Assignment 4

Aim: Implementation of Decision Tree Classifier on Gender Classification Dataset
Objective: To implement and evaluate a Decision Tree Classifier using Python to predict gender based on physical and behavioral features in the gender classification dataset.

Introduction:

Importance of Decision Tree:
Decision Trees are a powerful supervised learning algorithm for classification and regression tasks. They split the dataset into branches based on feature values to make predictions. In this assignment, we preprocess the **Gender classification dataset**, train a **Decision Tree Classifier**, analyze its performance, and visualize the results.

Advantages of Decision Trees:

- **Easy to Interpret and Understand:** The tree structure is intuitive.
- **Handles Both Categorical and Numerical Data:** Unlike many other algorithms, Decision Trees can process mixed data types.
- **Minimal Data Preprocessing Required:** No need for extensive feature scaling or transformation.
- **Captures Non-Linear Relationships:** Can efficiently model complex decision boundaries.
- **Useful for Feature Selection:** Identifies important variables based on how frequently they are used to split nodes.

Dataset:

The dataset used in this assignment is the **Gender Classification Dataset**, containing various features extracted from physical traits and characteristics. It includes both numerical and categorical attributes. The key features include:

- long_hair: Presence of long hair (0/1)
- forehead_width_cm: Width of forehead in centimeters
- forehead_height_cm: Height of forehead in centimeters
- nose_wide: Wide nose or not
- nose_long: Long nose or not
- lips_thin: Thin lips or not
- distance_nose_to_lip_long: Long distance from nose to lip
- gender: Target variable representing the gender (Male/Female)

Steps of Implementation:

1. Importing Libraries:

- Import the required Python libraries:
  - pandas, numpy: Data loading and manipulation
  - matplotlib.pyplot, seaborn: For visualization
  - sklearn.tree.DecisionTreeClassifier: For model training
  - sklearn.model_selection.train_test_split: For data splitting

- sklearn.metrics: For model evaluation
- sklearn.preprocessing: For encoding categorical variables

2. Loading the Dataset:

- Load the dataset using pd.read_csv()
- Perform initial data exploration using:
    - .shape() to check dataset size
    - .head() to preview the first few rows
    - .info() to examine datatypes and missing values

3. Data Preprocessing:

- Encoding Categorical Variables:
    - If any input features (like height, weight, foot_size) are categorical, encode them.
    - Encode the target variable Gender using label encoding (Male → 1, Female → 0).
- Handling Missing Values:
    - Fill categorical columns with mode
    - Fill numerical columns with median if any missing values exist
- Feature Selection:
    - Select relevant features such as height, weight, and foot_size as input variables (X)
    - Target variable: Gender
- Train-Test Split:
    - Split the dataset into 67% training and 33% testing using train_test_split()

4. Training the Decision Tree Model:

- Initialize a DecisionTreeClassifier with:
    - criterion="gini"
    - max_depth=3
- Train the model using the training data (X_train, y_train)

5. Making Predictions:

- Use the trained model to predict Gender on the test dataset (X_test)
- Store predictions in a separate variable

6. Model Evaluation:

- Use metrics from sklearn.metrics:
    - Accuracy Score: Measure of correct predictions
    - Confusion Matrix: To see true positives, false positives, etc.

- Classification Report: Precision, recall, F1-score per class

7. Visualization of Results:

- Plot the trained Decision Tree using:
  - sklearn.tree.plot_tree() or graphviz (for better visuals)
- Understand how the tree splits based on height, weight, or foot_size to classify gender

Conclusion:

- The Decision Tree Classifier was successfully implemented to classify gender based on facial features and traits.
- **Accuracy Score** indicates how well the model performs on unseen data.
- **Confusion Matrix** helps identify classification errors.
- **Classification Report** offers insights into precision, recall, and F1-score.
- **Tree Visualization** provides an intuitive understanding of the model's logic and key features used for decision-making.