OPIM 407

Term Project

Selin Ceydeli

Sinan Yılmaz

Göktuğ Oktaylar

12.06.2023

A. INTRODUCTION

In this study, our objective is to address the customer churn problem faced by businesses. Customer churn refers to the phenomenon where businesses lose their clientele. This case study will specifically focus on predicting churn behavior among bank customers. To tackle this problem, we will first visualize the factors affecting customer churn behavior within a banking context, followed by the development of predictive models capable of classifying whether a customer is likely to churn or not. This strategy will help the banks in optimizing their strategies to retain their customers.

Section B of this report will provide a brief introduction to the data used. In Section C, we will present the results of the descriptive statistics and exploratory data analysis conducted using the R programming language, aiming to analyze and investigate the data to understand the main characteristics of the variables. In Section D, the details regarding data preparation and partitioning will be presented. In Section E, we will discuss the factors influencing the churn behavior of the bank customers and explain the implementation of feature selection. Section F will detail the implementation of two predictive data mining methods aimed at classifying customer churn likelihood. We will also compare the predictive performances of these models and summarize our findings. In Section G, we will implement two ensemble methods, followed by a discussion of their results. Finally, in Section H, we will present our conclusions, summarizing the main findings of this case study.

B. BRIEF INTRODUCTION TO DATA

2.1 Number of Variables and Observations

The dataset assembled for understanding customer churn behavior within a banking context consists of 10000 records, where each record represents an individual customer's details. Each customer is characterized by 13 variables, providing comprehensive information about the customer. Figure 1 presents the R code used to determine the dimensions of the dataset:

```
> # Get the dimensions of the dataset (rows and columns)
> dimensions <- dim(data)
> dimensions
[1] 10000    14
```

Figure 1: Dimensions of the data

2.2 List of Attributes in the Data

Table 1 below displays a list of the variables in the data, specifying each feature and describing its type and measurement unit.

Table 1. List of Attributes in the Data

Variable	Description	Feature Type	Measurement Unit
CustomerID	ID of a customer	Identifier	15634602, 15647311, 15619304, …
Surname	Last name of a customer	Categorical	Hargrave, Hill, Onio, …
Credit_Score	Credit score of a customer	Quantitative	Min: 350, Max: 850
Geography	Country information of customer	Categorical	France, Spain, or Germany
Gender	Gender of a customer	Categorical	Male or Female
Age	Age of customer	Quantitative	Min: 18, Max: 92
Tenure	Time spent with the bank	Quantitative	Min: 0, Max: 10 in years
Balance	Money in the bank	Quantitative	Min: 0, Max: 250898 in Euros
NumOfProducts	The number of products bought by the customer among the various services offered by a bank to its customers	Quantitative	Min: 1, Max: 4
HasCrCard	Whether the customer has a credit card	Categorical	Yes = 1, No = 0
IsActiveMember	Whether the customer is active	Categorical	Yes = 1, No = 0
EstimetedSalary	Estimated salary of the customer	Quantitative	Min: 11.58, Max: 199992.48 in Euros
Exited	Whether the customer exited the bank	Categorical	Yes = 1, No = 0

2.3 Missing Value Analysis

Before proceeding with exploratory data analysis on the dataset, the missing values should be handled. Therefore, we checked if there exist any missing values in our dataset using R's is.na() function and as can be seen in Figure 2, we observed that there aren't any missing values in either of the columns:

```
> # Are there any null values in the dataset?
> sapply(data, function(x) sum(is.na(x))) #There aren't any NA values
     RowNumber     CustomerId        Surname    CreditScore      Geography         Gender            Age         Tenure 
             0              0              0              0              0              0              0              0 
       Balance  NumOfProducts      HasCrCard IsActiveMember EstimatedSalary         Exited 
             0              0              0              0              0              0 
```

Figure 2. Missing Value Analysis

C. EXPLORATORY DATA ANALYSIS

As we were introducing the data in the previous section and reviewed the entries in the columns, we encountered some puzzling aspects regarding the dataset. We realized that certain customers who had reportedly exited the bank still had account information, including a balance, present in the bank's dataset. We do not know why the account information of these customers are still present in the bank's dataset. Maybe, the customer have exited from a certain service provided by the bank but did not withdraw his money completely.

Another question that arose was regarding the meaning of being an active member. We are assuming that being an active member has degrees to it, which is designated by the number of services the customer bought from the bank or by the time it spent in the bank. Unfortunately, we do not have answers to these questions as in Kaggle, neither a detailed description of the dataset nor the columns were not provided. Regardless, we inferred the descriptions of the variables from the context, as we reported in Table 2 above. Therefore, we decided to continue on with exploratory data analysis to gain more insights about the data, after which we can proceed with feature selection and the development of predictive models to predict churn behavior of the customers.

3.1 The Dataset

Prior to visualizing the data, the structure of the dataset is checked. Figure 3 demonstrates that all the variables, except for Surname, Geography, and Gender, are represented as numerical variables in the dataset:

```
> # Displaying the types of the variables (i.e columns of data frame)
> str(data)
'data.frame':   10000 obs. of  14 variables:
 $ RowNumber      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ CustomerId     : int  15634602 15647311 15619304 15701354 15737888 15574012 15592531 15656148 15792365 15592389 ...
 $ Surname        : chr  "Hargrave" "Hill" "Onio" "Boni" ...
 $ CreditScore    : int  619 608 502 699 850 645 822 376 501 684 ...
 $ Geography      : chr  "France" "Spain" "France" "France" ...
 $ Gender         : chr  "Female" "Female" "Female" "Female" ...
 $ Age            : int  42 41 42 39 43 44 50 29 44 27 ...
 $ Tenure         : int  2 1 8 1 2 8 7 4 4 2 ...
 $ Balance        : num  0 83808 159661 0 125511 ...
 $ NumOfProducts  : int  1 1 3 2 1 2 2 4 2 1 ...
 $ HasCrCard      : int  1 0 1 0 1 1 1 1 0 1 ...
 $ IsActiveMember : int  1 1 0 0 1 0 1 0 1 1 ...
 $ EstimatedSalary: num  101349 112543 113932 93827 79084 ...
 $ Exited         : int  1 0 1 0 0 1 0 1 0 0 ...
```

Figure 3. Structure of the Dataset

Furthermore, the first 10 rows of the dataset is checked to understand the format in which data is stored in the dataset, as displayed in Figure 4:

```
> # Displaying the first 10 observations in your data frame
> head(data,10)
   RowNumber CustomerId  Surname CreditScore Geography Gender Age Tenure   Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited
1          1   15634602 Hargrave         619    France Female  42      2      0.00             1         1              1       101348.88      1
2          2   15647311     Hill         608     Spain Female  41      1  83807.86             1         0              1       112542.58      0
3          3   15619304     Onio         502    France Female  42      8 159660.80             3         1              0       113931.57      1
4          4   15701354     Boni         699    France Female  39      1      0.00             2         0              0        93826.63      0
5          5   15737888 Mitchell         850     Spain Female  43      2 125510.82             1         1              1        79084.10      0
6          6   15574012      Chu         645     Spain   Male  44      8 113755.78             2         1              0       149756.71      1
7          7   15592531 Bartlett         822    France   Male  50      7      0.00             2         1              1        10062.80      0
8          8   15656148   Obinna         376   Germany Female  29      4 115046.74             4         1              0       119346.88      1
9          9   15792365       He         501    France   Male  44      4 142051.07             2         0              1        74940.50      0
10        10   15592389       H?         684    France   Male  27      2 134603.88             1         1              1        71725.73      0
```

Figure 4. Displaying the First 10 Observations

3.2 Descriptive Statistics

As part of presenting a discussion about the descriptive statistics, summary statistics of the dataset is calculated using R's summary() function and the result is displayed in Figure 5 below:

```
> # Calculate summary statistics
> summary(data)
   RowNumber       CustomerId         Surname           CreditScore     Geography            Gender               Age        
 Min.   :    1   Min.   :15565701   Length:10000       Min.   :350.0   Length:10000       Length:10000       Min.   :18.00  
 1st Qu.: 2501   1st Qu.:15628528   Class :character   1st Qu.:584.0   Class :character   Class :character   1st Qu.:32.00  
 Median : 5000   Median :15690738   Mode  :character   Median :652.0   Mode  :character   Mode  :character   Median :37.00  
 Mean   : 5000   Mean   :15690941                      Mean   :650.5                                         Mean   :38.92  
 3rd Qu.: 7500   3rd Qu.:15753234                      3rd Qu.:718.0                                         3rd Qu.:44.00  
 Max.   :10000   Max.   :15815690                      Max.   :850.0                                         Max.   :92.00  
     Tenure          Balance       NumOfProducts    HasCrCard      IsActiveMember   EstimatedSalary         Exited      
 Min.   : 0.000   Min.   :     0   Min.   :1.00   Min.   :0.0000   Min.   :0.0000   Min.   :    11.58   Min.   :0.0000  
 1st Qu.: 3.000   1st Qu.:     0   1st Qu.:1.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 51002.11   1st Qu.:0.0000  
 Median : 5.000   Median : 97199   Median :1.00   Median :1.0000   Median :1.0000   Median :100193.91   Median :0.0000  
 Mean   : 5.013   Mean   : 76486   Mean   :1.53   Mean   :0.7055   Mean   :0.5151   Mean   :100090.24   Mean   :0.2037  
 3rd Qu.: 7.000   3rd Qu.:127644   3rd Qu.:2.00   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:149388.25   3rd Qu.:0.0000  
 Max.   :10.000   Max.   :250898   Max.   :4.00   Max.   :1.0000   Max.   :1.0000   Max.   :199992.48   Max.   :1.0000  
```

Figure 5. Summary Statistics

Summary statistic measures, such as the mean, median, minimum, maximum, and quantile information are meaningful for the quantitative variables. To understand the distribution of the categorical variables better, we created frequency tables for Geography, Gender, HasCrCard, IsActiveMember, and Exited variables. The frequency tables are displayed in Figure 6:

```
> # Create frequency tables for categorical variables
> table(data$Geography)

 France Germany   Spain
   5014    2509    2477
> table(data$Gender)

Female   Male
  4543   5457
> table(data$HasCrCard)

   0    1
2945 7055
> table(data$IsActiveMember)

   0    1
4849 5151
> table(data$Exited)

   0    1
7963 2037
```

Figure 6. Frequency Tables for Categorical Variables

Furthermore, a correlation matrix is created for numerical variables to display the correlation coefficients between each variable pair. It is concluded from the correlation matrix that the variable pairs have a strong correlation if the absolute value of the coefficient is relatively high, close to 1. On the other hand, variable pairs have a weak correlation if the absolute value of the coefficient is relatively low, close to 0. The correlation matrix we generated for the numerical variables in our dataset is shown in Figure 7:

```
> # Calculate correlation matrix for numerical variables
> cor(data %>% select(CreditScore, Age, Tenure, Balance, NumOfProducts, EstimatedSalary))
                  CreditScore          Age        Tenure      Balance NumOfProducts EstimatedSalary
CreditScore      1.0000000000 -0.003964906  0.0008419418  0.006268382    0.01223788    -0.001384293
Age             -0.0039649055  1.000000000 -0.0099968256  0.028308368   -0.03068009    -0.007201042
Tenure           0.0008419418 -0.009996826  1.0000000000 -0.012253926    0.01344376     0.007783825
Balance          0.0062683816  0.028308368 -0.0122539262  1.000000000   -0.30417974     0.012797496
NumOfProducts    0.0122378793 -0.030680088  0.0134437555 -0.304179738    1.00000000     0.014204195
EstimatedSalary -0.0013842929 -0.007201042  0.0077838255  0.012797496    0.01420420     1.000000000
```

Figure 7. Correlation Matrix for Numerical Variables

The correlation matrix depicted in Figure 7 demonstrates that the numerical variables are not significantly correlated to each other, as evidenced by the small magnitudes of the correlation coefficients between any pair of variables. This result signals that all features are independent.

Lastly, the histogram for credit score is drawn to get a sense of the overall distribution of the credit scores among the customers. We observe a left skewed histogram, where the mean is slightly less than the median (650.5 < 652.0). Based on this histogram, it can be concluded that the

customers with low credit scores are a minority, which indicates that the bank has a financially healthy customer base; customers with high credit scores are greater in number than those with low scores supports this thesis.

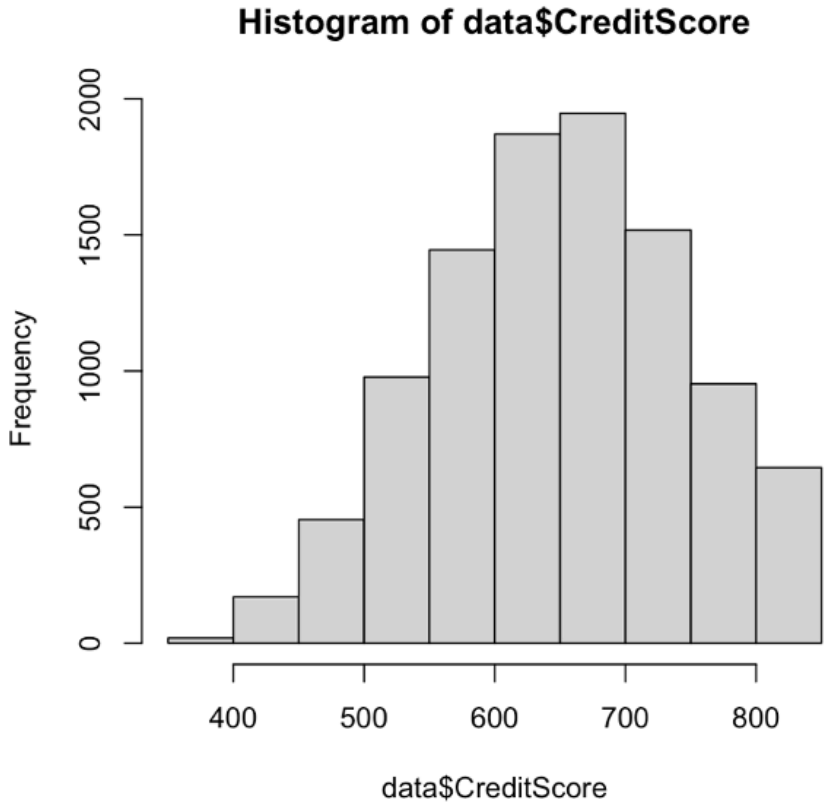

Figure 8. Histogram of Credit Score of Customers

3.3 Data Visualization

In this section, we included pie charts, bar graphs, and box plots to visualize the data. Pie charts are effective for representing categorical data or data that can be divided into distinct categories. In Figure 9, the pie chart that displays the distributions of churned and retained customers is shown:

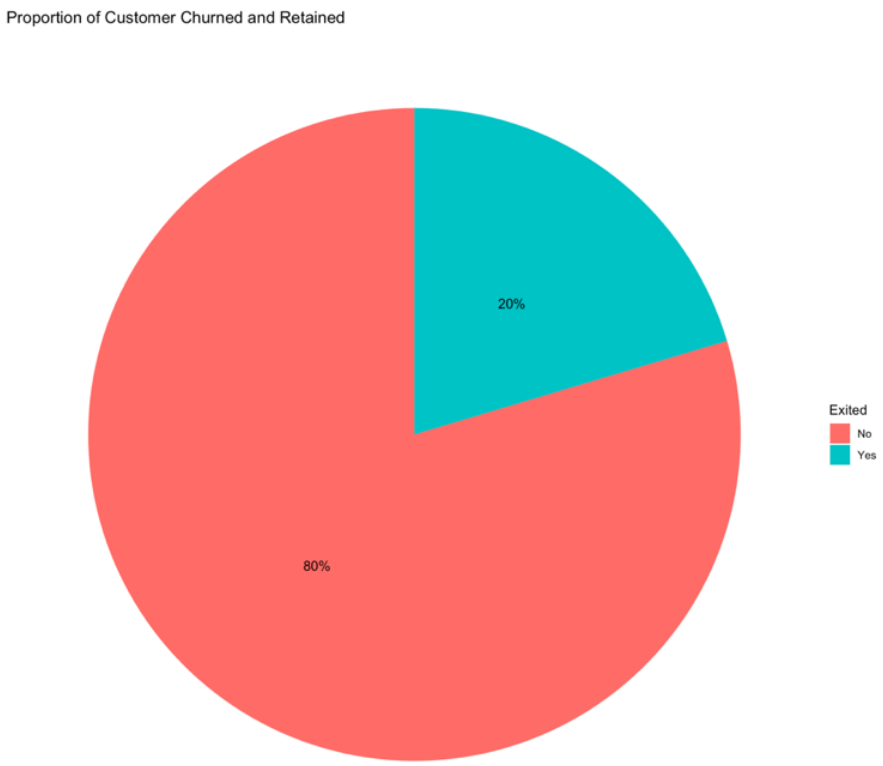

Figure 9. Pie Chart Showing the Proportions of Customers Exited

As shown in Figure 9, the portion of customers who withdrew from the bank's services is 20% and the rest of the pie chart constitutes the remaining customers. Since what is of interest to the bank is to correctly identify and keep the churned customers, we aim to develop predictive models that classify this 20% with high accuracy.

Bar charts can be used to visualize the distribution of categorical variables, showing their frequency in the dataset. In Figure 10, the bar charts that display the frequency distribution for each categorical variable, with respect to status relation, are shown:

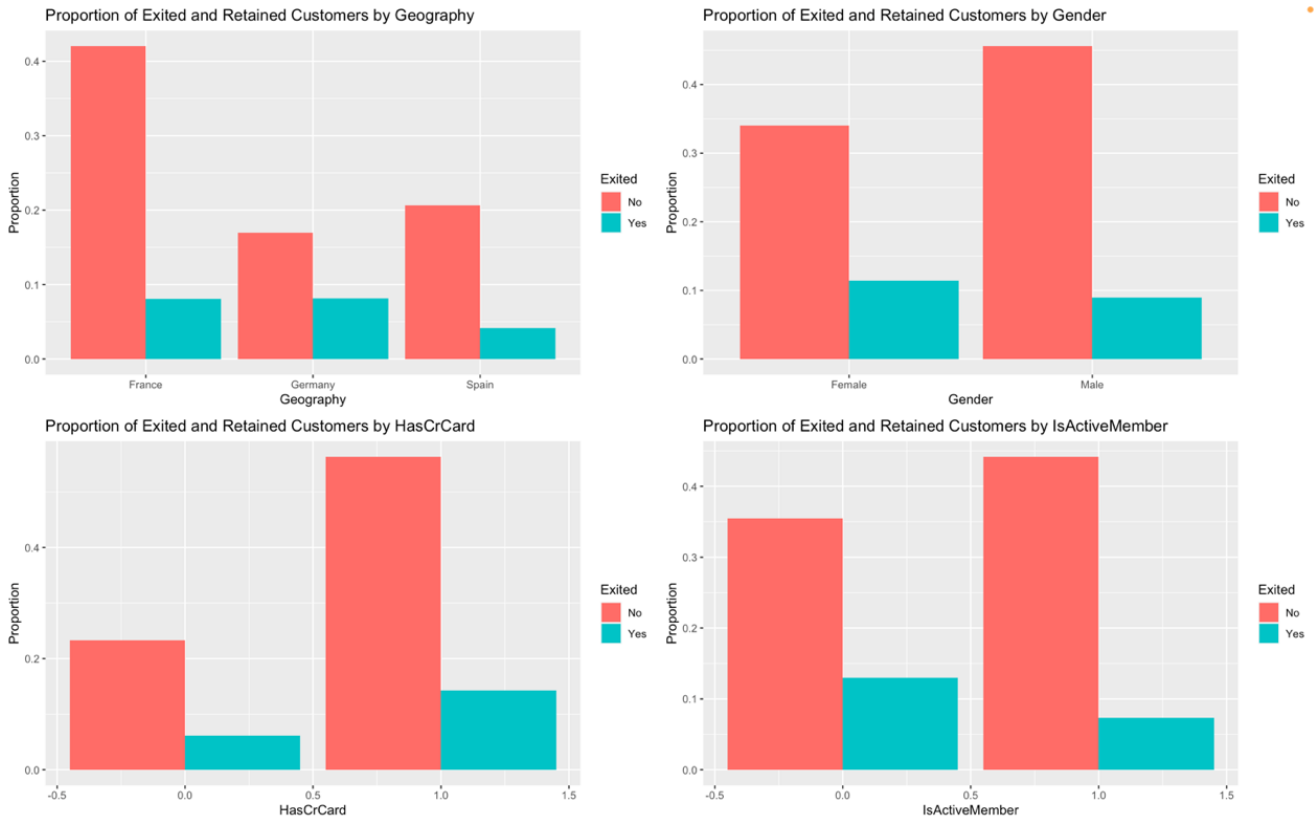

Figure 10. Bar Charts Displaying the Status Relation with Categorical Variables

In Figure 10 above, churning status is examined with respect to geography, gender, card status, and member activity. From the first chart, we observe that majority of the customers are French, but this does not necessarily mean that they have the most churned customers as Germany has as many churned customers as France; this could be caused by the lack of customer service resources allocated for Germany. The second chart illustrates that the number of female customers that show churn behavior is greater than male customers. The third chart displays that majority of churned customers are those who happen to have a credit card, which may suggest that dedicated/loyal customers are lost to some degree. Finally, the last chart explores the churning status and member activity relation and shows that majority of the churned customers are inactive.

To illustrate how continuous variables are distributed based on exit status, we created the following boxplots, as shown in Figure 11; each is provided with axis titles.

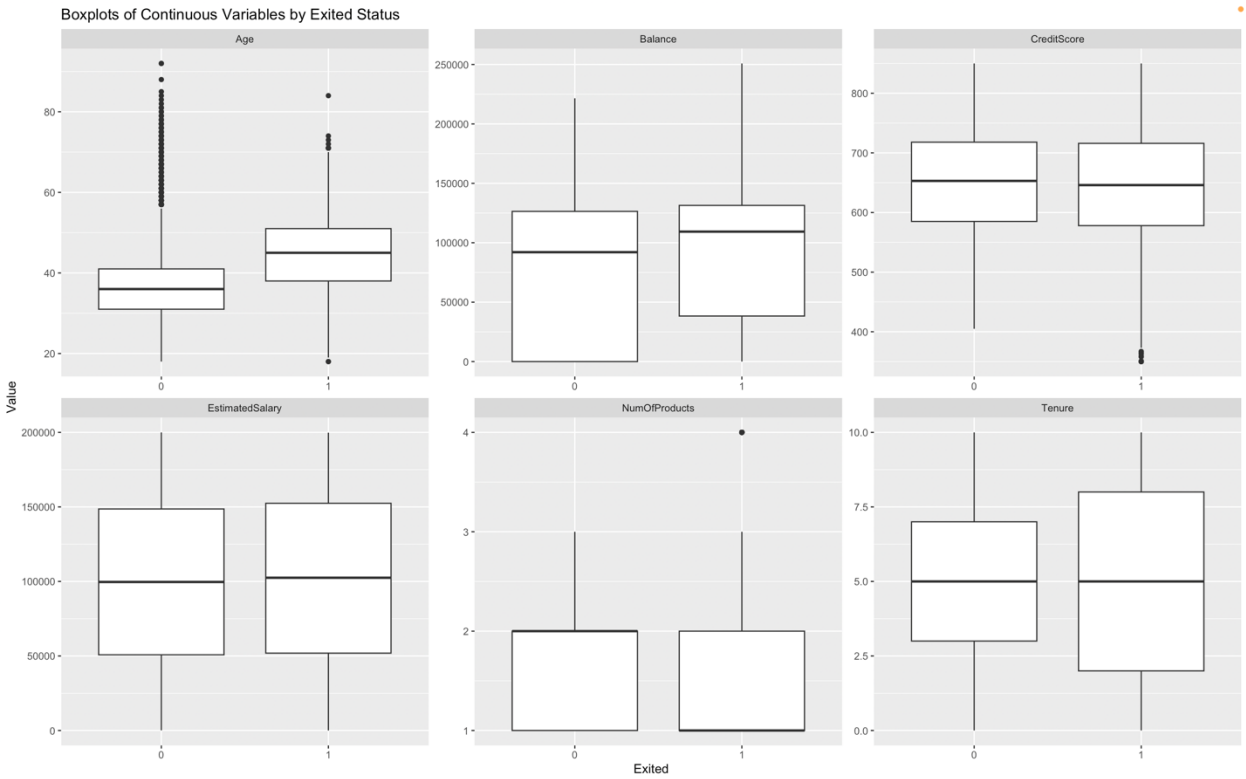

Figure 11. Boxplots of Continuous Variables by Exited Status

The first boxplot in Figure 11 shows that the older age groups are churning more than the younger customers since the outliers are very concentrated on the older age groups. This boxplot might be telling us that the bank's target age group may be inaccuracte or the bank may be overlooking its customers belonging to the older age groups. The second boxplot indicates that majority of churning customers have a negative net balance, which might affect the capital for lending. The credit score, salary and product distributions are not significantly different for the two customer groups according to the 3rd , 4th and 5th boxplots; we can say that their effect on customer churn is negligible. Interestingly, the customers that are on extreme ends of tenure, as shown in the last boxplot, have higher churning rates when compared to average customers.

All things considered, based on the inferences made from Figure 11, the bank must adjust its targeted market and bring up solutions for high churning rates among old age groups. Even though credit score, salary and product distribution do not have a major influence on churning, the bank must keep an eye on these variables to prevent the emergence of new churning patterns. The significant effect of tenure shows the importance of customer satisfaction, developing excellent customer experience will surely be effective on reducing churning rates.

D. DATA PREPARATION AND PARTITIONING

The first step in data preparation is handling missing data. In Section 2.3, we conducted a missing value analysis, which confirmed that the dataset contains no null values. Therefore, there is no need to take further steps to manage missing data.

The second step in data preparation involves factorization of categorical variables. The dataset contains categorical variables such as Surname, Geography, and Gender that are stored as characters in the dataset. To ensure compatibility with the predictive data mining models we aim to develop, Geography and Gender variables are transformed into factors using one-hot encoding. Surname is excluded from factorization because it is unimportant for predicting churn behavior and thus, will not be used in the predictive models. The factorization of Geography and Gender variables is shown in Figure 12:

```
> # Data preparation
> # One-hot encoding of categorical variables
> data <- data %>%
+   mutate(Geography = as.factor(Geography),
+          Gender = as.factor(Gender)) %>%
+   fastDummies::dummy_cols(c("Geography", "Gender"), remove_first_dummy = TRUE)
> # Removing original categorical columns and CustomerId which won't be used for modeling
> data <- data[ , !(names(data) %in% c("Geography", "Gender", "CustomerId", "Surname"))]
> View(data)
```

Figure 12. Factorization of the Categorical Variables

Even though HasCrCard and IsActiveMember variables are also categorical, they were already stored as binary in the dataset. Therefore, there is no need to factorize those variables.

As part of data preparation of the numerical variables, they can be normalized or standardized based on the requirements of the predictive data mining models we will implement. Therefore, data preparation for numerical variables will be conducted and explained if required by the models, in Section F.

After preparing the data, the dataset is partitioned into training and testing sets, as shown in Figure 13. 80% of the data is used for the training set and the remaining 20% is used for the testing dataset:

```
> # Data partitioning
> # Create a partition
> set.seed(123) # for reproducibility
> trainIndex <- createDataPartition(data$Exited, p = .8, list = FALSE)
>
> # Split the data into training and testing sets
> trainData <- data[ trainIndex,]
> testData  <- data[-trainIndex,]
```

Figure 13. Splitting the Dataset

E. FEATURE SELECTION

Before implementing a predictive data mining model, removing irrelevant and redundant features, and identifying key predictors in a dataset is very important. There are several approaches that can be used to conduct feature selection. The three of the most common feature selection approaches are: Filter methods, wrapper methods, and embedded methods. In this section, all three feature selection methods will be introduced and implemented in R.

5.1 Filter Method: Correlation-based Feature Selection

Filter methods evaluate each feature individually for its ability to predict the target variable. Therefore, filter methods are concerned with the intrinsic properties of the features. The advantages of the filter methods over other feature selection methods are their speed and computational efficiency because they do not require training a predictive model during feature selection. They filter out features that do not meet a certain criterion before the model is trained (Gupta).

Among the filter methods, correlation-based feature selection approach will be implemented to identify the key predictors in the dataset. The correlation-based feature selection approach evaluates subsets of features on the basis that the features that are highly correlated with the target variable but uncorrelated with each other are considered to be key predictors (Gupta).

As the first step of implementing correlation-based feature selection, the correlation between the numeric features and the target variable is computed using the training dataset and the correlations are sorted in descending order. After sorting, the top 8 features with the highest correlation are chosen as the predictors. The complete implementation of the correlation-based feature selection using the training dataset is shown in Figure 14:

```
> # Approach 1 - Filter Method - Correlation-based Feature Selection
>
> # Implement the Correlation-based Feature Selection using the training dataset:
> # Calculate the correlation between numeric features and the target variable
> correlations <- cor(trainData[, sapply(trainData, is.numeric)])
> correlations_with_target <- correlations["Exited",]
>
> # Sort the correlations in descending order
> sorted_correlations <- sort(abs(correlations_with_target), decreasing = TRUE)
>
> # Select the top k features with the highest correlation (e.g., k = 8)
> k <- 8
> top_k_features <- names(sorted_correlations)[2:(k+1)]  # Add 1 to exclude the 'Exited' variable itself
>
> cat("Top", k, "features based on correlation:\n", top_k_features, "\n")
Top 8 features based on correlation:
 Age Geography_Germany IsActiveMember Balance Gender_Male Geography_Spain NumOfProducts CreditScore
```

Figure 14. Implementing Correlation-based Feature Selection

The top 8 features selected based on correlation are Age, Geography_Germany, IsActiveMember, Balance, Gender_Male, Geography_Spain, NumOfProducts, and CreditScore.

5.2 Wrapper Method: Forward Feature Selection

Wrapper methods use a predictive machine learning algorithm to score feature subsets by following a greedy search process, evaluating all possible combinations of features to select the optimal feature subset. Due to the consideration of all possible feature subsets and selecting the locally optimal subset at each iteration, wrapper methods usually perform better in feature selection than filter methods (Gupta).

Among the wrapper methods, the forward feature selection approach will be implemented to identify the key predictors in the dataset. Forward feature selection is an iterative approach that starts with an empty set of features and iteratively adds the best feature at each step until the desired number of features are selected (Gupta).

As the first step of implementing forward feature selection, a formula for the linear regression model using all predictor variables except the target variable, i.e. Exited, is created to specify the structure of the model and define the predictors and the target variable. Next, forward feature selection is applied using the training dataset and the optimal number of features are selected using the adjusted R-squared criterion. Finally, the selected features are extracted. The complete implementation of the forward feature selection using the training dataset is shown in Figure 15:

```
> # Approach 2 - Wrapper Method: Forward Feature Selection
>
> # Create a formula for the linear regression model using all predictor variables except Exited:
> predictors <- names(trainData[, !(names(trainData) %in% "Exited")])
> formula <- as.formula(paste("Exited ~", paste(predictors, collapse = " + ")))
>
> # Apply forward feature selection using the training dataset:
> fwd_selection <- regsubsets(formula, data = trainData, method = "forward", nvmax = length(predictors))
> fwd_summary <- summary(fwd_selection)
>
> # Determine the optimal number of features using the adjusted R-squared criterion:
> optimal_n_features <- which.max(fwd_summary$adjr2)
> cat("Optimal number of features:", optimal_n_features, "\n")
Optimal number of features: 9
>
> # Extract the selected features:
> selected_features <- names(coef(fwd_selection, id = optimal_n_features))[-1] # Exclude intercept
> cat("Selected features using forward feature selection:\n", selected_features, "\n")
Selected features using forward feature selection:
 CreditScore Age Tenure Balance NumOfProducts IsActiveMember EstimatedSalary Geography_Germany Gender_Male
```

Figure 15. Implementing Forward Feature Selection

The selected features using forward feature selection are CreditScore, Age, Tenure, Balance, NumOfProducts, IsActiveMember, EstimatedSalary, Geography_Germany, and Gender_Male. In total, 9 features are selected as predictors based on the adjusted R-squared criterion of the implemented linear regression model.

5.3 Embedded Method: LASSO Regression

Embedded methods perform feature selection as part of the model construction process. In other words, the embedded approach for future selection is model-specific. Other advantages of the embedded methods include computational efficiency, simultaneous model training and feature selection. The principal idea behind the embedded methods is to select the most relevant features during the learning process typically by adding a regularization term into the objective function. It should be highlighted that the embedded methods combine the benefits of both filter and wrapper methods (Gupta).

Among the embedded methods, the LASSO regression method will be implemented to identify the key predictors in the dataset. The objective of the LASSO regression method is to obtain the subset of predictors that minimizes the prediction error for the target variable. LASSO performs feature selection by iteratively reducing the regularization parameter (lambda), shrinks some coefficients to zero and iteratively excludes those features with zero coefficients from the model (Gupta).

Different from the previous methods, in LASSO regression, two more preprocessing steps are applied. First, to prepare the data for LASSO regression, categorical variables are converted into dummy variables and then, missing values in the data are imputed, as shown in Figure 16:

```
> # Preprocessing
> # Prepare the data for LASSO regression by converting the categorical variables to dummy variables:
> train_dummies <- model.matrix(~ . - 1, data = trainData[, -1])
>
> # Impute missing values
> train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))
```

Figure 16. Data Preprocessing for LASSO Regression

Following data preprocessing, the predictors and target variable in the dataset are separated from the imputed training dataset. Following this separation, LASSO regression is performed to find the best lambda value. The final LASSO model is fitted using the best lambda value and the selected features are extracted. The complete implementation of the LASSO regression using the training dataset is shown in Figure 17:

```
> # Separate the predictors (x) and response (y) in the dataset
> # The response variable is the 9th column of the data
> x <- as.matrix(train_dummies_imputed[, -9])  # Exclude the 9th column for predictors
> y <- train_dummies_imputed[, 9]  # Response is the 9th column
>
> # Perform lasso regression
> set.seed(123)  # For reproducibility
> cv.lasso <- cv.glmnet(x, y, family="binomial", alpha=1)
>
> # Display the best lambda value
> best_lambda <- cv.lasso$lambda.min
> print(best_lambda)
[1] 0.0006959072
>
> # Fit the final model on the training data using the best lambda
> final_model <- glmnet(x, y, family="binomial", alpha=1, lambda=best_lambda)
>
> # Convert coefficients to matrix
> coef_final_model_matrix <- as.matrix(coef(final_model))
>
> # Get the names of the selected features
> selected_features <- rownames(coef_final_model_matrix)[coef_final_model_matrix[,1] != 0]
>
> # Print the selected features
> print(selected_features[-1])
 [1] "CreditScore"       "Age"               "Tenure"            "Balance"           "NumOfProducts"     "HasCrCard"
 [7] "IsActiveMember"    "EstimatedSalary"   "Geography_Germany" "Geography_Spain"   "Gender_Male"
```

Figure 17. Implementing LASSO Regression

The selected features using LASSO regression are CreditScore, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, Geography_Germany, Geography_Spain, and Gender_Male. In total, 11 features are selected as predictors by the LASSO regression method.

5.4 Comparison of the Results of the Feature Selection Approaches

In each of the feature selection approaches implemented, we did not observe any discrepancies. Furthermore, the features selected by each approach significantly overlap, with the highest number of features being selected by the LASSO regression model.

However, it is important to consider that in the correlation-based feature selection, we manually choose the number of features to be selected by the model (in this study, we chose k as 8). In contrast, the forward feature selection method selects the optimal number of features using the adjusted R-squared criterion. Similarly, the LASSO regression method selects features based on the optimal lambda value, which makes the selection process more data driven. Therefore, we would prefer a more data driven selection process for identifying the key predictors.

In choosing between forward feature selection and LASSO regression, we leaned towards LASSO regression. Being an embedded method, LASSO combines the advantages of both filter and wrapper methods, making it a more robust choice for our feature selection model. Consequently, we decided to use the selected features generated by the LASSO regression model as key predictors in our prediction models. These features include CreditScore, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, Geography_Germany, Geography_Spain, and Gender_Male.

F. PREDICTING CHURN BEHAVIOR: PREDICTIVE MODELS

6.1 Implementing Naive Bayes Classifier

Naive Bayes Classifier has been implemented to predict churn behavior in the banking context using the features selected by the LASSO regression model as predictors. Naïve Bayes Classifier is chosen as one of the predictive models in this case study because it works very well with the classification problems and is proven to be simple, fast, accurate, and reliable. Before implementing the Naive Bayes Classifier, we had to rearrange our training and testing datasets with the selected features from LASSO as shown in Figure 18:

```
> # Selected features by the LASSO method
> selected_features <- c("CreditScore", "Age", "Tenure", "Balance", "NumOfProducts",
+                        "HasCrCard", "IsActiveMember", "EstimatedSalary",
+                        "Geography_Germany", "Geography_Spain", "Gender_Male")
> # Select only the selected features from the training and test data
> trainData_selected <- trainData[, c("Exited", selected_features)]
> testData_selected <- testData[, c("Exited", selected_features)]
```

Figure 18. Rearranging Training and Testing Datasets

The Naive Bayes Classifier operates under two assumptions as outlined in our OPIM407 lecture: all features are equally important, and all features are independent. The first assumption is ensured by our feature selection procedure by selecting only the most important features as predictors. Hence, all features can be considered equally important. Additionally, the second assumption is validated by the correlation matrix of the features, displayed in Figure 7 of this paper. The correlation matrix reports negligible correlation coefficients between any pair of variables, confirming their independence. After confirming the assumptions, Naive Bayes Classifier has been fed with the training dataset as shown in Figures 19, 20, and 21:

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      0       1
0.79575 0.20425
```

Figure 19. Results of Naive Bayes Classifier

```
Conditional probabilities:
   CreditScore
Y       [,1]      [,2]
  0 651.1103  95.47774
  1 644.3960 100.59880

   Age
Y       [,1]      [,2]
  0 37.38281 10.063632
  1 44.83599  9.720971

   Tenure
Y       [,1]     [,2]
  0 5.034087 2.878138
  1 4.870869 2.930482

   Balance
Y       [,1]     [,2]
  0 72947.17 62948.28
  1 91146.16 58383.45

   NumOfProducts
Y       [,1]      [,2]
  0 1.537700 0.5095247
  1 1.468788 0.7960820
```

Figure 20. Results of Naive Bayes Classifier

```
   HasCrCard
Y       [,1]      [,2]
  0 0.7079799 0.4547272
  1 0.6995104 0.4586113

   IsActiveMember
Y       [,1]      [,2]
  0 0.5516808 0.4973610
  1 0.3598531 0.4801041

   EstimatedSalary
Y       [,1]     [,2]
  0  99637.54 57221.57
  1 101494.27 58094.81

   Geography_Germany
Y       [,1]      [,2]
  0 0.2095507 0.4070200
  1 0.3953488 0.4890752

   Geography_Spain
Y       [,1]      [,2]
  0 0.2577757 0.4374442
  1 0.2044064 0.4033906

   Gender_Male
Y       [,1]      [,2]
  0 0.5769714 0.4940787
  1 0.4369645 0.4961625
```

Figure 21. Results of Naive Bayes Classifier

The a-priori probabilities reported by the Naive Bayes Classifier, as displayed in Figure 19, indicate that before taking into account the effect of any predictor variables, the probability of a customer not exiting (Y = 0) is 0.79575, and the probability of a customer exiting (Y = 1) is 0.20425. These probabilities are decided by the percentages of customers that were reported to have exited the bank and not exited the bank in the training dataset. Furthermore, conditional probabilities were computed for each feature in the result of the Naive Bayes Classifier, as seen in the Figures 20 and 21, which are the probabilities of observing certain values of the predictor variables, given that a customer has exited or not exited the bank. The Laplace estimator was unnecessary since all predictor values were represented in the training data.

Next, using R's predict function, the object 'dnb' that is the Naive Bayes Classifier model has been fed with the testing dataset for calculating the probabilities and determining class membership, as shown in Figure 22:

```
> # Calculate probabilities of each class
> pred_prob <- predict(dnb, newdata = testData, type = "raw")
> # Predict class membership
> pred_class <- predict(dnb, newdata = testData)
> # Create a new data frame to see actual and predicted values
> # Along with probbailities of belonong to ontime or delayed class
> ndf <- data.frame(Actual = testData$Exited, Predicted = pred_class,
+                   Probaility = pred_prob)
> ndf
    Actual Predicted Probaility.0 Probaility.1
1        1         0  0.878126424  0.121873576
2        0         0  0.903817840  0.096182160
3        1         0  0.892700213  0.107299787
4        0         0  0.948363206  0.051636794
5        0         0  0.984183871  0.015816129
6        0         0  0.972705085  0.027294915
7        0         0  0.763254543  0.236745457
8        1         1  0.431563611  0.568436389
9        0         0  0.530507499  0.469492501
```

Figure 22. Predicting with Testing Dataset

The classifier computed the probabilities of each class and made predictions regarding each class's membership. These calculations were then merged into a new data frame where predictions were tabulated alongside their actual values and class probabilities, in which class labels were assigned based on the higher probability.

Upon implementing the Naive Bayes Classifier, we evaluated the model's performance using a variety of metrics, including a confusion matrix, accuracy, precision, recall, and F1 scores, and area under ROC curve. Figures 23 and 24 illustrate the evaluation of the Naive Bayes model:

```
> # Confusion Matrix and Statistics
> cm <- confusionMatrix(as.factor(pred_class), as.factor(testData$Exited))
> cm
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1508  254
         1   89  149

               Accuracy : 0.8285
                 95% CI : (0.8113, 0.8448)
    No Information Rate : 0.7985
    P-Value [Acc > NIR] : 0.0003667

                  Kappa : 0.3707

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9443
            Specificity : 0.3697
         Pos Pred Value : 0.8558
         Neg Pred Value : 0.6261
             Prevalence : 0.7985
         Detection Rate : 0.7540
   Detection Prevalence : 0.8810
      Balanced Accuracy : 0.6570

       'Positive' Class : 0
```

Figure 23. Confusion Matrix for Model Evaluation

```
> # Accuracy
> accuracy <- cm$overall['Accuracy']
> accuracy
Accuracy
  0.8285
> # Precision
> precision <- cm$byClass['Pos Pred Value']
> precision
Pos Pred Value
     0.8558456
> # Recall (Sensitivity)
> recall <- cm$byClass['Sensitivity']
> recall
Sensitivity
  0.9442705
> # F1 Score
> f1_score <- 2*precision*recall/(precision + recall)
> f1_score
Pos Pred Value
     0.8978863
> roc_obj <- roc(testData$Exited, pred_prob[, 2])
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc(roc_obj)
Area under the curve: 0.7886
```

Figure 24. Other Performance Metrics

The error metrics depicted in Figures 23 and 24 measure the performance of the Naive Bayes classifier. In Figure 23, the rows of the confusion matrix represent the instances in an actual class (0 being customers who stays and 1 being customers who churn) and the columns represent

the instances in a predicted class. According to the confusion matrix, the model correctly predicted 1508 instances of class ‘0’ (True Negatives) and 149 instances of class ‘1’ (true positives). However, it incorrectly predicted 254 instances as class ‘0’ (False Negatives) when they were indeed class ‘1’, and 89 instances as class ‘1’ (False Positives) when they were indeed class ‘0’. Furthermore, the accuracy of the model been calculated as 82.85% in Figure 23. Precision, recall, and f1 scores of the model have been respectively printed as 85.58%, 94.43% and 89.79%. The AUC score of the model, which is the measure of how well the model distinguishes between classes, suggest a fairly good model with a 0.7886 score.

6.2 Implementing Support Vector Machine (SVM) Model

Support Vector Machine (SVM) is another model that has been implemented to predict customer churn behavior in banking context. SVM is chosen as the second predictive model to be used in this case study because it is a powerful method to classify unstructured data. The focus of the Support Vector Machine is to maximize the margin around the separating hyperplane. As outlined in our OPIM 407 course, Kernel Trick is used to implicitly map the original features of our data into a high-dimensional space where the data becomes linearly separable. In this higher-dimensional feature space, a linear decision boundary (a hyperplane) can be found to separate the classes. For the churn analysis, radial, linear and polynomial kernel functions will be used to create the models. Then, the three models will be compared to choose the best. Before starting to implement SVM models, the target variable, “Exited”, is factorized for classification analysis on SVM as seen in Figure 25:

```
> # Convert 'Exited' column into factor
> trainData_selected$Exited <- as.factor(trainData_selected$Exited)
> testData_selected$Exited <- as.factor(testData_selected$Exited)
```

Figure 25. Factorizing "Exited" Feature

The Support Vector Machine has been implemented by choosing the radial kernel and the cost is fixed as 5 as shown in Figure 26:

```
> # Train SVM model with RADİAL kernel
> svmfit <- svm(Exited ~ ., data = trainData_selected, kernel = "radial", cost = 5, scale = FALSE)
>
> # Predict churn on test data
> predicted_churn <- predict(svmfit, testData_selected)
```

Figure 26. Implementing Radial SVM

For the evaluation of the radial SVM model, a confusion matrix is constructed. Figure 27 displays that the SVM model with radial kernel function resulted in an accuracy of 79.85%, which is a considerably high accuracy percentage. However, the model failed to predict any instance of the positive class (Exited = 1). This failure is evident from the precision, recall, and F1 scores which could not be calculated due to division by zero and reported to have NaN values, as shown in Figure 28. This means that although the overall accuracy is high, the model fails to correctly predict customer churn behavior, which is the primary interest in our case.

```
> confusionMatrix(table(predicted_churn, testData_selected$Exited))
Confusion Matrix and Statistics


predicted_churn    0    1
              0 1597  403
              1    0    0

               Accuracy : 0.7985
                 95% CI : (0.7802, 0.8159)
    No Information Rate : 0.7985
    P-Value [Acc > NIR] : 0.5133

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.7985
         Neg Pred Value :    NaN
             Prevalence : 0.7985
         Detection Rate : 0.7985
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

Figure 27. Confusion Matrix of Radial SVM

```
Radial Kernel
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.7985
> cat("Precision:", precision, "\n")
Precision: 0
> cat("Recall:", recall, "\n")
Recall: NaN
> cat("F1 Score:", f1_score, "\n")
F1 Score: NaN
> cat("AUC:", auc, "\n")
AUC: 0.5
```

Figure 28. Performance Evaluation of Radial SVM

Following the implementation of the SVM with radial kernel function, an SVM model with a linear kernel function is implemented, as shown in Figure 29. Linear kernel is the simplest kernel function we learned in our OPIM407 course.

For the evaluation of the linear SVM model, a confusion matrix is constructed. The SVM with linear kernel showed an overall accuracy of 76.8%. The linear SVM model could predict both classes, with Precision, Recall, and F1 scores reported as 0.223, 0.373, and 0.279 respectively for the positive class (class Exited = 1), as illustrated in Figure 31. The AUC for this model is calculated to be 0.564.

```
> # Train SVM model with LINEAR KERNAL
> svmfit_linear <- svm(Exited ~ ., data = trainData_selected, kernel = "linear", cost = 5, scale = FALSE)

WARNING: reaching max number of iterations
> # Predict churn on test data
> predicted_churn_linear <- predict(svmfit_linear, testData_selected)
```

Figure 29. Implementing Linear SVM

```
> # Evaluate model
> confusionMatrix(table(predicted_churn_linear, testData_selected$Exited))
Confusion Matrix and Statistics


predicted_churn_linear    0    1
                     0 1446  313
                     1  151   90

               Accuracy : 0.768
                 95% CI : (0.7489, 0.7863)
    No Information Rate : 0.7985
    P-Value [Acc > NIR] : 0.9996

                  Kappa : 0.1515

 Mcnemar's Test P-Value : 7.765e-14

            Sensitivity : 0.9054
            Specificity : 0.2233
         Pos Pred Value : 0.8221
         Neg Pred Value : 0.3734
             Prevalence : 0.7985
         Detection Rate : 0.7230
   Detection Prevalence : 0.8795
      Balanced Accuracy : 0.5644

       'Positive' Class : 0
```

Figure 30. Confusion Matrix of Linear SVM

```
Linear Kernel
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.768
> cat("Precision:", precision, "\n")
Precision: 0.2233251
> cat("Recall:", recall, "\n")
Recall: 0.373444
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.2795031
> cat("AUC:", auc, "\n")
AUC: 0.5643864
```

Figure 31. Performance Evaluation of Linear SVM

The Polynomial kernel, as the name suggests, generates a non-linear decision boundary through higher-degree polynomials. It introduces more complexity into the model than a linear kernel and less than an RBF kernel. The implementation of a polynomial kernel SVM model also provided a reasonable accuracy of 77.25%. It produced precision, recall, and F1 scores of 0.089, 0.290, and 0.137 respectively for the positive class (class Exited = 1). The AUC score was 0.517. The Figures 32, 33, and 34 respectively show the steps of the polynomial kernel implementation of SVM and the performance metrics of the resulting model:

```
> # Train SVM model with POLYNOMIAL kernel
> svmfit_poly <- svm(Exited ~ ., data = trainData_selected, kernel = "polynomial", cost = 5, scale = FALSE)

WARNING: reaching max number of iterations
>
> # Predict churn on test data
> predicted_churn_poly <- predict(svmfit_poly, testData_selected)
```

Figure 32. Implementing Polynomial SVM

```
> # Evaluate model
> confusionMatrix(table(predicted_churn_poly, testData_selected$Exited))
Confusion Matrix and Statistics


predicted_churn_poly    0    1
                   0 1509  367
                   1   88   36

               Accuracy : 0.7725
                 95% CI : (0.7535, 0.7907)
    No Information Rate : 0.7985
    P-Value [Acc > NIR] : 0.9981

                  Kappa : 0.0462

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.94490
            Specificity : 0.08933
         Pos Pred Value : 0.80437
         Neg Pred Value : 0.29032
             Prevalence : 0.79850
         Detection Rate : 0.75450
   Detection Prevalence : 0.93800
      Balanced Accuracy : 0.51711

       'Positive' Class : 0
```

Figure 33. Confusion Matrix of Polynomial SVM

```
Polynomial Kernel
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.7725
> cat("Precision:", precision, "\n")
Precision: 0.08933002
> cat("Recall:", recall, "\n")
Recall: 0.2903226
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.1366224
> cat("AUC:", auc, "\n")
AUC: 0.5171134
```

Figure 34. Performance Evaluation of Polynomial SVM

In the context of this problem, we are interested in predicting customer churn (class Exited=1). The radial kernel, despite its high accuracy, failed to predict any instance of customer churn. Between the linear and polynomial kernel, the linear kernel provided a better trade-off between precision and recall, as shown by the F1 score (0.279 vs 0.137), and also had a higher AUC value (0.564 vs 0.517).

Therefore, based on these results, the SVM model with the linear kernel is recommended as it performs better in predicting customer churn, which is the primary interest in our case study.

6.3 Model Selection

Among the implemented models, the Naive Bayes Classifier and the SVM with Linear Kernel were found to be best performing at predicting customer churn. However, the Naive Bayes Classifier outperformed the Linear SVM model in terms of all performance metrics, including accuracy, precision, recall, F1 score, and AUC score. Therefore, Naive Bayes Classifier is recommended to be implemented by the banks for predicting customer churn.

G. ENSEMBLE METHODS

As discussed in our OPIM407 course, ensemble methods are commonly used for combining multiple models, creating supermodels with enhanced predictive accuracy. In our case study, three approaches will be applied to construct ensembles: Bagging, Boosting, and Random Forests.

7.1 Bagging

Bagging generates multiple random data samples by sampling with replacement from the original dataset. This sampling technique used by the bagging approach is called bootstrap sampling. Each sample maintains the same size as the original dataset. Each of these samples is

then used to train a separate model, and equal weights are assigned to the resulting classification or prediction models to compute the ensemble's prediction.

The bagging approach is implemented in R using the training and testing datasets that contain the selected features from the LASSO regression model. The implementation of the bagging approach is illustrated in Figure 36:

```
> trainData_selected <- trainData[, c("Exited", selected_features)]
> testData_selected <- testData[, c("Exited", selected_features)]
>
> # 1. Bagging Model
> library(ipred)
>
> # Create a bagging model
> bagging_model <- ipred::bagging(Exited ~ ., data = trainData_selected, nbagg = 100)
>
> # Make predictions
> bagging_predictions <- predict(bagging_model, newdata = testData_selected)
```

Figure 36. Implementation of Bagging Method

Bagging method aims to reduce the variance of the predictions by combining multiple models trained and tested on the samples acquired by bootstrap sampling technique.

7.2 Boosting

Boosting is a sequential ensemble method as it iteratively adapts how it samples original data by focusing on the areas where it makes errors. In other words, boosting creates a series of models such that the later models are stronger than the initial models in terms of predictive accuracy.

The boosting approach is implemented in R using the training and testing datasets that contain the selected features from the LASSO regression model. The implementation of the boosting approach is illustrated in Figure 37:

```
> # 2. Boosting
> library(xgboost)
>
> # Convert data to xgb.DMatrix object
> train_matrix <- xgb.DMatrix(data = as.matrix(trainData[,-1]), label = trainData_selected$Exited)
> test_matrix <- xgb.DMatrix(data = as.matrix(testData[,-1]), label = testData_selected$Exited)
>
> # Set parameters
> params <- list(booster = "gbtree", objective = "binary:logistic", eta=0.3, gamma=0, max_depth=6, min_child_weight=1, subsample=1, colsample
_bytree=1)
>
> # Train the model
> boosting_model <- xgb.train(params = params, data = train_matrix, nrounds = 50)
>
> # Make predictions
> boosting_predictions <- predict(boosting_model, newdata = test_matrix)
```

Figure 37. Implementation of Boosting Method

Unlike bagging, boosting does not involve bootstrap sampling. Instead, each new model is improved based on the performance of the previously built models. At each iteration, the current model is forced to pay more attention to the misclassified records.

7.3 Random Forests

Random forest ensemble method is a variant of the bagging method that generates multiple classification or regression trees based on different random samples. Random forest technique differs from the bagging approach by the restriction that each individual tree is allowed to use only a limited number of randomly selected features (input variables).

The random forest approach is implemented in R using the training and testing datasets that contain the selected features from the LASSO regression model. The implementation of the random forest approach is illustrated in Figure 38:

```
> # 3. Random Forest
> library(randomForest)
>
> # Create a random forest model
> rf_model <- randomForest(Exited ~ ., data=trainData_selected, ntree=100)
Warning message:
In randomForest.default(m, y, ...) :
  The response has five or fewer unique values.  Are you sure you want to do regression?
>
> # Make predictions
> rf_predictions <- predict(rf_model, newdata=testData_selected)
```

Figure 38. Implementation of Random Forest Method

7.4 Evaluating the Performance of the Implemented Ensemble Models

The predictive performance of the ensemble models is assessed through creating confusion matrices and extracting accuracy, precision, recall, and F1 scores.

Before constructing confusion matrices based on the predictions made by the ensemble methods, the outputted probabilities are converted to class labels on a threshold. The reason for this conversion is that ensemble methods output probabilities that an observation belongs to each possible class, rather than explicit class labels. The threshold is chosen as 0.5 since the churn problem is a binary classification problem. probabilities less than 0.5 are assigned to the negative class (class Exited = 0), and probabilities 0.5 and higher are assigned to the positive class (class Exited = 1). The R code displaying this preprocessing step is shown in Figure 39:

```
> # Converting the probabilities outputted by the ensemble methods to class labels based on a threshold before evaluation
> # The threshold is chosen as 0.5
> bagging_predictions <- ifelse(bagging_predictions > 0.5, 1, 0)
> bagging_predictions <- as.factor(bagging_predictions)
>
> boosting_predictions <- ifelse(boosting_predictions > 0.5, 1, 0)
> boosting_predictions <- as.factor(boosting_predictions)
>
> rf_predictions <- ifelse(rf_predictions > 0.5, 1, 0)
> rf_predictions <- as.factor(rf_predictions)
```

Figure 39. Converting Probabilities into Class Labels

After converting the outputted probabilities into class labels, confusion matrices for each ensemble model are constructed and accuracy, precision, recall, and F1 scores are extracted from the matrices. The results of the performance evaluation of the bagging, boosting, and random forest ensemble methods are respectively illustrated in Figures 40, 41, and 42:

```
> # Bagging model evaluation
> bagging_cm <- confusionMatrix(table(bagging_predictions, testData_selected$Exited))
> print(paste("Bagging Model Accuracy: ", bagging_accuracy))
[1] "Bagging Model Accuracy:  0.8565"
> print(paste("Bagging Model Precision: ", bagging_precision))
[1] "Bagging Model Precision:  0.872159090909091"
> print(paste("Bagging Model Recall: ", bagging_recall))
[1] "Bagging Model Recall:  0.961177207263619"
> print(paste("Bagging Model F1 Score: ", bagging_F1))
[1] "Bagging Model F1 Score:  0.914507000297885"
```

Figure 40. Evaluating the Performance of the Bagging Model

```
> # Boosting model evaluation
> boosting_cm <- confusionMatrix(table(boosting_predictions, testData_selected$Exited))
> print(paste("Boosting Model Accuracy: ", boosting_accuracy))
[1] "Boosting Model Accuracy:  1"
> print(paste("Boosting Model Precision: ", boosting_precision))
[1] "Boosting Model Precision:  1"
> print(paste("Boosting Model Recall: ", boosting_recall))
[1] "Boosting Model Recall:  1"
> print(paste("Boosting Model F1 Score: ", boosting_F1))
[1] "Boosting Model F1 Score:  1"
```

Figure 41. Evaluating the Performance of the Boosting Model

```
> # Random Forest model evaluation
> rf_cm <- confusionMatrix(table(rf_predictions, testData_selected$Exited))
> print(paste("Random Forest Model Accuracy: ", rf_accuracy))
[1] "Random Forest Model Accuracy:  0.8675"
> print(paste("Random Forest Model Precision: ", rf_precision))
[1] "Random Forest Model Precision:  0.879703534777651"
> print(paste("Random Forest Model Recall: ", rf_recall))
[1] "Random Forest Model Recall:  0.966186599874765"
> print(paste("Random Forest Model F1 Score: ", rf_F1))
[1] "Random Forest Model F1 Score:  0.920919128618323"
```

Figure 42. Evaluating the Performance of the Random Forest Model

The performance metrics of the implemented ensemble methods, as displayed in Figures 40, 41, and 42, indicate that the boosting method, with the highest value across all metrics (value 1), surpasses the bagging and random forest models in terms of its predictive power. Consequently, we recommend using the boosting ensemble method to construct supermodels with improved predictive accuracy for the purpose of predicting bank customer churn.

