Homework Unit 6: Regularization and Penalized Models

Nicole Friedl

2025-08-07

Introduction

To begin, download the following from the course web book (Unit 6):

- `hw_unit_06_regularization.qmd` (notebook for this assignment)
- `ames_full_cln.csv` (data for this assignment)
- `ames_data_dictionary.pdf` (data dictionary for the ames dataset)

The data for this week's assignment are the Ames housing price data you've seen in class and worked with in previous homework assignments. However, this week you're working with all 81 variables.

The data are already cleaned (i.e., variable names tidied, levels of categorical variables tidied, "none" put in for missing values where indicated in the data dictionary). You don't need to submit any modeling EDA because we're describing for you the specific steps to implement in the recipe. Of course you **normally** would do modeling EDA before fitting any models, but we're trying to keep the assignment from getting too long! Nonetheless, you probably want to skim the data to become more familiar with it!

In this assignment, you will practice tuning regularization hyperparameters (α and λ) and selecting among model configurations using resampling methods. Like last week, **fitting models will take a little longer**. We've kept the active coding component of the assignment to a reasonable length, but please try to plan for the increased run time! Remember that setting up parallel processing will dramatically reduce your run times. And you may consider the use of cache if you are comfortable with that process.

Let's get started!

Setup

Handle conflicts

```
options(conflicts.policy = "depends.ok")
devtools::source_url("https://github.com/jjcurtin/lab_support/blob/main/fun_ml.R?raw=true")
```

```
i SHA-1 hash of file is "32a0bc8ced92c79756b56ddcdc9a06e639795da6"
```

```
tidymodels_conflictRules()

# We also will need to resolve a new conflict using the following code
# Alternatively, John demonstrates code you can use when you load this library to prevent co
conflictRules("Matrix", mask.ok = c("expand", "pack", "unpack"))
```

Load required packages

```
library(tidyverse)
library(tidymodels)
library(tune)
library(rsample)
library(parsnip)
library(recipes)
library(workflows)
library(here)
library(glmnet)
```

Specify other global settings

If you are going to use `cache_rds(), you might include`rerun_setting <- FALSE‘ in this chunk

```
rerun_setting <- FALSE
```

Paths

```r
path_data <- "homework/data"
```

Set up parallel processing

```r
cl <- parallel::makePSOCKcluster(parallel::detectCores(logical = FALSE))
doParallel::registerDoParallel(cl)
```

