Exploration of Bank Churn Rates

Erica Chen

Introduction

The purpose of this project is to explore the bank churn rates using a dataset that contains information about customers of a bank. The dataset includes various features such as customer demographics, credit score, geography, tenure, balance, number of products, credit card ownership, and estimated salary. The main goal is to identify the factors that are most strongly associated with customer churn and develop a predictive model to help the bank proactively address the risk of customer attrition. By analyzing the dataset, we hope to gain insights into customer behavior and preferences that can inform the bank's customer retention strategies. Ultimately, the aim of this project is to help the bank better understand its customers and improve their overall experience with the bank.

PSTAT 131 Final Project

```r
# Suppress warning messages
suppressWarnings({
library(readr)
library(vip)
library(naniar)
library(tidymodels)
library(ISLR)
library(ISLR2)
library(tidyverse)
library(glmnet)
library(modeldata)
library(ggthemes)
library(janitor)
library(kableExtra)
library(yardstick)
library(kknn)
library(corrplot)
library(themis)
library(dplyr)
library(ggplot2)
library(scales)
library(rpart.plot)
library(discrim)
library(klaR)
library(plotly)
library(ranger)
library(xgboost)
tidymodels_prefer()
})
```

```
##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
```

```
##
##     vi

## -- Attaching packages -------------------------------------- tidymodels 1.0.0 --

## v broom        1.0.3      v recipes      1.0.4
## v dials        1.1.0      v rsample      1.1.1
## v dplyr        1.0.10     v tibble       3.1.8
## v ggplot2      3.4.0      v tidyr        1.3.0
## v infer        1.0.4      v tune         1.0.1
## v modeldata    1.1.0      v workflows    1.1.2
## v parsnip      1.0.4      v workflowsets 1.0.0
## v purrr        1.0.1      v yardstick    1.1.0

## -- Conflicts ----------------------------------------- tidymodels_conflicts() --
## x purrr::discard()  masks scales::discard()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Learn how to get started at https://www.tidymodels.org/start/

##
## Attaching package: 'ISLR2'

## The following objects are masked from 'package:ISLR':
##
##     Auto, Credit

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v stringr 1.5.0     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x scales::col_factor() masks readr::col_factor()
## x purrr::discard()     masks scales::discard()
## x dplyr::filter()      masks stats::filter()
## x stringr::fixed()     masks recipes::fixed()
## x dplyr::lag()         masks stats::lag()
## x yardstick::spec()    masks readr::spec()
## Loading required package: Matrix
##
##
## Attaching package: 'Matrix'
##
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
##
## Loaded glmnet 4.1-6
##
##
```

```
## Attaching package: 'janitor'
##
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
##
##
##
## Attaching package: 'kableExtra'
##
##
## The following object is masked from 'package:dplyr':
##
##     group_rows
##
##
## corrplot 0.92 loaded
##
## Loading required package: rpart
##
##
## Attaching package: 'rpart'
##
##
## The following object is masked from 'package:dials':
##
##     prune
##
##
##
## Attaching package: 'discrim'
##
##
## The following object is masked from 'package:dials':
##
##     smoothness
##
##
## Loading required package: MASS
##
##
## Attaching package: 'MASS'
##
##
## The following object is masked from 'package:ISLR2':
##
##     Boston
##
##
## The following object is masked from 'package:dplyr':
##
##     select
##
```

```
##
##
## Attaching package: 'plotly'
##
##
## The following object is masked from 'package:MASS':
##
##     select
##
##
## The following object is masked from 'package:ggplot2':
##
##     last_plot
##
##
## The following object is masked from 'package:stats':
##
##     filter
##
##
## The following object is masked from 'package:graphics':
##
##     layout
##
##
##
## Attaching package: 'xgboost'
##
##
## The following object is masked from 'package:plotly':
##
##     slice
##
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

Data

Description

This data was found off Kaggle and can be found here

The data contains the following 14 variables.

- RowNumber—corresponds to the record (row) number.
- CustomerId— Customer's ID
- Surname—the surname of a customer
- CreditScore— the credit score of each customer
- Geography—a customer's location
- Gender— customer's gender (F or M)
- Age— customer's age

- Tenure— the number of years that the customer has been a client of the bank
- Balance— amount a customer has in his or her account
- NumOfProducts— refers to the number of products that a customer has purchased through the bank.
- HasCrCard— whether or not a customer has a credit card
- IsActiveMember— whether a customer is active or not
- EstimatedSalary— estimated salary of the customer
- Exited— whether or not the customer left the bank 0 = not exited 1 = exited

Data Cleaning

The daat itself is already pretty organized and tidy; however there are a few variables that we can disregard from the data. We can remove variables CustomerID, Surnames, and RowNumber since these variables have no effect on the customer leaving the bank. This leaves us with 10000 observations and 11 variables to work with.

```
bank_df <- read.csv('/Users/erica/Desktop/churn.csv') %>%
dplyr::select(-CustomerId,-Surname,-RowNumber)
dim(bank_df)
```

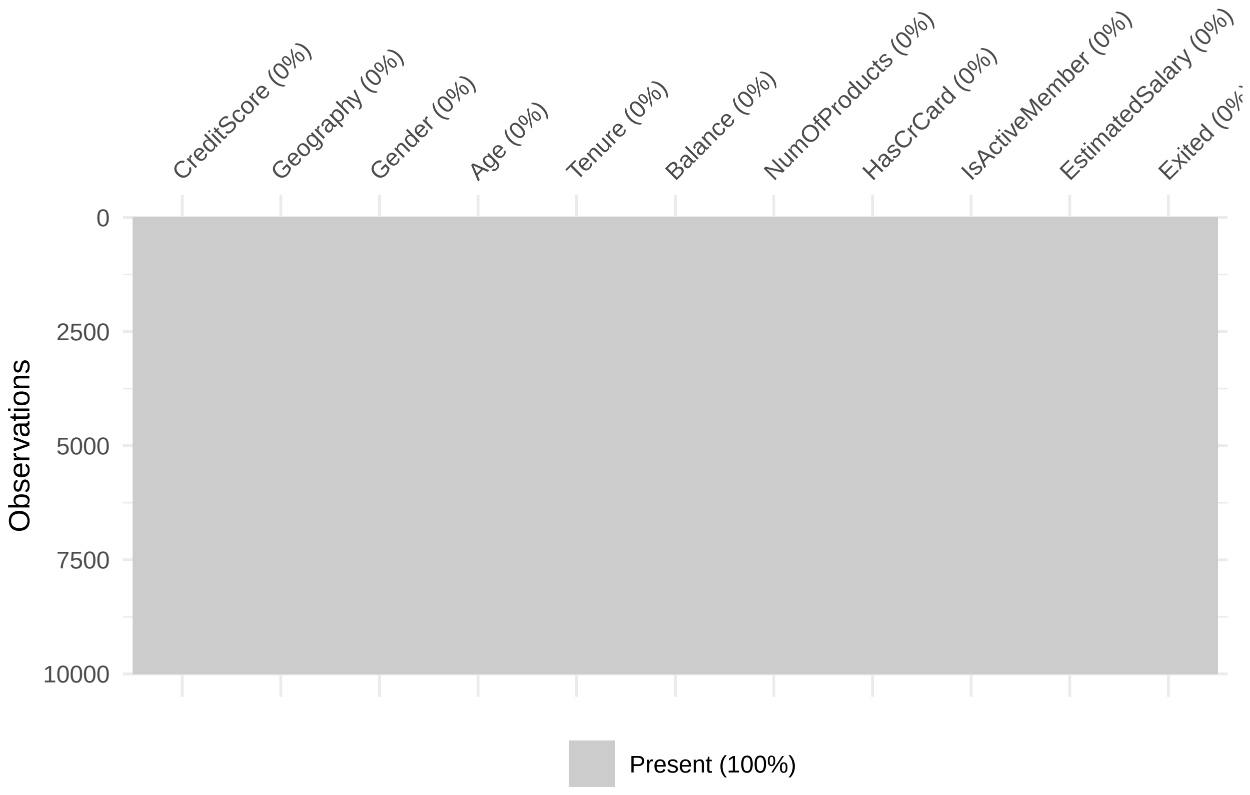
```
## [1] 10000    11
```

Data Exploration

First, let's check if there's any missing values in this dataset.

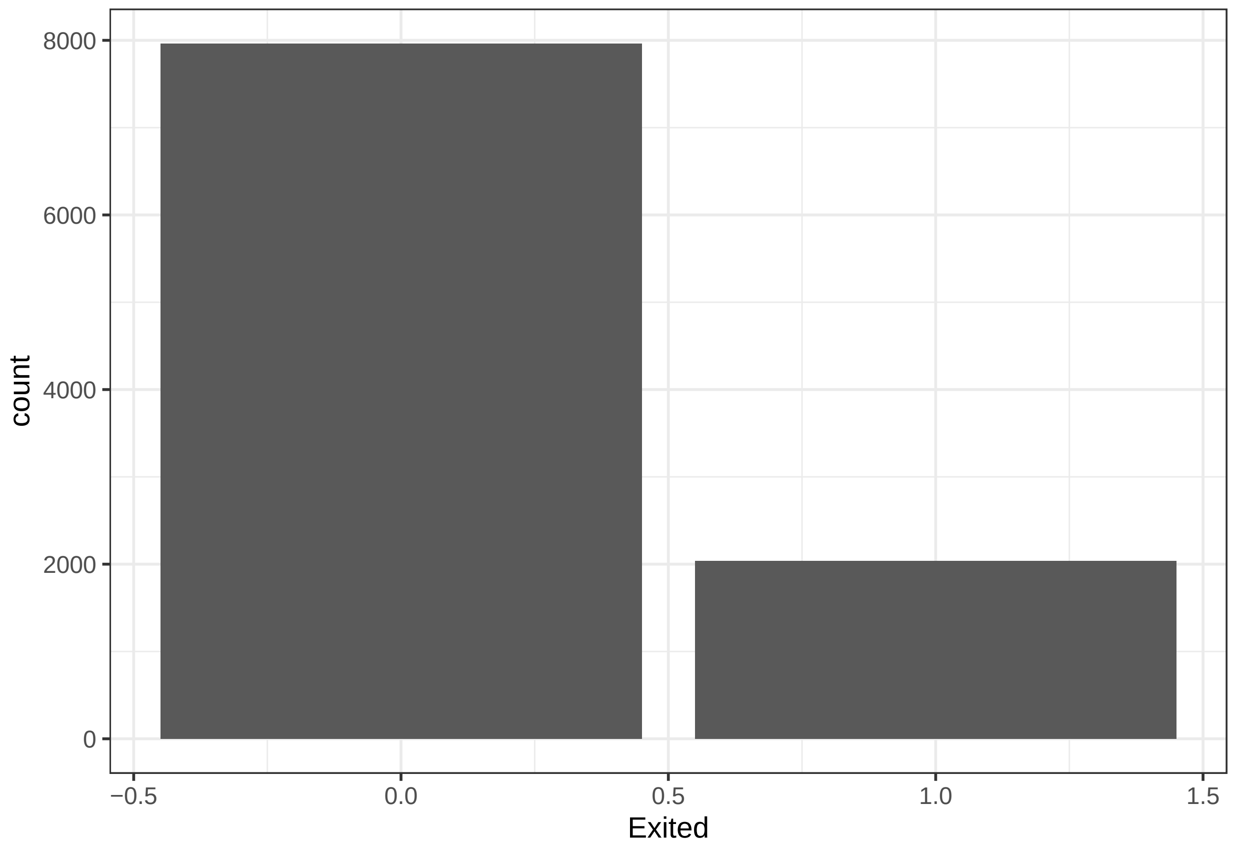
```
vis_miss(bank_df)
```

```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
## i Please use `gather()` instead.
## i The deprecated feature was likely used in the visdat package.
##   Please report the issue at <https://github.com/ropensci/visdat/issues>.
```



Since there's no missing values present, we can safely proceed to further explore the data.

```
bank_df %>%
  ggplot(aes(x = Exited)) + geom_bar() +
  theme_bw()
```

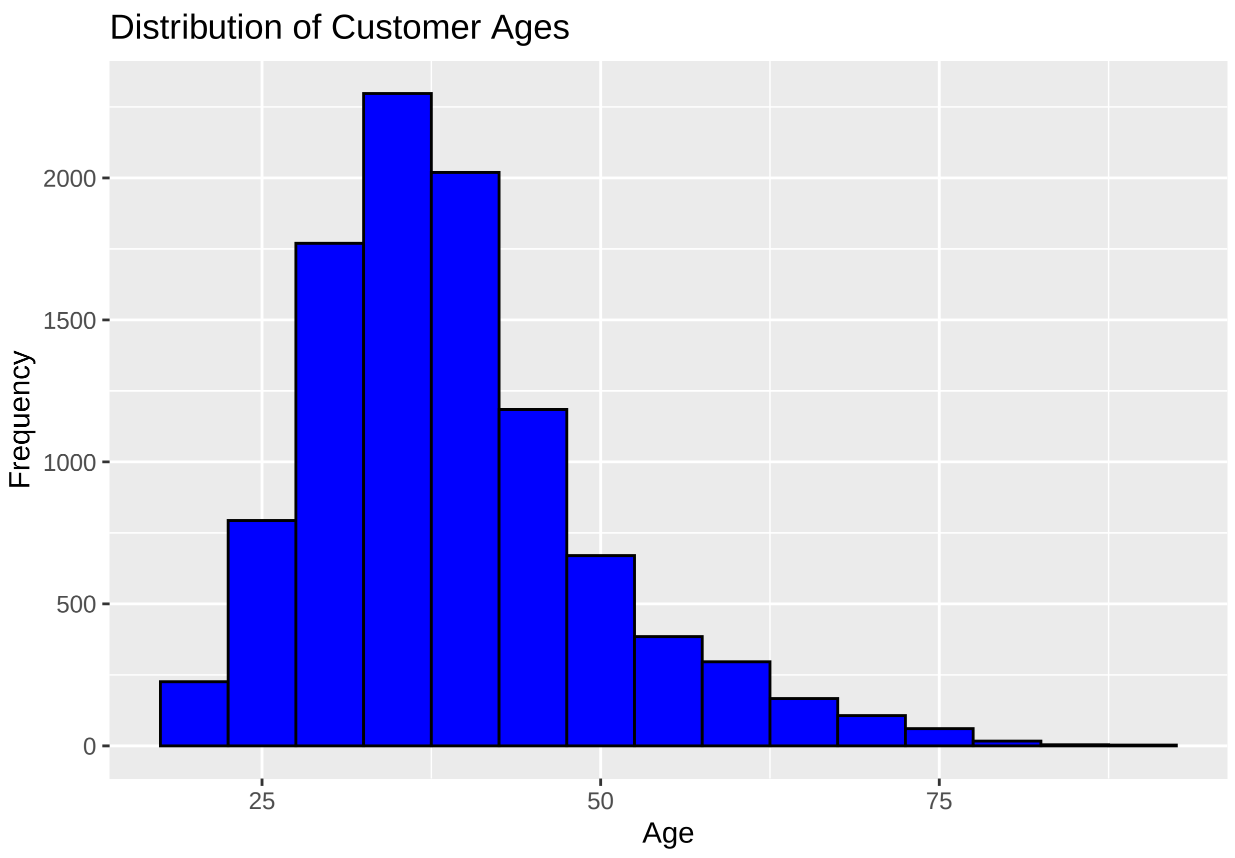


```r
bank_df%>%
  group_by(Exited) %>%
  summarise(prop = n()/(dim(bank_df)[1]))
```

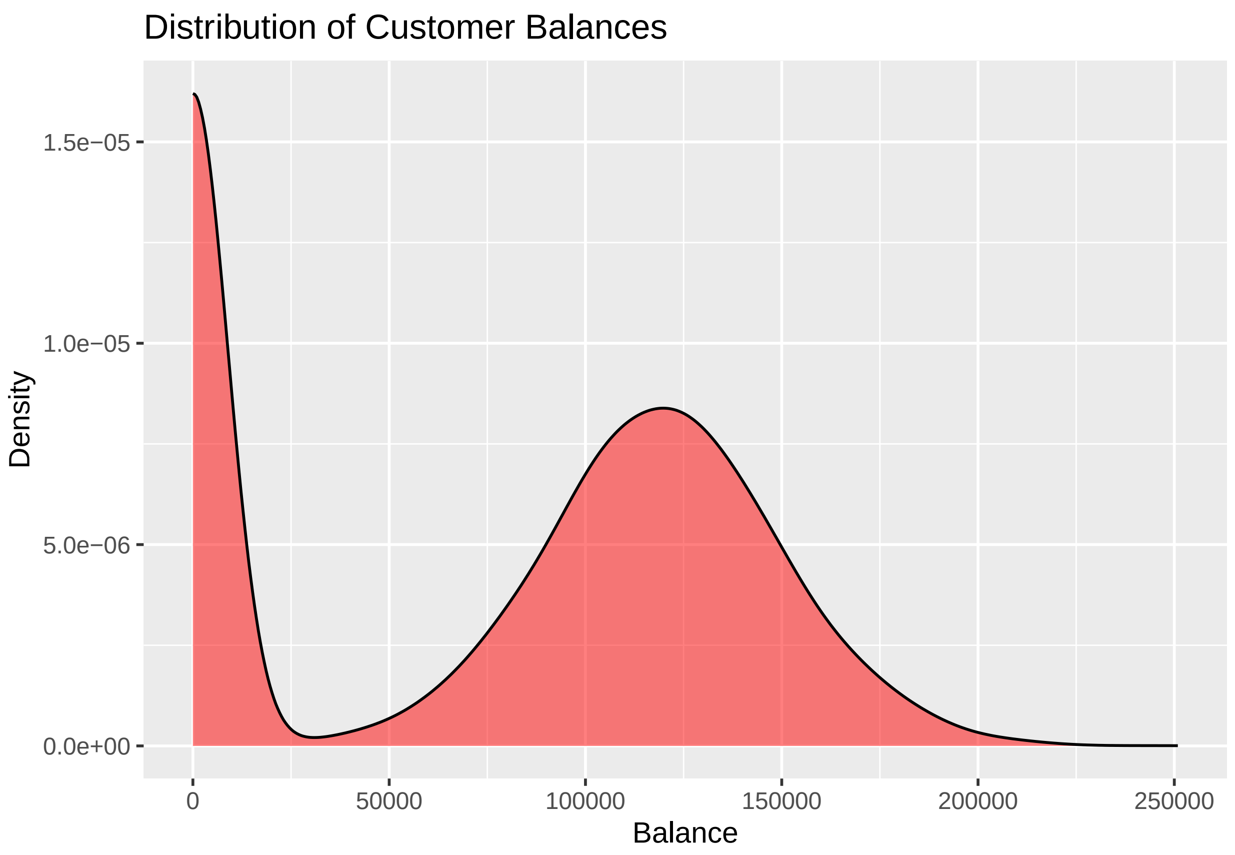
```
## # A tibble: 2 x 2
##   Exited  prop
##    <int> <dbl>
## 1      0 0.796
## 2      1 0.204
```

Here, we can see there's definetely an inbalance between the number of people who exited (1) and who didn't exit (0). Nearly 80% of the data consisted of customers who stayed and only 20% of customers who actually left the bank. This imbalance will likely make it difficult for the model to learn to predict yes accurately, but we'll adjust for it later.

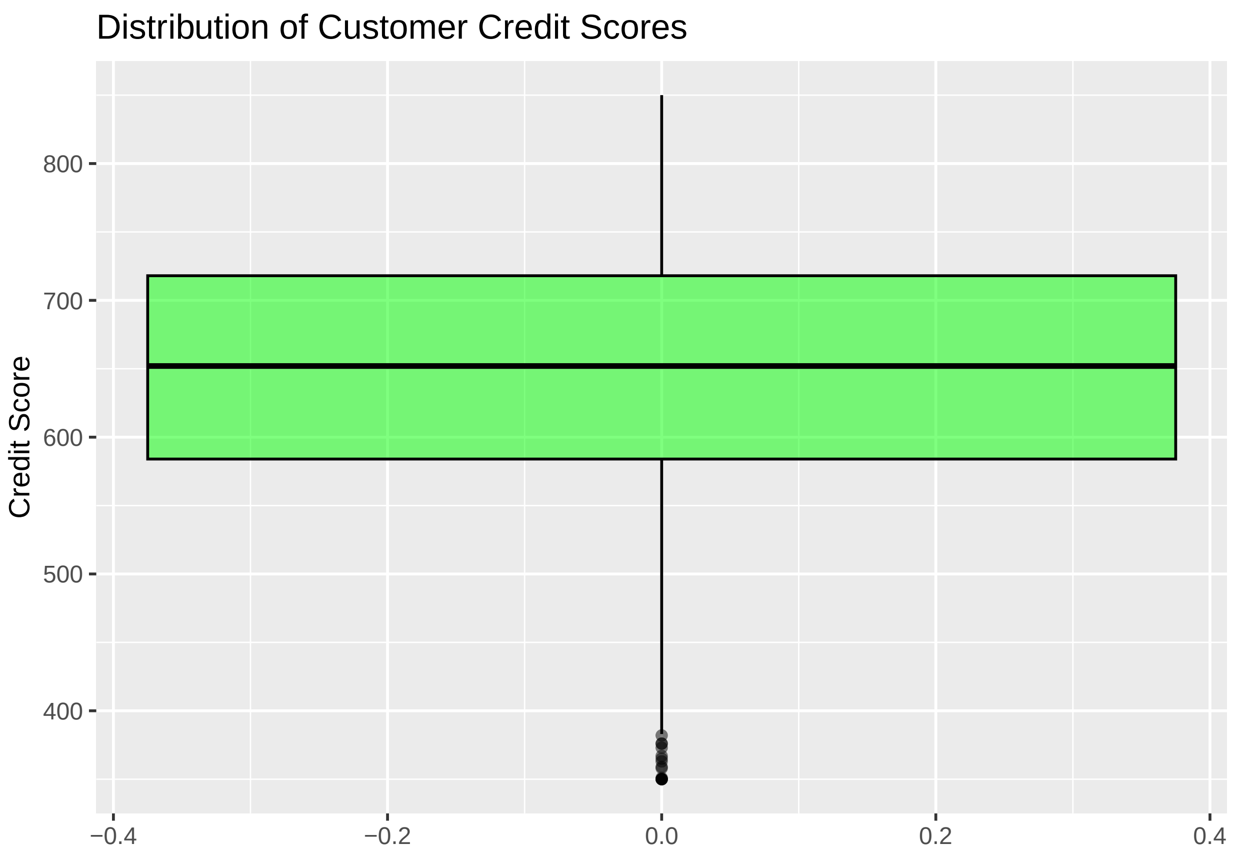
```r
# Plot the distribution of customer ages using a histogram
ggplot(bank_df, aes(x = Age)) +
  geom_histogram(binwidth = 5, color = "black", fill = "blue") +
  labs(title = "Distribution of Customer Ages", x = "Age", y = "Frequency")
```



```r
# Plot the distribution of customer balances using a density plot
ggplot(bank_df, aes(x = Balance)) +
  geom_density(fill = "red", alpha = 0.5) +
  labs(title = "Distribution of Customer Balances", x = "Balance", y = "Density")
```



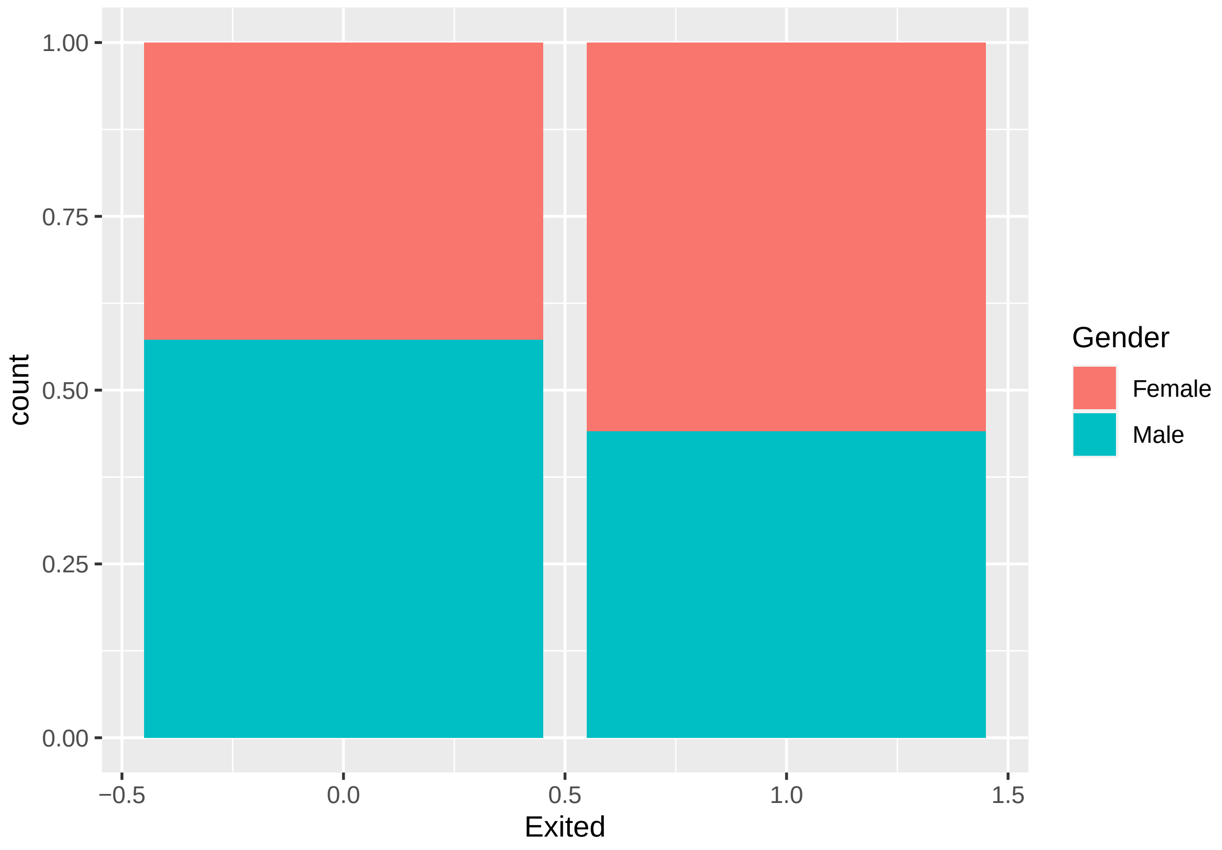
```
# Plot the distribution of customer credit scores using a box plot
ggplot(bank_df, aes(y = CreditScore)) +
  geom_boxplot(color = "black", fill = "green", alpha = 0.5) +
  labs(title = "Distribution of Customer Credit Scores", y = "Credit Score")
```



Here, we use ggplot2 to create three different univariate visualizations of the bank churn dataset. The first visualization is a histogram of customer ages, which shows the distribution of ages in the dataset. The distrubution is more right skewed, with the majorty of customers in their mid 30s to 40s. The second visualization is a density plot of customer balances, which shows the distribution of balances and how they are spread out across different values. The third visualization is a box plot of customer credit scores, which shows the distribution of scores and any outliers that may be present. So it seems 50% of the credit scores fall between 600 and 700 (the box), and the median credit score is 650 (the line within the box). ‘

Let’s see if there’s a significant difference between genders that decide to stay of leave the bank.

```r
# Percent stacked bar of exited by gender
ggplot(bank_df, aes(x = Exited, fill = Gender)) +
    geom_bar(position="fill")
```



```
# Create table of count and percent of customers by gender and churn status
exit_by_gender <- bank_df %>%
  group_by(Exited, Gender) %>%
  summarize(Customers = n()) %>%
  ungroup() %>%
  mutate(percent = round(Customers/nrow(bank_df)*100, 1)) %>%
  rename(Exited = Exited, Gender = Gender, Count = Customers, "Percent (%)" = percent)
```

```
## `summarise()` has grouped output by 'Exited'. You can override using the
## `.groups` argument.
```

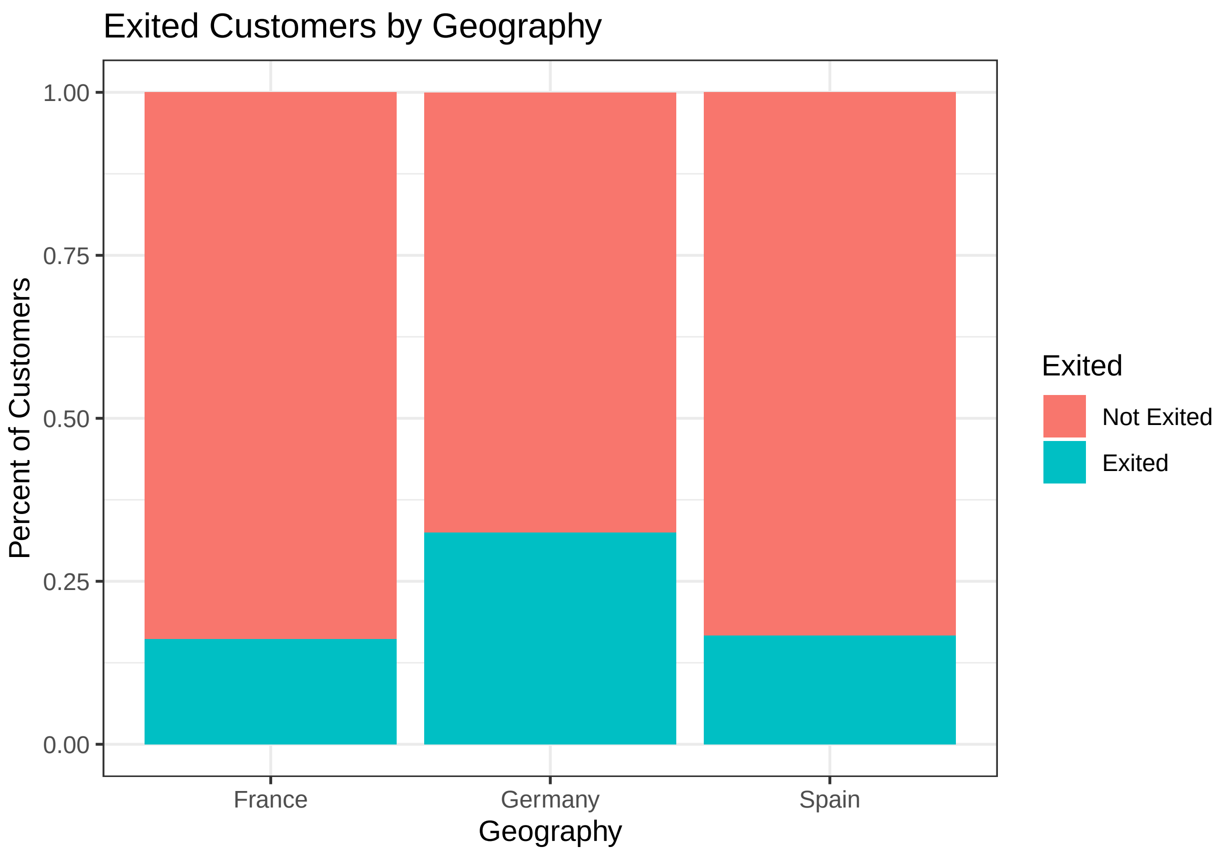
```
t(exit_by_gender)
```

```
##             [,1]     [,2]   [,3]     [,4]
## Exited      "0"      "0"    "1"      "1"
## Gender      "Female" "Male" "Female" "Male"
## Count       "3404"   "4559" "1139"   " 898"
## Percent (%) "34.0"   "45.6" "11.4"   " 9.0"
```

As we can see here from this graph, roughly 50% of both females and males stayed at this bank. Slightly more females left the bank compared to males. From the table, we are shown more specific numbers of how many females/males stayed and or left.

Now this graph and table provide information about customer retention and demographics based on their country of origin. It could be used to analyze patterns and trends in customer behavior across different countries.

```r
ggplot(bank_df, aes(x = Geography, fill = as.factor(Exited))) +
  geom_bar(position = "fill") +
  labs(title = "Exited Customers by Geography", x = "Geography", y = "Percent of Customers") +
  scale_fill_discrete(name = "Exited", labels = c("Not Exited", "Exited")) +
  theme_bw()
```



```r
# Calculate the count and percentage of customers by Exited and Geography
exit_by_geography <- bank_df %>%
  group_by(Exited, Geography) %>%
  summarise(Customers = n()) %>%
  ungroup() %>%
  mutate(percent = round(Customers/nrow(bank_df)*100, 2))
```

```
## `summarise()` has grouped output by 'Exited'. You can override using the
## `.groups` argument.
```

```r
# Transpose the data frame
t(exit_by_geography)
```

```
##           [,1]     [,2]      [,3]    [,4]     [,5]      [,6]
## Exited    "0"      "0"       "0"     "1"      "1"       "1"
## Geography "France" "Germany" "Spain" "France" "Germany" "Spain"
## Customers "4204"   "1695"    "2064"  " 810"   " 814"    " 413"
## percent   "42.04"  "16.95"   "20.64" " 8.10"  " 8.14"   " 4.13"
```

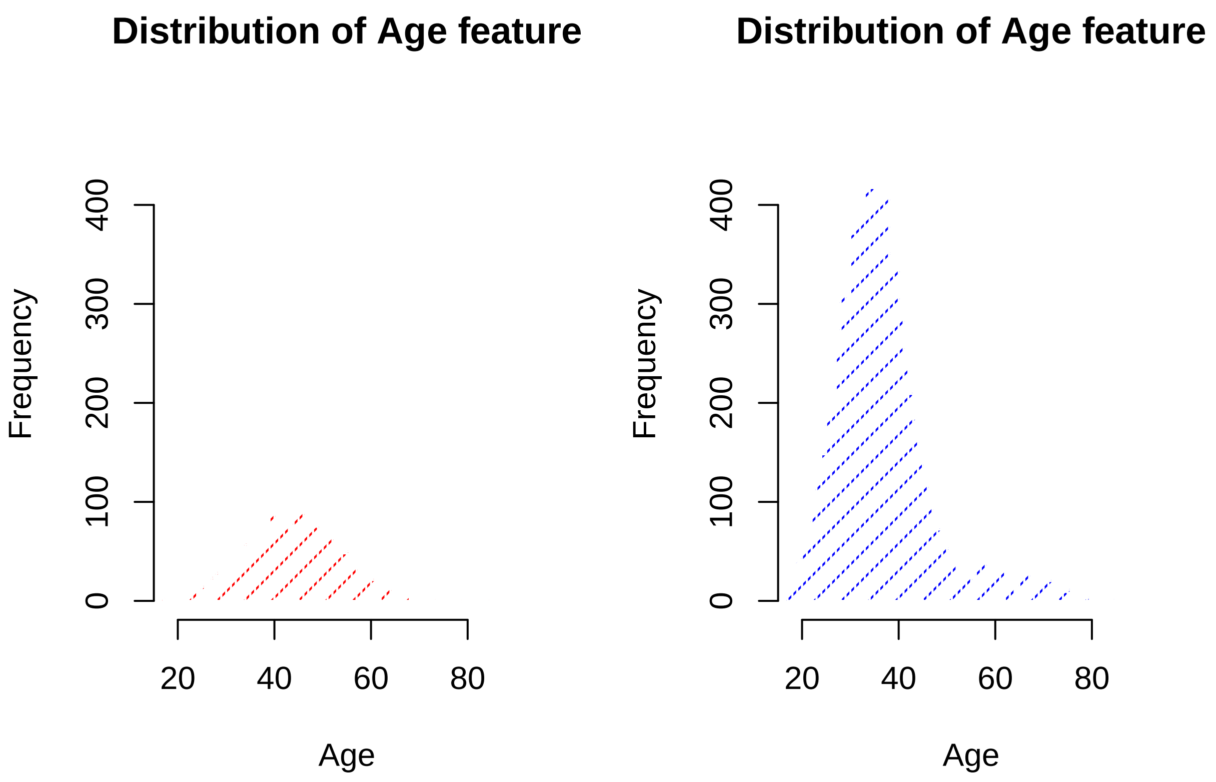
From the table, we can see that French customers embodies over 40% of the customers that stay at this bank. However, proportionally the amount of French customers that stayed and left are the same with Spain. Compared to the other two countries, Germany has the highest percentage of custoers that left.

```
conflicted::conflict_prefer("layout", "plotly")
```

```
## [conflicted] Will prefer plotly::layout over any other package
```

Now this graph provides information about customer retention based on their age . It could be used to analyze patterns and trends in customer behavior across different ages. Red indicates that they exited the bank and blue indicates they stayed.

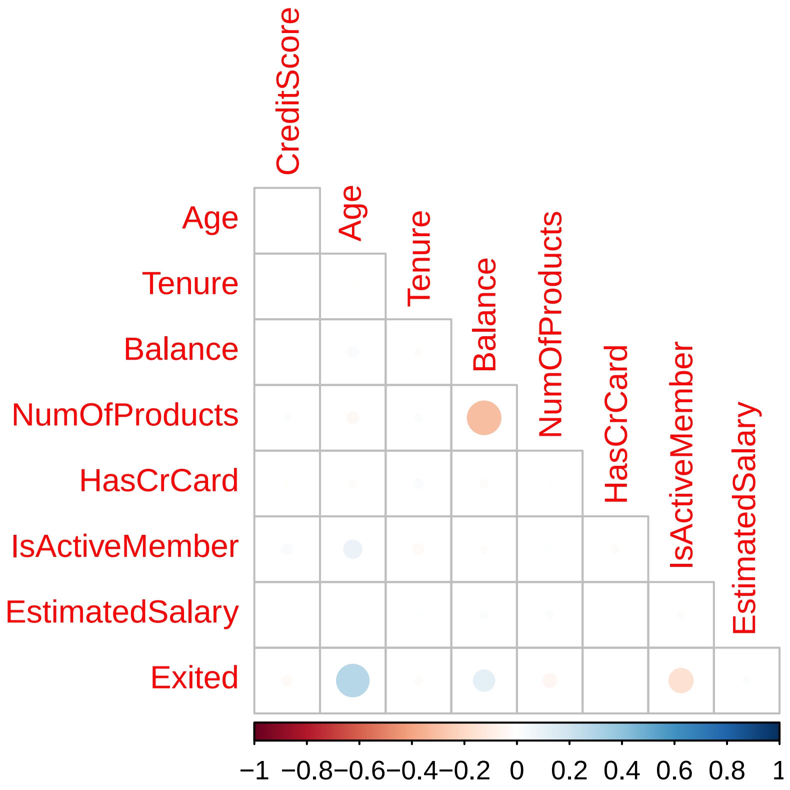
```
distribution <- function(dataframe, column) {
  min_col <- min(dataframe[[column]])
  max_col <- max(dataframe[[column]])

  # Split the data by "Exited" group
  exited_data <- dataframe[dataframe$Exited == 1, column]
  not_exited_data <- dataframe[dataframe$Exited == 0, column]

  # Set up the plot with two side-by-side histograms
  par(mfrow = c(1, 2))

  # Histogram for "Exited == 1"
  hist(exited_data,
       main = paste0("Distribution of ", column, " feature"),
       xlab = "Age",
       ylab = "Frequency",
       col = "red",
       breaks = seq(min_col - 1, max_col + 1, by = 1),
       xlim = c(min_col, max_col),
       ylim = c(0, max(hist(dataframe[[column]],
                            breaks = seq(min_col - 1, max_col + 1, by = 1),
                            plot = FALSE)$counts)),
       density = 10,
       angle = 45,
       border = "white")

  # Histogram for "Exited == 0"
  hist(not_exited_data,
       main = paste0("Distribution of ", column, " feature"),
       xlab = "Age",
       ylab = "Frequency",
       col = "blue",
       breaks = seq(min_col - 1, max_col + 1, by = 1),
       xlim = c(min_col, max_col),
       ylim = c(0, max(hist(dataframe[[column]],
                            breaks = seq(min_col - 1, max_col + 1, by = 1),
                            plot = FALSE)$counts)),
       density = 10,
       angle = 45,
       border = "white")
  # Reset the layout
  par(mfrow = c(1, 1))
}

distribution(bank_df, "Age")
```



So from this histogram, we can see that there's an higher percentage of older customers that left the bank compared to younger customers.

Now let us look at how numerical variables interact with one another.

```
#bank_df$Exited_num <- as.numeric(bank_df$Exited) # Convert Exited to numeric variable
bank_df_numeric <- bank_df %>%
  select_if(is.numeric) # Select only numeric columns
cor_matrix <- cor(bank_df_numeric) # Compute correlation matrix
corrplot(cor_matrix, type = "lower", diag = FALSE) # Plot correlation matrix
```



Conclusion of Exploration In conclusion, the exploration of the data revealed interesting insights about the relationship between various variables. Age was found to have a positive correlation with the number of customers who left the bank, indicating that older customers are more likely to exit. Additionally, there was a negative correlation between balance and the number of products a customer had, meaning that as the number of products increased, the balance decreased. The variable IsActiveMember showed a weak negative correlation with Exited, suggesting that customers who are active are less likely to leave the bank. Finally, there was no significant correlation found between EstimatedSalary, CreditScore, or Tenure and any of the other variables. Overall, these findings provide valuable information that can help the bank to better understand its customers and make informed decisions about how to improve customer satisfaction and retention.

```r
# covert survived and pclass into factors
bank_df$Exited <- as.factor(bank_df$Exited)
# sort the data frame by survived, so the yes will be on top
bank_sort <- bank_df %>% arrange(desc(Exited))
head(bank_sort)
```

```
##   CreditScore Geography Gender Age Tenure  Balance NumOfProducts HasCrCard
## 1         619    France Female  42      2      0.0             1         1
## 2         502    France Female  42      8 159660.8             3         1
## 3         645     Spain   Male  44      8 113755.8             2         1
## 4         376   Germany Female  29      4 115046.7             4         1
## 5         653   Germany   Male  58      1 132602.9             1         1
## 6         510     Spain Female  38      4      0.0             1         1
##   IsActiveMember EstimatedSalary Exited
## 1              1       101348.88      1
## 2              0       113931.57      1
## 3              0       149756.71      1
## 4              0       119346.88      1
## 5              0         5097.67      1
## 6              0       118913.53      1
```

##Cross-Validation + Data Splitting Here, we split the bank dataset into training and testing data using the initial_split() function, with 75% of the data allocated for training and 25% for testing. We also used the strata parameter to ensure that the split is balanced based on the "Exited" column. Next, we created a vfold_cv() object with six folds, which is a cross-validation technique that splits the training data into six equal-sized subsets, trains the model on five of them, and evaluates the model's performance on the remaining subset. This process is repeated six times, so each subset is used once as the validation set.

```
set.seed(3435)
bank_split <- initial_split(bank_df, prop = 0.75,
                               strata = "Exited")
bank_df_train <- training(bank_split)
bank_test <- testing(bank_split)
bank_fold <- vfold_cv(bank_df_train,v=6)
```

Recipe

To address the inbalance in the levels of outcome, I decided to step_upsample() to fix this problem. The step_dummy() function is used to convert all nominal variables into dummy variables. The step_normalize() function is used to scale all variables to have a mean of zero and standard deviation of one. The step_upsample() function is used to oversample the minority class (Exited == 1) by a factor of 0.5.

The prep() function is then used to prepare the recipe object, and bake() is used to apply the recipe to the training set. Finally, the group_by() and summarise() functions are used to count the number of instances of each value of the Exited variable in the oversampled training set.

```
bank_recipe_demo <- recipe(Exited ~ ., data = bank_df_train) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_upsample(Exited, over_ratio = 0.5, skip = FALSE)

prep(bank_recipe_demo) %>% bake(new_data = bank_df_train) %>%
  group_by(Exited) %>%
  summarise(count = n())
```

```
## # A tibble: 2 x 2
##   Exited count
##   <fct>  <int>
## 1 0       5972
## 2 1       2986
```

```
bank_recipe <- recipe(Exited ~ ., data = bank_df_train ) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors()) %>%
  step_upsample(Exited, over_ratio = 1)
```

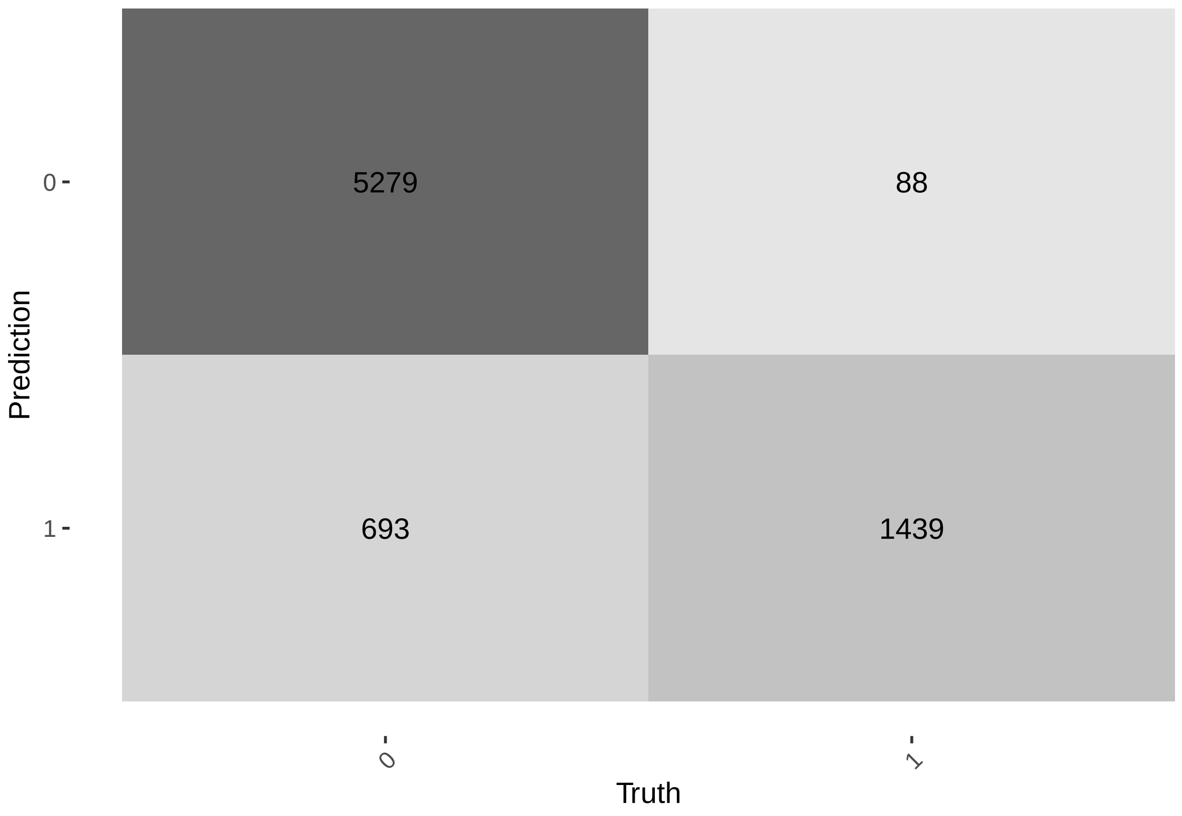
##KNN Model

```r
# knn
knn_model <- nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")
bank_knn_wflow <- workflow() %>%
  add_model(knn_model) %>%
  add_recipe(bank_recipe)
neighbors_grid <- grid_regular(neighbors(range = c(1, 10)), levels = 10)
# knn
bank_tune_knn <- tune_grid(
  object = bank_knn_wflow,
  resamples = bank_fold,
  grid = neighbors_grid
)


best_knn_bank <- select_best(bank_tune_knn,
                          metric = "roc_auc",
                          neighbors
                          )
```

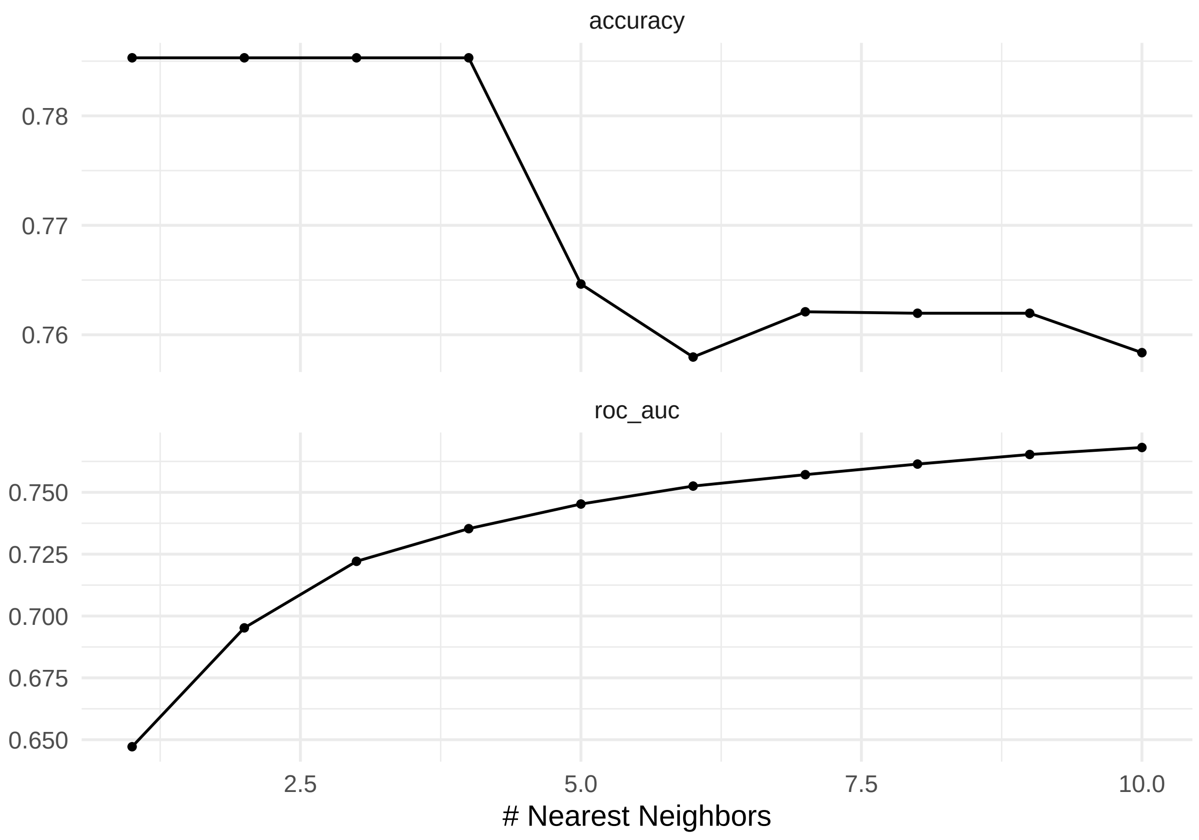
```r
best_knn_wf <- finalize_workflow(bank_knn_wflow,
                                     best_knn_bank)
knn_fit <- fit(best_knn_wf, data = bank_df_train)
augment(knn_fit, new_data = bank_df_train) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.983
```

```r
augment(knn_fit, new_data = bank_df_train) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



```
autoplot(bank_tune_knn) + theme_minimal()
```



##Logistic Regression

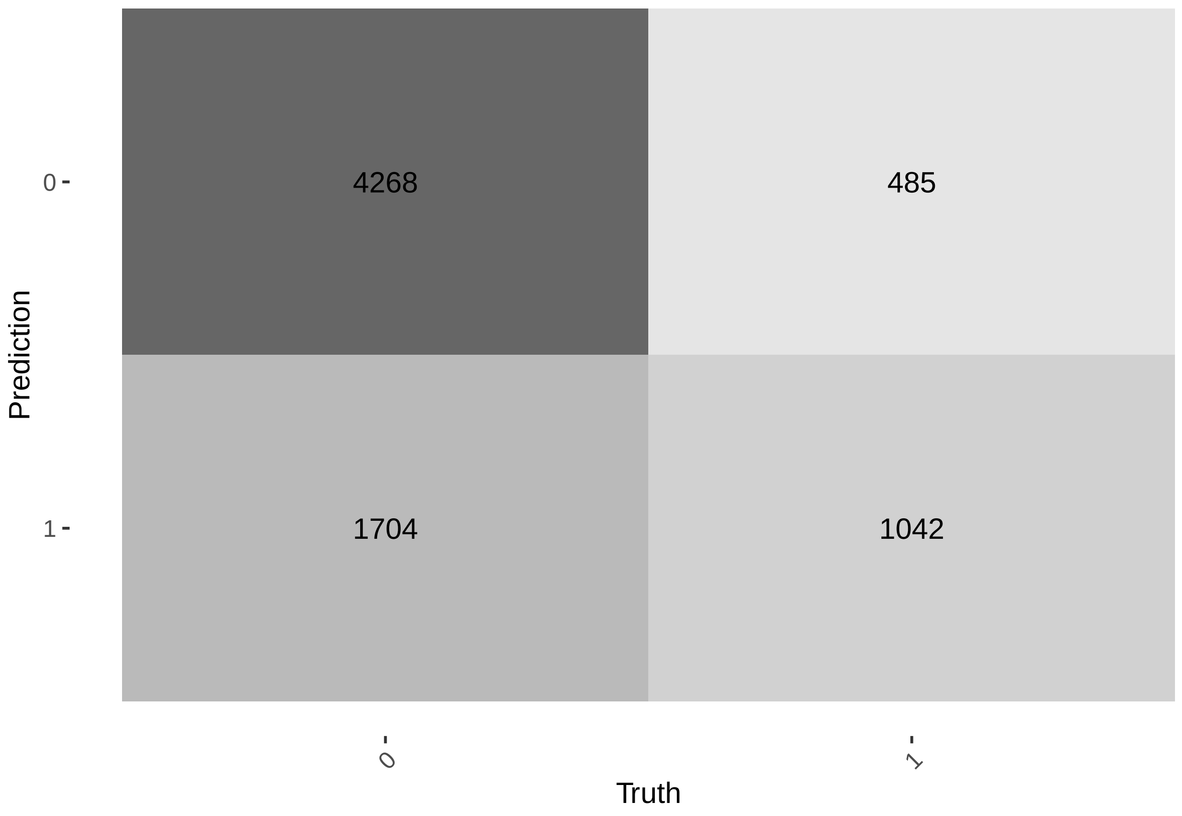
```
# logistic regression
log_reg <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

bank_log_wflow <- workflow() %>%
  add_model(log_reg) %>%
  add_recipe(bank_recipe)

bank_tune_reg <- tune_grid(
  object = bank_log_wflow,
  resamples = bank_fold
)
```

```
## Warning: No tuning parameters have been detected, performance will be evaluated
## using the resamples with no tuning. Did you want to [tune()] parameters?
```

```
log_fit <- fit(bank_log_wflow, bank_df_train)
augment(log_fit, new_data = bank_df_train) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.769
```

```
augment(log_fit, new_data = bank_df_train) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



```r
show_best(bank_tune_reg, n = 1)
```

```
## Warning: No value of `metric` was given; metric 'roc_auc' will be used.

## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.767     6 0.00308 Preprocessor1_Model1
```

```r
select_by_one_std_err(bank_tune_reg, n = 1)
```

```
## Warning: No value of `metric` was given; metric 'roc_auc' will be used.

