Introduction to machine learning

CSI 4106 - Fall 2025

Marcel Turcotte

Version: Sep 6, 2025 13:43

Preamble

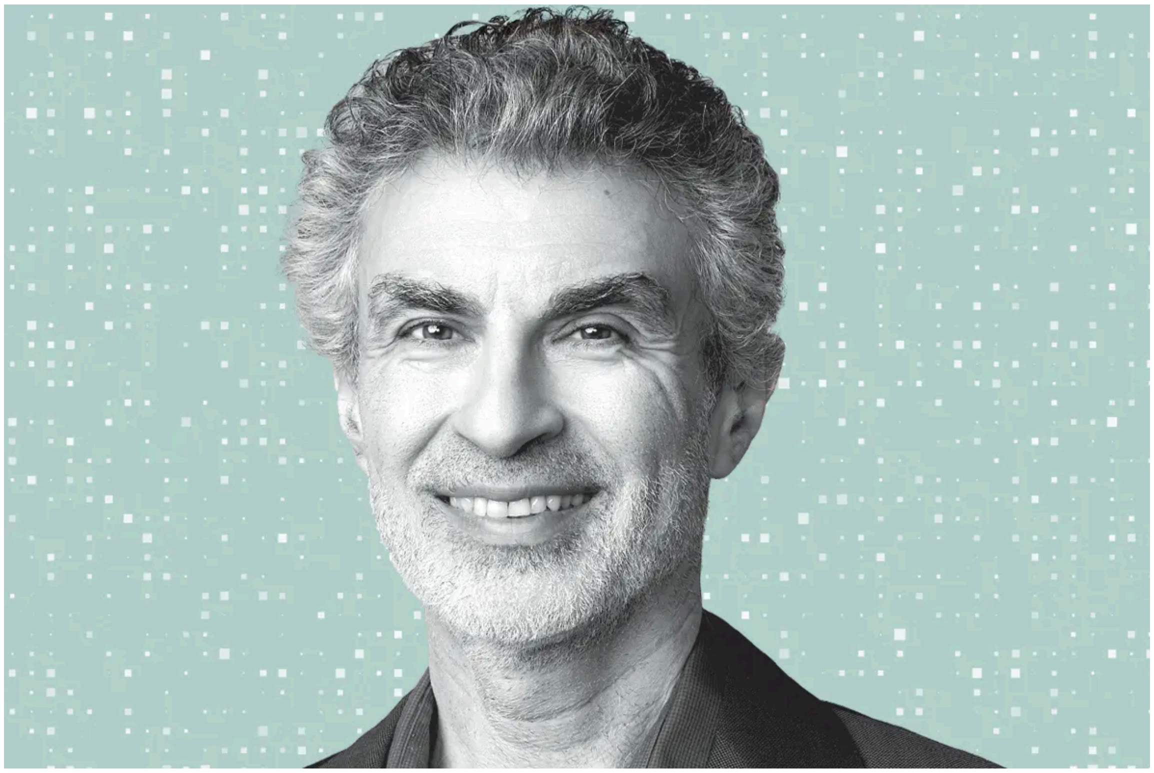
Message of the day

SEPT. 8, 2025
TIME100/AI
THE 100 MOST INFLUENTIAL PEOPLE IN ARTIFICIAL INTELLIGENCE
Artist Refik Anadol and the abstract visualization his studio's AI system created using more than 5,000 past TIME covers
time.com

TIME100 AI 2025 - The 100 Most Influential People in AI 2025, TIME, 2025-08-28.

For the third year in a row, TIME magazine has released its list of the 100 most influential figures in the field of artificial intelligence.

Quote of the day (continued)



Yoshua **Bengio**, Université de Montréal, was recognized again by TIME as one of the most influential individuals in the field of artificial intelligence.

Once again this year, Yoshua Bengio from Université de Montréal has been included in the list of the most influential people in artificial intelligence. Bengio, along with Geoffrey Hinton and Yann LeCun, received the ACM Turing Award in 2018 for their pioneering contributions to the field. The trio is often referred to as the "Fathers of Deep Learning."

- Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award
- The Most-Cited Computer Scientist Has a Plan to Make AI More Trustworthy, by Harry Booth, TIME, 2025-06-03.

> **Remark**
>
> In the evolution of intelligence, learning was one of the **first milestones to emerge**. It is also one of the most **thoroughly understood** mechanisms in natural intelligence.

Debunking the Myths

> ** (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, a machine learning expert based in Quebec City, Canada, authored **The Hundred-Page Machine Learning Book**, which is referenced at the end of this presentation. He is active on LinkedIn and publishes a newsletter, True Positive Weekly, where he shares significant developments and insights from the field that have attracted his attention each week.

Fundamentals of machine learning

In this lecture, we will introduce concepts essential for understanding machine learning, including the types of problems (tasks).

General objective:

- **Describe** the fundamental concepts of machine learning

Learning objectives

- **Summarize** the various types and tasks in machine learning
- **Discuss** the need for a training and test set

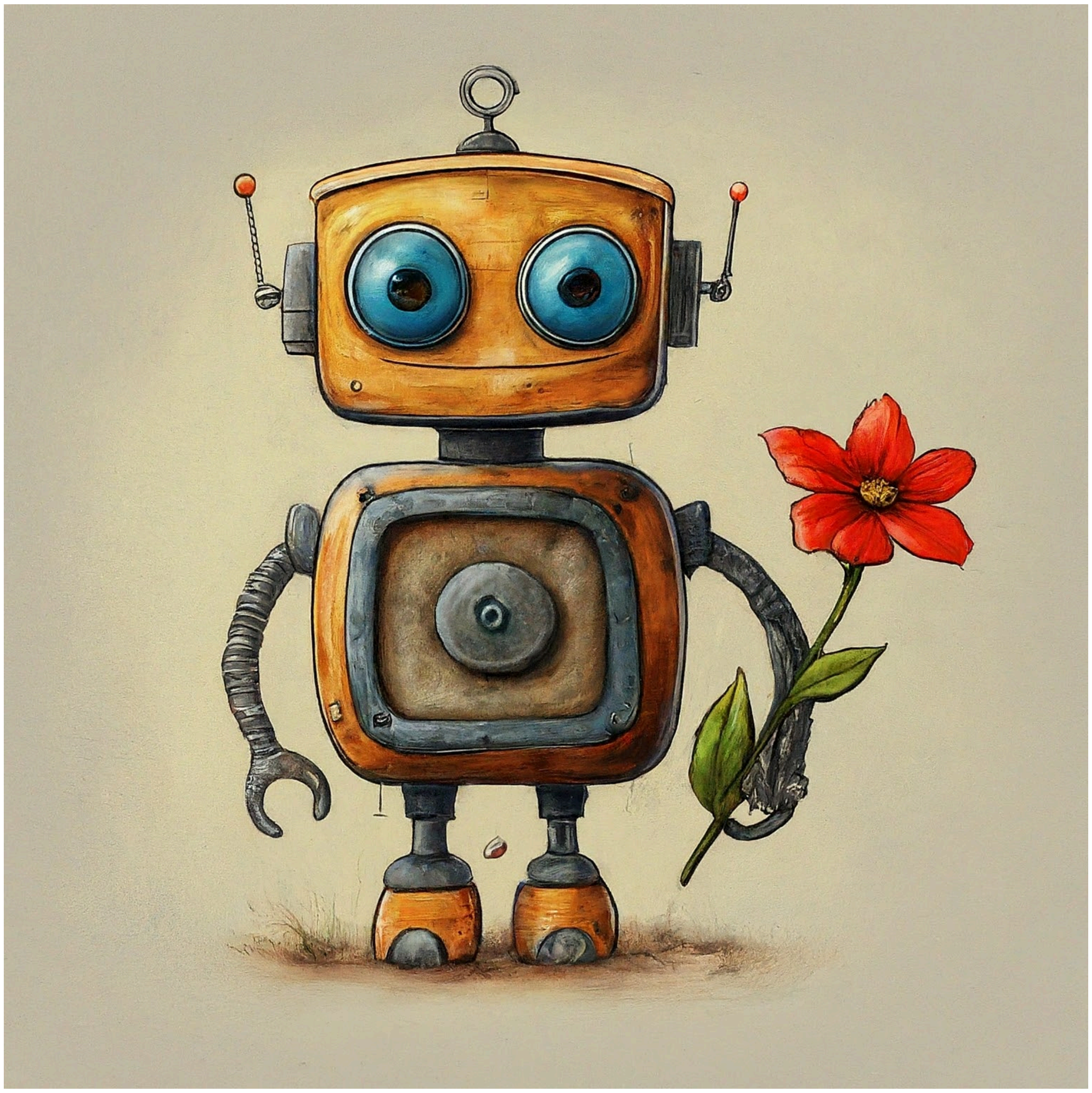
Readings

- Russell and Norvig (2020), Chapter 19: Learning from examples.

Introduction

Rationale

Why a computer program should *learn*?



Attribution: Gemini 1.5 Flash, Aug. 14, 2024, prompted with “In the style of a children’s book, create the image of a cute robot holding a red flower.”

1. **Adaptability and Continuous Improvement:**
    - **Adaptability to Dynamic Environments:** Programs that learn can adapt to changing conditions, ensuring fective operation in dynamic settings.
        - *Example:* Self-driving cars adjusting to traffic and weather changes.
    - **Continuous Improvement:** Learning enables systems to stay current with the latest data and trends without man intervention.
        - *Example:* Spam filters evolving to counter new spam techniques.
2. **Enhanced Performance and Efficiency:**
    - **Improved Performance:** Learning allows programs to enhance their performance based on past experiences or ta.
        - *Example:* Recommendation systems refining suggestions with more user data.

- **Cost-Effectiveness:** Automating the learning process reduces the need for manual updates, leading to cost savings.
    - *Example:* Predictive maintenance systems minimizing manual inspections.

3. **Complex Problem Solving and Hidden Pattern Discovery:**
    - **Handling Complex Problems:** Learning algorithms tackle problems that are too intricate for static, rule-based systems.
        - *Example:* Image recognition distinguishing between different objects.
    - **Discovering Hidden Patterns:** Learning models can uncover hidden relationships in data that are not evident human analysts.
        - *Example:* Identifying complex genomic relationships in bioinformatics.
4. **Personalization and Scalability:**
    - **Personalization:** Learning allows programs to provide tailored outputs to individual users, enhancing user perience.
        - *Example:* Virtual assistants learning user preferences.
    - **Scalability:** Learning algorithms efficiently manage and analyze large datasets, improving their utility.
        - *Example:* Search engines optimizing result relevance with machine learning.
5. **Innovation and Research:**
    - **Fostering Innovation:** Learning algorithms can simulate new ideas, leading to advancements and discoveries.
        - *Example:* Machine learning models predicting drug efficacy in pharmaceutical research.

Definition

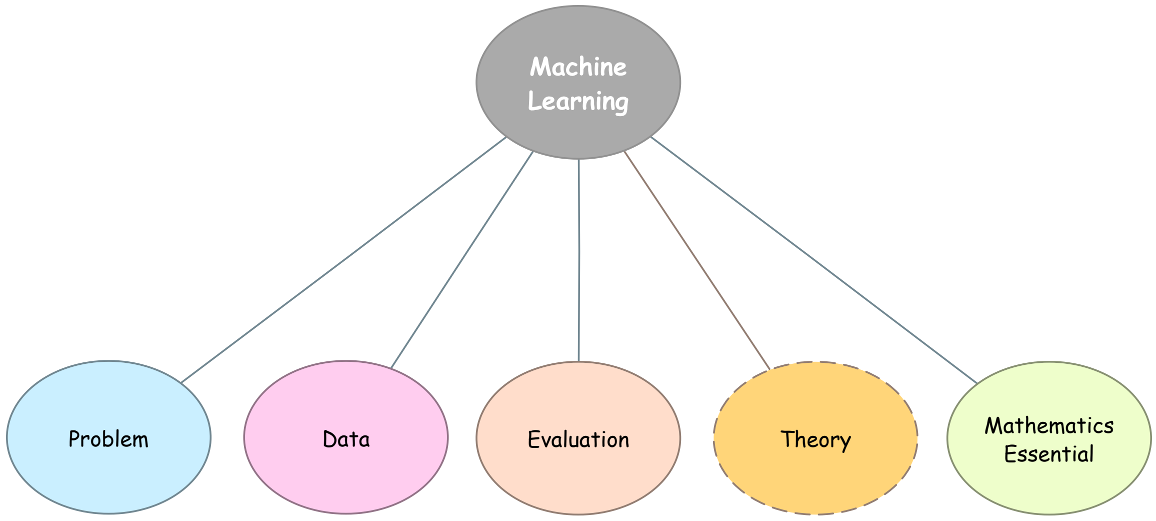
> ** Mitchell (1997), page 2**
>
> A computer program is said to **learn** from **experience** E with respect to some class of **tasks** T and **performance measure** P, if its performance at tasks in T, as measured by P, improves with experience E.

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

Although this book was published in 1997, it has been influential and remains relevant. Unlike many other machine learning books, it is particularly engaging.

This particular definition of learning is often reused.

Concepts



See: images/svg/ml_concepts-00.svg

Types of problems

There are **three (3)** distinct types of **feedback**:

1. **Unsupervised Learning:** No feedback is provided to the algorithm.

2. **Supervised Learning:** Each example is accompanied by a label.

3. **Reinforcement Learning:** The algorithm receives a reward or a punishment following each action.

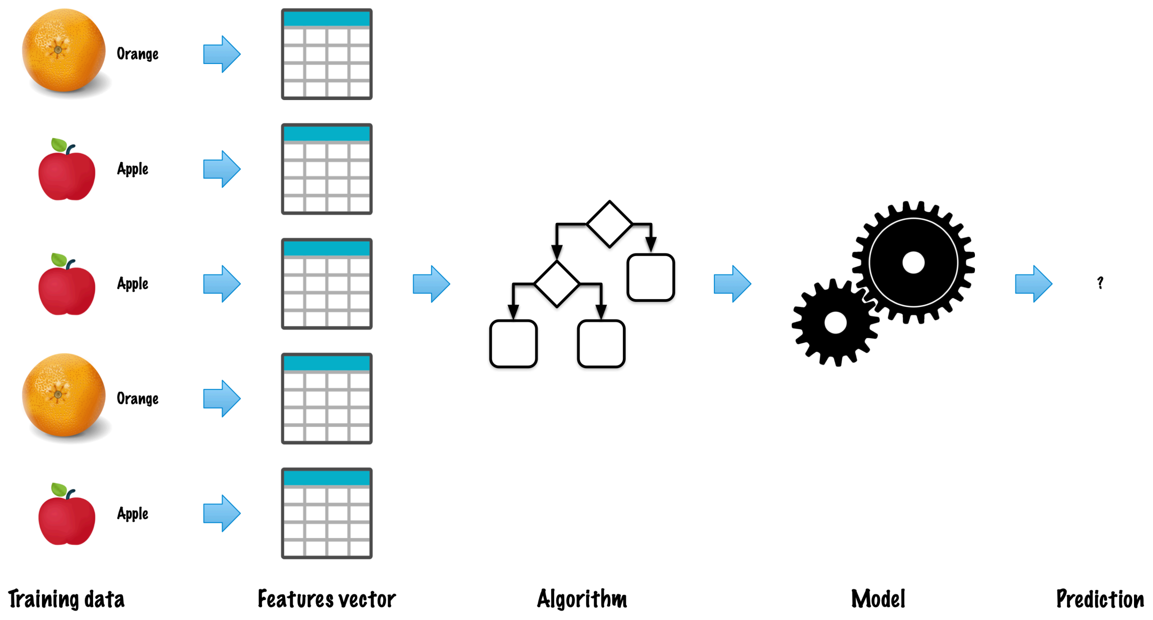
4. . .

Supervised learning is the most extensively studied and arguably the most intuitive type of learning. It is typically the first type of learning introduced in educational contexts.

Two phases

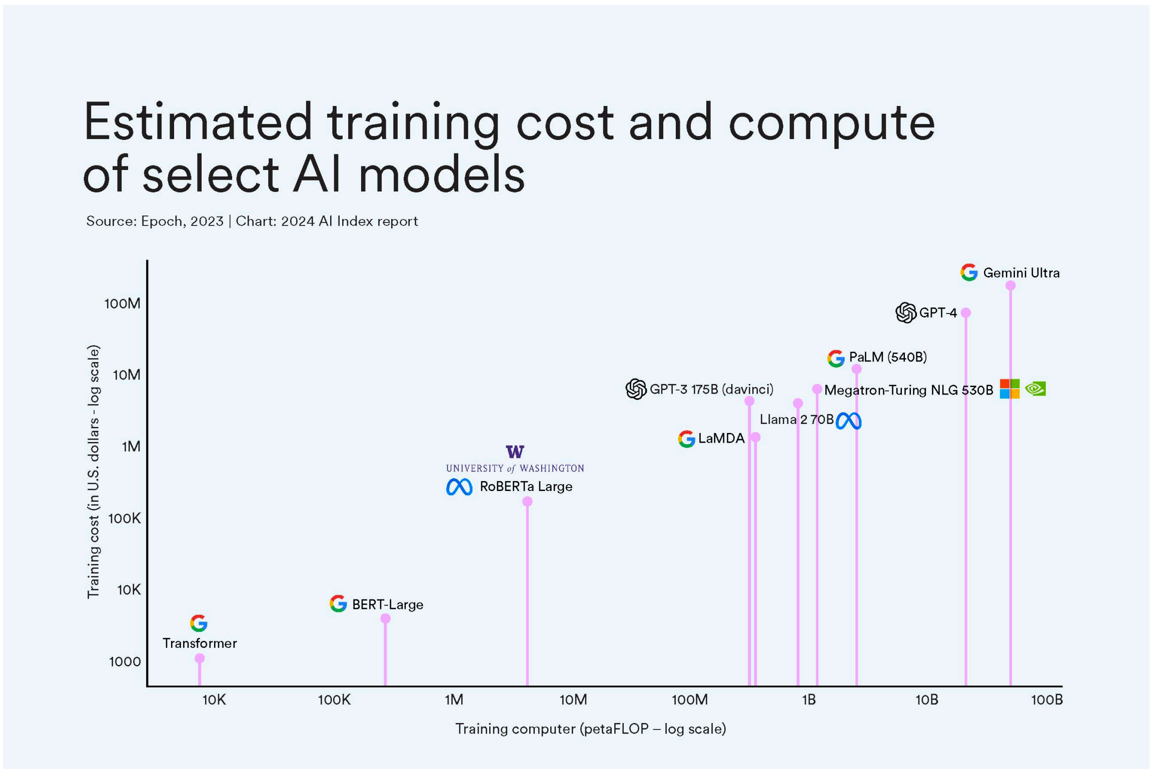
1. **Learning** (building a model)
2. **Inference** (using the model)

Learning (building a model)



In machine learning, the learning phase is often the most challenging and resource-intensive component. This stage necessitates meticulous attention to data curation, as well as the selection and training of an appropriate algorithm.

It is estimated that training contemporary large language models, such as OpenAI's GPT or Google's Gemini Ultra, incurs costs exceeding 100 million USD.

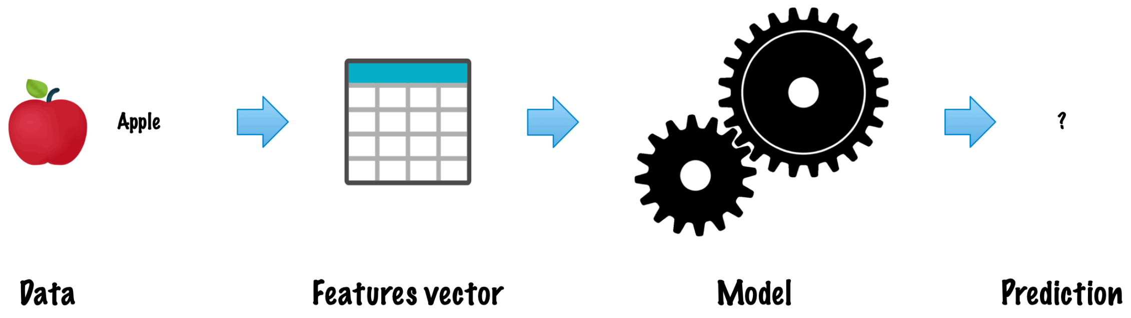


Source: Stanford University Human-Centered Artificial Intelligence 2025 AI Index Report

Developing such models necessitates infrastructure investments that could reach trillions of 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.

Inference (using a model)



Inference typically demands fewer computational resources because it involves utilizing a pre-trained model to predict the outcome for an individual instance. Nevertheless, the emergence of chain-of-thought reasoning models, often referred to as reasoning models, has substantially elevated the computational cost associated with inference.

- 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! (example)

1. Problem: Will They Bite Today?

Objective: Develop a predictive model to classify the likelihood of a successful fishing day into three categories: **'Poor'**, **'Average'**, or **'Excellent'**.

Attribution: Gemini 1.5 Flash, Sept. 10, 2024, prompted with “In the style of a children’s book, generate the image of a fish with a farmer’s hat.”

Supervised learning involves training a model on labeled data so that it can **make predictions on new, unseen data**.

The specific task is **classification**.

“Poor”, “Average”, and “Excellent” are **classes** (for the target variable).

2. Attributes (features)

Various sources, including **The Old Farmer’s Almanac**, suggest that the **moon phase** serves as a reliable predictor of fishing success.

- **Moon Phase (Categorical)**: ‘New Moon’, ‘First Quarter’, ‘Full Moon’, and ‘Last Quarter’.
- **Forecast (Categorical)**: ‘Rainy’, ‘Cloudy’, and ‘Sunny’.
- **Outdoor Temperature (Numerical):** The temperature in Celcius.
- **Water Temperature (Numerical):** The water temperature of the lake or river.

Obviously, real-world applications will have many more attributes.

3. Training data

Example	Moon Phase	Forecast	Outdoor Temperature (°C)	Water Temperature (°C)	Fishing Day Likelihood
1	Full Moon	Sunny	25	22	Excellent
2	New Moon	Cloudy	18	19	Average
3	First Quarter	Rainy	15	17	Poor
4	Last Quarter	Sunny	30	24	Excellent
5	Full Moon	Cloudy	20	20	Average
6	New Moon	Rainy	22	21	Poor

This is identified as a **supervised learning** problem because the value of the target variable is known for each training instance. Additionally, since the target variable's values are categorical, this problem is classified as a **classification task**.

In this context, the training set consists of 6 examples. It is important to note that real-world datasets typically contain a much larger number of examples to ensure robust model training and validation.

The choice of the attributes and the quality of the data are paramount for the performance of the model. An attribute such as the color of your socks is likely not have a great impact the predictions.

Therefore, a machine learning project typically begins with an exploratory phase, which involves analyzing the data, examining distributions, and identifying correlations.

3. Training data: data representation

Moon Phase	Forecast	Outdoor Temperature (°C)	Water Temperature (°C)
Full Moon	Sunny	25	22
New Moon	Cloudy	18	19
First Quarter	Rainy	15	17
Last Quarter	Sunny	30	24
Full Moon	Cloudy	20	20

Moon Phase	Forecast	Outdoor Temperature (°C)	Water Temperature (°C)
New Moon	Rainy	22	21

The **data** is often presented in a tabular (matrix) format, where each row represents an **attribute vector** (feature vector), typically denoted as x_i, which corresponds to the i-th example in the training set.

3. Training data: label representation

Fishing Day Likelihood
Excellent
Average
Poor
Excellent
Average
Poor

The **labels** are generally represented as a column vector, with y_i denoting the label for the i-th example.

4. Model Training

Model training involves using labeled data to teach a machine learning algorithm how to make predictions. This process **adjusts the model's parameters** to **minimize the error between the predicted and actual outcomes**.

4. Model Training (continued)

- **Excellent Fishing Day:**
    - Moon Phase: **Full Moon** or **New Moon**
    - Forecast: **Sunny**
    - Outdoor Temperature: 20°C to 30°C
    - Water Temperature: 20°C to 25°C

. . .

- **Poor Fishing Day:**
    - Moon Phase: **First Quarter** or **Last Quarter**
    - Forecast: **Rainy**
    - Outdoor Temperature: < 20°C or > 30°C
    - Water Temperature: < 20°C or > 25°C

5. Prediction

Given new, **unseen data**, predict whether today will be successful.

. . .

- Moon Phase: **New Moon**
- Forecast: **Sunny**
- Outdoor Temperature: **24°C**
- Water Temperature: **21°C**

Life cycle

1. Data collection and preparation
2. Feature engineering
3. Training
4. Model evaluation
5. Model deployment
6. Monitoring and maintenance

- **Data collection** and **preparation** are critical and labor-intensive processes.
  - There must be **sufficient** data.
  - The data must be of high **quality**; for instance, it should not be excessively noisy.
  - There should be few **missing values**.
  - Most importantly, the data should be **representative**. We expect that new data will be generated from the same process and have the same distribution.
    - The importance of this cannot be overstated.
      - Consider image classification software that was not trained on a diverse sample in terms of ethnicity, gender, body size, or social status.
      - Think of medical applications and the consequences of datasets that are not sufficiently diverse.
- **Feature engineering** is the process of selecting, transforming, and creating input variables (features) to improve the performance of a machine learning model. This involves techniques such as scaling, encoding categorical variables, and generating new features from existing ones to enhance the model's ability to learn patterns and make accurate predictions.
  - Feature engineering used to be a labor-intensive step. One of the main benefits of deep learning is that it can automatically learn features.
- **Model evaluation** is the process of assessing a machine learning model's performance using specific metrics, such as accuracy, precision, recall, F1-score, or AUC-ROC. This typically involves testing the model on a separate validation or test

dataset to ensure it generalizes well to unseen data and meets the desired criteria for accuracy and reliability.

- **Model deployment**: An application is built using the model. It is important to note that most of the time, the parameters of the system are frozen when deployed. First, training is expensive and further training is often unaffordable. Additionally, further training can cause the model to forget previously learned information, leading to degraded performance on previously seen examples.
- **Monitoring and maintenance**: The performance of the model needs to be continuously monitored. **Concept drift** is often observed, requiring the system to be retrained. Spam detection is a good example. Once the system is deployed, spammers adapt and find ways to circumvent the spam detection mechanisms put in place.

Formal definitions

Supervised learning (notation)

The **data set** ("experience") is a collection of labelled examples.

- $\{(x_i, y_i)\}_{i=1}^N$
  - Each x_i is a **feature** (**attribute**) vector with D dimensions.
  - $x_i^{(j)}$ is the value of the feature j of the example i, for $j \in 1 \ldots D$ and $i \in 1 \ldots N$.
  - The **label** y_i is either a **class**, taken from a finite list of classes, $\{1, 2, \ldots, C\}$, a **real number**, or a **complex object** (tree, graph, etc.).

. . .

Problem: Given the data set as input, create a **model** that can be used to predict the value of y for an unseen x.

This notation follows that of **The Hundred-Page Machine Learning Book** by Andriy Burkov.

Supervised learning (notation, contd)

- When the **label** y_i is a **class**, taken from a finite list of classes, $\{1, 2, \ldots, C\}$, we call the task a **classification** task.

- When the **label** y_i is a **real number**, we call the task a **regression** task.

Can you think of examples of regression tasks?

Here are several regression tasks along with their real-world applications:

1. **House Price Prediction**:
    - **Application**: Estimating the market value of residential properties based on features such as location, size, number of bedrooms, age, and amenities.
2. **Stock Market Forecasting**:
    - **Application**: Predicting future prices of stocks or indices based on historical data, financial indicators, and economic variables.
3. **Weather Prediction**:
    - **Application**: Estimating future temperatures, rainfall, and other weather conditions using historical weather data and atmospheric variables.
4. **Sales Forecasting**:
    - **Application**: Predicting future sales volumes for products or services by analyzing past sales data, market trends, and seasonal patterns.
5. **Energy Consumption Prediction**:
    - **Application**: Forecasting future energy usage for households, industries, or cities based on historical consumption data, weather conditions, and economic factors.
6. **Medical Cost Estimation**:
    - **Application**: Predicting healthcare costs for patients based on their medical history, demographic information, and treatment plans.
7. **Traffic Flow Prediction**:
    - **Application**: Estimating future traffic volumes and congestion levels on roads and highways using historical traffic data and real-time sensor inputs.
8. **Customer Lifetime Value (CLV) Estimation**:
    - **Application**: Predicting the total revenue a business can expect from a customer over the duration of their relationship, based on purchasing behavior and demographic data.
9. **Economic Indicators Forecasting**:
    - **Application**: Predicting key economic indicators such as GDP growth, unemployment rates, and inflation using historical economic data and market trends.
10. **Demand Forecasting**:
    - **Application**: Estimating future demand for products or services in various industries like retail, manufacturing, and logistics to optimize inventory and supply chain management.
11. **Real Estate Valuation**:
    - **Application**: Assessing the market value of commercial properties like office buildings, malls, and industrial spaces based on location, size, and market conditions.
12. **Insurance Risk Assessment**:
    - **Application**: Predicting the risk associated with insuring individuals or properties, which helps in determining premium rates, based on historical

claims data, and demographic factors.

13. **Ad Click-Through Rate (CTR) Prediction**:
    - **Application**: Estimating the likelihood that a user will click on an online advertisement based on user behavior, ad characteristics, and contextual factors.
14. **Loan Default Prediction**:
    - **Application**: Predicting the probability of a borrower defaulting on a loan based on credit history, income, loan amount, and other financial indicators.

Here are some regression task applications that can typically be found in mobile phone applications:

1. **Battery Life Prediction**:
    - **Application**: Estimating remaining battery life based on usage patterns, running applications, and device settings.
2. **Health and Fitness Tracking**:
    - **Application**: Predicting calorie burn, heart rate, or sleep quality based on user activity, biometrics, and historical health data.
3. **Personal Finance Management**:
    - **Application**: Forecasting future expenses or savings based on spending habits, income patterns, and budget goals.
4. **Weather Forecasting**:
    - **Application**: Providing personalized weather forecasts based on current location and historical weather data.
5. **Traffic and Commute Time Estimation**:
    - **Application**: Predicting travel times and suggesting optimal routes based on historical traffic data, real-time conditions, and user behavior.
6. **Image and Video Quality Enhancement**:
    - **Application**: Adjusting image or video quality settings (e.g., brightness, contrast) based on lighting conditions and user preferences.
7. **Fitness Goal Achievement**:
    - **Application**: Estimating the time needed to achieve fitness goals such as weight loss or muscle gain based on user activity and dietary input.
8. **Mobile Device Performance Optimization**:
    - **Application**: Predicting the optimal settings for device performance and battery life based on usage patterns and app activity.

These applications leverage regression tasks to provide personalized, efficient, and context-aware services that enhance the user experience on mobile devices.

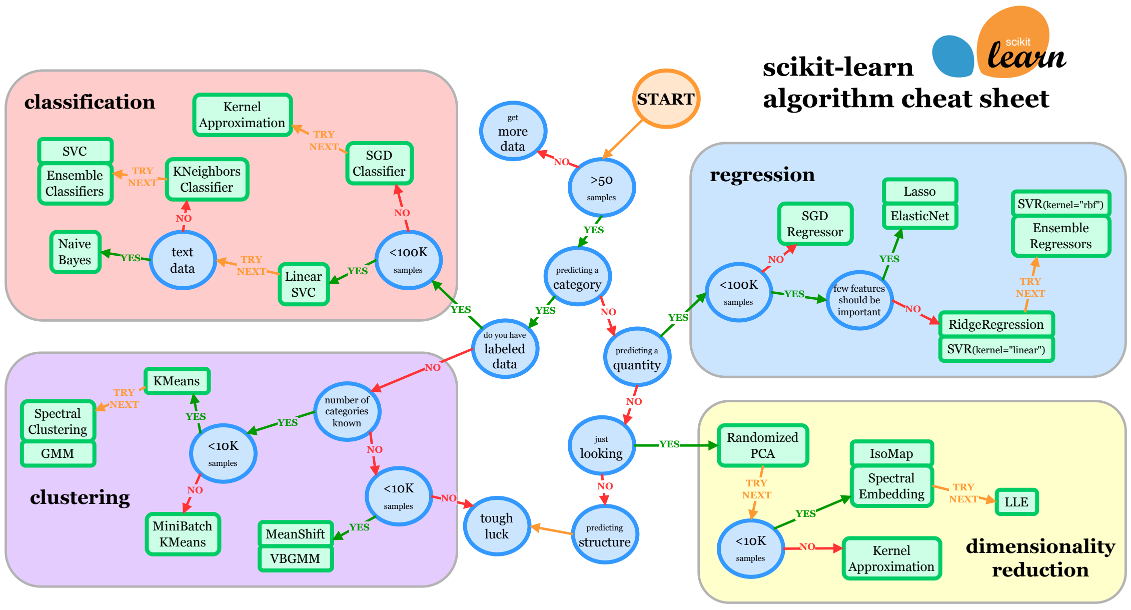
Example with code

`Scikit-learn`

> ** scikit-learn.org**
>
> `Scikit-learn` is an open source machine learning library that supports **supervised** and **unsupervised** learning. It also provides various tools for **model fitting**, **data preprocessing**, **model selection**, **model evaluation**, and many other utilities.
>
> `Scikit-learn` provides **dozens of built-in machine learning algorithms and models**, called estimators.
>
> Built on NumPy, SciPy, and matplotlib.

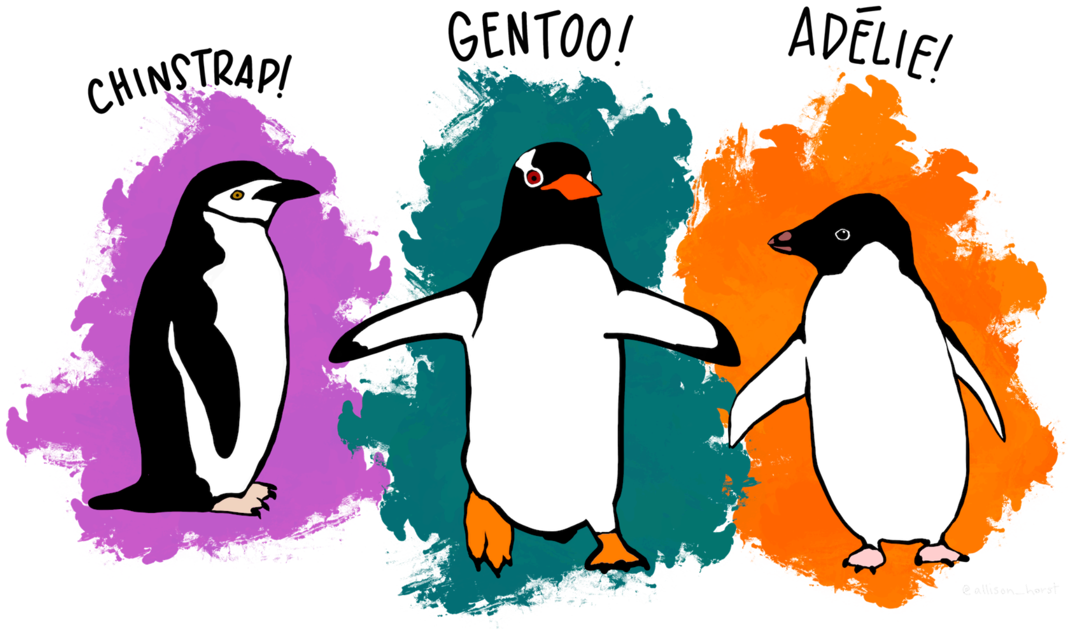
We will use `Scikit-learn` for our next example, but also throught out the coming weeks.

`Scikit-learn`



Attribution: Choosing the right estimator

Example: Palmer Pinguins Dataset



The Palmer penguins dataset by Allison Horst, Alison Hill, and Kristen Gorman. **Artwork** by @allison_horst

> The Palmer penguins dataset by Allison Horst, Alison Hill, and Kristen Gorman was first made publicly available as an R package. The goal of the Palmer Penguins dataset is to replace the highly overused Iris dataset for data exploration & visualization.

> Using this Python package you can easily load the Palmer penguins into your Python environment.

Initially, we will examine the entire example from a high-level perspective to provide a clear overview of the steps involved. Subsequently, we will revisit this example in greater detail.

Example: In Case of a Missing Library

In [1]:
```
try:
  from palmerpenguins import load_penguins
except:
  ! pip install palmerpenguins
  from palmerpenguins import load_penguins
```

If you are executing this Jupyter Notebook within Google Colab or any environment where the library is not pre-installed, proceed with the installation.

Example: Loading the Data

In [2]:
```python
# It is customary to use X and y for the data and labels

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

Example: Using a DecisionTree

In [3]:
```python
from sklearn import tree

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

There are dozens of classifiers, including these ones: **decision trees**, **support vector machines**, **k-nearest neighbors**, **logistic regression**, and **neural networks**.

Example: Training

In [4]:
```python
# Training

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

All the classifiers inherit from `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin`. Accordingly, all the classifiers implement `fit`, `predict`, and `score`.

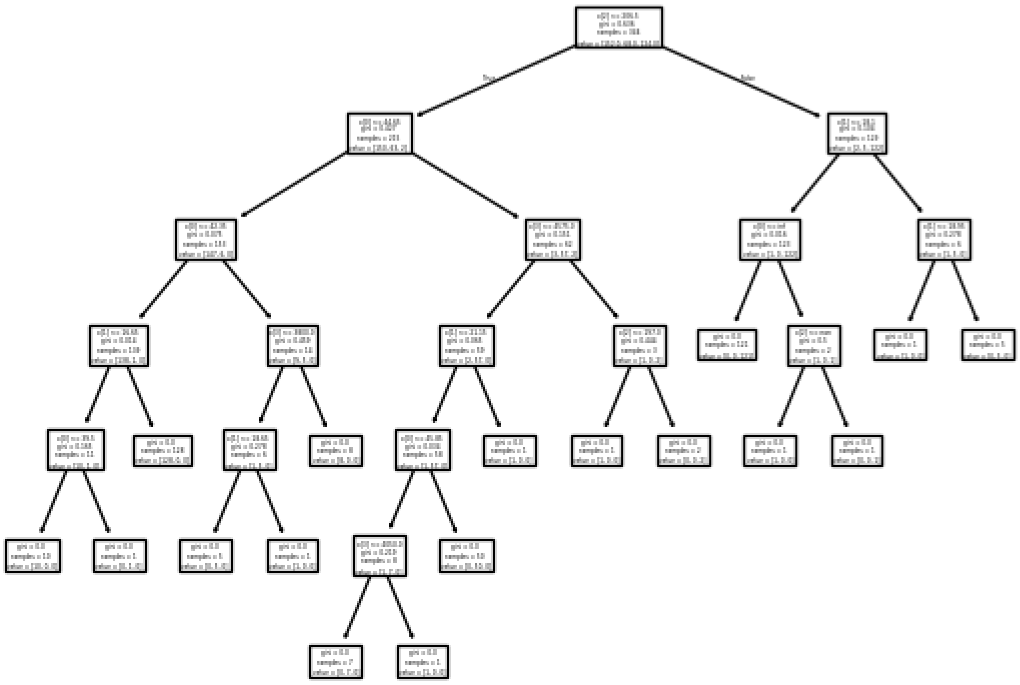
The `DecisionTreeClassifier` in scikit-learn constructs a decision tree by recursively splitting the dataset into subsets based on the attribute that results in the highest information gain (e.g., Gini impurity, entropy). Here is a concise description of the process:

1. **Initialization**: The algorithm starts with the entire dataset as the root node.
2. **Splitting Criteria**: For each node, it evaluates all possible splits across all attributes to find the one that best separates the classes. This is typically done by minimizing a criterion such as Gini impurity or entropy.
3. **Recursive Splitting**: The dataset is divided into subsets based on the selected attribute and threshold, creating child nodes. This process is repeated recursively for each child node.
4. **Stopping Conditions**: Splitting stops when a predefined criterion is reached, such as a maximum tree depth, a minimum number of samples per leaf, or if further splitting does not significantly improve information gain.
5. **Terminal Nodes**: Once splitting is complete, each terminal node is assigned a class label based on the majority class of the samples in that node.

The resulting tree can then be used to classify new samples by traversing from the root to a terminal node, following the decision rules defined at each node.

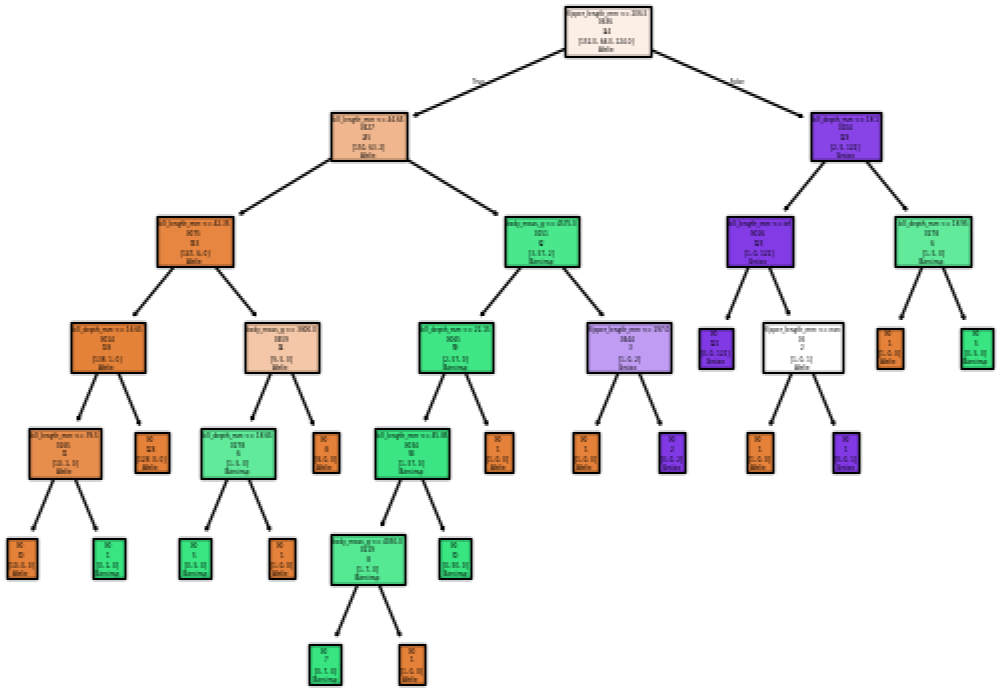
Example: Visualizing the tree (1/2)

```
In [5]: import matplotlib.pyplot as plt

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



Example: Visualizing the tree (2/2)

```
In [6]: target_names = ['Adelie','Chinstrap','Gentoo']

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



Example: Prediction

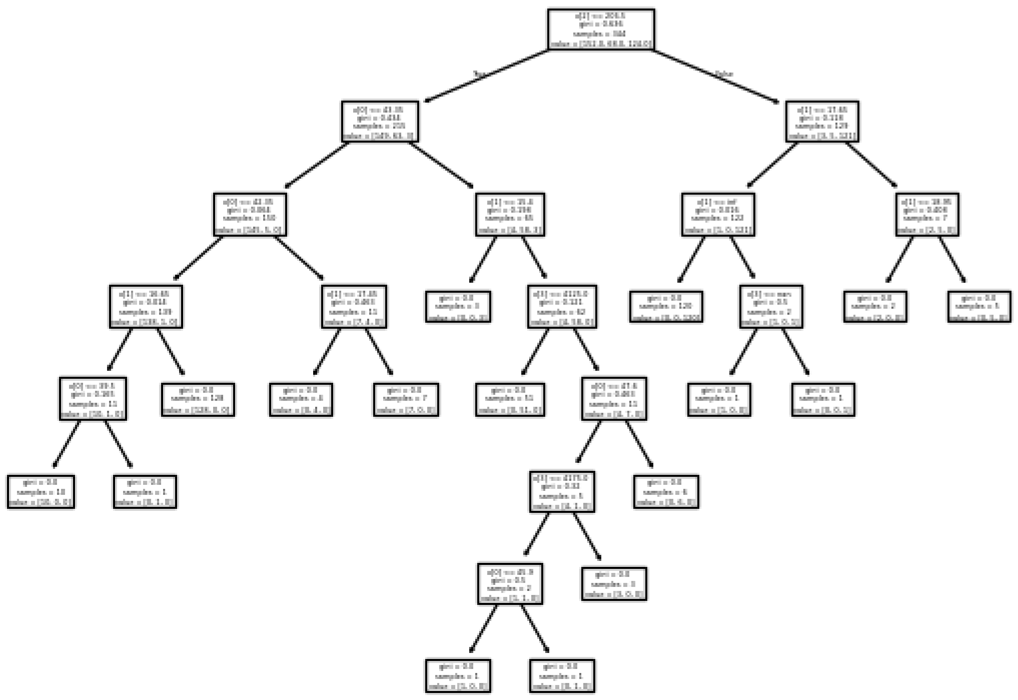
In [7]:
```
import pandas as pd

# Creating 2 test examples

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

# Prediction

y_test = clf.predict(X_test)

# Printing the predicted labels for our two examples

print(y_test)
```

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

Example: Complete

In [8]:
```
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']
```



Example: Performance

```
In [9]:
```

```python
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=target_names)

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

```
Accuracy: 1.00
Classification Report:
              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
```

Example: Discussion

We have demonstrated a complete example:

- Loading the data
- Selecting a classifier
- Training the model
- Visualizing the model
- Making a prediction

However, several simplifications were made throughout this process.

Example: Wait a Minute!

In [10]:
```
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=target_names)

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

> **Important**
>
> This example is misleading, or even flawed!

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

Example: Exploration

In [11]:
```
penguins = load_penguins()
```

...

In [12]:
```
type(penguins)
```

```
pandas.core.frame.DataFrame
```

...

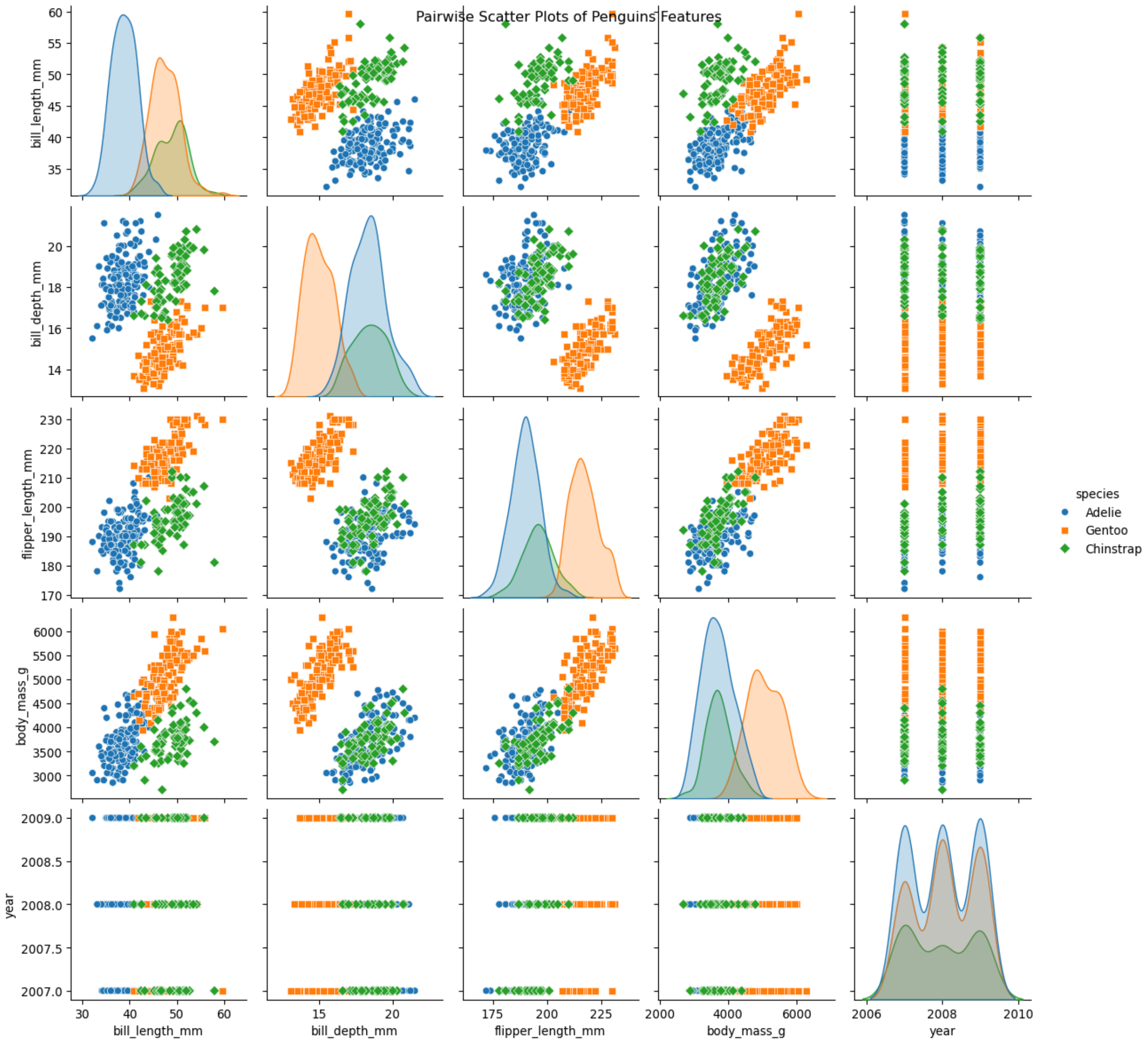
In [13]:
```
penguins.head()
```

Example: Exploration

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

Example: Using `Seaborn`

In [15]:
```
import seaborn as sns

# Pairplot using seaborn

sns.pairplot(penguins, hue='species', markers=["o", "s", "D"])
plt.suptitle("Pairwise Scatter Plots of Penguins Features")
plt.show()
```



What insights can be drawn from examining the graphs?

The image presents all pairwise scatter plots for the iris dataset features, with the diagonal displaying histograms for each individual feature. Each dot represents an example (x), and the colors indicate the corresponding labels (y).

Let's first consider the diagonal elements:

- Is it possible to classify the examples using a single feature?
- We observe that each feature alone cannot distinguish between the classes.
- However, in **bill_depth** vs **body_mass** or **flipper_length** allow us to differentiate **Gentoo** from the other two species, although they do not separate **Adélie** and **Chinstrap** effectively.

The class **Gentoo** frequently forms a distinct cluster.

Example: Training and Test Set

In [16]:
```
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
```

Our initial classifier, a decision tree, was constructed using the entire dataset. While it provides the **"best"** fit for the data, we cannot ascertain its **predictive accuracy** without further evaluation.

We will have a lot more to say about testing in the coming weeks.

Practice: Experiment with different values for `random_state`. What do you observe? Why do you think this occurs? For instance, set `random_state` to `42`. Why is it important to set the `random_state` value?

Example: Creating a New Classifier

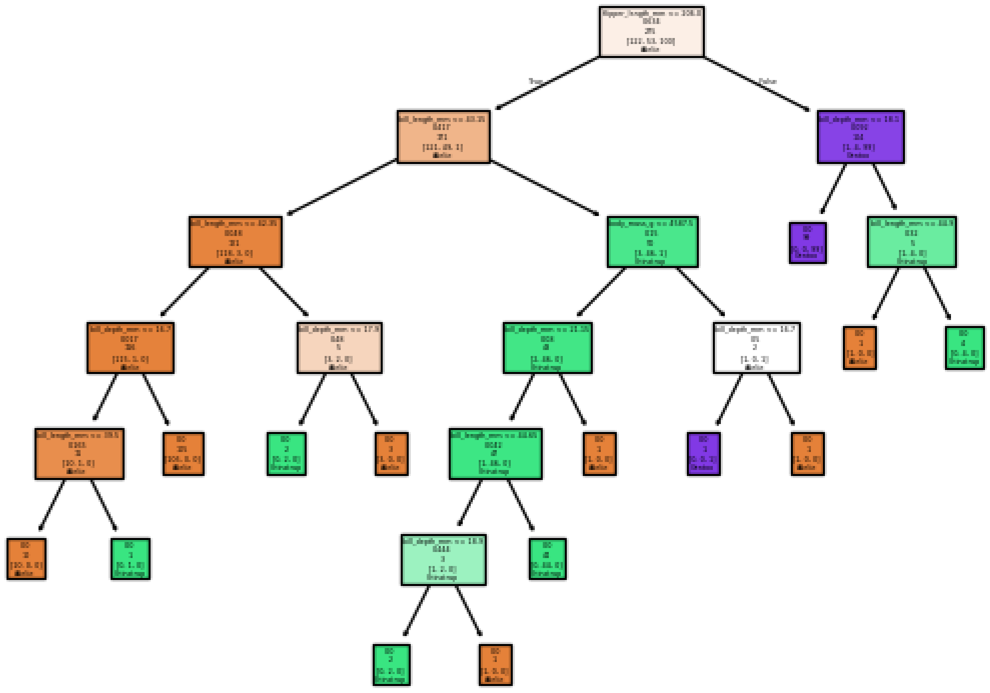
In [17]:
```
clf = tree.DecisionTreeClassifier()
```

Example: Training the New Classifier

In [18]:
```
clf.fit(X_train, y_train)
```

Example: Visualizing the Tree

In [19]:
```
tree.plot_tree(clf,
               feature_names = X.columns,
               class_names = target_names,
               label = 'none',
                filled = True)
plt.show()
```



Example: Making Predictions

In [20]:
```
# Make predictions
y_pred = clf.predict(X_test)
```

Example: Measuring the Performance

In [21]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=target_names)

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

