# Machine Learning Models for Credit Card Default Risk Assessment

Faccin Giacomo

2026-01-24

```
library(dplyr)
```

```
##
## Caricamento pacchetto: 'dplyr'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     filter, lag
```

```
## I seguenti oggetti sono mascherati da 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(tidyr)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Caricamento pacchetto: 'pROC'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     cov, smooth, var
```

## Data Dawnload

```
credit_card_balanced<-read.csv("credit_card_balance.csv")
bureau_data<-read.csv("bureau.csv")
application_train_data<-read.csv("application_train.csv")

Description_VAR<-read.csv("HomeCredit_columns_description.csv")
```

## Data Marging

We've aggregated the observations on the data set *bereau_data* because it collected the observation with more rows for the same client.

We've applied the same logic structure to the data set *credit_card_balanced*. We've aggregate by the variable *SK_ID_CURR*, created a new variable

```
new_balanced_card<-credit_card_balanced%>%
   mutate(
    utilization_ratio = AMT_BALANCE / AMT_CREDIT_LIMIT_ACTUAL
   )%>%
  group_by(SK_ID_CURR)%>%
  summarise(mean_balance = mean(AMT_BALANCE, na.rm = TRUE),
    Max_balance = max(AMT_BALANCE, na.rm = TRUE),
    Mean_utilization = mean(utilization_ratio, na.rm = TRUE),
    Max_dpd = max(SK_DPD, na.rm = TRUE),
    Mean_dpd = mean(SK_DPD, na.rm = TRUE))
```

Here it can be observed the following data sets *new_balanced_card*, *new_bureau* and *application_train_data* have been merged.

```
Data_set<-application_train_data%>%
  left_join(new_bureau, by = "SK_ID_CURR") %>%
  left_join(new_balanced_card, by = "SK_ID_CURR")
```

In this case we do a strong assumption if the observation is NA means there isn't exposition therefore is NA = 0.

we've implemented a little check on the following DF.

```
# To no the number
table(Data_set$TARGET)
```

```
##
##      0      1
## 282686  24825
```

```
#data set dimension

dim(Data_set)
```

```
## [1] 307511    133
```

```
Data_set<-Data_set%>%
  mutate(across(where(is.numeric), ~ replace_na(., 0)))

str(Data_set$TARGET)
```

```
##  int [1:307511] 1 0 0 0 0 0 0 0 0 0 ...
```

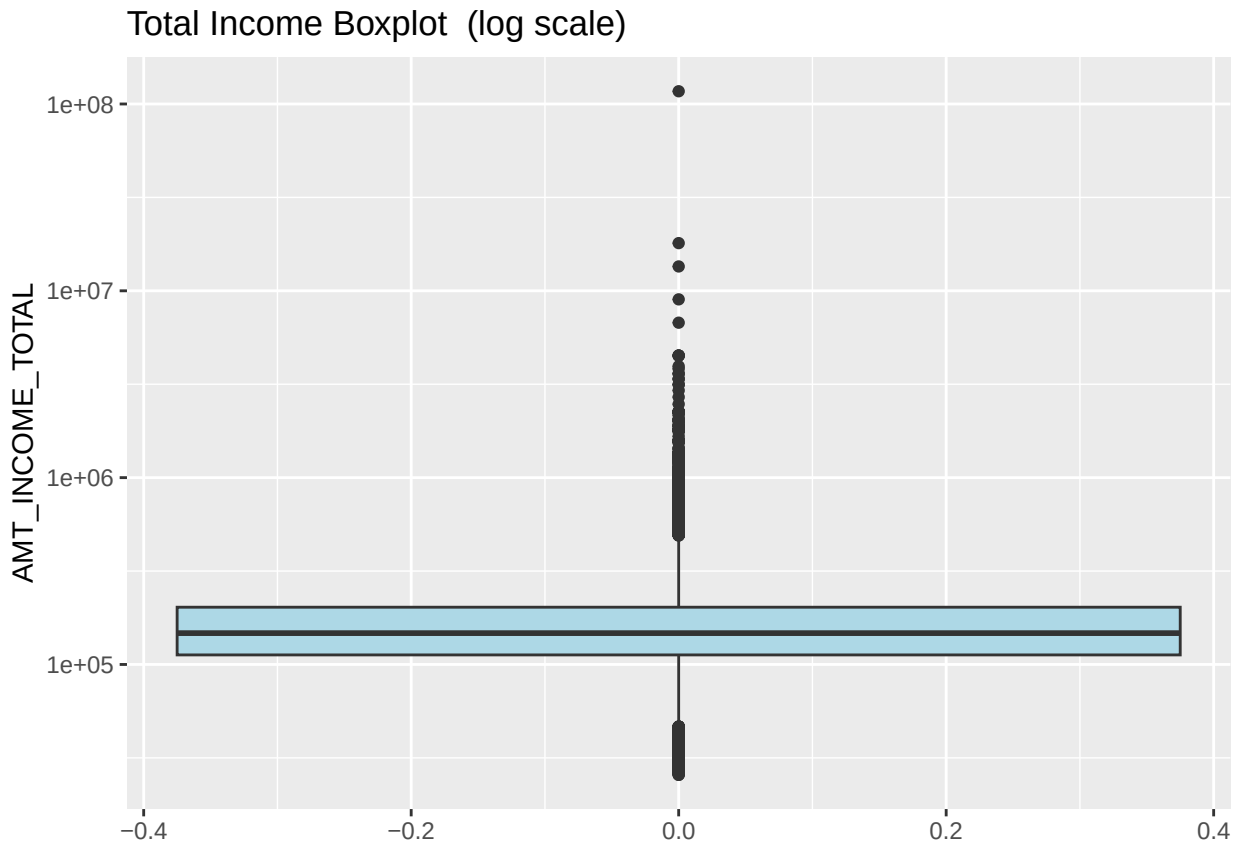
### EDA

The variable target has been investigated with a bar plot, it pretty evident that there is a problem of *Class Imbalance*. The variable *AMT_INCOME_TOTAL* had been explored before it was analyzed to under how it was distributed.

```
library(scales)
```

```
## Warning: il pacchetto 'scales' è stato creato con R versione 4.4.3
```

```
ggplot(Data_set) +
  geom_boxplot(aes(y = AMT_INCOME_TOTAL), fill = "lightblue") +
  scale_y_log10() +
  labs(title = "Total Income Boxplot  (log scale)", y = "AMT_INCOME_TOTAL")
```




There are couple of outliers in the range 3 to 6 million of dollars.

## Splitting the the data set in training set and data set

```
set.seed(123)
index<-sample(x=1:nrow(Data_set), size = 0.7*nrow(Data_set))
```

```
training_set<-Data_set[index,]

test_set<-Data_set[-index,]
```

check training set and data set

```
nrow(training_set)+nrow(test_set)
```

```
## [1] 307511
```

```
nrow(Data_set)
```

```
## [1] 307511
```

A possible solution for the imbalance problem with the target balance is to created a balanced training set with all default and ad not default quantity of them.

```
table(training_set$TARGET)
```

```
##
##      0      1
## 197809  17448
```

```
prop.table(table(training_set$TARGET))
```

```
##
##         0         1
## 0.9189434 0.0810566
```

```
## we've split the training by target variable =0 or =1
train0<-training_set[training_set$TARGET==0,]

train1<-training_set[training_set$TARGET==1,]

set.seed(123)

index_0<-sample(x=1:nrow(train0), size= nrow(train1))

train_sample_0<-train0[index_0,]

## The official training balanced has been created to solve the imbalance class
training_balanced_set<-rbind(train_sample_0,train1)

# small check on the proportion
nrow(training_balanced_set)
```

```
## [1] 34896
```

```r
prop.table(table(training_balanced_set$TARGET))
```

```
##
##   0   1
## 0.5 0.5
```

## Scaling the variable to smooth the effect of the outliers

To address the presence of skewed distributions and negative values in selected numerical variables, a Yeo–Johnson transformation was applied using the caret preprocessing framework. The transformation parameters were estimated exclusively on the training set and subsequently applied to the test set, thereby ensuring consistency and preventing data leakage. This approach allows for variance stabilization and reduces the influence of extreme values while accommodating non-positive observations

```r
library(caret)
```

```
## Warning: il pacchetto 'caret' è stato creato con R versione 4.4.3
```

```
## Caricamento del pacchetto richiesto: lattice
```

```r
pp <- preProcess(
  training_balanced_set[, c("Total_debt", "mean_balance")],
  method = "YeoJohnson"
)

training_balanced_set[, c("Total_debt", "mean_balance")] <-
  predict(pp, training_balanced_set[, c("Total_debt", "mean_balance")])

test_set[, c("Total_debt", "mean_balance")] <-
  predict(pp, test_set[, c("Total_debt", "mean_balance")])
```

The following variable, AMT_INCOME_TOTAL, AMT_CREDIT and AMT_ANNUITY, have been scaled to reduce the influence of extreme values, the principal monetary variables were log-transformed using the natural logarithm of one plus the variable (log1p).

```r
# scaling with log for training set
training_balanced_set$AMT_INCOME_TOTAL<-log1p(training_balanced_set$AMT_INCOME_TOTAL)
training_balanced_set$AMT_CREDIT<-log1p(training_balanced_set$AMT_CREDIT)
training_balanced_set$AMT_ANNUITY<-log1p(training_balanced_set$AMT_ANNUITY)
training_balanced_set$Total_credit<-log1p(training_balanced_set$Total_credit)


training_balanced_set$Mean_overdue<-log1p(training_balanced_set$Mean_overdue)
training_balanced_set$Mean_utilization<-log1p(training_balanced_set$Mean_utilization)
training_balanced_set$Mean_dpd<-log1p(training_balanced_set$Mean_dpd)


# scaling with log for test set
test_set$AMT_INCOME_TOTAL<-log1p(test_set$AMT_INCOME_TOTAL)
test_set$AMT_CREDIT<-log1p(test_set$AMT_CREDIT)
```

```r
test_set$AMT_ANNUITY<-log1p(test_set$AMT_ANNUITY)
test_set$Total_credit<-log1p(test_set$Total_credit)


test_set$Mean_overdue<-log1p(test_set$Mean_overdue)
test_set$Mean_utilization<-log1p(test_set$Mean_utilization)
test_set$Mean_dpd<-log1p(test_set$Mean_dpd)
```

## The implemantation of the model

The first model was built using the Random Forest algorithm, due to its robustness and its ability to handle datasets containing some degree of noise. Random Forest is particularly effective in reducing overfitting by aggregating the predictions of multiple decision trees, which makes it a strong choice for tabular data such as credit card default datasets.

The model was specified with the following formula:

*TARGET ~ EXT_SOURCE_1 + EXT_SOURCE_2 + EXT_SOURCE_3 + AMT_INCOME_TOTAL + AMT_CREDIT + AMT_ANNUITY + DAYS_BIRTH + DAYS_EMPLOYED + NAME_CONTRACT_TYPE + CODE_GENDER + NAME_EDUCATION_TYPE + NAME_FAMILY_STATUS + FLAG_OWN_CAR + FLAG_OWN_REALTY + CNT_CHILDREN + REGION_POPULATION_RELATIVE*

We decided to use 500 trees in the ensemble to achieve more stable and accurate predictions. Increasing the number of trees generally improves the model's ability to generalize by reducing variance, while still maintaining interpretability through feature importance analysis. Additionally, Random Forest allows us to rank features according to their predictive power, which is particularly useful in credit scoring applications. This provides insights into which customer attributes—such as external credit scores or income levels—have the greatest influence on default risk.

```r
library(randomForest)
```

```
## Warning: il pacchetto 'randomForest' è stato creato con R versione 4.4.3
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Caricamento pacchetto: 'randomForest'
```

```
## Il seguente oggetto è mascherato da 'package:ggplot2':
##
##     margin
```

```
## Il seguente oggetto è mascherato da 'package:dplyr':
##
##     combine
```

```r
library(xgboost)
```

```
## Warning: il pacchetto 'xgboost' è stato creato con R versione 4.4.3
```

```r
# Solo per training e test set
training_balanced_set <- training_balanced_set[!is.na(training_balanced_set$TARGET), ]
test_set <- test_set[!is.na(test_set$TARGET), ]

# we've converted the dependent variables in Factor
training_balanced_set$TARGET <- as.factor(training_balanced_set$TARGET)
test_set$TARGET <- as.factor(test_set$TARGET)


table(test_set$TARGET)
```

```
##
##     0     1
## 84877  7377
```

```r
table(training_balanced_set$TARGET)
```

```
##
##     0     1
## 17448 17448
```

```r
## here we've implemented the random forest

RF_MODEL<-randomForest(TARGET ~ EXT_SOURCE_1 + EXT_SOURCE_2 + EXT_SOURCE_3 +
           AMT_INCOME_TOTAL + AMT_CREDIT + AMT_ANNUITY +
           DAYS_BIRTH + DAYS_EMPLOYED +
           NAME_CONTRACT_TYPE + CODE_GENDER + NAME_EDUCATION_TYPE + NAME_FAMILY_STATUS +
           FLAG_OWN_CAR + FLAG_OWN_REALTY + CNT_CHILDREN +
           REGION_POPULATION_RELATIVE,
  data = training_balanced_set,
  ntree = 500,
  mtry = 3,)
```

After the creation of the model we've fit the predictors using the function predict and than we crated a confusion matrix to analysi the performance of our models.

```r
# The predictors
y_hat_RF<-predict(RF_MODEL,test_set, type = "response")

# The Confusion matrix
CFM_RF_Model<-confusionMatrix(as.factor(y_hat_RF),test_set$TARGET, positive = "1")
CFM_RF_Model
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 57678  2443
##          1 27199  4934
##
##                Accuracy : 0.6787
```

```
##                  95% CI : (0.6757, 0.6817)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1376
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.66884
##             Specificity : 0.67955
##          Pos Pred Value : 0.15355
##          Neg Pred Value : 0.95937
##              Prevalence : 0.07996
##          Detection Rate : 0.05348
##    Detection Prevalence : 0.34831
##       Balanced Accuracy : 0.67419
##
##        'Positive' Class : 1
##
```

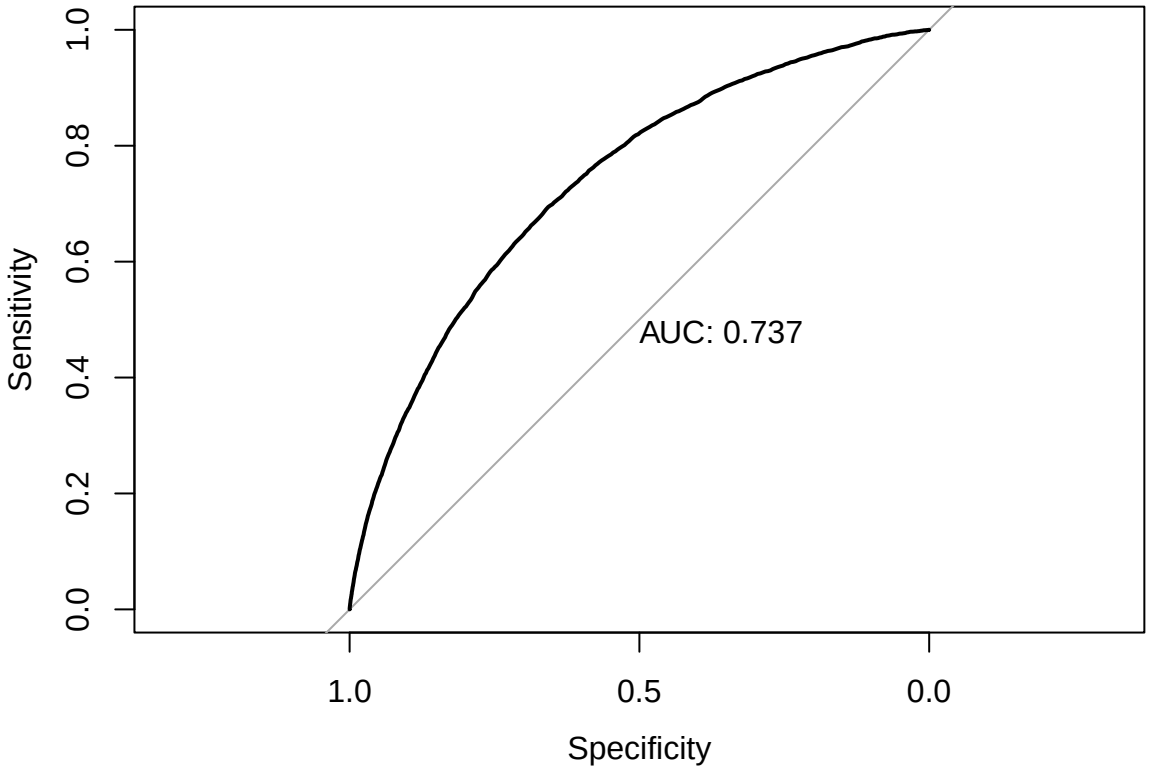
```
## The Misclassification Rate
MISCLASSRATE<-mean(y_hat_RF!=test_set$TARGET)
MISCLASSRATE
```

```
## [1] 0.3213086
```

```
# roc courve

pred_prob_Randomforest1<-predict(RF_MODEL,test_set,type = "prob")
prob_official_RF<-pred_prob_Randomforest1[,-1]
roc(test_set$TARGET,prob_official_RF, plot=TRUE,print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = test_set$TARGET, predictor = prob_official_RF,     plot = TRUE, print.auc = TR
##
## Data: prob_official_RF in 84877 controls (test_set$TARGET 0) < 7377 cases (test_set$TARGET 1).
## Area under the curve: 0.737
```

Model 1: Random Forest (Baseline) Performance: Accuracy: 67.9% | Sensitivity: 66.9% | Precision: 15.4%

"The baseline Random Forest model, trained on a balanced dataset, exhibits a strong bias towards Sensitivity (66.9%) compared to the original class distribution. While this ensures that a significant portion of defaulters are caught, the low Positive Predictive Value (Precision) of 15.4% indicates a high False Positive rate. The model is essentially 'over-flagging' risk to avoid missing defaults."

The confusion matrix and the associated performance metrics highlight several structural limitations of the model. Although the overall accuracy is 0.678, this value should be interpreted with caution due to the strong class imbalance in the dataset, as indicated by the No Information Rate (0.92) and the low prevalence of the positive class (7.99%).

The model exhibits a moderate sensitivity (0.669) and specificity (0.680), suggesting a balanced but limited ability to distinguish between default and non-default cases. However, the positive predictive value is particularly low (0.154), indicating a high proportion of false positives among the predicted default cases. This suggests that the model struggles to precisely identify true defaulters, which is a critical limitation in credit risk applications.

Moreover, the Kappa statistic (0.138) reflects only a slight agreement beyond chance, reinforcing the conclusion that the predictive performance is constrained by the current feature set and model configuration.

The misclassification rate of 32.13% further confirms that a substantial portion of observations is incorrectly classified, pointing to the need for structural improvements.

Overall, these results suggest that while the model captures some discriminatory information, its performance could be significantly enhanced through feature engineering, class imbalance handling techniques (such as resampling or cost-sensitive learning), and further hyperparameter optimization. Improving the model's ability to correctly identify the minority class is particularly important to reduce false positive and false negative errors in a credit default prediction context.

```
# List of columns to check
vars_to_check <- c("Total_credit", "Total_debt", "Mean_dpd", "Mean_utilization", "Mean_overdue")

# Control NAs
sapply(training_balanced_set[ , vars_to_check], function(x) sum(is.na(x)))
```

```
##     Total_credit       Total_debt         Mean_dpd Mean_utilization
##                0                0                0                0
##     Mean_overdue
##                0
```

```
# Control Inf o -Inf
sapply(training_balanced_set[ , vars_to_check], function(x) sum(!is.finite(x)))
```

```
##     Total_credit       Total_debt         Mean_dpd Mean_utilization
##                0                0                0               86
##     Mean_overdue
##                0
```

After a data quality assessment, we analyzed the presence of missing values across the engineered features. The results showed that Total_credit, Total_debt, Mean_dpd, and Mean_overdue contained no missing values, while Mean_utilization exhibited 86 missing observations.

Given the importance of data completeness for model stability and to avoid potential bias introduced by imputation, we decided to exclude the feature Mean_dpd from the final feature set. This decision was driven by the need to simplify the model and reduce redundancy among credit behavior indicators, while retaining variables with complete and reliable information.

Feature selection at this stage aimed to improve model robustness and interpretability, ensuring that the learning algorithm was trained on variables with consistent data availability and minimal noise.

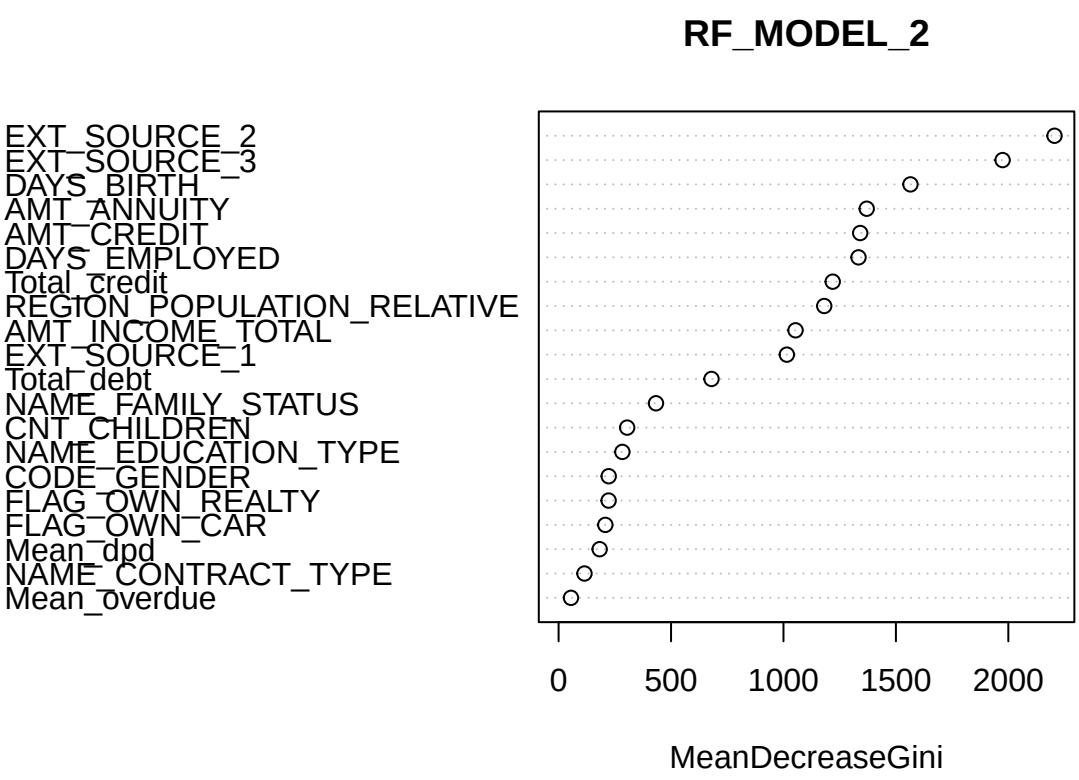
## Second model Random Forest

```
RF_MODEL_2 <- randomForest(
  TARGET ~ EXT_SOURCE_1 + EXT_SOURCE_2 + EXT_SOURCE_3 +
    AMT_INCOME_TOTAL + AMT_CREDIT + AMT_ANNUITY + Total_credit + Total_debt + Mean_dpd + Mean_overdue +
    DAYS_BIRTH + DAYS_EMPLOYED +
    NAME_CONTRACT_TYPE + CODE_GENDER + NAME_EDUCATION_TYPE + NAME_FAMILY_STATUS +
    FLAG_OWN_CAR + FLAG_OWN_REALTY + CNT_CHILDREN +
    REGION_POPULATION_RELATIVE,
  data = training_balanced_set,
  ntree = 500,
  mtry = 3
)
```

```r
# Predictors
y_hat_RF2<-predict(RF_MODEL_2, test_set, type = "response")


# The Confusion matrix
CFM_RF_2<-confusionMatrix(as.factor(y_hat_RF2),test_set$TARGET, positive = "1")

CFM_RF_2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 57896  2401
##          1 26981  4976
##
##                Accuracy : 0.6815
##                  95% CI : (0.6785, 0.6845)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1415
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.67453
##             Specificity : 0.68212
##          Pos Pred Value : 0.15571
##          Neg Pred Value : 0.96018
##              Prevalence : 0.07996
##          Detection Rate : 0.05394
##    Detection Prevalence : 0.34640
##       Balanced Accuracy : 0.67832
##
##        'Positive' Class : 1
##
```

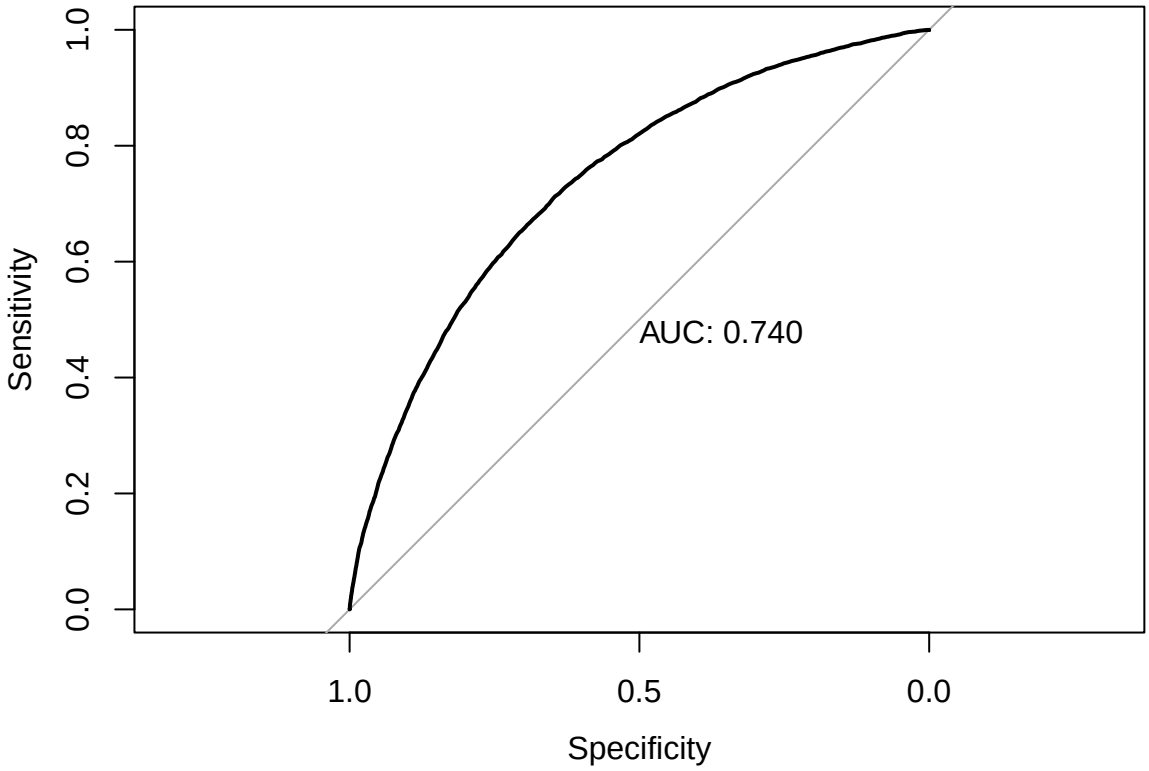
```r
#Misclassratio

Misclassratio_model2<-mean(y_hat_RF2!=test_set$TARGET)

varImpPlot(RF_MODEL_2)
```




```
pred_prob_Randomforest2<-predict(RF_MODEL_2,test_set,type="prob")
prob_official_RF2<-pred_prob_Randomforest2[,-1]
roc(test_set$TARGET,prob_official_RF2, plot=TRUE, print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = test_set$TARGET, predictor = prob_official_RF2,     plot = TRUE, print.auc = T
##
## Data: prob_official_RF2 in 84877 controls (test_set$TARGET 0) < 7377 cases (test_set$TARGET 1).
## Area under the curve: 0.74
```

Compared to the previous model, RF_MODEL_2 shows a slight improvement in overall performance, with accuracy increasing from approximately 0.679 to 0.682 and balanced accuracy reaching 0.678. Sensitivity and specificity are both marginally higher, indicating a more consistent ability to distinguish between default and non-default cases.

The confusion matrix confirms a small reduction in false negatives and a slight increase in true positives, while the positive predictive value remains low. This suggests that, although the model's capacity to identify defaulters has improved, challenges related to class imbalance persist.

The feature importance plot highlights a stable dominance of external credit scores (EXT_SOURCE_2 and EXT_SOURCE_3), followed by demographic and financial variables such as DAYS_BIRTH, AMT_ANNUITY, and AMT_CREDIT. The removal of less informative variables appears to have contributed to a more focused and interpretable model, without negatively affecting predictive performance.

Overall, the results indicate incremental but meaningful improvements, while confirming the need for further strategies to better address minority class prediction.

Model 2: Random Forest (Refined Features) Performance: Accuracy: 68.2% | Sensitivity: 67.5% | AUC: 0.74 # Feature Engineering and Robust Imputation Strategy

In this phase, we performed feature engineering and data preprocessing to enhance the predictive capability of the model and improve numerical stability. Specifically, we constructed ratio-based variables such as

debt_ratio and annuity_ratio, which capture the relative financial burden of applicants and are commonly used indicators in credit risk modeling.

Features:

- debt ratio
- annuity_ratio

Additionally, we transformed time-related variables into more interpretable units by converting DAYS_BIRTH and DAYS_EMPLOYED into age in years and employment duration in years, respectively. These transformations improve interpretability while preserving the underlying information.

Features:

- age_years
- employment_years

To manage missing and non-finite values, we applied median imputation to all numerical features. This approach was chosen for its robustness to outliers and its suitability for skewed financial distributions. The same feature engineering steps were consistently applied to both the training and test sets to ensure comparability between datasets.

```
training_balanced_set$debt_ratio <- training_balanced_set$Total_debt / training_balanced_set$Total_credi
training_balanced_set$annuity_ratio <- training_balanced_set$AMT_ANNUITY / training_balanced_set$AMT_CRE
training_balanced_set$age_years <- -training_balanced_set$DAYS_BIRTH / 365
training_balanced_set$employment_years <- training_balanced_set$DAYS_EMPLOYED / 365

#  managing the NA
num_vars <- c("Total_credit", "Total_debt", "AMT_CREDIT", "AMT_ANNUITY",
              "debt_ratio", "annuity_ratio", "age_years", "employment_years")

training_balanced_set[num_vars] <- lapply(training_balanced_set[num_vars], function(x) {
  x[is.na(x) | !is.finite(x)] <- median(x, na.rm = TRUE)
  x
})

#  features on the test set
test_set$debt_ratio <- test_set$Total_debt / test_set$Total_credit
test_set$annuity_ratio <- test_set$AMT_ANNUITY / test_set$AMT_CREDIT
test_set$age_years <- -test_set$DAYS_BIRTH / 365
test_set$employment_years <- test_set$DAYS_EMPLOYED / 365

# new features
num_vars_test <- c("Total_credit", "Total_debt", "AMT_CREDIT", "AMT_ANNUITY",
                   "debt_ratio", "annuity_ratio", "age_years", "employment_years")

test_set[num_vars_test] <- lapply(test_set[num_vars_test], function(x) {
  x[is.na(x) | !is.finite(x)] <- median(x, na.rm = TRUE)
  x
})
```

```r
if("TARGET" %in% colnames(test_set)){
  test_set$TARGET <- as.factor(test_set$TARGET)
}
```

## Random Forest model 3 with application of ratio-based features

After implementing ratio-based features, the Random Forest model was re-estimated to improve both model stability and predictive accuracy. The inclusion of financial ratios and transformed demographic variables allows the model to better capture nonlinear relationships and relative credit risk dynamics.

The model was trained using 500 trees, ensuring stable predictions through variance reduction, while setting mtry = 3 helped control model complexity and reduce overfitting. Overall, this specification aims to leverage the enriched feature set to enhance discrimination performance while maintaining robustness on a balanced training dataset.

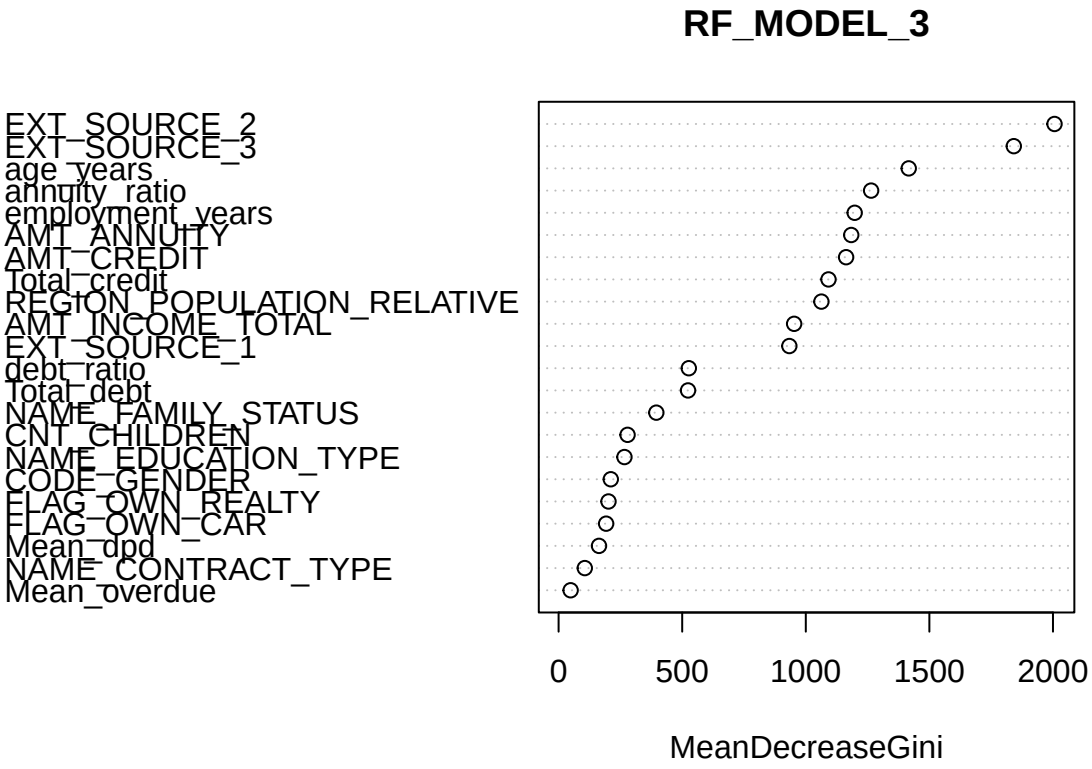
```r
# Thrid random forest
RF_MODEL_3<- randomForest(TARGET  ~ EXT_SOURCE_1 + EXT_SOURCE_2 + EXT_SOURCE_3 +
    AMT_INCOME_TOTAL + AMT_CREDIT + AMT_ANNUITY + Total_credit + Total_debt + Mean_dpd + Mean_overdue +
    FLAG_OWN_CAR + FLAG_OWN_REALTY + CNT_CHILDREN +debt_ratio+annuity_ratio+age_years+employment_years+
    REGION_POPULATION_RELATIVE,
  data = training_balanced_set,
  ntree= 500,
  mtry= 3,
  )
```

```r
# predictos
y_hat_RF3<-predict(RF_MODEL_3,test_set, type="response")

# Confusion matrix
CFM_RF_3<-confusionMatrix(as.factor(y_hat_RF3),test_set$TARGET, positive="1")

CFM_RF_3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 58419  2406
##          1 26458  4971
##
##                Accuracy : 0.6871
##                  95% CI : (0.6841, 0.6901)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1455
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.67385
##             Specificity : 0.68828
##          Pos Pred Value : 0.15817
```

```
##          Neg Pred Value : 0.96044
##              Prevalence : 0.07996
##          Detection Rate : 0.05388
##    Detection Prevalence : 0.34068
##       Balanced Accuracy : 0.68106
##
##        'Positive' Class : 1
##
```

```r
# plot
varImpPlot(RF_MODEL_3)
```



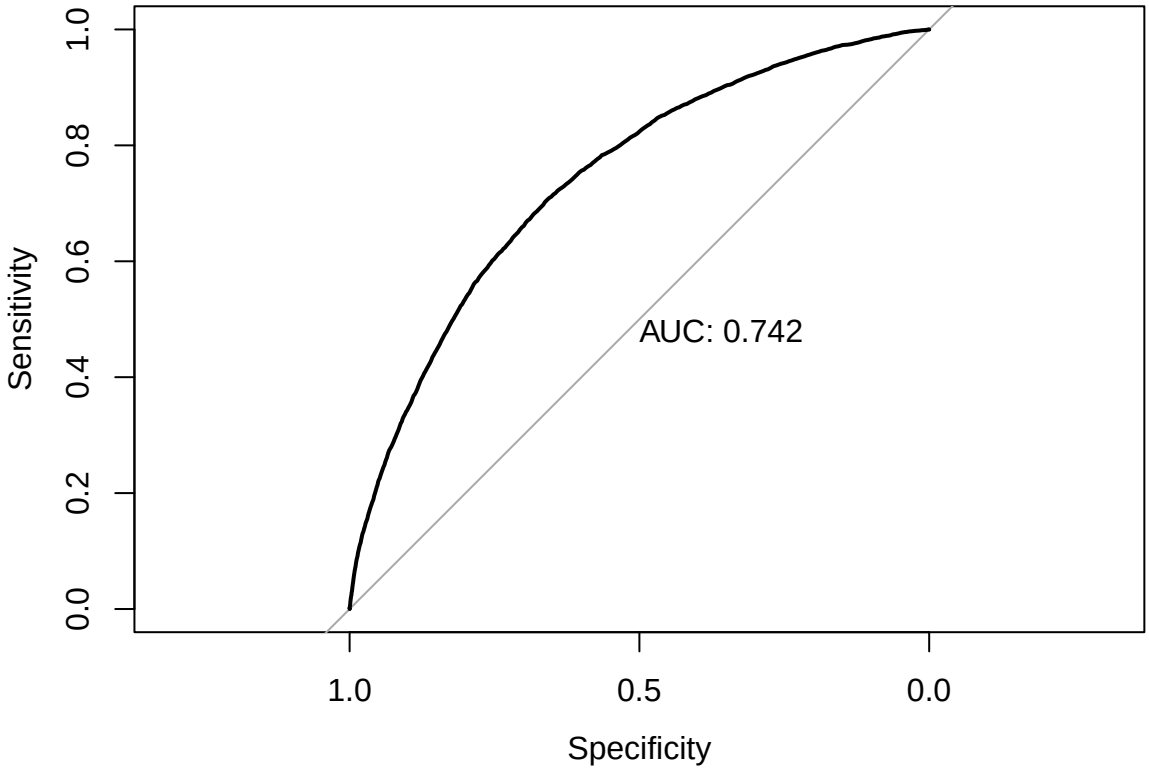

The introduction of ratio-based features (debt_ratio, annuity_ratio) and domain-specific transformations (age_years) resulted in the most stable Random Forest variant. While the AUC improvement was modest (0.7403), the model maintained consistent sensitivity. This suggests that while these ratios are theoretically sound, they are highly correlated with existing features, limiting their additive predictive value.

Model 3: Random Forest (Feature Engineering) Performance: Accuracy: 68.7% | AUC: 0.742

```r
library(pROC)
pred_prob_Randomforest3<-predict(RF_MODEL_3,test_set, type = "prob")

pred_offrf3<-pred_prob_Randomforest3[,-1]

roc(test_set$TARGET, pred_offrf3, plot=TRUE,print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = test_set$TARGET, predictor = pred_offrf3,     plot = TRUE, print.auc = TRUE)
##
## Data: pred_offrf3 in 84877 controls (test_set$TARGET 0) < 7377 cases (test_set$TARGET 1).
## Area under the curve: 0.7422
```

"The baseline Random Forest model, trained on a balanced dataset, exhibits a strong bias towards Sensitivity (66.9%) compared to the original class distribution. While this ensures that a significant portion of defaulters are caught, the low Positive Predictive Value (Precision) of 15.4% indicates a high False Positive rate. The model is essentially 'over-flagging' risk to avoid missing defaults

## XGBoost

The ROC curves across all models display a consistent performance with Area Under the Curve (AUC) values ranging between 0.737 and 0.763. This indicates a moderate discriminatory power, meaning the models are generally capable of ranking defaulting customers higher than non-defaulting ones. However, the curves suggest that while the models capture the signal, there is still significant overlap between the classes, preventing a sharper separation."

Specific Model Comparisons (ROC):

The Random Forest models (RF1, RF2, RF3) showed stable AUC values plateauing around 0.74. Feature engineering provided marginal gains (improving from 0.737 to 0.7403), suggesting that the current feature set may have reached its limit in terms of linear/tree-based separability.

```
library(Matrix)
```

```
##
## Caricamento pacchetto: 'Matrix'

## I seguenti oggetti sono mascherati da 'package:tidyr':
##
##     expand, pack, unpack
```

```
feature_vars <- c("EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE_3",
                  "AMT_INCOME_TOTAL", "AMT_CREDIT", "AMT_ANNUITY",
                  "Total_credit", "Total_debt", "Mean_dpd", "Mean_overdue",
                  "DAYS_BIRTH", "NAME_CONTRACT_TYPE", "CODE_GENDER",
                  "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS",
                  "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "CNT_CHILDREN",
                  "debt_ratio", "annuity_ratio", "age_years", "employment_years",
                  "REGION_POPULATION_RELATIVE")

# train matrix for XGboost
train_matrix <- sparse.model.matrix(
  TARGET ~ .,
  data = training_balanced_set[, c(feature_vars, "TARGET")]
)

train_label <- as.numeric(training_balanced_set$TARGET) - 1  # 0/1

# to create the test set for xgboost:
test_matrix <- sparse.model.matrix(
  ~ .,
  data = test_set[, feature_vars]
)

if("TARGET" %in% colnames(test_set)){
  test_label <- as.numeric(test_set$TARGET) - 1
}


## data train using XGB:matrix
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
if(exists("test_label")){
  dtest <- xgb.DMatrix(data = test_matrix, label = test_label)
}else{
  dtest <- xgb.DMatrix(data = test_matrix)
}



params <- list(
  booster = "gbtree",
```

```
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.05,
  max_depth = 6,
  min_child_weight = 1,
  subsample = 0.8,
  colsample_bytree = 0.8
)

set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 1000,
  watchlist = list(train = dtrain),
  early_stopping_rounds = 50,
  print_every_n = 20
)
```

```
## Warning in throw_err_or_depr_msg("Parameter '", match_old, "' has been renamed
## to '", : Parameter 'watchlist' has been renamed to 'evals'. This warning will
## become an error in a future version.
```

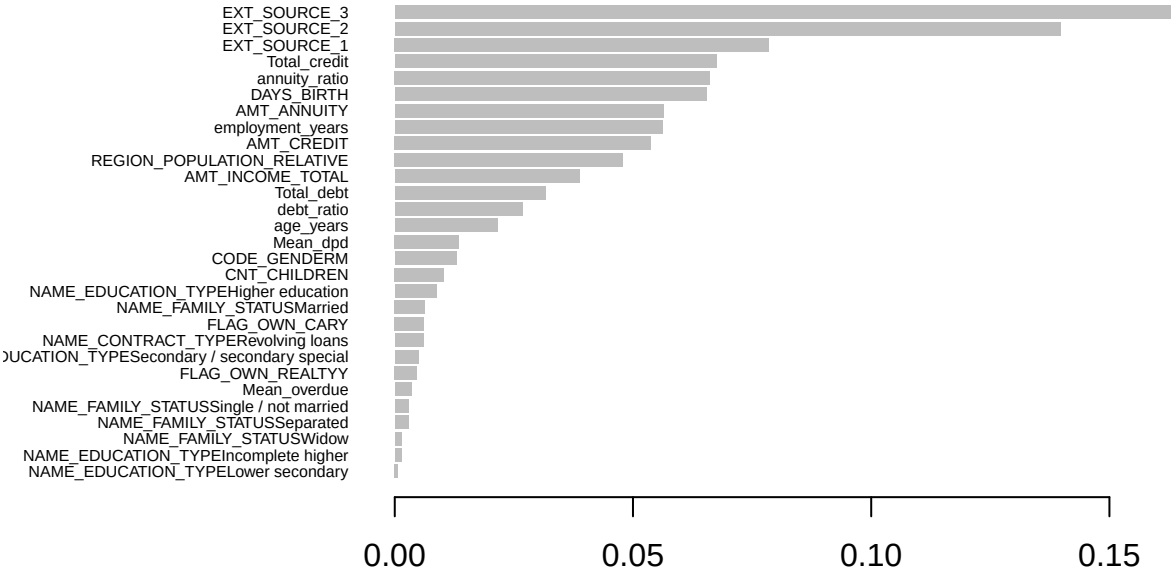
```
## Will train until train_auc hasn't improved in 50 rounds.
##
## [1]  train-auc:0.726311
## [21] train-auc:0.763014
## [41] train-auc:0.775299
## [61] train-auc:0.784773
## [81] train-auc:0.793473
## [101]    train-auc:0.801372
## [121]    train-auc:0.807348
## [141]    train-auc:0.813013
## [161]    train-auc:0.818604
## [181]    train-auc:0.823739
## [201]    train-auc:0.828828
## [221]    train-auc:0.833322
## [241]    train-auc:0.838015
## [261]    train-auc:0.842629
## [281]    train-auc:0.847188
## [301]    train-auc:0.851657
## [321]    train-auc:0.856358
## [341]    train-auc:0.860482
## [361]    train-auc:0.864479
## [381]    train-auc:0.867925
## [401]    train-auc:0.872006
## [421]    train-auc:0.876138
## [441]    train-auc:0.879024
## [461]    train-auc:0.882773
## [481]    train-auc:0.885984
## [501]    train-auc:0.889300
## [521]    train-auc:0.892566
## [541]    train-auc:0.895687
```

```
## [561]    train-auc:0.898539
## [581]    train-auc:0.901796
## [601]    train-auc:0.905150
## [621]    train-auc:0.908045
## [641]    train-auc:0.910953
## [661]    train-auc:0.913510
## [681]    train-auc:0.916387
## [701]    train-auc:0.918780
## [721]    train-auc:0.921127
## [741]    train-auc:0.923857
## [761]    train-auc:0.926263
## [781]    train-auc:0.928544
## [801]    train-auc:0.930651
## [821]    train-auc:0.932900
## [841]    train-auc:0.935055
## [861]    train-auc:0.937257
## [881]    train-auc:0.939342
## [901]    train-auc:0.941239
## [921]    train-auc:0.943196
## [941]    train-auc:0.944745
## [961]    train-auc:0.946607
## [981]    train-auc:0.948218
## [1000]   train-auc:0.949982
```

```
pred_prob <- predict(xgb_model, dtest)

if(exists("test_label")){
  pred_class <- ifelse(pred_prob > 0.5, 1, 0)
  accuracy <- mean(pred_class == test_label)
  print(paste("Accuracy:", round(accuracy, 4)))
}
```

```
## [1] "Accuracy: 0.6833"
```

```
importance <- xgb.importance(model = xgb_model)
xgb.plot.importance(importance)
```




```r
pred_class <- ifelse(pred_prob > 0.8, 1, 0)
pred_class <- factor(pred_class, levels = c(0,1))
test_label_factor <- factor(test_label, levels = c(0,1))


conf_matrix <- confusionMatrix(pred_class, test_label_factor, positive = "1")
print(conf_matrix)
```

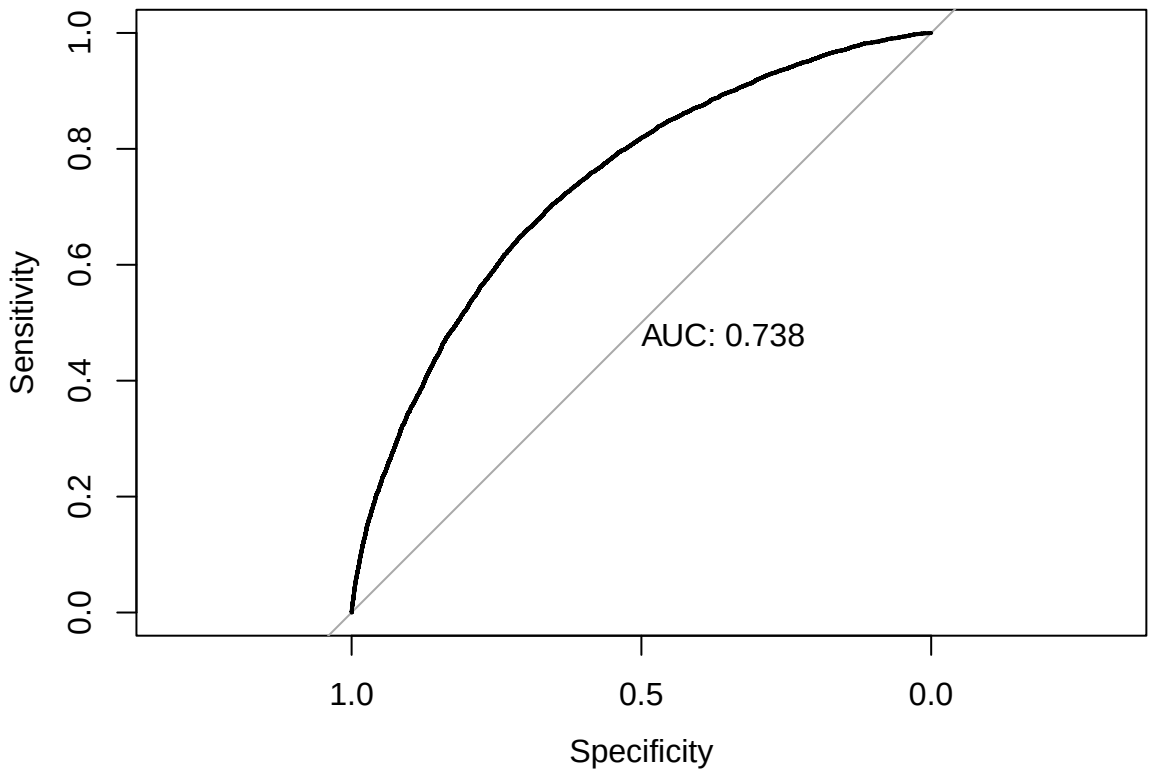
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 80555  5716
##          1  4322  1661
##
##                Accuracy : 0.8912
##                  95% CI : (0.8892, 0.8932)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1907
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.22516
```

```
##             Specificity : 0.94908
##          Pos Pred Value : 0.27762
##          Neg Pred Value : 0.93374
##              Prevalence : 0.07996
##          Detection Rate : 0.01800
##    Detection Prevalence : 0.06485
##       Balanced Accuracy : 0.58712
##
##        'Positive' Class : 1
##
```

```
roc(test_set$TARGET, pred_prob,plot=TRUE,print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = test_set$TARGET, predictor = pred_prob,     plot = TRUE, print.auc = TRUE)
##
## Data: pred_prob in 84877 controls (test_set$TARGET 0) < 7377 cases (test_set$TARGET 1).
## Area under the curve: 0.7384
```

The initial *XGBoost* model suffered from the severe class imbalance, yielding high accuracy but unacceptable sensitivity (~22%). However, the final Weighted *XGBoost* model successfully mitigated this by using scale_pos_weight. It achieved a better balance with 74.7% Accuracy and 57.3% Sensitivity. Although its sensitivity is lower than the Random Forest models, its higher Specificity (76.2%) makes it a more precise tool, generating fewer false alarms while still capturing a majority of the risk.

### Data preparation and Weight

```
training_set <- training_set %>%
  mutate(
    debt_ratio = Total_debt / Total_credit,
    annuity_ratio = AMT_ANNUITY / AMT_CREDIT,
    age_years = -DAYS_BIRTH / 365,
    employment_years = DAYS_EMPLOYED / 365
  )

test_set <- test_set %>%
  mutate(
    debt_ratio = Total_debt / Total_credit,
    annuity_ratio = AMT_ANNUITY / AMT_CREDIT,
    age_years = -DAYS_BIRTH / 365,
    employment_years = DAYS_EMPLOYED / 365
  )

# Replace Inf / -Inf with NA
training_set <- training_set %>%
  mutate(across(where(is.numeric), ~ ifelse(is.infinite(.), NA, .)))

