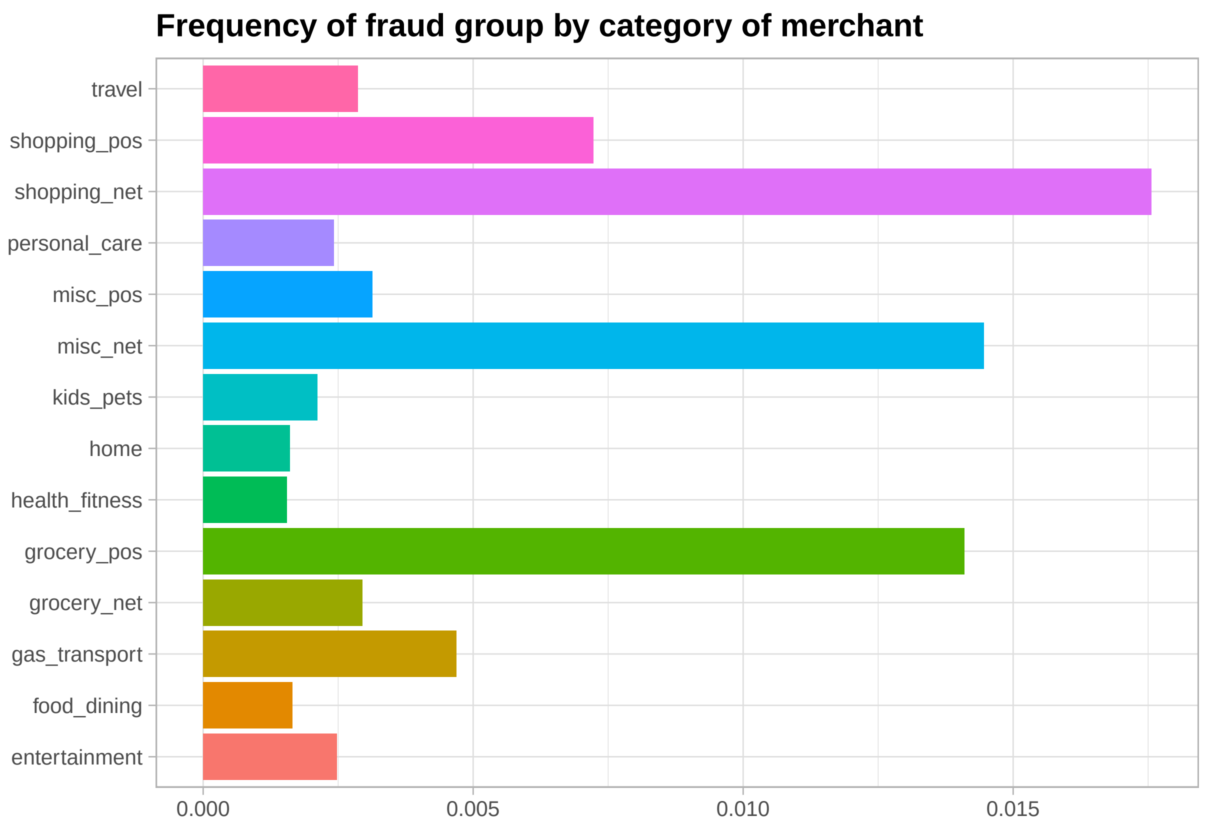


Figure 3.4: Frequancy of fraudulent transactions grouped by the category of purchase.

We can see this phenomenon on the Figure 3.4.

3.1.6 Transaction date, transaction time

Let us dive into transaction date and time in more details. First, we will create a variable `trans_hour` and see if at any time of the day, such as late night, chance of fraud is higher.

```r
data_train[, trans_hour := as_factor(strftime(trans_date_trans_time,
                                              format="%H",
                                              tz = "EST"))]
```

```r
hour_fraud <- data_train %>%
  .[, .(count = .N), by = c("trans_hour", "is_fraud")] %>%
  .[, .(is_fraud, freq = count/sum(count)), by = trans_hour] %>%
  .[is_fraud == 1]
```

As expected, at night the frequency of frauds is much higher than during a day. This insight, combined with the age of a Holder can be very important for our model. For example, it is much more likely that a younger person will be buying something at night, than an elderly one.

```r
ggplot(hour_fraud, aes(x = trans_hour, y = freq, fill = trans_hour)) +
  geom_bar(stat = 'identity') +
  my_style +
  labs(title = "Frequency of fraud group by transaction hour",
       x = "",
       y = "") +
  coord_flip()
```

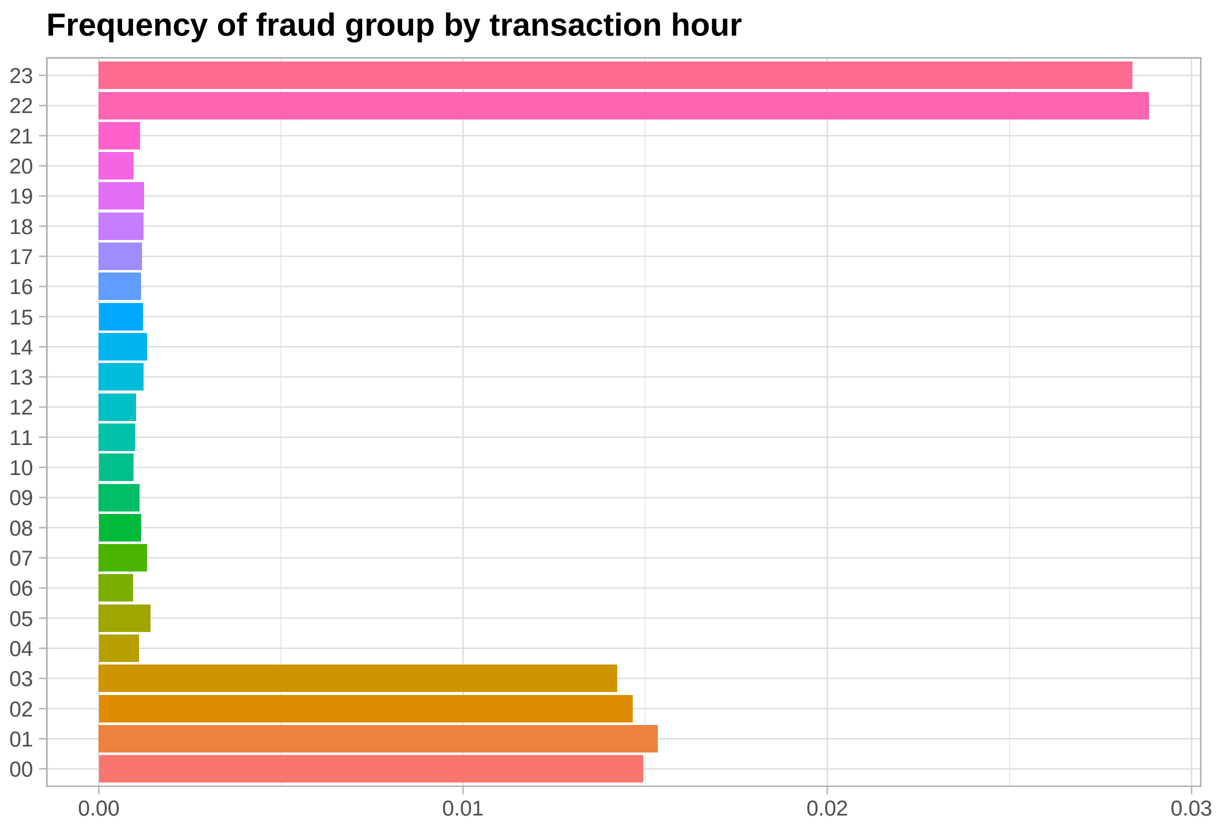


Figure 3.5: Frequancy of fraudulent transactions grouped by the transaction hour.

The analysis also shown, that month of a transaction is not very important.

