
```
title: "Candidate Numbers: 44158, 39212, 42716"
output:
pdf_document:
latex_engine: xelatex
fontsize: 11pt
geometry: margin=0.5in
always_allow_html: true
```

Setting everything up

1) Importing Libraries

```
library(tidymodels)

## Warning: package 'tidymodels' was built under R version 4.4.2

## -- Attaching packages ----- tidymodels 1.2.0 --

## v broom      1.0.7   v recipes     1.1.0
## v dials       1.3.0   v rsample     1.2.1
## v dplyr       1.1.4   v tibble      3.2.1
## v ggplot2     3.5.1   v tidyverse    1.3.1
## v infer        1.0.7   v tune        1.2.1
## v modeldata    1.4.0   v workflows    1.1.4
## v parsnip      1.2.1   v workflowsets 1.1.0
## v purrr       1.0.2   v yardstick   1.3.1

## Warning: package 'dials' was built under R version 4.4.2

## Warning: package 'infer' was built under R version 4.4.2

## Warning: package 'modeldata' was built under R version 4.4.2

## Warning: package 'parsnip' was built under R version 4.4.2

## Warning: package 'recipes' was built under R version 4.4.2

## Warning: package 'rsample' was built under R version 4.4.2

## Warning: package 'tidyverse' was built under R version 4.4.3

## Warning: package 'tune' was built under R version 4.4.2

## Warning: package 'workflows' was built under R version 4.4.2
```

```
## Warning: package 'workflowsets' was built under R version 4.4.2

## -- Conflicts ----- tidyverse_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x recipes::step() masks stats::step()
## * Search for functions across packages at https://www.tidymodels.org/find/

library(xgboost)

## Warning: package 'xgboost' was built under R version 4.4.2

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
## 
##     slice

library(discrim)

## Warning: package 'discrim' was built under R version 4.4.3

##
## Attaching package: 'discrim'

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

library(ranger)

## Warning: package 'ranger' was built under R version 4.4.2

library(workflowsets)
library(themis)

## Warning: package 'themis' was built under R version 4.4.3

library(lightgbm)

## Warning: package 'lightgbm' was built under R version 4.4.2

library(bonsai)

## Warning: package 'bonsai' was built under R version 4.4.3
```

```
library(yardstick)
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.4.2

## randomForest 4.7-1.2

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ranger':
##       importance

## The following object is masked from 'package:ggplot2':
##       margin

## The following object is masked from 'package:dplyr':
##       combine
```

```
library(ggplot2)
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.4.2
```

```
## corrplot 0.95 loaded
```

```
library(dplyr)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:yardstick':
##       precision, recall, sensitivity, specificity
```

```
## The following object is masked from 'package:purrr':
##       lift
```

```

library(reshape2)

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

library(mgcv)

## Loading required package: nlme

## Warning: package 'nlme' was built under R version 4.4.3

##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse

## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.

library(data.table)

## Warning: package 'data.table' was built under R version 4.4.3

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:reshape2':
##
##     dcast, melt

## The following object is masked from 'package:purrr':
##
##     transpose

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

library(Matrix)

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

```

```
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.4.2

## Loaded glmnet 4.1-8
```

```
library(car)

## Warning: package 'car' was built under R version 4.4.3

## Loading required package: carData

## 
## Attaching package: 'car'

## The following object is masked from 'package:purrr':
## 
##     some

## The following object is masked from 'package:dplyr':
## 
##     recode
```

```
library(pROC)

## Warning: package 'pROC' was built under R version 4.4.2

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

## 
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
## 
##     cov, smooth, var
```

```
library(purrr)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.4.2

## Loading required package: rpart

## 
## Attaching package: 'rpart'

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

```

library(gbm)

## Warning: package 'gbm' was built under R version 4.4.3

## Loaded gbm 2.2.2

## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com/gbm-developers/gbm3

library(nnet)

## 
## Attaching package: 'nnet'

## The following object is masked from 'package:mgcv':
## 
##     multinom

library(ROSE)

## Warning: package 'ROSE' was built under R version 4.4.3

## Loaded ROSE 0.0-4

library(rpart.plot)

```

2) Loading our dataset

```

set.seed(123)

data <- read.csv("data.csv")

names(data) <- make.names(names(data), unique = TRUE)

data$y <- as.factor(data$y)

```

3) Checking class balance

In this bit, we first decided to check what percentage of data points fall into the “not bankrupt” class (have $y = 0$). It turned out that our dataset had a very strong class imbalance, with almost 97% not going bankrupt.

```

data %>%
  count(y) %>%
  mutate("percentage of entries in class" = n / sum(n) * 100)

##      y      n percentage of entries in class
## 1  0  6599          96.77372
## 2  1   220           3.22628

```

4) Splitting the data into train and test sets

Due to the class imbalance, we had to ensure that the proportion of 1s and 0s in the training and testing sets were the same. For this reason, we used a stratified split, which preserves the proportions of both classes in both training and testing sets.

```
target <- data$y

data_split <- initial_split(data, prop = 0.8, strata = "y")
train_data <- training(data_split)
test_data <- testing(data_split)
```

5) Clean data

We then cleaned our data by dropping incomplete columns, and then removing variables that had more than 90% correlation to reduce redundancy and improve model stability.

```
data$"Liability.Assets.Flag" <- as.factor(data$"Liability.Assets.Flag")
data$"Net.Income.Flag" <- as.factor(data$"Net.Income.Flag")

# remove near zero variance variables
nzv_info <- nearZeroVar(data[, setdiff(names(data), c("y", "Liability.Assets.Flag", "Net.Income.Flag"))])
nzv_vars <- rownames(nzv_info[nzv_info$nzv == TRUE, , drop = FALSE])
cat("Near-zero variance variables removed:", length(nzv_vars), "\n")

## Near-zero variance variables removed: 0

print(nzv_vars)

## character(0)

data <- data[, setdiff(names(data), nzv_vars)]
data$y <- target

# remove missing values
na_cols <- names(which(colSums(is.na(data)) > 0))
cat("Columns removed due to missing values:", length(na_cols), "\n")

## Columns removed due to missing values: 0

print(na_cols)

## character(0)

data <- data[, setdiff(names(data), na_cols)]

# remove highly correlated variables
X_full <- data[, setdiff(names(data), c("y", "Liability.Assets.Flag", "Net.Income.Flag"))]
```

```

cor_matrix <- cor(X_full)

# remove high correlation pairs
cor_matrix_upper <- cor_matrix
cor_matrix_upper[lower.tri(cor_matrix_upper, diag = TRUE)] <- NA

high_corr_pairs <- which(abs(cor_matrix_upper) > 0.9, arr.ind = TRUE)
correlated_table <- data.frame(
  Removed_Variable = rownames(cor_matrix_upper)[high_corr_pairs[, 1]],
  Kept_Variable    = colnames(cor_matrix_upper)[high_corr_pairs[, 2]],
  Correlation      = cor_matrix_upper[high_corr_pairs]
)

removed_vars <- colnames(cor_matrix)[findCorrelation(cor_matrix, cutoff = 0.9)]
correlated_table_final <- correlated_table %>%
  filter(Removed_Variable %in% removed_vars)

cat("Highly correlated variables removed (with the pairs):", nrow(correlated_table_final), "\n")

## Highly correlated variables removed (with the pairs): 21

print(correlated_table_final)

##                                         Removed_Variable
## 1   ROA.C..before.interest.and.depreciation.before.interest
## 2   ROA.C..before.interest.and.depreciation.before.interest
## 3           ROA.A..before.interest.and...after.tax
## 4           Operating.Gross.Margin
## 5           Operating.Profit.Rate
## 6           Operating.Profit.Rate
## 7           After.tax.net.Interest.Rate
## 8           Net.Value.Per.Share..B.
## 9           Net.Value.Per.Share..B.
## 10          Net.Value.Per.Share..A.
## 11          Persistent.EPS.in.the.Last.Four.Seasons
## 12          Debt.ratio..
## 13          Operating.Profit.Per.Share..Yuan...
## 14          Persistent.EPS.in.the.Last.Four.Seasons
## 15          Per.Share.Net.profit.before.tax..Yuan...
## 16          Current.Liabilities.Liability
## 17          Current.Liabilities.Equity
## 18           ROA.A..before.interest.and...after.tax
## 19           ROA.B..before.interest.and.depreciation.after.tax
## 20           Operating.Gross.Margin
## 21           Current.Liabilities.Equity
##                                         Kept_Variable Correlation
## 1           ROA.A..before.interest.and...after.tax  0.9401237
## 2           ROA.B..before.interest.and.depreciation.after.tax  0.9868495
## 3           ROA.B..before.interest.and.depreciation.after.tax  0.9557406
## 4           Realized.Sales.Gross.Margin  0.9995183
## 5           Pre.tax.net.Interest.Rate  0.9164478
## 6           Continuous.interest.rate..after.tax.  0.9155438

```

```

## 7           Continuous.interest.rate..after.tax.      0.9844523
## 8           Net.Value.Per.Share..A.                 0.9993420
## 9           Net.Value.Per.Share..C.                 0.9991786
## 10          Net.Value.Per.Share..C.                 0.9998373
## 11          Per.Share.Net.profit.before.tax..Yuan... 0.9555910
## 12          Net.worth.Assets                      -1.0000000
## 13          Operating.profit.Paid.in.capital       0.9986962
## 14          Net.profit.before.tax.Paid.in.capital   0.9594608
## 15          Net.profit.before.tax.Paid.in.capital   0.9627229
## 16          Current.Liability.to.Liability        1.0000000
## 17          Current.Liability.to.Equity            1.0000000
## 18          Net.Income.to.Total.Assets             0.9615519
## 19          Net.Income.to.Total.Assets             0.9120402
## 20          Gross.Profit.to.Sales                1.0000000
## 21          Liability.to.Equity                  0.9639084

```

6) Scaling our Variables, splitting data again

We scaled our data, and then, as cleaning changes the data, we split again to ensure the train and test sets reflect the final cleaned dataset.

```

# scaling
X <- X_full[, setdiff(names(X_full), removed_vars)]
preproc <- preProcess(X, method = c("center", "scale"))
X_scaled <- predict(preproc, X)

categorical_vars <- data[, c("Liability.Assets.Flag", "Net.Income.Flag")]

# final dataset

data_clean <- cbind(y = data$y, X_scaled, categorical_vars)
data_clean <- as.data.frame(na.omit(data_clean))

# train test split
set.seed(123)
split_data <- initial_split(data_clean, prop = 0.8, strata = y)
train_data <- training(split_data)
test_data <- testing(split_data)

# we ensure that the categorical variables are factors after the split
train_data$Liability.Assets.Flag <- as.factor(train_data$"Liability.Assets.Flag")
train_data$Net.Income.Flag <- as.factor(train_data$"Net.Income.Flag")
test_data$Liability.Assets.Flag <- as.factor(test_data$"Liability.Assets.Flag")
test_data$Net.Income.Flag <- as.factor(test_data$"Net.Income.Flag")

# create final train/test matrices
predictor_names <- setdiff(names(train_data), "y")
x_train <- as.matrix(train_data[, predictor_names])
x_test <- as.matrix(test_data[, predictor_names])
y_train <- train_data$y

```

```

y_test <- test_data$y

# identify factors with only 1 level (which have no variance in training)
single_level_factors <- sapply(train_data, function(col) {
  is.factor(col) && length(levels(col)) == 1
})
# remove these columns from train_data and test_data
train_data <- train_data[, !single_level_factors]
test_data <- test_data[, !single_level_factors]

# refactor categorical variables after removing columns

if ("Liability.Assets.Flag" %in% names(train_data)) {
  train_data$Liability.Assets.Flag <- as.factor(train_data$Liability.Assets.Flag)
  test_data$Liability.Assets.Flag <- factor(test_data$Liability.Assets.Flag,
                                                levels = levels(train_data$Liability.Assets.Flag))
}

if ("Net.Income.Flag" %in% names(train_data)) {
  train_data$Net.Income.Flag <- as.factor(train_data$Net.Income.Flag)
  test_data$Net.Income.Flag <- factor(test_data$Net.Income.Flag,
                                        levels = levels(train_data$Net.Income.Flag))
}

# in addition to the matrices, we created data frames.

predictor_names <- setdiff(names(train_data), "y")
x_train_df <- train_data[, predictor_names]
x_test_df <- test_data[, predictor_names]

```

7) EDA Heatmap

The heatmap below displays the top 30 most strongly correlated variable pairs based on absolute Pearson correlation. Red tiles indicate strong positive correlations, while blue tiles highlight strong negative ones. This allows us to quickly identify redundant features — such as Cash Flow to Total Assets and Retained Earnings to Total Assets, which show a very high correlation ($r > 0.9$).