Read in data

Read in the `ames_full_cln.csv` data file

```r
data_all <- read_csv(here::here(path_data, "ames_full_cln.csv"),
              show_col_types = FALSE)

str(data_all)
```

```
spc_tbl_ [1,955 x 81] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ pid            : chr [1:1955] "x0526301100" "x0526350040" "x0526351010" "x0527105010" ...
 $ ms_sub_class   : chr [1:1955] "x020" "x020" "x020" "x060" ...
 $ ms_zoning      : chr [1:1955] "rl" "rh" "rl" "rl" ...
 $ lot_frontage   : num [1:1955] 141 80 81 74 41 43 39 60 75 63 ...
 $ lot_area       : num [1:1955] 31770 11622 14267 13830 4920 ...
 $ street         : chr [1:1955] "pave" "pave" "pave" "pave" ...
 $ alley          : chr [1:1955] "none" "none" "none" "none" ...
 $ lot_shape      : chr [1:1955] "ir1" "reg" "ir1" "ir1" ...
 $ land_contour   : chr [1:1955] "lvl" "lvl" "lvl" "lvl" ...
 $ utilities      : chr [1:1955] "all_pub" "all_pub" "all_pub" "all_pub" ...
 $ lot_config     : chr [1:1955] "corner" "inside" "corner" "inside" ...
 $ land_slope     : chr [1:1955] "gtl" "gtl" "gtl" "gtl" ...
 $ neighborhood   : chr [1:1955] "n_ames" "n_ames" "n_ames" "gilbert" ...
 $ condition_1    : chr [1:1955] "norm" "feedr" "norm" "norm" ...
 $ condition_2    : chr [1:1955] "norm" "norm" "norm" "norm" ...
 $ bldg_type      : chr [1:1955] "one_fam" "one_fam" "one_fam" "one_fam" ...
 $ house_style    : chr [1:1955] "x1story" "x1story" "x1story" "x2story" ...
 $ overall_qual   : num [1:1955] 6 5 6 5 8 8 8 7 6 6 ...
 $ overall_cond   : num [1:1955] 5 6 6 5 5 5 5 5 5 5 ...
 $ year_built     : num [1:1955] 1960 1961 1958 1997 2001 ...
 $ year_remod_add : num [1:1955] 1960 1961 1958 1998 2001 ...
```

```
$ roof_style     : chr [1:1955] "hip" "gable" "hip" "gable" ...
$ roof_matl      : chr [1:1955] "comp_shg" "comp_shg" "comp_shg" "comp_shg" ...
$ exterior_1st   : chr [1:1955] "brk_face" "vinyl_sd" "wd_sdng" "vinyl_sd" ...
$ exterior_2nd   : chr [1:1955] "plywood" "vinyl_sd" "wd_sdng" "vinyl_sd" ...
$ mas_vnr_type   : chr [1:1955] "stone" "none" "brk_face" "none" ...
$ mas_vnr_area   : num [1:1955] 112 0 108 0 0 0 0 0 0 0 ...
$ exter_qual     : chr [1:1955] "ta" "ta" "ta" "ta" ...
$ exter_cond     : chr [1:1955] "ta" "ta" "ta" "ta" ...
$ foundation     : chr [1:1955] "c_block" "c_block" "c_block" "p_conc" ...
$ bsmt_qual      : chr [1:1955] "ta" "ta" "ta" "gd" ...
$ bsmt_cond      : chr [1:1955] "gd" "ta" "ta" "ta" ...
$ bsmt_exposure  : chr [1:1955] "gd" "no" "no" "no" ...
$ bsmt_fin_type_1: chr [1:1955] "blq" "rec" "alq" "glq" ...
$ bsmt_fin_sf_1  : num [1:1955] 639 468 923 791 616 263 1180 0 0 0 ...
$ bsmt_fin_type_2: chr [1:1955] "unf" "lw_q" "unf" "unf" ...
$ bsmt_fin_sf_2  : num [1:1955] 0 144 0 0 0 0 0 0 0 0 ...
$ bsmt_unf_sf    : num [1:1955] 441 270 406 137 722 ...
$ total_bsmt_sf  : num [1:1955] 1080 882 1329 928 1338 ...
$ heating        : chr [1:1955] "gas_a" "gas_a" "gas_a" "gas_a" ...
$ heating_qc     : chr [1:1955] "fa" "ta" "ta" "gd" ...
$ central_air    : chr [1:1955] "y" "y" "y" "y" ...
$ electrical     : chr [1:1955] "s_brkr" "s_brkr" "s_brkr" "s_brkr" ...
$ x1st_flr_sf    : num [1:1955] 1656 896 1329 928 1338 ...
$ x2nd_flr_sf    : num [1:1955] 0 0 0 701 0 0 0 776 892 676 ...
$ low_qual_fin_sf: num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
$ gr_liv_area    : num [1:1955] 1656 896 1329 1629 1338 ...
$ bsmt_full_bath : num [1:1955] 1 0 0 0 1 0 1 0 0 0 ...
$ bsmt_half_bath : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
$ full_bath      : num [1:1955] 1 1 1 2 2 2 2 2 2 2 ...
$ half_bath      : num [1:1955] 0 0 1 1 0 0 0 1 1 1 ...
$ bedroom_abv_gr : num [1:1955] 3 2 3 3 2 2 2 3 3 3 ...
$ kitchen_abv_gr : num [1:1955] 1 1 1 1 1 1 1 1 1 1 ...
$ kitchen_qual   : chr [1:1955] "ta" "ta" "gd" "ta" ...
$ tot_rms_abv_grd: num [1:1955] 7 5 6 6 6 5 5 7 7 7 ...
$ functional     : chr [1:1955] "typ" "typ" "typ" "typ" ...
$ fireplaces     : num [1:1955] 2 0 0 1 0 0 1 1 1 1 ...
$ fireplace_qu   : chr [1:1955] "gd" "none" "none" "ta" ...
$ garage_type    : chr [1:1955] "attchd" "attchd" "attchd" "attchd" ...
$ garage_yr_blt  : num [1:1955] 1960 1961 1958 1997 2001 ...
$ garage_finish  : chr [1:1955] "fin" "unf" "unf" "fin" ...
$ garage_cars    : num [1:1955] 2 1 1 2 2 2 2 2 2 2 ...
$ garage_area    : num [1:1955] 528 730 312 482 582 506 608 442 440 393 ...
$ garage_qual    : chr [1:1955] "ta" "ta" "ta" "ta" ...
```

```
$ garage_cond    : chr [1:1955] "ta" "ta" "ta" "ta" ...
$ paved_drive    : chr [1:1955] "p" "y" "y" "y" ...
$ wood_deck_sf   : num [1:1955] 210 140 393 212 0 0 237 140 157 0 ...
$ open_porch_sf  : num [1:1955] 62 0 36 34 0 82 152 60 84 75 ...
$ enclosed_porch : num [1:1955] 0 0 0 0 170 0 0 0 0 0 ...
$ x3ssn_porch    : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
$ screen_porch   : num [1:1955] 0 120 0 0 0 144 0 0 0 0 ...
$ pool_area      : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
$ pool_qc        : chr [1:1955] "none" "none" "none" "none" ...
$ fence          : chr [1:1955] "none" "mn_prv" "none" "mn_prv" ...
$ misc_feature   : chr [1:1955] "none" "none" "gar2" "none" ...
$ misc_val       : num [1:1955] 0 0 12500 0 0 0 0 0 0 0 ...
$ mo_sold        : num [1:1955] 5 6 6 3 4 1 3 6 4 5 ...
$ yr_sold        : num [1:1955] 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010 ...
$ sale_type      : chr [1:1955] "wd" "wd" "wd" "wd" ...
$ sale_condition : chr [1:1955] "normal" "normal" "normal" "normal" ...
$ sale_price     : num [1:1955] 215000 105000 172000 189900 213500 ...
- attr(*, "spec")=
 .. cols(
 ..   pid = col_character(),
 ..   ms_sub_class = col_character(),
 ..   ms_zoning = col_character(),
 ..   lot_frontage = col_double(),
 ..   lot_area = col_double(),
 ..   street = col_character(),
 ..   alley = col_character(),
 ..   lot_shape = col_character(),
 ..   land_contour = col_character(),
 ..   utilities = col_character(),
 ..   lot_config = col_character(),
 ..   land_slope = col_character(),
 ..   neighborhood = col_character(),
 ..   condition_1 = col_character(),
 ..   condition_2 = col_character(),
 ..   bldg_type = col_character(),
 ..   house_style = col_character(),
 ..   overall_qual = col_double(),
 ..   overall_cond = col_double(),
 ..   year_built = col_double(),
 ..   year_remod_add = col_double(),
 ..   roof_style = col_character(),
 ..   roof_matl = col_character(),
 ..   exterior_1st = col_character(),
```

```
..   exterior_2nd = col_character(),
..   mas_vnr_type = col_character(),
..   mas_vnr_area = col_double(),
..   exter_qual = col_character(),
..   exter_cond = col_character(),
..   foundation = col_character(),
..   bsmt_qual = col_character(),
..   bsmt_cond = col_character(),
..   bsmt_exposure = col_character(),
..   bsmt_fin_type_1 = col_character(),
..   bsmt_fin_sf_1 = col_double(),
..   bsmt_fin_type_2 = col_character(),
..   bsmt_fin_sf_2 = col_double(),
..   bsmt_unf_sf = col_double(),
..   total_bsmt_sf = col_double(),
..   heating = col_character(),
..   heating_qc = col_character(),
..   central_air = col_character(),
..   electrical = col_character(),
..   x1st_flr_sf = col_double(),
..   x2nd_flr_sf = col_double(),
..   low_qual_fin_sf = col_double(),
..   gr_liv_area = col_double(),
..   bsmt_full_bath = col_double(),
..   bsmt_half_bath = col_double(),
..   full_bath = col_double(),
..   half_bath = col_double(),
..   bedroom_abv_gr = col_double(),
..   kitchen_abv_gr = col_double(),
..   kitchen_qual = col_character(),
..   tot_rms_abv_grd = col_double(),
..   functional = col_character(),
..   fireplaces = col_double(),
..   fireplace_qu = col_character(),
..   garage_type = col_character(),
..   garage_yr_blt = col_double(),
..   garage_finish = col_character(),
..   garage_cars = col_double(),
..   garage_area = col_double(),
..   garage_qual = col_character(),
..   garage_cond = col_character(),
..   paved_drive = col_character(),
..   wood_deck_sf = col_double(),
```

```
  ..   open_porch_sf = col_double(),
  ..   enclosed_porch = col_double(),
  ..   x3ssn_porch = col_double(),
  ..   screen_porch = col_double(),
  ..   pool_area = col_double(),
  ..   pool_qc = col_character(),
  ..   fence = col_character(),
  ..   misc_feature = col_character(),
  ..   misc_val = col_double(),
  ..   mo_sold = col_double(),
  ..   yr_sold = col_double(),
  ..   sale_type = col_character(),
  ..   sale_condition = col_character(),
  ..   sale_price = col_double()
  .. )
 - attr(*, "problems")=<externalptr>
```

Set variable classes

Set all variables to factor or numeric classes. Here is where you will also want to explicitly set factor levels for those with low frequency count levels (e.g., `neighbohood`, `ms_sub_class`) and do any ordering of factor levels.

```r
data_all <- data_all |>
  mutate(
    across(c(overall_qual, overall_cond), ~factor(., ordered = TRUE)),
    across(c(exter_qual, exter_cond, bsmt_qual, bsmt_cond, heating_qc, kitchen_qual, fireplac


    neighborhood = fct_lump_min(neighborhood, min = 10),
    ms_sub_class = fct_lump_min(ms_sub_class, min = 10),

    across(c(ms_zoning, street, alley, lot_shape, land_contour, utilities, lot_config, land_s
  )

str(data_all)
```

```
tibble [1,955 x 81] (S3: tbl_df/tbl/data.frame)
 $ pid            : chr [1:1955] "x0526301100" "x0526350040" "x0526351010" "x0527105010" ...
 $ ms_sub_class   : Factor w/ 14 levels "x020","x030",..: 1 1 1 4 10 10 10 4 4 4 ...
 $ ms_zoning      : Factor w/ 7 levels "a","c","fv","i",..: 6 5 6 6 6 6 6 6 6 6 ...
```

```
$ lot_frontage  : num [1:1955] 141 80 81 74 41 43 39 60 75 63 ...
$ lot_area      : num [1:1955] 31770 11622 14267 13830 4920 ...
$ street        : Factor w/ 2 levels "grvl","pave": 2 2 2 2 2 2 2 2 2 2 ...
$ alley         : Factor w/ 3 levels "grvl","none",..: 2 2 2 2 2 2 2 2 2 2 ...
$ lot_shape     : Factor w/ 4 levels "ir1","ir2","ir3",..: 1 4 1 1 4 1 1 4 1 1 ...
$ land_contour  : Factor w/ 4 levels "bnk","hls","low",..: 4 4 4 4 4 2 4 4 4 4 ...
$ utilities     : Factor w/ 2 levels "all_pub","no_sewr": 1 1 1 1 1 1 1 1 1 1 ...
$ lot_config    : Factor w/ 5 levels "corner","cul_d_sac",..: 1 5 1 5 5 5 5 5 1 5 ...
$ land_slope    : Factor w/ 3 levels "gtl","mod","sev": 1 1 1 1 1 1 1 1 1 1 ...
$ neighborhood  : Factor w/ 25 levels "blmngtn","br_dale",..: 12 12 12 8 21 21 21 8 8 8 ..
$ condition_1   : Factor w/ 9 levels "artery","feedr",..: 3 2 3 3 3 3 3 3 3 3 ...
$ condition_2   : Factor w/ 6 levels "artery","feedr",..: 3 3 3 3 3 3 3 3 3 3 ...
$ bldg_type     : Factor w/ 5 levels "duplex","one_fam",..: 2 2 2 2 3 3 3 2 2 2 ...
$ house_style   : Factor w/ 8 levels "s_foyer","s_lvl",..: 5 5 5 8 5 5 5 8 8 8 ...
$ overall_qual  : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 6 5 6 5 8 8 8 7 6 6 ...
$ overall_cond  : Ord.factor w/ 9 levels "1"<"2"<"3"<"4"<..: 5 6 6 5 5 5 5 5 5 5 ...
$ year_built    : num [1:1955] 1960 1961 1958 1997 2001 ...
$ year_remod_add : num [1:1955] 1960 1961 1958 1998 2001 ...
$ roof_style    : Factor w/ 6 levels "flat","gable",..: 4 2 4 2 2 2 2 2 2 2 ...
$ roof_matl     : Factor w/ 7 levels "cly_tile","comp_shg",..: 2 2 2 2 2 2 2 2 2 2 ...
$ exterior_1st  : Factor w/ 15 levels "asb_shng","asph_shn",..: 4 13 14 13 6 7 6 13 7 13 .
$ exterior_2nd  : Factor w/ 17 levels "asb_shng","asph_shn",..: 11 15 16 15 6 7 6 15 7 15
$ mas_vnr_type  : Factor w/ 5 levels "brk_cmn","brk_face",..: 5 4 2 4 4 4 4 4 4 4 ...
$ mas_vnr_area  : num [1:1955] 112 0 108 0 0 0 0 0 0 0 ...
$ exter_qual    : Ord.factor w/ 4 levels "ex"<"fa"<"gd"<..: 4 4 4 4 3 3 3 4 4 4 ...
$ exter_cond    : Ord.factor w/ 5 levels "ex"<"fa"<"gd"<..: 5 5 5 5 5 5 5 5 5 5 ...
$ foundation    : Factor w/ 6 levels "brk_til","c_block",..: 2 2 2 3 3 3 3 3 3 3 ...
$ bsmt_qual     : Ord.factor w/ 5 levels "ex"<"fa"<"gd"<..: 5 5 5 3 3 3 3 5 3 3 ...
$ bsmt_cond     : Ord.factor w/ 6 levels "ex"<"fa"<"gd"<..: 3 6 6 6 6 6 6 6 6 6 ...
$ bsmt_exposure  : chr [1:1955] "gd" "no" "no" "no" ...
$ bsmt_fin_type_1: chr [1:1955] "blq" "rec" "alq" "glq" ...
$ bsmt_fin_sf_1  : num [1:1955] 639 468 923 791 616 263 1180 0 0 0 ...
$ bsmt_fin_type_2: chr [1:1955] "unf" "lw_q" "unf" "unf" ...
$ bsmt_fin_sf_2  : num [1:1955] 0 144 0 0 0 0 0 0 0 0 ...
$ bsmt_unf_sf    : num [1:1955] 441 270 406 137 722 ...
$ total_bsmt_sf  : num [1:1955] 1080 882 1329 928 1338 ...
$ heating       : Factor w/ 6 levels "floor","gas_a",..: 2 2 2 2 2 2 2 2 2 2 ...
$ heating_qc    : Ord.factor w/ 5 levels "ex"<"fa"<"gd"<..: 2 5 5 3 1 1 1 3 3 3 ...
$ central_air   : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 2 2 2 2 ...
$ electrical    : Factor w/ 5 levels "fuse_a","fuse_f",..: 5 5 5 5 5 5 5 5 5 5 ...
$ x1st_flr_sf   : num [1:1955] 1656 896 1329 928 1338 ...
$ x2nd_flr_sf   : num [1:1955] 0 0 0 701 0 0 0 776 892 676 ...
$ low_qual_fin_sf: num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
```

```
 $ gr_liv_area    : num [1:1955] 1656 896 1329 1629 1338 ...
 $ bsmt_full_bath : num [1:1955] 1 0 0 0 1 0 1 0 0 0 ...
 $ bsmt_half_bath : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
 $ full_bath      : num [1:1955] 1 1 1 2 2 2 2 2 2 2 ...
 $ half_bath      : num [1:1955] 0 0 1 1 0 0 0 1 1 1 ...
 $ bedroom_abv_gr : num [1:1955] 3 2 3 3 2 2 2 3 3 3 ...
 $ kitchen_abv_gr : num [1:1955] 1 1 1 1 1 1 1 1 1 1 ...
 $ kitchen_qual   : Ord.factor w/ 5 levels "ex"<"fa"<"gd"<..: 5 5 3 5 3 3 3 3 5 5 ...
 $ tot_rms_abv_grd: num [1:1955] 7 5 6 6 6 5 5 7 7 7 ...
 $ functional     : Factor w/ 8 levels "maj1","maj2",..: 8 8 8 8 8 8 8 8 8 8 ...
 $ fireplaces     : num [1:1955] 2 0 0 1 0 0 1 1 1 1 ...
 $ fireplace_qu   : Ord.factor w/ 6 levels "ex"<"fa"<"gd"<..: 3 4 4 6 4 4 6 6 6 3 ...
 $ garage_type    : Factor w/ 7 levels "attchd","basment",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ garage_yr_blt  : num [1:1955] 1960 1961 1958 1997 2001 ...
 $ garage_finish  : Factor w/ 4 levels "fin","none","r_fn",..: 1 4 4 1 1 3 3 1 1 1 ...
 $ garage_cars    : num [1:1955] 2 1 1 2 2 2 2 2 2 2 ...
 $ garage_area    : num [1:1955] 528 730 312 482 582 506 608 442 440 393 ...
 $ garage_qual    : Ord.factor w/ 6 levels "ex"<"fa"<"gd"<..: 6 6 6 6 6 6 6 6 6 6 ...
 $ garage_cond    : Ord.factor w/ 6 levels "ex"<"fa"<"gd"<..: 6 6 6 6 6 6 6 6 6 6 ...
 $ paved_drive    : Factor w/ 3 levels "n","p","y": 2 3 3 3 3 3 3 3 3 3 ...
 $ wood_deck_sf   : num [1:1955] 210 140 393 212 0 0 237 140 157 0 ...
 $ open_porch_sf  : num [1:1955] 62 0 36 34 0 82 152 60 84 75 ...
 $ enclosed_porch : num [1:1955] 0 0 0 0 170 0 0 0 0 0 ...
 $ x3ssn_porch    : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
 $ screen_porch   : num [1:1955] 0 120 0 0 0 144 0 0 0 0 ...
 $ pool_area      : num [1:1955] 0 0 0 0 0 0 0 0 0 0 ...
 $ pool_qc        : Ord.factor w/ 5 levels "ex"<"fa"<"gd"<..: 4 4 4 4 4 4 4 4 4 4 ...
 $ fence          : Factor w/ 5 levels "gd_prv","gd_wo",..: 5 3 5 3 5 5 5 5 5 5 ...
 $ misc_feature   : Factor w/ 5 levels "gar2","none",..: 2 2 1 2 2 2 2 2 2 2 ...
 $ misc_val       : num [1:1955] 0 0 12500 0 0 0 0 0 0 0 ...
 $ mo_sold        : num [1:1955] 5 6 6 3 4 1 3 6 4 5 ...
 $ yr_sold        : num [1:1955] 2010 2010 2010 2010 2010 2010 2010 2010 2010 2010 ...
 $ sale_type      : Factor w/ 10 levels "cod","con","con_ld",..: 10 10 10 10 10 10 10 10 10 1
 $ sale_condition : Factor w/ 6 levels "abnorml","adj_land",..: 5 5 5 5 5 5 5 5 5 5 ...
 $ sale_price     : num [1:1955] 215000 105000 172000 189900 213500 ...
```

