Lecture Notes

Key Technologies (AI)

Master Wirtschftinformatik

Umberto Michelucci

Lucerne University of Applied Sciences and Arts (HSLU)

Course	Key Technologies (AI)
Term	Fall 2025
Version	3.1
Location	Lucerne, Switzerland
Contact	umberto.michelucci@hslu.ch

These notes provide a gentle, practice-oriented introduction to artificial intelligence, covering key milestones, core terminology, and typical project workflows. They are intended for students without a technical background, emphasizing clear concepts, intuition, and real-world examples.

Contents

Chapter 1

Foundations of Artificial Intelligence

Contents

Chapter Abstract

This chapter introduces core ideas and context for modern machine learning. We begin with a brief history from Turing's question "Can machines think?" through perceptrons, backpropagation, support vector machines, and the recent dominance of deep learning and generative models. We clarify the relationship between *AI*, *Machine Learning*, *Deep Learning*, and *Generative AI*, highlighting differences in scope, methods, and applications. We compare fundamental and applied research, and distinguish research, technology development, and product work with their goals, evidence, skills, and success criteria. A beginner-friendly *ML pipeline* (data collection, preparation, training, evaluation, deployment) provides a practical lens for projects. We survey representative application areas (healthcare, finance, retail, mobility, public sector, creative industries) and outline opportunities and risks. Finally, we introduce the technology life cycle and offer reflective prompts on what constitutes "intelligent behavior" and how to measure it. The emphasis is on intuition, clear concepts, and decision-oriented framing.

1.1 Brief History of Machine Learning

Machine learning appeared for the first time in the mid-20th century. Alan Turing, one of the pioneers, raised fundamental questions about the potential of machines to mimic aspects of human intelligence in his seminal paper "Computing Machinery and Intelligence" [1]. In this work, Turing endeavored to answer the question "can machines think?" and introduces the concept now known as the Turing test. This is designed to evaluate a machine's ability to exhibit intelligent behaviour that is indistinguishable from that of a human. It involves a human interrogator engaging in a conversation with both a human and a machine, both out of sight. The machine is considered to have passed the test if the interrogator cannot consistently tell it apart from the human. Throughout the paper, Turing addresses and counters various objections to the notion of machine intelligence. These objections range from theological arguments to the belief that consciousness is a prerequisite for intelligence, and even the idea that machines are incapable of errors or learning from experiences. Turing also discusses the potential of digital computers, which at the time were in their early stages of development. He highlights their ability to simulate any process of formal reasoning, acknowledging their existing limitations but predicting significant advancements in their capabilities. An intriguing part of Turing's paper is his speculation on the possibility of machines learning to evolve over time. He suggests that instead of equipping a machine with an extensive understanding of the world, it might be more effective to develop a simpler machine and allow it to learn from its interactions and experiences (that is, after all, what machine learning is about). This paper is important for its philosophical approach, as

Turing ponders over the broader implications and future possibilities of machine intelligence. His insights and hypotheses have profoundly influenced the development of artificial intelligence and continue to shape discussions about the nature of intelligence and consciousness today.

In the late 1950s, the perceptron, the first neural network, was introduced by Rosenblatt [2]. This development laid the groundwork for future research in neural networks. The perceptron model was conceptualised as a simplified mathematical abstraction of a biological neuron and was designed to perform certain types of classification. It was able to automatically learn the optimal weight coefficients, which are then multiplied with input features to determine whether a neuron fires or not, essentially making a decision based on a linear combination of its input signals. Rosenblatt's work was groundbreaking because it demonstrated how a machine could be programmed to learn from data, a concept that was relatively novel at the time. The perceptron was shown to have the ability to learn through a process of adjusting the weights based on errors made in previous predictions, an early form of what would later be known as supervised learning (see next section for an explanation of the term *supervised*). While the original perceptron had limitations, notably its inability to process data that are not linearly separable (which was later addressed by the invention of multilayer perceptrons), Rosenblatt's paper laid the first foundation for further research into neural networks. It sparked a wave of interest and optimism about the potential of machines to learn and make decisions, paving the way for the modern field of machine learning and artificial intelligence.

The book "Perceptrons" by Marvin Minsky and Seymour Papert [3], published in 1969, stands as a critical and influential work. This book is particularly notable for its analysis and criticism of the perceptron by Rosenblatt. Minsky and Papert's work provided a thorough mathematical critique of the perceptron, delving into its capabilities, and more importantly, its limitations. The authors demonstrated that perceptrons, as they were then conceived, were incapable of solving some relatively simple but fundamental problems, such as the XOR problem, which involves correctly classifying inputs that are not linearly separable. This limitation significantly restricted the range of problems to which the perceptron could be applied.

The criticism contained in "Perceptrons" was so compelling that it led to a significant reduction in interest and funding for neural network research. This period, often referred to as the first "AI Winter," was characterised by skepticism and reduced expectations for the field of artificial intelligence, particularly in the area of neural networks. However, in the long run, the rigour and depth of Minsky and Papert's analysis also laid the groundwork for later advancements. Their work highlighted the need for more complex and layered neural network architectures, leading eventually to the development of multilayer perceptrons and deep learning techniques. In this way, "Perceptrons" played a paradoxical role in the history of machine learning: while it temporarily dampened enthusiasm for neural network research, it also set the stage for some of the field's most significant breakthroughs in subsequent decades. The impact of "Perceptrons" is a testament to the importance of critical analysis and rigorous evaluation in

the advancement of scientific fields, serving as both a cautionary tale about the risks of hype and untested assumptions, and a beacon pointing the way to more robust and capable models in machine learning.

The renaissance in neural networks was sparked in the 1980s with many important new results. The decade of 1980 to 1990 was a period of significant advancement in the field of machine learning. One of the major developments was the popularisation of decision tree algorithms (first born in the 1960s) for classification tasks. The introduction of the ID3 algorithm by Quinlan in 1986 [4] marked a key moment in this area. ID3 was a revolutionary step using the concept of information entropy, a measure from information theory, to select the attribute that partitions the data in the most informative way at each branch of the tree. Following the success of ID3, further developments and iterations led to more sophisticated algorithms such as C4.5 and CART (Classification and Regression Trees), which expanded and refined the decision-making capabilities of these models. These advances included handling categorical and numerical data, dealing with missing data, and improving computational efficiency. The development of decision trees significantly impacted the field of machine learning, providing a foundation for more complex models and algorithms. Their ability to break down a complex decision-making process into a simpler form not only made them powerful tools in predictive modelling, but also contributed to the field's broader understanding of data-driven decision-making processes.

One of the major advances in the field of neural networks was the development of the backpropagation algorithm by Rumelhart, Hinton and Williams in 1986 [5], which revolutionised neural network training. Backpropagation, formally known as the *backpropagation of errors*, is a method used for artificial neural networks to calculate the weight updates contributed by each neuron after processing a batch of data. It is a cornerstone of neural network training, enabling efficient computation of gradients. Backpropagation's roots can be traced back to the early work in the 1960s and 1970s. However, it was not until the 1980s that the algorithm gained substantial recognition. The paper by Rumelhart *et al.* was crucial because it addressed a significant challenge of the time: how to adjust the weights of neurons in hidden layers of a neural network. Before this, while single-layer neural networks were used effectively, the addition of hidden layers, which allowed the modelling of more complex functions, posed a challenge in terms of adjusting the weights through learning. The backpropagation algorithm uses the chain rule from calculus to iteratively calculate gradients for each layer in the network, starting from the output layer and working backward. This method allowed for the effective training of deep neural networks by adjusting the weights in a way that minimises the loss function of the network. The introduction of backpropagation sparked renewed interest in neural networks, leading to what is often referred to as the second wave of neural network research. The algorithm's ability to train deep networks laid the groundwork for the development of deep learning, which has since revolutionised many aspects of machine learning and artificial intelligence. Despite its success, backpropagation is not without limitations, such as the vanishing gradient problem, where gradients can become increasingly small as they are propagated back

to early layers, hindering the learning of the network. Nevertheless, its development represents a pivotal moment in the history of machine learning, providing a key tool for the training of complex neural networks and enabling advances in a wide range of applications, from image and speech recognition to natural language processing.

This period also saw the emergence of the first practical applications of neural networks, such as the NetTalk system by Sejnowski and Rosenberg in 1987 [6], which demonstrated the potential of neural networks in speech synthesis. NetTalk was designed to convert written English text into phonetic pronunciations, essentially teaching a computer how to speak. At its core, NetTalk used a seven-layer neural network, a significant design for its time, considering the computational limitations and the relatively new state of neural network research. This neural network was trained using the backpropagation algorithm, which allows it to learn the appropriate phonetic pronunciation of English text by being exposed to examples of text and the corresponding spoken words. One of the remarkable aspects of NetTalk was its ability to learn and generalise from the training data. As it was exposed to more examples, the neural network improved its pronunciation skills, much like how a human child learns to speak by listening to adults. This learning process was not just about memorising specific pronunciations, but rather about understanding the underlying patterns and rules of English phonetics. The NetTalk system was instrumental in demonstrating practical applications of neural networks, particularly in the domain of speech synthesis. It showed that neural networks could learn to perform tasks without explicit programming of rules, a significant departure from the conventional approaches used in computer science at the time. Furthermore, NetTalk contributed to the growing interest in neural networks and their potential applications. It was an early example that helped pave the way for more advanced speech recognition and synthesis systems, which are now commonplace in devices and applications such as smartphones, virtual assistants, and language translation services. The development of NetTalk by Sejnowski and Rosenberg is a notable example of how innovative applications of neural networks can lead to breakthroughs in artificial intelligence, demonstrating the power of these systems to learn and adapt in ways similar to human learning.

The 1990s was also a pivotal era in the field of machine learning, marked by a transition from theoretical underpinnings to more practical applications. This period witnessed a significant evolution in algorithms driven by increasing computational power and data availability. One of the major milestones was the advancement of neural networks. The work of LeCun *et al.* [7] on handwriting recognition using convolutional neural networks laid the foundation for modern deep learning techniques. LeCun assembled a dataset that is now widely known as the MNIST (Modified National Institute of Standards and Technology) dataset. This has played a pivotal role in the field of machine learning, particularly in the development and benchmarking of algorithms for handwritten digit recognition. The dataset was created as a more accessible and pre-processed version of the earlier NIST dataset. It consists of 70,000 labelled images of handwritten digits (0 through 9), divided into a training set of

60,000 examples and a test set of 10,000 examples. Each image is a 28x28 pixel grayscale representation of a digit, making it ideal for testing image processing systems. The simplicity and size of the MNIST dataset have made it a standard benchmark for algorithms in image recognition, serving as a testbed for a wide range of approaches from classical machine learning to deep learning. In particular, the development and success of convolutional neural networks were significantly bolstered by experiments on the MNIST dataset, as demonstrated by LeCun *et al.* [8]. Despite its simplicity, MNIST continues to be a valuable resource for educational purposes and initial algorithm testing.

This period also saw the rise of Support Vector Machines (SVMs), introduced in their modern form by Cores and Vapnik in 1995 [9], which provided a robust method for classification and regression tasks. SVMs are a set of supervised learning methods that are used for classification, regression, and outlier detection. The origins of SVMs can be traced back to the work of Vapnik *et al.* from 1964 [10], where the concept of a linear classifier with maximum margin was first introduced. This early model laid the groundwork for what would eventually become modern SVMs. The formal development of SVMs began in the 1990s. The key paper by Boser *et al.* [11], introduced a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. This was further refined and popularised by Cores and Vapnik [9], who provided a comprehensive framework for training SVMs. Since then, SVMs have seen various enhancements and have been applied in numerous fields, from image recognition to bioinformatics. Their ability to handle large feature spaces and their flexibility in modelling diverse sources of data make them a powerful tool in machine learning. SVMs represent a significant development in the field of statistical learning, offering a robust approach to both classification and regression problems. Their evolution from a theoretical concept to a staple in machine learning toolkits is a testament to their versatility and effectiveness.

The 21st century has been particularly marked by the rise of deep learning, a subset of machine learning based on neural networks with a large number of layers and parameters. Pioneering work by researchers such as Yann LeCun, Geoffrey Hinton, and Yoshua Bengio [12] has led to breakthroughs in fields such as image and speech recognition and natural language processing. But the early 2000s also marked a period of significant advancements and milestones in many fields of machine learning, setting the stage for the rapid development that would follow in subsequent years. One of the most impactful changes at the beginning of the 2000s was the dramatic increase in computational power, largely due to the advancement of GPUs (Graphic Processing Units). This, coupled with the growing availability of large datasets, allowed researchers to train more complex models, particularly deep neural networks. The early 2000s also witnessed significant progress in natural language processing (NLP). The introduction of statistical methods over traditional rule-based approaches marked a paradigm shift in NLP. Seminal models like latent Dirichlet allocation (LDA) by Blei *et al.* [13] began to emerge, changing the way machines processed and understood human language. This period also saw the rise of ensemble methods in machine learning. Techniques such as Random Forests and boosting methods became popular

due to their robustness and effectiveness in various applications. Boosting, a powerful ensemble technique, involves combining multiple weak learners to form a strong classifier. A pivotal development in boosting was the AdaBoost algorithm, introduced by Freund and Schapire [14]. AdaBoost, short for Adaptive Boosting, works by sequentially adding weak learners, typically decision trees, and focusing on the instances that were incorrectly predicted in previous rounds. The algorithm assigns higher weights to these challenging instances, ensuring that subsequent learners focus more on them. AdaBoost demonstrated remarkable effectiveness in improving the accuracy of classification models. Another critical advance was the formulation of Gradient Boosting Machines (GBMs) by Friedman [15]. GBMs extend the boosting framework by optimising arbitrary loss functions. This method involves building a model in a stage-wise fashion and generalising them by allowing optimisation of an arbitrary differentiable loss function. GBMs have been highly successful in a wide range of practical applications, from standard regression tasks to complex learning problems.

Parallel to these developments, reinforcement learning has evolved, with significant contributions by Sutton and Barto [16]. This area of machine learning, focusing on how agents should take actions in an environment to maximise some kind of reward, has led to impressive displays such as AlphaGo's victories in the game of Go [17]. A foundational concept in reinforcement learning is temporal difference (TD) learning, introduced earlier by Sutton [18]. In the early 2000s, the refinement and application of TD learning methods, particularly TD-Gammon by Tesauro [19], demonstrated the potential of RL in complex domains. TD-Gammon, a computer backgammon program, used a neural network trained through TD learning and was able to achieve a performance level comparable to that of human experts. The beginning of the 2000s also witnessed significant advancements in policy gradient methods, an approach in RL where the policy, the core decision-making function of an agent, is directly optimised. The work by Sutton *et al.* [20] provided a comprehensive framework for these methods, which became crucial for solving tasks that require complex action sequences. The twentieth century also saw important theoretical advances in RL. The development of algorithms with guaranteed convergence and the exploration of the balance between exploration and exploitation were key focus areas. The formalisation of the exploration-exploitation trade-off, as discussed by Auer *et al.* [21], provided a deeper understanding of the decision-making process in RL. The development of machine learning algorithms capable of playing the ancient board game Go marks a significant milestone in the history of artificial intelligence (AI) and machine learning. Go, known for its deep strategic complexity, has long been a formidable challenge for AI researchers. Traditional AI approaches that had succeeded in chess, such as brute-force search, were ineffective for Go due to the game's vast number of possible positions. The breakthrough came with AlphaGo in 2016, developed by Google DeepMind. In the landmark paper by Silver *et al.* [17], the AlphaGo algorithm was introduced. It combined deep neural networks with Monte Carlo Tree Search (MCTS) to evaluate board positions and select moves. This approach allowed AlphaGo to defeat a professional human Go player in 2015, a feat previously thought to be

at least a decade away. Following this, AlphaGo Zero, as detailed by Silver *et al.* [22], represented a further advancement. It learnt to play Go solely by playing games against itself, without any human data, achieving superhuman performance. This was a significant step in unsupervised learning within machine learning. The success of AlphaGo and its successors has had a profound impact on the field of machine learning. It not only demonstrated the potential of deep learning and reinforcement learning in solving complex problems, but also inspired numerous applications of similar techniques in various domains. The development of AlphaGo and its successors marks a pivotal point in the history of machine learning, showcasing the power of integrating deep neural networks with advanced search strategies. This milestone in AI research has broadened the horizons for the application of machine learning in complex decision-making scenarios.

The decade 2010 to 2020 was marked by remarkable breakthroughs in the field of machine learning. One of the most significant milestones was the dominance of deep learning. The pivotal moment occurred in 2012 when [23] demonstrated the power of deep convolutional neural networks (CNNs) by winning the ImageNet competition. This breakthrough drastically improved the performance of image classification tasks and sparked a wave of research and applications in deep learning across various domains. In natural language processing (NLP), the development of transformer models, particularly the introduction of BERT (Bidirectional Encoder Representations from Transformers) by [24], marked a significant advance. This model and its architecture revolutionised the approach to tasks such as text classification, sentiment analysis, and question-response, setting new standards for performance in NLP. Another notable development was in generative models, particularly with the introduction of Generative Pretrained Transformer (GPT) models by OpenAI. As described by [25], these models showed an unprecedented ability in generating coherent and contextually relevant text.