3.1.7 Distance

Now we will take a look at the distance between the home of a Holder and the place of a Merchant. We will compute the distance based on `lat`, `long`, `merch_lat` and `merch_long` variables, that give us geo-coordinates for the places, using the formula:

$$distance = \sqrt{(long - merch\ long)^2 + (lat - merch\ lat)^2}$$

```r
data_train[, dist := sqrt((long-merch_long)^2 + (lat-merch_lat)^2)]
```

```r
ggplot(data_train, aes(x = is_fraud, y = dist)) +
  geom_boxplot() +
  my_style +
  theme(plot.title = element_text(size = text_size - 1, face = 'bold')) +
  labs(title = "Boxplot for distance between the home of a Holder and the place of a Merchant",
       subtitle = "group by not-fraudulent transactions and fraudulent transactions")
```

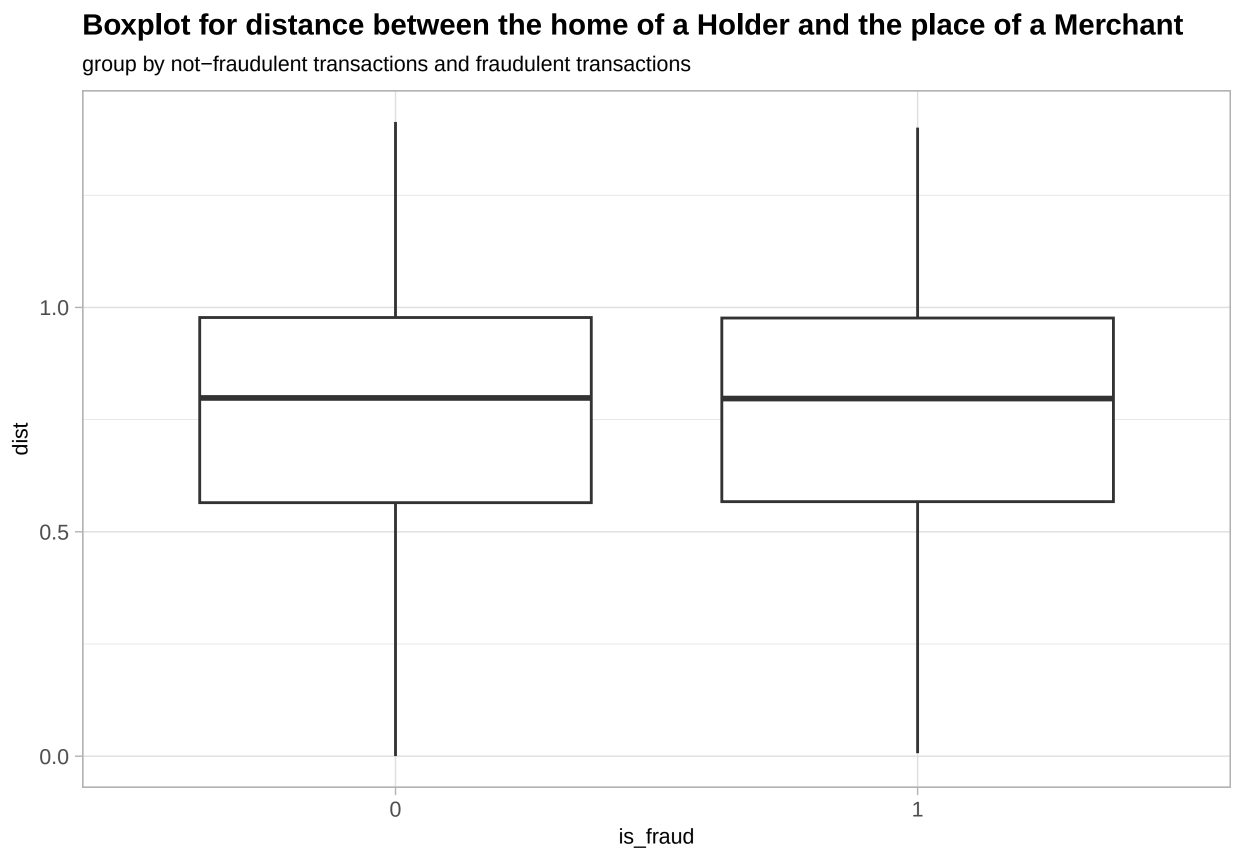


Figure 3.6: Boxplot of distance between the home of a Holder and the place of a Merchant, grouped by is_fraud.

The boxplot on Figure 3.6 shows no significant difference in distance for fraudulent and non-fraudulent transactions.

3.2 Data binning

Keeping in mind the analysis described above, we will bin some categorical variables, so that we have less categories (and less variables in a model), but still have much information, that the columns provide.

3.2.1 Information Value

We will perform binning with respect to Information Value and Weight of Evidence provided by function `WOETable` form a package `InformationValue` (code is based on the lecture materials).

```
categorical_cols <- c()
for (col in colnames(data_train %>% .[, -c("cc_num", "is_fraud")])) {

  if(is.factor(data_train[[col]])) {
    categorical_cols <- c(categorical_cols, col)
  }

}

woe_table <- woe(data_train[, ..categorical_cols],
                 as.factor(data_train$is_fraud),
                 zeroadj = 1)
```

```
## At least one empty cell (class x level) does exists. Zero adjustment applied!
## At least one empty cell (class x level) does exists. Zero adjustment applied!
## At least one empty cell (class x level) does exists. Zero adjustment applied!
## At least one empty cell (class x level) does exists. Zero adjustment applied!
```

```
woe_table
```

```
## Information values of transformed variables:
##
##                           IV
## trans_hour        1.86222611
## city              1.35711722
## merchant          0.86570833
## category          0.75817840
## job               0.69713416
## state             0.04938307
## week_day_of_trans 0.02518155
## gender            0.01008412
```

`job`, `merchant`, `city` and `category` seem to have high value, but they also have many categories (leading to higher sum of IV). `gender` performs poorly and that is a little bit surprising, giving the difference in fraud/non-fraud proportions.

Taking a look at the separate WOE tables for each variable, we can conclude that particular jobs have low IV, the same can be observed for `city`. For `category` the table looks better, we will work with this variable and with `trans_hour`. For `state`, `week_day_trans` and `gender` the results are rather poor, although we will try to work on the weekday of transaction.

Now, we will apply the results combined with analysis of the plots shown earlier and bin some variables.

```
data_train[, weekday_fraud := if_else(week_day_of_trans %in% 4:7, 1, 0)]
data_train[, category_fraud :=
             if_else(category %in% c('grocery_pos','misc_net','shopping_net','shopping_pos'),1,0)]
data_train[, hour_fraud :=
             if_else(trans_hour %in% c('00', '01', '02', '03', '22', '23'), 1, 0)]
data_train[,c("weekday_fraud", "category_fraud", "hour_fraud"):=
             lapply(.SD, as.factor), .SDcols = c("weekday_fraud", "category_fraud", "hour_fraud")]
```

We can also drop more variables (for binned and unbinned data):

- lat, long, merch_lat, merch_long - beacause they were used to create **`dist`** and we do not need them anymore,
- cc_num - we will not use credit card number,
- merchant - there are too many categories (693),
- dob - date of birth of a Holder will not be used, as we have created the age variable,
- trans_date_trans_time - was needed only to create **`trans_hour`**.

Also, we will add binned variables to **`data`**, because we will perform normalization and our data is too big to keep all three datasets with original, binned and normalized data.

```
data_train <- data_train %>%
  .[,-c('lat','long','merch_lat','merch_long','cc_num','merchant', 'dob', 'trans_date_trans_time')]
```

3.3 Normalization

As mentioned before, we will not create another dataset with unbinned and normalized data, because it is too big and takes too much memory to keep it. So let us normalize **`data_bin`** and leave **`data`** in its original ranges.

To perform normalization, we will use a function **`min_max_norm`** from lecture materials.

```
min_max_norm <- function(x) {
  (x - min(x, na.rm = T)) / (max(x, na.rm = T) - min(x, na.rm = T))
}

temp <- c('amt','city_pop','age','dist')
data_train_bin <- copy(data_train)
data_train_bin[, (temp) := lapply(.SD, min_max_norm), .SDcols = temp] # reference!
```

3.4 Correlation

```
ggcorr(data_train[, ..temp], label = T)
```

Figure 3.7 shows that correlation is very low for all numerical variables.

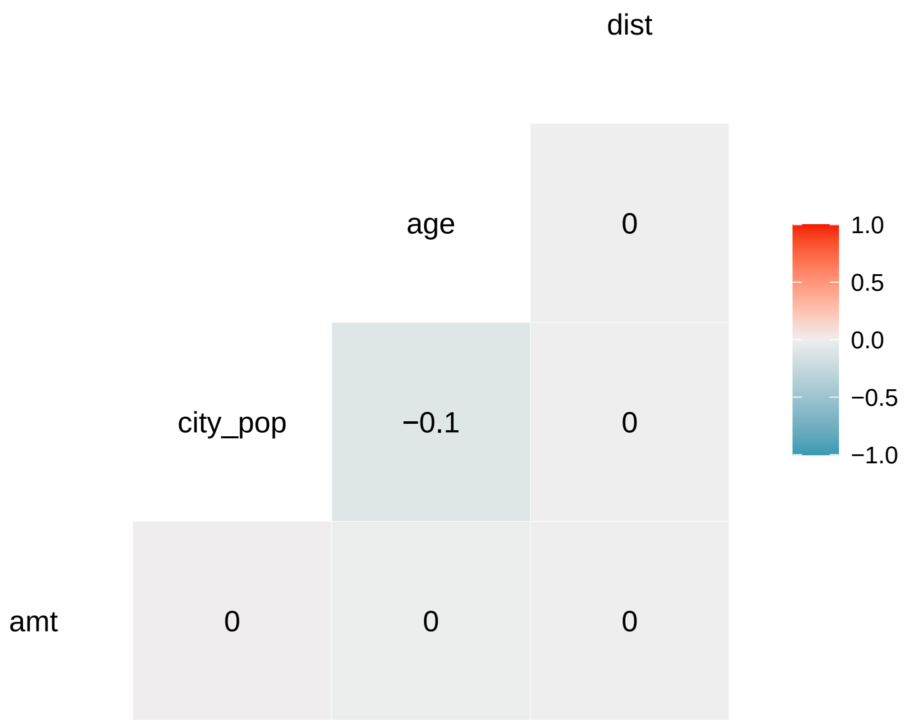


Figure 3.7: The correlation structure of the data.

Chapter 4

Models

4.1 Cross Validation

To investigate how well the model is dealing with new datapoints we set 20% of the data aside for cross validation. We make it on both datasets with normalized data and not-normalized data.

```
set.seed(125)
data_train_bin_cv <- resample_partition(data_train_bin, p = c(valid = 0.3, train = 0.7))
train_bin_cv <- data_train_bin[data_train_bin_cv$train$idx,]
valid_bin_cv <- data_train_bin[data_train_bin_cv$valid$idx,]
```

4.2 The base model

4.2.1 Logistic regression

As a base model we will consider logistic regression with variables which give the best results after investigation.

```
logreg <- glm(is_fraud ~
                category_fraud + amt + hour_fraud + age:hour_fraud + amt:hour_fraud +
                amt:category_fraud + age:category_fraud,
              data = train_bin_cv, family = binomial)
summary(logreg)
```

```
##
## Call:
## glm(formula = is_fraud ~ category_fraud + amt + hour_fraud +
##     age:hour_fraud + amt:hour_fraud + amt:category_fraud + age:category_fraud,
##     family = binomial, data = train_bin_cv)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904  -0.0711  -0.0443  -0.0392   4.4390
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -7.26220    0.08810 -82.427  < 2e-16 ***
## category_fraud1      -0.41384    0.07493  -5.523 3.33e-08 ***
## amt                 -85.53658    3.32968 -25.689  < 2e-16 ***
## hour_fraud1           2.67919    0.09282  28.864  < 2e-16 ***
## hour_fraud0:age       0.85232    0.17545   4.858 1.19e-06 ***
## hour_fraud1:age      -0.42094    0.12624  -3.334 0.000855 ***
## amt:hour_fraud1     107.92782    2.37217  45.498  < 2e-16 ***
## category_fraud1:amt 146.98121    2.98194  49.290  < 2e-16 ***
## category_fraud1:age   1.26192    0.15201   8.301  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 64850  on 907672  degrees of freedom
## Residual deviance: 41229  on 907664  degrees of freedom
## AIC: 41247
##
## Number of Fisher Scoring iterations: 9
```

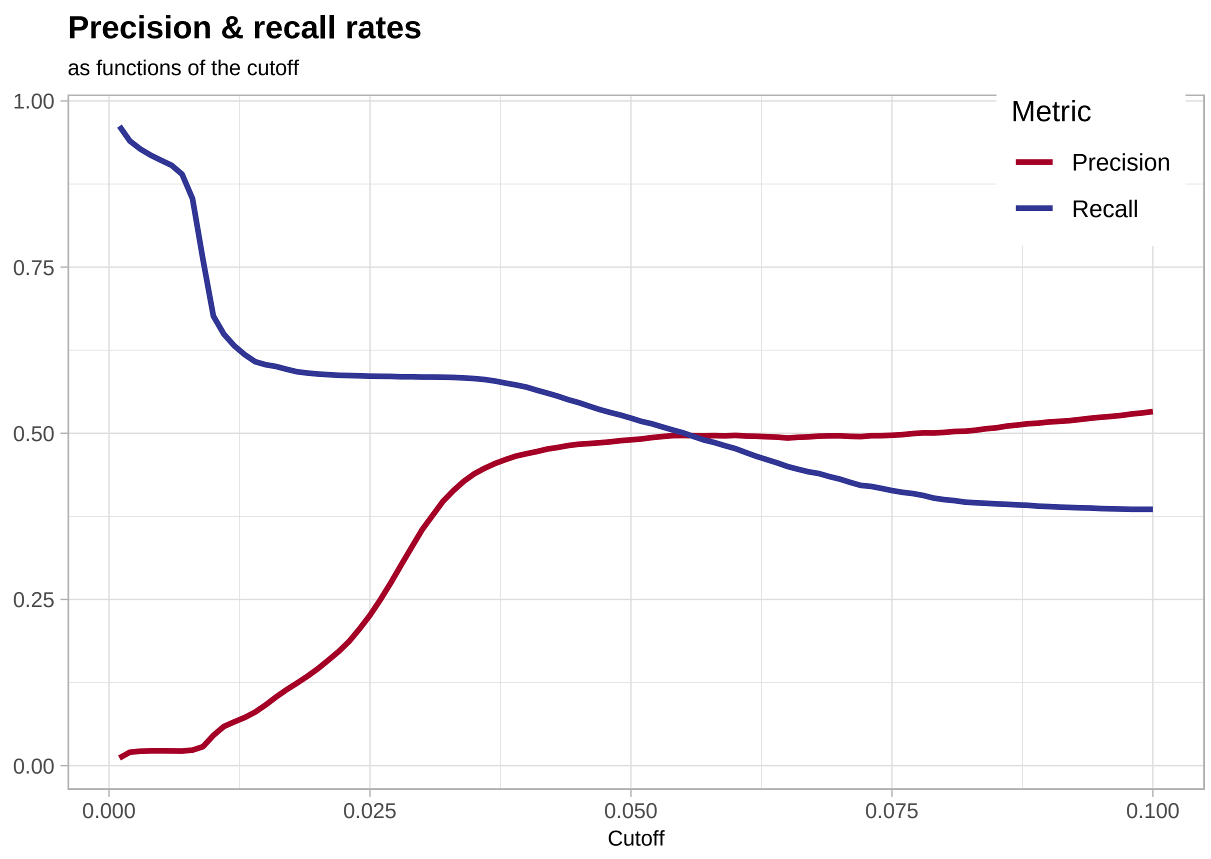
We would like to point out that we are focusing on maximizing recall, but also try to keep precision on a reasonable level. To check the model we created a function which computes a confusion matrix and also gives us values for recall and precision.

```r
confusion_matrix <- function(model, data, col, cutoff=0.5, model_type = "logistic") {

  # calculating prediction scores
  predictScore <- as.numeric(predict(object=model, type='response',newdat=data))

  if (model_type == "forest") predictScore = predictScore -1

  predic <- if_else(predictScore > cutoff, 1, 0) # assigning predictions
  confmat <- table(predic, data[[col]])

  rownames(confmat) <- c('pred_negative', 'pred_positive')
  colnames(confmat) <- c('obs_negative', 'obs_positive')

  precision <- confmat[2,2]/(confmat[2,1]+confmat[2,2]) # precision = TP/(TP+FP)
  recall <- confmat[2,2]/(confmat[1,2]+confmat[2,2]) # recall = TP/(TP+FN)

  return(c(precision, recall))
}
```

To fit the best model we create a function which produces a plot for precision and recall for each model by changing the value of parameter `cutoff` in the `glm` function.

```r
# looking for the most reasonable cutoff for a model - the function produces a plot
find_cutoff <- function(model, data, target_variable, min_cutoff=0.1, max_cutoff=0.8, by=0.05){
  cutoffs <- seq(min_cutoff, max_cutoff, by=by)
  precisions <- numeric(0)
  recalls <- numeric(0)
  for(x in cutoffs){
    # confusion matrix for the model with cutoff = x
    conf_mat <- confusion_matrix(model, data, target_variable, x)
    precisions <- c(precisions, conf_mat[1])
    recalls <- c(recalls, conf_mat[2])
  }
  # preparation for a plot
  metrics <- tibble(
    cutoffs = cutoffs,
    precisions = precisions,
    recalls = recalls
  )
  metrics_long <- melt(metrics, id='cutoffs')
  pl <- ggplot(data=metrics_long, aes(x=cutoffs, y=value, colour=variable)) +
    geom_line(size=1) +
    scale_colour_manual("Metric",
                        breaks=c("precisions", "recalls"),
                        labels=c("Precision", "Recall"),
                        values = c(my_palette[c(4,7)]))  +
    labs(title='Precision & recall rates',
         subtitle='as functions of the cutoff',
         x='Cutoff', y='') +
   theme_light() +
    theme(
      plot.subtitle = element_text(size = text_size - 4, colour='black'),
      plot.title = element_text(size = text_size, face = 'bold'),
      axis.text=element_text(size=text_size - 4),
      axis.title.y = element_text(size = text_size - 4),
      axis.title.x = element_text(size = text_size - 4),
      legend.position = c(0.9,0.9)
    )
  # plotting
  grid.arrange(pl, nrow=1, ncol=1)

}
```

```
find_cutoff(logreg, train_bin_cv, 'is_fraud', 0.001, 0.1, 0.001)
```



The function creates a graph, where we can we pick up a parameter optimal for us. We will go for value `0.03` as the best cutoff level which maximizes recall.

Let us look at the confusion matrix:

```
conf_mat_logreg_train <-
  confusion_matrix(logreg, train_bin_cv, col = "is_fraud", cutoff = 0.003)
conf_mat_logreg_train
```

```
## [1] 0.02168837 0.92761038
```

```
conf_mat_logreg_valid <-
  confusion_matrix(logreg, valid_bin_cv, col = "is_fraud", cutoff = 0.003)
conf_mat_logreg_valid
```

```
## [1] 0.02137988 0.92866756
```

As a result we get on training dataset precision equals 0.022, recall equals 0.928 and on testing dataset precision equals 0.021, recall equals 0.929. This model is a good contender, but in next consideration we will focus on getting higher precision with assuming high recall.

4.3 The Challenger models

4.3.1 Logistic regression with PCA

First challenger model will be a logistic regression with application of Principal Component Analysis (PCA) on numerical variables.

```r
# remove dependent variable from dataset and keep only numerical variables
numerical_cols <- c('amt', 'city_pop', 'age', 'dist', 'weekday_fraud', 'category_fraud', 'hour_fraud')
data_bin_PCA <- data_train_bin %>%
  .[, (numerical_cols):= lapply(.SD, as.numeric), .SDcols = numerical_cols] %>%
  .[, ..numerical_cols]

# the PCA analysis:
pca_decomposition <- prcomp(data_bin_PCA)
```

Calculate a percentage of variance explained by each PCA component.

```r
# calculating a variance of each component
variance <- pca_decomposition$sdev^2
variance_perc <- round(variance/sum(variance),4)
```

```r
ggplot(data = as.data.frame(variance_perc),
       aes(x = paste0('PC', 1:7), y = variance_perc)) +
  geom_col() +
  xlab("") +
  labs(title= 'Variance', subtitle='for each PCs',x='', y='') +
  my_style
```

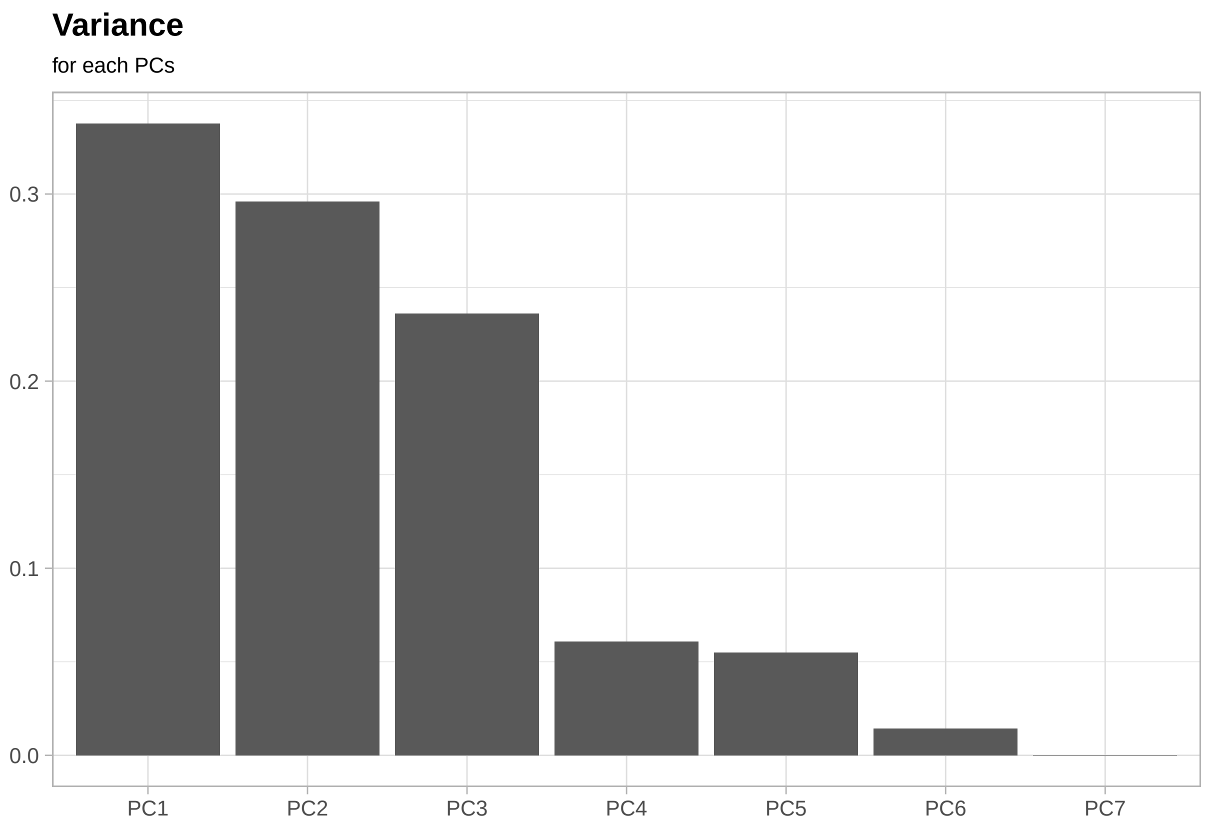


Figure 4.1: Variance for each PCs

As we can see from the Figure 4.1 first three components account the most of the variance of the data, but the third component could be also valuable for a model.

