HW1

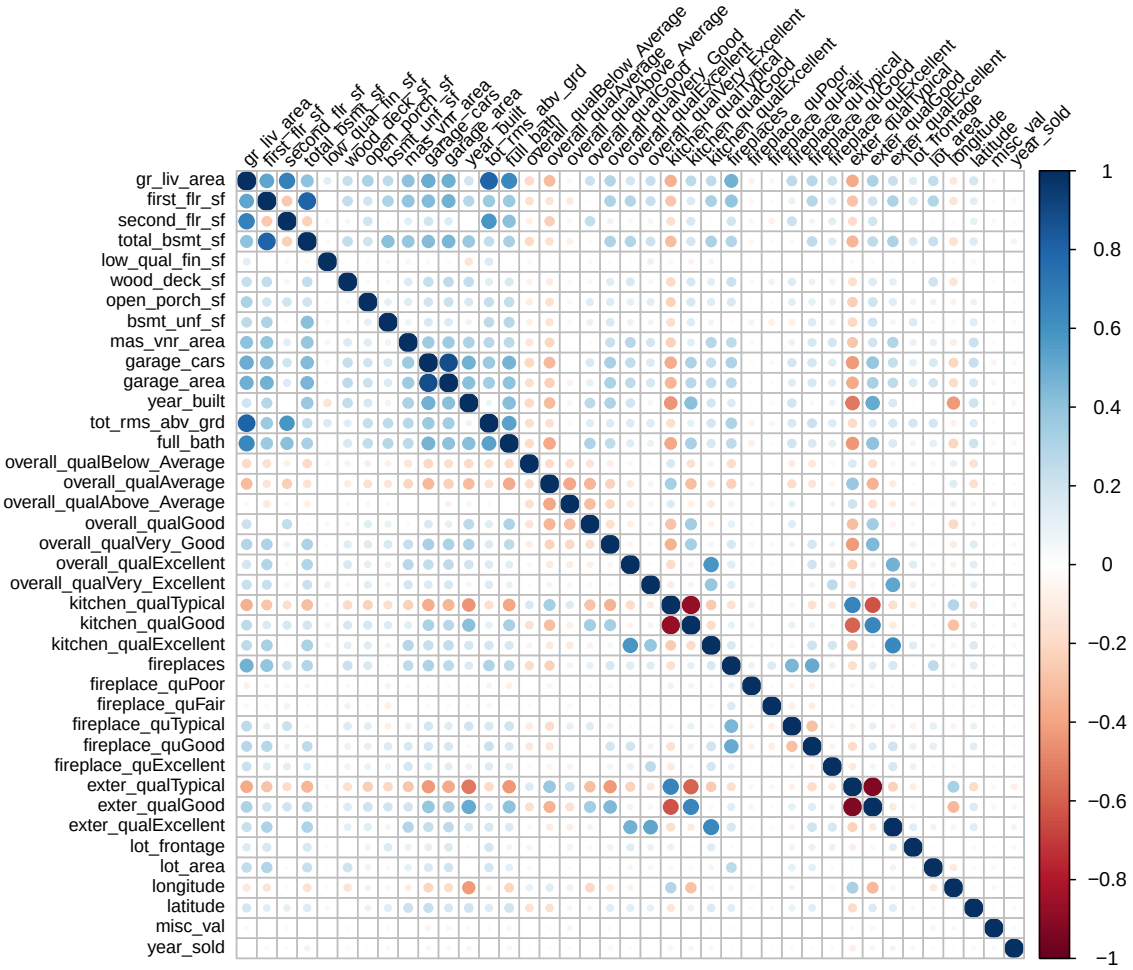
Zakari L. Billo

2026-02-25

In this exercise, we predict the sale price of a house based on various characteristics.

Data initialization

```
housing_test <- read.csv("data/housing_test.csv",  na = c("NA", ".", "")) |>
  janitor::clean_names()

housing_training <- read.csv("data/housing_training.csv",  na = c("NA", ".", "")) |>
  janitor::clean_names()

qual_levels <- c("No_Fireplace", "Poor", "Fair", "Typical", "Below_Average",
                 "Average", "Above_Average", "Good", "Very_Good",
                 "Excellent", "Very_Excellent")

# Code variables correctly
housing_training <- housing_training |>
  mutate(across(c(overall_qual, kitchen_qual, fireplace_qu, exter_qual),
                ~ factor(.x, levels = intersect(qual_levels, unique(.x)))))

housing_test <- housing_test |>
  mutate(
    overall_qual = factor(overall_qual, levels = levels(housing_training$overall_qual)),
    kitchen_qual = factor(kitchen_qual, levels = levels(housing_training$kitchen_qual)),
    fireplace_qu = factor(fireplace_qu, levels = levels(housing_training$fireplace_qu)),
    exter_qual   = factor(exter_qual,   levels = levels(housing_training$exter_qual))
  )

housing_training <- housing_training |> drop_na()
housing_test <- housing_test |> drop_na()

# matrix of predictors (glmnet uses input matrix)
x <- model.matrix(sale_price ~ ., housing_training)[,-1]
# vector of response
y <- housing_training[, "sale_price"]

corrplot(cor(x), method = "circle", tl.cex = 0.75, type = "full",
         tl.col = "black",
         tl.srt = 45)
```



Problems

(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?

```
set.seed(2)
cv.lasso <- cv.glmnet(x, y,
                      alpha = 1, # alpha = 1 enforces the necessary L1 penalty for Lasso
                      lambda = exp(seq(10, -2, length = 100)))

mat.coef <- coef(cv.lasso)

cv.lasso$lambda.min # we know the min lambda (L1 penalty) is 45.52137
```

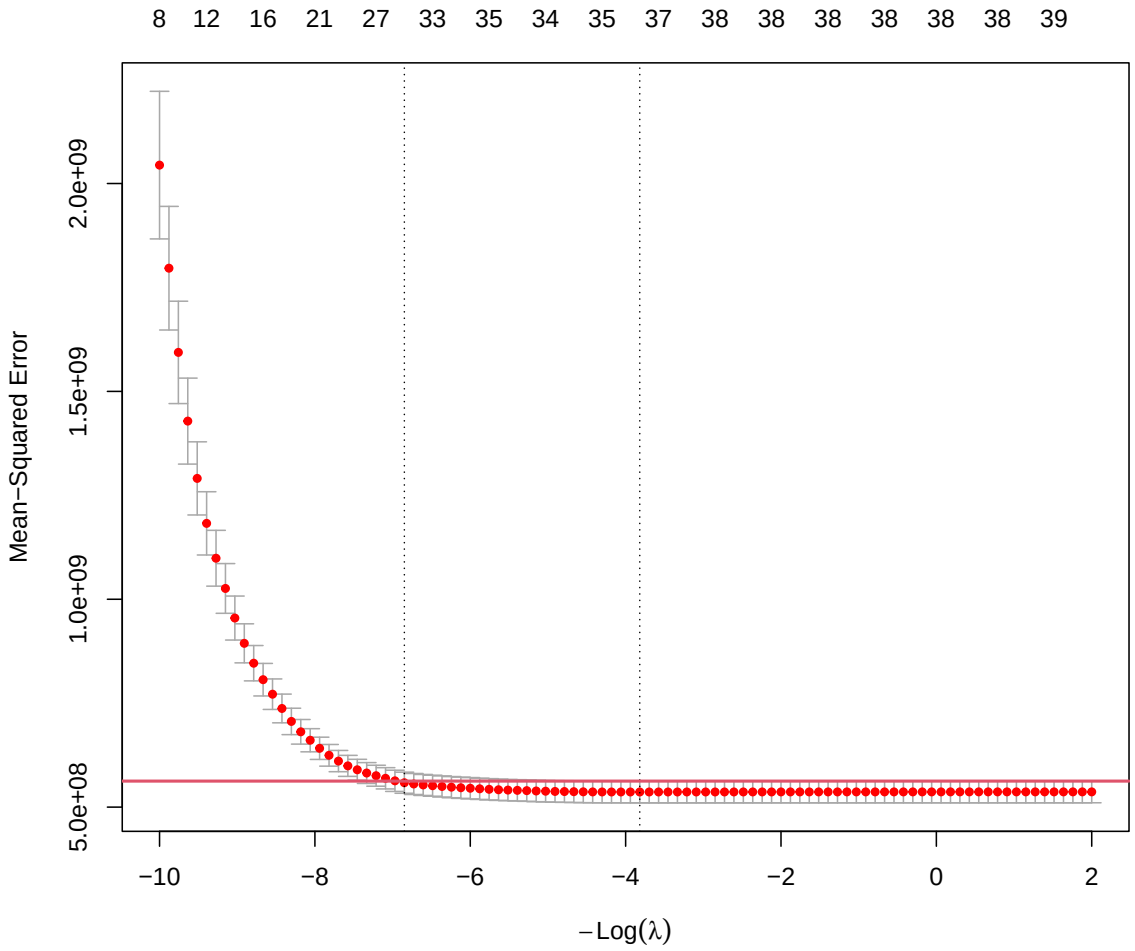
```
## [1] 45.52137
```

```
cv.lasso$lambda.1se # we know lambda (L1 penalty) with the 1se rule is 942.4519
```

```
## [1] 942.4519
```

Plot of cross-validated λ

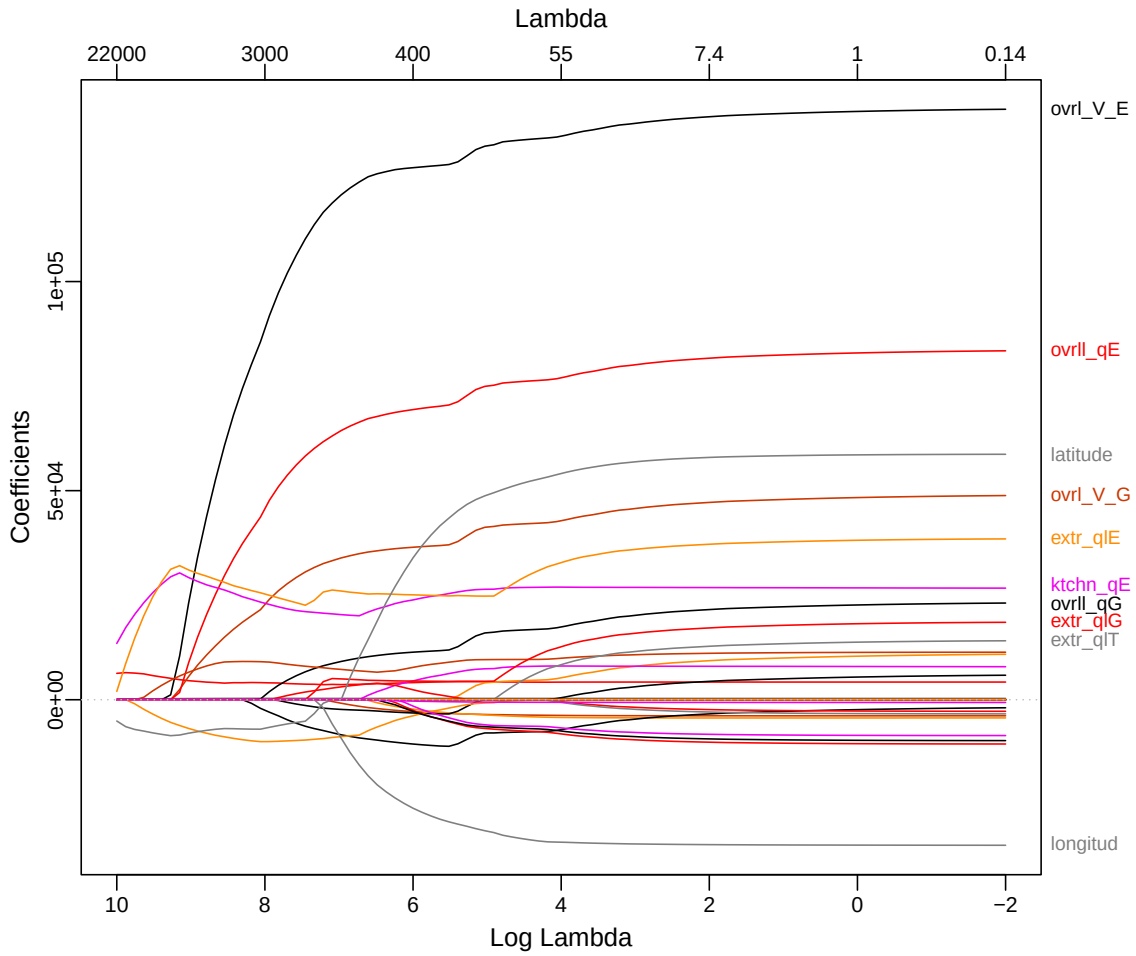
```
plot(cv.lasso)
abline(h = (cv.lasso$cvm + cv.lasso$cvsd)[which.min(cv.lasso$cvm)], col = 2, lwd = 2)
```



The lambda.min (L1 penalty) is 45.52137 and lambda.1se (L1 penalty) is 942.4519.