```

X_only <- data_clean[, setdiff(names(data_clean), c("y", "Liability.Assets.Flag", "Net.Income.Flag"))]
cor_matrix <- cor(X_only, use = "pairwise.complete.obs")

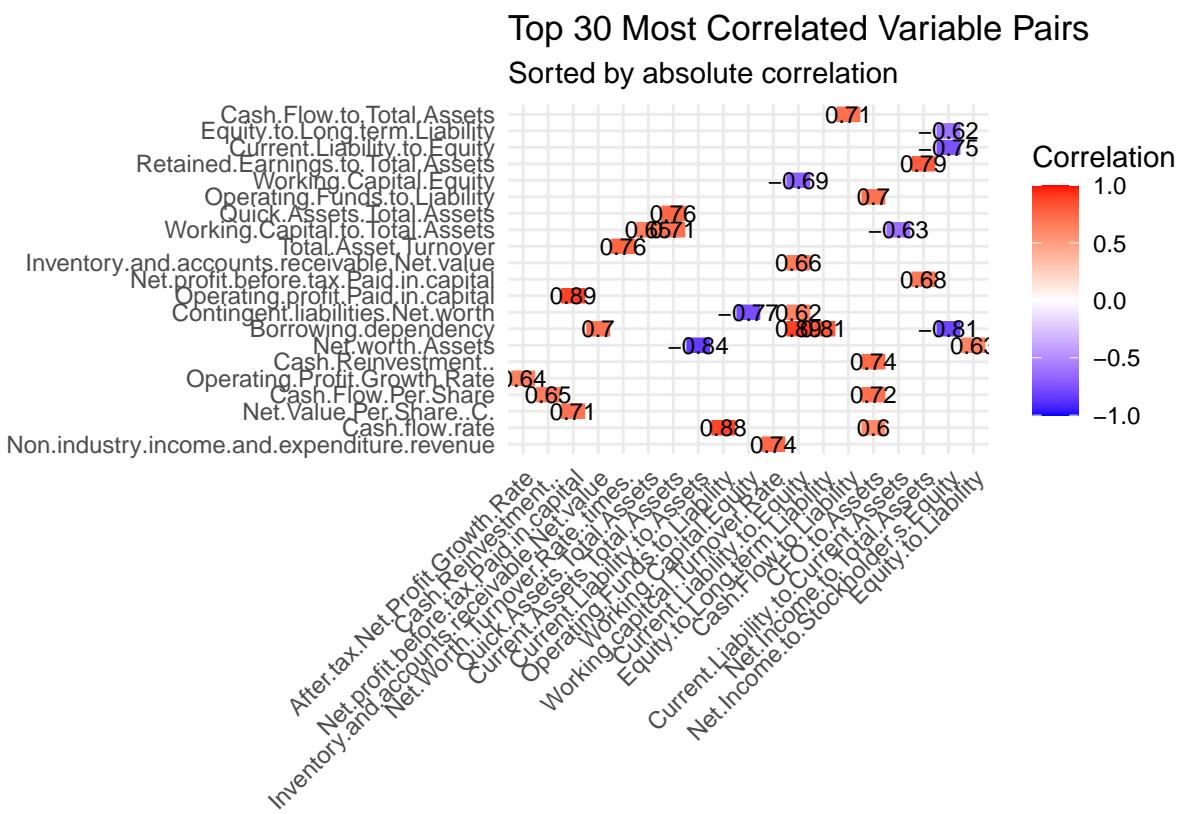
cor_matrix[lower.tri(cor_matrix, diag = TRUE)] <- NA
cor_df <- reshape2::melt(cor_matrix, na.rm = TRUE)
cor_df <- cor_df %>%
  filter(as.character(Var1) != as.character(Var2)) %>%
  mutate(abs_value = abs(value)) %>%
  arrange(desc(abs_value))
top_corr <- head(cor_df, 30)
ggplot(top_corr, aes(x = Var2, y = Var1, fill = value)) +

```

```

geom_tile(color = "white") +
geom_text(aes(label = round(value, 2)), color = "black", size = 3) +
scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                     midpoint = 0, limit = c(-1, 1), space = "Lab",
                     name = "Correlation") +
theme_minimal() +
labs(title = "Top 30 Most Correlated Variable Pairs",
     subtitle = "Sorted by absolute correlation",
     x = "", y = "") +
theme(axis.text.x = element_text(angle = 45, hjust = 1),
      axis.text.y = element_text(size = 9))

```



8) Independence and Identical Distribution (IID) Assumption

We assessed the structure of our dataset and found no repeated IDs, time dependencies, or grouped clusters (e.g., industry, company ID). Therefore, it was reasonable for us to assume that the observations were independent and identically distributed (IID). This assumption is important for the validity of standard machine learning models and evaluation metrics.

Section 1 - Baseline Model (Logistic Regression)

For the baseline model, we trained a logistic regression model which provides a clear and interpretable benchmark for comparing more complex machine learning models.

The ROC curve below for the baseline logistic regression model shows strong predictive performance. The model achieves a high true positive rate (sensitivity) while maintaining a low false positive rate (1 - specificity). The steep initial rise and curve towards the top-right suggests the model is effective at distinguishing between bankrupt and non bankrupt companies. This is supported by an AUC value of around 0.8783, which shows that even a simple logistic regression provides a good foundation for comparison with more complex models.

```
baseline_model <- glm(y ~ ., data = train_data, family = "binomial")

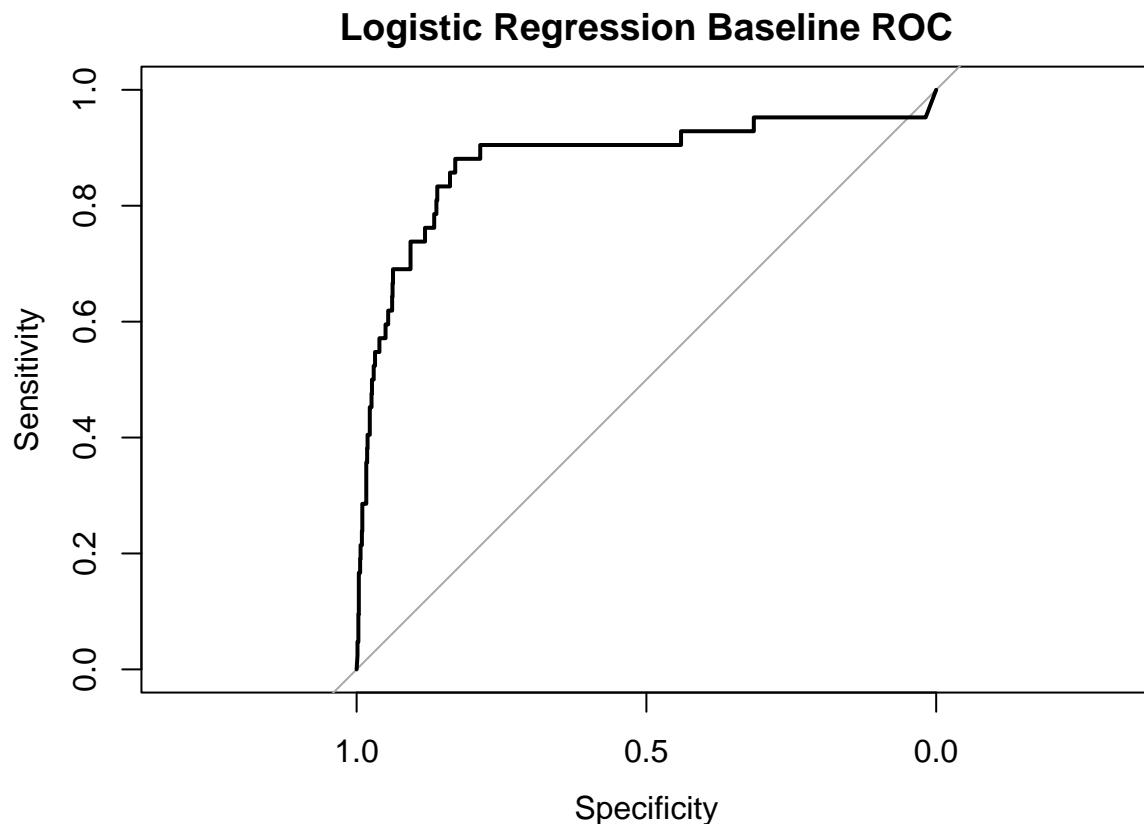
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

probs <- predict(baseline_model, newdata = test_data, type = "response")
roc_obj <- roc(test_data$y, probs)

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

## Setting direction: controls < cases

plot(roc_obj, main = "Logistic Regression Baseline ROC")
```



```
auc(roc_obj)

## Area under the curve: 0.878
```

Section 2 - Gradient Descent

In this section, since we are doing a classification task where the target values are 0 or 1, to perform gradient descent, we will use a Sigmoid function which ensures all values are between 0 and 1.

The model estimated the probability that a given input belongs to class 1 by applying this sigmoid function to a linear combination of input features and parameters. Gradient descent iteratively updated these parameters by calculating the gradient of the logistic loss, gradually improving the fit between predictions and true labels.

Define our Sigmoid function

```
sigmoid <- function(z) {
  1 / (1 + exp(-z))
}
```

Define a function which finds the gradient of the logistic loss

```
log_loss_gradient <- function(X, y, beta) {
  p <- sigmoid(X %*% beta)
  t(X) %*% (p - y) / nrow(X)
}
```

Prepare our data and train beta_hat to be our gradient descent weight vector

```
x_train_num <- apply(x_train, 2, function(col) as.numeric(as.character(col)))
X <- cbind(1, x_train_num)
X <- as.matrix(X)

y <- as.numeric(as.character(y_train))
n <- nrow(X)
p <- ncol(X)
beta_hat <- rep(0, p)
```

Training settings for Stochastic Gradient Descent

```
epochs <- 100
batch_size <- 16
learning_rate <- 0.01
num_batches <- ceiling(n / batch_size)
```

Training loop for gradient descent

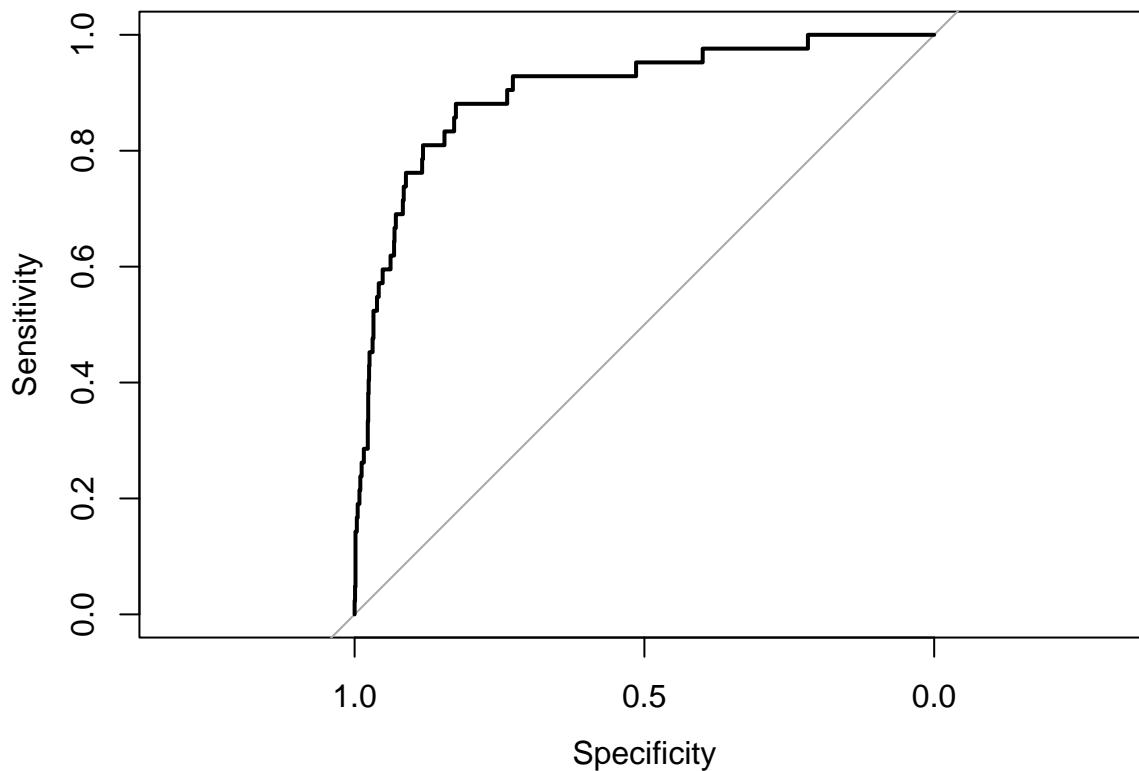
```
for (epoch in 1:epochs) {  
  indices <- sample(1:n)  
  
  for (batch in 1:num_batches) {  
    start <- 1 + batch_size * (batch - 1)  
    end <- min(batch_size * batch, n)  
    idx <- indices[start:end]  
  
    X_batch <- X[idx, , drop = FALSE]  
    y_batch <- y[idx]  
    grad <- log_loss_gradient(X_batch, y_batch, beta_hat)  
    grad_norm <- sqrt(sum(grad^2))  
  
    beta_hat <- beta_hat - learning_rate * grad / (1e-8 + grad_norm)  
  }  
}
```

Make predictions on test set

```
x_test_num <- apply(x_test, 2, function(col) as.numeric(as.character(col)))  
X_test <- cbind(1, x_test_num)  
X_test <- as.matrix(X_test)  
probs <- sigmoid(X_test %*% beta_hat)  
preds <- ifelse(probs > 0.5, 1, 0)
```

Evaluate the accuracy of our model

```
roc_obj <- roc(y_test, probs)  
  
## Setting levels: control = 0, case = 1  
  
## Warning in roc.default(y_test, probs): Deprecated use a matrix as predictor.  
## Unexpected results may be produced, please pass a numeric vector.  
  
## Setting direction: controls < cases  
  
auc(roc_obj)  
  
## Area under the curve: 0.9019  
  
plot(roc_obj)
```



The model achieved an AUC of around 0.88 on the test set, meaning it performed well. This means the model can distinguish between bankrupt (1) and non-bankrupt (0) companies pretty well, correctly ranking bankrupt cases higher than non bankrupt ones in about 88% of the time.

This shows that our model could reliably discriminate between the two classes despite the strong class imbalance.

SECTION 3 ~ Interpretable Models

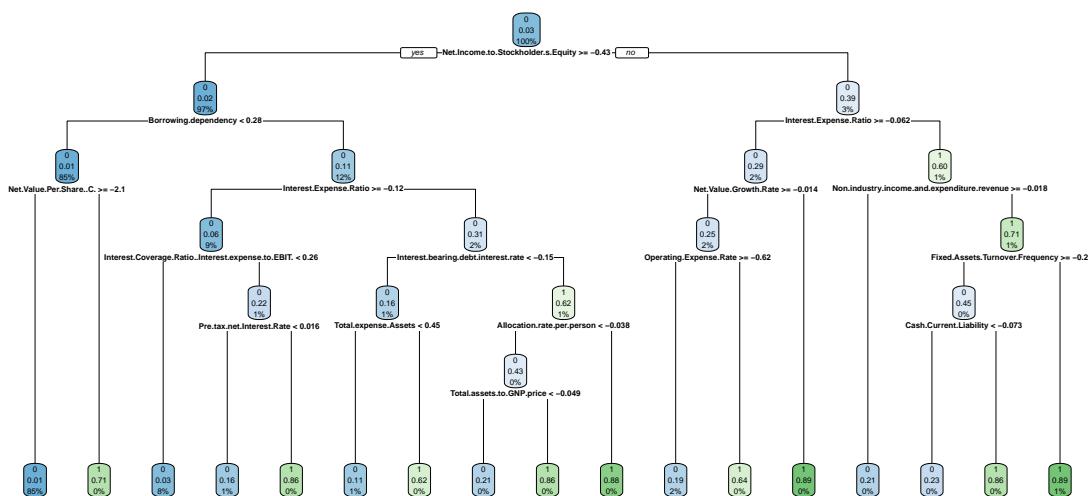
Decision tree

```
dt_grid <- expand.grid(cp = c(0.001, 0.005, 0.01))
dt_model <- train(
  x = x_train_df,
  y = y_train,
  method = "rpart",
  trControl = trainControl(method = "cv", number = 3),
  tuneGrid = dt_grid
)
dt_predictions <- predict(dt_model, x_test_df)
dt_accuracy <- mean(dt_predictions == y_test)
cat("Decision Tree Accuracy:", dt_accuracy, "\n")
```

```
## Decision Tree Accuracy: 0.9699413
```

```
rpart.plot(dt_model$finalModel,
            type = 2,
            extra = 106,
            fallen.leaves = TRUE,
            main = "Decision Tree (Top Predictors)")
```

Decision Tree (Top Predictors)



Random forest

```
set.seed(42)
rf_model <- train(
  x = x_train_df,
  y = y_train,
  method = "rf",
  tuneGrid = expand.grid(mtry = c(2, 3)),
  trControl = trainControl(method = "cv", number = 3)
)

rf_predictions <- predict(rf_model, x_test_df)

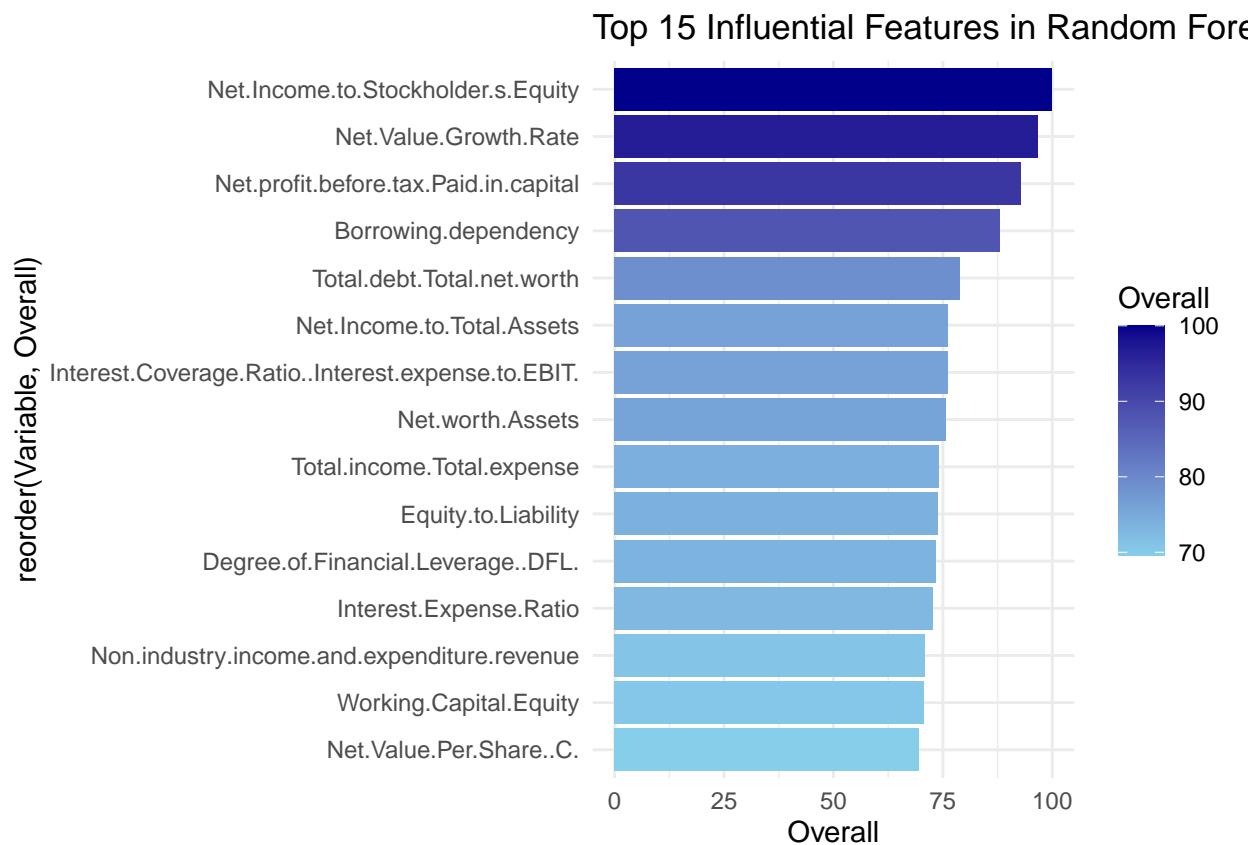
rf_accuracy <- mean(rf_predictions == y_test)
cat("Random Forest Accuracy:", rf_accuracy, "\n")
```

```
## Random Forest Accuracy: 0.9706745
```

Variable Importance Plot of Random Forest