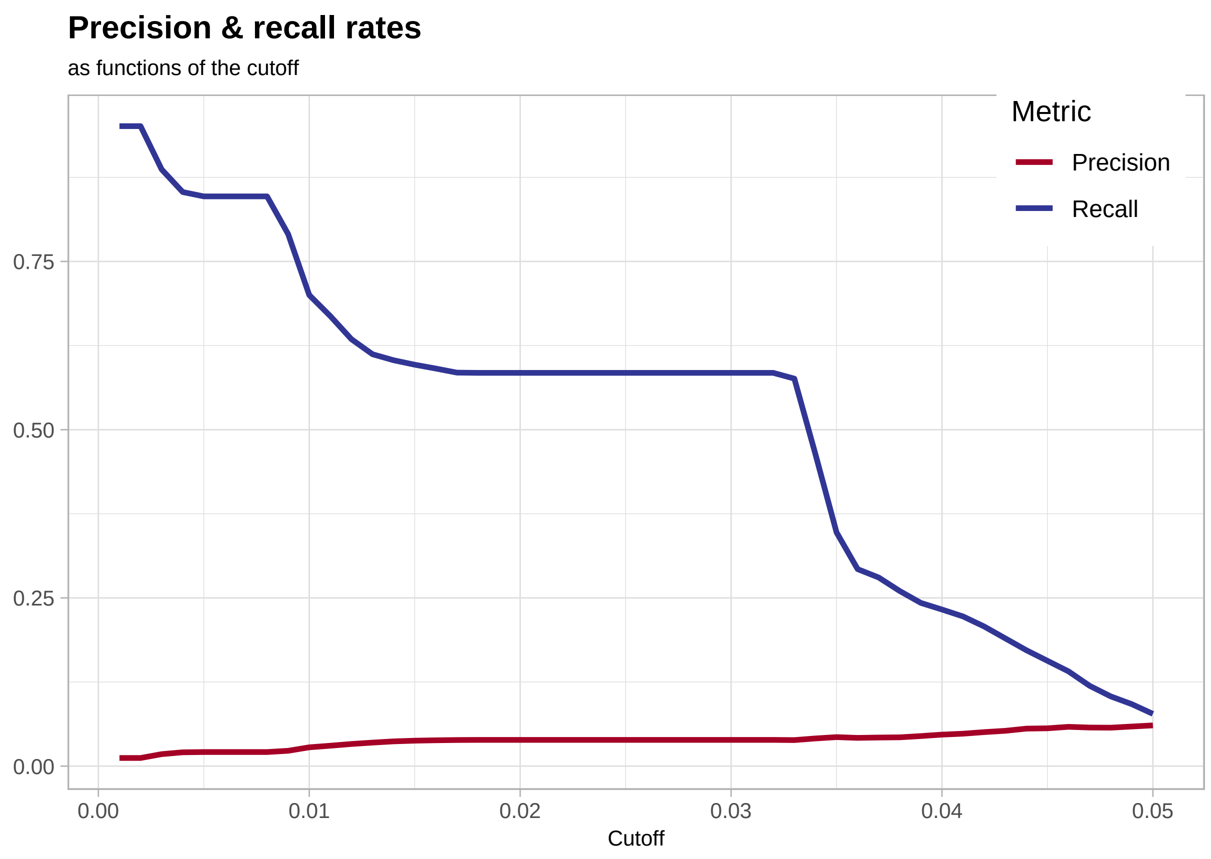
```
data_train_pca <- cbind(data_train$is_fraud, as.data.table(pca_decomposition$x)) %>%
  setNames(c("is_fraud", paste0("PC", 1:7)))
```

```
train_bin_cv_pca <- data_train_pca[data_train_bin_cv$train$idx,]
valid_bin_cv_pca <- data_train_pca[data_train_bin_cv$valid$idx,]
```

```
logreg_PCA <- glm(is_fraud ~
                    PC1 + PC2 + PC3 + PC4 + PC2*PC3 + PC1*PC4,
                  data = train_bin_cv_pca, family = binomial)
summary(logreg_PCA)
```

```
##
## Call:
## glm(formula = is_fraud ~ PC1 + PC2 + PC3 + PC4 + PC2 * PC3 +
##     PC1 * PC4, family = binomial, data = train_bin_cv_pca)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.3642  -0.0834  -0.0368  -0.0307   3.9304
##
## Coefficients:
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept) -6.37772    0.03539 -180.202  < 2e-16 ***
## PC1          0.26176    0.02819    9.286  < 2e-16 ***
## PC2         -2.46283    0.05994  -41.086  < 2e-16 ***
## PC3          2.25544    0.05020   44.930  < 2e-16 ***
## PC4         -0.32419    0.06483   -5.000 5.73e-07 ***
## PC2:PC3      0.39741    0.10803    3.679 0.000235 ***
## PC1:PC4     -0.59818    0.12892   -4.640 3.48e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 64850  on 907672  degrees of freedom
## Residual deviance: 53341  on 907666  degrees of freedom
## AIC: 53355
##
## Number of Fisher Scoring iterations: 10
```

```
find_cutoff(logreg_PCA, train_bin_cv_pca, 'is_fraud', 0.001, 0.05, 0.001)
```



```
conf_mat_logreg_PCA_train <-
  confusion_matrix(logreg_PCA, train_bin_cv_pca, col = "is_fraud", cutoff = 0.0025)
conf_mat_logreg_PCA_train
```

```
## [1] 0.01537381 0.90449119
```

```
conf_mat_logreg_PCA_valid <-
  confusion_matrix(logreg_PCA, valid_bin_cv_pca, col = "is_fraud", cutoff = 0.0025)
conf_mat_logreg_PCA_valid
```

```
## [1] 0.01523636 0.90937640
```

As a result we get on training dataset precision = 0.015, recall = 0.904 and on testing dataset precision = 0.015, recall = 0.909.

For that model we have similar results as for the previous logistic model. Moreover, we lose interpretation for numerical variables.

4.3.2 Random Forest

First, we need to create a train-test split on unnormalized data. As it was before, we use 80% of our data as a training set and 20% as a testing set.

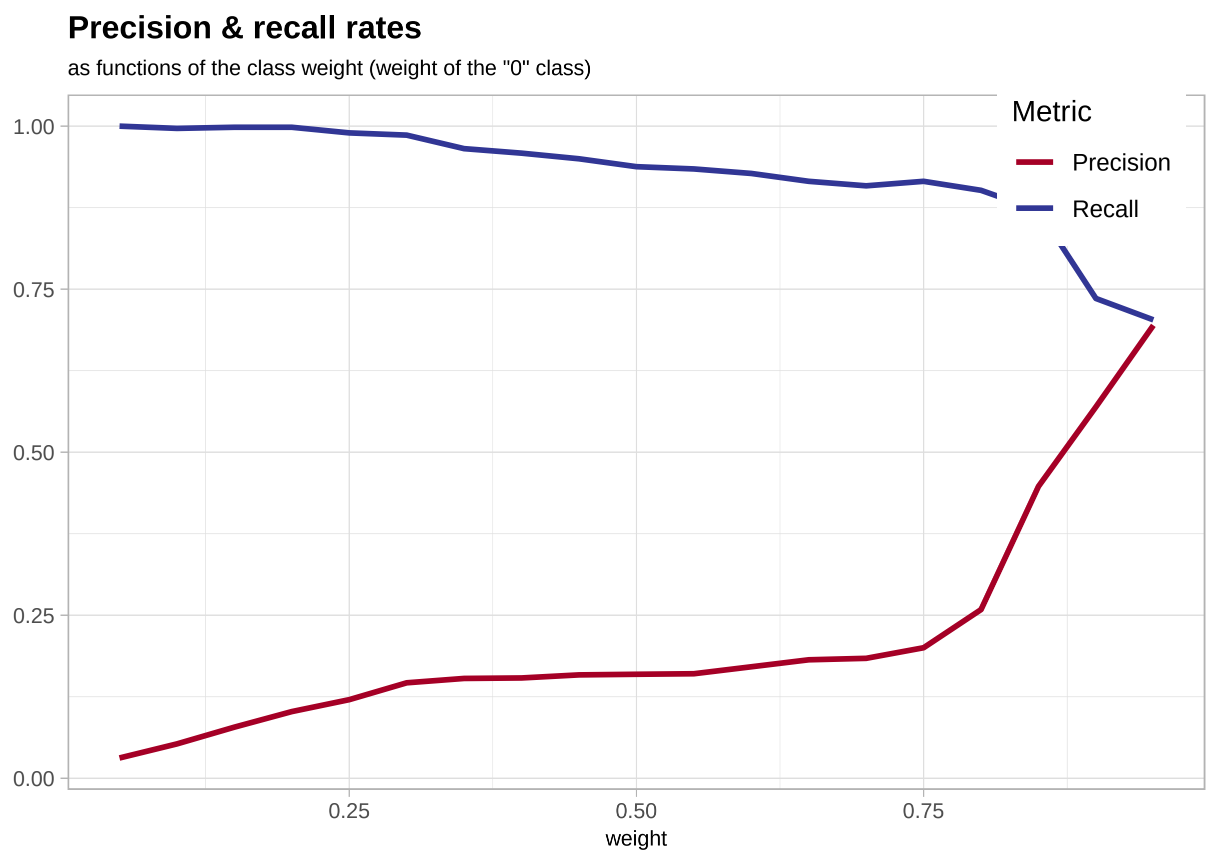
```
train_cv <- data_train[data_train_bin_cv$train$idx,]
valid_cv <- data_train[data_train_bin_cv$valid$idx,]
```

