Logistic Regression:

=> Model type :- Classification Model.

=> Equation :- $Y = A + BX$

- B → Slop
- A → Intercept

OR $Y = BX$

Equition in ML term :- $h = \theta X$ → Predictor

h → Hypothesis or predicted value

Note: θ is initelized randomly at the begining and we update it later.

=> Hypothesis function / Sigmond function / Logistic function.

- Returns value between 0 to 1.
- LR uses the sigmond function to predict the outcome.

$$h = \frac{1}{1 + e^{-z}}$$

z = I/P feature multiplied by randomly initelized term θ.

$z = \theta X$ → I/P features

Note: We use sigmond function as predictorg fun because it returns value between 0 to 1 and it is very useful in LR (Classification).

Return 0 if LR fun < 0.5

Return 1 if LR fun ≥ 0.5

⇒ Cost function

- Cost function is useful to determine how far the predicted output is from the original out put.

Note:- Here in Logistic Regression we can not use simple linear cost fun we use in Linear Regression because in Log.R we use sigmond fun and it is not a linear fun. And also simple cost function will not converge to global minima.

- Hence to overcome this situation we will use "log" to regularize the cost function.
- simplified & Combined cost fun for LR.

$$J = \frac{1}{m}\left(\sum_{i=0}^{m} y \cdot \log(h) + (1-y) \cdot \log(1-h)\right)$$

Note: If $y=0$ then the first term becomes 0.
If $y=1$ then the second term becomes 0.

⇒ Gradient Descent

- Gradient Descent is used to update randomly initalized θ values.

$$\theta = \theta - \alpha \sum_{i=0}^{m} (h-y)x_j$$

↳ Learning Rate.

=> Model Development.

Step 1: Develop hypothesis / Sigmond function.

Code:

```
def hypothesis (x, theta):
    z = np.dot (theta, x.T)
    return (1/ 1+np.exp(-(z))) - 0.0000001
```

Note: Here we deduct small num from O/P because if outcome of hypothesis comes out to b 1 then this expression will return log(0), which is 0.

Step 2: Determine the Cost function.

This step is just straight forward implementation of cost function equation.

Code:

```
def cost (X, y, theta):
    y1 = hypothesis (X, theta)
    return -(1/len(x)) * np.sum (y * np.log(y1)
           + (1-y) * np.log(1-y1))
```

Step 4: Update θ values.

θ value needs to be keep updating untill the cost function reaches its minimum. In this fun we should reaturn final θ values and cost of each iteration.

Code:

```
def gradient_decent(X, y, theta, alpha, epochs):
    m = len(X)
    J = [cost(X, y, theta)]
    for i in range(0, epochs):
        h = hypothesis(X, theta)
        for i in range(0, len(X.columns)):
            theta[i] -= (alpha/m) * np.sum((h-y) *
                        X.iloc[:, i])
        J.append(cost(X, y, theta))
    return(J, theta)
```

Step 4: Calculate the final predection and Accuracy.

- In this step we use the theta values that comes out of gradient-decent function and calculate the final pred.

Code:

```
def predict(X, y, theta, alpha, epochs):
    J, th = gradient_decent(X, y, theta, alpha, epochs)
    h = hypothesis(X, theta)
    for i in range(len(h)):
        h[i] = 1 if h[i] >= 0.5 else 0
    y = list(y)
    acc = np.sum([y[i] == h[i] for i in range(len(y))]) /
                                        len(y)
    return J, acc
```