Load the required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import shap
import janitor
import warnings
warnings.filterwarnings('ignore')
```

Load the dataset

In [2]:
```
df = pd.read_csv("C:/Users/ADMIN/Desktop/Data Science/Datasets/Datasets/Heart_At
```

Inspect the dataset

In [3]:
```
df.head(10)
```

Out[3]:

	age	gender	heart_rate	systolic_blood_pressure	diastolic_blood_pressure	blood_sug
0	63	1	66	160	83	160
1	20	1	94	98	46	296
2	56	1	64	160	77	270
3	66	1	70	120	55	270
4	54	1	64	112	65	300
5	52	0	61	112	58	87
6	38	0	40	179	68	102
7	61	1	60	214	82	87
8	49	0	60	154	81	135
9	65	1	61	160	95	100

Last few observations of the dataset

In [13]:
```
df.tail(10)
```

Out[13]:

	age	gender	heart_rate	systolic_blood_pressure	diastolic_blood_pressure	blood_
1309	47	1	94	105	81	
1310	70	0	80	135	75	
1311	85	1	112	115	69	
1312	48	1	84	118	68	
1313	86	0	40	179	68	
1314	44	1	94	122	67	
1315	66	1	84	125	55	
1316	45	1	85	168	104	
1317	54	1	58	117	68	
1318	51	1	94	157	79	

Data Types Check

In [15]:
```
df.dtypes
```

Out[15]:
```
age                            int64
gender                         int64
heart_rate                     int64
systolic_blood_pressure        int64
diastolic_blood_pressure       int64
blood_sugar                  float64
ck_mb                        float64
troponin                     float64
result                         int32
dtype: object
```

Structure of the dataset

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1319 entries, 0 to 1318
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       1319 non-null   int64  
 1   gender                    1319 non-null   int64  
 2   heart_rate                1319 non-null   int64  
 3   systolic_blood_pressure   1319 non-null   int64  
 4   diastolic_blood_pressure  1319 non-null   int64  
 5   blood_sugar               1319 non-null   float64
 6   ck_mb                     1319 non-null   float64
 7   troponin                  1319 non-null   float64
 8   result                    1319 non-null   int32  
dtypes: float64(3), int32(1), int64(5)
memory usage: 87.7 KB
```

Unique Values and Cardinality

In [16]:
```python
df.nunique()
```

Out[16]:
```
age                          75
gender                        2
heart_rate                   79
systolic_blood_pressure     116
diastolic_blood_pressure     73
blood_sugar                 244
ck_mb                       700
troponin                    352
result                        2
dtype: int64
```

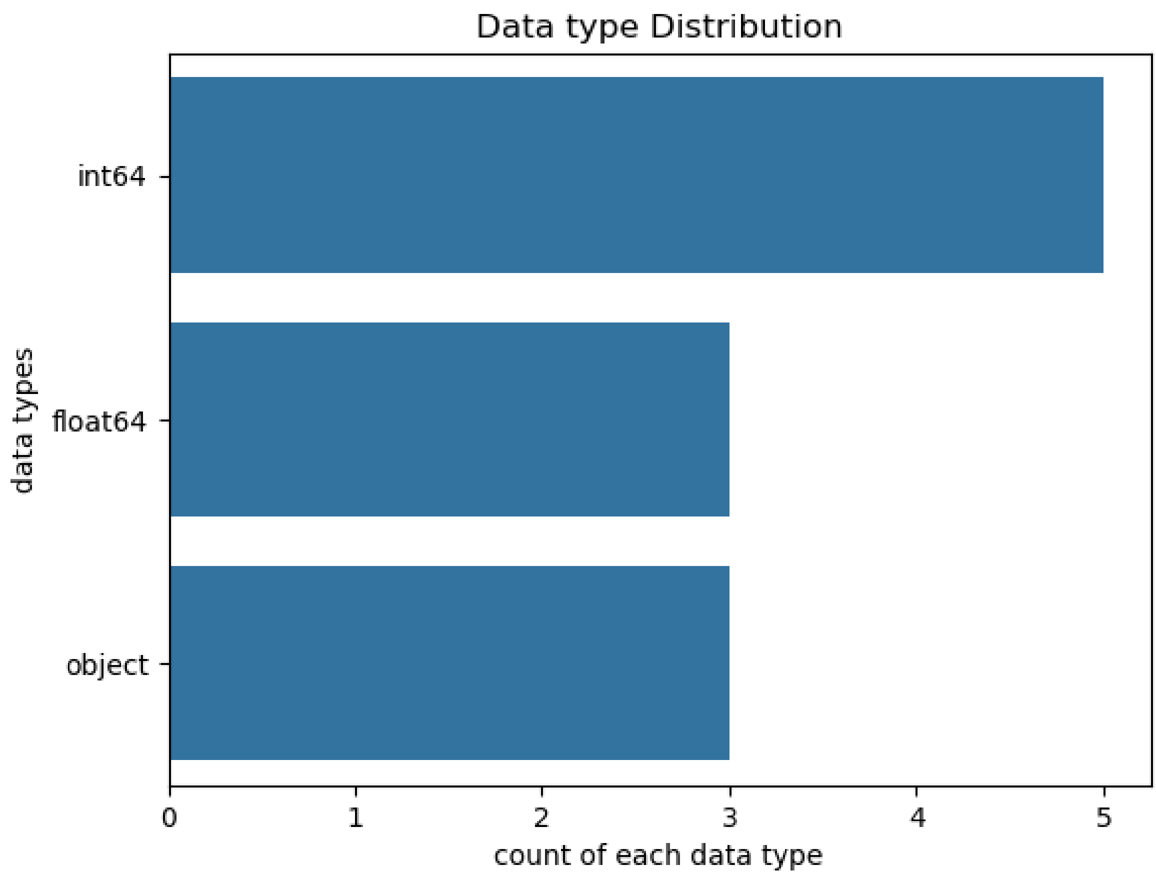
Checking for missing values

In [5]:
```python
df.isnull().sum()
```

Out[5]:
```
age                         0
gender                      0
heart_rate                  0
systolic_blood_pressure     0
diastolic_blood_pressure    0
blood_sugar                 0
ck_mb                       0
troponin                    0
result                      0
risk_level                  0
recommendation              0
dtype: int64
```

Data type distribution

In [6]:
```python
sns.countplot(y=df.dtypes ,data=df)
plt.title("Data type Distribution")
plt.xlabel("count of each data type")
plt.ylabel("data types")
plt.show()
```



Check for duplicates

In [7]:
```
df.duplicated().sum()
```

Out[7]: 0

Data Cleaning & Preprocessing

Drop unwanted columns

In [8]:
```
df = df.drop(columns=['risk_level', 'recommendation'])
```

Label encoding of the target variable

In [10]:
```
## Initialize LabelEncoder
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()

# Apply label encoding to the 'Result' column
df['result'] = label.fit_transform(df['result'])
```

Exploratory Data Analysis

Summary Statistics

In [11]:
```
df.describe()
```

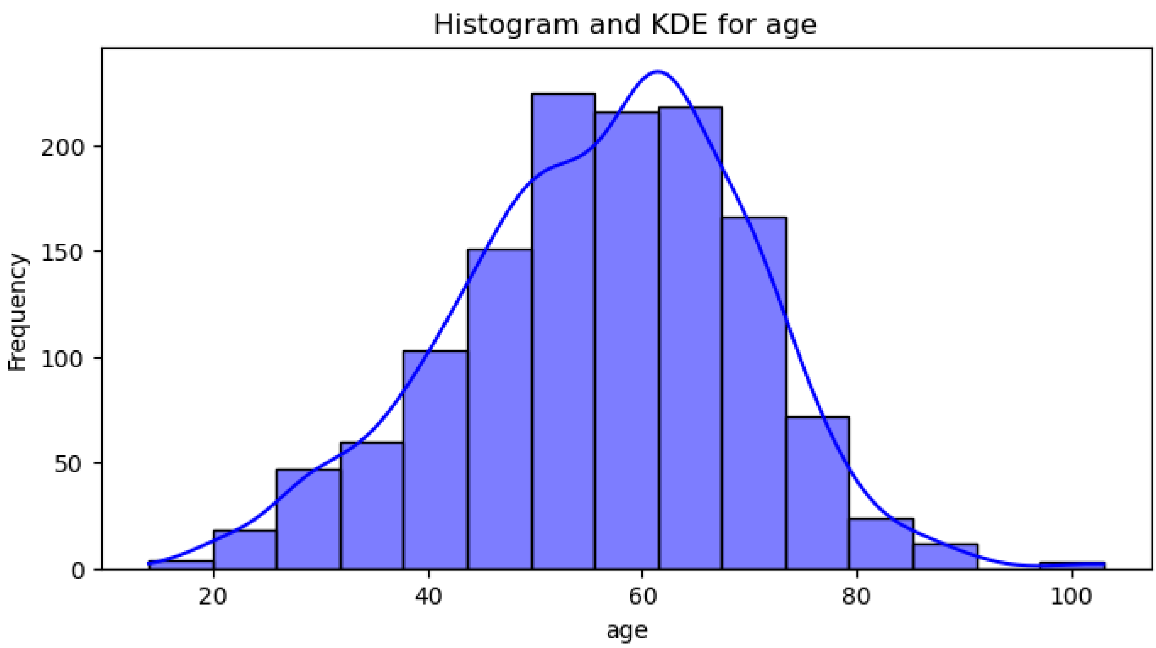
Out[11]:

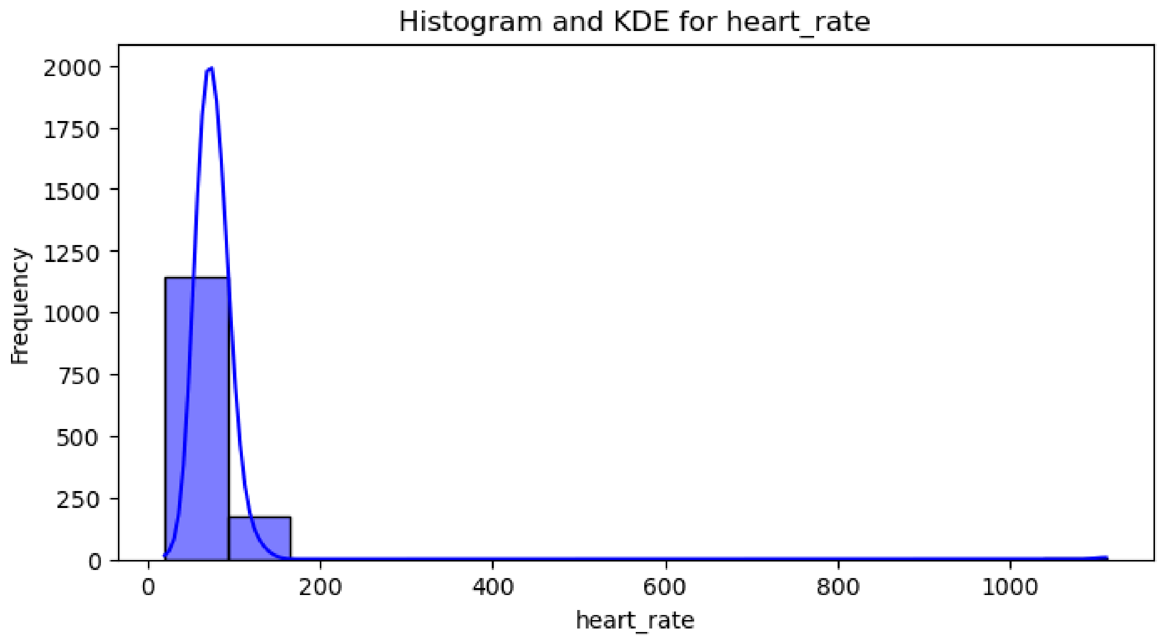
	age	gender	heart_rate	systolic_blood_pressure	diastolic_blood_pı
count	1319.000000	1319.000000	1319.000000	1319.000000	1319.
mean	56.193328	0.659591	78.336619	127.170584	72.
std	13.638173	0.474027	51.630270	26.122720	14.
min	14.000000	0.000000	20.000000	42.000000	38.
25%	47.000000	0.000000	64.000000	110.000000	62.
50%	58.000000	1.000000	74.000000	124.000000	72.
75%	65.000000	1.000000	85.000000	143.000000	81.
max	103.000000	1.000000	1111.000000	223.000000	154.

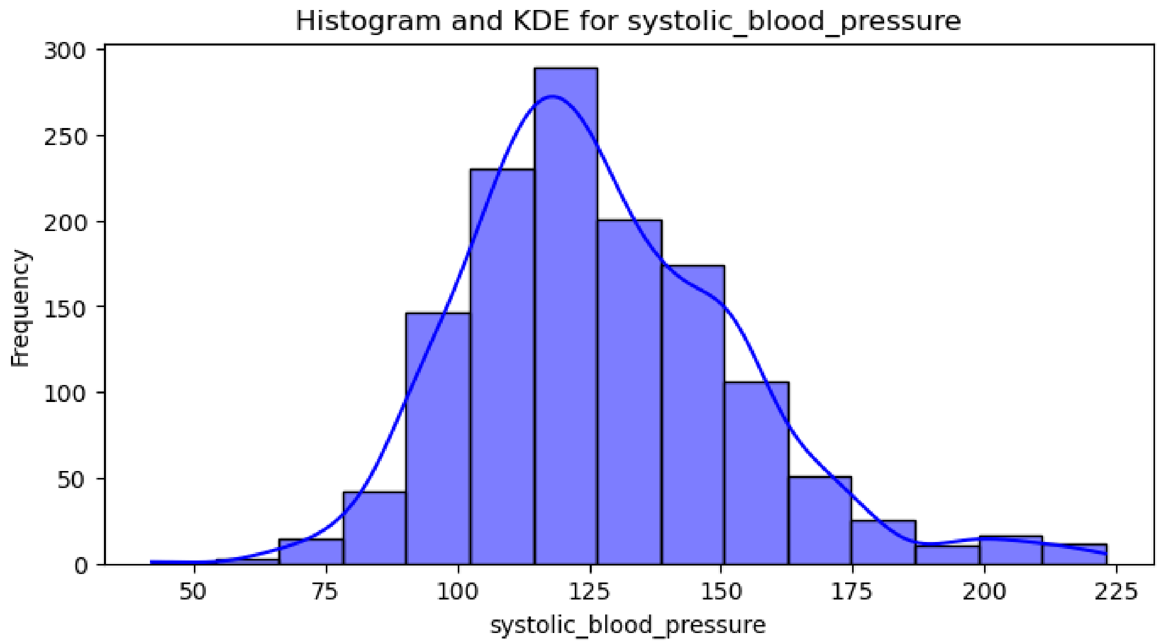
Distribution of Numerical Variables

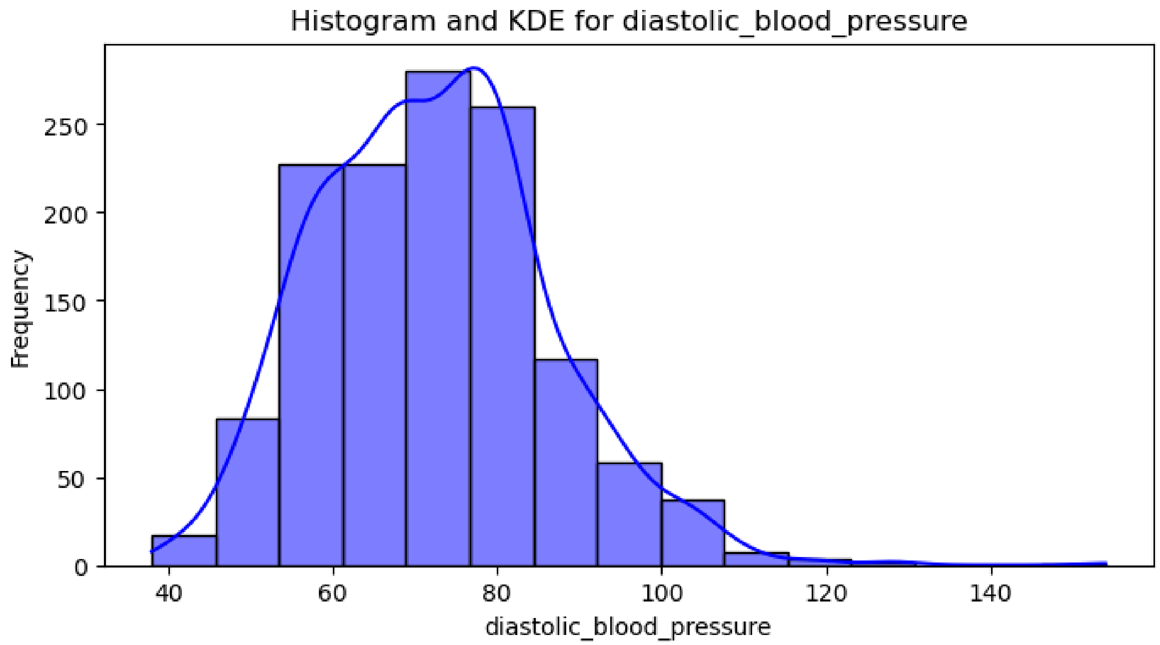
In [45]:

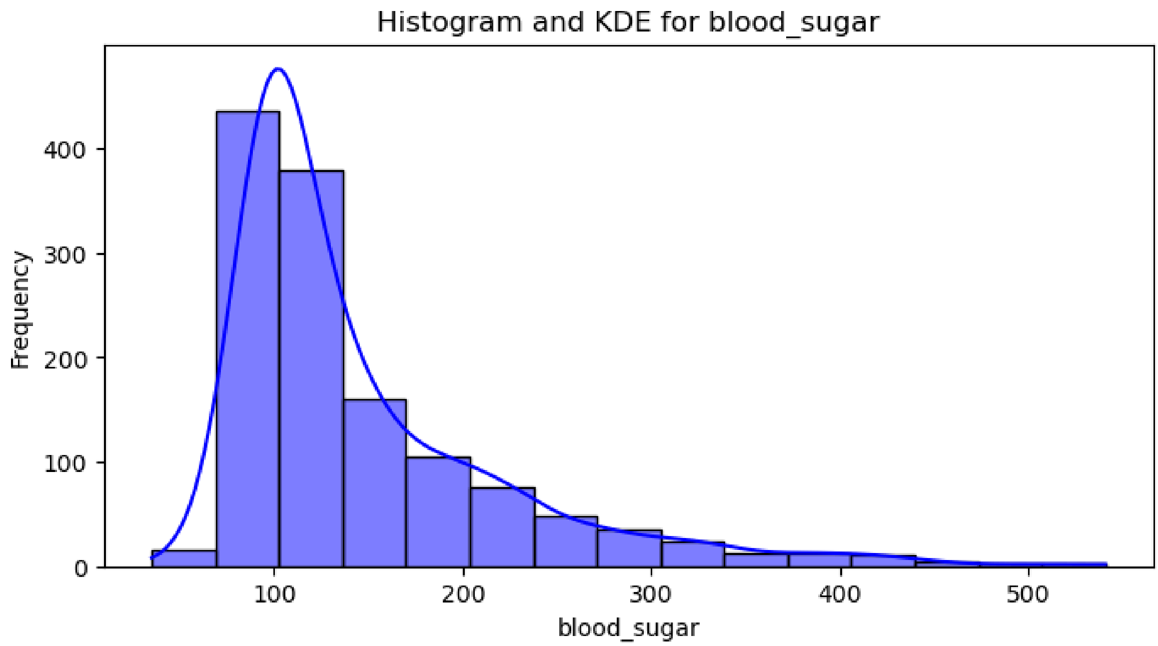
```
numeric_col = df.drop(columns = ["result", "gender"])
for col in numeric_col:
    plt.figure(figsize = (8, 4))
    sns.histplot(df[col], kde = True, bins = 15, color = "blue")
    plt.title(f'Histogram and KDE for {col}')
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

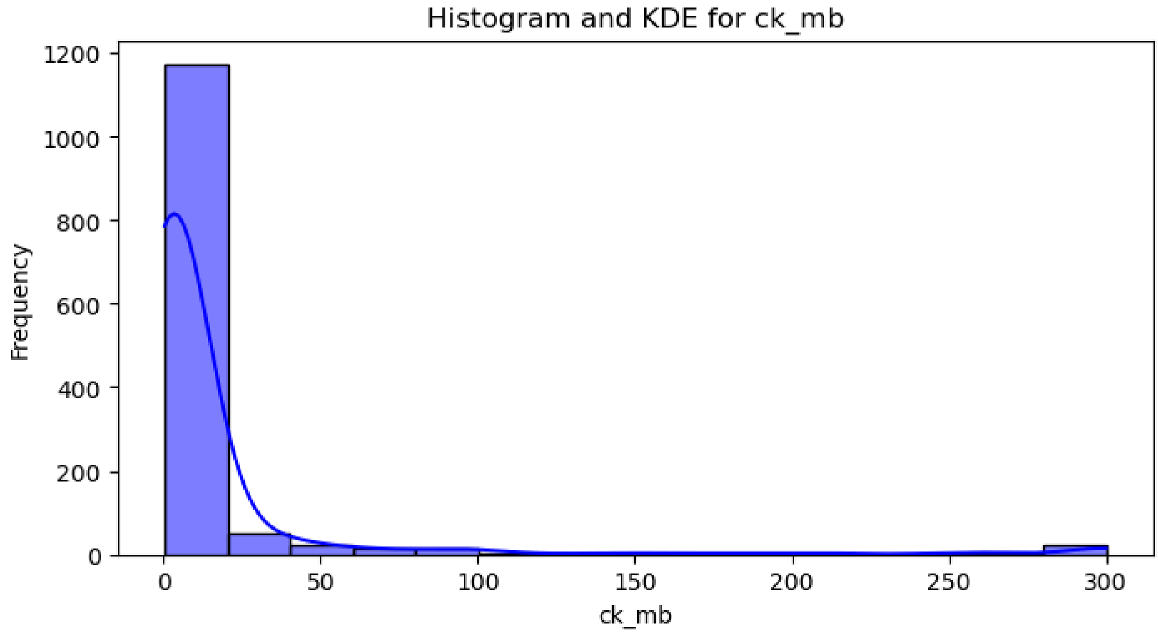


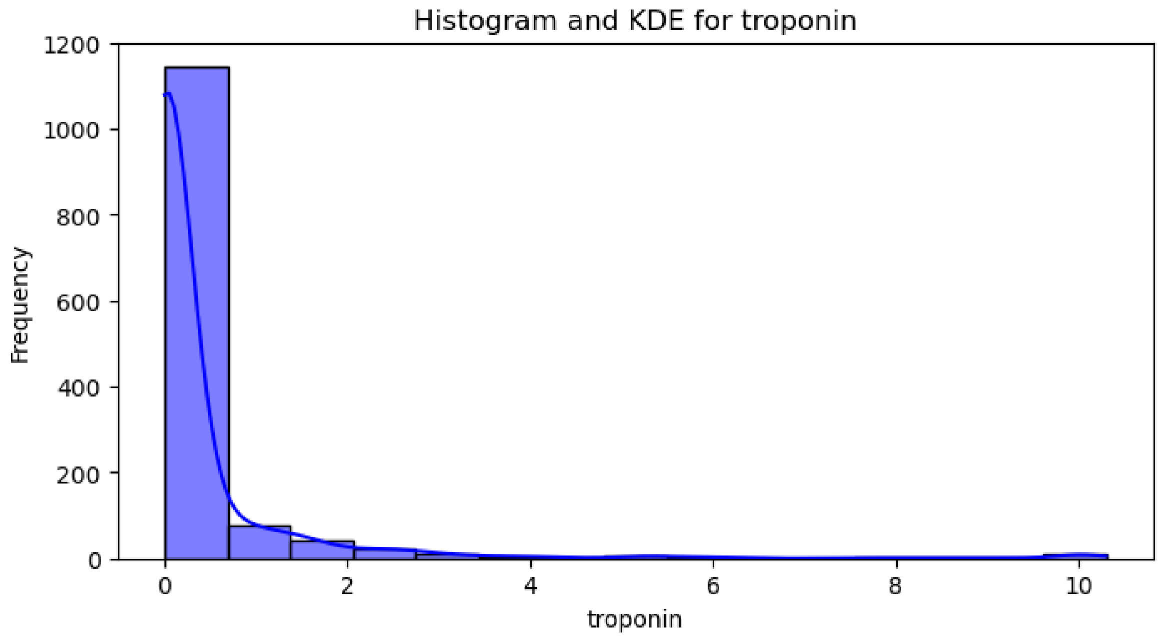

Histogram and KDE for heart_rate
Frequency
2000
1750
1500
1250
1000
750
500
250
0
0
200
400
600
800
1000
heart_rate


Histogram and KDE for systolic_blood_pressure
Frequency
300
250
200
150
100
50
0
50
75
100
125
150
175
200
225
systolic_blood_pressure


Histogram and KDE for diastolic_blood_pressure
Frequency
250
200
150
100
50
0
40
60
80
100
120
140
diastolic_blood_pressure

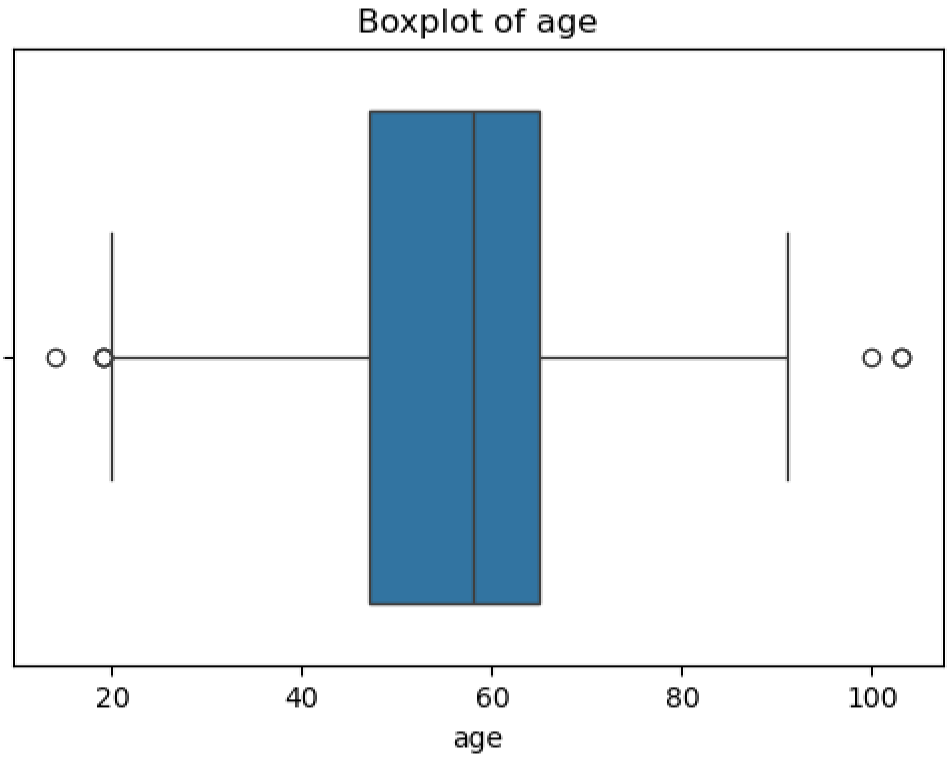

Histogram and KDE for blood_sugar
Frequency
0
100
200
300
400
100
200
300
400
500
blood_sugar

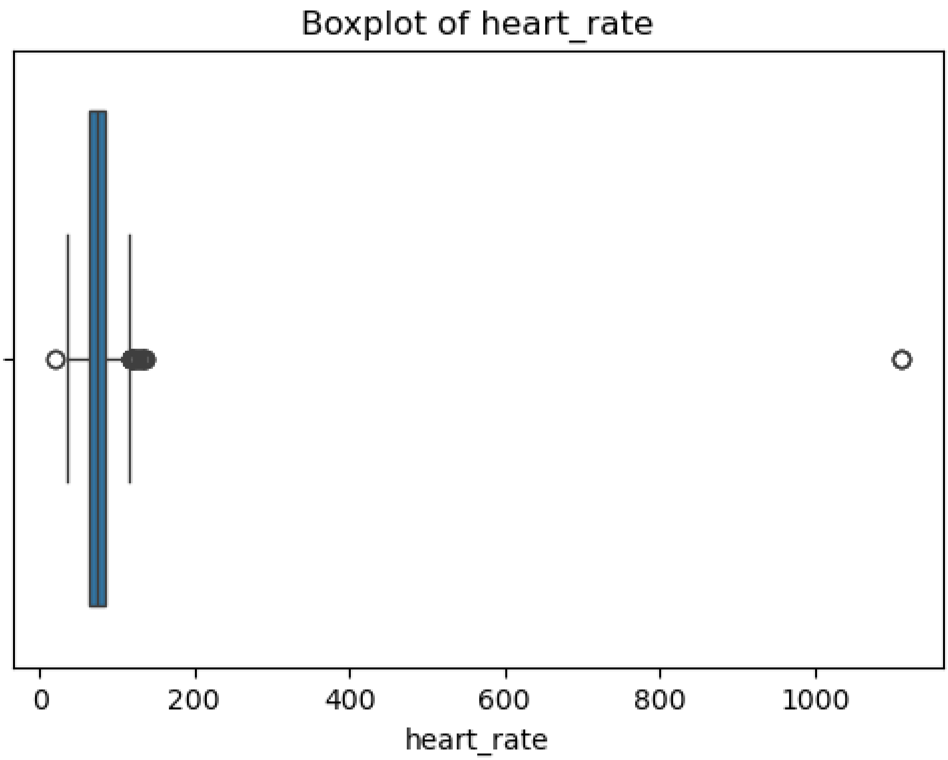

Histogram and KDE for ck_mb
Frequency
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
ck_mb

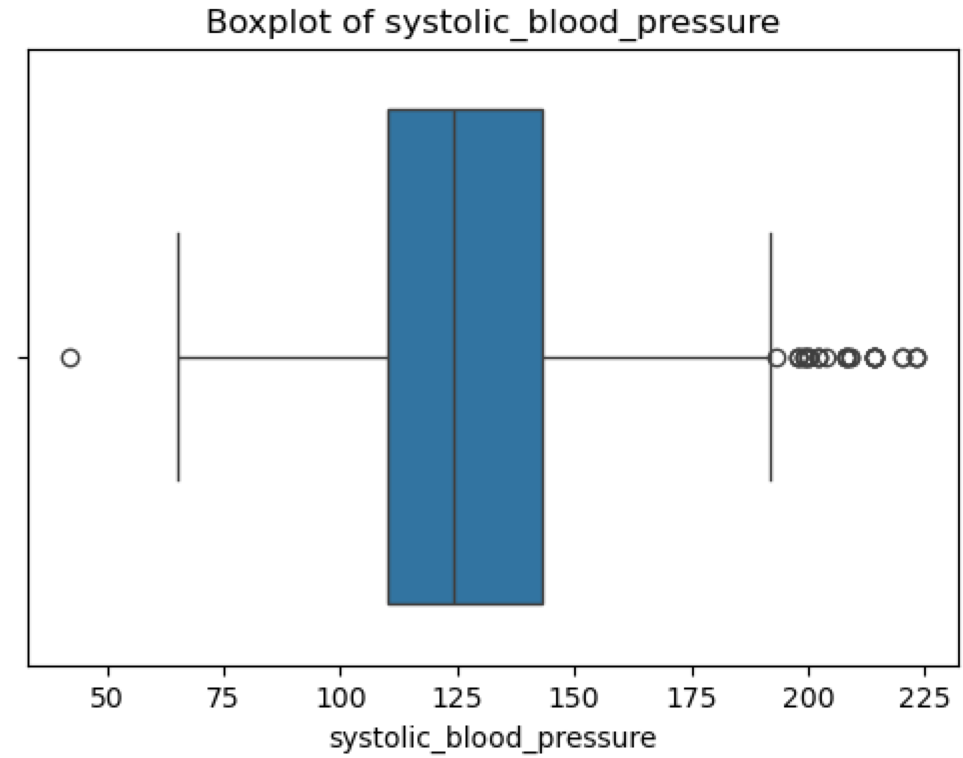

Histogram and KDE for troponin
Frequency
0
200
400
600
800
1000
1200
0
2
4
6
8
10
troponin

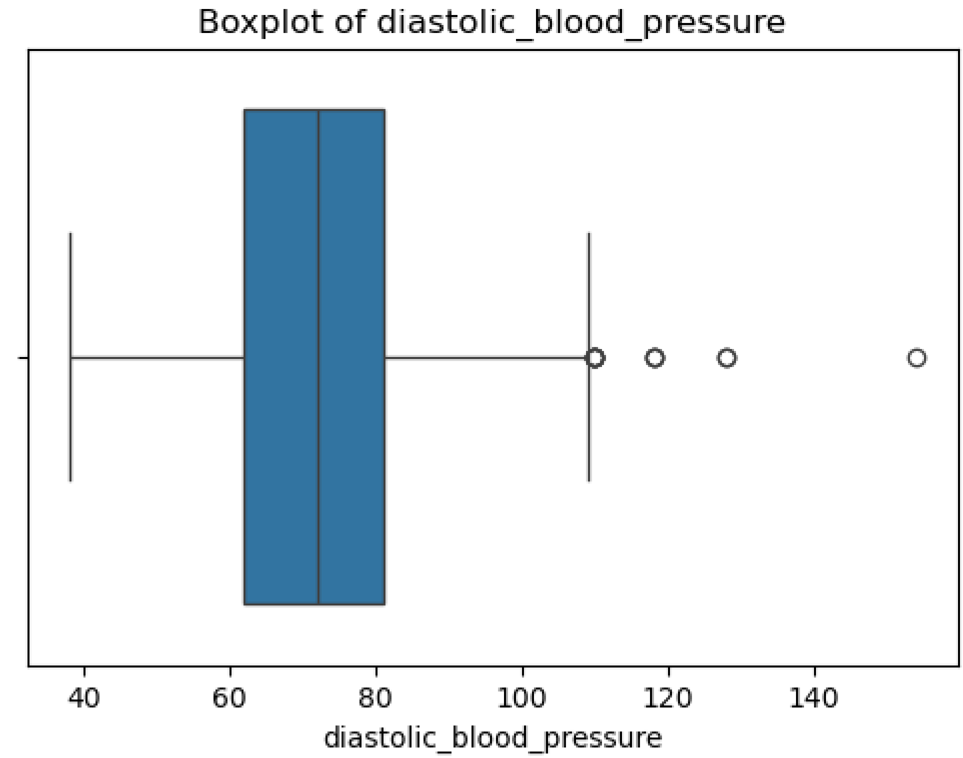
Boxplots to Detect Outliers

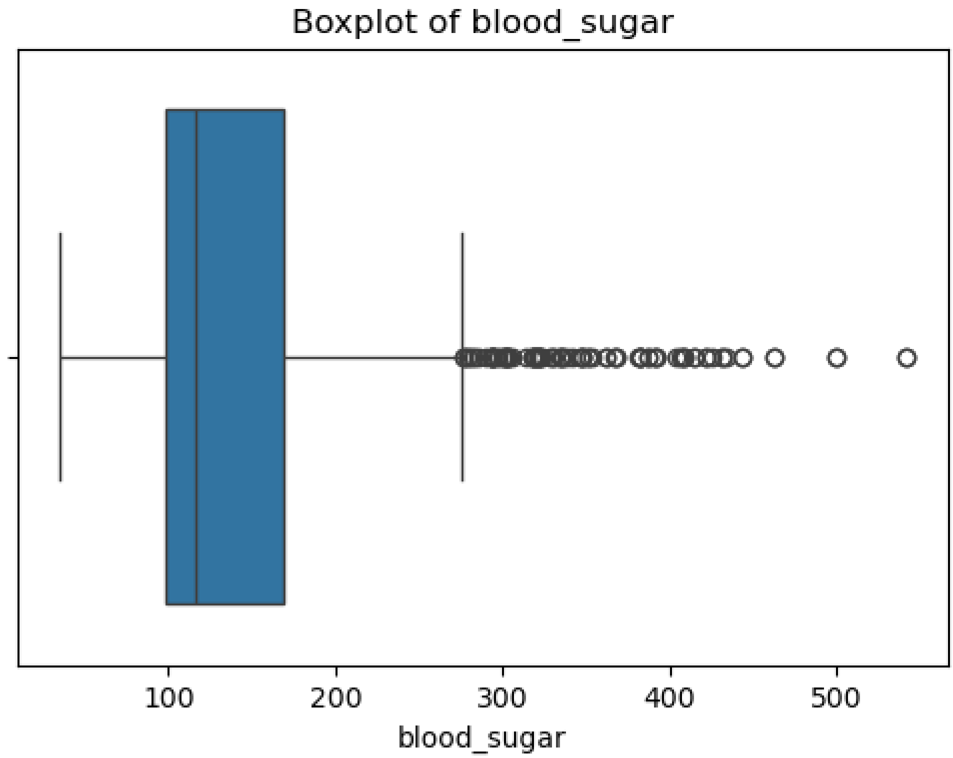
In [18]:
```
for col in numeric_col:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

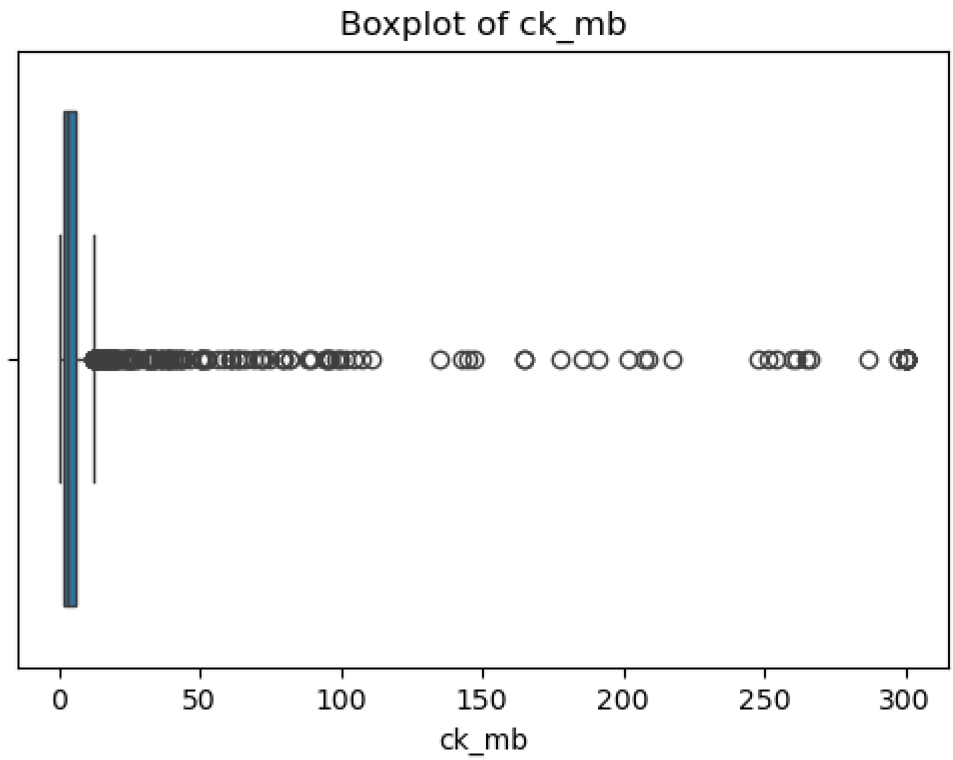


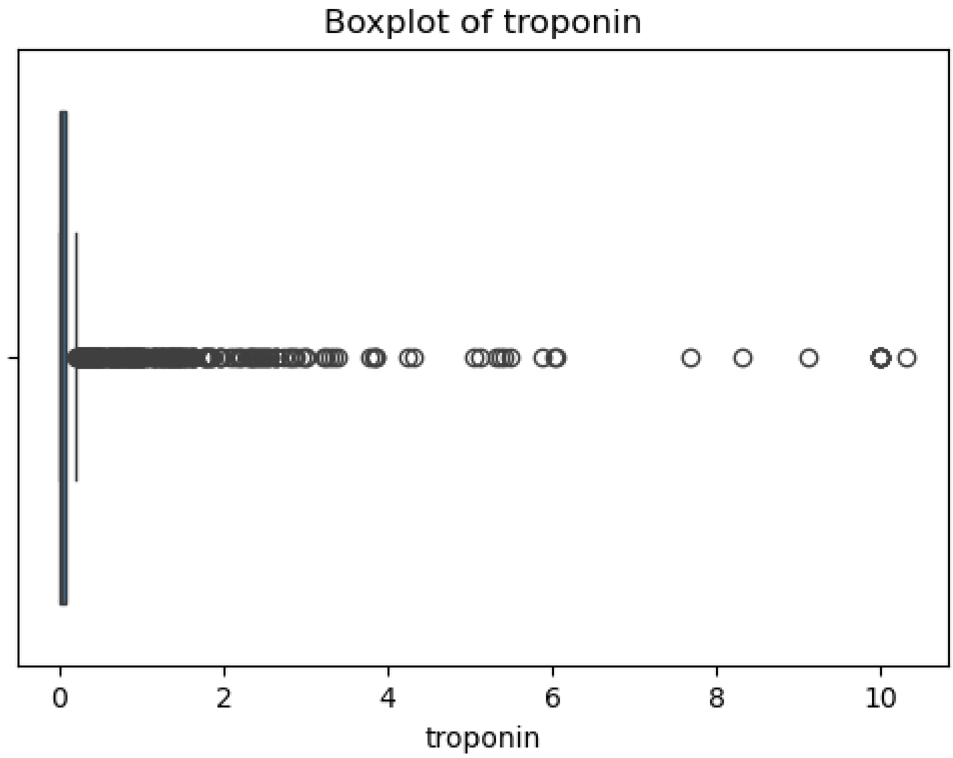



Boxplot of systolic_blood_pressure
50
75
100
125
150
175
200
225
systolic_blood_pressure


Boxplot of diastolic_blood_pressure
40
60
80
100
120
140
diastolic_blood_pressure

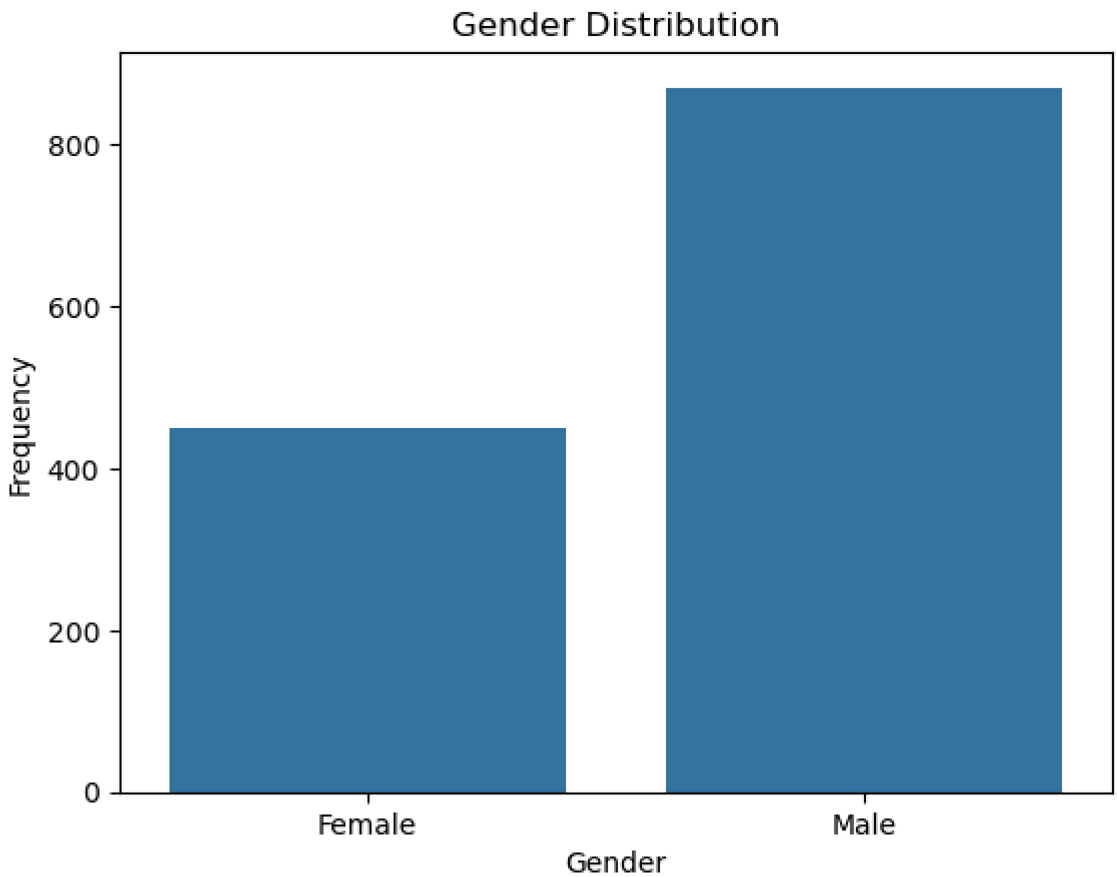

Boxplot of blood_sugar
100
200
300
400
500
blood_sugar


Boxplot of ck_mb
0
50
100
150
200
250
300
ck_mb



Gender Distribution

In [19]:
```
sns.countplot(x="gender", data=df)
plt.title('Gender Distribution')
plt.ylabel("Frequency")
plt.xlabel("Gender")
plt.xticks([0, 1], labels=["Female", "Male"])
plt.show()
```

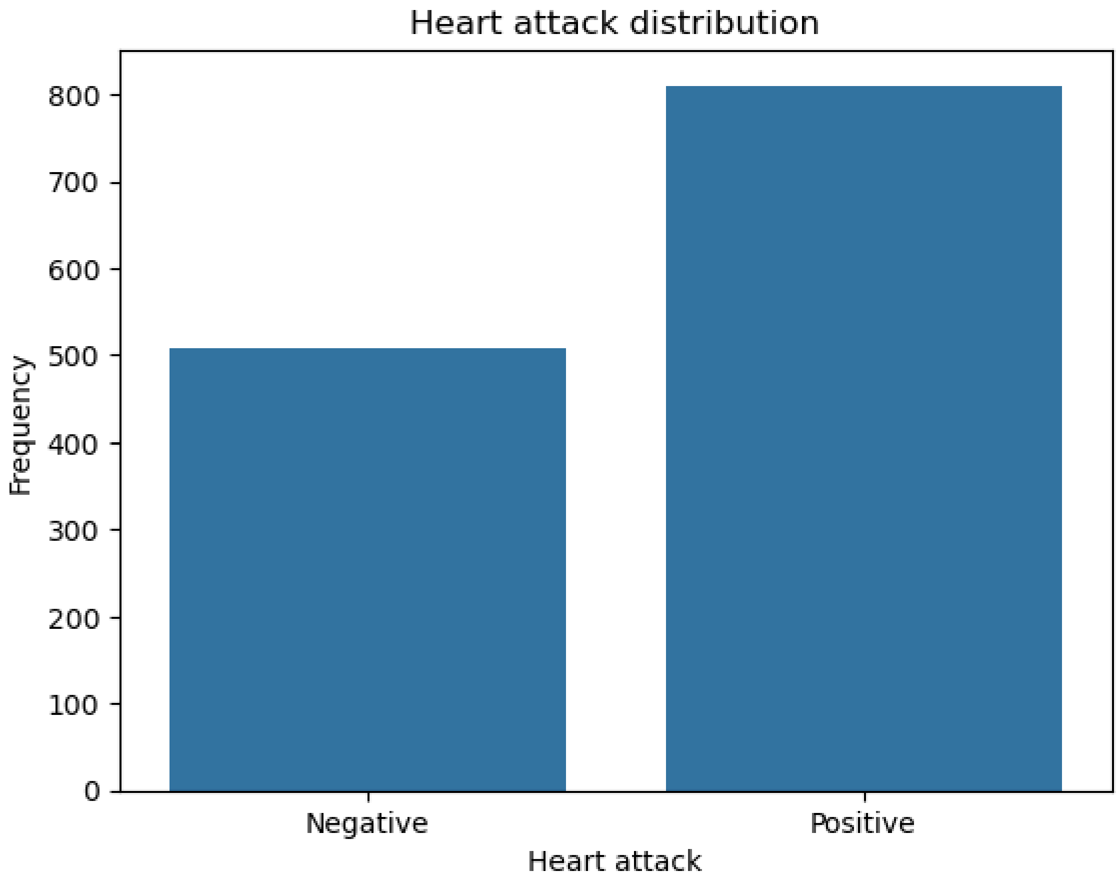


Distribution of the target variable

In [20]:
```
df["result"].value_counts()
```

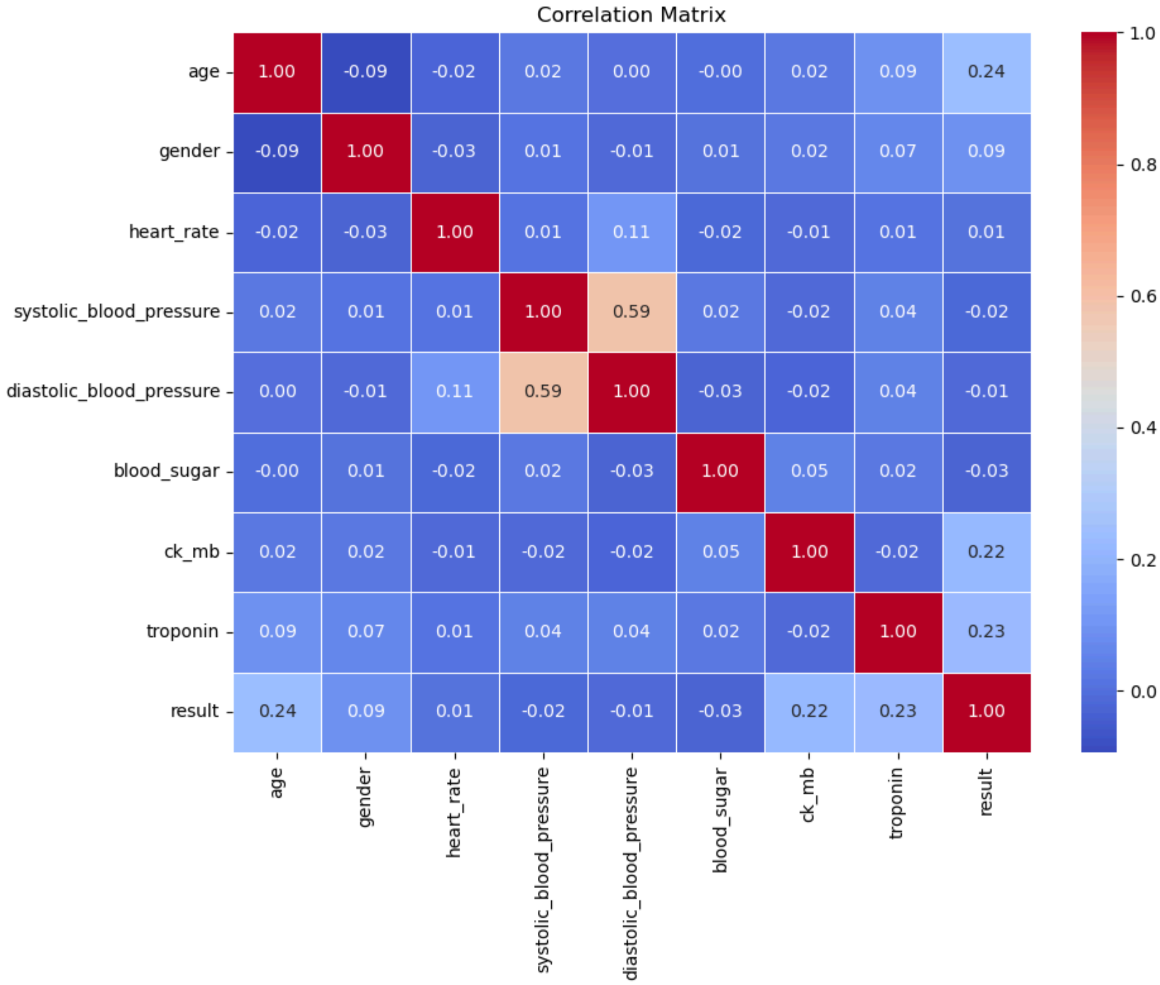
Out[20]:
```
result
1    810
0    509
Name: count, dtype: int64
```

In [21]:
```
## Class Distribution
sns.countplot(x="result", data=df)
plt.title('Heart attack distribution')
plt.ylabel("Frequency")
plt.xticks([0, 1], labels=["Negative", "Positive"])
plt.xlabel("Heart attack")
plt.show()
```



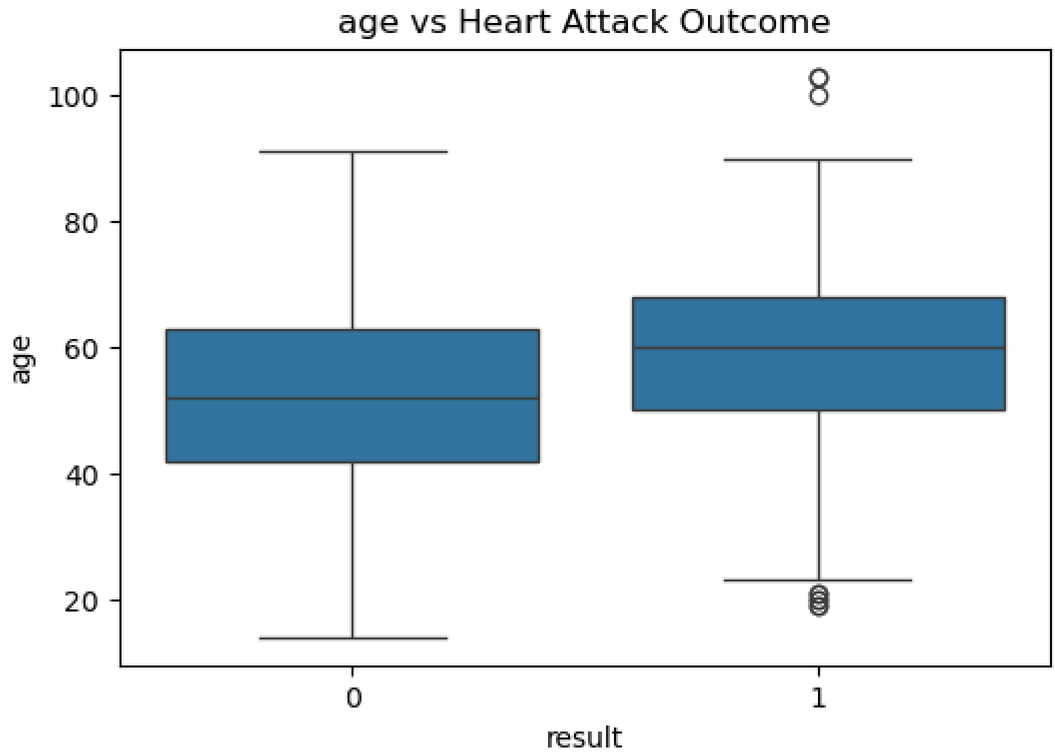
Correlation Matrix

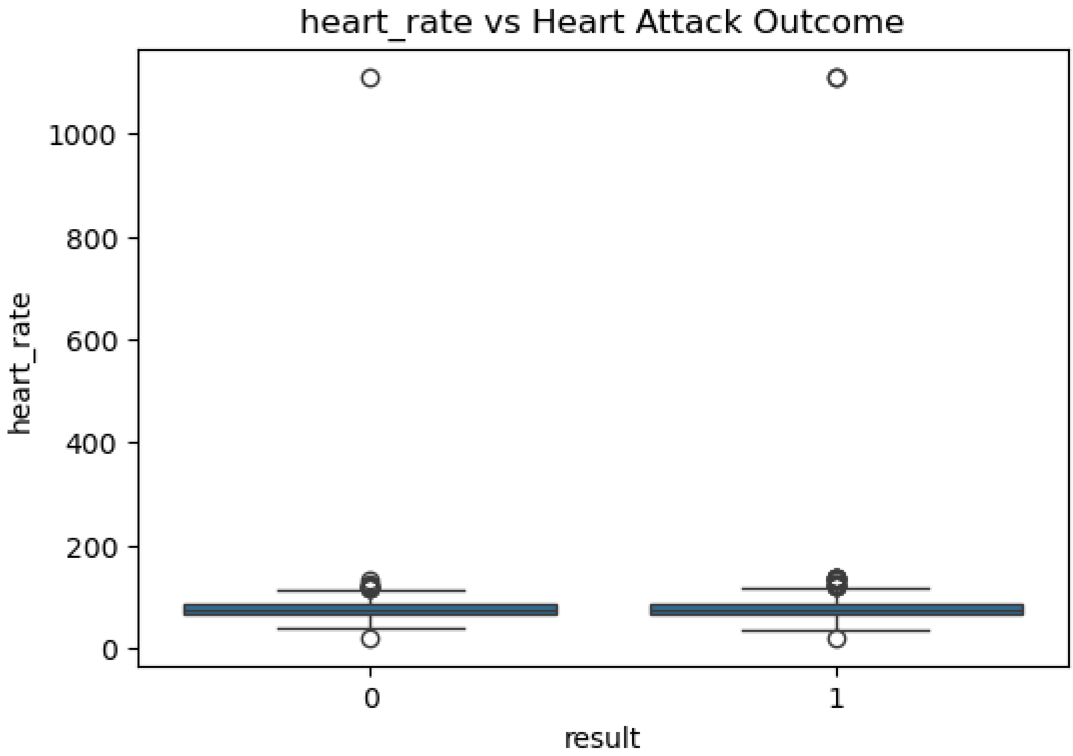
In [22]:
```
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Correlation Matrix")
plt.tight_layout()
plt.show()
```

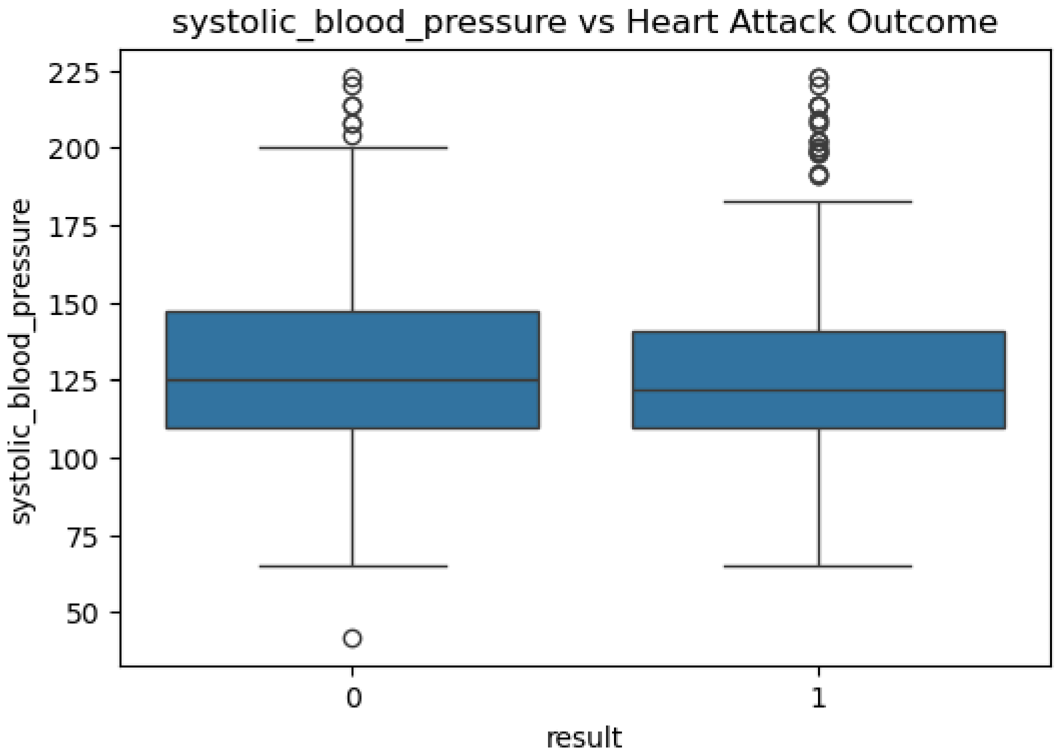


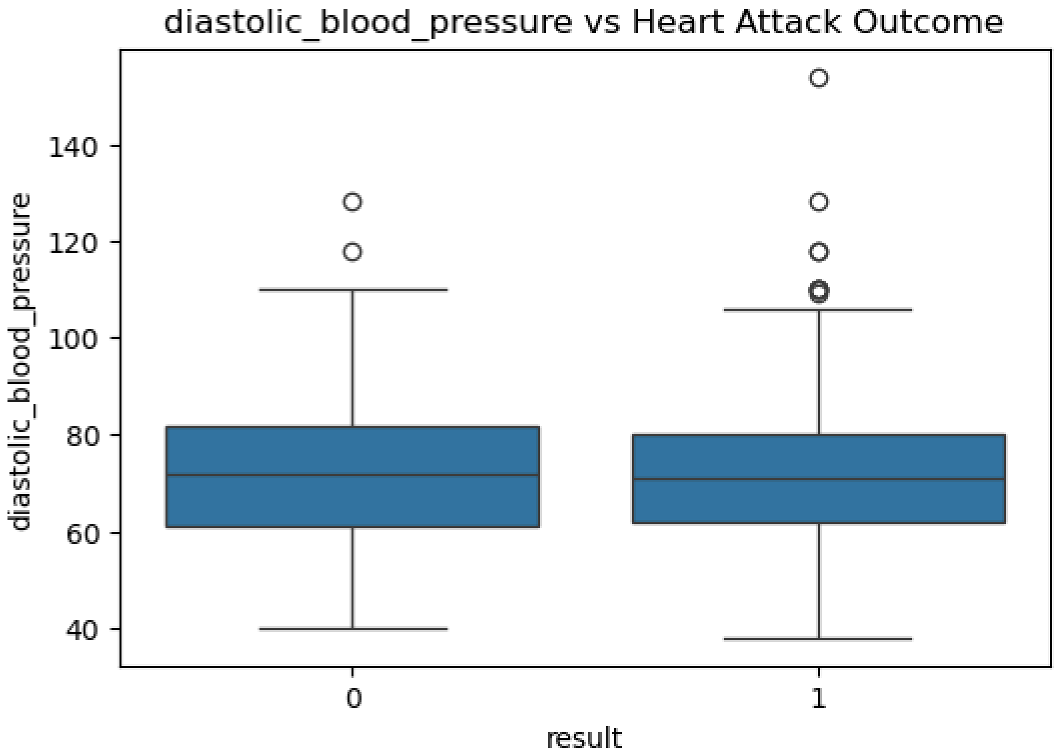
Group-wise Analysis (e.g., Risk Level vs Numeric Variables)

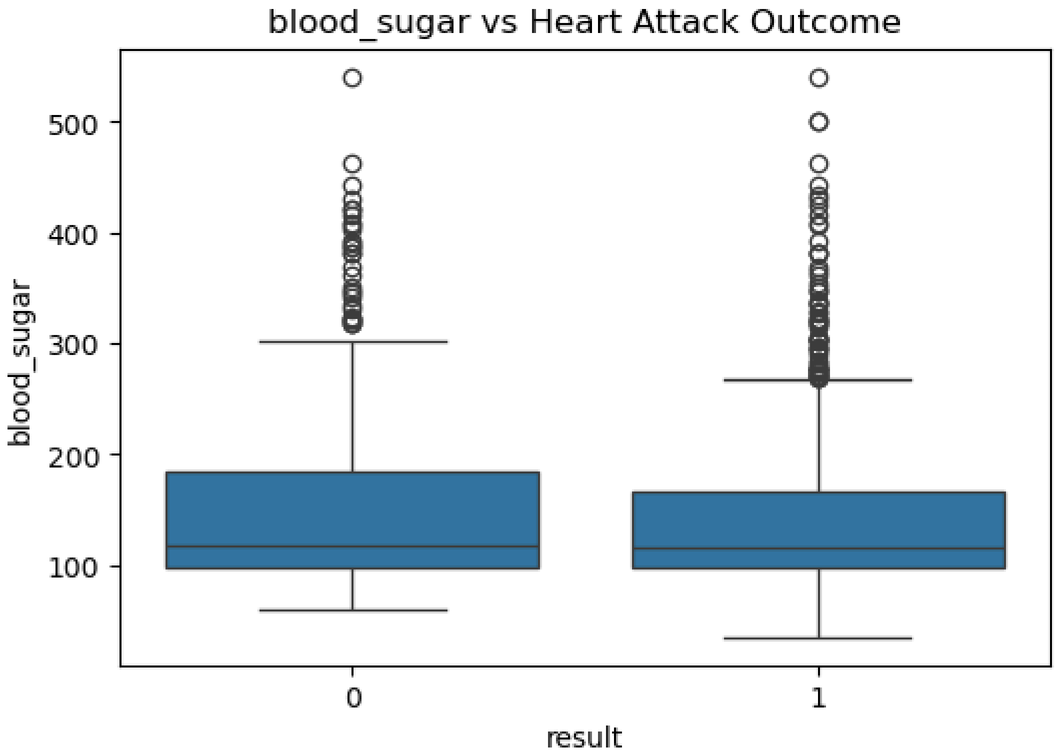
In [25]:
```
for col in numeric_col:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='result', y=col, data=df)
    plt.title(f'{col} vs Heart Attack Outcome')
    plt.show()
```

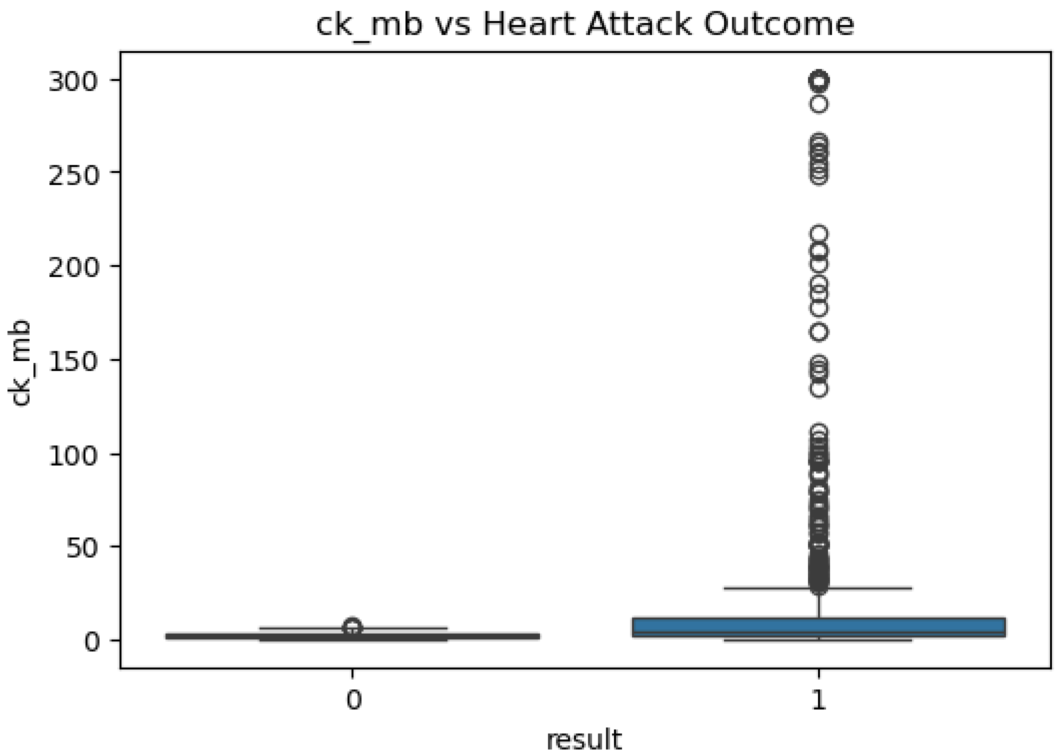

age vs Heart Attack Outcome
100
80
60
40
20
age
0
1
result

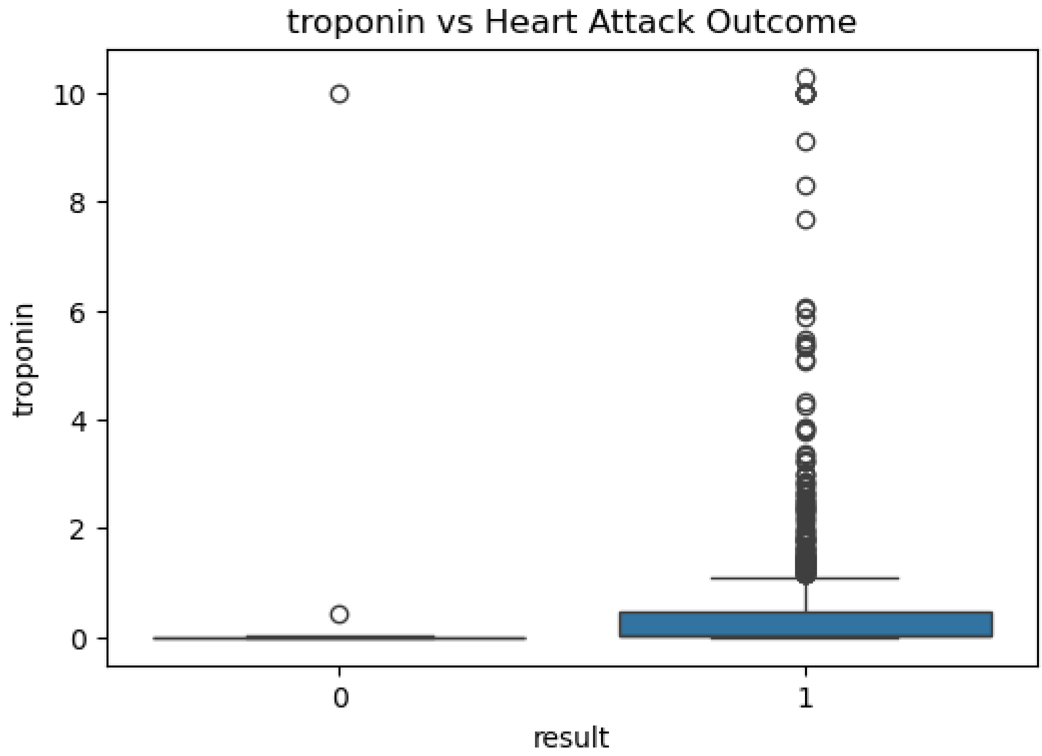

heart_rate vs Heart Attack Outcome
1000
800
600
400
200
0
heart_rate
0
1
result


systolic_blood_pressure vs Heart Attack Outcome
225
200
175
150
125
100
75
50
systolic_blood_pressure
0
1
result


diastolic_blood_pressure vs Heart Attack Outcome
140
120
100
80
60
40
diastolic_blood_pressure
0
1
result

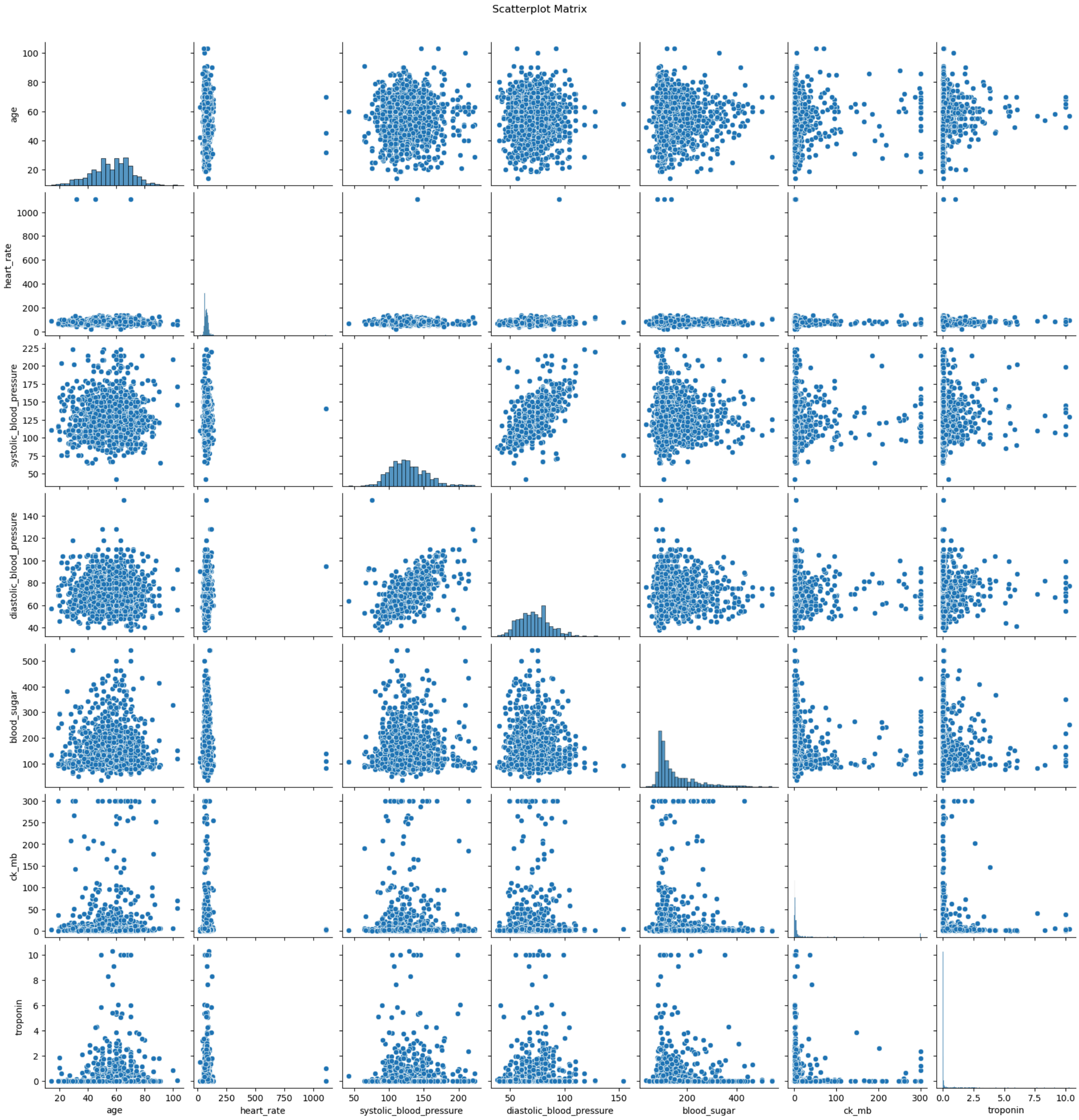

blood_sugar vs Heart Attack Outcome
500
400
300
200
100
blood_sugar
0
1
result


ck_mb vs Heart Attack Outcome
300
250
200
150
100
50
0
ck_mb
0
1
result



Bivariate Relationships (Scatterplots)

