Exercise 7 - Comparing penalized regression estimators

Stefan Merdian

2024-12-13

Task 1

1)

The LASSO Shooting Algorithm is an iterative method to solve the LASSO regression problem, which minimizes the objective function:

- Minimize the prediction error
- Apply L1 regularization to shrink some coefficients j to exactly 0, which results in feature selection

The intercept must not be regularized because it represents the mean of the target variable y. So we center the data to focus the regularization only on the coefficients.

So our approach:

1. Initialization: Start with all coefficients j=0.
2. For each coefficient j, we calculate the residual rj
3. Compute zj, the contribution of xj to reduce the error.
4. Apply Soft Thresholding to update j.
5. Check Convergence. If the coefficients change less than a given tolerance, stop the iterations.

```
soft_threshold <- function(z, lambda) {
  if (is.na(z)) return(0)
  if (z > lambda) {
    return(z - lambda)
  } else if (z < -lambda) {
    return(z + lambda)
  } else {
    return(0)
  }
}


lasso_shooting <- function(X, y, lambda, tol = 1e-6, max_iter = 1000) {
  n <- nrow(X)
  p <- ncol(X)

  intercept <- mean(y)

  y_centered <- y - intercept
```

```
  X_centered <- scale(X, center = TRUE, scale = FALSE)

  # coef init
  beta <- rep(0, p)

  for (iter in 1:max_iter) {
    beta_old <- beta

    for (j in 1:p) {
      # residuals error
      r_j <- y_centered - X_centered %*% beta + X_centered[, j] * beta[j]
      z_j <- sum(X_centered[, j] * r_j) / n
      beta[j] <- soft_threshold(z_j, lambda)
    }
    if (max(abs(beta - beta_old)) < tol) {
      break
    }
  }
  return(list(intercept = intercept, coefficients = beta))
}
```

2)

We now modify the function we implemented to allow for the input of multiple lambda values.

```
lasso_shooting_for_lambdas <- function(X, y, lambdas) {
  p <- ncol(X)
  n_lambdas <- length(lambdas)

  coef_matrix <- matrix(0, nrow = n_lambdas, ncol = p + 1)
  colnames(coef_matrix) <- c("Intercept", paste0("Beta_", 1:p))
  rownames(coef_matrix) <- paste0("Lambda_", lambdas)

  for (i in seq_along(lambdas)) {
    lambda <- lambdas[i]
    #lambda value for lasso_shooting
    result <- lasso_shooting(X, y, lambda = lambda)
    coef_matrix[i, 1] <- result$intercept
    coef_matrix[i, 2:(p + 1)] <- result$coefficients
  }

  return(coef_matrix)
}
```

3)

Simulating Data

```
library(glmnet)
```

```
## Warning: Paket 'glmnet' wurde unter R Version 4.4.2 erstellt

## Lade nötiges Paket: Matrix

## Loaded glmnet 4.1-8
```

```
set.seed(123)
n_samples <- 100
n_features <- 10

X <- matrix(rnorm(n_samples * n_features), nrow = n_samples, ncol = n_features)
true_coefs <- c(1.5, -2, 0, 0, 0, 3, 0, 0, -1.2, 0)
y <- X %*% true_coefs + rnorm(n_samples, sd = 0.5)
```

Splitting into train and test data

```
train_indices <- sample(1:n_samples, size = 0.7 * n_samples)
X_train <- X[train_indices, ]
y_train <- y[train_indices]
X_test <- X[-train_indices, ]
y_test <- y[-train_indices]
```

