1 : What is Information Gain, and how is it used in Decision Trees?

Information Gain (IG) is a metric used to decide which feature to split on at each node of a tree. It measures the reduction in uncertainty (Entropy) after a dataset is split.

- **The Goal:** To choose the attribute that results in the "purest" child nodes.
- **The Usage:** The algorithm calculates IG for every available feature. The feature with the highest Information Gain is selected as the decision node.

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

2: What is the difference between Gini Impurity and Entropy?

- Both measure the "disorder" of a node, but they use different mathematical approaches:

Feature	Gini Impurity	Entropy
Formula	$1 - \sum (p_i)^2$	$-\sum p_i \log_2(p_i)$
Range	0 to 0.5	0 to 1
Computation	Faster (no logarithmic calculations).	Slower (requires log math).
Sensitivity	Favors larger partitions.	Slightly more sensitive to changes in class probabilities.

3:What is Pre-Pruning in Decision Trees?

Pre-pruning (also called "Early Stopping") is a technique used to prevent **overfitting** by halting the growth of the tree before it becomes too complex.

Common pre-pruning constraints include:

- **Max Depth:** Limiting how deep the tree can go.
- **Min Samples Split:** Requiring a minimum number of samples to allow a split.
- **Min Samples Leaf:** Requiring a minimum number of samples to exist in a leaf node.

4:Write a Python program to train a Decision Tree Classifier using Gini Impurity as the criterion and print the feature importances (practical).

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

```
import pandas as pd

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

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

# Train Classifier with Gini
clf = DecisionTreeClassifier(criterion='gini', random_state=42)
clf.fit(X_train, y_train)

# Print Feature Importances
feature_importances = pd.DataFrame(clf.feature_importances_,
                                   index = iris.feature_names,

columns=['importance']).sort_values('importance', ascending=False)
print("Feature Importances (Gini):")
print(feature_importances)

# Optional: Print accuracy
print(f"\nAccuracy: {clf.score(X_test, y_test):.2f}")
```

Output:

Feature Importances (Gini):

	importance
petal width (cm)	0.575510
petal length (cm)	0.395724
sepal length (cm)	0.028765
sepal width (cm)	0.000000

Accuracy: 1.00

5: What is a Support Vector Machine (SVM)?

- An **SVM** is a powerful classifier that finds the **hyperplane** which maximizes the "margin" (distance) between the closest data points of different classes. These closest points are called **Support Vectors**.

6: What is the Kernel Trick in SVM?

- The **Kernel Trick** allows SVMs to solve non-linear problems by implicitly mapping the input data into a higher-dimensional space where it becomes linearly separable. Instead of performing complex transformations, it uses a "Kernel Function" to compute the dot product in that high-dimensional space directly.

7: Write a Python program to train two SVM classifiers with Linear and RBF kernels on the Wine dataset, then compare their accuracies.

```
from sklearn.datasets import load_wine
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler

# Load dataset
wine = load_wine()
X = wine.data
y = wine.target

# Standardize features (crucial for SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, random_state=42)

# Linear Kernel
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_acc = linear_svm.score(X_test, y_test)

# RBF Kernel
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_acc = rbf_svm.score(X_test, y_test)

print(f"Accuracy (Linear Kernel): {linear_acc:.4f}")
print(f"Accuracy (RBF Kernel): {rbf_acc:.4f}")
```

Output:

Accuracy (Linear Kernel): 0.9815
Accuracy (RBF Kernel): 0.9815

8: What is the Naïve Bayes classifier, and why is it called "Naïve"?

- Naïve Bayes is a classification algorithm based on **Bayes' Theorem**. It is called **"Naïve"** because it makes the strong (and often unrealistic) assumption that all features are **independent** of each other given the class label.

9: Explain the differences between Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes

- • **Gaussian:** Used when features follow a normal (Gaussian) distribution (e.g., height, weight).
- • **Multinomial:** Used for discrete counts (e.g., word counts in text classification).
- • **Bernoulli:** Used for binary/boolean features (e.g., whether a word exists in a document or not).

10: Breast Cancer Dataset
Write a Python program to train a Gaussian Naïve Bayes classifier on the Breast Cancer
dataset and evaluate accuracy.

```
from sklearn.datasets import load_breast_cancer
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

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

# Train Classifier
nb = GaussianNB()
nb.fit(X_train, y_train)

# Evaluate Accuracy
y_pred = nb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy (Gaussian Naive Bayes): {accuracy:.4f}")
```

Output:

Accuracy (Gaussian Naive Bayes): 0.9415