Regularization in Practice: Ames Housing

Gov 51: Data Analysis and Politics

Scott Cunningham

Harvard University

Week 10 Thursday
April 2, 2026

Where We Left Off

Tuesday: three penalized regressions that control variance

Method	Penalty	Key behavior
Ridge	$\lambda \sum \beta_j^2$	Shrinks all coefficients; keeps all variables
LASSO	$\lambda \sum \lvert\beta_j\rvert$	Shrinks and zeroes out; automatic variable selection
Elastic Net	$\lambda\left[\alpha\lvert\beta\rvert + (1-\alpha)\beta^2\right]$	Blend: selection + grouping

λ controls how hard the penalty pushes. Larger λ = more shrinkage.

Cross-validation scores each λ honestly — then we pick the best

The inner loop (one λ):

1. Fix λ
2. Do k-fold: rotate through held-out groups
3. Average the k held-out RMSEs
4. Result: one CV-RMSE for that λ

The outer loop (all λ):

1. Repeat the inner loop for 100 candidate λ values
2. Plot CV-RMSE vs $\log(\lambda)$
3. Pick λ_{min}: lowest error
4. Or λ_{1se}: simpler, nearly as good

`cv.glmnet()` does all of this in one call

Before `cv.glmnet`: implement CV by hand so you know what it's doing

```r
set.seed(51)
folds    <- sample(rep(1:5, length.out = nrow(train)))
rmse_cv <- numeric(5)

for (k in 1:5) {
  train_k <- train[folds != k, ]
  test_k  <- train[folds == k, ]
  mod_k   <- lm(Sale_Price ~ ., data = train_k[, key_vars])
  pred_k  <- predict(mod_k, newdata = test_k)
  rmse_cv[k] <- sqrt(mean((test_k$Sale_Price - pred_k)^2))
}

mean(rmse_cv)    # average across 5 folds
```

This is the inner loop — for one fixed λ. `cv.glmnet` wraps this in an outer loop over 100 λ values.

The outer loop: `cv.glmnet` runs the inner loop for every λ

```
# What cv.glmnet() does internally (pseudocode):
lambda_grid <- exp(seq(log(0.01), log(1000), length.out =
    100))
cv_rmse     <- numeric(length(lambda_grid))

for (j in seq_along(lambda_grid)) {
  # inner_loop_rmse: run 5-fold CV at this fixed lambda
  cv_rmse[j] <- inner_loop_rmse(X_train, y_train, lambda =
      lambda_grid[j])
}

lambda_min <- lambda_grid[which.min(cv_rmse)]  # winner:
    bottom of the U
```

Inner loop scores one $\lambda \rightarrow$ one CV-RMSE. Outer loop repeats 100 times $\rightarrow$ the U-curve. Pick the bottom.

Why is the CV estimate more reliable than a single train/test split?

Single split:

- One held-out set
- RMSE depends on which 20% happened to land in test
- High variance across random seeds

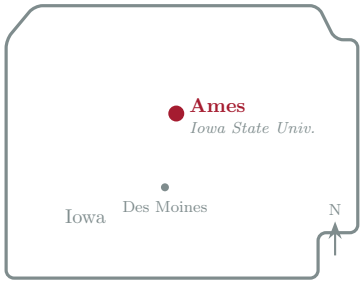
5-fold CV:

- Every observation is held out once
- RMSE averaged across 5 estimates
- Much lower variance

CV uses the data more efficiently —
and that's why we trust it to pick λ

The Ames Housing Data

Ames, Iowa: a mid-sized college town in the middle of the country



Ames in context:

- Population $\approx$ 66,000
- Home of Iowa State University
- 30 miles north of Des Moines
- Stable, Midwestern housing market
- Not coastal — no tech boom, no speculation

A good teaching dataset because it's *normal*: prices mostly reflect bedrooms, quality, location

The Ames housing data: a deliberate replacement for Boston

The problem with Boston:

- Harrison & Rubinfeld (1978) — classic ML benchmark
- Included `B`: a variable measuring racial composition of neighborhoods
- Used to predict home values — encoding discrimination as a "predictor"
- Withdrawn from `scikit-learn` in 2020 for this reason

De Cock (2011):

- Ames assessor's office data, 2006–2010
- 2,930 sales, 80 features
- No racially-coded variables
- Richer feature set than Boston
- Now the standard Kaggle benchmark

If you Google "house price prediction ML" — this is the dataset you'll find

2,930 houses in Ames, Iowa — one outcome, 80 features

The outcome:

`Sale_Price` — final sale price ($)

A few predictors:

- `Gr_Liv_Area` — living area (sq ft)
- `Overall_Qual` — quality rating (1–10)
- `Year_Built` — construction year
- `Neighborhood` — 28 neighborhoods
- `Garage_Area` — garage size (sq ft)

The question:

Can regularization beat well-specified OLS when features multiply?

The experiment:

1. OLS on curated predictors
2. OLS with all pairwise interactions
3. Ridge, LASSO, Elastic Net on the same matrix

Load the data and look at it first

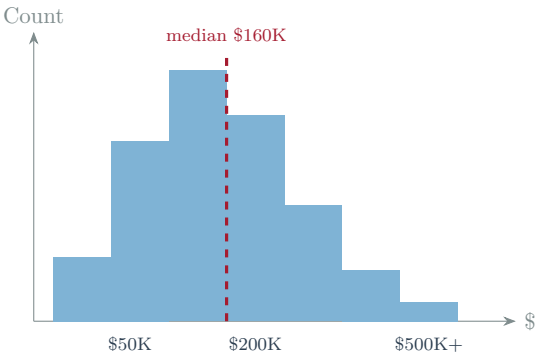
```
library(tidyverse)
library(glmnet)
library(AmesHousing)

ames <- make_ames() %>%
  mutate(Overall_Qual = as.numeric(Overall_Qual),
         Overall_Cond = as.numeric(Overall_Cond))

dim(ames)                          # 2930 x 81
median(ames$Sale_Price)            # roughly $160,000
summary(ames$Sale_Price)
```

Before touching a model: know your outcome. What is the range? Any outliers?

Sale price has a long right tail — a few houses pull the mean far above the median



What this means:

- Most houses cluster near the median
- A few expensive homes stretch the upper tail
- RMSE will be sensitive to those outliers

We predict on raw dollars — not log-transformed. Our RMSE is in dollars.

Step 0: Train/Test Split

Hold out 20% before building any model — the test set is sacred

```
set.seed(51)
n         <- nrow(ames)
train_idx <- sample(1:n, size = floor(0.8 * n))
train     <- ames[train_idx, ]
test      <- ames[-train_idx, ]

cat("Training:", nrow(train), " Test:", nrow(test))
# Training: 2344  Test: 586
```

- Once we split, the test set does not exist for modeling purposes
- We fit **everything** — including cross-validation — on training data only
- Test set appears exactly once: at the end, to report honest RMSE

Step 1: Simple OLS Baseline

Simple OLS on 16 hand-picked predictors

```
key_vars <- c("Sale_Price", "Gr_Liv_Area", "Lot_Area",
              "Year_Built", "Overall_Qual", "Overall_Cond",
              "Total_Bsmt_SF", "Garage_Area", "Full_Bath",
              "Bedroom_AbvGr", "TotRms_AbvGrd", "Fireplaces",
              "Year_Remod_Add", "Neighborhood",
              "House_Style", "Bldg_Type")

train_s  <- train[, key_vars]
test_s   <- test[,  key_vars]
ols_simple <- lm(Sale_Price ~ ., data = train_s)

rmse <- function(actual, pred) sqrt(mean((actual - pred)^2))

cat("Train RMSE:", rmse(train_s$Sale_Price,
                        predict(ols_simple, train_s)))
cat("Test RMSE: ", rmse(test_s$Sale_Price,
                        predict(ols_simple, test_s)))
```

Simple OLS generalizes well — the gap between train and test is modest

Model	Train RMSE	Test RMSE
OLS (16 predictors)	∼\$31,000	∼\$35,000

- Small model, curated predictors, low variance
- Gap ≈ \$4K: OLS generalizes reasonably here
- This is our **baseline** — any model that does worse is a failure

The question: can a more ambitious model do *better*? Or will it overfit?

Step 2: Deliberate Overfitting

The kitchen sink: every pairwise interaction — 666 features

The code builds the formula from all numeric predictors:

```
f_full <- Sale_Price ~ (x1 + x2 + ...  + x34)^2
```

`model.matrix(f_full, data=train)` then expands this formula into columns:

- 34 main effects
- $\binom{34}{2} = 561$ pairwise interactions
- Additional dummy columns for categorical variables (e.g., neighborhood levels)

Result: 666 numeric columns.
`model.matrix` did all the expansion.

$n/p \approx 3.5$: 2,344 observations, 666 features. OLS has almost no room to maneuver.

`model.matrix()` turns the formula into the numeric matrix `glmnet` needs

```
num_preds <- names(ames)[sapply(ames, is.numeric)]
num_preds <- num_preds[num_preds != "Sale_Price"]
# drop zero-variance columns
good_preds <- num_preds[sapply(train[, num_preds],
                               var, na.rm=TRUE) > 0]

f_full  <- as.formula(
  paste("Sale_Price ~ (",
        paste(good_preds, collapse=" + "), ")^2"))

X_train <- model.matrix(f_full, data=train)[, -1]
X_test  <- model.matrix(f_full, data=test)[, -1]

# align columns (test must match train)
common  <- intersect(colnames(X_train), colnames(X_test))
X_train <- X_train[, common]
X_test  <- X_test[,  common]

cat("Features:", ncol(X_train), " n/p:", round(nrow(X_train)/ncol(X_train),1))
```

OLS on the kitchen sink: watch it collapse

```
y_train <- train$Sale_Price
y_test  <- test$Sale_Price

ols_full <- lm(f_full, data=train)

full_train <- rmse(y_train, predict(ols_full, train))
full_test  <- rmse(y_test,  predict(ols_full, test))

cat("Kitchen sink train RMSE:", full_train)
cat("Kitchen sink test  RMSE:", full_test)
```

Before you run it — what do you predict will happen to the gap?

Kitchen sink OLS: in-sample looks fine — out-of-sample is catastrophic

Model	Train RMSE	Test RMSE	Gap
OLS (16 predictors)	∼$31,000	∼$35,000	modest
OLS (666 kitchen sink)	∼$3,000	∼$951,000	**catastrophic**

Train RMSE dropped to $3K — OLS memorized the training data

Test RMSE exploded to $951K — predictions are useless on new houses

This is textbook overfitting. The model is fitting noise, not signal.

Method 1: Ridge Regression

Defining our terms before the pictures

- **RSS contour** (or "equal-RSS curve")

  All coefficient vectors (β_1, β_2) that produce the *same* residual sum of squares for a given dataset. Like a topographic contour line, but for prediction error instead of elevation. We will just call these **contours**.

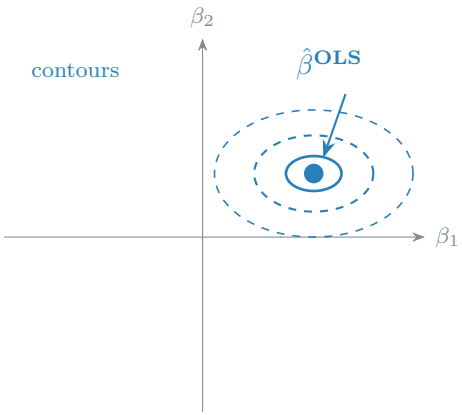
- **Constraint region**

  The set of coefficient vectors that satisfy the penalty budget: $\beta_1^2 + \beta_2^2 \leq t$ for Ridge, $|\beta_1| + |\beta_2| \leq t$ for LASSO. Drawn as a circle or diamond centered at the origin.

- **Solution**

  The coefficient vector where the smallest (innermost) contour just touches the constraint region. OLS has no constraint, so its solution is the center of all the contours.

OLS first: the unconstrained solution



Each ellipse = all (β_1, β_2) with the *same RSS* in this sample

- Inner: low RSS Outer: high RSS
- $\hat{\beta}^{\text{OLS}}$ sits at center: this sample's minimum

With many features, the center drifts:

- Different sample → center shifts
- Can move far from origin
- That instability is variance

The general form: every penalized regression solves the same trade-off

$$\min_{\beta} \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{RSS}} + \lambda \cdot \underbrace{P(\beta)}_{\text{penalty}}$$

- $\lambda = 0$: no penalty $\rightarrow$ OLS
- λ large: penalty dominates $\rightarrow$ coefficients shrink toward zero
- **The three methods differ only in how $P(\beta)$ is defined**

λ is not solved by the FOCs — it is a hyperparameter chosen by cross-validation (more on this later)

Ridge: $P(\beta) = \sum \beta_j^2$ — penalize the squared size of each coefficient

Penalty form:

$$\min_{\beta} \ \text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Equivalent constraint form (via Lagrange duality):

$$\min_{\beta} \ \text{RSS} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq t$$

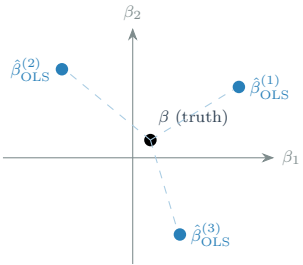
- t is a **budget** on the total squared size of all coefficients
- Small $t \leftrightarrow$ large λ: tight budget = heavy penalty
- Every t maps to exactly one λ and vice versa

Before the geometry: three things to be clear about

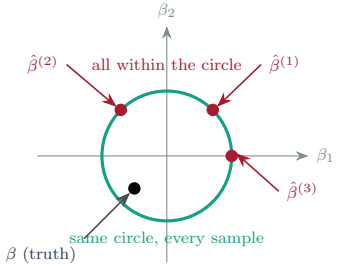
- **The circle is a budget on size — centered at the origin, not an estimand**
  $\{\beta : \beta_1^2 + \beta_2^2 \leq t\}$ — zero is just the cheapest point
- **Ridge has its own (biased) target:** $E[\hat{\beta}^{\mathbf{Ridge}}] = (X'X + \lambda I)^{-1} X'X\beta$
  Shrunken version of β, not β itself — wrong on purpose to cut variance
- **Variance falls because the circle is fixed across samples**
  OLS solutions scatter; Ridge solutions must land in the same bounded region

Why Ridge reduces variance: OLS scatters, Ridge clusters

OLS across three samples:

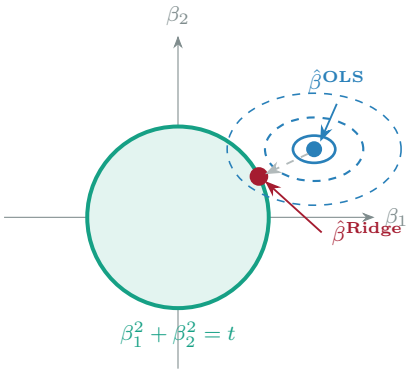


Ridge across the same three samples:



Ridge is biased (estimates miss β)
but stable (they cluster together)

Ridge geometry: the constraint is a circle — OLS solution is pulled to its surface



How to read this:

- Ellipses = equal-RSS contours; center = $\hat{\beta}^{\text{OLS}}$
- Circle = Ridge budget $\beta_1^2 + \beta_2^2 \leq t$
- $\hat{\beta}^{\text{OLS}}$ lies outside the circle
- Ridge solution = innermost ellipse touching the circle
- Never touches an axis → **no zeros**

No corners ⇒ smooth tangency ⇒ no exact zeros.

The Ridge Lagrangean: λ is the shadow price of the budget

Lagrangean:

$$\mathcal{L} = \text{RSS} + \lambda \left(\sum_{j=1}^{p} \beta_j^2 - t \right)$$

FOC for β_j (in simple orthogonal case):

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = -2(y_j - \beta_j) + 2\lambda\beta_j = 0 \quad \Rightarrow \quad \beta_j^* = \frac{\hat{\beta}_j^{\text{OLS}}}{1+\lambda}$$

- Every coefficient is divided by the same factor $(1 + \lambda)$
- λ large $\rightarrow$ strong shrinkage toward zero, but **never exactly zero**
- λ is the shadow price of relaxing the budget t by one unit

FOCs give $\beta^*(\lambda)$ for any fixed λ. CV then chooses which λ to use.

Ridge: numerical example — two coefficients, $\lambda = 2$

OLS: $\hat{\beta}_1 = 3.0$, $\hat{\beta}_2 = 0.5$, RSS$= 100$. Ridge: divide by $(1 + \lambda) = 3$.
Which (β_1, β_2) minimizes RSS $+2(\beta_1^2 + \beta_2^2)$?

Solution	β_1	β_2	**Penalty**	**RSS+Penalty**
OLS (ignores penalty)	3.00	0.50	$2(9.00 + 0.25) = 18.5$	$100 + 18.5 = 118.5$
Ridge (min total)	1.00	0.17	$2(1.00 + 0.03) = 2.1$	$108 + 2.1 =$ **110.1**
Force $\beta_2 = 0$	1.00	0.00	$2(1.00 + 0.00) = 2.0$	$108.4 + 2.0 = 110.4$

Why OLS loses: $\beta_1 = 3$ incurs penalty $2 \times 9 = 18$. Ridge accepts higher RSS to escape it.

Why not zero β_2? Saves only 0.06 in penalty but costs 0.4 in RSS.

Ridge on the Ames data

```
set.seed(51)
cv_ridge <- cv.glmnet(X_train, y_train, alpha=0, nfolds=10)

ridge_pred  <- predict(cv_ridge, s="lambda.min", newx=X_test
    )
ridge_rmse  <- rmse(y_test, ridge_pred)
ridge_coefs <- coef(cv_ridge, s="lambda.min")

cat("Optimal lambda:", round(cv_ridge$lambda.min))
cat("Test RMSE:     ", round(ridge_rmse))     # ~ $42,000
cat("Non-zero coefs:", sum(ridge_coefs[,1] != 0) - 1)
# 666 -- Ridge never zeros anything out
```

Ridge result: variance controlled — all 666 survive, but shrunk

What happened:

- From \$951K down to \$42K
- All 666 coefficients survive
- Most are very small but non-zero
- Division by $(1 + \lambda)$ shrinks everything equally

What this costs:

- Can't tell you which variables matter
- Every interaction term has some weight
- Interpretation is difficult
- Variance controlled; bias small

Ridge is excellent for prediction. It is poor for understanding which variables drive the outcome.

Predictions in action — what did Ridge say these houses cost?

```
data.frame(actual = y_test[1:6],
           predicted = round(as.vector(ridge_pred)[1:6]))
```

House	Actual	Ridge Pred.	Residual
1	\$215,000	\$209,200	+\$5,800
2	\$140,000	\$147,500	−\$7,500
3	\$325,000	\$298,000	+\$27,000
4	\$112,500	\$119,800	−\$7,300
5	\$450,000	\$388,000	+\$62,000
6	\$178,000	\$181,200	−\$3,200

Close on typical houses — misses badly on expensive ones. Residuals grow with price.

Method 2: LASSO

LASSO: $P(\beta) = \sum |\beta_j|$ — penalize the absolute size

Penalty form:

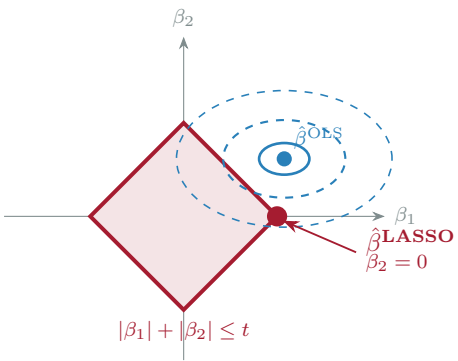
$$\min_{\beta} \ \text{RSS} + \lambda \sum_{j=1}^{p} |\beta_j|$$

Equivalent constraint form:

$$\min_{\beta} \ \text{RSS} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq t$$

- Now the budget is on total *absolute* coefficient size — the "L1 ball"
- Small t: tight budget forces some coefficients exactly to zero
- This is the key difference from Ridge

LASSO geometry: the constraint is a diamond — corners sit on the axes



The constraint is a *diamond*:

- Corners sit **on the axes**
- Expanding ellipses hit a corner first
- At a corner: $\beta_j = 0$ exactly
- **This is variable selection**

Ridge's circle has no corners.

The LASSO Lagrangean: the L1 penalty has a constant price near zero

$$\mathcal{L} = \text{RSS} + \lambda \left(\sum_{j=1}^{p} |\beta_j| - t \right)$$

FOC (soft-thresholding, orthogonal case):

$$\beta_j^* = \text{sign}(\hat{\beta}_j^{\text{OLS}}) \cdot \max\Big(|\hat{\beta}_j^{\text{OLS}}| - \lambda,\ 0 \Big)$$

- Penalty costs λ per unit of $|\beta_j|$ — same at $|\beta_j| = 5$ and $|\beta_j| = 0.01$
- Ridge: marginal penalty near zero is $2\lambda|\beta_j| \approx 0$ — easy to keep small β_j
- LASSO: marginal penalty near zero is still λ — never shrinks to zero cost
- **If $|\hat{\beta}_j^{\text{OLS}}| < \lambda$: penalty cost > fit benefit $\Rightarrow$ zero it**

LASSO: numerical example — the weak predictor is eliminated

OLS gives: $\hat{\beta}_1 = 3.0,\ \hat{\beta}_2 = 0.5 \quad \lambda = 0.8$

Soft-thresholding: $\beta_j^* = \text{sign}(\hat{\beta}_j) \cdot \max(|\hat{\beta}_j| - \lambda,\ 0)$

	$\hat{\beta}_j^{\text{OLS}}$	$\lvert\hat{\beta}_j\rvert - \lambda$	β_j^*	**Zeroed?**
β_1 (living area)	3.0	$3.0 - 0.8 = 2.2$	2.2	No
β_2 (weak interact.)	0.5	$0.5 - 0.8 = -0.3 < 0$	**0**	**Yes**

Verification: Zeroing β_2 saves $\lambda \times |\beta_2^{\text{OLS}}| = 0.8 \times 0.5 = 0.40$ in penalty.
RSS cost of dropping β_2: ≈ 0.35 (small). **Savings exceed cost — zero it.**

This is the rule: if $|\hat{\beta}_j^{\text{OLS}}| < \lambda$, the variable gets zeroed. That's LASSO variable selection.

LASSO on the Ames data

```
set.seed(51)
cv_lasso <- cv.glmnet(X_train, y_train, alpha=1, nfolds=10)

lasso_pred  <- predict(cv_lasso, s="lambda.min", newx=X_test
   )
lasso_rmse  <- rmse(y_test, lasso_pred)
lasso_coefs <- coef(cv_lasso, s="lambda.min")
n_kept      <- sum(lasso_coefs[,1] != 0) - 1

cat("Test RMSE:    ", round(lasso_rmse))  # ~ $57,000
cat("Vars kept:    ", n_kept)             # ~ 30 of 666

nonzero <- rownames(lasso_coefs)[lasso_coefs[,1] != 0]
nonzero <- nonzero[nonzero != "(Intercept)"]
top10   <- sort(abs(lasso_coefs[nonzero,1]), dec=TRUE)[1:10]
print(round(top10))
```

LASSO result: ∼30 of 666 survive — mostly interaction terms

Top survivors (by coefficient size):

- `Overall_Qual:Bsmt_Full_Bath` +816
- `Overall_Qual:Fireplaces` +702
- `Bedroom_AbvGr:Kitchen_AbvGr` +398
- `BsmtFin_SF_1:Kitchen_AbvGr` +146
- `Overall_Qual:Overall_Cond` +95

What this tells us:

- *Interactions* dominate, not main effects
- Quality matters most — but only when combined with other features
- 636 of 666 features zeroed

These are the features where $|\hat{\beta}_j^{\text{OLS}}| > \lambda$

Method 3: Elastic Net

Elastic Net: $P(\beta) = \alpha|\beta| + (1-\alpha)\beta^2$ — blend both penalties

Penalty form:

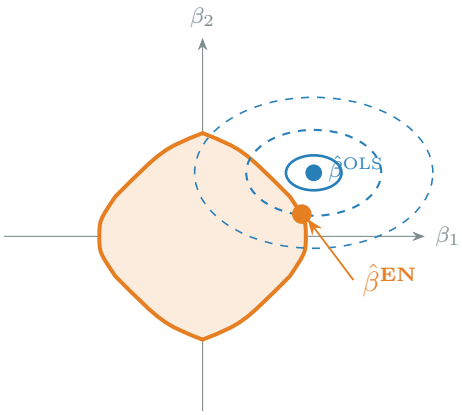
$$\min_{\beta} \ \text{RSS} + \lambda \sum_{j=1}^{p} \Big[\alpha|\beta_j| + (1-\alpha)\beta_j^2\Big]$$

Equivalent constraint form:

$$\min_{\beta} \ \text{RSS} \quad \text{s.t.} \quad \sum_{j=1}^{p} \Big[\alpha|\beta_j| + (1-\alpha)\beta_j^2\Big] \leq t$$

- $\alpha = 0$: pure Ridge (L2 only)
- $\alpha = 1$: pure LASSO (L1 only)
- $\alpha = 0.5$: equal blend — some zeroing, some proportional shrinkage

Elastic Net geometry: a rounded diamond — corners are blunted



The constraint is *between* circle and diamond:

- Corners are blunted, not sharp
- RSS ellipses *can* hit near an axis — so some zeroing happens
- But not as aggressively as LASSO

	Corners?	Zeros?
Ridge	No	Never
LASSO	Yes (sharp)	Often
EN	Soft	Sometimes

The Elastic Net Lagrangean: FOC blends soft-threshold and proportional shrinkage

Lagrangean ($\alpha = 0.5$):

$$\mathcal{L} = \text{RSS} + \lambda \left(\sum_j \left[0.5|\beta_j| + 0.5\beta_j^2 \right] - t \right)$$

FOC (orthogonal case, $\alpha = 0.5$):

$$\beta_j^* = \frac{\text{sign}(\hat{\beta}_j) \cdot \max\left(|\hat{\beta}_j| - 0.5\lambda,\ 0 \right)}{1 + 0.5\lambda}$$

- **Numerator:** soft-threshold by $\alpha\lambda$ (L1 part) — can zero coefficients
- **Denominator:** shrink by $(1 + (1 - \alpha)\lambda)$ (L2 part) — proportional shrinkage
- Result: more aggressive than LASSO on large coefficients, less aggressive on small

Elastic Net: numerical example — weak predictor barely survives

OLS: $\hat{\beta}_1 = 3.0,\ \hat{\beta}_2 = 0.5 \quad \alpha = 0.5,\ \lambda = 0.8$

Formula: $\beta_j^* = \text{sign}(\hat{\beta}_j) \cdot \max(|\hat{\beta}_j| - 0.4,\ 0)\ /\ 1.4$

	$\hat{\beta}_j^{\text{OLS}}$	**Threshold step**	**Shrink step**	β_j^*
β_1	3.0	$3.0 - 0.4 = 2.6$	$2.6/1.4$	**1.86**
β_2	0.5	$0.5 - 0.4 = 0.1$	$0.1/1.4$	**0.07** (survives!)

Method	β_1^*	β_2^*	**Zeroed?**
Ridge ($\lambda = 2$)	1.00	0.17	No
LASSO ($\lambda = 0.8$)	2.20	0.00	Yes
Elastic Net ($\alpha = 0.5$, $\lambda = 0.8$)	1.86	0.07	No (barely)

Elastic Net on the Ames data

```
set.seed(51)
cv_enet <- cv.glmnet(X_train, y_train, alpha=0.5, nfolds=10)

enet_pred  <- predict(cv_enet, s="lambda.min", newx=X_test)
enet_rmse  <- rmse(y_test, enet_pred)
enet_coefs <- coef(cv_enet, s="lambda.min")
enet_kept  <- sum(enet_coefs[,1] != 0) - 1

cat("Test RMSE:  ", round(enet_rmse))    # ~ $58,000
cat("Vars kept:  ", enet_kept)            # ~ 43
```

- Slightly more variables than LASSO (L2 part pulls correlated features in together)
- Slightly better RMSE than LASSO
- In practice: try all three, let CV tell you which wins

Reading the Results

Good variable selection beats regularization — but regularization beats none

Model	Features	Test RMSE	Regime
OLS (16 main effects)	16	∼$35,000	**Winner: expert curation**
OLS (666 kitchen sink)	666	∼$951,000	Variance explodes
Ridge	666	∼$42,000	Variance controlled
LASSO	∼30	∼$57,000	Selection + shrinkage
Elastic Net	∼43	∼$58,000	Blend

Kitchen sink failure:
27× worse than simple OLS

Regularization rescues:
$42K–$58K vs. $951K — but can't beat curated $35K

Where does our model over- and under-predict? Check by neighborhood

```
lasso_pred_train <- predict(cv_lasso, s="lambda.min",
                            newx = X_train)
train$predicted  <- as.vector(lasso_pred_train)

train %>%
  group_by(Neighborhood) %>%
  summarize(
    mean_actual  = mean(Sale_Price),
    mean_pred    = mean(predicted),
    gap          = mean_actual - mean_pred,
    n            = n()
  ) %>%
  arrange(desc(abs(gap)))
```

Same mechanics as PS3's race question: subset by group, compare prediction to actual, interpret the gap

Predictions are not equally good everywhere — some neighborhoods are systematically off

- Wealthy neighborhoods (e.g., `NridgHt`, `StoneBr`): model may underpredict — high-end features hard to capture
- Low-price neighborhoods: may overpredict — quality differences not fully encoded
- Gaps reveal what the model *doesn't know*, not just what it got wrong

A model that predicts well on average can still
be systematically wrong for particular groups

This is the same question as: does the COMPAS model predict equally well for Black and white defendants?

Why did kitchen sink OLS fail so completely?

The variance problem in numbers:

- 666 coefficients estimated from 2,344 observations
- Many interaction terms are nearly collinear
- OLS assigns large, noisy coefficients to explain quirks in the training data
- On new houses, those quirks don't repeat — predictions explode

Bias–Variance: OLS bias $= 0$, but variance $\to \infty$ as $p/n \to 1$

The penalty forces coefficients toward zero — and that *lower variance* more than compensates for the bias it introduces

What does a surviving LASSO coefficient actually mean?

Suppose the `Gr_Liv_Area` coefficient is $+18$:

- Each additional square foot of living area $\rightarrow$ +\$18 in predicted sale price
- This is **after** LASSO has zeroed out 600 noisier predictors
- It is **not** a causal estimate — it is a prediction coefficient

> Interpretation: if two houses are identical except one has 100 more sq ft of living area, our model predicts the larger house sells for ~\$1,800 more

LASSO selected this variable because it *reduced CV-RMSE* — not because of any theory about housing markets

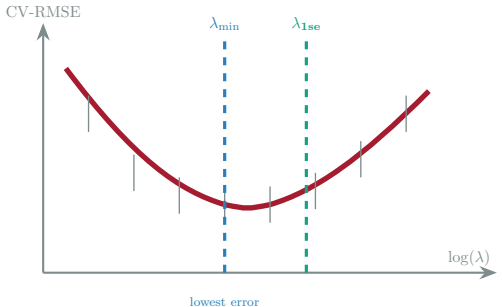
Reading the CV Curve

The CV curve for Ames LASSO — the U-shape we predicted

```
plot(cv_lasso)
```

- Left: low λ, model overfits
- Bottom: λ_{min} — best CV-RMSE
- Right: high λ, model underfits

The U-shape is the bias–variance tradeoff made visible



λ_{min} vs λ_{1se} in the Ames context

λ_{min}

- Lowest CV-RMSE
- Keeps ∼30 features
- Test RMSE ≈ \$57K
- Use this for prediction accuracy

λ_{1se}

- Within 1 SE of minimum
- Keeps fewer features
- Test RMSE ≈ \$38K
- Simpler model, nearly as good

Here: λ_{1se} costs only ∼\$1K in accuracy. For interpretability, worth it.

To generate predictions with your chosen λ:

```
final_pred <- predict(cv_lasso, s = "lambda.min", newx = X_test)
```

Swap in `"lambda.1se"` for simpler model. **PS3:** use `"lambda.min"`.

From Houses to People: Predictions with Higher Stakes

Everything we just did works on binary outcomes too — with one wrinkle

Ames: outcome is `Sale_Price` — continuous, in dollars

PS3 (COMPAS): outcome is `recidivism` — binary, 0 or 1

- The code is *identical*: same `cv.glmnet()`, same `alpha`, same `lambda.min`
- Predictions are now between 0 and 1 — interpreted as **probabilities**
- A prediction of 0.72 means: the model assigns a 72% chance of rearrest
- This is called a **Linear Probability Model** — OLS applied to a 0/1 outcome

RMSE on a 0/1 outcome is still interpretable: average distance from the true 0 or 1

Baker (2025): When Variable Selection Goes to Court

Before LASSO, the analyst picked the variables — and that invited manipulation

- Researcher chooses variables based on theory and judgment
- Different researchers → different variable sets → different results
- In courtrooms: whoever tells the most convincing story wins

After LASSO:

- Start with all candidate variables
- Let the penalty and cross-validation decide which ones stay
- The analyst's discretion is replaced by a transparent, reproducible rule

In Halliburton, two experts picked different peers — and reached opposite conclusions

	Defense Expert	**Plaintiff Expert**
Peer selection	S&P Energy + custom index	Analyst-report peers
Excess return	-2.9%	-3.7%
p-value	0.20	0.02
Conclusion	Not significant	Significant

Stakes: class certification worth billions — the entire case hinged on which peer firms you chose

LASSO applied to 29 candidate peers: all three methods agreed

Method	Excess Return	p-value
LASSO	-3.2%	≈ 0.06
Ridge	-3.3%	≈ 0.06
Elastic Net	-3.2%	≈ 0.06

- Data-driven answer falls *between* the two experts
- All three methods converge — the result is not sensitive to method choice
- Baker's argument: regularization removes the discretion that dueling experts exploit

Algorithmic Fairness: Who Bears the Cost of Prediction Errors?

COMPAS: a prediction algorithm used in real courtrooms

- Northpointe built COMPAS to predict recidivism
- Used by judges in bail and sentencing decisions across the US
- Input: criminal history, demographics, survey responses
- Output: risk score from 1–10

This is exactly the prediction problem you solved on PS3 — but with real consequences for real people

ProPublica found the algorithm's errors were racially asymmetric

Error type	Black defendants	White defendants
Falsely flagged as high risk	44.9%	23.5%
Falsely flagged as low risk	28.0%	47.7%

- Same overall accuracy ($\sim$65%) — but errors distributed unequally
- Black defendants: more likely to be wrongly labeled dangerous
- White defendants: more likely to be wrongly labeled safe

We saw the neighborhood version of this in Ames: average accuracy hides systematic errors in subgroups

Northpointe's defense reveals a mathematical impossibility

Northpointe: among defendants scored 7, the actual recidivism rate is the same across races — that's calibration

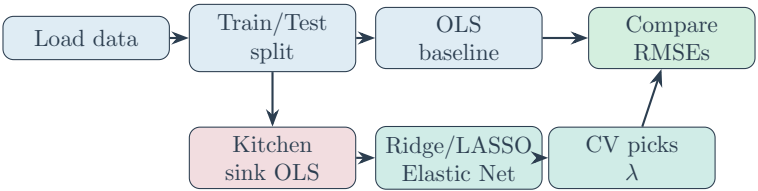
The impossibility theorem (Chouldechova, 2017):

- If base rates differ between groups...
- You **cannot** simultaneously equalize:
  1. False positive rates (ProPublica's concern)
  2. False negative rates
  3. Predictive parity (Northpointe's defense)

This is a mathematical impossibility, not a modeling failure — the tradeoff is real and someone must choose who bears the cost

What We Learned

The full pipeline in one view



This is the same workflow you applied to COMPAS on PS3 — same code, different data

The penalty found ~30 variables in 666 — mostly interactions.

The data did the variable selection.

A model that predicts well on average can still be systematically wrong for particular groups.