



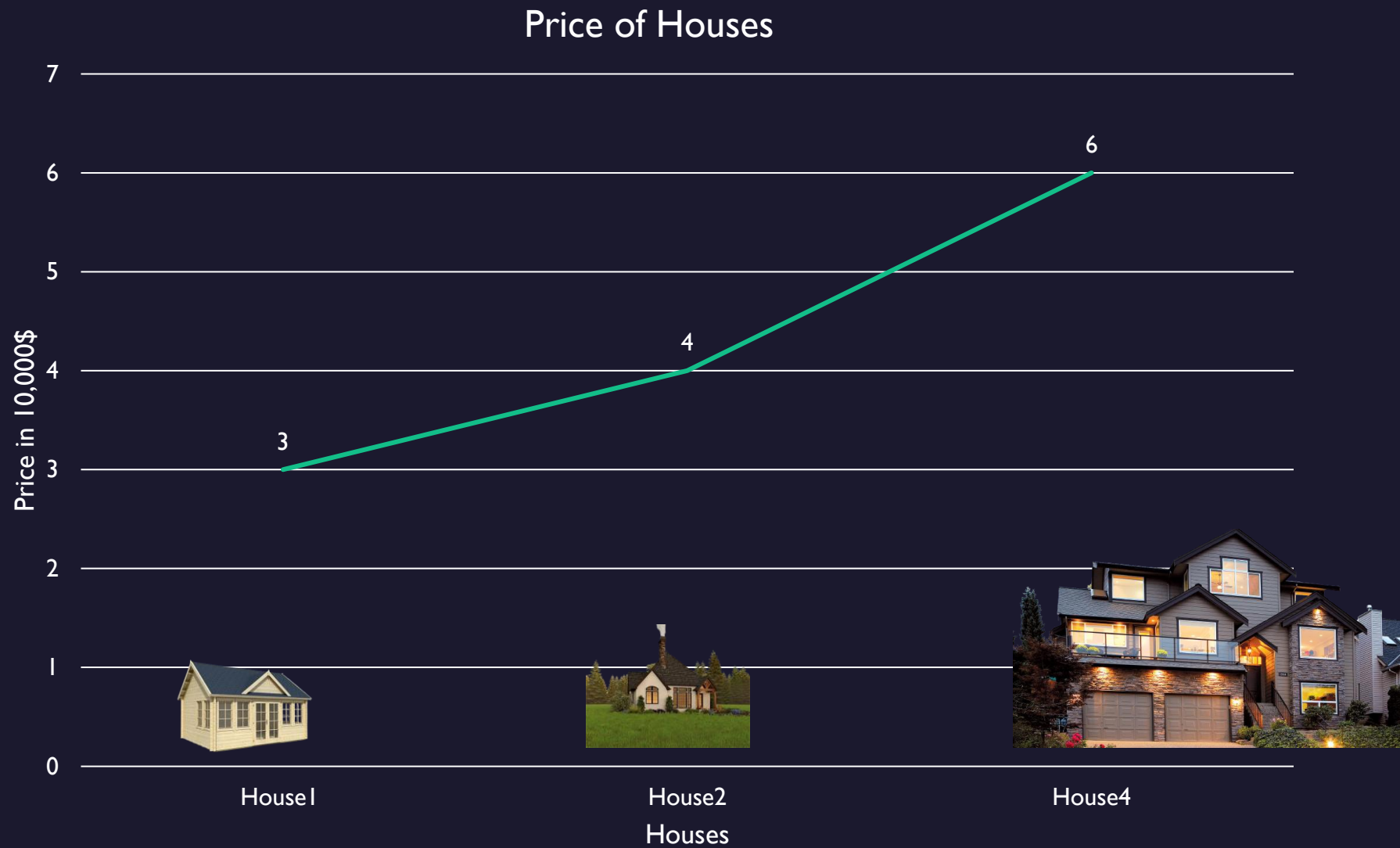
Linear Regression

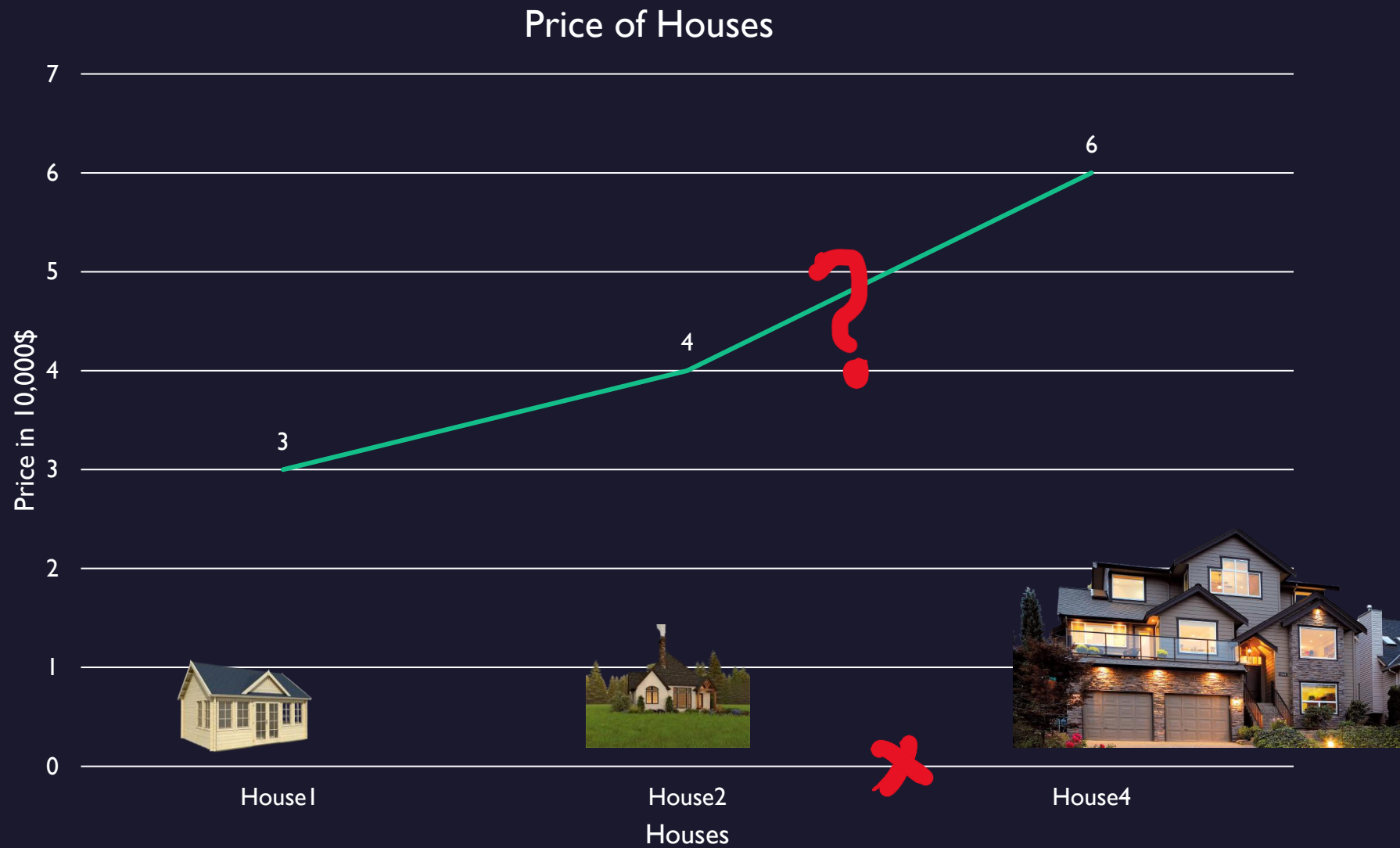
Hossam Ahmed

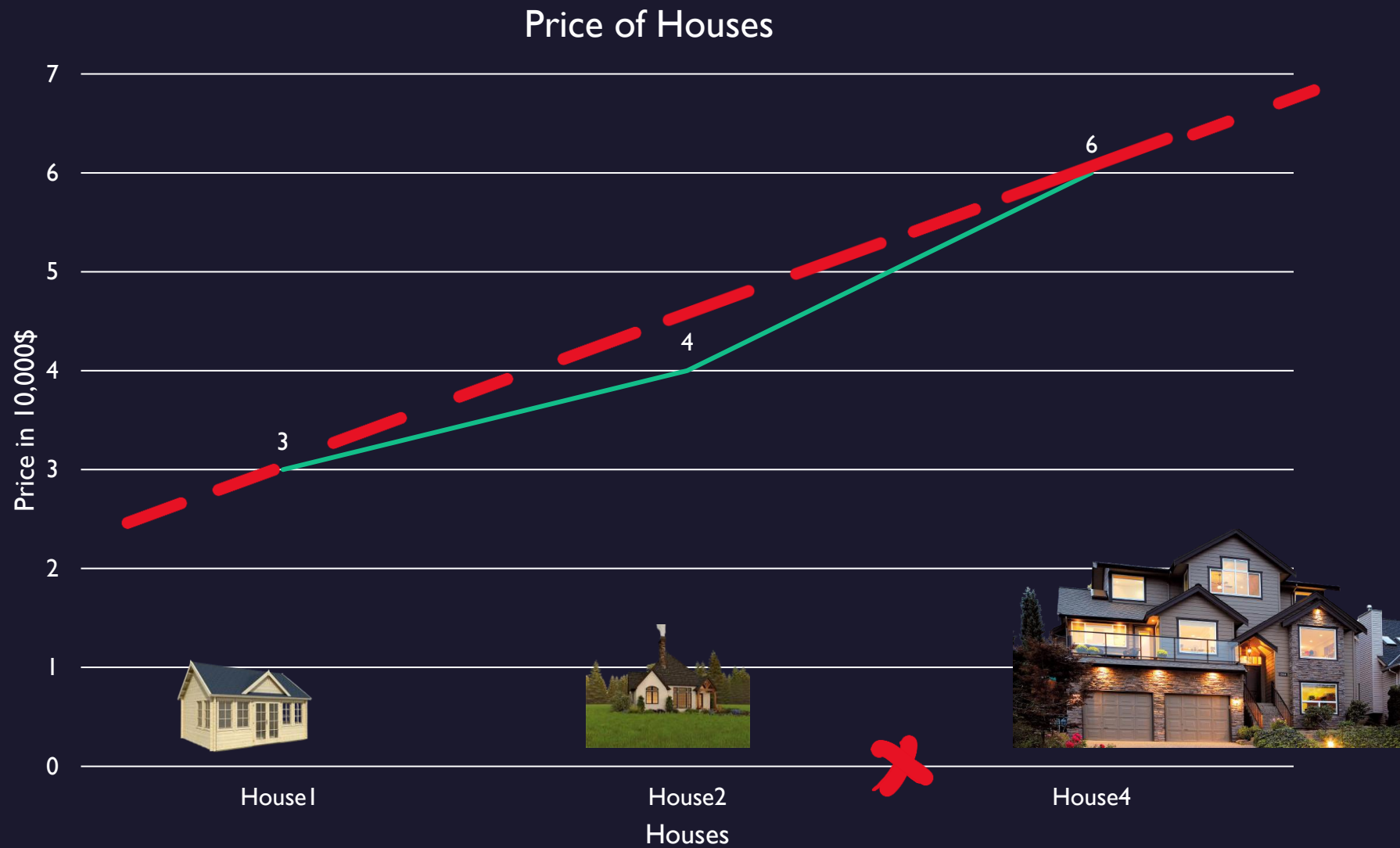
Ziad Waleed

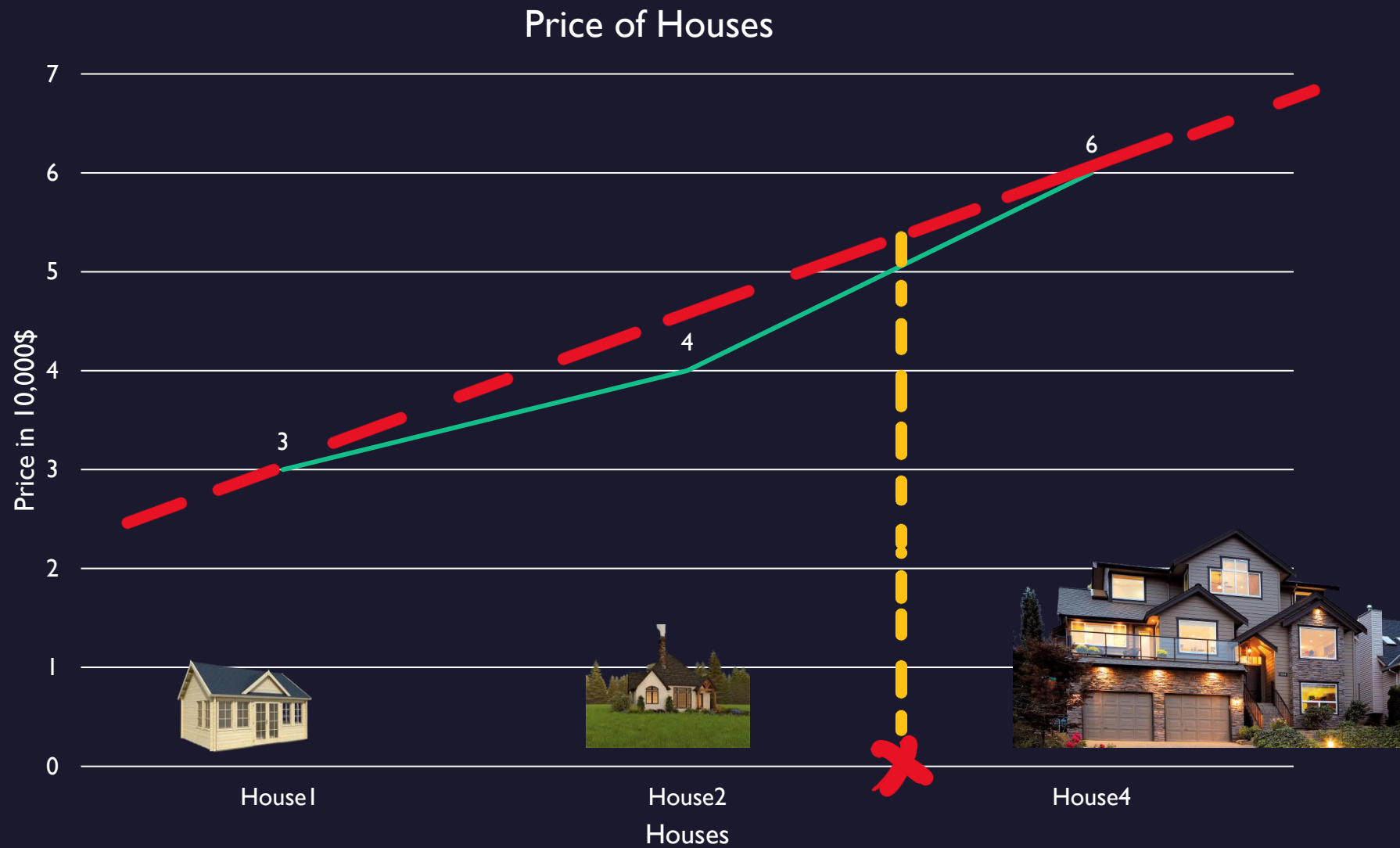
Mario Mamdouh

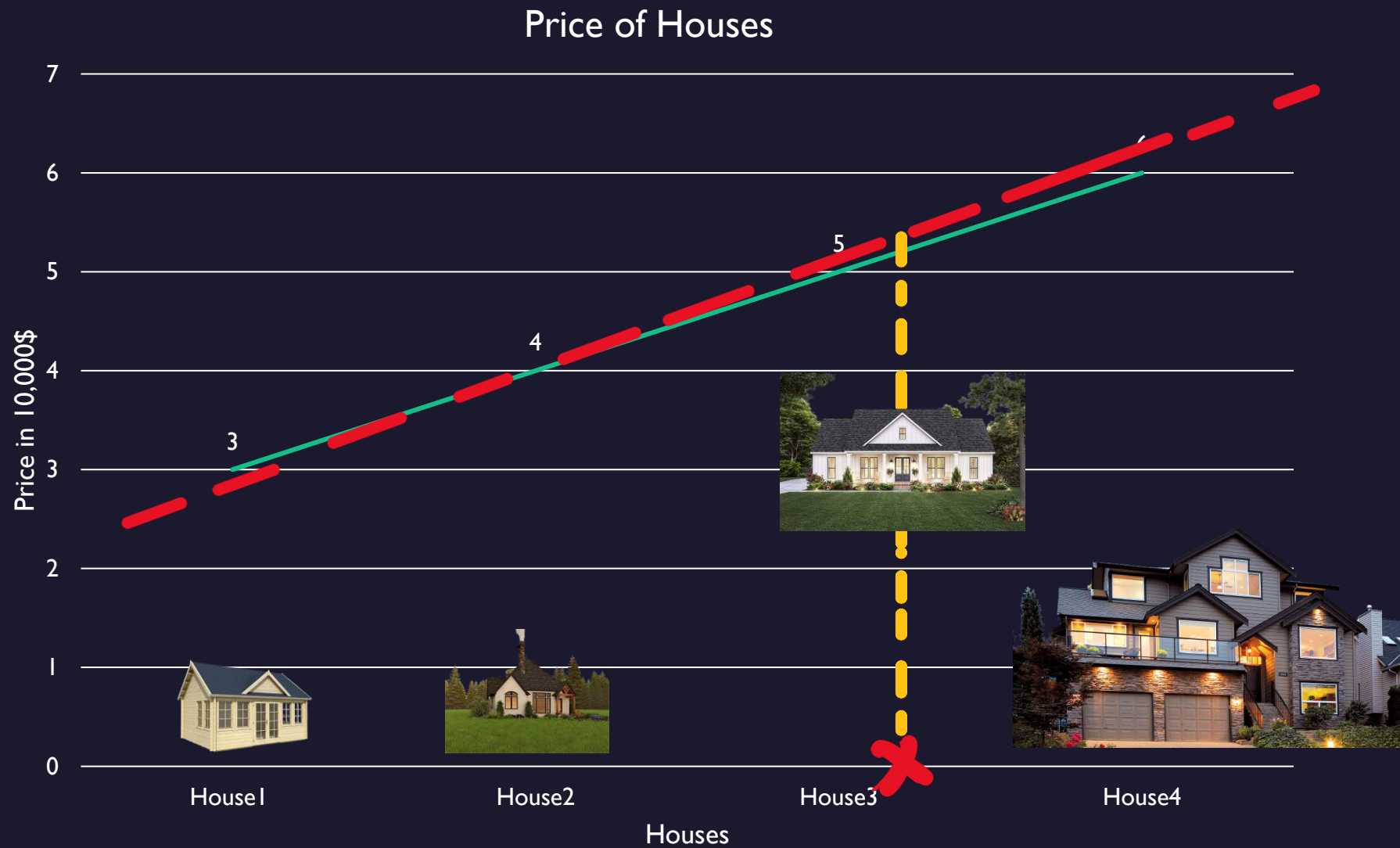
House pricing problem

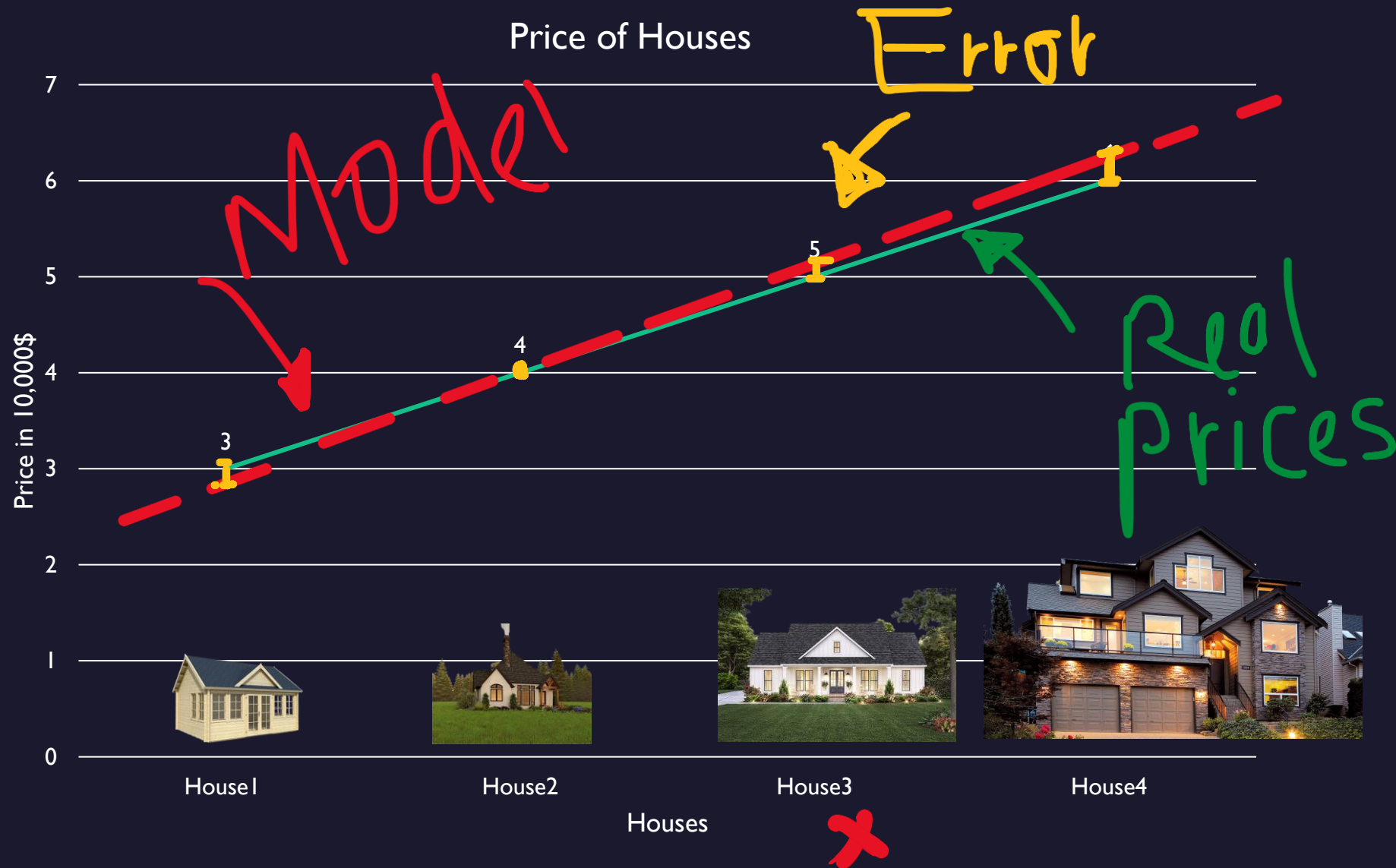


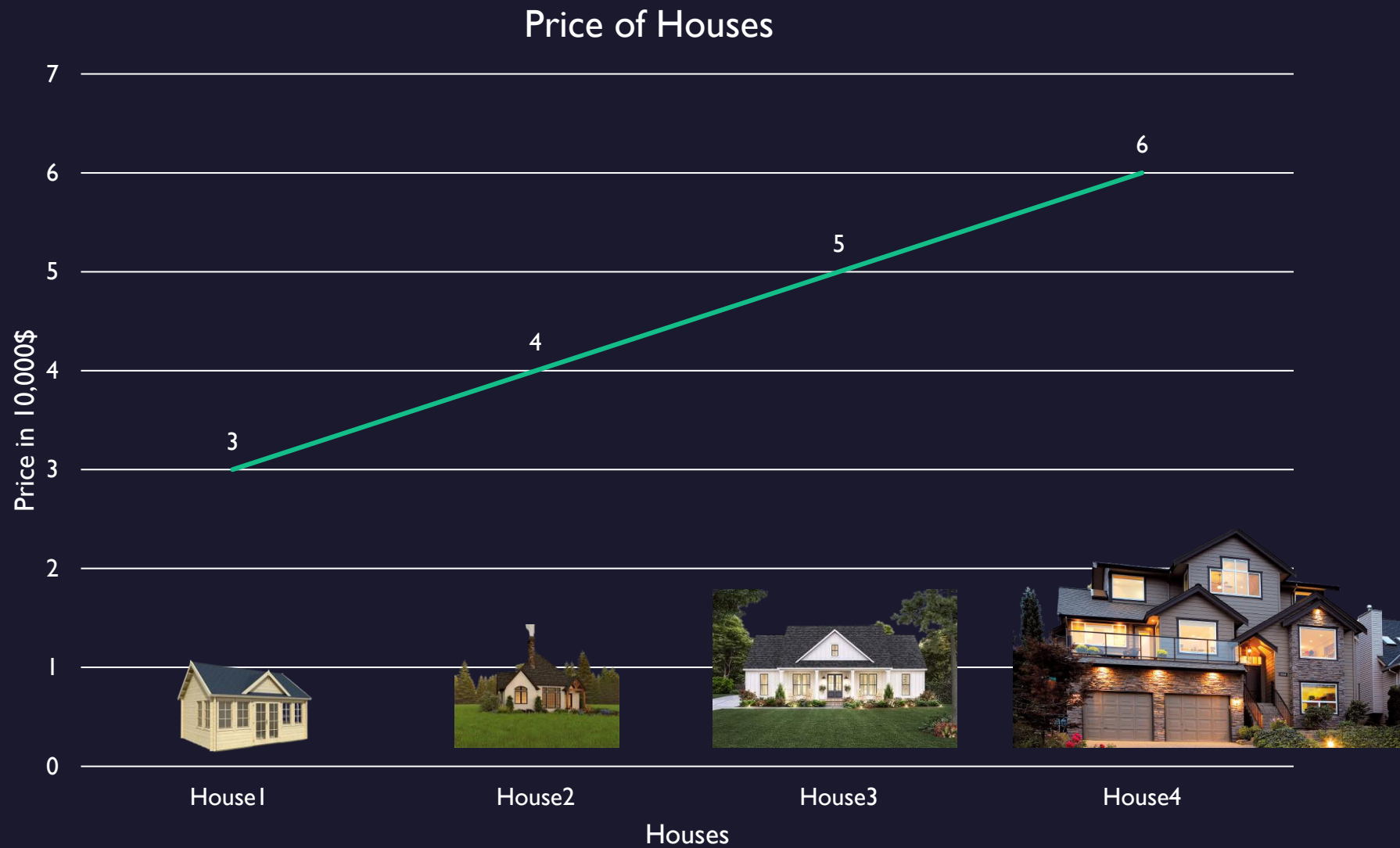




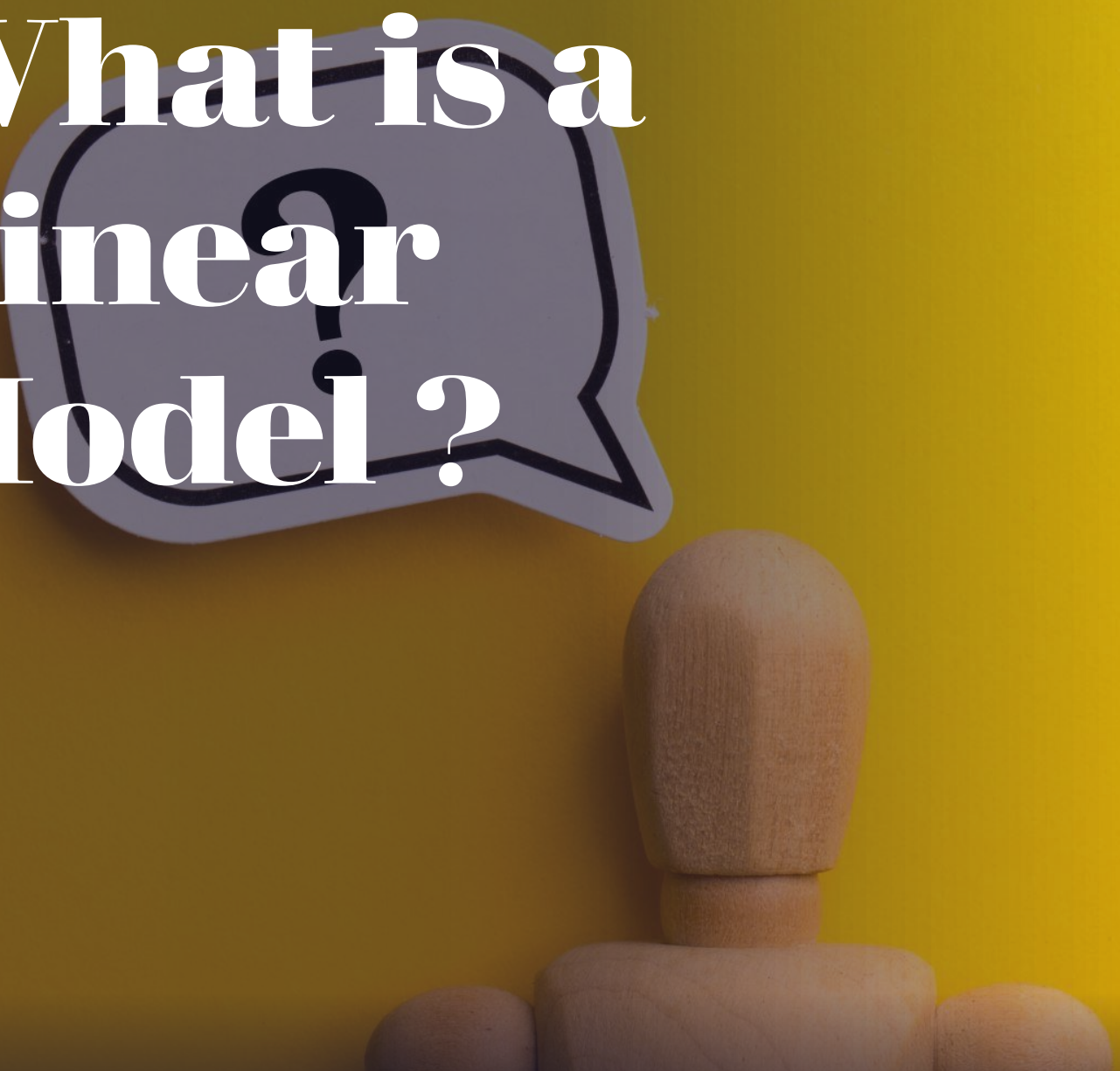








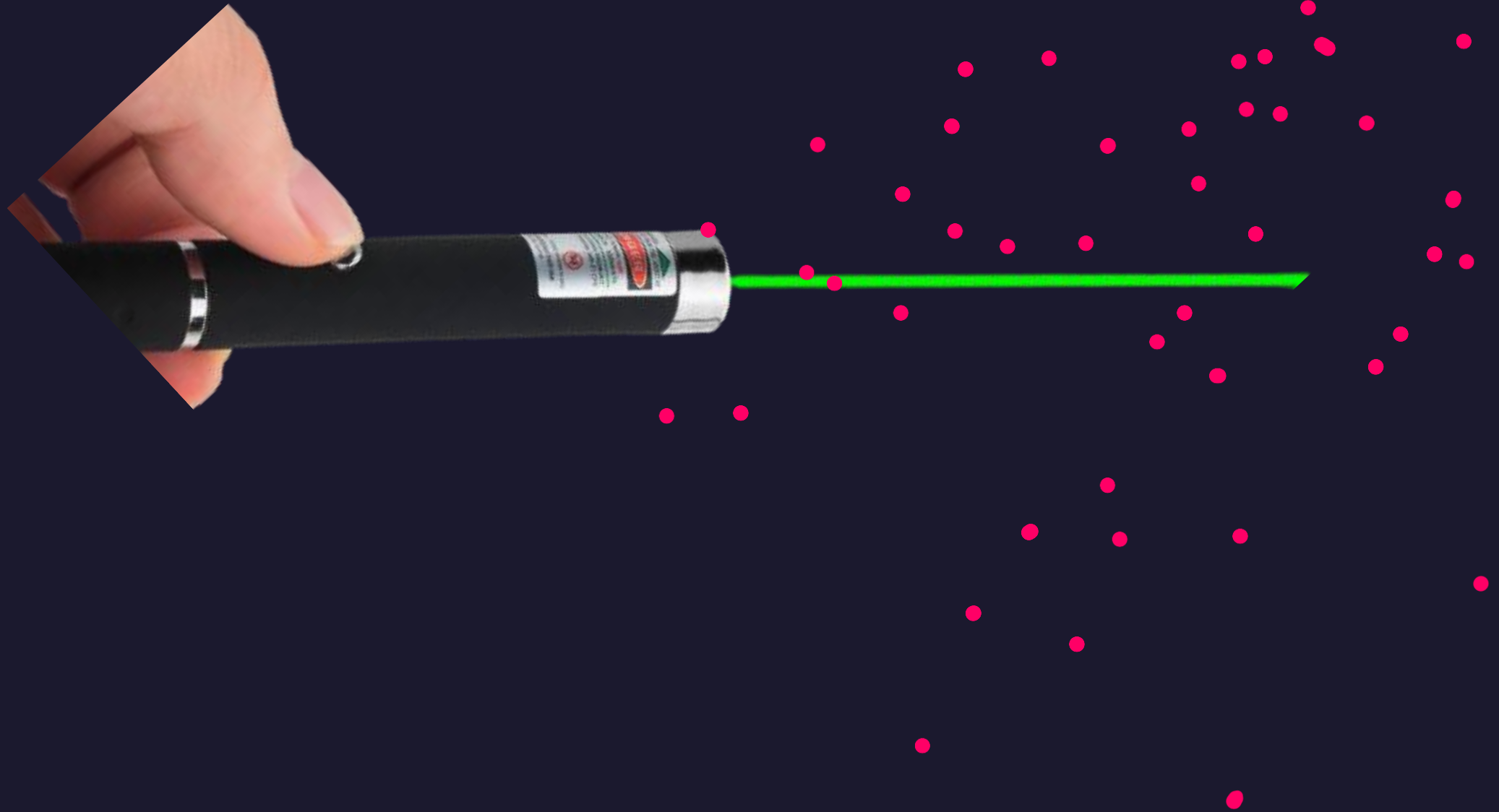
What is a Linear Model?

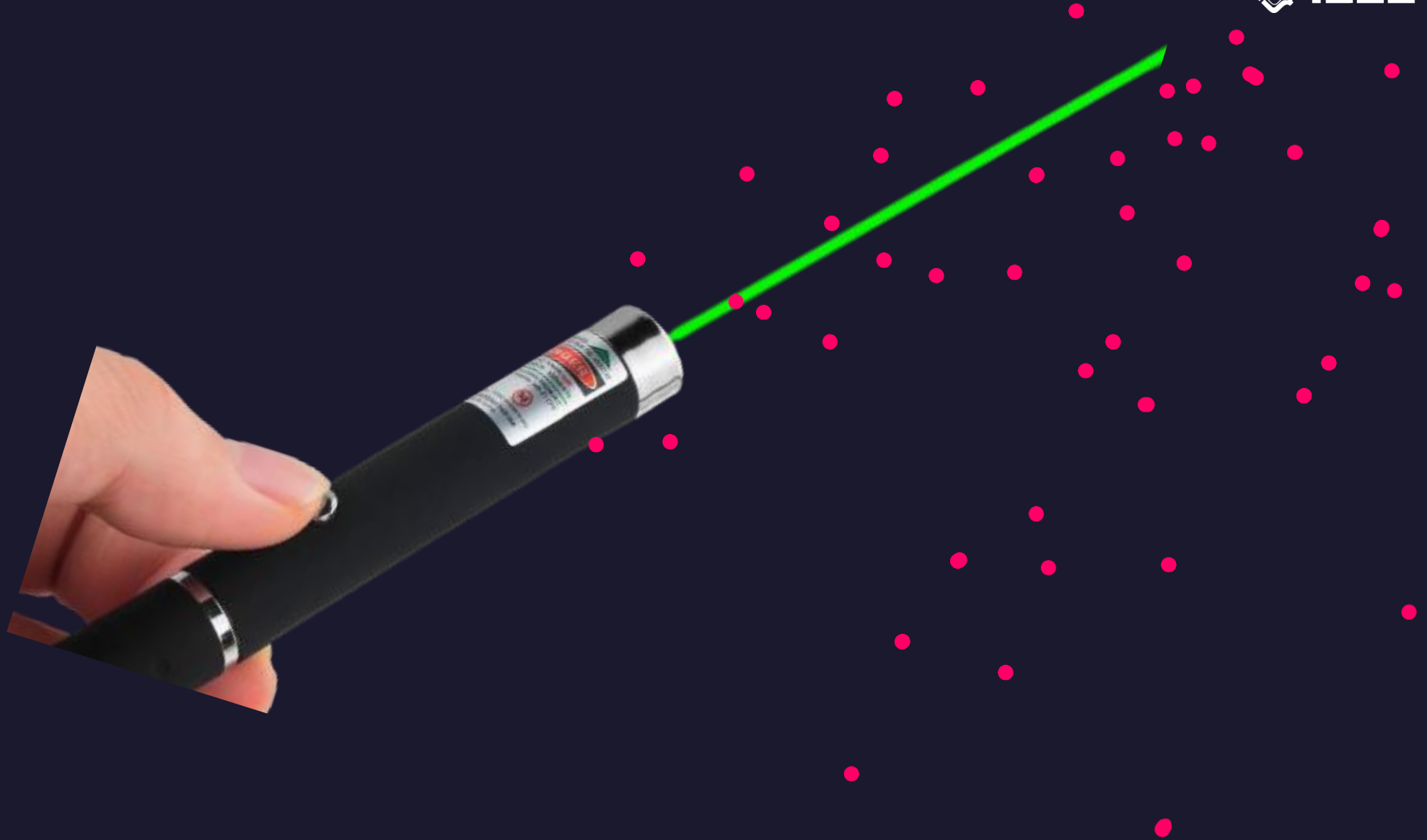
A wooden mannequin figure is positioned in the lower center of the frame. To its left, a purple question mark icon is visible, partially obscured by the text. The background is a smooth gradient from dark yellow to light yellow.



What is the Line ?








So, a line can move **UP/DOWN** , **Right/Left** or a **mix** of the two

Mathematically a line is

$$f(x) = mx + b$$

m : slope represent how much you want to turn the line (circular move )

b : the intercept of Y (  control)

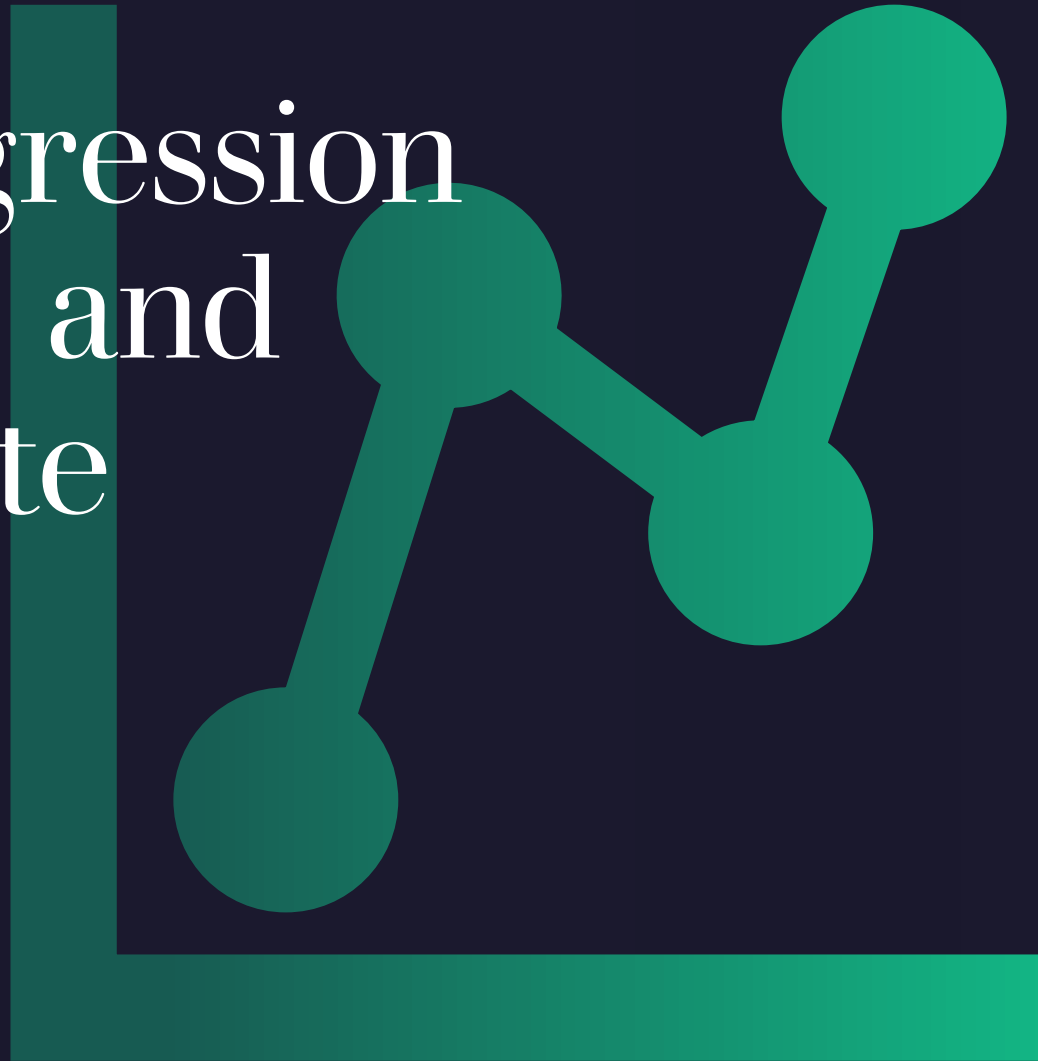
We have features of some object like the house
for example : {size, location, number of rooms,
price, ...}

The model is simply a function represent the
relation between the target feature and the rest of
the features

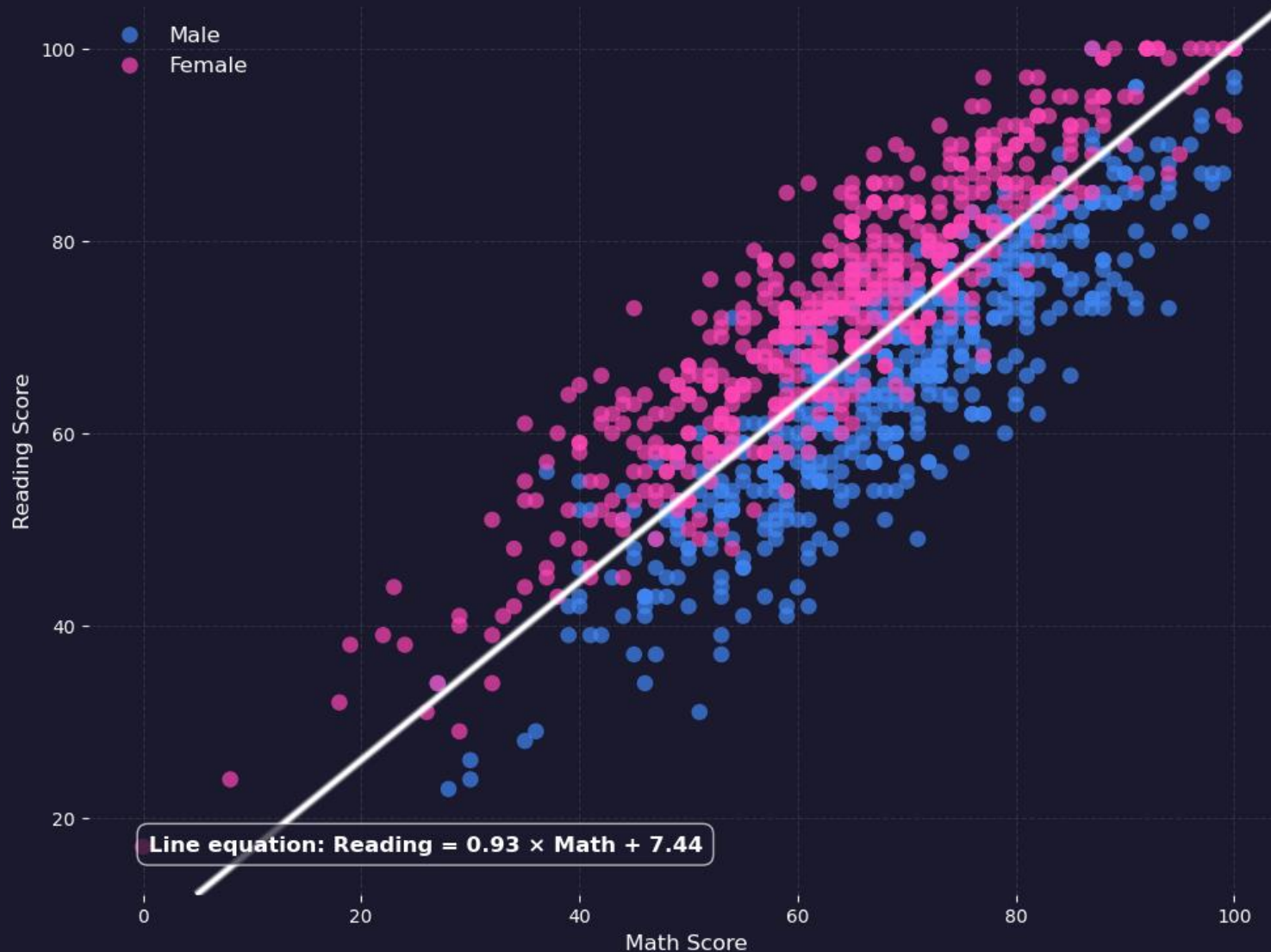
$$F(\text{Features}) = \text{Target}$$

$$F(\text{size, num of rooms, location, ...}) = \text{price}$$

Linear regression univariate and multivariate

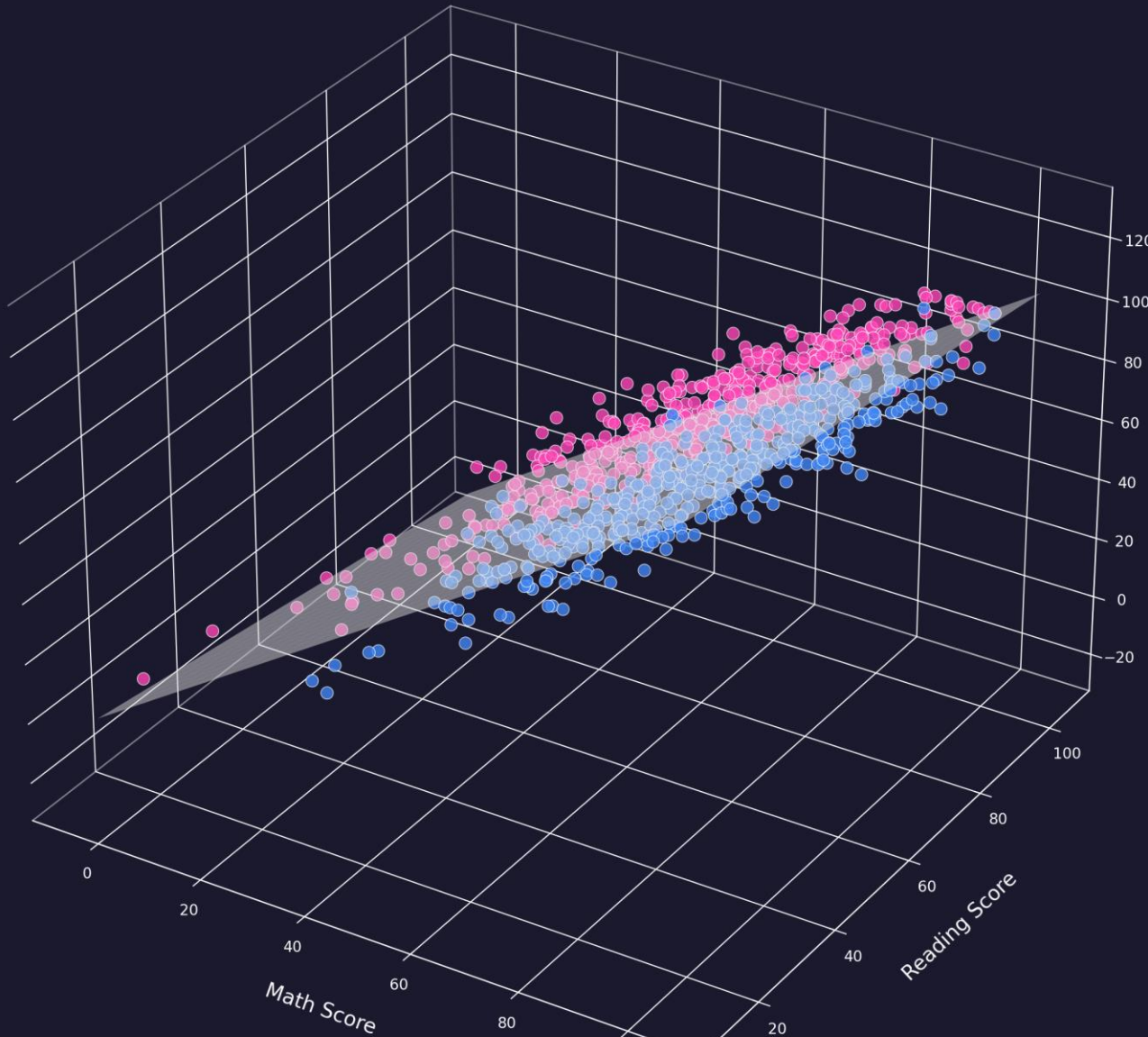


Student Performance with Gender Boundary



- A line that separate between Male/Female student based on their test performance on two subjects “Reading” and “Math” each out of 100.
- This is called univariant as we have only one independent variable (or feature) is used to predict the dependent variable (target)
- $\hat{y} = w_0 + w_1x$
 - x is the independent variable (Math)

