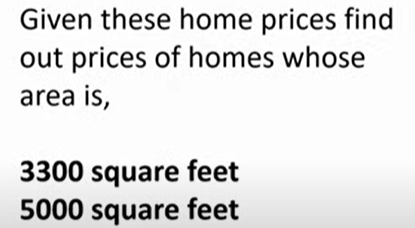
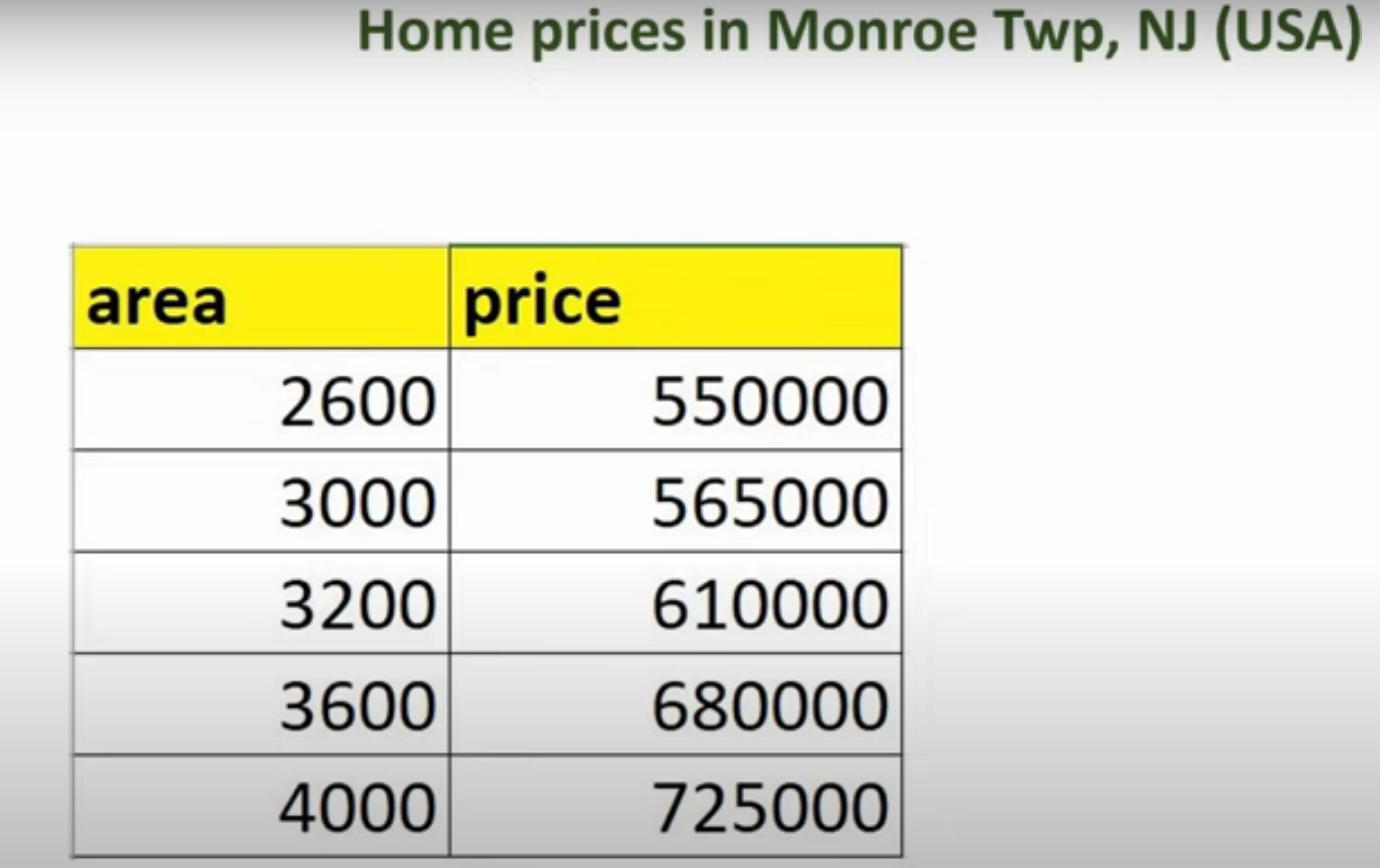
Predicting Home Prices Using Linear Regression

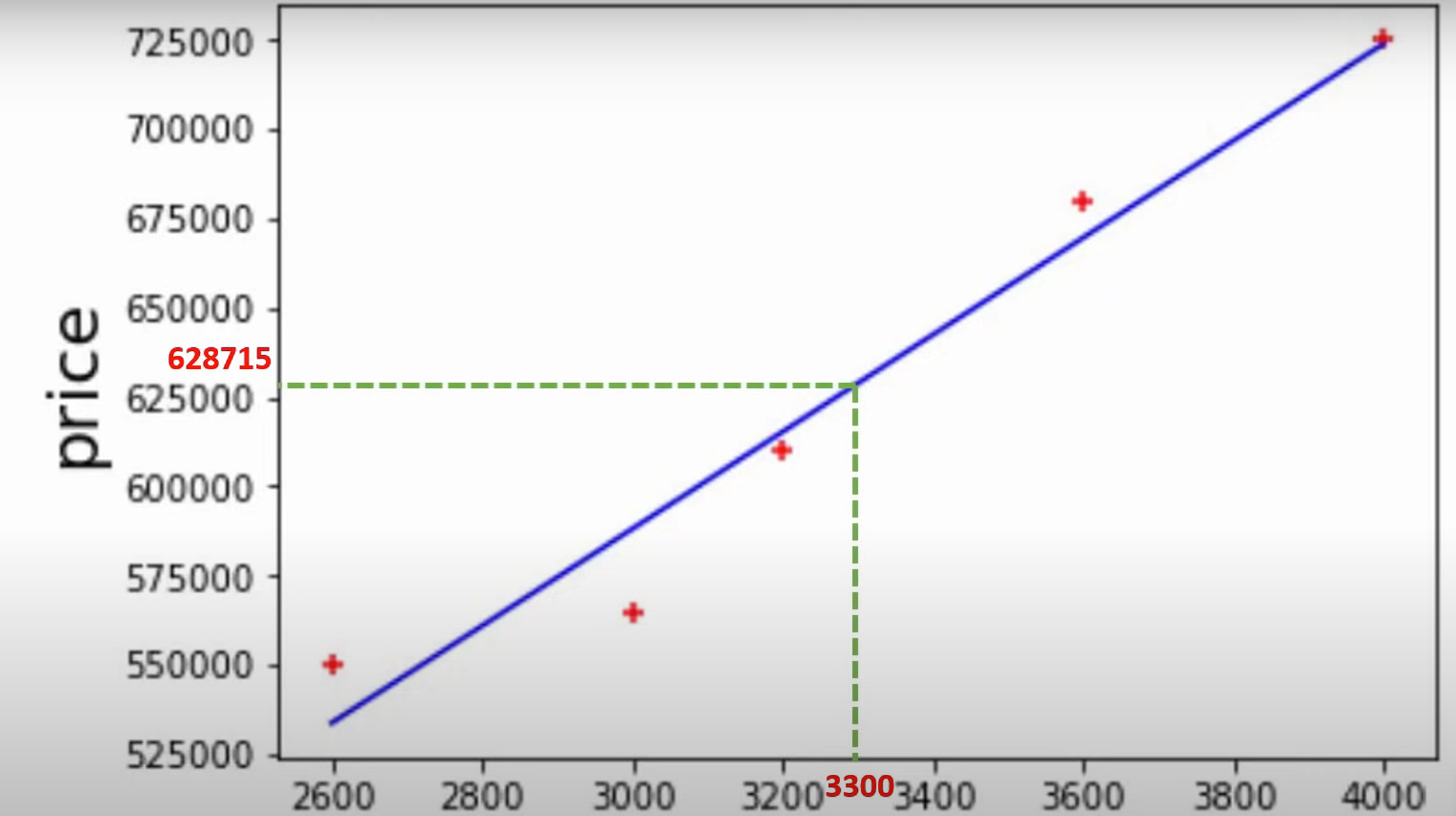
* collected data on home prices based on the areas in Monroe Township, NJ.
* With this data, construct a machine learning model called linear regression
* LR will predicting homes prices with areas of 3,300 and 5,000 square feet.





**Data Visualization**

* Visualize the available prices and areas with a scatter plot, where the red marker represents the data points.
* We'll then draw a blue line that best fits these data points.



**Purpose of the Blue Line**

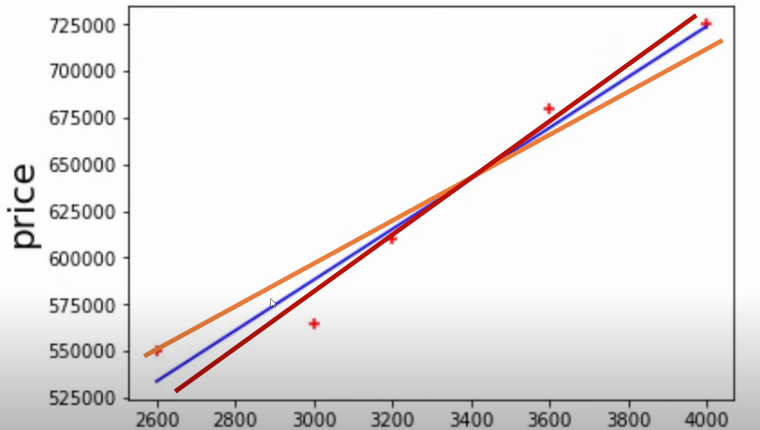
 Function: The blue line (represents a linear equation) estimate home prices for given areas.

 **Example**:

* Predict price for 3,300 sq. ft. Home based on linear equation (Blue Line)
* Predict price for for 5,000 sq. ft. Home based on linear equation (Blue Line)

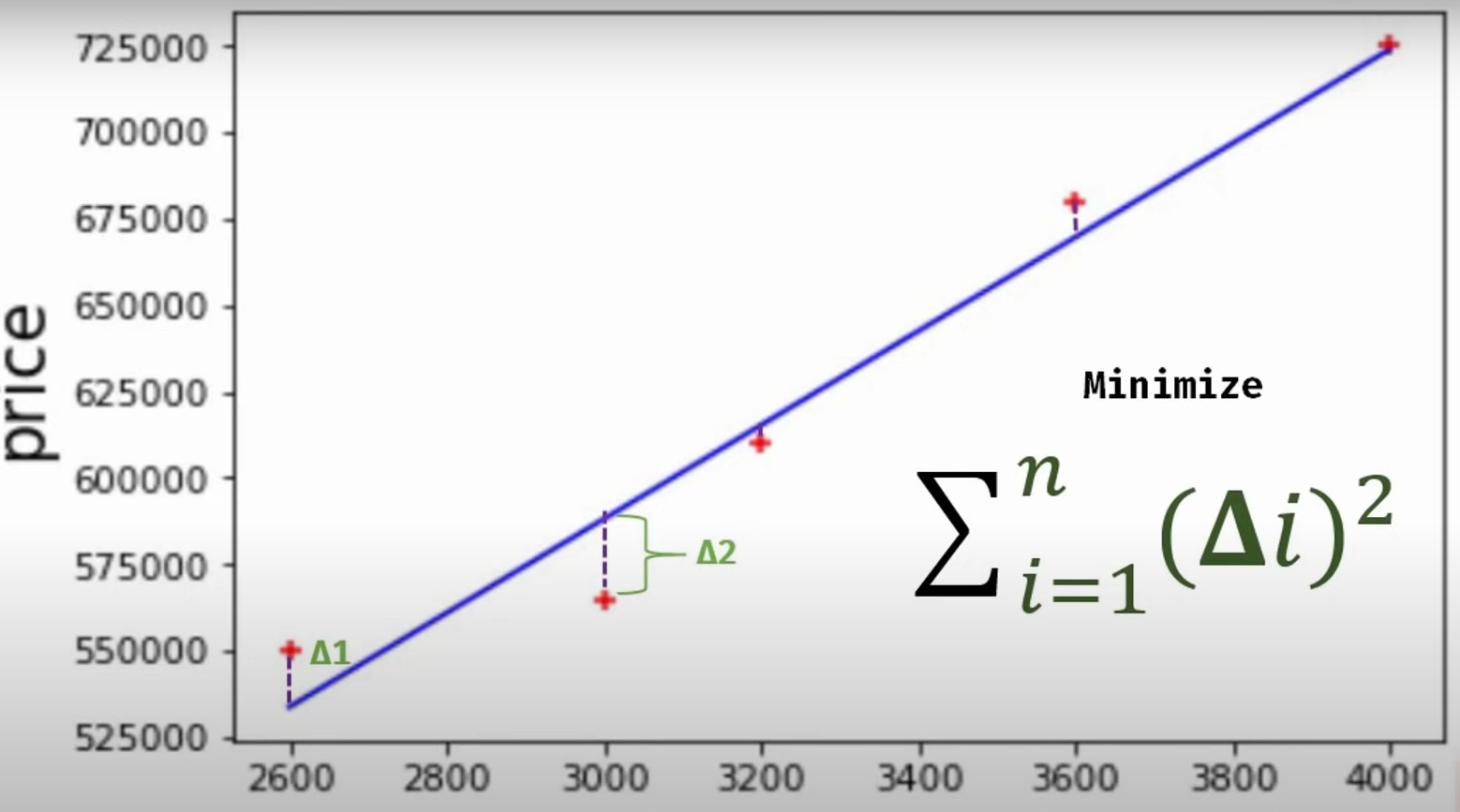
**Choosing the Best Fit Line**

* **Problem**: Many lines (e.g., red, orange) can be drawn in number of ways
* **Solution**:
  + Calculate errors (ΔDeltaΔ) between actual data points and points predicted by linear equation.
  + square individual errors, sum them up and then then try to minimize
  + We do this procedure for each line (red, orange, blue)

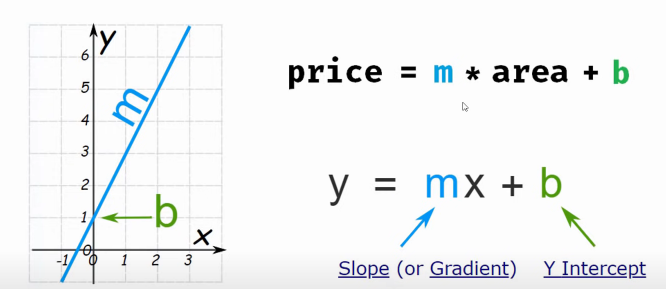


**Minimizing Error**

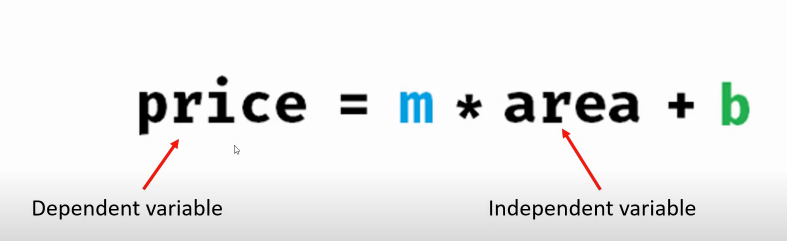
* **Approach**:
  + Minimize the total squared error.
  + Blue line is selected because found that it minimizes errors effectively ( better than others.)

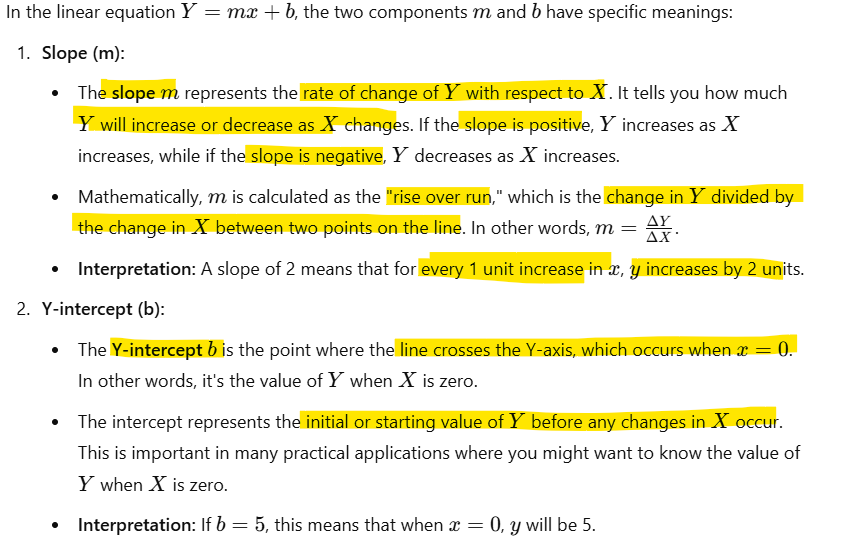


Area



Linear Equation looks like Y=mx +b



* We calculate price based on area, y=price, x= area.
* 
* Linear regression helps in estimating continuous values like home prices.

The best fit line minimizes prediction errors using mathematical optimization.

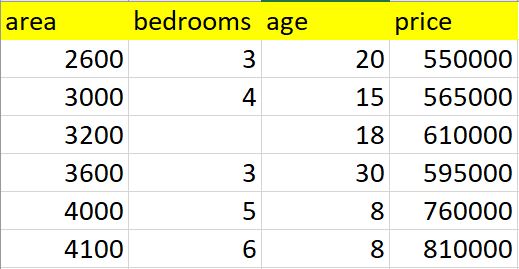
**Linear Regression with Multiple Variables: Predicting Home Prices**

* We will explore **linear regression with multiple variables**, also known as **multivariate regression**.
* Using this method, we aim to predict **home prices** in Monroe Township, New Jersey.

