📌 a) Fit classification tree

- **Drzewo decyzyjne** (np. `DecisionTreeClassifier`):
  - Algorytm klasyfikacyjny dzielący dane na podstawie pytań (np. „czy x[3] < 5?") w celu przewidywania klasy.
  - **Kryterium podziału**: np. gini impurity lub entropy — mierzy niejednorodność zbioru.
  - **Splitter**: 'best' (najlepszy podział w węźle) vs. 'random' (losowy podział).
  - **max_depth**: maksymalna głębokość drzewa.
  - **min_samples_split**: minimalna liczba próbek potrzebna do podziału w węźle.

✅ Eksperyment:

- Przetestuj różne ustawienia tych parametrów i zobacz, jak zmienia się dokładność, rozmiar drzewa (liczba węzłów) i przeuczenie.

📌 b) Draw a structure of the tree

- Wykorzystaj `plot_tree()` lub `export_text()` z `sklearn.tree`.
- Wizualizujesz jak drzewo podejmuje decyzje: jakie cechy są najważniejsze, jakie pytania zadano, ile gałęzi ma drzewo.

📌 c) Prune the tree

- **Cost-complexity pruning**:
  - Wybiera najlepsze drzewo balansujące między dokładnością a prostotą (rozmiarem).
  - W `sklearn`: `DecisionTreeClassifier.cost_complexity_pruning_path()` daje listę wartości `ccp_alpha`.
  - Większe `alpha` → drzewo prostsze (mniej gałęzi).
  - Wybierasz `alpha`, które daje najwyższą dokładność na zbiorze testowym.

🔍 Zadanie 2 — **Bagging**

- **Bagging (Bootstrap Aggregating)**:
  - Wielokrotnie losujesz (ze zwracaniem) próbki treningowe.
  - Uczysz model (np. drzewo) na każdej próbce.
  - Wynik to **średnia (regresja) lub głosowanie większościowe (klasyfikacja)**.
- Redukuje wariancję modelu (drzewa mają duże rozrzuty).
- Samodzielna implementacja baggingu polega na:
  1. Losowaniu bootstrapowych próbek.
  2. Trenowaniu wielu modeli.
  3. Zbieraniu predykcji i agregowaniu ich.

🔍 Zadanie 3 — **Porównanie ensemble**

- Porównujesz 3 metody:
  1. **Single tree** — drzewo decyzyjne (jeden model).
  2. **Bagging** — agregacja wielu drzew na bootstrapach (implementujesz sam).
  3. **Random Forest** — bagging + dodatkowo losowanie cech w każdym podziale węzła (gotowy w `sklearn`).
- Każdy model uczysz na danych treningowych i testujesz na danych testowych.
- Mierzysz dokładność — który model działa lepiej?

🔍 Teoretyczne podstawy — kluczowe pojęcia

🌳 Drzewo decyzyjne

- Model klasyfikacji/regresji — dzieli dane na węzły na podstawie cech.
- Łatwo przeucza się na danych (duża wariancja).

🛠️ Hyperparametry drzewa

- **criterion** — funkcja niejednorodności: Gini lub Entropy.
- **splitter** — sposób wyboru podziału (best, random).
- **max_depth** — maksymalna głębokość (przycinanie).
- **min_samples_split** — minimalna liczba próbek do podziału w węźle.
- **ccp_alpha** — parametr przycinania cost-complexity (usuwanie gałęzi).

🔀 Bagging

- Redukuje wariancję modelu, bo trenuje wiele modeli na losowych bootstrapach.
- Poprawia stabilność i często dokładność.

🌲 Random Forest

- Bagging + dodatkowy losowy wybór podzbioru cech przy każdym podziale węzła.
- Jeszcze bardziej zmniejsza korelację między drzewami.

📈 Pruning (Cost-Complexity)

- Ocenia koszt:

$$R_\alpha(T) = R(T) + \alpha \cdot |T|$$

  gdzie:

  - $R(T)$ — błąd klasyfikacji/regresji w drzewie.
  - $|T|$ — liczba węzłów w drzewie.
- Dla każdego `alpha` przycinamy drzewo.
- Wybieramy `alpha`, które daje najlepszy kompromis między dokładnością a rozmiarem drzewa.

📊 Implementacja (skrócony opis)

- `analyze_tree_parameters`: testuje różne parametry drzewa i mierzy dokładność + rozmiar drzewa.
- `visualize_tree_structure`: rysuje drzewo (łatwo zobaczyć, które cechy są ważne).
- `print_tree_rules`: drukuje zasady decyzyjne (interpretacja).
- `MyBaggingClassifier`: implementacja Baggingu (bootstrap, uczenie, głosowanie).
- `compare_ensembles`: porównuje Single Tree, Bagging i Random Forest na jednym zbiorze danych.
- `cost_complexity_pruning`: przycina drzewo i rysuje wykresy dokładności vs. alpha.
