Introduction to R

From Data Foundations to Time Series and Financial Analytics

Mohammad Safavi, Ph.D.

STATLAB Academy

Version 1.0 — August 2026

statistics.safavim.com

About this book

Introduction to R is a self-contained learning path from first expressions through forecasting, volatility, option pricing, market risk, and multivariate financial models. It is designed for students, analysts, and instructors who want executable examples with Canadian context and explicit interpretation—not a catalogue of commands.

The book uses a frozen teaching dataset built from Statistics Canada all-items CPI and Bank of Canada policy-rate and USD/CAD series. Every chapter has a complete R script, and the publication figures are the outputs of those scripts. The source package includes retrieval notes, editable SVG figures, Quarto files, bibliography, and validation logs.

Practical tip

Work through Chapters 1–9 in order if R is new to you. Readers already comfortable with data frames and regression can begin at Part II, but should still read the reproducibility and interpretation conventions in Chapters 4, 5, and 9.

How to use the chapters

Each chapter starts with a concrete motivation and learning objectives, then develops the model or workflow in plain language and notation. Worked examples always pair code with a result, table, or figure and an interpretation. "Why this matters," "Common mistake," and "Practical tip" boxes distinguish purpose, failure modes, and implementation advice. Guided practice and exercises promote active use; selected solutions demonstrate one approach while leaving meaningful work to the learner.

Code blocks are intentionally readable rather than compressed. Package installation belongs to project setup, not analysis scripts. Live data retrieval is kept out of rendering so the PDF is deterministic.

Scope, provenance, and responsible use

The 2013 lecture PDFs supplied for this project, based on themes from Ruey S. Tsay's financial time-series teaching, were read as a topic map. The present text, examples, figures, workflow, and exercises are newly written and modernized; the legacy notes are not redistributed. Relevant published methods are cited to original or standard sources.

This book is educational and is not investment, trading, legal, accounting, or regulatory advice. Models are simplified representations; historical and simulated performance do not guarantee future results. Verify current package documentation, data licences, and institutional requirements before operational use.

STATLAB Academy — statistics.safavim.com

Contents

List of Figures

List of Tables

Part I

Part I: Learning R

CHAPTER 1

A First Analysis in R

1.1 Motivation

R becomes useful the moment it helps answer a real question. We begin with Canadian inflation rather than a tour of menus, so every object, function, and plot has a purpose: describe what changed, make the work repeatable, and communicate uncertainty without overstating what the data can establish.

1.2 Learning objectives

By the end of this chapter, you should be able to:

- identify the console, script, environment, files, and plot panes in RStudio or Positron
- run an expression and assign its result with `<-`
- inspect a data frame with `head()`, `str()`, and `summary()`
- build and save a small, reproducible analysis
- distinguish a descriptive result from a causal claim

1.3 Packages and data

Base R (`utils`, `stats`, and `graphics`) and the frozen `data/canada_macro_monthly.csv` file. No installation is needed for this chapter.

The complete tested script is `scripts/ch01_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Starting with a real dataset makes syntax memorable. Every operator is attached to an analytic decision, and every decision can be checked against the source data.

1.4 Core ideas

R is both a language and an environment for statistical work. An expression such as `2 + 2` is evaluated immediately; an assignment such as `answer <- 2 + 2` gives the value a name. Names should describe meaning rather than position: `inflation` is safer than `x1`. R is case-sensitive, and a function call uses parentheses even when it has no arguments.

A script is the durable record of an analysis. The console is useful for experiments, but console history alone does not explain the order of operations or preserve the reasoning. A good script runs from a fresh R session, reads inputs using project-relative paths, creates outputs deliberately, and contains short comments about *why* a choice was made. The project—not a machine-specific working directory—is the unit of reproducibility.

Data frames are rectangular collections of equal-length columns. `read.csv()` imports the frozen file, `as.Date()` gives the date column calendar meaning, and `summary()` reports quick diagnostics. The latest inflation value is a description of the selected data release; it is not a forecast and does not by itself explain inflation. That distinction between computation and interpretation will recur throughout the book.

The analysis cycle in Figure 1.2 is iterative. A surprising plot may expose a data problem and send us back to import or cleaning. A model diagnostic may alter the original question. Reproducibility means those revisions are visible in code rather than hidden in manual edits.

1.5 Worked example 1: Import and inspect Canadian macro data

```
macro <- read.csv("data/canada_macro_monthly.csv")
macro$date <- as.Date(macro$date)

head(macro, 3)
nrow(macro)
tail(macro$inflation_yoy_percent, 1)
```

Result

The file contains **114 monthly observations**. The latest frozen value is **2.798%** year-over-year inflation (June 2026).

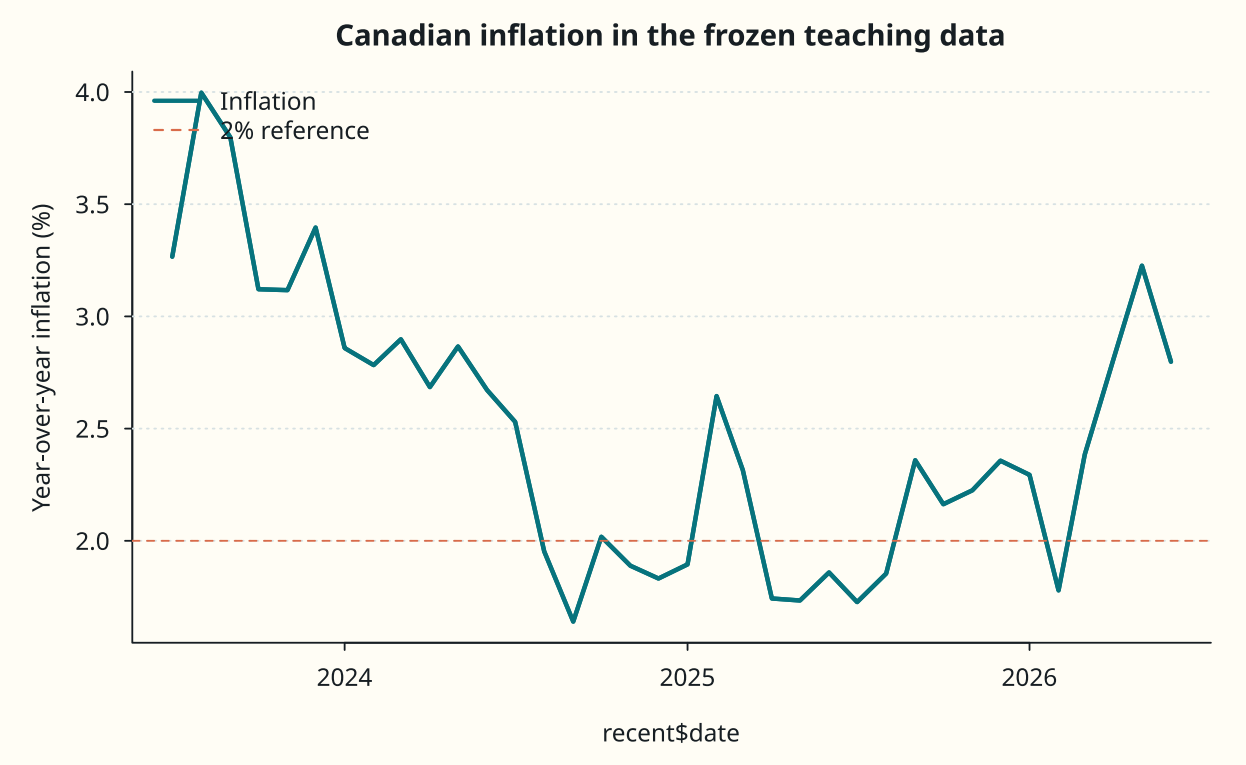

Figure 1.1: Canadian year-over-year inflation in the final 36 months of the frozen teaching data; the dashed line is a 2% reference, not a forecast.

Interpretation. The date conversion allows chronological plotting and comparison. Because the dataset is frozen, the number is reproducible even if an official series is later revised. Always include the reference period when communicating a latest value.

1.6 Worked example 2: Turn a question into a small workflow

```
recent <- tail(macro, 36)
mean_recent <- mean(recent$inflation_yoy_percent)
peak_recent <- max(recent$inflation_yoy_percent)

c(mean = mean_recent, peak = peak_recent)
```

Result

R returns named values. Naming the statistics prevents a reader from having to infer which number is which; the complete tested results are written to `outputs/worked_results.csv`.

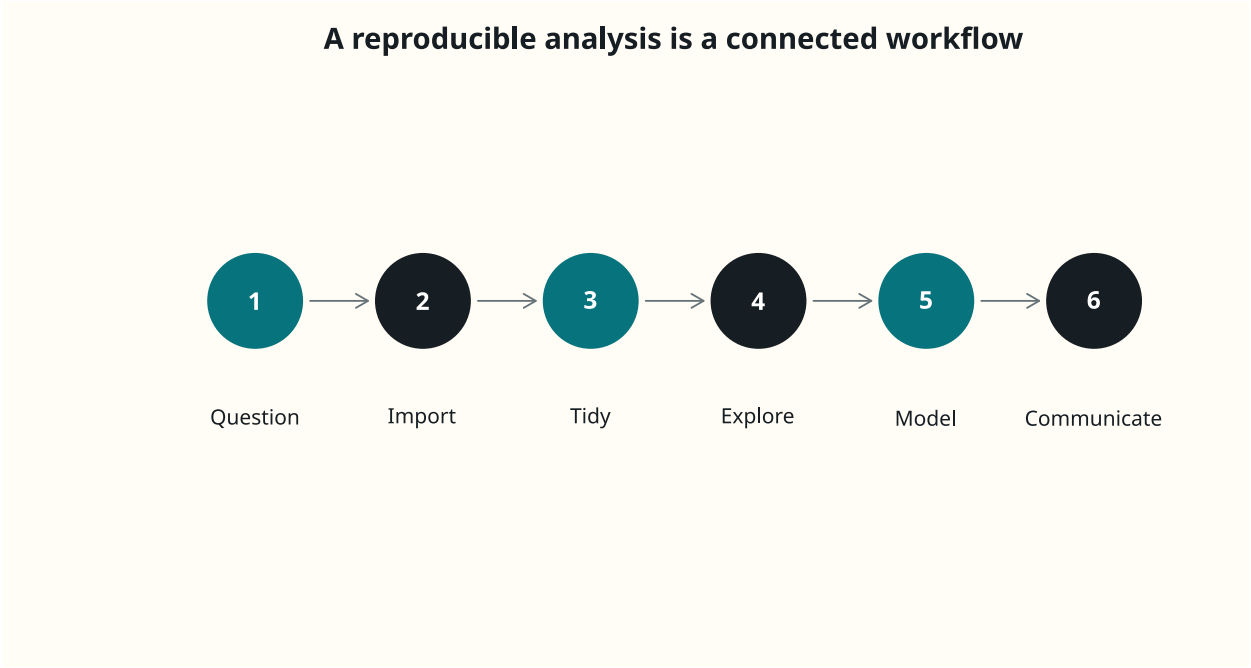

Figure 1.2: Six connected stages of a reproducible data analysis: question, import, tidy, explore, model, and communicate.

Interpretation. A three-line computation is already an analysis when it answers a defined question and retains its context. The workflow figure emphasizes that communication is part of the analysis, not an afterthought.

Common mistake

Do not paste results from the console into a report while leaving the generating code elsewhere. The code, data version, and interpretation must travel together.

Practical tip

Restart R and run the script from top to bottom before sharing it. A script that works only because an old object remains in memory is not reproducible.

1.7 Guided practice

Change the reference line in the inflation plot from 2% to 3%. Before running the code, predict which months will lie above the line. Then compute the count with a logical condition and explain the period covered.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

1.8 Exercises

1. Compute the mean and median inflation for the full frozen sample. Why might they differ?
2. Find the date and value of the maximum policy rate without sorting the CSV manually.
3. Write a five-line script that imports the data, validates that dates are ordered, and prints the latest USD/CAD value.

1.9 Selected solutions

Exercise 1.

```
mean(macro$inflation_yoy_percent)
median(macro$inflation_yoy_percent)
```

Exercise 2.

```
i <- which.max(macro$policy_rate_percent)
macro[i, c("date", "policy_rate_percent")]
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

1.10 Chapter summary

- R evaluates expressions and stores results in named objects.
- Scripts are the durable record; the console is a scratch space.
- A reproducible project uses explicit inputs, relative paths, and stated reference periods.
- Descriptive calculations do not automatically support causal conclusions.

1.11 Chapter cheat sheet

Table 1.1: Chapter 1 command cheat sheet.

Command or pattern	Purpose
`<-`	assign a value
`read.csv()`	import a CSV file
`head()` / `tail()`	inspect boundary rows
`str()`	inspect structure and types
`summary()`	quick numerical diagnostics
`which.max()`	locate a maximum

CHAPTER 2

Vectors, Types, and Subsetting

2.1 Motivation

Most R calculations act on vectors. Learning how vectors store values, recycle operations, propagate missingness, and respond to indices is the foundation for reliable data manipulation and statistical modelling.

2.2 Learning objectives

By the end of this chapter, you should be able to:

- create atomic vectors and identify their types
- use vectorized arithmetic instead of unnecessary loops
- subset by position, name, and logical condition
- handle `NA`, `NaN`, and `Inf` deliberately
- recognize coercion and recycling before they cause silent errors

2.3 Packages and data

Base R and the CPI and inflation columns from the Canadian macro dataset.

The complete tested script is `scripts/ch02_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Vectorized expressions are shorter, faster, and easier to audit than element-by-element editing. They also map directly to many statistical formulas.

2.4 Core ideas

An atomic vector contains values of one basic type: logical, integer, double, character, complex, or raw. Combining unlike types causes coercion toward a type that can represent all elements. For example, `c(1, "two")` becomes character, so numerical functions no longer behave as intended. Check with `typeof()` when an imported column is surprising.

Arithmetic is vectorized: `x * 100` multiplies every element, and `x / x[1]` rebases an entire series. When vector lengths differ, R may recycle the shorter one. Exact recycling can be useful, but non-multiple lengths are usually a bug and produce a warning. Explicitly align observations rather than relying on accidental position.

Subsetting is an analytic operation. Positive integers select positions, negative integers exclude positions, character indices select names, and logical vectors keep `TRUE` positions. A missing logical condition yields a missing selection, which is why `which()` and `is.na()` sometimes matter. Use `drop = FALSE` when a matrix or data-frame subset must retain its dimensions.

Missing values represent unknown values, not zero. Most summary functions return `NA` unless told how to handle missingness, for example `mean(x, na.rm = TRUE)`. Removing missing observations changes the estimand, so the choice must be justified rather than applied automatically.

2.5 Worked example 1: Transform an entire CPI vector

```
cpi <- macro$cpi_all_items_2002_100
rebased_cpi <- 100 * cpi / cpi[1]

c(first = rebased_cpi[1], last = tail(rebased_cpi, 1))
```

Result

The first element is exactly 100 by construction. The final value indicates cumulative index growth relative to January 2017; the full-sample CPI increase is **30.50%**.

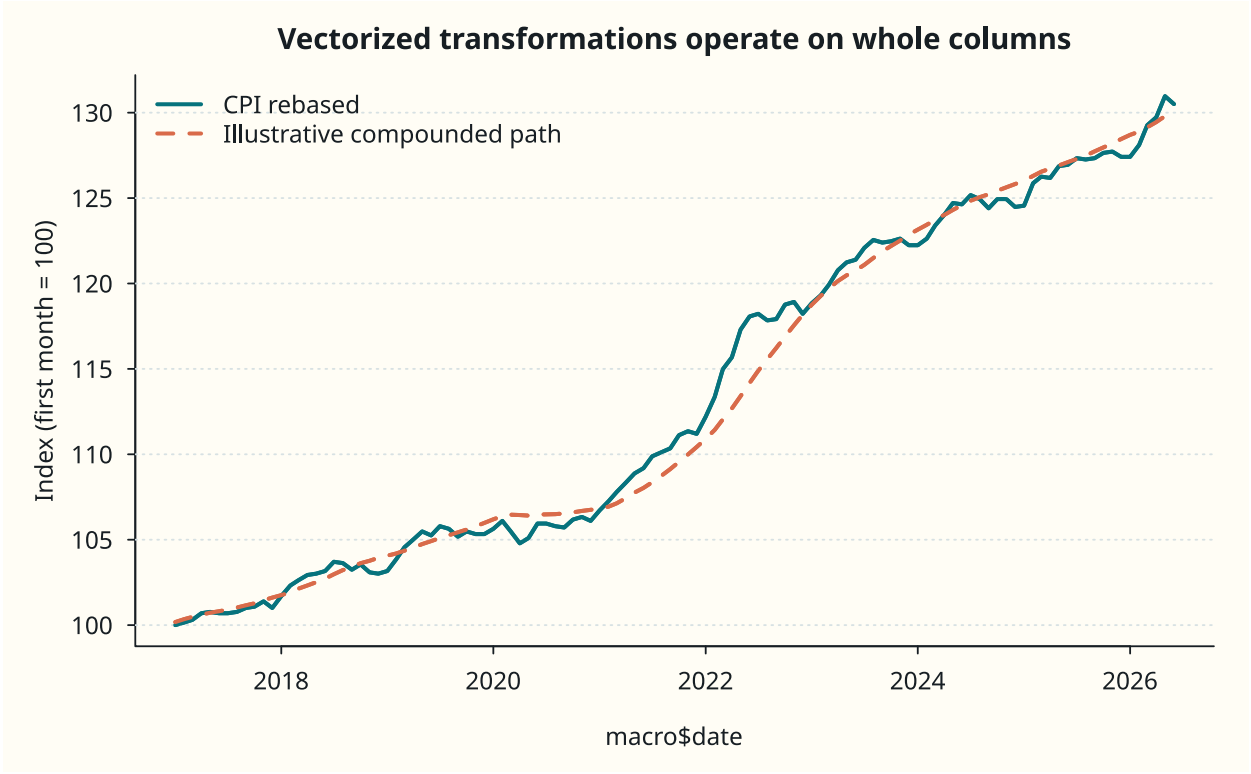

Figure 2.1: Two indexed paths constructed with vectorized arithmetic; both begin at 100 but encode different transformations.

Interpretation. Rebasing changes the reference point, not the underlying percentage changes. Vectorized code states the transformation once and lets R apply it consistently.

2.6 Worked example 2: Select high-inflation months with a logical vector

```
high <- macro$inflation_yoy_percent > 4
sum(high)
macro[high, c("date", "inflation_yoy_percent")]
```

Result

The condition selects **21 months** above 4%; the sample maximum is **8.133%**.

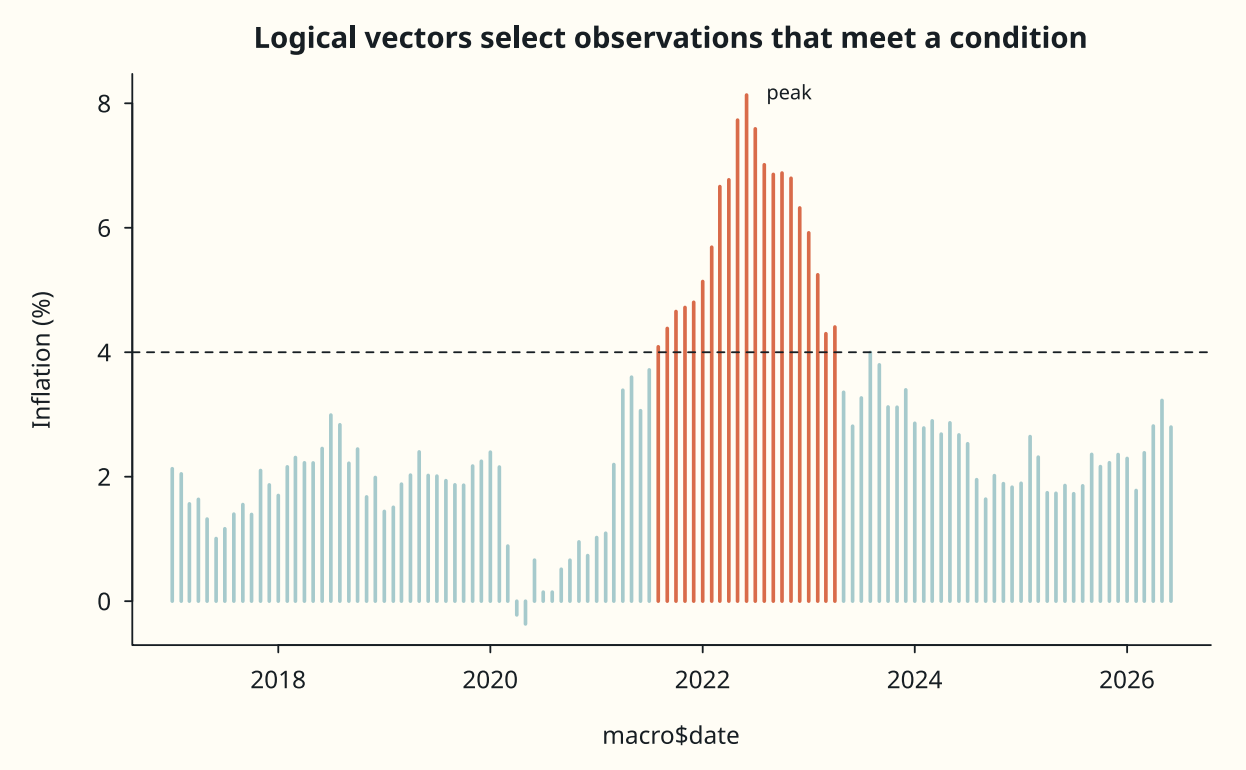

Figure 2.2: Monthly inflation shown as vertical marks, with observations above 4% highlighted.

Interpretation. The threshold is a choice made by the analyst. State it explicitly and keep the date column so selected values retain their temporal context.

Common mistake

Never compare to missingness with `x == NA`; the result is also unknown. Use `is.na(x)` or `!is.na(x)`.

Practical tip

After subsetting, verify both the row count and a few boundary cases. A syntactically valid index can still represent the wrong population.

2.7 Guided practice

Create a logical vector for months with inflation below 1%. Use it to return dates, count observations, and compare the mean policy rate inside and outside that subset.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

2.8 Exercises

1. Convert the USD/CAD column from CAD per USD to USD per CAD and name the new vector.
2. Select every twelfth CPI observation beginning with January 2017.
3. Introduce one `NA` into a copy of inflation and compare `mean(x)` with `mean(x, na.rm = TRUE)`; explain the difference.

2.9 Selected solutions

Exercise 1.

```
usd_per_cad <- 1 / macro$usd_cad_monthly_average
head(usd_per_cad)
```

Exercise 2.

```
january_cpi <- macro$cpi_all_items_2002_100[seq(1, nrow(macro), by = 12)]
january_cpi
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

2.10 Chapter summary

- Atomic vectors have one type; combining values may coerce that type.
- Vectorized operations apply one rule to many elements.
- Indices may be positional, named, or logical.
- Missing values require an explicit analytical choice.

2.11 Chapter cheat sheet

Table 2.1: Chapter 2 command cheat sheet.

Command or pattern	Purpose
`c()`	combine values
`typeof()`	inspect storage type
`seq()`	construct an index sequence
`x[condition]`	logical subset
`is.na()`	identify missing values
`%in%`	membership test

CHAPTER 3

Data Frames, Factors, and Dates

3.1 Motivation

Real analysis needs more than loose vectors. Data frames preserve relationships between columns, factors encode designed categories, and dates make calendar operations explicit.

3.2 Learning objectives

By the end of this chapter, you should be able to:

- construct and inspect data frames
- select rows and columns without dropping structure accidentally
- create ordered factor levels
- parse and format dates
- join metadata only after checking keys

3.3 Packages and data

Base R; `data.frame`, `factor`, and `Date` classes; the frozen Canadian macro data.

The complete tested script is `scripts/ch03_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Correct classes allow R to enforce useful rules: calendar subtraction for dates, level ordering for factors, and column alignment for data frames.

3.4 Core ideas

A data frame is a list of equal-length vectors with row and column structure. Each column may have a different type, but each row represents one observational unit. This book uses one month per row. That unit should be written down before any transformation because duplicated or incomplete keys can otherwise pass unnoticed.

Factors represent a finite set of category levels. The levels are part of the data definition, not merely labels on observed strings. An ordered factor can encode progression such as `low < medium < high`; an unordered factor encodes categories without magnitude. Numerical-looking factor labels should never be converted with `as.numeric(factor)` because that returns internal codes. Convert through character or, better, repair the import.

Dates are stored as days from an origin but printed in calendar form. `as.Date("2026-06-01")` creates a true `Date`; extracting `format(date, "%Y")` or `"%m"` supports grouping. Dates prevent lexicographic mistakes and allow differences in days. Monthly data require a convention—here, the first day labels the month—because a date object does not itself carry a "monthly" frequency.

Keys matter when data frames are combined. A join should have a declared key, such as `date`, and the analyst should check uniqueness before and row counts after. An unexplained many-to-many join can multiply records and corrupt every later statistic.

3.5 Worked example 1: Inspect a mixed-type data frame

```
macro <- read.csv("data/canada_macro_monthly.csv")
macro$date <- as.Date(macro$date)

str(macro)
stopifnot(!anyDuplicated(macro$date))
```

Result

The validated object has **114 rows**, one `Date` column, and four numeric measurement columns. No date is duplicated.

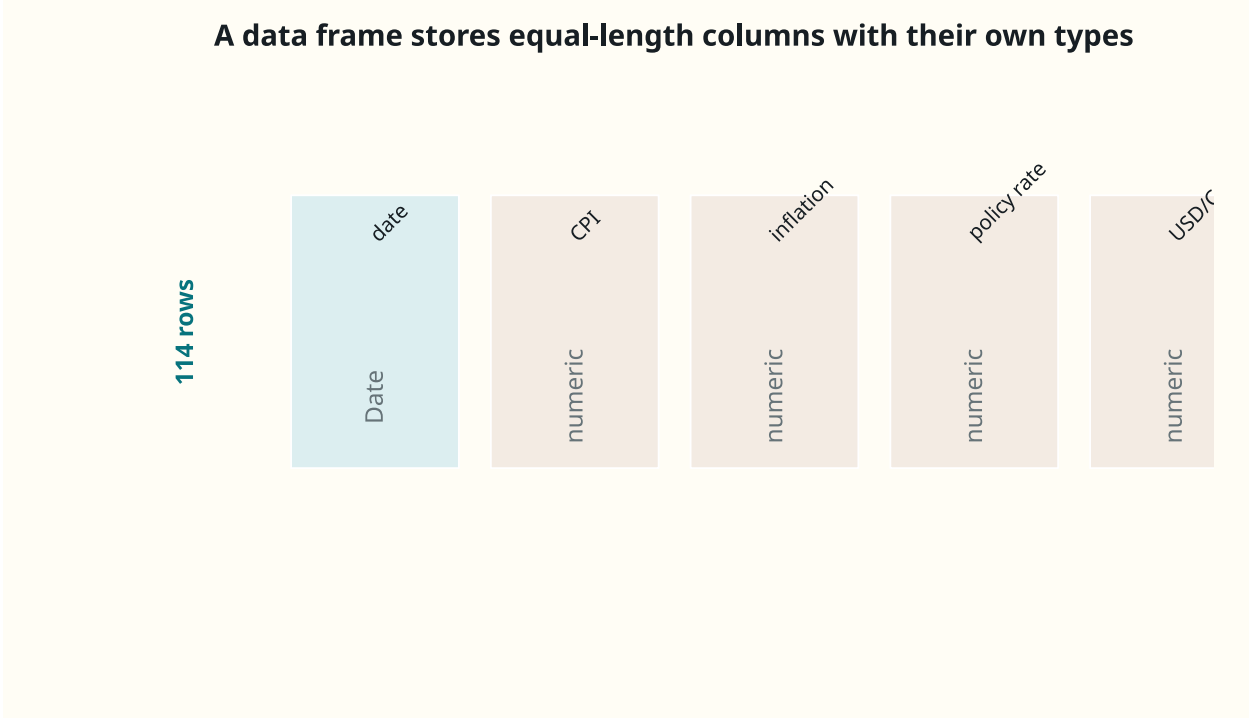

Figure 3.1: Schematic of the Canadian data frame showing 114 rows and the type of each column.

Interpretation. Structure inspection belongs immediately after import. A data frame can print plausibly even when dates are character strings or numeric columns were read as text.

3.6 Worked example 2: Create an economic-era factor

```
era <- cut(
  macro$date,
  breaks = as.Date(c("2016-12-31", "2019-12-31",
                     "2022-12-31", "2026-12-31")),
  labels = c("2017-2019", "2020-2022", "2023-2026")
```

```
)
table(era)
```

Result

The resulting factor has **3 levels** in chronological order. Counts make incomplete periods, such as 2026, visible.

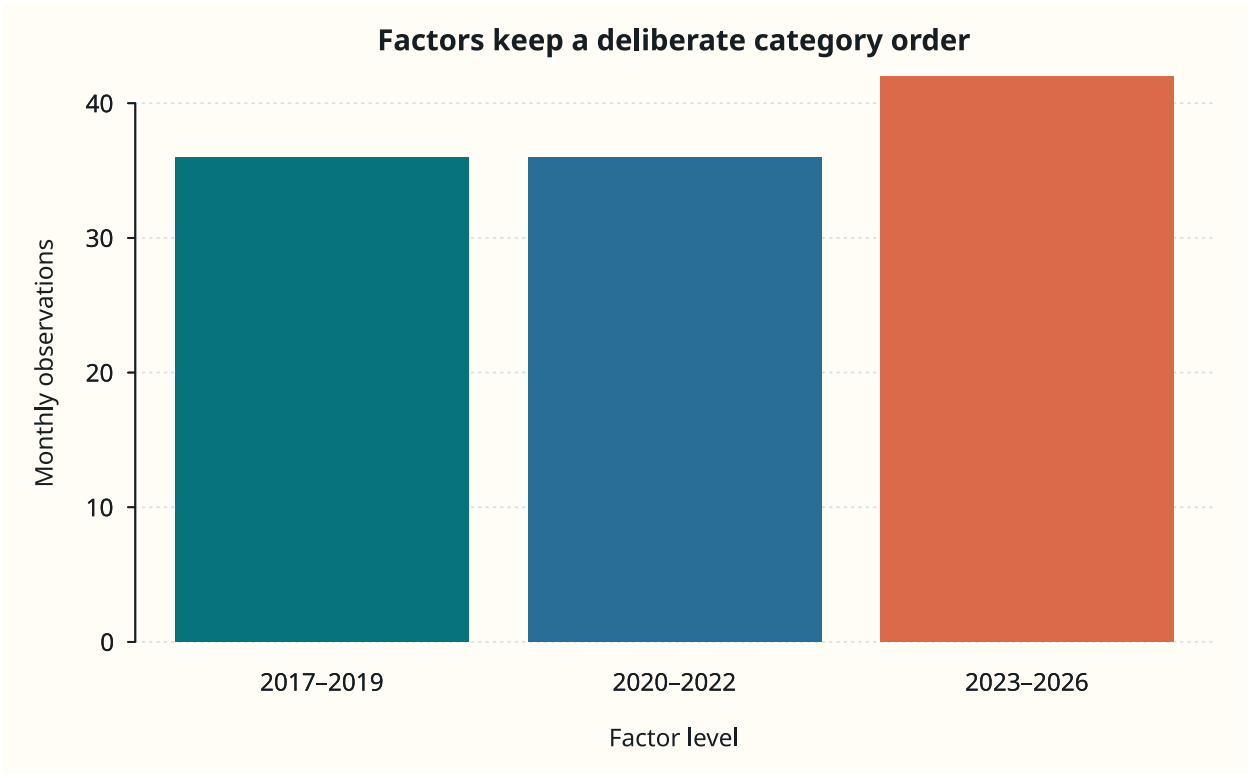

Figure 3.2: Observation counts for the three ordered economic-era factor levels.

Interpretation. Because the breakpoints and labels are in code, another analyst can audit the era definition. Different substantive questions may require different boundaries.

Common mistake

Row names are not a safe primary key. They can change after sorting or subsetting and are rarely preserved across file formats.

Practical tip

For every table, be able to finish the sentence: "one row represents …". Then validate the key that should identify that row.

3.7 Guided practice

Create an ordered factor for inflation bands: below 1%, 1% to under 3%, 3% to under 5%, and 5% or more. Tabulate the result and verify boundary values.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

3.8 Exercises

1. Extract year and month labels from `date` using `format()`.
2. Construct a two-column table mapping each month number to its abbreviated name and verify its key is unique.
3. Explain what would happen if a join key appeared twice in each of two tables.

3.9 Selected solutions

Exercise 1.

```
macro$year <- as.integer(format(macro$date, "%Y"))
macro$month <- format(macro$date, "%b")
```

Exercise 2.

```
calendar <- data.frame(month_number = 1:12, month_label = month.abb)
stopifnot(!anyDuplicated(calendar$month_number))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

3.10 Chapter summary

- A data frame combines equal-length columns around a defined observational unit.
- Factors store category levels and, optionally, order.
- Dates should be parsed, not left as display strings.
- Joining data requires explicit and validated keys.

3.11 Chapter cheat sheet

Table 3.1: Chapter 3 command cheat sheet.

Command or pattern	Purpose
`data.frame()`	construct a rectangular table
`str()`	show column classes
`factor()`	encode categories
`cut()`	bin numeric or date values
`as.Date()`	parse calendar dates
`anyDuplicated()`	validate key uniqueness

CHAPTER 4

Functions, Pipes, and Project Workflow

4.1 Motivation

Repeated analysis becomes reliable when it is expressed as small functions and organized as a portable project. Pipes then make the sequence of transformations readable from left to right.

4.2 Learning objectives

By the end of this chapter, you should be able to:

- write a function with arguments, validation, and a return value
- use the native pipe `|>`
- separate pure calculations from file input and output
- use project-relative paths
- control randomness and record session information

4.3 Packages and data

Base R, including the native pipe introduced in R 4.1; no path-changing helper is required.

The complete tested script is `scripts/ch04_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Functions and projects turn one successful analysis into a process another person can run, inspect, and safely modify.

4.4 Core ideas

A function bundles a rule behind a name. Its arguments state the inputs, the body performs the calculation, and the final evaluated expression is returned. Validation with `stopifnot()` or an informative `stop()` makes assumptions executable. Default arguments are useful when a choice is genuinely conventional, but hidden defaults can make an analysis hard to reproduce.

Pure functions depend only on their inputs and do not alter files or global objects. They are easy to test because the same inputs give the same outputs. File reading, figure creation, and network retrieval are side effects; keep them at the edge of a workflow. This separation lets a calculation be reused with a frozen CSV, a database extract, or a simulated test case.

The native pipe passes the left-hand result into the first argument of the next call. `x |> mean()` is equivalent to `mean(x)`. Pipes are clearest when each step has a descriptive verb and intermediate inspection would be meaningful. A long pipeline that hides branching or repeated computation should be split into named objects or functions.

A portable project stores paths relative to a known root. This book never recommends `setwd()` in a reusable script because it changes process-wide state and usually embeds assumptions about one computer. Random examples call `set.seed()` before simulation. `sessionInfo()` records R and package versions so future differences can be investigated.

4.5 Worked example 1: Write and validate a compounding function

```
compound_value <- function(principal, annual_rate, years) {
  stopifnot(principal >= 0, annual_rate > -1, years >= 0)
  principal * (1 + annual_rate)^years
}

compound_value(1000, 0.05, 10)
```

Result

The future value is **$1,628.895** before rounding. Keeping full precision in the object and rounding only for display avoids cumulative rounding error.

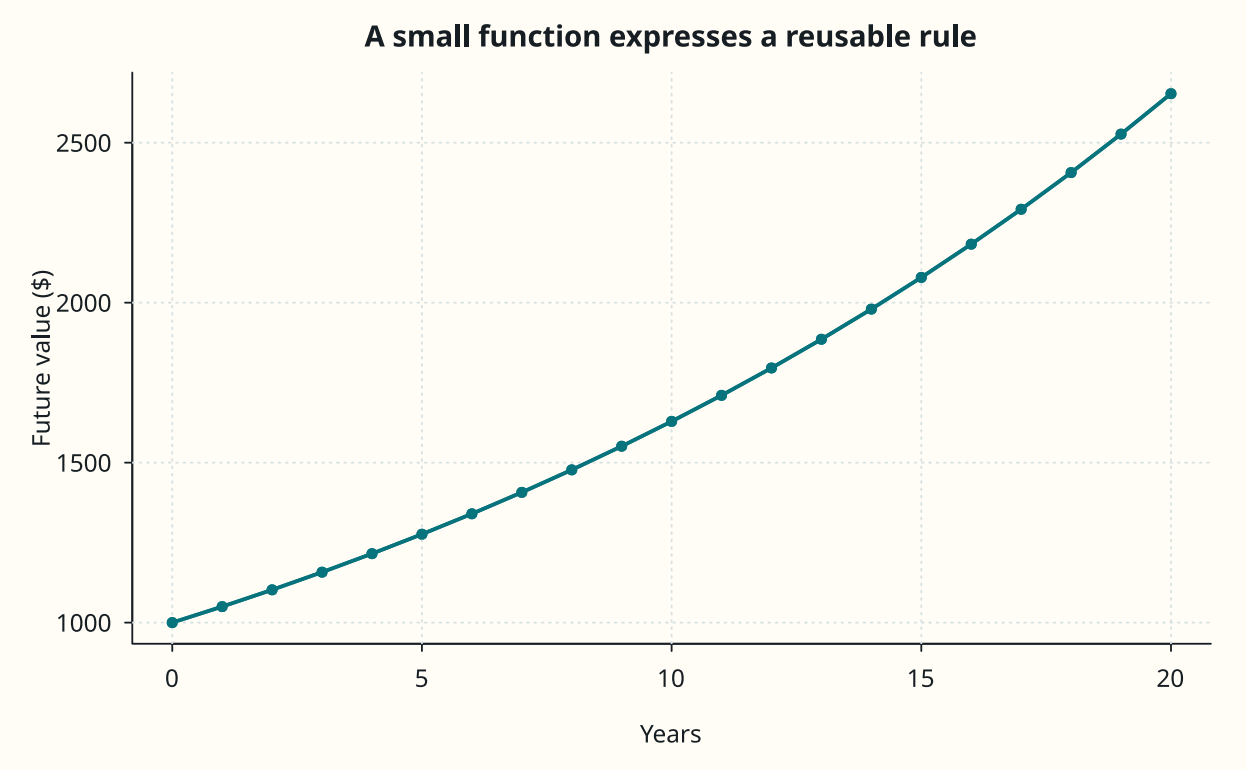

Figure 4.1: Future value over 0 to 20 years using the validated compounding function.

Interpretation. The function's name and arguments communicate the model. Its validation rejects negative principal, impossible rates below -100%, and negative horizons.

4.6 Worked example 2: Make randomness and paths reproducible

```
set.seed(404)
simulation <- rnorm(1000)
mean(simulation)

data_path <- file.path("data", "canada_macro_monthly.csv")
file.exists(data_path)
```

Result

The seeded simulation mean is **–0.01050**, and the relative data path resolves inside the project.

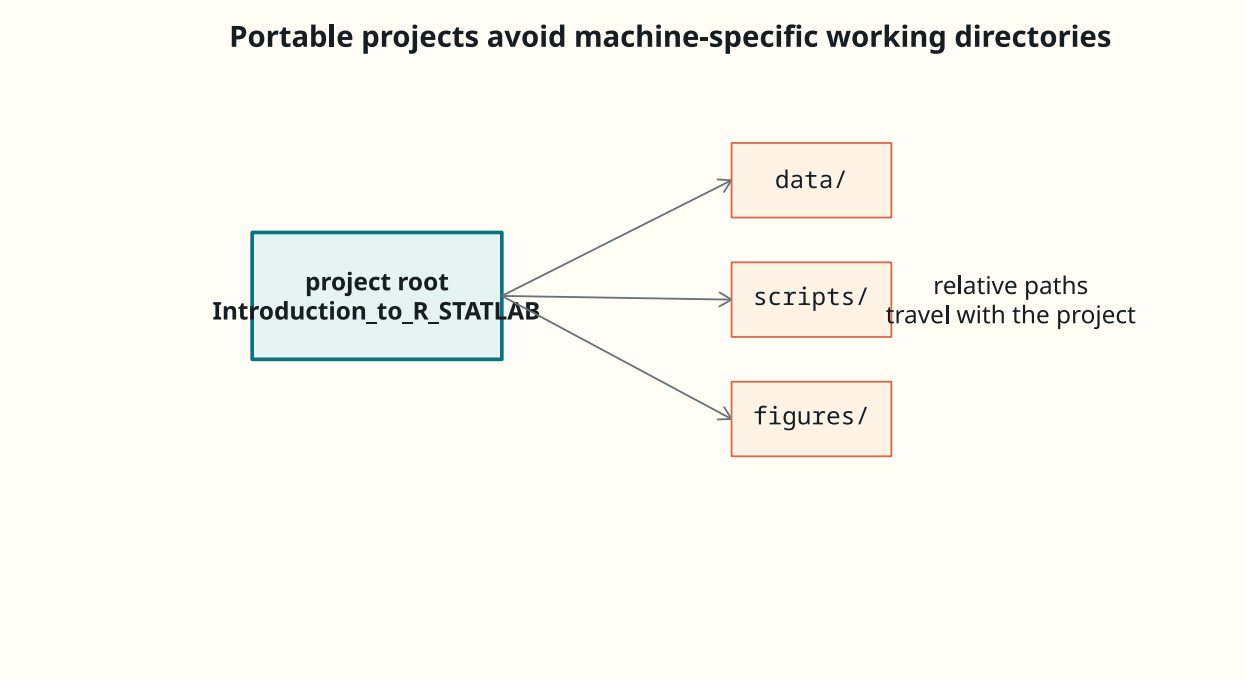

Figure 4.2: Project root connected to data, scripts, and figures through relative paths.

Interpretation. A seed reproduces a pseudorandom stream; it does not make one simulation representative. Relative paths keep the workflow portable across operating systems.

Common mistake

Do not call `install.packages()` inside an analysis script. Installation changes a library; analysis should declare and load dependencies, while setup instructions manage installation separately.

Practical tip

Give functions one job, use verbs for function names, and test normal, boundary, and invalid inputs.

4.7 Guided practice

Write `annualize_volatility(x, periods = 12)` that validates its inputs and returns `sd(x) * sqrt(periods)`. Test it on three constant values and explain the result.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

4.8 Exercises

1. Write a function that converts a decimal return to a percentage and rejects nonnumeric input.
2. Rewrite a nested call using the native pipe, then decide which version is clearer.
3. Capture `sessionInfo()` to a text file in `outputs/` without changing the working directory.

4.9 Selected solutions

Exercise 1.

```
to_percent <- function(x) {
  if (!is.numeric(x)) stop("x must be numeric")
  100 * x
}
```

Exercise 2.

```
capture.output(sessionInfo(), file = file.path("outputs",
 ↪  "session_info.txt"))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

4.10 Chapter summary

- Functions expose inputs, assumptions, and a reusable return value.
- Pipes clarify linear transformations but should not hide complex control flow.
- Pure calculations are easier to test than code with side effects.
- Relative paths, seeds, and session metadata support reproducibility.

4.11 Chapter cheat sheet

Table 4.1: Chapter 4 command cheat sheet.

Command or pattern	Purpose
`function()`	define a reusable rule
`stopifnot()`	assert input conditions
`|>`	native pipe
`file.path()`	portable path construction
`set.seed()`	reproduce random draws
`sessionInfo()`	record runtime versions

CHAPTER 5

Importing, Cleaning, and Validating Data

5.1 Motivation

A model cannot repair an incorrect unit, duplicated key, impossible value, or silently missing observation. Cleaning is therefore an auditable sequence of assertions—not a cosmetic prelude.

5.2 Learning objectives

By the end of this chapter, you should be able to:

- choose an import function for CSV, spreadsheet, and serialized R data
- preserve raw inputs and create cleaned derivatives
- diagnose missingness and outliers
- write validation checks for schema, ranges, and keys
- document transformations and provenance

5.3 Packages and data

`readr`, `readxl`, `dplyr`, and `janitor` are recommended for current workflows; the tested example uses `dplyr` 1.2.1 and the frozen CSV.

The complete tested script is `scripts/ch05_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Validation converts assumptions into executable checks. When a source changes, the script fails near the cause instead of producing a plausible but wrong model later.

5.4 Core ideas

Import is a translation from an external representation into R classes. A CSV contains text and no authoritative type metadata, so the importer must infer columns. Read the parser's specification, supply column types when stability matters, and inspect parsing problems. Spreadsheet files add sheets, formulas, merged cells, and display formatting; treat them as inputs to validate rather than databases to edit manually during analysis.

Preserve the original file unchanged. A cleaning script should create a new table from raw data through named steps: standardize names, parse types, identify sentinels, check keys, apply defensible corrections, and validate the result. A correction should be traceable to a rule or source—not an unexplained hand edit.

Missingness has meaning. `NA` may indicate unavailable, not applicable, suppressed, or failed collection. Counts by variable and time can reveal patterns. Complete-case deletion is valid only when its consequences for the target population are understood. Imputation introduces model assumptions and should be separated from basic cleaning.

An outlier is an observation that deserves attention, not automatically an error. Range rules are most convincing when based on domain constraints: a 99% policy rate in this teaching file conflicts with the source and known scale.

Statistical rules such as 1.5 IQR can flag candidates but cannot decide validity. Validation code should stop early when required columns are missing, dates duplicate, or impossible values remain.

5.5 Worked example 1: Map missing values before deciding what to do

```
dirty <- macro
dirty$inflation_yoy_percent[c(8, 31, 76)] <- NA_real_

colSums(is.na(dirty))
mean(is.na(dirty$inflation_yoy_percent))
```

Result

The constructed example has **3 missing cells**, all in inflation. The map shows their positions rather than hiding them behind a single total.

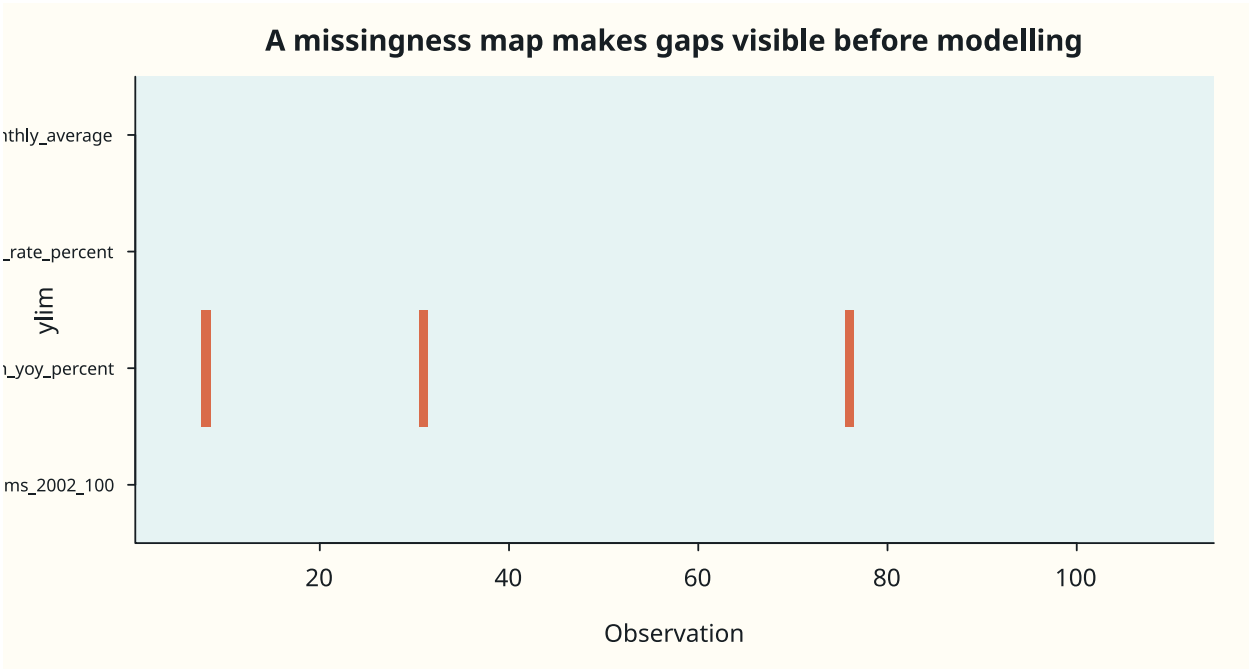

Figure 5.1: Binary missingness map for the constructed cleaning example; coral cells mark missing values.

Interpretation. A sparse pattern may support a different response from a long missing block. The missingness mechanism—not the convenience of `na.rm = TRUE`—should guide the decision.

5.6 Worked example 2: Apply explicit cleaning rules with dplyr

```
library(dplyr)

clean <- dirty |>
  filter(!is.na(inflation_yoy_percent)) |>
  mutate(policy_rate_percent = if_else(
    policy_rate_percent > 20, NA_real_, policy_rate_percent
  ))
```

```
stopifnot(nrow(clean) == 111, !anyDuplicated(clean$date))
```

Result

Filtering complete inflation rows leaves **111 rows**. The deliberately impossible 99% policy rate is replaced with NA pending source correction.

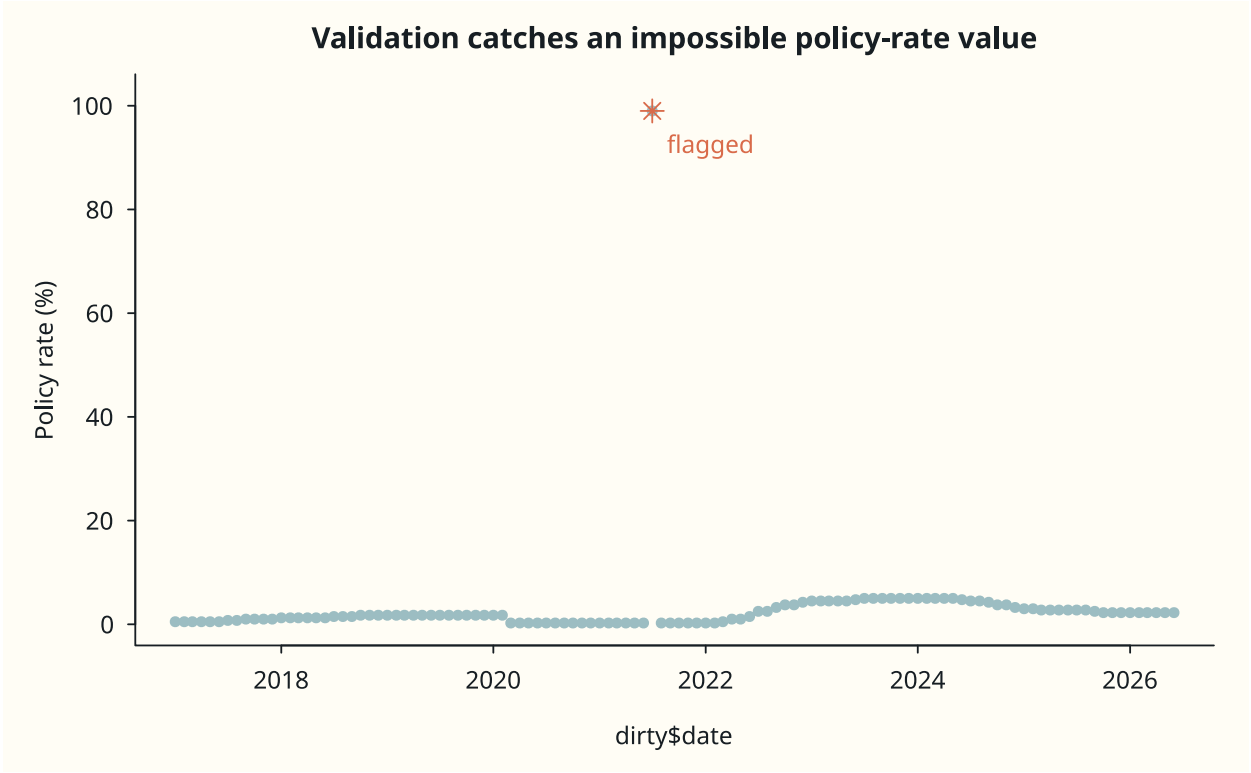

Figure 5.2: Constructed policy-rate data with an impossible 99% value flagged for investigation.

Interpretation. Replacing a suspect value with missingness is safer than inventing a correction. Production work should compare with the official source and record the correction rule.

Common mistake

Do not delete an outlier because it makes a model fit better. First decide whether it is an error, a rare valid event, or evidence that the model is inadequate.

Practical tip

Keep a compact data dictionary beside every teaching dataset: variable name, type, unit, definition, transformation, and source.

5.7 Guided practice

Create a copy of the macro data with one duplicated date. Write checks that detect the duplicate and report the offending rows without deleting them.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

5.8 Exercises

1. Write a schema check for the five required column names.
2. Count missing values by row and by column in a constructed example.
3. Propose a validation rule for USD/CAD, and explain why the limits are domain choices rather than universal constants.

5.9 Selected solutions

Exercise 1.

```
required <- c("date", "cpi_all_items_2002_100", "inflation_yoy_percent",
      "policy_rate_percent", "usd_cad_monthly_average")
stopifnot(setequal(names(macro), required))
```

Exercise 2.

```
row_missing <- rowSums(is.na(dirty))
column_missing <- colSums(is.na(dirty))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

5.10 Chapter summary

- Importers infer types, so inspect their decisions.
- Preserve raw files and express cleaning as code.
- Missingness and outliers require substantive interpretation.
- Schema, key, range, and row-count checks catch errors early.

5.11 Chapter cheat sheet

Table 5.1: Chapter 5 command cheat sheet.

Command or pattern	Purpose
`readr::read_csv()`	typed CSV import
`readxl::read_excel()`	spreadsheet import
`is.na()`	missingness indicator
`colSums()`	column totals
`stopifnot()`	validation assertion
`anyDuplicated()`	duplicate-key check

CHAPTER 6

Transforming Data with dplyr

6.1 Motivation

Most analysis time is spent turning source tables into the rows, columns, and groups that answer a question. `dplyr` supplies a small grammar whose verbs state those transformations directly.

6.2 Learning objectives

By the end of this chapter, you should be able to:

- select, filter, arrange, mutate, and summarize rows and columns
- build grouped summaries and remove grouping deliberately
- use `case_when()` for readable classifications
- join tables while protecting key relationships
- write pipelines that remain easy to inspect

6.3 Packages and data

`dplyr` 1.2.1, tested with R 4.6.0; the Canadian macro dataset.

The complete tested script is `scripts/ch06_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

A consistent data grammar lets reviewers focus on analytical choices rather than decoding ad hoc indexing.

6.4 Core ideas

`dplyr` verbs describe table transformations. `select()` chooses columns, `filter()` keeps rows, `arrange()` changes row order, `mutate()` creates or changes columns, and `summarise()` reduces many rows to statistics. The first argument is a data frame, so these verbs compose naturally with `|>`.

Grouping changes the meaning of later verbs. After `group_by(era)`, `mean(inflation)` is computed separately within each era; `summarise(.groups = "drop")` returns an explicitly ungrouped table. Hidden grouping can cause later mutations or counts to be calculated at the wrong level, so inspect `group_vars()` in longer workflows.

`case_when()` expresses mutually exclusive rules in priority order. Put narrow rules before broad rules and end with an explicit fallback. If categories are part of the analytical design, convert the result to a factor with deliberate levels. `across()` applies the same function to selected columns, which is useful for consistent validation or scaling.

Joins add columns or rows based on keys. Specify the key and expected relationship when the current version supports it. Check unmatched keys with anti-joins and row counts before and after. A join warning is a prompt to investigate data structure, not noise to suppress.

6.5 Worked example 1: Summarize inflation by economic era

```r
library(dplyr)

summaries <- macro |>
  mutate(era = case_when(
    date < as.Date("2020-01-01") ~ "2017-2019",
    date < as.Date("2023-01-01") ~ "2020-2022",
    TRUE ~ "2023-2026"
  )) |>
  group_by(era) |>
  summarise(months = n(),
    mean_inflation = mean(inflation_yoy_percent),
    volatility = sd(inflation_yoy_percent),
    .groups = "drop")
```

Result

The highest era mean is **3.638%**, and the largest within-era standard deviation is **2.711 percentage points**.

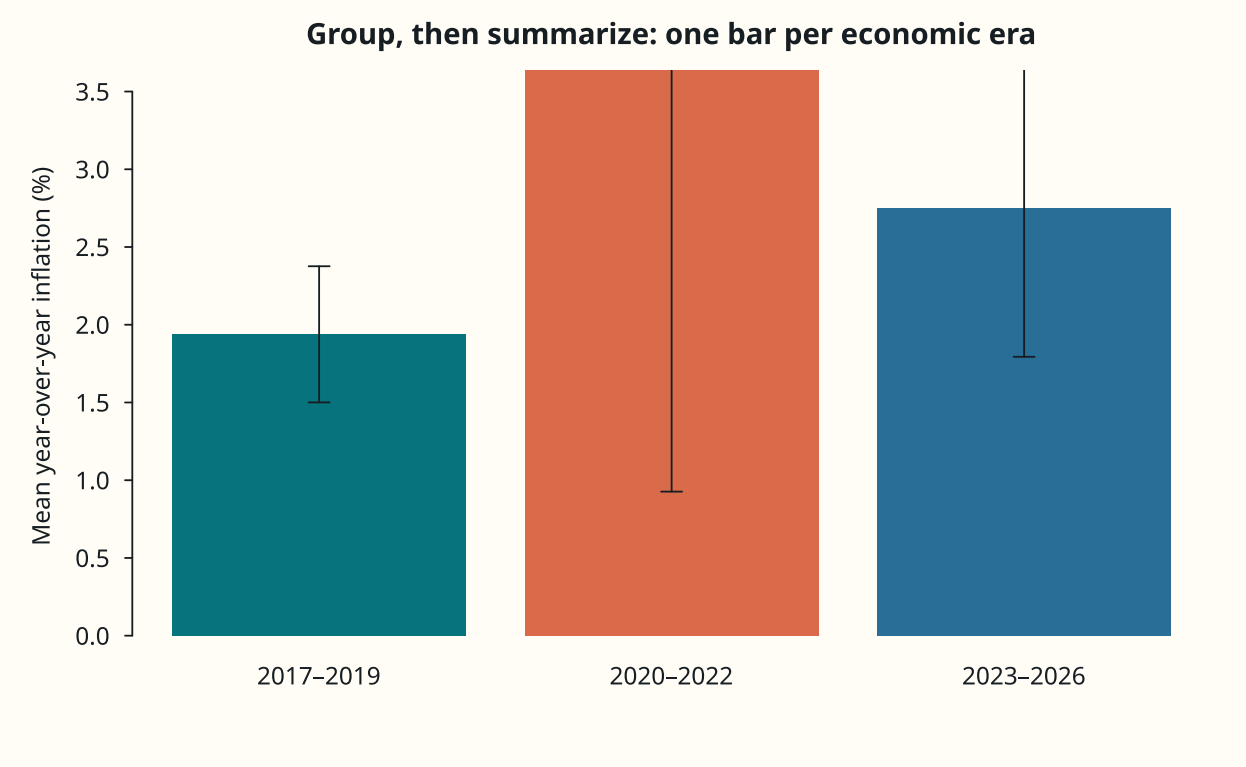

Figure 6.1: Mean inflation by economic era with plus or minus one within-era standard deviation.

Interpretation. The summaries depend on analyst-defined eras and do not isolate causes. Error bars in the figure show one standard deviation, not confidence intervals for the means.

6.6 Worked example 2: Read a pipeline as a sequence of questions

```
recent_summary <- macro |>
  filter(date >= as.Date("2023-01-01")) |>
  mutate(inflation_gap = inflation_yoy_percent - 2) |>
  summarise(months = n(),
    mean_gap = mean(inflation_gap),
    max_gap = max(inflation_gap))

recent_summary
```

Result

The output is a one-row table whose column names preserve the meaning of each result.

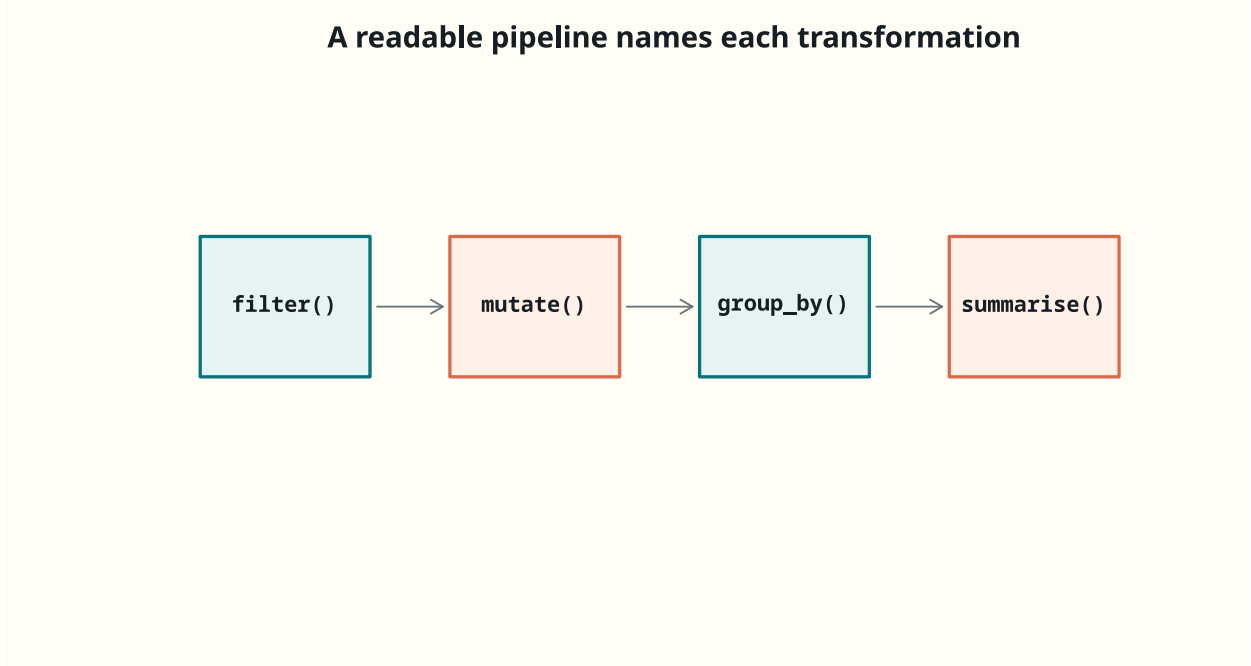

Figure 6.2: Four common dplyr verbs linked in a readable transformation pipeline.

Interpretation. Each verb has one role: define the period, create the quantity of interest, then reduce it. That structure makes review easier than a deeply nested expression.

Common mistake

`filter()` selects rows; `select()` selects columns. Confusing them often produces an error, but reversed base brackets can silently produce the wrong table.

Practical tip

Name important intermediate tables after what they represent, not after a step number: `monthly_rates` is better than `temp2`.

6.7 Guided practice

Filter 2020–2022, create the gap between inflation and policy rate, then summarize its mean, median, minimum, and maximum in one pipeline.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

6.8 Exercises

1. Return the five months with the largest absolute change in USD/CAD.
2. Use `across()` to compute means for all numeric columns.
3. Construct a tiny metadata table keyed by year and join it to `macro`; validate the relationship.

6.9 Selected solutions

Exercise 1.

```
macro |>
  mutate(fx_change = c(NA, diff(usd_cad_monthly_average))) |>
  arrange(desc(abs(fx_change))) |>
  slice_head(n = 5)
```

Exercise 2.

```
macro |>
  summarise(across(where(is.numeric), mean))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

6.10 Chapter summary

- dplyr verbs express row, column, ordering, and aggregation operations.
- Grouping changes the scope of later calculations.
- Pipelines should expose a readable chain of decisions.
- Joins require explicit keys and relationship checks.

6.11 Chapter cheat sheet

Table 6.1: Chapter 6 command cheat sheet.

Command or pattern	Purpose
`filter()`	keep rows
`select()`	keep columns
`mutate()`	create or change columns
`summarise()`	reduce rows
`group_by()`	define groups
`left_join()`	add matched columns

CHAPTER 7

Exploratory Data Analysis

7.1 Motivation

Exploration reveals scale, shape, relationships, and data-quality problems before a model compresses them into coefficients. Good EDA is question-driven and separates discovery from confirmation.

7.2 Learning objectives

By the end of this chapter, you should be able to:

- summarize centre, spread, quantiles, and shape
- choose histograms, densities, boxplots, and scatterplots appropriately
- compare distributions across groups
- identify influential or unusual observations
- avoid treating exploratory patterns as confirmatory evidence

7.3 Packages and data

Base `stats`, `dplyr`, and `ggplot2` 4.0.3; the Canadian macro dataset.

The complete tested script is `scripts/ch07_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

EDA is where analysts notice mismatched units, impossible values, nonlinear relationships, and unstable variance before those problems become model assumptions.

7.4 Core ideas

Centre and spread answer different questions. The mean uses every value and is sensitive to tails; the median is the middle order statistic and is resistant to extremes. Standard deviation measures root-mean-square deviation from the mean, while the interquartile range (IQR) spans the middle half. Always report units and sample period.

A histogram counts observations in bins. Its appearance depends on bin width and boundary placement, so try defensible alternatives. A kernel density estimate smooths observations and introduces a bandwidth choice; it may assign visual density beyond meaningful bounds. Neither is the population distribution itself. An empirical cumulative distribution is often a useful companion because it avoids bins and displays quantiles directly.

Boxplots compactly show median, quartiles, and rule-based whiskers. Points beyond 1.5 IQR are not automatically errors. Side-by-side plots help compare groups, but group sizes and partial periods matter. Scatterplots reveal association, curvature, clusters, and changing variance; overplotting can be reduced with transparency, jitter, bins, or summaries.

EDA generates hypotheses and checks assumptions. If a pattern inspired the model, later p-values calculated on the same data do not have a purely confirmatory interpretation. Holdout data, preregistration, or transparent labelling can preserve the distinction.

7.5 Worked example 1: Compare a histogram with a density estimate

```
library(ggplot2)

ggplot(macro, aes(inflation_yoy_percent)) +
  geom_histogram(aes(y = after_stat(density)), bins = 18) +
  geom_density(linewidth = 1) +
  labs(x = "Inflation (%)", y = "Density")
```

Result

The sample median is **2.237%** and the IQR is **1.464 percentage points**.

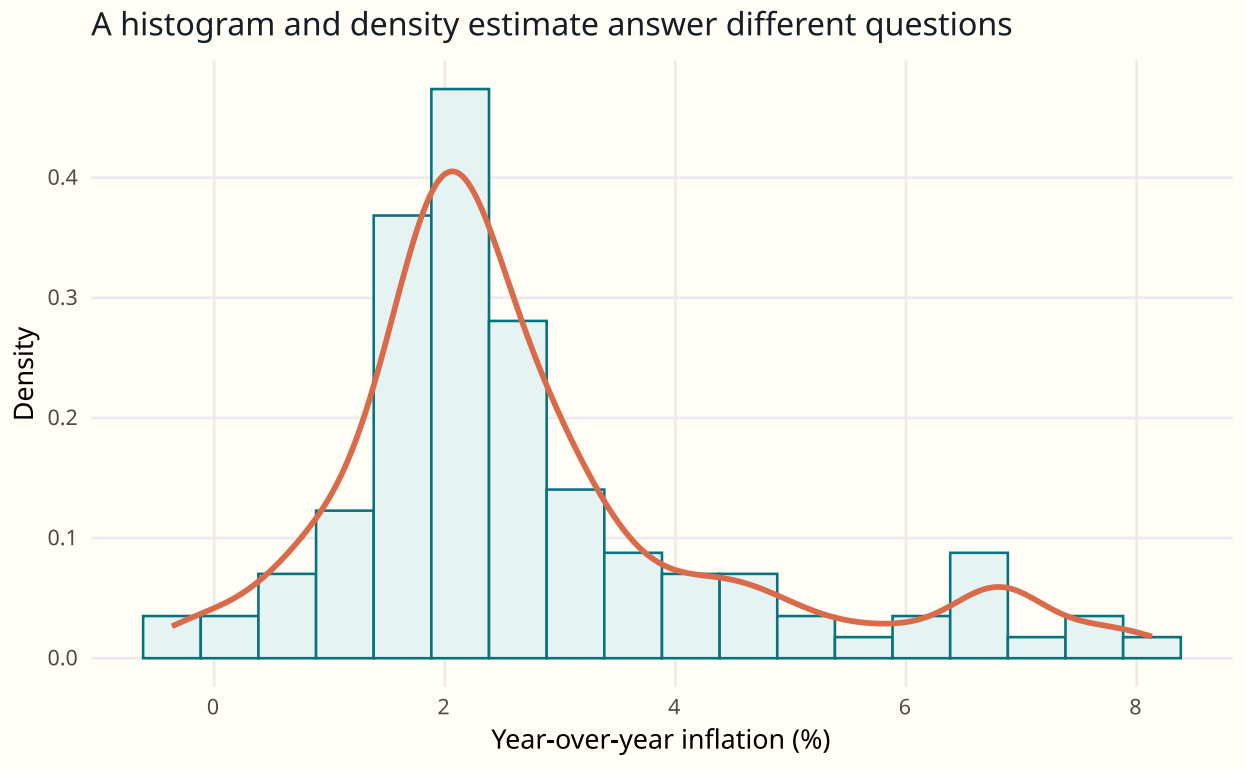

Figure 7.1: Histogram and kernel density estimate of Canadian year-over-year inflation.

Interpretation. The smoothed curve suggests shape, while the histogram preserves count structure. Neither justifies a distributional model without diagnostics.

7.6 Worked example 2: Compare distributions across eras

```
plot_data <- transform(
  macro,
  era = cut(date,
    breaks = as.Date(c("2016-12-31", "2019-12-31",
                       "2022-12-31", "2026-12-31")),
    labels = c("2017–2019", "2020–2022", "2023–2026"))
)
```

```
ggplot(plot_data, aes(era, inflation_yoy_percent, fill = era)) +
  geom_boxplot(show.legend = FALSE)
```

Result

The three boxplots differ in centre, spread, and tail behaviour; the plotted groups also contain different numbers of months.

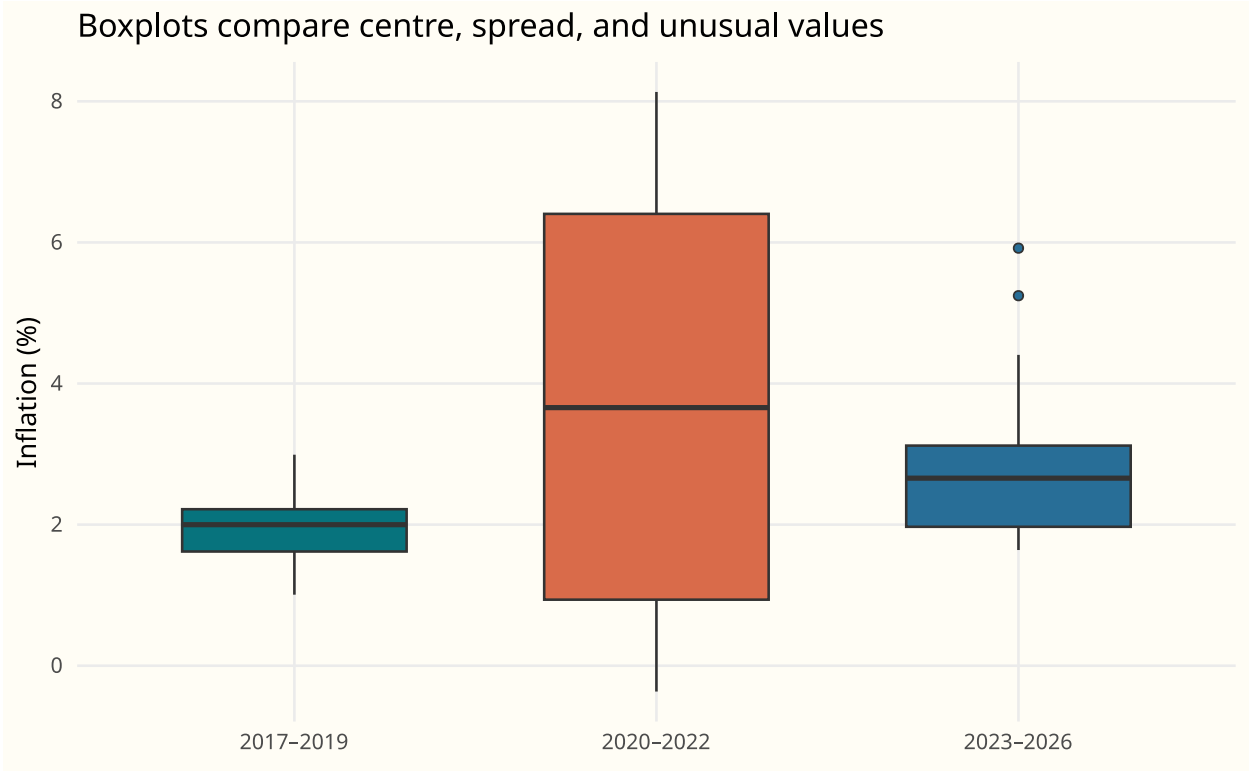

Figure 7.2: Boxplots of Canadian inflation for three analyst-defined economic eras.

Interpretation. The display is descriptive. Era boundaries were chosen by the analyst and observations within an era are serially dependent, so independent-sample comparisons require care.

Common mistake

A narrow density bandwidth can turn sampling noise into apparent modes; a wide one can erase meaningful structure. Show or state the choice.

Practical tip

Use a three-part description: what the plot directly shows, a plausible interpretation, and what additional evidence would be needed.

7.7 Guided practice

Draw an empirical cumulative distribution of inflation. Read approximate 25th, 50th, and 75th percentiles from it and compare with `quantile()`.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

7.8 Exercises

1. Compare policy-rate mean, median, SD, and IQR.
2. Create two inflation histograms with very different bin counts and explain what remains stable.
3. Make a scatterplot of policy rate versus inflation, label the time period, and list two reasons the pattern is not causal evidence.

7.9 Selected solutions

Exercise 1.

```
with(macro, c(mean = mean(policy_rate_percent),
  median = median(policy_rate_percent),
  sd = sd(policy_rate_percent),
  IQR = IQR(policy_rate_percent)))
```

Exercise 2.

```
quantile(macro$inflation_yoy_percent, c(.25, .5, .75))
plot(ecdf(macro$inflation_yoy_percent))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

7.10 Chapter summary

- Describe centre, spread, shape, and sample context together.
- Every statistical graphic has tuning choices.
- Unusual observations require investigation, not automatic removal.
- Exploration and confirmation are different inferential activities.

7.11 Chapter cheat sheet

Table 7.1: Chapter 7 command cheat sheet.

Command or pattern	Purpose
`mean()` / `median()`	centre
`sd()` / `IQR()`	spread
`quantile()`	ordered cut points
`hist()`	binned distribution
`density()`	smoothed distribution
`ecdf()`	empirical cumulative distribution

CHAPTER 8

Data Visualization with ggplot2

8.1 Motivation

A statistical graphic is an argument made with position, scale, colour, and annotation. The grammar of graphics helps construct that argument deliberately and consistently.

8.2 Learning objectives

By the end of this chapter, you should be able to:

- map variables to aesthetics and choose compatible geoms
- build plots in layers
- facet comparisons without hiding scale differences
- use scales, labels, and themes accessibly
- recognize misleading axes, dual scales, and decorative clutter

8.3 Packages and data

`ggplot2` 4.0.3 and base R reshaping for the tested examples; `patchwork` is an optional composition package.

The complete tested script is `scripts/ch08_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Visual encodings determine what comparisons readers can make accurately. Design choices therefore belong to analysis, not post-processing.

8.4 Core ideas

In the grammar of graphics, data and aesthetic mappings define variables such as `x`, `y`, colour, and group. A geom defines marks such as points, lines, or bars. Scales translate data values into positions and visual properties; coordinates define the viewing system; facets create repeated panels; and themes control non-data ink. Separating these components makes intent explicit.

Map a variable inside `aes()`; set a constant outside. `aes(colour = "teal")` creates a category literally named teal, whereas `colour = "#08747D"` sets a fixed colour. Lines need a correct grouping variable, especially after reshaping. Use semantic labels and units rather than raw column names.

Small multiples are often better than a second y-axis. Free scales reveal within-series movement but prevent direct magnitude comparison; fixed scales preserve magnitude but can flatten smaller series. State the choice. When a bar encodes magnitude through length, a zero baseline is normally required. Truncation can exaggerate small differences, as Figure 8.2 demonstrates.

Accessibility is part of statistical correctness. Use colour-blind-aware palettes, sufficient contrast, redundant encodings such as line type, readable text, and captions that state the key message. Alt text should describe chart type, variables, pattern, and notable reference marks without repeating every value.

8.5 Worked example 1: Build aligned small multiples

```
library(ggplot2)

ggplot(long, aes(date, value)) +
  geom_line(colour = "#08747D") +
  facet_wrap(~series, ncol = 1, scales = "free_y") +
  labs(x = NULL, y = NULL) +
  theme_minimal()
```

Result

The figure uses **3 facets**: inflation, policy rate, and USD/CAD. A shared time axis makes turning points easy to align.

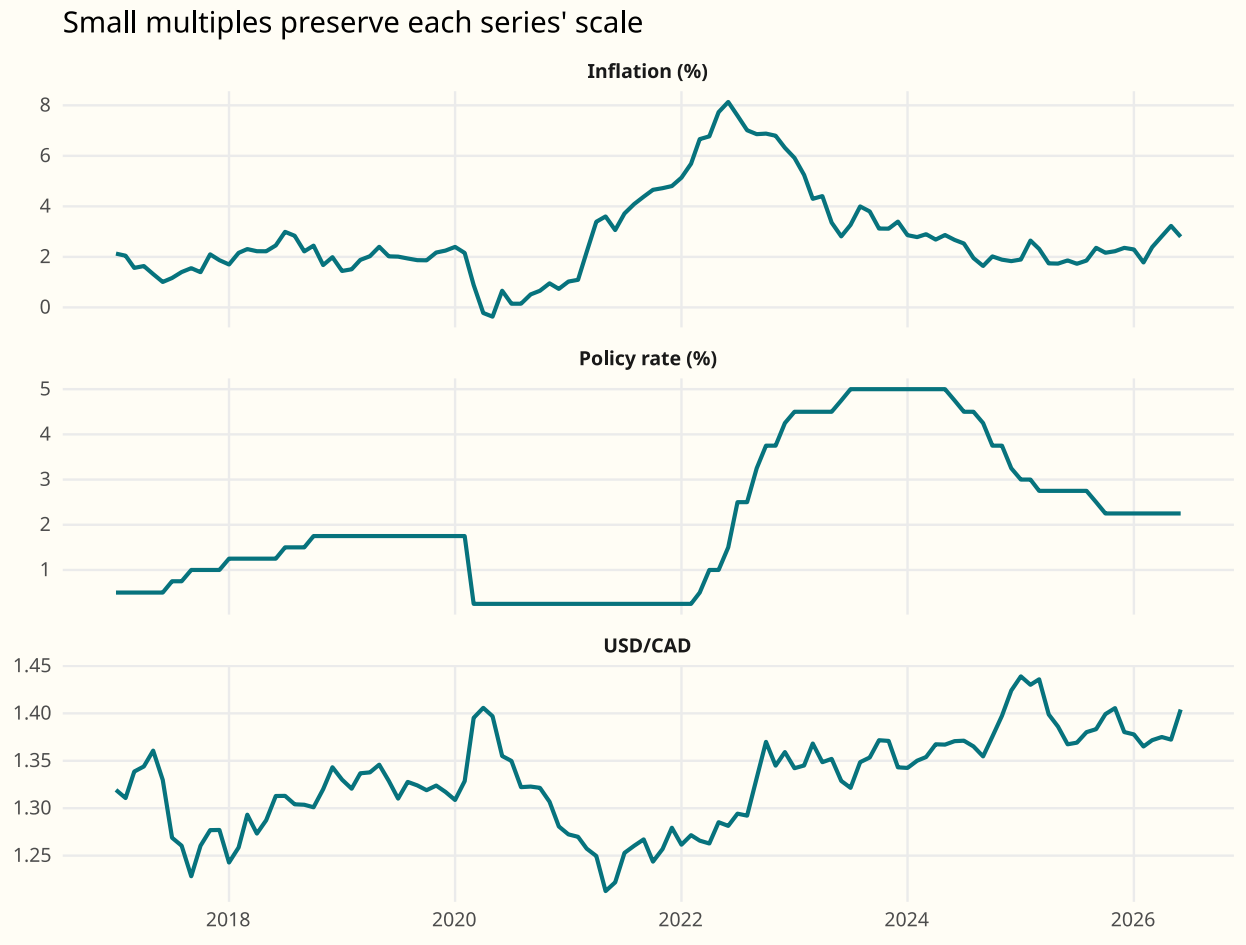

Figure 8.1: Three vertically aligned Canadian macroeconomic time series with separate y-scales.

Interpretation. Free y-scales make the shape of each series visible but do not imply equal numerical variation. Strip labels and axis ticks preserve that distinction.

8.6 Worked example 2: Show how a baseline changes perception

```r
annual <- aggregate(
  inflation_yoy_percent ~ cut(date, "years"),
  data = macro,
  FUN = mean
)

range(annual$inflation_yoy_percent)
```

Result

The annual means span **6.076 percentage points**. Both panels use the same values, but only the left bar chart begins at zero.

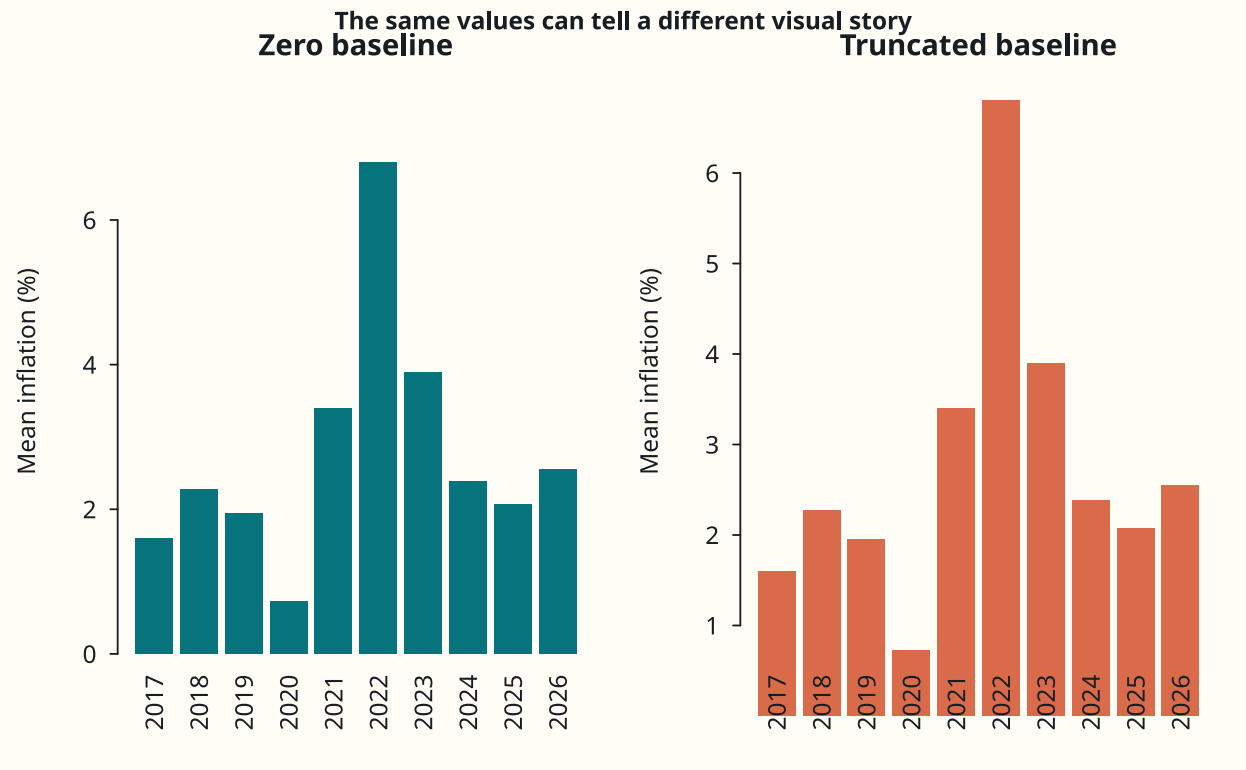

Figure 8.2: The same annual inflation means shown with a zero and a truncated baseline.

Interpretation. The truncated chart is not numerically false, yet bar length invites a ratio interpretation it cannot support. A dot plot is often safer when a nonzero baseline is analytically useful.

Common mistake

Avoid rainbow scales for ordered numerical data: perceived lightness is irregular, nearby values can look unrelated, and common colour-vision deficiencies reduce legibility.

Practical tip

Write the intended one-sentence takeaway before polishing a chart. Remove elements that do not support that reading or an essential qualification.

8.7 Guided practice

Redesign the truncated bar chart as a dot plot with a labelled 2% reference line. Explain which comparisons become easier and which visual emphasis changes.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

8.8 Exercises

1. Create an inflation line chart with a 2% reference and direct label at the final point.
2. Recreate the small multiples with fixed y-scales and compare the message.
3. Draft alt text for the forecast figure in Chapter 12 using chart type, variables, pattern, and uncertainty.

8.9 Selected solutions

Exercise 1.

```
ggplot(macro, aes(date, inflation_yoy_percent)) +
  geom_hline(yintercept = 2, linetype = 2) +
  geom_line(colour = "#08747D") +
  labs(x = NULL, y = "Inflation (%)")
```

Exercise 2.

```
# Change the facet scale decision:
facet_wrap(~series, ncol = 1, scales = "fixed")
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

8.10 Chapter summary

- A plot combines data, mappings, geoms, scales, coordinates, facets, and theme.
- Map variables inside `aes()` and set constants outside.
- Axis and facet choices affect permitted comparisons.
- Accessible labels, contrast, and redundant encodings improve accuracy.

8.11 Chapter cheat sheet

Table 8.1: Chapter 8 command cheat sheet.

Command or pattern	Purpose
`ggplot()`	initialize a graphic
`aes()`	map variables
`geom_point()`	scatterplot marks
`geom_line()`	ordered paths
`facet_wrap()`	small multiples
`labs()`	titles and units

CHAPTER 9

Statistical Foundations and Linear Regression

9.1 Motivation

Statistical models connect observed data to uncertain quantities. This chapter builds the language of sampling, estimation, intervals, tests, and regression while keeping assumptions visible.

9.2 Learning objectives

By the end of this chapter, you should be able to:

- distinguish populations, samples, estimands, estimates, and standard errors
- interpret confidence intervals and p-values accurately
- fit and diagnose a simple linear regression
- separate association, prediction, and causation
- evaluate residual assumptions rather than relying on one summary statistic

9.3 Packages and data

Base `stats`; bootstrap simulation; Canadian inflation and policy-rate data.

The complete tested script is `scripts/ch09_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Clear estimands and assumptions prevent mechanical output from being mistaken for evidence about a different population, time period, or causal question.

9.4 Core ideas

An estimand is the population quantity a study seeks; an estimate is a statistic calculated from observed data. Repeated samples would give different estimates, and their distribution is the sampling distribution. The standard error estimates the spread of that distribution. A bootstrap approximates it by resampling observed units under assumptions about how those units represent the population.

A 95% confidence interval is produced by a procedure that covers the target in 95% of repeated applications under the model. It is not generally a 95% posterior probability that a fixed parameter lies in one realized interval. A p-value is the probability, under the null model and at least as extreme a test statistic, of data as or more incompatible with that model. It is neither the probability the null is true nor an effect-size measure.

Simple regression writes

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i.$$

Least squares chooses coefficients minimizing $\sum_i e_i^2$. The slope is the expected change in Y associated with a one-unit change in X under the model. Residual plots assess linearity, constant variance, unusual influence, and dependence. Time-ordered data often violate independent-error assumptions, which Part II addresses.

R^2 is the fraction of sample variation around the mean explained by fitted values. A high R^2 does not establish a correct, causal, or useful model; a low value does not make a precisely estimated association irrelevant. Prediction must be evaluated on new or held-out data.

9.5 Worked example 1: Approximate a sampling distribution by resampling

```
set.seed(909)
sample_means <- replicate(
  2500,
  mean(sample(macro$inflation_yoy_percent, 24, replace = TRUE))
)

quantile(sample_means, c(0.025, 0.5, 0.975))
```

Result

The resampled means form a much tighter distribution than the individual monthly values because each statistic averages 24 observations.

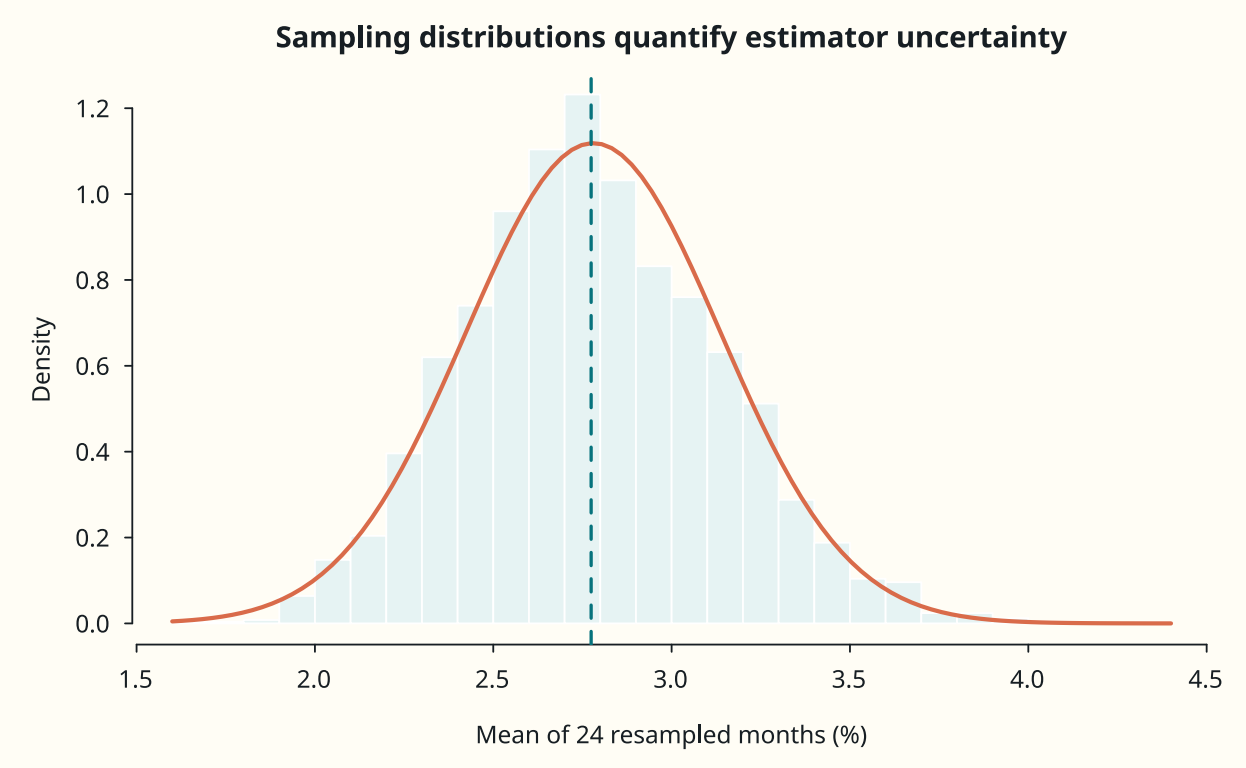

Figure 9.1: Bootstrap distribution of means from 2,500 samples of 24 months, with a normal approximation and full-sample mean.

Interpretation. Serial dependence in monthly inflation means this independent bootstrap is pedagogical, not the preferred time-series method. A block bootstrap or time-series model would preserve dependence.

9.6 Worked example 2: Fit and interpret a simple regression

```
model <- lm(
  inflation_yoy_percent ~ policy_rate_percent,
  data = macro
)

coef(model)
summary(model)$r.squared
```

Result

The fitted slope is **0.268 inflation percentage points per policy-rate point** and R^2 is **0.0597**.

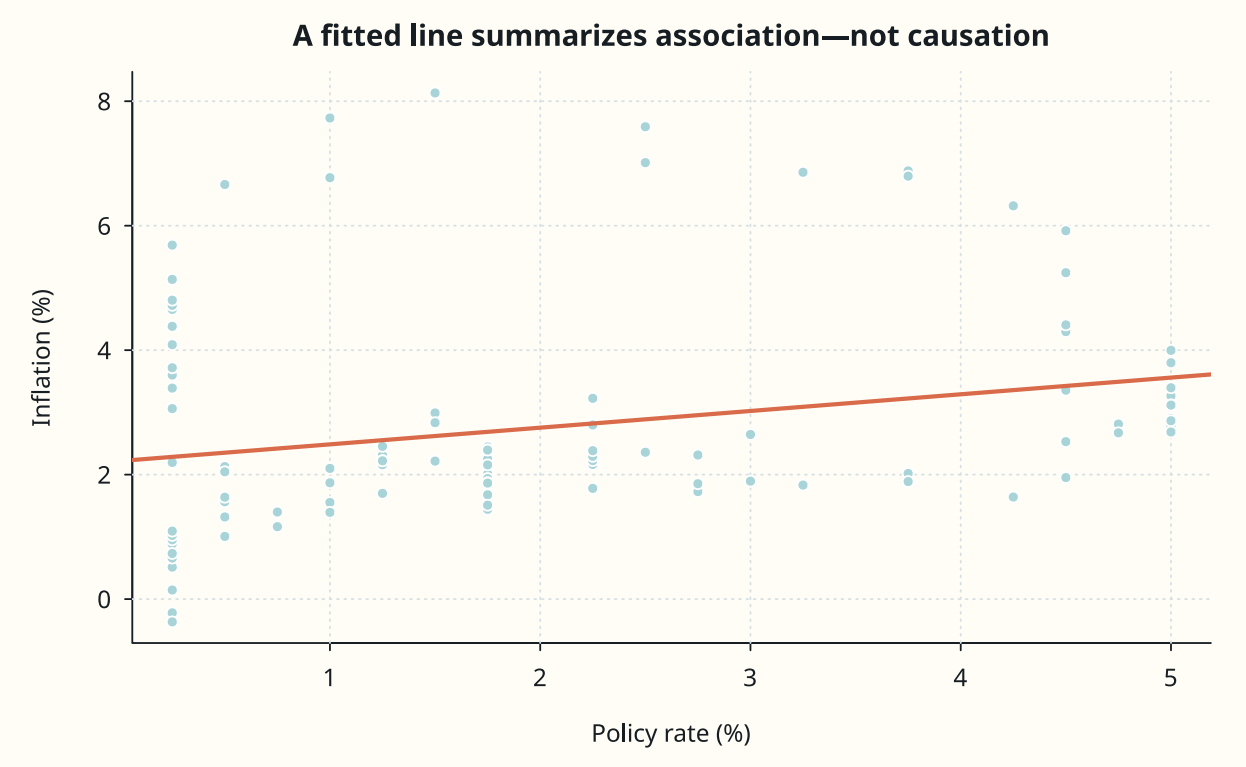

Figure 9.2: Scatterplot of Canadian inflation and the policy rate with an ordinary least-squares line.

Interpretation. The positive same-month association is not the causal effect of monetary policy. Policy reacts to inflation, both series are persistent, lags matter, and omitted variables remain.

Common mistake

Statistical significance does not imply practical importance, and a nonsignificant estimate is not proof of no effect. Report magnitude, uncertainty, design, and diagnostics.

Practical tip

Write an interpretation with units and scope: "Within this monthly sample, a one-point higher X is associated with an estimated Y-point difference in Y."

9.7 Guided practice

Fit inflation on USD/CAD. Plot residuals against fitted values and time. Identify at least two assumptions that the time plot makes questionable.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

9.8 Exercises

1. Calculate a 95% confidence interval for the policy-rate slope and interpret the procedure.
2. Add a quadratic policy-rate term and compare residual plots, not just R^2.
3. Explain why randomly shuffling a time series before a train/test split can produce misleading forecast evaluation.

9.9 Selected solutions

Exercise 1.

```
confint(model, "policy_rate_percent", level = 0.95)
```

Exercise 2.

```
quadratic <- lm(inflation_yoy_percent ~ policy_rate_percent +
          I(policy_rate_percent^2), data = macro)
par(mfrow = c(1, 2))
plot(model, which = 1)
plot(quadratic, which = 1)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

9.10 Chapter summary

- Estimands, estimates, and standard errors answer different questions.
- Intervals and p-values depend on a sampling model.
- Regression slopes describe conditional association under assumptions.
- Residual diagnostics and honest evaluation matter more than one fit statistic.

9.11 Chapter cheat sheet

Table 9.1: Chapter 9 command cheat sheet.

Command or pattern	Purpose
`sample()`	resample values
`replicate()`	repeat a computation
`lm()`	fit linear regression
`coef()`	extract coefficients
`confint()`	confidence intervals
`residuals()`	model residuals

Part II

Part II: Time-Series Foundations

CHAPTER 10

Time-Series Objects, Trend, and Decomposition

10.1 Motivation

Time adds order, dependence, seasonality, and the possibility that relationships change. Treating monthly observations as an unordered sample discards the structure needed for forecasting and valid uncertainty.

10.2 Learning objectives

By the end of this chapter, you should be able to:

- recognize time index, frequency, spacing, and aggregation choices
- construct and inspect regular `ts` objects
- distinguish levels, growth rates, trend, seasonality, and remainder
- apply and interpret STL decomposition
- avoid leakage when smoothing or decomposing forecasting data

10.3 Packages and data

Base `stats`; optional modern alternatives `tsibble`, `fable`, and `feasts`; monthly Canadian CPI from Statistics Canada (Statistics Canada, 2026).

The complete tested script is `scripts/ch10_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

A valid time index prevents future observations from entering past features and makes seasonal, lagged, and rolling operations meaningful.

10.4 Core ideas

A time series is a sequence y_t indexed in time. Its frequency describes how observations repeat within a seasonal cycle: 12 for monthly data, 4 for quarterly, and so on. Frequency is metadata, not proof that every date is present. Validate spacing before constructing a regular `ts`; an irregular business-day series should retain actual dates or use classes such as `xts`.

Levels and changes answer different questions. CPI is an index level; year-over-year inflation is approximately $100[\log(CPI_t) - \log(CPI_{t-12})]$. A rising level may coexist with declining inflation because the price index is still increasing, but at a slower rate. Log changes are additive across time and approximate percentage changes when movements are small.

An additive decomposition writes

$$y_t = T_t + S_t + R_t,$$

where T_t is trend, S_t seasonality, and R_t remainder. A multiplicative pattern can often be handled by applying an additive decomposition to $\log y_t$. STL uses locally weighted regressions, permits changing trend, and can use robust fitting to reduce the influence of isolated disturbances. Its components depend on smoothing choices and should not be treated as uniquely observed truths.

Smoothers use neighbouring observations. A centred moving average at time t uses future values and therefore leaks information in a real-time forecast setting. Exploratory decomposition may use the full sample, but forecast evaluation must refit transformations using training data only.

10.5 Worked example 1: Compare CPI levels with inflation changes

```
macro <- read.csv("data/canada_macro_monthly.csv")
macro$date <- as.Date(macro$date)
cpi_ts <- ts(macro$cpi_all_items_2002_100,
     start = c(2017, 1), frequency = 12)

frequency(cpi_ts)
100 * (tail(cpi_ts, 1) / cpi_ts[1] - 1)
```

Result

The series has monthly frequency 12. CPI rose **30.502%** from the first to final frozen observation, while year-over-year inflation moved above and below that long-run average pace.

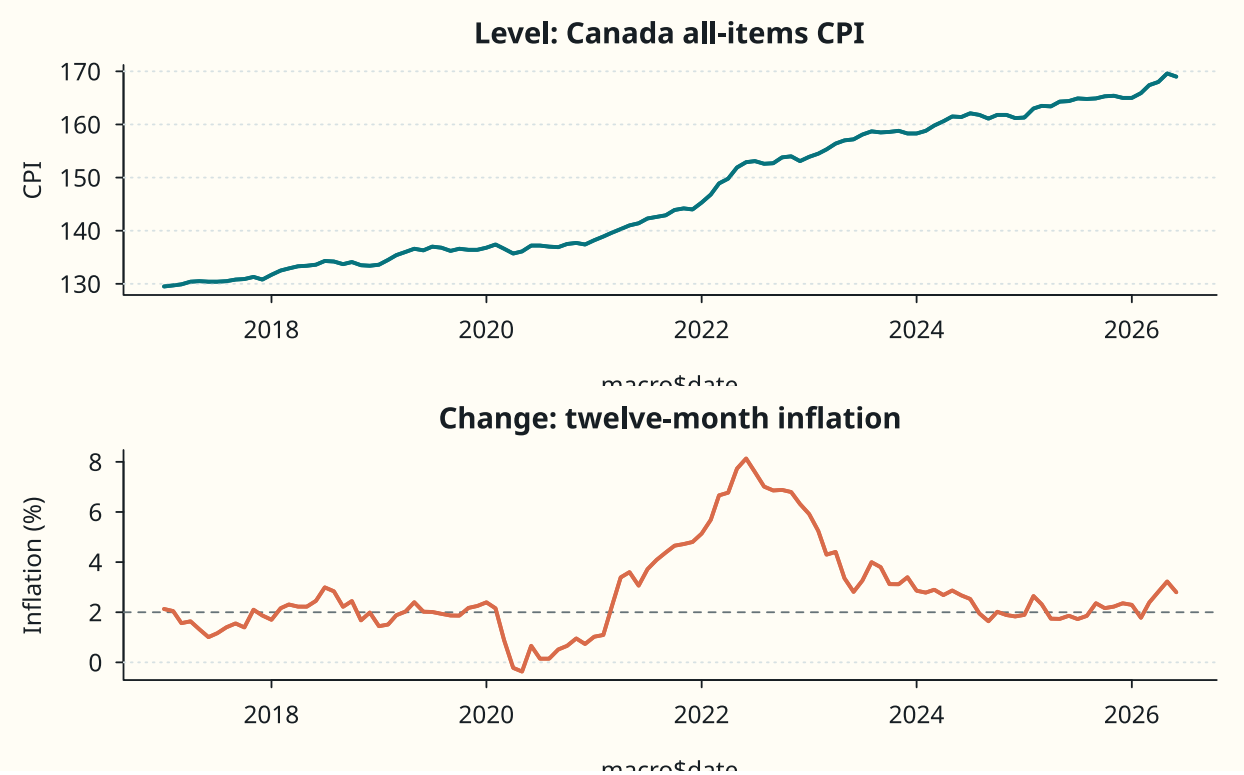

Figure 10.1: Canadian CPI level and twelve-month inflation shown in aligned panels.

Interpretation. A cumulative price-level increase is not the same quantity as the latest twelve-month inflation rate. State the horizon and denominator whenever reporting a percentage change.

10.6 Worked example 2: Decompose log CPI with robust STL

```
decomposition <- stl(
  log(cpi_ts),
  s.window = "periodic",
  robust = TRUE
)

head(decomposition$time.series)
sd(decomposition$time.series[, "remainder"])
```

Result

The remainder standard deviation is **0.00339 log points** in this fitted decomposition.

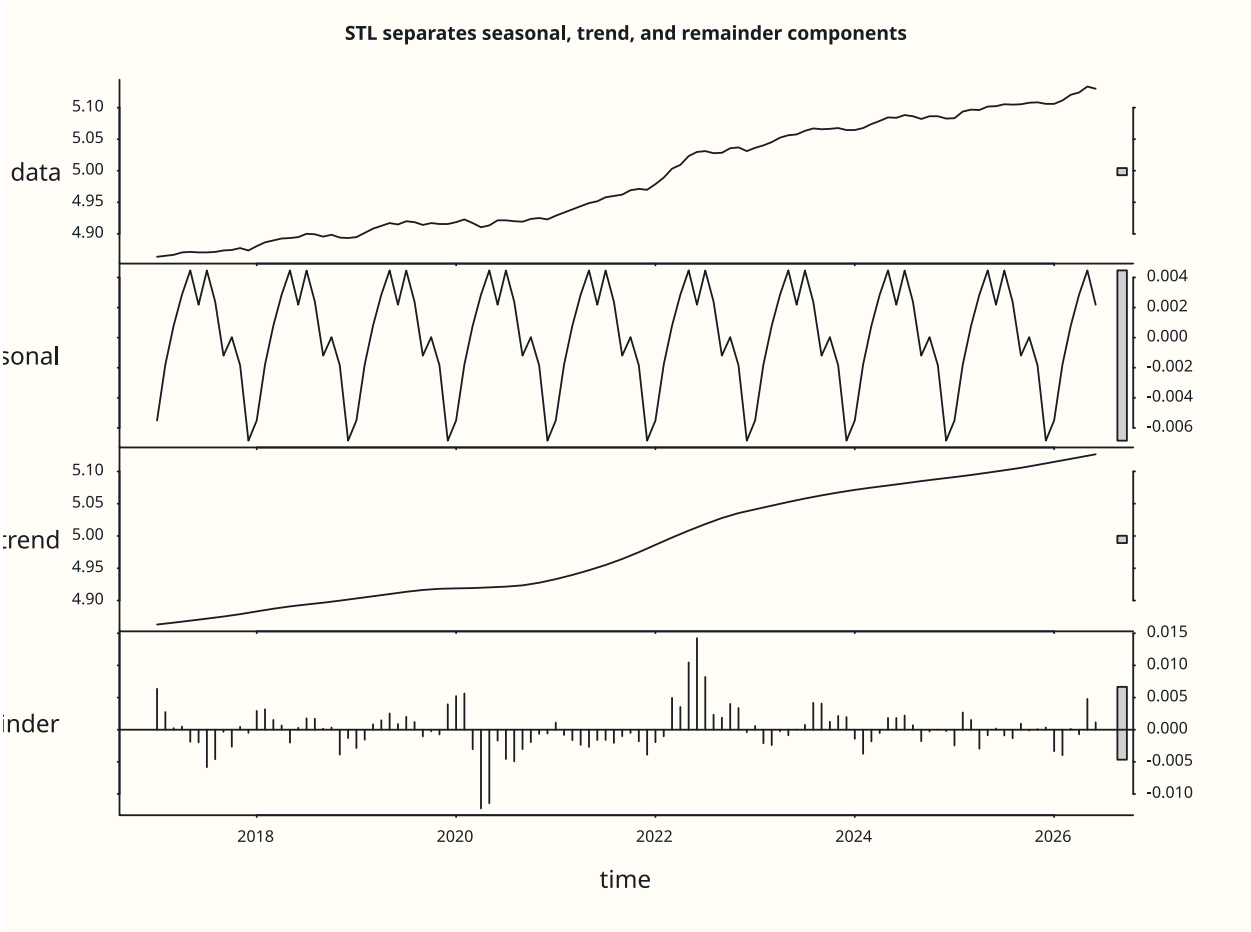

Figure 10.2: Robust STL decomposition of log Canadian CPI into data, seasonal, trend, and remainder panels.

Interpretation. Periodic seasonality assumes the same calendar pattern each year. Robust weighting limits isolated effects but does not make the components causal or immune to end-point uncertainty.

Common mistake

Do not infer a seasonal frequency from the number of rows. Missing months can make an apparently regular series misaligned.

Practical tip

Plot both the level and a substantively meaningful change. Many misleading time-series claims come from switching between them without notice.

10.7 Guided practice

Construct a quarterly CPI series by taking the final month of each quarter. Compare that choice with quarterly averaging and explain which question each answers.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

10.8 Exercises

1. Compute monthly and twelve-month log CPI changes.
2. Use `aggregate.ts()` to form annual average CPI and compare it with the December level.
3. Repeat STL with a nonperiodic seasonal window and describe visible differences at the sample ends.

10.9 Selected solutions

Exercise 1.

```
monthly_change <- 100 * diff(log(cpi_ts))
yoy_change <- 100 * diff(log(cpi_ts), lag = 12)
```

Exercise 2.

```
annual_average <- aggregate(cpi_ts, nfrequency = 1, FUN = mean)
december_level <- window(cpi_ts, start = c(2017, 12), deltat = 1)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

10.10 Chapter summary

- Time series require an ordered, validated index and meaningful frequency.
- Levels, differences, and growth rates answer distinct questions.
- STL produces model-dependent trend, seasonal, and remainder components.
- Future information must not enter real-time transformations.

10.11 Chapter cheat sheet

Table 10.1: Chapter 10 command cheat sheet.

Command or pattern	Purpose
`ts()`	regular time-series object
`frequency()`	seasonal frequency
`lag()`	time shift

Command or pattern	Purpose
`diff()`	first or seasonal difference
`stl()`	seasonal-trend decomposition
`window()`	time interval subset

CHAPTER 11

Stationarity, Autocorrelation, AR, and MA Models

11.1 Motivation

Forecasting depends on how information persists. Stationarity, autocorrelation, and simple AR/MA models provide a compact language for that persistence and a baseline against which richer models can be judged.

11.2 Learning objectives

By the end of this chapter, you should be able to:

- define weak stationarity and white noise
- calculate and interpret ACF and PACF
- state stationarity and invertibility conditions
- fit AR and MA models
- distinguish unit-root evidence from visual persistence

11.3 Packages and data

Base `stats`; `tseries` and `urca` are optional for formal unit-root tests; reproducible simulated series.

The complete tested script is `scripts/ch11_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Stationarity makes the past informative about the future through stable probabilistic relationships. Without it, historical averages may not represent the forecast origin.

11.4 Core ideas

A weakly stationary process has constant mean, finite constant variance, and autocovariance depending only on lag. White noise has mean zero, constant variance, and no serial correlation; independence is stronger than zero correlation. Financial returns can be nearly uncorrelated while their squares remain dependent.

The autocorrelation function is $\rho_k = \text{Corr}(y_t, y_{t-k})$. The sample ACF estimates it at each lag. The partial autocorrelation at lag k measures the relationship remaining after accounting for intervening lags. Approximate horizontal bounds in an ACF plot are pointwise heuristics, not simultaneous tests across all displayed lags.

An AR(1) model is

$$y_t = c + \phi y_{t-1} + \varepsilon_t.$$

It is stationary when $|\phi| < 1$; shocks decay geometrically at rate ϕ^h. An MA(1), $y_t = \mu + \varepsilon_t + \theta\varepsilon_{t-1}$, has ACF zero beyond lag 1 and is invertible under the conventional parameterization when $|\theta| < 1$. ARMA models combine both patterns; root conditions generalize these rules.

A unit root ($\phi = 1$) implies nonstationarity and persistent shocks. Augmented Dickey–Fuller tests use a null of a unit root and are sensitive to deterministic terms and lag choice. KPSS reverses the null toward stationarity. Tests should accompany plots, subject-matter reasoning, and awareness of structural breaks.

11.5 Worked example 1: Simulate white noise and a persistent AR(1)

```r
set.seed(1111)
white_noise <- rnorm(180)
ar_series <- arima.sim(
  model = list(ar = 0.72),
  n = 180,
  sd = 1
)

cor(head(ar_series, -1), tail(ar_series, -1))
```

Result

The simulated AR path shows runs above and below its mean, while white noise changes direction more freely. The sample lag-one ACF is **0.691**.

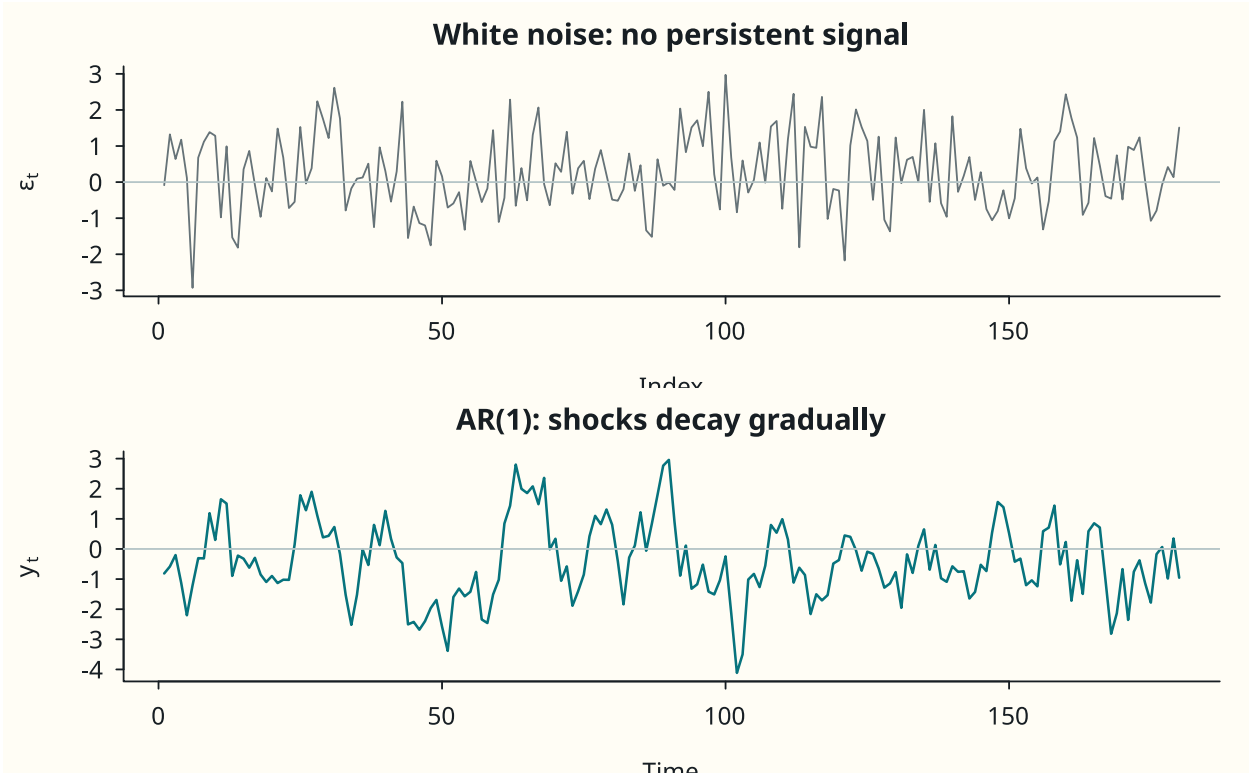

Figure 11.1: Simulated white-noise and stationary AR(1) paths using the same time length.

Interpretation. Finite-sample autocorrelation will not equal the generating value exactly. A seed makes this particular discrepancy reproducible.

11.6 Worked example 2: Use ACF and PACF as model diagnostics

```r
acf(ar_series, lag.max = 18)
pacf(ar_series, lag.max = 18)

fit <- arima(ar_series, order = c(1, 0, 0))
coef(fit)["ar1"]
```

Result

The fitted AR coefficient is **0.689**, close to the generating value 0.72. The ACF decays and the PACF is dominated by lag 1.

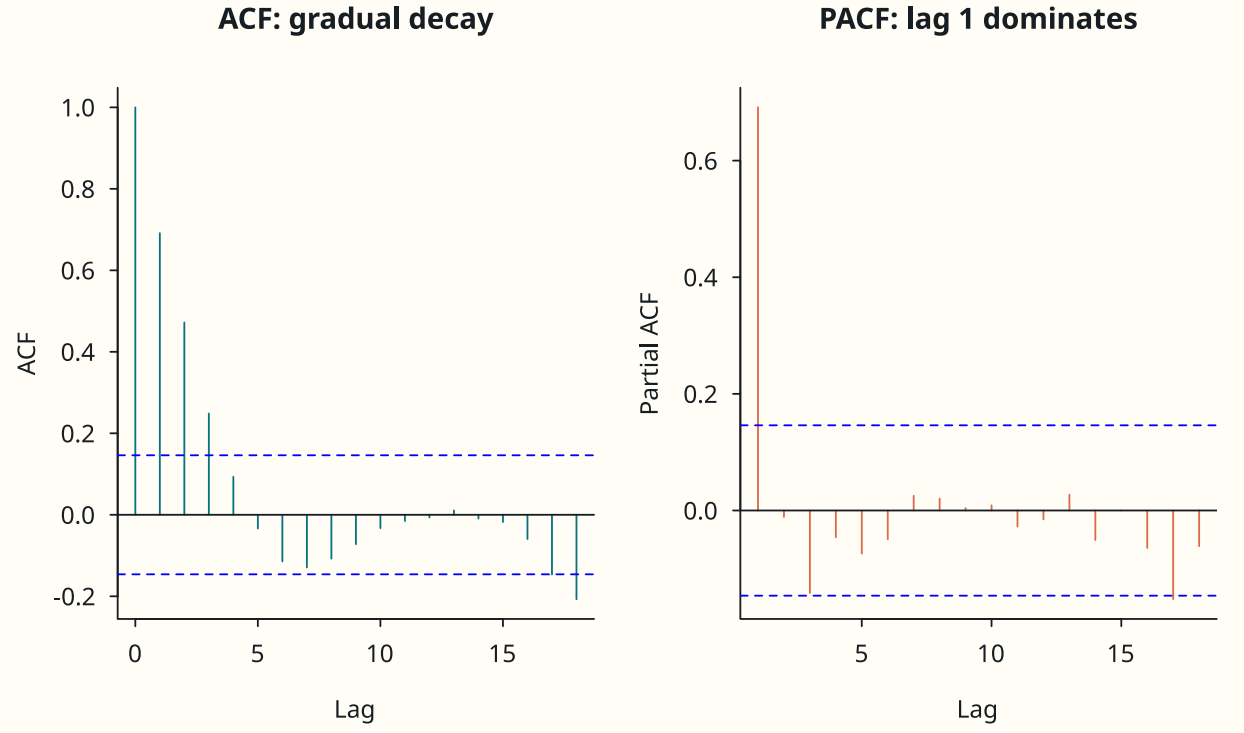

Figure 11.2: Sample ACF and PACF of the simulated AR(1) series.

Interpretation. ACF/PACF patterns suggest candidates; they do not replace likelihood, diagnostics, and forecast evaluation. Sampling noise can make several models look plausible.

Common mistake

Failing to reject a unit root is not proof that a unit root is present; these tests often have low power in short persistent series.

Practical tip

After choosing a candidate model from ACF/PACF, check residual ACF and rolling forecast accuracy. Identification is iterative.

11.7 Guided practice

Simulate MA(1) series with $\theta = 0.6$ and -0.6. Compare their lag-one ACF signs and show that later autocorrelations are near zero.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

11.8 Exercises

1. Simulate AR(1) paths for $\phi = 0.2$, 0.8, and 1.0 and compare persistence.
2. Calculate the theoretical half-life $\log(0.5)/\log(|\phi|)$ for the fitted AR coefficient.
3. Explain why differencing a stationary series can create unnecessary negative autocorrelation.

11.9 Selected solutions

Exercise 1.

```
set.seed(11)
paths <- lapply(c(.2, .8, 1), function(phi) {
  if (phi == 1) cumsum(rnorm(200)) else arima.sim(list(ar = phi), 200)
})
```

Exercise 2.

```
phi <- unname(coef(fit)["ar1"])
log(0.5) / log(abs(phi))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

11.10 Chapter summary

- Weak stationarity stabilizes mean, variance, and lag covariance.
- ACF and PACF summarize different forms of lag dependence.
- AR stationarity and MA invertibility are root conditions.
- Unit-root tests require explicit deterministic terms and careful interpretation.

11.11 Chapter cheat sheet

Table 11.1: Chapter 11 command cheat sheet.

Command or pattern	Purpose
`acf()`	sample autocorrelation
`pacf()`	partial autocorrelation
`arima.sim()`	simulate ARIMA process
`arima()`	estimate ARMA/ARIMA
`Box.test()`	portmanteau residual test
`urca::ur.df()`	augmented Dickey–Fuller test

CHAPTER 12

ARIMA Estimation, Diagnostics, and Forecasting

12.1 Motivation

An ARIMA model turns stationarity decisions and lag dependence into probabilistic forecasts. Its value comes from an auditable modelling loop: specify, estimate, diagnose, forecast, and evaluate out of sample.

12.2 Learning objectives

By the end of this chapter, you should be able to:

- interpret ARIMA(p,d,q) notation
- estimate a model by maximum likelihood
- check residual independence and distributional assumptions
- construct point and interval forecasts
- compare forecasts on a time-ordered holdout set

12.3 Packages and data

Base `stats`; optional `forecast` 9.0.2 and `fable` 0.5.0 for production workflows; Canadian inflation.

The complete tested script is `scripts/ch12_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Out-of-sample evaluation measures the task a forecast is intended to perform. In-sample fit alone rewards models for explaining observations they already saw.

12.4 Core ideas

ARIMA(p, d, q) applies d differences and models the result with p autoregressive and q moving-average terms. In backshift notation,

$$\phi(B)(1 - B)^d y_t = c + \theta(B)\varepsilon_t.$$

Seasonal ARIMA adds analogous terms at seasonal lag s. Differencing should remove stochastic trend without erasing predictable structure; information criteria and diagnostics guide p and q after that decision.

Maximum likelihood estimates parameters jointly with an innovation variance. AIC rewards fit but penalizes parameter count; AICc adds a finite-sample penalty. Comparisons require the same response data and likelihood treatment. An automatic search is a candidate generator, not permission to skip diagnostics or substantive constraints.

Residuals should resemble innovations: mean near zero, stable variance, and no remaining serial correlation. A Ljung–Box statistic aggregates residual autocorrelations through a selected lag. Its reference distribution depends

on estimated degrees of freedom and the test may reject minor imperfections in large samples or miss important structure in small ones. Plot residuals, ACF, distribution, and squared residuals.

A point forecast minimizes a specified expected loss; under squared error it is the conditional mean. A normal 95% interval is $\hat{y}_{T+h} \pm 1.96\, SE_h$ if the approximation is appropriate. Forecast uncertainty typically widens with horizon. Evaluate all preprocessing and model fitting using training data only, and benchmark against naive forecasts.

12.5 Worked example 1: Hold out the final twelve months

```
holdout <- 12
train <- head(macro$inflation_yoy_percent, -holdout)
test <- tail(macro$inflation_yoy_percent, holdout)

fit <- arima(train, order = c(1, 1, 1), method = "ML")
fc <- predict(fit, n.ahead = holdout)
sqrt(mean((test - fc$pred)^2))
```

Result

The twelve-month holdout RMSE is **0.624 percentage points**.

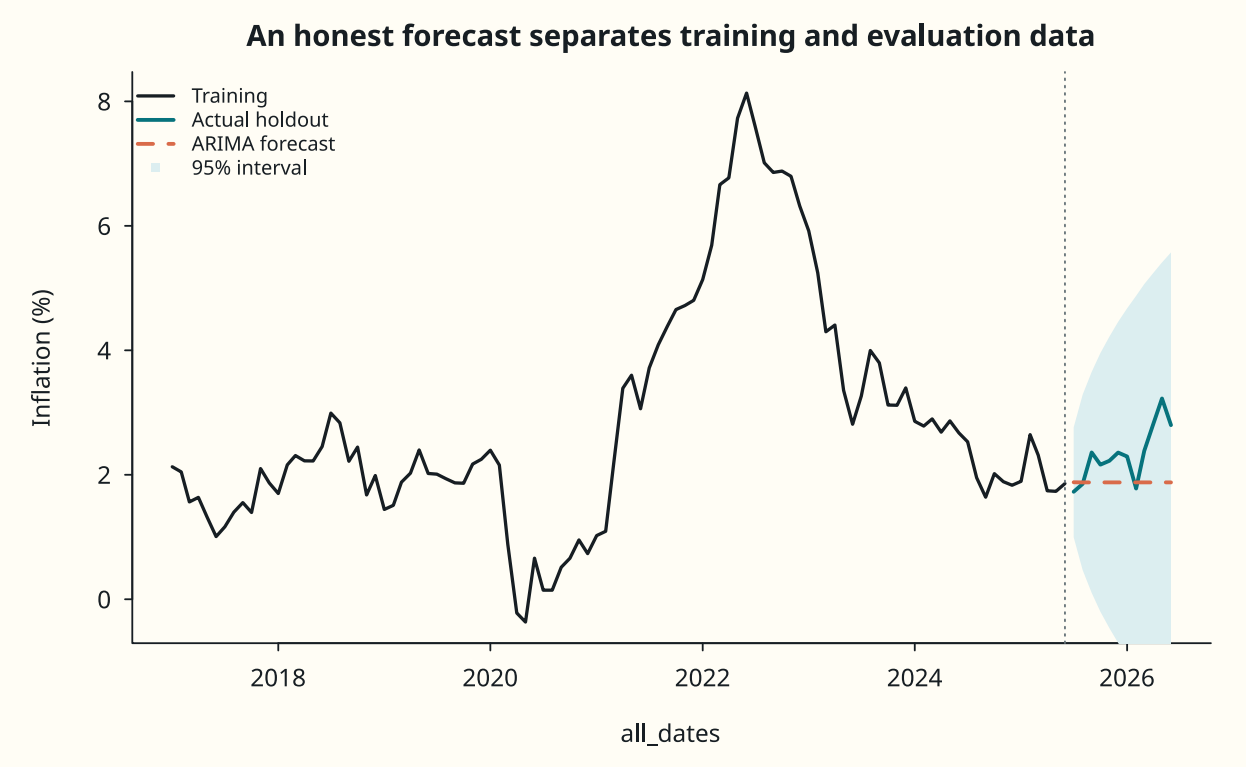

Figure 12.1: Training data, actual holdout inflation, ARIMA forecasts, and model-based 95% intervals.

Interpretation. One holdout period is transparent but uncertain. A rolling-origin comparison across several forecast origins is more robust when enough history is available.

12.6 Worked example 2: Diagnose model residuals

