# Statistical Simulation and Computerintensive Methods

Exercise 7 - Comparing penalized regression estimators

Markus Kiesel | 1228952

18.12.2020

## Contents

# Task 1

## 1.1)

Write your own function for the lasso using the shooting algorithm.

Give default values for the tolerance limit and the maximum number of iterations. Do not forget to compute the coefficient for the intercept.

```r
# create example data
SEED <- 1228952
set.seed(SEED)
n <- 500
p <- 20
X <- scale(matrix(rnorm(n * p), ncol = p))
beta <- 3:5
y <- 2 + X[, 1:3] %*% beta + rnorm(n, sd = 5)

# train test split (400 observations for training)
train_index <- sample(1:500, 400)
x_train <- X[train_index, ]
y_train <- y[train_index]
x_test <- X[-train_index, ]
y_test <- y[-train_index]
```

```r
# lambda: tuning parameter > 0 max_iter: maximal number of iterations eps:
# tolerance for exiting loop
lasso_shooting <- function(x, y, lambda, max_iter = 10^5, eps = 1e-07) {

    # Soft-thresholding of a scalar a at level lambda
    soft <- function(a, delta) {
        sign(a) * pmax((abs(a) - delta), 0)
    }

    p <- ncol(x)
    i <- 0
    diff <- 1
    # create initial beta values (like ridge)
    beta_0 <- mean(y)
    beta_m <- solve(t(x) %*% x - diag(lambda, p, p)) %*% t(x) %*% y
    beta_m1 <- beta_m

    while (i < max_iter & diff > eps) {
        for (j in 1:p) {
            aj <- 2 * sum(x[, j]^2)
            # cj <- 2 * sum(x[,j] * (y - x %*% beta_m1 + beta_m1[j] * x[,j]))
            # cj with intercept
            cj <- 2 * sum(x[, j] * (y - cbind(rep(1, nrow(x)), x) %*% c(beta_0, beta_m1) +
                beta_m1[j] * x[, j]))
            beta_m1[j] <- soft(cj/aj, lambda/aj)
        }
        i <- i + 1
        diff <- sum(abs(beta_m - beta_m1))
        beta_m <- beta_m1
    }
```

```
    return(list(coeff = beta_m1, interc = beta_0, iterations = i))
}
```

### 1.2)

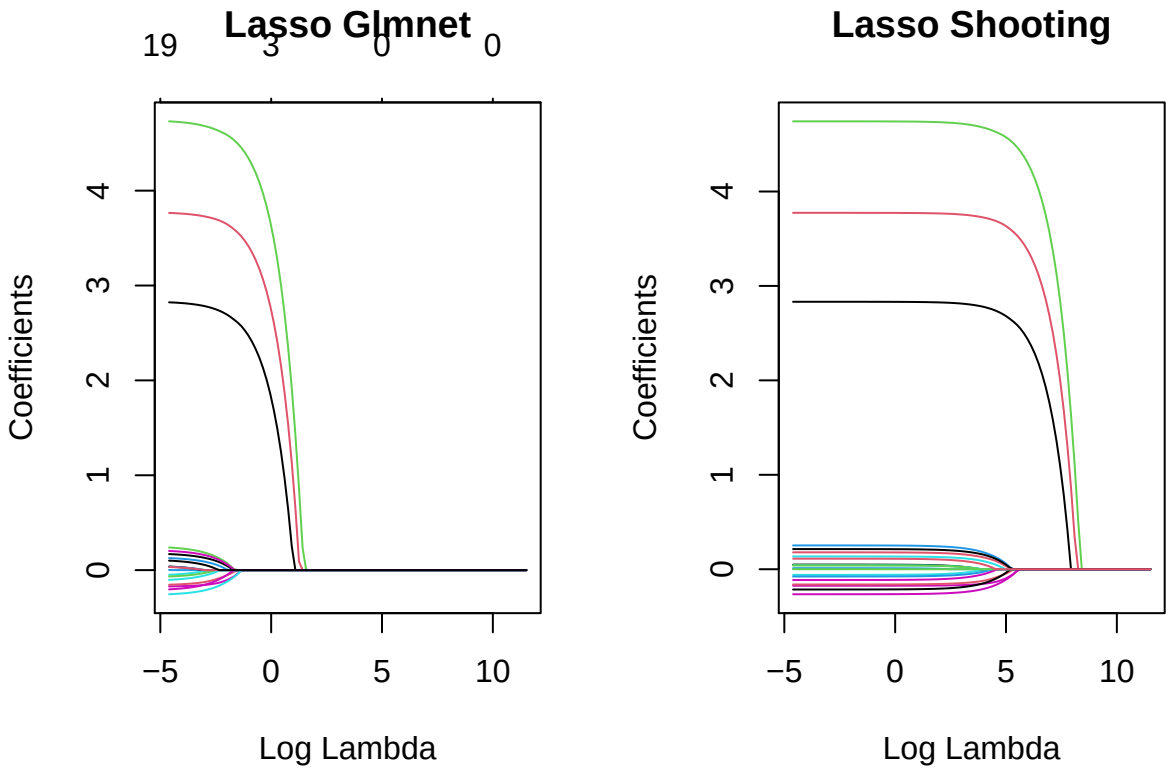
Write a function which computes the lasso using your algorithm for a vector of $\lambda s$ and which returns the matrix of coefficients and the corresponding $\lambda$ values.

```
# wrapper to run lasso algorithm by lambda
lasso_shooting_bylambda <- function(x, y, lambda_grid) {
    n <- length(lambda_grid)
    coeffs <- matrix(0, ncol(x), n)
    for (i in 1:n) {
        lasso <- lasso_shooting(x, y, lambda_grid[i])
        coeffs[, i] <- as.numeric(lasso$coeff)
    }
    return(list(lambda = lambda_grid, coeffs = coeffs, interc = rep(mean(y), n)))
}
```

### 1.3)

Compare the performance and output of your functions against the lasso implementation from glmnet.

```
# set lambda grid
lambda_grid <- 10^seq(-2, 5, length = 100)
# create lasso model using glmnet
model_glmnet <- glmnet(x = X, y = y, alpha = 1, lambda = lambda_grid)
# create lasso model using my implementation
model_my <- lasso_shooting_bylambda(X, y, lambda_grid)
# visualize comparison by lambda
par(mfrow = c(1, 2))
# plot glmnet by log lambda
plot(model_glmnet, xvar = "lambda", main = "Lasso Glmnet")
# plot my coefficents by lambda
matplot(log(model_my$lambda), t(model_my$coeffs), type = "l", lty = 1, ylab = "Coefficients",
    xlab = "Log Lambda", main = "Lasso Shooting")
```




We can see that tha coefficents shrink with increasing lambda. Our algorithm needs a higher lambda the have the same effect but performs otherwise very similar.

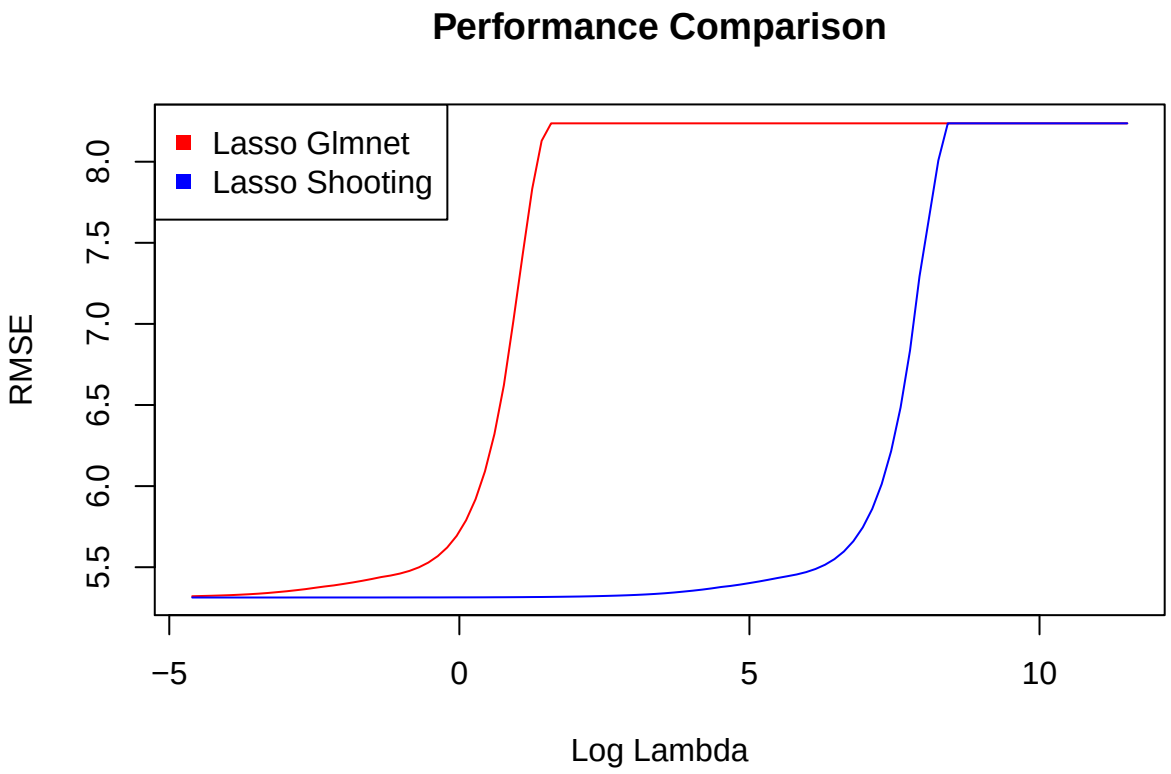
```
eval_from_coeffs <- function(interc, coeff, x, y) {
    # y_hat <- x %*% coeff
    y_hat <- cbind(rep(1, nrow(x)), x) %*% rbind(interc, coeff)
    rmse <- sqrt(apply((y_hat - y)^2, 2, mean))
    return(rmse)
}

rmse_glmnet <- eval_from_coeffs(model_glmnet$a0, model_glmnet$beta[, 100:1], x_test,
    y_test)
rmse_my_lasso <- eval_from_coeffs(model_my$interc, model_my$coeffs, x_test, y_test)

model_glmnet$lambda[which.min(rmse_glmnet[100:1])]
```

```
## [1] 0.01
```

```
model_my$lambda[which.min(rmse_my_lasso)]
```

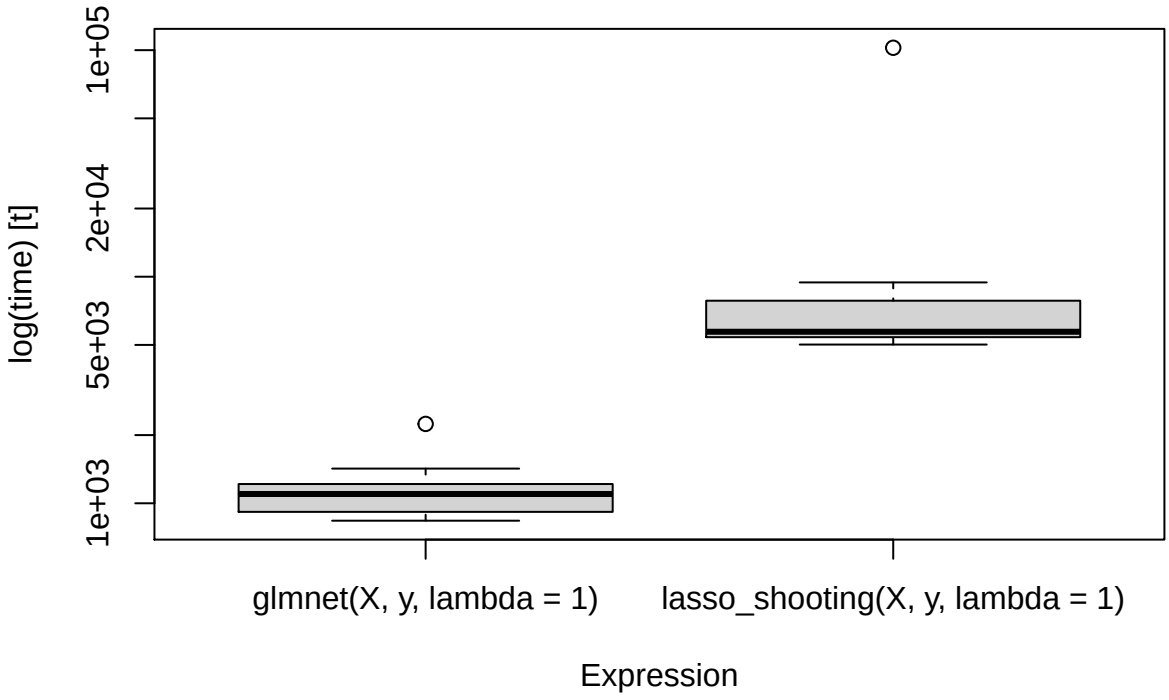
```
## [1] 0.01
```

```
plot(log(lambda_grid), rmse_glmnet, col = "red", type = "l", ylab = "RMSE", xlab = "Log Lambda",
    main = "Performance Comparison")
lines(log(lambda_grid), rmse_my_lasso, col = "blue")
legend("topleft", legend = c("Lasso Glmnet", "Lasso Shooting"), col = c("red", "blue"),
    pch = c(15, 15))
```




The minimal RMSE is similar for our implementation.

```
# benchmark the two models performance
benchmark <- microbenchmark(glmnet(X, y, lambda = 1), lasso_shooting(X, y, lambda = 1))

boxplot(benchmark)
```



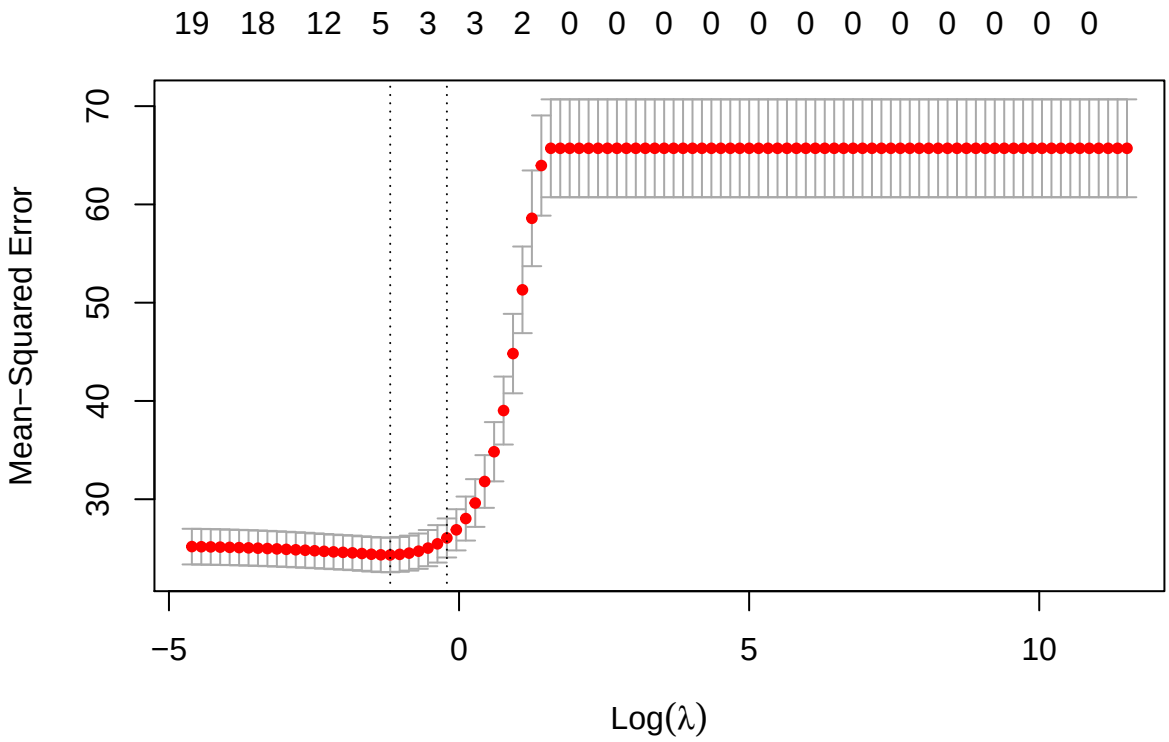

The glmnet implementation is marginally faster than our algorithm.

### 1.4)

Write a function to perform 10-fold cross-validation for the lasso using MSE as the performance measure. The object should be similarly to the cv.glmnet give the same plot and return the $\lambda$ which minimizes the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation, respectively.

```
lasso_shooting_cv <- function(x, y, lambda_grid, metric, K = 10) {
    metrics <- matrix(0, K, length(lambda_grid))
    for (i in 1:length(lambda_grid)) {
        folds <- createFolds(y, K)
        for (j in 1:length(folds)) {
            # train test split by fold
            train_index <- folds[[j]]
            x_train <- x[-train_index, ]
            y_train <- y[-train_index]
            x_test <- x[train_index, ]
            y_test <- y[train_index]
            # fit lasso model
            fit <- lasso_shooting(x_train, y_train, lambda_grid[i])
            # y_hat <- x_test %*% fit$coeff
            y_hat <- cbind(rep(1, nrow(x_test)), x_test) %*% c(fit$interc, fit$coeff)
            # calculate metric
            metrics[j, i] <- metric(y_test, y_hat)
        }
    }
    # calculate mean and sd metrics
    metrics_mean <- apply(metrics, 2, mean)
    metrics_sd <- apply(metrics, 2, sd)
    lambda_min <- lambda_grid[which.min(metrics_mean)]
    return(list(metrics = metrics_mean, metrics_sd = metrics_sd, metrics_raw = metrics,
        lambda = lambda_grid, lambda_min = lambda_min))
}
```

```
# plot glmnet implementation
model_cv.glmnet <- cv.glmnet(x = X, y = y, lambda = lambda_grid)
plot(model_cv.glmnet)
```




```
plot_cv_errors <- function(model_cv, metric_name) {
    df <- data.frame(log_lambda = log(model_cv$lambda), mean = model_cv$metrics,
        sd = model_cv$metrics_sd)
    ggplot(df, aes(x = log_lambda, y = mean)) + geom_point(color = "red", size = 1) +
        geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd), width = 0.3) + geom_vline(xintercept = l
        linetype = "dotted") + xlab("Log Lambda") + ylab(metric_name) + ggtitle(paste(metric_name,
        "CV error by Log Lambda"))
}

# create models to minimize given metric
model_cv_mse <- lasso_shooting_cv(X, y, lambda_grid, mse)
model_cv_rmse <- lasso_shooting_cv(X, y, lambda_grid, rmse)
model_cv_mad <- lasso_shooting_cv(X, y, lambda_grid, mad)

# optimal lambdas
model_cv_mse$lambda_min
```

```
## [1] 148.4968
```

