Predicting Customer Exit

2022-11-25

Predicting Customer Exit from a Bank

Introduction

Recently, machine learning a field under artificial intelligence has seen its significance proliferate. Simply put, machine learning is the capacity of machines to learn without explicit programming. Computers are able to learn through identifying patterns on which they are trained on before they are deployed for business they have been created in. Machine learning has a variety of applications, including finance. The various inventions made in the finance field and advent of mobile banking has seen the finance industry embrace technology and making work easier for their clients. However, with an increase in banking institutions and a bad economy banks face problems of clients leaving the bank or decrease in credit value. Hence the need to increase and or maintain the clientele they have and prepare for changes. The unpredictability in client behavior is causing banks problems. When an individual is confirmed to be leaving the bank, they can try to come up with strategies to retain them there longer because they are unclear of whether they will stay. These strategies include coming up with packages such as loans that can accommodate their various clients. The purpose of this study was to come up with a model that could predict a customer's exit.

Obectives

1. Come up with a model that can predict whether a client is exiting the company and/or whether they are leaving.
2. Determine which are the principle factors that determine if a client is leaving or staying.
3. Find patterns and insights on the clients who are staying or leaving using the data provided. #### Research Questions
4. Which kind of machine learning was used in the study? Supervised/Unsupervised?
5. Which was the best performing model, what was its accuracy?
6. Do the genders follow different patterns in banking (for those exiting or staying)?
7. How is the credit score of the banks of the banking customers?

Import data

The data was imported using tidyverse packages.

```
df = read_csv("Churn_Modelling.csv")

## Rows: 10000 Columns: 14
## ── Column specification
──────────────────────────────────────────────────────
```

```
## Delimiter: ","
## chr  (3): Surname, Geography, Gender
## dbl (11): RowNumber, CustomerId, CreditScore, Age, Tenure, Balance,
NumOfPro...
##
## ⓘ Use `spec()` to retrieve the full column specification for this data.
## ⓘ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
df %>% head()
```

```
## # A tibble: 6 × 14
##   RowNumber CustomerId Surname CreditScore Geography Gender   Age Tenure
Balance
##       <dbl>      <dbl> <chr>         <dbl> <chr>     <chr>  <dbl>  <dbl>
<dbl>
## 1         1   15634602 Hargra…         619 France    Female    42      2
0
## 2         2   15647311 Hill            608 Spain     Female    41      1
83808.
## 3         3   15619304 Onio            502 France    Female    42      8
159661.
## 4         4   15701354 Boni            699 France    Female    39      1
0
## 5         5   15737888 Mitche…         850 Spain     Female    43      2
125511.
## 6         6   15574012 Chu             645 Spain     Male      44      8
113756.
## # … with 5 more variables: NumOfProducts <dbl>, HasCrCard <dbl>,
## #   IsActiveMember <dbl>, EstimatedSalary <dbl>, Exited <dbl>
```

The columns Rownumber, customer surname and and customer Id this columns are unique, they are therefore not necessary for finding patterns in modelling or in data analysis.

```
df = df %>% select(-c(RowNumber, CustomerId, Surname))
df
```

```
## # A tibble: 10,000 × 11
##    CreditScore Geography Gender   Age Tenure Balance NumOfProducts
HasCrCard
##          <dbl> <chr>     <chr>  <dbl>  <dbl>   <dbl>         <dbl>
<dbl>
##  1         619 France    Female    42      2      0             1
1
##  2         608 Spain     Female    41      1  83808.            1
0
##  3         502 France    Female    42      8 159661.            3
1
##  4         699 France    Female    39      1      0             2
0
```

```
##  5         850 Spain     Female    43      2 125511.             1
1
##  6         645 Spain     Male      44      8 113756.             2
1
##  7         822 France    Male      50      7      0              2
1
##  8         376 Germany   Female    29      4 115047.             4
1
##  9         501 France    Male      44      4 142051.             2
0
## 10         684 France    Male      27      2 134604.             1
1
## # … with 9,990 more rows, and 3 more variables: IsActiveMember <dbl>,
## #   EstimatedSalary <dbl>, Exited <dbl>
```

There are 10000 records eleven columns, the response variable is_exited. The purpose of this study is to come up with a model that can predict whether a client that exits a bank or not based on the factors provided. Before proceeding with the exploratory data analysis, the data was split into training and testing sets.

Data Types

```
cols <- c("Gender", "HasCrCard", "IsActiveMember", "Exited", "Geography")

df %<>%
       mutate_each_(funs(factor(.)),cols)

## Warning: `mutate_each_()` was deprecated in dplyr 0.7.0.
## ℹ Please use `across()` instead.

## Warning: `funs()` was deprecated in dplyr 0.8.0.
## ℹ Please use a list of either functions or lambdas:
##
## # Simple named list: list(mean = mean, median = median)
##
## # Auto named with `tibble::lst()`: tibble::lst(mean, median)
##
## # Using lambdas list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))

str(df)

## tibble [10,000 × 11] (S3: tbl_df/tbl/data.frame)
##  $ CreditScore    : num [1:10000] 619 608 502 699 850 645 822 376 501 684
...
##  $ Geography      : Factor w/ 3 levels "France","Germany",..: 1 3 1 1 3 3
1 2 1 1 ...
##  $ Gender         : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 2 2 1 2
2 ...
##  $ Age            : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
##  $ Tenure         : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
##  $ Balance        : num [1:10000] 0 83808 159661 0 125511 ...
##  $ NumOfProducts  : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
```

```
##  $ HasCrCard      : Factor w/ 2 levels "0","1": 2 1 2 1 2 2 2 2 1 2 ...
##  $ IsActiveMember : Factor w/ 2 levels "0","1": 2 2 1 1 2 1 2 1 2 2 ...
##  $ EstimatedSalary: num [1:10000] 101349 112543 113932 93827 79084 ...
##  $ Exited         : Factor w/ 2 levels "0","1": 2 1 2 1 1 2 1 2 1 1 ...
```

Data Partitioning

```
set.seed(42)
df_split <- initial_split(df, prop = 0.7, strata = Exited)
df_train <- training(df_split)
df_test <- testing(df_split)


val_set <- validation_split(df_train, strata = Exited, prop = 0.7)
val_set
```

```
## # Validation Set Split (0.7/0.3)  using stratification
## # A tibble: 1 × 2
##   splits               id
##   <list>               <chr>
## 1 <split [4898/2101]> validation
```

```
nrow(df_train)
```

```
## [1] 6999
```

Significance of training and testing set(partitioning).

The quality of machine learning models depends on the training set's data. Even the most effective machine learning algorithms will not function well in the absence of high-quality training data. Early on in the training process, it becomes clear that relevant, full, accurate, and high-quality data are required. Only with sufficient training data can the algorithm quickly identify the features and discover the links required for future prediction. More specifically, the most important factor in machine learning (and artificial intelligence) is high-quality training data. The proper data must be used to train machine learning (ML) algorithms, which will then be more accurate and productive. There are 6999 records in the training set and 3001 records in the testing set.

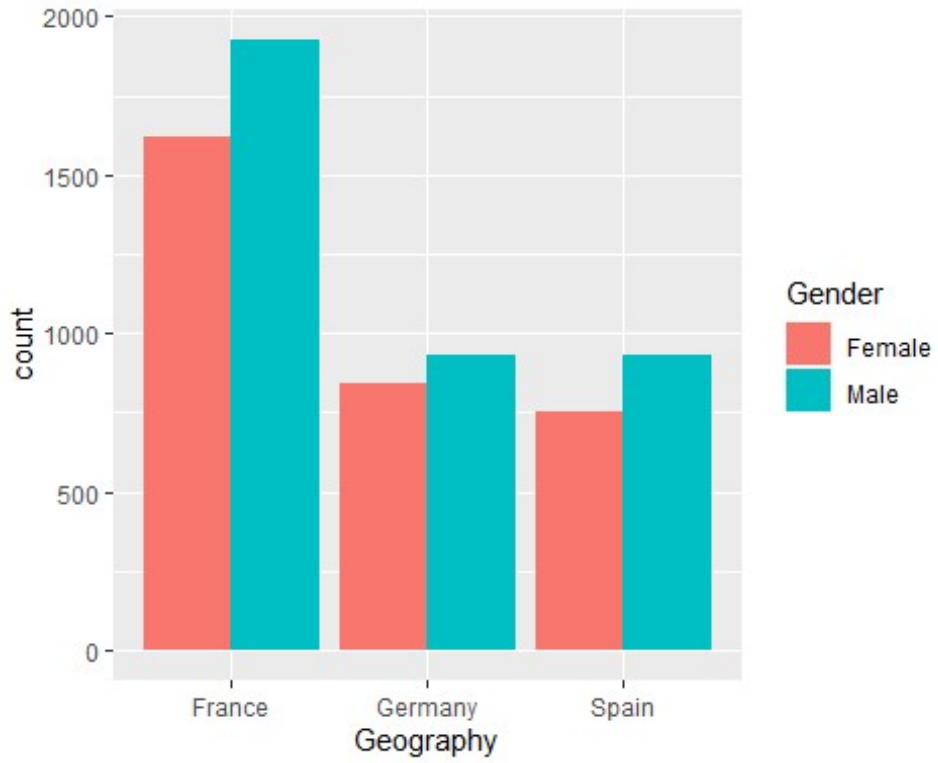
The terms training dataset, learning set, and training set are also used to refer to training data. Every machine learning model needs it since it enables them to accomplish desired tasks or generate correct predictions. Simply simply, the machine learning model is built using training data. It demonstrates what the desired result should look like. The model repeatedly studies the dataset to fully comprehend its characteristics and to modify itself for enhanced performance. Model training uses training data, which is data that is utilized to fit the model. On the other hand, test data are employed to assess the effectiveness or correctness of the model. It's a sample of data that is used to objectively assess how well the final model fit the training data. A training dataset is a starting set of data that teaches ML models how to recognize specific patterns or carry out a specific task. To assess how successful the training was or how accurate the model is, a testing dataset is used. An ML algorithm is more likely to have high accuracy if it has been trained on a specific dataset

and tested on that same dataset because the model is aware of what to anticipate. Everything will be fine if the training dataset includes every potential value that the model might meet in the future. However, it is never the case. There is no way that a training dataset could possibly cover everything that a model would face in the actual world. In order to assess the model's accuracy, a test dataset with obfuscated data points is employed.

```
nrow(df_test)

## [1] 3001
```

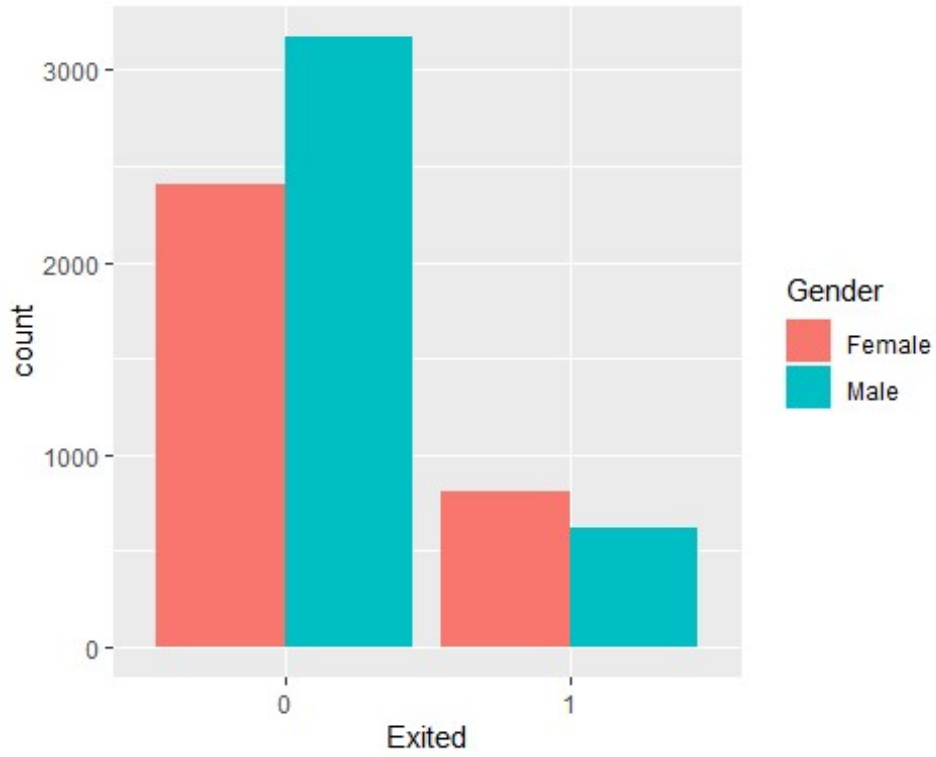
Exploratory Data Analysis

```
ggplot(df_train, aes(x = Geography, fill = Gender)) +
    geom_bar(position = "dodge")
```



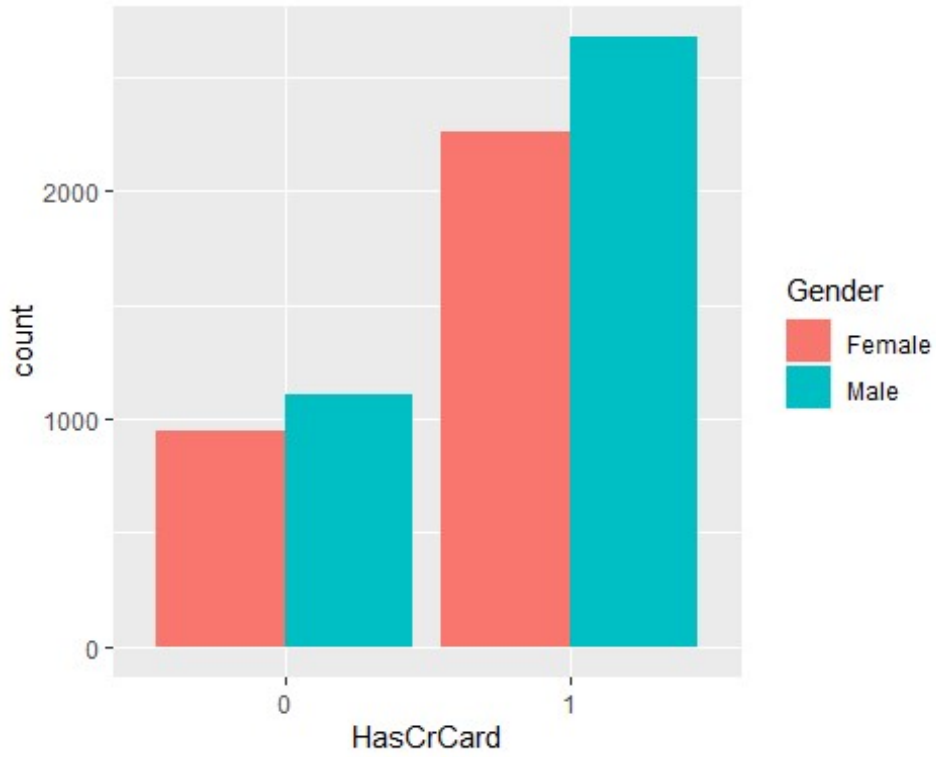
Germany and Spain Branches have the same number of customers and France has the highest number of clients. In all the countries the number of men is slightly higher than women. Gender is a significant factor in determining the bank's performance. Men and women have different spending, saving and investing patterns. Therefore, the bank has to prepare different packages for each individual.Understanding the patterns according to gender are therefore crucial to the banks future planning.

```
ggplot(df_train, aes(x = Exited, fill = Gender)) +
    geom_bar(position = "dodge")
```



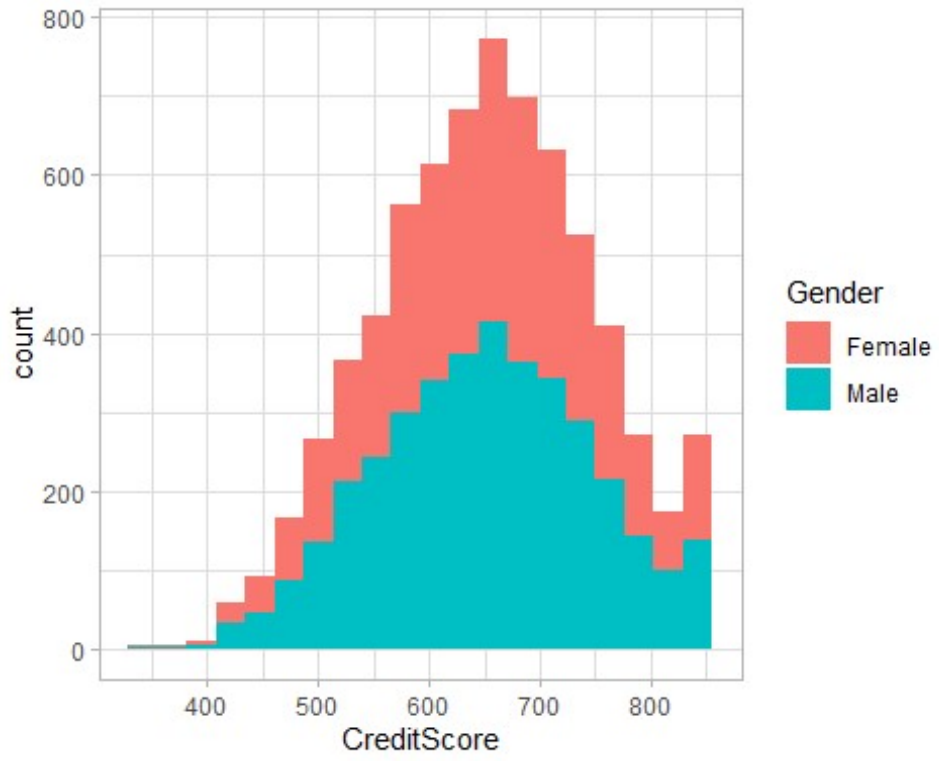
The proportion of males to females who exit is higher while those who do not exit, the proportion of females is higher.

```
ggplot(df_train, aes(x = HasCrCard, fill = Gender)) +
    geom_bar(position = "dodge")
```



Most customer do not have a credit card, and among those that do, males are more.

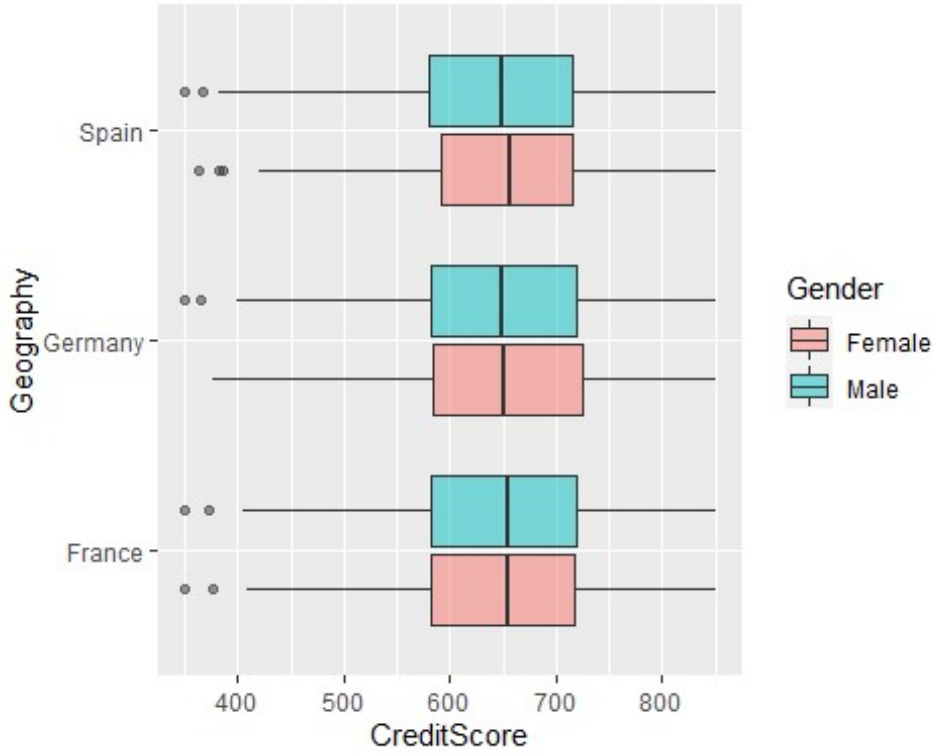
```
ggplot(df_train, aes(CreditScore, fill = Gender)) +
  geom_histogram(bins = 20)+
  theme_light()
```



Most of the employees have a credit score thats between 550 and 750. This shows that the bank has a mixture of clients with poor credit to those who have good credit. Note that Men have a lower credit score than women. This could be attributed to men being providers therefore have a history of borrowing compared to women.

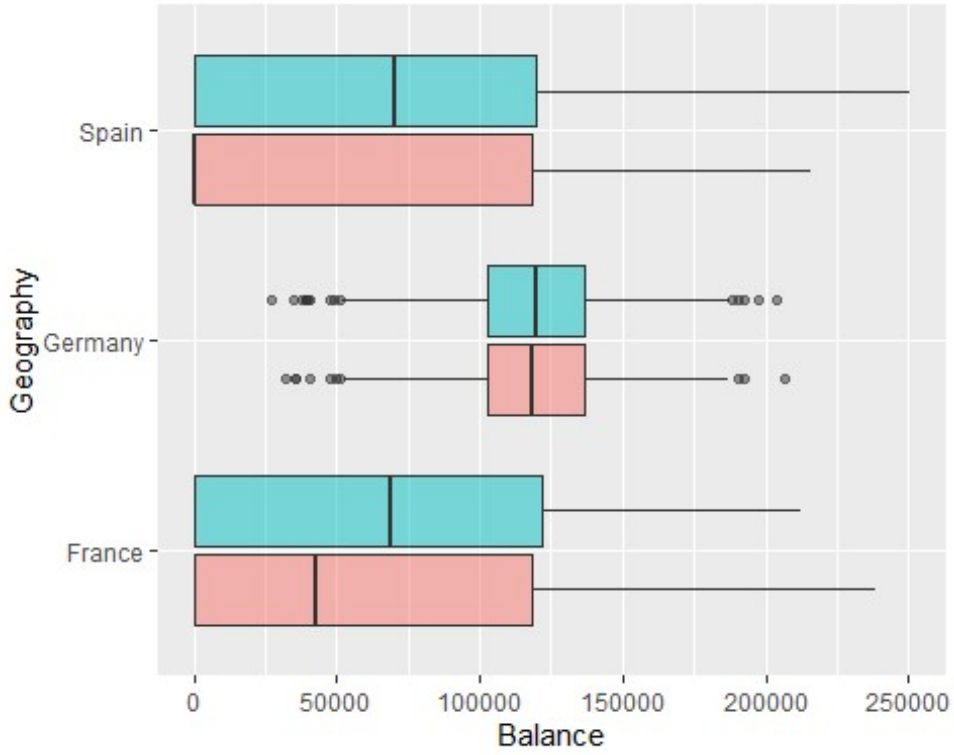
From the plot below we note that creditscore for all countries has almost the same range. However, note that in Spain and Germany the credit score range for men is lower than females.

```
ggplot(df_train, aes(CreditScore, Geography, fill = Gender)) +
  geom_boxplot(alpha = 0.5, show.legend = TRUE)
```

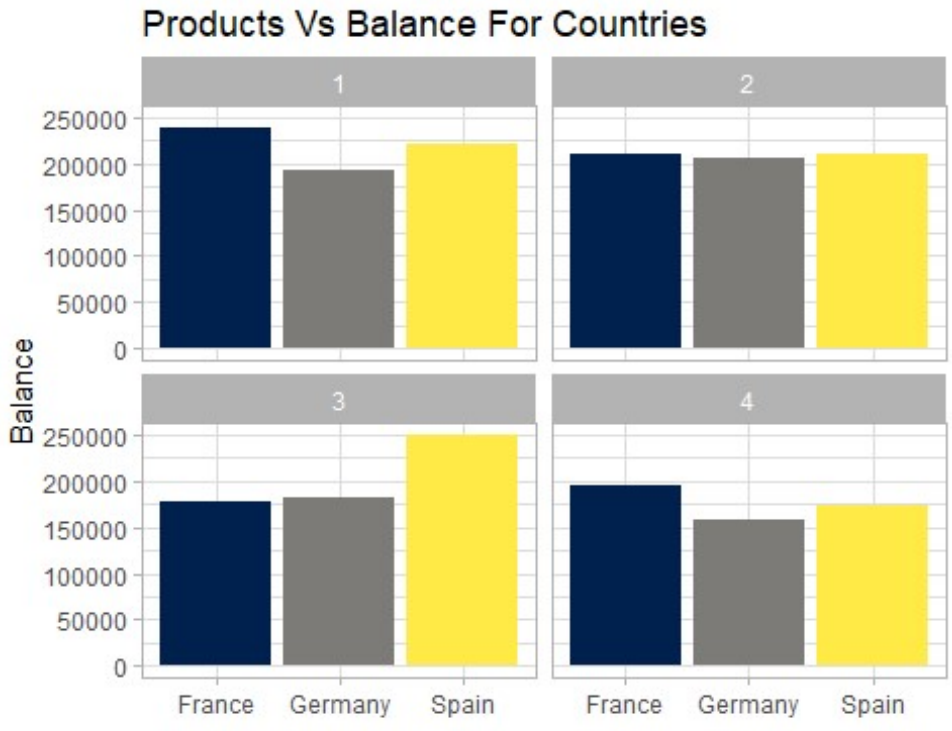


The bank customers in Germany are have bank balances in their accounts compared to spain and France which suggests that the banks customers in Germany are more high profile than in the other countries. However, note that there are no differences when it comes to gender for both countries.

```
ggplot(df_train, aes(Balance, Geography, fill = Gender)) +
  geom_boxplot(alpha = 0.5, show.legend = FALSE)
```




```
ggplot(df_train, aes(fill=Geography, y=Balance, x=Geography)) +
    geom_bar(position="dodge", stat="identity") +
    scale_fill_viridis(discrete = T, option = "E") +
    ggtitle("Products Vs Balance For Countries") +
    facet_wrap(~NumOfProducts) +
    theme_light() +
    theme(legend.position="none") +
    xlab("")
```



Summary

Statistics,

```
df_train %>%
  skimr::skim()
```

Data summary

Name	Piped data
Number of rows	6999
Number of columns	11
_____	
Column type frequency:	
factor	5
numeric	6
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Geography	0	1	FALSE	3	Fra: 3545, Ger: 1773, Spa: 1681
Gender	0	1	FALSE	2	Mal: 3788, Fem: 3211

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
HasCrCard	0	1	FALSE	2	1: 4940, 0: 2059
IsActiveMember	0	1	FALSE	2	1: 3612, 0: 3387
Exited	0	1	FALSE	2	0: 5574, 1: 1425

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
CreditScore	0	1	651.05	96.98	350.00	584.00	653.0	718.0	850.0	▁▁▆▇▇
Age	0	1	38.93	10.47	18.00	32.00	37.0	44.0	92.0	▆▇▃▁▁
Tenure	0	1	5.01	2.89	0.00	3.00	5.0	7.0	10.0	▆▆▇▆▅
Balance	0	1	75767.49	62171.56	0.00	0.00	96341.0	127041.2	250898.1	▇▁▁▆▁
NumOfProducts	0	1	1.53	0.58	1.00	1.00	1.0	2.0	4.0	▇▁▃▁▁
EstimatedSalary	0	1	100293.44	57337.75	11.58	51637.18	100919.2	149185.6	199970.7	▇▇▇▇▇

Crossvalidations:

Crossvalidation splits the training set into multiple sets which it trains on. The purpose of this dataset is to improve the model's perfomance on unseen data. In this study cross-validation was implemented using the vfold_cv function. The training set would be split into three sets on which the model will be trained on.

This is a very fundamental and straightforward method in which we split our entire dataset into training data and testing data. We train the model using training data, and then we test it using testing data. The data is divided 70:30 because training data are often configured to be larger than testing data by a factor of two. In this method, the data is separated after being first randomly shuffled. Every time the model is trained, it can produce different results because it was trained on a different set of data points. This can lead to instability. Furthermore, we can never be certain that the train set we choose is an accurate representation of the entire dataset.

One strategy to enhance the holdout method is K-fold cross validation. This approach ensures that our model's score is independent of how we chose the train and test sets. The holdout approach is done k times after dividing the data set into k sections. Let's proceed in the following order:

1. Divide your dataset into k number of folds at random (subsets)

2 .Build your model using k - 1 folds of the dataset for each fold in your dataset. Then, test the model to see if it works for the kth fold.

3. Continue doing this until all k-folds have served as the test set.

4. The cross-validation accuracy, which is defined as the average of your k recorded accuracy, will be used as your model's performance metric.

```
set.seed(42)
churn_folds <- df_train %>%
  vfold_cv(v = 3, repeats = 1, strata = Exited)
churn_folds
```

```
## #  3-fold cross-validation using stratification
## # A tibble: 3 × 2
##   splits              id
##   <list>              <chr>
## 1 <split [4666/2333]> Fold1
## 2 <split [4666/2333]> Fold2
## 3 <split [4666/2333]> Fold3
```

The following is a glimpse of the split training set.

```
churn_folds %>% purrr::pluck("splits", 1) %>% training()
```

```
## # A tibble: 4,666 × 11
##    CreditScore Geography Gender   Age Tenure Balance NumOfProducts
HasCrCard
##          <dbl> <fct>     <fct>  <dbl>  <dbl>   <dbl>         <dbl> <fct>
##  1         608 Spain     Female    41      1  83808.             1 0
##  2         850 Spain     Female    43      2 125511.             1 1
##  3         822 France    Male      50      7      0              2 1
##  4         528 France    Male      31      6 102017.             2 0
##  5         549 France    Female    25      5      0              2 0
##  6         587 Spain     Male      45      6      0              1 0
##  7         726 France    Female    24      6      0              2 1
##  8         732 France    Male      41      8      0              2 1
##  9         846 France    Female    38      5      0              1 1
## 10         574 Germany   Female    43      3 141349.             1 1
## # … with 4,656 more rows, and 3 more variables: IsActiveMember <fct>,
## #   EstimatedSalary <dbl>, Exited <fct>
```

Modelling

Data Preprocessing

The following are pre-processing procedures that were undertaken on the data.

Normalization

Due to the varying scales of the numeric data, it was best to normalize it; reduce the data to range from 0 to 1.The process of normalization is frequently used to prepare data for machine learning. The purpose of normalization is to convert the values of the dataset's numeric columns to a common scale without distorting variations in the value ranges. Every dataset does not need to be normalized for machine learning. Only when features have various ranges is it necessary. In this case variables like age, EstimatedSalary, number of products, balance, tenure and age.

Creating Dummies and One Hot Encoding

Dummy variables are qualitative or discrete variables that represent category data and can have values of 0 or 1 to denote the absence or existence of a certain property, respectively. To increase prediction accuracy, one-hot encoding in machine learning involves transforming categorical data into a format that can be used by machine learning algorithms. With this method, a new column is made for each distinct value in the initial category column. These fake variables are then filled with zeros and ones (1 meaning TRUE, 0 meaning FALSE). This approach can result in a significant issue (too many predictors) if the original column contains a lot of unique values because it creates a lot of additional variables.One-hot encoding also has the drawback of increasing multicollinearity among the numerous variables, which reduces the model's accuracy.

```
mod_recipe <- recipe(formula = Exited ~ ., data = df_train)
mod_recipe <- mod_recipe %>% step_impute_median(Age)%>%
   step_normalize(c(Age, CreditScore, Tenure, Balance, NumOfProducts,
EstimatedSalary)) %>%
    step_dummy(all_predictors(), -all_numeric(), one_hot = T)
mod_recipe <- mod_recipe %>% themis::step_upsample(Exited, over_ratio = 0.2,
seed = 42, skip = T)

churn_prep <- prep(mod_recipe)
juiced <- juice(churn_prep)
```

Model training

The machine learning approach in this case is supervised machine learning approach. It is distinguished by the way it trains computers to accurately classify data or predict outcomes using labeled datasets. The model modifies its weights as input data is fed into it until the model has been properly fitted, which takes place as part of the cross validation process. A training set is used in supervised learning to instruct models to produce the desired results. This training dataset has both the right inputs and outputs, enabling the model to develop over time. The loss function serves as a gauge for the algorithm's correctness, and iterations are made until the error is sufficiently reduced.

When using data mining, supervised learning may be divided into two sorts of issues: classification and regression. Classification is the approach in this case. In order to accurately classify test data into different categories, classification uses an algorithm. It identifies particular entities in the dataset and makes an effort to determine how those

things should be defined or labeled. The classification techniques that we are training below are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forests.

The following models were trained on the data, and the best performing was set to be selected. The models selected for this study were: 1. Logistic Regression, 2. Decision Trees, 3. Support Vector Machines and
4. Random Forest model. 5. kNN

1.Linear Classifiers (Logistic Regression)

```
log_cls <- logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glm") %>%
  set_mode("classification")
log_cls
```

```
## Logistic Regression Model Specification (classification)
##
## Main Arguments:
##   penalty = tune()
##   mixture = 1
##
## Computational engine: glm
```

```
churn_fit_log <-
  workflow(Exited ~ ., log_cls) %>%
  fit(data = df_train)

churn_fit_log
```

```
## == Workflow [trained]
═══════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: logistic_reg()
##
## ── Preprocessor
──────────────────────────────────────────────────────────────
## Exited ~ .
##
## ── Model
──────────────────────────────────────────────────────────────────
##
## Call:  stats::glm(formula = ..y ~ ., family = stats::binomial, data =
data)
##
## Coefficients:
##      (Intercept)       CreditScore  GeographyGermany    GeographySpain
##       -3.176e+00        -6.962e-04         7.395e-01         2.817e-02
##       GenderMale               Age            Tenure           Balance
##       -5.670e-01         7.097e-02        -2.080e-02         2.256e-06
##    NumOfProducts        HasCrCard1   IsActiveMember1   EstimatedSalary
```

```
##       -1.310e-01        -1.509e-02        -1.108e+00         5.144e-07
##
## Degrees of Freedom: 6998 Total (i.e. Null);  6987 Residual
## Null Deviance:       7074
## Residual Deviance: 6037  AIC: 6061
```

```
augment(churn_fit_log, new_data = df_test) %>%
  metrics(Exited, .pred_class)
```

```
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.815
## 2 kap      binary         0.256
```

Tuning Model

```
set.seed(42)
lr_mod <-
  logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

tune_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(lr_mod)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

tune_res %>%
  collect_metrics()
```

```
## # A tibble: 40 × 8
##     penalty mixture .metric  .estimator  mean     n  std_err .config
##       <dbl>   <dbl> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1 5.10e- 2  0.0884 accuracy binary     0.805     3 0.000892
Preprocessor1_Mode…
##  2 5.10e- 2  0.0884 roc_auc  binary     0.758     3 0.00153
Preprocessor1_Mode…
##  3 5.59e-10  0.122  accuracy binary     0.813     3 0.00401
Preprocessor1_Mode…
##  4 5.59e-10  0.122  roc_auc  binary     0.758     3 0.00196
Preprocessor1_Mode…
##  5 4.14e- 5  0.183  accuracy binary     0.813     3 0.00409
Preprocessor1_Mode…
##  6 4.14e- 5  0.183  roc_auc  binary     0.758     3 0.00195
```

```
Preprocessor1_Mode…
##  7 1.02e- 1  0.206  accuracy binary     0.797     3 0.000286
Preprocessor1_Mode…
##  8 1.02e- 1  0.206  roc_auc  binary     0.759     3 0.00270
Preprocessor1_Mode…
##  9 1.23e- 5  0.264  accuracy binary     0.813     3 0.00401
Preprocessor1_Mode…
## 10 1.23e- 5  0.264  roc_auc  binary     0.758     3 0.00194
Preprocessor1_Mode…
## # … with 30 more rows
```

The average accuracy of the logistic regression model did not change even after tuning the penalty and mixture parameters, the highest accuracy obtained is 81.28%.

2.K Nearest Neighbors

```
knn_cls <- nearest_neighbor() %>%
  set_mode("classification")
knn_cls
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Computational engine: kknn
```

```
churn_fit_knn <-
  workflow(Exited ~ ., knn_cls) %>%
  fit(data = df_train)

churn_fit_knn
```

```
## == Workflow [trained]
═══════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: nearest_neighbor()
##
## ── Preprocessor
──────────────────────────────────────────────────────────────
## Exited ~ .
##
## ── Model
──────────────────────────────────────────────────────────────────
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(5,
data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.1800257
## Best kernel: optimal
## Best k: 5
```

```
augment(churn_fit_knn, new_data = df_test) %>%
  metrics(Exited, .pred_class)
```

```
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.818
## 2 kap      binary         0.375
```

Tuning Model

```
set.seed(42)
knn_cls <- nearest_neighbor(neighbors = tune(), weight_func = tune(),
dist_power = tune()) %>%
  set_mode("classification")
knn_cls
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = tune()
##   weight_func = tune()
##   dist_power = tune()
##
## Computational engine: kknn
```

```
tune_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(knn_cls)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

tune_res %>%
  collect_metrics()
```

```
## # A tibble: 40 × 9
##    neighbors weight_func  dist_power .metric  .estimator  mean     n
std_err
##        <int> <chr>             <dbl> <chr>    <chr>      <dbl> <int>
<dbl>
##  1         6 biweight          0.483 accuracy binary     0.786     3
0.00216
##  2         6 biweight          0.483 roc_auc  binary     0.724     3
0.00455
##  3         1 biweight          1.24  accuracy binary     0.779     3
0.00213
```

```
##  4         1 biweight          1.24  roc_auc  binary     0.642     3
0.00410
##  5        12 cos               0.368 accuracy binary     0.807     3
0.00151
##  6        12 cos               0.368 roc_auc  binary     0.751     3
0.00544
##  7         4 cos               0.952 accuracy binary     0.787     3
0.00325
##  8         4 cos               0.952 roc_auc  binary     0.715     3
0.00459
##  9         8 epanechnikov      0.807 accuracy binary     0.802     3
0.00127
## 10         8 epanechnikov      0.807 roc_auc  binary     0.743     3
0.00722
## # … with 30 more rows, and 1 more variable: .config <chr>
```

The highest accuracy of the KNN model after tuning is 81.2%. All parameteres were tuned.

