# Library Used.

library(readr)

library(ggplot2)

library(dplyr)

library(glmnet)

library(Matrix)

library(interactions)

library(doParallel)

library(broom)

library(pscl)

### Importing Dataset

data\_CS <- read.csv("BodySystemTrainTable.csv")

# Calculate 10% sample size

sample\_size <- round(0.10 \* nrow(data\_CS))

# Assuming your dataset is named "my\_data"

data\_size <- dim(sample\_size)

# Printing the number of rows and columns

cat("Number of rows:", data\_size[1], "\n")

cat("Number of columns:", data\_size[2], "\n")

# Create a 10% random sample of your data

set.seed(123)

sampled\_indices <- sample(seq\_len(nrow(data\_CS)), size = sample\_size)

sampled\_datasets <- data\_CS[sampled\_indices, ]%>% filter(!is.na(bs7lr))

# Assuming your dataset is named "my\_data"

data\_size <- dim(sampled\_datasets)

# Printing the number of rows and columns

cat("Number of rows:", data\_size[1], "\n", "Number of columns:", data\_size[2], "\n")

# Create the binary variable for the circulatory system

sampled\_datasets <- sampled\_datasets %>%

 mutate(Circulatory\_Body\_System = as.factor(ifelse(bs7lr < 1, 0, 1))) %>%

 select(-bs7lr)

# Define a function for mean imputation of multiple columns

mean\_impute <- function(data, columns\_to\_impute) {

 for (col in columns\_to\_impute) {

 data[is.na(data[, col]), col] <- mean(data[, col], na.rm = TRUE)

 }

 return(data)

}

# Columns to impute missing values

columns\_to\_impute <- c("dm", "bs1lr", "bs2lr", "bs3lr", "bs4lr", "bs5lr", "bs6lr",

 "bs8lr", "bs9lr", "bs10lr", "bs11lr", "bs12lr", "bs13lr", "bs14lr",

 "bs16lr", "bs17lr", "bs18lr", "bs19lr")

sampled\_datasets <- mean\_impute(sampled\_datasets, columns\_to\_impute)

# Removing unnessasary variables

sampled\_datasets <- sampled\_datasets %>%

 select(-bs15lr, -Vlr, -TestTrain, -id)

#C, Include pairwis and triplecombination of independent variables

#in your analysis. Print out the values for the first 4

#rows of the dependent and independent variables

################## function to create the interactions ########################

create\_interactions <- function(data, independent\_vars, order = c(2)) {

 interaction\_data <- data

 # Check if the order is valid

 if (max(order) > length(independent\_vars)) {

 stop("Invalid order. The order cannot be greater than the number of independent variables.")

 }

 for (o in order) {

 # Creates the combinations of independent variables stated by o,

 # Then assigen the combination to a variable "interactions"

 interactions <- combn(independent\_vars, o, simplify = FALSE)

 #1. take each one from the interaction

 for (comb in interactions) {

 #2. Create interaction name

 interaction\_name <- paste(comb, collapse = "\_")

 #3. the product of values across the columns specified by the comb vector.

 interaction\_data <- cbind(

 interaction\_data,

 Reduce(`\*`, lapply(comb, function(var\_name) interaction\_data[, var\_name]))

 )

 colnames(interaction\_data)[ncol(interaction\_data)] <- interaction\_name

 }

 }

 return(interaction\_data)

}

independent\_vars <- c("dm","bs1lr","bs2lr","bs3lr", "bs4lr","bs5lr","bs6lr",

 "bs8lr","bs9lr","bs10lr","bs11lr","bs12lr","bs13lr","bs14lr",

 "bs16lr","bs17lr","bs18lr", "bs19lr")

sample\_data\_with\_interactions <- create\_interactions(sampled\_datasets, independent\_vars, order = c(2,3))

# Total column count after interaction

col\_count <- ncol(sample\_data\_with\_interactions)

cat("Total column count after interaction:", col\_count, "\n")

# Print the first 4 rows

View(head(sample\_data\_with\_interactions, n = 4))

# Split dataset into training and test sets

set.seed(123)

train\_percentage <- 0.7

num\_train\_samples <- round(train\_percentage \* nrow(sample\_data\_with\_interactions))

# Generate random training indices

# Step 1: Generate a sequence of integers representing row indices

all\_row\_indices <- seq\_len(nrow(sample\_data\_with\_interactions))

# Randomly sample row indices for the training dataset

train\_indices <- sample(all\_row\_indices, size = num\_train\_samples)

# Create the training and test datasets

train\_data <- sample\_data\_with\_interactions[train\_indices, ]

test\_data <- sampled\_datasets[-train\_indices, ]

# Set the number of CPU cores to use in parallel

num\_cores <- 8 # Adjust this number based on your available CPU cores

# Register the parallel backend

registerDoParallel(cores = num\_cores)

x\_train <- as.matrix(train\_data[, -c(19)]) # Features in the training data

y\_train <- train\_data[, c(19)] # Target variable in the training data

# Define class weights

imbalance\_ratio <- sum(y\_train == 1) / sum(y\_train == 0)

weight\_minority <- imbalance\_ratio

weight\_majority <- 1

class\_weights <- ifelse(y\_train == 0, weight\_minority, weight\_majority)

# Standardize the features

train\_x\_scaled <- scale(x\_train)

min\_desired\_non\_zero\_count <- 10

max\_desired\_non\_zero\_count <- 15

current\_non\_zero\_count <- Inf

lambda\_sequence <- NULL

lasso\_model <- NULL

# Iterate through different lambda values until the desired non-zero count is within the range

while (current\_non\_zero\_count < min\_desired\_non\_zero\_count || current\_non\_zero\_count > max\_desired\_non\_zero\_count) {

 # Fit the model with a range of lambda values

 lasso\_model <- glmnet(train\_x\_scaled, y\_train, family = "binomial", alpha = 1, , lambda = lambda\_sequence,weights = class\_weights)

 # Extract coefficients for the current lambda

 coefficients <- coef(lasso\_model, s = lambda\_sequence)

 # Count non-zero coefficients

 current\_non\_zero\_count <- sum(coefficients != 0)

 # Increase the lambda sequence to continue the search for sparsity

 lambda\_sequence <- c(lambda\_sequence, max(lasso\_model$lambda) \* 0.2) # Adjust the multiplier as needed

}

# Deregister the parallel backend when done

stopImplicitCluster()

# Extract coefficients for the best lambda (automatically selects non-zero coefficients)

# Extract the coefficients from the model

lasso\_model

optimal\_lambda <- lasso\_model$lambda

cat("Optimal Lambda:", optimal\_lambda, "\n")

# Get the coffcients of our model

coefficients <- coef(lasso\_model)

# printing only the non zero coefficients

non\_zero\_coeffs <- coefficients[coefficients != 0]

# Get the coefficients that are non-zero

non\_zero\_indices <- which(coefficients != 0)

# Assuming X is your original dataset

non\_zero\_features=colnames(train\_data)[non\_zero\_indices]

# Table of variables

# Create a data frame from the two vectors

variable\_table <- data.frame(Features= non\_zero\_features, Cofficients = non\_zero\_coeffs)

View(variable\_table)

############Calculating McFadden's R-squared################################

# Obtain the lambda value that minimizes deviance (or another suitable metric)

optimal\_lambda <- lasso\_model$lambda.min

# Fit the final Lasso logistic regression model with the optimal lambda

final\_model <- glmnet(train\_x\_scaled, y\_train, alpha = 1, family = "binomial", lambda = optimal\_lambda)

deviance\_full <- deviance(lasso\_model)

deviance\_lasso <- deviance(final\_model) # Assuming you already have the final Lasso model

# Calculate the proportion of deviance explained (analogous to R²)

mcfadden\_r2\_approx <- 1 - (deviance\_lasso / deviance\_full)

mcfadden\_r2\_approx