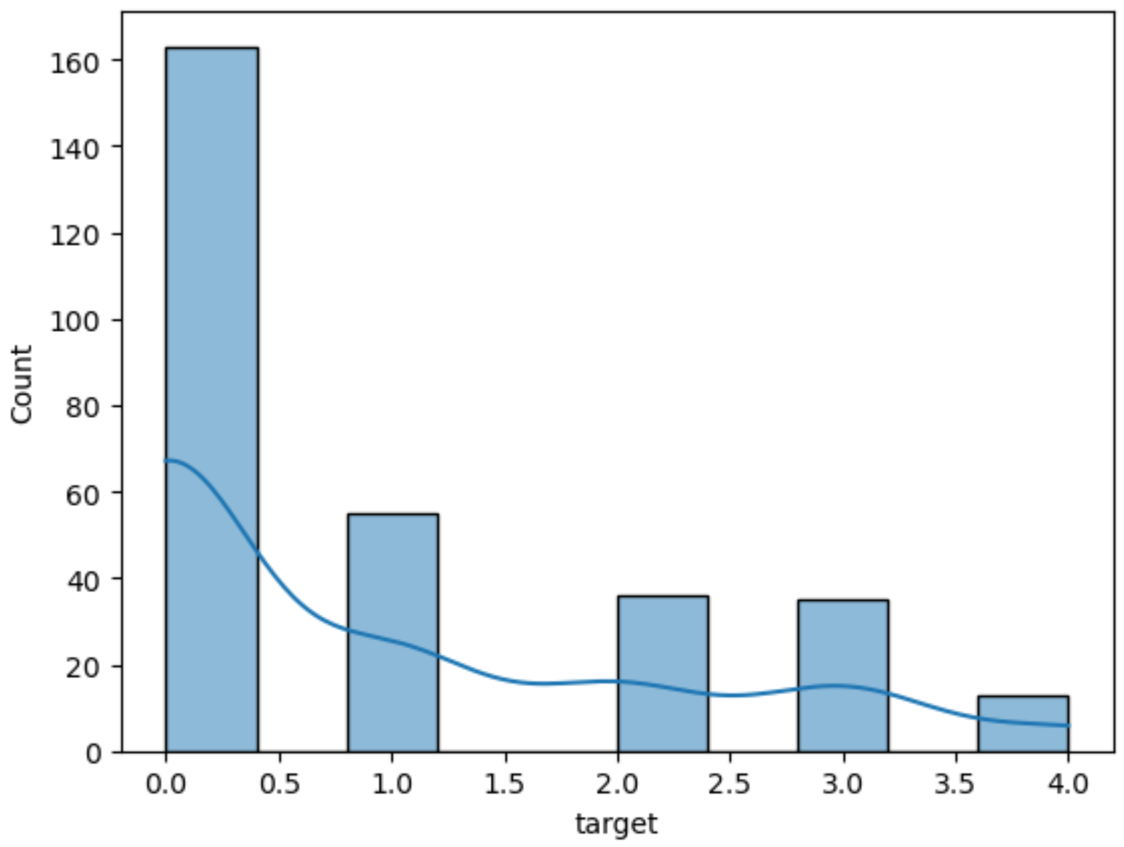
```
In… # -*- coding: utf-8 -*-
    """
    Created on Wed Mar  6 20:14:36 2024

    @author: elsyp
    """

    import pandas as pd
    from scipy.stats import zscore
    import seaborn as sns
    import matplotlib.pyplot as plt
    from scipy.stats import chi2_contingency
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error, r2_score

    # Importing Data's
    cancer_data=pd.read_csv(r"C:\Users\elsyp\OneDrive\Desktop\Downloads\hea




    # Assign meaningful column labels
    cancer_data.columns = ['age', 'sex', 'chest_pain', 'resting_blood_press
                  'rest_ecg', 'max_heart_rate', 'exercise_angina', 'st_depr
                  'thalassemia', 'target']


    # Descriptive Statistics
    print(cancer_data.describe())
    # Histogram
    sns.histplot(cancer_data['target'], kde=True)
    plt.show()


    # Display the DataFrame with your own column labels



    # Separating only the specific columns:
    fifth_columns = cancer_data.iloc[:,6]

    # Finding missing values:
    missing_values = cancer_data.isnull().sum()

    # Finding missing values in the specific rows
    cancer_data.replace('0', pd.NA, inplace=True)

    missing_values_row6 = cancer_data.iloc[:,6].isnull().sum()
    print(f"missing_values_row6: {missing_values_row6}")


    #So, we found, there is no missing values! the data set is perfect as w
    #Now, we have to consider, from the data set, which column is consider
    #For that we can use two techniques.1.correlation analysis 2.Domain Knc

    # Before normalization, we need to ensure all values are numeric where
    # Convert all columns to numeric, coercing errors to NaN (non-numeric s
cancer_data_numeric = cancer_data.apply(pd.to_numeric, errors='coerce')

# Now we handle NaN values. For this example, we'll fill NaN with the columr
# This is just an example strategy. The strategy for handling NaNs should be
cancer_data_filled = cancer_data_numeric.fillna(cancer_data_numeric.mean())

# Now, we can safely apply z-score standardization
standardization = cancer_data_numeric.apply(zscore)

## Untill hier, we have done with preprocessed the data.

#2. Data Processing:
#correlation analysis is to predict which column is to predict the heart dis
    # continious variable:-1 to 1(numerical, with numbers, blood pressure, a
    # categorical variable: 0 and 1(yes or no, male or female)
    #1. correlation matrix:
        # many steps:1. pearsons correlation coefficient, 2.Chi-square test,



#For Continuous Variables: High absolute values of Pearson's correlation coe
#indicate a strong relationship with the target variable.

# For Data Processing, we have selected correlation matrix to find out which
#column is relevant to predict the cancer disease.
correlation_matrix = cancer_data_filled.corr()
correlation_matrix = cancer_data_filled.corr()
print(correlation_matrix['target'])

sns.histplot(correlation_matrix['target'], kde=True)
plt.show()
#For Categorical Variables: A low p-value in the Chi-square test suggests a
# association with the target variable.
# Example for a categorical variable 'sex' and 'target'
contingency_table = pd.crosstab(cancer_data_filled['sex'], cancer_data_fille
chi2, p, dof, expected = chi2_contingency(contingency_table)
print(f"Chi2 Statistic: {chi2}, p-value: {p}")

# 3. Model Developement
# Regression model for continious variable and classification model is for c
# For Continious Variable: Linear regression and lasso regresssion
# For Categrical Variable: Logistic regression, decision tree, random forest




# Assuming 'target' is your continuous outcome and you've selected some feat
X = cancer_data_filled[['age', 'resting_blood_pressure', 'cholesterol']]  #
y = cancer_data_filled['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar
# For a simple linear regression


# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the testing set
```

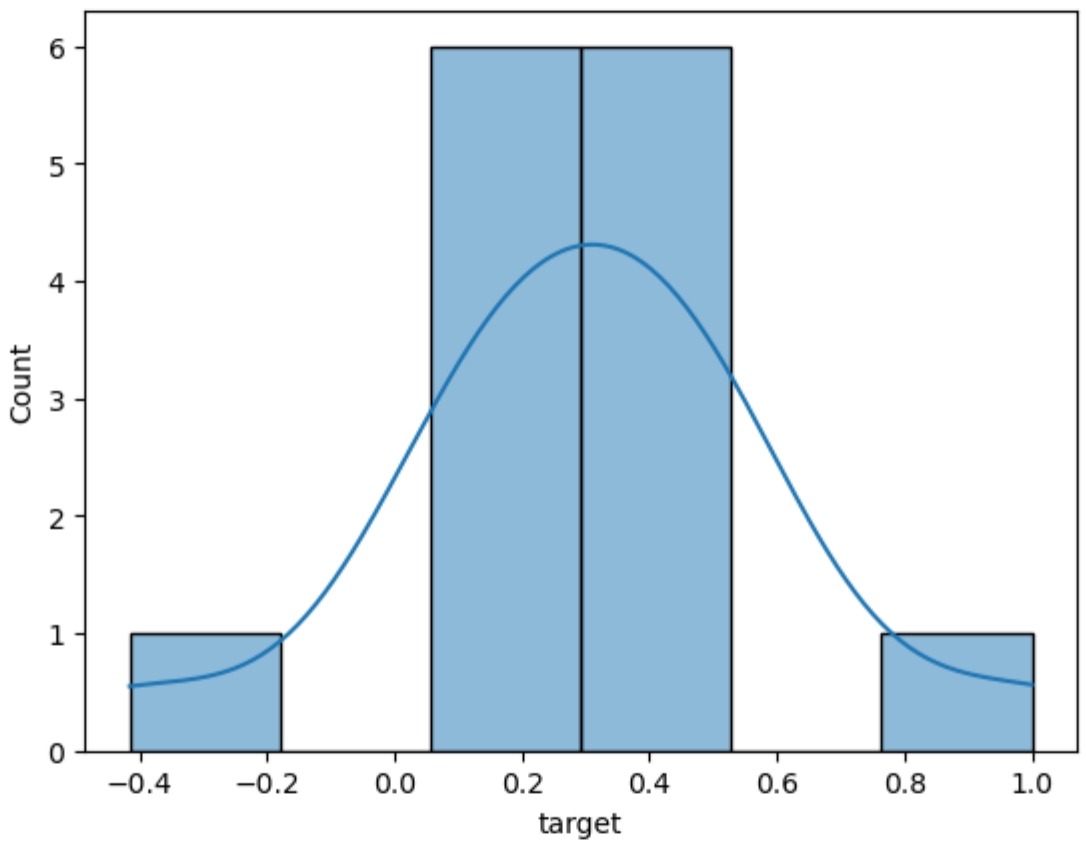
```python
y_pred = model.predict(X_test)

# Compute and print the metrics
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Coefficient of Determination (R^2):", r2_score(y_test, y_pred))

#MSE gives you the average squared difference between
#the estimated values and the actual value. A lower MSE indicates
#a better fit to the data.

#R^2 measures the proportion of the variance in
# the dependent variable that is predictable from the independent variables.
# R^2 values range from 0 to 1, where higher values indicate a better fit.
```

```
              age         sex  chest_pain  resting_blood_pressure  \
count  302.000000  302.000000  302.000000              302.000000   
mean    54.410596    0.678808    3.165563              131.645695   
std      9.040163    0.467709    0.953612               17.612202   
min     29.000000    0.000000    1.000000               94.000000   
25%     48.000000    0.000000    3.000000              120.000000   
50%     55.500000    1.000000    3.000000              130.000000   
75%     61.000000    1.000000    4.000000              140.000000   
max     77.000000    1.000000    4.000000              200.000000   

       cholesterol  fasting_blood_sugar    rest_ecg  max_heart_rate  \
count   302.000000           302.000000  302.000000      302.000000   
mean    246.738411             0.145695    0.986755      149.605960   
std      51.856829             0.353386    0.994916       22.912959   
min     126.000000             0.000000    0.000000       71.000000   
25%     211.000000             0.000000    0.000000      133.250000   
50%     241.500000             0.000000    0.500000      153.000000   
75%     275.000000             0.000000    2.000000      166.000000   
max     564.000000             1.000000    2.000000      202.000000   

       exercise_angina  st_depression       slope      target  
count       302.000000     302.000000  302.000000  302.000000  
mean          0.327815       1.035430    1.596026    0.940397  
std           0.470196       1.160723    0.611939    1.229384  
min           0.000000       0.000000    1.000000    0.000000  
25%           0.000000       0.000000    1.000000    0.000000  
50%           0.000000       0.800000    2.000000    0.000000  
75%           1.000000       1.600000    2.000000    2.000000  
max           1.000000       6.200000    3.000000    4.000000  
C:\Users\elsyp\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498: 
FutureWarning: is_categorical_dtype is deprecated and will be removed in a 
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\elsyp\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: 
FutureWarning: use_inf_as_na option is deprecated and will be removed in a 
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



```
missing_values_row6: 0
age                        0.225809
sex                        0.226601
chest_pain                 0.405182
resting_blood_pressure     0.159978
cholesterol                0.070315
fasting_blood_sugar        0.065937
rest_ecg                   0.186769
max_heart_rate            -0.415399
exercise_angina            0.395996
st_depression              0.508330
slope                      0.387417
num_major_vessels          0.516489
thalassemia                0.511316
target                     1.000000
Name: target, dtype: float64
```

```
C:\Users\elsyp\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed in a
future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\elsyp\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



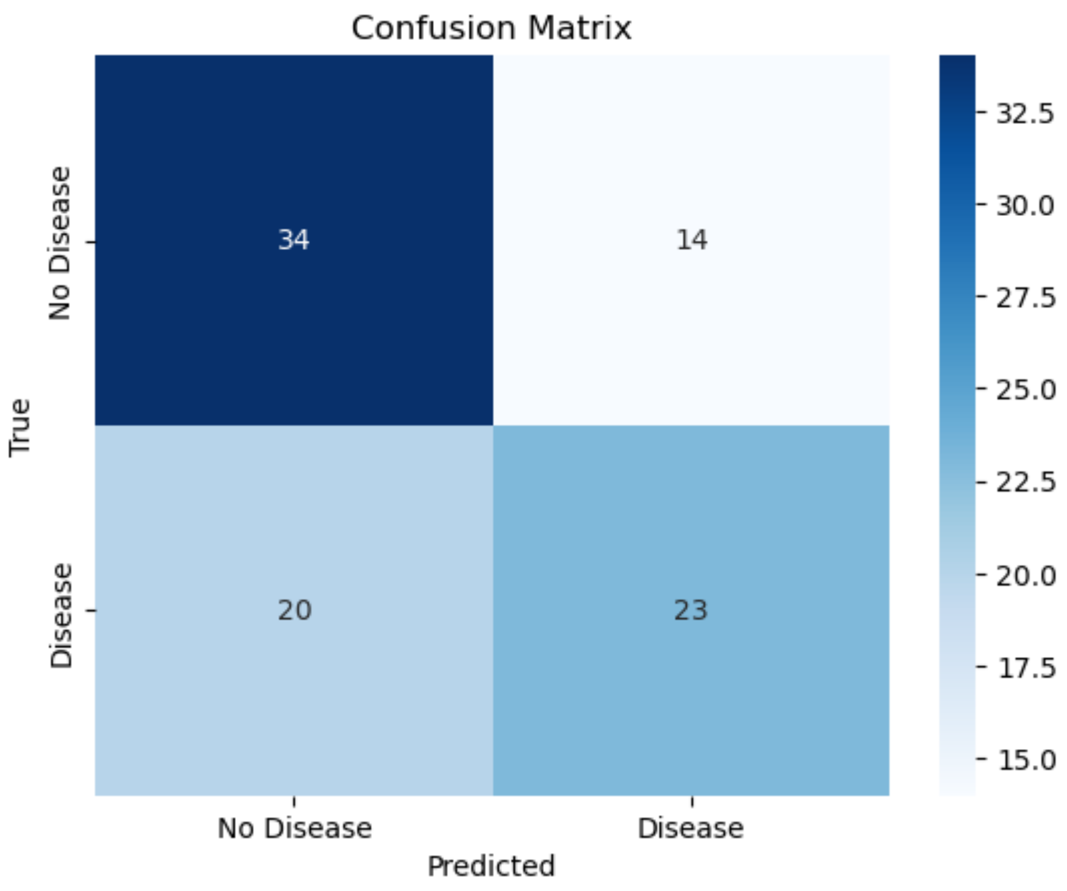
```
Chi2 Statistic: 23.80239806063686, p-value: 8.749938657615487e-05
Mean Squared Error (MSE): 1.4146083492468018
Coefficient of Determination (R^2): 0.023476847273027235
```

```
In…  # Import necessary libraries
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, confusion_matrix, ac

     # Import your data
     cancer_data = pd.read_csv(r"C:\Users\elsyp\OneDrive\Desktop\Downloads\h

     # Define column names
     cancer_data.columns = ['age', 'sex', 'chest_pain', 'resting_blood_press
                            'rest_ecg', 'max_heart_rate', 'exercise_angina',
                            'thalassemia', 'target']

     # Convert target to binary if it's not already
     cancer_data['target'] = (cancer_data['target'] != 0).astype(int)

     # Prepare data for training
     X = cancer_data[['sex', 'resting_blood_pressure', 'cholesterol']]  # Yo
     y = cancer_data['target']

     # Split the data
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

     # Initialize and train the Logistic Regression model
     model = LogisticRegression()
     model.fit(X_train, y_train)

     # Make predictions
     y_pred = model.predict(X_test)

     # Evaluate the model
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred)
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.6263736263736264
Classification Report:
               precision    recall  f1-score   support

           0       0.63      0.71      0.67        48
           1       0.62      0.53      0.57        43

    accuracy                           0.63        91
   macro avg       0.63      0.62      0.62        91
weighted avg       0.63      0.63      0.62        91

Confusion Matrix:
 [[34 14]
 [20 23]]
```

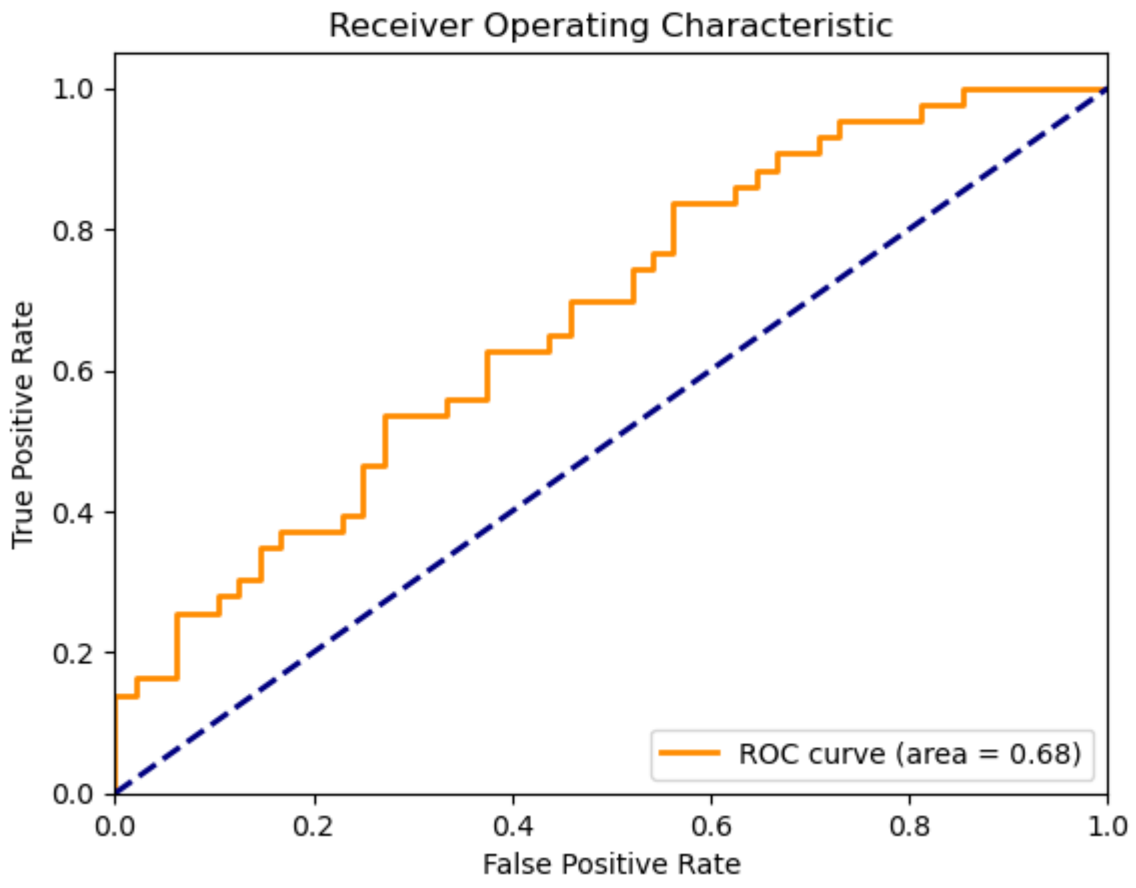
```
In [ ]:
```

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

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Dis
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



