

```
from IPython.display import Image, display
display(Image(filename='p1.png'))
```



What are the steps to develop an ML model?

- Data Collection: Gather relevant data from available sources.
- Data Cleaning: Handle missing values, outliers.
- Splitting the dataset
- Training models
- Evaluating models

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
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, f1_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading dataset

```
data = pd.read_csv('kidney_disease_clean_final.csv')
data.head()
```

	age	bp	sg	al	su	pc	pcc	ba	bgr	bu	...
0	48.0	80.0	1.020	1.0	0.0	1	0	0	121.000000	36.0	...
1	7.0	50.0	1.020	4.0	0.0	1	0	0	147.943503	18.0	...
2	62.0	80.0	1.010	2.0	3.0	1	0	0	423.000000	53.0	...
3	48.0	70.0	1.005	4.0	0.0	0	1	0	117.000000	56.0	...
4	51.0	80.0	1.010	2.0	0.0	1	0	0	106.000000	26.0	...

	wc	htn	dm	cad	appet	pe	ane	classification
0	72	1	4	1	0	0	0	0
1	56	0	3	1	0	0	0	0
2	70	0	4	1	1	0	1	0
3	62	1	3	1	1	1	1	0
4	68	0	3	1	0	0	0	0

```
[5 rows x 23 columns]
data['classification'].value_counts()
classification
0      248
1      150
Name: count, dtype: int64
```

Convert categorical features to numerical data.

```
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

```
X = data.drop('classification', axis=1).values # Replace
'classification' with your target column
y = data['classification'].values
```

Splitting data

What are the different methods of splitting a dataset?

1. Train–Test Split

The simplest form of splitting where the dataset is divided into:

- Training set: used to train the model.
- Testing set: used to evaluate model performance on unseen data.
- Typical ratio: 70/30, 80/20, or 90/10.

2. Train–Validation–Test Split

Common used in deep learning pipelines.

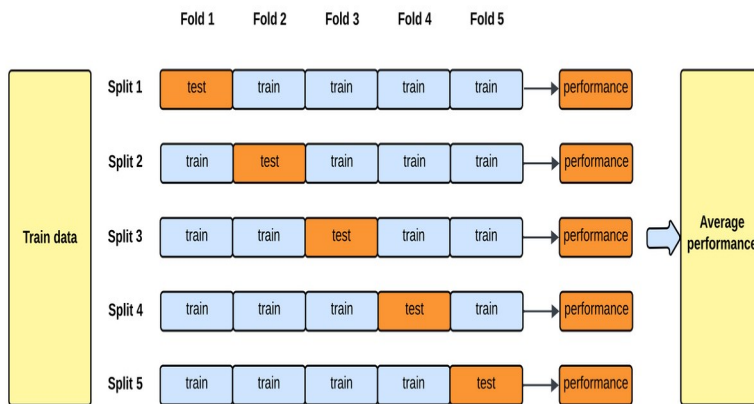
- Training set: used for learning parameters.
- Validation set: used for tuning hyperparameters and avoiding overfitting.
- Testing set: used for final evaluation only.
- Typical ratio: 70/10/20

3. Cross-Validation Splits

K-Fold Cross-Validation

In K-fold cross validation, the dataset is randomly split into n folds (usually 5). Over the course of 5 iterations, the model is trained on 4 out of the 5 folds while the remaining 1 acts as a test set for evaluating performance. This is repeated until all 5 folds have been used as a test set at one point in time. By the end of it, you'll have 5 error scores, which, averaged together, will give you your cross validation score.

