P8106_HW1

Naomi Simon-Kumar ns3782

2/16/2025

Contents

Loading libraries

```
library(ISLR)
library(glmnet)
library(caret)
library(tidymodels)
library(corrplot)
library(ggplot2)
library(plotmo)
library(ggrepel)
library(pls)
library(knitr)
```

Question (a): Lasso Model

To start, we load the training and testing data and subsequently set a seed for reproducibility.

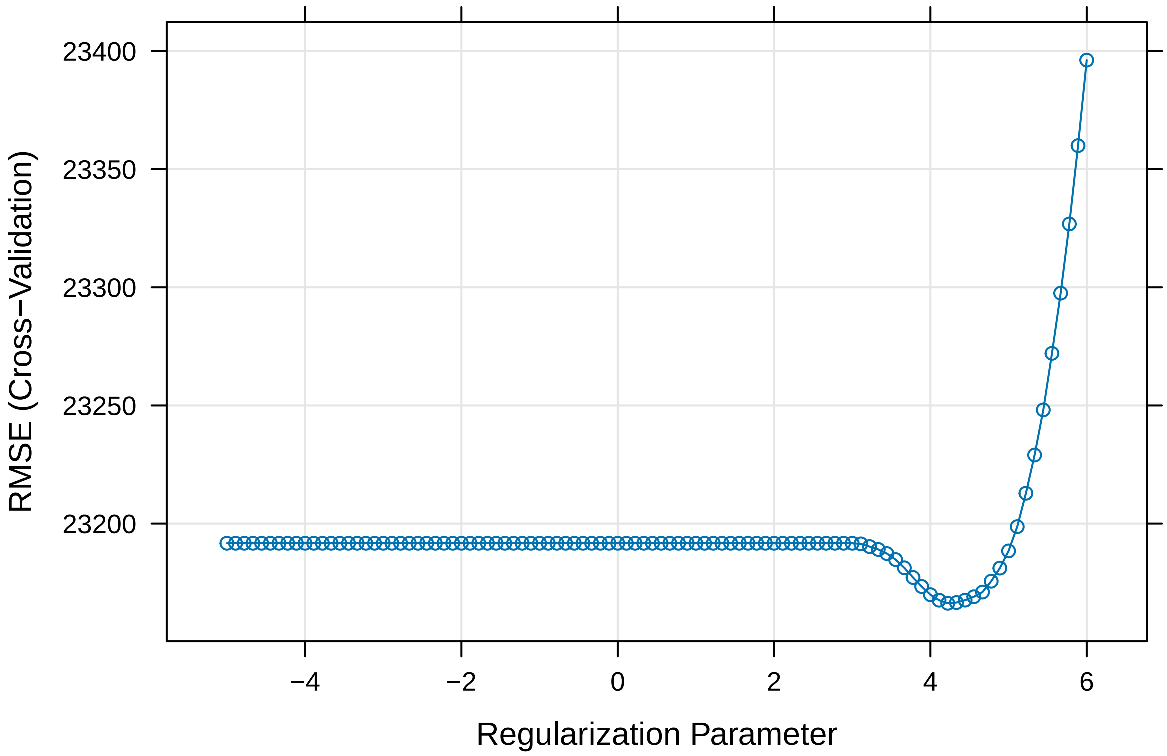
Next, we initialise 10-fold cross-validation to partition the training data into 10 equal subsets. This allows training the model on 9 folds while validating on the final fold. This ensures we evaluate the performance of the model, while avoiding overfitting.

```
# Load training and testing data

training_data <- read.csv("housing_training.csv")
testing_data <- read.csv("housing_test.csv")
```

```r
set.seed(29)  # Ensure results are reproducible

# Using 10 fold cross-validation

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

Next, we proceed to fit a lasso regression model using the training data. Sale_Price is the outcome variable, with all other variables as predictors. The lasso model is tuned over a sequence of 100 lambda values ranging from exp(6) to exp(-5).

```r
set.seed(29)  # Ensure results are reproducible

# Fit the Lasso model

lasso.fit  <- train(
  Sale_Price ~ .,
  data = training_data,
  method = "glmnet",
  tuneGrid = expand.grid(alpha = 1,
                         lambda = exp(seq(6, -5, length = 100))),
  trControl = ctrl1
)

# Plot

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



Based on the plot, it appears as though the optimal lambda value is around exp(4), as this is where the RMSE is minimised. Higher lambda values (i.e., greater penalisation) appear to result in poorer model performance, likely due to excessive shrinkage forcing too many coefficients to zero, leading to underfitting.

```
set.seed(29)  # Ensure results are reproducible

# Find optimal tuning parameter

lasso.fit$bestTune
```

```
##    alpha   lambda
## 84     1 68.18484
```

```
# Extracting coefficients for each predictor, at the optimal lambda

coef(lasso.fit$finalModel, lasso.fit$bestTune$lambda)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                  s1
## (Intercept)            -4.820791e+06
## Gr_Liv_Area             6.534680e+01
## First_Flr_SF            8.043483e-01
## Second_Flr_SF           .
## Total_Bsmt_SF           3.542591e+01
## Low_Qual_Fin_SF        -4.089879e+01
```

```
## Wood_Deck_SF                 1.161853e+01
## Open_Porch_SF                1.539927e+01
## Bsmt_Unf_SF                 -2.088675e+01
## Mas_Vnr_Area                 1.091770e+01
## Garage_Cars                  4.078354e+03
## Garage_Area                  8.182394e+00
## Year_Built                   3.232484e+02
## TotRms_AbvGrd               -3.607362e+03
## Full_Bath                   -3.820746e+03
## Overall_QualAverage         -4.845814e+03
## Overall_QualBelow_Average   -1.244202e+04
## Overall_QualExcellent        7.559703e+04
## Overall_QualFair            -1.073410e+04
## Overall_QualGood             1.211373e+04
## Overall_QualVery_Excellent   1.358907e+05
## Overall_QualVery_Good        3.788544e+04
## Kitchen_QualFair            -2.476713e+04
## Kitchen_QualGood            -1.713660e+04
## Kitchen_QualTypical         -2.525278e+04
## Fireplaces                   1.051146e+04
## Fireplace_QuFair            -7.657866e+03
## Fireplace_QuGood             .
## Fireplace_QuNo_Fireplace     1.385656e+03
## Fireplace_QuPoor            -5.632703e+03
## Fireplace_QuTypical         -7.010013e+03
## Exter_QualFair              -3.316061e+04
## Exter_QualGood              -1.492745e+04
## Exter_QualTypical           -1.936658e+04
## Lot_Frontage                 9.952901e+01
## Lot_Area                     6.042265e-01
## Longitude                   -3.285809e+04
## Latitude                     5.492284e+04
## Misc_Val                     8.240622e-01
## Year_Sold                   -5.568142e+02
```

Note that at the optimal lambda value, most of the predictors remain in the model. However, some are shrunk to zero (i.e., Second_Flr_SF, Fireplace_QuGood) during the variable selection process, and removed from the model. Therefore, this final model includes **37 predictors**.

```r
set.seed(29)  # Ensure results are reproducible

# Finding RMSE

lasso_preds <- predict(lasso.fit, newdata = testing_data)

lasso_rmse <- sqrt(mean((lasso_preds - testing_data$Sale_Price)^2))

print(lasso_rmse)
```

```
## [1] 20969.2
```

For the lasso model, the optimal tuning parameter lambda is **68.18484**, representing where RMSE is minimised. The test error (RMSE) at this lambda is **20969.2**.

```r
set.seed(29)  # Ensure results are reproducible

# Using 1se cross-validation.
# Code from: https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/oneSE

ctrl_1se <- trainControl(
  method = "cv",
  selectionFunction = "oneSE"
)

# Fit the lasso model using 1se

lasso_1se_fit <- train(
  Sale_Price ~ .,
  data = training_data,
  method = "glmnet",
  tuneGrid = expand.grid(
    alpha = 1,
    lambda = exp(seq(6, -5, length = 100))
  ),
  trControl = ctrl_1se
)

# Optimal lambda using 1SE

lasso_lambda_1se <- lasso_1se_fit$bestTune$lambda
print(lasso_lambda_1se)
```

```
## [1] 403.4288
```

```r
# Extracting coefficients for each predictor, at the optimal lambda

coef(lasso_1se_fit$finalModel, s = lasso_lambda_1se)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)               -3.919159e+06
## Gr_Liv_Area                6.099153e+01
## First_Flr_SF               9.477449e-01
## Second_Flr_SF              .
## Total_Bsmt_SF              3.627699e+01
## Low_Qual_Fin_SF           -3.523480e+01
## Wood_Deck_SF               1.000632e+01
## Open_Porch_SF              1.203918e+01
## Bsmt_Unf_SF               -2.059528e+01
## Mas_Vnr_Area               1.297320e+01
## Garage_Cars                3.491107e+03
## Garage_Area                9.740129e+00
## Year_Built                 3.150805e+02
## TotRms_AbvGrd             -2.518326e+03
## Full_Bath                 -1.415647e+03
## Overall_QualAverage       -4.006722e+03
## Overall_QualBelow_Average  -1.084480e+04
```

```
## Overall_QualExcellent       8.719850e+04
## Overall_QualFair           -8.763236e+03
## Overall_QualGood            1.111389e+04
## Overall_QualVery_Excellent  1.559964e+05
## Overall_QualVery_Good       3.730815e+04
## Kitchen_QualFair           -1.420320e+04
## Kitchen_QualGood           -7.639239e+03
## Kitchen_QualTypical        -1.644274e+04
## Fireplaces                  8.190689e+03
## Fireplace_QuFair           -3.809974e+03
## Fireplace_QuGood            2.196176e+03
## Fireplace_QuNo_Fireplace    .
## Fireplace_QuPoor           -1.484163e+03
## Fireplace_QuTypical        -4.125304e+03
## Exter_QualFair             -1.695505e+04
## Exter_QualGood              .
## Exter_QualTypical          -4.790664e+03
## Lot_Frontage                8.663344e+01
## Lot_Area                    5.915806e-01
## Longitude                  -2.246220e+04
## Latitude                    3.767830e+04
## Misc_Val                    3.093854e-01
## Year_Sold                  -1.654627e+02
```

```r
# Lasso 1SE RMSE

lasso_1SE_preds <- predict(lasso_1se_fit, newdata = testing_data)

lasso_1SE_rmse <- sqrt(mean((lasso_1SE_preds - testing_data$Sale_Price)^2))

print(lasso_1SE_rmse)
```

```
## [1] 20511.62
```

Using the 1SE rule, the optimal lambda is **403.4288**. During the variable selection process, variables Second_Flr_SF, Fireplace_QuNo_Fireplace, and Exter_QualGood are removed from the model. When the 1SE rule is applied, there are **36 predictors** included in the model, which is 1 fewer than the original lasso model.

Question (b): Elastic Net

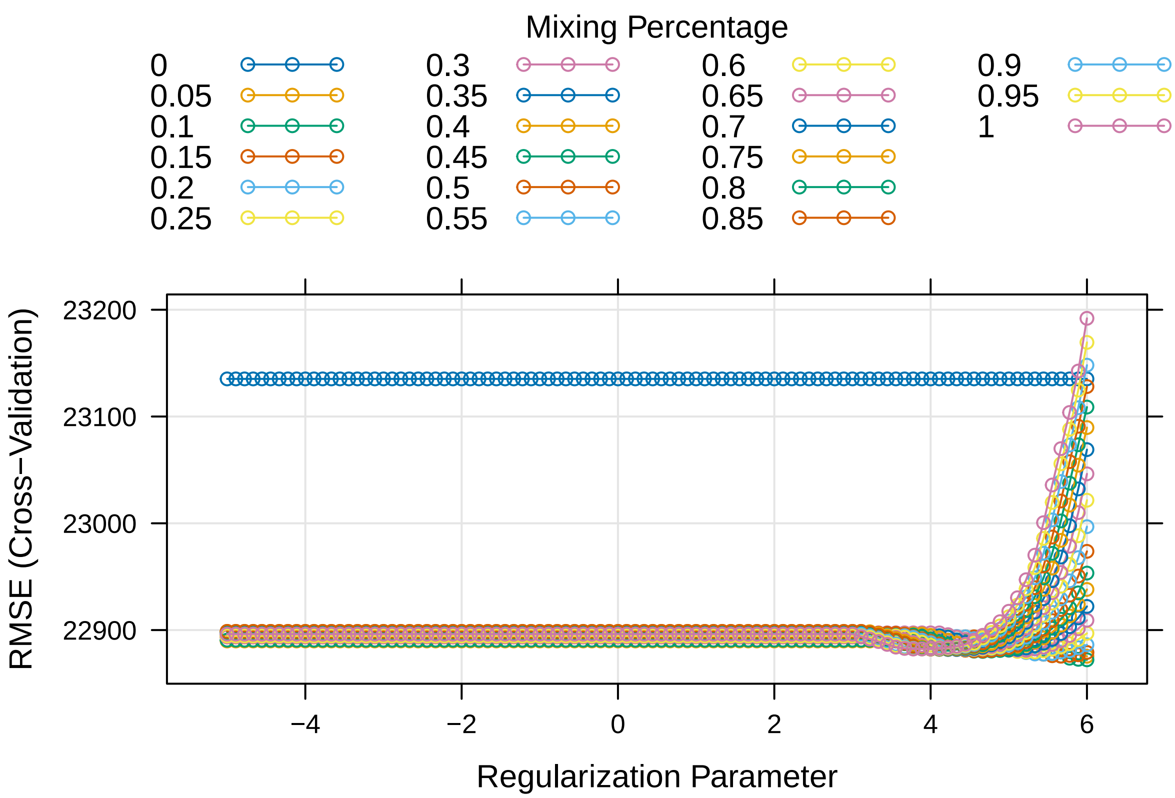
To fit the elastic net model, I began with a wide lambda range.

```r
# Set seed to ensure reproducibility

set.seed(16)

# Fit elastic net model
# Tuning the different lambda ranges

enet.fit <- train(Sale_Price ~ .,
                  data = training_data,
```

```r
                 method = "glmnet",
                 tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                        lambda = exp(seq(6, -5, length = 100))),
                 trControl = ctrl1)

# Results

print(enet.fit$bestTune)
```

```
##     alpha   lambda
## 300   0.1 403.4288
```

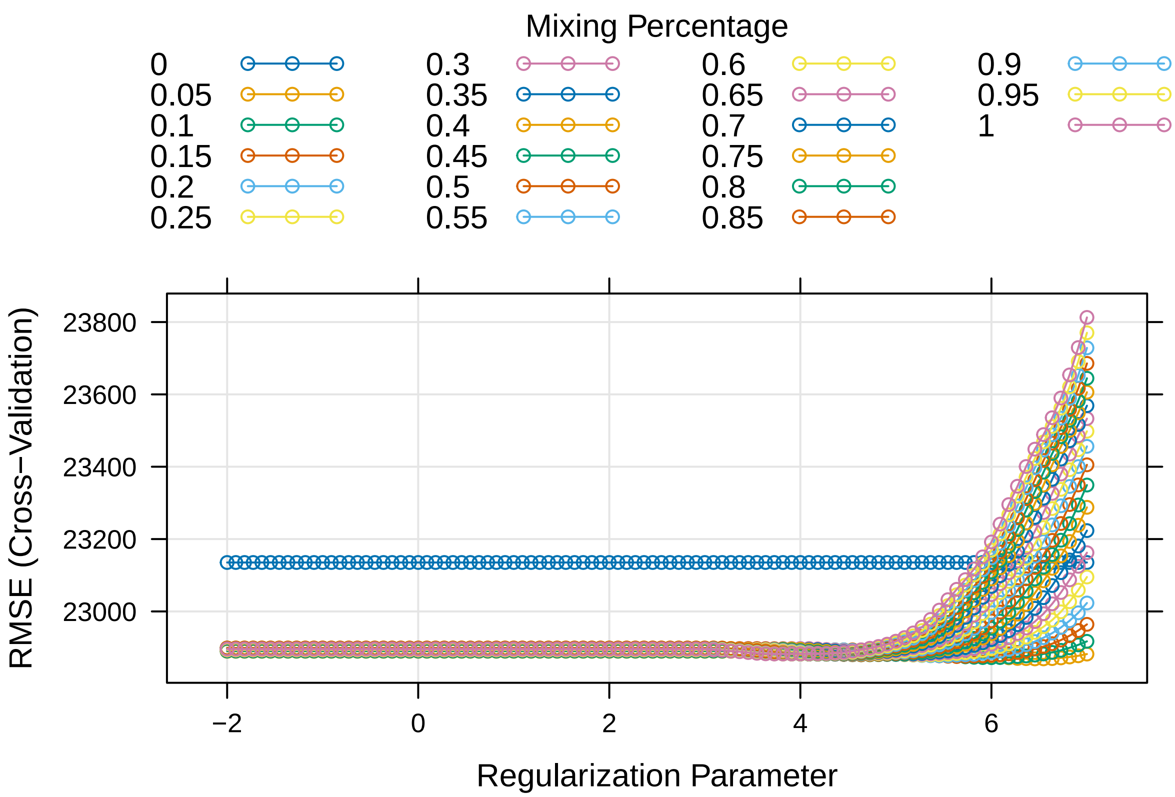
```r
# Cross validation plot

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



After reviewing the cross-validation plot, I refined the lambda range.

```r
# Set seed to ensure reproducibility

set.seed(16)

# Adjusting

enet.fit <- train(Sale_Price ~ .,
```

```r
                data = training_data,
                method = "glmnet",
                tuneGrid = expand.grid(alpha = seq(0, 1, length = 21),
                                       lambda = exp(seq(7, -2, length = 100))),
                trControl = ctrl1)

# Cross validation plot

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




```r
# Optimal lambda

print(enet.fit$bestTune)
```

```
##     alpha   lambda
## 194  0.05 635.5848
```

The cross validation plot shows the RMSE values were fairly stable at lower regularisation values, but increasing steeply when log(lambda)   6. Therefore, the selected tuning parameters are **alpha = 0.05** and **lambda = 635.5848**.

```r
# Set seed to ensure reproducibility

set.seed(16)
```

```r
# Predictions using testing dataset

enet.pred <- predict(enet.fit, newdata = testing_data)

# Test error

enet_test_mse <- mean((enet.pred - testing_data$Sale_Price)^2)

# Results

print(enet_test_mse)
```

```
## [1] 438041526
```

From this, the test error of the model is **438041526**.

```r
# Set seed to ensure reproducibility

set.seed(16)

# Applying 1SE rule to elastic net model

enet_1se_fit <- train(
  Sale_Price ~ .,
  data = training_data,
  method = "glmnet",
  tuneGrid = expand.grid(
    alpha = seq(0, 1, length = 21),
    lambda = exp(seq(7, -2, length = 100))
  ),
  trControl = ctrl_1se
)

enet_1se_fit$bestTune$lambda
```

```
## [1] 1096.633
```

```r
enet_1se_fit$bestTune$alpha
```

```
## [1] 0
```

Yes, it is possible to apply the 1SE rule to selecting tuning parameters for elastic net. The elastic net method includes penalties from both ridge regression and lasso (the mixing parameter alpha that determines the balance between ridge and lasso penalties, and the overall regularisation strength lambda). The 1SE rule is defined as the most regularised model such that error is within one standard error of the minimum.

Therefore, using the 1SE rule, it is possible to select the most regularised model (i.e., the largest lambda) for each alpha value that has error within one standard error of the minimum, then compare across different alpha values to give the effective regularisation via the ridge-type penalty and feature selection via the lasso penalty, as determined by cross-validation.

Based on our data, the 1SE rule model parameters are alpha = **0** and lambda = **1096.633**. Given that alpha = 0, this indicates ridge regression was the optimal model.

I proceeded to find the test error of the 1SE model.

```r
# Set seed to ensure reproducibility

set.seed(16)

# Predictions using testing dataset for 1SE model

enet_1se_pred <- predict(enet_1se_fit, newdata = testing_data)

# Test error

enet_1se_test_mse <- mean((enet_1se_pred - testing_data$Sale_Price)^2)

# Results

print(enet_1se_test_mse)
```

```
## [1] 426357707
```

The test error for the 1SE rule elastic net model is **426357707**.

Question (c): Partial least squares

I proceeded with fitting the partial least squares model.

```r
# Set seed for reproducibility

set.seed(29)

# Fitting the model

pls_mod <- plsr(Sale_Price ~ .,
                data = training_data,
                scale = TRUE,
                validation = "CV")

summary(pls_mod)
```

```
## Data:    X dimension: 1440 39
##  Y dimension: 1440 1
## Fit method: kernelpls
## Number of components considered: 39
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           73685    33422    27955    25141    24106    23504    23315
## adjCV        73685    33415    27918    25071    24032    23431    23248
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       23193    23167    23191     23193     23182     23178     23185
## adjCV    23131    23106    23126     23126     23115     23111     23117
```

```
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV        23187     23198     23201     23207     23211     23231     23237
## adjCV     23119     23129     23132     23138     23142     23160     23165
##        21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV        23239     23243     23243     23244     23244     23247     23250
## adjCV     23167     23171     23171     23171     23171     23174     23176
##        28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV        23250     23250     23250     23250     23250     23250     23250
## adjCV     23176     23177     23177     23177     23177     23177     23177
##        35 comps  36 comps  37 comps  38 comps  39 comps
## CV        23250     23250     23250     23250     23521
## adjCV     23177     23177     23177     23177     23373
##
## TRAINING: % variance explained
##             1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X             20.02    25.93    29.67    33.59    37.01    40.03    42.49
## Sale_Price    79.73    86.35    89.36    90.37    90.87    90.99    91.06
##             8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X             45.53    47.97     50.15     52.01     53.69     55.35     56.86
## Sale_Price    91.08    91.10     91.13     91.15     91.15     91.16     91.16
##             15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## X              58.64     60.01     62.18     63.87     65.26     67.10
## Sale_Price     91.16     91.16     91.16     91.16     91.16     91.16
##             21 comps  22 comps  23 comps  24 comps  25 comps  26 comps
## X              68.44     70.12     71.72     73.35     75.20     77.27
## Sale_Price     91.16     91.16     91.16     91.16     91.16     91.16
##             27 comps  28 comps  29 comps  30 comps  31 comps  32 comps
## X              78.97     80.10     81.83     83.55     84.39     86.34
## Sale_Price     91.16     91.16     91.16     91.16     91.16     91.16
##             33 comps  34 comps  35 comps  36 comps  37 comps  38 comps
## X              88.63     90.79     92.79     95.45     97.49    100.00
## Sale_Price     91.16     91.16     91.16     91.16     91.16     91.16
##             39 comps
## X             100.64
## Sale_Price     91.04
```

```r
# Determine the optimal number of components

cv_mse <- RMSEP(pls_mod)

ncomp_cv <- which.min(cv_mse$val[1,,]) - 1

# Optimal number of components

print(ncomp_cv)
```

```
## 8 comps
##       8
```

Based on the computation above, the optimal number of components is **8**.

```r
# Set seed for reproducibility
```

```r
set.seed(29)

# Calculate Test MSE

y2 <- testing_data$Sale_Price

predy2_pls <- predict(pls_mod, newdata = testing_data,
                      ncomp = ncomp_cv)

pls_test_mse <- mean((y2 - predy2_pls)^2)

print(pls_test_mse)
```

```
## [1] 440217938
```

The test error for this model is **440217938**.

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

```r
# Comparison table of models
# Code from: https://bookdown.org/yihui/rmarkdown-cookbook/kable.html

# Convert MSE to RMSE for comparison
enet_test_rmse <- sqrt(enet_test_mse)
pls_test_rmse <- sqrt(pls_test_mse)
enet_1se_test_rmse <- sqrt(enet_1se_test_mse)


comparison_table <- tibble(
  Model = c("Lasso", "Lasso 1SE", "Elastic Net", "Elastic Net 1SE", "Partial Least Square Regression"),
  Test_Error = c(lasso_rmse, lasso_1SE_rmse, enet_test_rmse, enet_1se_test_rmse, pls_test_rmse)
)

# Using kable to present table

knitr::kable(comparison_table)
```

Model	Test_Error
Lasso	20969.20
Lasso 1SE	20511.62
Elastic Net	20929.44
Elastic Net 1SE	20648.43
Partial Least Square Regression	20981.37

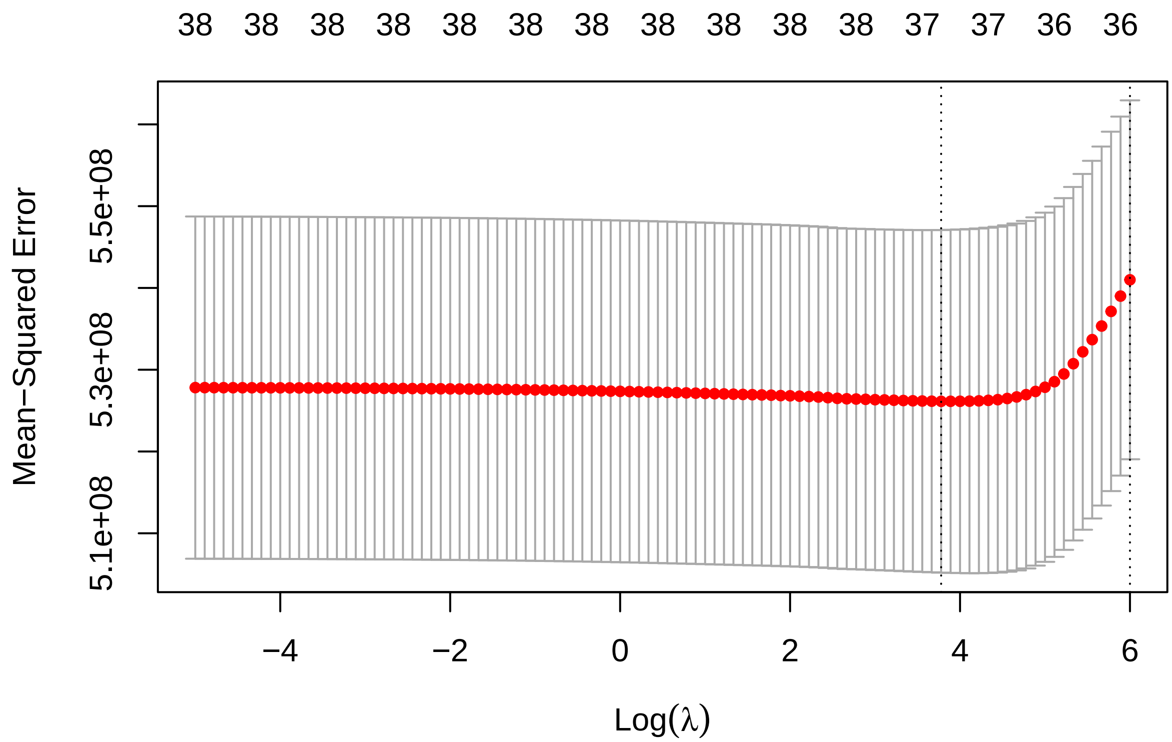
Based on the comparison table, the lasso model with the 1SE rule applied appears to perform the best as it has the lowest test error (RMSE = 20511.62) while undertaking feature selection. As discussed in class, the 1SE rule chooses the most regularised model whose error is within one standard error of the minimum, which allows for a parsimonious model that may generalise better to new data while maintaining predictive performance.

In the case of predicting the housing sale price, the lasso with 1SE model removed various less important predictors during variable selection, which reduced the overall complexity of the model while keeping good predictive performance. This aligns with the bias-variance tradeoff, in which with increasing regularisation (lambda), bias tends to increase but variance decreases, leading to improved generalisation.

The better performance of the lasso with 1SE compared to methods like elastic net and partial least squares regression suggests that this method is suitable as it provided the optimal balance in the bias-variance tradeoff for this housing price prediction task. This would be plausible since the lasso performs particularly well when there is a subset of true coefficients that are small or exactly zero, as appears to be the case with some of the housing price predictors.

Question (e): Retrain model using glmnet

```
# Set seed for reproducibility

set.seed(29)

# Matrix of training data predictors for glmnet

x.train <- model.matrix(Sale_Price ~ ., training_data)[,-1]

y.train <- training_data$Sale_Price

# Matrix of predictors for test data

x.test <- model.matrix(Sale_Price ~ ., testing_data)[,-1]


# Fit lasso

lasso_glmnet <- glmnet(x.train, y.train,
                       alpha = 1,
                       lambda = exp(seq(6, -5, length = 100)))

# Next, cross-validation for optimal lambda

cv.lasso <- cv.glmnet(x.train, y.train,
                      alpha = 1,
                      lambda = exp(seq(6, -5, length = 100)))

plot(cv.lasso)
```




```r
# Best lambda that minimises RMSE

print(cv.lasso$lambda.min)
```

```
## [1] 43.71878
```

```r
# Coefficients at optimal lambda

predict(cv.lasso, s = "lambda.min", type = "coefficients")
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                               lambda.min
## (Intercept)                -4.895849e+06
## Gr_Liv_Area                 6.563920e+01
## First_Flr_SF                7.866606e-01
## Second_Flr_SF               .
## Total_Bsmt_SF               3.534723e+01
## Low_Qual_Fin_SF            -4.125144e+01
## Wood_Deck_SF                1.175170e+01
## Open_Porch_SF               1.563393e+01
## Bsmt_Unf_SF                -2.089478e+01
## Mas_Vnr_Area                1.075014e+01
## Garage_Cars                 4.109831e+03
## Garage_Area                 8.092846e+00
## Year_Built                  3.235015e+02
```

```
## TotRms_AbvGrd              -3.679742e+03
## Full_Bath                  -3.994482e+03
## Overall_QualAverage        -4.909818e+03
## Overall_QualBelow_Average  -1.257695e+04
## Overall_QualExcellent       7.464381e+04
## Overall_QualFair           -1.091948e+04
## Overall_QualGood            1.216881e+04
## Overall_QualVery_Excellent  1.341508e+05
## Overall_QualVery_Good       3.792195e+04
## Kitchen_QualFair           -2.550193e+04
## Kitchen_QualGood           -1.779096e+04
## Kitchen_QualTypical        -2.585294e+04
## Fireplaces                  1.082636e+04
## Fireplace_QuFair           -7.714837e+03
## Fireplace_QuGood            .
## Fireplace_QuNo_Fireplace    1.883694e+03
## Fireplace_QuPoor           -5.696096e+03
## Fireplace_QuTypical        -7.002341e+03
## Exter_QualFair             -3.478720e+04
## Exter_QualGood             -1.640951e+04
## Exter_QualTypical          -2.086944e+04
## Lot_Frontage                1.005252e+02
## Lot_Area                    6.046880e-01
## Longitude                  -3.364932e+04
## Latitude                    5.645067e+04
## Misc_Val                    8.584413e-01
## Year_Sold                  -5.875701e+02
```

The final model includes 37 predictors, which is the same as the number of predictors identified in (a) using lasso (caret method).

Compared to the caret method implemented in Question (a), the tuning parameter lambda is notably different (glmnet = 43.71878, and caret = 68.18484). Both methods do use 10-fold cross validation to find the optimal lambda.

However, in glmnet, the built in cross validation function cv.glmnet() performs 10-fold cross validation once, computing RMSE for each lambda based on the 10 validation sets (Hastie et al., 2024). Whereas the caret package's trainControl(method = "cv", number = 10) also applies 10-fold cross validation, but averages the RMSE obtained across multiple resampling instances (Kuhn, 2020).

References

Kuhn, M. (2020). Caret package documentation.

Hastie, T., Tibshirani, R., & Friedman, J. (2024). An Introduction to glmnet.