Set up splits

Divide data into train and test

Hold out 25% of the data as `data_test` for evaluation using the `initial_split()` function. Stratify on `sale_price`. Don't forget to set a seed!

```r
set.seed(123)
splits_test <- initial_split(data_all, prop = 0.75, strata = "sale_price")

data_trn <- training(splits_test)
data_test <- testing(splits_test)
```

Make splits for hyperparameter tuning

For parts 2 and 3, you'll need splits within `data_trn` to select among among model configurations using held out data. Create 100 bootstrap splits stratified on `sale_price`. You do not need to set a new seed.

```r
splits_boot <- bootstraps(data_trn, times = 100, strata = "sale_price")
splits_boot
```

```
# Bootstrap sampling using stratification
# A tibble: 100 x 2
   splits             id
   <list>             <chr>
 1 <split [1465/538]> Bootstrap001
 2 <split [1465/530]> Bootstrap002
 3 <split [1465/556]> Bootstrap003
 4 <split [1465/538]> Bootstrap004
 5 <split [1465/539]> Bootstrap005
 6 <split [1465/528]> Bootstrap006
 7 <split [1465/547]> Bootstrap007
 8 <split [1465/529]> Bootstrap008
 9 <split [1465/530]> Bootstrap009
10 <split [1465/543]> Bootstrap010
# i 90 more rows
```

