Dibetes Prediction

In [1]:
```python
from warnings import filterwarnings
filterwarnings('ignore')
```

Importing the Library

In [2]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import pickle


%matplotlib inline
```

Importing the models Library

In [34]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

In [4]:
```python
df = pd.read_csv("D:\My End to End Projects\Logistic_Regression_Project\Dataset\diabetes.csv")
```

In [5]:
```python
df.head()
```

Out[5]:

	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 [6]:
```python
df.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 [7]:
```
df.shape
```

Out[7]:
```
(768, 9)
```

In [8]:
```
df.columns
```

Out[8]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [10]:
```
df['Outcome'].value_counts()
```

Out[10]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

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

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood Pressure which have value as 0. That's not possible,right? you can do a quick search to see that one cannot have 0 values for these. Let's deal with that. we can either remove such data or simply replace it with their respective mean values. Let's do the latter.

In [12]:

```
#here few misconception is there lke BMI can not be zero, BP can't be zero, glucose, insuline ca

# Replacing the BMI column 0 values with mean of the Column
df['BMI'] = df['BMI'].replace(0,df['BMI'].mean())

# Replacing the Gulcose column 0 values with mean of the Column
df['Glucose'] = df['Glucose'].replace(0,df['Glucose'].mean())

# Replacing the Insulin column 0 values with mean of the Column
df['Insulin'] = df['Insulin'].replace(0,df['Insulin'].mean())

# Replacing the BloodPressure column 0 values with mean of the Column
df['BloodPressure'] = df['BloodPressure'].replace(0,df['BloodPressure'].mean())

# Replacing the SkinThickness column 0 values with mean of the Column
df['SkinThickness'] = df['SkinThickness'].replace(0,df['SkinThickness'].mean())
```

In [13]:

```
df.head()
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

Seperate the Independent and Dependent Variable

In [14]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [15]:
```
X.head()
```

Out[15]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.000000	79.799479	33.6	0.627	50
1	1	85.0	66.0	29.000000	79.799479	26.6	0.351	31
2	8	183.0	64.0	20.536458	79.799479	23.3	0.672	32
3	1	89.0	66.0	23.000000	94.000000	28.1	0.167	21
4	0	137.0	40.0	35.000000	168.000000	43.1	2.288	33

In [16]:
```
y.head()
```

Out[16]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Perfoming the Train Test Split On X and Y data

In [17]:
```
X_train, X_test, ytrain, ytest  = train_test_split(X,y, test_size=0.25, random_state=0)
```

In [18]:
```
X_train.shape, X_test.shape
```

Out[18]:
```
((576, 8), (192, 8))
```

In [19]:
```
ytrain.shape, ytest.shape
```

Out[19]:
```
((576,), (192,))
```

Applying the Standardscaler into X dataset

In [20]:
```
def Standard_scaler(X_train, X_test):
    #Scaler the Data
    scaler = StandardScaler()
    xtrain_scaled = scaler.fit_transform(X_train)
    xtest_scaled = scaler.transform(X_test)

    # Saving the Model
    file = open('D:\My End to End Projects\Logistic_Regression_Project\Model\standardScaler.pkl'
```

```
    pickle.dump(scaler,file)
    file.close()

    return xtrain_scaled, xtest_scaled
```

In [21]:
```
scaled_xtrain, scaled_xtest = Standard_scaler(X_train,X_test)
```

In [22]:
```
scaled_xtrain
```

Out[22]:
```
array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,
        -0.98325882, -0.04863985],
       [-0.82986389, -0.1331471 , -1.23618124, ...,  0.09272955,
        -0.62493647, -0.88246592],
       [-1.12204091, -1.03283573,  0.61597784, ..., -0.03629955,
         0.39884168, -0.5489355 ],
       ...,
       [ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,
        -0.96519215, -1.04923114],
       [ 2.09190629, -1.23276654,  0.11084355, ..., -0.36604058,
        -0.5075031 ,  0.11812536],
       [ 0.33884418,  0.46664532,  0.78435594, ..., -0.09470985,
         0.51627505,  2.953134  ]])
```

Training the Logistic Regression Model

In [24]:
```
log_reg = LogisticRegression()

log_reg.fit(scaled_xtrain,ytrain)
```

Out[24]:
```
▾ LogisticRegression
LogisticRegression()
```

In [26]:
```
# Hyperparameter Tuning with GridSearch CV
## Parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

In [27]:
```
logreg = LogisticRegression()
clf = GridSearchCV(logreg,                    # model
                   param_grid = parameters,   # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=10)                     # number of folds

clf.fit(scaled_xtrain,ytrain)
```

Out[27]:

```
▸ GridSearchCV
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [28]:

```
clf.best_params_
```

Out[28]:

```
{'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

In [29]:

```
clf.best_score_
```

Out[29]:

```
0.763793103448276
```

-> let's see how well our model performs on the test data set.

In [31]:

```
y_pred = clf.predict(scaled_xtest)
```

In [32]:

```
y_pred
```

Out[32]:

```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0], dtype=int64)
```

In [33]:

```
ytest
```

Out[33]:

```
661    1
122    0
113    0
14     1
529    0
      ..
366    1
301    1
382    0
140    0
463    0
Name: Outcome, Length: 192, dtype: int64
```

In [46]:

```
print(f"Confusion Matrix :\n{confusion_matrix(y_pred,ytest)}")
print("\n=======================================\n")
print(f"Accuracy Score : {accuracy_score(y_pred,ytest)}")
print("\n=======================================\n")
print(f"Classification Report :\n \n{classification_report(y_pred,ytest)}")
```

```
Confusion Matrix :
[[117  26]
 [ 13  36]]

====================================

Accuracy Score : 0.796875

====================================

Classification Report :

              precision    recall  f1-score   support

           0       0.90      0.82      0.86       143
           1       0.58      0.73      0.65        49

    accuracy                           0.80       192
   macro avg       0.74      0.78      0.75       192
weighted avg       0.82      0.80      0.80       192
```

Saving the trained Model into Pickle File

In [47]:

```
loc_file = open('D:\My End to End Projects\Logistic_Regression_Project\Model\modelforprediction.
pickle.dump(log_reg,loc_file)
loc_file.close()
```

In [ ]: