In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    roc_curve,
    roc_auc_score,
    classification_report
)
```

In [ ]:

In [8]:

```
from google.colab import files
import pandas as pd

# Upload file
uploaded = files.upload()

# Load the dataset
import io
df = pd.read_csv(io.BytesIO(uploaded['heart.csv']))

# Display the first few rows
print(df.head())
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart.csv
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR
\
0   40   M           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   48   F           ASY        138          214          0     Normal    108
4   54   M           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
```

In [2]:

```
import pandas as pd
```

In [ ]:

```
# Dataset overview
print(df.info())

# Check the first few rows
print(df.head())

# Statistical summary
print(df.describe())
```

In [9]:

```
# Check for null values
print(df.isnull().sum())
```

```
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
```

In [14]:

```
# Check for missing values again
print(df.isnull().sum())

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

```
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
```

In [ ]:

In [20]:

```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart.csv
```

In [21]:

```
import pandas as pd
import io

# Load the dataset (make sure the file name matches the uploaded file's name)
df = pd.read_csv(io.BytesIO(uploaded['heart.csv']))

# Display the first few rows of the dataset to confirm it loaded properly
print(df.head())
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR
\
0   40   M           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   48   F           ASY        138          214          0     Normal    108
4   54   M           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
```

In [ ]:

In [22]:

```
# One-hot encode categorical features
df = pd.get_dummies(df, columns=['Sex', 'ChestPainType', 'ExerciseAngina',
'ST_Slope'], drop_first=True)

# Check the first few rows of the dataset after encoding
print(df.head())
```

```
   Age  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  Oldpeak  \
0   40        140          289          0     Normal    172      0.0
1   49        160          180          0     Normal    156      1.0
2   37        130          283          0         ST     98      0.0
3   48        138          214          0     Normal    108      1.5
4   54        150          195          0     Normal    122      0.0

   HeartDisease  Sex_M  ChestPainType_ATA  ChestPainType_NAP  \
0             0   True               True              False
1             1  False              False               True
2             0   True               True              False
3             1  False              False              False
4             0   True              False               True

   ChestPainType_TA  ExerciseAngina_Y  ST_Slope_Flat  ST_Slope_Up
0             False             False          False         True
1             False             False           True        False
2             False             False          False         True
3             False              True           True        False
4             False             False          False         True
```

In [23]:

```
print(df.head())  # Display the first few rows to ensure encoding is correct
```

```
   Age  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  Oldpeak  \
0   40        140          289          0     Normal    172      0.0
1   49        160          180          0     Normal    156      1.0
2   37        130          283          0         ST     98      0.0
3   48        138          214          0     Normal    108      1.5
4   54        150          195          0     Normal    122      0.0

   HeartDisease  Sex_M  ChestPainType_ATA  ChestPainType_NAP  \
0             0   True               True              False
1             1  False              False               True
2             0   True               True              False
3             1  False              False              False
4             0   True              False               True

   ChestPainType_TA  ExerciseAngina_Y  ST_Slope_Flat  ST_Slope_Up
0             False             False          False         True
1             False             False           True        False
```

```
2             False             False          False         True
3             False              True           True        False
4             False             False          False         True
```

In [26]:

```
import pandas as pd


# Check the data types of the columns
print(df.dtypes)
```

```
Age                     int64
RestingBP               int64
Cholesterol             int64
FastingBS               int64
RestingECG             object
MaxHR                   int64
Oldpeak               float64
HeartDisease            int64
Sex_M                    bool
ChestPainType_ATA        bool
ChestPainType_NAP        bool
ChestPainType_TA         bool
ExerciseAngina_Y         bool
ST_Slope_Flat            bool
ST_Slope_Up              bool
dtype: object
```

In [32]:

```
# Convert non-numeric columns to numeric (if they should be numeric)
# For example, if 'Age' and 'Cholesterol' columns are strings, convert them
to numeric
# You can also apply it to all columns that should be numeric
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')  # Replace 'Age' with
the actual column name
print(df.dtypes)
# Select only numerical columns (e.g., float64 and int64)
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Calculate the first (25th percentile) and third (75th percentile) quartiles
Q1 = numerical_df.quantile(0.25)
Q3 = numerical_df.quantile(0.75)

# Calculate the IQR (Interquartile Range)
IQR = Q3 - Q1

# Identify outliers (values that are outside the IQR range)
outliers = ((numerical_df < (Q1 - 1.5 * IQR)) | (numerical_df > (Q3 + 1.5 *
IQR)))

# Display outliers (True indicates outliers)
print(outliers)
```

```
Age                int64
Sex               object
ChestPainType     object
RestingBP          int64
Cholesterol        int64
FastingBS          int64
RestingECG        object
MaxHR              int64
ExerciseAngina    object
Oldpeak          float64
ST_Slope          object
HeartDisease       int64
dtype: object
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
..	...	...	...	...	...	...	...
913	False	False	False	False	False	False	False
914	False	False	False	True	False	False	False
915	False	False	False	False	False	False	False
916	False	False	False	False	False	False	False
917	False	False	False	False	False	False	False

[918 rows x 7 columns]

In [30]:

```
Age                int64
Sex               object
ChestPainType     object
RestingBP          int64
Cholesterol        int64
FastingBS          int64
RestingECG        object
MaxHR              int64
ExerciseAngina    object
Oldpeak          float64
ST_Slope          object
HeartDisease       int64
dtype: object
```

In [33]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

In [34]:

```
# Load your dataset into a pandas DataFrame
import io
df = pd.read_csv(io.BytesIO(uploaded['heart.csv']))  # Adjust the path if
necessary

# Display the first few rows to ensure data is loaded correctly
print(df.head())
```

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR
\
0   40   M           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   48   F           ASY        138          214          0     Normal    108
4   54   M           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
```

In [35]:

```
# Select only numerical columns for normalization (example columns)
numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
# Adjust to your dataset

# Create a new DataFrame with only the numerical columns
df_numerical = df[numerical_columns]

# Display the selected columns
print(df_numerical.head())
```

```
   Age  RestingBP  Cholesterol  MaxHR  Oldpeak
0   40        140          289    172      0.0
1   49        160          180    156      1.0
2   37        130          283     98      0.0
3   48        138          214    108      1.5
4   54        150          195    122      0.0
```

In [36]:

```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the numerical columns
df_numerical_normalized = scaler.fit_transform(df_numerical)

# Convert the normalized data back to a DataFrame
df_numerical_normalized = pd.DataFrame(df_numerical_normalized,
columns=numerical_columns)

# Display the normalized values
```

```
print(df_numerical_normalized.head())
```

```
        Age  RestingBP  Cholesterol     MaxHR   Oldpeak
0  0.244898       0.70     0.479270  0.788732  0.295455
1  0.428571       0.80     0.298507  0.676056  0.409091
2  0.183673       0.65     0.469320  0.267606  0.295455
3  0.408163       0.69     0.354892  0.338028  0.465909
4  0.530612       0.75     0.323383  0.436620  0.295455
```

In [37]:

```
# Replace the original numerical columns with the normalized values
df[numerical_columns] = df_numerical_normalized

# Display the updated DataFrame
print(df.head())
```

```
        Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
\
0  0.244898   M           ATA       0.70     0.479270          0     Normal
1  0.428571   F           NAP       0.80     0.298507          0     Normal
2  0.183673   M           ATA       0.65     0.469320          0         ST
3  0.408163   F           ASY       0.69     0.354892          0     Normal
4  0.530612   M           NAP       0.75     0.323383          0     Normal

      MaxHR ExerciseAngina   Oldpeak ST_Slope  HeartDisease
0  0.788732              N  0.295455       Up             0
1  0.676056              N  0.409091     Flat             1
2  0.267606              N  0.295455       Up             0
3  0.338028              Y  0.465909     Flat             1
4  0.436620              N  0.295455       Up             0
```

In [38]:

```
# Check the min and max values of the normalized data to ensure normalization
worked
print(df[numerical_columns].min())  # Check minimum values
print(df[numerical_columns].max())  # Check maximum values
```

```
Age           0.0
RestingBP     0.0
Cholesterol   0.0
MaxHR         0.0
Oldpeak       0.0
dtype: float64
Age           1.0
RestingBP     1.0
Cholesterol   1.0
MaxHR         1.0
Oldpeak       1.0
dtype: float64
```

In [39]:

```
# Display the updated DataFrame
print(df.head())
```

```
        Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
\
0  0.244898   M           ATA       0.70     0.479270          0     Normal
1  0.428571   F           NAP       0.80     0.298507          0     Normal
2  0.183673   M           ATA       0.65     0.469320          0         ST
3  0.408163   F           ASY       0.69     0.354892          0     Normal
4  0.530612   M           NAP       0.75     0.323383          0     Normal

      MaxHR ExerciseAngina   Oldpeak ST_Slope  HeartDisease
0  0.788732              N  0.295455       Up             0
1  0.676056              N  0.409091     Flat             1
2  0.267606              N  0.295455       Up             0
3  0.338028              Y  0.465909     Flat             1
4  0.436620              N  0.295455       Up             0
```

In [17]:

```
from sklearn.model_selection import train_test_split

# Define features and target
X = df.drop('HeartDisease', axis=1)
y = df['HeartDisease']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

In [42]:

```
# Verify dataset shapes and consistency
print(X_train.shape)
print(y_train.shape)
```

```
(734, 6)
(734,)
```

In [25]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# Train Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Predict
y_pred_dt = dt_model.predict(X_test)
```

```
# Evaluate
print("\n--- Decision Tree Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
--- Decision Tree Results ---
Accuracy: 0.7554347826086957
Confusion Matrix:
 [[55 22]
 [23 84]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.71      0.71        77
           1       0.79      0.79      0.79       107

    accuracy                           0.76       184
   macro avg       0.75      0.75      0.75       184
weighted avg       0.76      0.76      0.76       184
```

In [27]:

```
from sklearn.neighbors import KNeighborsClassifier

# Train KNN Model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Predict
y_pred_knn = knn_model.predict(X_test)

# Evaluate
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

print("\n--- K-Nearest Neighbors Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
--- K-Nearest Neighbors Results ---
Accuracy: 0.7445652173913043
Confusion Matrix:
 [[62 15]
 [32 75]]
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.81      0.73        77
           1       0.83      0.70      0.76       107
```

```
    accuracy                           0.74       184
   macro avg       0.75      0.75      0.74       184
weighted avg       0.76      0.74      0.75       184
```

In [44]:

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

# Decision Tree ROC
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_model.predict_proba(X_test)[:, 1])
dt_auc = auc(dt_fpr, dt_tpr)

# KNN ROC
knn_fpr, knn_tpr, _ = roc_curve(y_test, knn_model.predict_proba(X_test)[:,
1])
knn_auc = auc(knn_fpr, knn_tpr)

# Plot ROC Curves
plt.figure(figsize=(10, 6))
plt.plot(dt_fpr, dt_tpr, label=f"Decision Tree (AUC = {dt_auc:.2f})")
plt.plot(knn_fpr, knn_tpr, label=f"KNN (AUC = {knn_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

In [46]:

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

# Create a DataFrame for feature importances
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)


# Plot feature importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importances_df)
plt.title('Feature Importance - Decision Tree')
plt.show()
```

In [35]:

```
# Import required libraries
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Evaluate Decision Tree
dt_accuracy = accuracy_score(y_test, y_pred_dt)
dt_precision = precision_score(y_test, y_pred_dt)
dt_recall = recall_score(y_test, y_pred_dt)
dt_f1 = f1_score(y_test, y_pred_dt)

# Evaluate KNN
knn_accuracy = accuracy_score(y_test, y_pred_knn)
knn_precision = precision_score(y_test, y_pred_knn)
knn_recall = recall_score(y_test, y_pred_knn)
knn_f1 = f1_score(y_test, y_pred_knn)

# Create a performance comparison table
import pandas as pd
comparison_df = pd.DataFrame({
    'Model': ['Decision Tree', 'KNN'],
    'Accuracy': [dt_accuracy, knn_accuracy],
    'Precision': [dt_precision, knn_precision],
    'Recall': [dt_recall, knn_recall],
    'F1 Score': [dt_f1, knn_f1]
})

print(comparison_df)
```

```
           Model  Accuracy  Precision    Recall  F1 Score
0  Decision Tree  0.755435   0.792453  0.785047  0.788732
1            KNN  0.744565   0.833333  0.700935  0.761421
```

In [36]:

```
df.to_csv('preprocessed_heart_failure.csv', index=False)
```

In [37]:

```
import pickle

# Save Decision Tree Model
with open('decision_tree_model.pkl', 'wb') as file:
    pickle.dump(dt_model, file)

# Save KNN Model
with open('knn_model.pkl', 'wb') as file:
    pickle.dump(knn_model, file)
```

In [38]:

```
# Save plots as images
plt.figure(figsize=(10, 6))
plt.plot(dt_fpr, dt_tpr, label=f"Decision Tree (AUC = {dt_auc:.2f})")
plt.plot(knn_fpr, knn_tpr, label=f"KNN (AUC = {knn_auc:.2f})")
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.savefig('roc_curve_comparison.png')
plt.show()
```

In [39]:

In [6]:

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

# Train the Decision Tree model (assuming X train and y train are already
defined)
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Generate predictions on the test set
y_pred_dt = dt_model.predict(X_test)  # This is where the predictions are
made

# Calculate the confusion matrix for Decision Tree
dt cm = confusion matrix(y test, y pred dt)

# Plot the Decision Tree Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(dt_cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No',
'Yes'], yticklabels=['No', 'Yes'])
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

In [3]:

```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart.csv
```

In [4]:

```
import pandas as pd
```

```
import numpy as np
import io
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.inspection import permutation_importance

# Load your dataset (replace with your actual dataset)
df = pd.read_csv(io.BytesIO(uploaded['heart.csv'])) # Update with the correct
path

# Select the features (numerical columns for KNN)
numerical_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
# Adjust to your dataset
X = df[numerical_columns]
y = df['HeartDisease']  # Target column

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize KNN model
knn = KNeighborsClassifier(n_neighbors=5)

# Train the KNN model
knn.fit(X_train, y_train)

# Calculate permutation importance
results = permutation_importance(knn, X_test, y_test, n_repeats=10,
random_state=42)

# Get the importance values for each feature
importance = results.importances_mean

# Plot the feature importance
plt.figure(figsize=(10, 6))
plt.barh(numerical_columns, importance)
plt.xlabel('Importance')
plt.title('Feature Importance - KNN (Permutation Importance)')
plt.show()
```

In [8]:

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

# Train the KNN model (assuming X_train and y_train are already defined)
```

```
knn_model = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number
of neighbors
knn_model.fit(X_train, y_train)

# Generate predictions for the test set
y_pred_knn = knn_model.predict(X_test)

# Calculate the confusion matrix for KNN
knn_cm = confusion_matrix(y_test, y_pred_knn)

# Plot the KNN Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(knn_cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No',
'Yes'], yticklabels=['No', 'Yes'])
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```