H. CONCLUSION & SUMMARY OF FINDINGS

In this study, our primary objective was to build a predictive model to help the bank adjust their operations to prevent customer churning patterns; therefore, we have identified some serious problems regarding customer churn behavior. The most concerning problem is that the bank is losing loyal/dedicated customers, the churning tendency of customers with credit cards and a high tenure is stronger when compared to the ones that are without. Another serious problem is that the older customers are churning more, which signals that the needs of the older customers may be neglected by the bank. Finally, we have observed that the female customers' churning rate is higher than male customers', meaning that there may be a lack of service quality from the perspective of female customers. All these issues concern the bank's long-term customer base and service quality; it is evident that there are specific customer groups that are not content with the bank's services as it is observed that the committed customers are lost more than the non-committed. The bank must enhance the quality of its services and customer relations, otherwise, new churning patterns may emerge, and the bank's customer base may take a significant damage. Given the severity of the churn problem, the bank should consider implementing a churn prediction model. This will enable the bank to identify high-risk customers and take proactive measures.

The modelling stage was pivotal in our study, given its direct influence on the findings. Thus, to ensure the utmost accuracy in our churn predictions, each step of the modelling process is carefully implemented and thoroughly explained in this report. Feature selection, an important step for precise predictive modelling, was conducted using LASSO regression modelling and resulted in the selection of the following variables: CreditScore, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, Geography_Germany, Geography_Spain, and Gender_Male.

Following feature selection, we developed predictive models for churn analysis using both the Naïve Bayes Classifier and Support Vector Machines (SVM) models, the latter with radial,

linear, and polynomial kernel functions. Notably, the Naïve Bayes Classifier outperformed all three SVM models in nearly all aspects, leading to its recommendation in our study. We then implemented the ensemble models and observed that boosting model is superior than the bagging and random forest models in terms of predictive accuracy, outperforming them in across all metrics.

To reach a conclusion on our modelling, we can advocate that the predictive data mining models have been built with high precision due to variety and complexity of the techniques we have implemented. Our findings indicate that a Naive Bayes Classifier combined with a boosting ensemble method offers the most precise and accurate predictive model for preventing customer churn in banks. Banks are encouraged to implement this model if a trend of churn become apparent among their customers.

I. REFERENCES

Gupta, Aman. "Feature Selection Techniques in Machine Learning (Updated 2023)." *Analytics Vidhya*, Analytics Vidhya, 26 Apr. 2023, https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/.

J. APPENDIX

```
library(fastDummies)
library(ggplot2)
library(dplyr)
library(gridExtra)
library(tidyverse)
library(caret)
library(leaps)
library(glmnet)
library(rpart)
library(randomForest)
library(pROC)
library(e1071)
library(neuralnet)
library(kernlab)
library(pROC)

data <- read.csv("/Users/selinceydeli/Desktop/OPIM407 term project/Churn_Modelling.csv")
View(data)

# Get the dimensions of the dataset (rows and columns)
dimensions <- dim(data)
dimensions

# Displaying the summary for each variable in the dataset
summary(data)
table(data$Geography)

# Are there any null values in the dataset?
sapply(data, function(x) sum(is.na(x))) #There aren't any NA values
```

```
# Displaying the types of the variables (i.e columns of data frame)
str(data)

# Displaying the first 10 observations in our data frame
head(data,10)

# Calculate summary statistics
summary(data)

# Create frequency tables for categorical variables
table(data$Geography)
table(data$Gender)
table(data$HasCrCard)
table(data$IsActiveMember)
table(data$Exited)

# Calculate correlation matrix for numerical variables
cor(data %>% select(CreditScore, Age, Tenure, Balance, NumOfProducts, EstimatedSalary))

# Create histograms for numerical variables
ggplot(data, aes(x=Age)) + geom_histogram(binwidth=5) + labs(title="Histogram of Age")
ggplot(data, aes(x=CreditScore)) + geom_histogram(binwidth=5) + labs(title="Histogram of CreditScore")

# Chart-1: Pie Chart
# Group by 'Exited' status, count the numbers, and calculate proportions
data_exited <- data %>%
  group_by(Exited) %>%
  summarise(n = n()) %>%
  mutate(prop = n / sum(n))

# Plot the proportions as a pie chart
ggplot(data_exited, aes(x="", y=prop, fill=factor(Exited))) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +
  theme_void() +
  scale_fill_discrete(name="Exited", labels=c("No", "Yes")) +
  geom_text(aes(label = paste0(round(prop*100), "%")), position = position_stack(vjust = 0.5)) +
  ggtitle("Proportion of Customer Churned and Retained")

# Chart-2: 4 Bar Charts in a Single Screen
# Create a function to generate a plot for a specific variable
plot_churn <- function(var) {
  data_grouped <- data %>%
    group_by_at(vars(var, "Exited")) %>%
    summarise(n = n(), .groups = "drop") %>%
    mutate(prop = n / sum(n))

  p <- ggplot(data_grouped, aes_string(x=var, y="prop", fill="factor(Exited)")) +
    geom_bar(stat="identity", position="dodge") +
    scale_fill_discrete(name="Exited", labels=c("No", "Yes")) +
    labs(x=var, y="Proportion", title=paste("Proportion of Exited and Retained Customers by", var))

  return(p)
}

# Generate the plots
plot_geo <- plot_churn("Geography")
plot_gender <- plot_churn("Gender")
plot_card <- plot_churn("HasCrCard")
plot_active <- plot_churn("IsActiveMember")

# Arrange the plots in a 2x2 grid
grid.arrange(plot_geo, plot_gender, plot_card, plot_active, ncol=2, nrow=2)

# Chart-3: Box Plots
# List of continuous variables
cont_vars <- c("CreditScore", "Tenure", "Age", "NumOfProducts", "Balance", "EstimatedSalary")

# Convert data to long format
```

```
data_long <- data %>%
  pivot_longer(cols = all_of(cont_vars),
               names_to = "Variable",
               values_to = "Value")

# Create the boxplots
p <- ggplot(data_long, aes_string(x="factor(Exited)", y="Value")) +
  geom_boxplot() +
  facet_wrap(~ Variable, scales = "free") +
  labs(x="Exited", y="Value", title="Boxplots of Continuous Variables by Exited Status")

print(p)

# Data preparation
# One-hot encoding of categorical variables
data <- data %>%
  mutate(Geography = as.factor(Geography),
         Gender = as.factor(Gender)) %>%
  fastDummies::dummy_cols(c("Geography", "Gender"), remove_first_dummy = TRUE)

# Removing original categorical columns and CustomerId which won't be used for modeling
data <- data[ , !(names(data) %in% c("Geography", "Gender", "CustomerId", "Surname"))]

# Data partitioning
# Create a partition
set.seed(123) # for reproducibility
trainIndex <- createDataPartition(data$Exited, p = .8, list = FALSE)

# Split the data into training and testing sets
trainData <- data[ trainIndex,]
testData  <- data[-trainIndex,]



# Feature Selection Methods

# Approach 1 - Filter Method - Correlation-based Feature Selection

# Implement the Correlation-based Feature Selection using the training dataset:
# Calculate the correlation between numeric features and the target variable
correlations <- cor(trainData[, sapply(trainData, is.numeric)])
correlations_with_target <- correlations["Exited",]

# Sort the correlations in descending order
sorted_correlations <- sort(abs(correlations_with_target), decreasing = TRUE)

# Select the top k features with the highest correlation (e.g., k = 8)
k <- 8
top_k_features <- names(sorted_correlations)[2:(k+1)]  # Add 1 to exclude the 'Exited' variable itself

cat("Top", k, "features based on correlation:\n", top_k_features, "\n")

# Approach 2 - Wrapper Method: Forward Feature Selection

# Create a formula for the linear regression model using all predictor variables except Exited:
predictors <- names(trainData[, !(names(trainData) %in% "Exited")])
formula <- as.formula(paste("Exited ~", paste(predictors, collapse = " + ")))

# Apply forward feature selection using the training dataset:
fwd_selection <- regsubsets(formula, data = trainData, method = "forward", nvmax = length(predictors))
fwd_summary <- summary(fwd_selection)

# Determine the optimal number of features using the adjusted R-squared criterion:
optimal_n_features <- which.max(fwd_summary$adjr2)
cat("Optimal number of features:", optimal_n_features, "\n")

# Extract the selected features:
selected_features <- names(coef(fwd_selection, id = optimal_n_features))[-1] # Exclude intercept
cat("Selected features using forward feature selection:\n", selected_features, "\n")
```

```
# Approach 3 - Embedded Method: LASSO Regression

# Preprocessing
# Prepare the data for LASSO regression by converting the categorical variables to dummy variables:
train_dummies <- model.matrix(~ . - 1, data = trainData[, -1])

# Impute missing values
train_dummies_imputed <- apply(train_dummies, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))

# Check for missing values in the dataset
if (any(is.na(train_dummies_imputed))) {
  cat("There are still missing values in the dataset.\n")
} else {
  cat("No missing values found in the dataset.\n")
}

# Separate the predictors (x) and response (y) in the dataset
# The response variable is the 9th column of the data
x <- as.matrix(train_dummies_imputed[, -9])  # Exclude the 9th column for predictors
y <- train_dummies_imputed[, 9]  # Response is the 9th column

# Perform lasso regression
set.seed(123)  # For reproducibility
cv.lasso <- cv.glmnet(x, y, family="binomial", alpha=1)

# Display the best lambda value
best_lambda <- cv.lasso$lambda.min
print(best_lambda)

# Fit the final model on the training data using the best lambda
final_model <- glmnet(x, y, family="binomial", alpha=1, lambda=best_lambda)

# Convert coefficients to matrix
coef_final_model_matrix <- as.matrix(coef(final_model))

# Get the names of the selected features
selected_features <- rownames(coef_final_model_matrix)[coef_final_model_matrix[,1] != 0]

# Print the selected features
print(selected_features[-1])



# Naive Bayes Classifier Implementation

# Selected features by the LASSO method
selected_features <- c("CreditScore", "Age", "Tenure", "Balance", "NumOfProducts",
                       "HasCrCard", "IsActiveMember", "EstimatedSalary",
                       "Geography_Germany", "Geography_Spain", "Gender_Male")

# Select only the selected features from the training and test data
trainData_selected <- trainData[, c("Exited", selected_features)]
testData_selected <- testData[, c("Exited", selected_features)]
View(trainData_selected)

dnb <- naiveBayes(Exited ~ ., data = trainData_selected)
dnb

# 6. Predictions for Testing Data

# Calculate probabilities or determine class membership using predict function
# Syntax: predict(Name_of_NB_object, newdata = Name_of_Testing_Data_Frame.,
#                 type = "raw")
#                 type parameter: "raw" yields propensities

# 6.1 For Each Record Calculate class probabilities

# Calculate probabilities of each class
pred_prob <- predict(dnb, newdata = testData, type = "raw")
```

```
# 6.2 For Each Record Predict Class Membership

# Predict class membership
pred_class <- predict(dnb, newdata = testData)

# 6.3 For the Testing Data set See the Results

# Create a new data frame to see actual and predicted values
# Along with probbailities of belonong to ontime or delayed class
ndf <- data.frame(Actual = testData$Exited, Predicted = pred_class,
                  Probaility = pred_prob)
ndf

# Confusion Matrix and Statistics
cm <- confusionMatrix(as.factor(pred_class), as.factor(testData$Exited))
cm
# Accuracy
accuracy <- cm$overall['Accuracy']
accuracy
# Precision
precision <- cm$byClass['Pos Pred Value']
precision

# Recall (Sensitivity)
recall <- cm$byClass['Sensitivity']
recall

# F1 Score
f1_score <- 2*precision*recall/(precision + recall)
f1_score

roc_obj <- roc(testData$Exited, pred_prob[, 2])
auc(roc_obj)



# Support Vector Machine Implementation
# Selected features by the LASSO method
selected_features <- c("CreditScore", "Age", "Tenure", "Balance", "NumOfProducts",
                       "HasCrCard", "IsActiveMember", "EstimatedSalary",
                       "Geography_Germany", "Geography_Spain", "Gender_Male")

# Select only the selected features from the training and test data
trainData_selected <- trainData[, c("Exited", selected_features)]
testData_selected <- testData[, c("Exited", selected_features)]
head(trainData_selected)

# Convert 'Exited' column into factor
trainData_selected$Exited <- as.factor(trainData_selected$Exited)
testData_selected$Exited <- as.factor(testData_selected$Exited)

# Train SVM model with RADIAL kernel
svmfit <- svm(Exited ~ ., data = trainData_selected, kernel = "radial", cost = 5, scale = FALSE)

# Predict churn on test data
predicted_churn <- predict(svmfit, testData_selected)

# Evaluate model's performance
# Here, we will use a confusion matrix

confusionMatrix(table(predicted_churn, testData_selected$Exited))

# For radial kernel
predicted_churn <- factor(predicted_churn, levels = levels(testData_selected$Exited))
confusion <- table(predicted_churn, testData_selected$Exited)
accuracy <- sum(diag(confusion)) / sum(confusion)
precision <- confusion[2,2] / sum(confusion[,2])
recall <- confusion[2,2] / sum(confusion[2,])
f1_score <- 2 * ((precision * recall) / (precision + recall))
```

```
roc_obj <- roc(testData_selected$Exited, as.numeric(predicted_churn))
auc <- auc(roc_obj)

cat("\nRadial Kernel\n")
cat("Accuracy:", accuracy, "\n")
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1 Score:", f1_score, "\n")
cat("AUC:", auc, "\n")


# Train SVM model with LINEAR KERNAL
svmfit_linear <- svm(Exited ~ ., data = trainData_selected, kernel = "linear", cost = 5, scale = FALSE)

# Predict churn on test data
predicted_churn_linear <- predict(svmfit_linear, testData_selected)

# Evaluate model
confusionMatrix(table(predicted_churn_linear, testData_selected$Exited))

# For linear kernel
predicted_churn_linear <- factor(predicted_churn_linear, levels = levels(testData_selected$Exited))
confusion <- table(predicted_churn_linear, testData_selected$Exited)
accuracy <- sum(diag(confusion)) / sum(confusion)
precision <- confusion[2,2] / sum(confusion[,2])
recall <- confusion[2,2] / sum(confusion[2,])
f1_score <- 2 * ((precision * recall) / (precision + recall))

roc_obj <- roc(testData_selected$Exited, as.numeric(predicted_churn_linear))
auc <- auc(roc_obj)

cat("\nLinear Kernel\n")
cat("Accuracy:", accuracy, "\n")
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1 Score:", f1_score, "\n")
cat("AUC:", auc, "\n")


# Train SVM model with POLYNOMIAL kernel
svmfit_poly <- svm(Exited ~ ., data = trainData_selected, kernel = "polynomial", cost = 5, scale = FALSE)

# Predict churn on test data
predicted_churn_poly <- predict(svmfit_poly, testData_selected)

# Evaluate model
confusionMatrix(table(predicted_churn_poly, testData_selected$Exited))

# For polynomial kernel
predicted_churn_poly <- factor(predicted_churn_poly, levels = levels(testData_selected$Exited))
confusion <- table(predicted_churn_poly, testData_selected$Exited)
accuracy <- sum(diag(confusion)) / sum(confusion)
precision <- confusion[2,2] / sum(confusion[,2])
recall <- confusion[2,2] / sum(confusion[2,])
f1_score <- 2 * ((precision * recall) / (precision + recall))

roc_obj <- roc(testData_selected$Exited, as.numeric(predicted_churn_poly))
auc <- auc(roc_obj)

cat("\nPolynomial Kernel\n")
cat("Accuracy:", accuracy, "\n")
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1 Score:", f1_score, "\n")
cat("AUC:", auc, "\n")



# Ensemble Methods
trainData_selected <- trainData[, c("Exited", selected_features)]
```

```
testData_selected <- testData[, c("Exited", selected_features)]

# 1. Bagging Model
library(ipred)

# Create a bagging model
bagging_model <- ipred::bagging(Exited ~ ., data = trainData_selected, nbagg = 100)

# Make predictions
bagging_predictions <- predict(bagging_model, newdata = testData_selected)

# ---
# 2. Boosting
library(xgboost)

# Convert data to xgb.DMatrix object
train_matrix <- xgb.DMatrix(data = as.matrix(trainData[,-1]), label = trainData_selected$Exited)
test_matrix <- xgb.DMatrix(data = as.matrix(testData[,-1]), label = testData_selected$Exited)

# Set parameters
params <- list(booster = "gbtree", objective = "binary:logistic", eta=0.3, gamma=0, max_depth=6, min_child_weight=1, subsample=1,
colsample_bytree=1)

# Train the model
boosting_model <- xgb.train(params = params, data = train_matrix, nrounds = 50)

# Make predictions
boosting_predictions <- predict(boosting_model, newdata = test_matrix)

# ---
# 3. Random Forest
library(randomForest)

# Create a random forest model
rf_model <- randomForest(Exited ~ ., data=trainData_selected, ntree=100)

# Make predictions
rf_predictions <- predict(rf_model, newdata=testData_selected)

# ---
# Performance Evaluation of the Ensemble Models
library(caret)

# Converting the probabilities outputted by the ensemble methods to class labels based on a threshold before evaluation
# The threshold is chosen as 0.5
bagging_predictions <- ifelse(bagging_predictions > 0.5, 1, 0)
bagging_predictions <- as.factor(bagging_predictions)

boosting_predictions <- ifelse(boosting_predictions > 0.5, 1, 0)
boosting_predictions <- as.factor(boosting_predictions)

rf_predictions <- ifelse(rf_predictions > 0.5, 1, 0)
rf_predictions <- as.factor(rf_predictions)

# Bagging model evaluation
bagging_cm <- confusionMatrix(table(bagging_predictions, testData_selected$Exited))
bagging_cm
bagging_accuracy <- bagging_cm$overall['Accuracy']
bagging_precision <- bagging_cm$byClass['Precision']
bagging_recall <- bagging_cm$byClass['Recall']
bagging_F1 <- bagging_cm$byClass['F1']

bagging_cm <- confusionMatrix(table(bagging_predictions, testData_selected$Exited))
bagging_cm
print(paste("Bagging Model Accuracy: ", bagging_accuracy))
print(paste("Bagging Model Precision: ", bagging_precision))
print(paste("Bagging Model Recall: ", bagging_recall))
print(paste("Bagging Model F1 Score: ", bagging_F1))

# Boosting model evaluation
```

```
boosting_cm <- confusionMatrix(table(boosting_predictions, testData_selected$Exited))
boosting_cm
boosting_accuracy <- boosting_cm$overall['Accuracy']
boosting_precision <- boosting_cm$byClass['Precision']
boosting_recall <- boosting_cm$byClass['Recall']
boosting_F1 <- boosting_cm$byClass['F1']

boosting_cm <- confusionMatrix(table(boosting_predictions, testData_selected$Exited))
boosting_cm
print(paste("Boosting Model Accuracy: ", boosting_accuracy))
print(paste("Boosting Model Precision: ", boosting_precision))
print(paste("Boosting Model Recall: ", boosting_recall))
print(paste("Boosting Model F1 Score: ", boosting_F1))

# Random Forest model evaluation
rf_cm <- confusionMatrix(table(rf_predictions, testData_selected$Exited))
rf_accuracy <- rf_cm$overall['Accuracy']
rf_precision <- rf_cm$byClass['Precision']
rf_recall <- rf_cm$byClass['Recall']
rf_F1 <- rf_cm$byClass['F1']

rf_cm <- confusionMatrix(table(rf_predictions, testData_selected$Exited))
rf_cm
print(paste("Random Forest Model Accuracy: ", rf_accuracy))
print(paste("Random Forest Model Precision: ", rf_precision))
print(paste("Random Forest Model Recall: ", rf_recall))
print(paste("Random Forest Model F1 Score: ", rf_F1))
```