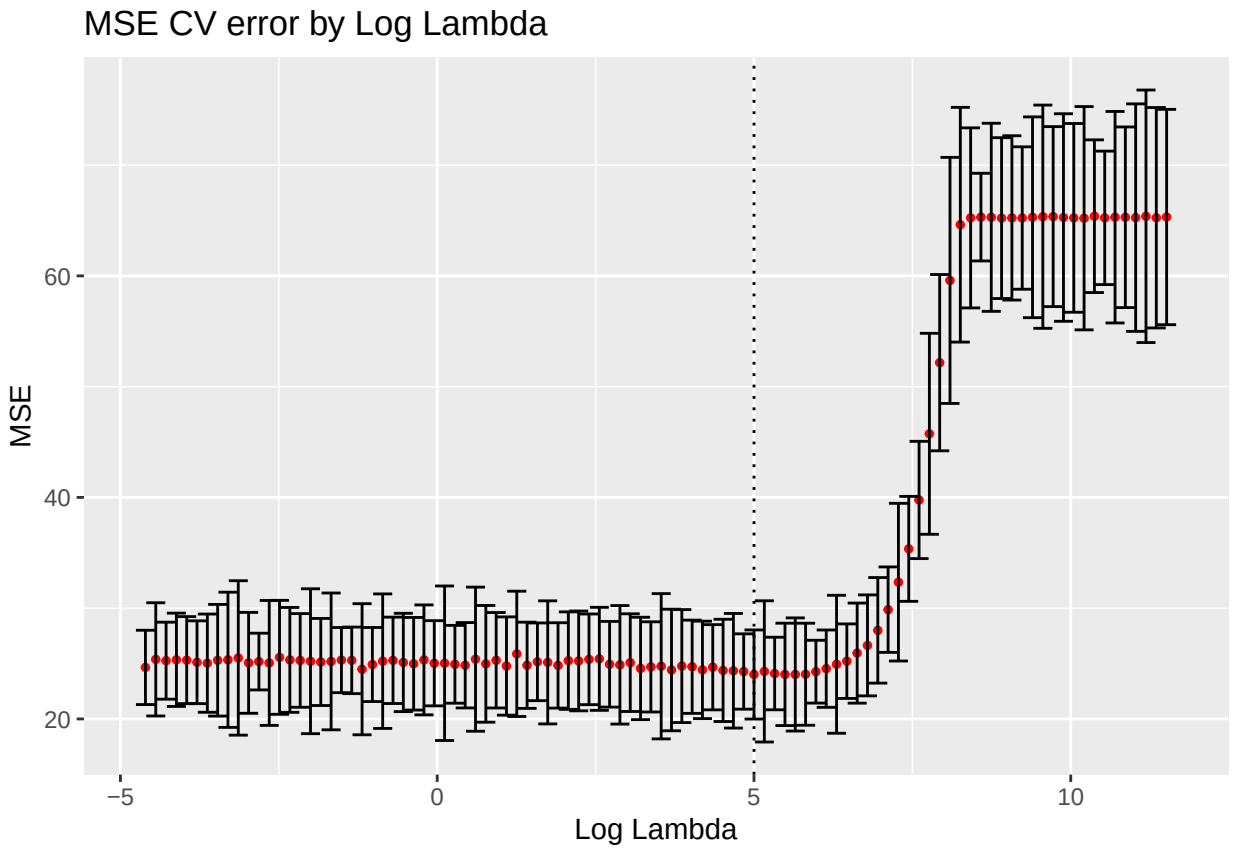
```
model_cv_rmse$lambda_min
```

```
## [1] 284.8036
```

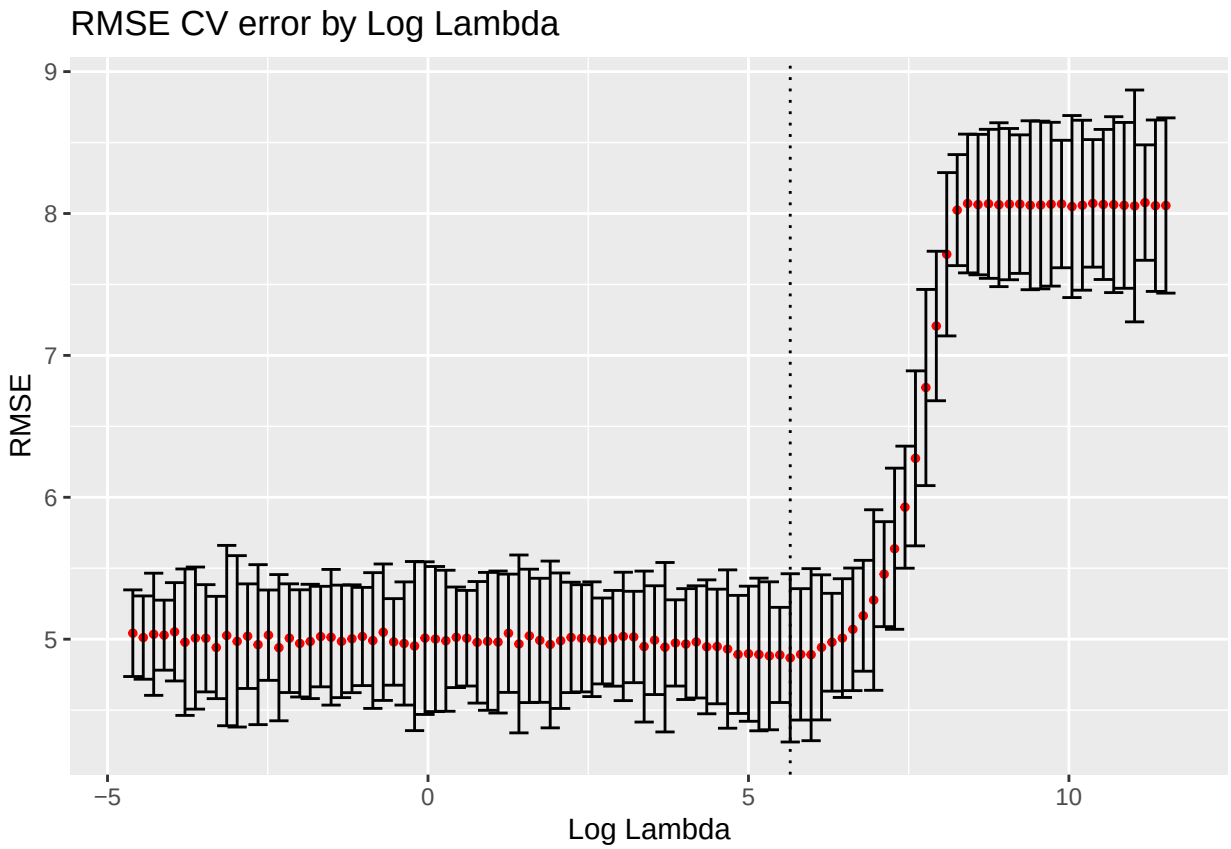
```
model_cv_mad$lambda_min
```

```
## [1] 2.983647
```

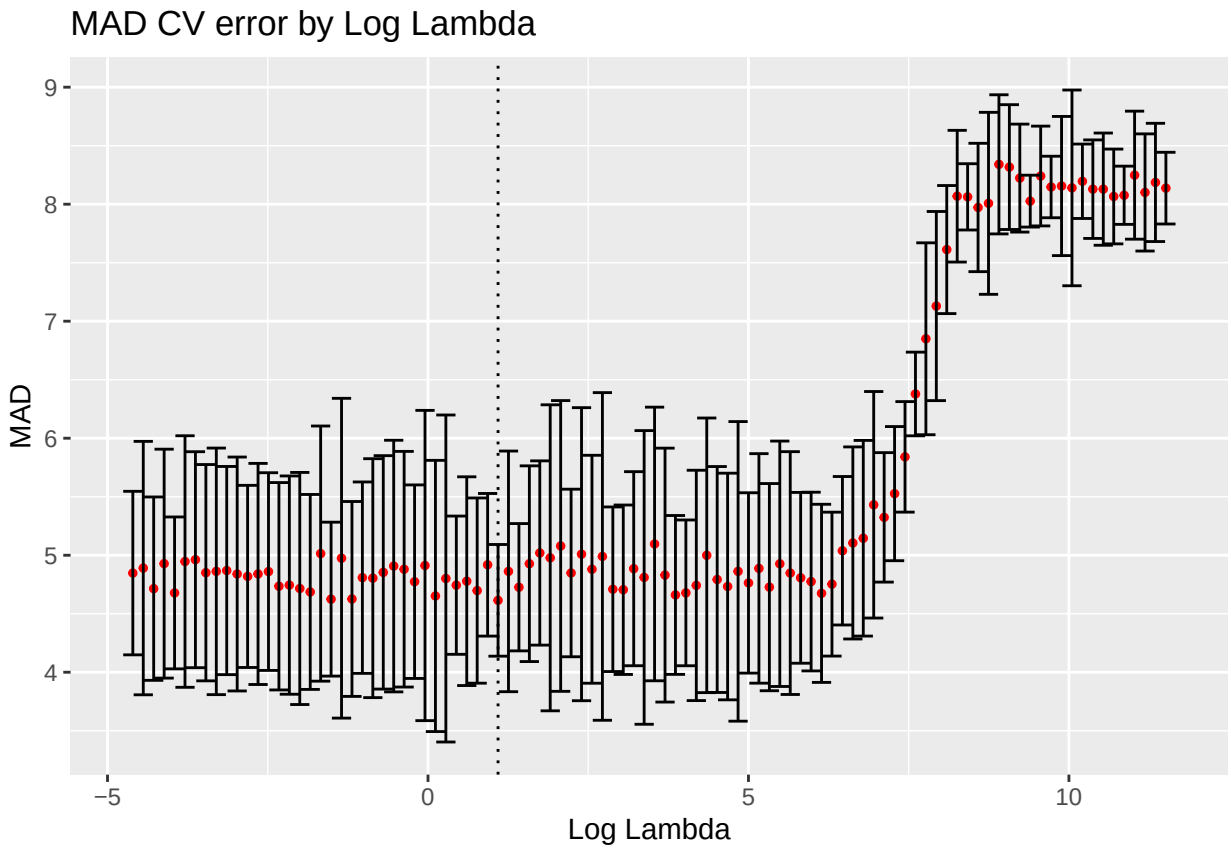
```
# plot the models
plot_cv_errors(model_cv_mse, "MSE")
```




```
plot_cv_errors(model_cv_rmse, "RMSE")
```




```
plot_cv_errors(model_cv_mad, "MAD")
```




## Task 2

We will work with the Hitters data in the ISLR package. Take the salary variable as the response variable and create the model matrix x based on all other variables in the data set. Then divide the data into training and testing data with a ratio of 70:30.

```
# prepare date (1-Hot encode, remove nulls)
str(Hitters)
```

```
## 'data.frame':    322 obs. of  20 variables:
##  $ AtBat    : int  293 315 479 496 321 594 185 298 323 401 ...
##  $ Hits     : int  66 81 130 141 87 169 37 73 81 92 ...
##  $ HmRun    : int  1 7 18 20 10 4 1 0 6 17 ...
##  $ Runs     : int  30 24 66 65 39 74 23 24 26 49 ...
##  $ RBI      : int  29 38 72 78 42 51 8 24 32 66 ...
##  $ Walks    : int  14 39 76 37 30 35 21 7 8 65 ...
##  $ Years    : int  1 14 3 11 2 11 2 3 2 13 ...
##  $ CAtBat   : int  293 3449 1624 5628 396 4408 214 509 341 5206 ...
##  $ CHits    : int  66 835 457 1575 101 1133 42 108 86 1332 ...
##  $ CHmRun   : int  1 69 63 225 12 19 1 0 6 253 ...
##  $ CRuns    : int  30 321 224 828 48 501 30 41 32 784 ...
##  $ CRBI     : int  29 414 266 838 46 336 9 37 34 890 ...
##  $ CWalks   : int  14 375 263 354 33 194 24 12 8 866 ...
##  $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
##  $ PutOuts  : int  446 632 880 200 805 282 76 121 143 0 ...
##  $ Assists  : int  33 43 82 11 40 421 127 283 290 0 ...
```

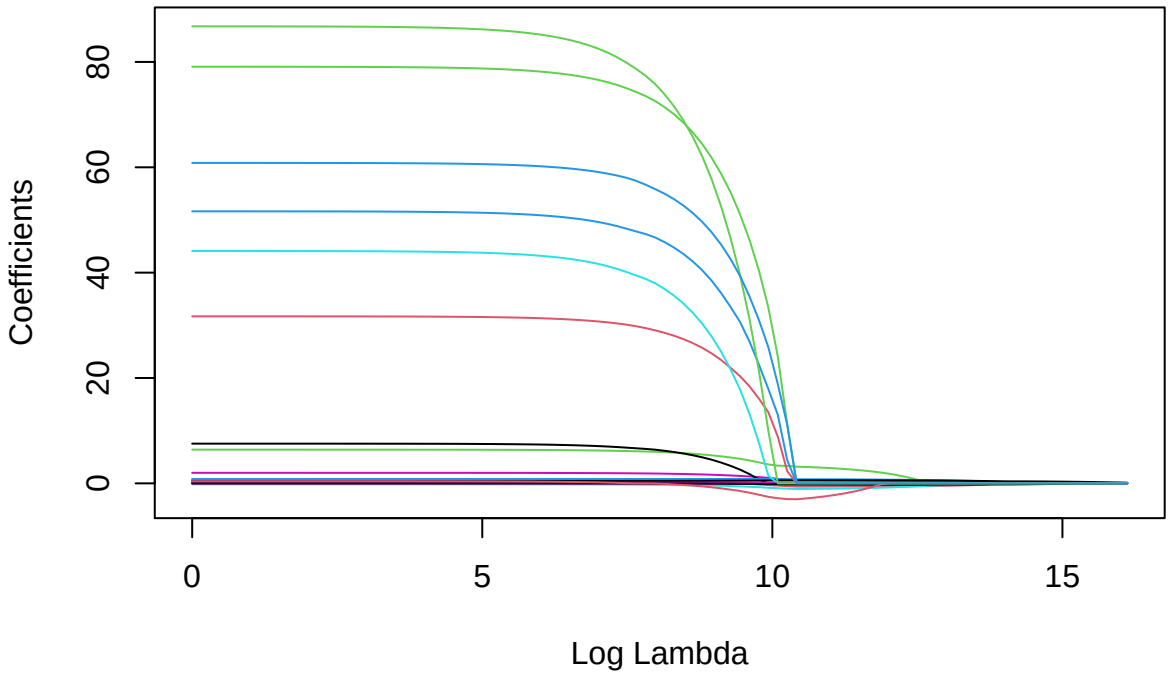
```
##  $ Errors   : int  20 10 14 3 4 25 7 9 19 0 ...
##  $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

```
df <- na.omit(Hitters)
dummy <- dummyVars(" ~ .", data = df)
df <- data.frame(predict(dummy, newdata = df))

# Train-Test split
n <- nrow(df)
train_index <- sample(1:n, round(n * 0.7))
test_index <- c(1:n)[-train_index]
x_train <- data.matrix(df[train_index, -21])
y_train <- df[train_index, 21]
x_test <- data.matrix(df[test_index, -21])
y_test <- df[test_index, 21]

# scale Data
means_train <- apply(x_train, 2, mean)
sds_train <- apply(x_train, 2, sd)
x_train <- (x_train - means_train)/sds_train
x_test <- (x_test - means_train)/sds_train
```

### 2.1)

Use your lasso function to decide which lambda is best here. Plot also the whole path for the coefficients.

