Introduction à l'apprentissage automatique

CSI 4506 - Automne 2026

Marcel Turcotte
Version: août 9, 2026 11h41

Préambule

Citation du jour

TIME100 AI 2025 - The 100 Most Influential People in AI 2025, TIME, 28 août, 2025

Pour la troisième année consécutive, le magazine TIME a publié sa liste des 100 personnalités les plus influentes dans le domaine de l'intelligence artificielle.

Citation du jour (suite)

Yoshua **Bengio**, Université de Montréal, a été à nouveau reconnu par TIME comme l'une des personnes les plus influentes dans le domaine de l'intelligence artificielle.

Encore cette année, Yoshua Bengio de l'Université de Montréal figure parmi les personnes les plus influentes en intelligence artificielle. Bengio, avec Geoffrey Hinton et Yann LeCun, a reçu le prix Turing de l'ACM en 2018 pour leurs contributions pionnières dans le domaine. Le trio est souvent désigné comme les "Pères de l'apprentissage profond".

- Les pères de la révolution de l'apprentissage profond reçoivent le prix A.M. Turing de l'ACM
- Le scientifique en informatique le plus cité a un plan pour rendre l'IA plus digne de confiance, par Harry Booth, TIME, 2025-06-03.

> **Remarque**
>
> Dans l'évolution de l'intelligence, l'apprentissage a été l'un des **premiers jalons à émerger**. C'est aussi l'un des mécanismes les plus **compris** dans l'intelligence naturelle.

Démystifier les mythes

> **(Burkov 2019)**
>
> Let's start by telling the truth: **machines don't learn**. (...) just like **artificial intelligence is not intelligence**, **machine learning is not learning**.

Andriy Burkov, un expert en apprentissage automatique basé à Québec, Canada, est l'auteur de **The Hundred-Page Machine Learning Book**, qui est référencé à la fin de cette présentation. Il est actif sur LinkedIn et publie une newsletter, True Positive Weekly, où il partage chaque semaine des développements significatifs et des perspectives du domaine qui ont attiré son attention.

Fondamentaux de l'apprentissage automatique

Dans ce cours, nous introduirons des concepts essentiels pour comprendre l'apprentissage automatique, y compris les types de problèmes (tâches).

Objectif général :

- **Décrire** les concepts fondamentaux de l'apprentissage automatique

Objectifs d'apprentissage

- **Résumer** les différents types et tâches en apprentissage automatique
- **Discuter** la nécessité des jeux de données d'entraînement et de test

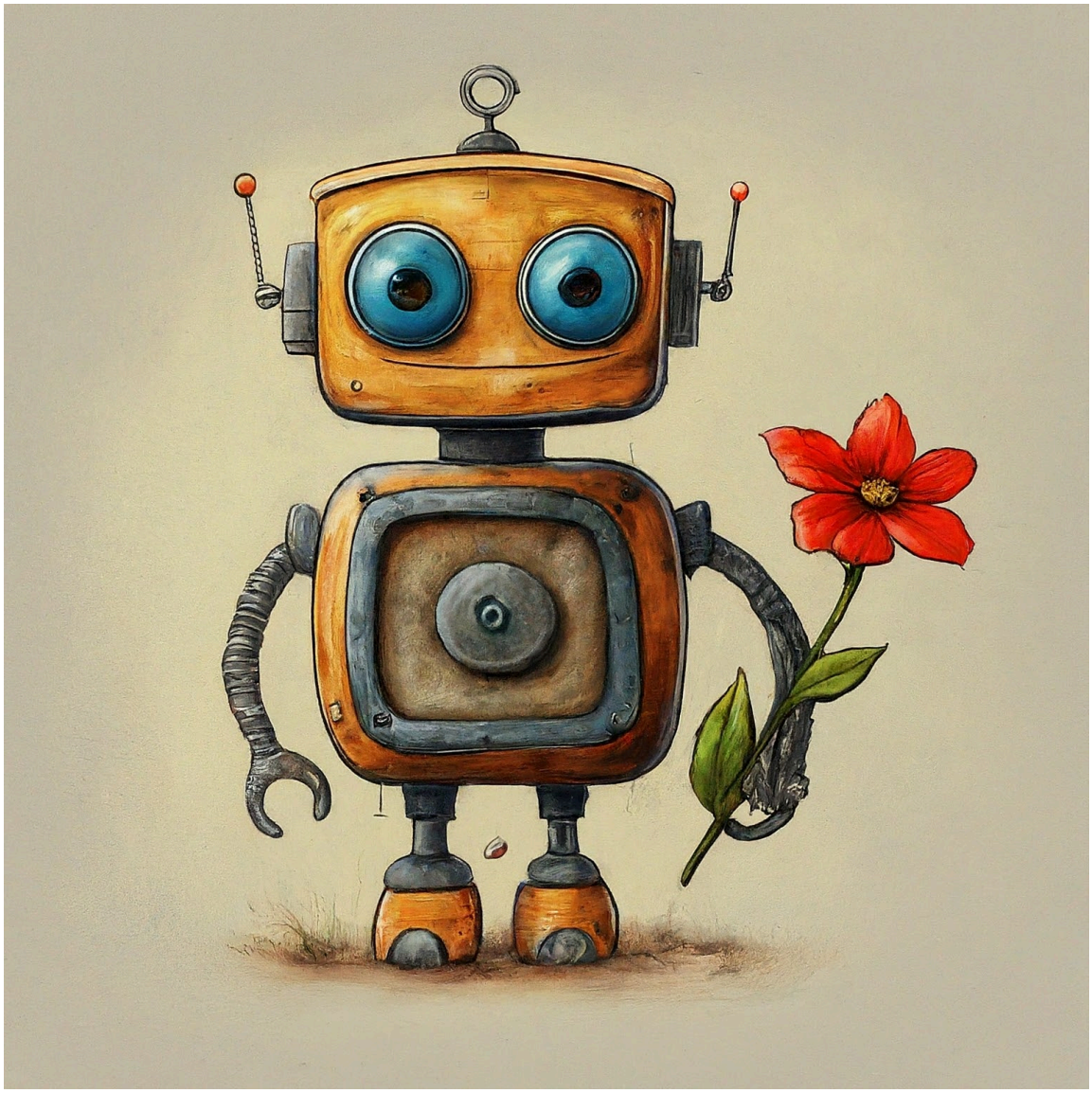
Lectures

- Russell et Norvig (2020), Chapitre 19 : Apprentissage à partir d'exemples.

Introduction

Justification

Pourquoi un programme informatique devrait-il *apprendre* ?



Attribution: Gemini 1.5 Flash, 14 août, 2024, avec l'invite: "In the style of a children's book, create the image of a cute robot holding a red flower."

1. **Adaptabilité et amélioration continue :**

- **Adaptabilité aux environnements dynamiques :** Les programmes qui apprennent peuvent s'adapter aux conditions changeantes, assurant ainsi un fonctionnement efficace dans des environnements dynamiques.
    - *Exemple :* Les voitures autonomes s'ajustant aux changements de trafic et de météo.
- **Amélioration continue :** L'apprentissage permet aux systèmes de rester à jour avec les dernières données et tendances sans intervention humaine.
    - *Exemple :* Les filtres anti-spam évoluant pour contrer les nouvelles techniques de spam.

1. **Performance et efficacité améliorées :**

- **Performance améliorée :** L'apprentissage permet aux programmes d'améliorer leurs performances en se basant sur des expériences passées ou des données.
  - *Exemple :* Les systèmes de recommandation affinant leurs suggestions avec plus de données utilisateurs.
- **Rentabilité :** L'automatisation du processus d'apprentissage réduit le besoin de mises à jour manuelles, entraînant des économies.
  - *Exemple :* Les systèmes de maintenance prédictive minimisant les inspections manuelles.

1. **Résolution de problèmes complexes et découverte de motifs cachés :**

- **Gestion des problèmes complexes :** Les algorithmes d'apprentissage abordent des problèmes trop complexes pour les systèmes statiques basés sur des règles.
  - *Exemple :* La reconnaissance d'images distinguant différents objets.
- **Découverte de motifs cachés :** Les modèles d'apprentissage peuvent découvrir des relations cachées dans les données qui ne sont pas évidentes pour les analystes humains.
  - *Exemple :* L'identification de relations génomiques complexes en bioinformatique.

1. **Personnalisation et évolutivité :**

- **Personnalisation :** L'apprentissage permet aux programmes de fournir des résultats adaptés aux utilisateurs individuels, améliorant ainsi l'expérience utilisateur.
  - *Exemple :* Les assistants virtuels apprenant les préférences des utilisateurs.
- **Évolutivité :** Les algorithmes d'apprentissage gèrent et analysent efficacement de grands ensembles de données, améliorant leur utilité.
  - *Exemple :* Les moteurs de recherche optimisant la pertinence des résultats grâce au machine learning.

1. **Innovation et recherche :**

- **Favoriser l'innovation :** Les algorithmes d'apprentissage peuvent simuler de nouvelles idées, conduisant à des avancées et découvertes.
  - *Exemple :* Les modèles d'apprentissage profond prédisant l'efficacité des médicaments en recherche pharmaceutique.

Définition

> **Mitchell (1997), page 2**

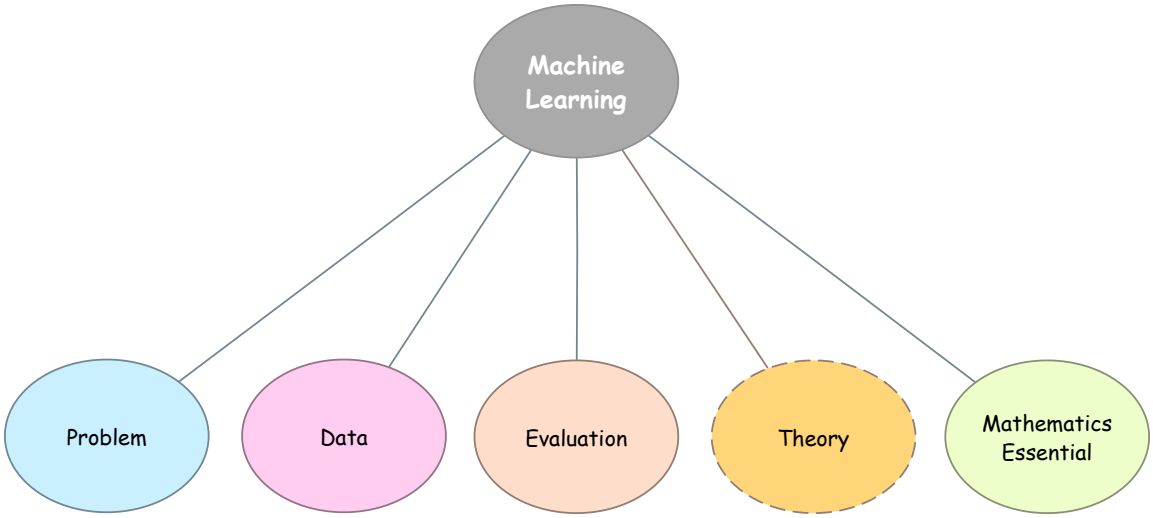
> Un programme informatique **apprend** à partir de **l'expérience** E par rapport à une certaine classe de **tâches** T et une **mesure de performance** P, si sa performance pour les tâches dans T, mesurée par P, s'améliore avec l'expérience E.

Tom M Mitchell. Machine Learning. McGraw-Hill, New York, 1997. (PDF)

Bien que ce livre ait été publié en 1997, il a exercé une influence considérable et reste pertinent. Contrairement à de nombreux autres livres sur l'apprentissage automatique, il est particulièrement captivant.

Cette définition particulière de l'apprentissage est souvent réutilisée.

Concepts



Voir: images/svg/ml_concepts-00.svg

Types de problèmes

Il existe **trois (3)** types distincts de **rétroaction** :

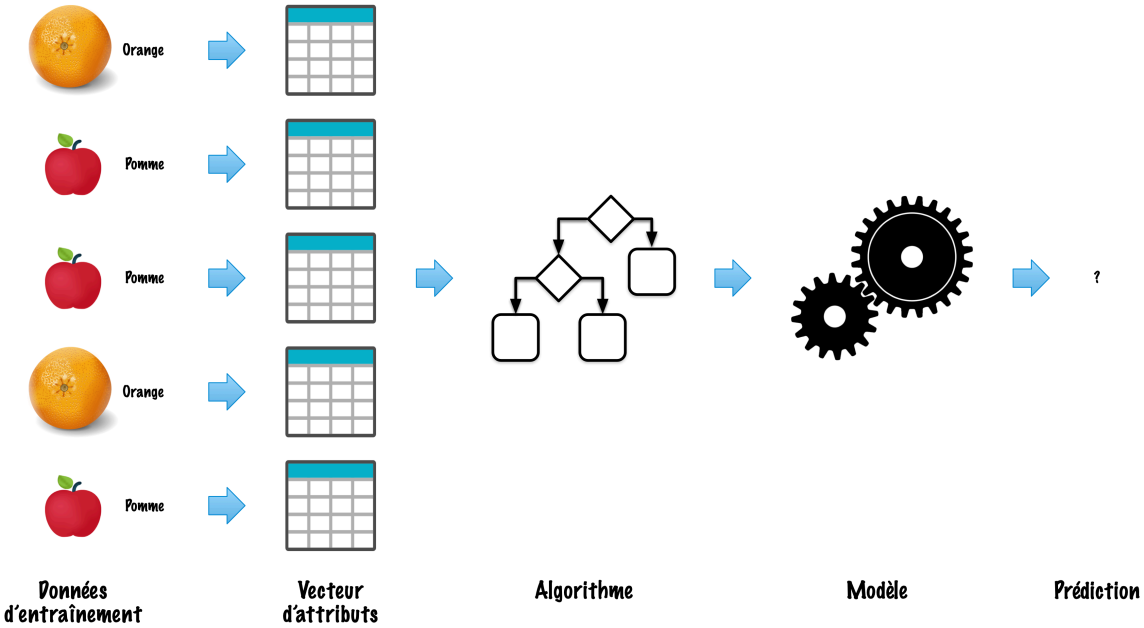
1. **Apprentissage non supervisé :** Aucune rétroaction n'est fournie à l'algorithme.
2. **Apprentissage supervisé :** Chaque exemple est accompagné d'une étiquette.
3. **Apprentissage par renforcement :** L'algorithme reçoit une récompense ou une punition après chaque action.
4. . .

L'apprentissage supervisé est le type d'apprentissage le plus étudié et sans doute le plus intuitif. C'est généralement le premier type d'apprentissage introduit dans les contextes éducatifs.

Deux phases

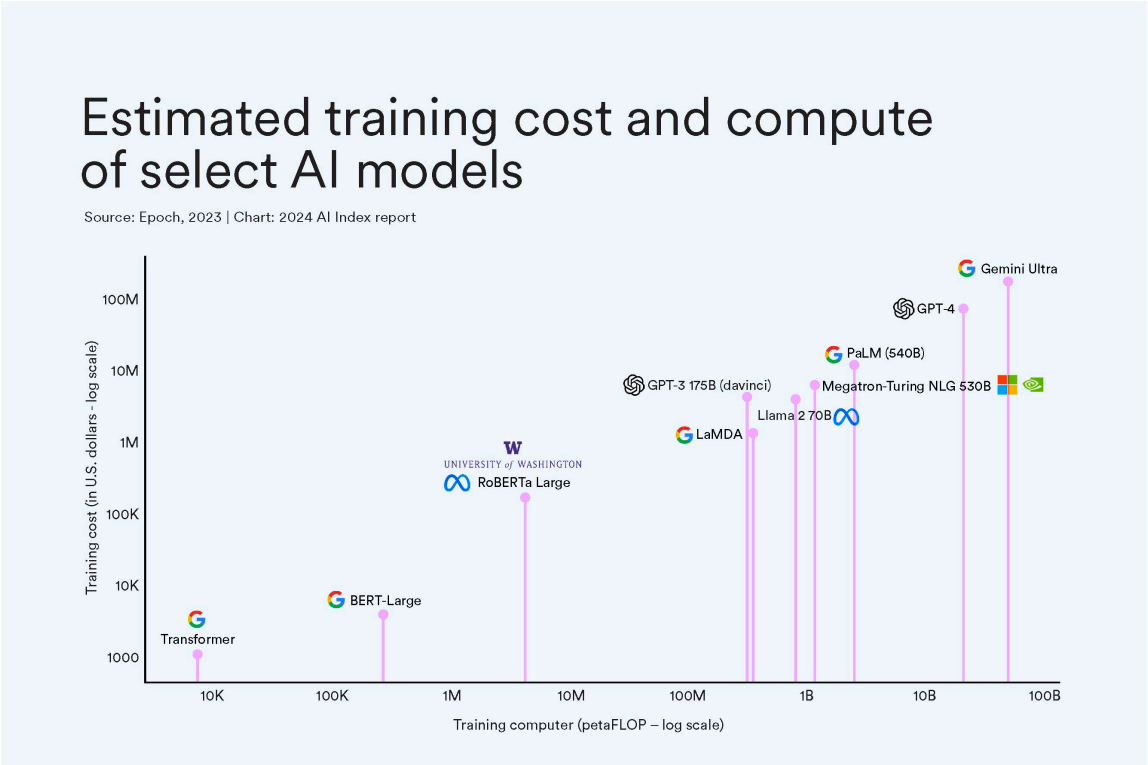
1. **Apprentissage** (construction d'un modèle)
2. **Inférence** (utilisation du modèle)

Apprentissage (construction)



En apprentissage automatique, la phase d'apprentissage est souvent le composant le plus difficile et le plus gourmand en ressources. Cette étape nécessite une attention méticuleuse à la curation des données, ainsi que la sélection et l'entraînement d'un algorithme approprié.

On estime que l'entraînement des modèles de langage de grande envergure contemporains, tels que le GPT d'OpenAI ou le Gemini Ultra de Google, engendre des coûts dépassant 100 millions de dollars américains.

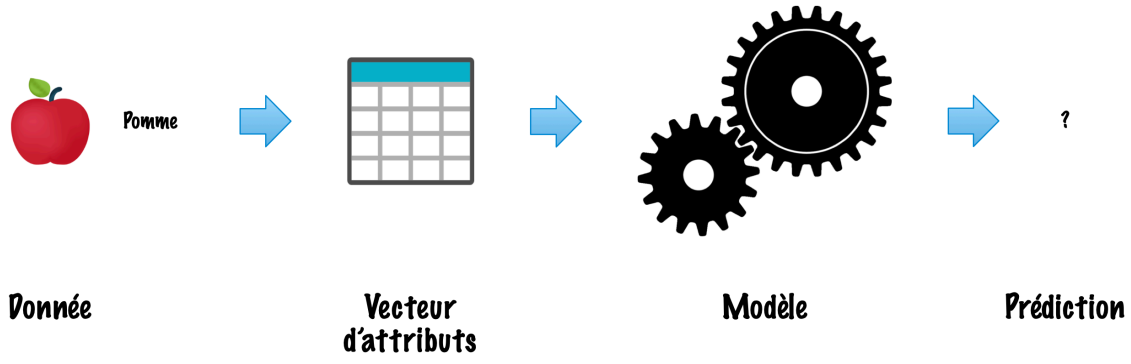


Source: Stanford University Human-Centered Artificial Intelligence 2025 AI Index Report

Le développement de tels modèles nécessite des investissements en infrastructure pouvant atteindre des milliers de milliards de dollars.

- "By the end of 2024, we're aiming to continue to grow our infrastructure build-out that will include **350,000 NVIDIA H100 GPUs** as part of **a portfolio that will feature compute power equivalent to nearly 600,000 H100s**."
  - Building Meta's GenAI Infrastructure, 2024-03-12.
- OpenAI's Sam Altman Expects to Spend 'Trillions' on Infrastructure, Bloomber, 2025-08-15.

Inférence (utilisation du modèle)



L'inférence nécessite généralement moins de ressources informatiques car elle implique l'utilisation d'un modèle pré-entraîné pour prédire le résultat d'un cas individuel.

Néanmoins, l'émergence des modèles de raisonnement en chaîne, souvent appelés modèles de raisonnement, a considérablement augmenté le coût informatique associé à l'inférence.

- OpenAI's o3 Reasoning Models Are Extremely Expensive to Run, by Alexandra Tremayne-Pengelly, Observer, 2025-04-04.
  - "In December, OpenAI's o3 reasoning model became the first A.I. system to pass the test with an 87.5 percent score."
  - "Testing OpenAI's o3 model may cost as much as $30,000 per task."

Carp-e Diem! (exemple)

1. Problème : Vont-ils mordre aujourd'hui ?

Objectif : Développer un modèle prédictif pour évaluer la probabilité de succès d'une journée de pêche, classée en trois catégories : **'Médiocre'**, **'Moyenne'** ou **'Excellente'**.

Attribution: Gemini 1.5 Flash, 10 septembre, 2024, avec l'invite: "In the style of a children's book, generate the image of a fish with a farmer's hat."

L'apprentissage supervisé consiste à entraîner un modèle sur des données étiquetées afin qu'il puisse **faire des prédictions sur de nouvelles données non vues**.

La tâche spécifique est la **classification**.

"Médiocre", "Moyenne" et "Excellente" sont des **classes** (pour la variable cible).

2. Attributs (caractéristiques)

Différentes sources, y compris **The Old Farmer's Almanac**, suggèrent que la **phase de la lune** est un prédicteur fiable du succès de la pêche.

- **Phase de la lune (catégorielle)** : 'Nouvelle lune', 'Premier quartier', 'Pleine lune' et 'Dernier quartier'.
- **Prévisions météorologiques (catégorielles)** : 'Pluvieux', 'Nuageux' et 'Ensoleillé'.
- **Température extérieure (numérique)** : La température en Celsius.
- **Température de l'eau (numérique)** : La température de l'eau du lac ou de la rivière.

Évidemment, les applications réelles auront beaucoup plus d'attributs.

3. Données d'entraînement

Exemple	Phase de la lune	Prévisions météorologiques	Température extérieure (°C)	Température de l'eau (°C)	Probabilité d'une journée de pêche
1	Pleine lune	Ensoleillé	25	22	Excellente
2	Nouvelle lune	Nuageux	18	19	Moyenne
3	Premier quartier	Pluvieux	15	17	Médiocre
4	Dernier quartier	Ensoleillé	30	24	Excellente
5	Pleine lune	Nuageux	20	20	Moyenne
6	Nouvelle lune	Pluvieux	22	21	Médiocre

Ceci est identifié comme un problème d'**apprentissage supervisé** car la valeur de la variable cible est connue pour chaque instance d'entraînement. De plus, comme les

valeurs de la variable cible sont catégorielles, ce problème est classé comme une **tâche de classification**.

Dans ce contexte, l'ensemble d'entraînement se compose de 6 exemples. Il est important de noter que les ensembles de données réels contiennent généralement un nombre beaucoup plus grand d'exemples pour garantir un entraînement et une validation robustes du modèle.

Le choix des attributs et la qualité des données sont essentiels pour la performance du modèle. Un attribut tel que la couleur de vos chaussettes n'a probablement pas un grand impact sur les prédictions.

Par conséquent, un projet d'apprentissage automatique commence généralement par une phase exploratoire, qui consiste à analyser les données, examiner les distributions et identifier les corrélations.

3. Données

Phase de la lune	Prévisions météorologiques	Température extérieure (°C)	Température de l'eau (°C)
Pleine lune	Ensoleillé	25	22
Nouvelle lune	Nuageux	18	19
Premier quartier	Pluvieux	15	17
Dernier quartier	Ensoleillé	30	24
Pleine lune	Nuageux	20	20
Nouvelle lune	Pluvieux	22	21

Les données sont souvent présentées sous forme tabulaire (matrice), où chaque ligne représente un **vecteur d'attributs** (*feature vector*), généralement noté x_i, correspondant au i-ème exemple dans l'ensemble d'entraînement.

3. Étiquettes

Probabilité d'une journée de pêche
Excellente
Moyenne
Médiocre
Excellente

Probabilité d'une journée de pêche
Moyenne
Médiocre

Les **étiquettes** sont généralement représentées comme un vecteur colonne, avec y_i désignant l'étiquette pour le i-ème exemple.

4. Entraînement du modèle

L'**entraînement du modèle** consiste à utiliser des données étiquetées pour enseigner à un algorithme d'apprentissage automatique comment faire des prédictions. Ce processus **ajuste les paramètres du modèle** pour **minimiser l'erreur entre les résultats prévus et les résultats réels**.

4. Entraînement du modèle (suite)

- **Journée de pêche excellente :**
  - Phase de la lune : **Pleine lune** ou **Nouvelle lune**
  - Prévisions météorologiques : **Ensoleillé**
  - Température extérieure : 20°C à 30°C
  - Température de l'eau : 20°C à 25°C

. . .

- **Journée de pêche médiocre :**
  - Phase de la lune : **Premier quartier** ou **Dernier quartier**
  - Prévisions météorologiques : **Pluvieux**
  - Température extérieure : < 20°C ou > 30°C
  - Température de l'eau : < 20°C ou > 25°C

5. Prédiction

Étant donné de nouvelles données **non vues**, prédisez si la journée sera réussie.

. . .

- Phase de la lune : **Nouvelle lune**
- Prévisions météorologiques : **Ensoleillé**
- Température extérieure : **24°C**
- Température de l'eau : **21°C**

Cycle de vie

1. Collecte et préparation des données
2. Ingénierie des attributs
3. Entraînement
4. Évaluation du modèle
5. Déploiement du modèle
6. Surveillance et maintenance

- La **collecte des données** et leur **préparation** sont des processus critiques et laborieux.
- Il doit y avoir une quantité **suffisante** de données.
- Les données doivent être de haute **qualité** ; par exemple, elles ne doivent pas être excessivement bruyantes.
- Il doit y avoir peu de **valeurs manquantes**.
- Plus important encore, les données doivent être **représentatives**. Nous nous attendons à ce que les nouvelles données soient générées par le même processus et aient la même distribution.
- L'importance de cela ne peut être surestimée.
- Pensez à un logiciel de classification d'images qui n'a pas été entraîné sur un échantillon diversifié en termes d'ethnicité, de genre, de taille corporelle ou de statut social.
- Pensez aux applications médicales et aux conséquences de jeux de données qui ne sont pas suffisamment diversifiés.
- L'**ingénierie des attributs** est le processus de sélection, de transformation et de création de variables d'entrée (attributs) pour améliorer la performance d'un modèle d'apprentissage automatique. Cela implique des techniques telles que la mise à l'échelle, le codage des variables catégorielles et la génération de nouvelles attributs à partir de celles existantes pour améliorer la capacité du modèle à apprendre des schémas et à faire des prédictions précises.
- L'ingénierie des attributs était autrefois une étape laborieuse. L'un des principaux avantages de l'apprentissage profond est qu'il peut apprendre automatiquement des attributs.
- L'**évaluation du modèle** est le processus d'évaluation des performances d'un modèle d'apprentissage automatique en utilisant des métriques spécifiques, telles que l'exactitude, la précision, le rappel, le F1-score ou l'AUC-ROC. Cela implique généralement de tester le modèle sur un ensemble de validation ou de test séparé pour s'assurer qu'il se généralise bien aux données non vues et qu'il répond aux critères souhaités d'exactitude et de fiabilité.
- Le **déploiement du modèle** : Une application est construite en utilisant le modèle. Il est important de noter que la plupart du temps, les paramètres du système sont figés lors du déploiement. Premièrement, l'entraînement est coûteux et un nouvel entraînement est souvent inabordable. De plus, un nouvel entraînement peut entraîner une perte d'informations précédemment apprises, ce qui dégrade les performances sur les exemples déjà vus.

- La **surveillance et la maintenance** : Les performances du modèle doivent être continuellement surveillées. On observe souvent une **dérive conceptuelle**, nécessitant la réentraînement du système. La détection de spam est un bon exemple. Une fois le système déployé, les spammeurs s'adaptent et trouvent des moyens de contourner les mécanismes de détection de spam mis en place.

Définitions formelles

Apprentissage supervisé (notation)

L'**ensemble de données** ("expérience") est une collection d'exemples étiquetés.

- $\{(x_i, y_i)\}_{i=1}^N$
- Chaque x_i est un vecteur d'**attributs** (*feature*) avec D dimensions.
- $x_i^{(j)}$ est la valeur de l'attribut j de l'exemple i, pour $j \in 1 \ldots D$ et $i \in 1 \ldots N$.
- L'**étiquette** y_i est soit une **classe**, prise d'une liste finie de classes, $\{1, 2, \ldots, C\}$, soit un **nombre réel**, soit un **objet complexe** (arbre, graphe, etc.).

. . .

Problème : Étant donné l'ensemble de données en entrée, créer un **modèle** qui peut être utilisé pour prédire la valeur de y pour un x non vu.

Cette notation suit celle du **The Hundred-Page Machine Learning Book** par Andriy Burkov.

Apprentissage supervisé (suite)

- Lorsque l'**étiquette** y_i est une **classe**, prise d'une liste finie de classes, $\{1, 2, \ldots, C\}$, nous appelons la tâche une tâche de **classification**.

- Lorsque l'**étiquette** y_i est un **nombre réel**, nous appelons la tâche une tâche de **régression**.

Anil Ananthaswamy dans son livre Why Machines Learn: The Elegant Math Behind Modern AI propose un example inspiré de la pandémie de COVID-19.

Dans ce scénario avancé, les hôpitaux du monde entier compilent systématiquement les données des patients depuis les premiers stades de la pandémie de COVID-19. Chaque patient est caractérisé par six variables clés : x_1 représentant l'âge, x_2 indiquant l'indice de masse corporelle, x_3 dénotant la difficulté à respirer (encodée comme 1 pour oui et 0 pour non), x_4 signifiant la présence de fièvre (oui/non), x_5 pour le statut diabétique (oui/non), et x_6 décrivant les résultats du scanner thoracique (0 pour clair, 1 pour

infection légère, 2 pour infection sévère). Ces variables forment collectivement un vecteur à six dimensions pour chaque patient.

Les cliniciens ont observé que les patients suivent généralement l'une des deux trajectoires suivantes : soit ils se stabilisent et sont libérés environ trois jours après leur admission, soit leur état se détériore, nécessitant un soutien ventilatoire. Par conséquent, chaque patient se voit attribuer une variable de résultat, y, où $y = -1$ indique qu'il n'y a pas besoin de soutien ventilatoire après trois jours, et $y = 1$ indique la nécessité de soutien ventilatoire après trois jours.

Pouvez-vous penser à des exemples de tâches de régression ?

Voici plusieurs tâches de régression avec leurs applications dans le monde réel :

1. **Prévision du prix des maisons** :
   - **Application** : Estimer la valeur marchande des propriétés résidentielles en fonction d'attributs telles que la localisation, la taille, le nombre de chambres, l'âge et les commodités.
2. **Prévision du marché boursier** :
   - **Application** : Prédire les prix futurs des actions ou des indices en fonction des données historiques, des indicateurs financiers et des variables économiques.
3. **Prévision météorologique** :
   - **Application** : Estimer les températures futures, les précipitations et d'autres conditions météorologiques à l'aide de données météorologiques historiques et de variables atmosphériques.
4. **Prévision des ventes** :
   - **Application** : Prédire les volumes de ventes futurs de produits ou de services en analysant les données de ventes passées, les tendances du marché et les modèles saisonniers.
5. **Prévision de la consommation d'énergie** :
   - **Application** : Prévoir la consommation d'énergie future des ménages, des industries ou des villes en fonction des données historiques de consommation, des conditions météorologiques et des facteurs économiques.
6. **Estimation des coûts médicaux** :
   - **Application** : Prédire les coûts des soins de santé pour les patients en fonction de leur historique médical, de leurs informations démographiques et de leurs plans de traitement.
7. **Prévision du trafic routier** :
   - **Application** : Estimer les volumes de trafic futurs et les niveaux de congestion sur les routes et les autoroutes en utilisant des données historiques de trafic et des entrées de capteurs en temps réel.
8. **Estimation de la valeur à vie du client (CLV)** :
   - **Application** : Prédire le revenu total qu'une entreprise peut attendre d'un client sur la durée de leur relation, en fonction des comportements d'achat et des

données démographiques.

9. **Prévision des indicateurs économiques** :
    - **Application** : Prédire les principaux indicateurs économiques tels que la croissance du PIB, les taux de chômage et l'inflation en utilisant des données économiques historiques et des tendances du marché.
10. **Prévision de la demande** :
    - **Application** : Estimer la demande future de produits ou de services dans diverses industries comme le commerce de détail, la fabrication et la logistique pour optimiser la gestion des stocks et de la chaîne d'approvisionnement.
11. **Évaluation immobilière** :
    - **Application** : Évaluer la valeur de marché des propriétés commerciales telles que les immeubles de bureaux, les centres commerciaux et les espaces industriels en fonction de la localisation, de la taille et des conditions du marché.
12. **Évaluation des risques d'assurance** :
    - **Application** : Prédire le risque associé à l'assurance des individus ou des propriétés, ce qui aide à déterminer les taux de prime, en se basant sur les données historiques des réclamations et les facteurs démographiques.
13. **Prédiction du taux de clics (CTR) des annonces** :
    - **Application** : Estimer la probabilité qu'un utilisateur clique sur une annonce en ligne en fonction du comportement de l'utilisateur, des caractéristiques de l'annonce et des facteurs contextuels.
14. **Prédiction du défaut de paiement de prêt** :
    - **Application** : Prédire la probabilité qu'un emprunteur fasse défaut sur un prêt en fonction de son historique de crédit, de ses revenus, du montant du prêt et d'autres indicateurs financiers.

Voici quelques applications de tâches de régression que l'on trouve généralement dans les applications mobiles :

1. **Prédiction de l'autonomie de la batterie** :
    - **Application** : Estimer l'autonomie restante de la batterie en fonction des habitudes d'utilisation, des applications en cours d'exécution et des paramètres de l'appareil.
2. **Suivi de la santé et de la forme physique** :
    - **Application** : Prédire la consommation de calories, la fréquence cardiaque ou la qualité du sommeil en fonction de l'activité de l'utilisateur, des biométries et des données de santé historiques.
3. **Gestion des finances personnelles** :
    - **Application** : Prévoir les dépenses ou les économies futures en fonction des habitudes de dépenses, des modèles de revenus et des objectifs budgétaires.
4. **Prévision météorologique** :

- **Application** : Fournir des prévisions météorologiques personnalisées en fonction de la localisation actuelle et des données météorologiques historiques.

5. **Estimation du temps de trajet et de la circulation** :
   - **Application** : Prédire les temps de trajet et suggérer des itinéraires optimaux en fonction des données historiques de trafic, des conditions en temps réel et du comportement de l'utilisateur.
6. **Amélioration de la qualité des images et des vidéos** :
   - **Application** : Ajuster les paramètres de qualité des images ou des vidéos (par exemple, la luminosité, le contraste) en fonction des conditions d'éclairage et des préférences de l'utilisateur.
7. **Atteinte des objectifs de remise en forme** :
   - **Application** : Estimer le temps nécessaire pour atteindre des objectifs de remise en forme tels que la perte de poids ou le gain musculaire en fonction de l'activité de l'utilisateur et des apports alimentaires.
8. **Optimisation des performances de l'appareil mobile** :
   - **Application** : Prédire les paramètres optimaux pour les performances de l'appareil et l'autonomie de la batterie en fonction des habitudes d'utilisation et de l'activité des applications.

Ces applications exploitent les tâches de régression pour fournir des services personnalisés, efficaces et contextuels qui améliorent l'expérience utilisateur sur les appareils mobiles.

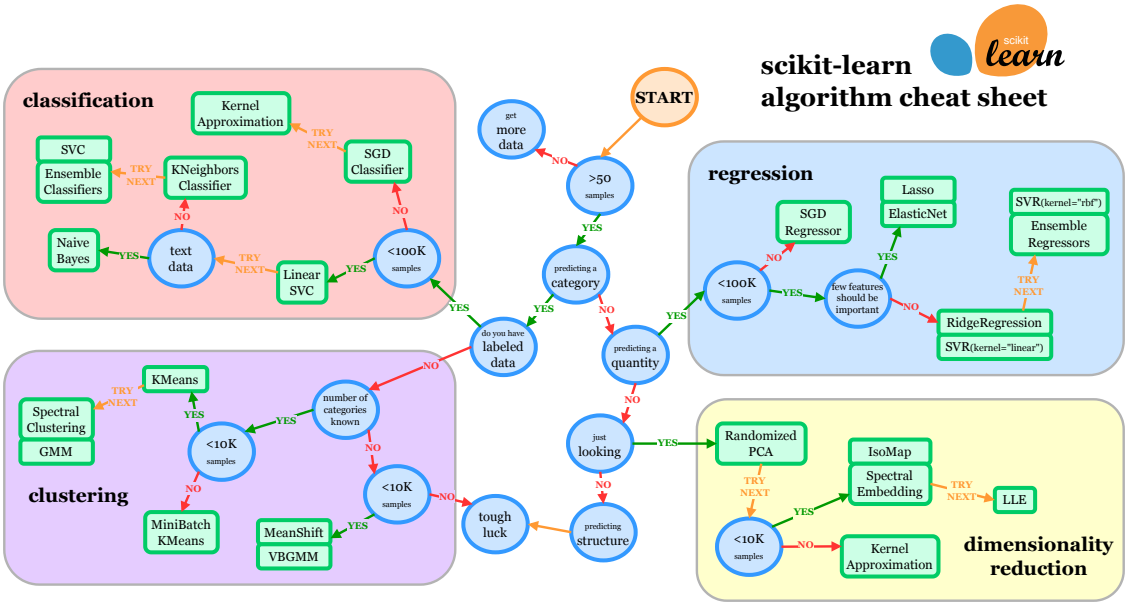
Exemple avec du code

`Scikit-learn`

> **scikit-learn.org**
>
> `Scikit-learn` est une bibliothèque open source (ou à code source ouvert) d'apprentissage automatique qui prend en charge l'apprentissage **supervisé** et **non supervisé**. Elle offre également divers outils pour **l'entraînement des modèles**, **le prétraitement des données**, **la sélection de modèle**, **l'évaluation de modèle**, et de nombreuses autres utilités.
>
> `Scikit-learn` fournit **des dizaines d'algorithmes et de modèles d'apprentissage automatique intégrés**, appelés estimateurs.
>
> Construite sur NumPy, SciPy et matplotlib.

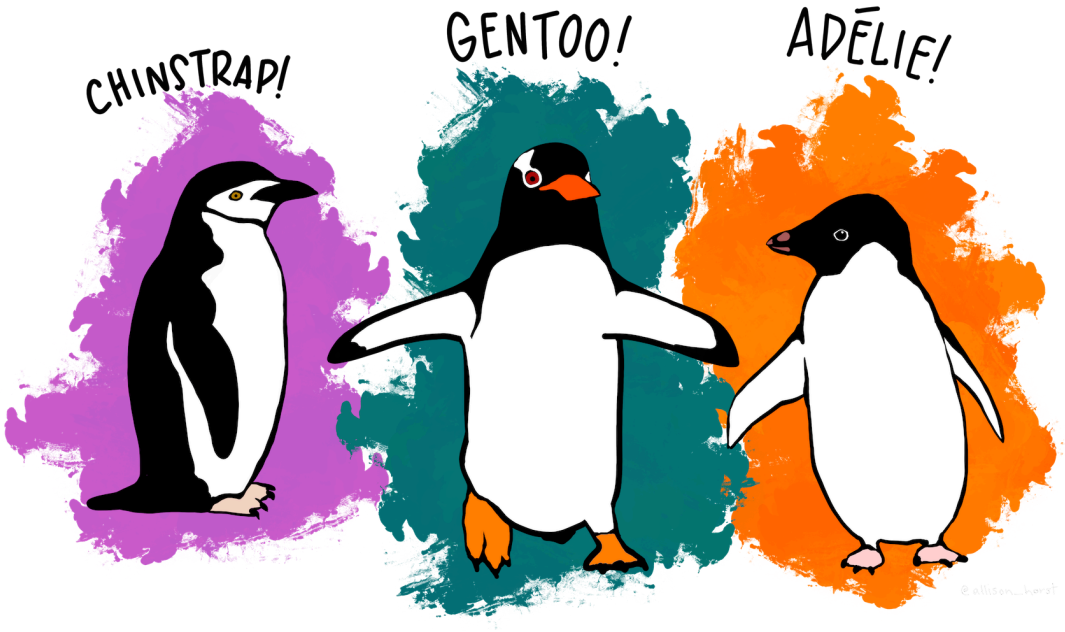
Nous utiliserons `Scikit-learn` pour notre prochain exemple, mais également au cours des semaines à venir.

`Scikit-learn`



Attribution: Choosing the right estimator

Exemple : Manchots de Palmer



Le jeu de données des manchots de Palmer par Allison Horst, Alison Hill, et Kristen Gorman. **Illustration** par @allison_horst

> Le jeu de données des manchots de Palmer par Allison Horst, Alison Hill, et Kristen Gorman a été rendu public pour la première fois sous forme de

> package R. L'objectif du jeu de données des manchots de Palmer est de remplacer le jeu de données Iris, qui est très surutilisé pour l'exploration et la visualisation des données.

> En utilisant ce package python, vous pouvez facilement charger les manchots de Palmer dans votre environnement python.

Initialement, nous examinerons l'ensemble de l'exemple d'un point de vue général pour fournir une vue d'ensemble claire des étapes impliquées. Par la suite, nous reviendrons sur cet exemple plus en détail.

En cas de bibliothèque manquante

In [2]:
```
try:
  from palmerpenguins import load_penguins
except:
  ! pip install palmerpenguins
  from palmerpenguins import load_penguins
```

Si vous exécutez ce Jupyter Notebook dans Google Colab ou tout autre environnement où la bibliothèque n'est pas préinstallée, procédez à l'installation.

Exemple : Chargement des données

In [3]:
```
# Il est d'usage d'utiliser X et y pour les données et les étiquettes

X, y = load_penguins(return_X_y = True)
```

Exemple : DecisionTree

In [4]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier(random_state=42)
```

Il existe des dizaines de classificateurs, y compris ceux-ci : **arbres de décision**, **machines à vecteurs de support**, **k-plus proches voisins**, **régression logistique**, et **réseaux de neurones**.

Exemple : Entraînement

In [5]:
```
# Entraînement

clf = clf.fit(X, y)
```

Tous les classificateurs héritent de `sklearn.base.BaseEstimator` et `sklearn.base.ClassifierMixin`. En conséquence, tous les classificateurs implémentent `fit`, `predict`, et `score`.

Le `DecisionTreeClassifier` dans scikit-learn construit un arbre de décision en scindant récursivement le jeu de données en sous-ensembles basés sur l'attribut qui entraîne le gain d'information le plus élevé (par exemple, impureté de Gini, entropie). Voici une description concise du processus :

1. **Initialisation** : L'algorithme commence avec l'ensemble du jeu de données comme nœud racine.

2. **Critères de Division** : Pour chaque nœud, il évalue toutes les divisions possibles sur tous les attributs pour trouver celle qui sépare le mieux les classes. Cela se fait généralement en minimisant un critère tel que l'impureté de Gini ou l'entropie.

3. **Division Récursive** : Le jeu de données est divisé en sous-ensembles basés sur l'attribut et le seuil sélectionnés, créant des nœuds enfants. Ce processus est répété de manière récursive pour chaque nœud enfant.

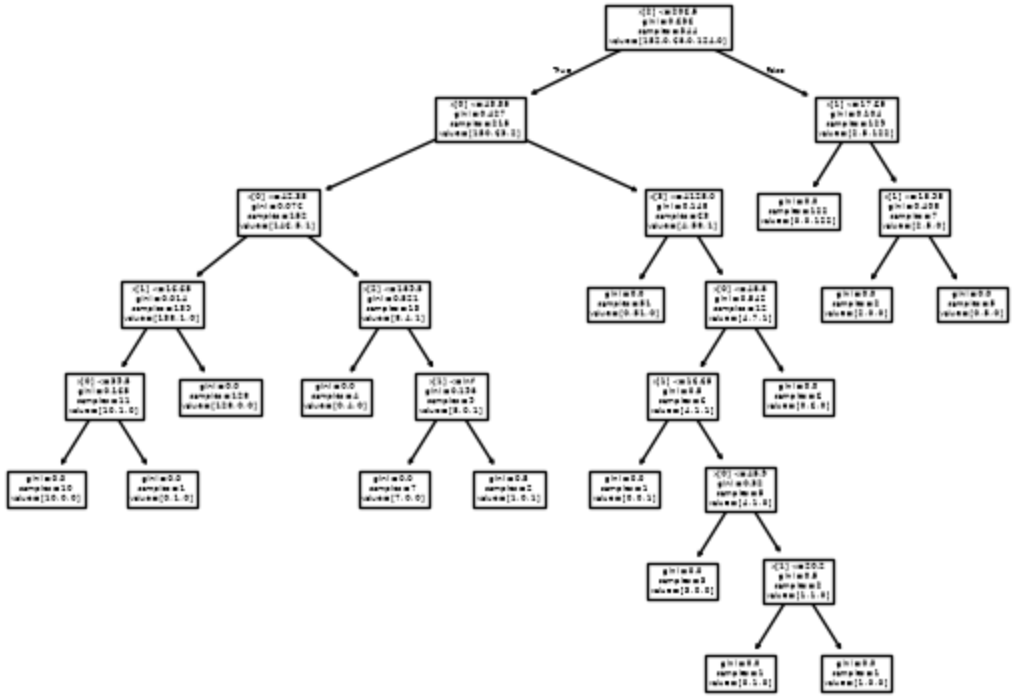
4. **Conditions d'Arrêt** : La division s'arrête lorsqu'un critère prédéfini est atteint, tel qu'une profondeur maximale de l'arbre, un nombre minimal d'échantillons par feuille, ou si une division supplémentaire n'améliore pas significativement le gain d'information.

5. **Nœuds Terminaux** : Une fois la division terminée, chaque nœud terminal se voit attribuer une étiquette de classe basée sur la classe majoritaire des échantillons dans ce nœud.

L'arbre résultant peut ensuite être utilisé pour classifier de nouveaux échantillons en parcourant de la racine à un nœud terminal, en suivant les règles de décision définies à chaque nœud.

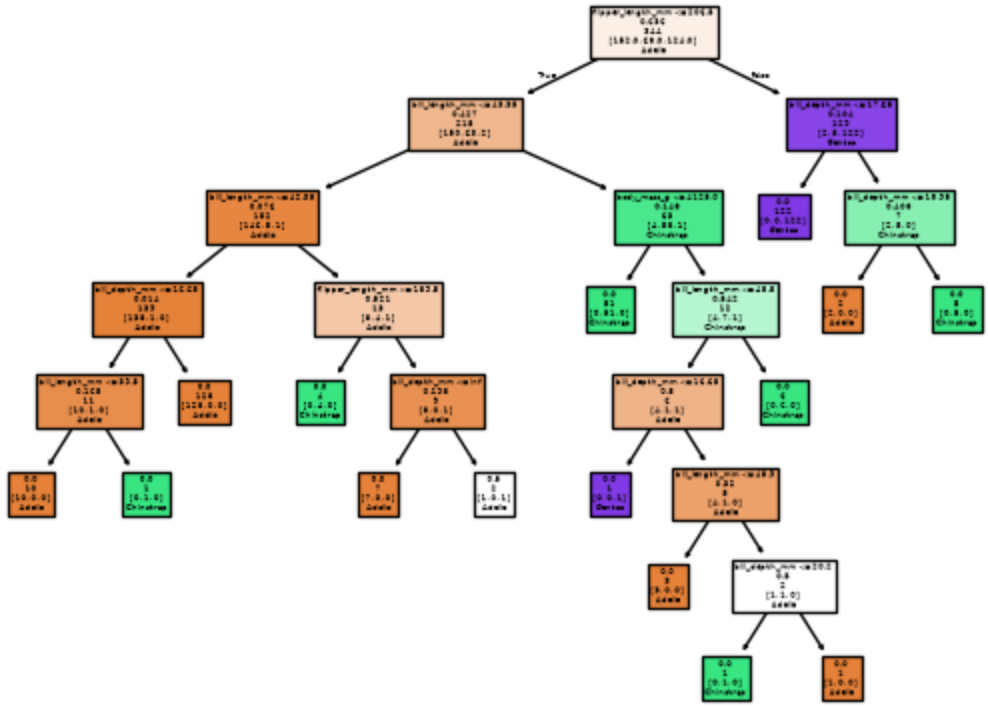
Exemple : Visualisation (1/2)

In [6]:
```
import matplotlib.pyplot as plt

tree.plot_tree(clf)
plt.show()
```



Exemple : Visualisation (2/2)

In [7]:
```
target_names = ['Adelie','Chinstrap','Gentoo']

tree.plot_tree(clf,
               feature_names = X.columns,
               class_names = target_names,
               label = 'none',
               filled = True)
plt.show()
```



Exemple : Prédiction

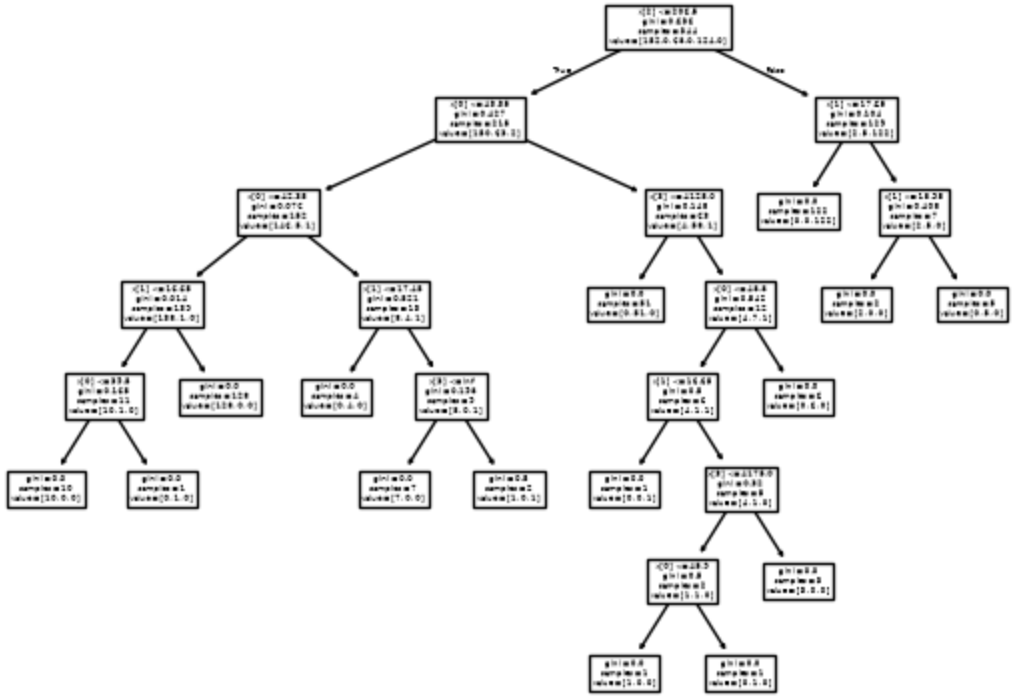
In [8]:
```
import pandas as pd

# Création de 2 exemples de test

columns_names = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'bo
X_test = pd.DataFrame([[34.2, 17.9, 186.8, 2945.0], [51.0, 15.2, 223.7, 5560

# Prédiction

y_test = clf.predict(X_test)

# Affichage des étiquettes prédites pour nos deux exemples

print(y_test)
```

```
['Adelie' 'Gentoo']
```

Exemple : Complet

In [9]:
```
X, y = load_penguins(return_X_y = True)
clf = tree.DecisionTreeClassifier(random_state=123)
clf = clf.fit(X, y)
tree.plot_tree(clf)
X_test = pd.DataFrame([[34.2, 17.9, 186.8, 2945.0], [51.0, 15.2, 223.7, 5560
print(clf.predict(X_test))
```

```
['Adelie' 'Gentoo']
```



Exemple : Performance

In [10]:
```
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=target_names)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 1.00
Rapport de classification :
              precision    recall  f1-score   support

      Adelie       0.99      1.00      1.00       152
   Chinstrap       1.00      1.00      1.00        68
      Gentoo       1.00      0.99      1.00       124

    accuracy                           1.00       344
   macro avg       1.00      1.00      1.00       344
weighted avg       1.00      1.00      1.00       344
```

Exemple : Discussion

Nous avons démontré un exemple complet :

- Chargement des données
- Sélection d'un classificateur
- Entraînement du modèle
- Visualisation du modèle
- Faire une prédiction

Cependant, plusieurs simplifications ont été faites tout au long de ce processus.

Exemple : Attendez une minute !

In [11]:
```
from sklearn.metrics import classification_report, accuracy_score

# Faire des prédictions

y_pred = clf.predict(X)

# Évaluer le modèle

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=target_names)

print(f'Précision : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

> **Important**
>
> Cet exemple est trompeur, voire erroné !

La performance de ce classificateur semble parfaite à première vue. Cependant, l'est-elle vraiment ?

Exemple : Exploration

In [12]:
```
penguins = load_penguins()
```

...

In [13]:
```
type(penguins)
```

```
pandas.DataFrame
```

...

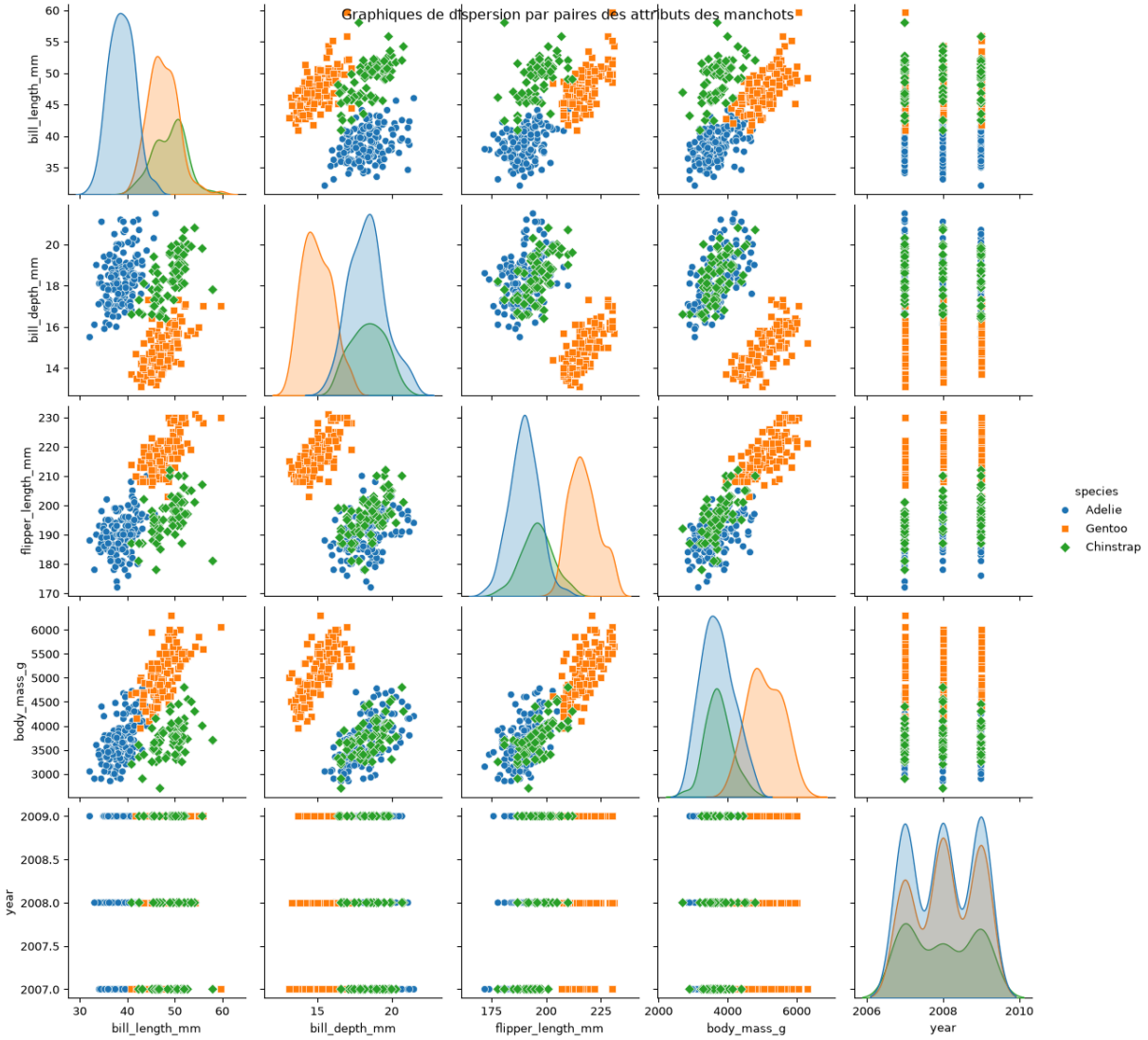
In [14]:
```
penguins.head()
```

Exemple : Exploration

In [15]:
```
penguins.describe()
```

Exemple : Utilisation de `Seaborn`

In [16]:
```
import seaborn as sns

# Pairplot utilisant seaborn

sns.pairplot(penguins, hue='species', markers=["o", "s", "D"])
plt.suptitle("Graphiques de dispersion par paires des attributs des manchots
plt.show()
```



Quelles informations peuvent être tirées de l'examen des graphiques ?

L'image présente tous les graphiques de dispersion par paires pour les attributs du jeu de données iris, avec les diagonales affichant des histogrammes pour chaque attribut

individuel. Chaque point représente un exemple (x), et les couleurs indiquent les étiquettes correspondantes (y).

Considérons d'abord les éléments diagonaux :

- Est-il possible de classer les exemples en utilisant un seul attribut ?
- Nous observons que chaque attribut seul ne peut pas distinguer entre les classes.
- Cependant, dans **bill_depth** vs **body_mass** ou **flipper_length**, nous pouvons différencier **Gentoo** des deux autres espèces, bien qu'elles ne séparent pas efficacement **Adélie** et **Chinstrap**.

La classe **Gentoo** forme fréquemment un groupe distinct.

Exemple : Ensemble d'entraînement et de test

In [17]:
```
from sklearn.model_selection import train_test_split

# Diviser le jeu de données en ensembles d'entraînement et de test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

Notre classificateur initial, un arbre de décision, a été construit en utilisant l'ensemble du jeu de données. Bien qu'il fournisse le meilleur ajustement pour les données, nous ne pouvons pas déterminer sa performance sans évaluation supplémentaire.

Nous aurons beaucoup plus à dire sur les tests dans les semaines à venir.

Pratique : Expérimentez avec différentes valeurs pour `random_state`. Que constatez-vous ? Pourquoi pensez-vous que cela se produit ? Par exemple, réglez `random_state` à `42`. Pourquoi est-il important de définir la valeur de `random_state` ?

Exemple : Création d'un nouveau classificateur

In [18]:
```
clf = tree.DecisionTreeClassifier()
```

Exemple : Entraînement du nouveau classificateur

In [19]:
```
clf.fit(X_train, y_train)
```

Exemple : Visualisation de l'arbre

In [20]:
```
tree.plot_tree(clf,
               feature_names = X.columns,
               class_names = target_names,
               label = 'none',
```

```
                 filled = True)
plt.show()
```

Exemple : Faire des prédictions

In [21]:
```
# Faire des prédictions
y_pred = clf.predict(X_test)
```

Exemple : Mesurer la performance

In [22]:
```
# Évaluer le modèle
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=target_names)

print(f'Exactitude : {accuracy:.2f}')
print('Rapport de classification :')
print(report)
```

```
Exactitude : 0.94
Rapport de classification :
              precision    recall  f1-score   support

      Adelie       0.96      0.90      0.93        30
   Chinstrap       0.88      1.00      0.94        15
      Gentoo       0.96      0.96      0.96        24

    accuracy                           0.94        69
   macro avg       0.93      0.95      0.94        69
weighted avg       0.94      0.94      0.94        69
```

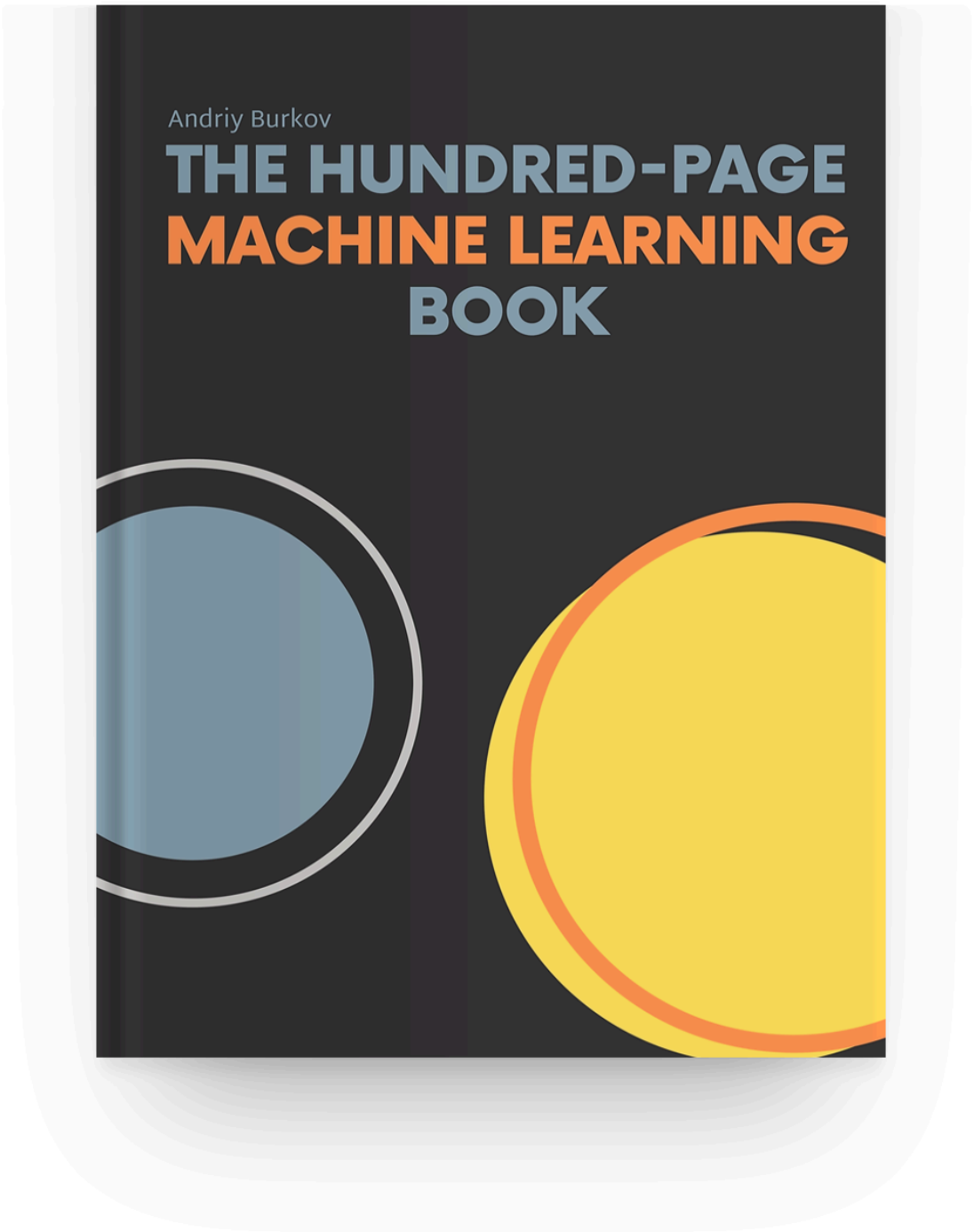
Voici une discussion sur la persistance des modèles pour les modèles scikit-learn.

Résumé

- Nous avons introduit la terminologie pertinente.
- Nous avons examiné un exemple hypothétique.
- Ensuite, nous explorerons un exemple complet en utilisant scikit-learn.
- Nous avons effectué une exploration détaillée de nos données.
- Enfin, nous avons reconnu la nécessité d'un ensemble de test indépendant pour mesurer précisément la performance.

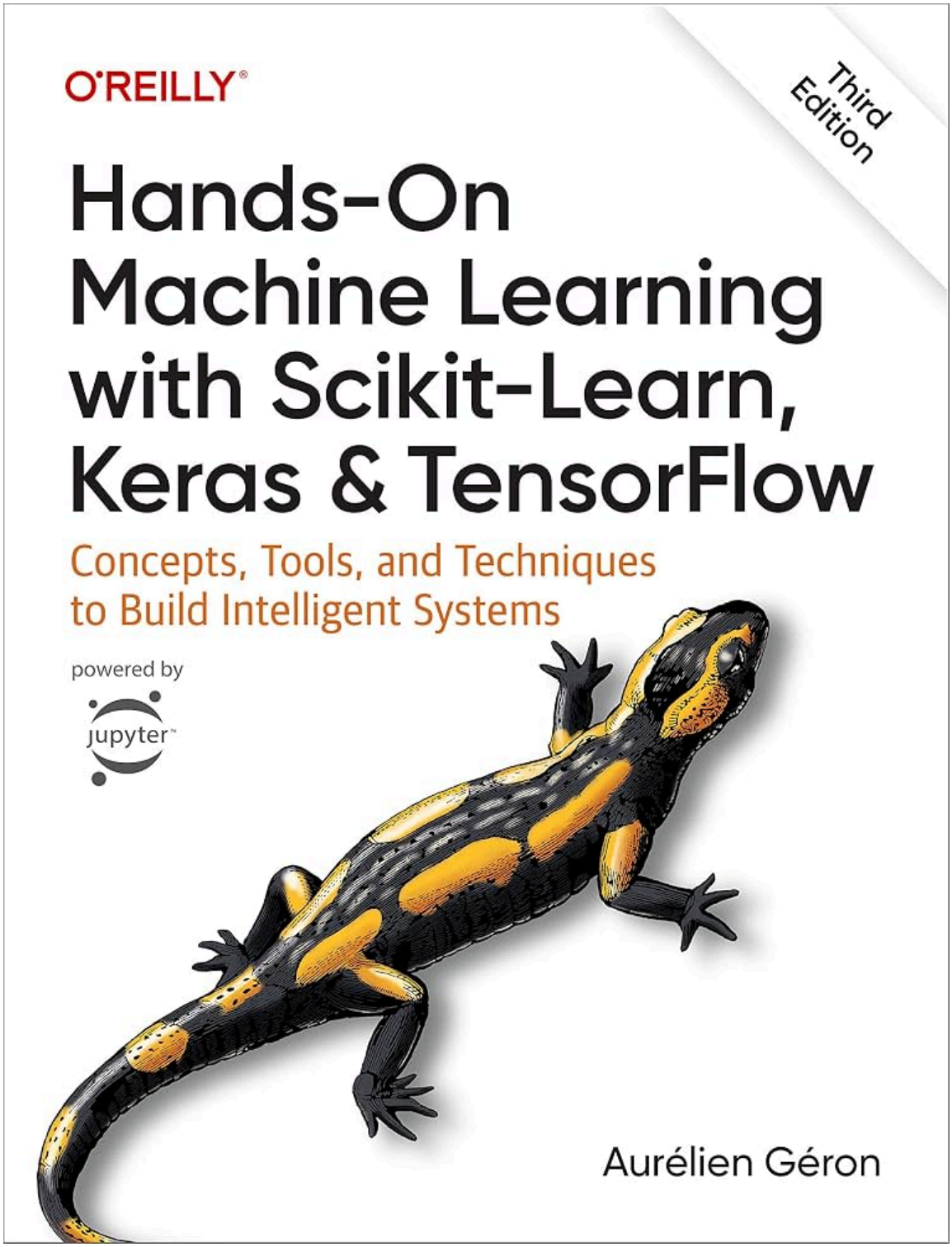
Prologue

Lectures supplémentaires (1/3)



- **The Hundred-Page Machine Learning Book** (Burkov 2019) est un manuel succinct et ciblé qui peut être lu en une semaine, ce qui en fait une excellente ressource d'introduction.
- Disponible sous un modèle « lire d'abord, acheter ensuite », permettant aux lecteurs d'évaluer son contenu avant de l'acheter.
- Son auteur, Andriy Burkov, a obtenu son doctorat en IA à l'Université Laval.

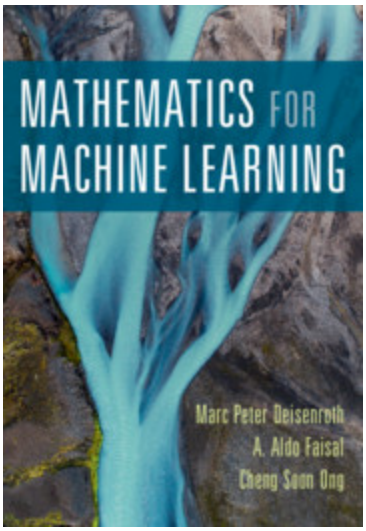
Lectures supplémentaires (2/3)



- **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow** (Géron 2022) fournit des exemples pratiques et exploite des frameworks Python prêts pour la production.
  - La couverture complète inclut non seulement les modèles, mais aussi des bibliothèques pour le réglage des hyperparamètres, le prétraitement des données et la visualisation.
  - Exemples de code et solutions aux exercices disponibles sous forme de Jupyter Notebooks sur GitHub.

- Aurélien Géron est un ancien chef de produit YouTube, qui a dirigé la classification vidéo pour la recherche et la découverte.

Lectures supplémentaires (3/3)



- **Mathematics for Machine Learning** (Deisenroth et al. 2020) vise à fournir les compétences mathématiques nécessaires pour lire des livres sur l'apprentissage automatique.
- PDF du livre
- "Ce livre offre une excellente couverture de tous les concepts mathématiques de base pour l'apprentissage automatique. J'ai hâte de le partager avec les étudiants, les collègues et toute personne intéressée à acquérir une compréhension solide des fondamentaux." Joelle Pineau, Université McGill et Facebook

Références

Burkov, Andriy. 2019. *The Hundred-Page Machine Learning Book*. Andriy Burkov.

Deisenroth, Marc Peter, A. Aldo Faisal, et Cheng Soon Ong. 2020. *Mathematics for Machine Learning*. Cambridge University Press. https://doi.org/10.1017/9781108679930.

Géron, Aurélien. 2022. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3e éd. O'Reilly Media, Inc.

Kingsford, C, et Steven L Salzberg. 2008. « What are decision trees? » *Nature biotechnology* 26 (9): 1011-13. https://doi.org/10.1038/nbt0908-1011.

Mitchell, Tom M. 1997. *Machine Learning*. McGraw-Hill.

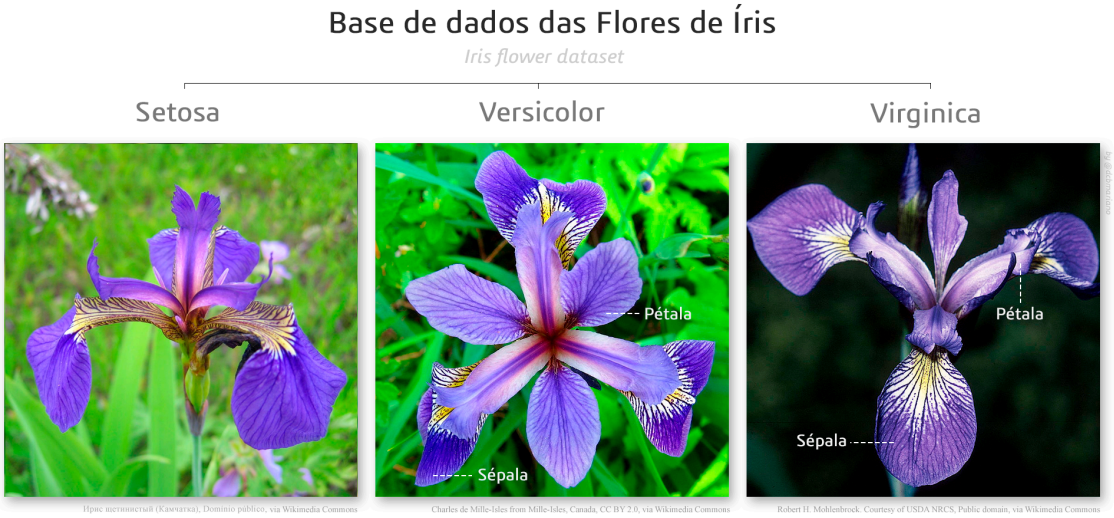
Russell, Stuart, et Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*. 4e éd. Pearson. http://aima.cs.berkeley.edu/.

Prochain cours

- Régression linéaire
- Descente de gradient
- Régression logistique

Annexe: Iris Data Set

Exemple: iris data set



Attribution: Diego Mariano, CC BY-SA 4.0, via Wikimedia Commons. Voir ici pour des informations sur cet ensemble de données.

Exemple: charger les données

In [23]:
```
from sklearn.datasets import load_iris

# Load the Iris dataset

iris = load_iris()
```

De manière pratique, scikit-learn est fourni avec des ensembles de données jouets et réels pour faciliter l'expérimentation. Voir ici pour une description de **load_iris**.

Lorsque vous utilisez votre propre ensemble de données, vous devrez télécharger les fichiers comme nous l'avons fait dans le cahier Jupyter sur la température de la rivière des Outaouais.

Exemple: Utilisant un arbre de décision

In [24]:
```
from sklearn import tree

clf = tree.DecisionTreeClassifier()
```

Il existe des dizaines de classificateurs, y compris ceux-ci : **arbres de décision**, **machines à vecteurs de support**, **k-plus proches voisins**, **régression logistique**, et **réseaux de neurones**.

Dans un `DecisionTreeClassifier`, chaque nœud interne de l'arbre représente une décision basée sur un attribut, chaque branche représente le résultat de cette décision, et chaque nœud terminal représente une étiquette de classe. L'arbre de décision fait des prédictions en parcourant de la racine à un nœud terminal, en suivant les règles de décision définies à chaque nœud. Les arbres de décision sont intuitifs et faciles à interpréter mais peuvent être sujets au surapprentissage s'ils ne sont pas correctement régulés.

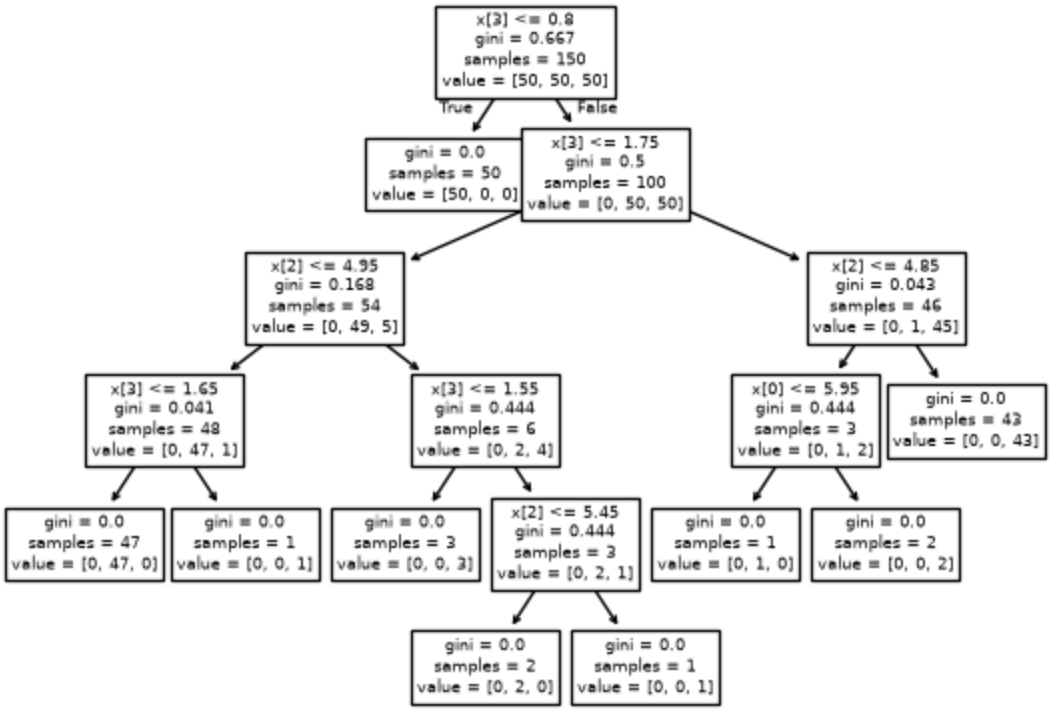
Exemple: Entraînement

In [25]:
```
# It is customary to use X and y for the data and labels

X, y = iris.data, iris.target

# Training

clf = clf.fit(X, y)
```

Tous les classificateurs héritent de `sklearn.base.BaseEstimator` et `sklearn.base.ClassifierMixin`. Par conséquent, tous les classificateurs implémentent `fit`, `predict`, et `score`.

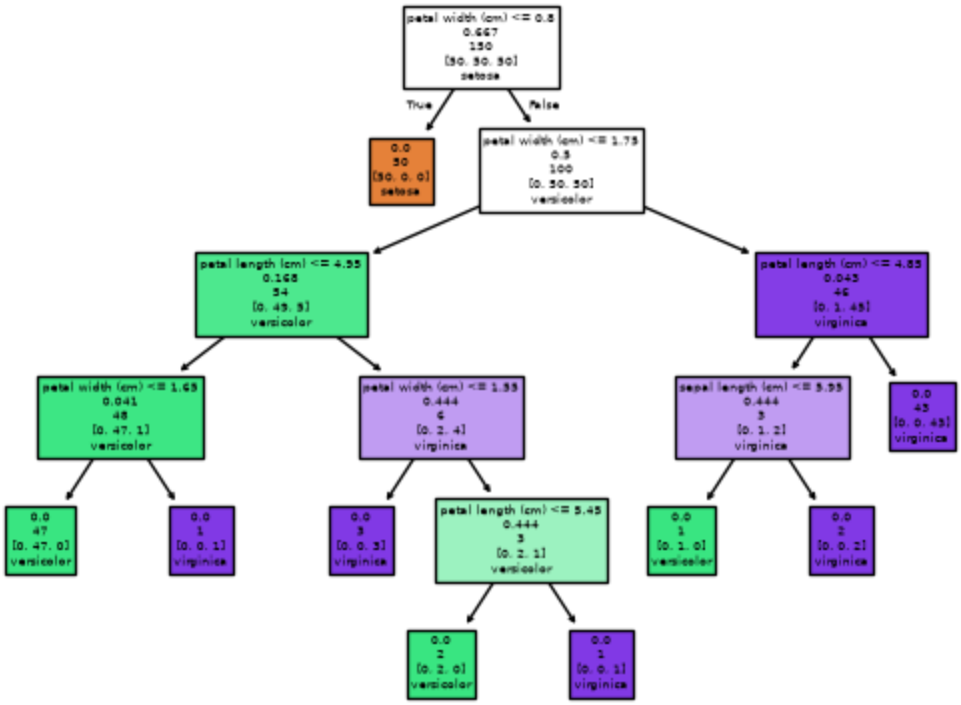
Exemple: Visualiser l'arbre (1/2)

In [26]:
```
import matplotlib.pyplot as plt

tree.plot_tree(clf)
plt.show()
```



Exemple: Visualiser l'arbre (2/2)

In [27]:
```
tree.plot_tree(clf,
               feature_names=iris.feature_names,
               class_names=iris.target_names,
               label='none',
               filled=True)
plt.show()
```



Dans un `DecisionTreeClassifier`, chaque nœud interne de l'arbre représente une décision basée sur un attribut, chaque branche représente le résultat de cette décision, et chaque nœud feuille représente une étiquette de classe. L'arbre de décision fait des prédictions en parcourant de la racine à un nœud feuille, suivant les règles de décision définies à chaque nœud. Les arbres de décision sont intuitifs et faciles à interpréter, mais peuvent être sujets au surapprentissage s'ils ne sont pas correctement régulés.

Pour construire un arbre de décision, la méthode `fit` suit les étapes suivantes :

1. **Sélectionner le Meilleur Attribut** : Choisir l'attribut qui divise le mieux les données en fonction d'un critère comme l'entropie, l'Indice de Gini (par défaut), ou la Log Loss.

2. **Créer un Nœud** : Faire de cet attribut le nœud racine de l'arbre, et créer des branches pour chaque valeur possible de l'attribut.

3. **Diviser le Jeu de Données** : Diviser le jeu de données en sous-ensembles, un pour chaque branche, en fonction des valeurs de l'attribut.

4. **Répéter Récursivement** : Pour chaque sous-ensemble, répéter les étapes 1 à 3 en utilisant uniquement les données de ce sous-ensemble et en excluant l'attribut utilisé au nœud parent.

5. **Conditions d'Arrêt** : Arrêter la récursion lorsque l'une des conditions suivantes est remplie :

- Toutes les instances dans un sous-ensemble appartiennent à la même classe.

- Il n'y a plus d'attributs disponibles pour diviser.
- Une limite de profondeur prédéfinie ou un nombre minimum d'instances par nœud est atteint.

1. **Attribuer des Étiquettes** : Pour chaque nœud feuille, attribuer une étiquette de classe basée sur la classe majoritaire des instances dans ce sous-ensemble.

Ce processus aboutit à un arbre où chaque chemin de la racine à une feuille représente une règle de classification.

Dans la figure ci-dessus, les nœuds de décision contiennent les informations suivantes :

- La règle de décision, par exemple `largeur des pétales (cm) <= 0.8`
- Le score Gini.
- Le nombre d'exemples dans le sous-ensemble correspondant à ce nœud de l'arbre.
- Le nombre d'exemples pour chacune des classes, dans le sous-ensemble correspondant à ce nœud de l'arbre.
- Une prédiction.

Les arbres de décision sont construits en ajoutant progressivement des nœuds de décision, guidés par des exemples d'entraînement étiquetés pour déterminer les divisions optimales. Une règle de décision efficace segmente idéalement les exemples d'entraînement parfaitement dans leurs classes respectives. Par exemple, la règle `largeur des pétales (cm) <= 0.8` illustre ce principe : lorsque la règle est vraie (enfant gauche), toutes les instances sont classées comme Setosa. Inversement, lorsque la règle n'est pas vraie (enfant droit), le sous-ensemble contient uniquement Versicolor et Virginica, sans aucune instance de Setosa. En essence, une bonne règle de décision est celle qui réduit significativement l'entropie.

Voir :

- Kingsford et Salzberg (2008), vous pouvez accéder à l'article ici, en HTML ou PDF, à partir d'un ordinateur avec une adresse IP de l'uOttawa.

Exemple: Prédiction

In [28]:
```
# Creatingg 2 test examples
# 'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width

X_test = [[5.1, 3.5, 1.4, 0.2],[6.7, 3.0, 5.2, 2.3]]

# Prediction

y_test = clf.predict(X_test)

# Printing the predicted labels for our two examples
```

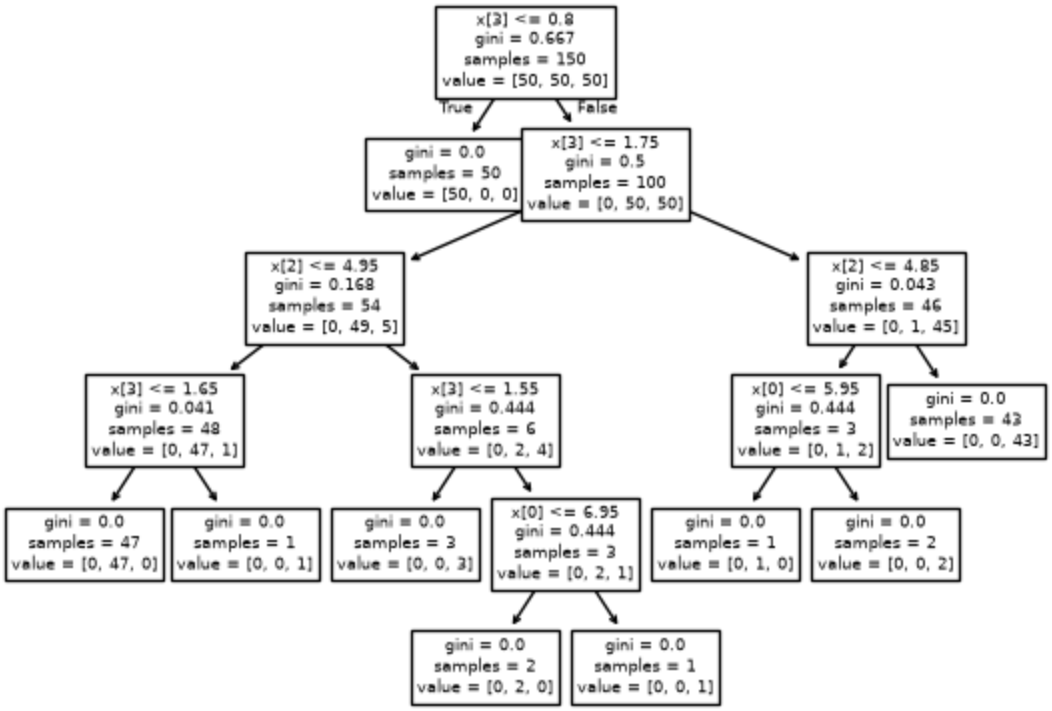
```
print(iris.target_names[y_test])
```

```
['setosa' 'virginica']
```

Exemple: Complet

In [29]:
```
iris = load_iris()
clf = tree.DecisionTreeClassifier()
X, y = iris.data, iris.target
clf = clf.fit(X, y)
tree.plot_tree(clf)
X_test = [[5.1, 3.5, 1.4, 0.2],[6.7, 3.0, 5.2, 2.3]]
y_test = clf.predict(X_test)
print(iris.target_names[y_test])
```

```
['setosa' 'virginica']
```



Exemple: Performance

In [30]:
```
from sklearn.metrics import classification_report, accuracy_score

# Make predictions

y_pred = clf.predict(X)

# Evaluate the model

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=iris.target_names)
```

```
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(report)
```

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       1.00      1.00      1.00        50
   virginica       1.00      1.00      1.00        50

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

The performance of this classifier appears to be perfect at first glance. However, is it really?

Exemple: Discussion

Nous avons vu un exemple complet :

- Chargement des données
- Sélection d'un classificateur
- Entraînement du modèle
- Visualisation du modèle
- Réalisation d'une prédiction

Cependant, plusieurs simplifications ont été faites au cours de ce processus.

Exemple: Prise 2

In [31]:
```
from sklearn.metrics import classification_report, accuracy_score

# Make predictions

y_pred = clf.predict(X)

# Evaluate the model

accuracy = accuracy_score(y, y_pred)
report = classification_report(y, y_pred, target_names=iris.target_names)

print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(report)
```

> **Important**

> Cet exemple est trompeur, voire même fautif!

The performance of this classifier appears to be perfect at first glance. However, is it really?

Exemple: Exploration

In [32]:
```
print(f'Dataset Description:\n{iris["DESCR"]}\n')
```

Dataset Description:
.. _iris_dataset:

Iris plants dataset

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation	
sepal length:	4.3	7.9	5.84	0.83	0.7826	
sepal width:	2.0	4.4	3.05	0.43	-0.4194	
petal length:	1.0	6.9	3.76	1.76	0.9490	(high!)
petal width:	0.1	2.5	1.20	0.76	0.9565	(high!)

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is take
n
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature. Fisher's paper is a classic in the field an
d
is referenced frequently to this day. (See Duda & Hart, for example.) The
data set contains 3 classes of 50 instances each, where each class refers to
a
type of iris plant. One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).

- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
  (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more ...

Exemple: Exploration

In [33]:
```
print(f'Feature Names: {iris.feature_names}')
```

```
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [34]:
```
print(f'Target Names: {iris.target_names}')
```

```
Target Names: ['setosa' 'versicolor' 'virginica']
```

In [35]:
```
print(f'Data Shape: {iris.data.shape}')
```

```
Data Shape: (150, 4)
```

In [36]:
```
print(f'Target Shape: {iris.target.shape}')
```

```
Target Shape: (150,)
```

Exemple: Utilisant `Pandas`

In [37]:
```
import pandas as pd

# Create a DataFrame

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
```

Exemple: Utilisant `Pandas` (suite)

In [38]:
```
# Display the first few rows of the DataFrame

print(df.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
\
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0
```

Exemple: Utilisant `Pandas` (suite)

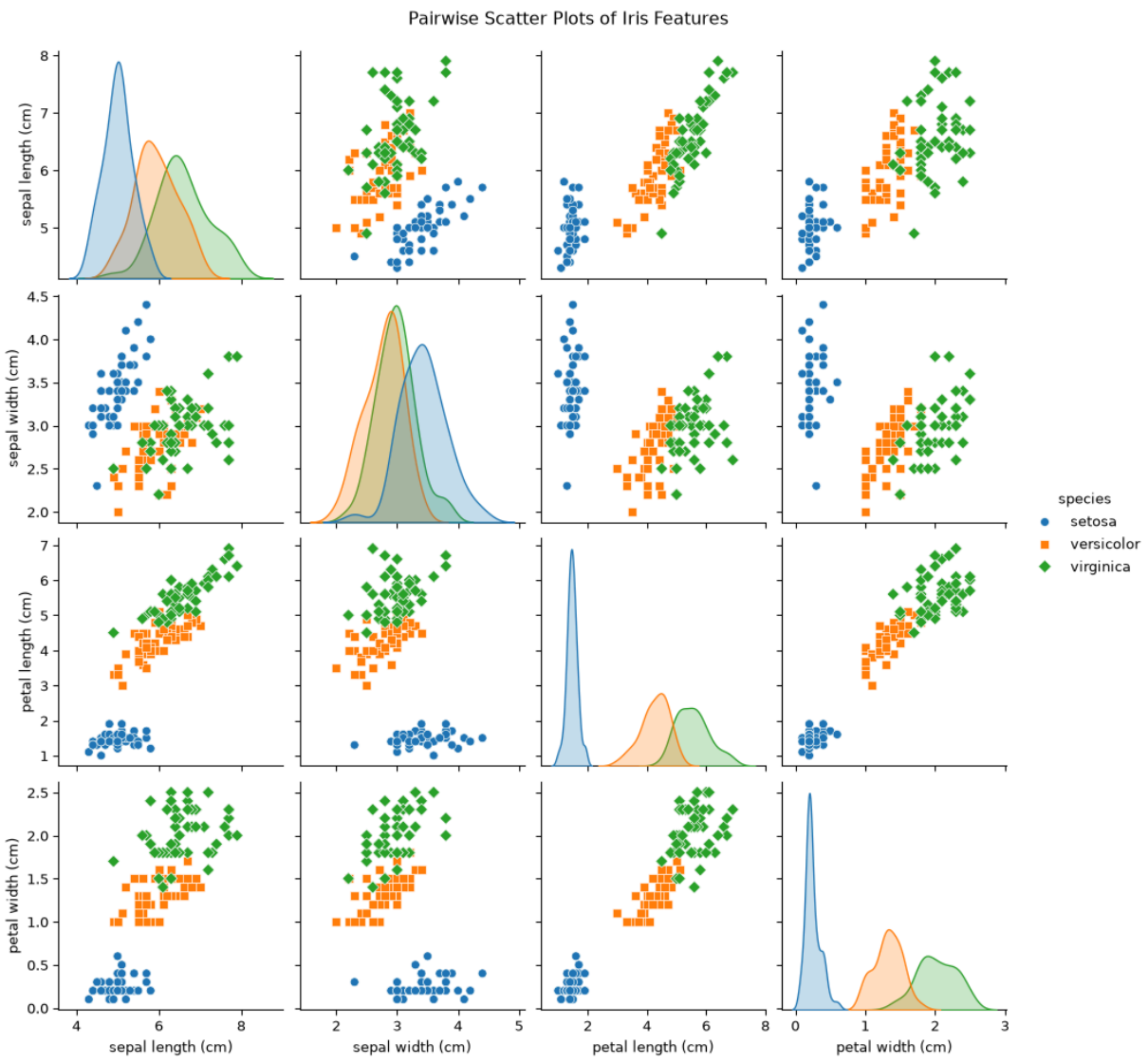
In [39]:
```python
# Summary statistics

print(df.describe())
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)     species
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

Exemple: Utilisant `Seaborn`

In [40]:
```python
import seaborn as sns

# Map target values to species names

df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virgini

# Pairplot using seaborn

sns.pairplot(df, hue='species', markers=["o", "s", "D"])
```

```python
plt.suptitle("Pairwise Scatter Plots of Iris Features", y=1.02)
plt.show()
```



Quelles perspectives peut-on tirer de l'examen des graphiques?

L'image présente tous les diagrammes de dispersion par paire pour les attributs de l'ensemble de données iris, avec les histogrammes pour chaque attribut individuel sur la diagonale. Chaque point représente un exemple (x), et les couleurs indiquent les étiquettes correspondantes (y).

Considérons d'abord les éléments diagonaux :

- Est-il possible de classer les exemples en utilisant un seul attribut ?
- Nous observons que la **longueur** ou la **largeur** du **sépale** seule ne peut pas distinguer les classes.
- Cependant, la **longueur** et la **largeur** du **pétale** nous permettent de différencier **setosa** des deux autres variétés, bien qu'elles ne séparent pas efficacement **versicolor** et **virginica**.

La classe **setosa** forme fréquemment un groupe distinct.

Exemple: Ensemble d'entraînement et ensemble de test

In [41]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

Notre classificateur initial, un arbre de décision, a été construit en utilisant l'ensemble des données complet. Bien qu'il fournisse le **"meilleur"** "ajustement" pour les données, nous ne pouvons pas déterminer sa **performance** sans une évaluation supplémentaire.

Nous aurons beaucoup plus à dire sur les tests dans les semaines à venir.

Pratique : Expérimentez avec différentes valeurs pour `random_state`. Qu'observez-vous ? Pourquoi pensez-vous que cela se produit ? Pourquoi est-il important de définir la valeur de `random_state` ?

Exemple: Créer un nouvel arbre

In [42]:
```
# Train the model
clf = tree.DecisionTreeClassifier()
```

Exemple: Entraîner le modèle

In [43]:
```
# Train the model
clf.fit(X_train, y_train)
```

Exemple: Prédictions

In [44]:
```
# Make predictions
y_pred = clf.predict(X_test)
```

Exemple: mesurer la performance

In [45]:
```
from sklearn.metrics import classification_report, accuracy_score
# Make predictions

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target_name
```

```python
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(report)
```

```
Accuracy: 0.87
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       0.83      0.83      0.83        12
   virginica       0.82      0.82      0.82        11

    accuracy                           0.87        30
   macro avg       0.88      0.88      0.88        30
weighted avg       0.87      0.87      0.87        30
```

Voici une discussion sur la persistance des modèles pour les modèles scikit-learn.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (**SI**GE)

Université d'Ottawa