STA_478_F25_Assignment_7_Kaleb_Coleman

Kaleb Coleman

2025-10-27

Exercise 1 — Construct a function that will calculate the Variance Inflation Factors of all predictors.

```r
# Function: compute_vif
# Purpose: Calculate Variance Inflation Factors (VIF) for all predictors
# Input: A data frame or matrix where each column is a predictor variable
# Output: A numeric vector of VIF values

compute_vif <- function(data) {
  # Ensure input is a data frame
  data <- as.data.frame(data)

  vif_values <- numeric(ncol(data))
  names(vif_values) <- colnames(data)

  # Loop over each predictor
  for (i in seq_along(data)) {
    # Define the target variable for predictor i
    target <- data[[i]]

    # Define all other predictors
    others <- data[, -i, drop = FALSE]

    # Fit regression model: target ~ others
    model <- lm(target ~ ., data = others)

    # Extract R-squared value
    R2 <- summary(model)$r.squared

    # Compute VIF
    vif_values[i] <- 1 / (1 - R2)
  }

  # Return out all of the values
  return(vif_values)
}
```

Note: This chunk builds a reusable helper that returns the VIF for every predictor; no printed output is expected here.

Exercise 2 — VIF-Based Feature Sets for the Boston Data

(a) VIF table for Boston predictors (excluding crim)

```r
# Load data
data("Boston")

# Exclude 'crim' and compute VIFs for all remaining predictors
X <- subset(Boston, select = -crim)

vif_vals <- compute_vif(X)

# formatted table (largest VIF first)
vif_tbl <- data.frame(
  Predictor = names(vif_vals),
  VIF = as.numeric(vif_vals)
)
vif_tbl <- vif_tbl[order(-vif_tbl$VIF), ]

# Print as a table (works well in Rmd)
kable(vif_tbl, digits = 3,
      caption = "VIF for each Boston predictor (excluding `crim`).")
```

Table 1: VIF for each Boston predictor (excluding `crim`).

	Predictor	VIF
9	tax	9.195
8	rad	7.159
4	nox	4.552
7	dis	4.289
2	indus	3.988
13	medv	3.773
12	lstat	3.561
6	age	3.101
1	zn	2.325
5	rm	2.258
10	ptratio	1.984
11	black	1.370
3	chas	1.094

Note: The first VIF table identifies multicollinearity among predictors in the Boston dataset, excluding *crim*. The highest VIF values occur for *tax* (9.20) and *rad* (7.16), indicating substantial redundancy between these two variables. Moderate collinearity appears for *nox*, *dis*, and *indus*, while predictors such as *zn*, *rm*, and *ptratio* show relatively low inflation factors. Overall, *tax* and *rad* would be prime candidates for removal in later models.

(b) Build the VIF.5 predictor set

```r
# Load data and set up predictors (excluding `crim`)
data("Boston")
X <- subset(Boston, select = -crim)

# Helper that iteratively removes the variable with the largest VIF
# Recursively drop the predictor with the largest VIF until all VIFs < threshold
stepwise_vif <- function(df, threshold = 5) {
  df <- as.data.frame(df)

  recurse <- function(df, step = 0L, removed = NULL) {
    vifs <- compute_vif(df)
    mx   <- max(vifs, na.rm = TRUE)

    # stop condition
    if (is.na(mx) || mx <= threshold || ncol(df) <= 1L) {
      return(list(kept = names(df),
                  removed = removed,
                  final_vif = vifs))
    }

    # drop the max-VIF variable and recurse
    drop_var <- names(which.max(vifs))
    row <- data.frame(Step = step + 1L,
                      Dropped = drop_var,
                      VIF = unname(mx),
                      row.names = NULL)
    df[[drop_var]] <- NULL
    recurse(df, step + 1L, rbind(removed, row))
  }

  recurse(df)
}
```

Note: This helper recursively drops the largest-VIF predictor until all remaining variables stay under the chosen threshold. Start with all predictors (excluding *crim*); while $\max_j VIF_j > 5$, remove the predictor with the largest VIF_j. The process ends once every $VIF_j \leq 5$, yielding the VIF_5 feature set.

```
# Run with threshold 5 and name the resulting set VIF.5
vif5_res <- stepwise_vif(X, threshold = 5)
VIF.5 <- vif5_res$kept

# Show the final VIFs
kable(data.frame(Predictor = names(vif5_res$final_vif),
                        VIF = as.numeric(vif5_res$final_vif))[order(-vif5_res$final_vif), ],
             digits = 3,
             caption = "Final VIFs for the VIF.5 predictor set")
```

Table 2: Final VIFs for the VIF.5 predictor set

	Predictor	VIF
4	nox	4.542
7	dis	4.287
12	medv	3.696
11	lstat	3.540
2	indus	3.226
6	age	3.098
8	rad	2.361
5	rm	2.258
1	zn	2.193
9	ptratio	1.983
10	black	1.370
3	chas	1.084

Note: After applying a threshold of five, the `VIF.5` procedure eliminated highly correlated predictors, leaving twelve variables whose VIF values all fall below the cutoff. The remaining predictors maintain moderate independence, with the largest values belonging to *nox* (4.54) and *dis* (4.29). This version of the dataset retains strong explanatory features while keeping multicollinearity at an acceptable level.

(c) Build VIF.3 and VIF.2 sets

```r
# VIF.3
vif3_res <- stepwise_vif(X, threshold = 3)
VIF.3 <- vif3_res$kept
kable(data.frame(Predictor = names(vif3_res$final_vif),
                           VIF = as.numeric(vif3_res$final_vif))[order(-vif3_res$final_vif), ],
            digits = 3,
            caption = "Final VIFs for the VIF.3 predictor set")
```

Table 3: Final VIFs for the VIF.3 predictor set

	Predictor	VIF
9	medv	2.714
2	indus	2.501
4	rm	2.071
5	age	2.061
6	rad	1.985
1	zn	1.747
7	ptratio	1.669
8	black	1.354
3	chas	1.080

Note: Tightening the threshold to three further reduces the set of predictors. The resulting `VIF.3` model contains nine variables, all of which exhibit low to moderate collinearity. The largest VIF belongs to *medv* (2.71), followed by *indus* (2.50). These results suggest that most redundant relationships have been removed, providing a cleaner feature space for modeling.

```r
# VIF.2
vif2_res <- stepwise_vif(X, threshold = 2)
VIF.2 <- vif2_res$kept
kable(data.frame(Predictor = names(vif2_res$final_vif),
                           VIF = as.numeric(vif2_res$final_vif))[order(-vif2_res$final_vif), ],
            digits = 3,
            caption = "Final VIFs for the VIF.2 predictor set")
```

Table 4: Final VIFs for the VIF.2 predictor set

	Predictor	VIF
4	age	1.742
5	rad	1.739
1	zn	1.674
6	ptratio	1.529
7	black	1.266
3	rm	1.208
2	chas	1.043

Note: Using an even stricter threshold of two produces the smallest and most conservative predictor set, denoted `VIF.2`. All remaining predictors have VIF values below 1.75, indicating minimal overlap among features. While this greatly reduces multicollinearity, it may also remove some explanatory power compared with previous sets, emphasizing the trade-off between simplicity and completeness.

(d) 10×10 repeated CV with logistic regression (above/below median crime)

```r
# Prepare response variable
data("Boston")
y_fac <- factor(ifelse(Boston$crim > median(Boston$crim), "High", "Low"))
y_bin <- as.integer(y_fac == "High")  # binary 1/0 response

# Predictor sets
pred_sets <- list(
  Full  = setdiff(names(Boston), "crim"),
  VIF.5 = VIF.5,
  VIF.3 = VIF.3,
  VIF.2 = VIF.2
)
```

Note: The response is now encoded as a binary factor indicating above/below median crime, and four predictor sets (Full, VIF.5, VIF.3, VIF.2) are staged for comparison.

```r
# Cross-validation setup
set.seed(split_seed)
repeats = 10
K = 10

# Storage
cv_results <- data.frame(
  Model = character(),
  MeanAccuracy = numeric(),
  SDAccuracy = numeric(),
  MeanError = numeric()
)

# Repeated 10-fold CV loop
for (m in names(pred_sets)) {
  acc <- numeric()
  for (r in 1:repeats) {
    folds <- createFolds(y_fac, k = K)
    for (i in 1:K) {
      test_idx <- folds[[i]]
      train_idx <- setdiff(seq_len(nrow(Boston)), test_idx)

      TrainX <- Boston[train_idx, pred_sets[[m]], drop = FALSE]
      TestX  <- Boston[test_idx,  pred_sets[[m]], drop = FALSE]

      model <- glm(y_bin[train_idx] ~ ., family = binomial(), data = TrainX)
      pred  <- predict(model, newdata = TestX, type = "response")
      yhat  <- ifelse(pred > 0.5, 1, 0)

      acc <- c(acc, mean(yhat == y_bin[test_idx]))
    }
  }
  cv_results <- rbind(
    cv_results,
    data.frame(
      Model = m,
      MeanAccuracy = mean(acc),
      SDAccuracy = sd(acc),
      MeanError = 1 - mean(acc)
    )
  )
}

kable(cv_results, caption = "10×10 Repeated CV Logistic Regression Results")
```

Table 5: 10×10 Repeated CV Logistic Regression Results

Model	MeanAccuracy	SDAccuracy	MeanError
Full	0.9047544	0.0365996	0.0952456
VIF.5	0.8998003	0.0409280	0.1001997
VIF.3	0.8272576	0.0549891	0.1727424
VIF.2	0.8353068	0.0543797	0.1646932

Note: Repeated 10-fold CV yields a mean accuracy of 90.5% for the full predictor set, with the VIF.5 model close behind at 90.0%. The aggressively filtered VIF.3 and VIF.2 sets fall to roughly 82.7% and 83.5%, highlighting the trade-off between multicollinearity control and predictive power. These top-line results are about 4–5 percentage points better than the 85.9% accuracy I reported for logistic regression on Boston in Assignment 5 (based on repeated 66/34 splits with four predictors), whereas the pared-down VIF.3/VIF.2 sets slide back toward that earlier baseline.

Exercise 4 — Regularized Regression on College Data

(a) Load data and remove variables

Note: We confirm the 18-variable structure of the data; we remove Accept, Enroll, Top10perc, and Top25perc before modeling to focus on the remaining predictors.

```r
 college_clean <- College %>%
    as_tibble(rownames = "College") %>%
    dplyr::select(-College, -Accept, -Enroll, -Top10perc, -Top25perc)

college_overview <- college_clean %>%
  summarise(across(where(is.numeric), list(mean = mean, sd = sd))) %>%
  pivot_longer(everything(),
                      names_to = c("Variable", ".value"),
                      names_sep = "_")

kable(college_overview, digits = 1,
      caption = "Selected predictors: mean and standard deviation")
```

Table 6: Selected predictors: mean and standard deviation

Variable	mean	sd
Apps	3001.6	3870.2
F.Undergrad	3699.9	4850.4
P.Undergrad	855.3	1522.4
Outstate	10440.7	4023.0
Room.Board	4357.5	1096.7
Books	549.4	165.1
Personal	1340.6	677.1
PhD	72.7	16.3
Terminal	79.7	14.7
S.F.Ratio	14.1	4.0
perc.alumni	22.7	12.4
Expend	9660.2	5221.8
Grad.Rate	65.5	17.2

Note: Apps, Outstate, and F.Undergrad dominate in both mean and spread, while variables such as Expend and Grad.Rate operate on tighter scales—an indication that regularization will help balance coefficients across these disparate magnitudes.

(b) Two-thirds / one-third split

```
set.seed(split_seed)

  n <- nrow(College)
  n_train <- floor(n * 0.66)
  index <- sample(1:n, n_train)

  college_train <- college_clean %>% slice(index)
  college_test  <- college_clean %>% slice(-index)

  data.frame(Set = c("Training", "Test"),
             Observations = c(nrow(college_train), nrow(college_test))) %>%
    kable(caption = "Train/test split sizes")
```

Table 7: Train/test split sizes

Set	Observations
Training	512
Test	265

Note: The 66/34 split gives 512 training observations and 265 held out for testing, providing ample data for both model fitting and validation.

(c) Hybrid backward stepwise model

```
step_full <- lm(Apps ~ ., data = college_train)
  step_model <- step(step_full, k = log(n), direction = "both", trace = 0)
```

Note: Hybrid (BIC) stepwise retains six predictors—F.Undergrad, P.Undergrad, Room.Board, perc.alumni, Expend, and Grad.Rate—with an adjusted R^2 of 0.805, indicating a strong yet interpretable baseline model.

```
step_pred <- predict(step_model, newdata = college_test)
step_rmse <- RMSE(step_pred, college_test$Apps)

  kable(data.frame(Model = "Hybrid stepwise", Test_RMSE = step_rmse),
        digits = 2, caption = "Hybrid stepwise test RMSE")
```

Table 8: Hybrid stepwise test RMSE

Model	Test_RMSE
Hybrid stepwise	2607.94

Note: The stepwise model delivers a test RMSE of about 2608 applications, which will serve as the baseline when comparing penalized regressions.

(d) Design matrices for glmnet

```
x_train <- model.matrix(Apps ~ ., data = college_train)[, -1]
x_test  <- model.matrix(Apps ~ ., data = college_test)[, -1]
y_train <- college_train$Apps
y_test  <- college_test$Apps

data.frame(Set = c("Training", "Test"),
           Rows = c(nrow(x_train), nrow(x_test)),
           Columns = c(ncol(x_train), ncol(x_test))) %>%
  kable(caption = "Design-matrix dimensions")
```

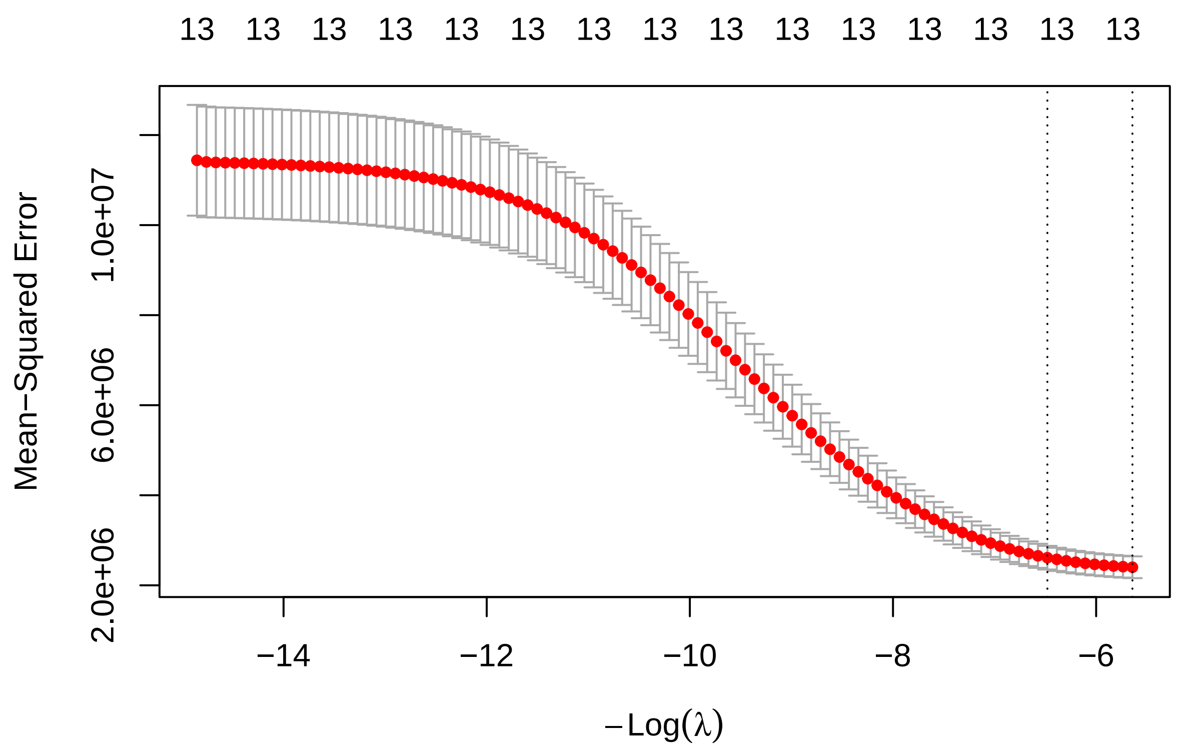
Table 9: Design-matrix dimensions

Set	Rows	Columns
Training	512	13
Test	265	13

Note: Both training and test design matrices share 13 predictors with 512 and 265 observations respectively, confirming they align with the split used above.

(e) Ridge regression (alpha = 0)

```
ridge_cv <- cv.glmnet(x_train, y_train, alpha = 0, nfolds = 10, standardize = TRUE)
plot(ridge_cv)
```



```r
ridge_lambda_min <- ridge_cv$lambda.min
ridge_pred <- as.vector(predict(ridge_cv, s = "lambda.min", newx = x_test))
ridge_rmse <- RMSE(ridge_pred, y_test)
ridge_coef <- as.matrix(coef(ridge_cv, s = "lambda.min"))

kable(data.frame(lambda_min = ridge_lambda_min,
                 Test_RMSE = ridge_rmse),
      digits = 3, caption = "Ridge performance at lambda_min")
```

Table 10: Ridge performance at lambda_min

lambda_min	Test_RMSE
282.207	2648.632

```r
ridge_coef[order(abs(ridge_coef[, 1]), decreasing = TRUE), , drop = FALSE][1:10, , drop = FALSE] %>%
  kable(digits = 4, col.names = "Coefficient",
        caption = "Top 10 ridge coefficients (absolute magnitude)")
```

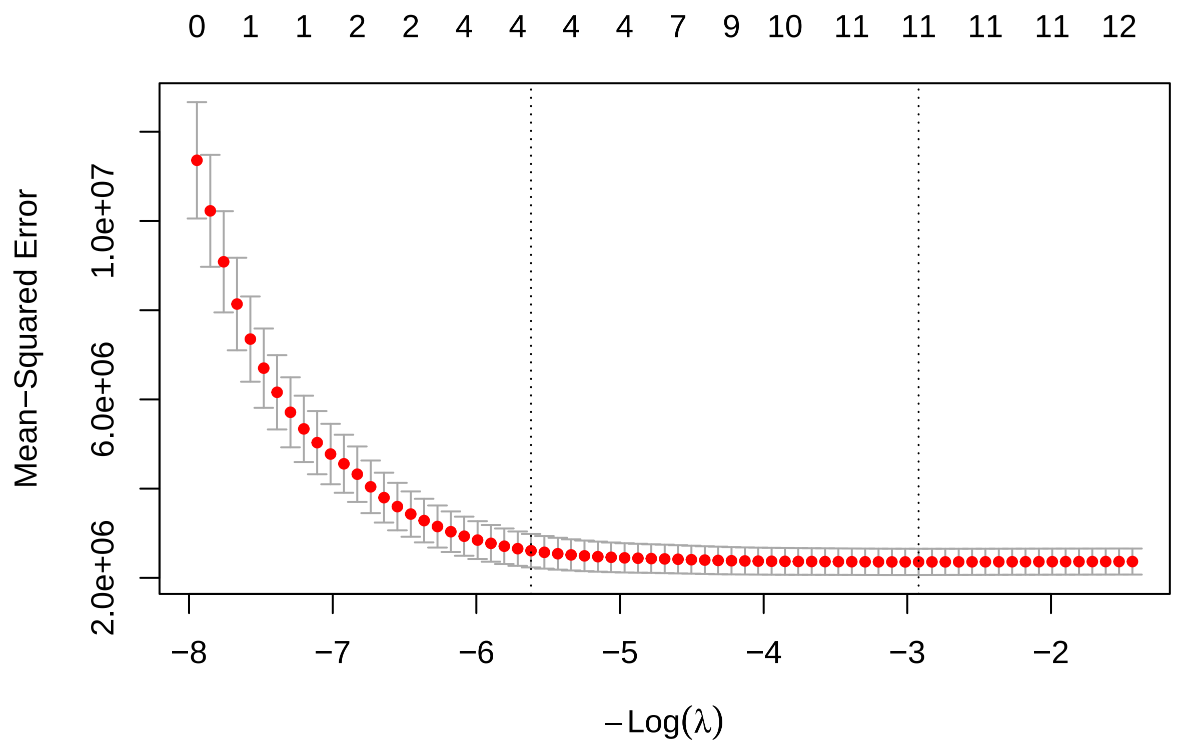
Table 11: Top 10 ridge coefficients (absolute magnitude)

	Coefficient
(Intercept)	-2499.2326
PrivateYes	-923.6406
Grad.Rate	20.8844
perc.alumni	-16.2725
S.F.Ratio	5.3640
Terminal	-3.7155
PhD	2.8897
F.Undergrad	0.5418
Books	0.5177
Room.Board	0.3030

Note: Ridge regression selects $\lambda_{\min} \approx 282.2$, producing a test RMSE of 2649 with the largest shrunken coefficients on `PrivateYes`, `Grad.Rate`, and `perc.alumni`, illustrating how shrinkage tempers the extreme baseline estimates.

(f) LASSO regression (alpha = 1)

```r
lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1, nfolds = 10, standardize = TRUE)
plot(lasso_cv)
```



```r
lasso_lambda_min <- lasso_cv$lambda.min
lasso_pred <- as.vector(predict(lasso_cv, s = "lambda.min", newx = x_test))
lasso_rmse <- RMSE(lasso_pred, y_test)
lasso_coef <- as.matrix(coef(lasso_cv, s = "lambda.min"))

kable(data.frame(lambda_min = lasso_lambda_min,
                 Test_RMSE = lasso_rmse),
      digits = 3, caption = "LASSO performance at lambda_min")
```

Table 12: LASSO performance at lambda_min

lambda_min	Test_RMSE
18.567	2604.957

```r
lasso_coef[abs(lasso_coef[, 1]) > 0, , drop = FALSE] %>%
  kable(digits = 4, col.names = "Coefficient",
        caption = "Non-zero LASSO coefficients at lambda_min")
```

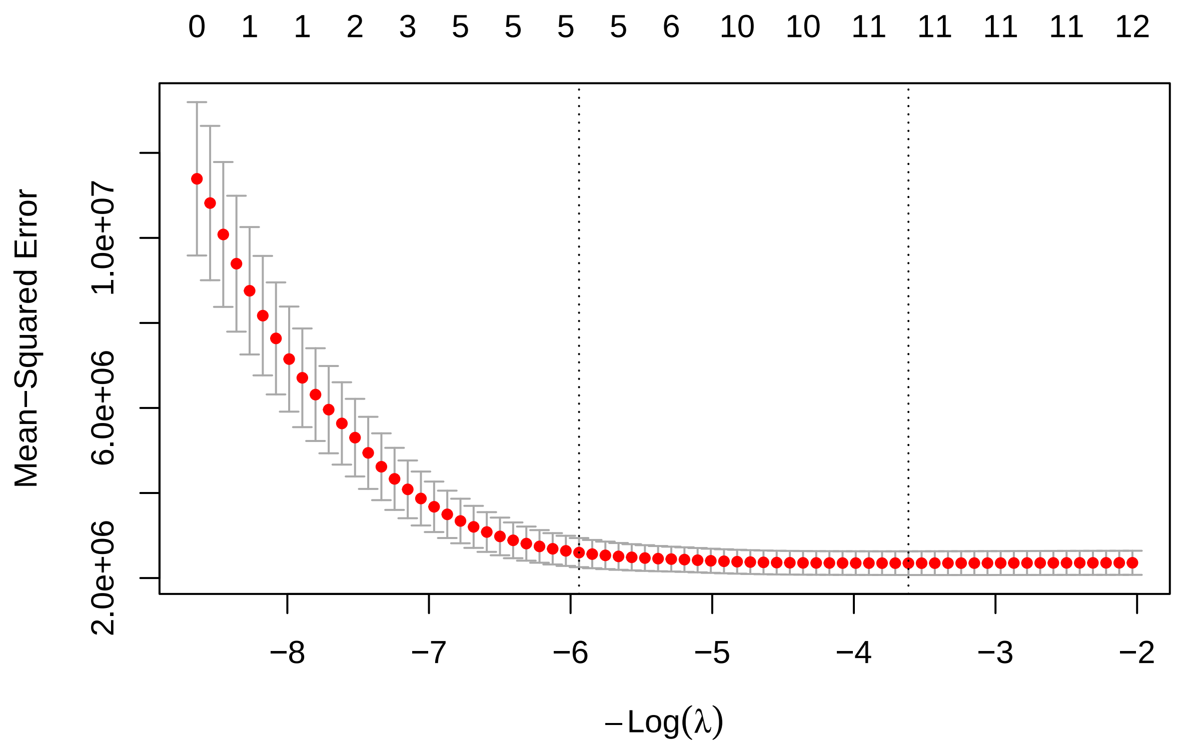
Table 13: Non-zero LASSO coefficients at lambda_min

	Coefficient
(Intercept)	-2339.1781
PrivateYes	-499.1755
F.Undergrad	0.6317
P.Undergrad	-0.1373
Outstate	0.0383
Room.Board	0.3087
Books	0.3302
Personal	-0.1529
Terminal	-4.7943
perc.alumni	-14.6336
Expend	0.1259
Grad.Rate	18.5647

Note: LASSO with $\lambda_{\text{min}} \approx 18.6$ edges out ridge, lowering the test RMSE to roughly 2605 while shrinking the field to a dozen active predictors (e.g., `PrivateYes`, `F.Undergrad`, `Outstate`) and driving the remaining coefficients precisely to zero.

(g) Elastic-net (alpha = 0.5)

```r
elastic_cv <- cv.glmnet(x_train, y_train, alpha = 0.5, nfolds = 10, standardize = TRUE)
plot(elastic_cv)
```



```r
elastic_lambda_min <- elastic_cv$lambda.min
elastic_pred <- as.vector(predict(elastic_cv, s = "lambda.min", newx = x_test))
elastic_rmse <- RMSE(elastic_pred, y_test)
elastic_coef <- as.matrix(coef(elastic_cv, s = "lambda.min"))

kable(data.frame(lambda_min = elastic_lambda_min,
                 Test_RMSE = elastic_rmse),
      digits = 3, caption = "Elastic-net performance at lambda_min")
```

Table 14: Elastic-net performance at lambda_min

lambda_min	Test_RMSE
37.135	2607.135

```r
elastic_coef[order(abs(elastic_coef[, 1]), decreasing = TRUE), , drop = FALSE][1:10, , drop = FALSE] %>
  kable(digits = 4, col.names = "Coefficient",
        caption = "Top 10 elastic-net coefficients (absolute magnitude)")
```

Table 15: Top 10 elastic-net coefficients (absolute magnitude)

	Coefficient
(Intercept)	-2353.5030
PrivateYes	-531.0151
Grad.Rate	18.7531
perc.alumni	-14.6361
Terminal	-4.2779
F.Undergrad	0.6240
Books	0.3339
Room.Board	0.3071
Personal	-0.1457
P.Undergrad	-0.1295

Note: The elastic-net compromise at $\lambda_{\text{min}} \approx 37.1$ balances ridge and LASSO behavior, yielding a test RMSE near 2607 while preserving moderate shrinkage and sparsity across the coefficient profile.

(h) Coefficient comparison

```r
step_vec <- coef(step_model)

ridge_vec <- ridge_coef[, 1]
names(ridge_vec) <- rownames(ridge_coef)

lasso_vec <- lasso_coef[, 1]
names(lasso_vec) <- rownames(lasso_coef)

elastic_vec <- elastic_coef[, 1]
names(elastic_vec) <- rownames(elastic_coef)

coef_terms <- sort(unique(c(names(step_vec),
                            names(ridge_vec),
                            names(lasso_vec),
                            names(elastic_vec))))

pull_coef <- function(coef_vec, terms) {
  out <- numeric(length(terms))
  names(out) <- terms
  common <- intersect(terms, names(coef_vec))
  out[common] <- coef_vec[common]
  out
}

coef_compare <- tibble(
  Term = coef_terms,
  Hybrid_Stepwise = pull_coef(step_vec, coef_terms),
  Ridge = pull_coef(ridge_vec, coef_terms),
  LASSO = pull_coef(lasso_vec, coef_terms),
  Elastic_Net = pull_coef(elastic_vec, coef_terms)
) %>%
  filter(if_any(-Term, ~ abs(.x) > 1e-6))

kable(coef_compare, digits = 3,
      caption = "Coefficient comparison across modeling approaches")
```

Table 16: Coefficient comparison across modeling approaches

Term	Hybrid_Stepwise	Ridge	LASSO	Elastic_Net
(Intercept)	-2960.091	-2499.233	-2339.178	-2353.503
Books	0.000	0.518	0.330	0.334
Expend	0.132	0.121	0.126	0.125
F.Undergrad	0.654	0.542	0.632	0.624
Grad.Rate	20.460	20.884	18.565	18.753
Outstate	0.000	0.052	0.038	0.039
P.Undergrad	-0.166	-0.063	-0.137	-0.129
perc.alumni	-18.062	-16.272	-14.634	-14.636
Personal	0.000	-0.101	-0.153	-0.146
PhD	0.000	2.890	0.000	0.000
PrivateYes	0.000	-923.641	-499.176	-531.015
Room.Board	0.331	0.303	0.309	0.307

Term	Hybrid_Stepwise	Ridge	LASSO	Elastic_Net
S.F.Ratio	0.000	5.364	0.000	0.000
Terminal	0.000	-3.716	-4.794	-4.278

Note: Comparing coefficients shows ridge shrinking everything toward zero, LASSO eliminating predictors such as `PhD` and `S.F.Ratio` entirely, and elastic-net occupying the middle ground—useful context for interpreting feature influence stability.

(i) Test-set RMSE comparison

```r
rmse_compare <- tibble(
  Model = c("Hybrid stepwise",
            "Ridge",
            "LASSO",
            "Elastic-net alpha = 0.5"),
  Test_RMSE = c(step_rmse, ridge_rmse, lasso_rmse, elastic_rmse)
)

kable(rmse_compare, digits = 2,
      caption = "Single-split test RMSE across models")
```

Table 17: Single-split test RMSE across models

Model	Test_RMSE
Hybrid stepwise	2607.94
Ridge	2648.63
LASSO	2604.96
Elastic-net alpha = 0.5	2607.13

Note: RMSE values cluster between 2605 and 2649, with LASSO slightly best on this split; the differences are small enough that repeated resampling would be needed before declaring a clear winner.

(j) Discuss what is happening

Response: Setting $\lambda = 0$ collapses ridge, LASSO, and elastic-net estimators back to the unpenalized OLS fit. As λ grows, coefficients shrink toward zero; ridge dampens them smoothly, LASSO introduces sparsity once the penalty exceeds a variable's threshold, and elastic-net blends both effects. In the limit $\lambda \to \infty$, every penalty drives the fitted coefficients to zero, leaving only an intercept that predicts the mean response.