```
Accuracy: 0.93
Classification Report:
              precision    recall  f1-score   support

      Adelie       0.96      0.87      0.91        30
   Chinstrap       0.83      1.00      0.91        15
      Gentoo       0.96      0.96      0.96        24

    accuracy                           0.93        69
   macro avg       0.92      0.94      0.93        69
weighted avg       0.93      0.93      0.93        69
```

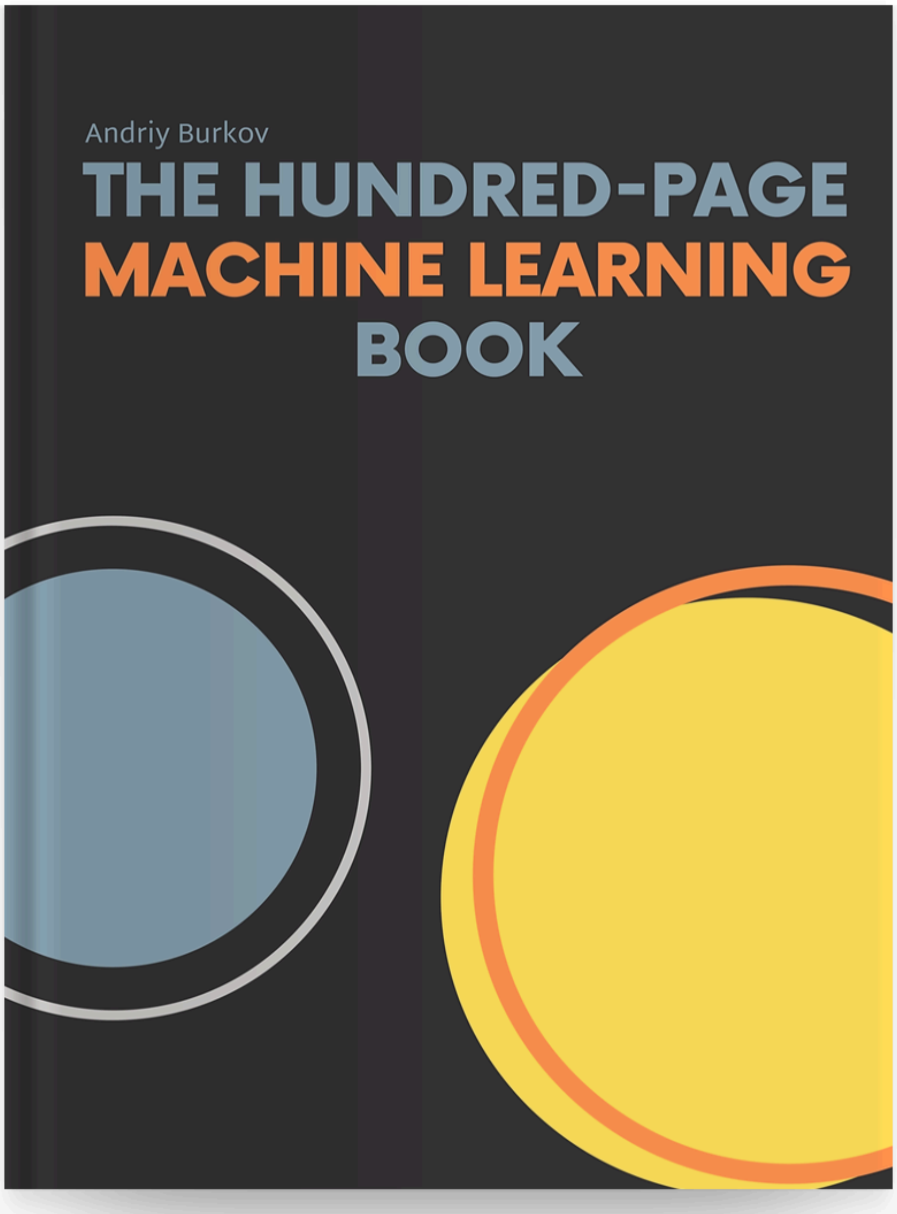
Here is a discussion on model persistence for scikit-learn models.

Summary

- We introduced relevant terminology.
- We examined a hypothetical example.
- Next, we explored a complete example using scikit-learn.
- We performed a detailed exploration of our data.
- Finally, we recognized the necessity of an independent test set to accurately measure performance.

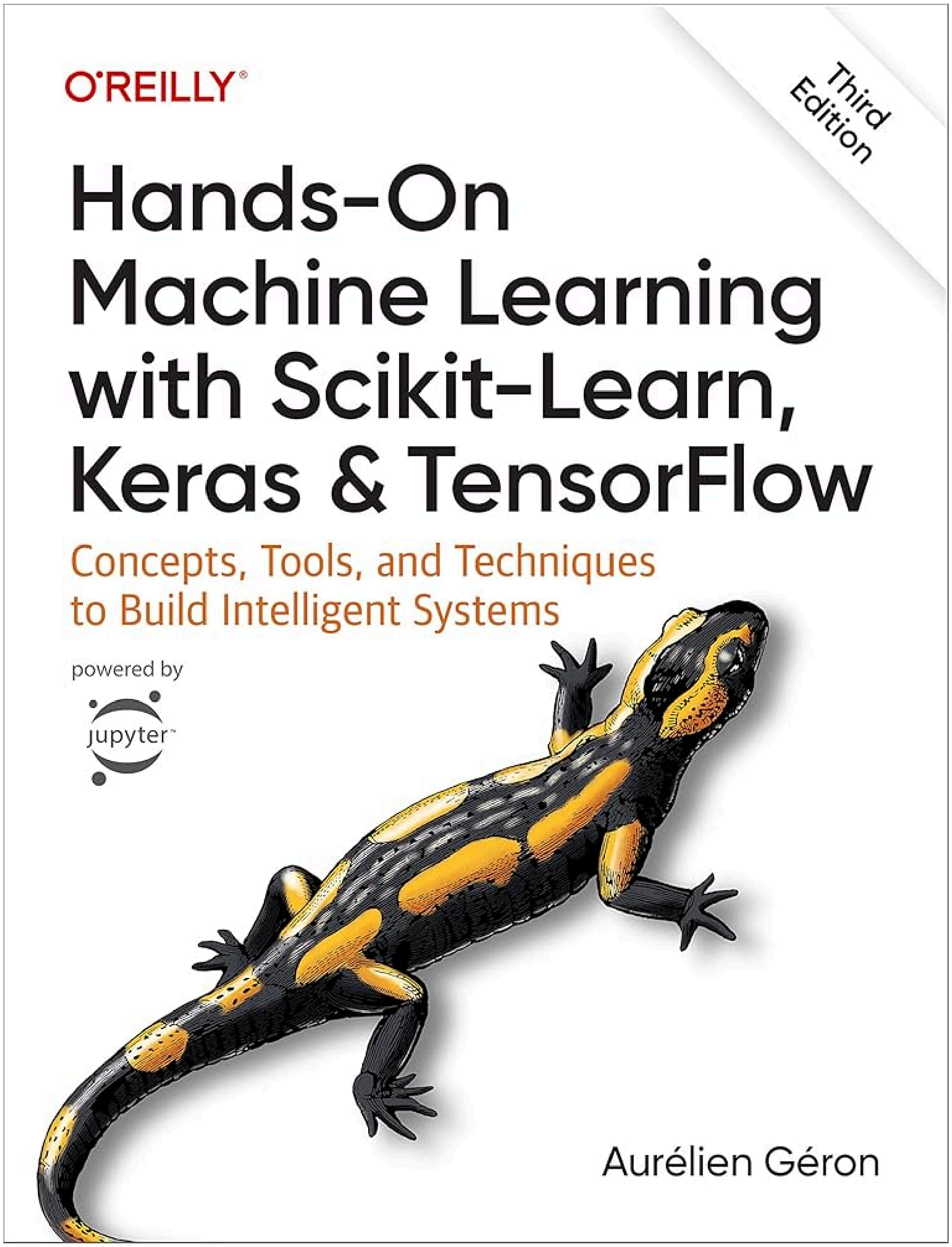
Prologue

Further readings (1/3)



- **The Hundred-Page Machine Learning Book** (Burkov 2019) is a succinct and focused textbook that can feasibly be read in one week, making it an excellent introductory resource.
- Available under a "read first, buy later" model, allowing readers to evaluate its content before purchasing.
- Its author, Andriy Burkov, received his Ph.D. in AI from Université Laval.

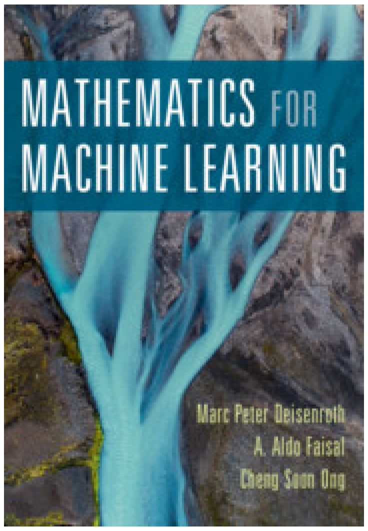
Further readings (2/3)


O'REILLY
Third Edition
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by
jupyter
Aurélien Géron

- **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow** (Géron 2022) provides practical examples and leverages production-ready Python frameworks.
  - Comprehensive coverage includes not only the models but also libraries for hyperparameter tuning, data preprocessing, and visualization.
  - Code examples and solutions to exercises available as Jupyter Notebooks on GitHub.

- Aurélien Géron is a former YouTube Product Manager, who lead video classification for Search & Discovery.

Further readings (3/3)



- **Mathematics for Machine Learning** (Deisenroth, Faisal, and Ong 2020) aims to provide the necessary mathematical skills to read machine learning books.
- PDF of the book
- “This book provides great coverage of all the basic mathematical concepts for machine learning. I’m looking forward to sharing it with students, colleagues, and anyone interested in building a solid understanding of the fundamentals.” Joelle Pineau, McGill University and Facebook

References

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

Deisenroth, Marc Peter, A. Aldo Faisal, and 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*. 3rd ed. O’Reilly Media, Inc.

Kingsford, C, and 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*. New York: McGraw-Hill.

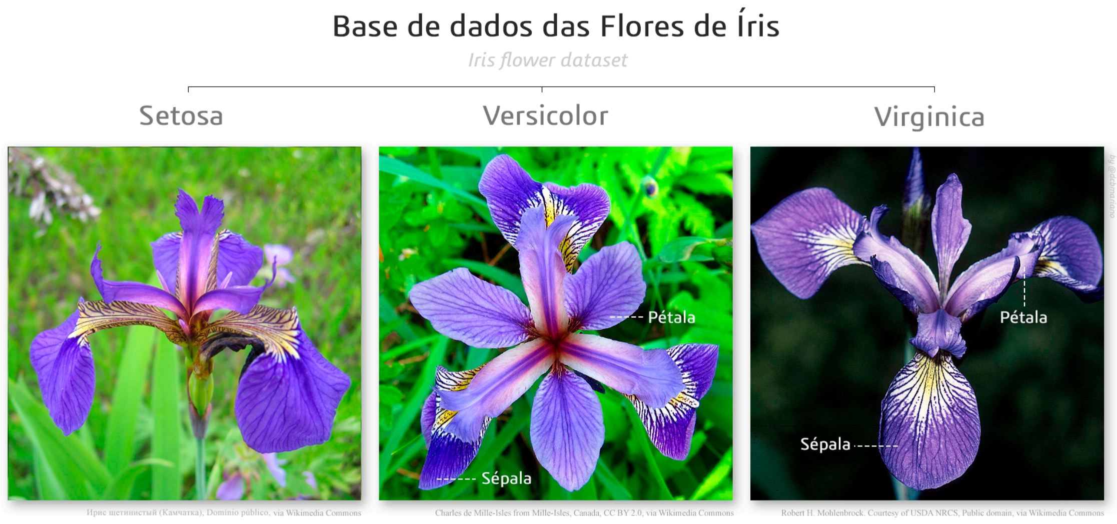
Russell, Stuart, and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*. 4th ed. Pearson. http://aima.cs.berkeley.edu/.

Next lecture

- Linear regression
- Gradient descent

Appendix: Iris Data Set

Example: iris data set


Base de dados das Flores de Íris
Iris flower dataset
Setosa
Versicolor
Virginica
Pétala
Sépala
Pétala
Sépala
Ирис щетинистый (Камчатка), Domínio público, via Wikimedia Commons
Charles de Mille-Isles from Mille-Isles, Canada, CC BY 2.0, via Wikimedia Commons
Robert H. Mohlenbrock. Courtesy of USDA NRCS, Public domain, via Wikimedia Commons

Attribution: Diego Mariano, CC BY-SA 4.0, via Wikimedia Commons. See here for information on this dataset.

Example: loading the data

In [22]:
```python
from sklearn.datasets import load_iris

# Load the Iris dataset

iris = load_iris()
```

Conveniently, scikit-learn comes with toy and real-world datasets to facilitate experimentation. See here for a description of **load_iris**.

When using your own dataset, you will have to download the files as we have in the Ottawa River Temperature Jupyter Notebook.

Example: Using a DecisionTree

In [23]:
```python
from sklearn import tree

clf = tree.DecisionTreeClassifier()
```

There are dozens of classifiers, including these ones: **decision trees**, **support vector machines**, **k-nearest neighbors**, **logistic regression**, and **neural networks**.

Example: Training

In [24]:
```python
# 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)
```

All the classifiers inherit from `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin`. Accordingly, all the classifiers implement `fit`, `predict`, and `score`.

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

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

2. **Critères de division** : Pour chaque nœud, il évalue toutes les divisions possibles à travers toutes les attributs pour trouver celle qui sépare le mieux les classes. Cela est généralement fait en minimisant un critère comme l'impureté de Gini ou l'entropie.

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

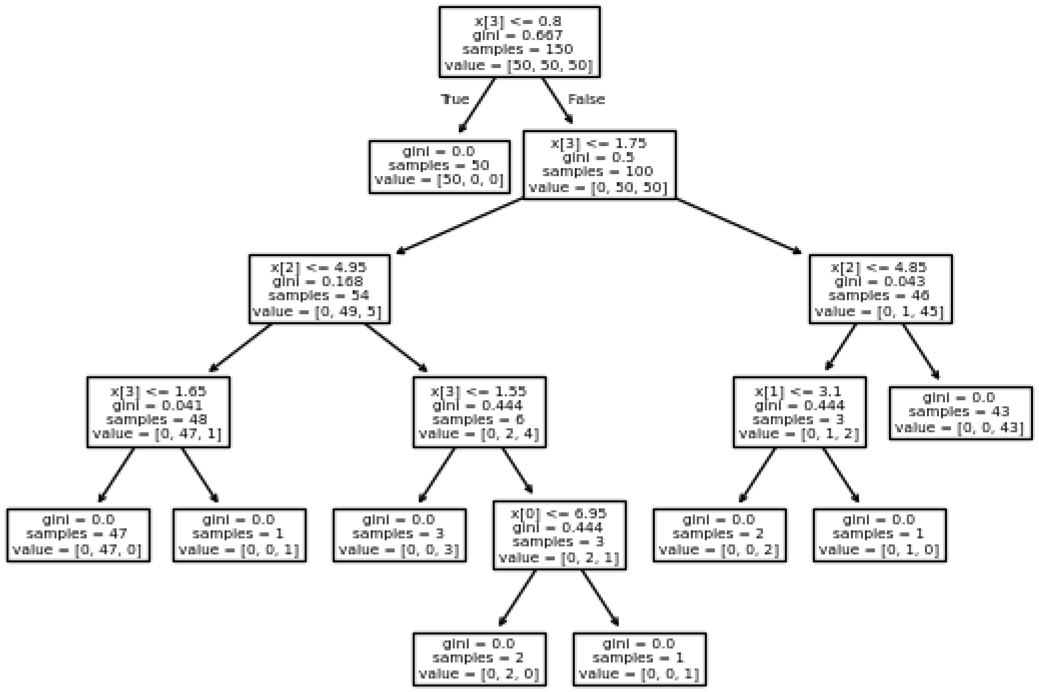
4. **Conditions d'arrêt** : La division s'arrête lorsqu'un critère prédéfini est atteint, comme une profondeur maximale de l'arbre, un nombre minimum d'échantillons par feuille, ou si une division supplémentaire n'améliore pas significativement le gain d'information.

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.

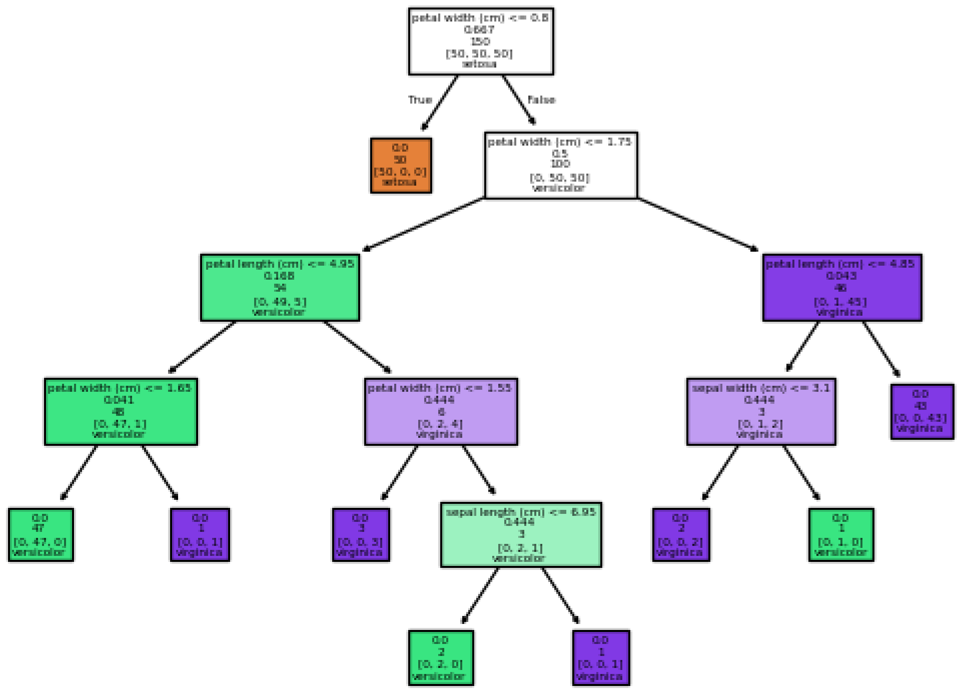
Example: Visualizing the tree (1/2)

In [25]:
```
import matplotlib.pyplot as plt

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



Example: Visualizing the tree (2/2)

In [26]:
```
tree.plot_tree(clf,
               feature_names=iris.feature_names,
               class_names=iris.target_names,
               label='none',
               filled=True)
plt.show()
```


petal width (cm) <= 0.8 0.667 150 [50, 50, 50] setosa
True
False
0.0 50 [50, 0, 0] setosa
petal width (cm) <= 1.75 0.5 100 [0, 50, 50] versicolor
petal length (cm) <= 4.95 0.168 54 [0, 49, 5] versicolor
petal length (cm) <= 4.85 0.043 46 [0, 1, 45] virginica
petal width (cm) <= 1.65 0.041 48 [0, 47, 1] versicolor
petal width (cm) <= 1.55 0.444 6 [0, 2, 4] virginica
sepal width (cm) <= 3.1 0.444 3 [0, 1, 2] virginica
0.0 43 [0, 0, 43] virginica
0.0 47 [0, 47, 0] versicolor
0.0 1 [0, 0, 1] virginica
0.0 3 [0, 0, 3] virginica
sepal length (cm) <= 6.95 0.444 3 [0, 2, 1] versicolor
0.0 2 [0, 0, 2] virginica
0.0 1 [0, 1, 0] versicolor
0.0 2 [0, 2, 0] versicolor
0.0 1 [0, 0, 1] virginica

In a `DecisionTreeClassifier`, each internal node of the tree represents a decision based on a feature, each branch represents the outcome of that decision, and each leaf node represents a class label. The decision tree makes predictions by traversing from the root to a leaf node, following the decision rules defined at each node. Decision trees are intuitive and easy to interpret but can be prone to overfitting if not properly regulated.

To build a decision tree, the method `fit` follows these steps:

1. **Select the Best Attribute**: Choose the attribute that best splits the data based on a criterion like Entropy, Gini Index (default), or Log Loss.
2. **Create a Node**: Make this attribute the root node of the tree, and create branches for each possible value of the attribute.
3. **Split the Dataset**: Divide the dataset into subsets, one for each branch, based on the attribute's values.
4. **Repeat Recursively**: For each subset, repeat steps 1-3 using only the data in that subset and excluding the attribute used at the parent node.
5. **Stop Conditions**: Stop the recursion when one of the following conditions is met:
   - All instances in a subset belong to the same class.
   - No more attributes are available for splitting.
   - A predefined depth limit or minimum number of instances per node is reached.
6. **Assign Labels**: For each leaf node, assign a class label based on the majority class of instances in that subset.

This process results in a tree where each path from the root to a leaf represents a classification rule.

In the figure above, the decision nodes contain the following information. - The decision rule, e.g. `petal width (cm) <= 0.8` - The Geni score. - The number of examples in the subset corresponding to this node of the tree. - The number of examples for each of the classes, in the subset corresponding to this node of the tree. - A prediction.

Decision trees are constructed by incrementally adding decision nodes, guided by labeled training examples to determine optimal splits. An effective decision rule ideally segregates the training examples perfectly into their respective classes. For instance, the rule `petal width (cm) <= 0.8` exemplifies this: when the rule holds true (left child), all instances are classified as Setosa. Conversely, when the rule does not hold (right child), the subset contains only Versicolor and Virginica, with no Setosa instances. In essence, a good decision rule is one that significantly reduces entropy.

See: - Kingsford and Salzberg (2008), you can access the paper here, html or PDF, from a computer with a uOttawa IP address.

Example: Prediction

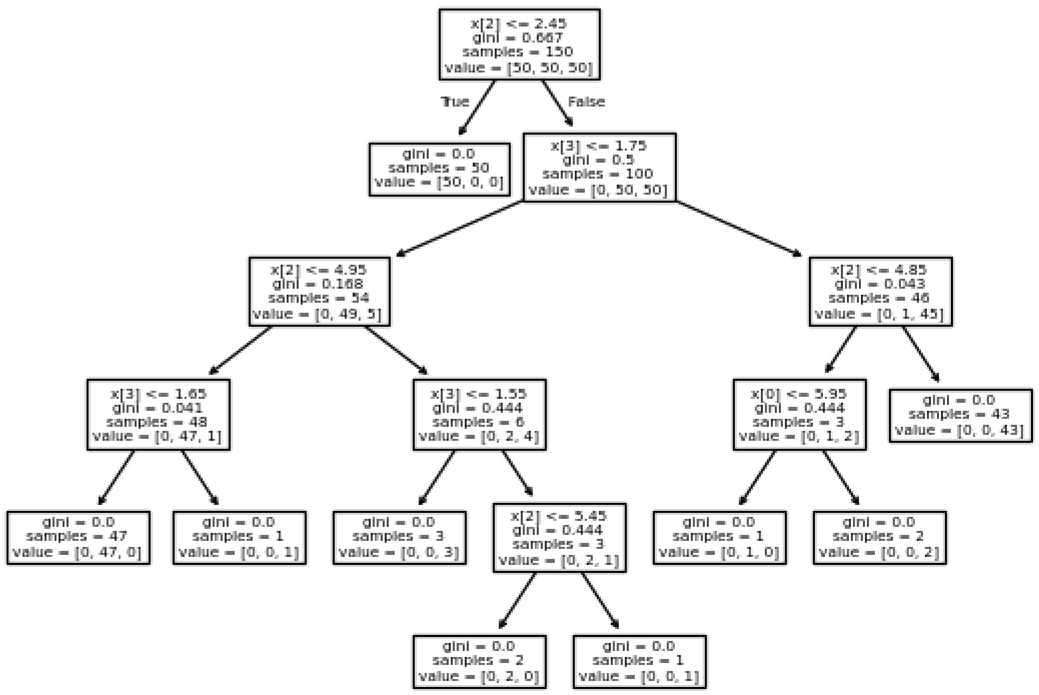
In [27]:
```
# Creatingg 2 test examples
# 'sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal widtl

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']
```

Example: Complete

In [28]:
```
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']
```



Example: Performance

In [29]:
```python
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?

Example: Discussion

We have demonstrated a complete example:

- Loading the data
- Selecting a classifier
- Training the model
- Visualizing the model
- Making a prediction

However, several simplifications were made throughout this process.

Example: Take 2

In [30]:
```python
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**
>
> This example is misleading, or even flawed!

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

Example: Exploration

In [31]:
```python
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 Analysi
s.
    (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 Transactio
ns
    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 ...
```

Example: Exploration

In [32]:
```
print(f'Feature Names: {iris.feature_names}')
```

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

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

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

In [34]:
```
print(f'Data Shape: {iris.data.shape}')
```

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

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

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

Example: Using `Pandas` (continued)

In [36]:
```
import pandas as pd

# Create a DataFrame

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

Example: Using `Pandas` (continued)

In [37]:
```
# 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  
```

Example: Using `Pandas` (continued)

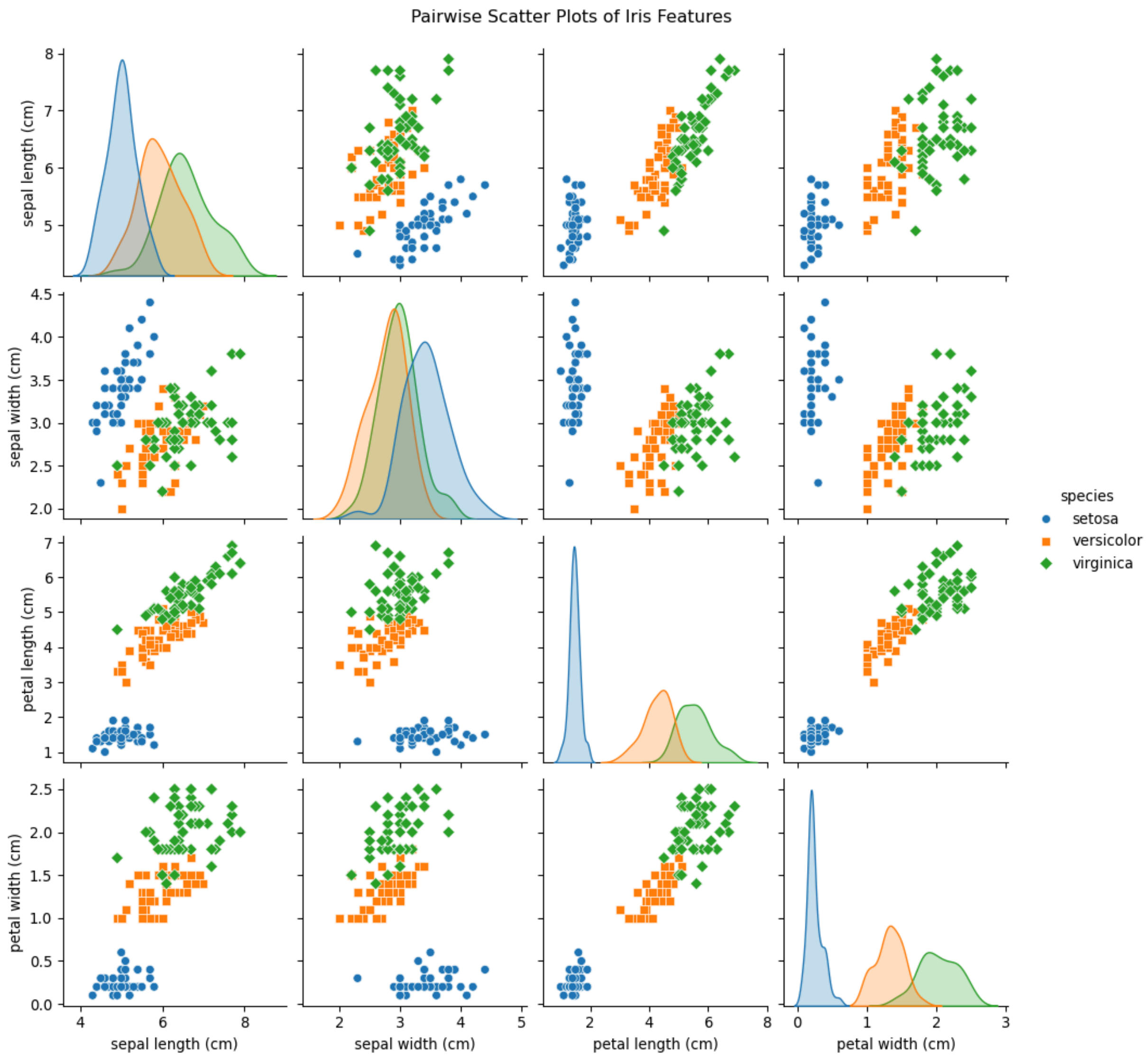
In [38]:
```
# Summary statistics

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  
```

Example: Using `Seaborn`

In [39]:
```
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"])
```

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



What insights can be drawn from examining the graphs?

The image presents all pairwise scatter plots for the iris dataset features, with the diagonal displaying histograms for each individual feature. Each dot represents an example (x), and the colors indicate the corresponding labels (y).

Let's first consider the diagonal elements:

- Is it possible to classify the examples using a single feature?
- We observe that **sepal length** or **width** alone cannot distinguish between the classes.
- However, **petal length** and **width** allow us to differentiate **setosa** from the other two varieties, although they do not separate **versicolor** and **virginica** effectively.

The class **setosa** frequently forms a distinct cluster.

Example: Training and test set

```
In [40]: 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
```

Our initial classifier, a decision tree, was constructed using the entire dataset. While it provides the **"best"** fit for the data, we cannot ascertain its **predictive accuracy** without further evaluation.

We will have a lot more to say about testing in the coming weeks.

Practice: Experiment with different values for `random_state`. What do you observe? Why do you think this occurs? Why is it important to set the `random_state` value?

Example: Creating a new classifier

```
In [41]: # Train the model
         clf = tree.DecisionTreeClassifier()
```

Example: Training the new classifier

```
In [42]: # Train the model
         clf.fit(X_train, y_train)
```

Example: Making predictions

```
In [43]: # Make predictions
         y_pred = clf.predict(X_test)
```

Example: measuring the performance

```
In [44]: 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

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

```
Accuracy: 0.90
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       0.91      0.83      0.87        12
   virginica       0.83      0.91      0.87        11

    accuracy                           0.90        30
   macro avg       0.91      0.91      0.91        30
weighted avg       0.90      0.90      0.90        30
```

Here is a discussion on model persistence for scikit-learn models.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

School of Electrical Engineering and **Computer Science** (EE**CS**)

University of Ottawa