P8106 - HW1

Ravi Brenner

Contents

Data and problem 1
a. Lasso model 2
b. Elastic net 5
c. Partial least squares 8
d. Model selection 9
e. Lasso using glmnet 10

```r
library(glmnet)
library(caret)
library(tidymodels)
library(pls)
library(readr)
```

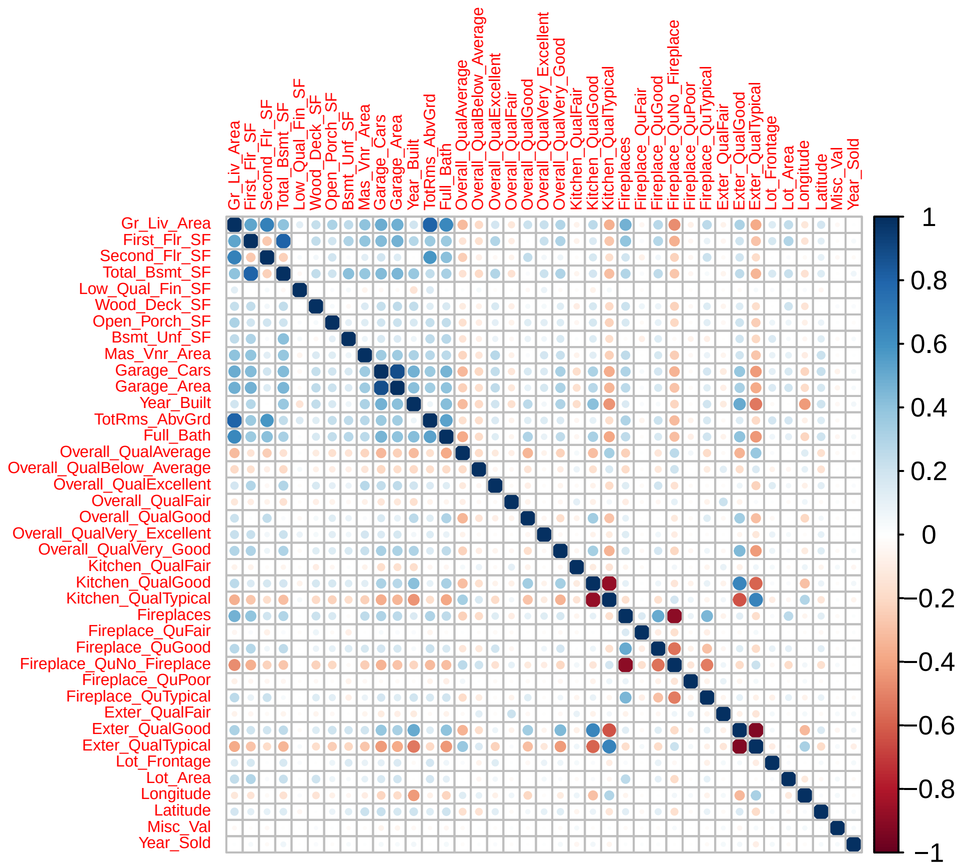
Data and problem

Load data

```r
training <- read_csv("housing_training.csv")
testing <- read_csv("housing_test.csv")
```

Very brief exploration of the training data

```r
x <- model.matrix(Sale_Price ~ ., training)[,-1]
corrplot::corrplot(cor(x),
                   method = "circle", type = "full", tl.cex = 0.5)
```



a. Lasso model

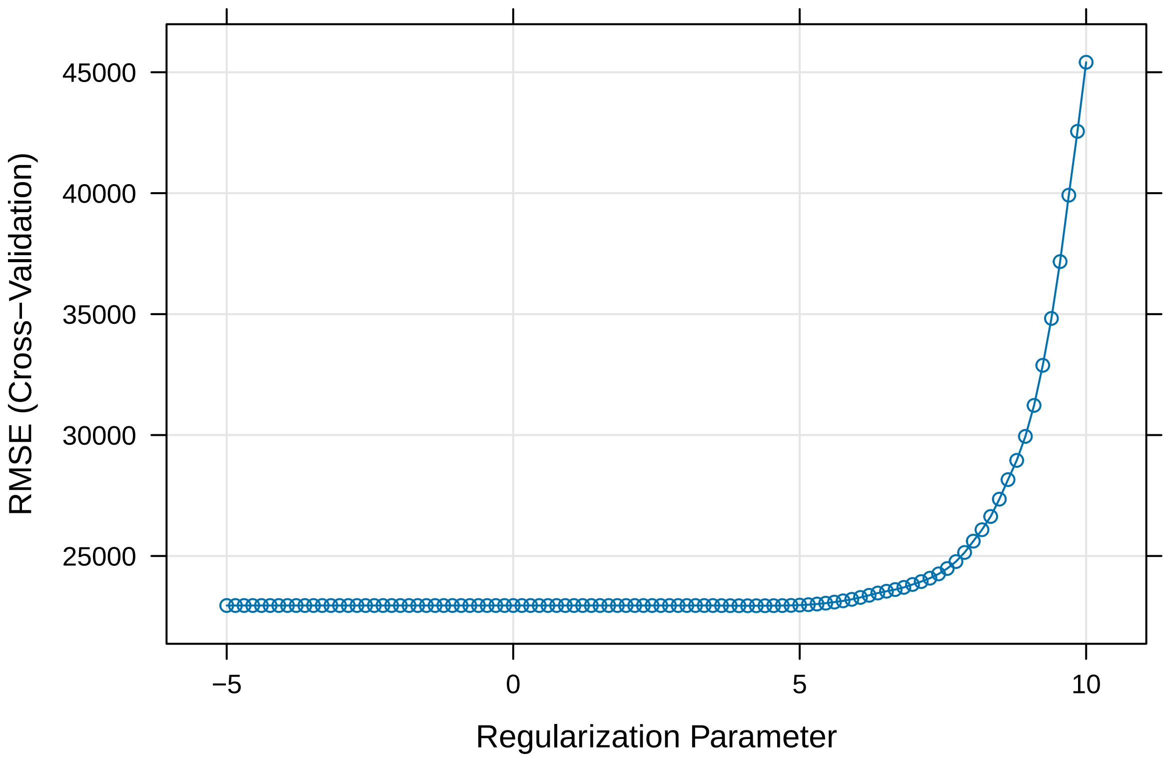
(a) Fit a lasso model on the training data. Report the selected tuning parameter and the test error. When the 1SE rule is applied, how many predictors are included in the model?

Fit model

```r
ctrl1 <- trainControl(method = "cv", number = 10, selectionFunction = "best")

set.seed(2025)
lasso.fit <- train(Sale_Price ~ .,
                   data = training,
                   method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1,
                                          lambda = exp(seq(10, -5, length = 100))),
                   trControl = ctrl1)

plot(lasso.fit, xTrans = log)
```



Best tuning parameter

```
best_lambda <- lasso.fit$bestTune$lambda
best_lambda
```

```
## [1] 69.57632
```

test error

```
lasso.pred <- predict(lasso.fit, newdata = testing)
mean((lasso.pred - testing[, "Sale_Price"])^2 |> pull()) #MSE
```

```
## [1] 439568493
```

```
mean((lasso.pred - testing[, "Sale_Price"])^2 |> pull()) |> sqrt() #RMSE
```

```
## [1] 20965.89
```

How many coefficients in 1se model?

First, find 1se lambda value

```
# fit 1se version
ctrl2 <- trainControl(method = "cv", number = 10, selectionFunction = "oneSE")
set.seed(2025)
lasso.1se <- train(Sale_Price ~ .,
                   data = training,
                   method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1,
                                          lambda = exp(seq(10, -5, length = 100))),
                   trControl = ctrl2)

lambda_1se <- lasso.1se$bestTune |> pull(lambda)
```

We can also calculate this value manually. The model output `results` gives the RMSE and RMSESD. To convert the SD to SE, we have to divide by the square root (because it is *root* mean square error) of the square root of the fold size (the standard way to calculate SE from SD). In this case the fold size is 144, so we divide by the square root of 12.

```
rmse_1se <- lasso.fit$results |>
  filter(lambda == best_lambda) |>
  mutate(RMSESE = RMSESD / sqrt(12),
         RMSE_1SE = RMSE + RMSESE) |>
  pull(RMSE_1SE)

lasso.fit$results |>
  filter(RMSE > 23500,
         RMSE < 23600)
```

```
##   alpha   lambda     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1     1 675.2963 23542.89 0.8973281 16778.67 2281.745 0.01771693 1527.013
```

