CAS Group 24

## Preprocessing (Copied from CAS Pricing 1)

library(readxl)  
library(dplyr)  
library(caret)  
library(pscl)  
  
dataset <- read\_excel("dataset.xlsx", sheet = 1)  
  
# Change all the "chr" to factor  
dataset <- dataset %>%  
 mutate(  
 across(  
 where(is.character), function(col) {  
 factor(col, levels = unique(col))  
 }  
 )  
 )  
  
# We use createDataPartition rather than sample[] is because createDataPartition keeps the distribution of the response variable.   
# From the example above we see approximate 5% of the response have 1 claim so the test set will also have approximate 5% of claim size of 1.  
# This train dataset will consist of 70% of the data from the full dataset  
set.seed(716)  
  
train\_index <- createDataPartition(dataset$ClaimNb,p = 0.7,list = FALSE)  
  
data\_train <- dataset[train\_index,]  
  
data\_test <- dataset[-train\_index,]  
  
# Construct Dataset for counting model comparison  
count\_cols <- c("Power", "CarAge", "DriverAge", "Brand", "Gas", "Region", "Density", "ClaimNb")  
  
claim\_number\_train\_set <- data\_train[,count\_cols]  
claim\_number\_test\_set <- data\_test[,count\_cols]  
  
claim\_number\_train\_scaled <- claim\_number\_train\_set  
claim\_number\_test\_scaled <- claim\_number\_test\_set  
  
# For these numerical variables, we change them into standardized data  
claim\_number\_train\_scaled[,c("CarAge","DriverAge","Density")] <- scale(claim\_number\_train\_scaled[,c("CarAge","DriverAge","Density")])  
  
claim\_number\_test\_scaled[,c("CarAge","DriverAge","Density")] <- scale(claim\_number\_test\_scaled[,c("CarAge","DriverAge","Density")])  
  
# We separate the modelling process into 2 part A|B:  
# The count component (A) models the number of claims (including the possibility of 0) using a Negative Binomial distribution to allow for over dispersion.  
# The zero-inflation component (B) models the probability that an observation is a guaranteed zero not due to the count process.  
zero\_nbinomial\_model <- zeroinfl(ClaimNb ~. | 1 + CarAge,   
 data = claim\_number\_train\_scaled,  
 dist = "negbin")  
  
# Hurdle Model does similar thing as the Zero-Inflated Model but the count part models positive count only (claim number of at least 1)  
hurdle\_model <- hurdle(ClaimNb ~.|., data = claim\_number\_train\_scaled, dist = "negbin")

## 1. Step-wise Variable Selection of the Zero Inflated Model

Variable selection is important as the choice of variable will ultimately affect its performance.

library(MASS)  
library(pscl)  
  
# Since there is no prebuilt function to do the stepwise selection for zero-inflated model so I will a general one.  
stepwise\_zeroinfl <- function(model, direction, use\_bic) {  
   
 # Returns model frame use to build the model  
 # Clean factor levels to prevent "factor level" errors  
 model\_data <- model.frame(model)  
 model\_data <- droplevels(model\_data)  
   
 # all.vars returns a vector with all the character of the variables used in the model and we use the [[1]] to extract the first element which is the response variable ClaimNb  
 response\_var <- all.vars(formula(model))[1]  
   
 # Create binary response for zero part  
 # model\_data[[response\_var]] == 0 will create a logical vector with TRUE and FALSE  
 # if claim number >0 then is TRUE, the as.numeric() will then change the TRUE to be 1 and FALSE to be 0 for the logistic regression  
 model\_data$zero\_response <- as.numeric(model\_data[[response\_var]] == 0)  
   
 # Other variables in the model except the response = all regressors  
 available\_vars <- setdiff(names(model\_data), c(response\_var, "zero\_response"))  
   
 # For the selection function  
 count\_formula\_init <- as.formula(paste(response\_var, "~ 1"))  
   
 count\_scope\_list = list(  
 lower = count\_formula\_init,  
 upper = as.formula(paste(response\_var, "~", paste(available\_vars,   
 collapse = " + ")))  
 )  
   
 # Set penalty (AIC vs BIC)  
 k\_val <- ifelse(use\_bic == FALSE, 2, log(nrow(model\_data)))  
   
   
 # Fit initial negative binomial model  
 count\_init <- glm.nb(formula = count\_formula\_init, data = model\_data,   
 na.action = na.exclude)  
   
 # Apply stepwise selection to count part  
 count\_optimized <- stepAIC(count\_init,   
 direction = direction,  
 scope = count\_scope\_list,  
 k = k\_val,   
 trace = TRUE)  
  
 # All variables from the count\_optimized excluding response is the selected variables  
 count\_final\_vars <- all.vars(formula(count\_optimized))[-1]  
   
   
 zero\_scope\_list = list(  
 lower = as.formula("zero\_response ~ 1"),  
 upper = as.formula(paste("zero\_response", "~", paste(available\_vars,   
 collapse = " + ")))  
 )  
   
 # Fit initial logistic model  
 zero\_init <- glm(zero\_response ~ 1,   
 data = model\_data,   
 family = binomial,   
 na.action = na.exclude)  
   
 zero\_optimized <- stepAIC(zero\_init,   
 direction = direction,   
 scope = zero\_scope\_list,  
 k = k\_val,   
 trace = TRUE)  
   
 zero\_final\_vars <- all.vars(formula(zero\_optimized))[-1]  
   
   
 # Combine and fit final Zero-Inflated model  
 cat("\n Final Fitting \n")  
   
 # Build final formula  
 count\_part <- ifelse(length(count\_final\_vars) > 0,   
 paste(count\_final\_vars, collapse = " + "),   
 "1")  
   
 zero\_part <- ifelse(length(zero\_final\_vars) > 0,   
 paste(zero\_final\_vars, collapse = " + "),   
 "1")  
   
 final\_formula\_str <- paste(response\_var, "~", count\_part, "|", zero\_part)  
   
 final\_formula <- as.formula(final\_formula\_str)  
   
 cat("Final formula:", final\_formula\_str, "\n")  
   
