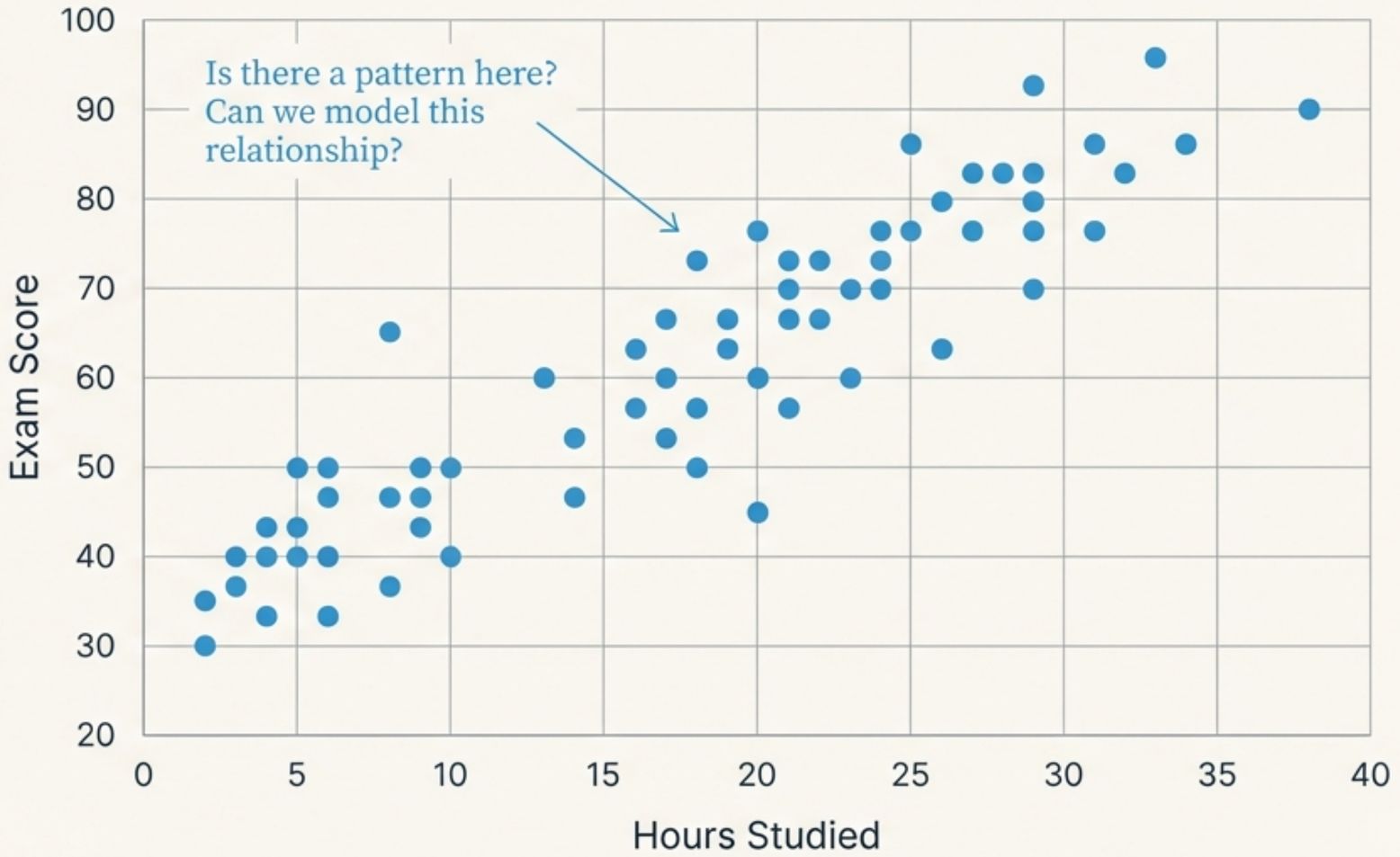
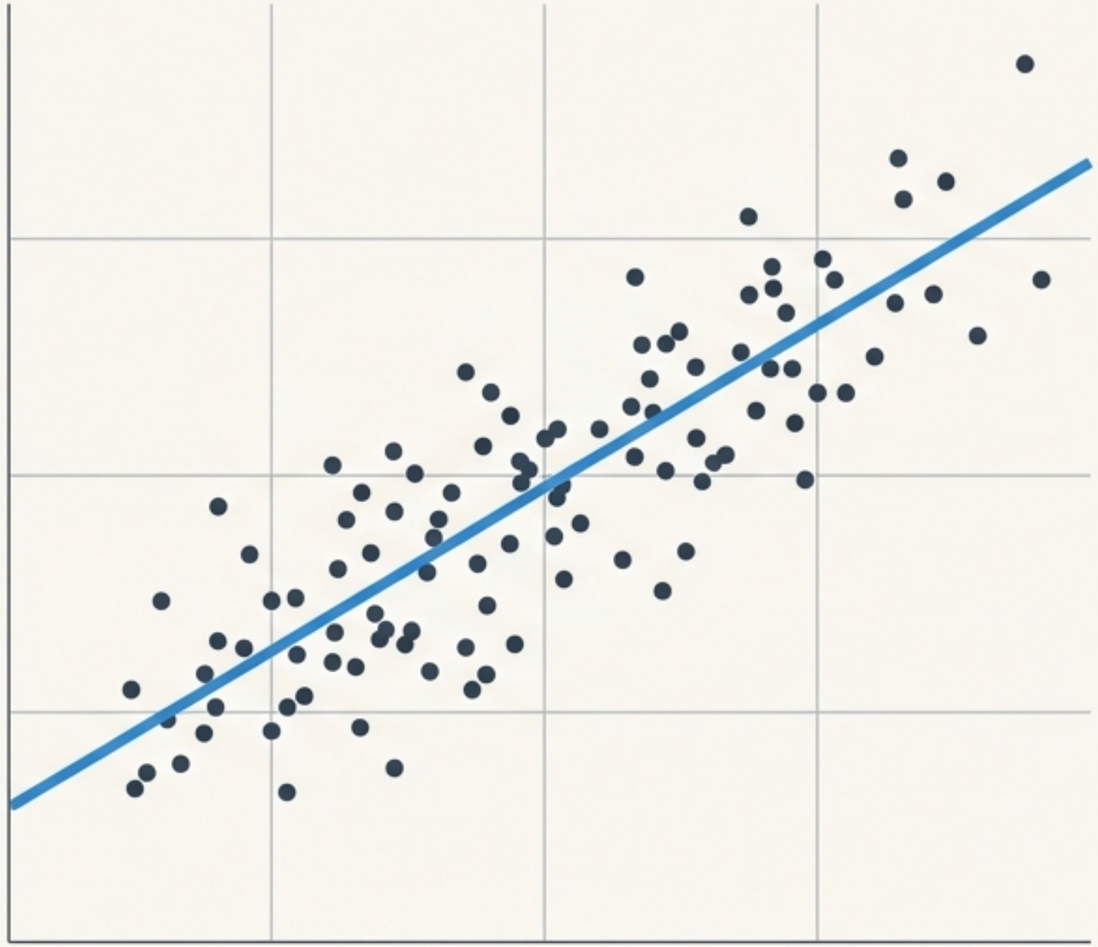
How can we predict an outcome based on what we know?

We often want to understand the relationship between two things. For example, can we predict a student's exam score based on the hours they study?



We can describe the relationship with a straight line.



Independent Variable (Input/Predictor)

The factor we control or observe.

Example: Hours Studied

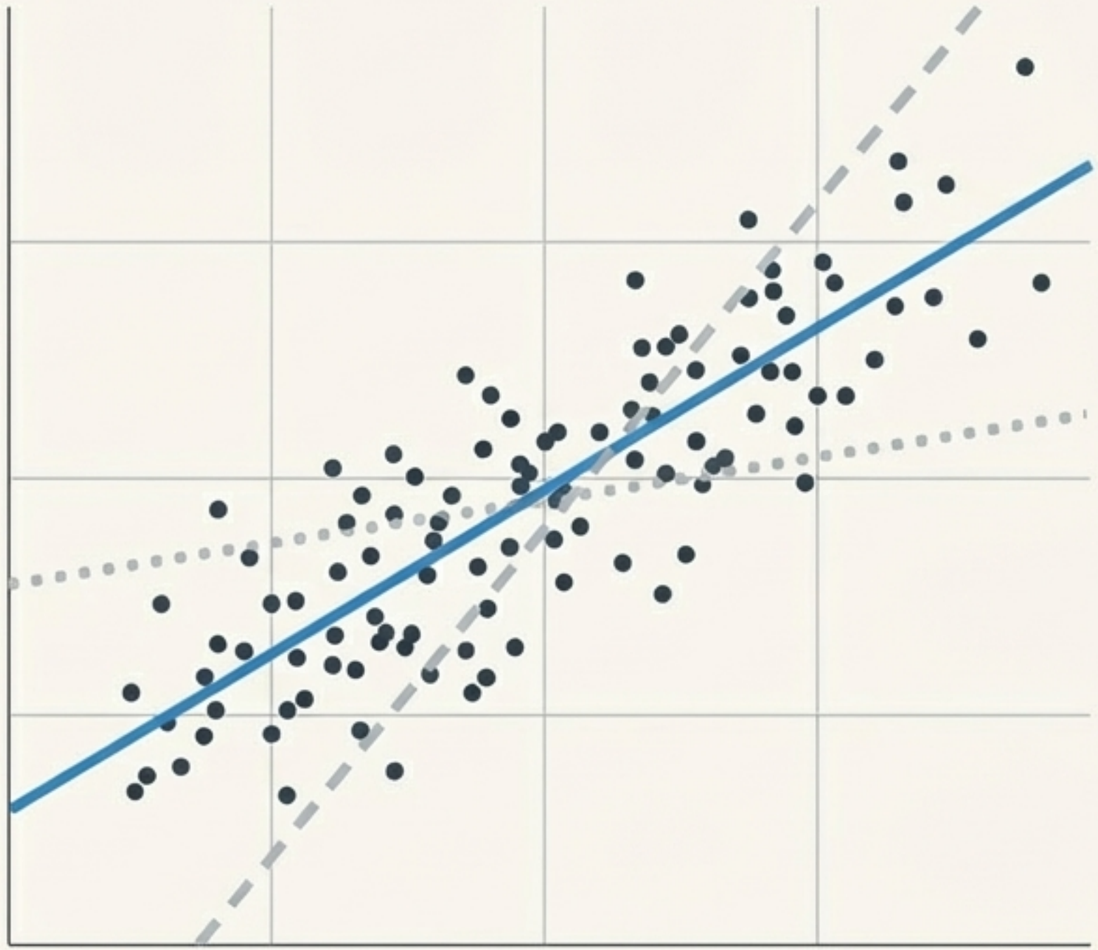
Dependent Variable (Output/Target)

The outcome that depends on the input.