Model with 1se lambda value:

```
coef(lasso.fit$finalModel, lambda_1se)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                      s1
## (Intercept)                -2.916836e+06
## Gr_Liv_Area                 5.794761e+01
## First_Flr_SF                1.006061e+00
## Second_Flr_SF               .
## Total_Bsmt_SF               3.674362e+01
## Low_Qual_Fin_SF            -2.879529e+01
## Wood_Deck_SF                8.702628e+00
## Open_Porch_SF               9.147142e+00
## Bsmt_Unf_SF                -2.005078e+01
## Mas_Vnr_Area                1.402927e+01
## Garage_Cars                 3.115356e+03
## Garage_Area                 1.082411e+01
## Year_Built                  3.110661e+02
## TotRms_AbvGrd              -1.695581e+03
## Full_Bath                   .
## Overall_QualAverage        -3.311972e+03
## Overall_QualBelow_Average  -9.646160e+03
## Overall_QualExcellent       9.132387e+04
## Overall_QualFair           -7.183517e+03
## Overall_QualGood            1.031218e+04
## Overall_QualVery_Excellent  1.614854e+05
## Overall_QualVery_Good       3.677171e+04
## Kitchen_QualFair           -5.938606e+03
## Kitchen_QualGood            .
## Kitchen_QualTypical        -9.520503e+03
## Fireplaces                  6.652772e+03
## Fireplace_QuFair           -1.054475e+02
## Fireplace_QuGood            4.425741e+03
## Fireplace_QuNo_Fireplace    .
## Fireplace_QuPoor            .
## Fireplace_QuTypical        -9.329979e+02
## Exter_QualFair             -1.522564e+04
```

```
## Exter_QualGood              .
## Exter_QualTypical          -4.883074e+03
## Lot_Frontage                7.603302e+01
## Lot_Area                    5.739267e-01
## Longitude                  -1.428573e+04
## Latitude                    2.408926e+04
## Misc_Val                    .
## Year_Sold                   .
```

```r
# Number of predictors
coef(lasso.fit$finalModel, lambda_1se) |> as.matrix() |> as.data.frame() |> filter(s1 > 0) |> nrow()
```

```
## [1] 18
```

There are 18 predictors in the 1SE lasso model, compared to 25 predictors in the whole data set.

b. Elastic net

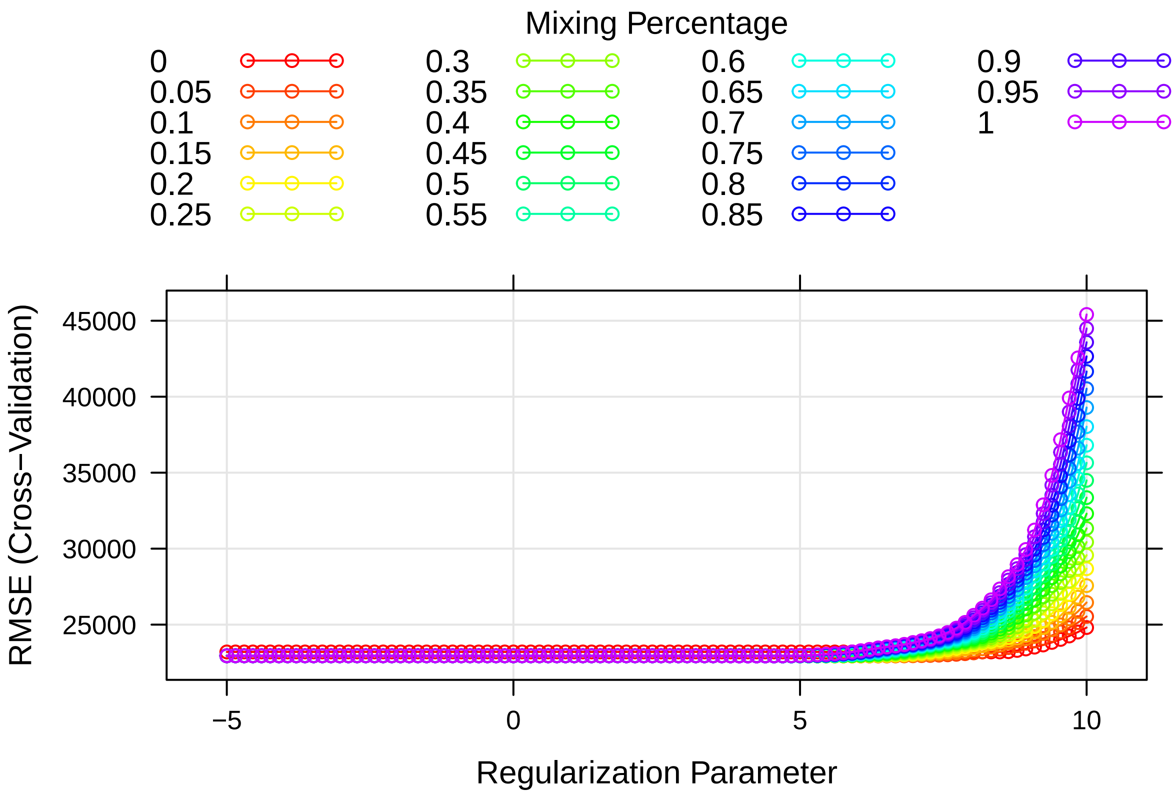
(b) Fit an elastic net model on the training data. Report the selected tuning parameters and the test error. Is it possible to apply the 1SE rule to select the tuning parameters for elastic net? If the 1SE rule is applicable, implement it to select the tuning parameters. If not, explain why.

Fit Elastic net model

```r
set.seed(2025)
enet.fit <- train(Sale_Price ~ .,
                  data = training,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                         lambda = exp(seq(10, -5, length = 100))),
                  trControl = ctrl1)


myCol <- rainbow(25)
myPar <- list(superpose.symbol = list(col = myCol),
              superpose.line = list(col = myCol))

plot(enet.fit, par.settings = myPar, xTrans = log)
```




```r
# coefficients in the final model
coef(enet.fit$finalModel, s = enet.fit$bestTune$lambda)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                     s1
## (Intercept)                -5.112158e+06
## Gr_Liv_Area                 3.877699e+01
## First_Flr_SF                2.670023e+01
## Second_Flr_SF               2.545573e+01
## Total_Bsmt_SF               3.494355e+01
## Low_Qual_Fin_SF            -1.586366e+01
## Wood_Deck_SF                1.232450e+01
## Open_Porch_SF               1.687896e+01
## Bsmt_Unf_SF                -2.072832e+01
## Mas_Vnr_Area                1.165193e+01
## Garage_Cars                 4.046992e+03
## Garage_Area                 8.889587e+00
## Year_Built                  3.192146e+02
## TotRms_AbvGrd              -3.441518e+03
## Full_Bath                  -3.694920e+03
## Overall_QualAverage        -5.116544e+03
## Overall_QualBelow_Average  -1.270853e+04
## Overall_QualExcellent       7.582058e+04
## Overall_QualFair           -1.147242e+04
## Overall_QualGood            1.198403e+04
## Overall_QualVery_Excellent  1.363756e+05
## Overall_QualVery_Good       3.765989e+04
## Kitchen_QualFair           -2.369556e+04
## Kitchen_QualGood           -1.611428e+04
```

```
## Kitchen_QualTypical        -2.416472e+04
## Fireplaces                  1.083070e+04
## Fireplace_QuFair           -7.857118e+03
## Fireplace_QuGood            1.490291e+02
## Fireplace_QuNo_Fireplace    1.823280e+03
## Fireplace_QuPoor           -5.803365e+03
## Fireplace_QuTypical        -6.962039e+03
## Exter_QualFair             -3.297575e+04
## Exter_QualGood             -1.457261e+04
## Exter_QualTypical          -1.916633e+04
## Lot_Frontage                1.001768e+02
## Lot_Area                    6.032398e-01
## Longitude                  -3.515129e+04
## Latitude                    5.776171e+04
## Misc_Val                    8.685961e-01
## Year_Sold                  -5.749899e+02
```

Best tuning parameters alpha and lambda

```
enet.fit$bestTune
```

```
##     alpha   lambda
## 176  0.05 580.3529
```

test error

```
enet.pred <- predict(enet.fit, newdata = testing)
mean((enet.pred - testing[, "Sale_Price"])^2 |> pull()) #MSE
```

```
## [1] 438615743
```

```
mean((enet.pred - testing[, "Sale_Price"])^2 |> pull()) |> sqrt() #RMSE
```

```
## [1] 20943.16
```

