# Credit Card Churn

Torres Reyes Ruben

2026-02-16

## 1 Introduction

According to IBM in its article "What is customer churn?" [**1**], customer churn is the number of existing customers lost in a given period of time. IBM underlines the importance of understanding and minimizing this metric, as it has an immediate impact on the bottom line, since replacing the value of one lost customer can require the acquisition of three new customers [**1**]. In addition, they mention other negative impacts such as the demoralization of executives and employees; the distraction of the company from focusing on existing customers; and the loss of new customers resulting from negative comments from lost customers [**1**].

Therefore, in this project, I will apply the knowledge acquired through the HarvardX data science program on edX by analyzing and creating a system to detect customers at risk of churn from a bank so that the bank can proactively contact those customers and offer them better services to change their decision. To do this, I will use the **BankChurners** database available on Kaggle [**2**], which contains information on 10,127 customers, including their demographics and the frequency and manner in which they use their banking products. This data will allow me to identify the patterns that lead a customer to the *Attrited Customer* category through the exploration and analysis of three different predictive models.

### 1.1 Loading, Transforming, and Describing Dataset

First, the packages to be used are downloaded and loaded.

```
# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos =
↪  "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(scales)) install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos =
↪  "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos =
↪  "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
library(tidyverse)
library(caret)
library(scales)
library(randomForest)
library(knitr)
library(kableExtra)
```

### 1.1.1 Dataset Loading and Initial Changes

In this initial stage, I load the **BankChurners** dataset into the variable **bankChurners**. I removed three specific columns: the first, corresponding to the unique customer identifier (***CLIENTNUM***), because it has no predictive-value, and the last two columns, which contain results from a Naive Bayes classifier that the author of the dataset recommends omitting to avoid bias in new models [**2**].

```
# Load BankChurners dataset:

options(timeout = 120)

bankChurners <- read.csv("BankChurners.csv")

# Drop unnecessary columns
bankChurners <- bankChurners[, 2:(ncol(bankChurners)-2)]

# Transform all character columns to factors and sort Income_Category
bankChurners <- bankChurners %>% mutate_if(is.character, as.factor)
bankChurners$Income_Category <- factor(
  bankChurners$Income_Category,
  levels = c(
    "Less than $40K",
    "$40K - $60K",
    "$60K - $80K",
    "$80K - $120K",
    "$120K +",
    "Unknown"
  )
)

# Configuring names to be compatible with r
levels(bankChurners$Attrition_Flag) <- make.names(levels(bankChurners$Attrition_Flag))
```

### 1.1.2 Variable description

The **bankChurners** dataset is composed of 20 variables, which include the following fields:

#### 1.1.2.1 Target Variable

The target variable for this analysis is ***Attrition_Flag***. A value of *Attrited Customer* indicates that the customer ended their relationship with the bank, while *Existing Customer* indicates an active account.

#### 1.1.2.2 Predictor Variables

| Variable | Type | Description |
|---|---|---|
| ***Customer_Age*** | Integer | Customer age in years. |
| ***Gender*** | Factor | Customer gender: 'F' for female and 'M' for male. |
| ***Dependent_count*** | Integer | Number of persons who are financially dependent on the customer. |
| ***Education_Level*** | Factor | Customer education level: 'Unknown', 'Uneducated', 'High School', 'Graduate', 'Post-Graduate', 'College', and 'Doctorate'. |

*(continued)*

| Variable | Type | Description |
|---|---|---|
| ***Marital_Status*** | Factor | Customer marital status: 'Unknown', 'Single', 'Married', and 'Divorced'. |
| ***Income_Category*** | Factor | Customer annual income range: 'Less than $40K', '$40K - $60K', '$60K - $80K', '$80K - $120K', '$120K +', and 'Unknown'. |
| ***Card_Category*** | Factor | Customer card type: 'Blue', 'Silver', 'Gold', and 'Platinum'. |
| ***Months_on_book*** | Integer | Length of relationship with the bank in months. |
| ***Total_Relationship_Count*** | Integer | Total number of different products held by the customer. |
| ***Months_Inactive_12_mon*** | Integer | Months without activity in the last year. |
| ***Contacts_Count_12_mon*** | Integer | Number of times the customer contacted the bank in the last year. |
| ***Credit_Limit*** | Numeric | Average credit limit over the last year. |
| ***Total_Revolving_Bal*** | Integer | Average total revolving balance over the last year. |
| ***Avg_Open_To_Buy*** | Numeric | Average available credit line in the last year (Credit_Limit - Total_Revolving_Bal). |
| ***Total_Amt_Chng_Q4_Q1*** | Numeric | Change in transaction amount (Q4 over Q1). |
| ***Total_Trans_Amt*** | Integer | Total amount of transactions in the last year. |
| ***Total_Trans_Ct*** | Integer | Total number of transactions in the last year. |
| ***Total_Ct_Chng_Q4_Q1*** | Numeric | Change in the number of transactions (Q4 over Q1). |
| ***Avg_Utilization_Ratio*** | Numeric | Average percentage of credit line usage (0 to 1). |

### 1.1.3 Last Data Transformation

In order for the system to process the information in the text columns of the **bankChurners** dataset, I converted them to numerical format using the `dummyVars()` function from the caret package **[3, pp. 42-43]** and stored them in **bankChurners_num**. Thus, each text option is now represented by numbers so that the models used in this project can work with them. The ***Attrition_Flag*** variable was kept as a factor because it is the dependent variable to be predicted.

```
# Convert character columns into columns with numbers
bankChurners_num <- bankChurners %>%
  {data.frame(
    # Keep the target variable as a factor for classification
    Attrition_Flag = .$Attrition_Flag,
    predict(
      # dummyVars creates an array where each category is a new numeric column
      dummyVars(~ ., data = select(., -Attrition_Flag), fullRank = TRUE),
      newdata = .
    )
  )}
```

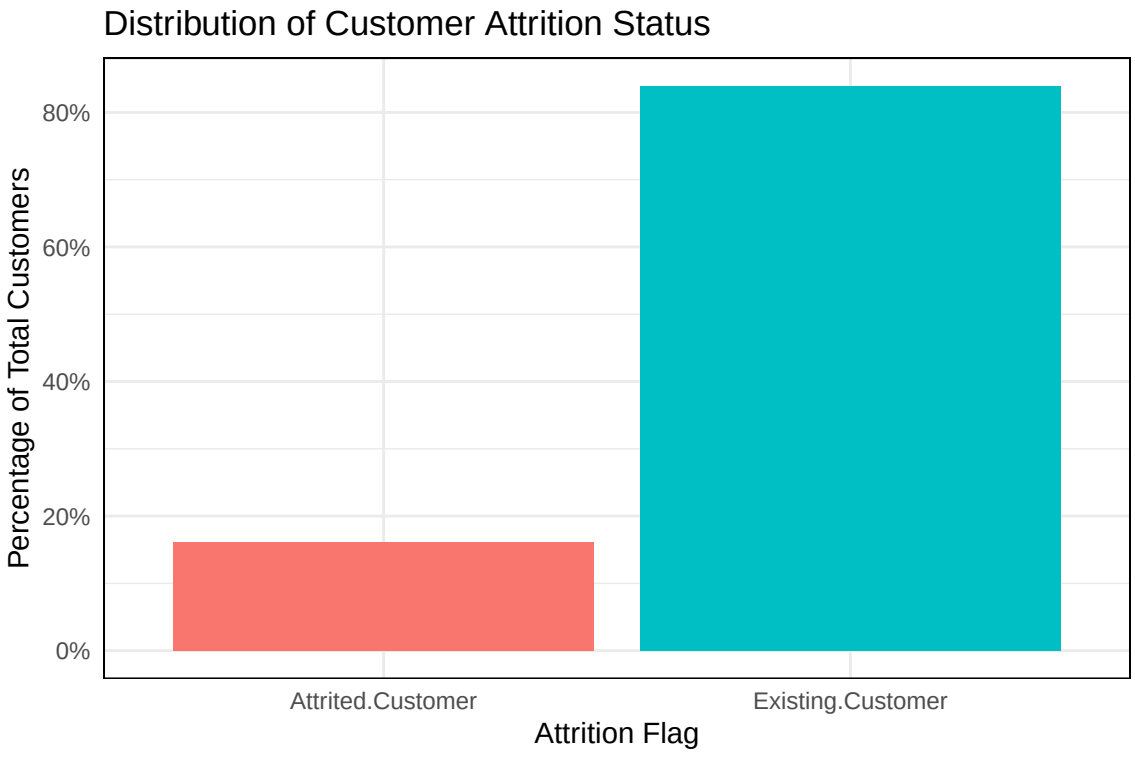
### 1.1.4 Data Splitting

To evaluate the performance of the models, I divided the data into three parts. First, I stored 10% as a final test set in **final_holdout_test**, which will be used exclusively to validate the final model at the end of the project. The remaining 90% was divided into a training group (80%) in **train_set** so that the models could learn to identify patterns in customers, and a test group (20%) in **test_set** to adjust and compare the results of the different models.

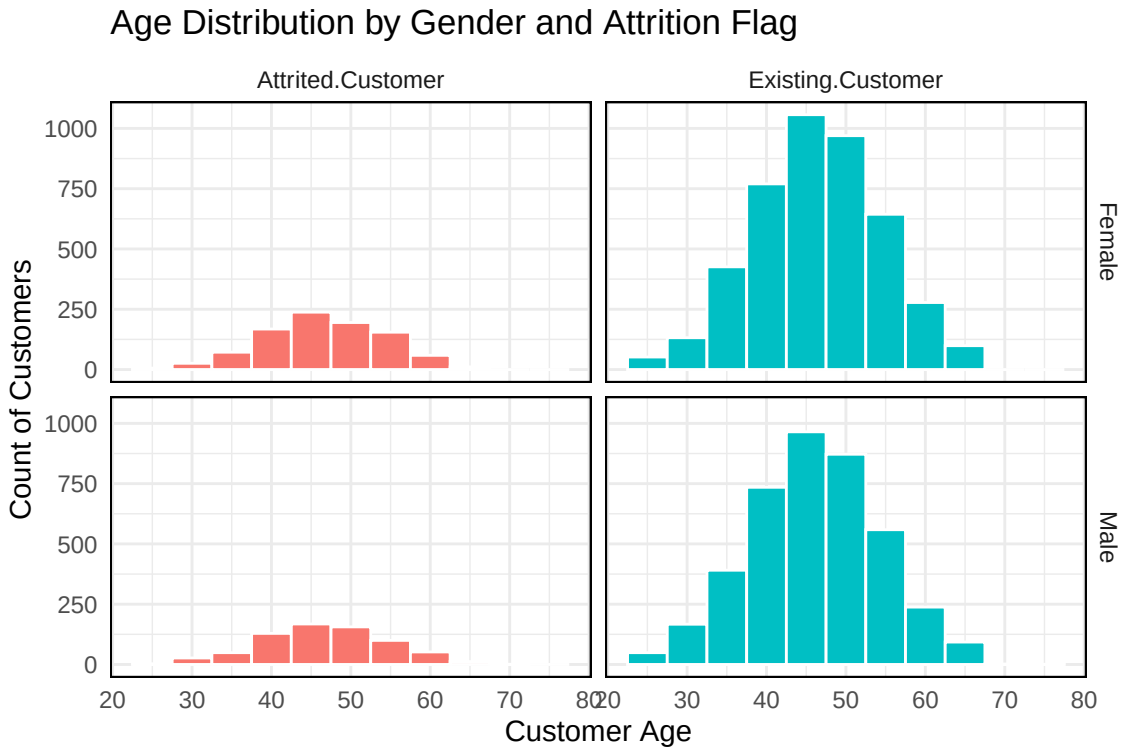
```
# Final hold-out test set will be 10% of bankChurners_num data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = bankChurners_num$Attrition_Flag, p = 0.1, list =
↪  FALSE)
total_train_set <- bankChurners_num[-test_index,]
final_holdout_test <- bankChurners_num[test_index,]

# Divide the rest into train_set (80%) and test_set (20%).
test_index <- createDataPartition(y = total_train_set$Attrition_Flag, p = 0.2, list =
↪  FALSE)
test_set <- total_train_set[test_index,]
train_set <- total_train_set[-test_index,]
```

# 2 Exploratory Data Analysis (EDA)

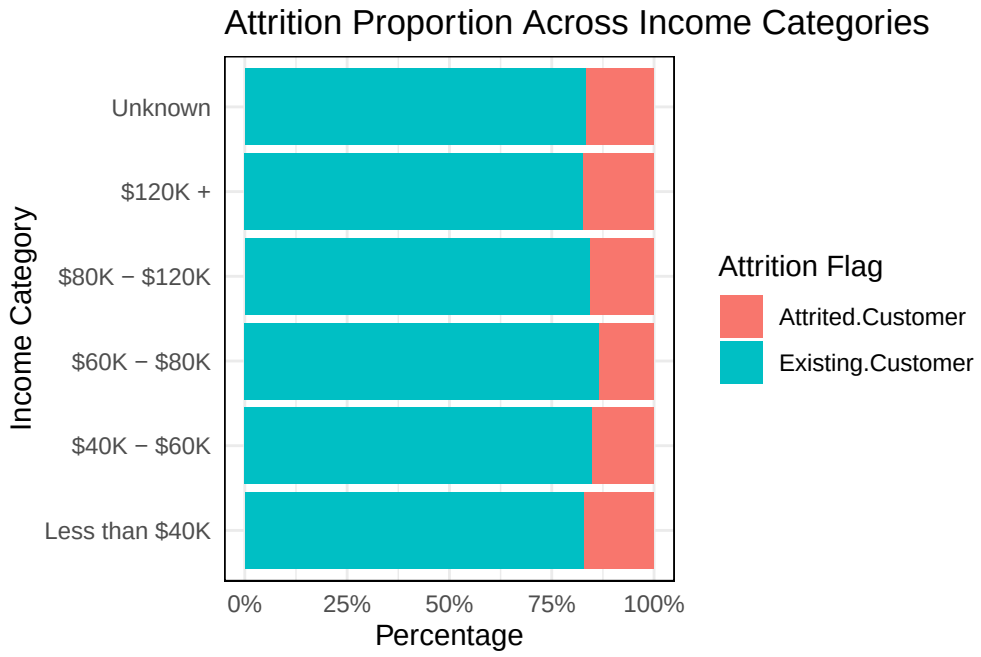
In this section, I will use plots to show the characteristics of the **bankChurners** dataset. We begin by analyzing the target variable, ***Attrition_Flag***. As can be seen in the graph, only 16% of customers fall into the *Attrited Customer* category; therefore, the models may have problems correctly identifying churn risk. I took this fact into account when choosing the comparison metrics to select the best model.



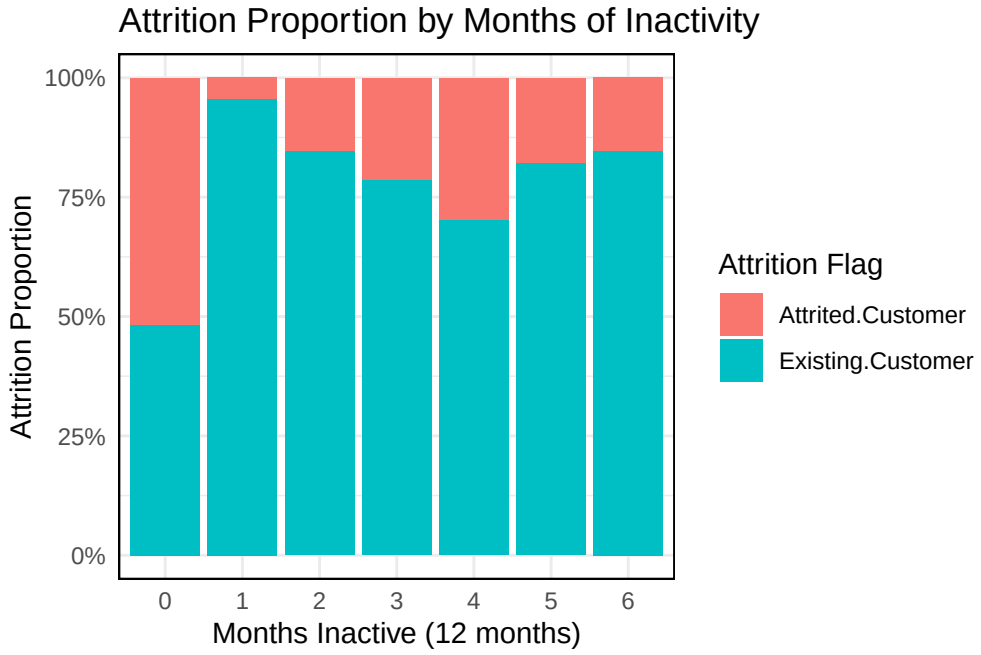
Another factor I analyzed was the age distribution segmented by gender and attrition status. The plots show that the customer age distribution follows a normal distribution centered around 45 years old, regardless of gender or attrition status. There is no clear indication that age or gender are significant factors in predicting churn.



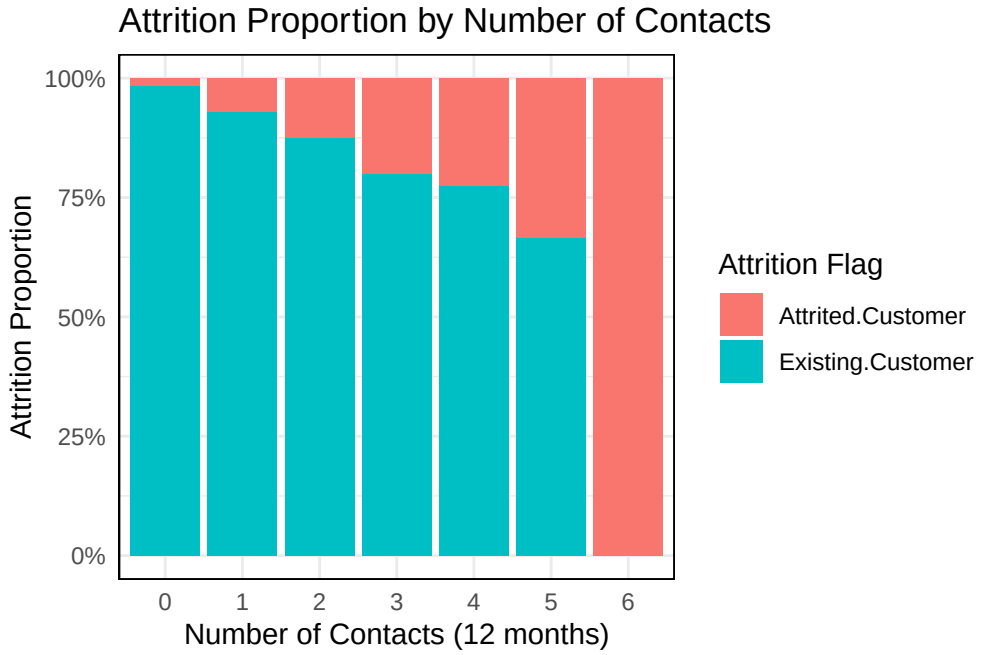

Next, I analyzed the churn proportion according to customer income range. This plot shows that all income ranges maintain similar proportions in customer attrition status, suggesting that this variable is not as relevant when identifying those at risk of churn.



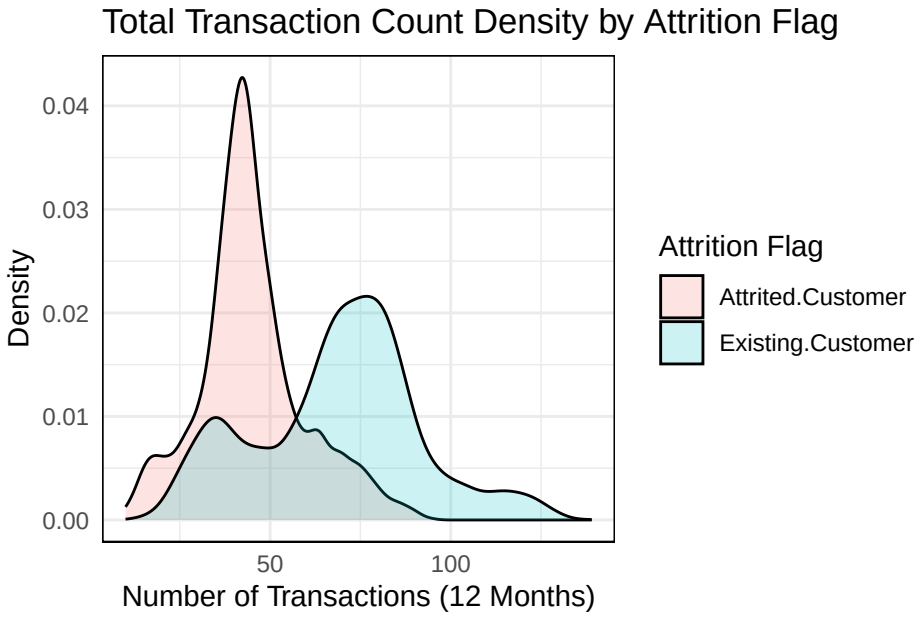
I also analyzed the churn rate based on the number of months of inactivity in the last year. It is noteworthy that customers with 0 months of inactivity have the highest churn rate and that, after 4 months, the proportion of attrited customers decreases. This suggests that the critical window to detect and retain customers at churn risk is within the first 4 months of inactivity.



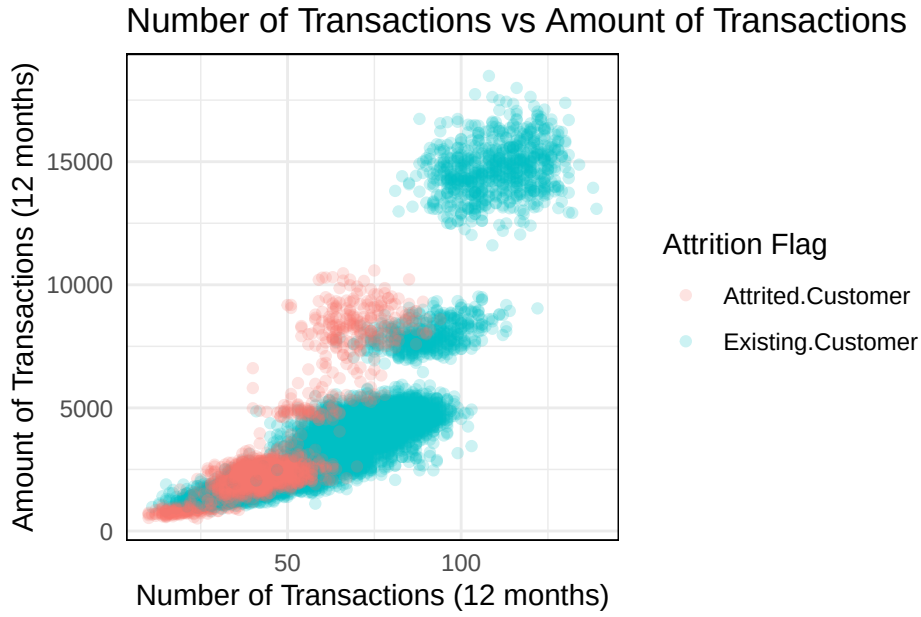

Subsequently, I analyzed the number of contacts the customer had with the bank in the last 12 months. It is important to note that as the number of contacts increases, so does the proportion of attrited customers, with significant spikes at 3, 5, and 6 contacts. Notably, every customer who reached 6 contacts left the bank. This demonstrates that a high number of contacts is a strong predictor of churn risk.



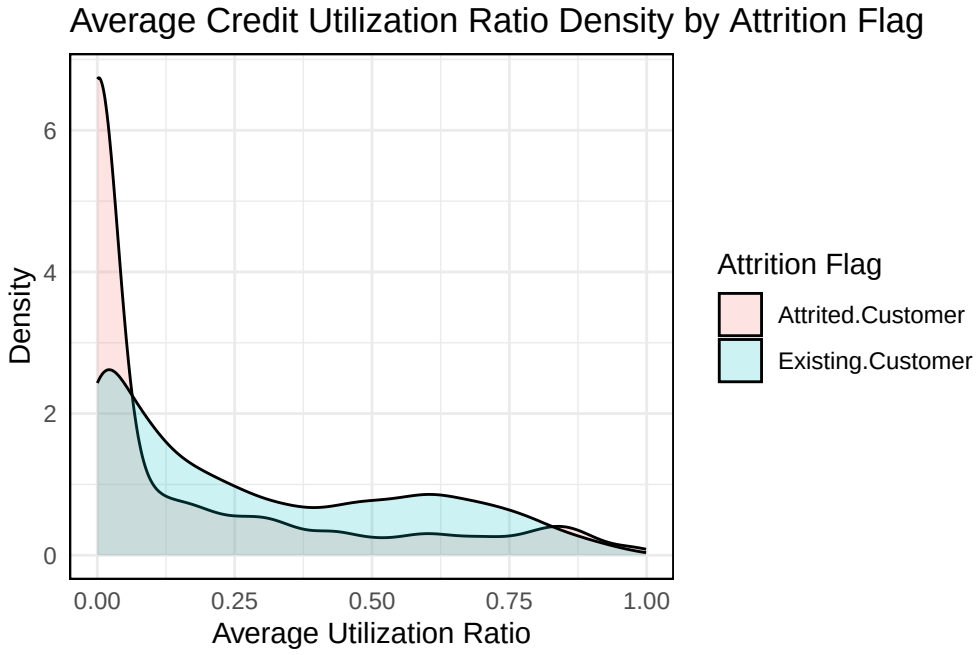
I continue the analysis by observing customer activity in terms of the number of transactions carried out in the last year. It can be seen that the group of attrited customers tends to carry out few transactions, reaching its peak in the graph at around 40 transactions. On the other hand, active customers show a higher volume of activity, with a maximum of around 75 transactions. This distinction suggests that the number of transactions is a key variable for predicting the risk of churn.




With respect to transactions, and understanding the direct relationship between their number and the total amount, I decided to analyze this interaction to visualize how both groups of customers are distributed. The plot reveals that customers who left the bank are almost entirely grouped in the area of few movements and low amounts. In contrast, as activity increases, it can be seen that from around $3,000 and a higher number of transactions, the presence of customers leaving the bank decreases rapidly until it disappears completely when reaching 80 transactions and approximately $12,500 in total amount. This clear difference in grouping is a determining factor in identifying customers at greater risk of churn.



Finally, I analyzed the average credit utilization ratio, which measures how much customers use their available credit limit. The plot shows that attrited customers are significantly concentrated near zero, whereas active customers are distributed across the entire range. This suggests that customers who do not use their card are at greater risk of churn.




# 3 Modeling

Based on the findings of the exploratory data analysis, I will proceed to develop predictive models of churn risk.

## 3.1 K-Nearest Neighbors

The first model I decided to implement is K-Nearest Neighbors (KNN). According to the program and the textbook "Introduction to Data Science" [**4, pp. 535-536**], KNN is an algorithm that estimates the conditional probability of belonging to a certain class in a manner similar to bin smoothing, but with the ability to adapt to multiple dimensions. For any point for which we want a prediction, the algorithm identifies the k nearest points and estimates the conditional probability by averaging the outcomes in that neighborhood. The observation is then assigned to a category based on this estimation.

In the plots, we saw that there are some customer characteristics with a clear distinction between existing and attrited customers. Therefore, I considered KNN to be a good algorithm to start modeling, given its ability to classify customers with patterns similar to others in their neighborhood.

Before training the models, as taught in the program, we will use k-fold cross validation using the functions seen in the caret package: `trainControl()` and `train()` [**4, pp. 551-554**].

To do this, I first familiarized myself with the necessary parameters by consulting the "A Short Introduction to the caret package" guide in the official package documentation [**5**]. Based on this, I configured the training parameters, specifying the cross-validation method (`method="cv"`) with 10 folds (`number=10`).

On the other hand, since this is a binary classification problem, the documentation **[5]** indicates that the `summaryFunction` parameter (which is used to pass in a function that takes the observed and predicted values and estimate some measure of performance) should be `twoClassSummary` to calculate metrics such as ROC, sensitivity, and specificity. However, since the ROC curve is based on the predicted class probabilities (which are not computed automatically), it is necessary to set the `classProbs=TRUE` option to include those calculations.

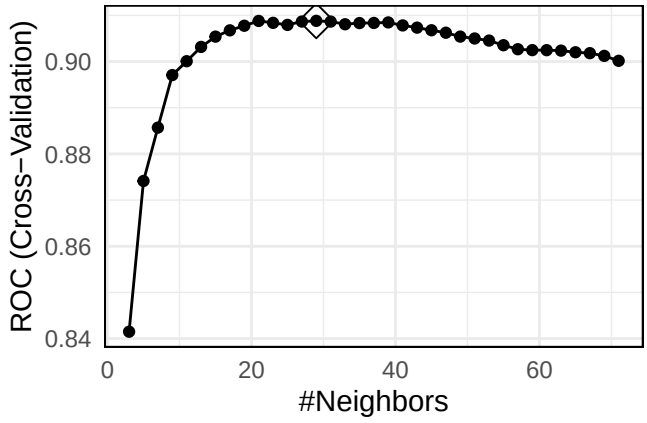
As I progress, I will compare the performance of different approaches to identify the most effective model for predicting churn. To facilitate this comparison, I created an empty tibble to store the outcomes of each model.

```
# Define the training control parameters
control <- trainControl(
  method="cv", # Cross-Validation
  number = 10, # Use 10 folds
  classProbs = TRUE, # Calculate class probabilities to enable ROC curve analysis
  summaryFunction = twoClassSummary # Use ROC, Sensitivity, and Specificity as evaluation
  ↪  metrics
)

# Create a tibble to store all the results of the different approaches
model_results <- tibble()
```

With the training control set, I will proceed to train the KNN model using the `train()` function. As seen in the program, to avoid over-training and over-smoothing, it is necessary to choose an optimal `k` **[4, pp. 535-540]**, so I tested 35 different values of `k` between 3 and 71.

```
# Train a KNN model by testing 35 values of k between 3 and 71 using the
# 10-fold cross validation created previously through the 'control' variable and
# ensuring that the best k is selected based on ROC instead of Accuracy
set.seed(1, sample.kind="Rounding")
fit_knn <- train(
  Attrition_Flag ~ .,
  method = "knn",
  data = train_set,
  tuneGrid = data.frame(k = seq(3, 71, 2)),
  trControl = control
)
```

After training, the following plot shows the `k` value that maximizes the area under the ROC curve.



The `k` value highlighted in the plot is the one below:

```
fit_knn$bestTune
```

```
##     k
## 14 29
```

I will evaluate each model's performance on the test set using the `predict()` function. To provide a comprehensive assessment, I will employ two complementary approaches: first, the `confusionMatrix()` function, which offers a detailed breakdown of predictions against actual values along with key classification statistics; and second, the `twoClassSummary()` function **[3, pp. 32-34]**, to specifically capture the Area Under the ROC Curve (AUC), Sensitivity, and Specificity.

Analyzing the results of the confusion matrix, the model achieved an accuracy of 88.21%, which exceeds the No Information Rate of 83.93%, which is the accuracy that would be obtained if it were simply assumed that all customers remain with the bank. However, it showed a specificity of 95.56%, demonstrating a great ability to correctly detect existing customers, but with a sensitivity of only 49.83%, indicating that it only successfully detects half of the attrited customers, which is in fact the main objective of the project and for which there is room for improvement.

```
cm_knn <- confusionMatrix(
  predict(fit_knn, test_set, type = "raw"),
  test_set$Attrition_Flag
)
cm_knn
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Attrited.Customer Existing.Customer
##   Attrited.Customer               146                68
##   Existing.Customer               147              1462
##
##                Accuracy : 0.8821
##                  95% CI : (0.8664, 0.8965)
##     No Information Rate : 0.8393
##     P-Value [Acc > NIR] : 1.407e-07
##
##                   Kappa : 0.5094
##
##  Mcnemar's Test P-Value : 1.040e-07
##
##             Sensitivity : 0.49829
##             Specificity : 0.95556
##          Pos Pred Value : 0.68224
##          Neg Pred Value : 0.90864
##              Prevalence : 0.16072
##          Detection Rate : 0.08009
##    Detection Prevalence : 0.11739
##       Balanced Accuracy : 0.72692
##
##        'Positive' Class : Attrited.Customer
##
```

Subsequently, I observed that the model achieved an AUC of 0.9166 despite the low sensitivity observed, which shows good discriminatory power and provides an additional metric for comparison with the following models.

```
# Create a dataframe in the format required for the twoClassSummary function
TCSdf <-data.frame(
  obs = test_set$Attrition_Flag,
  predict(fit_knn, test_set, type = "prob"),
  pred = predict(fit_knn, test_set, type = "raw")
)

# Display the ROC, sensitivity, and specificity metrics
results <- twoClassSummary(
  TCSdf,
  lev = levels(TCSdf$obs)
)
results
```

```
##       ROC      Sens      Spec
## 0.9166131 0.4982935 0.9555556
```

I store the model outcomes in the tibble created for comparing results.

```
# Store KNN performance metrics into the tibble
model_results <- bind_rows(
  model_results,
  tibble(
    Method = "k-nearest neighbors",
    Accuracy = cm_knn$overall["Accuracy"],
    ROC = results["ROC"],
    Sensitivity = results["Sens"],
    Specificity = results["Spec"]
  )
)
model_results
```

```
## # A tibble: 1 x 5
##   Method              Accuracy   ROC Sensitivity Specificity
##   <chr>                  <dbl> <dbl>       <dbl>       <dbl>
## 1 k-nearest neighbors    0.882 0.917       0.498       0.956
```

## 3.2 Binomial Logistic Regression

Following the program content, I decided to implement a binomial logistic regression. This model is a specific case, and the most commonly used, of the Generalized Linear Models (GLM), where $Y$ represents the customer attrition status (1 for attrited customers and 0 for existing customers) and $X$ represents the predictors, which include demographics and the frequency and manner of product usage [**4, p. 561**].

The main advantage of this approach is that it allows us to estimate the conditional probability of churn, denoted as $Pr(Y = 1|X = x)$, ensuring that this estimate always remains within the logical range of 0 to 1 through the logistic transformation:

$$g(p) = log\left(\frac{p}{1-p}\right)$$

This transformation converts probabilities into log odds, making the relationship symmetrical around 0 and providing a potentially better approximation of the expected result in the project [**4, pp. 561-563**].

Using the previously defined control set again to ensure consistent comparison, I proceed to train the logistic regression model using the `train()` function.

```
# Train a binomial logistic regression model using the 10-fold cross validation
# created previously through the 'control' variable and using ROC as the primary
# evaluation metric for comparison
set.seed(1, sample.kind="Rounding")
fit_glm <- train(
  Attrition_Flag ~ .,
  method = "glm",
  family = "binomial",
  data = train_set,
  trControl = control
)
```

When analyzing the confusion matrix, a general improvement in performance can be observed. Accuracy increased to 89.85%, specificity also increased slightly to 95.82%, and sensitivity saw a very good increase, rising to 58.70%. This means that the model is now able to correctly identify almost 60% of customers at risk of churn without losing the ability to detect existing customers, significantly outperforming the KNN model.

```
cm_glm <- confusionMatrix(
  predict(fit_glm, test_set, type = "raw"),
  test_set$Attrition_Flag
)
cm_glm
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           Attrited.Customer Existing.Customer
##   Attrited.Customer               172                64
##   Existing.Customer               121              1466
##
##                Accuracy : 0.8985
##                  95% CI : (0.8837, 0.912)
##     No Information Rate : 0.8393
##     P-Value [Acc > NIR] : 1.968e-13
##
##                   Kappa : 0.5917
##
##  Mcnemar's Test P-Value : 3.835e-05
##
```

```
##             Sensitivity : 0.58703
##             Specificity : 0.95817
##          Pos Pred Value : 0.72881
##          Neg Pred Value : 0.92376
##              Prevalence : 0.16072
##          Detection Rate : 0.09435
##    Detection Prevalence : 0.12946
##       Balanced Accuracy : 0.77260
##
##        'Positive' Class : Attrited.Customer
##
```

Looking at the AUC, there was a marginal decrease compared to the previous model, reaching 0.9161. However, this is not a concern as the model maintains strong discriminatory power while achieving a notable 10% improvement in the ability to detect churn risk. Despite these improvements, the current sensitivity still shows room for progress.

```
# Create a dataframe in the format required for the twoClassSummary function
TCSdf <-data.frame(
  obs = test_set$Attrition_Flag,
  predict(fit_glm, test_set, type = "prob"),
  pred = predict(fit_glm, test_set, type = "raw")
)

# Display the ROC, sensitivity, and specificity metrics
results <- twoClassSummary(
  TCSdf,
  lev = levels(TCSdf$obs)
)
results
```

```
##       ROC      Sens      Spec
## 0.9161570 0.5870307 0.9581699
```

I store the model outcomes in the tibble created for comparing results.

```
# Store glm performance metrics into the tibble
model_results <- bind_rows(
  model_results,
  tibble(
    Method = "binomial logistic regression",
    Accuracy = cm_glm$overall["Accuracy"],
    ROC = results["ROC"],
    Sensitivity = results["Sens"],
    Specificity = results["Spec"]
  )
)
model_results
```

```
## # A tibble: 2 x 5
##   Method                       Accuracy   ROC Sensitivity Specificity
##   <chr>                           <dbl> <dbl>       <dbl>       <dbl>
## 1 k-nearest neighbors             0.882 0.917       0.498       0.956
## 2 binomial logistic regression    0.899 0.916       0.587       0.958
```

### 3.3 Random Forest

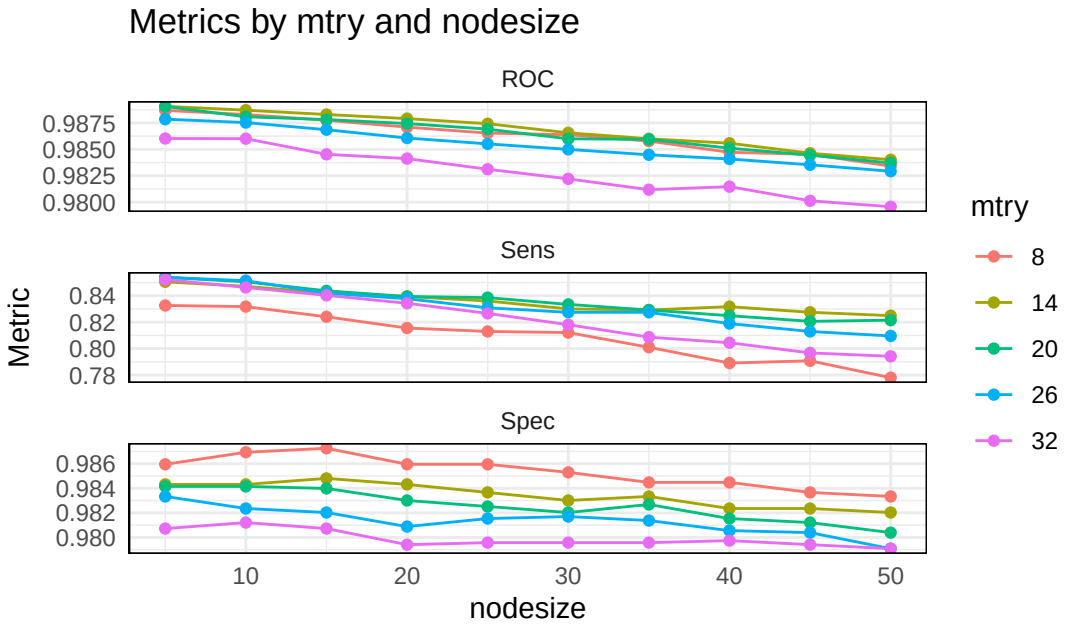
In the models trained so far, although they perform well in predicting the risk of churn, they show a bias towards detecting existing customers and deficiencies in the effective detection of attrited customers (reaching only around 50% to 60% sensitivity). This may be due to the problem described in the textbook known as the *curse of dimensionality* [**4, pp. 582-583**], where the neighborhood required to include a given percentage of data becomes too large as the number of predictors increases, causing the model to lose flexibility. To address this, I will implement a random forest, a model recommended to adapt to this problem.

According to the textbook [**4, p. 594**], random forests improve prediction performance and reduce instability by averaging multiple decision trees. This is achieved through *bootstrap aggregation* or *bagging*, which induces randomness by generating many predictors, each using regression or classification trees, and then forming a final prediction based on the average prediction of all those trees.

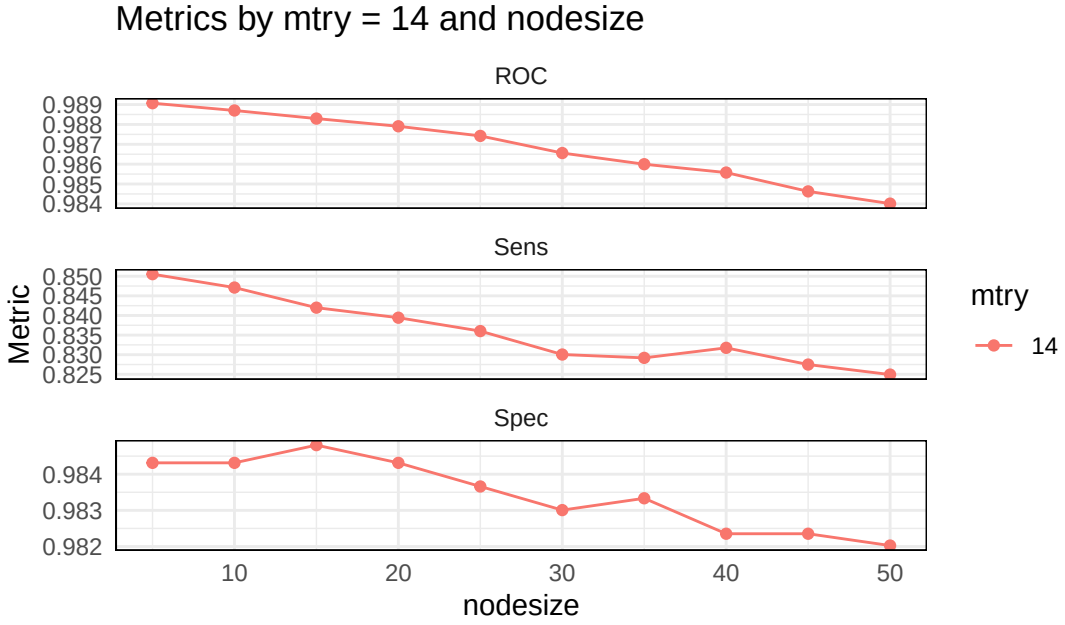
To maximize the model's performance, I adjusted two key random forest parameters: `mtry` (the number of variables randomly sampled as candidates at each split) and `nodesize` (the minimum size of terminal nodes) [**6, pp. 17-22**]. I implemented this by iterating over 10 different `nodesize` values between 5 and 50 using `sapply()`. Within each iteration, I evaluated 6 `mtry` values between 2 and 32 using the `tuneGrid` argument in the `train()` function, all while maintaining the consistency of the previously defined 10-fold cross-validation control set. Finally, the results for the ROC, sensitivity, and specificity metrics for each combination are stored in a tibble named **parameters** for further analysis.

```r
# mtry and nodesize range to optimize random forest parameters
mtry <- data.frame(mtry = seq(2, 32, 6))
nodesize <- seq(5, 50, 5)

# Create a tibble to store all the results of the different parameters
parameters <- tibble()

# Iterate over each nodesize to evaluate performance with each mtry
sapply(nodesize, function(ns) {
  set.seed(1, sample.kind="Rounding")
  # Train a random forest model using the 10-fold cross validation
  # created previously through the 'control' variable
  rm <- train(
    Attrition_Flag ~ .,
    method = "rf",
    data = train_set,
    tuneGrid = mtry,
    trControl = control,
    nodesize = ns
  )$results
  # Store the results of ROC, Sensitivity, and Specificity.
  parameters <<- bind_rows(
    parameters,
    tibble(
      mtry = rm$mtry,
      nodesize = ns,
      ROC = rm$ROC,
      Sens = rm$Sens,
      Spec = rm$Spec
    )
  )
})
```