We create a sequence of lambda values and iterate through each lambda using our simulated data. For each lambda, we apply the LASSO regression using the glmnet function, calculate the Mean Squared Error (MSE) on the test set, and extract the resulting coefficients

```
lambdas <- c(0.01, 0.1, 0.5, 1, 2)

results <- list()

for (lambda in lambdas) {
  lasso_model <- glmnet(
    x = X_train,
    y = y_train,
    alpha = 1,
    lambda = lambda
  )

  y_pred <- predict(lasso_model, s = lambda, newx = X_test)
  mse <- mean((y_test - y_pred)^2)

  results[[as.character(lambda)]] <- list(
    lambda = lambda,
    mse = mse,
    coefficients = coef(lasso_model, s = lambda)
  )
}

for (res in results) {
  cat("\nLambda:", res$lambda, "\n")
  cat("Mean Squared Error (MSE):", res$mse, "\n")
  cat("Coefficients:\n")
  print(res$coefficients)
}
```

```
##
## Lambda: 0.01
## Mean Squared Error (MSE): 0.4172609
## Coefficients:
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.09981100
## V1           1.52379782
## V2          -1.87929521
## V3          -0.03593411
## V4           .
## V5           0.01199084
## V6           2.97885133
## V7           0.04106389
## V8           0.07654415
## V9          -1.29849123
## V10          0.11644677
##
## Lambda: 0.1
## Mean Squared Error (MSE): 0.4549419
## Coefficients:
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.09993432
## V1           1.44295090
## V2          -1.78453785
## V3           .
## V4           .
## V5           .
## V6           2.84106857
## V7           .
## V8           0.01984190
## V9          -1.19901196
## V10          0.05189061
##
## Lambda: 0.5
## Mean Squared Error (MSE): 1.780805
## Coefficients:
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  0.1458923
## V1           0.9935561
## V2          -1.3078407
## V3           .
## V4           .
## V5           .
## V6           2.2165625
## V7           .
## V8           .
## V9          -0.7727823
## V10          .
##
## Lambda: 1
## Mean Squared Error (MSE): 6.046971
```

```
## Coefficients:
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  0.2005080
## V1           0.4210591
## V2          -0.6968723
## V3           .
## V4           .
## V5           .
## V6           1.4333617
## V7           .
## V8           .
## V9          -0.2394959
## V10          .
##
## Lambda: 2
## Mean Squared Error (MSE): 15.15292
## Coefficients:
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) 0.2490599
## V1          .
## V2          .
## V3          .
## V4          .
## V5          .
## V6          0.1407424
## V7          .
## V8          .
## V9          .
## V10         .
```

Now to compare the results we do the same for our functions.

```r
result_custom <- list()

for (lambda in lambdas) {
  result <- lasso_shooting(X_train, y_train, lambda)
  intercept <- result$intercept
  coefficients <- result$coefficients
  y_pred <- X_test %*% coefficients + intercept
  mse <- mean((y_test - y_pred)^2)
  result_custom[[as.character(lambda)]] <- list(
    lambda = lambda,
    mse = mse,
    coefficients = c(intercept, coefficients)
  )
}

for (res in result_custom) {
  cat("\nLambda:", res$lambda, "\n")
  cat("Mean Squared Error (MSE):", res$mse, "\n")
  cat("Coefficients:\n")
  print(res$coefficients)
```

```
}
```

```
##
## Lambda: 0.01
## Mean Squared Error (MSE): 1.288887
## Coefficients:
##  [1]  0.24555177  1.29143938 -1.56484484 -0.09316218 -0.14763554  0.10639939
##  [7]  2.34427312 -0.06135296  0.27835449 -1.46027970  0.19016185
##
## Lambda: 0.1
## Mean Squared Error (MSE): 1.350919
## Coefficients:
##  [1]  0.245551768  1.212132561 -1.532387038  0.000000000  0.000000000
##  [6]  0.007159284  2.268535885  0.000000000  0.180605385 -1.336256116
## [11]  0.069997973
##
## Lambda: 0.5
## Mean Squared Error (MSE): 3.210964
## Coefficients:
##  [1]  0.2455518  0.8424489 -1.1066540  0.0000000  0.0000000  0.0000000
##  [7]  1.7393313  0.0000000  0.0000000 -0.8649579  0.0000000
##
## Lambda: 1
## Mean Squared Error (MSE): 7.91757
## Coefficients:
##  [1]  0.2455518  0.3065487 -0.5401645  0.0000000  0.0000000  0.0000000
##  [7]  1.0559939  0.0000000  0.0000000 -0.3013822  0.0000000
##
## Lambda: 2
## Mean Squared Error (MSE): 15.96382
## Coefficients:
##  [1] 0.2455518 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000 0.0000000
```

Our custom LASSO implementation and cv.glmnet show similar overall trends, with increasing sparsity and MSE as lambda grows. However, for smaller lambda values (e.g., 0.01), cv.glmnet achieves slightly lower MSE and retains more non-zero coefficients, likely due to its more precise optimization. For larger lambda values (1 or 2), both approaches produce sparse models with very similar MSEs, as most coefficients shrink to zero. The main difference lies in how aggressively your custom implementation regularizes at smaller lambdas, leading to slightly higher MSE compared to cv.glmnet.

