TUMUHAISE DAVID BCS 00846

In [117]:
```
# importing the necessary libraies
import pandas as pd
import numpy as np
```

In [118]:
```
data = pd.read_csv("C:\\Users\\TUMUHAISE DAVID BCS\\Desktop\\davidologistic\\test.csv")
```

In [119]:
```
data.head()
```

Out[119]:

	id	age	education	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucos
0	3390	43	2.0	35.0	0.0	0	0	0	207.0	117.0	65.0	24.42	60	100
1	3391	56	3.0	0.0	0.0	0	0	0	192.0	122.0	82.5	28.61	68	58
2	3392	58	1.0	20.0	0.0	0	1	0	260.0	180.0	100.0	25.56	100	Na
3	3393	47	3.0	0.0	0.0	0	0	0	231.0	102.5	66.0	23.40	70	78
4	3394	44	1.0	0.0	0.0	0	0	0	160.0	118.5	87.0	25.81	54	Na

In [138]:
```
X=np.array(data[["sysBP","diaBP","heartRate","prevalentStroke","prevalentHyp","age"]])
y = np.array(data["diabetes"])
```

In [139]:
```
X
```

Out[139]:
```
array([[117. ,  65. ,  60. ,   0. ,   0. ,  43. ],
       [122. ,  82.5,  68. ,   0. ,   0. ,  56. ],
       [180. , 100. , 100. ,   0. ,   1. ,  58. ],
       ...,
       [134. ,  80. , 120. ,   0. ,   0. ,  55. ],
       [157.5, 104.5,  75. ,   0. ,   1. ,  45. ],
       [138. ,  80. ,  67. ,   0. ,   0. ,  56. ]])
```

In [140]:
```
# checking for the missing valves
data.isnull().sum()
```

Out[140]:
```
id                0
age               0
education        18
cigsPerDay        7
BPMeds            9
prevalentStroke   0
prevalentHyp      0
diabetes          0
totChol          12
sysBP             0
diaBP             0
BMI               5
heartRate         0
glucose          84
dtype: int64
```

In [141]:
```
# handling the missing values
data.fillna(data.mean(), inplace = True)
data.isnull().sum()
```

Out[141]:
```
id                0
age               0
education         0
cigsPerDay        0
BPMeds            0
prevalentStroke   0
prevalentHyp      0
diabetes          0
totChol           0
sysBP             0
diaBP             0
BMI               0
heartRate         0
glucose           0
dtype: int64
```

In [142]:
```
# spliting the dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

In [143]:
```
from sklearn.linear_model import LogisticRegression
```

In [144]:
```
model =LogisticRegression()
```

In [145]:
```
model.fit(X_train,y_train)
```

Out[145]:
```
▾ LogisticRegression

LogisticRegression()
```

In [146]:
```
y_pred = model.predict(X_test)
```

In [147]:
```
from sklearn.metrics import accuracy_score
```

In [148]:
```
accuracy = accuracy_score(y_test,y_pred)
```

In [149]:
```
accuracy
```

Out[149]:
```
0.9705882352941176
```

model optimization

In [150]:
```
# importing the necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,f1_score, confusion_matrix
```

In [151]:
```
data = pd.read_csv("C:\\Users\\TUMUHAISE DAVID BCS \\Desktop\\davidologistic\\test.csv")
```

In [152]:
```
X=np.array(data[["sysBP","diaBP","heartRate","prevalentStroke","prevalentHyp","age","education","cigsPerDay"]]
y = np.array(data["diabetes"])
```

In [153]:
```
# building the logistic regression model
model = LogisticRegression()
```

In [154]:
```
#Implementing the hyperparamter tuning using gridsearchCV


param_grid = {

             "C":[0.01,0.1,1,10,100],
               "penalty":["l1","l2"],
             "fit_intercept":[True,False],
              "solver":[ 'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga']

             }
```

In [155]:
```python
# perform a gridsearch (cross validation)

grid_search= GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train )
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:976: UserWarning: One or more o
f the test scores are non-finite: [        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464
        nan 0.97493464        nan        nan        nan 0.97493464
 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464 0.97493464]
  warnings.warn(
```

Out[155]:

▸ **GridSearchCV**

▸ **estimator: LogisticRegression**

▸ LogisticRegression

In [156]:
```
#Geting the best paramters found from Grid Search

best_params = grid_search.best_params_
print("Best parameters: ", best_params)

# use the best parameters to train the model

best_model = LogisticRegression(**best_params)
best_model.fit(X_train,y_train)
y_pred = best_model.predict(X_test)

#Evaluating the performance of the model
accuracy = accuracy_score(y_test,y_pred)
print("accuracy: ", accuracy)
```

```
Best parameters:  {'C': 0.01, 'fit_intercept': True, 'penalty': 'l1', 'solver': 'liblinear'}
accuracy:  0.9705882352941176
```

In [ ]: