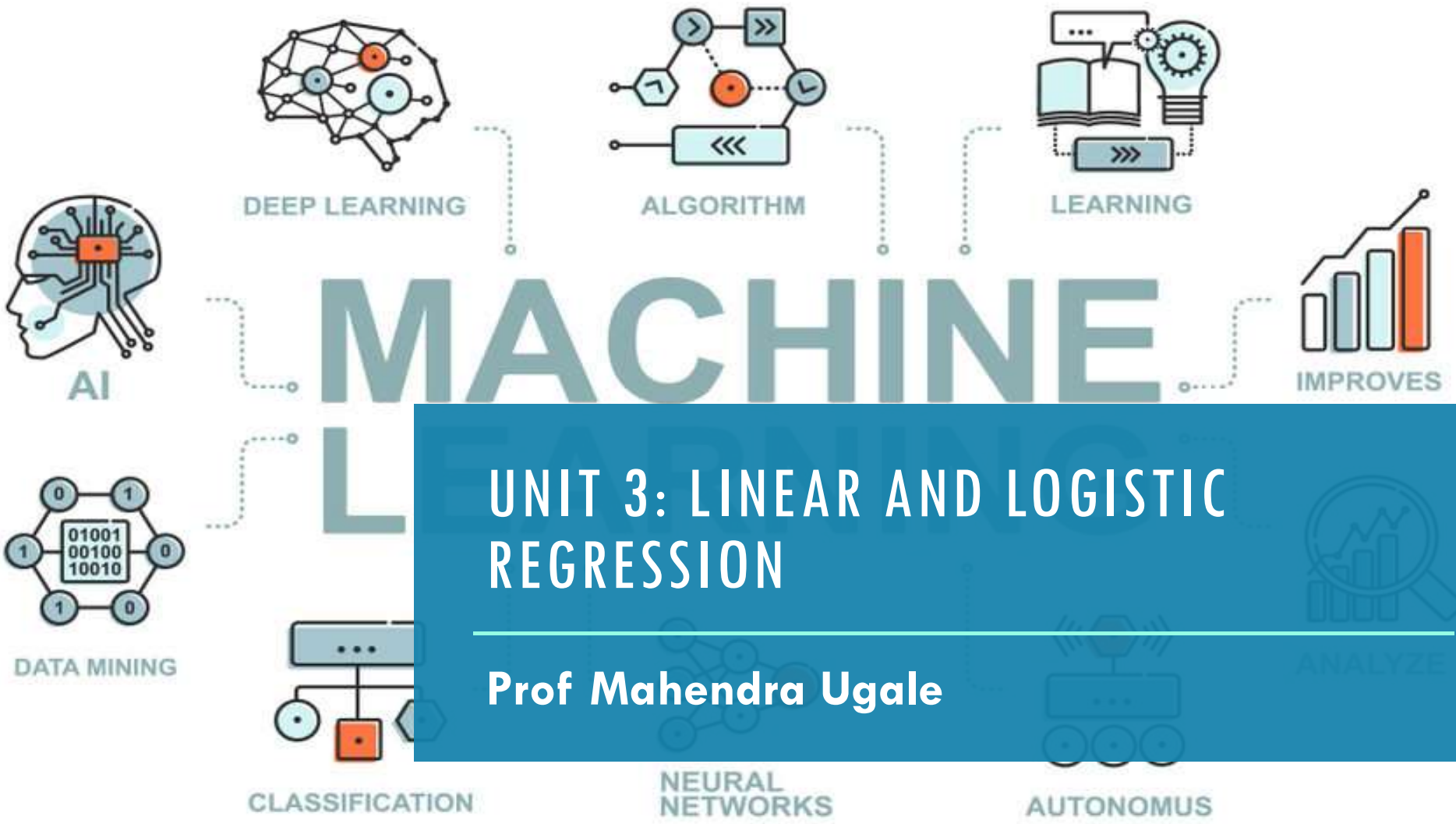



# UNIT 3: LINEAR AND LOGISTIC REGRESSION

**Prof Mahendra Ugale**

# INDEX

**Introduction to linear regression:**

- Introduction to Linear Regression,
- Optimal Coefficients,
- Cost function,
- Coefficient of Determination,
- Analysis of Linear Regression using dummy Data,
- Linear Regression Intuition.

**Multivariable regression and gradient descent:**

- Generic Gradient Descent,
- Learning Rate,
- Complexity Analysis of Normal Equation Linear Regression
- How to find More Complex Boundaries,
- Variations of Gradient Descent.

**Logistic regression:**

- Handling Classification Problems,
- Logistic Regression,
- Cost Function,
- Finding Optimal Values,
- Solving Derivatives,
- Multiclass Logistic Regression,
- Finding Complex Boundaries and Regularization,
- Using Logistic Regression from Sklearn.

# 1.INTRODUCTION TO LINEAR REGRESSION

- Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.

- More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed.

- It predicts continuous/real values such as **temperature, age, salary, price,** etc.

- We can understand the concept of regression analysis using the below example:

# 1.INTRODUCTION TO LINEAR REGRESSION

**Example:**

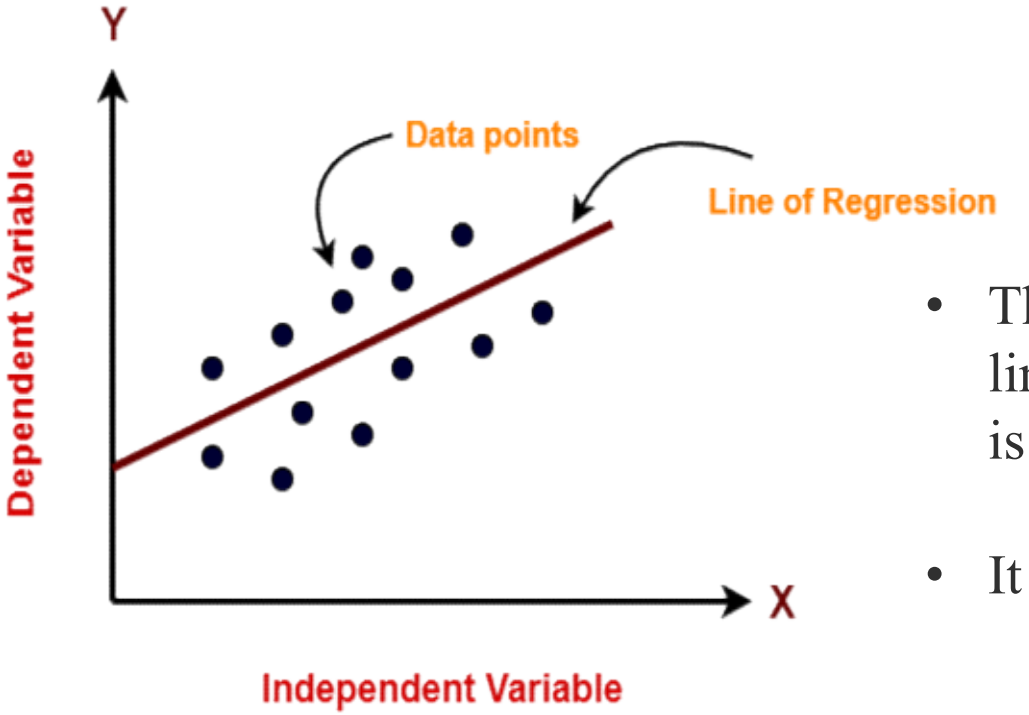
- Suppose there is a marketing company A, who does various advertisement every year and get sales on that.
- The below list shows the advertisement made by the company in the last 5 years and the corresponding sales.
- Now, the company wants to do the advertisement of $200 in the year 2019 and wants to know the prediction about the sales for this year. So to solve such type of prediction problems in machine learning, we need **regression analysis**.

| Advertisement | Sales |
|---|---|
| $90 | $1000 |
| $120 | $1300 |
| $150 | $1800 |
| $100 | $1200 |
| $130 | $1380 |
| $200 | ?? |

# 1.INTRODUCTION TO LINEAR REGRESSION

**Representing Linear Regression Model-**

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




- The sloped straight line representing the linear relationship that fits the given data best is called as a regression line.
- It is also called as best fit line.

# 1.INTRODUCTION TO LINEAR REGRESSION

- Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables**.
- In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data.
- In simple words, ***"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."***
- The distance between datapoints and line tells whether a model has captured a strong relationship or not.

# 1.INTRODUCTION TO LINEAR REGRESSION

**Some examples of regression can be as:**

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

# 1.INTRODUCTION TO LINEAR REGRESSION

**Terminologies Related to the Regression Analysis:**

- **Dependent Variable:** The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called **target variable**.

- **Independent Variable:** The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a **predictor**.

- **Outliers:** Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.

# 1.INTRODUCTION TO LINEAR REGRESSION

**Terminologies Related to the Regression Analysis:**

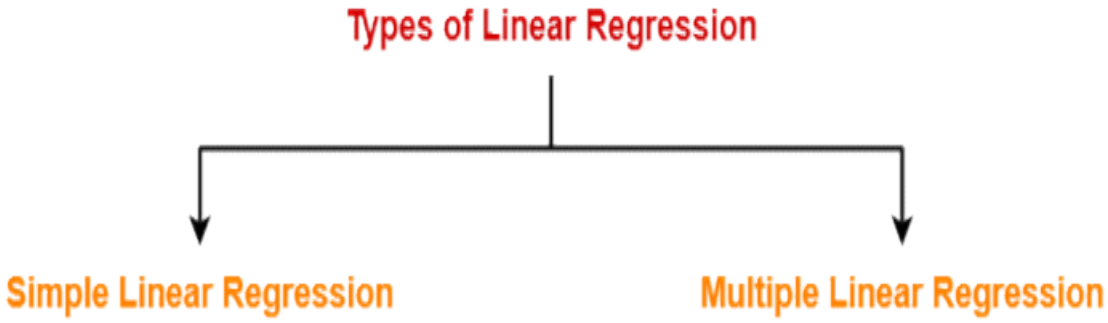
- **Multicollinearity:** If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.

- **Underfitting and Overfitting:** If our algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if our algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

# 1.INTRODUCTION TO LINEAR REGRESSION

**Types of Linear Regression-**

Based on the number of independent variables, there are two types of linear regression-




1.Simple Linear Regression
2.Multiple Linear Regression

# 1.INTRODUCTION TO LINEAR REGRESSION

**Types of Linear Regression-**

**1. Simple Linear Regression-**

In simple linear regression, the dependent variable depends only on a single independent variable.

For simple linear regression, the form of the model is-

**$Y = \beta_0 + \beta_1 X$ or $Y=mx+c$**

Were,

Y is a dependent variable.

X is an independent variable.

**$\beta_0$ or c** and **$\beta_1$ or m** are the regression coefficients.

**$\beta_0$ or c** is the intercept or the bias that fixes the offset to a line.

**$\beta_1$ or m** is the slope or weight that specifies the factor by which X has an impact on Y.

# 1.INTRODUCTION TO LINEAR REGRESSION

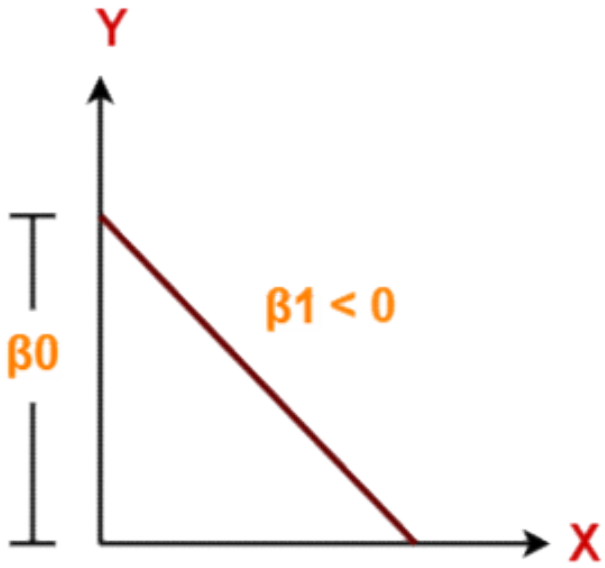
**Types of Linear Regression-**

**1. <u>Simple Linear Regression-</u>**

There are following 3 cases possible

**Case-01: $\beta_1 < 0$**

- It indicates that variable X has negative impact on Y.
- If X increases, Y will decrease and vice-versa.



# 1.INTRODUCTION TO LINEAR REGRESSION

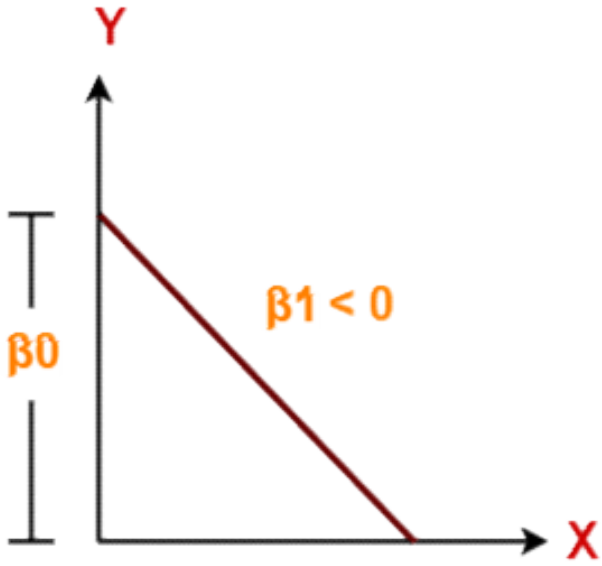
**Types of Linear Regression-**

**1. Simple Linear Regression-**

There are following 3 cases possible

**Case-02: $\beta_1 = 0$**

- It indicates that variable X has no impact on variable Y.
- If X changes, there will be no change in Y.



# 1.INTRODUCTION TO LINEAR REGRESSION

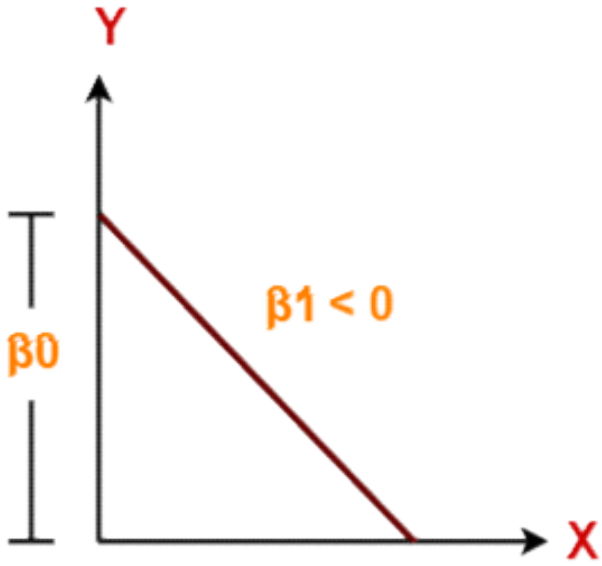
**Types of Linear Regression-**

**1. <u>Simple Linear Regression-</u>**

There are following 3 cases possible

**Case-02: $\beta_1 = 0$**

- It indicates that variable X has no impact on variable Y.
- If X changes, there will be no change in Y.



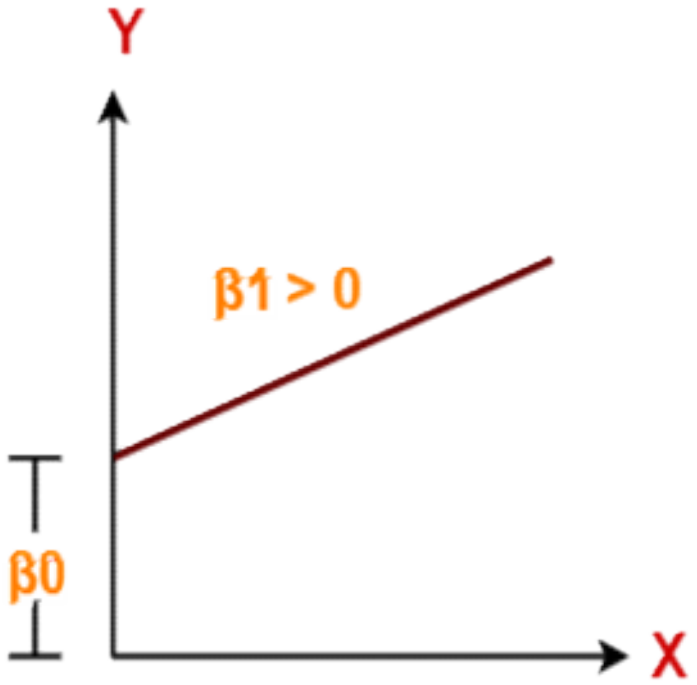
# 1.INTRODUCTION TO LINEAR REGRESSION

**1.Simple Linear Regression-**

**Case-03: $\beta_1 > 0$**

•It indicates that variable X has positive impact on Y.

•If X increases, Y will increase and vice-versa.



# 1.INTRODUCTION TO LINEAR REGRESSION

## 2. Multiple Linear Regression-

In multiple linear regression, the dependent variable depends on more than one independent variables.

For multiple linear regression, the form of the model is-

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots\ldots + \beta_n X_n$$

Here,

- Y is a dependent variable.
- $X_1, X_2, \ldots, X_n$ are independent variables.
- $\beta_0, \beta_1, \ldots, \beta_n$ are the regression coefficients.
- $\beta_j$ ($1 <= j <= n$) is the slope or weight that specifies the factor by which $X_j$ has an impact on Y.

# 2.OPTIMAL COEFFICIENT IN LINEAR REGRESSION

**What are Regression Coefficients?**

- Regression coefficients can be defined as estimates of some unknown parameters to describe the relationship between a predictor variable and the corresponding response.
- In other words, regression coefficients are used to predict the value of an unknown variable using a known variable.
- Linear regression is used to quantify how a unit change in an independent variable causes an effect in the dependent variable by determining the equation of the best-fitted straight line.
- This process is known as regression analysis.

# 2.OPTIMAL COEFFICIENT IN LINEAR REGRESSION

**What are Regression Coefficients?**

**Formula for Regression Coefficients**

- The goal of linear regression is to find the equation of the straight line that best describes the relationship between two or more variables.
- For example, suppose a simple regression equation is given by y = 7x - 3, then 7 is the coefficient, x is the predictor and -3 is the constant term.
- Suppose the equation of the best-fitted line is given by Y = aX + b then, the regression coefficients formula is given as follows:

$$a = \frac{n(\sum xy)-(\sum x)(\sum y)}{n(\sum x^2)-(\sum x)^2} \qquad b = \frac{(\sum y)(\sum x^2)-(\sum x)(\sum xy)}{n(\sum x^2)-(\sum x)^2}$$

here, n refers to the number of data points in the given data sets.

# 2.OPTIMAL COEFFICIENT IN LINEAR REGRESSION

**What are Regression Coefficients?**

- Regression coefficients are values that are used in a regression equation to estimate the predictor variable and its response.
- The most commonly used type of regression is linear regression. The equation of the best-fitted line is given by $Y = aX + b$.
- By using formulas, the values of the regression coefficient can be determined so as to get the regression line for the given variables.

# 2.OPTIMAL COEFFICIENT IN LINEAR REGRESSION

**What are Regression Coefficients?**

**Example 1**: Find the regression coefficients for the following data:

| Age | Glucose Level |
|---|---|
| 43 | 99 |
| 21 | 65 |
| 25 | 79 |
| 42 | 75 |
| 57 | 87 |
| 59 | 81 |

**Solution:**

| Age (x) | Glucose Level (y) | xy | $x^2$ | $y^2$ |
|---|---|---|---|---|
| 43 | 99 | 4257 | 1849 | 9801 |
| 21 | 65 | 1365 | 441 | 4225 |
| 25 | 79 | 1975 | 625 | 6241 |
| 42 | 75 | 3150 | 1764 | 5625 |
| 57 | 87 | 4959 | 3249 | 7569 |
| 59 | 81 | 4779 | 3481 | 6561 |
| Total = 247 | 486 | 20485 | 11409 | 40022 |

# 2.OPTIMAL COEFFICIENT IN LINEAR REGRESSION

## What are Regression Coefficients?

## Example 1: Solution

| Age (x) | Glucose Level (y) | xy | $x^2$ | $y^2$ |
|---|---|---|---|---|
| 43 | 99 | 4257 | 1849 | 9801 |
| 21 | 65 | 1365 | 441 | 4225 |
| 25 | 79 | 1975 | 625 | 6241 |
| 42 | 75 | 3150 | 1764 | 5625 |
| 57 | 87 | 4959 | 3249 | 7569 |
| 59 | 81 | 4779 | 3481 | 6561 |
| **Total = 247** | **486** | **20485** | **11409** | **40022** |

The formula for finding the regression coefficients are as follows:

$$a = \frac{n(\sum xy)-(\sum x)(\sum y)}{n(\sum x^2)-(\sum x)^2}$$

$$= 0.39$$

$$b = \frac{(\sum y)(\sum x^2)-(\sum x)(\sum xy)}{n(\sum x^2)-(\sum x)^2}$$

$$= 65.14$$

The regression equation is $Y = 0.39X + 65.14$

**Answer:** a = 0.39 and b = 65.14

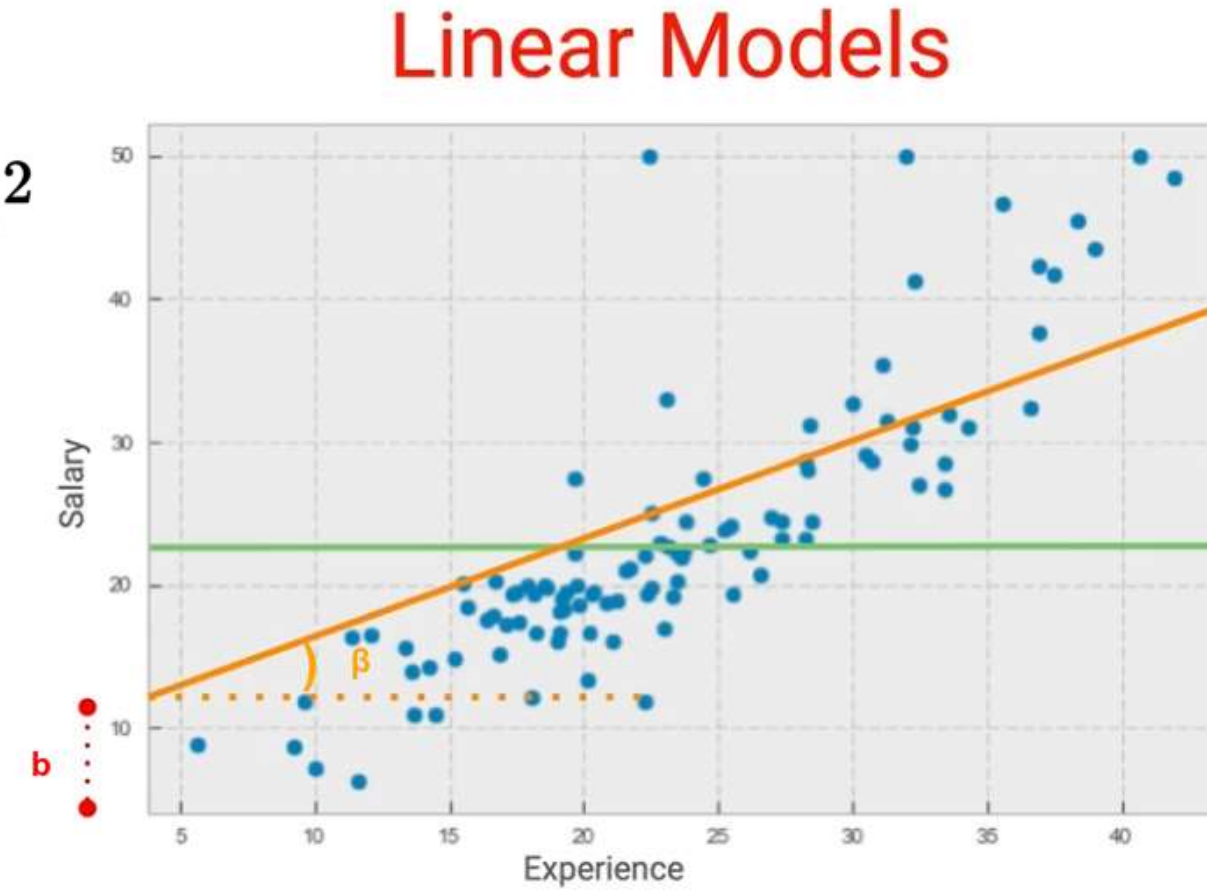
# 3.COST FUNCTION

- While dealing with Linear Regression we can have multiple lines for different values of slopes and intercepts.
- But the main question that arises is which of those lines actually represents the right relationship between the X and Y and in order to find that we can use the **Mean Squared Error** or **MSE** as the parameter.
- For linear regression, this MSE is nothing but the **Cost Function.**
- Mean Squared Error is the sum of the squared differences between the prediction and true value. And the output is a single number representing the **cost**.
- So the line with the minimum cost function or MSE represents the relationship between X and Y in the best possible manner. And once we have the slope and intercept of the line which gives the least error, we can use that line to predict Y.

# 3.COST FUNCTION

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

As we know that any line can be represented by two parameters- **slope** and **Intercept.**



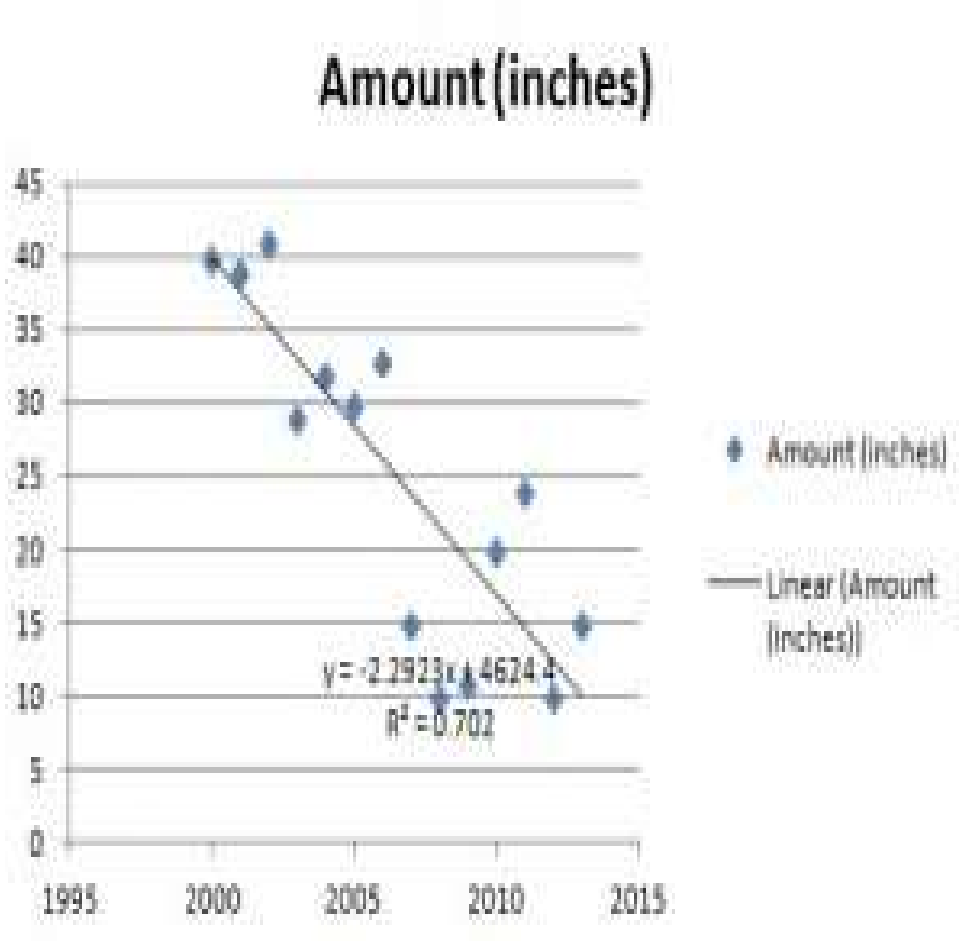
# 4. ANALYSIS OF LINEAR REGRESSION USING DUMMY DATA,

- In statistics, it's hard to stare at a set of random numbers in a table and try to make any sense of it. For example, global warming may be reducing average snowfall in your town and you are asked to predict how much snow you think will fall this year. Looking at the following table you might guess somewhere around 10-20 inches.
- That's a good guess, but you could make a *better* guess, by using regression.
- Essentially, regression is the "best guess" at using a set of data to make some kind of prediction. It's fitting a set of points to a graph. There's a whole host of tools that can run regression for you, including Excel, which I used here to help make sense of that snowfall data:

| Year | Amount (inches) |
|---|---|
| 2000 | 40 |
| 2001 | 39 |
| 2002 | 41 |
| 2003 | 29 |
| 2004 | 32 |
| 2005 | 30 |
| 2006 | 33 |
| 2007 | 15 |
| 2008 | 10 |
| 2009 | 11 |
| 2010 | 20 |
| 2011 | 24 |
| 2012 | 10 |
| 2013 | 15 |

# 4. ANALYSIS OF LINEAR REGRESSION USING DUMMY DATA,

| Year | Amount (inches) |
|---|---|
| 2000 | 40 |
| 2001 | 39 |
| 2002 | 41 |
| 2003 | 29 |
| 2004 | 32 |
| 2005 | 30 |
| 2006 | 33 |
| 2007 | 15 |
| 2008 | 10 |
| 2009 | 11 |
| 2010 | 20 |
| 2011 | 24 |
| 2012 | 10 |
| 2013 | 15 |




Just by looking at the regression line running down through the data, you can fine tune your best guess a bit. You can see that the original guess (20 inches or so) was way off.

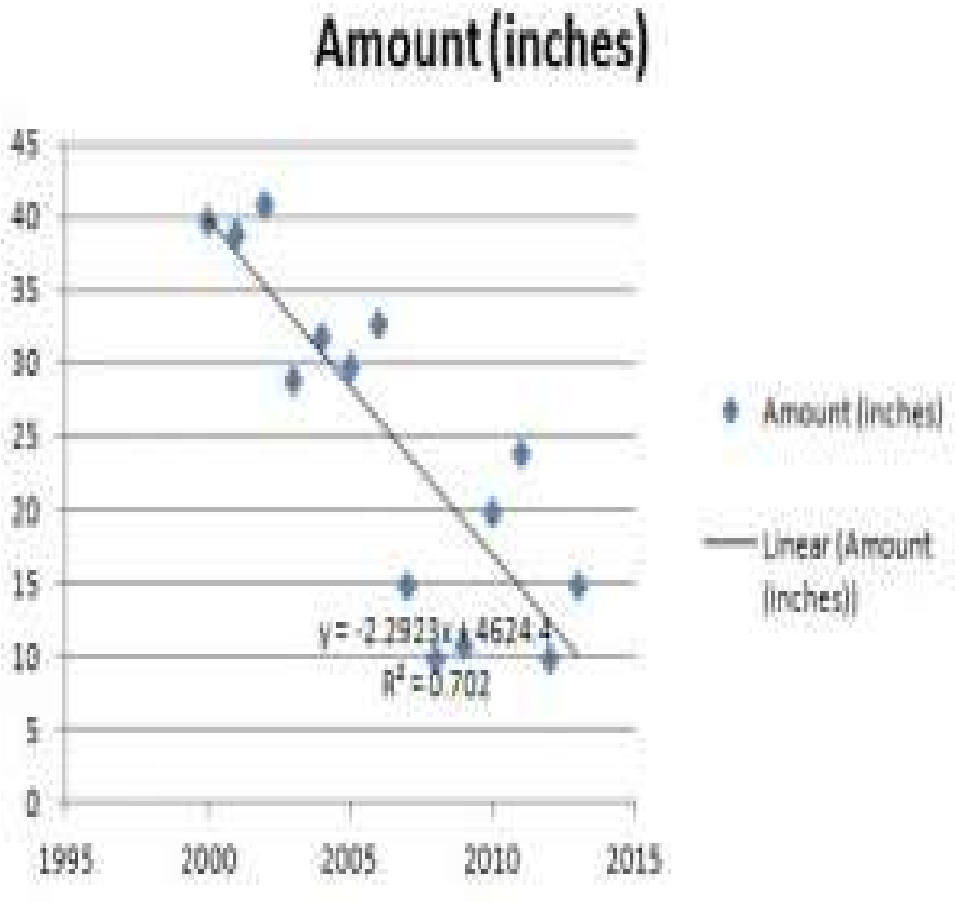
For 2015, it looks like the line will be somewhere between 5 and 10 inches! That might be “good enough”, but regression also gives you a useful equation, which for this chart is:

$y = -2.2923x + 4624.4$.

What that means is you can plug in an x value (the year) and get a pretty good estimate of snowfall for any year. For example, 2005:

$y = -2.2923(2005) + 4624.4 = 28.3385$ inches, which is pretty close to the actual figure of 30 inches for that year.

# 4. ANALYSIS OF LINEAR REGRESSION USING DUMMY DATA,

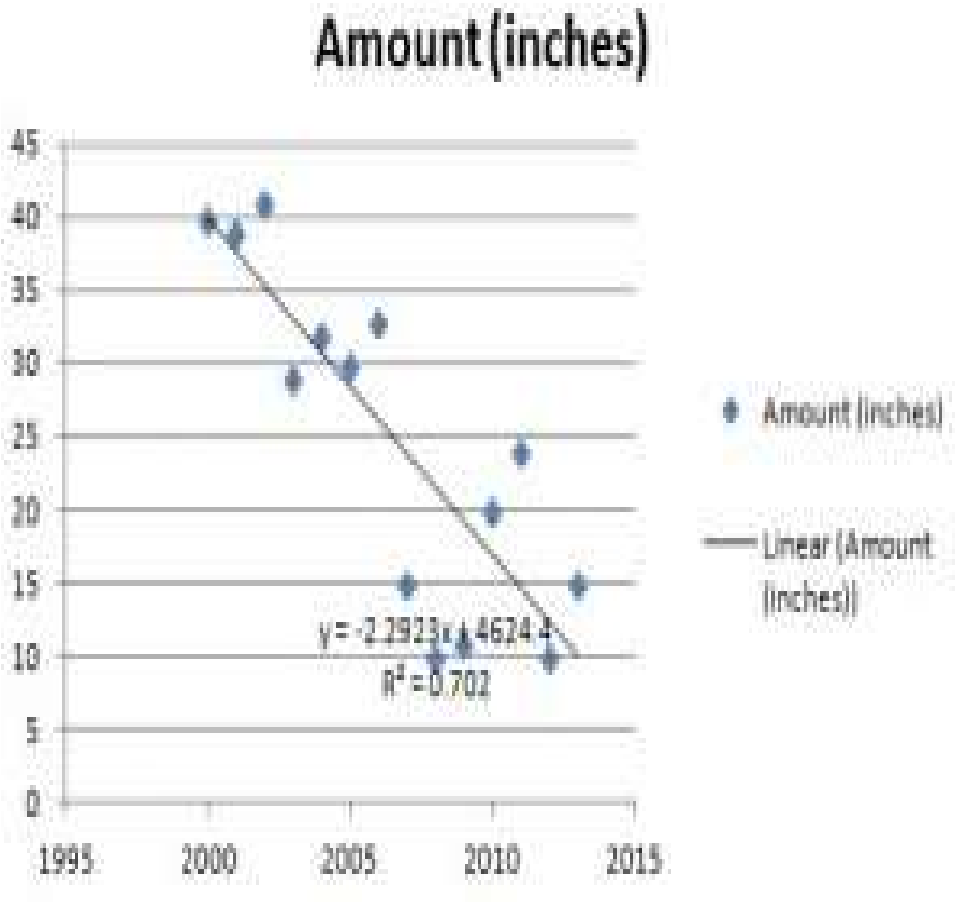



Best of all, you can use the equation to make predictions. For example, how much snow will fall in 2017?

**y = 2.2923(2017) + 4624.4 = 0.8 inches.**

Regression also gives you an R squared value, which for this graph is 0.702. This number tells you how good your model is. The values range from 0 to 1, with 0 being a terrible model and 1 being a perfect model. As you can probably see, 0.7 is a fairly decent model so you can be fairly confident in your weather prediction!

# 4. ANALYSIS OF LINEAR REGRESSION USING DUMMY DATA,




Best of all, you can use the equation to make predictions. For example, how much snow will fall in 2017?

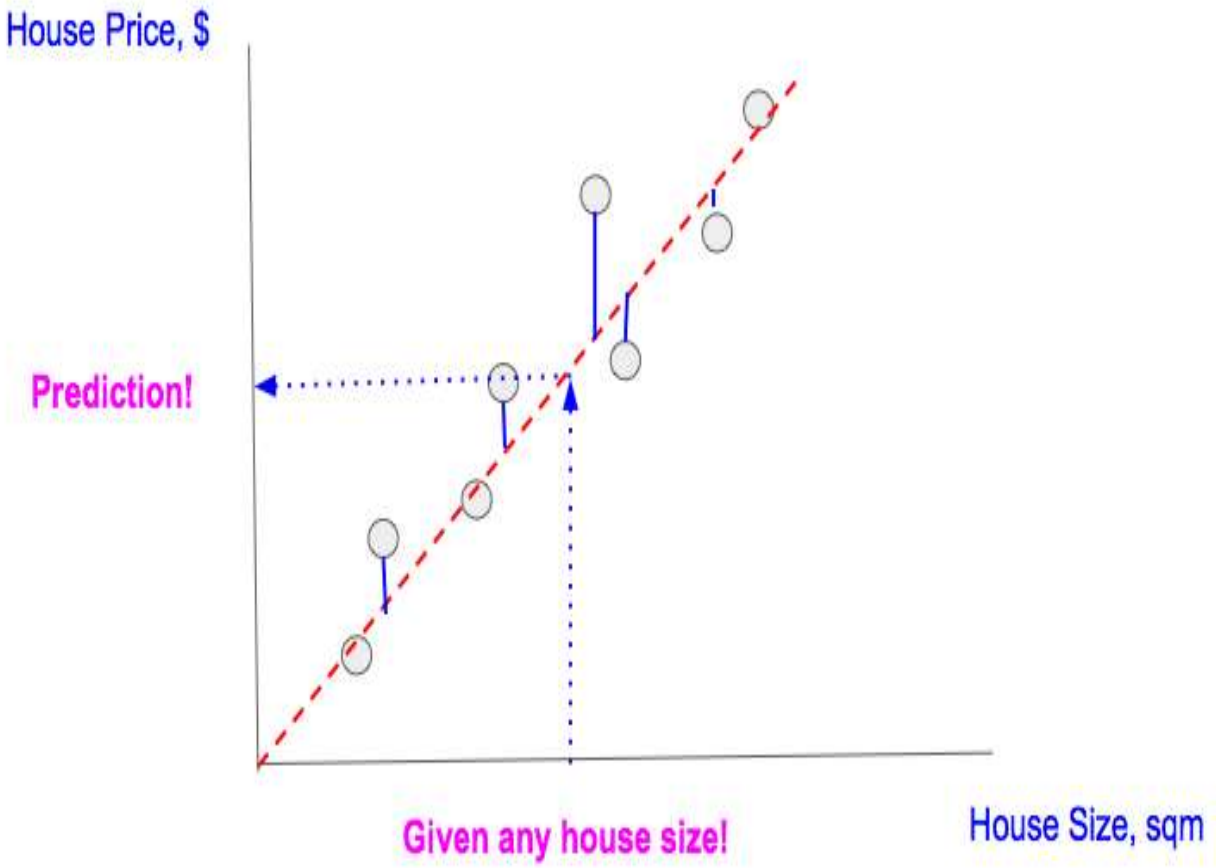
**y = 2.2923(2017) + 4624.4 = 0.8 inches.**

Regression also gives you an R squared value, which for this graph is 0.702. This number tells you how good your model is. The values range from 0 to 1, with 0 being a terrible model and 1 being a perfect model. As you can probably see, 0.7 is a fairly decent model so you can be fairly confident in your weather prediction!

# 5. LINEAR REGRESSION INTUITION.

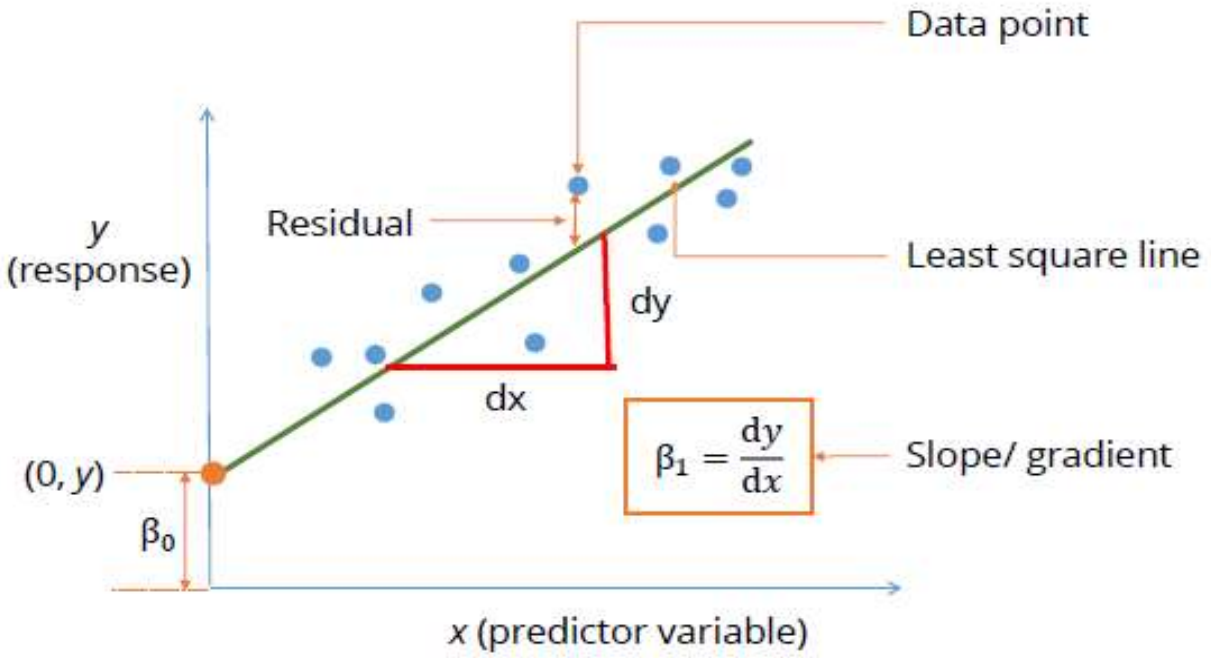
- Examples makes it easy to understand, so suppose you want to predict the price of a house by knowing its size. You have the data which has some house prices and corresponding sizes. Charting the data and fitting a line among them will look something like this:

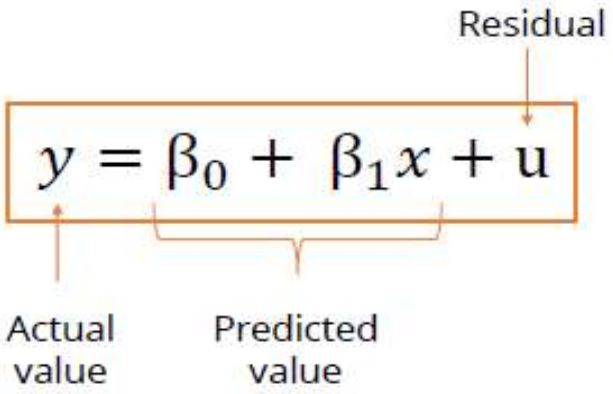
- To generalise, you draw a straight line such that it crosses through the maximum points. Once you get that line, for house of any size you just project that data point over the line which gives you the house price.

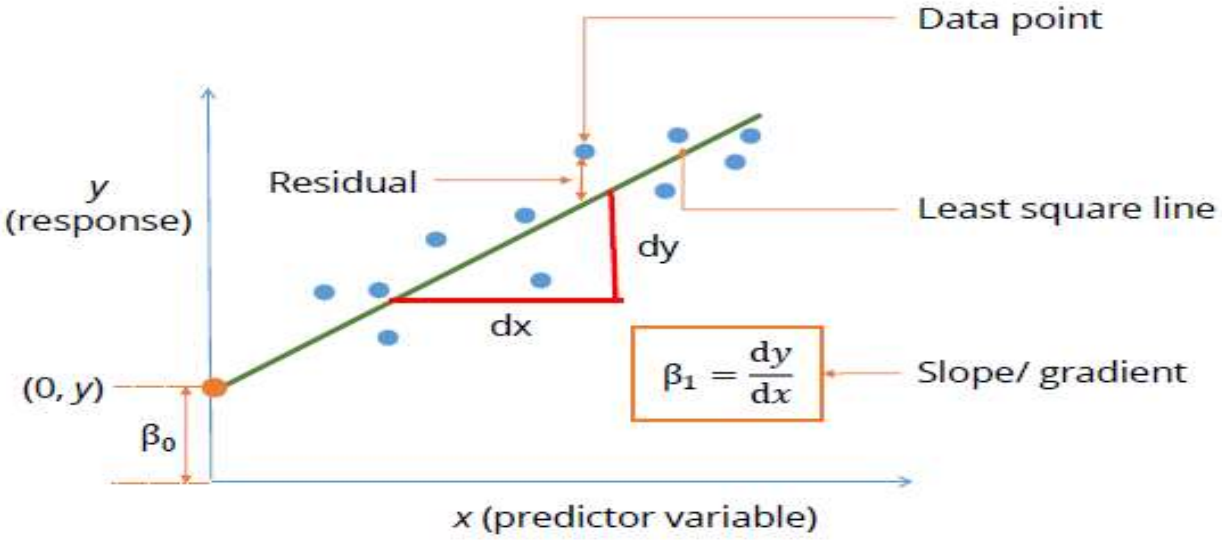


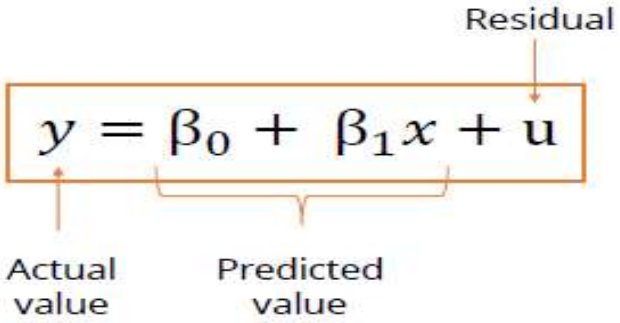
# 5. LINEAR REGRESSION INTUITION.

- The problem was never finding the house price. The problem was to find the best fit line which generalises over the data well.
- The same old line equation is used: $y = mx + c$ with a little statistical look and some below terminologies are added specific to Linear Regression modelling.






We will go over the elements of the equation one by one:

$y$ — The value that you want to predict

$\beta_0$ — The y-intercept of the line means where the line intersects the Y-axis

$\beta_1$ — The slope or gradient of the line means how steep is the line

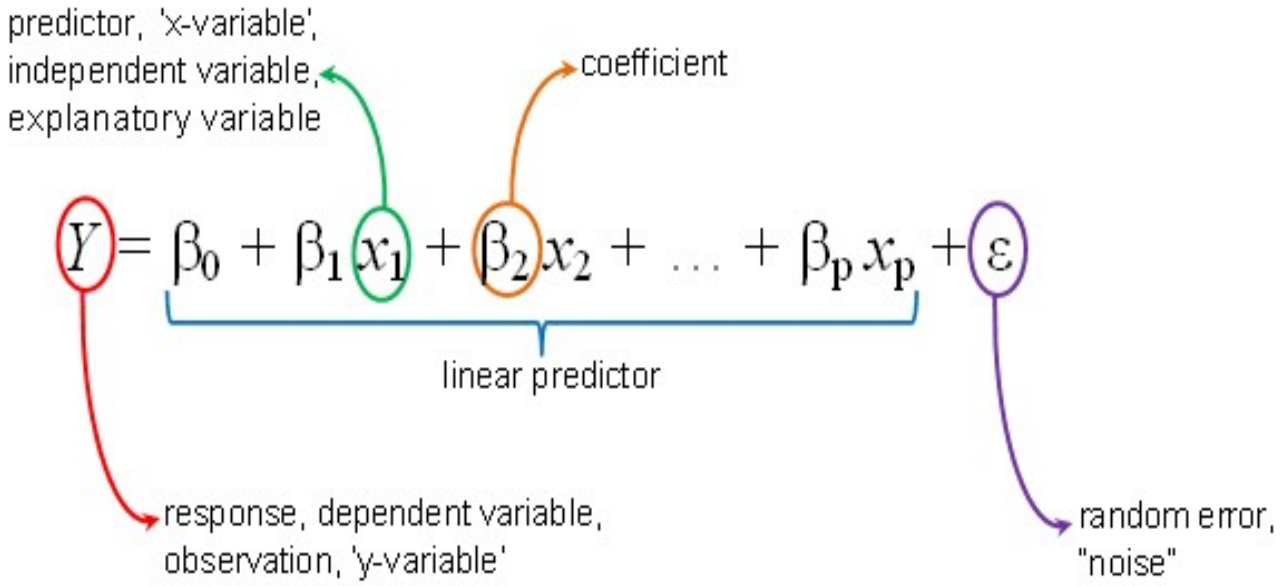
$x$ — The value of the data point

$u$ — The residual or noise that are caused by unexplained factors

# 5. LINEAR REGRESSION INTUITION.

- Seems like we can easily come to best parameter values by hit and trial method and thus finding the best fit line. But things are not that easy when the problem you are solving has more dimensions which is also known as the **The Curse of Dimensionality**.

- The Multiple Linear Regression will have n number of features and we have to find a line which fits all the data points for all the dimensions. You have started to realise that it is no longer a hit-and-trial solution.



# 5. LINEAR REGRESSION INTUITION.

- The Cost Function is a mathematical construct which is calculated by adding up the squared error terms
- .

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

## Gradient Descent

- Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.
- In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function f(x) parameterized by x.
- Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters.
- The main objective of gradient descent is to minimize the convex function using iteration of parameter updates.
- Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

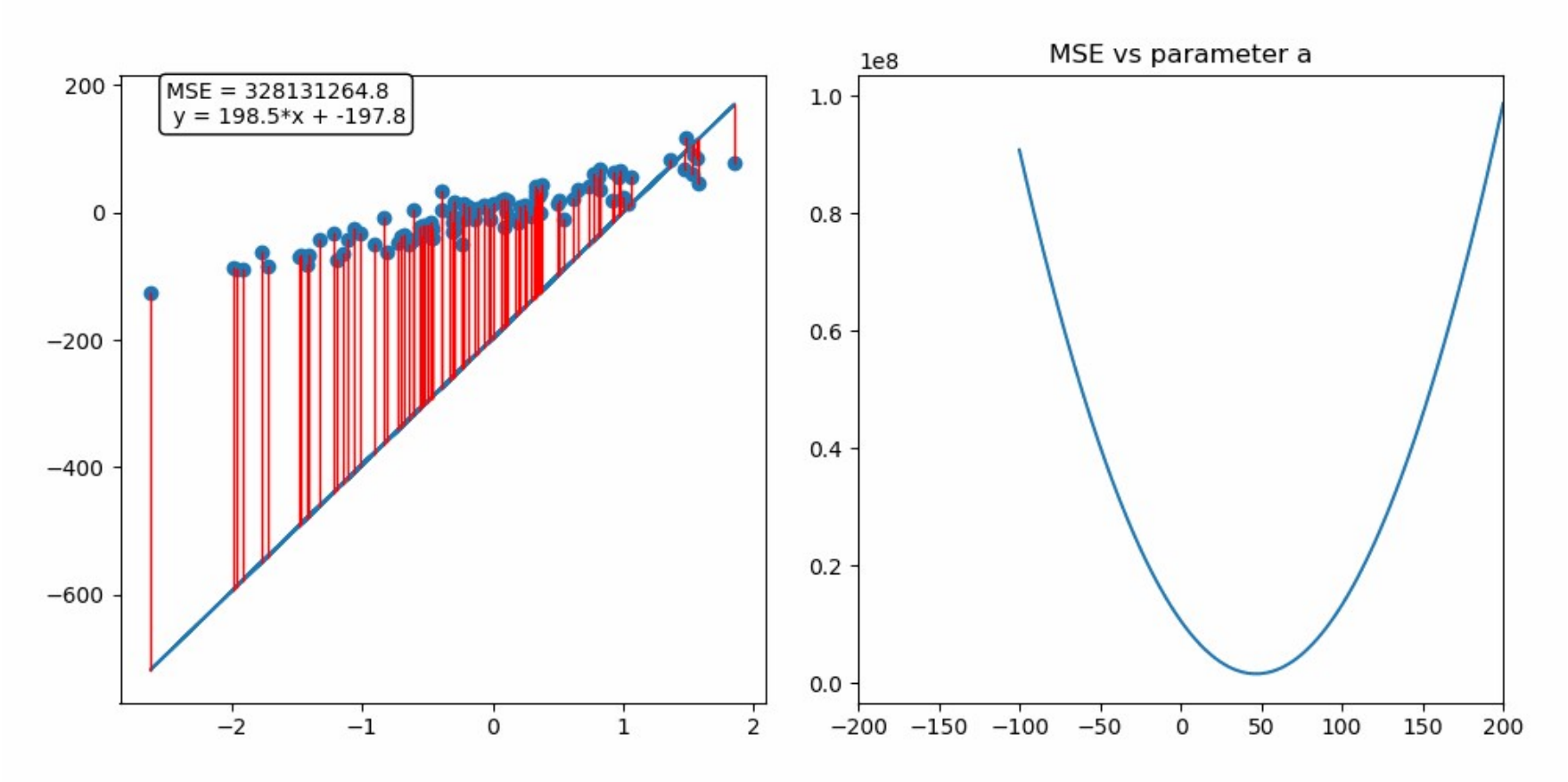
# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

## Gradient Descent

- *Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models.*
- *It helps in finding the* ***local minimum*** *of a function.*

# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

## Gradient Descent



# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

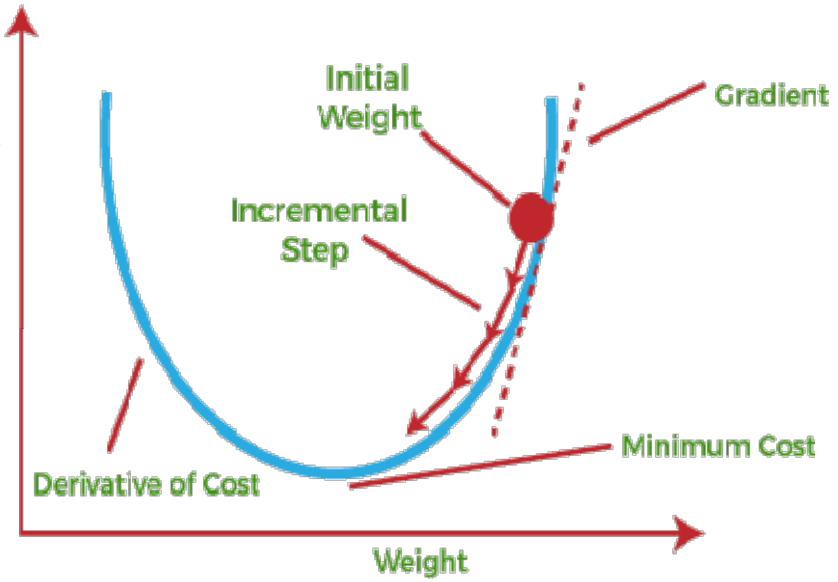
## Gradient Descent

- The best way to define the local minimum or local maximum of a function using gradient descent is as follows:
  - If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
  - Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.

***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

•Calculates the first-order derivative of the function to compute the gradient or slope of that function.

•Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.



# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

**Gradient Descent**

What is Cost-function?

- ***The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.***
- It helps to increase and improve machine learning efficiency by providing feedback to this model so that it can minimize error and find the local or global minimum.
- Further, it continuously iterates along the direction of the negative gradient until the cost function approaches zero.
- At this steepest descent point, the model will stop learning further. Although cost function and loss function are considered synonymous, also there is a minor difference between them.
- The slight difference between the loss function and the cost function is about the error within the training of machine learning models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

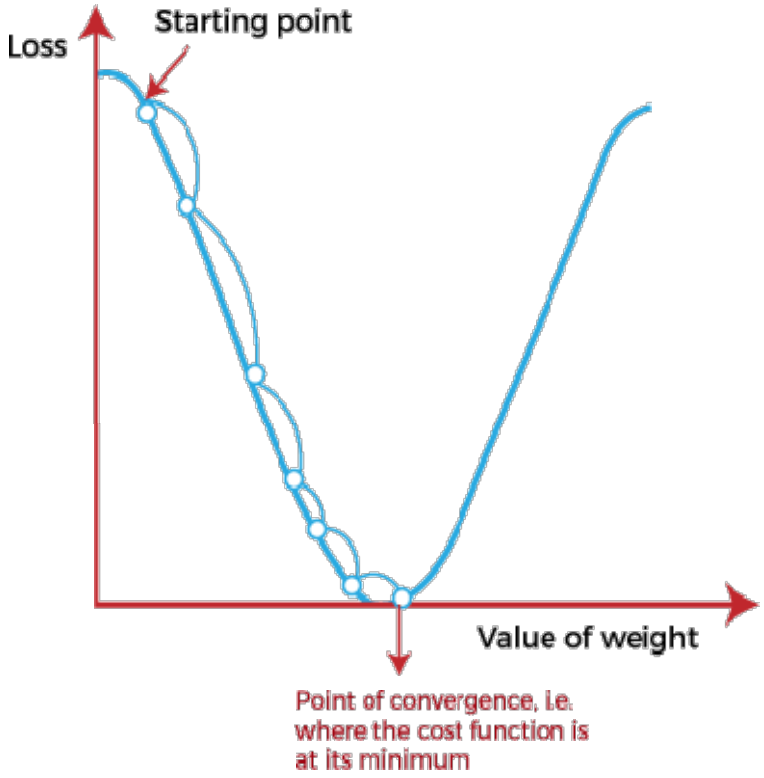
## Gradient Descent

**How does Gradient Descent work?**

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

$$Y=mX+c$$

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

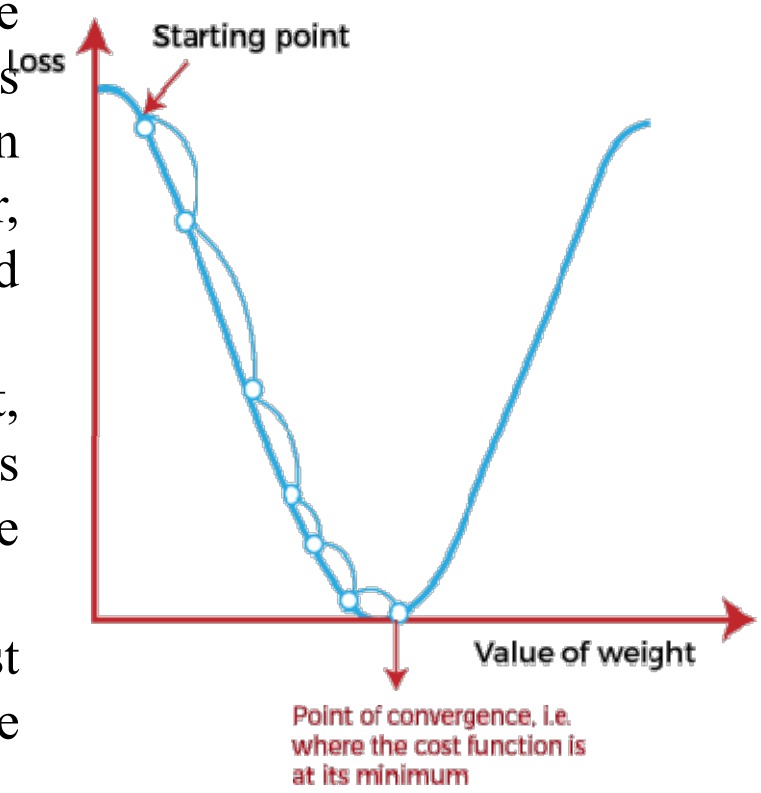
## Gradient Descent

**How does Gradient Descent work?**

The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called **a point of convergence.**

The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:



# 6. MULTIVARIABLE REGRESSION AND GRADIENT DESCENT:

## Gradient Descent

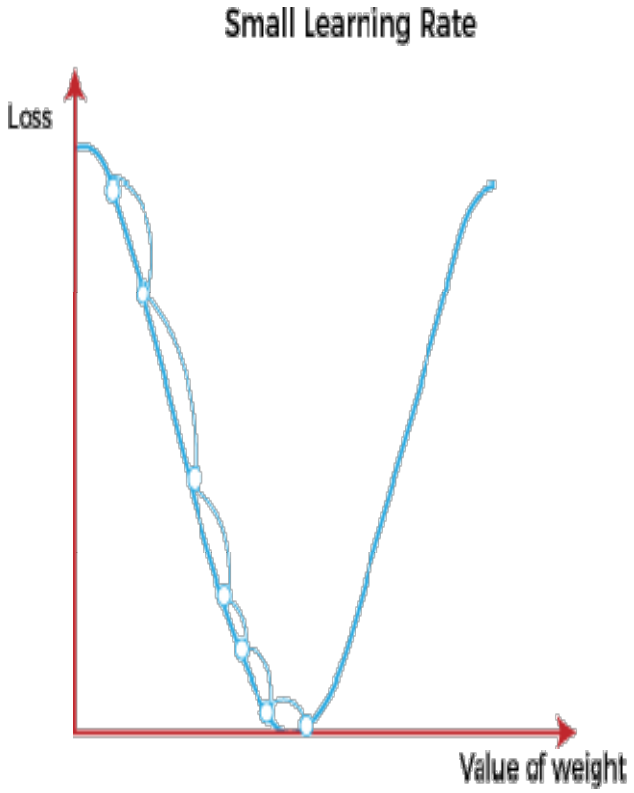
### How does Gradient Descent work?

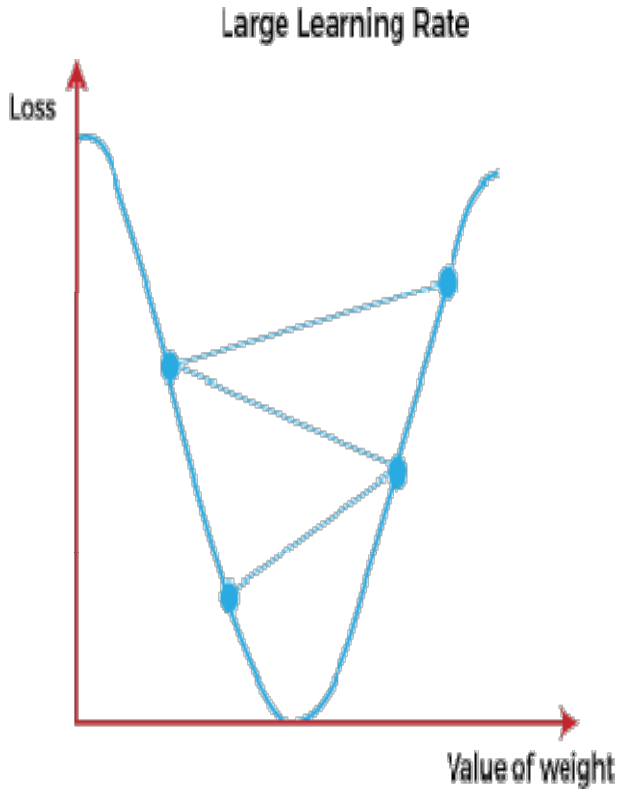
To minimize the cost function, two data points are required:

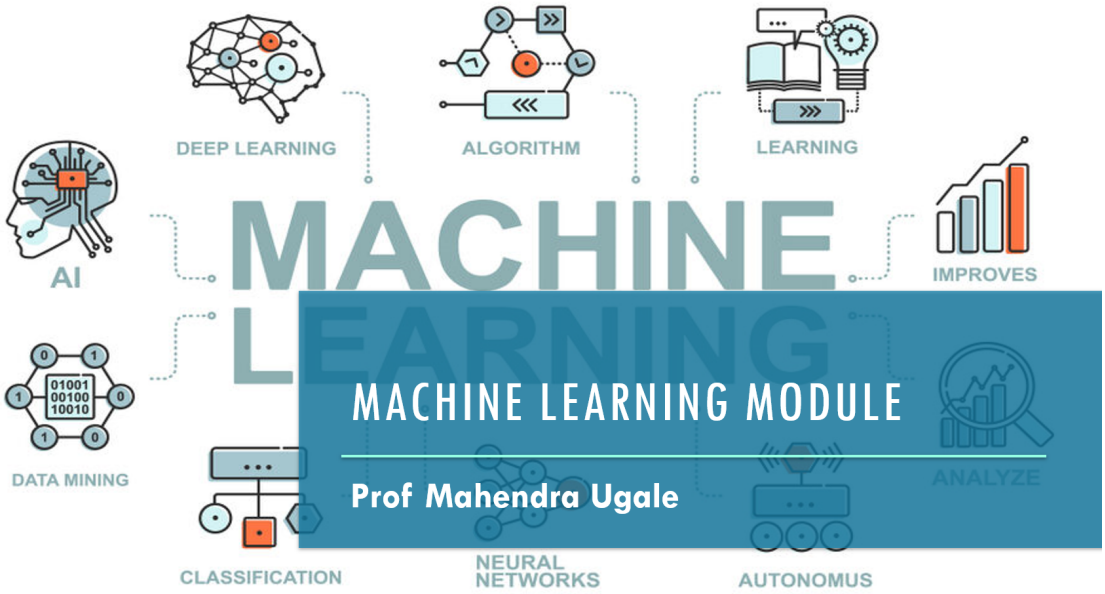
- **Direction &**
- **Learning Rate**

**Learning Rate:**

- It is defined as the step size taken to reach the minimum or lowest point.
- This is typically a small value that is evaluated and updated based on the behaviour of the cost function.
- If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum.
- At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.






# Regression Analysis in Machine learning

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.

More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed.

It predicts continuous/real values such as temperature, age, salary, price, etc.

We can understand the concept of regression analysis using the below example:

**Example:**

**Suppose there is a marketing company A, who does various advertisement every year and get sales on that.**

**The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:**

| Advertisement | Sales |
|---|---|
| $90 | $1000 |
| $120 | $1300 |
| $150 | $1800 |
| $100 | $1200 |
| $130 | $1380 |
| $200 | ?? |

**Now, the company wants to do the advertisement of $200 in the year 2019 and wants to know the prediction about the sales for this year.**

**So to solve such type of prediction problems in machine learning, we need regression analysis.**

**"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."**

## Terminologies Related to the Regression Analysis:

**- Dependent Variable:**

**The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called target variable.**

**- Independent Variable:**

**The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a predictor.**

**- Outliers:**

**Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.**

**- Multicollinearity:**

**If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.**

**- Underfitting and Overfitting:**

**If our algorithm works well with the training dataset but not well with test dataset, then such problem is called Overfitting. And if our algorithm does not perform well even with training dataset, then such problem is called underfitting.**

## Types of Regression

**There are various types of regressions which are used in data science and machine learning.**

**Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables.**

**Here we are discussing some important types of regression which are given below:**

- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**
- **Lasso Regression:**

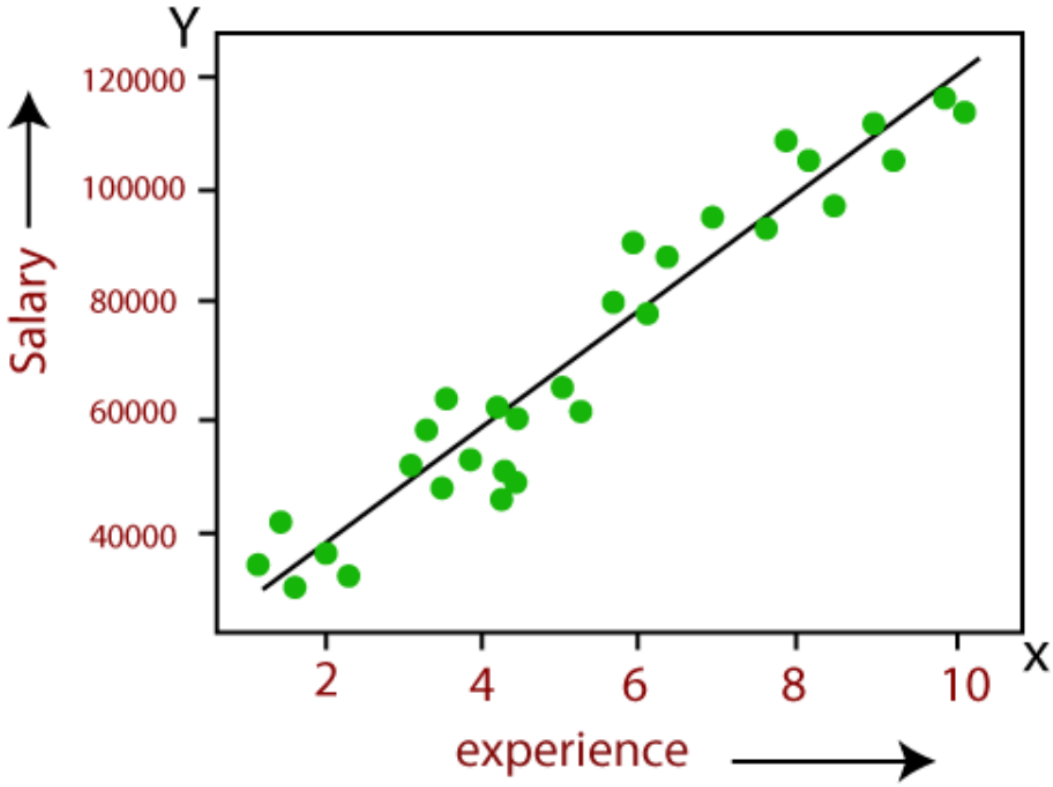
**Linear Regression:**

- **Linear regression is a statistical regression method which is used for predictive analysis.**
- **It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.**
- **It is used for solving the regression problem in machine learning.**
- **Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.**
- **If there is only one input variable (x), then such linear regression is called simple linear regression. And if there is more than one input variable, then such linear**

**regression is called multiple linear regression.**

- **The relationship between variables in the linear regression model can be explained using the below image.**

**Here we are predicting the salary of an employee on the basis of the year of experience.**




**Below is the mathematical equation for Linear regression:**

**Y= aX+b or Y=mX+c**

**Were,**

**Y = dependent variables (target variables),**

**X= Independent variables (predictor variables),**

**a and b are the linear coefficients**

**1. Linear Regression**

**Linear regression is one of the easiest and most popular Machine Learning algorithms.**

**It is a statistical method that is used for predictive analysis.**

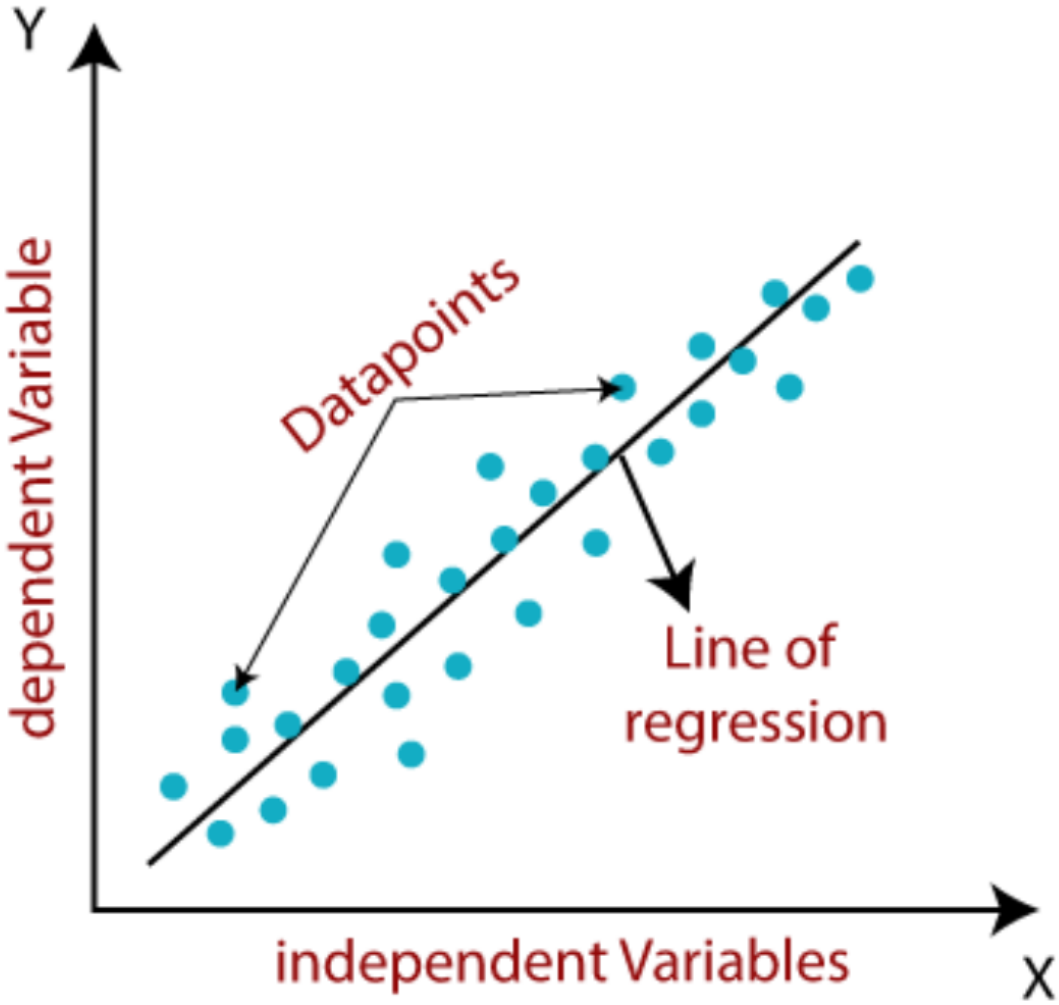
**Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.**

**Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.**

**Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.**

**The linear regression model provides a sloped straight line representing the relationship between the variables.**

**Consider the below image:**




**Mathematically, we can represent a linear regression as:**

**$y= a0+a1x+ \varepsilon$**

**Y= Dependent Variable (Target Variable)**

**X= Independent Variable (predictor Variable)**

**a0= intercept of the line (Gives an additional degree of freedom)**

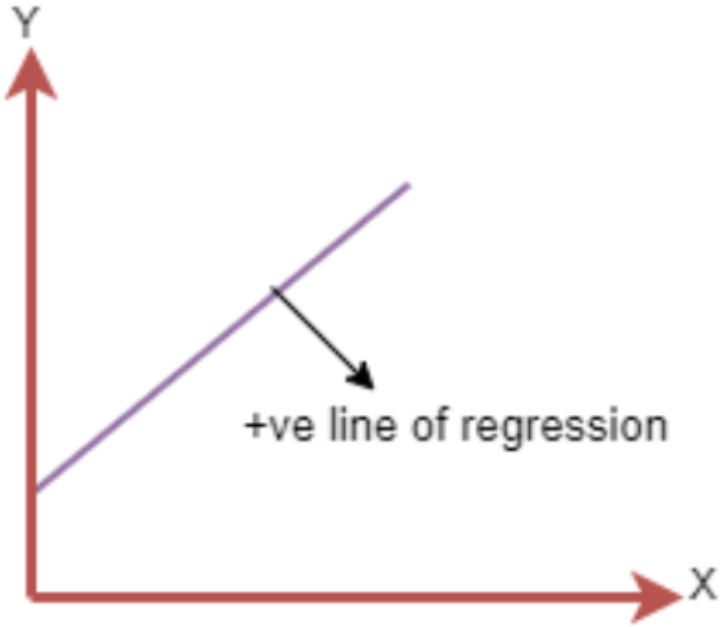
**a1 = Linear regression coefficient (scale factor to each input value).**

**$\varepsilon$ = random error**

**The values for x and y variables are training datasets for Linear Regression model representation.**

**Positive Linear Relationship:**

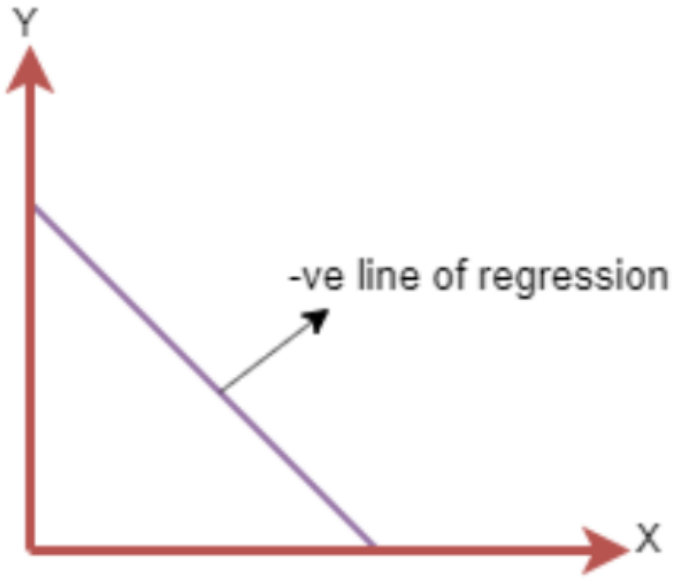
**If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.**




The line equation will be: $Y= a_0+a_1x$

**Negative Linear Relationship:**

**If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.**




The line of equation will be: $Y= -a_0+a_1x$

**Finding the best fit line:**

**When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized.**

**The best fit line will have the least error.**

**The different values for weights or the coefficient of lines (a0, a1) gives a different line of regression, so we need to calculate the best values for a0 and a1 to find the best fit line,**

**What problem we are solving??**

**Geometric Intuition??**

**Mathematatical Intuition**

**What problem we are solving??**

**Geometric Intuition??**

**Mathematatical Intuition**

# Regression Algorithms - Linear Regression

**Introduction to Linear Regression**

- **Linear regression may be defined as the statistical model that analyzes the linear relationship between a dependent variable with given set of independent variables.**
- **Linear relationship between variables means that when the value of one or more independent variables will change (increase or decrease), the value of dependent variable will also change accordingly (increase or decrease).**
- **Mathematically the relationship can be represented with the help of following equation –**

**Y = mX + b**

**Here,**

**Y is the dependent variable we are trying to predict**

**X is the dependent variable we are using to make predictions.**

**m is the slop of the regression line which represents the effect X has on Y**

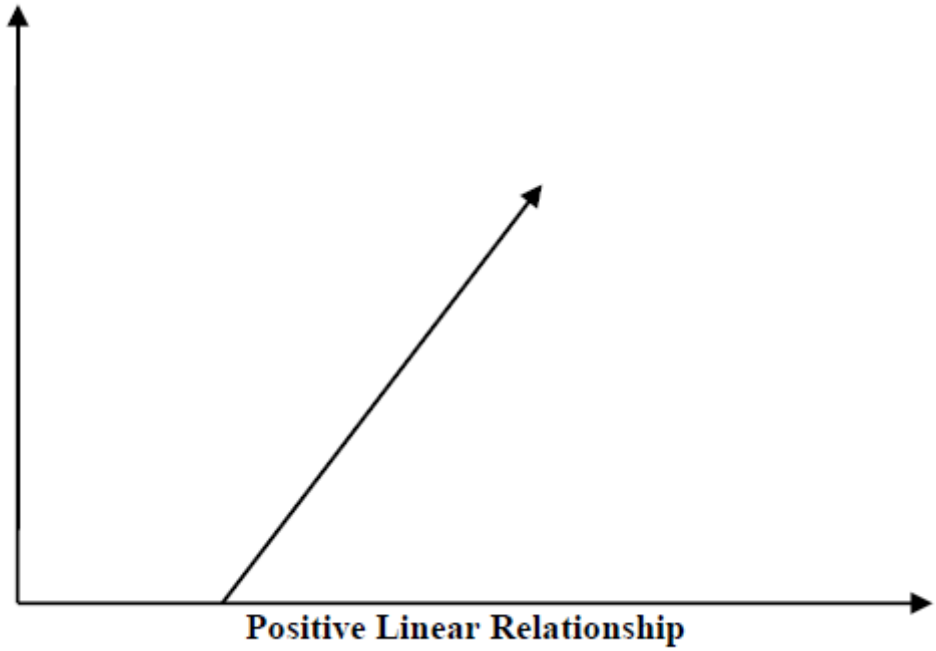
**b is a constant, known as the Y-intercept. If X = 0,Y would be equal to b.**

**Furthermore, the linear relationship can be positive or negative in nature as explained below −**

**Positive Linear Relationship**

**A linear relationship will be called positive if both independent and dependent variable increases.**

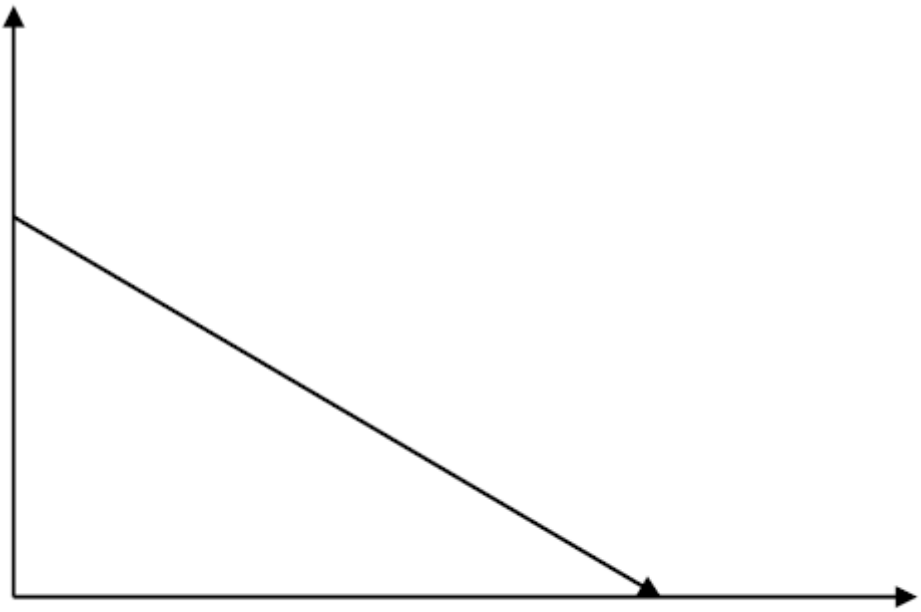
**It can be understood with the help of following graph −**




**Negative Linear relationship**

**A linear relationship will be called negative if independent increases and dependent variable decreases.**

**It can be understood with the help of following graph −**



Negative Linear Relationship

**Types of Linear Regression**

**Linear regression is of the following two types –**

- **Simple Linear Regression**
- **Multiple Linear Regression**

**Simple Linear Regression (SLR)**

**It is the most basic version of linear regression which predicts a response using a single feature.**

**The assumption in SLR is that the two variables are linearly related.**

**In the following Python implementation example, we are using diabetes dataset from scikit-learn.**

**6 Steps to build a Linear Regression model**

**Step 1: Importing the dataset**

**Step 2: Data pre-processing**

**Step 3: Splitting the test and train sets**

**Step 4: Fitting the linear regression model to the training set**

**Step 5: Predicting test results**

**Step 6: Visualizing the test results**

**Implementing a Linear Regression Model in Python**

**We will be using salary dataset.**

**Our dataset will have 2 columns namely:**

- Years of Experience and
- Salary.

**1. Importing the dataset**

**We will begin with importing the dataset using pandas and also import other libraries such as numpy and matplotlib.**

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
```

Out[3]:

| | YearsExperience | Salary |
|---|---|---|
| 0 | 1.1 | 39343.0 |
| 1 | 1.3 | 46205.0 |
| 2 | 1.5 | 37731.0 |
| 3 | 2.0 | 43525.0 |
| 4 | 2.2 | 39891.0 |

**2. Data Preprocessing**

**Now that we have imported the dataset, we will perform data preprocessing.**

In [4]:
```
X = dataset.iloc[:,:1].values  #independent variable array

y = dataset.iloc[:,1].values  #dependent variable vector
```

In [6]:
```
y
```

Out[6]:
```
array([ 39343.,  46205.,  37731.,  43525.,  39891.,  56642.,  60150.,
        54445.,  64445.,  57189.,  63218.,  55794.,  56957.,  57081.,
        61111.,  67938.,  66029.,  83088.,  81363.,  93940.,  91738.,
        98273., 101302., 113812., 109431., 105582., 116969., 112635.,
       122391., 121872.])
```

In [7]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state
```

**4. Fitting linear regression model into the training set**

**From sklearn's linear model library, import linear regression class.**

**reate an object for a linear regression class called regressor.**

**To fit the regressor into the training set, we will call the fit method – function to fit the regressor into the training set.**

**We need to fit X_train (training data of matrix of features) into the target values y_train.**

**Thus the model learns the correlation and learns how to predict the dependent variables based on the independent variable.**

In [8]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

regressor.fit(X_train,y_train) #actually produces the linear eqn for the data
```

Out[8]:
```
LinearRegression()
```

**5. Predicting the test set results**

**We create a vector containing all the predictions of the test set salaries.**

**The predicted salaries are then put into the vector called y_pred.(contains prediction for all observations in the test set)**

**predict method makes the predictions for the test set.**

**Hence, the input is the test set. The parameter for predict must be an array or sparse matrix, hence input is X_test.**

In [10]:
```
y_pred = regressor.predict(X_test)
y_pred
```

Out[10]:
```
array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
       115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
        76349.68719258, 100649.1375447 ])
```

In [20]:
```
y_test
```

Out[20]:
```
array([ 37731., 122391.,  57081.,  63218., 116969., 109431., 112635.,
        55794.,  83088., 101302.])
```

**y_test is the real salary of the test set.**

**y_pred are the predicted salaries.**

**Visualizing the results**

**Let's see what the results of our code will look like when we visualize it.**

**1. Plotting the points (observations)**

**To visualize the data, we plot graphs using matplotlib.**

**To plot real observation points ie plotting the real given values.**

**The X-axis will have years of experience and the Y-axis will have the predicted salaries.**

**plt.scatter plots a scatter plot of the data.**

**Parameters include :**

In [12]:
```
#plot for the TRAIN

plt.scatter(X_train, y_train, color='red') # plotting the observation line

plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regress

plt.title("Salary vs Experience (Training set)") # stating the title of the graph

plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph
```


In [13]:
```
#plot for the TEST
plt.scatter(X_test, y_test, color='red')
plt.plot(X_test, regressor.predict(X_test), color='blue') # plotting the regressio
plt.title("Salary vs Experience (Testing set)")
plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```


In [11]:
```
#Next, we will be printing some coefficient like MSE,
#Variance score etc. as follows -
```

```
print(' Intecept: \n',regressor.intercept_)
print('Coefficients: \n', regressor.coef_)
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, y_pred))
print("R2 Score: %.2f" % r2_score(y_test, y_pred))
#print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
 Intecept:
 26816.19224403119
Coefficients:
 [9345.94244312]
Mean squared error: 21026037.33
Root Mean squared error: 4585.42
R2 Score: 0.97
```

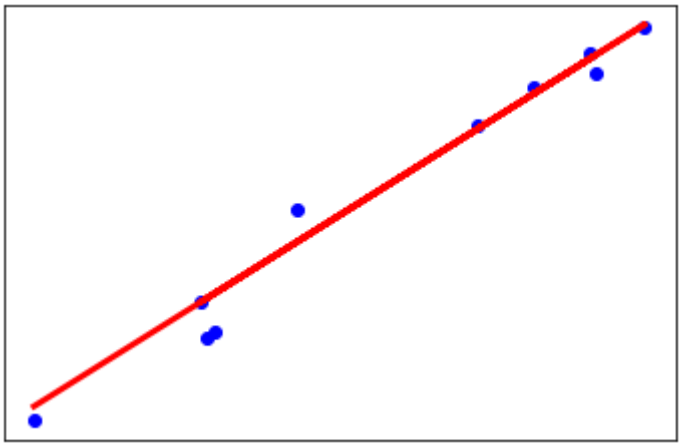
In [17]:
```
print(' Intecept: \n',regressor.intercept_)
```

```
 Intecept:
 26816.19224403119
```

In [18]:
```
print('Coefficients: \n', regressor.coef_)
```

```
Coefficients:
 [9345.94244312]
```

In [40]:
```
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```



# Multiple Linear Regression

**In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y).**

**But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.**

**Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.**

**We can define it as:**

**Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.**

**Some key points about MLR:**

- **For MLR, the dependent or target variable(Y) must be the continuous/real, but the predictor or independent variable may be of continuous or categorical form.**
- **Each feature variable must model the linear relationship with the dependent variable.**
- **MLR tries to fit a regression line through a multidimensional space of data-points.**

**MLR equation:**

**In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x1, x2, x3, ...,xn.**

**Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:**

**Y= b0+b1x1+ b2x2+ b3x3+.....bnxn ..... (a)**

**Where,**

**Y= Output/Response variable**

**b0, b1, b2, b3 , bn....= Coefficients of the model.**

**x1, x2, x3, x4,...= Various Independent/feature variable**

**Assumptions for Multiple Linear Regression:**

- **A linear relationship should exist between the Target and predictor variables.**
- **The regression residuals must be normally distributed.**
- **MLR assumes little or no multicollinearity (correlation between the independent variable) in data.**

**Implementation of Multiple Linear Regression model using Python:**

**We have a dataset of 50 start-up companies. This dataset contains five main information:**

**R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year.**

**Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.**

**Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables.**

**Below are the main steps of deploying the MLR model:**

- **Data Pre-processing Steps**
- **Fitting the MLR model to the training set**
- **Predicting the result of the test set**

**Step-1: Data Pre-processing Step:**

**The very first step is data pre-processing, which we have already discussed in this tutorial.**

**This process contains the below steps:**

**Importing libraries:**

**Firstly we will import the library which will help in building the model. Below is the code for it:**

In [20]:
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

**Importing dataset:**

**Now we will import the dataset(50_CompList), which contains all the variables.**

**Below is the code for it:**

In [21]:
```
#importing datasets
data_set= pd.read_csv('50_Startups.csv')
```

In [22]:
```
data_set.head()
```

Out[22]:

| | R&D Spend | Administration | Marketing Spend | State | Profit |
|---|---|---|---|---|---|
| 0 | 165349.20 | 136897.80 | 471784.10 | New York | 192261.83 |
| 1 | 162597.70 | 151377.59 | 443898.53 | California | 191792.06 |
| 2 | 153441.51 | 101145.55 | 407934.54 | Florida | 191050.39 |
| 3 | 144372.41 | 118671.85 | 383199.62 | New York | 182901.99 |
| 4 | 142107.34 | 91391.77 | 366168.42 | Florida | 166187.94 |

**In above output, we can clearly see that there are five variables, in which**

**four variables are continuous and one is categorical variable.**

In [25]:
```
data_set.shape
```

Out[25]:
```
(50, 5)
```

**Extracting dependent and independent Variables:**

In [27]:
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 4].values

print(x)
print(y)
```

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
 [67532.53 105751.03 304768.73 'Florida']
 [77044.01 99281.34 140574.81 'New York']
 [64664.71 139553.16 137962.62 'California']
 [75328.87 144135.98 134050.07 'Florida']
 [72107.6 127864.55 353183.81 'New York']
 [66051.52 182645.56 118148.2 'Florida']
 [65605.48 153032.06 107138.38 'New York']
 [61994.48 115641.28 91131.24 'Florida']
 [61136.38 152701.92 88218.23 'New York']
 [63408.86 129219.61 46085.25 'California']
 [55493.95 103057.49 214634.81 'Florida']
 [46426.07 157693.92 210797.67 'California']
 [46014.02 85047.44 205517.64 'New York']
 [28663.76 127056.21 201126.82 'Florida']
 [44069.95 51283.14 197029.42 'California']
 [20229.59 65947.93 185265.1 'New York']
 [38558.51 82982.09 174999.3 'California']
 [28754.33 118546.05 172795.67 'California']
 [27892.92 84710.77 164470.71 'Florida']
 [23640.93 96189.63 148001.11 'California']
 [15505.73 127382.3 35534.17 'New York']
 [22177.74 154806.14 28334.72 'California']
 [1000.23 124153.04 1903.93 'New York']
 [1315.46 115816.21 297114.46 'Florida']
 [0.0 135426.92 0.0 'California']
 [542.05 51743.15 0.0 'New York']
 [0.0 116983.8 45173.06 'California']]
[192261.83 191792.06 191050.39 182901.99 166187.94 156991.12 156122.51
 155752.6  152211.77 149759.96 146121.95 144259.4  141585.52 134307.35
 132602.65 129917.04 126992.93 125370.37 124266.9  122776.86 118474.03
 111313.02 110352.25 108733.99 108552.04 107404.34 105733.54 105008.31
 103282.38 101004.64  99937.59  97483.56  97427.84  96778.92  96712.8
  96479.51  90708.19  89949.14  81229.06  81005.76  78239.91  77798.83
  71498.49  69758.98  65200.33  64926.08  49490.75  42559.73  35673.41
  14681.4 ]
```

In [34]:
```
set(data_set["State"])  # 3 Categories are present
```

Out[34]:
```
{'California', 'Florida', 'New York'}
```

In [ ]:
```
data
```

In [18]:
```
# in the above data column 3 have categorical data so we have to encode
# Encoding categorical data
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [35]:
```
ct=ColumnTransformer(transformers=[("encoder", OneHotEncoder(), [3])], remainder="
```

In [36]:
```
X = ct.fit_transform(x)
X
```

Out[36]:
```
array([[0.0, 0.0, 1.0, 165349.2, 136897.8, 471784.1],
       [1.0, 0.0, 0.0, 162597.7, 151377.59, 443898.53],
       [0.0, 1.0, 0.0, 153441.51, 101145.55, 407934.54],
       [0.0, 0.0, 1.0, 144372.41, 118671.85, 383199.62],
       [0.0, 1.0, 0.0, 142107.34, 91391.77, 366168.42],
       [0.0, 0.0, 1.0, 131876.9, 99814.71, 362861.36],
       [1.0, 0.0, 0.0, 134615.46, 147198.87, 127716.82],
       [0.0, 1.0, 0.0, 130298.13, 145530.06, 323876.68],
       [0.0, 0.0, 1.0, 120542.52, 148718.95, 311613.29],
       [1.0, 0.0, 0.0, 123334.88, 108679.17, 304981.62],
       [0.0, 1.0, 0.0, 101913.08, 110594.11, 229160.95],
       [1.0, 0.0, 0.0, 100671.96, 91790.61, 249744.55],
       [0.0, 1.0, 0.0, 93863.75, 127320.38, 249839.44],
       [1.0, 0.0, 0.0, 91992.39, 135495.07, 252664.93],
       [0.0, 1.0, 0.0, 119943.24, 156547.42, 256512.92],
       [0.0, 0.0, 1.0, 114523.61, 122616.84, 261776.23],
       [1.0, 0.0, 0.0, 78013.11, 121597.55, 264346.06],
       [0.0, 0.0, 1.0, 94657.16, 145077.58, 282574.31],
       [0.0, 1.0, 0.0, 91749.16, 114175.79, 294919.57],
       [0.0, 0.0, 1.0, 86419.7, 153514.11, 0.0],
       [1.0, 0.0, 0.0, 76253.86, 113867.3, 298664.47],
       [0.0, 0.0, 1.0, 78389.47, 153773.43, 299737.29],
       [0.0, 1.0, 0.0, 73994.56, 122782.75, 303319.26],
       [0.0, 1.0, 0.0, 67532.53, 105751.03, 304768.73],
       [0.0, 0.0, 1.0, 77044.01, 99281.34, 140574.81],
       [1.0, 0.0, 0.0, 64664.71, 139553.16, 137962.62],
       [0.0, 1.0, 0.0, 75328.87, 144135.98, 134050.07],
       [0.0, 0.0, 1.0, 72107.6, 127864.55, 353183.81],
       [0.0, 1.0, 0.0, 66051.52, 182645.56, 118148.2],
       [0.0, 0.0, 1.0, 65605.48, 153032.06, 107138.38],
       [0.0, 1.0, 0.0, 61994.48, 115641.28, 91131.24],
       [0.0, 0.0, 1.0, 61136.38, 152701.92, 88218.23],
       [1.0, 0.0, 0.0, 63408.86, 129219.61, 46085.25],
       [0.0, 1.0, 0.0, 55493.95, 103057.49, 214634.81],
       [1.0, 0.0, 0.0, 46426.07, 157693.92, 210797.67],
       [0.0, 0.0, 1.0, 46014.02, 85047.44, 205517.64],
       [0.0, 1.0, 0.0, 28663.76, 127056.21, 201126.82],
       [1.0, 0.0, 0.0, 44069.95, 51283.14, 197029.42],
       [0.0, 0.0, 1.0, 20229.59, 65947.93, 185265.1],
       [1.0, 0.0, 0.0, 38558.51, 82982.09, 174999.3],
       [1.0, 0.0, 0.0, 28754.33, 118546.05, 172795.67],
       [0.0, 1.0, 0.0, 27892.92, 84710.77, 164470.71],
       [1.0, 0.0, 0.0, 23640.93, 96189.63, 148001.11],
       [0.0, 0.0, 1.0, 15505.73, 127382.3, 35534.17],
       [1.0, 0.0, 0.0, 22177.74, 154806.14, 28334.72],
       [0.0, 0.0, 1.0, 1000.23, 124153.04, 1903.93],
       [0.0, 1.0, 0.0, 1315.46, 115816.21, 297114.46],
       [1.0, 0.0, 0.0, 0.0, 135426.92, 0.0],
       [0.0, 0.0, 1.0, 542.05, 51743.15, 0.0],
       [1.0, 0.0, 0.0, 0.0, 116983.8, 45173.06]], dtype=object)
```

In [37]:
```
# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
```

In [40]:
```
# Training the Multiple Linear Regression model on the Training set

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[40]:
```
LinearRegression()
```

In [41]:
```
# Predicting the Test set results

y_pred = regressor.predict(X_test)
y_pred
```

Out[41]:
```
array([103015.20159796, 132582.27760816, 132447.73845174,  71976.09851258,
       178537.48221055, 116161.24230166,  67851.69209676,  98791.73374686,
       113969.43533013, 167921.06569551])
```

In [42]:
```
y_test
```

Out[42]:
```
array([103282.38, 144259.4 , 146121.95,  77798.83, 191050.39, 105008.31,
        81229.06,  97483.56, 110352.25, 166187.94])
```

**In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index.**

**For example, the first index has a predicted value of 103015$**

**profit and test/real value of 103282$ profit.**

**The difference is only of 267$, which is a good prediction, so, finally, our model is completed here.**

**We can also check the score for training dataset and test dataset. Below is the code for it:**

In [43]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

In [44]:
```
MAE = mean_absolute_error(y_test, y_pred)
MAE
```

Out[44]:
```
7514.2936596413765
```

In [45]:
```
MSE = mean_squared_error(y_test, y_pred)
MSE
```

Out[45]:
```
83502864.03259295
```

In [46]:
```
RMSE = nm.sqrt(MSE)
RMSE
```

Out[46]:
```
9137.990152795797
```

In [47]:
```
r2_score(y_test, y_pred)
```

Out[47]:
```
0.9347068473282303
```

# 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= b0+b1x1+ b2x12+ b2x13+..... bnx1n**

**It is also called the special case of Multiple Linear Regression in ML.**

**Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.**

**It is a linear model with some modification in order to increase the accuracy.**

**The dataset used in Polynomial regression for training is of non-linear nature.**

**It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.**

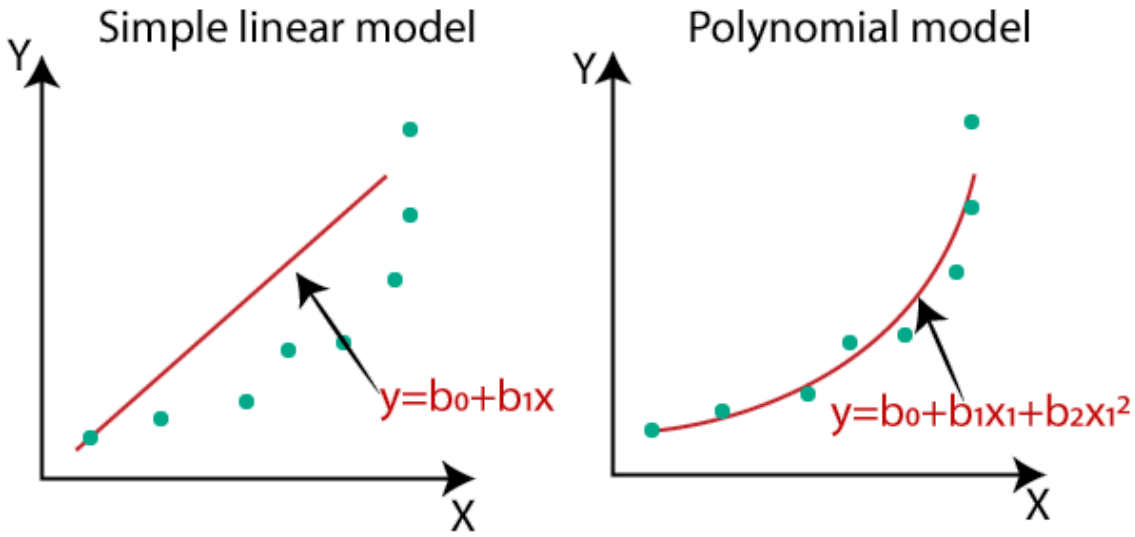
**Hence,**

**"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."**

## Need for Polynomial Regression:

**The need of Polynomial Regression in ML can be understood in the below points:**

- **If we apply a linear model on a linear dataset, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a non-linear dataset, then it will produce a drastic output.**
- **Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.**
- **So for such cases, where data points are arranged in a non-linear fashion, we need the Polynomial Regression model. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.**




**In the above image, we have taken a dataset which is arranged non-linearly.**

**So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.**

**Hence, if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.**

**Note:**

**A Polynomial Regression algorithm is also called Polynomial Linear Regression because it does not depend on the variables, instead, it depends on the coefficients, which are arranged in a linear fashion.**

**Equation of the Polynomial Regression Model**

**Simple Linear Regression equation:**
**y = b0+b1x .....(a)**

**Multiple Linear Regression equation:**
**y= b0+b1x+ b2x2+ b3x3+....+ bnxn .....(b)**

**Polynomial Regression equation:**
**y= b0+b1x + b2x2+ b3x3+....+ bnxn .....(c)**

**When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables.**

**The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree.**

**So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.**

**Problem Description:**

**There is a Human Resource company, which is going to hire a new candidate.**

**The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff.**

**So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels.**

**By checking the dataset available, we have found that there is a non-linear relationship between the Position levels and the salaries.**

**Our goal is to build a Bluffing detector regression model, so HR can hire an honest candidate. Below are the steps to build such a model.**

| Position | Level(X-variable) | Salary(Y-Variable) |
|---|---|---|
| Business Analyst | 1 | 45000 |
| Junior Consultant | 2 | 50000 |
| Senior Consultant | 3 | 60000 |
| Manager | 4 | 80000 |
| Country Manager | 5 | 110000 |
| Region Manager | 6 | 150000 |
| Partner | 7 | 200000 |
| Senior Partner | 8 | 300000 |
| C-level | 9 | 500000 |
| CEO | 10 | 1000000 |
| | | |

**Steps for Polynomial Regression:**

**The main steps involved in Polynomial Regression are given below:**

- **Data Pre-processing**
- **Build a Linear Regression model and fit it to the dataset**
- **Build a Polynomial Regression model and fit it to the dataset**
- **Visualize the result for Linear Regression and Polynomial Regression model.**
- **Predicting the output.**

**Data Pre-processing Step:**

**The data pre-processing step will remain the same as in previous regression models, except for some changes.**

**In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:**

**The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.**

**In this model, we want very accurate predictions for salary, so the model should have enough information.**

**The code for pre-processing step is given below:**

In [1]:
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [3]:
```
#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')
data_set.head()
```

Out[3]:

| | Position | Level | Salary |
|---|---|---|---|
| 0 | Business Analyst | 1 | 45000 |
| 1 | Junior Consultant | 2 | 50000 |
| 2 | Senior Consultant | 3 | 60000 |
| 3 | Manager | 4 | 80000 |
| 4 | Country Manager | 5 | 110000 |

In [4]:
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

In [9]:
```
y
```

Out[9]:
```
array([  45000,   50000,   60000,   80000,  110000,  150000,  200000,
        300000,  500000, 1000000], dtype=int64)
```

- **In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.**
- **Next, we have imported the dataset 'Position_Salaries.csv', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).**
- **After that, we have extracted the dependent(Y) and independent variable(X) from the dataset.**
- **For x-variable, we have taken parameters as [:,1:2], because we want 1 index(levels), and included :2 to make it as a matrix.**

**Building the Linear regression model:**

**Now, we will build and fit the Linear regression model to the dataset.**

**In building polynomial regression, we will take the Linear regression model as reference and compare both the results.**

**The code is given below:**

In [10]:
```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

Out[10]:
```
LinearRegression()
```

**Building the Polynomial regression model:**

**Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model.**

**Because here we will use PolynomialFeatures class of preprocessing library.**

**We are using this class to add some extra features to our dataset.**

In [11]:
```
 #Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

Out[11]:
```
LinearRegression()
```

**In the above lines of code, we have used poly_regs.fit_transform(x), because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model.**

**The parameter value(degree= 2) depends on our choice.**

**We can choose it according to our Polynomial features.**

**After executing the code, we will get another matrix x_poly, which can be seen under the variable explorer option:**

In [12]:
```
x_poly
```

Out[12]:
```
array([[  1.,   1.,   1.],
       [  1.,   2.,   4.],
       [  1.,   3.,   9.],
       [  1.,   4.,  16.],
       [  1.,   5.,  25.],
       [  1.,   6.,  36.],
       [  1.,   7.,  49.],
       [  1.,   8.,  64.],
       [  1.,   9.,  81.],
       [  1.,  10., 100.]])
```

**Next, we have used another LinearRegression object, namely lin_reg_2, to fit our**

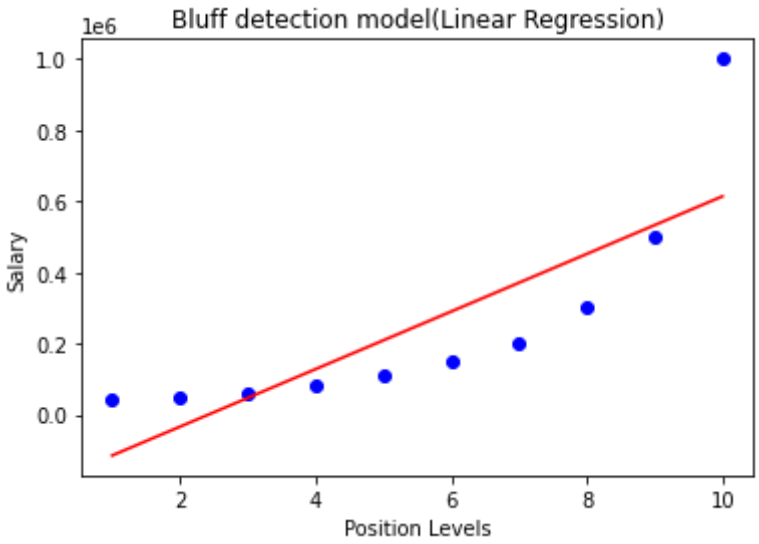
**x_poly vector to the linear model.**

**Visualizing the result for Linear regression:**

**Now we will visualize the result for Linear regression model as we did in Simple Linear Regression.**

**Below is the code for it:**

In [15]:
```
#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```




In [14]:
```
lin_regs.predict(x)
```

Out[14]:
```
array([-114454.54545455,  -33575.75757576,   47303.03030303,
        128181.81818182,  209060.60606061,  289939.39393939,
        370818.18181818,  451696.96969697,  532575.75757576,
        613454.54545455])
```

**In the above output image, we can clearly see that the regression line is so far from the datasets.**

**Predictions are in a red straight line, and blue points are actual values.**

**If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value.**

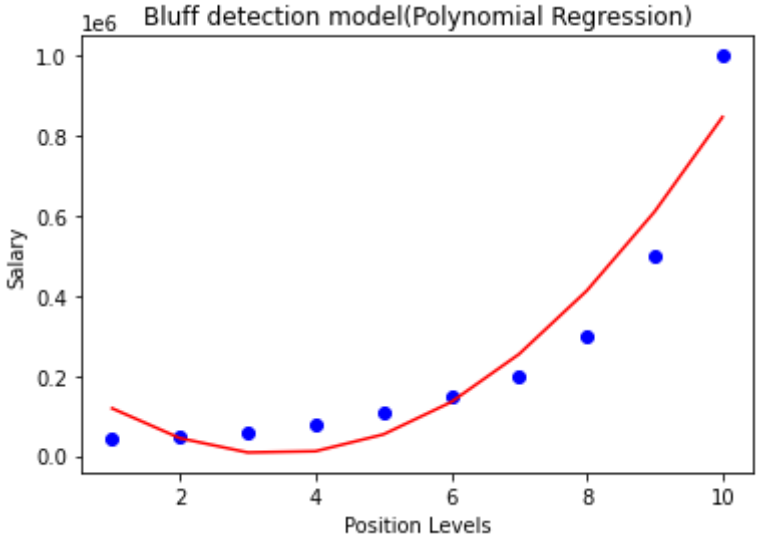
**So we need a curved model to fit the dataset other than a straight line.**

**Visualizing the result for Polynomial Regression**

**Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.**

**Code for this is given below:**

In [16]:
```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```




**In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x), instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.**

**As we can see in the above output image, the predictions are close to the real values.**

**The above plot will vary as we will change the degree.**

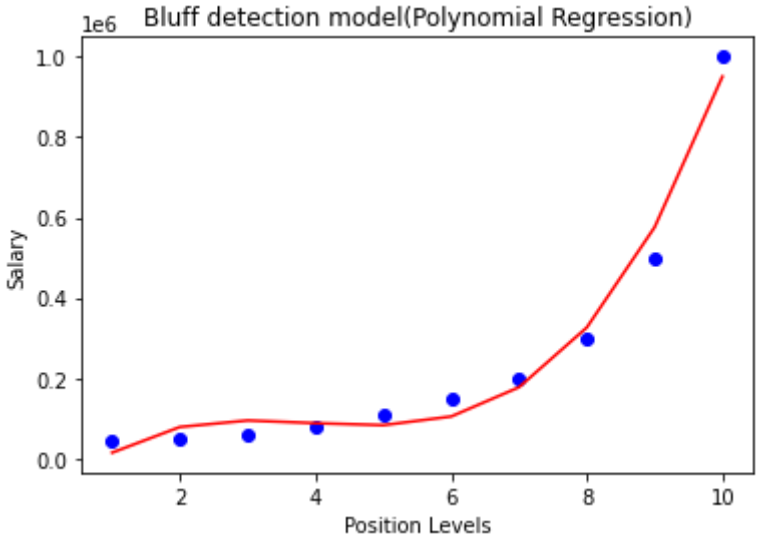
**For degree= 3:**

**If we change the degree=3, then we will give a more accurate plot, as shown in the below image.**

In [18]:
```
 #Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 3)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

Out[18]:
```
LinearRegression()
```

In [19]:
```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```




**SO as we can see here in the above output image, the predicted salary for level 6.5 is near to 170K$-190k$, which seems that future employee is saying the truth about his salary.**

**Degree= 4:**

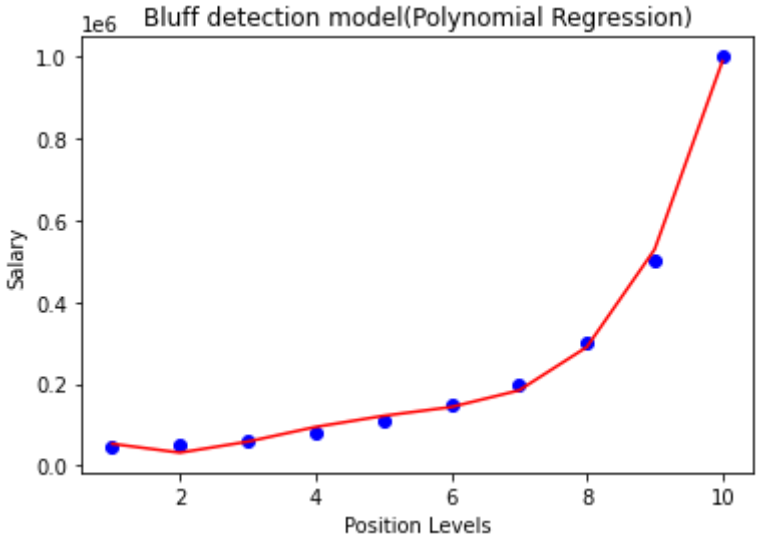
**Let's again change the degree to 4, and now will get the most accurate plot.**

**Hence we can get more accurate results by increasing the degree of Polynomial.**

In [28]:
```
 #Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 4)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

Out[28]:
```
LinearRegression()
```

In [29]:
```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```




**Predicting the final result with the Linear Regression model:**

**Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff.**

**So, for this, we will use the predict() method and will pass the value 6.5. Below is the code for it:**

In [26]:
```
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

```
[330378.78787879]
```

**Predicting the final result with the Polynomial Regression model:**

**Now, we will predict the final output using the Polynomial Regression model to compare with Linear model.**

**Below is the code for it:**

In [30]:
```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

```
[158862.45265153]
```

**As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.**

# Ridge and Lasso Regression

**Regularization**

**When it comes to training models, there are two major problems one can encounter:**

- **Overfitting and**

- **Underfitting.**

- **Overfitting happens when the model performs well on the training set but not so well on unseen (test) data.**
- **Underfitting happens when it neither performs well on the train set nor on the test set.**

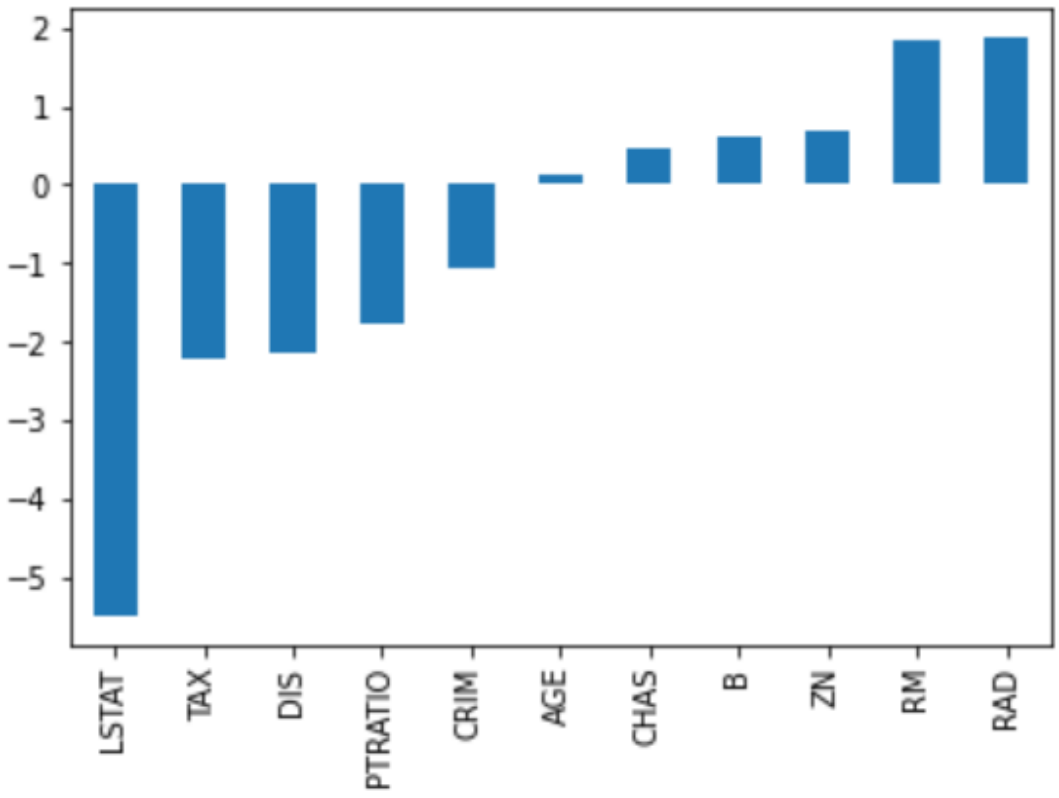
**Particularly, regularization is implemented to avoid overfitting of the data, especially when there is a large variance between train and test set performances.**

**With regularization, the number of features used in training is kept constant, yet the magnitude of the coefficients (w) is reduced.**

**Consider the image of coefficients below to predict house prices.**

**While there are quite a number of predictors, RM and RAD have the largest coefficients.**

**The implication of this will be that housing prices will be driven more significantly by these two features leading to overfitting, where generalizable patterns have not been learned.**




**There are different ways of reducing model complexity and preventing overfitting in linear models. This includes ridge and lasso regression models.**

**Introduction to Lasso Regression**

- **This is a regularization technique used in feature selection using a Shrinkage method also referred to as the penalized regression method.**
- **Lasso is short for Least Absolute Shrinkage and Selection Operator, which is used both for regularization and model selection.**
- **If a model uses the L1 regularization technique, then it is called lasso regression.**

**Lasso Regression for Regularization**

**In this shrinkage technique, the coefficients determined in the linear model are shrunk towards the central point as the mean by introducing a penalization factor called the alpha α (or sometimes lamda) values.**

$$L_{lasso}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m} |\hat{\beta}_j|.$$

**Alpha (α) is the penalty term that denotes the amount of shrinkage (or constraint) that will be implemented in the equation.**

**With alpha set to zero, you will find that this is the equivalent of the linear regression model and a larger value penalizes the optimization function.**

**Therefore, lasso regression shrinks the coefficients and helps to reduce the model complexity and multi-collinearity.**

**Alpha (α) can be any real-valued number between zero and infinity; the larger the value, the more aggressive the penalization is.**

**Lasso Regression for Model Selection**

**Due to the fact that coefficients will be shrunk towards a mean of zero, less important features in a dataset are eliminated when penalized. The shrinkage of these coefficients based on the alpha value provided leads to some form of automatic feature selection, as input variables are removed in an effective approach.**

# Ridge Regression

**Similar to the lasso regression, ridge regression puts a similar constraint on the coefficients by introducing a penalty factor.**

**However, while lasso regression takes the magnitude of the coefficients, ridge regression takes the square.**

$$L_{hridge}(\hat{\beta}) = \sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j-1}^{m} w_j \hat{\beta}_j^2.$$

**Ridge regression is also referred to as L2 Regularization.**

**Python Implementation**

**What we intend to see is:**

- **How to perform ridge and lasso regression in Python**
- **Compare the results with a linear regression model**

**Data Importation and EDA**

**We are going to use the Boston house prediction dataset.**

**This dataset is present in the datasets module of sklearn (scikit-learn) library.**

**We can import this dataset as follows.**

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [2]:
```
# Loading pre-defined Boston Dataset
df = datasets.load_boston()
df
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: Futu
reWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0
and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Out[2]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15.
,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n-------------------
```

```
--------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n
:Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute
14) is usually the target.\n\n    :Attribute Information (in order):\n        - CR
IM     per capita crime rate by town\n        - ZN       proportion of residential
land zoned for lots over 25,000 sq.ft.\n        - INDUS    proportion of non-retai
l business acres per town\n        - CHAS     Charles River dummy variable (= 1 if
tract bounds river; 0 otherwise)\n        - NOX      nitric oxides concentration
(parts per 10 million)\n        - RM       average number of rooms per dwelling\n
- AGE      proportion of owner-occupied units built prior to 1940\n        - DIS
weighted distances to five Boston employment centres\n        - RAD      index of
accessibility to radial highways\n        - TAX      full-value property-tax rate
per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n        - B        10
00(Bk - 0.63)^2 where Bk is the proportion of black people by town\n        - LSTA
T    % lower status of the population\n        - MEDV     Median value of owner-oc
cupied homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Ha
rrison, D. and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttp
s://archive.ics.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset w
as taken from the StatLib library which is maintained at Carnegie Mellon Universit
y.\n\nThe Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\npr
ices and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81
-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wiley,
1980.   N.B. Various transformations are used in the table on\npages 244-261 of th
e latter.\n\nThe Boston house-price data has been used in many machine learning pa
pers that address regression\nproblems.   \n     \n.. topic:: References\n\n   - B
elsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and So
urces of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993). Combining I
nstance-Based and Model-Based Learning. In Proceedings on the Tenth International
Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Mor
gan Kaufmann.\n",
 'filename': 'boston_house_prices.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [6]:
```
type(df)
```

Out[6]:
```
sklearn.utils.Bunch
```

In [7]:
```
dataset = pd.DataFrame(df.data)
print(dataset.head())
```

```
        0     1     2    3      4      5     6       7    8      9     10  \
0  0.00632  18.0  2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  15.3
1  0.02731   0.0  7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  17.8
2  0.02729   0.0  7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  17.8
3  0.03237   0.0  2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  18.7
4  0.06905   0.0  2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  18.7

       11    12
0  396.90  4.98
1  396.90  9.14
2  392.83  4.03
3  394.63  2.94
4  396.90  5.33
```

In [9]:
```
dataset.columns=df.feature_names
```

In [10]:
```
dataset.head()
```

Out[10]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LST |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4. |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9. |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4. |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2. |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5. |

In [11]:
```
df.target.shape
```

Out[11]:
```
(506,)
```

In [12]:
```
dataset["Price"]=df.target
```

In [13]:
```
dataset.head()
```

Out[13]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LST |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4. |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9. |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4. |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2. |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5. |

In [14]:
```
X=dataset.iloc[:,:-1] ## independent features
y=dataset.iloc[:,-1] ## dependent features
```

## Linear Regression

In [10]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
lin_regressor=LinearRegression()
mse=cross_val_score(lin_regressor,X,y,scoring='neg_mean_squared_error',cv=5)
mean_mse=np.mean(mse)
print(mean_mse)
```

```
-37.13180746769922
```

In [23]:
```
# Note- neg_mean_squared_error - if it is zero then consider as good
```

## Ridge Regression

In [15]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge=Ridge()
```

```
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=
ridge_regressor.fit(X,y)
```

Out[15]:
```
GridSearchCV(cv=5, estimator=Ridge(),
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.001, 0.01, 1, 5, 10,
                                   20, 30, 35, 40, 45, 50, 55, 100]},
             scoring='neg_mean_squared_error')
```

**What is GridSearchCV?**

**GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given model.**

**As mentioned above, the performance of a model significantly depends on the value of hyperparameters.**

**Note that there is no way to know in advance the best values for hyperparameters so ideally, we need to try all possible values to know the optimal values.**

**Doing this manually could take a considerable amount of time and resources and thus we use GridSearchCV to automate the tuning of hyperparameters**

In [45]:
```
print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 100}
-29.90570194754033
```

## Lasso Regression

In [16]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso=Lasso()
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
lasso_regressor=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=
lasso_regressor.fit(X,y)
print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.431e+03, tolerance: 3.919e+00
  model = cd_fast.enet_coordinate_descent(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.397e+03, tolerance: 3.307e+00
  model = cd_fast.enet_coordinate_descent(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 3.797e+03, tolerance: 2.814e+00
  model = cd_fast.enet_coordinate_descent(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.564e+03, tolerance: 3.307e+00
  model = cd_fast.enet_coordinate_descent(
C:\Users\Shreeji\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 4.294e+03, tolerance: 3.481e+00
  model = cd_fast.enet_coordinate_descent(
{'alpha': 1}
-35.531580220694856
```

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

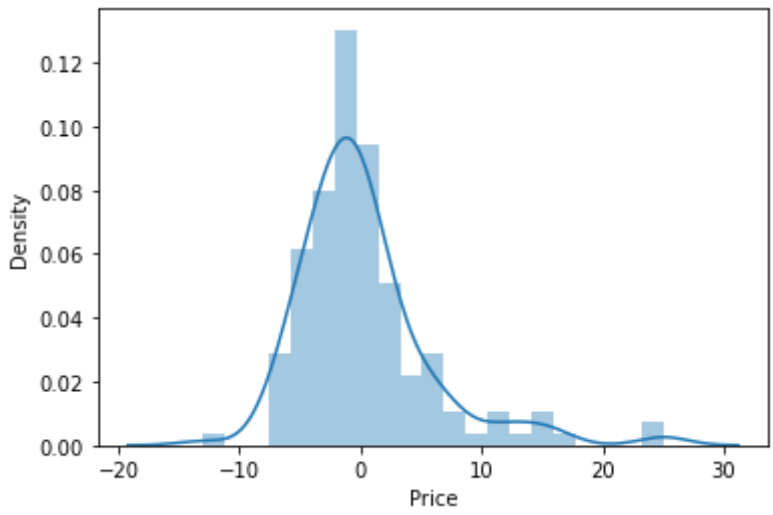
In [18]:
```
prediction_lasso=lasso_regressor.predict(X_test)
prediction_ridge=ridge_regressor.predict(X_test)
```

In [20]:
```
import seaborn as sns

sns.distplot(y_test-prediction_lasso)
```

```
C:\Users\Shreeji\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[20]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```




In [ ]:
```
import seaborn as sns
```

```
sns.distplot(y_test-prediction_ridge)
```

# Logistic Regression

## Handling Classification Problem

**What is the Classification Algorithm?**

**The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.**

**Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.**

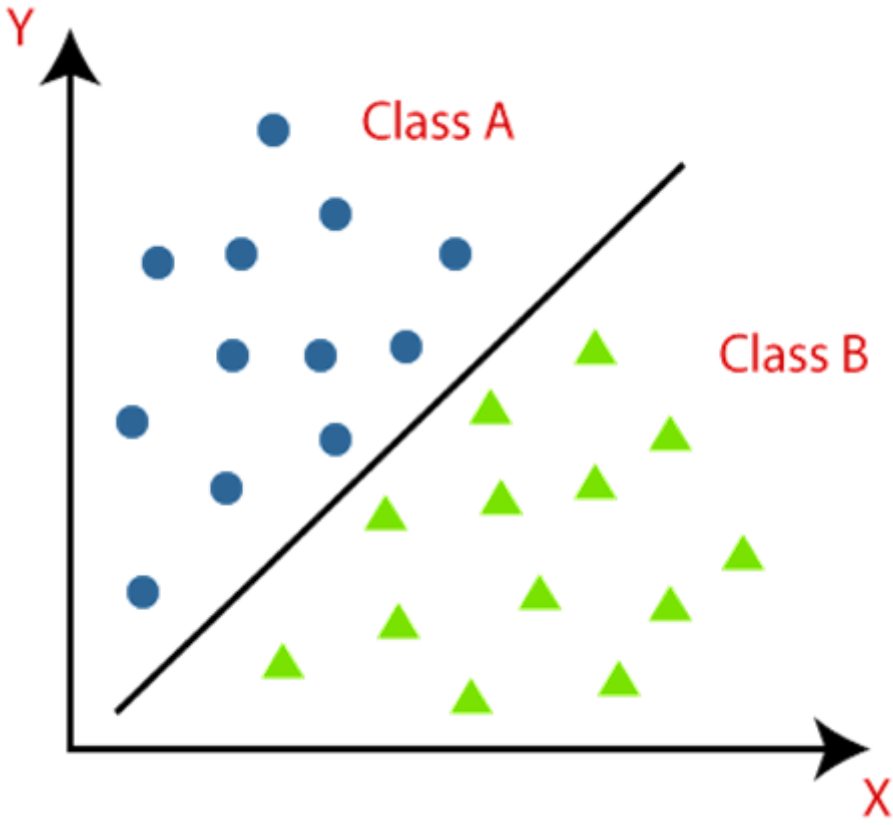
**In classification algorithm, a discrete output function(y) is mapped to input variable(x).**

**y=f(x), where y = categorical output**

**The best example of an ML classification algorithm is Email Spam Detector.**

**The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.**

**Classification algorithms can be better understood using the below diagram. In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.**




**The algorithm which implements the classification on a dataset is known as a classifier.**

**There are two types of Classifications:**

**Binary Classifier:**

**If the classification problem has only two possible outcomes, then it is called as Binary Classifier.**

**Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.**

**Multi-class Classifier:**

**If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.**

**Example:  Classifications of types of crops, Classification of types of music.**

**Types of ML Classification Algorithms:**

**Classification Algorithms can be further divided into the Mainly two category:**

**Linear Models**

- **Logistic Regression**
- **Support Vector Machines**

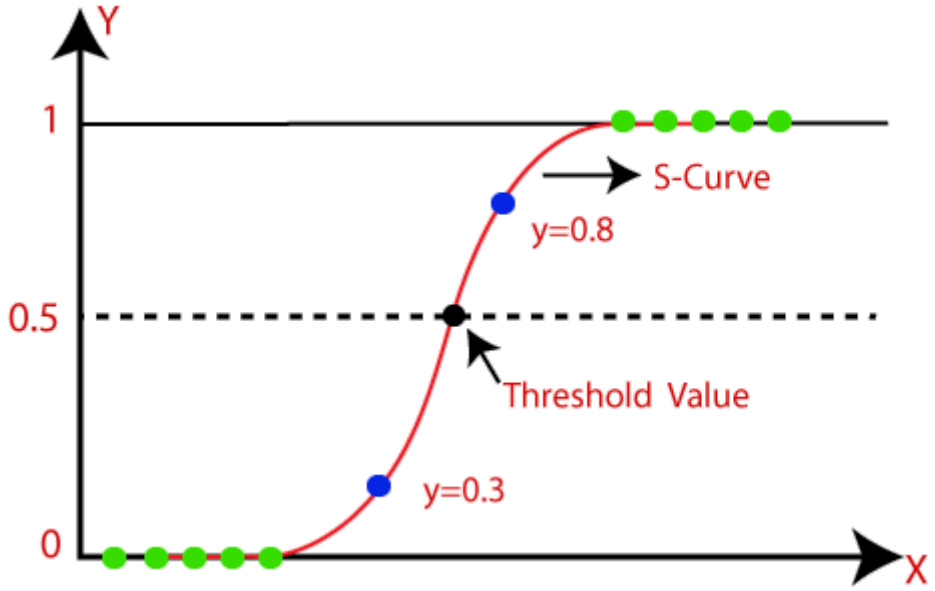
**Non-linear Models**

- **K-Nearest Neighbours**

- **Kernel SVM**
- **Naïve Bayes**
- **Decision Tree Classification**
- **Random Forest Classification**

# Logistic Regression

- **Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.**
- **Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.**
- **Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.**
- **In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).**
- **The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.**
- **Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.**
- **Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.**
- **The below image is showing the logistic function:**




**Note:**

**Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.**

**Logistic Function (Sigmoid Function):**

**The sigmoid function is a mathematical function used to map the predicted values to probabilities.**

**It maps any real value into another value within a range of 0 and 1.**

**The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form.**

**The S-form curve is called the Sigmoid function or the logistic function.**

**In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.**

**Assumptions for Logistic Regression:**

**The dependent variable must be categorical in nature.**

**The independent variable should not have multi-collinearity.**

**Logistic Regression Equation:**

$$S(x) = \frac{1}{1 + e^{-x}}$$

$S(x)$ = sigmoid function

$e$ = Euler's number

---

**Type of Logistic Regression:**

**On the basis of the categories, Logistic Regression can be classified into three types:**

**Binomial:**

**In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.**

**Multinomial:**

**In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"**

**Ordinal:**

**In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".**

**Example:**

**There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car.**

**So the company wanted to check how many users from the dataset, wants to purchase the car.**

**For this problem, we will build a Machine Learning model using the Logistic regression algorithm.**

**The dataset is shown in the below image.**

**In this problem, we will predict the purchased variable (Dependent Variable) by using age and salary (Independent variables).**

| User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|
| 15624510 | Male | 19 | 19000 | 0 |
| 15810944 | Male | 35 | 20000 | 0 |
| 15668575 | Female | 26 | 43000 | 0 |
| 15603246 | Female | 27 | 57000 | 0 |
| 15804002 | Male | 19 | 76000 | 0 |
| 15728773 | Male | 27 | 58000 | 0 |
| 15598044 | Female | 27 | 84000 | 0 |
| 15694829 | Female | 32 | 150000 | 1 |
| 15600575 | Male | 25 | 33000 | 0 |
| 15727311 | Female | 35 | 65000 | 0 |
| 15570769 | Female | 26 | 80000 | 0 |
| 15606274 | Female | 26 | 52000 | 0 |
| 15746139 | Male | 20 | 86000 | 0 |
| 15704987 | Male | 32 | 18000 | 0 |
| 15628972 | Male | 18 | 82000 | 0 |
| 15697686 | Male | 29 | 80000 | 0 |
| 15733883 | Male | 47 | 25000 | 1 |
| 15617482 | Male | 45 | 26000 | 1 |
| 15704583 | Male | 46 | 28000 | 1 |
| 15621083 | Female | 48 | 29000 | 1 |
| 15649487 | Male | 45 | 22000 | 1 |
| 15736760 | Female | 47 | 49000 | 1 |

**Steps in Logistic Regression:**

**To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:**

- **Data Pre-processing step**
- **Fitting Logistic Regression to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

**1. Data Pre-processing step:**

**In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.**

**It will be the same as we have done in Data pre-processing topic.**

**The code for this is given below:**

In [25]:
```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
#importing datasets
data_set= pd.read_csv('user_data.csv')
data_set
```

Out[25]:

| | User ID | Gender | Age | EstimatedSalary | Purchased |
|---|---|---|---|---|---|
| 0 | 15624510 | Male | 19 | 19000 | 0 |
| 1 | 15810944 | Male | 35 | 20000 | 0 |
| 2 | 15668575 | Female | 26 | 43000 | 0 |
| 3 | 15603246 | Female | 27 | 57000 | 0 |
| 4 | 15804002 | Male | 19 | 76000 | 0 |
| ... | ... | ... | ... | ... | ... |
| 395 | 15691863 | Female | 46 | 41000 | 1 |
| 396 | 15706071 | Male | 51 | 23000 | 1 |
| 397 | 15654296 | Female | 50 | 20000 | 1 |
| 398 | 15755018 | Male | 36 | 33000 | 0 |
| 399 | 15594041 | Female | 49 | 36000 | 1 |

**400 rows × 5 columns**

**Now, we will extract the dependent and independent variables from the given dataset.**

**Below is the code for it:**

In [26]:
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

**In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3.**

**And we have taken 4 for y variable because our dependent variable is at index 4.**

In [28]:
```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_st
```

**In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:**

In [29]:
```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In [32]:
```
x_test
```

Out[32]:
```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
       [ 0.08648817,  0.21503249],
       [-1.79512465,  0.47597078],
       [-0.60673761,  1.37475825],
       [-0.11157634,  0.21503249],
       [-1.89415691,  0.44697764],
       [ 1.67100423,  1.75166912],
       [-0.30964085, -1.37959044],
       [-0.30964085, -0.65476184],
       [ 0.8787462 ,  2.15757314],
       [ 0.28455268, -0.53878926],
       [ 0.8787462 ,  1.02684052],
       [-1.49802789, -1.20563157],
       [ 1.07681071,  2.07059371],
       [-1.00286662,  0.50496393],
       [-0.90383437,  0.30201192],
       [-0.11157634, -0.21986468],
       [-0.60673761,  0.47597078],
       [-1.6960924 ,  0.53395707],
       [-0.11157634,  0.27301877],
       [ 1.86906873, -0.27785096],
       [-0.11157634, -0.48080297],
       [-1.39899564, -0.33583725],
       [-1.99318916, -0.50979612],
       [-1.59706014,  0.33100506],
       [-0.4086731 , -0.77073441],
       [-0.70576986, -1.03167271],
       [ 1.07681071, -0.97368642],
       [-1.10189888,  0.53395707],
       [ 0.28455268, -0.50979612],
       [-1.10189888,  0.41798449],
       [-0.30964085, -1.43757673],
       [ 0.48261718,  1.22979253],
       [-1.10189888, -0.33583725],
       [-0.11157634,  0.30201192],
       [ 1.37390747,  0.59194336],
       [-1.20093113, -1.14764529],
       [ 1.07681071,  0.47597078],
       [ 1.86906873,  1.51972397],
       [-0.4086731 , -1.29261101],
       [-0.30964085, -0.3648304 ],
       [-0.4086731 ,  1.31677196],
       [ 2.06713324,  0.53395707],
       [ 0.68068169, -1.089659  ],
       [-0.90383437,  0.38899135],
       [-1.20093113,  0.30201192],
       [ 1.07681071, -1.20563157],
       [-1.49802789, -1.43757673],
       [-0.60673761, -1.49556302],
       [ 2.1661655 , -0.79972756],
       [-1.89415691,  0.18603934],
       [-0.21060859,  0.85288166],
```

```
[-1.89415691, -1.26361786],
[ 2.1661655 ,  0.38899135],
[-1.39899564,  0.56295021],
[-1.10189888, -0.33583725],
[ 0.18552042, -0.65476184],
[ 0.38358493,  0.01208048],
[-0.60673761,  2.331532  ],
[-0.30964085,  0.21503249],
[-1.59706014, -0.19087153],
[ 0.68068169, -1.37959044],
[-1.10189888,  0.56295021],
[-1.99318916,  0.35999821],
[ 0.38358493,  0.27301877],
[ 0.18552042, -0.27785096],
[ 1.47293972, -1.03167271],
[ 0.8787462 ,  1.08482681],
[ 1.96810099,  2.15757314],
[ 2.06713324,  0.38899135],
[-1.39899564, -0.42281668],
[-1.20093113, -1.00267957],
[ 1.96810099, -0.91570013],
[ 0.38358493,  0.30201192],
[ 0.18552042,  0.1570462 ],
[ 2.06713324,  1.75166912],
[ 0.77971394, -0.8287207 ],
[ 0.28455268, -0.27785096],
[ 0.38358493, -0.16187839],
[-0.11157634,  2.21555943],
[-1.49802789, -0.62576869],
[-1.29996338, -1.06066585],
[-1.39899564,  0.41798449],
[-1.10189888,  0.76590222],
[-1.49802789, -0.19087153],
[ 0.97777845, -1.06066585],
[ 0.97777845,  0.59194336],
[ 0.38358493,  0.99784738]])
```

**2. Fitting Logistic Regression to the Training set:**

**We have well prepared our dataset, and now we will train the dataset using the training set.**

**For providing training or fitting the model to the training set, we will import the LogisticRegression class of the sklearn library.**

**After importing the class, we will create a classifier object and use it to fit the model to the logistic regression.**

**Below is the code for it:**

In [33]:
```
#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

Out[33]:
```
LogisticRegression(random_state=0)
```

**3. Predicting the Test Result**

**Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:**

In [34]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

**In the above code, we have created a y_pred vector to predict the test set result.**

**4. Test Accuracy of the result**

**Now we will create the confusion matrix here to check the accuracy of the classification.**

**To create it, we need to import the confusion_matrix function of the sklearn library.**

**After importing the function, we will call it using a new variable cm.**

**The function takes two parameters, mainly y_true( the actual values) and y_pred (the targeted value return by the classifier).**

**Below is the code for it:**

In [21]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,y_pred)
cm
```

Out[21]:
```
array([[65,  3],
       [ 8, 24]], dtype=int64)
```

**We can find the accuracy of the predicted result by interpreting the confusion matrix.**

**By above output, we can interpret that 65+24= 89 (Correct Output) and 8+3= 11(Incorrect Output).**

# Numerical Data Scaling Methods

**Both normalization and standardization can be achieved using the scikit-learn library.**

**Let's take a closer look at each in turn.**

**Data Normalization**

**Normalization is a rescaling of the data from the original range so that all values are within the new range of 0 and 1.**

**A value is normalized as follows:**

**y = (x – min) / (max – min)**

**For example, for a dataset, we could guesstimate the min and max observable values as 30 and -10. We can then normalize any value, like 18.8, as follows:**

- **y = (x – min) / (max – min)**
- **y = (18.8 – (-10)) / (30 – (-10))**
- **y = 28.8 / 40**
- **y = 0.72**

In [6]:
```
# example of a normalization
import numpy as np
from sklearn.preprocessing import MinMaxScaler
# define data
data = np.array([[100, 0.001],
[8, 0.05],
[50, 0.005],
[88, 0.07],
[4, 0.1]])
print(data)
# define min max scaler
scaler = MinMaxScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```

```
[[1.0e+02 1.0e-03]
 [8.0e+00 5.0e-02]
 [5.0e+01 5.0e-03]
 [8.8e+01 7.0e-02]
 [4.0e+00 1.0e-01]]
[[1.         0.        ]
 [0.04166667 0.49494949]
 [0.47916667 0.04040404]
 [0.875      0.6969697 ]
 [0.         1.        ]]
```

**Data Standardization**

**Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1.**

**This can be thought of as subtracting the mean value or centering the data.**

**A value is standardized as follows:**

**y = (x – mean) / standard_deviation**

**We can guesstimate a mean of 10.0 and a standard deviation of about 5.0. Using these values, we can standardize the first value of 20.7 as follows:**

- **y = (x – mean) / standard_deviation**
- **y = (20.7 – 10) / 5**
- **y = (10.7) / 5**

- **y = 2.14**

In [9]:
```
# example of a standardization
import numpy as np
from sklearn.preprocessing import StandardScaler
# define data
data = np.array([[100, 0.001],
[8, 0.05],
[50, 0.005],
[88, 0.07],
                 [4, 0.1]])
print(data)
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```

```
[[1.0e+02 1.0e-03]
 [8.0e+00 5.0e-02]
 [5.0e+01 5.0e-03]
 [8.8e+01 7.0e-02]
 [4.0e+00 1.0e-01]]
[[ 1.26398112 -1.16389967]
 [-1.06174414  0.12639634]
 [ 0.         -1.05856939]
 [ 0.96062565  0.65304778]
 [-1.16286263  1.44302493]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

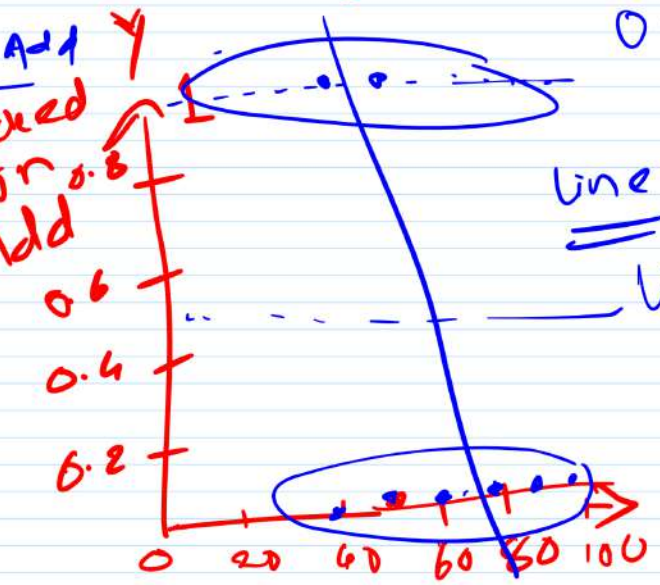
In [ ]:

Logistic Regression:→

→ I/p : continuous

→ O/p : categorical

| x Time | y clicked or Add |
|---|---|
| 68.95 | NO |
| 80.23 | NO |
| 69.45 | NO |
| 74.15 | NO |
| 50.0 | yes |
| 55.5 | yes |
| 80.0 | NO |
| 40.5 | NO |




$y_c = 1 + \frac{1}{1+e^{-x}}$

Euler's Rule

Z-score

0 ..... to ..... 1

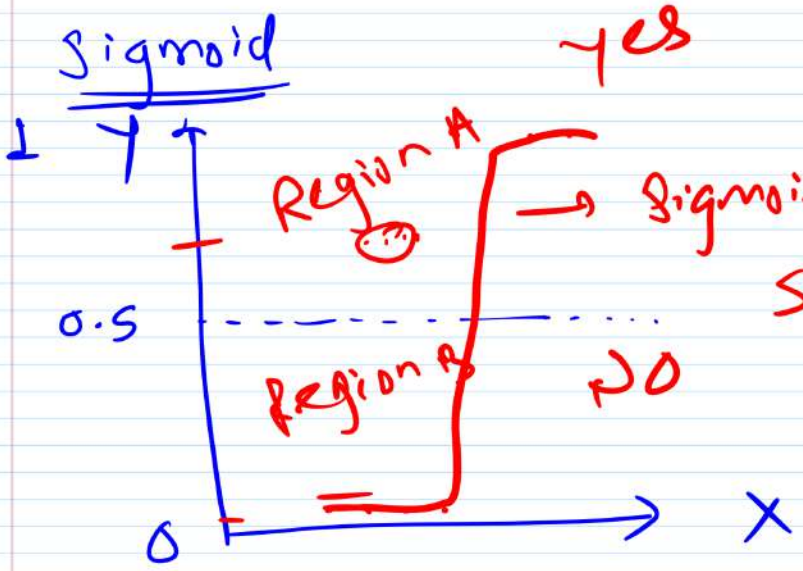
0 - NO

1 - Yes

line X

Linear Reg is not working as O/p is categorical

Sigmoid




Binary classificati

New obsa

0.7 yes

0.4 → NO

## Linear Regression:

$y = mx + c$

m → Slope, c → intercept

$$\text{cost} = \frac{1}{m}\sum_{i=0} |\hat{y} - y|$$

$\min_m \text{cost}$

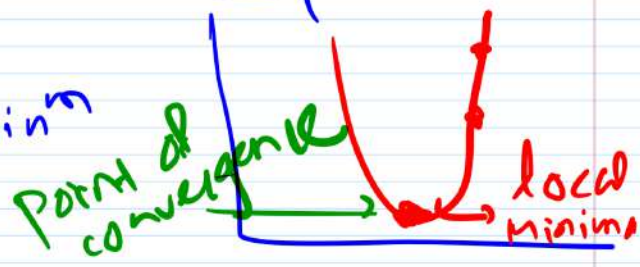
$y = w_n x_n + w_{n-1} x_{n-1} + \cdots + w_2 x_2 + w_1 x_1 + b$

$y = w^T x + b$

$\{ y = w^T x + b \}$ → Straight line / linear Regression eqn

Gradient Descent

⇒ which is having min cost



## Logistic Regression: →

⇒ Cost fn :- It is representation of Error.

⇒ It shows how our model is predicting compared to original data set

* Cost function ↓ Accuracy ↑

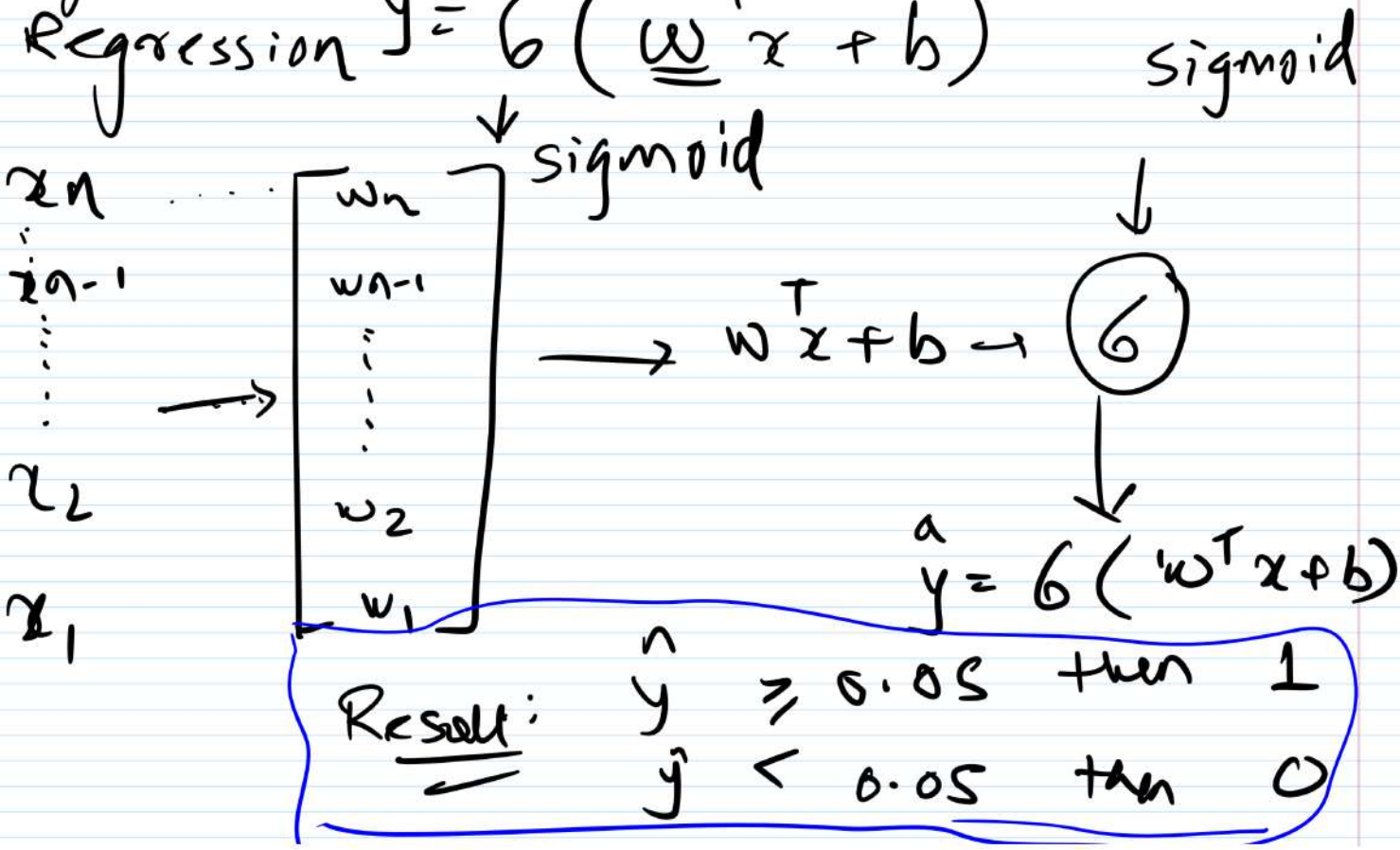
Cost function ↑ Accuracy ↓

Logistic Regression: →

LR → $y = w^T x + b$

Logistic Regression $\hat{y} = \sigma(w^T x + b)$

sigmoid




Result: $\hat{y} \geq 0.05$ then 1

$\hat{y} < 0.05$ then 0

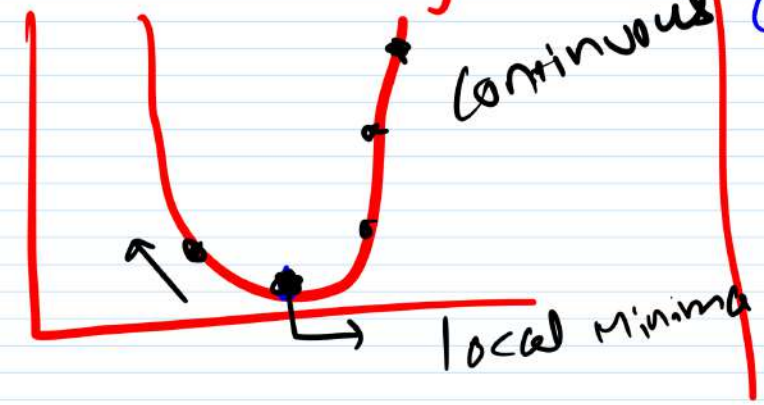
* How to compute cost in logistic Regression:

① Linear Regression → continuous

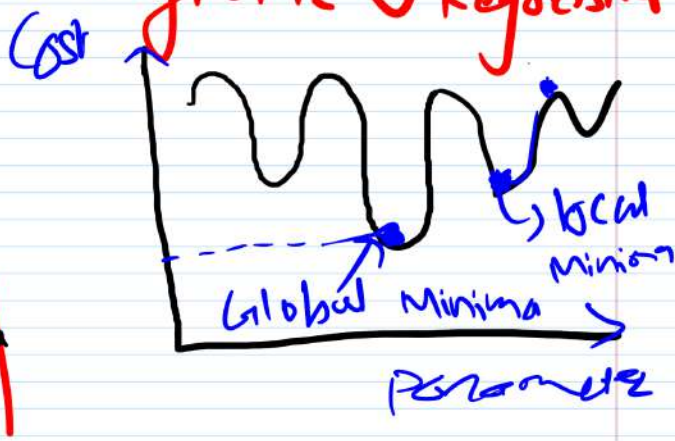
Cost : $\frac{1}{m}\sum_{i=0}^{m} |\hat{y} - y|$ => linear Regression

② $y = 0/1$ => Categorical => logistic Regression

Linear Regression




Logistic Regression



Logistic Regression:

$$\text{Cost} = -\frac{1}{m}\sum_{i=1}^{m}\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

Cost = average

O/p → yes → 1

O/p → no → not 1

$0.2 + 0.8 = 1$

$y \quad 1-y$

Sigmoid curve

$y \quad 1-y$

$P_1 = 0.4 \quad P_2 = 0.6$

$P_1 + P_2 = 1$


mahendra_ugale@csmssengg.org

# DISCLAIMER

The material for the presentation has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this lecture/presentation is for general information and education purpose only. While we endeavor to keep the information up to date and correct, we make no representation of any kind about the completeness and accuracy of the material. The information shared through this presentation material should be used for educational purpose only.