The early 21st century saw a shift towards more complex models with the introduction of neural networks in language tasks. A milestone was the development of sequence-to-sequence learning models [26], which improved the performance of machine translation systems. A pivotal moment in the history of large language models (LLMs) was the introduction of the Transformer model by Vaswani et al. in 2017 [27], which led to a paradigm shift in the way language models were designed. This model, based on self-attention mechanisms, significantly improved the efficiency and effectiveness of language understanding and generation. Transformers have revolutionised the field of natural language processing (NLP) and beyond, offering a new paradigm for handling sequential data. Unlike previous models that processed sequences step by step, transformers process entire sequences simultaneously, leading to significant gains in efficiency and effectiveness. The core innovation of transformers is the attention mechanism. This mechanism allows the model to focus on different parts of the input sequence when predicting each part of the output sequence. In simple terms, the attention mechanism lets the model dynamically pay attention to the most relevant parts of the input to make predictions, akin to how

humans focus on specific words or phrases when comprehending a sentence. A transformer model consists of an encoder and a decoder. The encoder reads the input sequence and generates a high-dimensional representation, which the decoder then uses to generate the output sequence. Both the encoder and decoder are composed of layers that include attention mechanisms and feedforward neural networks. The encoder's job is to process the input data (like a sentence in NLP tasks) and create a context-rich representation. Each layer in the encoder consists of two main parts: a self-attention mechanism and a feed-forward neural network. The self-attention mechanism allows each position in the encoder to attend to all positions in the previous layer of the encoder. The decoder, similar in structure to the encoder, generates the output sequence. It also contains a self-attention mechanism, but includes an additional attention layer that focusses on the encoder's output. This design enables the decoder to consider the entire input sequence when producing each element of the output. Transformers offer parallel processing of sequences, which significantly speeds up training and inference times. They have achieved state-of-the-art results in various NLP tasks, including translation, text summarisation, and question-answering. Their architecture has also inspired adaptations in other domains, such as computer vision. Building on the Transformer architecture, OpenAI introduced GPT (Generative Pretrained Transformer) and its subsequent iterations, which demonstrated remarkable language generation capabilities [25]. Similarly, Google's BERT (Bidirectional Encoder Representations from Transformers) model [24] brought advances in language understanding, especially in tasks like question answering and sentiment analysis. The current landscape of LLMs is characterised by their increasing size and sophistication, with models like GPT-3 and GPT-4 pushing the boundaries of what is possible in natural language processing and generation [28].

One of the most striking features of modern machine learning models, particularly in the domain of natural language processing, is their unprecedented size. Large Language Models (LLMs) have grown to colossal scales with a staggering number of parameters that defy conventional expectations. These parameters represent the learnt knowledge and patterns within the model. The scale of LLMs is often measured in billions or even trillions of parameters. For instance, models such as GPT-3 by OpenAI boast 175 billion parameters, dwarfing their predecessors by orders of magnitude. The growth of LLMs over the past few years has followed an exponential trajectory. From models with millions of parameters to those with billions and beyond, this rapid expansion has pushed the boundaries of what was previously thought possible. Each new iteration of LLMs pushes the envelope further, breaking records in terms of both performance and parameter count. Training and fine-tuning these gigantic models require an immense amount of computational resources. Training a model with trillions of parameters requires not only powerful GPUs or TPUs (Tensor Processing Units) but also extensive memory capacity and high-speed network connections. This computational demand has led to the development of specialised hardware and distributed training setups. Despite their enormous size, LLMs have found applications in a wide range of domains. They excel in natural

language understanding and generation tasks, including translation, summarisation, question answering, and content generation. Their immense knowledge base allows them to generate coherent and contextually relevant text across diverse topics.

I would be guilty of neglecting important parts of the history of machine learning if I did not at least mention something about generative AI. Probably the most important contribution to generative AI was the introduction of Generative Adversarial Networks (GANs) by Goodfellow *et al.* [29], which marked a significant leap in the field [29]. GANs revolutionised the way machines could generate realistic images, leading to a surge in research and applications of generative models. The development of variational autoencoders (VAE) by Kingma *et al.* [30], published in 2013, explored a statistical method, offering a new approach to generative modelling [30]. These milestones collectively represent the rapid and ongoing evolution of generative AI, charting the path from simple pattern recognition to the creation of rich, complex and diverse outputs that can mimic human creativity.

Generative AI has become an important tool in synthetic data generation within scientific research. This approach enables scientists to create large, realistic datasets that can be used to train machine learning models, especially in fields where real data are scarce or difficult to obtain. By generating high-quality synthetic data that closely mirror real-world conditions, generative AI can facilitate more robust and extensive experimentation, significantly advancing research in areas like healthcare, environmental science, and materials engineering.

1.2 Machine Learning in Science

It is necessary to briefly highlight some of the major achievements of machine learning in various scientific disciplines.

In chemistry, ML has revolutionised several areas, notably drug discovery and material science. The traditional drug discovery process is often long and expensive. ML algorithms have significantly accelerated this process by predicting the properties and activities of molecules. For example, ML models can quickly screen vast chemical spaces for potential drug candidates, a process highlighted in the work of Gomez *et al.* [31]. Furthermore, ML has been instrumental in materials science for predicting the properties of new materials, thereby guiding experimental efforts in a more focused and efficient manner. Physics, especially high-energy physics and astrophysics, has benefited immensely from ML. In high-energy physics, ML methods have been crucial in analysing data from particle accelerators. The detection of the Higgs boson, as explained in [32], is a prime example in which deep learning significantly improved the separation of signals from background noise. In astrophysics, ML assists in interpreting vast amounts of data from telescopes and space missions, aiding in the discovery of new celestial objects and phenomena. In biology, perhaps the most notable contribution of ML is in the field of genomics and proteomics. Protein structure

prediction, a long-standing challenge in biology, was revolutionised by the AlphaFold system developed by Google DeepMind [33]. The ability of this system to predict protein structures accurately and quickly has profound implications for understanding biological processes and designing new therapeutics. Furthermore, ML is increasingly used in genomics to understand genetic variations and their links to diseases. The integration of machine learning into scientific research not only has accelerated discoveries, but also has opened new avenues for exploration and innovation. As ML technology continues to evolve, its role in the advancement of scientific knowledge is expected to grow even further.

1.3 Types of Machine Learning

Now we need to focus on some terminology that is widely used in the machine learning community. The field of machine learning is vast, and there are many *types* of algorithms. It is important to shed some light on the various types to clarify what you will find here. A very broad and rough classification of ML approaches is given below.

- **Supervised Learning:** This method, fundamental to machine learning, involves training an algorithm on a labelled dataset, where each example is paired with some expected output. The model learns to predict outcomes based on these data, using examples to infer patterns.
- **Unsupervised Learning:** Unlike supervised learning, unsupervised learning involves training algorithms on data without predefined labels. It focusses on identifying hidden structures and patterns. Historically significant for its role in clustering and association problems, unsupervised learning is key in exploratory data analysis and dimensionality reduction techniques like Principal Component Analysis (PCA).
- **Semi-Supervised Learning:** This approach combines elements of supervised and unsupervised learning. It is particularly useful when labelled data are limited or costly to obtain, a common challenge in machine learning. By utilising a mix of a small amount of labelled data and a larger pool of unlabelled data, models in semi-supervised learning can achieve higher accuracy than unsupervised methods alone.
- **Reinforcement Learning:** This type of learning is distinguished by its focus on making sequences of decisions. The algorithm, often called an agent, learns to achieve a goal in an uncertain and potentially complex environment. Historically, reinforcement learning has roots in game theory and optimal control theory.

1.4 AI, Machine Learning, and Deep Learning

When people talk about Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL), they often use the terms interchangeably. However,

they describe different concepts with varying scopes and levels of abstraction. Understanding these distinctions is essential to correctly position a project, manage expectations, and communicate with both technical and non-technical stakeholders.

Artificial Intelligence (AI)

Scope	Includes reasoning, planning, decision-making, perception, natural language understanding, robotics, and learning.
Techniques	Encompasses symbolic AI (rule-based systems, expert systems), search algorithms, optimisation, and more modern approaches such as machine learning and reinforcement learning.
Examples	Chess-playing programs like Deep Blue, autonomous driving systems, and conversational agents.
Key Question	"Can we build machines that behave intelligently?"

Table 1.1: Overview of Artificial Intelligence: scope, techniques, examples, and guiding question.

Artificial Intelligence is the broadest term. It refers to the field of computer science that aims to create machines or systems capable of performing tasks that normally require human intelligence. In Table 1.1 you can see the main characteristics.

Machine Learning (ML)

Scope	Concerned with algorithms that can generalize from data, including supervised, unsupervised, and reinforcement learning.
Techniques	Linear regression, decision trees, support vector machines, clustering methods, ensemble methods, etc.
Examples	Spam email classification, credit card fraud detection, personalized recommendation engines.
Key Question	"Can the system learn to improve its performance on a task from data?"

Table 1.2: Overview of Machine Learning: scope, techniques, examples, and guiding question.

Machine Learning is a subset of AI that focuses specifically on algorithms that improve automatically through experience, usually by learning from data.

Instead of explicitly programming rules, ML systems discover patterns and make predictions. In Table 1.2 you can see the main characteristics.

Deep Learning (DL)

Scope	Focused on artificial neural networks with many layers (deep architectures) that can learn hierarchical representations.
Techniques	Convolutional neural networks (CNNs) for images, recurrent and transformer architectures for sequences, generative adversarial networks (GANs), and large language models (LLMs).
Examples	Image classification in computer vision, speech recognition, machine translation, ChatGPT and other large-scale generative AI systems.
Key Question	"Can we train large neural networks on massive amounts of data to solve tasks that were previously unsolvable?"

Table 1.3: Overview of Deep Learning: scope, techniques, examples, and guiding question.

Deep Learning is a specialized branch of machine learning that uses multilayer neural networks (so-called "deep" architectures) to automatically extract features and representations from data. In Table 1.3 you can see the main characteristics.

Generative AI (GenAI)

Generative Artificial Intelligence (GenAI) is a branch of artificial intelligence that not only analyzes data but also *creates new content* such as text, images, audio, or even computer code. In Table 1.4 you can see the main characteristics.

Relationship Between the Three Concepts

Artificial Intelligence (AI) is the broad umbrella term that encompasses all techniques and approaches designed to enable computers to simulate aspects of human intelligence. This includes reasoning, problem solving, perception, and decision-making. AI is not tied to a single method, but rather refers to the overarching goal of creating systems that can perform tasks traditionally associated with human cognition. Note that sometime even rule-based systems are called "AI". So this is a rather generic term and its meaning varies wildly from person to person.

Scope	Systems that learn the underlying patterns and structures of data in order to generate new, realistic outputs.
Techniques	Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), diffusion models for image synthesis, and Large Language Models (LLMs) for text generation.
Examples	Text generators such as ChatGPT, image generators like DALL-E and Stable Diffusion, music and speech synthesis systems, as well as applications in molecule design and drug discovery.
Key Question	"Can machines generate creative content that is of such high quality that it complements or even substitutes human-created work?"

Table 1.4: Overview of Generative AI: scope, techniques, examples, and guiding question.

Within this broad field, **Machine Learning (ML)** represents a major subset. Intuitively ML is a set of algorithms that learn from data rather than relying solely on explicitly programmed rules. Instead of manually encoding every possible scenario, machine learning systems infer patterns, and make predictions or decisions based on experience learned from examples (remember the supervised learning setting we discussed in Section 1.3). Classical examples include spam filtering, fraud detection, or recommendation systems, where large amounts of past data are used to generalise to new unseen cases.

A further subset is **Deep Learning (DL)**, which refers to methods based on deep neural networks with many layers (you can imagine a neural network with many layers as algorithms that are extremely flexible and thus can understand many small differences in data). These models excel at learning hierarchical representations of data, making them particularly effective for complex tasks such as image recognition, natural language understanding, and speech processing. Deep learning is the basis for recent advances in AI, powering applications such as self-driving cars, virtual assistants, and large language models such as chatGPT. Although DL is not the only approach in ML, it is currently the most powerful and widely used technique when dealing with high-dimensional, large-scale data.

A subset of deep learning is **Generative Artificial Intelligence** (Generative AI, or GenAI), which refers to methods that are designed not only to analyze data but also to create new content such as text, images, music, or even computer code. These models learn the underlying patterns and structures of training data and then generate outputs that are often indistinguishable from human-created content. Generative AI excels at tasks where it is required to create new material, making it particularly effective for applications such as automatic text generation, realistic image synthesis, or the creation of new molec-

ular structures in drug discovery. The most prominent examples of GenAI today include large language models (LLMs) like ChatGPT, image generators such as DALL-E or Stable Diffusion, and multimodal systems that combine text, vision, and audio.

Illustrative Analogy

- Think of AI as the entire discipline of building "intelligent machines."
- Machine Learning is a set of algorithms inside AI.
- Deep Learning is a very advanced and currently dominant technique inside Machine Learning based on large and complex neural networks.
- Generative AI is a recent and fast-growing branch of AI that focusses on creating new content (text, images, audio, code) rather than only analysing existing data.

AI, ML, DL, and GenAI differ in scope, methodology, and applications. Although deep learning currently dominates the public perception of AI due to breakthroughs in natural language processing, computer vision, and generative models, AI as a whole is much broader. Generative AI, in particular, has expanded the boundaries of what AI can do by enabling creative and synthetic tasks that were once thought to be uniquely human. To succeed in practice, organisations must recognise where their project is on this spectrum and adopt the right methods accordingly.

1.5 Fundamental and Applied Research

Table 1.5 compares **fundamental research** and **applied research**, emphasising that although both follow systematic and rigorous investigation, their goals and results are very different. Fundamental research is primarily concerned with truth seeking: It aims to answer precise scientific questions, test hypotheses, and develop generalisable theories or methods. Its success is measured in terms of reproducibility, statistical validity, and contribution to the larger body of knowledge. Typical outputs are scientific publications, datasets, or theoretical insights rather than immediate solutions for real-world stakeholders.

Applied research, on the other hand, is problem-driven. It is designed to generate actionable evidence for decision-making in a specific, real-world context. Instead of broad generalization, applied research focuses on producing prototypes, pilots, or validated interventions that demonstrate value for stakeholders. Evidence is evaluated in terms of cost–benefit, risks, and impact on concrete outcomes rather than universal validity. While fundamental research prioritizes understanding the *why*, applied research emphasizes demonstrating *what works* in practice.

The key message of the table is that both approaches are essential but serve different purposes. Confusing one for the other often leads to frustration: expecting general theory from applied projects, or demanding immediate business value from fundamental studies. Successful organizations and teams recognize the distinction and strategically balance both types of research, using fundamental work to expand the frontier of knowledge, and applied work to translate those insights into tangible impact.

Table 1.5: Comparison of Fundamental Research and Applied Research

Aspect	**Fundamental Research**	**Applied Research**
Definition	A systematic investigation to answer a precise question and produce generalizable knowledge.	Systematic investigation that generates actionable evidence; to enable specific real-world decisions.
Purpose	Explain/understand (the *why*), not just build; truth-seeking over utility.	Inform a concrete real-world decision for a specific stakeholder in a defined context.
Hallmarks	Clear research question, testable hypothesis, defined method, controls/baselines, transparency, reproducibility, peer review.	Credible, context-relevant causal effect on outcomes that matter, shown by a prototype or proof-of-concept.
Evidence	Statistical validity (effect sizes, confidence intervals), plus external validity/generalization.	Short pilot with learnings, cost–benefit analysis, and documented risks.
Typical Outputs	Papers/reports, code & data, theoretical or empirical results.	Field evaluation, decision guidance, validated intervention/prototype (often a pilot, not a full product).
Not	Proving a prototype meets a spec (technology development) or shipping a feature to move a KPI (product).	Broad general theory or an almost-production-ready solution; results don't guarantee transfer beyond the tested context.

1.6 Research, Technology Development, and Product Development

Artificial intelligence projects can involve very different types of work, which must be clearly distinguished in order to avoid confusion in expectations, plan-

ning, and outcomes. Generally, we can broadly separate three domains: **Research**, **Technology Development**, and **Product Development**, with the last two falling under the umbrella of applied research.

Research

Research focuses on creating new knowledge and answering fundamental questions such as: *"Can this be done?"* and *"Why does it work?"*.

- Typical example: inventing a new type of neural network architecture or proving a new theorem in mathematics.
- Usually carried out in academia or corporate research labs.
- Output: papers, algorithms, theoretical frameworks.

Technology Development

Technology Development takes research ideas and turns them into usable prototypes or practical implementations. The key question is: *"How can we make this work in practice?"*.

- Typical example: implementing a prototype of the new neural network and making it efficient, scalable, and applicable to a specific use case.
- Often performed in industrial research and development teams.
- Output: prototypes, proof-of-concept systems, optimized algorithms.

Product Development

Product Development focuses on delivering value to end users and businesses. The main guiding question is: *"Does this solve a real problem for our users?"*.

- Typical example: integrate the new neural network into a mobile application or SaaS platform as a concrete feature.
- Orientated toward business goals, user experience, and scalability.
- Output: deployed products, customer-facing services, measurable business impact.

These three domains require different skills, success criteria, and life cycles. Confusing one for the other leads to frustration and failed projects. Successful adoption of AI requires clarity: decide deliberately whether your project is research, technology development, or product development.

In Table 1.6 you can see a comparison between research, technology, and product development with the main aspects. Notable and relevant are the following aspects.

- **Time Horizon**: it goes from 3-5 years in research to 6 months-1 year in product development. Important to note is that even for developing a product, you should never expect to have something in less than 6-12 months. That is an optimistic estimate.
- **Validation**: it is important to note that the way you validate your project (in other words how you decide if your project was successful or not) is very different for the three types of projects. For companies *users testing* and *market fit* are the most important ways (although surely not the only ones).
- **Skills Needed**: it is interesting to note how for research a more academic skillset (focused on statistics, computer science, mathematics) is required, while for product development a more *engineering* (focused on business and IT infrastructure) skillset is much more important, and a more academic skillset almost irrelevant (although in many cases still useful).

Exercise

Exercise 1.6.1 (Classifying AI Work)
Time: 15 minutes **Group size:** 2–3 students
Task: Read the short project descriptions below. For each, decide whether it belongs to **Research**, **Technology Development**, or **Product Development**. Write down your choice and a one-sentence justification.

1) A university team designs a new optimization algorithm and proves it converges under certain conditions.
2) An R&D group implements the algorithm efficiently in TensorFlow and benchmarks it against existing optimizers.
3) A startup integrates the algorithm into their SaaS platform so customers can train models faster.
4) A PhD student tests whether large language models show "theory of mind" through experiments.
5) A company adapts an open-source speech-to-text model to run offline on mobile devices.
6) A product manager leads a team to build a voice-controlled meeting assistant that integrates speech-to-text with calendars.

Deliverable (ca. 5 min): Compare answers across groups and discuss where there was disagreement.

1.7 Introduction to the Technology Life Cycle

When we talk about the *technology life cycle*, we mean the typical journey that every new technology goes through from its first appearance to its eventual decline. Understanding this cycle helps us see why technologies rise, how they spread, and why they are eventually replaced.

At the **introduction stage**, a technology first emerges. This might be an early prototype, a pilot project, or a limited release. The focus here is on showing that the idea is feasible and that it solves a real problem, usually in a narrow but valuable use case. The user base is small, costs are high, and the technology changes quickly as developers learn and improve. The main risk at this stage is to overpromise or fail to demonstrate a clear value.

If the technology succeeds, it enters the **growth stage**. Adoption begins to accelerate, competitors enter the market, and demand increases. The focus shifts to reliability and repeatable processes, ensuring that users can trust the technology and that it can be delivered consistently. The key indicators for this stage are better performance metrics, more structured pricing, and improved support and documentation. But growth can also be painful: scaling creates new technical and organisational challenges, and there is pressure to add features too quickly.

Eventually, the technology reaches **maturity**. By this point, it has become mainstream, and the market begins to saturate. The focus now is on efficiency, integration into daily workflows and compliance with standards. Ecosystems are formed around technology, with partnerships and established best practices. Differentiation no longer comes mainly from core features, but from user experience and service quality. The danger at this stage is complacency, companies risk falling behind if they stop innovating.

Finally, every technology faces **decline**. Newer and more effective solutions appear, and the demand for old technology decreases. At this point, organizations must decide whether to phase out the technology (sunset) or transform it into something new (renewal). Signals of decline include shrinking market share, reduced investment, and a shift into maintenance-only mode. If end-of-life is poorly managed, risks include technical debt, stranded users, and loss of trust.

Exercise

Exercise 1.7.2 (Scoping an AI Project)
Time: 20 minutes **Group size:** 3–4 students
Task: Choose *one* realistic use-case (e.g., radiology triage, fraud detection, demand forecasting, tutoring chatbot).
In 15 minutes, discuss the following points:

1) **Stage & Type:** Life-cycle stage (Intro/Growth/Maturity/Decline) and project category (Research/Tech Dev/Product). One-sentence justification.

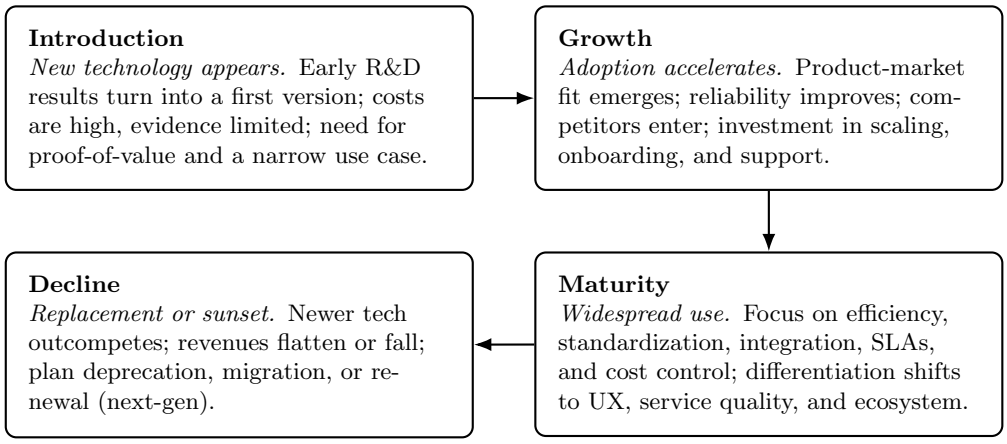

Figure 1.1: Technology life cycle (Introduction → Growth → Maturity → Decline).

2) **Method fit:** Pick the ML family (supervised/unsupervised/RL/-GenAI) and say *why* it fits (data, labels, etc.).

3) **Success & Evidence:** One primary success metric (task/KPI) *and* one generalization/robustness check you will require before shipping an hypothetical product.

4) **Risks & Guardrails:** One ethical/data-protection risk and one mitigation (privacy-by-design, fairness audit, human oversight).

5) **Cost/Benefit sketch:** Name one cost driver (compute, labeling, integration) and one benefit pathway (quality, safety, time, revenue).

Deliverable (ca. 5 min):A brief discussion in the class.

1.8 Machine Learning Pipeline

Machine learning may sound complex, but at its heart it follows a simple logic: we give the computer *examples*, the computer *learns patterns* from them, and then it can *make predictions* on new cases. To organize this process, it is useful to think in terms of a **pipeline**. A pipeline is just a sequence of steps that take us from raw data to useful insights or predictions.

1.8.1 Data Collection

Every ML project starts with **data**. This could be medical images, customer transactions, sensor readings, or text documents. The quality and quantity of the data are crucial, because the computer can only learn from what we provide.

2. Data Preparation

Raw data is often messy: it may have missing values, inconsistent formats, or irrelevant parts. In this step, we *clean* the data and organize it so that the computer can work with it. Sometimes we also transform it into a suitable format (for example, turning dates into numbers or resizing images).

1.8.2 Model Training

The **model** is the computer program that tries to learn the patterns in the data. Training a model means showing it many examples and adjusting its internal rules so that it improves step by step. For instance, if the task is to distinguish cats from dogs in photos, the model adjusts itself until it gets better at telling them apart.

1.8.3 Evaluation

Once trained, we need to check whether the model is reliable. We test it on data it has never seen before and measure how well it performs. If the performance is poor, we may need more or better data, or we may need to adjust the way we train.

1.8.4 Deployment and Use

When the model works well enough, it can be put into use: embedded in a phone app, integrated into hospital workflows, or used by analysts to support decisions. This stage is called **deployment**. From then on, the model can process new inputs and provide predictions automatically.

In summary, a machine learning pipeline typically follows the path (see Figure 1.2):

Data Collection $\rightarrow$ Preparation $\rightarrow$ Training $\rightarrow$ Evaluation $\rightarrow$ Deployment.

Understanding this flow helps us to see that ML is not “magic”, but a structured process with clear steps and responsibilities.

1.9 Application Areas of AI

Artificial Intelligence has found widespread application across business and society. Below are some representative use cases that illustrate its impact and diversity.

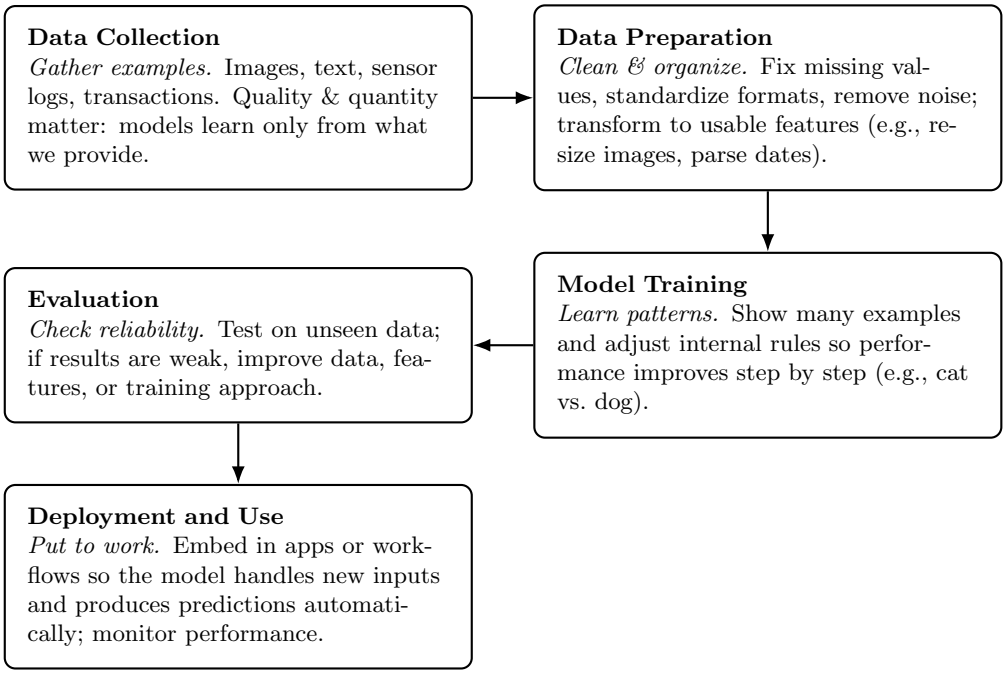

Figure 1.2: A simple machine learning pipeline: Data Collection → Data Preparation → Model Training → Evaluation → Deployment and Use.

Healthcare and Medicine

AI is transforming the healthcare sector in diagnostics, personalized medicine, and hospital operations.

- **Medical Imaging:** Deep learning models can detect tumors, fractures, or anomalies in radiology scans with accuracy comparable to or even surpassing human specialists.
- **Drug Discovery:** AI accelerates the search for new pharmaceuticals by predicting molecule interactions, drastically reducing time and cost.
- **Operational Efficiency:** Predictive analytics help hospitals manage patient flows, optimize resource allocation, and reduce waiting times.

Finance and Banking

The financial sector has been one of the earliest adopters of AI, primarily due to its reliance on data-driven decision making.

- **Fraud Detection:** Machine learning systems monitor millions of transactions in real time, detecting suspicious patterns that may indicate fraud.

- **Algorithmic Trading:** AI-driven trading algorithms adapt quickly to market conditions and execute trades at high frequency.
- **Credit Scoring:** Beyond traditional metrics, AI models assess creditworthiness using alternative data, enabling financial inclusion.

Retail and E-Commerce

In retail, AI enhances both customer experience and backend operations.

- **Recommendation Systems:** Platforms like Amazon or Netflix use collaborative filtering and deep learning to personalize product or content recommendations.
- **Demand Forecasting:** Predictive models anticipate customer demand, allowing retailers to optimize inventory and supply chains.
- **Chatbots and Virtual Assistants:** Natural language processing enables customer support systems that operate 24/7 with quick and relevant responses.

Transportation and Mobility

AI is at the core of the future of smart mobility.

- **Autonomous Vehicles:** Self-driving cars use AI for perception, planning, and control, aiming to increase safety and efficiency.
- **Traffic Management:** Predictive models optimize traffic lights and urban mobility patterns to reduce congestion.
- **Logistics Optimization:** Companies like UPS or DHL use AI to dynamically optimize delivery routes, saving fuel and improving service.

Public Sector and Society

Governments and public organizations are leveraging AI to improve decision-making and public services.

- **Smart Cities:** AI enables predictive maintenance of infrastructure, intelligent energy management, and safer public spaces through surveillance systems.
- **Education:** Intelligent tutoring systems personalize learning experiences for students, identifying strengths and weaknesses.
- **Crisis Response:** AI models are used in disaster management, for example predicting the spread of wildfires or floods.

Creative Industries

Beyond traditional sectors, AI is increasingly shaping creative domains.

- **Generative Art and Music:** AI systems such as generative adversarial networks (GANs) and large language models (LLMs) create paintings, music, or literature.
- **Media Production:** AI assists in video editing, dubbing, and creating realistic digital actors.
- **Marketing:** Natural language generation tools produce tailored advertising copy and social media content.

These examples highlight the breadth of AI applications. While the technical methods often overlap, the societal implications, regulatory requirements, and business models vary significantly across domains.

Exercise

Exercise 1.9.3
Goal: Think about 2-3 situation in your daily life where AI is present.

Task (ca. 15 min): Work in small groups of 3–4 students. Discuss the following questions and write down short notes:

- Where do you encounter technology that seems to make decisions or predictions for you?
- How can you tell that some form of AI is involved (and not just a simple rule or program)?
- What changes in your daily life because of this AI system?

1.10 Five main concepts and when to use them

To structure your work in projects you will need five main concepts: research questions, hypotheses, goals, deliverabls and metrics. Depending on the type of work you do (fundamental research, techology or product development) some are more important than others. The comparison in Table 1.7 highlights the five concepts and give an overview on when to use them. The ones marked with **primary** are the most fundamental for a specific project type, while the one marked **secondary** are useful, but not mandatory.

In research, the primary focus is on formulating precise questions and testable hypotheses in order to advance knowledge. In technology development, the emphasis shifts toward proving feasibility by meeting technical specifications

through prototypes and benchmarks. In product development, the central concern is to create business or user value, where goals, deliverables, and metrics are tied to concrete key performance indicators and customer outcomes.

The main message is therefore that while the vocabulary may look the same, its meaning and priority change fundamentally across contexts. Recognizing these differences helps teams set realistic expectations, align success criteria, and avoid confusion when moving from research to engineering to products.

Exercise

Exercise 1.10.4

Goal: Understand how the five concepts (research questions, hypotheses, goals, deliverables, and metrics) play different roles depending on project type.

Task: Work in small groups of 3–4 students. Discuss the following and write short notes:

1. Choose one example of a project in each category: *fundamental research*, *technology development*, and *product development*.
2. For each example, decide which of the five concepts are **primary** and which are **secondary**.
3. Compare across the three project types: What changes? What stays the same?
4. Reflect: Why do you think it is important to adapt the meaning and priority of these concepts to the type of project?

1.11 Additional Exercises

Exercise

Exercise 1.5 (Create a Blueprint for an AI Project)

Time: 20 minutes **Group size:** 3–4 students

Task: Produce a concise, decision-ready blueprint for an AI project that could be executed in your institution/company. The blueprint should make *scope, value, feasibility, and risks* explicit.

Choose one use case (e.g., radiology triage, demand forecasting, tutoring assistant, predictive maintenance, document summarization, fraud detection).

Deliverable: Fill a blueprint with the points below. Be concrete and

brief (bullet points; max 3 lines per box). You can use a sheet of paper, the whiteboard, etc.

1) **Problem & Stakeholders**
   Who is the user/customer? What decision/action will this system inform or automate?

2) **Success Metric & Business Value**
   One *primary* metric (task/KPI) and *why it matters*.

3) **Method Choice**
   Pick one family (supervised / unsupervised / RL / GenAI) and justify in one sentence.

4) **Risks & Guardrails**
   One ethical/fairness risk, one privacy risk, and one operational risk.

5) **System Sketch**
   Simple box-and-arrow: data ingestion → training → eval → deployment → monitoring.

6) **Cost & Timeline**
   Top cost drivers (labeling, compute, integration) and a rough timeline (e.g., 4–6–10 weeks) across phases.

7) **Lifecycle Stage & Type**
   Where are we in the Technology Life Cycle (Intro/Growth/Maturity/Decline)? Is this Research / Tech Dev / Product? One-line justification.

Tip: Scope tightly. Prefer one user, one decision, one primary metric, one initial data source.

Exercise

Exercise 1.6 (What is intelligence?)
Goal: To recognize that defining intelligence is not straightforward and depends on perspective.

Task: Work in small groups of 3–4 students. Discuss the following questions and write down short notes:

1. Each group member gives one example of behavior (human, animal, or machine) that they consider "intelligent."

2. Compare your examples: What do they have in common? Where do they differ?

3. Try to agree on one short definition of "intelligent behavior" that fits all examples.
4. Reflect: Did you find it easy or difficult to agree? Why do you think defining intelligence is challenging?

Glossary

Artificial Intelligence (AI)
Umbrella field aiming to build systems that perform tasks requiring human-like cognition (reasoning, perception, planning, language).

Machine Learning (ML)
Subset of AI where systems improve performance on a task by learning patterns from data rather than explicit rules.

Deep Learning (DL)
ML methods using multi-layer neural networks to learn hierarchical representations (e.g., CNNs, transformers).

Generative AI (GenAI)
Models that learn data distributions and *generate* new content (text, images, audio, code) consistent with them.

Supervised Learning
Learning from labelled examples (x, y) to predict y for new x.

Unsupervised Learning
Learning structure from unlabelled data (e.g., clustering, dimensionality reduction).

Semi-Supervised Learning
Training with a small labelled set plus a larger unlabelled set to boost accuracy.

Reinforcement Learning (RL)
Agents learn sequences of actions by maximising cumulative reward through interaction with an environment.

ML Pipeline
End-to-end process: data collection $\rightarrow$ preparation $\rightarrow$ training $\rightarrow$ evaluation $\rightarrow$ deployment.

Data Collection
Acquiring raw examples relevant to the task (images, text, logs, signals).

Data Preparation
Cleaning, standardising, transforming, and formatting data so models can consume it.

Model Training
Optimising model parameters on training data to minimise a loss and learn task-relevant patterns.

Evaluation
Assessing model reliability on unseen data using appropriate metrics and checks.

Deployment
Integrating a validated model into products or workflows, with monitoring for performance and drift.

Fundamental Research
Truth-seeking work that answers precise questions and produces generalisable knowledge (papers, theory, datasets).

Applied Research
Problem-driven investigation that generates actionable evidence in a specific context (prototypes, pilots).

Technology Development
Turning research ideas into usable, scalable prototypes and methods; proving feasibility against specs.

Product Development
Delivering user-facing features/services that create business or user value in production.

Success Metric / KPI
Primary quantitative measure used to judge progress or value (task metrics or business KPIs), plus guardrails.

Technology Life Cycle
Typical trajectory of a technology: *Introduction* (prove value) $\rightarrow$ *Growth* (scale, reliability) $\rightarrow$ *Maturity* (efficiency, standards) $\rightarrow$ *Decline* (sunset or renewal).

Table 1.6: Research, Technology Development, and Product Development

Aspect	**Research**	**Technology Development**	**Product Development**
Goal	Create new knowledge or understanding.	Turn research results into usable technology.	Build a deliverable, user-ready product.
Focus	Exploring ideas, discovering principles.	Making something functional, scalable, and testable.	Creating a polished product that meets user needs.
Output	Theories, publications, prototypes, proofs-of-concept.	Tools, libraries, methods, early platforms.	Applications, APIs, physical goods, services.
Uncertainty	Very high (outcome unknown).	Moderate (technical feasibility and performance tested).	Lower (user needs known, focus is execution).
Time Horizon	Long-term (3–5 years).	Mid-term ($\approx$ 2 years).	Short-to-mid term (0.5–1 years).
Validation	Peer-reviewed, experiments, theoretical soundness.	Benchmarks, prototypes, simulations.	User testing, market fit, deployment.
Skills Needed	Domain knowledge, statistics, theoretical thinking.	Engineering, system design, software/hardware integration.	UX, product management, agile development, QA.
Success Criteria	Novelty, rigor, contribution to knowledge.	Feasibility, scalability, performance.	User adoption, business value, reliability.
Examples	Developing a new ML algorithm.	Implementing it efficiently on GPUs.	Integrating it into a mobile app for real-time predictions.

Table 1.7: When to use what.

Area → / Context ↓	**Research**	**Technology Development (prototype/ feasibility)**	**Product Development (user/ business value)**
Research Question (RQ)	**Primary.** Frames the unknown/why.	*Occasional.* Use when a fundamental unknown blocks engineering.	*Rare.* User discovery may use "questions," but scientific RQs not needed.
Hypothesis (H)	**Primary.** Testable claim with effect size & error rate.	**Useful.** Engineering hypothesis about meeting a technical threshold.	**Useful.** Product/behavioral hypothesis for A/B tests (conversion, retention, NPS).
Goal (G)	*Secondary.* Usually meta-goals (publish, share dataset).	**Primary.** Technical SMART goals (meet spec).	**Primary.** Business/user SMART goals (move KPI).
Deliverables (D)	*Secondary.* Paper, code, dataset card, repo.	**Primary.** Prototype/API, benchmark report, deployment plan.	**Primary.** Shipped feature/service, dashboard, docs, acceptance tests.
Metrics	Significance, CIs, effect sizes, generalization, etc.	Spec thresholds (e.g., $\leq 50\,\text{ms}$ p95, $\geq 92\%$ F1, $\leq \$X/1\text{k}$ calls).	Business/UX KPIs (AHT, CTR, revenue), guardrails (fairness, latency, compliance).

Chapter 2

Data as the Basis for AI

Contents

Chapter Abstract

High-performance AI systems are built on the quality, fitness and governance of their data. This chapter surveys the landscape of data for AI: types and formats (structured, semi-structured, unstructured), common sources (internal and external), and core quality dimensions (completeness, accuracy, consistency, validity, uniqueness, timeliness). We discuss labels and metadata, typical pitfalls (sampling and label bias, measurement error, leakage, and shift), and practical guardrails for responsible use. Transparency practices such as datasheets and data cards are introduced alongside lightweight documentation habits (lineage, retention, access control, minimisation, and auditability). We connect process to platform with an overview of the data lifecycle and the Medallion architecture (Bronze - Silver - Gold) as a simple pattern for progressive data refinement.

2.1 Data Types

Model capacity helps, but data are what make the difference between a useful model and a useless one. Most failures come from bad data rather than the choice of algorithm. Data come in many forms, but we can group them into a few key categories. Understanding these categories is essential because the type of data you have determines the kind of machine learning approach you can use or what you can (or can not) expect. In Table 2.1 you can see an overview of the main data types that we will discuss. In Table 2.2 you can see some examples of each category.

2.1.1 Structured Data: a Spreadsheet

This is data that is highly organised in a fixed format, such as a table with rows and columns. Each column has a specific data type (such as number, date, or category), and each row is a single record. It is the most traditional form of data and the easiest for computers to process. Examples are a library card catalogue. Everything has a designated spot, and you know exactly where to find the information you need. Other examples are bank transactions with columns for Date, Amount, Merchant, and Category, a patient database with fields for Patient ID, Age, Blood Type, and Diagnosis Code and an order history table with User ID, Product Purchased, Price, and Purchase Date.

2.1.2 Semi-structured Data

This data are not restricted to a rigid table, but it contains tags, keywords, or other markers to separate different elements and create a hierarchy. It has some organisational properties, but also a degree of flexibility. Consider, for example, a recipe. It has structured elements like an ingredient list (item,

Data Type / Format	Description
Structured	Data organized in fixed fields and records; rows and columns follow a consistent schema.
Tabular	A subtype of structured data arranged in tables, such as spreadsheets or SQL tables.
Relational	Structured data stored with defined relationships between tables (e.g., primary/foreign keys).
Semi-structured	Data with some organizational elements but flexible schema (e.g., nested or hierarchical formats).
Hierarchical	Tree-like data organization, where elements contain sub-elements (common in XML, JSON).
Key–Value	Simple paired structure, where each key has an associated value; often used in logs or configurations.
Unstructured	Data without a predefined schema or format; often freeform and heterogeneous.
Textual	Sequential character data without strict structure, such as plain text or documents.
Multimedia	Non-text data such as images, audio, or video.
Spatial / Geospatial	Data including position or geometry in space (maps, coordinates, 3D models).
Temporal / Time Series	Data organized along a time axis, representing measurements or events over time.

Table 2.1: Common data types and formats in machine learning.

quantity) and cooking time, but it also has unstructured, free-flowing text for the instructions. An email is another example; it has a structured header ('From', 'To', 'Subject', 'Date') but an unstructured body of text that might contain images or documents.

2.1.3 Unstructured Data

This is information that lacks any predefined data model or organisation. It is the most abundant type of data in the world (over 80%) and often the most valuable, but it is also the most difficult to analyse. Consider, for example, a shoebox filled with a random assortment of handwritten letters, photos, audio tapes, and newspaper clippings. The information is rich, but it is not organised in any way. Other examples are customer reviews, social media posts, legal contracts, medical notes from a doctor. Consider in addition satellite imagery, medical scans (X-rays, MRIs), photos from a self-driving car's camera.

2.1.4 Labels and Targets

We have discussed how learning (in a supervised setting) requires labels (you need to know what you want to predict, to assess how good or bad your models are). Labels (sometimes called targets) are not raw data itself, but rather the

Data Type / Format	Typical Examples
Structured	Databases, spreadsheets, CSV files.
Tabular	Excel sheets, SQL tables, relational datasets.
Relational	Customer–order databases, hospital information systems, ERP data.
Semi-structured	JSON files, XML documents, HL7/FHIR messages.
Hierarchical	File system trees, nested JSON, XML configurations.
Key–Value	Configuration files, logs, NoSQL databases (e.g., Redis).
Unstructured	Free text, images, audio, video, PDFs.
Textual	Emails, clinical notes, reports, transcripts.
Multimedia	Images, CT scans, recorded speech, surveillance videos.
Spatial / Geospatial	Maps, GPS coordinates, satellite imagery, 3D scans.
Temporal / Time Series	Sensor streams, ECG traces, stock prices, server logs.

Table 2.2: Common data types and formats with typical examples.

"answer" or "outcome" that we want our machine learning model to predict. It is often created by humans who manually annotate the raw data. This is the foundation of *supervised learning* (see Section 1.3 for a review).

For example, labels might be the solutions key at the back of a maths textbook. The student (the model) practices on the problems (the data) and checks their work against the answers (the labels) to learn. For an email (unstructured data), the label could be ‘"spam"‘ or ‘"not spam"‘. For medical imaging (unstructured data), the label could be a diagnosis like ‘"cancerous"‘ or ‘"benign"‘.

Good labels are fundamental to good models. If the labels are wrong, inconsistent, or missing, the model will learn incorrect associations. Creating reliable labels often requires human expertise and is often time consuming and costly. Labels in general can take different forms:

- **Binary:** yes/no, positive/negative.
- **Categorical:** multiple possible classes such as *cat*, *dog*, *horse*.
- **Numerical:** continuous values such as temperature, price, or tumour size.

In many real-world applications, labels are not obvious or directly measurable. For example, the label “churn” in a business context might mean that a customer stopped using a service after six months, while in healthcare it might represent a disease outcome after treatment. In general **never trust** labels without checking how they have been generated, what process has been used, and how experienced the people who did that were. For example, in the medical field, it is known that young doctors make more mistakes than experienced ones, so if labels have been assigned by inexperienced doctors, more of them will be wrong.

Labels are never completely *neutral*. They reflect human decisions about what is relevant, normal, or desirable. Therefore, the process of defining and collecting labels raises important ethical questions.

- **Who defines the labels?** For example, what counts as "success", "risk", or "suspicious behaviour" depends on human judgement.
- **Are the labels objective?** Some labels may encode social or cognitive biases, leading to unfair or discriminatory models.
- **How reliable are the labels?** Human annotators may disagree or make errors, and automated labelling may propagate existing biases.
- **How much can we trust the data source?** Labels derived from historical decisions may reproduce past injustices.

Those issues are the reasons why most of the models used today show bias or unfair behaviour.

2.1.5 Metadata

It is worth mentioning *data about data*, called metadata. It usually provides context and information about the origin, quality, and characteristics of the primary data. It is used to understand the lineage and potential limitations of the data. For example, consider the information printed on the back of a photograph. It does not describe the people in the picture, but it tells you where and when it was taken, and by what kind of camera. For images, metadata might include the GPS location, timestamp, and camera settings. For a patient record, metadata could include the hospital that collected the data, the ID of the doctor who entered the note, and the date of the last update. For a text document, metadata could include the author, the date of creation, the file size, and language. In some cases metadata can be used as labels, but that depends on the specific use-case.

Exercise

Exercise 2.7 (Classifying Data Types and Labels)
Time: 15 minutes **Group size:** 2–3 students
Task: Discuss the short project situations below. For each, decide:

(a) What is the **main data type or format** involved (choose from: Structured, Semi-structured, Unstructured, Tabular, Relational, Hierarchical, Key–Value, Textual, Multimedia, Spatial/Geospatial, Temporal/Time Series).

(b) Whether the project uses or needs **labels** (if yes, describe what the label represents).

Write down your choices and a short justification (one or two sentences).

1) A hospital collects patient vital signs from monitors every second and wants to detect early signs of heart failure.
2) A news organisation trains a model to classify incoming articles as "politics", "sports", or "culture".
3) A logistics company analyses GPS data from delivery vehicles to optimise routes.
4) A research team builds a model to summarise long legal contracts automatically.
5) An online retailer studies user reviews and star ratings to understand customer satisfaction.
6) A telecom company stores call detail records (date, duration, caller, receiver) and predicts customer churn.
7) A city wants to use satellite images to estimate the coverage of the green space.
8) A developer maintains configuration files (YAML/JSON) that describe the microservice settings and dependencies.

Deliverable (ca. 5 min): Compare responses between groups and discuss where there was disagreement, especially where one dataset could fit into more than one category.

2.2 Sources, Access, and Governance

Machine learning projects rarely start from scratch. Data often exist somewhere, usually in an organisation's internal systems, in public repositories, or can be obtained from external providers. The origin and governance of the data determine its quality, reliability and whether it can be used ethically and legally.

Internal data are generated within an organisation during its normal operations. Typical examples include operational databases, enterprise resource planning (ERP) systems, data warehouses or data lakes, user logs, and internal surveys. For example, a hospital's electronic health record system stores structured medical data; a bank's transaction database holds detailed records of customer activity; a company's website produces clickstream logs that describe user behaviour. Internal data presents several challenges:

- Data are usually **distributed across multiple systems** with different formats and owners (so it might be difficult to get access to it, since a complex access right hierarchy might be necessary).
- Access may be restricted for privacy or security reasons, especially in

regulated sectors such as healthcare or finance.

- Data quality may vary: missing fields, inconsistent identifiers, or outdated records are really common.

Usually, it is very useful to involve data owners and IT administrators early in the project. Document where the data come from, how it was collected, and what transformations have been applied. Even simple steps such as checking for duplicates or missing values can greatly improve the usability of internal data.

Often external data are necessary to augment the internal data. The problem is that it originates outside the organisation and thus you might not have full control on it or its quality. Examples include open data portals (e.g., `data.gov`, `opendata.swiss`), licensed datasets obtained through contracts, or data shared by vendors and research partners. Examples range from weather and demographic data to satellite imagery or medical benchmarks. Working with external data brings its own challenges:

- **Access and licencing:** Always verify usage rights and whether redistribution or commercial use is allowed.
- **Compatibility:** External datasets often use different formats, units, or definitions and require careful harmonisation with internal data.
- **Timeliness and maintenance:** External data may not be updated regularly or may disappear if the provider changes terms or discontinues access.

Always keep a record of the download date, data source URL, licence type, and any preprocessing applied. External data should be treated as a living resource, always recheck for newer versions or corrections.

Governance Basics

Data governance defines how data are managed, protected, and used within ethical and legal boundaries. It ensures that data use aligns with the purpose for which it was collected and that privacy and security obligations are respected. Good governance is not just about compliance; it also builds trust and improves reproducibility. Key governance principles include the following.

- **Purpose limitation:** Use data only for the specific reason for which it was collected. For example, survey data collected for internal satisfaction should not be reused for marketing without consent.
- **Consent and contracts:** Ensure individuals or partners have agreed to the intended data use, especially when personal data is involved. When working with personal or partner-provided data, explicit and informed **consent** is the foundation of lawful and ethical data use. This means that individuals must understand what data is being collected, for what purpose, how long it will be stored, and whether it will be shared or used

to train AI systems. Consent must be *freely given, specific, informed, and revocable*, people should be able to withdraw it at any time. In organizational or collaborative contexts, consent is often formalized through **contracts or data-sharing agreements**. These documents define who owns the data, who may access it, and for what purposes it can be used. They also clarify responsibilities regarding anonymization, security, and deletion once the project ends. A common challenge is that data collected for one purpose (for example, medical treatment or customer service) may later be repurposed for another (such as research or product development). Under privacy regulations such as the EU **General Data Protection Regulation (GDPR)** or the Swiss **Data Protection Act (DSG)**, this "secondary use" is only allowed if it is compatible with the original purpose or if new consent is obtained.

- **Lineage:** Maintain a clear record of where the data came from, how it was processed, and which versions were used in each model. Usually *data lineage* tries to answer the questions (i) Where did the data originate (source system, file, or external provider)? (ii) What transformations, filters, or aggregations were applied? (iii) Which data version or snapshot was used for training and evaluation? (iv) Who modified the data, and when? Without proper lineage, it becomes almost impossible to explain why a model behaves in a certain way or to reproduce past results. For example, if a healthcare model trained on a 2022 snapshot of patient data performs differently on 2024 data, clear lineage records help identify whether the issue stems from changes in population, data collection, or preprocessing pipelines. Lineage is also a key requirement in regulated domains such as healthcare, finance, and the public sector, where transparency and accountability are mandatory. Modern data management tools (for example, *data catalogs* or *model registries*) often include lineage tracking as part of their functionality, automatically recording versions, transformations, and dependencies. Open soruces data lineage tools you might want to check are *OpenLineage*, *Marquez*, *DataHub*, *OpenMetaData* and *Apache Atlas* (just to cite a few).

- **Retention and deletion:** Store data only as long as necessary and have procedures to delete or anonymize it when no longer needed. Every dataset has a lifecycle. Data should be stored only as long as it serves a legitimate and necessary purpose, after which it should either be deleted or irreversibly anonymized. Keeping data indefinitely increases the risk of breaches, misuse, and regulatory violations, and it can also lead to unnecessary storage costs and confusion about which version is authoritative. This principle of **data retention and deletion** is embedded in most data protection laws, including the EU's GDPR and the Swiss Federal Act on Data Protection (DSG). Both require organizations to define how long data will be kept and to justify that duration. For example, medical data may be retained for a fixed number of years due to legal obligations,

while user activity logs for an online platform may only need to be kept for a few months.

- **Access controls:** Limit data access to authorized personnel and use secure storage environments. Access control means that only people who genuinely need the data for their work or research can view or modify it. This principle is often summarized as **least privilege**: give each user the minimum access necessary to perform their role. For example, a data scientist may need access to anonymized records for model training, while an administrator might handle raw data under stricter confidentiality agreements. Secure storage involves both **technical measures** (such as encryption, authentication, and network security) and **organizational measures** (such as role-based access policies and confidentiality training). Data should be stored on institutional servers or trusted cloud environments that comply with recognized standards (for example ISO 27001, HIPAA, or GDPR requirements), not on personal laptops or unencrypted drives. When possible, use encrypted file systems, password-protected repositories, and secure transfer methods such as `sftp` or HTTPS instead of email attachments or USB drives. Access control is also dynamic: permissions should be reviewed regularly and revoked when no longer needed, for instance when a team member leaves a project. Audit logs and monitoring systems can record who accessed data, when, and what actions were performed, this improves transparency and supports incident investigation in case of misuse or breach.

- **Data minimisation:** The principle of **data minimization** means that you should collect and process only the data that is strictly necessary for the specific purpose of your project. In other words, "just enough data to do the job, and no more." Collecting large amounts of irrelevant or overly detailed information increases risks, privacy violations, security breaches, and model bias, without necessarily improving performance. Before gathering or using data, ask what question am I trying to answer, and what data do I really need to answer it? For example, if the goal is to predict hospital readmission rates, it may be unnecessary to include patients' names, addresses, or full medical histories. Similarly, when analyzing customer churn, aggregated statistics such as age group or region might be sufficient instead of storing personally identifiable details like phone numbers or exact birth dates. Data minimization is not just an ethical ideal but also a legal requirement under regulations such as the EU's GDPR and the Swiss Data Protection Act (DSG). These frameworks require that data collection be proportionate to the stated purpose and that unnecessary information be avoided or anonymized. Smaller, well-targeted datasets are also easier to manage, secure, and document, reducing cost and complexity.

- **Auditability** means that all data usage and transformations can be reviewed, traced, and reproduced later. It ensures that anyone, whether

a colleague, an external reviewer, or a regulator, can understand how the data was obtained, how it was processed, and how it was used to train or evaluate a model. In practice, auditability requires keeping detailed records of data handling activities. This includes documenting data sources, preprocessing scripts, cleaning rules, filtering criteria, and any manual interventions or labeling processes. Each major version of a dataset or model should be identifiable, ideally through version control systems or metadata tracking tools. Automated logging systems can record who accessed or modified data and when, providing a verifiable history of operations. Lack of auditability is a common cause of irreproducible research and unexplainable model behavior. When the data pipeline is opaque, even small changes, for example, a missing preprocessing step or a re-exported CSV file, can make it impossible to replicate past results. In regulated environments such as healthcare or finance, inadequate documentation of data use can also lead to compliance violations.

Remember to treat data governance as an ongoing process, not a one-time task. Document everything, where the data came from, how they were changed, and who has access to them.

2.3 Data Quality: Dimensions and Quick Checks

The dimensions listed in Table 2.3 summarise the main aspects of data quality that determine whether a dataset is suitable for machine learning. Even the most sophisticated model will fail if the underlying data is incomplete, inaccurate, or inconsistent. Understanding and checking these dimensions helps identify common data issues early in a project, before they propagate into misleading results.

Completeness

Completeness describes whether all required data fields and records are present. Missing values may arise from system errors, optional fields, or differences in data entry practices. For example, if patient ages are missing in 20% of hospital records, the model may learn biased associations. Quick checks include computing null or missing-value rates per column or per group, and verifying whether certain entities (e.g., patients, customers, sensors) are systematically underrepresented.

Accuracy

Accuracy measures how closely the recorded values reflect the facts in the real world they represent. Errors can arise from faulty sensors, manual entry mistakes, or outdated sources. For example, if GPS coordinates are rounded to the nearest kilometre, location predictions may become unreliable. Accuracy can

be verified through range rules (e.g., the human temperature must be between 35–42 °C) or by cross-checking values against reference systems or independent datasets.

Consistency

Consistency refers to whether data have the same meaning, units, and interpretation across sources. Different systems may represent the same concept in slightly different ways, for example, "M" versus "Male", or weight in kilogrammes versus pounds. Inconsistencies can strongly affect model quality.

Validity

Validity checks whether data conforms to expected formats, types, or business rules. For example, dates should follow a standard format, postal codes must match known patterns, and categorical fields should only contain allowed values. Automated tools can flag invalid entries using regular expressions, enumerations, or parsing rules. High rates of invalid values often indicate upstream issues in data collection.

Uniqueness

Uniqueness ensures that entities are represented only once, without duplicates or leakage. Duplicate records inflate sample sizes and can bias models, while duplicated identifiers may cause confusion between individuals or cases. Typical checks involve detecting repeated primary keys, comparing hashes for near-duplicate rows, or verifying that training and test sets do not share overlapping entities.

Timeliness

Timeliness captures how up-to-date the data is relative to the phenomenon being modeled. For dynamic processes such as financial transactions or patient monitoring, stale data can lead to outdated predictions. Measuring data age distributions or calculating the lag between event occurrence and data availability provides a quick indication of freshness. When data pipelines are automated, continuous monitoring can alert teams if updates stop or delays increase.

Quick Exercise (10 min)

Exercise 2.8

Think to data you work with in your company or projects, or imaging one dataset. List *two* fields where missing or invalid values would critically harm decisions. For each, note one automated check you would implement and one manual check.

Dimension	Typical Questions	Quick Checks
Completeness	Are fields missing?	Null rates, per-group missingness.
Accuracy	Do values reflect reality?	Ranges (age is realistic?), consistent measurements, etc..
Consistency	Same meaning across sources?	Schema diffrences, unit of measurements checks (something measured in cm and something in m).
Validity	Conform to allowed formats?	date parsing errors.
Uniqueness	Duplicates? leakage?	Primary-key duplicates if not enforced, etc.
Timeliness	Fresh enough?	Age histograms, missing time periods, only old data.

Table 2.3: Data quality cheat sheet.

2.4 Transparency: Datasheets and Data Cards

Transparency is a cornerstone of trustworthy AI. Just as scientific experiments require detailed documentation of materials and methods, datasets used for machine learning should also come with clear and standardized documentation. This ensures that others can understand what the data represents, how it was collected, what its limitations are, and under what conditions it may be used. Poorly documented datasets are one of the main causes of hidden bias, misuse, and irreproducible results. A practical approach to dataset transparency is the use of short, structured summaries known as **datasheets** or **data cards**. A datasheet is a comprehensive report describing the motivation, composition, collection process, and recommended uses of a dataset, while a data card is a lighter, more accessible version that fits on one or two pages. Both serve the same purpose: to communicate essential information about the dataset to researchers, developers, and end users.

Every dataset used for machine learning should include at least a minimal *data card*, documenting its origin, intended use, coverage, known gaps, and legal or ethical constraints. Table 2.4 shows an example of a compact structure. Data cards make the invisible visible. They provide essential context that prevents datasets from being treated as neutral or universal when in fact they reflect specific choices and constraints. A well-prepared data card helps new users quickly understand what the dataset can and cannot support, reducing the risk of inappropriate use or overgeneralization. For example, a face recognition dataset that clearly documents its demographic distribution allows users to assess fairness risks before deployment. Creating a data card does not require legal training or a lengthy report, it can start as a simple text file stored alongside the dataset. The key is to record information as soon as it becomes available: who collected the data, why, how it is updated, and what issues are already known. Whenever data is shared internally or externally, the data card should accompany it. It

Field	Content
Name/Owner	Dataset title, maintainer, contact person or organization.
Provenance	Source systems, collection dates, jurisdictions, and responsible parties.
Intended Use	Supported decisions and model types; explicitly note any out-of-scope or inappropriate uses.
Population	Who or what is represented (patients, customers, devices, regions) and known exclusions or imbalances.
Labels	How labels were obtained, quality control procedures, and any inter-rater variability notes.
Quality Notes	Known limitations such as missing data, measurement errors, drift, or potential bias sources.
Legal/Ethics	Consent status, licensing terms, privacy measures, and reference to Data Protection Impact Assessment (DPIA) if available.
Updates	Frequency of refreshes, versioning strategy, and deprecation plan for outdated data.

Table 2.4: Minimal data card template.

should also be updated when the dataset changes, just like versioned code or model documentation.

Exercise (20 min)

Exercise 2.9 (Write a One-Page Data Card)
In groups, draft a data card for a dataset of your choosing. Choose one of the following real-world datasets or open data sources. Your goal is not to download or process the data, but to investigate its documentation online and draft a short data card describing its origin, purpose, content, and limitations. Use the guiding prompts from Table 2.4.

1) **OpenStreetMap (OSM):** A global, volunteer-created map of the world. Consider who contributes, what geographic regions are well or poorly represented, and how licensing (Open Database License) affects reuse (`https://www.openstreetmap.org/`).

2) **MIMIC-III Clinical Database:** An openly available dataset of intensive care unit (ICU) patient records from Beth Israel Deaconess Medical Center. Reflect on consent procedures, de-identification, and ethical use in medical AI (`https://physionet.org/content/mimiciii/1.4/`).

3) **COCO (Common Objects in Context):** A large image dataset used for object detection. Examine how images were collected and annotated, what categories exist, and what populations or contexts might be underrepresented (`https://cocodataset.org/`).

4) **IMDb Movie Dataset:** Widely used for research on recommendations and text analysis. Consider what entities are represented, potential commercial biases, and the challenges of keeping such a dataset current (`https://ai.stanford.edu/~amaas/data/sentiment/`).

5) **Humanitarian Data Exchange (HDX):** Managed by the UN, hosting global crisis and development data. Pick one dataset (e.g., refugee movements, food security) and describe its origin, sensitivity, and governance mechanisms (`https://data.humdata.org/`).

6) **Kaggle COVID-19 Datasets:** Community-shared pandemic data including case counts, mobility, and policy responses. Consider timeliness, reliability of sources, and differences between countries' reporting standards (`https://www.kaggle.com/datasets/imdevskp/corona-virus-report`).

2.5 The Data Lifecycle

Data is not static; it moves through a series of stages from initial collection to eventual archiving or deletion. Understanding the **data lifecycle** helps ensure that data is managed responsibly and efficiently across its entire existence. It also clarifies who is responsible for what at each step and where governance, quality, and transparency measures should be applied. The lifecycle typically includes six to eight stages, depending on the organization. Table 2.5 summarizes the main ones, while Figure 2.1 shows a diagram of it.

Always remember that the data lifecycle is iterative, not linear, feedback from deployment and monitoring often leads back to new data collection or improved cleaning steps. Each phase should include documentation, versioning, and checks for quality, privacy, and security.

2.6 The Medallion Architecture

Modern data platforms increasingly organize their data processing pipelines using the **Medallion Architecture**, a layered approach that improves data quality and governance. Originally popularized by Databricks, it defines a sequence of layers: Bronze, Silver, and Gold, each representing increasing levels of data refinement and reliability.

The goal of the Medallion Architecture is to separate raw, cleaned, and aggregated data into distinct, traceable stages. Each layer builds upon the previous one, ensuring that data transformations are transparent, reproducible, and reversible. Table 2.6 summarizes the three layers.
Each layer plays a distinct role:

Stage	Description and Typical Activities
1. Collection / Acquisition	Data is generated or gathered from sensors, transactions, surveys, or external APIs. Governance begins here, with consent, sampling, and documentation.
2. Ingestion / Storage	Raw data is imported into storage systems such as databases, data lakes, or warehouses. Metadata is captured, formats standardized, and access rights defined.
3. Cleaning / Preparation	Errors, duplicates, and missing values are corrected; data is transformed, joined, and validated. This stage often consumes 60–80% of a project's effort.
4. Integration / Enrichment	Data from multiple sources is combined and augmented (e.g., adding geolocation or demographic information). Lineage and version control are critical.
5. Analysis / Modeling	Machine learning models, dashboards, or reports are developed and tested. Data scientists and analysts work on curated, high-quality datasets.
6. Deployment / Use	Analytical results or models are integrated into products, decisions, or automated systems. Monitoring ensures ongoing performance and fairness.
7. Archiving / Retention	Data is stored long-term for compliance, reproducibility, or audit purposes, often in secure, low-cost environments.
8. Disposal / Deletion	When data is no longer needed, it is securely deleted or anonymized to reduce risk and comply with retention policies.

Table 2.5: Typical stages of the data lifecycle.

- The **Bronze** layer captures data as-is, ensuring nothing is lost and that raw sources can always be revisited.
- The **Silver** layer standardizes and cleans data, creating a "single source of truth" suitable for most analytical and modeling tasks.
- The **Gold** layer delivers trusted, curated data that supports business intelligence, dashboards, or deployed machine learning pipelines.

Layer	Purpose	Typical Contents / Actions
Bronze (Raw)	Store data exactly as received, preserving full fidelity.	Logs, CSVs, JSON files, or sensor streams ingested from source systems. Minimal cleaning; focus on completeness and traceability.
Silver (Cleaned)	Provide structured, validated, and joined datasets ready for analysis.	Deduplicated tables, consistent schema, enriched with metadata, unit conversions, and error handling. Often used by data scientists for feature extraction.
Gold (Curated)	Deliver high-value, business-ready data products.	Aggregated datasets, KPIs, dashboards, and model training tables. Serves decision-makers or production systems. Emphasis on accuracy, performance, and governance.

Table 2.6: The Medallion Architecture: layers of data refinement.

Glossary

Artificial Intelligence (AI)
: The broad field aiming to build systems that perform tasks requiring human-like cognition (reasoning, perception, planning, language).

Machine Learning (ML)
: A subset of AI where systems improve at a task by learning patterns from data rather than using hand-crafted rules.

Deep Learning (DL)
: Machine learning methods based on multi-layer neural networks that learn hierarchical representations (e.g., CNNs, transformers).

Generative AI (GenAI)
: Models that learn data distributions and create new content (text, images, audio, code) that resembles the training data.

Supervised Learning
: Learning from labelled pairs (x, y) to predict the correct y for new inputs x.

Unsupervised Learning
: Discovering structure in unlabelled data, such as clusters or low-dimensional representations.

Semi-Supervised Learning
: Training with a small labelled set plus a large unlabelled set to boost performance and data efficiency.

Reinforcement Learning (RL)
: Agents learn sequences of actions by maximising cumulative reward through interaction with an environment.

ML Pipeline
: The end-to-end process: data collection → preparation → training → evaluation → deployment.

Data Collection
: Acquiring raw, relevant examples (images, text, logs, signals) for the task at hand.

Data Preparation
: Cleaning, standardising, and transforming data (handling missingness, formatting, feature extraction) so models can use it.

Model Training
: Optimising model parameters on training data to minimise a loss function and learn task-relevant patterns.

Evaluation
: Testing on unseen data with appropriate metrics to verify reliability, generalisation, and trade-offs.

Deployment
: Integrating a validated model into products or workflows; includes monitoring for performance, drift, and failures.

Fundamental Research
: Truth-seeking work that answers precise questions and produces generalisable knowledge (theory, methods, datasets).

Applied Research
: Problem-driven investigation that generates actionable evidence in a specific context (prototypes, pilots, field tests).

Technology Development
: Turning research ideas into usable, scalable prototypes and methods; proving feasibility against technical specifications.

Product Development
: Delivering user-facing features/services that create business or user value in production environments.

Success Metric / KPI
: The primary quantitative measure of progress or value (task metrics or business KPIs), plus any guardrail metrics.

Technology Life Cycle

A technology's trajectory: *Introduction* (prove value) → *Growth* (scale, reliability) → *Maturity* (efficiency, standards) → *Decline* (sunset or renewal).

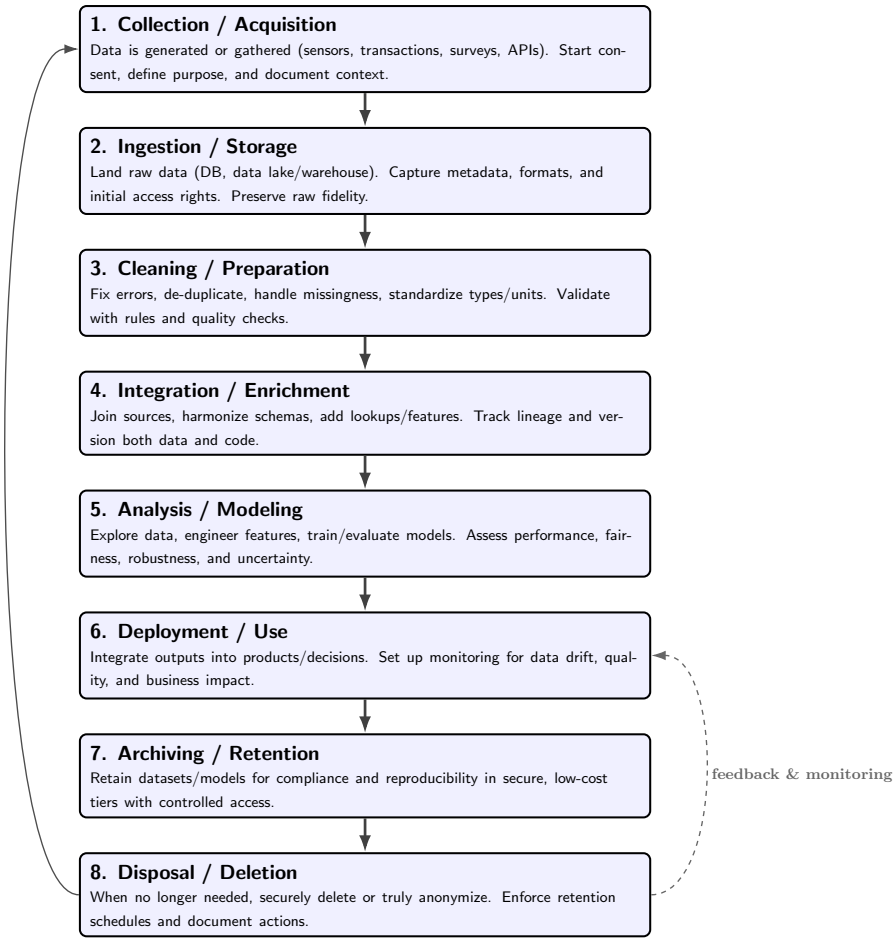

Figure 2.1: Data Lifecycle. The loop on the left emphasizes the continuous nature of data processes.

Chapter 3

Machine Learning in Practice: Tasks, Data, and Validation

Contents

Chapter Abstract

This module gives a practical mental model for how machines learn and how to use models responsibly. We discuss common task types (classification, regression, clustering), and explain inputs (*features*), training, and validation. We show what model validation is, and how to pick metrics that match real costs (e.g., precision/recall for detection, MAE/RMSE for forecasts), with a short advanced reference for formulas. We emphasise human oversight, simple checks for fairness and calibration, and clear communication of what a model can and cannot do. We also introduce AutoML/no-code tools, where they help, where they fail, and close with practical guidance on validation (hold-out and intuitive cross-validation) and deciding when a model is "good enough" for deployment.

3.1 Classification, Regression, and Clustering

Machine learning systems can handle many different types of questions. At a high level, most problems fall into three families depending on what we are trying to predict: **classification** (choose a category), **regression** (predict a number), and **clustering** (find natural groups). Each has a distinct goal, output, and way of being evaluated.

3.1.1 Classification

In a classification problem, the model learns to assign each case to one of a few predefined categories or labels. Typical questions are "Is this transaction fraudulent or not?", "Does this patient have disease A, B, or C?", or "Will this loan be approved or denied?". The output is a *label*, a discrete choice among possible classes. Often, instead of a single yes/no answer, the model also gives a *probability* or *confidence score*. For example, an email filter might say "80% chance this message is spam." Humans or downstream systems can then choose a threshold. perhaps flagging messages only when the probability is above 90%.

Definition - Classification

Classification is a type of machine learning used to assign data to discrete categories, such as classifying emails as spam or not spam.

Some examples of classification tasks are the following.

- Detecting spam vs. non-spam emails based on message content and sender.
- Approving or rejecting a loan based on applicant history.
- Recognising objects in an image (cat, dog, car, or person).
- Predicting whether a customer will churn (leave) or stay.

Types of Classification

Not all classification problems look the same. Below are common variants you will encounter.

Definition - Types of Classification

Binary classification is used when there are only two possible categories, for example deciding whether an email is spam or not spam.

Multi-class classification is used when there are more than two possible categories, for example recognizing whether an image shows a cat, a dog, or a horse.

Multi-label classification is used when each item can belong to several categories at once, for example tagging a photo as both "beach" and "sunset."

Binary Classification

in binary classification, as the name suggests, you have two possible categories.

- Examples: fraud vs. not-fraud; churn vs. stay; spam vs. not-spam.

Example - Binary Classification

Binary classification means teaching a computer to decide between two possible outcomes. For example, a program could learn to tell whether an email is "spam" or "not spam" by looking at many examples of both. After learning from these examples, the system can look at a new email and predict which category it belongs to. Similar methods are used in medicine (sick or healthy), banking (fraud or not fraud), and many other fields.

Multi-class Classification

In multi-class classification you have three or more *mutually exclusive* categories. It is important to note that each case belongs to exactly one of them.

- Examples: choose product category (`electronics / clothing / groceries`); disease A/B/C; image is `cat / dog / car / person`.

Example - Multi-Class Classification

Multi-class classification means teaching a computer to choose between more than two possible categories. For instance, an image recognition

program might learn to tell whether a picture shows a cat, a dog, or a horse. After training on many labeled examples, the system can look at a new image and decide which category it fits best. This type of approach is widely used in applications such as handwriting recognition, product sorting, and medical image analysis.

Multi-label Classification

In multi-label classification, a single input can belong to *several* categories at once.

- Examples: a photo tagged with {`beach`, `sunset`, `people`}; an email labelled both `billing` and `urgent`; a movie tagged {`comedy`, `romance`}.

Example - Multi-Label Classification

Multi-label classification means that each item can belong to several categories at the same time. For example, a photo of a beach might be labeled both "water" and "sunset," or a medical report could indicate several conditions for one patient. The computer learns from many examples how different labels can appear together and then predicts all relevant labels for a new case. This method is often used in areas like music recommendation, image tagging, and medical diagnosis.

Ordinal Classification

Categories have a natural order, but are still labels (not numbers you add/subtract meaningfully).

- Examples: customer satisfaction {`low`, `medium`, `high`}; credit rating bands; disease severity stages.

Example - Ordinal Classification

Ordinal classification is used when the possible categories have a natural order, but the exact distance between them is not known. For instance, a system might learn to predict customer satisfaction as "poor," "average," "good," or "excellent." These categories follow a clear order from low to high, but the difference between "average" and "good" is not necessarily the same as between "good" and "excellent." Such models are common in surveys, medical assessments, and risk evaluations.

One-vs-Rest (OvR) and One-vs-One (OvO) (Optional)

Sometime it is useful to tackle multi-class problems using simpler binary pieces.

- **OvR (One-vs-Rest):** train one small classifier per class (e.g., "is it `electronics` vs. not?"). At prediction time, pick the class with the strongest score. *Analogy:* each class has its own "advocate"; the loudest advocate wins.
- **OvO (One-vs-One):** train a classifier for every pair of classes (electronics vs. clothing, electronics vs. groceries, clothing vs. groceries, ...). *Analogy:* a mini tournament; the class that wins the most pairwise "matches" is chosen.
- When to use: OvR is simple and scales well; OvO can work nicely when classes are hard to separate overall but easy to separate in pairs.

In Table 3.1 you can find a concise overview.

Type	What It Means	Example
Binary	Exactly two categories; pick one.	Spam vs. not-spam; churn vs. stay.
Multi-class	One of many categories (mutually exclusive).	Product category: electronics/clothing/groceries.
Multi-label	Many labels can apply at once.	Photo tags: beach + sunset + people.
Ordinal	Ordered categories (labels with rank).	Satisfaction: low/medium/high.
One-vs-Rest	Multi-class via "one class vs. all others" mini-models.	Separate "is it electronics?" model, etc.
One-vs-One	Multi-class via "pairwise matches" between classes.	Electronics vs. clothing; winner overall is chosen.

Table 3.1: Common classification setups and everyday examples.

3.1.2 Regression

In regression, the output is not a category but a continuous number, something that can take any value within a range. These models try to estimate *how much*, *how long*, or *what value*. Rather than deciding between options, the goal is to make a numerical prediction as close as possible to the true answer.

Definition - Regression

Regression is a type of machine learning used to predict continuous values, such as temperature, price, or age.

Some examples of regression tasks are the following.

- Forecasting next week's sales for each store.
- Predicting the time a delivery will arrive based on distance and traffic.

- Estimating house prices from size, location, and number of rooms.
- Predicting a patient's blood glucose level one hour from now.

Example - Regression

Regression is used when we want a computer to predict a number rather than a category. For example, a model can learn to estimate the price of a house based on its size, location, and number of rooms. After learning from many examples, it can give a numerical prediction for a new house. Regression is widely used in areas such as economics, engineering, and healthcare to forecast values or measure trends.

3.1.3 Clustering (no labels, discover groups)

Clustering is different: there are no predefined labels or outcomes to predict. Instead, the goal is to find structure in the data, grouping items that behave or look similar. The model tries to organise the data into a small number of meaningful clusters so that items within a cluster are more alike than items in different clusters.

Definition - Clustering

Clustering is a type of unsupervised learning used to group similar data points together, so that items in the same group are more alike than those in different groups.

This is called **unsupervised learning** because we are not teaching the model with correct answers; it has to "discover patterns" on its own. Some examples of clustering use-cases are the following.

- Grouping customers into a few segments based on spending habits and demographics.
- Identifying patterns of website visitors with similar browsing behaviour.
- Detecting unusual patterns (anomalies) that do not fit any cluster—potentially signalling errors or fraud.
- Organising articles or news stories into related themes without predefined topics.

The Table 3.2 below summarises how classification, regression, and clustering differ in what they predict, what inputs they use, and how we interpret their outputs. Understanding which type of problem you are dealing with is the first step before choosing a model or evaluating performance. If you have labelled data with known outcomes (spam/not-spam, price, etc.), the task is supervised, classification or regression. If you only have unlabelled data and want to explore structure, it is unsupervised, in this case clustering. These three categories cover most practical cases and form the foundation for more advanced approaches.

Type	Question It Answers	Typical Inputs (Examples)	Outputs (Examples)
Classification	Which category does this item belong to?	Email text, sender, links; patient record; customer profile.	{spam, not-spam}; {approve, deny}; disease A/B/C; probability of class.
Regression	What is the expected numeric value?	Past sales, weather, advertising spend; distance, speed, time.	Sales next week = 500; price = $350,000; delay = 12 minutes.
Clustering	What natural groups or patterns exist?	Customer demographics, purchase history, web activity.	Three customer segments; five behaviour groups; outliers for investigation.

Table 3.2: Different types of machine learning tasks and their outputs.

3.2 Recommender Systems

Many products suggest (or recommend) items (movies, products, news, courses) by ranking a large catalogue for each user. Recommender systems predict which items a user is likely to engage with (for example buy, consider amazon, or wath, think netflix) next and return a short, ordered list. Unlike standard classification or regression, the output is typically a *top-k list* or a set of scores used to sort items.

Definition - Recommender Systems

A recommender system estimates user preference for items and produces a ranked shortlist (top-k) tailored to each user. It learns from historical interactions (e.g., clicks, views, purchases, ratings) and optional side information (item metadata, user attributes, context such as time or device).

Example - Amazon and Netflix

Amazon (Shopping). Imagine you are looking at a coffee grinder on Amazon. Under the product, you see suggestions such as "Frequently bought together" (grinder + filters + beans) or "Customers who viewed this also viewed" (other grinders at similar prices). Amazon's recommender system is doing two things at once: it notices what *you* have looked at or purchased before (for example, kitchen gadgets), and it learns from patterns across *millions* of other shoppers. If many people

who bought this grinder also bought filters and beans, it suggests those to you as well. It then mixes in a few new or popular products so you don't only see the same type of items repeatedly.

Netflix (Streaming). When you open Netflix, the rows such as "Top Picks for You" or "Because you watched…" are arranged specifically for your viewing habits. If you often finish light comedies in the evening and crime dramas on weekends, Netflix takes note. It looks at what you've watched, what you stopped halfway through, and what people with similar tastes enjoyed. From this, it builds a short list of titles you are most likely to click on next, perhaps a new comedy, a classic you missed, and a film starring your favourite actor. It also tries to balance variety, so you don't see twelve similar shows in a row, keeping the recommendations fresh and engaging.

3.3 Model Validation

A model that performs well on the data it was trained with does not automatically work well on new, unseen data. **Model validation** is the process of checking whether what the model has learned truly generalises beyond the training examples. It answers the question: "Will this model still perform well tomorrow, on future or unseen data?" Without proper validation, a model may look excellent on paper but fail in practice, a problem known as **overfitting**[1], where the model memorises details of the training data instead of learning general patterns. Model validation is fundamental since it provides an honest estimate of how the model will behave in real use. Furthermore, it prevents over-optimistic results that come from testing on the same data used for training.

Example — Why Validation Matters

A retailer trains a model to flag likely returns so staff can double-check those orders before shipping. On last month's data the model looks *amazing*: it "catches" almost every return.
What went wrong? Last month was the *holiday season*: lots of gift purchases, many quick returns, and several short-term promotions. The model quietly learned those holiday quirks (e.g., gift wrap, rush shipping) rather than general patterns. When January arrives, behaviour changes, fewer gifts, different items, and the model suddenly misses many real returns and raises false alarms on normal orders. It looked great on the past, but fails on the next month. This is overfitting.

[1] Overfitting is only one aspects of a model that behaves badly on new data, but is the most common one.

What simple validation would have shown. If we had *simulated the future* by training on earlier weeks and *validating on later weeks* (a time-based split), we would have seen the drop in performance outside the holiday period. Keeping a small *final test set* completely untouched until the end would have confirmed the real-world score before deployment.

3.3.1 The Hold-Out Approach

The most common and practical method for beginners is the **hold-out approach**. Here, we divide our available data into separate parts before training begins:

- a **training set**, used for the model to learn patterns;
- a **validation set**, used to test and tune model choices;
- and sometimes a final **test set**, kept completely aside until the very end.

Example — Why Two Datasets Aren't Enough When Tuning

Suppose you split your data into just two parts: *train* (to teach the model) and *test* (to check how good it is). You start adjusting the model's *settings* (parameters) to get a better score. Each time you tweak something, you look at the *test* result to see if it improved. After many rounds, the model looks great on this train/test pair. What can happen, is that by repeatedly checking the test score while tuning, you've *unknowingly trained to the test*. The model's settings have started to fit the *specific quirks* of that particular test set. So when you try the "final" model on a *new* dataset (a fresh validation), performance drops, sometimes a lot. It wasn't truly good; it was just good at that one test.

Imagine you're perfecting a soup recipe and keep asking the *same* friend (the test dataset) to taste it after each change. Eventually you tailor the soup to your friend's exact tongue. When you serve it to a new group (the validation dataset), many don't like it. You didn't make a generally tasty soup, you made a soup tuned to one taster (your friend).
To address this, you can do this instead.

- **Train set:** the model learns patterns here.
- **Test set:** you adjust settings based *only* on this set (no peeking at the test).
- **Validation set:** kept untouched to the very end; used once to estimate the *true* final performance.

When you're tuning model parameters, two sets are not enough, you might overfit to the test without noticing. Use train to learn, test to

tune, and validation once at the end to check if it will work in the real world.

Example - Loan Applications

Imagine we have 10,000 historical loan applications with outcomes ("approved" or "repaid late"). We could split them as:

- 70% (7,000 cases) for training,
- 15% (1,500 cases) for test during development,
- 15% (1,500 cases) for validation final test.

The model is trained only on the training data. After each training round, it is evaluated on the test data to see how well it generalises. Once the design is fixed, it is tested one final time on the untouched validation set to estimate real-world performance.

This simple method, also called a **train/validation/test split**, is the foundation of trustworthy model evaluation. The key rule is: the model should never "see" the data used for its final assessment.

Sometime in the literature you might find the names *test* and *validation* exchanged.

Exercise

Exercise 3.10
You have 12,000 customer records to predict churn (leave/stay). Propose a hold-out split (percentages or counts) and explain in one sentence why you chose those proportions.

3.3.2 (Optional) Advanced Validation Methods

For larger or more critical projects, we can improve reliability by repeating the validation several times. Two common approaches are worth knowing intuitively.

K-fold cross-validation. The data are divided into k equally sized parts (for example, $k = 5$). We train the model k times, each time using a different part for testing and the rest for training. The average performance across the k rounds gives a more stable estimate, because every data point is used for testing once. Think of it as taking several "rotating hold-out" samples to see how consistent the model really is. If you are interested in more details you can find them in chapter 7 in [34].

Monte Carlo (repeated random splits). Instead of fixed folds, we repeatedly make random train/test splits (e.g., 80%/20%) and average the results. This helps spot whether the model's performance changes much depending on how the data are split, useful for small or noisy datasets. If you are interested in more details you can find them in chapter 7 in [34].

Both of these methods are more computationally expensive but follow the same principle: *keep some data unseen, test repeatedly, and trust only results that remain stable across splits.*

3.4 Overfitting, Underfitting, and Baselines

Models can fail in many ways, but two are by far the most commone ones: over- and underfitting. **Overfitting** happens when a model learns training-set quirks (noise, coincidences, formatting artefacts) instead of the underlying pattern, so it looks great on past data but *fails on new data*. **Underfitting** happens when a model is too simple or poorly specified to capture real structure, so it performs poorly on both training and new data.

Think of studying for an exam.

- **Overfitting** is memorising last year's questions word-for-word; you ace the old paper but stumble on the new one.
- **Underfitting** is skimming headlines; you never learned enough detail to do well on any paper.

In practice, an overfitted model may trigger many false alarms in production or miss key cases when the environment shifts. An underfitted model will rarely be useful: it lacks the nuance to support decisions better than a simple rule of thumb.

Definition — Overfitting and Underfitting

Overfitting happens when a model learns the training examples *too precisely*, including their noise and random details. It performs extremely well on the data it has already seen but fails on new data because it has memorised the past instead of learning general patterns.

Underfitting happens when a model is *too simple* or not flexible enough to capture real patterns in the data. It performs poorly both on the training data and on new data because it has not learned enough.

3.4.1 Model Diagnostics

It is often possible to recognise overfitting or underfitting by looking at a few simple patterns in how the model behaves on training and validation data. Even without advanced tools, these signals give an intuitive sense of whether

the model has learned genuine structure or is simply memorising what it has seen.

The first and most direct clue comes from comparing the model's performance on the **training set** (the data it learned from) with its performance on the **validation or test set** (data it has never seen before). If the model does extremely well on the training data but much worse on the validation data, this "gap" usually means it has **overfit**, it learned specific details, noise, or coincidences from the training examples that do not generalise. For example, a loan approval model might learn that applicants from a specific ZIP code in the training data always repaid, but that pattern does not hold for new applicants. Conversely, if both the training and validation results are poor, the model is probably **underfitting**: it never captured the true signal and is simply too limited to learn meaningful patterns. For instance, using only the average sales across all stores to forecast next week's demand will ignore seasonality and local differences, it is too blunt to be useful anywhere.

A second way to diagnose the issue is through **learning curves**, which plot how the model's error changes as more training data are added. Conceptually, if the validation error keeps dropping as more data become available, it means the model is capable of learning but simply does not have enough examples yet, adding more data would likely help. However, if both the training and validation errors quickly flatten out at a high level and do not improve even with more data, the model is underpowered for the task: it is **underfitting**. In contrast, if the training error keeps improving but the validation error gets worse or fluctuates wildly, that is a clear sign of **overfitting**. In practice, this might look like a marketing prediction model that performs well when trained on recent campaigns but fails when applied to new ones, the model has memorised campaign, specific quirks rather than general buying patterns.

Another useful check is to look at **errors across different segments or cohorts**. If performance varies greatly between groups, say, the model works very well for one hospital but poorly for another, or for one region but not others, it may have overfit to the dominant group in the training data. Sometimes this happens when one group is overrepresented in the dataset, or when small differences in data collection introduce hidden biases. Examining results by segment helps reveal these weaknesses and ensures the model is reliable for all intended users.

Finally, consider how the model performs over time, its **time stability**. A model that performed well last month but suddenly deteriorates this month is likely overfitted to past conditions. Perhaps it learned relationships that no longer hold, such as patterns in older customer behaviour or seasonal pricing rules. For example, a demand forecast trained during stable months might fail as soon as new promotions or disruptions appear. Monitoring time stability helps detect such "data drift" early and reminds us that a model's value depends not only on accuracy today but also on how long that accuracy lasts.

Together, these simple diagnostic checks, comparing train and validation results, observing how performance changes with more data, checking group, level consistency, and monitoring stability over time, offer a practical toolkit

for understanding whether a model is learning the right patterns or merely memorising the past. In Table 3.3 you can see a comparison of overfitting,

	Symptoms	**Likely Causes**	**Quick Fixes**
Overfitting	Train score excellent; validation/test much worse; performance drifts fast in production.	Too complex for data; leakage; noisy labels; over-tuned on small validation.	Simplify model; remove leakage; regularise/early-stop; add data; use time/group-aware splits.
Underfitting	Both train and validation poor; errors look "blunt" across all cases.	Model too simple; missing inputs; poor encodings; over-smoothing.	Add informative features; better encodings; allow more flexibility; tune regularisation.
Healthy	Train and validation close; stable across cohorts and time; baseline clearly beaten.	Balanced complexity; good splits; appropriate features.	Monitor in production; revalidate on drift; keep baseline as a canary.

Table 3.3: Spotting and addressing overfitting vs. underfitting; keep a baseline as an anchor.

underfitting and healthy model behaviour.

3.4.2 Practical Remedies

When a model does not perform well, it is often because it has either **overfit** (memorised too much) or **underfit** (learned too little). Here are some simple ways to fix each situation.

- **If the model is overfitting:** make it simpler (for example when working with neural networks, you might consider a smaller network) so it focuses on the main patterns, not tiny details that are irrelevant. You can remove unnecessary inputs (for example you might have too many features), shorten training time, or give it more fresh data to learn from. Also, make sure you test it on data from different time periods or groups, so it cannot just memorise one pattern.
- **If the model is underfitting:** help it learn more by adding better information (for example if you are predicting the weather only by using the temperature at certain location, you might need to consider wind, precipitation, etc. in addition) or letting it be slightly more flexible. Give it clearer, more useful inputs, or allow it to train longer (when working

with neural network you decide how long a network should learn from data, the longer it is, often the better) so it can pick up real trends instead of just the broad average.

Example — Fixing Overfitting

A company builds a model to predict which customers will cancel their subscription. At first, the model seems perfect, it matches last year's data almost exactly. But when used on new customers this year, it fails badly. It had memorised small coincidences from the old data, such as a one-time marketing campaign.
You might consider fixing the model in the following ways.

- Simplify the model so that it focusses on stable patterns (for example, general spending habits, not specific past dates).
- Remove inputs that are only relevant to the past, such as the last year discount codes.
- Add more recent customer data, so that the model sees a wider variety of behaviour.
- Test it on different time periods to make sure it generalises.

After these changes, the model should no longer memorise history, it should learn the real reasons customers leave.

Example — Fixing Underfitting

A delivery company wants to predict how long each delivery will take. They build a simple model using only one piece of information: the distance between the restaurant and the customer. The predictions are always "average", sometimes too fast, sometimes too slow, because the model ignores other important clues.
You might consider fixing the model in the following ways.

- Add more useful information, such as time of day, traffic level, weather, and driver workload.
- Let the model learn a bit more detail instead of keeping it too simple.
- Consider a more flexible model. If you have used a linear regression, it might not be enough to capture the subtleties of your data.

With these improvements, the model becomes more accurate because it finally captures the real-world factors that affect delivery time.

3.5 Feature Engineering

Every machine learning model learns patterns from information we provide as input. These characteristics of the inputs are called **features**. They are the measurable properties or clues that help the model make predictions, such as a person's age, the number of previous purchases, the time since an event, or how many times a word appears in a text. Good features contain information that truly relates to the decision we want to support and, crucially, are available *before* the prediction moment.

Definition — Feature

A **feature** is a piece of information that describe a certain object or event that you want to understand. It describes something about each case in the data, for example, a customer's age, a product's price, or the number of days since the last purchase. Features are the information the model uses for its predictions.

3.5.1 What Makes a Good Feature

A useful feature helps models tell cases apart. For instance:

- In predicting loan repayment, income level, credit score, and existing debt are good features, they describe financial ability.
- The applicant's age or location might also help if they correlate with repayment risk.
- However, including hair color will not help the model (as you might imagine). This is a useless and bad feature.

Features can be numbers (like income), categories (like job type), time durations (like days since last purchase), or flags (yes/no values). Some features are directly measured, while others are *derived* from raw data, such as averages, counts, or time differences.

A subtle but very common issue in real projects is the use of **inconsistent or unclear units**. Data are often collected from different systems, teams, or countries, each using their own conventions. For example, one dataset may record height in centimetres while another uses metres; one branch of a company may store revenue in euros, another in dollars. If these differences are not harmonised before training, the model will interpret them as real variations rather than differences in scale. This can lead to bizarre results, such as the model thinking that a person 1.8 metres tall is shorter than someone listed as 180 (because it assumes both numbers are in the same unit). Similarly, a financial model might dramatically mispredict profits if some data are in thousands of euros and others in single euros. Models cannot "guess" which measurement system is correct; they simply take numbers at face value.

Another frequent mistake is including **too many weak or redundant features**. It is tempting to believe that adding more data columns will automatically make a model smarter, but this is rarely true. Features that add little unique information or strongly overlap with others can actually harm performance by introducing noise and distraction. For example, if we already have both "total spending last month" and "number of purchases last month," adding "average spend per purchase" gives no new insight, it is just a mathematical combination of the first two. Likewise, including hundreds of small, poorly understood indicators (like dozens of social media metrics) can make the model sensitive to random fluctuations rather than meaningful trends. In practical terms, a model overloaded with redundant features can become slower, harder to interpret, and more likely to overfit the training data. A sensible rule of thumb is to start simple: use a small set of features that make intuitive sense for the problem and that you can clearly explain to a non-expert. Only add new ones when you can justify why they might help. Good models are built on clear, relevant signals, not on a mountain of barely related numbers.

Intuitive Example

Example - Intuitive Example

Imagine a model predicting how long a food delivery will take. Useful features might include distance to customer, time of day, day of week, and number of active drivers nearby. But using box material as feature would be useless. A clean, realistic feature set focuses only on what can be known *before* the delivery begins.

3.5.2 Why Feature Engineering Matters

Feature engineering is where much of the real skill in machine learning lies. Even simple models can perform surprisingly well when the right features are chosen and cleaned. Conversely, even advanced algorithms fail if the input data are noisy, inconsistent, or unrealistic. In most real projects, time spent understanding, selecting, and improving features has more impact than switching to a more complex algorithm. In short: *better data beats fancier models.*

Exercise (10–12 min): Spot the Problem

Exercise 3.11

You are asked to build a model to predict whether a patient will visit the emergency department within 30 days after discharge. The proposed features are: age, diagnoses, average daily temperature next week, number of previous visits, and discharge notes. Which features are good or wrong and why?

3.6 Limits and Risks of No-Code Tools

Over the past few years, **no-code** and **AutoML** platforms have become increasingly popular. They promise to let anyone build machine learning models through a graphical interface, without writing code, and to automate many technical tasks such as feature selection, model training, and evaluation. These tools can be extremely useful for fast experimentation or for teams without programming expertise, but they also introduce serious limitations that can mislead non-specialists and create hidden risks when used in real-world decisions.

What No-Code Tools Are

No-code tools are visual environments where users can drag and drop data, select a goal (for example, "predict sales next month"), and let the system automatically train and test several models behind the scenes. They typically handle tasks like data cleaning, feature engineering, model comparison, and performance reporting without requiring the user to write a single line of code.

Examples include:

- **Google Cloud AutoML** – builds image, text, or tabular models automatically from uploaded data.
- **Microsoft Azure ML Designer** – offers a drag-and-drop interface to connect data sources, train models, and deploy them.
- **DataRobot** or **H2O.ai Driverless AI** – provide automated pipelines for business users with dashboards and leaderboards of model performance.
- **RapidMiner** or **KNIME** – older but still widely used visual workflow tools for machine learning and analytics.

In essence, these platforms try to replace manual coding and configuration with guided automation. A user uploads data, picks a target variable, and the system automatically chooses algorithms, tunes parameters, and outputs metrics such as accuracy or recall. However, this apparent simplicity can easily hide deep complexity and encourage overconfidence in results that have not been properly validated or interpreted.

Over-Simplification of Complex Problems

No-code systems often make machine learning look easier than it really is. A few clicks can yield a colourful dashboard showing high accuracy or a ready-to-deploy model, but what remains invisible is whether the data were appropriate, balanced, or temporally consistent. For example, a hospital administrator could upload patient records to predict readmissions and get impressive results, without realising that the model accidentally used future information (such as medication prescribed *after* discharge). The system would appear accurate, but it would fail completely in live use. By hiding steps like feature selection, data

leakage checks, and validation strategy, no-code tools can lead to false confidence and poorly generalised models.

Lack of Transparency and Control

Another serious limitation is that users cannot easily inspect what the tool has done internally. Most platforms do not reveal which features were used, how missing values were handled, or which algorithms were tried and why one was chosen. This "black box" approach makes it difficult to understand or explain results. When a model behaves strangely, for example, rejecting loan applicants from one region far more often than others, users have no way to trace back and understand the cause. This lack of transparency prevents effective debugging, makes reproducibility impossible, and undermines trust, especially in sensitive domains like healthcare or public policy.

Vendor Lock-In and Sustainability

No-code systems are usually proprietary. Once a workflow or model is created inside their platform, it often cannot be easily exported or reused elsewhere. This means that the organisation becomes dependent on that vendor for continued access, updates, and maintenance. If the company changes pricing, discontinues a service, or restricts access, critical logic may be lost. In practice, this can create "technical debt": systems that work only as long as a specific product remains available. For instance, a city government that builds a housing-allocation model in a closed cloud tool may later find that it cannot migrate the model to another provider without rebuilding it from scratch.

Ethical and Accountability Risks

Perhaps the most important risk is ethical. Because these systems automate modelling choices, users may not realise when they are training on biased, incomplete, or sensitive data. For example, a no-code HR screening tool trained on past hiring data could learn to reproduce existing gender or racial imbalances, even if the interface itself looks neutral. Similarly, a credit-risk model might unfairly penalise people from specific postal codes if the tool fails to correct for socioeconomic bias. Even if the model was built automatically, organisations remain legally and morally responsible for the outcomes it produces. No-code platforms do not remove accountability; they only make it easier to overlook the human judgment that must accompany automated decisions.

When and How to Use No-Code Tools Safely

No-code systems should be avoided if possible. They should not be used for high-stakes or regulated decisions without expert review. Machine learning involves statistical reasoning, ethical judgment, and domain expertise, none of which can

be automated away. Human oversight ensures that models are fair, transparent, and robust, and that they serve their intended purpose responsibly.

3.7 Evaluation Metrics in Practice

Models make mistakes in different ways. The goal of evaluation is not to find a perfect score, but to understand *how* the model fails and whether those errors are acceptable for the decision at hand. Every metric is a way of counting mistakes. Choose metrics that match the cost of being wrong in your setting, missing something important vs. raising too many false alarms. In Section 3.9 you will find, as a reference, the formulas for the metrics most used in machine learning.

3.7.1 For Yes/No or Category Decisions (Classification)

When you are classifying something (say, for example, images of cats and dogs, or emails as spam or non-spam), you are generally interested in finding out how many cases you get right and how many wrong. Probably, the most used metrics to measure how good (or bad) a model is of all are the following four. In the next sections we will discuss each in detail.

- **Accuracy** – the share of correct answers overall. Easy to grasp, but can be misleading if one outcome is very common (e.g., predicting "not-fraud" for everyone gives 99% accuracy but zero value).
- **Precision** – among the items the model flagged as positive, how many were actually right. Useful when false alarms are costly (e.g., wrongly rejecting good loan applications).
- **Recall (Sensitivity)** – among all the real positives, how many did the model catch. Important when missing a case is costly (e.g., failing to detect a disease).
- **F1 Score** – a simple balance between precision and recall. Handy when both missing and over-alerting are bad.

3.7.2 Understanding Accuracy (and Its Limits)

Accuracy is the simplest metric of all when doing classification: basically it measures the share of all predictions that are correct. Formally it is given by the following formula.

$$\text{Accuracy} = \frac{\text{Number of correctly identified cases}}{\text{Total number of cases}} \tag{3.1}$$

In plain terms, if a model makes 100 predictions and gets 95 right, its accuracy is 95%. Although this is easy to grasp, it can be seriously misleading when one outcome is much more common than the other. For example, in fraud

detection only about 1% of transactions may actually be fraudulent. A model that always predicts "not fraud" would be correct 99% of the time, giving a very high accuracy, but it would never detect real fraud. In such **imbalanced datasets**[2], accuracy hides the true weakness of the model because it rewards guessing the majority class. To judge performance fairly, we also need metrics that tell us how well the model recognises minority cases: **recall** (how many real frauds it caught) and **precision** (how often its fraud alerts were right) are the most used metrics and are discussed in Section 3.7.3). Accuracy is only informative when both classes are roughly balanced; in rare-event problems, it gives a false sense of quality.

3.7.3 Understanding Precision and Recall

When a model predicts categories (a classification problem), such as spam/not-spam, fraud/not-fraud, or disease present/absent, it can make two kinds of mistakes: *false alarms* (flagging something that is actually fine) and *missed detections* (failing to flag something that is actually a problem). **Precision** and **recall** are two complementary metrics that describe these errors and help us understand how the model behaves beyond using only accuracy.

Precision: "When the model says yes, how often is it right?" Precision tells us how trustworthy a positive prediction is. It measures the share of items flagged as positive that were truly positive. It is given by the following formulas.

$$\text{Precision} = \frac{\text{Number of correct positive predictions}}{\text{Total number of positive predictions made by the model}} \tag{3.2}$$

In plain terms, high precision means that most alerts raised by the model are correct. If a fraud detection model flags 100 transactions as suspicious and 90 of them are actually fraudulent, its precision is 90%. However, if it often raises false alarms (flagging many legitimate transactions), precision drops. This is critical in applications where each false alarm has a cost, for example, wrongly blocking customers' payments or sending unnecessary medical alerts.

Recall: "Out of all the real positives, how many did the model catch?" Recall (also called **sensitivity**) measures how good the model is at finding all actual positive cases.

$$\text{Recall} = \frac{\text{Number of correct positive predictions}}{\text{Total number of actual positive cases in reality}} \tag{3.3}$$

High recall means that the model detects most of the true positives; low recall means it misses many. For example, if there were 200 fraudulent transactions in total and the model correctly flagged 90, its recall is 45%. In problems like

[2] An inbalanced dataset is one in which you have much more cases in one cases than in the others.

disease detection or safety monitoring, missing a real case can be much worse than raising extra false alarms, so recall matters most.

Definition — Accuracy, Precision, and Recall

Accuracy tells us how often the model is right overall. It measures the share of all predictions that are correct, both the "yes" and the "no" cases. It is useful when both outcomes are equally common.

Precision answers the question: "When the model says *yes*, how often is it right?" It focuses on the quality of positive predictions. High precision means few false alarms, most of the flagged cases are truly positive.

Recall answers the question: "Out of all the real *yes* cases, how many did the model find?" It focuses on completeness. High recall means the model catches most of the real positives, even if it sometimes raises extra false alarms.

In short:

- **Accuracy** — how often the model is right overall.
- **Precision** — when it predicts something, how often it's correct.
- **Recall** — how many of the real cases it successfully finds.

The Trade-Off Between Precision and Recall. Improving one often reduces the other. If you make the model more cautious (only flagging when it is very confident), you get *fewer false positives*, precision goes up, but you also miss more true cases, so recall goes down. If you make it more sensitive (flagging even weak signals), you catch more positives (higher recall) but also raise more false alarms (lower precision). The right balance depends on context. For example:

- In **medical screening** or **safety monitoring**, missing a real case is costly ⇒ prioritise recall.
- In **spam filtering** or **loan approval**, false alarms are costly ⇒ prioritise precision.

Example - Precision vs. Recall

Imagine a spam filter. Consider the following situation.

- Out of 1000 emails, 100 are truly spam.
- Of those 1000, the model flags 120 as spam, 80 of which are correct.

We can calculate precision and recall.

$$\text{Precision} = \frac{80}{120} = 0.67\ (67\%), \qquad \text{Recall} = \frac{80}{100} = 0.80\ (80\%) \quad (3.4)$$

So the model catches 80% of all spam (good recall), but one in three flagged messages is actually not spam (moderate precision). If this filter were used in a corporate setting where missing spam is worse than misclassifying a few normal emails, this trade-off might be acceptable.

To reiterate, accuracy alone cannot tell whether a model is making the right kind of mistakes. Precision and recall together describe the *quality of the model's predictions* and whether it is cautious or aggressive in flagging positives. A well-balanced system depends on the context, and you decide whether it is better to miss a few correct predictions or to raise a few extra false alarms. To summarise

- **Precision** answers: "When I raise an alarm, how often am I right?"
- **Recall** answers: "Of all true alarms, how many did I catch?"

For many real-world systems, both metrics are monitored together and sometimes combined into a single balance score (the **F1 score**) that reflects how well the model balances precision and recall. But discussing the F1 score goes beyond the scope of this course. A quick summary table can be found in Table 3.4.

Metric	What it Tells You
Precision	"When I raise an alarm, how often am I right?" (avoid false positives)
Recall	"Out of all true alarms, how many did I catch?" (avoid false negatives)

Table 3.4: Precision and recall and what each metric tell you.

3.7.4 For Numbers and Forecasts (Regression)

When the model predicts a number rather than a category, such as sales next week, delivery time, or energy use, we need to measure how close its predictions are to the expected values (the labels). We do this by using **regression metrics**, which capture different ways of describing prediction errors. Some focus on the *average size* of mistakes, others on how much large errors matter, and some on how well the overall pattern of variation in the data is explained. The most common are summarised below. For the formulas, let us indicate with y_i the true labels (what we expect) and with $\hat{y}_i$ the predicted output for the i^{th} input.

- **Mean Absolute Error (MAE)**: how far off, on average, the model's predictions are. Easy to interpret in the same units as the outcome (e.g.,

"the prediction of the temperature is off by $5°C$ on average."). Its formula is

$$\text{MAE} = \frac{1}{n}(|y_1 - \hat{y}_1| + |y_2 - \hat{y}_2| + \cdots + |y_n - \hat{y}_n|) \tag{3.5}$$

assuming we have n data inputs. The symbol $|\cdot|$ indicates the absolute value, which simply means converting any number into its positive part. So for example $|3| = 3$ but $|-3| = 3$. In intuitive terms the MAE measures how far the model is from the expected output on average[3].

- **Mean Squared Error (MSE)**: this metric is based on a similar idea to the MAE but sums not the absolute values of the errors but their square according to the following formula.

$$\text{MSE} = \frac{1}{n}((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_n - \hat{y}_n)^2) \tag{3.6}$$

  This types of metric enhance large errors. In fact large errors becomes even larger when squared!

- **Explained Variation (R^2)**: the R-squared (R2 or R^2) is a number that tells you how well the independent variable(s) in a statistical model (inputs) explains the variation in the dependent variable (predictions). A higher number means the model works better, a lower number the model is not working well (an intuitive understanding will be enough, if you are interested in a more mathematical discussion you can check Section 13.2.2. in [35]).

Definition — MAE and MSE (Measuring Prediction Errors)

When a model predicts a number (such as a price, delay, or temperature), we can measure how far its predictions are from the real values. Two common ways to do this are the **Mean Absolute Error (MAE)** and the **Mean Squared Error (MSE)**.

Mean Absolute Error (MAE) measures how large the errors are on average, treating all mistakes equally. It is defined as:

$$\text{MAE} = \frac{1}{n}\left(|y_1 - \hat{y}_1| + |y_2 - \hat{y}_2| + \cdots + |y_n - \hat{y}_n|\right)$$

Here:

- y_i is the true (real) value,
- $\hat{y}_i$ is the model's predicted value,
- n is the total number of predictions.

[3] Remember we are in supervised learning, so our datasets consists in inputs and labels, or expected outputs.

The result tells us, on average, how far off the model is, using the same units as the data (for example, "5 minutes off on average").

Mean Squared Error (MSE) also measures the distance between predictions and reality, but it gives *more weight to big mistakes* by squaring each one:

$$\text{MSE} = \frac{1}{n}\left((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \cdots + (y_n - \hat{y}_n)^2\right)$$

Because large errors get squared, they count much more strongly in the final score. This makes MSE useful when large misses are especially costly or risky.

In short:

- **MAE** — average size of the model's mistakes, treating all equally.
- **MSE** — same idea, but large mistakes count much more.

3.7.5 Choosing the Right Metric

In general the problem you are trying to solve, dictates which metric is the right one. In Table 3.5 you can find a short overview that might help you in choosing the right metric.

3.8 Interpretability and Human-in-the-Loop

Models support human decisions, they do not replace accountability or judgement. Even when a model performs well, we still need to understand *why* it gives certain answers and to keep people involved wherever outcomes matter for safety, fairness, or ethics.

A model may find patterns that humans cannot see, but if its reasoning cannot be understood, it is hard to trust or improve. **Interpretability** means being able to explain in simple terms what drove a prediction. For example, a hospital triage tool might explain that a patient was flagged as high risk mainly because of age, recent admissions, and certain symptoms. The goal is not to read the mathematics, but to make the reasoning checkable and sensible to domain experts. Instead of focusing on abstract "features" or technical graphs, think of explanations as short stories about what the model noticed. We can ask:

- Which pieces of information most influenced this particular prediction?
- Are those factors reasonable for this context?
- Do similar cases get treated consistently?

Scenario (example)	Focus On (Primary Metric)	Secondy Metric
Disease screening (rare)	**Recall:** catch as many real cases as possible.	**Precision:** too many false positives waste time and resources.
Content moderation	**Precision:** avoid removing safe or neutral content.	**Recall:** must still catch severe harm or abuse.
Credit risk modelling	**Calibration:** predicted probabilities must reflect true default risk.	**Fairness:** performance and approval rates should be balanced across groups.
Fraud detection	**Recall:** identify most fraudulent cases.	**Precision:** limit false alarms on legitimate customers.
Demand forecasting	**MAE:** typical size of the error in units sold.	**Bias:** check for systematic under- or over-prediction by region or season.
Cost estimation or energy use	**MSE/RMSE:** penalise large misses more heavily.	**Outliers:** extreme values can dominate this metric.
Model fit (regression)	R^2**:** proportion of variation in the outcome explained by the model.	**Data quality:** high R^2 can still hide leakage or bias.

Table 3.5: Examples of suitable metrics and complementary guardrails across different modelling contexts.

These checks help humans spot when the model is relying on misleading or unfair patterns.

Interpretability also means checking performance across different **segments or cohorts** (such as by age, site, region, or season). If a model performs well for one group but poorly for another, it may be unfair or unreliable. Simple tables or charts showing results by group can reveal such gaps and guide improvements. No automated system should make sensitive decisions entirely on its own. Human oversight remains essential to:

- **Review unusual or borderline cases** that models may struggle with.
- **Override or adjust decisions** when local knowledge suggests the model is wrong.
- **Provide feedback** to improve future versions, such as correcting wrong labels or identifying new factors.

This collaboration between people and models combines the model's speed and consistency with human context, ethics, and common sense.

It is important to note that interpretability is not about technical transparency. It is about *making the model's behaviour understandable enough for responsible use.* The model can assist, but humans must stay accountable for decisions and for checking that the system continues to behave safely and fairly.

3.9 Advanced Reference: Metric Definitions (Optional)

Purpose. These formulas are provided for readers who wish to see the formal definitions behind the metrics introduced earlier. You do not need to memorise them, they are included as a mathematical reference.

Classification Metrics

Let:

- TP = true positives (model predicted positive and it was positive)
- FP = false positives (model predicted positive but it was negative)
- TN = true negatives (model predicted negative and it was negative)
- FN = false negatives (model predicted negative but it was positive)

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall (Sensitivity)} = \frac{TP}{TP + FN}$$

$$\textbf{Specificity} = \frac{TN}{TN + FP}$$

$$\textbf{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Regression Metrics

Let y_i be the true value, $\hat{y}_i$ the predicted value, and n the number of samples.

$$\textbf{Mean Absolute Error (MAE)} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\textbf{Mean Squared Error (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\textbf{Root Mean Squared Error (RMSE)} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\textbf{Coefficient of Determination } (R^2) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the true values.

Interpretation Summary.

- $\uparrow$ Precision: fewer false alarms.
- $\uparrow$ Recall: fewer missed detections.
- $\uparrow$ F1: balanced trade-off.
- $\downarrow$ MAE/MSE/RMSE: smaller average or squared errors.
- $R^2 \in (-\infty, 1]$: higher is better; can be negative if the model performs worse than predicting the mean.

Glossary

Classification
: Tasks where the output is a *category* (e.g., spam vs. not-spam; approve vs. deny). The model often also produces a probability for each class.

Regression
: Tasks where the output is a *number* (e.g., price, time-to-delivery, demand next week).

Clustering
: Grouping similar items without labels to discover natural segments or patterns in the data.

Feature
: An input variable (signal) used by the model (e.g., age, distance, days since last purchase, word count).

Label / Target
: The outcome we want to predict (class for classification; numeric value for regression).

Probability Threshold
: A cut-off on predicted probability used to turn scores into yes/no decisions (e.g., flag if $p \geq 0.8$).

Confusion Matrix
: A 2×2 table counting correct/incorrect decisions (true/false positives/negatives) for a binary classifier.

Accuracy
: Share of all predictions that are correct. Can be misleading with rare events.

Precision
: Of the items the model flagged as positive, the fraction that were truly positive (*"when I raise an alarm, how often am I right?"*).

Recall (Sensitivity)
: Of all truly positive items, the fraction the model successfully flagged (*"out of all real alarms, how many did I catch?"*).

F1 Score
: A single number balancing precision and recall (harmonic mean). Useful when both types of error matter.

Calibration
: How well predicted probabilities match reality (e.g., among cases with 70% score, about 70% should be positive).

MAE (Mean Absolute Error)
: Average absolute difference between predictions and actual values (same units as the target).

MSE / RMSE
: Mean of squared errors / its square root. Penalise large mistakes more than MAE; RMSE has target units.

R^2 (Explained Variation)
: Proportion of variation in the target explained by the model (1 is perfect; can be negative if worse than predicting the mean).

Baseline
: A simple reference to beat (e.g., majority class for classification; last week's value or average for forecasting).

Overfitting
: Model learns training quirks and performs poorly on new data (train score $\gg$ test score).

Underfitting
: Model is too simple to capture real structure (poor on both train and test).

Model Validation
: Testing how well a model generalises to unseen data; the antidote to over-optimistic results.

Hold-Out Split
: Single split of data into train/validation(/test) sets; the test set stays untouched until the end.

Train/Validation/Test
: *Train* to learn; *validation* to tune and choose thresholds; *test* to estimate final performance.

Cross-Validation
: Repeat training/testing on multiple splits to get a more stable performance estimate (e.g., k-fold; repeated random splits).

K-Fold Divide data into k parts; train k times, each time holding out a different part for testing; average the scores.

Monte Carlo CV
: Repeated random train/test splits (e.g., 80/20) with performance averaged across repeats.

Data Leakage
: Using information not available at decision time (or derived from the outcome), which inflates validation scores but fails in practice.

Split Strategies
: **Random** (independent cases), **time-based** (train on earlier, test on later), **group-aware** (keep entities from crossing splits).

Segment / Cohort
: A subgroup (e.g., site, region, device, age band). Checking results by cohort helps spot fairness gaps or drift.

Fairness (High Level)
: Performance and decisions should not systematically disadvantage protected or meaningful groups; check metrics across cohorts.

Human-in-the-Loop
: People review edge cases, can override decisions, and feed back corrections to improve future versions.

Interpretability
: Explaining, in simple terms, what influenced a prediction (local explanation) or how the model behaves overall.

Drift
: When data or relationships change over time, causing performance to degrade; requires monitoring and periodic recalibration/retraining.

Ordinal / Multi-Class / Multi-Label
: **Ordinal:** ordered labels (low/medium/high). **Multi-class:** one of many exclusive labels. **Multi-label:** several labels can apply at once.

One-vs-Rest / One-vs-One
: Schemes to solve multi-class problems with binary pieces: OvR trains one model per class vs. all; OvO trains one per class-pair.

No-Code / AutoML
: Tools that automate parts of ML (feature handling, model search) via a graphical interface. Fast for exploration; require careful validation and oversight.

Metric (Primary & Guardrail)
: **Primary** reflects main cost (e.g., recall for safety, MAE for money/units). **Guardrail** watches side-effects (e.g., precision, fairness, calibration).

Chapter 4

Understanding and Designing AI Projects: From Idea to Prototype

Contents

4.1 Phases of an AI Project

Designing an AI project means translating an idea into a structured, evidence-based process. While every project is different, most follow a similar sequence of steps — from the first question to a working prototype. For this course, the goal is not to build a perfect model, but to learn how to think, plan, and communicate like an AI project designer.

1. Define the Problem and Value

- Start with a clear and concrete question: *What decision, prediction, or insight do we want to support?*
- Identify who benefits (user, customer, organization) and how success will be measured.
- Keep it small: focus on one user, one data source, one measurable outcome.

2. Understand the Context

- Clarify where the project sits: is it *research*, *technology development*, or *product work*?
- Decide whether it explores feasibility ("Can it work?") or delivers value ("Does it help users?").
- Position it in the *technology life cycle*: introduction, growth, maturity, or decline. This helps set realistic goals and timelines.

3. Identify and Explore the Data

- Determine what data are needed to answer the question and whether they already exist.
- Assess accessibility, quality, and privacy: who owns the data, and are you allowed to use them?
- Remember that for prototypes, small and simple datasets (even spreadsheets) are perfectly acceptable.

4. Plan the Approach

- Choose the most suitable AI or ML family: supervised (predict labels), unsupervised (discover patterns), reinforcement (learn from feedback), or generative (create content).
- Think conceptually: What input → output relation should the system learn?
- Draft a minimal *pipeline*: data collection → preparation → training → evaluation → use.

5. Build a Prototype (Proof of Concept)

- Use accessible tools (e.g., no-code/low-code or cloud notebooks) to test your idea quickly.
- Document assumptions, data sources, and choices transparently — the prototype should explain the logic, not just show results.
- Expect to iterate: early failure and revision are normal parts of AI work.

6. Evaluate and Learn

- Assess performance using simple, interpretable metrics (accuracy, precision, coverage, etc.).
- Discuss what worked, what did not, and why — link results back to the original question and success criteria.
- Reflect on ethical, privacy, and fairness implications before any deployment.

7. Communicate and Decide Next Steps

- Prepare a short project summary ("AI Blueprint") showing problem, data, method, metric, risks, and benefits.
- Present findings as a story: problem → evidence → prototype → recommendation.
- Decide: continue to develop, adjust the question, or stop and document the learning.

Summary: The AI Project Flow

Idea → Question → Data → Prototype → Evaluation → Decision

Each phase builds on the previous one. A well-designed AI project does not start with code — it starts with a clear purpose, good data understanding, and structured reasoning. That mindset is the foundation for every successful AI initiative.

4.2 Roles and Competencies in the Team

- Typical roles: domain expert (knows the problem), data/AI specialist (knows the methods), and project facilitator (keeps goals, ethics, and communication aligned).
- Stress cross-functional collaboration and "translation" between technical and business language.
- Highlight the importance of data literacy and critical thinking for everyone, not coding.

4.3 Data, Infrastructure, and Cloud Platforms

- Explain that most AI projects today rely on cloud infrastructure — shared online environments for data storage, model training, and collaboration.
- Describe the basic layers: data (storage and access), compute (running models), and tools (interfaces for analysis and visualization).
- Emphasize practical entry points: using cloud notebooks (e.g., Google Colab, Azure ML, or AWS SageMaker) and low-code platforms for rapid prototyping.
- Highlight organizational considerations: data privacy, cost control, and clear responsibilities for who manages data and services.
- Encourage students to think of the cloud as an *enabler* — it reduces setup effort, supports teamwork, and allows scaling ideas from laptop to enterprise.

4.4 Integrating AI: Opportunities and Risks

- Opportunities: automation of repetitive tasks, better forecasts, improved customer or citizen experience.
- Risks: bias, opacity, over-reliance on automated decisions, and data misuse.
- Encourage critical reflection and “human-in-the-loop” design as default.

4.5 Preparing Project Work

- Help students define one realistic use case and scope it with a clear question, data source, and success metric.
- Use a short project canvas: problem, data, method, metric, risks, and timeline.
- Stress storytelling: how to explain your AI idea clearly to non-experts or decision-makers.

Final Project (Capstone)

Contents

A1 Report

A1.1 Purpose and Scope

This capstone asks you to apply the module's concepts to a small, realistic use case. Your goal is not to build a model but to make *responsible, well-reasoned choices*: frame the problem, define inputs (*features*), discuss how to split data honestly, pick fit-for-purpose metrics, reflect on risks, and explain how a human stays in the loop.

A1.2 Deliverables (what to submit)

- A **5–8 page report (PDF)** following the structure below (figures/tables encouraged).
- A **one-page appendix** with any extra figures (optional).
- If you used a no-code/AutoML tool, add a **brief tool note** (1/2 page): settings used, what was automated, and any limits you noticed.

A1.3 Suggested Report Structure

1. **Executive Summary (1/2 page).** One paragraph on the problem and why it matters; one paragraph on your approach and the main result (e.g., "precision 0.86 at 0.30 recall; MAE 4.8 units"); one sentence on risks and what you would do next.
2. **Problem Framing (3/4 page).** Name the decision you aim to support (e.g., flag a transaction for review; forecast next week's demand). State whether this is *classification*, *regression*, or *clustering* and why. Define the *label/target* in plain words. Clarify the decision timing (what is known *before* the prediction).
3. **Data and Features (1 page).** Briefly describe the dataset: source, size, time span, any known quirks. List 5–10 key *features* with one-line justifications each (why might this help the decision?). Explicitly note any units and conversions to avoid confusion (cm vs. m, EUR vs. USD).
4. **Validation Plan (3/4 page).** Describe your split choice and why (hold-out by time; random split; group-aware). Include a tiny diagram or bullet timeline (train → validation → test). State what you used the validation set for (thresholds, model selection) and what remained untouched for final testing.
5. **Metrics and Baselines (3/4 page).** Choose *one primary* metric that match the costs of errors in your setting (e.g., recall + precision; MAE + seasonal error). Explain in 3–4 sentences why these choices make sense for stakeholders.

6. **Fairness, Risk, and Human-in-the-Loop (1/2–1 page).** Identify who might be impacted. Note any fairness checks you think are important (for example checking that a model behaves in the same way for men and women).

7. **Limits and Next Steps (1/2 page).** Be candid about data limits (coverage, noise, possible leakage you ruled out), model limits (where it fails), and realistic next steps (better features, more data, different split, recalibration, additional guardrails).

A1.4 Allowed Tools (short note)

You may use spreadsheets, basic Python/Colab, or a no-code/AutoML tool. If you use no-code to analyse real data, include a 1/2-page tool note: what it automated (feature handling, model search), any settings you changed, and one limitation you observed. Remember: *use no-code for exploration, not blind deployment.*

A1.5 Ethics and Data Care (one paragraph)

Do not include personal or sensitive information.

A2 Capstone Video Pitch

Goal

Deliver a concise, professional summary of your project for non-technical stakeholders. Record yourself presenting your slides (e.g., PowerPoint *Record Slide Show*); quick, single-take is fine. No editing needed.

Format and Timing

- **Length:** 3–5 minutes total.
- **Slides:** ca. **5-7 slides** (title included).
- **Pacing:** $\approx$ 30–45 seconds per slide.
- **Delivery:** your voice over the slides; your camera is optional.

Content Outline (what to show & what to say)

1. **Title & Decision**
   - *Show:* Project title, team name, date, one-sentence problem statement.
2. **Data & Features (no leakage)**

- *Show:* Source, size, time span; list 5–7 key features (units clear).

3. **Validation & Metrics**
   - *Show:* Tiny split diagram (train → validation → test); primary metric (e.g., recall + precision; MAE + seasonal error). Why this choice?
4. **Fairness, Human-in-the-Loop, Next Steps**
   - *Show:* One cohort/segment check (e.g., by region/site/time) in a tiny table or bar; bullets for human review/override and monitoring.
   - *Say:* Who is impacted; how humans review edge cases; what you would monitor post-deployment (drift, error spikes); top 1–2 next steps.

Design Standards (keep it professional)

- **One idea per slide.** Minimise text (5–7 bullets max; 6–10 words each).
- **Readable fonts:** Titles $\geq 32\,\text{pt}$; body $\geq 20\,\text{pt}$; consistent typeface.
- **High contrast:** Dark text on light background (or the reverse). Avoid busy backgrounds.
- **Simple visuals:** One figure per slide (table or chart). Label axes and units. Avoid 3D effects.
- **Colour care:** Do not rely on colour alone; add labels. Prefer colourblind-friendly palettes.
- **Branding:** Add your name(s) and course in the footer.

Speaking Tips (no editing required)

- **Stakeholder language:** Explain metrics in words (*"precision: when we raise an alarm, how often we are right"*).
- **Tempo & clarity:** Slow enough to follow; pause between slides; avoid filler (like "um", "ehm", etc.).
- **Audio:** Quiet room; test mic once; keep laptop 30–40 cm away; speak towards the mic.

How to Record (PowerPoint)

1. Open your deck → *Slide Show* → *Record Slide Show*.
2. Check microphone input; optional camera on/off.
3. Advance slides while speaking; keep within 3–5 minutes.
4. Export: *File* → *Export* → *Create a Video* (1080p) *or* export as MP4 directly from the Record window.

Submission Requirements

- **Video:** MP4, 3–5 minutes, filename `Team_Name_Capstone.mp4`.
- **Report:** as described in this chapter.

Bibliography

[1] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[2] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[3] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[4] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[6] Terrence J Sejnowski and Charles R Rosenberg. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1):145–168, 1987.

[7] Yann LeCun et al. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2:396–404, 1990.

[8] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Vladimir N Vapnik and Alexey Ya Lerner. A class of algorithms for learning pattern recognition. *Automation and Remote Control*, 24(6), 1964.

[11] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[14] Yoav Freund and Robert E Schapire. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14:771–780, 1999.

[15] Jerome H Friedman. Greedy function approximation: A gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[18] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[19] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[20] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12:1057–1063, 2000.

[21] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[22] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[29] Ian J. Goodfellow et al. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[31] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[32] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Enhanced higgs boson to $\tau+ \tau-$ search with deep learning. *Physical review letters*, 114(11):111801, 2015.

[33] Andrew W. Senior et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577:706–710, 2020.

[34] Umberto Michelucci. *Fundamental Mathematical Concepts for Machine Learning in Science*. Springer.

[35] Umberto Michelucci. *Statistics for Scientists: A Concise Guide for Data-driven Research*. Springer Nature Switzerland, Cham, 2025.

Index