```
# set lambda grid
lambda_grid <- 10^seq(0, 7, length = 100)
# run my lasso algorithm
model_my <- lasso_shooting_bylambda(x_train, y_train, lambda_grid)
# plot my coefficients by lambda
matplot(log(model_my$lambda), t(model_my$coeffs), type = "l", lty = 1, ylab = "Coefficients",
    xlab = "Log Lambda", main = "Lasso Shooting")
```




```
# create models to minimize given metric
model_cv_mse <- lasso_shooting_cv(x_train, y_train, lambda_grid, mse)
model_cv_rmse <- lasso_shooting_cv(x_train, y_train, lambda_grid, rmse)
model_cv_mad <- lasso_shooting_cv(x_train, y_train, lambda_grid, mad)

# optimal lambdas
model_cv_mse$lambda_min
```

```
## [1] 1788.65
```

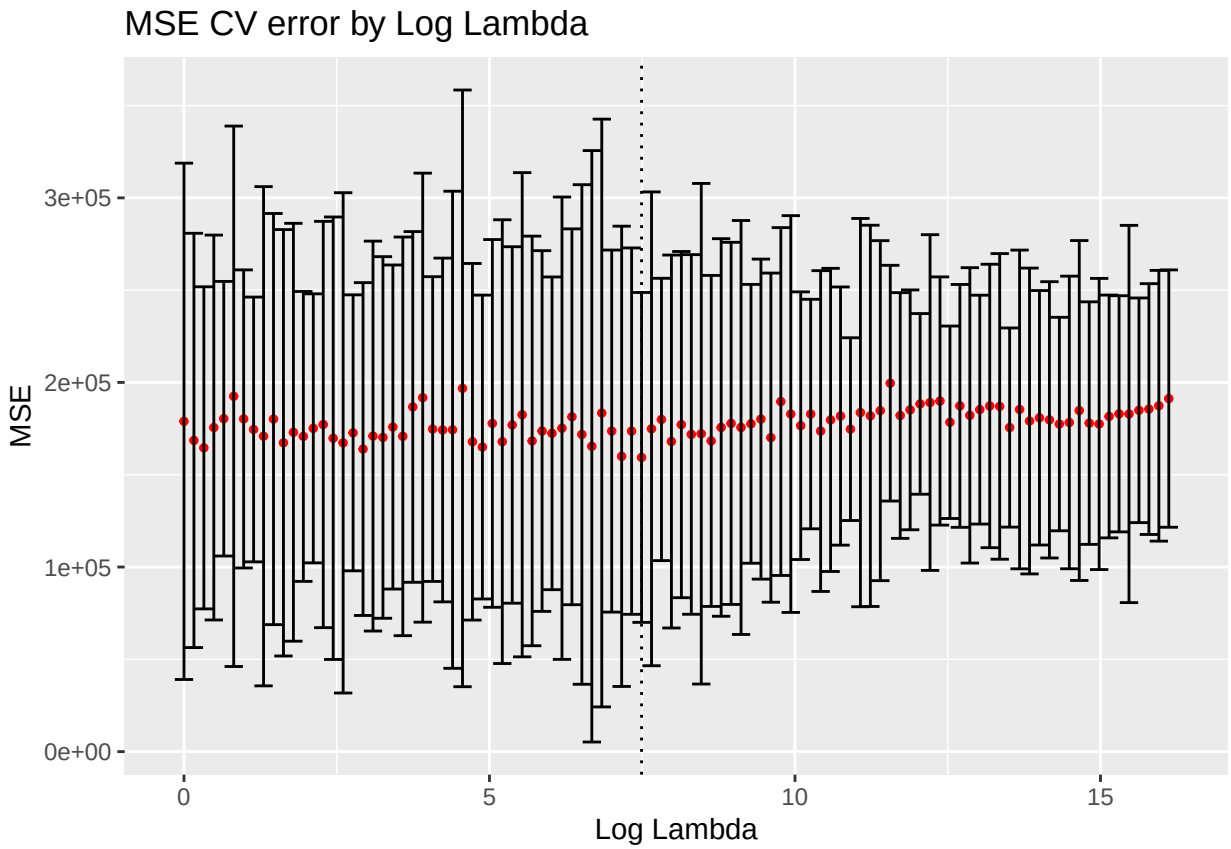
```
model_cv_rmse$lambda_min
```

```
## [1] 1.629751
```

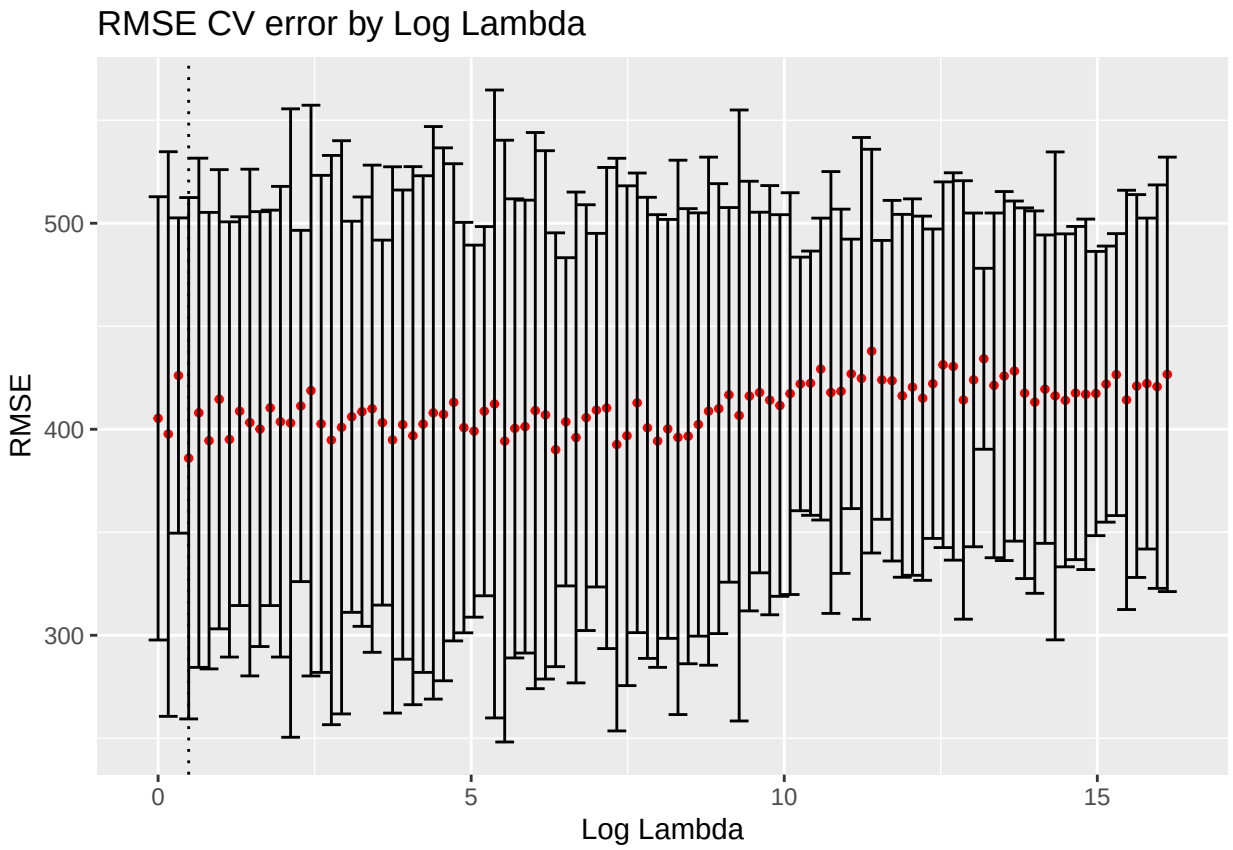
```
model_cv_mad$lambda_min
```