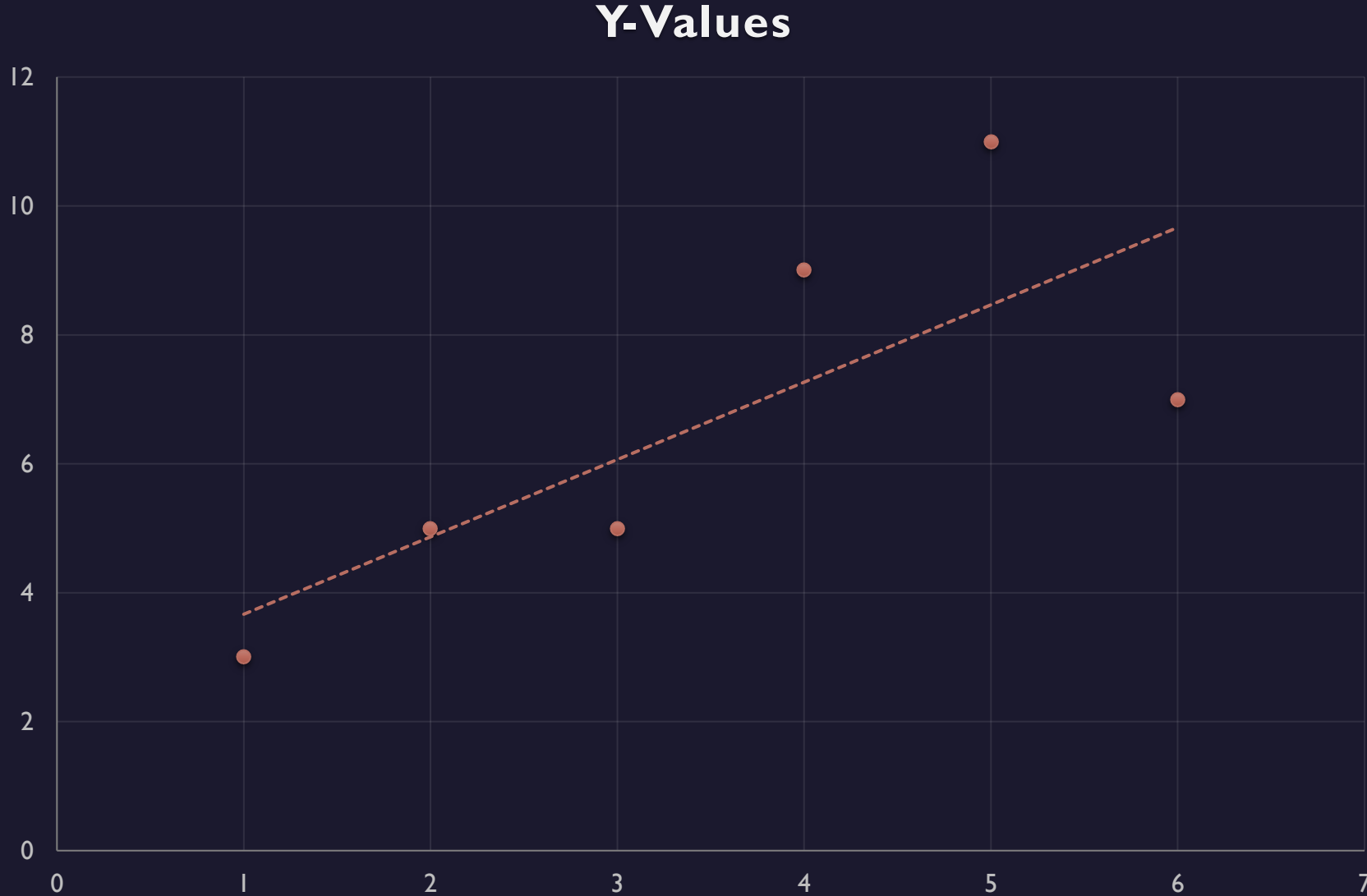
● Male
● Female



Line equation: Writing = 1.15 × Math + -0.23 × Reading + 7.69

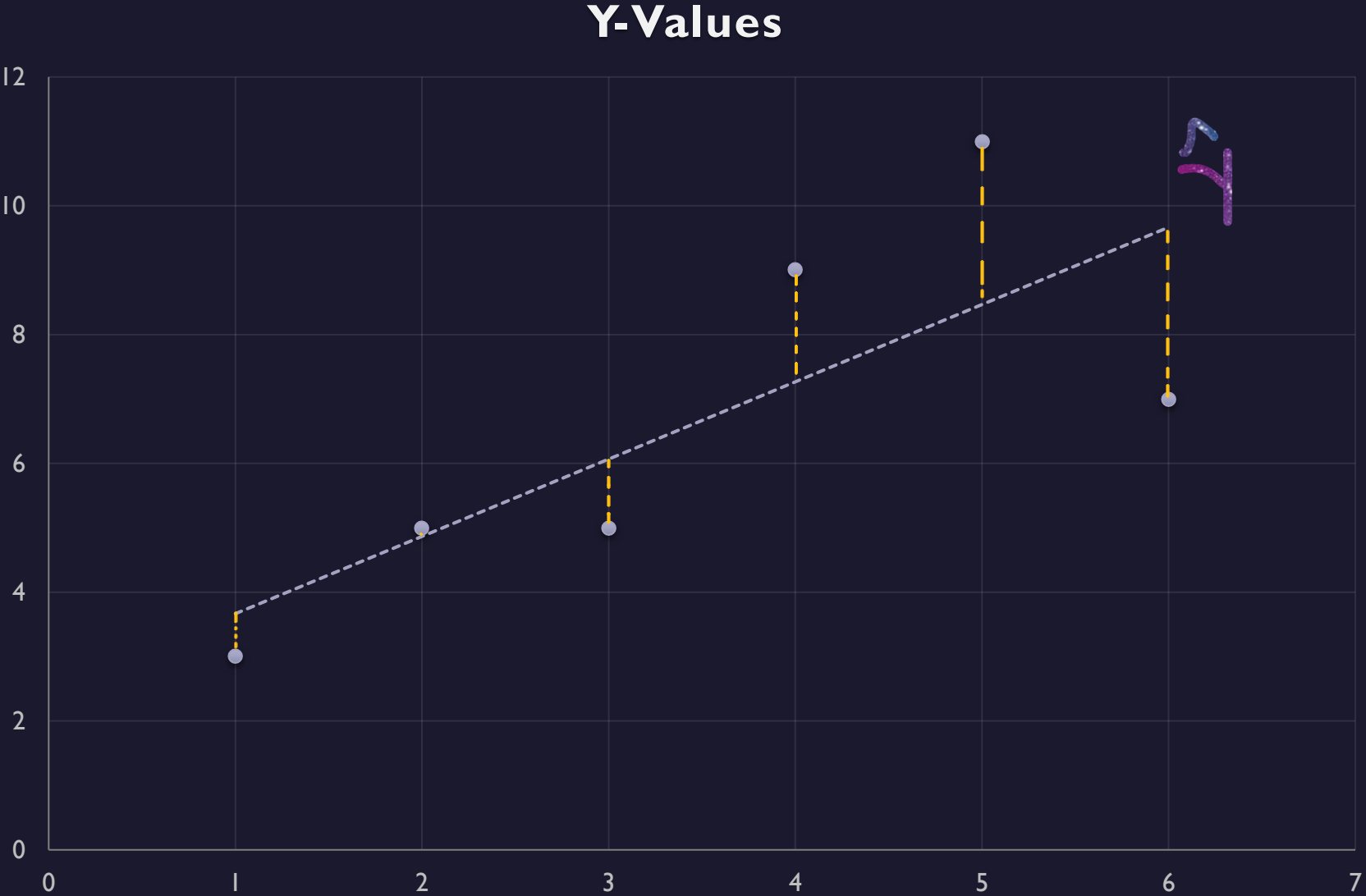
- A line that separate between Male/Female student based on their test performance on two subjects “Reading”, “Math” and “Writing” each out of 100.
- A line in 3D space become a 2D plane, and in 2D is just a line, in 1D would be just a point, (#D-1).
- This is multivariate involves multiple independent variables (or features)
 - $\hat{y} = w_0 + w_1x_1 + w_2x_2$

Regression Models Evaluation

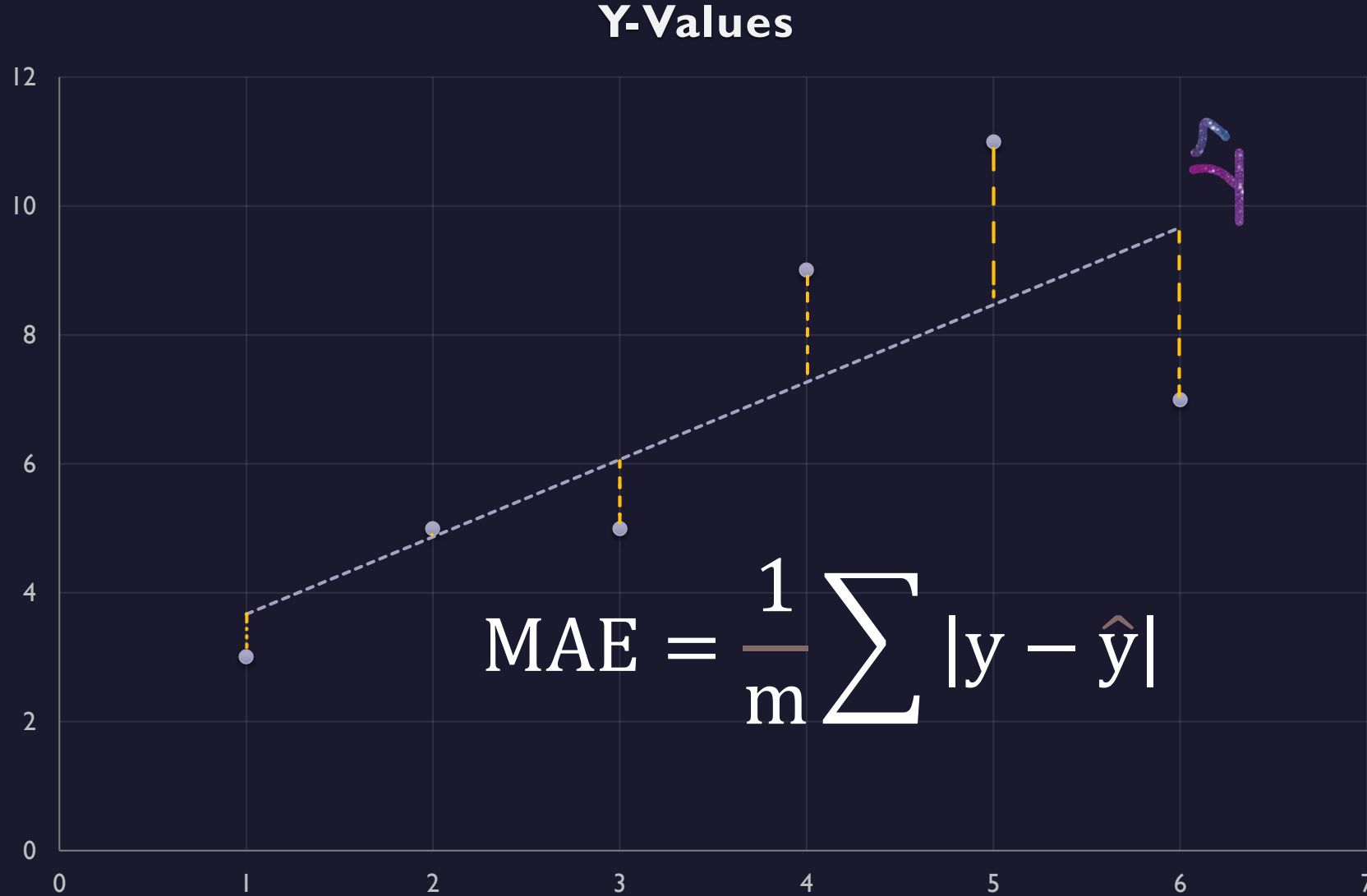


IEEE ML S25' training sessions

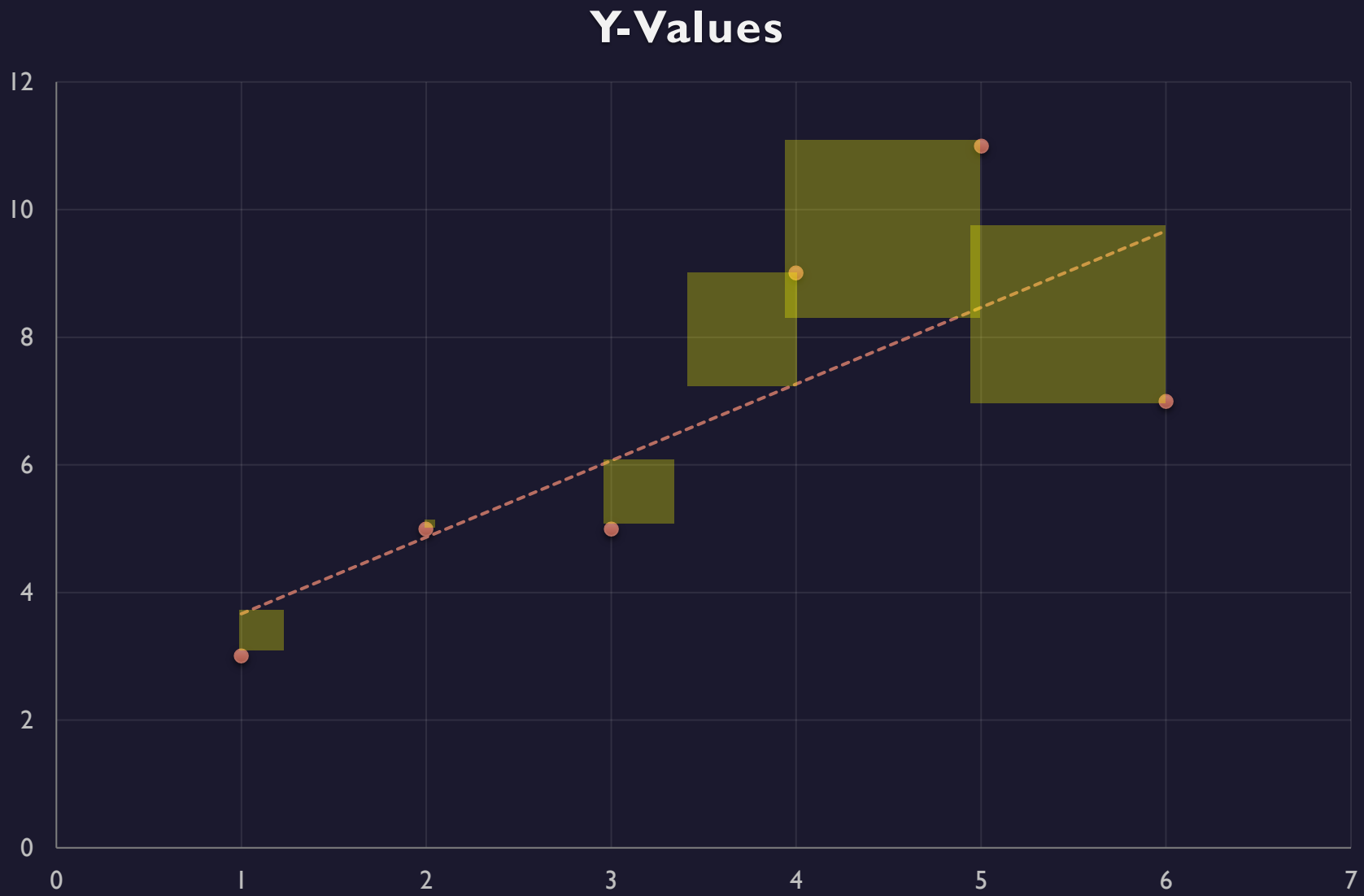
Regression Models Evaluation



Regression Models Evaluation

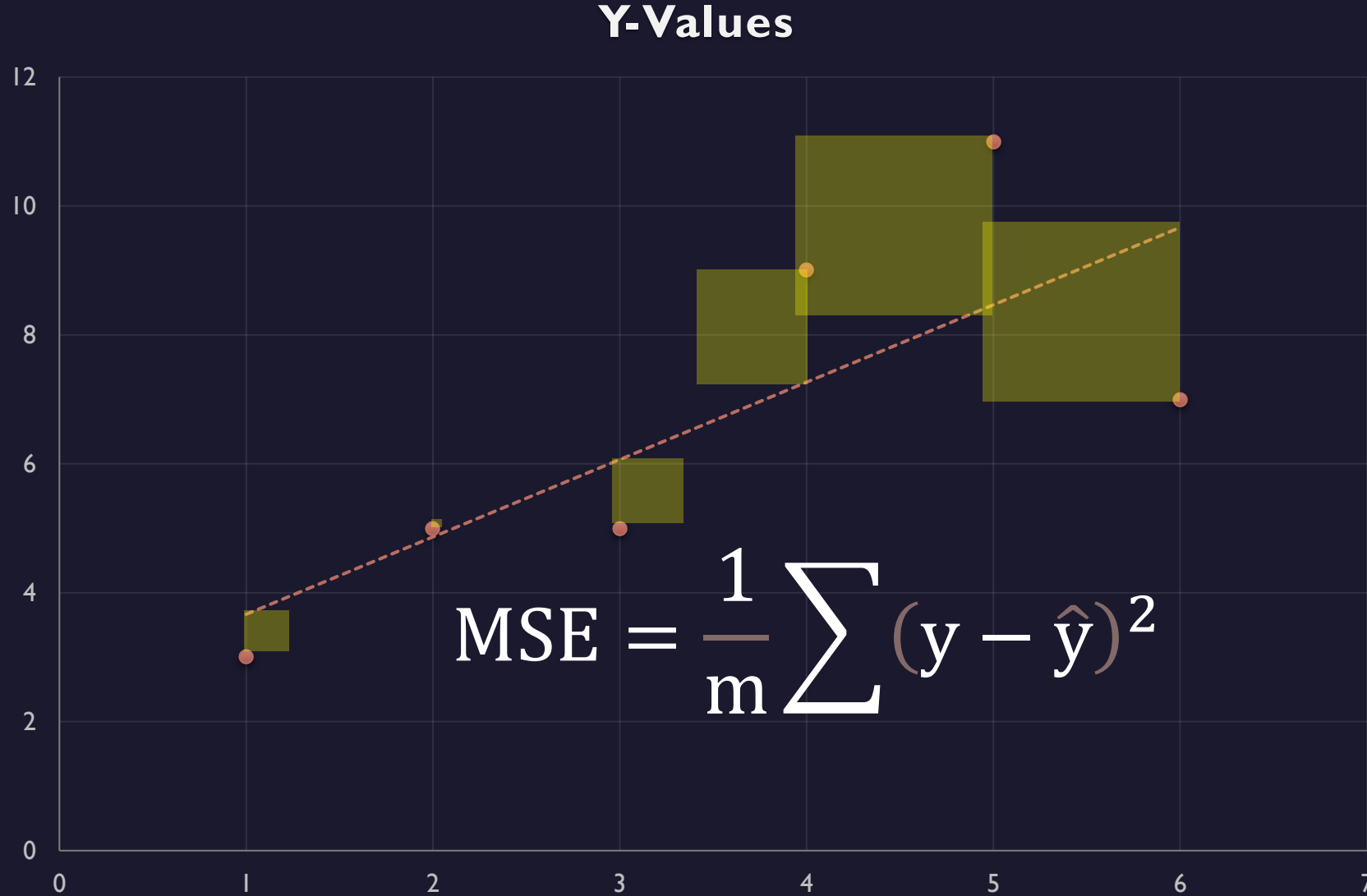


Regression Models Evaluation



IEEE ML S25' training sessions

Regression Models Evaluation



Regression Model Evaluation

- **Residual** refers to the difference between the **observed value** (the actual data point) and the **predicted value** (the value predicted by the regression model).
 - $e_i = (y_i - \hat{y}_i)$ where i is the index for the current data point x_i
- **Residual Sum of Squares (RSS)** measures the sum of squared residuals (errors).
 - $RSS = \sum_{i=1}^n e^2 = \sum_{i=1}^n (y - \hat{y})^2$
- **Mean Squared Error (MSE)** measures the average of the squared differences between the predicted and actual values. It penalizes larger errors more than MAE because of the squaring term.
 - $MSE = \frac{1}{n} RSS = \frac{1}{n} \sum_{i=1}^n e^2 = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$

Regression Model Evaluation

- **Root MSE (RMSE)** RMSE is the square root of MSE and represents the error in the same units as the target variable, making it easier to interpret.
 - $RMSE = \sqrt{MSE}$
 - For example, if you're predicting prices in dollars, RMSE will be in dollars, making it more understandable.
- **Coefficient of Determination (R-squared)** measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It gives an indication of how well the regression model explains the variability in the data.
 - **Total sum of squares** $TSS = \sum (y_i - \bar{y})^2$ proportional to the variance
 - **Explained sum of squares** $ESS = \sum (\hat{y}_i - \bar{y})^2$ measures the variation captured by the model
 - $TSS = ESS + RSS$
 - $R^2 = \frac{\text{Variability captured by the model}}{\text{Total variability in the data}} = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$
 - **R-squared ranges between 0 (worst) and 1 (best)**

Regression Model Evaluation

- **MAE, MSE, RMSE, and RSS:** All these metrics are non-negative, with **lower values indicating better model performance**.
 - A value of **0 indicates a perfect model** with no error, and larger values represent worse performance
- MAE is less influenced by extreme data points, and it's in the same unit as the target variable making it more interpretable, but it's not differentiable, doesn't penalize large errors (MAE treats all errors equally), less sensitive to small variations.
- R-squared can be misleading because it often looks good on training data (can be tricked by **overfitting** (*memorizing all the training points*) due to its dependence on variance)
 - Not good with other types of regression model except **linear** regression models.
 - Not a direct measure of prediction error, so in sometimes you have good score but bad model.

GD for minimizing regression error

- We have a linear model that generate error when trying to predict the target
 - Model $\hat{y} = w_0 + w_1x$
 - Error is a deviation from the true value y (e.g., MSE, MAE)
 - $MSE = \frac{1}{n} \sum_{i=1}^n e^2 = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2 = \frac{1}{n} \sum_{i=1}^n (y - (w_0 + w_1x))^2$
- We want to minimize the error MSE, this error is generated by the linear model, and the model is generated by its weights (w_0, w_1) the bias and the slope.
 - Weights (bias, slope) \rightarrow Linear Model \rightarrow Predictions \rightarrow Error (MSE)
 - So to minimize the error we need to change the origin of this chain...the weights.
- We need to find the optimal bias w_0 and the optimal slope w_1 that has the minimum error, we need to separate each factor (weight) alone to find its optimal, meaning we study it's partial change, how much the error is changed when changing only one of the two parameters.

GD for minimizing regression error

- We need to find the optimal bias w_0 and the optimal slope w_1 that has the minimum error MSE.

- $\frac{\partial \text{Error}}{\partial w_0} = \frac{\partial \text{MSE}}{\partial w_0}$
- $\frac{\partial \text{Error}}{\partial w_1} = \frac{\partial \text{MSE}}{\partial w_1}$



$$w_0^{\text{new}} = w_0^{\text{old}} - \eta \frac{\partial \text{Error}}{\partial w_0}$$

$$w_1^{\text{new}} = w_1^{\text{old}} - \eta \frac{\partial \text{Error}}{\partial w_1}$$

- Since we are utilizing Mean Squared Error (MSE) during the training process to quantify the error, we refer to the MSE function as the ‘ **loss function** ’ when calculated for an individual data point, and as the ‘ **cost function** ’ when aggregating the losses across the entire training dataset
 - Loss function $L(\hat{y}, y)$
 - Cost function $J(w) = \frac{1}{n} \sum \text{Loss}_i$ where w is the model parameters.

GD for minimizing regression error

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{\partial MSE}{\partial w_0} = \frac{\partial}{\partial w_0} \frac{1}{n} \sum_{i=1}^n e^2$$

$$\frac{\partial MSE}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n e * \frac{\partial}{\partial w_0} e$$

$$\frac{\partial MSE}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \frac{\partial}{\partial w_0} (y - \hat{y})$$

$$\frac{\partial MSE}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \frac{\partial}{\partial w_0} (y - (w_0 + w_1 x))$$

GD for minimizing regression error

$$\frac{\partial \text{MSE}}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) * \frac{\partial}{\partial w_0} (\mathbf{y} - (\mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}))$$

$$\frac{\partial \text{MSE}}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) * \left(\frac{\partial}{\partial w_0} \mathbf{y} - \left(\frac{\partial}{\partial w_0} \mathbf{w}_0 + \frac{\partial}{\partial w_0} \mathbf{w}_1 \mathbf{x} \right) \right)$$

$$\frac{\partial \text{MSE}}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) * \left(\cancel{\frac{\partial}{\partial w_0} \mathbf{y}} - \left(\frac{\partial}{\partial w_0} \mathbf{w}_0 + \cancel{\frac{\partial}{\partial w_0} \mathbf{w}_1 \mathbf{x}} \right) \right)$$

$$\frac{\partial \text{MSE}}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) * (-1)$$

GD for minimizing regression error

$$\frac{\partial \textcolor{red}{MSE}}{\partial \textcolor{teal}{w}_0} = \frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) * (-\textcolor{yellow}{1})$$

$$\frac{\partial \textcolor{red}{MSE}}{\partial \textcolor{teal}{w}_0} = -\frac{2}{n} \sum_{i=1}^n (\mathbf{y} - \hat{\mathbf{y}}) = -\frac{2}{n} \sum_{i=1}^n e$$

$$\mathbf{w}_0^{\text{new}} = \mathbf{w}_0^{\text{old}} - \eta \frac{\partial \textcolor{red}{Error}}{\partial \textcolor{teal}{w}_0}$$

$$\mathbf{w}_0^{\text{new}} = \mathbf{w}_0^{\text{old}} - \eta \left(-\frac{2}{n} \sum_{i=1}^n e \right)$$

GD for minimizing regression error

$$\frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{\partial MSE}{\partial w_1} = \frac{\partial}{\partial w_1} \frac{1}{n} \sum_{i=1}^n e^2$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n e * \frac{\partial}{\partial w_1} e$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \left(\frac{\partial}{\partial w_1} y - \left(\frac{\partial}{\partial w_1} w_0 + \frac{\partial}{\partial w_1} w_1 x \right) \right)$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \left(\cancel{\frac{\partial}{\partial w_1} y} - \left(\cancel{\frac{\partial}{\partial w_1} w_0} + \frac{\partial}{\partial w_1} w_1 x \right) \right)$$

GD for minimizing regression error

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \left(\cancel{\frac{\partial}{\partial w_1} y} - \left(\cancel{\frac{\partial}{\partial w_1} w_0} + \frac{\partial}{\partial w_1} w_1 x \right) \right)$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * \left(- \left(\frac{\partial}{\partial w_1} w_1 x \right) \right)$$

$$\frac{\partial MSE}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * (-x)$$

$$\frac{\partial MSE}{\partial w_1} = - \frac{2}{n} \sum_{i=1}^n (y - \hat{y}) * x$$

GD for minimizing regression error

$$\frac{\partial \text{MSE}}{\partial w_1} = - \left[\frac{2}{n} \sum_{i=1}^n (y - \hat{y}) \right] * x = -x * \frac{2}{n} \sum_{i=1}^n e$$

$$w_1^{\text{new}} = w_1^{\text{old}} - \eta \frac{\partial \text{Error}}{\partial w_1}$$

$$w_1^{\text{new}} = w_1^{\text{old}} - \eta \left(-x * \frac{2}{n} \sum_{i=1}^n e \right)$$

GD for minimizing regression error

$$w_0^{new} = w_0^{old} - \eta \left(-\frac{2}{n} \sum_{i=1}^n e \right)$$



$$w_1^{new} = w_1^{old} - \eta \left(-x * \frac{2}{n} \sum_{i=1}^n e \right)$$

$$\hat{y} = w_0^{new} + w_1^{new} x$$

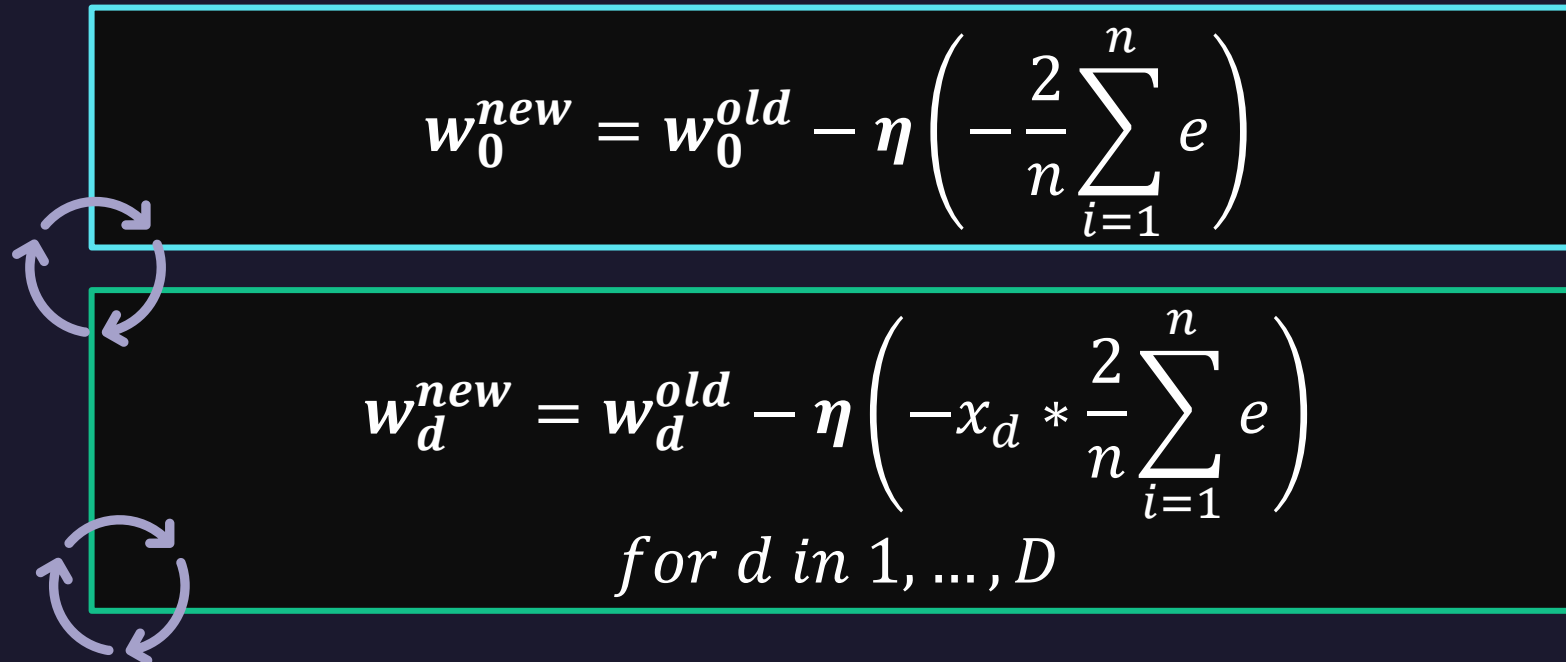
$$MSE = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$$

GD for minimizing regression error

- We minimized the MSE, but you can also minimize for MAE following the same steps.
- Some notation decide to use a modify the equation of the MSE so instead of dividing over n the number of data points, they divide on $2n$, the idea is to get rid of the $\frac{2}{n}$ in the final equation to be n only.
 - we used $J(W) = MSE = \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2$
 - Others may use $J(W) = \frac{1}{2n} \sum_{i=1}^n (y - \hat{y})^2$
- Also, some may change the order instead of calculating $(y - \hat{y})$ they use $(\hat{y} - y)$ it won't change the cost function because of the square power, but it would change the sign of the derivative, so instead of -1 and $-x$ you would get 1 and x .
- All ways lead to a correct formula we can use in gradient descent algorithm to find the optimal weights.

Multivariate linear regression update

- If you have D features (columns) or dimensions, that mean you have a linear equation like this
 - $\hat{y} = w_0 + w_1x_1 + \dots + w_Dx_D$
 - In this multivariate linear regression equation, you update each weight alone.
 - How? You can proof it by following the same steps.


$$w_0^{new} = w_0^{old} - \eta \left(-\frac{2}{n} \sum_{i=1}^n e \right)$$
$$w_d^{new} = w_d^{old} - \eta \left(-x_d * \frac{2}{n} \sum_{i=1}^n e \right)$$

for d in 1, ..., D

Batch Gradient Descent VS Stochastic Gradient Descent



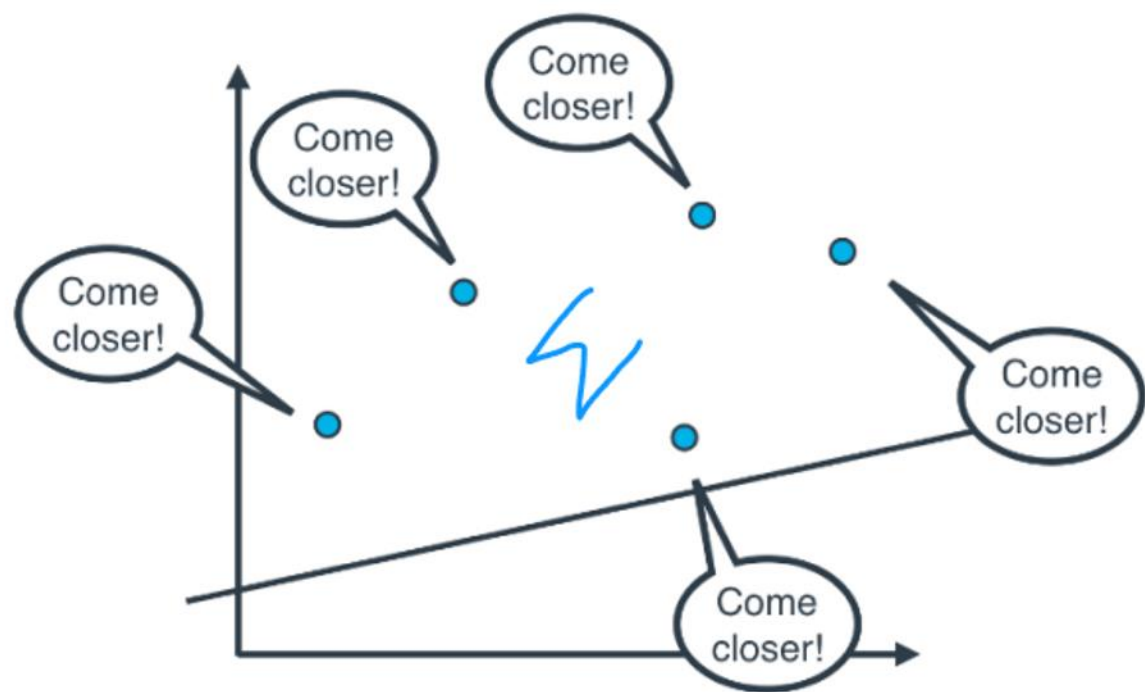
Batch gradient descent:

Calculate optimal values for all points, sum them, and update the weights accordingly.

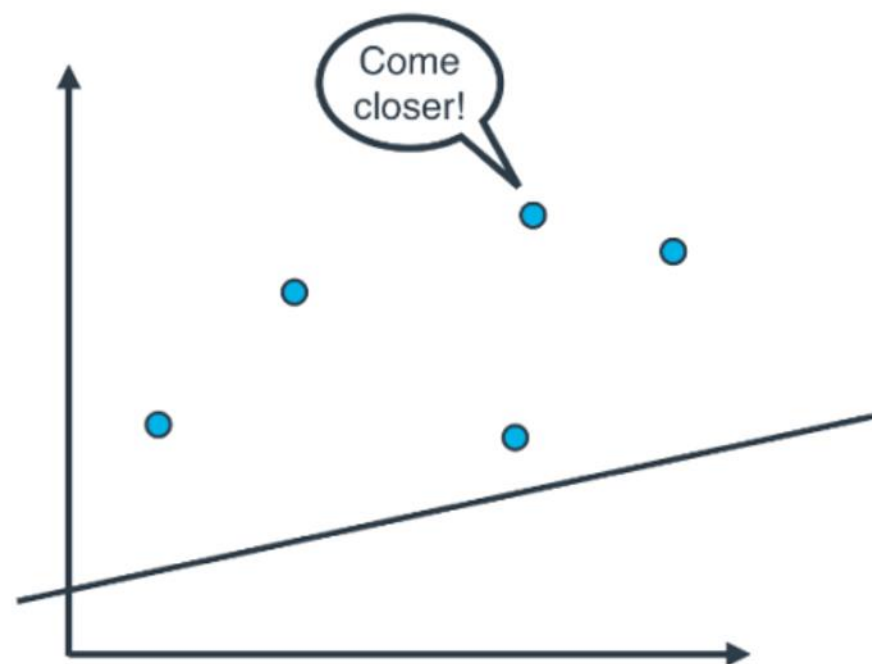


Stochastic gradient descent:

Calculate the optimal values for one point at a time, update the weights, and move on to the next point.