As our data is highly imbalanced, we need to find optimal parameters for a random forest in order to fit the model that performs well. To find this parameter we will use a function called `find_classwt`:

```r
find_classwt <- function(data) {

  recalls <- numeric(0)
  precisions <- numeric(0)

  for(t in seq(0.05, 0.95, by=0.05)){
    gc() # freeing unused memory
    # fitting random forest with parameter classwt = t for class '0' and = 1-t for '1'
    forest <- randomForest(as_factor(is_fraud) ~ amt + age + category_fraud + hour_fraud,
                          data=data, ntree=100, classwt = c('0' = t, '1' = 1-t))
    # confusion matrix for the model
    conf_mat <- confusion_matrix(forest, data, 'is_fraud', model_type = "forest")
    recalls <- c(recalls, conf_mat[2])
    precisions <- c(precisions, conf_mat[1])
  }
  # preparation for plotting
  metrics <- tibble(
    weights = seq(0.05, 0.95, by=0.05),
    precisions = precisions,
    recalls = recalls
  )
  metrics_long <- melt(metrics, id='weights')
  pl <- ggplot(data=metrics_long, aes(x=weights, y=value, colour=variable)) +
    geom_line(size=1) +
    scale_colour_manual("Metric",
                        breaks=c("precisions", "recalls"),
                        labels=c("Precision", "Recall"),
                        values = c(my_palette[c(4,7)]))  +
    labs(title='Precision & recall rates',
         subtitle='as functions of the class weight (weight of the "0" class)',
         x='weight', y='') +
     theme_light() +
      theme(
        plot.subtitle = element_text(size = text_size - 4, colour='black'),
        plot.title = element_text(size = text_size, face = 'bold'),
        axis.text=element_text(size=text_size - 4),
        axis.title.y = element_text(size = text_size - 4),
        axis.title.x = element_text(size = text_size - 4),
        legend.position = c(0.9,0.9)
      )

  return(pl)
}
```

The function produces a plot, where we can find a parameter optimal for us.

```r
set.seed(125)
subset_tr <- sample_n(train_cv, 10^5)
find_classwt(subset_tr)
```



Our goal is to maximize recall, so after the analysis we decided to use 0.25 for “0” class and 0.75 for “1” class, to balance our data.

Now, we will fit a model with these parameters. We will use 100 trees, because this setup gives us stable results.

```
forest <- randomForest(as_factor(is_fraud) ~ amt + age + category_fraud + hour_fraud,
                       data=train_cv,
                       ntree=100,
                       classwt = c('0' = 0.25, '1' = 0.75))
```

We will take a look at the confusion matrix:

```
conf_mat_forest_train <- confusion_matrix(forest, train_cv, 'is_fraud', model_type = "forest")
conf_mat_forest_train
```

```
## [1] 0.07620053 0.97157476
```

```
conf_mat_forest_valid <- confusion_matrix(forest, valid_cv, 'is_fraud', model_type = "forest")
conf_mat_forest_valid
```

```
## [1] 0.07370642 0.95603410
```

The model gives us very promising outcomes: on the training set recall equals 0.972 and precision is 0.076, while on the testing set it is 0.956 and 0.074, accordingly.

Chapter 5

Cross validation

For the cross validation we will use a function `crossv_mc`, which conducts Monte Carlo Cross Validation. Also, we will not analyse logistic regression with PCA, as this model gives not better results than the basic logistic regression and lacks the possibility of interpretation.

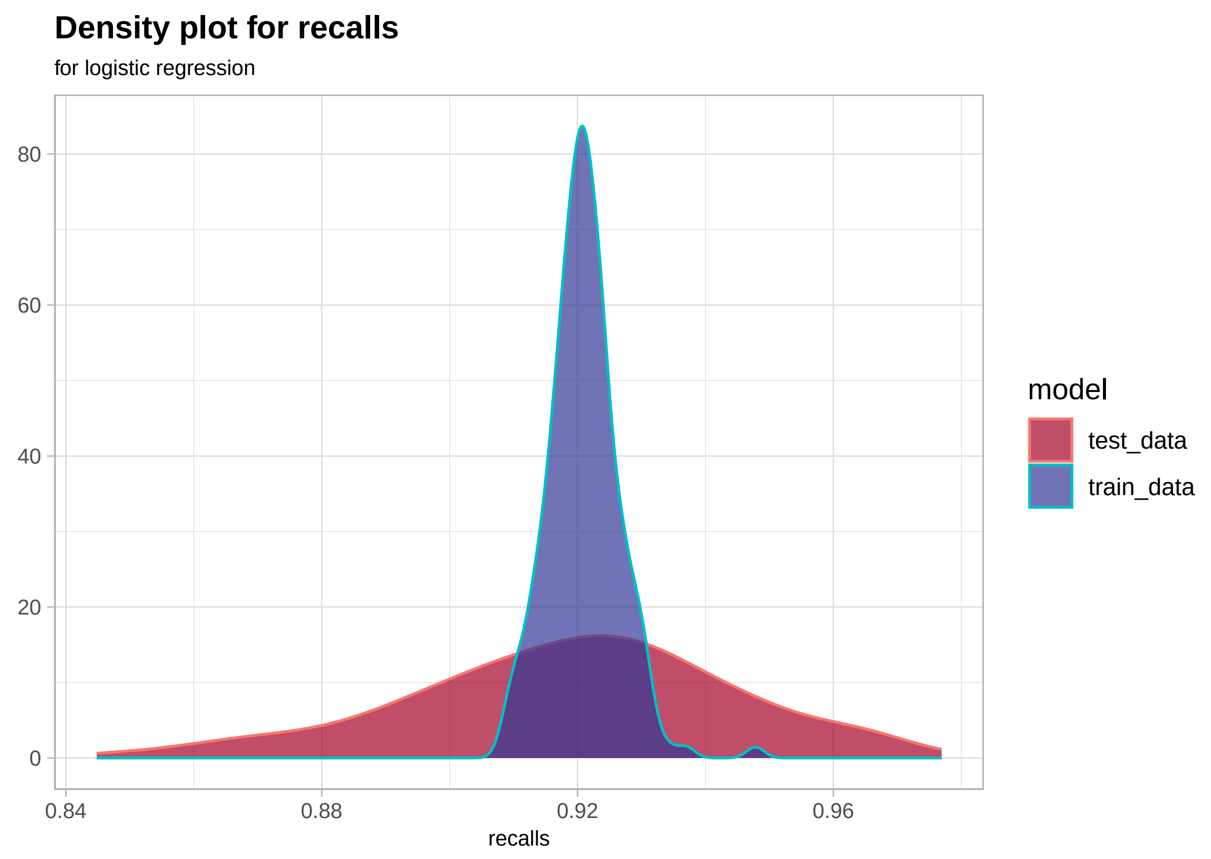
5.1 Logistic regression

First, we will take a look at the logistic regression. To do that, we will sample a subset of our data, containing 100′000 observations.

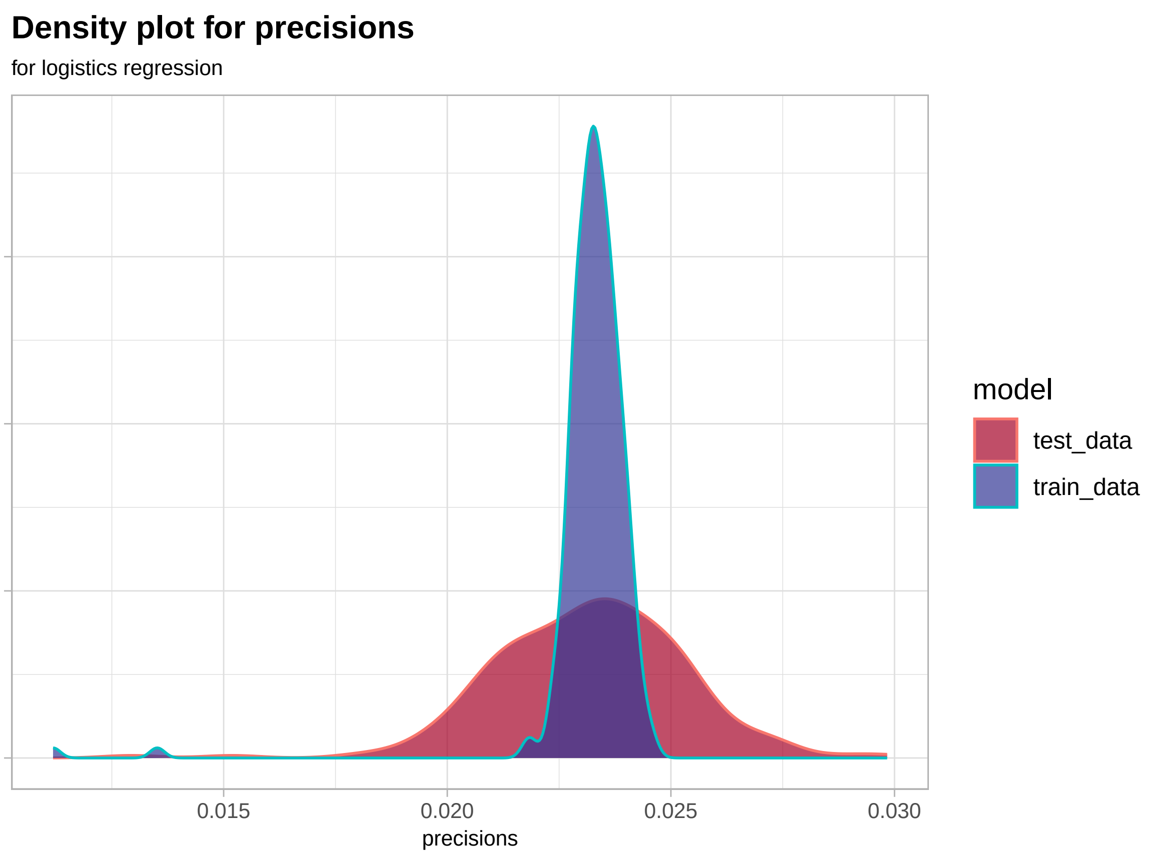
```
set.seed(125)
n_obs <- 1e5
subset <- sample_n(
  data_train_bin[,c("is_fraud", "amt", "age", "category_fraud", "hour_fraud")],
  n_obs)
test_ratio <- 0.2
N <- 200

# Creating train/test sets
cv_mc <- crossv_mc(subset, N, test=test_ratio)

# fitting models on the training sets
models <- map(cv_mc$train, ~ glm(is_fraud ~ category_fraud + amt + hour_fraud + age:hour_fraud +
                                   amt:hour_fraud + amt:category_fraud + age:category_fraud,
                                 data=., family=binomial))

recalls_train <- map2_dbl(models, cv_mc$train,
                          ~confusion_matrix(model=.x, data=as_tibble(.y), col='is_fraud',
                                            cutoff=0.003)[2])

precisions_train <- map2_dbl(models, cv_mc$train,
                             ~confusion_matrix(model=.x, data=as_tibble(.y), col='is_fraud',
                                               cutoff=0.003)[1])

recalls_test <- map2_dbl(models, cv_mc$test,
                         ~confusion_matrix(model=.x, data=as_tibble(.y), col='is_fraud',
                                           cutoff=0.003)[2])

precisions_test <- map2_dbl(models, cv_mc$test,
                            ~confusion_matrix(model=.x, data=as_tibble(.y), col='is_fraud',
                                              cutoff=0.003)[1])

recalls_all <- rbind(data.table(model = 'train_data',
                                recalls = recalls_train),
                     data.table(model = 'test_data',
                                recalls = recalls_test))

precisions_all <- rbind(data.table(model = 'train_data',
                                   precisions = precisions_train),
                        data.table(model = 'test_data',
                                   precisions = precisions_test))
```

To visualize the result we prepared a funtion which return a plot of density for metrics precision and recall.

```
# Cross-validation density of metrics
cv_metrics_plt <- function(metrics, model_nm = "logistic regression") {

  metric_nm <- names(metrics)[2]
  title_nm <- paste0("Density plot for ", metric_nm)
  subtitle_nm <- paste0("for ", model_nm)

  plt <- ggplot(as.data.frame(metrics), aes(get(metric_nm), colour = model, fill = model)) +
    geom_density(alpha = 0.7) +
    theme_light() +
    theme(
      plot.subtitle = element_text(size = text_size - 4, colour='black'),
      plot.title = element_text(size = text_size, face = 'bold'),
      axis.text=element_text(size=text_size - 4),
      axis.title.y = element_text(size = text_size - 4),
      axis.title.x = element_text(size = text_size - 4)
    ) +
    scale_fill_manual(values = c(my_palette[c(4,7)]))  +
    labs(title = title_nm,
         subtitle = subtitle_nm,
         y = "",
         x = metric_nm)

  return(plt)
}

cv_metrics_plt(recalls_all, "logistic regression")
```



```r
cv_metrics_plt(precisions_all, "logistics regression")
```

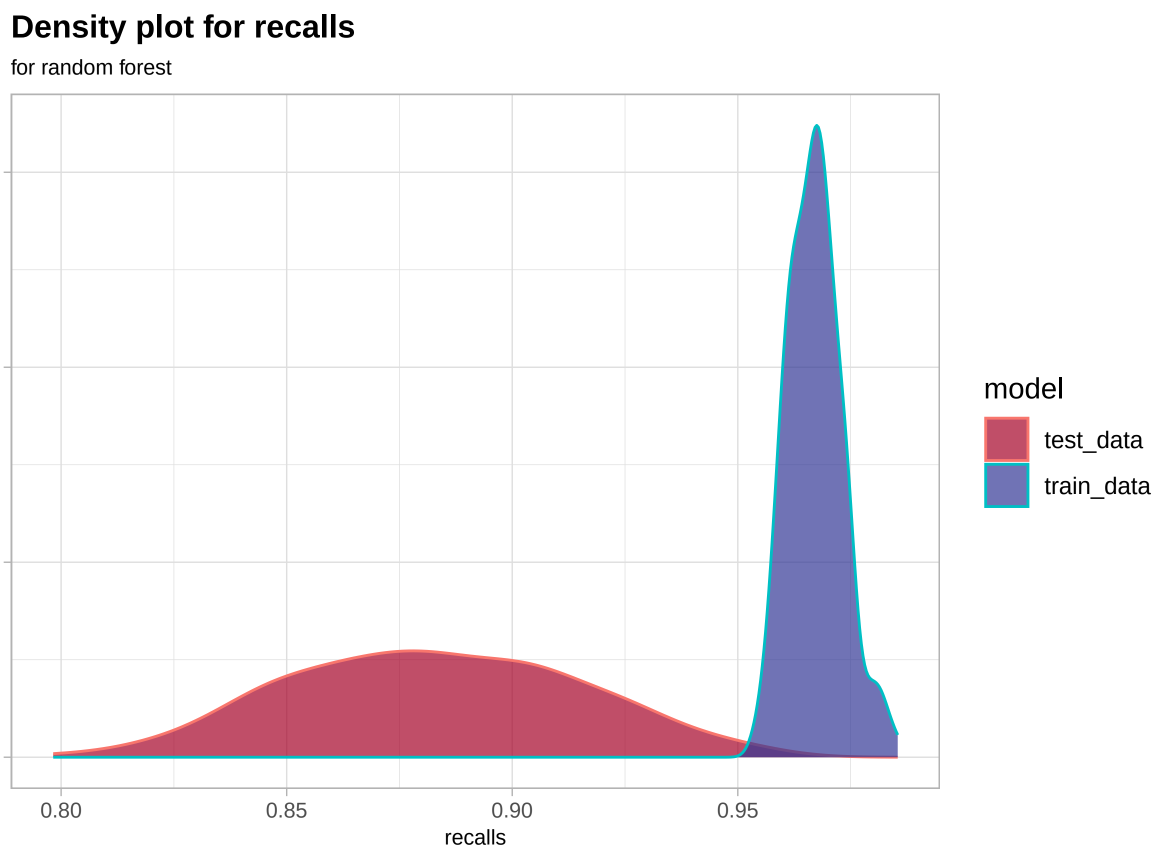


We can see on the density plots, that recall is high, the density for the testing set is wider, but no serious overfitting can be seen. The precision is very low, but also the densities for the training and the testing sets look similar.