3. Decision Trees

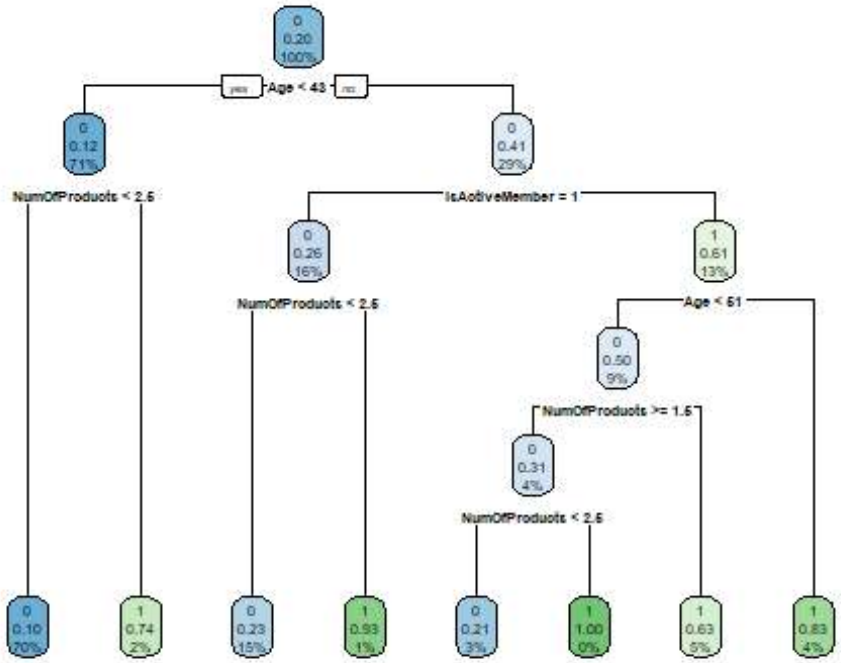
```
dt_cls <- decision_tree() %>%
  set_args(cost_complexity = 0.01, tree_depth = 30, min_n = 20) %>%
  set_mode("classification") %>%
  set_engine("rpart")
dt_cls
```

```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0.01
##   tree_depth = 30
##   min_n = 20
##
## Computational engine: rpart
```

```
churn_fit_dt <-
  workflow(Exited ~ ., dt_cls) %>%
  fit(data = df_train)

churn_fit_dt
```

```
## == Workflow [trained]
═══════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: decision_tree()
##
## ── Preprocessor
──────────────────────────────────────────────────────────────
## Exited ~ .
##
## ── Model
──────────────────────────────────────────────────────────────
```

```
## n= 6999
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 6999 1425 0 (0.79639949 0.20360051)
##    2) Age< 42.5 4987  595 0 (0.88068979 0.11931021)
##      4) NumOfProducts< 2.5 4867  506 0 (0.89603452 0.10396548) *
##      5) NumOfProducts>=2.5 120   31 1 (0.25833333 0.74166667) *
##    3) Age>=42.5 2012  830 0 (0.58747515 0.41252485)
##      6) IsActiveMember=1 1127  292 0 (0.74090506 0.25909494)
##       12) NumOfProducts< 2.5 1082  250 0 (0.76894640 0.23105360) *
##       13) NumOfProducts>=2.5 45    3 1 (0.06666667 0.93333333) *
##      7) IsActiveMember=0 885  347 1 (0.39209040 0.60790960)
##       14) Age< 50.5 597  298 0 (0.50083752 0.49916248)
##         28) NumOfProducts>=1.5 248   78 0 (0.68548387 0.31451613)
##           56) NumOfProducts< 2.5 214   44 0 (0.79439252 0.20560748) *
##           57) NumOfProducts>=2.5 34    0 1 (0.00000000 1.00000000) *
##         29) NumOfProducts< 1.5 349  129 1 (0.36962751 0.63037249) *
##       15) Age>=50.5 288   48 1 (0.16666667 0.83333333) *

library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.2.2

## Loading required package: rpart

## Warning: package 'rpart' was built under R version 4.2.2

##
## Attaching package: 'rpart'

## The following object is masked from 'package:dials':
##
##     prune

churn_fit_dt %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE)
```



Model Performance

```
augment(churn_fit_dt, new_data = df_test) %>%
  metrics(Exited, .pred_class)
```

```
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.852
## 2 kap      binary         0.475
```

Tuning Model

```
set.seed(42)
dt_cls <- decision_tree(tree_depth = tune(), min_n = tune(), cost_complexity
= tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification") %>%
  translate()
dt_cls
```

```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = tune()
##   tree_depth = tune()
##   min_n = tune()
##
## Computational engine: rpart
```

```
##
## Model fit template:
## rpart::rpart(formula = missing_arg(), data = missing_arg(), weights =
missing_arg(),
##     cp = tune(), maxdepth = tune(), minsplit = min_rows(tune(),
##         data))
```

```
tune_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(dt_cls)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

tune_res %>%
  collect_metrics()
```

```
## # A tibble: 40 × 9
##    cost_complexity tree_depth min_n .metric  .estimator  mean     n
std_err
##              <dbl>      <int> <int> <chr>    <chr>      <dbl> <int>
<dbl>
##  1     0.0151               3    14 accuracy binary     0.838     3
0.00410
##  2     0.0151               3    14 roc_auc  binary     0.735     3 0.0106
##  3     0.0000277           10    13 accuracy binary     0.824     3
0.00276
##  4     0.0000277           10    13 roc_auc  binary     0.783     3
0.00788
##  5     0.000360             3     8 accuracy binary     0.836     3
0.00216
##  6     0.000360             3     8 roc_auc  binary     0.735     3 0.0104
##  7     0.0000108           13    30 accuracy binary     0.828     3
0.00540
##  8     0.0000108           13    30 roc_auc  binary     0.813     3
0.00687
##  9     0.000000180          6    10 accuracy binary     0.845     3
0.00100
## 10     0.000000180          6    10 roc_auc  binary     0.809     3 0.0121
## # … with 30 more rows, and 1 more variable: .config <chr>
```

The highest accuracy of the model obtained from tuning was 85.3% .

4. Random Forest

```
random_forest = rand_forest() %>%
  set_engine("randomForest") %>%
  set_mode("classification") %>%
  translate()
churn_fit_rf =
  workflow(Exited ~ ., random_forest) %>%
  fit(data = df_train)

churn_fit_rf
```

```
## == Workflow [trained]
## ═══════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: rand_forest()
##
## ── Preprocessor
## ──────────────────────────────────────────────────────────
## Exited ~ .
##
## ── Model
## ──────────────────────────────────────────────────────────
##
## Call:
##  randomForest(x = maybe_data_frame(x), y = y)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.67%
## Confusion matrix:
##      0   1 class.error
## 0 5355 219  0.03928956
## 1  738 687  0.51789474
```

```
augment(churn_fit_rf, new_data = df_test) %>%
  metrics(Exited, .pred_class)
```

```
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.863
## 2 kap      binary         0.514
```

Tuning Model

```
set.seed(42)
rf_cls <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification") %>%
  translate()
```

```
tune_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(rf_cls)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

## i Creating pre-processing data to finalize unknown parameter: mtry

tune_res %>%
  collect_metrics()

## # A tibble: 40 × 9
##     mtry trees min_n .metric  .estimator  mean     n  std_err .config
##    <int> <int> <int> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1    14   351    14 accuracy binary     0.853     3 0.000990
Preprocessor1_Mod…
##  2    14   351    14 roc_auc  binary     0.832     3 0.00264
Preprocessor1_Mod…
##  3     9  1290    13 accuracy binary     0.856     3 0.000857
Preprocessor1_Mod…
##  4     9  1290    13 roc_auc  binary     0.840     3 0.00248
Preprocessor1_Mod…
##  5    11   295     8 accuracy binary     0.853     3 0.00112
Preprocessor1_Mod…
##  6    11   295     8 roc_auc  binary     0.835     3 0.00261
Preprocessor1_Mod…
##  7     9  1694    30 accuracy binary     0.856     3 0.00138
Preprocessor1_Mod…
##  8     9  1694    30 roc_auc  binary     0.842     3 0.00282
Preprocessor1_Mod…
##  9     6   681    10 accuracy binary     0.856     3 0.000655
Preprocessor1_Mod…
## 10     6   681    10 roc_auc  binary     0.842     3 0.00254
Preprocessor1_Mod…
## # … with 30 more rows
```

The highest accuracy obtained after tuning the random forest model is 85.7%.

5. Support Vector Machine(SVM)

```
SVM = svm_poly() %>%
  set_engine("kernlab") %>%
  set_mode("classification")
```

```
churn_fit_SVM =
  workflow(Exited ~ ., SVM) %>%
  fit(data = df_train)
```

```
##  Setting default kernel parameters
## maximum number of iterations reached 0.01362619 0.01122108
```

```
churn_fit_SVM
```

```
## == Workflow [trained]
═══════════════════════════════════════════════════════════
## Preprocessor: Formula
## Model: svm_poly()
##
## ── Preprocessor
──────────────────────────────────────────────────────────────
## Exited ~ .
##
## ── Model
──────────────────────────────────────────────────────────────────
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 3066
##
## Objective Function Value : -2850
## Training error : 0.203601
## Probability model included.
```

```
augment(churn_fit_SVM, new_data = df_test) %>%
  metrics(Exited, .pred_class)
```

```
## # A tibble: 2 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.796
## 2 kap      binary         0
```

Tuning Model

```
set.seed(42)
svm_cls <- svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification") %>%
  translate()

tune_wf <- workflow() %>%
```

```
  add_recipe(mod_recipe) %>%
  add_model(svm_cls)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

tune_res %>%
  collect_metrics()
```

```
## # A tibble: 40 × 8
##       cost     rbf_sigma .metric  .estimator  mean     n std_err .config
##      <dbl>         <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.0144  0.00177       accuracy binary     0.796     3 0
Preprocessor1_…
##  2 0.0144  0.00177       roc_auc  binary     0.768     3 0.00693
Preprocessor1_…
##  3 0.525   0.0000884     accuracy binary     0.796     3 0
Preprocessor1_…
##  4 0.525   0.0000884     roc_auc  binary     0.736     3 0.0117
Preprocessor1_…
##  5 0.00624 0.000000930   accuracy binary     0.796     3 0
Preprocessor1_…
##  6 0.00624 0.000000930   roc_auc  binary     0.710     3 0.0120
Preprocessor1_…
##  7 0.0244  0.0295        accuracy binary     0.796     3 0
Preprocessor1_…
##  8 0.0244  0.0295        roc_auc  binary     0.807     3 0.00507
Preprocessor1_…
##  9 0.336   0.00000000251 accuracy binary     0.796     3 0
Preprocessor1_…
## 10 0.336   0.00000000251 roc_auc  binary     0.745     3 0.00464
Preprocessor1_…
## # … with 30 more rows
```

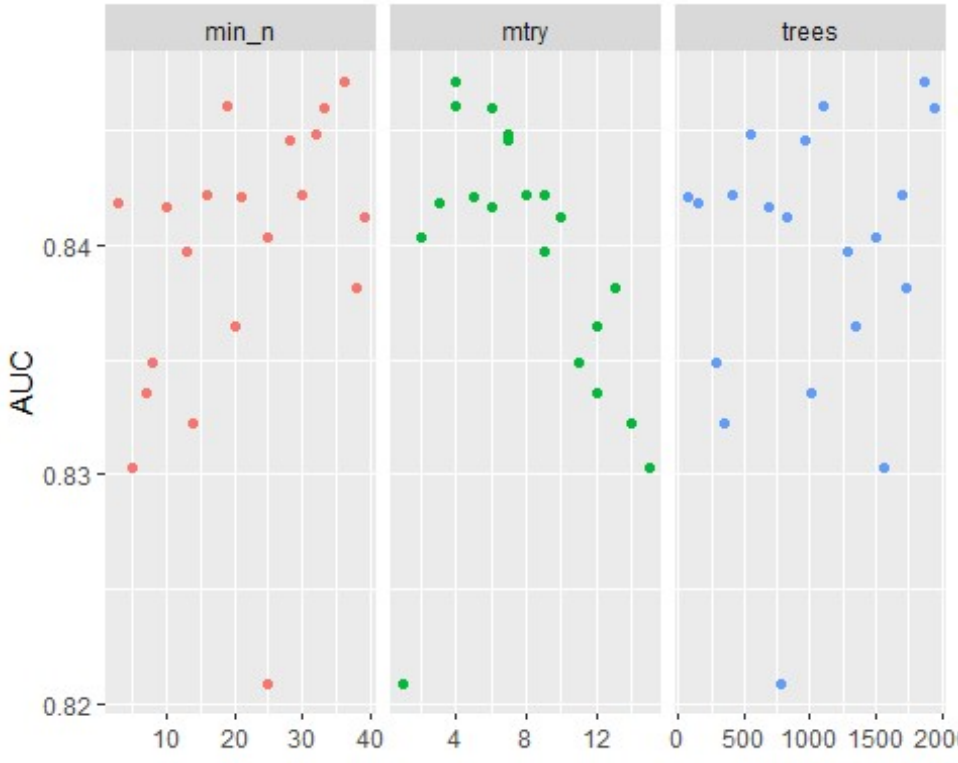
The highest accuracy obtained by the Support Vector Machine Model after training is 81.84%.

Final Model (Random Forests)

The highest performing model was the Random Forests model. We therefore fit it, train and tune it to get the best values of its parameters and then test on test data.

```
set.seed(42)
rf_cls <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine("ranger") %>%
```

```
  set_mode("classification") %>%
  translate()

tune_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(rf_cls)

doParallel::registerDoParallel()

tune_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = 20
)

## i Creating pre-processing data to finalize unknown parameter: mtry

tune_res %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  select(mean, min_n, mtry, trees) %>%
  pivot_longer(min_n:mtry:trees,
    values_to = "value",
    names_to = "parameter"
  ) %>%
  ggplot(aes(value, mean, color = parameter)) +
  geom_point(show.legend = FALSE) +
  facet_wrap(~parameter, scales = "free_x") +
  labs(x = NULL, y = "AUC")

## Warning in x:y: numerical expression has 2 elements: only the first used
```



Although not all possible min n and mtry combinations were used in this grid, we can still make sense of what is happening. It appears that lower values of mtry (4 and 6) and values of min n are that are advantageous are (between 30 and 40). We can tune once more, this time using regular grid, to better understand the hyperparameters. Based on the outcomes of our initial tuning, let's establish ranges of hyperparameters we wish to test.

The model is tuned again.

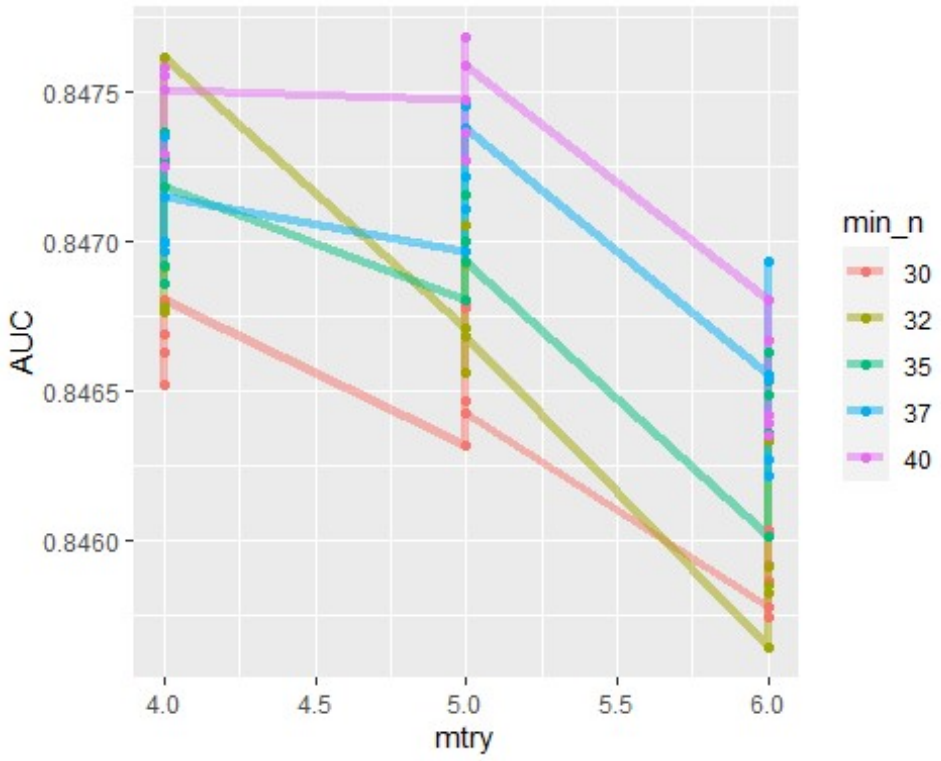
```
rf_grid <- grid_regular(
  trees(range =c(1750, 2000)),
  mtry(c(4, 6)),
  min_n(range = c(30, 40)),
  levels = 5
)

rf_grid
```

```
## # A tibble: 75 × 3
##    trees  mtry min_n
##    <int> <int> <int>
##  1  1750     4    30
##  2  1812     4    30
##  3  1875     4    30
##  4  1937     4    30
##  5  2000     4    30
##  6  1750     5    30
##  7  1812     5    30
```

```
##  8  1875     5    30
##  9  1937     5    30
## 10  2000     5    30
## # … with 65 more rows

set.seed(42)
regular_res <- tune_grid(
  tune_wf,
  resamples = churn_folds,
  grid = rf_grid)

regular_res

## # Tuning results
## # 3-fold cross-validation using stratification
## # A tibble: 3 × 4
##   splits               id    .metrics           .notes
##   <list>               <chr> <list>             <list>
## 1 <split [4666/2333]> Fold1 <tibble [150 × 7]> <tibble [0 × 3]>
## 2 <split [4666/2333]> Fold2 <tibble [150 × 7]> <tibble [0 × 3]>
## 3 <split [4666/2333]> Fold3 <tibble [150 × 7]> <tibble [0 × 3]>
```