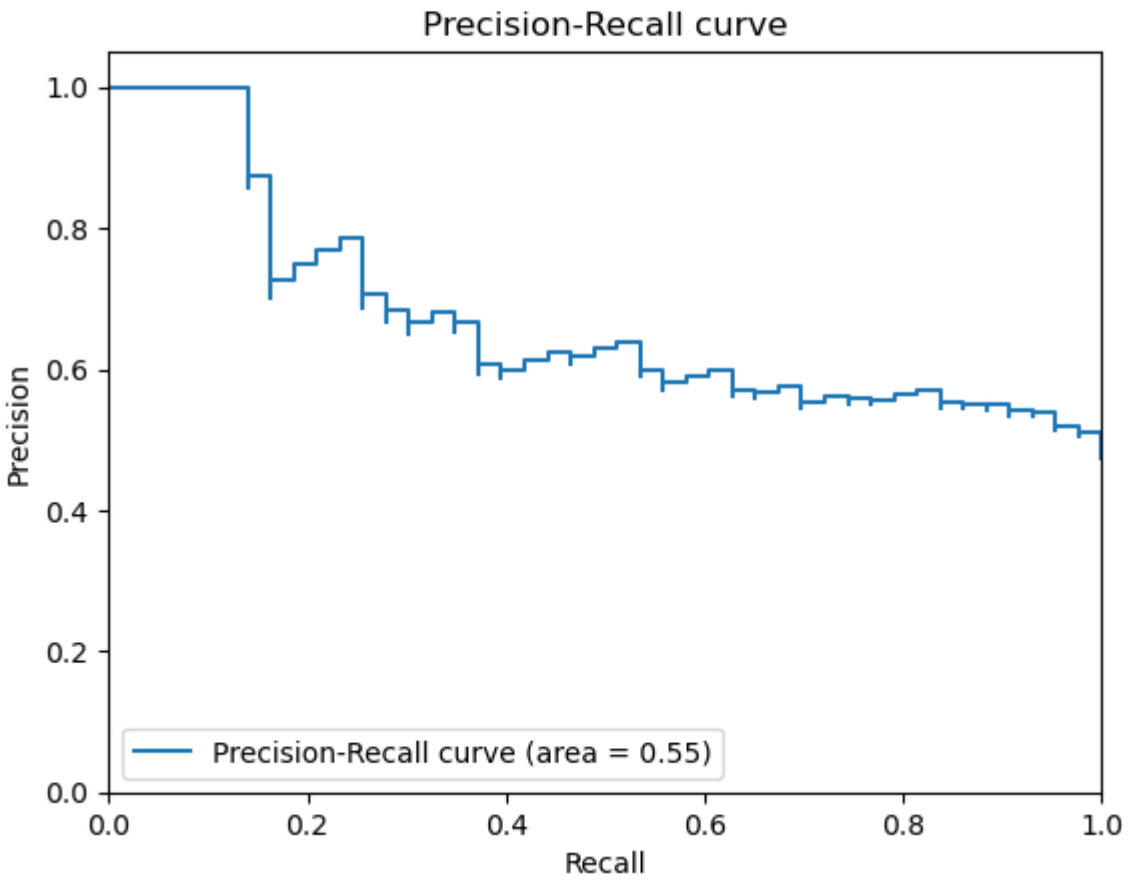
In [ ]:

```
In […  from sklearn.metrics import roc_curve, auc

       # Calculate the ROC curve
       fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[
       roc_auc = auc(fpr, tpr)

       # Plot the ROC curve
       plt.figure()
       plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
       plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
       plt.xlim([0.0, 1.0])
       plt.ylim([0.0, 1.05])
       plt.xlabel('False Positive Rate')
       plt.ylabel('True Positive Rate')
       plt.title('Receiver Operating Characteristic')
       plt.legend(loc="lower right")
       plt.show()
```



In …

```
from sklearn.metrics import precision_recall_curve, average_precision_s

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(y_test, model.predict_pro
average_precision = average_precision_score(y_test, y_pred)

# Plot Precision-Recall curve
plt.figure()
plt.step(recall, precision, where='post', label='Precision-Recall curve
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve')
plt.legend(loc="lower left")
plt.show()
```

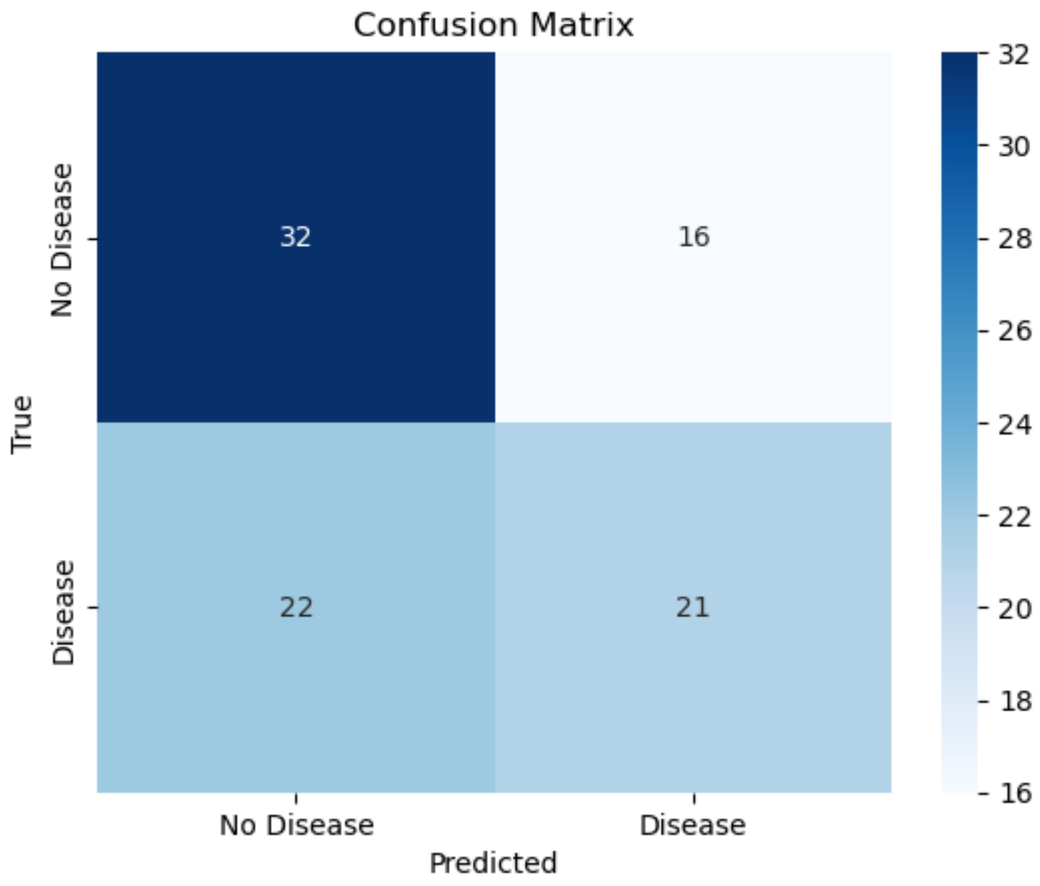


```
In…  # Import necessary libraries
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier  # Import RandomFor
     from sklearn.metrics import classification_report, confusion_matrix, ac

     # Import your data
     cancer_data = pd.read_csv(r"C:\Users\elsyp\OneDrive\Desktop\Downloads\h

     # Define column names
     cancer_data.columns = ['age', 'sex', 'chest_pain', 'resting_blood_press
                            'rest_ecg', 'max_heart_rate', 'exercise_angina',
                            'thalassemia', 'target']

     # Convert target to binary if it's not already
     cancer_data['target'] = (cancer_data['target'] != 0).astype(int)

     # Prepare data for training
     X = cancer_data[['sex', 'resting_blood_pressure', 'cholesterol']]  # Co
     y = cancer_data['target']

     # Split the data
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

     # Initialize and train the Random Forest Classifier
     model = RandomForestClassifier(n_estimators=100, random_state=42)  # Yo
     model.fit(X_train, y_train)

     # Make predictions
     y_pred = model.predict(X_test)

     # Evaluate the model
     print("Accuracy:", accuracy_score(y_test, y_pred))
     print("Classification Report:\n", classification_report(y_test, y_pred)
     print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.5824175824175825
Classification Report:
               precision    recall  f1-score   support

           0       0.59      0.67      0.63        48
           1       0.57      0.49      0.53        43

    accuracy                           0.58        91
   macro avg       0.58      0.58      0.58        91
weighted avg       0.58      0.58      0.58        91

Confusion Matrix:
 [[32 16]
 [22 21]]
```

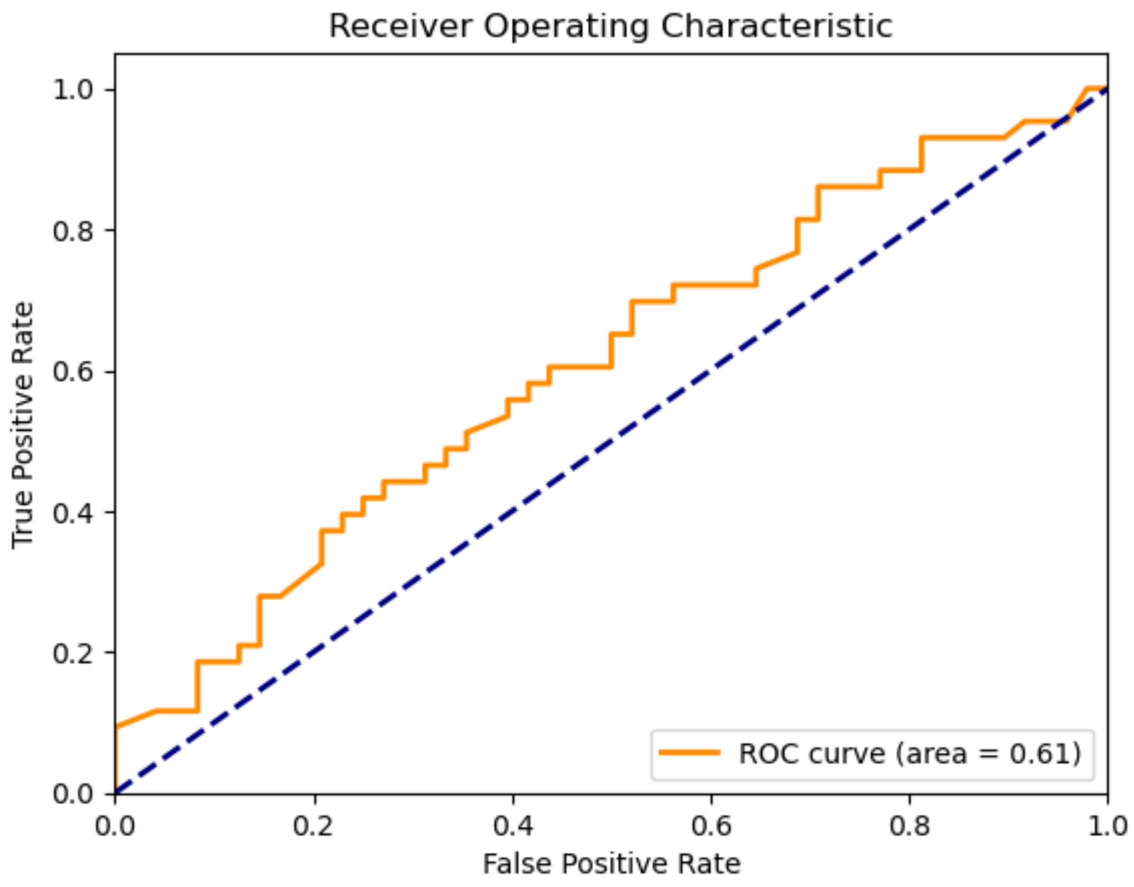
In…

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

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Dis
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

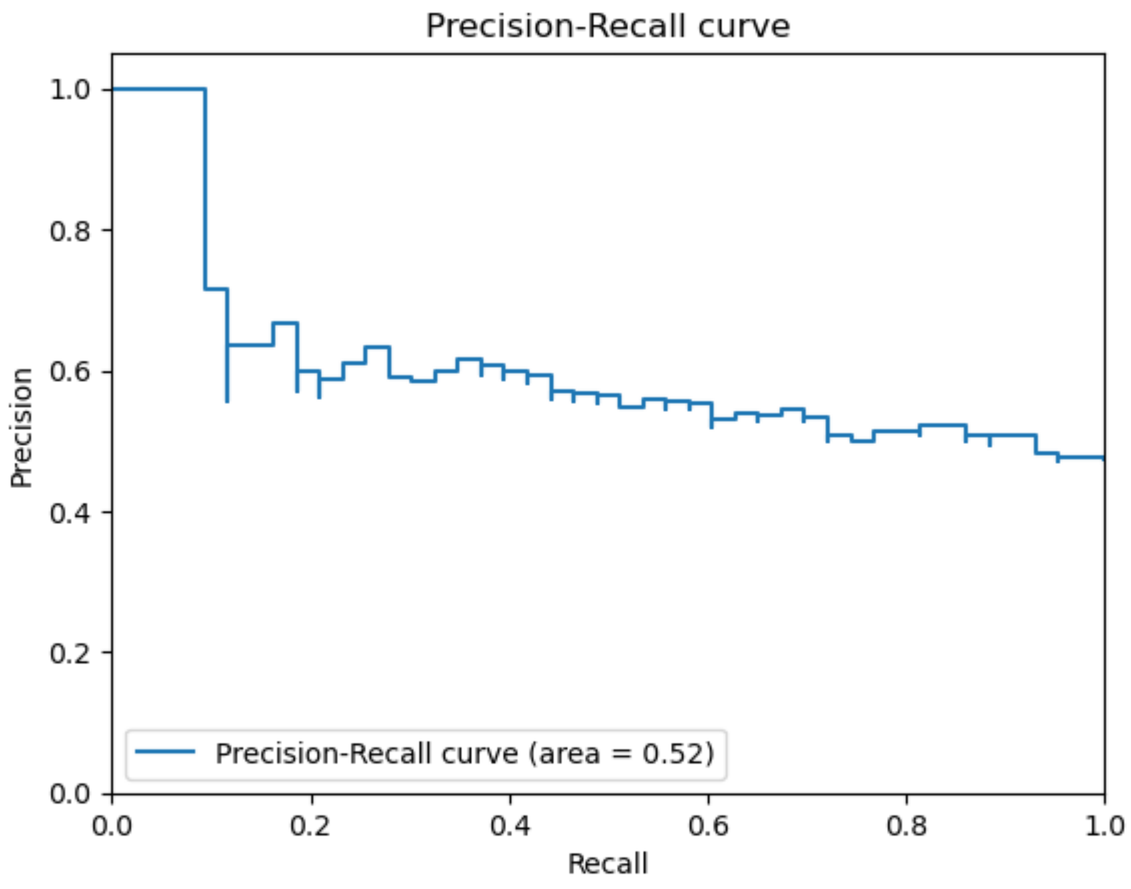


In [...

```
from sklearn.metrics import roc_curve, auc

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



```
In …  from sklearn.metrics import precision_recall_curve, average_precision_s

      # Calculate precision and recall
      precision, recall, _ = precision_recall_curve(y_test, model.predict_pro
      average_precision = average_precision_score(y_test, y_pred)

      # Plot Precision-Recall curve
      plt.figure()
      plt.step(recall, precision, where='post', label='Precision-Recall curve
      plt.xlabel('Recall')
      plt.ylabel('Precision')
      plt.ylim([0.0, 1.05])
      plt.xlim([0.0, 1.0])
      plt.title('Precision-Recall curve')
      plt.legend(loc="lower left")
      plt.show()
```



```
In […  # Hyperparameter tuning with grid search
       from sklearn.ensemble import RandomForestClassifier
       from sklearn.model_selection import GridSearchCV

       # Define the model
       rf = RandomForestClassifier(random_state=42)

       # Set up the parameters grid
       param_grid = {
           'n_estimators': [50, 100, 200],
           'max_depth': [None, 10, 20, 30],
           'min_samples_split': [2, 5, 10],
           'min_samples_leaf': [1, 2, 4]
       }

       # Set up the grid search
       grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,

       # Fit grid search to the data
       grid_search.fit(X_train, y_train)

       # Best parameters and best score
       print("Best parameters:", grid_search.best_params_)
       print("Best score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 5, 'n_estimators': 200}
Best score: 0.5919158361018827
```

```
In …  #2. Feature engineering:
      # Example of creating a new feature by combining existing features
      cancer_data['combined_feature'] = cancer_data['age'] * cancer_data['ch

      # Include this new feature in your model training
      X = cancer_data[['sex', 'resting_blood_pressure', 'cholesterol', 'comb
      y = cancer_data['target']
```

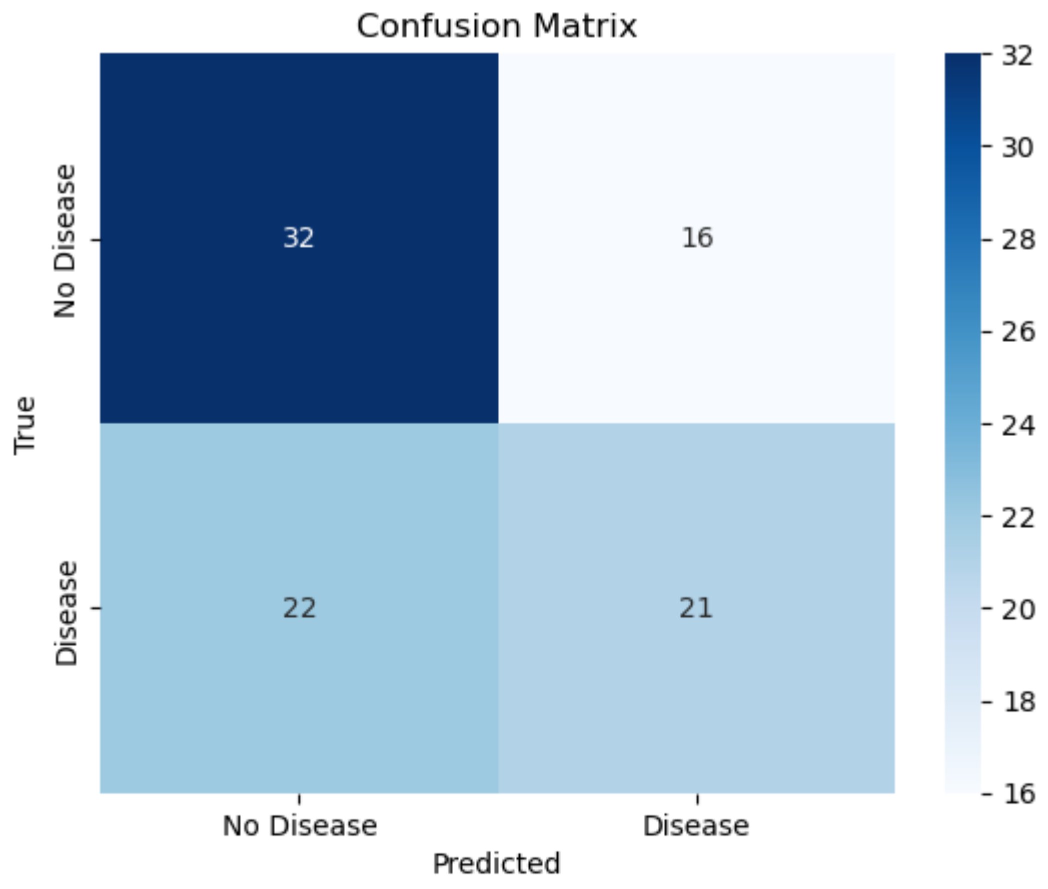
```
In [1…  #3. cross validation:
        from sklearn.model_selection import cross_val_score

        # Using the best parameters from the GridSearch
        best_rf = RandomForestClassifier(**grid_search.best_params_, random_

        # Perform cross-validation
        scores = cross_val_score(best_rf, X, y, cv=5)  # 5-fold cross-valida
        print("Cross-validated scores:", scores)
        print("Average score:", scores.mean())
```

```
Cross-validated scores: [0.59016393 0.55737705 0.65       0.65
0.48333333]
Average score: 0.5861748633879781
```

```
In [1…  #4. Handling imbalanced data
        from sklearn.model_selection import cross_val_score

        # Using the best parameters from the GridSearch
        best_rf = RandomForestClassifier(**grid_search.best_params_, random_

        # Perform cross-validation
        scores = cross_val_score(best_rf, X, y, cv=5)  # 5-fold cross-valida
        print("Cross-validated scores:", scores)
        print("Average score:", scores.mean())
```

```
Cross-validated scores: [0.59016393 0.55737705 0.65       0.65
0.48333333]
Average score: 0.5861748633879781
```

```
In [14]:
```

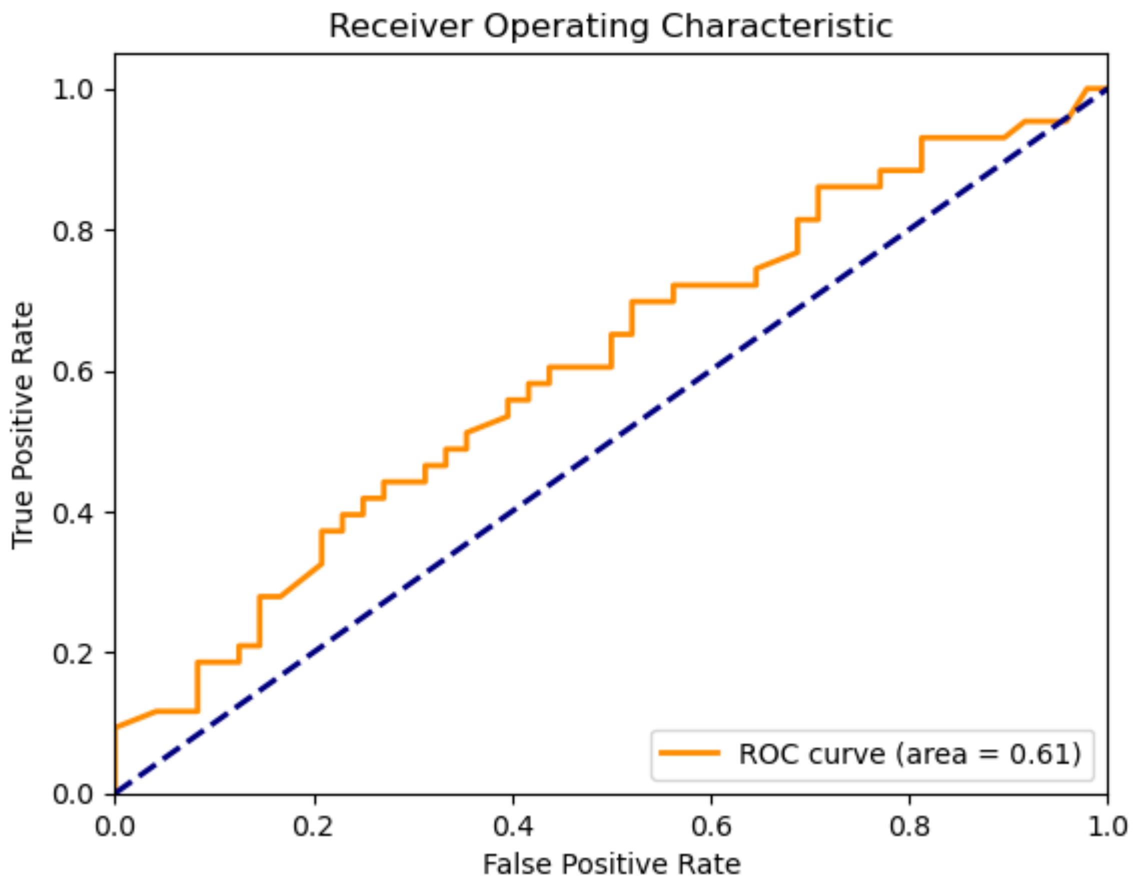


```
In …   from sklearn.metrics import accuracy_score, precision_score, recall_sc

       # Assuming you already have y_test and y_pred from your model predicti
       accuracy = accuracy_score(y_test, y_pred)
       precision = precision_score(y_test, y_pred, average='binary')
       recall = recall_score(y_test, y_pred, average='binary')
       f1 = f1_score(y_test, y_pred, average='binary')

       print(f"Accuracy: {accuracy:.2f}")
       print(f"Precision: {precision:.2f}")
       print(f"Recall: {recall:.2f}")
       print(f"F1 Score: {f1:.2f}")
       print("\nFull Classification Report:\n", classification_report(y_test,
```

```
Accuracy: 0.58
Precision: 0.57
Recall: 0.49
F1 Score: 0.53

Full Classification Report:
               precision    recall  f1-score   support

           0       0.59      0.67      0.63        48
           1       0.57      0.49      0.53        43

    accuracy                           0.58        91
   macro avg       0.58      0.58      0.58        91
weighted avg       0.58      0.58      0.58        91
```

In […

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)|
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

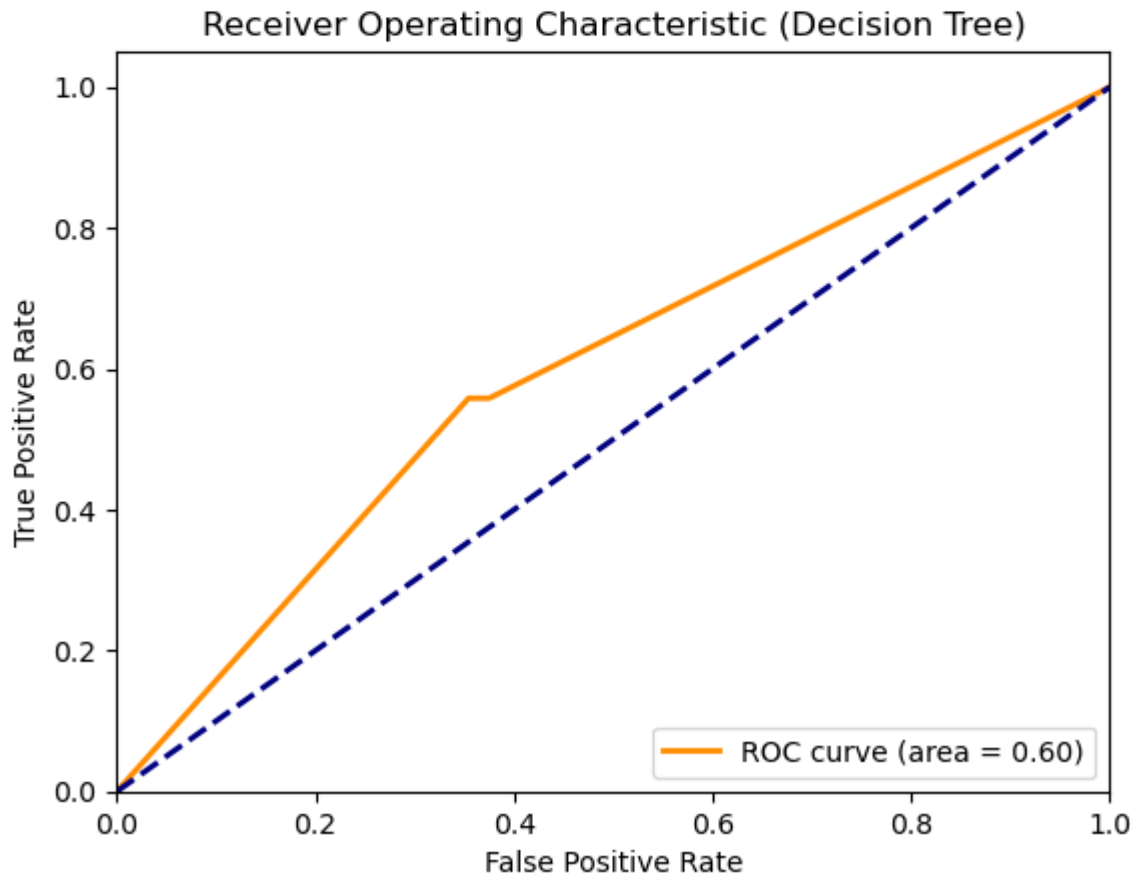


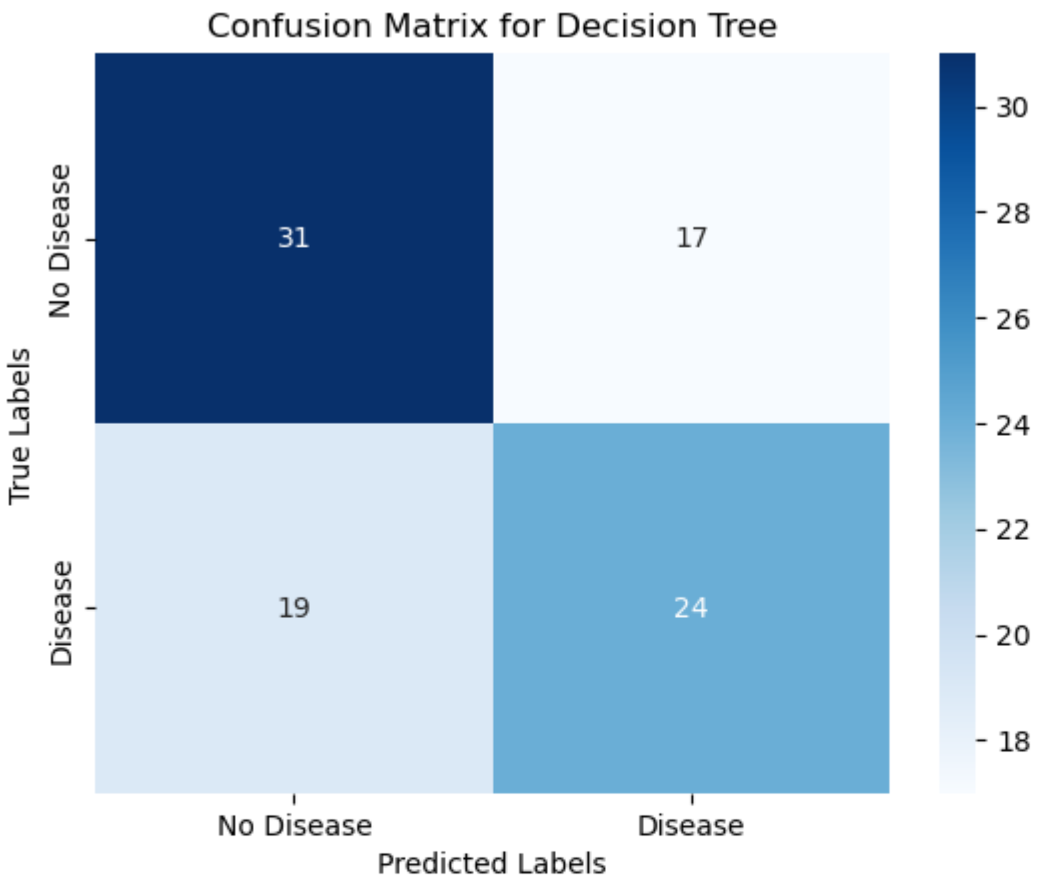
In [ ]:

In…

```python
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# classification metrics:
from sklearn.metrics import accuracy_score, precision_score, recall_scc

print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt, average='binary'
print("Recall:", recall_score(y_test, y_pred_dt, average='binary'))
print("F1 Score:", f1_score(y_test, y_pred_dt, average='binary'))
print("\nClassification Report:\n", classification_report(y_test, y_pre

#ROC curve and AUC
from sklearn.metrics import roc_curve, auc

fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, dt_model.predict_prob
roc_auc_dt = auc(fpr_dt, tpr_dt)

plt.figure()
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label=f'ROC curve (a
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (Decision Tree)')
plt.legend(loc="lower right")
plt.show()
# confusion amtrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, fmt="d", cmap='Blues', xticklabels=['No
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix for Decision Tree')
plt.show()
```

```
Accuracy: 0.6043956043956044
Precision: 0.5853658536585366
Recall: 0.5581395348837209
F1 Score: 0.5714285714285714

Classification Report:
               precision    recall  f1-score   support

           0       0.62      0.65      0.63        48
           1       0.59      0.56      0.57        43

    accuracy                           0.60        91
   macro avg       0.60      0.60      0.60        91
weighted avg       0.60      0.60      0.60        91
```




Confusion Matrix for Decision Tree
True Labels
No Disease
Disease
Predicted Labels
No Disease
Disease
31
17
19
24
30
28
26
24
22
20
18

```
In [ ]:
```