```
import zipfile
import os

# Define file paths
zip_file_path = "/content/heart+disease.zip"  # Ensure this matches the actual file name
extract_path = "/content/heart_disease/"  # Use a directory to extract

# Ensure the ZIP file exists
if not os.path.exists(zip_file_path):
    raise FileNotFoundError(f"ZIP file not found at {zip_file_path}")

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# List extracted files
extracted_files = os.listdir(extract_path)
print("Extracted Files:", extracted_files)
import pandas as pd
import os

# Define the dataset file path (update this)
extract_path = "/content/heart_disease/"  # Update to your actual extracted directory
csv_filename = "processed.cleveland.data"
file_path = os.path.join(extract_path, csv_filename)

# Check if the file exists
if os.path.exists(file_path):
    # Define column names (from UCI Heart Disease dataset)
    column_names = [
        "age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach",
        "exang", "oldpeak", "slope", "ca", "thal", "target"
    ]

    # Load dataset, handling missing values
    df = pd.read_csv(file_path, header=None, names=column_names, na_values="?")
```

```
    # Display dataset info
    print(df.info())  # Summary of the dataset
    print(df.head())  # Show first few rows

else:
    print(f"Error: File not found at {file_path}")
import pandas as pd

# Rename columns based on dataset documentation
df.columns = [
    "age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalach",
    "exang", "oldpeak", "slope", "ca", "thal", "num"
]

# Replace "?" with NaN (for missing values)
df.replace("?", pd.NA, inplace=True)

# Convert all columns to numeric
df = df.apply(pd.to_numeric)

# Drop rows with missing values
df.dropna(inplace=True)

# Reset index after dropping missing values
df.reset_index(drop=True, inplace=True)

# Convert target variable (num) into binary (0 = No Disease, 1+ = Disease)
df["num"] = df["num"].apply(lambda x: 1 if x > 0 else 0)

# Split features (X) and target (Y)
X = df.drop(columns=["num"])
Y = df["num"]

# Display the final dataset shape
print("Dataset shape:", df.shape)
print(df.head())
```

```python
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, stratify=Y, random_state=42
)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Train the model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, Y_train)
# Evaluate model
train_accuracy = accuracy_score(Y_train, model.predict(X_train))
test_accuracy = accuracy_score(Y_test, model.predict(X_test))

print(f"✅ Model Training Completed")
print(f"🎯 Training Accuracy: {train_accuracy:.2f}")
print(f"🎯 Testing Accuracy: {test_accuracy:.2f}")
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=500, max_depth=15, random_state=42)
rf_model.fit(X_train, Y_train)

rf_test_accuracy = accuracy_score(Y_test, rf_model.predict(X_test))
print(f"✅ Random Forest Accuracy: {rf_test_accuracy:.2f}")

from xgboost import XGBClassifier

xgb_model = XGBClassifier(n_estimators=500, learning_rate=0.05, max_depth=10, random_state=42)
xgb_model.fit(X_train, Y_train)

xgb_test_accuracy = accuracy_score(Y_test, xgb_model.predict(X_test))
```

```python
print(f"✅ XGBoost Accuracy: {xgb_test_accuracy:.2f}")

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, matthews_corrc

# Function to print confusion matrix and evaluation metrics
def evaluate_model(model, X_test, Y_test, model_name):
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else None

    cm = confusion_matrix(Y_test, y_pred)
    accuracy = accuracy_score(Y_test, y_pred)
    precision = precision_score(Y_test, y_pred)
    recall = recall_score(Y_test, y_pred)
    f1 = f1_score(Y_test, y_pred)
    specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])  # TNR = TN / (TN + FP)
    fpr = cm[0, 1] / (cm[0, 0] + cm[0, 1])  # FPR = FP / (TN + FP)
    fnr = cm[1, 0] / (cm[1, 0] + cm[1, 1])  # FNR = FN / (FN + TP)
    mcc = matthews_corrcoef(Y_test, y_pred)
    roc_auc = roc_auc_score(Y_test, y_pred_proba) if y_pred_proba is not None else "N/A"

    print(f"\n Confusion Matrix for {model_name}:\n", cm)
    print(f" Accuracy: {accuracy:.4f}")
    print(f" Precision (PPV): {precision:.4f}")
    print(f" Recall (Sensitivity): {recall:.4f}")
    print(f"  F1-Score: {f1:.4f}")
    print(f" Specificity (TNR): {specificity:.4f}")
    print(f" False Positive Rate (FPR): {fpr:.4f}")
    print(f" False Negative Rate (FNR): {fnr:.4f}")
    print(f" Matthews Correlation Coefficient (MCC): {mcc:.4f}")
    print(f" ROC-AUC Score: {roc_auc}")

# Evaluate all models
evaluate_model(model, X_test, Y_test, "Logistic Regression")
evaluate_model(rf_model, X_test, Y_test, "Random Forest")
evaluate_model(xgb_model, X_test, Y_test, "XGBoost")
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Function to plot confusion matrix
def plot_confusion_matrix(model, X_test, Y_test, model_name):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(Y_test, y_pred)

    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease", "Disease"], yticklabels=["No Disease", "Disease"])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix for {model_name}")
    plt.show()

# Plot confusion matrices for all models
plot_confusion_matrix(model, X_test, Y_test, "Logistic Regression")
plot_confusion_matrix(rf_model, X_test, Y_test, "Random Forest")
plot_confusion_matrix(xgb_model, X_test, Y_test, "XGBoost")

from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Function to evaluate regression metrics
def evaluate_regression_metrics(model, X_test, Y_test, model_name):
    y_pred = model.predict(X_test)

    mse = mean_squared_error(Y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(Y_test, y_pred)

    print(f"\n Regression Metrics for {model_name}:")
    print(f" Mean Squared Error (MSE): {mse:.4f}")
    print(f" Root Mean Squared Error (RMSE): {rmse:.4f}")
    print(f" R-Squared (R²): {r2:.4f}")
```

```
# Evaluate all models
evaluate_regression_metrics(model, X_test, Y_test, "Logistic Regression")
evaluate_regression_metrics(rf_model, X_test, Y_test, "Random Forest")
evaluate_regression_metrics(xgb_model, X_test, Y_test, "XGBoost")

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Regression Metrics Data
models = ["Logistic Regression", "Random Forest", "XGBoost"]
mse_values = [0.1667, 0.1333, 0.1500]
rmse_values = [0.4082, 0.3651, 0.3873]
r2_values = [0.3304, 0.4643, 0.3973]

# Set plot style
sns.set(style="whitegrid")

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Bar plot for MSE
sns.barplot(x=models, y=mse_values, ax=axes[0], palette="Blues")
axes[0].set_title("Mean Squared Error (MSE)")
axes[0].set_ylabel("MSE")

# Bar plot for RMSE
sns.barplot(x=models, y=rmse_values, ax=axes[1], palette="Greens")
axes[1].set_title("Root Mean Squared Error (RMSE)")
axes[1].set_ylabel("RMSE")

# Bar plot for R²
sns.barplot(x=models, y=r2_values, ax=axes[2], palette="Oranges")
axes[2].set_title("R-Squared (R²)")
axes[2].set_ylabel("R² Score")

# Show plots
```

```
plt.tight_layout()
plt.show()

import numpy as np
import matplotlib.pyplot as plt

feature_names = X.columns
importances = rf_model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 5))
plt.title("Feature Importance - Random Forest")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), feature_names[indices], rotation=90)
plt.show()
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, interaction_only=True)
X_poly = poly.fit_transform(X)

model = LogisticRegression(max_iter=2000)
model = LogisticRegression(C=0.5)
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=200, max_depth=10, min_samples_split=5, min_samples_leaf=3, random_state=42)
rf_model.fit(X_train, Y_train)
from sklearn.model_selection import GridSearchCV
import xgboost as xgb

param_grid = {
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.05, 0.1],
    "n_estimators": [100, 200, 300]
```

```
}
xgb_model = xgb.XGBClassifier()
grid_search = GridSearchCV(xgb_model, param_grid, cv=5)
grid_search.fit(X_train, Y_train)

print(grid_search.best_params_)

from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, Y_resampled = smote.fit_resample(X, Y)

from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, Y_train, epochs=50, batch_size=8, validation_split=0.2)

from xgboost import XGBClassifier

# Re-initialize and train the XGBoost model
xgb_model = XGBClassifier(n_estimators=500, learning_rate=0.05, max_depth=10, random_state=42)
xgb_model.fit(X_train, Y_train)

# Verify training completion
print("✅ XGBoost model trained successfully!")