Build recipe

You will build one recipe that can be used across three model fits. Please follow these instructions to build your recipe:

- Regress the outcome `sale_price` on all predictors
- Remove the ID variable (`pid`) with `step_rm()`

- Use `step_impute_median()` to impute the median for any missing values in numeric predictors
- Use `step_impute_mode()` to impute the mode for any missing values in the factor predictors
- Use `step_YeoJohnson()` to apply Yeo-Johnson transformations to all numeric predictors
- Use `step_normalize()` to normalize all numeric predictors (necessary for regularized models)
- Apply dummy coding to all factor predictors

```r
rec <- recipe(sale_price ~ ., data = data_trn) |>
  step_rm(pid) |>
  step_zv(yr_sold) |>
  step_mutate_at(all_of(c(
    "overall_qual", "overall_cond", "exter_qual", "exter_cond",
    "bsmt_qual", "bsmt_cond", "heating_qc", "kitchen_qual",
    "fireplace_qu", "garage_qual", "garage_cond", "pool_qc"
  )), fn = as.numeric) |>
  step_impute_median(all_numeric_predictors()) |>
  step_impute_mode(all_factor_predictors()) |>
  step_YeoJohnson(all_numeric_predictors()) |>
  step_normalize(all_numeric_predictors()) |>
  step_dummy(all_factor_predictors())

rec_prep <- prep(rec)
bake(rec_prep, new_data = NULL)
```

```
# A tibble: 1,465 x 245
   lot_frontage lot_area overall_qual overall_cond year_built year_remod_add
          <dbl>    <dbl>        <dbl>        <dbl>      <dbl>          <dbl>
 1       0.0912  -0.128        -1.51        -0.482    -0.0528         -0.651
 2       0.0912   0.306        -1.51        -0.482    -0.0192         -0.603
 3      -2.86    -3.08         -0.751       -0.482    -0.0192         -0.603
 4      -0.763   -1.51         -0.0275      -0.482     0.182          -0.318
 5      -0.656   -0.248        -0.0275      -0.482     0.249          -0.223
 6      -0.00352 -0.00978      -0.0275      -0.482     0.216          -0.270
 7       0.0440   0.175        -0.751       -0.482    -0.153          -0.794
 8       0.0440  -0.506        -0.751        2.17     -0.321           1.25
 9       0.413    0.0731       -1.51        -0.482    -0.422          -1.17
10       1.30     0.766        -0.751        0.436    -0.321           0.825
# i 1,455 more rows
```

```
# i 239 more variables: mas_vnr_area <dbl>, exter_qual <dbl>, exter_cond <dbl>,
#   bsmt_qual <dbl>, bsmt_cond <dbl>, bsmt_fin_sf_1 <dbl>, bsmt_fin_sf_2 <dbl>,
#   bsmt_unf_sf <dbl>, total_bsmt_sf <dbl>, heating_qc <dbl>,
#   x1st_flr_sf <dbl>, x2nd_flr_sf <dbl>, low_qual_fin_sf <dbl>,
#   gr_liv_area <dbl>, bsmt_full_bath <dbl>, bsmt_half_bath <dbl>,
#   full_bath <dbl>, half_bath <dbl>, bedroom_abv_gr <dbl>, ...
```

Create error tracking tibble

```
track_rmse <- tibble(model = character(),
                     rmse_trn = numeric(),
                     rmse_test = numeric(),
                     n_features = numeric())
```

Part 1: Fitting an OLS linear regression

Fit a regression model in the full training set

Make a feature matrix for training data and for test data.

```
x_train <- bake(rec_prep, new_data = data_trn) |>
  select(-sale_price) |>
  as.matrix()

x_test <- bake(rec_prep, new_data = data_test) |>
  select(-sale_price) |>
  as.matrix()

y_train <- data_trn$sale_price
y_test <- data_test$sale_price
```

Fit linear regression model. No resampling is needed because there are no hyperparameters to tune.

```r
ols_model <- linear_reg() |>
  set_engine("lm") |>
  fit(sale_price ~ ., data = bake(rec_prep, new_data = data_trn))

ols_model |>
  tidy() |>
  print(n = 21)
```

```
# A tibble: 245 x 5
   term              estimate std.error statistic  p.value
   <chr>                <dbl>     <dbl>     <dbl>    <dbl>
 1 (Intercept)     -461773.     87227.    -5.29   1.42e- 7
 2 lot_frontage        125.      1202.     0.104  9.17e- 1
 3 lot_area           9370.      1547.     6.06   1.82e- 9
 4 overall_qual      11498.      1570.     7.32   4.35e-13
 5 overall_cond       5804.       997.     5.82   7.45e- 9
 6 year_built         4273.      2668.     1.60   1.10e- 1
 7 year_remod_add     1391.      1280.     1.09   2.77e- 1
 8 mas_vnr_area       8579.      4895.     1.75   7.99e- 2
 9 exter_qual        -4628.      1215.    -3.81   1.47e- 4
10 exter_cond           97.2      790.     0.123  9.02e- 1
11 bsmt_qual         -4607.      1185.    -3.89   1.07e- 4
12 bsmt_cond           800.       791.     1.01   3.13e- 1
13 bsmt_fin_sf_1      5853.      3306.     1.77   7.69e- 2
14 bsmt_fin_sf_2       840.      9661.     0.0869 9.31e- 1
15 bsmt_unf_sf       -3607.      1949.    -1.85   6.44e- 2
16 total_bsmt_sf     14814.      2524.     5.87   5.65e- 9
17 heating_qc        -1367.       983.    -1.39   1.65e- 1
18 x1st_flr_sf        3638.      3336.     1.09   2.76e- 1
19 x2nd_flr_sf       12525.      4590.     2.73   6.45e- 3
20 low_qual_fin_sf    -264.      1035.    -0.256  7.98e- 1
21 gr_liv_area       14389.      3875.     3.71   2.14e- 4
# i 224 more rows
```

Get RMSE in train & test

Use `rmse_vec()` to get error in `feat_trn`

Notice the warnings we get when making these predictions. R is telling us that our models are rank-deficient - this means that this model may not have determined a unique set of parameter estimates to minimize the cost function. This can occur for a variety of reasons, some of which are real problems and other time for reasons that are not a problem. For now continue on!

```r
lin_trn_rmse <- rmse_vec(truth = y_train, estimate = predict(ols_model, new_data = bake(rec_p
```

```
Warning in predict.lm(object = object$fit, newdata = new_data, type =
"response", : prediction from rank-deficient fit; consider predict(.,
rankdeficient="NA")
```

Use `rmse_vec()` to get error in `feat_test`

```r
lin_test_rmse <- rmse_vec(truth = y_test, estimate = predict(ols_model, new_data = bake(rec_p
```

```
Warning in predict.lm(object = object$fit, newdata = new_data, type =
"response", : prediction from rank-deficient fit; consider predict(.,
rankdeficient="NA")
```

Get number of features

```r
lin_n_feat <- ols_model |>
  tidy() |>
  filter(estimate != 0 & term != "(Intercept)") %>%
  nrow()
```

Add to tracking tibble

```r
track_rmse <- add_row(track_rmse,
                      model = "OLS",
                      rmse_trn = lin_trn_rmse,
                      rmse_test = lin_test_rmse,
                      n_features = lin_n_feat)

track_rmse
```

```
# A tibble: 1 x 4
  model rmse_trn rmse_test n_features
  <chr>    <dbl>     <dbl>      <dbl>
1 OLS     22678.    34327.        238
```

How does performance compare in training and test? Type your response between the asterisks.

The OLS model's performance is significantly worse in the test set (RMSE = 34327) compared to the training set (RMSE = 22678). This indicates substantial overfitting, meaning the model has captured noise in the training data.

Part 2: Fitting a LASSO regression

Set up a hyperparameter grid

In the LASSO, the mixture hyperparameter (α) will be set to 1, but we'll need to tune the penalty hyperparameter (λ).

```
grid_penalty <- expand_grid(penalty = exp(seq(-6, 8, length.out = 500)))
```

Tune a LASSO regression

Use `linear_reg()`, `set_engine("glmnet")`, and `tune_grid()` to fit your LASSO models.

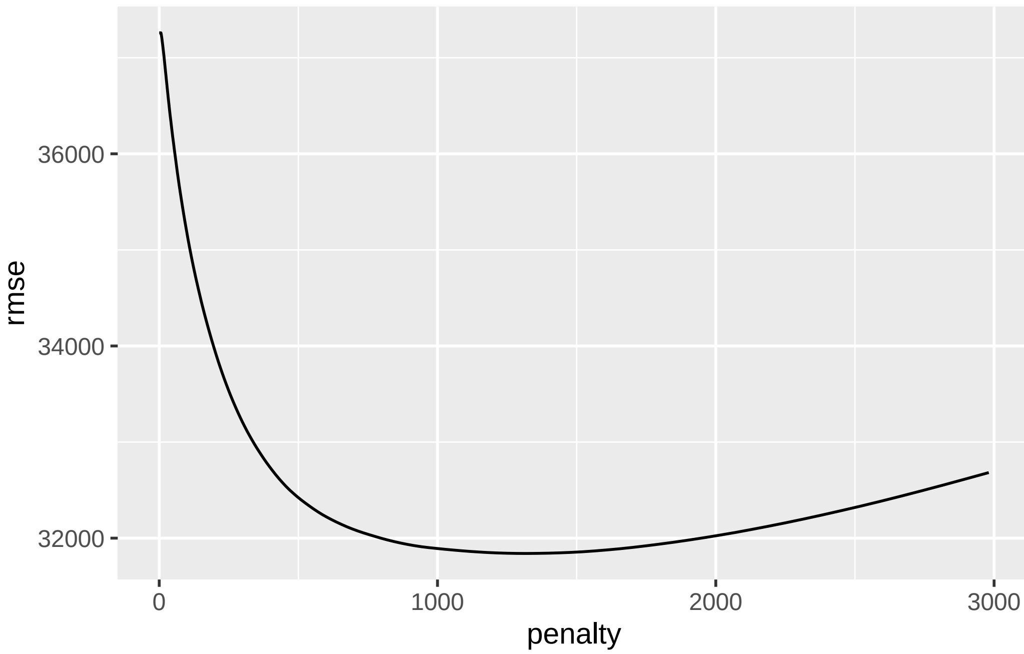
```
fits_lasso <- linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  tune_grid(
    preprocessor = rec,
    resamples = splits_boot,
    grid = grid_penalty,
    metrics = metric_set(rmse),
    control = control_grid(extract = function(x) prep(x$preprocessor))
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

Plot performance in the validation sets by hyperparameter

Use the `plot_hyperparameters()` function in `fun_ml.R` or your own code.

```
plot_hyperparameters(fits_lasso, hp1 = "penalty", metric = "rmse")
```



Fit your best configuration in data_trn

Use your best configuration (i.e., your best λ value) to fit a model in the full training set (`data_trn`) using `select_best()`.

```
fit_lasso <- linear_reg(penalty = select_best(fits_lasso)$penalty, mixture = 1) |>
  set_engine("glmnet") |>
  fit(sale_price ~ ., data = bake(rec_prep, new_data = data_trn))
```

```
Warning in select_best(fits_lasso): No value of `metric` was given; "rmse" will be used.
No value of `metric` was given; "rmse" will be used.
No value of `metric` was given; "rmse" will be used.
```

Examine parameter estimates

```
library(Matrix, exclude = c("expand", "pack", "unpack"))
best_lambda <- select_best(fits_lasso)$penalty
```

```
Warning in select_best(fits_lasso): No value of `metric` was given; "rmse" will
be used.
```