Example: Exam Score

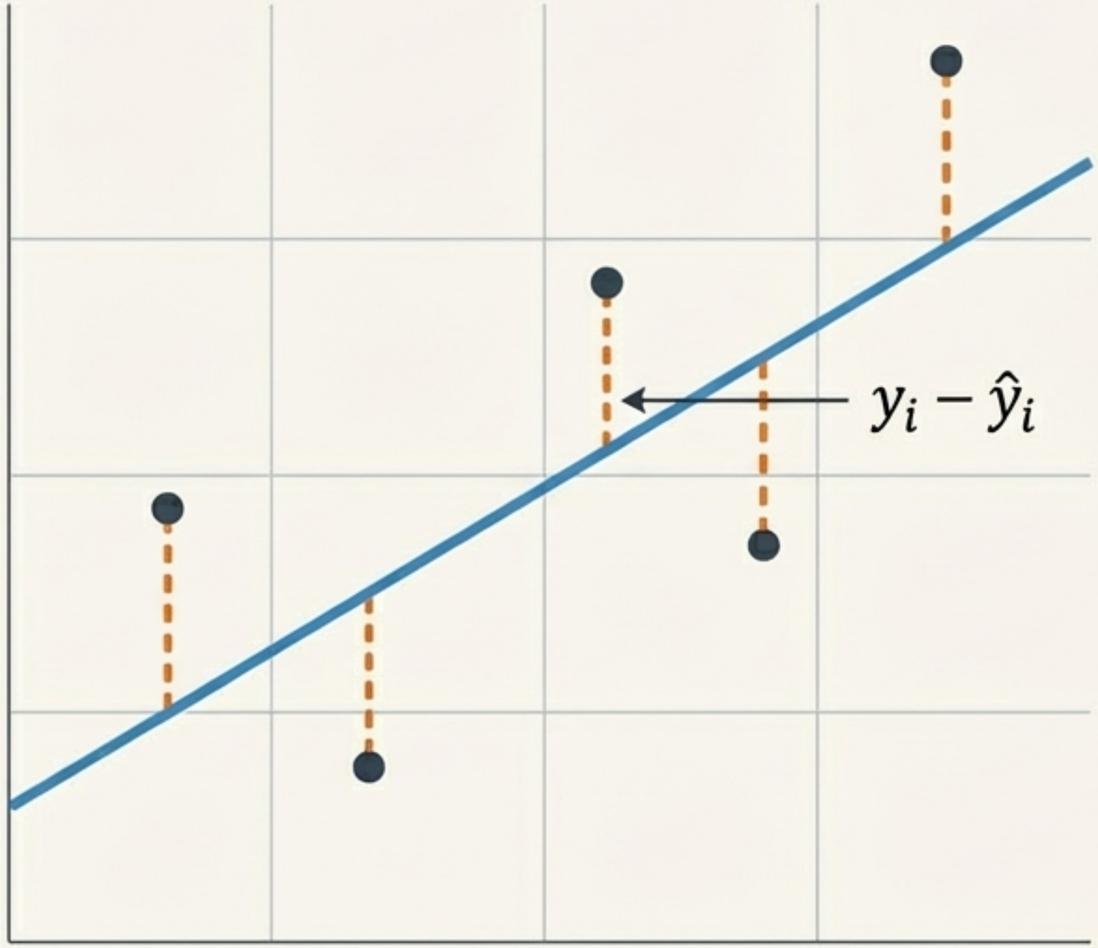
Linear regression assumes a linear relationship between the input and output. Our goal is to use the independent variable to predict the dependent variable.

What makes a ‘good’ line? The quest for the best fit.



Many lines can be drawn through the data, but which one is the *best*? The ‘best-fit line’ is the one that most accurately represents the data by minimizing the overall error between the actual data points and the line’s predictions.

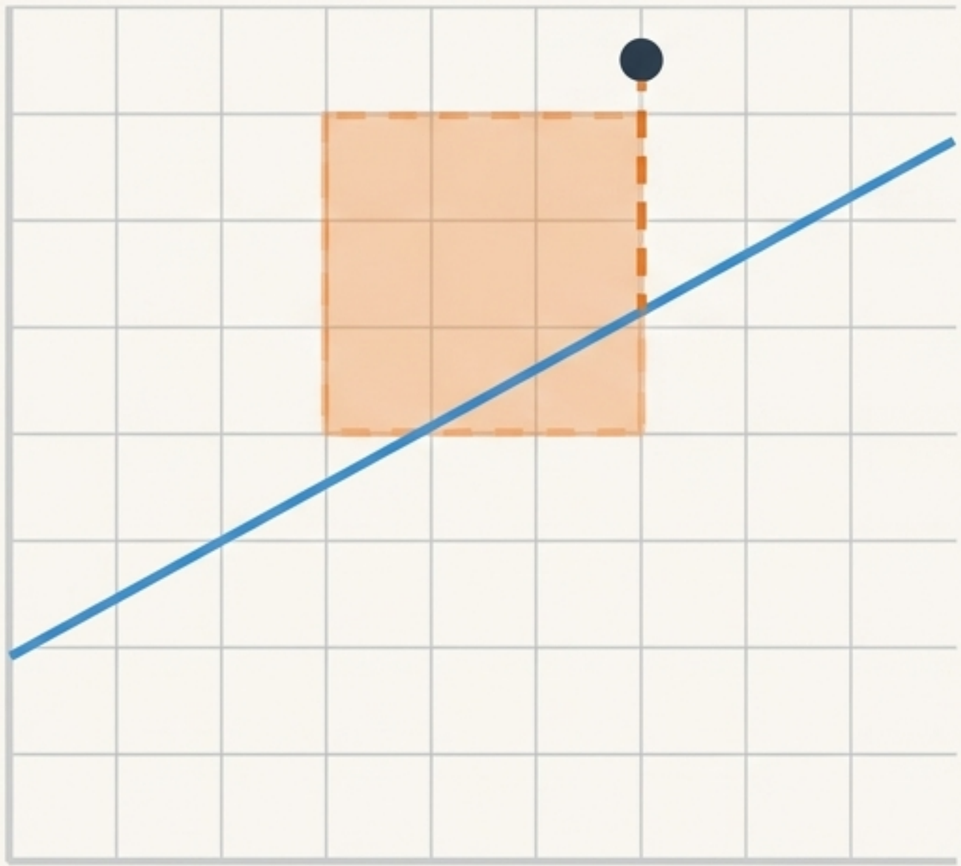
Measuring the error: Introducing the residuals.



The difference between the actual value (the data point) and the predicted value (the point on the line) is called a **residual.**

Residual = y_i (Actual Value) $- \hat{y}_i$ (Predicted Value)

The solution: The Method of Least Squares



The Least Squares method finds the line that minimizes the sum of the *squared* residuals.

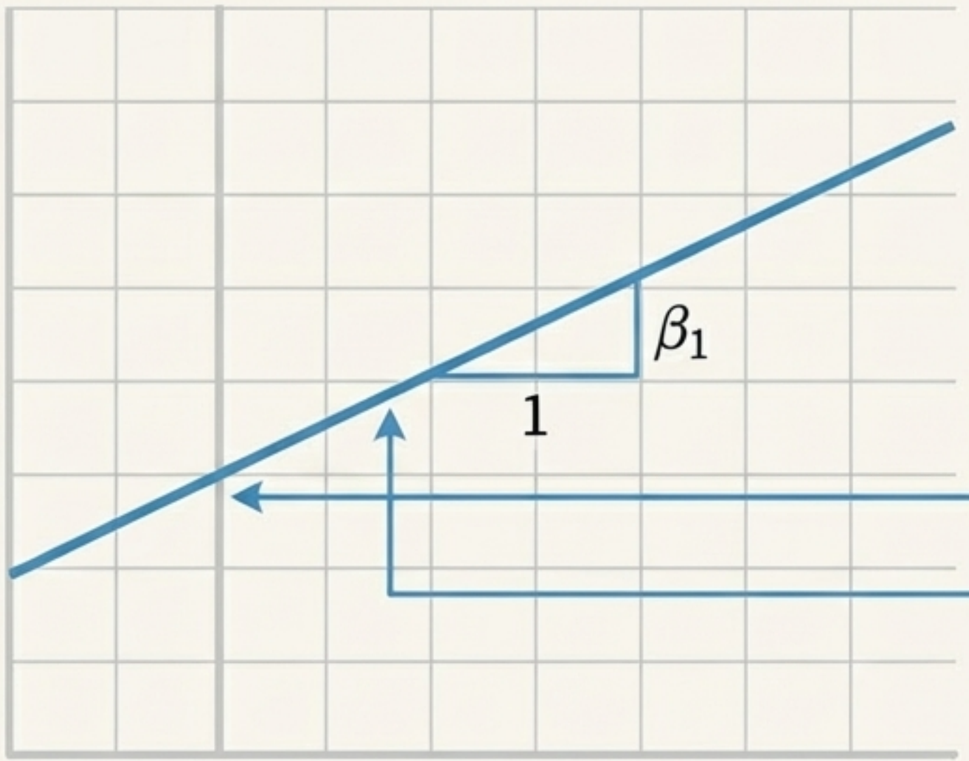
Why square the residuals?

1. Squaring ensures all error values are positive, so they don't cancel each other out.
2. It penalizes larger errors more heavily, forcing the line away from outliers.

$$\text{SSE} = \sum (y_i - \hat{y}_i)^2$$

The language of the line: The Hypothesis Function.

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



ŷ (or h(x)): The predicted value of the dependent variable.

x: The independent variable.

β_0 **(Intercept):** The predicted value of y when x is 0. It's where the line crosses the y-axis.

β_1 **(Slope/Coefficient):** How much ŷ changes for each one-unit change in x. A slope of 5 means y increases by 5 for every 1-unit increase in x.

How the machine learns: Minimizing the Cost Function.

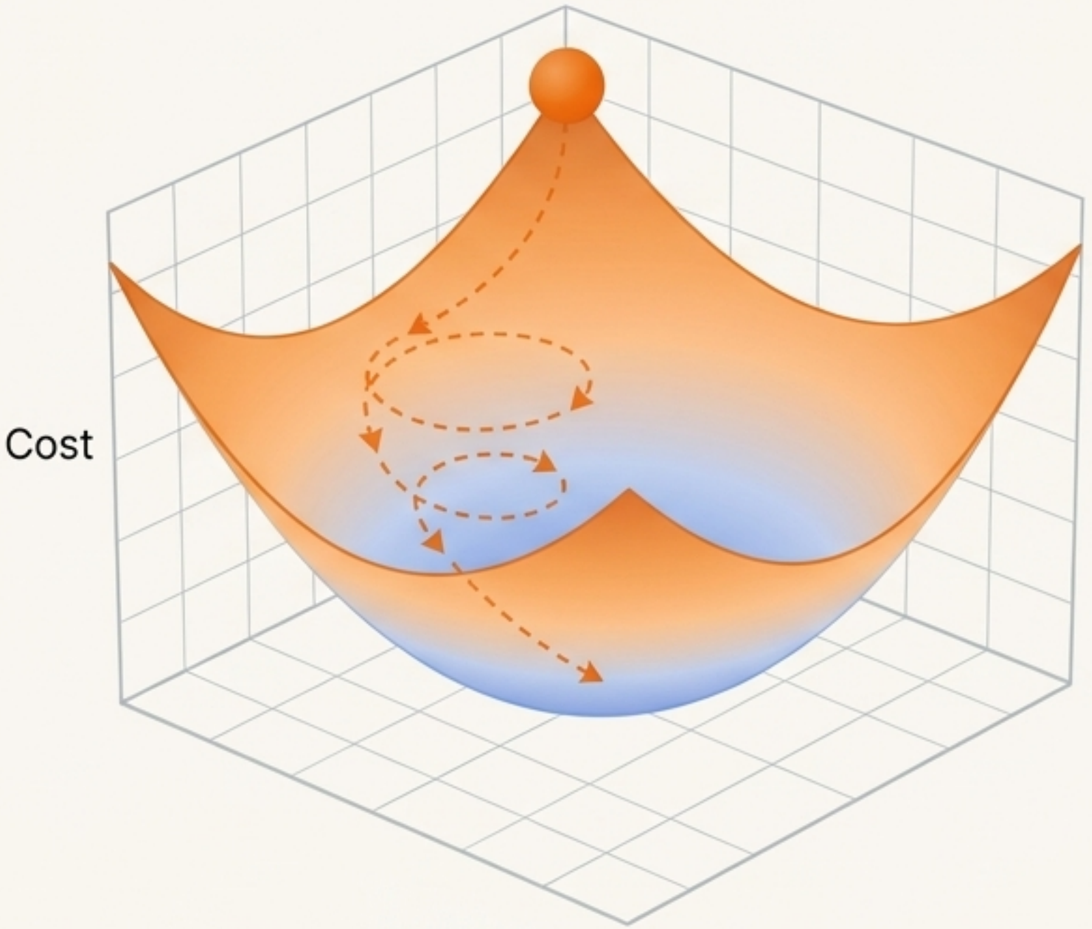
To find the best values for β_0 and β_1, the model needs a way to measure the total error for a given line. This is the **Cost Function**.

The most common cost function is the **Mean Squared Error (MSE)**.

$$\text{Cost } J(\beta_0, \beta_1) = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

The MSE is simply the average of the squared errors across all data points. The goal is to find the line (the β_0 and β_1 values) that makes this this average error as small as possible.

Finding the bottom: The Gradient Descent algorithm.



1. **Start:** Begin with random values for the slope (β_1) and intercept (β_0).
2. **Calculate:** Compute the error (the gradient, or slope, of the cost function).
3. **Update:** Adjust the parameters in the direction that reduces the error the most (i.e., take a step downhill).
4. **Repeat:** Continue this process until the error is minimized (you've reached the bottom of the valley).

Inter: How well does our model explain the git with the data? R-Squared.

Our model has found the best-fit line, but how good is it? How much of the variation in exam scores is actually explained by hours studied?

Coefficient of Determination (R-Squared or R^2)

R-Squared is a statistic that measures the proportion of variance in the dependent variable that is predictable from the independent variable(s).

It is always between 0 and 1.

In general, the higher the R-squared, the better the model fits the data.

A deeper look at model evaluation metrics

Mean Squared Error (MSE)

$$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

The average of the squared differences between actual and predicted values. Sensitive to outliers due to squaring.

Root Mean Squared Error (RMSE)

$$\sqrt{MSE}$$

The square root of MSE. It's in the same units as the target variable (e.g., exam points), making it more interpretable.

Mean Absolute Error (MAE)

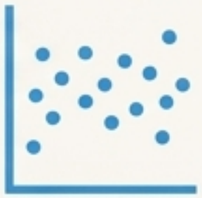
$$\frac{1}{n}\sum|y_i - \hat{y}_i|$$

The average of the absolute differences. It is less sensitive to outliers than MSE.

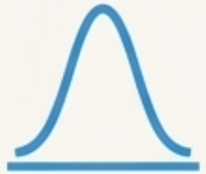
The rules of the game: Key assumptions for a reliable model

For the model's results to be valid and reliable, the data must satisfy several key assumptions.

Linearity: The relationship between X and Y is linear.



Homoscedasticity (Constant Variance): The errors have a consistent spread across all values of X.



Normality of Errors: The errors follow a normal (bell-shaped) distribution.

Independence of Errors: The errors are not correlated with each other.

Beyond a single factor: Multiple Linear Regression.

What if an outcome depends on more than one factor? We can extend our model to include multiple independent variables.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$$

Real-World Use Cases

Real Estate

Predict property prices using size, location, and number of rooms.

Finance

Forecast stock prices using interest rates, inflation, and market indices.

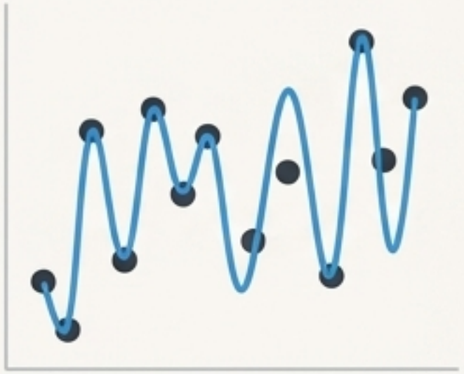
Agriculture

Estimate crop yield from rainfall, temperature, and soil quality.

Preventing overfitting with Regularization.

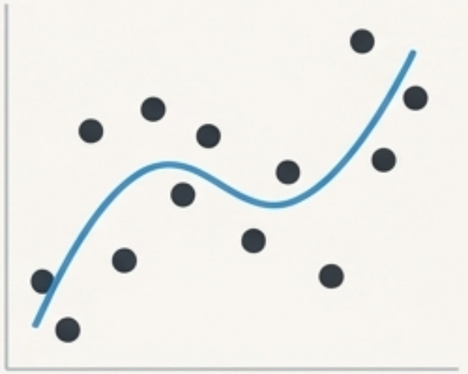
The Problem: Overfitting

Sometimes, a model learns the training data *too* well, including its noise. This hurts its ability to predict new, unseen data.



The Solution: Regularization

Regularization techniques add a penalty to the cost function for large coefficient (β) values. This discourages the model from becoming overly complex.



Lasso Regression (L1): Can shrink some coefficients to exactly zero, effectively performing feature selection.

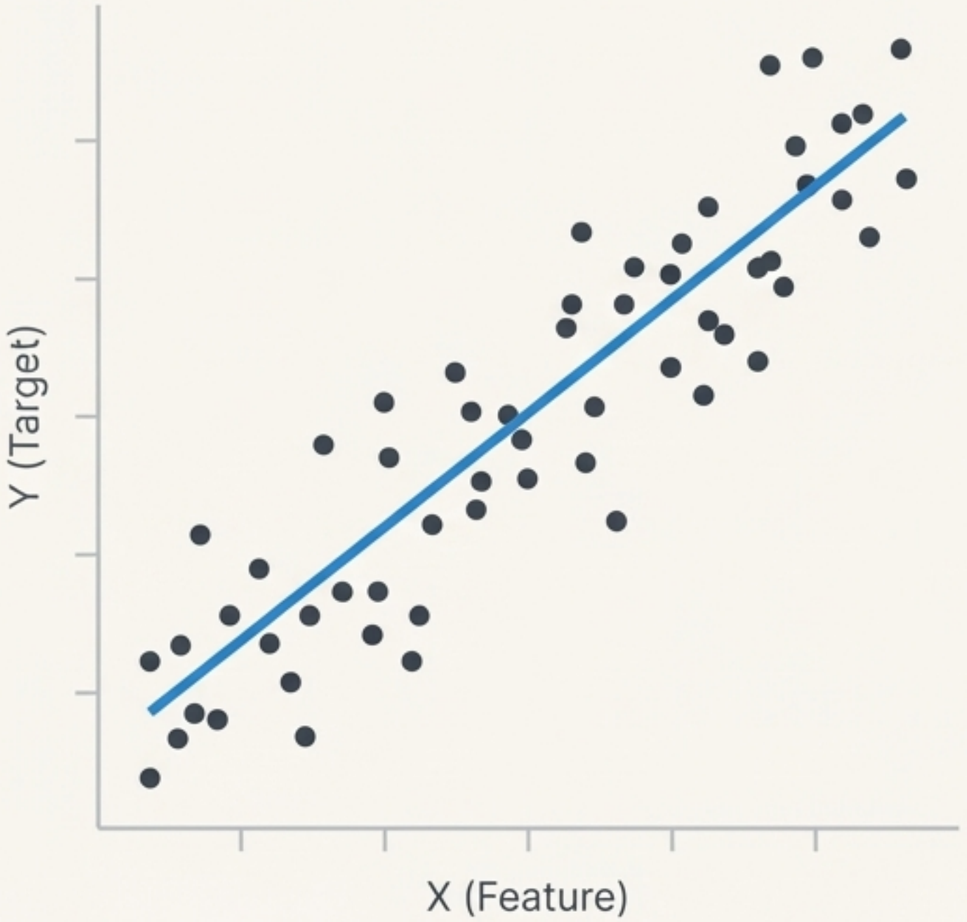
Ridge Regression (L2): Shrinks coefficients towards zero but doesn't eliminate them. Useful when predictors are highly correlated.

Linear Regression in practice: A Python example.

```python
# 1. Import the library
from sklearn.linear_model import LinearRegression
import numpy as np

# 2. Create data
X = np.array([[1], [2], [3], [4], [5]]) # Hours Studied
Y = np.array([50, 60, 70, 80, 90])      # Exam Score

# 3. Create and train the model
model = LinearRegression()
model.fit(X, Y)

# 4. Get the results
slope = model.coef_
intercept = model.intercept_
print(f"Slope (β₁): {slope[0]}")
print(f"Intercept (β₀): {intercept}")
```

> Libraries like Scikit-learn handle the complex math of Gradient Descent, allowing us to build models efficiently.

Simple, powerful, and fundamental.



Why It's Important

- **Simplicity & Interpretability:** Easy to understand and explain.
- **Efficiency:** Computationally fast, even on large datasets.
- **Predictive Power:** A strong baseline for prediction tasks.
- **Foundation:** Core concepts are the basis for more advanced algorithms.

Key Limitations

- **Assumes Linearity:** Performs poorly if the underlying relationship isn't linear.
- **Sensitivity to Outliers:** Extreme values can significantly skew the best-fit line.
- **Multicollinearity Issues:** Can be unstable if input variables are highly correlated.