```
from IPython.display import Image, display
display(Image(filename='CV.webp'))
```



```
# 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_state=42, stratify=y)
```

What are the different methods used to evaluate classification models?

1- Confusion matrix summarizes predictions versus actual labels.

For binary classification, it includes:

- TP (True Positive): Correctly predicted positives
 - TN (True Negative): Correctly predicted negatives
 - FP (False Positive): Incorrectly predicted as positive
 - FN (False Negative): Incorrectly predicted as negative
1. Accuracy The proportion of correct predictions made by a classification model out of the total number of predictions. It is calculated by dividing the sum of correct classifications (true positives + true negatives) by the total number of classifications. A model with 100% accuracy has made no errors

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision (Positive Predictive Value)

It measures the reliability of positive predictions, indicating how often a model's positive classification is actually correct.

Example use case: In medical diagnostics, precision represents how many patients identified as having a disease actually have it — critical in minimizing unnecessary treatments or false alarms.

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. Recall

Recall, also referred to as Sensitivity or the True Positive Rate (TPR), measures the proportion of actual positive instances that are correctly identified by the model.

It evaluates the model's ability to detect all relevant positive cases within the dataset. A high recall value indicates that the model successfully captures most of the true positives, while a low recall signifies that many actual positives were missed (false negatives).

Example use case: In disease screening (e.g., cancer detection), recall represents how many patients who truly have the disease are correctly identified.

$$\text{Recall} = \frac{TP}{TP + FN}$$

1. F1-score:

The F1-score is particularly useful when the dataset is imbalanced.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Function for evaluation models

```
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

```

Types of ML models

- Logistic Regression (LR)
- Decision Trees (DT)
- Random Forest (RF)
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)
- GaussianNB

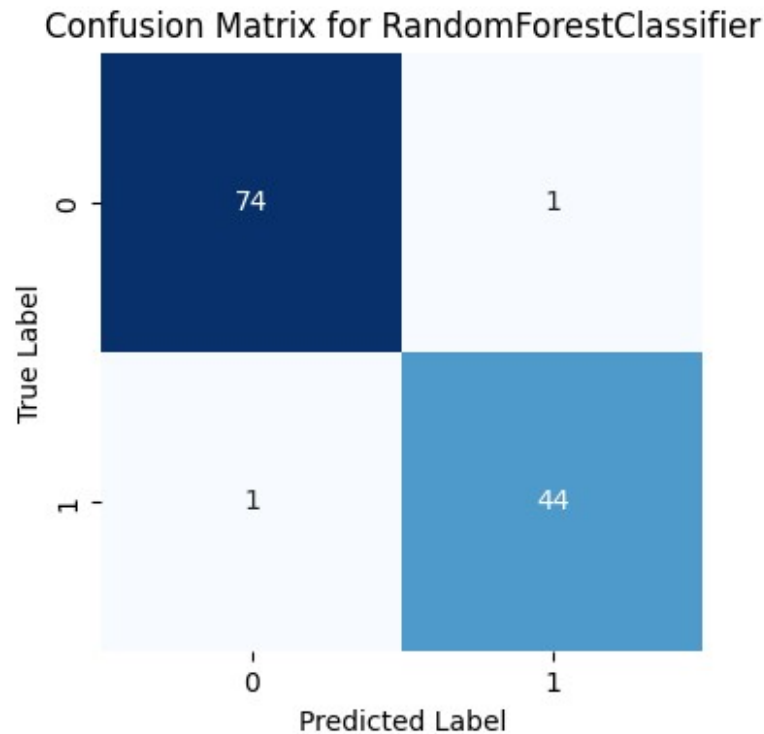
Random Forest Classifier

```

rf_classifier = RandomForestClassifier(n_estimators=100,
                                     random_state=42)
# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = 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{value}")

```



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

LogisticRegression

```
LR = LogisticRegression()
# Train the model
LR.fit(X_train, y_train)
# Make predictions
y_pred = 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{value}")
```

C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\linear_model_logistic.py:473: ConvergenceWarning: lbfgs failed to converge after 100 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence
(max_iter=100).

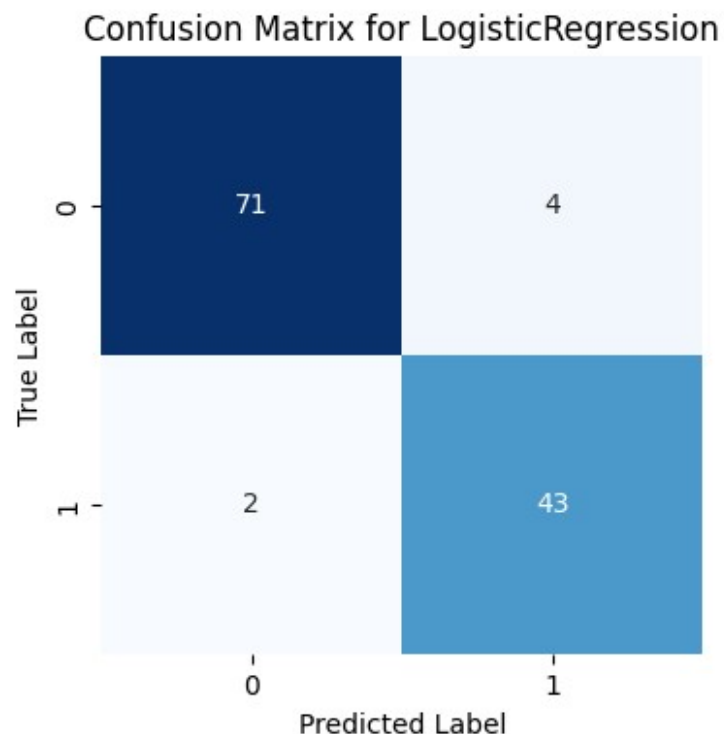
You might also want to scale the data as shown in:

<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: 0.9500

Precision: 0.9510

Recall: 0.9500

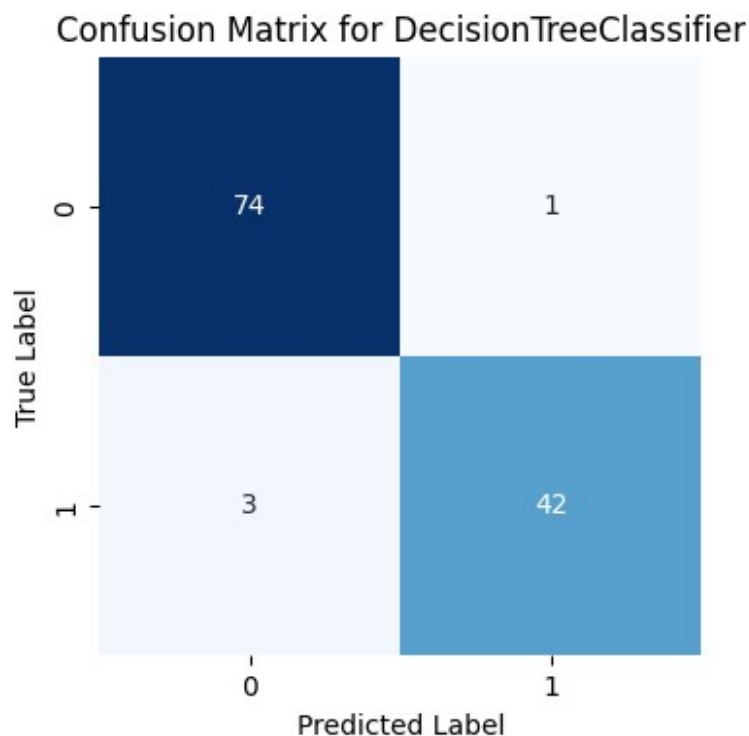
F1 Score: 0.9502

	precision	recall	f1-score	support
0	0.97	0.95	0.96	75
1	0.91	0.96	0.93	45
accuracy			0.95	120
macro avg	0.94	0.95	0.95	120
weighted avg	0.95	0.95	0.95	120

DecisionTreeClassifier

```
DT = DecisionTreeClassifier()
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = 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{value}")
```



Model Name:
DecisionTreeClassifier
Accuracy: 0.9667
Precision: 0.9669
Recall: 0.9667
F1 Score: 0.9665

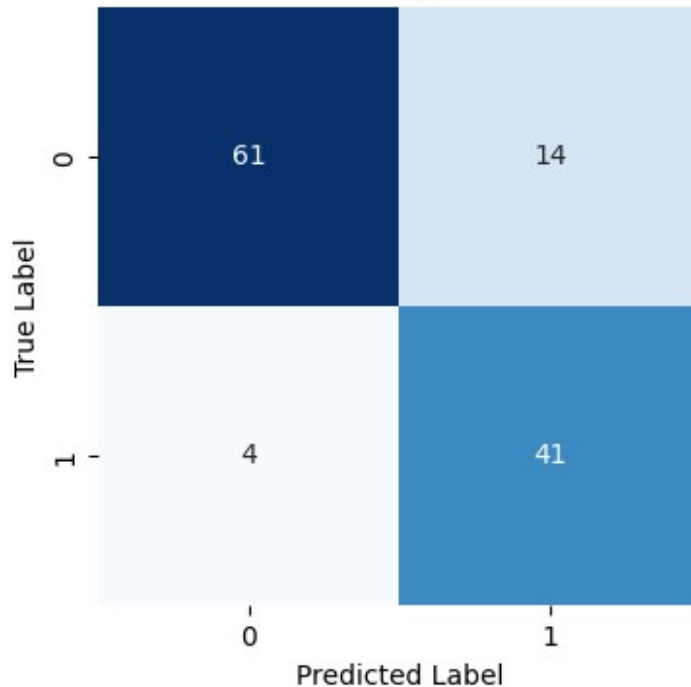
	precision	recall	f1-score	support
0	0.96	0.99	0.97	75
1	0.98	0.93	0.95	45
accuracy			0.97	120
macro avg	0.97	0.96	0.96	120
weighted avg	0.97	0.97	0.97	120

Support Vector Machine

```
SVM = SVC()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
y_pred = 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{value}")
```

Confusion Matrix for Support Vector Machine



Model Name:
Support Vector Machine
Accuracy: 0.8500
Precision: 0.8661
Recall: 0.8500
F1 Score: 0.8521

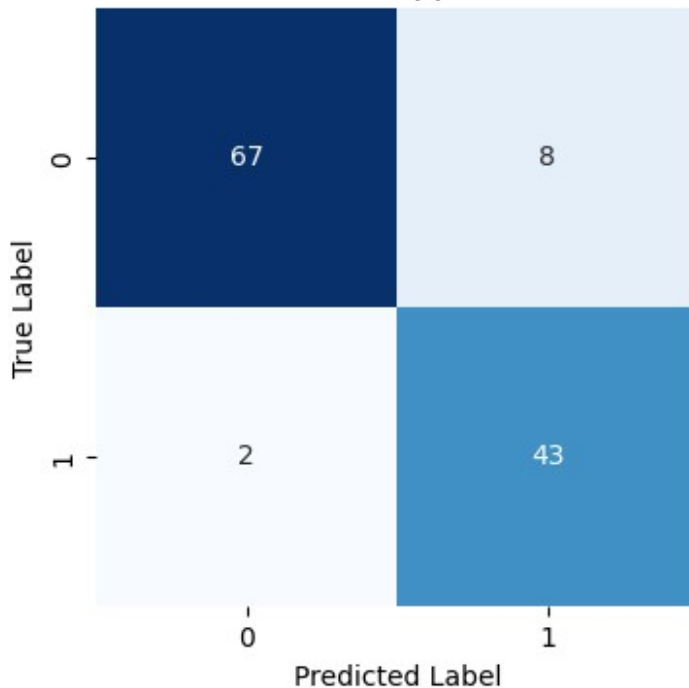
	precision	recall	f1-score	support
0	0.94	0.81	0.87	75
1	0.75	0.91	0.82	45
accuracy			0.85	120
macro avg	0.84	0.86	0.85	120
weighted avg	0.87	0.85	0.85	120

GaussianNB