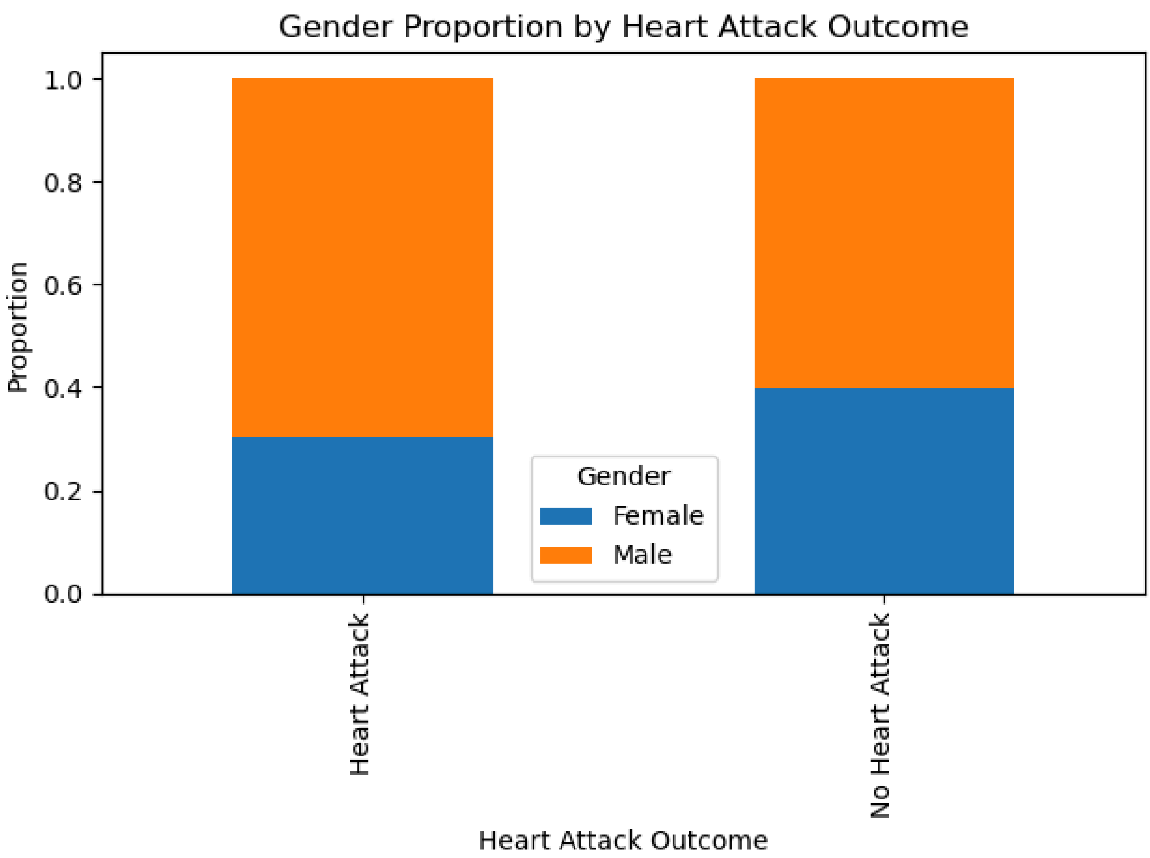
In [28]:
```
sns.pairplot(numeric_col)
plt.suptitle("Scatterplot Matrix", y=1.02)
plt.show()
```



Cross Tabulation and Stacked Bar Charts

In [44]:
```
## Load required modules
import matplotlib.pyplot as plt
import pandas as pd

## Create a copy of your DataFrame
df_plot = df.copy()

## Rename values for clarity
df_plot['result'] = df_plot['result'].replace({1 : 'Heart Attack', 0 : 'No Heart
df_plot['gender'] = df_plot['gender'].replace({0: 'Female', 1: 'Male'})

## Create normalized crosstab
crosstab = pd.crosstab(df_plot['result'], df_plot['gender'], normalize='index')

## Plot
crosstab.plot(kind='bar', stacked=True)

## Set labels and title
plt.title("Gender Proportion by Heart Attack Outcome")
plt.ylabel("Proportion")
plt.xlabel("Heart Attack Outcome")
```

```
plt.legend(title="Gender")
plt.tight_layout()
plt.show()
```



Chi-square Tests for Categorical Associations

In [34]:
```
## Load required module
from scipy.stats import chi2_contingency
import pandas as pd

## Create contingency table
table = pd.crosstab(df['gender'], df['result'])

## Perform Chi-square test
chi2, p, dof, ex = chi2_contingency(table)

## Print result
print(f"Chi-square test between Gender and Heart Attack Outcome: p-value = {p:.4
```

```
Chi-square test between Gender and Heart Attack Outcome: p-value = 0.0008
```

T-tests (Numeric vs. Categorical)

In [40]:
```
## Load required module
from scipy.stats import ttest_ind

## Get all numeric columns
numeric_col = df.drop(columns = ["result", "gender"])

## Split the dataset into two groups based on 'Result'
group1 = df[df['result'] == 1]
group2 = df[df['result'] == 0]
```

```
## Perform independent t-test for each numeric column
ttest_results = []
for col in numeric_col:
    t_stat, p_val = ttest_ind(group1[col], group2[col], equal_var=False)  # Welc
    ttest_results.append((col, p_val))

ttest_results
```

Out[40]:
```
[('age', 6.6419013877270196e-18),
 ('heart_rate', 0.7969260853088802),
 ('systolic_blood_pressure', 0.45574825157843557),
 ('diastolic_blood_pressure', 0.7279941855792156),
 ('blood_sugar', 0.23842221940845196),
 ('ck_mb', 4.140673523845025e-23),
 ('troponin', 7.200656876108758e-24)]
```

Defining the X and y features

In [46]:
```
X = df.drop(columns = ["result"])
y = df["result"]
```

Splitting data into training and test set

In [47]:
```
## Load the required module
from sklearn.model_selection import train_test_split

## Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, random_s
```

In [48]:
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (1055, 8)
X_test shape: (264, 8)
y_train shape: (1055,)
y_test shape: (264,)
```

Feature Scaling/Standardization

In [49]:
```
## Load the required module
from sklearn.preprocessing import MinMaxScaler

## Initialize the scaler
scaler = MinMaxScaler()

## Fit the scaler
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

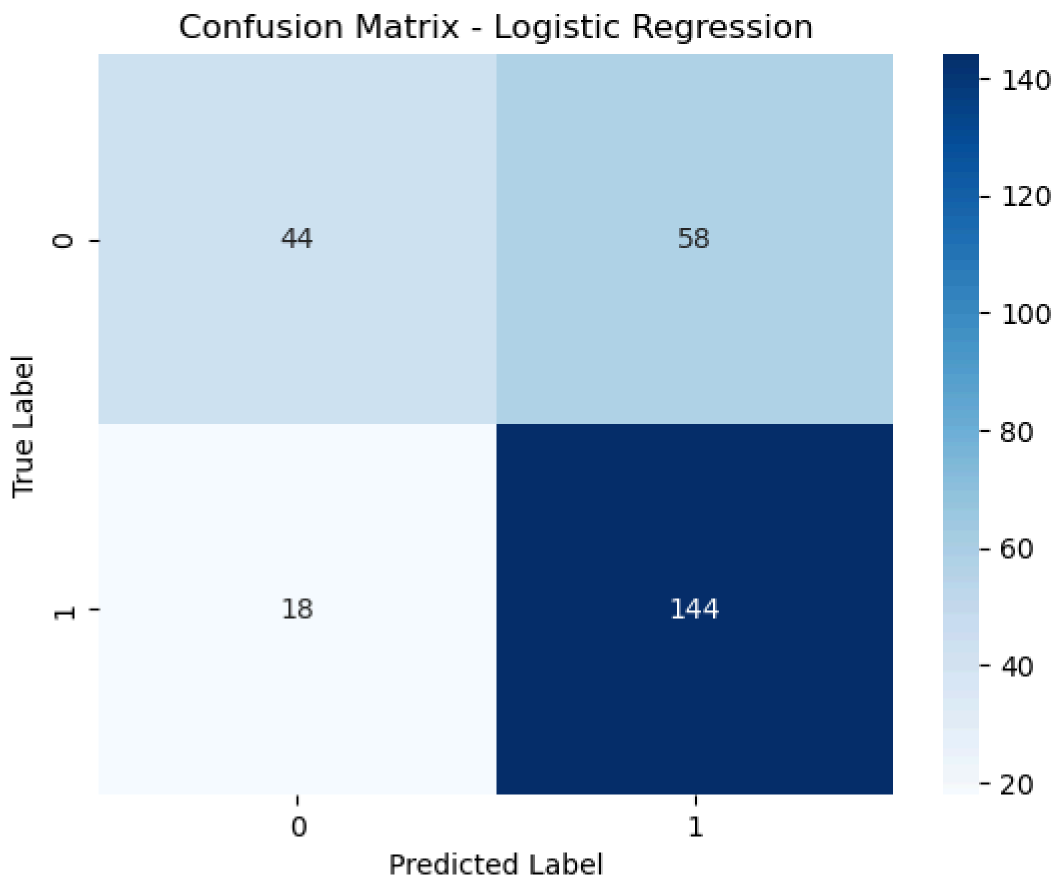
Model Training

1. Logistic Regression

In [50]:
```
## Load required modules
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, f1_score, p
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

## Initialize the model with better configuration
log = LogisticRegression(random_state=42, solver='lbfgs', max_iter=1000)

## Fit the model
log.fit(X_train, y_train)

## Make predictions
log_pred = log.predict(X_test)
log_score = accuracy_score(y_test, log_pred)

## Evaluate performance
print("Classification Report:\n", classification_report(y_test, log_pred))
print("F1 Score:", f1_score(y_test, log_pred, average='macro'))
print("Precision:", precision_score(y_test, log_pred, average='macro'))
print("Recall:", recall_score(y_test, log_pred, average='macro'))
print("Accuracy:", accuracy_score(y_test, log_pred))

## Confusion Matrix
cm = confusion_matrix(y_test, log_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.43      0.54       102
           1       0.71      0.89      0.79       162

    accuracy                           0.71       264
   macro avg       0.71      0.66      0.66       264
weighted avg       0.71      0.71      0.69       264

F1 Score: 0.6638970785312248
Precision: 0.7112743532417758
Recall: 0.6601307189542484
Accuracy: 0.7121212121212122
```



Decision Trees

In [52]:
```
## Load required modules
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, f1_score, precision_score, re
import matplotlib.pyplot as plt

## Initialize model
dt = DecisionTreeClassifier(random_state=42)

## Define a more efficient parameter grid
parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, 20, None],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'splitter': ['best']
}

## Grid Search
grid_search_dt = GridSearchCV(dt, parameters, cv=5, n_jobs=-1, verbose=1, scorin
grid_search_dt.fit(X_train, y_train)

## Best model prediction
dt_pred = grid_search_dt.predict(X_test)
dt_score = accuracy_score(y_test, dt_pred)

## Evaluation
print("Best Parameters Found:", grid_search_dt.best_params_)
print("Classification Report:\n", classification_report(y_test, dt_pred))
print("F1 Score:", f1_score(y_test, dt_pred, average='macro'))
```

```
print("Precision:", precision_score(y_test, dt_pred, average='macro'))
print("Recall:", recall_score(y_test, dt_pred, average='macro'))
print("Accuracy:", accuracy_score(y_test, dt_pred))
```

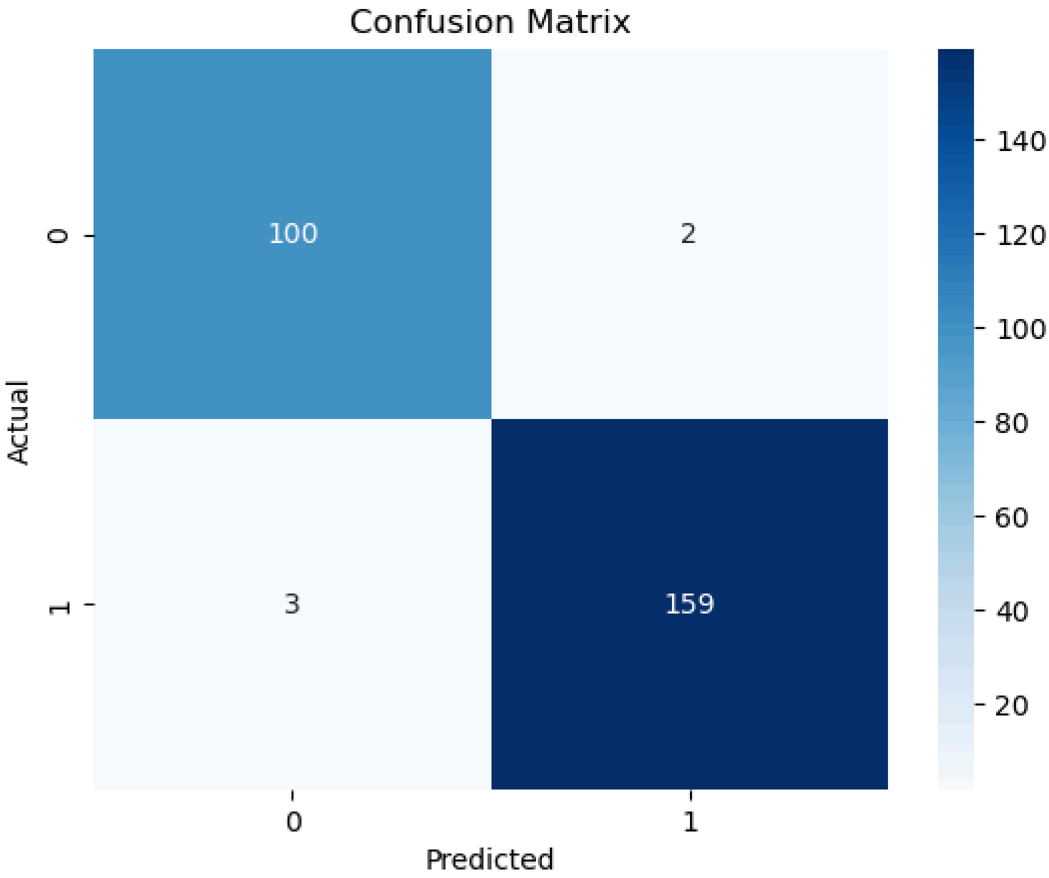
```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best Parameters Found: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf':
1, 'min_samples_split': 10, 'splitter': 'best'}
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       102
           1       0.98      0.99      0.98       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

F1 Score: 0.9799924213717317
Precision: 0.9808965559132601
Recall: 0.979121278140886
Accuracy: 0.9810606060606061
```

2. Random Forest

In [53]:
```
## Import libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import seaborn as sns
import matplotlib.pyplot as plt

## Define your model
rf = RandomForestClassifier(class_weight='balanced', random_state=42)

## Define a reduced hyperparameter grid
param_dist = {
    'n_estimators': [100, 300, 500, 800],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

## Cross-validation strategy
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=42)

## Use RandomizedSearchCV for efficiency
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=20,  # Try only 20 random combinations
    cv=cv,
    scoring='f1_macro',  # Better than accuracy for imbalanced data
    n_jobs=-1,
    verbose=2,
    random_state=42
)

## Fit the model
```

```
best_model = random_search.fit(X_train, y_train)

## Make predictions
rf_pred = best_model.predict(X_test)
rf_score = accuracy_score(y_test, rf_pred)

## Evaluate performance
print("Best Parameters:", random_search.best_params_)
print("Best Cross-Validated F1 Score:", random_search.best_score_)
print("\n Classification Report:\n", classification_report(y_test, rf_pred))
print("Accuracy Score:", accuracy_score(y_test, rf_pred))
print("F1 Score:", f1_score(y_test, rf_pred, average='macro'))
print("Precision:", precision_score(y_test, rf_pred, average='macro'))
print("Recall:", recall_score(y_test, rf_pred, average='macro'))

## Plot the confusion matrix
cm = confusion_matrix(y_test, rf_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Parameters: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_lea
f': 2, 'max_features': 'sqrt', 'max_depth': 20, 'bootstrap': True}
Best Cross-Validated F1 Score: 0.9870532516874215

 Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.98      0.98       102
           1       0.99      0.98      0.98       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

Accuracy Score: 0.9810606060606061
F1 Score: 0.9800649399682851
Precision: 0.9792257130796599
Recall: 0.9809368191721133
```

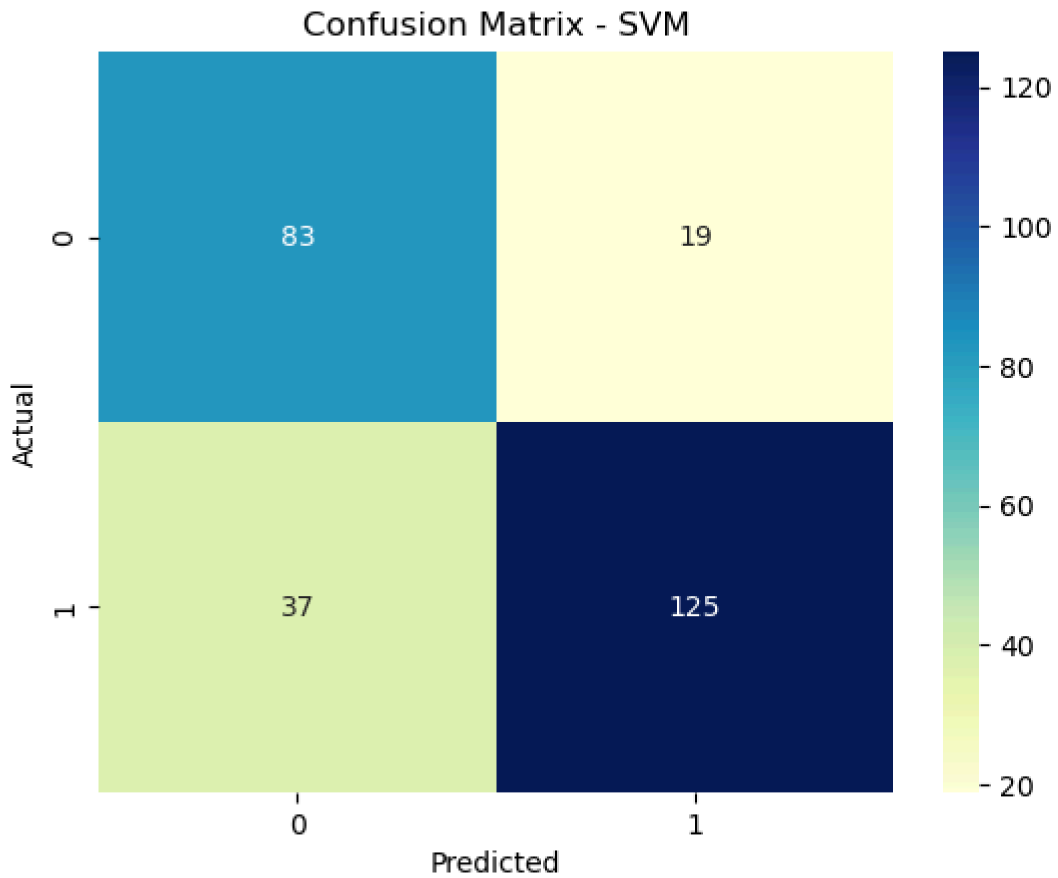


Support Vector Machines

In [54]:
```
## Import required libraries
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import (classification_report, confusion_matrix,
                             f1_score, precision_score, recall_score, accuracy_s

## Define CV strategy
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=42)

## Common SVM model with probability enabled
svm = SVC(probability=True, random_state=42)

## Define separate hyperparameter grids
# RBF Kernel
param_grid_rbf = {
    'kernel': ['rbf'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'shrinking': [True, False]
}

# Poly Kernel
param_grid_poly = {
    'kernel': ['poly'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3],
    'coef0': [0.0, 0.5],
    'shrinking': [True, False]
}
```

```
# Sigmoid Kernel
param_grid_sigmoid = {
    'kernel': ['sigmoid'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto'],
    'coef0': [0.0, 0.5],
    'shrinking': [True, False]
}

## Combine the grids
param_grid_combined = [
    param_grid_rbf,
    param_grid_poly,
    param_grid_sigmoid
]

## Use RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=svm,
    param_distributions=param_grid_combined,
    n_iter=20,  # Try 20 random combinations
    scoring='f1_macro',
    n_jobs=-1,
    cv=cv,
    verbose=2,
    random_state=42
)

## Train the model
best_model = random_search.fit(X_train, y_train)

## Make predictions
svm_pred = best_model.predict(X_test)
svm_score = accuracy_score(y_test, svm_pred)

## Evaluate performance
print("Best Parameters:", best_model.best_params_)
print("Accuracy:", accuracy_score(y_test, svm_pred))
print("F1 Score (macro):", f1_score(y_test, svm_pred, average='macro'))
print("Precision (macro):", precision_score(y_test, svm_pred, average='macro'))
print("Recall (macro):", recall_score(y_test, svm_pred, average='macro'))
print("\n Classification Report:\n", classification_report(y_test, svm_pred))

## onfusion matrix visualization
cm = confusion_matrix(y_test, svm_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu')
plt.title("Confusion Matrix - SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best Parameters: {'shrinking': False, 'kernel': 'poly', 'gamma': 'scale', 'degre
e': 3, 'coef0': 0.0, 'C': 10}
Accuracy: 0.7878787878787878
F1 Score (macro): 0.7823706059000177
Precision (macro): 0.779861111111111
Recall (macro): 0.7926652142338417

 Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.81      0.75       102
           1       0.87      0.77      0.82       162

    accuracy                           0.79       264
   macro avg       0.78      0.79      0.78       264
weighted avg       0.80      0.79      0.79       264
```



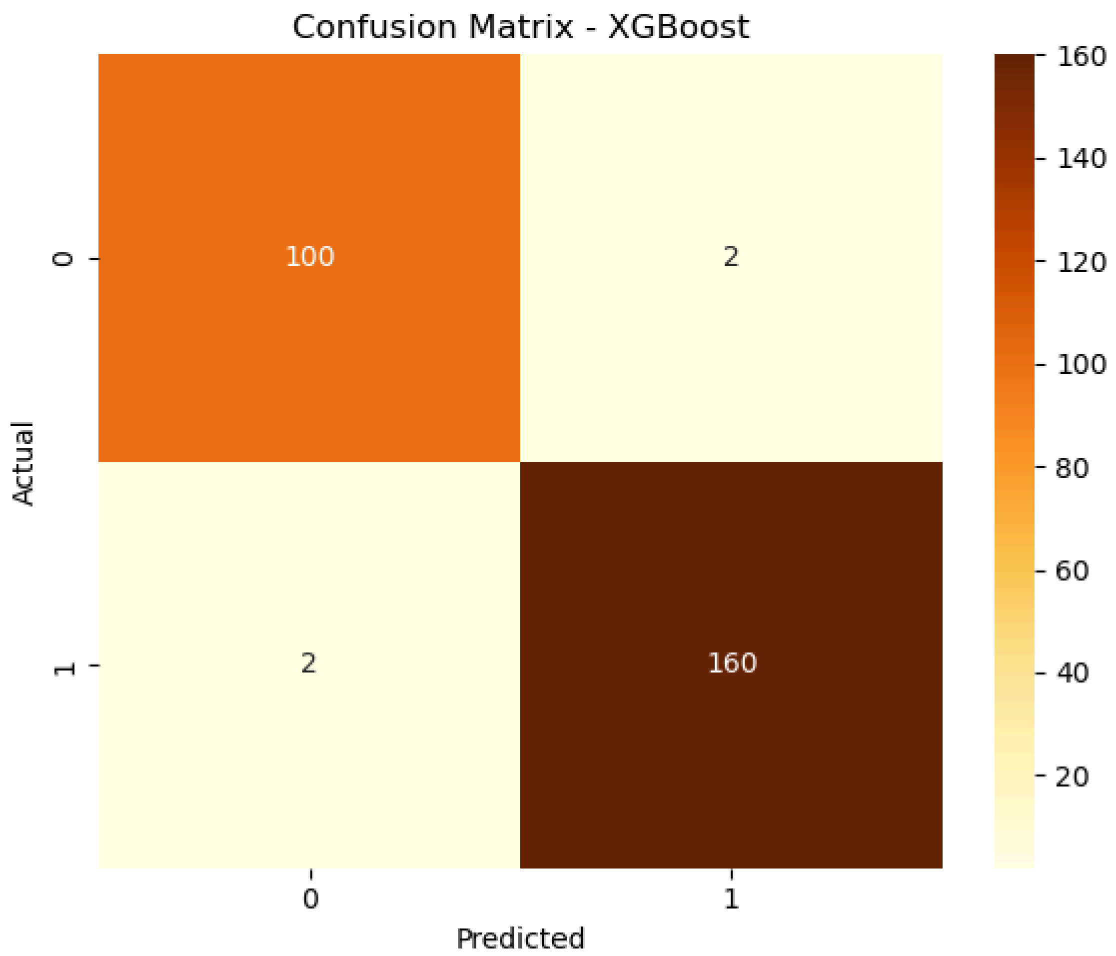
XGBoost Classifier

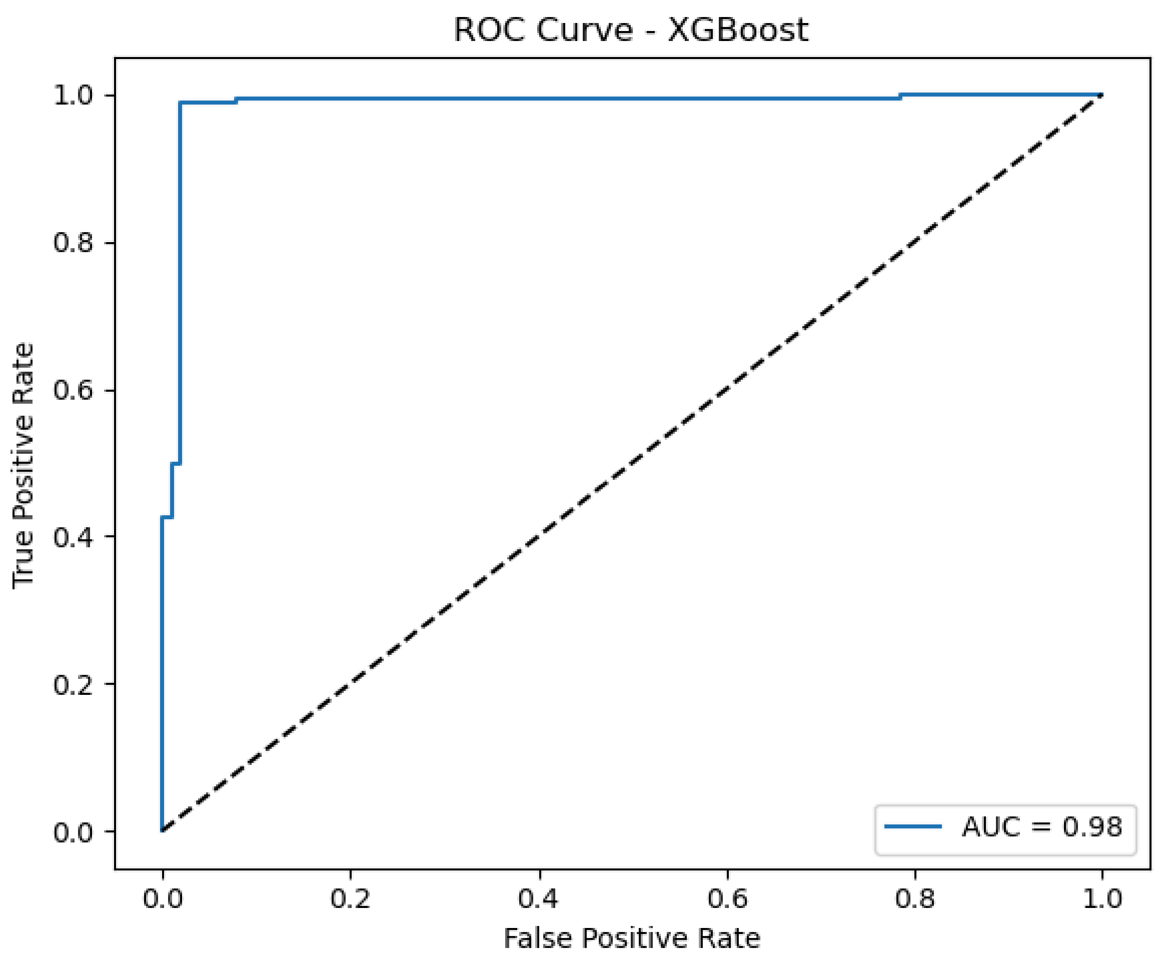
In [55]:
```
## Load required modules
from xgboost import XGBClassifier, plot_importance
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import (classification_report, confusion_matrix,
                             f1_score, precision_score, recall_score, accuracy_s
import matplotlib.pyplot as plt
import seaborn as sns

## Define the parameter grid
param_grid = {
    'n_estimators': [100, 300, 500],
    'max_depth': [3, 5, 7],
```

```
    'learning_rate': [0.01, 0.1, 0.3],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 1],
    'min_child_weight': [1, 5],
    'scale_pos_weight': [1]
}

## Define cross-validation strategy
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=42)

## Initialize the model
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state

## Grid Search
grid_search = GridSearchCV(estimator=xgb,
                           param_grid=param_grid,
                           scoring='f1_macro',
                           cv=cv,
                           n_jobs=-1,
                           verbose=1)

## Fit the model
grid_result = grid_search.fit(X_train, y_train)

## Best estimator
best_xgb = grid_result.best_estimator_

## Predict class labels
xgb_pred = best_xgb.predict(X_test)

## Predict probabilities
xgb_prob = best_xgb.predict_proba(X_test)
xgb_score = accuracy_score(y_test, xgb_pred)

## Get probabilities of the positive class (label = 1)
xgb_prob_positive = xgb_prob[:, 1]

## Evaluate performance
print(" Best Hyperparameters:", grid_result.best_params_)
print("\n Classification Report:\n", classification_report(y_test, xgb_pred))
print(" F1 Score (macro):", f1_score(y_test, xgb_pred, average='macro'))
print(" Precision (macro):", precision_score(y_test, xgb_pred, average='macro'))
print(" Recall (macro):", recall_score(y_test, xgb_pred, average='macro'))
print(" Accuracy:", accuracy_score(y_test, xgb_pred))

## Print some of the predicted probabilities
print("\nSample predicted probabilities:\n", xgb_prob[:10])

## Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, xgb_pred), annot=True, fmt='d', cmap='YlOrB
plt.title("Confusion Matrix - XGBoost")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

## ROC Curve
if len(set(y_test)) == 2:
```

```
    fpr, tpr, thresholds = roc_curve(y_test, xgb_prob_positive)
    auc_score = roc_auc_score(y_test, xgb_prob_positive)

    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, label=f"AUC = {auc_score:.2f}")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve - XGBoost")
    plt.legend(loc="lower right")
    plt.tight_layout()
    plt.show()
```

```
Fitting 10 folds for each of 432 candidates, totalling 4320 fits
 Best Hyperparameters: {'colsample_bytree': 0.8, 'gamma': 1, 'learning_rate': 0.
3, 'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 100, 'scale_pos_weigh
t': 1, 'subsample': 0.8}

 Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       102
           1       0.99      0.99      0.99       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

 F1 Score (macro): 0.9840232389251997
 Precision (macro): 0.9840232389251997
 Recall (macro): 0.9840232389251997
 Accuracy: 0.9848484848484849

Sample predicted probabilities:
 [[0.00240093 0.99759907]
 [0.99061626 0.00938373]
 [0.00845742 0.9915426 ]
 [0.99079925 0.00920074]
 [0.9816779  0.01832212]
 [0.01100898 0.988991  ]
 [0.01200908 0.9879909 ]
 [0.0034337  0.9965663 ]
 [0.00201058 0.9979894 ]
 [0.00792599 0.992074  ]]
```


Confusion Matrix - XGBoost
Actual
0
1
100
2
2
160
Predicted
0
1
160
140
120
100
80
60
40
20

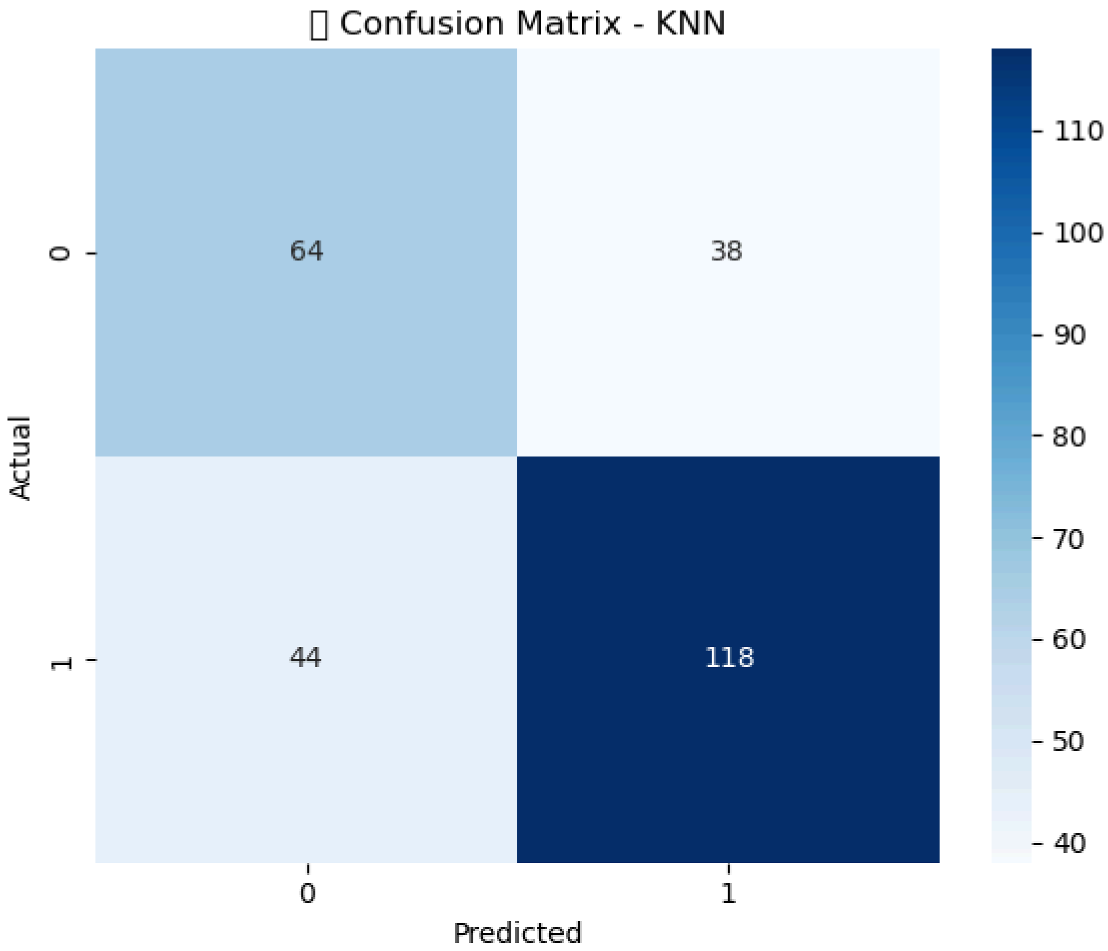

ROC Curve - XGBoost
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.98

K Nearest Neighbors

In [56]:
```
## Load the required libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    classification_report, confusion_matrix,
    accuracy_score, f1_score, precision_score, recall_score
)

## Define the pipeline
pipeline = Pipeline([
    ('knn', KNeighborsClassifier())
])

## Define hyperparameters to tune
param_grid = {
    'knn__n_neighbors': range(15, 25),
    'knn__weights': ['uniform', 'distance'],
    'knn__metric': ['euclidean', 'manhattan']
}

## Define cross-validation strategy
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)

## Setup Grid Search
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    n_jobs=-1,
    cv=cv,
    scoring='f1_macro',
    error_score=0,
    verbose=1
)

## Fit the model
best_model = grid_search.fit(X_train, y_train)

## Make predictions
knn_pred = best_model.predict(X_test)
knn_score = accuracy_score(y_test, knn_pred)

## Display best hyperparameters
print("Best Hyperparameters:\n", grid_search.best_params_)
print("Best Cross-Validated F1 Macro Score:\n", grid_search.best_score_)

## Classification metrics
print("\n📊 Classification Report:\n", classification_report(y_test, knn_pred))
print("Accuracy Score:", accuracy_score(y_test, knn_pred))
print("F1 Score (Macro):", f1_score(y_test, knn_pred, average='macro'))
print("Precision (Macro):", precision_score(y_test, knn_pred, average='macro'))
print("Recall (Macro):", recall_score(y_test, knn_pred, average='macro'))

## Confusion matrix plot
cm = confusion_matrix(y_test, knn_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
plt.title("🔍 Confusion Matrix - KNN")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
Fitting 30 folds for each of 40 candidates, totalling 1200 fits
Best Hyperparameters:
 {'knn__metric': 'manhattan', 'knn__n_neighbors': 24, 'knn__weights': 'distance'}
Best Cross-Validated F1 Macro Score:
 0.6705433917066951

📊 Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.63      0.61       102
           1       0.76      0.73      0.74       162

    accuracy                           0.69       264
   macro avg       0.67      0.68      0.68       264
weighted avg       0.69      0.69      0.69       264

Accuracy Score: 0.6893939393939394
F1 Score (Macro): 0.6758310871518418
Precision (Macro): 0.6745014245014245
Recall (Macro): 0.677923021060276
```

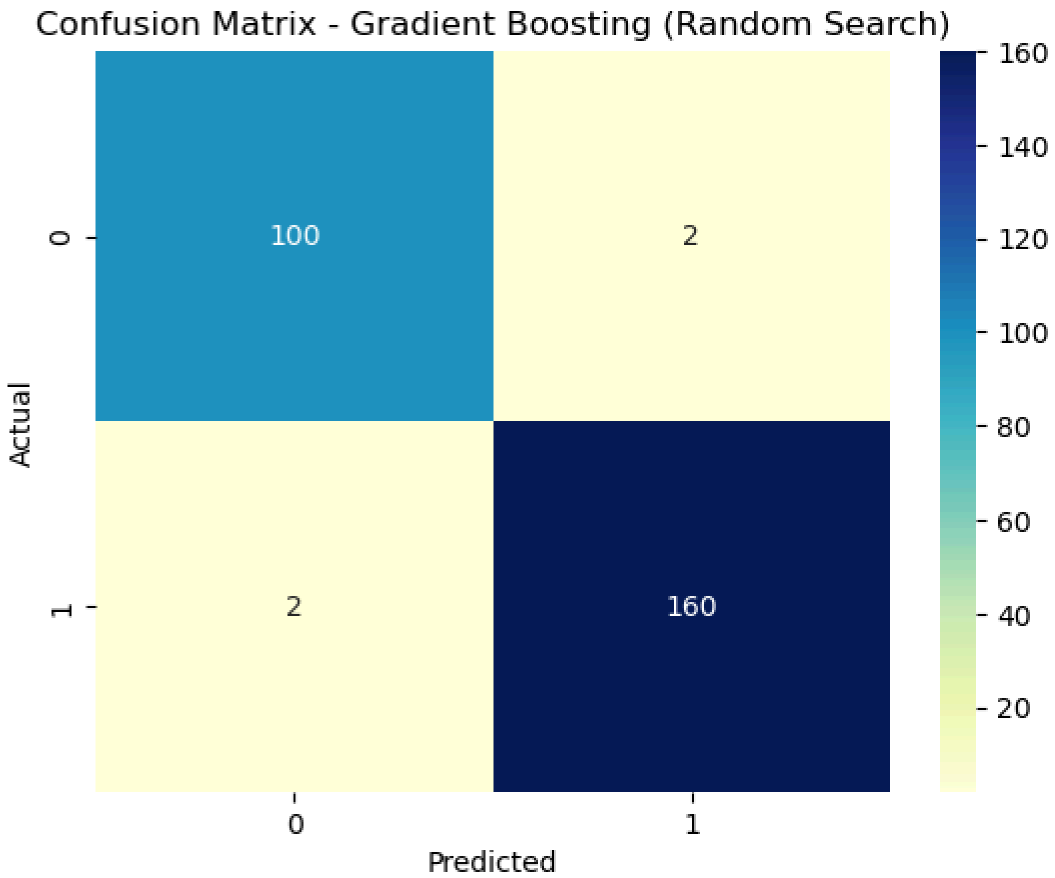


Gradient Boosting Machines

In [57]:
```
## Import required modules
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

## Define the model
gbm = GradientBoostingClassifier(random_state=42)

## Define hyperparameter space
param_dist = {
    'n_estimators': np.arange(80, 201, 20),         # 80 to 200 in steps of 20
    'learning_rate': [0.01, 0.03, 0.05, 0.1],
    'max_depth': [3, 4, 5, 6],
    'subsample': [0.6, 0.8, 1.0],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

## Cross-validation strategy
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=42)

## Setup RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=gbm,
                                   param_distributions=param_dist,
                                   n_iter=20,                  # try only 20 ran
                                   scoring='f1',
                                   n_jobs=-1,
                                   cv=cv,
                                   verbose=1,
                                   random_state=42)

## Fit the model
best_model = random_search.fit(X_train, y_train)

## Predict on test data
gbm_pred = best_model.predict(X_test)
gbm_score = accuracy_score(y_test, gbm_pred)

## Print best hyperparameters and CV score
print("Best Parameters:\n", random_search.best_params_)
print("Best CV F1 Score:\n", random_search.best_score_)

## Evaluation metrics
print("\nClassification Report:\n", classification_report(y_test, gbm_pred))
print("Accuracy Score:", accuracy_score(y_test, gbm_pred))
print("F1 Score:", f1_score(y_test, gbm_pred, average='macro'))
print("Precision:", precision_score(y_test, gbm_pred, average='macro'))
print("Recall:", recall_score(y_test, gbm_pred, average='macro'))

## Confusion Matrix Plot
cm = confusion_matrix(y_test, gbm_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu')
plt.title("Confusion Matrix - Gradient Boosting (Random Search)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best Parameters:
 {'subsample': 0.8, 'n_estimators': 100, 'min_samples_split': 5, 'min_samples_lea
f': 2, 'max_depth': 5, 'learning_rate': 0.01}
Best CV F1 Score:
 0.9930439894613061

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       102
           1       0.99      0.99      0.99       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

Accuracy Score: 0.9848484848484849
F1 Score: 0.9840232389251997
Precision: 0.9840232389251997
Recall: 0.9840232389251997
```



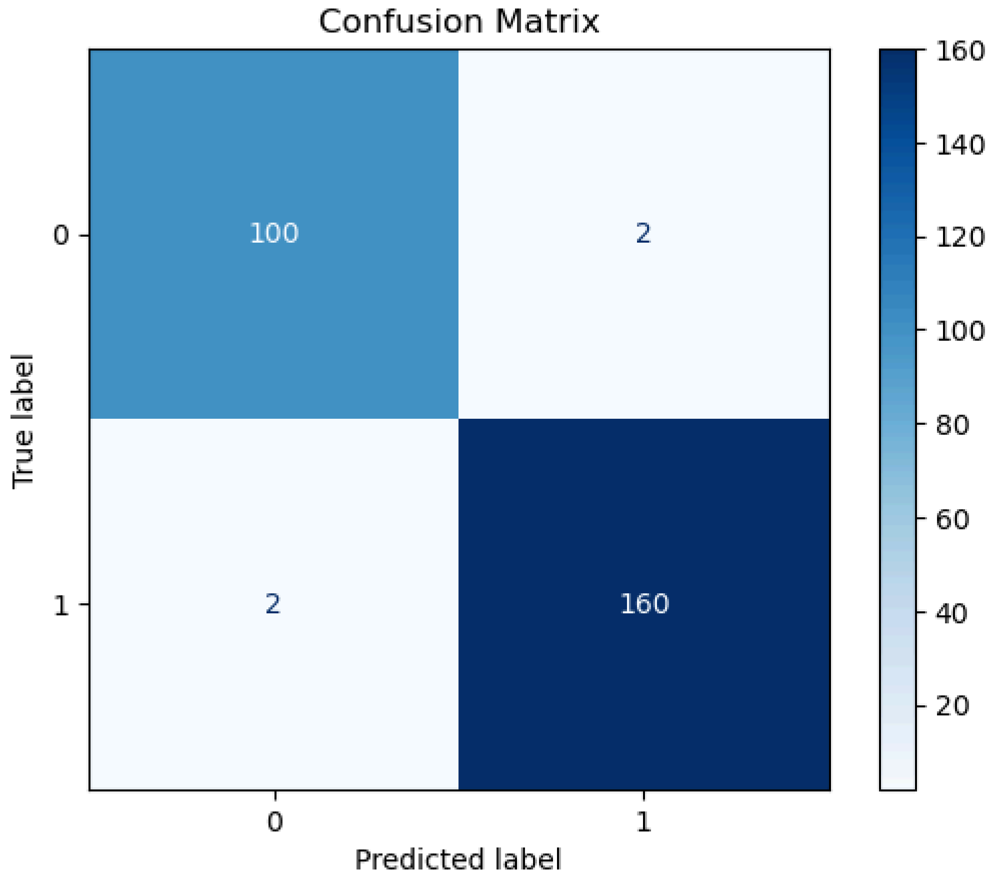
Ada Boost Classifier

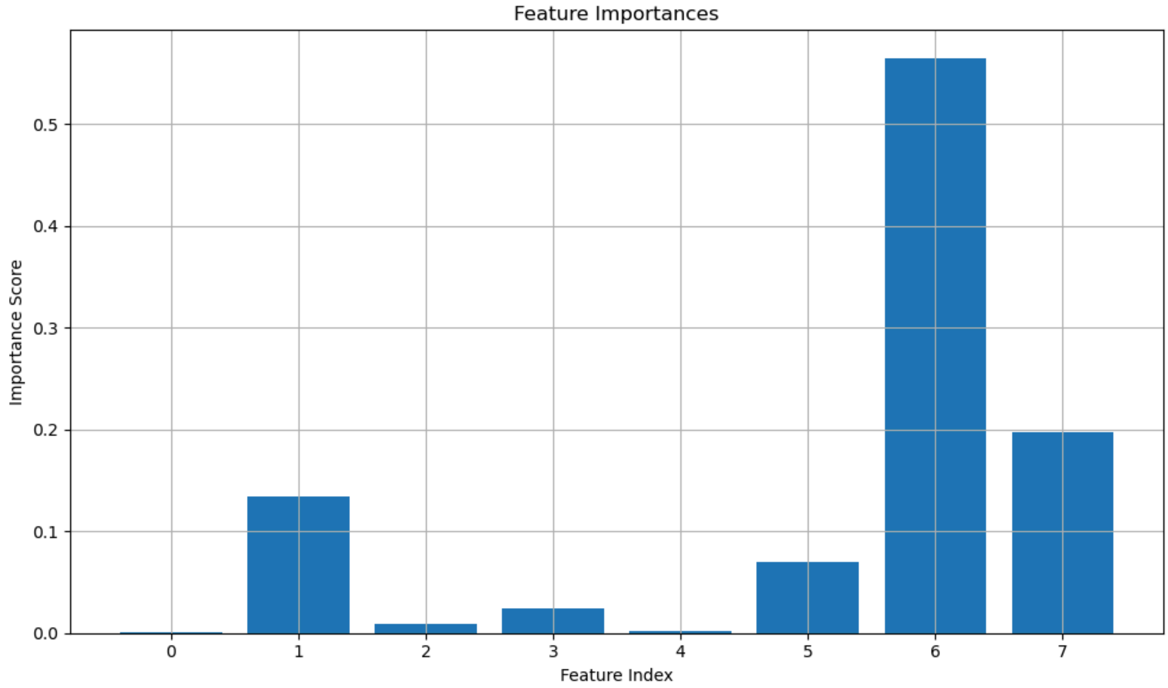
In [58]:
```python
## Import required modules
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sc
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    f1_score,
    precision_score,
```

```
    recall_score,
    confusion_matrix,
    ConfusionMatrixDisplay
)
import matplotlib.pyplot as plt
import numpy as np

## Define base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)

## Define AdaBoost model
ada = AdaBoostClassifier(estimator=base_estimator, n_estimators=180, learning_ra

## Hyperparameter tuning with GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 180, 250],
    'learning_rate': [0.01, 0.1, 1.0],
    'estimator__max_depth': [1, 2, 3]
}

grid_search = GridSearchCV(ada, param_grid, cv=5, scoring='f1_macro', n_jobs=-1)
grid_search.fit(X_train, y_train)

## Best model
best_ada = grid_search.best_estimator_
print("Best Parameters from Grid Search:", grid_search.best_params_)

## Cross-validation scores
cv_scores = cross_val_score(best_ada, X_train, y_train, cv=5, scoring='f1_macro'
print(f"Cross-validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f

## Fit model on full training set
best_ada.fit(X_train, y_train)

## Predict on test set
ada_pred = best_ada.predict(X_test)
ada_score = accuracy_score(y_test, ada_pred)

## Evaluate model
print("\n=== Evaluation on Test Set ===")
print("Classification Report:\n", classification_report(y_test, ada_pred))
print("Accuracy:", accuracy_score(y_test, ada_pred))
print("F1 Score:", f1_score(y_test, ada_pred, average='macro'))
print("Precision:", precision_score(y_test, ada_pred, average='macro'))
print("Recall:", recall_score(y_test, ada_pred, average='macro'))

## Confusion Matrix
cm = confusion_matrix(y_test, ada_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

## Feature Importances
importances = best_ada.feature_importances_
plt.figure(figsize=(10, 6))
plt.bar(np.arange(len(importances)), importances)
plt.title("Feature Importances")
plt.xlabel("Feature Index")
plt.ylabel("Importance Score")
```

```
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Best Parameters from Grid Search: {'estimator__max_depth': 2, 'learning_rate': 0.
01, 'n_estimators': 250}
Cross-validation Accuracy: 0.9880 ± 0.0067

=== Evaluation on Test Set ===
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       102
           1       0.99      0.99      0.99       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

Accuracy: 0.9848484848484849
F1 Score: 0.9840232389251997
Precision: 0.9840232389251997
Recall: 0.9840232389251997
```



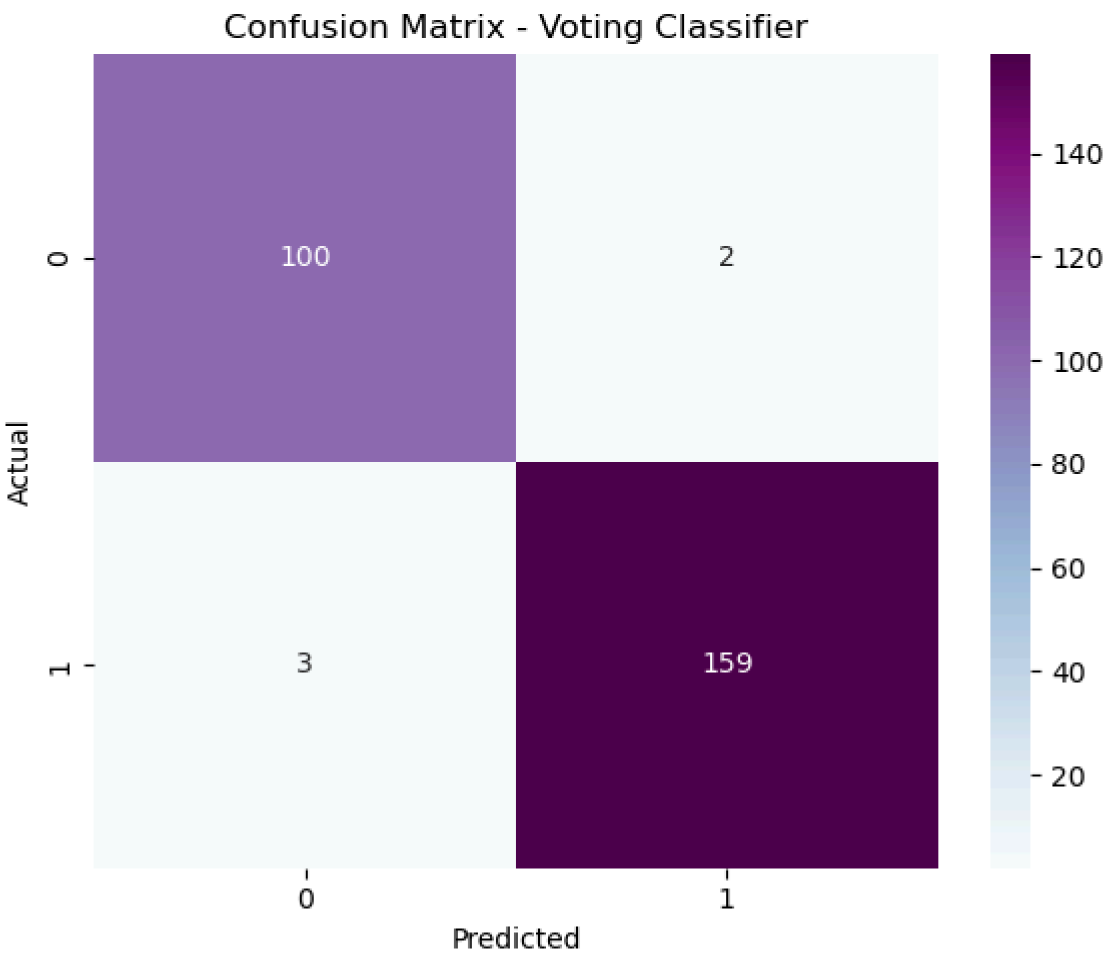


Voting Classifier

In [59]:
```
## Load required libraries
from sklearn.ensemble import VotingClassifier, RandomForestClassifier, AdaBoostC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    classification_report, confusion_matrix,
    accuracy_score, f1_score, precision_score, recall_score
)
from sklearn.model_selection import RepeatedStratifiedKFold

## Define individual base models
log_clf = LogisticRegression(solver='liblinear', random_state=42)
rf_clf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
ada_clf = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1, random_state=42),
    n_estimators=100,
    learning_rate=0.5,
    random_state=42
)

## Combine them into a Voting Classifier (soft voting)
voting_clf = VotingClassifier(
    estimators=[
        ('lr', log_clf),
        ('rf', rf_clf),
        ('ada', ada_clf)
    ],
    voting='soft',  # soft = uses probabilities
    n_jobs=-1
)

## Fit the model
voting_clf.fit(X_train, y_train)

## Make predictions
vote_pred = voting_clf.predict(X_test)
```

```
vote_score = accuracy_score(y_test, vote_pred)

## Evaluate
print("Classification Report:\n", classification_report(y_test, vote_pred))
print("Accuracy:", accuracy_score(y_test, vote_pred))
print("F1 Score:", f1_score(y_test, vote_pred, average='macro'))
print("Precision:", precision_score(y_test, vote_pred, average='macro'))
print("Recall:", recall_score(y_test, vote_pred, average='macro'))

## Confusion matrix
cm = confusion_matrix(y_test, vote_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='BuPu')
plt.title("Confusion Matrix - Voting Classifier")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.98      0.98       102
           1       0.99      0.98      0.98       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

Accuracy: 0.9810606060606061
F1 Score: 0.9800649399682851
Precision: 0.9792257130796599
Recall: 0.9809368191721133
```

