Bank Customer Churn Analysis

Can Zhang

2022-06-17

Introduction

Customer churn analysis is widely used nowadays by companies all over the world. With the massive volume of data available and fast-developing big data tools and technologies, businesses can easily identify customers at risk of churn and develop better strategies to improve customer retention.

This project is part of the Data Science Professional Certificate program offered by Harvard University through edX platform.

This project tries to predict churning behavior in the banking industry using a data set originated from a U.S. bank. "F1 score" is chosen as the evaluation metric as we try to balance Precision and Recall.

Four models were used for this project:

1. glm
2. ranger
3. KNN
4. LightGBM

The best performing model - ranger - is used for the final validation.

This report will present an overview of the data set, the model building process, the results of the models and final validation, and the conclusion of this project.

Load packages

The packages used in this project include:

```
library(readr)
library(tidyverse)
library(dplyr)
library(caret)
library(ggplot2)
library(gridExtra)
library(ranger)
library(tidymodels)
library(MLmetrics)
library(lightgbm)
library(knitr)
```

The data set

The data set used in this project originated from a US bank. The original data set is shared by Shantanu Dhakad on Kaggle (click link to go to the source: https://www.kaggle.com/datasets/shantanudhakadd/bank-customer-churn-prediction). To facilitate the process, the data set is uploaded to GitHub.

Download the data

The data set can be downloaded with the following code:

```
download.file("https://raw.githubusercontent.com/chiarazh/Bank-Customer-Churn-Analysis/main/Churn_Modell

bank <- read_csv("Churn_Modelling.csv", show_col_types = FALSE )
```

Data pre-process

The data set has 10,000 observations of 14 variables.

```
str(bank, give.attr = FALSE)
```

```
## spec_tbl_df [10,000 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ RowNumber      : num [1:10000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ CustomerId     : num [1:10000] 15634602 15647311 15619304 15701354 15737888 ...
##  $ Surname        : chr [1:10000] "Hargrave" "Hill" "Onio" "Boni" ...
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : chr [1:10000] "France" "Spain" "France" "France" ...
##  $ Gender         : chr [1:10000] "Female" "Female" "Female" "Female" ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : num [1:10000] 1 0 1 0 1 1 1 1 0 1 ...
##  $ IsActiveMember : num [1:10000] 1 1 0 0 1 0 1 0 1 1 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : num [1:10000] 1 0 1 0 0 1 0 1 0 0 ...
```

There are no NA values in the data set.

```
na_values <- cbind(lapply(lapply(bank, is.na), sum))
na_values
```

```
##                 [,1]
## RowNumber       0
## CustomerId      0
## Surname         0
## CreditScore     0
## Geography       0
## Gender          0
## Age             0
## Tenure          0
```

```
## Balance         0
## NumOfProducts   0
## HasCrCard       0
## IsActiveMember  0
## EstimatedSalary 0
## Exited          0
```

As "RowNumber", "Surname" and "CustomerId" are not useful for our analysis, we drop these three columns.

```
bank <- bank %>% select(-c("RowNumber", "Surname", "CustomerId"))
str(bank, give.attr = FALSE)
```

```
## tibble [10,000 x 11] (S3: tbl_df/tbl/data.frame)
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : chr [1:10000] "France" "Spain" "France" "France" ...
##  $ Gender         : chr [1:10000] "Female" "Female" "Female" "Female" ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : num [1:10000] 1 0 1 0 1 1 1 1 0 1 ...
##  $ IsActiveMember : num [1:10000] 1 1 0 0 1 0 1 0 1 1 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : num [1:10000] 1 0 1 0 0 1 0 1 0 0 ...
```

We plan to mainly use the caret package for modeling, therefore we'll convert "Geography", "Gender", "Exited", "IsActiveMember" and "HasCrCard" variables from "numeric" to "factor". The "Exited" variable is the response variable, due to some models' requirements, we replace "Exited = 0" with "no", and "Exited = 1" with "yes".

```
bank_temp <- bank

bank_temp$Geography <- as.factor(bank_temp$Geography)
bank_temp$Gender <- as.factor(bank_temp$Gender)
bank_temp$IsActiveMember <- as.factor(bank_temp$IsActiveMember)
bank_temp$HasCrCard <- as.factor(bank_temp$HasCrCard)

bank_temp <- bank_temp %>%
  mutate(Exited = ifelse(Exited == "1", "yes", "no") %>%
           factor(levels = c("yes", "no")))

str(bank_temp)
```

```
## tibble [10,000 x 11] (S3: tbl_df/tbl/data.frame)
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : Factor w/ 3 levels "France","Germany",..: 1 3 1 1 3 3 1 2 1 1 ...
##  $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 1 2 2 ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 2 2 1 2 ...
```

```
##  $ IsActiveMember : Factor w/ 2 levels "0","1": 2 2 1 1 2 1 2 1 2 2 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : Factor w/ 2 levels "yes","no": 1 2 1 2 2 1 2 1 2 2 ...
```

Now we generate the data set for training(bank1) and the validation set(final hold-out set). The final validation set will be 10% of the whole data set.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = bank_temp$Exited, times = 1, p = 0.1, list = FALSE)
validation <- bank_temp[test_index,]
bank1 <- bank_temp[-test_index,]

dim(bank1)
```

```
## [1] 8999   11
```

```
dim(validation)
```

```
## [1] 1001   11
```

Data Exploration

The bank1 data set contains 8999 observations of 11 variables.

```
str(bank1)
```

```
## tibble [8,999 x 11] (S3: tbl_df/tbl/data.frame)
##  $ CreditScore    : num [1:8999] 619 608 502 699 850 645 822 376 501 684 ...
##  $ Geography      : Factor w/ 3 levels "France","Germany",..: 1 3 1 1 3 3 1 2 1 1 ...
##  $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 1 2 2 ...
##  $ Age            : num [1:8999] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:8999] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:8999] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:8999] 1 1 3 2 1 2 2 4 2 1 ...
##  $ HasCrCard      : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 2 2 1 2 ...
##  $ IsActiveMember : Factor w/ 2 levels "0","1": 2 2 1 1 2 1 2 1 2 2 ...
##  $ EstimatedSalary: num [1:8999] 101349 112543 113932 93827 79084 ...
##  $ Exited         : Factor w/ 2 levels "yes","no": 1 2 1 2 2 1 2 1 2 2 ...
```

Of all the customers in bank1 data set, about 20% churned.

```
table(bank1$Exited)
```

```
##
##  yes   no
## 1833 7166
```

```
churn_rate <- sum(bank1$Exited == "yes")/nrow(bank1)
churn_rate
```

```
## [1] 0.2036893
```

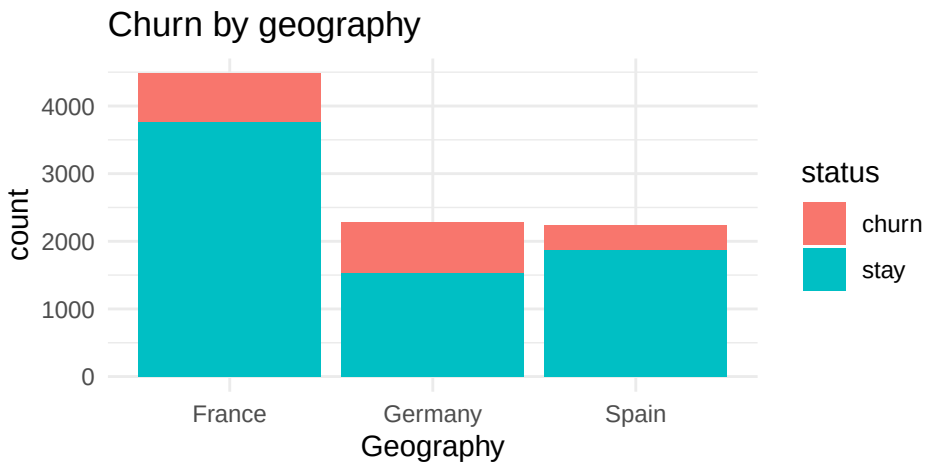
Churn and geography

Customers come from three countries: France, Germany and Spain. By calculating the proportions of churn of each country, we can see that the churn rate of Germany is much higher than the other two countries. The reason could be that the bank's service in Germany is not as good, or the market in Germany is more competitive.

```
churn_country <- bank1 %>% group_by(Geography) %>%
  summarise(total_customer = n(),
            churn = sum(Exited == "yes"),
            stay = total_customer - churn,
            churn_rate = churn/total_customer)
churn_country
```

```
## # A tibble: 3 x 5
##   Geography total_customer churn  stay churn_rate
##   <fct>              <int> <int> <int>      <dbl>
## 1 France              4478   713  3765      0.159
## 2 Germany             2280   745  1535      0.327
## 3 Spain               2241   375  1866      0.167
```

```
plot_country <- churn_country %>% select(Geography, churn, stay) %>%
  pivot_longer(!Geography, names_to = "status", values_to = "count") %>%
  ggplot(aes(x=Geography, y = count, fill = status)) +
  geom_bar(position="stack", stat="identity") +
  ggtitle("Churn by geography") +
  theme_minimal()
plot_country
```



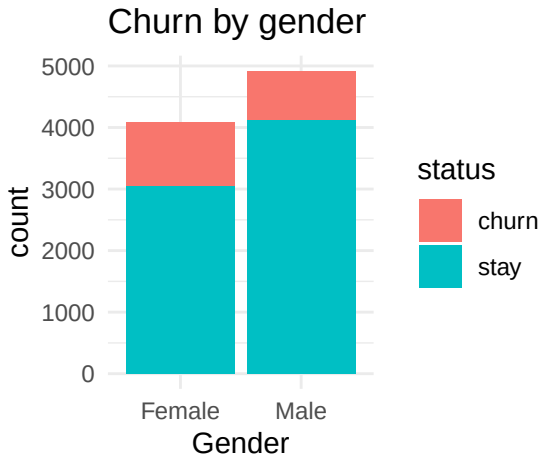
Churn and gender

There are 4078 females and 4921 males in bank1 data set, and the churn rate of females is about 25% percent, much higher than males (16%).

```
churn_gender <- bank1 %>%
  group_by(Gender) %>%
  summarise(total_customer = n(),
            churn = sum(Exited== "yes"),
            stay = total_customer - churn,
            churn_rate = churn/total_customer)
churn_gender
```

```
## # A tibble: 2 x 5
##   Gender total_customer churn  stay churn_rate
##   <fct>           <int> <int> <int>      <dbl>
## 1 Female           4078  1025  3053      0.251
## 2 Male             4921   808  4113      0.164
```

```
plot_gender <- churn_gender %>%
  select(Gender, churn, stay) %>%
  pivot_longer(!Gender, names_to = "status", values_to = "count") %>%
  ggplot(aes(x = Gender, y = count, fill = status)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Churn by gender") +
  theme_minimal()
plot_gender
```



Churn and NumOfProducts

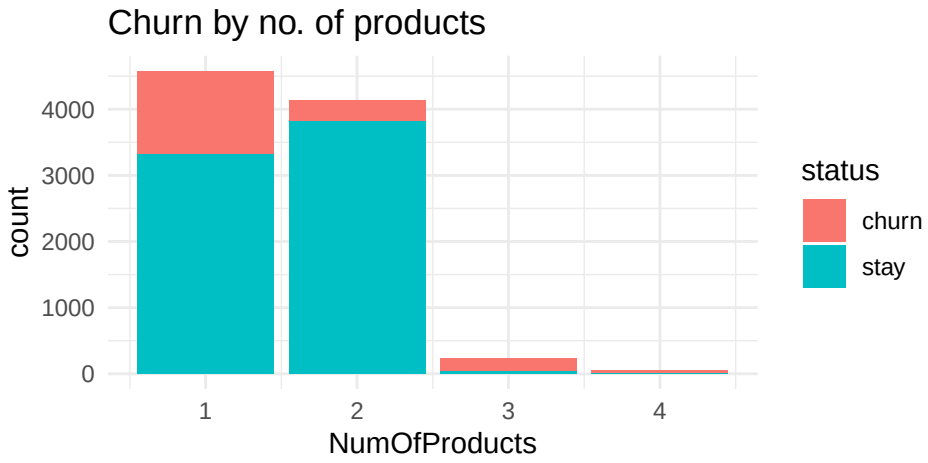
The number of products that customers have ranges from 1 to 4. We notice that Customers with a high number of products are very likely to exit (all customers with 4 products exited, 84% of customers with 3 products exited), and customers with 2 products are very loyal. The reason could be that customers with high number of products tend to be people who are more active financially(e.g. more diversified investments, more activities in foreign markets, etc.), and the variety of the bank's products might not be satisfying enough.

```
churn_product <- bank1 %>%
  group_by(NumOfProducts) %>%
  summarise(total_customer = n(),
```

```
            churn = sum(Exited == "yes"),
            stay = total_customer - churn,
            churn_rate = churn/total_customer)
churn_product
```

```
## # A tibble: 4 x 5
##   NumOfProducts total_customer churn  stay churn_rate
##           <dbl>          <int> <int> <int>      <dbl>
## 1             1           4579  1267  3312     0.277
## 2             2           4131   315  3816     0.0763
## 3             3            238   200    38     0.840
## 4             4             51    51     0     1
```

```
plot_product <- churn_product %>%
  select(NumOfProducts, churn, stay) %>%
  pivot_longer(!NumOfProducts, names_to = "status", values_to = "count") %>%
  ggplot(aes(x = NumOfProducts, y = count, fill = status)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Churn by no. of products") +
  theme_minimal()
plot_product
```



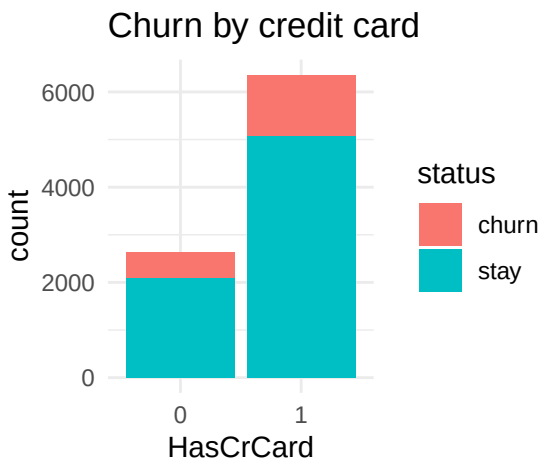
Churn and HasCrCard

The majority of customers has credit card, but there's not much difference between the churn of these two groups.

```
churn_crcard <- bank1 %>%
  group_by(HasCrCard) %>%
  summarise(total_customer = n(),
            churn = sum(Exited == "yes"),
            stay = total_customer - churn,
            churn_rate = churn/total_customer)
churn_crcard
```

```
## # A tibble: 2 x 5
##   HasCrCard total_customer churn  stay churn_rate
##   <fct>              <int> <int> <int>      <dbl>
## 1 0                   2639   554  2085      0.210
## 2 1                   6360  1279  5081      0.201
```

```
plot_crcard <- churn_crcard %>%
  select(HasCrCard, churn, stay) %>%
  pivot_longer(!HasCrCard, names_to = "status", values_to = "count") %>%
  ggplot(aes(x = HasCrCard, y = count, fill = status)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Churn by credit card") +
  theme_minimal()
plot_crcard
```



Churn and IsActiveMember

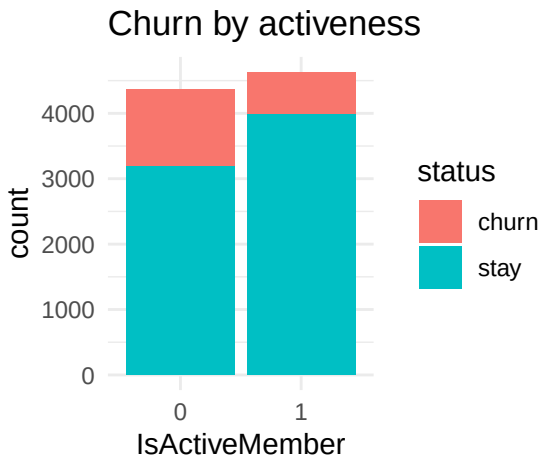
About half of the customers are active members, and they are less likely to leave the bank.

```
churn_active <- bank1 %>%
  group_by(IsActiveMember) %>%
  summarise(total_customer = n(),
            churn = sum(Exited == "yes"),
            stay = total_customer - churn,
            churn_rate = churn/total_customer)
churn_active
```

```
## # A tibble: 2 x 5
##   IsActiveMember total_customer churn  stay churn_rate
##   <fct>                  <int> <int> <int>      <dbl>
## 1 0                       4368  1178  3190      0.270
## 2 1                       4631   655  3976      0.141
```

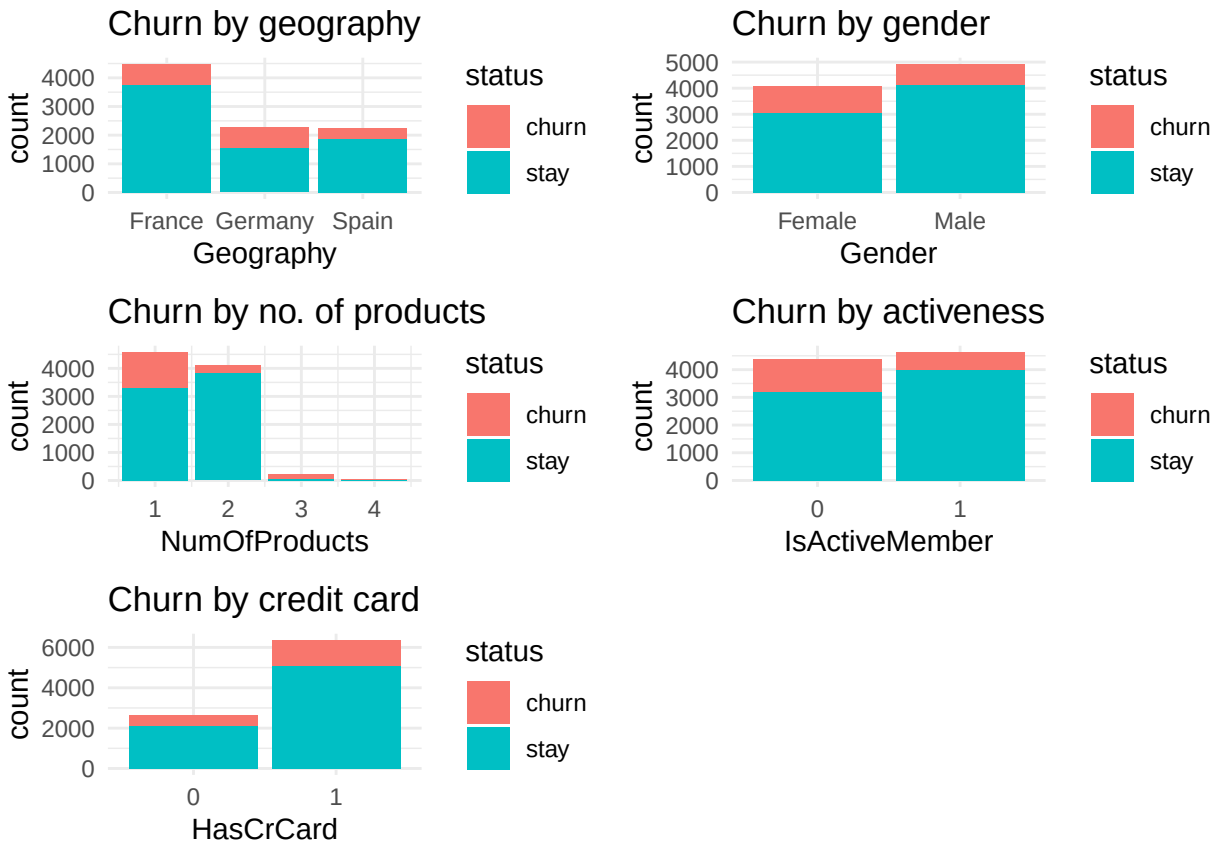
```
plot_active <- churn_active %>%
  select(IsActiveMember, churn, stay) %>%
```

```
  pivot_longer(!IsActiveMember, names_to = "status", values_to = "count") %>%
  ggplot(aes(x = IsActiveMember, y = count, fill = status)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Churn by activeness") +
  theme_minimal()
plot_active
```



We can put the plots of all factor variables together:

```
plot_combine1 <- grid.arrange(plot_country,
                              plot_gender,
                              plot_product,
                              plot_active,
                              plot_crcard,
                              nrow = 3)
```



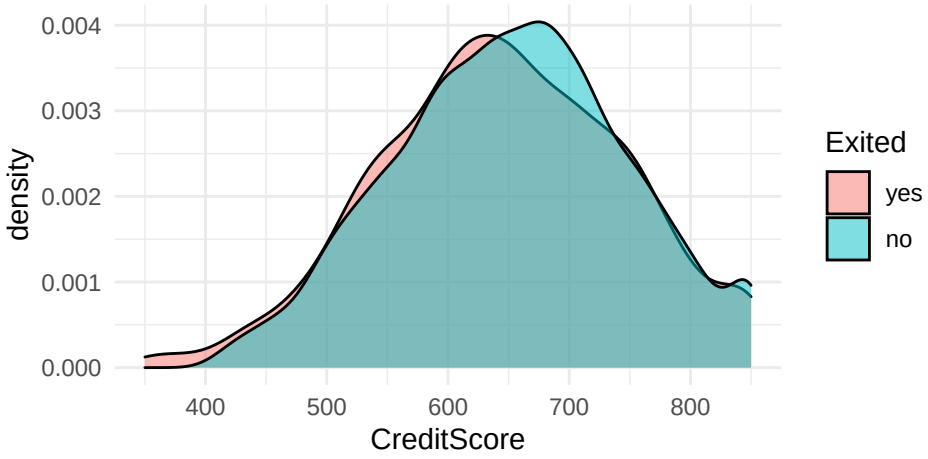

```
plot_combine1
```

```
## TableGrob (3 x 2) "arrange": 5 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (2-2,1-1) arrange gtable[layout]
## 4 4 (2-2,2-2) arrange gtable[layout]
## 5 5 (3-3,1-1) arrange gtable[layout]
```

Churn and credit score

Customers whose credit score is lower than 400 almost all left the bank.

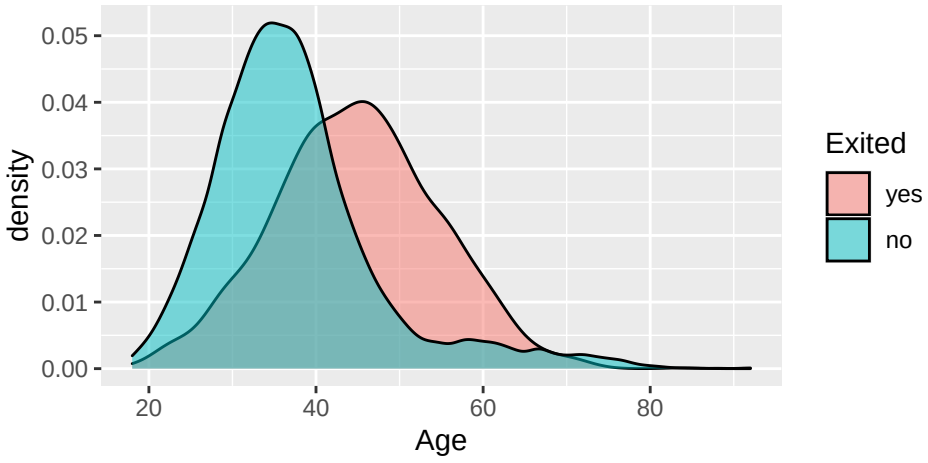
```
plot_credit <- bank1 %>%
  ggplot(aes(x = CreditScore, fill = Exited)) +
  geom_density(alpha = 0.5) +
  theme_minimal()
plot_credit
```



Churn and age

Customers aged between 40 and 65 are more likely to leave than stay. Maybe the bank's services and platform are more attractive for young people, and older people might find it difficult to use.

```
plot_age <- bank1 %>%
  ggplot(aes(x = Age, fill = Exited)) +
  geom_density(alpha = 0.5)
  theme_minimal()
plot_age
```

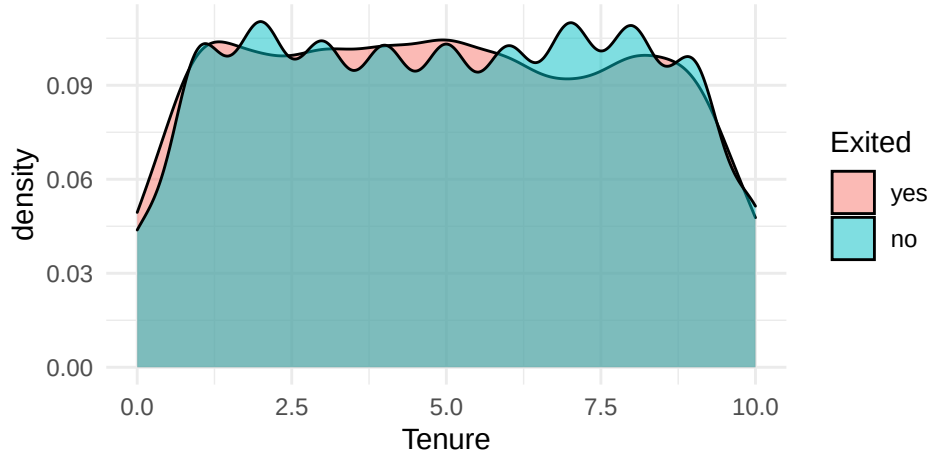


Churn and tenure

Customers who used the bank for less than 1 year are a bit more likely to leave the bank, but in general, "tenure" doesn't have much influence on churn.

```
plot_tenure <- bank1 %>%
  ggplot(aes(x = Tenure, fill = Exited)) +
  geom_density(alpha = 0.5) +
```

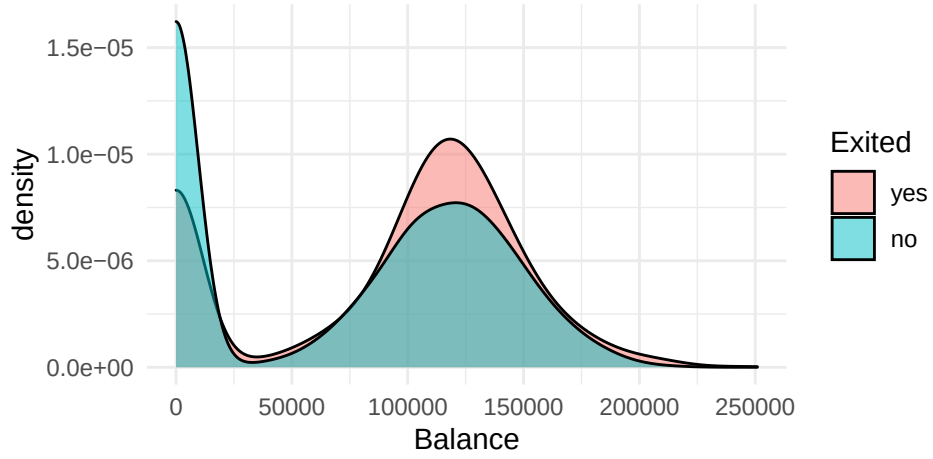
```
  theme_minimal()
plot_tenure
```



Churn and balance

It's quite interesting that customers with low balance mainly stay while customers with middle-range or high balance are more likely to leave. This could be related to the diversity of the products or quality of service.

```
plot_balance <- bank1 %>%
  ggplot(aes(x = Balance, fill = Exited)) +
  geom_density(alpha = 0.5) +
  theme_minimal()
plot_balance
```



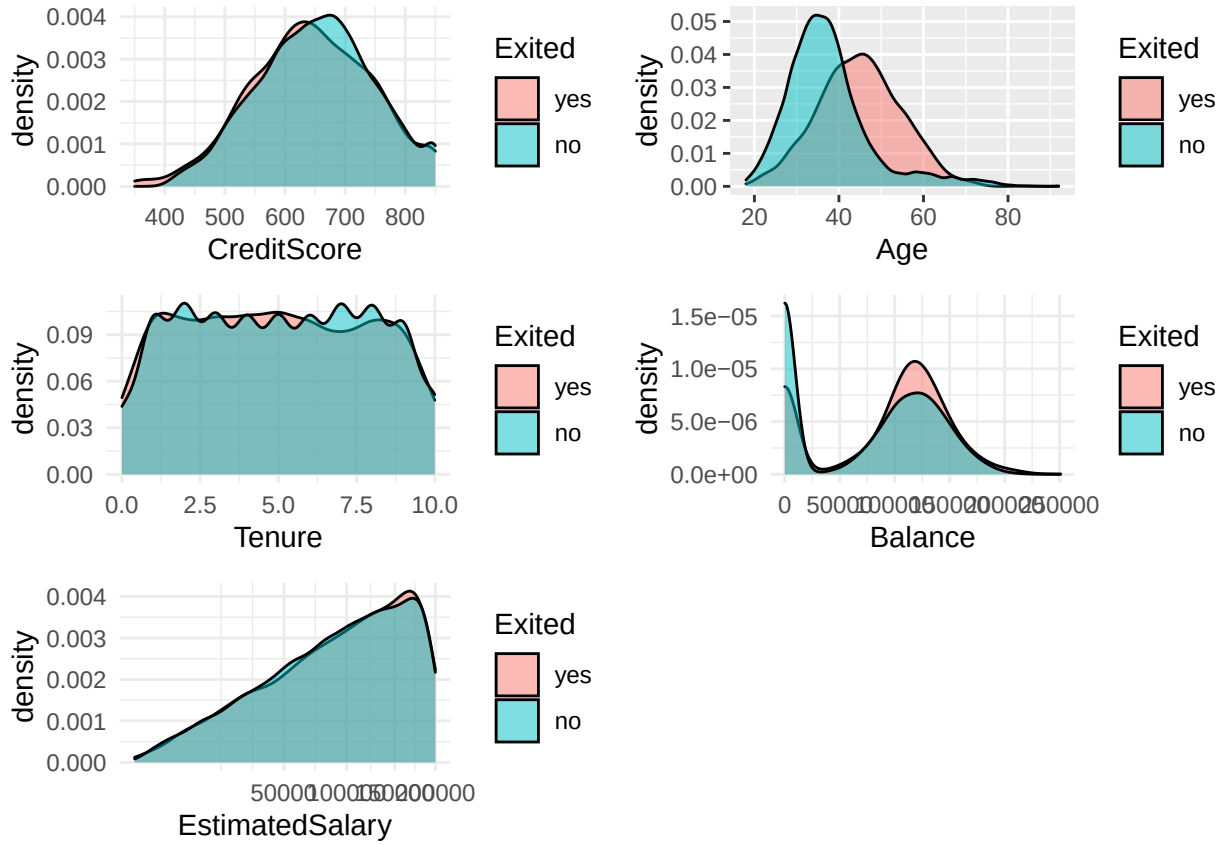
Churn and estimated salary

It seems that the estimated salary variable doesn't have much impact on the churn rate.

```
plot_salary <- bank1 %>%
  ggplot(aes(x = EstimatedSalary, fill = Exited)) +
  geom_density(alpha = 0.5) +
  scale_x_sqrt() +
  theme_minimal()
plot_salary
```

We can put the plots of all numeric variables together:

```
plot_combine2 <- grid.arrange(plot_credit,
                              plot_age,
                              plot_tenure,
                              plot_balance,
                              plot_salary,
                              nrow = 3 )
```




```
plot_combine2
```

```
## TableGrob (3 x 2) "arrange": 5 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
## 3 3 (2-2,1-1) arrange gtable[layout]
## 4 4 (2-2,2-2) arrange gtable[layout]
## 5 5 (3-3,1-1) arrange gtable[layout]
```

Model building

Choose the evaluation metric

Precision and recall are two metrics often used in churn analysis. Precision is also known as "positive predictive value", it is defined as the number of true positives(TP) divided by the number of true positives(TP) plus the number of false positives(FP).It measures the proportion of predicted churn that really churns. Recall is also known as sensitivity, it is defined as the number of true positives (TP) divided by the number of true positives(TP) plus the number of false negatives(FN). It measures the proportion of true churn that the model correctly gets.

In real-life scenarios, the evaluation metric is never decided by the data analyst alone. It's a complex business decision that involves both sales and marketing teams since the budget dedicated to customer retention is

often quite limited. For this report, we'll use F1 score - the harmonic balance of precision and recall - as the evaluation metric for different models. F1-score is calculated as follows:

$$F1-score = 2 \times \frac{precison \times recall}{precision + recall}$$

We'll also keep precision and recall in the model results for reference.

Generate training and test sets

The following code is used to generate the training set and test set from bank1:

```
set.seed(5, sample.kind = "Rounding")
test_index_1 <- createDataPartition(y = bank1$Exited, times = 1, p = 0.1, list = FALSE)
test_set <- bank1[test_index_1,]
train_set <- bank1[-test_index_1,]
```

The train set contains 8098 observations and the test set contains 901.

```
dim(train_set)
```

```
## [1] 8098   11
```

```
dim(test_set)
```

```
## [1] 901  11
```

glm with caret

Run the model

First, we run a glm model with the caret package by setting "Exited" as the response variable.

```
control_glm <- trainControl(summaryFunction = prSummary,
                            classProbs = TRUE,
                            savePredictions = T,
                            method = "none"
                            )

set.seed(123, sample.kind = "Rounding")
train_glm <- train(Exited ~.,
                   data = train_set,
                   method = "glm",
                   trControl = control_glm)
```

Find the best cutoff

In the data exploration we found that our data set is not balanced: only about 20% the observations exited the bank. If we use the model to predict directly the expected "class" on the test set, the automatic process would choose 0.5 as the threshold for "yes" or "no", which might not be optimal for generating the highest F1-score for our specific case. Therefore, we'll choose to predict the probability of "Exited = yes" on the test set, then try to find the best cutoff manually.

```
pred_glm <- predict(train_glm, test_set, type = "prob")
head(pred_glm)
```

```
##          yes        no
## 1 0.22919062 0.7708094
## 2 0.05320823 0.9467918
## 3 0.06284752 0.9371525
## 4 0.38260944 0.6173906
## 5 0.19401522 0.8059848
## 6 0.46633360 0.5336664
```

In order to get the best cutoff, we create the PR curve of every threshold, then calculate the F1-score with precision and recall. Then we arrange the result by F1-score.

```
glm_cutoffs <- tibble(obs = test_set$Exited,
                      pred = pred_glm$`yes`) %>%
  pr_curve(obs, pred) %>%
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  arrange(-f1)
head(glm_cutoffs)
```

```
## # A tibble: 6 x 4
##   .threshold recall precision    f1
##        <dbl>  <dbl>     <dbl> <dbl>
## 1      0.284  0.538     0.463 0.497
## 2      0.287  0.533     0.467 0.497
## 3      0.237  0.614     0.417 0.497
## 4      0.284  0.538     0.460 0.496
## 5      0.287  0.533     0.464 0.496
## 6      0.235  0.620     0.413 0.496
```

```
glm_best_cutoff <- pull(glm_cutoffs[1,1])
```

The best cutoff of glm model is 0.2836466.

Predict and calculate F1

Now that we have found the best cutoff, we can get the expected value by assigning "yes" when the probability is equal or higher than the best cutoff, and assigning "no" when the probability is lower than the best cutoff.

```
y_hat_glm <- factor(ifelse(pred_glm$`yes` >= glm_best_cutoff, "yes", "no"),
                    levels = c("yes", "no"))
```

With the following code, we calculate the F1-score of the glm model:

```
cm_glm <- confusionMatrix(data = y_hat_glm,
                          reference = test_set$Exited,
                          mode = "prec_recall", positive = "yes")

glm_result <- tibble(Model = "glm",
```

```
                      Cutoff = glm_best_cutoff,
                      F1 = cm_glm$byClass["F1"],
                      Precision = cm_glm$byClass["Precision"],
                      Recall = cm_glm$byClass["Recall"])
glm_result
```

```
## # A tibble: 1 x 5
##   Model Cutoff    F1 Precision Recall
##   <chr>  <dbl> <dbl>     <dbl>  <dbl>
## 1 glm    0.284 0.497     0.463  0.538
```

ranger with caret

Ranger is a fast implementation of Random Forests, which is suitable for classifications. We'll choose ranger as our second model.

Train the model with 5-fold cross validation

The caret package allows us to tune three parameters of ranger: mtry, min.node.size, and splitrule. We'll set up a tune grid and use 5-fold cross validation for the training.

```
control_ranger <- trainControl(method = "cv",
                               number = 5,
                               p = 0.8,
                               summaryFunction = prSummary,
                               classProbs = TRUE,
                               savePredictions = T,
                               verboseIter = T,
                               allowParallel = T
                               )

set.seed(123, sample.kind = "Rounding")
train_ranger <- train(Exited ~.,
                      data = train_set,
                      method = "ranger",
                      num.trees = 500,
                      tuneGrid = expand.grid(mtry = 1:10,
                                             min.node.size = seq(10,50,10),
                                             splitrule = "gini"),
                      metric = "F",
                      trControl = control_ranger)
```

Find the best tune parameters

With the following code, we are supposed to get the best tuning parameters. But we face the same problem as in the glm model: the cutoff of this process is not the best, therefore, the "bestTune" parameters we get with the following code are very likely not the correct ones.

```
train_ranger$bestTune
```

```
##    mtry splitrule min.node.size
## 37    8      gini            20
```

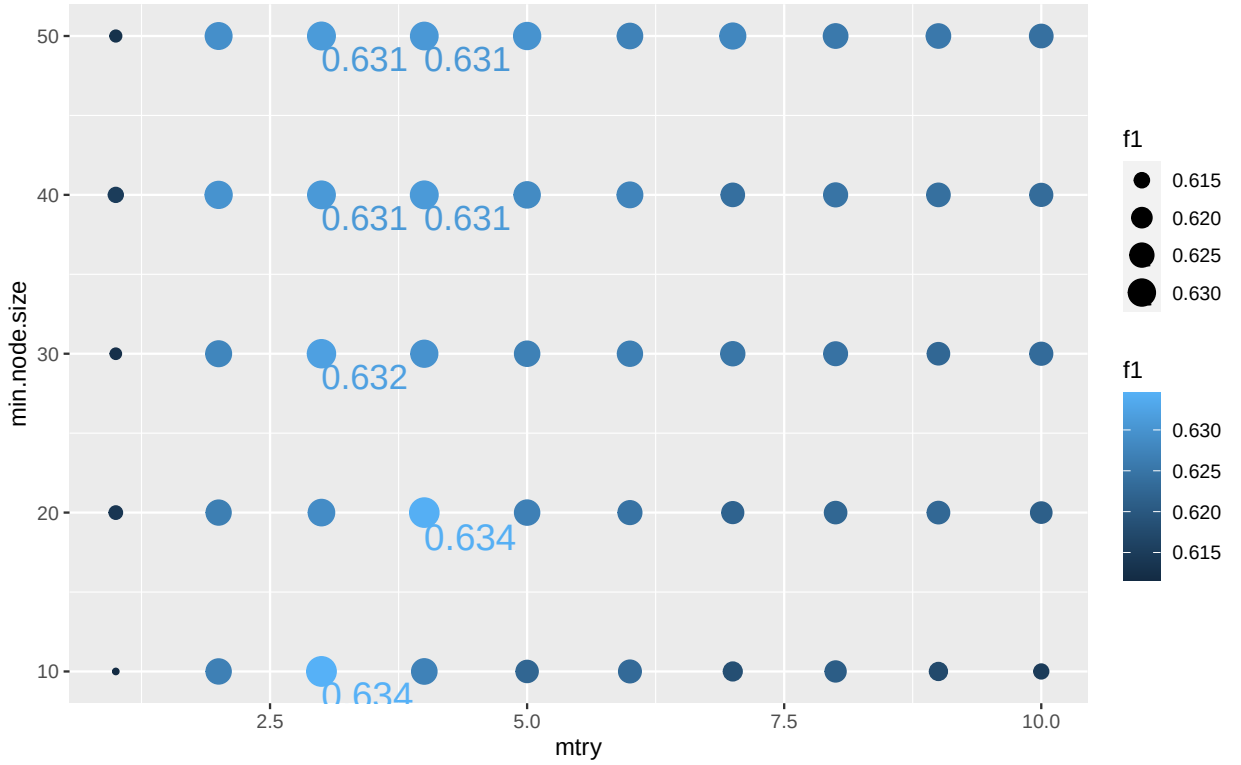
In order to find the true best tuning parameters, we need to calculate the F1 score of each tuning combination.

```
pred_ranger <- train_ranger$pred

f1_scores_ranger <- pred_ranger %>%
  group_by(mtry, min.node.size, Resample) %>%
  pr_curve(obs, yes) %>%
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  slice_max(f1, n = 1) %>%
  group_by(mtry, min.node.size) %>%
  summarise(f1 = mean(f1))
```

We can plot "mtry" against "min.node.size", and look for the trend of F1 score. We notice that the highest F1-scores appear when both mtry and min.node.size are small.

```
f1_scores_ranger %>%
  ggplot(aes(mtry, min.node.size, color = f1, size = f1, label = f1)) +
  geom_point() +
  geom_text(aes(label=ifelse(f1>0.630,round(f1, digits = 3),'')),
            hjust=0,
            vjust=1.4)
```



We can arrange the result by F1-score:

```r
f1_scores_ranger_desc <- f1_scores_ranger %>% arrange(-f1)
head(f1_scores_ranger_desc)
```

```
## # A tibble: 6 x 3
## # Groups:   mtry [2]
##    mtry min.node.size    f1
##   <int>         <dbl> <dbl>
## 1     3            10 0.634
## 2     4            20 0.634
## 3     3            30 0.632
## 4     3            50 0.631
## 5     4            40 0.631
## 6     3            40 0.631
```

Then we can find the true best mtry is 3, and best min.node.size is 10.

```r
best_mtry_ranger <- as.numeric(f1_scores_ranger_desc[1,1])
best_mtry_ranger
```

```
## [1] 3
```

```r
best_min.node.size_ranger <- as.numeric(f1_scores_ranger_desc[1,2])
best_min.node.size_ranger
```

```
## [1] 10
```

Re-run the model with the true best tune parameters

With the following code, we re-run the model with the best parameters(mtry = 3. min.node.size = 10):

```r
Control_ranger_best <- trainControl(method = "none",
                                    summaryFunction = prSummary,
                                    classProbs = TRUE,
                                    savePredictions = T,
                                    verboseIter = T,
                                    allowParallel = T
                                    )

set.seed(123, sample.kind = "Rounding")
train_ranger_best <- train(Exited ~.,
                           data = train_set,
                           method = "ranger",
                           num.trees = 500,
                           tuneGrid = expand.grid(mtry = best_mtry_ranger,
                                                  min.node.size = best_min.node.size_ranger,
                                                  splitrule = "gini"),
                           metric = "F",
                           trControl = Control_ranger_best)
```

```
## Fitting mtry = 3, min.node.size = 10, splitrule = gini on full training set
```

Find the best cutoff

We'll manually calculate the F1-score with precision and recall of every cutoff value.

```
prob_ranger <- predict(train_ranger_best, test_set, type = 'prob')

ranger_cutoffs <- tibble(obs = test_set$Exited,
                         pred = prob_ranger$`yes`) %>%
  pr_curve(obs, pred) %>%
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  arrange(-f1)
head(ranger_cutoffs)
```

```
## # A tibble: 6 x 4
##   .threshold recall precision    f1
##        <dbl>  <dbl>     <dbl> <dbl>
## 1      0.404  0.549     0.765 0.639
## 2      0.435  0.533     0.797 0.638
## 3      0.401  0.549     0.759 0.637
## 4      0.432  0.533     0.790 0.636
## 5      0.400  0.549     0.754 0.635
## 6      0.406  0.543     0.763 0.635
```

```
ranger_best_cutoff <- pull(ranger_cutoffs[1,1])
```

The best cutoff is 0.4035559.

Predict and calculate F1

We'll get the expected value with the best cutoff, and then calculate the F1-score of the ranger model.

The F1-score of ranger is much higher than the glm model.

```
y_hat_ranger <- factor(ifelse(prob_ranger$`yes` >= ranger_best_cutoff, "yes", "no"),
                       levels = c("yes", "no"))

cm_ranger <- confusionMatrix(data = y_hat_ranger,
                             reference = test_set$Exited,
                             mode = "prec_recall", positive = "yes")

ranger_result <- tibble(Model = "ranger",
                        Cutoff = ranger_best_cutoff,
                        F1 = cm_ranger$byClass["F1"],
                        Precision = cm_ranger$byClass["Precision"],
                        Recall = cm_ranger$byClass["Recall"])
ranger_result
```

```
## # A tibble: 1 x 5
##   Model  Cutoff    F1 Precision Recall
##   <chr>   <dbl> <dbl>     <dbl>  <dbl>
## 1 ranger  0.404 0.639     0.765  0.549
```

KNN with caret

The KNN model (k-nearest neighbors algorithm) is mostly used for classification tasks. It classifies the data point based on how the neighboring points are classified. We'll choose KNN as our third model.

Train the model with 5-fold cross validation

The caret package allows us to tune "k" for KNN. There are about 8000 observations in the train set. A good way to find the best k value is to use the square root of the number of observations in a data set. In our case, the suitable k should be around 90, so we'll create a sequence of number from 3 to 153 as the tune grid for k, to be sure that we cover the range of the best k.

First, we train a model with 5-fold cross validation:

```
control_knn <- trainControl(method = "cv",
                            number = 5,
                            p = 0.8,
                            summaryFunction = prSummary,
                            classProbs = TRUE,
                            savePredictions = T,
                            verboseIter = T,
                            allowParallel = T
                            )

set.seed(123, sample.kind = "Rounding")
train_knn <- train(Exited ~.,
                   data = train_set,
                   method = "knn",
                   preProcess = c("center", "scale"),
                   tuneGrid = data.frame(k = seq(3, 153, 2)),
                   metric = "F",
                   trControl = control_knn)
```

Find the best k

According to the automatic process, the best k is 3.

```
train_knn$bestTune
```

```
##   k
## 1 3
```

This might not be the true best k because of the unbalanced data set. Plus, with k=3, the value is so small that we might run the risk of over-training.

We can calculate the F1-score of each k value with the following code:

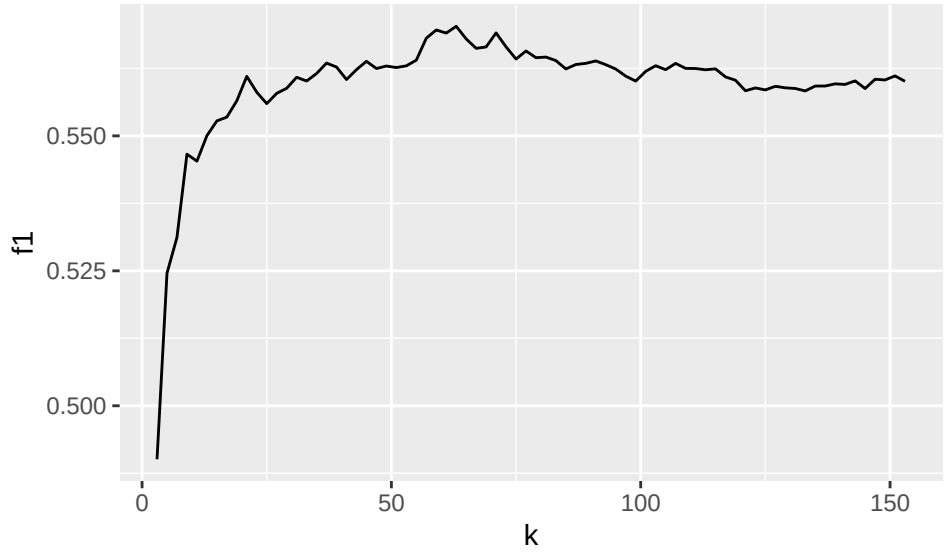
```
pred_knn <- train_knn$pred

f1_scores_knn <- pred_knn %>%
  group_by(k, Resample) %>%
  pr_curve(obs, yes) %>%
```

```
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  slice_max(f1, n = 1) %>%
  group_by(k) %>%
  summarise(f1 = mean(f1))
```

we can plot k against F1:

```
f1_scores_knn %>%
  ggplot(aes(k, f1)) +
  geom_line()
```



From the plot we can see that F1 increases as k increases, but when k is bigger than 50, F1 mainly fluctuates within a small range.This indicates that we have used a reasonable sequence of k in the training process.

The following code gives us the best k: 63.

```
f1_scores_knn_desc <- f1_scores_knn %>% arrange(-f1)
best_k_knn <- as.numeric(f1_scores_knn_desc[1,1])
best_k_knn
```

```
## [1] 63
```

Re-run the model with the true best k

With the following code, we re-run the model with the best parameter(k = 63):

```
control_knn_best <- trainControl(method = "none",
                                 summaryFunction = prSummary,
                                 classProbs = TRUE,
                                 savePredictions = T,
                                 verboseIter = T,
                                 allowParallel = T
                                 )
```

```
set.seed(123, sample.kind = "Rounding")
train_knn_best <- train(Exited ~.,
                        data = train_set,
                        method = "knn",
                        preProcess = c("center", "scale"),
                        tuneGrid = data.frame(k = best_k_knn),
                        metric = "F",
                        trControl = control_knn_best)
```

```
## Fitting k = 63 on full training set
```

Find the best cutoff

We'll manually calculate the F1-score with precision and recall of every cutoff value.

```
prob_knn <- predict(train_knn_best, test_set, type = 'prob')

knn_cutoffs <- tibble(obs = test_set$Exited,
                      pred = prob_knn$`yes`) %>%
  pr_curve(obs, pred) %>%
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  arrange(-f1)

knn_best_cutoff <- pull(knn_cutoffs[1,1])
```

The best cutoff for KNN model is 0.2698413.

Predict and calculate F1

We'll get the expected value with the best cutoff, and then calculate the F1-score of the KNN model.

The F1-score of KNN is not as good as ranger.

```
y_hat_knn <- factor(ifelse(prob_knn$`yes` >= knn_best_cutoff, "yes", "no"),
                    levels = c("yes", "no"))

cm_knn <- confusionMatrix(data = y_hat_knn,
                          reference = test_set$Exited,
                          mode = "prec_recall", positive = "yes")

knn_result <- tibble(Model = "knn",
                     Cutoff = knn_best_cutoff,
                     F1 = cm_knn$byClass["F1"],
                     Precision = cm_knn$byClass["Precision"],
                     Recall = cm_knn$byClass["Recall"])
knn_result
```

```
## # A tibble: 1 x 5
##   Model Cutoff    F1 Precision Recall
##   <chr>  <dbl> <dbl>     <dbl>  <dbl>
## 1 knn    0.270 0.574     0.535  0.620
```

LightGBM

LightGBM(Light Gradient Boosting Machine) is a gradient boosting framework that uses learning algorithms based on trees. It's very fast and efficient in training and requires less memory. We'll choose it as our fourth model.

Format the data set

Firstly, we need to convert all factor variables(Geography, Gender, HasCrCard, IsActiveMember, Exited) to numeric.

```
train_set_lgbm <- train_set
train_set_lgbm$Geography <- as.numeric(train_set_lgbm$Geography)
train_set_lgbm$Gender <- as.numeric(train_set_lgbm$Gender)
train_set_lgbm$HasCrCard <- as.numeric(train_set_lgbm$HasCrCard) - 1
train_set_lgbm$IsActiveMember <- as.numeric(train_set_lgbm$IsActiveMember) - 1
train_set_lgbm$Exited <- ifelse(as.numeric(train_set_lgbm$Exited) == 2, 0, 1)

test_set_lgbm <- test_set
test_set_lgbm$Geography <- as.numeric(test_set_lgbm$Geography)
test_set_lgbm$Gender <- as.numeric(test_set_lgbm$Gender)
test_set_lgbm$HasCrCard <- as.numeric(test_set_lgbm$HasCrCard) - 1
test_set_lgbm$IsActiveMember <- as.numeric(test_set_lgbm$IsActiveMember) - 1
test_set_lgbm$Exited <- ifelse(as.numeric(test_set_lgbm$Exited) == 2, 0, 1)
```

Next, we consider the response variable "Exited" as "label", and define the other four variables as categorical features.

```
categoricals_vector  <- c(2, 3, 8, 9)
```

Then we split train set and test set into data sets that contain feature and label, and load them into LightGBM dataset object.

```
train_set_x <- as.matrix(train_set_lgbm[, -11])
train_set_y <- as.matrix(train_set_lgbm[, 11])

test_set_x <- as.matrix(test_set_lgbm[, -11])
test_set_y <- as.matrix(test_set_lgbm[, 11])

dtrain <- lgb.Dataset(data = train_set_x,
                      label = train_set_y,
                      params = list(categorical_feature = categoricals_vector))

dtest <- lgb.Dataset.create.valid(dtrain,
                                  data = test_set_x,
                                  label = test_set_y,
                                  params = list(categorical_feature = categoricals_vector))
```

Train a model

First, we define the parameters of the model. Because F1-score is not an available metric of LightGBM, we'll choose "average precision score" instead:

```
train_params <- list(
  objective= 'binary',
  is_unbalance = TRUE,
  metric = "average_precision",
  num_iterations = 100,
  learning_rate = 0.1
  )
```

We run a 5-fold cross-validation and find the best iteration.

```
set.seed(123, sample.kind = "Rounding")
lgm_cv <- lgb.cv(params = train_params,
                 data = dtrain,
                 nfold = 5,
                 stratified = TRUE)
```

```
best_iter <- lgm_cv$best_iter
best_iter
```

```
## [1] 62
```

Re-run the model with best iteration

With the following code, we re-run the model with the best iteration.

```
set.seed(123, sample.kind = "Rounding")
train_lgbm <- lgb.train(params = train_params,
                        data = dtrain,
                        nrounds = best_iter,
                        verbose = 2)
```

Find the best cutoff

we'll manually calculate the F1-score of every threshold.

```
pred_lgbm <- predict(train_lgbm, test_set_x)

lgbm_cutoffs <- tibble(obs = test_set$Exited,
                       pred = pred_lgbm) %>%
  pr_curve(obs, pred) %>%
  mutate(f1 = (2 * precision * recall) / (precision + recall)) %>%
  arrange(-f1)

lgbm_best_cutoff <- pull(lgbm_cutoffs[1,1])
```

The best cutoff is 0.6324961.

Predict and calculate F1

We'll get the expected value with the best cutoff, and then calculate the F1-score of the LightGBM model. The F1-score of LightGBM is better than KNN but not as good as ranger.

```
y_hat_lgbm <- factor(ifelse(pred_lgbm >= lgbm_best_cutoff, 1, 0), levels = c(0, 1))

cm_lgbm <- confusionMatrix(data = y_hat_lgbm,
                           reference = factor(test_set_lgbm$Exited, levels = c(0, 1)),
                           mode = "prec_recall", positive = "1")

lgbm_result <- tibble(Model = "LightGBM",
                      Cutoff = lgbm_best_cutoff,
                      F1 = cm_lgbm$byClass["F1"],
                      Precision = cm_lgbm$byClass["Precision"],
                      Recall = cm_lgbm$byClass["Recall"])
lgbm_result
```

```
## # A tibble: 1 x 5
##   Model    Cutoff    F1 Precision Recall
##   <chr>     <dbl> <dbl>     <dbl>  <dbl>
## 1 LightGBM  0.632 0.614     0.619  0.609
```

Results and Final Validation

Model results

From the table below, we can see that ranger yielded the highest F1 score, therefore we'll choose ranger as the model for final validation.

```
rbind(glm_result, ranger_result, knn_result,lgbm_result)
```

```
## # A tibble: 4 x 5
##   Model    Cutoff    F1 Precision Recall
##   <chr>     <dbl> <dbl>     <dbl>  <dbl>
## 1 glm       0.284 0.497     0.463  0.538
## 2 ranger    0.404 0.639     0.765  0.549
## 3 knn       0.270 0.574     0.535  0.620
## 4 LightGBM  0.632 0.614     0.619  0.609
```

Final validation

We'll use the bank1 data set(train set and test set combined) as the final train set, and validation as the final test set. We'll also use the best mtry and min.node.size we found in model building phase.

```
control_final <- trainControl(method = "none",
                              summaryFunction = prSummary,
                              classProbs = TRUE,
                              savePredictions = T,
```

```
                              verboseIter = T,
                              allowParallel = T
                              )

set.seed(123, sample.kind = "Rounding")
train_ranger_final <- train(Exited ~.,
                            data = bank1,
                            method = "ranger",
                            num.trees = 500,
                            tuneGrid = expand.grid(mtry = best_mtry_ranger,
                                                   min.node.size = best_min.node.size_ranger,
                                                   splitrule = "gini"),
                            metric = "F",
                            trControl = control_final)
```

```
## Fitting mtry = 3, min.node.size = 10, splitrule = gini on full training set
```

Then we predict with the best cutoff that we found previously and calculate the final F1-score.

```
prob_ranger_final <- predict(train_ranger_final, validation, type = 'prob')

y_hat_final <- factor(ifelse(prob_ranger_final$`yes` >= ranger_best_cutoff, "yes", "no"),
                      levels = c("yes", "no"))

cm_final <- confusionMatrix(data = y_hat_final,
                            reference = validation$Exited,
                            mode = "prec_recall", positive = "yes")

final_validation_result <- tibble(Model = "ranger",
                                  Cutoff = ranger_best_cutoff,
                                  F1 = cm_final$byClass["F1"],
                                  Precision = cm_final$byClass["Precision"],
                                  Recall = cm_final$byClass["Recall"])
final_validation_result
```

```
## # A tibble: 1 x 5
##   Model  Cutoff    F1 Precision Recall
##   <chr>   <dbl> <dbl>     <dbl>  <dbl>
## 1 ranger  0.404 0.573     0.654  0.510
```

Conclusion

For this project, we used four models to analysis and predict churning behavior of bank customers. The tuned ranger model is the best-performing model, which yielded a F1-score of 0.573 in the final validation.

This project has a few limitations:

1. The data set is relatively small: There are only 10000 observations in the whole data set. The size of the data set limited the the performance of the models, which might explain the big difference in testing (F1 = 0.639) and validation (F1 = 0.573).

2. The tuning of parameters. Due to limited time and resources, we only tuned the most important parameters. For ranger model, we tuned mtry and min.node.size, but tuning other parameters, for example the split rule and num of trees, may improve the model's performance. As for the LightGBM model, it's easy and fast to run but the parameter tuning is quite complicated. Since the result of LightGBM without tuning is already quite good (F1 = 0.614), there's a pretty good chance that with fine tuning, it'll outperform ranger .

3. Lack of background information. More information of the data, for example, how "IsActiveMember" variable is defined, how "EstimatedSalary" is estimated, etc., could help us understand the data much better, or even lead to development of new algorithms.

With more time and resources, a few more approaches could be implemented to improve the project:

1. Other algorithms, for instance XGBoost, can be tested.

2. The models can be improved by testing and tuning on bigger data sets or data sets of other industries.

3. Input from sales and marketing perspectives may help understand the dynamics of variables.