```
gnb = GaussianNB()
# Train the model
gnb.fit(X_train, y_train)
# Make predictions
y_pred = gnb.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{value}")
```

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.9167

Precision: 0.9231

Recall: 0.9167

F1 Score: 0.9175

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

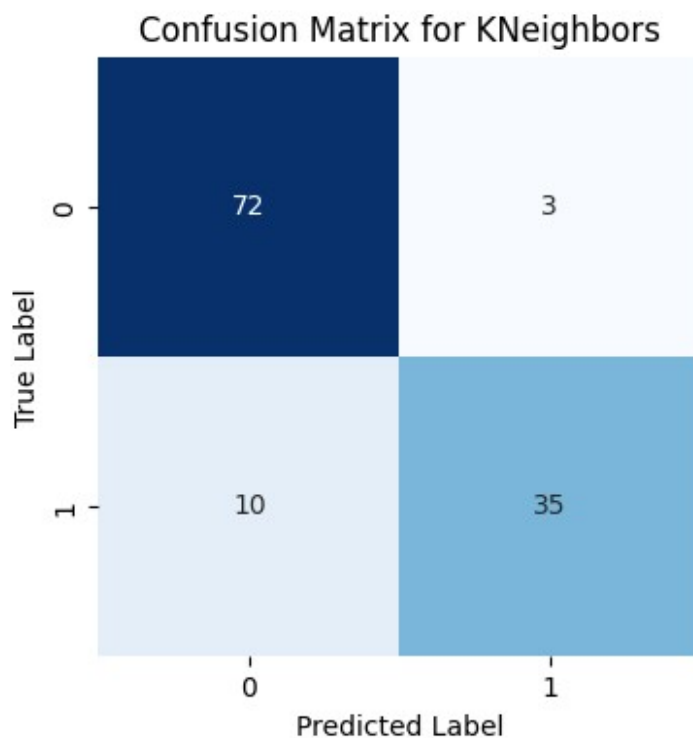
	0	0.97	0.89	0.93	75
	1	0.84	0.96	0.90	45
accuracy				0.92	120
macro avg		0.91	0.92	0.91	120
weighted avg		0.92	0.92	0.92	120

KNeighborsClassifier

```
knn = KNeighborsClassifier(n_neighbors=2)

knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighbors', 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{value}")
```



Model Name:

KNeighbors

Accuracy: 0.8917

Precision: 0.8942

Recall: 0.8917

F1 Score: 0.8895

	precision	recall	f1-score	support
0	0.88	0.96	0.92	75
1	0.92	0.78	0.84	45
accuracy			0.89	120
macro avg	0.90	0.87	0.88	120
weighted avg	0.89	0.89	0.89	120

Hyperparameter optimization

Grid Search is a hyperparameter optimization technique used to find the best combination of parameters for a machine learning model.

Random Forest Classifier with grid search

```
from sklearn.model_selection import GridSearchCV
# 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, scoring='accuracy', n_jobs=-1)

# 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,
```

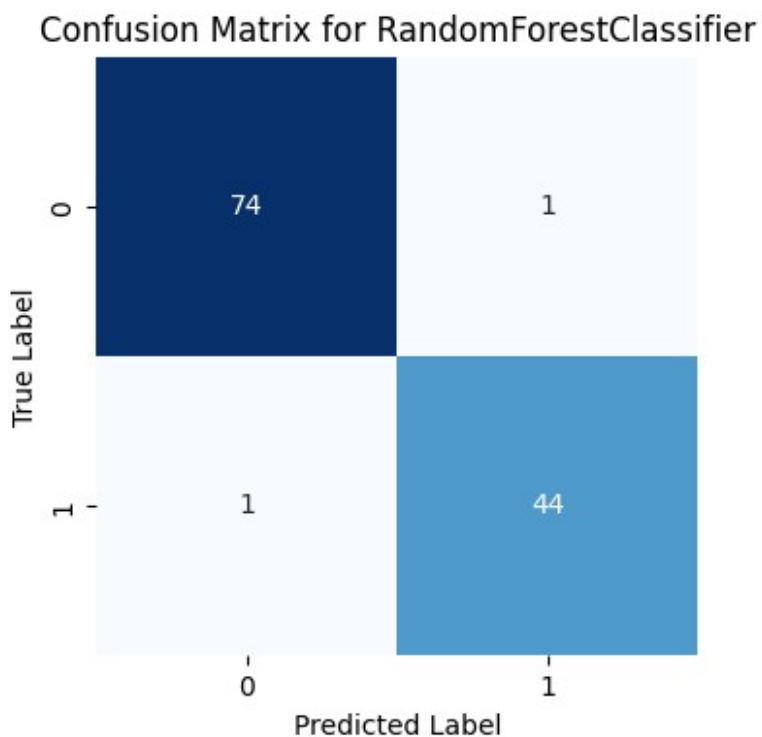
```

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{value}")

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

```



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 with grid search

```
# 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='accuracy', n_jobs=-1)

# 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{value}")

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

```
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 300 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=300).

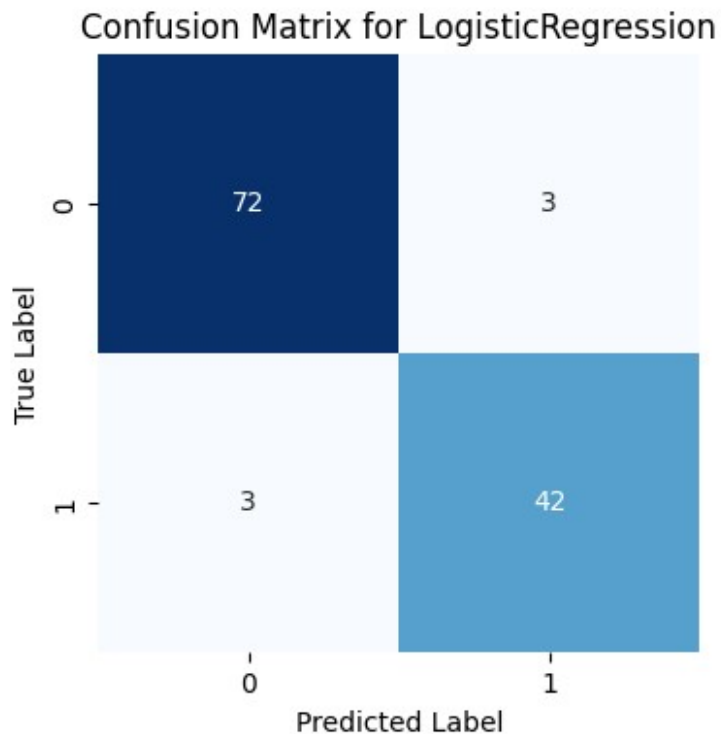
You might also want to scale the data as shown in:

<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: 0.9500

Precision: 0.9500

Recall: 0.9500

F1 Score: 0.9500

	precision	recall	f1-score	support
0	0.96	0.96	0.96	75
1	0.93	0.93	0.93	45

accuracy			0.95	120
macro avg	0.95	0.95	0.95	120
weighted avg	0.95	0.95	0.95	120

Best hyperparameters found by GridSearchCV:
{'C': 0.1, 'max_iter': 300, 'solver': 'lbfgs'}

DecisionTreeClassifier with grid search

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],           # The function to
    'max_depth': [None, 10, 20, 30],           # The maximum depth
    'min_samples_split': [2, 5, 10],           # The minimum number
    'min_samples_leaf': [1, 2, 4],             # The minimum number
    'max_features': [None, 'sqrt', 'log2']     # 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='accuracy', n_jobs=-1)

# 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_

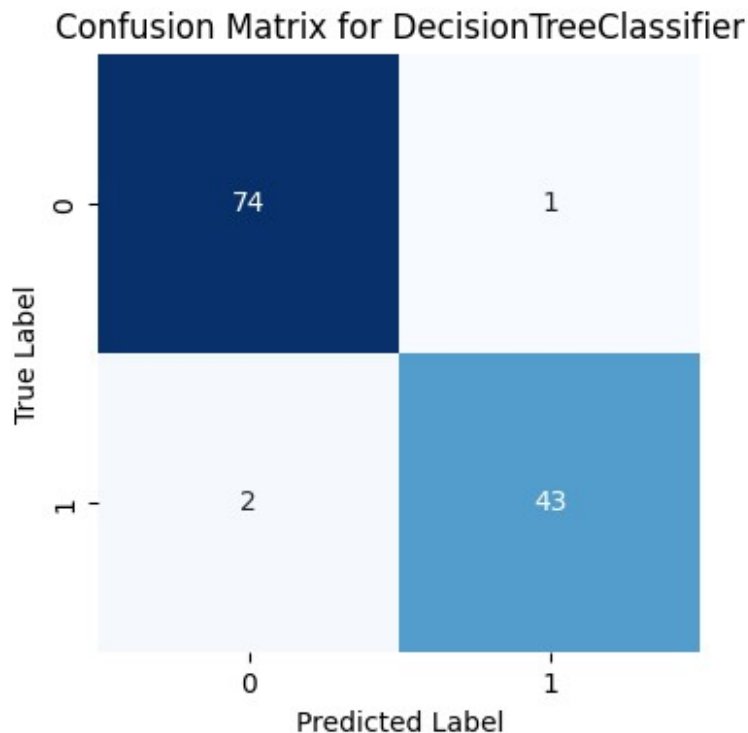
# Make predictions using the best model
y_pred = best_DT.predict(X_test)

# 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{value}")

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



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 with grid search

