**What is a Random Forest?**

Imagine you want to make a decision, but instead of asking just one person, you ask **a group of people** (a "forest") and take a vote. Each person in the group has their own opinion. After hearing from everyone, you choose the **majority opinion**. This is how a **Random Forest** works.

A **Random Forest** is like a team of **many decision trees**, where each tree makes its own decision, and the final result is decided by the most common answer from all the trees.

**Key Things to Understand:**

1. **Decision Trees:** A **Decision Tree** is like asking a series of yes/no questions to split your data. For example, to decide if someone will pass a test, you might first ask, "Did they study?" If yes, they might pass. If no, they might fail. Each **decision tree** makes a prediction based on these kinds of questions.
2. **What Makes Random Forest Different:** Instead of having just **one decision tree**, the **Random Forest** uses **many trees** to make a prediction. Each tree is trained on **different data** and might look at **different features** of the data.

**How Does It Work?**

1. **Random Data Selection (Bootstrapping):** Each tree gets its own **random sample** of data to train on, meaning no two trees see the exact same data.
2. **Random Features:** Each tree also picks a **random set of features** (like test scores, attendance, etc.) to use for making decisions. So even though all the trees are working with the same dataset, each one might focus on different things.
3. **Majority Voting:** After all the trees make their predictions, **Random Forest** chooses the most popular answer from all the trees. If it’s a **classification problem** (like predicting if someone passes or fails), it will choose the class that most trees agree on. If it’s a **regression problem** (like predicting a number), it will average the numbers predicted by all the trees.

**Example with a Simple Scenario:**

Let's say you have a dataset of students' **study hours** and **attendance** to predict whether they **pass or fail** a test.

* **Tree 1** might be trained on a random sample of students and may focus on **attendance** for making decisions.
* **Tree 2** might be trained on a different sample and might focus on **study hours** for making decisions.
* **Tree 3** might mix both attendance and study hours, focusing on both.

In the end, the **Random Forest** will ask **many trees** for their opinions and decide based on the **majority vote**. For example, if 7 trees say "pass" and 3 trees say "fail", the Random Forest will predict **"pass"**.

**Summary:**

* **Decision Tree** is like asking one person to make a decision based on certain conditions.
* **Random Forest** is like asking **many people** (trees) and taking the majority opinion. This makes it more reliable because it avoids the risk of one person (or one tree) making the wrong decision.

So, instead of trusting a single decision tree, **Random Forest** uses many trees to make a more accurate decision.

# Import necessary libraries

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

# Load the Iris dataset

data = load\_iris()

X = data.data

y = data.target

# Split the data into training and testing sets (80% training, 20% testing)

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

# Initialize the classifiers

decision\_tree = DecisionTreeClassifier(random\_state=42)

random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the Decision Tree classifier

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

y\_pred\_dt = decision\_tree.predict(X\_test)

# Train the Random Forest classifier

random\_forest.fit(X\_train, y\_train)

y\_pred\_rf = random\_forest.predict(X\_test)

# Evaluate both models on test data

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

# Perform cross-validation for both models

cv\_scores\_dt = cross\_val\_score(decision\_tree, X, y, cv=5)

cv\_scores\_rf = cross\_val\_score(random\_forest, X, y, cv=5)

# Print the results

print(f"Accuracy of Decision Tree: {accuracy\_dt \* 100:.2f}%")

print(f"Accuracy of Random Forest: {accuracy\_rf \* 100:.2f}%")

# Print cross-validation scores

print("\nCross-validation results:")

print(f"Decision Tree Cross-Validation Scores: {cv\_scores\_dt}")

print(f"Random Forest Cross-Validation Scores: {cv\_scores\_rf}")

# Print average cross-validation scores

print(f"\nAverage Cross-validation score for Decision Tree: {np.mean(cv\_scores\_dt) \* 100:.2f}%")

print(f"Average Cross-validation score for Random Forest: {np.mean(cv\_scores\_rf) \* 100:.2f}%")

# Plotting the comparison of models

models = ['Decision Tree', 'Random Forest']

accuracy = [accuracy\_dt, accuracy\_rf]

cv\_avg = [np.mean(cv\_scores\_dt), np.mean(cv\_scores\_rf)]

# Create a figure for comparison

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Accuracy Bar Plot

sns.barplot(x=models, y=accuracy, ax=ax[0], palette="viridis")

ax[0].set\_title('Accuracy Comparison')

ax[0].set\_ylabel('Accuracy (%)')

ax[0].set\_ylim(0, 1)

# Cross-Validation Average Bar Plot

sns.barplot(x=models, y=cv\_avg, ax=ax[1], palette="viridis")

ax[1].set\_title('Cross-Validation Average Comparison')

ax[1].set\_ylabel('Average CV Score')

ax[1].set\_ylim(0, 1)

plt.tight\_layout()

plt.show()