**Logistic Regression:**

$$\hat{Y} = \frac{1}{1 + e^{-Z}} \qquad\qquad Z = w.X + b$$

Y_hat --> predicted value

X --> Input Variable

w --> weight

b --> bias

**Gradient Descent:**

Gradient Descent is an optimization algorithm used for minimizing the loss function in various machine learning algorithms. It is used for updating the parameters of the learning model.

w = w - α*dw

b = b - α*db

**Learning Rate:**

Learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

**Derivatives:**

$$dw = \frac{1}{m} * (\hat{Y} - Y).X$$

$$db = \frac{1}{m} * (\hat{Y} - Y)$$

Importing the Dependencies

In [1]:
```
# importing numpy library
import numpy as np
```

**Logistic Regression**

In [2]:
```
class Logistic_Regression():


  # declaring learning rate & number of iterations (Hyperparametes)
  def __init__(self, learning_rate, no_of_iterations):

    self.learning_rate = learning_rate
    self.no_of_iterations = no_of_iterations



  # fit function to train the model with dataset
  def fit(self, X, Y):

    # number of data points in the dataset (number of rows)  -->  m
    # number of input features in the dataset (number of columns)  --> n
    self.m, self.n = X.shape


    #initiating weight & bias value

    self.w = np.zeros(self.n)

    self.b = 0

    self.X = X

    self.Y = Y


    # implementing Gradient Descent for Optimization

    for i in range(self.no_of_iterations):
      self.update_weights()



  def update_weights(self):

    # Y_hat formula (sigmoid function)

    Y_hat = 1 / (1 + np.exp( - (self.X.dot(self.w) + self.b ) ))


    # derivaties

    dw = (1/self.m)*np.dot(self.X.T, (Y_hat - self.Y))

    db = (1/self.m)*np.sum(Y_hat - self.Y)


    # updating the weights & bias using gradient descent

    self.w = self.w - self.learning_rate * dw

    self.b = self.b - self.learning_rate * db
```

```
  # Sigmoid Equation & Decision Boundary

  def predict(self, X):

    Y_pred = 1 / (1 + np.exp( - (X.dot(self.w) + self.b ) ))
    Y_pred = np.where( Y_pred > 0.5, 1, 0)
    return Y_pred
```

Importing the Dependencies

In [3]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Data Collection and Analysis

PIMA Diabetes Dataset

In [4]:
```
# loading the diabetes dataset to a pandas DataFrame
diabetes_dataset = pd.read_csv(r'C:\Users\NITHISH\Desktop\DATA SCIENCE\NITISH-ML\
```

In [5]:
```
# printing the first 5 rows of the dataset
diabetes_dataset.head()
```

Out[5]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | A |
|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | |

In [6]:
```
# number of rows and Columns in this dataset
diabetes_dataset.shape
```

Out[6]: (768, 9)

In [7]:
```
# getting the statistical measures of the data
diabetes_dataset.describe()
```

Out[7]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPe |
|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | |

In [8]:
```
diabetes_dataset['Outcome'].value_counts()
```

Out[8]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

0 --> Non-Diabetic

1 --> Diabetic

In [9]:
```
diabetes_dataset.groupby('Outcome').mean()
```

Out[9]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | Diabetes |
|---|---|---|---|---|---|---|---|
| **Outcome** | | | | | | | |
| **0** | 3.298000 | 109.980000 | 68.184000 | 19.664000 | 68.792000 | 30.304200 | |
| **1** | 4.865672 | 141.257463 | 70.824627 | 22.164179 | 100.335821 | 35.142537 | |

In [10]:
```
# separating the data and labels
features = diabetes_dataset.drop(columns = 'Outcome', axis=1)
target = diabetes_dataset['Outcome']
```

In [11]:
```
print(features)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6   
1              1       85             66             29        0  26.6   
2              8      183             64              0        0  23.3   
3              1       89             66             23       94  28.1   
4              0      137             40             35      168  43.1   
..           ...      ...            ...            ...      ...   ...   
763           10      101             76             48      180  32.9   
764            2      122             70             27        0  36.8   
765            5      121             72             23      112  26.2   
766            1      126             60              0        0  30.1   
767            1       93             70             31        0  30.4   

     DiabetesPedigreeFunction  Age  
0                       0.627   50  
1                       0.351   31  
2                       0.672   32  
3                       0.167   21  
4                       2.288   33  
..                        ...  ...  
763                     0.171   63  
764                     0.340   27  
765                     0.245   30  
766                     0.349   47  
767                     0.315   23  

[768 rows x 8 columns]
```

In [12]:
```
print(target)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Data Standardization

In [13]:
```
scaler = StandardScaler()
```

In [14]:
```
scaler.fit(features)
```

Out[14]:
```
StandardScaler()
```

In [15]:
```
standardized_data = scaler.transform(features)
```

In [16]:
```
print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

In [17]:
```
features = standardized_data
target = diabetes_dataset['Outcome']
```

In [18]:
```
print(features)
print(target)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Train Test Split

In [19]:
```
X_train, X_test, Y_train, Y_test = train_test_split(features,target, test_size =
```

In [20]:
```
print(features.shape, X_train.shape, X_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

Training the Model

In [21]:
```
classifier = Logistic_Regression(learning_rate=0.01, no_of_iterations=1000)
```

In [22]:
```
#training the support vector Machine Classifier
classifier.fit(X_train, Y_train)
```

Model Evaluation

Accuracy Score

In [23]:
```
# accuracy score on the training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score( Y_train, X_train_prediction)
```

In [24]:
```
print('Accuracy score of the training data : ', training_data_accuracy)
```

```
Accuracy score of the training data :  0.7768729641693811
```

In [25]:
```
# accuracy score on the test data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score( Y_test, X_test_prediction)
```

In [26]:
```
print('Accuracy score of the test data : ', test_data_accuracy)
```

```
Accuracy score of the test data :  0.7662337662337663
```

Making a Predictive System

In [27]:
```
input_data = (5,166,72,19,175,25.8,0.587,51)

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the input data
std_data = scaler.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[[ 0.3429808   1.41167241  0.14964075 -0.09637905  0.82661621 -0.78595734
   0.34768723  1.51108316]]
[1]
The person is diabetic

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\base.py:445: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature n
ames
  warnings.warn(
```

In [ ]: