Importing Liberaries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from warnings import filterwarnings
filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Load Data

In [2]:

```
diabetes = pd.read_csv("diabetes.csv")
```

In [3]:

```
diabetes.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [4]:

```
diabetes.shape
```

Out[4]:

```
(768, 9)
```

In [5]:

```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [6]:

```
diabetes.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
count	768.000000	768.000000	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	0.471876	33.240885
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000

In [7]:

```
diabetes.isnull().sum()
```

```
Out[7]: Pregnancies                 0
        Glucose                     0
        BloodPressure               0
        SkinThickness               0
        Insulin                     0
        BMI                         0
        DiabetesPedigreeFunction    0
        Age                         0
        Outcome                     0
        dtype: int64
```

```
In [8]: diabetes.Outcome.value_counts(normalize=True)*100
```

```
Out[8]: Outcome
        0    65.104167
        1    34.895833
        Name: proportion, dtype: float64
```

0 --> Non-Diabetic 1 --> Diabetic

Segregate The X and Y

```
In [9]: X = diabetes.drop('Outcome',axis =1)
        y = diabetes.Outcome
```

```
In [10]: print(X)
```

```
     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 [11]: print(y)
```

```
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 [12]: scaler = StandardScaler()
```

```
In [13]: X_scaled = scaler.fit_transform(X)
```

```
In [14]: print(X_scaled)
```

```
[[ 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]]
```

Train and Split

In [15]:

```
X_train,X_test,y_train,y_test = train_test_split(X_scaled,y,test_size=0.2,random_state=42)
```

In [16]:

```
print("X_train shape:",X_train.shape)
print("y_train shape:",y_train.shape)

print('-'*30)

print("X_test shape:",X_test.shape)
print("y_test shape:",y_test.shape)
```

```
X_train shape: (614, 8)
y_train shape: (614,)
------------------------------
X_test shape: (154, 8)
y_test shape: (154,)
```

In [17]:

```
lr = LogisticRegression()
lr
```

Out[17]:

```
▾ LogisticRegression
LogisticRegression()
```

In [18]:

```
lr.fit(X_train,y_train)
```

Out[18]:

```
▾ LogisticRegression
LogisticRegression()
```

In [45]:

```
X_train_pred = lr.predict(X_train)
```

In [46]:

```
pd.DataFrame(y_pred,columns=["Predicted"])
```

Out[46]:

	Predicted
0	0
1	0
2	0
3	1
4	1
...	...
609	0
610	0
611	1
612	1
613	0

614 rows × 1 columns

In [47]:

```
pd.DataFrame(y_train)
```

Out[47]:

	Outcome
60	0
618	1
346	0
294	0
231	1
...	...
71	0
106	0
270	1
435	1
102	0

614 rows × 1 columns

In [48]:

```
print("Accuracy Score :",accuracy_score(y_train,X_train_pred))
```

```
Accuracy Score : 0.7703583061889251
```

In [50]:

```
#Accuracy Score test data
y_test_pred = lr.predict(X_test)
print("Accuracy Score : ",accuracy_score(y_test,y_test_pred))
```

```
Accuracy Score :  0.7532467532467533
```

SVM

In [23]:

```
classifier = svm.SVC(kernel='linear')
```

In [24]:

```
#training support vector Machine classifier
classifier.fit(X_train,y_train)
```

Out[24]:

```
▾ SVC
SVC(kernel='linear')
```

In [36]:

```
y_pred_svm = classifier.predict(X_train)
```

In [37]:

```
y_pred_svm
```

Out[37]:

```
array([0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0])
```

In [40]:

```
print("Accuracy Score :",accuracy_score(y_pred_svm,y_train))
```

```
Accuracy Score : 0.7719869706840391
```

In [52]:

```
# test data Accuracy Score
y_test_pred = classifier.predict(X_test)
print("Accuracy Score :",accuracy_score(y_test_pred,y_test))
```

```
Accuracy Score : 0.7597402597402597
```

In [ ]:

Making Predicting System

In [59]:

```
input_data = (4,110,92,0,0,37.6,0.191,30)
#changing the input data to numpy array
input_data_numpy_array = np.asarray(input_data)

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

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

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

```
[[ 0.04601433 -0.34096773  1.18359575 -1.28821221 -0.69289057  0.71168975
  -0.84827977 -0.27575966]]
[0]
```

In [ ]: