Name: Durva Kadam Roll No: 23

St. Francis Institute of Technology, Mumbai-400 103
Department Of Information Technology

A.Y. 2025-2026
Class: BE-ITA/B, Semester: VII
Subject: Data Science Lab

Experiment – 8

1. **Aim:** To implement Supervised Learning algorithm - Random Forest.
2. **Objectives:** Students should be familiarize with Learning Architectures and Frameworks
3. **Prerequisite:** Python basics

4. **Pre-Experiment Exercise:**
Theory:
Random Forest Algorithm
Decision trees involve the greedy selection of the best split point from the dataset at each step.

This algorithm makes decision trees susceptible to high variance if they are not pruned. This high variance can be harnessed and reduced by creating multiple trees with different samples of the training dataset (different views of the problem) and combining their predictions. This approach is called bootstrap aggregation or bagging for short.

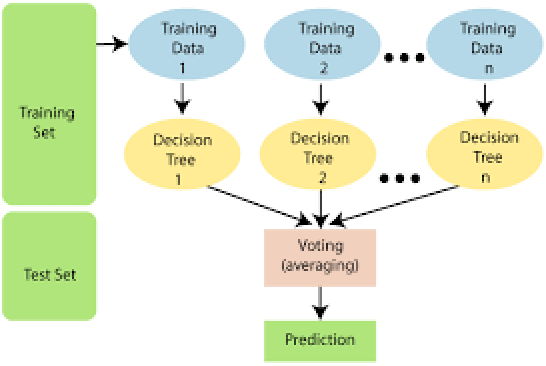
A limitation of bagging is that the same greedy algorithm is used to create each tree, meaning that it is likely that the same or very similar split points will be chosen in each tree making the different trees very similar (trees will be correlated). This, in turn, makes their predictions similar, mitigating the variance originally sought.

We can force the decision trees to be different by limiting the features (rows) that the greedy algorithm can evaluate at each split point when creating the tree. This is called the Random Forest algorithm.

Like bagging, multiple samples of the training dataset are taken and a different tree trained on each. The difference is that at each point a split is made in the data and added to the tree, only a fixed subset of attributes can be considered.

For classification problems, the type of problems we will look at in this tutorial, the number of attributes to be considered for the split is limited to the square root of the number of input features.

num_features_for_split = sqrt(total_input_features)
The result of this one small change are trees that are more different from each other (uncorrelated) resulting predictions that are more diverse and a combined prediction that often has better performance that single tree or bagging alone.



6. Laboratory Exercise

Procedure

i. Use google colab for programming.
ii. Import required packages.
iii. Demonstrate random forest classifier for any given dataset.
iv. Add relevant comments in your programs and execute the code. Test it for various cases.

Post-Experiments Exercise:

A. Extended Theory:

a. Write real life applications of Random Forest Classifier.

B. Conclusion:

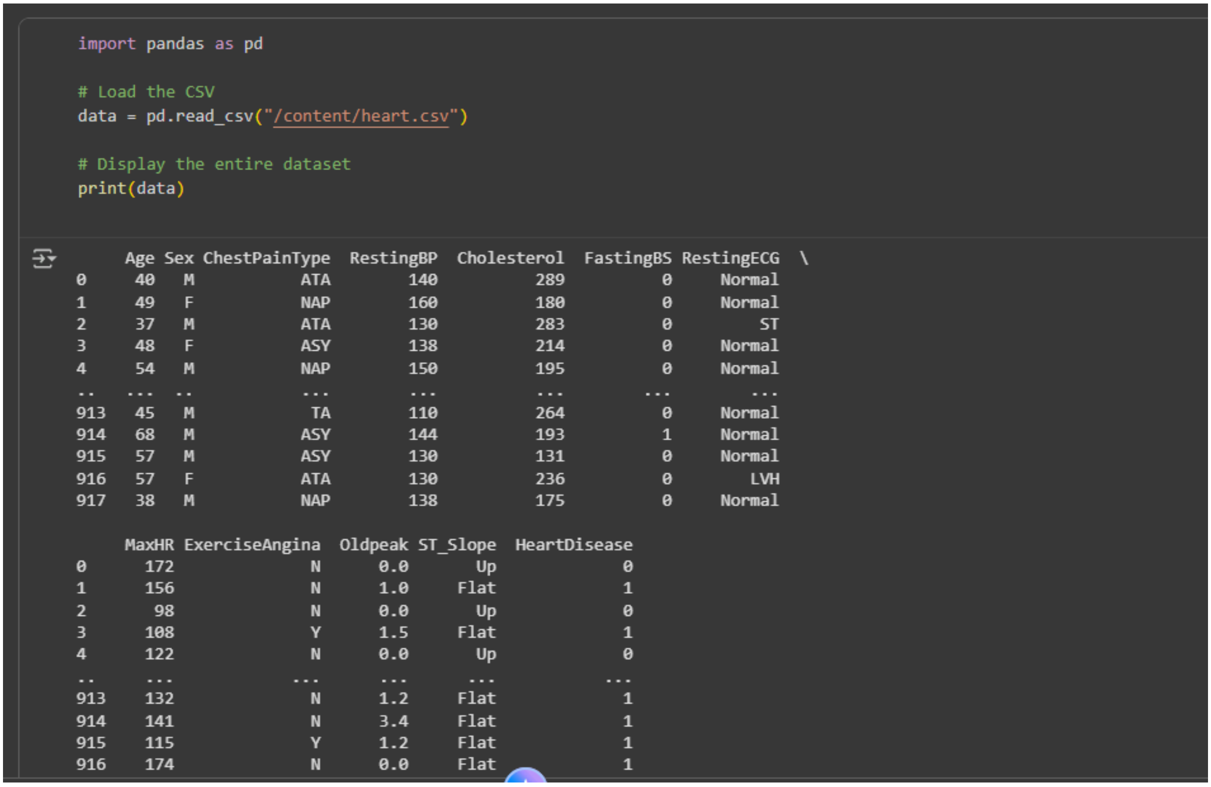
1. Write what was performed in the program (s) .
2. What is the significance of program and what Objective is achieved?

C. References:

[1] https://machinelearningmastery.com/implement-random-forest-scratch-python/.
[2] https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/

Name: Durva Kadam Roll No: 23

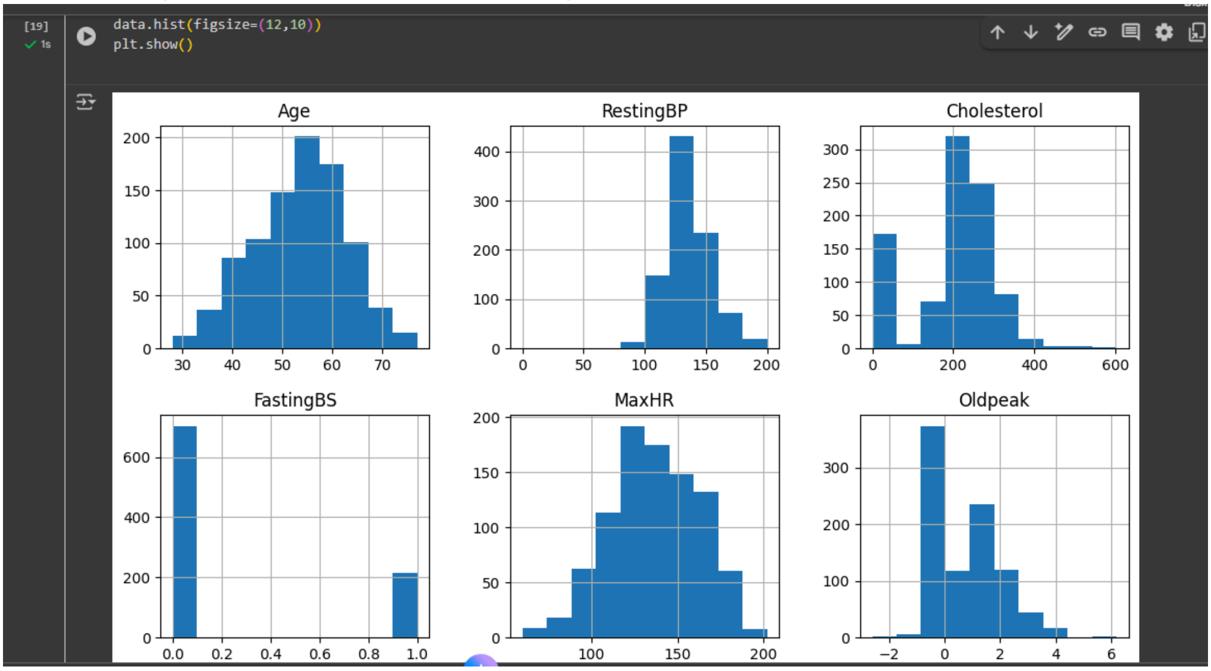
1. Importing the dataset:



```
import pandas as pd

# Load the CSV
data = pd.read_csv("/content/heart.csv")

# Display the entire dataset
print(data)
```

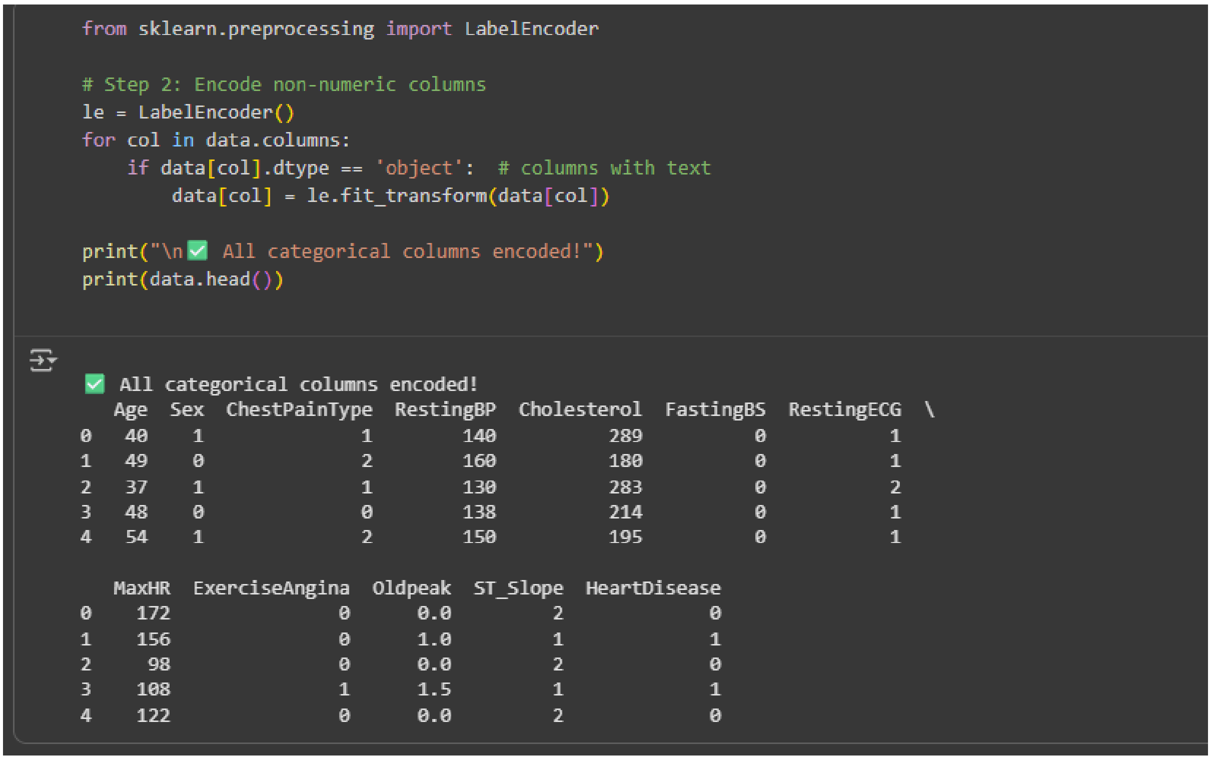
```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
0     40   M           ATA        140          289          0     Normal   
1     49   F           NAP        160          180          0     Normal   
2     37   M           ATA        130          283          0         ST   
3     48   F           ASY        138          214          0     Normal   
4     54   M           NAP        150          195          0     Normal   
..   ...  ..           ...        ...          ...        ...        ...   
913   45   M            TA        110          264          0     Normal   
914   68   M           ASY        144          193          1     Normal   
915   57   M           ASY        130          131          0     Normal   
916   57   F           ATA        130          236          0        LVH   
917   38   M           NAP        138          175          0     Normal   

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease  
0      172              N      0.0       Up             0  
1      156              N      1.0     Flat             1  
2       98              N      0.0       Up             0  
3      108              Y      1.5     Flat             1  
4      122              N      0.0       Up             0  
..     ...            ...      ...      ...           ...  
913    132              N      1.2     Flat             1  
914    141              N      3.4     Flat             1  
915    115              Y      1.2     Flat             1  
916    174              N      0.0     Flat             1  
```

2. Histograms of numeric columns (like Age, Cholesterol) to see distributions.



```
data.hist(figsize=(12,10))
plt.show()
```

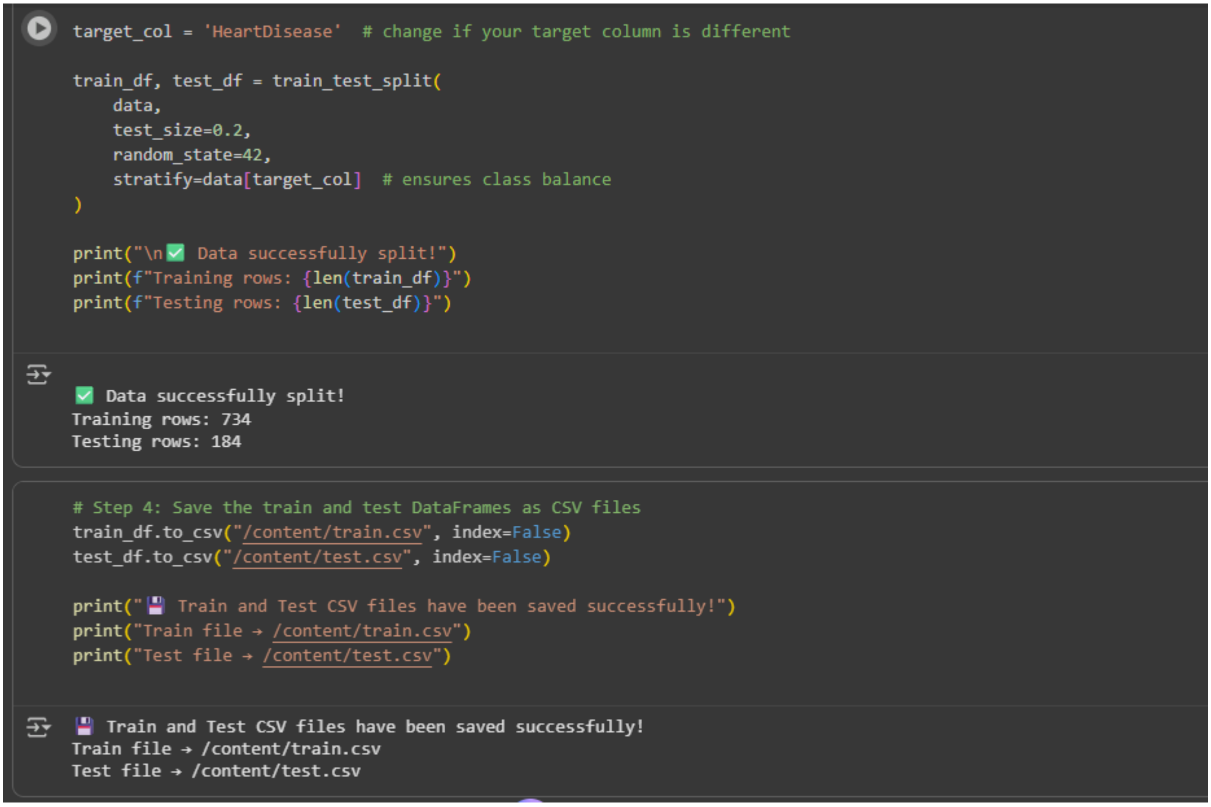
3. Encoding Categorical Columns



```
from sklearn.preprocessing import LabelEncoder

# Step 2: Encode non-numeric columns
le = LabelEncoder()
for col in data.columns:
    if data[col].dtype == 'object':  # columns with text
        data[col] = le.fit_transform(data[col])

print("\n✅ All categorical columns encoded!")
print(data.head())
```

```
✅ All categorical columns encoded!
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0   40    1              1        140          289          0           1
1   49    0              2        160          180          0           1
2   37    1              1        130          283          0           2
3   48    0              0        138          214          0           1
4   54    1              2        150          195          0           1

   MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
0    172               0      0.0         2             0
1    156               0      1.0         1             1
2     98               0      0.0         2             0
3    108               1      1.5         1             1
4    122               0      0.0         2             0
```

4. Splitting the training and testing data:



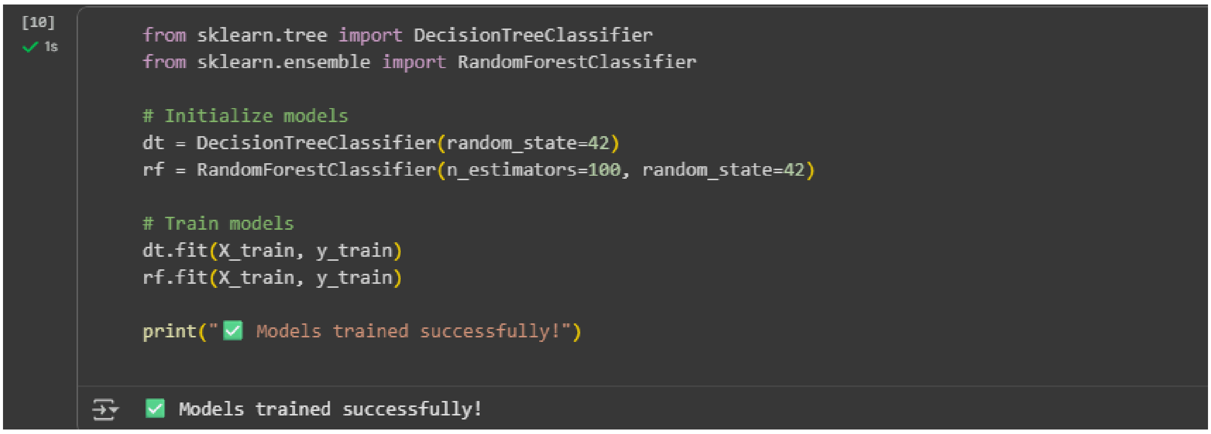
```
target_col = 'HeartDisease'  # change if your target column is different

train_df, test_df = train_test_split(
    data,
    test_size=0.2,
    random_state=42,
    stratify=data[target_col]  # ensures class balance
)

print("\n✅ Data successfully split!")
print(f"Training rows: {len(train_df)}")
print(f"Testing rows: {len(test_df)}")
```

```
✅ Data successfully split!
Training rows: 734
Testing rows: 184
```

```
# Step 4: Save the train and test DataFrames as CSV files
train_df.to_csv("/content/train.csv", index=False)
test_df.to_csv("/content/test.csv", index=False)

print("💾 Train and Test CSV files have been saved successfully!")
print("Train file → /content/train.csv")
print("Test file → /content/test.csv")
```

```
💾 Train and Test CSV files have been saved successfully!
Train file → /content/train.csv
Test file → /content/test.csv
```

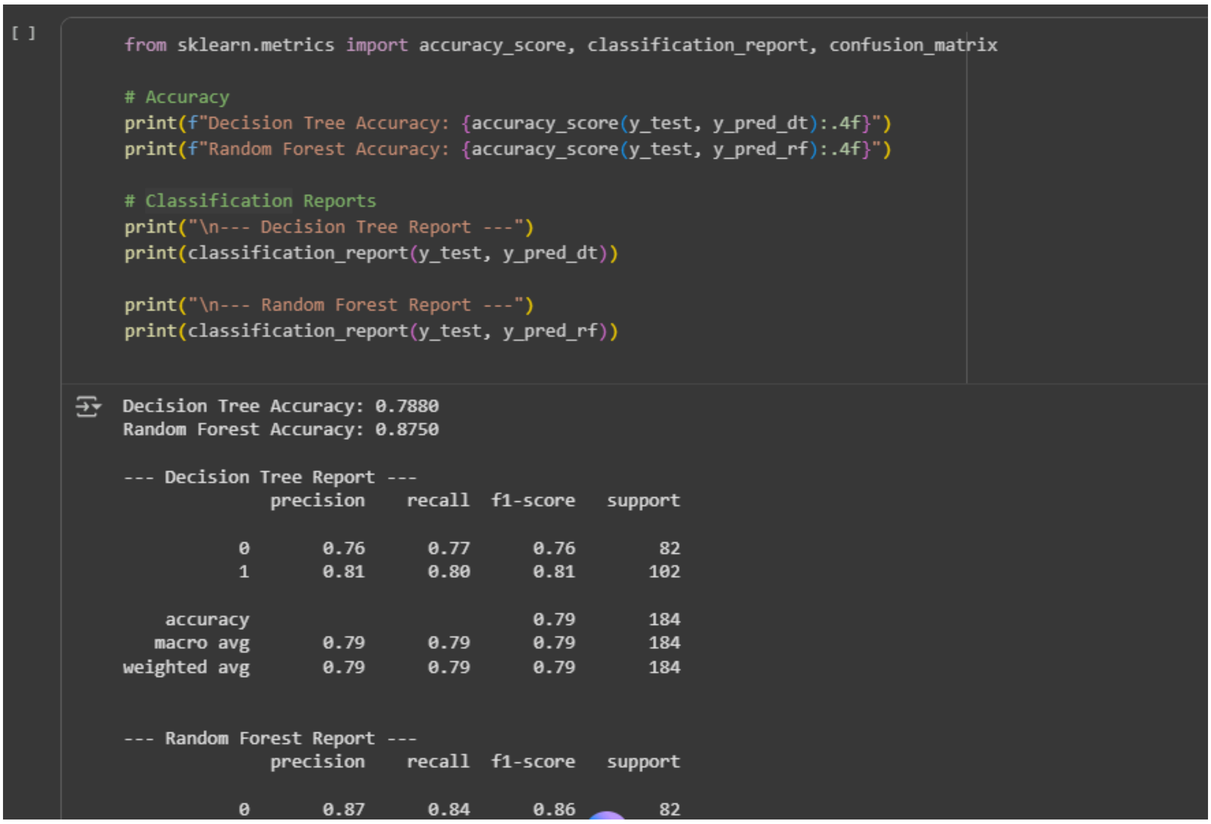
5. TRAINING DECISION TREE:



```
[10]
✓ 1s
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Initialize models
dt = DecisionTreeClassifier(random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train models
dt.fit(X_train, y_train)
rf.fit(X_train, y_train)

print("✅ Models trained successfully!")
```

```
✅ Models trained successfully!
```

6. PRINTING ACCURACY

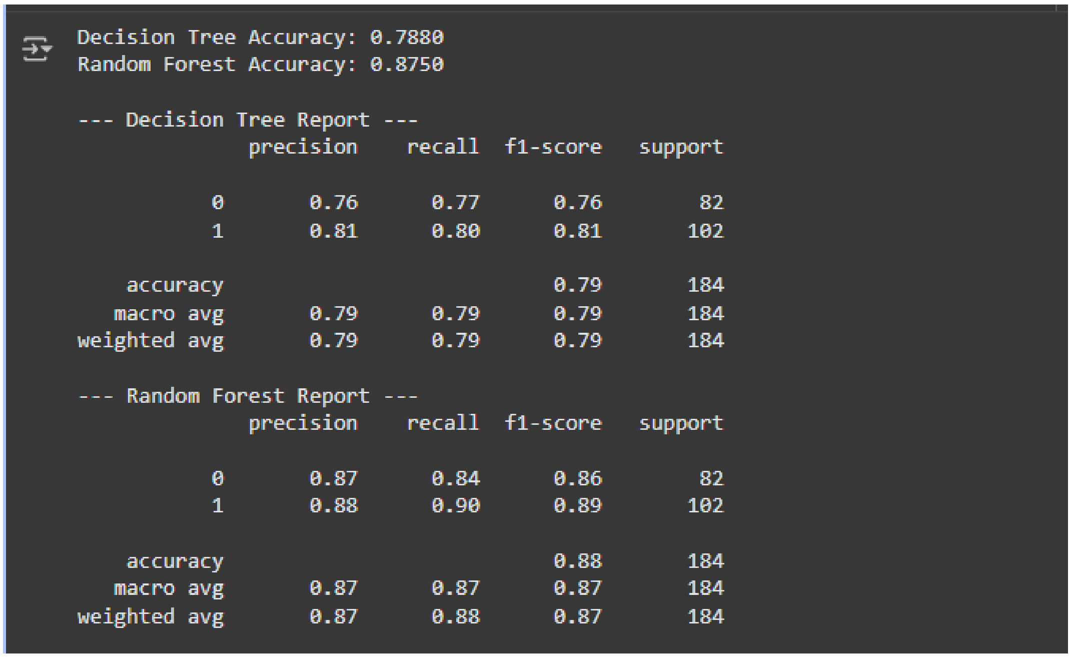


```
[ ]
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Accuracy
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}")
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")

# Classification Reports
print("\n--- Decision Tree Report ---")
print(classification_report(y_test, y_pred_dt))

print("\n--- Random Forest Report ---")
print(classification_report(y_test, y_pred_rf))
```

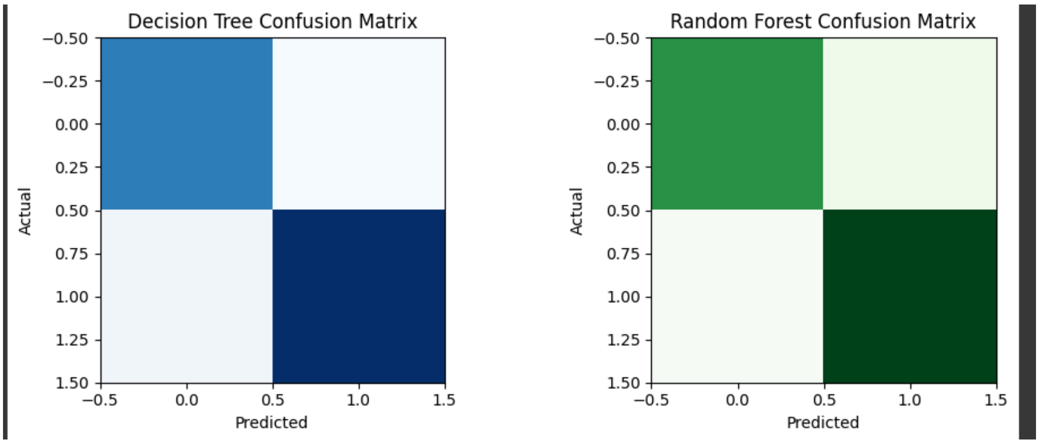
```
Decision Tree Accuracy: 0.7880
Random Forest Accuracy: 0.8750

--- Decision Tree Report ---
              precision    recall  f1-score   support

           0       0.76      0.77      0.76        82
           1       0.81      0.80      0.81       102

    accuracy                           0.79       184
   macro avg       0.79      0.79      0.79       184
weighted avg       0.79      0.79      0.79       184


--- Random Forest Report ---
              precision    recall  f1-score   support

           0       0.87      0.84      0.86        82
```

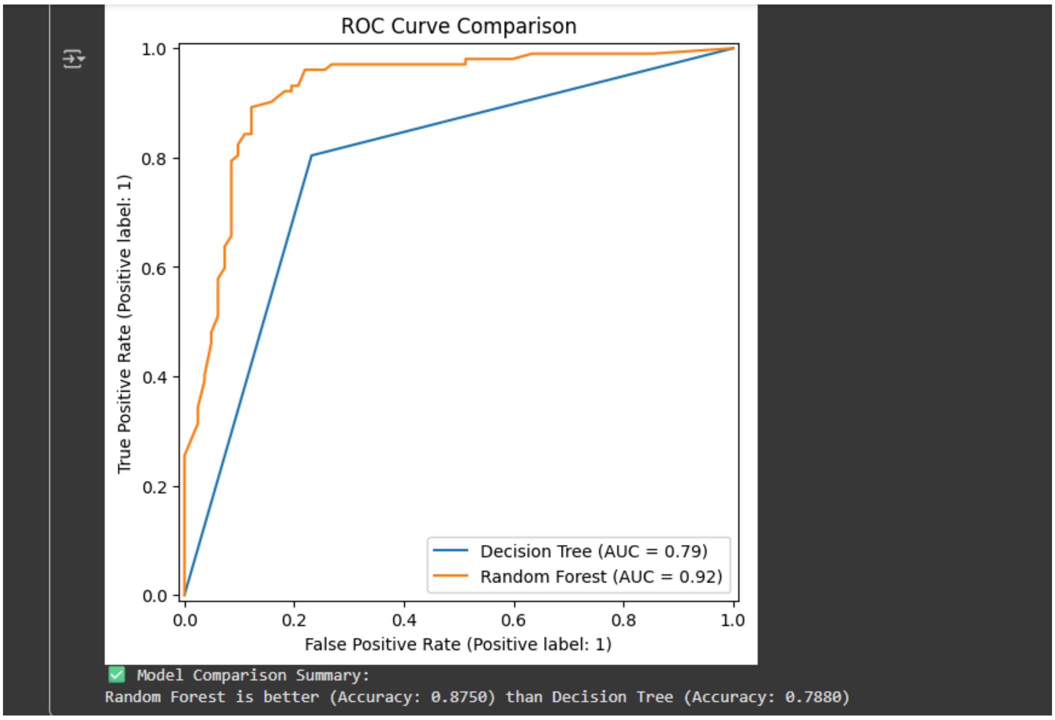
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, RocCurveDisplay
import matplotlib.pyplot as plt
# Step 1: Load pre-split CSVs
train_df = pd.read_csv("/content/train.csv")
test_df = pd.read_csv("/content/test.csv")
# Step 2: Separate features and target
target_col = 'HeartDisease'  # change if your CSV uses a different name
```

```
X_train = train_df.drop(target_col, axis=1)
y_train = train_df[target_col]
X_test = test_df.drop(target_col, axis=1)
y_test = test_df[target_col]
# Step 3: Train models
dt = DecisionTreeClassifier(random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42)

dt.fit(X_train, y_train)
rf.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
y_pre_rf = rf.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Decision Tree Accuracy: {accuracy_dt:.4f}")
print(f"Random Forest Accuracy: {accuracy_rf:.4f}\n")
print("--- Decision Tree Report ---")
print(classification_report(y_test, y_pred_dt))
print("--- Random Forest Report ---")
print(classification_report(y_test, y_pred_rf))
# Step 6: Confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(10,4))
cm_dt = confusion_matrix(y_test, y_pred_dt)
cm_rf = confusion_matrix(y_test, y_pred_rf)
axes[0].imshow(cm_dt, cmap='Blues')
axes[0].set_title("Decision Tree Confusion Matrix")
axes[1].imshow(cm_rf, cmap='Greens')
axes[1].set_title("Random Forest Confusion Matrix")
for ax in axes:
    ax.set_xlabel("Predicted")
    ax.set_ylabel("Actual")

plt.tight_layout()
plt.show()
plt.figure(figsize=(6,6))
RocCurveDisplay.from_estimator(dt, X_test, y_test, name="Decision Tree", ax=plt.gca())
RocCurveDisplay.from_estimator(rf, X_test, y_test, name="Random Forest", ax=plt.gca())
plt.title("ROC Curve Comparison")
plt.show()
print("✅ Model Comparison Summary:")
if accuracy_rf > accuracy_dt:
    print(f"Random Forest is better (Accuracy: {accuracy_rf:.4f}) than Decision Tree (Accuracy: {accuracy_dt:.4f})")
else:
    print(f"Decision Tree is better (Accuracy: {accuracy_dt:.4f}) than Random Forest (Accuracy: {accuracy_rf:.4f})")
```



```
Decision Tree Accuracy: 0.7880
Random Forest Accuracy: 0.8750

--- Decision Tree Report ---
              precision    recall  f1-score   support

           0       0.76      0.77      0.76        82
           1       0.81      0.80      0.81       102

    accuracy                           0.79       184
   macro avg       0.79      0.79      0.79       184
weighted avg       0.79      0.79      0.79       184

--- Random Forest Report ---
              precision    recall  f1-score   support

           0       0.87      0.84      0.86        82
           1       0.88      0.90      0.89       102

    accuracy                           0.88       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.88      0.87       184
```




ROC Curve Comparison
True Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
Decision Tree (AUC = 0.79)
Random Forest (AUC = 0.92)

```
✅ Model Comparison Summary:
Random Forest is better (Accuracy: 0.8750) than Decision Tree (Accuracy: 0.7880)
```