Experiment Number : 4

Problem Statement: **Write a python code to implement Decision tree and random forest classifier for a suitable use case.**

NAME: Harshwardhan PatilROLLNO: 50

CLASS: IT-B BATCH: B1

DATE OF PERFORMANCE: 11-02-2025

CODE:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"

data = pd.read\_csv(url)

# Preprocess the dataset

# Select relevant columns for simplicity (including the target column 'Survived')

data = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Survived']]

# Handle missing data

data['Age'] = data['Age'].fillna(data['Age'].median())  # Fill missing Age with median

data.dropna(subset=['Survived', 'Sex'], inplace=True)    # Drop rows where 'Survived' or 'Sex' is missing

# Convert categorical variables to numerical values

data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

# Define features and target variable

X = data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]

y = data['Survived']

# Train a Decision Tree Classifier to calculate Gini and Information Gain

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize and train the Decision Tree model

clf = DecisionTreeClassifier(criterion='gini', max\_depth=5)

clf.fit(X\_train, y\_train)

# Predict using the trained model

y\_pred = clf.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy of the Decision Tree: {accuracy \* 100:.2f}%')

# Visualize the decision tree

plt.figure(figsize=(12,8))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=['Not Survived', 'Survived'], rounded=True)

plt.title("Decision Tree for Titanic Survival Prediction")

plt.show()

# Calculate Gini Index and Information Gain for a sample split

# Function to calculate Gini index for a dataset

def gini\_index(data):

    total = len(data)

    if total == 0:

        return 0

    # Count the occurrences of each class in the target variable

    class\_counts = data['Survived'].value\_counts()

    gini = 1 - sum((class\_counts / total) \*\* 2)

    return gini

# Function to calculate Information Gain for a feature

def information\_gain(data, feature):

    total\_gini = gini\_index(data)  # Gini index of the dataset before the split

    values = data[feature].unique()

    weighted\_gini = 0

    # For each value of the feature, calculate the weighted Gini index

    for value in values:

        subset = data[data[feature] == value]

        subset\_gini = gini\_index(subset)

        weighted\_gini += (len(subset) / len(data)) \* subset\_gini

    # Information Gain is the reduction in Gini index after the split

    return total\_gini - weighted\_gini

# Example: Calculate Gini index and Information Gain for the feature 'Pclass'

print("\nCalculating Gini index and Information Gain for the feature 'Pclass':")

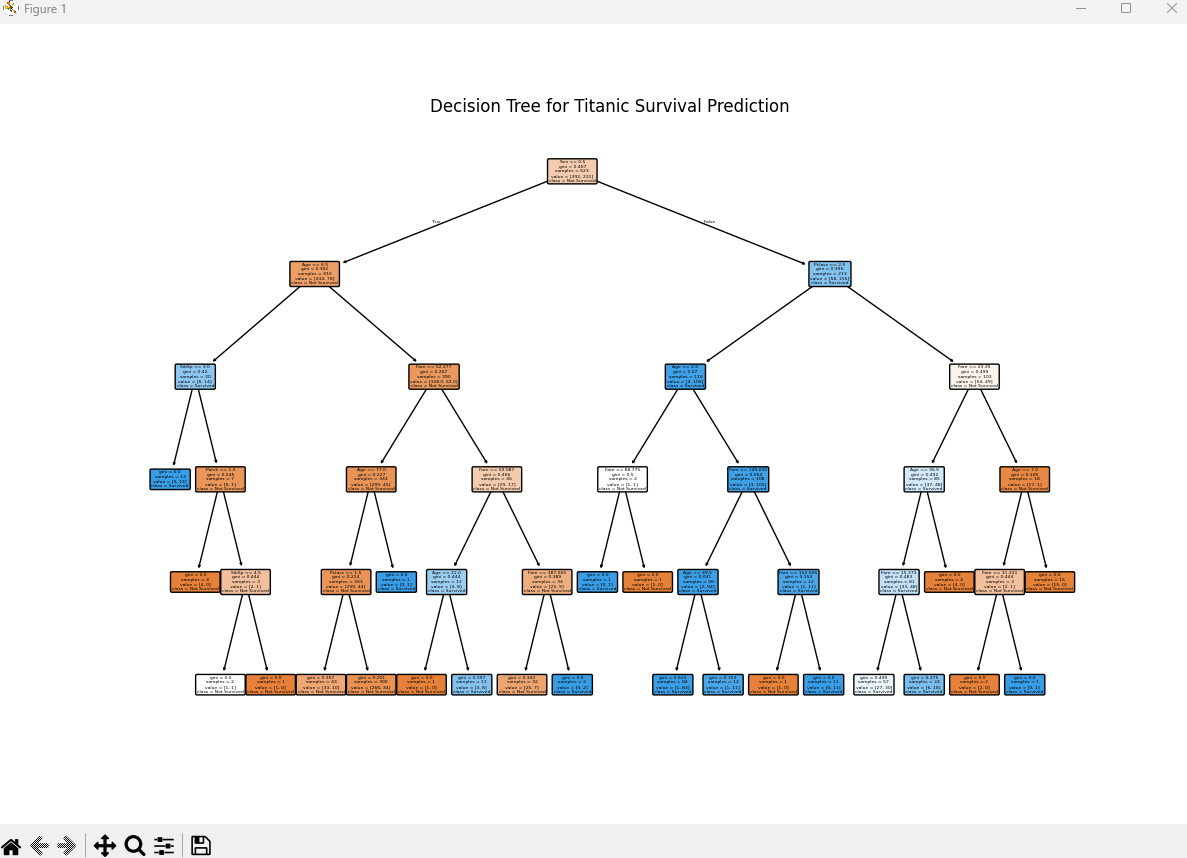
gini\_before\_split = gini\_index(data)

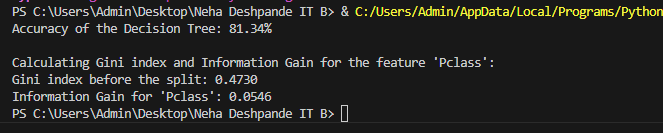
info\_gain\_pclass = information\_gain(data, 'Pclass')

print(f"Gini index before the split: {gini\_before\_split:.4f}")

print(f"Information Gain for 'Pclass': {info\_gain\_pclass:.4f}")

OUTPUT:





Conclusion: **We have wrote a Python Code to design Decision Tree for the given dataset. Calculate Gini index and Information Gain.  
  
  
Random Forest:**  
PROGARM:

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 LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.tree import plot\_tree

# Load dataset from CSV

column\_names = ["buying", "maint", "doors", "persons", "lug\_boot", "safety", "unacc"]

df = pd.read\_csv("car\_evaluation.csv", names=column\_names, header=None)

# Strip spaces from column names to avoid errors

df.columns = df.columns.str.strip()

# Verify available columns

print("Available columns:", df.columns)

# Check if 'class' column exists

target\_col = "unacc"

if target\_col not in df.columns:

    raise KeyError(f"Error: Column '{target\_col}' not found in dataset. Available columns: {df.columns}")

# Encode categorical data

label\_encoders = {}

for col in df.columns:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

# Split data into features and target

X = df.drop(columns=[target\_col])

y = df[target\_col]

# Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = clf.predict(X\_test)

# Evaluate model

accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.4f}")

print("Classification Report:\n", report)

print("Confusion Matrix:\n", conf\_matrix)

# Visualize one of the trees

plt.figure(figsize=(20,10))

plot\_tree(clf.estimators\_[0], feature\_names=X.columns, class\_names=label\_encoders[target\_col].classes\_, filled=True, rounded=True)

plt.show()

# Plot confusion matrix

plt.figure(figsize=(8,6))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=label\_encoders[target\_col].classes\_, yticklabels=label\_encoders[target\_col].classes\_)

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

Output :

A screenshot of a computer

AI-generated content may be incorrect.