```r
best_coef <- coef(fit_lasso$fit, s = best_lambda)
tidy_lasso <- as.data.frame(as.matrix(best_coef))
colnames(tidy_lasso) <- "estimate"
tidy_lasso$term <- rownames(best_coef)
tidy_lasso <- tidy_lasso[-1, ]
tidy_lasso$penalty <- best_lambda
tidy_lasso$estimate <- as.numeric(tidy_lasso$estimate)
tidy_lasso <- na.omit(tidy_lasso)
tidy_lasso <- tidy_lasso[, c("term", "estimate", "penalty")]

print(tidy_lasso[1:21, ], na.print = "NA")
```

```
                             term   estimate  penalty
lot_frontage       lot_frontage     0.0000 1321.302
lot_area               lot_area  6132.0921 1321.302
overall_qual       overall_qual 17100.9369 1321.302
overall_cond       overall_cond  3211.9722 1321.302
year_built           year_built  1523.3478 1321.302
year_remod_add   year_remod_add  1583.3662 1321.302
mas_vnr_area       mas_vnr_area     0.0000 1321.302
exter_qual           exter_qual -4800.0196 1321.302
exter_cond           exter_cond     0.0000 1321.302
bsmt_qual             bsmt_qual -4166.4738 1321.302
bsmt_cond             bsmt_cond     0.0000 1321.302
bsmt_fin_sf_1     bsmt_fin_sf_1  5013.0562 1321.302
bsmt_fin_sf_2     bsmt_fin_sf_2     0.0000 1321.302
bsmt_unf_sf         bsmt_unf_sf     0.0000 1321.302
total_bsmt_sf     total_bsmt_sf  1452.9663 1321.302
heating_qc           heating_qc  -877.1897 1321.302
x1st_flr_sf         x1st_flr_sf  3347.7581 1321.302
x2nd_flr_sf         x2nd_flr_sf     0.0000 1321.302
low_qual_fin_sf low_qual_fin_sf     0.0000 1321.302
gr_liv_area         gr_liv_area 18144.9764 1321.302
bsmt_full_bath   bsmt_full_bath  2639.8681 1321.302
```

Get RMSE in train & test

Use `rmse_vec()` to get error in `feat_trn`

```r
lasso_trn_rmse <- rmse_vec(
  truth = data_trn$sale_price,
  estimate = predict(fit_lasso, new_data = bake(rec_prep, new_data = data_trn))$.pred
)
```

Use `rmse_vec()` to get error in `feat_test`

```r
lasso_test_rmse <- rmse_vec(
  truth = data_test$sale_price,
  estimate = predict(fit_lasso, new_data = bake(rec_prep, new_data = data_test))$.pred
)
```

Get number of features

```r
lasso_n_feat <- sum(best_coef != 0) - 1

print(lasso_n_feat)
```

```
[1] 73
```

Add to `track_rmse`

```r
track_rmse <- add_row(
  track_rmse,
  model = "LASSO",
  rmse_trn = lasso_trn_rmse,
  rmse_test = lasso_test_rmse,
  n_features = lasso_n_feat
)

track_rmse
```

```
# A tibble: 2 x 4
  model rmse_trn rmse_test n_features
  <chr>    <dbl>     <dbl>      <dbl>
1 OLS     22678.    34327.        238
2 LASSO   27421.    28365.         73
```

How does performance compare in training and test for LASSO? Type your response between the asterisks.

The LASSO model's performance is slightly better in the test set (RMSE = 28365) compared to the training set (RMSE = 27421). This suggests that the LASSO model is doing well to unseen data and is not overfitting as much as the OLS model. Additionally, the LASSO model uses only 73 features, which is significantly less than the 238 features used by the OLS model. This indicates that the LASSO model is able to achieve comparable performance with a smaller set of features, which can be beneficial for interpretability.

How many features were retained (i.e., parameter estimates not dropped to zero) in this model? How does this compare to the OLS linear regression? Type your response between the asterisks.

The LASSO model retained 73 features, meaning 73 parameter estimates were not dropped to zero. This is a significant reduction compared to the OLS linear regression, which retained 238 features. The LASSO model's feature selection capability reduced the model's complexity.

Part 3: Fitting an Elastic Net regression

Set up a hyperparameter grid

Now we'll need to tune both the mixture hyperparameter (α) and the penalty hyperparameter (λ).

```
grid_glmnet <- expand_grid(penalty = exp(seq(-10, 11, length.out = 250)),
                           mixture = seq(0, 1, length.out = 11))
```

Tune an elasticnet regression

Use `linear_reg()`, `set_engine("glmnet")`, and `tune_grid()` to fit your LASSO models.

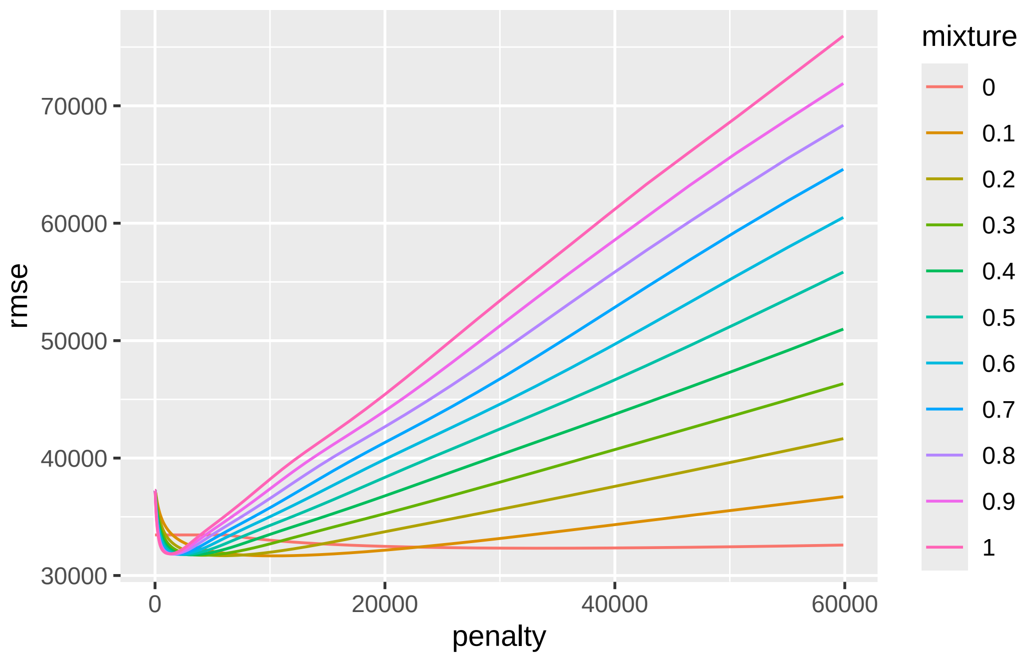
```
fits_glmnet <- linear_reg(penalty = tune(), mixture = tune()) |>
  set_engine("glmnet") |>
  tune_grid(
    preprocessor = rec,
    resamples = splits_boot,
    grid = grid_glmnet,
    metrics = metric_set(rmse),
    control = control_grid(extract = function(x) prep(x$preprocessor))
  )
```

```
Warning: ! tune detected a parallel backend registered with foreach but no backend
  registered with future.
i Support for parallel processing with foreach was soft-deprecated in tune
  1.2.1.
i See ?parallelism (`?tune::parallelism()`) to learn more.
```

Plot performance in the validation sets by hyperparameter

Use the `plot_hyperparameters()` function or your own code.

```r
plot_hyperparameters(fits_glmnet, hp1 = "penalty", hp2 = "mixture", metric = "rmse")
```



Fit your best configuration in training data

Use your best configuration (i.e., your best combination of α & λ values) to fit a model in the full training set (`data_trn`) using `select_best()`.

```r
best_glmnet <- select_best(fits_glmnet)
```

```
Warning in select_best(fits_glmnet): No value of `metric` was given; "rmse"
will be used.
```

```r
fit_glmnet <- linear_reg(penalty = best_glmnet$penalty, mixture = best_glmnet$mixture) |>
  set_engine("glmnet") |>
  fit(sale_price ~ ., data = bake(rec_prep, new_data = data_trn))
```

Examine parameter estimates

```r
best_lambda <- select_best(fits_glmnet)$penalty
```

```
Warning in select_best(fits_glmnet): No value of `metric` was given; "rmse"
will be used.
```

```r
best_coef <- coef(fit_glmnet$fit, s = best_lambda)
tidy_glmnet <- data.frame(
  term = rownames(best_coef),
  estimate = as.vector(best_coef)
)
tidy_glmnet <- tidy_glmnet[-1, ]
tidy_glmnet$penalty <- best_lambda
tidy_glmnet$estimate <- as.numeric(tidy_glmnet$estimate)
tidy_glmnet <- na.omit(tidy_glmnet)
tidy_glmnet <- tidy_glmnet[, c("term", "estimate", "penalty")]

print(tidy_glmnet[1:21, ], na.print = "NA")
```

```
             term    estimate  penalty
2     lot_frontage     0.00000 10187.49
3         lot_area  5577.42758 10187.49
4     overall_qual 13418.49842 10187.49
5     overall_cond  3345.28027 10187.49
6       year_built  1012.85907 10187.49
7   year_remod_add  1882.74987 10187.49
8     mas_vnr_area     0.00000 10187.49
9       exter_qual -5041.26731 10187.49
10      exter_cond     0.00000 10187.49
11       bsmt_qual -4362.88918 10187.49
12       bsmt_cond    42.27295 10187.49
13   bsmt_fin_sf_1  3874.27115 10187.49
14   bsmt_fin_sf_2     0.00000 10187.49
15     bsmt_unf_sf     0.00000 10187.49
```

```
16   total_bsmt_sf  2886.31373 10187.49
17      heating_qc -1454.64527 10187.49
18     x1st_flr_sf  5019.65930 10187.49
19     x2nd_flr_sf     0.00000 10187.49
20 low_qual_fin_sf     0.00000 10187.49
21     gr_liv_area 12193.77893 10187.49
22  bsmt_full_bath  2744.89755 10187.49
```

Get RMSE in train & test

Use `rmse_vec()` to get error in `feat_trn`

```
glmnet_trn_rmse <- rmse_vec(
  truth = data_trn$sale_price,
  estimate = predict(fit_glmnet, new_data = bake(rec_prep, new_data = data_trn))$.pred
)
```

Use `rmse_vec()` to get error in `feat_test`

```
glmnet_test_rmse <- rmse_vec(
  truth = data_test$sale_price,
  estimate = predict(fit_glmnet, new_data = bake(rec_prep, new_data = data_test))$.pred
)
```

Get number of features

```
best_coef_glmnet <- coef(fit_glmnet$fit, s = best_lambda)
glmnet_n_feat <- sum(best_coef_glmnet != 0) - 1
```

Add to `track_rmse`

```
track_rmse <- add_row(
  track_rmse,
  model = "Elastic Net",
  rmse_trn = glmnet_trn_rmse,
  rmse_test = glmnet_test_rmse,
  n_features = glmnet_n_feat
)

track_rmse
```

```
# A tibble: 3 x 4
  model        rmse_trn rmse_test n_features
  <chr>           <dbl>     <dbl>      <dbl>
1 OLS            22678.    34327.        238
2 LASSO          27421.    28365.         73
3 Elastic Net    27081.    27625.         96
```

How does performance compare in training and test for glmnet? Type your response between the asterisks.

The glmnet (Elastic Net) model's performance is slightly better in the test set (RMSE = 27625) compared to the training set (RMSE = 27081). This suggests good generalization and minimal overfitting.

How many features were retained (i.e., parameter estimates not dropped to zero) in this model? How does this compare to the OLS linear regression? Type your response between the asterisks.

The Elastic Net model retained 96 features. This is significantly fewer than the 238 features retained by the OLS linear regression, indicating that Elastic Net performed feature selection and reduced model complexity. However, it retained more features than the LASSO model, which had 73 features.

Looking back across the OLS linear regression, the LASSO regression, and the elastic net regression, what comparisons can you make about performance in training, performance in test, evidence of overfitting, etc.? Which model configuration would you select and why? Type your response between the asterisks.

Across the three models, the OLS linear regression exhibited the most significant overfitting, with a much lower RMSE in the training set (22678) compared to the test set (34327). This indicates that the OLS model captured noise in the training data, leading to poor generalization. Both LASSO and Elastic Net showed better generalization, with test RMSEs closer to their training RMSEs. LASSO had a training RMSE of 27421 and a test RMSE of 28365, while Elastic Net had a training RMSE of 27081 and a test RMSE of 27625.In terms of feature selection, OLS retained all 238 features, while LASSO retained 73 and Elastic Net retained 96. This demonstrates the effectiveness of LASSO and Elastic Net in reducing model complexity. For model selection, the Elastic Net configuration would be preferred. It had the lowest test RMSE (27625), indicating the best performance on unseen data, and a good balance between model complexity and predictive accuracy. While LASSO had fewer features, the slight improvement in test RMSE for Elastic Net suggests a better trade-off between bias and variance in this particular dataset. Also, the Elastic Net provides a more flexible approach because it can replicate both Ridge (mixture=0) and Lasso (mixture=1) regression, and any value in between.

Save & render

Save this .qmd file with your last name at the end (e.g., hw_unit_06_regularization_wyant). Make sure you changed "Your name here" at the top of the file to be your own name. Render the file to .html, and upload the rendered file to Canvas.

Way to go!!