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

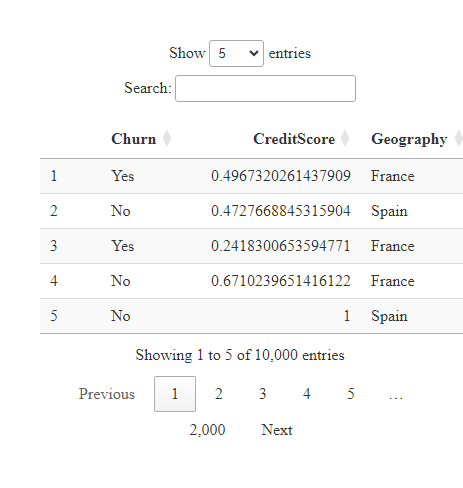
### 2.3.1 Cleaning data

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.

# 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.

# 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')



## 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](https://www.reviewtrackers.com/blog/customer-acquisition-banking/#:~:text=How%20much%20does%20it%20cost%20a%20bank%20to,%2866%25%29%2C%20and%20SEO%20%2F%20pay-per-click%20%28PPC%29%20advertising%20%2865%25%29.) - 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.

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  
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())  
  
  
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

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  
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.

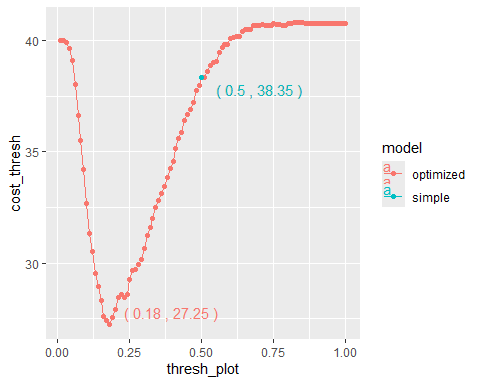
glm.pred <- rep("No", length(churn\_probs))  
glm.pred[churn\_probs>0.5] <- "Yes"  
glm.pred <- as.factor(glm.pred)  
x <- confusionMatrix(glm.pred, test$Churn, positive = "Yes")  
total <- x$table[1] + x$table[2] + x$table[3] + x$table[4]  
TN <- x$table[1]/total  
FP <- x$table[2]/total  
FN <- x$table[3]/total  
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.

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.

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)

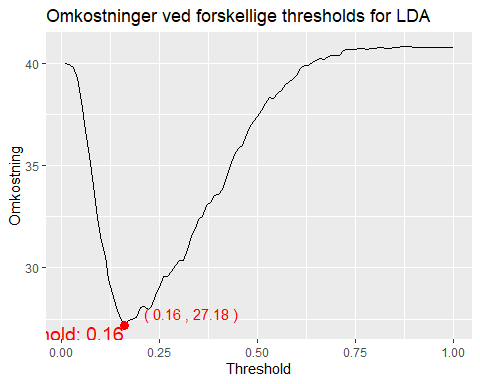
LDA is a method used for classification and dimensionality reduction. It finds the best linear combination of features to separate different classes in the dataset. LDA works well when classes are distinct and follows normal distributions with equal covariance. It’s useful for understanding which features 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,]  
  
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")



# 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)

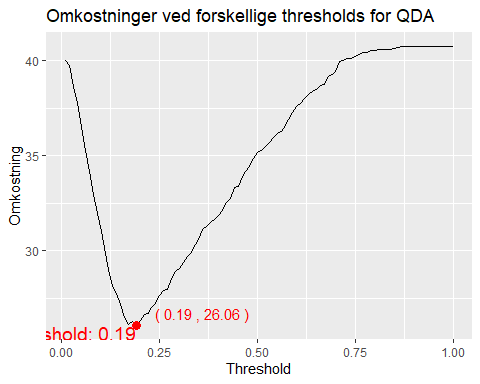
QDA is similar to LDA but allows for different covariance matrices for each class, making it more flexible in capturing complex relationships between features. It works by estimating separate covariance matrices for each class, which can better capture non-linear decision boundaries. QDA is beneficial 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)  
  
  
  
# 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"]))  
  
  
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

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")



# 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

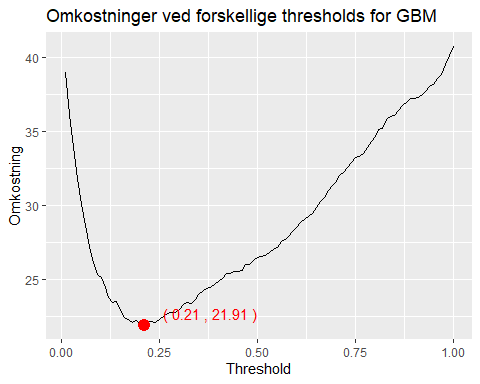
# Brug model til at lave forudsigelser  
predictions <- predict(gbmFit, newdata = testData, type = "prob")  
  
# Beregn AUC for at evaluere modellens præstation  
library(pROC)  
# Opret ROC-kurven og beregn AUC  
roc\_curve\_gbm <- roc(testData$Churn, predictions[,2])  
  
# Vis AUC-værdien  
auc\_roc\_gbm <- 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.

# 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)

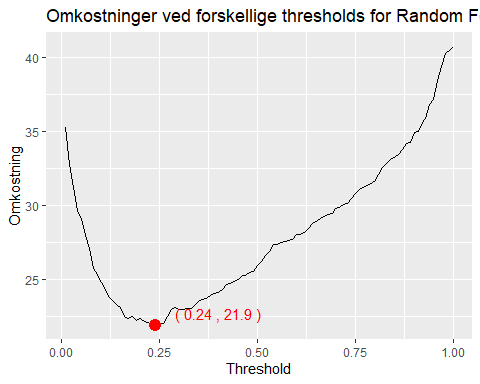
Call:  
 randomForest(formula = Churn ~ ., data = trainData, mtry = mtry, ntree = n\_trees)   
 Type of random forest: classification  
 Number of trees: 250  
No. of variables tried at each split: 6  
  
 OOB estimate of error rate: 13.96%  
Confusion matrix:  
 No Yes class.error  
No 5341 234 0.04197309  
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  
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

# 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

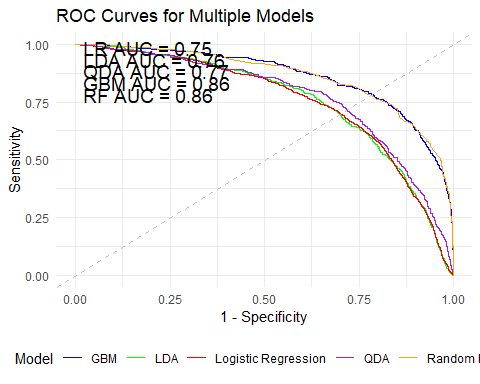
savingstotal <- savings \* 10000  
  
print(savingstotal)

[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.