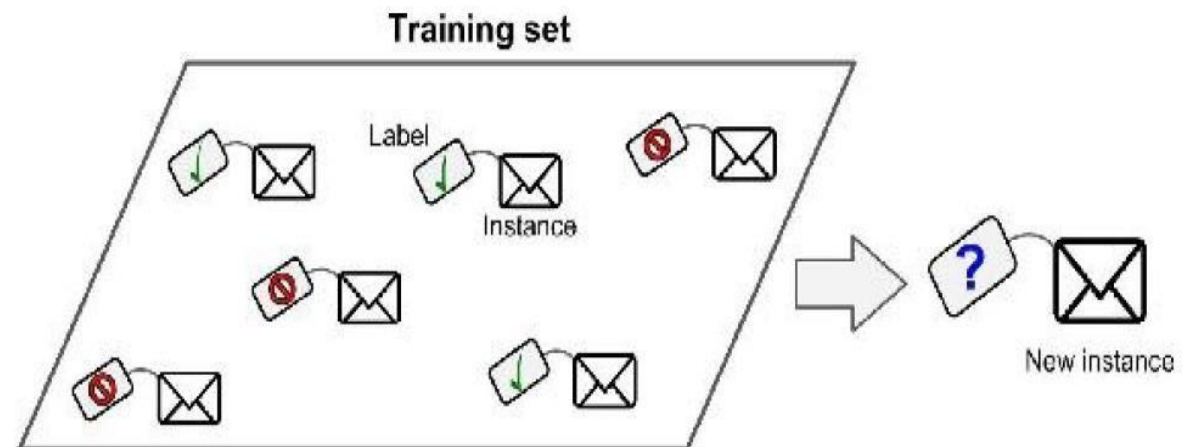


# Objectives

- Supervised Learning
- Models in Classification and Regression.
- Supervised Learning Algorithms.

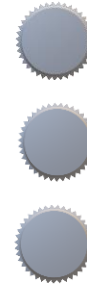
# Supervised Learning

- Works under supervision
  - Algorithm learns from labelled samples
- Predict the output/label of an unseen sample
- Eg: Email Spam Filter.



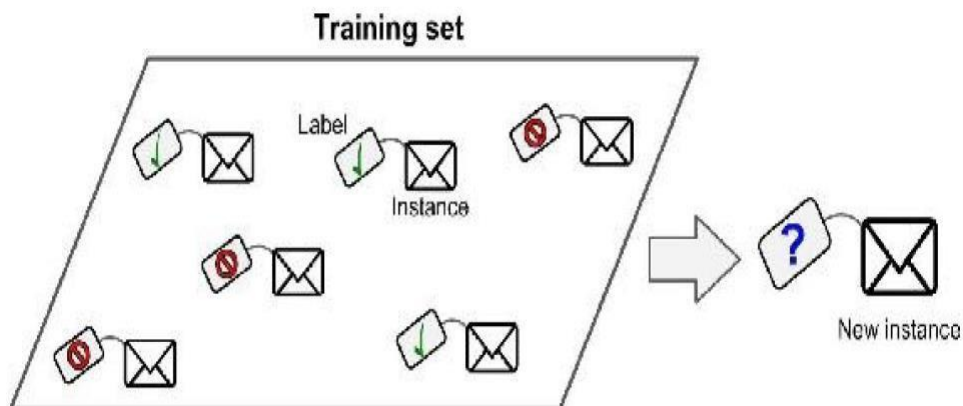
# Classification vs Regression

**Target** : Spam/Non-Spam(Discrete)  
**Features** : Phrases in emails  
**Model** : A linear/polynomial function that separates data samples

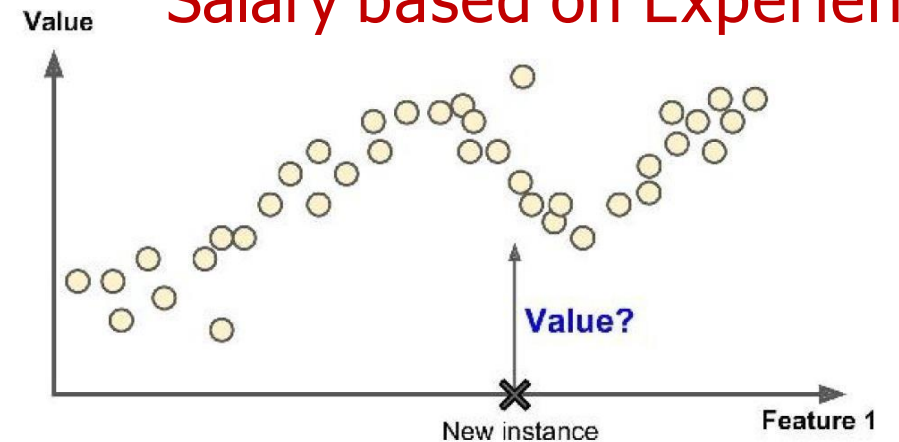


Salary (Continuous)  
Experience of person  
A linear/polynomial function that fits data samples

## Spam/Non-Spam Mails

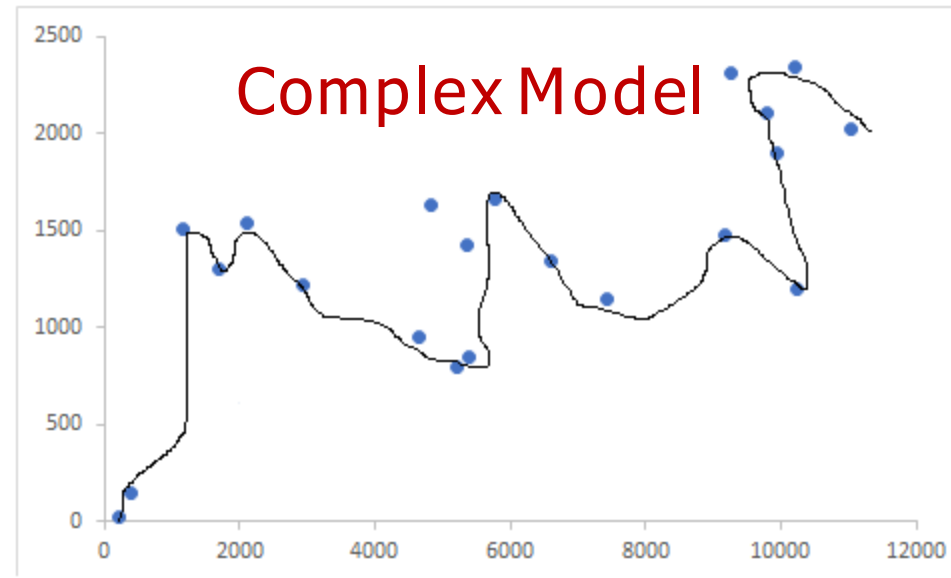
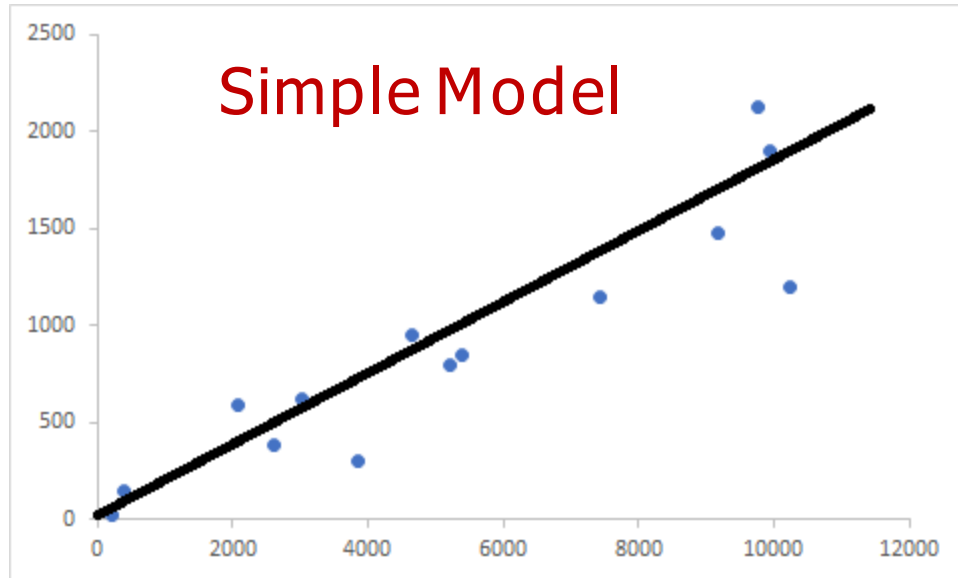


## Salary based on Experience



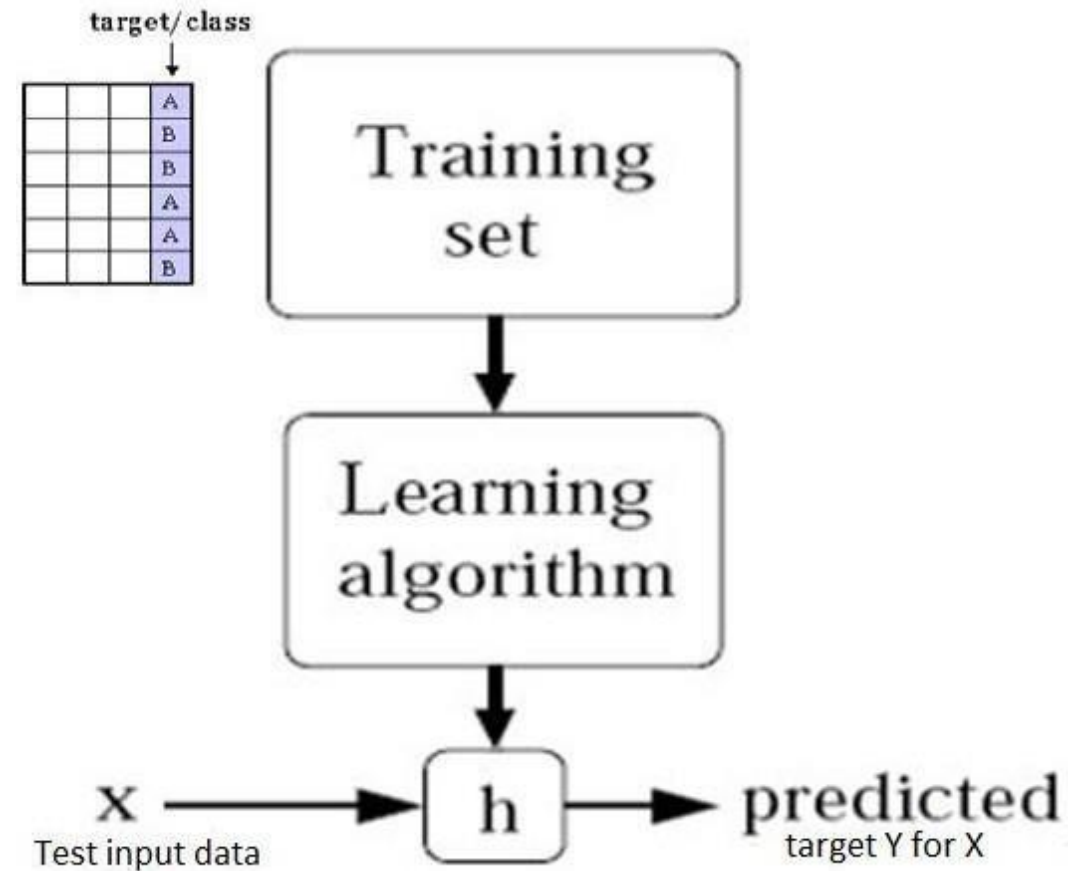
# Model Complexity and Dataset Size

- Model complexity related to the variation of inputs in training data set.
- Larger variety of data points → More complex models without overfitting.
- Can collect more data in real world problems.



# Supervised Learning Algorithms

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks Algorithms



# Takeaways

- Target, Features, and Model in supervised learning.
- Classification and Regression Tasks.
- Important supervised learning algorithms.

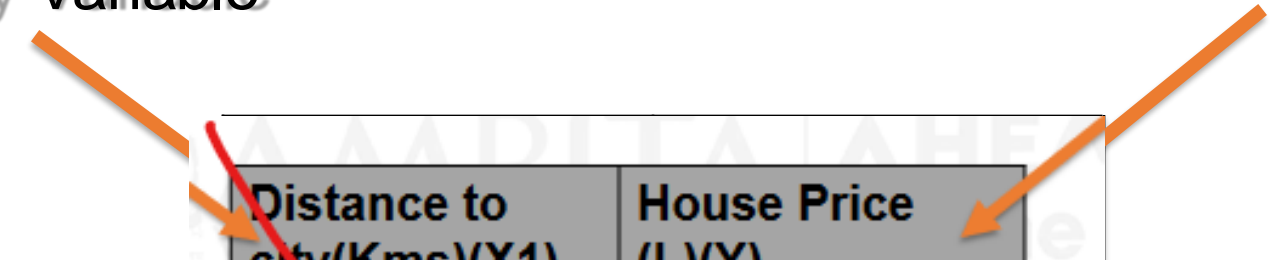
# Objectives

- Dependent Variable and Independent Variable.
- Variants of Linear Regression
- Model Representation
- Cost(Optimization) Function for Regression

# Dependent/Independent Variables

Independent/  
Explanatory Variable

Dependent /  
Response Variable

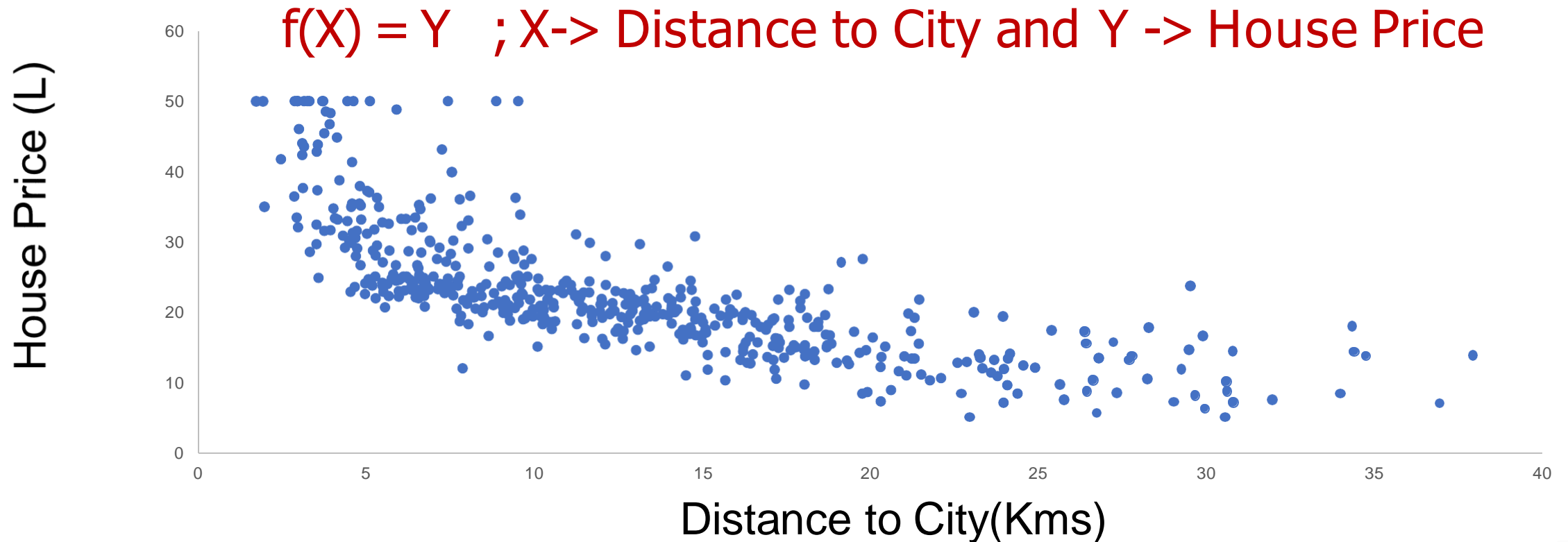


Distance to city(Kms)(X1)	House Price (L)(Y)
4.98	24
9.14	21.6
4.03	34.7
2.94	33.4
2.33	36.2

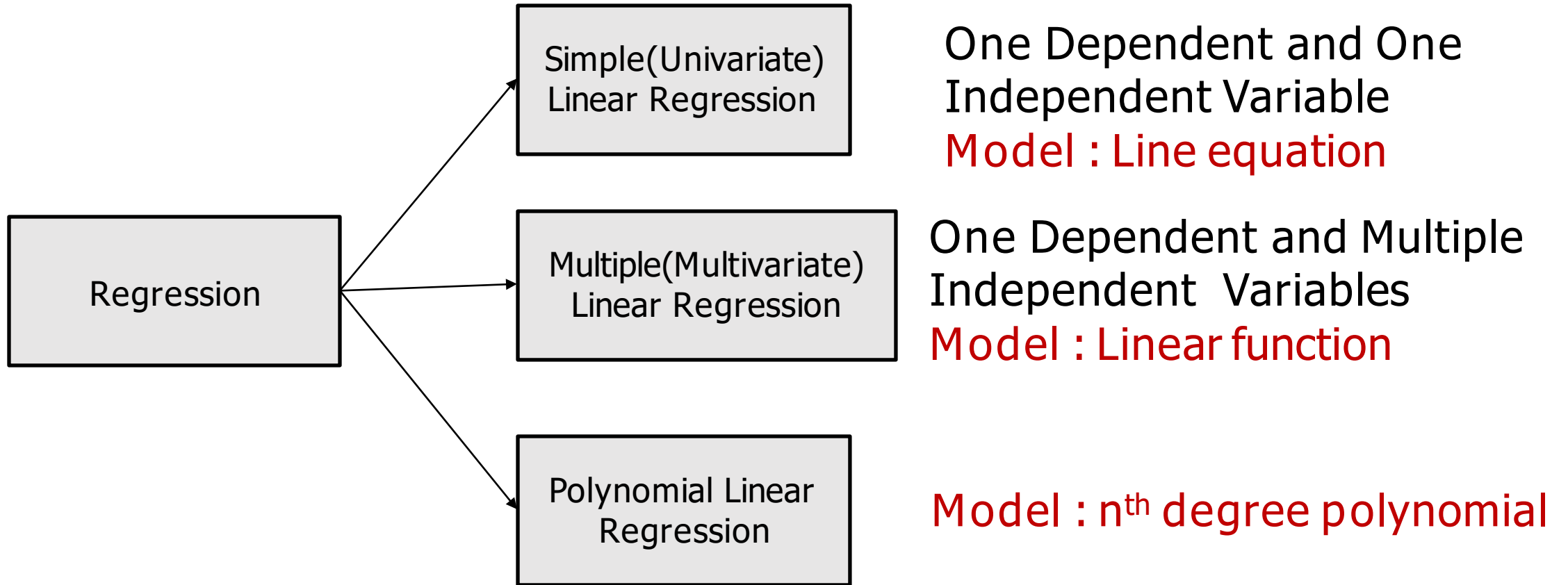


# What is Regression ?

- Statistical method to find relationship of a Dependent variable with Independent variables



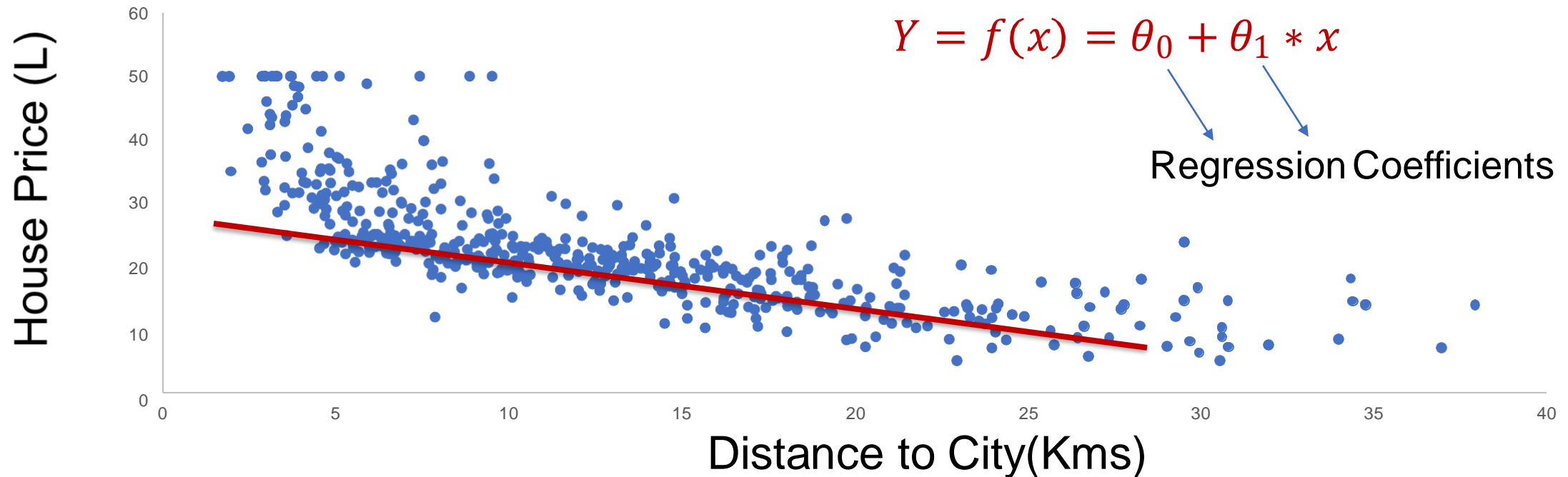
# Regression



# Simple(Univariate) Linear Regression

- Linear Regression produces a linear model
- Simplest form : One dependent and one independent variable.

Linear Model :  $y = mx + c$  (Line equation)



$$Y = f(x) = \theta_0 + \theta_1 * x$$

# Multiple(Multivariate) Linear Regression

- Y depends on more than one independent variables.
- Linear Function :  $f(X) = f(x_1, x_2, \dots, x_p) = Y$

X1	X2	X3	X4	X5	House Price(Y)
0.00632	2.31	0.538	6.575	4.98	24
0.02731	7.07	0.469	6.421	9.14	21.6
0.02729	7.07	0.469	7.185	4.03	34.7
0.03237	2.18	0.458	6.998	2.94	33.4
0.06905	2.18	0.458	7.147	5.33	36.2
0.02985	2.18	0.458	6.43	5.21	28.7
0.08829	7.87	0.524	6.012	12.43	22.9

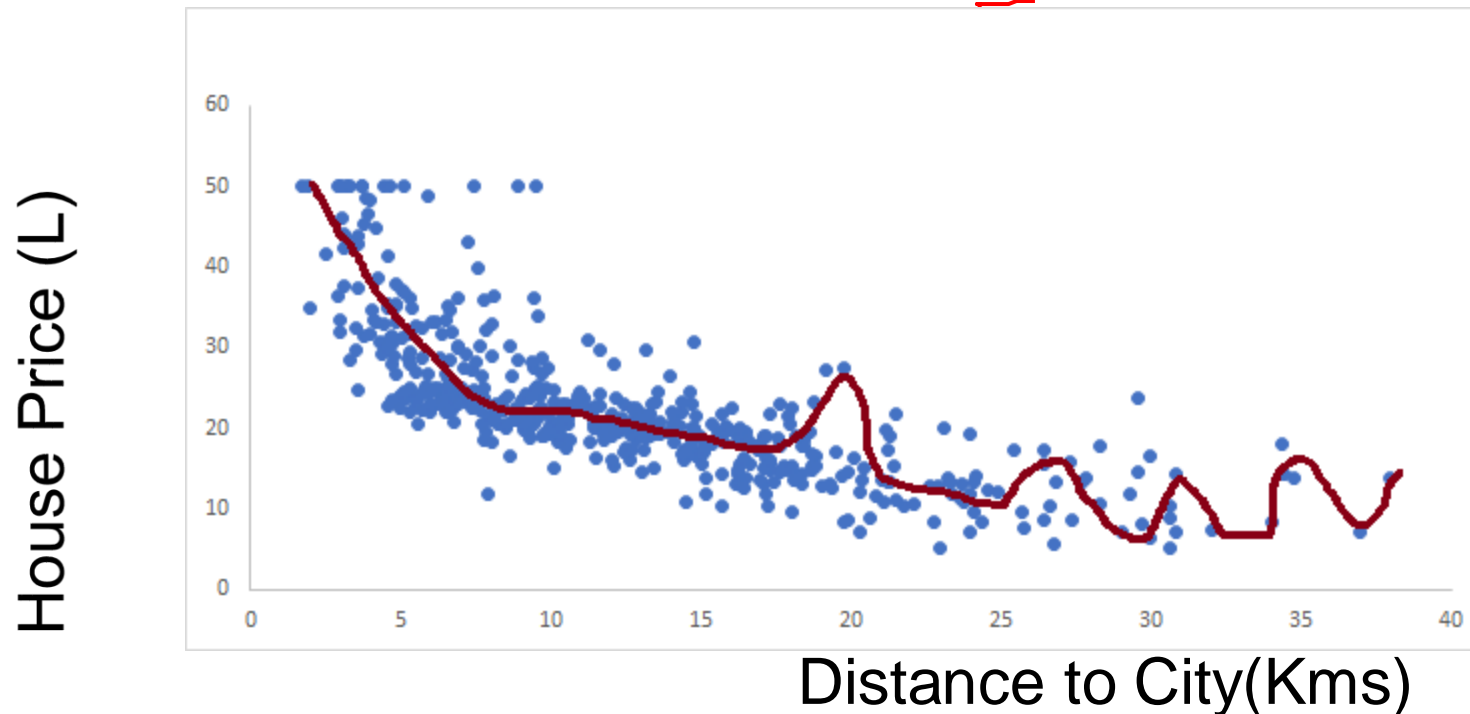
p = number of features  
in the dataset except  
target feature Y

$$Y = f(X) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 \dots \theta_p * x_p$$

# Polynomial Regression

- Relationship modelled as an  $n^{\text{th}}$  degree polynomial.
- Allows for non-linear relationship
- Still considered Linear as its linear in the regression coefficients.

$$Y = f(X) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2^2 \dots \theta_p * x_p^n$$



# Model / Hypothesis Representation

- Univariate Linear Regression Hypothesis

$$y = \theta_0 + \theta_1 * x_1$$

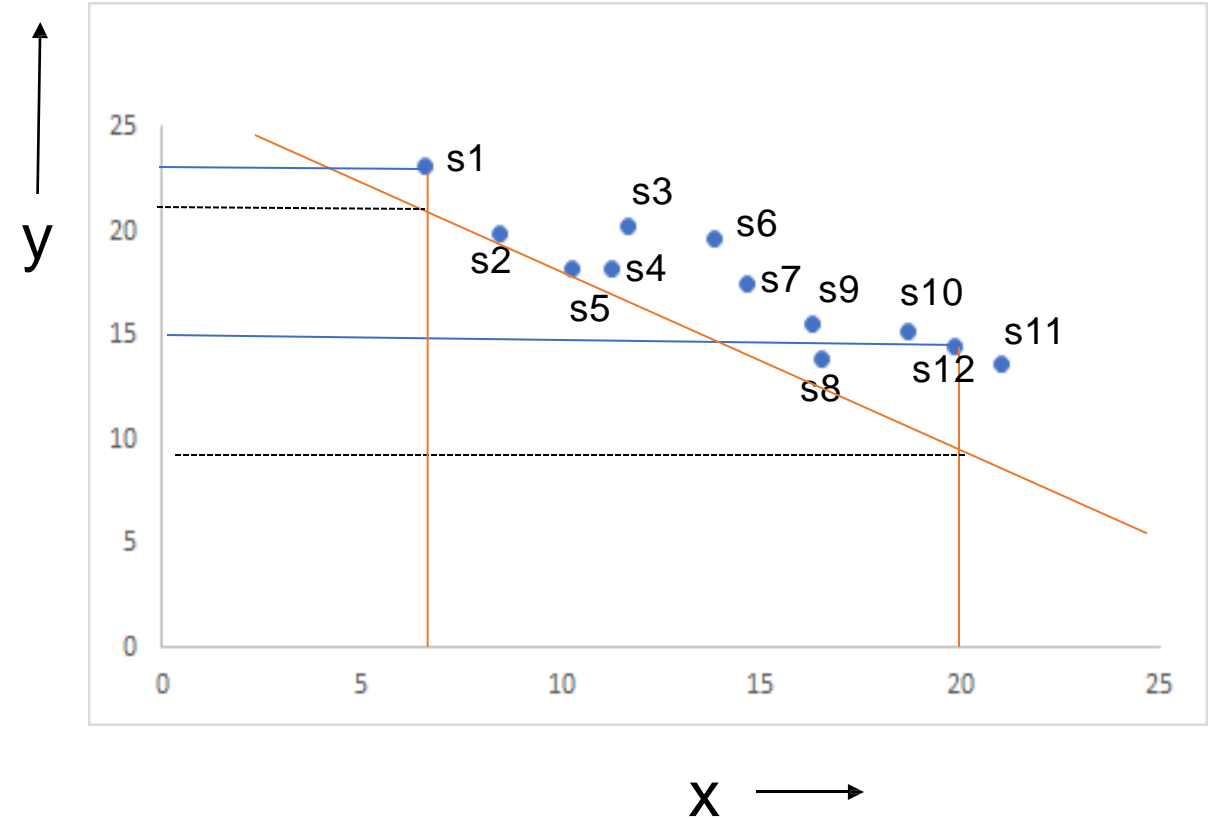
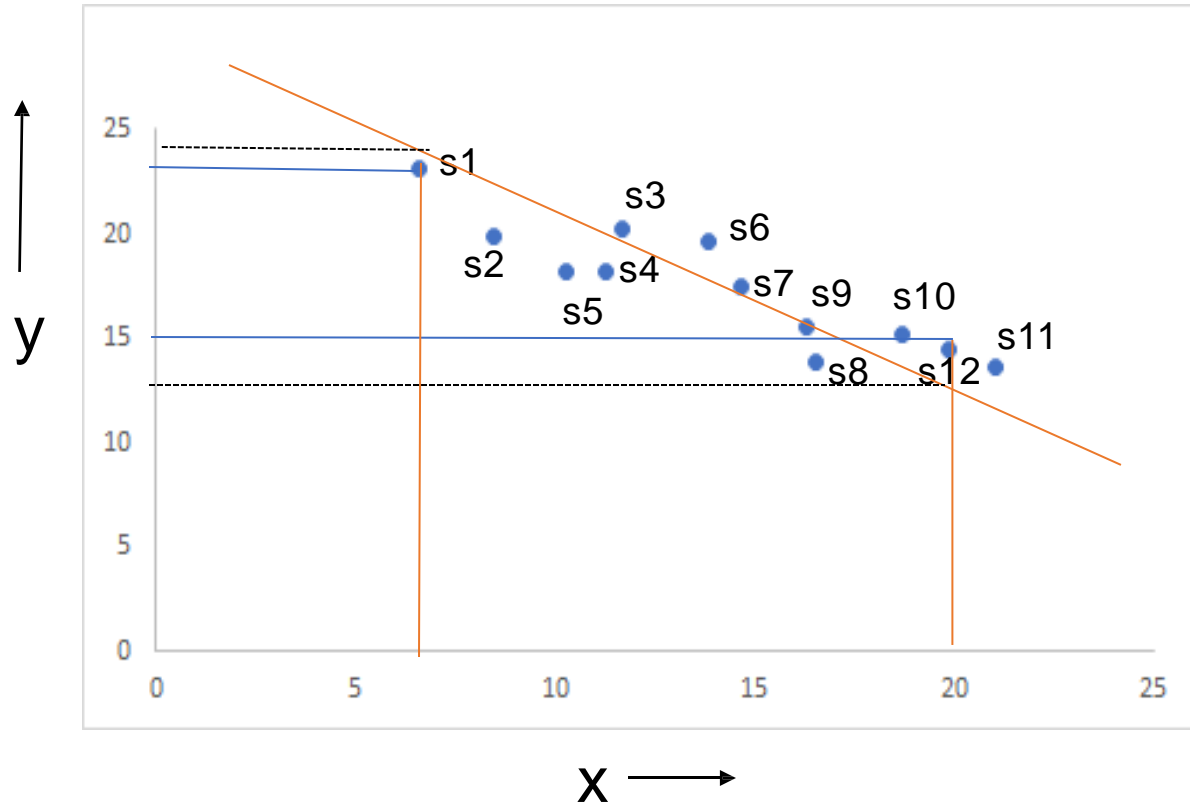
- Multivariate Linear Regression Hypothesis

$$y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \cdots + \theta_p * x_p = \sum_{j=0}^p \theta_j * x_j$$

Assume  $x_0=0$

where  $x_j$  is feature and  $\theta_j$  is model parameter

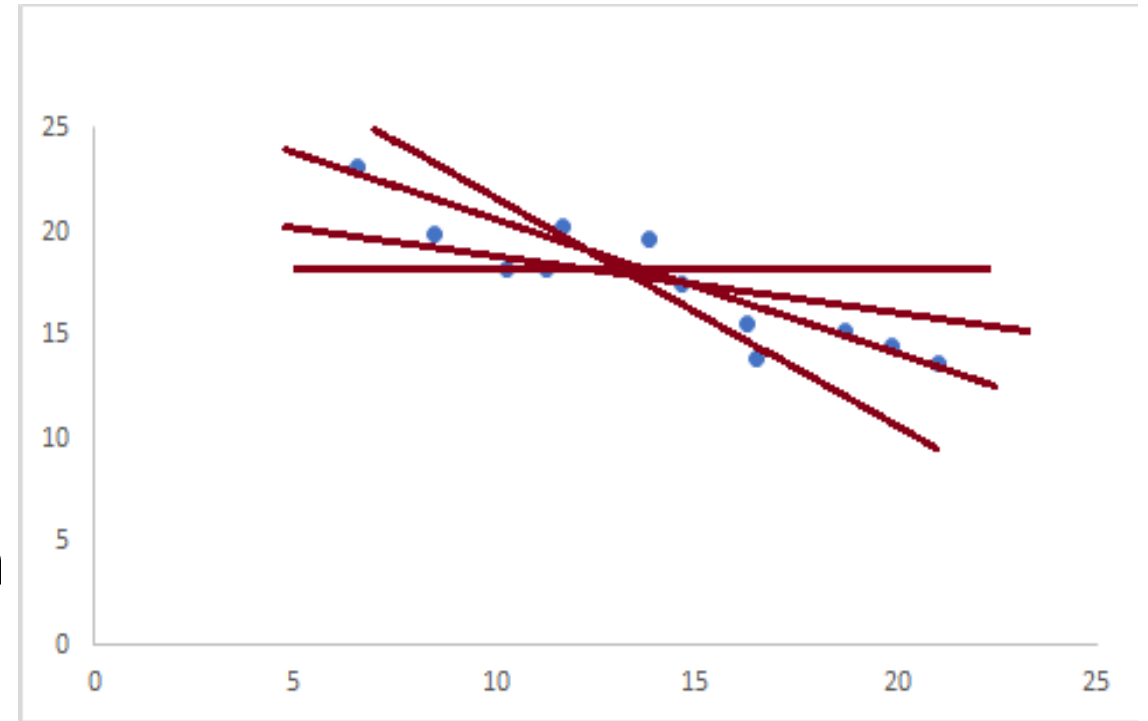
# Optimal / Best Fit



$$F(x) = y = mx + c$$

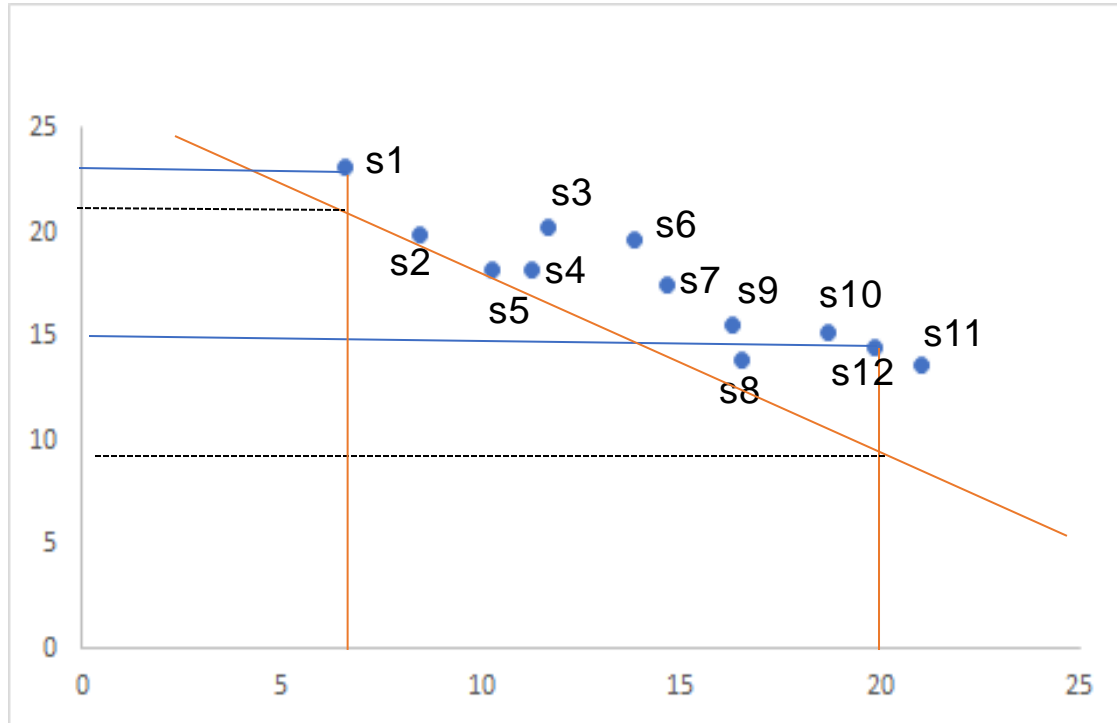
# How does Linear Regression work ?

- Find best fitting line
- For best fit line, error between the predicted and the observed values is minimum
- Define error function as cost function
- Learning algorithm find model parameter  $\theta$  such that it minimize the cost.





# Cost Function



x

$$e_i = h_{\theta}(x_i) - y_i$$

Residual Sum of Squares

$$RSS = e_1^2 + e_2^2 + \dots + e_m^2; m = \text{number of samples}$$

Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y)^2$$

Minimize  $J(\theta)$

# Takeaways

- Dependent and independent variables
- Variants of Linear Regression
- Best Fit
- Hypothesis  $h_{\theta}(x_i)$
- Cost Function  $J(\theta)$

# Objective s

- Ordinary Least Square Estimation.
- The Linear Algebra Operations to find optimal parameters
- OLS algorithm
- Computational Cost

# Hypothesis for Linear Regression

- Data :  $\langle x_1, x_2, x_3, \dots, x_p, y \rangle$  :  $y \rightarrow$  Target feature
- Hypothesis :  $p \rightarrow$  no. of features;  $m \rightarrow$  no. of samples

$$h_{\theta}(X) = y = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \dots + \theta_p * x_p$$
$$= \sum_{j=0}^p \theta_j * x_j \quad \text{Assume } x_0=0$$

In Vector Form :

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} [x_0 \ x_1 \ \dots \ x_p] = \theta^T X = h(X)$$

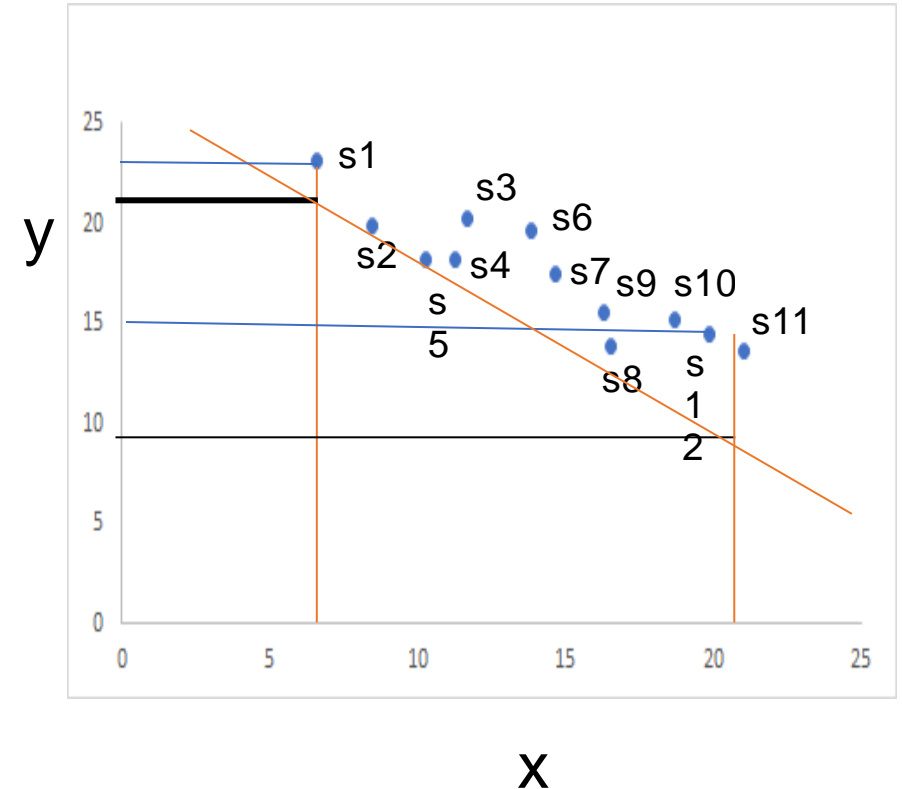
# Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y)^2$$

Residuals/Error

Distance between the best fit  
and actual values

- Best fit by Solving  $J(\theta)$  ;  
Minimize  $J(\theta)$



# Derivative of Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$
$$(X\theta - Y)^T (X\theta - Y)$$

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} [(X\theta - Y)^T (X\theta - Y)]$$
$$= 2X^T X \theta - 2X^T Y$$

$$Cost' \theta = 0$$

$$2X^T X \theta - 2X^T Y = 0$$

$$2X^T X \theta = 2X^T Y$$

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

$$X = \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ & \vdots & \\ - & (x^{(m)})^T & - \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

# Ordinary Least Square(OLS) Estimation

## OLS Linear Regression Algorithm:

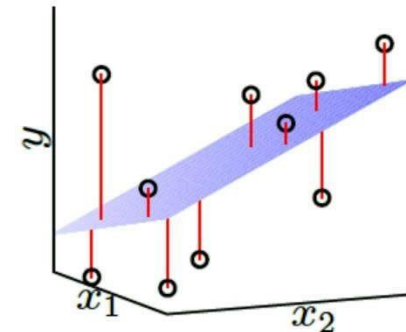
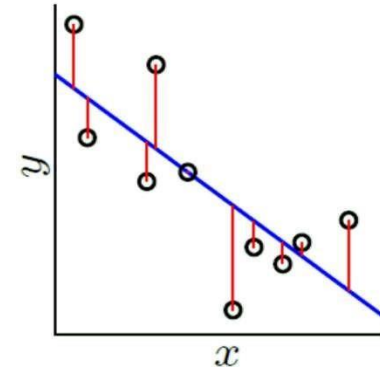
1. From the training data set, construct the input matrix  $\mathbf{X}$  and the output vector  $\mathbf{Y}$

2. Assuming  $\mathbf{X}^T\mathbf{X}$  is invertible (positive definite and non-singular),

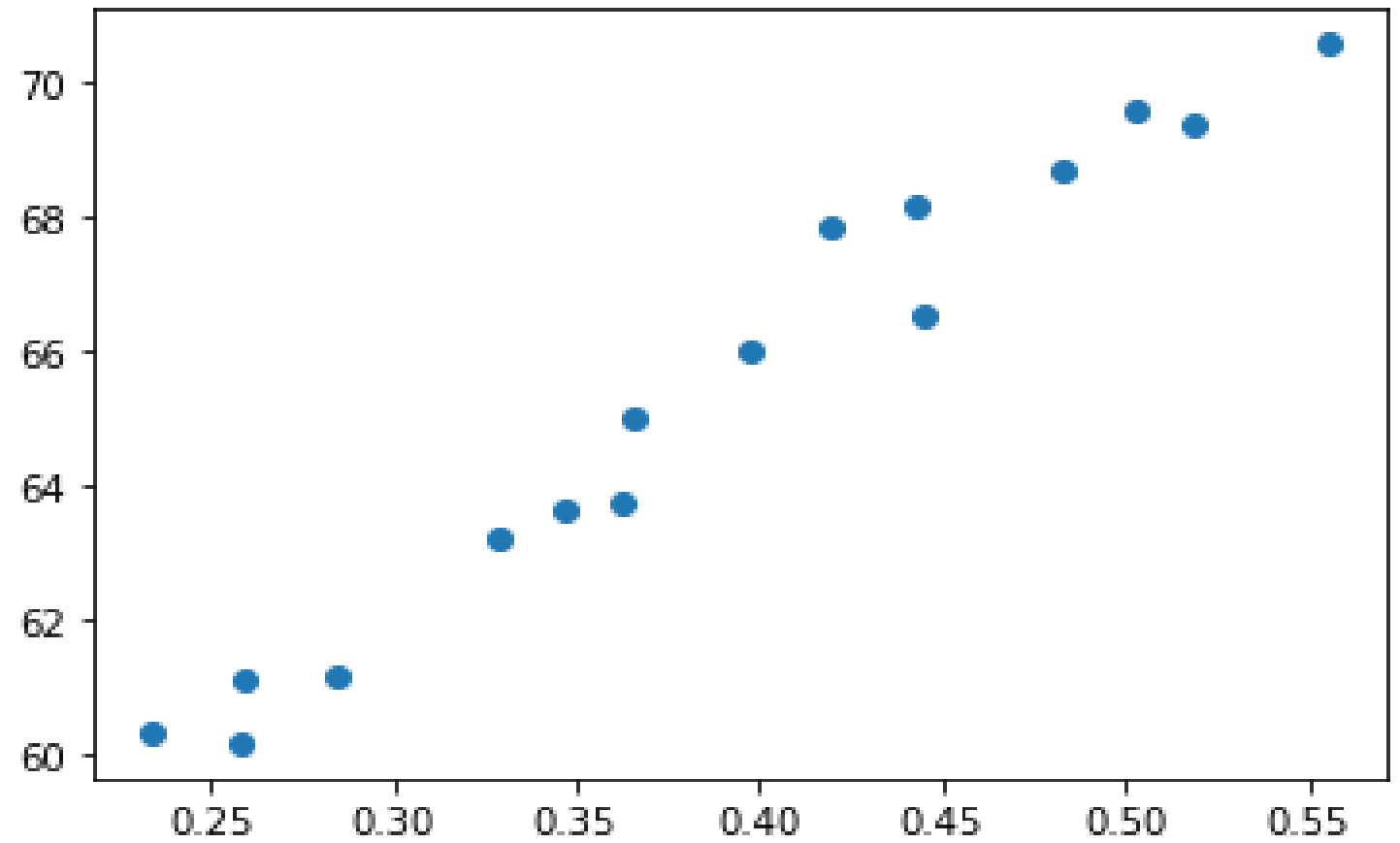
Compute

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

3. Return  $\theta = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{Y})$



# Exemplar





# Example

- $X^T X$

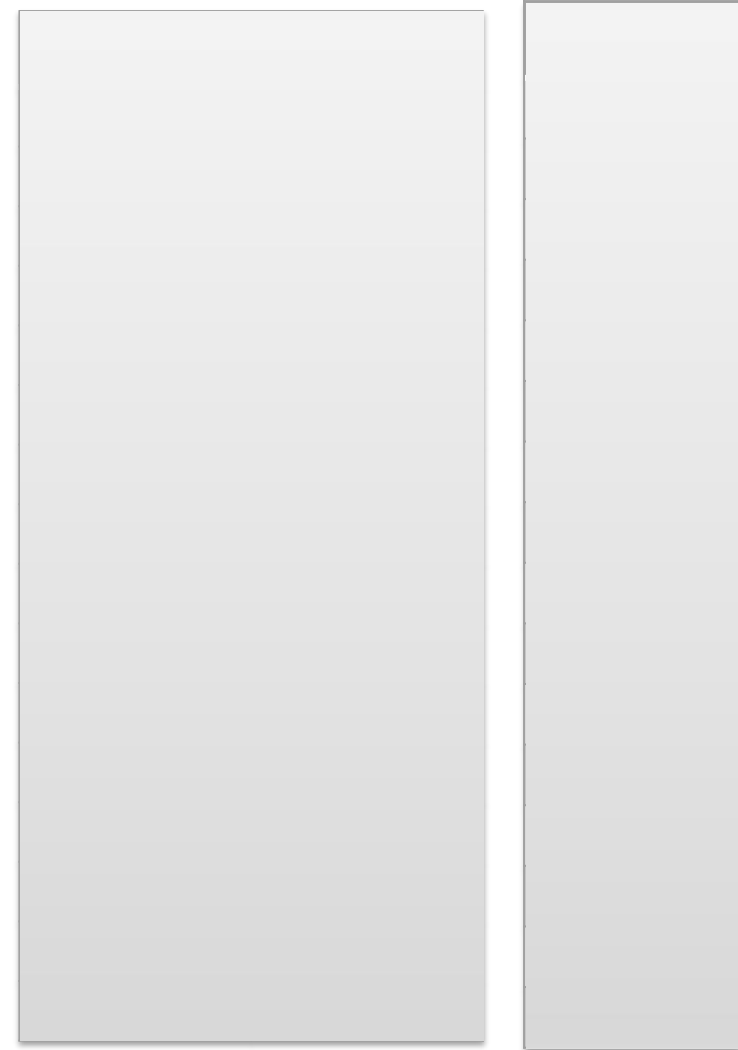
$$\begin{bmatrix} 2.55315156 & 6.203175 \\ 6.203175 & 16. \end{bmatrix}$$

- $X^T Y$

$$[1045.072 \quad 410.32273457]$$

- $(X^T X)^{-1} (X^T Y)$

$$[51.84358978 \quad 34.75229435]$$



# Example

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

[51.84358978 34.75229435]

$$\theta_0 = 51.844$$

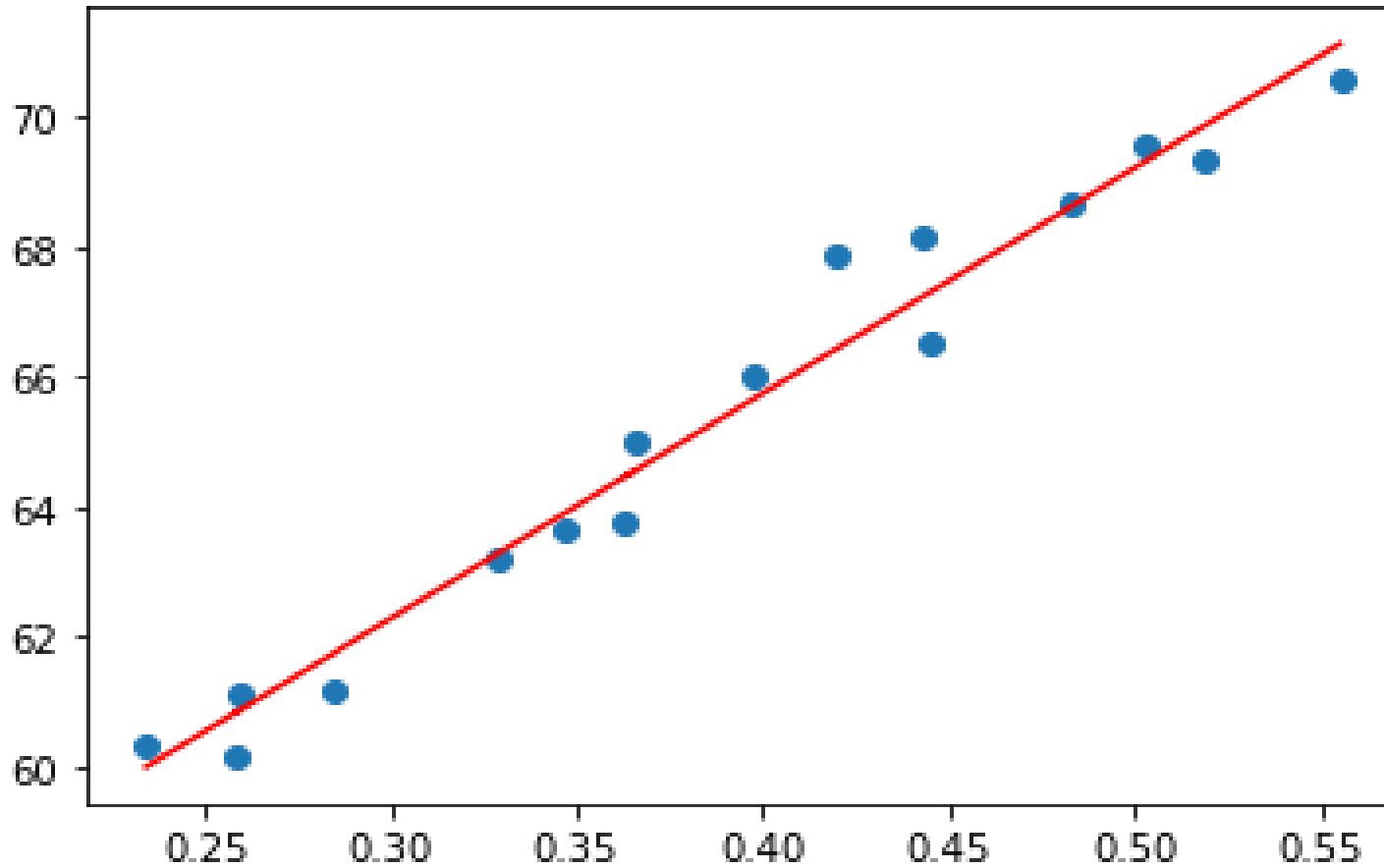
$$\theta_1 = 34.75$$

$$y' = \theta_0 + \theta_1 x$$

Hypothesis  
 $h(x)$

# Examl

- *Best Fit Line Equation* =  $51.844 + 34.75 * x$



# Computational cost of OLS

- What operations are necessary?  
Overall: 1 matrix inversion + 3 matrix multiplications
- What if  $X$  is too big to compute this explicitly (e.g.  $m \sim 10^6$ )?
- If  $X$  is reasonably small, Can we always evaluate Inverse ?  
 $X^T X$  should be nonsingular

# Takeaways

- Residual/Error in Linear regression line
- Cost function and its derivative
- Ordinary Least square Method
- Example of Best Fit

# Objective s

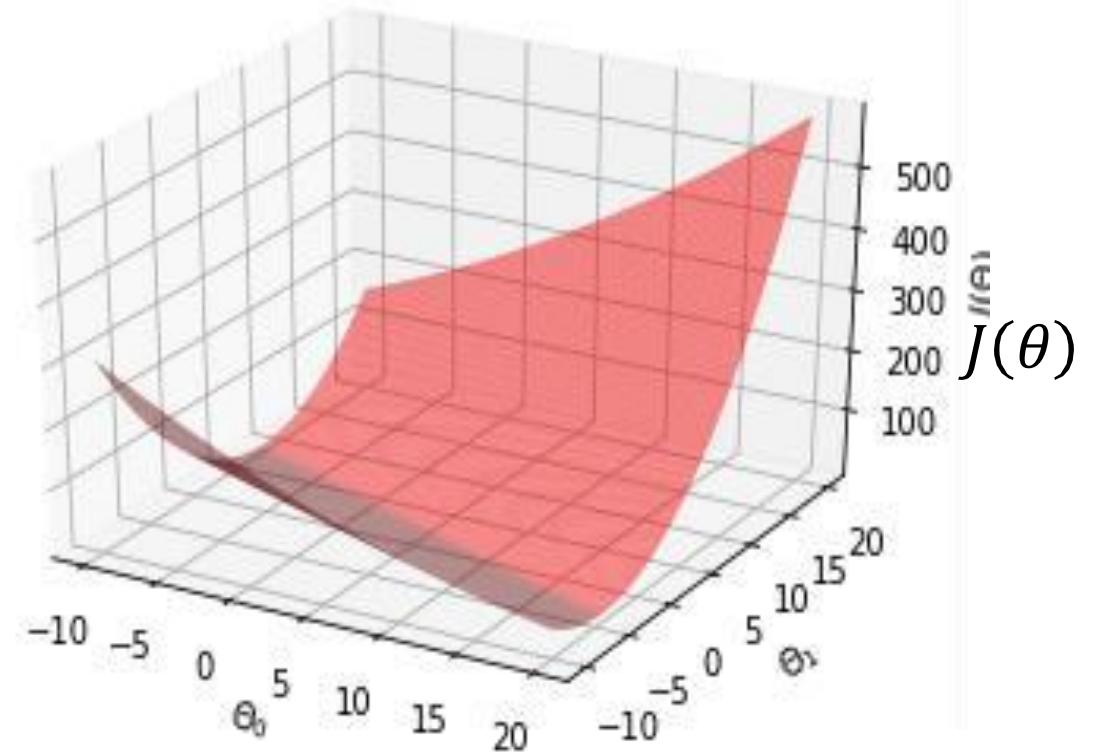
- Gradient descent
- Linear Regression Algorithm

# Optimization Objective of Linear Regression

Cost Function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y)$$

- Best fit by Solving  $J(\theta)$  ;  
Minimize  $J(\theta)$



# Search Strategy

- Choose the initial value of parameter  $\theta$ .
- Until we reach a minimum  
Choose new value for  $\theta$   
to reduce  $J(\theta)$

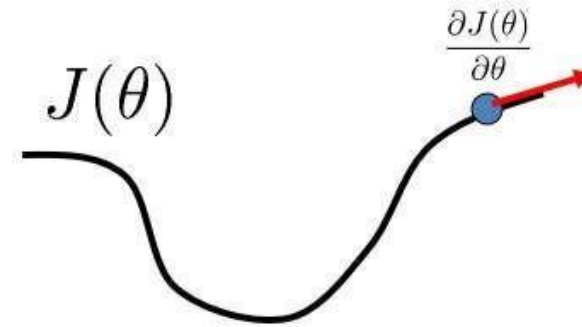
Choose values for  $\theta$  such that  $J(\theta)$   
reduces during each iteration





# Direction of Search

- Gradient (Derivative) is the rate of change of a function
- It's a vector that
  - Points in the direction of greatest increase of a function
  - Is zero at a local maximum or local minimum (because there is no single direction of increase)



- Learning rate :  
Step size taken to decrease  
the function value

**Negative Gradient!!**

- Choose a direction in which  $J(\theta)$  is decreasing
- Derivative  $\frac{\partial J(\theta)}{\partial \theta}$ 
  - Positive  $\Rightarrow$  increasing
  - Negative  $\Rightarrow$  decreasing

# Gradient of Cost Function

$$\begin{aligned}
 J(\theta) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (y_i - h_{\theta}(x_i))^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (y_i - (\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_p x_{ip}))^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (e_i(\theta))^2
 \end{aligned}$$

$$\begin{aligned}
 \frac{\partial J}{\partial \theta_0} &= \frac{1}{2m} \sum_i 2e_i(\theta) \frac{\partial}{\partial \theta_0} e_i(\theta) \\
 &= \frac{1}{m} \sum_i (h_{\theta}(x_i) - y_i) x_{i0}
 \end{aligned}$$

$$\begin{aligned}
 \text{So, } \frac{\partial J}{\partial \theta_j} &= -\frac{1}{2m} \sum_i 2e_i(\theta) \frac{\partial}{\partial \theta_j} e_i(\theta) \\
 &= -\frac{1}{m} \sum_i (h_{\theta}(x_i) - y_i) x_{ij}
 \end{aligned}$$

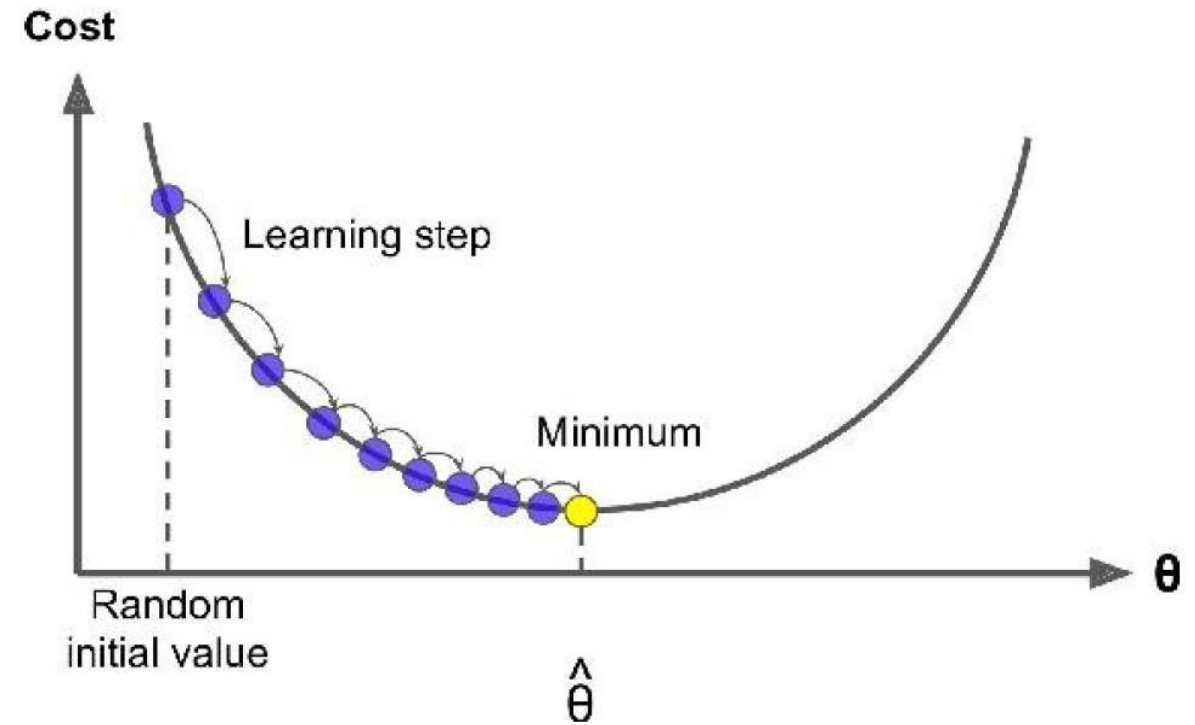
$$\begin{aligned}
 \frac{\partial}{\partial \theta_0} e_i(\theta) &= \frac{\partial}{\partial \theta_0} y_i - \left( \frac{\partial}{\partial \theta_0} \theta_0 x_{i0} + \frac{\partial}{\partial \theta_0} \theta_1 x_{i1} + \frac{\partial}{\partial \theta_0} \theta_2 x_{i2} + \dots + \frac{\partial}{\partial \theta_0} \theta_p x_{ip} \right) \\
 &= -x_{i0}
 \end{aligned}$$

$$\begin{aligned}
 J(\theta) &= \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (y_i - (\theta_0 x_{i0} + \theta_1 x_{i1} + \theta_2 x_{i2} + \dots + \theta_p x_{ip}))^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (e_i(\theta))^2
 \end{aligned}$$

# Gradient Descent

- Optimization algorithm
- Idea : Tweak parameters iteratively in order to minimize cost function.

$\alpha \frac{\partial J(\theta)}{\partial \theta}$  ;  $\alpha$  is the learning rate/step



# Gradient Descent Algorithm

*Input : Dataset :*

$\langle x_1, x_2, \dots, x_p, y \rangle$

*Learning rate :  $\alpha$*

1. Initialize the model parameters  $(\theta_0, \theta_1, \dots, \theta_p)$  with some random values.
2. Compute the partial derivatives of the cost function w.r.to each of the parameters  $\theta_0, \theta_1, \dots, \theta_p$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$$

3. After computing the derivative update the parameters

$$\theta_j = \theta_j - \frac{\alpha}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$$

4. Compute  $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$  with new set of parameters  $\theta$

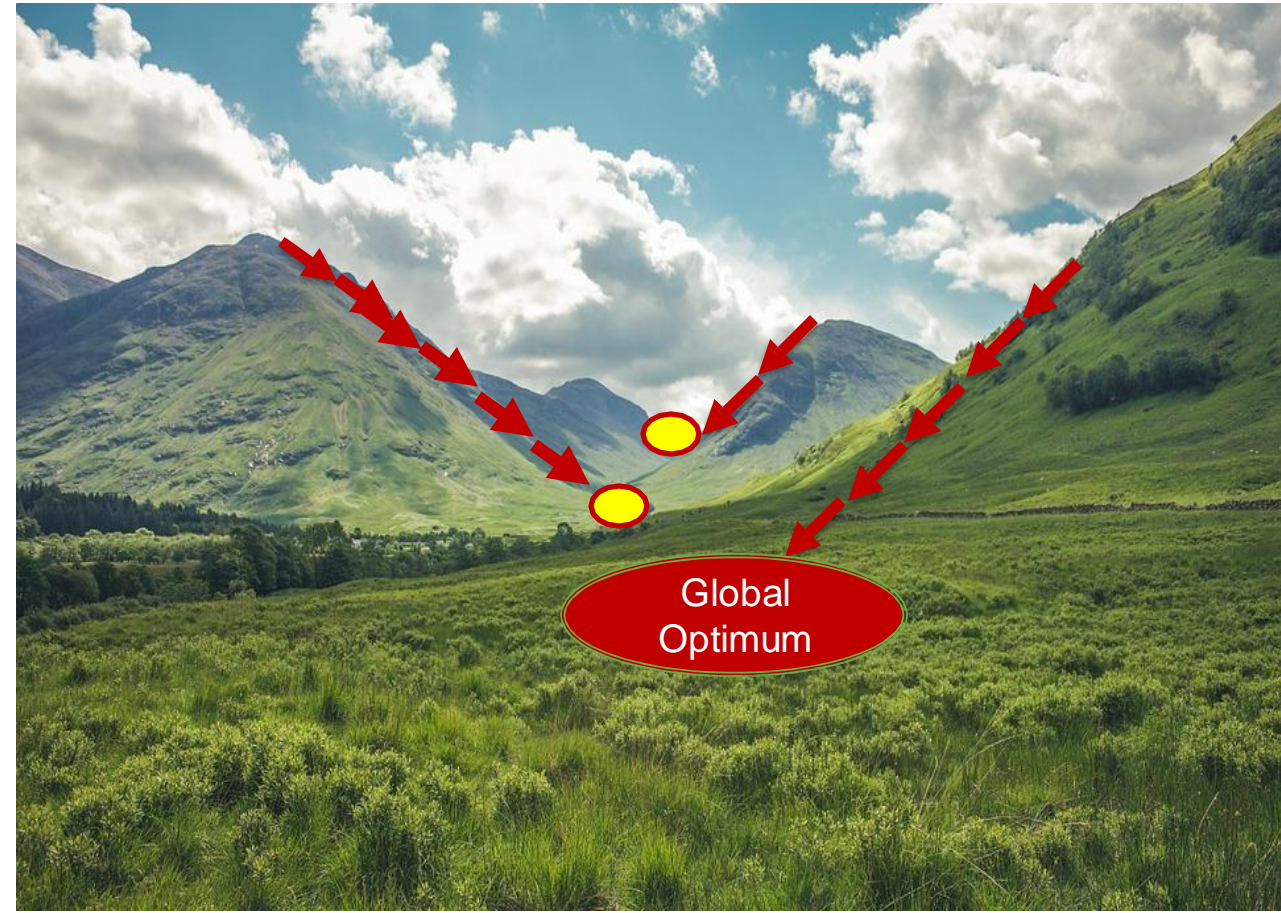
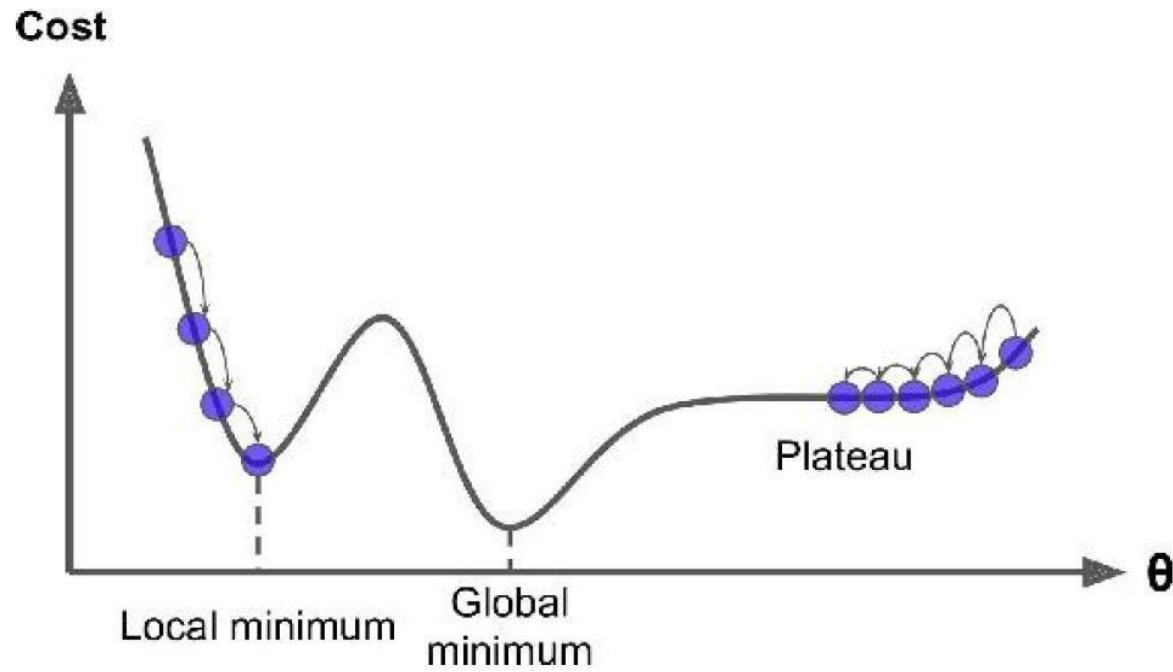
5. Repeat steps 2,3,4 until the cost function  $J(\theta)$  converges to a minimum value

*Output :  $\theta$*



# Challenge

## S



# Takeaways

- Gradient descent
- Linear Regression Algorithm
- Local optima and Global optima problem

# Objective s

- Parameter Update
- Impact of learning rate
- Gradient descent variants and their features.

# Gradient Descent Algorithm

## An Iterative Optimization Algorithm

1. Initialize  $\theta$  with random values
2. Repeat Until Convergence

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(for  $j=1,2,\dots,p$  simultaneously)

## Linear Regression Model

$$h_{\theta}(x_i) = \theta_0 + \theta_1 x_{i1} + \dots + \theta_p x_{ip}$$

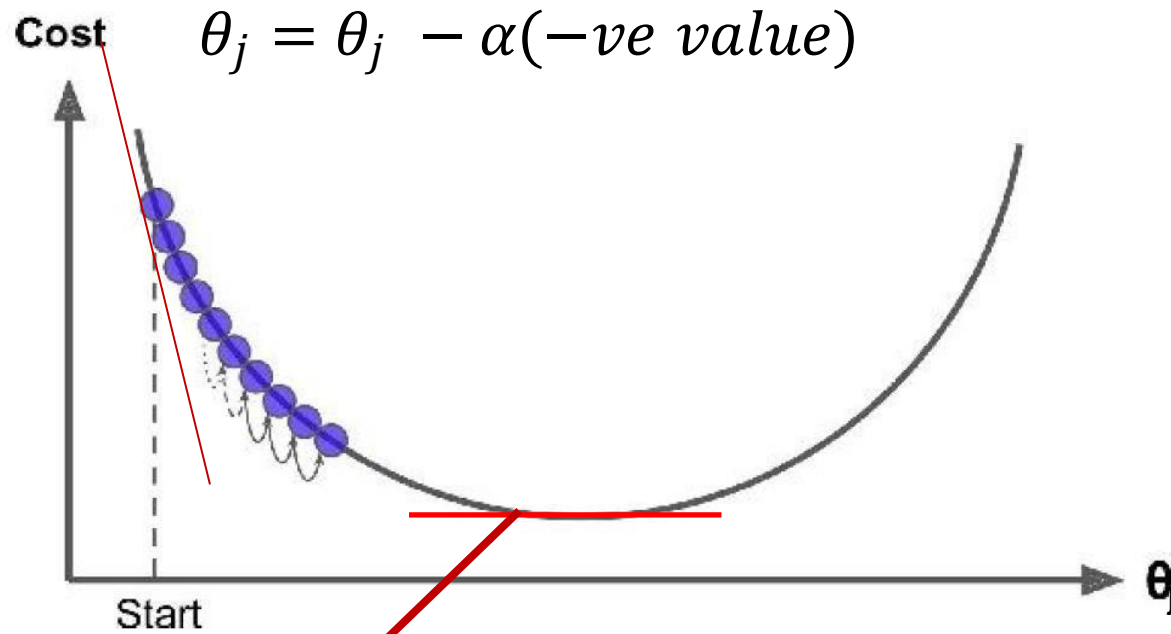
$$h_{\theta}(X) = \theta^T X$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x) - y)^2$$