## # A tibble: 1 x 8
##   .metric .estimator  mean     n std_err .config              .best .bound
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>                <dbl>  <dbl>
## 1 roc_auc binary     0.767     6 0.00308 Preprocessor1_Model1 0.767  0.764
```

#Elastic Net Regression

```r
# elastic net linear regression
enl_model <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")
```

```r
bank_enl_wflow <- workflow() %>%
  add_recipe(bank_recipe) %>%
  add_model(enl_model)

en_grid <- grid_regular(penalty(range = c(-5,5)),
                           mixture(range = c(0, 1)),
                           levels = 10)

# elastic net linear regression
bank_tune_enl <- tune_grid(
  object = bank_enl_wflow,
  resamples = bank_fold,
  grid = en_grid
)

collect_metrics(bank_tune_enl)
```

```
## # A tibble: 200 x 8
##     penalty mixture .metric  .estimator  mean     n std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00001        0 accuracy binary     0.710     6 0.00577 Preprocessor1_Model~
##  2 0.00001        0 roc_auc  binary     0.767     6 0.00309 Preprocessor1_Model~
##  3 0.000129       0 accuracy binary     0.710     6 0.00577 Preprocessor1_Model~
##  4 0.000129       0 roc_auc  binary     0.767     6 0.00309 Preprocessor1_Model~
##  5 0.00167        0 accuracy binary     0.710     6 0.00577 Preprocessor1_Model~
##  6 0.00167        0 roc_auc  binary     0.767     6 0.00309 Preprocessor1_Model~
##  7 0.0215         0 accuracy binary     0.710     6 0.00574 Preprocessor1_Model~
##  8 0.0215         0 roc_auc  binary     0.767     6 0.00309 Preprocessor1_Model~
##  9 0.278          0 accuracy binary     0.712     6 0.00434 Preprocessor1_Model~
## 10 0.278          0 roc_auc  binary     0.765     6 0.00343 Preprocessor1_Model~
## # ... with 190 more rows
```

```r
show_best(bank_tune_enl, n = 1)
```

```
## Warning: No value of 'metric' was given; metric 'roc_auc' will be used.

## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1  0.0215   0.333 roc_auc binary     0.768     6 0.00290 Preprocessor1_Model034
```

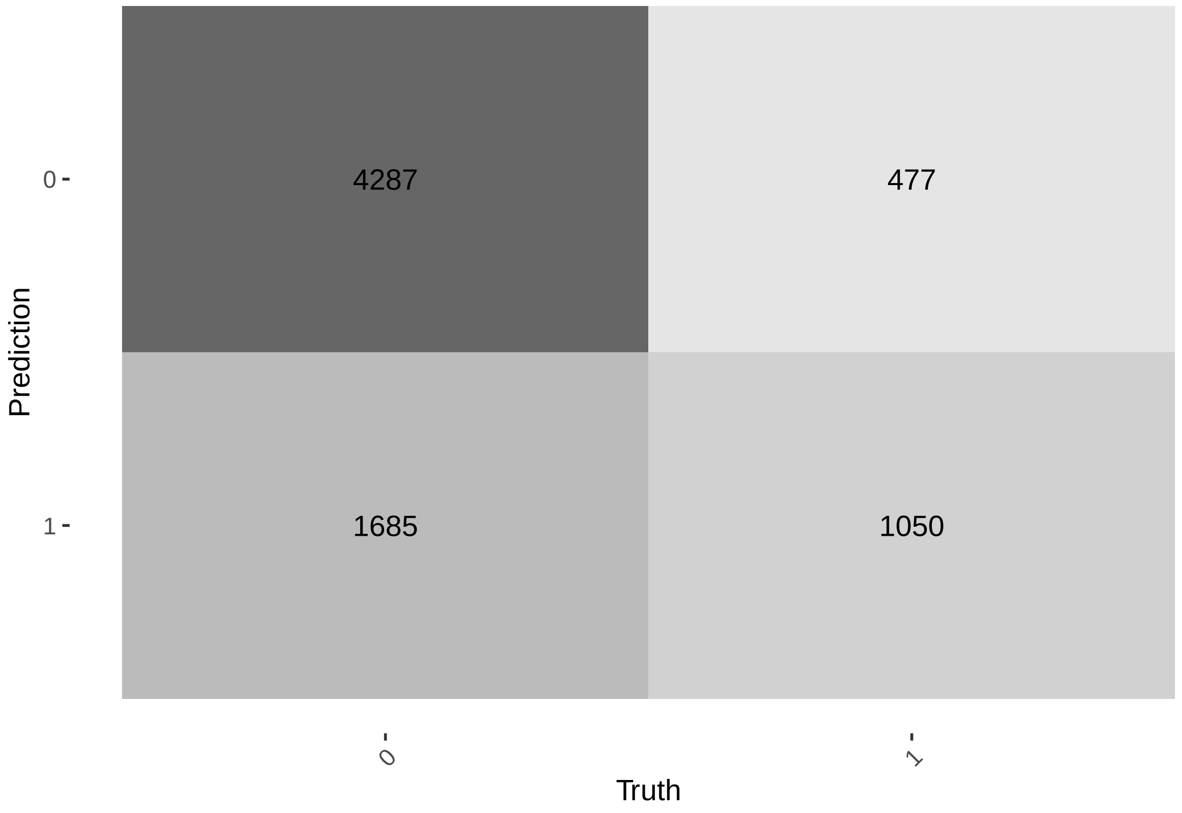
```r
best_enl_bank <- show_best(bank_tune_enl, n = 1)
```

```
## Warning: No value of 'metric' was given; metric 'roc_auc' will be used.
```

```r
bank_enl_final <- finalize_workflow(bank_enl_wflow, best_enl_bank)
bank_enl_final <- fit(bank_enl_final,
                      data = bank_df_train)
#enl_fit <- fit(bank_enl_wflow, bank_df_train)
augment(bank_enl_final, new_data = bank_df_train) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

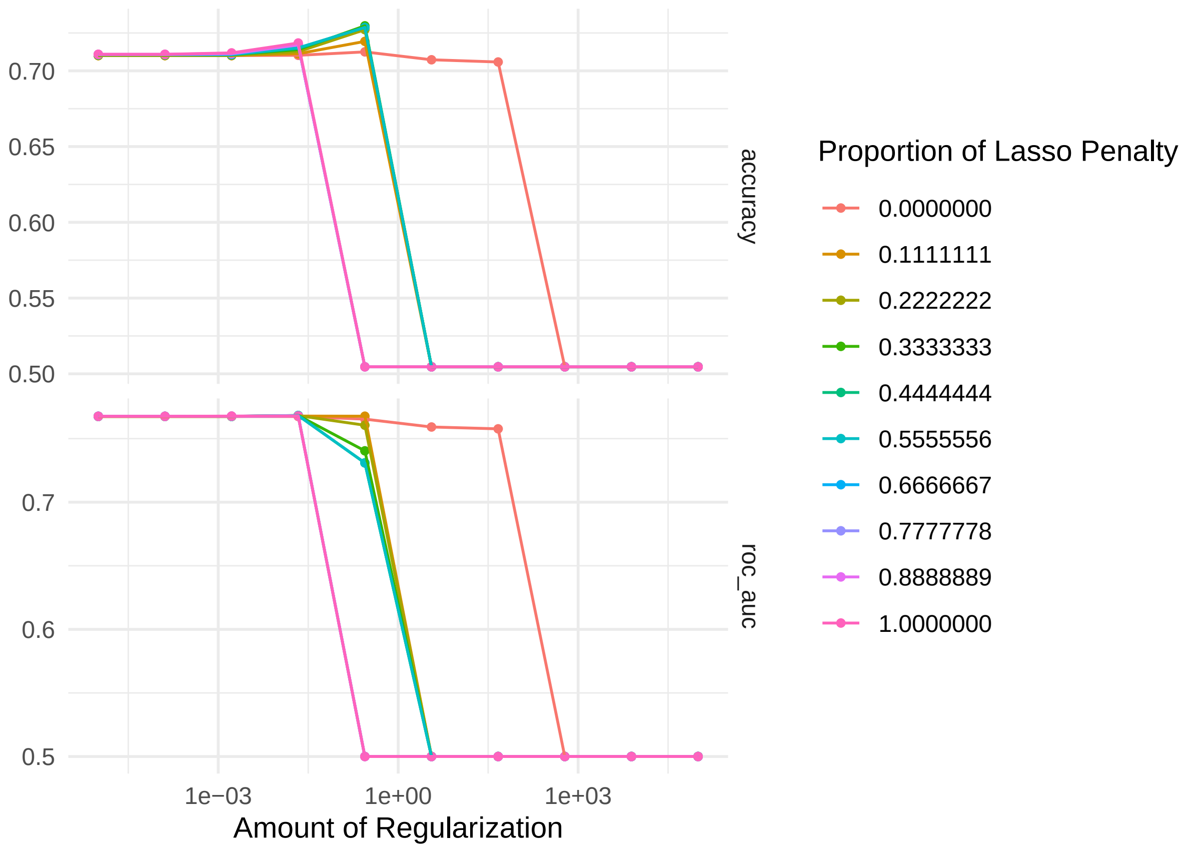
```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.769
```

```
augment(bank_enl_final, new_data = bank_df_train) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



The Lasso penalty can be particularly useful in cases where there are many features, as it can help to identify the most important predictors by shrinking the coefficients of less important features to zero. This can improve the model's interpretability and reduce the risk of overfitting. From the graph, it seems like the best performing line is the orange one.

```
autoplot(bank_tune_enl) + theme_minimal()
```

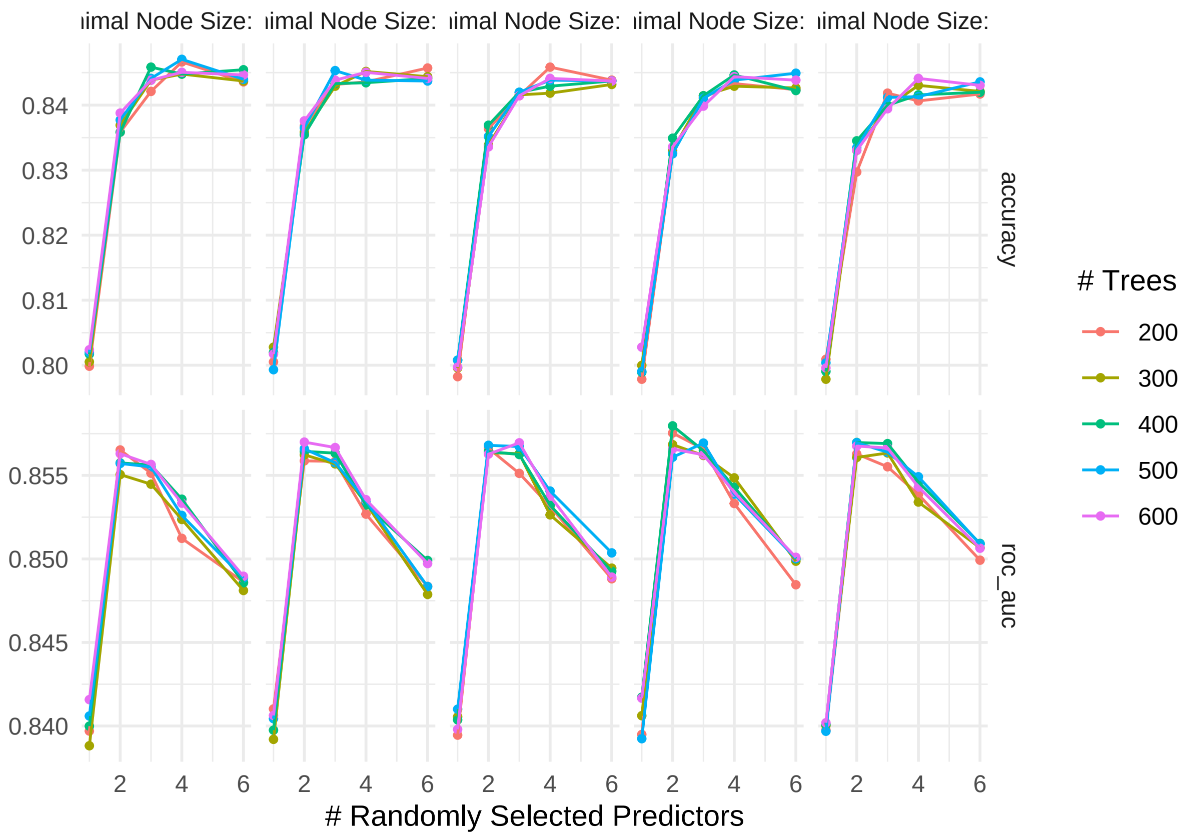


Random Forest

```
rf_class_model <- rand_forest(mtry = tune(),
                              trees = tune(),
                              min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")

rf_class_wf <- workflow() %>%
  add_model(rf_class_model) %>%
  add_recipe(bank_recipe)

param_grid <- grid_regular(
  mtry(range = c(1, 6)),
  trees(range = c(200, 600)),
  min_n(range = c(10, 20)),
  levels = 5
)

bank_tune_rf <- tune_grid(
  object = rf_class_wf,
  resamples = bank_fold,
  grid = param_grid)

save(bank_tune_rf, file = 'bank_tune_rf.rda')
```

```r
load("bank_tune_rf.rda")
```

```r
autoplot(bank_tune_rf) + theme_minimal()
```

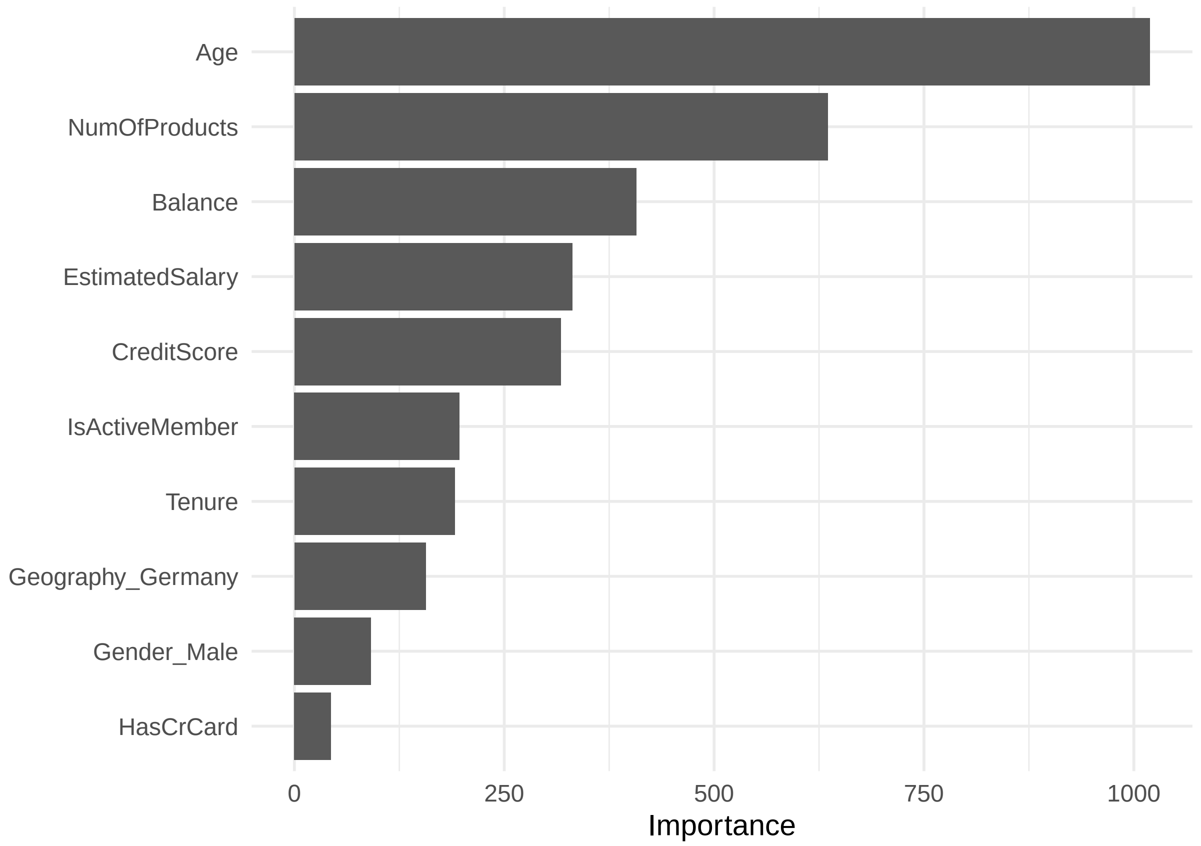


```r
show_best(bank_tune_rf, n = 1)
```

```
## Warning: No value of 'metric' was given; metric 'roc_auc' will be used.

## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     2   400    17 roc_auc binary     0.858     6 0.00455 Preprocessor1_Model0~
```

The output shows that the best Random Forest model has hyperparameters mtry = 2, trees = 400, and min_n = 17

```r
best_rf_model = rand_forest(mtry = 2,
                            trees = 400,
                            min_n = 17) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("classification")
final_rf_wf <- workflow() %>%
  add_model(best_rf_model) %>%
  add_recipe(bank_recipe)
bank_rf_fit <- fit(final_rf_wf, bank_df_train)
bank_rf_fit %>% extract_fit_parsnip() %>%
  vip() +
  theme_minimal()
```

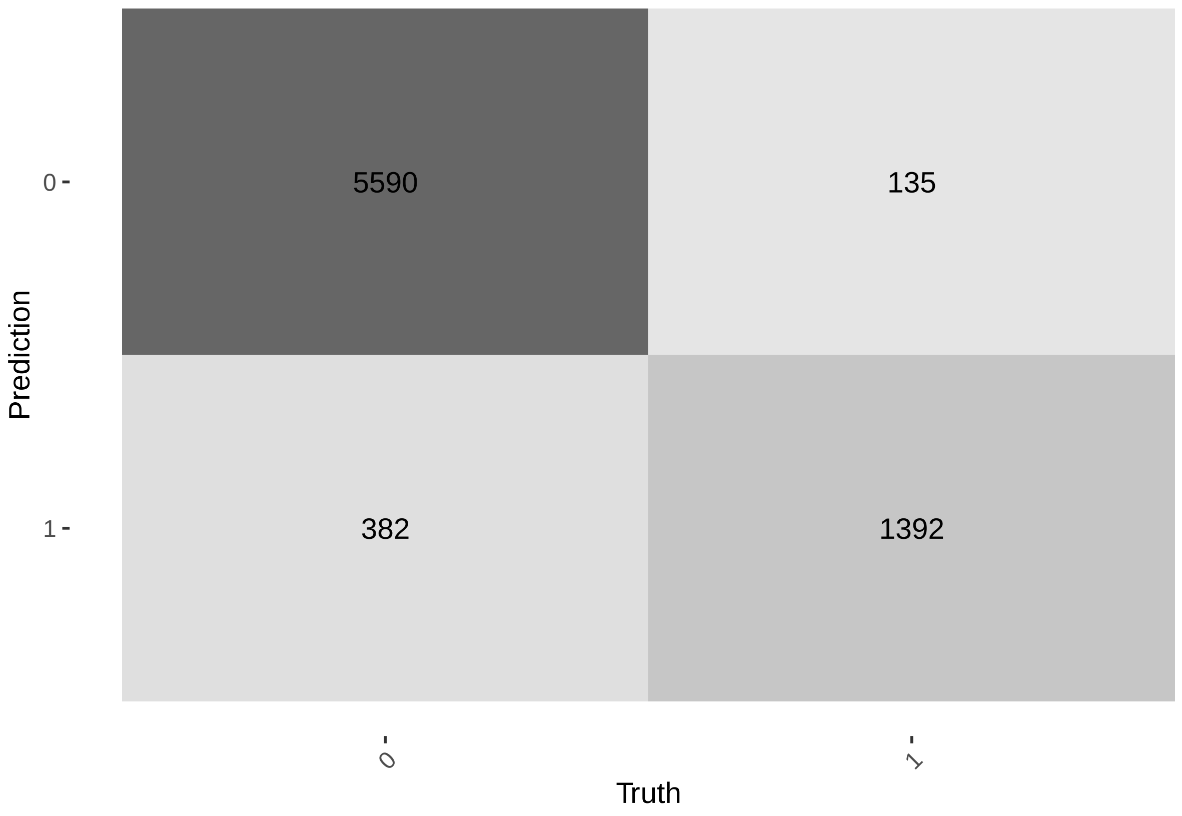


The top 3 predictors are age, num of products, and balance. This was close to my predictions, but I thought balance would be more important.

```
augment(bank_rf_fit, new_data = bank_df_train) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.980
```

```
augment(bank_rf_fit, new_data = bank_df_train) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



##Boosted Tree Model

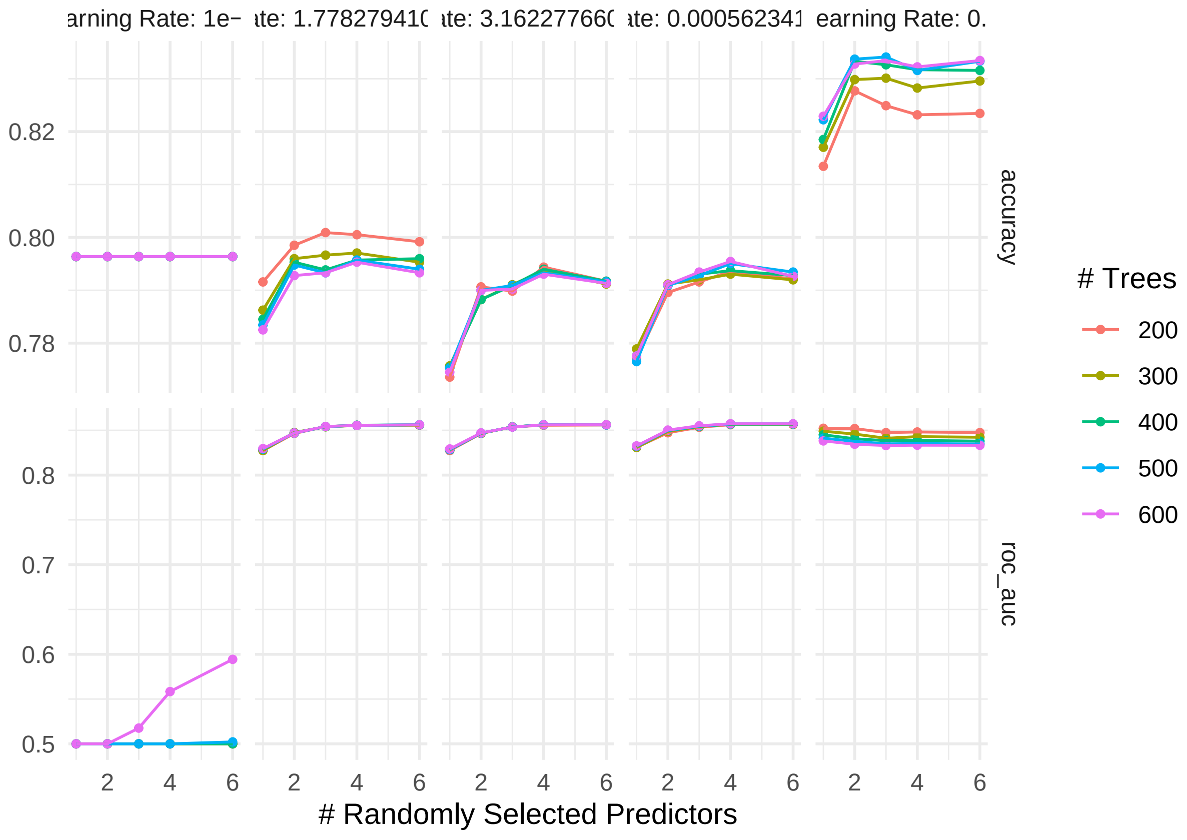
```
bt_class_spec <- boost_tree(mtry = tune(),
                            trees = tune(),
                            learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

bt_class_wf <- workflow() %>%
  add_model(bt_class_spec) %>%
  add_recipe(bank_recipe)
```

```
bt_grid <- grid_regular(mtry(range = c(1, 6)),
                        trees(range = c(200, 600)),
                        learn_rate(range = c(-10, -1)),
                        levels = 5)
bt_grid
```

```
## # A tibble: 125 x 3
##      mtry trees   learn_rate
##     <int> <int>        <dbl>
##   1     1   200 0.0000000001
##   2     2   200 0.0000000001
##   3     3   200 0.0000000001
##   4     4   200 0.0000000001
##   5     6   200 0.0000000001
##   6     1   300 0.0000000001
##   7     2   300 0.0000000001
##   8     3   300 0.0000000001
##   9     4   300 0.0000000001
## 10     6   300 0.0000000001
## # ... with 115 more rows
```

```
tune_bt_class <- tune_grid(
  bt_class_wf,
  resamples = bank_fold,
  grid = bt_grid
)
save(tune_bt_class, file = "tune_bt_class.rda")
```

```
load("tune_bt_class.rda")
autoplot(tune_bt_class) + theme_minimal()
```



```
show_best(tune_bt_class, n = 1)
```

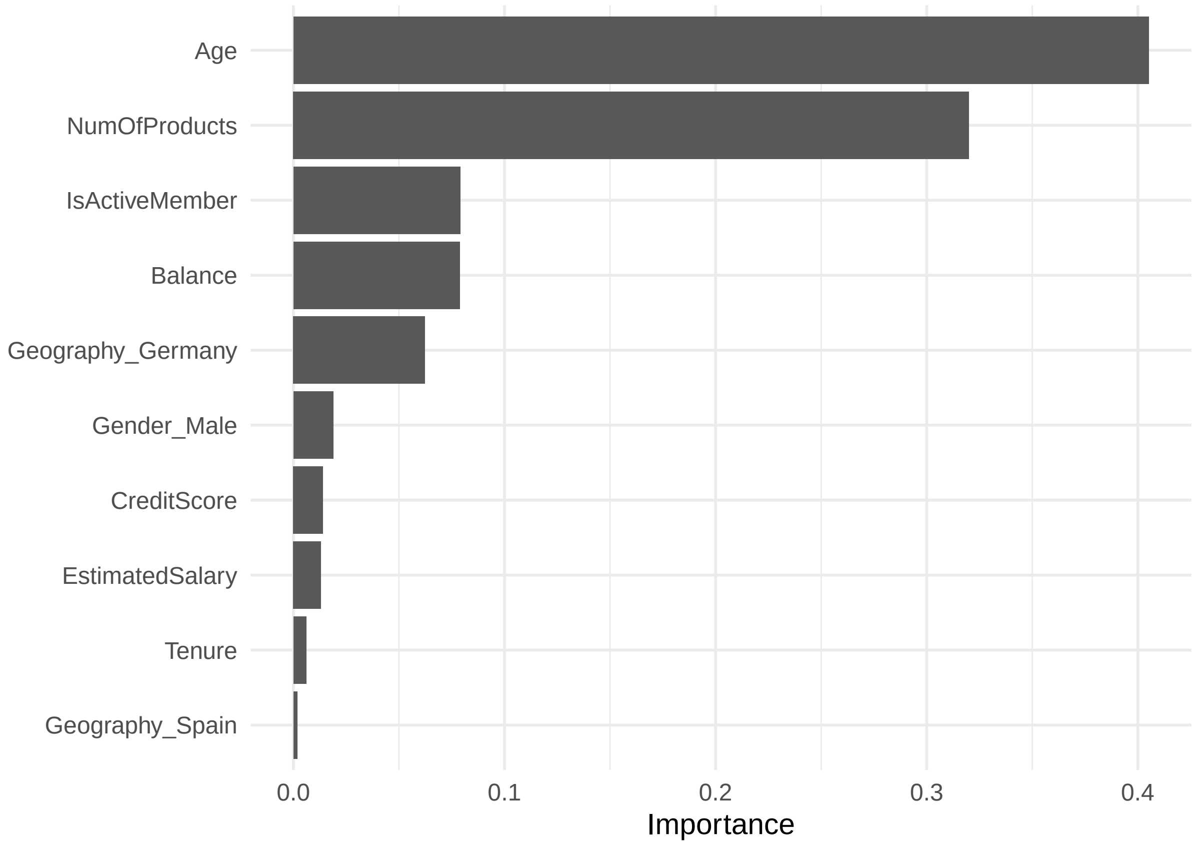
```
## Warning: No value of ‘metric‘ was given; metric ’roc_auc’ will be used.
```

```
## # A tibble: 1 x 9
##    mtry trees learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4   600   0.000562 roc_auc binary     0.857     6 0.00447 Preprocessor1_M~
```

```
best_bt_class <- select_best(tune_bt_class)
```

```
## Warning: No value of `metric` was given; metric 'roc_auc' will be used.
```

