1. What is a Decision Tree? What are the steps involved in constructing a Decision Tree?

A Decision Tree is a supervised learning algorithm used for classification and regression tasks. It models decisions and their possible consequences in a tree-like structure of nodes and branches.

Step 1: Select the Best Attribute
• Choose the feature that best splits the data using a criterion:
• Information Gain (Entropy)
• Gini Index
• Gain Ratio
Step 2: Create a Decision Node
• Based on the best attribute, create a decision node and split the dataset accordingly.
• Each branch represents one of the values or ranges of the selected feature.
Step 3: Recurse on Subsets
• For each subset:
• If all samples belong to the same class → make it a leaf node.
• If not, repeat steps 1 & 2 recursively for each subset.
Step 4: Stopping Criteria
• No further gain from splitting.
• All instances in a node belong to the same class.
• Maximum tree depth reached.
• Minimum samples in nodes reached.

2. Given the following dataset, construct a Decision Tree to classify the data. The Decision Tree should determine whether to play a game or not based on the four features of a given day.

Outlook	Temp.	Humidity	Wind	Decision
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

```
|--- Outlook <= 0.50
|   |--- class: 1
|--- Outlook >  0.50
|   |--- Humidity <= 0.50
|   |   |--- Outlook <= 1.50
|   |   |   |--- Wind <= 0.50
|   |   |   |   |--- class: 0
|   |   |   |--- Wind >  0.50
|   |   |   |   |--- class: 1
|   |   |--- Outlook >  1.50
|   |   |   |--- class: 0
|   |--- Humidity >  0.50
|   |   |--- Wind <= 0.50
|   |   |   |--- Outlook <= 1.50
|   |   |   |   |--- class: 0
|   |   |   |--- Outlook >  1.50
|   |   |   |   |--- class: 1
|   |   |--- Wind >  0.50
|   |   |   |--- class: 1

Decision for new day: No
```

```python
from sklearn.tree import DecisionTreeClassifier, export_text
import pandas as pd
from sklearn.preprocessing import LabelEncoder

data = {
    'Outlook': ['Sunny','Sunny','Overcast','Rain','Rain','Rain','Overcast','Sunny',
                'Sunny','Rain','Sunny','Overcast','Overcast','Rain'],
    'Temp': ['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild',
             'Cool','Mild','Mild','Mild','Hot','Mild'],
    'Humidity': ['High','High','High','High','Normal','Normal','Normal','High',
                 'Normal','Normal','Normal','High','Normal','High'],
    'Wind': ['Weak','Strong','Weak','Weak','Weak','Strong','Strong','Weak',
             'Weak','Weak','Strong','Strong','Weak','Strong'],
    'Decision': ['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
}

df = pd.DataFrame(data)

feature_encoders = {}
for column in ['Outlook','Temp','Humidity','Wind']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    feature_encoders[column] = le  # store encoder for later use

target_encoder = LabelEncoder()
df['Decision'] = target_encoder.fit_transform(df['Decision'])

X = df[['Outlook','Temp','Humidity','Wind']]
y = df['Decision']

clf = DecisionTreeClassifier(criterion='entropy')  # ID3 algorithm
clf.fit(X, y)

tree_rules = export_text(clf, feature_names=list(X.columns))
print(tree_rules)

new_day = pd.DataFrame({
    'Outlook':[feature_encoders['Outlook'].transform(['Sunny'])[0]],
    'Temp':[feature_encoders['Temp'].transform(['Hot'])[0]],
    'Humidity':[feature_encoders['Humidity'].transform(['High'])[0]],
    'Wind':[feature_encoders['Wind'].transform(['Weak'])[0]]
})

prediction = clf.predict(new_day)
prediction_label = target_encoder.inverse_transform(prediction)
print("Decision for new day:", prediction_label[0])
```

3. What is Ensemble Learning? Write the steps involved in constructing a Random Forest.

Ensemble Learning combines predictions from multiple models (called learners) to produce a more accurate, stable, and robust final prediction than any individual model alone.

Steps:

1.Take multiple bootstrap samples from the training data

2.For each sample, build a decision tree

At each split, consider a random subset of features

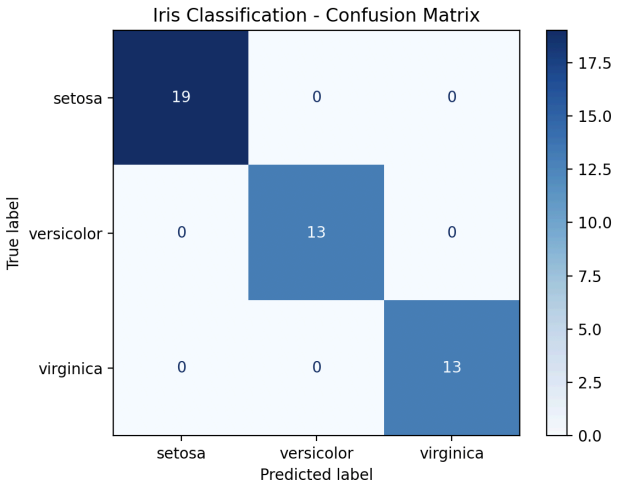
3.Repeat until many trees are built

4.For a new sample:

Get prediction from each tree

Use majority voting to decide the final class

4. Download any publicly available dataset suitable for classification (make sure it is not the same dataset used in class). Using this dataset, build a Random Forest Classifier and evaluate its performance.




```
Accuracy: 1.0

Classification Report:

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Confusion Matrix (raw):

[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

OOB Score: 0.9428571428571428
OOB Error: 0.05714285714285716
```

```python
from sklearn import datasets
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

iris = datasets.load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
data["species"] = iris.target
data["species_name"] = data["species"].apply(lambda x: iris.target_names[x])

X = data[iris.feature_names]
y = data["species"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

clf = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    oob_score=True,
    bootstrap=True
)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("\nAccuracy:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix (raw):\n")
print(cm)

print("\nOOB Score:", clf.oob_score_)
print("OOB Error:", 1 - clf.oob_score_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=iris.target_names)
disp.plot(cmap="Blues")
plt.title("Iris Classification - Confusion Matrix")
plt.show()

example = clf.predict([[5.0, 3.2, 1.5, 0.2]])[0]
print("\nPredicted Species for Example:", iris.target_names[example])
```