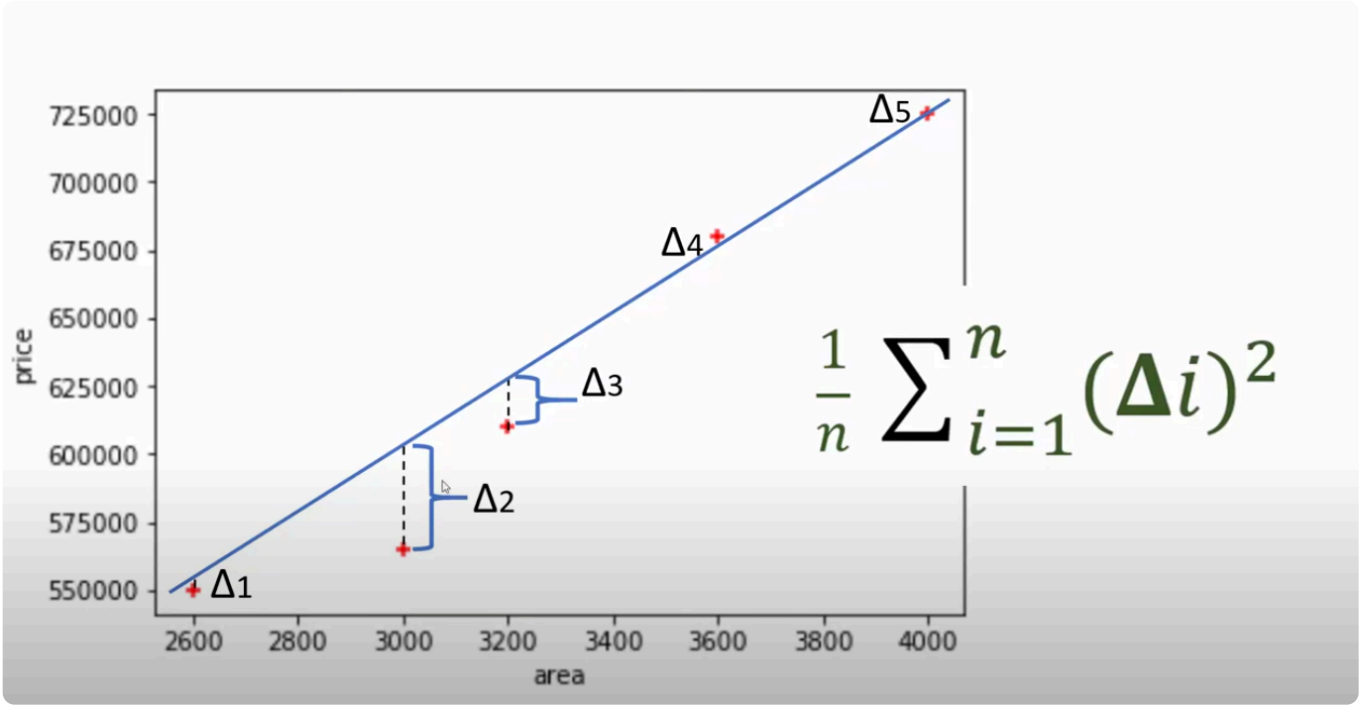
Gradient Descent and Cost Function



$$\frac{1}{n}\sum_{i=1}^{n}(\Delta i)^2$$

$$\sum$$

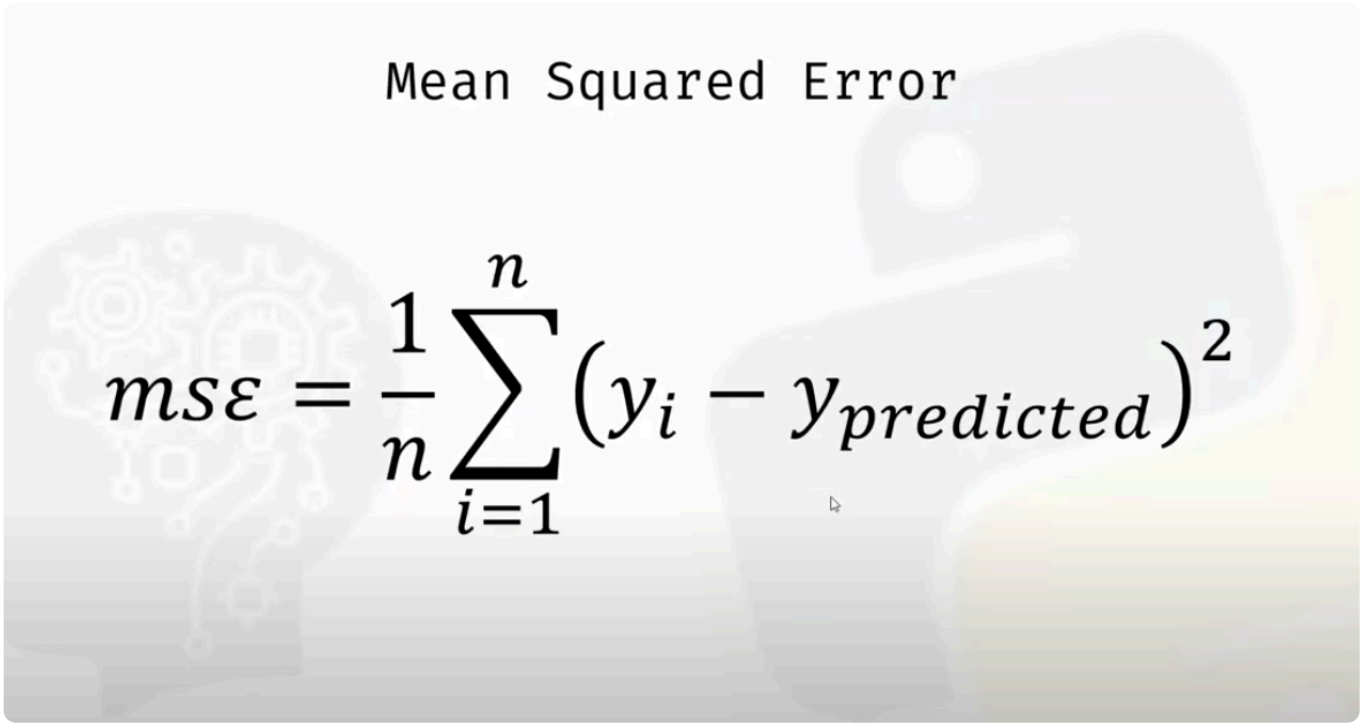
Where

Is the summation(Sum) of all number(i=1) to n

