Linear Regression

1

Dr. Jayaraj P B
Associate Professor

Department of Computer Science & Engineering,
NIT Calicut

Outline

- Liner Regression – Overview
- Parameters of the Regression
- How to model the parameters
- Cost Function
- Gradient Descent Algorithm
- Model Evaluation

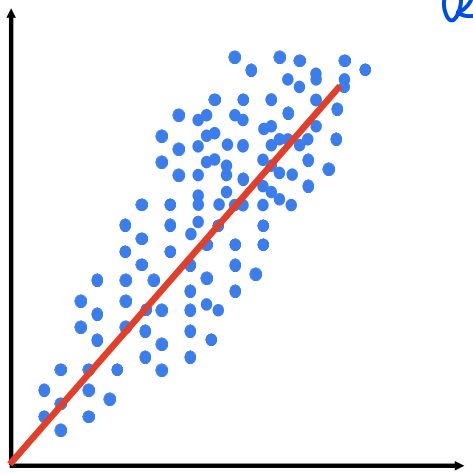
∴ In linear Reg we have a dependent variable y that is dependent on independent variable x.

Linear Regression

and this relation is a linear relation.

1. Linear Regression is a method in statistics used to predict the value of a variable based on another variable.
2. Linear regression focuses on the linear dependence of variables.
3. Graphically, given a set of points the aim of linear regression is to find the line that best fits the points.

∴ We call it Linear Regression.



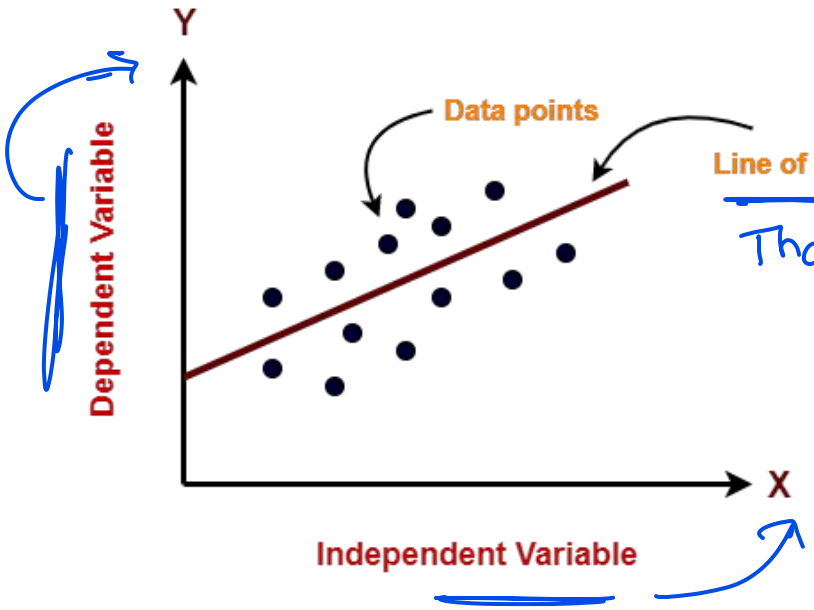
We are given set of data points and our main aim is to fit the best line

Linear regression

- Linear regression is a way to identify a relationship between two or more variables and use these relationships to predict values of one variable for given value(s) of other variable(s).
- Regression captures the correlation between variables observed in a data set
- Linear regression assume the relationship between variables can be modelled through linear equation or an equation of line

Linear Regression

- Linear regression model represents the linear relationship between a dependent variable and independent variable(s) via a sloped straight line.



That best fitted line is known as Line of Regression.

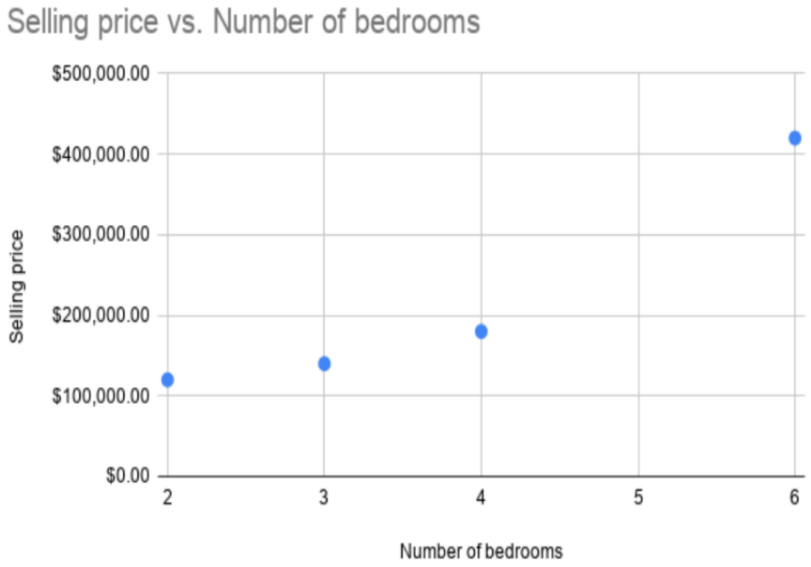
Problem Definition

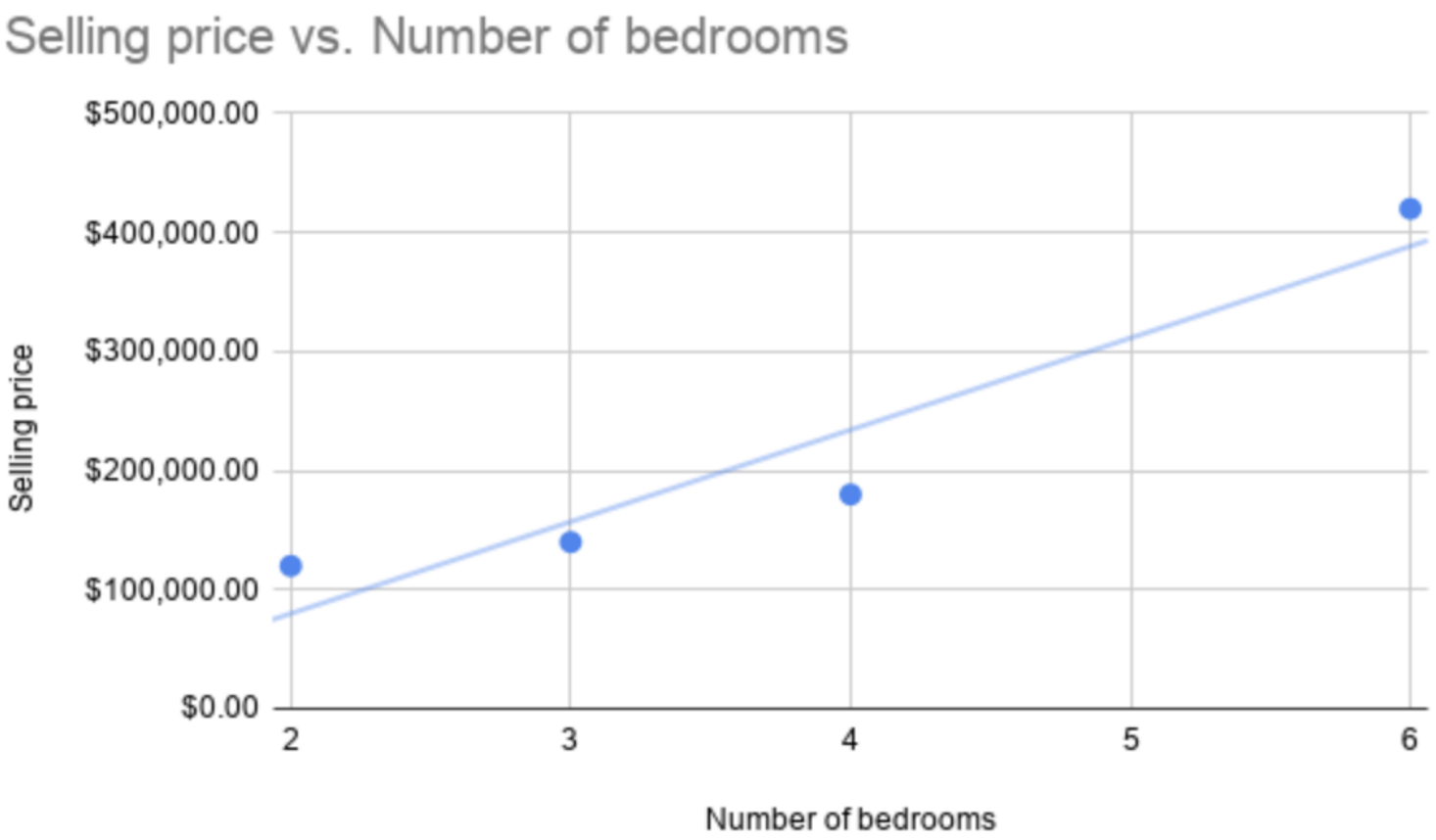
1. Consider the problem of house price prediction.
2. Let the price of a house (R), depend on only one factor the number of rooms (A)
3. The input dataset contains (number of rooms, price of house)
4. The task is to predict the price of a house for a given input of number of rooms.

Price (dep) of house depends on no. of rooms (indep)

Number of Rooms (A)	Price of House
1	20
2	30
3	40

Number of bedrooms	Selling price
2	$120,000
3	$140,000
4	$180,000
6	$420,000

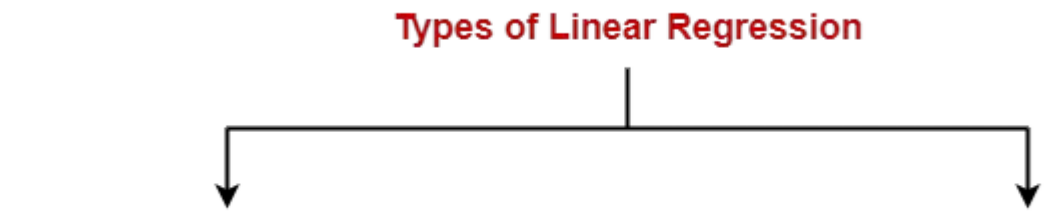




This model assumes a linear relationship between the dependent variable and the predictor variable

Types of Linear Regression

We predict continuous values of o/p in linear Regression



↓

Dependent Variable → y

Independent variable → x

↓

Many x will be there here

$y = \theta_0 + \theta_1 x$

$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \cdots + \theta_n x_n$.

Simple Linear Regression

$$y = w_0 + w_1 x$$

- In this equation:
- **y is the output variable**. It is also called the dependent variable in statistical modelling. It represents the continuous value that we are trying to predict.
- **x is the input variable**. In machine learning, x is referred to as the **feature**, while in statistics, it is called the independent variable.
- **w0** is the **bias** term or y-axis intercept.
- **w1** is the **regression coefficient** or scale factor. In classical statistics, it is the equivalent of the slope on the best-fit straight line that is produced after the linear regression model has been fitted.
- **wi** are called **weights** in general.
- The goal of the regression analysis (**modelling**) is to find the values for the unknown parameters of the equation; that is, to find the values for the weights **w0 & w1**.

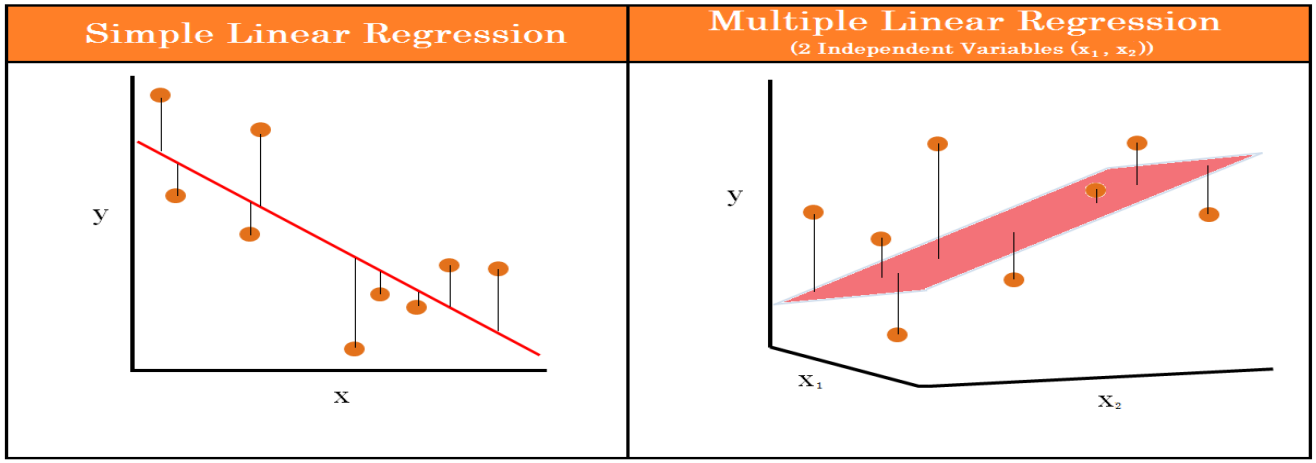
Multiple Linear Regression

Both simple and multiple linear regressions assume that there is a **linear relationship** between the input variable(s) and the output target variable.

The main difference is the number of independent variables that they take as inputs. Simple linear regression just takes a single feature, while multiple linear regression takes multiple x values.

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

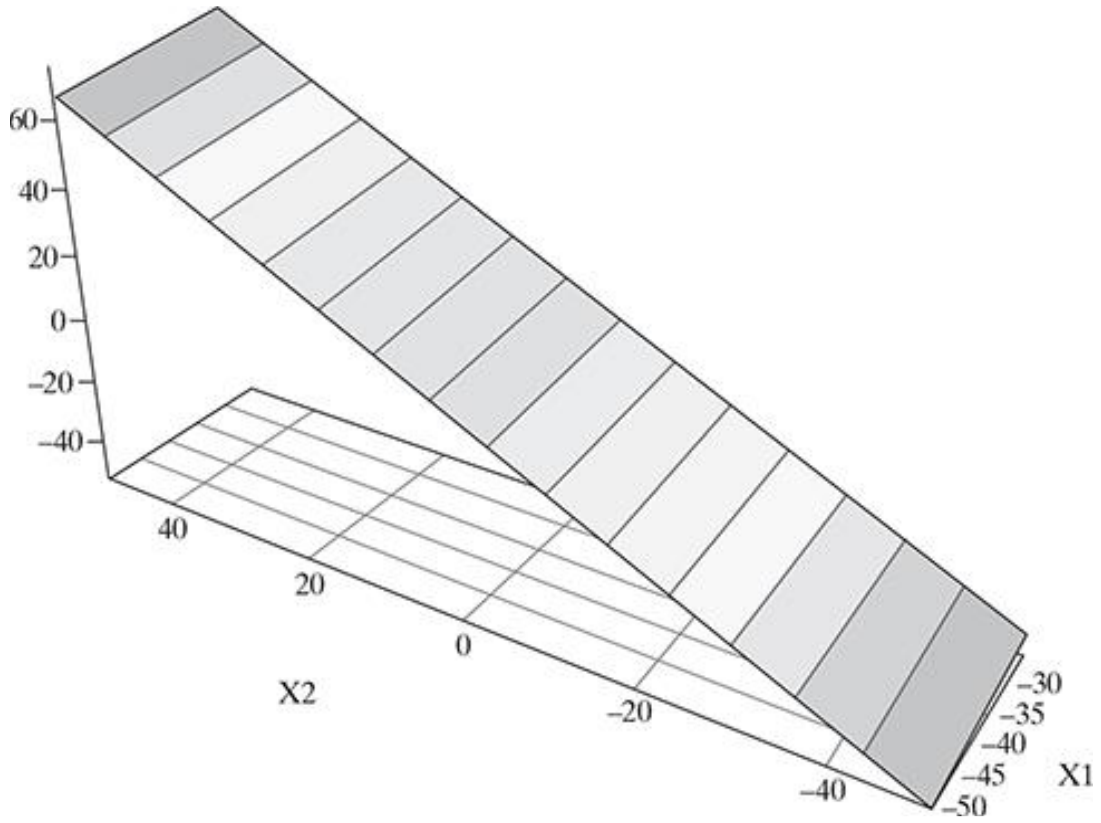
Where xi is the i-th feature with its own wi weight



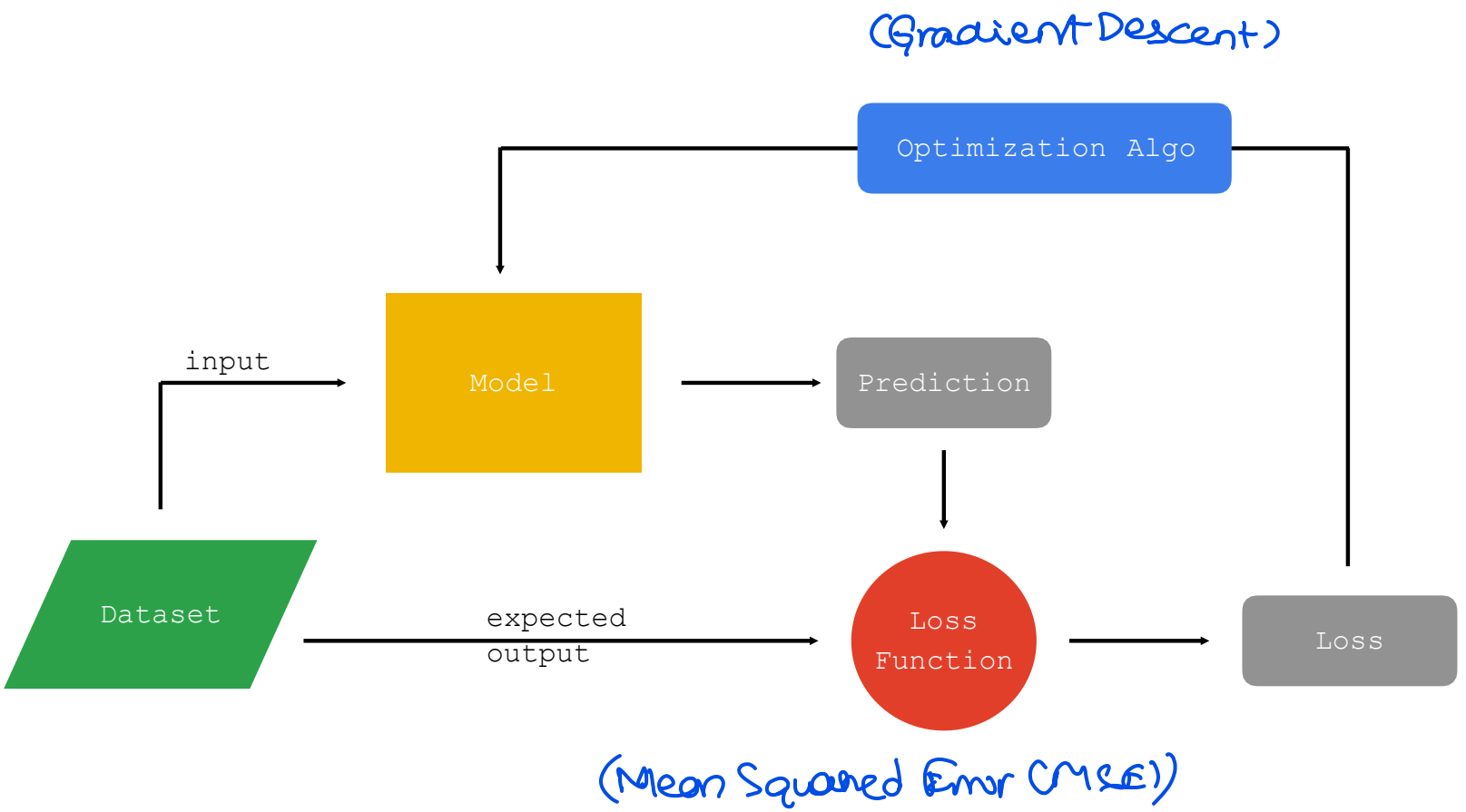
The simple linear regression model can be represented graphically as a best-fit line between the data points, while the multiple linear regression model can be represented as a plane (in 2-dimensions) or a hyperplane (in higher dimensions).

Multiple Linear Regression

- Price = f (Area location, floor, Ageing, Amenities)



Training the Linear Regression



Training the Linear Regression

- We train the linear regression algorithm with a method named Mean Squared Error (MSE). The goal of training is to find the weights wi in the linear equation y = w0 + w1x.
- This has four main steps in machine learning:
- 1. Random weight initialization. In practice, w0 and w1 are unknown at the beginning. The goal of the procedure is to find the appropriate values for these model parameters. To start the process, we set the values of the weights at random.
- 2. Input the initialized weights into the linear equation and generate a prediction for each observation point. To continue with the example that we've already used:

Training the Linear Regression

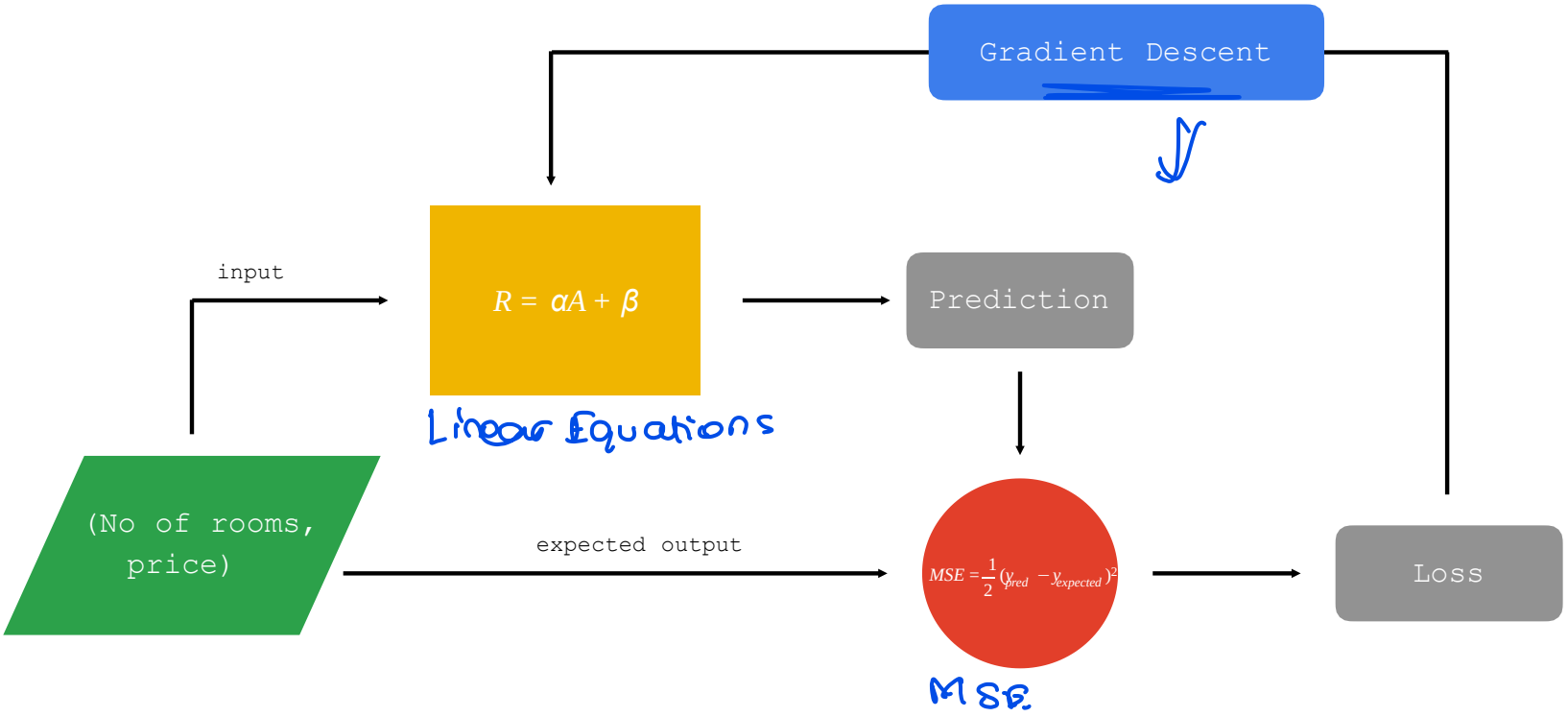
3. Calculate the MSE

We obtain residuals by calculating actual values - predicted values for each observation. We square the residuals. And we sum the residues

4. Model parameter selection to minimize the error.

We find the best parameters for the linear model by defining a cost function and minimizing it via gradient descent.

Training the Linear Regression



Cost Function

- The cost function is a formal representation of an objective that the algorithm is trying to achieve.
- In the case of linear regression, the cost function is the MSE
- The algorithm solves the minimization problem - it tries to minimize the cost function in order to achieve the best fitting line with the lowest errors.
- This is achieved through **gradient descent**.

$$\text{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Number of Bedrooms	Actual Selling Price	Predicted Selling Price	Residuals (Actual-predicted)
2	$120,000	$80,000	$40,000
3	$140,000	$157,143	-$17,143
4	$180,000	$234,286	-$54,286
6	$420,000	$388,572	$31,428

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Gradient Descent

- Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

Gradient → minimization algorithm

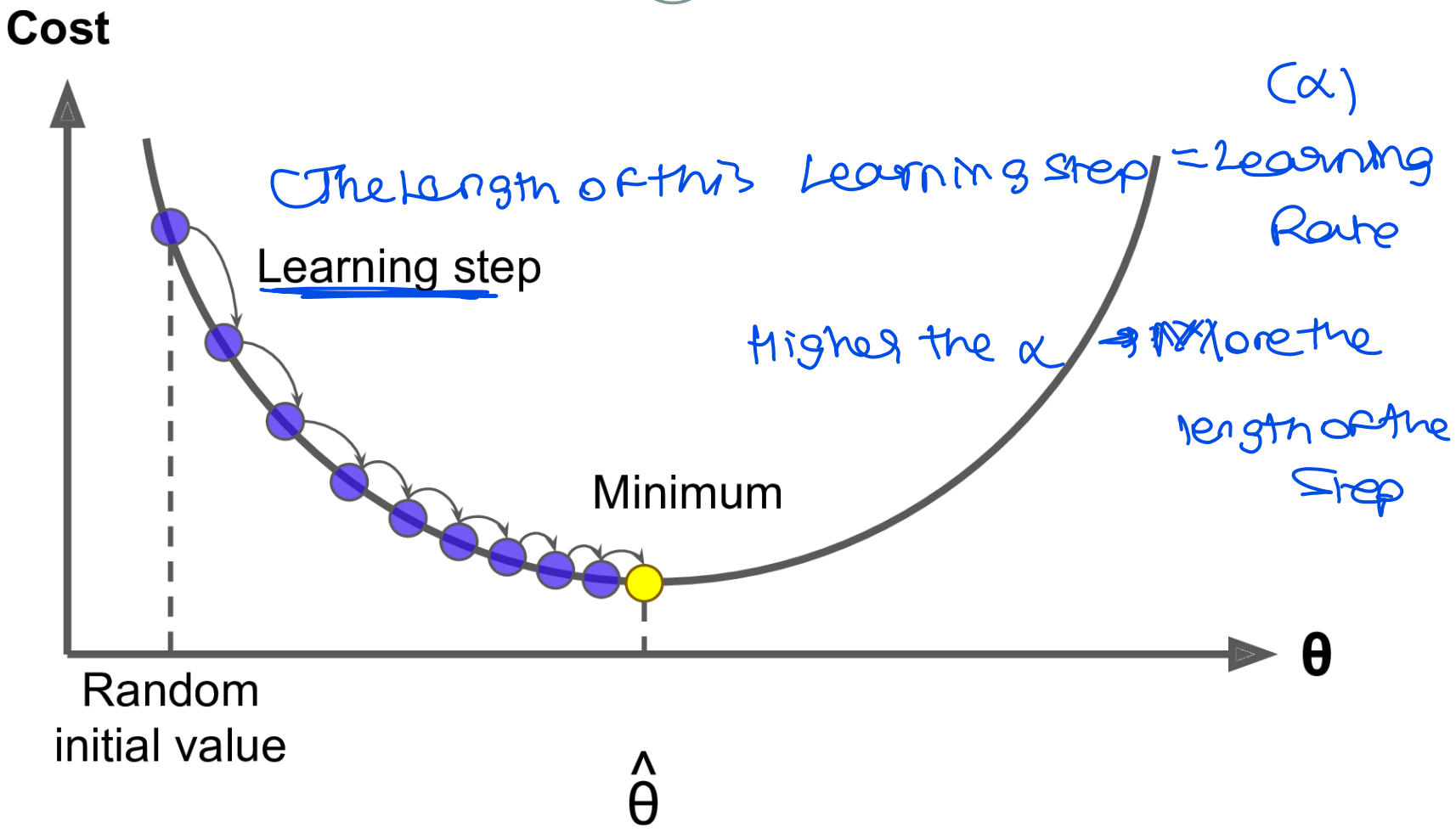
Descent

We calculate error at each point - We take partial derivative to find slope - Based on it we reiterate until we get minimum Error.

Gradient Descent

- Gradient descent is a method of changing weights based on the loss function for each data point. We calculate the sum of squared errors at each input-output data point.
- We take a partial derivative of the weight and bias to get the slope of the cost function at each point.
- Based on the slope, gradient descent updates the values for the set of weights and the bias and re-iterates the training loop over new values
- This iterative approach is repeated until a minimum error is reached, and gradient descent cannot minimize the cost function any further.

Gradient Descent



Gradient Descent Steps

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \text{MSE}(\theta)$$

We iterate this until we get minimum error.

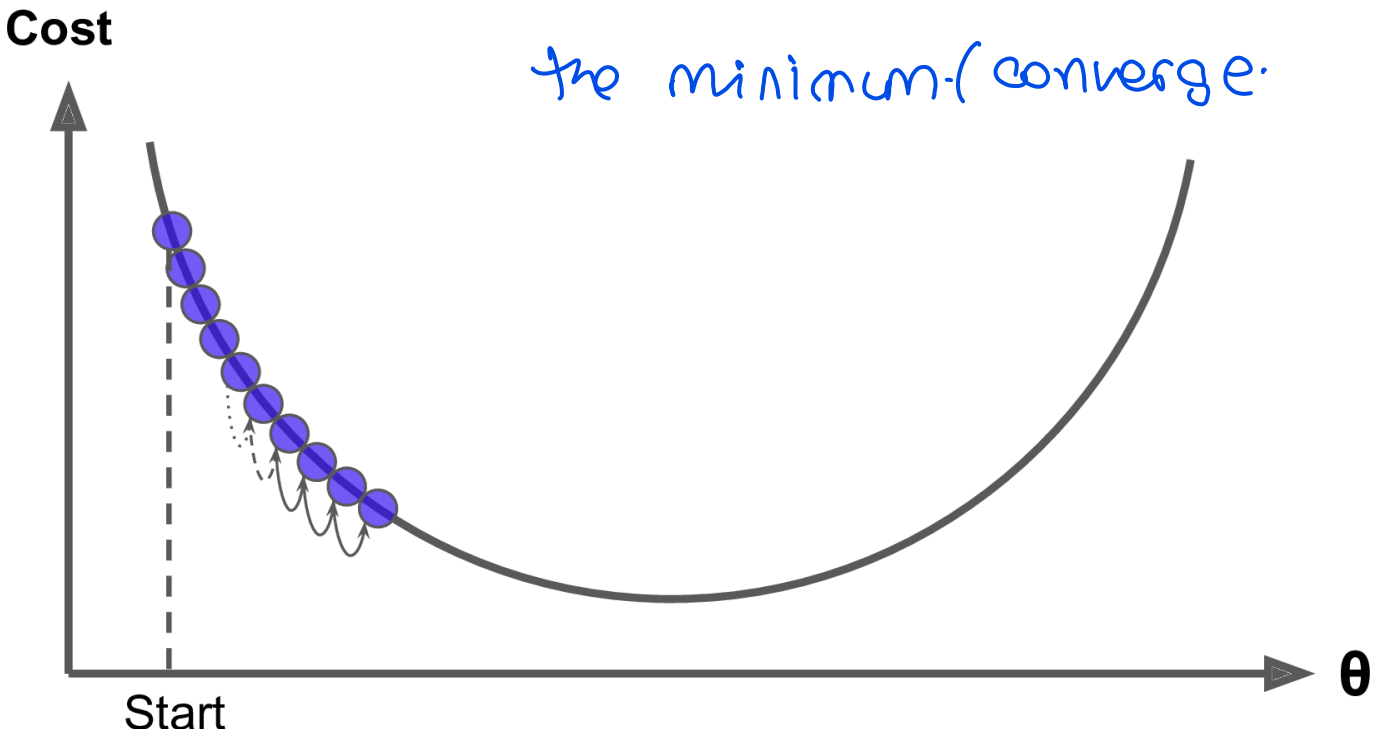
$\theta_{new} = \theta_{curr} - \alpha \left(\frac{\partial (MSE)}{\partial \theta} \right)$

α : learning rate

$\frac{\partial (MSE)}{\partial \theta}$: partial derivative

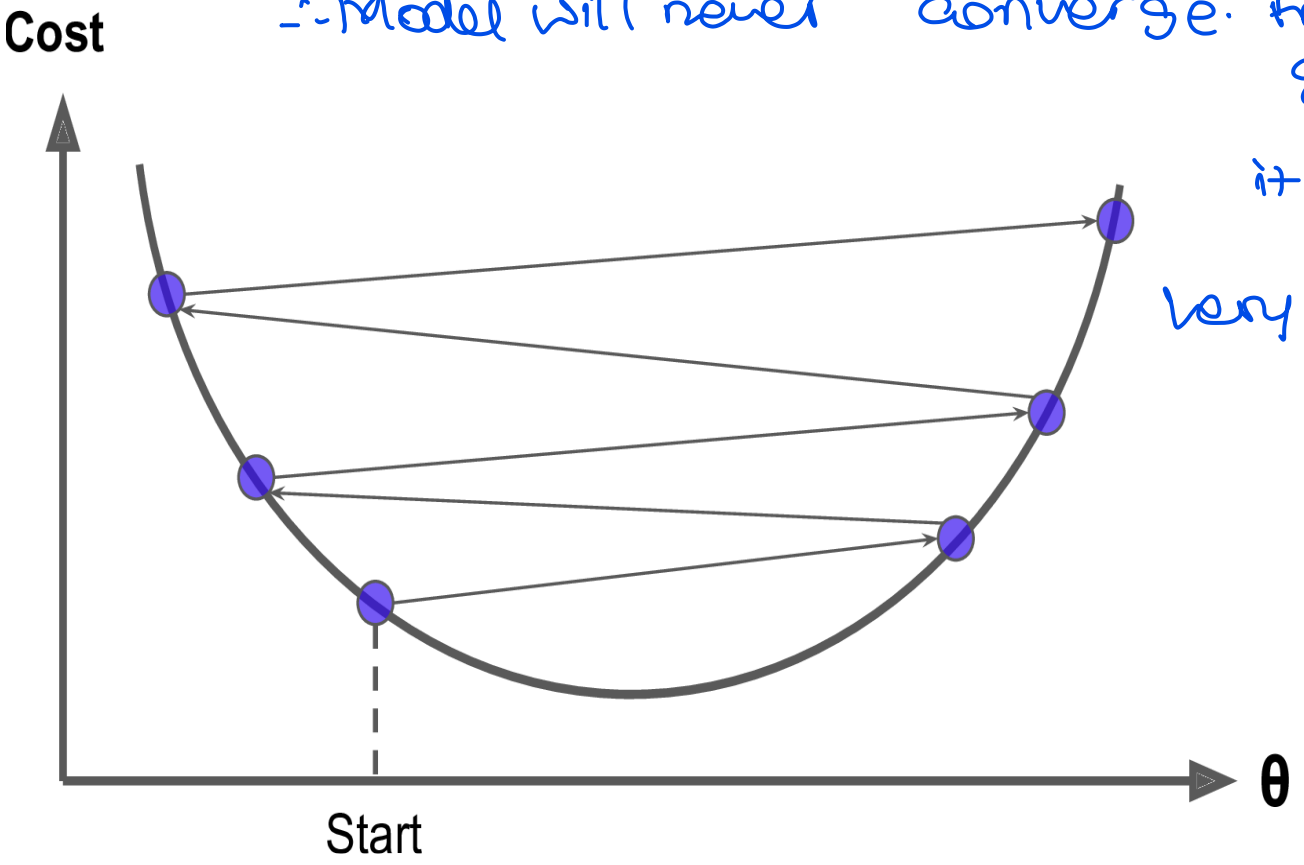
Learning rate too small

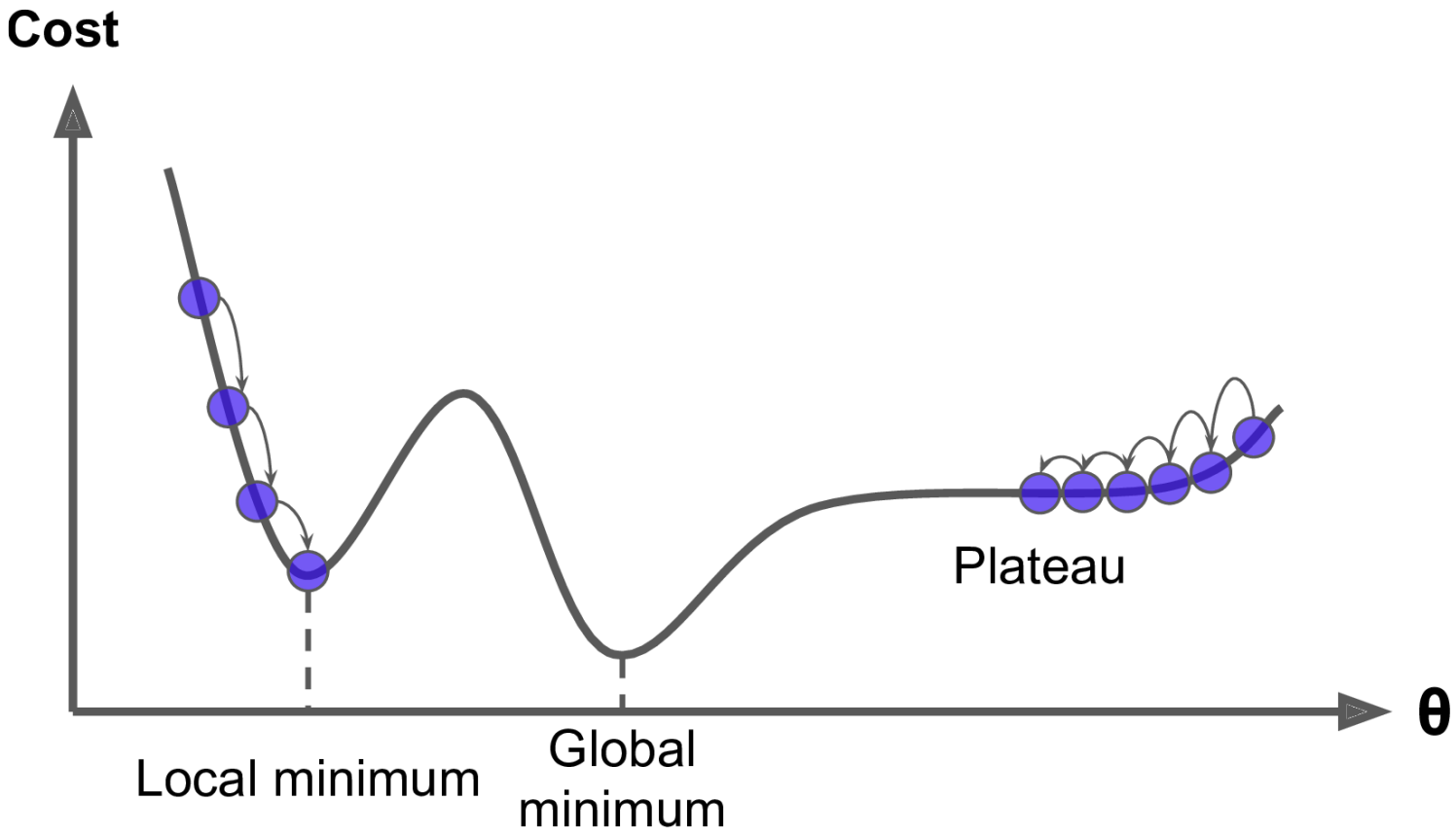
∴ Model takes forever to reach the minimum (converge.



