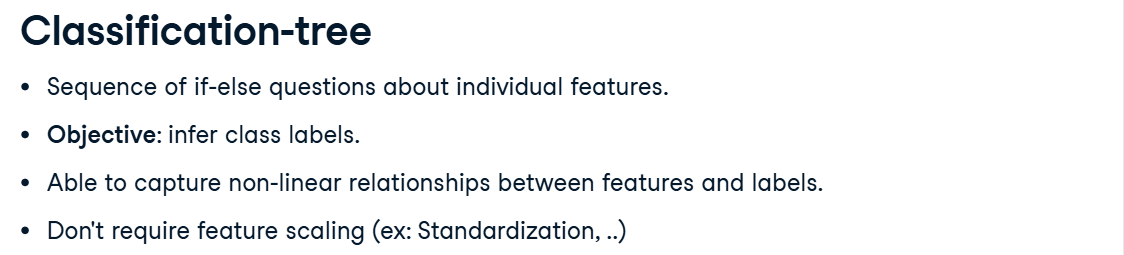
**Machine Learning with Tree-Based Models in Python**



**Classification Tree**

A **classification tree** is a type of **decision tree** used for **classification problems** (where the output/label is categorical, like "yes/no", "spam/not spam", "disease/no disease").

**How it works**

1. You have input features (like age, income, symptoms, etc.).
2. The tree splits the data step by step (like asking questions):
   * *“Is age < 30?”*
   * *“Is income > 50k?”*
   * *“Does patient have fever?”*
3. At each split, it chooses the feature that **best separates the classes** (using measures like **Gini index** or **Entropy/Information Gain**).
4. Eventually, you reach a **leaf node**, which gives the predicted class.

**Example**

Suppose you want to classify if a person will buy a product:

[Age < 30?]

/ \

Yes No

/ \

[Income > 50k?] Buy=No

/ \

Yes No

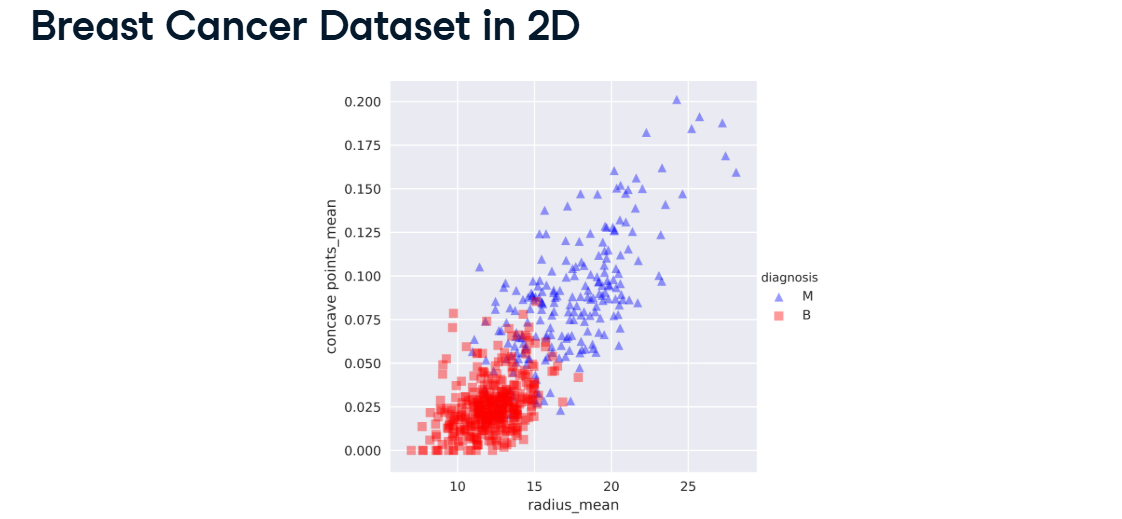
Buy=Yes Buy=No

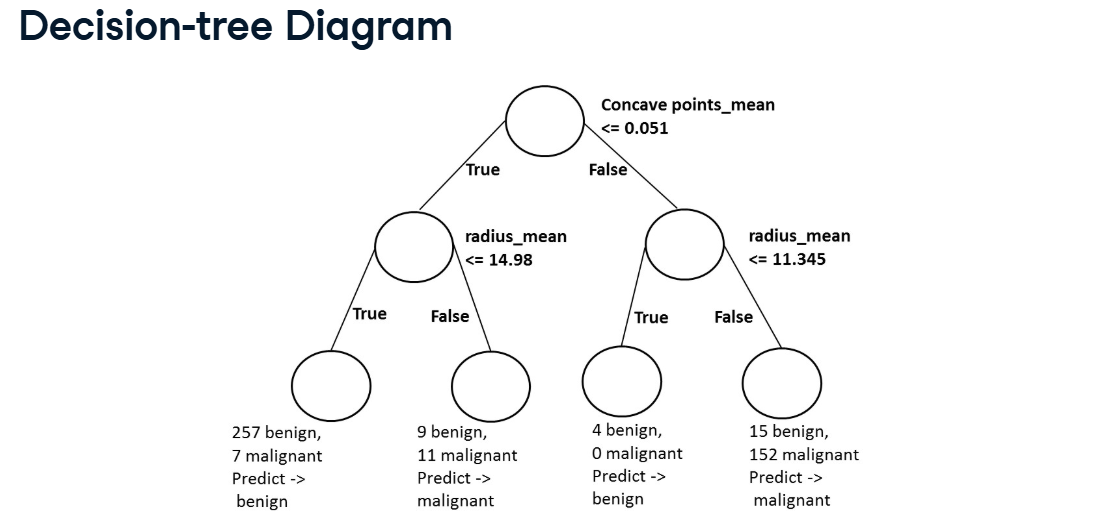
* If age < 30 and income > 50k → **Buy=Yes**
* Otherwise → **Buy=No**

**Key Points**

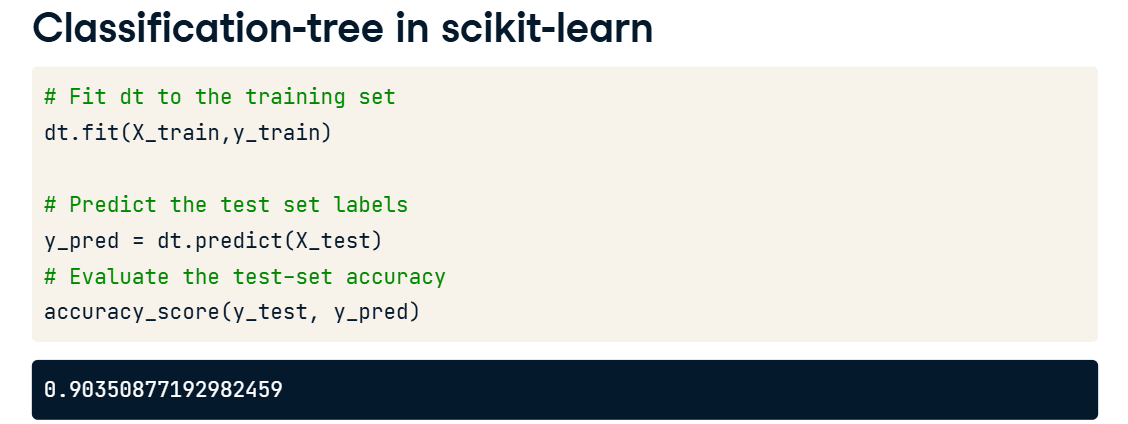
* It’s called a **tree** because it looks like an upside-down tree (root → branches → leaves).
* It’s interpretable (you can **see the decisions**).
* But it can **overfit** if the tree is too deep (so we use pruning or ensemble methods like Random Forest).

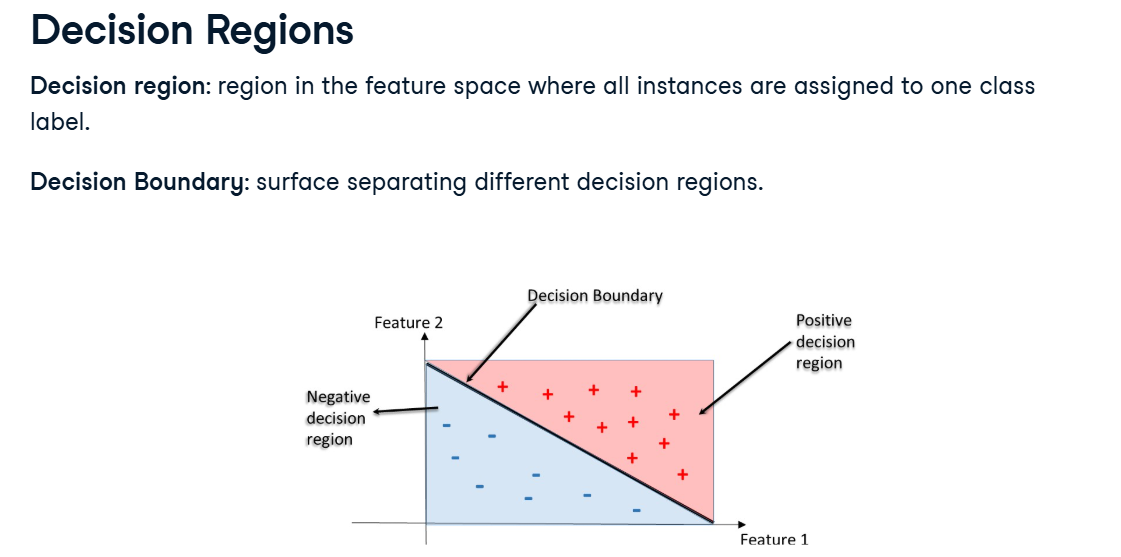
⚡ In short:  
A **classification tree** is a simple yet powerful machine learning model that splits data into branches based on features until it reaches a class decision.

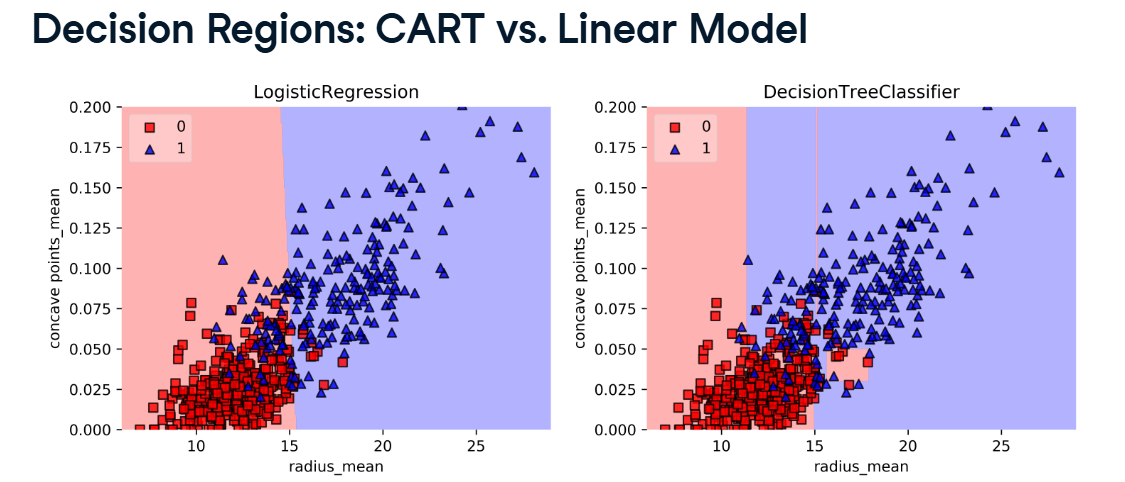








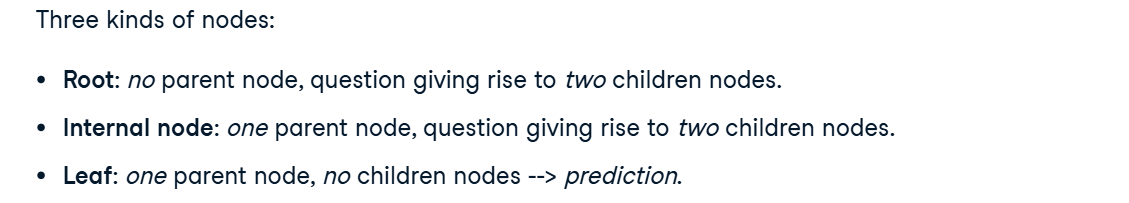


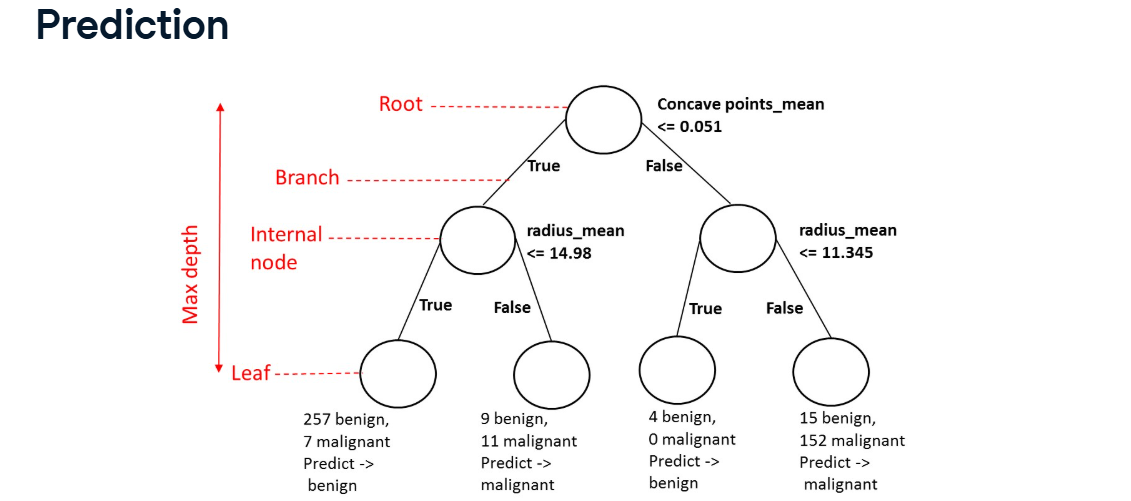


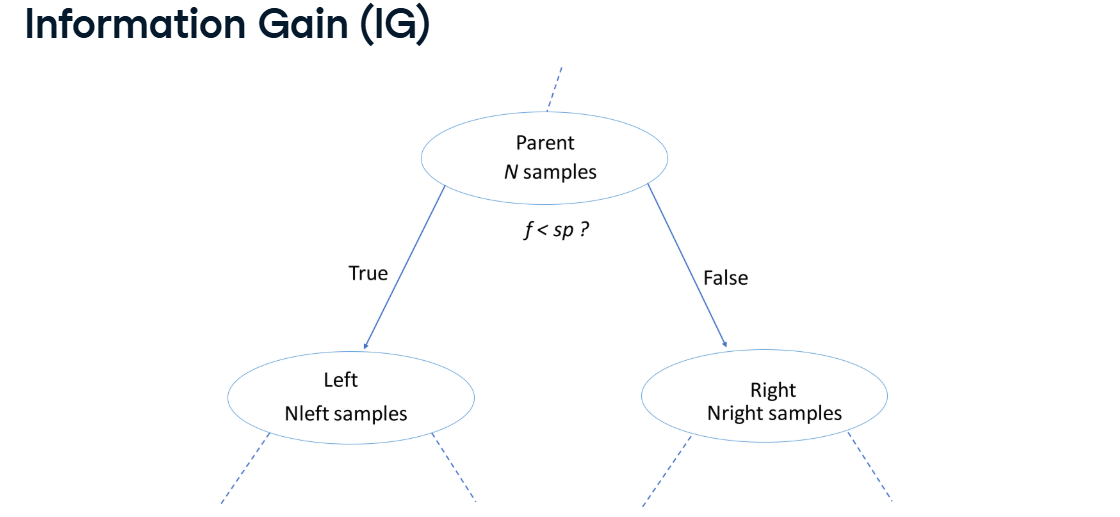
In right the decision region is rectangular because at each split made by the tree only one feature gets involved.

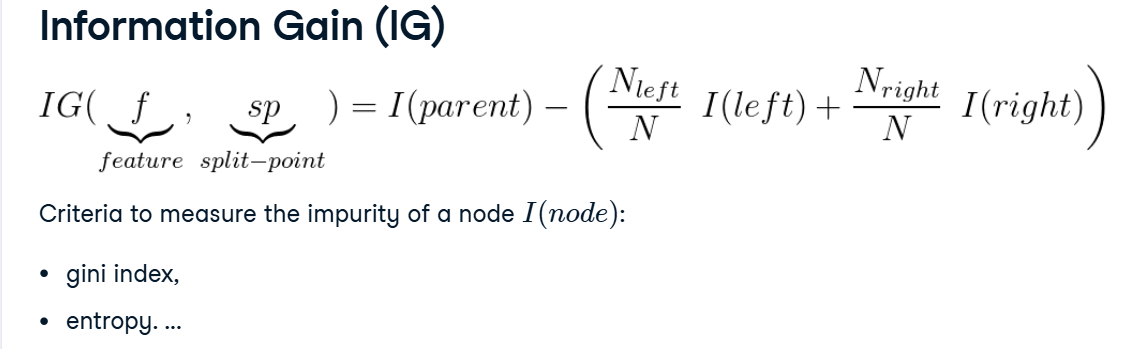
**Decision Tree :** Data Structure consisting of a hierarchy of nodes.

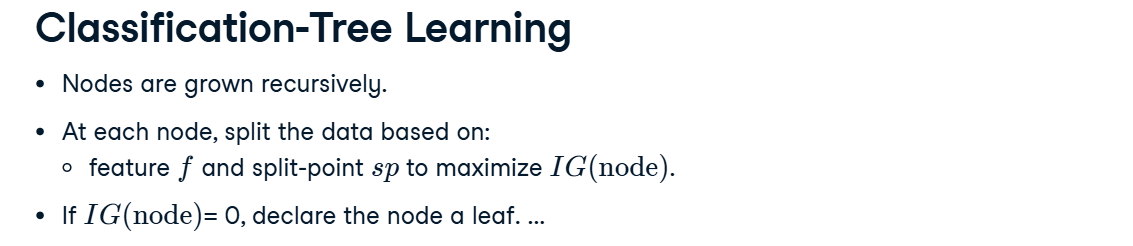
**Node :** Question or Prediction.

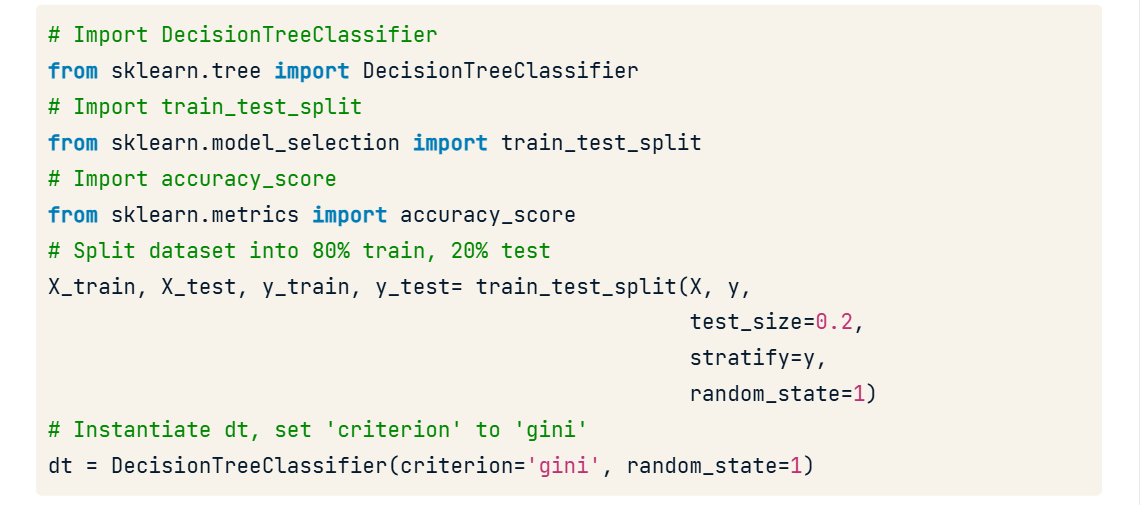


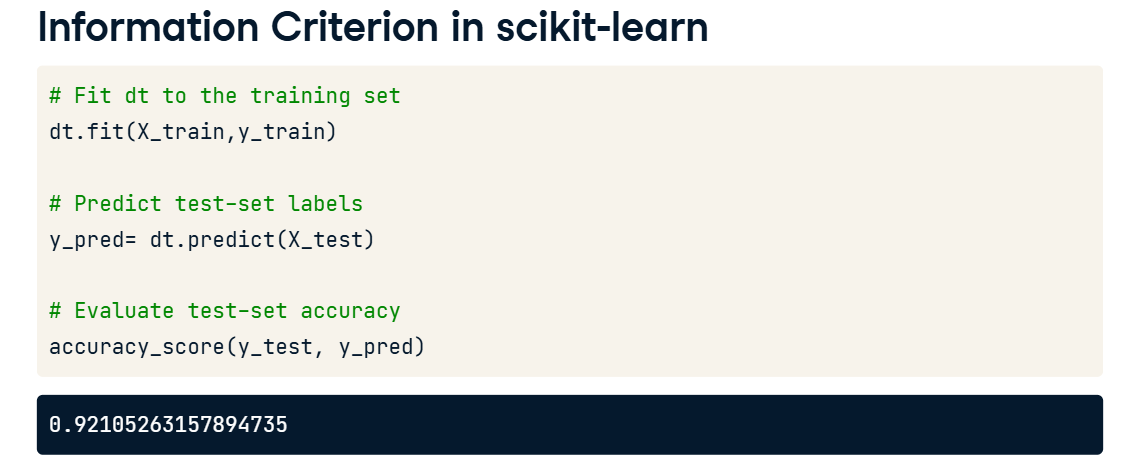




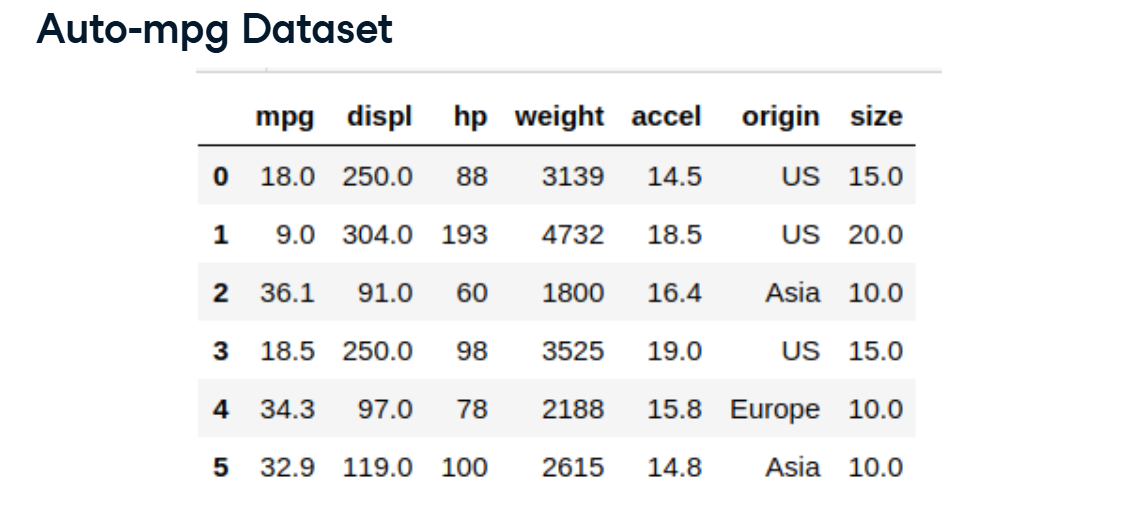






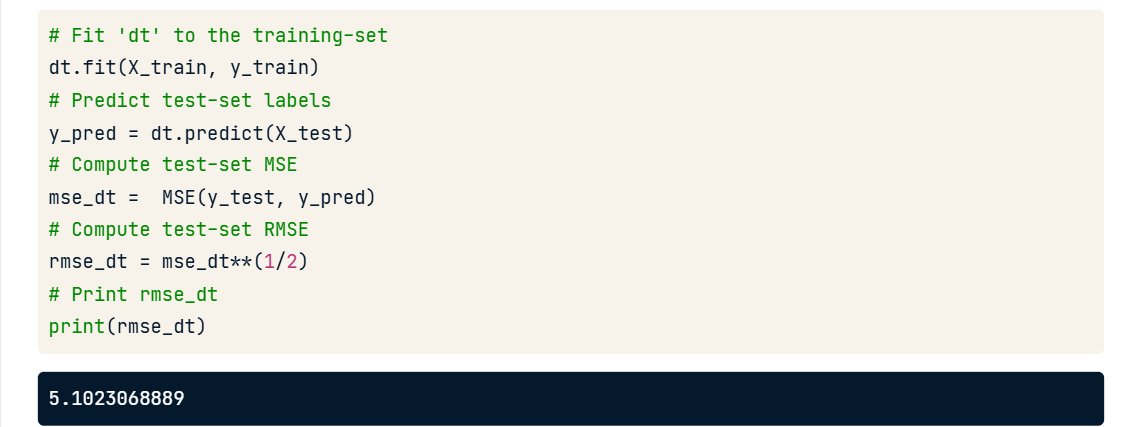


**Decision Tree for Regression**

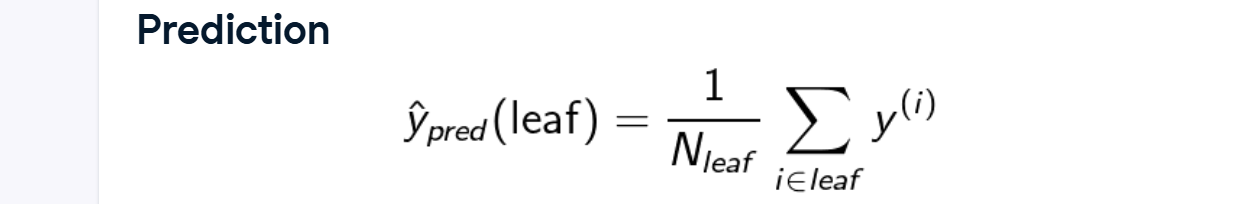


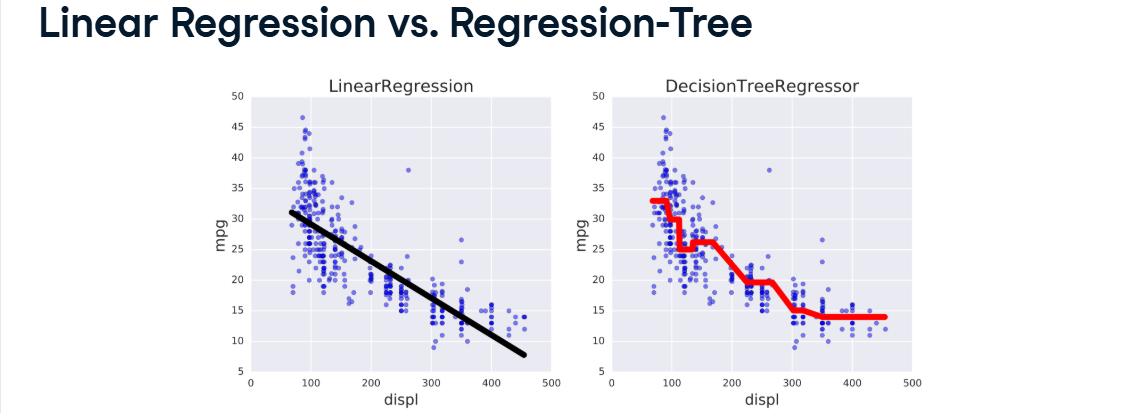


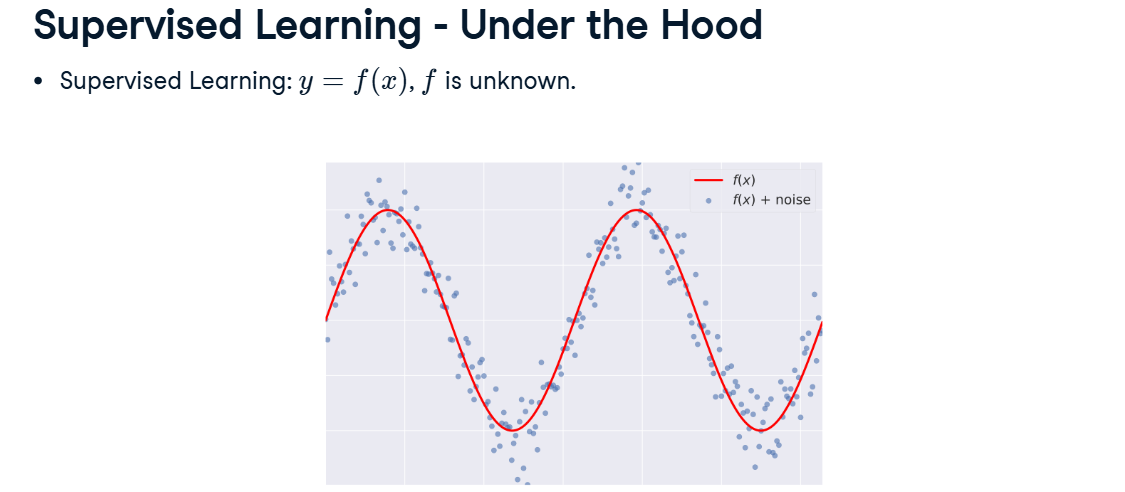


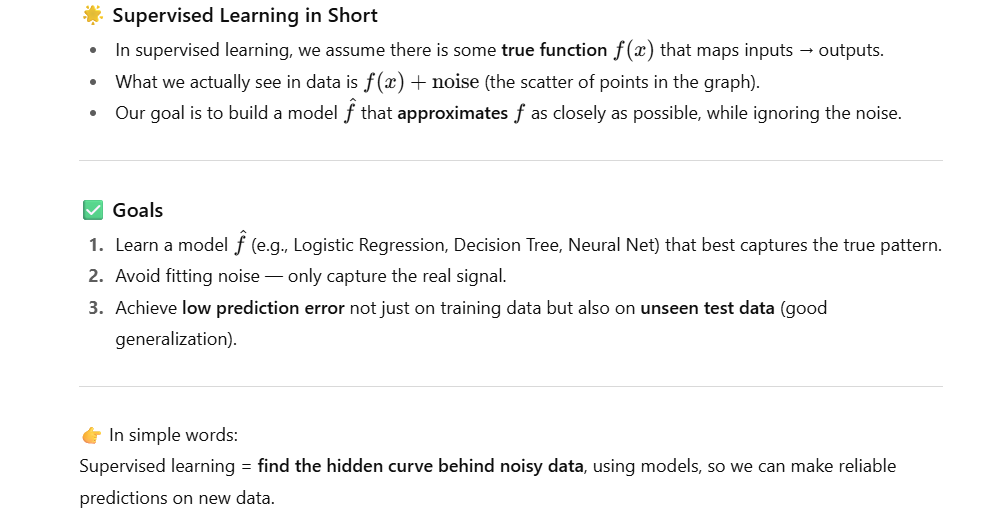


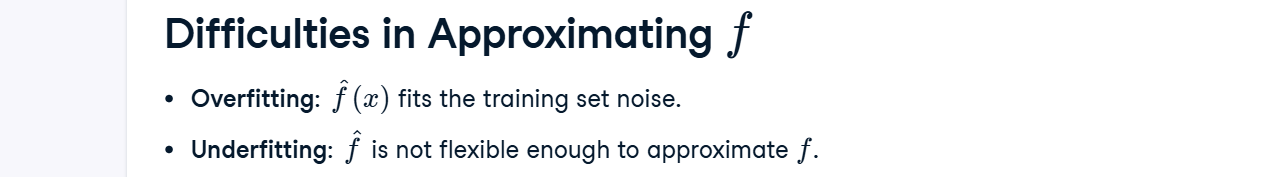


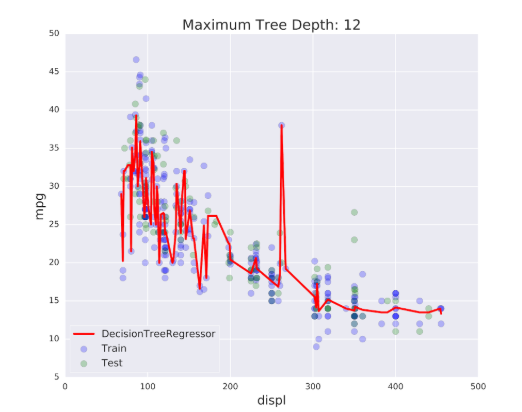
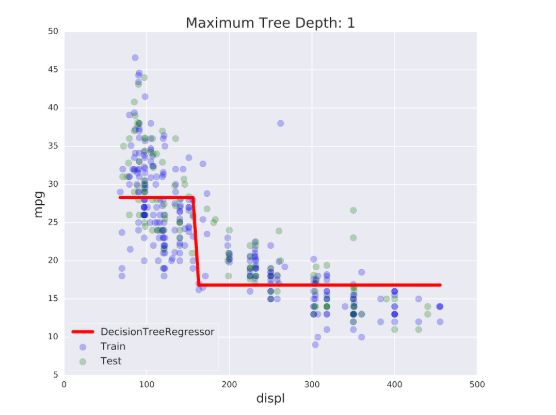


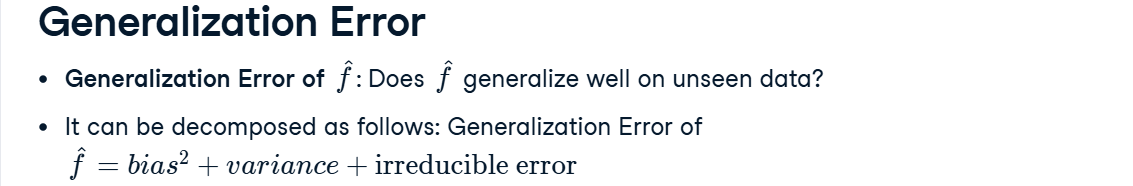


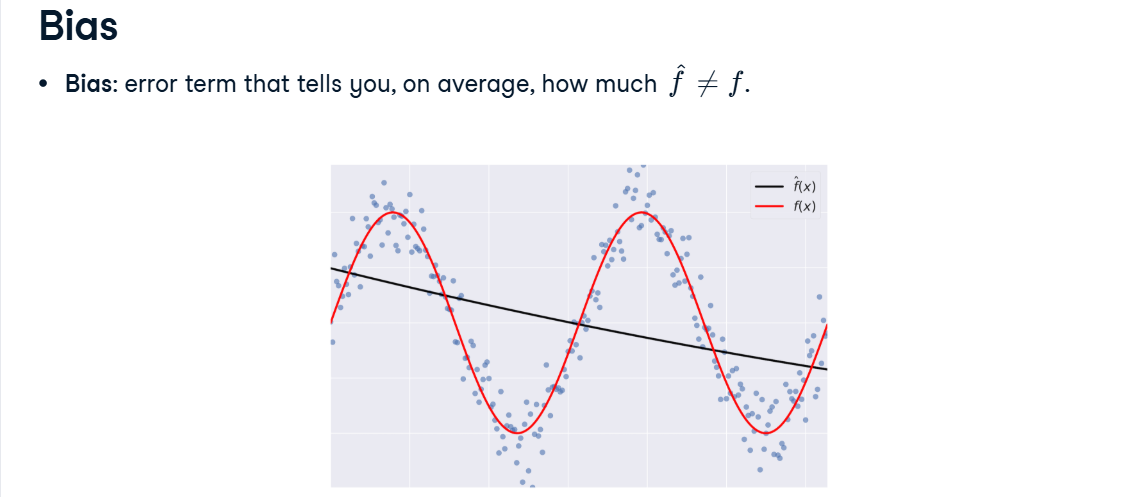




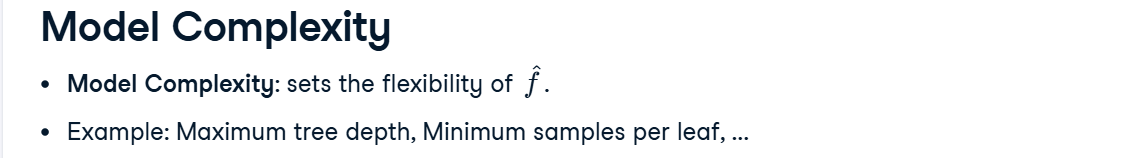


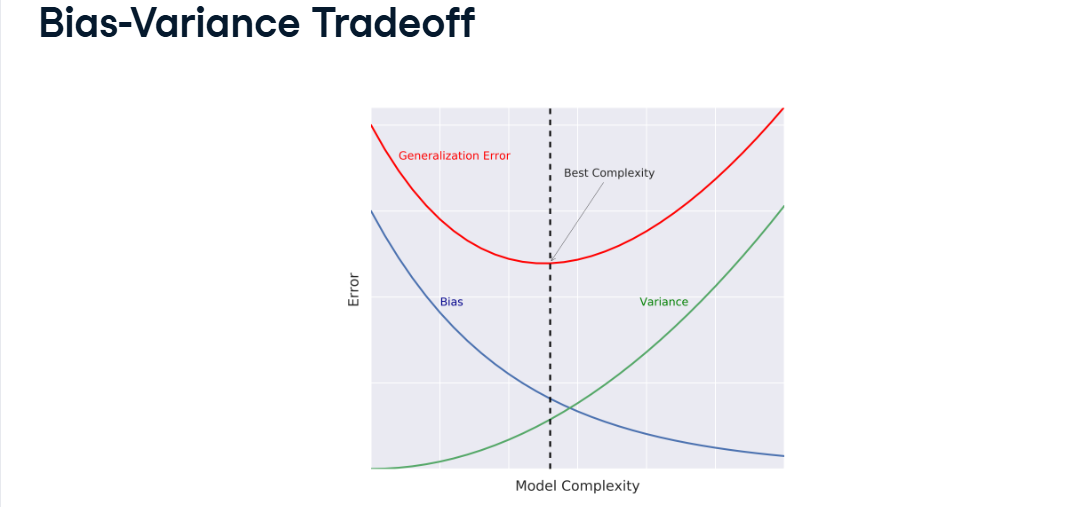
 **Overfitting Underfitting**

****

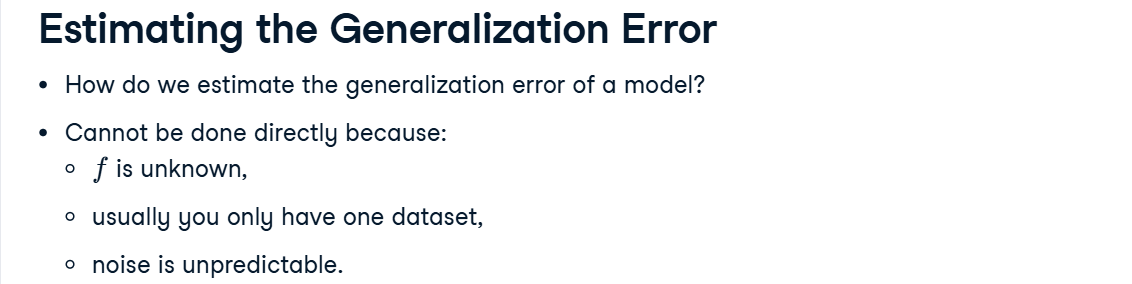


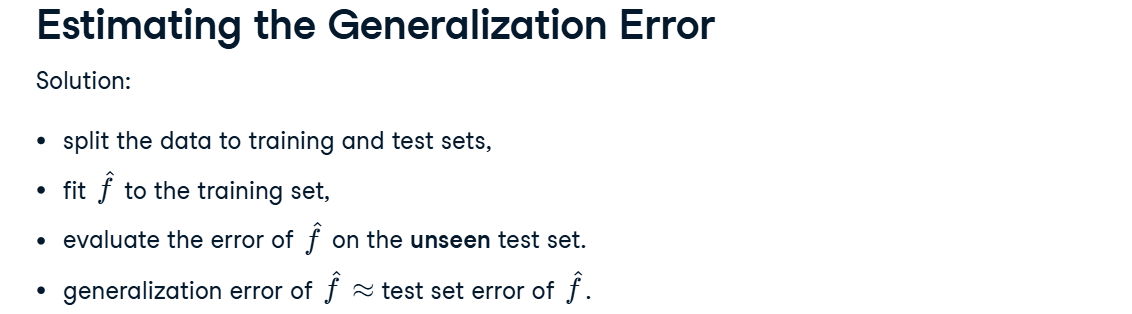


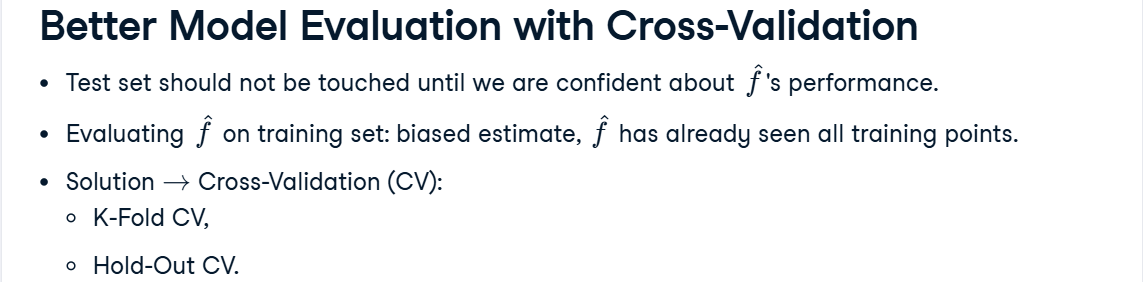


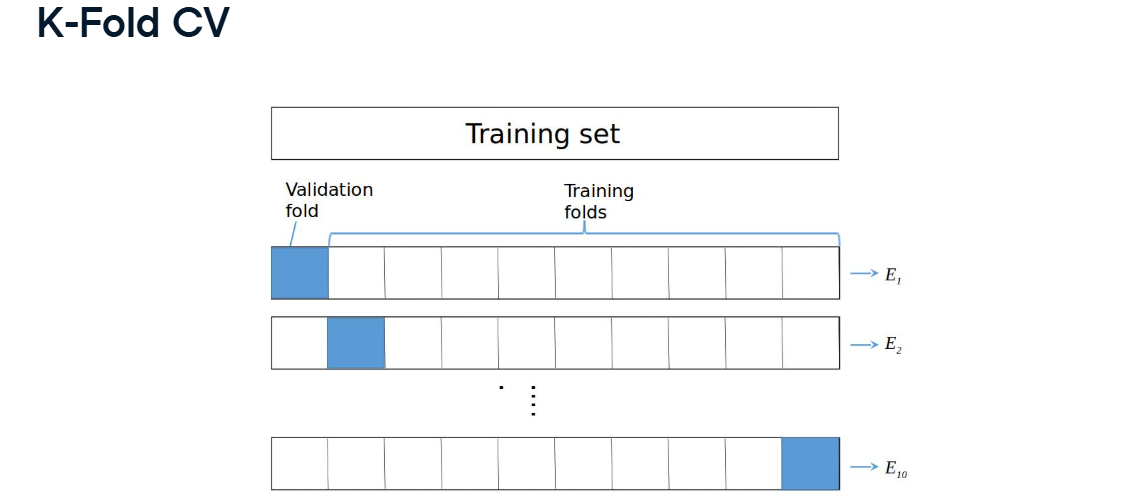




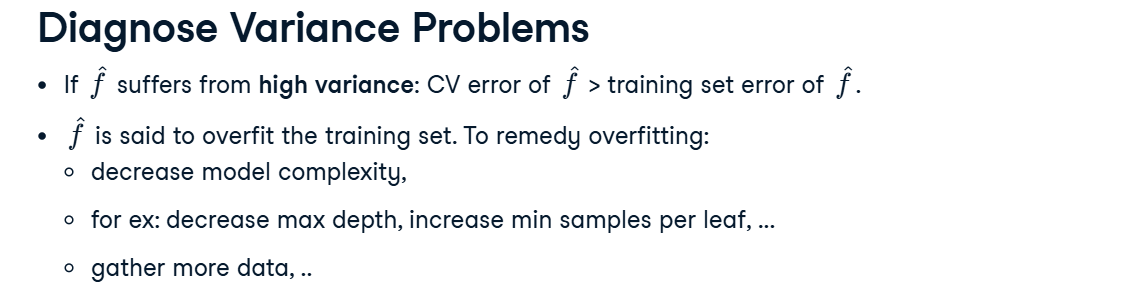


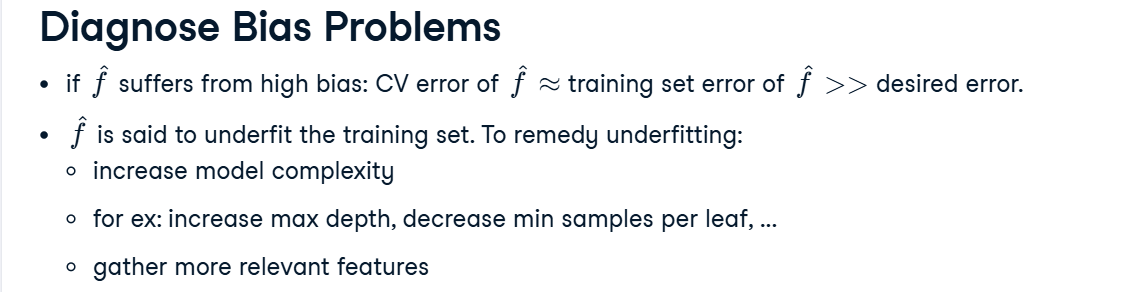


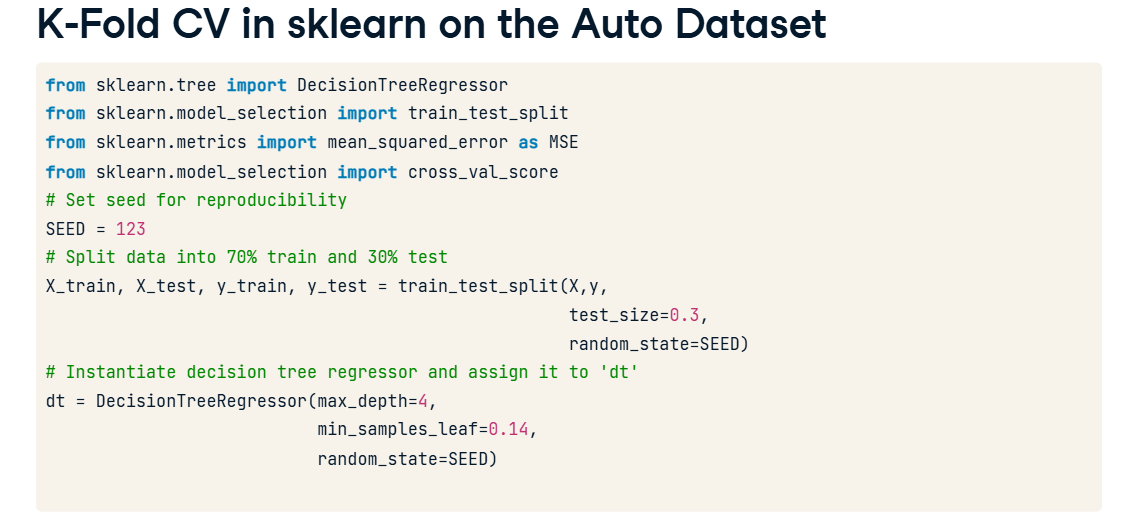


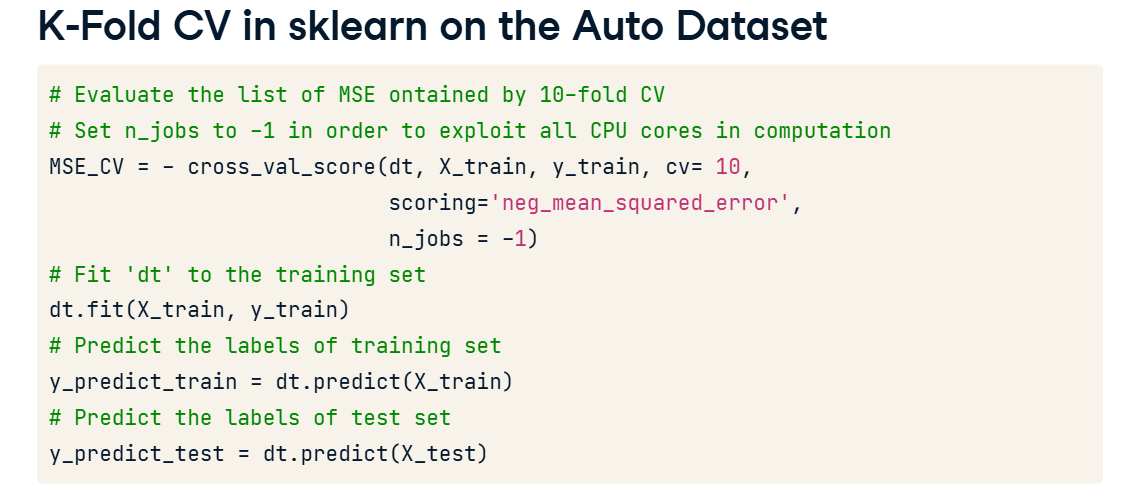


CV error = , We calculate the mean of all cv errors.





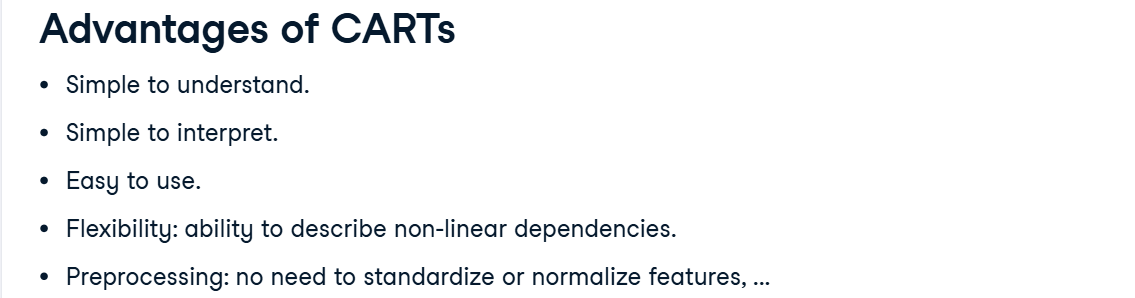


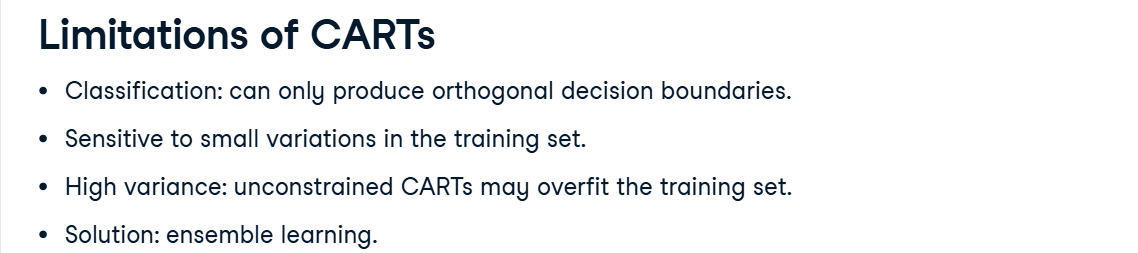


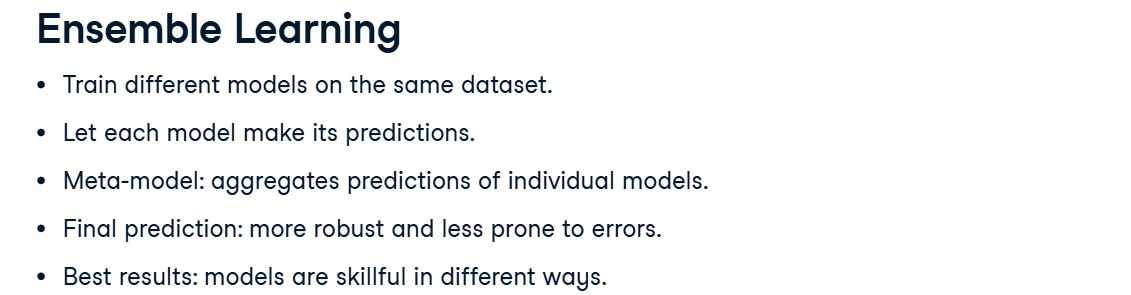


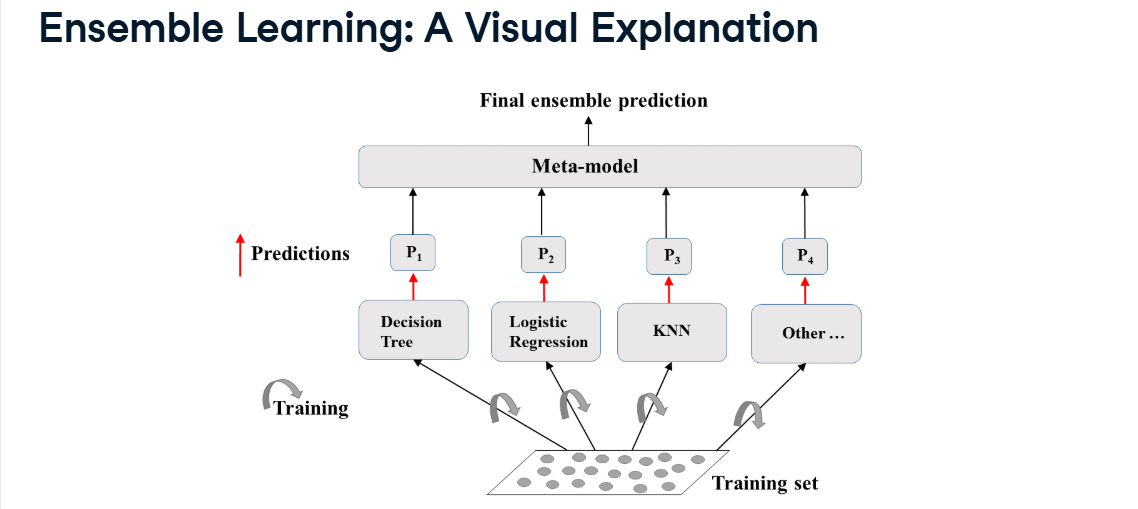
As CV error > Training set error , so the model suffers from high variance (overfitting).

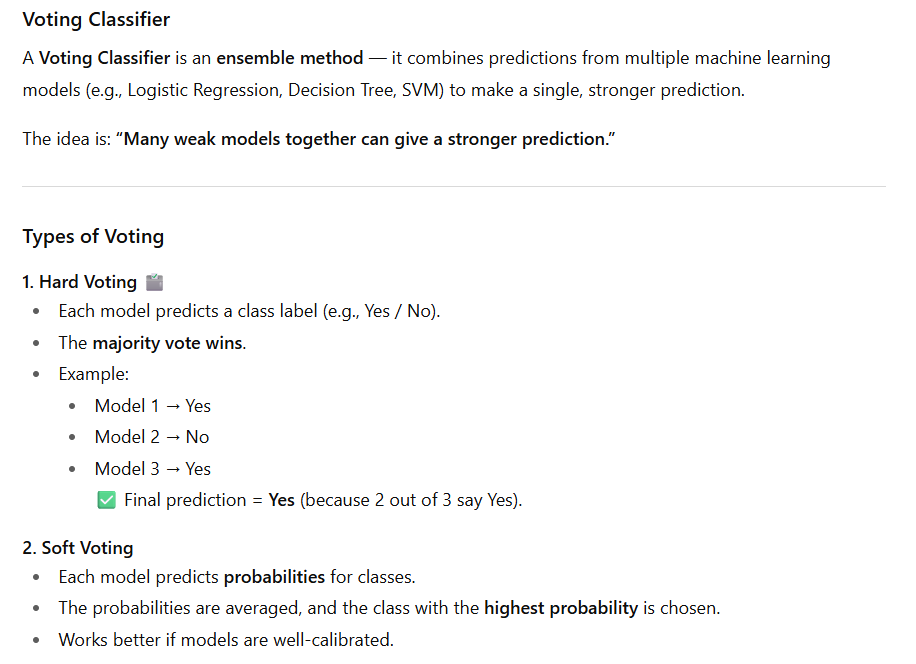
**Ensemble Learning**

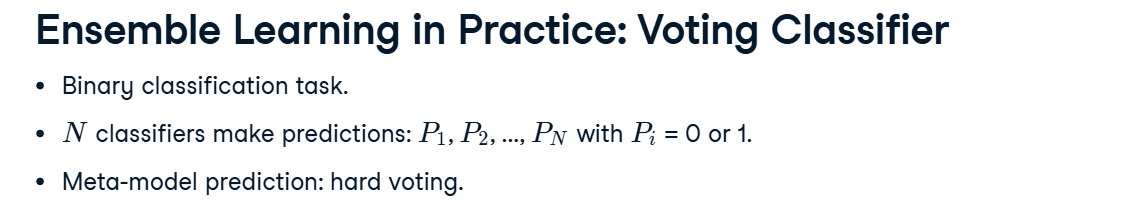


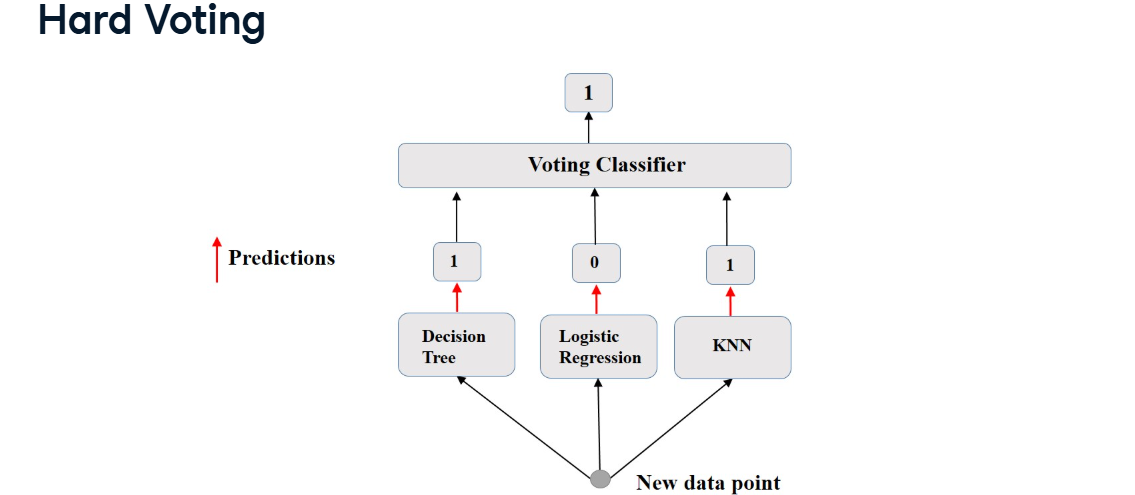


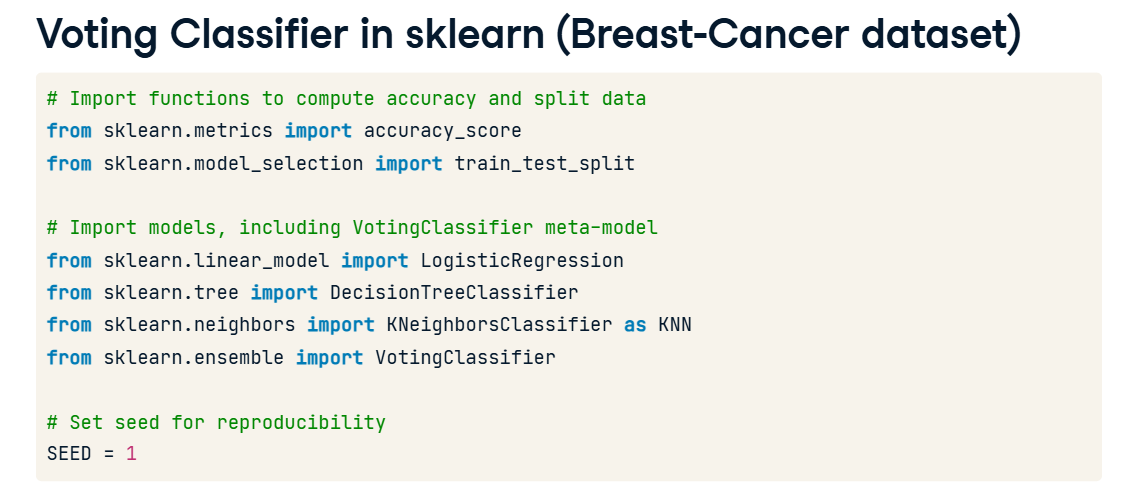


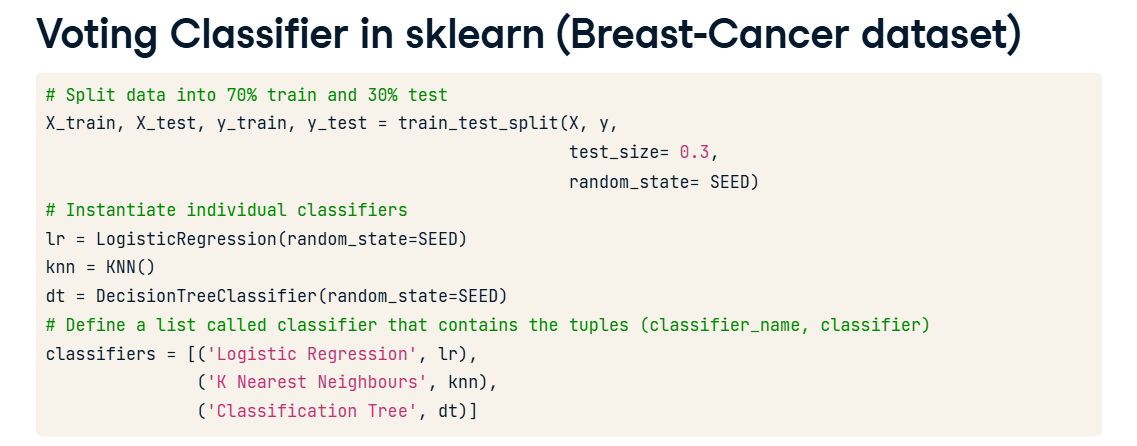


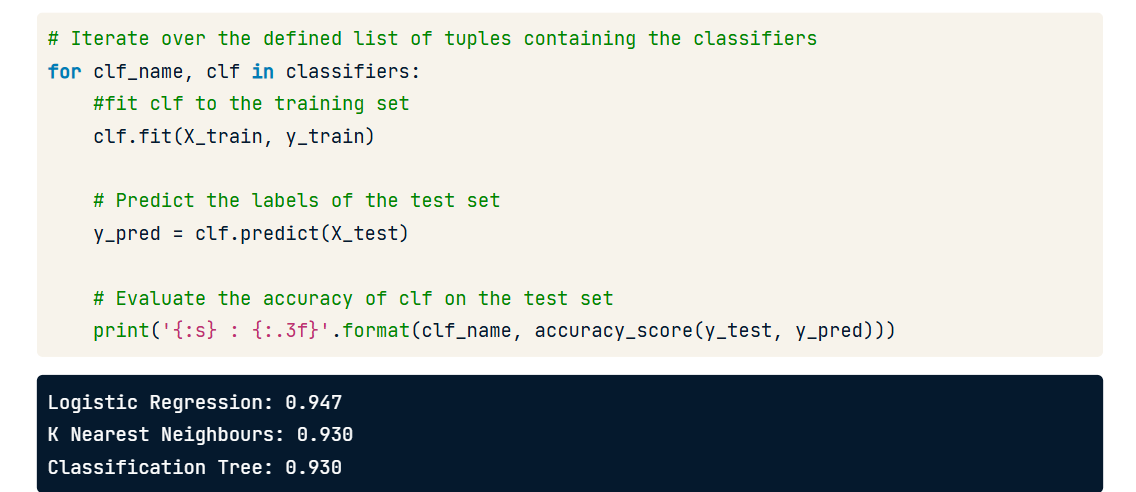


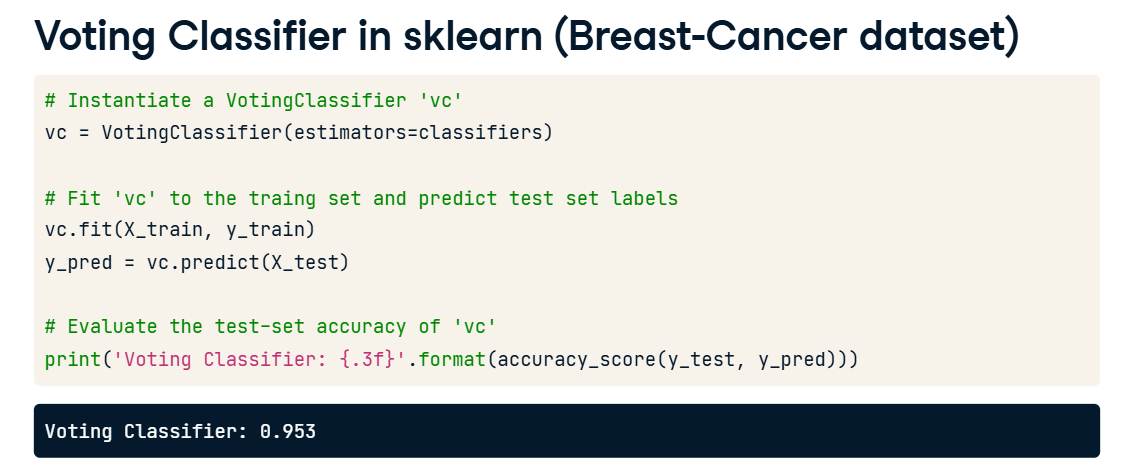












This accuracy is higher than the accuracy achieved by any individual model.

**Bagging**

**🌳 Bagging (Bootstrap Aggregation) — Short & Simple**

* **Bagging** is a type of **ensemble learning** method.
* It uses **the same model** (like Decision Tree) and trains **many copies** of it on **different random samples** of the **same dataset** (bootstrapped data).
* Each model gives its own prediction → all predictions are **combined** (majority vote for classification or average for regression).
* This helps reduce **overfitting** and makes predictions more **stable and accurate**.
* Reduces variance of individual models in the ensemble.

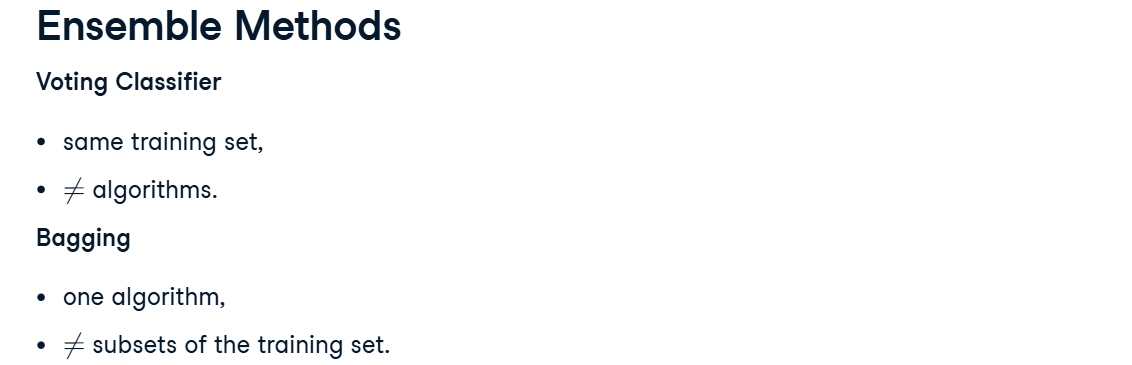
🧩 Example:  
**Random Forest** = Bagging applied to Decision Trees.

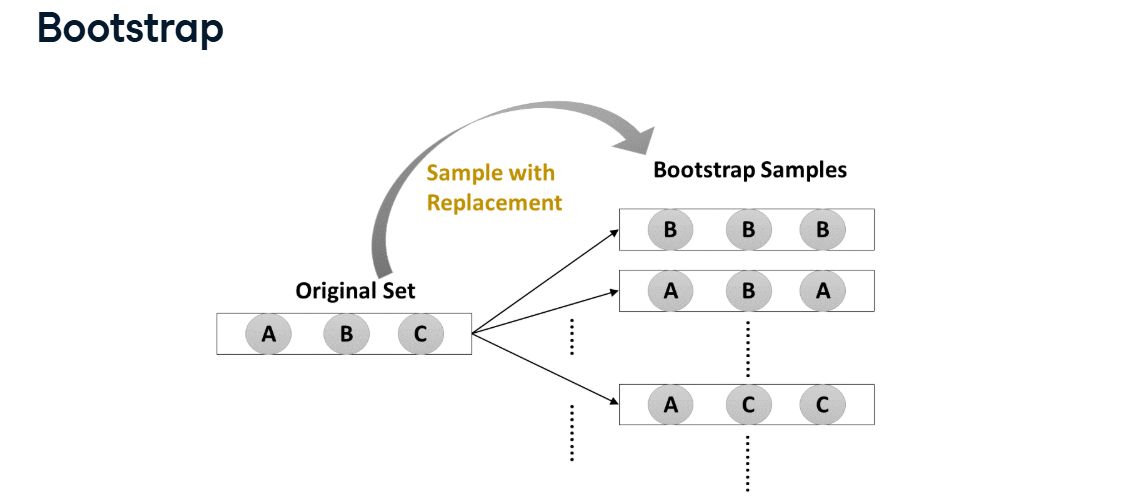
**🔹 Difference from Ensemble Learning**

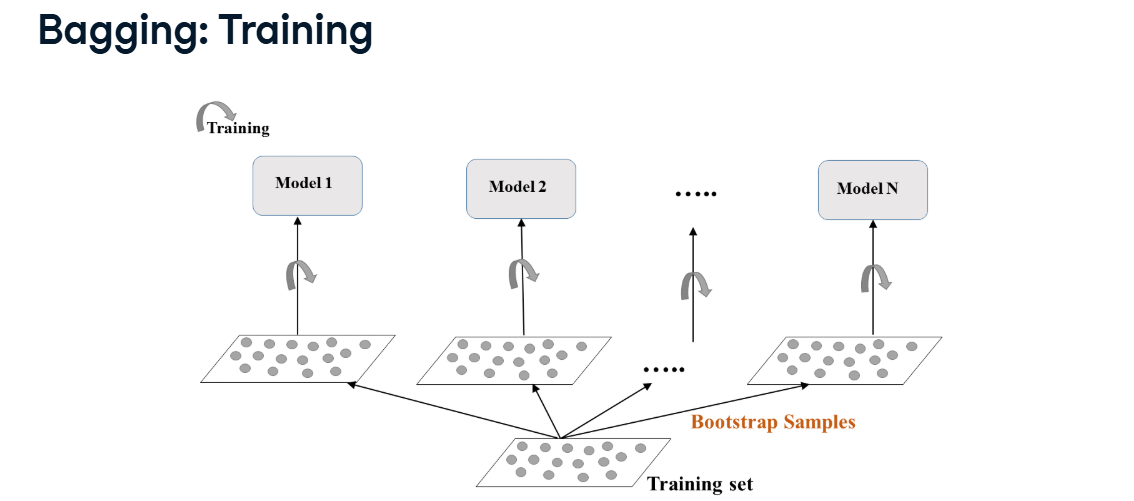
| **Concept** | **Meaning** |
| --- | --- |
| **Ensemble Learning** | The broad idea of combining multiple models to improve accuracy (includes bagging, boosting, stacking, voting, etc.) |
| **Bagging** | A specific ensemble method that trains many **same models** on **different random subsets** and **aggregates** their results. |

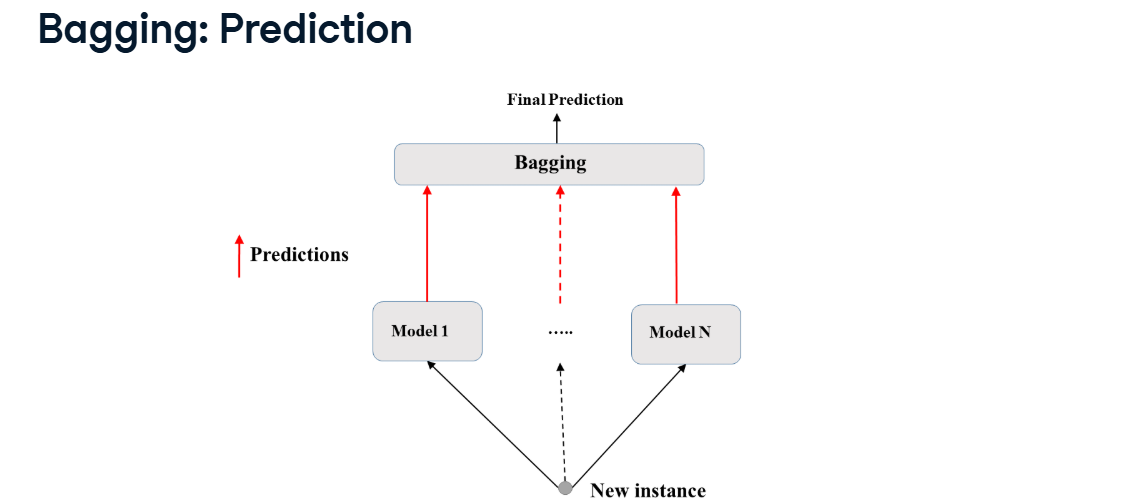
👉 **In short:**

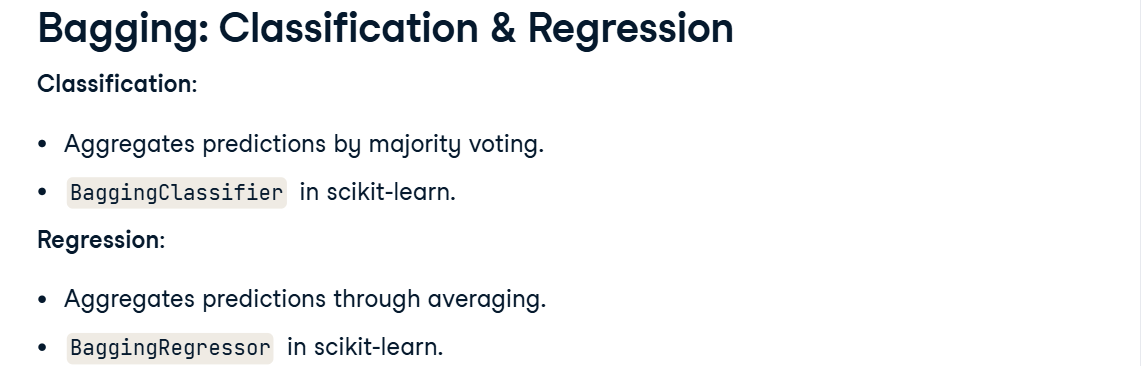
Bagging = “Train same model on different random samples and combine their outputs.”  
Ensemble = “Any method that combines multiple models (bagging is one of them).”



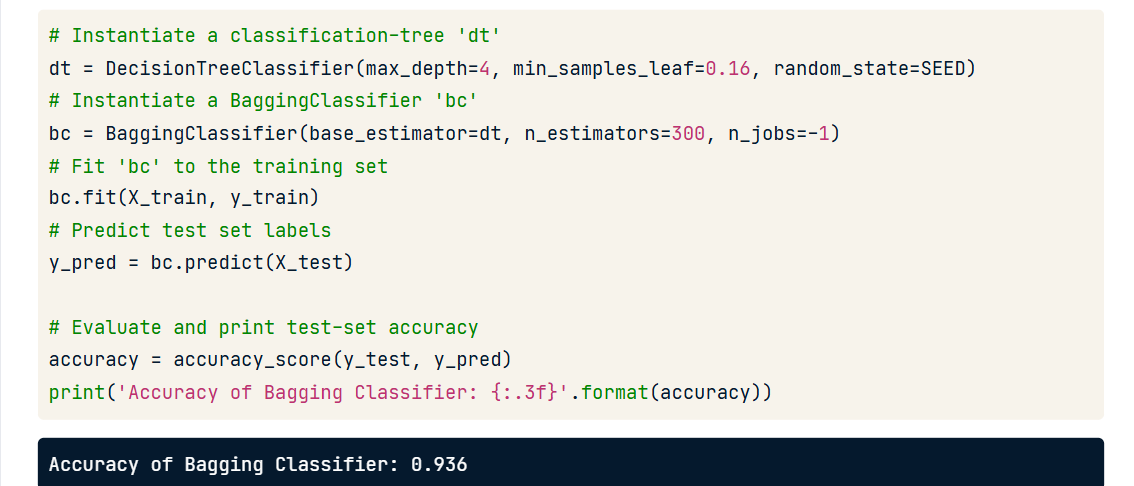






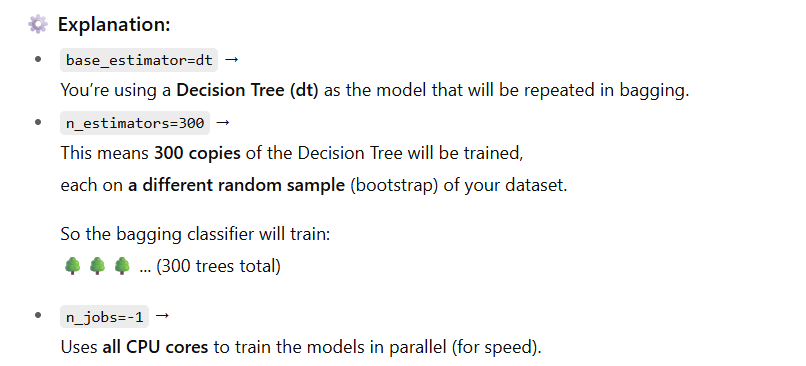






If we train ‘dt’ on the same training set then it would lead to an accuracy of 88.9 % .

Bagging outperforms this best estimator ‘dt’ by about 5% .



**🌱 Out-of-Bag (OOB) Instances**

When Bagging takes random samples from your dataset using **bootstrapping (sampling with replacement)**:

* Some data points are picked **multiple times**.
* Some data points are **never picked** for that bootstrap sample.

👉 The data points **not selected** for a particular model are called **Out-of-Bag (OOB) instances** for that model.

* On average, for each model, 63% of training instances are sampled.
* The remaining 37% constitute to the OOB instances.

**🎯 OOB Evaluation**

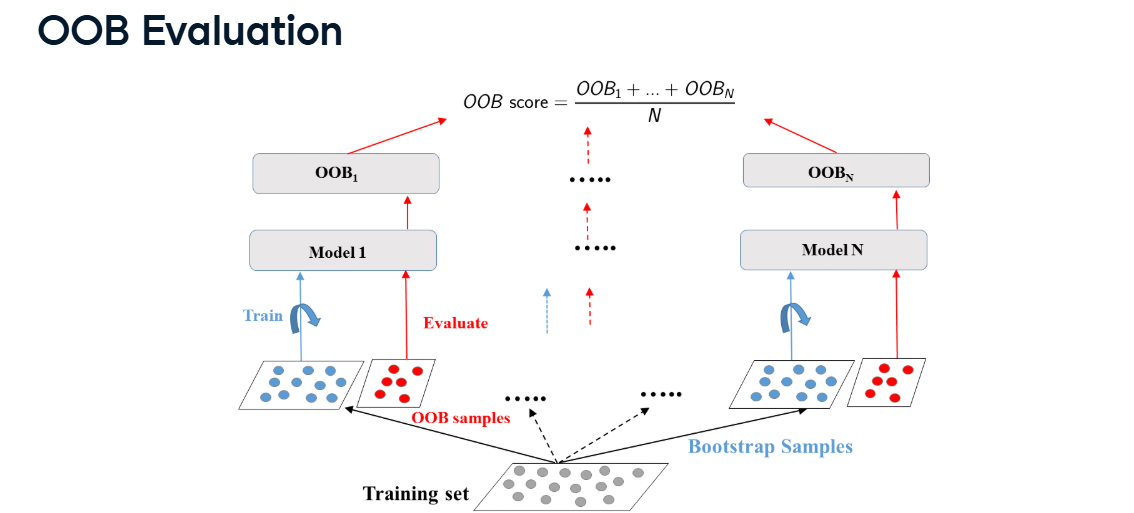
Instead of using a separate test set, we can use these **OOB instances** to estimate model performance.

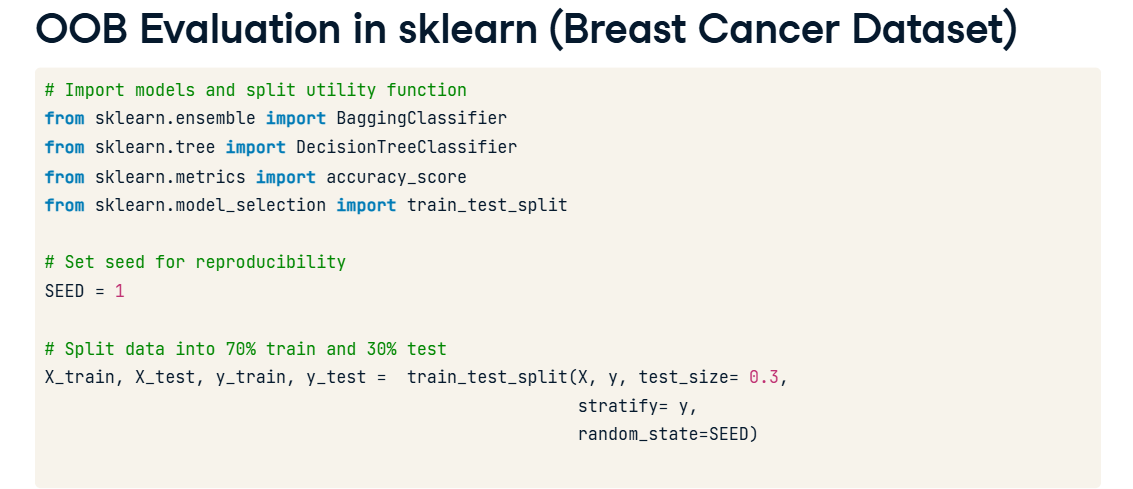
Here’s how:

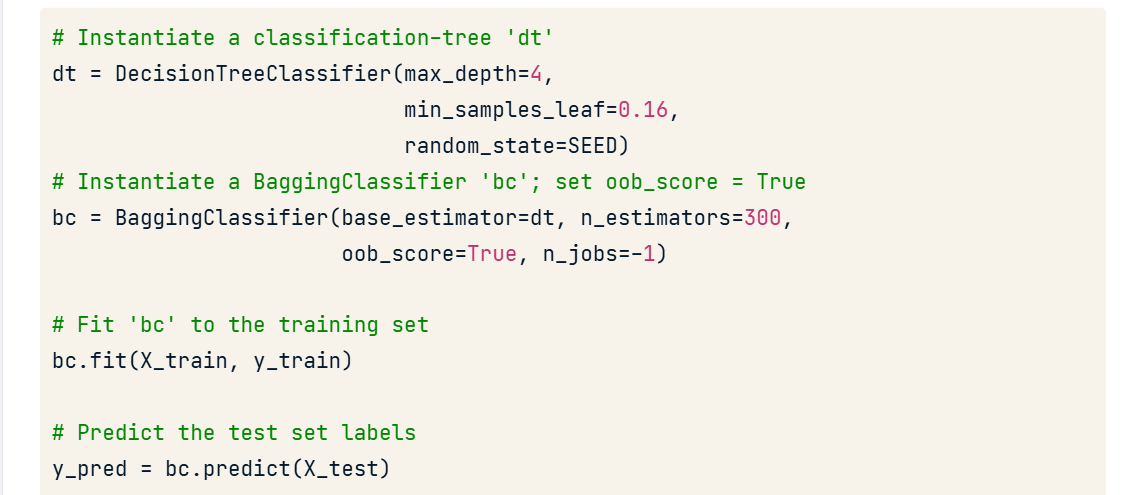
1. Each model (e.g., tree) is trained on its bootstrap sample.
2. That model predicts on its **OOB data** (the rows it never saw).
3. The predictions from all models are combined to estimate the overall **OOB score (accuracy or error)**.

**✅ Why OOB Evaluation is Useful**

* Gives an **unbiased estimate of test accuracy**.
* Saves data — no need to make a separate validation/test split.
* Often close to cross-validation performance.







Set ‘oob\_score = True’ to evaluate OOB accuracy after training.



**Random Forests**

**🌳 Random Forest — Simple Explanation**

A **Random Forest** is an **ensemble of many Decision Trees**, combined to make a stronger, more accurate model.

It’s basically **Bagging + extra randomness** 🌟

**⚙️ How It Works (Step-by-Step)**

1. **Bootstrap sampling:**
   * Just like in bagging, it takes **random samples (with replacement)** from the dataset for each tree.
   * So each tree gets a slightly different version of the data.
2. **Random feature selection:**
   * When a tree splits a node, it doesn’t look at *all* features.
   * Instead, it looks at a **random subset of features** each time.
   * ‘d’ features are sampled at each node without replacement

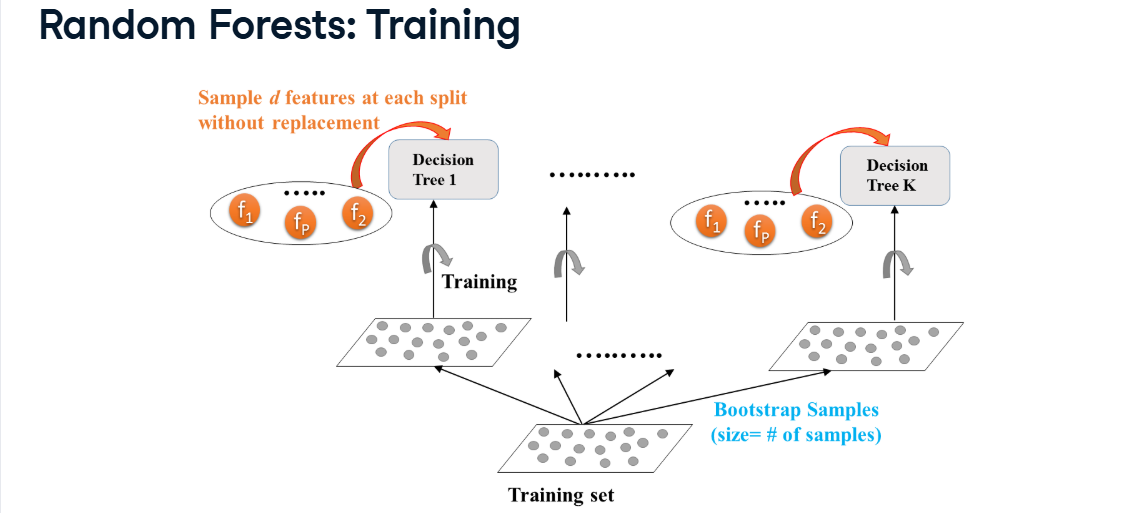
(d < total no. of features)

* + This adds *extra randomness* → makes trees more different and reduces correlation between them.

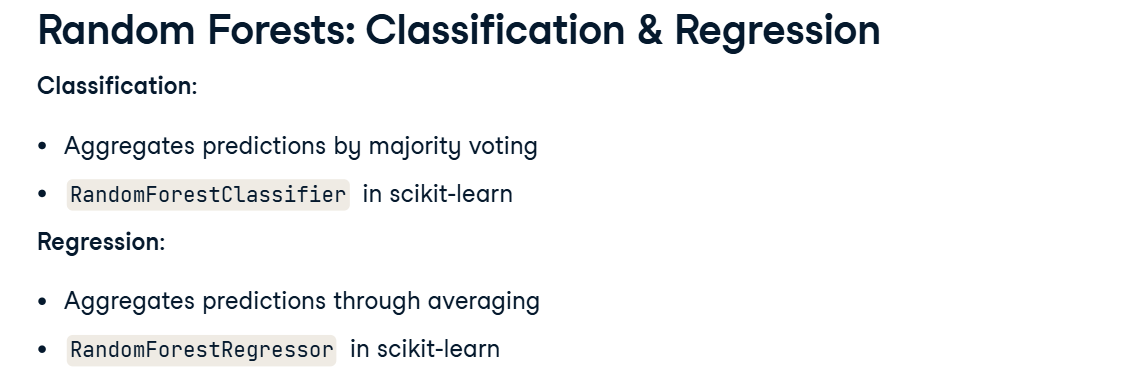
1. **Many trees trained:**
   * It trains many (like 100 or 300) Decision Trees independently.
2. **Combine results:**
   * For **classification** → takes the **majority vote**.
   * For **regression** → takes the **average** of all tree outputs.

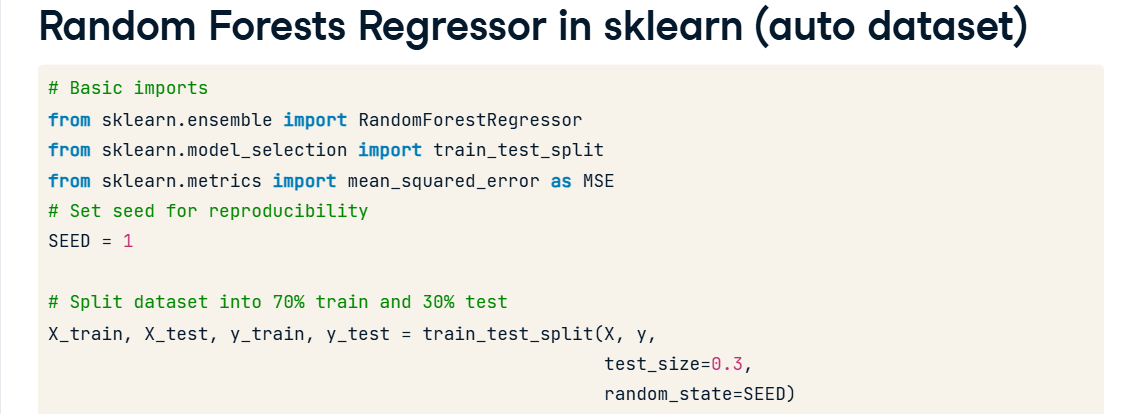
**✅ Why Random Forest is Great**

* Reduces **overfitting** (since trees are decorrelated).
* Improves **accuracy** and **stability**.
* Works well even without fine-tuning.
* Supports **OOB evaluation** (built-in accuracy check).

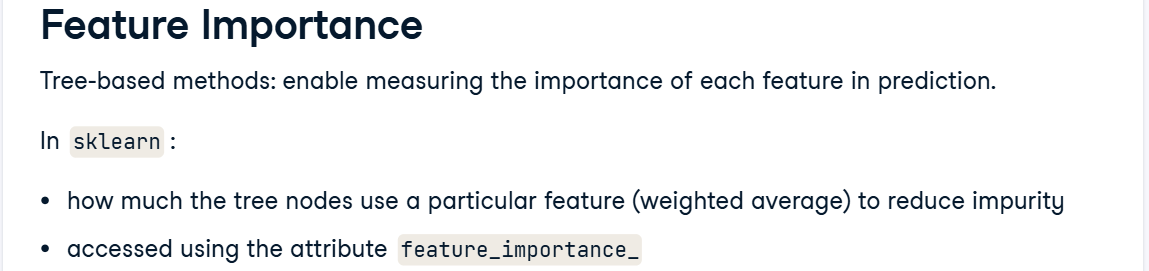




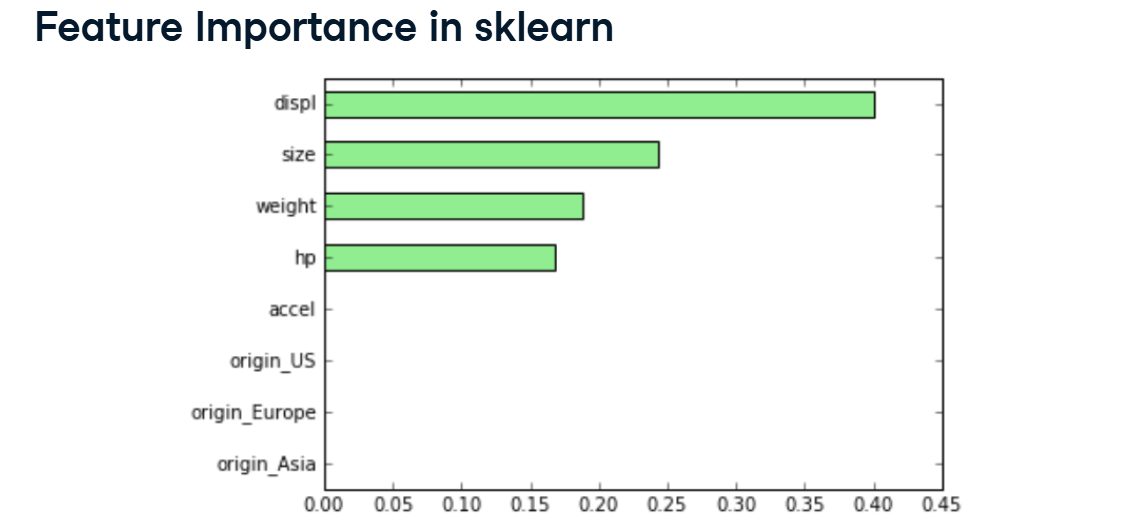












**⚙️ AdaBoost (Adaptive Boosting)**

**🔹 What it is:**

**AdaBoost** is a type of **Boosting algorithm** — another ensemble learning method (like Bagging),  
but instead of training models *independently*, it trains them **sequentially**, each one learning from the mistakes of the previous one.

**🧠 Main Idea**

“Train weak models one after another, and make each new model focus on the mistakes made by the previous models.”

**⚙️ How AdaBoost Works (Step-by-Step)**

1. **Start:**
   * Give **equal weight** to all training samples.
2. **Train the first weak model:**
   * Usually a **simple Decision Tree** (called a *stump* — just 1 split).
   * Predict the data.
3. **Find the errors:**
   * Identify which samples were predicted **wrong**.
4. **Adjust weights:**
   * Increase weights of the **misclassified** samples (so the next model pays more attention to them).
   * Decrease weights of the **correctly classified** ones.
5. **Train the next model:**
   * Focus more on the “hard” samples (the ones previously misclassified).
6. **Combine all models:**
   * The final prediction is a **weighted sum** of all weak models.
   * Better-performing models get **higher weight**.

**🌟 Result**

* Models are weak individually (simple trees),  
  but when combined, they form a **strong and accurate model**.
* Works well for **classification** and **regression**.

**🔹 Example Intuition:**

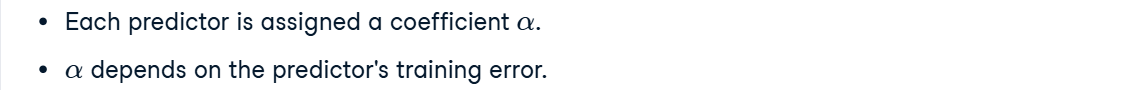
Imagine a teacher:

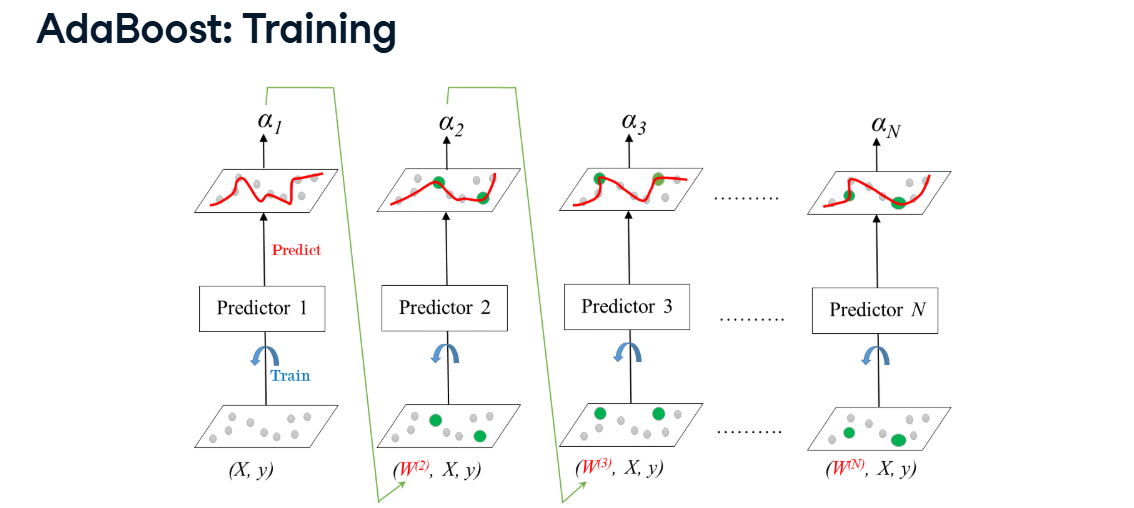
* First explains a topic to all students (equal focus).
* Then notices some students didn’t understand certain parts.
* Next time, the teacher focuses more on those weak students.
* Repeats this until almost everyone gets it right. 🎓

That’s exactly what **AdaBoost** does with data samples.

**✅ Summary Table**

| **Concept** | **Bagging** | **Boosting (AdaBoost)** |
| --- | --- | --- |
| Model training | Parallel (independent) | Sequential (one after another) |
| Focus | Reduce variance | Reduce bias |
| Data sampling | Random subsets | Weighted focus on hard samples |
| Combination | Simple averaging/voting | Weighted combination of models |





**⚙️ Learning Rate (η or eta)**

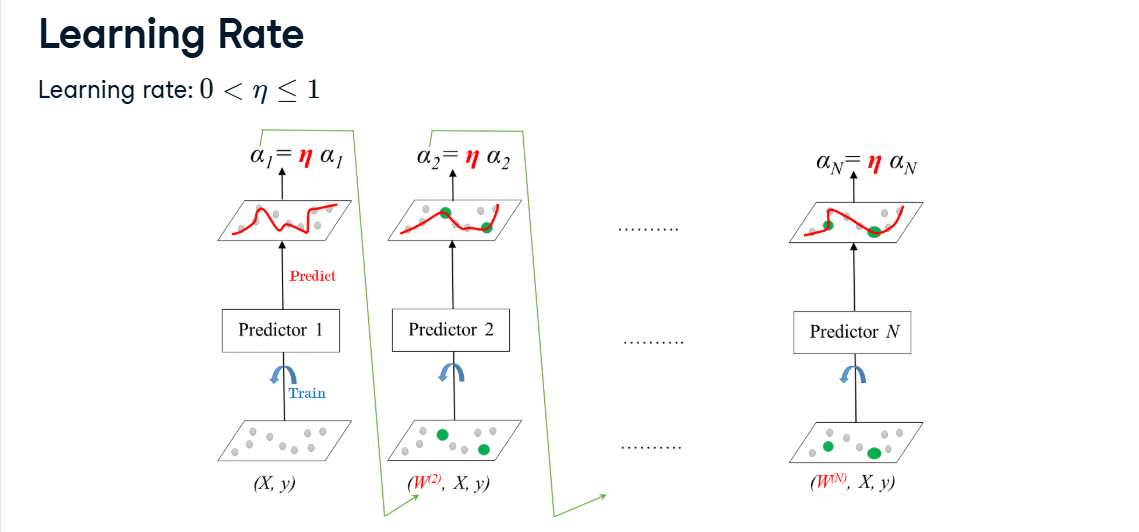
**🔹 Definition:**

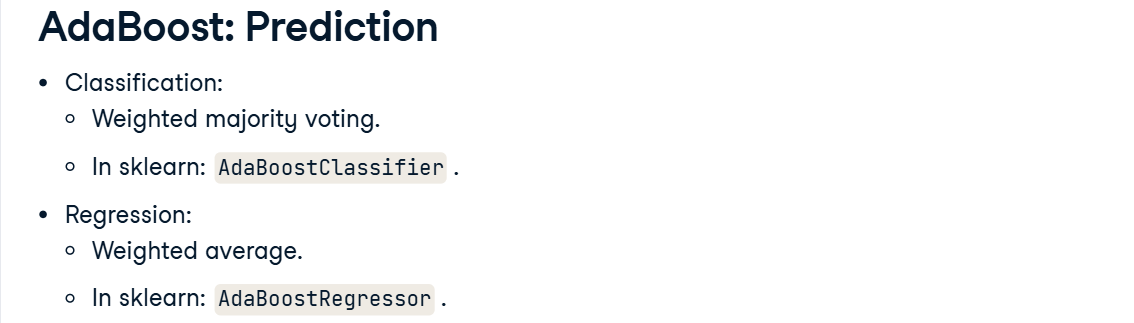
In AdaBoost (and other boosting algorithms), the **learning rate** controls **how much influence each weak learner (tree/stump)** has on the final combined model.

It’s a **scaling factor** that reduces or controls the contribution of each new model.

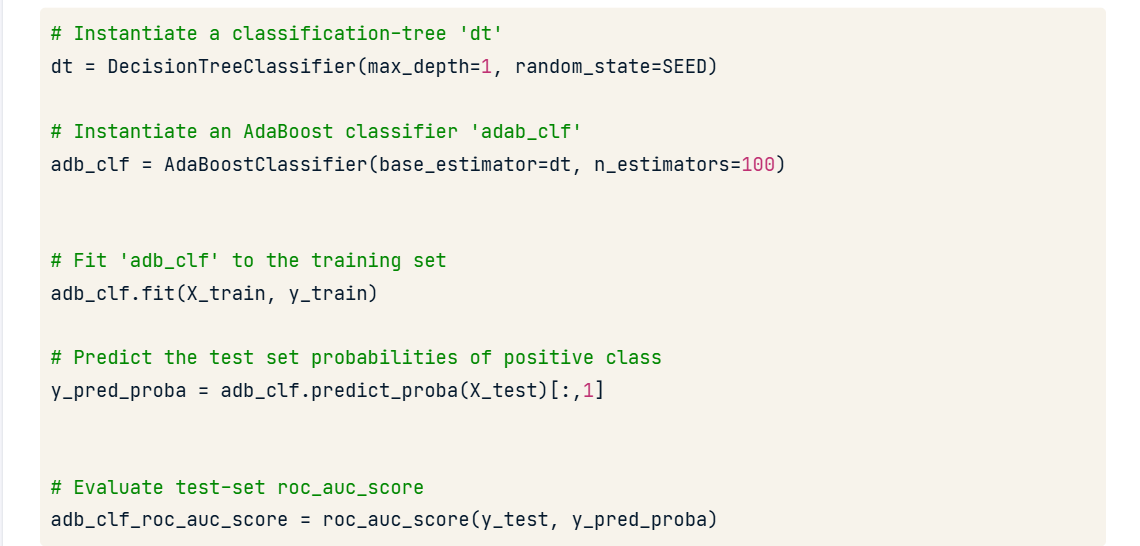
**🌟 Effect of Learning Rate:**

| **Learning Rate** | **What Happens** |
| --- | --- |
| **High (η → 1)** | Learns faster but may overfit (too aggressive). |
| **Low (η → 0.1 or 0.01)** | Learns slower but more stable and generalizes better. |
|  |  |



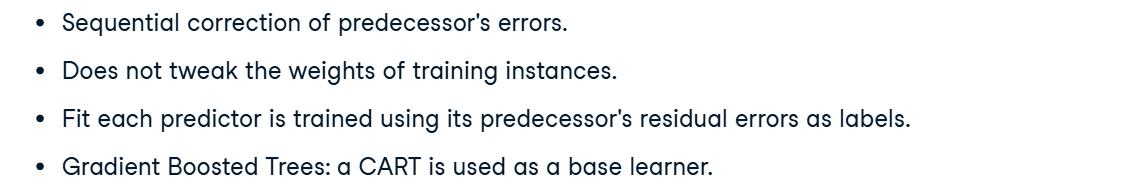








**Gradient Boosting**

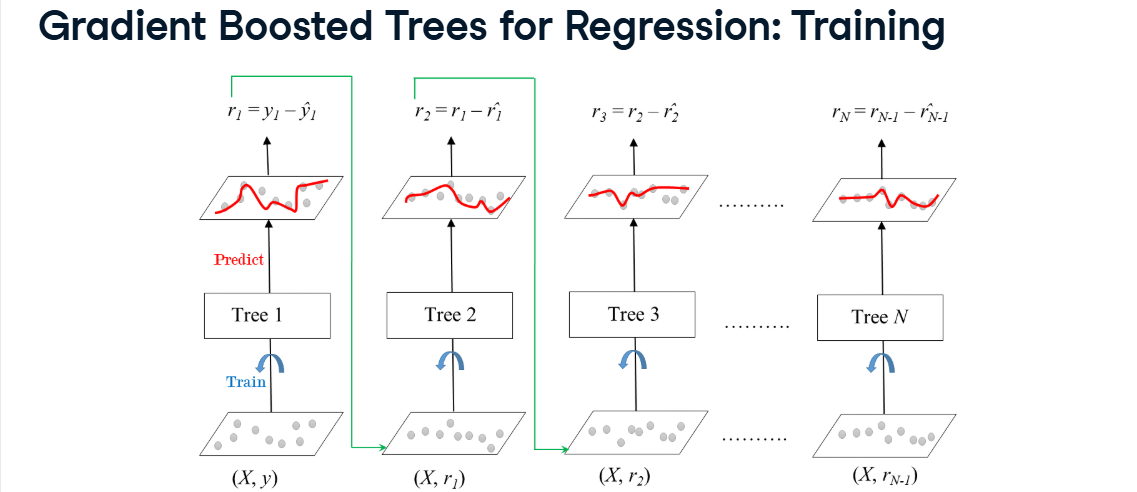


**⚙️ How it works (in 3 simple steps):**

1. **Start simple**
   * Begin with a basic model (like predicting the average value).
   * This model will make some errors.
2. **Learn from mistakes**
   * The next model is trained to **predict the errors (residuals)** made by the previous model.
   * This means it’s focusing on what the last model did wrong.
3. **Add improvements gradually**
   * Keep adding new small models (trees) that correct the mistakes of the previous ones.
   * Each new tree “boosts” the performance a little bit.
   * After many steps, you get a strong, accurate model.

**🧠 Why “Gradient”?**

Because each new model is trained in the **direction of the gradient** of the loss function (i.e., it reduces the loss step by step — like gradient descent in optimization).



**⚙️ Shrinkage (also called Learning Rate)**  
👉 Each new tree’s contribution to the final model is **reduced (shrunk)** by multiplying it with a small number (the **learning rate**, often called eta).

**📉 Mathematically:**

When adding a new tree’s prediction:

where

* : current model,
* : new tree (predicts residuals/errors),
* : **learning rate** which is btwn 0 and 1. (e.g., 0.1, 0.05, etc).

**💡 Meaning:**

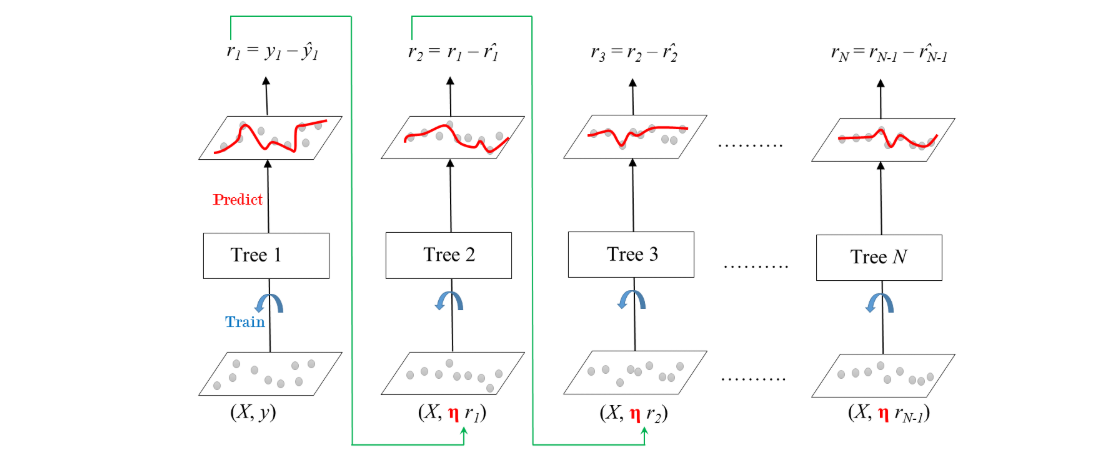
Shrinkage means we take **smaller correction steps** — we don’t fully trust each new tree.  
So, each tree makes **a small adjustment** instead of a big one.

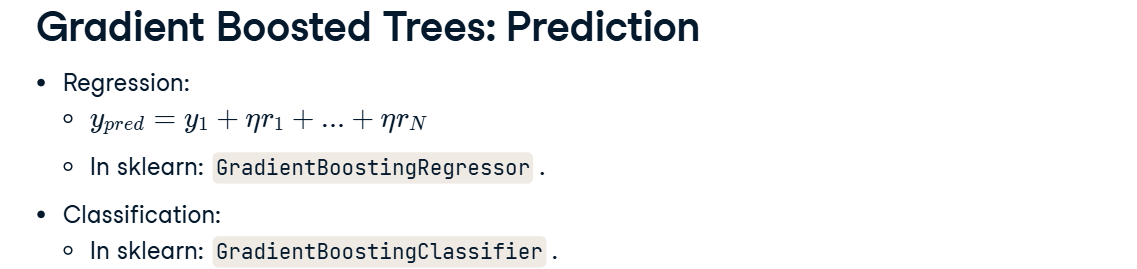
**🧠 Why it’s useful:**

* It **slows down learning**, making the model more **stable**.
* Prevents **overfitting**.
* Usually combined with **more trees** (since each tree has smaller impact).

**⚡ Example:**

* Learning rate = 1.0 → take big jumps (may overshoot).
* Learning rate = 0.1 → take smaller steps (learn slowly but carefully).







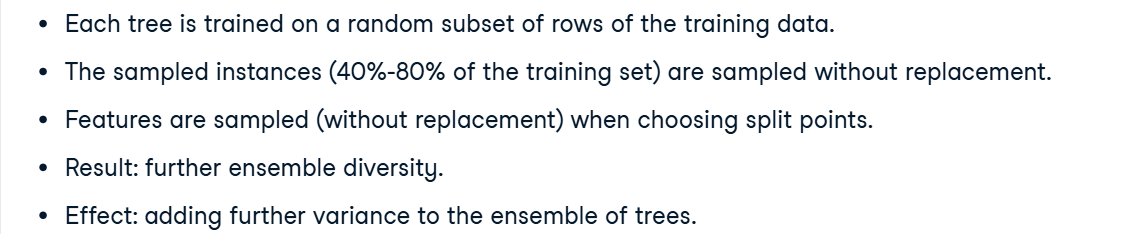


**⚡ Stochastic Gradient Boosting (SGB) — the smarter version**

It adds **randomness** to make the model more **robust and faster**.

It does this by using:

1. **A random subset of training data** (called *row subsampling*), and/or
2. **A random subset of features** (*column subsampling*)  
   when training each tree.



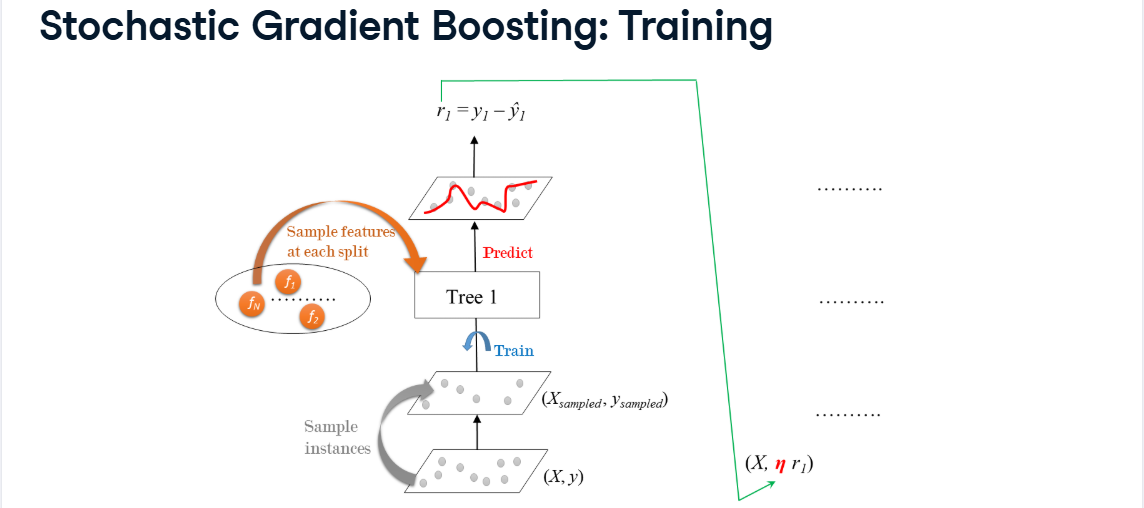
**🧠 Why this helps:**

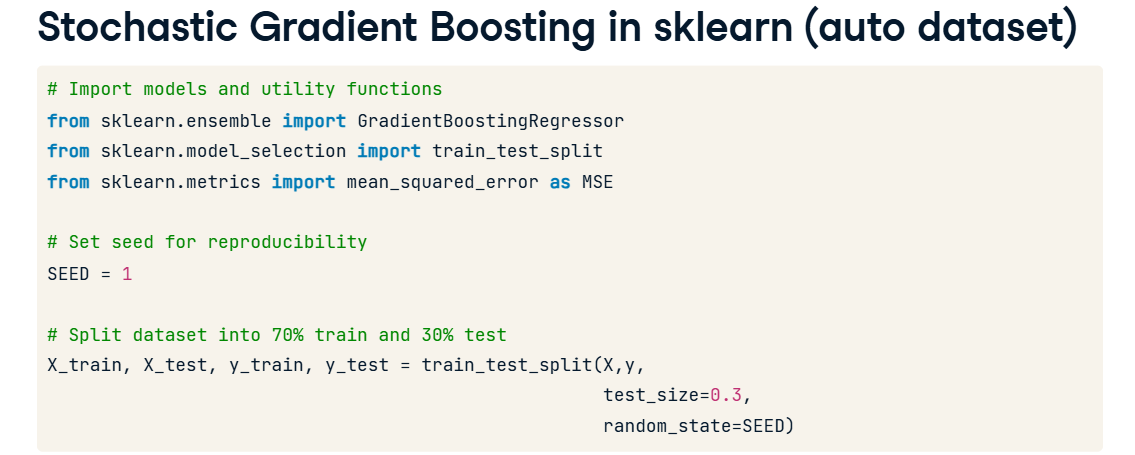
* Reduces **overfitting** (trees don’t all see the same data).
* Makes training **faster**.
* Adds a bit of **diversity** like in Random Forests.

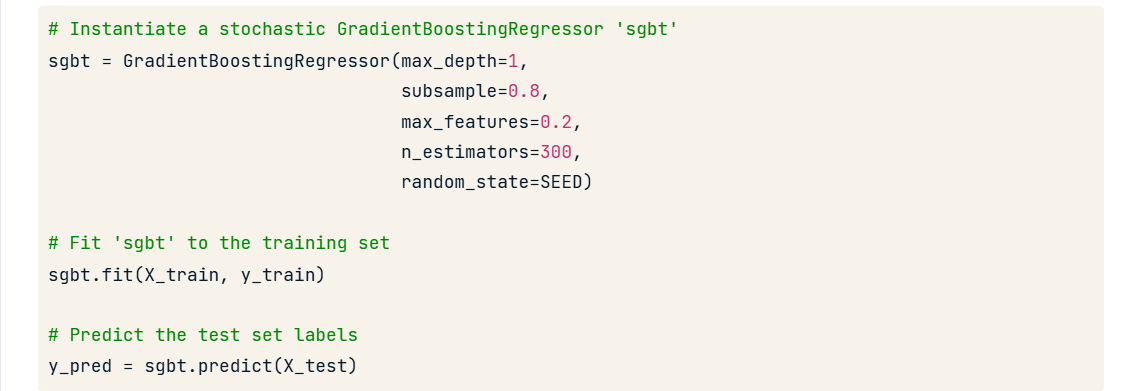
**🧩 Example:**

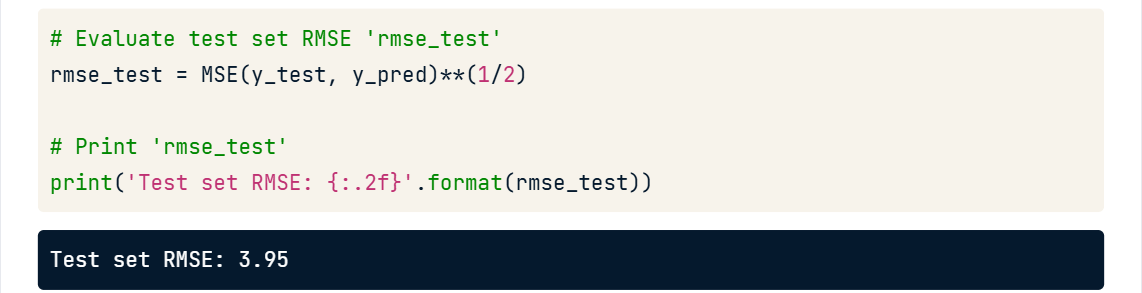
Suppose you have 1000 data points.

* Normal GB → every tree uses all 1000 points.
* Stochastic GB → each tree may use only 70–80% of them (randomly selected).









 **max\_depth=1** → Each tree is shallow (a decision stump).

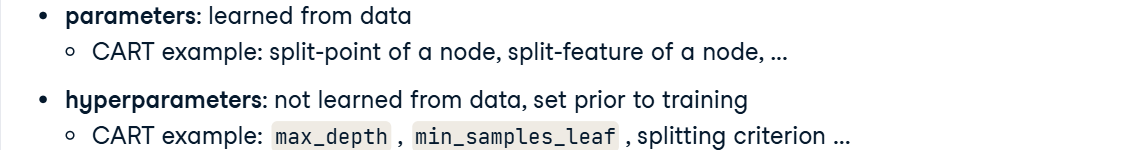
 **subsample=0.8** → Uses 80% of training data for each tree to add randomness.

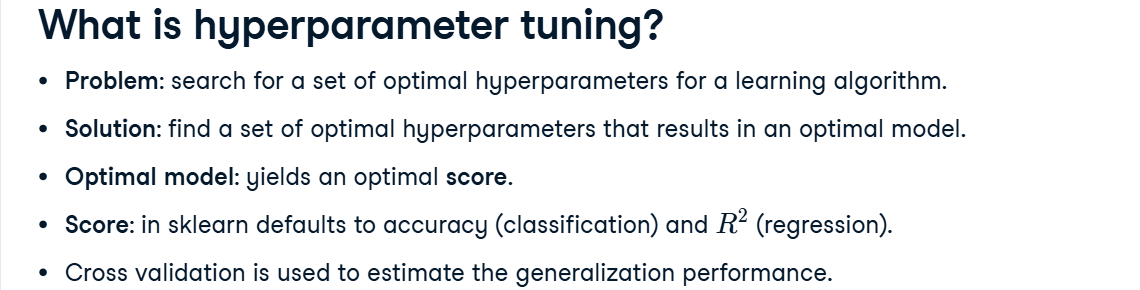
 **max\_features=0.2** → Each tree uses 20% of features randomly when splitting.

 **n\_estimators=300** → Builds 300 trees sequentially.

 **random\_state=SEED** → Ensures same random results every run.

**Tuning a CART’s Hyperparameters**





**🌳 Tuning a CART’s hyperparameters**

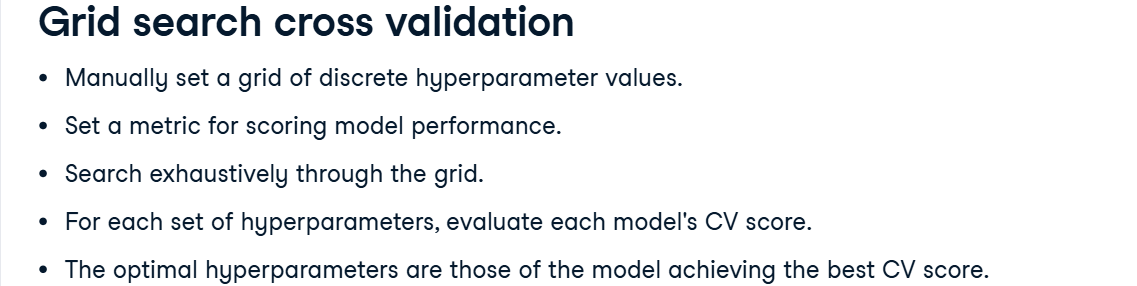
* **CART (Classification and Regression Tree)** models have settings called **hyperparameters** (like max\_depth, min\_samples\_split, etc.) that control how the tree grows.
* **Tuning** means trying different combinations of these hyperparameters to find the best-performing model.
* Example: A deep tree might overfit; a shallow one might underfit — tuning finds the right balance.

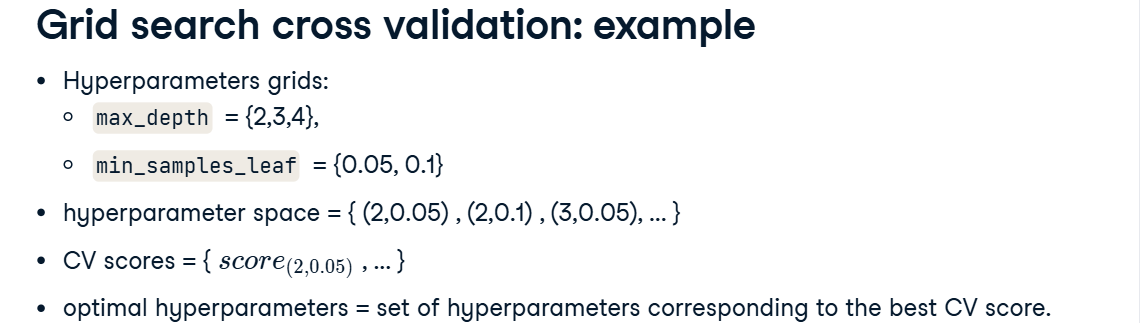
**🔍 Grid Search with Cross-Validation (GridSearchCV)**

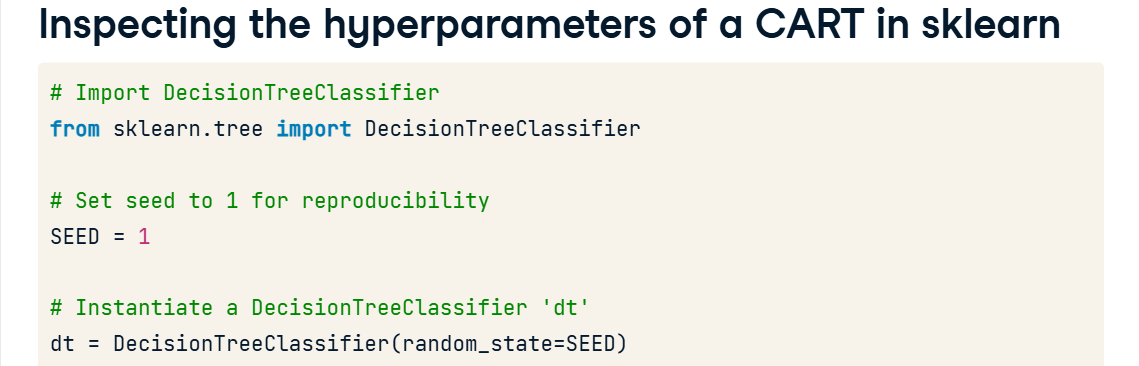
* **Grid Search** automatically tests all possible combinations of chosen hyperparameter values.
* **Cross-Validation (CV)** splits the data into multiple folds to ensure the model’s performance is consistent, not just by luck.
* Together, **GridSearchCV** finds the **best hyperparameter set** that gives the **highest accuracy (or lowest error)** on validation data.

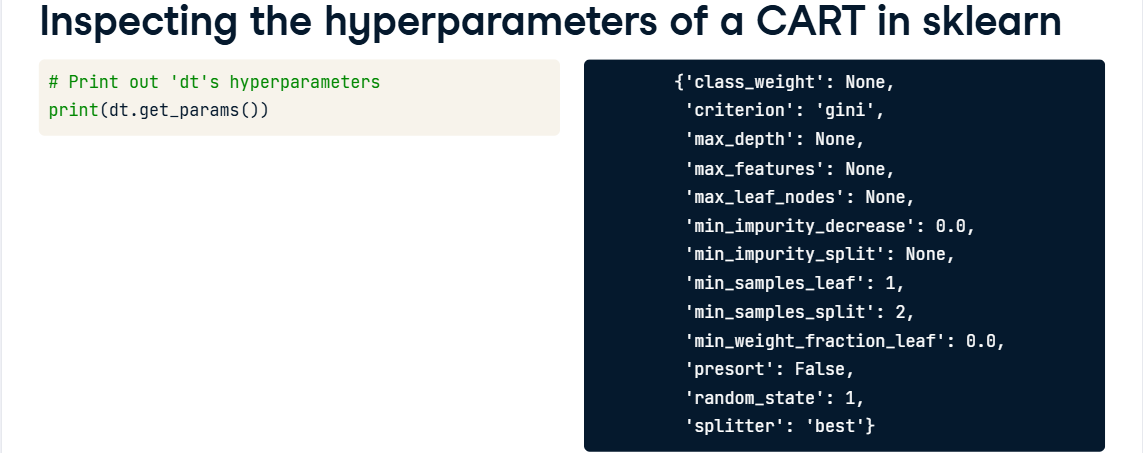
✅ **In short:**  
**GridSearchCV** = an automatic way to **test many CART settings** → **evaluate each using cross-validation** → **pick the best model.**

**Approaches to Hyperparameter tuning : Grid Search, Random Search, Bayesian Optimization, Genetic Algorithms etc.**



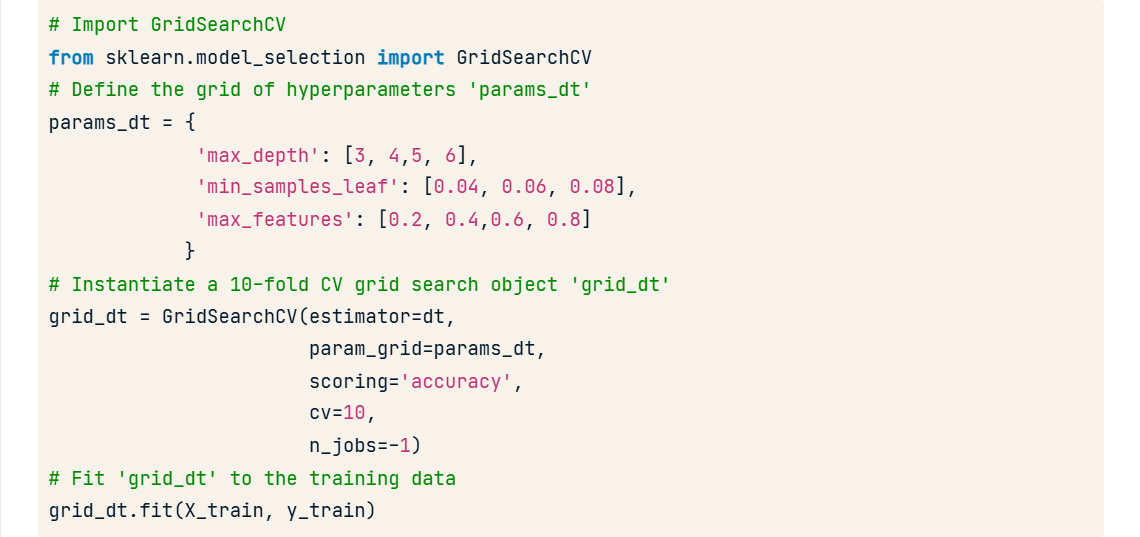


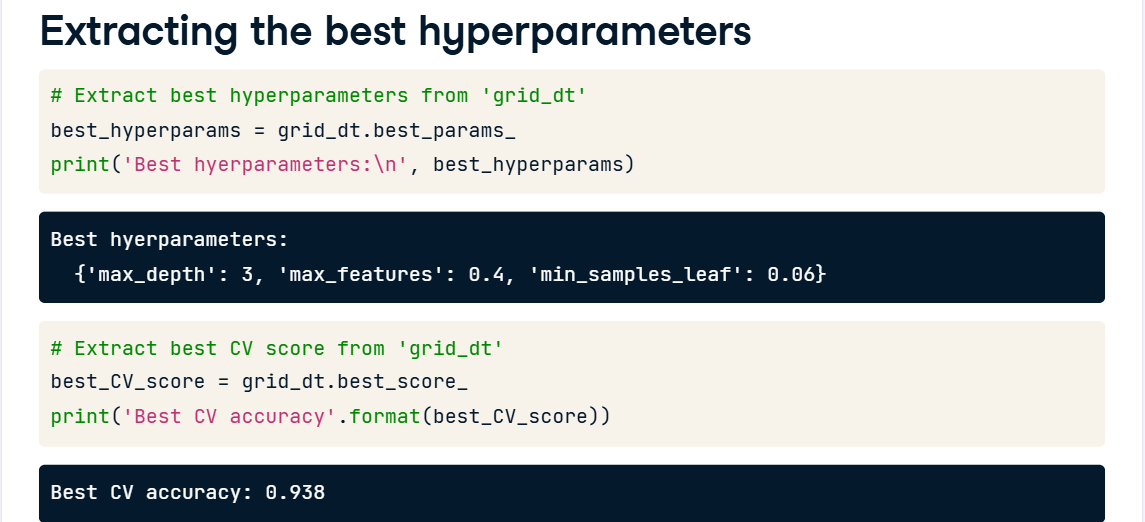


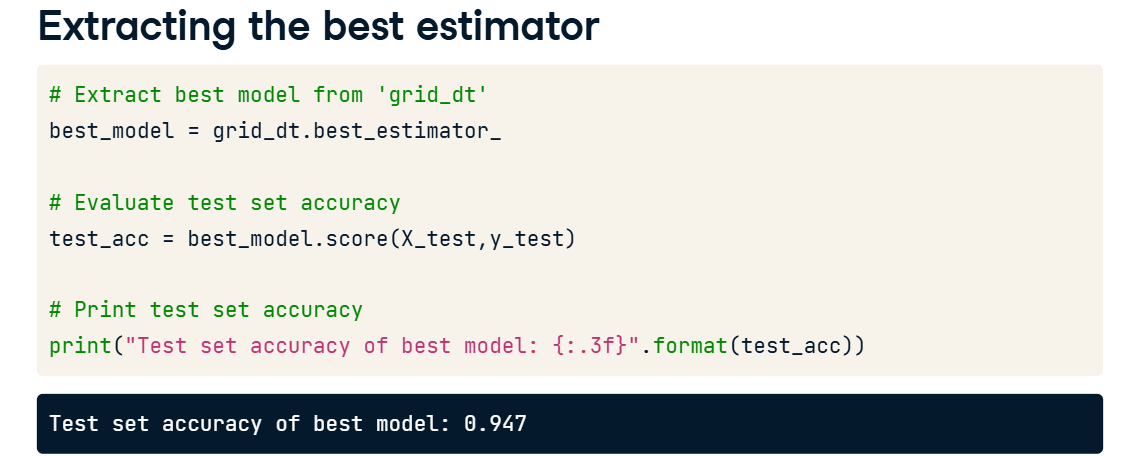


‘max\_features’ is the number of features to account for consideration when looking for a best split.

When it is a float, it is interpreted as a percentage.







**Random Forests Hyperparameters**

* Cart Hyperparameters, number of estimators, bootstrap etc.