**Key Factors in Home Pricing**

The table provided includes various metrics such as:

* **Area (square footage)**
* **Number of bedrooms**
* **Age of the home**



These factors influence the ultimate price of a home.

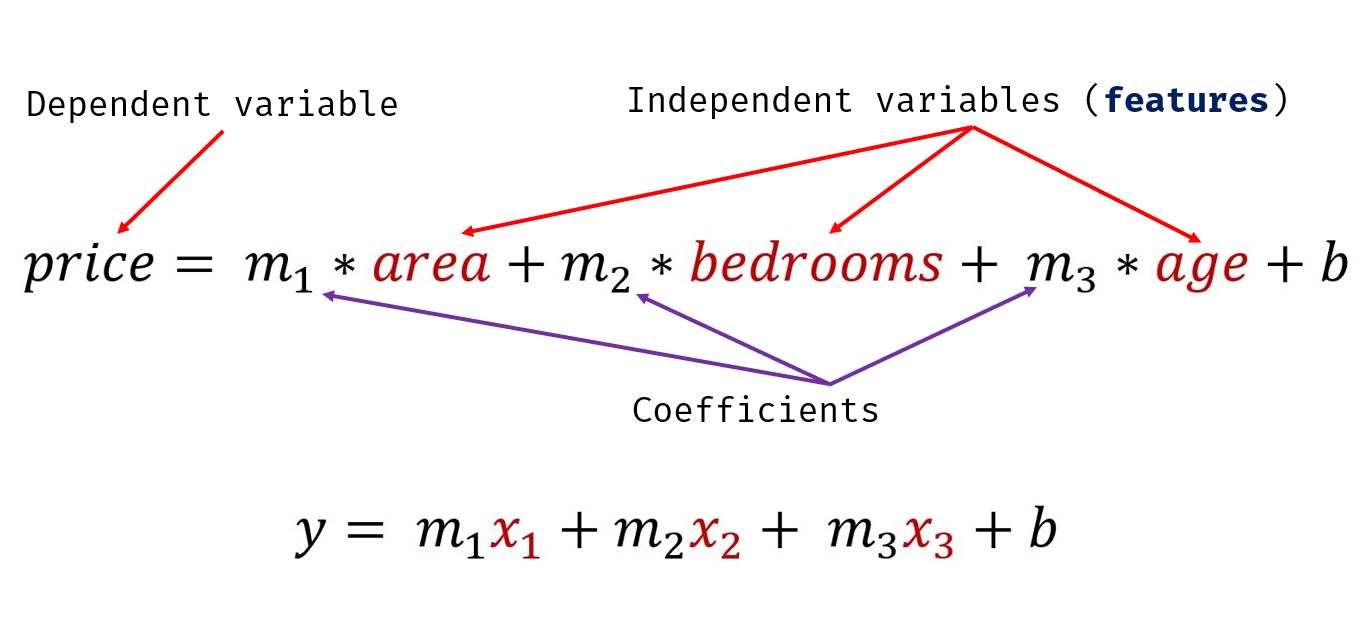
* Previously, we examined **simple linear regression**, where the price depended only on the area.
* Now, we make the problem more complex by incorporating additional variables like **bedrooms** and **age**, reflecting real-world scenarios where home prices depend on multiple factors, not just square footage.
* After building our model, we will use it to predict the prices of two homes.

**Data Analysis: A Crucial First Step**

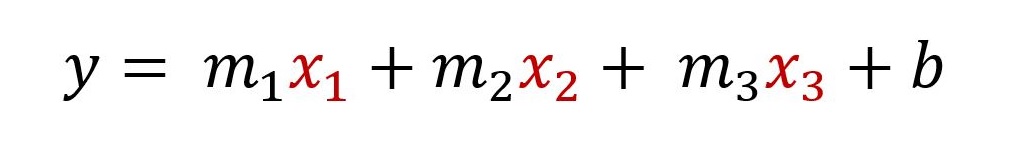
Before tackling any machine learning problem, it is essential to carefully analyze the dataset. Upon examining the data, here are the observations:

1. **Missing Data**: There is a data point missing in the table, which must be handled appropriately to ensure the model's accuracy.
2. **Linear Relationships**: A clear linear relationship exists between the independent variables (area, bedrooms, and age) and the target variable (price).
   * For instance, as a home ages, its price tends to decrease. Consider these examples:
     + A **3,200-square-foot home** with **18 years of age** is priced above **$600,000**.
     + A slightly larger **3,600-square-foot home**, due to its greater age, is priced lower than the smaller home.
   * Similarly, as the **area** and **number of bedrooms** increase, the price generally rises.

Based on the analysis, it is reasonable to apply **linear regression** to this dataset. Our linear equation will look like this where price is dependent on three features; area, bedrooms and age.



The above equation can be generalized into the following equation where u can have ‘n’ number of independent variables or features. In our case, we have 3 features



Check python code for the remaining stuff

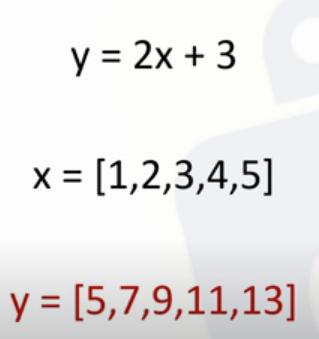
**Gradient Descent and Cost Function**

**Understanding Linear Algebra in Machine Learning**

Let’s revisit a concept from our school days: a simple equation where we calculate the value of **y** based on an input **x**. For example: we drive the formula manually

If **x = 2**, we calculate **y** as:

y=3x+3



Here, multiplying **3** with **2** gives **6**, and adding **3** results in **9**. This equation provides a way to predict the value of **y** given **x**.

**What is a Prediction Function?**

* In **machine learning**, the process is somewhat similar but more data-driven.
* Instead of starting with a predefined equation, we derive it from a given **training dataset**. This dataset consists of **observations**, including inputs and corresponding outputs.
* The goal is to learn a mathematical formula, also known as the **prediction function**, that can model the relationship between inputs and outputs.
* Once derived, this function can predict future values based on new inputs.

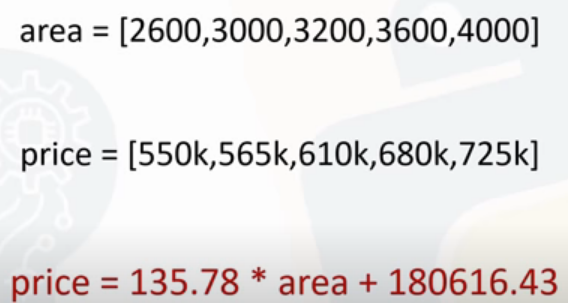
**Example: Predicting Home Prices**

Consider the problem of predicting home prices. In earlier linear regression topic, we worked with data on **area** (input) and **price** (output).

By analyzing this data, we derived an equation that models the relationship between these two variables. This equation becomes the **prediction function**, enabling us to estimate home prices based on area.

Y= mx + b

Price= 135.78 X + 18016

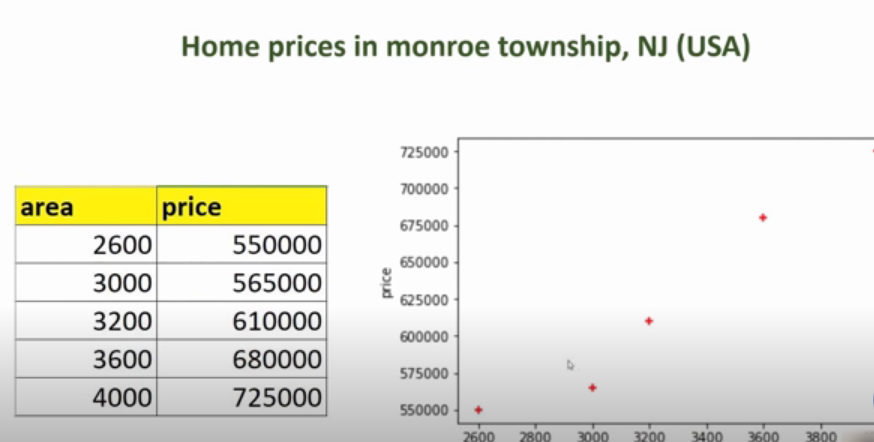


Prediction ffn to predict future values(unseen data)

By leveraging linear algebra, we transition from manually creating equations to learning them from data, a foundational concept in machine learning.

