Churn Case

Mette Aygün

2024-07-08

Table of contents

Churn rates

1. Introduction

This case focuses on customer churn (also known as customer attrition or customer turnover). Customer churn is interesting because it is usually much cheaper to retain existing customers than to acquire new ones. Instead of focusing on each individual customer, we will attempt to build a predictive model that can help us decide which customers we should focus our retention efforts on. The dataset has been downloaded from https://www.kaggle.com/datasets/santoshd3/bank-customers, and is free to use.

2. Analysis

I follow the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework in my data mining projects, guiding me through six phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment. This structured approach ensures I effectively extract insights and apply data science.

2.1 Business Understanding

Understanding customer churn is critical for banks as it aids in cost reduction by prioritizing the retention of existing customers over acquiring new ones. This not only helps in stabilizing revenue but also enhances customer satisfaction by addressing their specific needs and concerns. By gaining insights into churn patterns, banks can develop targeted strategies, optimize resource allocation, and gain a competitive edge in the market.

2.2 Data Understanding

2.2.1 Reading libraries

We will start by loading the nessescary libraries

```
pacman::p_load("tidyverse", "magrittr", "nycflights13", "gapminder",
               "Lahman", "maps", "lubridate", "pryr", "hms", "hexbin",
               "feather", "htmlwidgets", "broom", "pander", "modelr",
               "XML", "httr", "jsonlite", "lubridate", "microbenchmark",
               "splines", "ISLR2", "MASS", "testthat",  "caret",
               "RSQLite", "class", "babynames", "nasaweather", "pls",
               "fueleconomy", "viridis", "boot", "devtools", "tree", "leaps",
               "glmnet", "gam", "akima", "factoextra", "randomForest", "gbm",
               "ggrepel", "GGally", "fmsb", "sjPlot", "rcompanion", "DT")
# Installer nødvendige pakker, hvis du ikke allerede har dem

# Indlæs pakkerne
library(pROC)
library(caret)
library(MASS)
library(gbm)

options(repos = c(CRAN = "https://cran.rstudio.com/"))
```

2.2.2 Importing data

The dataset we are going to work with will be imported and investigated.

```
#bank_churn <- read.csv("Churn_Modelling.csv")
bank_churn <- read.csv("C:/Users/mette/OneDrive/Skrivebord/PB
dataanalyse/Programmering og statistical
learning/data/Portfolio/Churn_Modelling.csv")

bank_churn_bi <- bank_churn
bank_churn1 <- bank_churn
bank_churn_lasso <- bank_churn

#tjekker data og klasser
str(bank_churn)
```

2.3 Data Preparation

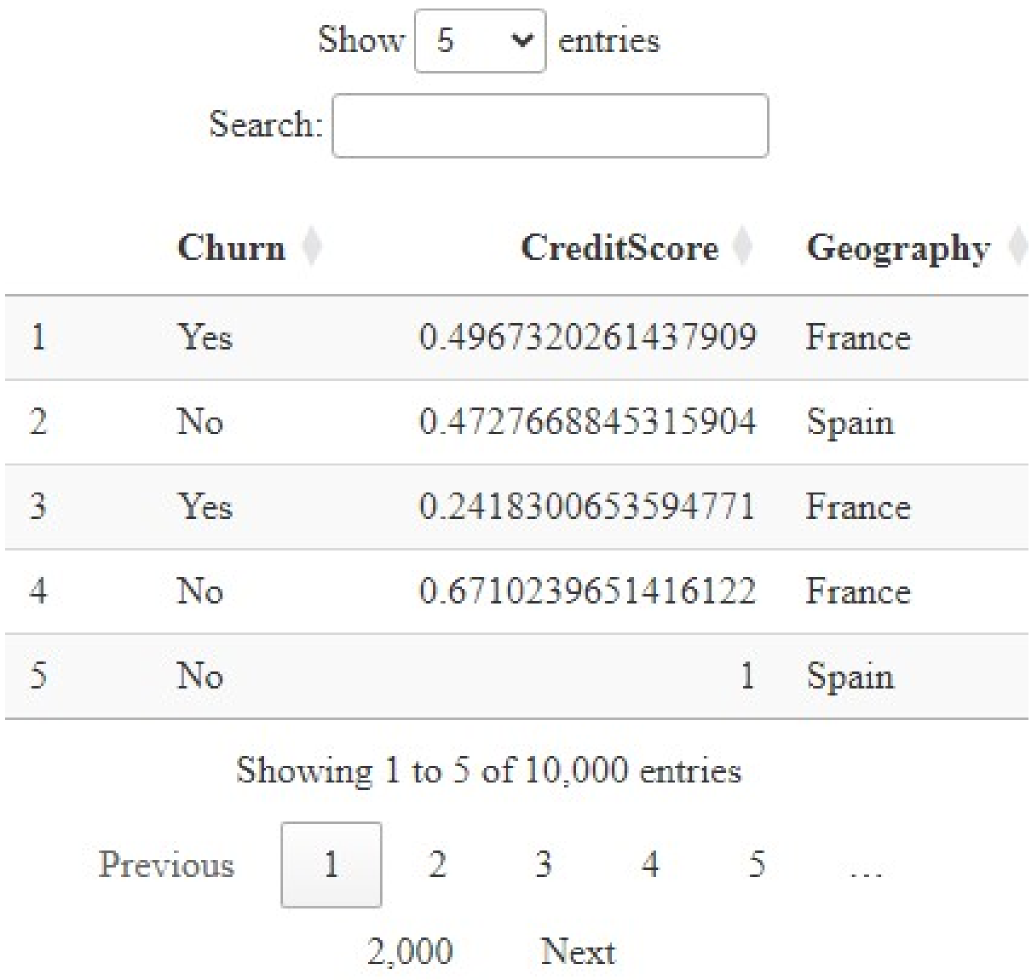
2.3.1 Cleaning data

First we are checking for missing values. There are no missing values in the TotalCharges variable. We are imputing thesevalues with the meanvalue, since the quantity is low. After this we are changes all the character class variables to factors for later statistical analysis. The data is normalised and finally the churndataset is again changes to a dataframe and the CustomerID variable is removed.

```r
# Beregn antallet af missing værdier i hver kolonne. No missing
bank_churn %>% purrr::map(~ sum(is.na(.)))

summary(bank_churn)

# There are no missing values, so we can proceed.

#the relevant variables are converted into factors.

# Type = factor and integers------------------------------------------------------
-----

bank_churn_fact <- bank_churn %>%
  mutate_if(is.character, as.factor) %>%
  mutate_if(is.integer, as.factor)

str(bank_churn_fact)
bank_churn_fact$CreditScore <- as.integer(bank_churn_fact$CreditScore)
bank_churn_fact$Age <- as.integer(bank_churn_fact$Age)
bank_churn_fact$Tenure <- as.integer(bank_churn_fact$Tenure)
str(bank_churn_fact)

# Normalisering -----------------------------------------------------------

# Det er ikke nødvendigt at normalisere data i forbindelse med de statistiske
# modeller, som vi skal køre her. Der er forskellige typer af normalisering.
# Vi ser her på følgende:

normalize <- function(x) {
  ((x-min(x))/(max(x)-min(x)))
}

bank_churn_fact <- bank_churn_fact %>%
  mutate_if(is.numeric, normalize)

bank_churn_fact <- bank_churn_fact %>%
  dplyr::select(Exited, everything())

glimpse(bank_churn_fact)

#numeric_columns <- sapply(bank_churn_fact, is.numeric)

# Konverter numeriske variable fra dbl til int
#bank_churn_fact[numeric_columns] <- lapply(bank_churn_fact[numeric_columns],
as.integer)

# Fravalg af customerID, Efternavn og ID ----------------------------------------
----------

bank_churn_fact <- bank_churn_fact %>%
  dplyr::select(-RowNumber, -CustomerId, -Surname)

names(bank_churn_fact)

#Ændre variablen Exited til Churn

bank_churn_fact <- bank_churn_fact %>%
  rename(Churn = Exited)

bank_churn_fact$Churn <- ifelse(bank_churn_fact$Churn == 1, "Yes", "No")
bank_churn_fact$Churn <- as.factor(bank_churn_fact$Churn)
str((bank_churn_fact))
```

This creates a dataset with 10.000 observations, that can be investigated and is ready for analysis.

```r
# Install DT package
install.packages("DT")

# Load DT package
library(DT)

datatable(bank_churn_fact,
          options = list(pageLength = 5, autoWidth = TRUE),
          caption = 'Table 1: Telecommunication data')
```


Show 5 entries

Search:

	Churn	CreditScore	Geography
1	Yes	0.4967320261437909	France
2	No	0.4727668845315904	Spain
3	Yes	0.2418300653594771	France
4	No	0.6710239651416122	France
5	No	1	Spain

Showing 1 to 5 of 10,000 entries

Previous 1 2 3 4 5 … 2,000 Next

2.4 Modeling

Training and test data

```
#træningsdata og testdata


# Vi bruger funktionen set.seed, så vi kan reproducere vores resultater
set.seed(5)
# træningsdel og testdel:
intrain <- createDataPartition(y=bank_churn_fact$Churn,
                               p=0.70, list=FALSE)
```

```
# list=FALSE betyder at outputtet bliver en matrix og denne kan bruges
# i koden nedenfor:

train <- bank_churn_fact[intrain,]
test <- bank_churn_fact[-intrain,]
```

Cost Assessment

The cost assessment is significant when deciding on the threshold in connection with, for example, logistic regression. The relative cost of the different errors that can be made affects where it is optimal to place the threshold. Optimal in terms of reducing costs. When we do not have any a priori knowledge about the relative costs, we use a 50/50 split. But in this example, the situation is different. It does not cost the same to commit the different errors.

```
#                        |  Vil churne   | vil ikke churne
# predikte churne        |      TP       |      FP
# predikte ikke churne   |      FN       |      TN
```

Customer acquisition $200 Se documentation here - the cost for a false negative (FN) prediction That is, predicting that a customer is satisfied when in reality they churn. Customer retention $40 (Source: Bain & Company, “The Value of Online Customer Loyalty in Retail Banking,” 2016.) - the cost of a false positive (FP) That is, predicting that a customer will churn when in reality the customer was satisfied, and a true positive (TP) that is, correctly predicting dissatisfied customers. Correctly predicted true negatives (TN) cost nothing. That is, correctly predicting that a customer is satisfied.

The total savings from the new model based on a customer base of 10.000 customers:

```
FN_omk <- 200

TP_omk <- 40

FP_omk <- TP_omk

TN_omk <- 0
```

2.4.1 K-Means

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping clusters. The algorithm iteratively assigns each data point to the nearest cluster center and recalculates the center based on the mean of all points assigned to that cluster. This process continues until convergence, aiming to minimize the within-cluster sum of squares. K-means is widely used for clustering analysis in various fields, such as image segmentation, customer segmentation, and anomaly detection.

In the following code K-Means clustering are used to form 3 clusters that are added to the orginal datasat in order to se if the clusters will make the model better.

```r
bank_churn_kmeans <- bank_churn

str(bank_churn_kmeans)

bank_churn_kmeans <- bank_churn %>%
  dplyr::select(Exited, everything()) %>%
  dplyr::rename(Churn = Exited) %>%
  mutate(Churn = ifelse(Churn == 1, "Yes", "No"),
         Churn = as.factor(Churn),
         IsActiveMember = ifelse(IsActiveMember == "1", "Yes", "No"),
         HasCrCard = ifelse(HasCrCard == "1", "Yes", "No")) %>%
  dplyr::select(-RowNumber, -CustomerId, -Surname, -Churn)

str(bank_churn_kmeans)

# Specifikt standardisere de valgte kolonner
specific_columns <- c("NumOfProducts","CreditScore", "Age", "Tenure", "Balance",
"EstimatedSalary")
Standard <- bank_churn_kmeans
Standard[specific_columns] <- scale(bank_churn_kmeans[specific_columns])

view(Standard)
# Indlæs caret pakken
library(caret)

# Opret et dummyVars objekt, specificer dit datasæt
# note: til ~ . betyder, at alle variabler bliver behandlet, men du kan også
specificere specifikke variabler
dummies <- dummyVars(~ ., data = Standard)

# Anvend dummyVars objektet til dit datasæt for at skabe de One-Hot Encoded
variabler
encoded_data <- predict(dummies, newdata = Standard)

# Konverter til en dataframe, hvis nødvendigt
encoded_data <- as.data.frame(encoded_data)

# Vis de første par rækker for at tjekke resultatet
head(encoded_data)

view(encoded_data)
str(encoded_data)

library(cluster)  # For silhouette analysis
library(factoextra)  # For visualizing clusters and elbow method

#elbow metoden for at bestemme det optimale antal clusters
```

```r
set.seed(123)  # Sikrer reproducerbarhed
wss <- numeric(20)  # WSS for k fra 1 til 20

for (k in 1:20) {
  model <- kmeans(encoded_data, centers = k, nstart = 25)
  wss[k] <- model$tot.withinss
}

plot(1:20, wss, type = "b", xlab = "Antal af Clusters", ylab = "Total WSS", main =
"Elbow Metode")

#det optimale antale clusters er 3, da kurven efter k=3 begynder at flade ud.


library(stats)

# Antager at dit standardiserede datasæt er gemt i et objekt kaldet Standard
set.seed(123)  # For reproducerbarhed
k_optimal <- 3  # Det antal clusters du har valgt

# Træn k-means modellen med det optimale antal clusters
kmeans_model <- kmeans(encoded_data, centers = k_optimal, nstart = 25)

# Se resultaterne
print(kmeans_model)
print(kmeans_model$centers)

# Tilføj cluster-tilhørsforhold til dit datasæt
bank_churn_fact$clusterKmeans <- as.factor(kmeans_model$cluster)
bank_churn_kmeans$clusterKmeans <- kmeans_model$cluster
bank_churn_lasso$clusterKmeans <- kmeans_model$cluster
bank_churn1$clusterKmeans <- kmeans_model$cluster
# Se de første par rækker for at bekræfte tilføjelsen af cluster-tilhørsforhold
head(bank_churn_kmeans)


#par(mfrow=c(1,1))
library(reshape2)

centers_long <- melt(kmeans_model$centers)
ggplot(centers_long, aes(x = Var2, y = value, fill = Var1)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "Feature", y = "Centroid Value", fill = "Cluster")
```

2.4.2 Hierachial Clustering

Hierarchical clustering is a method of cluster analysis that builds a hierarchy of clusters. It starts by considering each data point as a separate cluster and then iteratively merges the closest clusters based on a chosen distance metric until all points belong to a single cluster. Hierarchical clustering can be agglomerative, where clusters are successively merged together, or divisive, where clusters are successively divided. It results in a dendrogram, which visually represents the merging process and allows for the exploration of different levels of granularity in the clustering.

In the following code Hierachial Clustering is used to form 4 clusters that are added to the orginal datasat in order to se if the clusters will make the model better.

```
#Hierakisk clustering

hc.complete <- hclust(dist(encoded_data), method="complete")
hc.average <- hclust(dist(encoded_data), method="average")
hc.single <- hclust(dist(encoded_data), method="single")

# Opdel pladsen til at vise plots
par(mfrow=c(1,3))

# Plot hierarkiske klynger for forskellige linkages
plot(hc.complete, main="Complete Linkage", xlab="", sub="", cex=.9)
plot(hc.average, main="Average Linkage", xlab="", sub="", cex=.9)
plot(hc.single, main="Single Linkage", xlab="", sub="", cex=.9)


segments(0, 2, nrow(encoded_data), 2, col="red")

# Udskriv klyngerne for forskellige linkages
cutree(hc.complete, 3)
cutree(hc.average, 2)
cutree(hc.single, 2)
cutree(hc.single, 4)


#vi vælger complete linkage, k=4

#visualisering
library(reshape2)
library(ggplot2)

# Eksempeldata for klyngemodel
hc.complete <- hclust(dist(encoded_data), method="complete")
cluster_labels <- cutree(hc.complete, k = 3)

# Beregn klyngecentre
```

```
centers <- aggregate(encoded_data, by=list(cluster_labels), FUN=mean)

# Navngiv kolonner
colnames(centers)[-1] <- colnames(encoded_data)

# Lav en dataframe for klyngecentre i "long" format
centers_long <- melt(centers, id.vars="Group.1")

# Plot klyngecentre
ggplot(centers_long, aes(x = variable, y = value, fill = factor(Group.1))) +
  geom_bar(stat = "identity", position = "dodge") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "Feature", y = "Centroid Value", fill = "Cluster")

hc.complete <- hclust(dist(encoded_data), method="complete")
cluster_labels <- cutree(hc.complete, k = 4)

# Tilføj klyngemærker som en ekstra kolonne til dit datasæt
bank_churn_fact$ClusterHC <- factor(cluster_labels)
bank_churn_lasso$clusterKmean <-factor(cluster_labels)
bank_churn1$clusterHC <- factor(cluster_labels)

str(bank_churn_fact)
```

2.4.3 Choosing relevant variables - Lasso

In order to evalutate the relevant variables we will perform Lasso modelling. Lasso modeling applies regularization to regression, shrinking less important feature coefficients to zero. This effectively performs variable selection, highlighting the most relevant predictors for the outcome. It’s particularly useful in scenarios with many variables, aiding in identifying the most impactful ones while reducing model complexity and overfitting.

```
bank_churn_lasso <- bank_churn_lasso %>%
  dplyr::select(-RowNumber, -CustomerId, -Surname)

names(bank_churn_lasso)

#Ændre variablen Exited til Churn

bank_churn_lasso <- bank_churn_lasso %>%
  dplyr::rename(Churn = Exited)

# Move Churn column to the first position
bank_churn_lasso <- bank_churn_lasso %>%
  dplyr::select(Churn, everything())
```

```r
str(bank_churn_lasso)


x <- model.matrix(Churn ~ ., bank_churn_lasso)[, -1]
y <- bank_churn_lasso$Churn

grid <- 10^seq(10, -2, length = 100)
lasso.mod <- glmnet(x, y, alpha = 1, lambda = grid)

coef(lasso.mod)
dim(coef(lasso.mod))

names(bank_churn_lasso)

set.seed(5)
train <- sample(1:nrow(x), nrow(x)*2/3)
test <- (-train)
y.test <- y[test]


set.seed(5)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
par(mfrow=c(1,1))
plot(cv.out)

bestlam <- cv.out$lambda.min
bestlam # optimale
cv.out$lambda
cv.out$lambda.1se

log(bestlam)

lasso.pred <- predict(lasso.mod, s = bestlam, newx = x[test, ])
mse_lasso <- mean((lasso.pred - y.test)^2)

mse_lasso

out <- glmnet(x, y, alpha = 1)
lasso.coef <- predict(out, type = "coefficients", s = bestlam)[1:8, ]

#print koeficienterne
lasso.coef
```

Based on the lassomodel, the following variables are relevant

```r
lasso.coef[lasso.coef != 0]
```

```
     (Intercept)      CreditScore GeographyGermany   GeographySpain 
   -2.066091e-01    -8.596351e-05     1.219525e-01     2.850319e-03 
      GenderMale              Age           Tenure          Balance 
   -6.606308e-02     1.234790e-02    -3.601200e-03     4.477120e-07 
```

2.4.4 Logistic regression

Now we will perform logistic regression.

Logistic regression is a statistical method used to analyze the relationship between a binary dependent variable and one or more independent variables. Its purpose is to predict the probability of the binary variable taking a particular value based on the values of the independent variables. It differs from linear regression by applying a logistic function to estimate the probability. This method is particularly useful in fields such as medical research, economics, and marketing, where predicting probabilities of events like disease occurrence, customer purchases, or market segmentation is desired.

```
#training and test data

train <- bank_churn_fact[intrain,]
test <- bank_churn_fact[-intrain,]
view(bank_churn_fact)
names(test)

outcome <- "Churn"

#Vi så i lasso regression, hvilke variabler der ikke havde relevans, disse
eksluderes

variables <- c( ".", "ClusterHC", "clusterKmeans", "HasCrCard", "NumOfProducts",
"IsActiveMember")



f <- as.formula(paste(outcome,
                      paste(variables, collapse = " - "), sep = " ~ "))

str(bank_churn_fact)
str(bank_churn1)
# Vi fitter en logistisk regressionsmodel:

fit_logit <- glm(f, data=train, family = "binomial")

# Forudsige sandsynlighederne på træningsdataene
predictions <- predict(fit_logit, type = "response")

# Opret ROC-kurven og beregn AUC
```

```r
roc_curve <- roc(train$Churn, predictions)

# Vis AUC-værdien
auc_roc_log <- auc(roc_curve)
print(auc_roc_log)

# Visualiser ROC-kurven
roc_data <- data.frame(
  specificity = rev(roc_curve$specificities),
  sensitivity = rev(roc_curve$sensitivities)
)

ggplot(roc_data, aes(x = specificity, y = sensitivity)) +
  geom_line(color = "blue") +
  geom_abline(linetype = "dashed", color = "red") +
  labs(
    title = paste("ROC Curve (AUC =", round(auc_roc_log, 2), ")"),
    x = "1 - Specificity",
    y = "Sensitivity"
  ) +
  theme_minimal()

# Foretage prædiktioner på testsættet og gemmer dem i et objekt, churn_probs:

churn_probs <- predict(fit_logit, test, type = "response")

head(churn_probs)

# Kan vi gøre det bedre end den simple model (og modellen med 50/50 split)
# Loop:

thresh <- seq(0.01, 1.0, length = 100)
omk <- rep(0, length(thresh))

for (i in 1:length(thresh)) {
  glm.pred <- rep("No", length(churn_probs))
  glm.pred[churn_probs>thresh[i]] <- "Yes"
  glm.pred <- as.factor(glm.pred)
  x <- confusionMatrix(glm.pred, test$Churn, positive = "Yes")
  total <- x$table[1] + x$table[2] + x$table[3] + x$table[4]
  TN <- x$table[1]/total
  FP <- x$table[2]/total
  FN <- x$table[3]/total
  TP <- x$table[4]/total
  omk[i] <- FN*FN_omk + TP*TP_omk + FP*FP_omk + TN*0
}
```

We repeat the above procedure, but only for a model where threshold = 0.5. We call this model simple_model, and we compare it with different thresholds.

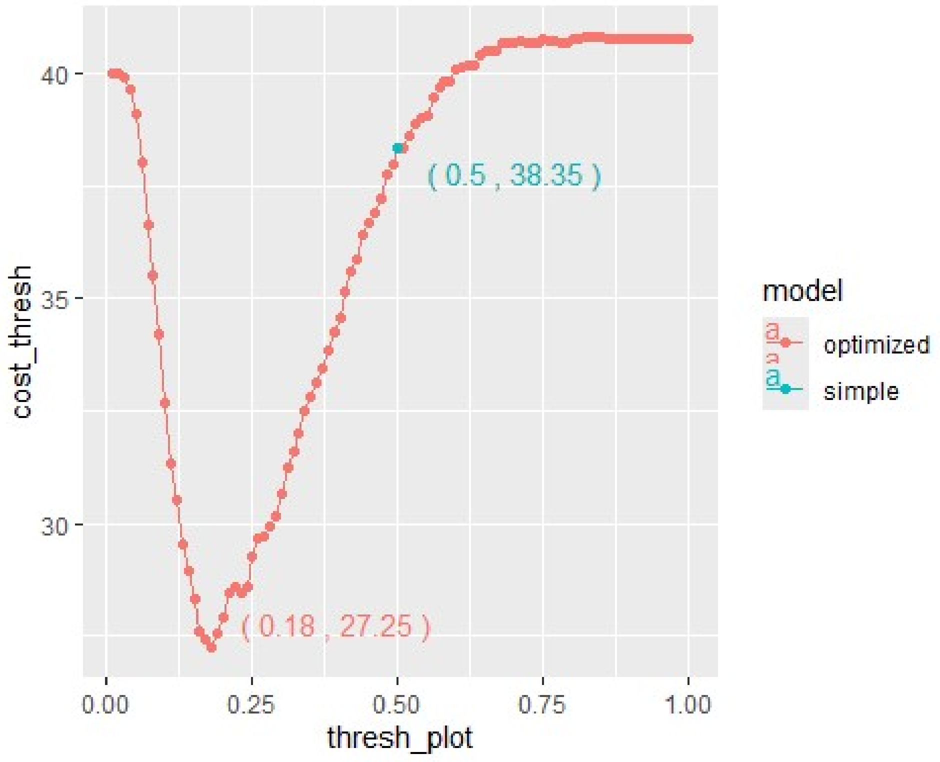
```r
glm.pred <- rep("No", length(churn_probs))
glm.pred[churn_probs>0.5] <- "Yes"
glm.pred <- as.factor(glm.pred)
x <- confusionMatrix(glm.pred, test$Churn, positive = "Yes")
total <- x$table[1] + x$table[2] + x$table[3] + x$table[4]
TN <- x$table[1]/total
FP <- x$table[2]/total
FN <- x$table[3]/total
TP <- x$table[4]/total
omk_simple <- FN*FN_omk + TP*TP_omk + FP*FP_omk + TN*0

# adding a column with the propability of the customer churning based on the
optimal threshold.

bank_churn_bi <- bank_churn_fact

bank_churn_bi$Log_Churn_Prob <- predict(fit_logit, newdata = bank_churn_fact, type
= "response")
```

We visualize the results to be able to see in a single way how much the costs depend on the thresholds.

```r
model <- c(rep("optimized", 100), "simple")
cost_thresh <- c(omk, omk_simple)
thresh_plot <- c(thresh, 0.5)
```

The visualization shows that the optimum threshold is 0,18.

```r
dataII <- data.frame(
  model,
  cost_thresh,
  thresh_plot
)

optimized_cost <- min(dataII$cost_thresh)
threshold_0_5_cost <- subset(dataII, thresh_plot == 0.5)$cost_thresh

ggplot(dataII, aes(x = thresh_plot, y = cost_thresh, group = model, colour =
model)) +
  geom_line() +
  geom_point() +
  geom_text(data = subset(dataII, cost_thresh == optimized_cost),
            aes(label = paste("(", round(thresh_plot, 2), ",", round(cost_thresh,
2), ")"),
                x = thresh_plot + 0.05, y = cost_thresh),
            vjust = -0.5, hjust = 0) +
  geom_text(data = subset(dataII, thresh_plot == 0.5),
            aes(label = paste("(", round(thresh_plot, 2), ",", round(cost_thresh,
2), ")"),
                x = thresh_plot + 0.05, y = cost_thresh),
            vjust = 1.5, hjust = 0)
```



```
# Vi finder først rækken, der svarer til threshold 0.5
threshold_0_5_row <- subset(dataII, thresh_plot == 0.5)

# Vi tager den første værdi af omkostningerne ved dette threshold
threshold_0_5_cost <- threshold_0_5_row$cost_thresh[1]

# Find det index, hvor omkostningen er minimal
optimal_index <- which.min(dataII$cost_thresh)



# Gem det optimale threshold og omkostningerne ved dette threshold
optimal_threshold_log <- dataII$thresh_plot[optimal_index]
optimal_cost_log <- dataII$cost_thresh[optimal_index]



# Og endelig kan vi udskrive værdierne for thrshold 0,5
print(paste("Threshold 0.5:", 0.5))
print(paste("Omkostninger ved threshold 0.5:", threshold_0_5_cost))

# Print dem ud
print(paste("Optimalt threshold:", optimal_threshold_log))
print(paste("Omkostninger ved optimalt threshold:", optimal_cost_log))


#adding a column with a binary outcome if the customer is churning based on the
logistic regression optimal threshold
bank_churn_bi <- bank_churn_bi %>%
  mutate(Log_Churn_Prediction = ifelse(Log_Churn_Prob > optimal_threshold_log,
"Yes", "No"))
```

Note that we can find an optimum with a different threshold than 0.50.

Calculates the saved costs of the optimized model (threshold=0.18) compared to the baseline model (threshold=0.5)."

```
besparelse_pr_kunde <- omk_simple - min(omk)

besparelse_pr_kunde*10000

[1] 110970.3
```

We can clearly outperform the common 50/50 split.

Can we do even better with an alternative model: Let's use linear and quadratic discriminant analysis and qudratic discriminant analysis, and we calculate again the total cost savings, and compare them with the best logistic regression.

2.4.5 Linear Discriminant Analysis (LDA)

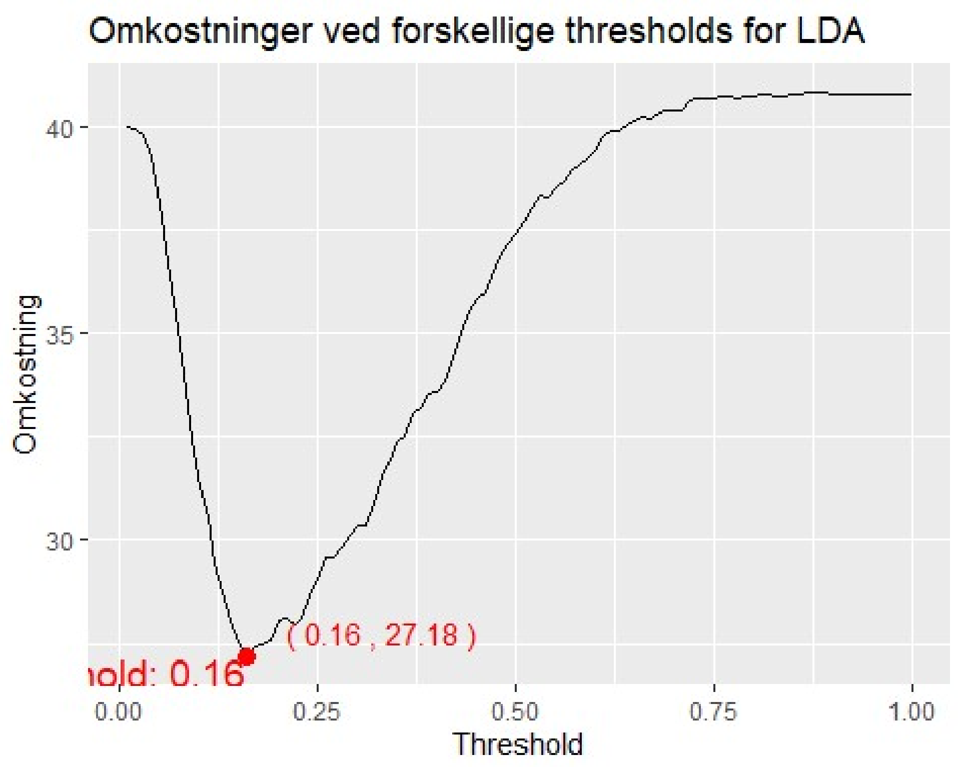
LDA is a method used for classification and dimensionality reduction. It finds the best linear combination of features to separate different classes in the dataset. LDA works well when classes are distinct and follows normal distributions with equal covariance. It's useful for understanding which features are most important for classification. However, it may not perform well with overlapping classes or outliers. Overall, LDA is effective for classification tasks with well-separated classes and can provide valuable insights into the data structure.

```
# Indlæs pakkerne
# Indlæs pakkerne
library(MASS)
library(pROC)
library(ggplot2)


train <- bank_churn_fact[intrain,]
test <- bank_churn_fact[-intrain,]
```

```r
lda.fit <- lda(f, data=train)
lda.pred <- data.frame(predict(lda.fit, test))

# Check the structure of the predicted object
str(lda.pred)


# Opret ROC-kurven og beregn AUC
roc_curve_lda <- roc(test$Churn, lda.pred$posterior.Yes)

# Vis AUC-værdien
auc_roc_lda <- auc(roc_curve_lda)
print(auc_roc_lda)

# Visualiser ROC-kurven
roc_data_lda <- data.frame(
  specificity = rev(roc_curve_lda$specificities),
  sensitivity = rev(roc_curve_lda$sensitivities)
)

ggplot(roc_data_lda, aes(x = specificity, y = sensitivity)) +
  geom_line(color = "blue") +
  geom_abline(linetype = "dashed", color = "red") +
  labs(
    title = paste("ROC Curve for LDA Model (AUC =", round(auc_roc_lda, 2), ")"),
    x = "1 - Specificity",
    y = "Sensitivity"
  ) +
  theme_minimal()

omk_lda <- rep(0,length(thresh))
thresh <- seq(0.01, 1.0, length = 100)

results_lda <- data.frame(threshold = numeric(), cost = numeric())

# Kør for løkken for at beregne omkostningerne ved forskellige thresholds
for (i in seq_along(thresh)) {
  # Skaber forudsigelser baseret på threshold
  glm.pred <- ifelse(lda.pred$posterior.Yes > thresh[i], "Yes", "No")
  glm.pred <- factor(glm.pred, levels = c("No", "Yes"))

  # Beregn confusion matrix
  cm <- confusionMatrix(glm.pred, test$Churn, positive = "Yes")
  total <- sum(cm$table)
  TN <- cm$table[1] / total
  FP <- cm$table[2] / total
  FN <- cm$table[3] / total
  TP <- cm$table[4] / total

  # Beregn omkostninger
  cost <- FN * FN_omk + TP * TP_omk + FP * FP_omk + TN * TN_omk

  # Tilføj til results_lda
  results_lda <- rbind(results_lda, data.frame(threshold = thresh[i], cost =
cost))
}
```

The visualization shows that the optimum threshold is 0,16 with a cost of 27,18

```
# Find det threshold med den laveste omkostning
optimal_threshold_lda <- results_lda$threshold[which.min(results_lda$cost)]
optimal_cost_lda <- min(results_lda$cost)


ggplot(results_lda, aes(x = threshold, y = cost)) +
  geom_line() +
  geom_point(data = subset(results_lda, threshold == optimal_threshold_lda),
             aes(x = threshold, y = cost), color = "red", size = 3) +
  geom_text(data = subset(results_lda, threshold == optimal_threshold_lda),
            aes(label = paste("(", round(threshold, 2), ",", round(cost, 2), ")"),
                x = threshold + 0.05, y = cost),
            vjust = -0.5, hjust = 0, color = "red") +
  labs(title = "Omkostninger ved forskellige thresholds for LDA",
       x = "Threshold",
       y = "Omkostning") +
  annotate("text", x = optimal_threshold_lda, y = min(results_lda$cost),
           label = paste("Threshold:", round(optimal_threshold_lda, 2)),
           hjust = 1, vjust = 1, size = 5, color = "red")
```



```r
# Print den optimale threshold og omkostning
#print(optimal_threshold_lda)
#print(optimal_cost_lda)

lda.fit <- lda(f, data=train)
lda.pred <- predict(lda.fit, test)

# Antag, at du allerede har beregnet dit optimale threshold og gemt det i
variablen optimal_threshold

# Trin 2: Generer optimal_predictions
optimal_predictions <- ifelse(lda.pred$posterior[, "Yes"] > optimal_threshold_lda,
"Yes", "No")

# Trin 3: Tjek længden af optimal_predictions
print(length(optimal_predictions)) # Dette skal matche antallet af observationer i
testdatasættet
print(length(test$Churn))


# Beregn forudsigelser baseret på det optimale threshold
optimal_predictions <- ifelse(lda.pred$posterior[, "Yes"] > optimal_threshold_lda,
"Yes", "No")
optimal_predictions <- factor(optimal_predictions, levels = c("No", "Yes"))
```

```
bank_churn_bi$LDA_Churn_Prob <- predict(lda.fit, newdata =
bank_churn_fact)$posterior[, "Yes"]

# Tilføj en ny kolonne baseret på det optimale QDA threshold
bank_churn_bi <- bank_churn_bi %>%
  mutate(LDA_Churn_Prediction = ifelse(LDA_Churn_Prob > optimal_threshold_lda,
"Yes", "No"))




# Beregn confusion matrix baseret på disse forudsigelser
cm <- confusionMatrix(optimal_predictions, test$Churn, positive = "Yes")
print(cm)
# Udskriv antallet af TP, FP, FN, og TN
#cat("True Positives (TP):", cm$table["Yes","Yes"], "\n")
#cat("False Positives (FP):", cm$table["No","Yes"], "\n")
#cat("False Negatives (FN):", cm$table["Yes","No"], "\n")
#cat("True Negatives (TN):", cm$table["No","No"], "\n")
```

2.4.6 Quadratic Discriminant Analysis (QDA)

QDA is similar to LDA but allows for different covariance matrices for each class, making it more flexible in capturing complex relationships between features. It works by estimating separate covariance matrices for each class, which can better capture non-linear decision boundaries. QDA is beneficial when classes have different variances or when the decision boundary is non-linear. However, it requires more parameters to estimate compared to LDA and may overfit with small datasets. Overall, QDA is useful for classification tasks with non-linear decision boundaries and varying variances between classes.

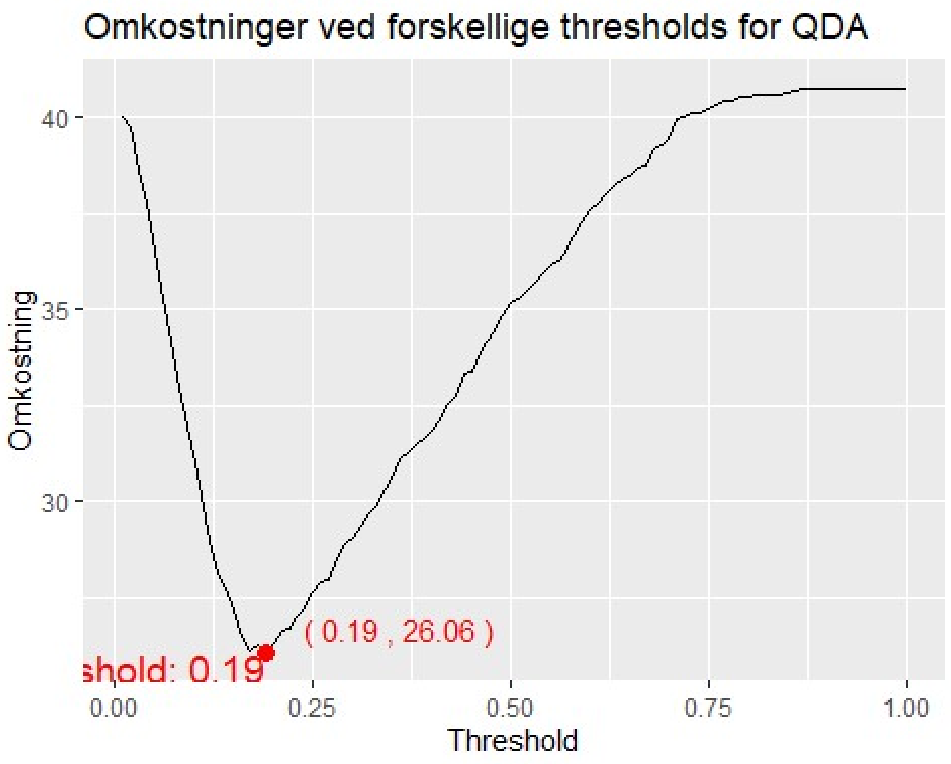
```
#qda


# Load necessary library
library(caret)



view(bank_churn_fact)

bank_churn_fact <- as.data.frame(bank_churn_fact)
class(bank_churn_fact)
class(bank_churn_fact$IsActiveMember.0)
class(bank_churn_fact$IsActiveMember.1)

class(bank_churn_fact)
```

```r
# Split data into training and test sets
set.seed(123)  # For reproducibility
intrain <- createDataPartition(bank_churn_fact$Churn, p = 0.7, list = FALSE)
train <- bank_churn_fact[intrain,]
test <- bank_churn_fact[-intrain,]

# Ensure Churn is a factor with the correct levels
train$Churn <- factor(train$Churn, levels = c("No", "Yes"))
test$Churn <- factor(test$Churn, levels = c("No", "Yes"))



# Fit QDA model
qda.fit <- qda(f, data=train)
qda.pred <- predict(qda.fit, test)

# Check the structure of the predicted posterior probabilities
str(qda.pred)

# Opret ROC-kurven og beregn AUC
roc_curve_qda <- roc(test$Churn, qda.pred$posterior[, "Yes"])

# Vis AUC-værdien
auc_roc_qda <- auc(roc_curve_qda)
print(paste("AUC for QDA model:", auc_roc_qda))

# Visualiser ROC-kurven
roc_data_qda <- data.frame(
  specificity = rev(roc_curve_qda$specificities),
  sensitivity = rev(roc_curve_qda$sensitivities)
)

ggplot(roc_data_qda, aes(x = specificity, y = sensitivity)) +
  geom_line(color = "blue") +
  geom_abline(linetype = "dashed", color = "red") +
  labs(
    title = paste("ROC Curve for QDA Model (AUC =", round(auc_roc_qda, 2), ")"),
    x = "1 - Specificity",
    y = "Sensitivity"
  ) +
  theme_minimal()

sum(is.na(qda.pred$posterior[, "Yes"]))
```

```r
omk_qda <- rep(0,length(thresh))
thresh <- seq(0.01, 1.0, length = 100)

results_qda <- data.frame(threshold = numeric(), cost = numeric())

# Loop to calculate costs for different thresholds
for (i in seq_along(thresh)) {
  # Create predictions based on the threshold
  glm.pred <- ifelse(qda.pred$posterior[, "Yes"] > thresh[i], "Yes", "No")
  glm.pred <- factor(glm.pred, levels = c("No", "Yes"))

  # Check lengths of predictions and test data
  print(length(glm.pred)) # This should match the number of observations in the
test dataset
  print(length(test$Churn))

  # Beregn confusion matrix
  cm <- confusionMatrix(glm.pred, test$Churn, positive = "Yes")
  total <- sum(cm$table)
  TN <- cm$table[1] / total
  FP <- cm$table[2] / total
  FN <- cm$table[3] / total
  TP <- cm$table[4] / total

  # Beregn omkostninger
  cost <- FN * FN_omk + TP * TP_omk + FP * FP_omk + TN * TN_omk

  # Tilføj til results_qda
  results_qda <- rbind(results_qda, data.frame(threshold = thresh[i], cost =
cost))
}



# Find det threshold med den laveste omkostning
optimal_threshold_qda <- results_qda$threshold[which.min(results_qda$cost)]
optimal_cost_qda <- min(results_qda$cost)
```

The visualization shows that the optimum threshold is 0,19 with a cost of 26,06

```r
ggplot(results_qda, aes(x = threshold, y = cost)) +
  geom_line() +
  geom_point(data = subset(results_qda, threshold == optimal_threshold_qda),
             aes(x = threshold, y = cost), color = "red", size = 3) +
  geom_text(data = subset(results_qda, threshold == optimal_threshold_qda),
            aes(label = paste("(", round(threshold, 2), ",", round(cost, 2), ")"),
                x = threshold + 0.05, y = cost),
            vjust = -0.5, hjust = 0, color = "red") +
  labs(title = "Omkostninger ved forskellige thresholds for QDA",
       x = "Threshold",
       y = "Omkostning") +
  annotate("text", x = optimal_threshold_qda, y = min(results_qda$cost),
           label = paste("Threshold:", round(optimal_threshold_qda, 2)),
           hjust = 1, vjust = 1, size = 5, color = "red")
```



```r
# Print den optimale threshold og omkostning
#print(optimal_threshold_qda)
#print(optimal_cost_qda)

qda.fit <- qda(f, data=train)
qda.pred <- predict(qda.fit, test)

# Antag, at du allerede har beregnet dit optimale threshold og gemt det i
variablen optimal_threshold

#Generer optimal_predictions
optimal_predictions <- ifelse(qda.pred$posterior[, "Yes"] > optimal_threshold_qda,
"Yes", "No")
```

```
# Beregn forudsigelser baseret på det optimale threshold
optimal_predictions <- ifelse(qda.pred$posterior[, "Yes"] > optimal_threshold_qda,
"Yes", "No")
optimal_predictions <- factor(optimal_predictions, levels = c("No", "Yes"))

# Beregn confusion matrix baseret på disse forudsigelser
cm <- confusionMatrix(optimal_predictions, test$Churn, positive = "Yes")
# Udskriv antallet af TP, FP, FN, og TN

#cat("True Positives (TP):", cm$table["Yes","Yes"], "\n")
#cat("False Positives (FP):", cm$table["No","Yes"], "\n")
#cat("False Negatives (FN):", cm$table["Yes","No"], "\n")
#cat("True Negatives (TN):", cm$table["No","No"], "\n")



bank_churn_bi$QDA_Churn_Prob <- predict(qda.fit, newdata =
bank_churn_fact)$posterior[, "Yes"]

# Tilføj en ny kolonne baseret på det optimale QDA threshold
bank_churn_bi<- bank_churn_bi %>%
  mutate(QDA_Churn_Prediction = ifelse(QDA_Churn_Prob > optimal_threshold_qda,
"Yes", "No"))
```

2.4.7 Gradient Boosting Machine (GBM)

Gradient Boosting Machine (GBM) is a versatile machine learning technique that improves prediction accuracy by sequentially correcting mistakes of prior models, often using decision trees. It works well for both regression and classification tasks, handling diverse data types. While GBM can offer high precision, it requires careful parameter tuning to avoid overfitting and can be computationally intensive. Despite these considerations, its effectiveness in various applications makes it a favored choice among data scientists.

First we wil convert the variables to the correct classes. We are defining af tuning grid, in order to find the best model that is performing best.

```
#GBM med tuning grid



bank_churn_gbm <- bank_churn1 %>%
  dplyr::select(-RowNumber, -CustomerId, -Surname) %>%
  rename(Churn = Exited) %>%
  mutate(Churn = factor(Churn, levels = c(0, 1)))

library(gbm)

# Sikrer reproducerbarhed
set.seed(123)
```

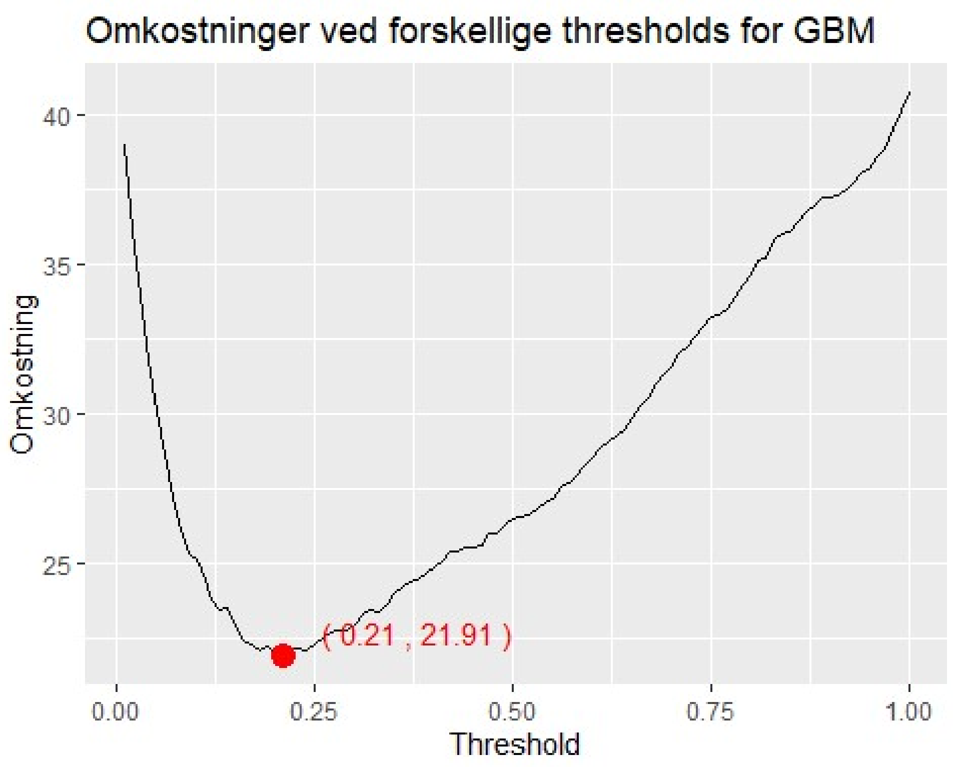
```
# Oprette trænings- og testdatasæt
trainIndex <- createDataPartition(bank_churn_gbm$Churn, p = .7,
                                  list = FALSE,
                                  times = 1)

trainData <- bank_churn_gbm[trainIndex, ]
testData <- bank_churn_gbm[-trainIndex, ]

# Omdøb faktorniveauerne til "Class1" og "Class0"
trainData$Churn <- factor(trainData$Churn, levels = c(0, 1), labels = c("Class0",
"Class1"))

# Opdater også testData, hvis du har det
testData$Churn <- factor(testData$Churn, levels = c(0, 1), labels = c("Class0",
"Class1"))




# Definer et grid af parametre at prøve (kun n.trees og shrinkage)
tuneGrid <- expand.grid(.n.trees = c(100, 500, 1000),
                        .shrinkage = c(0.01, 0.05, 0.1),
                        .interaction.depth = 1,
                        .n.minobsinnode = 10) # Tilføjer n.minobsinnode med en
værdi af 10


control <- trainControl(method = "cv", number = 5, classProbs = TRUE,
summaryFunction = twoClassSummary)

gbmFit <- train(Churn ~ ., data = trainData, method = "gbm",
                trControl = control, verbose = FALSE,
                tuneGrid = tuneGrid,
                metric = "ROC",
                distribution = "bernoulli")

# Se de bedste parametre
print(gbmFit$bestTune)

  n.trees interaction.depth shrinkage n.minobsinnode
6    1000                 1      0.05             10

# Brug model til at lave forudsigelser
predictions <- predict(gbmFit, newdata = testData, type = "prob")

# Beregn AUC for at evaluere modellens præstation
```

```r
library(pROC)
# Opret ROC-kurven og beregn AUC
roc_curve_gbm <- roc(testData$Churn, predictions[,2])

# Vis AUC-værdien
auc_roc_gbm <- auc(roc_curve_gbm)

# Forbered data til plotting
roc_data_gbm <- data.frame(
  specificity = rev(roc_curve_gbm$specificities),
  sensitivity = rev(roc_curve_gbm$sensitivities)
)


# Beregn forudsagte sandsynligheder for testdatasættet
predicted_probs_gbm <- predict(gbmFit, newdata = testData, type =
"prob")[,"Class1"]

# Beregn forudsagte klasser for testdatasættet ved et threshold på 0,5
predicted_class_gbm <- ifelse(predicted_probs_gbm > 0.5, "Class1", "Class0")

# Sørg for at både Predicted og Actual er faktorer med de samme niveauer
predicted_factor_gbm <- factor(predicted_class_gbm, levels = c("Class0",
"Class1"))
actual_factor_gbm <- factor(testData$Churn, levels = c("Class0", "Class1")) #
Juster variabelnavnet efter dit datasæt

# Beregn og udskriv confusion matrix
cm <- confusionMatrix(predicted_factor_gbm, actual_factor_gbm)

# Definer thresholds
thresh <- seq(0.01, 1, by = 0.01)

# Initialiser en dataframe til at holde omkostningerne ved hvert threshold
omkostninger_gbm <- data.frame(threshold = numeric(), cost = numeric())

# Loop over hvert threshold for at beregne omkostningerne
for(t in thresh) {
  # Generer klassificering baseret på det aktuelle threshold
  predicted_class_gbm <- ifelse(predicted_probs_gbm > t, "Class1", "Class0")

  # Sørg for at både Predicted og Actual er faktorer med de samme niveauer
  predicted_factor_gbm <- factor(predicted_class_gbm, levels = c("Class0",
"Class1"))
  actual_factor_gbm <- factor(testData$Churn, levels = c("Class0", "Class1"))

  # Beregn confusion matrix
  cm_gbm <- confusionMatrix(predicted_factor_gbm, actual_factor_gbm)

  # Ekstraher værdier fra confusion matrix
  TN_gbm <- cm_gbm$table["Class0","Class0"]
  FP_gbm <- cm_gbm$table["Class1","Class0"]
  FN_gbm <- cm_gbm$table["Class0","Class1"]
  TP_gbm <- cm_gbm$table["Class1","Class1"]

  # Beregn omkostningerne
  cost_gbm <- (FN_gbm * FN_omk + TP_gbm * TP_omk + FP_gbm * FP_omk + TN_gbm *
TN_omk) / sum(cm_gbm$table)

  # Tilføj threshold og omkostninger til dataframe
  omkostninger_gbm <- rbind(omkostninger_gbm, data.frame(threshold = t, cost =
cost_gbm))
}

# Find det optimale threshold og de tilhørende omkostninger
optimal <- omkostninger_gbm[which.min(omkostninger_gbm$cost), ]
print(optimal)

   threshold     cost
21      0.21 21.91397
```

The best model GBM model has 1000 tress and a 0.05 shrinkage.

The visualization shows that the optimal thres old is 0,23 with a cost of 22,27

```
optimal_threshold_gbm <- optimal$threshold
optimal_cost_gbm <- optimal$cost
#print(paste("Optimalt threshold: ", optimal_threshold_gbm))
#print(paste("Omkostninger ved optimalt threshold: ", optimal_cost_gbm))



ggplot(omkostninger_gbm, aes(x = threshold, y = cost)) +
  geom_line() +
  geom_point(data = optimal, aes(x = threshold, y = cost), color = "red", size =
4) +
  geom_text(data = optimal,
            aes(label = paste("(", round(threshold, 2), ",", round(cost, 2), ")"),
                x = threshold + 0.05, y = cost),
            vjust = -0.5, hjust = 0, color = "red") +
  labs(title = "Omkostninger ved forskellige thresholds for GBM",
       x = "Threshold",
       y = "Omkostning")
```



```
# Beregn sandsynligheder for hele datasættet med din GBM-model
predicted_probs_whole_gbm <- predict(gbmFit, newdata=bank_churn_gbm,
type="prob")[,2]

# Tilføjer sandsynlighederne som en kolonne i bi datasættet
bank_churn_bi$GBM_Churn_Prob <- predicted_probs_whole_gbm

# Bestem det optimale threshold fra dine tidligere resultater
optimal_threshold_gbm <- optimal$threshold  # Sørg for, at dette er opdateret
baseret på GBM-resultaterne

# Generer forudsigelser baseret på det optimale threshold for hele datasættet
optimal_predictions_gbm <- ifelse(predicted_probs_whole_gbm >
optimal_threshold_gbm, "Yes", "No")

# Tilføjer en kolonne, der viser om en kunde churner på baggrund af det optimale
GBM threshold
bank_churn_bi <- bank_churn_bi %>%
  mutate(GBM_Churn_Prediction = optimal_predictions_gbm)
```

