## Q1

The estimated depth of a Decision Tree trained on a one million instance training set without any restrictions (unrestricted) can be quite deep. In fact, it could potentially reach a depth close to the number of instances (1 million) if the tree keeps splitting until each leaf node contains only one instance. However, in practice, tree depth is controlled by various hyperparameters like max\_depth, min\_samples\_split, and min\_samples\_leaf. The depth of the tree will depend on the data and these hyperparameters.

## Q2

The Gini impurity of a node is usually lower than that of its parent when splitting a Decision Tree. The splitting criterion in Decision Trees aims to decrease impurity, so nodes after the split are expected to be "purer" (lower Gini impurity) than their parent node. However, it's important to note that while it's a common expectation, it's not an absolute rule. In some cases, due to random chance or specific data distributions, the impurity of a child node may be slightly higher than that of its parent, but the overall trend is a reduction in impurity.

## Q3

Yes, it's generally a good idea to reduce the max depth of a Decision Tree if it's overfitting the training set. Overfitting occurs when the tree becomes too deep and fits the training data's noise, making it perform poorly on unseen data. By reducing the max depth, you limit the tree's complexity and encourage it to capture more general patterns in the data, which can improve its performance on new data.

## Q4

No, scaling the input features is not typically necessary for Decision Trees. Decision Trees are not sensitive to the scale of input features because they make binary decisions based on feature values and thresholds. They split nodes based on comparisons like "Is feature A greater than 5?" The scale of feature A doesn't impact this decision as long as the threshold (5 in this case) is adjusted accordingly.

## Q5

Training time for Decision Trees generally scales with the number of instances and the complexity of the tree. If it takes an hour to train a Decision Tree on a training set with 1 million instances, training on a set with 10 million instances could take significantly longer, possibly around 10 hours or more, assuming the same hardware and conditions.

## Q6

Setting presort=True in scikit-learn's Decision Tree classifier can speed up training for small datasets but is unlikely to provide a significant speedup for datasets with 100,000 instances. The presorting option is designed to speed up training when the dataset is small and fits comfortably in memory. For larger datasets, the overhead of presorting can outweigh any potential benefits. It's generally not recommended to use presort=True for datasets of this size, and scikit-learn's Decision Trees disable presorting by default when the dataset size exceeds a certain threshold (controlled by the presort hyperparameter).

## Q7

# Import necessary libraries

from sklearn.datasets import make\_moons

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

from sklearn.tree import DecisionTreeClassifier

# Step a: Create the moons dataset

X, y = make\_moons(n\_samples=10000, noise=0.4, random\_state=42)

# Step b: Split the dataset into a training and a test set

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

# Step c: Perform grid search with cross-validation to find optimal hyperparameters

param\_grid = {

'max\_leaf\_nodes': [None, 2, 4, 6, 8, 10]

}

grid\_search = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid, cv=5,

scoring='accuracy', return\_train\_score=True)

grid\_search.fit(X\_train, y\_train)

# Print the best hyperparameters found by grid search

print("Best hyperparameters:", grid\_search.best\_params\_)

# Step d: Train the model with the best hyperparameters on the entire training set

best\_tree = DecisionTreeClassifier(max\_leaf\_nodes=grid\_search.best\_params\_['max\_leaf\_nodes'], random\_state=42)

best\_tree.fit(X\_train, y\_train)

# Assess the model's performance on the test set

test\_accuracy = best\_tree.score(X\_test, y\_test)

print("Test set accuracy:", test\_accuracy)

## Q8

# Import necessary libraries

from sklearn.datasets import make\_moons

from sklearn.model\_selection import ShuffleSplit

from sklearn.base import clone

from scipy.stats import mode

import numpy as np

# Step a: Create the moons dataset

X, y = make\_moons(n\_samples=10000, noise=0.4, random\_state=42)

# Step b: Create 1,000 subsets of the training set

n\_trees = 1000

n\_instances = 100

subsets = []

rs = ShuffleSplit(n\_splits=n\_trees, test\_size=len(X) - n\_instances, random\_state=42)

for train\_index, \_ in rs.split(X):

X\_subset = X[train\_index]

y\_subset = y[train\_index]

subsets.append((X\_subset, y\_subset))

# Initialize a list to store the predictions from each tree

y\_pred = []

# Step b: Train one Decision Tree on each subset and evaluate on the test set

for X\_subset, y\_subset in subsets:

tree = DecisionTreeClassifier(max\_leaf\_nodes=grid\_search.best\_params\_['max\_leaf\_nodes'], random\_state=42)

tree.fit(X\_subset, y\_subset)

y\_pred.append(tree.predict(X\_test))

# Step c: Combine predictions using majority voting

y\_pred\_majority\_votes, \_ = mode(np.array(y\_pred), axis=0)

# Step d: Evaluate the ensemble model's accuracy

ensemble\_accuracy = np.mean(y\_pred\_majority\_votes == y\_test)

print("Random Forest accuracy:", ensemble\_accuracy)