Diabetes Prediction using Logistic Regression

Import libraries

In [1]:

```
import pandas as pd
import numpy as np
```

Import data frame

In [2]:

```
df=pd.read_csv(r'https://github.com/YBI-Foundation/Dataset/raw/main/Diabetes.csv'
```

In [3]:

```
df.head()
```

Out[3]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
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

metadata:

In [4]:

```
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   diastolic    768 non-null    int64  
 3   triceps      768 non-null    int64  
 4   insulin      768 non-null    int64  
 5   bmi          768 non-null    float64
 6   dpf          768 non-null    float64
 7   age          768 non-null    int64  
 8   diabetes     768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:
```
df.describe()
```

Out[5]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.2
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.7
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.0
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.0
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.0
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.0
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.0

Getting columns:

In [7]:
```
df.columns
```

Out[7]:
```
Index(['pregnancies', 'glucose', 'diastolic', 'triceps', 'insulin', 'bmi',
       'dpf', 'age', 'diabetes'],
      dtype='object')
```

In [10]:
```
df.shape
```

Out[10]:
```
(768, 9)
```

Get unique values(class/label) in y variables

In [8]:
```
df['diabetes'].value_counts()
```

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

In [9]:
```
df.groupby('diabetes').mean()
```

Out[9]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	ag
diabetes								
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.19000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.06716

Define y(dependent/label/target variable) and X (independent/feature/attribute variable)

```
In [50]: y=df['diabetes']
```

```
In [51]: y.shape
```

```
Out[51]: (768,)
```

```
In [52]: y
```

```
Out[52]: 0      1
         1      0
         2      1
         3      0
         4      1
               ..
         763    0
         764    0
         765    0
         766    1
         767    0
         Name: diabetes, Length: 768, dtype: int64
```

```
In [53]: X=df.drop(['diabetes'],axis=1)
```

```
In [54]: X.shape
```

```
Out[54]: (768, 8)
```

In [55]:
```
X
```

Out[55]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	0	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	0	0	30.1	0.349	47
767	1	93	70	31	0	30.4	0.315	23

768 rows × 8 columns

Getting X variable standardized using MinMaxScaler

In [24]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [25]:
```
mm=MinMaxScaler()
```

In [26]:
```
X=mm.fit_transform(X)
```

In [27]:
```
X
```

Out[27]:
```
array([[0.35294118, 0.74371859, 0.59016393, ..., 0.50074516, 0.23441503,
        0.48333333],
       [0.05882353, 0.42713568, 0.54098361, ..., 0.39642325, 0.11656704,
        0.16666667],
       [0.47058824, 0.91959799, 0.52459016, ..., 0.34724292, 0.25362938,
        0.18333333],
       ...,
       [0.29411765, 0.6080402 , 0.59016393, ..., 0.390462  , 0.07130658,
        0.15      ],
       [0.05882353, 0.63316583, 0.49180328, ..., 0.4485842 , 0.11571307,
        0.43333333],
       [0.05882353, 0.46733668, 0.57377049, ..., 0.45305514, 0.10119556,
        0.03333333]])
```

Get model trained

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.3,random_state=25
```

In [30]:
```
from sklearn.linear_model import LogisticRegression
```

In [31]:
```
lr=LogisticRegression()
```

In [32]:
```
lr.fit(X_train,y_train)
```

Out[32]:
```
LogisticRegression()
```

Get model predictions:

In [56]:
```
y_pred=lr.predict(X_test)
```

In [35]:
```
y_pred.shape
```

Out[35]:
```
(538,)
```

In [36]:
```
y_pred
```

Out[36]:
```
array([0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 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, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1], dtype=int64)
```

Get probability of each predicted class

In [37]:
```
lr.predict_proba(X_test)
```

Out[37]:
```
array([[0.84199514, 0.15800486],
       [0.61583618, 0.38416382],
       [0.45354833, 0.54645167],
       ...,
       [0.74520246, 0.25479754],
       [0.85004994, 0.14995006],
       [0.24202189, 0.75797811]])
```

Get model evaluation

In [38]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [39]:
```
print(confusion_matrix(y_test,y_pred))
```

```
[[333  13]
 [135  57]]
```

In [40]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.96      0.82       346
           1       0.81      0.30      0.44       192

    accuracy                           0.72       538
   macro avg       0.76      0.63      0.63       538
weighted avg       0.75      0.72      0.68       538
```

Get future predictions
steps:

1. **extract a random row using sample function**
2. **separate X and y**
3. **standardize X**
4. **predict**

In [41]:
```
X_new=df.sample(1)
```

In [42]:
```
X_new
```

Out[42]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
348	3	99	62	19	74	21.8	0.279	26	0

In [43]:
```
X_new.shape
```

Out[43]:
```
(1, 9)
```

In [44]:
```
X_new=X_new.drop('diabetes',axis=1)
```

In [45]:
```
X_new
```

Out[45]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
348	3	99	62	19	74	21.8	0.279	26

In [46]:
```
X_new=mm.fit_transform(X_new)
```

In [47]:
```
y_pred_new=lr.predict(X_new)
```

In [48]:
```
y_pred_new
```

Out[48]:
```
array([0], dtype=int64)
```

In [49]:
```
lr.predict_proba(X_new)
```

Out[49]:
```
array([[0.97981187, 0.02018813]])
```

predicted and actual class is zero(0) i.e. Non-Diabetic

In [ ]:
```

```