```
res <- residuals(fit)
acf(res, lag.max = 18, na.action = na.pass)

Box.test(res, lag = 12, type = "Ljung-Box", fitdf = 2)
```

Result

The Ljung–Box p-value is **0.00135**, providing evidence that this compact specification leaves serial dependence.

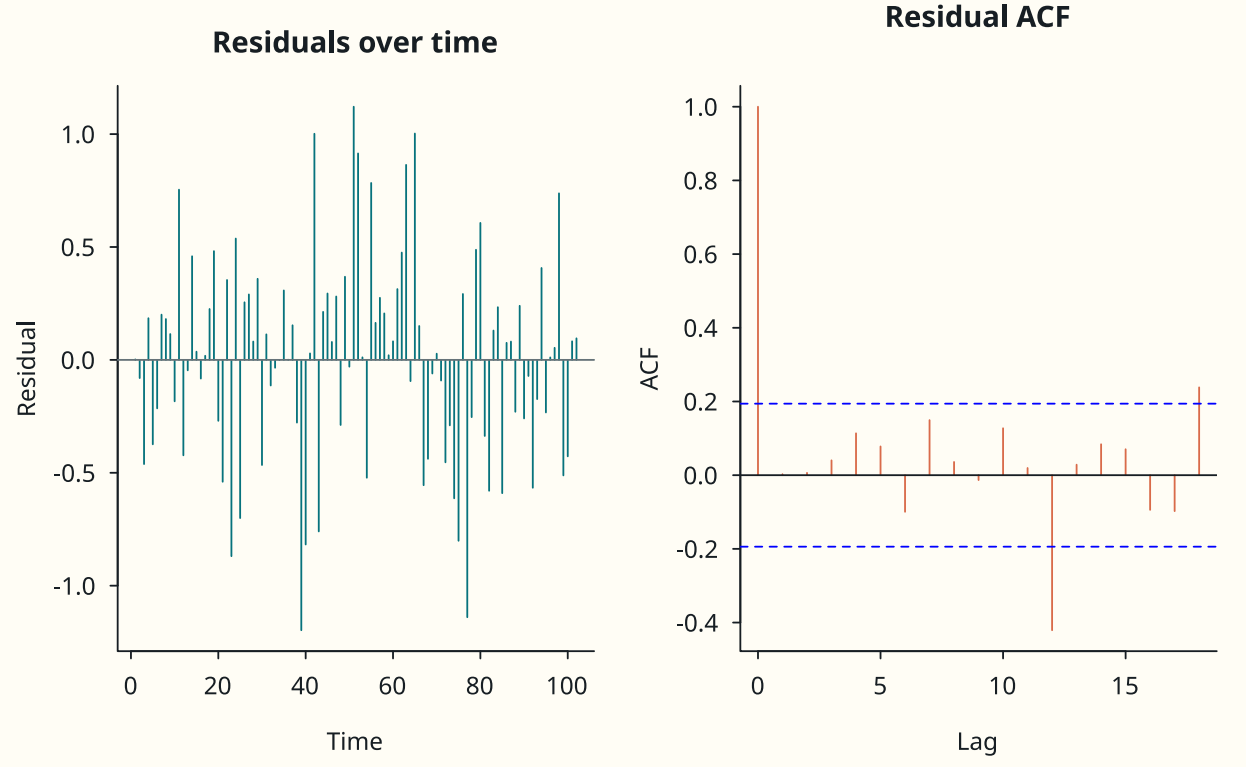

Figure 12.2: ARIMA residuals over time and their sample autocorrelation function.

Interpretation. A failing diagnostic is useful information, not a reason to hide the result. Consider seasonality, alternative differencing, interventions, or a different model, then reevaluate out of sample.

Common mistake

Do not report a 95% prediction interval as a 95% guarantee. It is calibrated only under the fitted model and may omit parameter, revision, or structural-break uncertainty.

Practical tip

Always include a naive benchmark. A complex model that cannot beat a transparent baseline is not earning its complexity.

12.7 Guided practice

Fit ARIMA(0,1,1) and ARIMA(2,1,0) to the same training data. Compare AIC, residual ACF, holdout RMSE, and interval coverage.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

12.8 Exercises

1. Calculate MAE for the chapter forecast and compare its ranking implications with RMSE.
2. Construct 80% forecast intervals using `qnorm(0.90)`.
3. Design a rolling-origin evaluation with a 60-month initial window and six-month horizon.

12.9 Selected solutions

Exercise 1.

```
mean(abs(test - fc$pred))
```

Exercise 2.

```
lower80 <- fc$pred - qnorm(.90) * fc$se
upper80 <- fc$pred + qnorm(.90) * fc$se
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

12.10 Chapter summary

- ARIMA combines differencing with AR and MA dependence.
- Information criteria compare candidates; residuals test what remains.
- Forecast intervals are conditional on model assumptions.
- Time-ordered holdouts and naive benchmarks are essential.

12.11 Chapter cheat sheet

Table 12.1: Chapter 12 command cheat sheet.

Command or pattern	Purpose
`arima()`	fit ARIMA
`AIC()`	penalized fit comparison
`predict()`	forecast and standard errors
`residuals()`	extract innovations
`Box.test()`	residual autocorrelation test
`forecast::Arima()`	extended ARIMA interface

CHAPTER 13

Seasonality and Dynamic Regression

13.1 Motivation

Calendar structure and external predictors can improve forecasts, but ordinary regression standard errors fail when errors remain autocorrelated. Dynamic regression combines interpretable predictors with time-series errors.

13.2 Learning objectives

By the end of this chapter, you should be able to:

- identify and difference seasonal patterns
- use seasonal indicators and Fourier terms
- fit regression with ARIMA errors
- interpret distributed lags and interventions
- forecast only when predictor futures are known or scenario-defined

13.3 Packages and data

Base `stats`; optional `forecast`/`fable`; Canadian CPI and inflation.

The complete tested script is `scripts/ch13_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Dynamic regression separates an interpretable mean relationship from serially dependent errors, avoiding the false precision of independent-error regression.

13.4 Core ideas

Seasonality is systematic variation tied to position within a fixed calendar cycle. A seasonal plot aligns Januarys, Februarys, and so on; a seasonal-subseries plot emphasizes distributions at each position. Seasonal differencing $(1 - B^{12})y_t = y_t - y_{t-12}$ removes a stable annual pattern and can also remove seasonal stochastic trend.

Deterministic seasonality can be represented by eleven monthly indicators plus an intercept, or by Fourier pairs $\sin(2\pi kt/s)$ and $\cos(2\pi kt/s)$. Indicators are flexible; Fourier terms are parsimonious and smooth. Seasonal ARIMA instead models dependence at lags $s, 2s, \ldots$. These approaches express different assumptions and may be combined carefully.

Dynamic regression writes

$$y_t = \beta_0 + \beta^\top x_t + n_t,$$

where n_t follows an ARIMA process. Coefficients explain the conditional mean while the error model supplies dependence-aware inference and forecasts. A distributed lag includes $x_t, x_{t-1}, \ldots$; collinearity and causal timing must be considered.

Intervention variables encode known events: a pulse for one period, step for a sustained level change, or ramp for a gradual change. They are not automatic causal estimators. Forecasting with regression requires future x_t. Calendar terms are known; policy rates or exchange rates require separate forecasts or explicit scenarios. Scenario uncertainty should be communicated.

13.5 Worked example 1: Compare seasonal and first differences

```
cpi <- macro$cpi_all_items_2002_100
monthly_log_change <- 100 * diff(log(cpi))
seasonal_log_change <- 100 * diff(log(cpi), lag = 12)

c(monthly_sd = sd(monthly_log_change),
  seasonal_sd = sd(seasonal_log_change))
```

Result

The monthly log-change SD is **0.399 points**; the twelve-month log-change SD is **1.750 points**. The scales differ because the horizons differ.

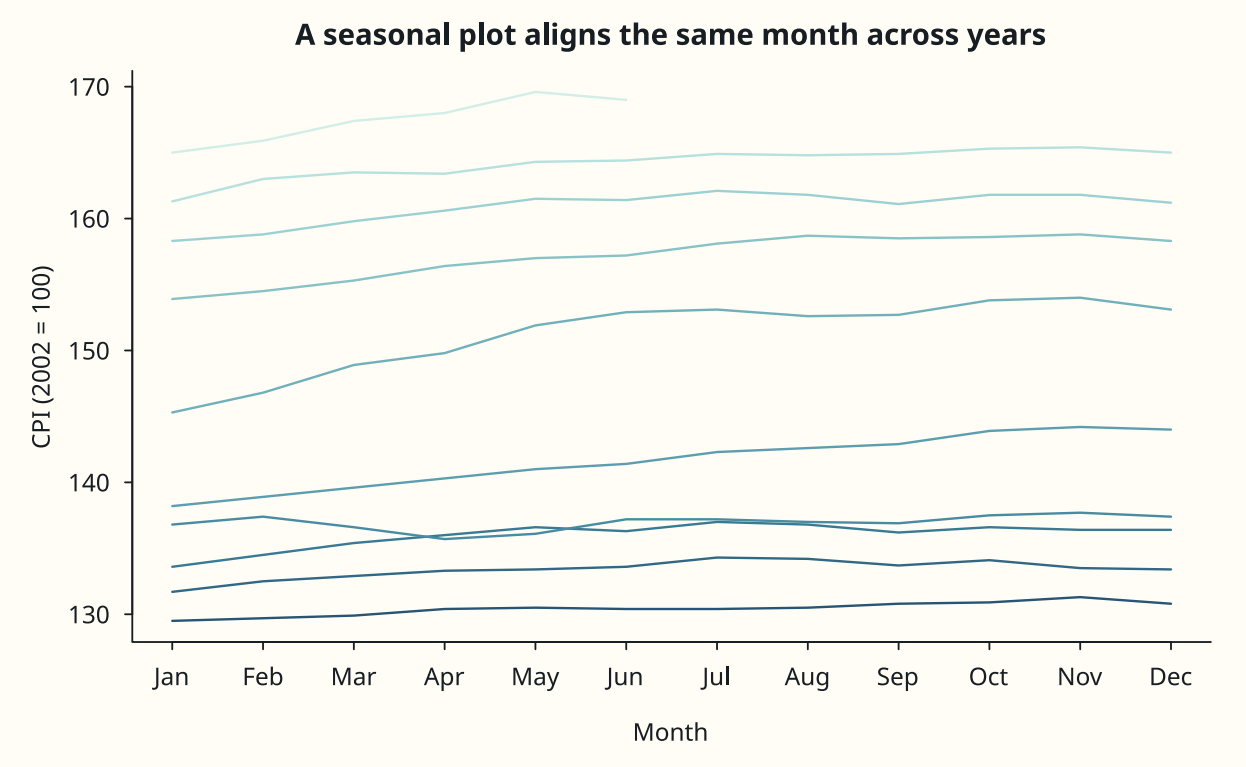

Figure 13.1: Monthly CPI paths aligned by calendar month, one line per year.

Interpretation. Lower variability does not by itself identify the correct transformation. The target quantity, stationarity, seasonality, and forecast horizon determine the choice.

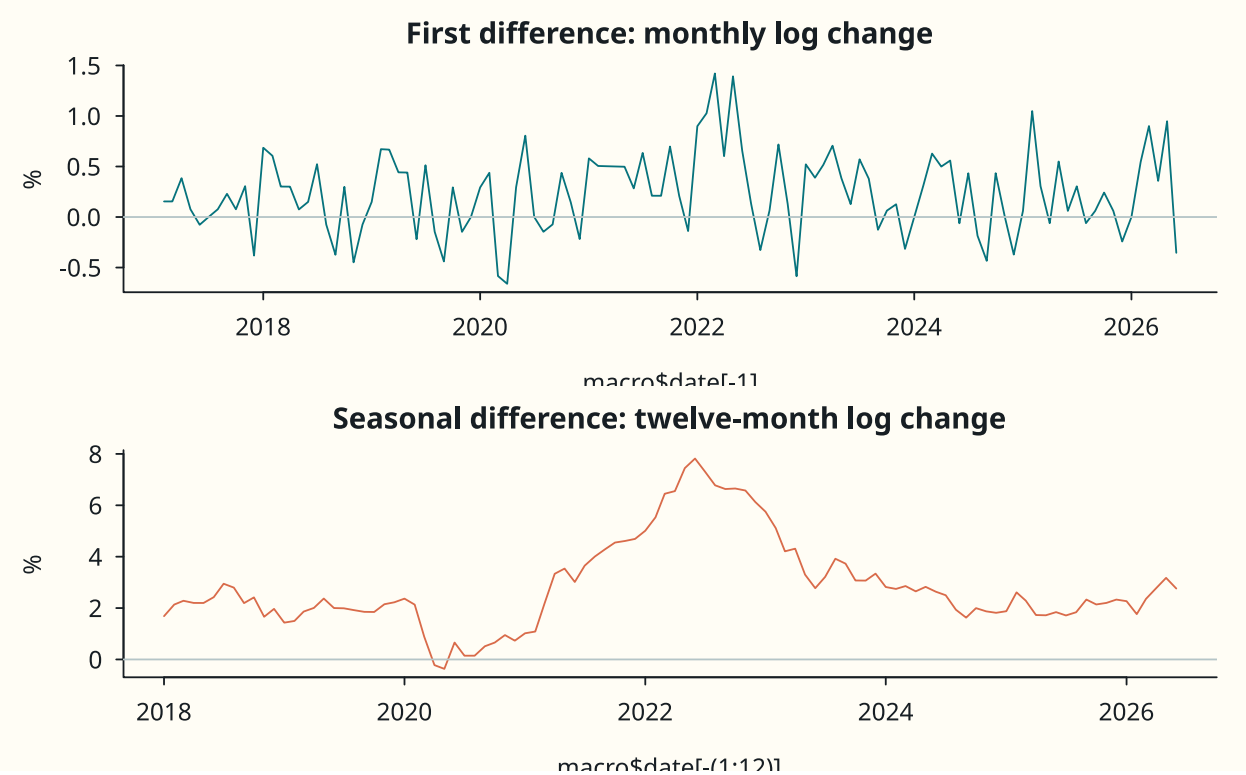

Figure 13.2: Monthly first differences and twelve-month seasonal differences of log CPI.

13.6 Worked example 2: Regress inflation on lagged policy rate

```
analysis <- transform(
  macro,
  policy_lag1 = c(NA, head(policy_rate_percent, -1))
)
ols <- lm(inflation_yoy_percent ~ policy_lag1, data = analysis)

acf(residuals(ols), na.action = na.pass)
Box.test(residuals(ols), lag = 12, type = "Ljung-Box")
```

Result

The residual ACF is the decisive follow-up: if dependence remains, ordinary OLS standard errors and forecast intervals are not trustworthy.

Interpretation. A lag can establish temporal ordering but not exogeneity. Policy may respond to expected inflation, and omitted macroeconomic shocks can affect both variables.

Common mistake

Do not use a predictor's realized future value when evaluating a forecast that would have been made earlier. That is target leakage.

Practical tip

Draw a timeline showing when every predictor becomes available relative to the forecast origin. Release lags matter as much as model lags.

13.7 Guided practice

Create monthly factor indicators and fit CPI log changes on them. Inspect coefficient patterns, then check whether residual ACF remains.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

13.8 Exercises

1. Create a December pulse and a post-2020 step variable.
2. Generate the first two Fourier harmonic pairs for monthly data.
3. Describe two scenarios for future policy rate and how each changes a dynamic-regression forecast.

13.9 Selected solutions

Exercise 1.

```
month <- as.integer(format(macro$date, "%m"))
december <- as.integer(month == 12)
post_2020 <- as.integer(macro$date >= as.Date("2020-03-01"))
```

Exercise 2.

```
t <- seq_len(nrow(macro)); s <- 12
fourier <- cbind(sin1 = sin(2*pi*t/s), cos1 = cos(2*pi*t/s),
          sin2 = sin(4*pi*t/s), cos2 = cos(4*pi*t/s))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

13.10 Chapter summary

- Seasonality may be deterministic, stochastic, or both.
- Seasonal differences and indicators impose different structures.
- Dynamic regression models predictors and serially correlated errors jointly.
- Future predictor availability and scenario uncertainty govern forecasting.

13.11 Chapter cheat sheet

Table 13.1: Chapter 13 command cheat sheet.

Command or pattern	Purpose
`diff(x, lag=12)`	seasonal difference

Command or pattern	Purpose
`factor(month)`	season indicators
`sin()` / `cos()`	Fourier terms
`lag()`	distributed-lag feature
`lm()`	mean regression
`forecast::Arima(xreg=)`	regression with ARIMA errors

Part III

Part III: Financial Time Series and Risk

CHAPTER 14

Financial Data and Return Series

14.1 Motivation

Financial analysis begins with careful data provenance, calendar alignment, and return definitions. A polished price chart is not enough: corporate actions, quote conventions, nontrading days, and provider revisions all affect the series being modelled.

14.2 Learning objectives

By the end of this chapter, you should be able to:

- document a market-data source and retrieval window
- work with dated `xts`/`zoo` objects
- compute simple and log returns
- distinguish prices, adjusted prices, yields, and exchange-rate quotes
- summarize return dependence and tail behaviour

14.3 Packages and data

`xts`, `zoo`, `quantmod` 0.4.29, `tidyquant`, and `PerformanceAnalytics` are current options. The tested example uses Bank of Canada Valet series and base R (Bank of Canada, 2026).

The complete tested script is `scripts/ch14_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Return calculations can be perfectly coded yet economically reversed if a quote direction, adjustment field, or calendar rule is misunderstood.

14.4 Core ideas

Market data require provenance: instrument identifier, venue or provider, field, currency, time zone, adjustment method, retrieval date, and licence. Adjusted equity prices incorporate splits and distributions according to provider rules; unadjusted closes do not. APIs change, so cache a legally shareable teaching extract and keep a retrieval script. Google Finance examples common in older notes are not a reproducible R data source today.

For price P_t, the simple return is $R_t = P_t/P_{t-1} - 1$ and the log return is $r_t = \log(P_t) - \log(P_{t-1}) = \log(1 + R_t)$. Simple returns compound multiplicatively, $\prod(1 + R_t) - 1$; log returns add. They are close for small moves but differ in tails. Returns must be multiplied by 100 only when the unit "percent" is intended.

An exchange-rate quote is directional. `FXUSDCAD` is Canadian dollars per U.S. dollar; an increase means the U.S. dollar buys more Canadian dollars. Its reciprocal answers another question and has opposite return signs. Interest rates quoted in percent should not be logged as if they were asset prices, and negative rates make log transforms undefined.

Calendar alignment is an implicit modelling decision. Merging two markets can use the intersection of trading days, a union with missingness, or explicit previous-tick carry-forward. Forward filling may fabricate contemporaneous information across holidays or time zones. Display missing dates and state the rule before computing cross-market correlations.

14.5 Worked example 1: Align policy rates and USD/CAD by month

```
macro <- read.csv("data/canada_macro_monthly.csv")
macro$date <- as.Date(macro$date)

stopifnot(!anyNA(macro), !anyDuplicated(macro$date))
range(macro$date)
tail(macro[, c("policy_rate_percent",
       "usd_cad_monthly_average")])
```

Result

The frozen monthly table is complete from January 2017 through June 2026. The policy rate is the last business-daily observation in each month; USD/CAD is the mean of available daily averages.

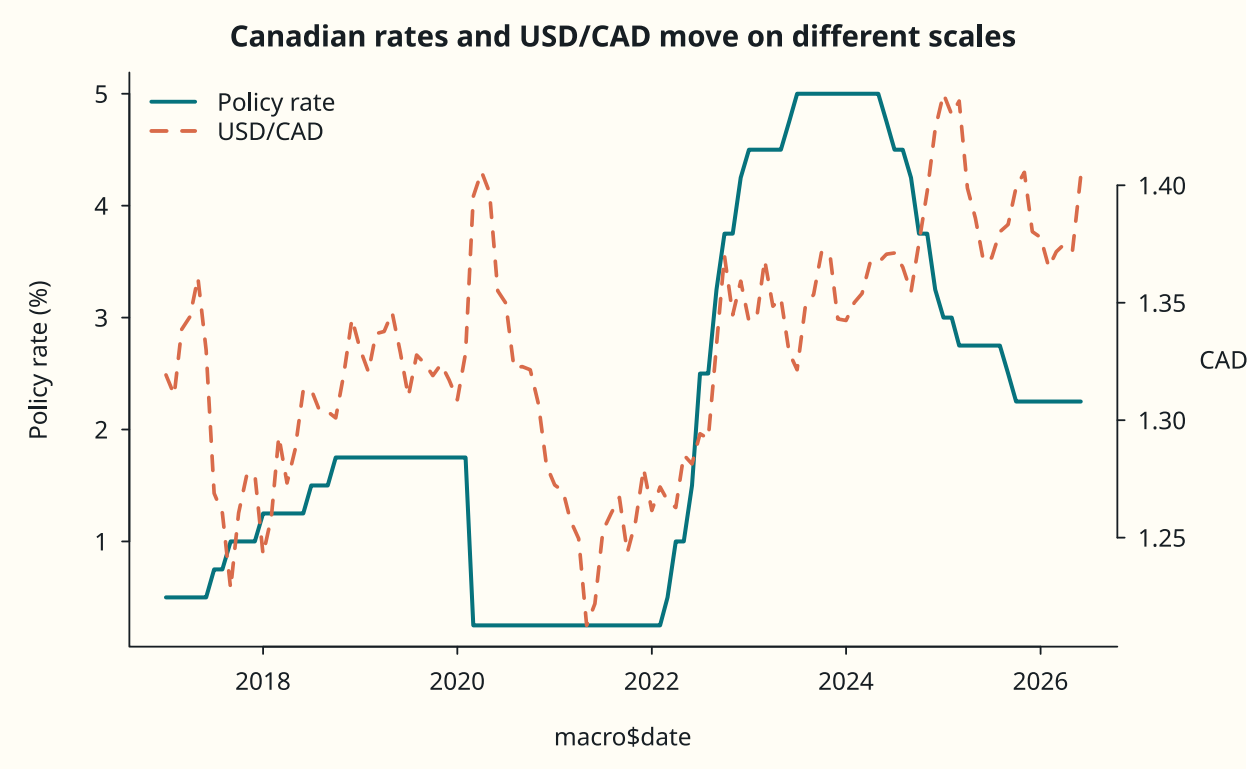

Figure 14.1: Bank of Canada policy rate and USD/CAD monthly average shown with separate, labelled axes.

Interpretation. The two aggregation rules reflect different measurement questions. Neither should be described simply as "the monthly value" without its definition.

14.6 Worked example 2: Compare simple and log FX returns

```
fx <- macro$usd_cad_monthly_average
simple_return <- 100 * (fx[-1] / head(fx, -1) - 1)
log_return <- 100 * diff(log(fx))

c(sd = sd(log_return),
  max_absolute = max(abs(log_return)),
  max_approximation_error = max(abs(simple_return - log_return)))
```

Result

Monthly log-return SD is **1.474%** and the largest absolute monthly move is **4.896%**.

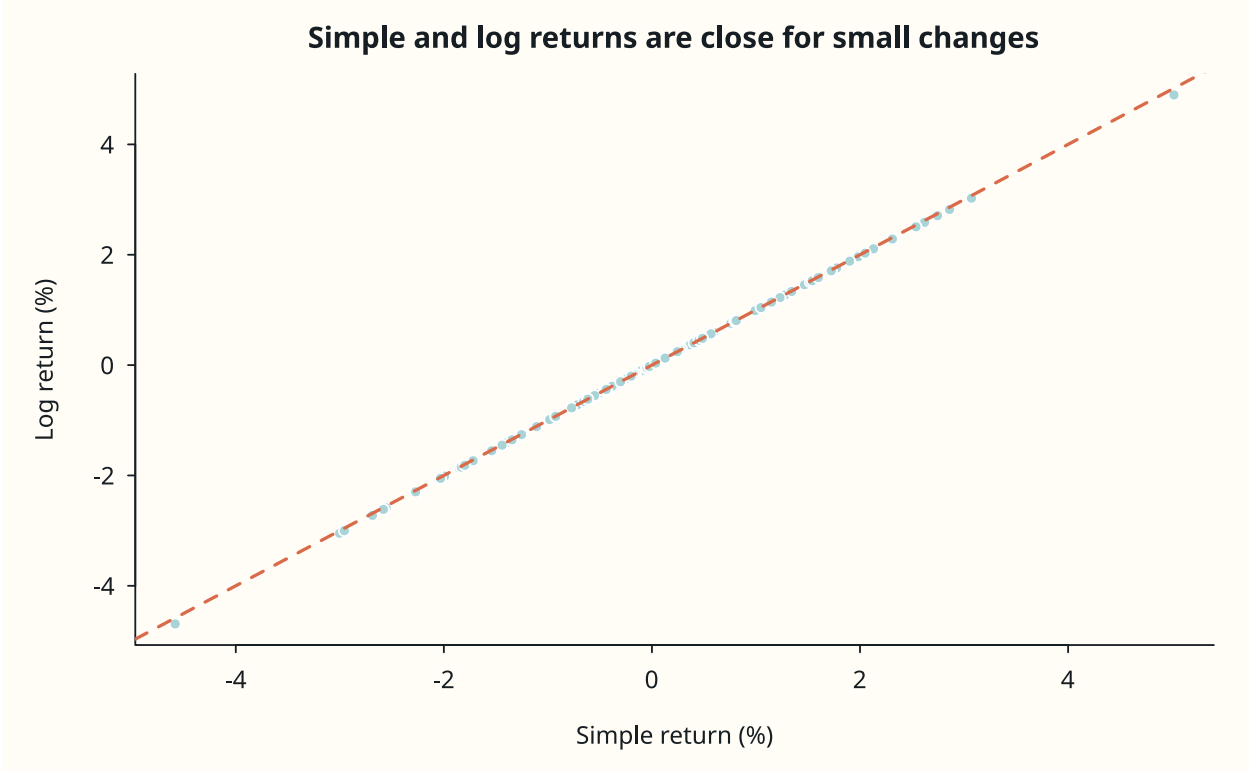

Figure 14.2: Simple versus log monthly USD/CAD returns with a 45-degree reference line.

Interpretation. The diagonal comparison confirms the small-return approximation in this sample. For crisis moves or long horizons, choose the return definition based on the task rather than convenience.

Common mistake

Do not download live data during every book render. A provider outage or revision can change figures and break a supposedly fixed publication.

Practical tip

Print a compact metadata record beside every cached series: source URL, series code, unit, transformation, coverage, and retrieval timestamp.

14.7 Guided practice

Invert USD/CAD to USD per CAD, compute log returns, and verify algebraically and numerically that the returns change sign.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

14.8 Exercises

1. Show that cumulative log return equals the log of the price ratio.
2. Create a date sequence and identify any missing months in the frozen table.
3. Explain why percentage changes are inappropriate for a yield that can cross zero.

14.9 Selected solutions

Exercise 1.

```
sum(diff(log(fx)))
log(tail(fx, 1) / fx[1])
```

Exercise 2.

```
expected <- seq(min(macro$date), max(macro$date), by = "month")
setdiff(expected, macro$date)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

14.10 Chapter summary

- Financial data definitions and provenance are part of the model.
- Simple returns compound; log returns add.
- Quote direction and units determine interpretation.
- Calendar alignment must be explicit and information-safe.

14.11 Chapter cheat sheet

Table 14.1: Chapter 14 command cheat sheet.

Command or pattern	Purpose
`diff(log(x))`	log returns
`cumprod(1+r)`	wealth index
`xts::xts()`	dated market series
`merge()`	calendar alignment
`quantmod::getSymbols()`	provider retrieval
`PerformanceAnalytics::Return.calculate()`	return transformation

CHAPTER 15

ARCH and GARCH Volatility Models

15.1 Motivation

Financial returns often have little linear predictability but strongly predictable variance. ARCH and GARCH models describe volatility clustering and produce time-varying risk forecasts.

15.2 Learning objectives

By the end of this chapter, you should be able to:

- recognize conditional heteroskedasticity
- state ARCH and GARCH recursions
- interpret persistence and unconditional variance
- estimate and diagnose a volatility model
- produce conditional variance forecasts without confusing them with realized volatility

15.3 Packages and data

The examples use tested base-R simulation. `rugarch` 1.5-6 is recommended for production estimation and distributional choices; foundations follow Engle (1982) and Bollerslev (1986).

The complete tested script is `scripts/ch15_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Volatility forecasts influence portfolio sizing, derivative valuation, risk limits, and prediction intervals even when the expected return is essentially zero.

15.4 Core ideas

Write returns as $r_t = \mu_t + \varepsilon_t$ with $\varepsilon_t = \sigma_t z_t$, where z_t has mean zero and variance one. ARCH(q) models $\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2$. Large shocks increase the next conditional variance regardless of sign.

GARCH(1,1) adds its own lagged variance:

$$\sigma_t^2 = \omega + \alpha \varepsilon_{t-1}^2 + \beta \sigma_{t-1}^2.$$

Positive ω, nonnegative α, β, and $\alpha + \beta < 1$ give a finite unconditional variance $\omega/(1 - \alpha - \beta)$ under standard conditions. Persistence $\alpha + \beta$ near one means shocks decay slowly. IGARCH sets the sum to one and removes finite long-run variance in this form.

Maximum-likelihood estimation requires an innovation distribution. Gaussian likelihood is often a quasi-likelihood; Student-t or skewed distributions may fit tails better. Distributional flexibility does not repair a poor variance recursion. Standardized residuals $\hat{z}_t = \hat{\varepsilon}_t / \hat{\sigma}_t$ should have little autocorrelation, and their squares should show little remaining volatility dependence.

A conditional variance forecast is the model's expectation of future squared innovation given current information. It is latent. Realized absolute or squared returns are noisy proxies, and realized measures from high-frequency data use different information. Compare volatility forecasts with proper losses such as QLIKE and robust out-of-sample design.

15.5 Worked example 1: Simulate a persistent GARCH(1,1)

```
garch <- simulate_garch11(
  900,
  omega = 0.000004,
  alpha = 0.08,
  beta = 0.90,
  seed = 1515
)

head(garch)
```

Result

The persistence is **0.98** and implied unconditional volatility is **1.414%** per period.

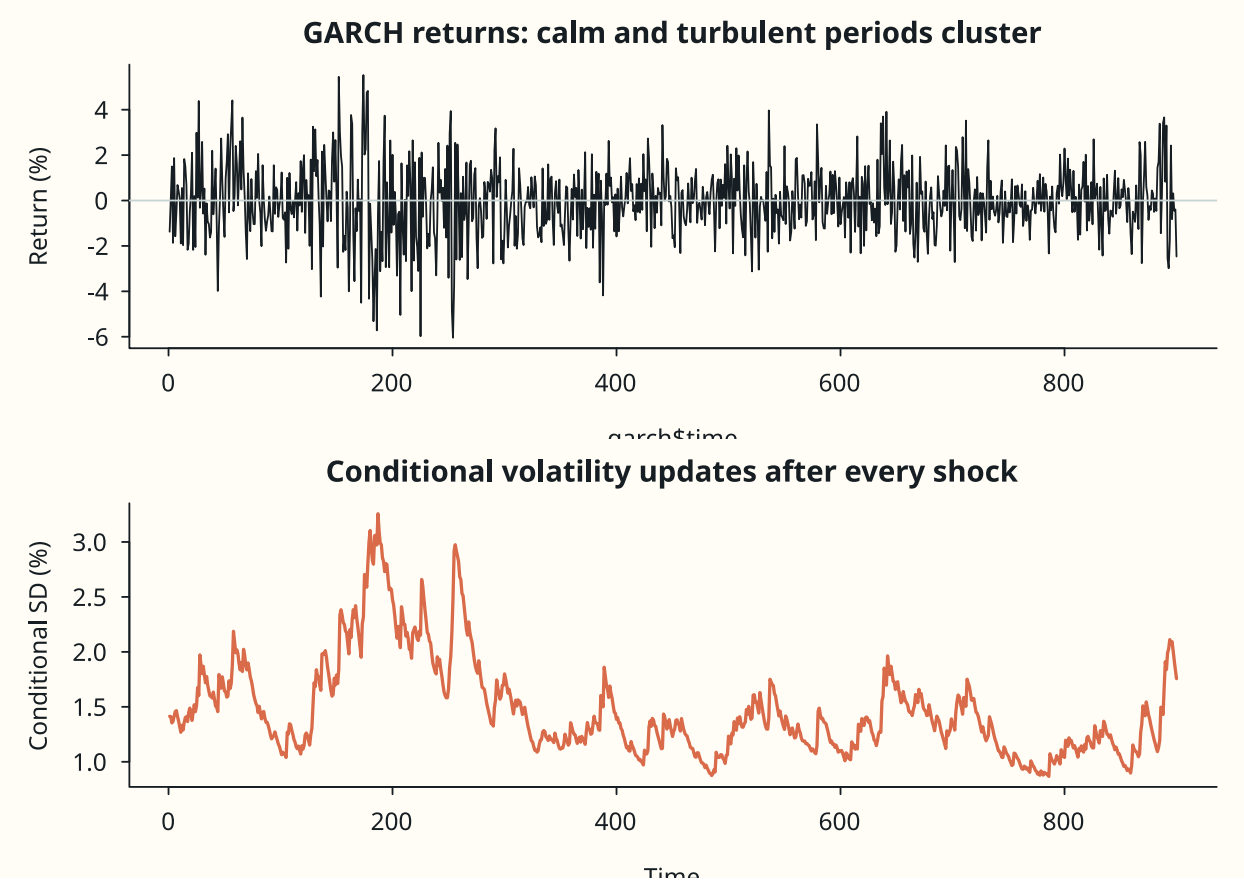

Figure 15.1: Simulated GARCH returns and their latent conditional standard deviation.

Interpretation. The conditional standard deviation changes smoothly while returns remain noisy. Clusters emerge from the variance recursion, not from serial correlation in return signs.

15.6 Worked example 2: Diagnose dependence in squared returns

```r
acf(garch$return, lag.max = 24)
acf(garch$return^2, lag.max = 24)

c(return_acf1 = acf(garch$return, plot = FALSE)$acf[2],
  squared_acf1 = acf(garch$return^2, plot = FALSE)$acf[2])
```

Result

The raw-return ACF is small, while squared returns retain positive dependence across several lags.

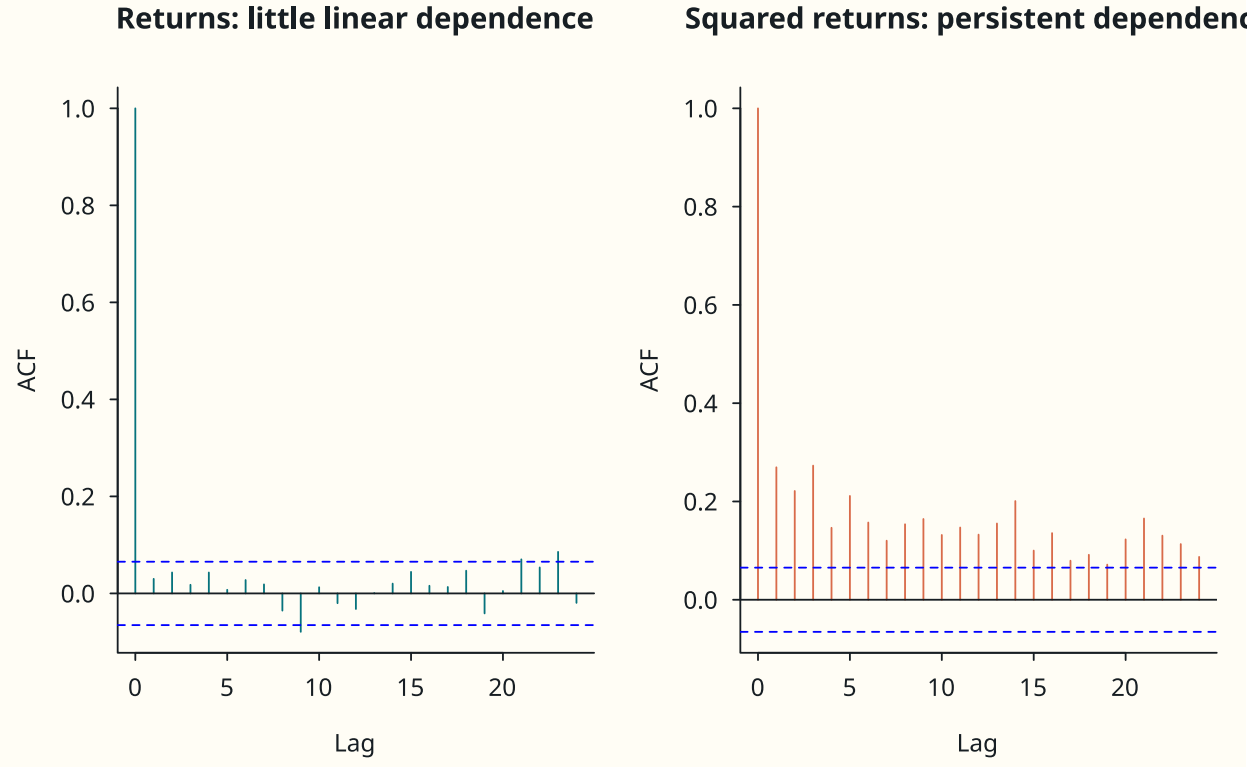

Figure 15.2: ACFs of simulated GARCH returns and squared returns.

Interpretation. A return mean model can look adequate while its intervals and risk estimates remain wrong because conditional variance was ignored.

Common mistake

Do not interpret $\alpha + \beta$ as a probability. It is a persistence measure in the conditional variance recursion.

Practical tip

Inspect standardized residuals, their squares, and a QQ plot. A single Ljung–Box test cannot diagnose the full conditional distribution.

15.7 Guided practice

Simulate the same innovations under persistence 0.70 and 0.98 while holding unconditional variance fixed. Compare volatility response and decay after a large shock.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

15.8 Exercises

1. Compute the half-life $\log(0.5)/\log(\alpha+\beta)$ for the chapter model.
2. Derive the unconditional variance by taking expectations of the GARCH recursion.
3. Explain why fitting GARCH to price levels rather than returns is usually inappropriate.

15.9 Selected solutions

Exercise 1.

```
log(0.5) / log(0.08 + 0.90)
```

Exercise 2.

```
omega <- 0.000004; alpha <- .08; beta <- .90
unconditional_variance <- omega / (1 - alpha - beta)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

15.10 Chapter summary

- ARCH/GARCH model conditional variance, not the return level itself.
- Persistence determines how quickly volatility shocks decay.
- Innovation distribution and variance recursion are separate choices.
- Diagnostics use standardized residuals and their squares.

15.11 Chapter cheat sheet

Table 15.1: Chapter 15 command cheat sheet.

Command or pattern	Purpose
`acf(r^2)`	volatility-dependence diagnostic
`rugarch::ugarchspec()`	model specification
`rugarch::ugarchfit()`	GARCH estimation
`rugarch::ugarchforecast()`	variance forecast
`residuals(..., standardize=TRUE)`	standardized residuals
`sigma()`	conditional volatility

CHAPTER 16

Asymmetric and Extended Volatility Models

16.1 Motivation

Equal-sized positive and negative return shocks often have unequal volatility effects. Asymmetric models encode this leverage-like response and extensions connect expected return, power transforms, and latent volatility dynamics.

16.2 Learning objectives

By the end of this chapter, you should be able to:

- interpret GARCH-M, EGARCH, GJR-GARCH, and APARCH terms
- draw and compare news-impact curves
- state model-specific positivity and stationarity conditions
- contrast observation-driven GARCH with stochastic volatility
- select extensions using diagnostics and forecast performance

16.3 Packages and data

`rugarch` and `stochvol` are modern implementations. Tested base-R simulations illustrate GJR mechanics; key references are Nelson (1991) and Glosten et al. (1993).

The complete tested script is `scripts/ch16_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Missed asymmetry can understate risk following negative shocks and distort option or tail-risk calculations precisely when risk control matters most.

16.4 Core ideas

A news-impact curve holds earlier variance fixed and maps the previous innovation to next variance. Symmetric GARCH depends on ε_{t-1}^2, so equal positive and negative shocks have the same effect. Empirical equity volatility often rises more following negative returns, although the economic mechanism may combine leverage, volatility feedback, and market conditions.

GJR-GARCH writes

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + \gamma I(\varepsilon_{t-1} < 0)\varepsilon_{t-1}^2 + \beta\sigma_{t-1}^2.$$

When $\gamma > 0$, a negative shock adds $\alpha + \gamma$ rather than α. Under symmetric innovations, a common covariance-stationarity condition is $\alpha + \gamma/2 + \beta < 1$.

EGARCH models $\log \sigma_t^2$, so variance is positive without nonnegative coefficient constraints and sign/magnitude effects enter separately. APARCH estimates a power δ and asymmetric absolute-shock term, nesting several specifications. GARCH-in-mean adds volatility or variance to the conditional mean to study a risk–return relationship; its coefficient is not a guaranteed risk premium.

Stochastic-volatility models give latent log variance its own innovation, for example $h_t = \mu + \phi(h_{t-1} - \mu) + \eta_t$ and $r_t = \exp(h_t/2)z_t$. Unlike observation-driven GARCH, volatility is not a deterministic function of past returns. Bayesian or simulation-based estimation is common. More flexible models must justify added parameters through diagnostics, stability, and out-of-sample value.

16.5 Worked example 1: Compare symmetric and asymmetric news impact

```
shock <- seq(-0.05, 0.05, length.out = 301)
base <- 0.000004 + 0.88 * 0.0002
garch_news <- base + 0.06 * shock^2
gjr_news <- base + (0.06 + 0.10 * (shock < 0)) * shock^2

gjr_news[shock == -0.03] / gjr_news[shock == 0.03]
```

Result

At a 3% shock magnitude, the negative-to-positive next-variance ratio is **1.385**.

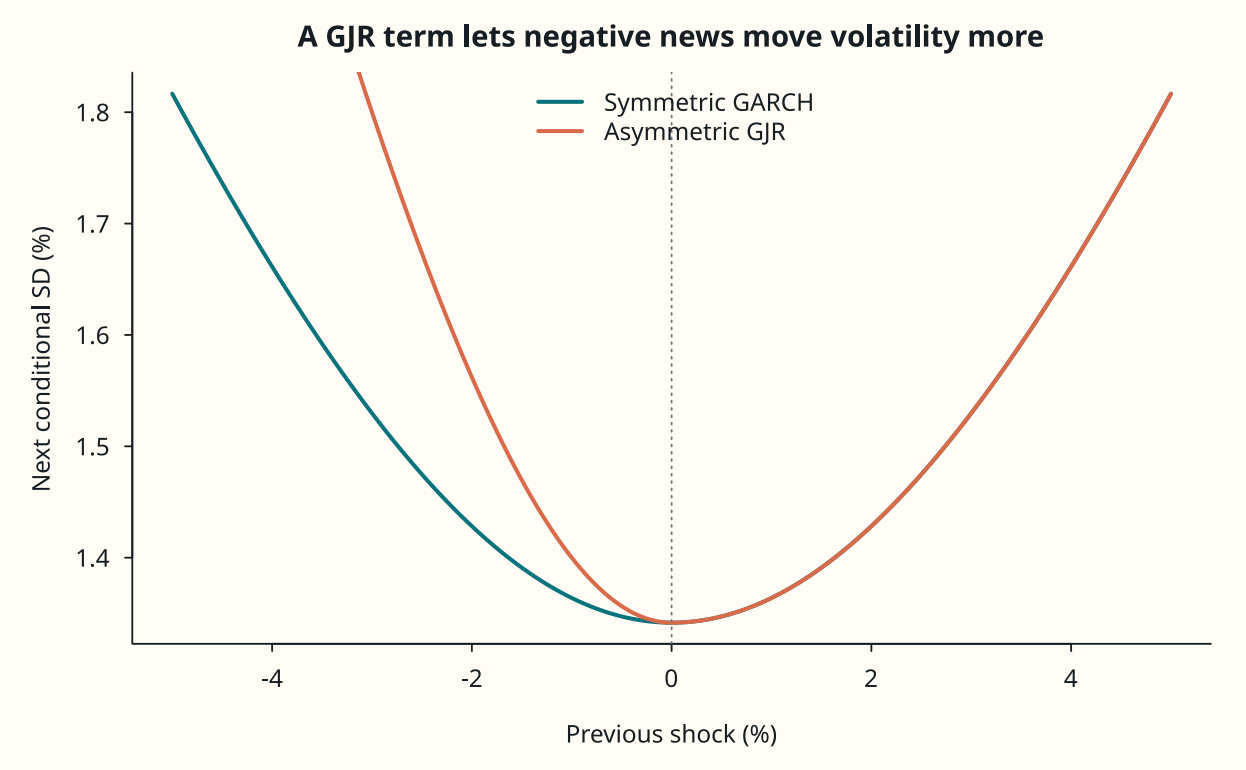

Figure 16.1: Symmetric GARCH and asymmetric GJR news-impact curves across positive and negative shocks.

Interpretation. The ratio is conditional on the chosen previous variance and parameters. The curve visualizes the model specification, not an observed causal law.

16.6 Worked example 2: Use the same innovations in two variance recursions

```
set.seed(1616)
z <- rnorm(450)
```

```
# The complete tested loop is in scripts/ch16_examples.R.
# Both paths use z; only their variance update differs.

stationarity_sum <- 0.06 + 0.10 / 2 + 0.82
stationarity_sum
```

Result

The GJR stationarity sum is **0.93**, below one under the symmetric-innovation condition.

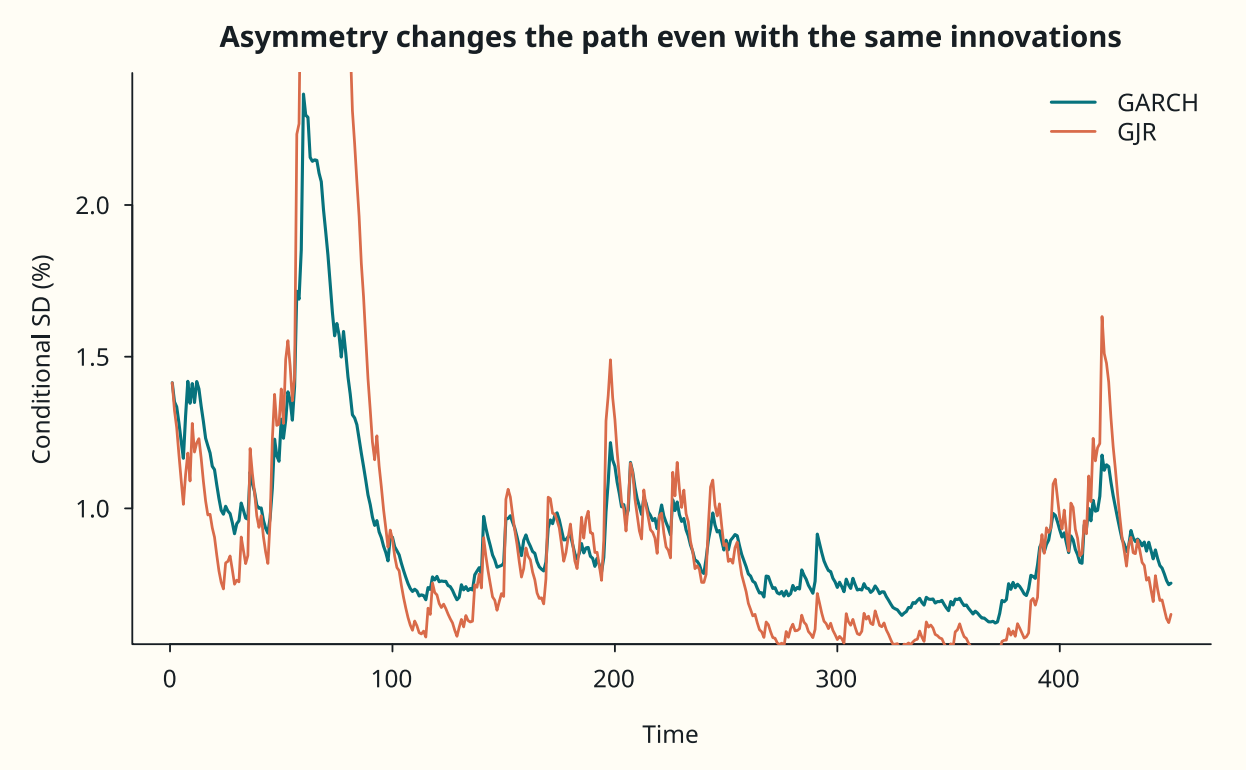

Figure 16.2: Conditional volatility paths from symmetric GARCH and GJR recursions driven by the same standardized innovations.

Interpretation. Holding innovations fixed isolates the effect of the recursion. The asymmetric path can react more strongly after negative returns even when future standardized shocks are identical.

Common mistake

Do not transfer stationarity constraints from one GARCH variant to another. EGARCH, GJR, and APARCH have different parameter meanings and conditions.

Practical tip

Plot the fitted news-impact curve beside standardized residual diagnostics. It makes sign effects much easier to explain than a coefficient table alone.

16.7 Guided practice

Vary γ from 0 to 0.2 while holding other parameters fixed. Plot three news-impact curves and describe the change on each side of zero.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

16.8 Exercises

1. Calculate next variance after +2% and −2% shocks in the chapter GJR model.
2. Explain how log variance guarantees positivity in EGARCH.
3. Give one reason a stochastic-volatility model may fit differently from a GARCH model after an isolated extreme return.

16.9 Selected solutions

Exercise 1.