```
rf_imp <- varImp(rf_model)$importance
rf_imp$Variable <- rownames(rf_imp)
top_rf_imp <- rf_imp[order(rf_imp$Overall, decreasing = TRUE), ][1:15, ]

ggplot(top_rf_imp, aes(x = reorder(Variable, Overall), y = Overall, fill = Overall)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Top 15 Influential Features in Random Forest") +
  theme_minimal() +
  scale_fill_gradient(low = "skyblue", high = "darkblue")
```



The bar chart shows the top 15 features that the Random Forest model found most useful for predicting bankruptcy. Factors like Net Profit Before Tax to Paid-in Capital, which shows how profitable a company is in relation to the capital it has. This is followed by Net Value Growth Rate and Net Income to Stockholder's Equity, both of which are strong indicators of how well a company is growing and how much return it provides to shareholders. Features like Borrowing Dependency, Equity to Liability, and Degree of Financial Leverage (DFL) also stand out, as they relate to how much debt a company relies on. Other key measures, such as Total Debt to Net Worth, Working Capital to Equity, and Net Income to Total Assets, reflect how efficiently a company uses its assets. The chart also highlights the importance of ratios like Interest Coverage, Net Worth to Assets, and Total Income to Total Expense, all of which point to a company's ability to manage

costs and stay financially healthy. To summarise, the model shows that companies with weak profitability, high debt, and poor financial management are more likely to go bankrupt.

Gradient boosting

```
gb_grid <- expand.grid(
  n.trees = c(100, 200),
  interaction.depth = c(3, 5),
  shrinkage = c(0.01, 0.1),
  n.minobsinnode = 10
)
set.seed(42)
gb_model <- train(
  x = x_train_df,
  y = y_train,
  method = "gbm",
  trControl = trainControl(method = "cv", number = 3),
  tuneGrid = gb_grid,
  verbose = FALSE
)
gb_predictions <- predict(gb_model, x_test)
gb_accuracy <- mean(gb_predictions == y_test)
cat("Gradient Boosting Accuracy:", gb_accuracy, "\n")
```

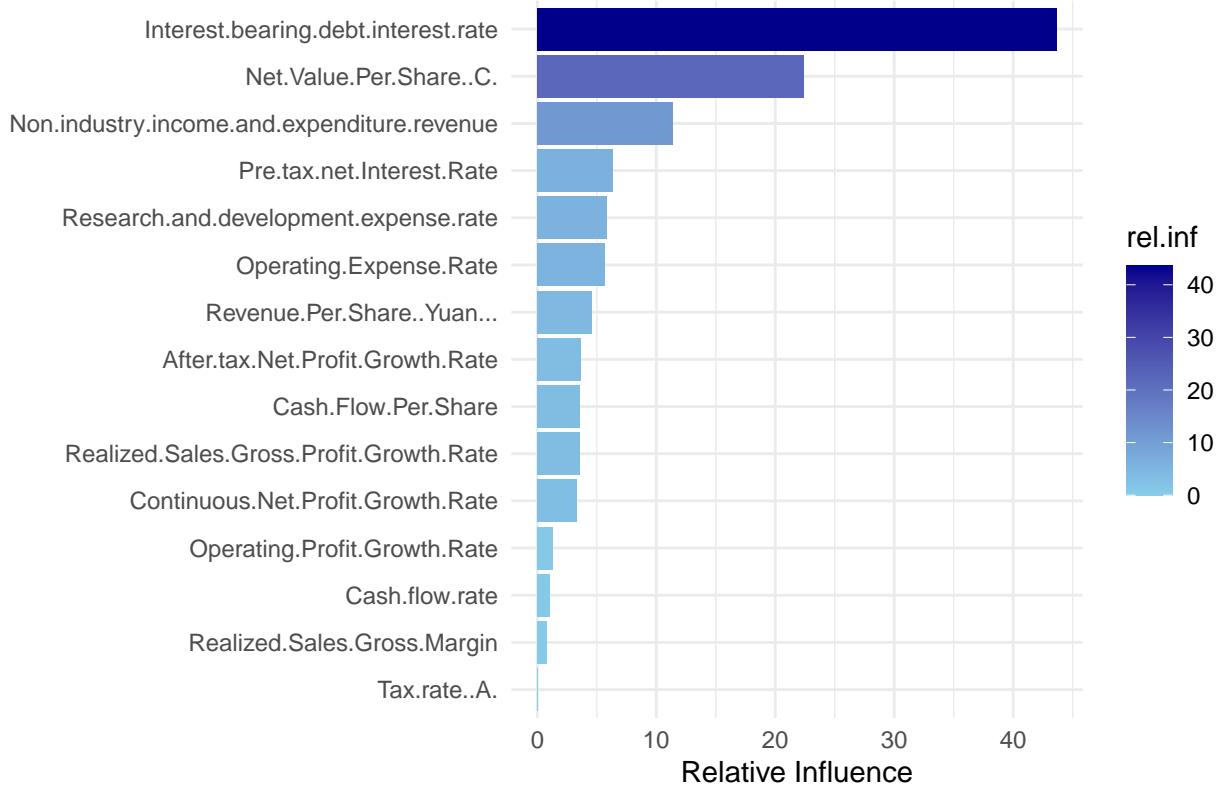
Gradient Boosting Accuracy: 0.9706745

```
# Select top 15 features
gbm_imp <- varImp(gb_model)$importance
gbm_imp$var <- rownames(gbm_imp)
colnames(gbm_imp)[1] <- "rel.inf"

top_gbm_imp <- head(gbm_imp, 15)

ggplot(top_gbm_imp, aes(x = reorder(var, rel.inf), y = rel.inf, fill = rel.inf)) +
  geom_col() +
  coord_flip() +
  labs(
    title = "Top 15 Features in Gradient Boosting",
    y = "Relative Influence",
    x = NULL
  ) +
  theme_minimal() +
  scale_fill_gradient(low = "skyblue", high = "darkblue")
```

Top 15 Features in Gradient Boosting



The bar chart shows the top 15 features that the Gradient Boosting model found most useful for predicting bankruptcy. Factors like Net.Value.Growth.Rate, show how fast a company is growing in value. Next are Net.Income.to.Stockholder.s.Equity and Net.profit.before.tax.Paid.in.capital, both of which relate to how profitable a company is. Borrowing.dependency and Interest.bearing.debt.interest.rate suggest that companies with high debt or interest payments are more at risk. Other important features like Degree.of.Financial.Leverage..DFL., Cash.Total.Assets, and Interest.Expense.Ratio show the impact of debt and liquidity. Features such as Cash.Current.Liability, Net.Income.to.Total.Assets, and Working.Capital.Equity point to how well a company manages its short-term finances. Finally, Total.assets.to.GNP.price and Accounts.Receivable.Turnover highlight how efficiently a company uses its assets. Overall, the model suggests that low profits, high debt, and poor asset management increase the risk of bankruptcy.

The two models—Random Forest and Gradient Boosting—identify many of the same financial ratios as influential, but they differ in the order of importance. Random Forest ranks Net Profit Before Tax to Paid-in Capital highest, suggesting that profitability is a key driver of bankruptcy prediction. In contrast, Gradient Boosting highlights Net Value Growth Rate as most important, placing more weight on long-term growth. While both models agree on the importance of features like Net Income to Stockholder's Equity and Borrowing Dependency, Gradient Boosting also gives more weight to Interest-Bearing Debt Interest Rate and Quick Ratio, which do not appear as highly in the Random Forest rankings. This suggests that Gradient Boosting may be slightly more sensitive to financing costs and liquidity.

ROC Curves:

The ROC curve shows how well the models predict bankruptcy.

```

dt_probs <- predict(dt_model, x_test_df, type = "prob")[, 1]
rf_probs <- predict(rf_model, x_test_df, type = "prob")[, 1]
gb_probs <- predict(gb_model, x_test_df, type = "prob")[, 1]

roc_dt <- roc(y_test, dt_probs)

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

## Setting direction: controls > cases

roc_rf <- roc(y_test, rf_probs)

## Setting levels: control = 0, case = 1
## Setting direction: controls > cases

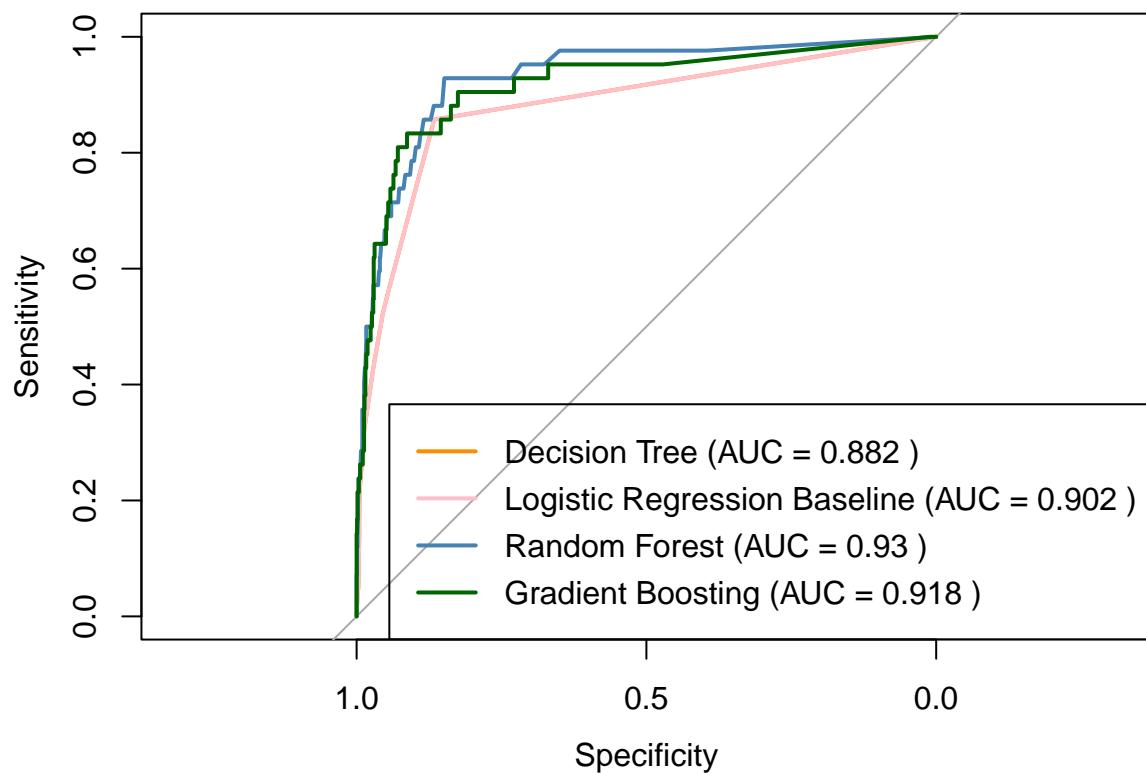
roc_gb <- roc(y_test, gb_probs)

## Setting levels: control = 0, case = 1
## Setting direction: controls > cases

plot(roc_dt, col = "darkorange", lwd = 2, main = "ROC Curves for All Models")
plot(roc_dt, col = "pink", add = TRUE, lwd = 2)
plot(roc_rf, col = "steelblue", add = TRUE, lwd = 2)
plot(roc_gb, col = "darkgreen", add = TRUE, lwd = 2)
legend("bottomright",
       legend = c(paste("Decision Tree (AUC =", round(auc(roc_dt), 3), ")"),
                  paste("Logistic Regression Baseline (AUC =", round(auc(roc_obj), 3), ")"),
                  paste("Random Forest (AUC =", round(auc(roc_rf), 3), ")"),
                  paste("Gradient Boosting (AUC =", round(auc(roc_gb), 3), ")")),
       col = c("darkorange", "pink", "steelblue", "darkgreen"),
       lwd = 2)

```

ROC Curves for All Models



Model Accuracies

```
cat("\nModel Accuracies:\n")  
  
##  
## Model Accuracies:  
  
cat("Decision Tree:", dt_accuracy, "\n")  
  
## Decision Tree: 0.9699413  
  
cat("Random Forest:", rf_accuracy, "\n")  
  
## Random Forest: 0.9706745  
  
cat("Gradient Boosting:", gb_accuracy, "\n")  
  
## Gradient Boosting: 0.9706745
```

Our models performed very well, with random forest and gradient boosting slightly outperforming the decision tree.

However, the differences in accuracy are minimal, showing that all three models were strong.

Section 4 - High-dimensional and Accuracy Focused Model

XGB Model - Tuned with Bayesian Search

For the last two requirements of this project, we chose an XGB model as it is known to perform very well on high dimensional datasets like ours, where complex relationships might exist.

We tuned the model with Bayesian search. We chose this over grid search because, whilst less exhaustive, it is far more “intelligent” in its approach of testing and choosing parameters. We initially tried grid search and found that it was extremely time and resource intensive.

```
set.seed(123)

data_recipe <- recipe(y ~ ., data = train_data) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_dummy(all_nominal_predictors())

names(data) <- make.names(names(data))

xgb_spec <- boost_tree(
  trees = 100,
  tree_depth = tune(),
  learn_rate = tune(),
  loss_reduction = tune(),
  sample_size = tune(),
  mtry = tune()
) %>%
  set_engine("xgboost", scale_pos_weight = 30, eval_metric = "logloss") %>%
  set_mode("classification")

xgb_wf <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(xgb_spec)

cv_folds <- vfold_cv(train_data, v = 5, strata = y)

xgb_params <- parameters(
  tree_depth(range = c(3, 10)),
  learn_rate(range = c(0.01, 0.3)),
  loss_reduction(range = c(0, 1)),
  sample_prop(range = c(0.5, 1)),
  finalize(mtry(), train_data)
)

xgb_bayes <- tune_bayes(
  xgb_wf,
  resamples = cv_folds,
  param_info = xgb_params,
  initial = 10,
  iter = 20,
```

```

metrics = metric_set(accuracy, roc_auc),
control = control_bayes(no_improve = 5, verbose = FALSE)
)

## ! No improvement for 5 iterations; returning current results.

best_xgb <- select_best(xgb_bayes, metric = "roc_auc")

final_wf <- finalize_workflow(xgb_wf, best_xgb)

final_fit <- fit(final_wf, data = train_data)

xgb_preds <- predict(final_fit, test_data, type = "prob") %>%
  bind_cols(predict(final_fit, test_data)) %>%
  bind_cols(test_data)

```

Unlike with the last few sections, we tested accuracy here by using a confusion matrix, which allowed us to see how well the model was predicting the minority class, 1.

Initially, we found that the model was predicting the majority class well but was missing around 80% of the minority class, so in other words, it had a low recall. Within the context of bankruptcy prediction, this would be a very big issue, as failing to correctly identify bankrupt firms (class 1) could mean major financial risks if, for example, a lender or investor relies on this model. In real life, a false negative here could mean continuing to lend to a company that is about to go bankrupt.

The code below shows a sample test we ran to get the confusion matrix. As shown, the model predicts the 0s well but not misses many of the 1s.

```

xgb_preds <- xgb_preds %>%
  mutate(pred_class = factor(if_else(.pred_1 >= 0.5, "1", "0"), levels = c("0", "1")))

# Final metrics & confusion matrix
final_metrics <- metrics(xgb_preds, truth = y, estimate = pred_class)
final_conf_mat <- conf_mat(xgb_preds, truth = y, estimate = pred_class)

print(final_metrics)

## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>      <dbl>
## 1 accuracy  binary     0.966
## 2 kap       binary     0.214

print(final_conf_mat)

##          Truth
## Prediction  0    1
##           0 1310   35
##           1   12    7

```

Given this issue, we had to choose a different threshold to the automatic one used (which is 0.5). The goal was to ensure that the model maximised its recall (predicted as many of the 1s correctly as possible) whilst making sure that it was not missclassifying the 0s too often.

So, to do this, we looped through a range of thresholds and for each one, we calculated the recall and precision, storing those results. Then we filtered for thresholds where recall was at least 0.9, and among those, chose the one with the highest precision. This allowed us to balance the trade off and pick a cutoff that performed better for the minority class.

```
thresholds <- seq(0.001, 0.5, by = 0.01)

results <- data.frame(threshold = numeric(), recall = numeric(), precision = numeric())

for (thresh in thresholds) {
  preds <- xgb_preds %>%
    mutate(pred_class = factor(if_else(.pred_1 >= thresh, "1", "0"), levels = c("0", "1")))

  rec <- recall_vec(truth = preds$y, estimate = preds$pred_class)
  prec <- precision_vec(truth = preds$y, estimate = preds$pred_class)

  results <- rbind(results, data.frame(threshold = thresh, recall = rec, precision = prec))
}

print(results)

##      threshold      recall  precision
## 1      0.001 0.9296520 0.9879421
## 2      0.011 0.9674735 0.9815810
## 3      0.021 0.9689864 0.9801071
## 4      0.031 0.9742814 0.9794677
## 5      0.041 0.9773071 0.9787879
## 6      0.051 0.9780635 0.9788039
## 7      0.061 0.9780635 0.9788039
## 8      0.071 0.9795764 0.9788360
## 9      0.081 0.9810893 0.9788679
## 10     0.091 0.9810893 0.9788679
## 11     0.101 0.9818457 0.9788839
## 12     0.111 0.9833585 0.9781791
## 13     0.121 0.9848714 0.9782119
## 14     0.131 0.9856278 0.9782282
## 15     0.141 0.9856278 0.9782282
## 16     0.151 0.9856278 0.9782282
## 17     0.161 0.9856278 0.9782282
## 18     0.171 0.9863843 0.9782446
## 19     0.181 0.9863843 0.9782446
## 20     0.191 0.9871407 0.9775281
## 21     0.201 0.9878971 0.9775449
## 22     0.211 0.9878971 0.9760837
## 23     0.221 0.9878971 0.9760837
## 24     0.231 0.9886536 0.9761016
## 25     0.241 0.9886536 0.9761016
## 26     0.251 0.9886536 0.9761016
## 27     0.261 0.9886536 0.9761016
## 28     0.271 0.9886536 0.9761016
## 29     0.281 0.9886536 0.9753731
## 30     0.291 0.9886536 0.9753731
## 31     0.301 0.9886536 0.9753731
## 32     0.311 0.9886536 0.9746458
```

```

## 33      0.321 0.9886536 0.9746458
## 34      0.331 0.9886536 0.9746458
## 35      0.341 0.9886536 0.9746458
## 36      0.351 0.9886536 0.9746458
## 37      0.361 0.9901664 0.9746835
## 38      0.371 0.9901664 0.9746835
## 39      0.381 0.9901664 0.9746835
## 40      0.391 0.9901664 0.9746835
## 41      0.401 0.9901664 0.9746835
## 42      0.411 0.9901664 0.9746835
## 43      0.421 0.9901664 0.9746835
## 44      0.431 0.9901664 0.9746835
## 45      0.441 0.9901664 0.9746835
## 46      0.451 0.9901664 0.9746835
## 47      0.461 0.9909228 0.9747024
## 48      0.471 0.9909228 0.9747024
## 49      0.481 0.9909228 0.9747024
## 50      0.491 0.9909228 0.9739777

# filter thresholds with recall >= 0.9

good_recall <- results %>% filter(recall >= 0.9)

if (nrow(good_recall) == 0) {
  cat("No threshold found with recall >= 0.9\n")
} else {
  # Pick threshold with highest precision among them
  best <- good_recall %>% filter(precision == max(precision))
  cat("Best threshold with recall >= 0.9:\n")
  print(best)
}

## Best threshold with recall >= 0.9:
##   threshold    recall  precision
## 1      0.001 0.929652 0.9879421

```

Conclusion

This project explored multiple classification models to predict corporate bankruptcy, balancing interpretability with predictive performance.

Traditional models like decision trees and random forests achieved high accuracy. We also implemented a high dimensional XGBoost model, tuning it for recall.

While accuracy across models was similar, XGBoost allowed us to focus on capturing bankrupt companies more reliably — a crucial factor in real-world financial risk settings.

Overall, we demonstrated the importance of model selection, threshold tuning, and evaluation beyond accuracy when dealing with imbalanced classification problems.