```
## [1] 112.3324
```

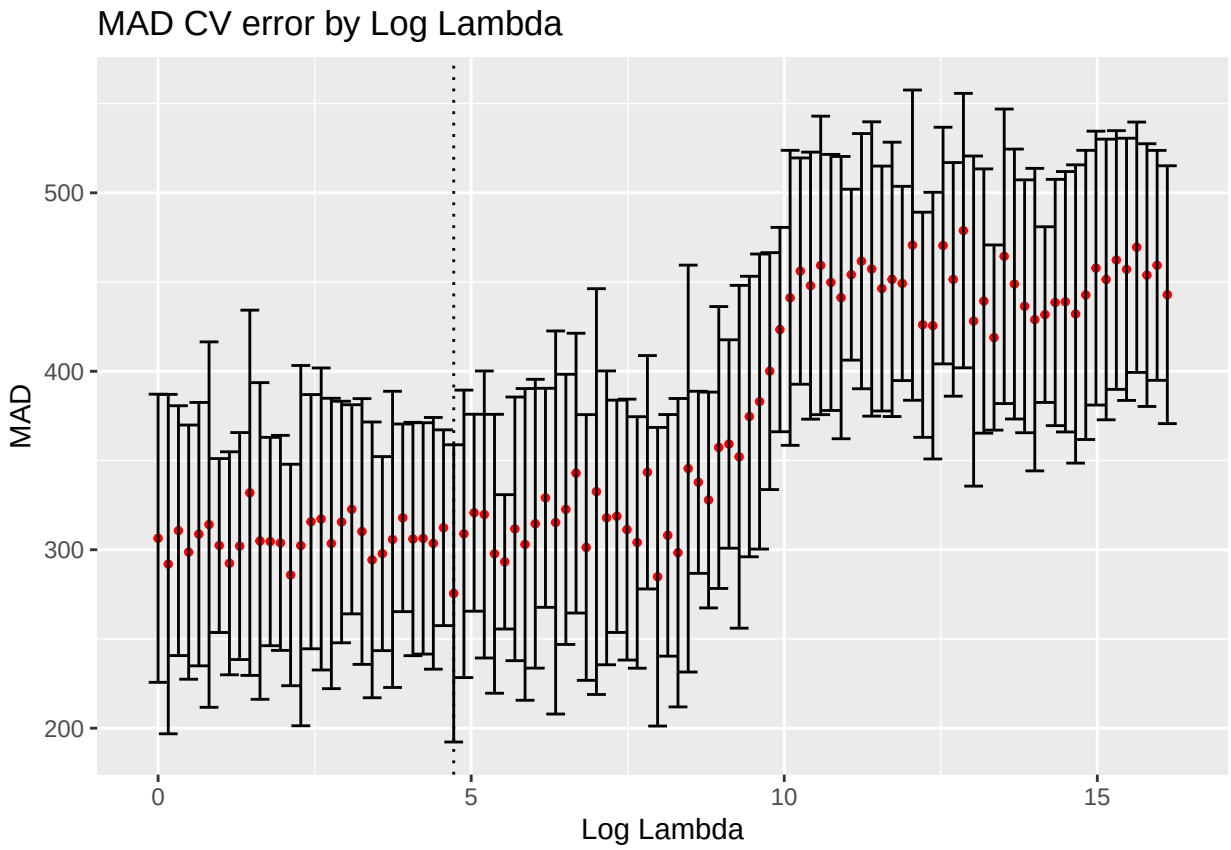
```
plot_cv_errors(model_cv_mse, "MSE")
```




```
plot_cv_errors(model_cv_rmse, "RMSE")
```




```
plot_cv_errors(model_cv_mad, "MAD")
```



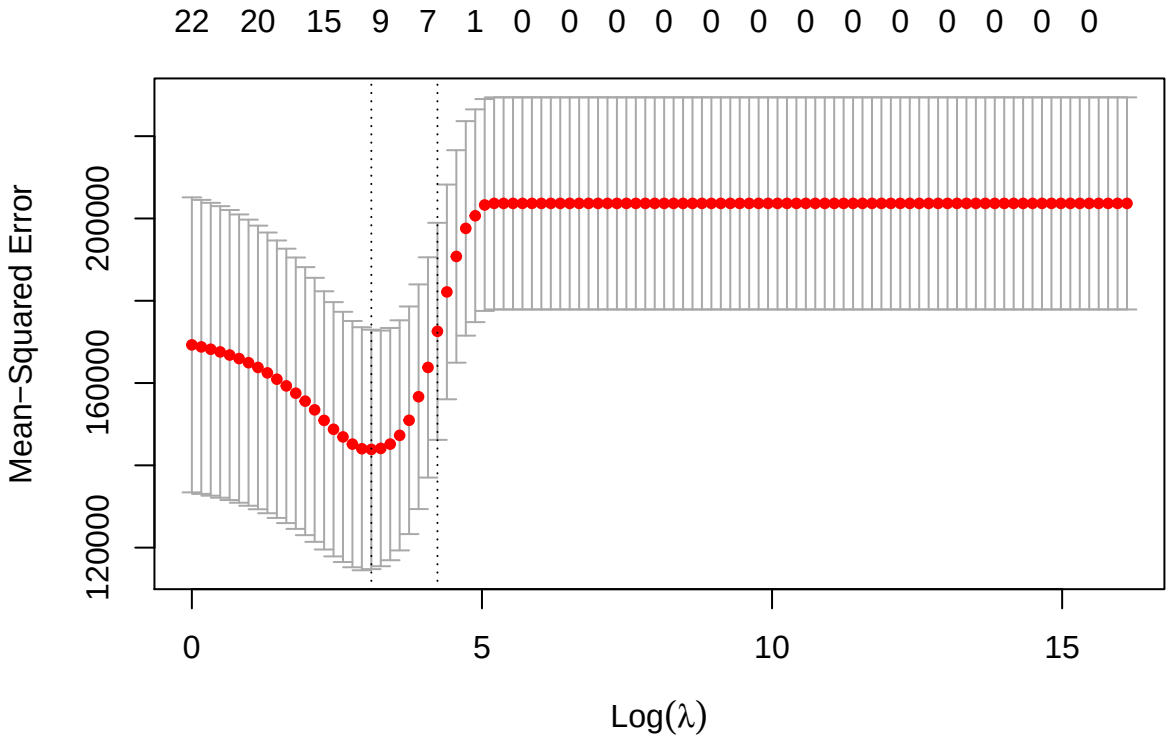

### 2.2)

Compare your fit against the lasso implementation from glmnet.

