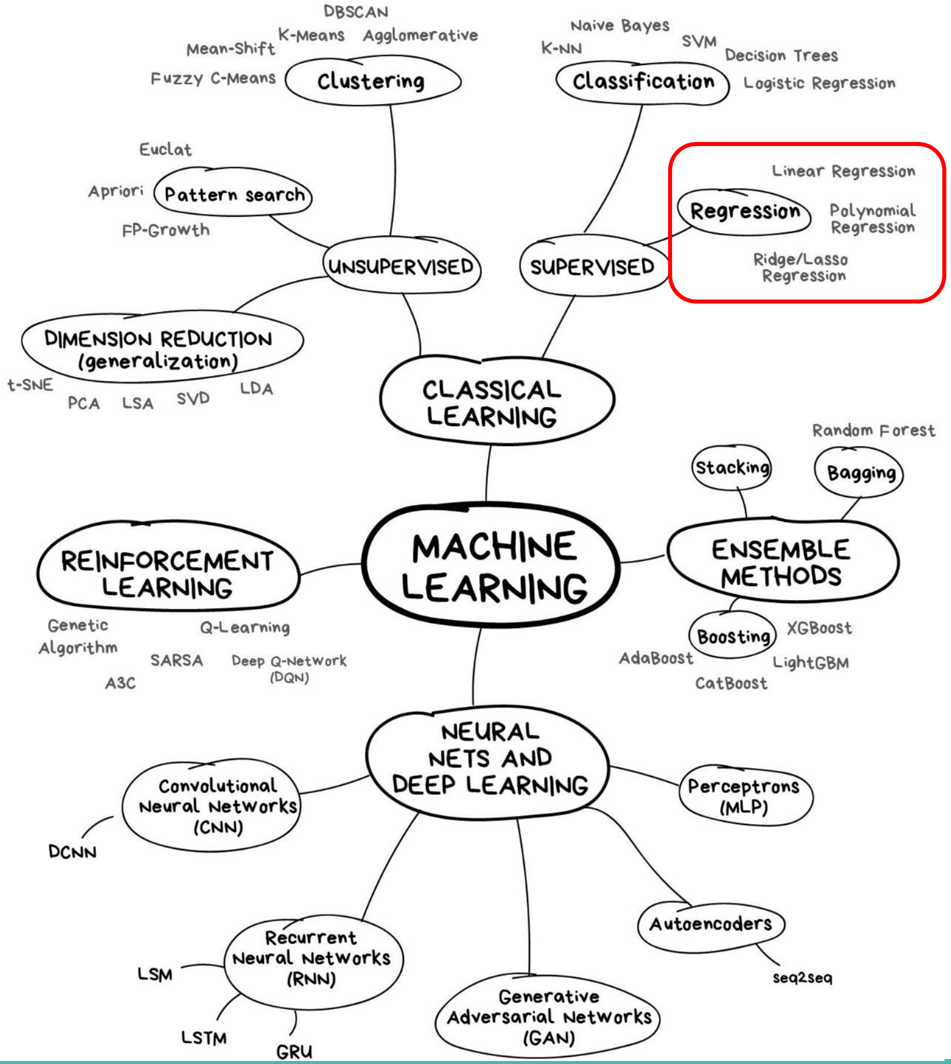
Machine Learning

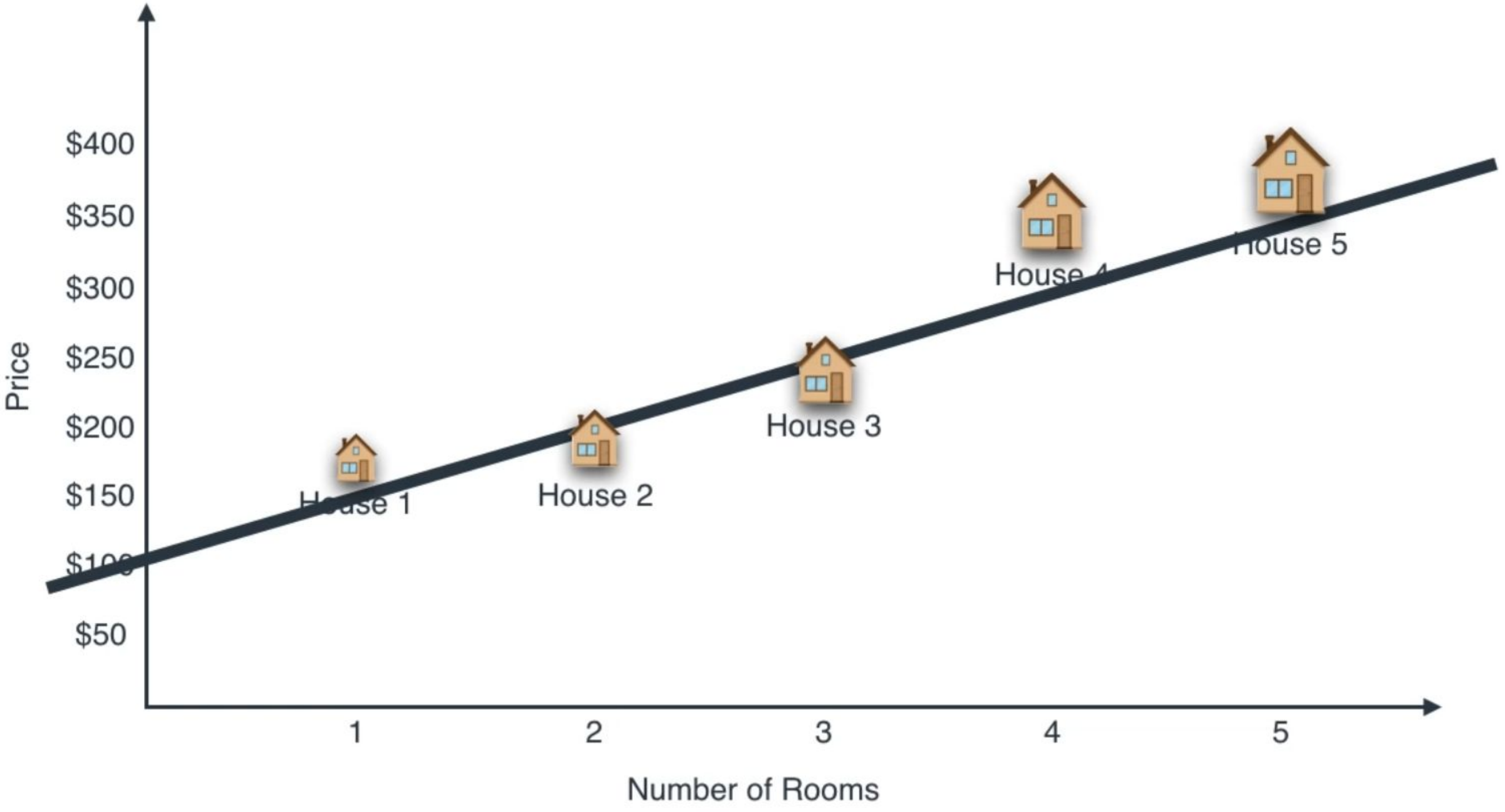
Linear Regression

Source:
https://youtu.be/wYPUhge9w5c?si=5viLbgNf9v46upP_
https://youtu.be/PaFPbb66DxQ?si=uhqgmEm9_0Zdu0_X

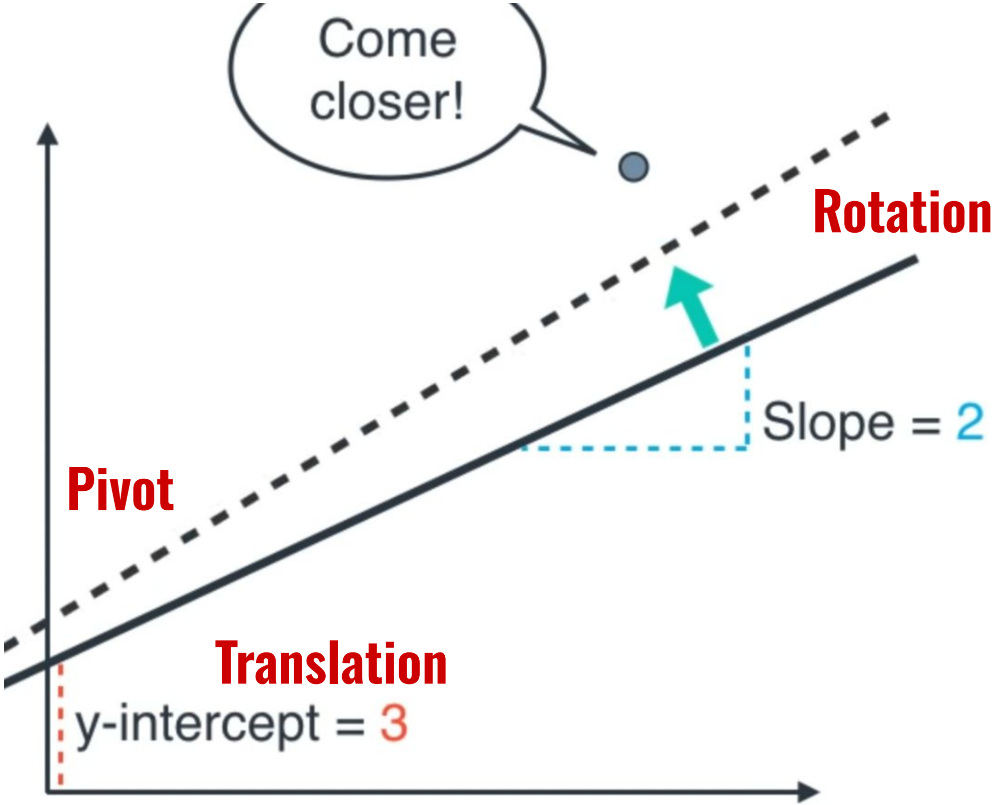
Machine Learning Algorithms



What is LR – Housing Price Example



How is LR



$$y = 2x + 3$$

+0.01 +0.01

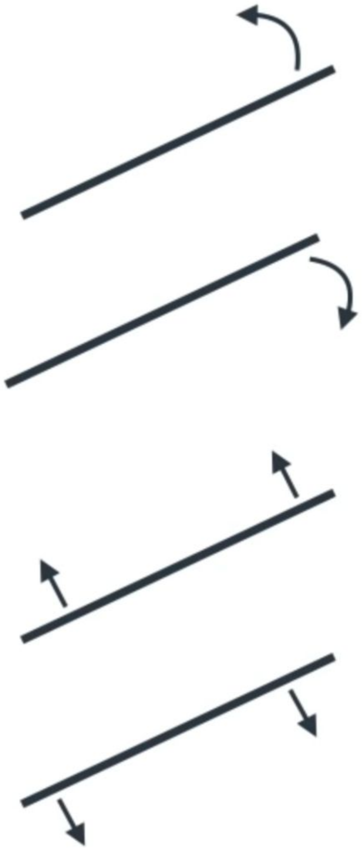
$$y = 2.01x + 3.01$$

Step 1: Pick a small number. 0.01 (learning rate)

Step 2:

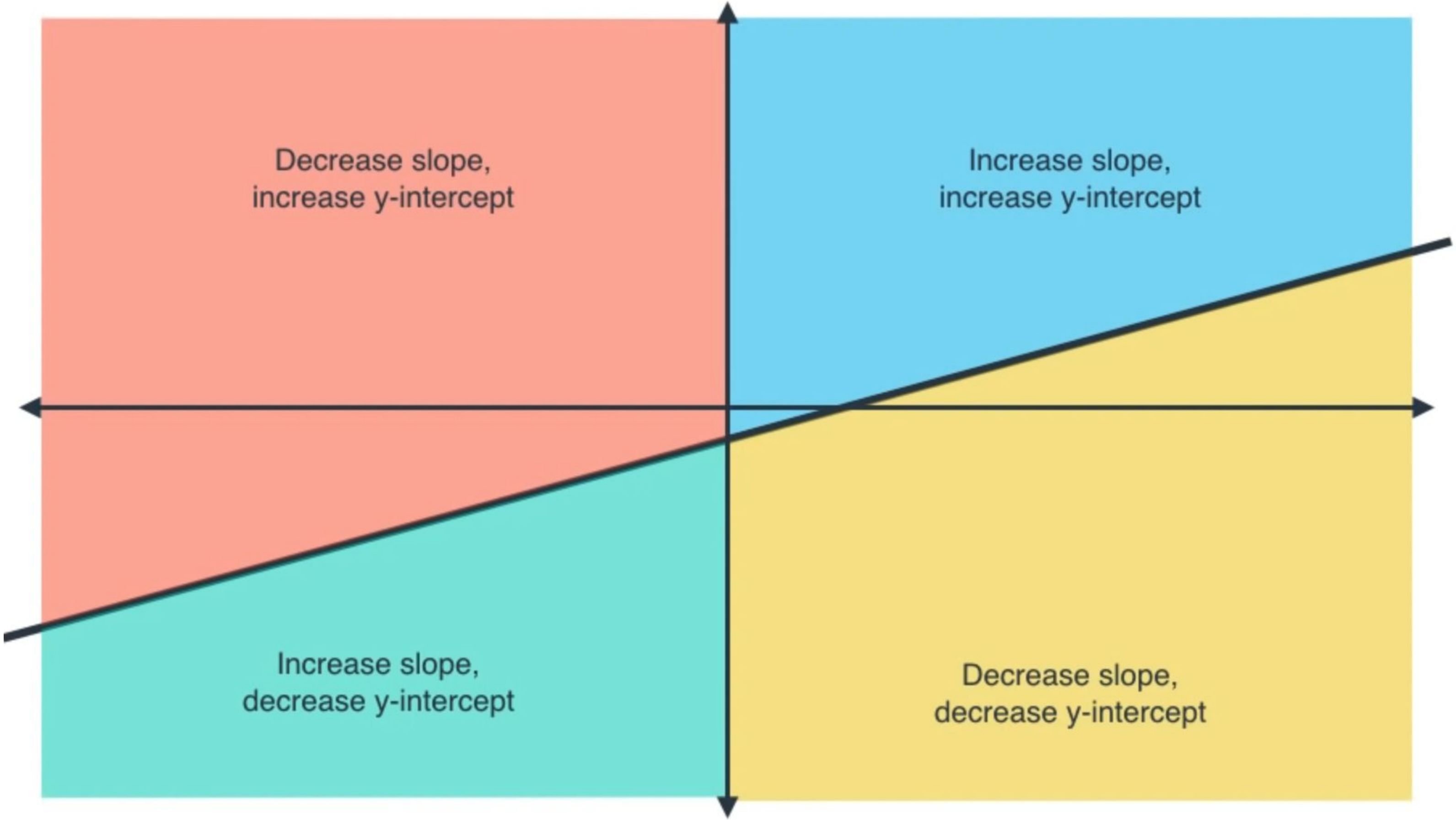
- Add learning rate to slope
- Add learning rate to y-intercept

How to move a line

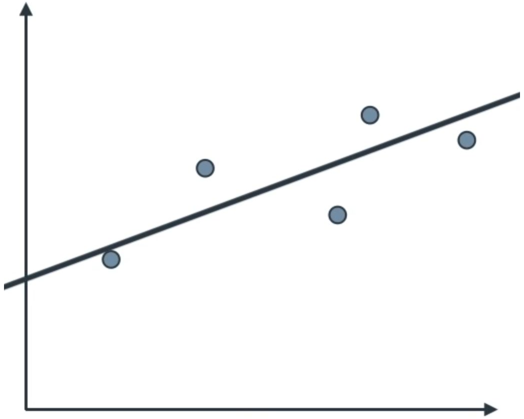


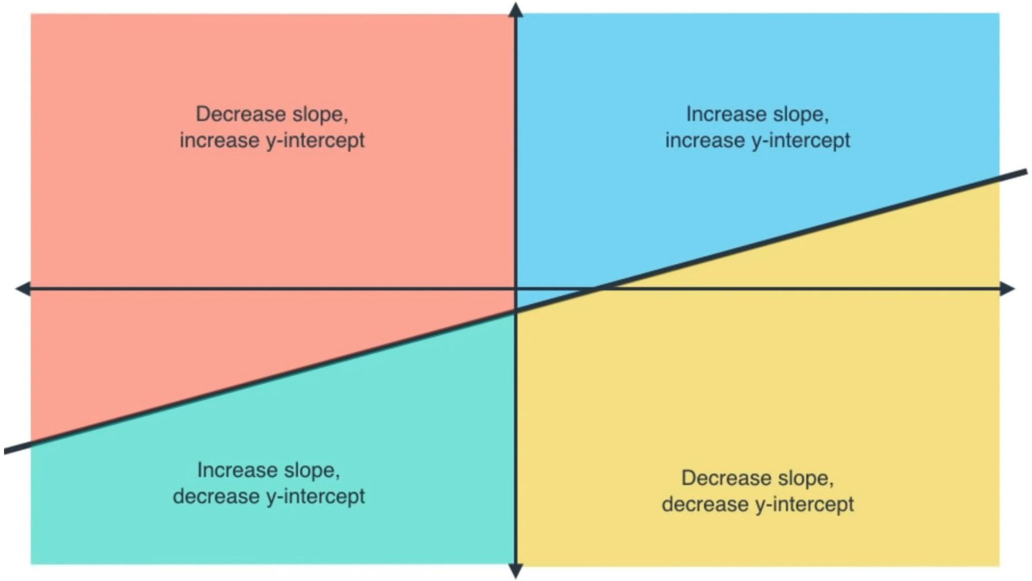
Rotate line counter-clockwise	Increase slope
Rotate line clockwise	Decrease slope
Translate line up	Increase y-intercept
Translate line down	Decrease y-intercept

How to move a line



LR Algorithm





Step 1: Start with a random line

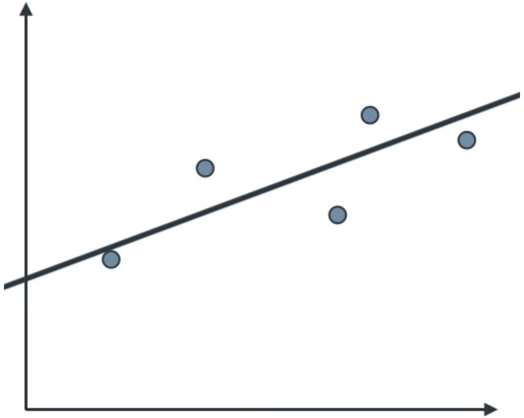
Step 2: Pick a large number. 1000 (number of repetitions, or epochs)

Step 3: Pick a small number. 0.01 (learning rate)

Step 4: (repeat 1000 times)

