Machine Learning Intuition

Eliott Kalfon

2026-01-04

Table of contents

Preface

Machine Learning and AI have taken our world by storm. And yet, so few people truly understand these technologies.

I have spent a large part of my career solving practical problems with Machine Learning. This book is my attempt to explain the intuition behind those solutions for family, friends and curious readers.

If you want to understand how an algorithm can learn from examples and make useful predictions, welcome! You are in the right place. Expect clear explanations and simple visualisations rather than heavy mathematics.

The first part is a self-contained introduction to prediction with Machine Learning. It explains the core ideas you need to make sense of models that learn from data.

The following chapters explore important aspects of Machine Learning practice — for example: modelling approaches, model evaluation, and data preprocessing.

The final section brings these pieces together with an accessible end-to-end Machine Learning project.

The last chapter reflects on how modern generative AI models connect with the ideas presented here.

Notes

- **Intuition first:** the book focuses on intuition and real-world thinking rather than formal proofs. If you would like a more code-heavy or mathematical approach, please get in touch.
- **Synthetic data:** all examples use synthetic data for clarity. Nothing in this book is medical, legal or financial advice.

About Me

I am an Applied Science Manager and Computer Science Lecturer in Berlin. I have a blog and newsletter, subscribe here to read my latest work.

Part I

Introduction to Machine Learning

Chapter 1

First Prediction with Machine Learning

With Machine Learning, we can generate predictions based on historical data. These predictions can solve problems in the real world.

Imagine this scenario: you are an early-career doctor and you receive a biopsy report. The lab lists **two measurements** from the cell nuclei of a suspicious mass — an average perimeter of **100 µm** and an average area of **1200 µm²**. From just those two numbers: **is the mass more likely benign (non-cancerous) or malignant (cancerous)?**

If that vocabulary sounds unfamiliar, don't worry — the question is straightforward: **can we predict the diagnosis of this tumour from two measurements**?

1.1 A visual approach

At first it might feel impossible. But suppose we have a database of many tumours, each labelled benign or malignant. A natural first step is to **plot those examples** using the two measurements from the report: perimeter and area. Each point on the chart would carry a label.

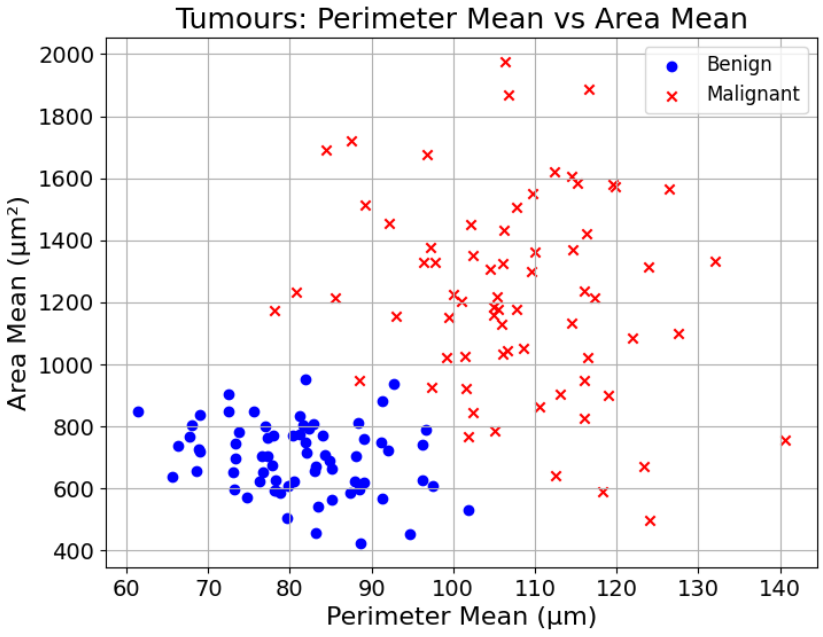


Figure 1.1: Plotting historical observations

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

benign_center = [80, 700]
malignant_center = [110, 1200]

n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando

benign_std = [10, 120]
malignant_std = [12, 300]

X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

plt.figure(figsize=(8,6))
```

```
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malignant')
plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.title('Tumours: Perimeter Mean vs Area Mean', fontsize=18)
plt.legend(fontsize=12)
plt.grid(True)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

We could then plot the new observation on the same chart. We label it with a question mark as its diagnosis is still **unknown**.

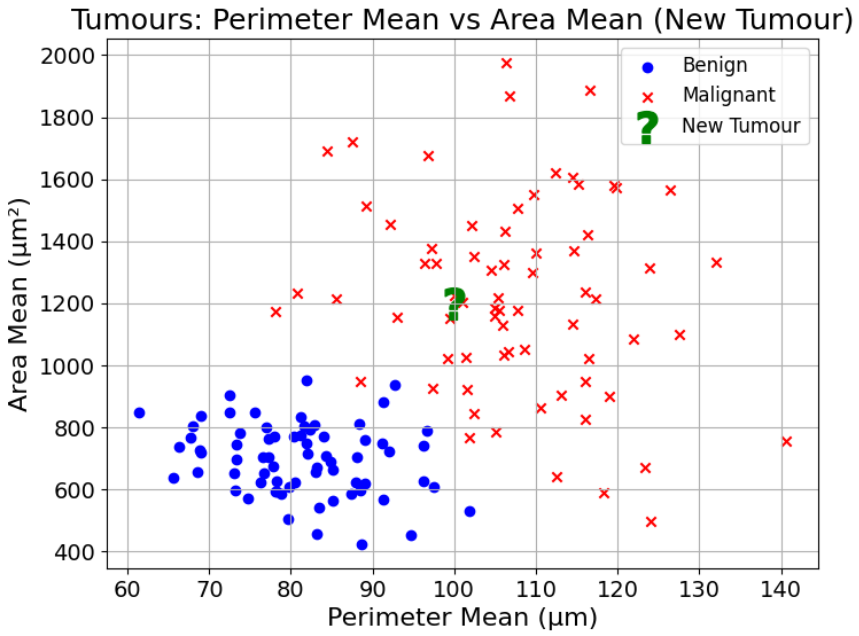


Figure 1.2: Looking at the unknown observation

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs

benign_center = [80, 700]
```

```
malignant_center = [110, 1200]

n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando

benign_std = [10, 120]
malignant_std = [12, 300]

X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

plt.figure(figsize=(8,6))
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Maligna

plt.scatter(100, 1200, marker=r'$\mathbf{?}$', color='green', s=400, label='New Tumour

plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.title('Tumours: Perimeter Mean vs Area Mean (New Tumour)', fontsize=18)
plt.legend(fontsize=12)
plt.grid(True)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

Would you now feel comfortable making an educated guess?

Look carefully: most of the unknown point's nearest neighbours are malignant. From the data's perspective, the most reasonable guess is that the tumour is malignant. That is not a certainty — it's a probabilistic judgement based on the available examples.

1.2 From intuition to algorithms

How could **computers** do the same? Computers do not (yet) have eyes, or an understanding of distance and closeness.

A prediction algorithm would need to find an observation's **closest neighbours**. To do so, it would need to measure the distance between this observation and others, and pick the ones with the **shortest distance**.

These distance calculations will be studied later. As a brain teaser, how would you calculate the distance between point A and B on this figure?

Hint: the Pythagorean theorem should be useful

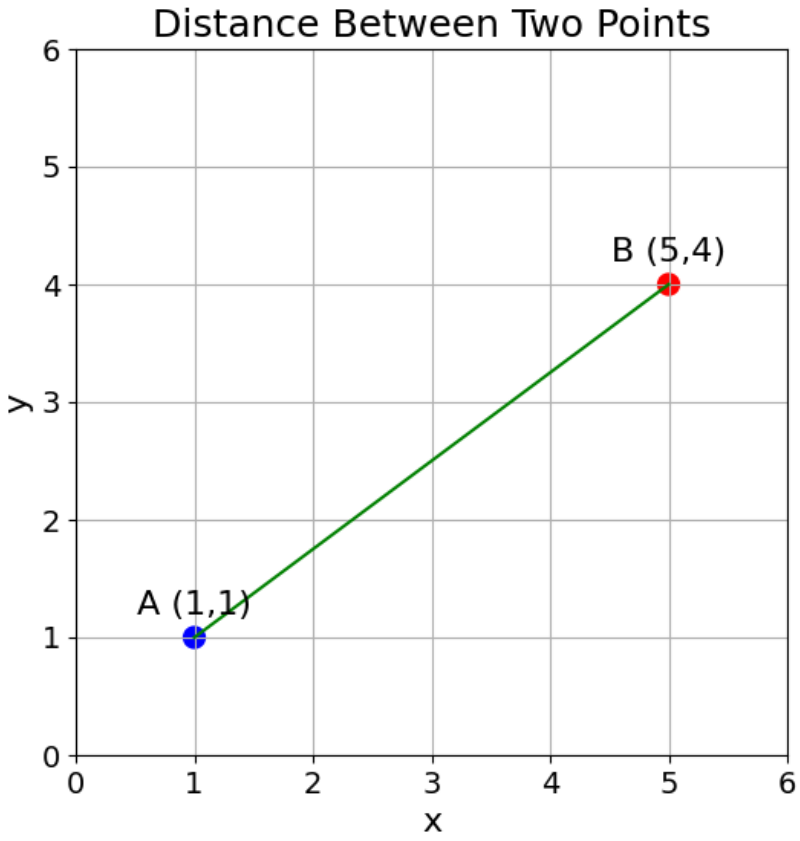


Figure 1.3: Calculating the distance between A and B

Figure code

```
plt.figure(figsize=(6,6))
plt.scatter([1,5], [1,4], color=['blue','red'], s=100)

plt.plot([1,5],[1,4],'g-')
plt.text(1,1.2, 'A (1,1)', fontsize=16, ha='center')
plt.text(5,4.2, 'B (5,4)', fontsize=16, ha='center')
plt.xlabel('x', fontsize=16)
plt.ylabel('y', fontsize=16)
plt.title('Distance Between Two Points', fontsize=18)
plt.xticks(fontsize=14)
```

```
plt.yticks(fontsize=14)
plt.xlim(0, 6)
plt.ylim(0, 6)
plt.grid(True)

plt.show()
```

Answer

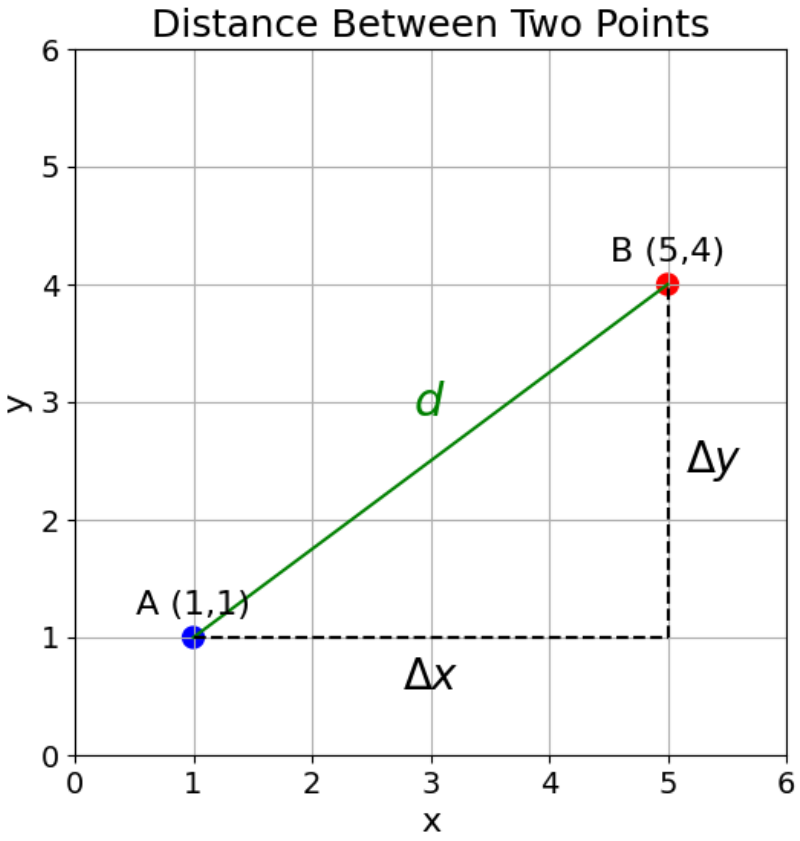


Figure 1.4: Thinking of distance as the hypotenuse of a right-angle triangle

To find the distance between point A $(1, 1)$ and point B $(5, 4)$ using the Pythagorean theorem, we can consider these points as two vertices of a right-angled triangle.

The horizontal distance (Δx) between the points is:

$$\Delta x = x_2 - x_1 = 5 - 1 = 4$$

The vertical distance (Δy) between the points is:

$$\Delta y = y_2 - y_1 = 4 - 1 = 3$$

According to the Pythagorean theorem, the square of the hypotenuse (the distance d between points A and B) is equal to the sum of the squares of the other two sides (Δx and Δy):

$$d^2 = (\Delta x)^2 + (\Delta y)^2$$

Substituting the values:

$$\begin{aligned} d^2 &= (4)^2 + (3)^2 \\ d^2 &= 16 + 9 \\ d^2 &= 25 \end{aligned}$$

To find d, we take the square root of both sides:

$$d = \sqrt{25} = 5$$

The distance between point A $(1, 1)$ and point B $(5, 4)$ is 5.

With the list of closest neighbours, how would you make a **prediction**? Before coming up with an answer, think of how you would classify the following examples:

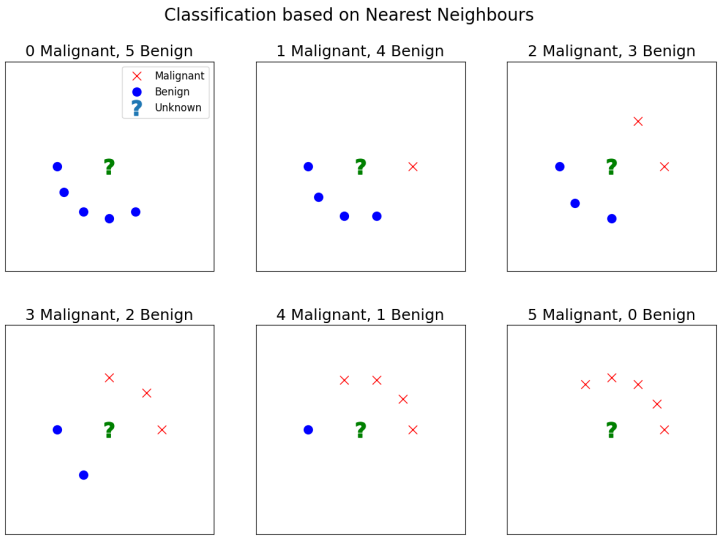


Figure 1.5: Different neighbour scenarios

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

def plot_neighbours(num_malignant, ax, total_neighbours=5):
    num_malignant = min(num_malignant, total_neighbours)
    num_benign = total_neighbours - num_malignant
    angles_malignant = np.linspace(0, np.pi, num_malignant + 1, endpoint=False)[:-1] i
    angles_benign = np.linspace(np.pi, 2*np.pi, num_benign + 1, endpoint=False)[:-1] i
    radius = 0.5

    # Plot malignant neighbours
    for i in range(num_malignant):
        ax.plot(radius * np.cos(angles_malignant[i]), radius * np.sin(angles_malignant

    # Plot benign neighbours
    for i in range(num_benign):
        ax.plot(radius * np.cos(angles_benign[i]), radius * np.sin(angles_benign[i]),

    # Central unknown point
    ax.plot(0, 0, 'g', markersize=18, marker=r'$\mathbf{?}$')
```

```
    ax.set_title(f'{num_malignant} Malignant, {num_benign} Benign', fontsize=18)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_xlim([-1, 1])
    ax.set_ylim([-1, 1])
    ax.set_aspect('equal', adjustable='box')

# Create a 2x3 subplot layout
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Classification based on Nearest Neighbours', fontsize=20, y=0.98)

# Flatten the axes array for easier iteration
axes = axes.flatten()

for i in range(6):
    plot_neighbours(i, axes[i])

# Add a legend to the first subplot
axes[0].plot([], [], 'rx', markersize=10, label='Malignant')
axes[0].plot([], [], 'bo', markersize=10, label='Benign')
axes[0].plot([], [], ' ', markersize=18, marker=r'$\mathbf{?}$', label='Unknown')
axes[0].legend(loc='upper right', fontsize=12)


plt.subplots_adjust(top=0.9)
plt.show()
```

As you may have figured out, a **simple majority vote** should do. To generate predictions, count the number of neighbours belonging to each diagnosis, and **assign the diagnosis with the highest number of neighbours**.

That is it! We have our first **model** that can **generate predictions** about the world based on **historical observations**. This method could generate a diagnosis for any observation based on the value of the two measurements of its cell nuclei: average area and average perimeter.

This is only our first step into the fascinating world of Machine Learning. ML models can map **any input**, such as the above measurements, **to any output**, like tumour diagnosis.

Ready? Let's get started.

Chapter 2

Defining Prediction

2.1 The Prediction Task

Pre-diction, or, "to say before". To be able to describe the future, to **make the unknown known**.

Predictions are all around us. When you unlock your phone and look at the weather forecast (another word for "prediction"). When you open your email inbox, a prediction algorithm has already classified your incoming messages as "spam" or "non-spam". But you do not need your phone to be exposed to predictions.

In many regards, prediction is indistinguishable from perception. When we perceive the world around us, we are constantly predicting what we will be **perceiving next**. For example, as you read this sentence, you are predicting its next ____.

Thinking about predictions, there are two main prediction tasks:

- **Regression:** prediction of a continuous value, a real number
  - The temperature tomorrow
  - The price of a property
- **Classification:** prediction of class labels
  - Emails: "spam" or "non-spam"
  - Images: "dog" or "cat"
  - Type of tumour: "malignant" or "benign"
  - The following word in this ____

Exercise 2.1. Come up with more examples of problems for:

1. Classification
2. Regression

We do not necessarily need Machine Learning to make these predictions. Historically, we have relied on both (1) hand-crafted rules and (2) human intuition.

2.1.1 Hand-crafted rules

These hand-crafted rules are simplified models of reality. As an example, to price a property, I could simply multiply the **average price per square metre** in the neighbourhood by the number of square metres of the property. Rules like this one can work surprisingly well.

The number of special characters in an email address can be a relatively reliable spam filter. For instance, an address like `!_!urgent$!_deal@secure.offer.xyz` immediately raises red flags due to its chaotic combination of **punctuation** and a suspicious, **non-standard top-level domain**. These red flags could be coded into the spam filter program to classify incoming messages as "spam".

2.1.2 Human intuition

The Merriam-Webster dictionary defines intuition as:

> "intuition, noun:
>
> a. The power or faculty of attaining direct knowledge or cognition without evident rational thought and inference
>
> b. immediate apprehension or cognition"
> (Merriam-Webster Dictionary 2024b)

Intuition is access to knowledge, making the unknown known, **without apparent effort**. It is generally built over time from the following (Parrish 2016):

- A slow and unchanging environment
- Lots of practice and a large sample size
- Frequent and accurate feedback

Making this concrete, after 20 years of experience, a good oncologist can spot a malignant tumour on an x-ray without even having to think about it. They have seen so many examples that they have built an intuition over time. A seasoned real estate agent (in a slowly changing market) can "feel" the price of a property. An experienced recruiter can spot highly talented individuals after only a short conversation.

Looking at the example of spam prevention, if I receive an email from the address `!_!urgent$!_deal@secure.offer.xyz` telling me I have won the lottery and just need to click a link to claim my prize, I will be sceptical. I do not need to use explicit rules for this, human experience is enough. (This may not apply to my grandparents)

2.1.3 Drawbacks

The world is complex, messy, and in a state of constant change.

- Complexity makes building a rule-based system nearly impossible
- Constant change means that market trends fluctuate, attackers learn to circumvent simple spam-detection algorithms, and new types of diseases emerge. Organisations relying on rules or human intuition must constantly update their approach
- Building intuition is a costly and time-consuming process tied to specific individuals
- Human performance is inconsistent, we all have bad days

Machine Learning systems are not perfect either, but address many of these drawbacks.

2.2 What is Machine Learning?

Now, how is Machine Learning (ML) different?

Machine Learning can be defined from its terms:

- **Machine:** a computerised system, not human
- **Learning:** a system that adapts to data, not a rule-based method

As its name indicates, ML algorithms **learn** to predict either continuous values or class labels from historical data. As an example, a Machine Learning **algorithm** or **model** would learn the relationship between the features of a property and its price, or the dimensions of a tumour and its type ("malignant" or "benign"). Exactly how this learning happens is the purpose of this book, but let's not get ahead of ourselves.

2.2.1 Machine Learning Models

Throughout this book, Machine Learning "model" and "algorithm" will be used interchangeably. A model can be defined by what it does. In the context of predictive Machine Learning, it takes an **input** and outputs a **prediction**.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

Adapting this framework to the tumour diagnosis example, the model takes tumour measurements as features and outputs a diagnosis:

$$\text{Tumour Measurements} \longrightarrow \text{Model} \longrightarrow \text{Diagnosis}$$

For property pricing, the input is the characteristics of the property and the output a price prediction.

$$\text{Property Characteristics} \longrightarrow \text{Model} \longrightarrow \text{Price Prediction}$$

In English, models and algorithms have slightly different meanings. The Cambridge Dictionary defines a model as:

> "model, noun: a simple representation of a system or process, especially one that can be used in calculations or predictions of what might happen"
> (Cambridge Dictionary 2024c)

On the other hand, an algorithm is defined as:

> "algorithm, noun: a set of [...] instructions or rules that, especially if given to a computer, will help to calculate an answer to a problem"
> (Cambridge Dictionary 2024a)

Combining these two definitions, a Machine Learning model is a system adapting to data to generate predictions. It **learns** the relationship between an input and an output.

2.2.2 What makes a good model?

The job of a Machine Learning model is to make the most accurate predictions, to be as close to reality as possible.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

Predictions must be as close as possible to the **ground truth**, which is the true label or output. For example, the ground truth for a tumor is its actual diagnosis. The ground truth for a property is its final sale price. The ground truth for an email is whether it is actually spam or not.

Defining Truth

I love definitions, but defining truth is often disappointing. In this book, anything that is true is confirmed by the reality around us; i.e., empirical observation. This choice is made out of convenience. If you want to have some fun, I would recommend opening a few dictionaries and reading the definitions of the word "truth".

In the Cambridge Dictionary, "truth" is defined as:

> "the quality of being true"
> (Cambridge Dictionary 2024e)

Hoping to find an answer there, I looked up the definition of "true":

> "right and not wrong; correct"
> (Cambridge Dictionary 2024d)

As you can see, with limited success. This links back to the circularity of words and dictionaries. We only define words with more words.

Measuring this degree of closeness to ground truth and building accurate models are topics that will be discussed in the next sections.

2.3 Final Thoughts

Machine Learning models are everywhere. At their essence, these are just **adaptive prediction systems**. They learn relationships between input and output from historical data.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

How these models adapt and learn is the topic of this book.

Yet, these prediction models are only one part of the field of Machine Learning. The following chapter will explore two other types of Machine Learning: Unsupervised Learning and Reinforcement Learning.

2.4 Solutions

Solution 2.1. Exercise 2.1

Some ideas:

1. Classification: fraud detection, object detection, credit approval
2. Regression: energy demand prediction, sales prediction, stock price prediction

Chapter 3

Other Types of Machine Learning

Generating predictions by learning from input/output pairs only refers to a part of the discipline of Machine Learning, also known as **supervised learning**.

It is called "supervised" because the algorithm learns from **labelled examples**. As an example, a model learns to predict the diagnosis of a suspicious mass from its measurements.

Supervised learning will be the main focus of this book. This section will briefly explore other types of Machine Learning.

Sometimes, these labels do not exist. Let's imagine an archaeologist who stumbles upon a **pile of bones**. They want to understand if all the bones belong to the same species. One possible approach would be to measure all of these bones. In a simplified case, there are two measurements, length and width.

Plotting all these bone measurements on a chart, two clusters emerge:

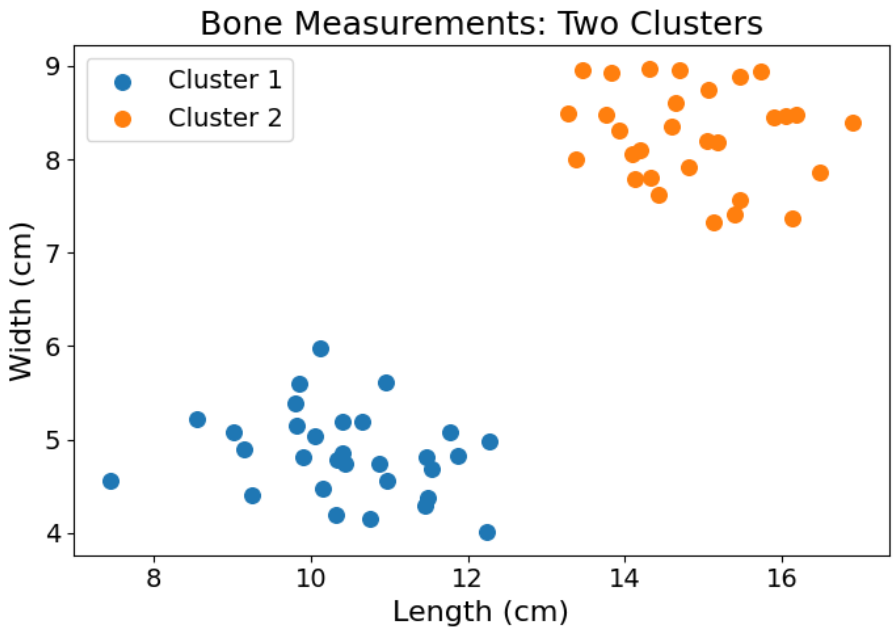


Figure 3.1: Plotting bones by measurements

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)
# Cluster 1
x1 = np.random.normal(10, 1, 30)
y1 = np.random.normal(5, 0.5, 30)
# Cluster 2
x2 = np.random.normal(15, 1, 30)
y2 = np.random.normal(8, 0.5, 30)

plt.figure(figsize=(7, 5))
plt.scatter(x1, y1, label='Cluster 1', s=70)
plt.scatter(x2, y2, label='Cluster 2', s=70)
plt.xlabel('Length (cm)', fontsize=16)
plt.ylabel('Width (cm)', fontsize=16)
plt.title('Bone Measurements: Two Clusters', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(fontsize=14)
```

```
plt.tight_layout()
plt.savefig("images/defining-prediction/clusters.png")
plt.show()
```

The exact species are still unknown, but the bones seem to belong to two different clusters. You have gained this information through the use of **unsupervised learning**, learning patterns or underlying structures of the data.

Exercise 3.1. Could a similar approach be used to discover the different customer groups of an online business? What data would you collect? And what would you do with it?

3.0.1 Reinforcement Learning

Another type of Machine Learning is **reinforcement learning** (RL), in which the Machine Learning algorithm does not generate predictions, but selects **actions** to maximise a certain **reward** within an **environment**. As an example, the objective of a RL algorithm could be to win at chess. At each move:

- the algorithm chooses an action: the next move
- based on its environment: the current state of the board
- to maximise a reward/objective: probability of winning the game

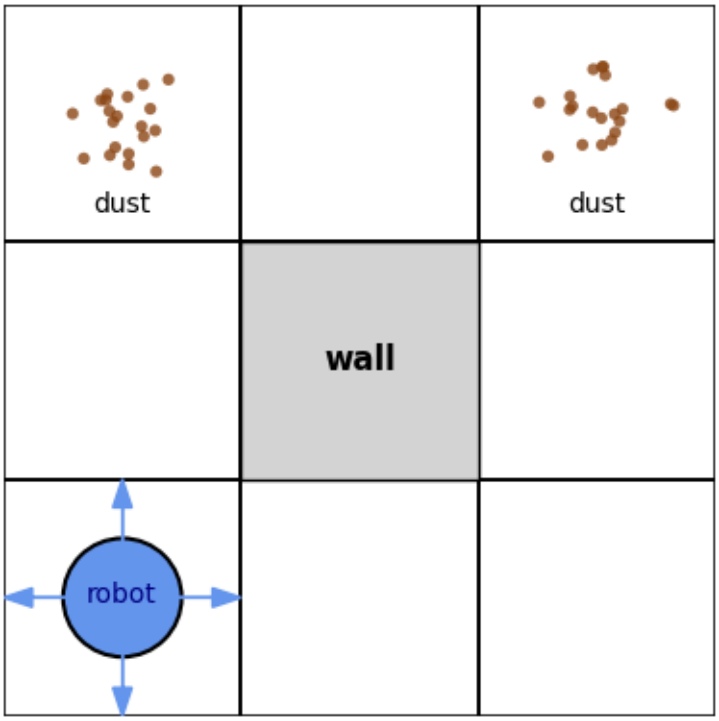


Figure 3.2: Autonomous vacuum cleaner, an example RL problem

Exercise 3.2. Autonomous vacuum cleaners can be powered by reinforcement learning systems. Give examples of their:

- actions
- environment
- reward/objective

Exercise 3.3. Think of another example application of reinforcement learning.

3.1 Final Thoughts

There are three main types of Machine Learning:

- Supervised learning: generate predictions based on input/output pairs (previous chapter)
- Unsupervised learning: discover patterns and underlying structure in data
- Reinforcement learning: build models that select actions to maximise a reward while observing an environment

Machine Learning is a very exciting field. But how is this related to Artificial Intelligence and Data Science? This will be covered by the next chapter.

3.2 Solutions

Solution 3.1. Exercise 3.1

Data to collect:

- Purchase history: frequency of purchase, quantity/amount purchased
- Browsing behaviour: activity in the last month
- Demographic data: gender, age

What to do with it:

- Plot the data and observe trends or clusters
- Use clustering algorithms to identify these clusters in higher dimensions (out of this book's scope)

Solution 3.2. Exercise 3.2

Autonomous vacuum cleaners can be powered by reinforcement learning systems. Give examples of their:

- actions: move forward/back/left/right, change power, start/stop, dock to recharge
- environment: room layout, location of dirt, battery level
- reward/objective: maximise dust collected, reduce cleaning time, avoid collisions

Solution 3.3. Exercise 3.3

Other RL example: self-driving cars

- actions: accelerate, brake, steer, signal, change lanes
- environment: road, traffic, pedestrians, weather conditions
- reward/objective: reach destination safely

Many other examples work here.

Chapter 4

Machine Learning, Artificial Intelligence and Data Science

The hype is real. Words like Artificial Intelligence (AI), Machine Learning (ML) and Data Science (DS) are frequently thrown around. This short chapter will explore the relations between these different concepts.

4.1 Artificial Intelligence

Artificial Intelligence is the design of computational (i.e., non-human) systems that can perform tasks typically requiring human intelligence (see Russell and Norvig 2021, 1–4). These tasks include learning, reasoning, problem-solving and planning, among others. Anything that we commonly associate with our own cognitive capabilities. AI is concerned both with the understanding of intelligence and the building of intelligent systems.

The reason the new Generative AI models like ChatGPT or Claude feel so "AI" is that they replicate aspects of human intelligence. They can generate helpful answers to user queries, in a human-like way.

4.2 Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence focused on designing algorithms that learn from data. These Machine Learning algorithms, also called "models", can generalise the rules learnt on training data to new, unseen data points. As opposed to rule-based systems, ML algorithms are **not** explicitly

programmed; they learn rules from the data. The next chapter will explore Machine Learning in more detail.

4.3 Deep Learning

Deep Learning is a subset of Machine Learning concerned with the development of multi-layer Neural Networks to perform tasks like classification and regression. These Neural Networks are (roughly) inspired by the workings of the human brain. Since the 1990s they have pushed the state of the art in Machine Learning research and are the backbone of the current AI revolution. As these models are more complex, they will not be covered in this book.

4.4 Data Science

Machine Learning is closely related to Data Science. For a few years (my first years on the job market), Data Scientist was the sexiest job of the 21st century (Davenport and Patil 2012).

Data Science is the extraction of generalisable knowledge from data. This is not knowledge at a given point in time, but **generalisable** knowledge. Knowledge that can be used on unseen data to predict the future.

Whereas Data Analysis is the study of what happened, Data Science focusses on predicting future trends. Looking at the case of property pricing, Data Analysts would create reports and analyse what happened. On the other hand, Data Scientists would build models to predict the price of new properties.

Beyond the hype, Data Science is a field concerned with extracting and extrapolating knowledge from data (Dhar 2013).

- Extract: to obtain something from something else, to get information from data.
- Extrapolate: to predict by projecting past experience or known data (Merriam-Webster Dictionary 2024a).

To summarise, Data Science uses data to extract useful knowledge from data to inform the future.

4.5 Final Thoughts

This chapter defined some important concepts:

- Artificial Intelligence (AI): design of intelligent systems, emulation of human cognition
- Machine Learning (ML): algorithms that learn from data
- Deep Learning: branch of ML focused on Neural Network modelling
- Data Science (DS): extraction of generalisable knowledge from data

Now that these definitions are out of the way, the next chapters will show how Machine Learning models work. The next chapter will explore the role of **data** in Machine Learning.

Chapter 5

Data and Space

Prediction models need information stored as input/output pairs. To predict the diagnosis of a tumour based on its characteristics, these models require many examples of tumour characteristics and tumour diagnosis pairs.

This collection is called a **dataset**, or a data sample. **Data** refers to any information being recorded and stored. From its Latin origin, it means "what is given".

From the point of view of the practitioner, data is rarely given, it is earned with sweat and tears. A sacrifice worth making, as a prediction model is only as good as its training data.

The simplest form of data is tabular data. The first known examples of data tables are 4000 years old (!) dating back to the Old Babylonian period. Then, these clay tablets were used for accounting and trade.

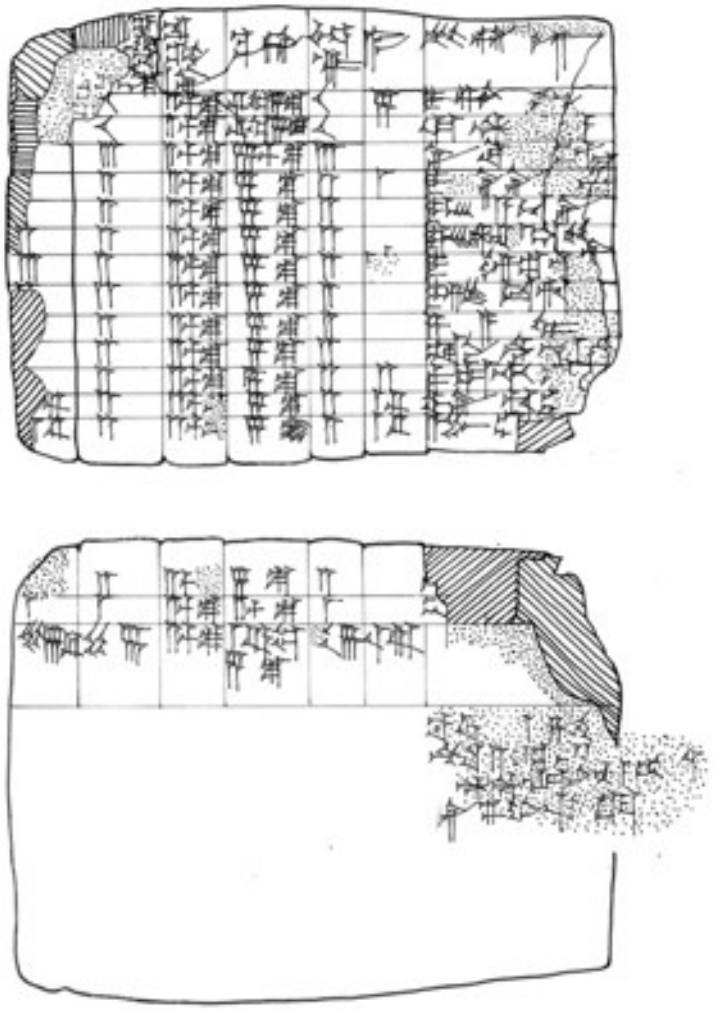


Figure 5.1: Ancient Babylonian clay tablet showing a table (CDLI contributors 2025)

5.1 The Anatomy of a Table

Tables are made of columns and rows.

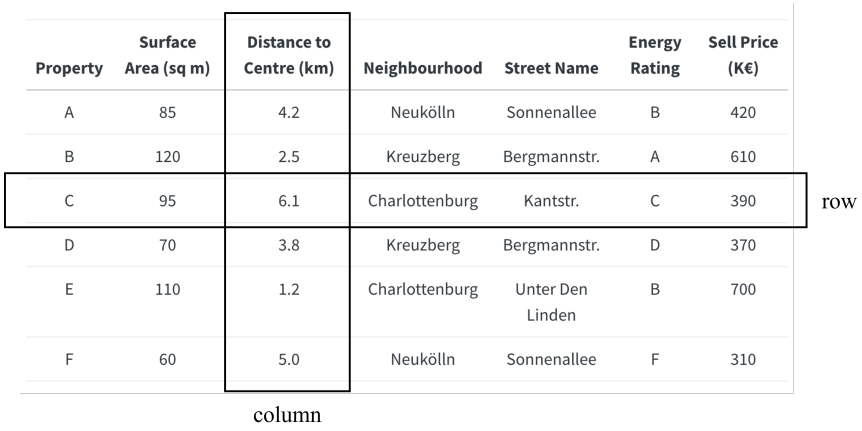


Figure 5.2: Basic elements of a table

Rows generally represent a real-world **entity**, with columns describing the **attributes** of this entity. Here, each row is a property, and each column describes an attribute of this property.

As shown in this table, the values stored in tables can have different data types:

- numeric: integer or continuous values
- categorical: text or category
- boolean: true or false
- datetime: representing a point in time

For prediction purposes, the label to predict is generally one of the many columns describing various entities. In the case of the example above, it is the "Sale Price (€)" column.

In most organisations today, tabular data is stored in **relational databases**, also called Relational Database Management Systems. It is a mouthful, frequently abbreviated as RDBMS, still a mouthful.

5.2 Data is Anything Stored

Data is any piece of information that is recorded. The text that I am typing now is data. The signal sent from my laptop keyboard strokes, to the CPU, and over a network to Google Docs… All of this is data.

To put some structure to this, the following are the main data modalities:

- Images: generally represented as grids of pixel (Picture Elements) values
- Audio: sound wave, amplitude of a signal over time
- Video: a combination of frames and one or more audio tracks
- Text: sequence of characters, each encoded as bits

- Binary files: any file stored with bits, sequence of 0s and 1s

The above data types are ultimately stored as binary data on a computer. Some, like text files, are **human-readable**, while others—such as images, audio, and video—are often stored in binary file formats that are **not** directly readable by humans.

This book will focus on building an intuition for Machine Learning using tabular data only.

5.3 From Data to Space

Getting back to tabular data, these first sections will focus on **numerical columns only**. Categorical and datetime features will be explored in the later sections. Spoiler alert: they will all be converted to numbers.

Looking at the table below, each tumour can be represented by a list of its characteristics, a list of numbers.

Diagnosis	Perimeter Mean (µm)	Area Mean (µm²)	Texture Mean
Benign	80	700	17.5
Malignant	110	1200	23.0

In Mathematics, a list of numbers can represent a **point in space**. In the example above, each tumour is associated with three numbers. We can use these numbers to represent each tumour as a point in a three-dimensional space.

Note: But what is space?

This is a fascinating question. The short answer is: a **set** with a degree of **structure**. A set can be thought of as a collection. But any set is not necessarily a space, as spaces require structure.

As an example, I can see objects on my desk: a glass of water and my notebook. These objects lie somewhere in a three dimensional space, with a certain position with regards to my computer. Using these positions, I could compute the **distance** between the different objects in space around me. But there, we are already getting carried away.

Below are two charts plotting suspicious mass observations in two and three dimensions:

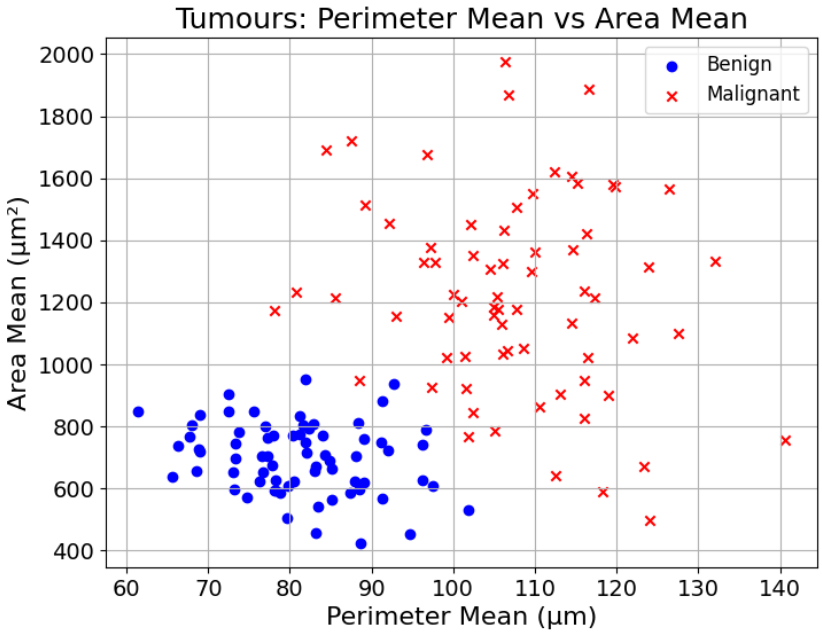


Figure 5.3: Plotting observations in two dimension

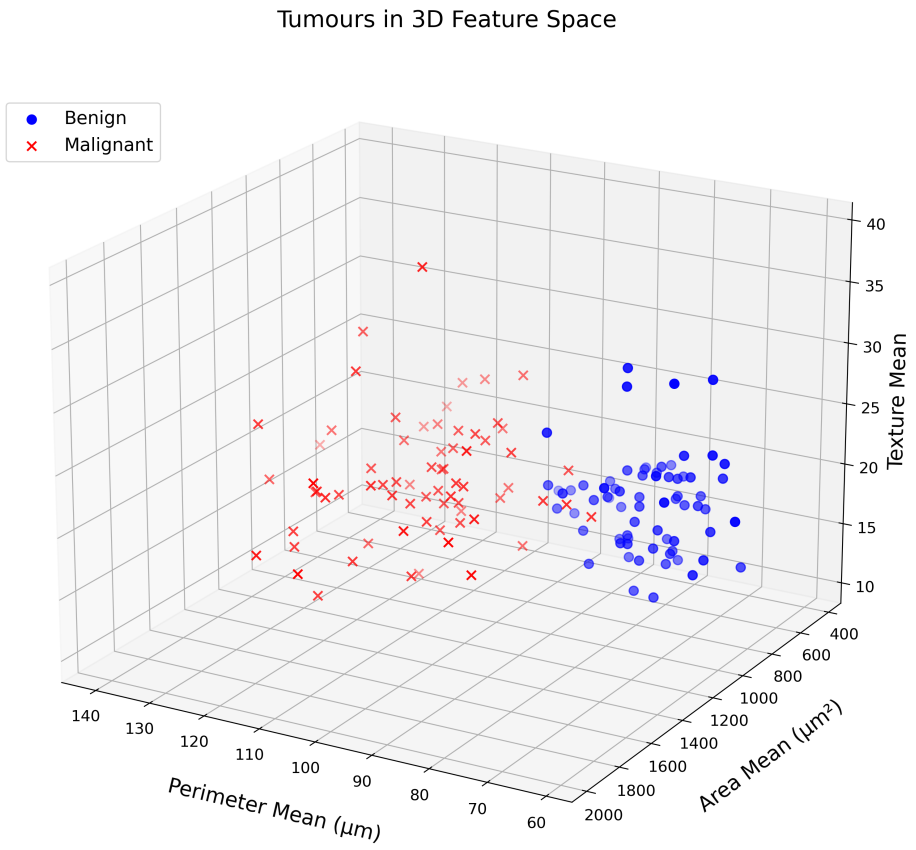


Figure 5.4: Plotting observations in three dimensions

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.datasets import load_breast_cancer

# Use breast cancer dataset for realistic texture means
data = load_breast_cancer()
benign_texture = data.data[data.target == 1][:, 1]  # texture_mean for benign
malignant_texture = data.data[data.target == 0][:, 1]  # texture_mean for malignant

benign_center = [80, 700]
malignant_center = [110, 1200]
```

```
n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=1)
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=2)

benign_std = [10, 120]
malignant_std = [12, 300]

X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

# Sample realistic texture means
np.random.seed(42)
benign_texture_sample = np.random.choice(benign_texture, n_samples)
malignant_texture_sample = np.random.choice(malignant_texture, n_samples)

plt.figure(figsize=(8,6))
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malignant')
plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.title('Tumours: Perimeter Mean vs Area Mean', fontsize=18)
plt.legend(fontsize=12)
plt.grid(True)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()

# 3D plot
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_benign[:,0], X_benign[:,1], benign_texture_sample, marker='o', color='blue', label='
ax.scatter(X_malignant[:,0], X_malignant[:,1], malignant_texture_sample, marker='x', color='red',
ax.set_xlabel('Perimeter Mean (µm)', fontsize=16)
ax.set_ylabel('Area Mean (µm²)', fontsize=16)
ax.set_zlabel('Texture Mean', fontsize=16)
ax.set_title('Tumours in 3D Feature Space', fontsize=18)
ax.legend(fontsize=12)
ax.tick_params(axis='x', labelsize=14)
ax.tick_params(axis='y', labelsize=14)
ax.tick_params(axis='z', labelsize=14)
plt.show()
```

This text is printed on a flat surface. And yet, using perspective and transparency, one can create the illusion of a third dimension. If you think that this 3D chart is barely legible, I agree with you.

What is true in a two- and three-dimensional space also applies to **any number of dimensions**. As the number of dimensions increases, plotting data on paper gets more difficult.

5.4 Final Thoughts

But why do we bother with space and with these lists of numbers? Representing entities and observations in this way allows us to use **mathematical tricks to make predictions**. The nature of these tools will be studied in the next sections, starting with **distance**, and answering the question: how **similar** are two observations?

Chapter 6

Distance and Similarity

Now that points exist in space, we could use the distance between them to make **inferences**. Inference refers to using information we have, to make **guesses** about the information we do not have.

As an example, we could infer that similar tumours, or tumours with similar characteristics, have the same diagnosis. But how could we measure the **similarity** between two observations?

This is where **distance** comes in.

Looking at the plot below, how would you classify the tumour labelled by a question mark?

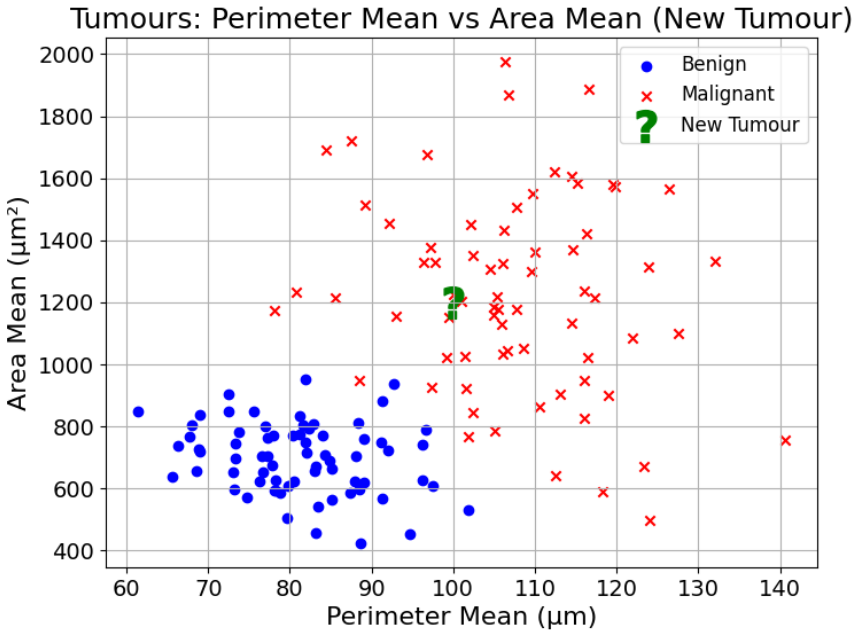


Figure 6.1: Labelling a new observation using distance

Humans looking at this chart would **intuitively** compare the diagnosis of nearby observations with the unknown observation, and base their guess on **neighbours**.

This type of inference is something we do every day. As an example, I have seen many rabbits. If I see a small animal with long ears pointing up, I will classify this as a rabbit, as it is similar to my previous observations of rabbits.

6.1 Starting with Subtraction

To understand the concept of distance, we can start with **one-dimensional space**. This notion may seem strange, as we generally associate space with either two or three dimensions.

Let's consider a set of three flats in Berlin, labelled A, B and C, with the following surface areas:

Flat	Surface Area (m^2)
A	22
B	35
C	55

In this example, the surface area represents the single dimension of this space. We can plot the different flats on this dimension:

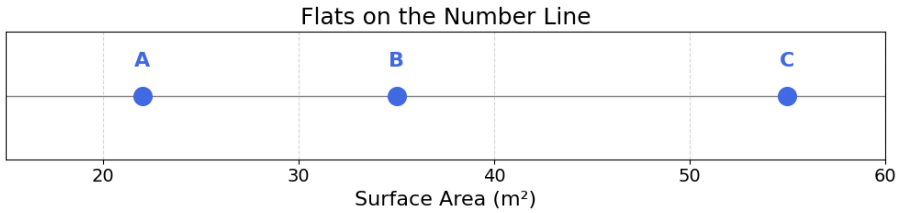


Figure 6.2: Distance in one dimension

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

areas = np.array([22, 35, 55])
labels = np.array(['A', 'B', 'C'])

sort_idx = np.argsort(areas)
areas_sorted = areas[sort_idx]
labels_sorted = labels[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorder=2)

plt.yticks([])
plt.xlabel('Surface Area (m²)', fontsize=16)
plt.title('Flats on the Number Line', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

xtick_vals = np.arange(20, 61, 10)
plt.xticks(xtick_vals, xtick_vals, fontsize=14)

for x, label in zip(areas_sorted, labels_sorted):
    plt.annotate(label, (x, 0), xytext=(0, 20), textcoords='offset points',
                 ha='center', va='bottom', fontsize=16, fontweight='bold', color='royalblue')

plt.xlim(15, 60)
plt.tight_layout()
plt.show()
```

In this space, what is the difference between A and C? How would you compute

it?

A simple subtraction would be a good start:

$$\text{surface}_C - \text{surface}_A = 55 - 22 = 33$$

Now, calculate the distance between B and C. You should get:

$$\text{surface}_C - \text{surface}_B = 55 - 35 = 20$$

In line with our intuition, we see that the distance between B and C is **shorter** than the distance between A and C. The difference in their surface area is **smaller**.

Moving forward, we will note the distance between A and C (or any other two points) as $d(A, C)$.

Can you see an **issue** with using subtraction as a distance calculation?

One of the main issues is that $d(A, C) \neq d(C, A)$. From the above, we know $d(A, C) = 33$. Calculating $d(C, A)$, we get:

$$\text{surface}_A - \text{surface}_C = 22 - 55 = -33$$

This is unfortunate, as the distance between two points should not have a direction. It should be the same regardless of the starting point.

How could we solve this?

6.2 Alternatives to Subtraction

There are two mathematical tricks we could use to tackle this challenge:

- Absolute Value
- Squared Difference

6.2.1 Absolute Value

You can think of the **absolute value** of a number as removing any negative sign. More rigorously, the absolute value is the **magnitude** of a number, not its sign. The absolute value of number x is noted $|x|$.

For example: $|2| = |-2| = 2$

For mathematically-inclined readers (others can close their eyes for two lines), the absolute value **function** is defined as:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

Plotting the absolute value of all numbers between -5 and 5, we get the following triangular shape:

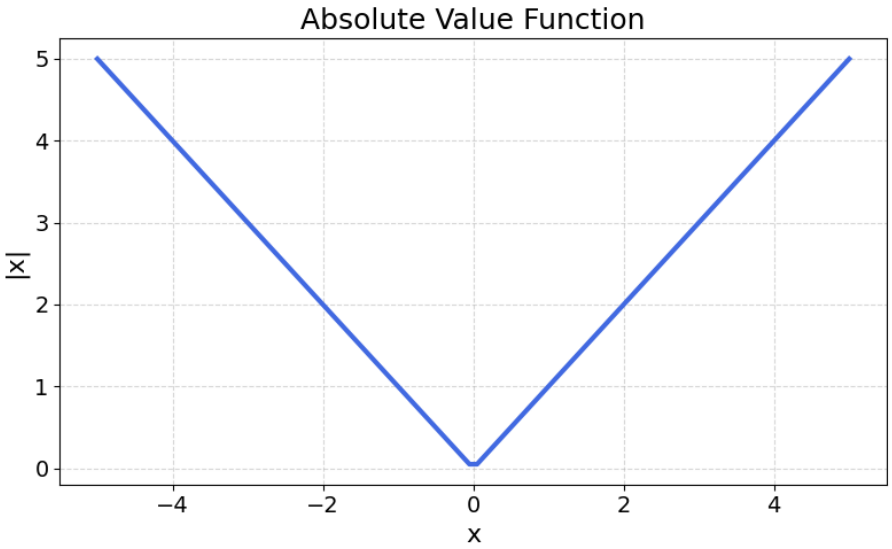


Figure 6.3: Absolute value function

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5, 5, 100)
y = np.abs(x)
plt.figure(figsize=(8, 5))
plt.plot(x, y, color='royalblue', linewidth=3)
plt.title('Absolute Value Function', fontsize=18)
plt.xlabel('x', fontsize=16)
plt.ylabel('|x|', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

One of the main advantages of the absolute value function as a distance metric is the following:

$$|x - y| = |y - x|$$

The absolute value of the difference between two numbers is the same regardless of their order. Going back to our example:

$$|\text{surface}_C - \text{surface}_A| = |55 - 22| = |33| = 33$$
$$|\text{surface}_A - \text{surface}_C| = |22 - 55| = |-33| = 33$$

Exercise 6.1. Compute the absolute value distance between B and C, and between C and B. Show that both are equal to 20.

6.2.2 Squared Difference

Another way to make sure the distance between two points is the same is to **square the difference**. A number squared, noted x^2, is a number multiplied by itself: $x \cdot x$

This operation has the same property as the absolute value:

$$(x - y)^2 = (y - x)^2$$

Revisiting the example above:

$$(\text{surface}_C - \text{surface}_A)^2 = (55 - 22)^2 = 33^2 = 1089$$
$$(\text{surface}_A - \text{surface}_C)^2 = (22 - 55)^2 = (-33)^2 = 1089$$

Plotting the square of all numbers in the range -5 to 5, we notice that it has a parabolic shape:

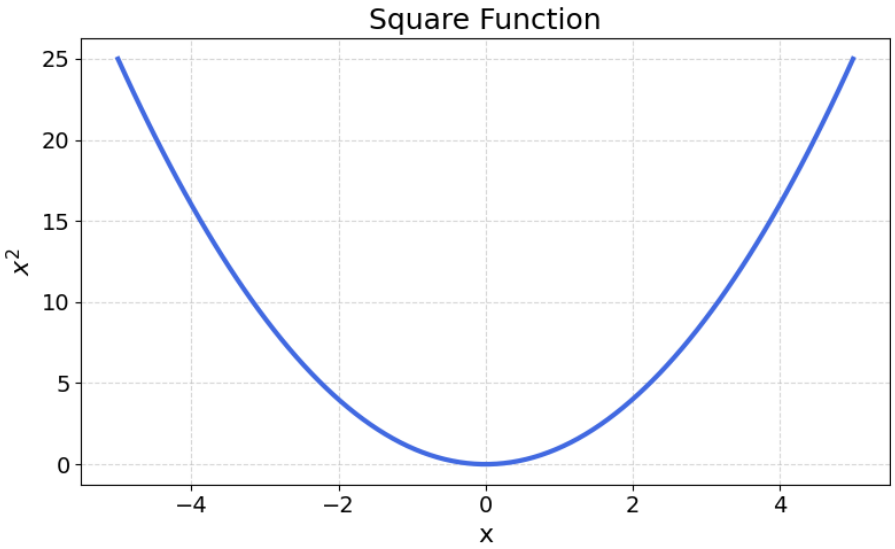


Figure 6.4: Square function

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5, 5, 100)
y = x**2
plt.figure(figsize=(8, 5))
plt.plot(x, y, color='royalblue', linewidth=3)
plt.title('Square Function', fontsize=18)
plt.xlabel('x', fontsize=16)
plt.ylabel('$x^2$', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

This function **increases faster** as the input number grows.

One potential issue with squared difference is that this number can grow very fast, and is sometimes hard to interpret. Considering the example above, it seems strange that the distance between 22 and 55 would be 1089.

To make squared differences more interpretable, it is common to use the square root of the squared difference. The square root (noted $\sqrt{\ }$) is the number which,

when multiplied by itself, gives the original number. For example, $\sqrt{9} = 3$ because $3 \cdot 3 = 9$

Going back to the example, the square root difference between A and C would be:

$$\sqrt{(\text{surface}_C - \text{surface}_A)^2} = \sqrt{33^2} = \sqrt{1089} = 33$$

Exercise 6.2. Compute the squared difference between B and C, and between C and B. Show that both are equal to 400.

6.3 Two Dimensions

The methods above can accurately measure the distance between points in one dimension. But how to measure the distance between **points in two dimensions**?

What would be the distance between the points A and B shown on the picture below?

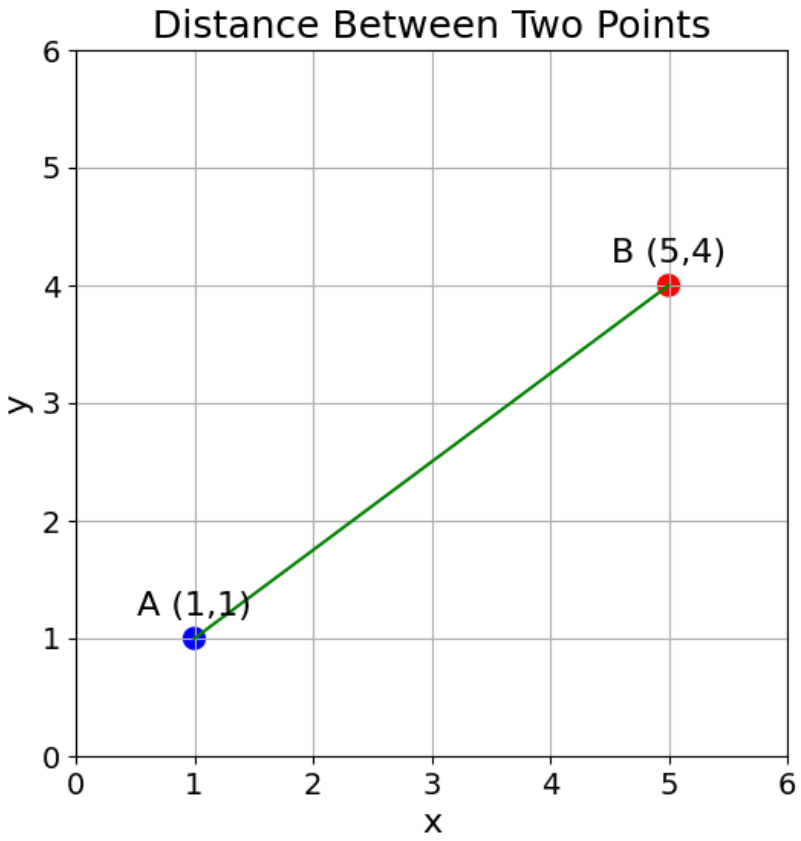


Figure 6.5: Two points in space

Hint: The Pythagorean theorem may be useful.

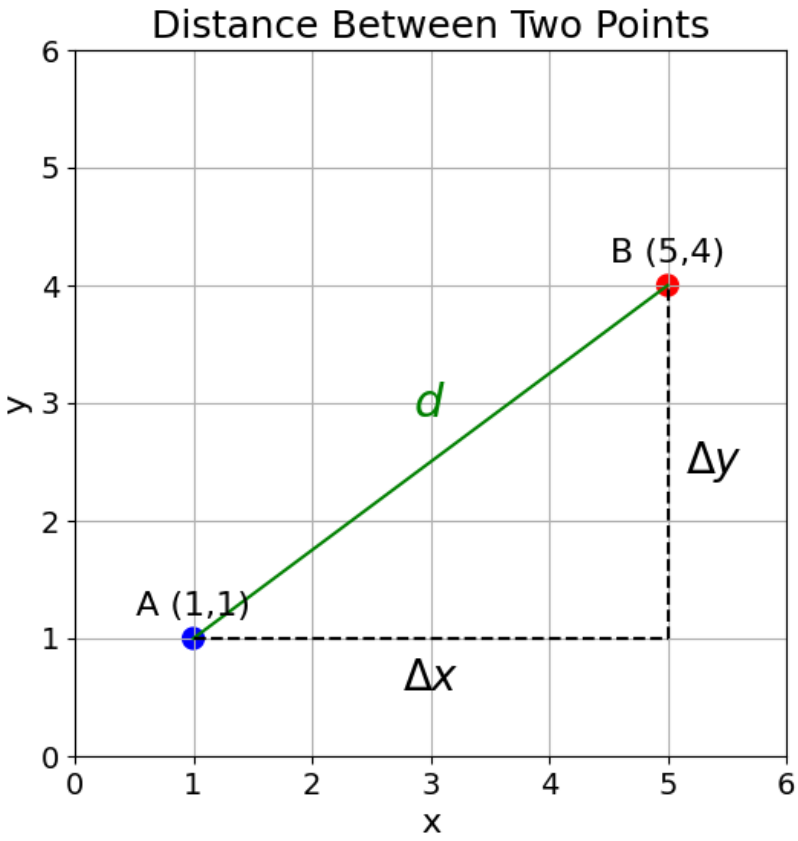


Figure 6.6: Thinking of distance as the hypotenuse of a right-angle triangle

To find the distance between point A $(1, 1)$ and point B $(5, 4)$ using the Pythagorean theorem, we can consider these points as two vertices of a right-angled triangle.

The horizontal distance (Δx) between the points is:

$$\Delta x = x_2 - x_1 = 5 - 1 = 4$$

The vertical distance (Δy) between the points is:

$$\Delta y = y_2 - y_1 = 4 - 1 = 3$$

According to the Pythagorean theorem, the square of the hypotenuse (which is the distance d between points A and B) is equal to the **sum of the squares** of the other two sides (Δx and Δy):

$$d^2 = (\Delta x)^2 + (\Delta y)^2$$

Substituting the values:

$$d^2 = (4)^2 + (3)^2$$
$$d^2 = 16 + 9$$
$$d^2 = 25$$

To find d, we take the square root of both sides:

$$d = \sqrt{25}$$
$$d = 5$$

So, the distance between point A $(1, 1)$ and point B $(5, 4)$ is 5.

You may notice a striking similarity between the Pythagorean theorem and the **square root of the squared distance** defined in the previous section.

The distance function derived from the Pythagorean theorem is called the **Euclidean distance**. Let us compare the Euclidean distance in one and two dimensions:

- In one dimension:
$$d(A, B) = \sqrt{(x_2 - x_1)^2}$$
- In two dimensions:
$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Exercise 6.3. Calculate the Euclidean Distance between point A and B defined in this table:

Point	x_1	x_2
A	2	4
B	5	1

Hint: It may be helpful to plot these two points.

6.4 To Infinity and Beyond

As shown in the last chapter, data represents **points in space in many dimensions**. It is not uncommon to have datasets with hundreds of columns. How to measure distance in such a high-dimensional space?

The Euclidean Distance could be used for two, three or any n number of dimensions. It can be noted in the following way:

$$d(A, B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2 + \cdots + (b_n - a_n)^2}$$

Exercise 6.4. Compute the Euclidean Distance between the points A and B in 5 dimensions (you can use a calculator):

Point	Dimension 1	Dimension 2	Dimension 3	Dimension 4	Dimension 5
A	2	4	1	3	7
B	5	1	6	2	9

6.4.1 Scary Sigma

6.4.1.1 Some Context

A more concise notation of the Euclidean distance uses the Σ (pronounced "sigma") summation operator. This is a scary symbol, though its meaning is relatively simple.

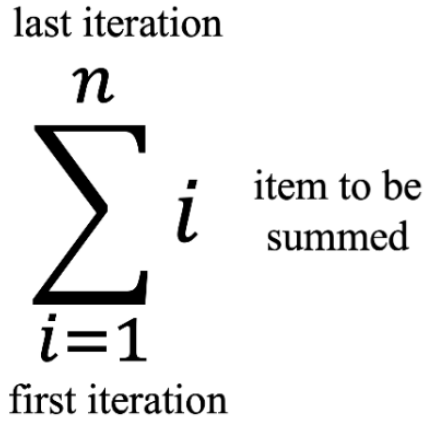


Figure 6.7: Sigma explained

As an example, $\sum_{i=1}^{n} i$ represents the sum of all integers from 1 to n:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n$$

In this expression:

- $i = 1$ (at the bottom): Starting value of our counter
- n (at the top): Ending value of our counter
- i (after the sigma): The expression to sum for each value of the counter

To make this more concrete, the sum of all integers from 1 to 4 can be written:

$$\sum_{i=1}^{4} i = 1 + 2 + 3 + 4$$

The following expression is the sum of all integers from 1 to 3, divided by 2:

$$\sum_{i=1}^{3} \frac{i}{2} = \frac{1}{2} + \frac{2}{2} + \frac{3}{2}$$

Exercise 6.5. Calculate the following summations:

1. $\sum_{i=1}^{4}(i+2)$
2. $\sum_{i=1}^{3} \frac{i}{3}$
3. $\sum_{i=1}^{5} \frac{1}{i}$

The Σ operator is very useful when dealing with **collections of numbers** and dimensions; something that is very common in Machine Learning.

6.4.1.2 Sigma and the Euclidean Distance

Using the Σ notation, how to represent the Euclidean Distance in a more concise format?

In dot notation for n dimensions:

$$d(A, B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \cdots + (b_n - a_n)^2}$$

In sigma notation, with n the number of dimensions:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

6.5 Final Thoughts

That is it! Using the above formula, you can compute the **distance between any two points in a space of n dimensions**. This will be very useful when building the first prediction model of this book, K-Nearest Neighbours.

6.6 Solutions

Solution 6.1. Exercise 6.1

$$|\text{surface}_C - \text{surface}_B| = |55 - 35| = |20| = 20$$
$$|\text{surface}_B - \text{surface}_C| = |35 - 55| = |-20| = 20$$

Solution 6.2. Exercise 6.2

$$(\text{surface}_C - \text{surface}_B)^2 = (55 - 35)^2 = 20^2 = 400$$
$$(\text{surface}_B - \text{surface}_C)^2 = (35 - 55)^2 = (-20)^2 = 400$$

Solution 6.3. Exercise 6.4 First, compute the squared difference for each of the five dimensions:

$$(5-2)^2 = 9$$
$$(1-4)^2 = 9$$
$$(6-1)^2 = 25$$
$$(2-3)^2 = 1$$
$$(9-7)^2 = 4$$

Sum:

$$9 + 9 + 25 + 1 + 4 = 48$$

Take the square root:

$$d(A, B) = \sqrt{48} \approx 6.93$$

Solution 6.4. Exercise 6.5

1. $\sum_{i=1}^{4}(i+2) = (1+2) + (2+2) + (3+2) + (4+2) = 3 + 4 + 5 + 6 = 18$
2. $\sum_{i=1}^{3} \frac{i}{3} = \frac{1}{3} + \frac{2}{3} + \frac{3}{3} = 2$
3. $\sum_{i=1}^{5} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \approx 1 + 0.5 + 0.333 + 0.25 + 0.2 = 2.283$

Chapter 7

Neighbours

7.1 Intuition

It is all starting to come together. It is now time to build the first prediction model in this book.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Predictions}$$

A Machine Learning model learns the **relationship** between input/output pairs to generate predictions on new inputs. For the first time in this book, this chapter will explore **how these models learn**.

Using the example of tumour diagnosis:

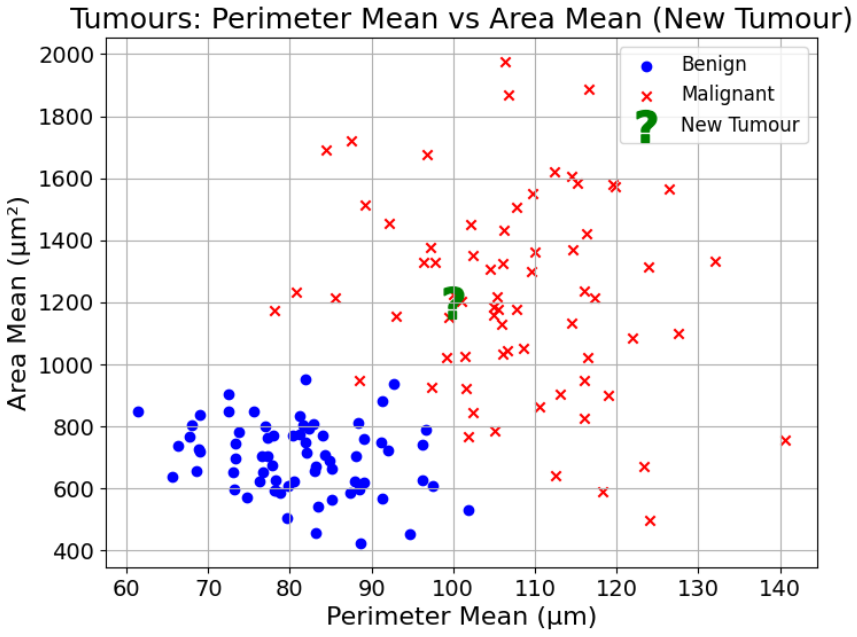


Figure 7.1: Tumour dataset: Perimeter Mean vs Area Mean

We want to predict the diagnosis of the observation labelled with a question mark (?). How could this be done using distance?

A good start would be to get the new observation's **closest neighbours**. This can be done by calculating the **distance** between the new observation and the existing observations, and picking the ones with shortest distances.

Using your intuition, how would you classify this new observation?

What did our intuition rely on to make this judgement? When I came up with a prediction, I looked at the observation's **neighbours** and applied a **majority vote**.

7.2 Algorithm

How can we build this intuition into a **program**, an **algorithm**?

The goal is to craft a list of rules that could be executed by a computer. Such a list could look like this:

- Find the 5 closest neighbours of the observation
- Count the number of neighbours per class

- The new observation is labelled with the class that has the **highest number of neighbours**

In Machine Learning, this model is referred to as K-Nearest Neighbours or KNN. K simply means any positive integer, referring to the number of neighbours considered (here 5).

This model acts as a **map from feature values to a prediction**. Using the algorithm described above, **any tumour** observation can be assigned to a diagnosis. This map can be visualised by applying this majority vote to every point and colouring it by its diagnosis:

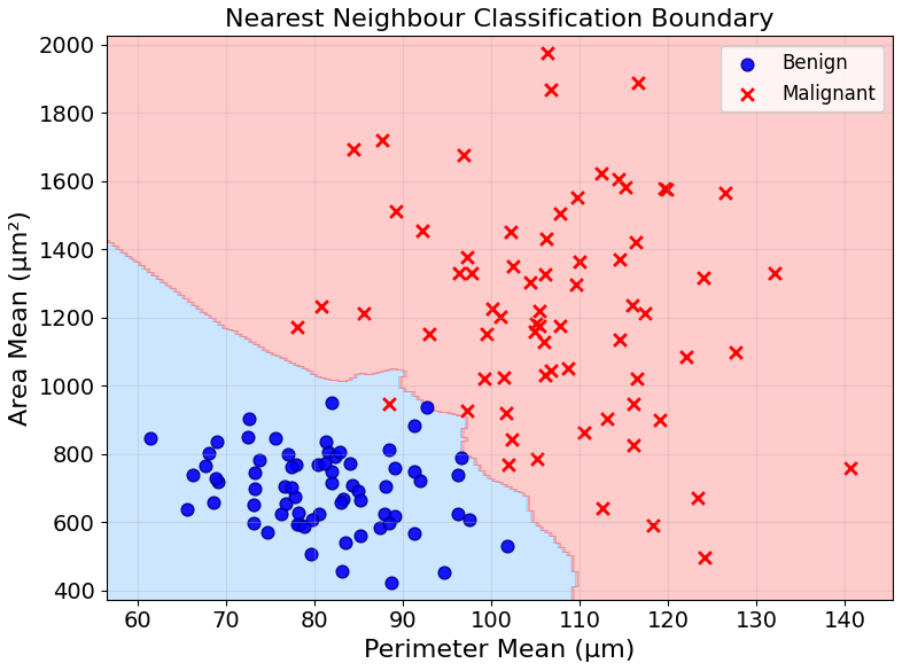


Figure 7.2: KNN prediction map

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_blobs
```

```
benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70
X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando
benign_std = [10, 120]
malignant_std = [12, 300]
X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

# Combine into features and target
X = np.vstack([X_benign, X_malignant])
y = np.hstack([np.zeros(len(X_benign)), np.ones(len(X_malignant))])

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Create custom colors with brighter hues
custom_cmap = ListedColormap(['#80C2FF', '#FF8080'])

# Fit a KNN classifier with scaled data
clf = KNeighborsClassifier(
    n_neighbors=5,
    weights='uniform',
    algorithm='auto',
    metric='euclidean'
)
clf.fit(X_scaled, y)

# Create the decision boundary plot
plt.figure(figsize=(8, 6))

# Create a meshgrid for displaying the decision boundary
h = 0.5  # step size in the mesh
x_min, x_max = X[:, 0].min() - 5, X[:, 0].max() + 5
y_min, y_max = X[:, 1].min() - 50, X[:, 1].max() + 50
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Scale the mesh points the same way the training data was scaled
mesh_points = np.c_[xx.ravel(), yy.ravel()]
mesh_points_scaled = scaler.transform(mesh_points)

# Predict using the scaled mesh points
```

```
Z = clf.predict(mesh_points_scaled)
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, alpha=0.4, cmap=custom_cmap)

# Plot the data points (using original unscaled coordinates for display)
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign',
            s=60, edgecolor='darkblue', alpha=0.9, linewidth=1)
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malignant',
            s=60, linewidth=2)

plt.title('Nearest Neighbour Classification Boundary', fontsize=16)
plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(fontsize=12)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("images/neighbours/classification_boundary.png")
plt.show()
```

7.3 Adding some nuance

This is all very exciting. But let's consider the following example from a **patient's perspective**. Let's imagine we have to predict the diagnosis of the following tumours:

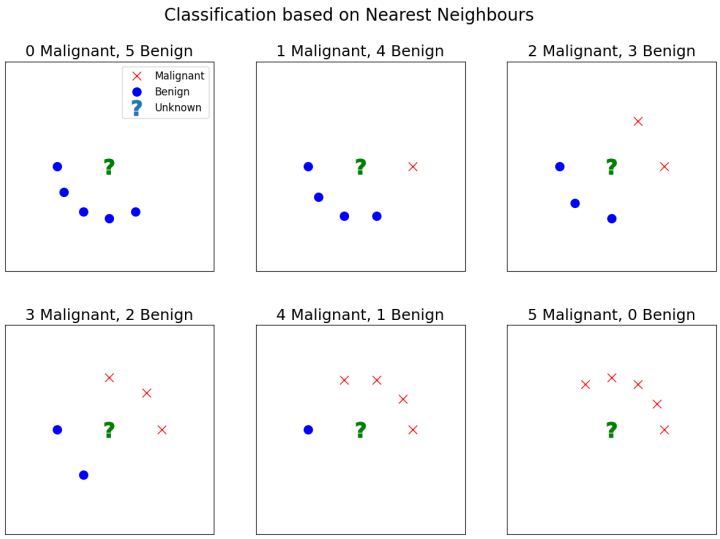


Figure 7.3: Neighbours with different class ratios

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

def plot_neighbours(num_malignant, ax, total_neighbours=5):
    num_malignant = min(num_malignant, total_neighbours)
    num_benign = total_neighbours - num_malignant
    angles_malignant = np.linspace(0, np.pi, num_malignant + 1, endpoint=False)[:-1] i
    angles_benign = np.linspace(np.pi, 2*np.pi, num_benign + 1, endpoint=False)[:-1] i
    radius = 0.5

    # Plot malignant neighbours
    for i in range(num_malignant):
        ax.plot(radius * np.cos(angles_malignant[i]), radius * np.sin(angles_malignant

    # Plot benign neighbours
    for i in range(num_benign):
        ax.plot(radius * np.cos(angles_benign[i]), radius * np.sin(angles_benign[i]),

    # Central unknown point
    ax.plot(0, 0, 'g', markersize=18, marker=r'$\mathbf{?}$')
```

```
    ax.set_title(f'{num_malignant} Malignant, {num_benign} Benign', fontsize=18)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_xlim([-1, 1])
    ax.set_ylim([-1, 1])
    ax.set_aspect('equal', adjustable='box')

# Create a 2x3 subplot layout
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
fig.suptitle('Classification based on Nearest Neighbours', fontsize=20, y=0.98)

# Flatten the axes array for easier iteration
axes = axes.flatten()

for i in range(6):
    plot_neighbours(i, axes[i])

# Add a legend to the first subplot
axes[0].plot([], [], 'rx', markersize=10, label='Malignant')
axes[0].plot([], [], 'bo', markersize=10, label='Benign')
axes[0].plot([], [], ' ', markersize=18, marker=r'$\mathbf{?}$', label='Unknown')
axes[0].legend(loc='upper right', fontsize=12)


plt.subplots_adjust(top=0.9)
plt.show()
```

There, a simple majority vote would mean that the observation with 2 malignant neighbours could be classified as "benign". As a patient, I would prefer to have a second opinion there.

As 2 out of the 5 neighbours of this observation are malignant, we are **less certain** that this tumour is benign. Could the model prediction reflect this uncertainty?

One way to do this would be, instead of taking the winner of a majority vote, we could estimate the **probability** of a tumour to be malignant.

Note: Defining Probability

The probability assigns a degree of belief to an event, between 0 and 1. For example, the probability of rolling a 4 on a fair six-sided die is $1/6 \approx 0.167$. The probability of getting heads on a fair coin is $1/2 = 0.5$.

Before computing the predicted probability of malignancy, we could start with intuition. If the 5 neighbours of the observation are malignant, the model should be 100% sure that the tumour is malignant. Conversely, if none of its 5 neighbours is malignant, the model would assign a probability of 0% to malignancy.

So far so good. But what about 3 and 2, or 1 and 4? By calculating the average, we could come up with a predicted probability.

If 3 neighbours are malignant and 2 are benign, we could compute the probability of malignancy with the expression:

$$\text{Probability} = \frac{3}{3+2} = \frac{3}{5} = 0.6 = 60\%$$

Exercise 7.1. Calculate the probability of malignancy if four out of five neighbours are malignant. Show that it is 80%

By using this approach, the model can output more nuanced predictions that reflect the *uncertainty* in the training data.

This approach can be used to revise the map visualised in the previous section. Instead of being coloured by the predicted label, it is coloured by the **predicted probability** of malignancy (1 malignant, 0 benign).

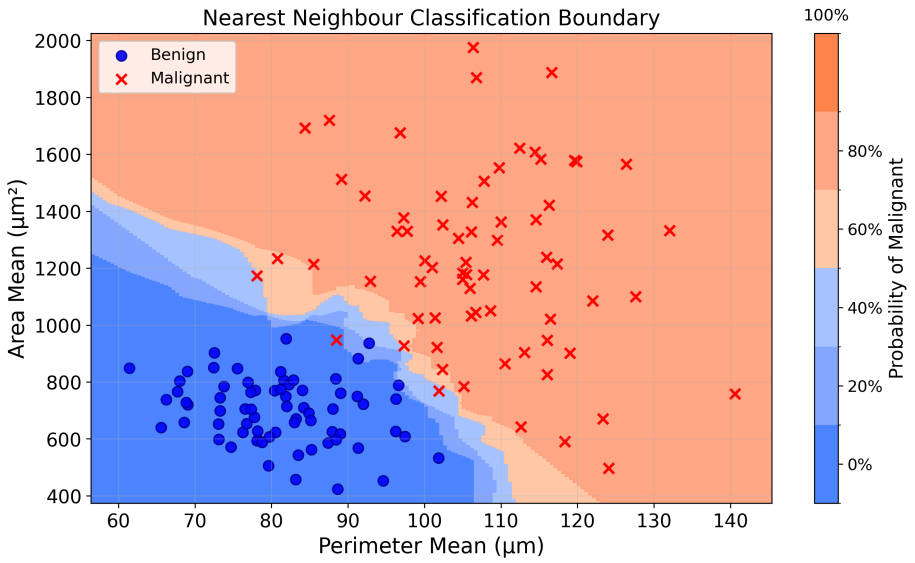


Figure 7.4: KNN probability map

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap, BoundaryNorm
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.datasets import make_blobs

benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70
X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=1)
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=2)
benign_std = [10, 120]
malignant_std = [12, 300]
X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

# Combine into features and target
X = np.vstack([X_benign, X_malignant])
y = np.hstack([np.zeros(len(X_benign)), np.ones(len(X_malignant))])

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit a KNN classifier with scaled data
clf = KNeighborsClassifier(
    n_neighbors=5,
    weights='uniform',
    algorithm='auto',
    metric='euclidean'
)
clf.fit(X_scaled, y)

# Create the decision boundary plot
plt.figure(figsize=(10, 6))

# Create a meshgrid for displaying the decision boundary
h = 0.5  # step size in the mesh
x_min, x_max = X[:, 0].min() - 5, X[:, 0].max() + 5
y_min, y_max = X[:, 1].min() - 50, X[:, 1].max() + 50
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Scale the mesh points the same way the training data was scaled
mesh_points = np.c_[xx.ravel(), yy.ravel()]
mesh_points_scaled = scaler.transform(mesh_points)

# Predict probabilities using the scaled mesh points
Z_proba = clf.predict_proba(mesh_points_scaled)[:, 1]  # Probability of class 1 (malignant)
```

```
Z_proba = Z_proba.reshape(xx.shape)

# Create a discrete colormap with exactly 6 levels
# Define the boundaries for the 6 levels
bounds = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.01]  # Adding 1.01 to ensure 1.0 is included
# Define the colors for each level - blue to red gradient
colors = ['#0050FF', '#5080FF', '#80AAFF', '#FFB080', '#FF8050', '#FF5000']
# Create a custom discrete colormap with these colors and boundaries
cmap = ListedColormap(colors)
norm = BoundaryNorm(bounds, cmap.N)

# Plot the decision boundary with discrete probability levels
contour = plt.contourf(xx, yy, Z_proba, levels=bounds, cmap=cmap, norm=norm, alpha=0.7)

# Plot the data points (using original unscaled coordinates for display)
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign',
           s=60, edgecolor='darkblue', alpha=0.9, linewidth=1)
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malign
           s=60, linewidth=2)

# Add a colorbar with discrete ticks
cbar = plt.colorbar(contour, ticks=[0.1, 0.3, 0.5, 0.7, 0.9])  # Position ticks in mid
cbar.set_label('Probability of Malignant', fontsize=14)

# Set custom labels on the colorbar
cbar.ax.set_yticklabels(['0%', '20%', '40%', '60%', '80%'])
cbar.ax.tick_params(labelsize=12)

# Add a 100% label at the top
cbar.ax.text(0.5, 1.02, '100%', transform=cbar.ax.transAxes,
             ha='center', va='bottom', fontsize=12)

plt.title('Nearest Neighbour Classification Boundary', fontsize=16)
plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(fontsize=12, loc='upper left')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

7.4 From classification to regression

The above section only showed the application of KNN to a **classification** problem, predicting the diagnosis of a suspicious mass.

Could the same model be used for a regression problem (predicting a continuous quantity)?

To do so, let's turn to the problem of property pricing. When selling a property, customers need to know how much their property is worth. This can be used as a basis to compute the listing price, price for which the property is first listed. It is also in the buyer's interest to have a very good estimation of the value of a property before agreeing to the transaction.

As explained in the Defining Prediction chapter, this pricing can be done with:

1. Intuition: From experience, real estate have an understanding of the property market in their area of specialisation. They could "know" how much a property would be worth
2. Rule-based Systems: Property analysts could build models that price properties based on their characteristics. A simple rule working surprisingly well is: average price per square meter * surface area of the property

Can we use the Nearest Neighbour model to learn from historical data and generate new price predictions for unseen data? Another leading question. Yes.

For simplicity, let's use the following features: number of rooms and distance from centre (km).

The training data looks like this:

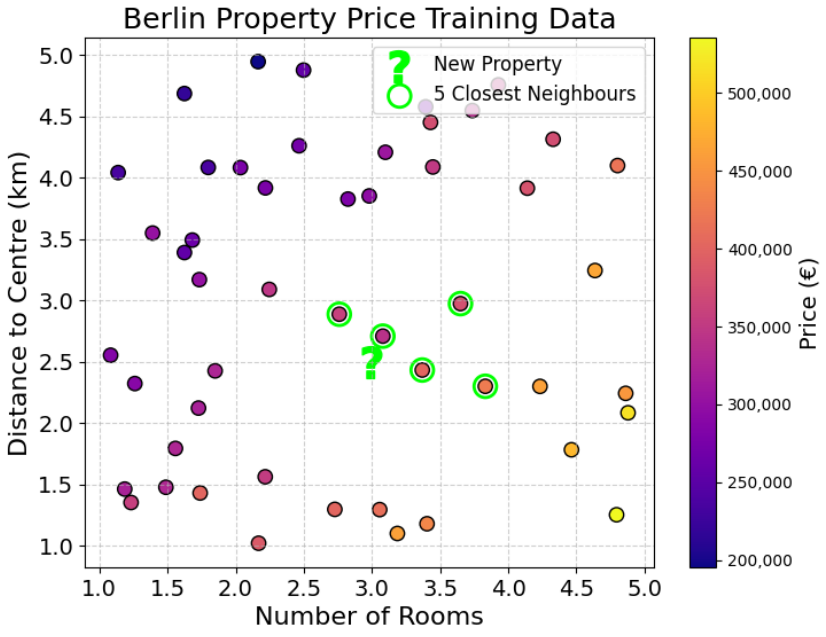


Figure 7.5: Property pricing training data

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor, NearestNeighbors

np.random.seed(42)

num_samples = 50
number_of_rooms = np.random.uniform(1, 5, num_samples)
distance_to_centre = np.random.uniform(1, 5, num_samples)

# Price formula: base price + price per room - price per km from centre + noise
price = 300000 + 60000 * number_of_rooms - 40000 * distance_to_centre + np.random.norma

X = np.vstack([number_of_rooms, distance_to_centre]).T

new_observation = np.array([[3, 2.5]])

nbrs = NearestNeighbors(n_neighbors=5, algorithm='ball_tree').fit(X)
distances, indices = nbrs.kneighbors(new_observation)
```

```
# Print the neighbours' data for the table
print("The 5 closest neighbours are:")
for i in range(len(indices.flatten())):
    idx = indices.flatten()[i]
    # Rounding for display purposes
    print(
        f"Rooms: {round(X[idx, 0])}, "
        f"Distance: {X[idx, 1]:.1f} km, "
        f"Price: €{price[idx]:,.0f}"
    )

plt.figure(figsize=(8, 6))
sc = plt.scatter(number_of_rooms, distance_to_centre, c=price, cmap='plasma', s=80, edgecolor='k'
plt.scatter(new_observation[0, 0], new_observation[0, 1], marker=r'$\mathbf{?}$', color='lime', s

plt.scatter(X[indices, 0], X[indices, 1], facecolors='none', edgecolors='lime', s=200, linewidth=

plt.xlabel('Number of Rooms', fontsize=16)
plt.ylabel('Distance to Centre (km)', fontsize=16)
plt.title('Berlin Property Price Training Data', fontsize=18)
cbar = plt.colorbar(sc)
cbar.set_label('Price (€)', fontsize=14)
cbar.ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: format(int(x), ',')))
plt.legend(fontsize=12)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
```

Each dot represents a property, coloured by its price. The question mark (?) is the new property we want to price.

How would you predict the new observation's price? You could start by selecting its **5 closest neighbours**. The five closest neighbours are highlighted on the chart and shown in the table below:

Number of Rooms	Distance to Centre (km)	Price (k€)
3	2.7	341
3	2.4	401
3	2.9	358
4	3.0	377
4	2.3	425

Now, how could we generate a single prediction from this list?

The simplest approach would be to compute an **average** of all the neighbouring prices and use this as the prediction:

$$\text{Predicted Price} = \frac{451 + 467 + 457 + 468 + 436}{5} = \frac{2279}{5} = 456$$

That is it, we have our predictions! This process should remind you of probability predictions.

Using this method, we could also compute a **map of property prices** over the two features: number of rooms and distance from centre.

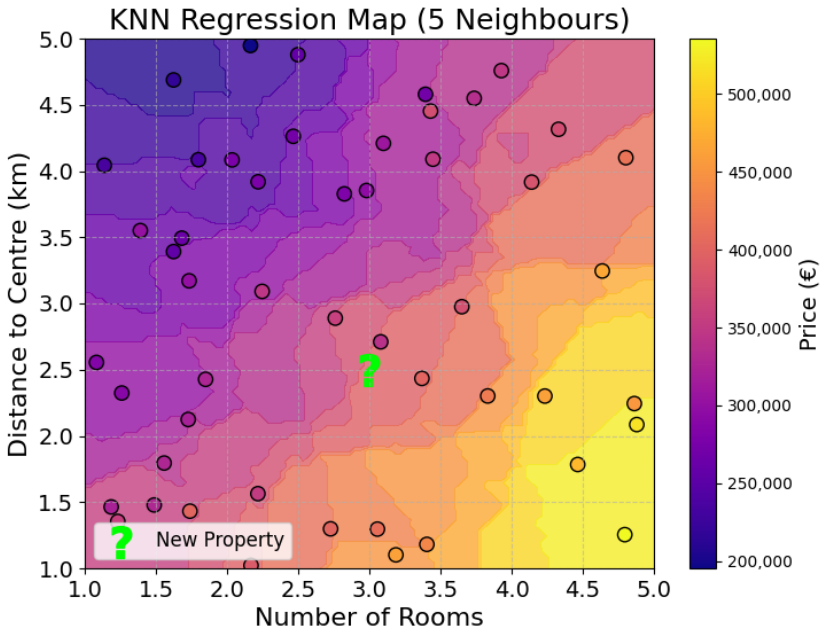


Figure 7.6: KNN regression map

Figure code

```
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X, price)

xx, yy = np.meshgrid(np.linspace(1, 5, 100), np.linspace(1, 5, 100))
Z = knn_reg.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, cmap='plasma', alpha=0.8, levels=20)
```

```
sc = plt.scatter(number_of_rooms, distance_to_centre, c=price, cmap='plasma', s=80, edgecolor='k'
plt.scatter(new_observation[0, 0], new_observation[0, 1], marker=r'$\mathbf{?}$', color='lime', s

plt.xlabel('Number of Rooms', fontsize=16)
plt.ylabel('Distance to Centre (km)', fontsize=16)
plt.title('KNN Regression Map (5 Neighbours)', fontsize=18)
cbar = plt.colorbar(sc)
cbar.set_label('Price (€)', fontsize=14)
cbar.ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: format(int(x), ',')))

plt.legend(fontsize=12)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
```

The map shows that there is a positive **relationship** between number of rooms and price, and the negative relationship between distance to centre and price. The most expensive properties are both central and have a high number of rooms. The (simple) model built allows us to map **any** given combination of number of rooms and distance to centre to a price.

7.5 Final Thoughts

KNN is only the first of the Machine Learning models studied in this book. This model learns the relationship between input features and a target, using **distance calculations** and **average over neighbours**. Can you think of a way you could use KNN to predict something in your everyday life?

We have our first model, but how good is it? We will explore this in the next chapter.

7.6 Practice Exercise

Exercise 7.2. Suppose you are building a model to detect fraudulent transactions. You use two features, both measured on a 0–100 scale:

- **Transaction Amount ($)** (0–100)
- **Customer Age (years)** (0–100)

You have the following 10 transactions in your training data:

Transaction Amount	Customer Age	Fraudulent?
95	22	Yes
90	25	Yes
92	23	Yes

Transaction Amount	Customer Age	Fraudulent?
97	21	Yes
93	24	Yes
94	23	No
20	80	No
25	78	No
18	82	No
23	77	No

A new transaction occurs with an amount of **93** and customer age **23**.

Question:
1. Calculate the distance from each observation to the new transaction.
2. Identify the 5 nearest neighbours and their labels.
3. What is the predicted probability that the new transaction is fraudulent?

7.7 Solutions

Solution 7.1. Exercise 7.1

$$\text{Probability} = \frac{4}{4+1} = \frac{4}{5} = 0.8 = 80\%$$

Solution 7.2. Exercise 7.2

The distance between two observations is calculated as:

$$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where (x_1, y_1) are the features of the training observation, and (x_2, y_2) are the features of the new transaction (93, 23).

Let's compute the distance for each observation:

ID	Transaction Amount	Customer Age	Fraudulent?	Distance to (93,23)
1	95	22	Yes	$\sqrt{(95-93)^2 + (22-23)^2} \approx 2.24$
2	90	25	Yes	$\sqrt{(90-93)^2 + (25-23)^2} = \approx 3.61$
3	92	23	Yes	$\sqrt{(92-93)^2 + (23-23)^2} = 1.00$
4	97	21	Yes	$\sqrt{(97-93)^2 + (21-23)^2} = \approx 4.47$

ID	Transaction Amount	Customer Age	Fraudulent?	Distance to (93,23)
5	93	24	Yes	$\sqrt{(93-93)^2+(24-23)^2} = 1.00$
6	94	23	No	$\sqrt{(94-93)^2+(23-23)^2} = 1.00$
7	20	80	No	$\sqrt{(20-93)^2+(80-23)^2} \approx 92.6$
8	25	78	No	$\sqrt{(25-93)^2+(78-23)^2} \approx 87.46$
9	18	82	No	$\sqrt{(18-93)^2+(82-23)^2} \approx 95.43$
10	23	77	No	$\sqrt{(23-93)^2+(77-23)^2} \approx 88.41$

Sorted by distance (nearest first):

ID	Distance	Fraudulent?
3	1.00	Yes
5	1.00	Yes
6	1.00	No
1	2.24	Yes
2	3.61	Yes

The 5 nearest neighbours are:

- 4 fraudulent: ID 3, 5, 1, 2
- 1 non-fraudulent: ID 6

Predicted probability of fraud:

$$P(\text{Fraud}) = \frac{4}{5} = 0.8 = 80\%$$

Part II

Model Evaluation

Chapter 8

Model Evaluation

We have now built our first model, but how good is it really? To answer this question, let us go back to the main function of a Machine Learning model. It is to generate predictions for **new observations** that are as **close** to the truth as possible.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

A good model makes predictions with **low distance** to the truth.

Taking this into account, how can we determine which model best performs its function?

There are two important elements to consider:

- New observations or unseen data
- Closeness to the truth

8.1 Unseen Data

Unseen data is, by definition, **unseen**. We cannot have access to it, as the moment we see it it becomes **seen**. Does that mean that we cannot evaluate a model on unseen data? Do we just wait for new observations to come?

8.1.1 Train-Test Split

One way to estimate a model's performance on new data is to set aside a **portion of the training data** for testing. This will remain unseen to the model, and will **not** be used for training.

At the end of the training process, the trained model (like K-Nearest Neighbours) can be used to generate predictions on the test data. For this portion of unseen data, we have both model predictions and the true label.

A good model will have predictions **as close as possible** to the true labels.

8.1.2 Representativeness

8.1.2.1 Simulating Prediction Conditions

We use a test set to **estimate** the performance of the model on unseen data.

For this estimation to make sense, the test set must be an **accurate representation** of what this unseen data will look like.

To make this more concrete, to evaluate a model trained to predict US property prices, it makes no sense to use Berlin flats as a test set. This is a bit of an extreme example, but illustrates the main idea.

In the training process, the model learns the relationship between input and output using observations on the training data.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

If the training data is not representative of the data that the model will generate predictions for, it cannot properly learn the relationship between input and output.

If the test data is not representative of the unseen data the model will be used to generate predictions on, the model performance on this dataset will not be a good estimation of future performance.

8.1.2.2 Ensuring Representativeness

But how do we make sure that the training and test sets are **consistent**? A single dataset can contain a lot of variations. In the property pricing example, there can be many types of properties with very different characteristics (e.g., surface area or number of rooms). Some of them are easier to price than others. How do we make sure that the train and test set look alike?

Here, the **law of large numbers** comes to the rescue. The law of large numbers states that if we **randomly sample** a large enough number of observations from a population (here, training data), the randomly selected sample will be representative of the original population.

This is the statistical foundation of **polling**. Before elections, polling agencies **randomly sample** a large group of people to get a representative sample of the population.

Some issues with polling

Polling suffers from selection bias. Some people will not reply to a call from a polling agency. Some groups are more likely to reply than others, which introduces a bias in the composition of the sample. As an example, retired individuals spend more time at home close to their phones and are more likely to pick up the phone.

To make sure that they obtain representative samples, these agencies have to use some sophisticated mathematical tricks to ensure representativeness. With mixed success.

Going back to Machine Learning, by randomly selecting a large number of observations from the training set, you can ensure that the test set is a representative sample of the training data.

This can be visualised with a simple example. Let's imagine that the training data contains both flats and single houses. These types of properties generally have different surface areas. The following chart shows the distribution of surface areas of both flats and houses for the entire data:

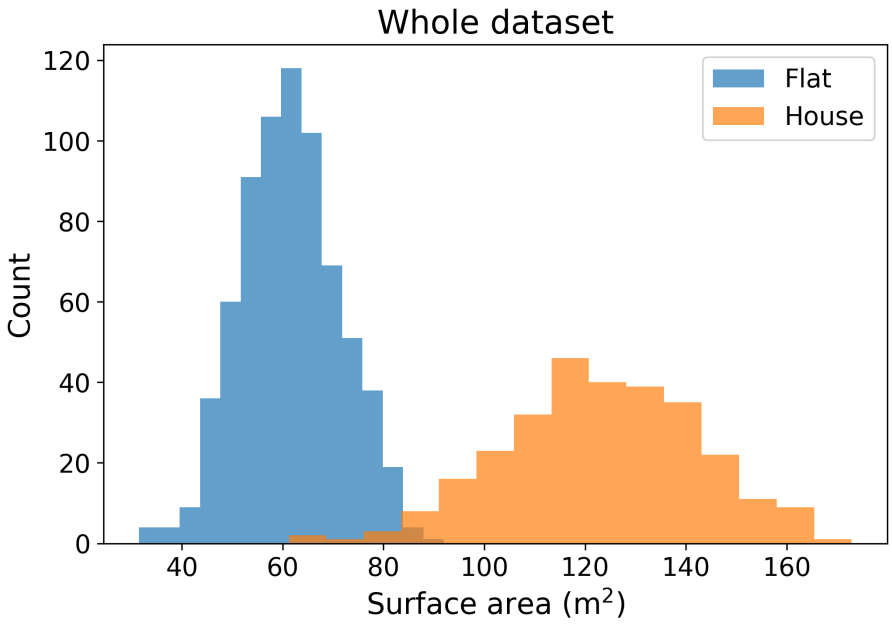


Figure 8.1: Surface area distribution for both Flats and Houses

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
```

```
np.random.seed(42)

# Simulate a population with two property types
n_total = 1000
types = np.random.choice(['Flat', 'House'], size=n_total, p=[0.7, 0.3])
areas = np.where(types == 'Flat', np.random.normal(60, 10, n_total), np.random.normal(

fig, ax = plt.subplots(figsize=(7, 5))

ax.hist(areas[types == 'Flat'], bins=15, alpha=0.7, label='Flat')
ax.hist(areas[types == 'House'], bins=15, alpha=0.7, label='House')
ax.set_title('Whole dataset', fontsize=18)
ax.set_xlabel('Surface area (m$^2$)', fontsize=16)
ax.set_ylabel('Count', fontsize=16)
ax.tick_params(axis='both', labelsize=14)
ax.legend(fontsize=14)

plt.tight_layout()
plt.show()
```

By randomly sampling 200 of the observations to create a test set, both the training and test sets have roughly similar distributions:

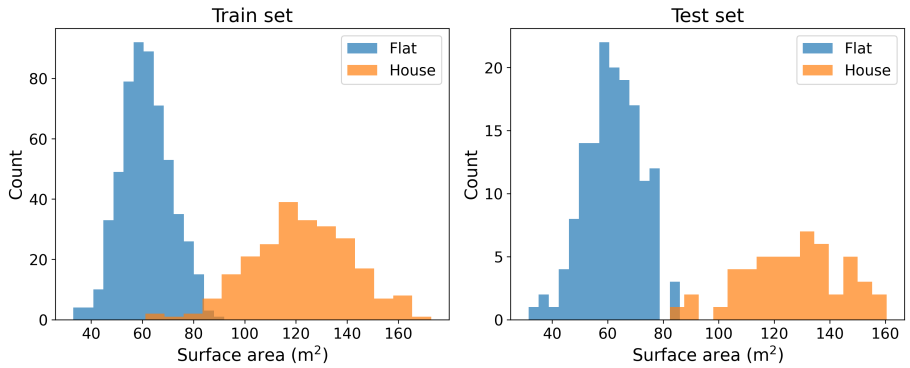


Figure 8.2: Surface area distributions in the training and test sets

Figure code

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
```

```
n_total = 1000
types = np.random.choice(['Flat', 'House'], size=n_total, p=[0.7, 0.3])
areas = np.where(types == 'Flat', np.random.normal(60, 10, n_total), np.random.normal(120, 20, n_

test_idx = np.random.choice(n_total, size=200, replace=False)
train_idx = np.setdiff1d(np.arange(n_total), test_idx)

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

for ax, idx, title in zip(axes, [train_idx, test_idx], ['Train set', 'Test set']):
    ax.hist(areas[idx][types[idx] == 'Flat'], bins=15, alpha=0.7, label='Flat')
    ax.hist(areas[idx][types[idx] == 'House'], bins=15, alpha=0.7, label='House')
    ax.set_title(title, fontsize=18)
    ax.set_xlabel('Surface area (m$^2$)', fontsize=16)
    ax.set_ylabel('Count', fontsize=16)
    ax.tick_params(axis='both', labelsize=14)
    ax.legend(fontsize=14)

plt.tight_layout()
plt.show()
```

This is the law of large numbers in action! The larger the dataset and test set, the more similar the two distributions will look like.

8.1.3 Information leakage

Randomly selecting a portion of the training data sounds good. But can you see an issue with this method?

Let us go back to the example of property prices. Property prices evolve over time. If I randomly select 20% of the past transactions as test set, the training set will see prices from the entire period. For instance, if the training data contains transactions from January 2023 to January 2025, the randomly selected test set will also contain transactions from January 2023 to January 2025.

This will give the model the opportunity to learn the price trend of the whole period and then predict **past prices**. A model trained until January 2025 predicting the price of a property sold in 2024 will benefit from **future knowledge**.

The performance of the model in pricing **past transactions** will not be a good estimation of how the model will predict **future property prices**. How would you estimate the performance of a model in predicting future prices?

One way to do so is to use a **time-based** train/test split. If the training data contains data from Jan 2023 to Jan 2025, you could keep the last two months of the data (Dec 2024 and Jan 2025) as test set and use the rest for training.

This way, you estimate model performance on future price prediction. This is

exactly how the model will be used once released. After training from Jan 2023 to Jan 2025, the model will be used to predict prices for Feb 2025 and beyond.

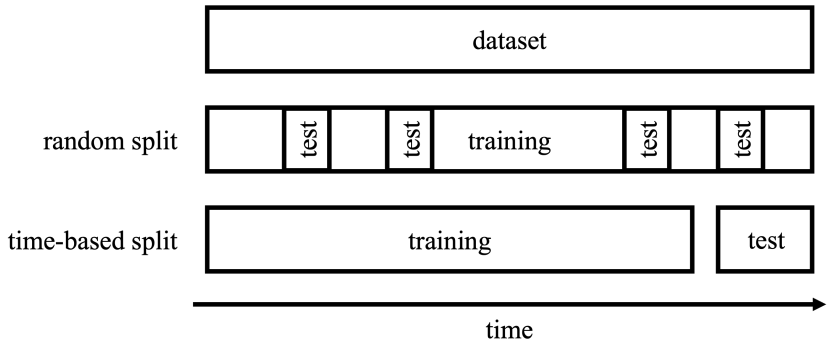


Figure 8.3: Random split vs time-based split

8.1.4 Wrapping up

Model evaluation is a critical aspect of Machine Learning practice. Before using model predictions in the real world, it is necessary to assess the model's accuracy; the quality of the model's predictions.

To do so, a part of the training data should be kept as **test set**. This test set should be representative of the training data. You should avoid information leakage by making sure that the model will not have access to **future information** when predicting on the test set.

8.2 Distance to the truth

Measuring the distance between the predictions and true labels depends on the type of problem. For classification tasks, predictions and ground truth will be class labels like "spam" or "malignant". In regression problems, predictions and ground truth will be numbers.

As shown in a previous chapter, calculating distances is a fascinating topic, that will be further explored in the next two chapters.

8.3 Final Thoughts

To evaluate the performance of a prediction model, this chapter reviewed the following method:

- Set aside a fraction of the training dataset as test set
- Train the model on the train set, do not use the test set
- Use the trained model to generate predictions on the test set

- Compute the distance between the predictions and the labels of the test set

Exactly how to measure this distance will be the topic of the next two chapters.

Chapter 9

Evaluating Classification Models

The previous chapter described the process of splitting a dataset between training and test sets to estimate a model's performance on **unseen data**. This evaluation process has the following four steps:

- Set aside a fraction of the training dataset as test set
- Train the model on the train set, do not use the test set
- Use the trained model to generate predictions on the test set
- Compute the distance between the predictions and the labels of the test

The previous chapter left this last point open: how to determine whether the predictions of a model are **close to the truth**? This chapter will tackle this problem for classification models. The following chapter will explore regression models.

As a reminder, classification tasks are concerned with predicting category labels such as "spam" or "malignant".

9.1 Evaluating Distances

Imagine that we trained two models, Model A and Model B. We now want to determine which of the two is best.

To do so, we remove a test set from the training data, and generate predictions using both models. For these observations, we also have the ground truth, the true label of each.

We get the following results by generating predictions for the test set:

Observation	Model A	Model B	Truth
1	×	∘	∘
2	×	∘	×
3	×	×	×
4	∘	∘	∘
5	∘	×	×

Which of the models is the most accurate?

You could start by counting the number of errors of each model:

- Model A made two errors: observation 1 and 5
- Model B made one error: observation 2

In other words, Model A was correct three times out of five whereas Model B was correct four times out of five. Model B seems **more correct on average**.

Sample Size

A sample of five observations is too low to draw conclusions about the performance of these two models. As a general rule, the **more test samples the better**. This allows us to get a better estimate of the performance of the model. As a rule of thumb, practitioners allocate between **5% and 20%** of the training data as a test set.

There is always a **trade-off** between the amount of data given to the model for training, and the number of observations in the test set. Data set aside for testing reduces the size of the training data available to the model which can result in lower model performance.

The later sections of this book will cover some methods designed to address this challenge.

More than just "three out of five", we can say that Model A was correct **60% of the time**. This metric is called the **accuracy** of a Machine Learning model.

$$\text{Accuracy} = \frac{\text{Number of observations correctly predicted}}{\text{Total observation count}}$$

Exercise 9.1. Calculate the accuracy of Model B.

When you hear the words "Model Accuracy" in the media, this is it.

9.1.1 Beyond Accuracy: Recall and Precision

But is accuracy sufficient? To answer this question, we need to ask another question: Are all errors the same? Do they have the same **consequences on the world**?

To do so, let us move beyond simple $\times$ and $\circ$ and into the world of tumour diagnosis. There, a benign mass misdiagnosed as a malignant tumour would generate stress and inconvenience. A malignant tumour misdiagnosed as benign could have **fatal consequences**.

Let us now consider the two following models, with $\circ$ representing a **benign mass** and $\times$ representing a **malignant tumour**:

Observation	Model A	Model B	Truth
1	$\circ$	$\circ$	$\circ$
2	$\circ$	$\circ$	$\circ$
3	$\circ$	$\times$	$\circ$
4	$\circ$	$\times$	$\circ$
5	$\circ$	$\times$	$\times$
6	$\times$	$\times$	$\times$
7	$\times$	$\times$	$\times$
8	$\times$	$\times$	$\times$

Exercise 9.2. Show that model A has a higher accuracy than model B.

Even though model A has higher accuracy than model B, it misclassified one malignant tumour as **benign** (Observation 5).

On the other hand, model B classified two benign masses as malignant (Observations 3 and 4) but **caught all the malignant tumours**.

This goes to show that Accuracy is only a part of the picture. How can we move from the description above to actual metrics?

9.1.1.1 Useful Vocabulary

Before going into error metrics, it is important to introduce some vocabulary.

In binary classification, the model learns to assign observations into two categories, such as "benign" and "malignant" in the case of tumour diagnosis, or "spam" and "non-spam" for email filtering.

Mathematically, these two labels are represented as 1 and 0. Generally, the class that the model was built to detect is assigned 1 and the other 0. In tumour diagnosis, the malignant label is generally assigned the number 1 as these are the cases the model was designed for. Similarly, in email filtering, the label "spam" is assigned the number 1, as the model aims at identifying spam messages to filter them out of the inbox.

Exercise 9.3. If you were building a fraud detection model for an online payments company, which labels would you predict for? Which one would be assigned to 1 and 0?

In the example of tumour diagnosis, a malignant tumour correctly classified as "malignant" is called a **True Positive**. On the other hand, a malignant tumour misclassified as a benign mass is a **False Negative**.

Here, the words "positive" or "negative" are not associated with any value judgement. They are simply another way to say 1 or 0. Thinking about medical examples may make more sense here. When a medical test is positive, it means that the targeted substance is **present**. The same applies to spam detection. A positive spam detection means classifying an email as "spam".

Going back to jargon, a **True Positive** is when the variable of interest is correctly detected (e.g., "spam" or "malignant"). A **False Negative** is when the variable of interest goes **undetected**. As an example, a malignant tumour is misdiagnosed as "benign", or a spam email landing into the inbox.

Based on these, what would be a **False Positive** and a **True Negative**? Think about it in terms of tumours and spam emails before reading on.

- **False Positive**: model **wrongly** predicts the presence of the variable of interest, e.g., a legitimate email predicted as "spam"
- **True Negative**: model **correctly** predicts the absence of the variable of interest, e.g., a benign mass is correctly classified as a benign mass

These can be summarised in the following table:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

This is also called the **Confusion Matrix**.

To make this more concrete, this table could be adapted to the tumour diagnosis example:

	Predicted Malignant	Predicted Benign
Actual Malignant	Malignant tumour correctly classified as "malignant"	Malignant tumour incorrectly classified as "benign"
Actual Benign	Benign mass incorrectly classified as "malignant"	Benign mass correctly classified as "benign"

Or for spam filtering:

	Predicted Spam	Predicted Not Spam
Actual Spam	Spam correctly classified as "spam"	Spam incorrectly classified as "not spam"
Actual Not Spam	Legitimate email incorrectly classified as "spam"	Legitimate email correctly classified as "not spam"

To test your understanding, try building a Confusion Matrix for a payment fraud detection model.

	Predicted Fraudulent	Predicted Legitimate
Actual Fraudulent	Fraudulent transaction correctly classified as "fraudulent"	Fraudulent transaction incorrectly classified as "legitimate"
Actual Legitimate	Legitimate transaction incorrectly classified as "fraudulent"	Legitimate transaction correctly classified as "legitimate"

Now that we clearly understand the language of True/False Negative/Positive, let's get back to measuring the performance of a Machine Learning model.

9.1.1.2 Recall

In the example of tumour diagnosis, we would like a model that would catch all malignant tumours. This is because False Negatives, i.e. misdiagnosing a malignant tumour as benign, can have **fatal** consequences. We want to compare the performances of Model A and B:

Observation	Model A	Model B	Truth
1	∘	∘	∘
2	∘	∘	∘
3	∘	×	∘
4	∘	×	∘
5	∘	×	×
6	×	×	×
7	×	×	×
8	×	×	×

Recall is the metric that answers the question: out of all the positive cases, how many did the model catch?

Rephrasing this for the tumour diagnosis example: out of all the malignant tumours, how many did the model catch?

This is calculated as follows:

$$\text{Recall} = \frac{\text{Positive examples caught by Model}}{\text{All positive examples}}$$

Rephrasing this in term of True/False Positive, we get:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Calculating Recall for model B:

$$\text{Recall}_{\text{Model B}} = \frac{4}{4+0} = 100\%$$

Exercise 9.4. Calculate the recall of Model A, and prove that it is 75.

In the case of disease diagnosis, Recall is a critical metric as **missing positive cases** can have dire consequences on a patient's life.

9.1.1.3 Precision

In other scenarios, when the cost of a False Positive is high, we care for the **Precision** of the model. In other words, we want to **avoid False Positives**.

Precision answers the question: Out of all the observations predicted as positive, how many were True Positives, i.e., actually positive?

Building a spam detection algorithm, the objective is to classify incoming emails as either "spam" or "non-spam". Every email classified as "spam" would be **filtered out of the inbox**. In line with the previous section, the positive case would be "spam", as it is the case that would require action; filtering email out of the inbox.

In spam filtering, False Positives can have serious negative consequences. Once, a recruiter's email ended up in my spam folder. Without her reminder, I would have never worked at my current company because of a spam filtering error.

For this reason, the most important metric here is Precision: out of the messages classified as spam, how many were actually spam messages?

This can be computed as follows:

$$\text{Precision} = \frac{\text{Number of emails correctly classified as spam}}{\text{Total number of emails classified as spam}}$$

Rephrasing this expression in Confusion Matrix jargon:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Looking at the example below:

Observation	Model A	Model B	Truth
1	∘	∘	∘
2	∘	∘	∘
3	∘	×	∘
4	∘	×	∘
5	∘	×	×
6	×	×	×
7	×	×	×
8	×	×	×

the precision of Model A is:

$$\text{Precision}_{\text{Model A}} = \frac{3}{3+0} = 100\%$$

Exercise 9.5. Calculate the Precision of Model B.

9.1.1.4 Revisiting Accuracy

Accuracy, the first error metric explored in this chapter, can also be calculated with Confusion Matrix terms.

As a reminder, accuracy is calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of observations correctly predicted}}{\text{Total observation count}}$$

Using the language of True/False Positive/Negative, it can be computed with the following formula:

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Observation Count}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

This section described Accuracy, Recall, Precision and the Confusion Matrix. It is now to apply them to model selection; to choose the best performing model for a given task.

9.2 Practical Model Selection

After having built two different Machine Learning tumour diagnosis models (A and B), you get the following Confusion Matrices:

Model A

	Predicted Malignant	Predicted Benign
Actual Malignant	40	10
Actual Benign	10	40

Model B

	Predicted Malignant	Predicted Benign
Actual Malignant	45	5
Actual Benign	5	45

Which model you pick?

If you picked B, that is correct. Why did you choose it?

Exercise 9.6. If you have not done so already, compute the Accuracy, Precision and Recall of both models.

Making this decision more complex, which of the following two models would you pick?

Model A

	Predicted Malignant	Predicted Benign
Actual Malignant	48	2
Actual Benign	18	32

Model B

	Predicted Malignant	Predicted Benign
Actual Malignant	50	0
Actual Benign	20	30

If you picked B, that is correct again. Why did you choose model B? In this case, model B has the same Accuracy and the **highest Recall**. In the case of tumour diagnosis, this is probably the most important metric to look at.

Would you pick a different model for spam detection? Probably, as **Precision** becomes more important then. You do not want legitimate emails to end up in your spam folder.

Exercise 9.7. Calculate Recall and Precision for model A and B

9.3 Probabilities and Model Evaluation

For the sake of simplicity, this chapter has only considered **binary predictions**: either malignant or benign, either spam or non-spam.

As we have seen in the KNN chapter, classification models can also output **predicted probabilities**. Instead of simply predicting an observation as "malignant" or "benign", the model can output a predicted probability of malignancy.

To convert these probabilities to a binary label, a **threshold** of 0.5 is generally used. Any predicted probability beyond this threshold (here 0.5) would be classified as "malignant", otherwise, it would be classified as "benign". For example, a predicted probability of 48% would be classified as "benign", while a predicted probability of 51% as "malignant".

The threshold can be any number between 0 and 1. The lower the threshold, the **higher** the number of Positives. In the example of tumour diagnosis, this would mean a higher number of observations predicted as "malignant". On the other hand, a higher threshold would lead to **fewer** positives. In the example of spam detection, this would lead to fewer emails classified as "spam".

Let us show this with an example:

Observation	Predicted Probability	Threshold 0.3	Threshold 0.5	Threshold 0.7	Ground Truth
1	0.6	×	×	∘	×
2	0.75	×	×	×	×
3	0.2	∘	∘	∘	∘
4	0.8	×	×	×	×
5	0.4	×	∘	∘	∘
6	0.1	∘	∘	∘	∘

Which translates to the following Confusion Matrices:

Threshold 0.3

	Predicted ×	Predicted ∘
Actual ×	3	0
Actual ∘	1	2

Threshold 0.5

	Predicted ×	Predicted ∘
Actual ×	3	0
Actual ∘	0	3

Threshold 0.7

	Predicted ×	Predicted ∘
Actual ×	2	1
Actual ∘	0	3

The following line chart shows the evolution of the different metrics described in this chapter as the prediction **threshold increases**:

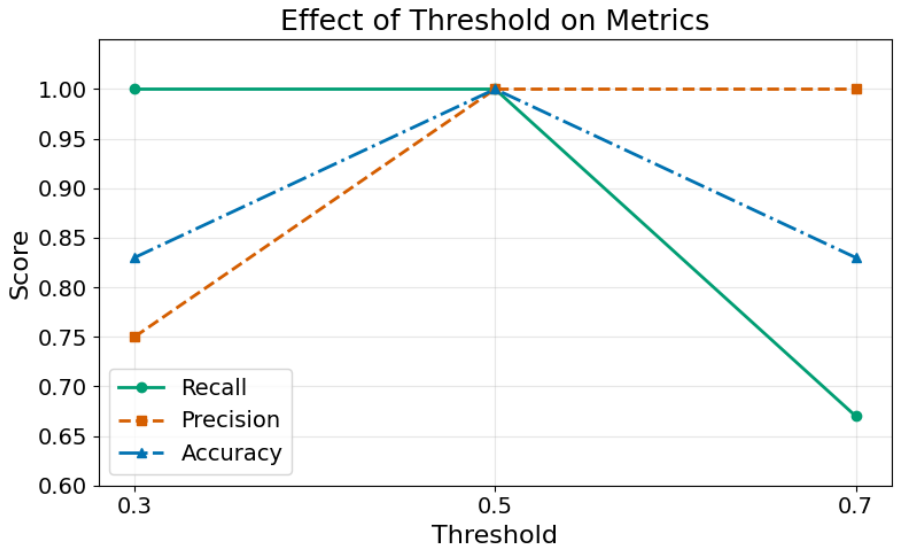


Figure 9.1: Visualising recall and precision as the threshold increases

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

thresholds = [0.3, 0.5, 0.7]
recall = [1.0, 1.0, 0.67]
```

```
precision = [0.75, 1.0, 1.0]
accuracy = [0.83, 1, 0.83]

plt.figure(figsize=(8,5))
plt.plot(thresholds, recall, marker='o', label='Recall')
plt.plot(thresholds, precision, marker='o', label='Precision')
plt.plot(thresholds, accuracy, marker='o', label='Accuracy')
plt.xlabel('Threshold', fontsize=16)
plt.ylabel('Score', fontsize=16)
plt.title('Effect of Threshold on Metrics', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(fontsize=14)
plt.grid(True)
plt.tight_layout()
plt.show()
```

We see that **decreasing** the prediction threshold to 0.3 has the following effects on the three metrics we analysed:

- **Increase** in Recall: with more observations predicted as "malignant", recall can only increase or stay constant
- **Decrease** in Precision: with more observations predicted as "malignant" despite a low predicted probability, the risk of misclassifying observations as "malignant" increases

Exercise 9.8. Describe the effect of increasing the threshold on Precision, Recall and Accuracy.

For tumour diagnosis, a **lower threshold** may make more sense. Going back to the Nearest Neighbour example, from a patient's perspective, if the observation has two malignant neighbours and three benign ones, for a predicted probability of $2/5 = 40\%$, I would still want further checks.

For spam filtering, a **higher threshold** could be preferred, to reduce the risk of a legitimate email being filtered out. You could filter out results only when they have predicted probability of 0.8 or 0.9.

This example illustrates the **Precision/Recall trade-off**. Setting a higher threshold will increase Precision and reduce Recall. Setting a lower threshold will reduce Precision and increase Recall.

9.4 Final Thoughts

This section has shown how to evaluate a binary classification model. There are three steps to this process:

- Set aside a share of the training data as test set

- Using the trained model, generate predictions on this test set
- Calculate the distance to the truth of the predictions generated using performance metrics such as Accuracy, Recall, and Precision

It is important to remember that there is no optimal performance metric. The best metric for a problem depends on **the consequences of model error** on the real world.

After this description of **classification** model evaluation, the next section will explore the evaluation of **regression models**.

9.5 Solutions

Solution 9.1. Exercise 9.1

Accuracy of Model B $= \frac{4}{5} = 80\%$

Solution 9.2. Exercise 9.2

Model A: Correct on observations 1, 2, 3, 4, 6, 7, 8 (7 out of 8). Incorrect on 5.

Model B: Correct on 1, 2, 5, 6, 7, 8 (6 out of 8). Incorrect on 3 and 4.

Accuracy of Model A $= \frac{7}{8} = 87.5\%$

Accuracy of Model B $= \frac{6}{8} = 75\%$

Solution 9.3. Exercise 9.3 Labels:

- fraudulent transaction 1
- non fraudulent or legitimate transaction 0

Solution 9.4. Exercise 9.4

Model A: Out of 4 malignant tumours (observations 5, 6, 7, 8), Model A caught 3 (6, 7, 8), True Positives. Missed 5, a False Negative.

$\text{Recall}_{\text{Model A}} = \frac{3}{3+1} = 75\%$

Solution 9.5. Exercise 9.5

$$\text{Precision}_{\text{Model B}} = \frac{4}{4+2} = \frac{4}{6} \approx 67\%$$

Solution 9.6. Exercise 9.6

For Model A:

- TP = 40
- TN = 40
- FP = 10
- FN = 10

Accuracy = $\frac{40+40}{100} = 80\%$

Precision = $\frac{40}{40+10} = \frac{40}{50} = 80\%$

Recall = $\frac{40}{40+10} = \frac{40}{50} = 80\%$

For Model B:

- TP = 45
- TN = 45
- FP = 5
- FN = 5

Accuracy = $\frac{45+45}{100} = 90\%$

Precision = $\frac{45}{45+5} = \frac{45}{50} = 90\%$

Recall = $\frac{45}{45+5} = \frac{45}{50} = 90\%$

Solution 9.7. Exercise 9.7

For Model A:

- TP = 48
- TN = 32
- FP = 18
- FN = 2

Precision = $\frac{48}{48+18} = \frac{48}{66} \approx 73\%$

Recall = $\frac{48}{48+2} = \frac{48}{50} = 96\%$

For Model B:

- TP = 50
- TN = 30
- FP = 20
- FN = 0

Precision = $\frac{50}{50+20} = \frac{50}{70} \approx 71\%$

Recall = $\frac{50}{50+0} = \frac{50}{50} = 100\%$

Solution 9.8. Exercise 9.8

Increasing the threshold generally:

- **Increases** Precision: fewer observations are classified as positive, so those that are classified as positive are more likely to be true positives
- **Decreases** Recall: more actual positives are missed, as the model becomes more conservative

Chapter 10

Evaluating Regression Models

The previous chapter described how to evaluate classification models. The main approach was simple:

- Set aside a fraction of the training dataset as test set
- Train the model on the train set, do not use the test set
- Use the trained model to generate predictions on the test set
- Compute the distance between the predictions and the labels of the test

The exact same approach can be applied to regression models. The only difference with the previous chapter is the computation of the distance between predictions and true labels.

As a reminder, a regression task is the prediction of **any continuous quantity**, such as the temperature tomorrow, the price of a property or a crop yield.

The general idea is still the same:

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

10.1 Evaluating Distances

Let's start with a simple example of a model predicting the price of a property:

Property	Predicted Price	Actual Price
1	140	120
2	110	100
3	100	130

How would you measure the error of this model? One way to do this is to measure the **distance** between each prediction and the ground truth using the distance functions explored in the Distance chapter.

The error of each prediction can be computed with the following formula:

$$\text{Prediction Error} = \text{Prediction} - \text{Ground Truth}$$

The error in the prediction for property 1 is:

$$\text{Prediction Error}_1 = 140 - 120 = 20$$

The model over-predicted the price of the property by 20.

Exercise 10.1. Show that the prediction errors for property 2 and 3 are 10 and -30 respectively.

Solution 10.1. You should get the following table:

Property	Predicted Price	Actual Price	Prediction Error
1	140	120	20
2	110	100	10
3	100	130	-30

Now, how can we **aggregate** these errors to come up with an evaluation of the model? One idea would be to compute the **average error** (for n observations):

$$\text{Average Error} = \frac{\text{Error}_1 + \text{Error}_2 + \cdots + \text{Error}_n}{n}$$

Computing the average error of the example model, we get:

$$\text{Average Error} = \frac{20 + 10 + (-30)}{3} = \frac{0}{3} = 0$$

Using the Σ operator described in the Distance chapter, this notation can be made more compact:

$$\text{Average Error} = \frac{1}{n}\sum_{i=1}^{n}(\text{Prediction}_i - \text{Truth}_i)$$

If you are not familiar with the Σ operator, please refer to the Distance chapter in which this concept is clearly explained.

10.2 Beyond Subtraction

Do you notice something strange? The average model error is 0, which would describe a *perfect model*. Yet, we do see that this model is **not** perfect, it makes errors, it does not perfectly predict the actual price of any property listed above.

The issue with averaging errors is that they cancel out. The numerator of the fraction becomes 0.

Remembering the Distance chapter, could we use another distance function to avoid this problem?

There are two main methods we could use:

- Averaging the **absolute values** of the errors
- Averaging the **squared errors**

10.2.1 Mean Absolute Error

The absolute value of a number x is noted $|x|$. It is the **magnitude** of the number, regardless of its sign. As an example $|2| = |-2| = 2$. When we average the absolute values of the error, we can compute the **Mean Absolute Error (MAE)**:

$$\text{Mean Absolute Error} = \frac{|\text{Error}_1| + |\text{Error}_2| + \cdots + |\text{Error}_n|}{n}$$

Using the Σ notation to make this more compact:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\text{Prediction}_i - \text{Truth}_i|$$

Computing the Mean Absolute Error for the example data:

Property	Predicted Price	Actual Price	Prediction Error
1	140	120	20
2	110	100	10
3	100	130	-30

We get:

$$\text{MAE} = \frac{|20| + |10| + |-30|}{3} = \frac{20 + 10 + 30}{3} = \frac{60}{3} = 20$$

This metric is much better than the average error as it gives us an idea of the average distance between individual predictions and the ground truth. Looking at the example data, the model is on average 20 away from the ground truth.

10.2.2 Mean Squared Error

Another way to do so is to compute the average of the **squared errors**. This metric is called the **Mean Squared Error (MSE)**:

$$\text{Mean Squared Error} = \frac{\text{Error}_1^2 + \text{Error}_2^2 + \cdots + \text{Error}_n^2}{n}$$

Or, in Σ notation:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{Prediction}_i - \text{Truth}_i)^2$$

This method has the advantage of turning every error into a positive number before averaging. This way, errors do not cancel out.

Computing the Mean Squared Error for the example data, we get:

$$\text{MSE} = \frac{20^2 + 10^2 + (-30)^2}{3} = \frac{400 + 100 + 900}{3} = \frac{1400}{3} \approx 466.67$$

Do you notice something strange? The resulting metric is **much larger** than expected, beyond the scale of the original errors.

One way to make this metric more interpretable is to take the square root of this number:

$$\sqrt{466.67} \approx 21.6$$

This is called the **Root Mean Squared Error (RMSE)**, another commonly used regression model performance metric. It is computed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{Prediction}_i - \text{Truth}_i)^2} = \sqrt{\text{MSE}}$$

10.2.3 Average Error is still useful

Does that mean that we should never use average Error? Not exactly.

The Average Error is still a useful metric to see if a prediction model has a **bias**; i.e., if a model consistently over- or under-predicts. If the average error of the model is not close to 0, it means that there is **systematic** over- or under-prediction.

This can be illustrated with the following example:

Property	Predicted Price	Actual Price
1	150	120
2	130	100
3	140	130

Exercise 10.2. Calculate the Average Error of this model and determine whether the model over- or under-predicts.

Solution 10.2.

$$\begin{aligned}\text{Prediction Error}_1 &= 150 - 120 = 30\\ \text{Prediction Error}_2 &= 130 - 100 = 30\\ \text{Prediction Error}_3 &= 140 - 130 = 10\\ \text{Average Error} &= \frac{30 + 30 + 10}{3} = \frac{70}{3} \approx 23.33\end{aligned}$$

Since the average error is positive, the model **over-predicts**.

10.3 Practice Exercise

Looking at these two pricing models, which one would you pick?

Property	Model A Prediction	Model B Prediction	Truth
1	150	140	120
2	110	100	100
3	100	140	140

Exercise 10.3. Show that Model B generates predictions that are closer to the truth than Model A, using the error metrics shown above.

Solution 10.3. **Model A:**

- Errors: $150 - 120 = 30$, $110 - 100 = 10$, $100 - 140 = -40$
- Average Error: $\frac{30+10+(-40)}{3} = 0$
- MAE: $\frac{|30|+|10|+|-40|}{3} = \frac{30+10+40}{3} = 26.67$
- MSE: $\frac{30^2+10^2+(-40)^2}{3} = \frac{900+100+1600}{3} = \frac{2600}{3} \approx 866.67$
- RMSE: $\sqrt{866.67} \approx 29.43$

Model B:

- Errors: $140 - 120 = 20$, $100 - 100 = 0$, $140 - 140 = 0$
- Average Error: $\frac{20+0+0}{3} = 6.67$
- MAE: $\frac{|20|+|0|+|0|}{3} = \frac{20}{3} \approx 6.67$

- MSE: $\frac{20^2+0^2+0^2}{3} = \frac{400}{3} \approx 133.33$
- RMSE: $\sqrt{133.33} \approx 11.55$

All metrics show that Model B is **closer** to the truth.

10.4 Choosing Between Metrics

This chapter has introduced four performance metrics:

- Average Error
- Mean Absolute Error
- Mean Squared Error
- Root Mean Squared Error

In the example above, all metrics **agreed**; in other words, all metrics gave an advantage to Model B. This is not always the case.

Exercise 10.4. Compute the MSE and MAE of the two models below:

Property	Model A Prediction	Model B Prediction	Truth
1	115	110	100
2	105	110	120
3	125	130	140
4	100	90	150

Solution 10.4. **Model A:**

- Errors: $115-100 = 15$, $105-120 = -15$, $125-140 = -15$, $100-150 = -50$
- MAE: $\frac{|15|+|-15|+|-15|+|-50|}{4} = \frac{15+15+15+50}{4} = \frac{95}{4} = 23.75$
- MSE: $\frac{15^2+(-15)^2+(-15)^2+(-50)^2}{4} = \frac{225+225+225+2500}{4} = \frac{3175}{4} = 793.75$

Model B:

- Errors: $110-100 = 10$, $110-120 = -10$, $130-140 = -10$, $90-150 = -60$
- MAE: $\frac{|10|+|-10|+|-10|+|-60|}{4} = \frac{10+10+10+60}{4} = \frac{90}{4} = 22.5$
- MSE: $\frac{10^2+(-10)^2+(-10)^2+(-60)^2}{4} = \frac{100+100+100+3600}{4} = \frac{3900}{4} = 975$

As you can see, Model A has a higher MAE than Model B, and Model B has a higher MSE than model A. How can that be? To understand the reason why, we need to understand the difference between the absolute value and the squared value of a number. These two functions are visualised below:

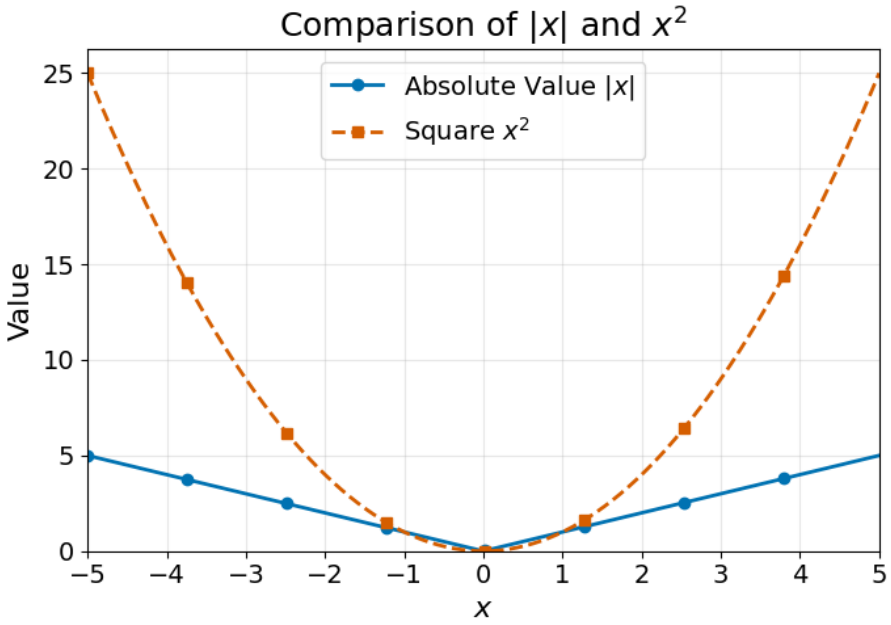


Figure 10.1: Absolute value and square functions

Figure code

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 200)
abs_x = np.abs(x)
sq_x = x**2

plt.figure(figsize=(7,5))
plt.plot(x, abs_x, label='Absolute Value $|x|$', linewidth=2)
plt.plot(x, sq_x, label='Square $x^2$', linewidth=2)
plt.title('Comparison of $|x|$ and $x^2$', fontsize=18)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('Value', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(fontsize=14)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

You may see that the square function **increases faster** as numbers grow larger or more negative, whereas the growth rate of the absolute value function **stays constant**. This means that the Mean Squared Error will strongly penalise **extreme errors**. Remembering the previous example:

Property	Model A Prediction	Model B Prediction	Truth
1	115	110	100
2	105	110	120
3	125	130	140
4	100	90	150

Model B has a very high prediction error on property 4 (-60), which results in a higher MSE value than Model A, despite having a lower Mean Absolute Error. What to do in these cases?

First, this does not happen often. It was a bit of work to build an example that would show this edge case.

Practical considerations aside, there is no one-size-fits-all metric. The best metric is the one that best measures the **consequences of an error** in the real world.

Taking the example of property pricing, **large errors** can have a strong negative impact:

- Over-pricing may lead to **financial losses** as the pricing company may not be able to sell the property
- Under-pricing may **reduce** the number of deals a real estate company can make

Given these considerations, the Mean Squared Error is a better choice.

There are also mathematical reasons why the Mean Squared Error and Root Mean Squared Errors are **sometimes preferred** over the Mean Absolute Error. One of them is explained in the note below.

Mean Square Error and Differentiability

The Mean Absolute Error (MAE) function is **not differentiable** at 0, because the absolute value function $|x|$ has a "sharp corner" at $x = 0$.

This is an issue for many machine learning algorithms, which rely on **differentiability** for optimisation (such as gradient descent). In contrast, the Mean Squared Error (MSE) is differentiable everywhere, making it easier to use for training models.

10.5 Final Thoughts

This section described regression model evaluation. It is very similar to the approach used for classification models described in the previous chapter:

- Set aside a share of the training data as a test set
- Using the trained model, generate predictions on this test set
- Calculate the distance to the truth of the predictions generated using performance metrics such as Average Error, MAE, MSE or RMSE

The main difference is the **distance calculation** used.

This is it for model evaluation. The next chapter will introduce this book's second Machine Learning model architecture: Decision Trees.

Part III

Decision Trees

Chapter 11

Decision Trees

It is now time to introduce a second model architecture. How would you predict the class of the unknown observation **without** using distance functions or nearest neighbours?

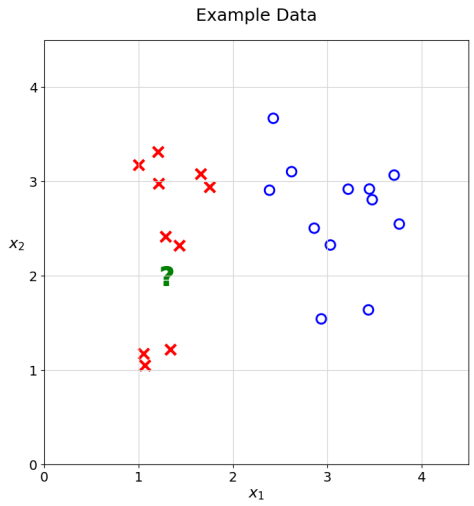


Figure 11.1: New observation and example data

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig, ax = plt.subplots(figsize=(8, 8))

ax.set_xticks(np.arange(0, 5, 1))
ax.set_yticks(np.arange(0, 5, 1))
ax.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)

ax.set_xlim(0, 4.5)
ax.set_ylim(0, 4.5)

ax.set_xlabel('$x_1$', fontsize=16)
ax.set_ylabel('$x_2$', rotation=0, ha='right', fontsize=16)

ax.tick_params(axis='both', which='major', labelsize=14)
np.random.seed(0)
x1_plus = np.random.uniform(0.5, 1.8, 10)
x2_plus = np.random.uniform(1.0, 3.5, 10)
ax.scatter(x1_plus, x2_plus, marker='x', color='red', s=120, linewidths=3, label='Class

x1_circle = np.random.uniform(2.2, 3.8, 12)
x2_circle = np.random.uniform(1.5, 3.8, 12)
ax.scatter(x1_circle, x2_circle, marker='o', color='blue', s=100, facecolors='none', ed

plt.scatter(1.3, 2, marker=r'$\mathbf{?}$', color='green', s=400)


ax.set_title('Example Data', fontsize=18, pad=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.savefig("images/trees/example_data.png")
plt.show()
```

One possibility would be to **split the data at** $x_1 = 2$. Any observation with x_1 greater than 2 would be classified as $\circ$; the rest would be classified as $\times$.

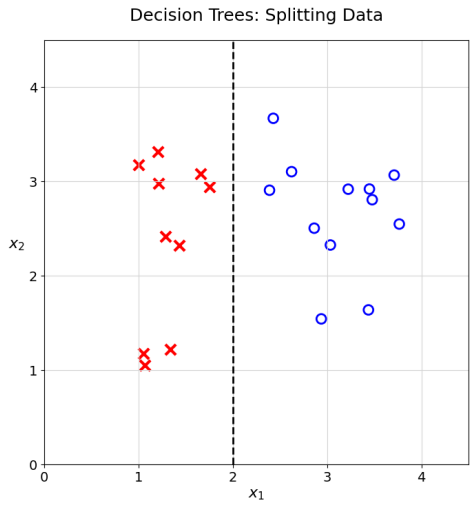


Figure 11.2: Example data with a single split

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(8, 8))

ax.set_xticks(np.arange(0, 5, 1))
ax.set_yticks(np.arange(0, 5, 1))
ax.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)

ax.set_xlim(0, 4.5)
ax.set_ylim(0, 4.5)

ax.set_xlabel('$x_1$', fontsize=16)
ax.set_ylabel('$x_2$', rotation=0, ha='right', fontsize=16)

ax.tick_params(axis='both', which='major', labelsize=14)
np.random.seed(0)
x1_plus = np.random.uniform(0.5, 1.8, 10)
x2_plus = np.random.uniform(1.0, 3.5, 10)
ax.scatter(x1_plus, x2_plus, marker='+', color='red', s=120, linewidths=3)

x1_circle = np.random.uniform(2.2, 3.8, 12)
x2_circle = np.random.uniform(1.5, 3.8, 12)
```

```
ax.scatter(x1_circle, x2_circle, marker='o', color='blue', s=100, facecolors='none', ec

ax.plot([2, 2], [0, 4.5], color='black', linestyle='--', linewidth=2)

ax.set_title('Decision Trees: Splitting Data', fontsize=18, pad=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.savefig("images/trees/split.png")
plt.show()
```

11.1 Multiple Splits

For a more complex case, how would you predict the class of the unknown observation with this training data? How would you split the × from the ◦?

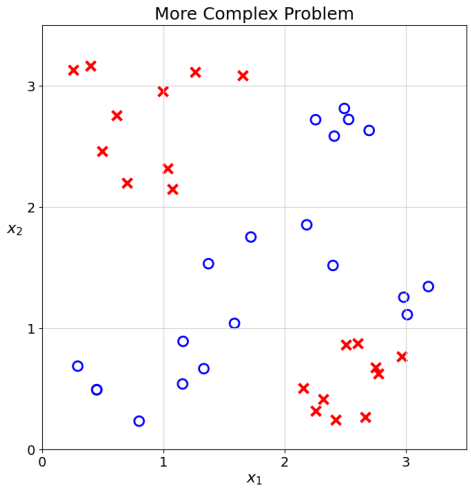


Figure 11.3: A more complex example problem

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(8, 8))

ax.set_xticks(np.arange(0, 4, 1))
ax.set_yticks(np.arange(0, 4, 1))
ax.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)
```

```
ax.set_xlim(0, 3.5)
ax.set_ylim(0, 3.5)

ax.set_xlabel('$x_1$', fontsize=16)
ax.set_ylabel('$x_2$', rotation=0, ha='right', fontsize=16)

ax.tick_params(axis='both', which='major', labelsize=14)

np.random.seed(42)
x1_region1 = np.random.uniform(0.2, 1.8, 10)
x2_region1 = np.random.uniform(0.2, 1.8, 10)
ax.scatter(x1_region1, x2_region1, marker='o', color='blue', s=100, facecolors='none', edgecolors

x1_region2 = np.random.uniform(2.1, 3.2, 10)
x2_region2 = np.random.uniform(0.2, 0.9, 10)
ax.scatter(x1_region2, x2_region2, marker='x', color='red', s=100, linewidths=3, label='Class X')

x1_region3 = np.random.uniform(0.2, 1.8, 10)
x2_region3 = np.random.uniform(2.1, 3.2, 10)
ax.scatter(x1_region3, x2_region3, marker='x', color='red', s=100, linewidths=3)

x1_region4 = np.random.uniform(2.1, 3.2, 10)
x2_region4 = np.random.uniform(1.1, 3.2, 10)
ax.scatter(x1_region4, x2_region4, marker='o', color='blue', s=100, facecolors='none', edgecolors

ax.set_title('More Complex Problem', fontsize=18)
plt.gca().set_aspect('equal', adjustable='box')
plt.savefig("images/trees/complex_problem.png")
plt.show()
```

In this example, there is **no single line** that can perfectly split the two groups. In Computer Science, this is also called the **XOR problem**.

XOR Problem

The XOR (Exclusive Or) Problem refers to a foundational Machine Learning and Computer Science problem in which two classes cannot be separated by any single straight line. It demonstrates **the weaknesses of simple linear models**.

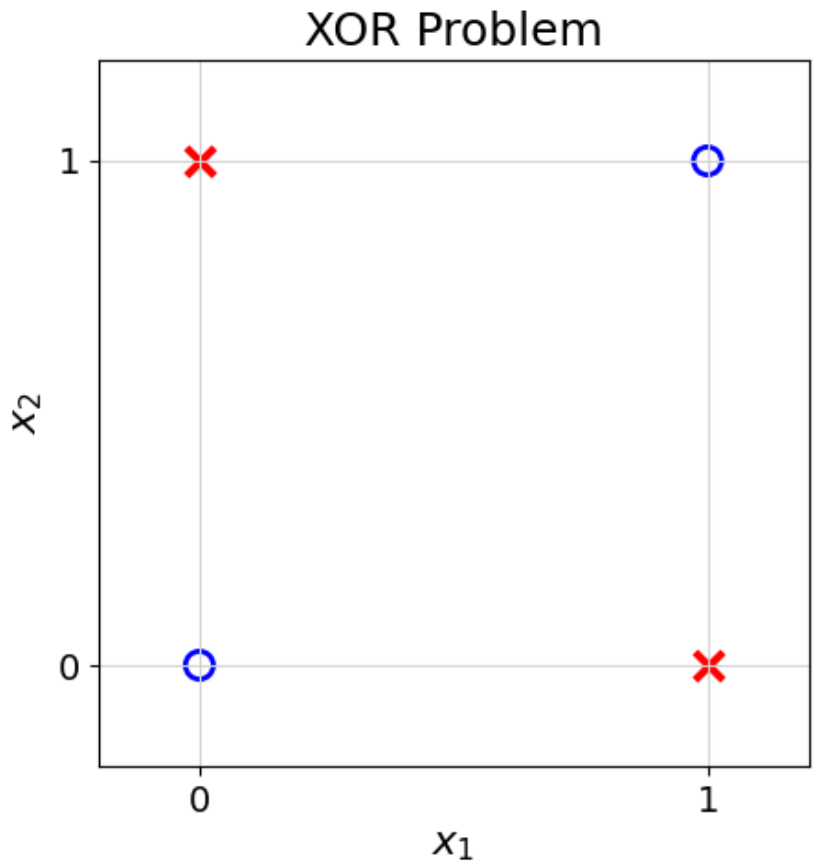


Figure 11.4: XOR Problem: Can you find a single line separating the x from the o?

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize=(5, 5))
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xlim(-0.2, 1.2)
ax.set_ylim(-0.2, 1.2)
ax.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)

# XOR points
```

```
points = np.array([[0,0],[0,1],[1,0],[1,1]])
labels = [0,1,1,0]
for pt, lbl in zip(points, labels):
    if lbl == 0:
        ax.scatter(pt[0], pt[1], marker='o', color='blue', s=120, edgecolor='blue', facecolors='n
    else:
        ax.scatter(pt[0], pt[1], marker='x', color='red', s=120, linewidths=3)
ax.set_xlabel('$x_1$', fontsize=16)
ax.set_ylabel('$x_2$', fontsize=16)
ax.set_title('XOR Problem', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig("images/trees/xor_problem.png")
plt.show()
```

The XOR is a logical operation that takes two Boolean (`True`/`False` or 1/0) values as inputs and returns 1 (or `True`) if they are different and 0 or `False` otherwise. You can see the truth table for this operation below.

XOR Truth Table

Input A	Input B	Output (A XOR B)
False	False	False
False	True	True
True	False	True
True	True	False

More information on the XOR problem at ("Exclusive Or," n.d.)

Instead of splitting the data once, what if you could split the data **multiple times**? An example is shown below:

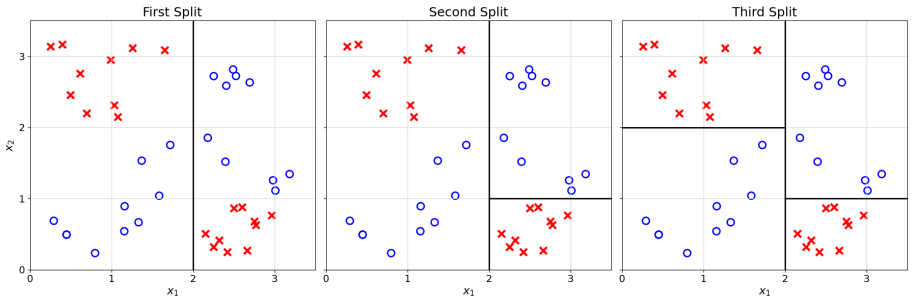


Figure 11.5: Splitting the data multiple times

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(42)
x1_region1 = np.random.uniform(0.2, 1.8, 10)
x2_region1 = np.random.uniform(0.2, 1.8, 10)
x1_region2 = np.random.uniform(2.1, 3.2, 10)
x2_region2 = np.random.uniform(0.2, 0.9, 10)
x1_region3 = np.random.uniform(0.2, 1.8, 10)
x2_region3 = np.random.uniform(2.1, 3.2, 10)
x1_region4 = np.random.uniform(2.1, 3.2, 10)
x2_region4 = np.random.uniform(1.1, 3.2, 10)

fig, axs = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
splits = [
    {'lines': [([2, 2], [0, 3.5])]},
    {'lines': [([2, 2], [0, 3.5]), ([2, 3.5], [1, 1])]},
    {'lines': [([2, 2], [0, 3.5]), ([2, 3.5], [1, 1]), ([0, 2], [2, 2])]}
]
titles = ['First Split', 'Second Split', 'Third Split']

for i, ax in enumerate(axs):
    ax.set_xticks(np.arange(0, 4, 1))
    ax.set_yticks(np.arange(0, 4, 1))
    ax.set_xlim(0, 3.5)
    ax.set_ylim(0, 3.5)
    ax.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)
    ax.scatter(x1_region1, x2_region1, marker='o', color='blue', s=100, facecolors='no
    ax.scatter(x1_region2, x2_region2, marker='x', color='red', s=100, linewidths=3)
    ax.scatter(x1_region3, x2_region3, marker='x', color='red', s=100, linewidths=3)
    ax.scatter(x1_region4, x2_region4, marker='o', color='blue', s=100, facecolors='no
    for line in splits[i]['lines']:
        ax.plot(line[0], line[1], color='black', linestyle='-', linewidth=2)
    ax.set_title(titles[i], fontsize=18)
    ax.set_xlabel('$x_1$', fontsize=16)
    if i == 0:
        ax.set_ylabel('$x_2$', fontsize=16)
    ax.tick_params(axis='both', which='major', labelsize=14)
plt.tight_layout()
plt.savefig("images/trees/three_splits.png")
plt.show()
```

11.2 From Splits to Predictions

Using these three splits, how could you classify a **new observation**?

One way to do so would be to go through the splits we did one by one:

- If $x_1 \geq 2$:
  - If $x_2 \geq 1$: assign $\circ$
  - Else: assign $\times$
- Else:
  - If $x_2 \geq 2$: assign $\times$
  - Else: assign $\circ$

The observation is then assigned the label that is the **majority** in the resulting partition.

Exercise 11.1. Using the above list, generate predictions for the following observations:

- Observation 1: $x_1 = 3, x_2 = 3$
- Observation 2: $x_1 = 1, x_2 = 3$
- Observation 3: $x_1 = 1, x_2 = 1$

11.3 From Partition to Trees

The above list may be difficult to follow. This type of logic could be more elegantly represented as a **tree** (also called a **decision tree**):

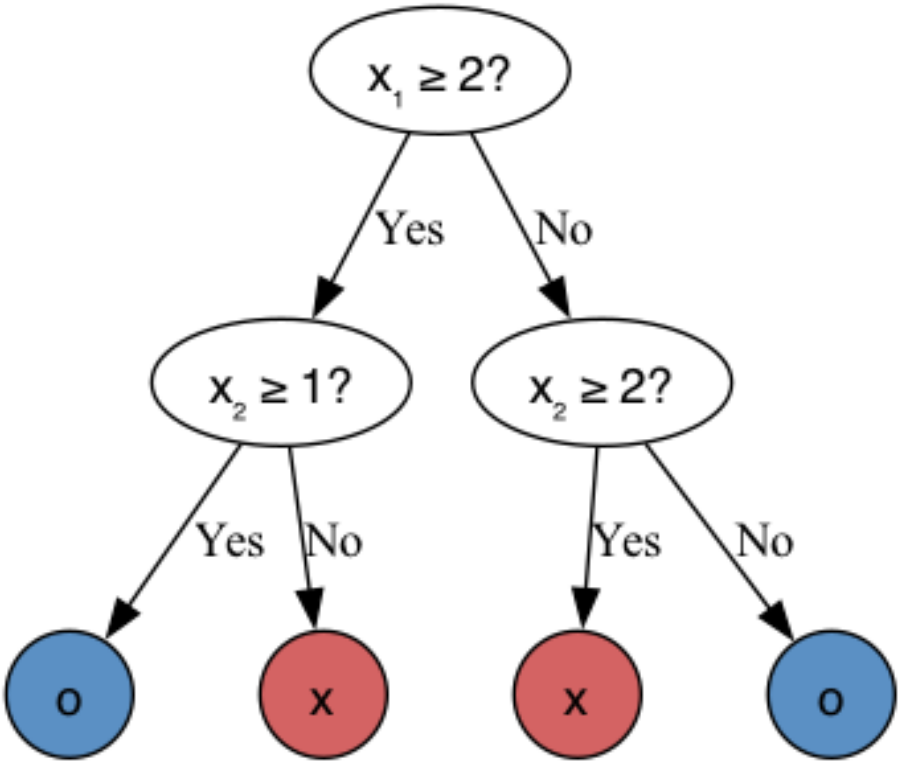


Figure 11.6: Visualising splits as a Decision Tree

Figure code

```
from graphviz import Digraph
dot = Digraph()
dot.attr(rankdir='TB', fontsize='16', fontname='Helvetica') # Top to bottom, Helvetica
# Decision nodes (ovals)
dot.attr('node', shape='ellipse', fontname='Helvetica')
dot.node('A', 'x   2?')
dot.node('B', 'x   1?')
dot.node('C', 'x   2?')
# Leaf nodes (circles)
dot.attr('node', shape='circle', style='filled', fillcolor="#D46363", fontname='Helveti
dot.node('E', 'x')
dot.node('F', 'x')
dot.attr('node', shape='circle', style='filled', fillcolor="#5B90C4", fontname='Helveti
dot.node('D', 'o')
dot.node('G', 'o')
```

```
with dot.subgraph() as s:
    s.attr(rank='same')
    s.node('D')
    s.node('E')
    s.node('F')
    s.node('G')
    s.edge('D', 'E', style='invis')
    s.edge('E', 'F', style='invis')
    s.edge('F', 'G', style='invis')

# Edges
dot.edge('A', 'B', 'Yes')
dot.edge('A', 'C', 'No')
dot.edge('B', 'D', 'Yes')
dot.edge('C', 'F', 'Yes')
dot.edge('B', 'E', 'No')
dot.edge('C', 'G', 'No')

# Render the graph
dot.render('images/trees/tree_example', format='png', cleanup=True)
```

Exercise 11.2. Generate predictions for the following observations by going down the Decision Tree above:

- Observation 1: $x_1 = 3, x_2 = 3$
- Observation 2: $x_1 = 1, x_2 = 3$
- Observation 3: $x_1 = 1, x_2 = 1$

To better understand the relationship between trees and data splits, the tree and the partition can be visualised next to one another:

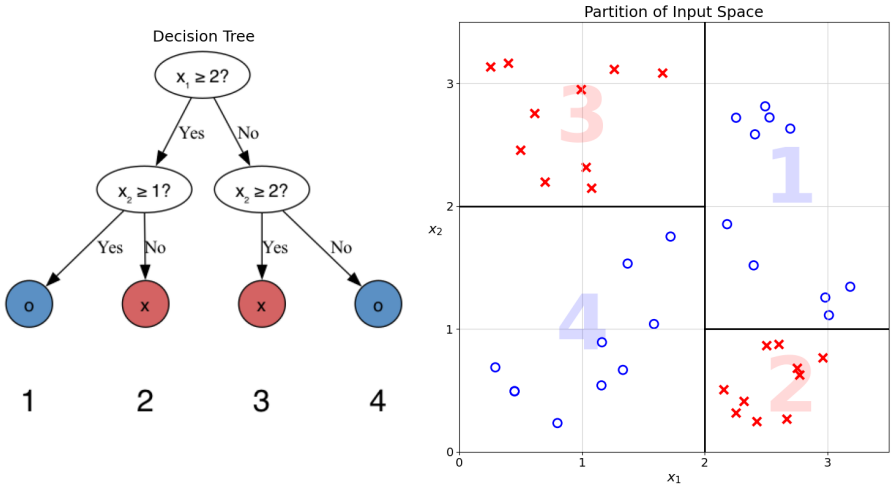


Figure 11.7: Decision Tree as a partition of space

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
from graphviz import Digraph

# 1. Create and render the Graphviz tree with leaf numbers BELOW the node
dot = Digraph()
dot.attr(rankdir='TB', fontsize='16', fontname='Helvetica')
dot.attr('node', shape='ellipse', fontname='Helvetica')
dot.node('A', 'x   2?')
dot.node('B', 'x   1?')
dot.node('C', 'x   2?')

# Leaves: label is just the class, number is a separate node below
dot.attr('node', shape='circle', style='filled', fillcolor="#D46363", fontname='Helveti
dot.node('E', 'x')
dot.node('F', 'x')
dot.attr('node', shape='circle', style='filled', fillcolor="#5B90C4", fontname='Helveti
dot.node('D', 'o')
dot.node('G', 'o')

# Add invisible nodes for numbers below each leaf
dot.attr('node', shape='plaintext', fontname='Helvetica', fontsize='24', fillcolor='whi
dot.node('E_num', '2')
dot.node('F_num', '3')
```

```
dot.node('D_num', '1')
dot.node('G_num', '4')

# Position numbers below leaves using invisible edges
dot.edge('E', 'E_num', style='invis', weight='100')
dot.edge('F', 'F_num', style='invis', weight='100')
dot.edge('D', 'D_num', style='invis', weight='100')
dot.edge('G', 'G_num', style='invis', weight='100')

with dot.subgraph() as s:
    s.attr(rank='same')
    s.node('D')
    s.node('E')
    s.node('F')
    s.node('G')
    s.edge('D', 'E', style='invis')
    s.edge('E', 'F', style='invis')
    s.edge('F', 'G', style='invis')

with dot.subgraph() as s:
    s.attr(rank='same')
    s.node('D_num')
    s.node('E_num')
    s.node('F_num')
    s.node('G_num')
    s.edge('D_num', 'E_num', style='invis')
    s.edge('E_num', 'F_num', style='invis')
    s.edge('F_num', 'G_num', style='invis')

dot.edge('A', 'B', 'Yes')
dot.edge('A', 'C', 'No')
dot.edge('B', 'D', 'Yes')
dot.edge('B', 'E', 'No')
dot.edge('C', 'F', 'Yes')
dot.edge('C', 'G', 'No')

tree_img_name = 'images/trees/tree_numbered'
dot.render(tree_img_name, format='png', cleanup=True)

fig, axes = plt.subplots(1, 2, figsize=(15, 8))

# --- Tree plot ---
ax0 = axes[0]
ax0.axis('off')
tree_img = Image.open(tree_img_name + ".png")
```

```
ax0.imshow(tree_img)
ax0.set_title('Decision Tree', fontsize=18)

# --- Partition plot ---
ax1 = axes[1]
ax1.set_xticks(np.arange(0, 4, 1))
ax1.set_yticks(np.arange(0, 4, 1))
ax1.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)
ax1.set_xlim(0, 3.5)
ax1.set_ylim(0, 3.5)
ax1.set_xlabel('$x_1$', fontsize=16)
ax1.set_ylabel('$x_2$', rotation=0, ha='right', fontsize=16)
ax1.tick_params(axis='both', which='major', labelsize=14)

np.random.seed(42)
x1_region1 = np.random.uniform(0.2, 1.8, 10)
x2_region1 = np.random.uniform(0.2, 1.8, 10)
ax1.scatter(x1_region1, x2_region1, marker='o', color='blue', s=100, facecolors='none'

x1_region2 = np.random.uniform(2.1, 3.2, 10)
x2_region2 = np.random.uniform(0.2, 0.9, 10)
ax1.scatter(x1_region2, x2_region2, marker='x', color='red', s=100, linewidths=3, label

x1_region3 = np.random.uniform(0.2, 1.8, 10)
x2_region3 = np.random.uniform(2.1, 3.2, 10)
ax1.scatter(x1_region3, x2_region3, marker='x', color='red', s=100, linewidths=3)

x1_region4 = np.random.uniform(2.1, 3.2, 10)
x2_region4 = np.random.uniform(1.1, 3.2, 10)
ax1.scatter(x1_region4, x2_region4, marker='o', color='blue', s=100, facecolors='none'

# Partition lines
ax1.plot([2, 2], [0, 3.5], color='black', linestyle='-', linewidth=2)
ax1.plot([0, 2], [2, 2], color='black', linestyle='-', linewidth=2)
ax1.plot([2, 3.5], [1, 1], color='black', linestyle='-', linewidth=2)

ax1.set_title('Partition of Input Space', fontsize=18)
ax1.set_aspect('equal', adjustable='box')

# Overlay region numbers (matching tree leaves)
ax1.text(1, 2.7, '3', fontsize=90, color='red', alpha=0.15, ha='center', va='center', v
ax1.text(1, 1, '4', fontsize=90, color='blue', alpha=0.15, ha='center', va='center', we
ax1.text(2.7, 0.5, '2', fontsize=90, color='red', alpha=0.15, ha='center', va='center'
ax1.text(2.7, 2.2, '1', fontsize=90, color='blue', alpha=0.15, ha='center', va='center
```

```
plt.tight_layout()
plt.savefig("images/trees/tree_numbered_partition.png")
plt.show()
```

Each split or comparison creates a **linear separation** in the feature space, represented by a black line in the chart above.

In Computer Science, trees are a type of graph. Graphs are collections of **nodes and edges**.

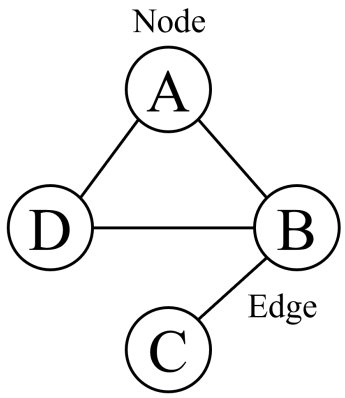


Figure 11.8: Graphs are collections of nodes and edges connecting them

Edges can be **directed** or **undirected**. Directed edges can only be traversed in one direction, usually represented as arrows. Edges, i.e., connections between nodes, can form **cycles**. These cycles are paths that start and end with the same node.

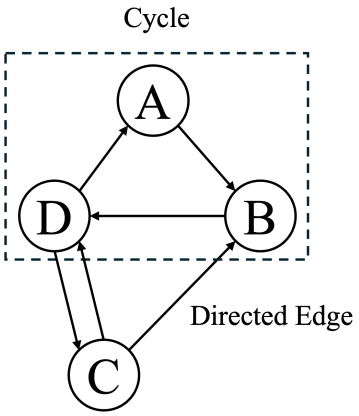


Figure 11.9: Edges can be directed

Exercise 11.3. The diagram above highlights the cycle A→B→D→A. Can you find another cycle?

Trees are **Directed Acyclic Graphs** (DAGs). This is a bit of a mouthful, let's take these terms one by one:

- **Directed**: They flow from the root to the leaves
- **Acyclic**: They do not contain any cycles; there is only a single path from the root to each node

Below is an example of a tree with seven nodes:

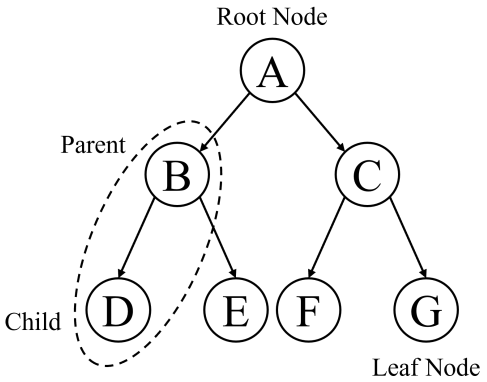


Figure 11.10: A simple tree with seven nodes

Trees, like all graphs, are composed of nodes and edges. In addition to the usual graph jargon, there are a few tree-specific concepts. To understand them,

consider the example above:

- **Parent and child nodes**: both connected by an edge (nodes B and D)
- **Root node**: The topmost node of a tree (node A)
- **Leaf node**: A node with no children (node G)

Exercise 11.4. Find another example of:

- Leaf node
- Parent node

11.4 From Trees to Maps

Just like the KNN model, Decision Trees build a **map** from the input features to the target variable. Using a Decision Tree to generate predictions for all possible feature values, we get the following map:

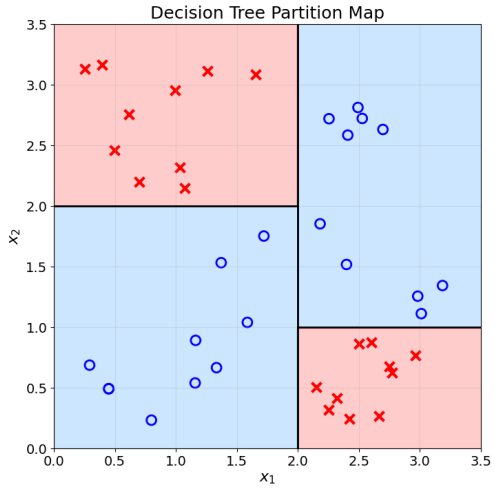


Figure 11.11: Building a map with Decision Trees

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Generate meshgrid
xx, yy = np.meshgrid(np.linspace(0, 3.5, 200), np.linspace(0, 3.5, 200))
Z = np.zeros_like(xx, dtype=int)
```

```
# Apply the tree rules
for i in range(xx.shape[0]):
    for j in range(xx.shape[1]):
        x1, x2 = xx[i, j], yy[i, j]
        if x1 >= 2:
            if x2 >= 1:
                Z[i, j] = 0  # circle
            else:
                Z[i, j] = 1  # x
        else:
            if x2 >= 2:
                Z[i, j] = 1  # x
            else:
                Z[i, j] = 0  # circle

cmap = ListedColormap(['#80C2FF', '#FF8080'])
plt.figure(figsize=(8, 8))
plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.4)

# Overlay the data points
np.random.seed(42)
x1_region1 = np.random.uniform(0.2, 1.8, 10)
x2_region1 = np.random.uniform(0.2, 1.8, 10)
x1_region2 = np.random.uniform(2.1, 3.2, 10)
x2_region2 = np.random.uniform(0.2, 0.9, 10)
x1_region3 = np.random.uniform(0.2, 1.8, 10)
x2_region3 = np.random.uniform(2.1, 3.2, 10)
x1_region4 = np.random.uniform(2.1, 3.2, 10)
x2_region4 = np.random.uniform(1.1, 3.2, 10)

plt.scatter(x1_region1, x2_region1, marker='o', color='blue', s=100, facecolors='none'
plt.scatter(x1_region2, x2_region2, marker='x', color='red', s=100, linewidths=3)
plt.scatter(x1_region3, x2_region3, marker='x', color='red', s=100, linewidths=3)
plt.scatter(x1_region4, x2_region4, marker='o', color='blue', s=100, facecolors='none'

plt.plot([2, 2], [0, 3.5], color='black', linestyle='-', linewidth=2)
plt.plot([0, 2], [2, 2], color='black', linestyle='-', linewidth=2)
plt.plot([2, 3.5], [1, 1], color='black', linestyle='-', linewidth=2)

plt.xlabel('$x_1$', fontsize=16)
plt.ylabel('$x_2$', fontsize=16)
plt.title('Decision Tree Partition Map', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, alpha=0.3)
```

```
plt.savefig("images/trees/decision_tree_partition_map.png")
plt.show()
```

11.5 Final Thoughts

That is it, we have built our first Decision Tree! We can use this model to predict any **new observation** based on its two **features** and position on the map above.

This is only a simple introduction to Decision Trees. The next chapters will explore the inner workings of this model family, starting with the evaluation of data splits. What makes a good split of a dataset?

11.6 Solutions

Solution 11.1. Exercise 11.1

- Observation 1: ∘
- Observation 2: ×
- Observation 3: ∘

Solution 11.2. Exercise 11.2

- Observation 1: ∘
- Observation 2: ×
- Observation 3: ∘

Solution 11.3. Exercise 11.3 There are many possible cycles, here are two examples:

- C→D→C
- C→B→D→C

Exercise 11.5.

Solution. Exercise 11.5

- Leaf node: D, E, F, G
- Parent node: A, B, C

Chapter 12

Evaluating Splits

In most problems, the different groups cannot be **separated**. Going back to the tumour diagnosis example, the training data looks messy:

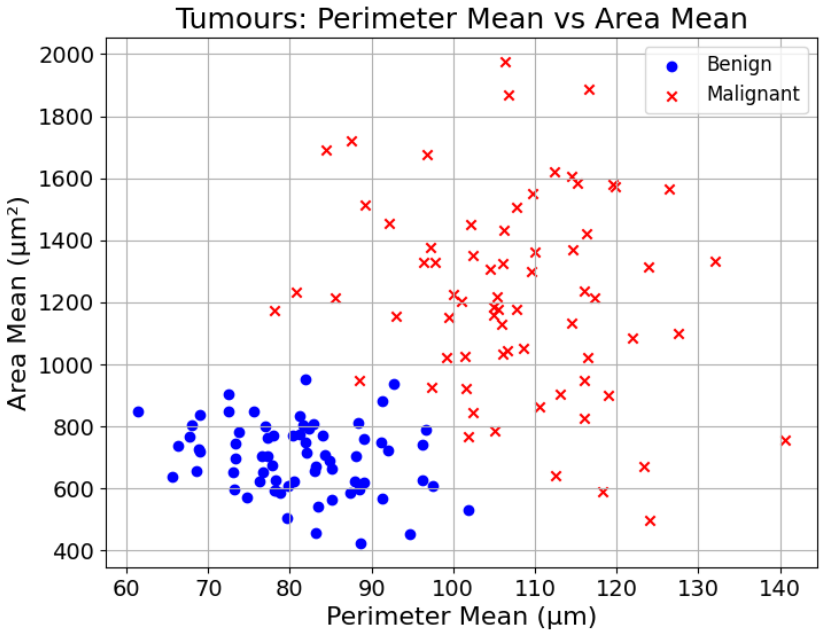


Figure 12.1: Tumour diagnosis classification problem

How could we build a tree from there?

12.1 Best Split First

We could start by finding the **best split**, the split that separates the data best. The split will create two groups:

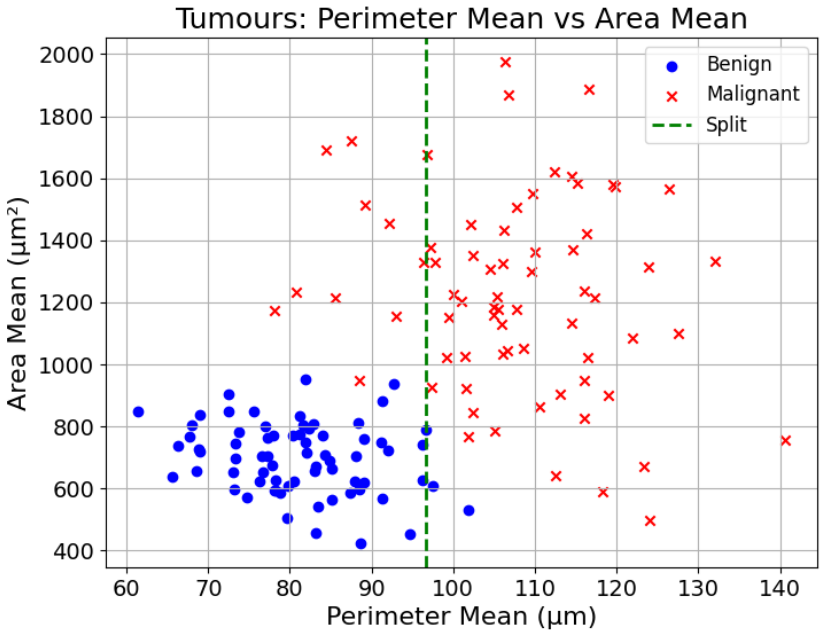


Figure 12.2: First split

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.tree import DecisionTreeClassifier

benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando

benign_std = [10, 120]
malignant_std = [12, 300]
```

```
X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

# Stack data and create labels
X = np.vstack([X_benign, X_malignant])
y = np.array([0]*n_samples + [1]*n_samples)  # 0: Benign, 1: Malignant

# Fit a decision tree with depth=1 (stump)
tree = DecisionTreeClassifier(max_depth=1, random_state=0)
tree.fit(X, y)

# Get split info
feature_names = ['Perimeter Mean (µm)', 'Area Mean (µm²)']
split_feature = tree.tree_.feature[0]
split_threshold = tree.tree_.threshold[0]
print(f"First split: {feature_names[split_feature]} < {split_threshold:.2f}")

# Plot
plt.figure(figsize=(8,6))
plt.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
plt.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malignant')

# Plot the split
if split_feature == 0:
    plt.axvline(split_threshold, color='green', linestyle='--', linewidth=2, label=f'Split')
elif split_feature == 1:
    plt.axhline(split_threshold, color='green', linestyle='--', linewidth=2, label=f'Split')

plt.xlabel('Perimeter Mean (µm)', fontsize=16)
plt.ylabel('Area Mean (µm²)', fontsize=16)
plt.title('Tumours: Perimeter Mean vs Area Mean', fontsize=18)
plt.legend(fontsize=12)
plt.grid(True)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig("images/trees/example_first_split.png")
plt.show()
```

For each of the new groups, we can find the **best splits**, apply them, and **create two new groups**:

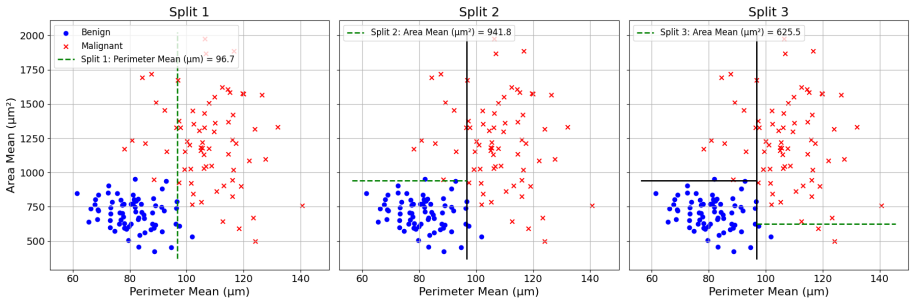


Figure 12.3: All splits

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.tree import DecisionTreeClassifier

benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando

benign_std = [10, 120]
malignant_std = [12, 300]

X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

# Stack data and create labels
X = np.vstack([X_benign, X_malignant])
y = np.array([0]*n_samples + [1]*n_samples)  # 0: Benign, 1: Malignant

# Fit a decision tree (no max_depth limit)
tree = DecisionTreeClassifier(random_state=0,max_depth=2)
tree.fit(X, y)

feature_names = ['Perimeter Mean (µm)', 'Area Mean (µm²)']

# Helper function to recursively collect all splits
def collect_splits(tree, node_id=0, path=[], splits=[]):
    feature = tree.tree_.feature[node_id]
```

```
    threshold = tree.tree_.threshold[node_id]
    if feature >= 0:
        # Save the split with its path (list of (feature, threshold, direction))
        splits.append((path.copy(), feature, threshold))
        # Left child: feature <= threshold
        collect_splits(tree, tree.tree_.children_left[node_id], path + [(feature, threshold, 'lef
        # Right child: feature > threshold
        collect_splits(tree, tree.tree_.children_right[node_id], path + [(feature, threshold, 'ri
    return splits

splits = collect_splits(tree, 0, [], [])

# For plotting, get the data limits
x0min, x0max = X[:,0].min() - 5, X[:,0].max() + 5
x1min, x1max = X[:,1].min() - 50, X[:,1].max() + 50

n_splits = len(splits)
fig, axes = plt.subplots(1, n_splits, figsize=(6*n_splits, 6), sharex=True, sharey=True)

if n_splits == 1:
    axes = [axes]  # Make iterable

for i in range(n_splits):
    ax = axes[i]
    ax.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign' if i==0 els
    ax.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Malignant' if

    # Draw all previous splits as black lines, in their respective regions
    for j in range(i):
        path, feat, thresh = splits[j]
        xlims = [x0min, x0max]
        ylims = [x1min, x1max]
        for (pf, pt, dirn) in path:
            if pf == 0:
                if dirn == 'left':
                    xlims[1] = min(xlims[1], pt)
                else:
                    xlims[0] = max(xlims[0], pt)
            else:
                if dirn == 'left':
                    ylims[1] = min(ylims[1], pt)
                else:
                    ylims[0] = max(ylims[0], pt)
        if feat == 0:
            ax.plot([thresh, thresh], ylims, color='black', linewidth=2)
```

```
        else:
            ax.plot(xlims, [thresh, thresh], color='black', linewidth=2)

    # Draw current split as green dashed line, in its region
    path, feat, thresh = splits[i]
    xlims = [x0min, x0max]
    ylims = [x1min, x1max]
    for (pf, pt, dirn) in path:
        if pf == 0:
            if dirn == 'left':
                xlims[1] = min(xlims[1], pt)
            else:
                xlims[0] = max(xlims[0], pt)
        else:
            if dirn == 'left':
                ylims[1] = min(ylims[1], pt)
            else:
                ylims[0] = max(ylims[0], pt)
    if feat == 0:
        ax.plot([thresh, thresh], ylims, color='green', linestyle='--', linewidth=2,
                label=f'Split {i+1}: {feature_names[feat]} = {thresh:.1f}')
    else:
        ax.plot(xlims, [thresh, thresh], color='green', linestyle='--', linewidth=2,
                label=f'Split {i+1}: {feature_names[feat]} = {thresh:.1f}')

    ax.set_xlabel('Perimeter Mean (µm)', fontsize=16)
    if i == 0:
        ax.set_ylabel('Area Mean (µm²)', fontsize=16)
    ax.set_title(f'Split {i+1}', fontsize=18)
    ax.grid(True)
    ax.tick_params(axis='both', which='major', labelsize=14)
    ax.legend(fontsize=12, loc='upper left')

plt.tight_layout()
plt.savefig("images/trees/example_three_splits.png")
plt.show()
```

We can keep splitting until the subgroups are "pure"; i.e., there are only observations belonging to the same class.

In practice, other constraints are applied. In general it makes sense to limit the **depth** of a tree to make the size of the resulting model manageable. These simple models generally have stronger generalisation abilities.

What is tree depth?

The depth of a tree is the **distance** from the root node, here the first split, to

the leaf node, here the last node, responsible for assigning the prediction.

What is the depth of the tree built in the previous examples?

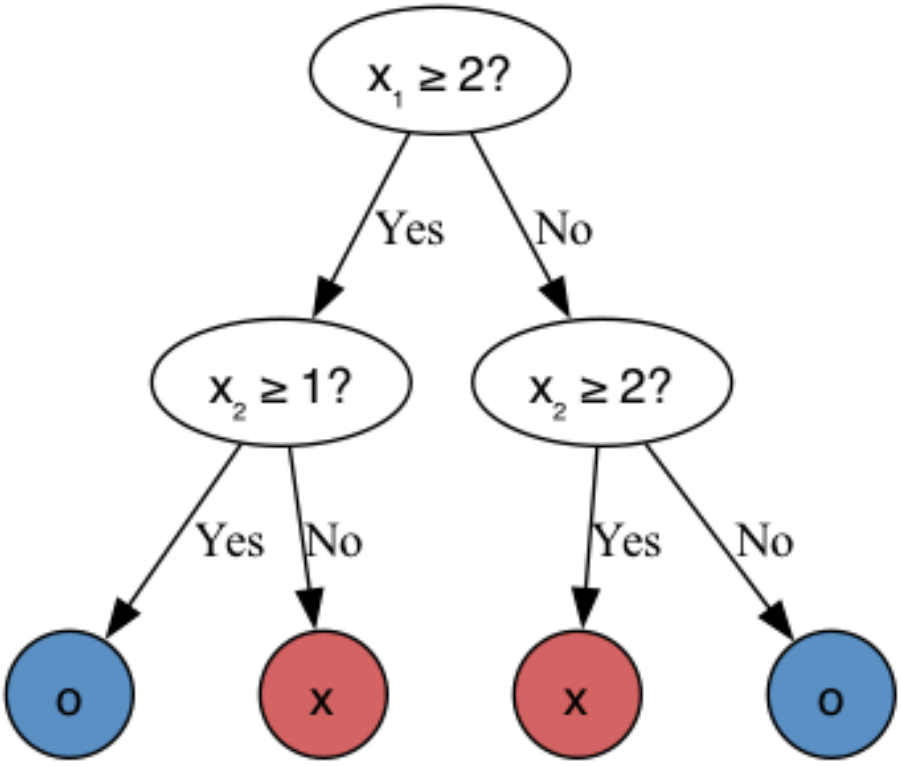


Figure 12.4: Example tree

It is two, as there are two edges (connections) on the path from the root node (top) to the leaf node (bottom).

12.2 Gini Impurity Coefficient

How would you measure the **quality of a split**? Or the purity of the resulting groups?

Imagine that Split 1 creates the following two groups:

Group	Malignant	Benign
A	9	1
B	2	8

And that Split 2 creates the following two groups:

Group	Malignant	Benign
C	6	4
D	5	5

Intuitively, we know that Split 1 has done better, as the resulting subgroups are **more homogeneous**. Group A contains mostly malignant observations and Group B more benign ones. On the other hand, Group C and D are both mixed. But how to quantify this intuition?

This is exactly what the **Gini Impurity Coefficient** measures.

The Gini Coefficient measures "impurity" through the probability of randomly picking **two items of different classes** within the same group.

But why would this probability measure impurity? If this probability is low, it means that we are not likely to pick items of different classes. In other words, we are likely to pick items from the **same class**. This probability will be 0 if all the items of a group are of a single class.

On the other hand, when this probability is high, we are likely to pick items of a different class. This will happen when the group is **mixed**. It will be 1 in the extreme case in which the group contains only a single item from each class.

Let's illustrate this with Group A in the table below:

Group	Malignant	Benign
A	9	1

If we randomly picked two items in this group (with replacement) what would be the probability of picking items **from two different classes** (malignant and benign)?

Try thinking about it before reading on.

To make this simpler, what is the probability to pick first a malignant tumour, then a benign tumour?

The probability of picking a malignant and benign tumour are the following:

$$P(\text{malignant}) = \frac{9}{9+1} = 0.9$$

$$P(\text{benign}) = \frac{1}{9+1} = 0.1$$

As the two picks are random and with replacement, the probability of picking one, then the other is:

$$P(\text{malignant then benign}) = P(\text{malignant}) \cdot P(\text{benign}) = 0.9 \cdot 0.1 = 0.09$$

Now, what would be the probability of picking a benign observation then a malignant one (reversing the order)? We have already computed the probabilities above, we just need to reverse the order:

$$P(\text{benign then malignant}) = P(\text{benign}) \cdot P(\text{malignant}) = 0.1 \cdot 0.9 = 0.09$$

Which, **by symmetry**, gives the same results. As this sampling is independent, the probability of picking malignant then benign is the same as benign then malignant. To get the probability of these two events we just have to sum them:

$$\begin{aligned} P(\text{picking items of different class}) &= P(\text{malignant then benign}) + P(\text{benign then malignant}) \\ &= 0.09 + 0.09 = 0.18 \end{aligned}$$

That is it, we have calculated the Gini Impurity Coefficient for Group A.

Exercise 12.1. Compute the Gini Impurity Coefficient for Group B and show that it is 0.32.

Group	Malignant	Benign
B	2	8

The Gini Impurity Coefficient of Group B is slightly higher than Group A, as it is slightly more mixed.

12.2.1 Evaluating Splits

Now, let's get back to quantifying the quality of a split. To do so, we will compute the average Gini Coefficient of each split.

Split 1:

Group	Malignant	Benign
A	9	1
B	2	8

Split 2:

Group	Malignant	Benign
C	6	4
D	5	5

As covered in the previous section, we know the Gini Coefficient of both Group A and B: 0.18 and 0.32 respectively.

The average Gini Coefficient for Split 1, Group A and B, can be computed as follows:

$$\frac{n_a}{n_a + n_b} \cdot \text{Gini}_a + \frac{n_b}{n_a + n_b} \cdot \text{Gini}_b$$

$$\frac{10}{10 + 10} \cdot 0.18 + \frac{10}{10 + 10} \cdot 0.32 = 0.25$$

Exercise 12.2. Compute the Gini Coefficients for both groups (C and D) in Split 2, and prove that they are equal to: 0.48 and 0.5 respectively.

Group	Malignant	Benign
C	6	4
D	5	5

Now, computing the average Gini Impurity Coefficient for Split 2, we get:

$$\frac{n_c}{n_c + n_d} \cdot \text{Gini}_c + \frac{n_d}{n_c + n_d} \cdot \text{Gini}_d$$

We get:

$$\frac{10}{10 + 10} \cdot 0.48 + \frac{10}{10 + 10} \cdot 0.5 = 0.49$$

Concluding this section, Split 1 has an average Gini Impurity Coefficient of 0.25 compared to 0.49 for Split 2. It is the split that separates the data best.

Now that we have a way to measure the quality of a split, we can simply try splitting the data at **all feature values** for **all features**, and pick the best one.

12.3 Trying Different Splits

To find the best data split, the Decision Tree learning algorithm evaluates **each possible splitting feature and value**, and picks the one that has the lowest Gini Impurity Coefficient.

The example above only shows two different splits. This process can be repeated for all features and all splitting values. At each trial, the Gini Impurity Coefficient is recorded. The algorithm then selects the split achieving the lowest Gini Impurity Coefficient. This can be represented visually:

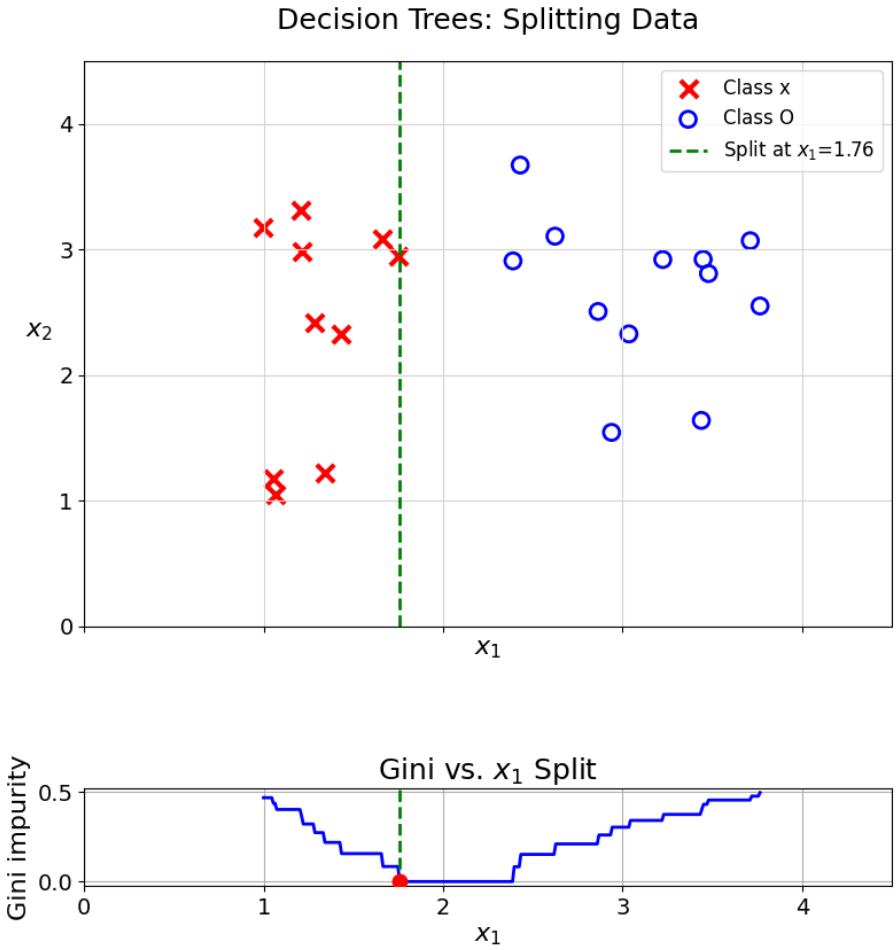


Figure 12.5: Trying all combinations and picking the first split achieving the minimum Gini Impurity Coefficient

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

# --- Data generation ---
```

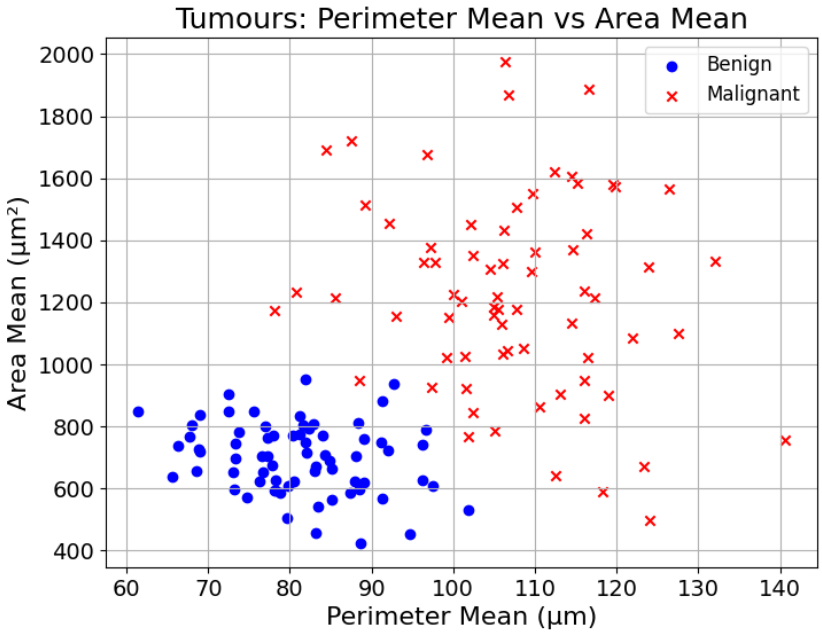
```
np.random.seed(0)
x1_plus = np.random.uniform(0.5, 1.8, 10)
x2_plus = np.random.uniform(1.0, 3.5, 10)
x1_circle = np.random.uniform(2.2, 3.8, 12)
x2_circle = np.random.uniform(1.5, 3.8, 12)

X_plus = np.column_stack([x1_plus, x2_plus])
X_circle = np.column_stack([x1_circle, x2_circle])
X = np.vstack([X_plus, X_circle])
y = np.array([1]*len(X_plus) + [0]*len(X_circle))  # 1: x, 0: o

# --- Gini impurity for a split ---
def gini_impurity(y_left, y_right):
    def gini(y):
        if len(y) == 0:
            return 0
        p = np.mean(y)
        return 2 * p * (1 - p)
    n = len(y_left) + len(y_right)
    return (len(y_left) * gini(y_left) + len(y_right) * gini(y_right)) / n

# --- Gini vs x1 threshold ---
fidx = 0  # x1
fmin, fmax = X[:, fidx].min(), X[:, fidx].max()
thresholds = np.linspace(fmin, fmax, 300)
ginis = []
for t in thresholds:
    left = y[X[:, fidx] <= t]
    right = y[X[:, fidx] > t]
    gini = gini_impurity(left, right)
    ginis.append(gini)
ginis = np.array(ginis)
min_idx = np.argmin(ginis)
min_thresh = thresholds[min_idx]
min_gini = ginis[min_idx]

# --- Plot ---
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 10), sharex=True)

# Top: Data scatter
ax1.set_xticks(np.arange(0, 5, 1))
ax1.set_yticks(np.arange(0, 5, 1))
ax1.grid(True, linestyle='-', color='lightgrey', linewidth=0.8)
ax1.set_xlim(0, 4.5)
ax1.set_ylim(0, 4.5)
```

```
ax1.set_xlabel('$x_1$', fontsize=16)
ax1.set_ylabel('$x_2$', rotation=0, ha='right', fontsize=16)
ax1.tick_params(axis='both', which='major', labelsize=14)

ax1.scatter(x1_plus, x2_plus, marker='x', color='red', s=120, linewidths=3, label='Clas
ax1.scatter(x1_circle, x2_circle, marker='o', color='blue', s=100, facecolors='none', e

# Green split line at best threshold
ax1.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at $x_1$={mi

ax1.set_title('Decision Trees: Splitting Data', fontsize=18, pad=20)
plt.gca().set_aspect('equal', adjustable='box')

# To avoid duplicate legend
handles, labels = ax1.get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax1.legend(by_label.values(), by_label.keys(), fontsize=12)

# Bottom: Gini vs x1
ax2.plot(thresholds, ginis, color='blue', lw=2)
ax2.scatter([min_thresh], [min_gini], color='red', s=80, zorder=5, label='Minimum Gini
ax2.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at $x_1$={mi
ax2.set_xlabel('$x_1$', fontsize=16)
ax2.set_ylabel('Gini impurity', fontsize=16)
ax2.set_title('Gini vs. $x_1$ Split', fontsize=18)
ax2.grid(True)
ax2.tick_params(axis='both', which='major', labelsize=14)


plt.tight_layout()
plt.savefig("images/trees/split_and_gini_vs_x1.png")
plt.show()
```

From the value 1.76, the split **perfectly separates** the two data classes, achieving a Gini Impurity Coefficient of 0.

But what happens when data is **not** fully separable? For these cases, the Gini Impurity Coefficient will not reach 0. Still, the split that achieves the lowers Gini Impurity Coefficient will be selected.

This splitting trial and error can be visualised with the tumour diagnosis dataset:



We first try splitting the data using the Perimeter Mean of the cell nuclei. The following line chart plots the Gini Impurity criterion for **different splitting values** over this feature.

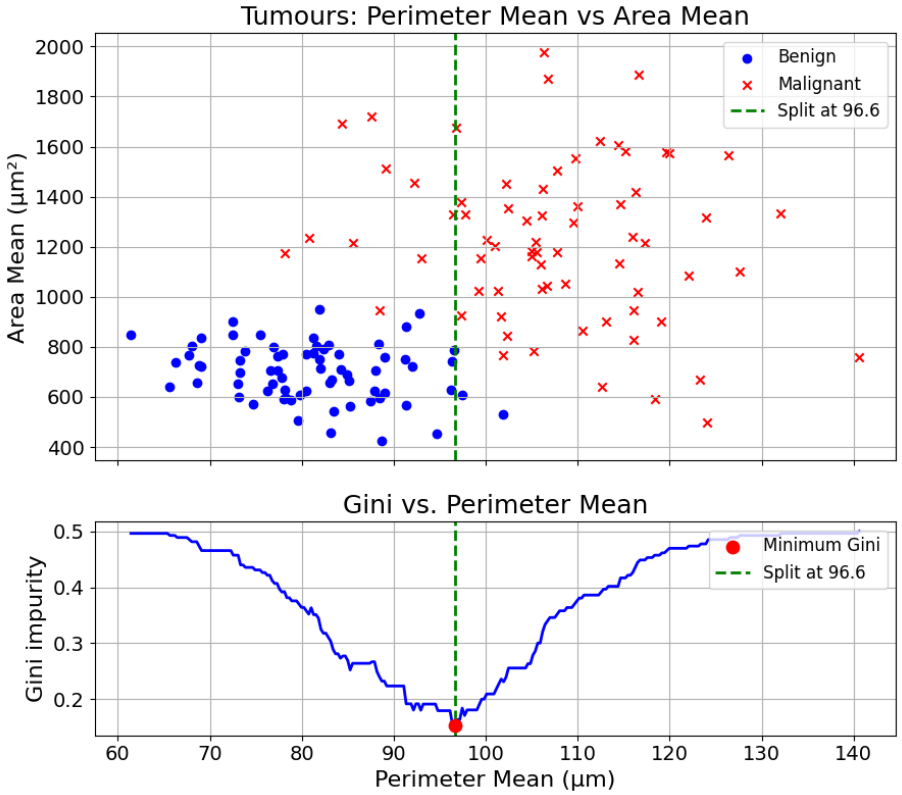


Figure 12.6: Trying different split values for Perimeter Mean

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.tree import DecisionTreeClassifier

# --- Data generation ---
benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_s
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, rando

benign_std = [10, 120]
malignant_std = [12, 300]
```

```
X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

X = np.vstack([X_benign, X_malignant])
y = np.array([0]*n_samples + [1]*n_samples)  # 0: Benign, 1: Malignant

feature_names = ['Perimeter Mean (µm)', 'Area Mean (µm²)']

# --- Fit tree ---
tree = DecisionTreeClassifier(random_state=0, max_depth=2)
tree.fit(X, y)

# --- Gini impurity for a split ---
def gini_impurity(y_left, y_right):
    def gini(y):
        if len(y) == 0:
            return 0
        p = np.mean(y)
        return 2 * p * (1 - p)
    n = len(y_left) + len(y_right)
    return (len(y_left) * gini(y_left) + len(y_right) * gini(y_right)) / n

# --- Gini vs Perimeter Mean ---
fidx = 0  # Perimeter Mean
fmin, fmax = X[:, fidx].min(), X[:, fidx].max()
thresholds = np.linspace(fmin, fmax, 300)
ginis = []
for t in thresholds:
    left = y[X[:, fidx] <= t]
    right = y[X[:, fidx] > t]
    gini = gini_impurity(left, right)
    ginis.append(gini)
ginis = np.array(ginis)
min_idx = np.argmin(ginis)
min_thresh = thresholds[min_idx]
min_gini = ginis[min_idx]

# Actual split(s) used by the tree for Perimeter Mean
splits = []
tree_ = tree.tree_
for node in range(tree_.node_count):
    if tree_.feature[node] == fidx:
        splits.append(tree_.threshold[node])

# --- Plot ---
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(9, 8), sharex=True, gridspec_kw={'height_

# Top: Data scatter
ax1.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
ax1.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Maligna
ax1.set_ylabel('Area Mean (µm²)', fontsize=16)
ax1.set_title('Tumours: Perimeter Mean vs Area Mean', fontsize=18)
ax1.legend(fontsize=12)
ax1.grid(True)
ax1.tick_params(axis='both', which='major', labelsize=14)
# Green split line
ax1.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at {min_thre
# To avoid duplicate legend
handles, labels = ax1.get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax1.legend(by_label.values(), by_label.keys(), fontsize=12)

# Bottom: Gini vs Perimeter Mean
ax2.plot(thresholds, ginis, color='blue', lw=2)
ax2.scatter([min_thresh], [min_gini], color='red', s=80, zorder=5, label='Minimum Gini
ax2.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at {min_thre
ax2.set_xlabel('Perimeter Mean (µm)', fontsize=16)
ax2.set_ylabel('Gini impurity', fontsize=16)
ax2.set_title('Gini vs. Perimeter Mean', fontsize=18)
ax2.grid(True)
ax2.tick_params(axis='both', which='major', labelsize=14)
# To avoid duplicate legend
handles, labels = ax2.get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax2.legend(by_label.values(), by_label.keys(), fontsize=12, loc='upper right')

plt.tight_layout(h_pad=2)
plt.savefig("images/trees/scatter_and_gini_vs_perimeter.png")
plt.show()
```

This Gini Impurity Coefficient reaches a minimum of approximately 15.2 at a split value of about 96.6.

The same could be done with the Mean Area of the cell nuclei:

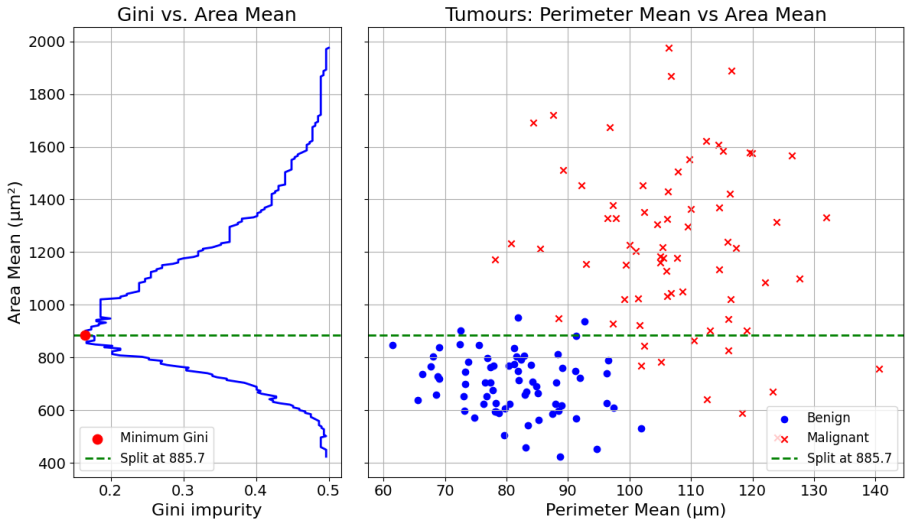


Figure 12.7: Trying different split values for Perimeter Mean

Figure code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.tree import DecisionTreeClassifier

# --- Data generation ---
benign_center = [80, 700]
malignant_center = [110, 1200]
n_samples = 70

X_benign, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=1)
X_malignant, _ = make_blobs(n_samples=n_samples, centers=[(0, 0)], cluster_std=1, random_state=2)

benign_std = [10, 120]
malignant_std = [12, 300]

X_benign = X_benign * benign_std + benign_center
X_malignant = X_malignant * malignant_std + malignant_center

X = np.vstack([X_benign, X_malignant])
y = np.array([0]*n_samples + [1]*n_samples)  # 0: Benign, 1: Malignant

feature_names = ['Perimeter Mean (µm)', 'Area Mean (µm²)']
```

```
# --- Fit tree ---
tree = DecisionTreeClassifier(random_state=0, max_depth=2)
tree.fit(X, y)

# --- Gini impurity for a split ---
def gini_impurity(y_left, y_right):
    def gini(y):
        if len(y) == 0:
            return 0
        p = np.mean(y)
        return 2 * p * (1 - p)
    n = len(y_left) + len(y_right)
    return (len(y_left) * gini(y_left) + len(y_right) * gini(y_right)) / n

# --- Gini vs Perimeter Mean ---
fidx = 0  # Perimeter Mean
fmin, fmax = X[:, fidx].min(), X[:, fidx].max()
thresholds = np.linspace(fmin, fmax, 300)
ginis = []
for t in thresholds:
    left = y[X[:, fidx] <= t]
    right = y[X[:, fidx] > t]
    gini = gini_impurity(left, right)
    ginis.append(gini)
ginis = np.array(ginis)
min_idx = np.argmin(ginis)
min_thresh = thresholds[min_idx]
min_gini = ginis[min_idx]

# Actual split(s) used by the tree for Perimeter Mean
splits = []
tree_ = tree.tree_
for node in range(tree_.node_count):
    if tree_.feature[node] == fidx:
        splits.append(tree_.threshold[node])

# --- Plot ---
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(9, 8), sharex=True, gridspec_kw={'height_

# Top: Data scatter
ax1.scatter(X_benign[:,0], X_benign[:,1], marker='o', color='blue', label='Benign')
ax1.scatter(X_malignant[:,0], X_malignant[:,1], marker='x', color='red', label='Maligna
ax1.set_ylabel('Area Mean (μm²)', fontsize=16)
ax1.set_title('Tumours: Perimeter Mean vs Area Mean', fontsize=18)
ax1.legend(fontsize=12)
```

```
ax1.grid(True)
ax1.tick_params(axis='both', which='major', labelsize=14)
# Green split line
ax1.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at {min_thresh:.1f}')
# To avoid duplicate legend
handles, labels = ax1.get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax1.legend(by_label.values(), by_label.keys(), fontsize=12)

# Bottom: Gini vs Perimeter Mean
ax2.plot(thresholds, ginis, color='blue', lw=2)
ax2.scatter([min_thresh], [min_gini], color='red', s=80, zorder=5, label='Minimum Gini')
ax2.axvline(min_thresh, color='green', linestyle='--', lw=2, label=f'Split at {min_thresh:.1f}')
ax2.set_xlabel('Perimeter Mean (µm)', fontsize=16)
ax2.set_ylabel('Gini impurity', fontsize=16)
ax2.set_title('Gini vs. Perimeter Mean', fontsize=18)
ax2.grid(True)
ax2.tick_params(axis='both', which='major', labelsize=14)
# To avoid duplicate legend
handles, labels = ax2.get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax2.legend(by_label.values(), by_label.keys(), fontsize=12, loc='upper right')

plt.tight_layout(h_pad=2)
plt.savefig("images/trees/scatter_and_gini_vs_perimeter.png")
plt.show()
```

The Gini Impurity Coefficient reaches a minimum of approximately 16.5 at a split value of about 885.7.

The best split is obtained by splitting the data based on Perimeter Mean at 96.6. This process is repeated every time a group is split.

12.4 Final Thoughts

The Gini Impurity Coefficient is the measure used to evaluate the quality of a split in Decision Tree learning. A good split partitions the data into two homogeneous groups.

To train a Decision Tree model, the algorithm finds the best data splits, using the Gini Impurity Coefficient as evaluation criterion.

The next chapter will explore how this splitting process is applied **recursively**.

12.5 Solutions

Solution 12.1. Exercise 12.1

First, compute the probabilities:

$$P(\text{malignant}) = \frac{2}{2+8} = 0.2$$
$$P(\text{benign}) = \frac{8}{2+8} = 0.8$$

Now, the probability of picking first a malignant tumour, then a benign tumour:

$$P(\text{malignant then benign}) = P(\text{malignant}) \cdot P(\text{benign}) = 0.2 \cdot 0.8 = 0.16$$

The probability of picking first a benign tumour, then a malignant tumour:

$$P(\text{benign then malignant}) = P(\text{benign}) \cdot P(\text{malignant}) = 0.8 \cdot 0.2 = 0.16$$

Sum the two to get the probability of picking items of different classes:

$$\begin{aligned} P(\text{picking items of different class}) &= P(\text{malignant then benign}) + P(\text{benign then malignant}) \\ &= 0.16 + 0.16 = 0.32 \end{aligned}$$

The Gini Impurity Coefficient is therefore 0.32. This is in line with our intuition as this number should be higher than for Group A, as Group B is more mixed.

Solution 12.2. Exercise 12.2

Group C

Group	Malignant	Benign
C	6	4

$$P(\text{malignant}) = \frac{6}{6+4} = 0.6$$
$$P(\text{benign}) = \frac{4}{6+4} = 0.4$$

Now, the probability of picking first a malignant tumour, then a benign tumour:

$$P(\text{malignant then benign}) = P(\text{malignant}) \cdot P(\text{benign}) = 0.6 \cdot 0.4 = 0.24$$

The probability of picking first a benign tumour, then a malignant tumour:

$$P(\text{benign then malignant}) = P(\text{benign}) \cdot P(\text{malignant}) = 0.4 \cdot 0.6 = 0.24$$

Sum the two to get the probability of picking items of different classes:

$$\begin{aligned} P(\text{picking items of different class}) &= P(\text{malignant then benign}) + P(\text{benign then malignant}) \\ &= 0.24 + 0.24 = 0.48 \end{aligned}$$

The Gini Impurity Coefficient for Group C is therefore 0.48.

Group D

Group	Malignant	Benign
D	5	5

$$P(\text{malignant}) = \frac{5}{5+5} = 0.5$$
$$P(\text{benign}) = \frac{5}{5+5} = 0.5$$

Now, the probability of picking first a malignant tumour, then a benign tumour:

$$P(\text{malignant then benign}) = P(\text{malignant}) \cdot P(\text{benign}) = 0.5 \cdot 0.5 = 0.25$$

The probability of picking first a benign tumour, then a malignant tumour:

$$P(\text{benign then malignant}) = P(\text{benign}) \cdot P(\text{malignant}) = 0.5 \cdot 0.5 = 0.25$$

Sum the two to get the probability of picking items of different classes:

$$\begin{aligned} P(\text{picking items of different class}) &= P(\text{malignant then benign}) + P(\text{benign then malignant}) \\ &= 0.25 + 0.25 = 0.5 \end{aligned}$$

The Gini Impurity Coefficient for Group D is therefore 0.5.

Chapter 13

Splitting Recursively

The Decision Tree learning algorithm is a **recursive** algorithm.

13.1 Functions

Recursion is a fascinating idea that builds on the concept of **function**. In programming, functions are reusable pieces of code that execute a piece of code

As an example, the following function adds two numbers and returns the result:

```
Adder(a, b):
    result = a + b
    return result
```

This is a simple program represented in pseudocode, a language between natural language (like English) and code.

What is pseudocode?

Pseudocode is a language in between natural language and code.

Natural languages are languages like English, French or Chinese that we use to communicate with one another. It is both:

- ambiguous: the same word can mean two or more meanings
- context-dependent: the meaning of a word depends on its context

Code is an unambiguous and context-free language. We use it to define programs to be run by computers.

Pseudocode is closer to natural language and allows us to communicate programs and ideas.

Going through this function line by line:

1. Defines a new function called `Adder`, that takes two input parameters `a` and `b`
2. Sums the two parameters and stores the result in the `result` variable
3. Returns the `result` variable to the user

In python code, this function definition would look like this:

```
def adder(a,b):
    result = a + b
    return result
```

Once defined, this function can be called over and over again, without having to copy-paste any code. Functions are a core building block of programming.

To call this function in python, you could write `adder(2,3)` and `5` would be the result.

13.2 Recursion

In Computer Science, recursion is when a **function calls itself**, directly or indirectly, to solve a problem by breaking it down into smaller subproblems.

This is the technical definition, this chapter will make this concept more concrete.

Let's take the Fibonacci sequence as an example:

0, 1, 1, 2, 3, 5, 8, …

In the Fibonacci sequence, the n^{th} term is the sum of the previous two, with the first two terms of the series being 0 and 1.

To calculate the 3rd term, we add the first two numbers together: $0 + 1 = 1$ To calculate the 4th, we add the second and the third number: $1 + 1 = 2$

We can continue this process:

- 5th: $1 + 2 = 3$
- 6th: $2 + 3 = 5$

Exercise 13.1. Compute the 7th and 8th term of the Fibonacci sequence.

To make this process more general, we can compute the n^{th} Fibonacci term, $Fibonacci(n)$, with the following expression:

$$Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)$$

This could be confusing, as a function is used in its own definition. Yet, it simply means that each term is the sum of the **two previous ones** in the series.

This expression could also be turned into the following program, which would compute the n^{th} Fibonacci term:

```
Fibonacci(n):
    if n == 0: return 0
    if n == 1: return 1
    return Fibonacci(n-1) + Fibonacci(n-2)
```

If you have never seen pseudocode before, this may look intimidating. It simply means the following (numbers in the list follow the line number in the code snippet):

1. Defines a new function called `Fibonacci`, this function takes one parameter called `n`
2. If that parameter is 0, the function returns 0
3. If that parameter is 1, the function returns 1
4. Otherwise, the function should return the sum of the two previous Fibonacci terms, represented as `Fibonacci(n-1)` and `Fibonacci(n-2)`

The last line is recursion in action: a function calling itself! To get `Fibonacci(3)` we need to compute `Fibonacci(2)` and `Fibonacci(1)`. To get `Fibonacci(2)`, we need `Fibonacci(1)` and `Fibonacci(0)`. This is exactly what recursion is about.

Note: In Computer Science, lists start at index 0. So the first term is index 0, the second is index 1 etc…

If you find this confusing, you are not alone. Let's compute an example with `Fibonacci(3)`, which will return the 4th term of the Fibonacci series (starting at index 0). The following will run the function line by line:

```
Fibonacci(3)
3 is neither 0 nor 1
Calling Fibonacci(2) and Fibonacci(1)
    Fibonacci(2)
    2 is neither 0 nor 1
        Calling Fibonacci(1) and Fibonacci(0)
            Fibonacci(1)
            Return 1
            Fibonacci(0)
            Return 0
    Return 1 + 0 = 1
    Fibonacci(1)
    Return 1
Return 1 + 1 = 2
```

In the above example, each indentation represents a level of recursion.

13.2.1 Recursive Splitting

The splitting logic of a Decision Tree works in a similar way. It keeps splitting subgroups until the stopping condition is met. It could be defined in pseudocode as follows:

```
1 Split(Group):
2     If the group is "pure" do nothing
3     Find the best split feature and split value
4     Split the group into two: Subgroup A, and Subgroup B
5     Apply Split(SubgroupA) and Split(SubgroupB)
```

In the program defined above, line 2 defines the stopping criterion. In the case that the group contains only one class, the function terminates. Otherwise, it continues recursion with lines 3-5. It keeps splitting the data.

Let's apply this logic to the example data step by step:

```
Split(Dataset)
Group contains two classes, proceed to splitting
Best split is found at x1 = 2
Splits Dataset into Groups A and B
Calls Split(Group A) and Split(Group B)
```

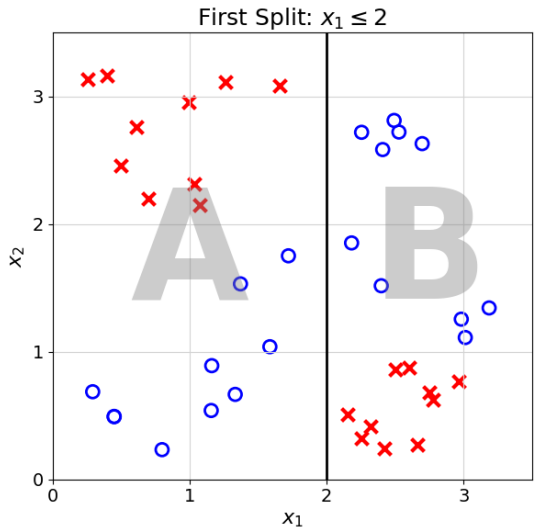


Figure 13.1: Splitting the dataset

```
    Split(Group B)
    Proceed to splitting as group contains two classes
    Best split is found at x2 = 1
```

```
    Splits B into Groups C and D
    Calls Split(Group C) and Split(Group D)
    :::
```

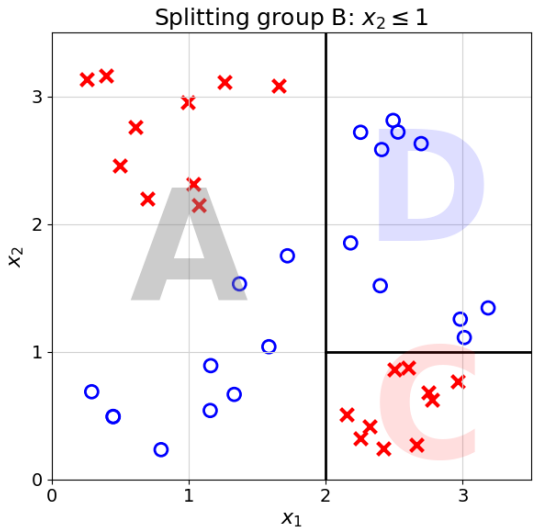


Figure 13.2: Splitting Group B

```
        Split(Group C)
        Does nothing as Group C contains only one class
        Split(Group D)
        Does nothing as Group D contains only one class
    Split(Group A)
    Proceed to splitting as group contains two classes
    Best split is found at x2 = 2
    Splits Group A into Groups E and F
    Calls Split(Group E) and Split(Group F)
```

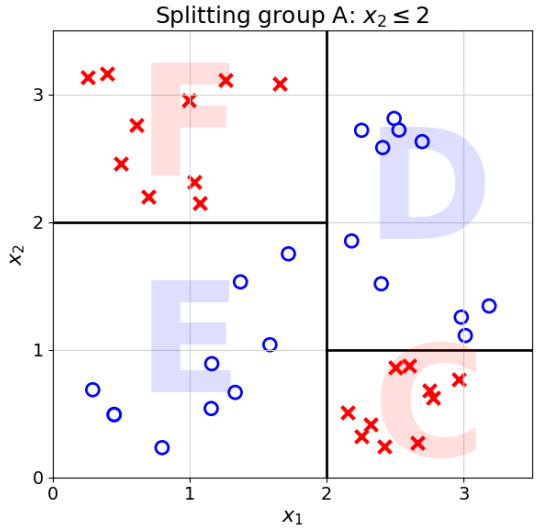


Figure 13.3: Splitting Group A

```
        Split(Group E)
        Does nothing as Group E contains only one class
        Split(Group F)
        Does nothing as Group F contains only one class
Algorithm terminates
```

In Computer Science, it is not recursion if your head does not hurt thinking about it.

13.3 Reviewing the Decision Tree Learning Algorithm

The previous chapter has reviewed two foundational concepts of Decision Tree learning:

1. **Recursion**: process calling itself
2. **Gini Impurity Coefficient**: a way to grade different data splits

The Decision Tree Learning algorithm can then be summarised as follows:

```
Split(Group)
    If the group contains only one class or is too small, do nothing
    Find the best split using the Gini Impurity criterion
    Split the data into two subgroups
    Apply the Split function to the two resulting subgroups
```

This function will build a tree, i.e., a collection of splits and leaf nodes, until there is no further way to split the data. That is it!

To predict for a new observation, they can be passed down the tree. The resulting prediction will be the label that constitutes the majority of the observations in the leaf.

13.4 Final Thoughts

This chapter introduced the foundational concept of **recursion**. Recursion happens when a function calls itself. In the case of Decision Trees the splitting function first splits the dataset into two groups (see previous chapter), and then **calls itself** on each of the subgroups. This is done until no further groups can be split.

The next chapter will add some nuance and bring the entire Decision Tree algorithm together.

13.5 Solutions

Solution 13.1. Exercise 13.1

- 7th: $3 + 5 = 8$
- 8th: $5 + 8 = 13$

Chapter 14

Probabilities and Regression with Decision Trees

The Decision Tree algorithm introduced in this section has two components:

- Identifying the best split with an **evaluation criterion** (e.g., the Gini Impurity Coefficient)
- Splitting the data **recursively** until no more splitting is possible

The following chapter will explore how to use this algorithm to output probabilities instead of class labels, and extend this to regression with Decision Trees.

14.1 From labels to probabilities

The Decision Tree algorithm shown above only outputs class labels as prediction ("malignant" or "benign"). In the simple version described above, an observation is assigned a prediction using a **majority vote** within the leaf. This is similar to the KNN model. How could the Decision Tree algorithm be used to output **probabilities**?

Let's think about this problem using a modified version of the example data:

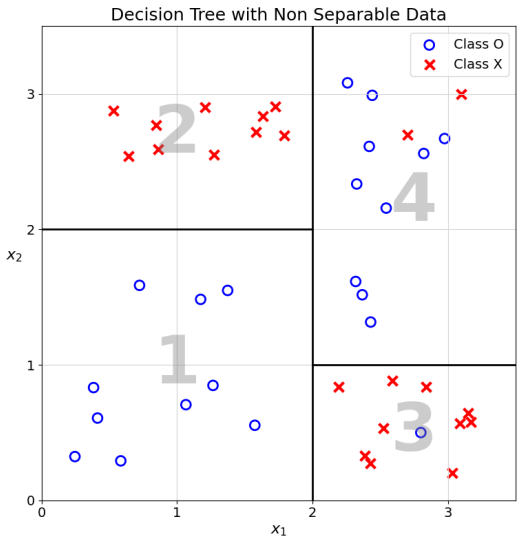


Figure 14.1: Non separable example data

In this example, leaf number 3 and 4 both contain observations from the two classes. Now, imagine that a new observation has $x_1 = 2.1$ and $x_2 = 0.5$, what prediction would we output?

Based on the values of this observation's feature, it will land in leaf number 3. In this leaf, the large majority of training observations are of class $\times$ (10 out of 11) with a single $\circ$ observation.

Using a similar approach to what was used in KNN, instead of applying a majority vote within a leaf, we could use the **frequency** of the class as a predicted probability.

In this example, the new observation will have a probability of $\times$ of:

$$\frac{10}{11} \approx 0.909$$

Exercise 14.1. Compute the probability of $\times$ for an observation with features: $x_1 = 3$ and $x_2 = 2$. Show that it is approximately 0.17

That is it, nothing more complex.

14.2 From classification to regression

So far, we have focussed on classification problems, in which the objective is to assign a label (such as the malignancy of a tumour) to new observations. How

can the same model be applied to a regression problem, i.e., to the prediction of a continuous variable, like the price of a property.

The core idea would remain the same: split the data recursively into subgroups to make them as "pure" as possible. This concept of "purity" or "homogeneity" is easy to define in a classification problem. A group that is "pure" is a group that contains a large majority of one class.

In regression problems, how could we define this concept of homogeneity? Think about it in terms of property prices. A homogeneous group would be a group that contains properties with **similar prices**. In such a group, each item would have little difference with the average price.

Now, how to generate predictions of **continuous values** from these subgroups? In the classification case, we could just use majority vote or average for probability predictions. In line with what was shown with KNN, the same principle of average can be used for regression problems.

The predicted price of each new property would be the **average of all the property prices in the same subgroup.** This is similar to the logic of probability prediction described earlier.

A split is good if the average in each leaf has a low error, if the average is an accurate prediction of the observations in this leaf.

Let's illustrate this with a simple example.

14.2.0.1 Splitting Data

Imagine we are building a model predicting property prices based on Property Size (m^2) and Distance to Centre (km):

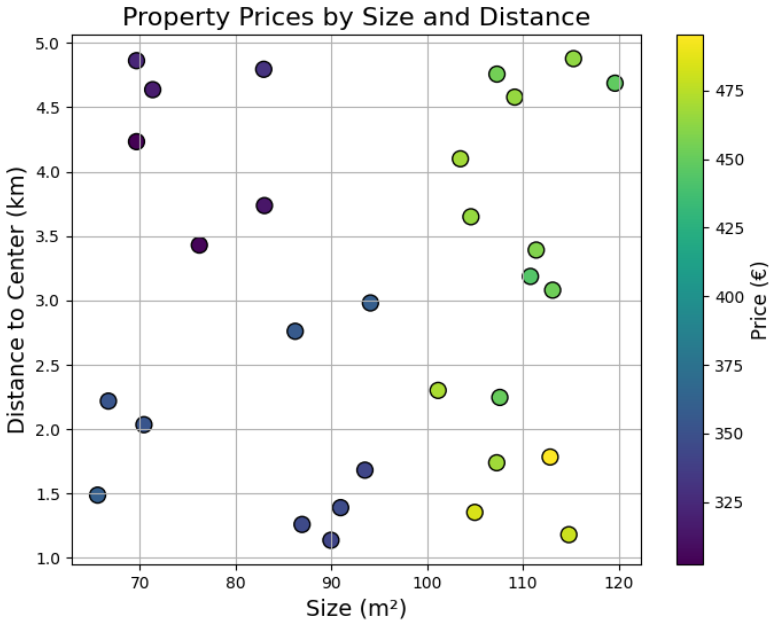


Figure 14.2: Properties by Size and Distance coloured by Property Price

Figure Code

```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Generate features
size = np.random.uniform(50, 150, 30)
distance = np.random.uniform(1, 5, 30)

# Generate price with some relationship: higher size increases price, higher distance d
price = 200000 + size * 2500 - distance * 15000 + np.random.normal(0, 15000, 30)

# Plot
plt.figure(figsize=(8,6))
sc = plt.scatter(size, distance, c=price, cmap='viridis', s=100, edgecolor='k')
plt.xlabel('Size (m²)', fontsize=14)
plt.ylabel('Distance to Center (km)', fontsize=14)
plt.title('Property Prices by Size and Distance', fontsize=16)
```

```
cbar = plt.colorbar(sc)
cbar.set_label('Price ($)', fontsize=12)
plt.grid(True)
plt.savefig("images/trees/property_prices.png")
plt.show()
```

We could take a subset of this data (prices now in **k €**):

Size (m^2)	Distance (km)	Price (k €)
60	2.0	320
80	1.5	400
120	5.0	350
70	3.0	310
150	1.0	600
90	4.0	330

Suppose the first split is on `size < 100`. This divides the data into two groups:

- **Group A sizes** (`size < 100`): 60, 80, 70, 90
- **Group B sizes** (`size  100`): 120, 150

How good is this split? To do this, we would compute the average of Group A and B, these will be our prediction for both groups.

The prediction for Group A is the average price in Group A:

Group A Mean $= \frac{320+400+310+330}{4} = 340$

The prediction for Group B is the average price in Group B:

Group B Mean $= \frac{350+600}{2} = 475$

We now need to measure how good of a prediction these averages are. How would you do it?

If you thought of the Mean Squared Error, well done! You can refer to the Model Evaluation chapter if this concept is still not clear enough.

For each group, we can compute the **Mean Squared Error (MSE)** of the prices:

- Group A:

$$\begin{aligned} \text{MSE}_a &= \frac{(320-340)^2 + (400-340)^2 + (310-340)^2 + (330-340)^2}{4} \\ &= \frac{400 + 3600 + 900 + 100}{4} \\ &= \frac{5000}{4} = 1250 \end{aligned}$$

- Group B:

$$\begin{aligned}\text{MSE}_b &= \frac{(350-475)^2 + (600-475)^2}{2} \\ &= \frac{(-125)^2 + (125)^2}{2} \\ &= \frac{15625 + 15625}{2} \\ &= \frac{31250}{2} = 15625\end{aligned}$$

This is a good start, but leaves us with two MSE numbers. How could we summarise these into one?

One idea is to compute the **average MSE**. It is a weighted average of the MSEs of the two groups (weighted by the number of observations in each group):

$$\begin{aligned}\text{Weighted MSE} &= \frac{n_a}{n_a + n_b} \cdot \text{MSE}_a + \frac{n_b}{n_a + n_b} \cdot \text{MSE}_b \\ &= \frac{4}{6} \cdot 1250 + \frac{2}{6} \cdot 15625 = 833.33 + 5208.33 = 6041.67\end{aligned}$$

14.2.1 Trying Different Splits

At each splitting step, the algorithm would try different splitting features and values, and would pick the one that minimises average MSE.

This process can be visualised by showing the average MSE for each splitting value of the **size** feature:

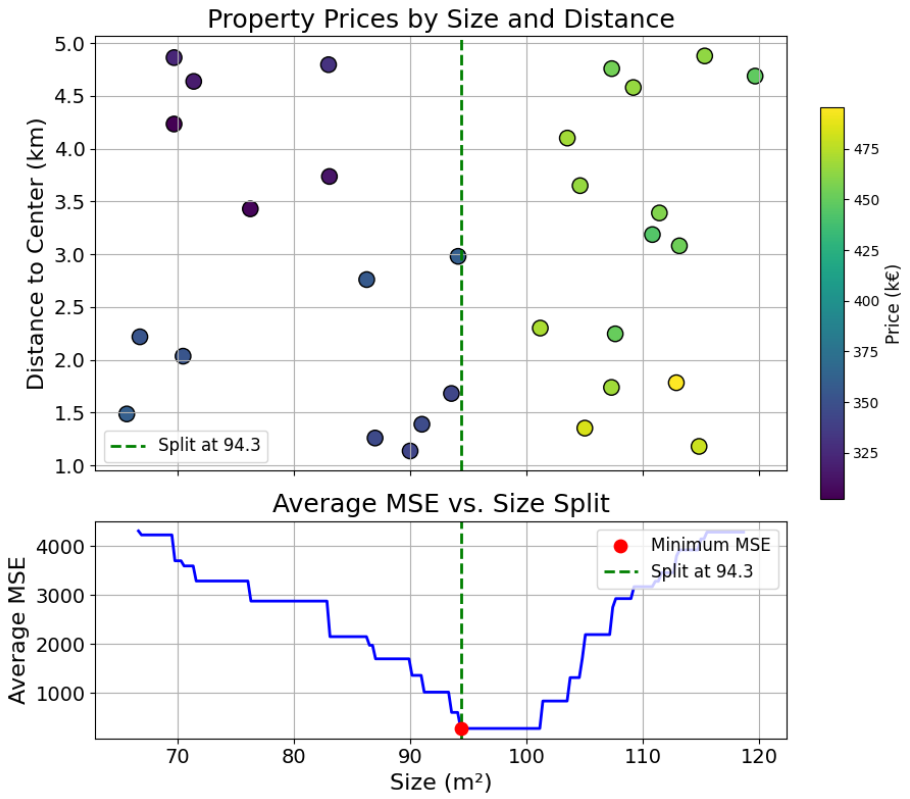


Figure 14.3: Trying different splitting values of the size feature

Figure Code

```
import numpy as np
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(42)
size = np.random.uniform(50, 150, 30)
distance = np.random.uniform(1, 5, 30)
price = 200000 + size * 2500 - distance * 15000 + np.random.normal(0, 15000, 30)
price = price / 1000  # convert to k€

X = np.column_stack([size, distance])

def avg_mse(split_value, X, price):
    left_mask = X[:, 0] < split_value
    right_mask = ~left_mask
```

```
    n = len(price)
    n_left = np.sum(left_mask)
    n_right = np.sum(right_mask)
    if n_left == 0 or n_right == 0:
        return np.nan
    mean_left = price[left_mask].mean()
    mean_right = price[right_mask].mean()
    mse_left = np.mean((price[left_mask] - mean_left) ** 2)
    mse_right = np.mean((price[right_mask] - mean_right) ** 2)
    avg = (n_left / n) * mse_left + (n_right / n) * mse_right
    return avg

split_values = np.linspace(size.min() + 1, size.max() - 1, 200)
avg_mses = np.array([avg_mse(sv, X, price) for sv in split_values])
best_idx = np.nanargmin(avg_mses)
best_split = split_values[best_idx]
best_mse = avg_mses[best_idx]

# Plot
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(9, 8), sharex=True, gridspec_kw={'height

# Top: Data scatter
sc = ax1.scatter(size, distance, c=price, cmap='viridis', s=120, edgecolor='k')
ax1.set_ylabel('Distance to Center (km)', fontsize=16)
ax1.set_title('Property Prices by Size and Distance', fontsize=18)
ax1.axvline(best_split, color='green', linestyle='--', lw=2, label=f'Split at {best_sp
ax1.legend(fontsize=12)
ax1.grid(True)
ax1.tick_params(axis='both', which='major', labelsize=14)

# Bottom: MSE vs Size
ax2.plot(split_values, avg_mses, color='blue', lw=2)
ax2.scatter([best_split], [best_mse], color='red', s=80, zorder=5, label='Minimum MSE'
ax2.axvline(best_split, color='green', linestyle='--', lw=2, label=f'Split at {best_sp
ax2.set_xlabel('Size (m²)', fontsize=16)
ax2.set_ylabel('Average MSE', fontsize=16)
ax2.set_title('Average MSE vs. Size Split', fontsize=18)
ax2.grid(True)
ax2.tick_params(axis='both', which='major', labelsize=14)
ax2.legend(fontsize=12, loc='upper right')

# Add a colorbar that doesn't affect axis alignment
fig.subplots_adjust(right=0.88)
cbar_ax = fig.add_axes([0.90, 0.38, 0.025, 0.48])
cbar = fig.colorbar(sc, cax=cbar_ax)
```

```
cbar.set_label('Price (k€)', fontsize=12)

plt.tight_layout(rect=[0, 0, 0.88, 1])
plt.savefig("images/trees/property_mse_vs_size.png")
plt.show()
```

Following the same **recursive process** as the one shown in the classification case, we can build a Decision Tree that partitions the overall space into subgroups.

This stepwise learning process is shown below:

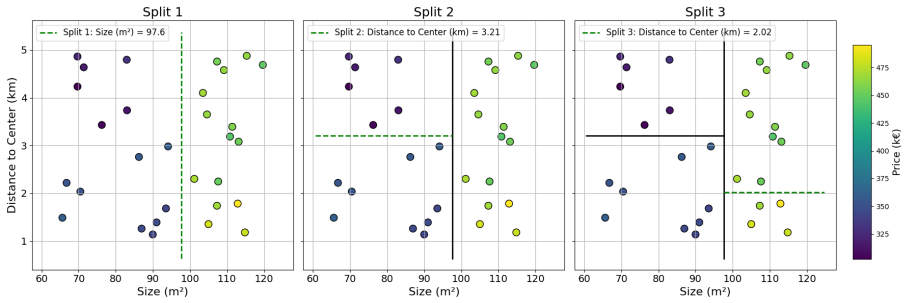


Figure 14.4: The Decision Tree learning algorithm selects the best split at each step

Figure Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

np.random.seed(42)
n = 30

# Generate two size bands, each with a narrow range
size = np.concatenate([
    np.random.uniform(65, 95, n//2),   # left band
    np.random.uniform(100, 125, n//2)  # right band
])

# Distance: spread out for both bands
distance = np.concatenate([
    np.random.uniform(1, 5, n//2),
    np.random.uniform(1, 5, n//2)
])
```

```
# Price: within each band, a strong step at different distance values
price = np.empty(n)
# Left band: step at distance=3
price[:n//2] = np.where(distance[:n//2] < 3, 350, 320) + np.random.normal(0, 10, n//2)
# Right band: step at distance=2.2
price[n//2:] = np.where(distance[n//2:] < 2.2, 480, 455) + np.random.normal(0, 10, n//2

X = np.column_stack([size, distance])
feature_names = ['Size (m²)', 'Distance to Center (km)']

# Fit regression tree with depth=2
tree = DecisionTreeRegressor(max_depth=2, random_state=0)
tree.fit(X, price)

# Helper: collect splits recursively
def collect_splits(tree, node_id=0, path=[], splits=[]):
    feature = tree.tree_.feature[node_id]
    threshold = tree.tree_.threshold[node_id]
    if feature >= 0:
        splits.append((path.copy(), feature, threshold))
        collect_splits(tree, tree.tree_.children_left[node_id], path + [(feature, thres
        collect_splits(tree, tree.tree_.children_right[node_id], path + [(feature, thre
    return splits

splits = collect_splits(tree, 0, [], [])

# Print splits for verification
for i, (path, feat, thresh) in enumerate(splits):
    print(f"Split {i+1}: feature {feat} ({feature_names[feat]}), threshold {thresh:.2f}

# Plot (reuse your plotting code here)
x0min, x0max = size.min() - 5, size.max() + 5
x1min, x1max = distance.min() - 0.5, distance.max() + 0.5

n_splits = len(splits)
fig, axes = plt.subplots(1, n_splits, figsize=(6*n_splits, 6), sharex=True, sharey=True

if n_splits == 1:
    axes = [axes]

for i in range(n_splits):
    ax = axes[i]
    sc = ax.scatter(size, distance, c=price, cmap='viridis', s=100, edgecolor='k')
    # Draw all previous splits as black lines, in their respective regions
    for j in range(i):
```

```
        path, feat, thresh = splits[j]
        xlims = [x0min, x0max]
        ylims = [x1min, x1max]
        for (pf, pt, dirn) in path:
            if pf == 0:
                if dirn == 'left':
                    xlims[1] = min(xlims[1], pt)
                else:
                    xlims[0] = max(xlims[0], pt)
            else:
                if dirn == 'left':
                    ylims[1] = min(ylims[1], pt)
                else:
                    ylims[0] = max(ylims[0], pt)
        if feat == 0:
            ax.plot([thresh, thresh], ylims, color='black', linewidth=2)
        else:
            ax.plot(xlims, [thresh, thresh], color='black', linewidth=2)

    # Draw current split as green dashed line, in its region
    path, feat, thresh = splits[i]
    xlims = [x0min, x0max]
    ylims = [x1min, x1max]
    for (pf, pt, dirn) in path:
        if pf == 0:
            if dirn == 'left':
                xlims[1] = min(xlims[1], pt)
            else:
                xlims[0] = max(xlims[0], pt)
        else:
            if dirn == 'left':
                ylims[1] = min(ylims[1], pt)
            else:
                ylims[0] = max(ylims[0], pt)
    if feat == 0:
        ax.plot([thresh, thresh], ylims, color='green', linestyle='--', linewidth=2,
                label=f'Split {i+1}: {feature_names[feat]} = {thresh:.1f}')
    else:
        ax.plot(xlims, [thresh, thresh], color='green', linestyle='--', linewidth=2,
                label=f'Split {i+1}: {feature_names[feat]} = {thresh:.2f}')
    ax.set_xlabel('Size (m²)', fontsize=16)
    if i == 0:
        ax.set_ylabel('Distance to Center (km)', fontsize=16)
    ax.set_title(f'Split {i+1}', fontsize=18)
    ax.grid(True)
```

```
    ax.tick_params(axis='both', which='major', labelsize=14)
    ax.legend(fontsize=12, loc='upper left')

# Add colorbar
fig.subplots_adjust(right=0.92)
cbar_ax = fig.add_axes([0.93, 0.15, 0.02, 0.7])
cbar = fig.colorbar(sc, cax=cbar_ax)
cbar.set_label('Price (k€)', fontsize=14)

plt.tight_layout(rect=[0, 0, 0.92, 1])
plt.savefig("images/trees/property_regression_splits_stepwise.png")
plt.show()
```

As described in this section, Decision Trees can be easily applied to regression tasks. The only significant change is the evaluation of split quality with the **Mean Squared Error** instead of the Gini Impurity Coefficient.

14.3 Final Thoughts

This chapter concludes our exploration of Decision Trees. Summarising once more the Decision Tree Learning algorithm:

- Evaluate splits using a homogeneity criterion: Gini or MSE
- Split groups recursively
- Output either class labels or probabilities using averaging

To test your knowledge, you can try the practice exercise below.

Looking back, we now know how to train and evaluate two different Machine Learning models. This is already a lot. The next section will explore the last missing piece of the puzzle: Data Preprocessing, making data ready for modelling.

14.4 Practice Exercise

Exercise 14.2. Suppose you are building a Decision Tree to detect fraudulent transactions. You use two features, both measured on a 0–100 scale:

- **Transaction Amount ($)** (0–100)
- **Customer Age (years)** (0–100)

You have the following 10 transactions in your training data:

Transaction Amount	Customer Age	Fraudulent?
95	22	Yes
90	25	Yes

Transaction Amount	Customer Age	Fraudulent?
92	23	Yes
97	21	Yes
93	24	Yes
94	23	No
20	80	No
25	78	No
18	82	No
23	77	No

A new transaction occurs with an amount of **93** and customer age **23**.

1. Build a Decision Tree with a **single split** (for simplicity) using the training data and finding the splits that minimise the Gini Impurity Coefficient?
2. Using the tree generated, what is the predicted probability of fraud of the new observation?

Solution 14.2

14.5 Solutions

Solution 14.1. Exercise 14.1

$\frac{2}{12} = \frac{1}{6} \approx= 0.17$

Solution 14.2. Exercise 14.2

1. We try splitting the data using different features and feature values. For each split, we compute the weighted Gini Impurity Coefficient. The resulting coefficients can be seen in the two tables below. For each table, the symbols $\leq$ and $>$ represent the two resulting subgroups:

- $\leq$: observations with feature value lower or equal to the splitting value
- $>$: observations with feature value greater than the splitting value

Splits for **Transaction Amount**

Split at	Split Fraud	Split Non-Fraud	> Split Fraud	> Split Non-Fraud	Weighted Gini
19.00	0	1	5	4	0.444
21.50	0	2	5	3	0.375
24.00	0	3	5	2	0.286
57.50	0	4	5	1	**0.167**
91.00	1	4	4	1	0.320
92.50	2	4	3	1	0.417
93.50	3	4	2	1	0.476

Split at	Split Fraud	Split Non-Fraud	> Split Fraud	> Split Non-Fraud	Weighted Gini
94.50	3	5	2	0	0.375
96.00	4	5	1	0	0.444

Splits for **Customer Age**

Split at	Split Fraud	Split Non-Fraud	> Split Fraud	> Split Non-Fraud	Weighted Gini
21.50	1	0	4	5	0.444
22.50	2	0	3	5	0.375
23.00	3	1	2	4	0.417
23.50	3	1	2	4	0.417
24.50	4	1	1	4	0.320
51.00	5	1	0	4	**0.167**
77.50	5	2	0	3	0.286
79.00	5	3	0	2	0.375
81.00	5	4	0	1	0.444

Both Transaction Amount of 57.5 and Customer Age of 51 achieve a Gini Coefficient of 0.167. Both of them would work. This solution will pick Customer Age = 51 as splitting value.

The resulting tree is:

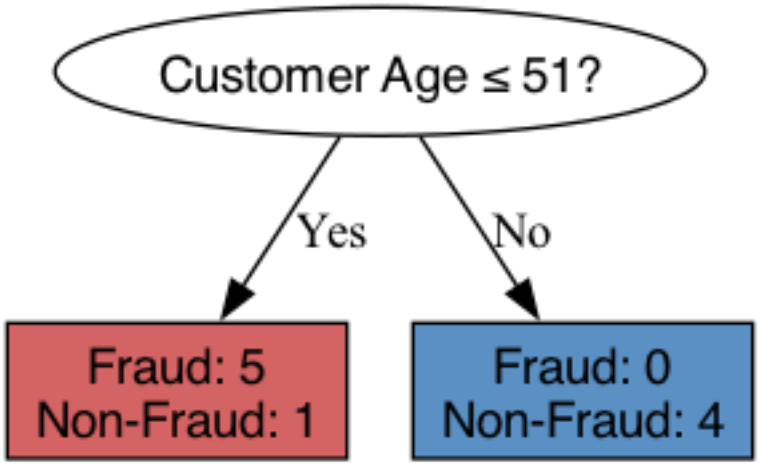


Figure 14.5: Resulting tree splitting on Customer Age 51

1. Generating the prediction for observation: Customer Age: 23 and Transaction Amount: 93, the observation will fall into the left leaf, as $23 \leq 51$.

The predicted probability for this observation is then the average probability of Fraud in the leaf:

$$P(\text{Fraud}) = \frac{5}{5+1} \approx 83\%$$

Part IV

Data Preprocessing

Chapter 15

Data Preprocessing

Now that we have a good idea of what prediction with Machine Learning looks like, let's start implementing these methods to real-world examples.

Can you, today, take data on something you want to predict and apply a KNN or a Decision Tree algorithm? Most likely not, because we are still missing a step.

This chapter will bridge the gap between the big ideas described in the previous chapters and real-world ML applications: **data preprocessing**.

You may have noticed that all the data used in this book was **synthetic**, i.e., generated by a quick Python script. There are several reasons for this:

- Ensuring that the data has the required properties
- Copyrights, legal and ethical issues
- Convenience

This generated data had some characteristics you may have noted:

- Only numeric features, no categorical or date fields
- Features on the same scale (most of the time)
- No missing data

In other words, in all of the examples, the data was represented as a clean table of data with numbers on the same scale.

This will **not** be the case for most of the datasets you will come across. It is a messy world out there. Some data will be missing, some data will not even be numbers.

The following chapters will explore ways to deal with these issues one by one. This last part of the book will allow you to apply Machine Learning models to any regression or classification problem you come across.

Note: Preprocessing is a common source of **information leakage** between the training set and the test set. If the idea of train/test split is not clear, I would recommend reading through the Model Evaluation chapter once more. The concept of information leakage will be explored further in this section.

Chapter 16

Encoding Categorical Features

Not all data is numerical. Categorical features represent data **groupings**. As an example, a property in Berlin can have a given neighbourhood, e.g., "Neukölln", "Kreuzberg" or "Charlottenburg".

Property	Surface Area (sq m)	Distance to Centre (km)	Neighbourhood	Street Name	Energy Rating	Sell Price (K€)
A	85	4.2	Neukölln	Sonnenallee	B	420
B	120	2.5	Kreuzberg	Bergmannstr.	A	610
C	95	6.1	Charlottenburg	Kantstr.	C	390
D	70	3.8	Kreuzberg	Bergmannstr.	D	370
E	110	1.2	Charlottenburg	Unter Den Linden	B	700
F	60	5.0	Neukölln	Sonnenallee	F	310

Exercise 16.1. Can you spot the other categorical variables in the above table?

All of the Machine Learning models studied so far deal with observations as **points in space**. Numerical features naturally determine coordinates of points in space. Properties can be plotted by their surface area and distance to the centre of town without problems:

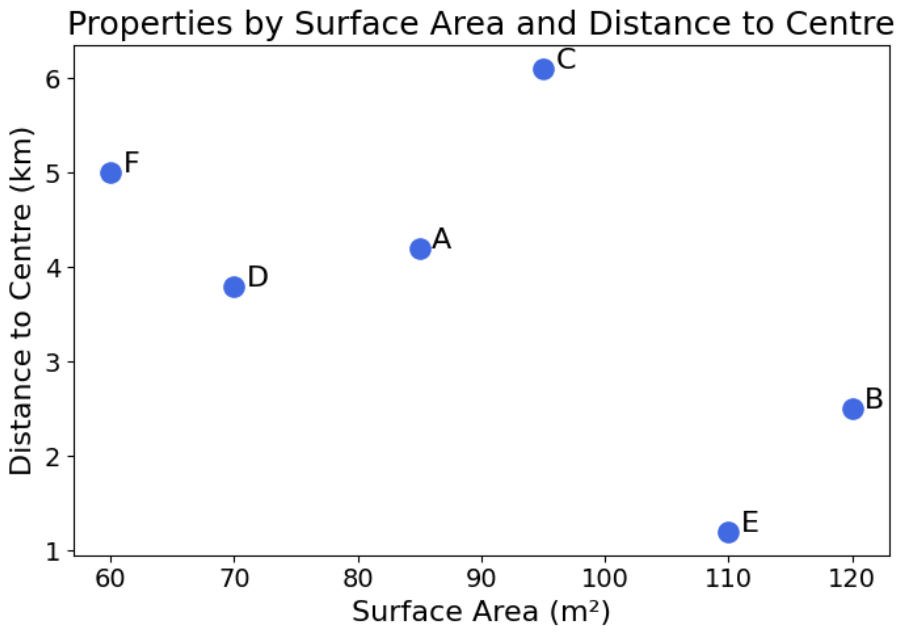


Figure 16.1: Properties plotted by surface area and distance to centre

Figure code

```
import matplotlib.pyplot as plt

surface_area = [85, 120, 95, 70, 110, 60]
distance_centre = [4.2, 2.5, 6.1, 3.8, 1.2, 5.0]
labels = ['A', 'B', 'C', 'D', 'E', 'F']

plt.figure(figsize=(7,5))
plt.scatter(surface_area, distance_centre, s=120, c='royalblue')
for i, label in enumerate(labels):
    plt.text(surface_area[i]+1, distance_centre[i], label, fontsize=16)
plt.xlabel('Surface Area (sq m)', fontsize=16)
plt.ylabel('Distance to Centre (km)', fontsize=16)
plt.title('Properties by Surface Area and Distance to Centre', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

We could try building a model that ignores neighbourhood, and uses “Distance to the Centre” as a proxy. This is not a bad idea, but a lot of information is

lost in the process. And what is the centre of Berlin anyway?

How would models handle these categorical variables like neighbourhoods? This is what this chapter is about.

16.1 Types of Categorical

There are two main types of categorical features:

- Ordinal variables: these variables **have an order**. The energy efficiency rating of a property ranges from A (most efficient) to G (least efficient)
- Nominal variables: these variables **have no intrinsic order**. No possible ranking can be made of the neighbourhoods (e.g., "Neukölln" or "Kreuzberg") mentioned in the previous example

16.2 Ordinal Variables

Ordinal variables can be converted to numbers relatively easily. As an example, you could map the energy rating from A to G to the **numbers 1 to 7**. This would create another dimension in the feature space of properties:

Property	Surface Area (sq m)	Distance to Centre (km)	Energy Rating	Encoded Energy Rating
A	85	4.2	B	2
B	120	2.5	A	1
C	95	6.1	C	3
D	70	3.8	D	4
E	110	1.2	B	2
F	60	5.0	F	6

Below is a plot of the properties in three dimensions: surface area, distance to centre, and encoded energy class.

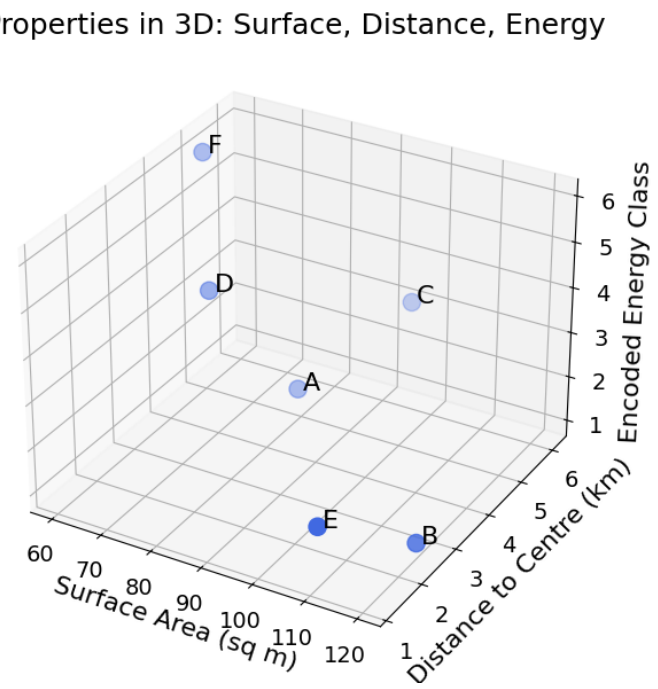


Figure 16.2: Properties in 3D: Surface Area, Distance to Centre, Encoded Energy Class

Figure code

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

surface_area = [85, 120, 95, 70, 110, 60]
distance_centre = [4.2, 2.5, 6.1, 3.8, 1.2, 5.0]
energy_rating = ['B', 'A', 'C', 'D', 'B', 'F']
encoded_energy = [2, 1, 3, 4, 2, 6]
labels = ['A', 'B', 'C', 'D', 'E', 'F']

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(surface_area, distance_centre, encoded_energy, s=120, c='royalblue')
for i, label in enumerate(labels):
    ax.text(surface_area[i]+1, distance_centre[i], encoded_energy[i], label, fontsize=
ax.set_xlabel('Surface Area (sq m)', fontsize=16)
ax.set_ylabel('Distance to Centre (km)', fontsize=16)
ax.set_zlabel('Encoded Energy Class', fontsize=16)
ax.set_title('Properties in 3D: Surface, Distance, Energy', fontsize=18)
```

```
ax.tick_params(axis='both', labelsize=14)
plt.tight_layout()
plt.savefig('figures/3d_surface_distance_energy.png')
plt.show()
```

If the notion of space is not fully clear, refer to the chapter on data space.

16.3 Nominal Variables

Handling nominal variables is trickier, as a direct conversion to numbers would not make sense. Why would "Neukölln" be 1 and "Kreuzberg" be 2? Or the other way around? We need a smarter solution. The following sections will describe two of them.

16.3.1 One-Hot Encoding

The best way to understand One-Hot Encoding is to visualise it before explaining it. Imagine the following dataset:

Property	Neighbourhood	Surface Area (sq m)	Sell Price (K€)
A	Neukölln	85	420
B	Kreuzberg	120	610
C	Charlottenburg	95	390
D	Kreuzberg	70	370
E	Charlottenburg	110	700
F	Neukölln	60	310

The One-Hot Encoded dataset would look like this:

Property	Surface Area (sq m)	Sell Price (K€)	Kreuzberg	Charlottenburg	Neukölln
A	85	420	0	0	1
B	120	610	1	0	0
C	95	390	0	1	0
D	70	370	1	0	0
E	110	700	0	1	0
F	60	310	0	0	1

What happened there? The original neighbourhood column disappeared, and was replaced by three columns containing either 1 or 0.

One-Hot Encoding converts a categorical variable into a list of **binary columns**, indicating whether or not the property **belongs** to a category. The advantage of this method is that it assumes no order.

A drawback of this approach is that it can create **many columns**. Imagine that you want to apply the same method to the street name. You may end up with **thousands of columns**. For mathematical reasons we will not go into, this can be an issue for Machine Learning algorithms. This is sometimes referred to as **the curse of dimensionality**. If you want to develop a quick intuition of why, just remember that a lot of strange things happen in a space with many dimensions.

One-Hot Encoding in practice

In practice, One-Hot Encoding converts a categorical variable with N distinct values into $N-1$ binary columns. Why do we remove one?

In statistics and Machine Learning, many issues arise when the features of a dataset are **perfectly correlated**. To avoid this, One-Hot Encoding drops one of the binary columns.

Going back to the example with three neighbourhood values (Kreuzberg, Charlottenburg and Neukölln), the processed dataset would look like this:

Property	Surface Area (sq m)	Sell Price (K€)	Kreuzberg	Charlotten-burg
A	85	420	0	0
B	120	610	1	0
C	95	390	0	1
D	70	370	1	0
E	110	700	0	1
F	60	310	0	0

Having both Kreuzberg and Charlottenburg as 0 would mean that the property is in Neukölln.

One-Hot Encoding solves the categorical variable problem for variables with **few categories**. How could we encode categorical variables like street name? Ignoring these variables with many distinct values is one possibility. However, in the example of property pricing, the street name could carry some relevant information. The next section will explore one idea.

16.4 Target Encoding

Thinking about a property pricing model, how could we represent the street name of a property as a number?

We want to be able to capture the **impact of the street name on the price**. To do so, we could encode each value of the categorical variable (e.g., "Sonnenallee" or "Bergmannstr.") as the **average** of the target variable for the

observations in this category. In this case, each street name would be replaced by the average price on that street.

This is an abstract definition, let's make it more concrete with an example:

Property	Surface Area (sq m)	Street Name	Sell Price (K€)
A	85	Sonnenallee	420
B	120	Bergmannstr.	610
C	95	Kantstr.	390
D	70	Bergmannstr.	370
E	110	Unter Den Linden	700
F	60	Sonnenallee	310

To encode the street name with Target Encoding, compute the average value of the target for each street name:

- Bergmannstr.: $\frac{610+370}{2} = 490$
- Unter Den Linden: 700 (only one property)

Exercise 16.2. Show that the average target value for Sonnenallee is 365.

The dataset with a target encoded street name feature now looks like this:

Property	Surface Area (sq m)	Street Name	Sell Price (K€)	Encoded Street Name
A	85	Sonnenallee	420	365
B	120	Bergmannstr.	610	490
C	95	Kantstr.	390	390
D	70	Bergmannstr.	370	490
E	110	Unter Den Linden	700	700
F	60	Sonnenallee	310	365

This way, a lot of the price relevant information of the street name is preserved. The main advantage of this method is that it can deal with categorical variables with many values **without creating too many dimensions**.

A possible drawback of this method is that if there are street names for which there is only one or a few properties, the average would only be the price of the **single property on this street**. There are smart ways to deal with these cases, though they are beyond the scope of this book.

If you are curious

One way to avoid the problem described above, you can decide on a threshold (e.g., 10), and encode any street name with fewer observations than 10 with the **global average** instead of the average for this street name.

This averaging process can also be used in the binary classification context. The average of the target variable for a given categorical value could be the average of 0's and 1's of the target label (i.e., negatives and positives).

16.5 Information Leakage

As mentioned in the introductory section, categorical encoding can be a source of **information leakage** between the training and test set. To avoid this leakage, it is critical to compute all these transformations on the **training set only**.

In One-Hot Encoding, only create columns for the categorical values **seen in the training set**. If a new value is found in the test set, it could be either excluded or flagged as missing (with another binary column). Using the entire dataset to encode categorical features would **not** simulate real prediction conditions.

For Target Encoding, the average target value for each category should only be computed on the **training data**. These averages can then be applied to the test set. If a new categorical value is found in the test set, it can be excluded or encoded with the global average of the target variable (over the entire dataset).

Why worry about this? When the model is used for prediction, the goal of splitting the data between train and test set is to estimate the performance of the model on unseen data. To do so, it is critical to compute preprocessing transformations on the training data. In the future, data will come one or more row at a time. The only averages available will be from the training data.

16.6 Final Thoughts

Without the appropriate encoding, categorical variables cannot be handled by Machine Learning models. ML models learn the relationships between inputs and a target variable using mathematical tricks applicable to lists of numbers (distance or splitting of space).

Encoding categorical variables converts them to numbers that can then be used in the training and prediction process. The encoding methods in this chapter include:

- **Ordinal Encoding**: mapping categorical values with an intrinsic order to a number
- **One-Hot Encoding**: representing categorical variables to a list of binary columns
- **Target Encoding**: mapping categorical values to the average target value for this group

The next chapter will explore another data type: dates and times.

16.7 Solutions

Solution 16.1. Exercise 16.1 The other two categorical variables are:

- Street Name
- Energy Rating

Solution 16.2. Exercise 16.2

$\frac{420+310}{2} = 365$

Chapter 17

Representing Dates

The world is in a constant state of change. Time is the concept we have created to reason about the wild **sequence of events** that we order in past, present and future. At our human scale, this time is represented using standard units such as: year, month, day, hour, minute and second.

I am writing these lines on Friday the 15th of August 2025 at 08:31 AM. Using the ISO 8601 standard, this point in time can be represented as: `2025-08-15T08:31:00`. The following can be extended to also include my time zone: `2025-08-15T08:31:00+02:00`, the `+02:00` at the end.

17.1 Dates and Prediction

Time indications like dates can contain information that can improve the quality of Machine Learning model predictions.

17.1.1 Overall Trend

Keeping the property pricing example, in which a Machine Learning model is trained to predict property prices from its characteristics like their surface area or number of rooms. Let's use the following training data as an example:

Property	Surface (m^2)	Rooms	Pricing date	Sale date	Sale price (K€)
A	85	3	2024-12-15	2025-02-13	302
B	86	3	2025-01-15	2025-03-16	305
C	84	3	2025-02-15	2025-04-16	303
D	87	3	2025-03-15	2025-05-14	311
E	85	3	2025-04-15	2025-06-14	309
F	86	3	2025-05-15	2025-07-14	316

Property	Surface (m^2)	Rooms	Pricing date	Sale date	Sale price (K€)
G	84	3	2025-06-15	2025-08-14	314
H	87	3	2025-07-15	2025-09-13	322

Which date column would you use to predict the sale price and why? The selling date could seem like a good idea. However, when pricing properties that have not been sold yet, this information is **not available** to the model. We do not yet know the sale date. The only date available to the pricing model would be the **pricing date**, date at which the property is put on the market.

For this reason, the sale date cannot be used in the model. The pricing date can still contain useful price information. Plotting the average sale price by pricing date (from the example data), we see the following trend:

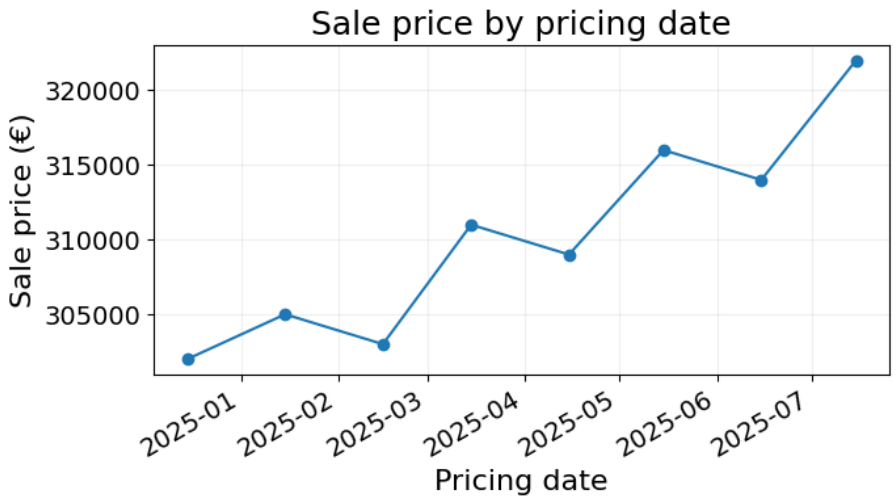


Figure 17.1: Sale price by pricing date

Figure code

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter

data = [
    ("A", 85, 3, "2024-12-15", "2025-02-13", 302000),
    ("B", 86, 3, "2025-01-15", "2025-03-16", 305000),
    ("C", 84, 3, "2025-02-15", "2025-04-16", 303000),
```

```
    ("D", 87, 3, "2025-03-15", "2025-05-14", 311000),
    ("E", 85, 3, "2025-04-15", "2025-06-14", 309000),
    ("F", 86, 3, "2025-05-15", "2025-07-14", 316000),
    ("G", 84, 3, "2025-06-15", "2025-08-14", 314000),
    ("H", 87, 3, "2025-07-15", "2025-09-13", 322000),
]
df = pd.DataFrame(data, columns=["Property","Surface","Rooms","Pricing date","Sale date","Sale pr
df["Pricing date"] = pd.to_datetime(df["Pricing date"])

fig, ax = plt.subplots(figsize=(7,4))
ax.plot(df["Pricing date"], df["Sale price"], marker="o", linestyle="-", color="tab:blue")
ax.set_title("Sale price by pricing date", fontsize=18)
ax.set_xlabel("Pricing date", fontsize=16)
ax.set_ylabel("Sale price (£)", fontsize=16)
ax.tick_params(axis="both", labelsize=14)
ax.grid(True, alpha=0.2)
ax.xaxis.set_major_formatter(DateFormatter("%Y-%m"))
fig.autofmt_xdate()
plt.tight_layout()
plt.show()
```

17.1.2 Seasonality

Changing example, let's say that as the manager of a pub, you are trying to predict the quantity of beer the bar will sell next week. Having an accurate beer sales forecast could help you better plan your staff and inventory. If you predict to sell a lot of beer, you should buy a lot of inventory and hire a larger team.

Beer sales over the last two years look like this:

Figure 17.2: Beer sales over the last two years with seasonality and trend

Figure code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(42)
dates = pd.date_range(end="2025-08-15", periods=104, freq="W-SUN")
t = np.arange(len(dates))

baseline = 800
trend = 5 * t
seasonality = 150 * np.sin(2 * np.pi * t / 52)
noise = np.random.normal(0, 40, size=len(dates))

sales = baseline + trend + seasonality + noise
sales = np.clip(sales, a_min=0, a_max=None)

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(dates, sales, color="tab:green", linewidth=2)
ax.set_title("Weekly beer sales (last two years)", fontsize=18)
ax.set_xlabel("Week", fontsize=16)
ax.set_ylabel("Pints sold", fontsize=16)
ax.tick_params(axis="both", labelsize=14)
ax.grid(True, alpha=0.2)
fig.autofmt_xdate()
plt.tight_layout()
plt.show()
```

From this chart, we could see both an **upward trend** and **seasonality effects**. The next section will investigate how to encode this information into numbers.

17.2 Encoding Dates as Numbers

As described in previous chapters, Machine Learning models generate predictions by learning the relationships between inputs and outputs. To do so, they represent each observation as a list of numbers, and use mathematical tricks (like distance or space splitting) to map features to a target variable.

To make dates readable to a model, they need to be encoded as numbers. Take a moment to think about how you would represent `2025-08-15T08:31:00` as numbers before reading on.

17.2.1 Splitting Dates into Parts

The date `2025-08-15T08:31:00` can be split into the following parts:

Year: 2025
Month: August, or 8
Week number: 33
Day: 15
Day of week: Friday, or 5 given a start at 1 on Monday
Hour: 8
Minute: 31
Second: 00

This date could then be represented as the following **list of numbers**: $2025, 8, 33, 15, 5, 8, 31, 00$. These values could be used as normal numeric features.

Encoding the pricing date of the example data using this method, we get:

Pricing date	Year	Month	ISO week	Day	Day of week (Mon=1)	Hour	Minute	Second
2024-12-15 09:00:00	2024	12	50	15	7	9	0	0
2025-01-15 09:00:00	2025	1	3	15	3	9	0	0
2025-02-15 09:00:00	2025	2	7	15	6	9	0	0

Pricing date	Year	Month	ISO week	Day	Day of week (Mon=1)	Hour	Minute	Second
2025-03-15 09:00:00	2025	3	11	15	6	9	0	0
2025-04-15 09:00:00	2025	4	16	15	2	9	0	0
2025-05-15 09:00:00	2025	5	20	15	4	9	0	0
2025-06-15 09:00:00	2025	6	24	15	7	9	0	0
2025-07-15 09:00:00	2025	7	29	15	2	9	0	0

That should already achieve good performance. You may notice that some date features may be more relevant than others depending on the task.

As an example, when predicting ice cream sales, the month of year, week number and day of week (e.g., Friday, Saturday, Sunday) could be important. Whereas the day number (e.g., 1 or 15) may not be such a significant feature. When pricing properties, the year and month of year could also be very important while the day number is practically irrelevant.

17.2.2 Representing Dates on Computers

The date splitting described above is a very good way to extract relevant information from a date. But is this the only way to represent dates as numbers?

Most computers use the **number of seconds** since a predefined point in time, also called the "epoch". For systems based on UNIX, this starting point is the 1st of January 1970. The UNIX timestamp for `2025-08-15T08:31:00` is the number of seconds since the 1st of January 1970: 1755239460.

The more you know. Spreadsheet systems like Excel and Google Sheets store dates as the number of days since an epoch, and the time of day as the decimal part of a number between 0 and 1. This way, a datetime can be stored as a decimal number. Using Google Sheets to store the example date, we get: 45 884.35. With 45884 being the number of days since the 1st of January 1900 and .35 representing `08:31`, very close to a third of the 24-hour day.

Why go through the trouble of learning about this alternative representation? It turns out that representing time as a continuous number like the number of days since a given date can help model trends.

Going back to the example of property prices over time, imagine that the average price evolves in the following way:

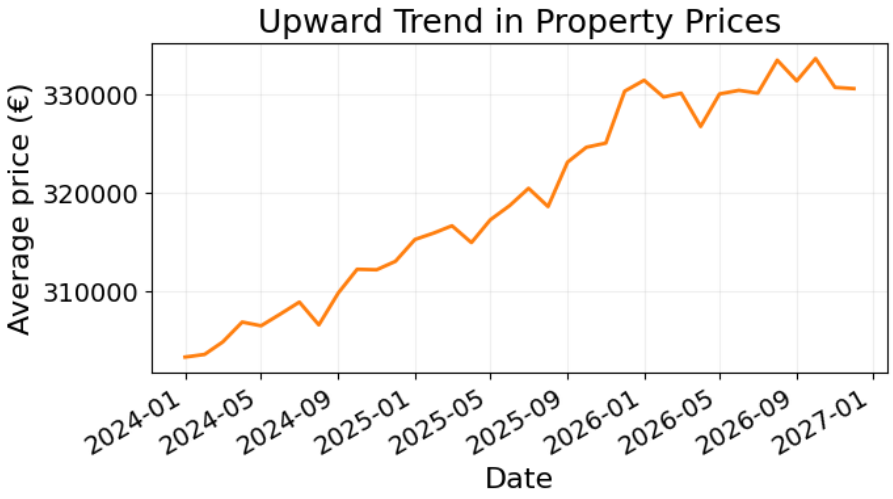


Figure 17.3: Upward linear trend

Figure code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter

np.random.seed(7)
n = 36
dates = pd.date_range(start="2024-01-01", periods=n, freq="MS")
t = np.arange(n)
baseline = 300000
trend = 1200 * t
noise = np.cumsum(np.random.normal(0, 2000, size=n))
y = baseline + trend + noise

fig, ax = plt.subplots(figsize=(7,4))
ax.plot(dates, y, color="tab:orange", linewidth=2)
ax.set_title("Upward Trend in Property Prices", fontsize=18)
ax.set_xlabel("Date", fontsize=16)
```

```
ax.set_ylabel("Average price (€)", fontsize=16)
ax.tick_params(axis="both", labelsize=14)
ax.grid(True, alpha=0.2)
ax.xaxis.set_major_formatter(DateFormatter("%Y-%m"))
fig.autofmt_xdate()
plt.tight_layout()
plt.show()
```

Having a feature that represents a **point in time**, can help determine the price of a property taking this upward trend into account. For example, a model could learn that the average price of property increases by 1% every 130 days.

Replacing the date features by their UNIX timestamp, we get the following table:

Property	Pricing date (UTC 09:00)	UNIX timestamp
A	2024-12-15 09:00:00	1734253200
B	2025-01-15 09:00:00	1736931600
C	2025-02-15 09:00:00	1739610000
D	2025-03-15 09:00:00	1742029200
E	2025-04-15 09:00:00	1744707600
F	2025-05-15 09:00:00	1747386000
G	2025-06-15 09:00:00	1749978000
H	2025-07-15 09:00:00	1752570000

17.3 Final Thoughts

It was about time to end this section. We covered two different ways to encode date features for a Machine Learning model:

- Splitting dates into their parts
- Represent dates as number of seconds or days since a given epoch

There is no optimal way to encode date features, it always depends on the problem at hand. For what you are trying to predict, how does time enter the picture?

That is all on alternative data types. Now that we can convert any data type to numbers, the next two chapters will explore scenarios in which data is missing.

Chapter 18

Dealing with Missing Numeric Data

Data can sometimes be missing. These missing points are often represented as null values.

Flat	Surface Area (sq m)	Distance to Centre (km)	Neighbour-hood	Energy Rating	Sell Price (K€)
A	85	4.2	Neukölln	B	420
B		2.5	Kreuzberg	A	610
C	95	6.1	Charlott.	C	390
D	70		Kreuzberg	D	370
E	110	1.2	Charlott.	B	700
F	60	5.0	Neukölln	F	310

This is problematic for most Machine Learning models, as they **cannot handle** missing values.

Note: many modern implementations of Machine Learning models, like LightGBM, do natively handle missing values using some of the methods shown in this chapter. This is due to smart implementation; the underlying models still cannot read missing values.

As shown in previous chapters, these Machine Learning models learn the relationship between inputs and outputs:

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

They use mathematical tricks (e.g., distance, data splitting) to learn the relationship between the features of each observation (e.g., property surface area) and the target variable (e.g., sell price).

The following chapter will explore different strategies to deal with missing values in a Machine Learning project.

18.1 Are All Missing Values Equal?

18.1.1 Starting with Why

Why is this data point missing? There could be many reasons which may inform our approach to missing data.

With data problems, it is a good idea to start with the **source**. Is there an issue in the data collection process? Could there be a problem with the database? How many observations is this affecting?

Before developing a strategy, it is critical to understand if there is an issue with the data source. Data source problems could affect many other applications, such as reporting or operations.

Missing values can have different meanings:

- Missing at Random: the observation is missing for an unknown reason, without any apparent pattern
- None: a missing transaction count could mean that no (or 0) transactions were made
- Not Applicable: Missing product sub-category could mean that the current product does not have a sub-category
- Missing Measurement: a missing temperature reading in some sensor data may mean that the measurement was skipped

Before coming up with a strategy, it is critical to understand why these values are missing.

18.1.2 Excluding Rows with Missing Values

Is the missing data point critical? In the context of property pricing with only two features (a very simplified example), the surface area is a critical data point. You may want to **exclude** the rows that have a key feature value missing.

Flat	Surface Area (sq m)	Distance to Centre (km)	Neighbour-hood	Energy Rating	Sell Price (K€)
A	85	4.2	Neukölln	B	420
B		2.5	Kreuzberg	A	610
C	95	6.1	Charlott.	C	390

Looking at the example above, you would remove property B from the training data.

18.1.3 Excluding Features with Missing Values

Property	Surface Area (sq m)	Distance to Centre (km)	Neighbourhood	Energy Rating	Sell Price (K€)	Years since Build
A	85	4.2	Neukölln	B	420	
B	120	2.5	Kreuzberg	A	610	8
C	95	6.1	Charlott.	C	390	
D	70	3.8	Kreuzberg	D	370	22
E	110	1.2	Charlott.	B	700	12
F	60	5.0	Neukölln	F	310	

In the above example, the column **Years since Build** has many missing values. It would make sense to either review the underlying data or remove it from the dataset.

Filtering out both rows and columns containing missing values may **reduce the amount of data available** to the model. For this reason, more sophisticated approaches are sometimes needed.

18.1.4 Imputation of Missing Numerical Values

For numerical features, missing values can be handled with a method called **imputation**. This fills missing values with another substitute value.

18.1.4.1 Imputation by 0

When the missing value means "None", like the missing number of transactions, it is a good idea to replace missing values with 0.

Imagine the data above also included "Balcony Area" in square meters. A missing value could mean "no balcony" and could safely be replaced by 0. Before doing so, ensure that a missing "Balcony Area" really means "no balcony". In general, these data processing methods rely on a solid understanding of the data and problem.

Property	Total Area (sq m)	Balcony Area (sq m)	Sell Price (K€)
A	95	12	450
B	55		280

Property	Total Area (sq m)	Balcony Area (sq m)	Sell Price (K€)
C	110	15	510

18.1.4.2 Imputation by the Mean

When values are **missing at random**, the mean or median could be good candidates for imputation.

In a dataset containing student heights:

Student	Height (cm)	Gender	Grade
A	170	M	10
B		F	10
C	165	F	9
D	180	M	11
E	172	M	
F	160	F	9

A good strategy to deal with missing heights is to impute student height using the **average height** of all students. With a large enough dataset, a more fine-grained approach could be used. You could, for example, impute the missing heights based on students with the same grade.

To impute the Height of student B, we calculate the average height over all students:

$$\text{Mean} = \frac{170 + 165 + 180 + 172 + 160}{5} = 169.4$$

The Height of student B would be replaced by 169.4.

Exercise 18.1. Using mean imputation to fill missing Grade values, what would be the Grade of student E?

18.1.4.3 Imputation by the Median

The **median** could also be a good choice of substitute value for imputation. The median of a series is the **middle number** in a sorted list. Taking the list: $[2, 3, 4, 5, 7, 9, 50]$ as example, the median is 5. In other words, it is the value such that at least 50% of the values are inferior or equal to. One advantage of the median is that it is not sensitive to outliers.

Mean, median and outliers

An **outlier** is a data point that is very different to the others

Considering the following series:

$[2, 3, 4, 5, 7, 9, 50]$

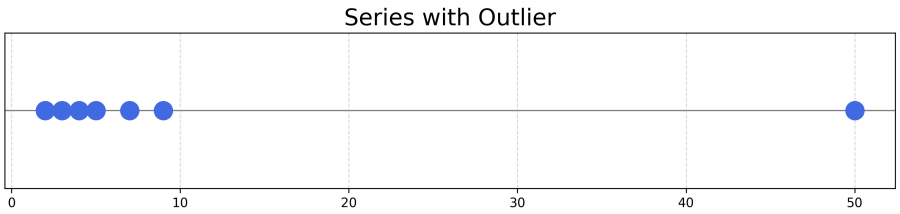


Figure 18.1: Series with outlier

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([2,3,4,5,7,9,50])

sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)  # horizontal line
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorder=2)

plt.yticks([])
plt.title('Series with Outlier', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```

The point at 50 is an outlier of this distribution, as it is very far from the others. The mean of the series is affected by this number:

Exercise 18.2. Calculate the mean and median of this series. Show that the mean is 11.43, and that the median is 5.

Exercise 18.3. Show that after removing the item 50, the mean is 5.0, and the median is 4.5.

As you may have observed, the mean varies widely, whereas the median remains stable.

Exercise 18.4. Find the median of the following series:

- $5, 7, 9, 12, 15$

- $1, 2, 2, 4, 8, 10, 13$

What happens when the length of the series is an even number?

Looking at the following series: $[1, 2, 3, 4]$, what would be the middle value? As the series contains an even number of items, there is no middle value.

By convention, the median of these series is the average between the two middle values. Here, the median would be $\frac{2+3}{2} = 2.5$.

Exercise 18.5. Find the median of the following series:

- $2, 3, 5, 8$
- $4, 6, 8, 10, 12, 14$

Going back to the student data:

Student	Height (cm)	Gender	Grade
A	170	M	10
B		F	10
C	165	F	9
D	180	M	11
E	172	M	
F	160	F	9

To impute the Height of student B with the median, we find the median. The median of the heights is 170, as it is the middle number in the sorted Height series: $160, 165, 170, 172, 180$.

Exercise 18.6. Impute the missing Grade of student E using median imputation

18.1.4.4 Imputation of Time-Series Data

Time-series data is data with a **time** dimension. As an example, the following data could be the temperature of a machine in a factory:

Time (s)	Temperature (°C)
1	18.0
2	18.5
3	19.1
4	20.0
5	20.8
6	
7	22.1
8	22.8
9	23.5
10	

Time (s)	Temperature (°C)
11	26.2

Time-series data is also often represented as a line chart:

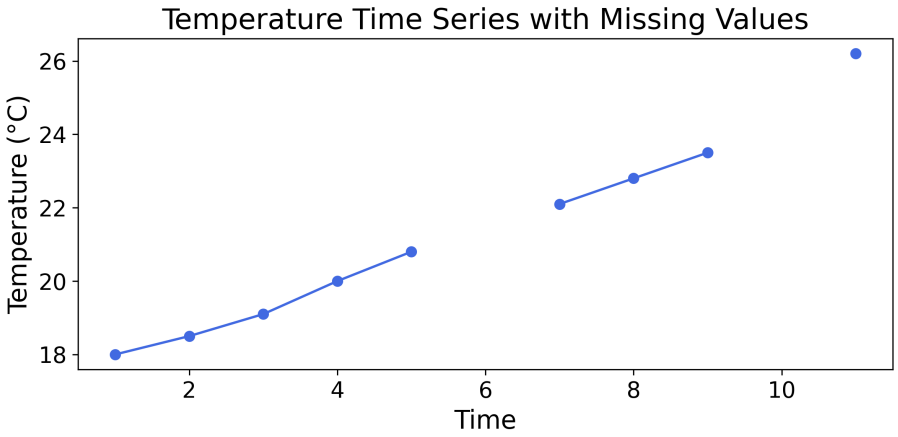


Figure 18.2: Temperature time series with missing values

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

time = np.arange(1, 12)
temp = [18.0, 18.5, 19.1, 20.0, 20.8, np.nan, 22.1, 22.8, 23.5, np.nan, 26.2]

plt.figure(figsize=(8,4))
plt.plot(time, temp, marker='o', linestyle='-', color='royalblue', label='Temperature')
plt.xlabel('Time', fontsize=16)
plt.ylabel('Temperature (°C)', fontsize=16)
plt.title('Temperature Time Series with Missing Values', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

But what is time?

Time can be defined as the continuous and apparently irreversible progress of existence. Just like “truth” or “space”, the definitions of the foundational aspects of experience can sometimes be disappointing.

18.1.4.4.1 Using the Mean and Median

Imputing null values with the mean and median could lead to some unintuitive results.

Exercise 18.7. Impute the null values of the following series using mean, then the median. To do so, show that the mean of the series is 21.22 and the median is 20.8.

Time (s)	Temperature (°C)
1	18.0
2	18.5
3	19.1
4	20.0
5	20.8
6	
7	22.1
8	22.8
9	23.5
10	
11	26.2

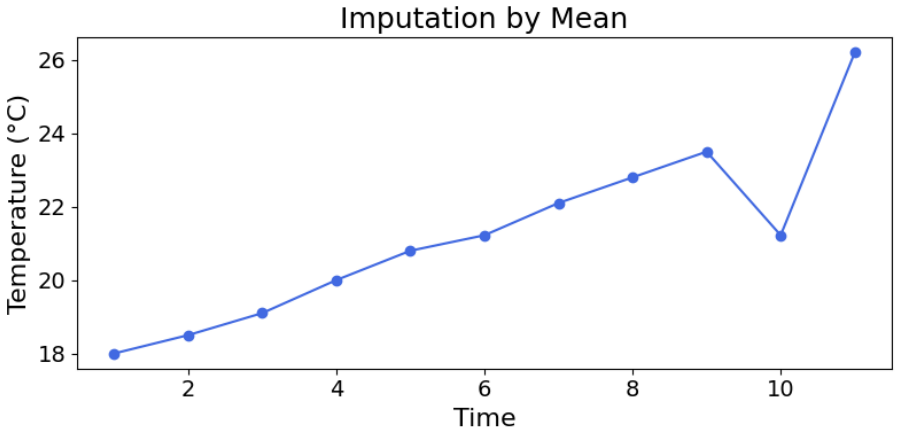


Figure 18.3: Imputation of missing values by mean

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

time = np.arange(1, 12)
```

```
temp = [18.0, 18.5, 19.1, 20.0, 20.8, np.nan, 22.1, 22.8, 23.5, np.nan, 26.2]
mean_temp = np.nanmean(temp)
temp_mean = [v if not np.isnan(v) else mean_temp for v in temp]

plt.figure(figsize=(8,4))
plt.plot(time, temp_mean, marker='o', linestyle='-', color='royalblue', label=f'Imputed by Mean (
plt.xlabel('Time', fontsize=16)
plt.ylabel('Temperature (°C)', fontsize=16)
plt.title('Imputation by Mean', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

As you can see, the mean imputation works very well for the observation 6. However, observation 10 is not imputed in a realistic way and introduces a jump in the series.

18.1.4.4.2 Alternative Methods

What would be a better method? In these time-series tasks, imputation by the **previous value** could be a good idea. This method is called **forward-fill**. It assumes that the value remains constant until a new measurement is taken.

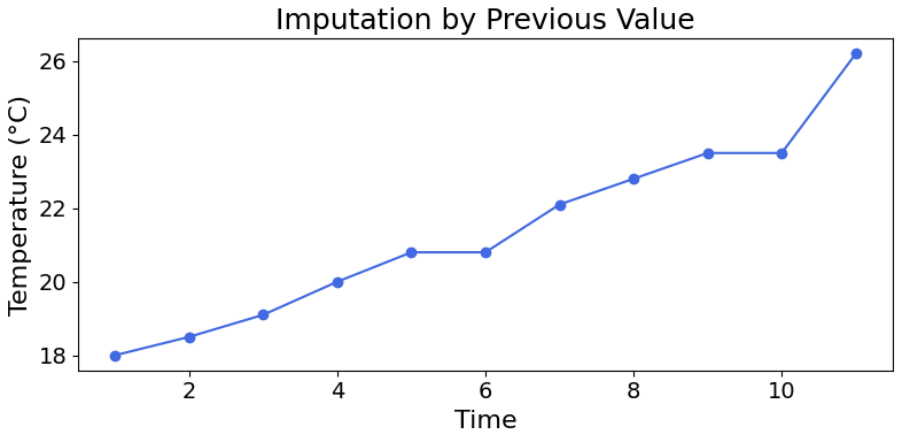


Figure 18.4: Imputation by previous value

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd

time = np.arange(1, 12)
temp = [18.0, 18.5, 19.1, 20.0, 20.8, np.nan, 22.1, 22.8, 23.5, np.nan, 26.2]
temp_series = pd.Series(temp)
temp_ffill = temp_series.ffill()

plt.figure(figsize=(8,4))
plt.plot(time, temp_ffill, marker='o', linestyle='-', color='royalblue', label='Imputed
plt.xlabel('Time', fontsize=16)
plt.ylabel('Temperature (°C)', fontsize=16)
plt.title('Imputation by Previous Value', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

The series looks more natural than with mean-imputation. Still, we can do better.

Another approach is to impute the missing values by computing the average of the two neighboring values. This method, often called **linear interpolation**, is particularly useful for data with a clear trend, as it smoothly connects the known data points.

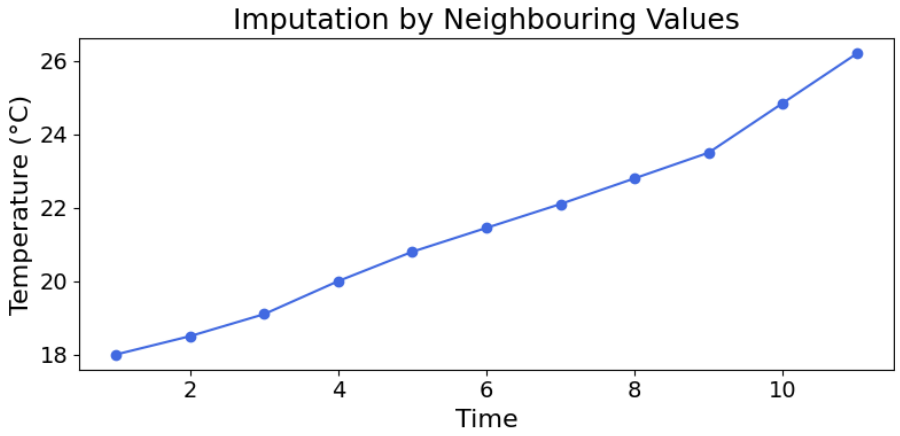


Figure 18.5: Imputation by neighbours

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

time = np.arange(1,12)
temp = [18.0, 18.5, 19.1, 20.0, 20.8, np.nan, 22.1, 22.8, 23.5, np.nan, 26.2]
temp_neigh = temp.copy()
for i in range(1, len(temp_neigh)-1):
    if np.isnan(temp_neigh[i]):
        temp_neigh[i] = (temp_neigh[i-1] + temp_neigh[i+1]) / 2

plt.figure(figsize=(8,4))
plt.plot(time, temp_neigh, marker='o', linestyle='-', color='royalblue', label='Imputed by Neighb
plt.xlabel('Time', fontsize=16)
plt.ylabel('Temperature (°C)', fontsize=16)
plt.title('Imputation by Neighbouring Values', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

That looks more like it!

18.1.4.5 Wrapping Up

There are many ways to impute numerical values, and no optimal solution. The choice comes down to the specific problem at hand. Remember that before coming up with an imputation strategy, it is critical to understand why the values are missing.

18.2 Information Leakage

As mentioned in the introduction to this chapter, data preprocessing is a step in which information leakage commonly occurs. Dealing with missing values is no exception.

In the process of filling numerical null values with the median or the average (or anything calculated from the data), it is important to calculate these numbers from **the training set only**.

Then, null values in the test set would be imputed with the metrics calculated with the training data.

Why is this the case? Because this is exactly what would happen as the model is used to predict **new observations**. When the model will predict the price of a new property, and that property has a missing value, how will it be imputed?

At this point, the only information available will be the training data. The

missing values will be imputed using **the mean or median of the training data**.

The purpose of generating predictions on the test set is to estimate model performance on unseen data. Null values in the test set should be treated in the same way as missing values in unseen data.

In Machine Learning practice, these preprocessing methods are said to be **fitted** on the training set and applied to the test set. This fitting is the calculation of descriptive statistics.

18.3 Final Thoughts

This chapter walked through handling missing numerical values with the following steps:

- Understand why the values are missing
- Develop a strategy: filtering or different types of imputation
- Fit the imputation method to the training data
- Impute null values in the test set using training data statistics

It is important to remember to avoid information leakage in null value handling. This can be done by a clear separation of the training and test set, before data preprocessing.

Looking at the previous example, what if the "Neighbourhood" column was missing? The following section will explore how to handle missing **categorical** values.

18.4 Solutions

Solution 18.1. Exercise 18.1

Student	Height (cm)	Gender	Grade
A	170	M	10
B		F	10
C	165	F	9
D	180	M	11
E	172	M	
F	160	F	9

The average Grade is:

$$\text{Mean} = \frac{10 + 10 + 9 + 11 + 9}{5} = \frac{49}{50} = 9.8$$

The missing Grade of student E would be replaced by 9.8.

Solution 18.2. Exercise 18.2

Series: $[2, 3, 4, 5, 7, 9, 50]$

$$\text{Mean} = \frac{2+3+4+5+7+9+50}{7} = \frac{80}{7} \approx 11.43$$

The median is the middle number of the sorted series. Here, 5.

Solution 18.3. Exercise 18.3

Series: $[2, 3, 4, 5, 7, 9]$

$$\text{Mean} = \frac{2+3+4+5+7+9}{6} = \frac{30}{6} = 5$$

As the series now contains an even number of observations, the median is the average of the two middle numbers, here $\frac{4+5}{2} = 4.5$.

Solution 18.4. Exercise 18.4

- $5, 7, 9, 12, 15, \text{Median} = 9$
- $1, 2, 2, 4, 8, 10, 13, \text{Median} = 4$

Solution 18.5. Exercise 18.5

- $2, 3, 5, 8, \text{Median} = \frac{3+5}{2} = 4$
- $4, 6, 8, 10, 12, 14, \text{Median} = \frac{8+10}{2} = 9$

Solution 18.6. Exercise 18.6

The median of the Grade series is the middle number of the sorted series: $9, 9, 10, 10, 11$. Here, it is 10. The grade of student E would therefore be filled with 10.

Solution 18.7. Exercise 18.7

For imputation, we need to compute both the mean and median:

$$\text{Mean} = \frac{18+18.5+19.1+20+20.8+22.1+22.8+23.5+26.2}{9} = \frac{191}{9} = 21.22$$

The median is the middle number of the sorted series: $18, 18.5, 19.1, 20.0, 20.8, 22.1, 22.8, 23.5, 26.2$, here, 20.8

Time	Temperature (°C)	Temperature (Mean Imputation)	Temperature (Median Imputation)
1	18.0	18.0	18.0
2	18.5	18.5	18.5

Time	Temperature (°C)	Temperature (Mean Imputation)	Temperature (Median Imputation)
3	19.1	19.1	19.1
4	20.0	20.0	20.0
5	20.8	20.8	20.8
6		21.22	20.8
7	22.1	22.1	22.1
8	22.8	22.8	22.8
9	23.5	23.5	23.5
10		21.22	20.8
11	26.2	26.2	26.2

Chapter 19

Dealing with Missing Categorical Data

Categorical values can also be missing. As a reminder, categorical features represent **groups** or **categories**. In the Berlin property pricing example, the neighbourhood of the property (e.g., "Neukölln", "Kreuzberg" or "Charlottenburg") is a categorical feature. As neighbourhood is such an important feature in property pricing, it may make sense to filter out observations for which it is missing.

Just like numerical features, there are many reasons why a categorical value may be missing:

- Missing at Random: the observation is missing for an unknown reason, without any apparent pattern
- Not Collected: optional question in a survey
- Not Applicable: as a man donating blood, I leave the answer to the question "are you pregnant?" blank, as this is not applicable
- Missing Measurement: a missing energy efficiency grade reading could mean that the property still needs an inspection

It is important to understand why data may be missing before coming up with an imputation strategy.

19.1 Excluding Rows with Missing Observations

If a missing value is critical to the prediction task, filtering out rows with missing values is the safest strategy. You would not want to price a property without knowing its neighbourhood or its post code.

19.2 Excluding Columns with Missing Observations

If a column contains many missing values, removing the column or reviewing the data processing script should be the preferred strategy. This is similar to the method used for numerical missing values.

19.3 Excluding Rows with Missing Observations

Likewise, if the missing categorical value is critical to the prediction task, excluding the observation is the safest approach. Could you consider a property without knowing if it is a house or a flat? Or without knowing its neighbourhood?

19.4 Creating a new Null Category

In other cases, all missing values could be replaced by a "missing" category and treated as another value for that categorical variable. Let's make this more concrete with this example:

Property	Surface Area (sq m)	Distance to Centre (km)	Neighbourhood	Energy Rating	Outdoor Space	Sell Price (K€)	Years since Build
A	85	4.2	Neukölln	B	Balcony	420	
B	120	2.5	Kreuzberg	A	Terrace	610	8
C	95	6.1	Charlott.	C		390	
D	70	3.8	Kreuzberg	D	Garden	370	22
E	110	1.2	Charlott.	B	Balcony	700	12
F	60	5.0	Neukölln	F		310	

Here, some of the "Outdoor Space" values are missing. As mentioned above, the missing values can be replaced by an "Unknown" value.

Property	Surface Area (sq m)	Distance to Centre (km)	Neighbourhood	Energy Rating	Outdoor Space	Sell Price (K€)	Years since Build
A	85	4.2	Neukölln	B	Balcony	420	
B	120	2.5	Kreuzberg	A	Terrace	610	8
C	95	6.1	Charlott.	C	**Unknown**	390	
D	70	3.8	Kreuzberg	D	Garden	370	22
E	110	1.2	Charlott.	B	Balcony	700	12
F	60	5.0	Neukölln	F	**Unknown**	310	

This value can then be treated in the same way as all the other categorical values, with methods like One-Hot Encoding or Target Encoding.

19.5 Information Leakage

Just like categorical variable encoding, handling missing categorical values can create information leakage.

This process should be done on the training data only, and applied to the test set. Any new category seen in the test set only should be either excluded or labelled as missing.

19.6 Final Thoughts

This chapter walked through handling missing values with the following steps:

- Understand why the values are missing
- Explore whether filtering is needed
- Flag missing values with a new categorical value

It is important to remember to avoid information leakage in null value handling. This can be done by a clear separation of the training and test set, before data preprocessing.

The next chapter will go back to numerical data and explore the issue of scaling.

Chapter 20

Numerical Feature Scaling

Lists of numbers come in all shapes and sizes. But why should we care?

Many Machine Learning algorithms operate on data as **vectors** or **points in space**. KNN generates predictions based on the **distance** or similarities between observations. Decision Trees split the feature space to separate data from different classes or labels.

Thinking of nearest neighbours, feature scaling does matter. Let's imagine you are building a model to predict property prices, using two features:

- Surface Area in m^2
- Number of rooms

Example properties in the training data include:

Property	Size (m^2)	Number of rooms	Price (k €)
A	60	2	320
B	80	3	400
C	120	5	350
D	70	2	310
E	150	6	600
F	90	4	330

When plotting this data over a two-dimensional space on the interval [0, 200], we get:

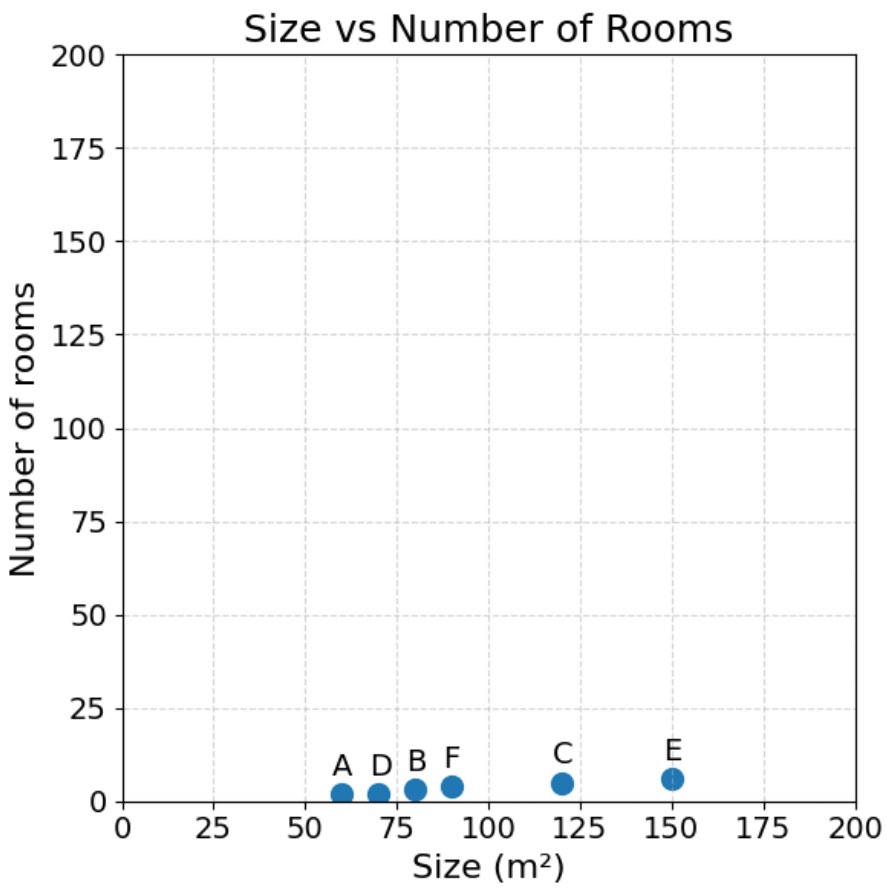


Figure 20.1: Property data without scaling

Figure code

```
import matplotlib.pyplot as plt

sizes = [60, 80, 120, 70, 150, 90]
rooms = [2, 3, 5, 2, 6, 4]
labels = ['A', 'B', 'C', 'D', 'E', 'F']

plt.figure(figsize=(6,6))
plt.scatter(sizes, rooms, s=100)
for i, label in enumerate(labels):
    plt.text(sizes[i]-2.5, rooms[i]+5, label, fontsize=14)
plt.xlim(0, 200)
plt.ylim(0, 200)
```

```
plt.xlabel('Size (m²)', fontsize=16)
plt.ylabel('Number of rooms', fontsize=16)
plt.title('Properties: Size vs Number of Rooms (0-200 scale)', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

As you can see, most of the variation is generated by the **surface area**. This is why the data appears so compressed.

The dominance of surface area over the number of rooms can also be seen when calculating the Euclidean Distance between properties.

Distance between property A and B:

$$\begin{aligned} d_{AB} &= \sqrt{(80-60)^2 + (3-2)^2} \\ &= \sqrt{20^2 + 1^2} \\ &= \sqrt{400+1} \quad \text{surface area dwarfs the difference in number of rooms} \\ &= \sqrt{401} \\ &\approx 20.02 \end{aligned}$$

Distance between property A and C:

$$\begin{aligned} d_{AC} &= \sqrt{(120-60)^2 + (5-2)^2} \\ &= \sqrt{60^2 + 3^2} \\ &= \sqrt{3600+9} \quad \text{here again} \\ &= \sqrt{3609} \\ &\approx 60.08 \end{aligned}$$

As you can see, in both cases, the distance between two properties is nearly fully determined by the surface area difference. The distance between number of rooms is dwarfed by the difference in surface area. We could get a more even distribution by rescaling the axes:

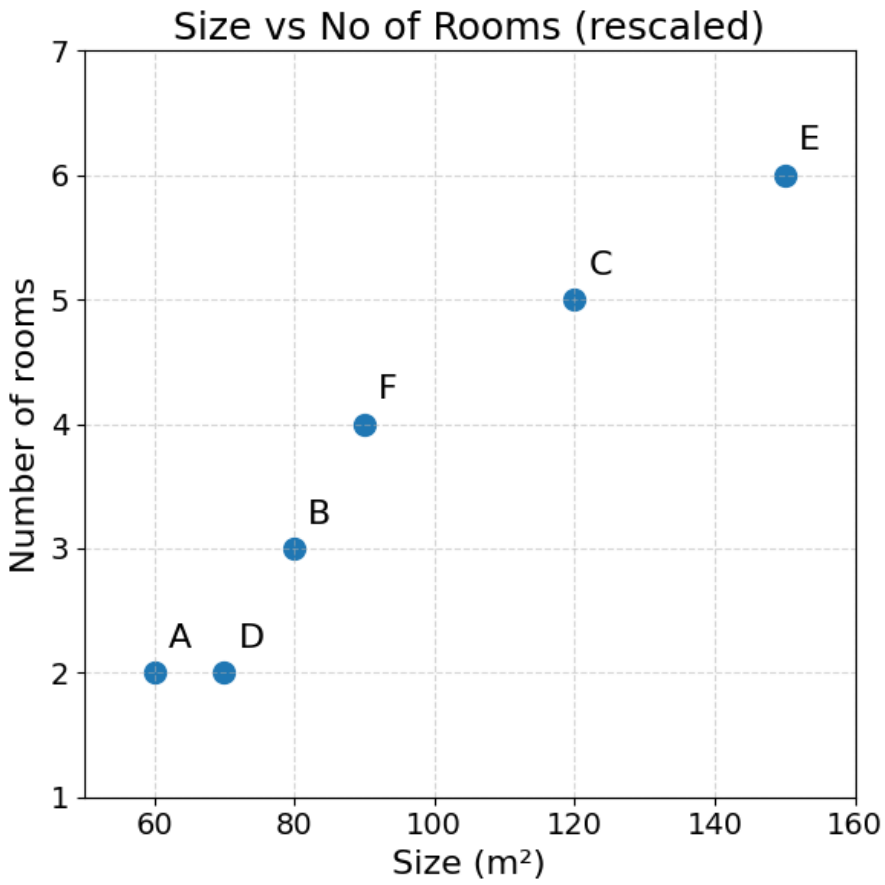


Figure 20.2: Property Data Rescaled

Figure code

```
plt.figure(figsize=(6,6))
plt.scatter(sizes, rooms, s=100)
for i, label in enumerate(labels):
    plt.text(sizes[i]+2, rooms[i]+0.2, label, fontsize=16)
plt.xlim(50, 160)
plt.ylim(1, 7)
plt.xlabel('Size (m²)', fontsize=16)
plt.ylabel('Number of rooms', fontsize=16)
plt.title('Properties: Size vs Number of Rooms (rescaled axes)', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, linestyle='--', alpha=0.5)
```

```
plt.tight_layout()
plt.show()
```

While rescaling the axes is a better way to **visualise** the data, it does not solve the problem in calculating the distance between two observations.

20.1 The Mechanics

To do so, we need to **scale** both features so that they (roughly) have the same **range, mean and variance**:

- Range is the distance between the minimum and the maximum of a series.
- Mean is the average of a series
- Variance can have many definitions, for now, let's define it as the degree to which **numbers fluctuate**

As an example, the surface area column has a higher variance than the number of rooms column. The numbers **fluctuate more** from one to the other. As a consequence, surface area has a larger impact than the number of rooms on the distance function.

What if we could transform both the number of rooms and the surface area to series of number between 0 and 1? This would solve the issue of distance calculation. The following section will explore a way to do this.

20.2 Min-Max Scaling

Let's imagine the following series of numbers: $\{0, 30, 40, 60, 90, 100\}$

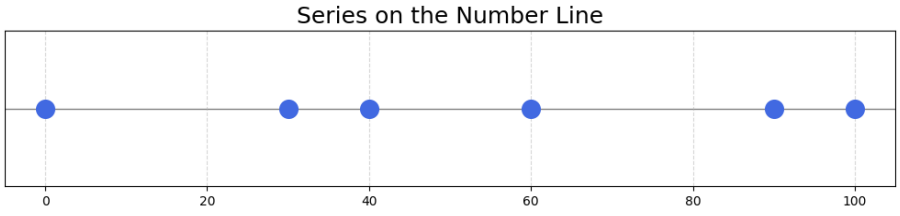


Figure 20.3: Series 0-100

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([0, 30, 40, 60, 90, 100])
```

```
# Sort for better label spacing
sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)  # horizontal line
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorde

plt.yticks([])
plt.title('Series on the Number Line', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

A simple way to get these numbers in the 0-1 interval would be to divide all of them by 100:

Value	Scaled
0	0.00
30	0.30
40	0.40
60	0.60
90	0.90
100	1.00

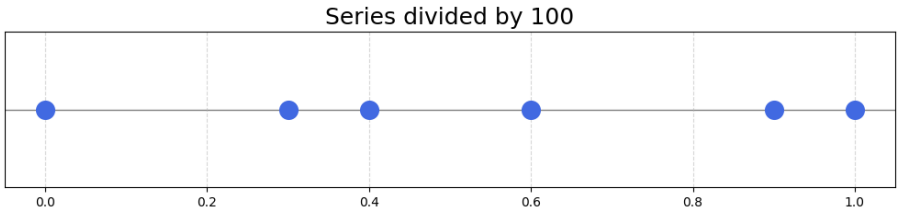


Figure 20.4: Series scaled to 0-1

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([0, 30, 40, 60, 90, 100])/100

sort_idx = np.argsort(series1)
```

```
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorder=2)

plt.yticks([])
plt.title('Series divided by 100', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```

If the original series has numbers going from 0-150: $\{0, 10, 60, 70, 120, 150\}$

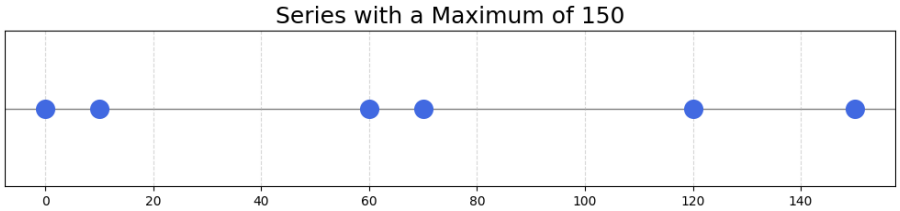


Figure 20.5: Series 0-150

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([0, 10, 60, 70, 120, 150])

sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorder=2)

plt.yticks([])
plt.title('Series with a Maximum of 150', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```

We could also get it back to the 0-1 interval by dividing all numbers by 150, the **maximum** of the series:

Value	Scaled
0	0.00
10	0.07
60	0.40
70	0.47
120	0.80
150	1.00

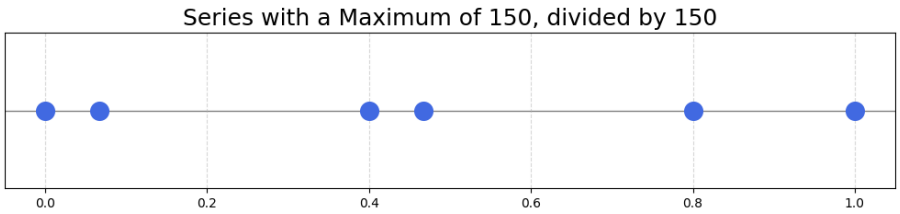


Figure 20.6: Series 0-1 (by 150)

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([0, 10, 60, 70, 120, 150])/150

sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)  # horizontal line
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorde

plt.yticks([])
plt.title('Series with a Maximum of 150, divided by 150', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```

This is a good start. For series ranging from 0 to a given number, we can divide all numbers in the series by their maximum. This scales the series to the range 0-1.

Now, what if the numbers in the series range from 50 to 250?

Series: $\{50, 90, 120, 170, 220, 250\}$

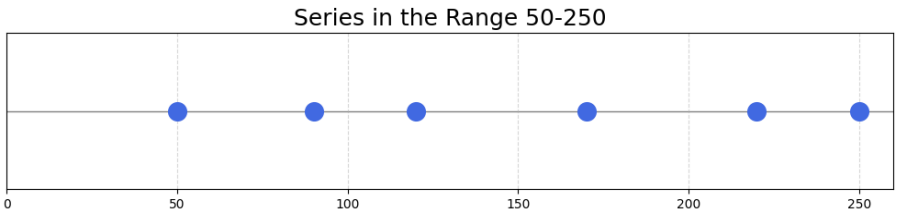


Figure 20.7: Series 50-250

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([50, 90, 120, 170, 220, 250])

# Sort for better label spacing
sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)  # horizontal line
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorder=2)

plt.yticks([])
plt.title('Series in the Range 50-250', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.xlim(0,260)

plt.tight_layout()
plt.show()
```

Dividing the numbers of the series by the maximum would not make the full use of the 0-1 interval:

Value	Scaled (÷250)
50	0.20
90	0.36
120	0.48
170	0.68
220	0.88

Value	Scaled (÷250)
250	1.00

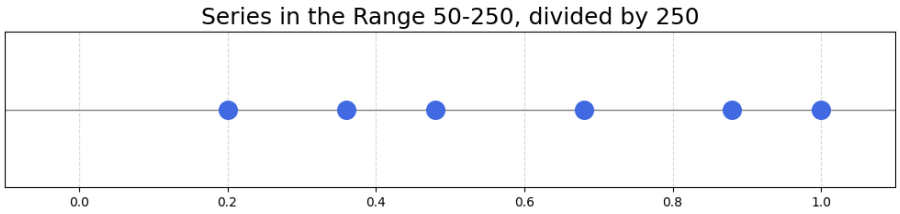


Figure 20.8: Series 0.2-1 (by 250)

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([50, 90, 120, 170, 220, 250])/250

sort_idx = np.argsort(series1)
areas_sorted = series1[sort_idx]

plt.figure(figsize=(10, 2.5))
plt.axhline(0, color='grey', linewidth=1, zorder=1)
plt.scatter(areas_sorted, np.zeros_like(areas_sorted), s=200, color='royalblue', zorde

plt.yticks([])
plt.title('Series in the Range 50-250, divided by 250', fontsize=18)
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.xlim(-0.1,1.1)

plt.tight_layout()
plt.show()
```

The minimum achieved there is $50/250 = 0.2$. This means that 20% of the allowed range will be left **empty**, a suboptimal outcome. How could this problem be solved?

In the end, we want the minimum of the series to be mapped to 0, and the maximum to 1. To achieve this, we can use the following formula:

$$\text{Scaled Value} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

With:

- x the value to be scaled
- $x_{\min}$ and $x_{\max}$ the minimum and maximum of the series

Let's take this formula part by part:

- The numerator: $x - x_{\min}$, computes the **distance** between the value and the minimum
- The denominator: $x_{\max} - x_{\min}$, divides this distance by the **largest distance** in the series, the distance between the maximum and minimum values

This formula will be maximised when the value is the maximum of the series, original value $= x_{\max}$:

$$\text{Scaled Value of Max} = \frac{x_{\max} - x_{\min}}{x_{\max}} - x_{\min} = 1$$

And be minimised when the value is the minimum of the series, original value $= x_{\min}$:

$$\text{Scaled Value of Min} = \frac{x_{\min} - x_{\min}}{x_{\max} - x_{\min}} = 0$$

With this formula, any series of numbers can be mapped to the range $[0, 1]$, solving the problems.

The following table shows its application to the example series above:

Value	Scaled (÷250)	Scaled (MinMax)
50	0.20	0.00
90	0.36	0.20
120	0.48	0.35
170	0.68	0.60
220	0.88	0.85
250	1.00	1.00

Exercise 20.1. Use Min-Max Scaling to map both surface area and number of rooms to the range $[0, 1]$

Property	Size (m²)	Number of rooms
A	60	2
B	80	3
C	120	5
D	70	2

Property	Size (m^2)	Number of rooms
E	150	6
F	90	4

20.3 Standardisation

The following section will get a bit more mathematical and can be skipped (next section).

Min-Max Scaling is not the only way to preprocess a series of numbers in different scales. Another approach is called **Standardisation**. The idea of standardisation is to make the **distributions of the series** similar. The distribution is a representation of how frequently the values of a variable occur.

Let's make this more concrete with an example:

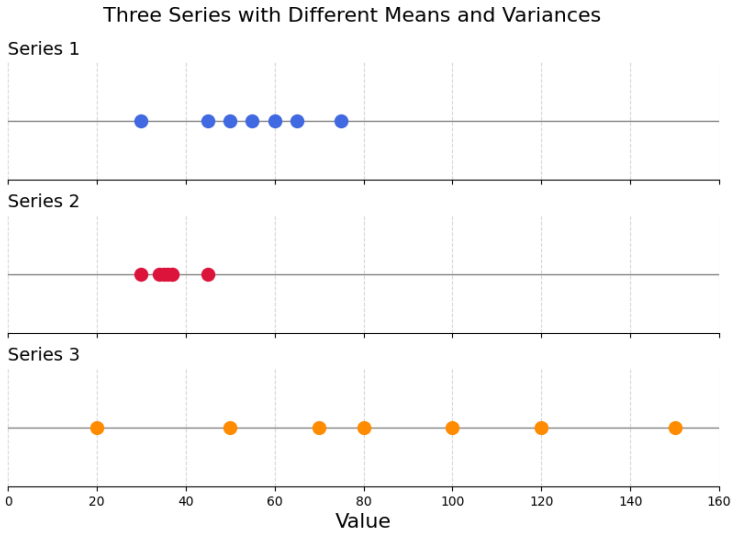


Figure 20.9: Three series with different means and variances

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([30, 55, 60, 50, 45, 75, 65])
series2 = np.array([35, 37, 36, 34, 45, 30, 37])
series3 = np.array([ 20,  50, 100,  80,  70, 120, 150])
```

```
series = [series1, series2, series3]
labels = ['Series 1', 'Series 2', 'Series 3']
colors = ['royalblue', 'crimson', 'darkorange']

fig, axes = plt.subplots(3, 1, figsize=(10, 6), sharex=True, gridspec_kw={'hspace': 0.3})

all_data = np.concatenate(series)
xmin, xmax = 0, all_data.max() + 10

for i, ax in enumerate(axes):
    y = np.zeros_like(series[i])
    ax.scatter(series[i], y, s=100, color=colors[i], zorder=2)
    ax.axhline(0, color='grey', linewidth=1, zorder=1)
    ax.set_yticks([])
    ax.set_xlim(xmin, xmax)
    ax.set_title(labels[i], fontsize=14, loc='left')
    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    if i < 2:
        ax.tick_params(labelbottom=False)
    else:
        ax.set_xlabel('Value', fontsize=16)
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

fig.suptitle('Three Series with Different Means and Variances', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

These series have very different distributions. The difference of **scale** would have a negative impact on Machine Learning algorithms like KNN.

Why can we say that these two series have different distributions?

- They do not have the same **mean** or **centre**
- They do not have the same **spread** or **variance** around their respective means

These are the two discrepancies that Standardisation aims to correct.

20.3.1 Describing a series

Before we start, it is important to define some important concepts of **descriptive statistics**.

Breaking this term down:

- Statistics is a branch of mathematics focussing on the collection and analysis of **collections of data**
- Descriptive statistics is a branch of statistics concerned with the **summarisation and description** of a collection of data

The Minimum and Maximum of a series are descriptive statistics, representing the lowest and highest number of a collection of numbers.

This section will introduce further concepts in descriptive statistics:

- Mean
- Variance
- Standard Deviation

If you are already familiar with these, feel free to skip to the next section.

20.3.1.1 Mean

The mean represents the "centre" of a collection of numbers. There are different types of means in mathematics. This book will focus on the arithmetic mean, also referred to as the "average". It is noted $\bar{x}$ or μ.

Other kinds of mean

There are other types of means, used to avearge different types of values. They all have different properties, advantages and drawbacks

Geometric mean: $\left(\prod_{i=1}^{n} x_i\right)^{1/n}$. It is useful for averaging ratios, growth rates, or percentages. The scary $\prod_{i=1}^{n}$ is the letter pi in capital letter, representing a product. $\prod_{i=1}^{n} i = 1 \cdot 2 \cdot \cdots \cdot n$

Using geometric means, a single 0 in the series will make the mean 0. It is still widely used in economics to average multiplicative phenomena like rates of growth.

Harmonic mean: $\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{x_i}\right)^{-1}$. It is useful for rates, such as model Precision or Recall (Model Evaluation chapter).

The harmonic mean of model Precision and Recall is called the F1-score. It is commonly used to find models that have a good trade-off of both metrics.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The arithmetic mean of a series of n observations, $\{x_1, x_2, \dots, x_n\}$ is computed with the following formula:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

Using the Σ summation operator:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Both notations are equivalent. If you find the Σ notation intimidating, refer to the Distance chapter for an intuitive explanation.

If you ever calculated your average grade at school, you probably used an arithmetic mean or a weighted average.

As an exmaple, these two series have different means:

Series 1	Series 2
1	4
2	8
4	10
5	12
8	16

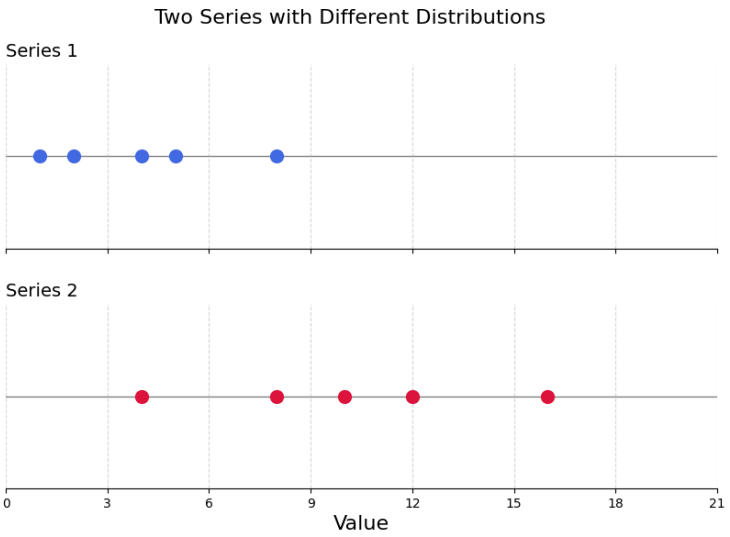


Figure 20.10: Two series with different means

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from matplotlib.ticker import MaxNLocator

series1 = np.array([7, 8, 10, 11, 14]) - 6
series2 = [4, 8, 10, 12, 16]

series = [series1, series2]
labels = ['Series 1', 'Series 2']
colors = ['royalblue', 'crimson']

fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True, gridspec_kw={'hspace': 0.3

all_data = np.concatenate(series)
xmin, xmax = 0, all_data.max() + 5

for i, ax in enumerate(axes):
    y = np.zeros_like(series[i])
    ax.scatter(series[i], y, s=100, color=colors[i], zorder=2)
    ax.axhline(0, color='grey', linewidth=1, zorder=1)
    ax.set_yticks([])
    ax.set_xlim(xmin, xmax)
    ax.set_title(labels[i], fontsize=14, loc='left')
    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    if i < 1:
        ax.tick_params(labelbottom=False)
    else:
        ax.set_xlabel('Value', fontsize=16)
        ax.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

fig.suptitle('Two Series with Different Distributions', fontsize=16)
plt.tight_layout()
plt.show()
```

The mean of the first series can be calculated with the formula shown above:

$$\text{Mean}_{x_1} = \frac{1+2+4+5+8}{5} = 4$$

Exercise 20.2. Show that the mean of the second series is equal to 10.

20.3.1.2 Variance and Standard Deviation

The following two series have the same mean, and yet, they look very different:

Series 1	Series 2
7	4
8	8
10	10
11	12
14	16

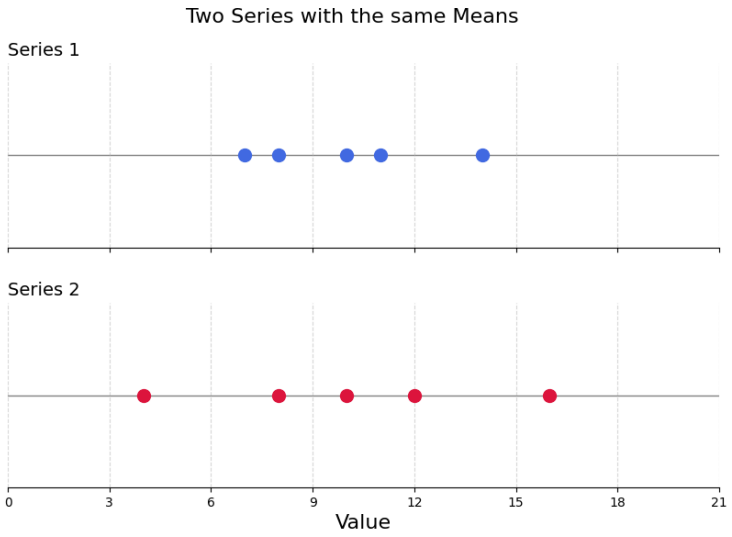


Figure 20.11: Two series, same mean, different variance

Figure code

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.ticker import MaxNLocator

series1 = [7, 8, 10, 11, 14]
series2 = [4, 8, 10, 12, 16]

series = [series1, series2]
labels = ['Series 1', 'Series 2']
colors = ['royalblue', 'crimson']

fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True, gridspec_kw={'hspace': 0.3})

all_data = np.concatenate(series)
xmin, xmax = 0, all_data.max() + 5
```

```
for i, ax in enumerate(axes):
    y = np.zeros_like(series[i])
    ax.scatter(series[i], y, s=100, color=colors[i], zorder=2)
    ax.axhline(0, color='grey', linewidth=1, zorder=1)
    ax.set_yticks([])
    ax.set_xlim(xmin, xmax)
    ax.set_title(labels[i], fontsize=14, loc='left')
    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    if i < 1:
        ax.tick_params(labelbottom=False)
    else:
        ax.set_xlabel('Value', fontsize=16)
        ax.xaxis.set_major_locator(MaxNLocator(integer=True))
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

fig.suptitle('Two Series with the same Means', fontsize=16)
plt.tight_layout()
plt.show()
```

The main difference between these two series is that the second has a much greater variance than the first. Variance refers to the **spread** of the individual values around the mean.

How would you quantify the spread of values around the mean? How would you summarise this spread in a single number?

Hint: You could refer to the Regression Model Evaluation chapter.

The distance between an individual value (x) and the mean ($\bar{x}$) can be calculated as follows:

$$x - \bar{x}$$

This works for a single value. Now, how would you aggregate these distances? Remember that positive and negative distances should **not** cancel out.

If you thought of using the **absolute value** or **squared differences**, well done! This is the right intuition. The variance is the average square difference between individual values and the mean.

$$\text{Var}(x) = \frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n}$$

Sigma notation:

$$\text{Var}(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Does this remind you of a formula? This is very similar to the **Mean Squared Error** (MSE), the average squared difference between model predictions and true labels. The variance is in a sense the MSE of the average.

If a model kept predicting the average for all observations, its Mean Squared Error would be the variance!

The variance for the first series can be computed as follows:

$$\begin{aligned}
\bar{x}_1 &= \frac{7+8+10+11+14}{5} = 10 \\
\text{Var}(x_1) &= \frac{(7-10)^2 + (8-10)^2 + (10-10)^2}{5} \\
&\quad + \frac{(11-10)^2 + (14-10)^2}{5} \\
&= \frac{9+4+0+1+16}{5} = \frac{30}{5} = 6
\end{aligned}$$

Exercise 20.3. Compute the variance of the second series. Prove that it is 16.

Is there something strange with these variance numbers? Similar to the Mean Squared Errors, they are much larger than the scale of the original series.

How would you solve this problem? Here again, the Regression Model Evaluation chapter could be helpful. Just like the Mean Squared Error and the Root Mean Squared Error, you could simply take the **square root** of the variance.

This number gives you a much more interpretable idea of the typical spread between individual values and the mean of a series. This measure is called the Standard Deviation, commonly noted σ.

$$\sigma = \sqrt{\text{Var}(x)}$$

This standard deviation measures the **average spread** from the mean in a series.

Are descriptive statistics this descriptive?

As surprising as it may seem, all the datasets have the same mean, variance and standard deviation for both x and y. This is the famous Ascombe Quartet.

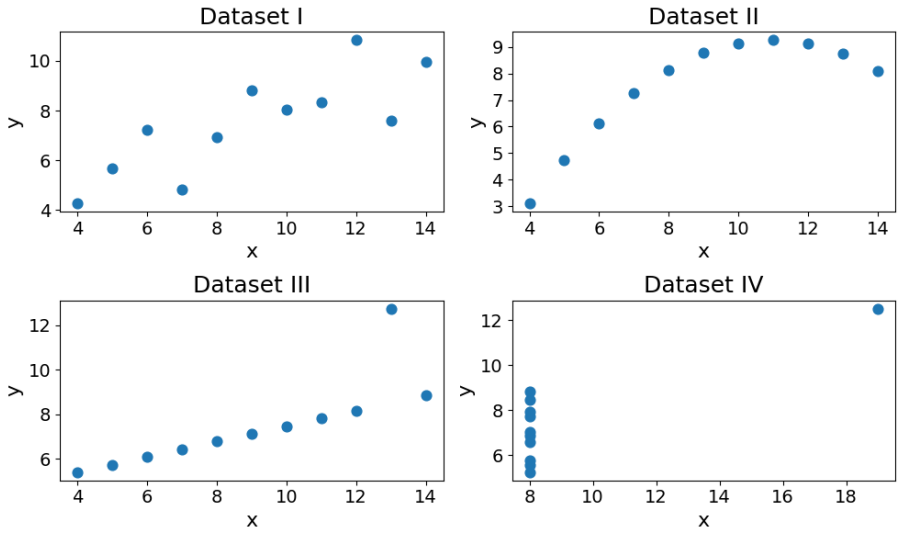


Figure 20.12: Anscombe's quartet

Figure code

```
import seaborn as sns
from seaborn import load_dataset
import matplotlib.pyplot as plt

anscombe = sns.load_dataset("anscombe")
plt.figure(figsize=(10,6))
for i, group in enumerate(['I', 'II', 'III', 'IV']):
    subset = anscombe[anscombe['dataset'] == group]
    plt.subplot(2,2,i+1)
    plt.scatter(subset['x'], subset['y'], s=60)
    plt.title(f"Dataset {group}", fontsize=18)
    plt.xlabel('x', fontsize=16)
    plt.ylabel('y', fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
plt.tight_layout()
plt.show()
```

Descriptive statistics for Anscombe's quartet:

Dataset	Mean x	Mean y	Var x	Var y	Std x	Std y
I	9	7.5	10	3.75	3.16	1.94
II	9	7.5	10	3.75	3.16	1.94

Dataset	Mean x	Mean y	Var x	Var y	Std x	Std y
III	9	7.5	10	3.75	3.16	1.94
IV	9	7.5	10	3.75	3.16	1.94

This statistical paradox is good reminder that descriptive statistics are remain a **reduction of reality**.

20.3.2 Standardising series with Mean and Standard Deviation

20.3.2.1 Mean Centre

Going back to the three example series:

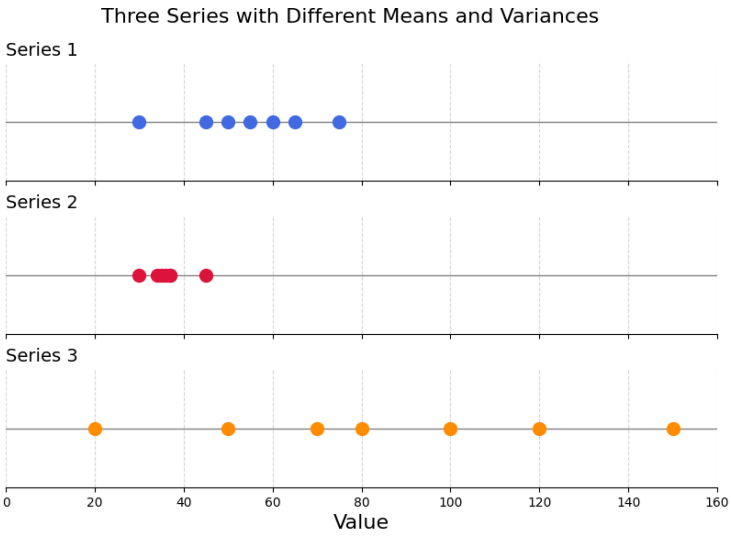


Figure 20.13: Three series with different means and variances

How could these be standardised so that they have the same mean and spread?

First, by subtracting the mean to each observation, the two series could both be centred around 0.

$$x_{\text{mean centred}} = x - \bar{x}$$

A good first step:

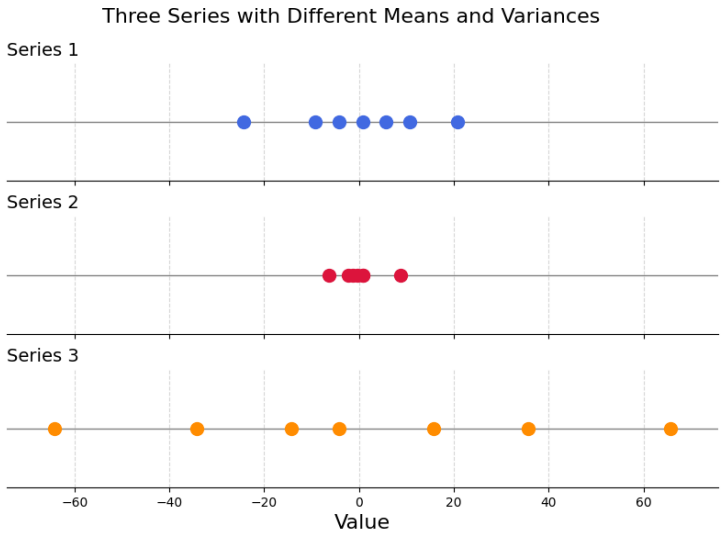


Figure 20.14: Three series, mean centred

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([30, 55, 60, 50, 45, 75, 65])
series2 = np.array([35, 37, 36, 34, 45, 30, 37])
series3 = np.array([ 20,  50, 100,  80,  70, 120, 150])

series = [series1, series2, series3]
labels = ['Series 1', 'Series 2', 'Series 3']
colors = ['royalblue', 'crimson', 'darkorange']

for idx, ind_series in enumerate(series):
    mean = np.mean(ind_series)
    series[idx] = ind_series.astype(float) - mean

fig, axes = plt.subplots(3, 1, figsize=(10, 6), sharex=True, gridspec_kw={'hspace': 0.3

# Find global min/max for consistent x-axis
all_data = np.concatenate(series)
xmin, xmax = all_data.min() - 10, all_data.max() + 10

for i, ax in enumerate(axes):
    y = np.zeros_like(series[i])
```

```
    ax.scatter(series[i], y, s=100, color=colors[i], zorder=2)
    ax.axhline(0, color='grey', linewidth=1, zorder=1)
    ax.set_yticks([])
    ax.set_xlim(xmin, xmax)
    ax.set_title(labels[i], fontsize=14, loc='left')
    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    if i < 2:
        ax.tick_params(labelbottom=False)
    else:
        ax.set_xlabel('Value', fontsize=16)
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

fig.suptitle('Three Series, Mean-Centred', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

But the three series still have widely different spreads.

20.3.2.2 Variance Scaling

To make sure both series have the same **variance**, or spread around the mean, you could simply **divide** all numbers of the resulting series by the standard deviation:

$$x_{\text{standardised}} = \frac{x - \bar{x}}{\sigma}$$

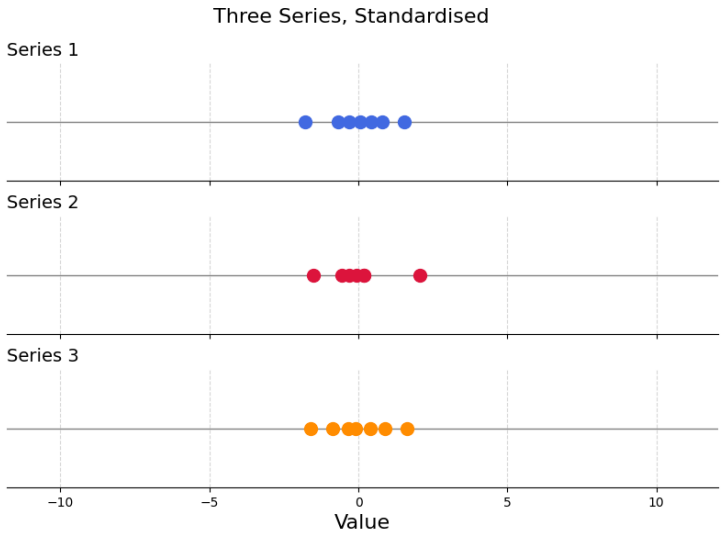


Figure 20.15: Three series, standardised

Figure code

```
import matplotlib.pyplot as plt
import numpy as np

series1 = np.array([30, 55, 60, 50, 45, 75, 65])
series2 = np.array([35, 37, 36, 34, 45, 30, 37])
series3 = np.array([ 20,  50, 100,  80,  70, 120, 150])

series = [series1, series2, series3]
labels = ['Series 1', 'Series 2', 'Series 3']
colors = ['royalblue', 'crimson', 'darkorange']

for idx, ind_series in enumerate(series):
    mean = np.mean(ind_series)
    std = np.std(ind_series)
    series[idx] = (ind_series.astype(float) - mean)/std

fig, axes = plt.subplots(3, 1, figsize=(10, 6), sharex=True, gridspec_kw={'hspace': 0.3

# Find global min/max for consistent x-axis
all_data = np.concatenate(series)
xmin, xmax = all_data.min() - 10, all_data.max() + 10

for i, ax in enumerate(axes):
```

```
    y = np.zeros_like(series[i])
    ax.scatter(series[i], y, s=100, color=colors[i], zorder=2)
    ax.axhline(0, color='grey', linewidth=1, zorder=1)
    ax.set_yticks([])
    ax.set_xlim(xmin, xmax)
    ax.set_title(labels[i], fontsize=14, loc='left')
    ax.grid(True, axis='x', linestyle='--', alpha=0.5)
    if i < 2:
        ax.tick_params(labelbottom=False)
    else:
        ax.set_xlabel('Value', fontsize=16)
    ax.spines['left'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['top'].set_visible(False)

fig.suptitle('Three Series, Standardised', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

Standardising series in this way, we preserve the information they contain while aligning their mean and variance.

20.3.2.3 Example

Let's go back to the example data and scale the Size (m^2) series:

Property	Size (m^2)	Number of rooms	Price (k €)
A	60	2	320
B	80	3	400
C	120	5	350
D	70	2	310
E	150	6	600
F	90	4	330

To do so, we need to compute the series mean and variance:

$$\begin{aligned}
\text{Mean} &= \frac{60+80+120+70+150+90}{6} = \frac{570}{6} = 95 \\
\text{Variance} &= \frac{(60-95)^2+(80-95)^2+(120-95)^2}{6} \\
&\quad + \frac{(70-95)^2+(150-95)^2+(90-95)^2}{6} \\
&= \frac{5750}{6} \approx 958.3 \\
\text{Standard Deviation} &= \sqrt{\text{Variance}} = \sqrt{958.3} \approx 31
\end{aligned}$$

Standardised series:

Property	Size (m²)	Scaled Size
A	60	$\frac{60-95}{31} = -1.13$
B	80	$\frac{80-95}{31} = -0.48$
C	120	$\frac{120-95}{31} = 0.81$
D	70	$\frac{70-95}{31} = -0.81$
E	150	$\frac{150-95}{31} = 1.77$
F	90	$\frac{90-95}{31} = -0.16$

Exercise 20.4.

Property	Size (m²)	Number of rooms	Price (k €)
A	60	2	320
B	80	3	400
C	120	5	350
D	70	2	310
E	150	6	600
F	90	4	330

Standardise the "Number of Rooms" series.

That is it for scaling methods, well done for making it this far! The following section will cover some of the practical details, but no more maths, I promise.

20.4 Information Leakage

Data preprocessing is a common source of information leakage. Descriptive statistics like the minimum, maximum, mean and standard deviation of a series are a source of information.

When applying Min-Max Scaling or Standardisation, use descriptive statistics (like min and max) computed from the **training set only**. When generating

predictions on the test set, preprocess the data using the descriptive statistics of the training set.

As an example, for Min-Max Scaling, you would use the minimum and maximum of the training set to preprocess the test data. This sounds counter-intuitive and a lot of work, why should we bother?

The answer to that question can be found in the reason we split the data into train and test sets. We want to estimate how the model would perform on **unseen data**.

In the case of property pricing, imagine you have trained your model using Min-Max Scaler, and that a new property comes in, ready to be priced. How would you scale the features (area and number of rooms) of this new property?

The descriptive statistics of this property are irrelevant as you only have a single observation. You would use the min and max of the **training data**, the only data you have. Because the purpose of the test set is to simulate what would happen at prediction time, it makes sense to treat it in the same way.

In Machine Learning jargon, these scaling methods are **fitted** to the training data, and applied to the test set. The transformation is based on metrics calculated on the training set and applied to the test set.

20.5 Final Thoughts

Some models like KNN require numerical features to be on the same scale. Otherwise, features with the largest variance (here surface area) will dominate distance calculations. This chapter reviewed two ways to scale numerical features:

- Min-Max Scaling
- Standardisation

Both of these need to be applied with care to avoid information leakage between the train and test sets. The way to do so is to use the descriptive statistics of the training set to preprocess the test data. This is what happens at inference.

This is it, you made it! This was the last chapter of the Data Preprocessing section. The next section will put everything together - applying the book's content to an end-to-end Machine Learning project.

20.6 Solutions

Solution 20.1. Exercise 20.1

Given the data:

Property	Size (m²)	Number of rooms
A	60	2
B	80	3
C	120	5
D	70	2
E	150	6
F	90	4

Min-Max Scaling formula:

$$\text{Scaled Value} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

1. Surface Area (Size in m²)

- $x_{\min} = 60$
- $x_{\max} = 150$

Property	Size (m²)	Scaled Size (x')
A	60	$\frac{60-60}{150-60} = 0$
B	80	$\frac{80-60}{150-60} = \frac{20}{90} \approx 0.222$
C	120	$\frac{120-60}{150-60} = \frac{60}{90} \approx 0.667$
D	70	$\frac{70-60}{150-60} = \frac{10}{90} \approx 0.111$
E	150	$\frac{150-60}{150-60} = 1$
F	90	$\frac{90-60}{150-60} = \frac{30}{90} \approx 0.333$

2. Number of Rooms

- $x_{\min} = 2$
- $x_{\max} = 6$

Property	Number of rooms	Scaled Rooms (x')
A	2	$\frac{2-2}{6-2} = 0$
B	3	$\frac{3-2}{6-2} = \frac{1}{4} = 0.25$
C	5	$\frac{5-2}{6-2} = \frac{3}{4} = 0.75$
D	2	$\frac{2-2}{6-2} = 0$
E	6	$\frac{6-2}{6-2} = 1$
F	4	$\frac{4-2}{6-2} = \frac{2}{4} = 0.5$

Scaled Data

Property	Scaled Size	Scaled Rooms
A	0	0
B	0.222	0.25
C	0.667	0.75
D	0.111	0
E	1	1
F	0.333	0.5

Solution 20.2. Exercise 20.2

Series 1	Series 2
1	4
2	8
4	10
5	12
8	16

$$\text{Mean}_{x_2} = \frac{4+8+10+12+16}{5} = 10$$

Solution 20.3. Exercise 20.3

Series 1	Series 2
7	4
8	8
10	10
11	12
14	16

For Series 2: [4, 8, 10, 12, 16]:

$$\bar{x}_2 = \frac{4+8+10+12+16}{5} = 10$$

Variance:

$$\begin{aligned}\text{Var}(x_2) &= \frac{(4-10)^2+(8-10)^2+(10-10)^2+(12-10)^2+(16-10)^2}{5} \\ &= \frac{36+4+0+4+36}{5} = \frac{80}{5} = 16\end{aligned}$$

Solution 20.4. Exercise 20.4

Property	Size (m²)	Number of rooms	Price (k €)
A	60	2	320
B	80	3	400
C	120	5	350
D	70	2	310
E	150	6	600
F	90	4	330

$$\text{Mean} = \frac{2+3+5+2+6+4}{6} = \frac{22}{6} \approx 3.67$$

$$\begin{aligned}\text{Variance} &= \frac{(2-3.67)^2 + (3-3.67)^2 + (5-3.67)^2}{6} \\ &\quad + \frac{(2-3.67)^2 + (6-3.67)^2 + (4-3.67)^2}{6} \\ &= \frac{13.33}{6} \approx 2.22\end{aligned}$$

$$\text{Standard Deviation} = \sqrt{\text{Variance}} = \sqrt{2.22} \approx 1.49$$

Standardised series

Property	Number of rooms	Scaled Number of Rooms
A	2	$\frac{2-3.67}{1.49} = -1.12$
B	3	$\frac{3-3.67}{1.49} = -0.45$
C	5	$\frac{5-3.67}{1.49} = 0.89$
D	2	$\frac{2-3.67}{1.49} = -1.12$
E	6	$\frac{6-3.67}{1.49} = 1.56$
F	4	$\frac{4-3.67}{1.49} = 0.22$

Part V

Bringing it all Together

Chapter 21

End-to-End Machine Learning Project

It's now time to bring everything we've studied together to build a Machine Learning model to predict **anything**.

21.1 Problem Formulation

The most important step is to **define** the problem you want to solve as a supervised learning problem. Using examples studied in this book:

- What is the diagnosis of this suspicious mass?
- What is the price of this property?

There are many other examples. The important part is to follow the supervised learning paradigm:

$$\text{Input Features} \rightarrow \text{Model} \rightarrow \text{Prediction}$$

What would you like to predict to solve an everyday problem?

Exercise 21.1. How would you solve these problems with a Machine Learning model? What would you predict?

1. How many bartenders do I need to hire for that date?
2. Should I increase the stock of a particular item at my shop?
3. Is this online transaction fraudulent?

21.2 Evaluation Metric

Now that the problem is formulated as a supervised learning task, let's select an error metric to minimise. This error metric should reflect the real-world consequences of an error.

In the tumour diagnosis case, **False Negatives**, malignant tumours diagnosed as "benign", can have fatal consequences. For that reason, **Recall** and **F1 Score** could be interesting metrics to track.

In the property pricing example, extreme pricing errors can have a negative impact on any real estate business. For this reason, the **Mean Squared Error** (MSE) or **Root Mean Squared Error** (RMSE) could be good choices.

You may want to select several error metrics, but you will generally have to choose one to rank different models.

Exercise 21.2. Which error metrics would you choose for the following problems?

1. Spam detection
2. Credit card fraud detection
3. Customer churn prediction

21.3 Data Collection

Now that you have a problem, gather as much relevant data as possible. Keeping supervised learning in mind, the goal is to give the model enough data to learn the relationship between input features and the target variable.

Looking at the property pricing example, the model should have access to as many price-relevant features as possible:

- Surface Area
- Number of Rooms
- Neighbourhood
- Balcony
- Floor
- etc…

It is generally a good idea to include whatever information about the property that would help humans to price it, and a bit more. Why more? Because models can sometimes learn patterns that we cannot spot.

Exercise 21.3. Beer sales forecasting: Imagine you want to forecast beer sales at your bar. What features would you choose?

21.4 Partitioning the Data

Once you have gathered the data, and before you start data preprocessing, it is critical to set aside a portion of the data for testing. You could either take a random sample of the data or use a time cut-off to mimic the model's prediction conditions.

If you develop a property pricing model, you may want to keep the latest weeks of your training set as a test set, to make sure that the model does not have access to future price trends in training. For the tumour diagnosis case, a random sample of the entire dataset should be enough.

21.5 Data Preprocessing

You will most likely have to clean and preprocess the data you have gathered:

- Is there missing data?
- Are the numeric values on the same scale?
- How do you want to handle date features?
- Are there categorical variables to process?

Note: different models sometimes require different preprocessing. As this is an introductory text, we will leave finer distinctions to more advanced material. As a quick example, unlike K-Nearest Neighbours, Decision Trees do not require numerical feature scaling.

Once you've preprocessed the training data, apply the same transformations on the test set, using the statistics computed with the training data. If this is not clear enough, you can refer to the Data Preprocessing section.

21.6 Model Evaluation and Selection

This is already a lot of work, and we still haven't trained a single Machine Learning model. Welcome to the reality of Machine Learning professionals. A lot of our time is spent formulating problems, gathering and preprocessing data.

Now, train a KNN model and a Decision Tree model on the training data. You can then use both of these models to generate predictions on the test set.

With these predictions, compare the error metric of both models and pick the best one! You can then use this model to generate predictions on unseen observations. The goal of this book.

21.7 Final Thoughts

This chapter reviewed the main steps of solving a problem with Machine Learning predictions:

- Problem Formulation
- Evaluation Metric
- Data Collection
- Data Partition
- Data Preprocessing
- Model Evaluation and Selection

That's it! The purpose of this book was to give an idea of what building Machine Learning solutions can look like. Interested readers can explore how to put this knowledge in practice with further resources like: *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* (Géron 2022).

The Appendix below explores some of the differences between the description made above and Machine Learning in practice.

The following chapter links Generative AI models to traditional Machine Learning; showing how the models we use every day were built upon everything studied in this book.

21.8 Appendix: Some Nuances

This chapter presents a simplified view of Machine Learning practice. If you are not interested in the details, you can skip directly to the conclusion.

First, a lot of time would be spent on "**Feature Engineering**", the task of computing features from raw data to make models more accurate.

Then, model selection would involve more than just two models. The book compared two model families: KNN and Decision Trees. There are many more. Model selection would also involve "**Hyperparameter Tuning**". These hyperparameters are the settings or configuration of the models. They determine how the models work. Some examples of hyperparameters include:

- **KNN**: The number of neighbours used to generate predictions. The text used 5, but 3, 10 or 15 can also be viable choices
- **Decision Tree**: the maximum depth of a tree, to avoid making too many data splits

To choose between all of these model families and hyperparameter combinations, a single test set is not enough. ML practitioners generally use **cross-validation** over the training set. The interested reader can find more on this at (scikit-learn contributors 2025).

21.9 Final Thoughts

This is it. You now have an intuition for the Machine Learning workflow. This was the core purpose of this book.

But what about Generative AI and LLMs? I thought you would never ask. If you are interested in understanding the many parallels between traditional Machine Learning and LLMs, read the next chapters.

Otherwise, I wish you all the best on your Machine Learning journey.

21.10 Solutions

Solution 21.1. Exercise 21.1

1. How much staff do I need to hire for that date at my bar? Beer sales forecasting
2. Should I increase the stock of a particular item at my shop? Unit sales forecasting
3. Is this online transaction fraudulent? Transaction classification as "legitimate"/"fraudulent"

Solution 21.2. Exercise 21.2

1. Spam detection: Precision, Recall Accuracy. A False Positive, a legitimate email marked as "spam" can be more problematic than a False Negative, a spam email classified as "legitimate".
2. Credit card fraud detection: Recall, Precision, F1 Score. The model should catch most fraudulent transactions (high Recall) while not having too many False Positives, as they could be an inconvenience to customers.
3. Customer churn prediction: F1 Score. A balance between Precision and Recall is needed to identify customers who are likely to churn without having too many False Positives.

Solution 21.3. Exercise 21.3

Some potential features for beer sales forecasting could be:

- **Date/Time**: Month, day of the week, time of day.
- **Weather**: Temperature, rain, sun.
- **Events**: Are there any major events happening in the city, like a football match or a concert?
- **Promotions**: Is there a special offer on beer?
- **Historical sales data**: Sales from previous days, weeks, or months.

Chapter 22

Machine Learning and Generative AI

These days, it is hard to write anything about Machine Learning without mentioning Generative Artificial Intelligence (GenAI). GenAI took the world by storm with the public release of ChatGPT in 2022.

Breaking down this term:

- **Generative:** producing content such as texts, images, audio or video
- **Artificial:** not human
- **Intelligence:** "the ability to learn, understand, and make judgments or have opinions that are based on reason"(Cambridge Dictionary 2024b)

More generally, artificial intelligence can be thought of as the emulation of human intelligence, defined above. Further developments may invalidate this definition as the algorithms developed acquire capabilities going beyond human understanding.

GenAI was one of the major technological breakthroughs of the early 21st century. It was built on a **combination of the different Machine Learning types** listed earlier.

22.1 Starting with the Foundation

The foundation of GenAI models is a **next word (or token) predictor**. When we send a request to a GenAI model, we send the following input:

```
User: What is the capital of France?

Assistant:
```

The model takes this input and predicts `The` as output.

The answer would be a bit disappointing if it ended there. To continue, the model uses the previous input sequence and adds the `The` token it predicted:

```
User: What is the capital of France?

Assistant: The
```

and outputs `capital`. This process goes on until the model predicts an `<end>` token, meaning that the response is over. The practice of appending a prediction to the original input sequence to generate another prediction is called **auto-regression**.

Note: Large Language Models do not predict the next word but the next **token**. A token is a string of characters which could be a part of a word ("ing") or a full word ("the"). We will stick to words in this simple explanation.

Exercise 22.1. What type of prediction is the task of next word/token prediction?

Solution 22.1. It is a classification task, with as many labels as possible words/tokens. This number is generally called the **vocabulary size**, which is just above 100,000 for OpenAI's GPT-4 model.

Even though this is a classification task, the foundational training of Large Language Models is generally referred to as **self-supervised** learning, instead of just **supervised** learning. This is because, unlike the classification tasks listed in the previous section, there is no dataset of input and outputs. The model is simply trained to predict the next word of every sentence it finds.

As an example, if the training corpus contains the sentence:

> "The Second World War ended in 1945."

It would include the following input/output pairs in its training:

Input	Output
The	Second
The Second	World
The Second World	War
The Second World War	ended
The Second World War ended	in
The Second World War ended in	1945
The Second World War ended in 1945	.

You may see that some guesses are much easier than others.

The foundation model is trained with supervised learning as it learns from input/output pairs. Yet, this is self-supervised as these input/output pairs do not require special curation.

22.2 Is Next Word Prediction Enough?

If you simply train a model to predict the next word using all of the text of the internet, you may come across surprising behaviours. As an example, the request:

> "3 x 1="

could be answered:

> "3, 3 x 2 = 6, 3 x 3 = 9" …

This is helpful, but only "3" was needed there; unless you are in the business of writing schoolbooks.

In this case, only the answer of "3 x 1" was needed, and yet, most texts including the string "3 x 1" simply list the multiplication table for 3. You could validate this by looking for this sequence of characters in the books you have at home, many of which should be elementary Maths textbooks.

More work is needed to build a model that helps users and answers queries. There are two main ways to do this.

22.3 Learning from Questions and Answers

In addition to using the text published on the internet for training, one could further train a foundation model to predict the next word of texts involving **useful** questions and answers. This could solve the issue described above. This additional training is called **fine-tuning**. It is called **supervised** fine-tuning as instead of learning from raw internet texts, it learns from **selected** question/answer interactions.

As an example, we can fine-tune the model with the following interactions:

```
User: 3 x 2 =
Assistant: 3 x 2 = 6
```

```
User: 7 x 2 =
Assistant: 7 x 2 = 14
```

etc.

This should train the model to reply to the user query instead of simply completing a text. Supervised fine-tuning does help, but is not enough to build the GenAI models we use every day.

22.4 Optimising for Helpfulness

The goal of GenAI model providers is to offer models that are as **helpful** as possible. Achieving high degrees of helpfulness cannot be done through the pretraining of foundation models or supervised fine-tuning.

From these two steps, helpfulness can emerge as a by-product (see previous section). Instead, could we optimise a model for helpfulness? If we could, what type of Machine Learning task could we use?

Hint: consider the degree of helpfulness as a reward the model can maximise by choosing the word to use in its reply.

If this makes you think of reinforcement learning, well done. Helpfulness is the **reward** the model would try to maximise, and the words it uses are its **actions**. But how would you measure helpfulness?

First, you could ask human judges to rate each model output on a scale of 0 (not helpful) to 100 (unbelievably helpful). These may be problematic as the helpfulness scales of different humans may vary. Can we do better?

Easier than a rating, human judges could simply choose the most helpful of two model outputs. This is called **pairwise ranking**. It has several advantages:

- It is less cognitively demanding than rating or grading
- It is more robust to variations of individual helpfulness scales

During training, the model would learn to generate more helpful output. This is called **Reinforcement Learning from Human Feedback** (RLHF), the last building block of today's GenAI models.

Beyond human pairwise ranking

Once we gather enough examples of human pairwise assessments, we can train a model to predict the winner of two candidate suggestions.

In doing so, we get back to the supervised learning territory.

$$\text{Input} \longrightarrow \text{Model} \longrightarrow \text{Prediction}$$

The input here would be the two candidate suggestions, and the prediction would be the winning prediction (0 for the first and 1 for the second). The training data is all the candidate suggestions and human assessments collected in the process described above.

However, this is not a **tabular** Machine Learning problem. The candidate suggestions are two variable-length sequences of text. This is part of Natural Language Processing, a fascinating area of research.

22.5 Final Thoughts

In essence, Generative AI models are next word predictors, further trained to provide more helpful answers to user questions. They are built on the same principles studied in this book. Next word prediction is yet another (complex) classification task.

This description of Large Language Models is simplified. Please refer to *Hands-on Large Language Models* (Alammar and Grootendorst 2023) for a more rigorous presentation.

References

Alammar, Jay, and Maarten Grootendorst. 2023. *Hands-on Large Language Models: Understand, Build, and Use LLMs*. O'Reilly Media. https://www.oreilly.com/library/view/hands-on-large-language/9781098150952/.

Cambridge Dictionary. 2024a. "Algorithm." Cambridge Dictionary. 2024. https://dictionary.cambridge.org/dictionary/english/algorithm.

———. 2024b. "Intelligence." Cambridge Dictionary. 2024. https://dictionary.cambridge.org/dictionary/english/intelligence.

———. 2024c. "Model." Cambridge Dictionary. 2024. https://dictionary.cambridge.org/dictionary/english/model.

———. 2024d. "True." Cambridge Dictionary. 2024. https://dictionary.cambridge.org/dictionary/english/true.

———. 2024e. "Truth." Cambridge Dictionary. 2024. https://dictionary.cambridge.org/dictionary/english/truth.

CDLI contributors. 2025. "Terms of Use." https://cdli.earth/terms-of-use. https://cdli.earth/terms-of-use.

Davenport, Thomas H., and D. J. Patil. 2012. "Data Scientist: The Sexiest Job of the 21st Century." *Harvard Business Review* 90 (10): 70–76. https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century.

Dhar, Vasant. 2013. "Data Science and Prediction." *Communications of the ACM* 56 (12): 64–73. https://doi.org/10.1145/2500499.

"Exclusive Or." n.d. https://en.wikipedia.org/wiki/Exclusive_or.

Géron, Aurélien. 2022. *Hands-on Machine Learning with Scikit-Learn, Keras, & TensorFlow*. O'Reilly Media. https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/.

Merriam-Webster Dictionary. 2024a. "Extrapolate." Merriam-Webster. 2024. https://www.merriam-webster.com/dictionary/extrapolate.

———. 2024b. "Intuition." Merriam-Webster. 2024. https://www.merriam-webster.com/dictionary/intuition.

Parrish, Shane. 2016. "Daniel Kahneman and Herbert Simon on Intuition." Farnam Street. 2016. https://fs.blog/daniel-kahneman-on-intuition/.

Russell, Stuart J., and Peter Norvig. 2021. *Artificial Intelligence: A Modern Approach*. 4th ed. Hoboken: Pearson.

scikit-learn contributors. 2025. "Cross-Validation: Evaluating Estimator Performance." scikit-learn. 2025. https://scikit-learn.org/stable/modules/cross_

validation.html.