

In [83]:



What Are the Steps to Develop an ML Model?

- Data Collection: Gather relevant data from available sources.
- Data Cleaning: Handle missing values, outliers.
- Splitting dataset
- Choose a Model and Train: Logistic Regression, Decision Trees, Random Forest, Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Naïve Bayes
- Performance Metrics: accuracy, precision, recall, F1 score to evaluate the model's performance.

Import necessary libraries

```
In [60]: 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
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Reading dataset

```
In [63]: data = pd.read_csv('kidney_disease_clean_final.csv')
```

```
In [64]: data.head()
```

```
Out[64]:
```

	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 [79]: data['classification'].value_counts()
```

```
Out[79]: classification
0      248
1      150
Name: count, dtype: int64
```

Convert categorical features to numerical data.

```
In [65]: 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 [66]: X = data.drop('classification', axis=1).values # Replace 'classification' with you
y = data['classification'].values
```

Splitting data

```
In [67]: # 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 [68]: 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

```

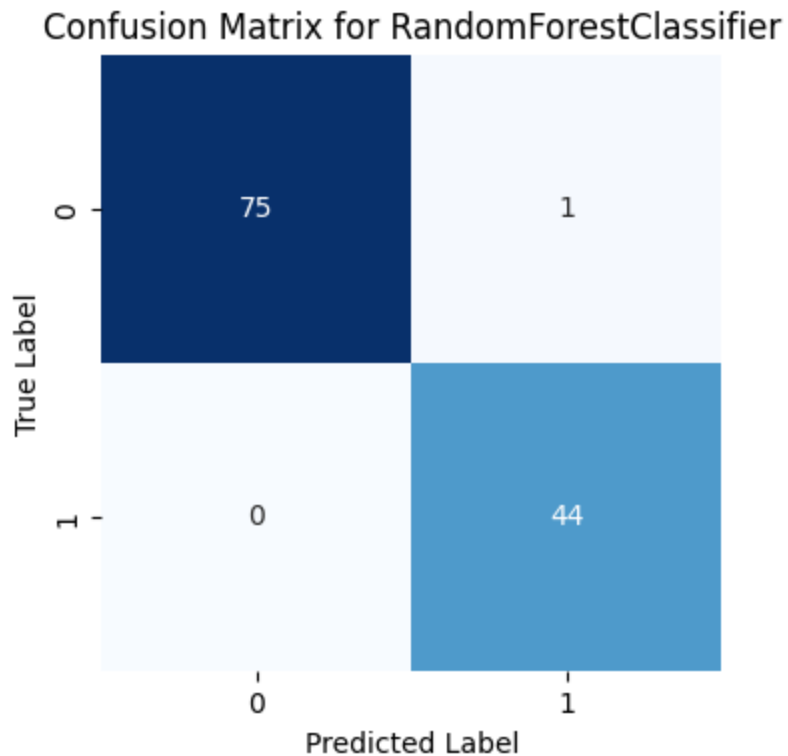
Random Forest Classifier

```

In [69]: 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{va

```



Model Name:

RandomForestClassifier

Accuracy: 0.9917

Precision: 0.9919

Recall: 0.9917

F1 Score: 0.9917

	precision	recall	f1-score	support
0	1.00	0.99	0.99	76
1	0.98	1.00	0.99	44
accuracy			0.99	120
macro avg	0.99	0.99	0.99	120
weighted avg	0.99	0.99	0.99	120

LogisticRegression

```
In [70]: 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{va

```

C:\Users\hager\anaconda3\envs\hhh\lib\site-packages\sklearn\linear_model_logistic.py: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:

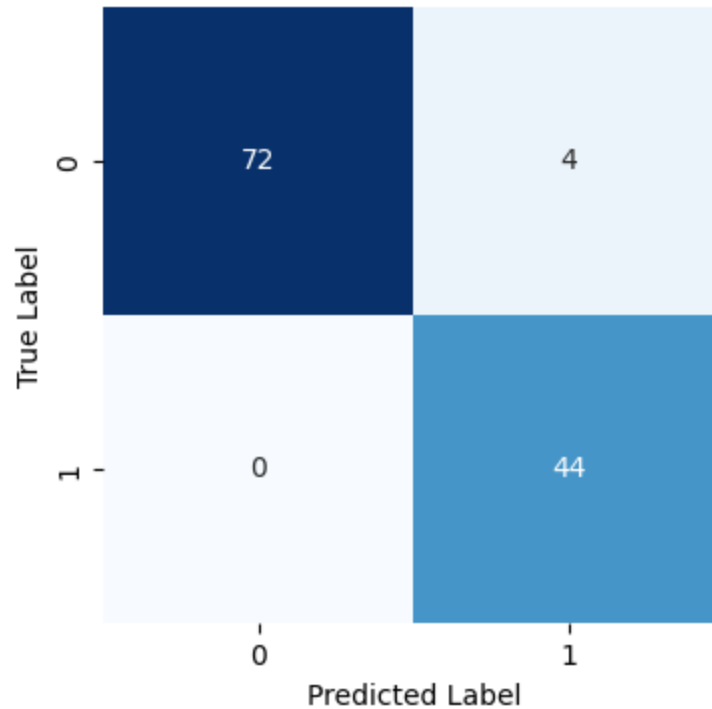
<https://scikit-learn.org/stable/modules/preprocessing.html>

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(

Confusion Matrix for LogisticRegression



Model Name:

LogisticRegression

Accuracy: 0.9667

Precision: 0.9694

Recall: 0.9667

F1 Score: 0.9669

	precision	recall	f1-score	support
0	1.00	0.95	0.97	76
1	0.92	1.00	0.96	44
accuracy			0.97	120
macro avg	0.96	0.97	0.96	120
weighted avg	0.97	0.97	0.97	120

DecisionTreeClassifier

```

In [71]: DT = DecisionTreeClassifier()
          # Train the model

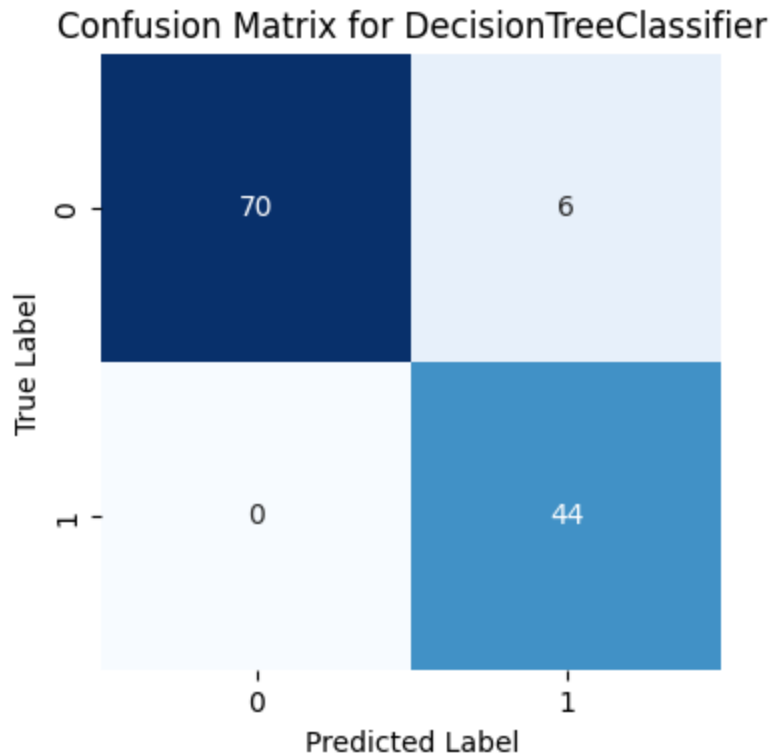
```

```

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{va

```



Model Name:

DecisionTreeClassifier

Accuracy: 0.9500

Precision: 0.9560

Recall: 0.9500

F1 Score: 0.9506

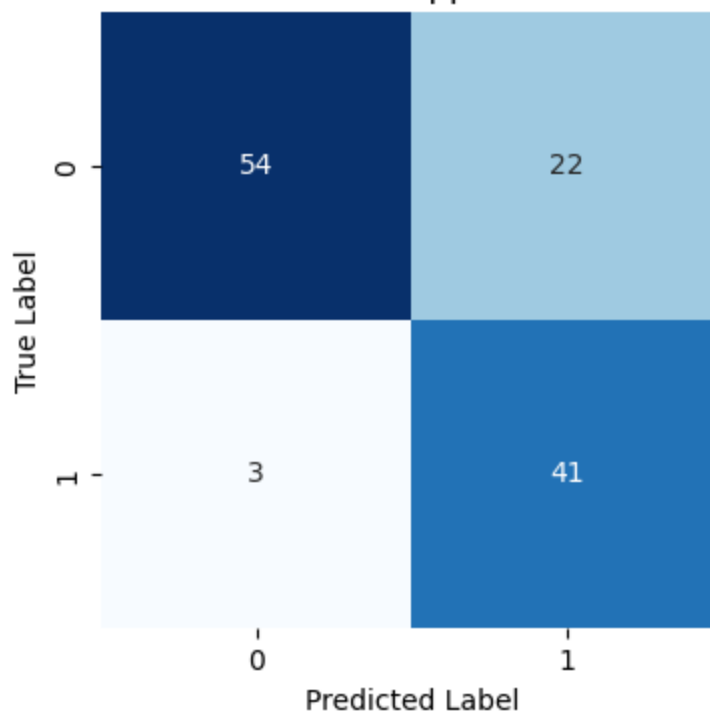
	precision	recall	f1-score	support
0	1.00	0.92	0.96	76
1	0.88	1.00	0.94	44
accuracy			0.95	120
macro avg	0.94	0.96	0.95	120
weighted avg	0.96	0.95	0.95	120

Support Vector Machine

```
In [72]: 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{va
```

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.7917

Precision: 0.8386

Recall: 0.7917

F1 Score: 0.7953

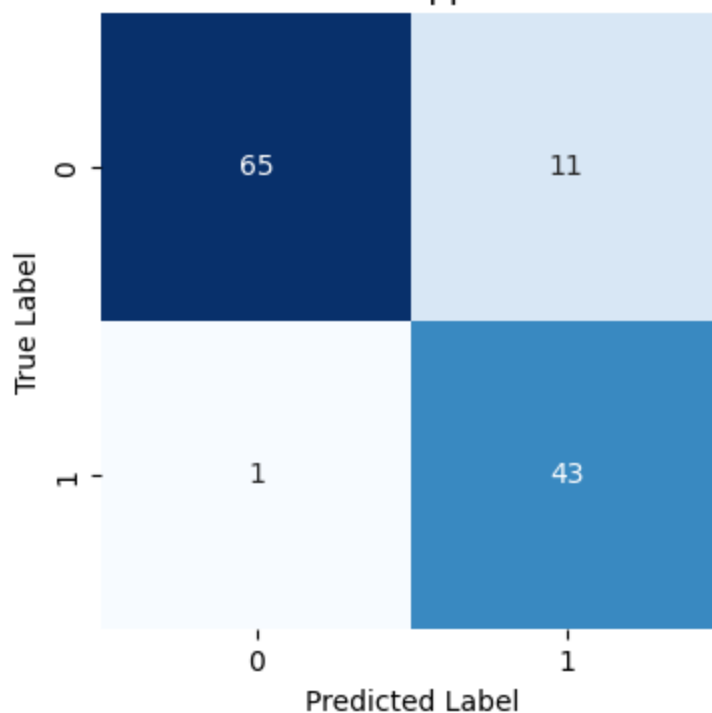
	precision	recall	f1-score	support
0	0.95	0.71	0.81	76
1	0.65	0.93	0.77	44
accuracy			0.79	120
macro avg	0.80	0.82	0.79	120
weighted avg	0.84	0.79	0.80	120

GaussianNB

```
In [73]: 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{va
```

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

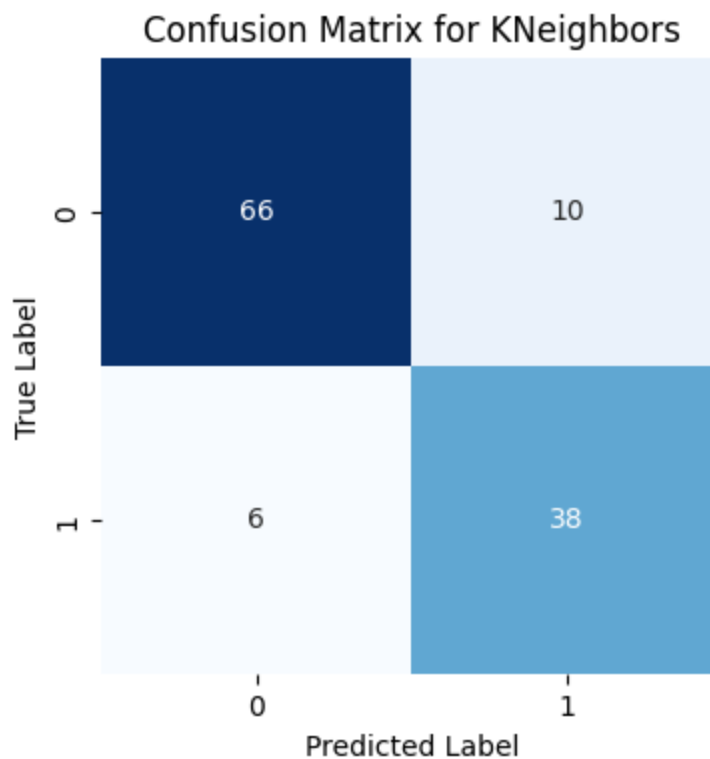
```
In [74]: KNeighborsClassifier
```

```
Out[74]: sklearn.neighbors._classification.KNeighborsClassifier
```

```
In [75]: 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{va
```



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