Supervised Learning I: Linear Regression & Optimization

Zuu Crew
Machine Learning
Acadamy

Today's Agenda

Fundamentals

- Supervised learning concepts
- Regression vs. classification
- Real-world applications

Linear Regression

- Mathematical foundations
- Optimization techniques
- Modeling FOML student success

Polynomial Regression

- Non-linear relationships
- Prod-Ready ML Systems cohort analysis
- Model comparison & evaluation

What is Supervised Learning?

Supervised learning trains a model to predict an output from labeled input data.

- **Input Features**

  Student behaviors (attendance, projects completed, platform usage)

- **Output Target**

  Capstone Score (0-100 scale)

- **Goal**

  Generalize to new students and estimate performance accurately

Regression Use Case – Foundations of ML Cohort

We want to predict capstone scores for students enrolled in the **Foundations of ML** course using **linear regression**.

This helps us model simple, interpretable relationships between features and outcomes.

Key benefits:

- Clear interpretation of feature importance
- Transparent model behavior
- Easy implementation for baseline predictions
- Foundation for more complex models

Why Predict Zuu Crew Scores?

Understand Success Drivers

Identify which behaviors and activities most strongly correlate with student achievement

Personalized Support

Provide early feedback and targeted interventions for at-risk students

Program Evaluation

Assess the effectiveness of different program components based on their impact on outcomes

Data-Driven Improvement

Enhance future cohorts by focusing resources on the most impactful activities

What is a Line of Best Fit?

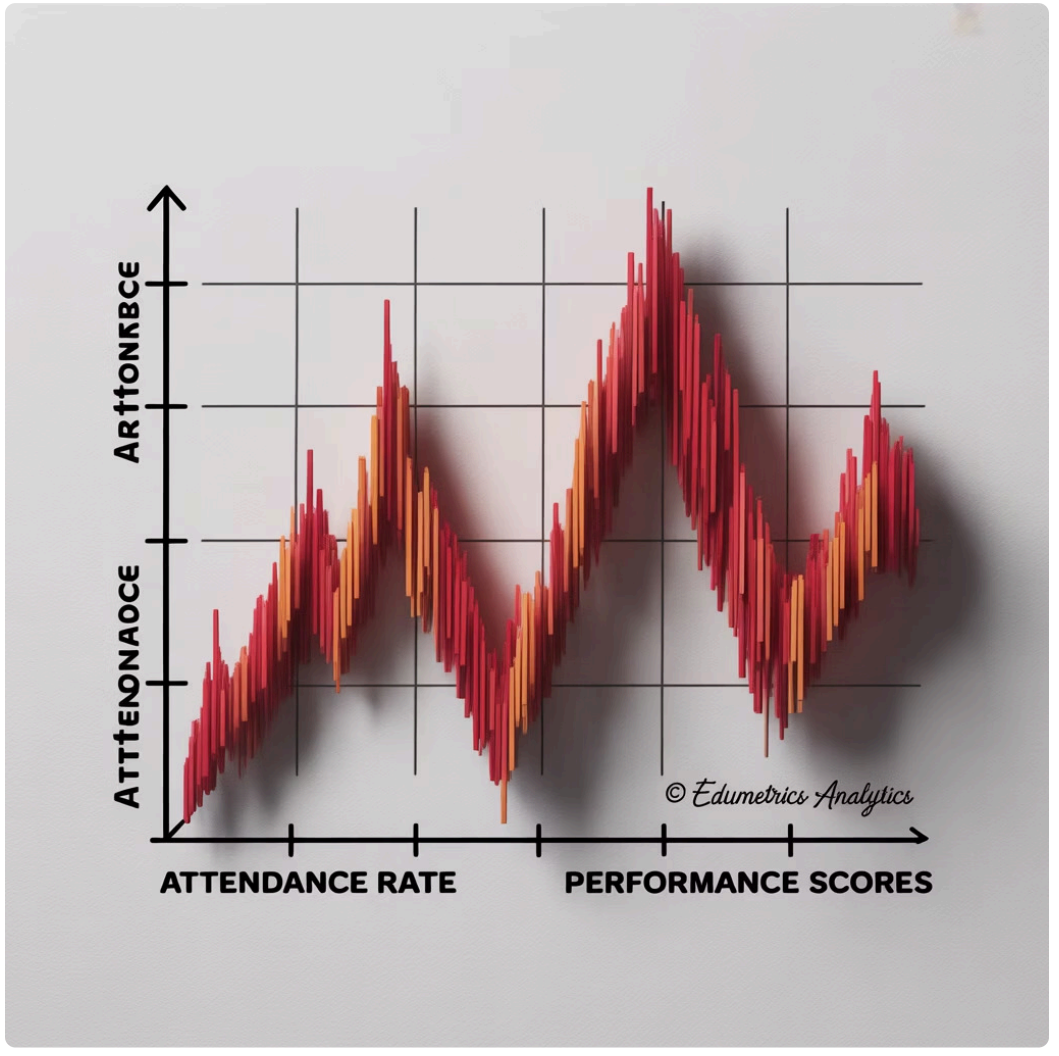
Linear regression draws a straight line through data that minimizes error.

$$CapstoneScore = \beta_0 + \beta_1 \times Attendance$$

Where:

- β_0 = y-intercept (baseline score)
- β_1 = slope (impact of attendance)

A good line predicts scores close to actual student performance.



Introducing the Dataset

Zuu Crew collected the following features per student:

Student Demographics

- Member Name
- Education Level (A/L, UG, PG)
- Course Name (FOML or Prod-Ready)

Engagement Metrics

- Attendance (%)
- Total Hours on Platform
- Assignments Completed

Participation Indicators

- Hackathon Participation
- GitHub Activity Score
- Peer Review Score

These variables will serve as our predictors in the regression models.