Mechanically speaking, it is possible to apply the 1SE rule to elastic net:

```
# fit 1se version
ctrl2 <- trainControl(method = "cv", number = 10, selectionFunction = "oneSE")
set.seed(2025)
enet.1se <- train(Sale_Price ~ .,
                  data = training,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                         lambda = exp(seq(10, -5, length = 100))),
                  trControl = ctrl2)

enet.1se$bestTune
```

```
##    alpha  lambda
## 94     0 8874.25
```

```
enet.fit$results |>
  filter(lambda == enet.fit$bestTune |> pull(lambda),
         alpha == enet.fit$bestTune |> pull(alpha)) |>
  mutate(RMSESE = RMSESD / sqrt(12),
         RMSE_1SE = RMSE + RMSESE) |>
  pull(RMSE_1SE)
```

```
## [1] 23531.45
```

```
enet.fit$results |>
  filter(RMSE > 23500,
         RMSE < 23600)
```

```
##    alpha    lambda     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1   0.00 8874.2499 23519.64 0.8984303 16761.08 2374.337 0.01694508 1441.975
## 2   0.10 4160.2620 23519.39 0.8980426 16711.68 2355.696 0.01724977 1501.709
## 3   0.20 2640.6698 23533.60 0.8977256 16719.61 2342.145 0.01741654 1522.439
## 4   0.25 2269.4045 23551.72 0.8975295 16734.28 2345.117 0.01748456 1530.335
## 5   0.30 1950.3372 23552.90 0.8974688 16740.23 2340.162 0.01753041 1532.095
## 6   0.35 1676.1293 23540.45 0.8975203 16738.65 2327.784 0.01754849 1530.642
## 7   0.40 1440.4737 23516.82 0.8976635 16730.12 2308.848 0.01753529 1525.858
## 8   0.45 1440.4737 23581.69 0.8971593 16769.62 2334.959 0.01765471 1540.065
## 9   0.50 1237.9501 23544.89 0.8974051 16756.52 2309.808 0.01762218 1531.465
## 10  0.55 1063.9003 23500.92 0.8977170 16736.79 2283.527 0.01757529 1521.637
## 11  0.55 1237.9501 23586.43 0.8970913 16779.15 2330.421 0.01770803 1541.007
## 12  0.60 1063.9003 23548.71 0.8973405 16765.87 2301.477 0.01767086 1531.244
## 13  0.65  914.3211 23500.88 0.8976889 16744.02 2275.420 0.01760253 1521.442
## 14  0.65 1063.9003 23579.91 0.8971107 16782.52 2319.611 0.01773336 1538.715
## 15  0.70  914.3211 23539.90 0.8973802 16767.27 2290.061 0.01768902 1528.191
## 16  0.75  914.3211 23566.41 0.8971865 16781.60 2305.820 0.01773699 1535.181
## 17  0.80  785.7720 23524.48 0.8974788 16764.19 2275.506 0.01766438 1522.893
## 18  0.80  914.3211 23592.75 0.8969926 16794.16 2320.076 0.01778222 1541.487
## 19  0.85  785.7720 23547.45 0.8973089 16775.94 2289.412 0.01771678 1529.466
## 20  0.90  785.7720 23568.71 0.8971532 16786.94 2301.596 0.01775285 1534.844
## 21  0.95  675.2963 23525.42 0.8974557 16769.99 2270.027 0.01768056 1521.366
## 22  0.95  785.7720 23591.16 0.8969892 16798.55 2314.198 0.01778728 1539.736
## 23  1.00  675.2963 23542.89 0.8973281 16778.67 2281.745 0.01771693 1527.013
```

Although it is possible to select a 1SE lambda value using the available caret functions, I do not think this is the best idea for elastic net. It is very challenging to accurately apply the 1SE rule to the elastic net model, because there are multiple combinations of alpha and lambda that yield similar RMSE values which are close to the 1SE value. You could further refine the grid to create a more precise search for the exact 1SE value, but ultimately multiple combinations of alpha and lambda will remain as potential solutions. The new value of lambda that corresponds to the "1SE solution" also comes with a different alpha value, meaning you are then jumping onto a different curve to compare RMSE values, and thereby changing the meaning of the lambda value too. It is therefore hard to interpret which combination of values is best with this added flexibility. Thus I would not use the 1SE method with elastic net.

c. Partial least squares

(c) Fit a partial least squares model on the training data and report the test error. How many components are included in your model?

```
set.seed(2025)

pls.fit <- train(Sale_Price ~ .,
                 data = training,
                 method = "pls",
                 tuneGrid = data.frame(ncomp = 1:(ncol(training)-1)),
                 trControl = ctrl1,
                 preProcess = c("center","scale"))
```

