BIOSOC

SUMMER PROJECT

BIOBYTES

ML IN BIOLOGY

MENTORS: NISCHAY PATEL AND SIKHA VAMSI

Group 4:
Md. Waqar Moid
Sritama Sarkar
Bhavnoor Singh

LINEAR REGRESSION

- Linear regression is a supervised machine-learning algorithm that learns from the labelled datasets and maps the data points to the most optimized linear functions.
- When there is only one independent feature, it is known as Simple Linear Regression, and when there are more than one feature, it is known as Multiple Linear Regression.
- Linear regression is classified as a regression algorithm because it is designed to predict a continuous outcome (dependent variable) based on one or more input features (independent variables).

Equation of simple linear regression

$$y = a + bx$$

Where,

x and *y* are two variables on the regression line.

b = Slope of the line.

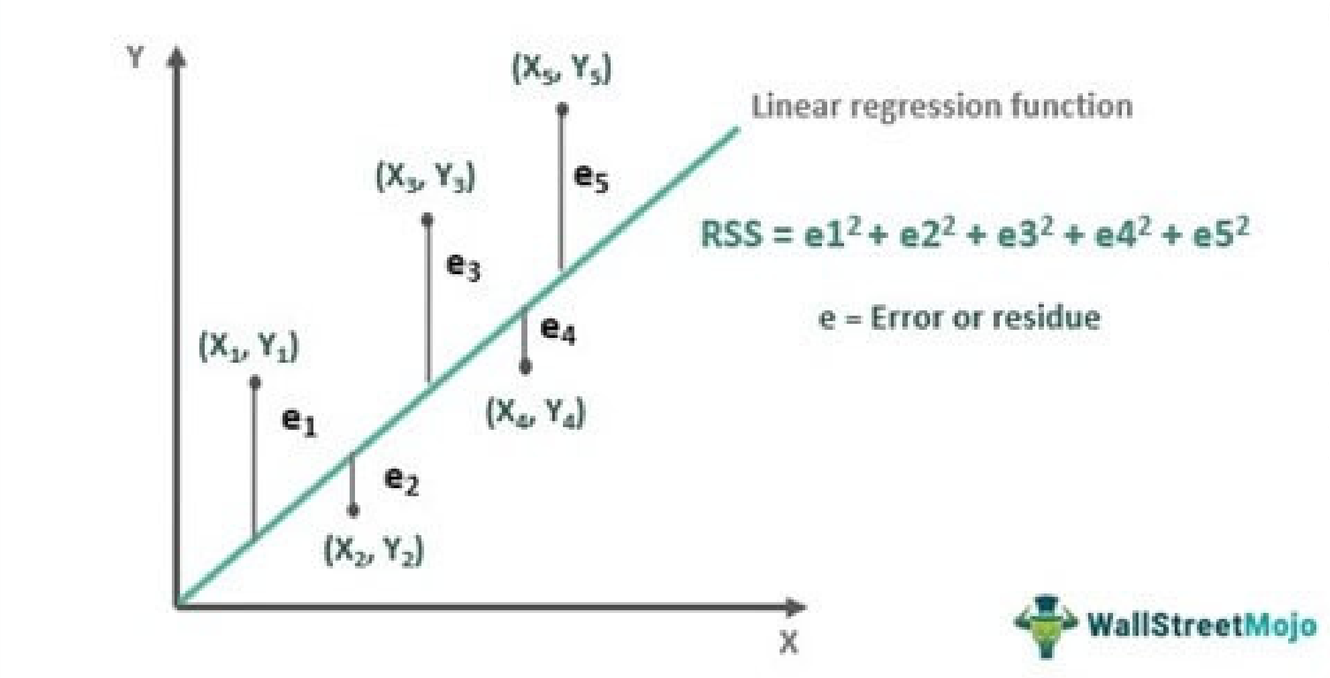
a = *y*-intercept of the line.

x = Values of the first data set.

y = Values of the second data set.

LINE OF BEST-FIT

- A **Line of best fit** is a straight line that represents the best approximation of a scatter plot of data points. It is used to study the nature of the relationship between those points.



$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- The **sum of squares distance minimization** refers to the process of finding the best-fitting line through a set of data points by minimizing the sum of the squared differences (distances) between the observed values and the values predicted by the line.

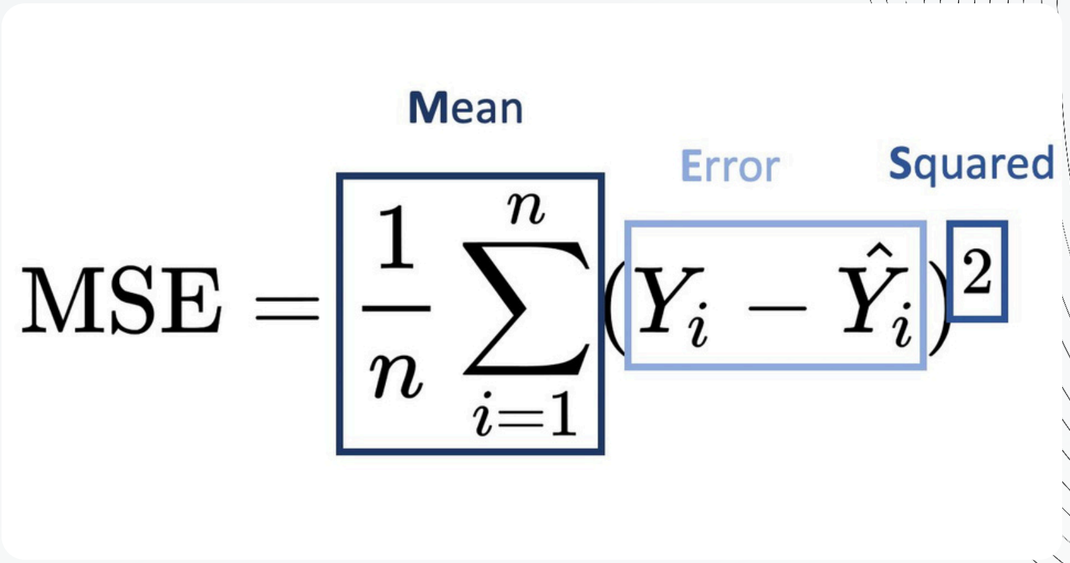
COST FUNCTION

- Note that, if y = a+bx is our model's predicted Best-Fit Line for some x, and y' is the actual values, then it is intuitive to say that y'-y is the error in the prediction.

- By observation, we can say that this error can be positive as well as negative, implying that the predicted value can be more or less than the actual value, and our goal is to minimize the gap between the predicted and actual values.

- As a result, we use the squared errors as a measure to figure out the gap, rather than simply the difference between them.

- Therefore, our cost function for Linear Regression (generally) is MSE or Mean Squared Error an our target is to minimize this cost function.

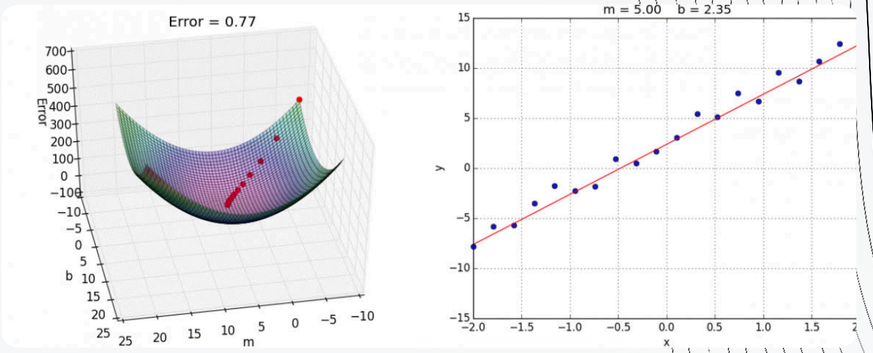


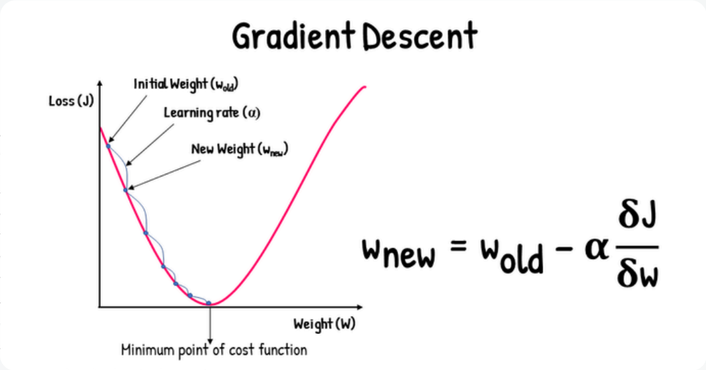
HOW TO MINIMIZE THE COST FUNCTION?

#Technique 1

GRADIENT DESCENT

Gradient Descent is one of the optimization algorithms that optimize the cost function (objective function) to reach the optimal minimal solution. To find the optimum solution, we need to reduce the cost function (MSE) for all data points.





A regression model optimizes the gradient descent algorithm to update the coefficients of the line by reducing the cost function by randomly selecting coefficient values and then iteratively updating the coefficient values to reach the minimum cost function as shown in the images.

#Technique 1

GRADIENT DESCENT

$y(x) = a + b*x$ *(Hypothesis or predicted function)*

$J = \frac{1}{2m}\sum_{i=1}^{m}(y(x) - y')^2$ (Cost Function)

Gradient descent formula for a

$$\frac{\partial J}{\partial a} = \frac{1}{m}\sum_{i=1}^{m}(y(x) - y')\frac{\partial y(x)}{\partial a} = \frac{1}{2m}\sum_{i=1}^{m}(y(x) - y') * 1$$

$$=> a_{new} = a_{old} - \alpha\frac{\partial J}{\partial a} = a_{old} - \alpha * \frac{1}{m}\sum_{i=1}^{m}(y(x) - y')$$

Gradient descent formula for b

$$\frac{\partial J}{\partial b} = \frac{1}{2m}\sum_{i=1}^{m}(y(x) - y')\frac{\partial y(x)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(y(x) - y') * b$$

$$=> b_{new} = b_{old} - \alpha\frac{\partial J}{\partial b} = b_{old} - \alpha * \frac{1}{m}\sum_{i=1}^{m}(y(x) - y') * b$$

#Technique 2

LINEAR ALGEBRA (NORMAL METHOD)

The problem of minimizing the cost funciton can be alternatively written as:

$$min \left\| \underbrace{\begin{bmatrix} y'(x_1) \\ y'(x_2) \\ \dots \\ y'(x_n) \end{bmatrix}}_{Y} - \underbrace{\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \\ 1 & x_n \end{bmatrix}}_{X} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2$$

We get the solution to the above Least Squares Problem via linear algebraic techniques as:

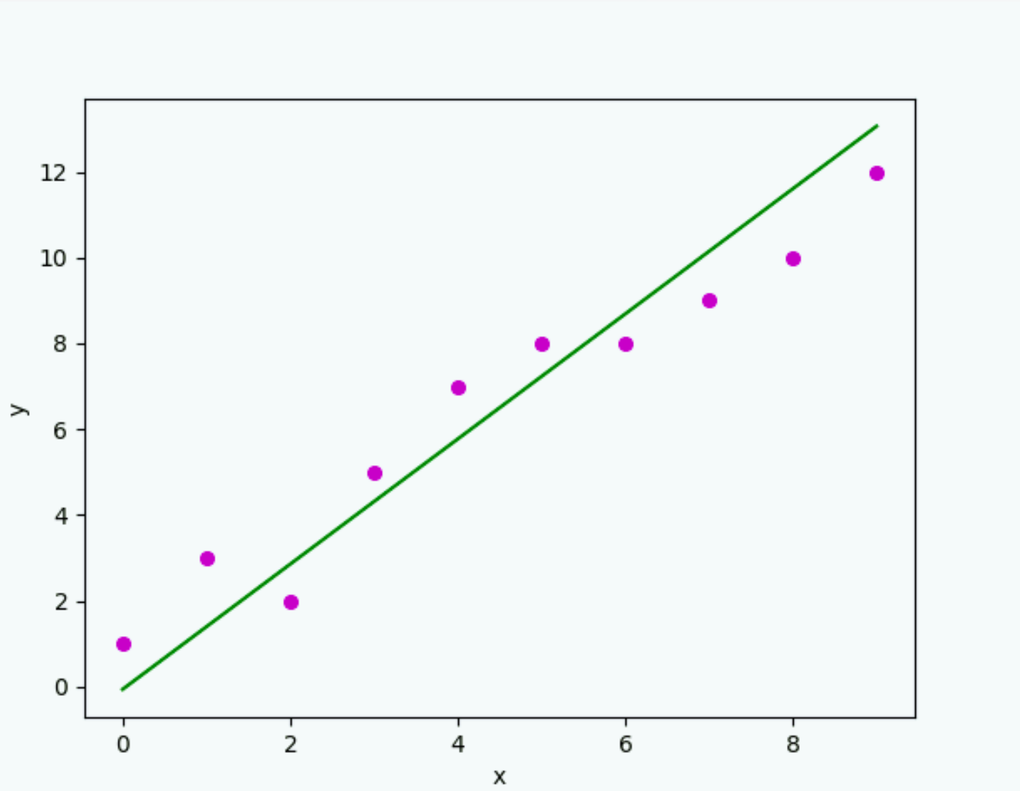
$$\begin{bmatrix} a \\ b \end{bmatrix} = (X^T X)^{-1} X^T Y$$

MODEL IMPLEMENTATION

BUILDING MODEL FROM SCRATCH

```
def estimate_coef(x, y):
n = np.size(x)
m_x = np.mean(x)
m_y = np.mean(y)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
return (b_0, b_1)
def plot_regression_line(x, y, b):
  plt.scatter(x, y, color = "m",
        marker = "o", s = 30)
y_pred = b[0] + b[1]*x
plt.plot(x, y_pred, color = "g")
plt.xlabel('x')
plt.ylabel('y')
```

OUTPUT : REGRESSION LINE



MODEL IMPLEMENTATION

USING SCIKIT LEARN LIBRARY

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> # y = 1 * x_0 + 2 * x_1 + 3
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.score(X, y)
1.0
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

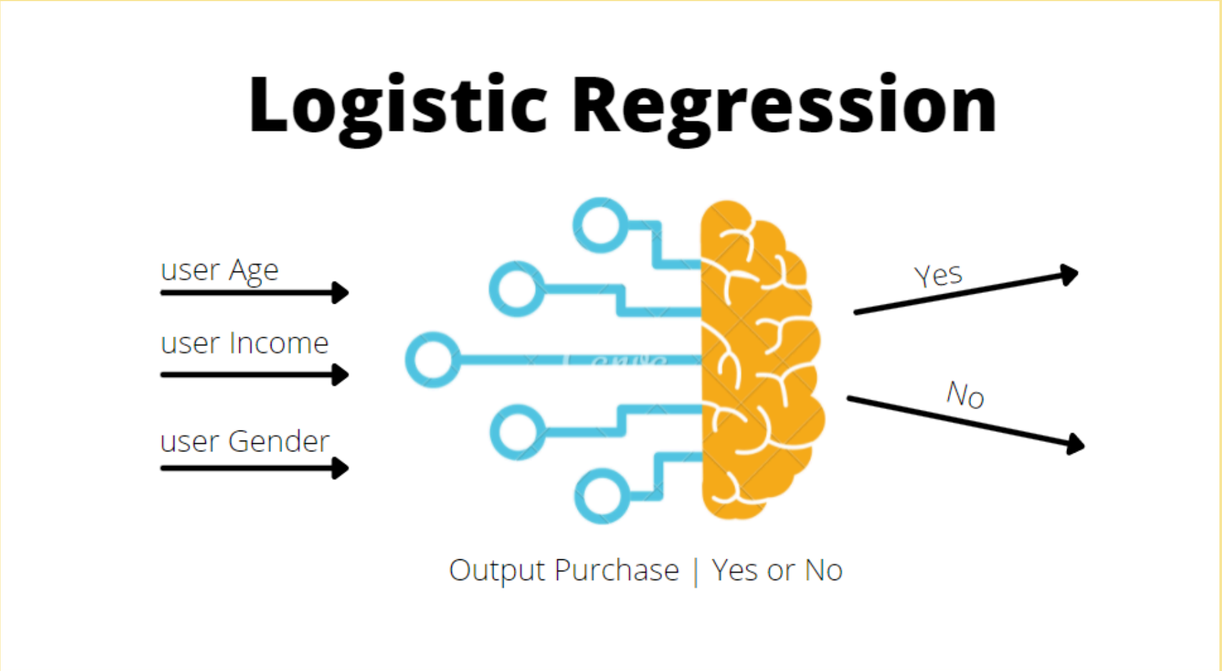
UNDERSTANDING LOGISTIC REGRESSION

WHAT IS LOGISTIC REGRESSION ?

A foundational technique in predictive modelling that bridges the gap between simple linear models and complex neural networks in deep learning. It is used to describe data and to explain the relationship between dependent binary variables and one or more nominal, ordinal, interval or ratio-level independent variables.

WHEN TO USE?

Appropriate for conducting when the dependent variable is dichotomous (binary). It just means that a variable that has only two outputs can either be yes or no.

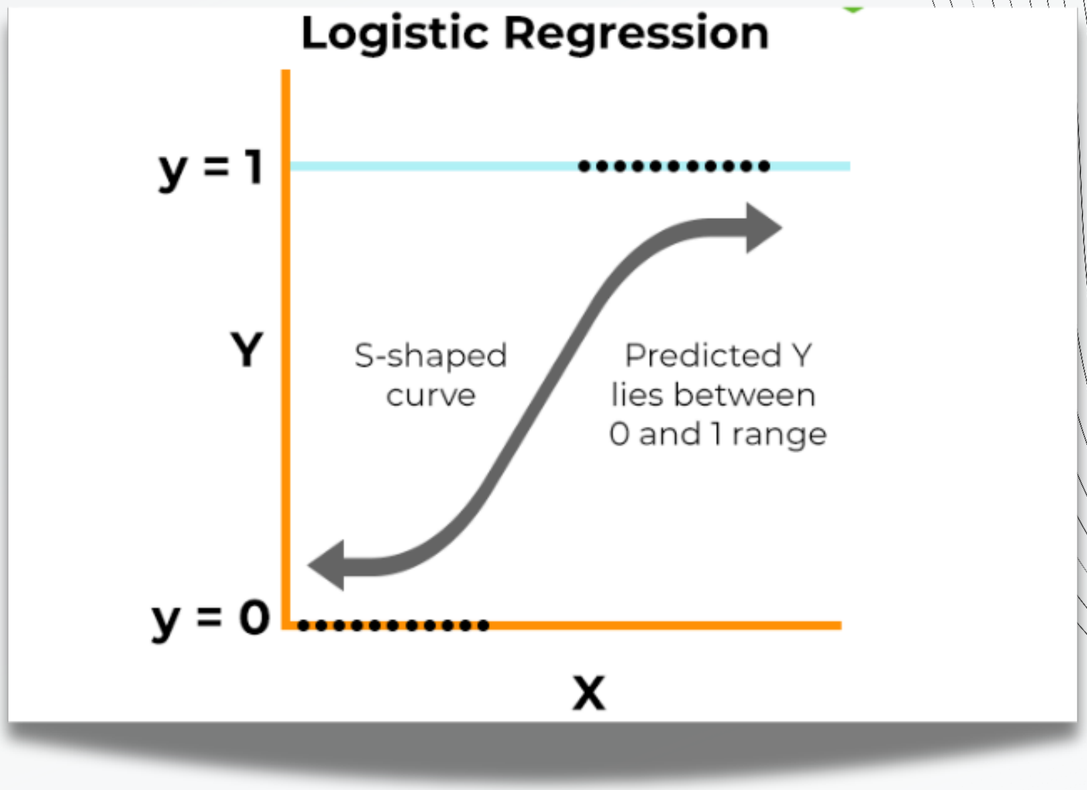


TYPES:

Binary Logistic Regression

Ordinal Logistic Regression

Multinomial Logistic Regression



WHY USE LOGISTIC REGRESSION?

Versatility: Can handle binary, ordinal, and multinomial outcomes

Interpretability: Provides probabilities for outcomes, making it intuitive.

Applications: Widely used in fields such as medical research, finance, and marketing

THE LOGISTIC FUNCTION

Formula:

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Cost Function:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

The logistic function (also called the sigmoid function) maps any real-valued number into a value between 0 and 1.

MODEL IMPLEMENTATION

```python
def model(X, Y, learning_rate, iterations):

    m = X_train.shape[1]
    n = X_train.shape[0]
    W = np.zeros((n,1))
    B = 0

    cost_list = []

    for i in range(iterations):

        Z = np.dot(W.T, X) + B
        A = sigmoid(Z)
        cost = -(1/m)*np.sum( Y*np.log(A) + (1-Y)*np.log(1-A))
        dW = (1/m)*np.dot(A-Y, X.T)
        dB = (1/m)*np.sum(A - Y)
        W = W - learning_rate*dW.T
        B = B - learning_rate*dB

        cost_list.append(cost)

        if(i%(iterations/10) == 0):
            print("cost after ", i, "iteration is : ", cost)

    return W, B, cost_list
```

$$W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_n \end{bmatrix}_{nx1}$$ ..... initialize with zeros

$B = single\ weight/parameter$

$$X = \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}_{nxm}$$

$$Y = [. \quad . \quad . \quad . \quad .]_{1xm}$$

$\sigma = \frac{1}{(1+e^{-x})}$ ..... (sigmoid function)

A = $\sigma(W^T * X + b)$ ..... (probabilistic predictions of shape (1 x m))

Cost function :

$$cost = -\frac{1}{m}\sum_{i=1}^{m}[y * log(a) + (1-y) * log(1-a)]$$

Gradient Descent

$dW = \frac{\partial COST}{\partial W} = (A-Y) * X^T$ ..... shape (1 x n)

$dB = \frac{\partial COST}{\partial B} = (A-Y)$

$W = W - \alpha * dW^T$

$B = B - \alpha * dB$

```
iterations = 100000
learning_rate = 0.0015
W, B, cost_list = model(X_train, Y_train, learning_rate = learning_rate, iterations = iterations)
```

```
cost after  0 iteration is :  0.6931471805599454
cost after  10000 iteration is :  0.4965277769389531
cost after  20000 iteration is :  0.46674868550665993
cost after  30000 iteration is :  0.45687787762434423
cost after  40000 iteration is :  0.45288994293089646
cost after  50000 iteration is :  0.4509326025222643
cost after  60000 iteration is :  0.44977087490094686
cost after  70000 iteration is :  0.4489640829216279
cost after  80000 iteration is :  0.44834126966124827
cost after  90000 iteration is :  0.44783045246935776
```

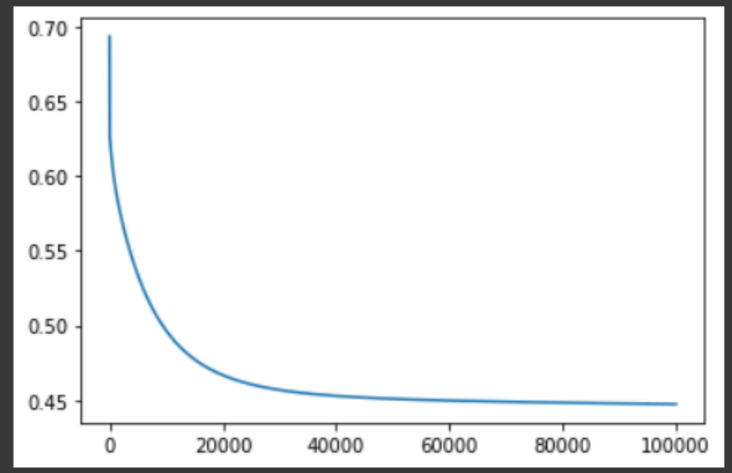
MODEL IMPLEMENTATION

(Using Titanic Dataset)

Cost vs Iteration

Plotting graph to see if Cost Function is decreasing or not

```
[ ]  plt.plot(np.arange(iterations), cost_list)
     plt.show()
```

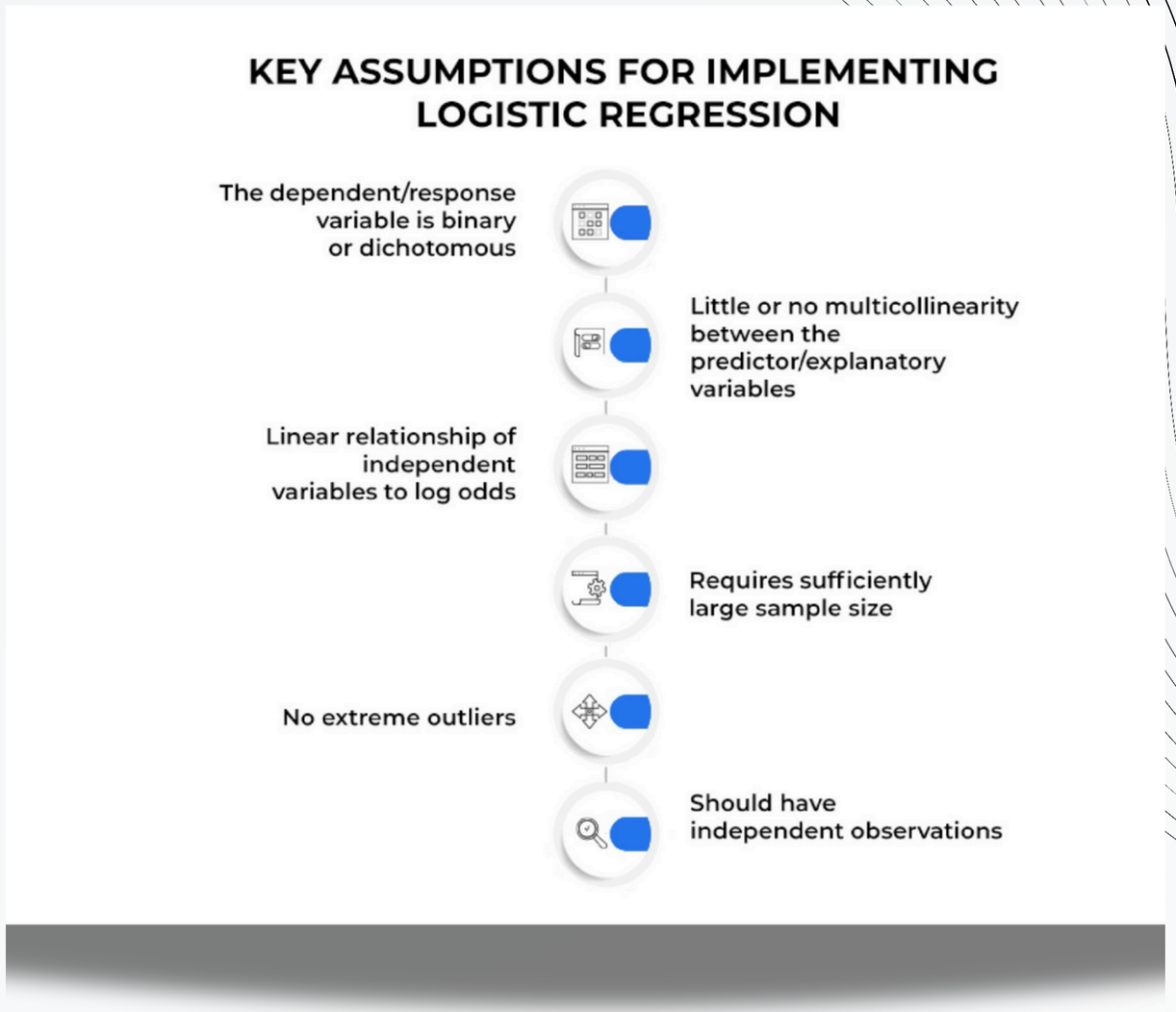


ASSUMPTIONS:

Data Specific: Binary dependent variable, independent observations

Relationship Between Variables: Linearity in the logit, no multicollinearity

Others: Absence of outliers, adequate sample size



MODEL EVALUATION:

(Using Diabetes Dataset)

Confusion Matrix Method:

```
# evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Output:

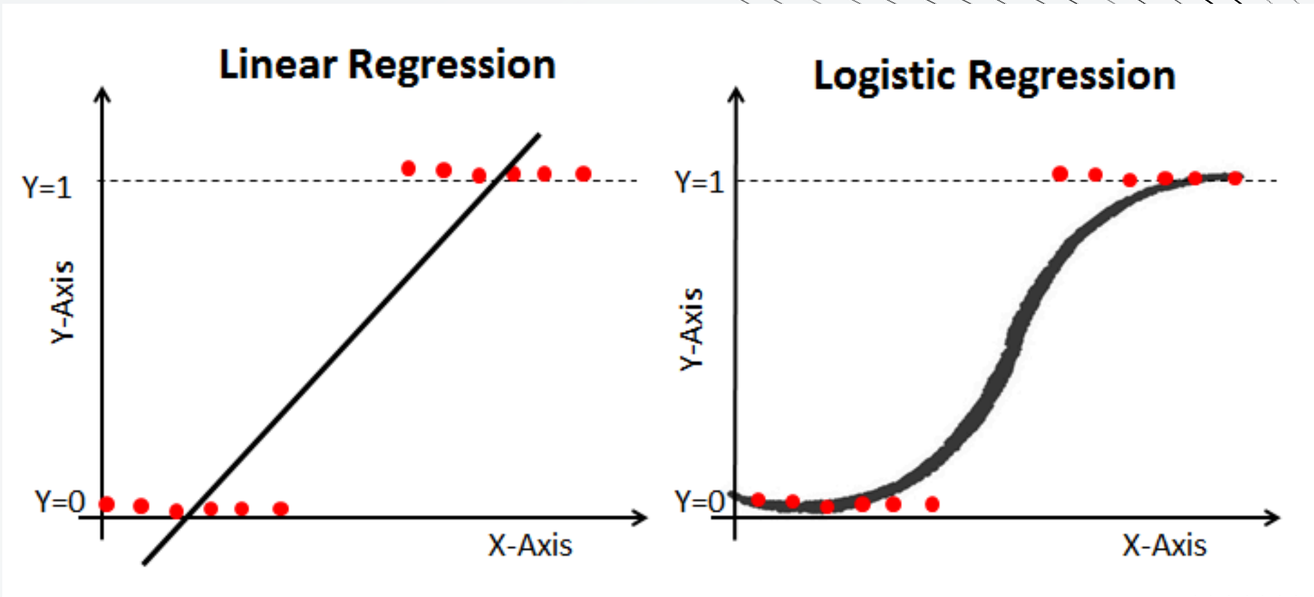
```
Confusion Matrix:
 [[36 13]
 [11 29]]

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        49
           1       0.69      0.72      0.71        40

    accuracy                           0.73        89
   macro avg       0.73      0.73      0.73        89
weighted avg       0.73      0.73      0.73        89
```

ADVANTAGES AND LIMITATIONS:



Advantages (Logistic Regression vs. Linear Regression): Linear Regression excels at predicting continuous values along a spectrum. Logistic Regression, on the other hand, deals with categories. It doesn't predict a specific value but rather the likelihood of something belonging to a particular class.

Limitations:

- Assumes linear relationship between logit and predictors.
- Sensitive to outliers and multicollinearity.
- It is tough to obtain complex relationships using logistic regression. More powerful and compact algorithms such as Neural Networks can easily outperform this algorithm

CONCLUSION:

Logistic regression is a powerful tool for handling categorical variables and predicting binary outcomes. Unlike decision trees, which create non-linear decision boundaries, logistic regression uses a linear relationship transformed by the exponential function through an activation function.

KEY TAKEAWAYS:

- Validation is a key step to ensure the model's performance on unseen data, by splitting data
- The sklearn library in Python provides robust tools for implementing logistic regression models.
- The decision boundary in logistic regression is a linear separator, unlike decision trees, which create non-linear boundaries.
- Overfitting can occur in logistic regression, it's crucial to employ techniques such as regularization to mitigate this issue.
- Logistic regression can be viewed as a linear model where the output is passed through an activation function, specifically the sigmoid function.

Thank You!!