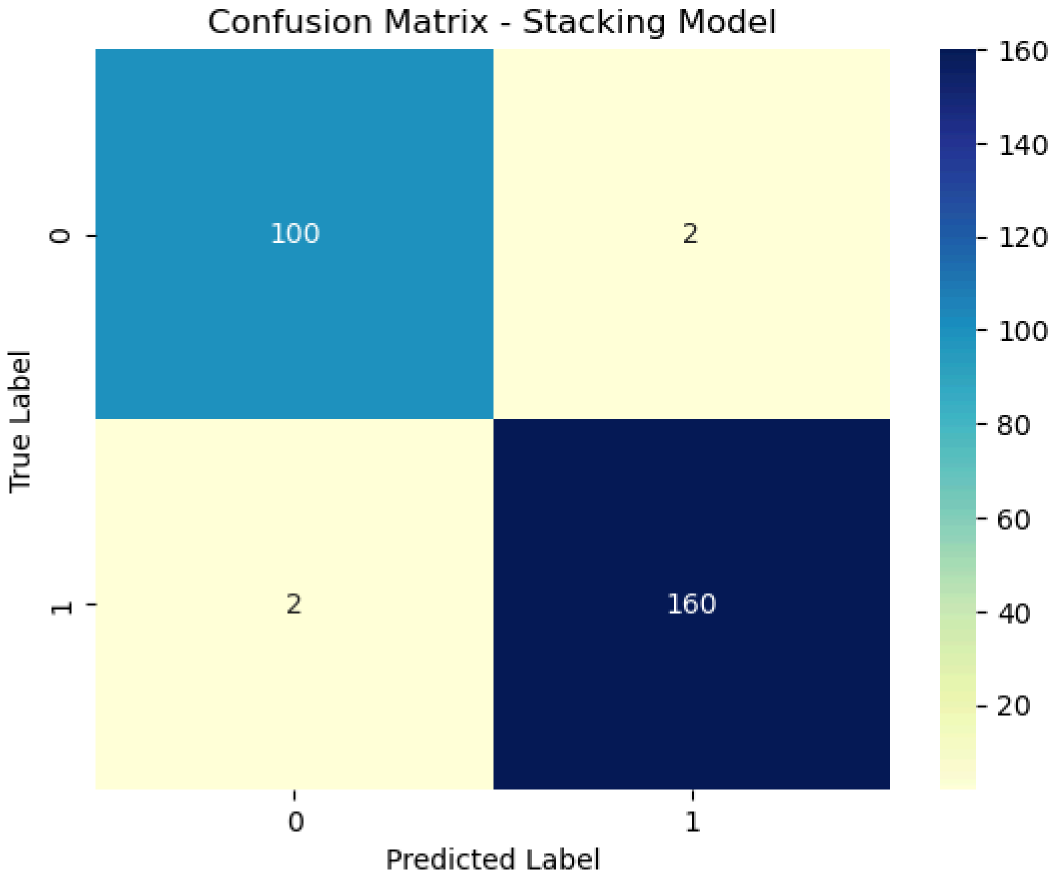


Stacking Classifier

In [60]:
```
## Import required libraries
from sklearn.ensemble import StackingClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_mat

## Define base models
base_learners = [
    ('logreg', LogisticRegression(max_iter=1000)),
    ('dt', DecisionTreeClassifier()),
    ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
    ('svc', SVC(C=10, kernel='rbf', gamma='scale', probability=True)),  # SVM ne
    ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss', random
    ('knn', KNeighborsClassifier(n_neighbors=5)),
]

## Define the meta-model
meta_learner = LogisticRegression()

## Define the stacking classifier
stack_model = StackingClassifier(
    estimators=base_learners,
    final_estimator=meta_learner,
    cv=5,
    n_jobs=-1,
    passthrough=False
)
```

```python
## Fit the model
stack_model.fit(X_train, y_train)

## Make predictions
y_pred = stack_model.predict(X_test)
stack_score = accuracy_score(y_test, y_pred)

## Evaluate performance
print("STACKING MODEL PERFORMANCE")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))
print("Precision:", precision_score(y_test, y_pred, average='macro'))
print("Recall:", recall_score(y_test, y_pred, average='macro'))

## Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='YlGnBu'
plt.title("Confusion Matrix - Stacking Model")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
STACKING MODEL PERFORMANCE
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98       102
           1       0.99      0.99      0.99       162

    accuracy                           0.98       264
   macro avg       0.98      0.98      0.98       264
weighted avg       0.98      0.98      0.98       264

Accuracy: 0.9848484848484849
F1 Score: 0.9840232389251997
Precision: 0.9840232389251997
Recall: 0.9840232389251997
```

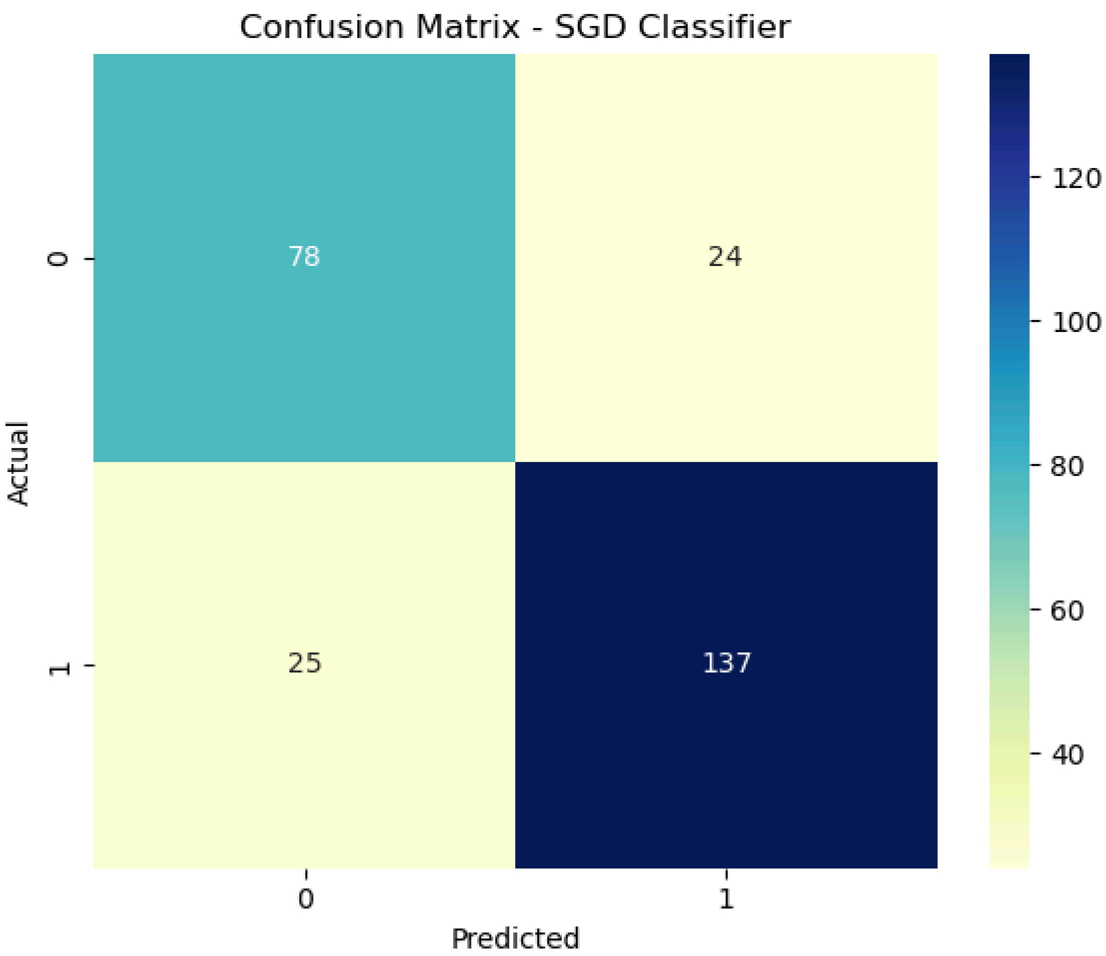


Stochastic Gradient Descent

In [61]:
```
## Load the required modules
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    classification_report, confusion_matrix,
    accuracy_score, f1_score, precision_score, recall_score
)

## Initialize the model
sgd = SGDClassifier(random_state=42)

## Define the parameters to tune
param_grid = {
    'alpha': [0.0001, 0.001, 0.01, 0.1, 1],
    'loss': ['hinge', 'log_loss'],  # 'log_loss' for logistic regression
    'penalty': ['l1', 'l2']
}

## Setup Grid Search
grid_search = GridSearchCV(estimator=sgd,
                           param_grid=param_grid,
                           cv=10,
                           scoring='f1_macro',  # Better for multi-class or imba
                           n_jobs=-1,
                           verbose=1)

## Fit the model
grid_search.fit(X_train, y_train)

## Make Predictions
```

```python
sgd_pred = grid_search.predict(X_test)
sgd_score = accuracy_score(y_test, sgd_pred)

## Best parameters and CV score
print("Best Parameters:", grid_search.best_params_)
print("Best CV F1 Macro Score:", grid_search.best_score_)

## Performance evaluation
print("\n📊 Classification Report:\n", classification_report(y_test, sgd_pred))
print("Accuracy:", accuracy_score(y_test, sgd_pred))
print("F1 Score (Macro):", f1_score(y_test, sgd_pred, average='macro'))
print("Precision (Macro):", precision_score(y_test, sgd_pred, average='macro'))
print("Recall (Macro):", recall_score(y_test, sgd_pred, average='macro'))

## Confusion Matrix
cm = confusion_matrix(y_test, sgd_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu')
plt.title("Confusion Matrix - SGD Classifier")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best Parameters: {'alpha': 0.0001, 'loss': 'log_loss', 'penalty': 'l1'}
Best CV F1 Macro Score: 0.7523741862801936

📊 Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.76      0.76       102
           1       0.85      0.85      0.85       162

    accuracy                           0.81       264
   macro avg       0.80      0.81      0.80       264
weighted avg       0.81      0.81      0.81       264

Accuracy: 0.8143939393939394
F1 Score (Macro): 0.8046364116891943
Precision (Macro): 0.8041066152083459
Recall (Macro): 0.8051924473493101
```



Model Comparison

In [62]:
```
models = pd.DataFrame({
    'Model': ['Logistic Regression', "Decision Trees", 'Random Forest Classifier
             'Stachastic Gradient Boosting'],
    'Score': [log_score, dt_score, rf_score, svm_score, xgb_score, knn_score, gb
})

models.sort_values(by = 'Score', ascending = False)
```

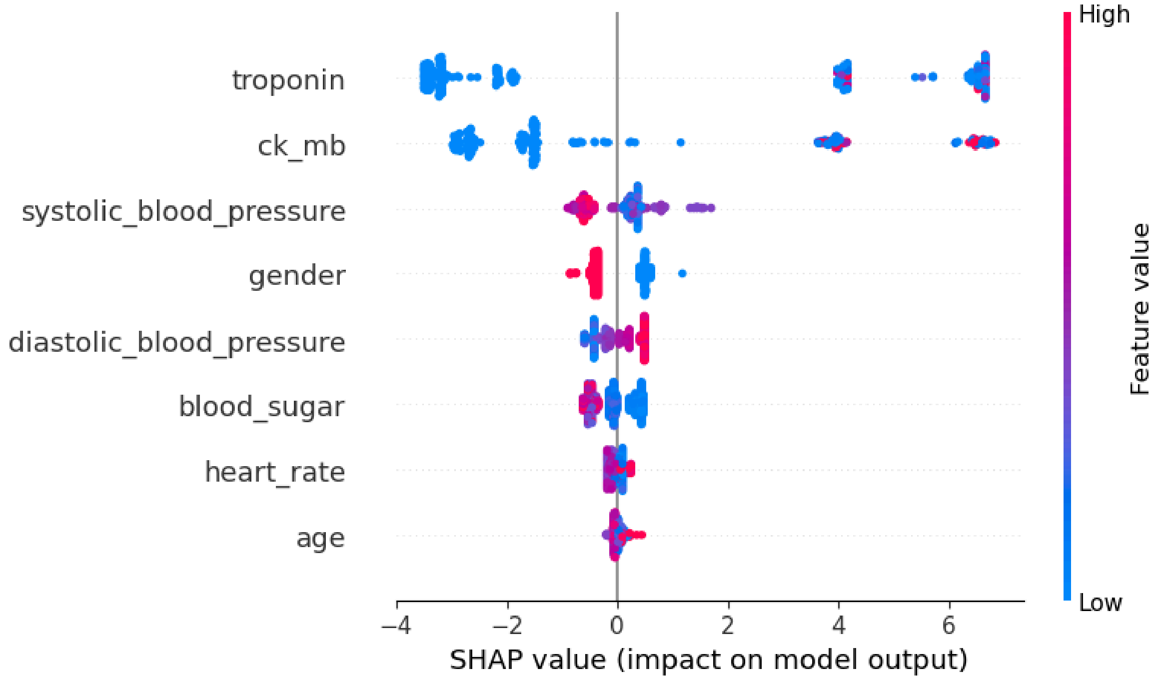
Out[62]:

	Model	Score
4	XG Boost	0.984848
6	Gradient Boosting	0.984848
7	Ada Boost Classifier	0.984848
9	Stacking Classifier	0.984848
1	Decision Trees	0.981061
2	Random Forest Classifier	0.981061
8	Voting Classifier	0.981061
10	Stachastic Gradient Boosting	0.814394
3	SVM	0.787879
0	Logistic Regression	0.712121
5	KNN	0.689394

Best model - XGBoost

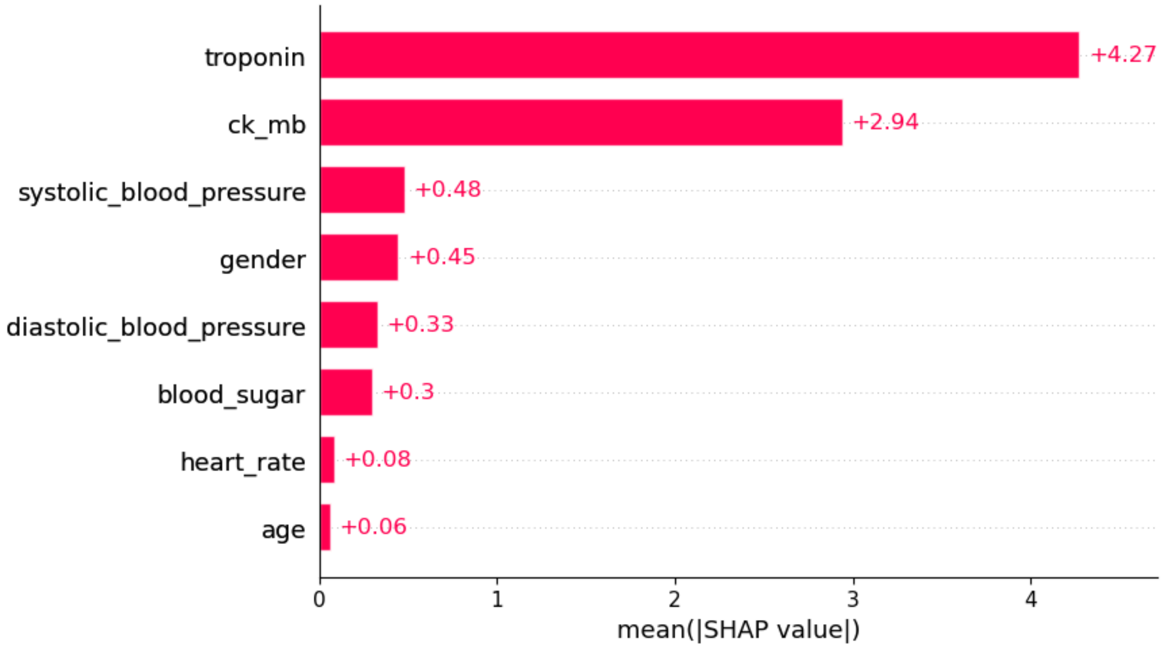
SHAP Values for Deeper Interpretability

In [63]:
```
feature_names = [
    'age',
    'gender',
    'heart_rate',
    'systolic_blood_pressure',
    'diastolic_blood_pressure',
    'blood_sugar',
    'ck_mb',
    'troponin'
]

import shap

# Create SHAP explainer for XGBoost
explainer = shap.Explainer(best_xgb)

# Compute SHAP values
shap_values = explainer(X_test)

# Plot SHAP summary (global feature importance)
shap.summary_plot(shap_values, X_test, feature_names=feature_names)
```



Bar Chart

In [64]:
```
import pandas as pd
import shap

# Step 1: Define feature names
feature_names = [
    'age',
    'gender',
    'heart_rate',
    'systolic_blood_pressure',
    'diastolic_blood_pressure',
    'blood_sugar',
    'ck_mb',
    'troponin'
]

# Step 2: Convert X_test to a DataFrame with column names
X_test_df = pd.DataFrame(X_test, columns=feature_names)

# Step 3: Create SHAP explainer and compute values
explainer = shap.Explainer(best_xgb)
shap_values = explainer(X_test_df)

# Step 4: Plot SHAP bar chart with proper feature names
shap.plots.bar(shap_values, max_display=8)
```



In [41]:

```
# Save the best model to a file
import pickle
with open('heart_attack_xgb_model.pkl', 'wb') as f:
    pickle.dump(best_xgb, f)
```