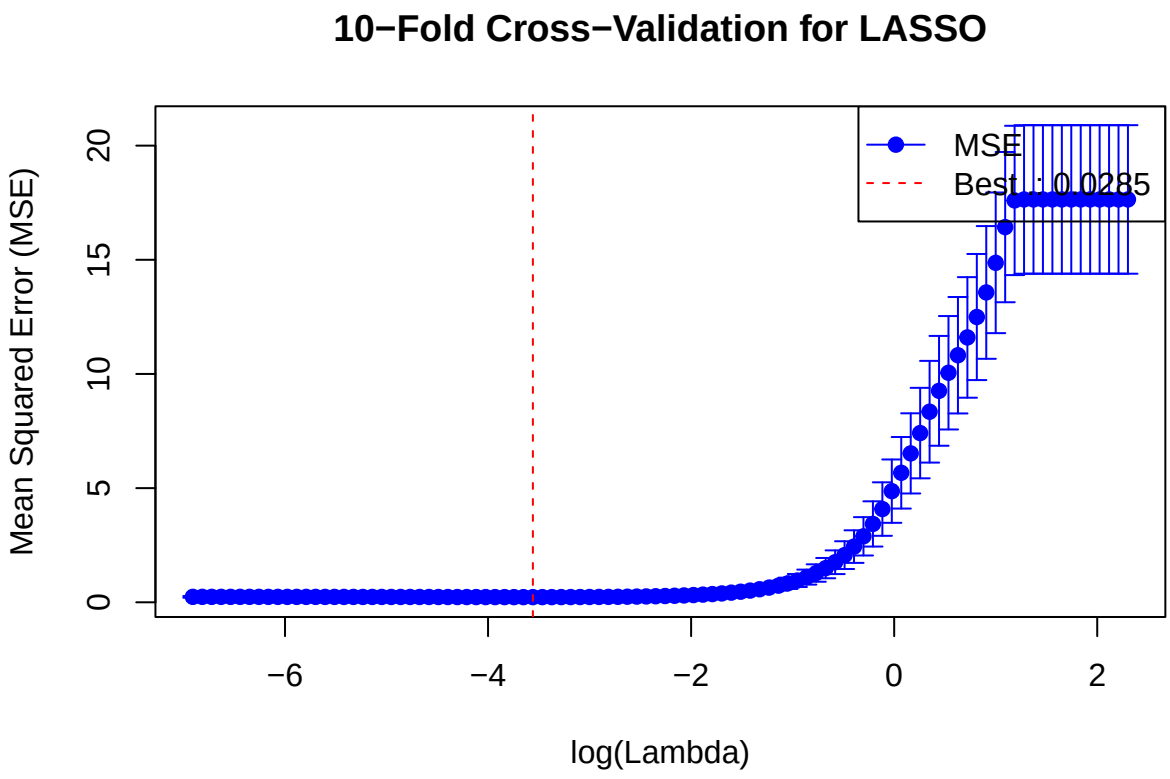
4)

```
library(glmnet)

lasso_cv <- function(X, y, lambdas, n_folds = 10) {
  set.seed(123)
  n <- nrow(X)

  folds <- sample(rep(1:n_folds, length.out = n))
  mse_matrix <- matrix(NA, nrow = n_folds, ncol = length(lambdas))
```

```r
colnames(mse_matrix) <- lambdas

# Perform cross-validation
for (fold in 1:n_folds) {
  train_idx <- which(folds != fold)
  val_idx <- which(folds == fold)

  X_train <- X[train_idx, ]
  y_train <- y[train_idx]
  X_val <- X[val_idx, ]
  y_val <- y[val_idx]
  for (lambda in lambdas) {
    lasso_model <- glmnet(X_train, y_train, alpha = 1, lambda = lambda)
    y_pred <- predict(lasso_model, newx = X_val)
    mse_matrix[fold, as.character(lambda)] <- mean((y_val - y_pred)^2)
  }
}

mean_mse <- colMeans(mse_matrix, na.rm = TRUE)
se_mse <- apply(mse_matrix, 2, function(x) sd(x, na.rm = TRUE) / sqrt(n_folds))

#lambda minimum MSE
best_lambda_mse <- as.numeric(names(which.min(mean_mse)))

# lambda minimum RMSE
best_lambda_rmse <- as.numeric(names(which.min(sqrt(mean_mse))))

plot(
  x = log(lambdas),
  y = mean_mse,
  type = "b",
  ylim = c(min(mean_mse - se_mse), max(mean_mse + se_mse)),
  xlab = "log(Lambda)",
  ylab = "Mean Squared Error (MSE)",
  main = "10-Fold Cross-Validation for LASSO",
  pch = 19,
  col = "blue"
)
arrows(
  x0 = log(lambdas), y0 = mean_mse - se_mse,
  x1 = log(lambdas), y1 = mean_mse + se_mse,
  angle = 90, code = 3, length = 0.05, col = "blue"
)
abline(v = log(best_lambda_mse), col = "red", lty = 2)
legend(
  "topright", legend = c("MSE", paste("Best  :", round(best_lambda_mse, 4))),
  col = c("blue", "red"), lty = c(1, 2), pch = c(19, NA)
)

return(list(
  best_lambda_mse = best_lambda_mse,
  best_lambda_rmse = best_lambda_rmse,
  mean_mse = mean_mse,
```

```
    se_mse = se_mse,
    mse_matrix = mse_matrix
  ))
}
```

```
n_samples <- 100
n_features <- 10

X <- matrix(rnorm(n_samples * n_features), nrow = n_samples, ncol = n_features)
true_coefs <- c(1.5, -2, 0, 0, 0, 3, 0, 0, -1.2, 0)
y <- X %*% true_coefs + rnorm(n_samples, sd = 0.5)
lambdas <- exp(seq(log(0.001), log(10), length.out = 100))
cv_results <- lasso_cv(X, y, lambdas)
```




```
cat("Lambda minimizing MSE:", cv_results$best_lambda_mse, "\n")
```

```
## Lambda minimizing MSE: 0.02848036
```

```
cat("Lambda minimizing RMSE:", cv_results$best_lambda_rmse, "\n")
```

```
## Lambda minimizing RMSE: 0.02848036
```