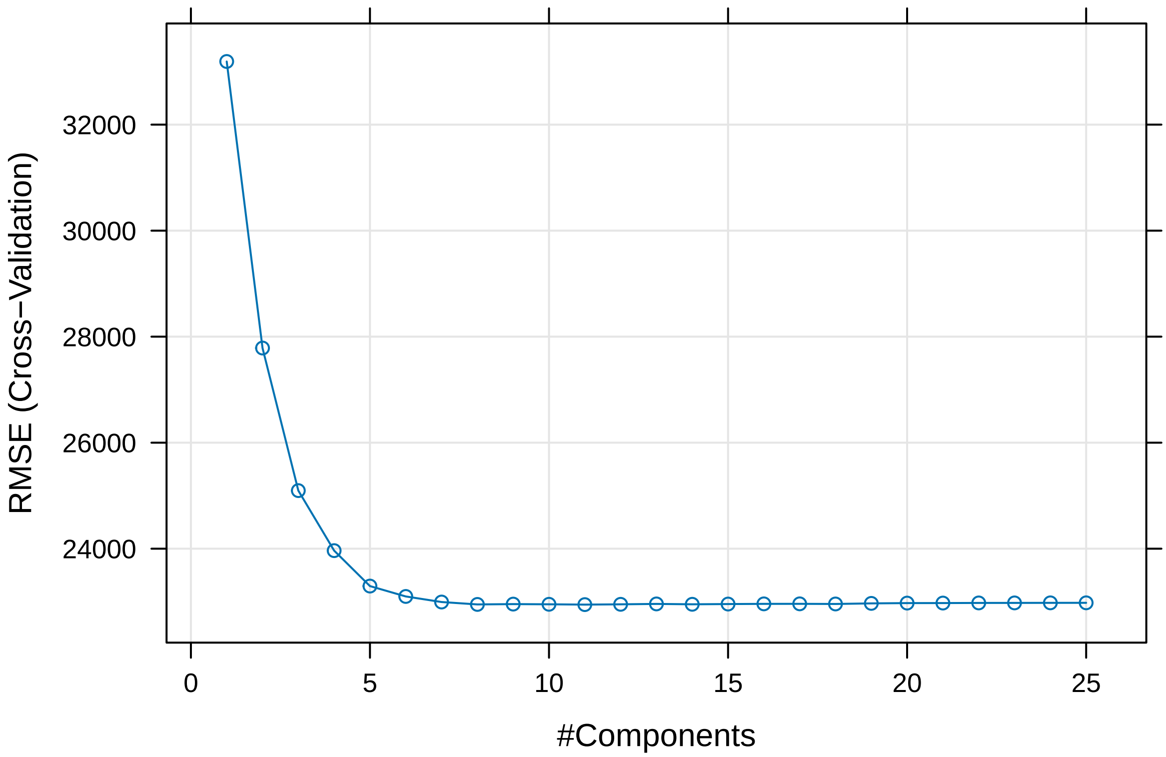
PLS model test error

```r
pred_pls <- predict(pls.fit, newdata = testing)
mean((testing$Sale_Price - pred_pls)^2) |> sqrt() #RMSE
```

```
## [1] 21243.27
```

PLS number of components

```r
plot(pls.fit)
```




```r
pls.fit$bestTune
```

```
##    ncomp
## 11    11
```

There are 11 components in the optimal PLS model.

d. Model selection

We can select the model based on the testing error. Since the models were trained on the training data only, choosing the model with lowest test error will give a less biased sense of which model will perform best on out-of-sample data, and is a good safeguard against overfit models

Testing error for each model

```r
# Lasso
mean((lasso.pred - testing$Sale_Price)^2) |> sqrt()
```

```
## [1] 20965.89
```

```r
# Lasso 1SE
mean((predict(lasso.1se,newdata = testing) - testing$Sale_Price)^2) |> sqrt()
```

```
## [1] 20492.55
```

```r
# Elastic Net
mean((enet.pred - testing$Sale_Price)^2) |> sqrt()
```

```
## [1] 20943.16
```

```r
# PLS
mean((pred_pls - testing$Sale_Price)^2) |> sqrt()
```

```
## [1] 21243.27
```

It looks like the Lasso 1SE model has the lowest testing error, so it is the one I would select. This is interesting because, by definition, the 1SE model does not have the lowest possible training error. However because it uses fewer predictors, it is a simpler model, and this simplicity may serve it well when encountering out of sample data. It's possible that our models were somewhat overfit on the training data, and that a simpler model may perform better.

e. Lasso using glmnet

(e) If R package "caret" was used for the lasso in (a), retrain this model using R package "glmnet", and vice versa. Compare the selected tuning parameters between the two software approaches. Should there be discrepancies in the chosen parameters, discuss potential reasons for these differences.