**Deriving the Best Fit Line for Home Prices**

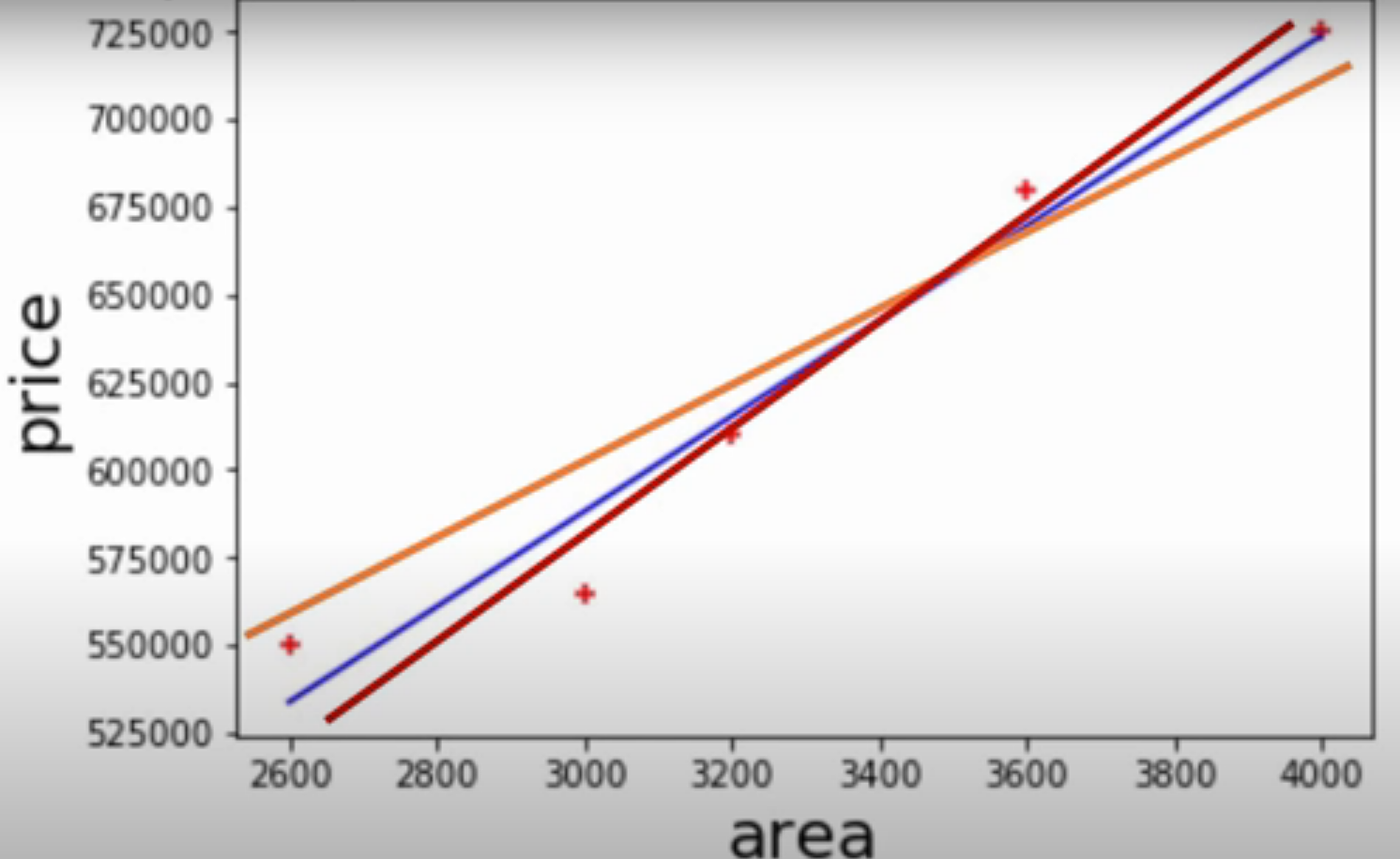
Here, we have home prices in Monroe Township, and we’ve plotted them on a chart.



Our goal is to derive an equation that best represents the relationship between the input (such as area) and the output (price). This equation will be the **best fit line**, shown as the **blue line** on the chart.

**The Challenge of Finding the Best Fit Line**

The data points are scattered, making it difficult to draw a perfect line through them. In fact, there could be many different lines that might fit these points. As the dataset becomes more complex or scattered, finding the right line becomes even more challenging. So, how do we determine which line is the best fit?



**Error Calculation and Mean Squared Error**

One approach is to draw a random line and calculate the error between each data point and the predicted value on that line. This error is called **delta** (the difference between the actual and predicted values).

Here’s the process:

1. **Calculate the error** (delta) for each data point:

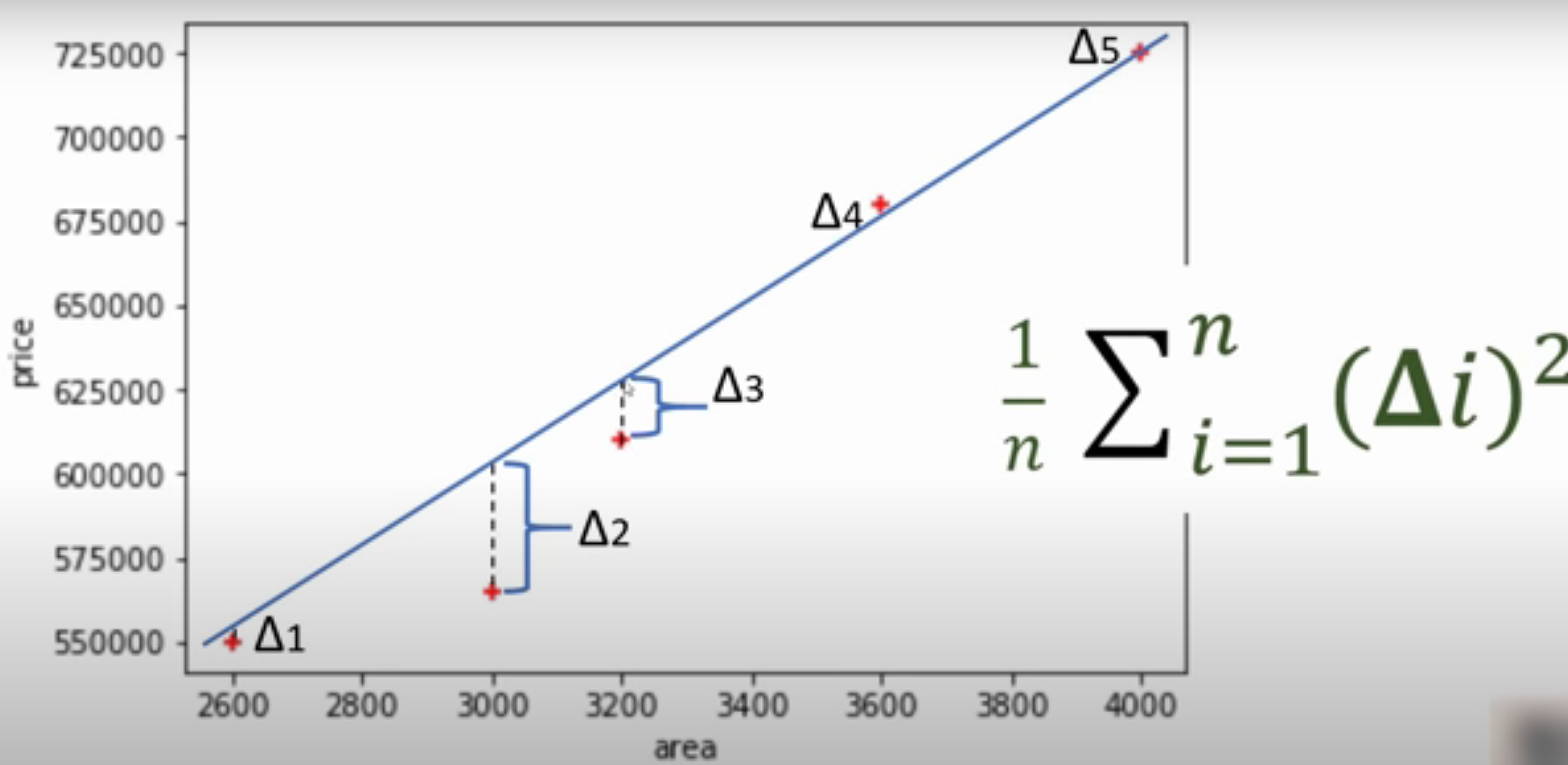


1. **Square the errors** to avoid negative values skewing the results:

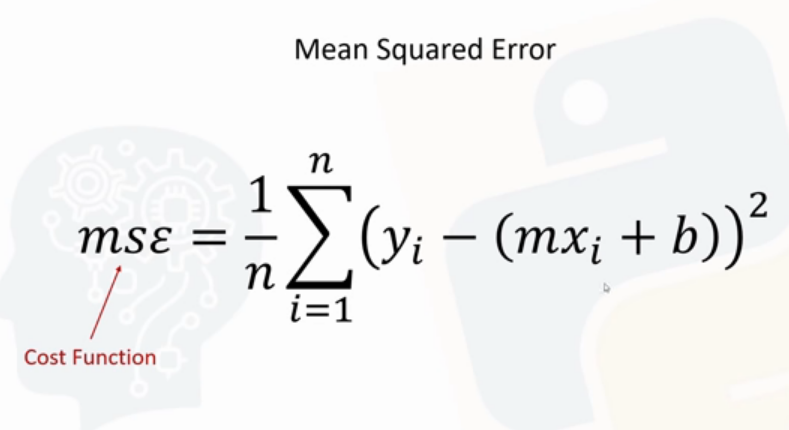
****

1. **Sum all squared errors** and divide by the number of data points, **n** (for example, **n = 5**):

****



* The **Mean Squared Error (MSE)** represents how well your line fits the data. The lower the MSE, the better the fit.
* This MSE is often referred to as the **cost function**. While there are different cost functions, **mean squared error** is the most commonly used.



In the equation for MSE, the **predicted value (y)** is represented by the formula:



Where **m** is the slope, and **b** is the intercept.

**Efficiently Finding the Best Fit Line: Gradient Descent**

Instead of testing every possible combination of **m** and **b** (which would be inefficient), we can use an efficient algorithm to find the optimal values. This is where **gradient descent** comes in.

Gradient descent allows us to iteratively adjust **m** and **b** to minimize the cost function (MSE) in just a few iterations, making it much more efficient than trying every possibility.

By using Gradient Descent, we can quickly and effectively find the best fit line for our training dataset.

**Understanding the Gradient Descent Algorithm**

* The **Gradient Descent** algorithm is an efficient way to find the **best fit line** in a minimal number of iterations.
* It helps optimize the parameters (slope **m** and intercept **b**) of the linear model to minimize the **cost function** (mean squared error).

**Visualizing Gradient Descent**

To understand how gradient descent works, we plot the values of **m** and **b** against the **cost function** (mean squared error). Here’s how it looks:

* Each combination of **m** and **b** corresponds to a **cost value**.
* If we plot these combinations, we get a 3D surface (Plane) that resembles a **bowl** shape, where the lowest point represents the minimum cost (optimal values of **m** and **b**).

**Step-by-Step Process of Gradient Descent**

1. **Starting Point**: Begin at a random point, typically where **m = 0** and **b = 0**. At this starting point, the cost (error) might be high (e.g., 1000).
2. **Taking Small Steps**: Gradually reduce/adjust **m** and **b** by small amounts. These steps are aimed at lowering the cost/error. The size of each step is determined by a parameter called the **learning rate**. After each step, you recalculate the error, and it should decrease (e.g., from 1000 to 900).
3. **Repeat Until Convergence**: Keep adjusting **m** and **b** in small steps, recalculating the cost after each iteration. Eventually, you will reach the **minimum point** (the lowest point on the bowl), where the error is minimized.
4. **Optimal Values**: At the minimum, you have found the best values for **m** and **b** that minimize the error. These values will be used in your **prediction function** to make accurate predictions.

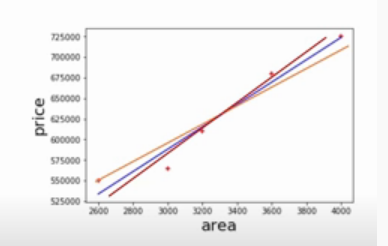
### 

### Visualizing Multiple Lines

Each line you plot corresponds to a different combination of **m** and **b**. For example:

* The **orange line** might correspond to **m1** and **b1**.(refers to a point somewhere in the above plane)
* The **blue line** might correspond to **m2** and **b2** which is the red dot in the above surface.
* The **red line** might correspond to **m3** and **b3**.

Each of these lines will have a corresponding point on the plot, showing different values of **m** and **b** and their associated costs. There are potentially many different lines, but gradient descent helps you find the one with the lowest cost efficiently.



**Understanding the Gradient Descent Step-by-Step**

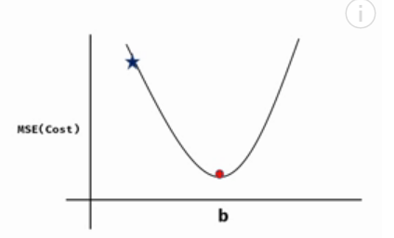
While it sounds simple to take "baby steps" towards finding the best fit line, implementing it mathematically requires a more concrete approach. Let's break down how this actually works.

**Visualizing the Process**

We can visualize the gradient descent process using 3D charts. If you view the first chart from one angle, it will show a graph of **b** (intercept) against the **cost** (error).

If you view it from another angle, you’ll see a different perspective (graph of **m** (slope) against the **cost** (error)).

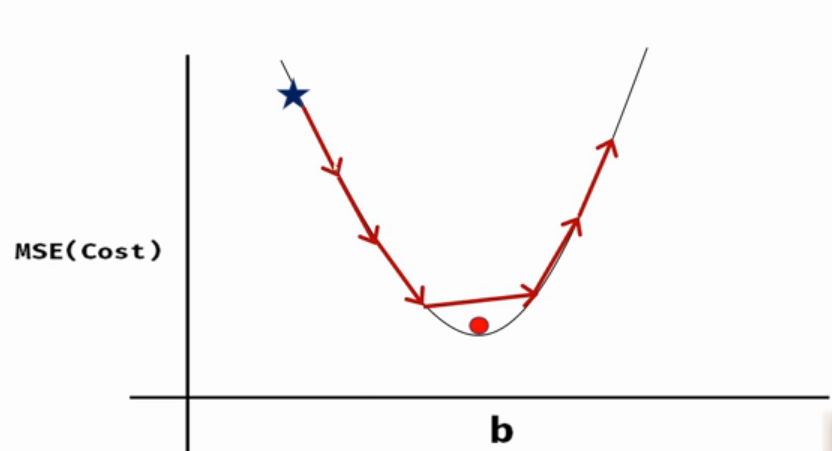
In both cases, the goal is to start from an initial point (represented as a star) and take steps towards the minimum (represented by a red dot).



**The Challenge of Taking Steps**

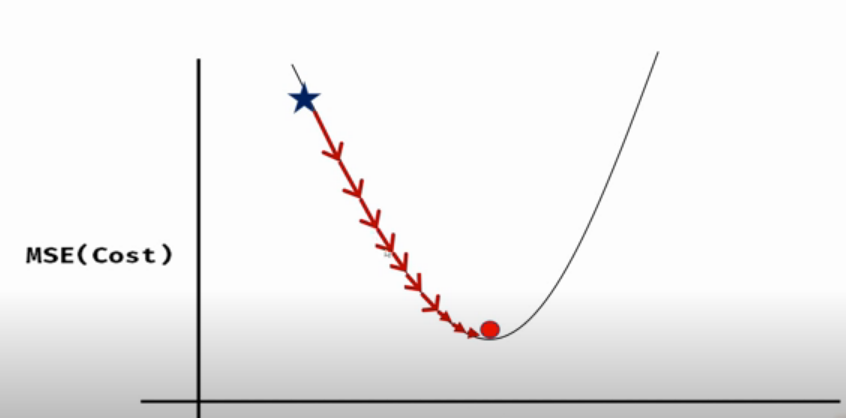
Now, how do you take these steps? One approach might be to take **fixed size steps** in the direction of the slope. However, this can be problematic because if the steps are too large, you might miss the global minimum, and the gradient descent could fail to converge.

Imagine starting at a point and taking large, fixed steps. You might overshoot the minimum, and from there, the algorithm could just keep moving away from the optimal point. Clearly, this approach isn’t effective.



**A Better Approach: Adaptive Step Sizes**

Instead of taking fixed-size steps, a better approach is to **adjust the step size** based on the slope of the curve. As you get closer to the minimum (red point), the step size should reduce. This way, you make large adjustments when you're far from the minimum, and smaller adjustments as you approach it. This technique allows you to gradually reach the minimum more precisely.

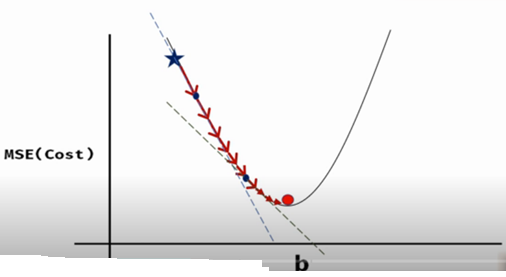


**How to Calculate the Steps**

To make this work, at each point, you need to calculate the **slope** of the curve. For instance, if you're at a point on the graph, the slope is the tangent to the curve at that point.

* If you're at a particular point (let's say the blue dot on the chart), the slope will tell you in which direction to move.
* The **learning rate** helps determine how large the step should be in that direction. The learning rate essentially scales the size of each step.

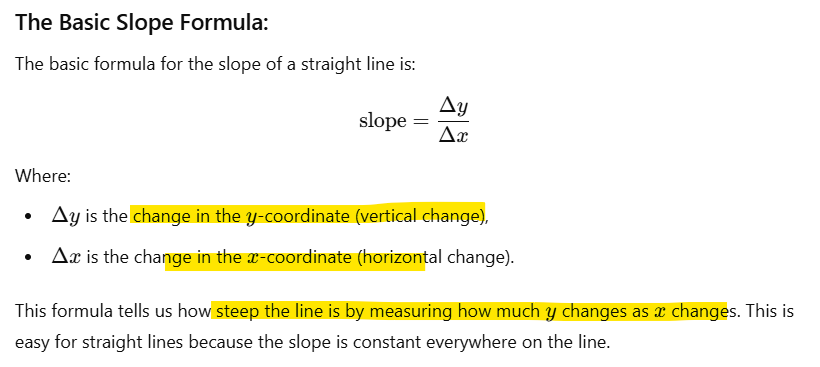
For example, if you're at this blue dot and you look at the green line, you'll know you need to move in this direction. Then, you can use something called a learning rate, in combination with the slope, to take a step and reach the next point.

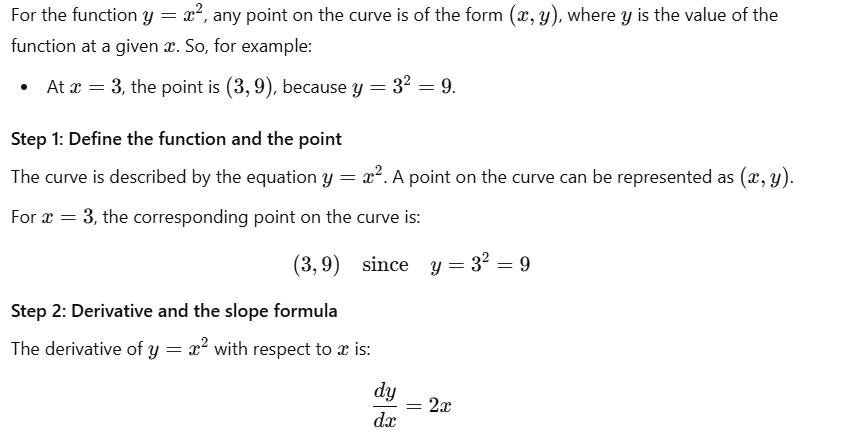


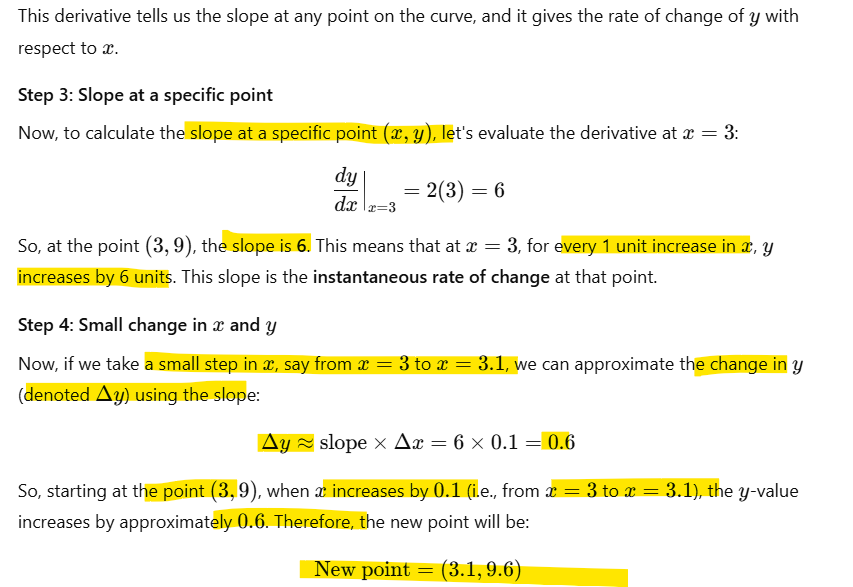
**The Role of Calculus**

At this point, we need to dive into **calculus** a bit. The slope of the curve is essentially the **derivative** of the **cost function** with respect to the parameter you're adjusting (in this case, **b**, the intercept). By calculating this derivative, we can determine the direction and magnitude of the step to take at each iteration.

<https://www.3blue1brown.com/lessons/derivatives-and-transforms>





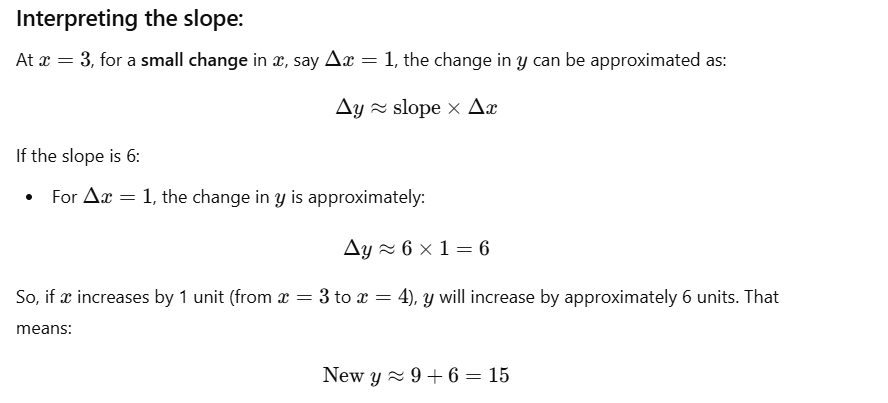


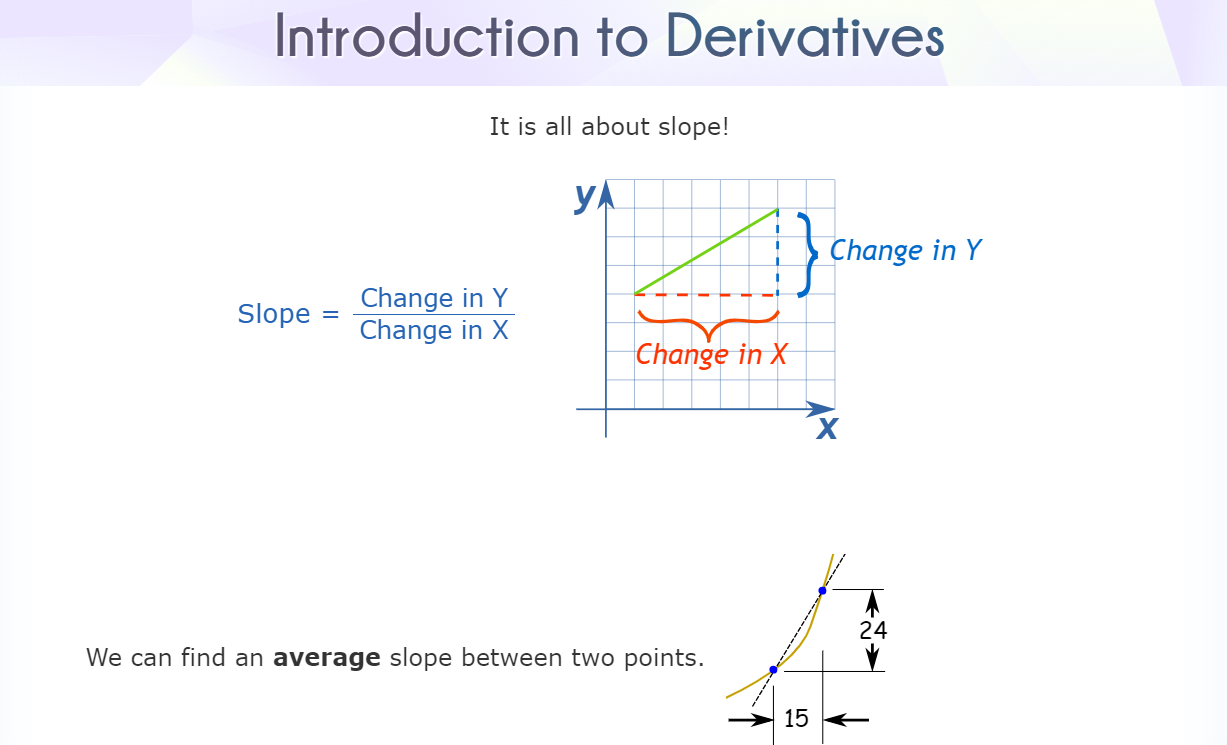
This shows how we use the slope (derivative) to estimate the change in y as x changes, even when the function is curved.

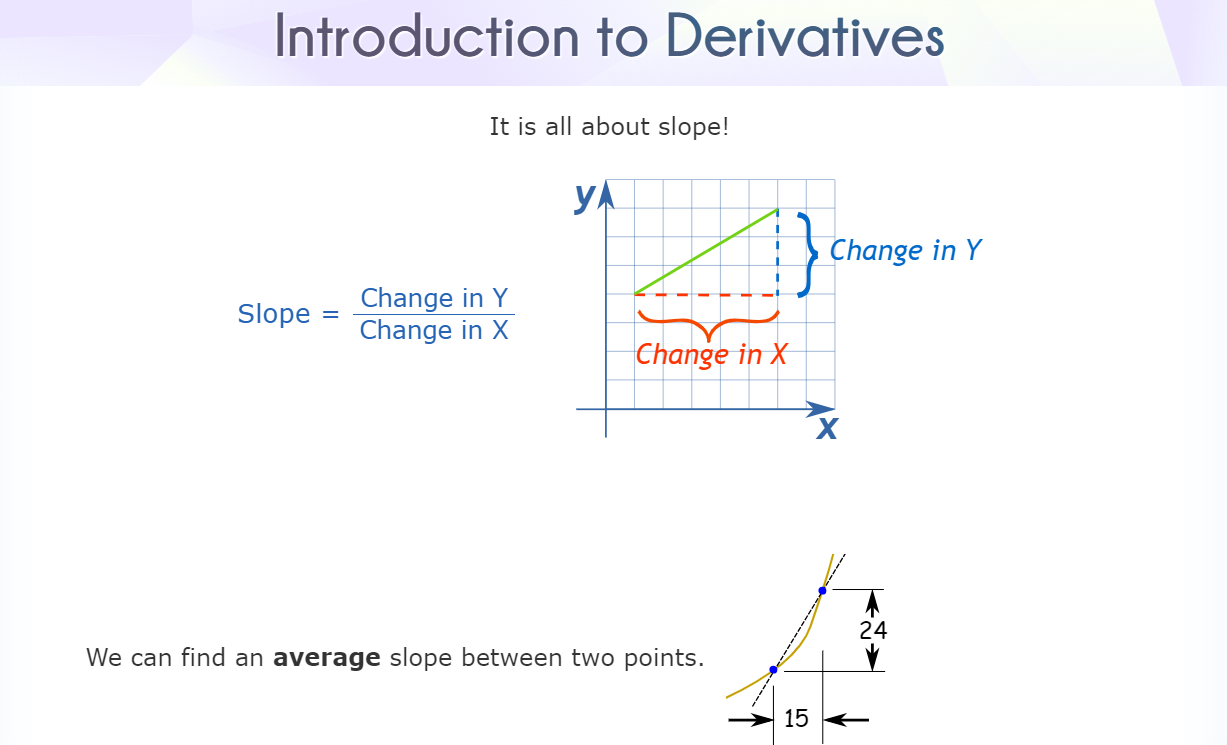
**What the slope represents:**

1. The **slope** of a curve at a point (given by the derivative) tells us the **rate of change of y with respect to x** at that point.
2. A slope of 6 means that **for every 1 unit increase in x, y increases by 6 units.**

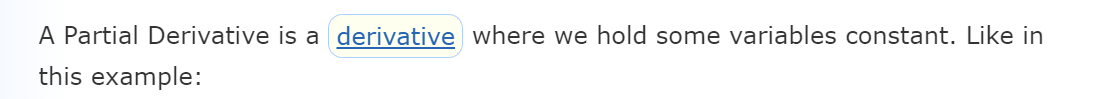
Task

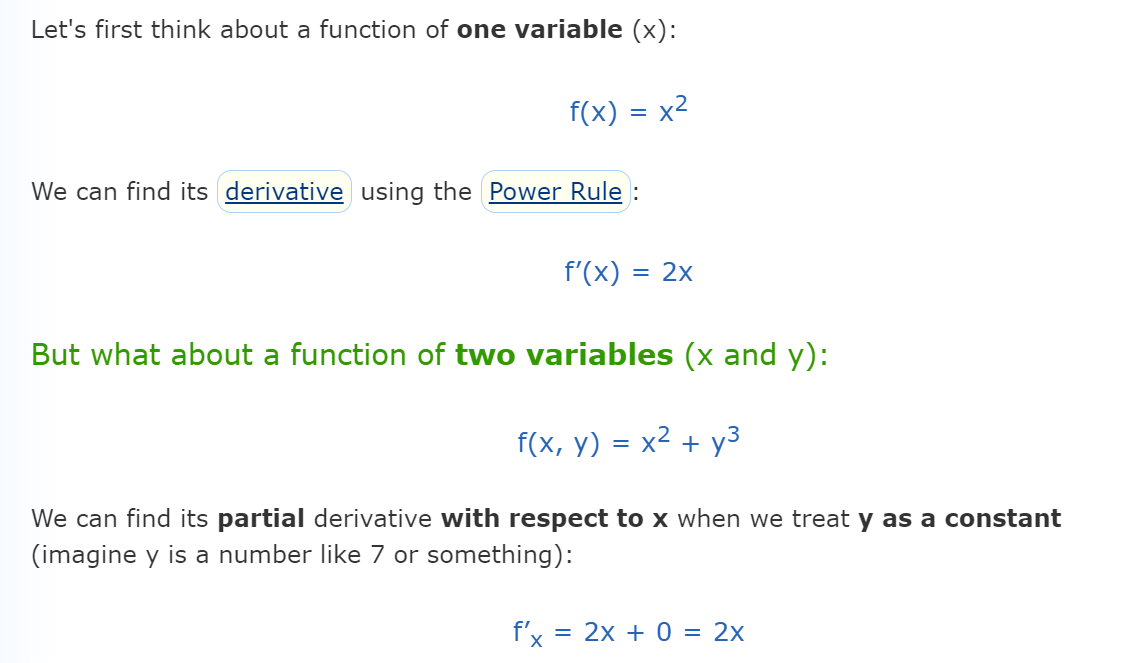
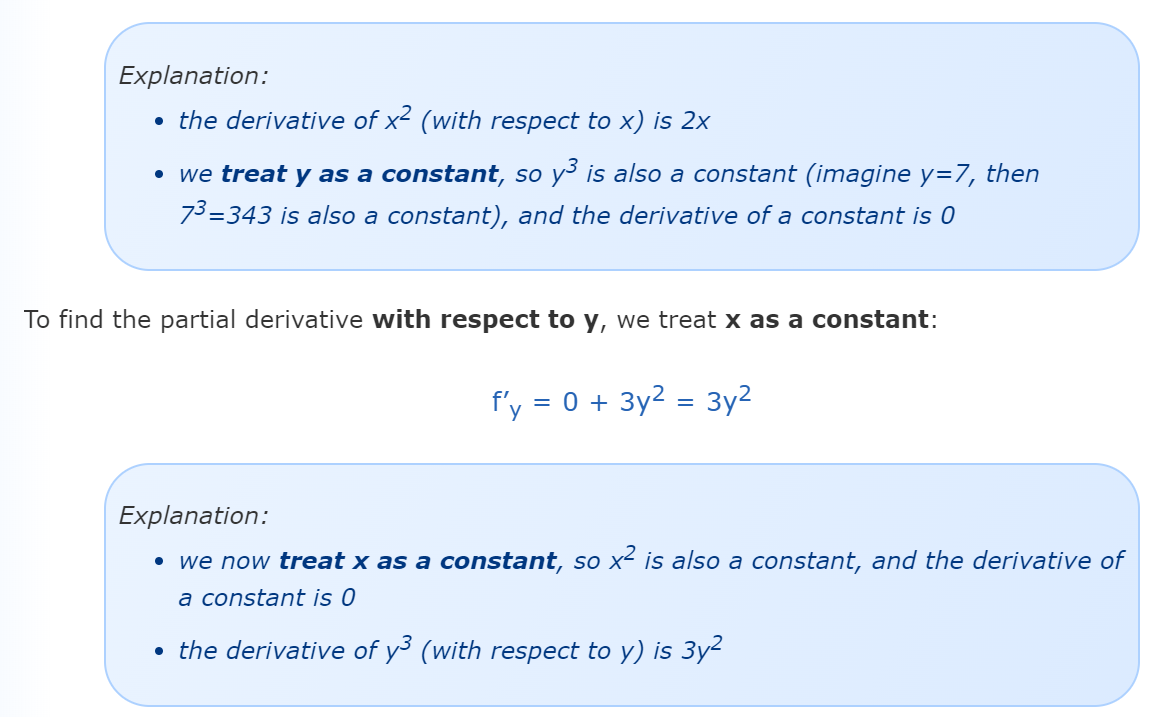


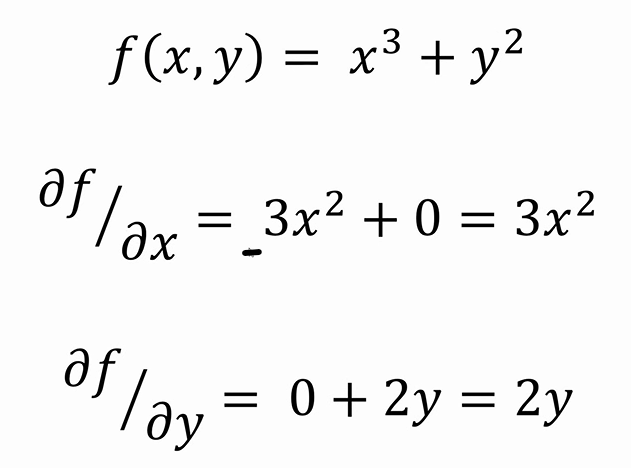




If function depends on two variables, 'x' and 'y,' you aim to calculate a partial derivative w.r.t 'x' by keeping 'y' fixed at zero.



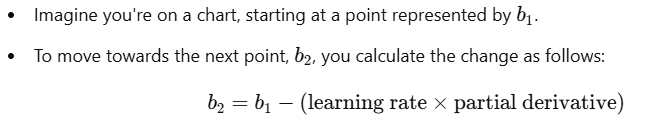
 

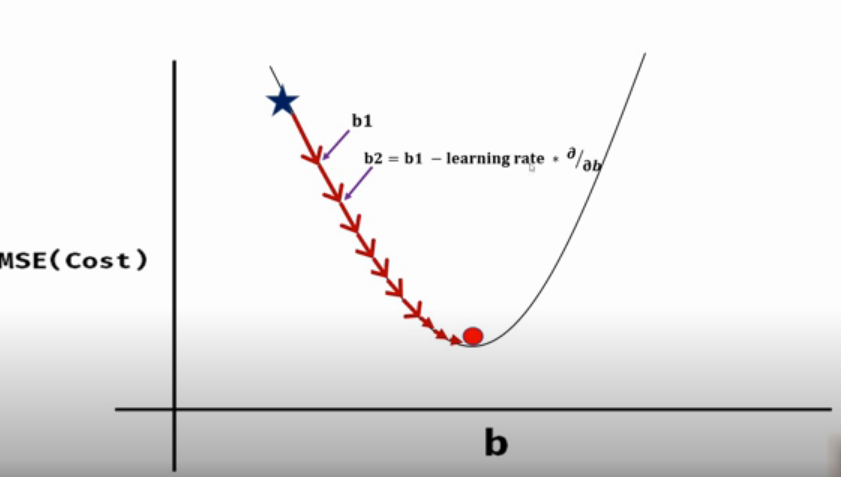
Once the partial derivative is calculated, it provides the slope, indicating the direction to move in. To move in that direction, we use a concept called the **learning rate**.

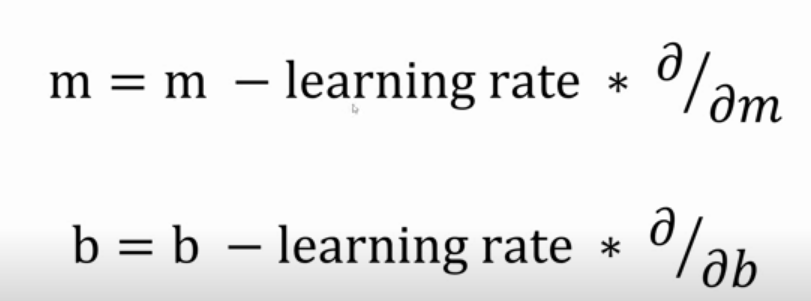
Starting with an initial value for m, the next step involves updating ‘m’ by subtracting the product of the learning rate and the partial derivative (slope).

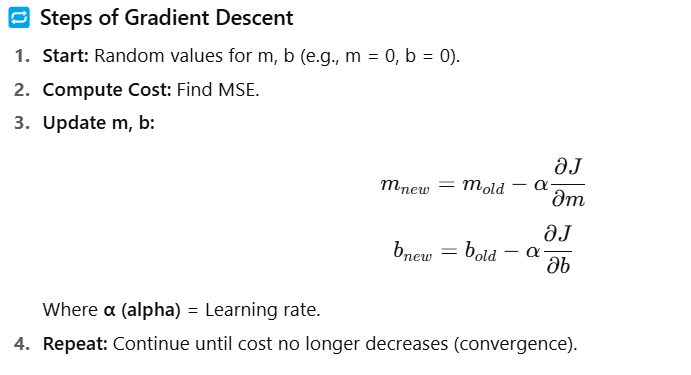
For example:



This process iteratively adjusts m (or b) to minimize the cost function, effectively guiding the model toward the optimal values.



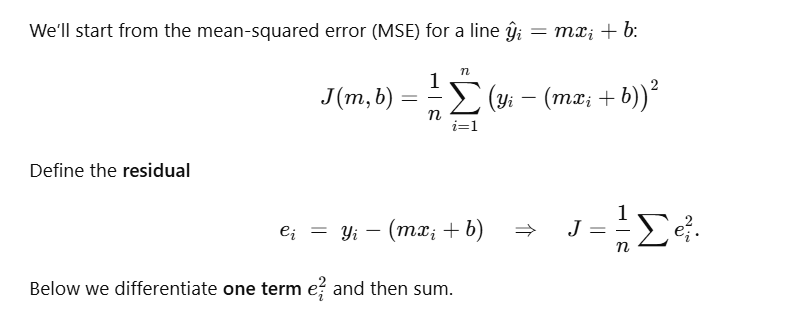


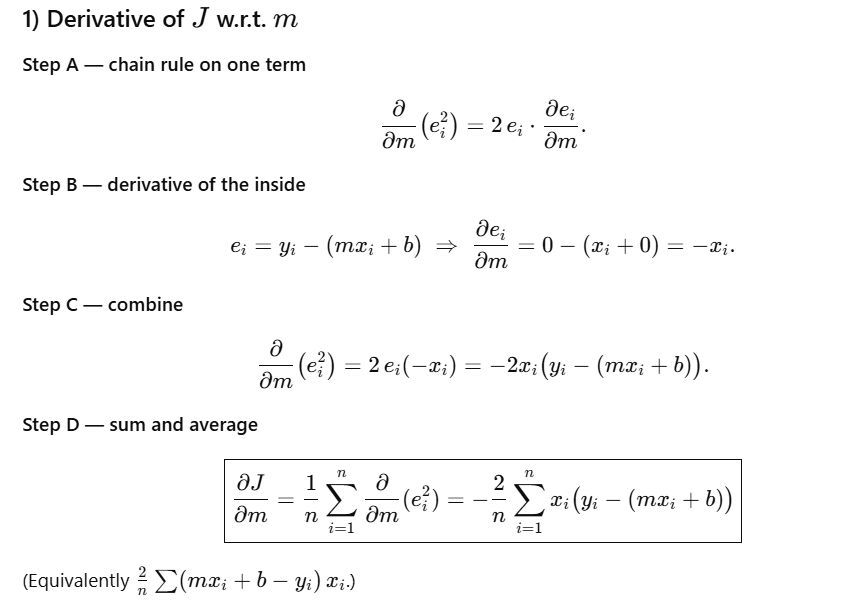


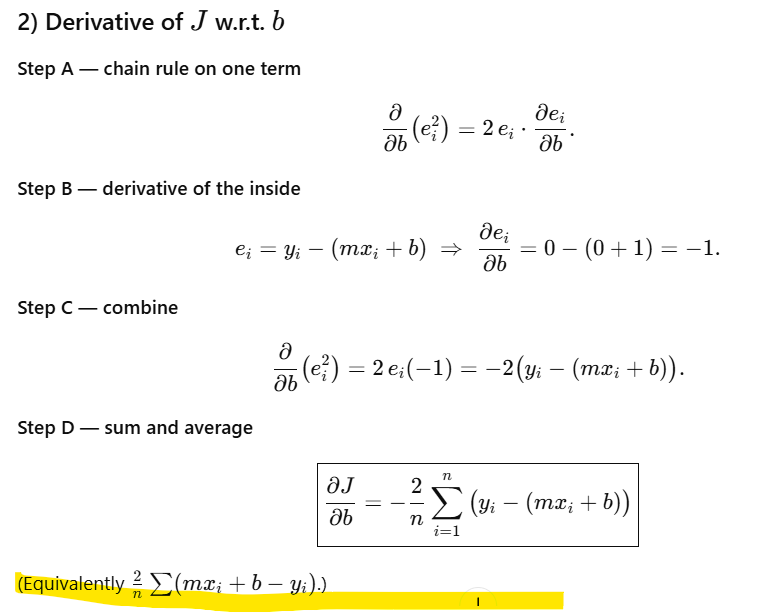
**🔹 Intuitive Analogy**

Think of learning rate as **walking down a hill toward the lowest point (global minimum)**:

* A **small α** = tiny steps → safe but **very slow convergence**
* A **large α** = big jumps → may **overshoot** the minimum or **diverge**
* A **balanced α** = smooth descent → **fast and stable learning**







|  |  |  |
| --- | --- | --- |
| Iteration | Learning rate | Cost |
| 10 | 0.001 | Reducing |
| 10 | 0.01 bigger step | reducing |
| 10 | 0.1 | Increasing, crossing global minima and shooting in another direction |
| 10 | 0.09 | Increasing |
| 10 | 0.08 | Reducing cost |
| 10,000 | 0.08 take this value |  |