[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score

import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# Load the data
df = pd.read_csv('/content/heart.csv')
df.head()
```

[2]:
```
Age Sex ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
\
     0   40M     ATA   140   289   0     Normal     172
     1   49 F NAP 160 180 0 Normal 156 2 37 M ATA 130 283 0 ST 98
     3   48F     ASY   138   214   0     Normal     108
     4   54M     NAP   150   195   0     Normal     122

       ExerciseAngina Oldpeak ST_Slope HeartDisease
     0              N   0.0   Up    0
     1              N   1.0   Flat  1
     2              N   0.0   Up    0
     3              Y   1.5   Flat  1
     4              N   0.0   Up    0
```

[3]:
```
df.info()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to
917 Data columns (total 12
columns):
 #  Column        Non-Null Count Dtype
--- ------        -------------- -----

0   Age     918 non-null    int64

1   Sex     918 non-null    object
```

```
2   ChestPainType      918  non-null     object
3   RestingBP     918  non-null     int64
4   Cholesterol   918  non-null     int64
5   FastingBS     918  non-null     int64
6   RestingECG    918  non-null     object
7   MaxHR   918  non-null     int64
8   ExerciseAngina 918  non-null   object
9   Oldpeak 918  non-null     float64
10  ST_Slope      918  non-null     object
11  HeartDisease 918  non-null     int64
    dtypes: float64(1), int64(6), object(5)
    memory usage: 86.2+ KB
```

[4]:
```
df.describe()
```

[4]:
```
              Age RestingBP Cholesterol  FastingBS      MaxHR \
count 918.000000 918.000000         918.000000 918.000000
918.000000
mean   53.510893 132.396514 198.799564   0.233115
                                          136.809368
std     9.432617  18.514154 109.384145   0.423046 25.460334
min    28.000000   0.000000   0.000000   0.000000 60.000000
25%    47.000000 120.000000 173.250000   0.000000
                                          120.000000
50%    54.000000 130.000000 223.000000   0.000000
                                          138.000000
75%    60.000000 140.000000 267.000000   0.000000
                                          156.000000
max    77.000000 200.000000 603.000000   1.000000
                                          202.000000
        Oldpeak HeartDisease
count 918.000000 918.000000
mean  0.887364  0.553377  std
1.066570   0.497414   min   -
2.600000     0.000000    25%
0.000000 0.000000
50%   0.600000   1.000000 75%
      1.500000   1.000000 max
      6.200000   1.000000
```

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

[5]:
```
Age              0
```

```
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

[6]:
```
# Check for missing values (already confirmed none in previous
step) # Handle any zero values that might be errors (like
Cholesterol=0) df['Cholesterol'] = df['Cholesterol'].replace(0,
df['Cholesterol'].median())
```

[7]:
```
# Check for duplicates print(f"Number of
duplicates: {df.duplicated().sum()}") df =
df.drop_duplicates()
```

```
Number of duplicates: 0
```

[8]:
```
# Check target variable distribution
df['HeartDisease'].value_counts(normalize=True)
```

[8]:
```
HeartDisease
1    0.553377
0    0.446623
Name: proportion, dtype: float64
```

[9]:
```
# Separate features and
target X =
df.drop('HeartDisease',
axis=1) y = df['HeartDisease']

# Identify categorical and numerical columns categorical_cols
= ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG',␣
↪'ExerciseAngina', 'ST_Slope'] numerical_cols = ['Age',
'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

# Create preprocessing pipelines
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
```

```python
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer,
    categorical_cols) ])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

[10]:
```python
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier

# Initialize models
models = {
    'SVM': SVC(random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'KNN': KNeighborsClassifier(),
    'XGBoost': XGBClassifier(random_state=42, eval_metric='logloss')
}

# Create a dictionary to store results
results = {}

# Train and evaluate each model
for name, model in models.items():
    # Create pipeline
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

    # Train model
    pipeline.fit(X_train, y_train)

    # Make predictions
    y_pred = pipeline.predict(X_test)

    # Evaluate model
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    # Store results
    results[name] = {
        'model': pipeline,
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': cm
    }

    print(f"\n{name} Results:")
```

```
SVM Results:
```

```
Decision Tree Results:

Random Forest Results:

KNN Results:

XGBoost Results:
```

[11]:
```
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.8370
```

[12]:
```
print("Classification Report:")
print(report)
```

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

           0       0.80      0.84      0.82        82
           1       0.87      0.83      0.85       102

    accuracy                           0.84       184
   macro avg       0.83      0.84      0.84       184
weighted
avg                0.84      0.84      0.84       184
```

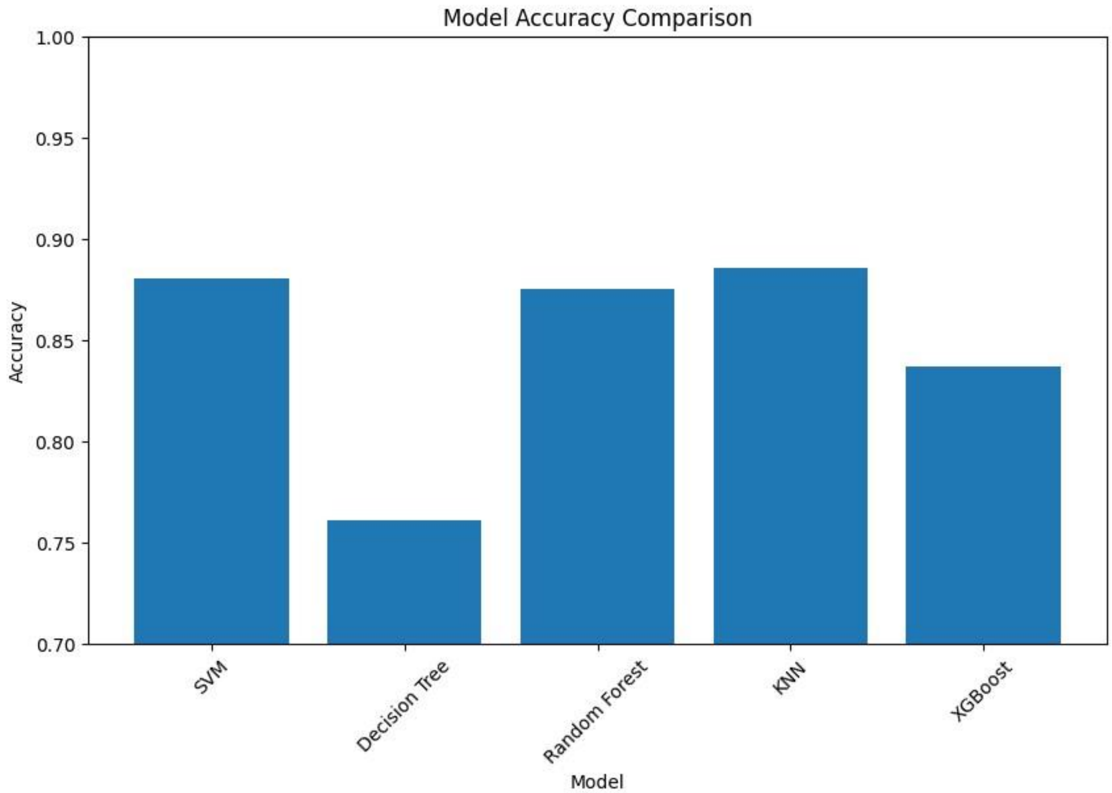
[13]:
```
print("Confusion Matrix:")
print(cm)
```

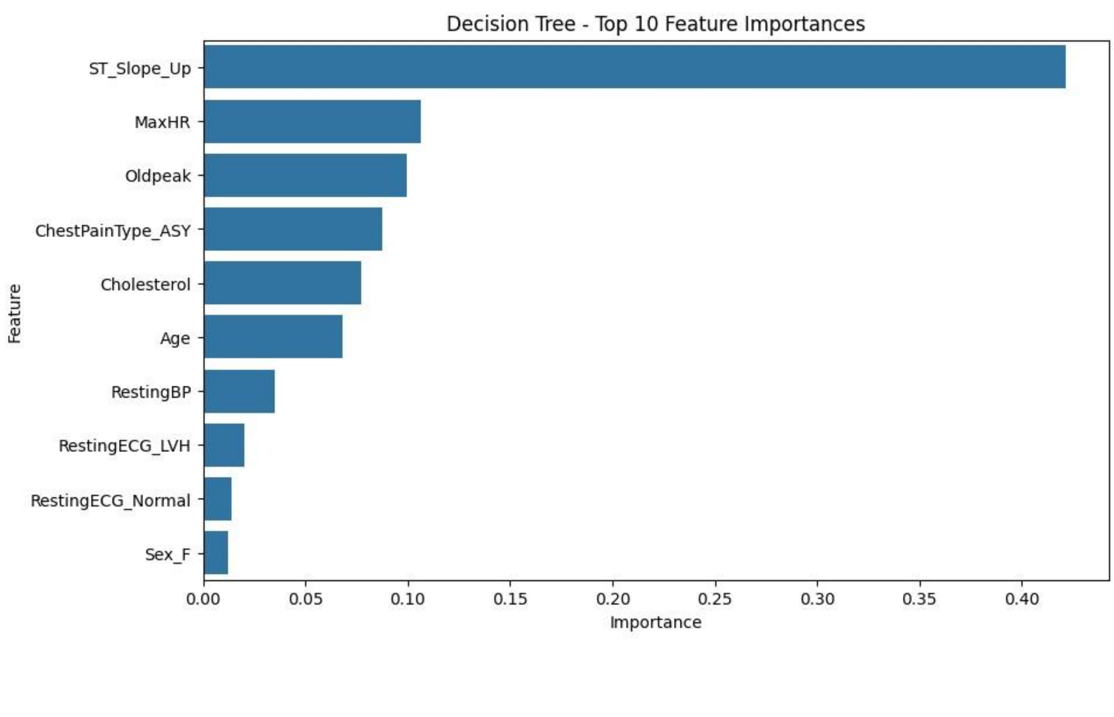
```
Confusion Matrix:
[[69 13]
 [17 85]]
```