Here are the results.

```
regular_res %>%
  collect_metrics() %>%
  filter(.metric == "roc_auc") %>%
  mutate(min_n = factor(min_n)) %>%
  ggplot(aes(mtry, mean, color = min_n)) +
  geom_line(alpha = 0.5, size = 1.5) +
  geom_point() +
  labs(y = "AUC")
```

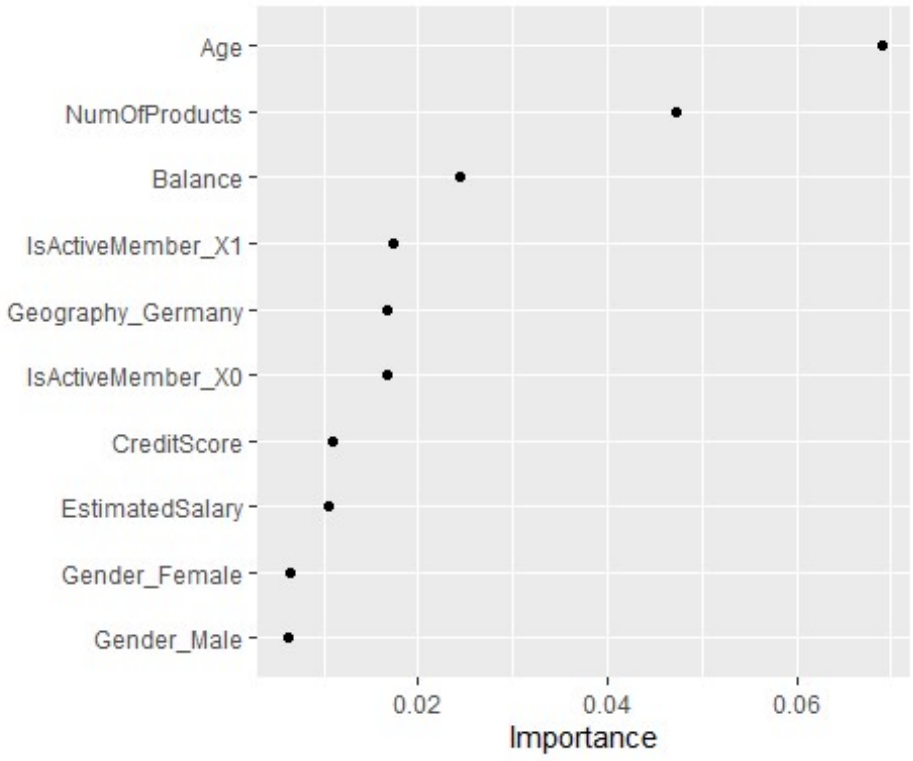



```
best_auc <- select_best(regular_res, "roc_auc")

final_rf <- finalize_model(
  rf_cls,
  best_auc)

final_rf
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 5
##   trees = 1937
##   min_n = 40
##
## Computational engine: ranger
```

The best model requires 1937 trees, 5 mtries and min_n OF 40.

```
final_rf %>%
  set_engine("ranger", importance = "permutation") %>%
  fit(Exited ~ .,
    data = juice(churn_prep))  %>%
  vip(geom = "point")
```



The most factors that explain the clients exit are Age, Number of products, is active member and balance.

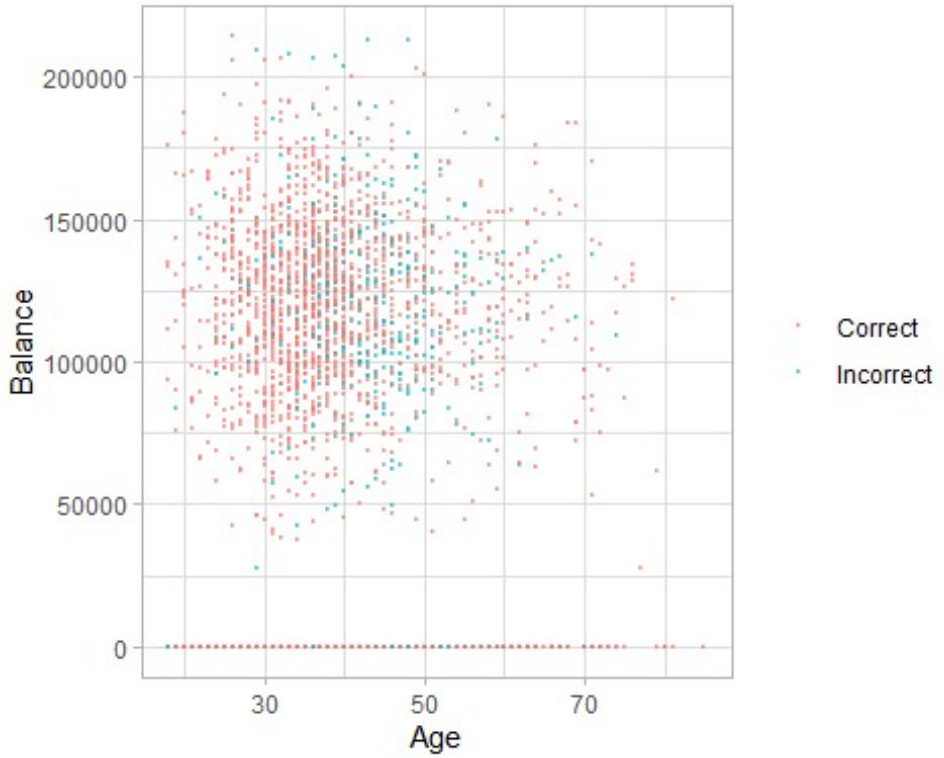
```
final_wf <- workflow() %>%
  add_recipe(mod_recipe) %>%
  add_model(final_rf)

final_res <- final_wf %>%
  last_fit(df_split)

final_res %>%
  collect_metrics()
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.858 Preprocessor1_Model1
## 2 roc_auc  binary         0.853 Preprocessor1_Model1
```

The model had an accuracy of 85.8%

```
final_res %>%
  collect_predictions() %>%
  mutate(correct = case_when(
    Exited == .pred_class ~ "Correct",
    TRUE ~ "Incorrect"
  )) %>%
```

```
  bind_cols(df_test) %>%
  ggplot(aes(Age, Balance, color = correct)) +
  geom_point(size = 0.5, alpha = 0.5) +
  labs(color = NULL) + theme_light()

## New names:
## • `Exited` -> `Exited...6`
## • `Exited` -> `Exited...19`
```



Most of clients are aged between 30 and 50 years.

Conclusion

The purpose of this study was to come up with a model that could predict a customer's exit. The following models were trained on the data, and the best performing was set to be selected. The models selected for this study were Logistic Regression, K nearest neighbors, Decision Trees, Support Vector Machines and Random Forest model. These was a classification type of supervised machine learning. Simply defined, an unsupervised learning algorithm does not employ labeled input and output data. Supervised learning does. When using supervised learning, the algorithm iteratively predicts the data and modifies for the proper response in order to "learn" from the data that has been provided. The random forest model has the highest accuracy of the 4 models. The model has an accuracy of 85.8%. after tuning.