1. What is a Decision Tree, and how does it work in the context of

classification?

A decision tree is a supervised machine learning algorithm that works like a flowchart to classify data into distinct categories by creating a tree-like structure of decisions. It starts with a root node and branches out, with each internal node representing a test on a feature and each leaf node representing the final classification or outcome. In classification, the tree splits data recursively based on feature values to create pure, homogeneous subsets at the leaf nodes, enabling it to predict the class of new data points by traversing down the branches. How a Decision Tree Works in Classification 1. Root Node: The process begins at the root node, which is the initial question or test based on a specific feature of the dataset. 2. Branches: The branches represent the possible outcomes or answers (e.g., "yes" or "no," or a value range) to the test at an internal node. 3. Internal Nodes: Each branch leads to an internal node, which poses another question or performs another test on a different feature. 4. Recursive Splitting: This splitting process continues, recursively dividing the data into smaller and smaller subsets based on the feature tests. The goal is to create subsets where all instances belong to the same class, a state known as a "pure" node. 5. Leaf Nodes: When a split results in a pure subset, it becomes a leaf node, representing a final predicted class label for that data. 6. Prediction: To classify a new data instance, you start at the root node and follow the branches based on the instance's feature values until you reach a leaf node, which provides the predicted class.

2. Explain the concepts of Gini Impurity and Entropy as impurity measures.

How do they impact the splits in a Decision Tree?

Gini Impurity and Entropy are both metrics for measuring the "impurity" or disorder of data within a node in a decision tree. Gini Impurity quantifies the probability of an incorrect classification if a data point is randomly assigned to a class, while Entropy measures the uncertainty or disorder of the node's data distribution. In decision trees, the algorithm selects splits that minimize the impurity (maximize purity) of the child nodes, aiming to create more homogeneous, pure leaf nodes with the lowest possible Gini or Entropy values. Gini Impurity Explained:- Concept: Gini Impurity measures how often a randomly selected element from a dataset would be incorrectly classified if it were randomly labeled based on the class distribution at that node. Range: It ranges from 0 to 1. Interpretation: 0: Represents a perfectly "pure" node, meaning all data points belong to the same class. 1: Represents a maximum impurity, where data points are randomly distributed across different classes. Entropy Explained:- Concept: Entropy measures the amount of uncertainty or disorder in a set of data. It quantifies the expected "information," "surprise," or "uncertainty" associated with the potential outcomes of a random variable. Range: It also ranges from 0 to 1. Interpretation: 0: Denotes a pure node with no uncertainty or disorder. 1: Denotes the most impure node, such as a 50-50 split between two classes, indicating maximum uncertainty. Impact on Splits in a Decision Tree:- Both Gini Impurity and Entropy serve as splitting criteria in decision trees to determine the best feature and split point. 1. Measuring Node Purity: At each node, the algorithm calculates the impurity (either Gini or Entropy) for the current node. 2. Evaluating Splits: It then evaluates all possible splits for each feature and their corresponding thresholds. 3. Calculating Child Impurity: For each potential split, the impurity is calculated for the resulting child nodes. 4. Weighted Average: A weighted average impurity for the child nodes is computed, based on the number of instances in each child node. 5. Selecting the Best Split: The split that results in the lowest weighted average impurity is selected as the best split for that node. 6. Goal: The ultimate goal is to reduce impurity from the root node down to the leaf nodes, creating a tree that effectively separates data into pure, distinct categories.

3. What is the difference between Pre-Pruning and Post-Pruning in Decision

Trees? Give one practical advantage of using each.

Pre-pruning stops a decision tree's growth during training by setting conditions to prevent it from becoming too complex, while post-pruning builds a full tree and then removes unnecessary branches to simplify it and improve generalization. A practical advantage of pre-pruning is its computational efficiency because it avoids building unnecessary parts of the tree. A practical advantage of post-pruning is its robustness, as it often produces a more accurate model than pre-pruning because it doesn't suffer from a greedy approach that might prematurely stop growth. Pre-Pruning (Early Stopping):- How it works: During the construction of the decision tree, growth is stopped based on specific criteria, such as a maximum depth or a minimum number of samples in a node. Practical Advantage: Computational efficiency and Speed. By not growing the tree to its full potential, it saves computation time and resources, making it more efficient for large datasets. Post-Pruning (Backward Pruning):- How it works: A complete decision tree is first generated, and then branches that do not significantly contribute to the model's performance are removed or simplified. Practical Advantage: Improved Accuracy and Robustness. By pruning after the tree is built, it can be more effective at identifying and removing irrelevant branches, often leading to a more accurate and robust final model than pre-pruning, which can be too aggressive.

