Practical 4

Aim : To implement a Machine Learning Classification model using a Logistic regression algorithm

Data Preprocessing

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.metrics import precision_recall_fscore_support
from sklearn import metrics
```

In [24]:
```
data = pd.read_csv(r"practical4.csv")
```

In [25]:
```
data.head()
```

Out[25]:

	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.

In [26]:
```
data.isna().sum()
```

Out[26]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [27]:
```
data.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 [28]:
```
data.describe()
```

Out[28]:

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

Parameter setup

In [29]:
```
x= data.iloc[:, [0,7]].values
y=data.iloc[:, -1].values
```

In [30]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_
```

Model Training

In [31]:
```
classifier= LogisticRegression()
classifier.fit(x_train, y_train)
```

Out[31]:
```
LogisticRegression()
```

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.

Making Predication

In [32]:
```
y_pred= classifier.predict(x_test)
```

In [33]:
```
print(y_test)
```

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

In [34]:
```
print(y_pred)
```

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

Confusion Matrix

In [35]:
```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[117  13]
 [ 49  13]]
```

In [36]:
```
precision_recall_fscore_support(y_test, y_pred, average='macro')
```

Out[36]:
```
(0.6024096385542168, 0.5548387096774193, 0.542997542997543, None)
```

In [37]:
```
precision_recall_fscore_support(y_test, y_pred, average='micro')
```

Out[37]:
```
(0.6770833333333334, 0.6770833333333334, 0.6770833333333334, None)
```

In [38]:
```
precision_recall_fscore_support(y_test, y_pred, average='weighted')
```

Out[38]: (0.638679718875502, 0.6770833333333334, 0.6306690212940212, None)

In [39]:
```
accuracy_score(y_test,y_pred)
```

Out[39]: 0.6770833333333334

In [40]:
```
Accuracy = metrics.accuracy_score(y_test,y_pred)
Accuracy
```

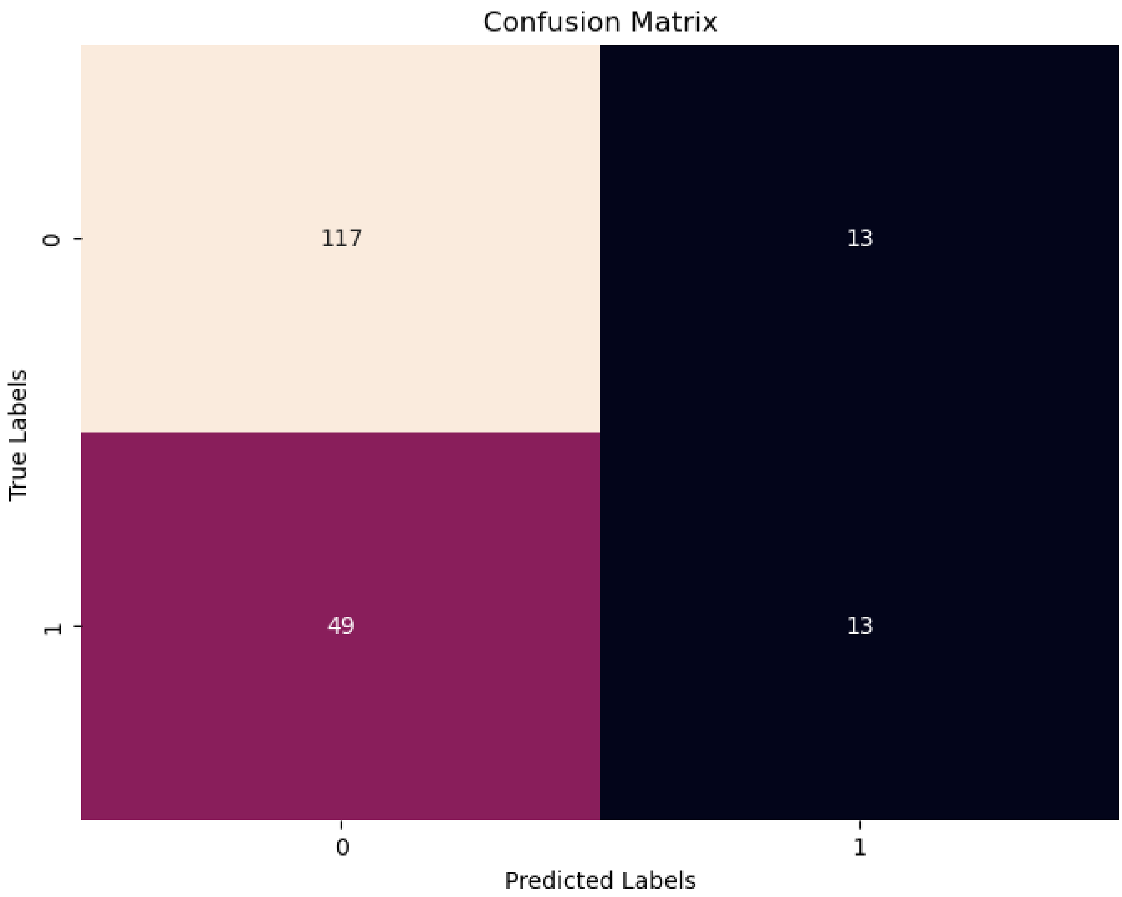
Out[40]: 0.6770833333333334

In [41]:
```
classifier.intercept_
classifier.coef_
```

Out[41]: array([[0.05180623, 0.03442028]])

Ploting of Confusion Matrix

In [42]:
```
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [43]:

```
from sklearn.metrics import precision_recall_fscore_support
precision, recall, f1_score,_ = precision_recall_fscore_support(y_test, y_p
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1_score)
```

```
Precision: [0.70481928 0.5       ]
Recall: [0.9        0.20967742]
F1 Score: [0.79054054 0.29545455]
```