```
# use glmnet implementation to find lambda min
model_lasso <- cv.glmnet(x = x_train, y = y_train, alpha = 1, lambda = lambda_grid)
model_lasso$lambda.min
```

```
## [1] 22.05131
```

```
plot(model_lasso)
```




The lambda min is very similar to the lambda min found by our algorithm.

### 2.3)

Fit also a ridge regression and a least squares regression for the data (you can use here glmnet).

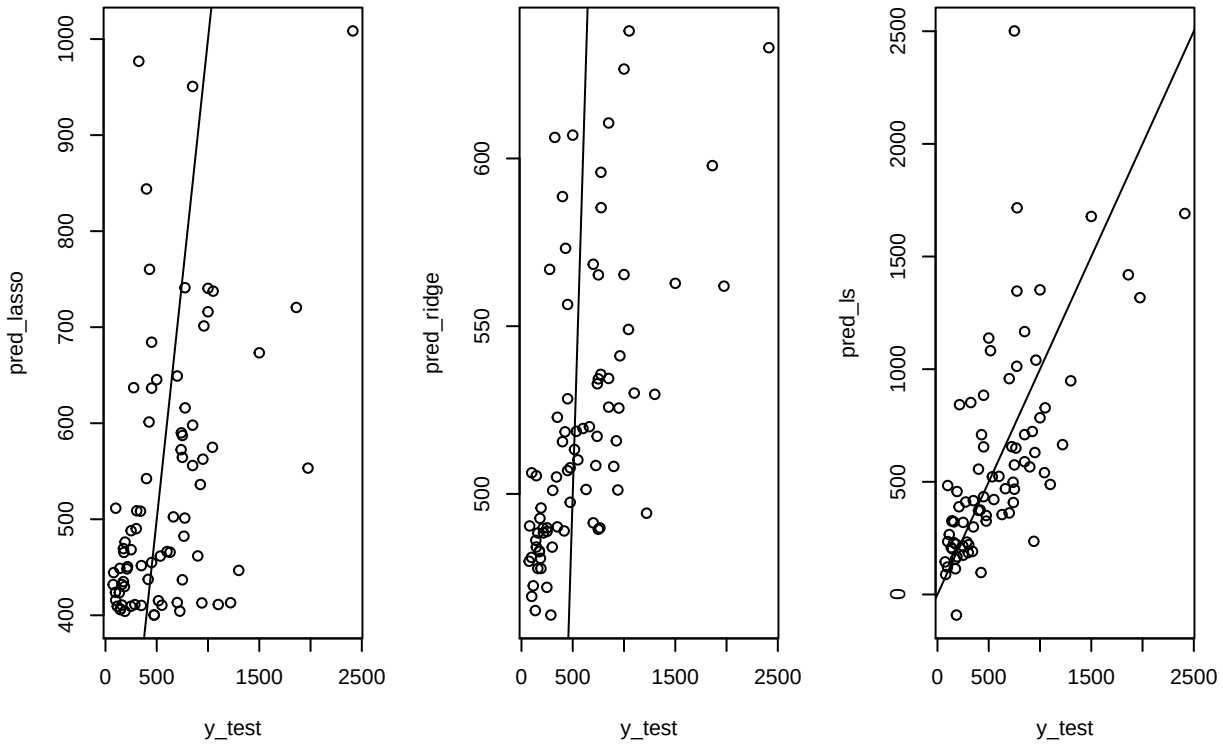
```
# fit ridge regression
model_ridge <- cv.glmnet(x = x_train, y = y_train, alpha = 0, lambda = lambda_grid)
# fit least squares regression
model_ls <- lm(Salary ~ ., data = df, subset = train_index)
```

### 2.4)

Compute the lasso, ridge regression and ls regression predictions for the testing data. Which method gives the better predictions? Interpret all three models and argue about their performances.

```
# create models
pred_lasso <- predict(model_lasso, x_test)
pred_ridge <- predict(model_ridge, x_test)
pred_ls <- predict(model_ls, df[test_index, ])
```

```
# plot predicted vs actual
par(mfrow = c(1, 3))
plot(y_test, pred_lasso)
abline(c(0, 1))
plot(y_test, pred_ridge)
abline(c(0, 1))
plot(y_test, pred_ls)
abline(c(0, 1))
```




```
results <- matrix(0, 3, 3)

results[1, 1] <- rmse(y_test, pred_lasso)
results[2, 1] <- rmse(y_test, pred_ridge)
results[3, 1] <- rmse(y_test, pred_ls)

results[1, 2] <- mse(y_test, pred_lasso)
results[2, 2] <- mse(y_test, pred_ridge)
results[3, 2] <- mse(y_test, pred_ls)

results[1, 3] <- mad(y_test, pred_lasso)
results[2, 3] <- mad(y_test, pred_ridge)
results[3, 3] <- mad(y_test, pred_ls)

colnames(results) <- c("RMSE", "MSE", "MAD")
rownames(results) <- c("LASSO", "RIDGE", "LM")

kable(results)
```

| | RMSE | MSE | MAD |
|---|---|---|---|
| LASSO | 405.4823 | 164415.9 | 384.8273 |
| RIDGE | 428.0103 | 183192.8 | 407.9159 |
| LM | 369.6942 | 136673.8 | 269.3187 |

The Linear Model seems to perfom best in our case. It can be the case that the scaling of the data has a negative impact on the results.

## Task 3

Explain the notion of regularised regression, shrinkage and how Ridge regression and LASSO regression differ.

The main motivation of regularised regression was that it adds a positive constant to the diagonal of $X^T X$ before inversion. This makes the problem nonsingular, even if $X^T X$ is not of full rank.

Shrinkage methods keep all variables in the model and assign different weights. In this way we obtain a smoother procedure with a smaller variability. Ridge regression shrinks the coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized residual sum of squares. By penalizing the RSS we try to avoid that highly correlated regressors.

Lasso and ridge differ in their penalty term (lasso minimizes the absolute value of residuals). The lasso solutions are nonlinear and a quadratic. This will cause some of the coefficients to be exactly 0. So Lasso can be used as feature selection.

The lasso is a shrinkage method like ridge, but L1 norm rather than the L2 norm is used in the constraints.

Both methods use a tuning parameter lambda which determines the penalty. The optimal parameter for both can be usually found by cross validation. The larger the value of lambdda, the greater the amount of shrinkage. The coefficients are shrunk towards zero and towards each other.

In R we can use the Elastic Net (glmnet) to combine ridge and lasso.