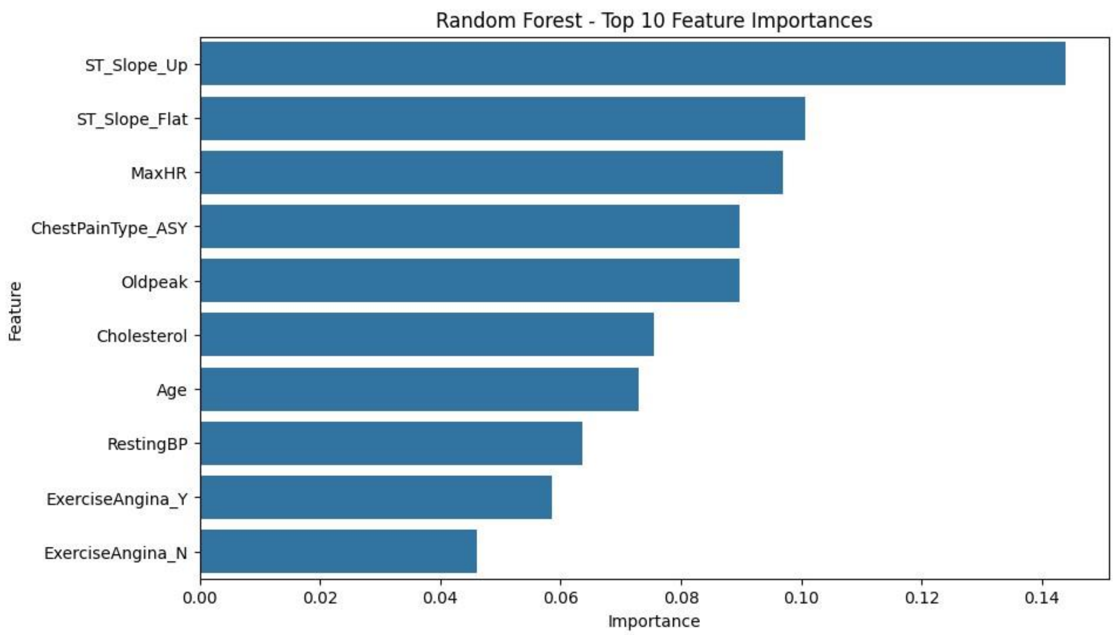
[14]:
```
# Compare model accuracies accuracies = {name: result['accuracy']
for name, result in results.items()} sorted_accuracies =
sorted(accuracies.items(), key=lambda x: x[1], reverse=True)

print("\nModel Accuracy
Comparison:") for name, acc in
sorted_accuracies: print(f"{name}:
{acc:.4f}")

# Visualize accuracy comparison
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(),
accuracies.values()) plt.title('Model Accuracy
Comparison')
```

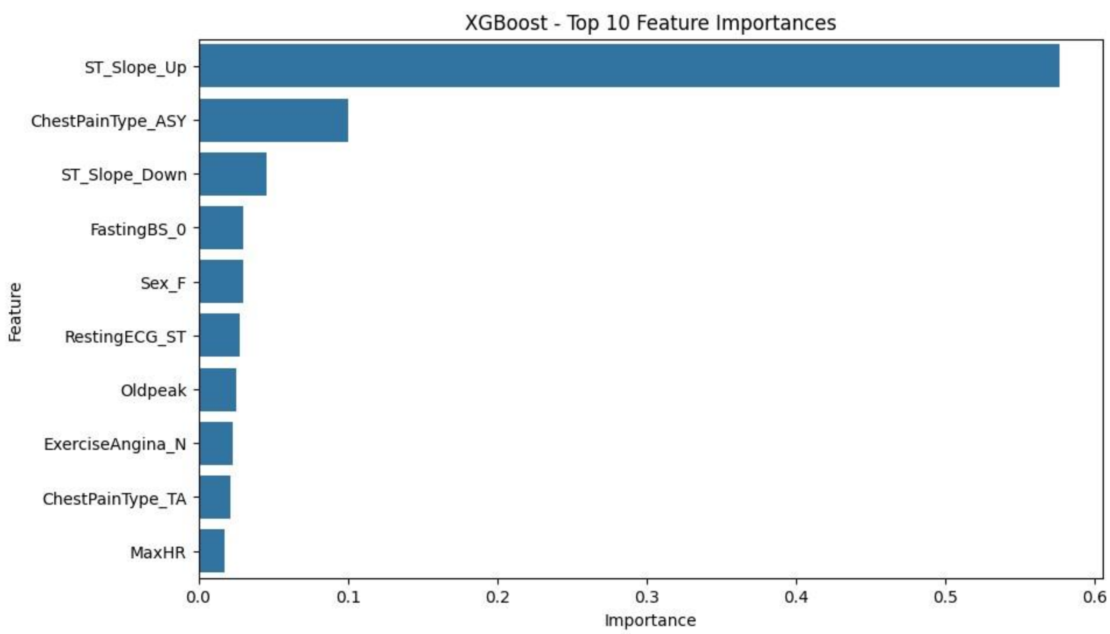
```
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.7, 1.0)
plt.xticks(rotation=45)
plt.show()

# Feature importance for tree-based models
for name in ['Decision Tree', 'Random Forest', 'XGBoost']:
    try:
        # Get feature names after one-hot encoding
        preprocessor.fit(X_train)
        feature_names = numerical_cols + list(preprocessor.
 ↪named_transformers_['cat'].get_feature_names_out(categorical_cols))

        # Get feature importances
        if name == 'XGBoost':
            importances = results[name]['model'].named_steps['classifier'].
 ↪feature_importances_
        else:
            importances = results[name]['model'].named_steps['classifier'].
 ↪feature_importances_

        # Create DataFrame for visualization
        importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
 ↪importances})
        importance_df = importance_df.sort_values('Importance',␣
 ↪ascending=False).head(10)

        # Plot
        plt.figure(figsize=(10, 6))
        plt.title(f'{name} - Top 10 Feature Importances')
        sns.barplot(x='Importance', y='Feature', data=importance_df)
        plt.show()
    except Exception as e:
        print(f"Could not plot feature importance for {name}: {str(e)}")
```

```
Model Accuracy Comparison:
KNN: 0.8859
SVM: 0.8804
Random Forest: 0.8750
XGBoost: 0.8370
Decision Tree: 0.7609
```


Model Accuracy Comparison
1.00
0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
SVM
Decision Tree
Random Forest
KNN
XGBoost
Model


Decision Tree - Top 10 Feature Importances
ST_Slope_Up
MaxHR
Oldpeak
ChestPainType_ASY
Cholesterol
Age
RestingBP
RestingECG_LVH
RestingECG_Normal
Sex_F
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance





[15]:
```
from sklearn.model_selection import GridSearchCV

# Example for Random Forest
param_grid = {
    'classifier__n_estimators': [100, 200, 300],
```

```
    'classifier__max_depth':  None, 5, 10],
    'classifier__min_samples_split':  2, 5, 10]
}

rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

grid_search = GridSearchCV(rf_pipeline, param_grid, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best accuracy: {grid_search.best_score_:.4f}")

# Update the best model in results
results['Random Forest (Tuned)'] = {
    'model': grid_search.best_estimator_,
    'accuracy': accuracy_score(y_test, grid_search.best_estimator_.
 ↪predict(X_test)),
    'report': classification_report(y_test, grid_search.best_estimator_.
 ↪predict(X_test)),
    'confusion_matrix': confusion_matrix(y_test, grid_search.best_estimator_.
 ↪predict(X_test))
}
```

```
Best parameters: {'classifier__max_depth': 5,
'classifier__min_samples_split':
10, 'classifier__n_estimators':
200} Best accuracy: 0.8637
```

[16]:
```
from sklearn.neighbors import KNeighborsClassifier from
sklearn.model_selection import GridSearchCV,
RandomizedSearchCV from sklearn.metrics import
make_scorer, accuracy_score, f1_score import numpy as np
```

[17]:
```
param_grid = {
    'classifier__n_neighbors': np.arange(3, 30, 2), # Odd numbers to
    avoid ties
    'classifier__weights': ['uniform', 'distance'],
    'classifier__p': [1, 2], # 1: Manhattan, 2: Euclidean
    'classifier__metric': ['minkowski', 'cosine',
'manhattan'] }
```

[18]:
```
# Reuse the preprocessor from previous steps
knn_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier())
])
```

[19]:
```
# Using accuracy as the scoring metric
scorer = make_scorer(accuracy_score)

# GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=knn_pipeline,
    param_grid=param_grid,
    scoring=scorer,
    cv=5,
    n_jobs=-1,  # Use all available CPU cores
    verbose=1
)
```

[20]:
```
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 168 candidates, totalling 840 fits
```

[20]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor
',
                                        ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                                         ['Age',
'RestingBP',
'Cholesterol',
'MaxHR',
'Oldpeak']),
                                                                        ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                                         ['Sex',
'ChestPainType',
'FastingBS',
'RestingECG',
'ExerciseAngina',
'ST_Slope'])])),
                                       ('classifier', KNeighborsClassifier())]),
             n_jobs=-1,
             param_grid={'classifier__metric': ['minkowski', 'cosine',
                                                'manhattan'],
                         'classifier__n_neighbors': array([ 3,  5,  7,  9, 11,
```

```
13, 15, 17, 19, 21, 23, 25, 27, 29]),
                         'classifier__p': [1,
                                  2],
                         'classifier__weights': ['uniform',
                         'distance']},
             scoring=make_scorer(accuracy_score,
             response_method='predict'), verbose=1)
```

[21]:
```
# Best parameters and score print("Best parameters found: ",
grid_search.best_params_) print("Best cross-validation accuracy:
{:.2f}%".format(grid_search.best_score_␣
↪* 100))

# Evaluate on test set
best_knn = grid_search.best_estimator_ y_pred =
best_knn.predict(X_test) test_accuracy = accuracy_score(y_test,
y_pred) print("\nTest set accuracy with best KNN:
{:.2f}%".format(test_accuracy * 100))
```