from sklearn.exceptions import NotFittedError

# Function to predict heart disease
def predict_heart_disease(model, scaler, user_input):
    try:
```

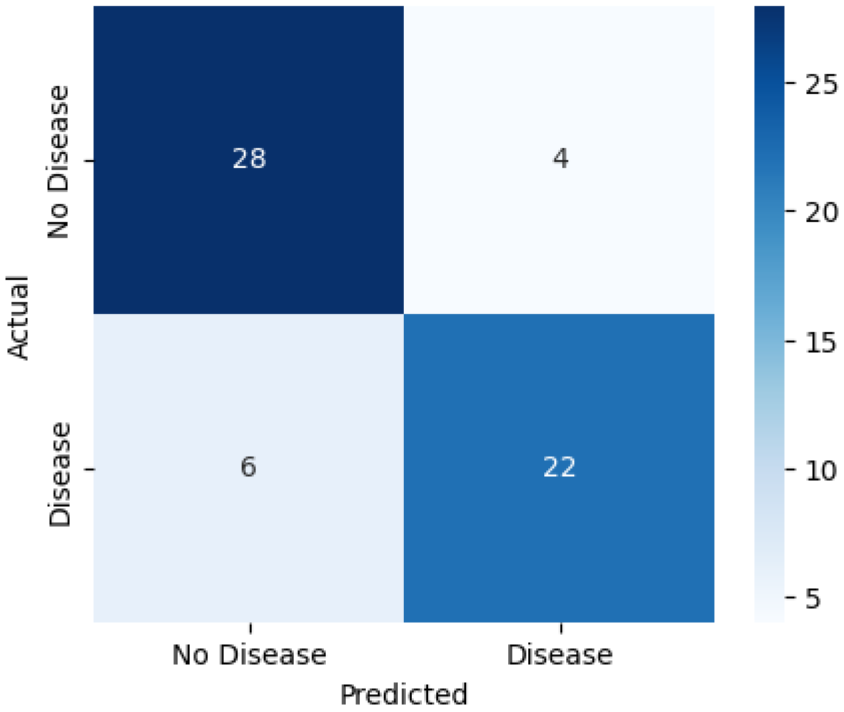
```
        # Convert input into a NumPy array and scale it
        user_input_scaled = scaler.transform([user_input])  # Ensure correct shape

        # Make prediction
        prediction = model.predict(user_input_scaled)[0]

        # Return result based on prediction
        return "🟢 The person is unlikely to have heart disease." if prediction == 0 else "🔴 The person is likely to have heart di

    except NotFittedError:
        return "❌ Error: The model is not trained. Train the model before predicting."

# Take user input
print("Enter patient details (comma or space separated):")
user_input = list(map(float, input().replace(",", " ").split()))

# Test with Logistic Regression
print("\n1. Test with Logistic Regression Model")
print(predict_heart_disease(model, scaler, user_input))

# Test with Random Forest
print("\n2. Test with Random Forest Model")
print(predict_heart_disease(rf_model, scaler, user_input))

# Test with XGBoost
print("\n3. Test with XGBoost Model")
print(predict_heart_disease(xgb_model, scaler, user_input))
```

```
Extracted Files: ['heart-disease.names', 'cleveland.data', 'bak', 'processed.hungarian.data', 'hungarian.data', 'switzerland.dat
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    float64
 1   sex       303 non-null    float64
 2   cp        303 non-null    float64
 3   trestbps  303 non-null    float64
 4   chol      303 non-null    float64
 5   fbs       303 non-null    float64
 6   restecg   303 non-null    float64
 7   thalach   303 non-null    float64
 8   exang     303 non-null    float64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    float64
 11  ca        299 non-null    float64
 12  thal      301 non-null    float64
 13  target    303 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 33.3 KB
None
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3   
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5   
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6   
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5   
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4   

   slope   ca  thal  target  
0    3.0  0.0   6.0       0  
1    2.0  3.0   3.0       2  
2    2.0  2.0   7.0       1  
3    3.0  0.0   3.0       0  
4    1.0  0.0   3.0       0  
Dataset shape: (297, 14)
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3   
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5   
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6   
```

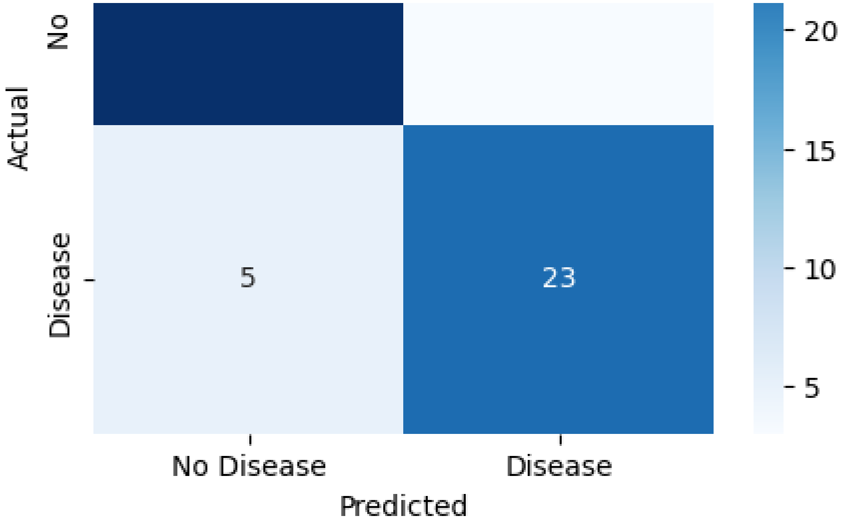
```
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4

   slope   ca  thal  num
0    3.0  0.0   6.0    0
1    2.0  3.0   3.0    1
2    2.0  2.0   7.0    1
3    3.0  0.0   3.0    0
4    1.0  0.0   3.0    0
✅ Model Training Completed
🎯 Training Accuracy: 0.85
🎯 Testing Accuracy: 0.83
✅ Random Forest Accuracy: 0.87
✅ XGBoost Accuracy: 0.85

 Confusion Matrix for Logistic Regression:
 [[28  4]
 [ 6 22]]
 Accuracy: 0.8333
 Precision (PPV): 0.8462
 Recall (Sensitivity): 0.7857
  F1-Score: 0.8148
 Specificity (TNR): 0.8750
 False Positive Rate (FPR): 0.1250
 False Negative Rate (FNR): 0.2143
 Matthews Correlation Coefficient (MCC): 0.6652
 ROC-AUC Score: 0.9497767857142857

 Confusion Matrix for Random Forest:
 [[29  3]
 [ 5 23]]
 Accuracy: 0.8667
 Precision (PPV): 0.8846
 Recall (Sensitivity): 0.8214
  F1-Score: 0.8519
 Specificity (TNR): 0.9062
 False Positive Rate (FPR): 0.0938
 False Negative Rate (FNR): 0.1786
 Matthews Correlation Coefficient (MCC): 0.7326
 ROC-AUC Score: 0.9408482142857143

 Confusion Matrix for XGBoost:
```

```
Confusion Matrix for XGBoost:
[[28  4]
 [ 5 23]]
Accuracy: 0.8500
Precision (PPV): 0.8519
Recall (Sensitivity): 0.8214
 F1-Score: 0.8364
Specificity (TNR): 0.8750
False Positive Rate (FPR): 0.1250
False Negative Rate (FNR): 0.1786
Matthews Correlation Coefficient (MCC): 0.6984
ROC-AUC Score: 0.8995535714285714
```

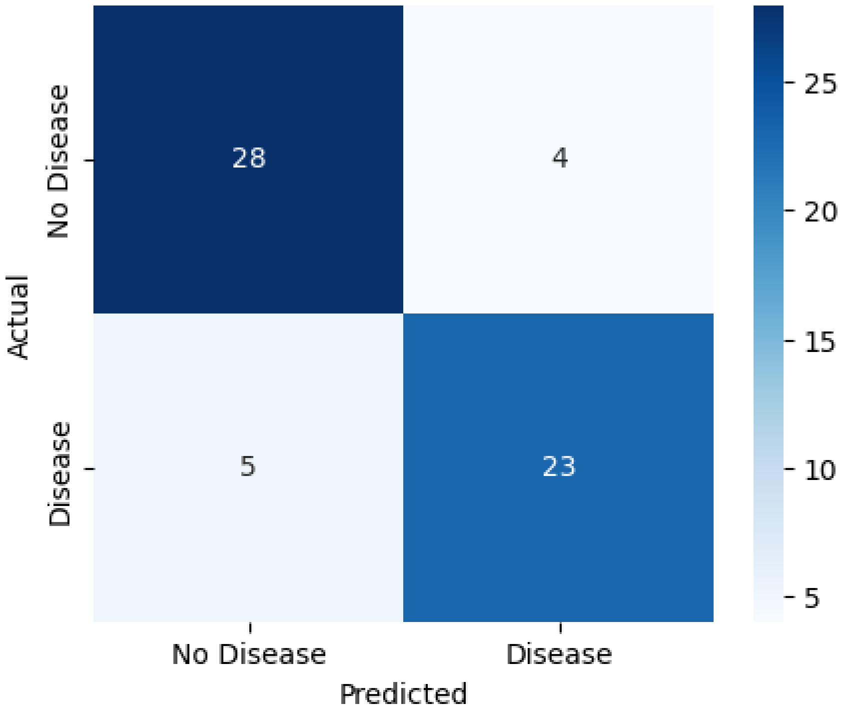
Confusion Matrix for Logistic Regression



Confusion Matrix for Random Forest

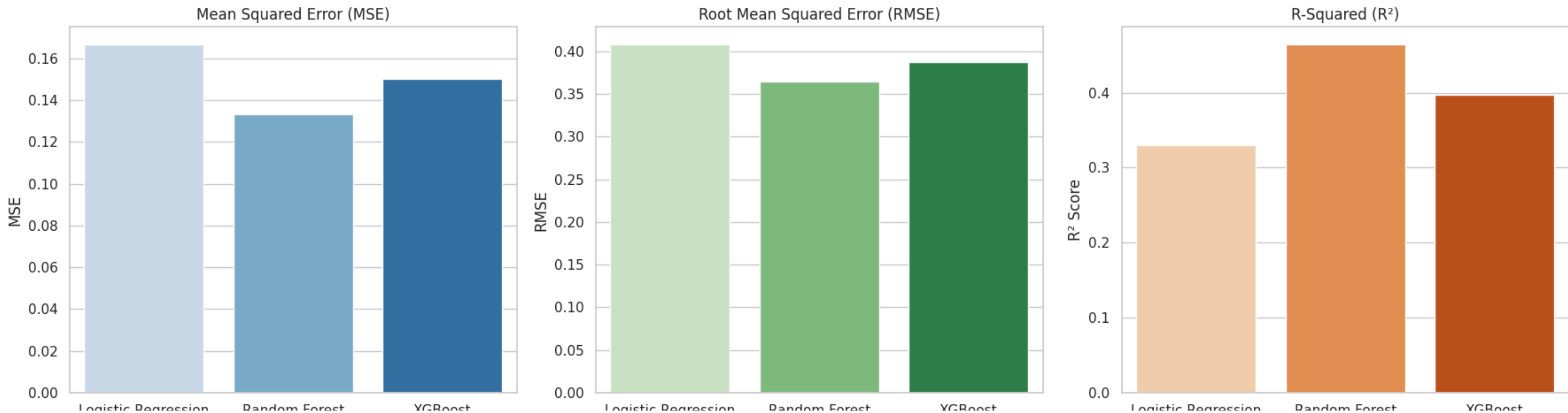


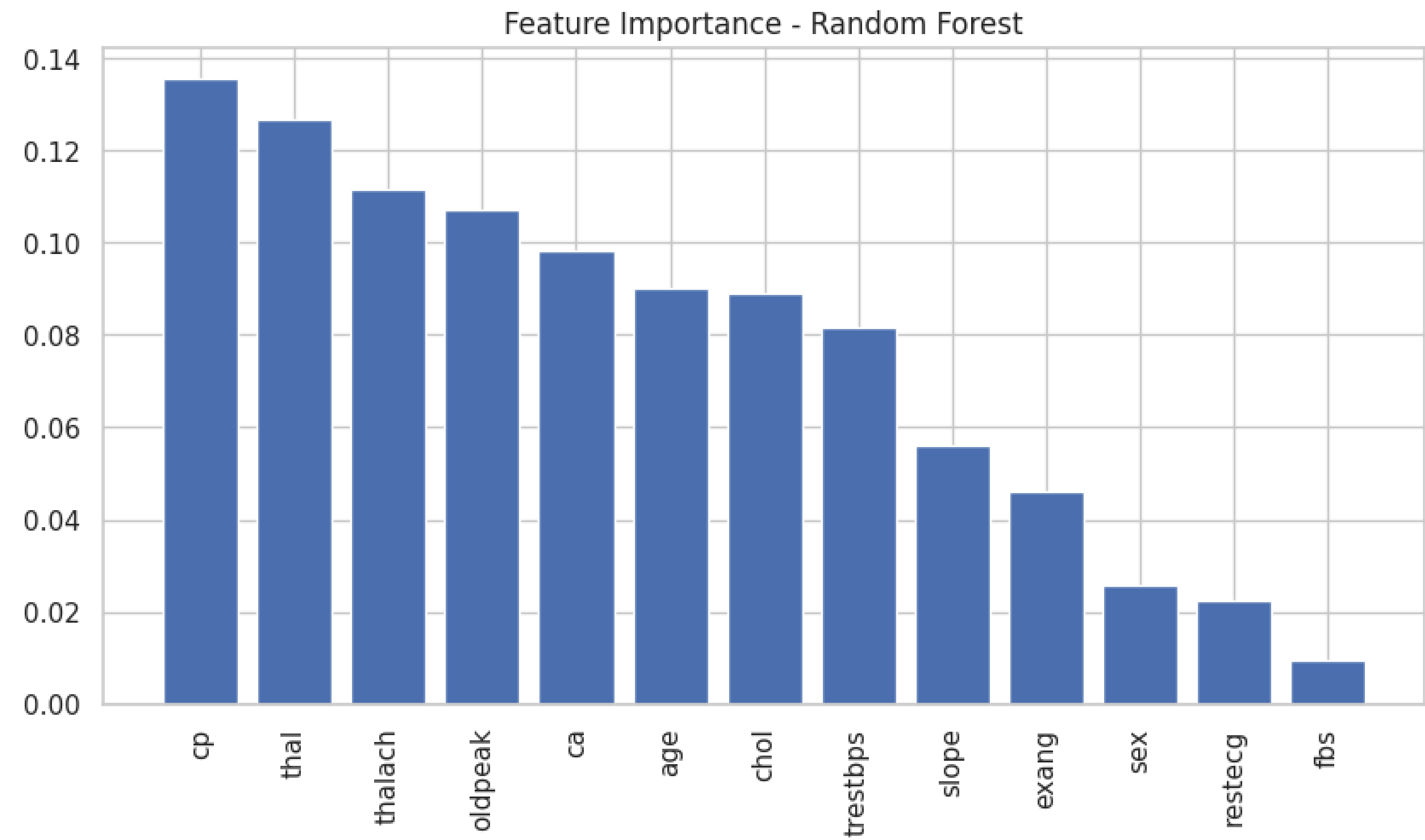
Confusion Matrix for XGBoost




```
Regression Metrics for Logistic Regression:
Mean Squared Error (MSE): 0.1667
```

```
Root Mean Squared Error (RMSE): 0.4082
R-Squared (R²): 0.3304

Regression Metrics for Random Forest:
Mean Squared Error (MSE): 0.1333
Root Mean Squared Error (RMSE): 0.3651
R-Squared (R²): 0.4643

Regression Metrics for XGBoost:
Mean Squared Error (MSE): 0.1500
Root Mean Squared Error (RMSE): 0.3873
R-Squared (R²): 0.3973
<ipython-input-2-90153cdced38>:207: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=models, y=mse_values, ax=axes[0], palette="Blues")
<ipython-input-2-90153cdced38>:212: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=models, y=rmse_values, ax=axes[1], palette="Greens")
<ipython-input-2-90153cdced38>:217: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.barplot(x=models, y=r2_values, ax=axes[2], palette="Oranges")
```






```
{'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 300}
Epoch 1/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.3937 - loss: 0.7479 - val_accuracy: 0.6667 - val_loss: 0.6457
Epoch 2/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7023 - loss: 0.6032 - val_accuracy: 0.6875 - val_loss: 0.5919
Epoch 3/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7875 - loss: 0.5043 - val_accuracy: 0.6875 - val_loss: 0.5809
Epoch 4/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8918 - loss: 0.3629 - val_accuracy: 0.6667 - val_loss: 0.5955
Epoch 5/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8544 - loss: 0.3412 - val_accuracy: 0.6875 - val_loss: 0.6201
Epoch 6/50
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8606 - loss: 0.3293 - val_accuracy: 0.7292 - val_loss: 0.6340
Epoch 7/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9142 - loss: 0.2404 - val_accuracy: 0.7292 - val_loss: 0.6809
Epoch 8/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8759 - loss: 0.2603 - val_accuracy: 0.7292 - val_loss: 0.7014
Epoch 9/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9194 - loss: 0.2239 - val_accuracy: 0.7292 - val_loss: 0.7404
Epoch 10/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8922 - loss: 0.2567 - val_accuracy: 0.7292 - val_loss: 0.7608
Epoch 11/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9152 - loss: 0.2390 - val_accuracy: 0.7292 - val_loss: 0.7926
Epoch 12/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9434 - loss: 0.1953 - val_accuracy: 0.7292 - val_loss: 0.8161
Epoch 13/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9447 - loss: 0.1903 - val_accuracy: 0.6875 - val_loss: 0.8282
Epoch 14/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9387 - loss: 0.1791 - val_accuracy: 0.6875 - val_loss: 0.8715
Epoch 15/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9452 - loss: 0.1514 - val_accuracy: 0.6667 - val_loss: 0.8804
Epoch 16/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9424 - loss: 0.1701 - val_accuracy: 0.7083 - val_loss: 0.9057
Epoch 17/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9494 - loss: 0.1621 - val_accuracy: 0.6875 - val_loss: 0.9375
Epoch 18/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9670 - loss: 0.1339 - val_accuracy: 0.7083 - val_loss: 0.9556
Epoch 19/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9636 - loss: 0.1322 - val_accuracy: 0.7083 - val_loss: 0.9862
Epoch 20/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9702 - loss: 0.1115 - val_accuracy: 0.7292 - val_loss: 1.0022
Epoch 21/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9563 - loss: 0.1330 - val_accuracy: 0.7292 - val_loss: 1.0282
Epoch 22/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9862 - loss: 0.0966 - val_accuracy: 0.7292 - val_loss: 1.0629
Epoch 23/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9761 - loss: 0.0993 - val_accuracy: 0.7083 - val_loss: 1.0920
Epoch 24/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9671 - loss: 0.1230 - val_accuracy: 0.7083 - val_loss: 1.1160
Epoch 25/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9930 - loss: 0.0752 - val_accuracy: 0.7083 - val_loss: 1.1469
Epoch 26/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9781 - loss: 0.0903 - val_accuracy: 0.7292 - val_loss: 1.1786
Epoch 27/50
```

```
Epoch 27/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9928 - loss: 0.0696 - val_accuracy: 0.7083 - val_loss: 1.1944
Epoch 28/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9884 - loss: 0.0754 - val_accuracy: 0.7083 - val_loss: 1.2190
Epoch 29/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9914 - loss: 0.0845 - val_accuracy: 0.6875 - val_loss: 1.2617
Epoch 30/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9907 - loss: 0.0671 - val_accuracy: 0.7083 - val_loss: 1.2866
Epoch 31/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9851 - loss: 0.0704 - val_accuracy: 0.6875 - val_loss: 1.3265
Epoch 32/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9837 - loss: 0.0746 - val_accuracy: 0.6875 - val_loss: 1.3437
Epoch 33/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9856 - loss: 0.0568 - val_accuracy: 0.6875 - val_loss: 1.3817
Epoch 34/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9900 - loss: 0.0515 - val_accuracy: 0.6875 - val_loss: 1.4057
Epoch 35/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9885 - loss: 0.0437 - val_accuracy: 0.6875 - val_loss: 1.4304
Epoch 36/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9941 - loss: 0.0369 - val_accuracy: 0.6875 - val_loss: 1.4704
Epoch 37/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9837 - loss: 0.0550 - val_accuracy: 0.6875 - val_loss: 1.4951
Epoch 38/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9900 - loss: 0.0380 - val_accuracy: 0.6875 - val_loss: 1.5226
Epoch 39/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9964 - loss: 0.0305 - val_accuracy: 0.6875 - val_loss: 1.5466
Epoch 40/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9981 - loss: 0.0301 - val_accuracy: 0.6875 - val_loss: 1.5761
Epoch 41/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9884 - loss: 0.0359 - val_accuracy: 0.6875 - val_loss: 1.5926
Epoch 42/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9984 - loss: 0.0343 - val_accuracy: 0.6875 - val_loss: 1.6274
Epoch 43/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9975 - loss: 0.0280 - val_accuracy: 0.6875 - val_loss: 1.6637
Epoch 44/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9968 - loss: 0.0289 - val_accuracy: 0.6875 - val_loss: 1.6956
Epoch 45/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9939 - loss: 0.0217 - val_accuracy: 0.6875 - val_loss: 1.7195
Epoch 46/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0192 - val_accuracy: 0.6875 - val_loss: 1.7341
Epoch 47/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0366 - val_accuracy: 0.6875 - val_loss: 1.7726
```

```
Epoch 48/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0243 - val_accuracy: 0.6875 - val_loss: 1.8001
Epoch 49/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0199 - val_accuracy: 0.6875 - val_loss: 1.8363
Epoch 50/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0137 - val_accuracy: 0.6875 - val_loss: 1.8674
✅ XGBoost model trained successfully!
Enter patient details (comma or space separated):
56 1 1 130 256 1 0 142 1 2.6 1 2 6

1. Test with Logistic Regression Model
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but
  warnings.warn(
🔴 The person is likely to have heart disease.

2. Test with Random Forest Model
🔴 The person is likely to have heart disease.

3. Test with XGBoost Model
🔴 The person is likely to have heart disease.
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but
  warnings.warn(
```

Start coding or generate with AI.