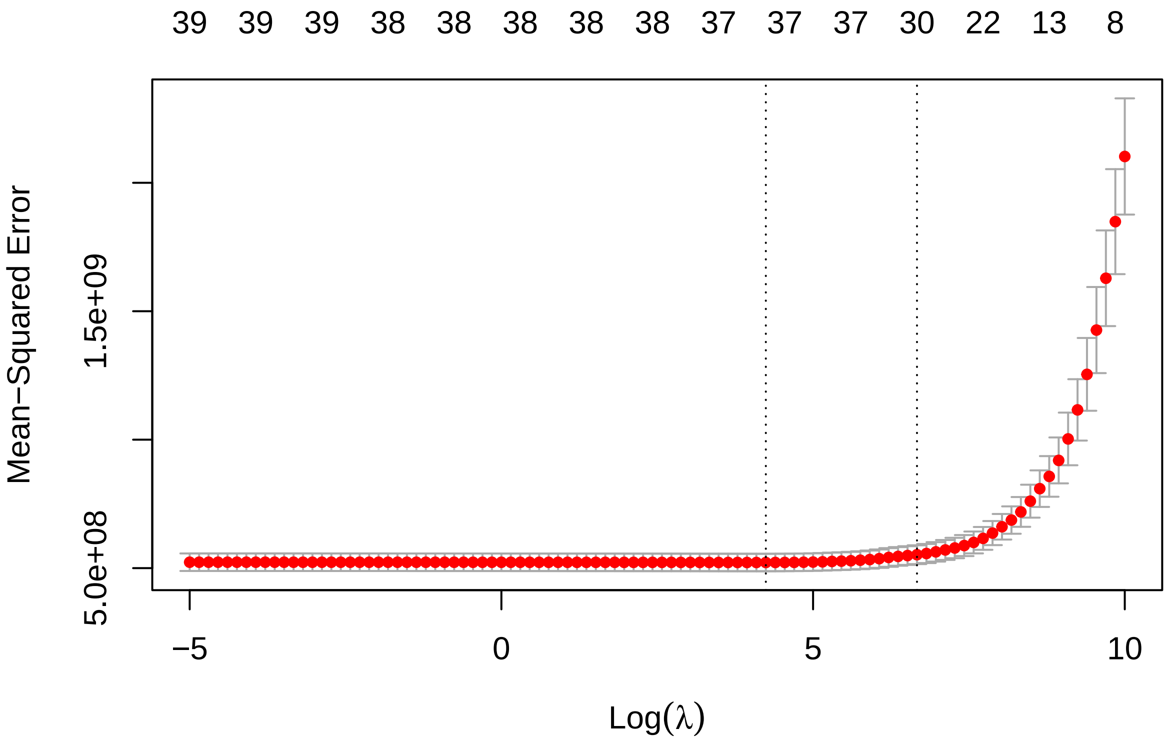
Fitting lasso using glmnet

```r
x <- model.matrix(Sale_Price ~ ., training)[,-1]
y <- training$Sale_Price

set.seed(2025)

lasso.glmnet <- cv.glmnet(x, y,
                          alpha = 1,
                          lambda = exp(seq(10, -5, length = 100)))

plot(lasso.glmnet)
```




```
lasso.glmnet$lambda.min
```

```
## [1] 69.57632
```

```
lasso.glmnet$lambda.1se
```

```
## [1] 785.772
```

```
# compare to the values from caret
lasso.fit$bestTune
```

```
##    alpha   lambda
## 62     1 69.57632
```

```
lasso.1se$bestTune
```

```
##    alpha   lambda
## 77     1 675.2963
```

The minimum lambda value using glmnet and caret was exactly the same. The 1SE lambda value was slightly different (~785 for glmnet and ~675 for caret), but a plausible explanation for this may caret choosing the first value below the 1SE cutoff and glmnet choosing the first value above above the 1SE cutoff, rather than a more substantial estimation difference between methods.