```r
#------------------------------Initial Setup----------------------------

# Set random number seed for reproducibility
set.seed(7406)

# Set the working directory
setwd("C:/Users/ns14555/Desktop/Projects/001 Project 1 (HW2)/003 Project
work")

# Check the current working directory to confirm it was set correctly
getwd()

# Load required libraries
library(leaps)
library(MASS)
library(glmnet)
library(lars)
library(pls)

#-------------------------Data Loading and Preparation--------------------

# Load Data
fat <- read.csv("C:/Users/ns14555/Desktop/Projects/001 Project 1 (HW2)/003
Project work/fat.csv")
n = dim(fat)[1] # total number of observations
n1 = round(n/10) # number of observations randomly selected for testing
data

# Split the data into training and test set
train <- sample(c(TRUE, FALSE), nrow(fat),replace=TRUE, prob=c(0.7, 0.3))
fat_train <- fat[train, ]
fat_test <- fat[!train,]

# Exploratory Data Analysis
summary(fat)
#head(fat)

# Initial Linear Regression
model_1 <-lm(brozek ~., data=fat_train)
#summary(model_1)

# Plot relationship between 'siri' and 'brozek'
```

```r
plot(fat_train$siri, fat_train$brozek, main = "Scatter Plot", xlab =
'siri', ylab = 'brozek')

# Remove correlated predictors and re-model
fat_train <- fat_train[,!colnames(fat_train) %in% c('siri','free')]
fat_test <- fat_test[,!colnames(fat_test) %in% c('siri','free')]
model_2 <-lm(fat_train$brozek ~., data=fat_train)
#summary(model_2)

# Regression focused on 'density' predictor
model_density <- lm(brozek ~ density, data = fat_train)
#summary(model_density)
#Scatterplot between these two
#plot(fat_train$density, fat_train$brozek, main = "Scatter Plot",
#     xlab = 'density', ylab = 'brozek')

dim(fat_train) ## Distribution of the data labels in the training data
# Boxplots of train and test dataset
boxplot(fat_train)
boxplot(fat_test)

# Response variable summary
summary(fat_train$brozek)
summary(fat_test$brozek)

# Details about train dataset
head(fat_train)
dim(fat_train)

#--------------------------------Model Building--------------------------

# Initialize testing error collection
test_MSE <- matrix(nrow = 1, ncol =7)
y_true <- fat_test$brozek

# Model 1:Linear regression with all predictors
model_lm_full <- lm(brozek~., data = fat_train)
#summary(model_lm_full)
pred_lm_full <- predict(model_lm_full, fat_test[,2:16])
test_mse_lm_full <- mean((pred_lm_full-y_true)^2)

# Model 2:Linear regression with best subset
# Fit the best subset selection model to identify the best combination
```

```
# of predictors
subset_model <-regsubsets(brozek ~., data=fat_train, nvmax=5, nbest=1)
subset_summary <- summary(subset_model)
# Determine the best model based on adjusted R-squared values
best_subset_index <- which.max(subset_summary$adjr2)
# Get the variable names from the best subset
best_subset_vars <- names(which(coef(subset_model,
id=best_subset_index)!=0))
# Remove the intercept
best_subset_vars <- best_subset_vars[best_subset_vars!="(Intercept)"]
# Construct the formula
best_formula <-reformulate(termlabels = best_subset_vars,
response='brozek')
# Fit the model
model_lm_subset <- lm(formula = best_formula, data = fat_train)
pred_lm_subset <- predict(model_lm_subset, fat_test[,2:16])
test_mse_lm_subset <- mean((pred_lm_subset-y_true)^2)

# Model 3: Linear regression with the stepwise variable selection that
minimizes
# the AIC criterion
model_step_aic <- step(model_lm_full, direction = 'both', trace = 0)
pred_step_aic <- predict(model_step_aic, fat_test[,2:16])
test_mse_step_aic <- mean((pred_step_aic-y_true)^2)

# Model 4: Ridge regression
model_ridge <- lm.ridge( brozek ~ ., data = fat_train, lambda=
seq(0,100,0.001))
# Prepare the data
x_matrix <- model.matrix(brozek~., data = fat_train)[,-1] #excluding
intercept
y_vector <- fat_train$brozek
# Prepare the predictor matrix for the test data
x_test_matrix <- model.matrix(brozek~., data = fat_test) #excluding
intercept
# Fit the ridge regression model with cross validation
model_ridge <- cv.glmnet(x_matrix, y_vector, alpha=0, type.measure='mse')
# Extract the optimum lambda value
lambda_optimal_ridge <-model_ridge$lambda.min
pred_test_ridge <- predict(model_ridge, s=lambda_optimal_ridge,
                           newx=x_test_matrix[,2:16])
test_mse_ridge <- mean((pred_test_ridge-y_true)^2)
```

```r
# Model 5: Lasso regression
# model_lasso <- lars(as.matrix(fat_train[2,16],fat_train$brozek,
type=TRUE))
# Fit the lasso regression model with cross validation
model_lasso <- cv.glmnet(x_matrix, y_vector, alpha=1, type.measure='mse')
# Extract the optimum lambda value
lambda_optimal_lasso <-model_lasso$lambda.min
pred_test_lasso <- predict(model_lasso, s=lambda_optimal_lasso,
                           newx=x_test_matrix[,2:16])
test_mse_lasso <- mean((pred_test_lasso-y_true)^2)
# Extract coefficients at the optimal lambda
lasso_coefs <- coef(model_lasso, s = lambda_optimal_lasso, exact = TRUE)

# Convert to a more readable format, if not already clear
lasso_coefs_df <- as.data.frame(lasso_coefs)
# Convert the sparse matrix to a regular matrix
lasso_coefs_matrix <- as.matrix(lasso_coefs)

# Convert the matrix to a data frame
lasso_coefs_df <- as.data.frame(lasso_coefs_matrix)

# Add names to the data frame for clarity
rownames(lasso_coefs_df) <- rownames(lasso_coefs_matrix)
names(lasso_coefs_df)[1] <- "Coefficients"

# Model 6: Principal Component Regression (PCR)
model_pcr <- pcr(brozek ~., data=fat_train, scale = TRUE, validation='CV')
# Determine the optimal number of components with minimum cross validation
MSEP
ncomp_optimal_pcr <- which.min(MSEP(model_pcr)$val[,,1])
pred_test_pcr <- predict(model_pcr,newdata = fat_test[,2:16],
                         ncomp=ncomp_optimal_pcr)
# Convert PCR model predictions to a vector format for consistant
calculation
pred_test_pcr_vec <- as.vector(pred_test_pcr)
test_mse_pcr<- mean((pred_test_pcr_vec-y_true)^2)

# Model 7. Partial Least Squares (PLS) Regression
model_pls <- plsr(brozek ~., data=fat_train, scale = TRUE, validation='CV')
# test error
pred_test_pls <- predict(model_pls, fat_test[,2:16])
pred_test_pls <- as.vector(pred_test_pls)
test_mse_pls <- mean((pred_test_pls-y_true)^2)
```

```r
# Collect the testing errors
test_MSE <- c(test_mse_lm_full,test_mse_lm_subset,test_mse_step_aic,
              test_mse_ridge,test_mse_lasso, test_mse_pcr,test_mse_pls)

#------------------------Monte Carlo Cross-Validation--------------------

set.seed(7407) # Reset seed fr reproducibility
B <- 100
TEALL <- matrix(nrow = B, ncol =7) # Preallocate matrix for efficiency

for (b in 1:B){
  #-----------------------Initial Preparation--------------------------
  indices <- sample(1:nrow(fat), size = round(0.3*nrow(fat)))
  fat_train <- fat[-indices,]
  fat_test <- fat[indices,]
  # Extract the response variable for the test set
  y_true <- fat_test$brozek
  # Lets remove some columns from this dataset & try the regression  again
  fat_train <- fat_train[,!colnames(fat_train) %in% c('siri','free')]
  fat_test <- fat_test[,!colnames(fat_test) %in% c('siri','free')]

  #------------------------ Model Building------------------------------
  # Model 1:Linear regression with all predictors
  model_lm_full <- lm(brozek~., data = fat_train)
  pred_lm_full <- predict(model_lm_full, fat_test[,2:16])
  te1 <- mean((pred_lm_full-y_true)^2)

  # Model 2:Linear regression with best subset
  # Fit the best subset selection model to identify the best combination
  # of predictors
  subset_model <-regsubsets(brozek ~., data=fat_train, nvmax=5, nbest=1)
  sv_subset_summary <- summary(subset_model)
  # Determine the best model based on adjusted R-squared values
  best_subset_index <- which.max(sv_subset_summary$adjr2)
  # Get the variable names from the best subset
  best_subset_vars <- names(which(coef(subset_model,
id=best_subset_index)!=0))
  # Remove the intercept
  best_subset_vars <- best_subset_vars[best_subset_vars!="(Intercept)"]
  # Construct the formula
  best_formula <-reformulate(termlabels = best_subset_vars,
response='brozek')
```

```
  # Fit the model
  model_lm_subset <- lm(formula = best_formula, data = fat_train)
  pred_lm_subset <- predict(model_lm_subset, fat_test[,2:16])
  te2 <- mean((pred_lm_subset-y_true)^2)

  # Model 3: Linear regression with the stepwise variable selection that
  # minimizes the AIC criterion
  model_step_aic <- step(model_lm_full, direction = 'both', trace = 0)
  pred_step_aic <- predict(model_step_aic, fat_test[,2:16])
  te3 <- mean((pred_step_aic-y_true)^2)

  # Model 4: Ridge regression
  model_ridge <- lm.ridge( brozek ~ ., data = fat_train,
                           lambda= seq(0,100,0.001))
  # Prepare the data
  x_matrix <- model.matrix(brozek~., data = fat_train)[,-1] #excluding
intercept
  y_vector <- fat_train$brozek
  # Prepare the predictor matrix for the test data
  x_test_matrix <- model.matrix(brozek~., data = fat_test) #excluding
intercept
  # Fit the ridge regression model with cross validation
  model_ridge <- cv.glmnet(x_matrix, y_vector, alpha=0, type.measure='mse')
  # Extract the optimum lambda value
  lambda_optimal_ridge <-model_ridge$lambda.min
  pred_test_ridge <- predict(model_ridge, s=lambda_optimal_ridge,
                             newx=x_test_matrix[,2:16])
  te4 <- mean((pred_test_ridge-y_true)^2)

  # Model 5: Lasso regression
  # model_lasso <- lars(as.matrix(fat_train[2,16],fat_train[,1],
type=TRUE))
  # Fit the lasso regression model with cross validation
  model_lasso <- cv.glmnet(x_matrix, y_vector, alpha=1, type.measure='mse')
  # Extract the optimum lambda value
  lambda_optimal_lasso <-model_lasso$lambda.min
  pred_test_lasso <- predict(model_lasso, s=lambda_optimal_lasso,
                             newx=x_test_matrix[,2:16])
  te5 <- mean((pred_test_lasso-y_true)^2)

  # Model 6: Principal Component Regression (PCR)
  model_pcr <- pcr(brozek ~., data=fat_train, scale = TRUE,
validation='CV')
```

```
  # Determine the optimal number of components with minimum cross
validation MSEP
  ncomp_optimal_pcr <- which.min(MSEP(model_pcr)$val[,,1])
  pred_test_pcr <- predict(model_pcr,newdata = fat_test[,2:16],
                           ncomp=ncomp_optimal_pcr)
  # Convert PCR model predictions to a vector format for consistant
calculation
  pred_test_pcr_vec <- as.vector(pred_test_pcr)
  te6<- mean((pred_test_pcr_vec-y_true)^2)

  # Model 7. Partial Least Squares (PLS) Regression
  model_pls <- plsr(brozek ~., data=fat_train, scale = TRUE,
validation='CV')
  pred_test_pls <- predict(model_pls, fat_test[,2:16])
  pred_test_pls <- as.vector(pred_test_pls)
  te7 <- mean((pred_test_pls-y_true)^2)

  # Collect the testing errors
  TEALL = cbind(te1, te2, te3, te4, te5, te6, te7)
}
TEALL

# if you want, you can change the column name
colnames(TEALL) <- c("lm_full", "lm_subset","lm_stepwise", "ridge","lasso",
"PCR","PLS")
names(test_MSE) <- c("lm_full", "lm_subset","lm_stepwise", "ridge","lasso",
"PCR","PLS")

#Before Cross Validation
test_MSE
#After Cross Validation
TEALL

# From the mean standard error, we found the LASSO Regression to be the
best
# performing model. The most important predictors are:
lasso_coefs_df[lasso_coefs_df$Coefficients != 0, , drop = FALSE]
#----------------------------- The End -------------------------------
```