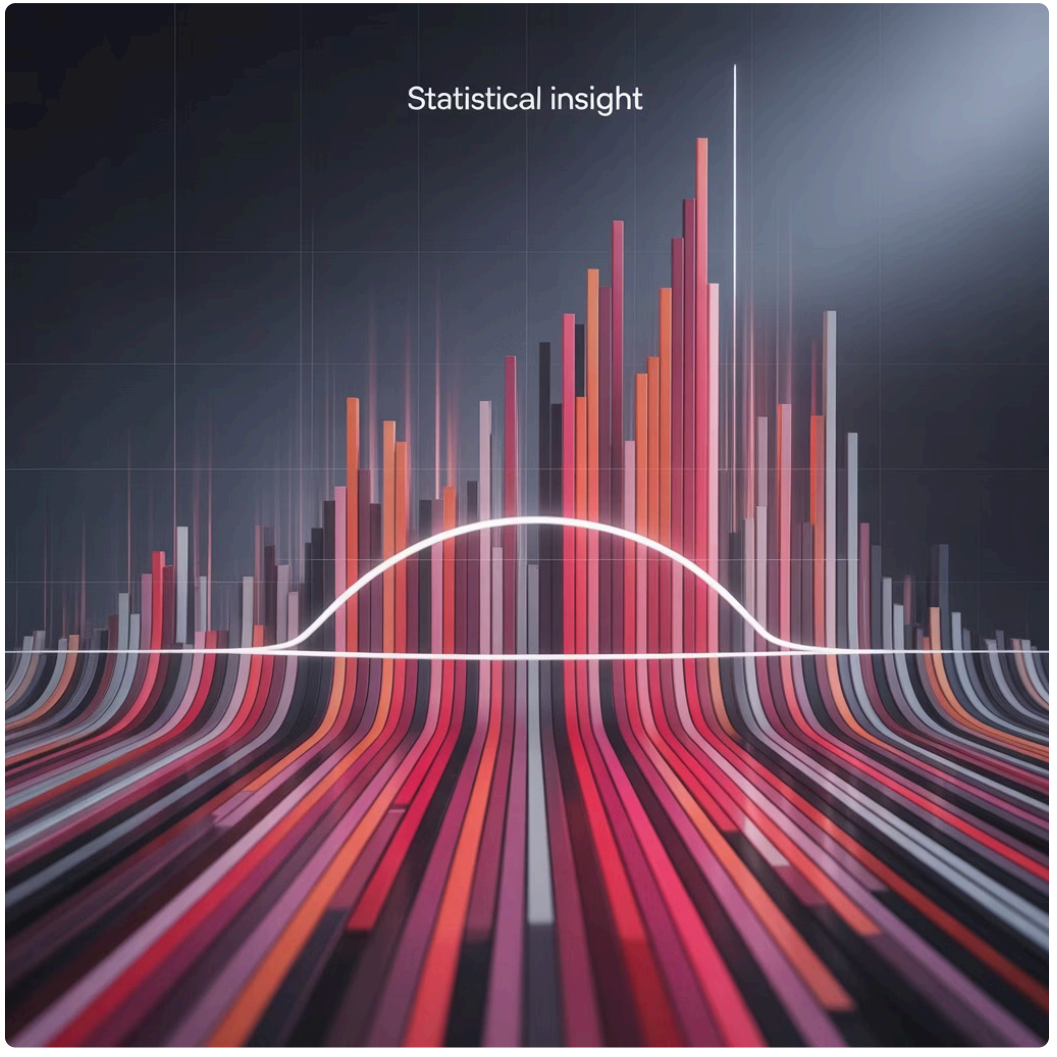
The Target – Capstone Score

Score Range: 0 to 100

The capstone score reflects final performance in the course and serves as our prediction target.

It is influenced by multiple academic and behavioral factors:

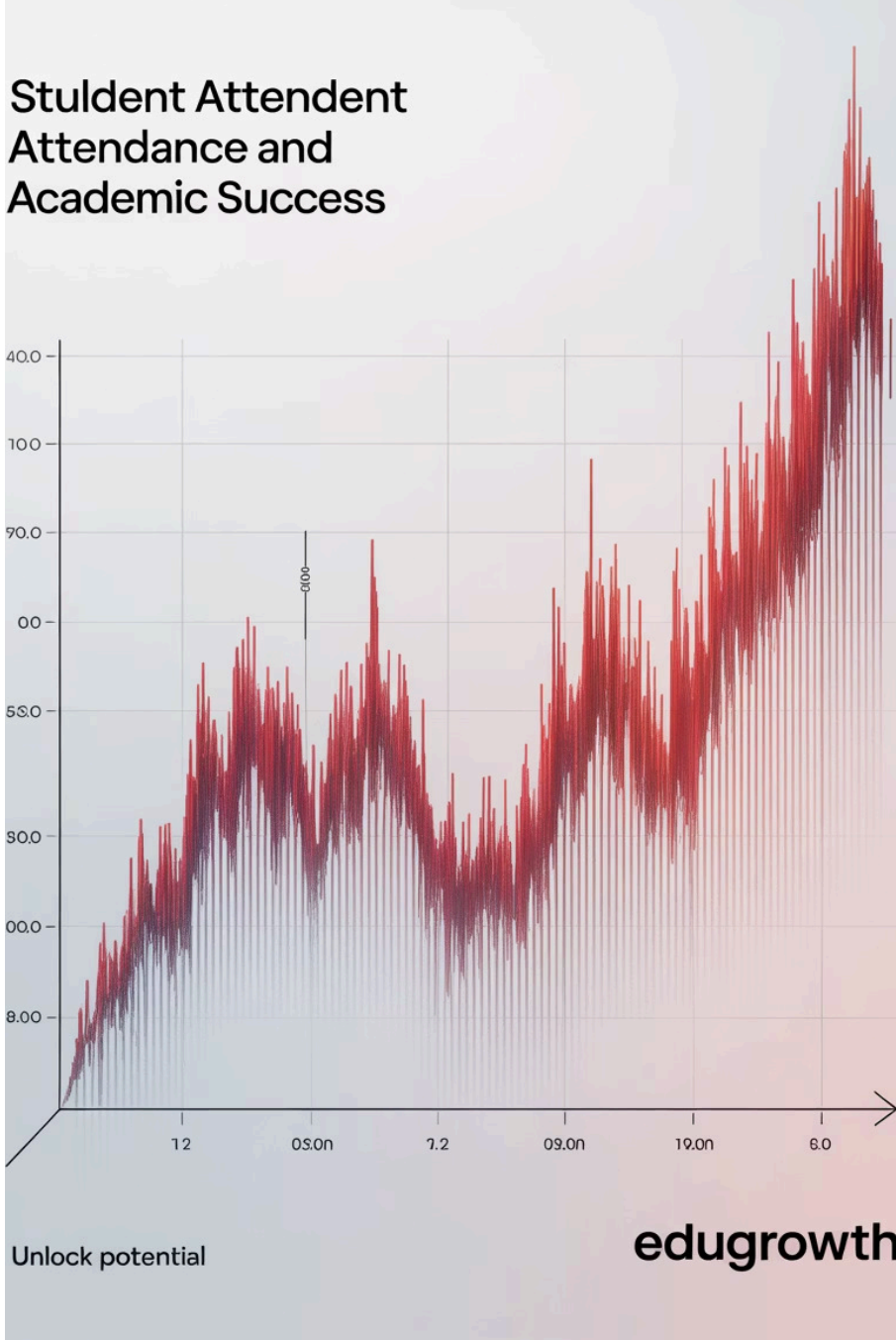
- Technical knowledge demonstration
- Project implementation quality
- Documentation and communication
- Problem-solving approach
- Code quality and reproducibility



Example – Attendance vs Capstone Score

The scatter plot shows a clear positive correlation: higher attendance generally leads to higher capstone scores.

This relationship appears roughly linear, making it a good candidate for linear regression modeling.



Model Hypothesis – Linear Form

Linear regression assumes the outcome is a **weighted sum of inputs**:

$$Score = \beta_0 + \beta_1 \cdot Attendance + \beta_2 \cdot Hours + \beta_3 \cdot Assignments + \dots$$

Where:

- β_0 = Intercept (base score with zero for all features)
- β_1, β_2, ... = Coefficients (weights for each feature)

The model learns these coefficients to minimize prediction error across all students.

What Is Error in Prediction?

Residual = Actual Score – Predicted Score

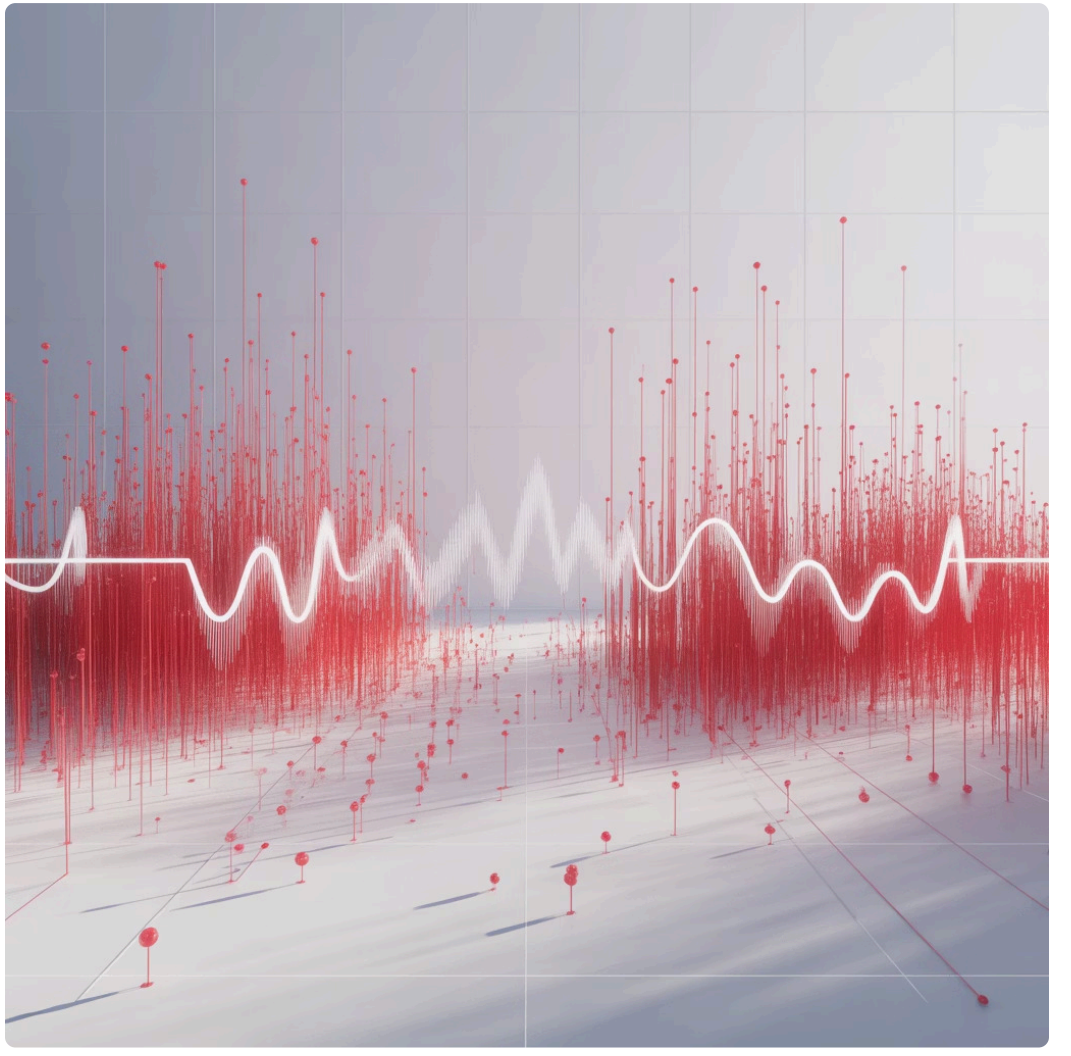
$$e_i = \hat{y}_i - y_i$$

Where:

- e_i = error for student i
- ŷ_i = predicted capstone score
- y_i = actual capstone score

Goal: Minimize errors across all students

- Too large → poor fit
- Balanced small residuals → good fit



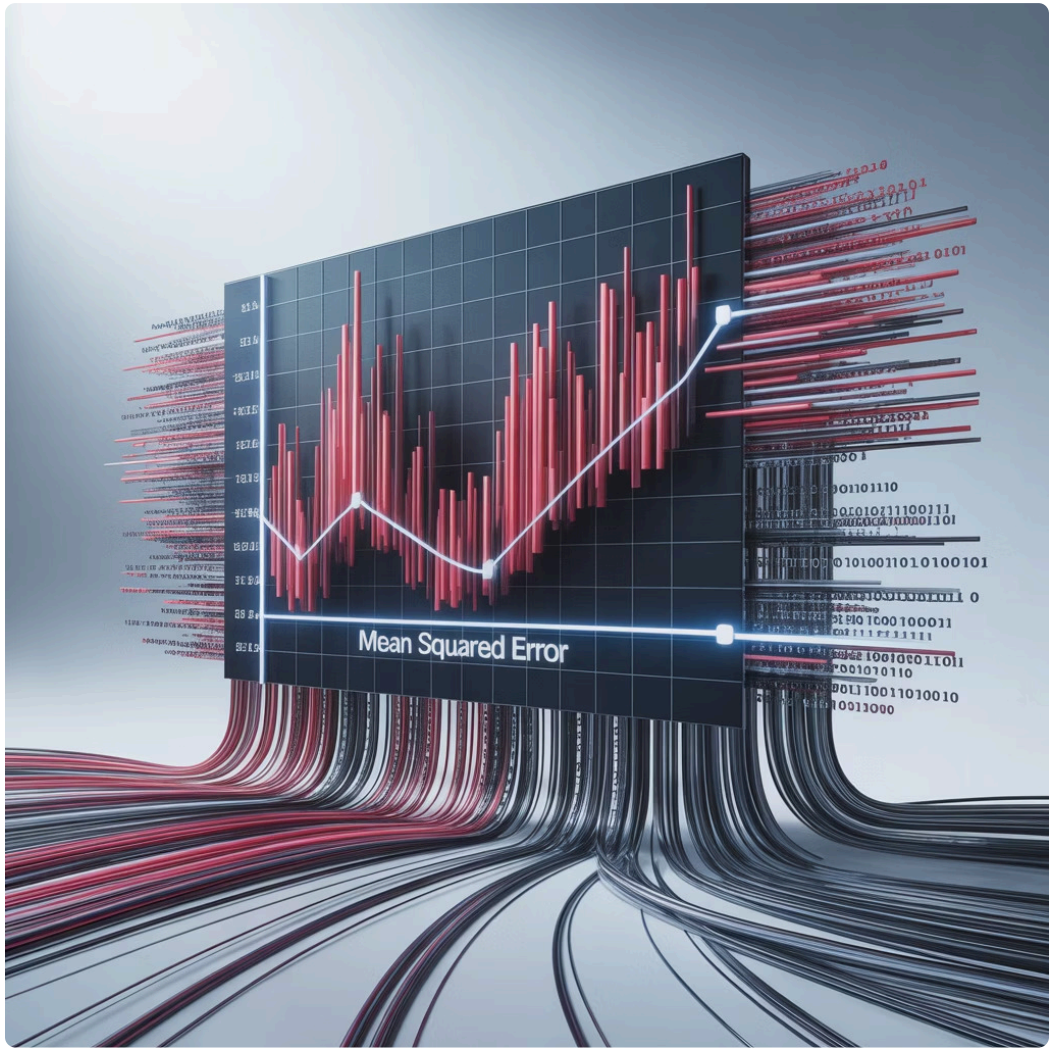
Mean Squared Error (MSE)

MSE is our cost function for regression:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

Why square the errors?

- Prevents negative errors from canceling positives
- Penalizes large errors more heavily
- Creates a differentiable function for optimization
- Results in a convex surface with single minimum



Mathematical Implementation: The Cost Function

For linear regression with multiple features, we define:

$$h_\beta(x^{(i)}) = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + ... + \beta_n x_n^{(i)}$$

Our cost function J(β) measures how well our hypothesis fits the data:

$$J(\beta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})^2$$

Where:

- m = number of training examples (students)
- x^(i) = feature vector for student i
- y^(i) = actual capstone score for student i

The factor of 1/2 is a convention that makes derivative calculations cleaner.

Cost Surface Visualization

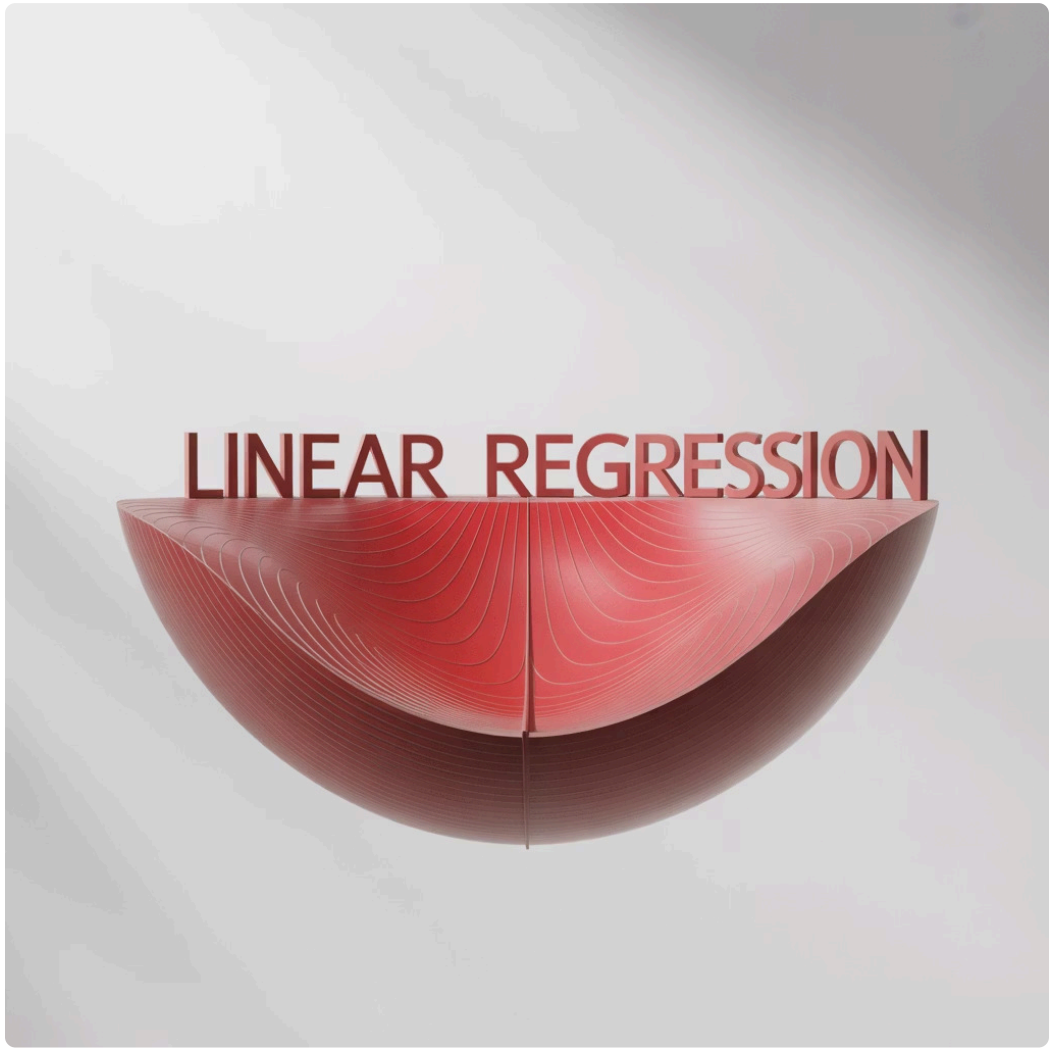
Each pair of parameters (θ_0, θ_1) produces a specific cost value.

All these costs form a 3D "bowl-shaped" surface.

Properties of this surface:

- The bottom point = optimal solution
- Steepness = gradient magnitude
- Direction of steepest descent = negative gradient

Our goal: Find the bottom of this bowl algorithmically.

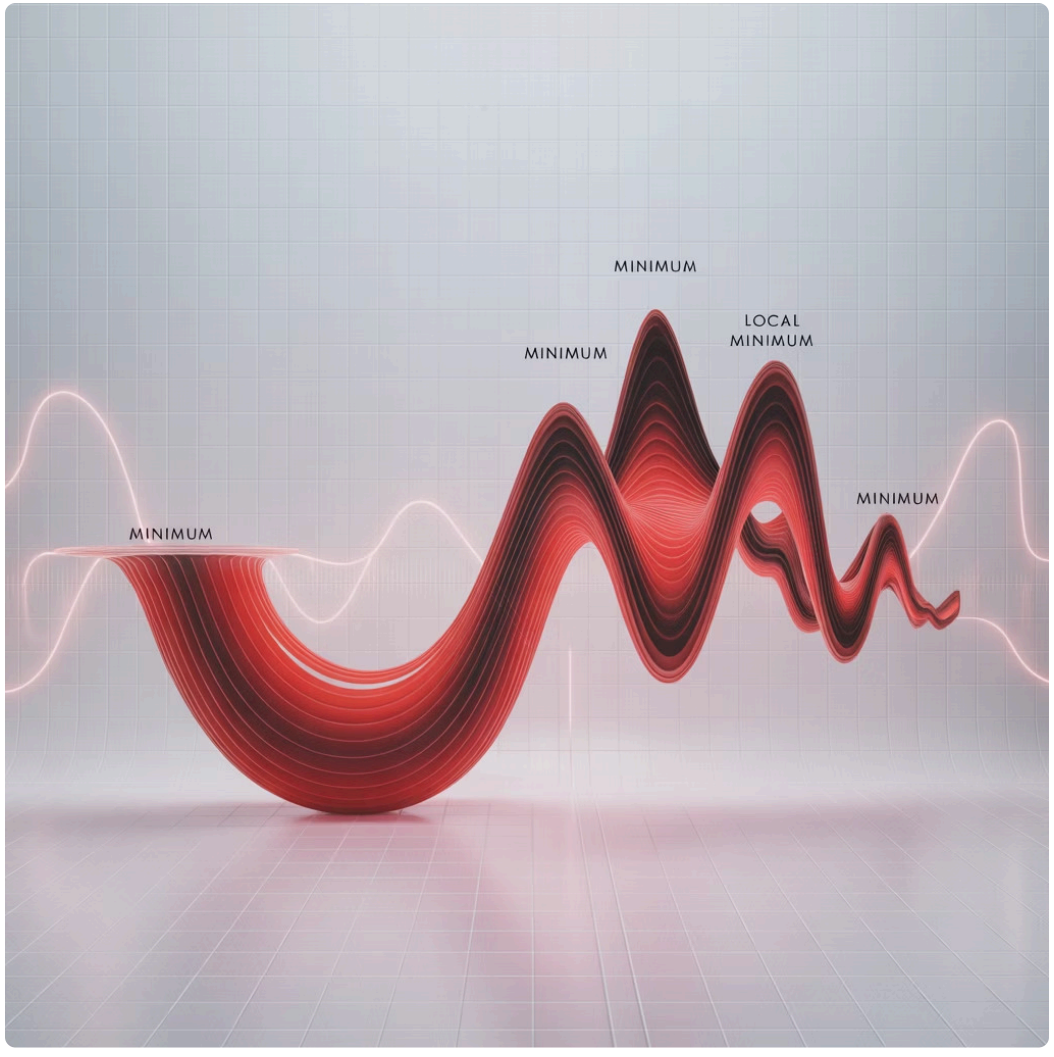


Why Convexity Matters

MSE is a **convex function** for linear regression, which guarantees:

- One global minimum (no false solutions)
- No local minima to get trapped in
- Gradient descent always converges to the best point
- Different starting points reach same solution

This property makes linear regression optimization reliable and consistent.



Mathematical Implementation: Partial Derivatives

To minimize J(β), we need its partial derivatives with respect to each parameter:

$$\frac{\partial J(\beta)}{\partial \beta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

For β_0 specifically (where $x_0^{(i)} = 1$ for all i):

$$\frac{\partial J(\beta)}{\partial \beta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\beta(x^{(i)}) - y^{(i)})$$

These derivatives tell us the direction and magnitude to adjust each parameter to reduce the cost.

Gradient Descent Overview

To find optimal weights (β), we iteratively adjust them to reduce error:

$$\beta_j := \beta_j - \eta \cdot \frac{\partial J(\beta)}{\partial \beta_j}$$

Where:

- η = learning rate (controls step size)
- ∇J(β) = gradient of cost function
- := denotes parameter update

Each iteration moves β closer to the optimal solution.

Mathematical Implementation: Gradient Descent Algorithm

The complete algorithm for linear regression with gradient descent:

```
Repeat until convergence {
   For j = 0 to n {
      β_j := β_j - α · (1/m) · Σ[(h_β(x^(i)) - y^(i)) · x_j^(i)]
   }
}
```

For our Zuu Crew student prediction model:

$$\beta_j := \beta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} ((\beta_0 + \beta_1 \cdot Attendance^{(i)} + \beta_2 \cdot Hours^{(i)} + ...) - Score^{(i)}) \cdot Feature_j^{(i)}$$

All parameters must be updated simultaneously using values from the previous iteration.

Analogy – Ball Rolling Down a Hill

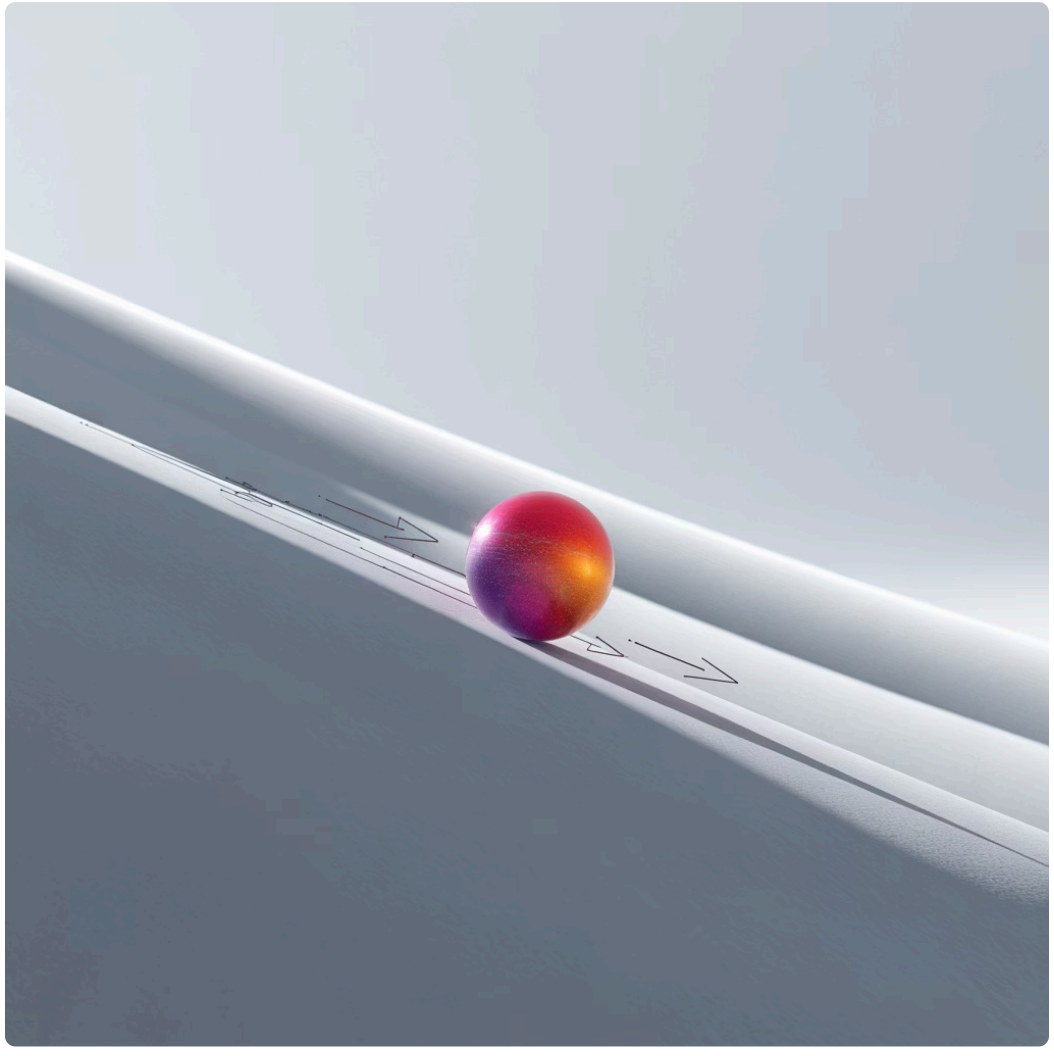
Gradient descent can be visualized as a ball rolling down a hill:

- The ball = current parameter values
- The hill = cost surface
- Steepness = gradient magnitude
- Path taken = negative gradient direction

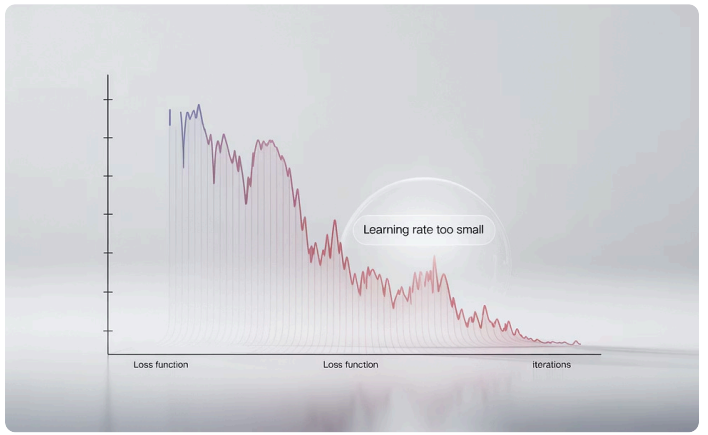
Each iteration moves θ a little closer to the optimal point.

The steeper the slope, the bigger the step.

Eventually, steps become tiny as we reach the bottom (convergence).

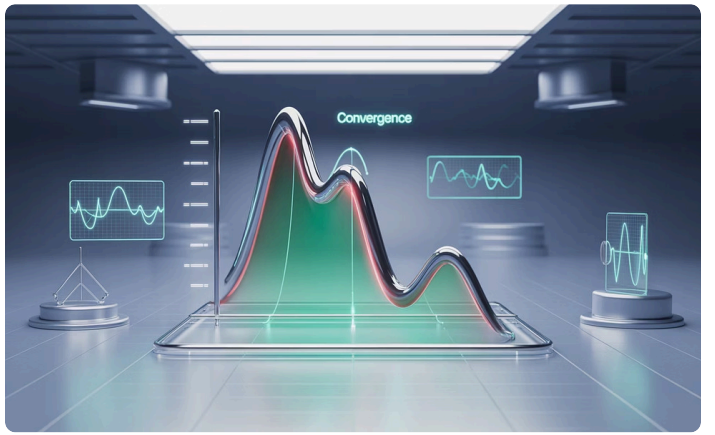


Learning Rate Effect



Too Small (η = 0.001)

Very slow training progress, requires many iterations, computationally inefficient



Just Right (η = 0.1)

Steady progress toward minimum, efficient convergence, balanced approach

Too Large (η = 2.0)

Overshoots minimum, oscillates or diverges completely, fails to converge

Choosing the appropriate learning rate η is critical to training success.

Mathematical Implementation: Normal Equation

For linear regression, we can also solve for optimal parameters analytically:

$$\beta = (X^T X)^{-1} X^T y$$

Where:

- X = design matrix (features for all students)
- y = vector of actual capstone scores
- X^T = transpose of X
- (X^T X)^{-1} = inverse of X^T X

Advantages: No learning rate to tune, no iterations needed

Disadvantages: Computationally expensive with many features, requires matrix inversion

Sample Model Output – Foundations Cohort

After training on FOML student data, we might get:

$$Score = 20 + 0.5 \cdot Attendance + 1.2 \cdot Assignments + 4 \cdot Hackathon$$

Interpretation:

Baseline

20 points is the base score with zero attendance, no assignments, no hackathon

Attendance Impact

Each 1% increase in attendance = +0.5 points on final score

Assignments

Each additional assignment completed = +1.2 points

Hackathon

Participation in hackathon = +4 points boost

Predicting a Student Score

Student A Profile:

- Attendance: 88%
- Assignments: 5 completed
- Hackathon: Yes (participated)

Predicted Score:

$$Score = 20 + (0.5 \times 88) + (1.2 \times 5) + 4$$

$$Score = 20 + 44 + 6 + 4 = 74$$

This student is predicted to score 74 out of 100 on their capstone project.

Comparing Student Profiles

Feature	Student A	Student B	Score Impact
Attendance (%)	88	70	+9 points for A
Assignments	5	3	+2.4 points for A
Hackathon	Yes	No	+4 points for A
Predicted Score	**74**	**58.6**	**15.4 point gap**

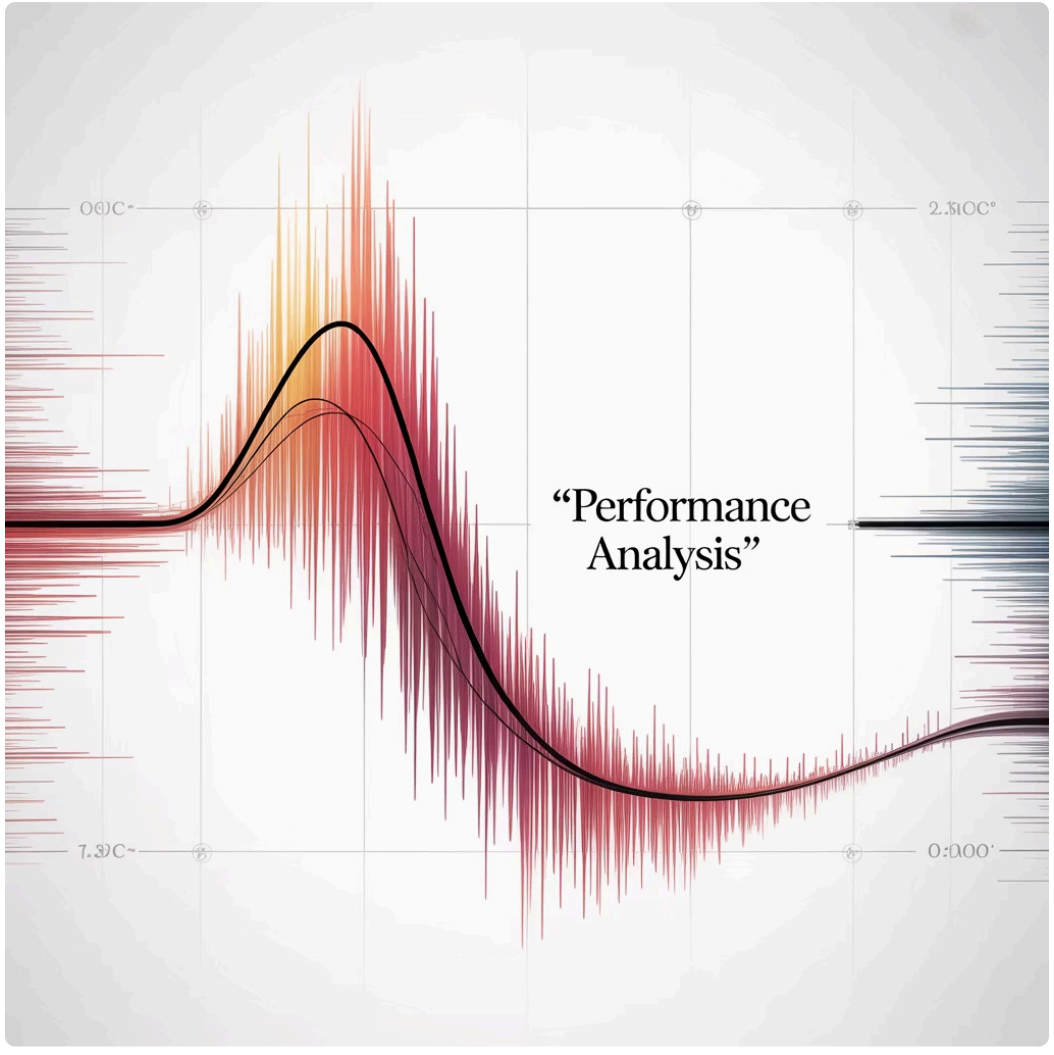
Student A scores significantly higher due to consistent effort across all key performance areas. The linear model clearly quantifies the contribution of each factor.

Enter Polynomial Regression

In **advanced cohorts** like *Production-Ready ML Systems*, linear assumptions often break down:

- Learning curve is non-linear
- More study hours → diminishing returns
- Peer scores → non-linear influence
- Complex interaction effects

Polynomial regression helps model these curved relationships.



Polynomial Regression Intuition

Adds **higher-order terms** to capture curvature:

$$Score = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots$$

This allows the model to represent:

- Upward curves (accelerating returns)
- Plateaus (diminishing returns)
- U-shapes (threshold effects)
- S-curves (learning patterns)

Mathematical Implementation: Polynomial Features

To implement polynomial regression, we transform the original features:

$$X_{poly} = [1, x, x^2, x^3, \ldots, x^n]$$

For multiple features, we can include interaction terms:

$$Score = \beta_0 + \beta_1 \cdot Attendance + \beta_2 \cdot Hours + \beta_3 \cdot Attendance^2 + \beta_4 \cdot Hours^2 + \beta_5 \cdot Attendance \cdot Hours$$

After feature transformation, we apply standard linear regression algorithms:

- Same cost function (MSE)
- Same optimization methods (gradient descent)
- Same implementation, but with transformed features

Visual – Curved Fit

The plot shows study hours vs. capstone score for Production-Ready ML students:

- Linear fit (dashed line) misses the subtle patterns in the data
- Polynomial regression (solid curve) captures **initial acceleration** and **eventual saturation**
- Few students benefit from extremely long study sessions (diminishing returns)

Two Cohorts, Two Fits

Course	Model Type	Example Fit	Interpretation
Foundations of ML	Linear	Score = $\beta_0 + \beta_1 \times$ Attendance + ...	Score steadily rises with effort
Prod-Ready ML	Polynomial	Score = $\beta_0 + \beta_1 \times$ Hours + $\beta_2 \times$ Hours^2 + $\beta_3 \times$ Hours^3	Score accelerates then plateaus

Model Comparison Summary

1

Linear Regression

- Fit type: Straight line
- Interpretability: High
- Data complexity: Low to medium
- Primary risk: Underfitting complex patterns
- Best for: Foundations cohort with clear linear trends

2

Polynomial Regression

- Fit type: Curve (quadratic, cubic)
- Interpretability: Moderate
- Data complexity: Medium to high
- Primary risk: Overfitting if degree too high
- Best for: Production-Ready cohort with non-linear patterns