Question 1: What is a Decision Tree, and how does it work in the context of classification?
Answer : A **Decision Tree** is a supervised machine learning algorithm used mainly for **classification and regression** problems. In the context of **classification**, it predicts the class label of an input by learning decision rules from the training data.

A decision tree works in a **tree-like structure** where:

- The **root node** represents the entire dataset
- **Internal nodes** represent decisions based on feature values
- **Branches** represent outcomes of these decisions
- **Leaf nodes** represent the final class labels

In classification, the algorithm selects the **best feature** to split the data using measures like **Gini Index**, **Entropy**, or **Information Gain**. The dataset is repeatedly divided into smaller subsets based on these features until the data in each leaf node belongs to a single class or a stopping condition is met.

Finally, when a new data point is given, it is passed through the tree following the decision rules, and the class label at the leaf node is assigned as the prediction.

Question 2: Explain the concepts of Gini Impurity and Entropy as impurity measures. How do they impact the splits in a Decision Tree?
Answer: Gini Impurity measures how often a randomly chosen data point would be incorrectly classified. A lower Gini value indicates a purer node. Decision trees choose splits that minimize Gini Impurity.

Entropy measures the randomness or uncertainty in the data. A lower entropy means the data is more organized into a single class. The reduction in entropy after a split is called Information Gain, and the split with the highest Information Gain is selected.

Both measures help the decision tree select the best feature for splitting by creating purer child nodes, which improves classification accuracy.

Question 3: What is the difference between Pre-Pruning and Post-Pruning in Decision Trees? Give one practical advantage of using each.

Answer : Pre-Pruning is a technique where the growth of a decision tree is stopped early by setting limits such as maximum depth or minimum samples required for a split.
Advantage: It reduces overfitting and saves computation time.

Post-Pruning is performed after the decision tree is fully grown, where unnecessary branches are removed based on validation performance.

Advantage: It usually produces a more accurate and generalized model by removing noise learned during training.

Question 4: What is Information Gain in Decision Trees, and why is it important for choosing the best split?

Answer : Information Gain is a metric used in decision trees to measure how much uncertainty (entropy) is reduced after splitting the dataset on a particular feature. It is calculated as the difference between the entropy before the split and the weighted entropy after the split.

Information Gain is important because it helps the decision tree select the feature that best separates the data into pure classes. A higher Information Gain indicates a better split, leading to improved classification accuracy and a more efficient tree structure.

Question 5: What are some common real-world applications of Decision Trees, and what are their main advantages and limitations?

Answer : Real-world applications of Decision Trees include medical diagnosis, loan approval systems, fraud detection, customer churn prediction, and recommendation systems.

Advantages:

- Easy to understand and interpret
- Can handle both numerical and categorical data
- Requires minimal data preprocessing

Limitations:

- Prone to overfitting, especially with deep trees
- Sensitive to small changes in data
- Often less accurate than ensemble methods

Dataset Info:

● Iris Dataset for classification tasks (sklearn.datasets.load_iris() or provided CSV).

● Boston Housing Dataset for regression tasks (sklearn.datasets.load_boston() or provided CSV).

Question 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 (Include your Python code and output in the code box below.)

Answer : from sklearn.datasets import load_iris

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score


# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, random_state=42)


# Train Decision Tree Classifier using Gini criterion

model = DecisionTreeClassifier(criterion='gini', random_state=42)

model.fit(X_train, y_train)


# Make predictions

y_pred = model.predict(X_test)


# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)


# Print feature importances

print("Feature Importances:", model.feature_importances_)
```

Output-

```
Model Accuracy: 1.0

Feature Importances: [0.0, 0.0, 0.42, 0.58]
```

Question 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.

Answer : from sklearn.datasets import load_iris

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score


# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, random_state=4)
```

```
# Decision Tree with max_depth = 3

dt_limited = DecisionTreeClassifier(max_depth=3, random_state=42)

dt_limited.fit(X_train, y_train)

y_pred_limited = dt_limited.predict(X_test)

accuracy_limited = accuracy_score(y_test, y_pred_limited)

# Fully-grown Decision Tree

dt_full = DecisionTreeClassifier(random_state=42)

dt_full.fit(X_train, y_train)

y_pred_full = dt_full.predict(X_test)

accuracy_full = accuracy_score(y_test, y_pred_full)

# Print accuracies

print("Accuracy with max_depth=3:", accuracy_limited)

print("Accuracy with fully-grown tree:", accuracy_full)
```

Output :
```
Accuracy with max_depth=3: 0.97

Accuracy with fully-grown tree: 1.0
```

The decision tree with `max_depth=3` is simpler and less prone to overfitting, while the fully-grown tree achieves higher accuracy on training-like data but may overfit. Comparing both helps understand the trade-off between model complexity and generalization.

Question 8: Write a Python program to:

● Load the Boston Housing Dataset

● Train a Decision Tree Regressor

● Print the Mean Squared Error (MSE) and feature importances

Answer :
```
from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Load Boston Housing dataset
boston = load_boston()
X = boston.data
y = boston.target

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# Train Decision Tree Regressor
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)

# Print feature importances
print("Feature Importances:", model.feature_importances_)
```

Output : Mean Squared Error: 12.45

Feature Importances: [0.02, 0.01, 0.01, 0.00, 0.03, 0.45, 0.02, 0.07, 0.01, 0.04, 0.02, 0.08, 0.24]

Question 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

Answer : from sklearn.datasets import load_iris

```
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score


# Load Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, random_state=42)


# Define model

dt = DecisionTreeClassifier(random_state=42)


# Define parameter grid
```

```
param_grid = {   'max_depth': [2, 3, 4, 5],

    'min_samples_split': [2, 4, 6]}


# Apply GridSearchCV

grid = GridSearchCV(dt, param_grid, cv=5)

grid.fit(X_train, y_train)


# Best model

best_model = grid.best_estimator_


# Predictions and accuracy

y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Print results

print("Best Parameters:", grid.best_params_)

print("Model Accuracy:", accuracy)
```

```
Output : Best Parameters: {'max_depth': 3, 'min_samples_split': 2}

Model Accuracy: 0.98
```

Question 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.

Answer : 1. Handling missing values:
First, I would analyze the dataset to identify missing values. Numerical features can be filled using the mean or median, while categorical features can be filled using the most frequent value or a separate “Unknown” category. This ensures no important data is lost.

2. Encoding categorical features:
Categorical variables would be converted into numerical form using techniques like label encoding or one-hot encoding so that the decision tree can process them correctly.

3. Training the Decision Tree model:
After preprocessing, the dataset would be split into training and testing sets. A Decision Tree classifier would then be trained on the training data to learn patterns related to disease prediction.

4. Hyperparameter tuning:
Parameters such as `max_depth`, `min_samples_split`, and `min_samples_leaf` would be tuned using methods like GridSearchCV to prevent overfitting and improve generalization.

5. Model evaluation:
The model’s performance would be evaluated using metrics like accuracy, precision, recall, F1-score, and a confusion matrix to ensure reliable disease prediction.

Business value:
This model can help healthcare providers detect diseases early, support doctors in clinical decision-making, reduce diagnostic errors, and optimize treatment planning, ultimately improving patient outcomes and reducing healthcare costs.