```
Best parameters found: {'classifier__metric': 'minkowski',
'classifier__n_neighbors': np.int64(15), 'classifier__p': 1,
'classifier__weights': 'uniform'}
Best cross-validation accuracy:

86.37% Test set accuracy with best

KNN: 91.30%
```

[22]:
```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.90      0.90        82
           1       0.92      0.92      0.92       102

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted           0.91      0.91      0.91       184
avg
```

[23]:
```
# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[74  8]
 [ 8 94]]
```

[24]:
```python
# Get the best estimator from GridSearchCV
best_knn = grid_search.best_estimator_

# View the best parameters
print("Best Parameters Found:")
print(grid_search.best_params_)
```

```
Best Parameters Found:
{'classifier__metric': 'minkowski', 'classifier__n_neighbors':
np.int64(15),
'classifier__p': 1, 'classifier__weights': 'uniform'}
```

[25]:
```python
# Get the best estimator from GridSearchCV
best_knn = grid_search.best_estimator_

# View the best parameters
print("Best Parameters Found:")
print(grid_search.best_params_)
```

```
Best Parameters Found:
{'classifier__metric': 'minkowski', 'classifier__n_neighbors':
np.int64(15),
'classifier__p': 1, 'classifier__weights': 'uniform'}
```

[26]:
```python
# Predict on test set
y_pred = best_knn.predict(X_test)
y_pred_proba = best_knn.predict_proba(X_test)[:, 1]  # Probability estimates
 ↪for class 1
```

[27]:
```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")
```

```
Test Accuracy: 0.9130 (91.30%)
```

[28]:
```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
print(f"\nTest Accuracy: {accuracy:.4f}
({accuracy*100:.2f}%)")
```

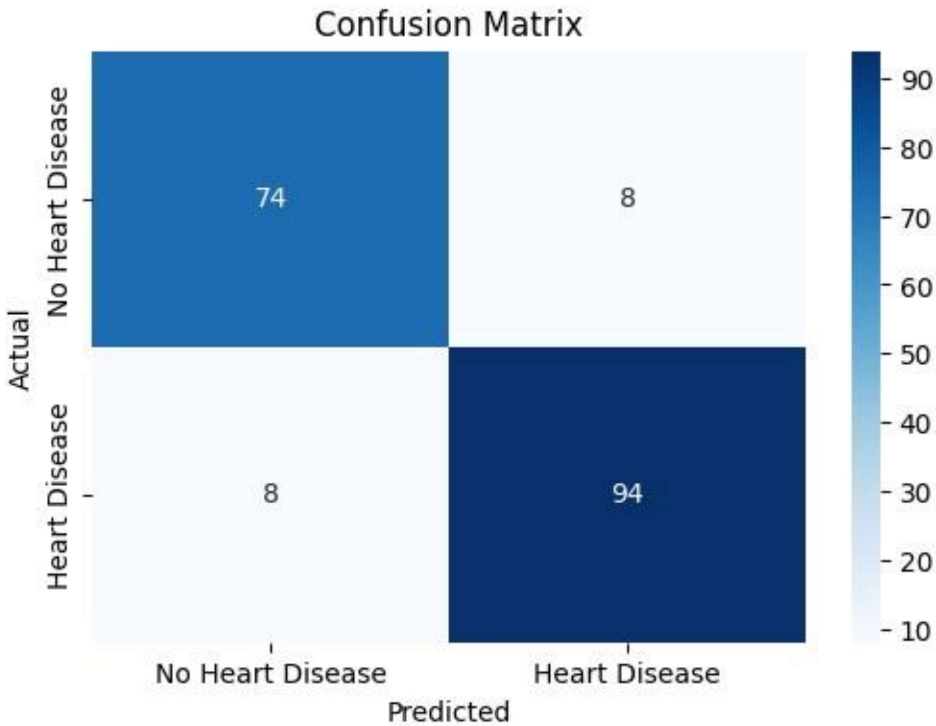
```
Test Accuracy: 0.9130 (91.30%)
```

[29]:
```python
from sklearn.metrics import classification_report