Batch



Stochastic



The question is, which one is used in practice?

- In most cases, **neither**.
- Think about this: If your data is huge, both are a bit slow, computationally. The best way to do linear regression, is to **split your data into many small batches. Each batch, with roughly the same number of points.** Then, use each batch to update your weights. This is still called ***mini-batch gradient descent***.

Numerical example 1

$$w_0^{new} = w_0^{old} - \eta \left(-\frac{2}{n} \sum_{i=1}^n e \right)$$

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

$$w_1^{new} = w_1^{old} - \eta \left(-x * \frac{2}{n} \sum_{i=1}^n e \right)$$

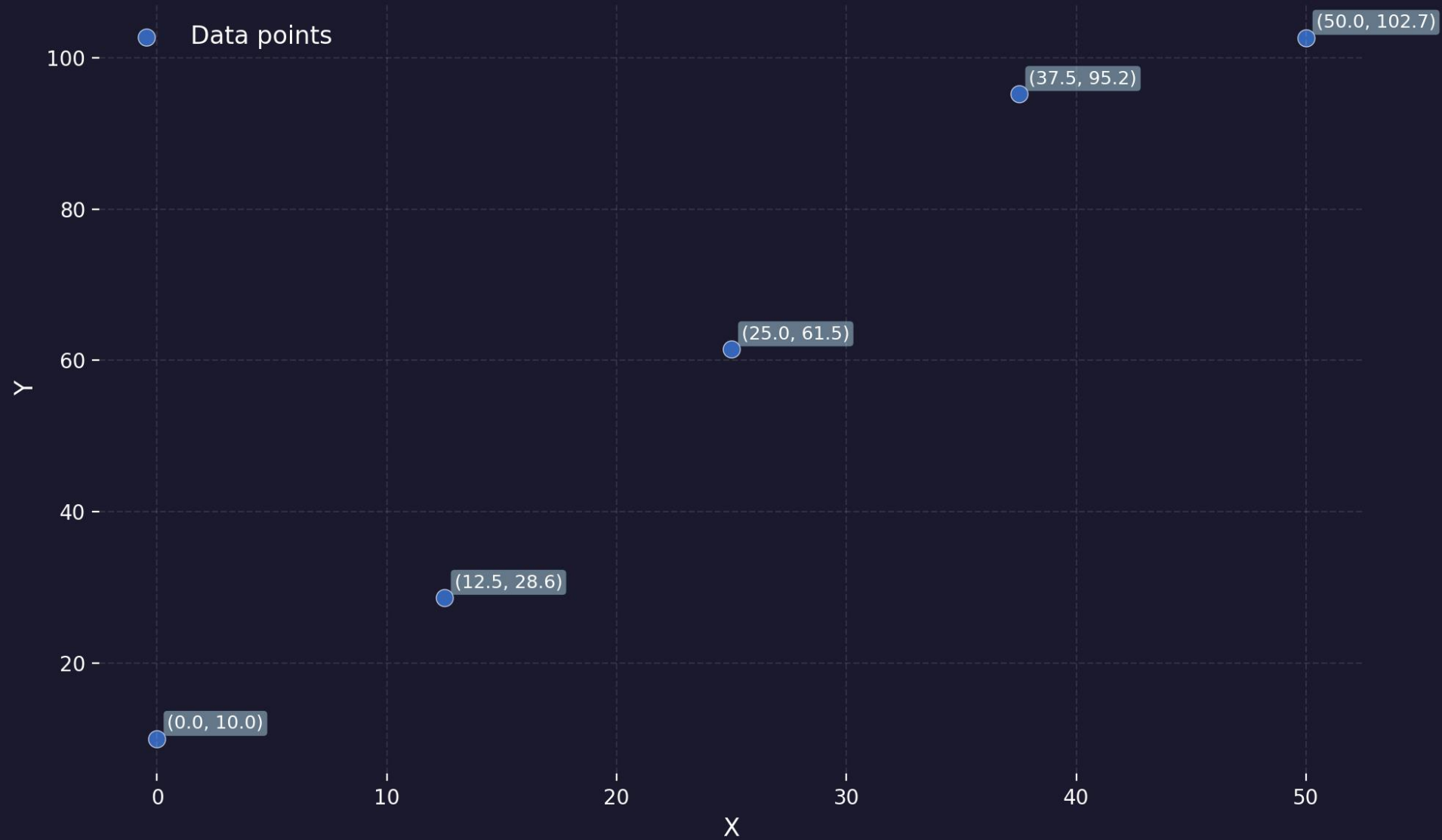
$$\hat{y} = w_0^{new} + w_1^{new} x$$

- We would use batch-GD, updating the weights using all the points.
- we need to choose initial values for w_0 and w_1 and to setup the η learning rate.
- Let's say it's $w_0 = 4, w_1 = 1.5, \eta = 0.03$

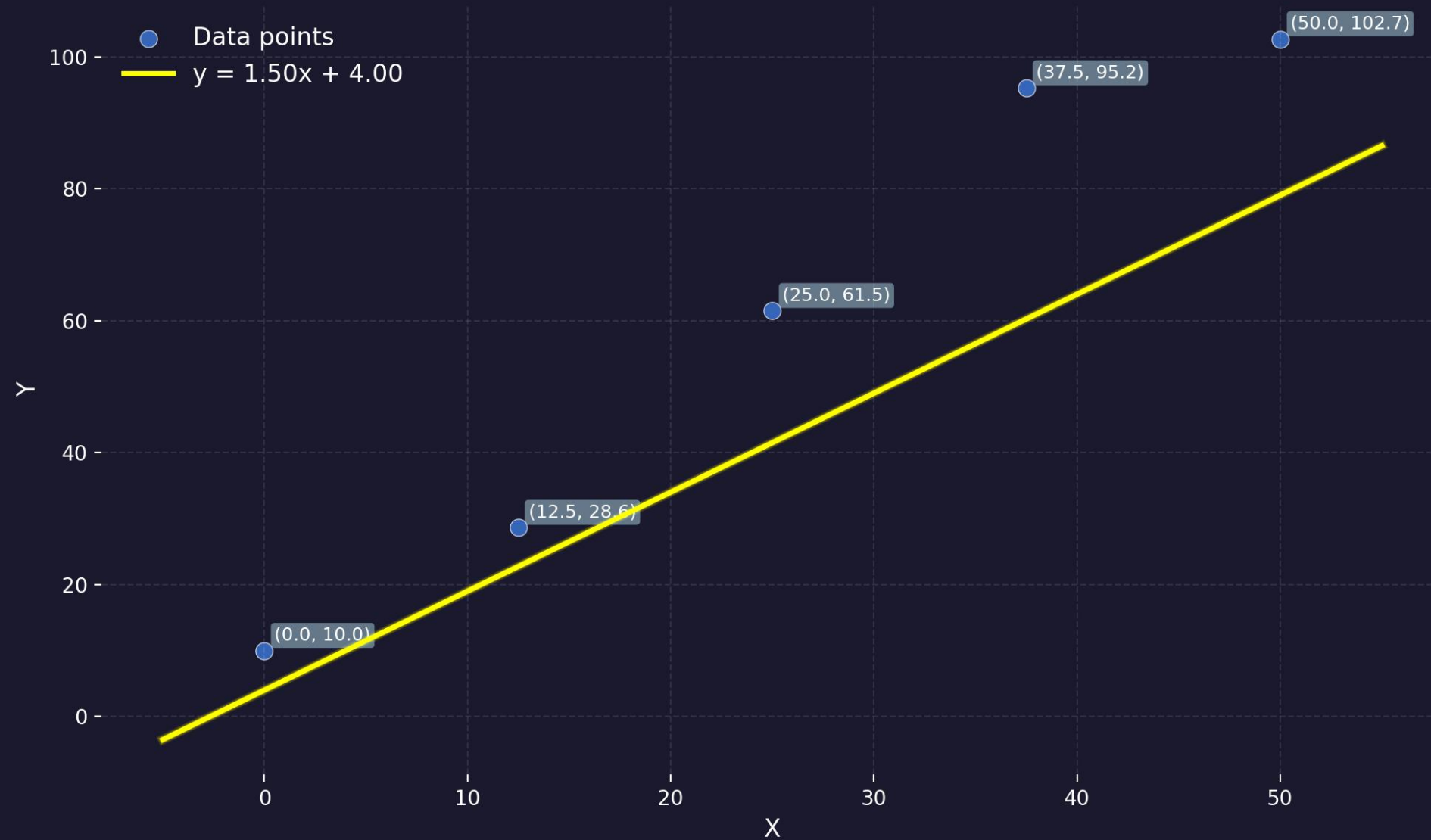
$$\hat{y} = 4 + 1.5 x$$

| Feature x | Target (y) |
|-------------|----------------|
| 0 | 9.96 |
| 12.5 | 28.61 |
| 25 | 61.47 |
| 37.5 | 95.23 |
| 50 | 102.65 |

Numerical example 1



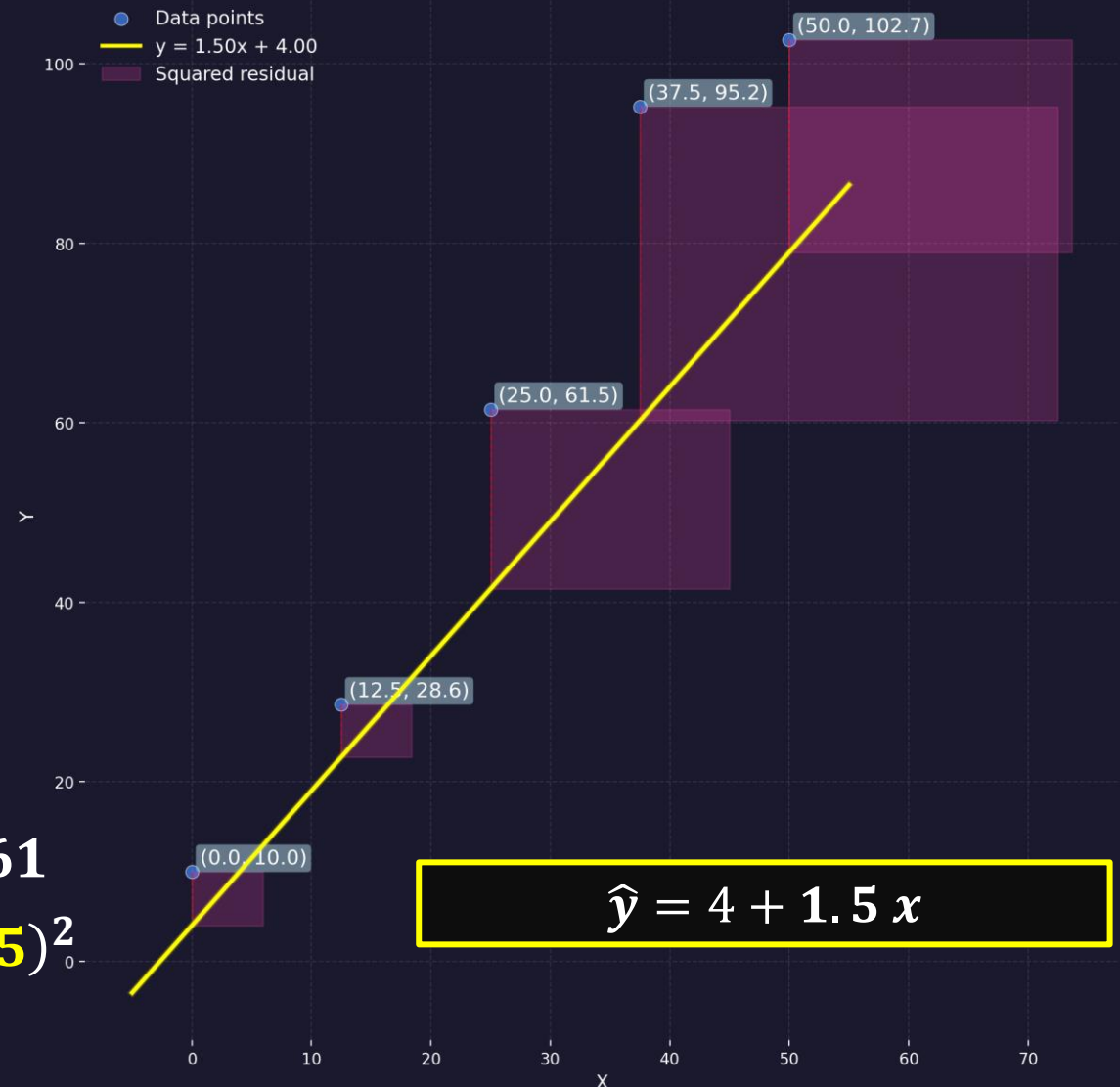
Numerical example 1



Numerical example 1

| Feature 1 (x) | Target (y) | Prediction ($\hat{y} = 4 + 1.5x$) |
|----------------------|-------------------|--|
| 0 | 9.96 | $4 + 1.5(0) = 4$ |
| 12.5 | 28.61 | $4 + 1.5(12.5) = 22.75$ |
| 25 | 61.47 | $4 + 1.5(25) = 41.5$ |
| 37.5 | 95.23 | $4 + 1.5(37.5) = 60.25$ |
| 50 | 102.65 | $4 + 1.5(50) = 79$ |

$$\begin{aligned}
 \text{MSE} &= \frac{1}{n} \sum_{i=1}^n (y - \hat{y})^2 = \frac{1}{5} \left((9.96 - 4)^2 + (28.61 - 22.75)^2 + (61.47 - 41.5)^2 + (95.23 - 60.25)^2 + (102.65 - 79)^2 \right) = 450.317
 \end{aligned}$$



Numerical example 1

$$w_0^{new} = w_0^{old} - \eta \left(-\frac{2}{n} \sum_{i=1}^n e \right)$$

$$w_1^{new} = w_1^{old} - \eta \left(-x * \frac{2}{n} \sum_{i=1}^n e \right)$$

$$\hat{y} = 4 + 1.5x$$



| Feature I (x) | Target (y) | Prediction ($\hat{y} = 4 + 1.5x$) |
|------------------|---------------|--|
| 0 | 9.96 | $4 + 1.5(0) = 4$ |
| 12.5 | 28.61 | $4 + 1.5(12.5) = 22.75$ |
| 25 | 61.47 | $4 + 1.5(25) = 41.5$ |
| 37.5 | 95.23 | $4 + 1.5(37.5) = 60.25$ |
| 50 | 102.65 | $4 + 1.5(50) = 79$ |

$$w_0^{new} = 4 - 0.03 * \left(-\frac{2}{5} ((9.96 - 4)) + ((28.61 - 22.75)) + (61.47 - 41.5) + (95.23 - 60.25) + (102.65 - 79) \right) = 5.08504$$


$$w_1^{new} = 1.5 - 0.03 * \left(-\frac{2}{5} (0 * (9.96 - 4)) + (12.5 * (28.61 - 22.75)) + (25 * (61.47 - 41.5)) + (37.5 * (95.23 - 60.25)) + (50 * (102.65 - 79)) \right) = 38.30100$$

Numerical example 1

- If you continue your algorithm would overshoot or never converge to good weights that minimize the loss.
- Because you didn't standardized/normalized the scale of you variables, and GD algorithm is sensitive to scaling.

```
Iteration 1: w0 = 5.08504, w1 = 38.30100
Iteration 2: w0 = -49.09652, w1 = -1996.58181
Iteration 3: w0 = 2952.29702, w1 = 110511.96579
Iteration 4: w0 = -162989.21444, w1 = -6110087.37922
Iteration 5: w0 = 9011924.78230, w1 = 337826938.69945
Iteration 6: w0 = -498269195.17877, w1 = -18678456123.14185
Iteration 7: w0 = 27549311144.81977, w1 = 1032732104723.53137
Iteration 8: w0 = -1523201804605.59180, w1 = -57099772752565.16406
Iteration 9: w0 = 84217849432522.06250, w1 = 3157047247286260.50000
Iteration 10: w0 = -4656406092462816.00000, w1 = -174553187186714528.00000

Predicted Y values:
[-4.65640609e+15 -2.18657125e+18 -4.36848609e+18 -6.55040093e+18
-8.73231577e+18]
```



Numerical example 1

- Gradient descent can converge faster when the data is scaled.
- Scaling the data allows you to use a **consistent learning rate across all features**, preventing situations where large features require a much smaller learning rate to avoid overshooting, while smaller features would require a larger one (happened in our example).

```
Iteration 1: w0 = 0.80000, w1 = 1.08767
Iteration 2: w0 = 0.16000, w1 = 1.00520
Iteration 3: w0 = 0.03200, w1 = 0.98871
Iteration 4: w0 = 0.00640, w1 = 0.98541
Iteration 5: w0 = 0.00128, w1 = 0.98475
Iteration 6: w0 = 0.00026, w1 = 0.98462
Iteration 7: w0 = 0.00005, w1 = 0.98459
Iteration 8: w0 = 0.00001, w1 = 0.98459
Iteration 9: w0 = 0.00000, w1 = 0.98459
Iteration 10: w0 = 0.00000, w1 = 0.98459
Predicted Y values (reversed scaling):
[ 9.18401212  34.38401348  59.58401483  84.78401618 109.98401753]

Original w0 and w1 after reversing scaling:
w0_original = 9.18401
w1_original = 2.01600
```



Numerical example 1

- We would use standardization $Z = \frac{x-\mu}{\sigma}$ one scaler for each column
 - $X_{scaled} = \frac{X-\mu_X}{\sigma_X}$
 - $Y_{scaled} = \frac{Y-\mu_Y}{\sigma_Y}$
- $X_{scaled} = [-1.41421356, -0.70710678, 0. , 0.70710678, 1.41421356]$
- $Y_{scaled} = [-1.37097471, -0.85572648, 0.052105 , 0.98480099, 1.1897952]$
- To return the founded weights (w_0, w_1) to the original scale we can use these formulas
 - $w_0^{original} = w_0^{scald} \cdot \sigma_Y + \mu_Y - w_1^{scaled} \cdot \mu_X$
 - $w_1^{original} = w_1^{scaled} \cdot \frac{\sigma_Y}{\sigma_X}$
 - All these formulas just a manipulation for our model
 - $Y_{scaled} = w_0^{scaled} + w_1^{scaled} X_{scaled}$

Numerical example 1

```
from sklearn.preprocessing import StandardScaler
scaler_X, scaler_Y = StandardScaler(), StandardScaler()
X_values_scaled = scaler_X.fit_transform(np.array(X_values).reshape(-1, 1))
Y_values_scaled = scaler_Y.fit_transform(np.array(Y_values).reshape(-1, 1))

# Run gradient descent on the scaled data
w_0_new, w_1_new = gradient_descent(X_values_scaled, Y_values_scaled, iterations=10)

# Revert the scaled predictions back to the original scale
Y_pred_scaled = (w_0_new + w_1_new * X_values_scaled) # line equation
# Inverse transform to get back to original scale
Y_pred = scaler_Y.inverse_transform(Y_pred_scaled.reshape(-1, 1))
```

Numerical example 1

```
# Reverse the scaling to get the original space weights
w1_original = w_1_new * (scaler_Y.scale_ / scaler_X.scale_)
w0_original = w_0_new * scaler_Y.scale_ + scaler_Y.mean_ - w1_original * scaler_X.mean_

# Print the original w0 and w1
print("\nOriginal w0 and w1 after reversing scaling:")
w0_original = w0_original.item()
w1_original = w1_original.item()
print(f"w0_original = {w0_original:.5f}")
print(f"w1_original = {w1_original:.5f}")
w0_original + (np.dot(w1_original, X_values))
```


Numerical example 1

```
# (GD) Initialize weights
w0, w1 = w0_init, w1_init
# Perform iterations
for i in range(iterations):
    # Compute predictions
    Y_pred = w0 + w1 * X

    # Compute errors
    errors = Y - Y_pred

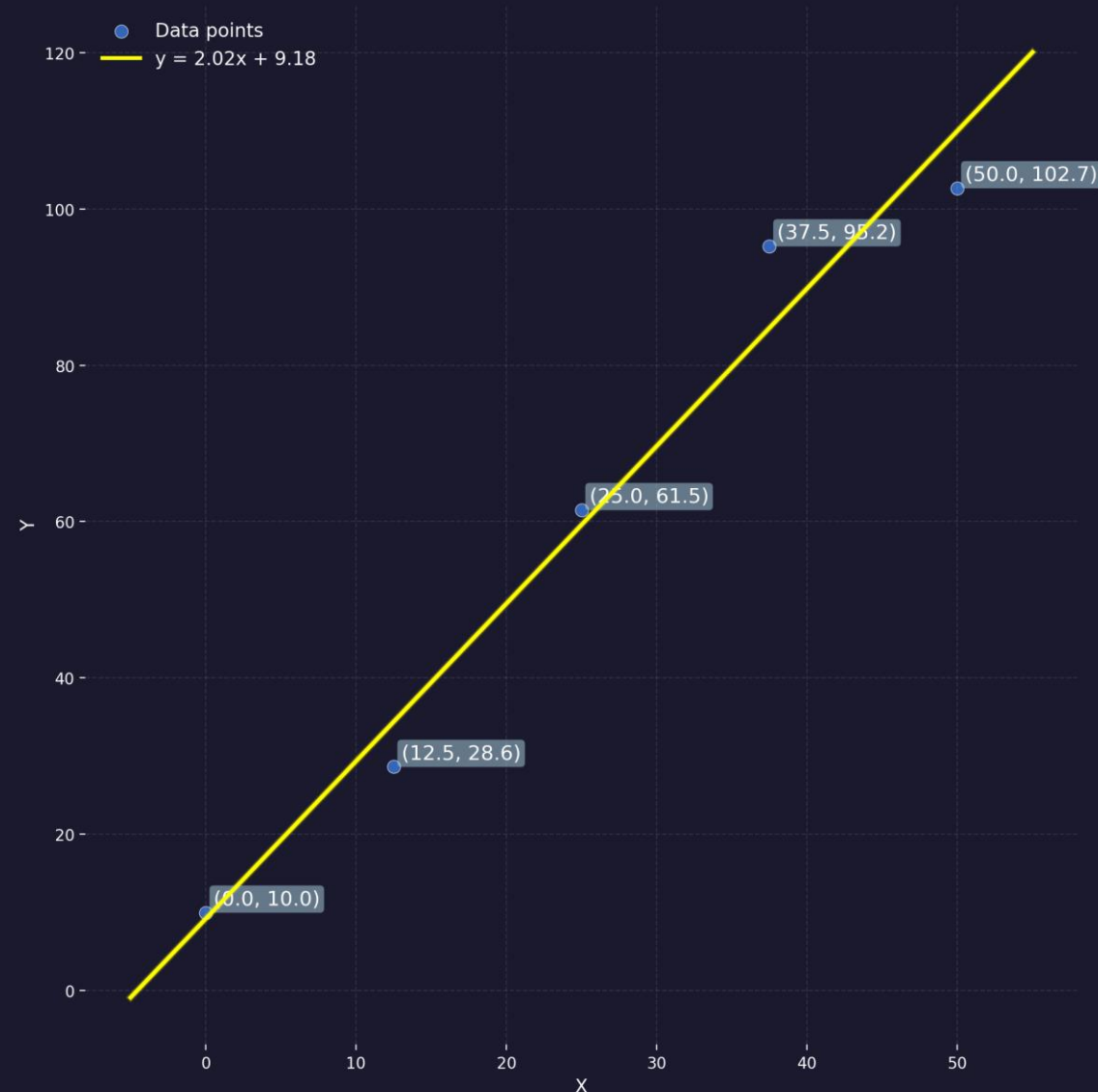
    # Compute gradients
    dw0 = - (2/n) * np.sum(errors)
    dw1 = - (2/n) * np.sum(X * errors)

    # Update weights
    w0 -= learning_rate * dw0
    w1 -= learning_rate * dw1

    # Print progress
    print(f"Iteration {i+1}: w0 = {w0:.5f}, w1 = {w1:.5f}")
```

Numerical example 1

- Using the new weights we produce better line, that fit the data in a good way.
- $\hat{y} = 2.02x + 9.18$ this equation produced **MSE = 40**
- The data is generated using the equation $y = 2x + 5 + \epsilon$, the last term ϵ is a bias (noise) or deviation factor that add some randomness to each point generated without it all the points would be in the same line $2x + 5$, its **MSE=61**.



Normal equation

- Linear regression equation $\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d$ we can rewrite it as
 - $\hat{y} = w_0 + \sum_{i=1}^d w_i x_i$
 - And to simplify it more we can make an artificial neutral feature $x_0 = 1$ to get
 - $\hat{y} = \sum_{i=0}^d w_i x_i$
- We can write it in vector notation as $\hat{y}_i = \mathbf{x}_i^T \mathbf{w}_i$ where i is one sample, \mathbf{x}_i is the features vector including the artificial feature, and \mathbf{w}_i is the weights vectors
 - This formula use dot product, so the output is a scaler

$$\bullet \mathbf{x}_i^T = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix}^T \text{ and } \mathbf{w}_i = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$$

Normal equation

- Let's say we have several samples, where each sample has its own feature set

$$\bullet \mathbf{x}_1^T = \begin{bmatrix} \mathbf{1} \\ x_{11} \\ x_{12} \\ \dots \\ x_{1d} \end{bmatrix}^T, \mathbf{x}_2^T = \begin{bmatrix} \mathbf{1} \\ x_{21} \\ x_{22} \\ \dots \\ x_{2d} \end{bmatrix}^T, \dots, \mathbf{x}_n^T = \begin{bmatrix} \mathbf{1} \\ x_{n1} \\ x_{n2} \\ \dots \\ x_{nd} \end{bmatrix}^T \text{ and } \mathbf{w}_i = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$$

- Notice that the weights would be the same for all samples
- Let's transpose them
 - $\mathbf{x}_1^T = [\mathbf{1}, x_{11}, x_{12}, \dots, x_{1d}]$,
 - $\mathbf{x}_2^T = [\mathbf{1}, x_{21}, x_{22}, \dots, x_{2d}]$,
 - \dots ,
 - $\mathbf{x}_n^T = [\mathbf{1}, x_{n1}, x_{n2}, \dots, x_{nd}]$
- Why not putting them in a matrix?
 - $\mathbf{X} = \begin{bmatrix} \mathbf{1} & \dots & x_{1d} \\ \vdots & \ddots & \vdots \\ \mathbf{1} & \dots & x_{nd} \end{bmatrix}$

Normal equation

- Why not putting them in a matrix?

- $$X = \begin{bmatrix} 1, x_{11}, x_{12}, \dots, x_{1d} \\ 1, x_{21}, x_{22}, \dots, x_{2d} \\ \dots \\ 1, x_{n1}, x_{n2}, \dots, x_{nd} \end{bmatrix}$$

The line equation $\hat{y} = Xw$

$$XW = \begin{bmatrix} 1, x_{11}, x_{12}, \dots, x_{1d} \\ 1, x_{21}, x_{22}, \dots, x_{2d} \\ \dots \\ 1, x_{n1}, x_{n2}, \dots, x_{nd} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$$

- X is matrix of shape $\mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{\text{samples} \times \text{features}}$, each column in this matrix represent a feature while the row is the observation (sample), it's like how we handle the datasets in DataFrames.
- \hat{y} in this notation is a vector of predictions and its shape is $\mathbb{R}^{n \times 1} \rightarrow \mathbb{R}^{\text{features} \times 1}$
 - a prediction for each sample

Normal equation

- The **Normal Equation** is a mathematical approach to finding the optimal parameters (weights) for a linear regression model.
- It provides a closed-form solution to the problem of minimizing the cost function (specifically, RSS) and finding the best-fitting line to a set of data points.

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

- Linear Regression has a closed-form solution because its cost function (RSS or MSE) is **quadratic**, ensuring **convexity**. Unlike iterative methods, the Normal Equation provides a direct solution—simply plug in the values to obtain the result. However, not all machine learning algorithms have such analytical solutions.

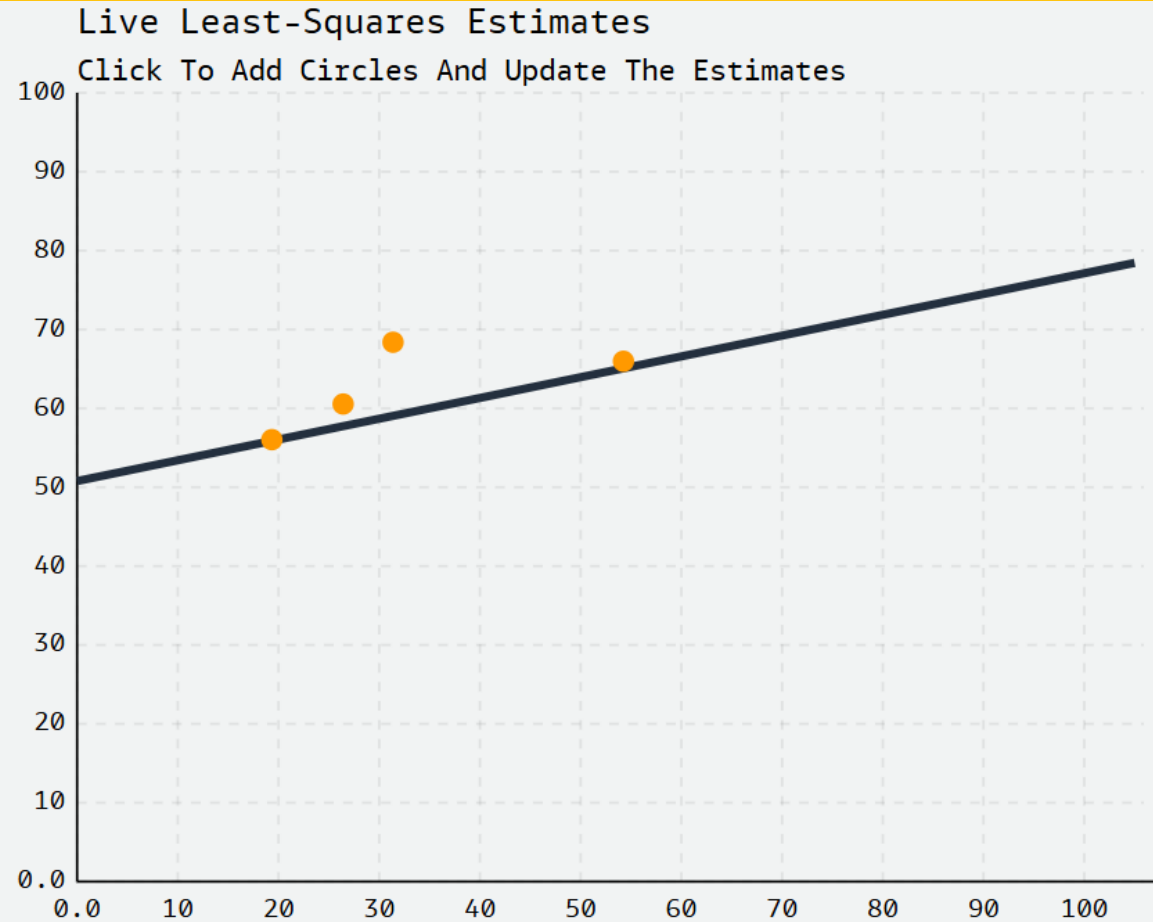
Normal equation

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

$$= \begin{pmatrix} \begin{bmatrix} 1 & 19.3 \\ 1 & 26.4 \\ \vdots & \vdots \\ 1 & 54.2 \end{bmatrix}^T \begin{bmatrix} 1 & 19.3 \\ 1 & 26.4 \\ \vdots & \vdots \\ 1 & 54.2 \end{bmatrix} \end{pmatrix}^{-1} \begin{bmatrix} 1 & 19.3 \\ 1 & 26.4 \\ \vdots & \vdots \\ 1 & 54.2 \end{bmatrix}^T \begin{bmatrix} 56.0 \\ 60.5 \\ \vdots \\ 66.0 \end{bmatrix}$$

$$= \begin{pmatrix} 4 & 677 \\ 677 & 126877 \end{pmatrix}^{-1} \begin{bmatrix} 1 & 19.3 \\ 1 & 26.4 \\ \vdots & \vdots \\ 1 & 54.2 \end{bmatrix}^T \begin{bmatrix} 56.0 \\ 60.5 \\ \vdots \\ 66.0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 50.77 \\ 1 & 0.26 \end{bmatrix}$$



Normal equation

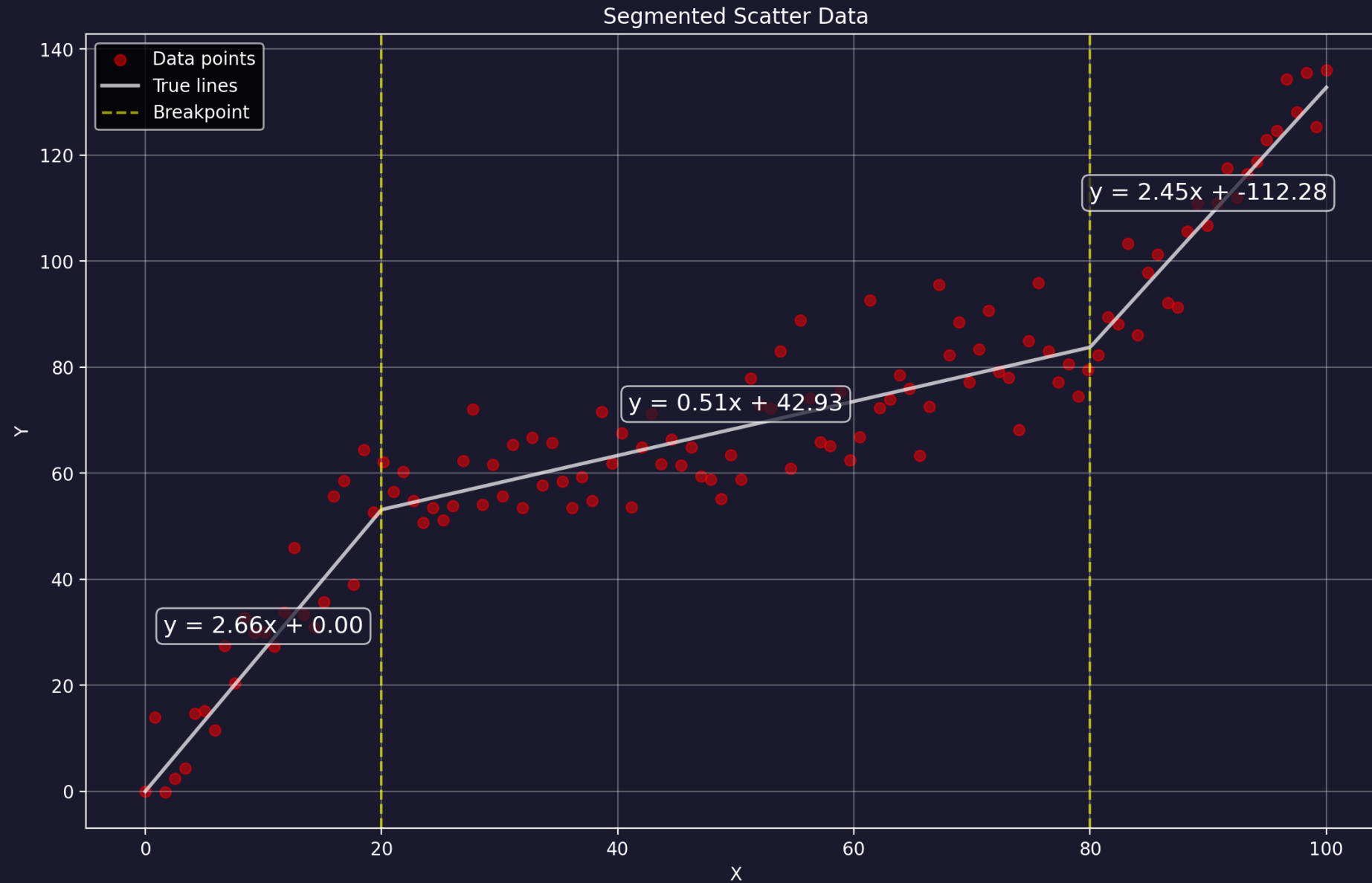
- $X^T X$ must be invertible, and it has $O(n^3)$ making it inefficient for high-dimensional datasets
- It's simple and easy to implement. It only requires basic matrix operations like multiplication and inversion.
- Unlike iterative methods (e.g., Gradient Descent), there is no need to tune learning rates or other hyperparameters.
- The Normal Equation does not scale well with large numbers of features or data points.
- GD is computationally faster for larger datasets, that is why it's used although there is an analytical solution, time complexity for GD is $O(k * n * d) = O(\text{iterations} * \text{samples} * \text{features})$, the complexity grows linearly while normal equation grows polynomially (3rd degree)

Piecewise Linear Regression

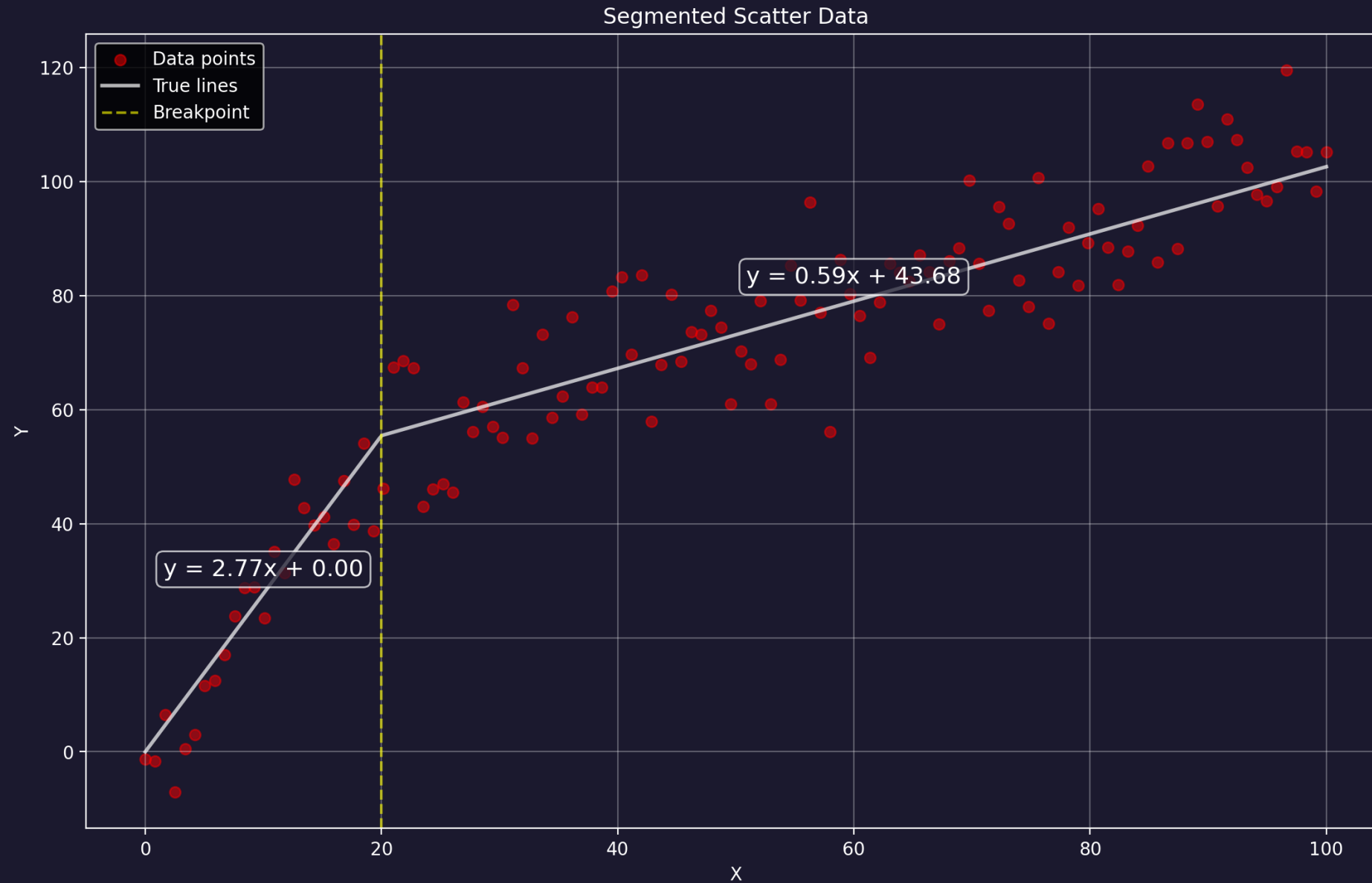
- A **Piecewise Linear Regression** model, also known as **Segmented Linear Regression**, involves fitting different linear regression models to different segments of the data.
- The basic idea is that the data is divided into multiple regions, and within each region, a different linear model is applied.
- These regions are defined by **breakpoints** where the slope of the regression line changes.
- The model can be formulated as where x_1, x_2, \dots, x_k are the breakpoints

$$\hat{y} = \begin{cases} w_0^1 + w_1^1 x, & \text{if } x \leq x_1 \\ w_0^2 + w_1^2 x, & \text{if } x_1 \leq x \leq x_2 \\ \vdots & \\ w_0^{k+1} + w_1^{k+1} x, & \text{if } x > x_k \end{cases}$$

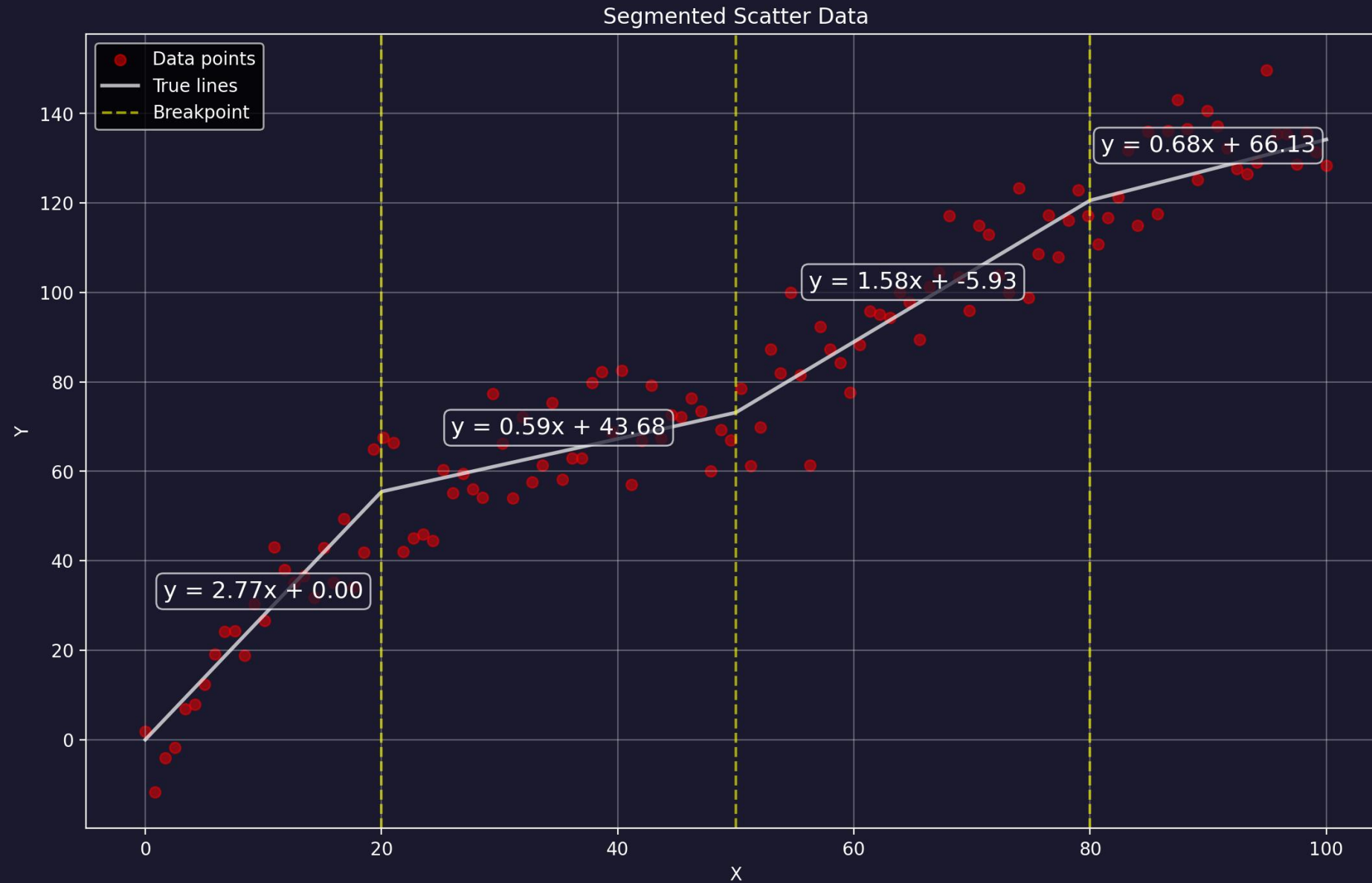
Piecewise Linear Regression



Piecewise Linear Regression



Piecewise Linear Regression



Polynomial regression

- Polynomial regression is an extension of **linear regression** where we model the relationship between the independent variable x and the dependent variable y as n -degree polynomial.
 - It is useful when the data exhibits **non-linear** trends that a simple linear model cannot capture.

- Polynomial regression model of degree n is given by

$$\hat{y} = w_0 + w_1x^1 + w_2x^2 + w_3x^3 + \dots + w_dx^d$$

- It's still linear regression model, you can think of polynomial regression as a feature engineering process, we take the features (data) we have and transform them polynomially so we get a higher dimensional space, where the line is no longer a line but a higher n -dimensional line (dot, line, plane, cube,...) and when we project it down to our original space we got a higher dimensional model that can model complex data pattern.

Polynomial regression

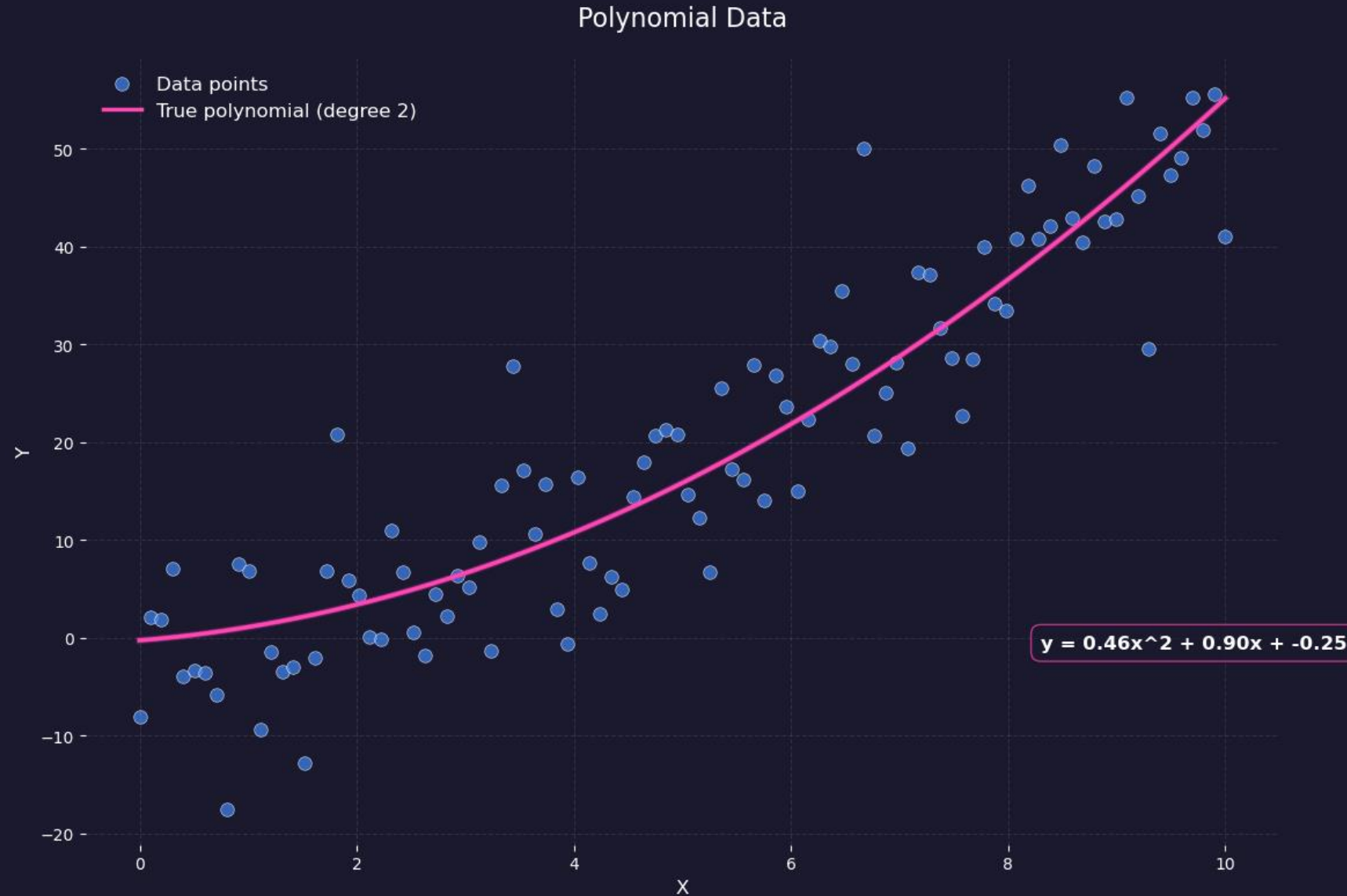
The line equation $\hat{\mathbf{y}} = X_{transformed} \mathbf{w}$

$$\hat{\mathbf{y}} = \mathbf{X}_t \mathbf{W} = \begin{bmatrix} 1, x_1, x_1^2, \dots, x_1^d \\ 1, x_2, x_2^2, \dots, x_2^d \\ \dots \\ 1, x_n, x_n^2, \dots, x_n^d \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_d \end{bmatrix}$$

- If you have more than one feature the matrix would include them too, so you just concatenate them in the matrix.
 - We can also use the product of the feature to capture the interaction between them
 - So, the transformed feature set can be written as (for degree = 2)

$$\Phi(X) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$

Polynomial regression



See

- <https://mlu-explain.github.io/linear-regression/> (Very impressive visual article 🌟)
- [People hate \$R^2\$](#) (about the cons of R-squared)
- <https://observablehq.com/@yizhe-ang/interactive-visualization-of-linear-regression> (Another visualization)
- https://aegis4048.github.io/mutiple_linear_regression_and_visualization_in_python#Multiple%20linear%20regression
- <https://online.stat.psu.edu/stat501/lesson/8/8.8> (Piecewise Linear Regression Models)
- <https://piecewise-regression.readthedocs.io/en/latest/> (Piecewise Linear Regression Models)
- https://jazznbass.github.io/scan-Book/ch_piecewise_regression.html (Piecewise Linear Regression Models)
- https://www.fs.usda.gov/rm/pubs/rmrs_gtr189.pdf (Piecewise Linear Regression Models)
- https://visualize-it.github.io/polynomial_regression/simulation.html (polynomial regression visualization 🌟)

THANK YOU

- Time for code now