Task 2

1)

First we load the data and split into train and test set.

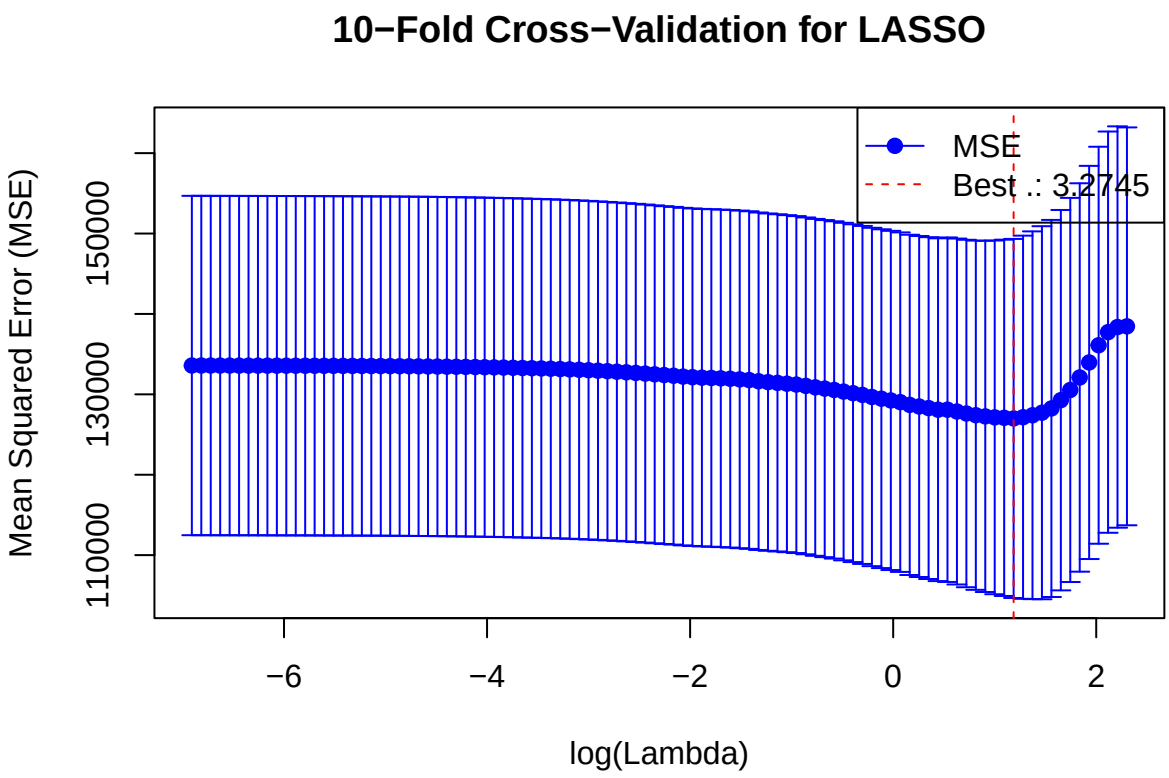
```
library(ISLR)
```

```
## Warning: Paket 'ISLR' wurde unter R Version 4.4.2 erstellt
```

```
data("Hitters")

Hitters <- na.omit(Hitters)

numeric_vars <- sapply(Hitters, is.numeric)
Hitters_numeric <- Hitters[, numeric_vars]
y <- Hitters_numeric$Salary
X <- as.matrix(Hitters_numeric[, colnames(Hitters_numeric) != "Salary"])

n <- nrow(X)

train_indices <- sample(1:n, size = 0.7 * n)


X_train <- X[train_indices, ]
y_train <- y[train_indices]

X_test <- X[-train_indices, ]
y_test <- y[-train_indices]
```

We generate a sequence of lambdas and use our custom function to determine the minimum lambda.

```
lambdas <- exp(seq(log(0.001), log(10), length.out = 100))

lasso_custom <- lasso_cv(X_train, y_train, lambdas)
```




```
cat("Lambda minimizing MSE:", lasso_custom$best_lambda_mse, "\n")
```

```
## Lambda minimizing MSE: 3.274549
```

```
cat("Lambda minimizing RMSE:", lasso_custom$best_lambda_rmse, "\n")
```

