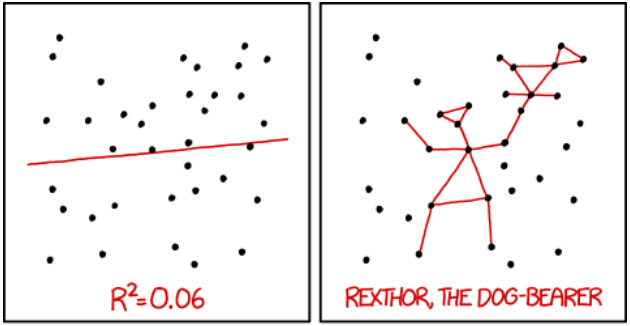
Linear Regression

Full Technical Lecture

Methods & Algorithms

MSc Data Science – Spring 2026

Can You Really Just Draw a Line Through Anything?



"I found a strong correlation between the number of people who drowned in swimming pools and the number of Nicolas Cage films."

Why Do Banks Struggle to Value Properties Consistently?

Situation

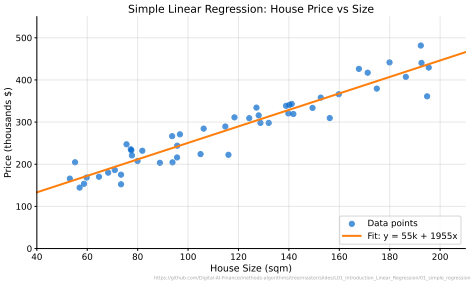
- A bank receives 500 mortgage applications per week
- Each property has size, location, age, condition

Complication

- Human appraisers disagree by 10–20%
- Inconsistent valuations create risk

Question

Can we learn a systematic rule from past transactions that predicts price from features – and quantifies uncertainty?



Linear regression turns this question into a mathematical optimization problem with a unique, closed-form solution

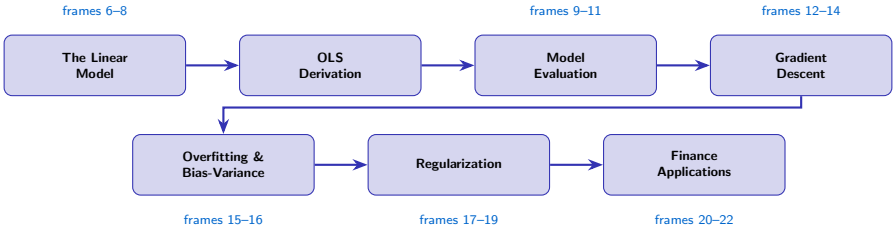
What Will You Be Able to Do After This Lecture?

1. **Derive** the OLS estimator from the loss function using matrix calculus, and explain each step of the derivation
2. **Analyze** residual plots, multicollinearity diagnostics, and learning curves to detect model violations
3. **Evaluate** when Ridge, Lasso, or Elastic Net regularization is appropriate for a given dataset
4. **Compare** linear regression with gradient-descent-based optimization and select the right approach for the problem scale

---

All four objectives target Bloom's levels 4–5 (Analyze, Evaluate, Compare)

Roadmap: From Scatter Plot to Production Model



Each block builds on the previous one. By the end, you will have a complete toolkit for building, evaluating, and regularizing linear models.

We follow the PMSP framework: Problem – Method – Solution – Practice

How Do We Write the Linear Model in Matrix Form?

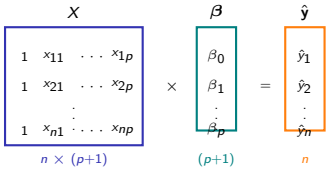
Scalar form (single observation):

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}$$

Matrix form (all n observations):

$$\hat{\mathbf{y}} = X\boldsymbol{\beta}$$

- X is the $n \times (p{+}1)$ **design matrix** (column of 1s for intercept)
- $\boldsymbol{\beta}$ is the $(p{+}1) \times 1$ coefficient vector
- $\hat{\mathbf{y}}$ is the $n \times 1$ prediction vector



The column of 1s in X absorbs the intercept β_0 into the matrix product – no special case needed

How Does OLS Find the Best Line?

Step 1: Define the loss

$$L(\boldsymbol{\beta}) = \|\mathbf{y} - X\boldsymbol{\beta}\|^2 = (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta})$$

Step 2: Expand

$$L = \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T X^T\mathbf{y} + \boldsymbol{\beta}^T X^T X\boldsymbol{\beta}$$

Step 3: Differentiate and set to zero

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = -2X^T\mathbf{y} + 2X^T X\boldsymbol{\beta} = \mathbf{0}$$

Step 4: Solve the normal equations

$$X^T X\boldsymbol{\beta} = X^T\mathbf{y} \quad \Longrightarrow \quad \boxed{\hat{\boldsymbol{\beta}} = (X^T X)^{-1}X^T\mathbf{y}}$$

The loss is convex (quadratic), so this unique solution is the global minimum. Requires $X^T X$ invertible (no perfect multicollinearity).

Can We Compute OLS by Hand?

Data: 4 houses with size (100s sqft) $\rightarrow$ price ($1000s)

i	x_i	y_i
1	1	2
2	2	3
3	3	5
4	4	6

$$X = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 2 \\ 3 \\ 5 \\ 6 \end{pmatrix}$$

$$X^T X = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}, \quad X^T \mathbf{y} = \begin{pmatrix} 16 \\ 49 \end{pmatrix}$$

$$(X^T X)^{-1} = \tfrac{1}{20} \begin{pmatrix} 30 & -10 \\ -10 & 4 \end{pmatrix}$$

$$\hat{\boldsymbol{\beta}} = \tfrac{1}{20} \begin{pmatrix} 30 & -10 \\ -10 & 4 \end{pmatrix} \begin{pmatrix} 16 \\ 49 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 1.6 \end{pmatrix}$$

Result: $\hat{y} = -0.5 + 1.6x$
$R^2 = 1 - \frac{1.40}{10} = 0.86$

Verify: $\hat{y}_1 = 1.1$ **(residual** 0.9**),** $\hat{y}_2 = 2.7$ **(residual** 0.3**),** $\hat{y}_3 = 4.3$ **(residual** 0.7**),** $\hat{y}_4 = 5.9$ **(residual** 0.1**)**

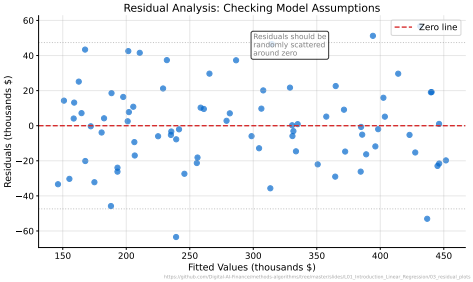
How Do We Know If the Model Is Any Good?

R^2: Proportion of variance explained

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

- $R^2 = 1$: perfect predictions
- $R^2 = 0$: no better than predicting the mean
- **Adjusted R^2** penalizes adding useless features:

$$R^2_{\text{adj}} = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$



Warning

High R^2 does not mean correct model. Always inspect residuals.

R^2 can only increase with more features – adjusted R^2 guards against overfitting by penalizing model complexity

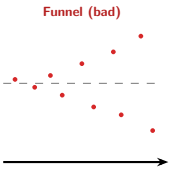
What Do Residual Patterns Tell Us?

Good residuals (random scatter):

- No visible pattern around zero
- Constant spread (homoscedasticity)
- Approximately normal distribution

Bad residuals (systematic patterns):

- **Funnel shape**: heteroscedasticity – variance changes with $\hat{y}$
- **Curved pattern**: non-linearity – need polynomial or transform
- **Clusters**: missing categorical variable



LINE assumptions: Linearity, Independence, Normality of residuals, Equal variance (homoscedasticity)

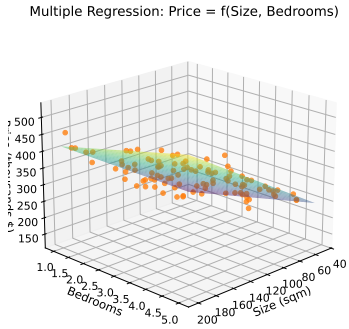
What Changes When We Add More Features?

From 1D line to p-dimensional hyperplane

- With p features, OLS fits a hyperplane in $(p+1)$-dimensional space
- Each β_j measures the effect of x_j *holding all other features constant* (ceteris paribus)
- **Multicollinearity**: if features are correlated, coefficient estimates become unstable

Variance Inflation Factor

$\text{VIF}_j = \frac{1}{1-R_j^2}$ where R_j^2 is from regressing x_j on all other features. VIF > 10 signals serious collinearity.



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L01_Introduction_Linear_Regression/02_multiple_regression_3d

In 3D the regression surface is a plane. In higher dimensions it is a hyperplane – the math is identical.

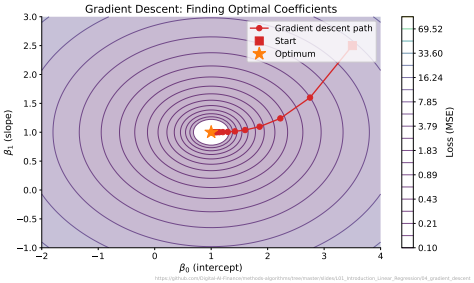
Why Not Always Use the Closed-Form Solution?

OLS cost: $O(np^2 + p^3)$ for matrix inversion
When n or p is very large, this becomes prohibitive.

Gradient descent alternative:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \alpha \nabla L(\boldsymbol{\beta}_t)$$

- α is the **learning rate**
- $\nabla L = -\frac{2}{n} X^T(\mathbf{y} - X\boldsymbol{\beta})$ (gradient of MSE)
- Each step costs $O(np)$ – scales linearly
- Converges to the same solution as OLS



OLS: exact but $O(p^3)$. Gradient descent: iterative but $O(np)$ per step. For $n > 10{,}000$ or $p > 1{,}000$, GD wins.

Which Flavor of Gradient Descent Should You Use?

Batch GD

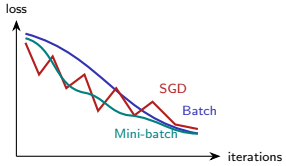
- Uses *all* n samples per step
- Smooth convergence, expensive per step
- Best for small-to-medium data

Stochastic GD (SGD)

- Uses *one* random sample per step
- Noisy updates, very fast per step
- Can escape local minima (non-convex)

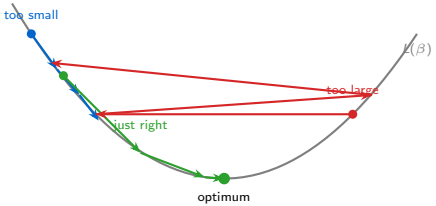
Mini-batch GD

- Uses a batch of b samples (e.g., $b = 32$)
- Best of both: stable yet fast
- Standard choice in deep learning



For linear regression (convex loss), all three converge to the same optimum. The choice is about computational cost.

How Does the Learning Rate Affect Convergence?



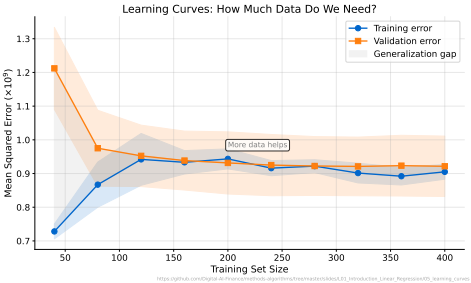
- **Too small**: converges but wastes computation – may need thousands of steps
- **Just right**: reaches the optimum in a few steps
- **Too large**: overshoots and diverges – the loss increases

In practice, use learning rate schedules or adaptive methods (Adam, RMSProp) that adjust α automatically

How Do Training and Validation Error Behave?

Learning curves plot error vs. training set size:

- **Training error** starts low, increases with more data (harder to memorize)
- **Validation error** starts high, decreases with more data (better generalization)
- The **gap** between them diagnoses the problem



Diagnostic Rules

High bias: both curves plateau high → model too simple
High variance: large gap → model too complex, need more data or regularization

Learning curves are the single best diagnostic for deciding whether you need more data, more features, or regularization

What Is the Bias-Variance Tradeoff?

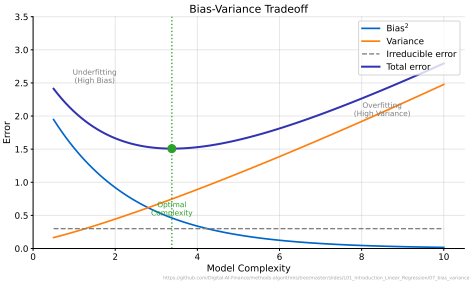
Expected test error decomposes as:

$$\text{MSE} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

- **Bias**: error from wrong assumptions (e.g., fitting a line to a curve)
- **Variance**: sensitivity to training data fluctuations
- As model complexity increases: bias decreases, variance increases
- The optimum is where their sum is minimized

Key Insight

Regularization trades a small increase in bias for a large decrease in variance, improving test error.



An overfit model has low bias but high variance. An underfit model has high bias but low variance. Regularization helps find the sweet spot.

Why Shrink Coefficients Toward Zero?

Problem: with many features, OLS overfits – coefficients become large and unstable.

Solution: add a **penalty term** to the loss function that punishes large coefficients.

Ridge Regression (L2)

$$L_{\text{Ridge}} = \|\mathbf{y} - X\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Shrinks all coefficients toward zero
- Never sets them *exactly* to zero
- Closed-form: $\hat{\boldsymbol{\beta}} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$

Lasso Regression (L1)

$$L_{\text{Lasso}} = \|\mathbf{y} - X\boldsymbol{\beta}\|^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

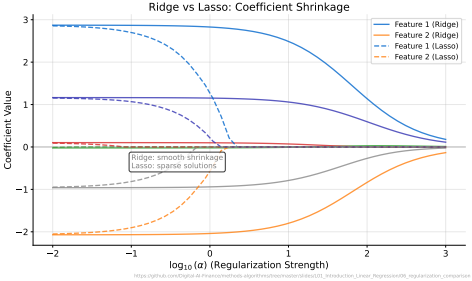
- Drives some coefficients to **exact zero**
- Performs automatic feature selection
- No closed-form – requires iterative solver

λ controls regularization strength: $\lambda = 0$ is OLS; $\lambda \to \infty$ gives $\boldsymbol{\beta} = \mathbf{0}$.

Always standardize features before regularization – otherwise the penalty treats large-scale features differently from small-scale ones

Why Does Lasso Produce Sparse Solutions?

- **Ridge**: constraint region is a circle ($\sum \beta_j^2 \leq t$)
- **Lasso**: constraint region is a diamond ($\sum |\beta_j| \leq t$)
- OLS contours touch the diamond at *corners* – where some $\beta_j = 0$
- **Elastic Net**: $\lambda_1 \sum |\beta_j| + \lambda_2 \sum \beta_j^2$ – combines both



Use Lasso when you expect many irrelevant features. Use Ridge when all features contribute. Use Elastic Net when unsure.

How Do We Implement This in Python?

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler

# Prepare data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)

# OLS
ols = LinearRegression().fit(X_scaled, y_train)
print(f"OLS R2: {ols.score(X_scaled, y_train):.3f}")

# Ridge (L2) with cross-validation
ridge = Ridge(alpha=1.0).fit(X_scaled, y_train)
cv_r2 = cross_val_score(ridge, X_scaled, y_train, cv=5)
print(f"Ridge CV R2: {cv_r2.mean():.3f} +/- {cv_r2.std():.3f}")

# Lasso (L1) -- sparse solution
lasso = Lasso(alpha=0.1).fit(X_scaled, y_train)
n_zero = sum(lasso.coef_ == 0)
print(f"Lasso: {n_zero}/{len(lasso.coef_)} features zeroed out")
```

Always scale features before Ridge/Lasso. Use cross-validation (RidgeCV, LassoCV) to tune λ automatically.

How Does Finance Use Linear Regression Every Day?

Capital Asset Pricing Model (CAPM)

$$R_i - R_f = \alpha + \beta(R_m - R_f) + \epsilon$$

- β: sensitivity to market risk (systematic)
- α: excess return beyond market exposure
- $\alpha > 0$ signals manager skill

Fama-French 3-Factor Model

$$R_i - R_f = \alpha + \beta_1(R_m - R_f) + \beta_2 \cdot \text{SMB} + \beta_3 \cdot \text{HML} + \epsilon$$

- SMB: Small Minus Big (size premium)
- HML: High Minus Low (value premium)

Interpretation:

$\beta_1 = 1.2 \rightarrow$ stock moves 1.2% per 1% market move

$\beta_2 = 0.5 \rightarrow$ tilted toward small caps

$\beta_3 = -0.3 \rightarrow$ tilted toward growth

$\alpha = 0.02 \rightarrow$ 2% annual excess return

Extensions: Carhart 4-factor (momentum), Fama-French 5-factor (profitability, investment)

CAPM (Sharpe, 1964) won the Nobel Prize. Every hedge fund's risk report is built on these regressions.

Can We Decompose a Fund's Returns Into Risk Factors?

Scenario: Fund XYZ returned 12% annually. Is the manager skilled, or just taking risk?

```
import statsmodels.api as sm
# Monthly returns: fund vs factors (60 months)
X = sm.add_constant(factors_df[['Mkt-RF','SMB','HML']])
model = sm.OLS(fund_returns - rf, X).fit()
print(model.summary())
```

Factor	Coef	Std Err	t-stat	p-value
α (intercept)	0.001	0.002	0.50	0.62
β_{Mkt} (market)	1.15	0.08	14.4	< 0.001
β_{SMB} (size)	0.45	0.12	3.75	< 0.001
β_{HML} (value)	-0.20	0.10	-2.00	0.05

Verdict: α is not significant ($p = 0.62$). The 12% return is explained by high market beta and small-cap tilt – no evidence of skill.

Regression-based factor decomposition is the foundation of performance attribution in asset management

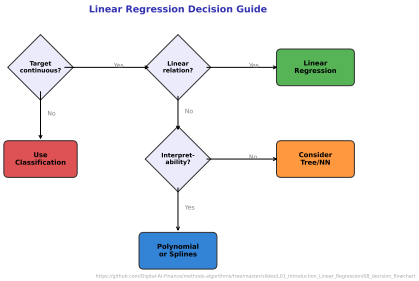
When Should You Use Linear Regression?

Choose linear regression when:

- Target variable is continuous
- Relationship is approximately linear
- Interpretability of coefficients matters
- Fast training is needed

Consider alternatives when:

- Non-linear patterns in residuals → try polynomial features, trees, or neural nets
- Binary or categorical target → logistic regression (L02)
- Many irrelevant features → Lasso or tree-based methods
- Complex interactions → random forests (L04)



Linear regression is always a strong baseline. Start here, check residuals, then decide if you need something more complex.

Five Key Takeaways

1. **The linear model** $\hat{\mathbf{y}} = X\boldsymbol{\beta}$ has a unique closed-form solution $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \mathbf{y}$ – derived by minimizing squared residuals
2. **Model evaluation** requires both metrics (R^2, adjusted R^2) and visual diagnostics (residual plots, learning curves)
3. **Gradient descent** is the scalable alternative to OLS – essential when data is large or the model is extended to non-linear forms
4. **Regularization** (Ridge, Lasso, Elastic Net) prevents overfitting by trading a small bias increase for a large variance reduction
5. **Finance applications**: CAPM and factor models are linear regressions – β coefficients decompose risk and return

---

Master these five ideas and you have the foundation for all of machine learning

References and Further Reading

Textbooks

- James, Witten, Hastie, Tibshirani (2021). *Introduction to Statistical Learning*, Ch. 3. Free at `statlearning.com`
- Hastie, Tibshirani, Friedman (2009). *Elements of Statistical Learning*, Ch. 3. Free at `hastie.su.domains`

Finance

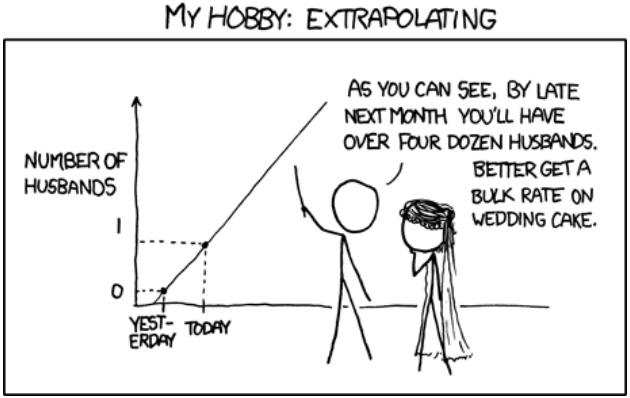
- Fama, French (1993). Common Risk Factors in the Returns on Stocks and Bonds. *J. Financial Economics*.
- Sharpe (1964). Capital Asset Prices. *Journal of Finance*.

Practice

- Course notebook: `L01_linear_regression.ipynb` (housing data)
- sklearn documentation: `LinearRegression`, `Ridge`, `Lasso`

Start with ISLR Chapter 3 for intuition, then ESL Chapter 3 for the math

The Danger of Extrapolation



What happens when you trust your linear model beyond the range of the training data?

XKCD #605 by Randall Munroe (CC BY-NC 2.5)