Now we examine the trace plots

```
plot_glmnet(cv.lasso$glmnet.fit)
```



Let's check how many coefficients end up in the final CV training model when we apply min CV lambda.min vs. CV lambda.1se

```
predict(cv.lasso, s = cv.lasso$lambda.min, type = "coefficients") # 37/39 predictors are in the model
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                    s=45.52137
## (Intercept)                    -4.881453e+06
## gr_liv_area                     6.625807e+01
## first_flr_sf                    5.226550e-01
## second_flr_sf                   .
## total_bsmt_sf                   3.564367e+01
## low_qual_fin_sf                -4.013031e+01
## wood_deck_sf                    1.178783e+01
## open_porch_sf                   1.562222e+01
## bsmt_unf_sf                    -2.081785e+01
## mas_vnr_area                    1.051878e+01
## garage_cars                     4.222702e+03
## garage_area                     8.001250e+00
## year_built                      3.287144e+02
## tot_rms_abv_grd                -3.767205e+03
## full_bath                      -4.262653e+03
## overall_qualBelow_Average      -6.562236e+03
## overall_qualAverage             1.099296e+03
```

```
## overall_qualAbove_Average   5.886887e+03
## overall_qualGood            1.805733e+04
## overall_qualVery_Good       4.349956e+04
## overall_qualExcellent       7.773687e+04
## overall_qualVery_Excellent  1.355001e+05
## kitchen_qualTypical         .
## kitchen_qualGood            8.056440e+03
## kitchen_qualExcellent       2.690608e+04
## fireplaces                  1.015894e+04
## fireplace_quPoor           -7.030393e+03
## fireplace_quFair           -8.560844e+03
## fireplace_quTypical        -7.852129e+03
## fireplace_quGood           -8.341184e+02
## fireplace_quExcellent      -1.079194e+03
## exter_qualTypical           9.308313e+03
## exter_qualGood              1.361232e+04
## exter_qualExcellent         3.371647e+04
## lot_frontage                9.928260e+01
## lot_area                    6.031813e-01
## longitude                  -3.416527e+04
## latitude                    5.484054e+04
## misc_val                    8.189010e-01
## year_sold                  -6.202472e+02
```

```
predict(cv.lasso, s = cv.lasso$lambda.1se, type = "coefficients") # 28/39 predictors are in the model
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                s=942.4519
## (Intercept)                -2.012424e+06
## gr_liv_area                 5.688465e+01
## first_flr_sf                6.542043e-01
## second_flr_sf               .
## total_bsmt_sf               3.613203e+01
## low_qual_fin_sf            -2.253097e+01
## wood_deck_sf                9.421370e+00
## open_porch_sf               8.082841e+00
## bsmt_unf_sf                -1.776993e+01
## mas_vnr_area                1.295584e+01
## garage_cars                 3.671365e+03
## garage_area                 1.156070e+01
## year_built                  3.251195e+02
## tot_rms_abv_grd            -1.225085e+03
## full_bath                   .
## overall_qualBelow_Average  -8.779707e+03
## overall_qualAverage        -2.414087e+03
## overall_qualAbove_Average   .
## overall_qualGood            9.592465e+03
## overall_qualVery_Good       3.444601e+04
## overall_qualExcellent       6.545374e+04
## overall_qualVery_Excellent  1.223989e+05
## kitchen_qualTypical        -8.616441e+03
## kitchen_qualGood            .
## kitchen_qualExcellent       2.020010e+04
## fireplaces                  7.056383e+03
## fireplace_quPoor            .
```

```
## fireplace_quFair            .
## fireplace_quTypical         .
## fireplace_quGood            3.342065e+03
## fireplace_quExcellent       .
## exter_qualTypical           .
## exter_qualGood              4.883579e+03
## exter_qualExcellent         2.571473e+04
## lot_frontage                6.105466e+01
## lot_area                    5.309617e-01
## longitude                  -1.264294e+04
## latitude                    5.386502e+03
## misc_val                    .
## year_sold                   .
```

We'll go with the 1se rule, since it gives the most regularized model such that error is within one standard error of the minimum

```
# Predictions for the test set
preds_1se <- predict(cv.lasso,
                     newx = model.matrix(sale_price ~ ., housing_test)[,-1],
                     s = "lambda.1se")

# Calculate Test MSE
test_error <- mean((housing_test$sale_price - preds_1se)^2)

test_error # Test MSE is 437332020
```

```
## [1] 437332020
```

The selected tuning parameter (λ) is 942.4519. Using this λ, the model includes 28 predictors and results in a Test Mean Squared Error (MSE) of 437,332,020 (which corresponds to an average error of roughly $20,912 per house, the RMSE).

(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.

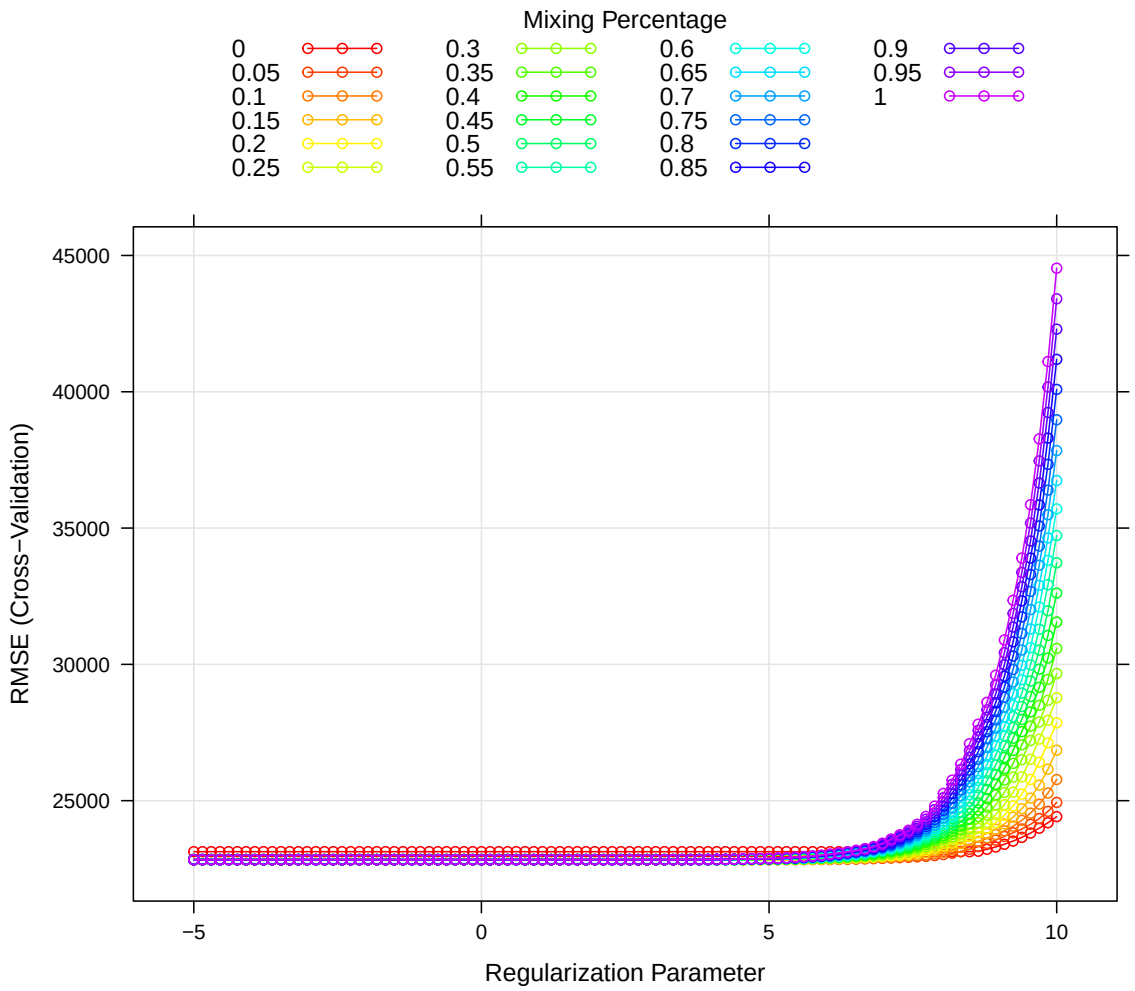
```
set.seed(2)

ctrl1_1se <- trainControl(method = "cv", number = 10, selectionFunction = "oneSE")

# Train using x and y
enet.fit <- train(x, y,
                  method = "glmnet",
                  tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                         lambda = exp(seq(10, -5, length = 100))),
                  trControl = ctrl1_1se) # Allow 1SE optimal parameters

enet.fit$bestTune
```

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

Elastic net has determined that our optimal hyperparameters are $\alpha = 0.0$ and $\lambda = 8874.25$.

Let's look at a rainbow plot of the hyperparameters

```
myCol <- rainbow(25)
myPar <- list(superpose.symbol = list(col = myCol),
superpose.line = list(col = myCol))
plot(enet.fit, par.settings = myPar, xTrans = log)
```




```
# Coefficients in the final model
coef(enet.fit$finalModel, enet.fit$bestTune$lambda) # 39/39 coeffs remain
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                 s=8874.25
## (Intercept)                -6.468012e+06
## gr_liv_area                 3.019283e+01
## first_flr_sf                2.493289e+01
## second_flr_sf               2.050251e+01
## total_bsmt_sf               3.041405e+01
## low_qual_fin_sf            -1.494554e+01
## wood_deck_sf                1.784955e+01
## open_porch_sf               2.610057e+01
## bsmt_unf_sf                -1.654368e+01
## mas_vnr_area                1.868583e+01
## garage_cars                 4.546105e+03
## garage_area                 1.531910e+01
## year_built                  2.759514e+02
```

```
## tot_rms_abv_grd            -1.207843e+03
## full_bath                  -1.234230e+03
## overall_qualBelow_Average  -1.484664e+04
## overall_qualAverage        -7.898273e+03
## overall_qualAbove_Average  -4.179601e+03
## overall_qualGood            6.017506e+03
## overall_qualVery_Good       2.781750e+04
## overall_qualExcellent       5.200755e+04
## overall_qualVery_Excellent  1.045129e+05
## kitchen_qualTypical        -5.197707e+03
## kitchen_qualGood            4.202988e+03
## kitchen_qualExcellent       2.731672e+04
## fireplaces                  9.083698e+03
## fireplace_quPoor           -4.620594e+03
## fireplace_quFair           -5.139362e+03
## fireplace_quTypical        -2.448621e+03
## fireplace_quGood            3.122815e+03
## fireplace_quExcellent       7.945691e+03
## exter_qualTypical          -1.350917e+03
## exter_qualGood              5.586991e+03
## exter_qualExcellent         3.040613e+04
## lot_frontage                8.253550e+01
## lot_area                    5.624373e-01
## longitude                  -4.901383e+04
## latitude                    6.109584e+04
## misc_val                    7.226971e-01
## year_sold                  -5.920663e+02
```

Find Test MSE

```
# Calculate predictions the caret way
preds <- predict(enet.fit, newdata = model.matrix(sale_price ~ ., housing_test)[,-1])

# Calculate Test MSE
test_error <- mean((housing_test$sale_price - preds)^2)
test_error
```

```
## [1] 450345913
```

The selected tuning parameters are $\alpha = 0$ and $\lambda = 8874.25$. Using these parameters λ, the model includes 38 predictors and results in a Test Mean Squared Error (MSE) of 450,345,913 (which corresponds to an average error of roughly \$21,221 per house, the RMSE). The 1SE rule is applicable because "caret" allows us to apply the 1SE rule across a 2D grid of α and λ, we just need to tell trainControl to use it by adding selectionFunction = "oneSE".