$$(\Delta i)^2$$

The square is used to avoid negatives in graph

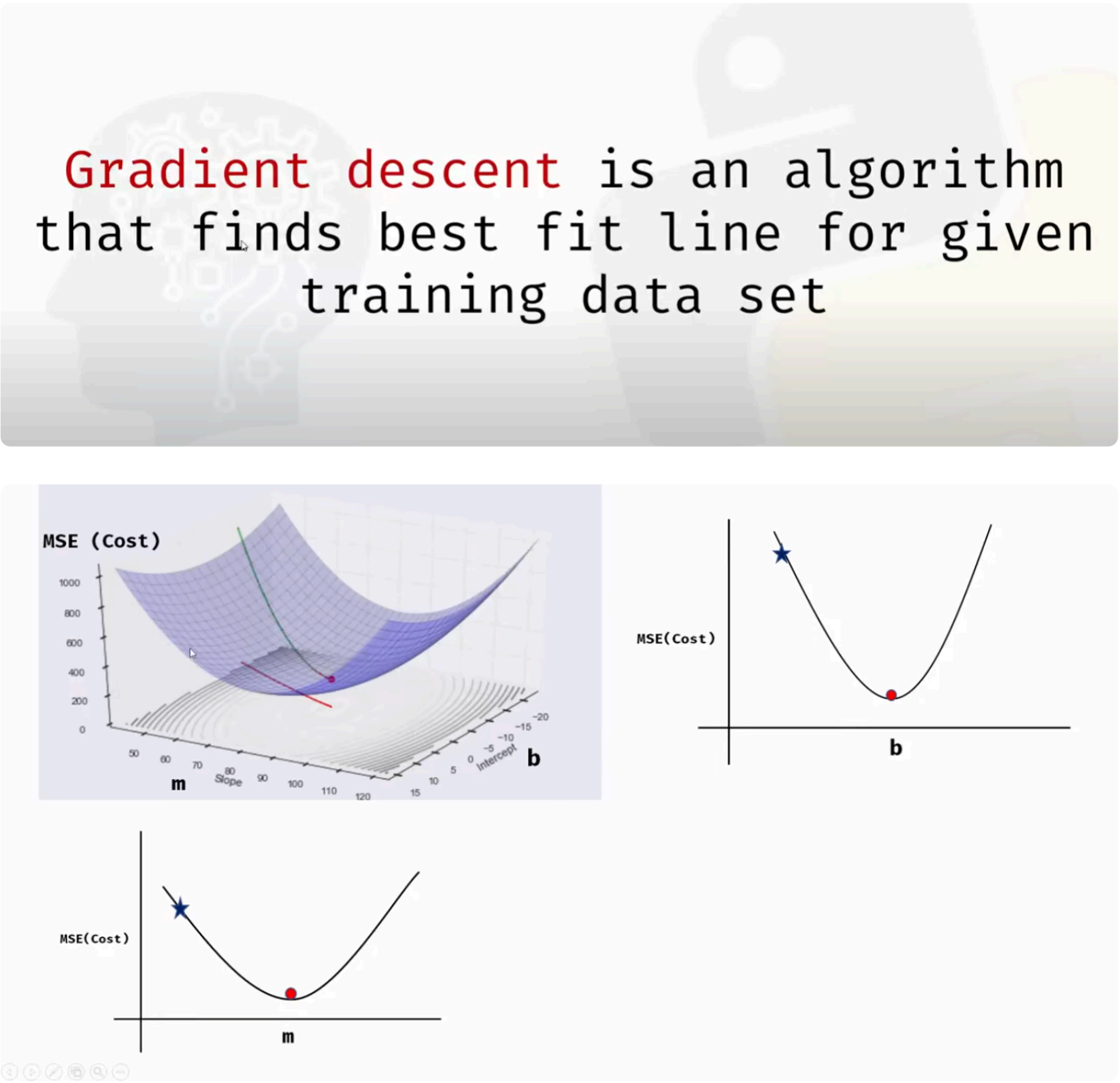
The whole equation is to Find the delta distances between the lines to the point and Find there square and sum them up !!!



y_i **= Actual Data**

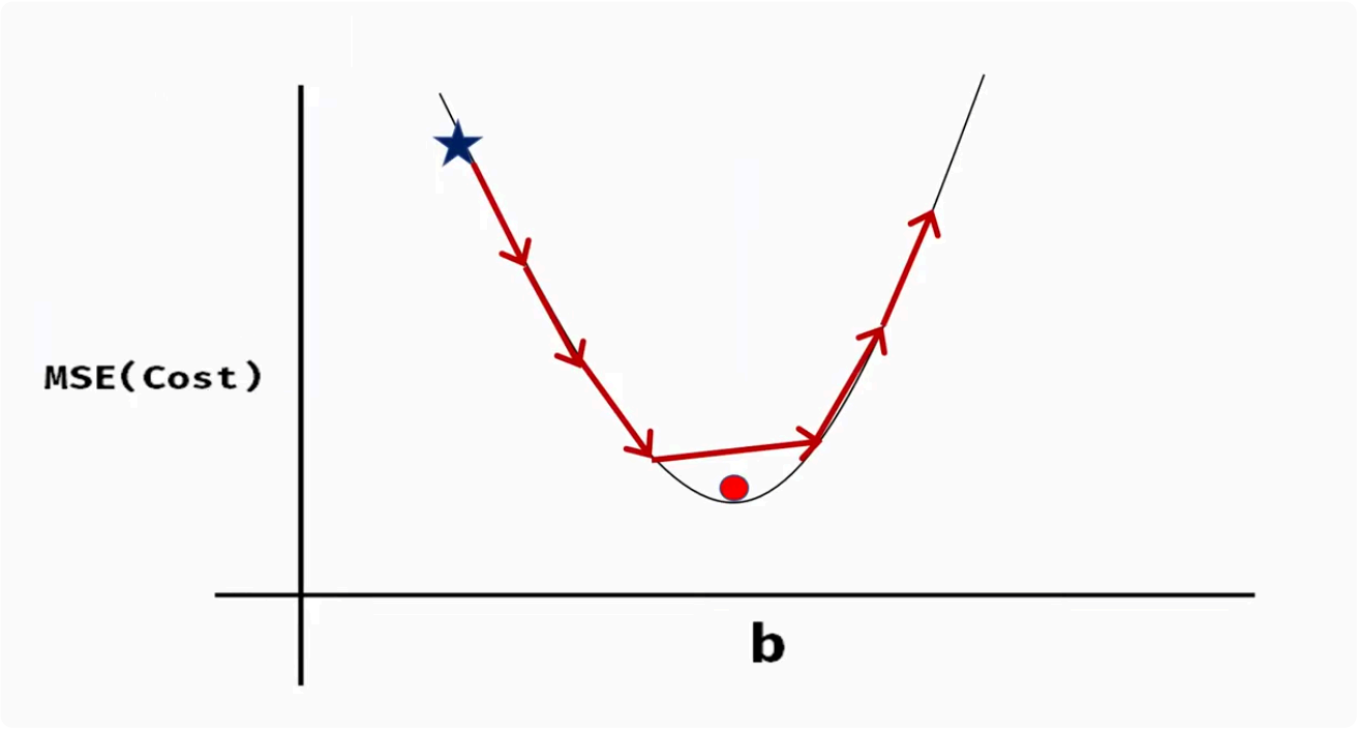
$y_{predicted}$ **= Predicted data**

$(y_i - y_{predicted})^2$ **= To find the delta or Error**

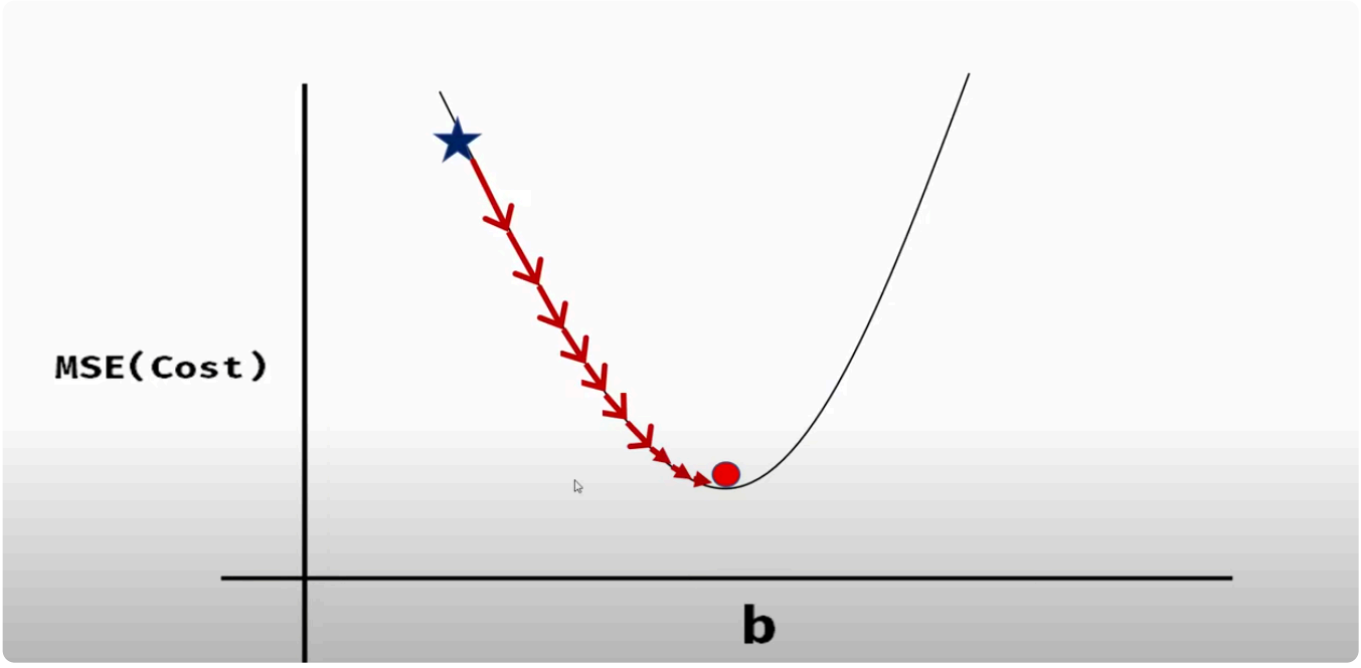


View of the MSE(cost Function)

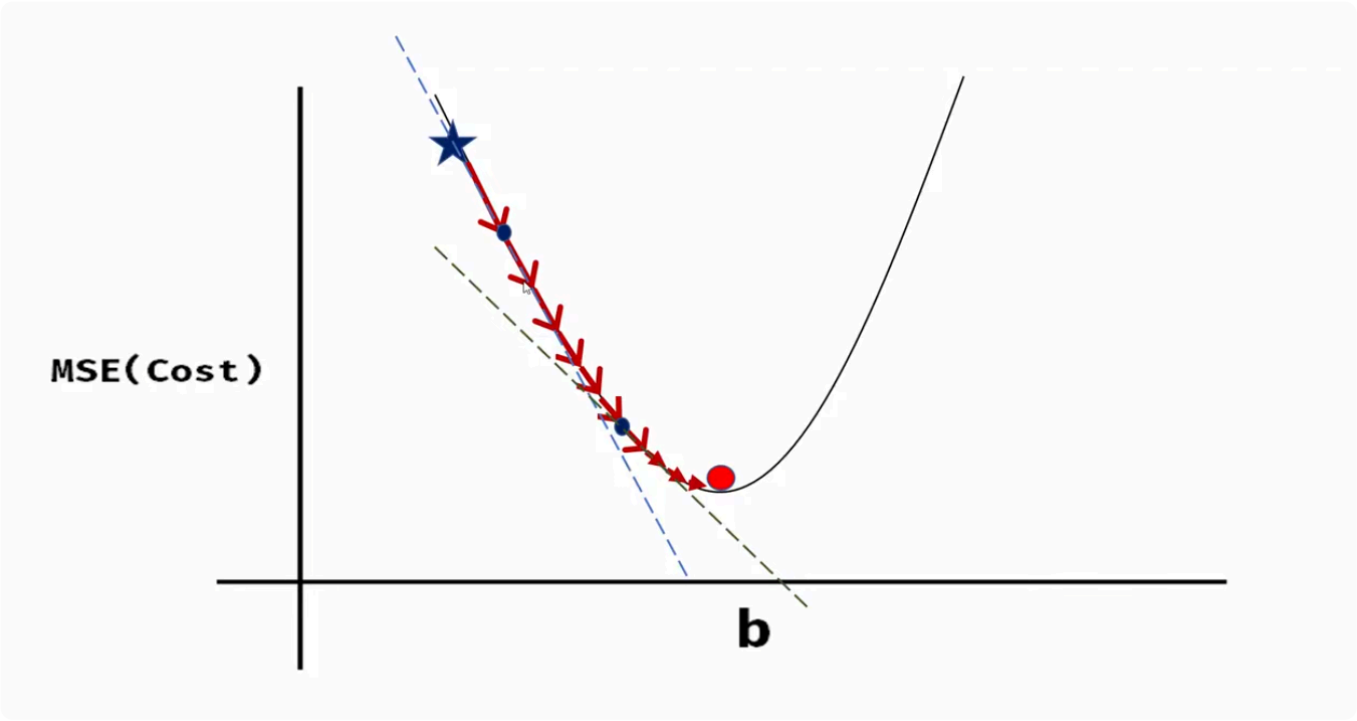
- **The 3D graph shows the the green line which is the Cost Functions & It gets lesser when its moving towards the Red Point it is the Minimum point of a cost .**
- **M & B curves shows the point of minima**



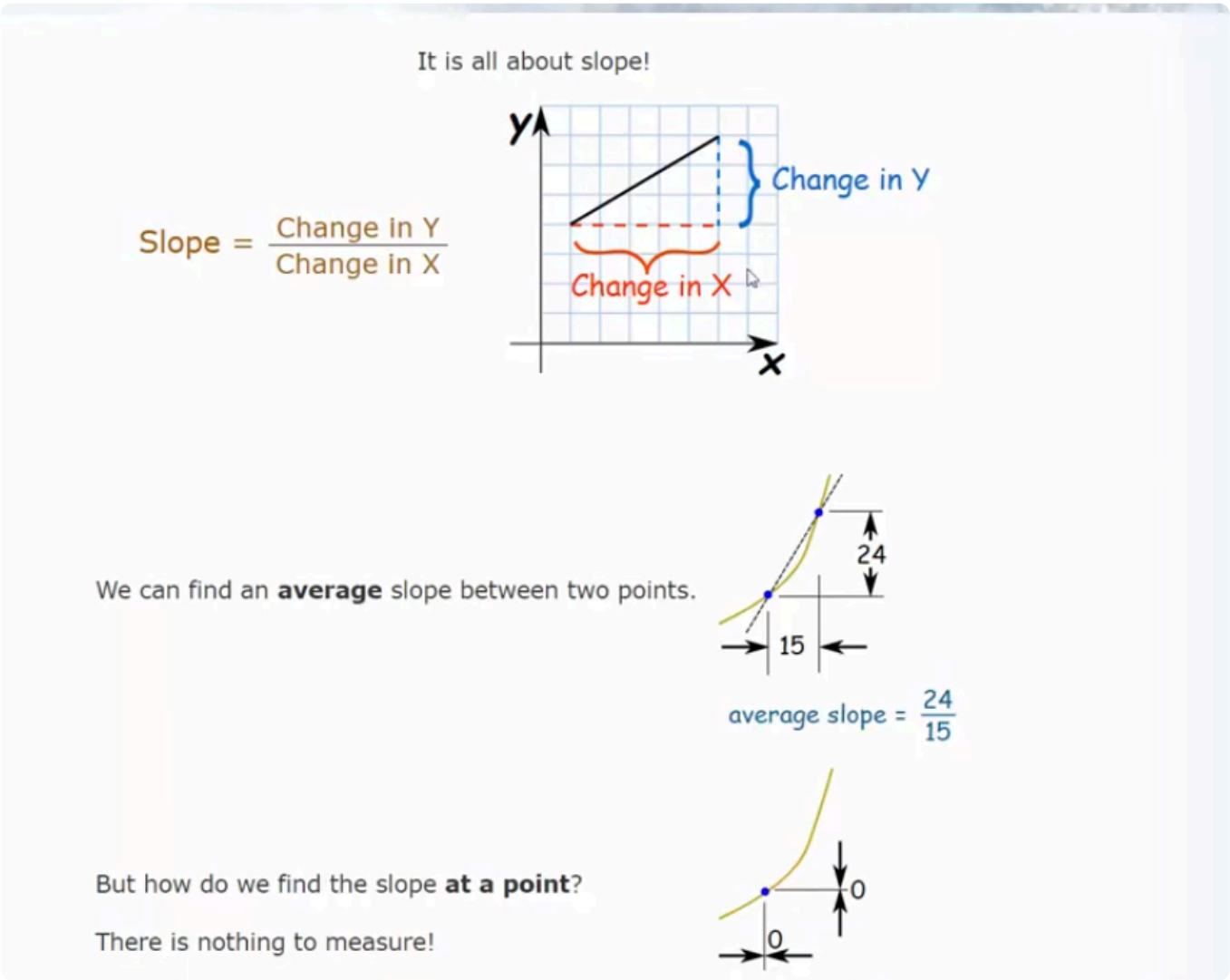
The Steps or the iteration are identical here from Initial to Final so it can miss the minima point of the curve(The red dot shows the minimum of the cost after decrement) . Hence the price will be increasing without getting to its minimal fall.



The arrows are not identical Here . Where as it decreases its length from Initial To final according to which it reaches the Final of minima of the point ,The arrows are getting smaller Hence to find the arrows path

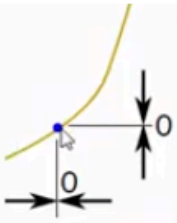


Drawing the Tangent to Next or Closest Point Available



But how do we find the slope **at a point**?

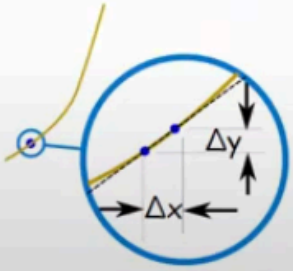
There is nothing to measure!



$$\text{slope} = \frac{0}{0} = ???$$

But with derivatives we use a small difference ...

... then have it **shrink towards zero**.



Let us Find a Derivative!

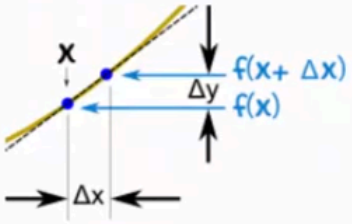
To find the derivative of a function $y = f(x)$ we use the slope formula:

$$\text{Slope} = \frac{\text{Change in Y}}{\text{Change in X}} = \frac{\Delta y}{\Delta x}$$

And (from the diagram) we see that:

x changes from x to $x+\Delta x$

y changes from $f(x)$ to $f(x+\Delta x)$



Now follow these steps:

- Fill in this slope formula: $\frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$
- Simplify it as best we can
- Then make **Δx** shrink towards zero.

```
![[Screenshot 2024-07-07 092902.png]]
```

$$f'(x) = 2x$$

But what about a function of **two variables** (x and y):

$$f(x,y) = x^2 + y^3$$

To find its **partial** derivative **with respect to x** we treat **y as a constant** (imagine y is a number like 7 or something):

$$f'_x = 2x + 0 = 2x$$

Explanation:

- *the derivative of x^2 (with respect to x) is 2x*
- *we **treat y as a constant**, so y^3 is also a constant (imagine y=7, then 7^3=343 is also a constant), and the derivative of a constant is 0*

To find the partial derivative **with respect to y**, we treat **x as a constant**:

$$f'_y = 0 + 3y^2 = 3y^2$$

```
![[Screenshot 2024-07-07 093124.png]]
```

Overview:

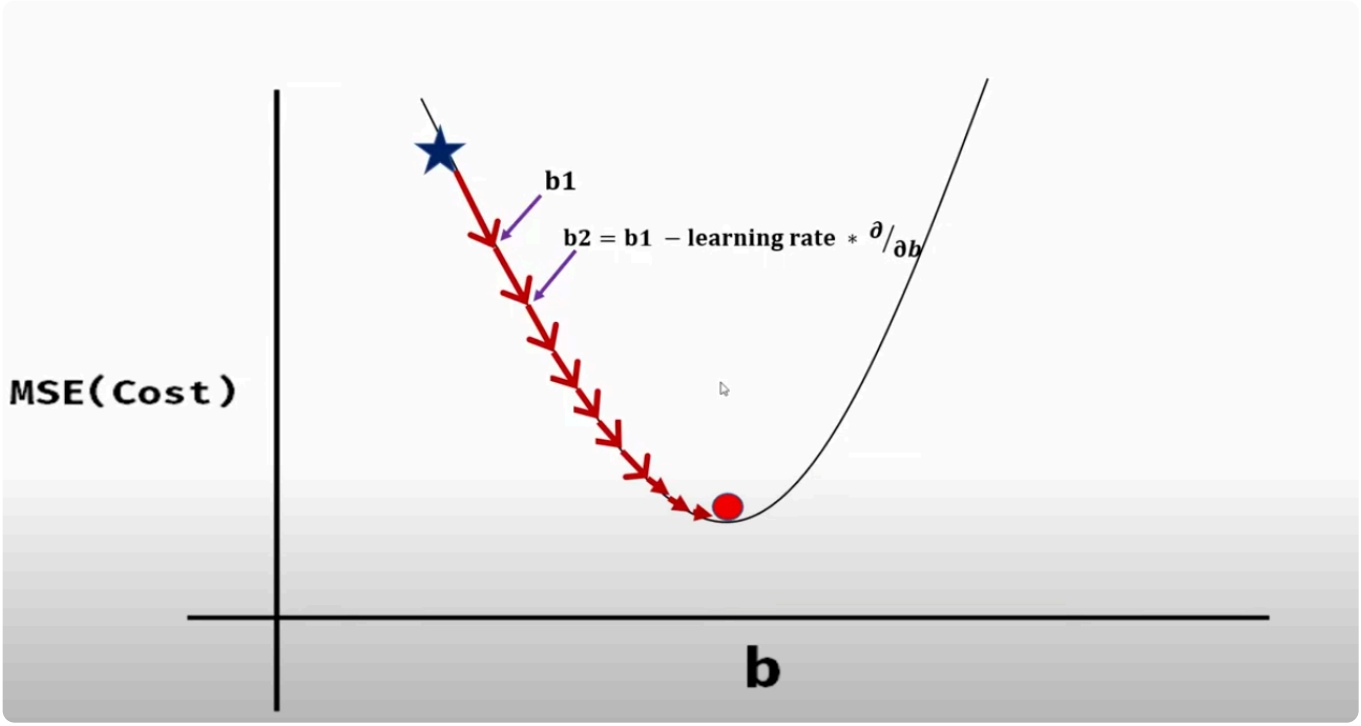
$$f(x,y) = x^3 + y^2$$

$$\partial f / \partial x = 3x^2 + 0 = 3x^2$$

$$\partial f / \partial y = 0 + 2y = 2y$$

```
![[Screenshot 2024-07-07 093316.png]]
```

As Above Plotting to Find the Actual distance of From one Arrow to Other Arrow we Use : -



\

Its the partial Derivative which shows Slope and Learning rate shows the Iteration Steps = (**learning rate** * $\frac{\partial}{\partial m}$)

$$m = m - \text{learning rate} * {}^{\partial}/_{\partial m}$$

$$b = b - \text{learning rate} * {}^{\partial}/_{\partial b}$$

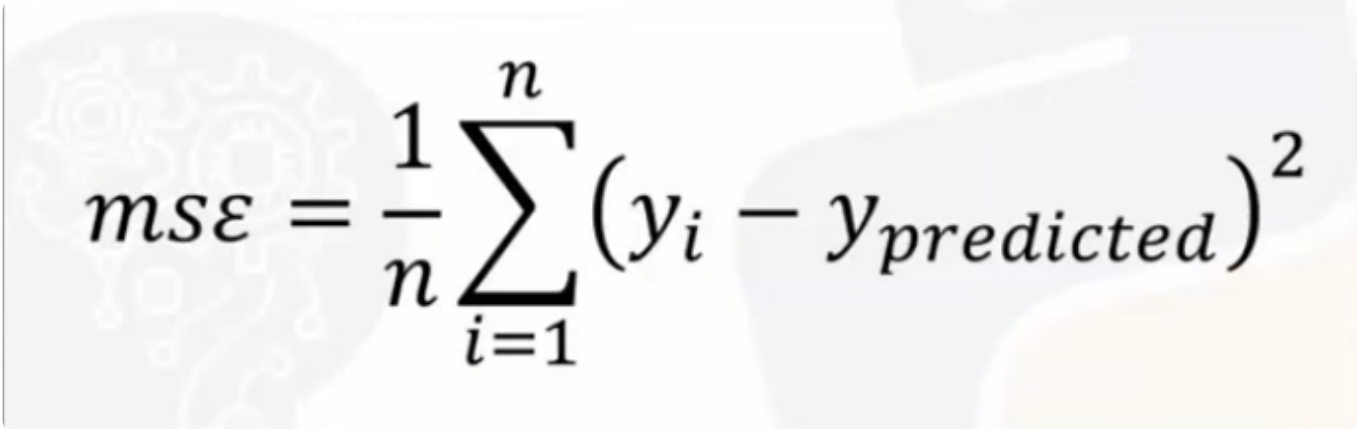
Code To calculate Cost Functions:

```
import numpy as np
def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 1000
```

```
    n = len(x)
    learning_rate = 0.001
    for i in range(iterations):
        yp = m_curr *x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-yp)])
        md = -(2/n)  * sum(x*(y-yp))
        bd = -(2/n) * sum(y-yp)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print("m {}, b{} , cost{} iterations {} " .format(m_curr,b_curr,cost,
i ))

pass

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])
gradient_descent(x,y)
```



$$mse = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_{predicted}\right)^2$$

Code:- cost = $(1/n) * sum([val**2 for val in (y-yp)])$

$$\partial/_{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i\left(y_i - (mx_i + b)\right)$$

$$\partial/_{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -\left(y_i - (mx_i + b)\right)$$

Code:-

$md = -(2/n) * sum(x * (y - yp))$

$bd = -(2/n) * sum(y - yp)$

$$m = m - \text{learning rate} * \partial/_{\partial m}$$

$$b = b - \text{learning rate} * \partial/_{\partial b}$$

CODE:-

```
m_curr = m_curr - learning_rate * md
b_curr = b_curr - learning_rate * bd
```

GradentDescent.ipynb

`import numpy as np` import matplotlib.pyplot as plt %matplotlib inline

`def gradient_descent(x,y):` %matplotlib inline

`def gradient_descent(x,y):`

`m_curr = b_curr = 0`

```
rate = 0.01
n = len(x)
plt.scatter(x,y,color='red' ,marker='+' ,linewidth='5')
for i in range(10000):
y_predicated = m_curr * x + b_curr
print (m_curr,b_curr,i)
plt.plot(x,y_predicated,color='green')
md = -(2/n) * sum(x*(y-y_predicated))
yd = -(2/n) * sum(y-y_predicated)
m_curr = m_curr - rate *md
b_curr = b_curr - rate * yd
x = np.array([1,2,3,4,5])
```
y = np.array([5,7,9,11,13])
gradient_descent(x,y)