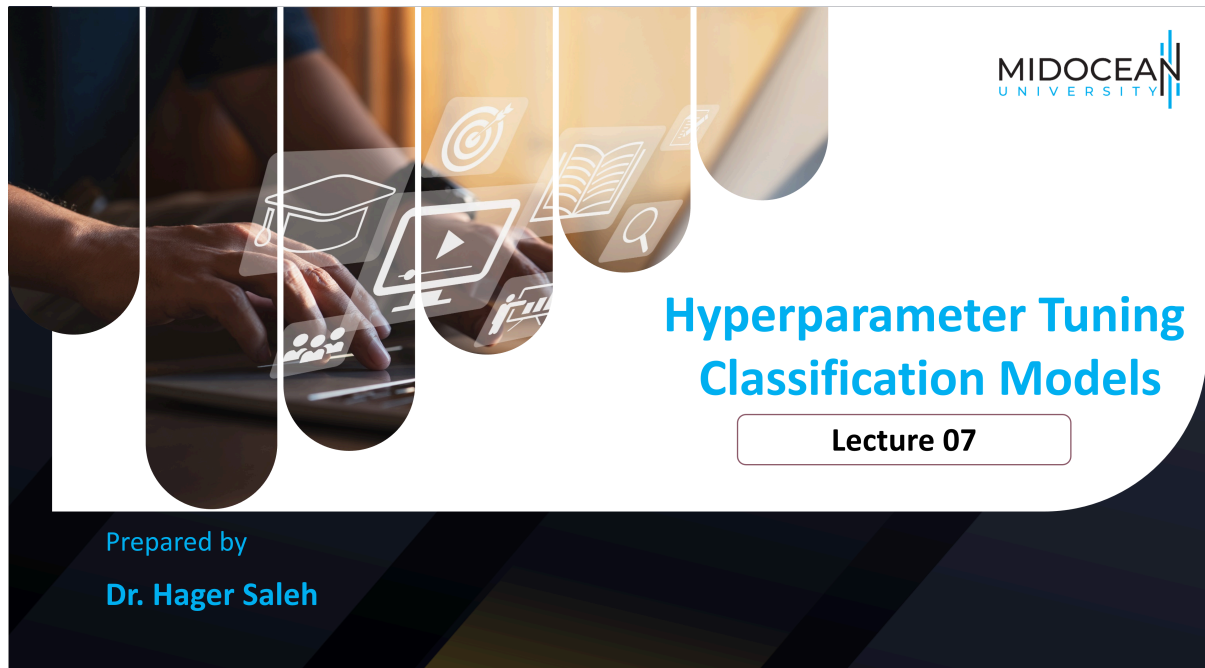


In [14]:



Import necessary libraries

```
In [15]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

Reading dataset

```
In [16]: data = pd.read_csv('kidney_disease_clean_final.csv')
```

```
In [17]: data.head()
```

```
Out[17]:
```

	age	bp	sg	al	su	pc	pcc	ba	bgr	bu	...	hemo	pcv	wc	htn	dm
0	48.0	80.0	1.020	1.0	0.0	1	0	0	121.000000	36.0	...	15.4	32	72	1	4
1	7.0	50.0	1.020	4.0	0.0	1	0	0	147.943503	18.0	...	11.3	26	56	0	3
2	62.0	80.0	1.010	2.0	3.0	1	0	0	423.000000	53.0	...	9.6	19	70	0	4
3	48.0	70.0	1.005	4.0	0.0	0	1	0	117.000000	56.0	...	11.2	20	62	1	3
4	51.0	80.0	1.010	2.0	0.0	1	0	0	106.000000	26.0	...	11.6	23	68	0	3

5 rows × 23 columns

```
In [18]: data['classification'].value_counts()
```

```
Out[18]: classification
0      248
1      150
Name: count, dtype: int64
```

Convert categorical features to numerical data.

```
In [19]: label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le
```

Define features and target variable

```
In [20]: X = data.drop('classification', axis=1).values # Replace 'classification' with you
y = data['classification'].values
```

Splitting data

```
In [21]: # Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_s
```

Evaluation models

```
In [22]: def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
```

```

report = classification_report(y_true, y_pred)

# Output results
metrics = {
    'Model Name': model_name,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1,

    'Classification Report': report
}

# Plot Confusion Matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(y_true), yticklabels=np.unique(y_true))
plt.title(f'Confusion Matrix for {model_name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

return metrics

```

Random Forest Classifier

```

In [23]: # Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, sc

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test, y_pred)

# Print the evaluation results

```

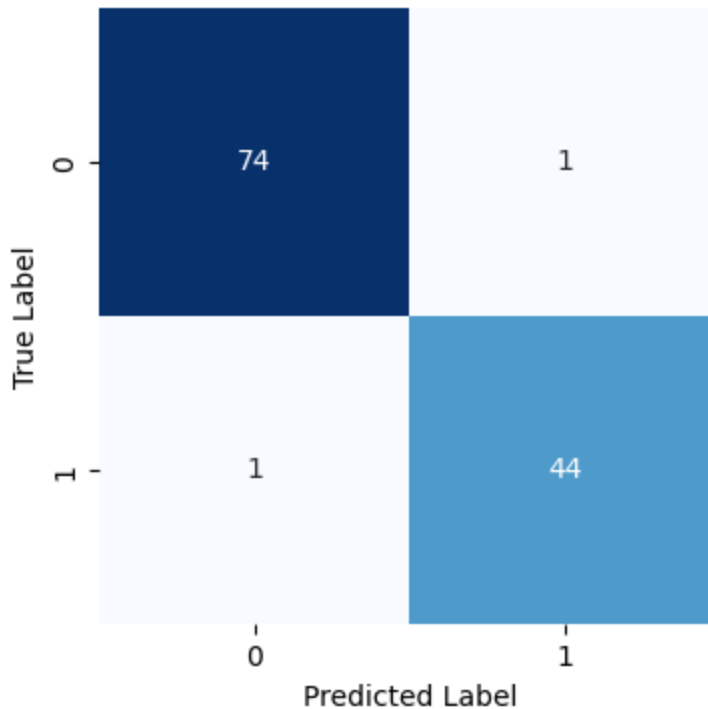
```

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va}

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



Model Name:

RandomForestClassifier

Accuracy: 0.9833

Precision: 0.9833

Recall: 0.9833

F1 Score: 0.9833

	precision	recall	f1-score	support
0	0.99	0.99	0.99	75
1	0.98	0.98	0.98	45
accuracy			0.98	120
macro avg	0.98	0.98	0.98	120
weighted avg	0.98	0.98	0.98	120

Best hyperparameters found by GridSearchCV:

```
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
```

LogisticRegression

```

In [24]: # Define the Logistic Regression model
LR = LogisticRegression(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10],          # Regularization strength
    'solver': ['liblinear', 'lbfgs'],  # Solver type
    'max_iter': [100, 200, 300]       # Maximum iterations for convergence
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5, scoring='accu

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_LR = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_LR.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

C:\Users\hager\anaconda3\envs\hhh\lib\site-packages\sklearn\linear_model_logistic.p
y:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

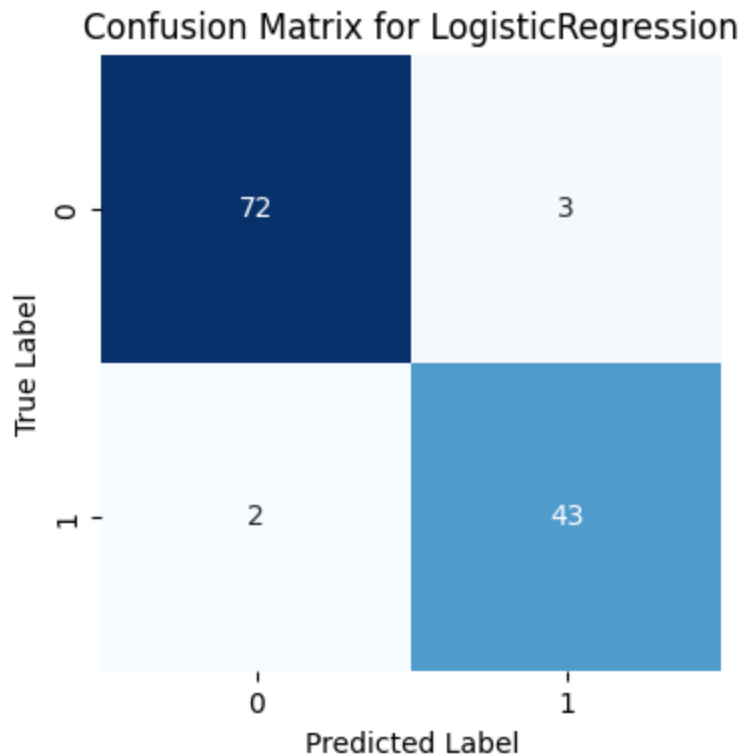
Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-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-regression

n_iter_i = _check_optimize_result(



Model Name:

LogisticRegression

Accuracy: 0.9583

Precision: 0.9587

Recall: 0.9583

F1 Score: 0.9584

	precision	recall	f1-score	support
0	0.97	0.96	0.97	75
1	0.93	0.96	0.95	45
accuracy			0.96	120
macro avg	0.95	0.96	0.96	120
weighted avg	0.96	0.96	0.96	120

Best hyperparameters found by GridSearchCV:

{'C': 0.1, 'max_iter': 300, 'solver': 'lbfgs'}

DecisionTreeClassifier

```
In [25]: # Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}
```

The function to measure the quality of a split
The maximum depth of the tree
The minimum number of samples required to split an internal node
The minimum number of samples required to be at a leaf node
The number of features to consider when looking for the best split

```

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5, scoring='accu

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_DT = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_DT.predict(X_test)

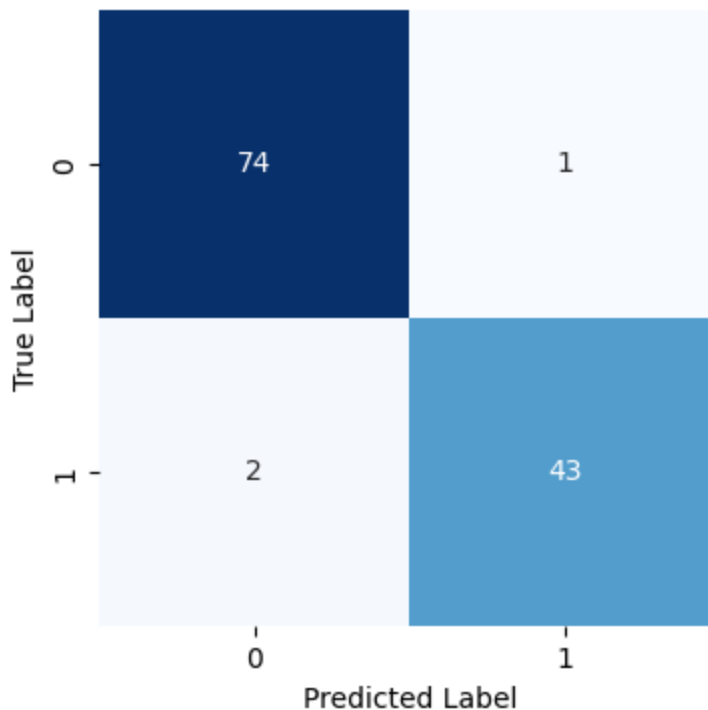
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for DecisionTreeClassifier



Model Name:
DecisionTreeClassifier
Accuracy: 0.9750
Precision: 0.9750
Recall: 0.9750
F1 Score: 0.9749

	precision	recall	f1-score	support
0	0.97	0.99	0.98	75
1	0.98	0.96	0.97	45
accuracy			0.97	120
macro avg	0.98	0.97	0.97	120
weighted avg	0.98	0.97	0.97	120