4.What is Information Gain in Decision Trees, and why is it important for

choosing the best split?

Information Gain measures the reduction in entropy (data impurity or randomness) after a dataset is split using a particular feature, indicating how much that feature helps to classify the data. It's important for choosing the best split because decision tree algorithms select the feature that yields the highest Information Gain, as this feature best separates the data into more homogeneous (purer) subsets, leading to a more effective and accurate classification tree. What is Information Gain? Measures Reduction in Uncertainty: Information Gain quantifies how much the uncertainty or "impurity" in a dataset decreases after it's split by a specific attribute. Based on Entropy: It is calculated by subtracting the weighted average entropy of the child nodes from the entropy of the parent node. A Proxy for Purity: A higher Information Gain indicates a more effective split, resulting in subsets that are more "pure" (contain instances of a single class). Why is it Important for Choosing the Best Split? Identifies the Most Informative Feature: At each internal node of a decision tree, the algorithm evaluates the Information Gain for every available feature. Maximizes Purity: The feature with the maximum Information Gain is chosen for the split because it creates subsets that are the most distinct and well-separated. Builds a More Effective Tree: By maximizing Information Gain, the decision tree construction process ensures that the most useful features are prioritized, leading to a more homogeneous and pure classification at the leaf nodes. Greedy Approach: The algorithm is "greedy," meaning it selects the best split at each step without looking ahead or backtracking, making Information Gain the critical metric for making these local decisions.

5. What are some common real-world applications of Decision Trees, and

what are their main advantages and limitations?

Decision trees are used in healthcare for diagnoses, in finance for credit risk assessment, and in marketing for customer segmentation. Their main advantages include interpretability, ease of use, and the ability to handle both numerical and categorical data, while limitations are overfitting, instability with data changes, a bias towards features with many levels, and potential for poor generalization on new data. Common Real-World Applications:- Healthcare: To assist in diagnosing diseases by analyzing patient symptoms and medical history. Finance: For credit scoring, risk assessment, and fraud detection in financial transactions. Marketing: To understand customer behavior, predict customer churn, and for targeted marketing strategies. Retail: In inventory management by predicting sales trends and optimizing stock levels. Manufacturing:

For process optimization by analyzing factors that impact operational efficiency. Telecommunications: To predict and identify customer churn based on usage patterns. Advantages:- Understandable & Interpretable: Decision trees are easy to understand, with logical choices and tangible results, making complex decisions clear. Handles Diverse Data: They can process both numerical and categorical data, making them versatile for various datasets. Simple Data Preparation: Requires relatively little data preparation compared to other complex machine learning algorithms. Handles Non-linear Relationships: Decision trees can capture and model complex, non-linear relationships within the data. Limitations:- Overfitting: Trees can become too complex and deep, leading to poor performance on new, unseen data because they perfectly fit the training data. Instability: Small variations or noise in the data can result in completely different tree structures, making them sensitive to data changes. Bias Towards Dominant Classes: In datasets with imbalanced class distributions, the decision tree may favor the dominant class, leading to biased predictions. Limited Precision for Complex Problems: The tree structure can struggle to capture certain complex relationships effectively, potentially limiting precision for highly intricate problems.

6. Write a Python program to:

● Load the Iris Dataset ● Train a Decision Tree Classifier using the Gini criterion ● Print the model's accuracy and feature importances

In [1]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Decision Tree using Gini criterion
clf = DecisionTreeClassifier(criterion='gini', random_state=42)
clf.fit(X_train, y_train)

# Predict and calculate accuracy
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")

# Display feature importances
importances = clf.feature_importances_
print("Feature Importances:")
for name, importance in zip(feature_names, importances):
    print(f"{name}: {importance:.4f}")
```

```
Model Accuracy: 1.00
Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

7. Write a Python program to:

● Load the Iris Dataset ● Train a Decision Tree Classifier with max_depth=3 and compare its accuracy to a fully-grown tree.

In [2]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model 1: Decision Tree with max_depth=3
tree_depth3 = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
tree_depth3.fit(X_train, y_train)
y_pred_depth3 = tree_depth3.predict(X_test)
accuracy_depth3 = accuracy_score(y_test, y_pred_depth3)

# Model 2: Fully-grown Decision Tree (no max_depth)
tree_full = DecisionTreeClassifier(criterion='gini', random_state=42)
tree_full.fit(X_train, y_train)
y_pred_full = tree_full.predict(X_test)
accuracy_full = accuracy_score(y_test, y_pred_full)

# Print results
```

```
print(f"Accuracy with max_depth=3: {accuracy_depth3:.2f}")
print(f"Accuracy with fully-grown tree: {accuracy_full:.2f}")
```

```
Accuracy with max_depth=3: 1.00
Accuracy with fully-grown tree: 1.00
```

8. Write a Python program to:

● Load the California Housing dataset from sklearn ● Train a Decision Tree Regressor ● Print the Mean Squared Error (MSE) and feature importances

In [3]:
```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load California Housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target
feature_names = housing.feature_names

# Split 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 Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Predict and calculate MSE
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

# Display feature importances
importances = regressor.feature_importances_
print("\nFeature Importances:")
for name, importance in zip(feature_names, importances):
    print(f"{name}: {importance:.4f}")
```

```
Mean Squared Error: 0.4952

Feature Importances:
MedInc: 0.5285
HouseAge: 0.0519
AveRooms: 0.0530
AveBedrms: 0.0287
Population: 0.0305
AveOccup: 0.1308
Latitude: 0.0937
Longitude: 0.0829
```

9. Write a Python program to:

● Load the Iris Dataset ● Tune the Decision Tree's max_depth and min_samples_split using GridSearchCV ● Print the best parameters and the resulting model accuracy

In [4]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Define parameter grid for tuning
param_grid = {
    'max_depth': [2, 3, 4, 5, None],
    'min_samples_split': [2, 4, 6, 8, 10]
}

# Create Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=42)
```

```
# Use GridSearchCV to search for best parameters
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best model and its parameters
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")

# Evaluate on test set
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'max_depth': 4, 'min_samples_split': 2}
Model Accuracy: 1.00
```

10. Imagine you're working as a data scientist for a healthcare company that

wants to predict whether a patient has a certain disease. You have a large dataset with mixed data types and some missing values. Explain the step-by-step process you would follow to: ● Handle the missing values ● Encode the categorical features ● Train a Decision Tree model ● Tune its hyperparameters ● Evaluate its performance And describe what business value this model could provide in the real-world setting.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# ---------------------
# 1. Load your dataset
# ---------------------
# Replace this with your actual data loading logic
df = pd.read_csv("your_dataset.csv")  # placeholder
target_col = 'disease_present'        # binary target column (0 or 1)

# Separate features and target
X = df.drop(columns=target_col)
y = df[target_col]

# Identify column types
categorical_cols = X.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# ------------------------------
# 2. Define preprocessing steps
# ------------------------------
# Impute numerical and categorical features
num_imputer = SimpleImputer(strategy='median')
cat_imputer = SimpleImputer(strategy='most_frequent')

# One-hot encode categorical features
cat_encoder = OneHotEncoder(handle_unknown='ignore')

# Column transformer for preprocessing
preprocessor = ColumnTransformer(transformers=[
    ('num', num_imputer, numerical_cols),
    ('cat', Pipeline(steps=[
        ('imputer', cat_imputer),
        ('encoder', cat_encoder)
    ]), categorical_cols)
])

# ----------------------------
# 3. Build pipeline with model
# ----------------------------
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# ---------------------------------
# 4. Split data and define grid search
# ---------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

```
param_grid = {
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# ----------------------------
# 5. Evaluate best model
# ----------------------------
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print("Best Parameters:", grid_search.best_params_)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

Business Value in a Real-World Setting Implementing this model can bring tangible business and societal value: 1. Early Disease Detection Helps identify at-risk patients sooner, leading to timely intervention. Reduces progression and complications. 2. Improved Clinical Decisions Acts as a decision-support tool for doctors, highlighting high-risk patients. Enables personalized treatment plans. 3. Cost Reduction Preventive care is often cheaper than treating late-stage disease. Reduces unnecessary tests by targeting the right patients. 4. Resource Optimization Helps hospitals allocate resources (staff, diagnostics) more efficiently. 5. Compliance and Risk Management Aids in meeting quality benchmarks (e.g., avoid missed diagnoses). Strengthens reputation with regulators and insurers.

In [ ]: