Experiment - 3

Nikhil Kale
D15A/50

Aim - Apply Decision Tree and Random Forest for classification tasks

1. Dataset Source

Dataset used:
https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Title: Heart Disease Dataset

2. Dataset Description

The dataset contains medical attributes used to predict the presence of heart disease.

Number of records: 303
Number of features: 13 input features
Target variable: 1 output variable

Target Variable:

- 0 → No Heart Disease
- 1 → Heart Disease

Features include:

- age
- sex
- chest pain type (cp)
- resting blood pressure (trestbps)
- cholesterol (chol)
- fasting blood sugar (fbs)
- resting ECG (restecg)
- maximum heart rate (thalach)
- exercise induced angina (exang)
- oldpeak
- slope
- number of major vessels (ca)

- thal

Dataset Characteristics:

- All numeric features
- No missing values
- Binary classification problem

3. Mathematical Formulation

Decision Tree

Uses entropy or Gini index.

Gini Index:

$$Gini = 1 - \sum p_i^2$$

Entropy:

$$Entropy = -\sum p_i \log_2 p_i$$

Goal:
Select splits that minimize impurity.

Random Forest

Ensemble of Decision Trees.

Prediction:

$$\hat{y} = \text{Majority Voting of Trees}$$

Reduces variance and overfitting.

4. Algorithm Limitations

Decision Tree

- Prone to overfitting
- High variance
- Sensitive to noise

Random Forest

- Computationally expensive
- Less interpretable
- Large memory usage

5. Methodology / Workflow

1. Dataset loading
2. Data exploration
3. Train-test split
4. Feature scaling
5. Model training
6. Hyperparameter tuning
7. Performance evaluation
8. Model comparison

Workflow:

Dataset → Preprocessing → Split → Scaling → Train Model → Tune → Evaluate → Compare

6. Performance Analysis

Evaluation Metrics Used:

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix

Observations:

- Random Forest usually achieves higher accuracy
- Decision Tree may overfit

- Random Forest generalizes better

7. Hyperparameter Tuning

Decision Tree tuned parameters:

- max_depth
- min_samples_split
- criterion

Random Forest tuned parameters:

- n_estimators
- max_depth
- min_samples_split

GridSearchCV with 5-fold cross validation used.

Tuning improves generalization and prevents overfitting.

Code and Output -

[1] 2s

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

plt.style.use('seaborn-v0_8')
sns.set_palette("Set2")
```

[2] 0s

```
df = pd.read_csv('/content/heart.csv')
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

[3]

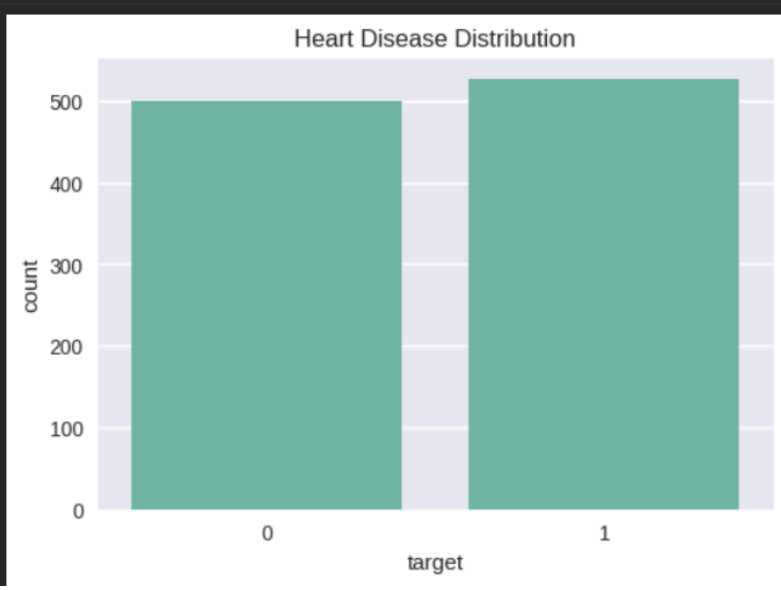
```
df.info()
df.describe()
df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

	0
age	0
sex	0
cp	0

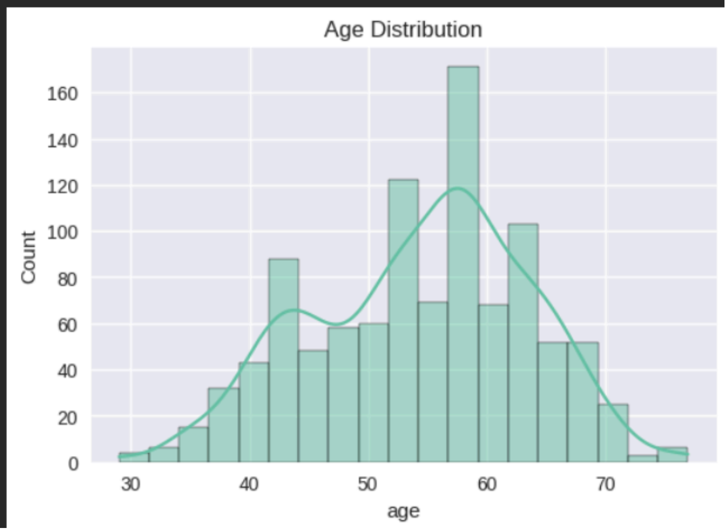
[4]

```
plt.figure(figsize=(6,4))
sns.countplot(x='target', data=df)
plt.title("Heart Disease Distribution")
plt.show()
```



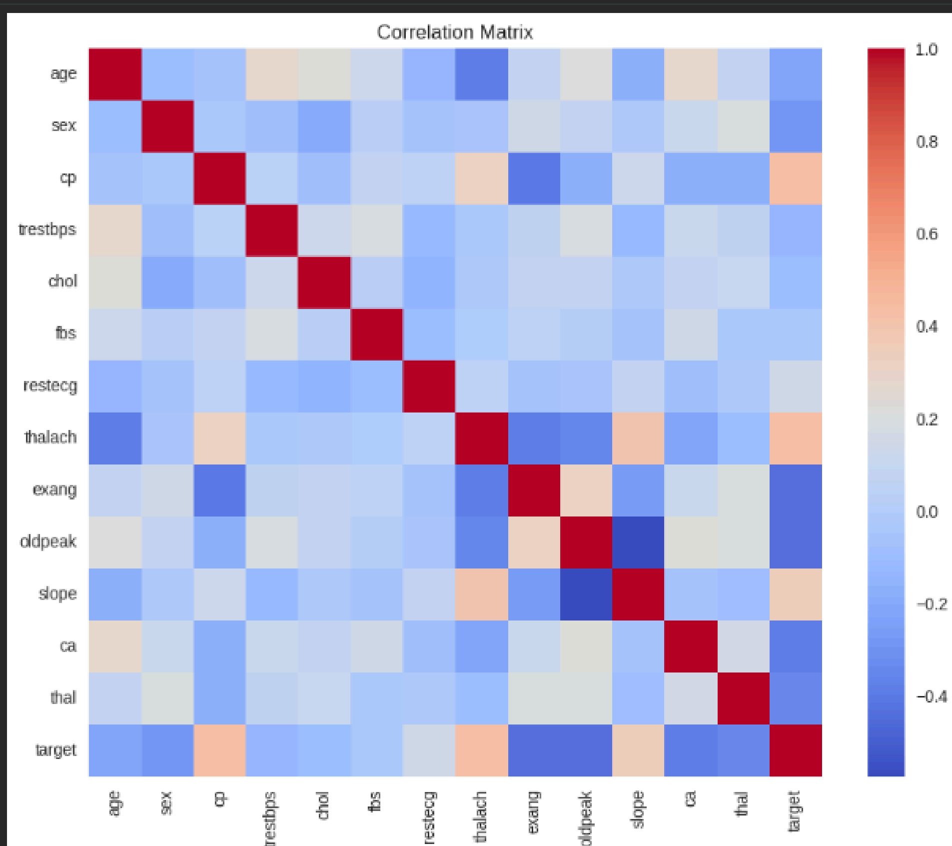
[5]
0s

```
plt.figure(figsize=(6,4))
sns.histplot(df['age'], kde=True)
plt.title("Age Distribution")
plt.show()
```



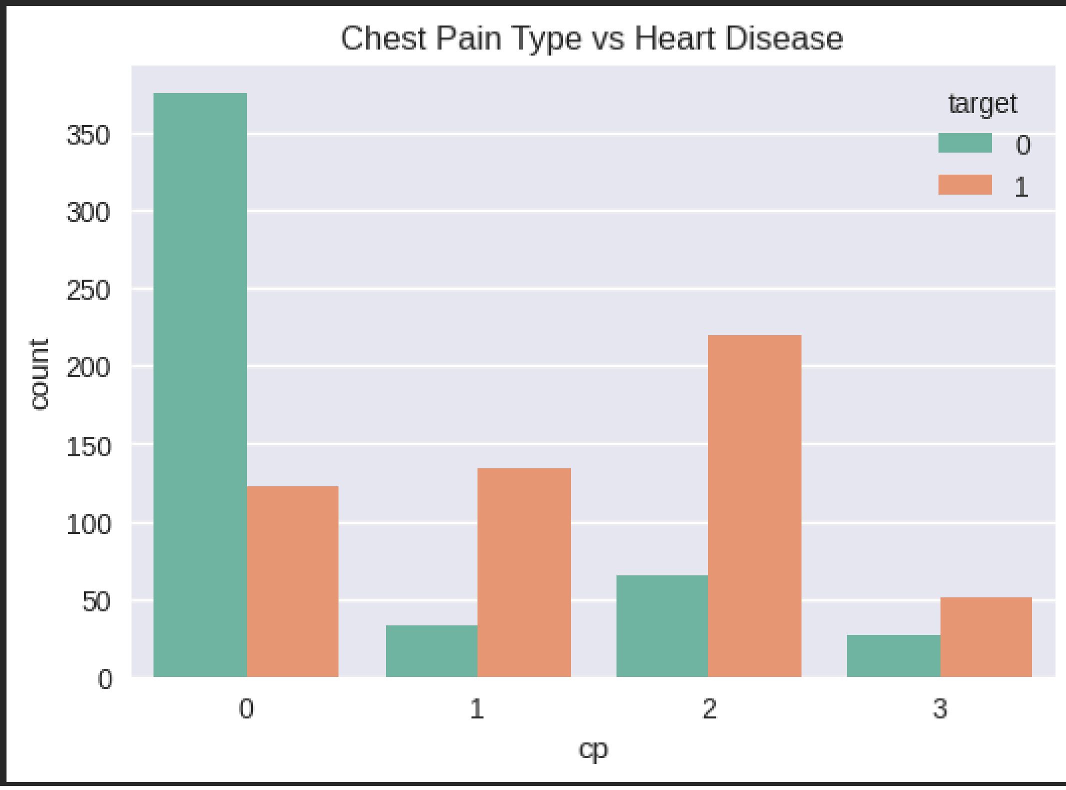
[6]
0s

```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



[7]
✓ 0s

```
plt.figure(figsize=(6,4))
sns.countplot(x='cp', hue='target', data=df)
plt.title("Chest Pain Type vs Heart Disease")
plt.show()
```



[8]
✓ 0s

```
X = df.drop('target', axis=1)
y = df['target']
```

[9]
✓ 0s

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

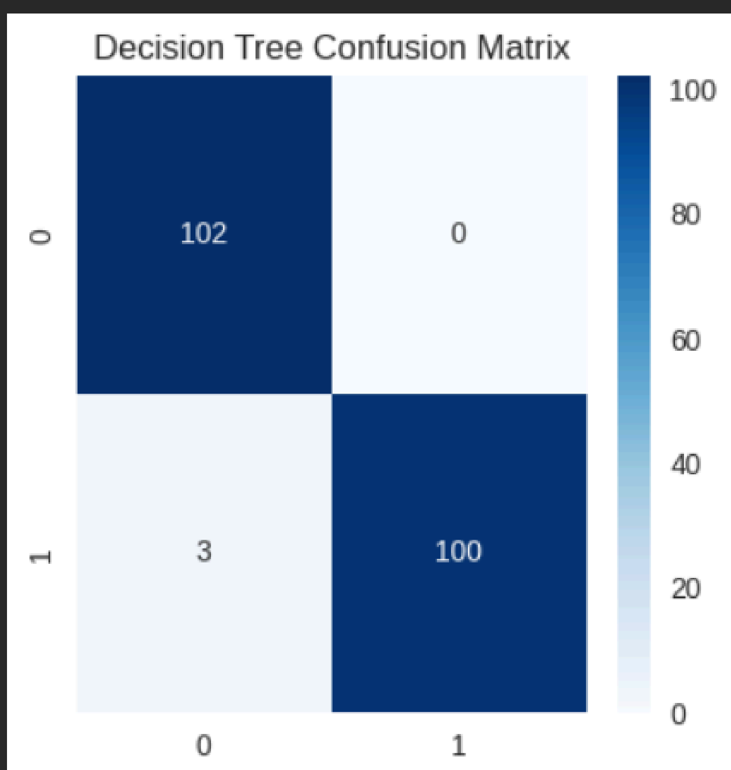
[10]
✓ 0s

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
[11]
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.9853658536585366
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

```
[12]
plt.figure(figsize=(4,4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues')
plt.title("Decision Tree Confusion Matrix")
plt.show()
```



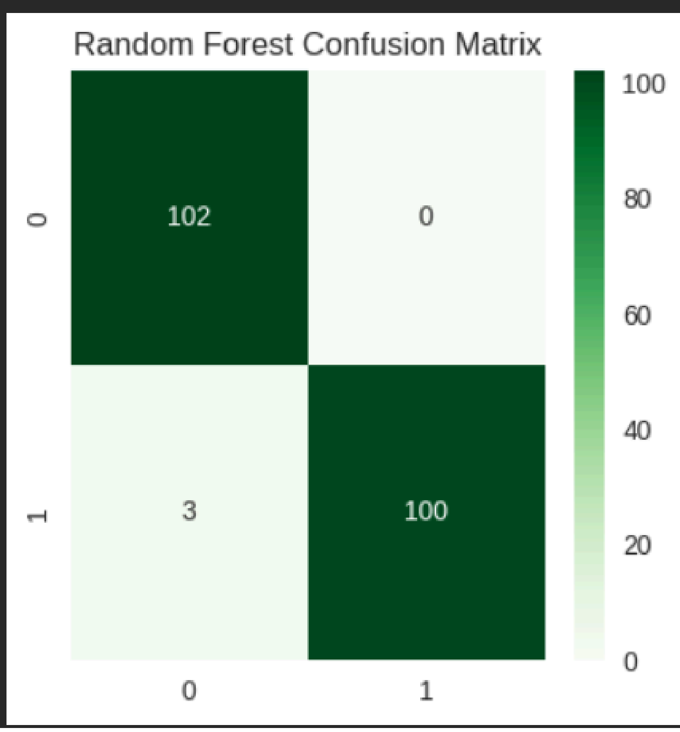
[13]
✓ 0s

```python
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.9853658536585366
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205
```

[14]
✓ 0s

```python
cm_rf = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(4,4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Greens')
plt.title("Random Forest Confusion Matrix")
plt.show()
```



[15]
0s

```python
dt_params = {
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

dt_grid = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    dt_params,
    cv=5
)

dt_grid.fit(X_train, y_train)

print("Best Parameters for Decision Tree:", dt_grid.best_params_)

best_dt = dt_grid.best_estimator_
y_pred_dt_tuned = best_dt.predict(X_test)

print("Tuned Decision Tree Accuracy:",
      accuracy_score(y_test, y_pred_dt_tuned))
```

```
Best Parameters for Decision Tree: {'criterion': 'gini', 'max_depth': None, 'min_samples_split': 2}
Tuned Decision Tree Accuracy: 0.9853658536585366
```

[16]
25s

```python
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5]
}

rf_grid = GridSearchCV(
    RandomForestClassifier(random_state=42),
    rf_params,
    cv=5
)

rf_grid.fit(X_train, y_train)

print("Best Parameters for Random Forest:", rf_grid.best_params_)

best_rf = rf_grid.best_estimator_
y_pred_rf_tuned = best_rf.predict(X_test)

print("Tuned Random Forest Accuracy:",
      accuracy_score(y_test, y_pred_rf_tuned))
```

```
Best Parameters for Random Forest: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Tuned Random Forest Accuracy: 0.9853658536585366
```

```
[17]
accuracies = [
    accuracy_score(y_test, y_pred_dt_tuned),
    accuracy_score(y_test, y_pred_rf_tuned)
]

plt.figure(figsize=(6,4))
sns.barplot(x=models, y=accuracies)
plt.title("Model Comparison (Accuracy)")
plt.ylabel("Accuracy")
plt.show()
```