 # Fit final zero-inflated model  
 final\_model <- zeroinfl(formula = final\_formula,   
 data = model\_data,   
 dist = "negbin",  
 control = zeroinfl.control(method = "BFGS", maxit = 1000))  
   
 return(final\_model)  
}  
  
# Model Comparison General Function  
compare\_models <- function(original\_model, optimized\_model) {  
  
 cat("Original AIC:", AIC(original\_model), "\n")  
 cat("Optimized AIC:", AIC(optimized\_model), "\n")  
 cat("AIC improvement:", AIC(original\_model) - AIC(optimized\_model), "\n\n")  
   
 cat("Original BIC:", BIC(original\_model), "\n")  
 cat("Optimized BIC:", BIC(optimized\_model), "\n")  
 cat("BIC improvement:", BIC(original\_model) - BIC(optimized\_model), "\n\n")  
}

stepwise\_zeroinfNb <- stepwise\_zeroinfl(zero\_nbinomial\_model,   
 direction = "both",   
 use\_bic = FALSE)

Start: AIC=31536.67  
ClaimNb ~ 1  
  
 Df AIC  
+ CarAge 1 31386  
+ Region 9 31425  
+ Density 1 31477  
+ Gas 1 31500  
+ Brand 6 31510  
+ Power 11 31516  
+ DriverAge 1 31524  
<none> 31537  
  
Step: AIC=31386.11  
ClaimNb ~ CarAge  
  
 Df AIC  
+ Region 9 31289  
+ Density 1 31333  
+ Power 11 31366  
+ Gas 1 31366  
+ Brand 6 31370  
+ DriverAge 1 31372  
<none> 31386  
- CarAge 1 31537  
  
Step: AIC=31289.04  
ClaimNb ~ CarAge + Region  
  
 Df AIC  
+ Gas 1 31263  
+ Power 11 31272  
+ Density 1 31279  
+ DriverAge 1 31281  
+ Brand 6 31281  
<none> 31289  
- Region 9 31386  
- CarAge 1 31425  
  
Step: AIC=31262.83  
ClaimNb ~ CarAge + Region + Gas  
  
 Df AIC  
+ Density 1 31249  
+ Power 11 31253  
+ Brand 6 31255  
+ DriverAge 1 31259  
<none> 31263  
- Gas 1 31289  
- Region 9 31366  
- CarAge 1 31381  
  
Step: AIC=31248.75  
ClaimNb ~ CarAge + Region + Gas + Density  
  
 Df AIC  
+ Power 11 31240  
+ Brand 6 31242  
+ DriverAge 1 31244  
<none> 31249  
- Density 1 31263  
- Gas 1 31279  
- Region 9 31304  
- CarAge 1 31364  
  
Step: AIC=31240.41  
ClaimNb ~ CarAge + Region + Gas + Density + Power  
  
 Df AIC  
+ DriverAge 1 31236  
<none> 31240  
+ Brand 6 31242  
- Power 11 31249  
- Density 1 31253  
- Gas 1 31262  
- Region 9 31295  
- CarAge 1 31361  
  
Step: AIC=31235.67  
ClaimNb ~ CarAge + Region + Gas + Density + Power + DriverAge  
  
 Df AIC  
<none> 31236  
+ Brand 6 31238  
- DriverAge 1 31240  
- Power 11 31244  
- Density 1 31249  
- Gas 1 31254  
- Region 9 31286  
- CarAge 1 31357  
Start: AIC=30099.87  
zero\_response ~ 1  
  
 Df Deviance AIC  
+ CarAge 1 29947 29951  
+ Region 9 29983 30003  
+ Density 1 30041 30045  
+ Gas 1 30056 30060  
+ Brand 6 30056 30070  
+ Power 11 30057 30081  
+ DriverAge 1 30079 30083  
<none> 30098 30100  
  
Step: AIC=29950.99  
zero\_response ~ CarAge  
  
 Df Deviance AIC  
+ Region 9 29846 29868  
+ Density 1 29898 29904  
+ Gas 1 29923 29929  
+ Power 11 29906 29932  
+ Brand 6 29917 29933  
+ DriverAge 1 29927 29933  
<none> 29947 29951  
- CarAge 1 30098 30100  
  
Step: AIC=29867.55  
zero\_response ~ CarAge + Region  
  
 Df Deviance AIC  
+ Gas 1 29815 29839  
+ Power 11 29807 29851  
+ DriverAge 1 29832 29856  
+ Brand 6 29823 29857  
+ Density 1 29835 29859  
<none> 29846 29868  
- Region 9 29947 29951  
- CarAge 1 29983 30003  
  
Step: AIC=29838.69  
zero\_response ~ CarAge + Region + Gas  
  
 Df Deviance AIC  
+ Density 1 29800 29826  
+ Brand 6 29792 29828  
+ Power 11 29784 29830  
+ DriverAge 1 29806 29832  
<none> 29815 29839  
- Gas 1 29846 29868  
- Region 9 29923 29929  
- CarAge 1 29933 29955  
  
Step: AIC=29826.21  
zero\_response ~ CarAge + Region + Gas + Density  
  
 Df Deviance AIC  
+ Brand 6 29779 29817  
+ Power 11 29771 29819  
+ DriverAge 1 29791 29819  
<none> 29800 29826  
- Density 1 29815 29839  
- Gas 1 29835 29859  
- Region 9 29864 29872  
- CarAge 1 29916 29940  
  
Step: AIC=29816.71  
zero\_response ~ CarAge + Region + Gas + Density + Brand  
  
 Df Deviance AIC  
+ DriverAge 1 29771 29811  
<none> 29779 29817  
+ Power 11 29758 29818  
- Brand 6 29800 29826  
- Density 1 29792 29828  
- Gas 1 29813 29849  
- Region 9 29837 29857  
- CarAge 1 29887 29923  
  
Step: AIC=29810.58  
zero\_response ~ CarAge + Region + Gas + Density + Brand + DriverAge  
  
 Df Deviance AIC  
<none> 29771 29811  
+ Power 11 29749 29811  
- DriverAge 1 29779 29817  
- Brand 6 29791 29819  
- Density 1 29785 29823  
- Gas 1 29800 29838  
- Region 9 29826 29848  
- CarAge 1 29881 29919  
  
 Final Fitting   
Final formula: ClaimNb ~ CarAge + Region + Gas + Density + Power + DriverAge | CarAge + Region + Gas + Density + Brand + DriverAge

Warning in value[[3L]](cond): system is computationally singular: reciprocal  
condition number = 2.41426e-35FALSE

summary(stepwise\_zeroinfNb)

Call:  
zeroinfl(formula = final\_formula, data = model\_data, dist = "negbin",   
 control = zeroinfl.control(method = "BFGS", maxit = 1000))  
  
Pearson residuals:  
 Min 1Q Median 3Q Max   
-0.4738 -0.2431 -0.2149 -0.1853 14.1657   
  
Count model coefficients (negbin with log link):  
 Estimate Std. Error z value Pr(>|z|)  
(Intercept) -2.607679 NA NA NA  
CarAge -0.099012 NA NA NA  
RegionAquitaine 0.249528 NA NA NA  
RegionBretagne 0.014284 NA NA NA  
RegionPays-de-la-Loire 0.150679 NA NA NA  
RegionIle-de-France 0.135969 NA NA NA  
RegionBasse-Normandie -0.125166 NA NA NA  
RegionPoitou-Charentes 0.103417 NA NA NA  
RegionHaute-Normandie -0.457946 NA NA NA  
RegionNord-Pas-de-Calais 0.006223 NA NA NA  
RegionLimousin 1.030638 NA NA NA  
GasRegular -0.148579 NA NA NA  
Density 0.012488 NA NA NA  
Powerd -0.150473 NA NA NA  
Powerg -0.041692 NA NA NA  
Powerf -0.044682 NA NA NA  
Powerk 0.249613 NA NA NA  
Powere -0.087928 NA NA NA  
Poweri 0.080907 NA NA NA  
Powerj 0.202730 NA NA NA  
Powero 0.269191 NA NA NA  
Powerl 0.069492 NA NA NA  
Powerm 0.134462 NA NA NA  
Powern 0.471090 NA NA NA  
DriverAge -0.068066 NA NA NA  
Log(theta) 24.596808 NA NA NA  
  
Zero-inflation model coefficients (binomial with logit link):  
 Estimate Std. Error z value Pr(>|z|)  
(Intercept) -2.45270 NA NA NA  
CarAge 0.34953 NA NA NA  
RegionAquitaine 0.01013 NA NA NA  
RegionBretagne -0.11695 NA NA NA  
RegionPays-de-la-Loire -0.05044 NA NA NA  
RegionIle-de-France -0.24209 NA NA NA  
RegionBasse-Normandie -27.29734 NA NA NA  
RegionPoitou-Charentes 0.36080 NA NA NA  
RegionHaute-Normandie -54.03000 NA NA NA  
RegionNord-Pas-de-Calais -1.27894 NA NA NA  
RegionLimousin 1.04030 NA NA NA  
GasRegular 0.22302 NA NA NA  
Density -6.85593 NA NA NA  
BrandFiat -0.22170 NA NA NA  
BrandOpel, General Motors or Ford -0.38682 NA NA NA  
BrandVolkswagen, Audi, Skoda or Seat 0.12509 NA NA NA  
BrandJapanese (except Nissan) or Korean -0.38250 NA NA NA  
BrandMercedes, Chrysler or BMW -0.07448 NA NA NA  
Brandother -0.26151 NA NA NA  
DriverAge -0.14016 NA NA NA  
  
Theta = 48112510528.0258   
Number of iterations in BFGS optimization: 127   
Log-likelihood: -1.554e+04 on 46 Df

compare\_models(zero\_nbinomial\_model, stepwise\_zeroinfNb)

Original AIC: 31242   
Optimized AIC: 31166.48   
AIC improvement: 75.5142   
  
Original BIC: 31557.04   
Optimized BIC: 31592.72   
BIC improvement: -35.67966

The model above shows a highly collinearity issue present in the model which is detected by by some of test from mctest().

library(mctest)  
  
mctest(zero\_nbinomial\_model)

Call:  
omcdiag(mod = mod, Inter = TRUE, detr = detr, red = red, conf = conf,   
 theil = theil, cn = cn)  
  
  
Overall Multicollinearity Diagnostics  
  
 MC Results detection  
Determinant |X'X|: 0.0439 0  
Farrar Chi-Square: 244146.1132 1  
Red Indicator: 0.0614 0  
Sum of Lambda Inverse: 46.7480 0  
Theil's Method: 6.0545 1  
Condition Number: 11.8059 0  
  
1 --> COLLINEARITY is detected by the test   
0 --> COLLINEARITY is not detected by the test

## 2. Step-wise Variable Selection for Hurdle Model

library(MASS)  
library(pscl)  
  
stepwise\_hurdle <- function(model, direction, use\_bic) {  
 model\_data <- model.frame(model)  
 model\_data <- droplevels(model\_data)  
 response\_var <- all.vars(formula(model))[1]  
   
 # For hurdle model: binary response is whether response > 0  
 # This is the key difference from zero-inflated models  
 model\_data$hurdle\_response <- as.numeric(model\_data[[response\_var]] > 0)  
   
 available\_vars <- setdiff(names(model\_data), c(response\_var, "hurdle\_response"))  
   
 k\_val <- ifelse(use\_bic == FALSE, 2, log(nrow(model\_data)))  
   
 # The Hurdle Part 0 vs >0  
 hurdle\_scope\_list = list(  
 lower = as.formula("hurdle\_response ~ 1"),  
 upper = as.formula(paste("hurdle\_response", "~", paste(available\_vars,   
 collapse = " + ")))  
 )  
   
 # Fit initial logistic model for hurdle part  
 # Fit initial logistic model with better control parameters  
 hurdle\_init <- glm(hurdle\_response ~ 1,   
 data = model\_data,   
 family = binomial(link = "logit"),  
 control = glm.control(maxit = 1000, epsilon = 1e-8))  
   
 # Perform stepwise selection with error handling  
 # Try Catch will run the second expression if the first expression shows error  
 hurdle\_optimized <- tryCatch({  
 stepAIC(hurdle\_init,   
 direction = direction,   
 scope = hurdle\_scope\_list,  
 k = k\_val,   
 trace = TRUE,  
 steps = 1000)  
 }, error = function(e) {  
 cat("Error in hurdle stepwise selection:", e$message, "\n")  
 return(hurdle\_init)  
 })  
   
 hurdle\_final\_vars <- all.vars(formula(hurdle\_optimized))[-1]  
   
 # As for the Count Part we have to filter data to include only positive values  
 # Only positive observation is used  
 positive\_data <- model\_data[model\_data[[response\_var]] > 0, ]  
   
 count\_formula\_init <- as.formula(paste(response\_var, "~ 1"))  
 count\_scope\_list = list(  
 lower = count\_formula\_init,  
 upper = as.formula(paste(response\_var, "~", paste(available\_vars,   
 collapse = " + ")))  
 )  
   
 # Initial Count Model  
 count\_init <- tryCatch({  
 glm.nb(formula = count\_formula\_init,   
 data = positive\_data,  
 control = glm.control(maxit = 1000, epsilon = 1e-8),  
 init.theta = 1,  
 link = log)  
 }, error = function(e) {  
 cat("Error fitting initial count model, trying Poisson:", e$message, "\n")  
 glm(formula = count\_formula\_init,   
 data = positive\_data,   
 family = poisson,  
 control = glm.control(maxit = 1000))  
 })  
   
 # Perform stepwise selection for count part if error then return the initial model  
 count\_optimized <- tryCatch({  
 stepAIC(count\_init,   
 direction = direction,  
 scope = count\_scope\_list,  
 k = k\_val,   
 trace = TRUE,  
 steps = 1000)  
 }, error = function(e) {  
 cat("Error in count stepwise selection:", e$message, "\n")  
 return(count\_init)  
 })  
   
 count\_final\_vars <- all.vars(formula(count\_optimized))[-1]  
  
   
 # Build final formula  
 hurdle\_part <- ifelse(length(hurdle\_final\_vars) > 0,   
 paste(hurdle\_final\_vars, collapse = " + "),   
 "1")  
   
 count\_part <- ifelse(length(count\_final\_vars) > 0,   
 paste(count\_final\_vars, collapse = " + "),   
 "1")  
   
 final\_formula\_str <- paste(response\_var, "~", count\_part, "|", hurdle\_part)  
   
 final\_formula <- as.formula(final\_formula\_str)  
   
 cat("Final hurdle formula:", final\_formula\_str, "\n")  
   
 # Fit final hurdle model  
 final\_model <- tryCatch({  
 hurdle(formula = final\_formula,   
 data = model\_data,   
 dist = "negbin",  
 zero.dist = "binomial",  
 control = hurdle.control(method = "BFGS",   
 maxit = 1000,   
 reltol = 1e-12,  
 start = NULL))  
 }, error = function(e) {  
 cat("Error fitting hurdle model with negbin, trying poisson:", e$message, "\n")  
 hurdle(formula = final\_formula,   
 data = model\_data,   
 dist = "poisson",  
 zero.dist = "binomial",  
 control = hurdle.control(method = "BFGS", maxit = 1000))  
 })  
   
 return(final\_model)  
}  
  
# Model Comparison function for hurdle models  
compare\_hurdle\_models <- function(original\_model, optimized\_model) {  
 cat("=== Hurdle Model Comparison ===\n")  
 cat("Original AIC:", round(AIC(original\_model), 2), "\n")  
 cat("Optimized AIC:", round(AIC(optimized\_model), 2), "\n")  
 cat("AIC improvement:", round(AIC(original\_model) - AIC(optimized\_model), 2), "\n\n")  
   
 cat("Original BIC:", round(BIC(original\_model), 2), "\n")  
 cat("Optimized BIC:", round(BIC(optimized\_model), 2), "\n")  
 cat("BIC improvement:", round(BIC(original\_model) - BIC(optimized\_model), 2), "\n\n")  
}

stepwise\_hurdle\_model <- stepwise\_hurdle(hurdle\_model,   
 direction = "both",   
 use\_bic = FALSE)

Start: AIC=30099.87  
hurdle\_response ~ 1  
  
 Df Deviance AIC  
+ CarAge 1 29947 29951  
+ Region 9 29983 30003  
+ Density 1 30041 30045  
+ Gas 1 30056 30060  
+ Brand 6 30056 30070  
+ Power 11 30057 30081  
+ DriverAge 1 30079 30083  
<none> 30098 30100  
  
Step: AIC=29950.99  
hurdle\_response ~ CarAge  
  
 Df Deviance AIC  
+ Region 9 29846 29868  
+ Density 1 29898 29904  
+ Gas 1 29923 29929  
+ Power 11 29906 29932  
+ Brand 6 29917 29933  
+ DriverAge 1 29927 29933  
<none> 29947 29951  
- CarAge 1 30098 30100  
  
Step: AIC=29867.55  
hurdle\_response ~ CarAge + Region  
  
 Df Deviance AIC  
+ Gas 1 29815 29839  
+ Power 11 29807 29851  
+ DriverAge 1 29832 29856  
+ Brand 6 29823 29857  
+ Density 1 29835 29859  
<none> 29846 29868  
- Region 9 29947 29951  
- CarAge 1 29983 30003  
  
Step: AIC=29838.69  
hurdle\_response ~ CarAge + Region + Gas  
  
 Df Deviance AIC  
+ Density 1 29800 29826  
+ Brand 6 29792 29828  
+ Power 11 29784 29830  
+ DriverAge 1 29806 29832  
<none> 29815 29839  
- Gas 1 29846 29868  
- Region 9 29923 29929  
- CarAge 1 29933 29955  
  
Step: AIC=29826.21  
hurdle\_response ~ CarAge + Region + Gas + Density  
  
 Df Deviance AIC  
+ Brand 6 29779 29817  
+ Power 11 29771 29819  
+ DriverAge 1 29791 29819  
<none> 29800 29826  
- Density 1 29815 29839  
- Gas 1 29835 29859  
- Region 9 29864 29872  
- CarAge 1 29916 29940  
  
Step: AIC=29816.71  
hurdle\_response ~ CarAge + Region + Gas + Density + Brand  
  
 Df Deviance AIC  
+ DriverAge 1 29771 29811  
<none> 29779 29817  
+ Power 11 29758 29818  
- Brand 6 29800 29826  
- Density 1 29792 29828  
- Gas 1 29813 29849  
- Region 9 29837 29857  
- CarAge 1 29887 29923  
  
Step: AIC=29810.58  
hurdle\_response ~ CarAge + Region + Gas + Density + Brand + DriverAge  
  
 Df Deviance AIC  
<none> 29771 29811  
+ Power 11 29749 29811  
- DriverAge 1 29779 29817  
- Brand 6 29791 29819  
- Density 1 29785 29823  
- Gas 1 29800 29838  
- Region 9 29826 29848  
- CarAge 1 29881 29919  
Start: AIC=2  
ClaimNb ~ 1  
  
 Df AIC  
<none> 2  
+ Power 11 24  
+ CarAge   
+ DriverAge   
+ Brand   
+ Gas   
+ Region   
+ Density   
Final hurdle formula: ClaimNb ~ 1 | CarAge + Region + Gas + Density + Brand + DriverAge

summary(stepwise\_hurdle\_model)

Call:  
hurdle(formula = final\_formula, data = model\_data, dist = "negbin", zero.dist = "binomial",   
 control = hurdle.control(method = "BFGS", maxit = 1000, reltol = 1e-12,   
 start = NULL))  
  
Pearson residuals:  
 Min 1Q Median 3Q Max   
-0.4420 -0.2352 -0.2126 -0.1910 17.2246   
  
Count model coefficients (truncated negbin with log link):  
 Estimate Std. Error z value Pr(>|z|)   
(Intercept) -2.38144 0.07481 -31.83 <2e-16 \*\*\*  
Log(theta) 11.68069 NaN NaN NaN   
Zero hurdle model coefficients (binomial with logit link):  
 Estimate Std. Error z value Pr(>|z|)  
(Intercept) -3.06261 0.03179 -96.336 < 2e-16  
CarAge -0.19350 0.01886 -10.261 < 2e-16  
RegionAquitaine 0.32809 0.07611 4.311 1.63e-05  
RegionBretagne 0.10066 0.05065 1.987 0.046894  
RegionPays-de-la-Loire 0.28101 0.05631 4.990 6.03e-07  
RegionIle-de-France 0.31495 0.08146 3.866 0.000110  
RegionBasse-Normandie 0.24546 0.09322 2.633 0.008460  
RegionPoitou-Charentes 0.04378 0.08155 0.537 0.591380  
RegionHaute-Normandie -0.04385 0.18871 -0.232 0.816266  
RegionNord-Pas-de-Calais 0.28728 0.09255 3.104 0.001910  
RegionLimousin 0.41635 0.16181 2.573 0.010079  
GasRegular -0.18796 0.03465 -5.425 5.80e-08  
Density 0.06636 0.01739 3.815 0.000136  
BrandFiat 0.01478 0.08352 0.177 0.859520  
BrandOpel, General Motors or Ford 0.11753 0.05436 2.162 0.030610  
BrandVolkswagen, Audi, Skoda or Seat 0.04857 0.06166 0.788 0.430924  
BrandJapanese (except Nissan) or Korean 0.32821 0.08735 3.757 0.000172  
BrandMercedes, Chrysler or BMW 0.11263 0.07752 1.453 0.146261  
Brandother 0.20555 0.09834 2.090 0.036605  
DriverAge -0.04982 0.01751 -2.845 0.004444  
   
(Intercept) \*\*\*  
CarAge \*\*\*  
RegionAquitaine \*\*\*  
RegionBretagne \*   
RegionPays-de-la-Loire \*\*\*  
RegionIle-de-France \*\*\*  
RegionBasse-Normandie \*\*   
RegionPoitou-Charentes   
RegionHaute-Normandie   
RegionNord-Pas-de-Calais \*\*   
RegionLimousin \*   
GasRegular \*\*\*  
Density \*\*\*  
BrandFiat   
BrandOpel, General Motors or Ford \*   
BrandVolkswagen, Audi, Skoda or Seat   
BrandJapanese (except Nissan) or Korean \*\*\*  
BrandMercedes, Chrysler or BMW   
Brandother \*   
DriverAge \*\*   
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1   
  
Theta: count = 118265.7277  
Number of iterations in BFGS optimization: 987   
Log-likelihood: -1.56e+04 on 22 Df

compare\_models(hurdle\_model,stepwise\_hurdle\_model)

Original AIC: 31265   
Optimized AIC: 31249.41   
AIC improvement: 15.59724   
  
Original BIC: 31848.77   
Optimized BIC: 31453.26   
BIC improvement: 395.5096

## 3. XGBoost Model

### 1. Data Preprocessing

library(caret)  
library(dplyr)  
  
set.seed(719)  
  
train\_xgindex <- createDataPartition(dataset$ClaimNb,p = 0.7,list = FALSE)  
  
data\_xgtrain <- dataset[train\_xgindex,]  
  
data\_xgtest <- dataset[-train\_xgindex,]  
  
# Drop PolicyID  
data\_xgtrain <- data\_xgtrain[,-1]  
  
data\_xgtest <- data\_xgtest[,-1]  
  
# After that we create an additional variable "Individual Claim Size"  
# Then drop the ClaimAmount  
data\_xgtrain <- data\_xgtrain %>%  
 mutate(  
 Individual\_Claim\_Size = ifelse(ClaimNb > 0,ClaimAmount/ClaimNb,0)  
 ) %>%  
 dplyr::select(-ClaimAmount)  
  
data\_xgtest <- data\_xgtest %>%  
 mutate(  
 Individual\_Claim\_Size = ifelse(ClaimNb >0, ClaimAmount/ClaimNb, 0)  
 ) %>%  
 dplyr::select(-ClaimAmount)  
  
str(data\_xgtrain)

tibble [78,132 × 9] (S3: tbl\_df/tbl/data.frame)  
 $ Power : Factor w/ 12 levels "h","d","g","f",..: 2 3 1 2 2 3 4 5 4 6 ...  
 $ CarAge : num [1:78132] 10 25 9 10 1 25 4 29 3 6 ...  
 $ DriverAge : num [1:78132] 71 87 56 63 73 55 69 69 72 80 ...  
 $ Brand : Factor w/ 7 levels "Renault, Nissan or Citroen",..: 1 1 2 1 1 1 1 1 1 1 ...  
 $ Gas : Factor w/ 2 levels "Diesel","Regular": 2 2 1 2 2 1 2 2 1 1 ...  
 $ Region : Factor w/ 10 levels "Centre","Aquitaine",..: 1 1 2 1 1 1 1 1 1 1 ...  
 $ Density : num [1:78132] 91 18 272 10 44 16 35 101 11 51 ...  
 $ ClaimNb : num [1:78132] 0 0 1 0 0 0 0 0 0 0 ...  
 $ Individual\_Claim\_Size: num [1:78132] 0 0 1147 0 0 ...

str(data\_xgtest)

tibble [33,485 × 9] (S3: tbl\_df/tbl/data.frame)  
 $ Power : Factor w/ 12 levels "h","d","g","f",..: 1 6 4 4 4 4 4 3 2 3 ...  
 $ CarAge : num [1:33485] 7 13 6 6 4 6 4 14 19 7 ...  
 $ DriverAge : num [1:33485] 55 50 52 72 80 54 72 55 67 54 ...  
 $ Brand : Factor w/ 7 levels "Renault, Nissan or Citroen",..: 1 1 3 1 1 3 1 1 1 3 ...  
 $ Gas : Factor w/ 2 levels "Diesel","Regular": 1 1 2 1 1 2 1 1 2 2 ...  
 $ Region : Factor w/ 10 levels "Centre","Aquitaine",..: 1 1 1 4 3 3 1 1 1 1 ...  
 $ Density : num [1:33485] 67 13 1943 56 229 ...  
 $ ClaimNb : num [1:33485] 0 0 0 0 0 0 0 0 0 0 ...  
 $ Individual\_Claim\_Size: num [1:33485] 0 0 0 0 0 0 0 0 0 0 ...

### 2. XGboost Model Matrix Generating Function

library(Matrix)  
  
prepare\_xgb\_data <- function(data, target = NULL) {  
 # One-hot encoding  
 # Exclude target variables from feature matrix  
 feature\_cols <- setdiff(names(data), c("ClaimNb", "Individual\_Claim\_Size"))  
   
 model\_matrix <- sparse.model.matrix(~ . - 1, data = data[, feature\_cols])  
 # Exclude the intercept term  
   
 # Create DMatrix  
 # If the DMatrix is used for prediction then it will not need a target  
 if(!is.null(target)) {  
 xgb.DMatrix(data = model\_matrix, label = target)  
 } else {  
 xgb.DMatrix(data = model\_matrix)  
 }  
}

### 3. Frequency Model

library(xgboost)  
  
dtrain\_freq <- prepare\_xgb\_data(  
 data\_xgtrain,  
 target = data\_xgtrain$ClaimNb  
)  
  
# Train frequency model  
xgb\_freq <- xgb.train(  
 params = list(  
 objective = "count:poisson",  
 eval\_metric = "rmse",  
 max\_depth = 6,  
 eta = 0.1,  
 subsample = 0.8,  
 colsample\_bytree = 0.8  
 ),  
 data = dtrain\_freq,  
 nrounds = 2500,  
 early\_stopping\_rounds = 50,  
 watchlist = list(train = dtrain\_freq),  
 print\_every\_n = 500,  
 verbose = 1  
)

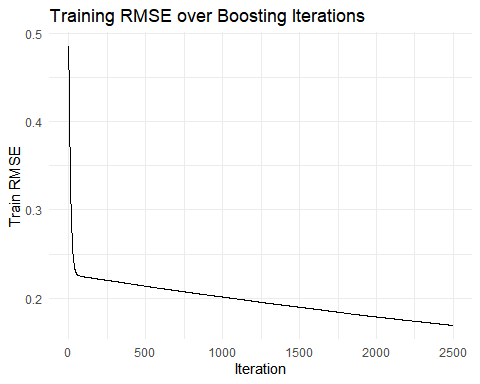
[1] train-rmse:0.484742   
Will train until train\_rmse hasn't improved in 50 rounds.  
  
[501] train-rmse:0.213602   
[1001] train-rmse:0.201409   
[1501] train-rmse:0.189877   
[2001] train-rmse:0.178744   
[2500] train-rmse:0.169211

#### Model Evaluation

eval\_log <- xgb\_freq$evaluation\_log  
tail(eval\_log)

iter train\_rmse  
 <num> <num>  
1: 2495 0.1692565  
2: 2496 0.1692312  
3: 2497 0.1692330  
4: 2498 0.1692262  
5: 2499 0.1692256  
6: 2500 0.1692114

library(ggplot2)  
ggplot(eval\_log, aes(x = iter, y = train\_rmse)) +  
 geom\_line() +  
 labs(  
 title = "Training RMSE over Boosting Iterations",  
 x = "Iteration",  
 y = "Train RMSE"  
 ) +  
 theme\_minimal()



### 4. Severity Model

# We only take the ones with claim size > 0  
severity\_train <- data\_xgtrain[data\_xgtrain$Individual\_Claim\_Size > 0, ]  
  
dtrain\_sev <- prepare\_xgb\_data(  
 severity\_train,  
 target = log(severity\_train$Individual\_Claim\_Size)   
 # Log-transform for skewness  
 )  
   
 # Train severity model  
 xgb\_sev <- xgb.train(  
 params = list(  
 objective = "reg:squarederror", # Regression for continuous target  
 eval\_metric = "rmse",  
 max\_depth = 6,  
 eta = 0.1,  
 subsample = 0.8,  
 colsample\_bytree = 0.8  
 ),  
 data = dtrain\_sev,  
 nrounds = 2500,  
 early\_stopping\_rounds = 50,  
 watchlist = list(train = dtrain\_sev),  
 print\_every\_n = 500,  
 verbose = 1  
 )

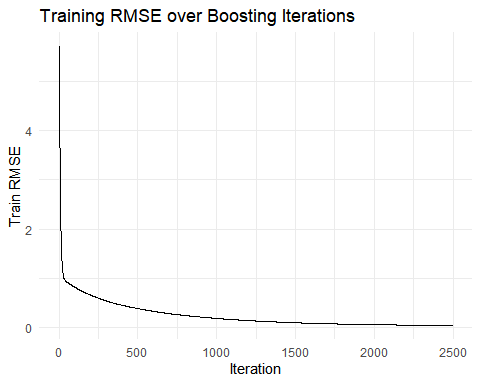
[1] train-rmse:5.700563   
Will train until train\_rmse hasn't improved in 50 rounds.  
  
[501] train-rmse:0.388428   
[1001] train-rmse:0.189411   
[1501] train-rmse:0.100865   
[2001] train-rmse:0.062545   
[2500] train-rmse:0.045726

#### Model Evaluation

eval\_log\_sev <- xgb\_sev$evaluation\_log  
tail(eval\_log\_sev)

iter train\_rmse  
 <num> <num>  
1: 2495 0.04581411  
2: 2496 0.04578044  
3: 2497 0.04574660  
4: 2498 0.04571921  
5: 2499 0.04575261  
6: 2500 0.04572587

ggplot(eval\_log\_sev, aes(x = iter, y = train\_rmse)) +  
 geom\_line() +  
 labs(  
 title = "Training RMSE over Boosting Iterations",  
 x = "Iteration",  
 y = "Train RMSE"  
 ) +  
 theme\_minimal()



## 4. Pure Premium Calculation

library(readxl)  
library(dplyr)  
library(xgboost)  
# We will predict the premiums along with the test set to prevent error of incomplete variables (The dataset that we are going to apply the model have missing variables factors for example, brandFiat)  
  
data\_xgpredict <- read\_xlsx("dataset.xlsx", sheet = 2)  
  
names(data\_xgpredict)

[1] "PolicyID" "Power" "CarAge" "DriverAge" "Brand" "Gas"   
[7] "Region" "Density"

# Add-In and remove variables so that names(data\_xgpredict) = names(data\_xgtest)  
data\_xgpredict <- data\_xgpredict %>%  
 mutate(  
 ClaimNb = 0,  
 Individual\_Claim\_Size = 0  
 ) %>%  
 dplyr::select(-PolicyID)  
  
  
data\_xgtest <- rbind(data\_xgtest,data\_xgpredict)  
  
# Apply the model and Predict frequency  
# The test set is used for prediction, we do not set a target  
dtest\_freq <- prepare\_xgb\_data(data\_xgtest)  
freq\_pred\_test <- predict(xgb\_freq, dtest\_freq)  
  
  
# Predict severity on test data  
dtest\_sev\_all <- prepare\_xgb\_data(data\_xgtest)  
sev\_pred\_test\_log <- predict(xgb\_sev, dtest\_sev\_all)  
sev\_pred\_test <- exp(sev\_pred\_test\_log)   
# Convert back to original scale since we log-transform  
  
  
# Pure Premium = Frequency × Severity  
pure\_premium\_test <- freq\_pred\_test \* sev\_pred\_test  
  
# Add predictions to the data frame  
results\_test <- data\_xgtest %>%  
 mutate(  
 predicted\_frequency = freq\_pred\_test,  
 predicted\_severity = sev\_pred\_test,  
 predicted\_pure\_premium = pure\_premium\_test,  
 actual\_pure\_premium = ClaimNb \* Individual\_Claim\_Size  
 )  
  
head(results\_test)

# A tibble: 6 × 13  
 Power CarAge DriverAge Brand Gas Region Density ClaimNb  
 <fct> <dbl> <dbl> <fct> <fct> <fct> <dbl> <dbl>  
1 h 7 55 Renault, Nissan or Citroen Dies… Centre 67 0  
2 e 13 50 Renault, Nissan or Citroen Dies… Centre 13 0  
3 f 6 52 Opel, General Motors or F… Regu… Centre 1943 0  
4 f 6 72 Renault, Nissan or Citroen Dies… Pays-… 56 0  
5 f 4 80 Renault, Nissan or Citroen Dies… Breta… 229 0  
6 f 6 54 Opel, General Motors or F… Regu… Breta… 125 0  
# ℹ 5 more variables: Individual\_Claim\_Size <dbl>, predicted\_frequency <dbl>,  
# predicted\_severity <dbl>, predicted\_pure\_premium <dbl>,  
# actual\_pure\_premium <dbl>

library(writexl)  
  
results\_write <- tail(results\_test, 10000)  
  
# write\_xlsx(results\_write, path = "premiums\_test.xlsx")

## 5. Determination of Loading Factors

### 1. Calculate Parameters

library(dplyr)  
library(fitdistrplus)   
# For distribution fitting  
library(MASS)   
# For additional distributions  
  
 # Frequency  
 freq\_data <- dataset$ClaimNb   
 # Fit Poisson using MLE  
 pois\_fit <- fitdist(freq\_data, "pois", method = "mle")  
 freq\_distribution <- "poisson"  
 rate\_lambda <- pois\_fit$estimate[1] # lambda parameter  
   
 cat("Poisson Parameters (MLE):\n")

Poisson Parameters (MLE):

cat("- lambda:", round(rate\_lambda, 4), "\n")

- lambda: 0.0506

cat("- AIC:", round(pois\_fit$aic, 2), "\n")

- AIC: 45352.38

cat("- Log-likelihood:", round(pois\_fit$loglik, 2), "\n")

- Log-likelihood: -22675.19

# Severity  
 # Get positive claims only  
 severity\_data <- dataset %>%  
 filter(ClaimNb > 0) %>%  
 mutate(Individual\_Claim\_Size = ClaimAmount / ClaimNb) %>%  
 pull(Individual\_Claim\_Size)  
   
 # Fit log normal distribution using MLE  
 lnorm\_fit <- fitdist(severity\_data, "lnorm", method = "mle")  
 lnorm\_params <- list(meanlog = lnorm\_fit$estimate[1], sdlog =  
 lnorm\_fit$estimate[2])  
   
 cat("Lognormal Parameters (MLE):\n")

Lognormal Parameters (MLE):

cat("- meanlog:", round(lnorm\_params$meanlog, 4), "\n")

- meanlog: 6.7187

cat("- sdlog:", round(lnorm\_params$sdlog, 4), "\n")

- sdlog: 1.0693

cat("- AIC:", round(lnorm\_fit$aic, 2), "\n")

- AIC: 88843.93

cat("- Log-likelihood:", round(lnorm\_fit$loglik, 2), "\n")

- Log-likelihood: -44419.96

### 2. K-Means Clustering to Assign a Risk Tier

library(dplyr)  
library(tibble)  
  
drivers <- results\_write %>%  
 # Drop the part we append previously  
 rename(  
 freq = predicted\_frequency,  
 sev = predicted\_severity,  
 prem = predicted\_pure\_premium  
 ) %>%  
 dplyr::select(freq, sev, prem, Brand, Region)  
  
# One‑hot encode all categorical factors  
k\_means\_mm <- model.matrix(~ . - 1, data = drivers)  
  
# Scale each column to mean=0, sd=1  
k\_means\_mm\_scaled <- scale(k\_means\_mm)  
  
# Run K‑Means Clustering and cluster all policies into 3 categories  
set.seed(720)  
km3 <- kmeans(k\_means\_mm\_scaled, centers = 3, nstart = 25)  
  
# Map clusters with ordered Tier labels A/B/C  
# The km3$cluster is actually a vector with 1,2,3 for the 30000+ policies in the test set # Since the clusters 1 2 3 does not have a ordinal order so it is reasonable to use the premium as the reference, the tapply calculates the mean for each clusters  
# tapply(X, INDEX, FUN) does: 1. Splits the vector X into groups according to the factor or integer vector INDEX. 2. Applies the function FUN (here mean) to each group. 3. Returns a vector whose names are the unique values of INDEX (1,2,3) and whose values are mean(X[INDEX == i])  
# Arrange will then sort them from the lowest to highest and then assign tier factor A,B and C  
cluster\_map <- tibble(  
 cluster\_raw = 1:3,  
 mean\_prem = tapply(drivers$prem, km3$cluster, mean)  
) %>%  
 arrange(mean\_prem) %>%  
 mutate(Tier = c("A","B","C"))  
  
# Attach Tier back to portfolio frame  
port <- results\_write %>%  
 mutate(cluster\_raw = km3$cluster) %>%  
 left\_join(cluster\_map, by = "cluster\_raw") %>%  
 dplyr::select(-cluster\_raw, -mean\_prem)  
# now port$Tier is A/B/C based on cluster, using brand & other factors too

The cluster\_map shall look like:

|  | cluster\_raw | mean\_prem | Tier |
| --- | --- | --- | --- |
| 1 | 2 | 43.14 | A |
| 2 | 3 | 47.99 | B |
| 3 | 1 | 49.87 | C |

The function left\_join(df1, df2, by = “key) combines 2 data frames by keeping all the rows in df1, and adds matching columns from df2 based on a common key column.

The results\_test we add a column that marks each policy in the data frame and then left join the data frame cluster\_map, after removing the cluster\_raw and mean\_prem all we left with is the Tier

### 3. Risk Loading Optimization

#### Simulation and Objective Function

library(rBayesianOptimization)  
  
# Fitted distribution parameters  
lambda\_hat <- rate\_lambda  
  
lnorm\_params\_vec <- unname(unlist(lnorm\_params))  
meanlog\_hat <- lnorm\_params\_vec[1]  
sdlog\_hat <- lnorm\_params\_vec[2]  
  
# Monte Carlo simulation  
run\_simulation <- function(port, loadings, n\_sim) {  
 lam\_vec <- lambda\_hat  
 base\_prem <- port$predicted\_pure\_premium  
 tiers <- port$Tier  
  
 profits <- replicate(n\_sim, {  
 # Poisson Claim Count  
 n\_claims <- rpois(nrow(port), lam\_vec)  
  
 # Log‑normal severities  
 losses <- sapply(n\_claims, function(k) {  
 if (k > 0) sum(rlnorm(k, meanlog = meanlog\_hat, sdlog = sdlog\_hat))  
 else 0  
 })  
  
 # Underwriting profit  
 total\_prem <- sum(base\_prem \* loadings[tiers])  
 total\_loss <- sum(losses)  
 total\_prem - total\_loss  
 })  
  
 list(  
 ruin\_prob = mean(profits < 0),  
 expected\_profit = mean(profits)  
 )  
}  
  
# Objective function  
target\_ruin <- 0.005  
  
objective\_function <- function(load\_A, load\_B, load\_C) {  
   
 loads <- c(A = load\_A, B = load\_B, C = load\_C)  
   
 sim <- run\_simulation(port, loads, n\_sim = 2000)  
   
 ruin\_excess <- max(0, sim$ruin\_prob - target\_ruin)  
   
 penalty <- 10000\*ruin\_excess^2  
  
 score <- - penalty  
   
 return(list(Score = score))  
}

#### Bayesian Optimization

# Optimization setup  
set.seed(1206)  
opt\_res\_be <- BayesianOptimization(  
 FUN = objective\_function,  
 bounds = list(  
 load\_A = c(1.00, 1.50),  
 load\_B = c(1.50, 2.50),  
 load\_C = c(2.00, 3.00)  
 ),  
 init\_points = 5,  
 n\_iter = 10,  
 acq = "ei",  
 verbose = TRUE  
)

elapsed = 20.14 Round = 1 load\_A = 1.134894 load\_B = 1.702678 load\_C = 2.25257 Value = -2525.0625   
elapsed = 19.92 Round = 2 load\_A = 1.467308 load\_B = 1.584364 load\_C = 2.299564 Value = -1789.2900   
elapsed = 20.38 Round = 3 load\_A = 1.141987 load\_B = 1.929397 load\_C = 2.513471 Value = -80.1025   
elapsed = 19.75 Round = 4 load\_A = 1.235948 load\_B = 1.593796 load\_C = 2.370048 Value = -2714.4100   
elapsed = 19.64 Round = 5 load\_A = 1.354807 load\_B = 2.36714 load\_C = 2.886838 Value = 0.0000   
elapsed = 19.69 Round = 6 load\_A = 1.199918 load\_B = 2.105935 load\_C = 2.663799 Value = -1.1025   
elapsed = 19.86 Round = 7 load\_A = 1.0000 load\_B = 2.022707 load\_C = 2.293635 Value = -254.4025   
elapsed = 19.72 Round = 8 load\_A = 1.441509 load\_B = 2.000826 load\_C = 2.848157 Value = -0.3025   
elapsed = 19.82 Round = 9 load\_A = 1.265308 load\_B = 2.005492 load\_C = 2.833027 Value = -0.7225   
elapsed = 19.84 Round = 10 load\_A = 1.491368 load\_B = 2.231192 load\_C = 2.548575 Value = 0.0000   
elapsed = 19.69 Round = 11 load\_A = 1.059263 load\_B = 2.27733 load\_C = 2.92476 Value = 0.0000   
elapsed = 21.08 Round = 12 load\_A = 1.0000 load\_B = 2.5000 load\_C = 2.0000 Value = -1.4400   
elapsed = 19.80 Round = 13 load\_A = 1.5000 load\_B = 2.5000 load\_C = 2.0000 Value = 0.0000   
elapsed = 19.88 Round = 14 load\_A = 1.483996 load\_B = 1.5000 load\_C = 2.09888 Value = -4893.0025   
elapsed = 19.58 Round = 15 load\_A = 1.476378 load\_B = 2.172586 load\_C = 2.383395 Value = -0.1225   
  
 Best Parameters Found:   
Round = 5 load\_A = 1.354807 load\_B = 2.36714 load\_C = 2.886838 Value = 0.0000

#### Apply the loadings

library(dplyr)  
  
best\_loads\_be <- opt\_res\_be$Best\_Par  
  
tier\_map <- c(  
 A = best\_loads\_be[["load\_A"]],  
 B = best\_loads\_be[["load\_B"]],  
 C = best\_loads\_be[["load\_C"]]  
)  
  
port\_final\_be <- port %>%  
 mutate(  
 OptimalLoading = tier\_map[as.character(Tier)],  
 FinalPremium = predicted\_pure\_premium \* OptimalLoading  
 )  
  
print(  
 port\_final\_be %>%  
 group\_by(Tier) %>%  
 summarize(  
 n\_policies = n(),  
 avg\_loading = round(mean(OptimalLoading), 4),  
 avg\_final\_prem = round(mean(FinalPremium), 2),  
 total\_final\_prem = round(sum(FinalPremium), 0),  
 .groups = "drop"  
 )  
)

# A tibble: 3 × 5  
 Tier n\_policies avg\_loading avg\_final\_prem total\_final\_prem  
 <chr> <int> <dbl> <dbl> <dbl>  
1 A 3119 1.35 42.6 132801  
2 B 6544 2.37 80.1 524121  
3 C 337 2.89 947. 318979

library(writexl)  
  
write\_xlsx(port\_final\_be, path = "premiums\_loaded.xlsx"  
)

#### Testing

# Extract the values with the corresponding name  
loads <- c(A = best\_loads\_be[["load\_A"]], B = best\_loads\_be[["load\_B"]], C = best\_loads\_be[["load\_C"]])  
  
run\_simulation(port,loads,5000)

$ruin\_prob  
[1] 4e-04  
  
$expected\_profit  
[1] 234267.7