2.4.8 Random Forrest

Random Forest is a powerful ensemble learning method that builds multiple decision trees and merges them for more accurate and stable predictions. It’s effective for both classification and regression tasks, easily handling categorical and numerical data. Random Forest can manage large datasets with high dimensionality but remains relatively efficient and interpretable. Its ability to estimate feature importance is invaluable for understanding the impact of variables on predictions.

```r
# We are here using random forrest.

bank_churn_RF <- bank_churn1

bank_churn_RF <- bank_churn1 %>%
  dplyr::select(Exited, everything()) %>%
  dplyr::rename(Churn = Exited) %>%
  mutate(Churn = ifelse(Churn == 1, "Yes", "No"),
         Churn = as.factor(Churn)) %>%
  dplyr::select(-RowNumber, -CustomerId, -Surname)

bank_churn_RF <- bank_churn_RF %>%
  mutate_if(is.character, as.factor)
sum(is.na(bank_churn_RF
          ))


set.seed(123) # Sikrer reproducerbarhed

# Opret train/test split
trainIndex <- createDataPartition(bank_churn_RF$Churn, p = .7, list = FALSE)
trainData <- bank_churn_RF[trainIndex,]
testData <- bank_churn_RF[-trainIndex,]

# Definer de værdier, du vil teste for mtry og n.trees
mtry_values <- c(2, 4, 6, 8, 10, 12)  # Eksempel: Test 2, 4, 6 og 8 for mtry
n_trees_values <- c(50, 100, 250, 500, 1000)  # Eksempel: Test 50, 100, 250, 500
og 1000 for n.trees

# Initialiser variabler til at holde den bedste model og dens nøjagtighed
best_accuracy <- 0
best_model <- NULL
best_mtry <- NULL
best_ntrees <- NULL

# Løkke til at teste forskellige kombinationer af mtry og n.trees
for (mtry in mtry_values) {
  for (n_trees in n_trees_values) {
    # Byg random forest-modellen med aktuelle mtry og n.trees værdier
    rfModel <- randomForest(Churn ~ ., data = trainData, mtry = mtry, ntree =
n_trees)
```

```
    # Forudsige testdatasættet
    predictions <- predict(rfModel, testData)

    # Evaluér modellens præstation
    confMat <- confusionMatrix(predictions, testData$Churn)

    # Beregn nøjagtighed (accuracy) og sammenlign med den bedste fundne
    accuracy <- confMat$overall['Accuracy']
    if (accuracy > best_accuracy) {
      best_accuracy <- accuracy
      best_model <- rfModel
      best_mtry <- mtry
      best_ntrees <- n_trees
    }
  }
}
```

We have tried a number of variables and number of trees. The model with the best accuracy has 4 variables and 250 trees.

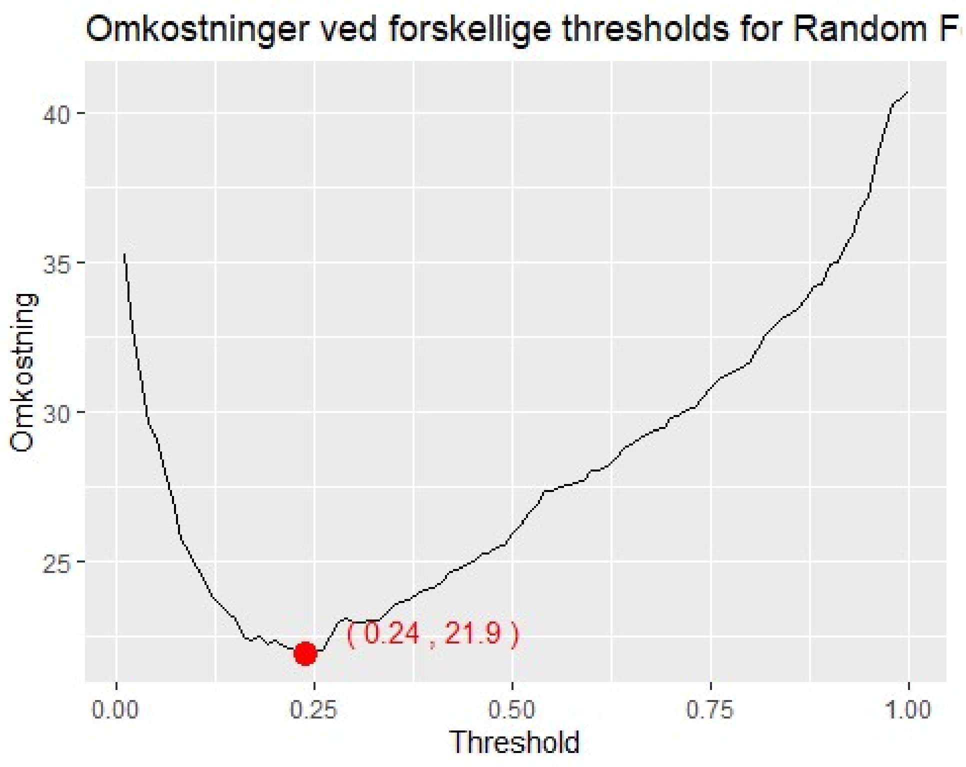
```
# Udskriv den bedste model og dens nøjagtighed
#print("Bedste model:")
print(best_model)


Call:
 randomForest(formula = Churn ~ ., data = trainData, mtry = mtry,      ntree = 
n_trees)
               Type of random forest: classification
                     Number of trees: 250
No. of variables tried at each split: 6

        OOB estimate of  error rate: 13.96%
Confusion matrix:
      No Yes class.error
No  5341 234  0.04197309
Yes  743 683  0.52103787

#print(paste("Nøjagtighed (Accuracy):", best_accuracy))
#print(paste("Bedste mtry værdi:", best_mtry))
#print(paste("Bedste n.trees værdi:", best_ntrees))
# Beregn AUC for den bedste model
# Beregn ROC-kurven og AUC-værdien
predicted_probs <- predict(best_model, testData, type = "prob")[, "Yes"]
roc_curve_rf <- roc(testData$Churn, predicted_probs)
auc_roc_rf <- auc(roc_curve_rf)

# Forbered data til plotting
```

```r
roc_data_rf <- data.frame(
  specificity = rev(roc_curve_rf$specificities),
  sensitivity = rev(roc_curve_rf$sensitivities)
)

# Forudsige sandsynligheder for klassen "Yes"

predicted_probs <- predict(best_model, newdata = testData, type = "prob")[, "Yes"]

# Definer thresholds
thresh <- seq(0.01, 1, by = 0.01)

# Initialiser en dataframe til at holde omkostningerne ved hvert threshold
omkostninger <- data.frame(threshold = numeric(), cost = numeric())

# Loop over hvert threshold
for(t in thresh){
  # Generer forudsigelser baseret på det aktuelle threshold
  predicted_class <- ifelse(predicted_probs > t, "Yes", "No")

  # Sørg for at både Predicted og Actual er faktorer med de samme niveauer
  predicted_factor <- factor(predicted_class, levels = c("No", "Yes"))
  actual_factor <- factor(testData$Churn, levels = c("No", "Yes"))

  # Beregn confusion matrix
  cm <- table(Predicted = predicted_factor, Actual = actual_factor)

  # Beregn omkostninger. Brug safe indexing for at undgå "subscript out of bounds"
fejl.
  TN <- ifelse(!is.na(cm["No","No"]), cm["No","No"], 0)
  FP <- ifelse(!is.na(cm["Yes","No"]), cm["Yes","No"], 0)
  FN <- ifelse(!is.na(cm["No","Yes"]), cm["No","Yes"], 0)
  TP <- ifelse(!is.na(cm["Yes","Yes"]), cm["Yes","Yes"], 0)
  total_omk <- (FN * FN_omk + TP * TP_omk + FP * FP_omk + TN * TN_omk) / sum(cm)

  # Tilføj threshold og omkostninger til dataframe
  omkostninger <- rbind(omkostninger, data.frame(threshold = t, cost = total_omk))
}


# Find det optimale threshold
optimal <- omkostninger[which.min(omkostninger$cost), ]
print(optimal)
```

The visualization shows that the optimum threshold is 0,26 with a cost of 21,7

```r
# Udskriv det optimale threshold og omkostninger separat
optimal_threshold_rf <- optimal$threshold
```

```
optimal_cost_rf <- optimal$cost
#print(paste("Optimalt threshold: ", optimal_threshold_rf))
#print(paste("Omkostninger ved optimalt threshold: ", optimal_cost_rf))

# Plot omkostninger mod threshold

ggplot(omkostninger, aes(x = threshold, y = cost)) +
  geom_line() +
  geom_point(data = optimal, aes(x = threshold, y = cost), color = "red", size =
4) +
  geom_text(data = optimal,
            aes(label = paste("(", round(threshold, 2), ",", round(cost, 2), ")"),
                x = threshold + 0.05, y = cost),
            vjust = -0.5, hjust = 0, color = "red") +
  labs(title = "Omkostninger ved forskellige thresholds for Random Forest",
       x = "Threshold",
       y = "Omkostning")
```



```
# Trin 1: Beregn sandsynligheder for hele datasættet bank_churn_RF
predicted_probs_whole <- predict(best_model, newdata=bank_churn_RF,
type="prob")[,"Yes"]

# Trin 2: Tilføj sandsynlighederne som en ny kolonne til bank_churn_bi datasættet
bank_churn_bi$RF_Churn_Prob <- predicted_probs_whole
```

```
# Trin 3: Konverter sandsynligheder til forudsigelser baseret på det optimale
threshold
# og tilføj disse forudsigelser som en anden ny kolonne
bank_churn_bi$RF_Churn_Prediction <- ifelse(predicted_probs_whole >
optimal_threshold_rf, "Yes", "No")
```

2.5 Evaluation

2.5.0.1 Compare Costs

In this section we vill compare the costs of the simple model and the advanced models. By comparing the comparing the costs we can decide the optimal solution in a business perspective.

```
omk_simple <- threshold_0_5_cost
omk_log <-optimal_cost_log
omk_lda <- optimal_cost_lda
omk_qda <- optimal_cost_qda
omk_boost <- optimal_cost_gbm
omk_rf <- optimal_cost_rf


omk_simple

[1] 38.34612

omk_log

[1] 27.24908

omk_lda

[1] 27.18239

omk_qda

[1] 26.06202

omk_boost

[1] 21.91397

omk_rf

[1] 21.90063

#tilføj omkostninger pr. kunde til powerbi datasætttet
#Beregn omkostninger for Random forrest
#bank_churn_bi$Simple_cost <- omk_simple
```

```
# Beregn omkostninger for Logistisk Regression
#bank_churn_bi$log_reg_cost <- omk_log
# Beregn omkostninger for LDA
#bank_churn_bi$lda_cost <- omk_lda

# Beregn omkostninger for QDA
#bank_churn_bi$qda_cost <- omk_qda
# Beregn omkostninger for Boost
#bank_churn_bi$boost_cost <- omk_boost
#Beregn omkostninger for Random forrest
#bank_churn_bi$rf_cost <- omk_rf



# Find den mindste omkostning blandt de avancerede modeller
min_advanced_omk <- min(c(omk_simple, omk_log, omk_lda, omk_qda, omk_boost,
omk_rf))

# Beregn besparelserne ved at skifte fra den simple model til den bedste
avancerede model
savings <- omk_simple - min_advanced_omk

# Opret en matrix eller data frame til at vise disse resultater
savings_matrix <- matrix(c(omk_simple, min_advanced_omk, savings), nrow = 1)
colnames(savings_matrix) <- c("Simple Model Cost", "Best Advanced Model Cost",
"Savings")
savings_matrix <- as.data.frame(savings_matrix)

print(optimal)

   threshold     cost
24      0.24 21.90063
```

we can here determine that the simple model has a cost of 38. 34 in average per customer. The lowest cost is given by The Random Forrest model at 21.90 at at threshold 0.24

After comparing the 6 models, we can compare the costs, and calculate the savings per customer and in total.

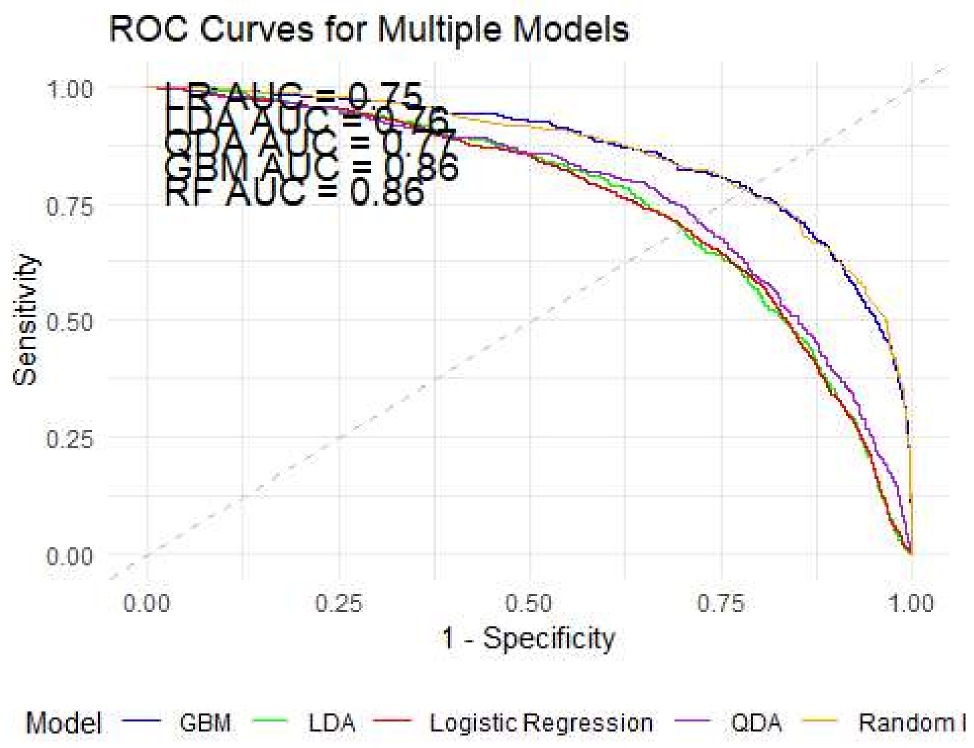
```
# Vis matrix
print(savings_matrix)

  Simple Model Cost Best Advanced Model Cost  Savings
1          38.34612                 21.90063 16.44548

savingstotal <- savings * 10000

print(savingstotal)
```

```
[1] 164454.8
```

2.5.0.2 ROC curves

We can now asses the performance of the models. As we can see we are able to save money by focusing on data driven decisions, and optimized tresholds.

```
#print(auc_roc_log)
#print(auc_roc_lda)
#print(auc_roc_qda)
#print(auc_roc_gbm)
#print(auc_roc_rf)
library(ggplot2)
library(dplyr)

# Samler alle dataframes til en
all_roc_data <- bind_rows(
  mutate(roc_data, model = "Logistic Regression", auc = round(auc_roc_log, 2)),
  mutate(roc_data_lda, model = "LDA", auc = round(auc_roc_lda, 2)),
  mutate(roc_data_qda, model = "QDA", auc = round(auc_roc_qda, 2)),
  mutate(roc_data_gbm, model = "GBM", auc = round(auc_roc_gbm, 2)),
  mutate(roc_data_rf, model = "Random Forest", auc = round(auc_roc_rf, 2))
)

# Antager du allerede har defineret all_roc_data og har plottet kurverne
plot <- ggplot(all_roc_data, aes(x = specificity, y = sensitivity, color = model))
+
  geom_line() +
  geom_abline(linetype = "dashed", color = "gray") +
  scale_color_manual(values = c("blue", "green", "red", "purple", "orange")) +
  labs(
    title = "ROC Curves for Multiple Models",
    x = "1 - Specificity",
    y = "Sensitivity",
    color = "Model"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")

# Tilføj AUC værdier direkte fra de gemte objekter
plot + annotate("text", x = 0.02, y = 0.95, label = paste("LR AUC =",
round(auc_roc_log, 2)), color = "black", hjust = 0, vjust = 0, size = 5) +
      annotate("text", x = 0.02, y = 0.90, label = paste("LDA AUC =",
round(auc_roc_lda, 2)), color = "black", hjust = 0, vjust = 0, size = 5) +
      annotate("text", x = 0.02, y = 0.85, label = paste("QDA AUC =",
round(auc_roc_qda, 2)), color = "black", hjust = 0, vjust = 0, size = 5) +
      annotate("text", x = 0.02, y = 0.80, label = paste("GBM AUC =",
round(auc_roc_gbm, 2)), color = "black", hjust = 0, vjust = 0, size = 5) +
      annotate("text", x = 0.02, y = 0.75, label = paste("RF AUC =",
round(auc_roc_rf, 2)), color = "black", hjust = 0, vjust = 0, size = 5)
```



It is here clear that the best performing model based on AUC is Random forrest and GBM with a AUC on 0,86.

Therefor it is recommended to chose Random Forest since the cost are lowest, and the model has slighty better precsion than the follow up GBM.

By implementing a threshold on 0,24, meaning that implementing a retention strategy for all custmer with a churn risk over 24%, the compan will save money.

2.6 Deployment

The code is scalable since it can be adapted to changes in costs. If we assume that retention cost is double, we only have to change one parameter.

```
TP_omk <- 80
```

And run the code again.

Furthermore once models are trained and evaluated, they're deployed for real-world use. Here's how:

1. **Packaging**: Bundle the trained model with preprocessing steps for easy deployment.

2. **Integration**: Integrate the model into existing systems, ensuring compatibility.
3. **Optimization**: Optimize for scalability and performance to handle real-time requests.
4. **Monitoring**: Monitor model performance over time and update regularly.
5. **Security**: Ensure compliance with security regulations and protect data privacy.
6. **Documentation**: Provide user-friendly documentation and support resources.
7. **Feedback**: Gather feedback for continuous improvement and iteration.

3. Conslusion

Based on the analysis, it is recommended that the bank focuses its retention efforts on customers identified as high-risk churners by the predictive models. This targeted approach can help optimize resource allocation and improve overall customer retention strategies.

In conclusion, the analysis highlights the importance of leveraging data-driven approaches to understand and address customer churn effectively. By implementing the recommended strategies, the bank can enhance customer satisfaction, reduce churn rates, and ultimately, drive long-term business growth and profitability.