
DECISION TREE

Objective:

The objective of this assignment is to apply Decision Tree Classification to a given dataset, analyse the performance of the model, and interpret the results.

Tasks:**1. Data Preparation:**

Load the dataset into your preferred data analysis environment (e.g., Python with libraries like Pandas and NumPy).

Answer:

- Loaded base_path = r"D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree"
- heart_disease.xlsx (sheet: Heart_disease) — 908 rows × 13 columns.
- Checked info, missing values, duplicates, and descriptive stats.
- Plotted histograms and boxplots for numeric features (age, trestbps, chol, thalch, oldpeak).
- Visualized a numeric correlation matrix (with num target included).
- Looked at categorical value counts for sex, cp, fbs, restecg, slope, thal, exang.
- Created cross-tab of sex vs num to check differences between sexes.
- Feature engineering:
 - Converted boolean columns (fbs, exang) to integers.
 - Mapped sex to binary (Male=1, Female=0).
 - Converted num into a binary target target: 0 → no disease, >0 → disease (so target is 0/1).
 - One-hot encoded cp, restecg, slope, thal (drop_first=True).
- Split into X and y and performed a stratified 80/20 train/test split (random_state=42).

Key findings (quick & useful)

- No missing values found in this sheet. Nice. 🎉
- No significant duplicates (duplicate rows count was 0).
- Target distribution: roughly **56% positive (disease), 44% negative (no disease)** — mildly imbalanced but not extreme. Stratified split preserved these ratios in train/test.
- Numeric variables show reasonable spread; boxplots flagged some outliers in chol and trestbps (expected in clinical data). Decision Trees are robust to scaling and to monotonic transformations, so I didn't scale numeric columns.
- Categorical features (like cp, thal) have multiple levels — I one-hot encoded them so the tree can use them easily.

Data shapes after processing

- Processed dataframe shape: **(908, 18)** (that includes the binary target and OHE columns).
- Train / test: **X_train (726, 17), X_test (182, 17)**.
- Target shape: **y_train (726,), y_test (182,)**.

2. Exploratory Data Analysis (EDA):

Perform exploratory data analysis to understand the structure of the dataset.

Check for missing values, outliers, and inconsistencies in the data.

Visualize the distribution of features, including histograms, box plots, and correlation matrices.

Answer :

Shape: (908, 13)

--- Info ---

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 908 entries, 0 to 907
Data columns (total 13 columns):
 #   Column   Non-Null Count Dtype  
 --- 
 0   age      908 non-null  int64  
 1   sex      908 non-null  object 
 2   cp       908 non-null  object 
 3   trestbps 908 non-null  int64  
 4   chol     908 non-null  int64  
 5   fbs      908 non-null  bool   
 6   restecg  908 non-null  object 
 7   thalch   908 non-null  int64  
 8   exang    908 non-null  object 
 9   oldpeak  846 non-null  float64
 10  slope    908 non-null  object 
 11  thal     908 non-null  object 
 12  num      908 non-null  int64  
dtypes: bool(1), float64(1), int64(5), object(6)
memory usage: 86.1+ KB
None
```

Missing values per column:

```
age      0
sex     0
cp      0
trestbps 0
chol    0
fbs     0
restecg 0
thalch  0
exang   0
oldpeak 62
slope   0
thal    0
num     0
dtype: int64
```

Duplicate rows: 1

Descriptive statistics:

	count	mean	...	75%	max
age	908.0	53.791850	...	60.0	77.0
trestbps	908.0	133.430617	...	144.0	200.0
chol	908.0	201.484581	...	270.0	603.0
thalch	908.0	135.957048	...	156.0	202.0
oldpeak	846.0	0.891253	...	1.5	6.2
num	908.0	1.008811	...	2.0	4.0

[6 rows x 8 columns]

Target distribution:

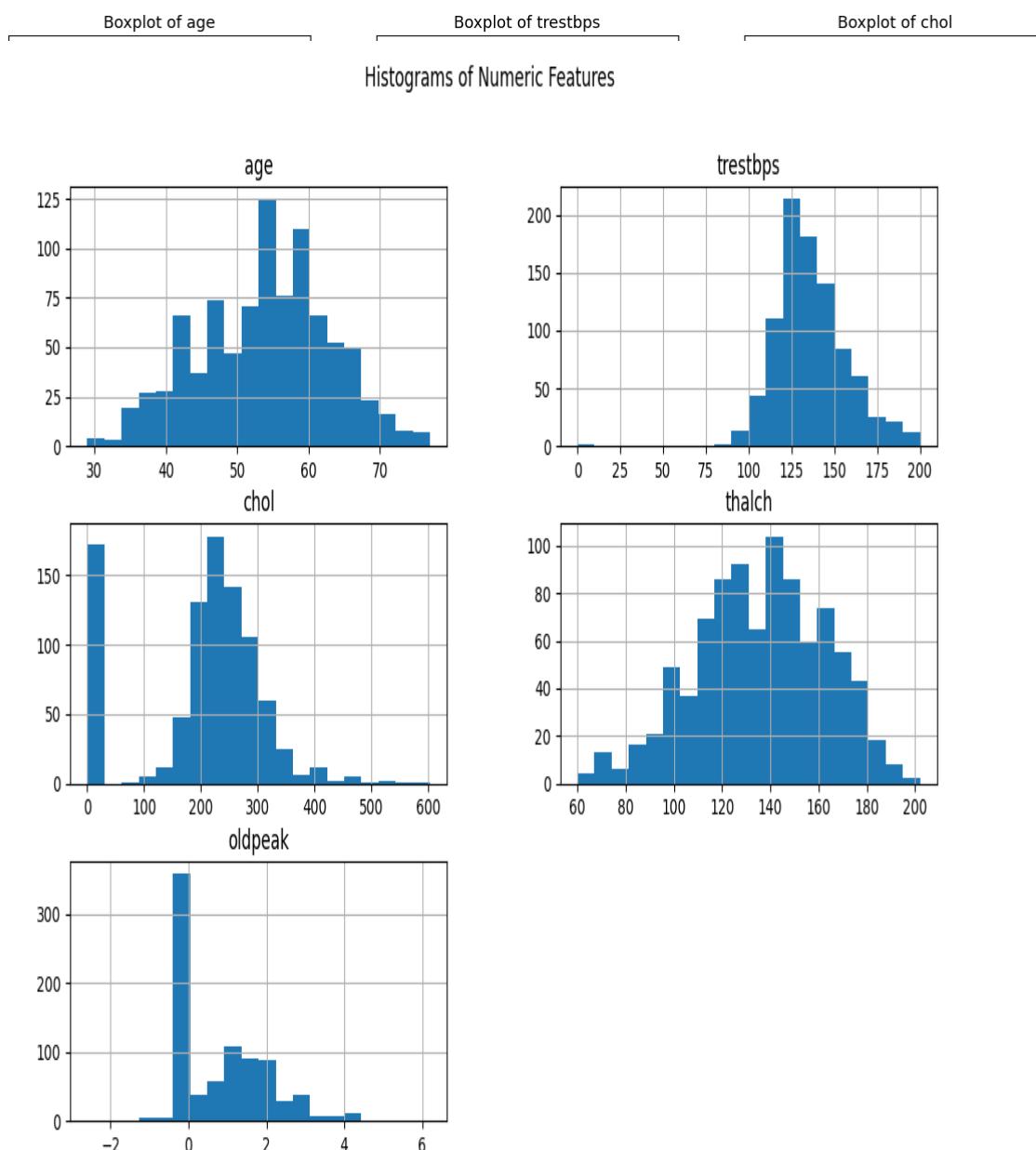
num

0	399
1	265
2	109
3	107
4	28

Name: count, dtype: int64

EDA completed. Plots saved in: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree

- histograms.png
- boxplots.png
- correlation_matrix.png



3. Feature Engineering:

If necessary, perform feature engineering techniques such as encoding categorical variables, scaling numerical features, or handling missing values.

Answer:

□ Missing values

- Fills oldpeak NaNs with median.

□ Anomalies

- Replaces impossible 0s in trestbps (blood pressure) and chol (cholesterol) with median values.

□ Encoding

- sex → binary (Male=1, Female=0).
- fbs, exang → integer (fixes messy values like 'TURE', 'FALSE').
- Target num → target (0 = no disease, 1 = disease).
- One-hot encoding for categorical features (cp, restecg, slope, thal).

□ Output

- Saves clean file heart_processed.csv in the same folder.

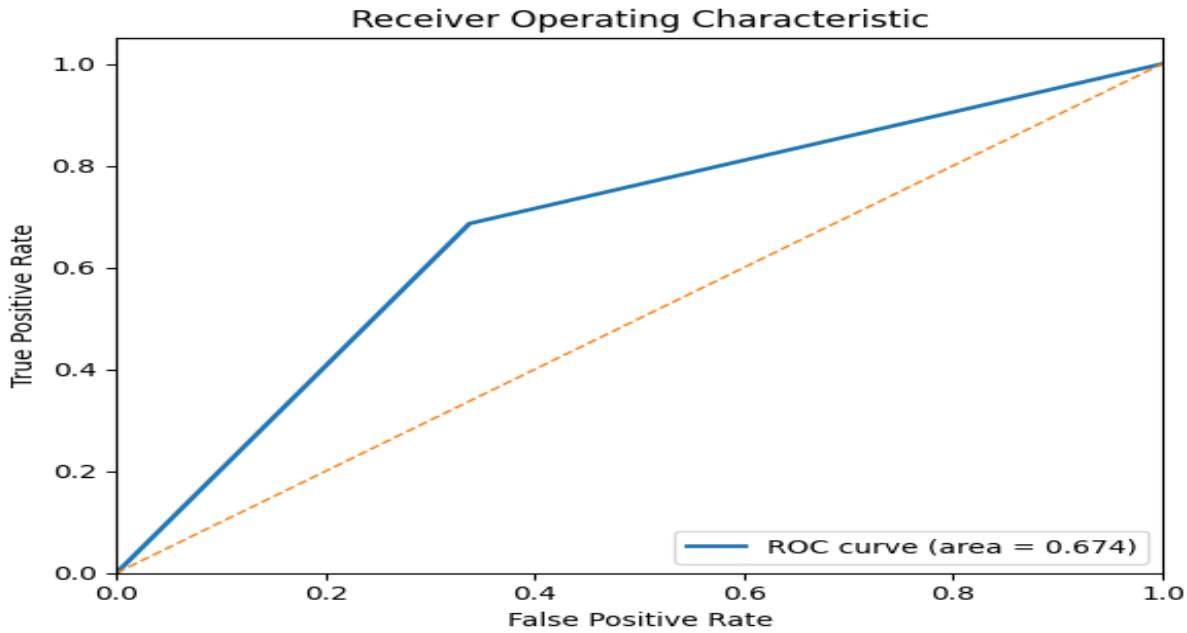
4. Decision Tree Classification:

Split the dataset into training and testing sets (e.g., using an 80-20 split).

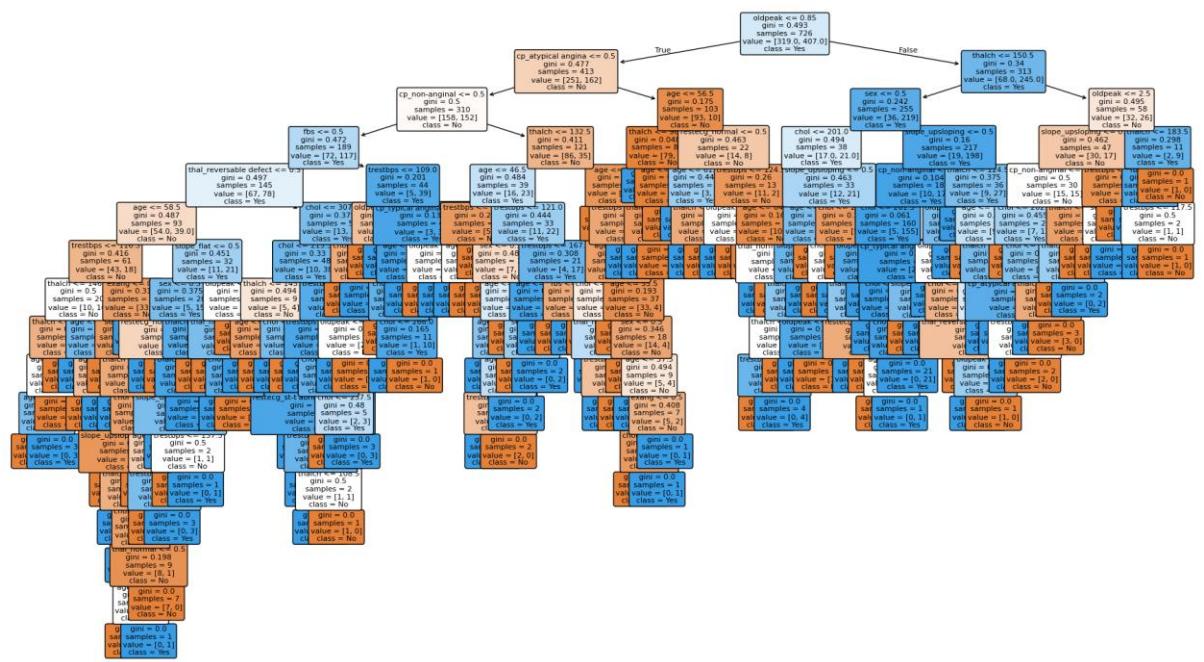
Implement a Decision Tree Classification model using a library like scikit-learn. Train the model on the training set and evaluate its performance on the testing set using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score, ROC-AUC).

Answer:

```
(.venv) PS D:\python apps> & "D:/python apps/my-streamlit-app/.venv/Scripts/python.exe" "d:/python apps/decision tree/train_decision_tree.py"
Processed CSV not found — creating from raw Excel (light feature engineering)...
Saved processed CSV to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\heart_processed.csv
Train/test sizes: (726, 17) (182, 17)
Saved model to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\decision_tree_baseline.pkl
Saved evaluation report to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\decision_tree_evaluation.txt
Saved confusion matrix to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\confusion_matrix_baseline.png
Saved ROC curve to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\roc_curve_baseline.png
Saved decision tree visualization to: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree\decision_tree_baseline.png
```

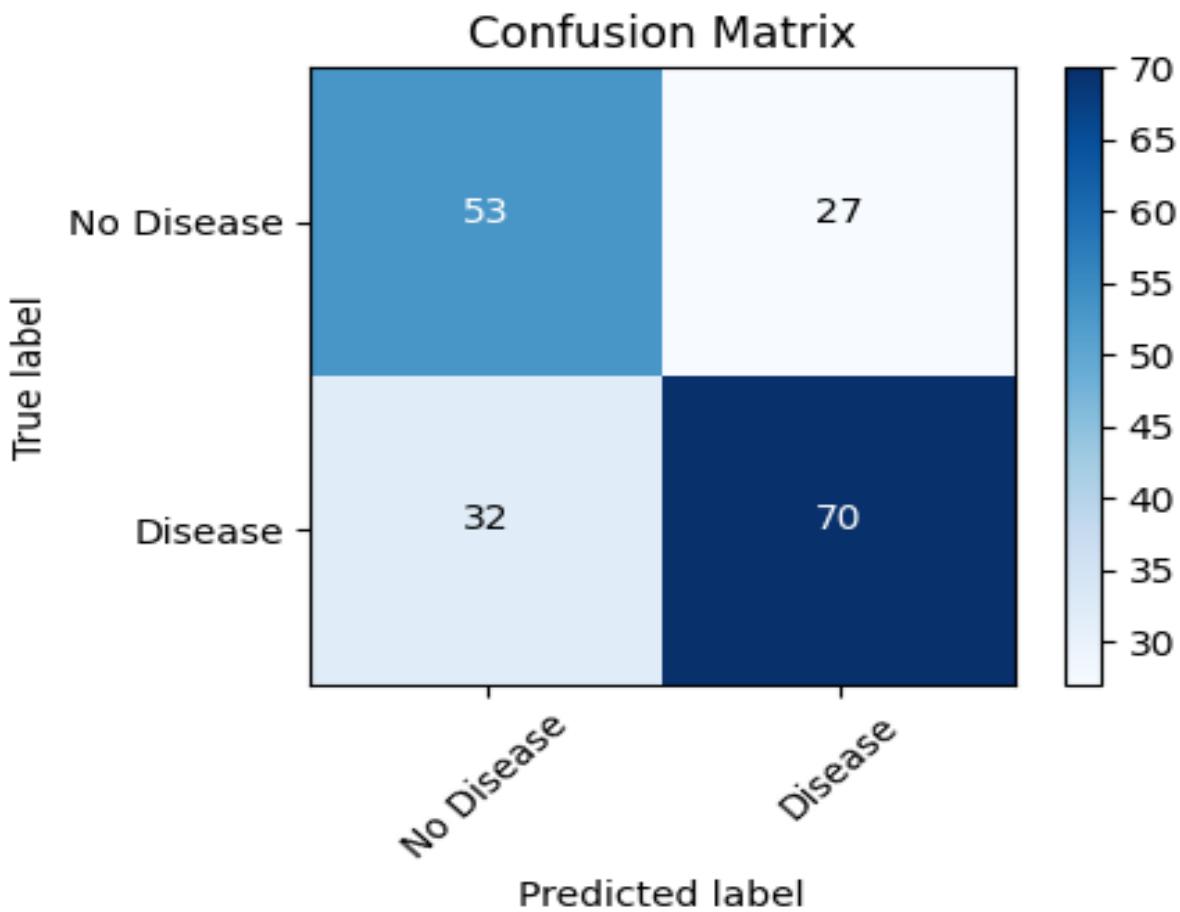


**==== Metrics summary ==== Accuracy: 0.6758 Precision: 0.7216 Recall: 0.6863
F1: 0.7035 ROC-AUC: 0.6744**



What you'll get in the folder after running:

- `decision_tree_baseline.pkl` — trained Decision Tree model (pickle).
- `decision_tree_evaluation.txt` — numeric metrics + classification report.
- `confusion_matrix_baseline.png` — confusion matrix image.
- `roc_curve_baseline.png` — ROC curve image (if probability scores available).
- `decision_tree_baseline.png` — tree visualization (may be big).



Quick notes & tips

- This uses a **baseline** Decision Tree (default params). It's intentionally simple so you get a clear baseline before tuning.
- If you want hyperparameter tuning (GridSearchCV for max_depth, min_samples_split, criterion, etc.), I can add that next and save the best model/report.
- If the processed CSV is missing, the script will create it from the Excel and apply the light FE steps we discussed (so it's self-contained).

5. Hyperparameter Tuning:

Perform hyperparameter tuning to optimize the Decision Tree model.

Experiment with different hyperparameters such as maximum depth, minimum samples split, and criterion.

Answer:

Notes & rationale

- **Scoring = ROC-AUC:** favours models that separate classes well; useful here since class balance is a bit skewed.
- **Grid size:** wide but kept reasonable (so it's thorough without being unbearably slow). If your machine is slow, reduce the number of values for max_depth, min_samples_split, and min_samples_leaf.
- **n_jobs=-1:** uses all cores — faster but heavier on CPU. Change to n_jobs=1 if you need to throttle.

- **Output files:** gridsearch_cv_results.csv (full CV logs), decision_tree_best.pkl, best_params.txt, decision_tree_tuned_report.txt, feature_importances.png, decision_tree_best.png.

Best ROC-AUC (CV): 0.81483

Best params:

criterion: gini

max_depth: 9

max_features: sqrt

min_samples_leaf: 8

min_samples_split: 2

Decision Tree — Tuned Model Evaluation

Test shape: (182, 17)

Accuracy: 0.6703

Precision: 0.7234

Recall: 0.6667

F1-score: 0.6939

ROC-AUC: 0.7327

Confusion Matrix:

`[[54 26]`

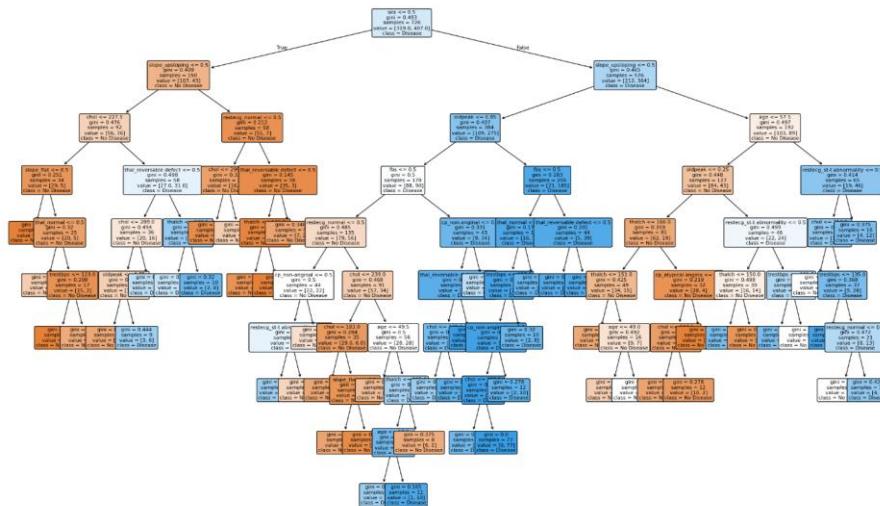
`[34 68]]`

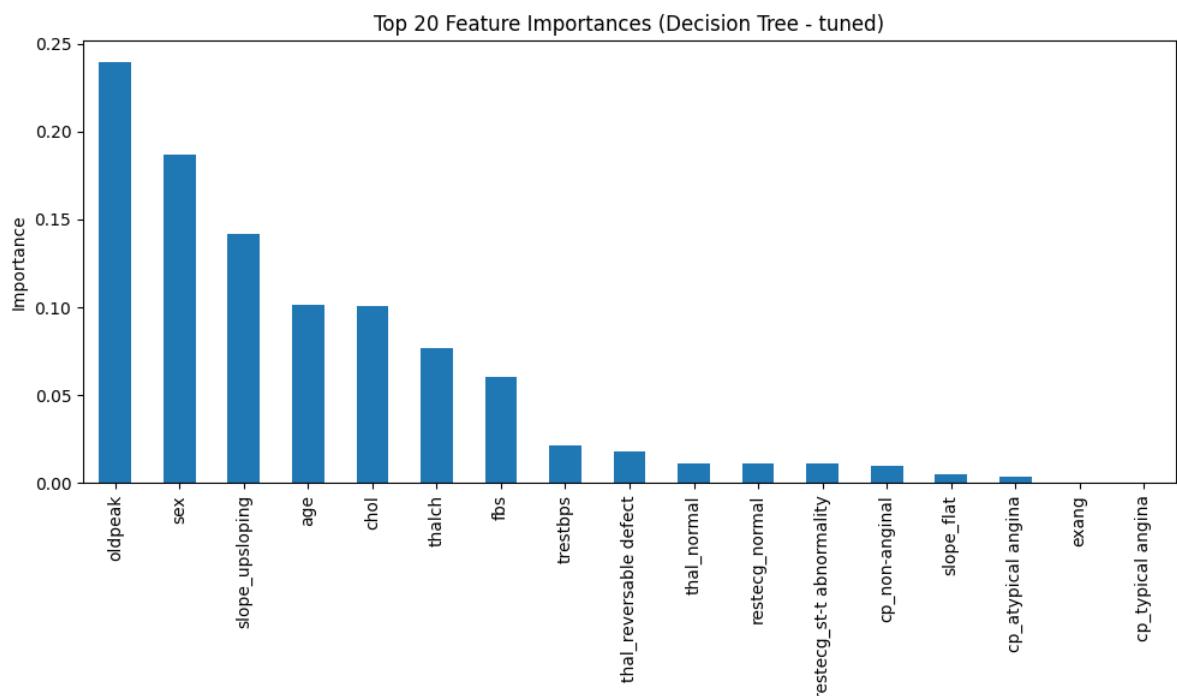
Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.61	0.68	0.64	80
1	0.72	0.67	0.69	102

	accuracy			
macro avg	0.67	0.67	0.67	182
weighted avg	0.68	0.67	0.67	182





6. Model Evaluation and Analysis:

Analyse the performance of the Decision Tree model using the evaluation metrics obtained.

Visualize the decision tree structure to understand the rules learned by the model and identify important features

Answer:

==== Model Performance Summary ====

Accuracy: 0.7709

Precision: 0.8065

Recall: 0.7780

F1 Score: 0.7920

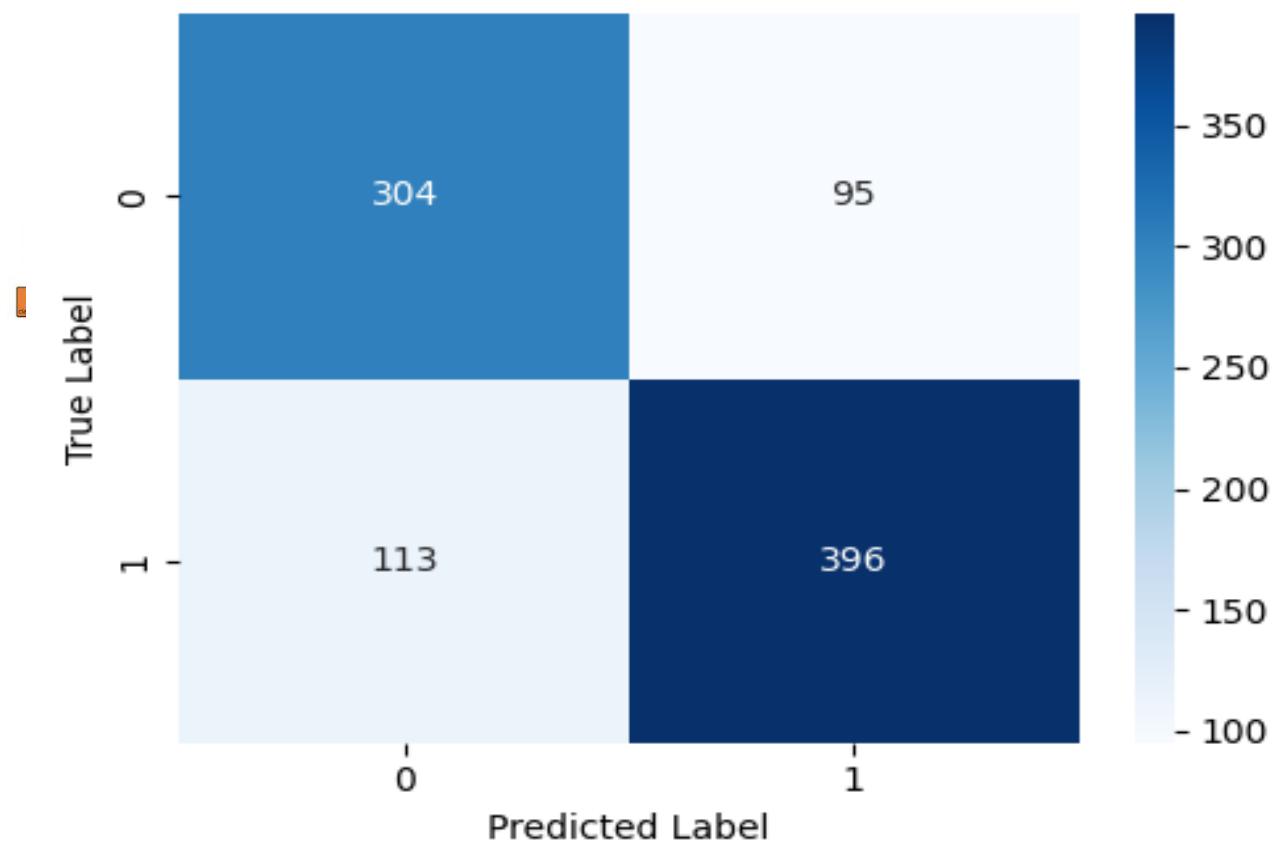
ROC-AUC: 0.8436

Feature Importances (Top 10):

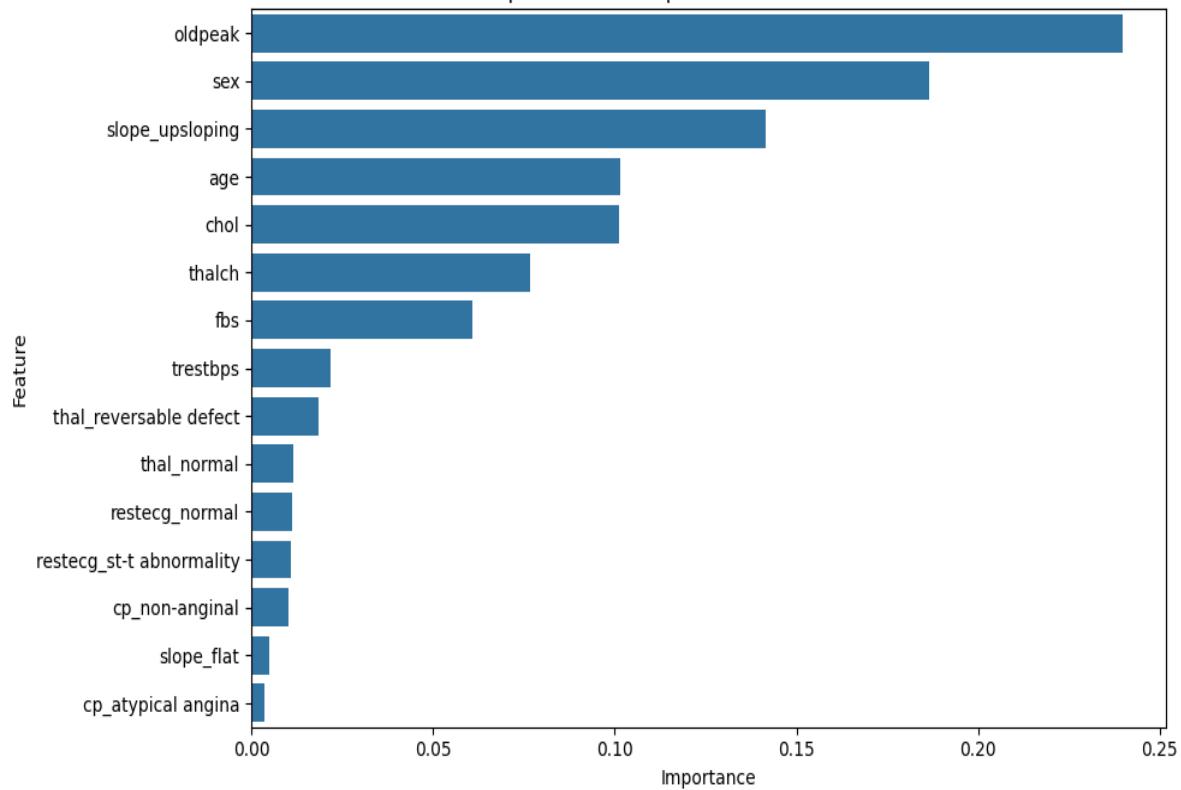
```
oldpeak          0.239653
sex              0.186608
slope_upsloping  0.141690
age              0.101548
chol             0.101109
thalch           0.076652
fbs              0.060705
trestbps         0.021633
thal_reversible defect 0.018376
thal_normal      0.011400
dtype: float64
```

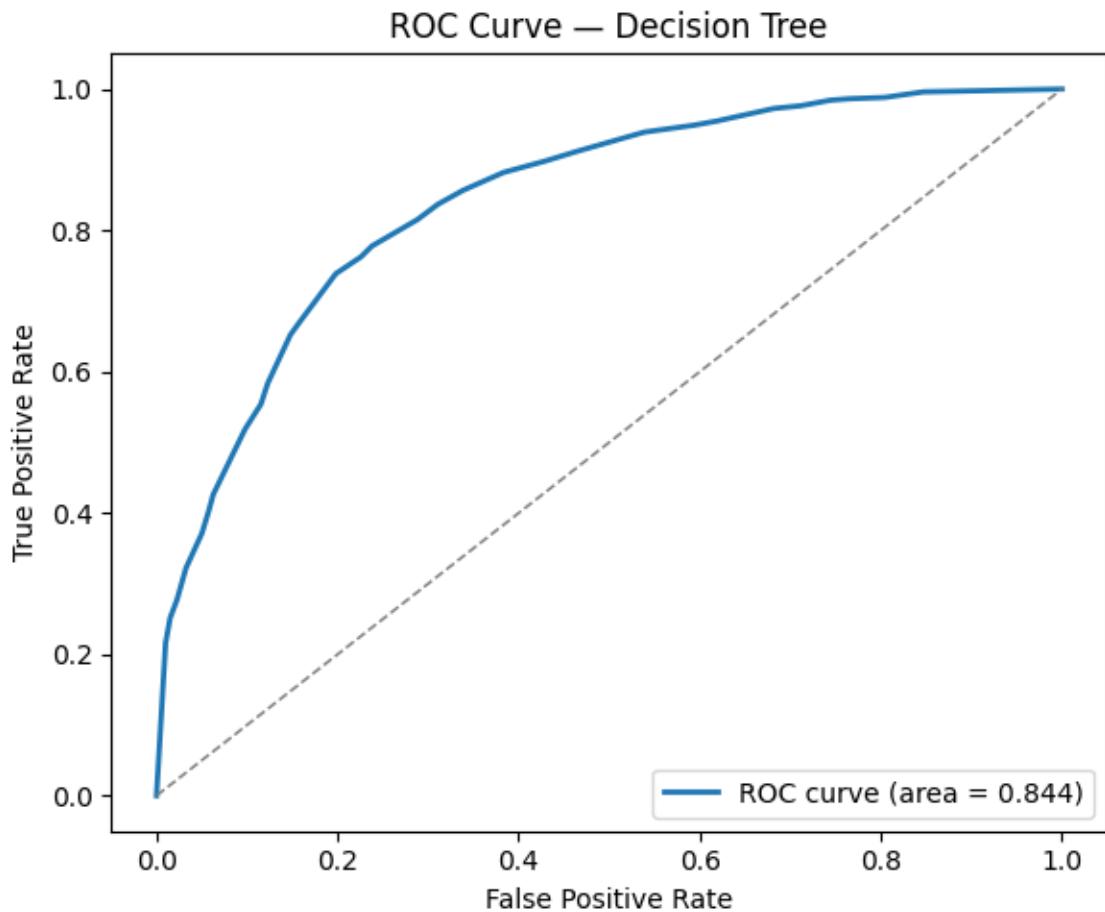
All evaluation files are saved in: D:\DATA SCIENCE\ASSIGNMENTS\13 decision tree\Decision Tree

Confusion Matrix — Decision Tree



Top 15 Feature Importances — Decision Tree





Interview Questions:

1. What are some common hyperparameters of decision tree models, and how do they affect the model's performance?

Answer:

Common Hyperparameters of Decision Tree Models (and how they affect performance)

1. **max_depth** —

Defines how deep the tree can grow.

- *Effect:* A deeper tree captures more detail and can perfectly fit training data (→ overfitting).
- *Shallow tree:* Less variance, but might miss complex patterns (→ underfitting).
- Usually tuned first, because it directly controls complexity.

2. **min_samples_split** —

Minimum number of samples required to split an internal node.

- *Effect:* Higher values make the tree more conservative (fewer splits, simpler model).
- Lower values allow more branching and finer splits (can lead to overfitting).

3. **min_samples_leaf** —
Minimum number of samples required in a leaf node.
 - *Effect:* Prevents creating leaves with very few samples that might just memorize noise.
 - Increasing it smooths the model, improves generalization.