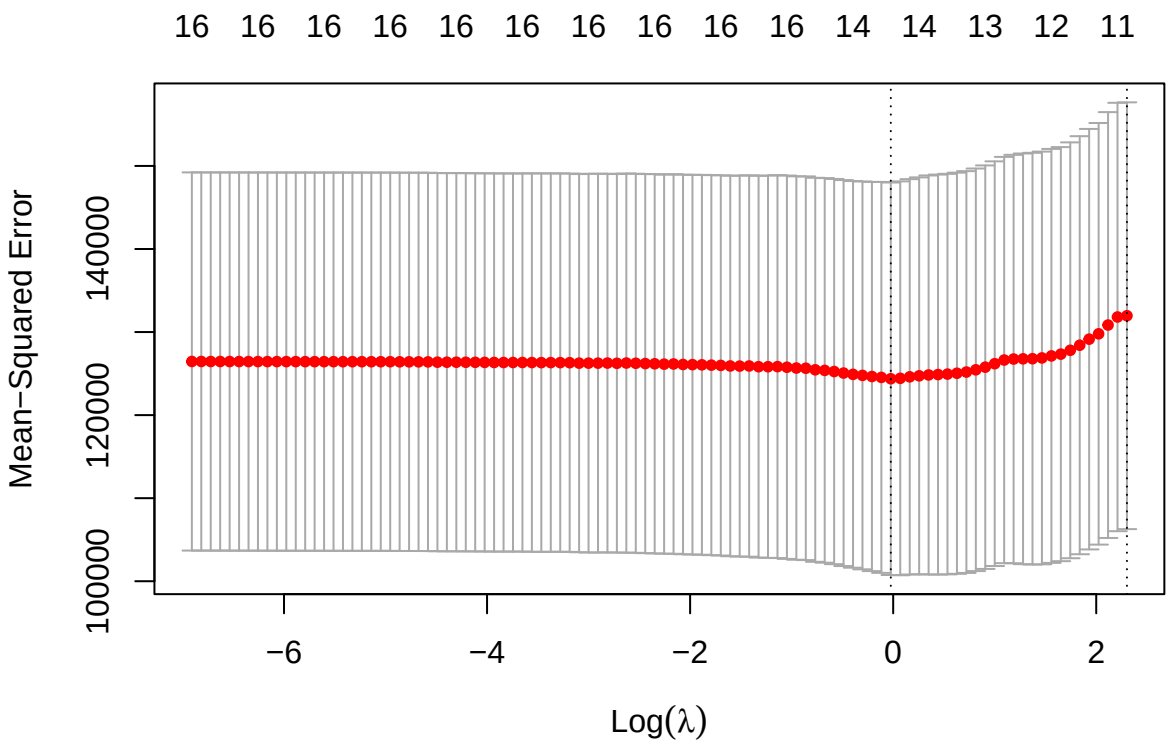
```
## Lambda minimizing RMSE: 3.274549
```

Our best lambda is: 3.2745

2)

Now we do the same with cv.glmnet function.

```
lasso_model <- cv.glmnet(X_train, y_train, lambda = lambdas,alpha=1)
plot(lasso_model)
```




```
cat("Lambda minimizing MSE:", lasso_model$lambda.min, "\n")
```

```
## Lambda minimizing MSE: 0.97701
```

```
cat("Lambda within 1 SE of MSE:", lasso_model$lambda.1se, "\n")
```

```
## Lambda within 1 SE of MSE: 10
```

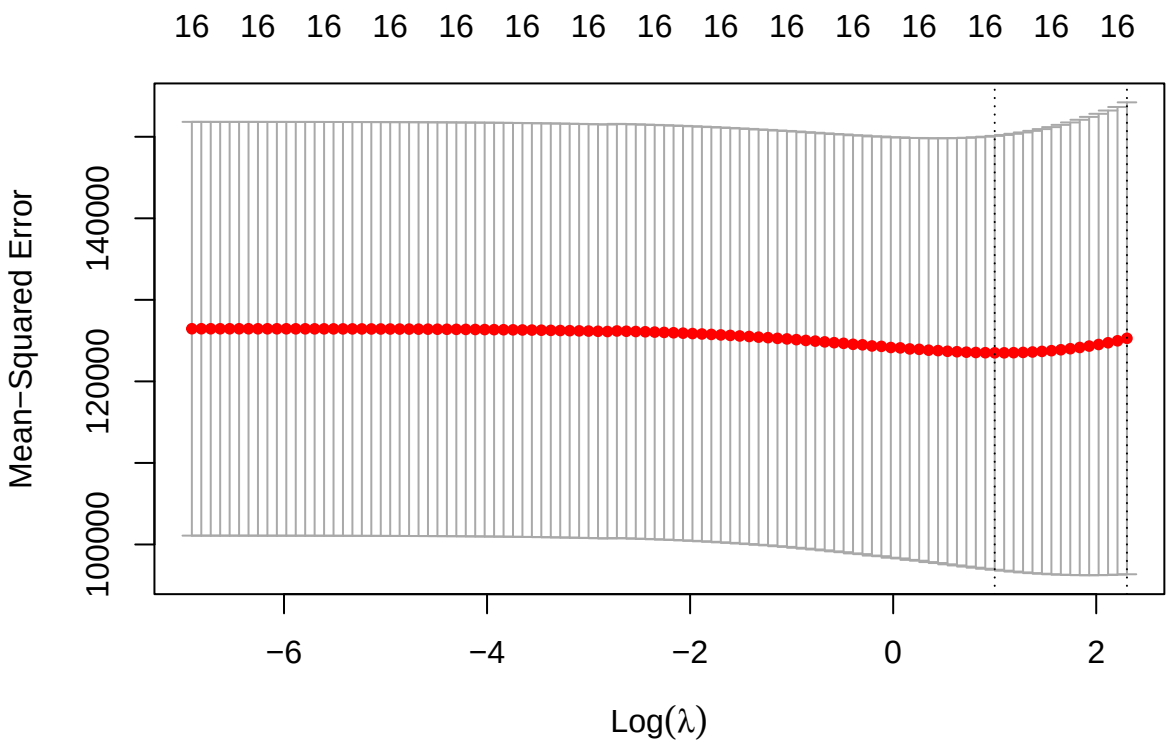
When comparing the minimum lambda (0.977) and the 1-SE lambda (10) from cv.glmnet with our custom minimum lambda (3.2745), our result falls within the range between the two values.