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

Support Vector Machine

```
In [13]: # Define the Support Vector Machine model
SVM = SVC(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10],          # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'], # Kernel type
    'gamma': ['scale', 'auto'],        # Kernel coefficient for 'rbf', 'poly',
    'degree': [3, 4, 5],              # Degree of the polynomial kernel funct
    'class_weight': [None, 'balanced'] # Weighing classes in the decision func
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5, scoring='acc

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

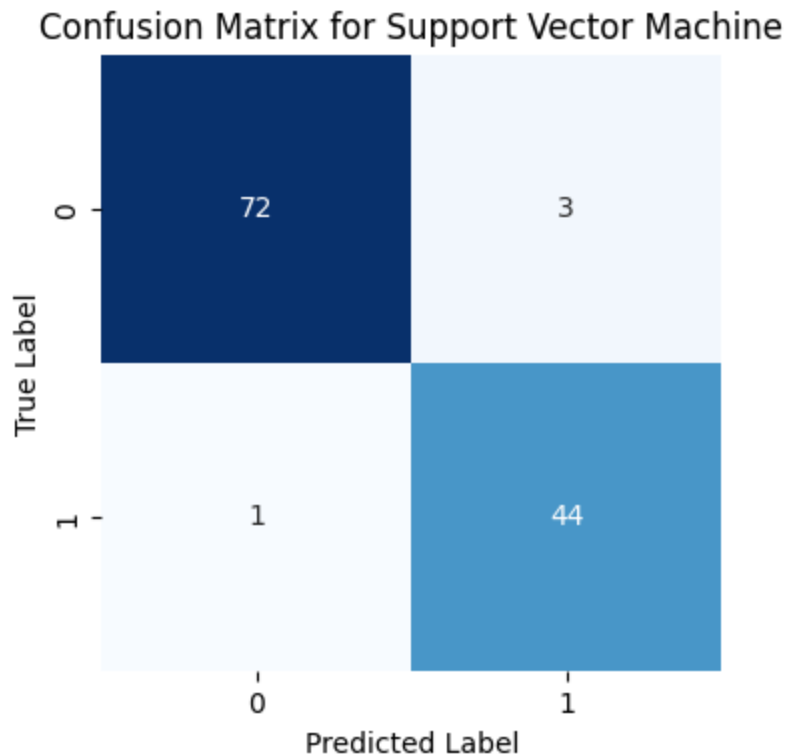
# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_SVM.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va
```

```
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:

Support Vector Machine

Accuracy: 0.9667

Precision: 0.9675

Recall: 0.9667

F1 Score: 0.9668

	precision	recall	f1-score	support
0	0.99	0.96	0.97	75
1	0.94	0.98	0.96	45
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

Best hyperparameters found by GridSearchCV:

```
{'C': 0.01, 'class_weight': None, 'degree': 3, 'gamma': 'scale', 'kernel': 'linear'}
```

GaussianNB

```
In [73]: # Define the Gaussian Naive Bayes model
gnb = GaussianNB()

# Define the hyperparameters for grid search
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6] # Smoothing parameter
```

```

}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=gnb, param_grid=param_grid, cv=5, scoring='acc

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_gnb = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_gnb.predict(X_test)

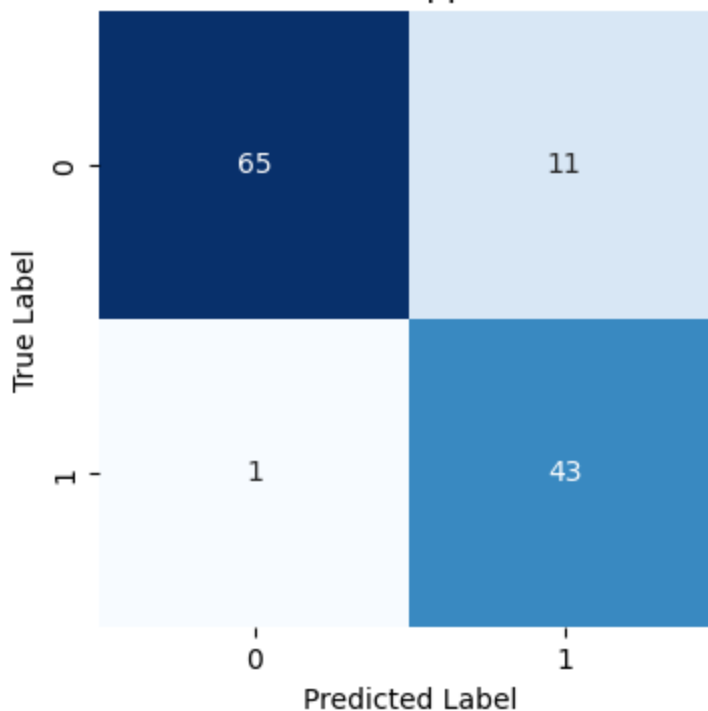
# Evaluate the model
evaluation_results = evaluate_model('Gaussian Naive Bayes', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for Support Vector Machine



Model Name:
Support Vector Machine
Accuracy: 0.9000
Precision: 0.9157
Recall: 0.9000
F1 Score: 0.9016

	precision	recall	f1-score	support
0	0.98	0.86	0.92	76
1	0.80	0.98	0.88	44
accuracy			0.90	120
macro avg	0.89	0.92	0.90	120
weighted avg	0.92	0.90	0.90	120

KNeighborsClassifier

```
In [75]: # Define the K-Nearest Neighbors model
knn = KNeighborsClassifier()

# Define the hyperparameters for grid search
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9],          # Number of neighbors to use
    'weights': ['uniform', 'distance'],      # Weight function used in prediction
    'metric': ['euclidean', 'manhattan'],    # Distance metric
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'] # Algorithm to compute
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='acc

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

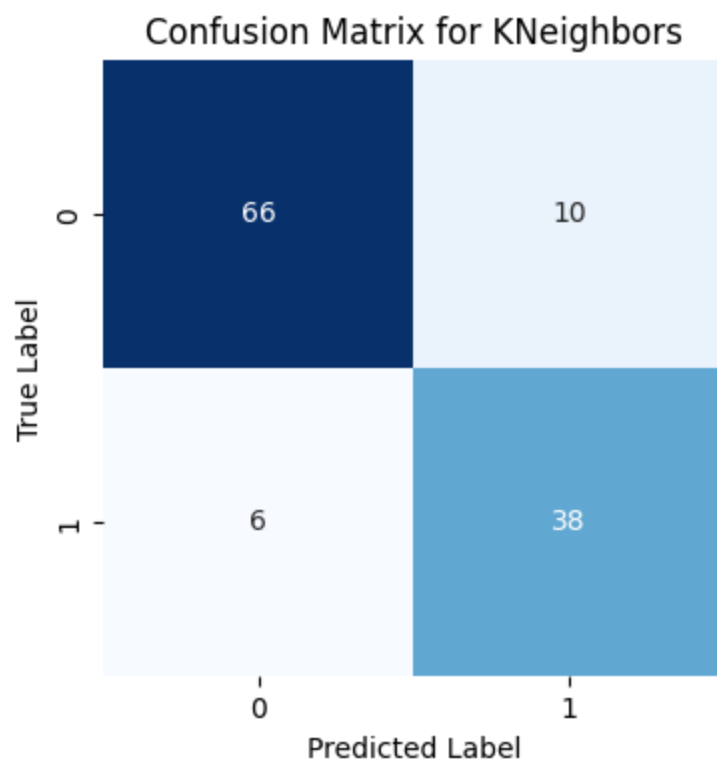
# Get the best estimator from grid search
best_knn = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_knn.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test, y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{va

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:

KNeighbors

Accuracy: 0.8667

Precision: 0.8708

Recall: 0.8667

F1 Score: 0.8678

	precision	recall	f1-score	support
0	0.92	0.87	0.89	76
1	0.79	0.86	0.83	44
accuracy			0.87	120
macro avg	0.85	0.87	0.86	120
weighted avg	0.87	0.87	0.87	120