Notebook

July 29, 2025

1 Heart Attack Risk Prediction - Machine Learning Project

1.1 DTSC-691: Applied Data Science & Analytics

1.2 Name: Edwin Mutevane

1.2.1 Load the required libraries

[1]:
```python
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')
```

1.2.2 Load dataset

[2]:
```python
df = pd.read_csv('Heart_Attack_Risk_Levels_Dataset.csv').clean_names()
```

1.2.3 Inspect the dataset - first and last few observations of the dataset

[3]:
```python
df.head(5)
```

[3]:
```
   age  gender  heart_rate  systolic_blood_pressure  diastolic_blood_pressure  \
0   63       1          66                      160                        83
1   20       1          94                       98                        46
2   56       1          64                      160                        77
3   66       1          70                      120                        55
4   54       1          64                      112                        65

   blood_sugar  ck_mb  troponin    result risk_level  \
0        160.0   1.80     0.012  negative   Moderate
1        296.0   6.75     1.060  positive       High
2        270.0   1.99     0.003  negative   Moderate
3        270.0  13.87     0.122  positive       High
4        300.0   1.08     0.003  negative   Moderate
```

```
                         recommendation
0  Monitor closely and consult doctor
1          Immediate medical attention
2  Monitor closely and consult doctor
3          Immediate medical attention
4  Monitor closely and consult doctor
```

[4]:
```python
df.tail(5)
```

[4]:
```
      age  gender  heart_rate  systolic_blood_pressure  \
1314   44       1          94                      122
1315   66       1          84                      125
1316   45       1          85                      168
1317   54       1          58                      117
1318   51       1          94                      157

      diastolic_blood_pressure  blood_sugar  ck_mb  troponin    result  \
1314                        67        204.0   1.63     0.006  negative
1315                        55        149.0   1.33     0.172  positive
1316                       104         96.0   1.24     4.250  positive
1317                        68        443.0   5.80     0.359  positive
1318                        79        134.0  50.89     1.770  positive

     risk_level                      recommendation
1314   Moderate  Monitor closely and consult doctor
1315       High         Immediate medical attention
1316       High         Immediate medical attention
1317       High         Immediate medical attention
1318       High         Immediate medical attention
```

1.2.4 Data Types Check

[5]:
```python
df.dtypes
```

[5]:
```
age                           int64
gender                        int64
heart_rate                    int64
systolic_blood_pressure       int64
diastolic_blood_pressure      int64
blood_sugar                 float64
ck_mb                       float64
troponin                    float64
result                       object
risk_level                   object
recommendation               object
dtype: object
```

1.2.5 Structure of the dataset

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1319 entries, 0 to 1318
Data columns (total 11 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   object 
 9   risk_level                1319 non-null   object 
 10  recommendation            1319 non-null   object 
dtypes: float64(3), int64(5), object(3)
memory usage: 113.5+ KB
```

1.2.6 Unique Values and Cardinality

[7]:
```
df.nunique()
```

[7]:
```
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
risk_level                    3
recommendation                3
dtype: int64
```

1.2.7 Checking for missing values

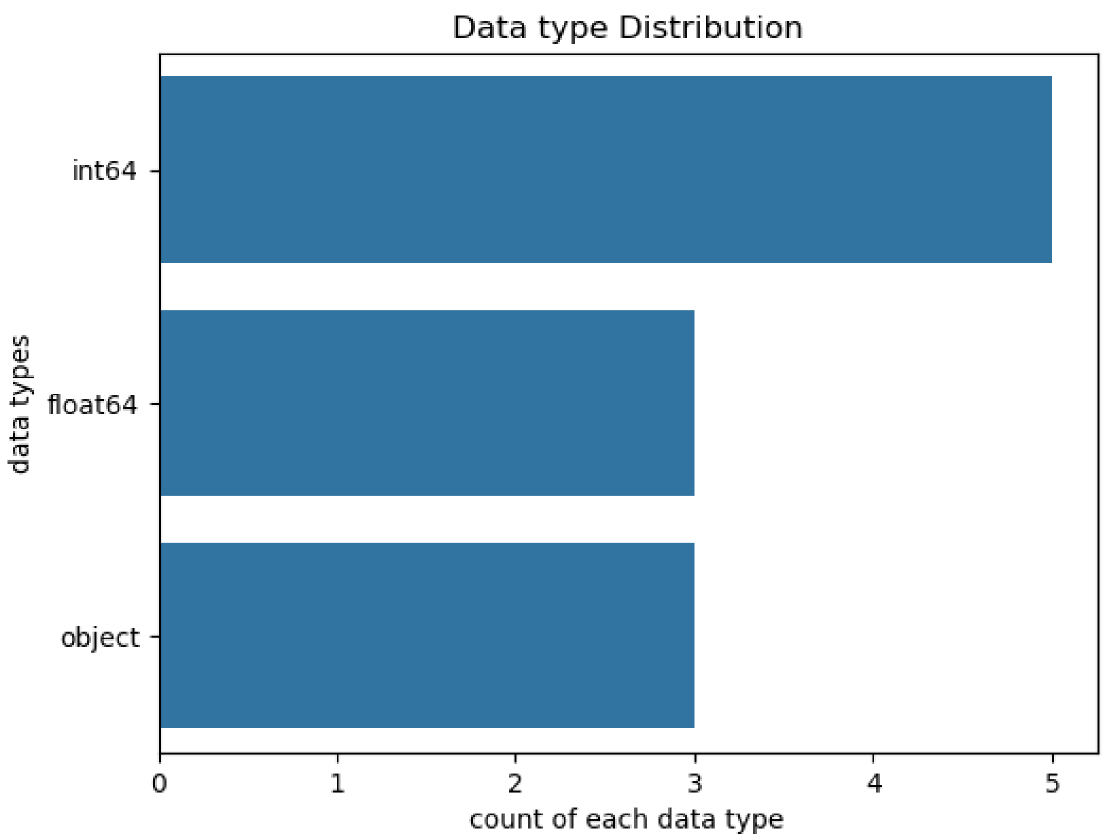
[8]:
```
df.isnull().sum()
```

[8]:
```
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
```

1.2.8 Data type distribution

[9]:
```
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()
```



1.2.9 Check for duplicates

[10]:
```
df.duplicated().sum()
```

[10]:
```
0
```

1.2.10 Data Cleaning & Preprocessing

Drop unwanted columns

[11]:
```
df = df.drop(columns=['risk_level', 'recommendation'])
```

1.2.11 Label encoding of the target variable

[12]:
```
## Initialize LabelEncoder
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()

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

[12]:
```
0       0
1       1
2       0
3       1
4       0
       ..
1314    0
1315    1
1316    1
1317    1
1318    1
Name: result, Length: 1319, dtype: int32
```

1.3 Exploratory Data Analysis

1.3.1 Summary Statistics

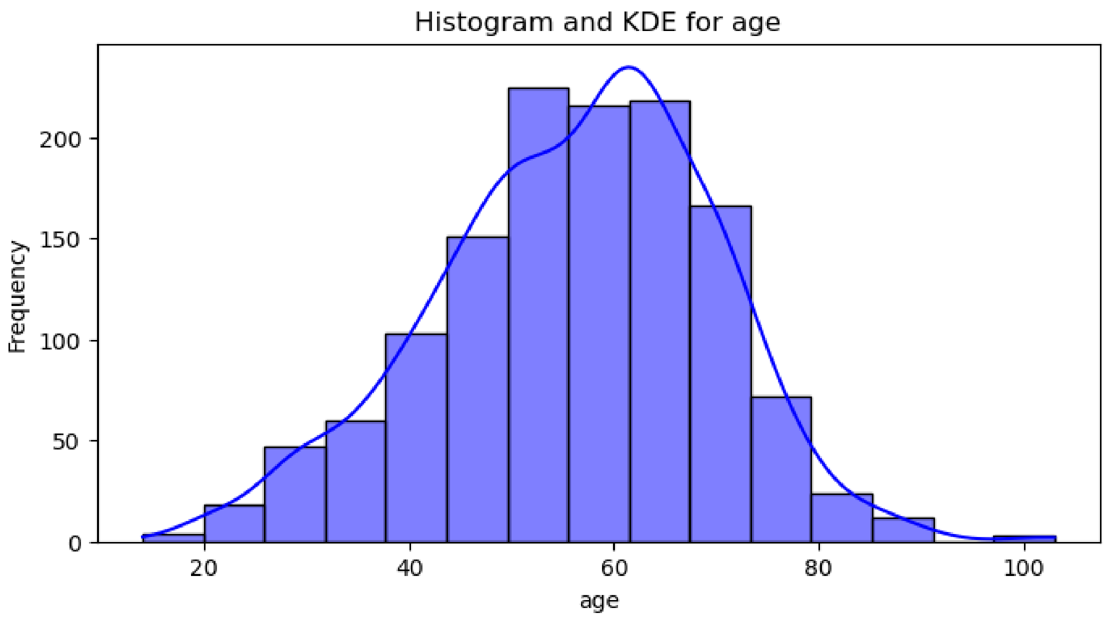
[13]:
```
df.describe()
```

[13]:
```
               age       gender   heart_rate  systolic_blood_pressure  \
count  1319.000000  1319.000000  1319.000000              1319.000000
mean     56.193328     0.659591    78.336619               127.170584
std      13.638173     0.474027    51.630270                26.122720
min      14.000000     0.000000    20.000000                42.000000
25%      47.000000     0.000000    64.000000               110.000000
50%      58.000000     1.000000    74.000000               124.000000
75%      65.000000     1.000000    85.000000               143.000000
```

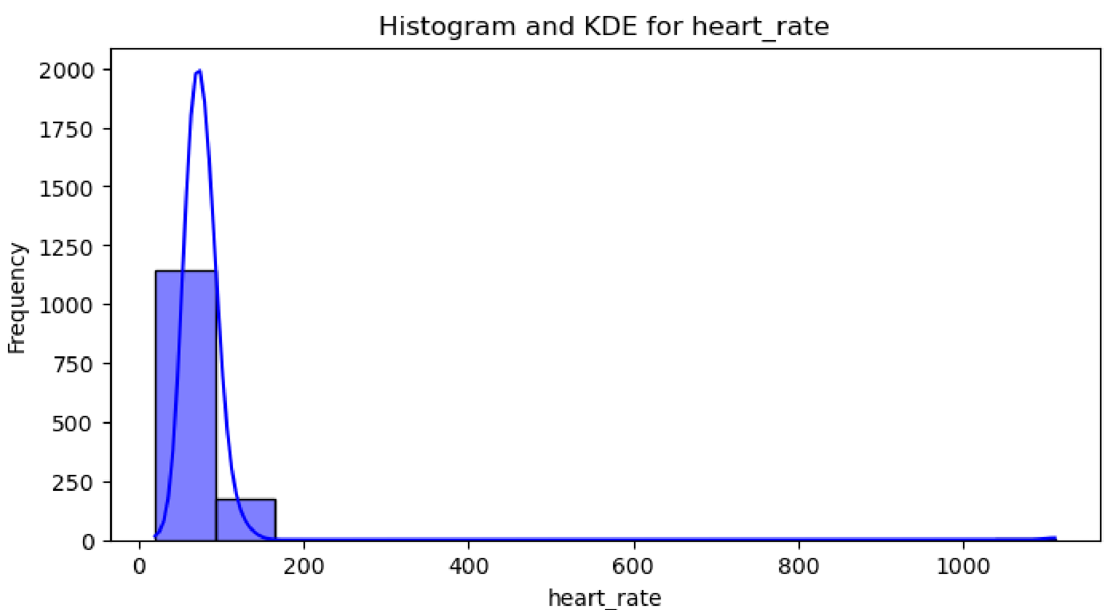
```
max     103.000000     1.000000  1111.000000              223.000000

       diastolic_blood_pressure  blood_sugar        ck_mb     troponin  \
count               1319.000000  1319.000000  1319.000000  1319.000000
mean                  72.269143   146.634344    15.274306     0.360942
std                   14.033924    74.923045    46.327083     1.154568
min                   38.000000    35.000000     0.321000     0.001000
25%                   62.000000    98.000000     1.655000     0.006000
50%                   72.000000   116.000000     2.850000     0.014000
75%                   81.000000   169.500000     5.805000     0.085500
max                  154.000000   541.000000   300.000000    10.300000

            result
count  1319.000000
mean      0.614102
std       0.486991
min       0.000000
25%       0.000000
50%       1.000000
75%       1.000000
max       1.000000
```

