Practical 5

Aim: To implement a Machine Learning Classification model using a K Nearest Neighbors Classifier algorithm and enhance the model by K Fold and GridSearchCV cross-validation.

In [29]:
```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
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
import matplotlib.pyplot as plt
```

Loading dataset

specifying the x and y values

In [30]:
```
data = pd.read_csv(r"Practical5.csv")
X = data.iloc[:, [1, 2, 3, 4, 5, 6, 7]].values
y = data.iloc[:, -1].values
```

In [31]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
```

splitting up the dataset

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=42)
```

In [33]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

Training the model

In [34]:
```
knn.fit(X_train, y_train)
```

Out[34]:
```
KNeighborsClassifier(n_neighbors=7)
```

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 the predictions

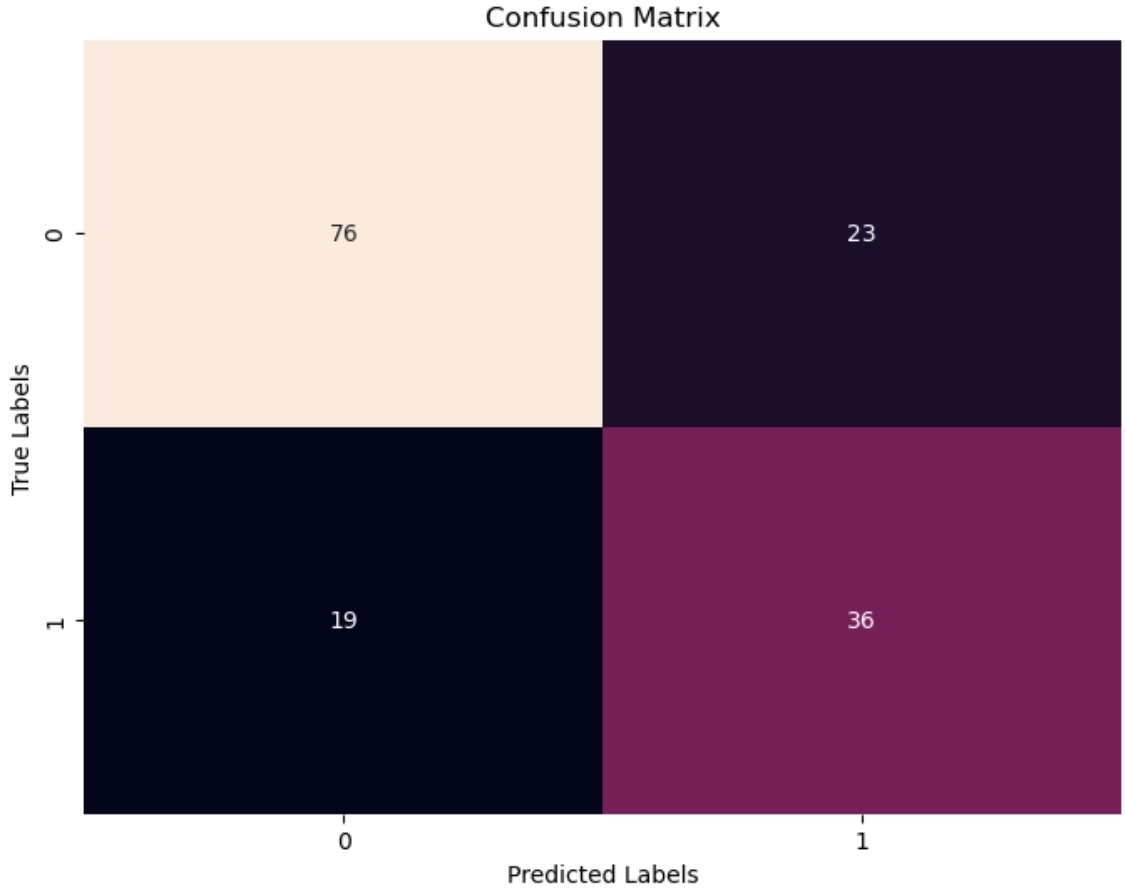
In [35]:
```
y_pred=knn.predict(X_test)
print(knn.predict(X_test))
```

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

In [36]:
```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[76 23]
 [19 36]]
```

In [37]:
```
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()
```



Use of KFold cross validation to find best input configuration

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.2, random_state=1)
```

In [39]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
#Data is splited into 10 same parts
cv = KFold(n_splits=10)
# perform cross-validation procedure
for train_ix, test_ix in cv.split(X):
    # split data
    X_train, X_test = X[train_ix, :], X[test_ix, :]
    y_train, y_test = y[train_ix], y[test_ix]
    # fit and evaluate a model
    knn = KNeighborsClassifier(n_neighbors=7)
    knn.fit(X_train, y_train)
    y_pred=knn.predict(X_test)
    print(knn.predict(X_test))
    #draw confusion matrix
    cm= confusion_matrix(y_test,y_pred)
    print(cm)
    #find metrices of evalution
    precision, recall, f1_score,_ = precision_recall_fscore_support(y_test,
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1_score)
```

```
[0 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0
 0 0 0]
[[35 10]
 [16 16]]
Precision: [0.68627451 0.61538462]
Recall: [0.77777778 0.5       ]
F1 Score: [0.72916667 0.55172414]
[0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0
 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0
 0 1 1]
[[50  5]
 [ 9 13]]
Precision: [0.84745763 0.72222222]
Recall: [0.90909091 0.59090909]
F1 Score: [0.87719298 0.65      ]
[1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0
 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1
 1 0 0]
[[37  6]
 [17 17]]
Precision: [0.68518519 0.73913043]
Recall: [0.86046512 0.5       ]
F1 Score: [0.7628866  0.59649123]
[0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 0
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0
 0 1 0]
[[36 11]
 [17 13]]
Precision: [0.67924528 0.54166667]
Recall: [0.76595745 0.43333333]
F1 Score: [0.72       0.48148148]
[0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0
 0 0 0]
[[42  8]
 [12 15]]
Precision: [0.77777778 0.65217391]
Recall: [0.84       0.55555556]
F1 Score: [0.80769231 0.6       ]
[0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0
 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1
 0 0 0]
[[43  4]
 [12 18]]
Precision: [0.78181818 0.81818182]
Recall: [0.91489362 0.6       ]
F1 Score: [0.84313725 0.69230769]
[0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 1
 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0]
[[53 10]
 [ 8  6]]
Precision: [0.86885246 0.375     ]
Recall: [0.84126984 0.42857143]
F1 Score: [0.85483871 0.4       ]
[1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1
 0 1 0]
[[45  7]
 [ 6 19]]
```

```
Precision: [0.88235294 0.73076923]
Recall: [0.86538462 0.76      ]
F1 Score: [0.87378641 0.74509804]
[0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 1
 0 1]
[[43  9]
 [10 14]]
Precision: [0.81132075 0.60869565]
Recall: [0.82692308 0.58333333]
F1 Score: [0.81904762 0.59574468]
[0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 1
 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0
 1 0]
[[38  8]
 [12 18]]
Precision: [0.76       0.69230769]
Recall: [0.82608696 0.6       ]
F1 Score: [0.79166667 0.64285714]
```

Applying The GridSearchCV for getting best classifier no. of neighbours

In [40]:
```
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, precision_recall_fscore_suppo

# Define the KFold cross-validation
cv = KFold(n_splits=12)

param_grid = {'n_neighbors': list(range(1, 51, 2))}



# Initialize the KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=cv, sco

# Perform cross-validation procedure with GridSearchCV
for train_ix, test_ix in cv.split(X):
    # Split data
    X_train, X_test = X[train_ix, :], X[test_ix, :]
    y_train, y_test = y[train_ix], y[test_ix]

    # Fit model using GridSearchCV
    grid_search.fit(X_train, y_train)

    # Get the best KNN model found by GridSearchCV
    best_knn = grid_search.best_estimator_

    # Predict
    y_pred = best_knn.predict(X_test)

    # Evaluate
    cm = confusion_matrix(y_test, y_pred)
    precision, recall, f1_score, _ = precision_recall_fscore_support(y_test

    # Print results
    print("Confusion Matrix:")
    print(cm)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1_score)

    # You can also access the best hyperparameters found
    print("Best parameters found by GridSearchCV:", grid_search.best_params
```

```
Confusion Matrix:
[[29  7]
 [13 15]]
Precision: [0.69047619 0.68181818]
Recall: [0.80555556 0.53571429]
F1 Score: [0.74358974 0.6       ]
Best parameters found by GridSearchCV: {'n_neighbors': 15}
Confusion Matrix:
[[43  2]
 [12  7]]
Precision: [0.78181818 0.77777778]
Recall: [0.95555556 0.36842105]
F1 Score: [0.86 0.5 ]
Best parameters found by GridSearchCV: {'n_neighbors': 23}
Confusion Matrix:
[[35  7]
 [13  9]]
Precision: [0.72916667 0.5625    ]
Recall: [0.83333333 0.40909091]
F1 Score: [0.77777778 0.47368421]
Best parameters found by GridSearchCV: {'n_neighbors': 23}
Confusion Matrix:
[[27  8]
 [14 15]]
Precision: [0.65853659 0.65217391]
Recall: [0.77142857 0.51724138]
F1 Score: [0.71052632 0.57692308]
Best parameters found by GridSearchCV: {'n_neighbors': 9}
Confusion Matrix:
[[30  8]
 [15 11]]
Precision: [0.66666667 0.57894737]
Recall: [0.78947368 0.42307692]
F1 Score: [0.72289157 0.48888889]
Best parameters found by GridSearchCV: {'n_neighbors': 17}
Confusion Matrix:
[[35  8]
 [ 9 12]]
Precision: [0.79545455 0.6       ]
Recall: [0.81395349 0.57142857]
F1 Score: [0.8045977  0.58536585]
Best parameters found by GridSearchCV: {'n_neighbors': 13}
Confusion Matrix:
[[35  3]
 [11 15]]
Precision: [0.76086957 0.83333333]
Recall: [0.92105263 0.57692308]
F1 Score: [0.83333333 0.68181818]
Best parameters found by GridSearchCV: {'n_neighbors': 17}
Confusion Matrix:
[[42  8]
 [ 7  7]]
Precision: [0.85714286 0.46666667]
Recall: [0.84 0.5 ]
F1 Score: [0.84848485 0.48275862]
Best parameters found by GridSearchCV: {'n_neighbors': 15}
Confusion Matrix:
[[46  5]
 [ 4  9]]
Precision: [0.92       0.64285714]
Recall: [0.90196078 0.69230769]
```

```
F1 Score: [0.91089109 0.66666667]
Best parameters found by GridSearchCV: {'n_neighbors': 15}
Confusion Matrix:
[[41  2]
 [ 6 15]]
Precision: [0.87234043 0.88235294]
Recall: [0.95348837 0.71428571]
F1 Score: [0.91111111 0.78947368]
Best parameters found by GridSearchCV: {'n_neighbors': 15}
Confusion Matrix:
[[32  8]
 [ 9 15]]
Precision: [0.7804878  0.65217391]
Recall: [0.8   0.625]
F1 Score: [0.79012346 0.63829787]
Best parameters found by GridSearchCV: {'n_neighbors': 17}
Confusion Matrix:
[[34  5]
 [ 9 16]]
Precision: [0.79069767 0.76190476]
Recall: [0.87179487 0.64      ]
F1 Score: [0.82926829 0.69565217]
Best parameters found by GridSearchCV: {'n_neighbors': 17}
```

In [44]:
```
print("Best parameters found by GridSearchCV:", grid_search.best_params_)
```

```
Best parameters found by GridSearchCV: {'n_neighbors': 17}
```

In [46]:
```
knn = KNeighborsClassifier(n_neighbors=17)
```

In [47]:
```
knn.fit(X_train, y_train)
```

Out[47]:
```
KNeighborsClassifier(n_neighbors=17)
```

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.

In [48]:
```
y_pred=knn.predict(X_test)
```

In [49]:
```
cm= confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[34  5]
 [ 9 16]]
```

In [50]:
```
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()
```