Learning rate too large

∴ Model will never converge. from good to it goes to very bad error.




Cost
Plateau
θ
Local minimum
Global minimum

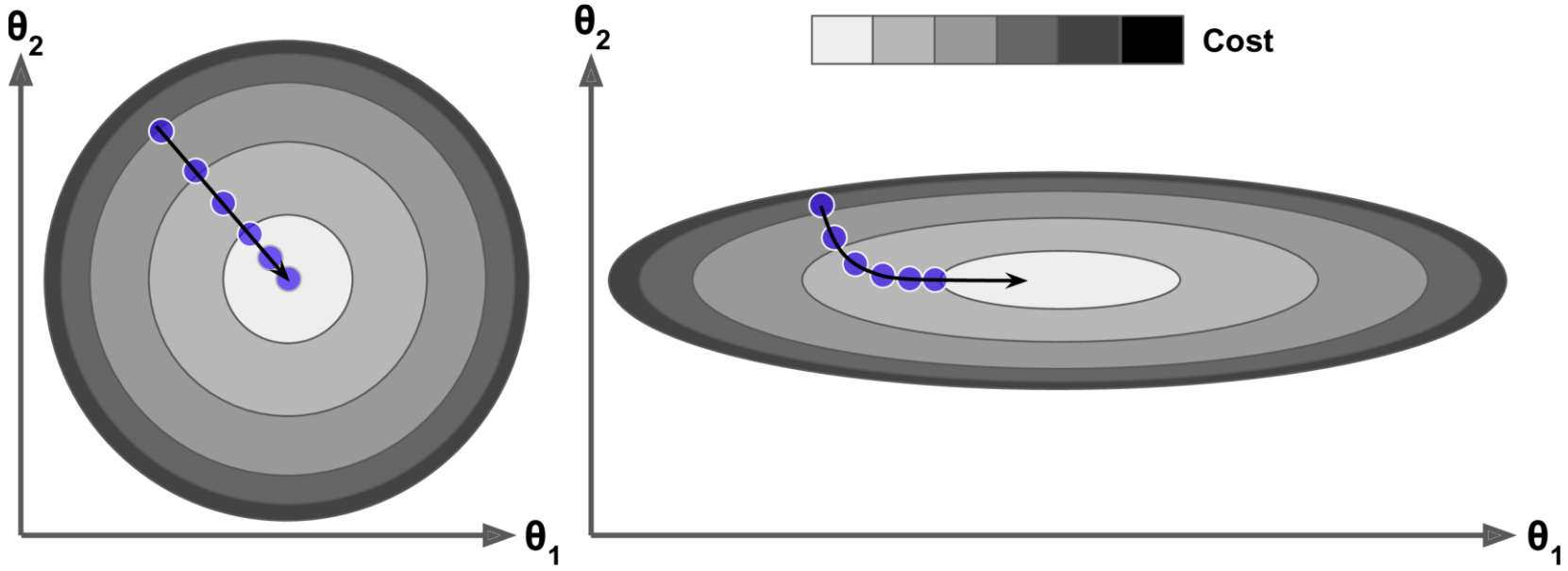
Challenges

- Finally, not all cost functions look like nice regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult.

- if the random initialization starts the algorithm on the left, then it will converge to a *local minimum*, which is not as good as the *global minimum*.

- If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.

elongated bowl



- In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales.

- Figure above shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right)

Batch Gradient Descent

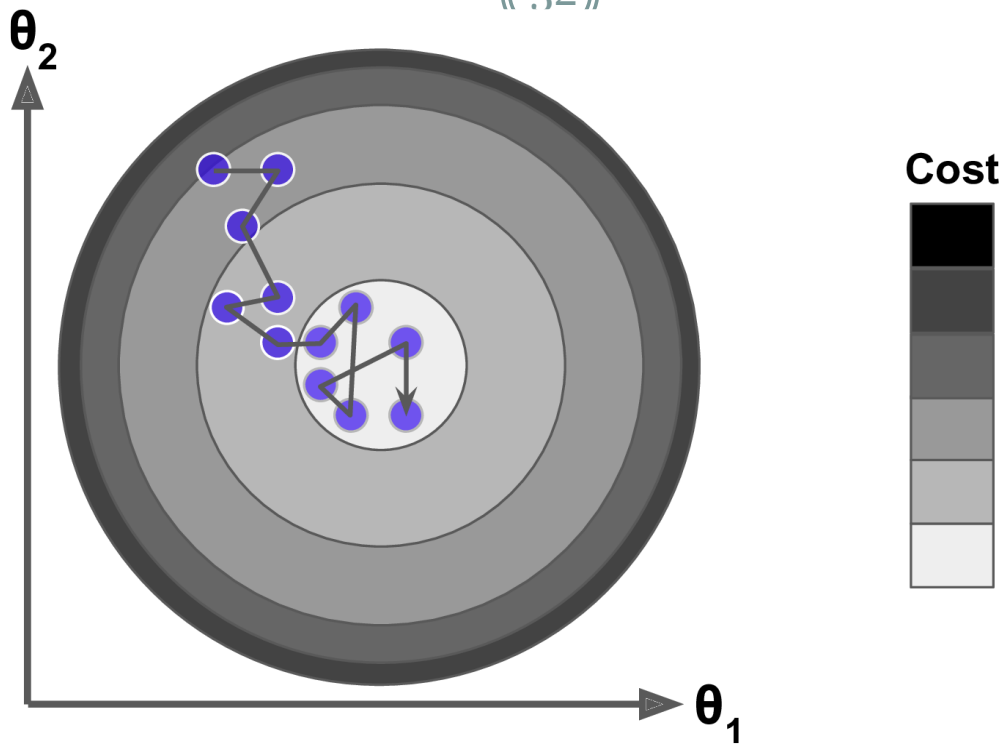
- To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter θj.
- In other words, you need to calculate how much the cost function will change if you change θj just a little bit. This is called a ***partial derivative***.

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- At the opposite extreme, *Stochastic Gradient Descent* just picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- Obviously this makes the algorithm much faster since it has very little data to manipulate at every iteration.
- It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration

Stochastic Gradient Descent



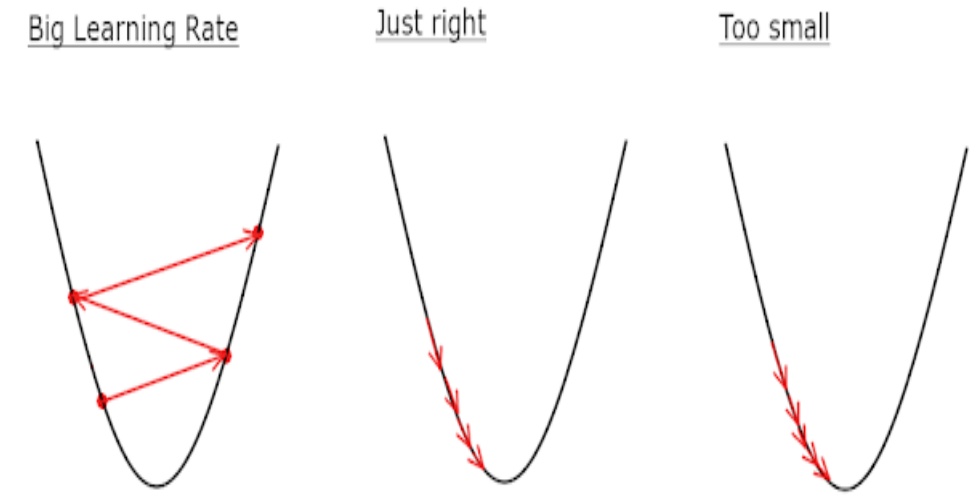
If the learning rate is reduced too slowly, you may jump around the minimum for a long time and end up with a suboptimal solution if you halt training too early.

Mini-batch Gradient Descent

- The last Gradient Descent algorithm we will look at is called *Mini-batch Gradient Descent*. It is quite simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Minibatch GD computes the gradients on small random sets of instances called *minibatches*.
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.

hyperparameter learning rate

- One consideration to bear in mind when using gradient descent: the hyperparameter learning rate.
- The learning rate refers to how much the parameters are changed at each iteration.
- If the learning rate is too high, the model fails to converge and jumps from good to bad cost optimizations.
- If the learning rate is too low, the model will take too long to converge to the minimum error.


Big Learning Rate
Just right
Too small

Model evaluation

- How do we evaluate the accuracy of our model?

- There are various metrics to evaluate the goodness of fit:
  - Mean Squared Error (MSE).
  - R-squared is a measure of how much variance in the dependent variable that our linear function accounts for. d Error (RMSE).

Polynomial Regression

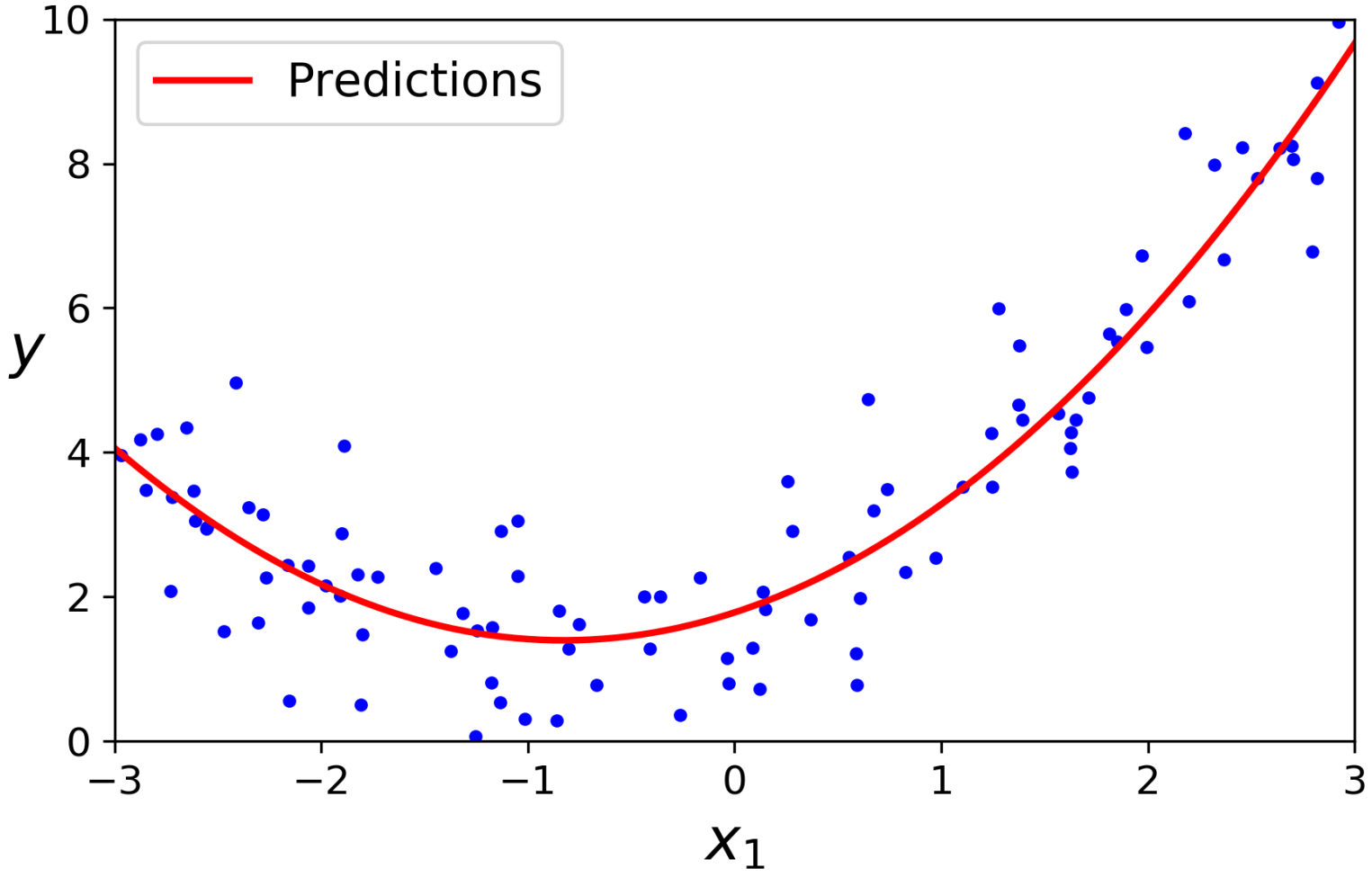
- What if your data is actually more complex than a simple straight line? Surprisingly, you can actually use a linear model to fit nonlinear data.
- A simple way to do this is to add powers of each feature as new features, then train a linear model on this extended set of features. This technique is called *Polynomial Regression*

Polynomial Regression

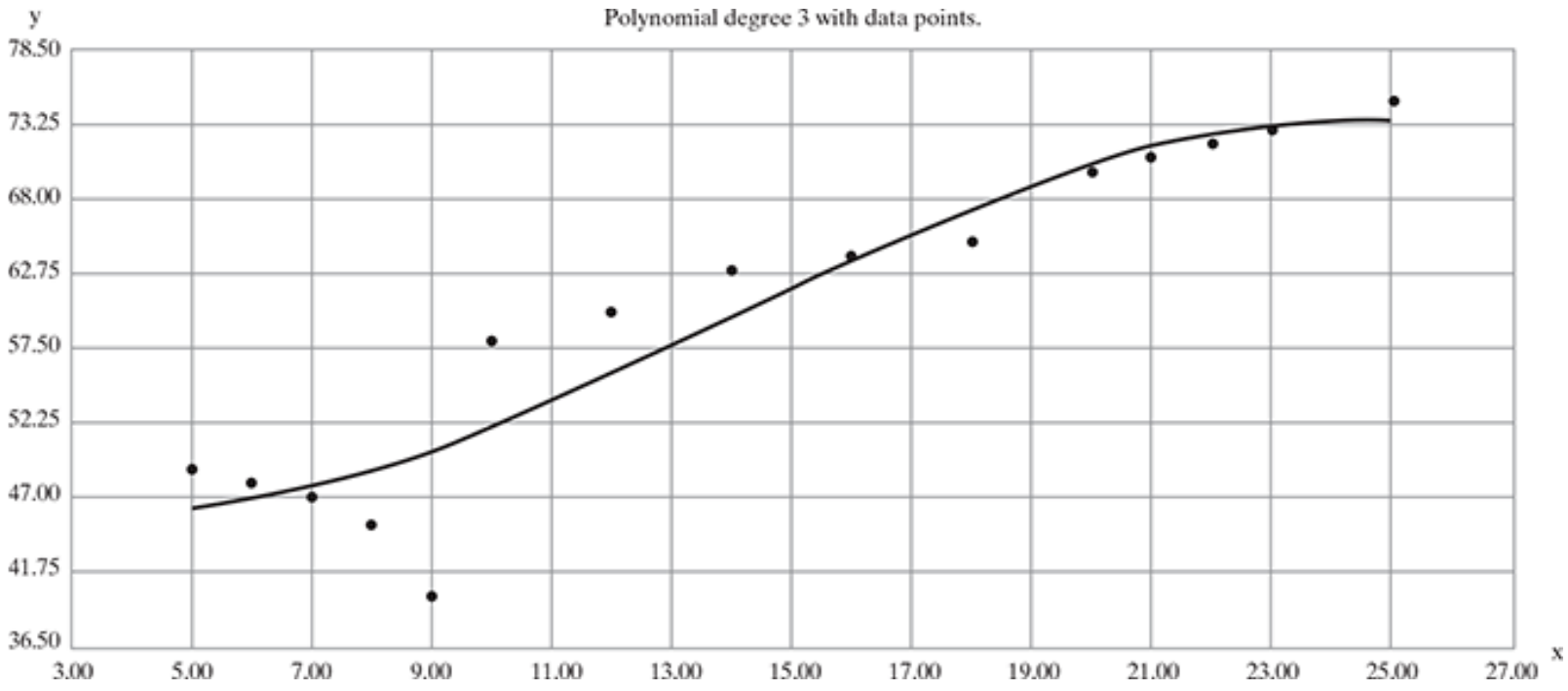
- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

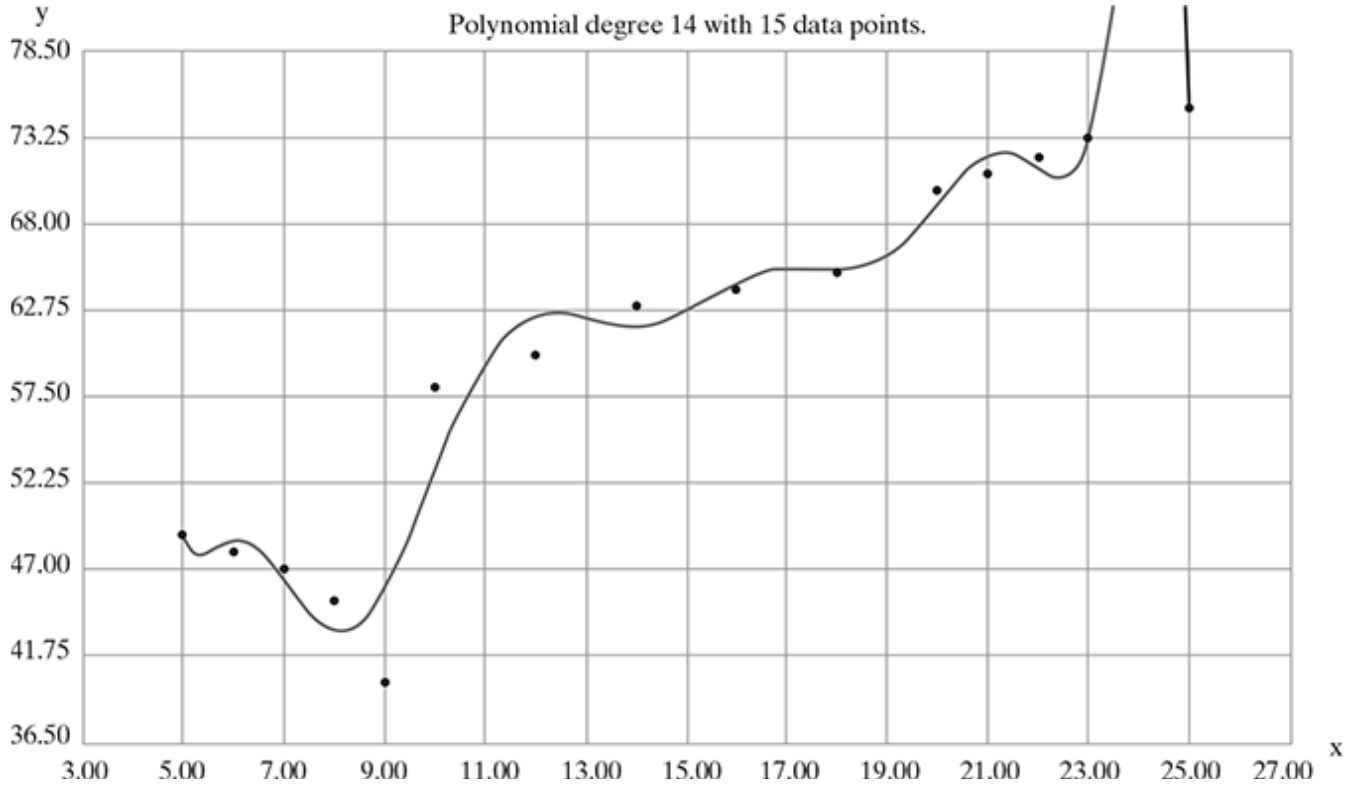
Polynomial Regression



Polynomial regression degree 3

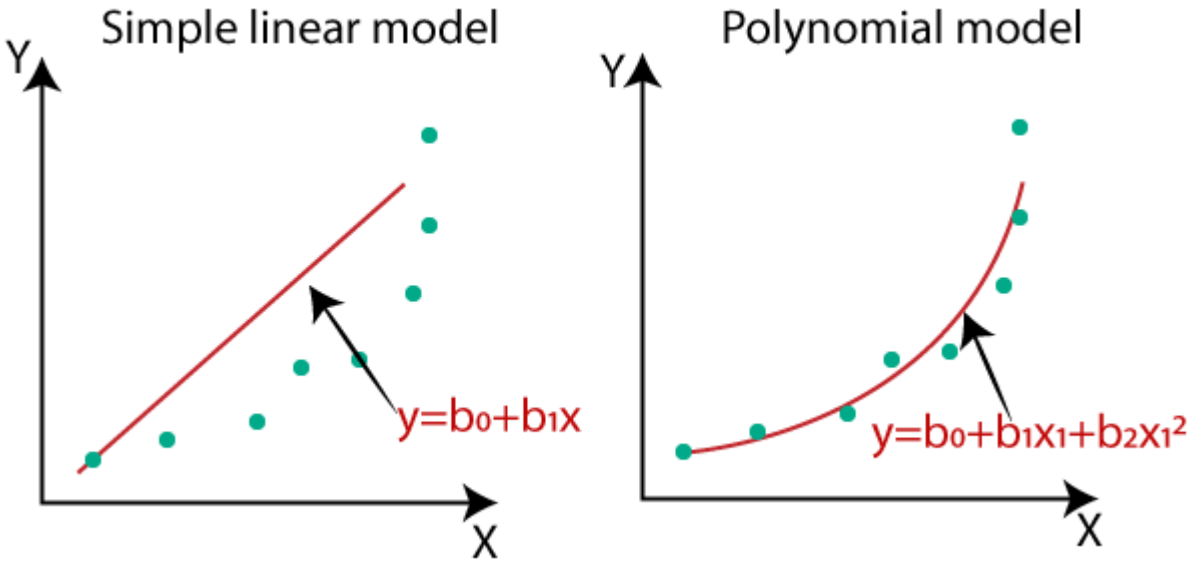


Polynomial regression degree 14



The regression line will curve further if we increase the polynomial degree, the regression line will be overfitting into all the original values of X.

Polynomial Regression



- Equation of the Polynomial Regression Model:
- Simple Linear Regression equation:

  $\mathbf{y = b_0 + b_1 x}$ .....(a)
- Multiple Linear Regression equation

  $\mathbf{y = b_0 + b_1 x + b_2 x_2 + b_3 x_3 + ... + b_n x_n}$ .....(b)
- Polynomial Regression equation:

  $\mathbf{y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + ... + b_n x^n}$ .....(c)

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

$\hat{y}_i = \theta^T \cdot x$ → np.dot(mat1, mat2)

matrix mult

Python implementation


```
def model(X, Y, learning_rate, iteration):
    m = Y.size
    theta = np.zeros((2, 1))
    cost_list = []

    for i in range(iteration):

        y_pred = np.dot(X, theta)
        cost = (1/(2*m))*np.sum(np.square(y_pred - Y))

        d_theta = (1/m)*np.dot(X.T, y_pred - Y)
        theta = theta - learning_rate*d_theta

        cost_list.append(cost)

    return theta, cost_list
```

→ - is a array of 2 elements

→ we are iterating for many times.

we find cost each time store it

from that list

at final we find min cost