- Pick random point
  - If point **above** line, and to the **right** of the y-axis:
    - add 0.01 to slope
    - add 0.01 to y-intercept
  - If point **above** line, and to the **left** of the y-axis:
    - subtract 0.01 to slope
    - add 0.01 to y-intercept
  - If point **below** line, and to the **right** of the y-axis:
    - subtract 0.01 to slope
    - subtract 0.01 to y-intercept
  - If point **below** line, and to the **left** of the y-axis:
    - add 0.01 to slope
    - subtract 0.01 to y-intercept

LR Algorithm Improvement



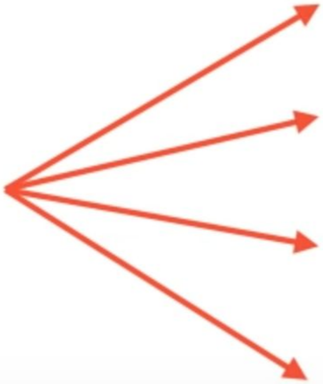
Step 1: Start with a random line

Step 2: Pick a large number. 1000
(number of repetitions, or epochs)

Step 3: Pick a small number. 0.01
(learning rate)

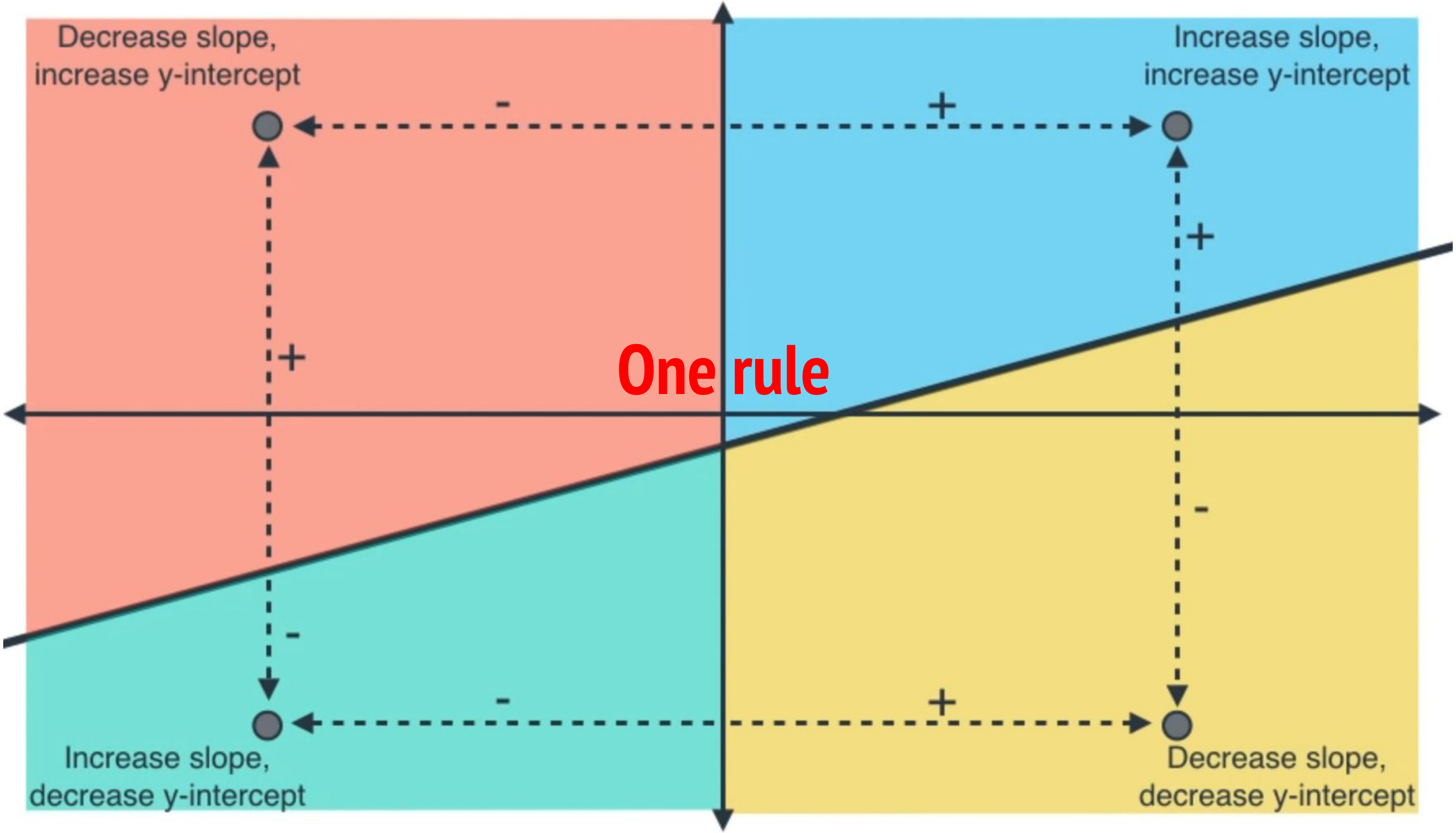
Step 4: (repeat 1000 times)

- Pick random point
  - If point **above** line, and to the **right** of the y-axis:
    add 0.01 to slope
    add 0.01 to y-intercept
  - If point **above** line, and to the **left** of the y-axis:
    subtract 0.01 to slope
    add 0.01 to y-intercept
  - If point **below** line, and to the **right** of the y-axis:
    subtract 0.01 to slope
    subtract 0.01 to y-intercept
  - If point **below** line, and to the **left** of the y-axis:
    add 0.01 to slope
    subtract 0.01 to y-intercept

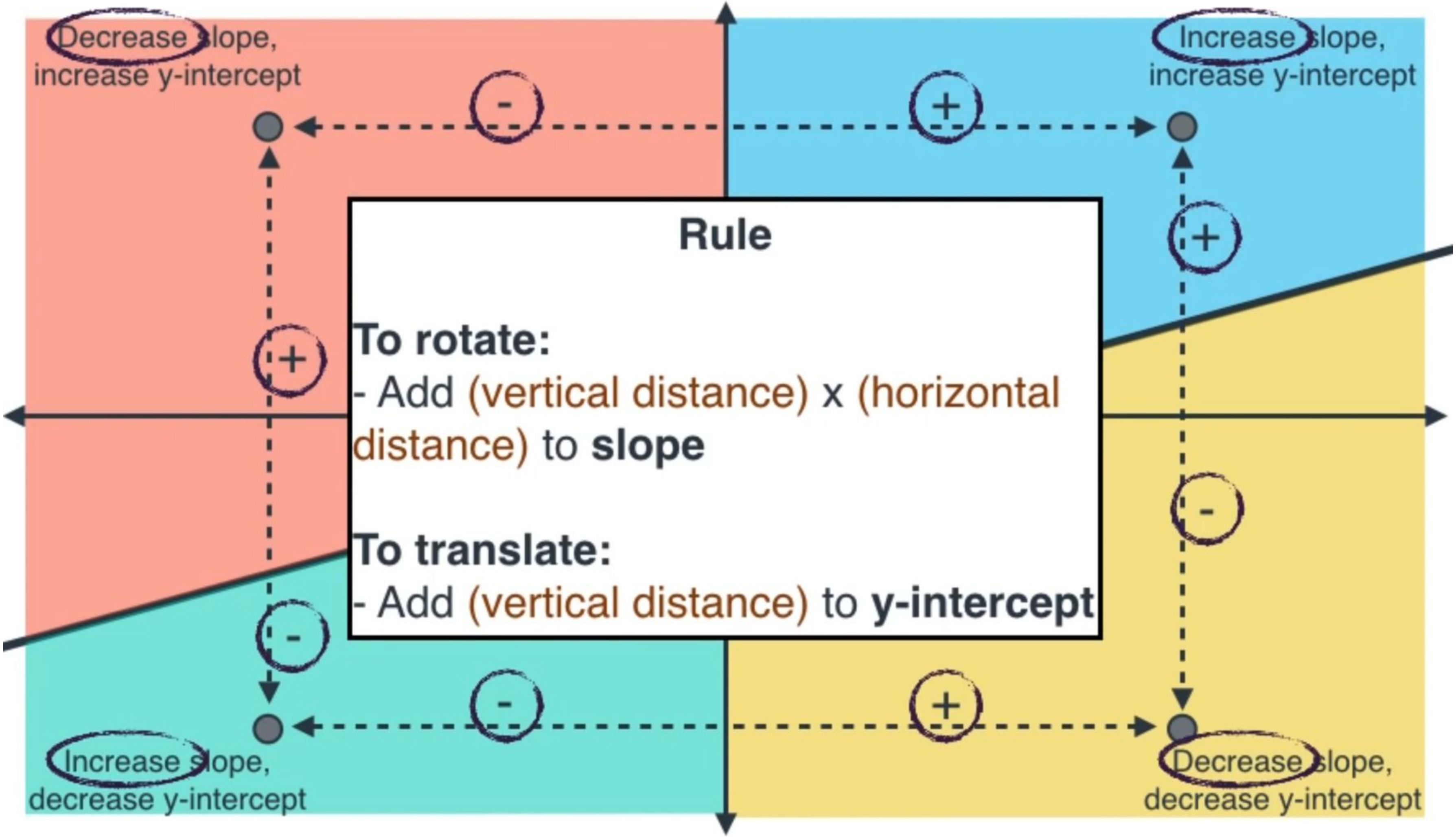


4 cases!

1 case?


Decrease slope,
increase y-intercept
Increase slope,
increase y-intercept
-
+
+
+
-
-
-
+
Increase slope,
decrease y-intercept
Decrease slope,
decrease y-intercept

One rule

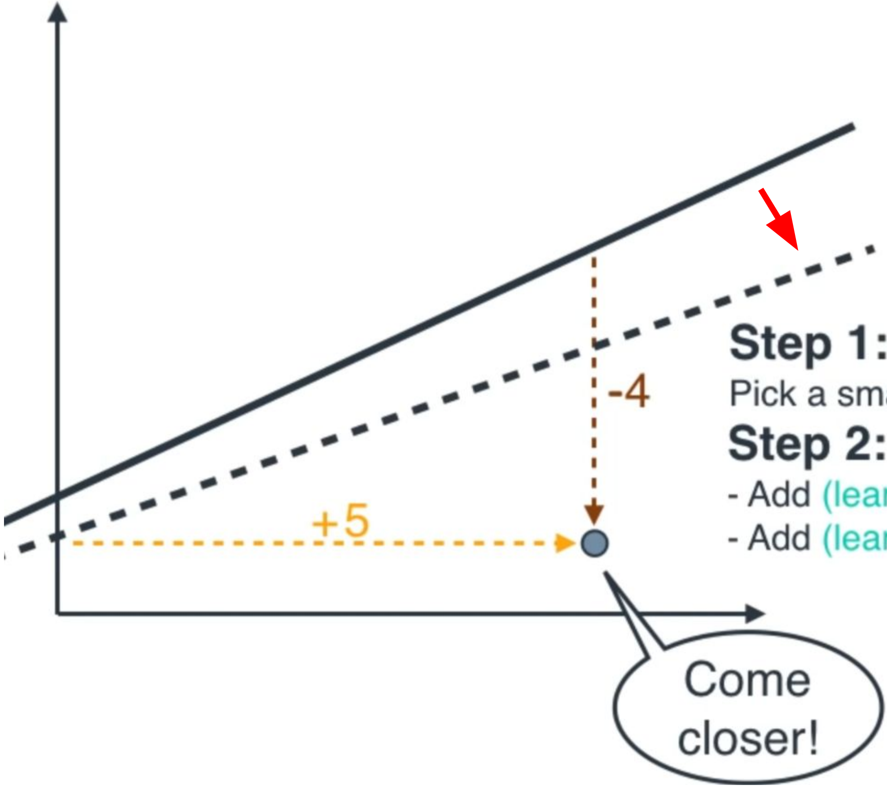

Decrease slope, increase y-intercept
Increase slope, increase y-intercept
-
+
+
+
-
-
-
+
Increase slope, decrease y-intercept
Decrease slope, decrease y-intercept

Rule

To rotate:
- Add (vertical distance) x (horizontal distance) to **slope**

To translate:
- Add (vertical distance) to **y-intercept**

LR Algorithm – Square Trick



$$\begin{tikzcd} y = 2x + 3 \arrow[d, "-0.2"', "-0.04"] \\ y = 1.8x + 2.96 \end{tikzcd}$$

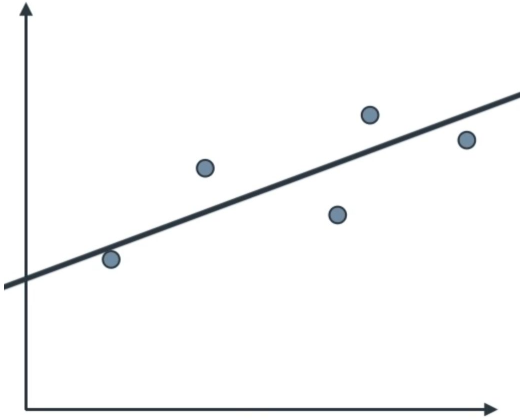
Step 1:

Pick a small number (learning rate) 0.01

Step 2:

- Add (learning rate) x (vertical distance) x (horizontal distance) to **slope**
- Add (learning rate) x (vertical distance) to **y-intercept**

LR Algorithm Improvement



Step 1: Start with a random line

Step 2: Pick a large number. 1000 (number of repetitions, or epochs)

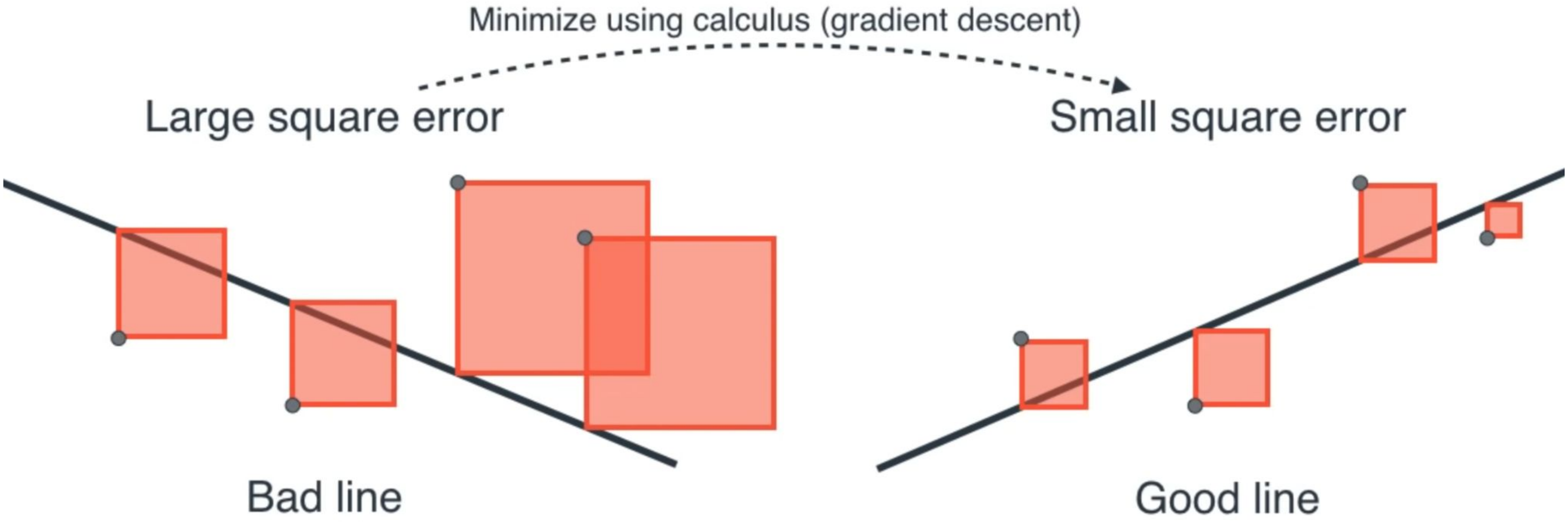
Step 3: Pick a small number. 0.01 (learning rate)

Step 4: (repeat 1000 times)

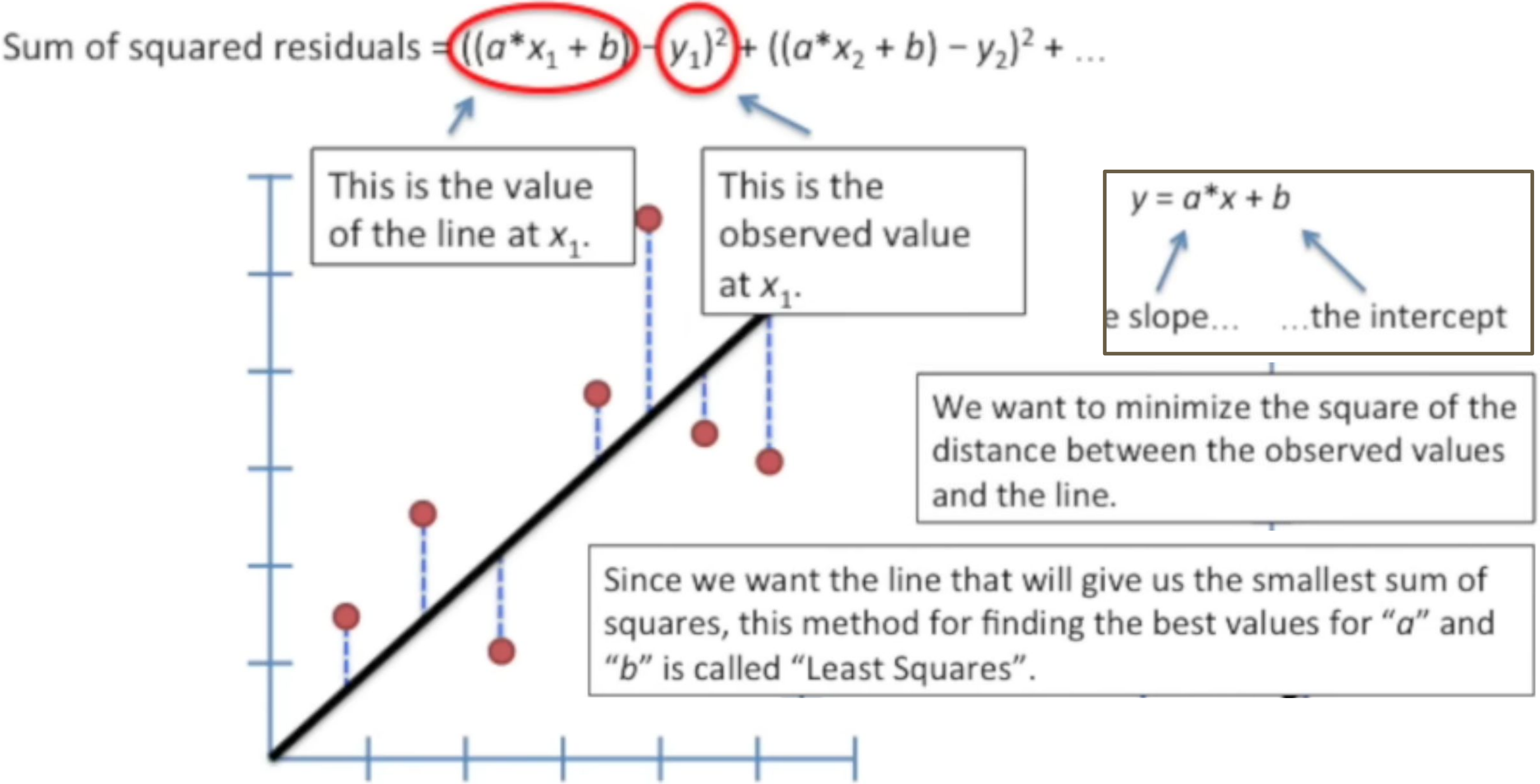
-Pick random point

- Add (learning rate)x(vertical distance) x (horizontal distance) to **slope**
- Add (learning rate)x(vertical distance) to **y-intercept**

LR Algorithm-2 – GD over Squared Error



LR Algorithm-2 – GD over Squared Error – Step-1



Sum of squared residuals = $((a{*}x_1 + b) - y_1)^2 + ((a{*}x_2 + b) - y_2)^2 + \ldots$

This is the value of the line at x_1.

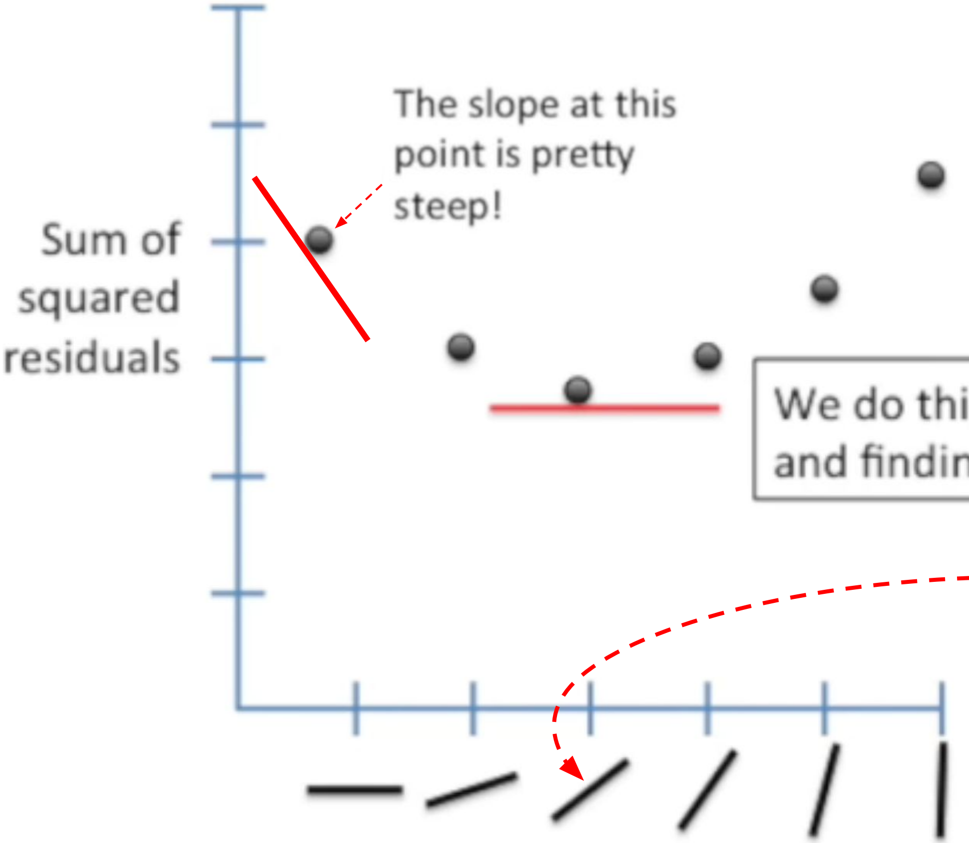
This is the observed value at x_1.

$y = a{*}x + b$

We want to minimize the square of the distance between the observed values and the line.

Since we want the line that will give us the smallest sum of squares, this method for finding the best values for "*a*" and "*b*" is called "Least Squares".

LR Algorithm-2 – GD over Squared Error – Step-2



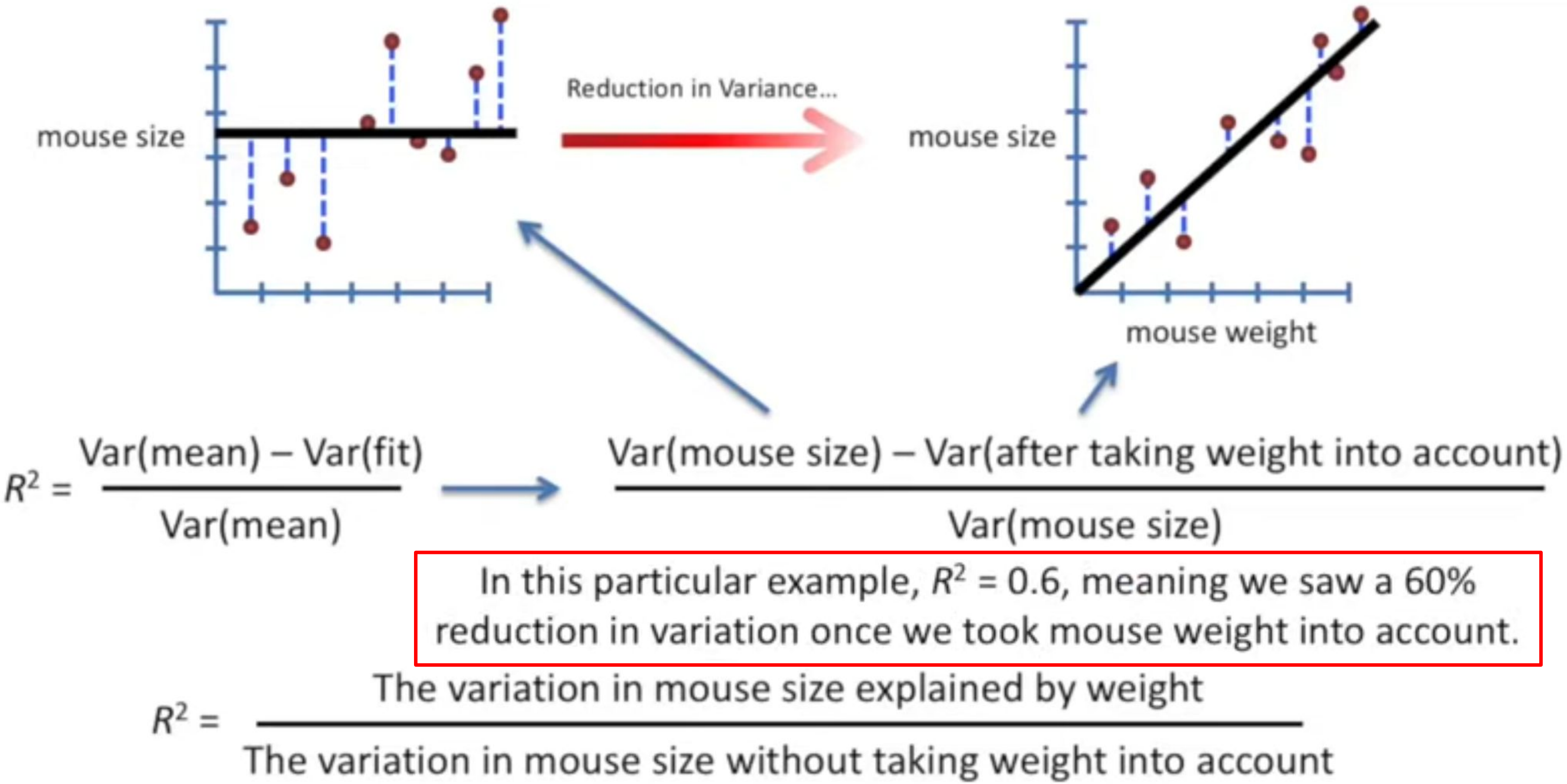
The slope at this point is pretty steep!

The derivative tells us the slope of the function at every point.

We do this by taking the derivative and finding where it is equal to = 0.

The final line minimizes the sums of squares (it gives the “least squares”) between it and the real data.

R-Sqare

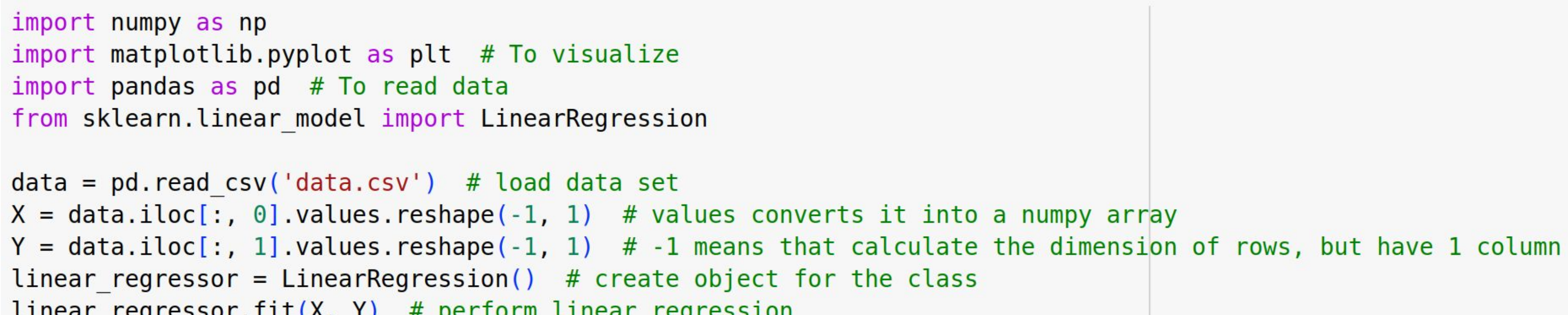


$$R^2 = \frac{\text{Var(mean)} - \text{Var(fit)}}{\text{Var(mean)}} \rightarrow \frac{\text{Var(mouse size)} - \text{Var(after taking weight into account)}}{\text{Var(mouse size)}}$$

In this particular example, R^2 = 0.6, meaning we saw a 60% reduction in variation once we took mouse weight into account.

$$R^2 = \frac{\text{The variation in mouse size explained by weight}}{\text{The variation in mouse size without taking weight into account}}$$

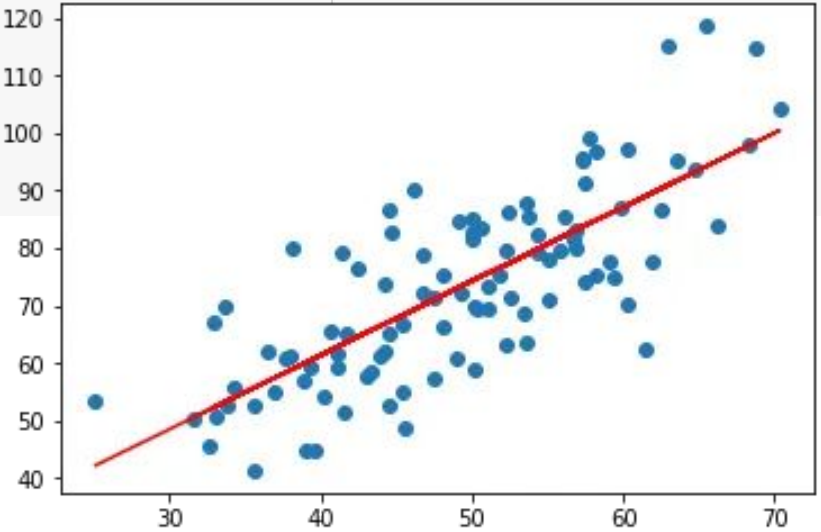
LR – Implementation



```
import numpy as np
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression

data = pd.read_csv('data.csv')  # load data set
X = data.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y = data.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of rows, but have 1 column
linear_regressor = LinearRegression()  # create object for the class
linear_regressor.fit(X, Y)  # perform linear regression
Y_pred = linear_regressor.predict(X)  # make predictions

plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```


120
110
100
90
80
70
60
50
40
30
40
50
60
70