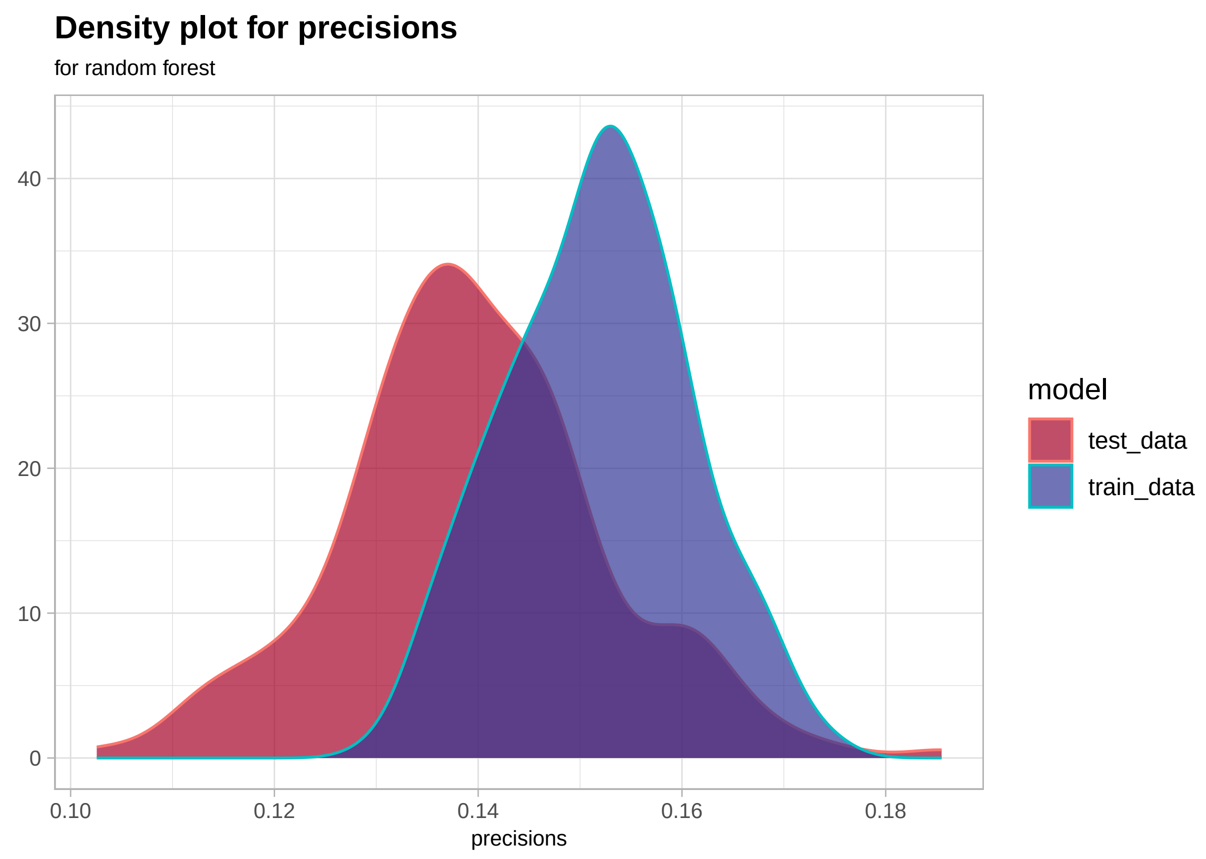
5.2 Random forest

Now, we will do the same for the random forest.

```r
set.seed(125)
n_obs <- 1e5
subset <- sample_n(
  data_train[,c("is_fraud", "amt", "age", "category_fraud", "hour_fraud")],
  n_obs)
test_ratio <- 0.2
N <- 200

# Creating train/test sets
cv_mc <- crossv_mc(subset, N, test=test_ratio)

# fitting models on the training sets
models <- map(cv_mc$train, ~ randomForest(as_factor(is_fraud) ~ amt + age + category_fraud + hour_fraud,
                                          data=., ntree=100, classwt = c('0' = 0.25, '1' = 0.75)))

recalls_train <- map2_dbl(models, cv_mc$train,
                          ~confusion_matrix(model=.x, data=as_tibble(.y),
                                            col='is_fraud', model_type = "forest")[2])

precisions_train <- map2_dbl(models, cv_mc$train,
                             ~confusion_matrix(model=.x, data=as_tibble(.y),
                                                        col='is_fraud', model_type = "forest")[1])

recalls_test <- map2_dbl(models, cv_mc$test,
                         ~confusion_matrix(model=.x, data=as_tibble(.y),
                                           col='is_fraud', model_type = "forest")[2])

precisions_test <- map2_dbl(models, cv_mc$test,
                            ~confusion_matrix(model=.x, data=as_tibble(.y),
                                              col='is_fraud', model_type = "forest")[1])

recalls_all <- rbind(tibble(model = 'train_data', recalls = recalls_train),
                     tibble(model = 'test_data', recalls = recalls_test))

precisions_all <- rbind(tibble(model = 'train_data', precisions = precisions_train),
                        tibble(model = 'test_data', precisions = precisions_test))

# Density plot of metrics
cv_metrics_plt(recalls_all, "random forest")
```



```
cv_metrics_plt(precisions_all, "random forest")
```



Here, we can see that the model overfits quite seriously, although the metrics for the testing set are still relatively high. The overfit can be seen on the recall, which is not surprising, because we were focusing on finding the parameters that maximize this metric, so the model is not as robust as it should be.

Chapter 6

Testing dataset

Finally, to provide an unbiased evaluation of a final models fit, we check models performance on independent testing dataset. Similarly, as in previous Section we will not analyse logistic regression with PCA, because of poor output comparing with the basic logistic regression.

We created functions used to data preprocessing, before fit the models.

```r
# Data preprocessing used for test dataset (random forest)
data_preprocess_forest <- function(data) {

  # Select useful columns
  data <- data %>%
    .[,c("is_fraud", "category", "amt", "trans_date_trans_time", "dob")]

  # Convert into factors
  dt <- c( "category", "is_fraud")
  data[,(dt):= lapply(.SD, as.factor), .SDcols = dt]

  # Convert into data class
  data[, trans_date_trans_time := strptime(trans_date_trans_time,
                                          format ="%Y-%m-%d %H:%M:%OS",
                                          tz = "EST")]
  # Compute age of the customer in a day of transaction
  data[, age := round(as.numeric(difftime(trans_date_trans_time, dob, units = "days")/365), 0)]
  # Adding a weekday of transaction
  data[, week_day_of_trans := as.factor(wday(trans_date_trans_time))]
  # Hour of the transaction
  data[, trans_hour := as_factor(strftime(trans_date_trans_time,
                                         format="%H",
                                         tz = "EST"))]
  # Adding bin variables
  data[, weekday_fraud := if_else(week_day_of_trans %in% 4:7, 1, 0)]
  data[, category_fraud :=
         if_else(category %in% c('grocery_pos','misc_net','shopping_net','shopping_pos'),1,0)]
  data[, hour_fraud :=
         if_else(trans_hour %in% c('00', '01', '02', '03', '22', '23'), 1, 0)]
  data[,c("weekday_fraud", "category_fraud", "hour_fraud"):=
         lapply(.SD, as.factor), .SDcols = c("weekday_fraud", "category_fraud", "hour_fraud")]

  return(data)
}
```

```r
# Data preprocessing used for test dataset (logistic regression)
data_preprocess_logreg <- function(data) {

  # Same data manipulation as for random forest
  data <- data_preprocess_forest(data)
  # Normalization
  temp <- c('amt','age')
  data[, (temp) := lapply(.SD, min_max_norm), .SDcols = temp]

  return(data)
}
```

6.1 Logistic regression

Performance check on testing dataset for logistic regression.

```r
# Logistic regression
data_test_logreg <- data_preprocess_logreg(data_test)
logreg <- glm(is_fraud ~
                category_fraud + amt + hour_fraud + age:hour_fraud + amt:hour_fraud + amt:category_fraud +
              data = data_test_logreg,
              family = binomial)
```

We will take a look at the confusion matrix:

```r
conf_mat_logreg_test <- confusion_matrix(logreg, data_test_logreg, col = "is_fraud", cutoff = 0.003)
conf_mat_logreg_test
```

```
## [1] 0.01467141 0.90163170
```

The model gives outcomes on testing data set: recall equals 0.902 and precision is 0.015.

6.2 Random forest

Performance check on testing dataset for random forest.

```r
# Random forest
data_test_forest <- data_preprocess_forest(data_test)
forest <- randomForest(as_factor(is_fraud) ~ amt + age + category_fraud + hour_fraud,
              data = data_test_forest,
              ntree=100,
              classwt = c('0' = 0.25, '1' = 0.75))
```

We will take a look at the confusion matrix:

```r
conf_mat_forest_test <- confusion_matrix(forest, data_test_forest, col = "is_fraud", cutoff = 0.0025, mode
conf_mat_forest_test
```

```
## [1] 0.06196626 0.96410256
```

The model gives outcomes on the testing data set: recall equals 0.964 and precision is 0.062.

Chapter 7

Conclusion

Comparing the three models, the logistic regression gives the best recall and also do not overfit much. On the other hand, even though the random forest overfits our data heavily, it gives not much lower recall on the testing set and also much higher precision. The PCA for the logistic regression model gives similar results as the basic logistic regression, while it does not provide possibility to interpret the model. For that reasons, we reject this model for our problem.

Table 7.1: Table with results for the logistic regression and the random forest.

Model	Quality	Recall_train	Precision_train	Recall_test	Precision_test
logistic regression	not overfited	0.928	0.022	0.902	0.015
random forest	significantly overfited	0.972	0.076	0.964	0.062

We think that the logistic regression is more transparent and in this case also more robust, while the random forest is better in terms of not giving too many false alarms. We would recommend implementing the model `logreg` for now. On the other hand, we believe, that random forest could give better results in the long run, if implemented with additional variables like "time from the last transaction" or "number of transactions made on the same day".

Overall, for now we recommend the model `logreg` as the best of the models presented in this paper.