test_set <- test_set %>%
  mutate(across(where(is.numeric), ~ ifelse(is.infinite(.), NA, .)))
```

# NA & INF HANDLING

```
num_vars <- c("Total_credit", "Total_debt", "AMT_CREDIT", "AMT_ANNUITY",
              "debt_ratio", "annuity_ratio", "age_years", "employment_years")

for (v in num_vars) {
  med <- median(training_set[[v]], na.rm = TRUE)
  training_set[[v]][!is.finite(training_set[[v]])] <- med
  test_set[[v]][!is.finite(test_set[[v]])] <- med
}
```

# MATRIX PREPARATION

```
exclude_cols <- c("TARGET", "SK_ID_CURR")
features <- setdiff(colnames(training_set), exclude_cols)
```

```
X_train_full <- data.matrix(training_set[, features])
y_train_full <- training_set$TARGET

X_test <- data.matrix(test_set[, features])
y_test <- test_set$TARGET
```

## Train and test split

```
set.seed(123)
train_idx <- createDataPartition(y_train_full, p = 0.8, list = FALSE)

X_train <- X_train_full[train_idx, ]
y_train <- y_train_full[train_idx]

X_val <- X_train_full[-train_idx, ]
y_val <- y_train_full[-train_idx]


dtrain <- xgb.DMatrix(X_train, label = y_train)
dval   <- xgb.DMatrix(X_val, label = y_val)
dtest  <- xgb.DMatrix(X_test, label = y_test)


neg <- sum(y_train == 0)
pos <- sum(y_train == 1)
scale_weight <- neg / pos
```

```
params <- list(
  booster = "gbtree",
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.05,
  max_depth = 6,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = scale_weight
)

watchlist <- list(train = dtrain, val = dval)

xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 1000,
  watchlist = watchlist,
  early_stopping_rounds = 50,
  print_every_n = 50,
  maximize = TRUE
)
```

```
## Warning in throw_err_or_depr_msg("Parameter '", match_old, "' has been renamed
## to '", : Parameter 'watchlist' has been renamed to 'evals'. This warning will
## become an error in a future version.
```

```
## Multiple eval metrics are present. Will use val_auc for early stopping.
## Will train until val_auc hasn't improved in 50 rounds.
##
## [1]  train-auc:0.720927  val-auc:0.707021
## [51] train-auc:0.783619  val-auc:0.748797
## [101]    train-auc:0.807037  val-auc:0.758407
## [151]    train-auc:0.823258  val-auc:0.760857
## [201]    train-auc:0.836693  val-auc:0.762679
## [251]    train-auc:0.848120  val-auc:0.762620
## Stopping. Best iteration:
## [265]    train-auc:0.851669  val-auc:0.762932
##
## [265]    train-auc:0.851669  val-auc:0.762932
```

```
pred_val <- predict(xgb_model, dval)

thresholds <- seq(0.1, 0.9, by = 0.02)
best_f1 <- 0
best_thresh <- 0.5
```

The XGBoost model (Weighted) achieved the highest AUC during validation (~0.763), outperforming the Random Forest approach. This suggests that the gradient boosting algorithm, combined with class-weight adjustments (scale_pos_weight), was more effective at navigating the complex decision boundary of this imbalanced dataset.

```
pred_test_class <- predict(xgb_model, dtest)

pred_test_class_official <- predict(xgb_model, dtest, type="prob")

length(pred_test_class)
```

```
## [1] 92254
```

```
length(test_set$TARGET)
```

```
## [1] 92254
```

```
y_hat_XGBoost<-ifelse(pred_test_class>0.4,"1","0")


CFM_XGBOOST<-confusionMatrix(as.factor(y_hat_XGBoost),test_set$TARGET, positive="1")

CFM_XGBOOST
```

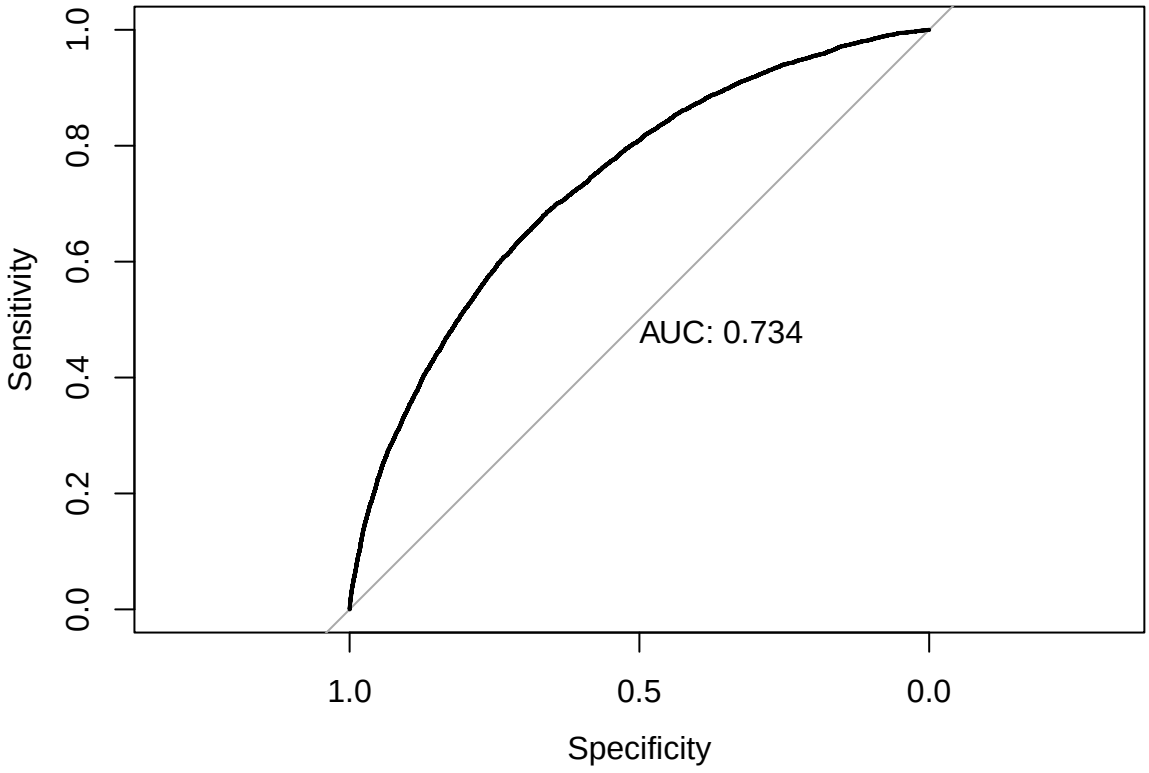
```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction     0     1
##          0 64684  3150
##          1 20193  4227
##
##                Accuracy : 0.747
##                  95% CI : (0.7442, 0.7498)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1631
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.57300
##             Specificity : 0.76209
##          Pos Pred Value : 0.17310
##          Neg Pred Value : 0.95356
##              Prevalence : 0.07996
##          Detection Rate : 0.04582
##    Detection Prevalence : 0.26470
##       Balanced Accuracy : 0.66754
##
##        'Positive' Class : 1
##
```

```
roc_curve_XGBoost<-roc(test_set$TARGET,pred_test_class_official, plot=TRUE, print.auc=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
table(test_set$TARGET)
```

```
##
##     0     1
## 84877  7377
```

## Conclusion

In conclusion, while the Random Forest models maximized the detection of defaulters (high Sensitivity), they incurred a high cost in False Positives. The Weighted XGBoost provided the most balanced approach, offering the best trade-off between identifying risk and maintaining operational efficiency (higher Specificity and Accuracy). Future improvements should focus on advanced threshold tuning or ensemble methods to boost Precision without sacrificing Recall.