3)

Now we build different models for the date set.

Ridge

```
ridge_model <- cv.glmnet(X_train, y_train, lambda = lambdas, alpha=0)
plot(ridge_model)
```




```
cat("Lambda minimizing MSE:", ridge_model$lambda.min, "\n")
```

```
## Lambda minimizing MSE: 2.718588
```

```
cat("Lambda within 1 SE of MSE:", ridge_model$lambda.1se, "\n")
```

```
## Lambda within 1 SE of MSE: 10
```

Least Square

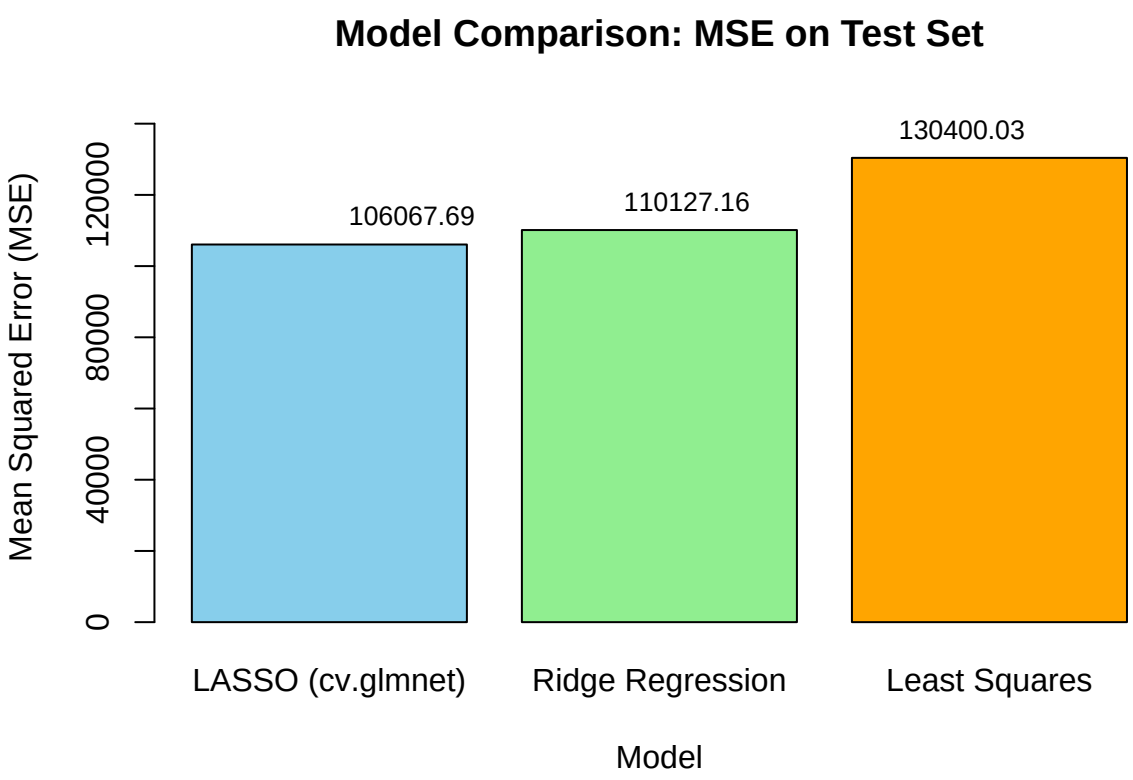
```
least_squares_model <- glmnet(X_train, y_train, alpha = 0, lambda = 0)
```

4)

To compare all models, we calculate the predictions for each model and compute their respective MSE values.

Prediction

```r
y_pred_lasso <- predict(lasso_model, s = lasso_model$lambda.1se, newx = X_test)
mse_lasso <- mean((y_test - y_pred_lasso)^2)

y_pred_ridge <- predict(ridge_model, s = ridge_model$lambda.1se, newx = X_test)
mse_ridge <- mean((y_test - y_pred_ridge)^2)

y_pred_ls <- predict(least_squares_model, s = 0, newx = X_test)
mse_ls <- mean((y_test - y_pred_ls)^2)


mse_values <- c(

  "LASSO (cv.glmnet)" = mse_lasso,
  "Ridge Regression" = mse_ridge,
  "Least Squares" = mse_ls
)

barplot(
  mse_values,
  main = "Model Comparison: MSE on Test Set",
  ylab = "Mean Squared Error (MSE)",
  xlab = "Model",
  col = c("skyblue", "lightgreen", "orange", "red"),
  ylim = c(0, max(mse_values) * 1.1)
)

# Add values above bars
text(
  x = seq_along(mse_values),
  y = mse_values,
  label = round(mse_values, 2),
  pos = 3, cex = 0.8, col = "black"
)
```



From the results, LASSO regression gives the best predictions with the lowestMSE of 106,067. This shows that LASSO effectively balances between fitting the data and avoiding overfitting by shrinking some coefficients to zero, which simplifies the model and improves its generalizability.

Ridge regression comes in second with an MSE of 110,127. Ridge also uses regularization to shrink coefficients, but unlike LASSO, it doesn't set any coefficients to zero. This means Ridge keeps all predictors in the model, which can slightly affect performance when some predictors are less important.

Lastly, Least Squares regression performs the worst, with the highest MSE of 130,400. This method fits the training data without any regularization, making it more likely to overfit, especially when the data has noise or multicollinearity. As a result, its predictions on unseen data are less accurate.

In conclusion, LASSO performs the best because it simplifies the model by removing unnecessary predictors, making it more robust and generalizable to new data. Ridge is close behind, while Least Squares struggles due to overfitting.

Task 3

Regularized regression refers to a method where a penalty term is added to the standard regression model to prevent overfitting. Overfitting occurs when a model becomes too complex and fits the noise in the training data rather than capturing the true relationship. Regularization controls this by shrinking the size of the regression coefficients, which makes the model simpler and more generalizable to unseen data.

The concept of shrinkage describes how the coefficients of the predictors are pushed toward zero as the regularization strength increases. By shrinking the coefficients, the model reduces its sensitivity to individual predictors, especially those that have little impact on the response variable. Two common forms of regularized regression are Ridge Regression and LASSO Regression, which differ in how they apply the shrinkage

Ridge Regression uses an L2 penalty (squared coefficients), shrinking all coefficients closer to zero but never exactly zero. It's useful when predictors are highly correlated because it keeps all variables but reduces their impact.

On the other hand, LASSO Regression uses an L1 penalty (absolute values of coefficients), which not only shrinks coefficients but can set some to exactly zero, effectively performing variable selection. This makes LASSO great for identifying the most important predictors and simplifying the model.

To find the best regularization strength, we use cross-validation. The lambda.min gives the smallest error, while lambda.1se provides a more regularized model that generalizes better. In short, Ridge keeps all predictors but shrinks them, while LASSO selects only the most relevant ones by forcing others to zero.