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

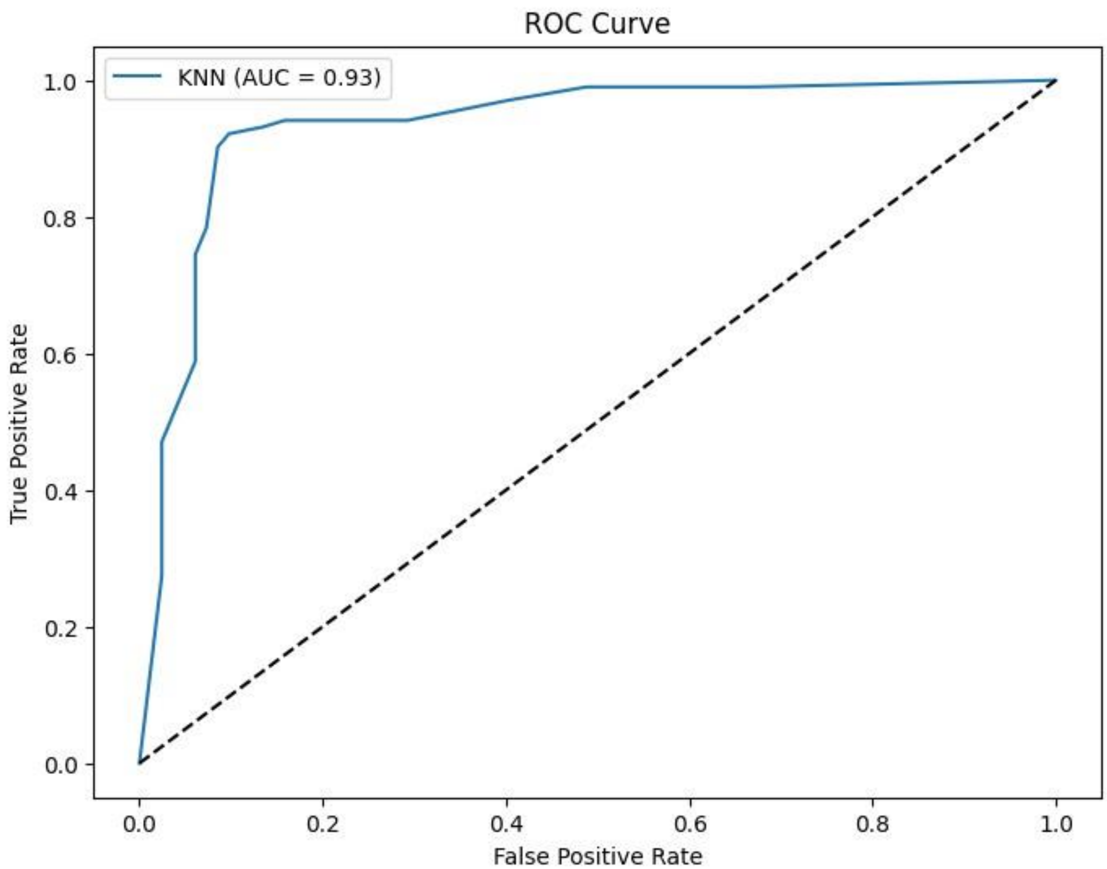
```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        82
           1       0.92      0.92      0.92       102

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184
```

[30]:
```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Visualize confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Heart Disease', 'Heart Disease'],
            yticklabels=['No Heart Disease', 'Heart Disease'])
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

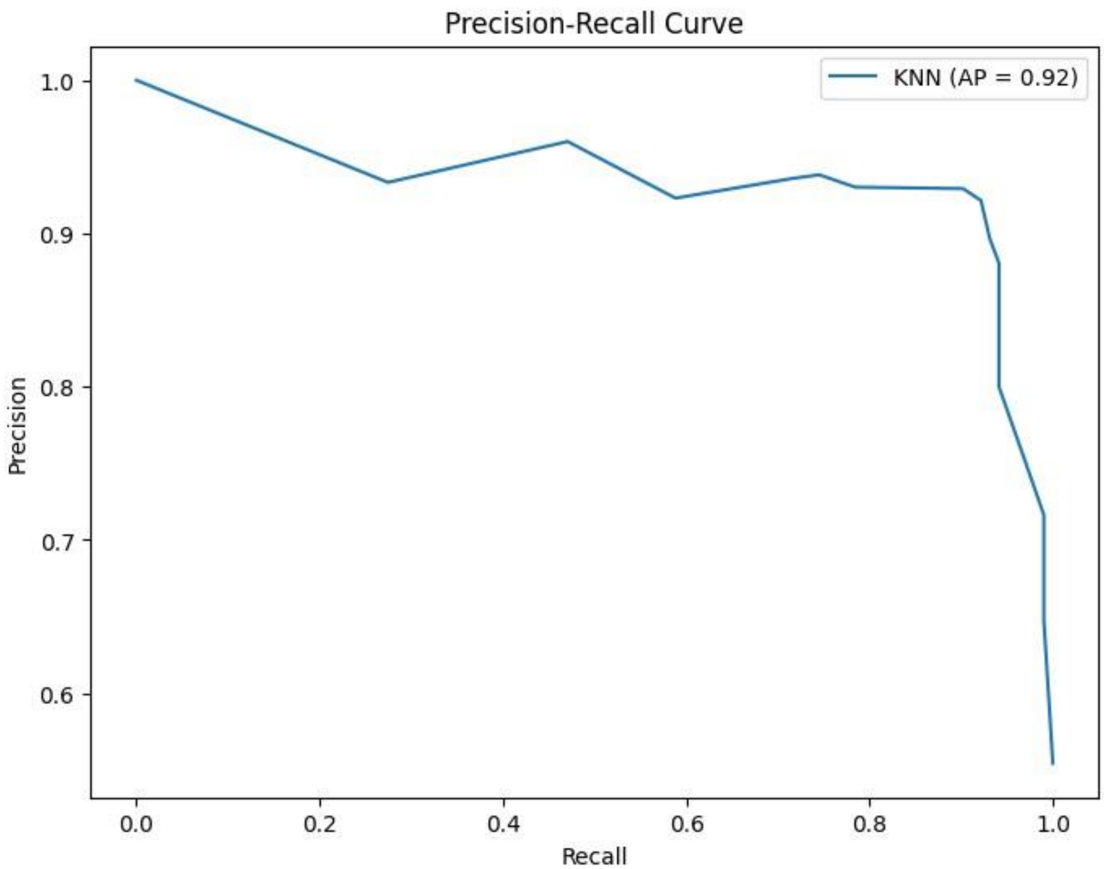
```
Confusion Matrix:
[[74  8]
 [ 8 94]]
```



[31]:
```python
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc_score = roc_auc_score(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'KNN (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')
plt.legend()
plt.show()
```

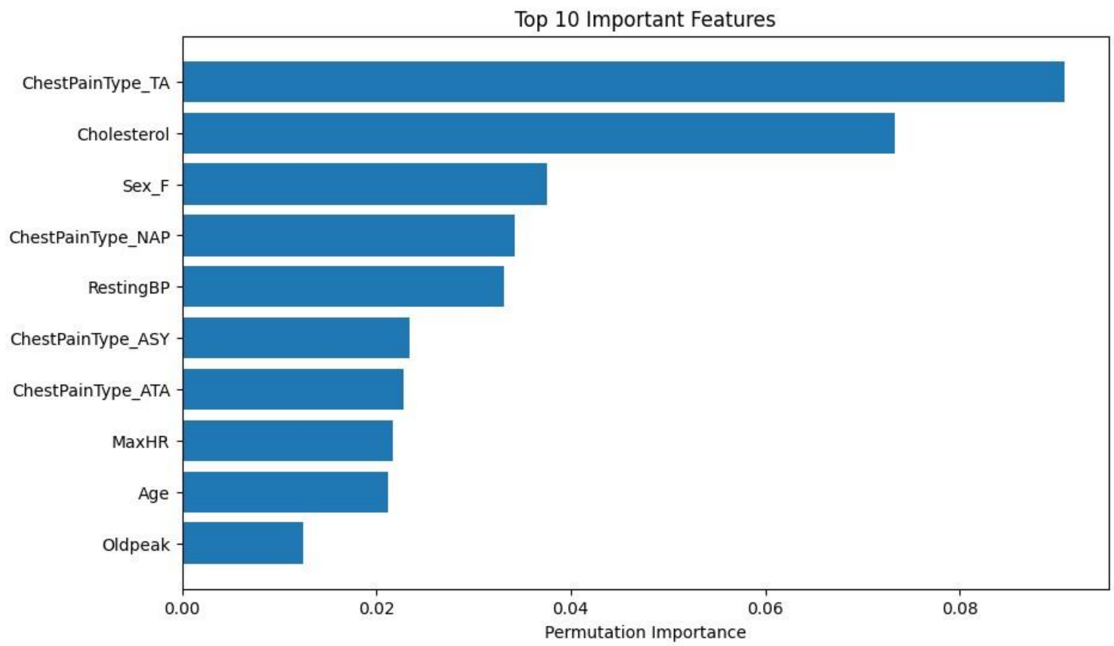


[32]:
```
from sklearn.metrics import precision_recall_curve,
average_precision_score

precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
avg_precision = average_precision_score(y_test, y_pred_proba)

plt.figure(figsize=(8, 6)) plt.plot(recall, precision,
label=f'KNN (AP = {avg_precision:.2f})')
plt.xlabel('Recall') plt.ylabel('Precision')
plt.title('Precision-Recall Curve') plt.legend() plt.show()
```




```
[33]: from sklearn.inspection import permutation_importance

# Get feature names after preprocessing
preprocessor.fit(X_train) feature_names = numerical_cols +
list(preprocessor.named_transformers_['cat'].
↪get_feature_names_out(categorical_cols))

# Calculate permutation importance result =
permutation_importance(best_knn, X_test, y_test,
n_repeats=10,␣ ↪random_state=42)

# Sort features by importance
sorted_idx = result.importances_mean.argsort()[::-1]

# Plot top 10 features
plt.figure(figsize=(10, 6))
plt.barh(np.array(feature_names)[sorted_idx][:10],
result.importances_mean[sorted_idx][:10])
plt.xlabel("Permutation Importance")
```

```python
plt.title("Top 10 Important Features")
plt.gca().invert_yaxis()
plt.show()
```



[34]:
```python
from sklearn.metrics import precision_score, recall_score, f1_score

print("\nFinal Evaluation Metrics:")
print(f"Accuracy: {accuracy_score(y_test,
y_pred):.4f}") print(f"Precision:
{precision_score(y_test, y_pred):.4f}")
print(f"Recall: {recall_score(y_test, y_pred):.4f}")
print(f"F1 Score: {f1_score(y_test, y_pred):.4f}")
print(f"ROC AUC: {roc_auc_score(y_test,
y_pred_proba):.4f}") print(f"Average Precision:
{avg_precision:.4f}")
```

```
Final Evaluation Metrics:
Accuracy: 0.9130
Precision: 0.9216
Recall: 0.9216
F1 Score: 0.9216
ROC AUC: 0.9338
Average Precision: 0.9228
```

[35]:

```
import pandas as pd
import numpy as np
```

```python
def predict_heart_disease(model, input_data):
    """
    Predicts the probability of heart disease using the trained KNN model.

    Parameters:
    -----------
    model : Pipeline
        The trained scikit-learn pipeline (including preprocessing and␣
↪classifier)
    input_data : dict or pandas.DataFrame
        Input features for prediction. Can be:
        - A dictionary of feature names and values
        - A pandas DataFrame with one row of data

    Returns:
    --------
    dict
        A dictionary containing:
        - 'prediction': 0 (no heart disease) or 1 (heart disease)
        - 'probability': Probability of heart disease (0-1)
        - 'interpretation': Text description of the result
    """

    # Define expected features and their validation ranges
    feature_ranges = {
        'Age': (20, 100),
        'Sex': ['M', 'F'],
        'ChestPainType': ['ATA', 'NAP', 'ASY', 'TA'],
        'RestingBP': (80, 200),
        'Cholesterol': (100, 600),
        'FastingBS': [0, 1],
        'RestingECG': ['Normal', 'ST', 'LVH'],
        'MaxHR': (60, 220),
        'ExerciseAngina': ['Y', 'N'],
        'Oldpeak': (-2.5, 6.5),
        'ST_Slope': ['Up', 'Flat', 'Down']
    }

    try:
        # Convert input to DataFrame if it's a dictionary
        if isinstance(input_data, dict):
            input_df = pd.DataFrame([input_data])
        else:
            input_df = input_data.copy()

        # Validate input features
        missing_features = set(feature_ranges.keys()) - set(input_df.columns)
```

```
        if missing_features:
            raise ValueError(f"Missing features: {missing_features}")

        # Validate feature values
        for feature, valid_range in feature_ranges.items():
            value = input_df[feature].iloc[0]

            if feature in ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG',␣
↪'ExerciseAngina', 'ST_Slope']:
                if value not in valid_range:
                    raise ValueError(f"Invalid value for {feature}. Must be one␣
↪of: {valid_range}")
            else:
                if not (valid_range[0] <= value <= valid_range[1]):
                    raise ValueError(f"Invalid value for {feature}. Must be␣
↪between {valid_range[0]} and {valid_range[1]}")

        # Make prediction
        proba = model.predict_proba(input_df)[0][1]
        prediction = model.predict(input_df)[0]

        # Create interpretation
        if prediction == 1:
            interpretation = f"High risk of heart disease ({proba*100:.1f}%␣
↪probability)"
        else:
            interpretation = f"Low risk of heart disease ({(1-proba)*100:.1f}%␣
↪probability)"

        return {
            'prediction': int(prediction),
            'probability': float(proba),
            'interpretation': interpretation
        }

    except Exception as e:
        return {
            'error': str(e),
            'suggestion': 'Please check your input data format and values'
        }
```

[36]:
```
import joblib

# Save the model
joblib.dump(best_knn, 'heart_disease_knn_model.pkl')

# Later, load the model
```

```python
model = joblib.load('heart_disease_knn_model.pkl')
```

[37]:
```python
# Example input (as dictionary)
patient_data = {
    'Age': 52,
    'Sex': 'M',
    'ChestPainType': 'ASY',
    'RestingBP': 125,
    'Cholesterol': 212,
    'FastingBS': 0,
    'RestingECG': 'Normal',
    'MaxHR': 168,
    'ExerciseAngina': 'N',
    'Oldpeak': 1.0,
    'ST_Slope': 'Flat'
}

# Get prediction
result = predict_heart_disease(model, patient_data)
print(result)
```

```
{'prediction': 1, 'probability': 0.7333333333333333,
'interpretation': 'High risk of heart disease (73.3% probability)'}
```

[38]:
```python
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV(
    estimator=knn_pipeline,
    param_distributions=param_grid,
    n_iter=50,  # Number of parameter settings sampled
    scoring=scorer,
    cv=5,
    n_jobs=-1,
    verbose=1,
    random_state=42
)

random_search.fit(X_train, y_train)

# Analyze results similarly to grid search
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

[38]:
```
RandomizedSearchCV(cv=5,
                   estimator=Pipeline(steps=[('preprocessor
',
ColumnTransformer(transformers=[('num',
StandardScaler(),
```

```
['Age',
'RestingBP',
'Cholesterol',
'MaxHR',
'Oldpeak']),
('cat',
OneHotEncoder(handle_unknown='ignore'),
['Sex',
'ChestPainType',
'FastingBS',
'RestingECG',
'ExerciseAngina',
'ST_Slope'])])),
                                    ('classifier',
                                  KNeighborsClassifier())]),
                n_iter=50, n_jobs=-1,
                param_distributions={'classifier__metric':
                ['minkowski',
                                                   'cosine',
                                                  'manhattan'],
                                  'classifier__n_neighbors':
array([ 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]),
                                  'classifier__p': [1, 2],
                                'classifier__weights': ['uniform',
                                                   'distance']},
                random_state=42,
                scoring=make_scorer(accuracy_score,
response_method='predict'),
                verbose=1)
```