When analyzing the results of the parameter optimization, we observe variability in performance across the different metrics driven by the `mtry` value. An `mtry` of 2 showed considerably lower performance than the other selected parameters, specifically a decrease of 0.014 in AUC compared to the next lowest value, whereas subsequent parameters showed differences between them of only around 0.001. Based on this, `mtry = 2` was discarded and removed from the visualization to allow for a clearer comparison of the top performing configurations. Looking at the remaining metrics, `mtry = 14` was selected as it consistently maximizes the AUC across all `nodesize` values. Next, I will show the specific metrics for this parameter to determine the optimal `nodesize`.




When reviewing the specific metrics for `mtry = 14`, it can be seen that the `nodesize` that maximizes both AUC and sensitivity is 5. However, the data also shows that increasing the `nodesize` to 10 results in only marginal decreases in both metrics. As a result, I preferred to select a `nodesize` of 10 to achieve, as suggested in the textbook [**4, p. 597**], a smoother final estimate. This choice reduces the noise associated with very small nodes.



Using the previously defined control set again to ensure consistent comparison, I proceed to train the random forest model using the `train()` function with the selected parameters.

```
set.seed(1, sample.kind="Rounding")
fit_rf <- train(
  Attrition_Flag ~ .,
  method = "rf",
  data = train_set,
  tuneGrid = data.frame(mtry = 14),
  trControl = control,
  nodesize = 10,
  importance = TRUE
)
```

The analysis of the confusion matrix and statistics shows significantly better results than previous models, achieving an Accuracy of 96.27% with a Sensitivity of 88.40%. The model correctly identified 259 of the 293 customers who actually attrited, which largely meets the primary objective of the analysis. Furthermore, this strong performance in sensitivity did not have a negative impact on specificity, as 97.78% of existing customers were also correctly classified.

```
cm_rf <- confusionMatrix(
  predict(fit_rf, test_set, type = "raw"),
  test_set$Attrition_Flag
)
cm_rf
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Attrited.Customer Existing.Customer
##   Attrited.Customer               259                34
##   Existing.Customer                34              1496
##
##                Accuracy : 0.9627
##                  95% CI : (0.9529, 0.9709)
##     No Information Rate : 0.8393
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8617
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8840
##             Specificity : 0.9778
##          Pos Pred Value : 0.8840
##          Neg Pred Value : 0.9778
##              Prevalence : 0.1607
##          Detection Rate : 0.1421
##    Detection Prevalence : 0.1607
##       Balanced Accuracy : 0.9309
##
##        'Positive' Class : Attrited.Customer
##
```

Additionally, the model achieved an AUC of 0.9879, reflecting its great ability to distinguish attrited customers from existing ones.

```
# Create a dataframe in the format required for the twoClassSummary function
TCSdf <-data.frame(
  obs = test_set$Attrition_Flag,
  predict(fit_rf, test_set, type = "prob"),
  pred = predict(fit_rf, test_set, type = "raw")
)

# Display the ROC, sensitivity, and specificity metrics
results <- twoClassSummary(
  TCSdf,
  lev = levels(TCSdf$obs)
)
results
```

```
##       ROC      Sens      Spec
## 0.9879230 0.8839590 0.9777778
```

I store the model outcomes in the tibble created for comparing results.

```
# Store rf performance metrics into the tibble
model_results <- bind_rows(
  model_results,
  tibble(
    Method = "random forest",
    Accuracy = cm_rf$overall["Accuracy"],
    ROC = results["ROC"],
    Sensitivity = results["Sens"],
    Specificity = results["Spec"]
  )
)
model_results
```

```
## # A tibble: 3 x 5
##   Method                         Accuracy   ROC Sensitivity Specificity
##   <chr>                             <dbl> <dbl>       <dbl>       <dbl>
## 1 k-nearest neighbors               0.882 0.917       0.498       0.956
## 2 binomial logistic regression      0.899 0.916       0.587       0.958
## 3 random forest                     0.963 0.988       0.884       0.978
```

The comparative metrics for all approaches shown in the table indicate that the Random Forest model is the most effective in meeting the project's objectives. Also, these results confirm that the optimization of the `mtry` and `nodesize` parameters was successful, as they presented significantly superior metrics compared to the KNN and logistic regression models. Therefore, I will proceed to use this final model to evaluate the algorithm's performance on the **final_holdout_test** dataset.

# 4 Results

The results obtained from the selected approach confirm its great ability to manage churn risk, achieving an accuracy of 96.05% on the **final_holdout_set**. Specifically, the model correctly identified 137 of the 163 customers who attrited, reaching a sensitivity of 84.05%, while misclassifying only 14 existing customers reaching a specificity of 98.35%.

```
# Observe the results of predicting using the random forest model in final_holdout_test
cm_final <- confusionMatrix(
  predict(fit_rf, final_holdout_test, type = "raw"),
  final_holdout_test$Attrition_Flag
)
cm_final
```

```
## Confusion Matrix and Statistics
##
##                    Reference
## Prediction          Attrited.Customer Existing.Customer
##   Attrited.Customer               137                14
##   Existing.Customer                26               836
##
##                Accuracy : 0.9605
##                  95% CI : (0.9466, 0.9716)
##     No Information Rate : 0.8391
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8493
##
##  Mcnemar's Test P-Value : 0.08199
##
##             Sensitivity : 0.8405
##             Specificity : 0.9835
##          Pos Pred Value : 0.9073
##          Neg Pred Value : 0.9698
##              Prevalence : 0.1609
##          Detection Rate : 0.1352
##    Detection Prevalence : 0.1491
##       Balanced Accuracy : 0.9120
##
##        'Positive' Class : Attrited.Customer
##
```

The model's performance can be observed through the AUC of 0.9860. This value confirms its power to differentiate between attrited and existing customers.

```
# Create a dataframe in the format required for the twoClassSummary function
TCSdf <-data.frame(
  obs = final_holdout_test$Attrition_Flag,
  predict(fit_rf, final_holdout_test, type = "prob"),
  pred = predict(fit_rf, final_holdout_test, type = "raw")
)
```

```
# Display the ROC, sensitivity, and specificity metrics
results <- twoClassSummary(
  TCSdf,
  lev = levels(TCSdf$obs)
)
results
```

```
##       ROC      Sens      Spec
## 0.9860231 0.8404908 0.9823529
```

I store the final model outcomes in the tibble created for comparing results.

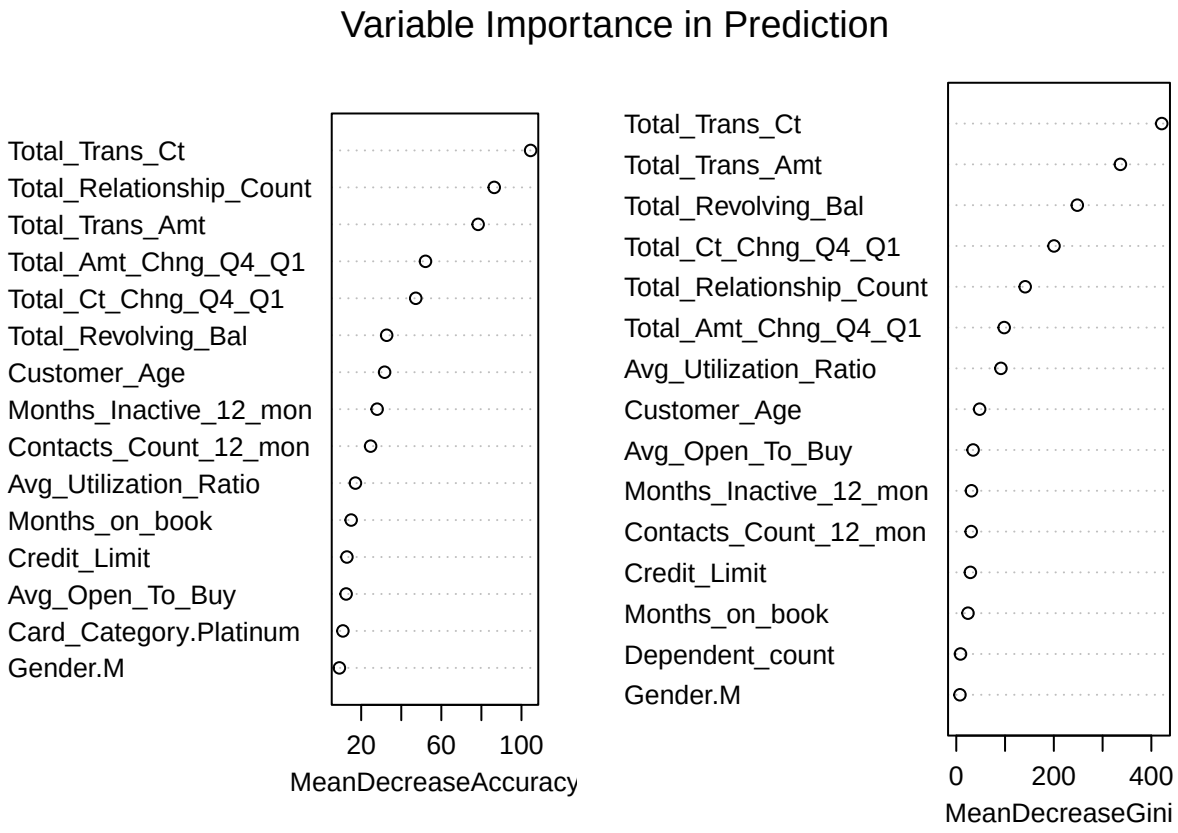
```
# Store final rf performance metrics into the tibble
model_results <- bind_rows(
  model_results,
  tibble(
    Method = "random forest final holdout set",
    Accuracy = cm_final$overall["Accuracy"],
    ROC = results["ROC"],
    Sensitivity = results["Sens"],
    Specificity = results["Spec"]
  )
)
model_results
```

```
## # A tibble: 4 x 5
##   Method                          Accuracy   ROC Sensitivity Specificity
##   <chr>                              <dbl> <dbl>       <dbl>       <dbl>
## 1 k-nearest neighbors                0.882 0.917       0.498       0.956
## 2 binomial logistic regression       0.899 0.916       0.587       0.958
## 3 random forest                      0.963 0.988       0.884       0.978
## 4 random forest final holdout set    0.961 0.986       0.840       0.982
```

Finally, while the program textbook mentions the varImp function from the caret package to extract variable importance, I opted for the `varImpPlot()` function from the randomForest package instead, which allows the importance of predictors to be visualized using two dot charts. As described in "An Introduction to Statistical Learning" **[7, p. 330]**, the first chart shows the average decrease in prediction accuracy on out-of-bag samples when a given variable is excluded from the model, while the second shows the average decrease in the Gini coefficient, which measures the total decrease in node impurity resulting from splits over that variable.

The resulting plots reveal that the most important predictors are the frequency and amount of transactions over the last year (***Total_Trans_Ct*** and ***Total_Trans_Amt***), along with the fluctuations between the first and fourth quarters (***Total_Ct_Chng_Q4_Q1*** and ***Total_Amt_Chng_Q4_Q1***). These metrics indicate that a reduction in card usage is the most significant warning sign for attrition. In addition, the model highlights the importance of the total number of products held by the customer (***Total_Relationship_Count***) and the average revolving balance (***Total_Revolving_Bal***).

In the case of demographic variables, these were generally less important, with the exception of customer age, which remained a significant factor.




# 5 Conclusion

The results of the Random Forest model implemented in the **final_holdout_test** proved to be the most effective algorithm for predicting customer churn risk, achieving 96.05% accuracy and effectively identifying 84.05% of churned customers (sensitivity). This significantly outperformed the KNN and GLM models implemented in the **test_set**, which showed sensitivities of approximately 50% and 60%, respectively. In addition, the model identified that the frequency and manner in which customers use their products are the main predictors of customer attrition status, surpassing demographic characteristics. Therefore, implementing this system meets the objective of effectively detecting customers at risk of churn, allowing the bank to contact them proactively and offer better services to influence their decision, thus minimizing customer loss and its associated costs.

However, the model has certain limitations: it is based on a dataset from only the last year, which may not capture long term seasonal trends; it relies on a relatively small sample of 10,127 rows, which might not reflect overall customer patterns as the volume increases significantly; and it does not take external economic factors into account. Therefore, as future work, a system for continuous data integration could be established to keep the model updated, additional research could incorporate external economic variables, and the process of sending personalized offers to customers at risk could be automated.

# 6 References

[**1**] O'Brien, K., & Downie, A. (n.d.). *What is customer churn?*. IBM: https://www.ibm.com/think/topics/customer-churn

[**2**] Goyal, S. (2021). *Credit card customers* [Dataset]. Kaggle: https://www.kaggle.com/datasets/sakshigoyal7/credit-card-customers/data

[**3**] Kuhn, M. (2025, July 22). *Package 'caret'*. CRAN: https://cran.r-project.org/web/packages/caret/caret.pdf

[**4**] Irizarry, R. (2019, October 24). *Introduction to data science*. Leanpub: https://leanpub.com/datasciencebook

[**5**] Kuhn, M. (n.d.). *A short introduction to the caret package*. CRAN: https://cran.r-project.org/web/packages/caret/vignettes/caret.html

[**6**] Breiman, L., Cutler, A., Liaw, A., & Wiener, M. (2025, July 23). *Package 'randomForest'*. CRAN: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[**7**] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in R*. Springer.