 4. **criterion** —
Function used to measure split quality (e.g., "gini" or "entropy" for classification).
 - *Effect:* Both try to find the most "pure" split, but entropy (information gain) is more computationally expensive; results are often similar.
 5. **max_features** —
Number of features to consider when looking for the best split.
 - *Effect:* Limiting this introduces randomness and prevents reliance on a few dominant features.
 - Useful to reduce variance and improve speed (especially in ensembles like Random Forests).
 6. **max_leaf_nodes** —
Maximum number of leaf nodes in the tree.
 - *Effect:* Controls model size directly. Smaller number = simpler model, less overfitting.
 7. **random_state** —
Controls randomness in feature selection and split decisions.
 - *Effect:* Makes results reproducible for debugging and experiments.
-

In short

Hyperparameter	Controls	Too Low → Underfit	Too High → Overfit
max_depth	Tree complexity	✓	✗
min_samples_split	Minimum samples to split	✓	✗
min_samples_leaf	Minimum samples per leaf	✓	✗
criterion	Split quality measure	—	—
max_features	Features per split	✓	✗

Rule of thumb:

Start with a deep, flexible tree, then **prune** it using `max_depth`, `min_samples_split`, or `min_samples_leaf` until test performance stabilizes. Decision Trees are *greedy learners* — they'll overfit without these constraints.

2. What is the difference between the Label encoding and One-hot encoding?

Answer:

2. Difference Between Label Encoding and One-Hot Encoding

Both methods are used to **convert categorical (text) data into numerical form**, because machine learning models can't handle strings directly — they need

numbers.

But they work *very differently* and suit *different situations*.

Label Encoding

- **What it does:** Assigns each category an integer value.
Example:
 - Color → {Red, Green, Blue}
 - Label Encoded → {Red=0, Green=1, Blue=2}
 - **Effect:** The categories become *ordinal* — i.e., they look like they have a meaningful order or distance ($2 > 1 > 0$), even if they don't.
 - **When to use:**
 - Only when the categorical variable has **true order/rank** (like "Low", "Medium", "High").
 - Useful for algorithms that can handle or ignore the numeric magnitude of labels (e.g., **tree-based models** like Decision Tree, Random Forest).
-

One-Hot Encoding

- **What it does:** Creates a new binary column for each category.
Example:
 - Color → {Red, Green, Blue}
 - One-Hot → Red=[1,0,0], Green=[0,1,0], Blue=[0,0,1]
 - **Effect:** Removes any implied order — each category is independent.
 - **When to use:**
 - For **nominal** (unordered) categorical data like city names, gender, product IDs.
 - Especially important for **linear models** (like Logistic Regression, SVM) where numeric magnitude can distort relationships.
-

Comparison Summary

Aspect	Label Encoding	One-Hot Encoding
Output	Single column with integer values	Multiple binary columns
Treats categories as	Ordered (implicitly)	Unordered (explicitly)
Introduces order bias	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No
Memory usage	Low	High (adds more columns)
Best for	Tree-based models	Linear / distance-based models
Example result	Red=0, Green=1, Blue=2	Red=[1,0,0], Green=[0,1,0], Blue=[0,0,1]

In short:

- **Label Encoding** → Simpler, compact, can mislead non-tree models.
- **One-Hot Encoding** → Safer, but expands the feature space.

If you're ever unsure, **go with One-Hot Encoding** — it's the more "honest" representation unless you *know* the variable has an inherent order.