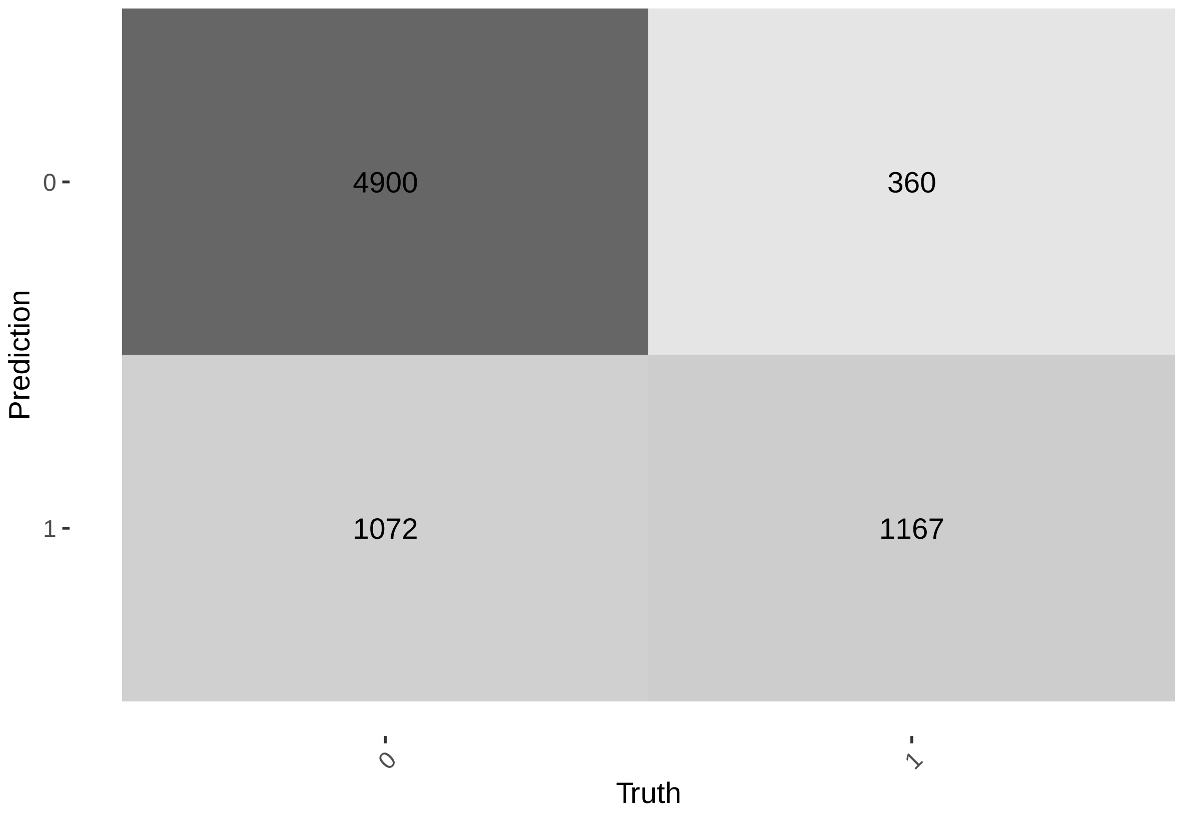
```
bt_mode_fit <- finalize_workflow(bt_class_wf, best_bt_class)
bt_mode_fit <- fit(bt_mode_fit, bank_df_train)


bt_mode_fit %>% extract_fit_parsnip() %>%
  vip() +
  theme_minimal()
```



```
bt_mode_fit <- finalize_workflow(bt_class_wf, best_bt_class)
bt_mode_fit <- fit(bt_mode_fit, bank_df_train)
augment(bt_mode_fit, new_data = bank_df_train) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.883
```

```
augment(bt_mode_fit, new_data = bank_df_train) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```

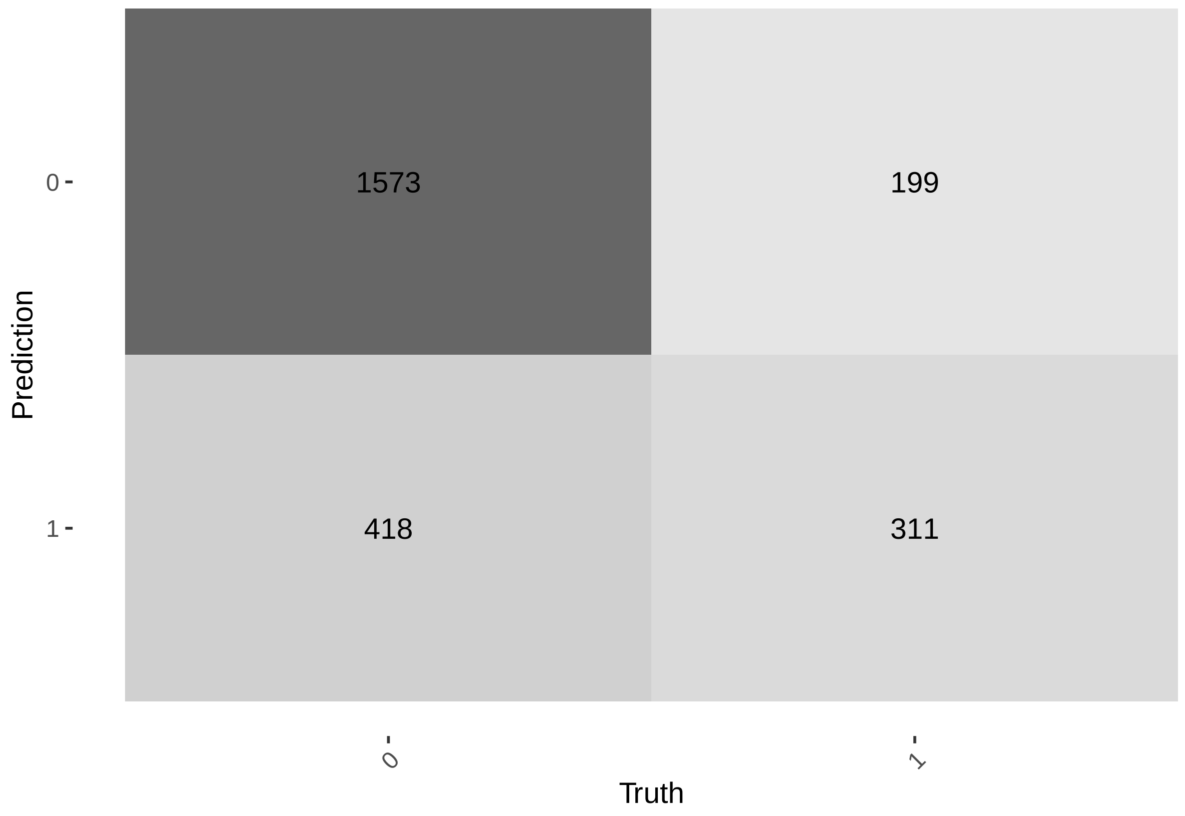


##Test

```
augment(knn_fit, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.763
```

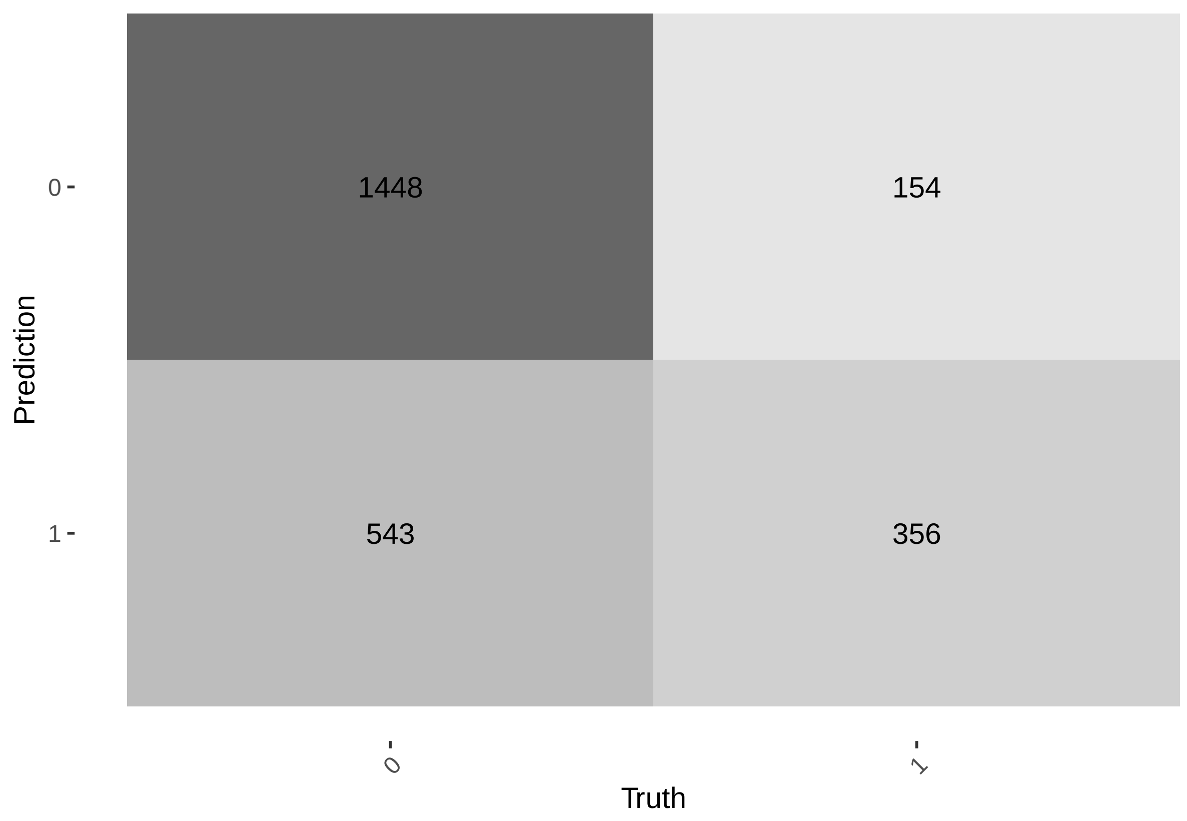
```
augment(knn_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



```
augment(log_fit, new_data = bank_test) %>%
  accuracy(truth = Exited, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.721
```

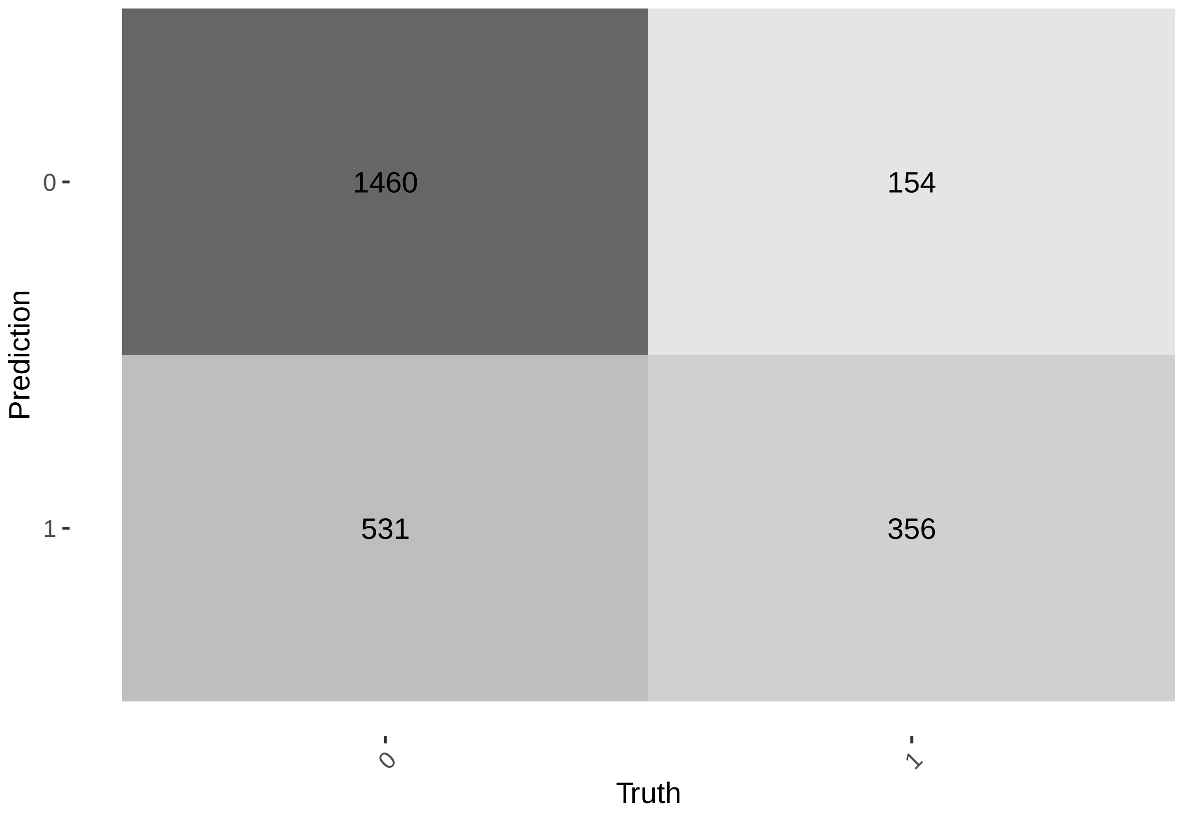
```
augment(log_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



```
augment(bank_enl_final, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.774
```

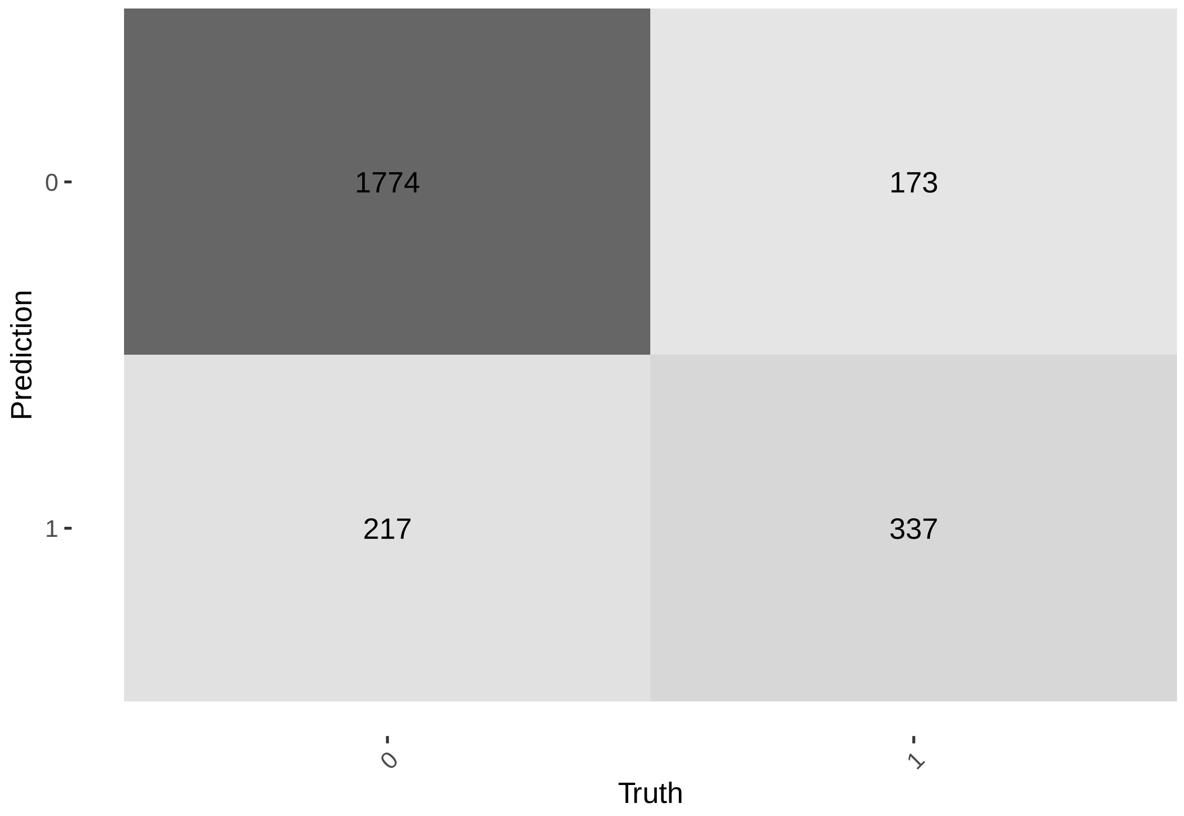
```
augment(bank_enl_final, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = eleme
```



```r
augment(bank_rf_fit, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.865
```

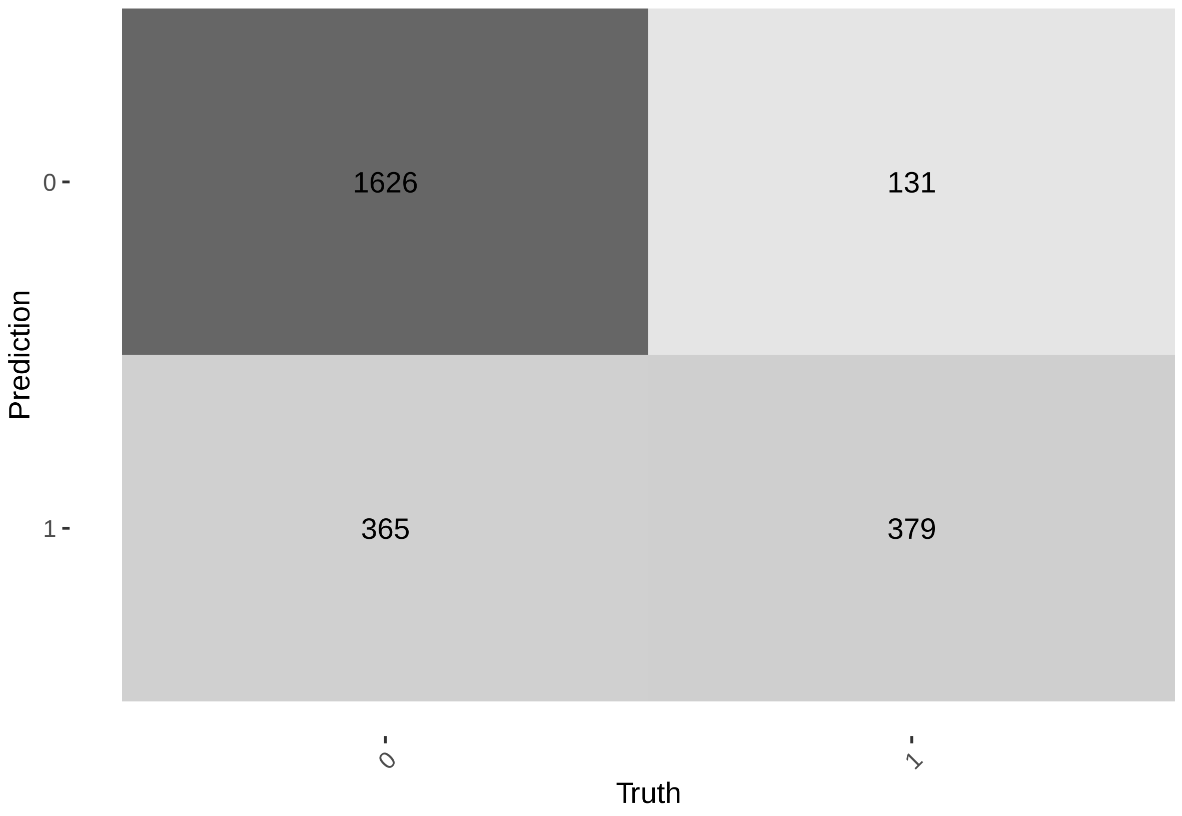
```r
augment(bank_rf_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



```
augment(bt_mode_fit, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.865
```

```
augment(bt_mode_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')+theme(axis.text.x = elemen
```



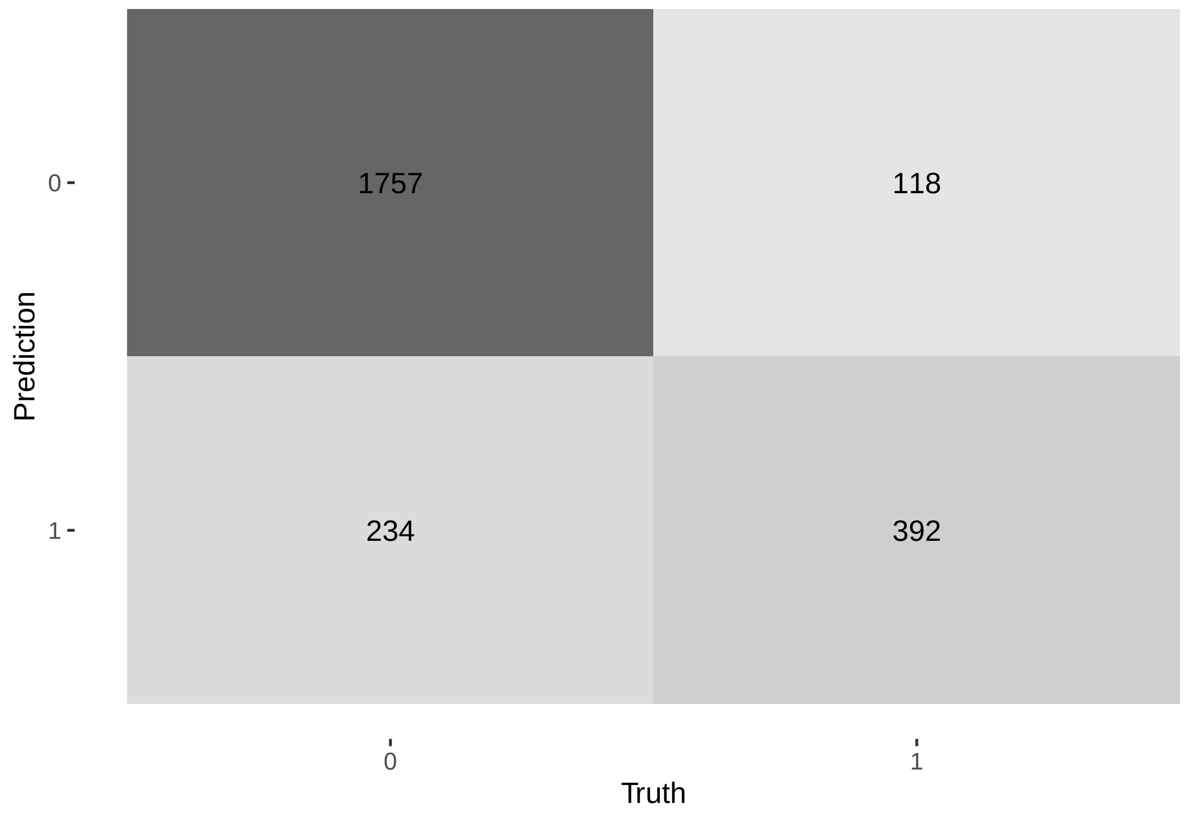
With each model, roc_auc increased, demonstrating the power of each model.Although the boosted tree model has the highest roc_auc, its confusion matrix did not do so well. Both ROC AUC and the confusion matrix are important metrics for evaluating the performance of a classification model, and since the random tree forest roc auc is basically just as good as the boosted tree one, I'm going to choose the random forest tree model.

Final Fit

```
final_final_fit <- fit(bt_mode_fit , data = bank_test)
augment(final_final_fit, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.915
```

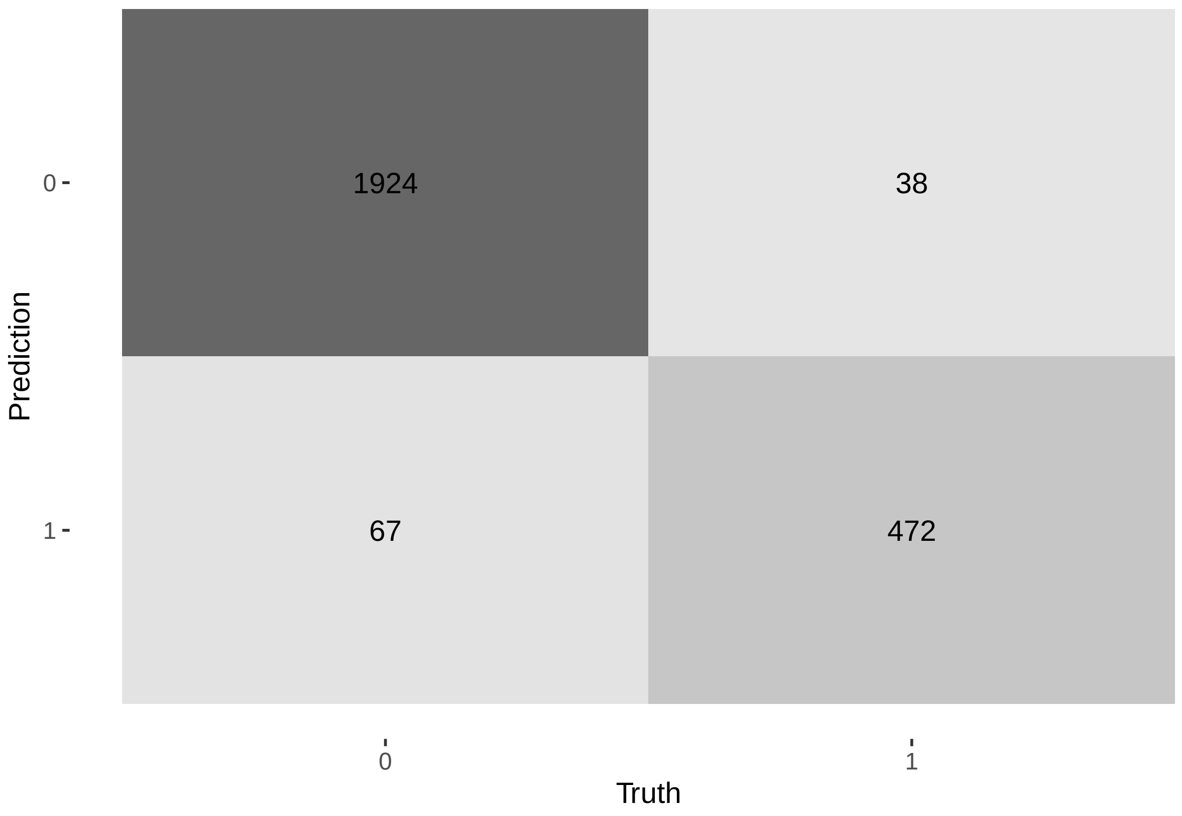
```
augment(final_final_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')
```



```r
final_final_fit <- fit(bank_rf_fit , data = bank_test)
augment(final_final_fit, new_data = bank_test) %>%
  roc_auc(truth = Exited, estimate = .pred_0)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.992
```

```r
augment(final_final_fit, new_data = bank_test) %>%
  conf_mat(truth = Exited, estimate = .pred_class)%>%autoplot(type='heatmap')
```



Conclusion

Just like I thought, random tree forest model did better thann the boosted tree forest model.

Conclusion

In conclusion, after comparing five different models, it was found that the random forest and boosted tree models performed the best in predicting whether a customer would stay or leave the bank. Although the knn model did really well on the training dataset (roc_auc of 98%) It didn't do as well with the testing dataset. In the case of the customer churn analysis, the Random Forest model was able to handle the combination of both categorical and numerical data to create multiple decision trees that accurately predict customer churn. It also reduced the chances of overfitting by constructing multiple trees with randomly selected features and subsets of data. Additionally, the model was able to improve accuracy by combining the predictions of all individual trees. Ultimately, the random forest tree model was chosen due to its high roc_auc of 99% (nearly 100!) and better performing confusion matrix. This model was able to accurately predict customer churn, with age being identified as the most significant predictor. Overall, this analysis provides valuable insights for the bank in understanding and addressing customer churn.