```
# 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', 'sigmoid'
    'degree': [3, 4, 5],              # Degree of the polynomial
    kernel_function('poly')
    'class_weight': [None, 'balanced'] # Weighing classes in the
    decision function
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# 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_

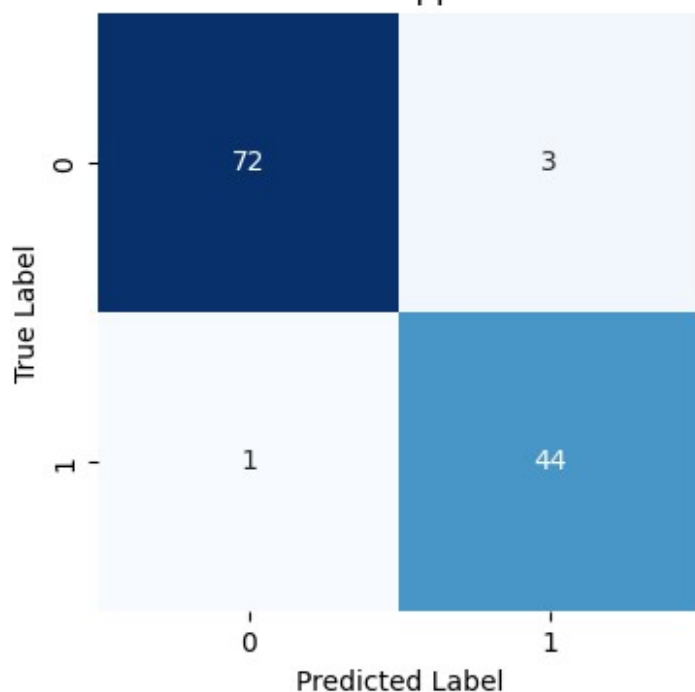
# Make predictions using the best model
y_pred = best_SVM.predict(X_test)