1.3.2 Distribution of Numerical Variables

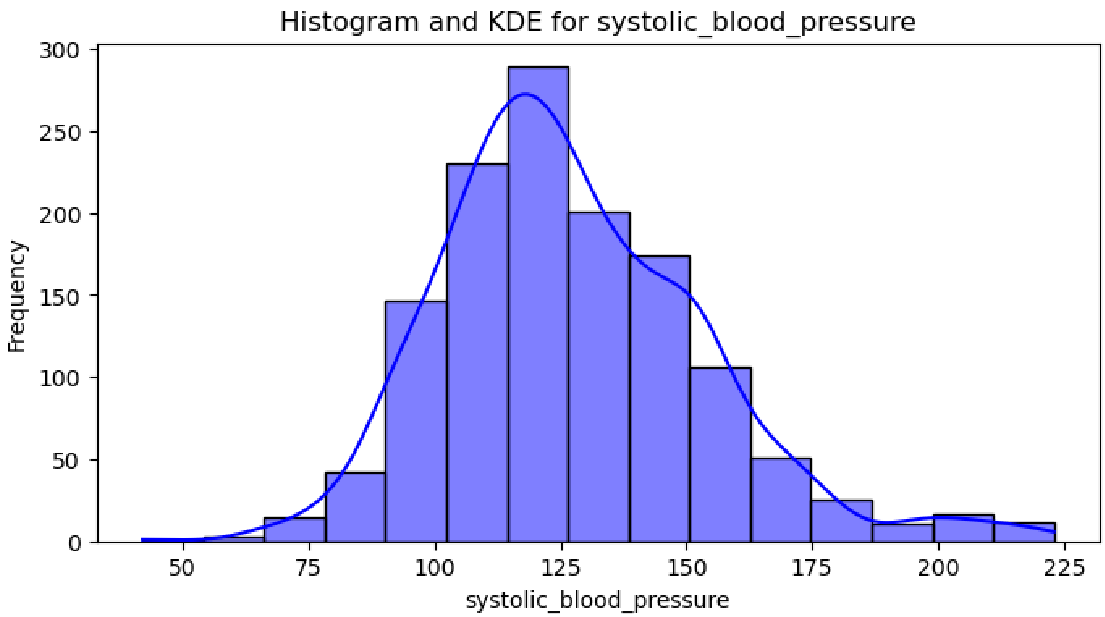
[14]:
```python
numeric_col = df.drop(columns = ["result", "gender"])
for col in numeric_col:
    plt.figure(figsize = (8, 4))
    sns.histplot(df[col], kde = True, bins = 15, color = "blue")
    plt.title(f'Histogram and KDE for {col}')
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

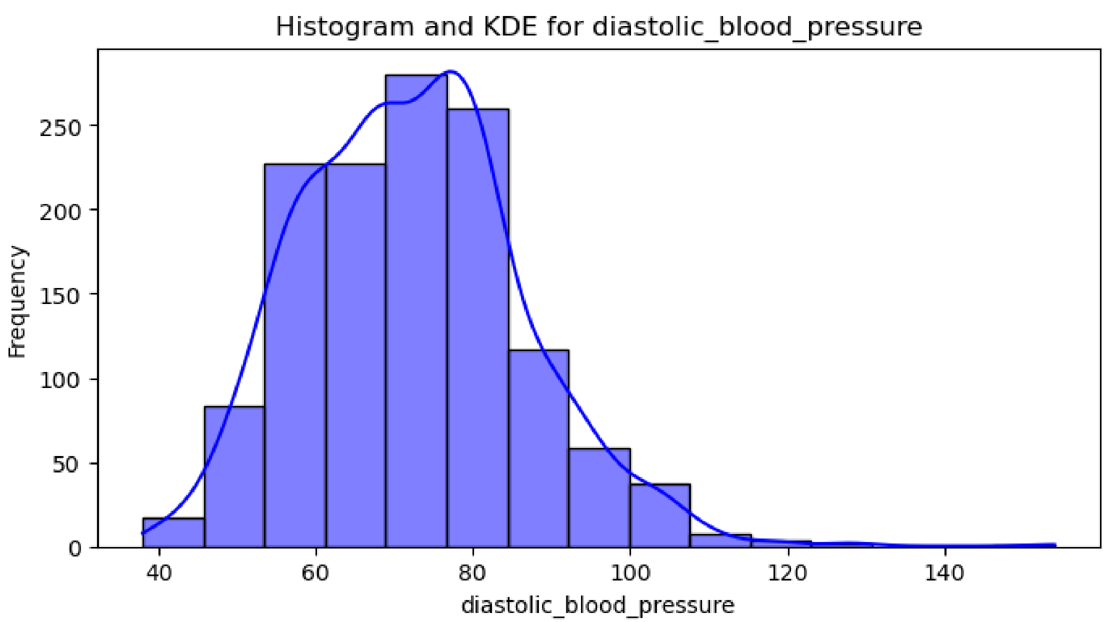


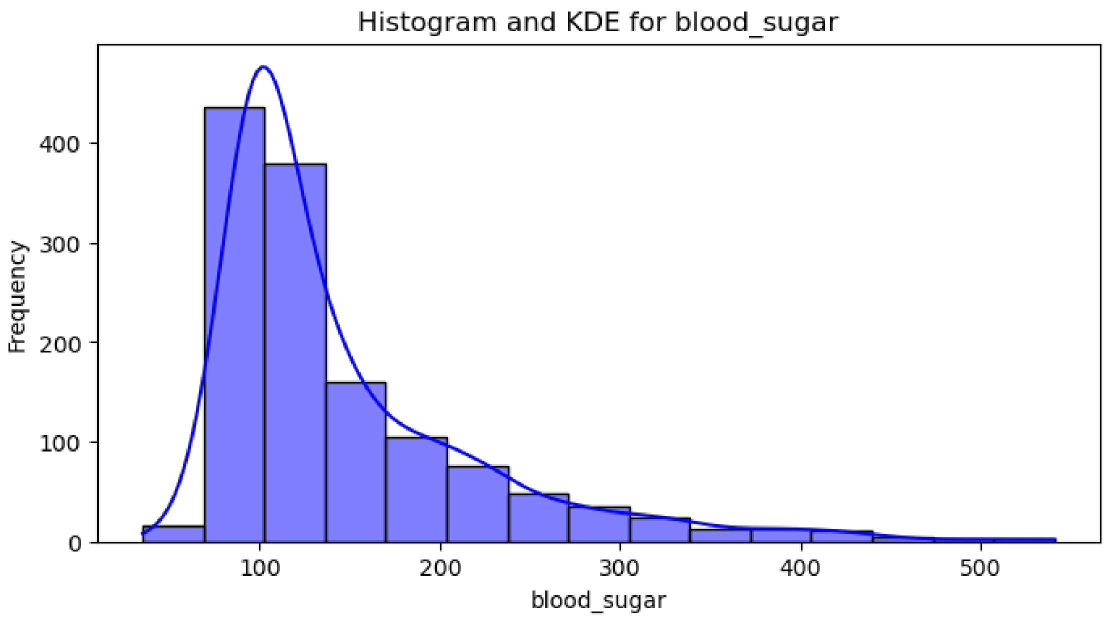
Histogram and KDE for age
Frequency
0
50
100
150
200
20
40
60
80
100
age



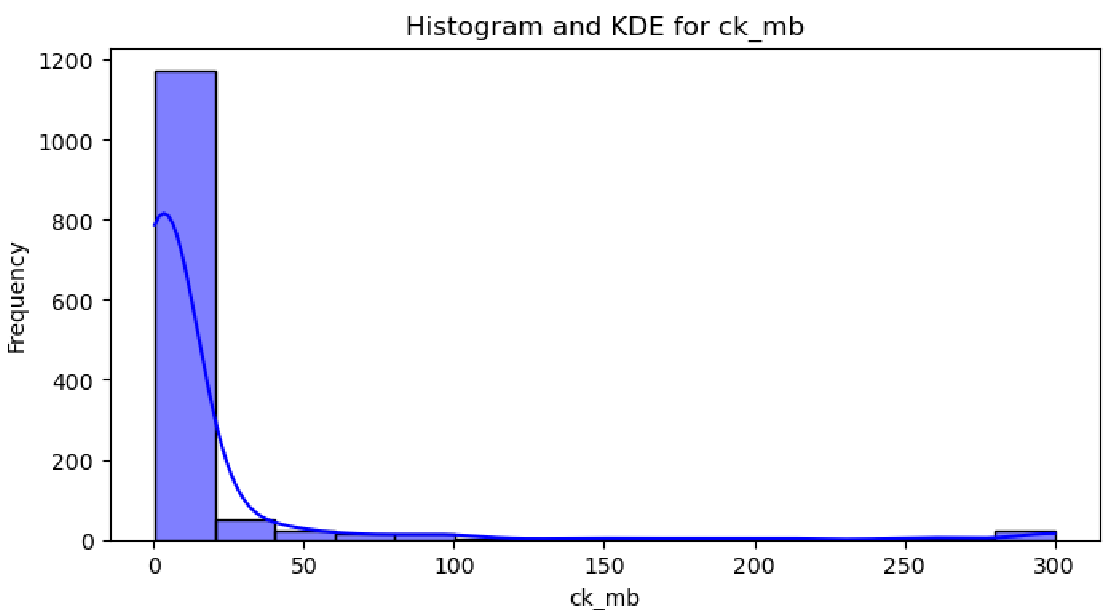
Histogram and KDE for heart_rate
Frequency
0
250
500
750
1000
1250
1500
1750
2000
0
200
400
600
800
1000
heart_rate


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

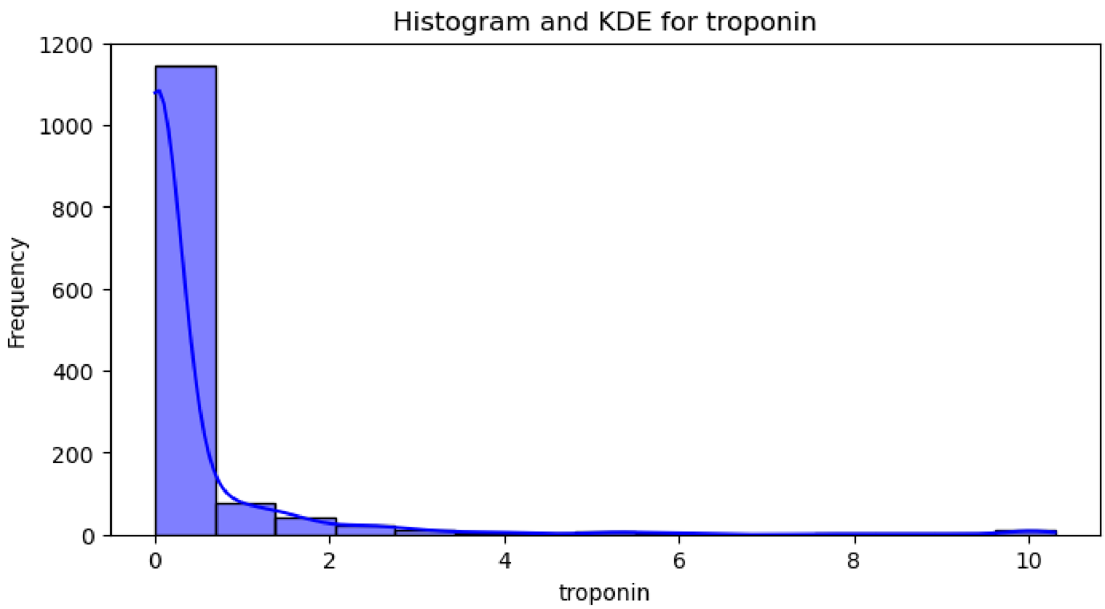

Histogram and KDE for diastolic_blood_pressure
Frequency
diastolic_blood_pressure
0
50
100
150
200
250
40
60
80
100
120
140



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

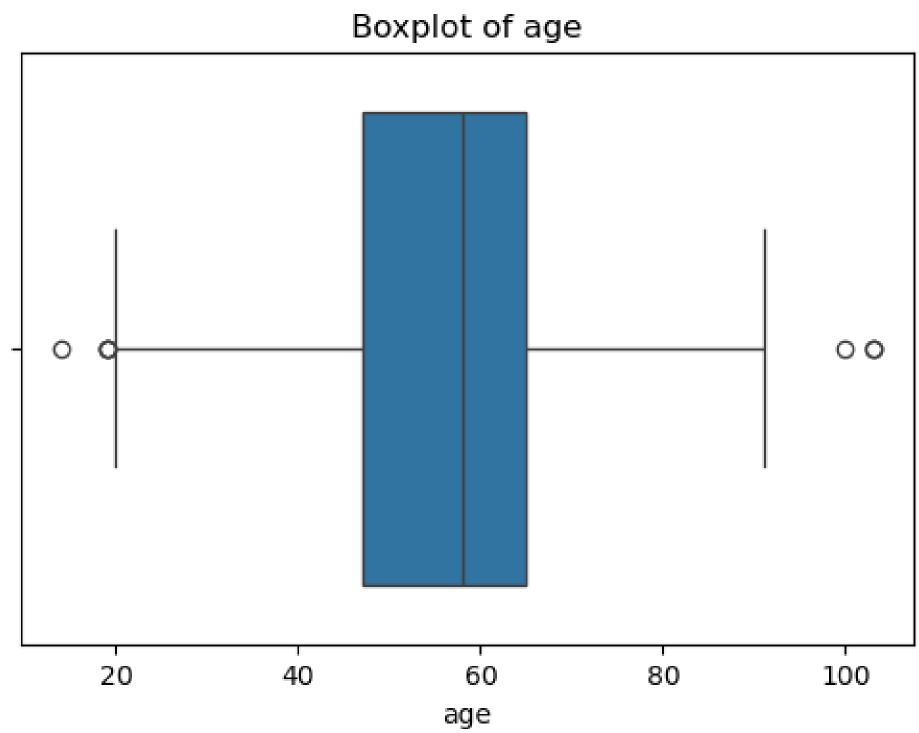


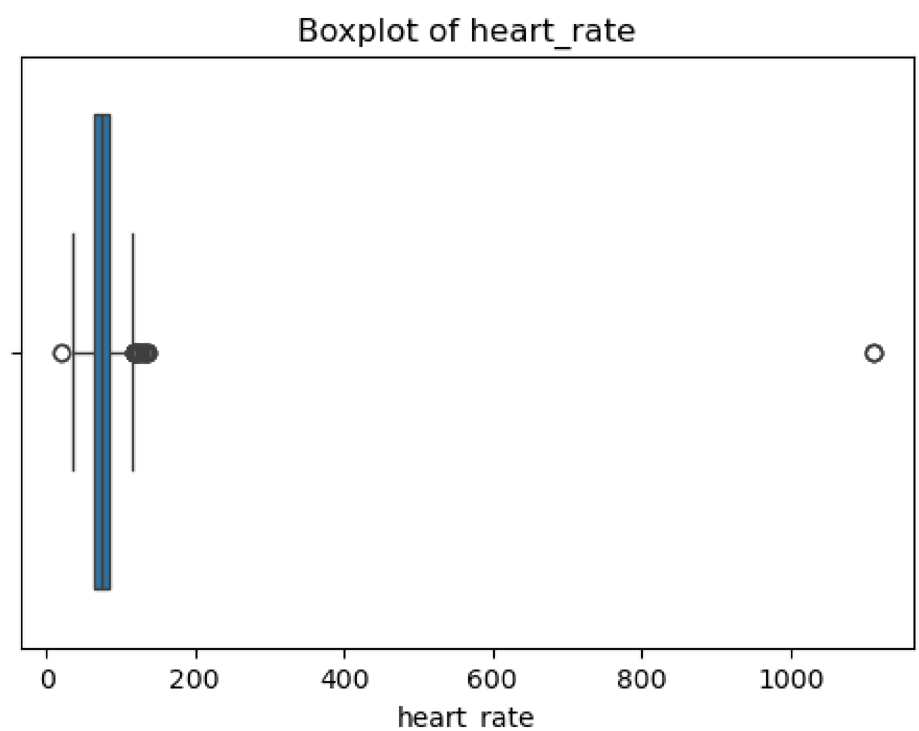
Histogram and KDE for ck_mb
Frequency
ck_mb
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300

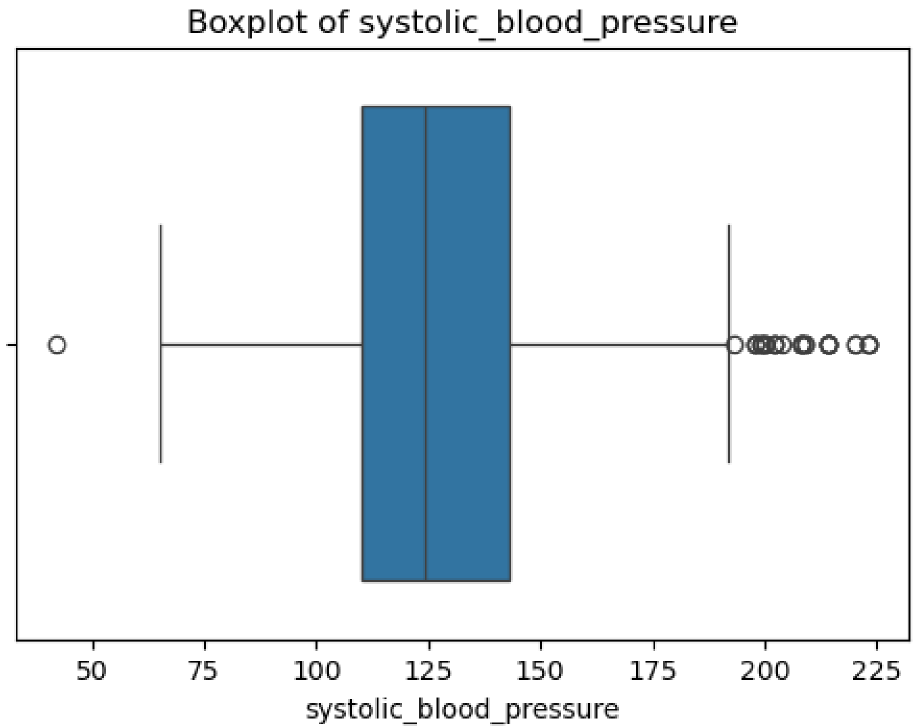


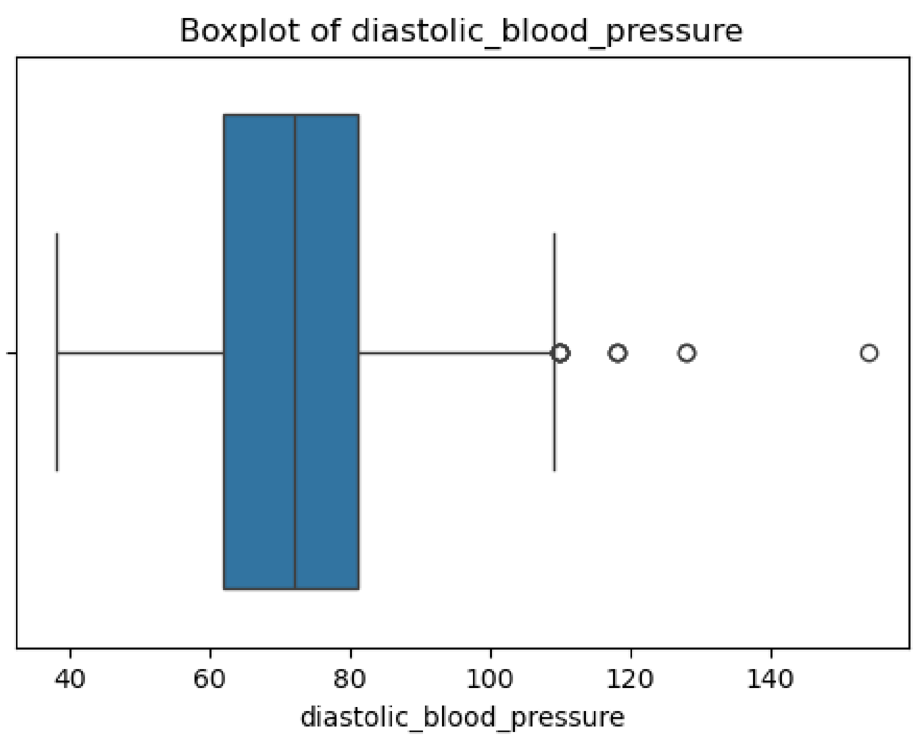
1.3.3 Boxplots to Detect Outliers

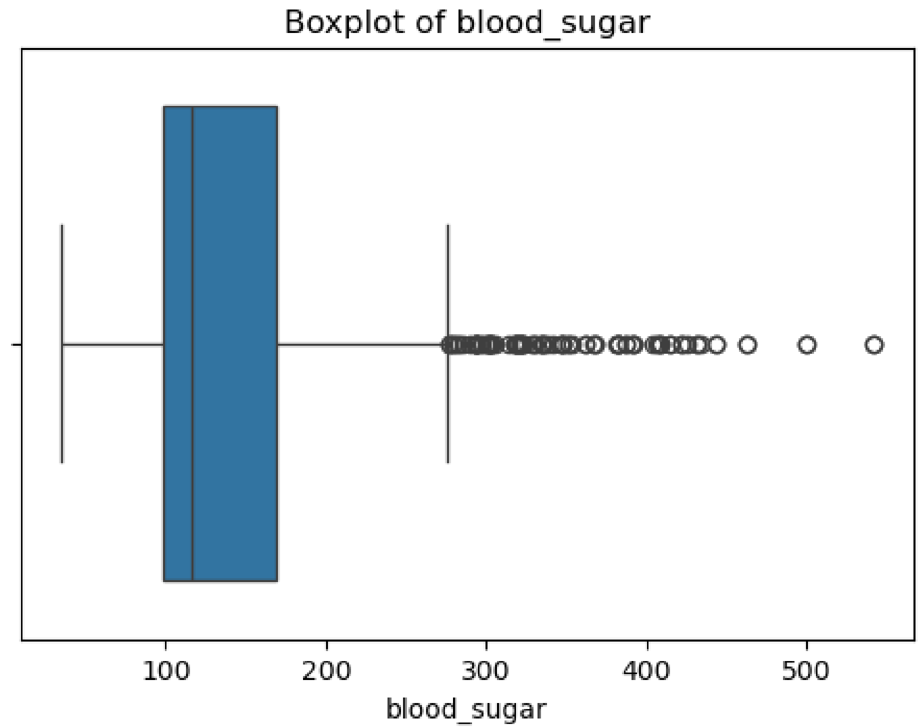
[15]:
```
for col in numeric_col:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