# Parameter update

- Negative slope

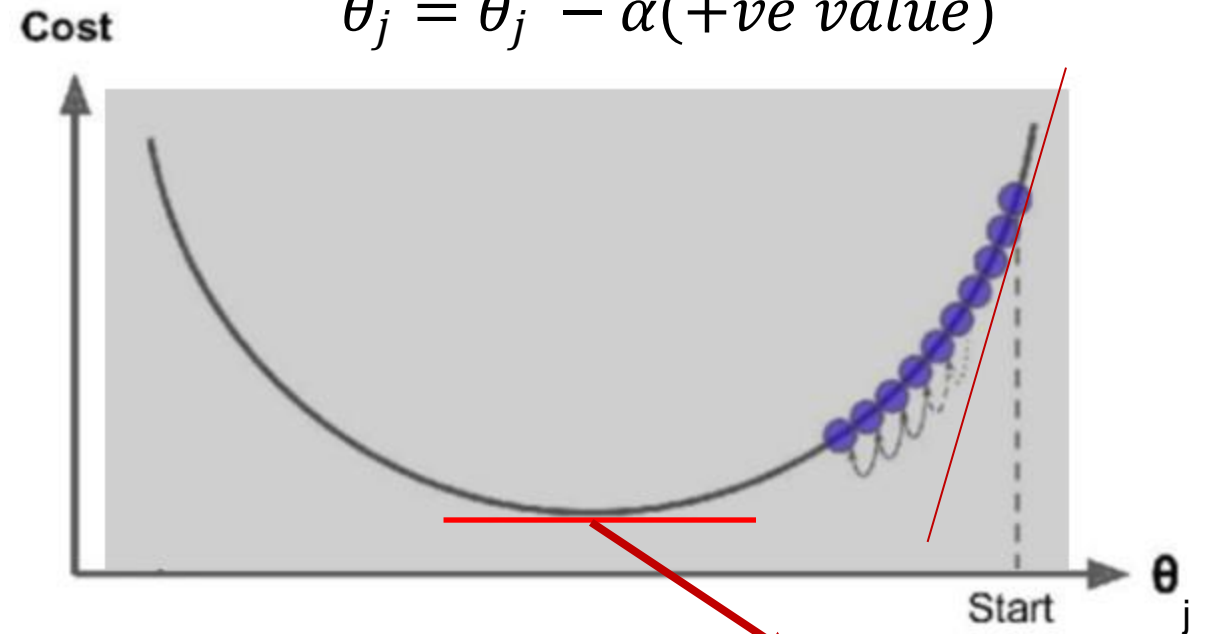


Slope = 0

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Positive slope

$\theta_j = \theta_j - \alpha(+ve \text{ value})$

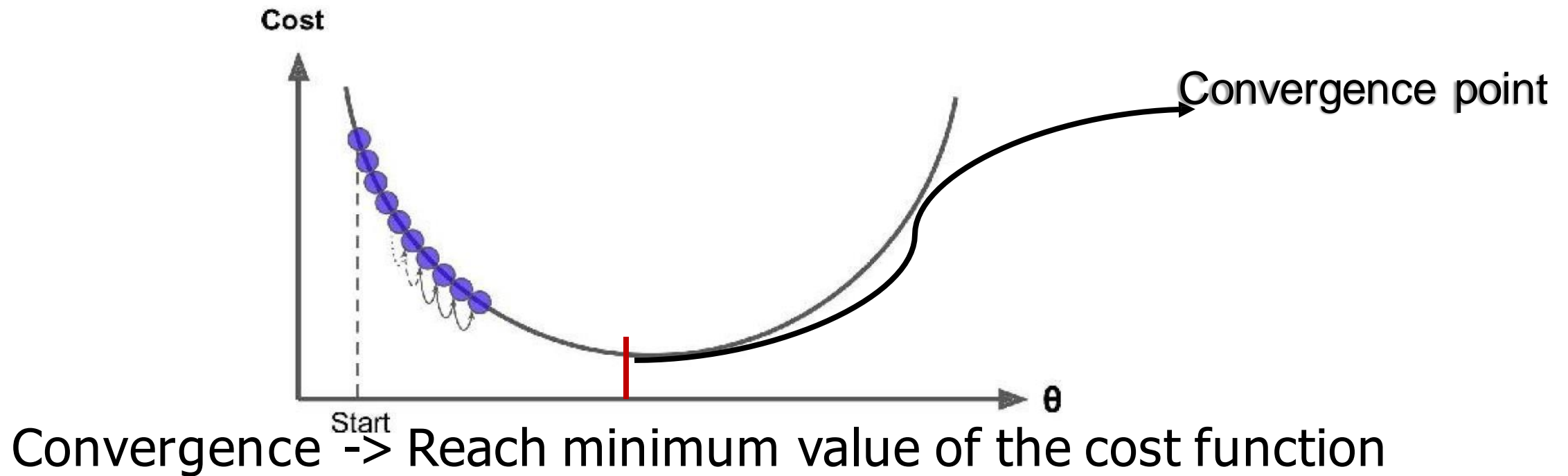


Slope = 0

General Update function

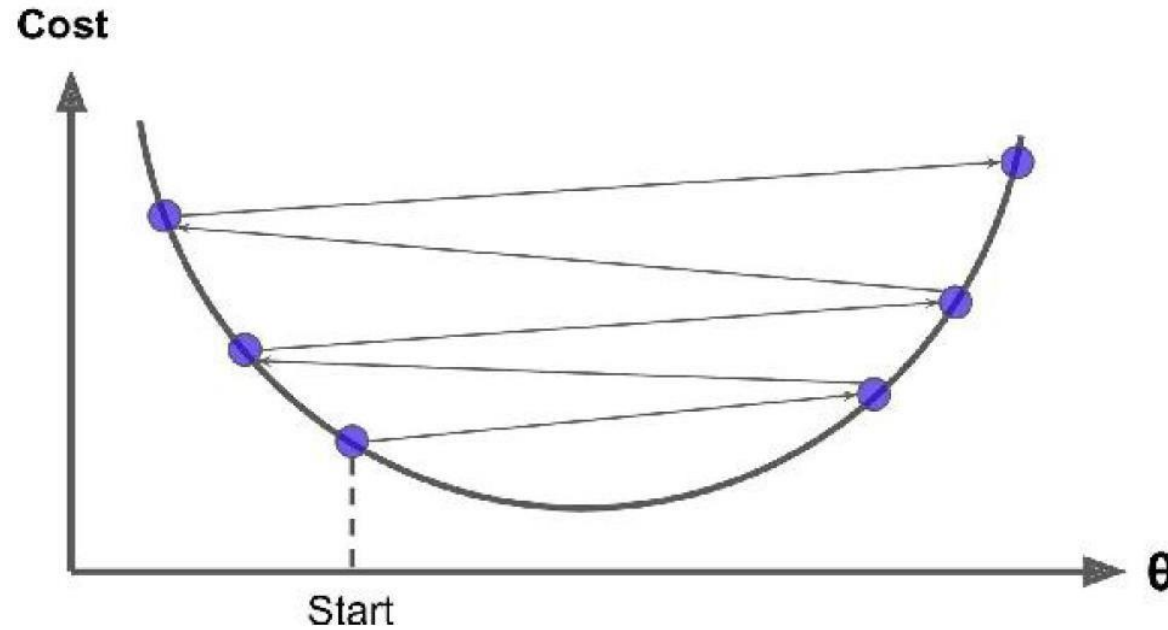
# Impact of Learning rate

- Too small learning rate  $\rightarrow$  Algorithm requires many iterations to converge.



# Impact of Learning rate

- High learning rate  $\rightarrow$  Algorithm diverges with larger values.
- Not able to find good solution.



# Gradient Descent Variants

- Batch Gradient Descent.
- Stochastic Gradient Descent.
- Mini Batch Gradient Descent.

# Batch Gradient Descent

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

- $\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$
- Calculations over the full training set  $X$  of size  $m$ , at each Gradient Descent step.
- Slow on very large training set.

# Stochastic Gradient Descent

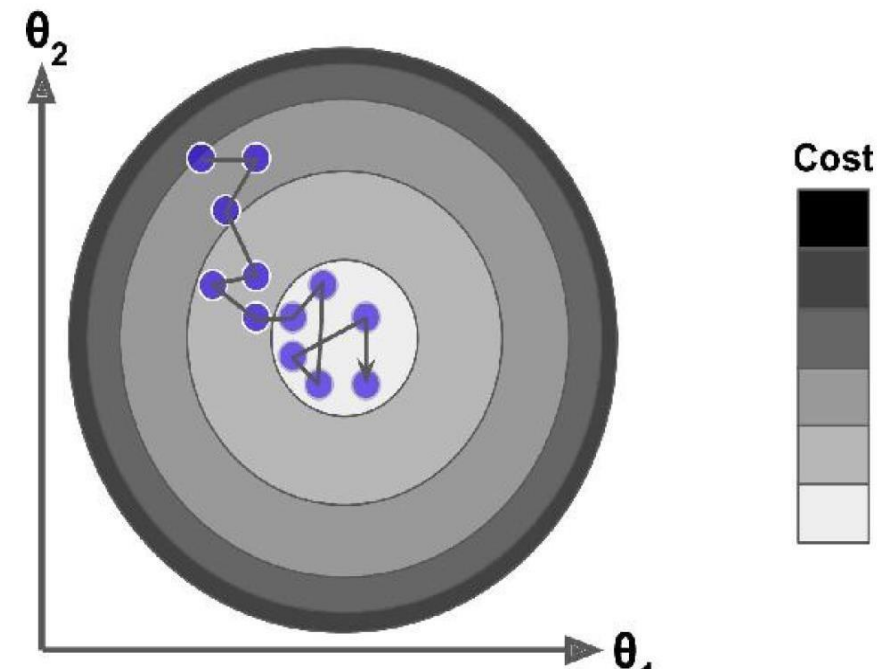
- At each step pick a random instance  $x_i$  of the training set.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

- Pick a sample  $x_i$  one by one to update the parameter

$$\theta_j = \theta_j - \alpha (h_{\theta}(x_i) - y_i) x_{ij}$$

- Faster than batch processing.
- Better chance of finding the global minimum.



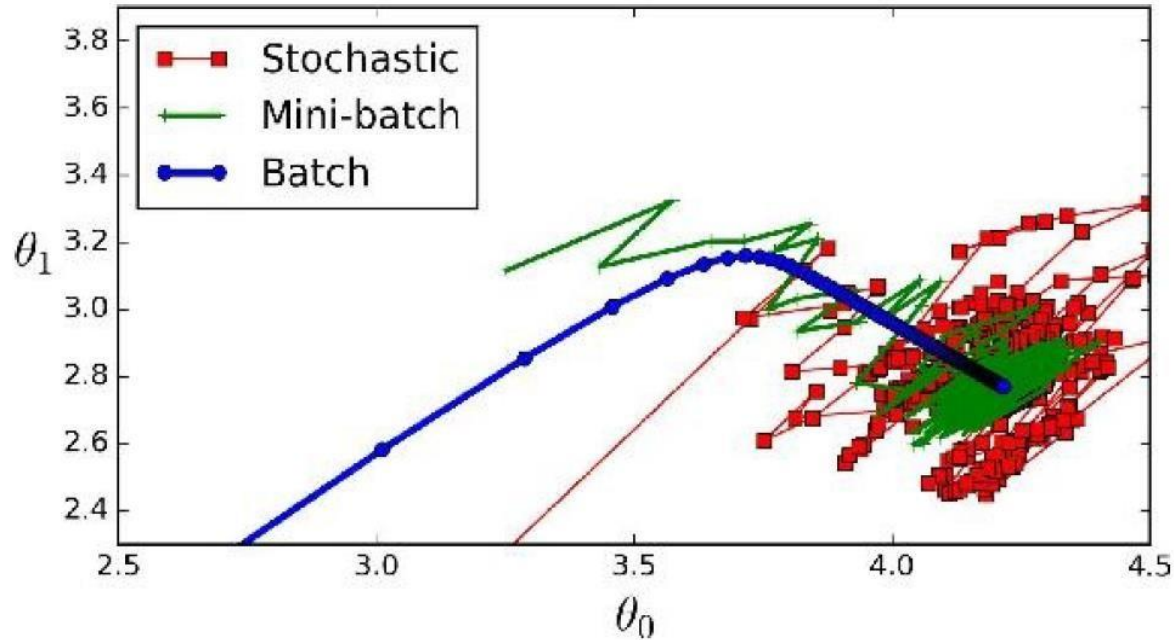
# Mini Batch Gradient Descent

- Computes the gradients on small random sets of instances called mini batches.
- Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm.

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta} J(\theta)$$

- $$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h(x_i) - y_i) x_{ij}$$

# Comparison



- In parameter space batch gradient descent stops at the minimum while stochastic and mini batch continue to walk around.



# Takeaways

- Parameter Update
- Impact of learning rate
- Variants of gradient descent.

# References

- 1 Andreas Muller, "Introduction to Machine Learning with Python: A Guide for Data Scientists", Shroff/O'Reilly, 2016.
- 2 Alexey Grigorev, Machine Learning Bookcamp, Manning, 2020.
- 3 Aurélien Geron, "Hands-On Machine Learning with Scikit-Learn and TensorFlow, Shroff/O'Reilly", 2017.