# 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{value}")

# 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: 0.9667

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 with grid search

```
# 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='accuracy', n_jobs=-1)

# 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_

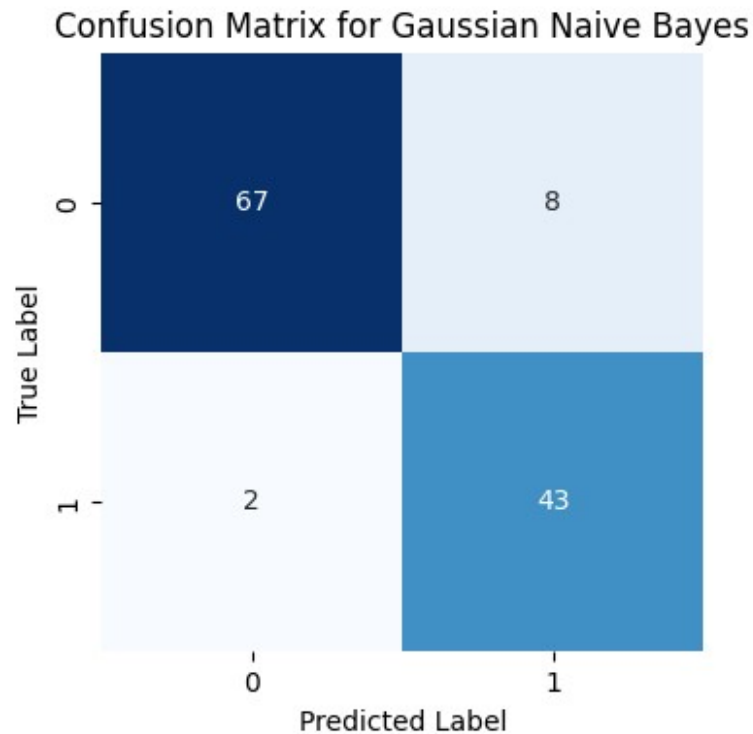
# Make predictions using the best model
y_pred = best_gnb.predict(X_test)

# 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{value}")

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

```



Model Name:
Gaussian Naive Bayes
Accuracy: 0.9167
Precision: 0.9231
Recall: 0.9167
F1 Score: 0.9175

	precision	recall	f1-score	support
0	0.97	0.89	0.93	75
1	0.84	0.96	0.90	45
accuracy			0.92	120
macro avg	0.91	0.92	0.91	120
weighted avg	0.92	0.92	0.92	120

Best hyperparameters found by GridSearchCV:
{'var_smoothing': 1e-09}

KNeighborsClassifier with grid search

```
# 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 the nearest neighbors
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# 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_

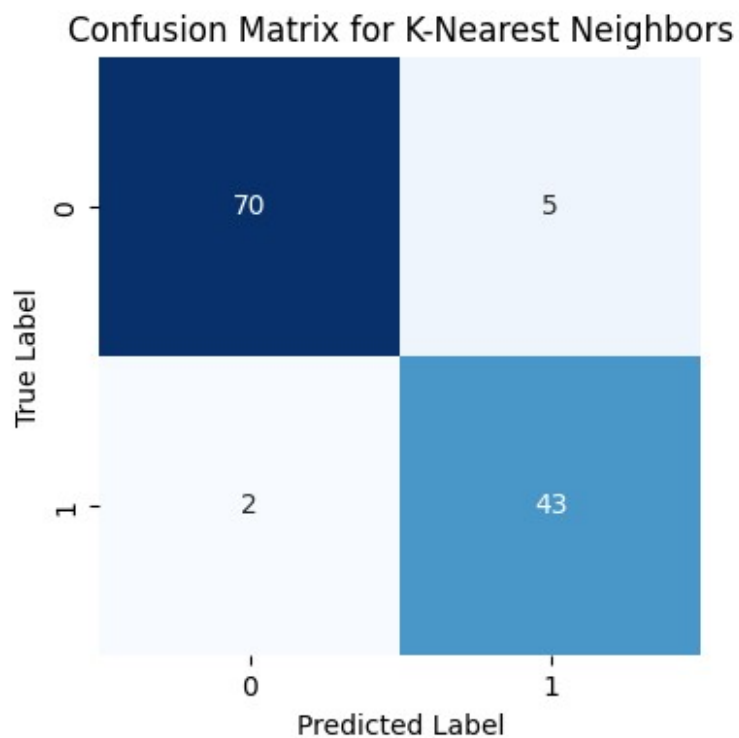
# Make predictions using the best model
y_pred = best_knn.predict(X_test)

# 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{value}")

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

```



Model Name:

K-Nearest Neighbors

Accuracy: 0.9417

Precision: 0.9436

Recall: 0.9417

F1 Score: 0.9420

	precision	recall	f1-score	support
0	0.97	0.93	0.95	75
1	0.90	0.96	0.92	45
accuracy			0.94	120
macro avg	0.93	0.94	0.94	120
weighted avg	0.94	0.94	0.94	120

Best hyperparameters found by GridSearchCV:

```
{'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'}
```