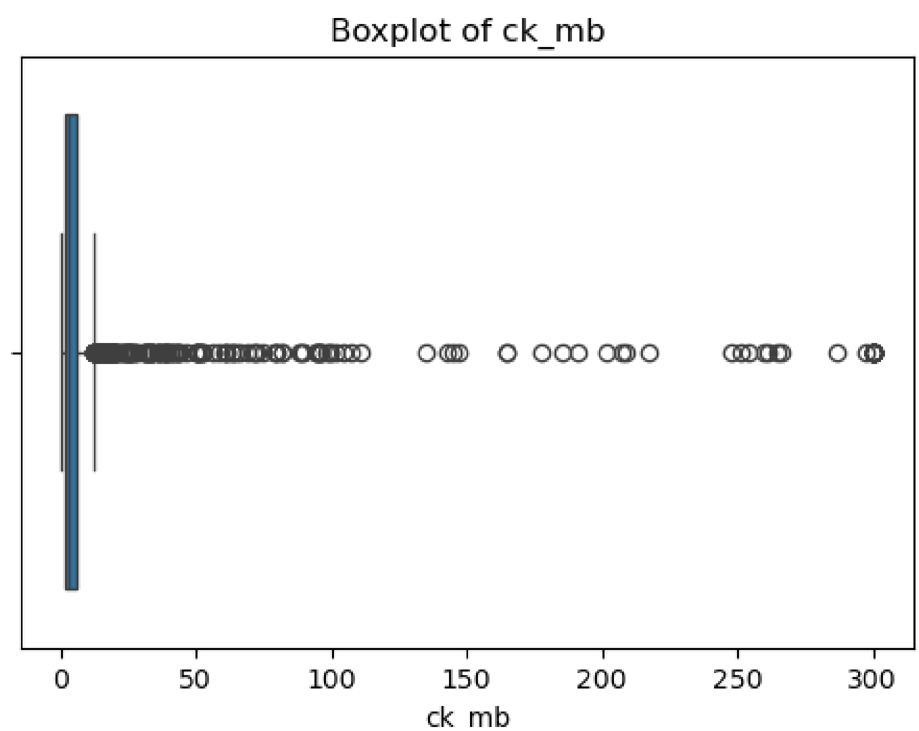

Boxplot of age
20
40
60
80
100
age

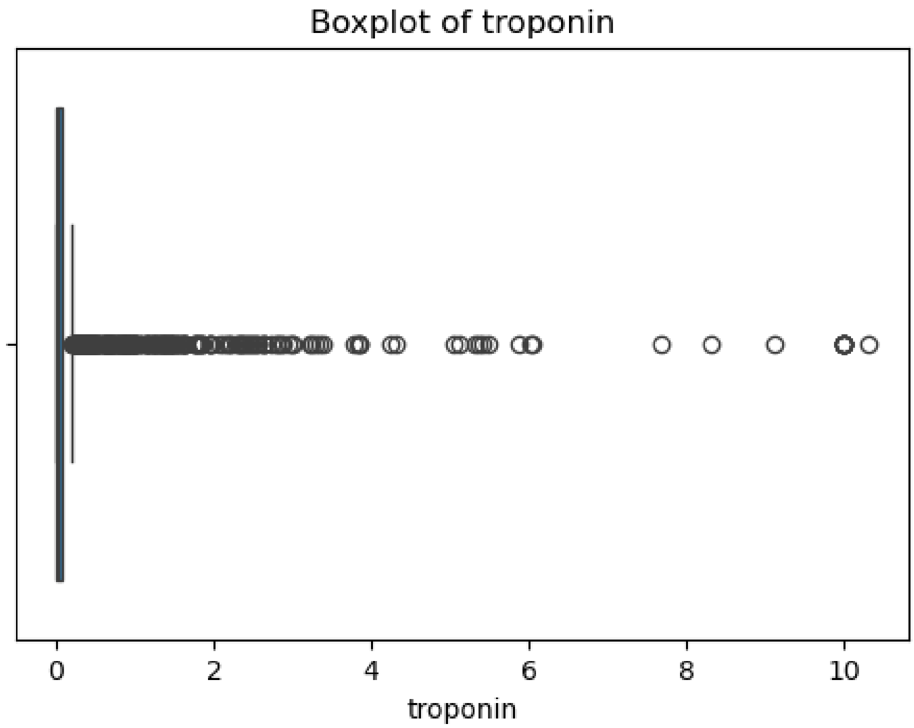

Boxplot of heart_rate
0
200
400
600
800
1000
heart_rate


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

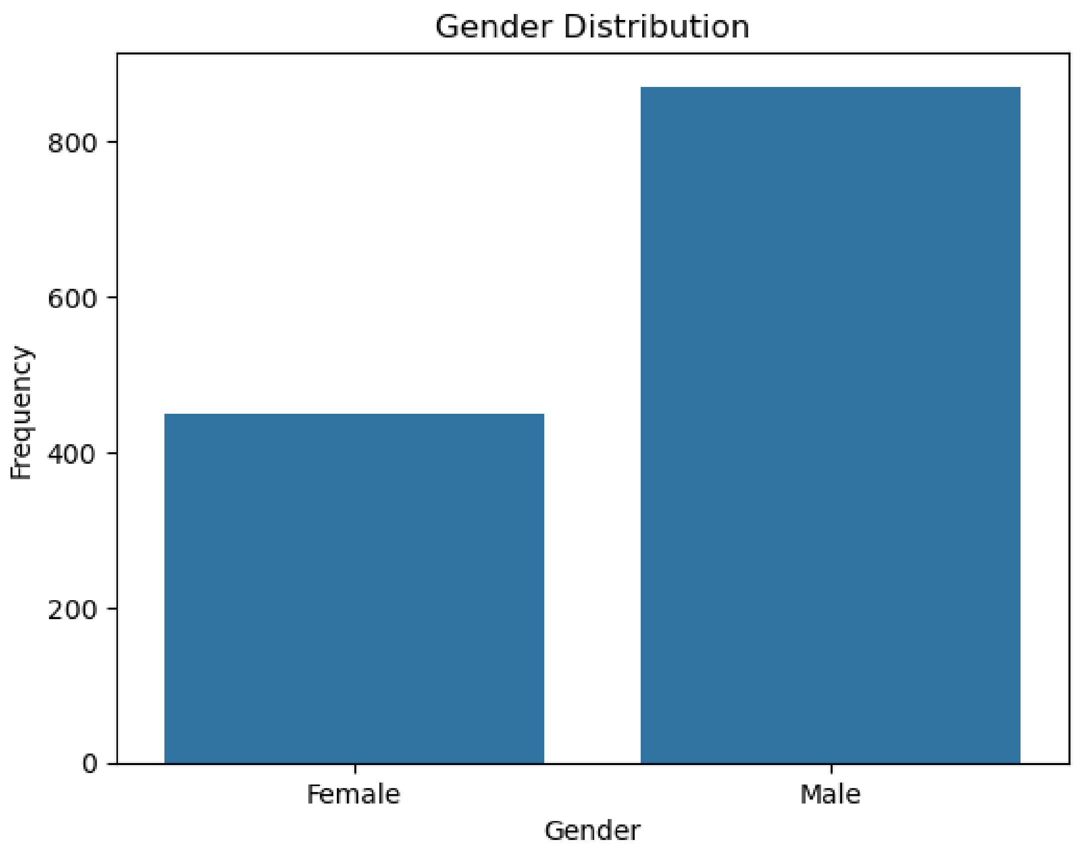

Boxplot of blood_sugar
100
200
300
400
500
blood_sugar


Boxplot of ck_mb
0
50
100
150
200
250
300
ck_mb



1.3.4 Gender Distribution

[16]:
```
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()
```

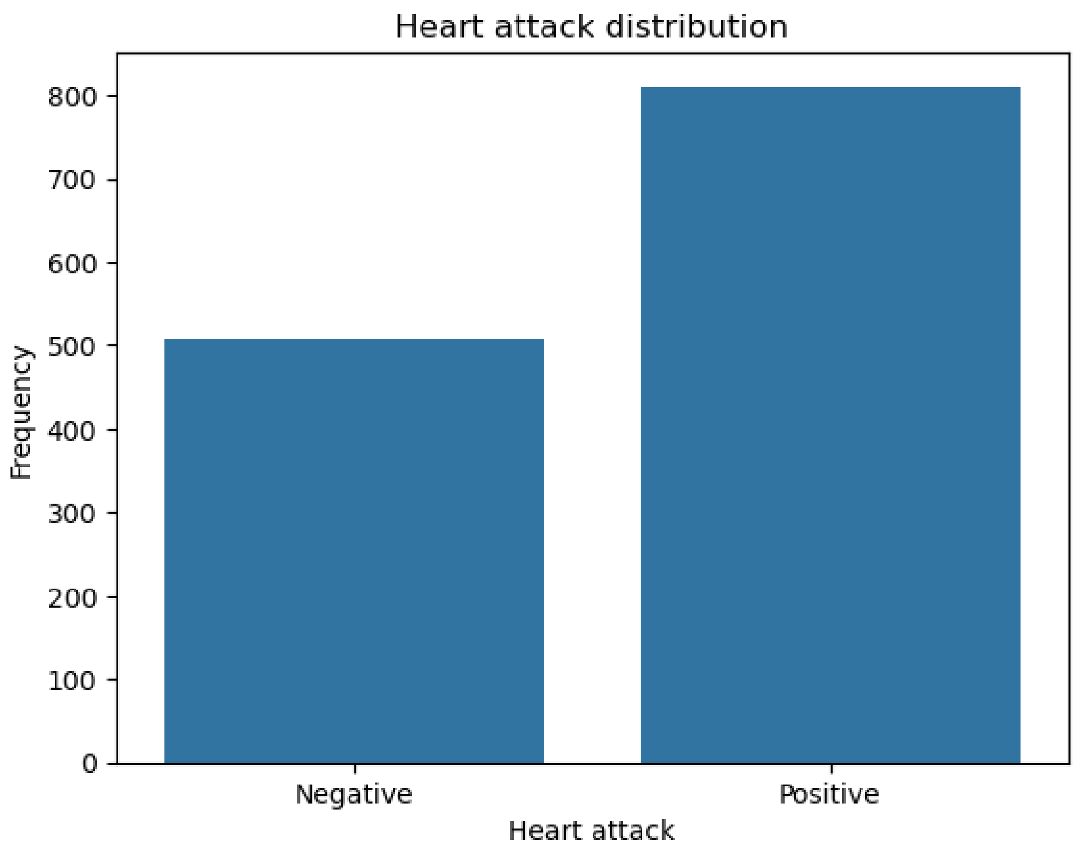


1.3.5 Distribution of the target variable

[17]:
```
df["result"].value_counts()
```

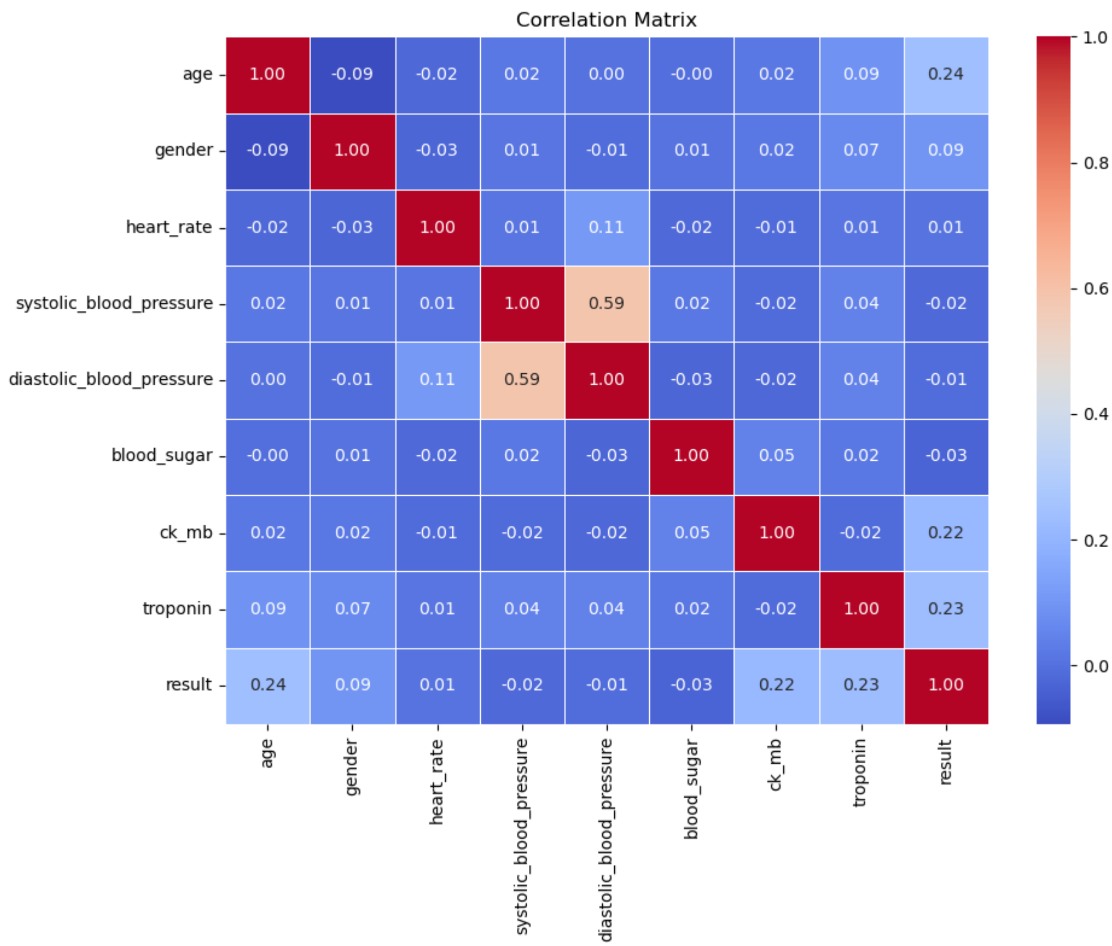
[17]:
```
result
1    810
0    509
Name: count, dtype: int64
```

[18]:
```
## 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()
```



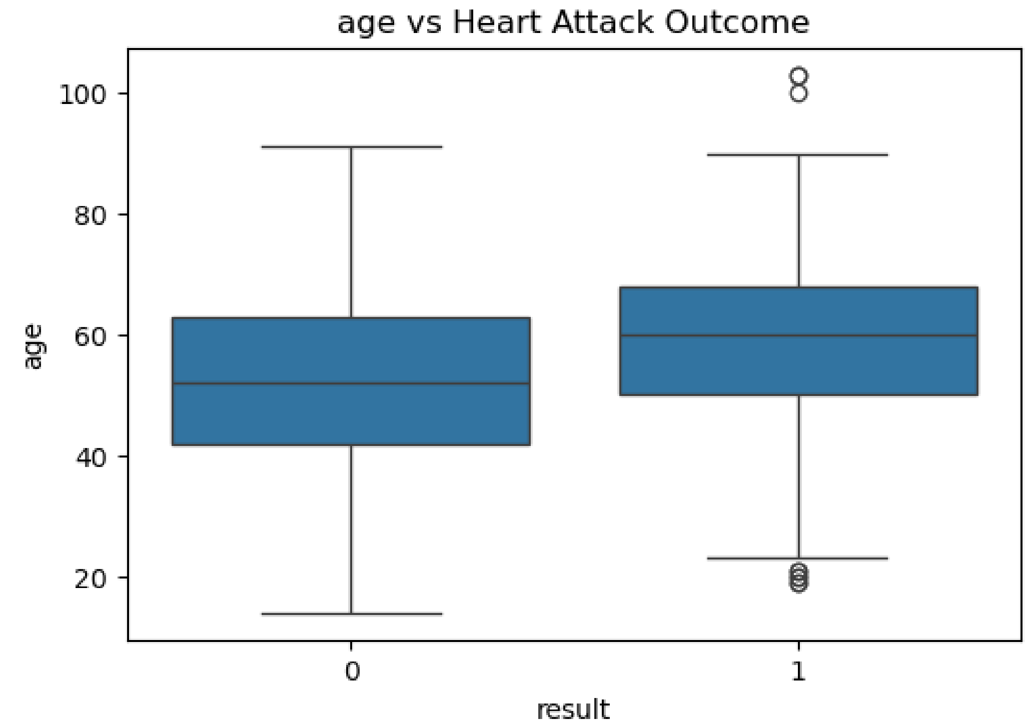
1.3.6 Correlation Matrix

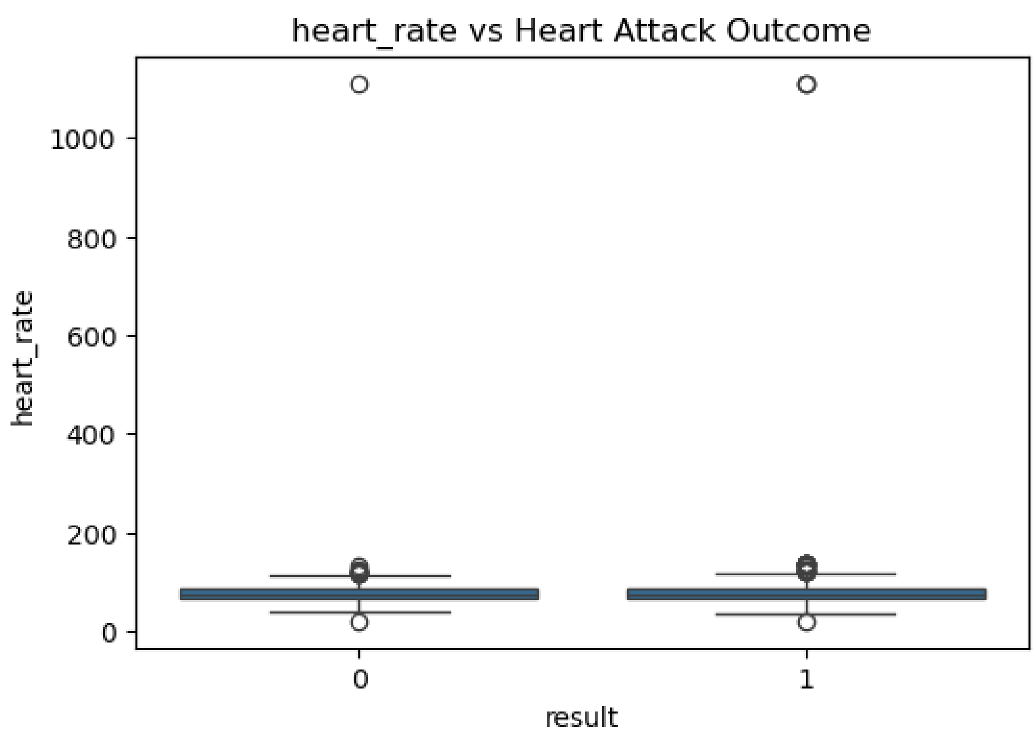
[19]:
```
corr_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

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

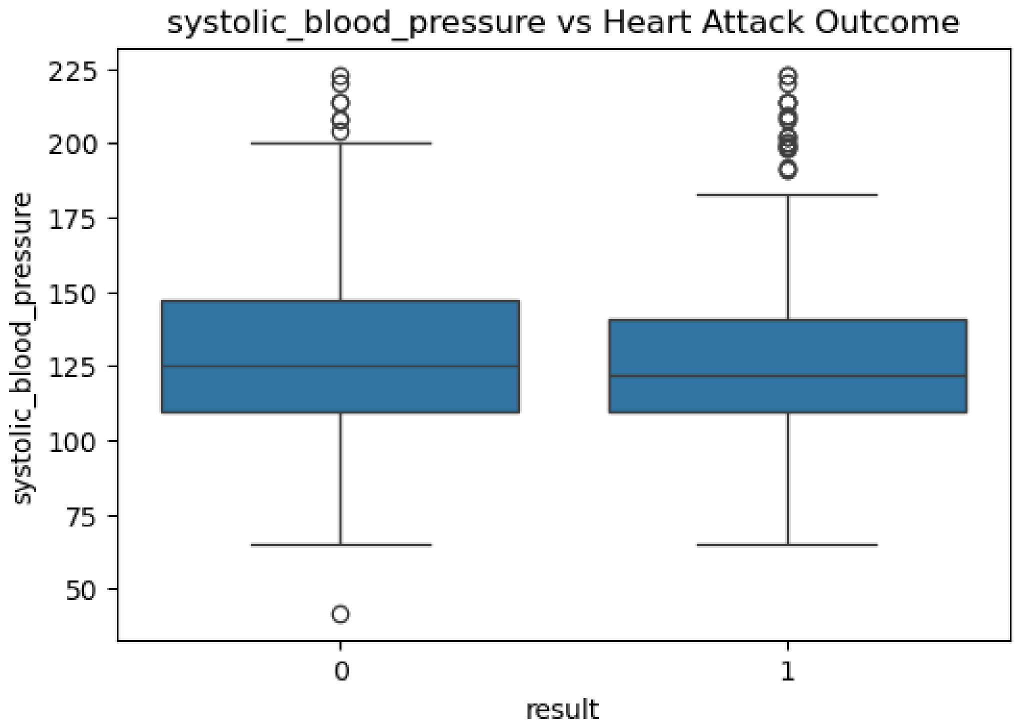


1.3.7 Group-wise Analysis (e.g., Risk Level vs Numeric Variables)

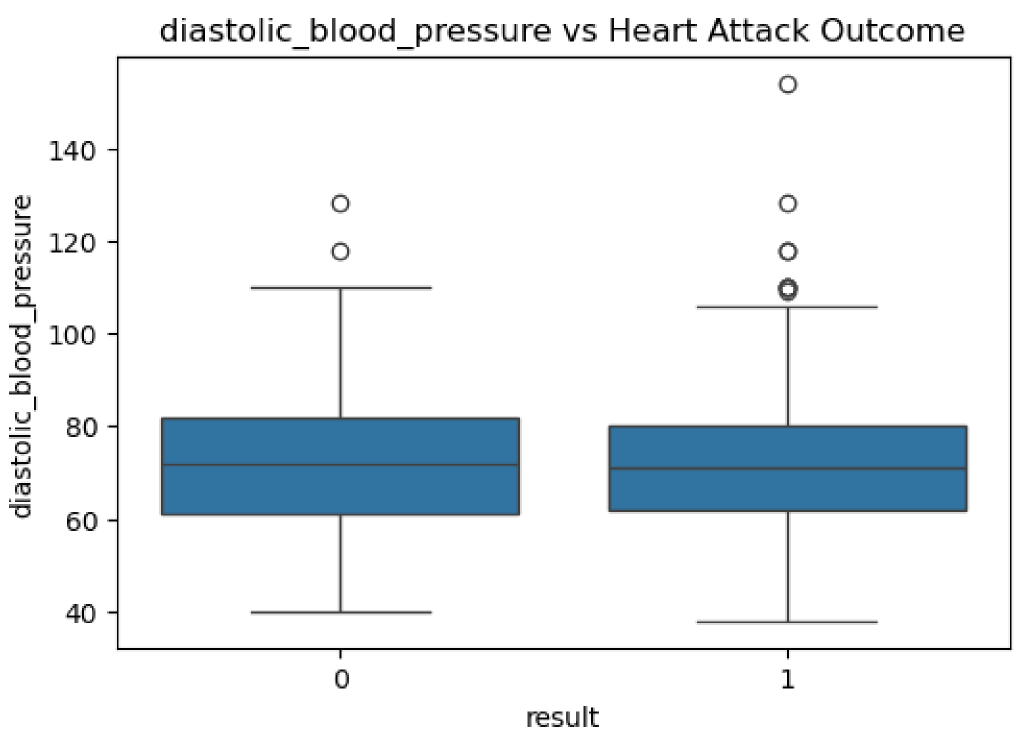
[20]:
```
for col in numeric_col:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x='result', y=col, data=df)
    plt.title(f'{col} vs Heart Attack Outcome')
    plt.show()
```


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

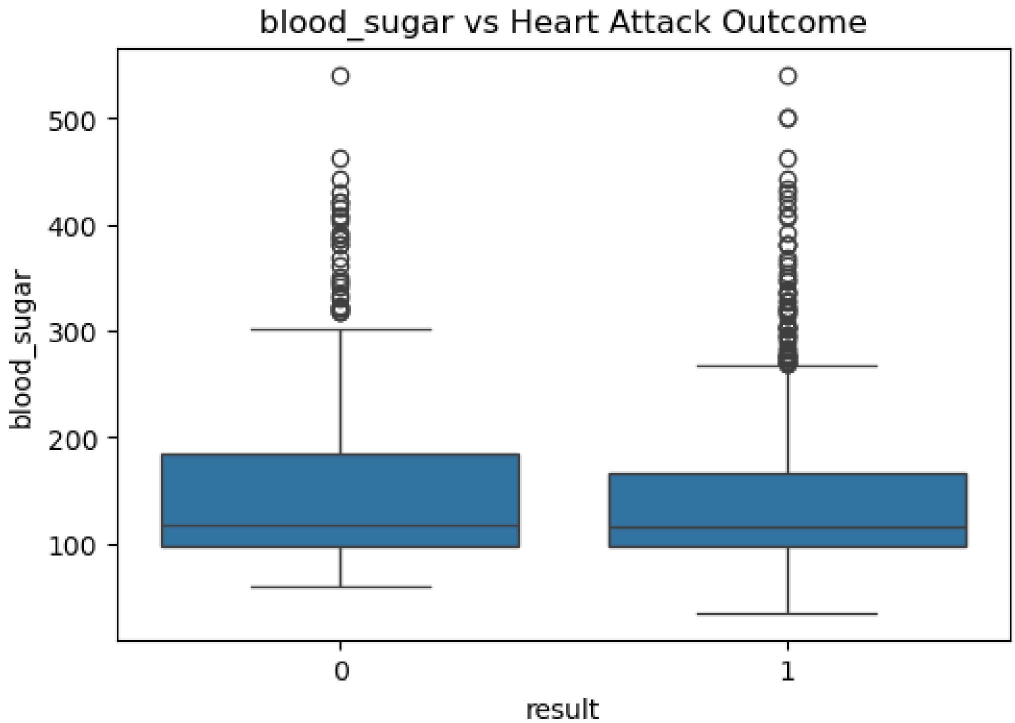

heart_rate vs Heart Attack Outcome
heart_rate
1000
800
600
400
200
0
0
1
result

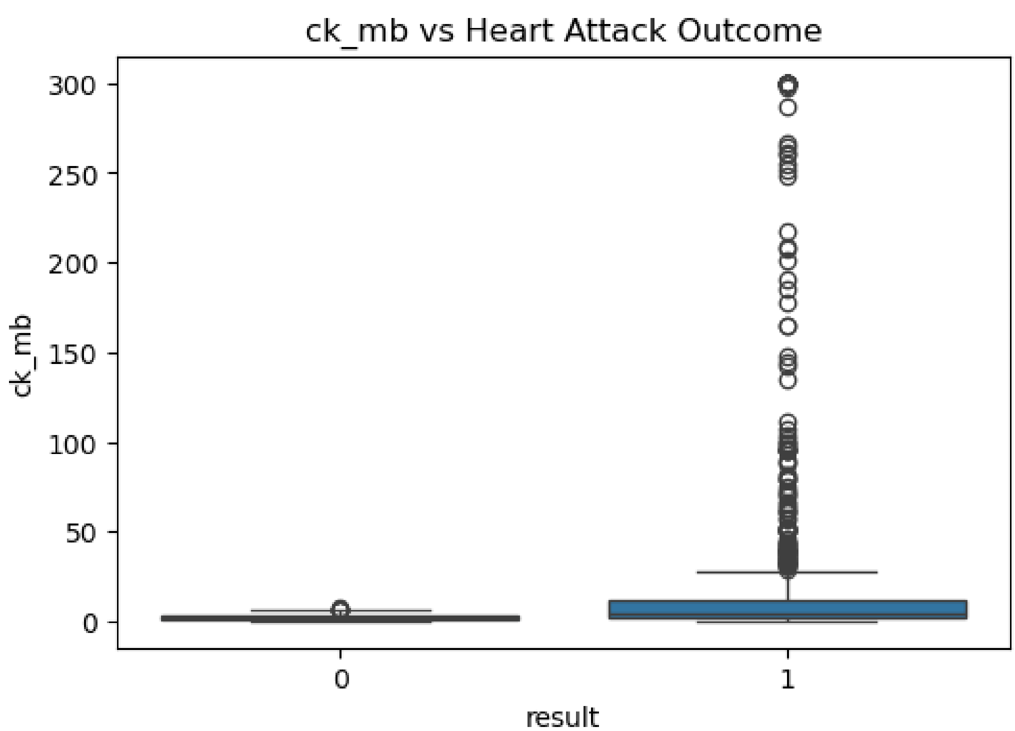


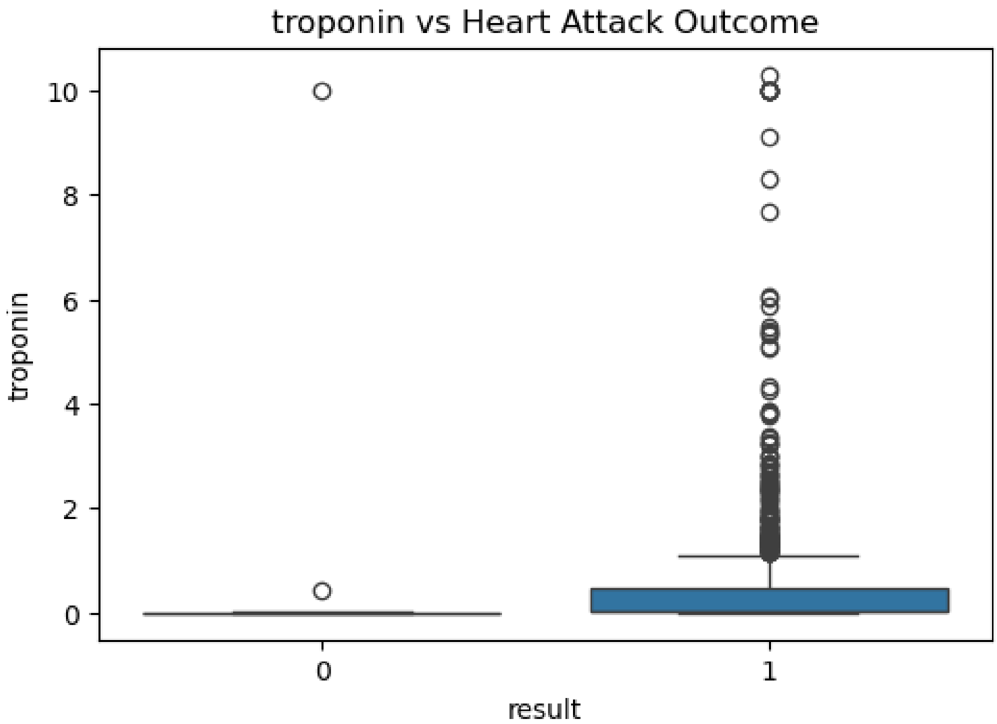
systolic_blood_pressure vs Heart Attack Outcome
225
200
175
150
125
100
75
50
systolic_blood_pressure
0
1
result



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

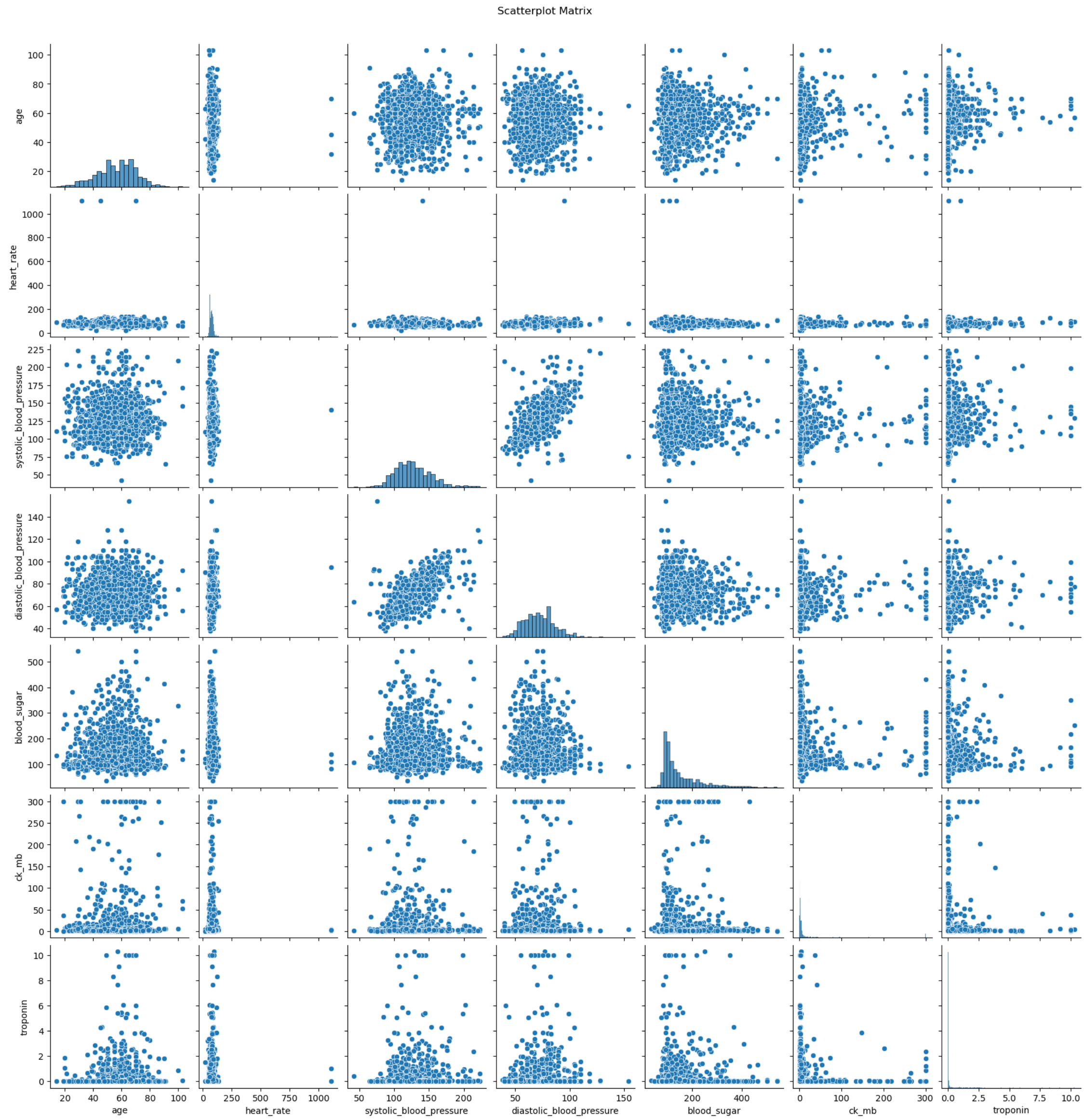

blood_sugar vs Heart Attack Outcome
500
400
300
200
100
blood_sugar
0
1
result


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



1.3.8 Bivariate Relationships (Scatterplots)

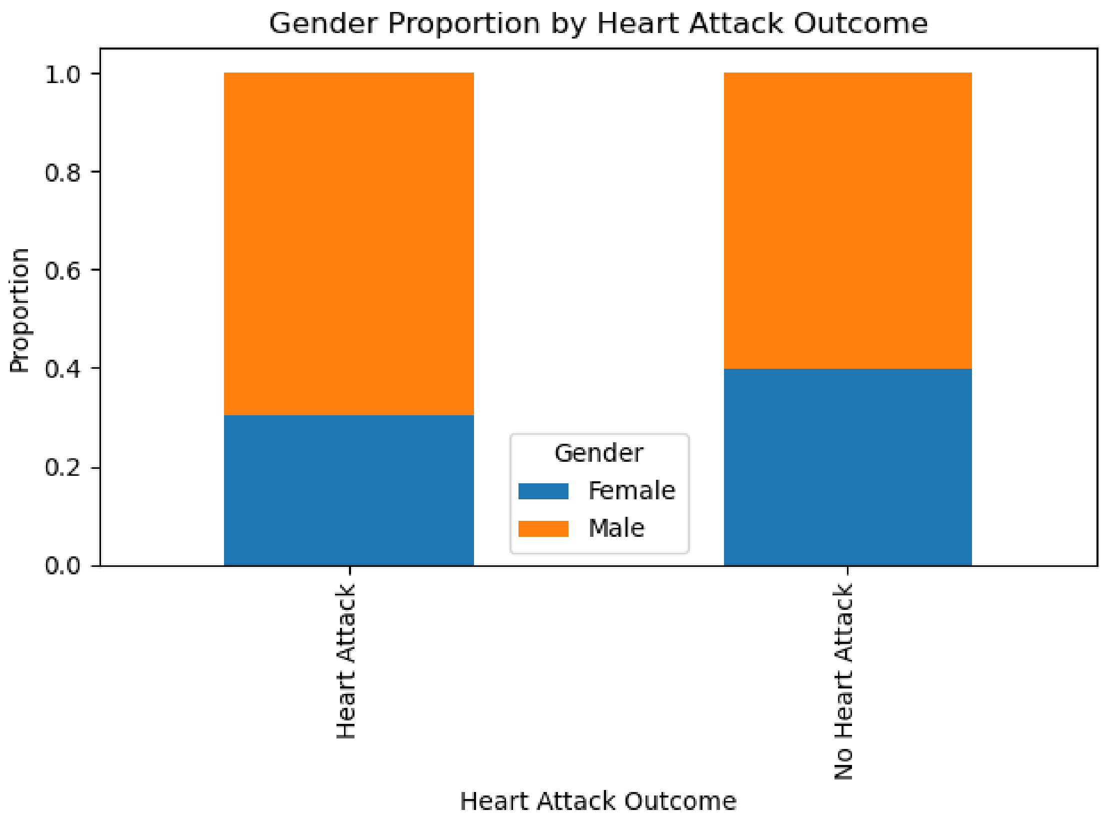
[21]:
```
sns.pairplot(numeric_col)
plt.suptitle("Scatterplot Matrix", y=1.02)
plt.show()
```



This scatterplot matrix illustrates pairwise relationships among clinical variables related to heart health. A strong positive correlation is observed between systolic and diastolic blood pressure, as expected. Age shows a moderate spread across most variables, with a visible increase in CK-MB and troponin levels among certain older individuals. CK-MB and troponin are notably clustered and skewed, indicating their potential diagnostic value in detecting cardiac events. In contrast, heart rate and blood sugar show little to no clear relationship with other variables.

1.3.9 Cross Tabulation and Stacked Bar Charts

[22]:
```
# 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 Attack'})
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()
```



1.3.10 Chi-square Tests for Categorical Associations

[23]:
```
# Load required module
from scipy.stats import chi2_contingency

# 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:.
↪4f}")
```

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

The Chi-square test revealed a statistically significant association between gender and heart attack outcome (p = 0.0008), suggesting that gender may influence the likelihood of experiencing a heart attack.

1.3.11 T-tests (Numeric vs. Categorical)

[24]:
```
# 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)  #␣
↪Welch's t-test
    ttest_results.append((col, p_val))

ttest_results
```

[24]:
```
[('age', 6.641901387727019e-18),
 ('heart_rate', 0.7969260853088802),
 ('systolic_blood_pressure', 0.45574825157843557),
```

```
 ('diastolic_blood_pressure', 0.7279941855792156),
 ('blood_sugar', 0.23842221940845199),
 ('ck_mb', 4.140673523845025e-23),
 ('troponin', 7.200656876108758e-24)]
```

T-test results indicate that age, CK-MB, and troponin levels are statistically significant ($p < 0.05$), suggesting they are strongly associated with the outcome under investigation. In contrast, heart rate, blood pressure (both systolic and diastolic), and blood sugar levels did not show statistically significant differences between groups.

1.3.12 Pre processing

Defining the X and y features

[25]:
```
X = df.drop(columns = ["result"])
y = df["result"]
```

Splitting data into training and test set

[26]:
```
# 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_state = 42, test_size = 0.2)
```

[27]:
```
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

[28]:
```
# 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)
```

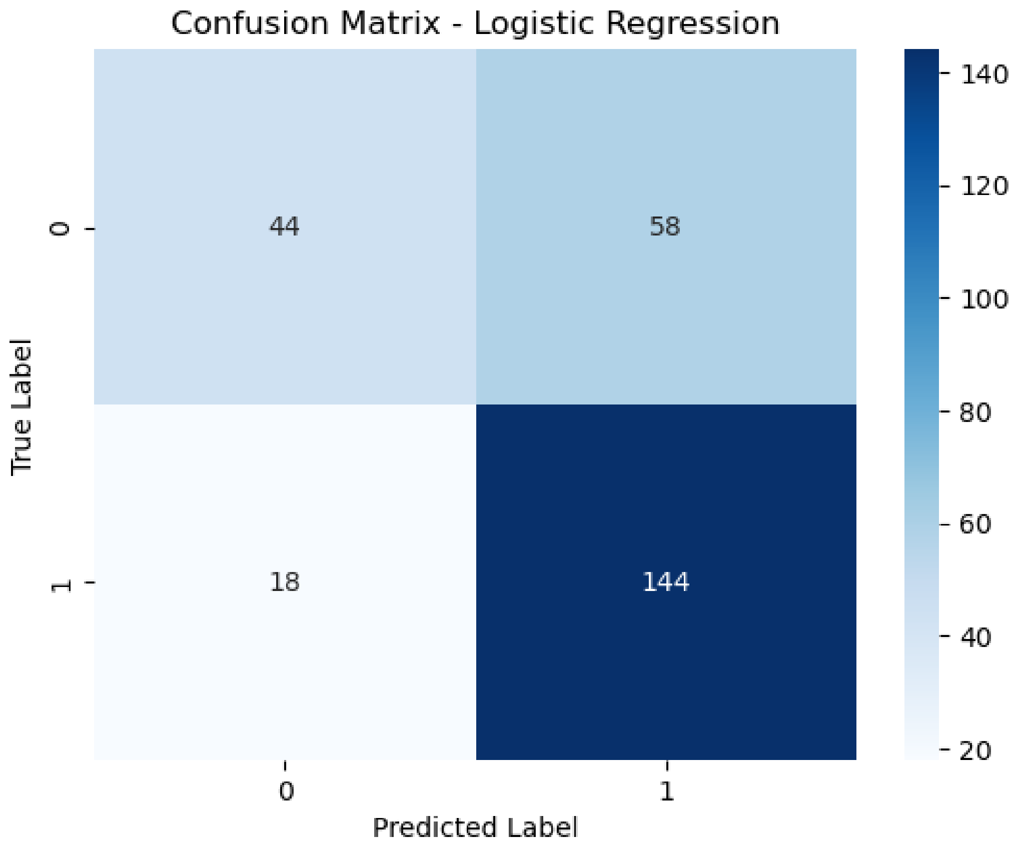
1.3.13 Model Training

1.3.14 1. Logistic Regression

[29]:
```python
## Load required modules
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, f1_score,␣
↪precision_score, recall_score, accuracy_score
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
```



1.3.15 Decision Trees

[30]:
```
# Load required modules
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, f1_score, precision_score,␣
↪recall_score, accuracy_score
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,␣
↪scoring='f1_macro')
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
```

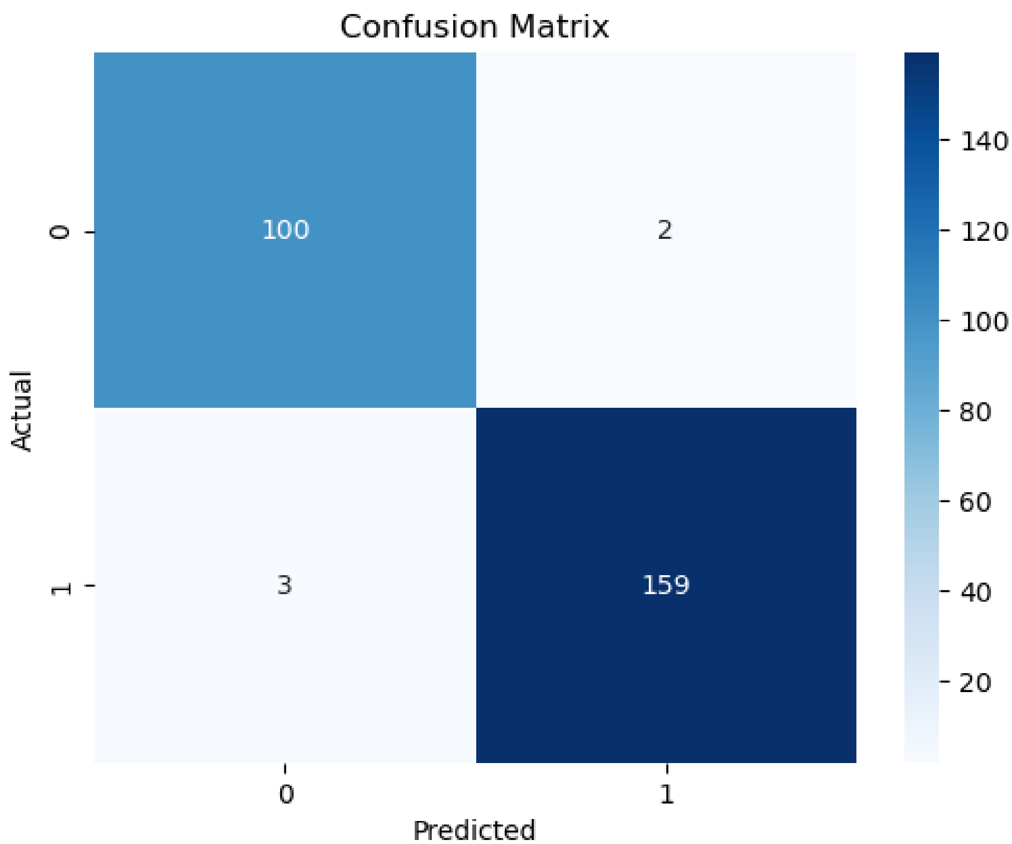
1.3.16 2. Random Forest

[31]:
```
## Import libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, f1_score, precision_score, recall_score
import seaborn as sns
```

```python
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
```

```python
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_leaf': 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: 0.980064939968285
Precision: 0.9792257130796599
Recall: 0.9809368191721133
```



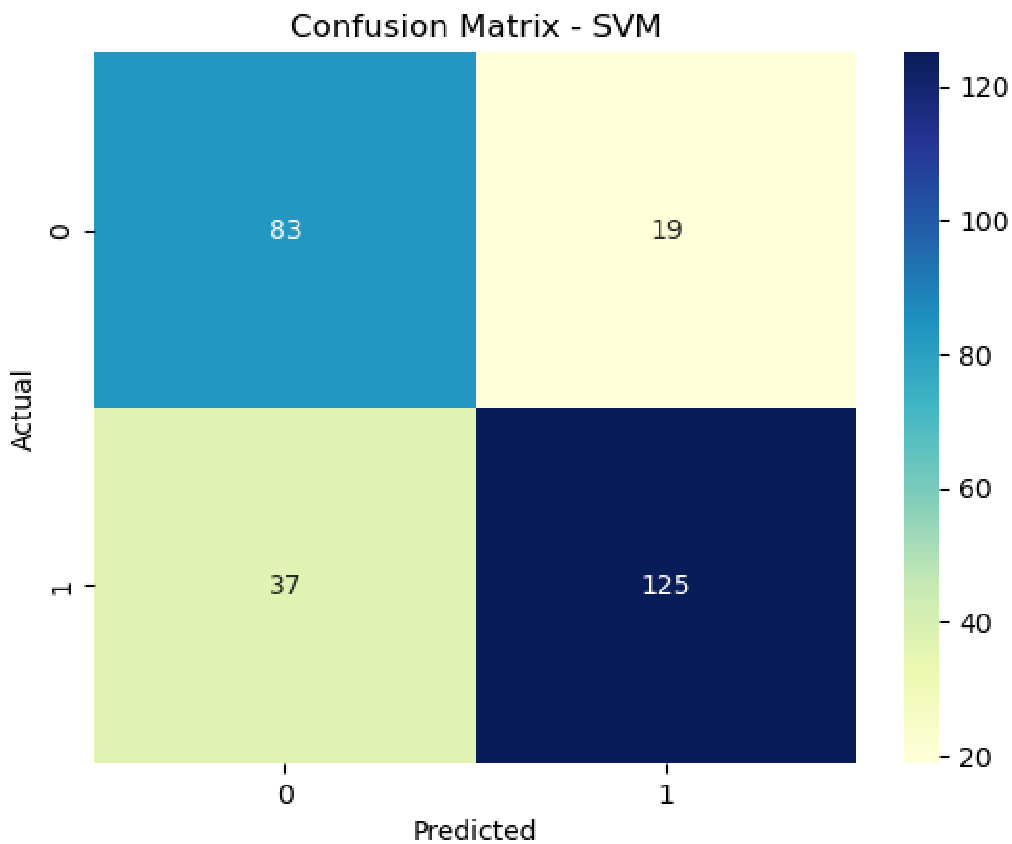
1.3.17 Support Vector Machines

[32]:
```
## 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_score)

## 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],
```

```python
    '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,  # 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))

## confusion 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',
'degree': 3, 'coef0': 0.0, 'C': 10}
Accuracy: 0.7878787878787878
F1 Score (macro): 0.7823706059000176
Precision (macro): 0.7798611111111111
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
```



1.3.18 XGBoost Classifier

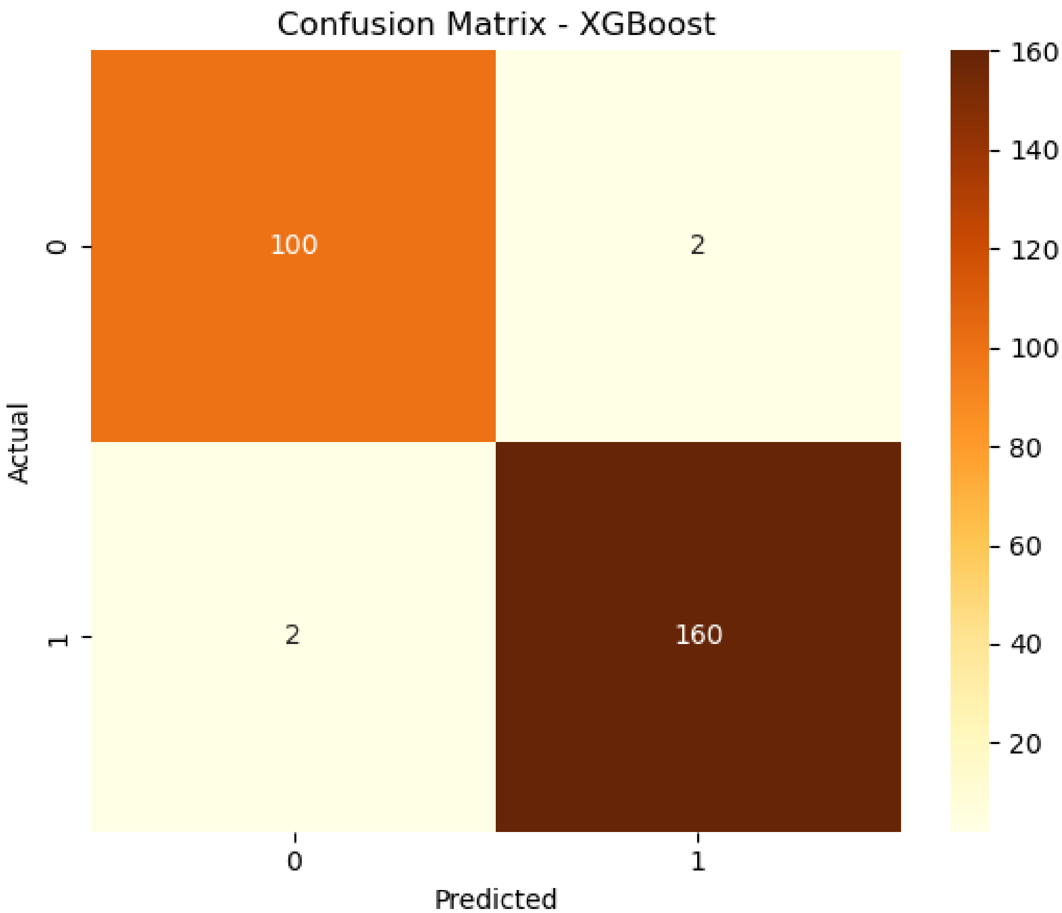
[33]:
```
# Load required modules
from xgboost import XGBClassifier, plot_importance
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import (classification_report, confusion_matrix,
                             f1_score, precision_score, recall_score,␣
 ↪accuracy_score, roc_auc_score, roc_curve)
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],
```

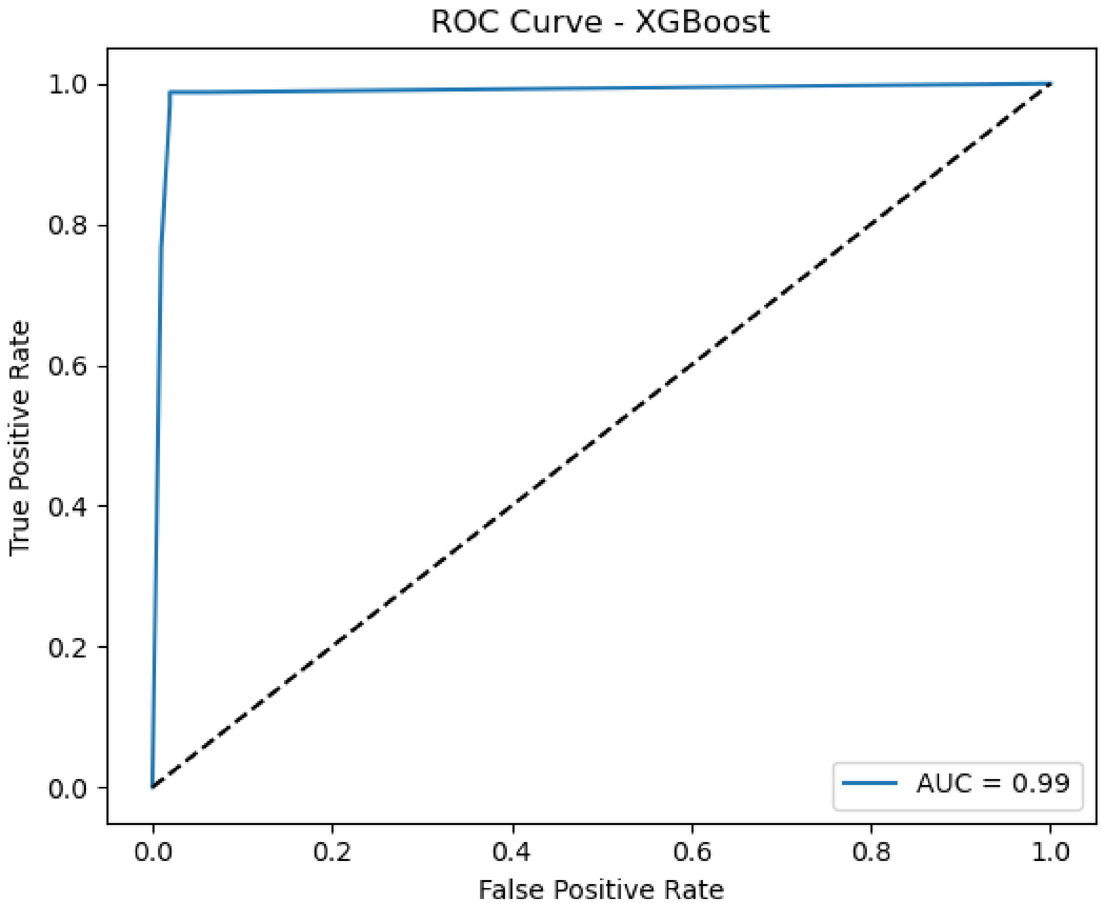
```
    'scale_pos_weight': [1]
}

# 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=42)

# 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))
```

```python
sns.heatmap(confusion_matrix(y_test, xgb_pred), annot=True, fmt='d',␣
 ↪cmap='YlOrBr')
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': 1.0, 'gamma': 0, 'learning_rate':
0.01, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 100,
'scale_pos_weight': 1, 'subsample': 1.0}

 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:
 [[0.14388084 0.85611916]
 [0.7696514  0.23034856]
 [0.14911664 0.85088336]
 [0.7696514  0.23034856]
```

```
[0.7696514  0.23034856]
[0.14911664 0.85088336]
[0.14911664 0.85088336]
[0.14388084 0.85611916]
[0.14388084 0.85611916]
[0.14388084 0.85611916]]
```





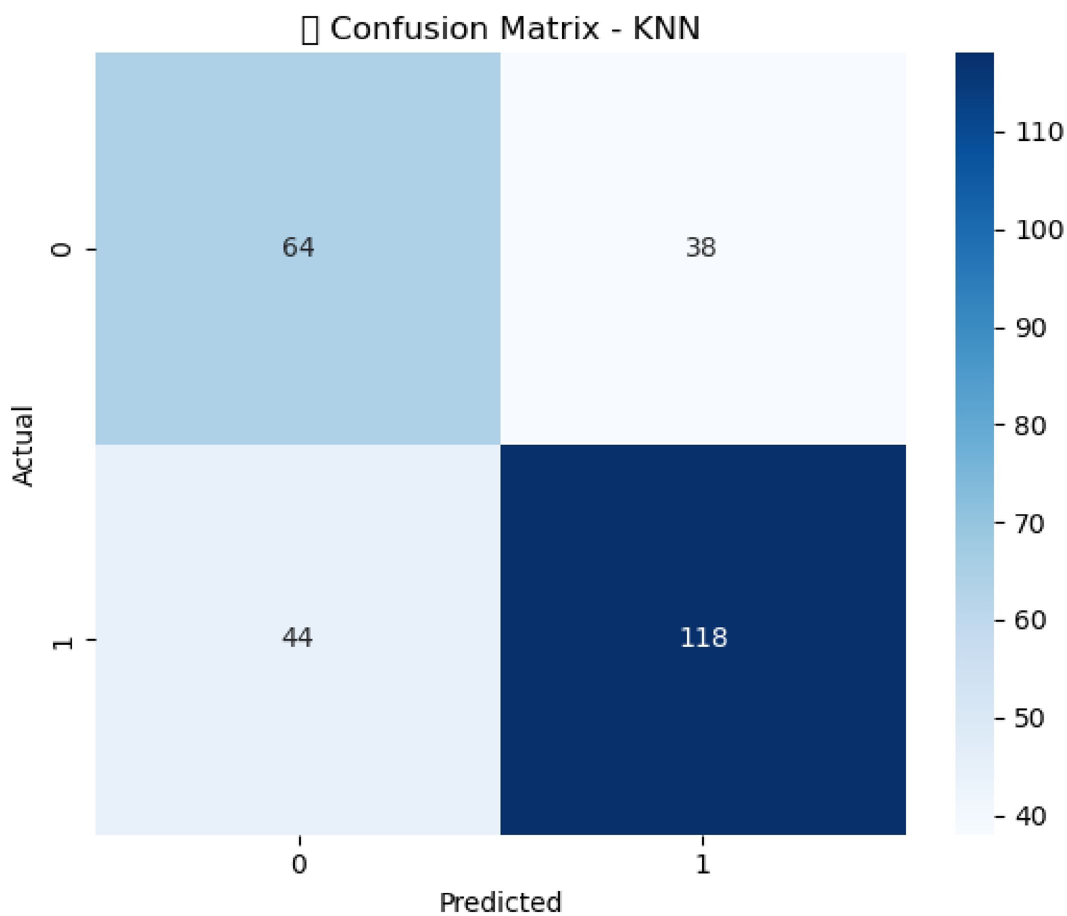
1.3.19 K Nearest Neighbors

[34]:
```python
# 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
```

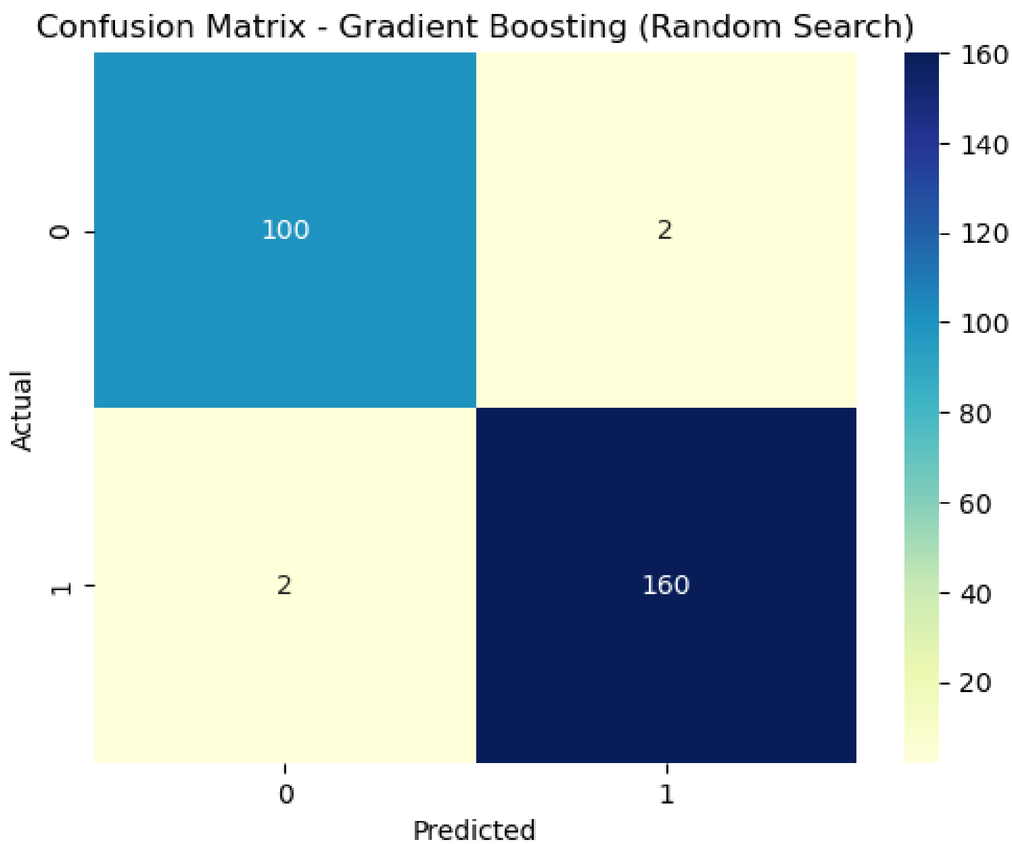


1.3.20 Gradient Boosting Machines

[35]:
```python
# Import required modules
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV, RepeatedStratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, f1_score, precision_score, recall_score
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␣
↪random combinations
                                   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)
```

```python
# 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_leaf': 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
```



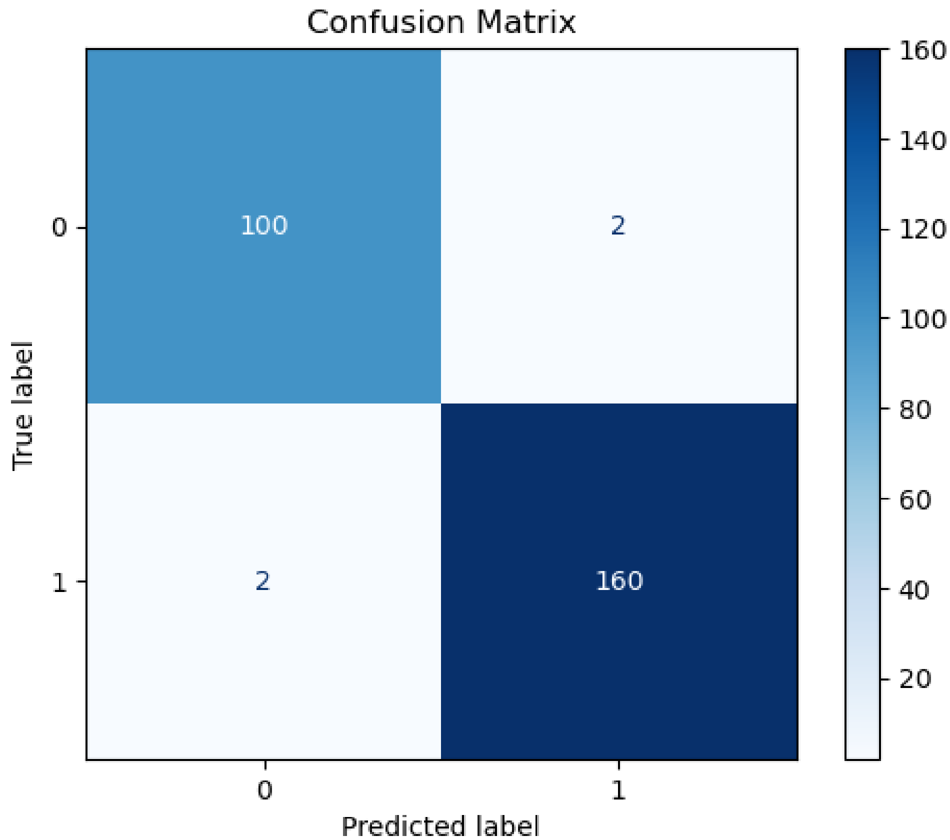
1.3.21 Ada Boost Classifier

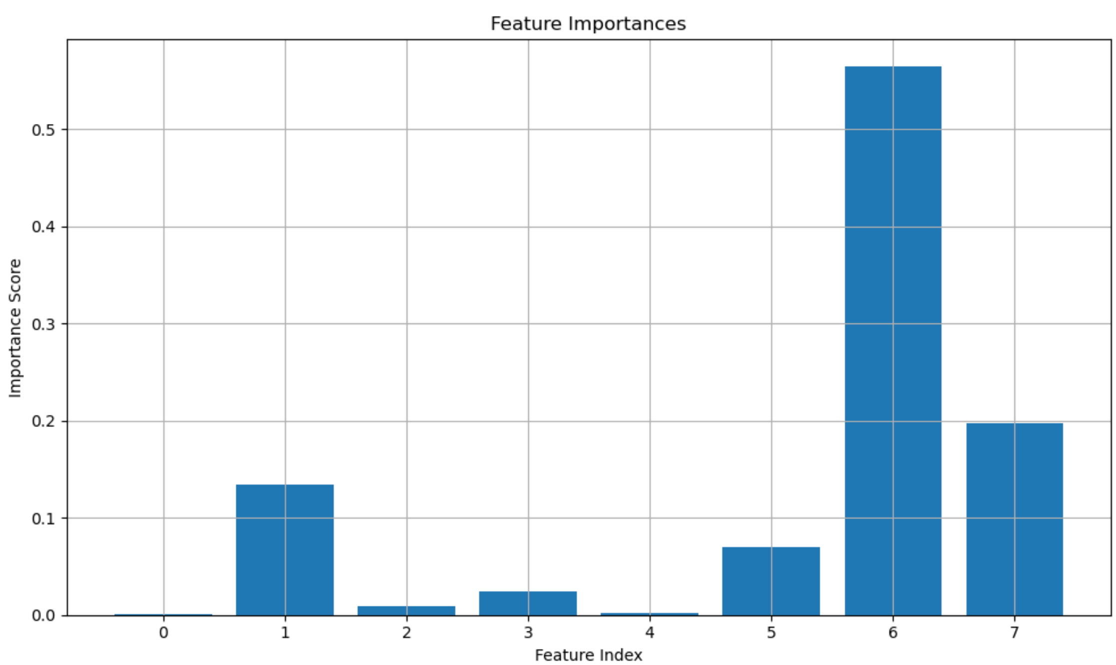
[36]:
```
# Import required modules
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    f1_score,
    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_rate=1.0)

# 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()
```

```python
# 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
```


Confusion Matrix
True label
Predicted label
0
1
100
2
2
160
160
140
120
100
80
60
40
20

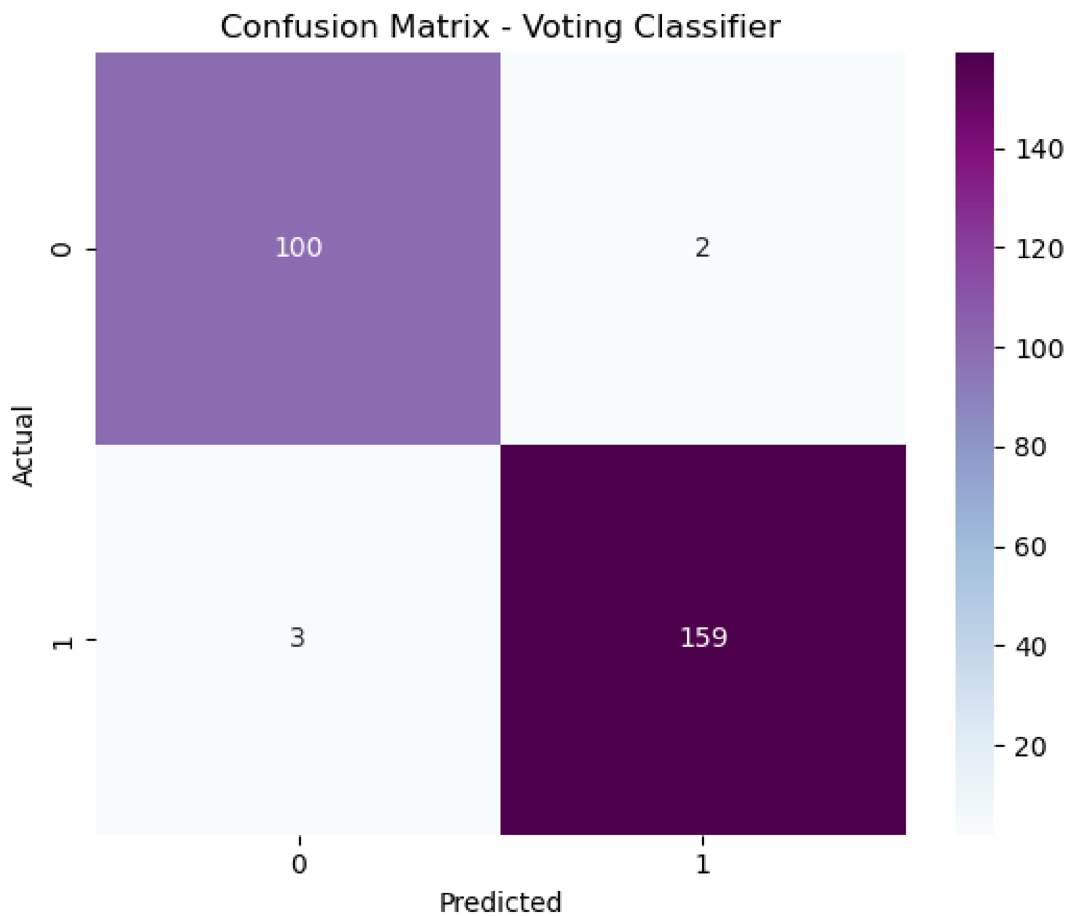

Feature Importances
Importance Score
Feature Index
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7

1.3.22 Voting Classifier

[37]:
```
# Load required libraries
from sklearn.ensemble import VotingClassifier, RandomForestClassifier,␣
↪AdaBoostClassifier
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'))
```

```python
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: 0.980064939968285
Precision: 0.9792257130796599
Recall: 0.9809368191721133
```



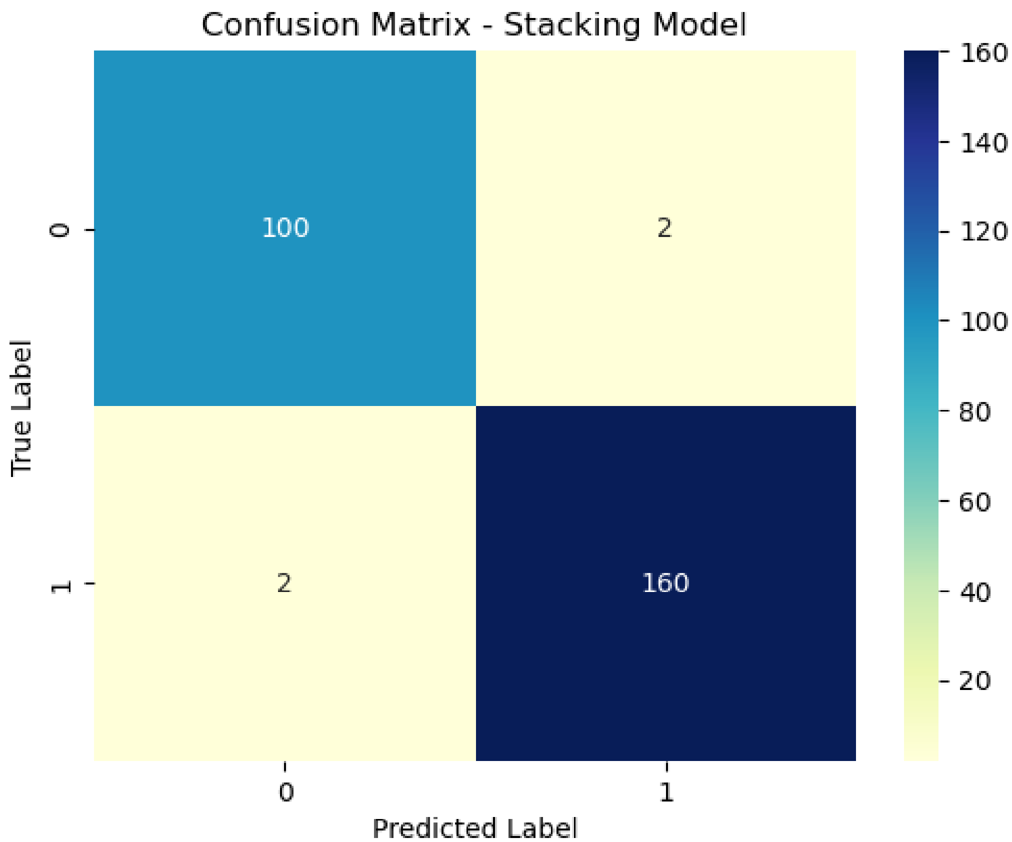
1.3.23 Stacking Classifier

[38]:
```
# 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_matrix, f1_score, precision_score, recall_score

# 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␣
↪needs probability=True for stacking
```

```
    ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss',␣
↪random_state=42)),
    ('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
)

# 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
```



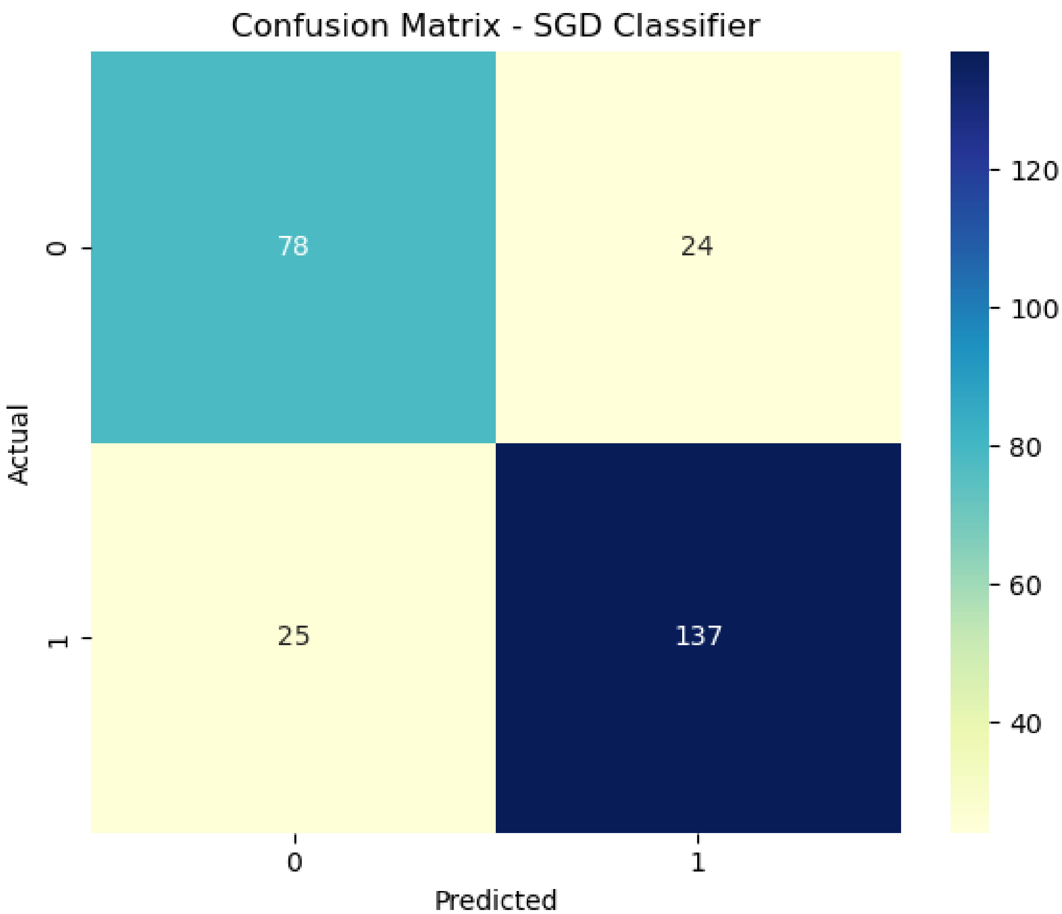
1.3.24 Stochastic Gradient Descent

[39]:
```python
# 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
```

```python
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␣
 ↪imbalanced data
                           n_jobs=-1,
                           verbose=1)

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

# Make Predictions
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
```



1.3.25 Model Comparison

[40]:
```
models = pd.DataFrame({
    'Model': ['Logistic Regression', "Decision Trees", 'Random Forest␣
↪Classifier', 'SVM', "XG Boost", 'KNN', "Gradient Boosting", "Ada Boost␣
↪Classifier", 'Voting Classifier', 'Stacking Classifier',
              'Stachastic Gradient Boosting'],
    'Score': [log_score, dt_score, rf_score, svm_score, xgb_score, knn_score,␣
↪gbm_score, ada_score, vote_score, stack_score, sgd_score]
})

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

[40]:

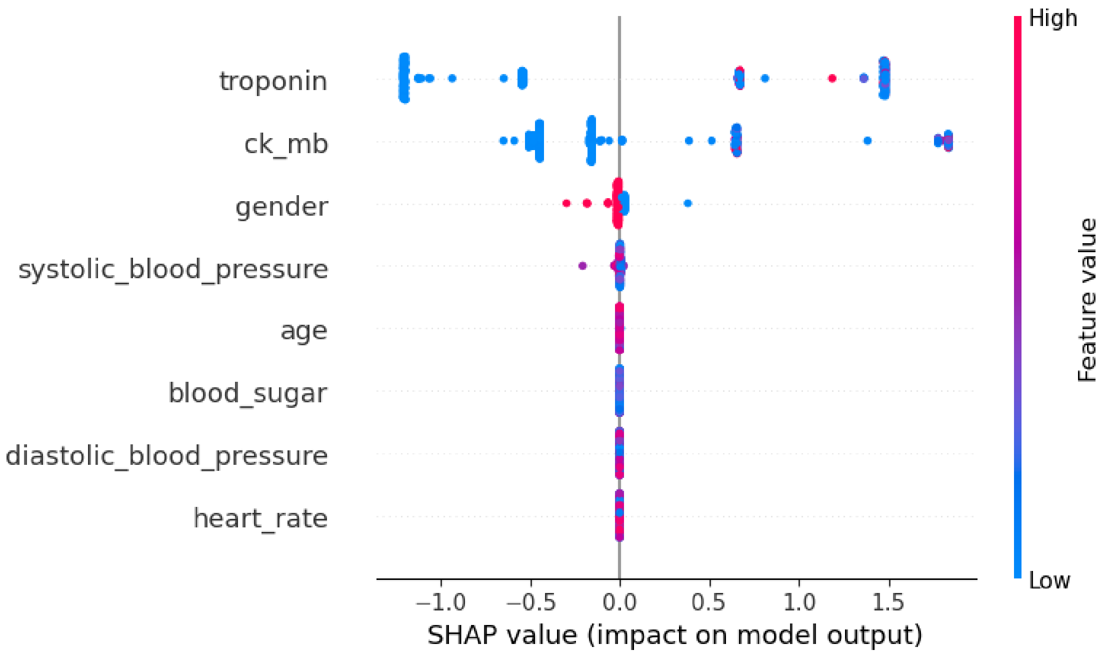
	Model	Score
4	XG Boost	0.984848
6	Gradient Boosting	0.984848
7	Ada Boost Classifier	0.984848
9	Stacking Classifier	0.984848
1	Decision Trees	0.981061
2	Random Forest Classifier	0.981061
8	Voting Classifier	0.981061
10	Stachastic Gradient Boosting	0.814394
3	SVM	0.787879
0	Logistic Regression	0.712121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

Best model - XGBoost

1.3.26 SHAP Values for Deeper Interpretability

[41]:
```
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)
```

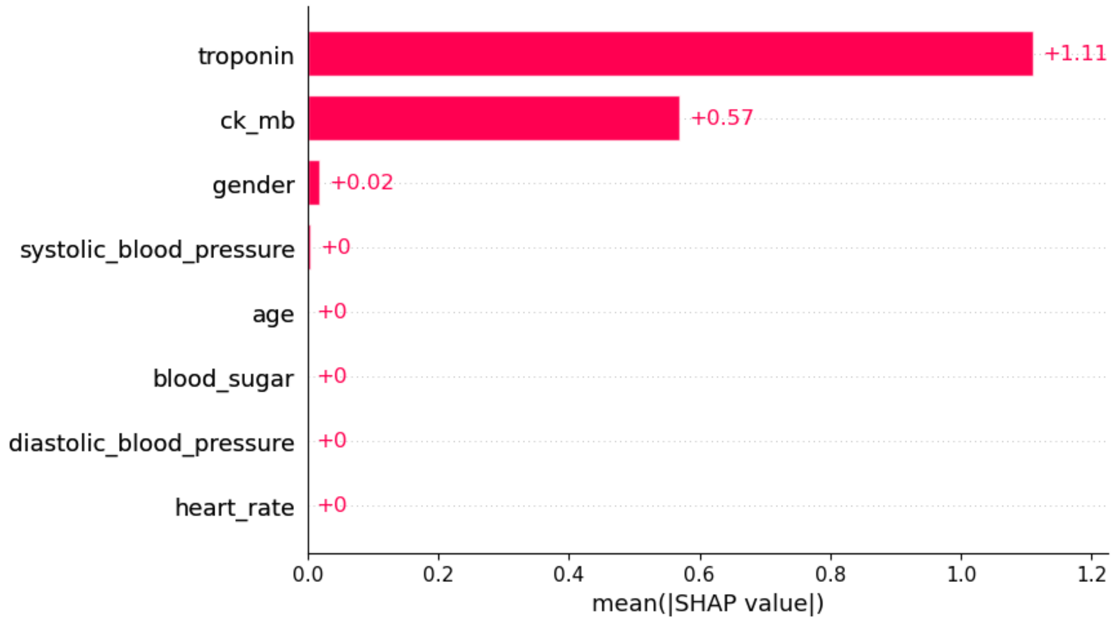
```
# Plot SHAP summary (global feature importance)
shap.summary_plot(shap_values, X_test, feature_names=feature_names)
```



1.3.27 Bar Chart

[42]:
```
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
```

```python
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)
```



[43]:
```python
# Save the best model to a file
import pickle
with open('heart_attack_xgb_model.pkl', 'wb') as f:
    pickle.dump(best_xgb, f)
```