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

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

set.seed(2)
pls_fit <- train(x, y,
                 method = "pls",
                 tuneGrid = data.frame(ncomp = 1:39),
                 trControl = ctrl1,
                 preProcess = c("center", "scale"))
```

```
pls_fit # 23 components were selected
```

```
## Partial Least Squares
##
## 1440 samples
##   39 predictor
##
## Pre-processing: centered (39), scaled (39)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1296, 1296, 1296, 1295, 1297, 1296, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE          Rsquared   MAE
##    1     2.996777e+04  0.8382349  2.202904e+04
##    2     2.670543e+04  0.8689856  1.932573e+04
##    3     2.503199e+04  0.8856103  1.830569e+04
##    4     2.356104e+04  0.8984314  1.726588e+04
##    5     2.314876e+04  0.9014744  1.692356e+04
##    6     2.295818e+04  0.9029274  1.677086e+04
##    7     2.296371e+04  0.9028302  1.676587e+04
##    8     2.296100e+04  0.9028332  1.682354e+04
##    9     2.294563e+04  0.9028042  1.683196e+04
##   10     2.290020e+04  0.9031629  1.679514e+04
##   11     2.290049e+04  0.9030938  1.677314e+04
##   12     2.290532e+04  0.9032031  1.675413e+04
##   13     2.286561e+04  0.9034521  1.673552e+04
##   14     2.285461e+04  0.9035090  1.669879e+04
##   15     2.286753e+04  0.9033927  1.668131e+04
##   16     2.284580e+04  0.9035777  1.664807e+04
##   17     2.284341e+04  0.9036062  1.664892e+04
##   18     2.284349e+04  0.9035898  1.665496e+04
##   19     2.284185e+04  0.9036304  1.664608e+04
##   20     2.283926e+04  0.9036576  1.664055e+04
##   21     2.283877e+04  0.9036548  1.664444e+04
##   22     2.284156e+04  0.9036318  1.664310e+04
##   23     2.283748e+04  0.9036695  1.663961e+04
##   24     2.284151e+04  0.9036377  1.664175e+04
##   25     2.284085e+04  0.9036366  1.664255e+04
##   26     2.283858e+04  0.9036521  1.664144e+04
##   27     2.283896e+04  0.9036506  1.664166e+04
##   28     2.283968e+04  0.9036464  1.664133e+04
##   29     2.283877e+04  0.9036544  1.664041e+04
##   30     2.283897e+04  0.9036528  1.664001e+04
##   31     2.283923e+04  0.9036513  1.664010e+04
##   32     2.283930e+04  0.9036502  1.664002e+04
##   33     2.283927e+04  0.9036506  1.663999e+04
##   34     2.283927e+04  0.9036505  1.663999e+04
##   35     2.283927e+04  0.9036505  1.663999e+04
##   36     2.283927e+04  0.9036505  1.663999e+04
##   37     2.283927e+04  0.9036505  1.663999e+04
##   38     2.283927e+04  0.9036505  1.663999e+04
##   39     1.086824e+19  0.5296533  8.282142e+18
##
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final value used for the model was ncomp = 23.
```

```
pred_pls <- predict(pls_fit, newdata = model.matrix(sale_price ~ ., housing_test)[,-1])

test_error_pls <- mean((housing_test$sale_price - pred_pls)^2)

test_error_pls # Test MSE is 447100921
```

```
## [1] 447100921
```

Partial least squares regression selected 23 components in the model includes and results in a Test Mean Squared Error (MSE) of 447,100,921 (which corresponds to an average error of roughly $21,145 per house, the RMSE).

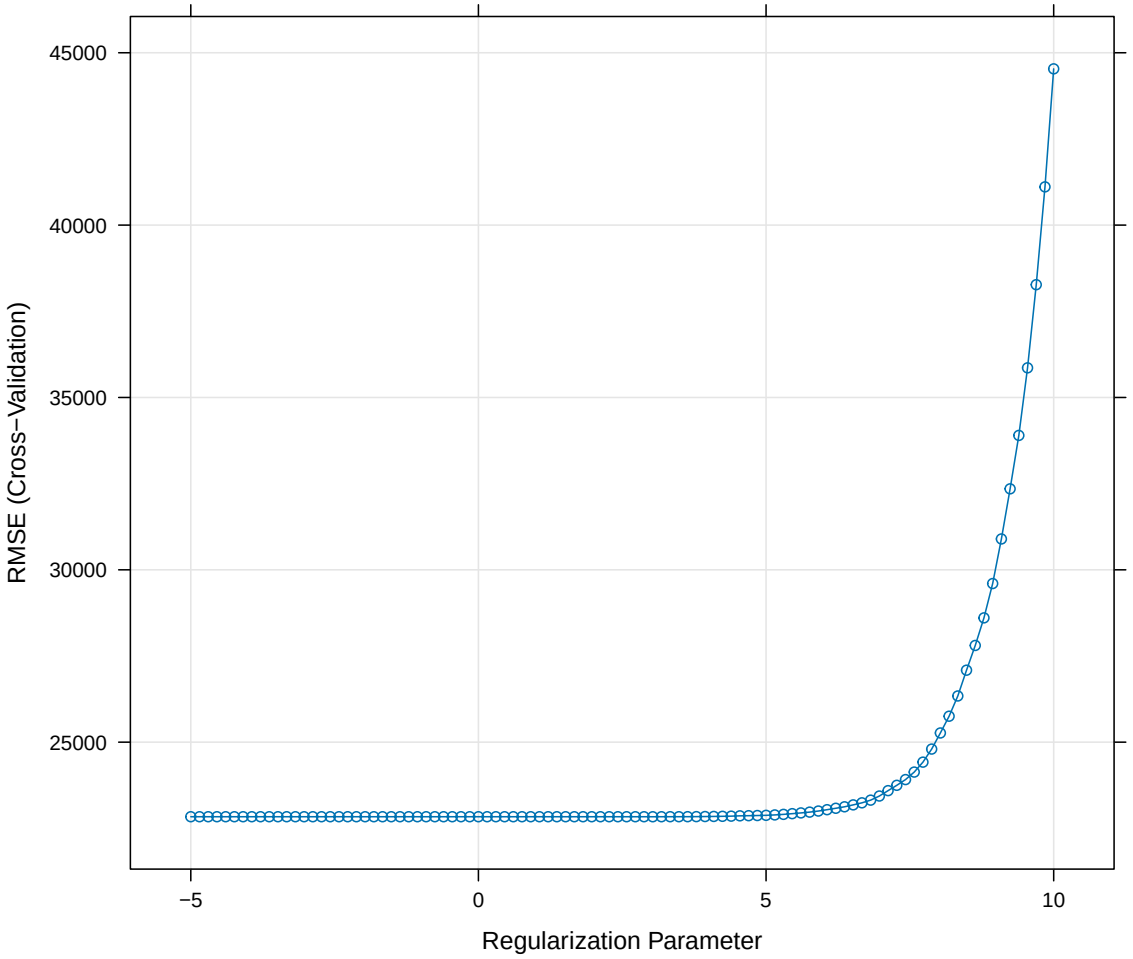
(d) Choose the best model for predicting the response and explain your choice.

The best model (between Lasso, Elastic net, and PLS) was Lasso, as it had the lowest Test MSE of 437,332,020.

(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.

I will retrain with "caret" instead of the previously used "glmnet"

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

set.seed(2)

lasso.fit <- train(sale_price ~ . ,data = housing_training,
                   method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1, # Alpha = 1 corresponds to L1 penalty (Lasso)
                   lambda = exp(seq(10, -5, length=100))),
trControl = ctrl1)

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




```
lasso.fit$bestTune
```

```
##    alpha   lambda
## 79     1 914.3211
```

The optimal $\lambda = 914.3211$ for $\alpha = 1$ (Lasso). Let's look at the Test MSE now

```
pred_caret <- predict(lasso.fit, newdata = housing_test)

mse_caret <- mean((housing_test$sale_price - pred_caret)^2)

mse_caret
```

```
## [1] 436922925
```

Lasso with "caret" has a Test MSE of 436,922,925 (which corresponds to an average error of roughly $20,903 per house, the RMSE). This is MSE is slightly lower than what resulted from the Lasso procedure with "glmnet". As noted in class, if a lambda sequence is not explicitly specified, glmnet and caret will use completely different default grids. Specifically, caret's implementation does specify lambda, meaning the default grid of lambda is different from the assigned lambda.grid. I explicitly passed the same lambda sequence to both functions to minimize this discrepancy. Also, when setting the same random seed (set.seed(2)) and forcing the exact same lambda grid, caret and glmnet use different internal algorithms to implement the random partition of the data into 10 folds. Because the observations inside each fold differ between the two packages, the Mean Squared Error calculated for each λ varies.