Logistic Regression

In [1]:
```
import pandas as pd
Data=pd.read_csv("C:/Users/SAGNIK SAMANTA/OneDrive/Desktop/Datasets/diabete
Data.columns
```

Out[1]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insul
in',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [2]:
```
Data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

Selecting Feature

Here, you need to divide the given columns into two types of variables dependent(or target variable) and independent variable(or feature variables).

In [3]:
```
features_cols=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
X=Data[features_cols]
y=Data.Outcome
```

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Let's split the dataset by using the function train_test_split(). You need to pass 3 parameters: features, target, and test_set size. Additionally, you can use random_state to select records randomly.

In [4]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_st
```

Here, the Dataset is broken into two parts in a ratio of 75:25. It means 75% data will be used for model training and 25% for model testing.

Model Development and Prediction

First, import the Logistic Regression module and create a Logistic Regression classifier object using the LogisticRegression() function with random_state for reproducibility.

Then, fit your model on the train set using fit() and perform prediction on the test set using predict().

In [7]:
```
from sklearn.linear_model import LogisticRegression
```

In [9]:
```
logr=LogisticRegression(random_state=16)
logr.fit(X_train,y_train)
```

```
C:\Users\SAGNIK SAMANTA\anaconda3\Lib\site-packages\sklearn\linear_model
\_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  n_iter_i = _check_optimize_result(
```

Out[9]:
```
LogisticRegression(random_state=16)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Now we have a logistic regression object that is ready to decide whether a parient is diabetic or not :

In [11]:
```
y_predict=logr.predict(X_test)
print(y_predict)
```

```
[1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0
 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1
0
 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0
0
 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
1
 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1
0
 1 1 0 0 0 1 1]
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. You can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions summed up class-wise.

In [12]:
```
from sklearn import metrics
confusion_matrix=metrics.confusion_matrix(y_test,y_predict)
confusion_matrix
```

Out[12]:
```
array([[116,   9],
       [ 26,  41]], dtype=int64)
```

Here, you can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification. You have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions. In the output, 116 and 41 are actual predictions, and 9 and 26 are incorrect predictions.

Visualizing Confusion Matrix using Heatmap

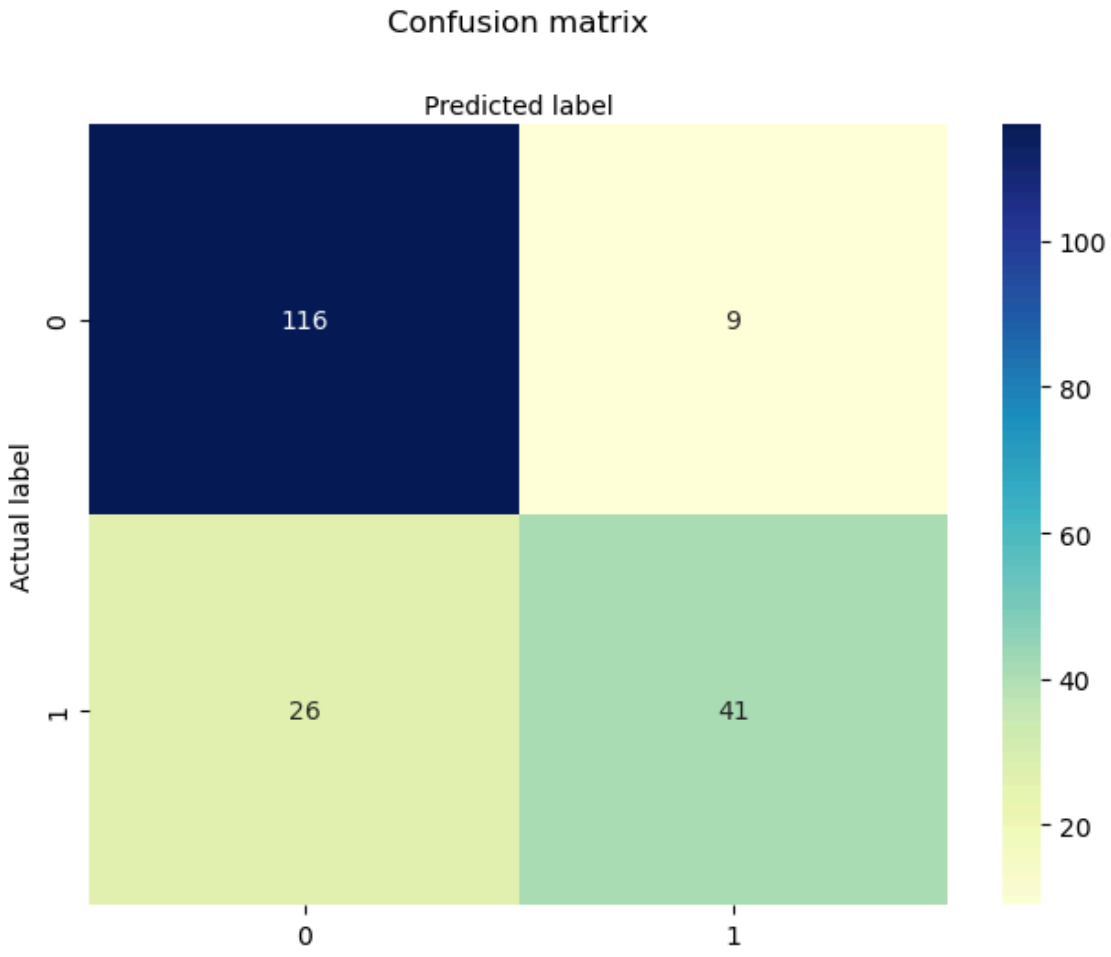
Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

Here, you will visualize the confusion matrix using Heatmap.

In [17]:
```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(confusion_matrix), annot=True, cmap="YlGnBu" ,fmt=
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[17]:
```
Text(0.5, 427.9555555555555, 'Predicted label')
```



Confusion Matrix Evaluation Metrics

Let's evaluate the model using classification_report for accuracy, precision, and recall.

In [18]:
```
from sklearn.metrics import classification_report
target_names=["Without Diabetes","With Diabetes"]
print(classification_report(y_test,y_predict,target_names=target_names))
```

```
                  precision    recall  f1-score   support

Without Diabetes       0.82      0.93      0.87       125
   With Diabetes       0.82      0.61      0.70        67

        accuracy                           0.82       192
       macro avg       0.82      0.77      0.78       192
    weighted avg       0.82      0.82      0.81       192
```

Precision: Precision is about being precise, i.e., how accurate your model is. In other words, you can say, when a model makes a prediction, how often it is correct. In your prediction case, when your Logistic Regression model predicted patients are going to suffer from diabetes, that patients have 82% of the time.

Recall: If there are patients who have diabetes in the test set and your Logistic Regression model can identify it 61% of the time. Well, you got a classification rate of 82%, considered as good accuracy.

In [ ]:
```
```