ELEC3544 · Fall 2026

Data Science with Foundation Models

Yang Liu

The University of Hong Kong



Regression:
Predicting Continuous Outcomes

You've prepared the data. Now let's make predictions.

From Prepared Data to Predictive Models

Module 1: Foundations
L1–L4: Math, Stats, Data Prep

Module 2: Core Methods
L5–L7: Regression, Classification, Unsupervised

Module 3: Modern Integration
L8–L10: FM Collaboration, DL, How FMs Work

L4 → L5 Connection: In L4, you learned to clean, explore, and prepare data. Now we use that prepared data to build predictive models — starting with regression.

Today's Learning Objectives

1. Explain what linear regression computes and why squared errors are the standard loss function
2. Interpret regression coefficients in simple and multiple regression; diagnose multicollinearity
3. Evaluate models using train/test splits, R^2, residual plots, and RMSE
4. Apply Ridge and Lasso regularization with geometric intuition for constraint behavior
5. Direct CTA-assisted model building: structured requests, verification, and iteration
6. Connect L4's data preparation pipeline to model performance
7. Describe the bias-variance tradeoff using the dartboard analogy

The CTA's New Role: You Direct, It Implements, You Evaluate

Before (L2–L4): Learning Partner

"Explain what a covariance matrix is"

"Why does PCA use eigenvalues?"

"Help me understand Bayes' theorem"

→

Now (L5+): Implementation Assistant

"Fit a regression with these 3 features and report R^2"

"Generate a residual plot and interpret it"

"Apply Lasso for 50 α values and plot the path"

Your Three New Responsibilities

1. Specify clearly
Tell the CTA exactly what to produce

2. Verify outputs
Check that computations and plots are correct

3. Interpret results
Decide what the results mean for your problem

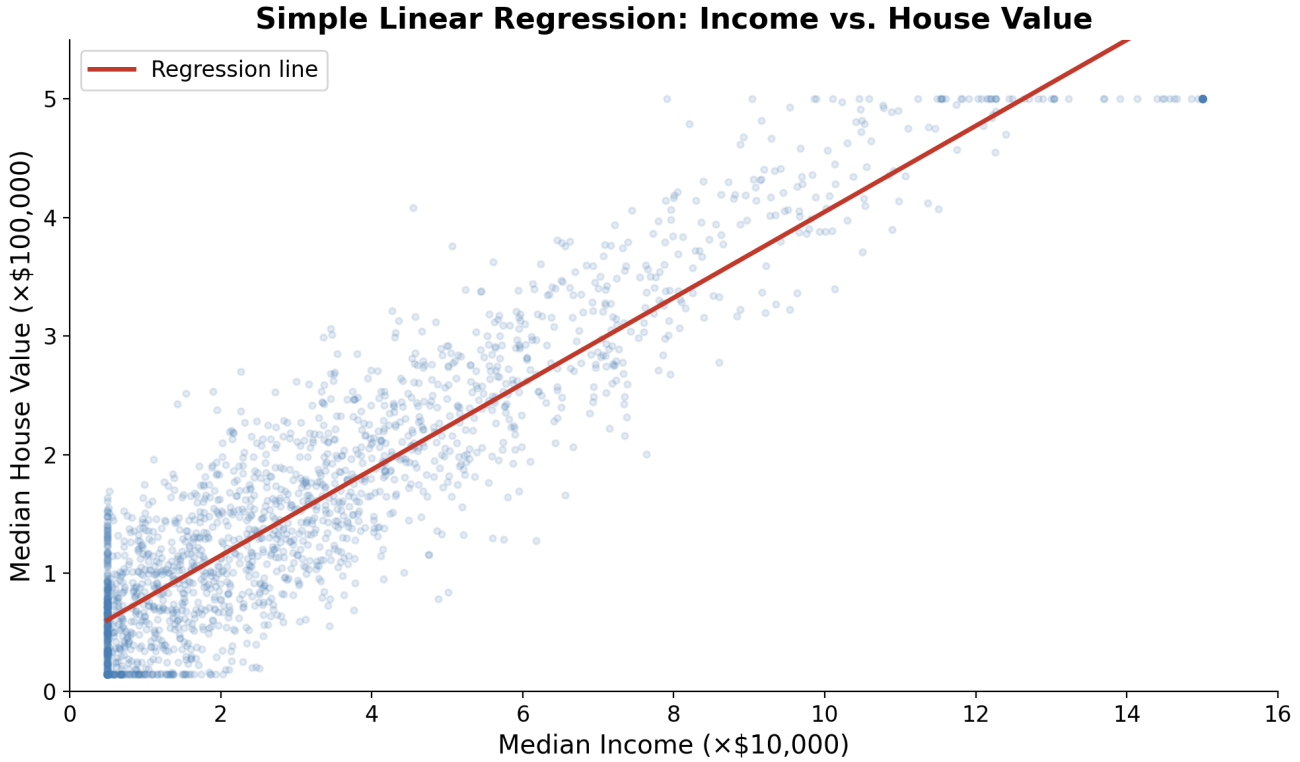
Regression Answers the Question of "How Much"

Regression predicts continuous outcomes — numbers on a scale, like price or temperature.

How much will this house sell for?	→ **Price ($)**
How many units will we sell next quarter?	→ **Quantity**
What will the temperature be tomorrow?	→ **Degrees °C**
How long until this machine needs maintenance?	→ **Hours**

Regression vs. Classification (L6 preview): Regression predicts "how much" (continuous). Classification predicts "which category" (discrete).

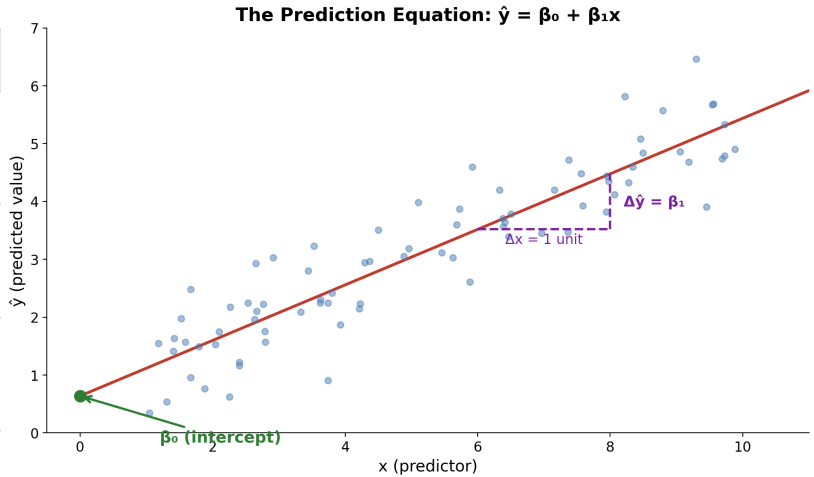
Simple Linear Regression Draws the Best-Fitting Line



The line captures the trend. But "closest" how? What makes this line the best one?

The Prediction Equation

Symbol	Meaning
y	Target variable — what we want to predict (e.g., house price)
x	Predictor — the input feature (e.g., median income)
$\hat{y}$	Predicted value — our model's estimate of y for a given x
β_0	Intercept — predicted value when $x = 0$
β_1	Slope — change in $\hat{y}$ per one-unit increase in x



Simple Linear Regression

$$\hat{y} = \beta_0 + \beta_1 x$$

Interpretation: β_1 = "for each 1-unit increase in x, $\hat{y}$ changes by β_1 units"

Residuals Measure How Far Each Prediction Misses

y_i = observed value (what actually happened)

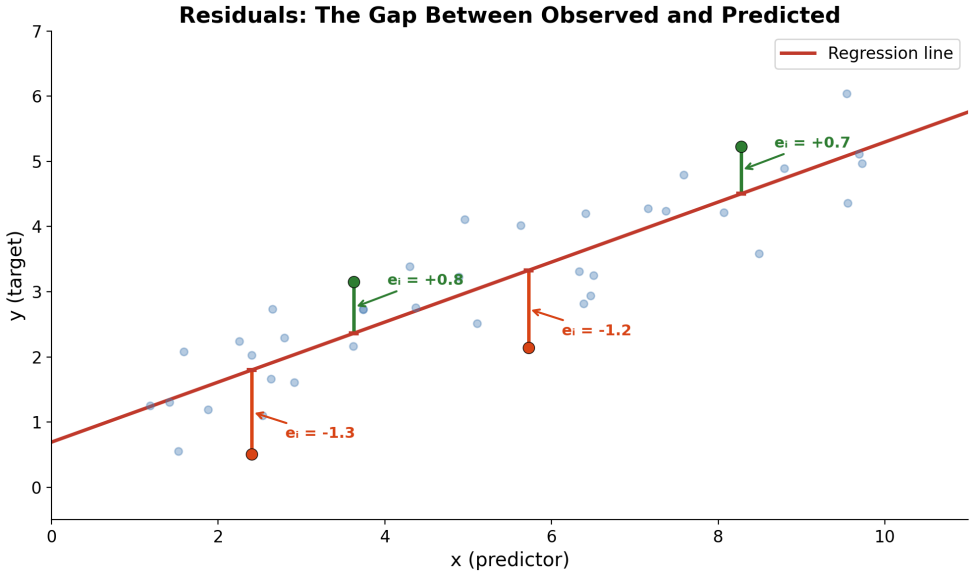
$\hat{y}_i$ = predicted value (what our model said)

e_i = residual = the prediction error for observation *i*

Residual Definition

$$e_i = y_i - \hat{y}_i$$

Positive = under-predicted (actual > predicted)
Negative = over-predicted



We need a single number summarizing all residuals ⟶ the objective function.

OLS Finds the Line That Minimizes Total Squared Error

Ordinary Least Squares (OLS) is the standard approach: square each residual, add them up, and find the β_0 and β_1 that make this sum as small as possible.

The OLS Objective

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Squaring makes all errors nonnegative and penalizes large errors disproportionately

Step 1: Compute the prediction for each data point: $\hat{y}_i = \beta_0 + \beta_1 x_i$

Step 2: Compute each residual: $e_i = y_i - \hat{y}_i$

Step 3: Square and sum: $\Sigma e_i^2 \longrightarrow$ find β_0, β_1 that minimize this

We can do derivation of the closed-form solution — but the CTA handles computation and coding. Our job: understand the choice.

But why squared? Would absolute errors work just as well? Let's find out...

🤖 CTA Demo: Does the Choice of Error Measure Matter?

What We'll Explore

We'll ask the CTA to generate a dataset with an outlier, fit two different lines — one minimizing squared errors (OLS), one minimizing absolute errors (LAD) — and compare how they respond to the outlier. Then we'll remove the outlier and see what changes.

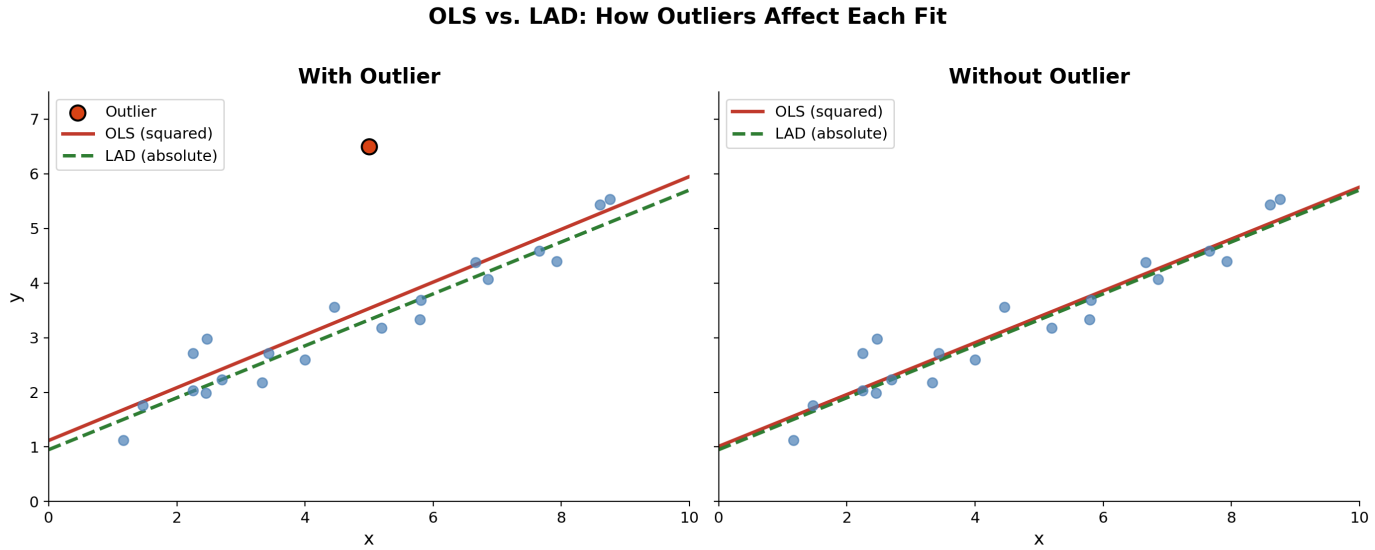
Key Questions:

- Which line is pulled more toward the outlier, and why?
- How much does each line change when the outlier is removed?
- When is outlier sensitivity a feature versus a bug?

✓ VERIFY Connection: Execute — does the code produce the expected comparison? Reason — does the visual evidence support the claim about outlier sensitivity?

Pattern: Iterative Refinement

Squared Errors Penalize Large Mistakes More Heavily



Key Insight: Squaring amplifies large errors. OLS is pulled toward outliers because a single large residual contributes disproportionately to the total squared error.

✓ **Feature:** When outliers are real signal that should influence the model

✗ **Bug:** When outliers are noise or data errors that distort the model

🤔 *What's the connection between OLS and maximum likelihood under normal errors? (Explore with CTA)*

Multiple Regression Uses Several Features Simultaneously

House prices don't depend on income alone — rooms, location, age, and more all matter.

Multiple regression lets us use all of these simultaneously.

Simple (1 feature):
$\hat{y} = \beta_0 + \beta_1 x$

→

Multiple (p features):
$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$

Multiple Linear Regression

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Each β_j measures the effect of feature x_j, controlling for all other features

- p predictors, each with its own coefficient β_j
- β_0 is still the intercept (predicted value when all $x_j = 0$)
- Geometry shifts from a line to a hyperplane — but the math is the same idea

The key insight is that each coefficient has a specific, interpretable meaning.

Each Coefficient Measures Effect While Holding Others Constant

The Most Important Practical Concept: β_1 = the change in $\hat{y}$ when x_1 increases by 1, holding all other predictors constant. This is the partial effect of x_1.

Example: California Housing

Note: House value is measured in $100,000s.

If β(`MedInc`) = 0.45 → Each $10K increase in median income is associated with a $45K increase in house value, holding rooms, age, location constant

If β(`AveRooms`) = 0.12 → A 1-room rise in block-group average rooms per household is associated with $12K higher median house value, holding other predictors constant.

⚠ **Caution:** Association, not causation. "Holding constant" is a statistical concept, not a physical experiment. The coefficient describes pattern in the data, not a causal mechanism.

Two questions every coefficient answer should address:

1. Direction: Is the association positive or negative?

2. Magnitude: How large is the effect in meaningful units?

California Housing: A Real Dataset Ready for Regression

20,640	8	House Value
Observations	Features	Target
California block groups	median income, rooms, age, ...	median, in $100,000s

Recall from L2: each row is one observation (n = 20,640), each column is one feature (p = 8).

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Lat	Long	MedHouseVal
0	8.3252	41.0	6.984	1.024	322.0	2.556	37.88	-122.23	4.526
1	8.3014	21.0	6.238	0.972	2401.0	2.11	37.86	-122.22	3.585
2	7.2574	52.0	8.288	1.073	496.0	2.802	37.85	-122.24	3.521
3	5.6431	52.0	5.817	1.073	558.0	2.548	37.85	-122.25	3.413
4	3.8462	52.0	6.282	1.081	565.0	2.181	37.85	-122.25	3.422

L4 Connection: This data has already been cleaned. In practice, you'd apply L4's data wrangling pipeline before reaching this point.

`Features: MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude`

Coefficients Change When You Add Predictors

The motivating puzzle: what happens to a coefficient when we add another feature?

Simple Regression
Price ~ AveRooms only

β(AveRooms) = **+0.35**

→

Multiple Regression
Price ~ AveRooms + MedInc

β(AveRooms) = **+0.12**

🤔 **The coefficient for rooms dropped by roughly two-thirds. Why?**
Is the simple regression wrong? Is the multiple regression wrong? Or are they answering different questions?

Let's ask the CTA to explain — and then verify its explanation with data.

🤖 CTA Demo: Why Did the Rooms Coefficient Shrink?

What We'll Explore

The CTA should be able to explain why a coefficient changes when we add a predictor. But can we verify its explanation? We'll ask for an explanation, then test it by computing the correlation between the two predictors.

Key Turns:

Turn 1: Ask: "Why does the rooms coefficient change when we add income?"

Turn 2: CTA explains shared variance / confounding between predictors

Turn 3: Test: "Compute the correlation between `AveRooms` and `MedInc`; `Price` and `MedInc`"

Turn 4: CTA computes r(`AveRooms`, `MedInc`) ≈ 0.33; r(`Price`, `MedInc`) ≈ 0.69 — substantial correlation confirms the explanation

✓ VERIFY Connection: Validate — is the explanation consistent with data? Execute — does the correlation computation confirm the claim?

Pattern: Verification Through Computation

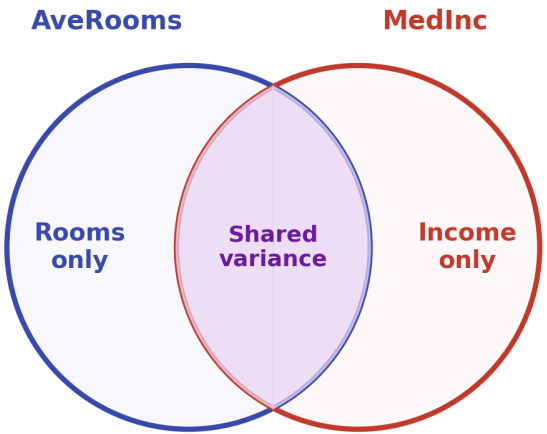
Shared Variance Explains Why Coefficients Shift

Recall from L3: correlation measures linear association between two variables, from −1 to +1.

When two predictors are correlated, they share explanatory power. In simple regression, rooms gets credit for everything — including the part that's really due to income.

Key Insight: The verification loop — CTA explains → human designs a test → CTA runs the test → human evaluates. This is the Implementation Assistant workflow in action.

Shared Variance Between Correlated Predictors



Simple regression gives Rooms credit for the shared region.
Multiple regression splits it — each feature gets only its unique contribution.

Demo result: r(`AveRooms`, `MedInc`) ≈ 0.33; r(`Price`, `MedInc`) ≈ 0.69 — substantial correlation. The CTA's explanation checks out.

Four Statistics Tell You Whether a Coefficient Is Trustworthy

From L3: p-values test hypotheses; confidence intervals give plausible ranges. Now you'll see them applied to regression coefficients.

1 Estimate
$\widehat{\beta}$ (coefficient)

Our best guess for the effect of this feature

(Taught on Slides 7, 13)

↓

2 How uncertain?
Std Error (SE)

If we collected new data and refit, how much would $\hat{\beta}$ vary?

🤖 **Ask the CTA:** *"Why is the SE for `MedInc` smaller than for `AveRooms`?"*

↓

3 Signal vs. noise?
$t = \widehat{\beta}$ / SE

Is the coefficient far enough from zero, relative to its uncertainty, to be convincing?

🤖 **Ask the CTA:** *"What t-value would we need to be confident this feature matters?"*

↓

4 Verdict
p-value + 95% CI

How likely is this if the feature had no effect? What range of true values is plausible?

🤖 **Ask the CTA:** *"Show me how the CI changes if I double the sample size"*

The chain: estimate ⟶ uncertainty ⟶ signal-to-noise ⟶ verdict

💡 **Key Insight:** The CTA helps generate all four statistics. Your job: understand the chain so you can spot when the numbers don't tell a trustworthy story. In Session 2, you'll see multicollinearity break this chain by inflating standard errors.

Session 1 Summary: Regression Finds Relationships, Coefficients Tell the Story

1. Simple linear regression finds the best-fitting line by minimizing squared errors (OLS)

2. Squaring amplifies large errors — a deliberate choice, not the only option

3. Multiple regression uses p features simultaneously, each with a partial-effect coefficient

4. Coefficients change when you add predictors because correlated features share explanatory power

5. A regression summary table connects to L3's statistics: coefficients, standard errors, p-values, CIs

Coming Next — Session 2: We've built a model. But can we trust it? How do we know it will work on new data?

From Building Models to Judging Them

Multicollinearity Means Predictors Share Information

Recall from L3: correlation measures linear association between two variables, from −1 to +1.

Definition: Multicollinearity or Collinearity occurs when two or more predictor variables are highly correlated with each other — they carry overlapping information about the target.

Consequences:

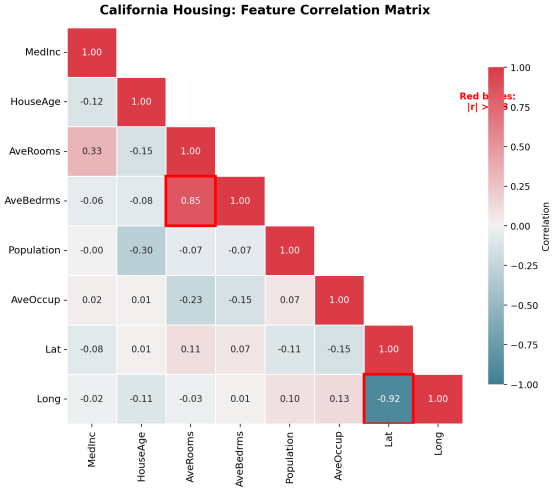
- ⚠ Coefficient instability — small changes in data cause large swings in individual β values
- ⚠ Inflated standard errors — harder to determine if individual features are significant
- ⚠ Unreliable individual interpretation — can't cleanly separate each feature's contribution

Key Insight: Predictions can still be accurate even with multicollinearity. The problem is with interpreting individual coefficients, not with the model's overall predictions.

This formalizes what we saw in Demo 2: rooms and income share variance ($r \approx 0.33$), making each coefficient unstable across models.

Diagnose Multicollinearity with Correlation Matrices and VIF

Tool 1: Correlation Matrix Heatmap



Look for $|r| > 0.7–0.8$ between predictor pairs.

Tool 2: Variance Inflation Factor (VIF)

VIF	Interpretation
= 1	No overlap with other predictors
1–5	Moderate — usually acceptable
> 5	Concerning — investigate further
> 10	Serious — take action

VIF measures how much a coefficient estimate's variance is inflated by correlation with other predictors. Higher VIF = more shared information.

What to do if you find it:

Remove
Drop one of the correlated features

Combine
Create a composite feature (e.g., average)

Regularize
Use Ridge or Lasso (latter slides)

When Multicollinearity is a Problem and When It isn't

Matters for Interpretation

- "Which features drive house prices?"
- "Is income or rooms more important?"
- "Should we invest in feature X?"

→ Individual coefficients unreliable

→ Standard errors inflated → lower precision

Less of an Issue for Prediction

- "What will this house sell for?"
- "Predict next quarter's revenue"
- "Estimate tomorrow's temperature"

→ Overall predictions still accurate

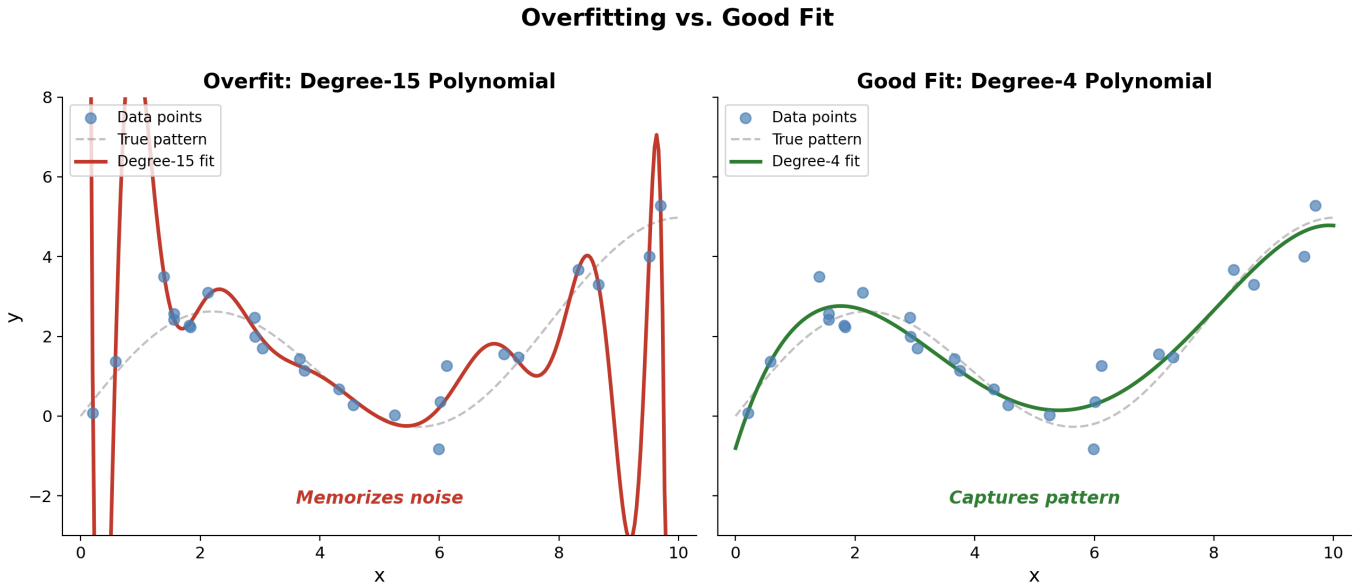
→ Model uses combined information effectively

Practical Takeaway: Always check for multicollinearity. Whether it's a problem depends on your goal: interpretation requires action, prediction usually doesn't.

Multicollinearity threatens interpretation. But there's a more fundamental problem: a model can look great on training data and fail on new data. This is overfitting.

A Model That Memorizes Noise Has Overfit

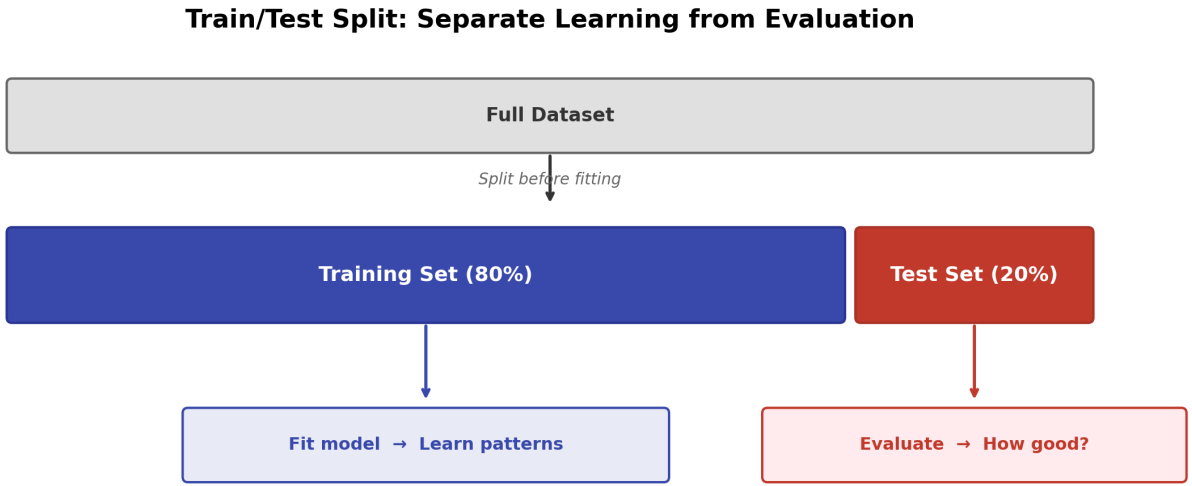
Student Analogy: Memorizing answers to past exams vs. understanding the material. The memorizer scores 100% on practice tests but fails on new questions. Overfitting is the same problem: the model learns the noise in the training data instead of the underlying pattern.



Overfitting: A model performs well on the data it was trained on but poorly on new, unseen data. It has captured noise rather than signal.

How do we detect this? We need to test the model on data it has never seen.

Train/Test Split Separates Learning from Evaluation



1. Split your data before fitting: typically, 70–80% training, 20–30% test. [keep it random and balanced]

2. The model learns patterns from the training set only. [keep test set unseen and not leaked]

3. Evaluate performance on the test set — data the model has never seen

Implementation: `X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`

R^2 Measures the Fraction of Variance Your Model Explains

From L2: variance measures how spread out values are — the average squared deviation from the mean ($\bar{y}$).

$SS_{\text{res}} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ → unexplained variance (residual sum of squares)

$SS_{\text{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2$ → total variance (how much *y* varies overall)

Coefficient of Determination

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

R^2 = 0.7 means the model explains 70% of the variance in *y*

Interpretation Guide:

R^2 = 0
Model is no better than predicting the mean

R^2 = 0.5
Model explains half the variance

R^2 = 1
Perfect prediction (suspicious — likely overfit!)

What counts as a "good" R^2 depends entirely on the domain. Predicting stock prices? R^2 = 0.05 may be valuable. Predicting physics experiments? R^2 = 0.95 may be expected.

R^2 has Limitations: High R^2 Does Not Guarantee a Good Model

Limitation 1: R^2 can NOT decrease as OLS gains features (on training data)
Adding even random noise as a feature cannot decrease training R^2. More features = more flexibility = better fit to this specific dataset — even if the new features are meaningless.

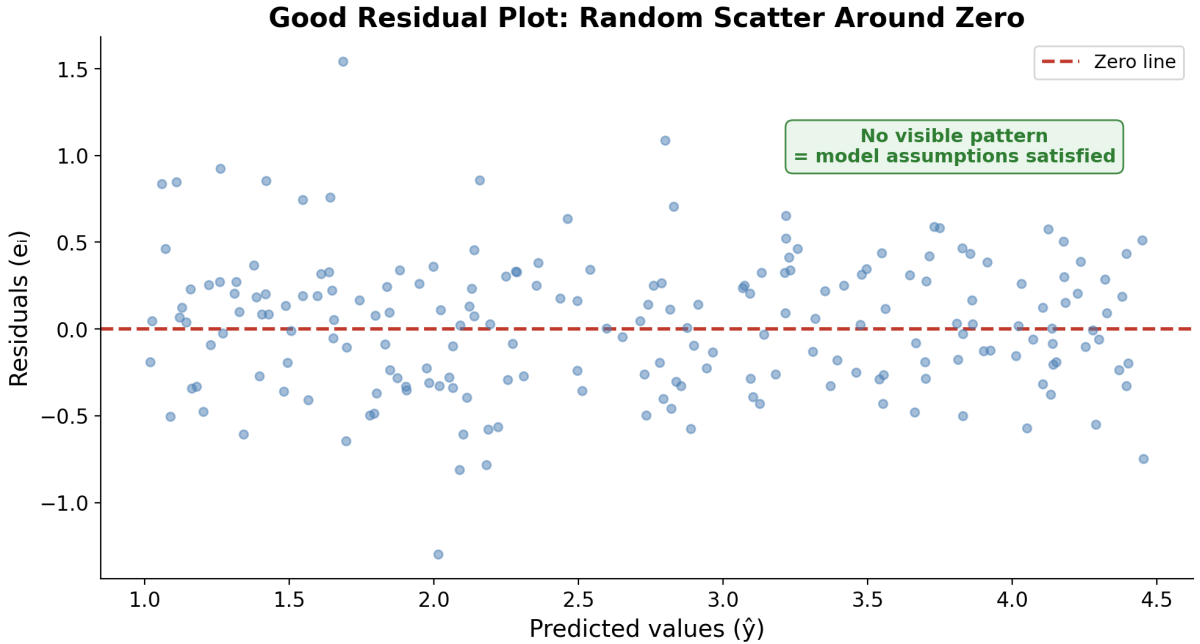
Limitation 2: R^2 does NOT detect wrong patterns
A model that fits a straight line to curved data can still have a moderate R^2. The number alone doesn't tell you whether the model captures the right relationship.

Limitation 3: R^2 can NOT compare across different datasets
$R^2 = 0.8$ on a noisy dataset may reflect a better model than $R^2 = 0.95$ on a clean one. The baseline variance differs.

This is why we need additional tools: RMSE for absolute error and residual plots for pattern detection.

Residual Plots Reveal What Metrics Cannot

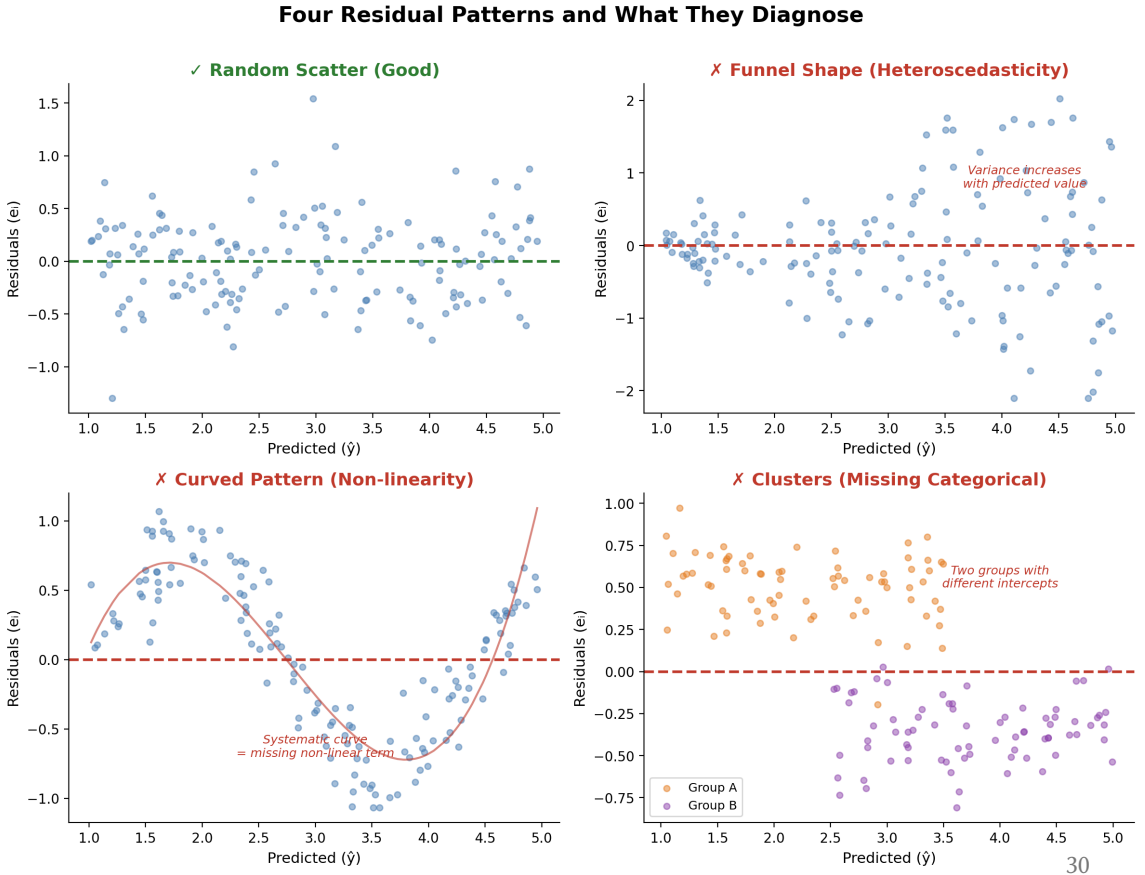
Plot residuals (y-axis) vs. predicted values $\hat{y}$ (x-axis). A good model produces random scatter around zero. Any visible pattern reveals a problem.



Key Insight: Metrics give a number. Residual plots give a diagnosis. A model with $R^2 = 0.8$ and patterned residuals needs fixing; a model with $R^2 = 0.6$ and random residuals may be fine.

Four Residual Patterns and What They Diagnose

Reference Slide: When you generate a residual plot, compare it against these four patterns. Random scatter = good. Anything else = diagnosis needed.



🤖 CTA Demo: A Complete Model Evaluation Workflow

What We'll Explore

We'll give the CTA a single structured request specifying six deliverables. It will help execute a complete evaluation pipeline in seconds. But its interpretation of the residual plot will be incomplete — revealing that visual diagnostic judgment is a human skill.

The Structured Request (6 deliverables):

1. Split into 80/20 train/test
2. Fit linear regression with all features
3. Report R^2 on both train and test
4. Generate residual plot (residuals vs. predicted)
5. Generate histogram of residuals
6. Provide 3-sentence interpretation

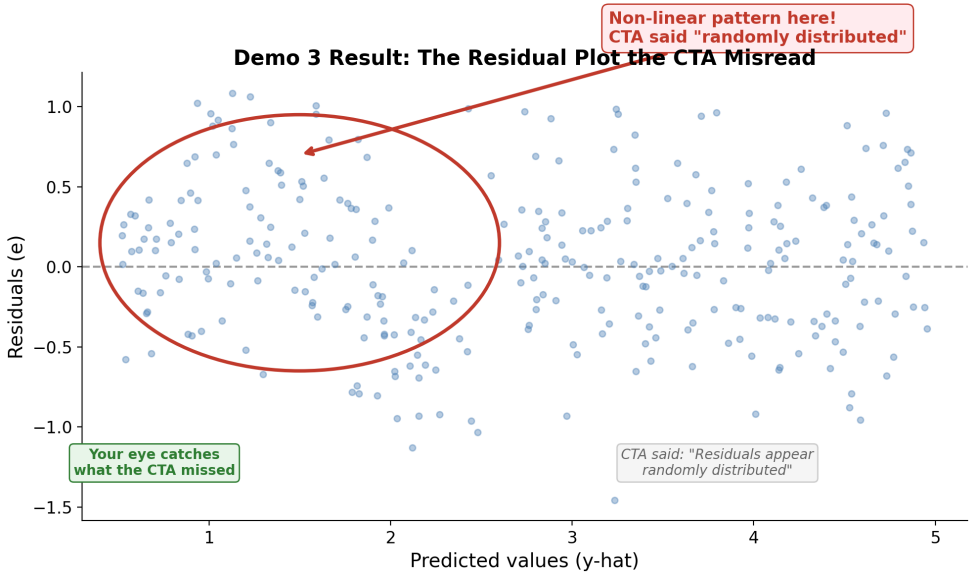
✓ **VERIFY:** Execute — do all 6 outputs appear? Inspect — does the residual plot show patterns the CTA missed? Fix — iterate based on diagnostics.

Pattern: Structured Requests + Verification

The CTA Computes Quickly - But Interpreting Residuals Requires You

What happened:

✓ The CTA executed all 6 deliverables correctly and quickly
⟶ The CTA said residuals "look randomly distributed"
✗ But the residual plot shows a non-linear curve in the lower range
⟶ We asked the CTA what could cause this — it suggested feature transformations



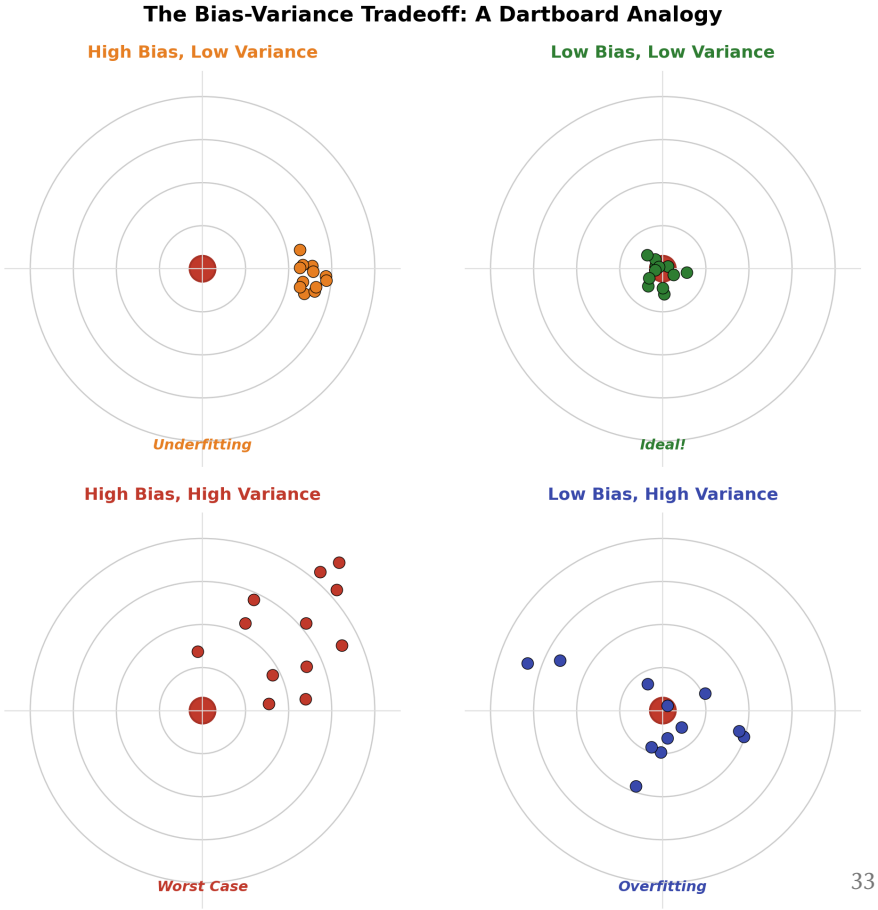
Key Insight: The CTA is your computational engine. You are the diagnostic eye. The cycle: diagnose from residuals ⟶ iterate with CTA ⟶ re-evaluate.

The Bias-Variance Tradeoff: The Dartboard Analogy

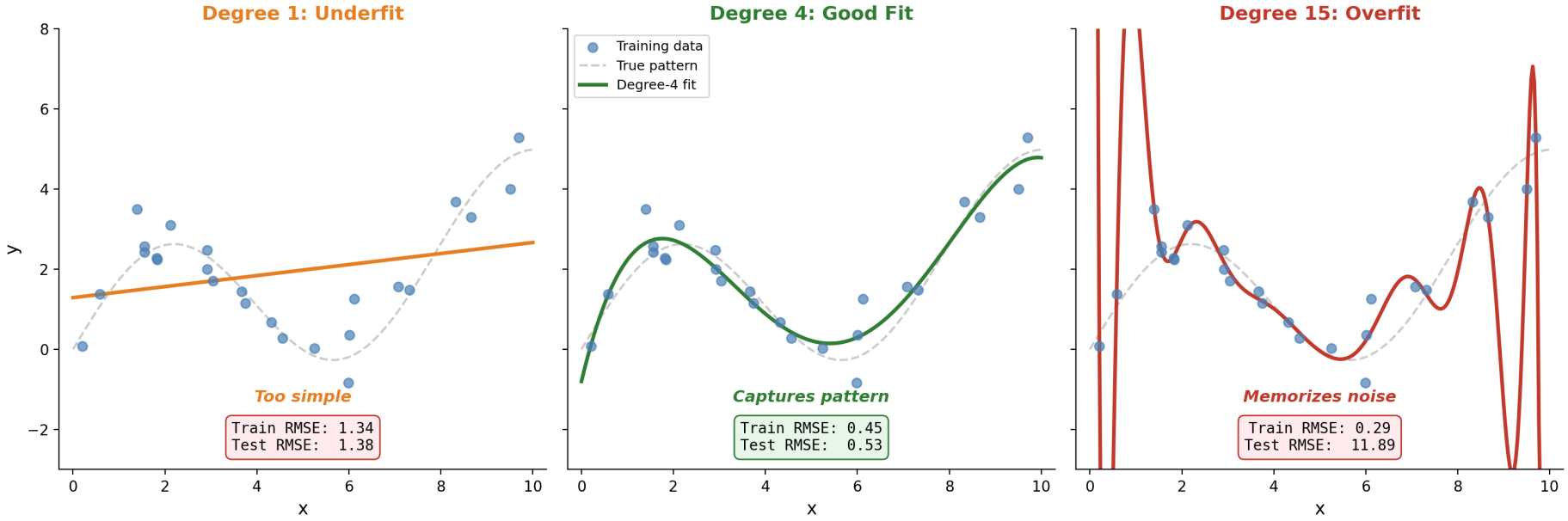
Bias
How far off the average prediction is from the true value.
High bias = systematically wrong = underfitting.

Variance
How much predictions change with different training data.
High variance = unstable = overfitting.

Key Insight: More flexible models often trade lower bias for higher variance. Aim to minimize expected squared error: squared bias + variance + irreducible noise.



Underfitting, Good Fit, and Overfitting Visualized



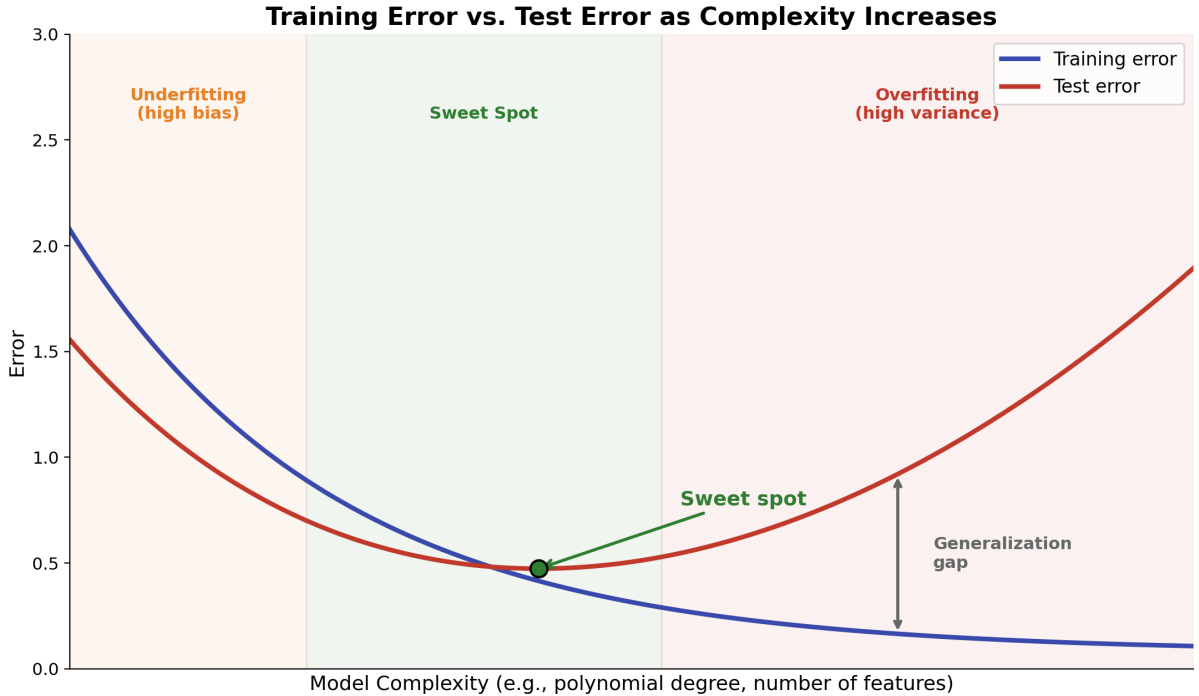
Degree 1: Underfit
Too simple — misses the pattern entirely

Degree 4: Good Fit
Captures the pattern without memorizing noise

Degree 15: Overfit
Memorizes training data, fails on new data

Notice: Degree 15 has the lowest training error but the highest test error. More complexity doesn't mean better generalization.

More Complex Models Are Not Always Better



This is the most important figure in the course. Training error cannot increase as OLS gains features — the model gets better at memorizing. Test error first decreases (less bias), then increases (more variance). The gap between the two curves is the generalization gap. Your job is to find the sweet spot.

Practical implication: always evaluate on test data. Training performance alone tells you nothing about generalization.

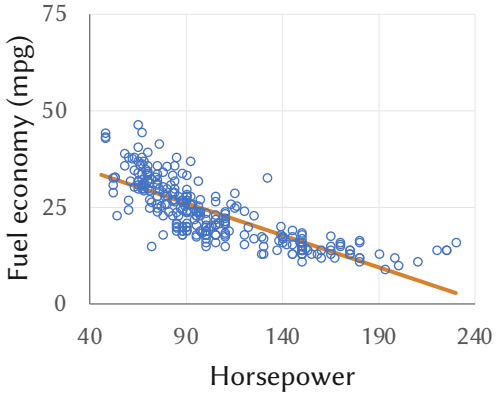
Real Cars: More Coefficients, More Flexible Fits

Predict city-cycle fuel economy (mpg) from horsepower.

UCI Auto MPG (1970–82): 274 training cars (blue), 118 held out. Fixed split (seed 42).

2 fitted coefficients

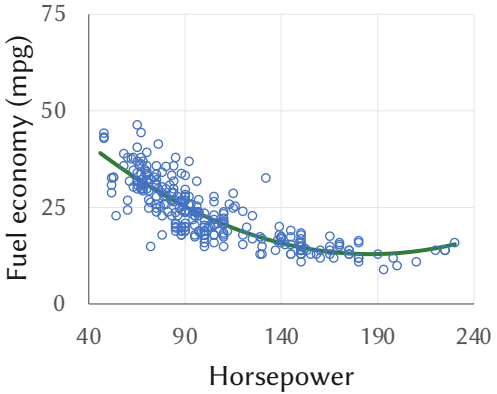
Polynomial degree 1



Train 4.89 Test 4.96

3 fitted coefficients

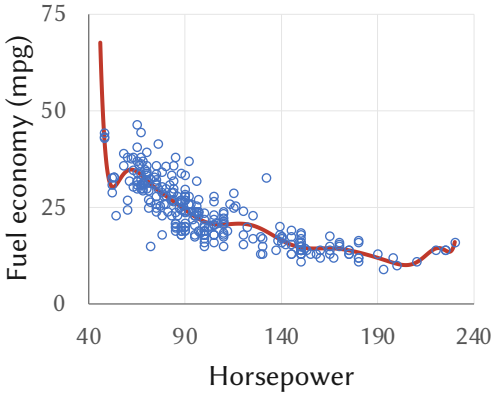
Polynomial degree 2



Train 4.31 Test 4.52

16 fitted coefficients

Polynomial degree 15



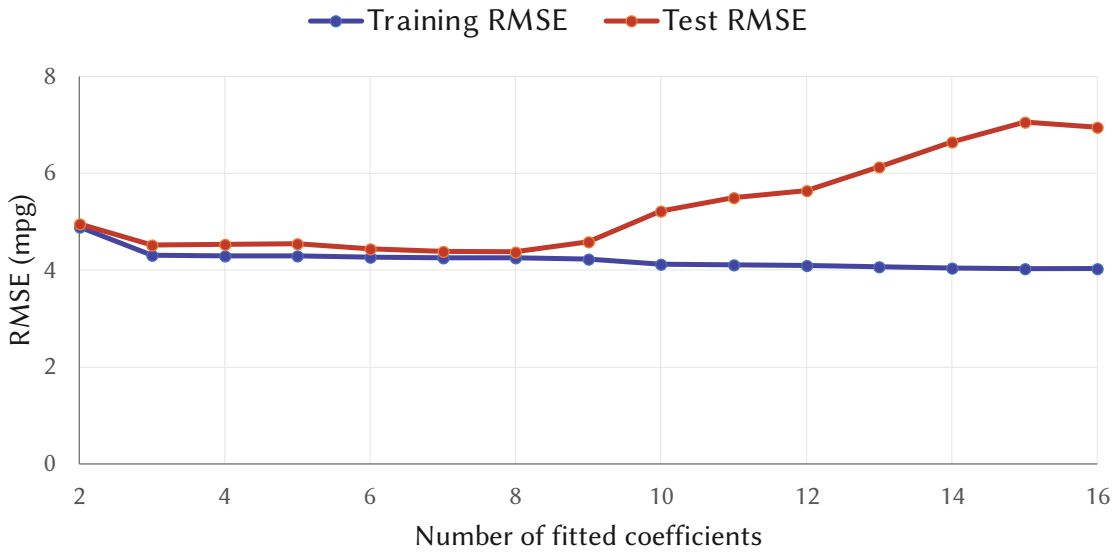
Train 4.03 Test 6.95

Coefficients = degree + 1 (including the intercept). Train/test errors are RMSE in mpg.

Same data and fitting rule. Only the number of fitted coefficients changes.

Training Error Falls, but Test Error Can Rise

Least squares minimizes MSE. RMSE is its square root, in mpg (lower is better).



3 versus 16 coefficients

Training error
4.31 to 4.03 mpg
6% lower

Test error
4.52 to 6.95 mpg
54% higher

Training: the 274 cars used to fit. Test: 118 cars the fit never saw.

Extra flexibility improves fitting, but can make predictions less reliable.

Pause and Think: Why Does Training R^2 Never Decrease with More Features?

🤔 **Take 30 seconds to think about this.**

If you add a completely random column of numbers as a feature to your regression, training R^2 cannot decrease under OLS. Why? And what happens to test R^2?

The Answer:

Adding a feature gives the model more flexibility. Even a random column will correlate slightly with the target by chance in any finite sample. The model exploits this spurious correlation to reduce training error.

Bias-variance connection: An irrelevant random feature adds no predictive signal, but can increase variance by fitting noise. Training R^2 cannot decrease under OLS, but test R^2 typically goes down.

🤔 Want to see this in action? Ask the CTA to add 1, 5, then 10 random columns to California Housing and plot train vs. test R^2.

Session 2 Summary: Good Models Generalize, Not Just Memorize

1. Multicollinearity makes individual coefficients unreliable — diagnose with correlation matrices and VIF
2. Overfitting means a model memorizes noise; train/test split detects it
3. R^2 measures variance explained; RMSE measures error in target units — use both
4. Residual plots reveal problems that metrics miss — the CTA computes, you interpret
5. The bias-variance tradeoff: more complexity reduces bias but increases variance
6. Always evaluate on test data — training performance alone is misleading

Coming Next: The bias-variance tradeoff seems like a dilemma — but regularization offers a way out. We can constrain complex models to improve generalization.

Constraining Models to Improve Them

The Problem: Too Many Features, Not Enough Signal

Imagine you start with the California Housing dataset (8 features). As you'll see in projects and practice, feature engineering can expand your feature set dramatically. Starting from 8 raw features, you might end up with 100+.

p small relative to n
8 features, 20,640 observations
$n/p \approx 2{,}580$
OLS works well — enough data per feature

p large relative to n
100+ features, same 20,640 observations
$n/p \approx 200$ (or worse with subsets)
OLS overfits — too many knobs to turn

Bias-Variance Connection: More features means more model complexity, which means lower bias but higher variance. When p is large relative to n, the variance penalty dominates — OLS overfits.

The solution: keep all the features, but constrain how much influence each one can have. This is regularization.

Regularization Adds a Penalty for Model Complexity

Regularized Objective

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

First term = fit the data | Second term = keep the model simple | λ controls the balance

$\Sigma(y_i - \hat{y}_i)^2$
Data fidelity — how well the model fits the training data (same as OLS)

$\lambda \cdot$ Penalty(β)
Complexity penalty — discourages large coefficient values

The penalty strength λ (also called α in `sklearn`):

$\lambda = 0$
No penalty ⟶ pure OLS (high variance)

Small λ
Light constraint ⟶ slightly less variance, slightly more bias

Optimal λ
Best balance ⟶ minimizes total test error

Large λ
Heavy constraint ⟶ all coefficients near zero (high bias)

Key Insight: Regularization trades a small, controlled increase in bias for a substantial reduction in variance. The net effect is usually better generalization.

Ridge Regression Shrinks Coefficients Toward Zero

Ridge Regression (L2 Penalty)

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Penalty = sum of squared coefficients ⟶ coefficients are shrunk, but typically remain nonzero

⟶ Coefficients are penalized more strongly as λ increases — and typically remain nonzero

⟶ Useful when you believe many features each contribute a small amount to the prediction

⟶ Handles multicollinearity well — shrinking correlated coefficients stabilizes them

Implementation: `from sklearn.linear_model import Ridge; model = Ridge(alpha=1.0).fit(X_train, y_train)`

Intuition: Think of it as adding friction to the coefficients. OLS can swing wildly; Ridge dampens those swings. Every feature stays in the model, but with reduced influence.

Lasso Regression Can Eliminate Features Entirely

Lasso Regression (L1 Penalty)

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Penalty = sum of absolute coefficients ⟶ some coefficients set to exactly zero = automatic feature selection

⟶ Some coefficients shrink to exactly zero — the model drops those features entirely

⟶ Automatic feature selection: Lasso tells you which features matter most

⟶ Useful when you believe many features are irrelevant — the model becomes sparse

```
Implementation: from sklearn.linear_model import Lasso; model = Lasso(alpha=1.0).fit(X_train, y_train)
```

L2 Connection: Both Lasso and PCA (from L2) reduce dimensionality — but differently. PCA creates new composite features. Lasso selects existing features by zeroing out the rest.

Feature Scaling is Required Before Regularization

From L1/L4: standardization (subtract mean, divide by std dev) puts all features on the same scale.

The penalty $\lambda\Sigma\beta_j^2$ (or $\lambda\Sigma|\beta_j|$) treats all coefficients equally.

But if features are on different scales, their coefficients will be on different scales too — the penalty would unfairly penalize some features more than others.

Example:
`MedInc` ranges roughly from 0.5 to 15 in units of \$10,000, while AveRooms ranges roughly from 1 to 8. The scales are different enough that the penalty would treat their coefficients unequally without standardization.

Solution: `from sklearn.preprocessing import StandardScaler; scaler = StandardScaler().fit(X_train)`

⚠ **Critical: Fit the scaler on training data only!**
If you fit the scaler on all data (including test), you leak test information into training. Use `scaler.fit(X_train)`, then `scaler.transform(X_train)` and `scaler.transform(X_test)` separately.

Rule: Always standardize features before Ridge or Lasso. Fit on training data only. Essential when penalties treat all coefficients equally.

🤖 CTA Demo: How Coefficients Change as Penalty Increases

What We'll Explore

We'll ask the CTA to fit Lasso for 50 different penalty strengths and plot how each coefficient changes. Some coefficients hit zero — why? We'll challenge the CTA to explain the geometric intuition: diamond vs. circle constraint regions.

Key Turns:

Turn 1: "Fit Lasso on California Housing for 50 α values (regularization factor λ) from 0.001 to 10. Plot coefficient paths."

Turn 2: Observe which coefficients hit zero first. Pose: "Ridge typically keeps coefficients nonzero. Why is Lasso different?"

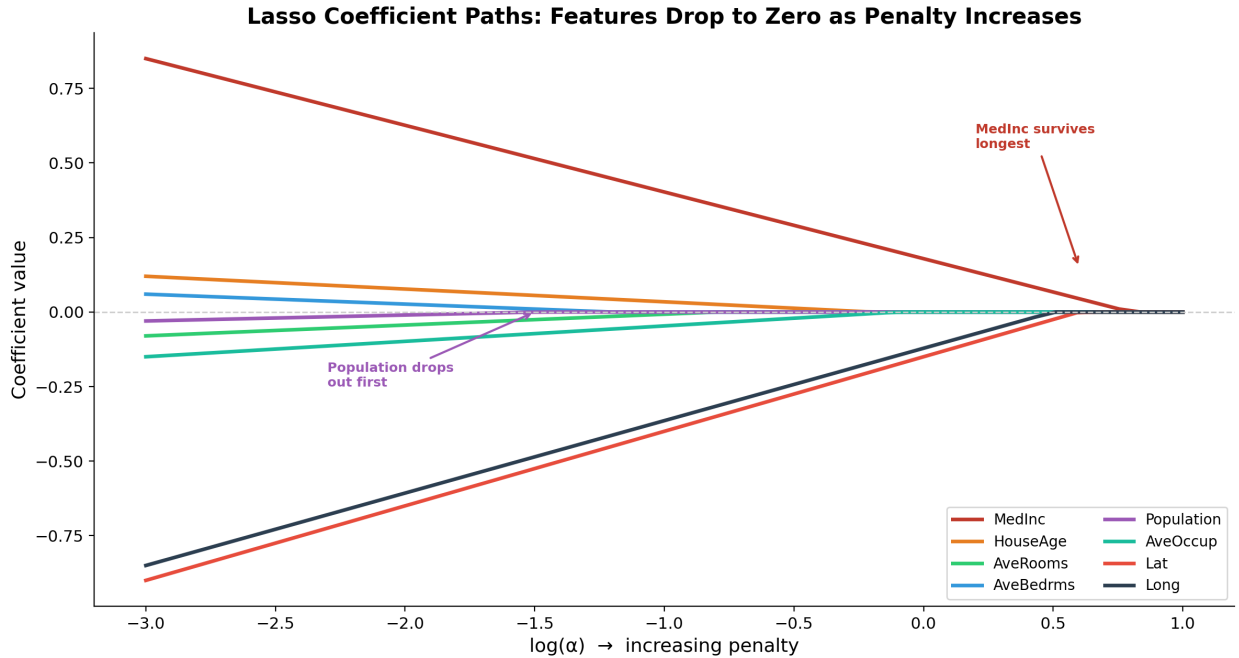
Turn 3: CTA explains diamond vs. circle constraint regions — challenge for geometric visualization.

Turn 4: Evaluate CTA's visualization.

✓ VERIFY: Reason — does the geometric explanation logically account for the empirical observation?
Yours — can you explain the diamond-vs-circle argument yourself?

Pattern: Iterative Refinement + Challenging for Completeness

Lasso Coefficients Hit Zero at Different Penalty Strengths



Observation: Less important features hit zero first. At large α, only the strongest predictors survive. This is automatic feature selection — the model tells you what matters.

🤔 **Key question:** Ridge typically keeps nonzero coefficients. Lasso coefficients hit exactly zero. Why the difference? The answer lies in geometry.

Why Does Lasso Produce Exact Zeros but Ridge Doesn't?

We've seen the empirical evidence: Lasso zeros out coefficients, Ridge merely shrinks them. Both add a penalty. Both constrain the coefficients. So what's different?

The difference comes from the shape of the constraint region.

Lasso constrains coefficients to a diamond (L1 ball).
Ridge constrains coefficients to a circle (L2 ball).

The OLS solution's loss contours are ellipses.

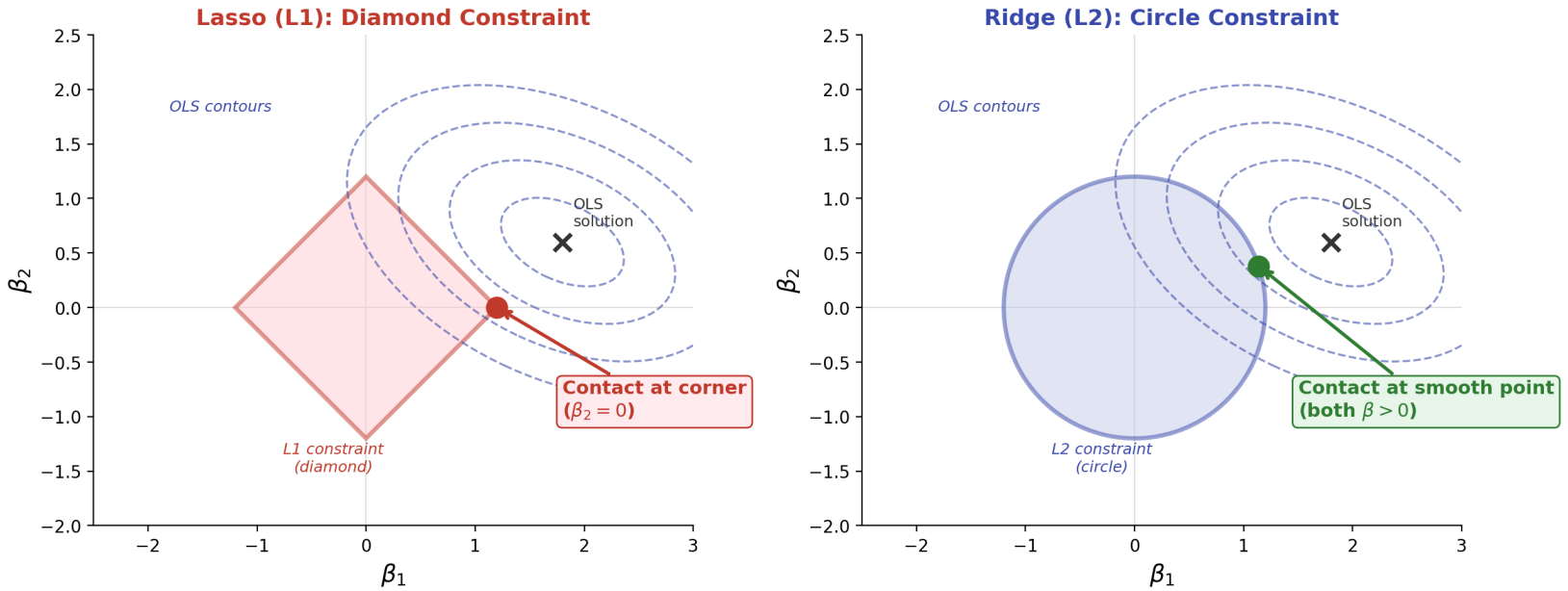
The question becomes: where does the first ellipse touch the constraint boundary?

Diamond (Lasso): Has corners on axes ⟶ ellipse hits a corner ⟶ one $\beta = 0$

Circle (Ridge): Has no corners ⟶ ellipse touches a smooth point ⟶ typically $\beta \neq 0$

The Geometric Intuition: Diamond Meets Ellipse

Why Lasso Produces Zeros: Corners vs. Curves



Key Insight: Corners produce zeros. The L1 diamond has corners on the axes. The L2 circle has no corners. That geometric difference is why Lasso produces sparse solutions and Ridge doesn't.

Ridge vs. Lasso: Choosing the Right Regularization

	Ridge (L2)	Lasso (L1)
Penalty	$\lambda\Sigma\beta_j^2$ (squared)	$\lambda\Sigma\lvert\beta_j\rvert$ (absolute)
Coefficients	Shrink; typically remain nonzero	Shrink, some reach exactly zero
Feature selection	No — keeps all features	Yes — automatic
Best when	Many features, all contribute; correlated features	Many features, few relevant; want sparse model
Geometry	Circle — no corners ⟶ no zeros	Diamond — corners ⟶ zeros

Elastic Net: Combines L1 and L2 penalties: $\lambda_1\Sigma\lvert\beta_j\rvert + \lambda_2\Sigma\beta_j^2$. Gets feature selection from Lasso and stability from Ridge. Explore with CTA.

⚠ **Lasso + Multicollinearity Caveat:** When features are highly correlated, Lasso arbitrarily picks one and zeros the others. Which one it keeps can change with small data changes. Ridge or Elastic Net may be more stable.

Selecting the Penalty Strength: Try, Evaluate, Pick the Best

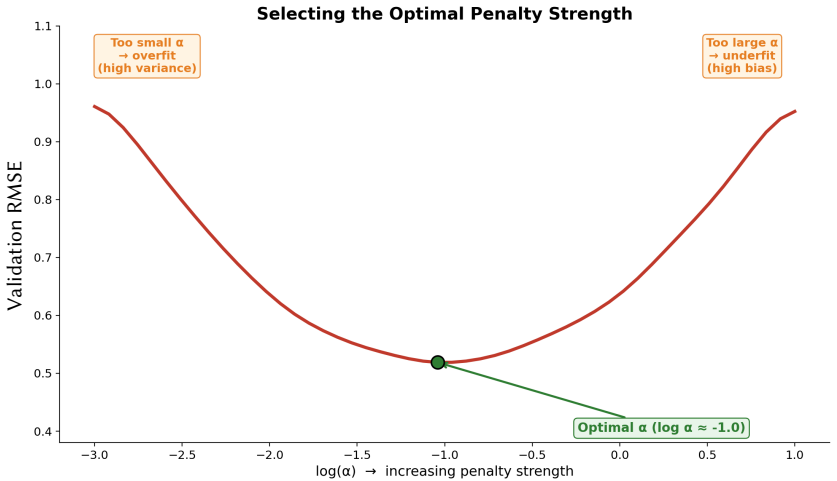
The simplified approach (full cross-validation treatment in L6):

1. Try: Fit the model for a range of α values (regularization factor λ): e.g., 50 values from 0.001 to 10

2. Evaluate: Compute validation RMSE (or validation R^2) for each α

3. Pick: Select the α that minimizes validation error

Shortcut: `sklearn` provides `LassoCV` and `RidgeCV`, which automate α selection using cross-validation internally. We'll cover cross-validation formally in L6 — for now, understand the principle: try a range, evaluate, pick the best.



The Complete Regression Workflow

Prepare ⟶ Build ⟶ Evaluate ⟶ Improve

Prepare
Lecture 4
Wrangle, encode, scale features

Build
Session 1
OLS, multiple regression

Evaluate
Session 2
R^2, RMSE, residual diagnostics

Improve
Session 3
Regularization, penalty tuning

Key Insight: This workflow is iterative, not linear. Poor residual plots send you back to Prepare (better features) or Improve (different regularization). The CTA executes each stage; you direct the iteration.

Summary: Seven Key Takeaways

1. Regression predicts continuous outcomes using linear combinations of features
2. OLS minimizes squared residuals — but coefficients need interpretation, not just computation
3. Multiple regression introduces multicollinearity — diagnose with VIF before interpreting
4. Overfitting means low training error, high test error — always use train/test split
5. R^2 and RMSE quantify model quality; residual plots reveal problems metrics miss (DO Visualize!)
6. The bias-variance tradeoff is the central tension — regularization manages it
7. Ridge shrinks, Lasso selects — choose based on your assumption about feature relevance

The CTA computes; you evaluate, interpret, and direct improvement.

Preview: Classification Reuses This Evaluation Framework

Lecture 6 moves from predicting how much (continuous) to predicting which (categorical).

What Stays the Same

→ Train/test split methodology
→ Overfitting diagnosis and prevention
→ Regularization (Ridge/Lasso concepts)
→ The VERIFY framework
→ CTA workflow: compute → evaluate → iterate

What Changes

→ Target: categorical (churn yes/no)
→ Model: logistic regression replaces OLS
→ Metrics: accuracy, precision, recall, AUC
→ Challenge: accuracy misleads (imbalanced)
→ Dataset: Telco Churn from L4

Same Dataset, New Question: In L4 you wrangled the Telco Churn dataset. In L5 you learned how to evaluate models rigorously. In L6, you'll apply that evaluation framework to a classification problem on the same data — proving that evaluation skills transfer across model types.

⚠ **Preview puzzle:** If only 5% of customers churn, a model that predicts "no churn" for everyone achieves 95% accuracy. Is that a good model? (Hint: no. L6 will explain why and what to do about it.)

Technically Intriguing Questions for CTA Exploration

Explore these with your CTA outside class to deepen your understanding.

TIQ 1: What is the connection between OLS and maximum likelihood estimation under normal errors?
Ask the CTA to derive the connection. Verify that maximizing the Gaussian likelihood gives the same β as OLS.

TIQ 2: Why does adding a random feature never decrease training R^2 but usually decrease test R^2?
Ask the CTA to add 1, 5, then 10 random columns to California Housing data and plot train vs. test R^2.

TIQ 3: What happens with Elastic Net? When might it beat Ridge or Lasso alone?
Ask the CTA to fit `ElasticNet` on California Housing data; compare coefficient paths to Ridge and Lasso.

TIQ 4: Is polynomial regression ($y = \beta_0 + \beta_1 x + \beta_2 x^2$) actually a "linear" model?
The answer reveals what "linear" means linearly combining features. The coefficients β_i are linear; $\mathbf{x}^2$ is a new feature.

They build the conceptual depth that distinguishes strong data scientists.

Recommended Readings and Resources

Core Readings

ISLP Chapter 3 (Sections 3.1–3.3)
Linear regression fundamentals — simple regression, multiple regression, interpretation

ISLP Chapter 6.2
Shrinkage methods — Ridge and Lasso regression, penalty selection

An Introduction to Statistical Learning `https://hastie.su.domains/ISLP/ISLP_website.pdf.download.html`

Documentation

→ `sklearn.linear_model` — `LinearRegression`, `Ridge`, `Lasso`, `ElasticNet`, `LassoCV`, `RidgeCV`

→ `sklearn.preprocessing` — `StandardScaler`; `sklearn.model_selection` — `train_test_split`

→ `sklearn.metrics` — `r2_score`, `mean_squared_error`, `mean_absolute_error`

💡 **Pro Tip:** Ask the CTA to explain any concept from ISLP in simpler terms, or to generate code examples that implement the textbook equations. Reading + CTA exploration = deeper understanding.

Questions?

CONTACT INFORMATION



yangliu.mit.edu

Instructor: Prof. Yang Liu

Email: yliu12@hku.hk

Office Hours: 5:00 - 5:50 PM, Thursdays

Location: CB-714, Chow Yei Ching Building

Let me know if you want remote!

🤖 **CTA is your 24/7 interactive and responsive learning partner!**

TA contacts

Tuesday 4-5PM

Yushi Du:
u3012725@connect.hku.hk
https://us04web.zoom.us/j/78561597481

Zhengyang Hu:
u3010250@connect.hku.hk
https://hku.zoom.us/j/9021481973

Thursday 3-4PM

Jun Dai:
jundai332@connect.hku.hk
https://rochester.zoom.us/j/97202452759

Lulu Cai:
u303011209@connect.hku.hk
https://hku.zoom.us/j/5010926726

TA office hours on Zoom! Two slots (1 hour each) on Tuesdays and Thursdays