```
next_var <- function(epsilon) {
  0.000004 + (0.06 + 0.10 * (epsilon < 0)) * epsilon^2 + 0.88 * 0.0002
}
c(positive = next_var(.02), negative = next_var(-.02))
```

Exercise 2.

```
# exp(h_t) is positive for every real h_t,
# so modelling log variance avoids negative fitted variances.
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

16.10 Chapter summary

- News-impact curves expose sign and magnitude effects.
- GJR adds an indicator-weighted response to negative shocks.
- EGARCH, APARCH, and GARCH-M extend different parts of the model.
- Stochastic volatility has its own latent innovation and estimation demands.

16.11 Chapter cheat sheet

Table 16.1: Chapter 16 command cheat sheet.

Command or pattern	Purpose
`rugarch::ugarchspec(model='gjrGARCH')`	GJR specification
`model='eGARCH'`	EGARCH specification
`model='apARCH'`	APARCH specification
`variance.targeting=TRUE`	variance-targeting option
`stochvol::svsample()`	stochastic-volatility sampling
`newsimpact()`	fitted impact curve

CHAPTER 17

Measuring Volatility and Realized Variation

17.1 Motivation

Volatility is latent, so every estimate embeds a window, sampling grid, or model. Comparing rolling, range-based, and high-frequency measures reveals what information each uses and what bias it may introduce.

17.2 Learning objectives

By the end of this chapter, you should be able to:

- calculate rolling historical volatility
- annualize volatility with the correct sampling frequency
- construct realized variance from intraday returns
- explain microstructure-noise and sampling-grid trade-offs
- compare return- and range-based volatility estimators

17.3 Packages and data

Base R for tested examples; optional `zoo`, `xts`, `highfrequency`, and `TTR` for rolling and intraday workflows.

The complete tested script is `scripts/ch17_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Volatility is a measurement problem before it is a forecasting problem. Inconsistent units or sampling conventions can dominate model differences.

17.4 Core ideas

A rolling historical estimate over m returns is $\hat{\sigma}_t = \sqrt{k}\, sd(r_{t-m+1}, \ldots, r_t)$, where k is periods per year. It is transparent but backward-looking, equally weights the window, and changes abruptly when an old observation leaves. Exponentially weighted estimators decay weights smoothly but introduce a decay parameter.

For intraday log returns $r_{t,j}$, realized variance is

$$RV_t = \sum_{j=1}^{M} r_{t,j}^2.$$

Under ideal semimartingale conditions, it estimates integrated variance plus jump variation as sampling becomes finer. Realized volatility is $\sqrt{RV_t}$; confusing variance and volatility produces a unit error.

At very high frequency, bid–ask bounce, price discreteness, asynchronous trading, and recording error create microstructure noise. Sampling ever more finely can therefore increase bias. Signature plots compare realized variance across grids; subsampling, pre-averaging, and realized kernels are designed for noisy data. The "best" grid depends on liquidity and instrument.

Daily high–low ranges contain intraday information. Parkinson and Garman–Klass estimators can be more efficient under idealized diffusion assumptions, but jumps, opening gaps, drift, and market microstructure affect them. Measurement choice should match data availability and the downstream forecast or risk task.

17.5 Worked example 1: Estimate rolling USD/CAD volatility

```
fx_return <- diff(log(macro$usd_cad_monthly_average))
roll12 <- sqrt(12) * 100 * rolling_sd(fx_return, 12)

tail(na.omit(roll12), 1)
```

Result

The final twelve-month rolling USD/CAD volatility estimate is **3.524% annualized**.

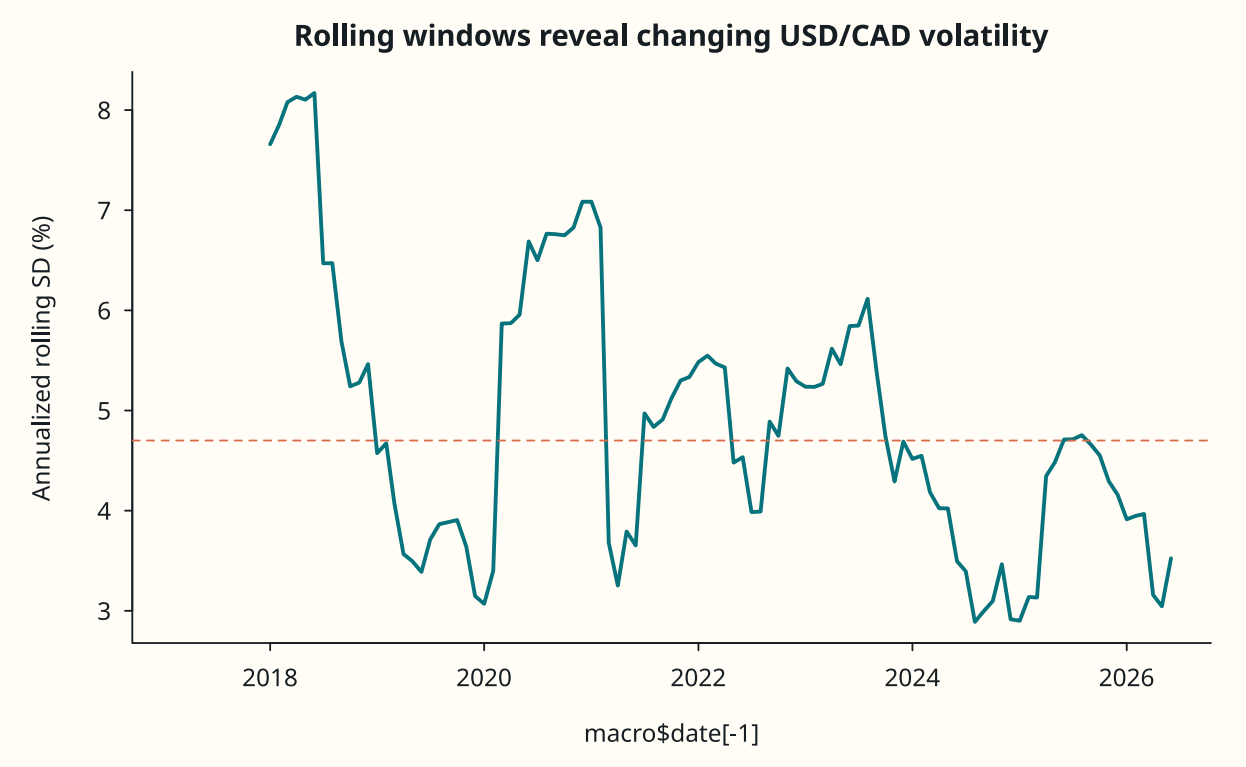

Figure 17.1: Twelve-month rolling annualized volatility of monthly USD/CAD log returns.

Interpretation. The estimate describes the variation in the most recent 12 monthly returns. It is not a forecast unless a separate rule assumes future volatility will equal the window estimate.

17.6 Worked example 2: Change the realized-volatility sampling grid

```
set.seed(1717)
minute_return <- rnorm(390, sd = 0.012 / sqrt(390))
intervals <- c(1, 5, 15, 30, 65)

realized <- sapply(intervals, function(k) {
```

```
  groups <- ceiling(seq_along(minute_return) / k)
  aggregated <- as.numeric(rowsum(minute_return, groups))
  sqrt(sum(aggregated^2))
})
100 * realized
```

Result

The one-minute estimate is **1.129%** against a 1.2% simulation target; coarser grids differ because aggregation changes the finite-sample cross-products.

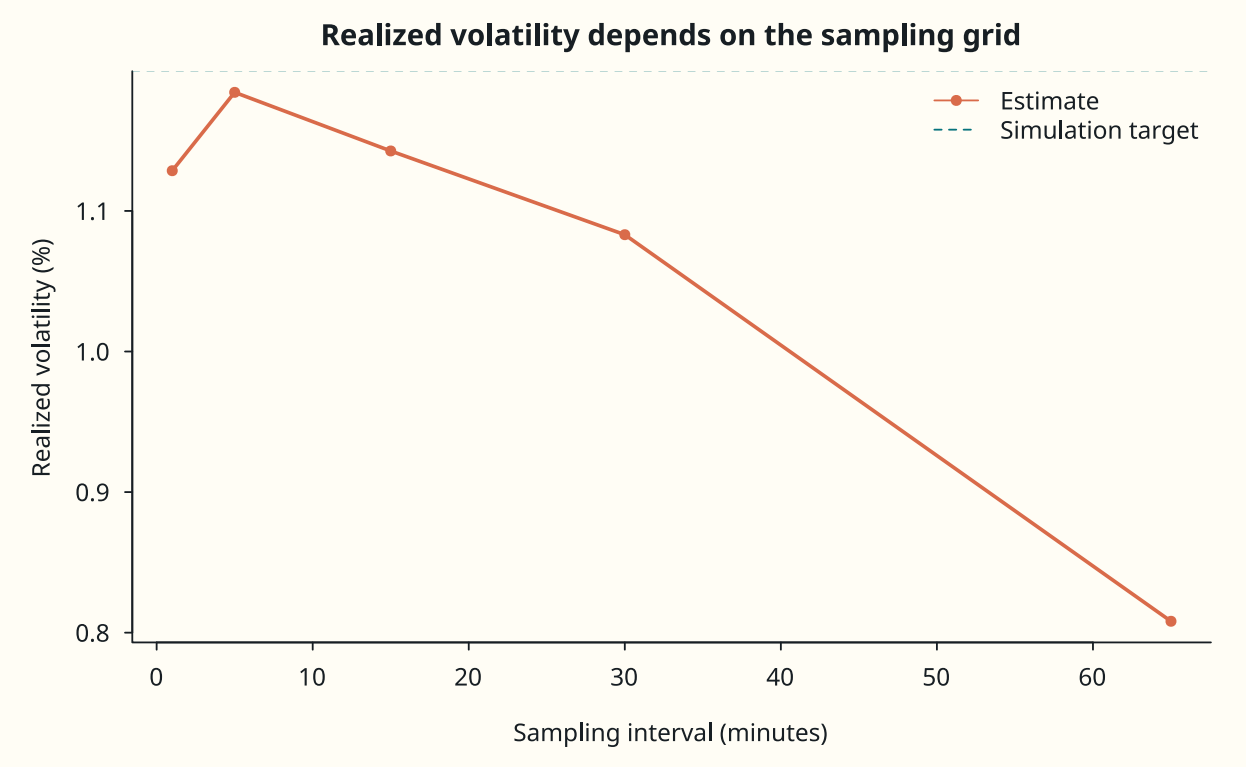

Figure 17.2: Realized volatility estimates across intraday sampling intervals with a simulation target.

Interpretation. The simulation has no microstructure noise, so it illustrates sampling variation only. Real tick data add the bias–variance trade-off described above.

Common mistake

Do not annualize a standard deviation by multiplying by the number of periods. Under the usual independent-increment scaling, multiply by the square root.

Practical tip

Put the unit in the object name or table header: `daily_variance`, `monthly_sd`, and `annualized_vol_percent` are harder to confuse.

17.7 Guided practice

Recompute rolling FX volatility with 6-, 12-, and 24-month windows. Describe responsiveness, smoothness, and the number of unavailable early estimates.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

17.8 Exercises

1. Show algebraically that realized variance is in squared-return units.
2. Add alternating bid–ask noise to minute prices and draw a signature plot.
3. Compute Parkinson variance from a small table of daily high and low prices and state its assumptions.

17.9 Selected solutions

Exercise 1.

```
windows <- c(6, 12, 24)
estimates <- lapply(windows, function(w) sqrt(12) * rolling_sd(fx_return,
 ↪  w))
```

Exercise 2.

```
parkinson_variance <- function(high, low) {
  mean(log(high / low)^2) / (4 * log(2))
}
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

17.10 Chapter summary

- Rolling volatility depends on window length and annualization convention.
- Realized variance sums squared intraday returns.
- Finer sampling faces microstructure-noise bias.
- Range estimators use more within-period information under additional assumptions.

17.11 Chapter cheat sheet

Table 17.1: Chapter 17 command cheat sheet.

Command or pattern	Purpose
`zoo::rollapply()`	rolling window
`sqrt(k) * sd(r)`	annualized volatility
`sum(r^2)`	realized variance
`rowsum()`	aggregate intraday returns
`highfrequency::rCov()`	realized covariance
`TTR::runSD()`	rolling SD

CHAPTER 18

Nonlinear Dynamics and Discrete Outcomes

18.1 Motivation

Economic behaviour can change by regime, react nonlinearly, or arrive as durations and ordered categories rather than continuous returns. Threshold, neural, duration, and ordered-response models extend the linear toolkit without abandoning diagnostics.

18.2 Learning objectives

By the end of this chapter, you should be able to:

- fit and interpret threshold autoregressions
- distinguish smooth and abrupt regime changes
- describe a feed-forward neural time-series model
- explain market microstructure and ACD duration models
- calculate ordered-probit category probabilities

18.3 Packages and data

Base R simulations; optional `tsDyn`, `nnet`, `ordinal`, and duration-model tools. The chapter modernizes threshold, neural-network, market-duration, and ordered-probit material from the legacy notes.

The complete tested script is `scripts/ch18_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Nonlinear models should represent a plausible data-generating mechanism or a demonstrated forecast gain—not merely improve in-sample fit.

18.4 Core ideas

A two-regime threshold autoregression (TAR) can be written

$$y_t = (a_1 + \phi_1 y_{t-1}) I(q_{t-d} \leq c) + (a_2 + \phi_2 y_{t-1}) I(q_{t-d} > c) + \varepsilon_t.$$

The threshold variable q, delay d, and threshold c define the regimes. TAR transitions are abrupt; smooth-transition autoregressions replace the indicator with a logistic or exponential weight. Regime-specific stationarity and occupancy both matter.

A feed-forward autoregressive neural network uses lagged values as inputs, hidden nonlinear activations, and a linear or task-specific output. It can approximate nonlinear conditional means, but weights are hard to interpret and regularization matters. Time-ordered validation, scaling fitted on training data, multiple initializations, and comparison with simple benchmarks are essential.

Market microstructure studies transaction-level price formation, bid–ask spread, order flow, and irregular event times. Duration $x_i = t_i - t_{i-1}$ between trades is positive and clustered. An ACD model writes $x_i = \psi_i \epsilon_i$ with a positive conditional mean such as $\psi_i = \omega + \alpha x_{i-1} + \beta \psi_{i-1}$. Timestamp resolution, overnight gaps, and market-session boundaries must be handled before estimation.

Ordered outcomes—low, middle, high—have rank but unknown spacing. Ordered probit assumes latent $y_i^* = x_i^\top \beta + \epsilon_i$, $\epsilon_i \sim N(0,1)$, and cutpoints κ. Category probabilities are differences of normal CDFs. Coefficient signs shift probability mass, but marginal effects vary with x and category; calculate probabilities rather than interpreting coefficients like OLS slopes.

18.5 Worked example 1: Simulate a threshold autoregression

```
set.seed(1818)
y <- numeric(500)
for (i in 2:length(y)) {
  slope <- if (y[i - 1] <= 0) 0.25 else 0.82
  y[i] <- slope * y[i - 1] + rnorm(1, sd = 0.7)
}

mean(head(y, -1) > 0)
```

Result

The upper regime contains **67.3%** of lagged observations in this realization.

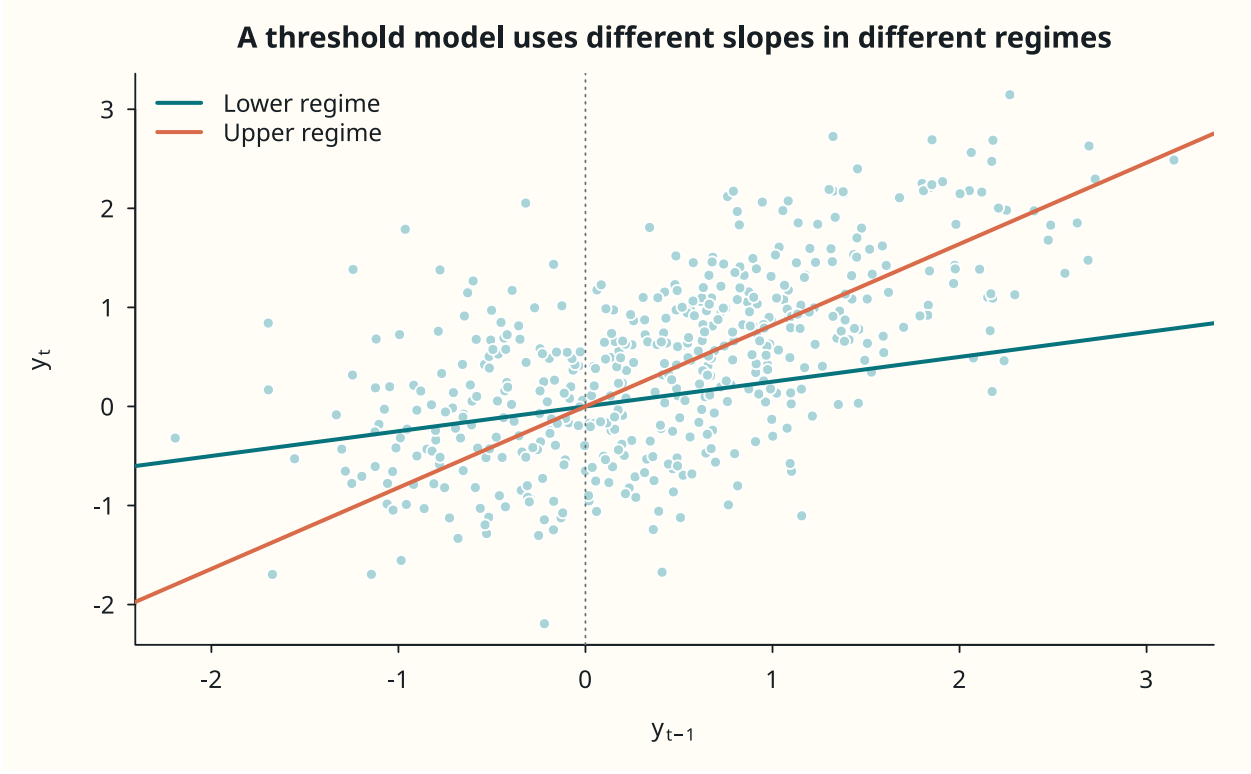

Figure 18.1: Lag plot of a two-regime TAR simulation with different fitted slopes below and above zero.

Interpretation. Unequal regime counts affect estimation precision. A threshold near the data boundary may create a flexible-looking model supported by too few observations.

18.6 Worked example 2: Turn a latent ordered index into probabilities

```
x <- seq(-2.5, 2.5, length.out = 201)
probability <- ordered_probabilities(
  x,
  beta = 1.2,
  cutpoints = c(-0.5, 0.7)
)

max(abs(rowSums(probability) - 1))
```

Result

Maximum row-sum error is approximately 1.1×10^{-16}, numerical roundoff from one.

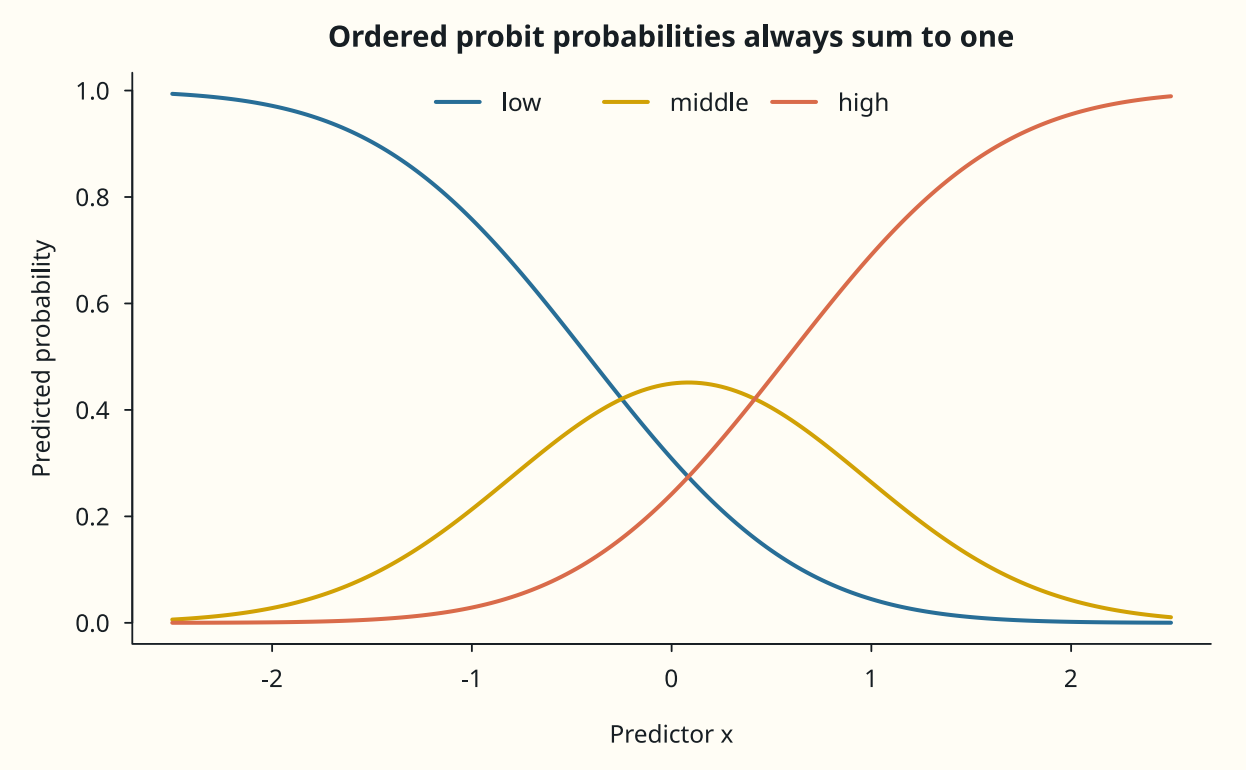

Figure 18.2: Low, middle, and high ordered-probit probabilities across a continuous predictor.

Interpretation. As the predictor rises, probability moves from low toward high, but the middle category can rise and then fall. This is why one coefficient cannot be read as a constant probability change.

Common mistake

Do not randomly split lagged or transaction data. Neighbouring observations can leak nearly identical information across training and test sets.

Practical tip

Plot the fitted response surface, regime rule, or category probabilities. A nonlinear model becomes reviewable when its implications are visible.

18.7 Guided practice

Change the TAR threshold from 0 to the sample median and compare regime counts, fitted slopes, and holdout error.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

18.8 Exercises

1. Simulate a smooth-transition model using a logistic weight.
2. Write an ACD(1,1) simulator with positive exponential innovations.
3. Calculate ordered-probit probabilities at $x = -1, 0, 1$ and explain how each category changes.

18.9 Selected solutions

Exercise 1.

```
logistic_weight <- function(q, gamma = 3, c = 0) {
  1 / (1 + exp(-gamma * (q - c)))
}
```

Exercise 2.

```
ordered_probabilities(c(-1, 0, 1), beta = 1.2,
                      cutpoints = c(-0.5, 0.7))
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

18.10 Chapter summary

- TAR models switch abruptly between regime-specific dynamics.
- Neural autoregressions need regularization and time-aware validation.
- ACD models positive, irregularly spaced event durations.
- Ordered probit coefficients are best communicated through category probabilities.

18.11 Chapter cheat sheet

Table 18.1: Chapter 18 command cheat sheet.

Command or pattern	Purpose
`ifelse()`	threshold rule
`tsDyn::setar()`	threshold autoregression
`nnet::nnet()`	neural network
`diff(timestamp)`	event durations
`MASS::polr()`	ordered logit/probit
`pnorm()`	ordered-probit probabilities

CHAPTER 19

Continuous-Time Models and Option Pricing

19.1 Motivation

Continuous-time models connect random paths, risk-neutral valuation, and derivative payoffs. Simulation in R makes the mathematics tangible while exposing discretization and model risk.

19.2 Learning objectives

By the end of this chapter, you should be able to:

- simulate Brownian motion and geometric Brownian motion
- apply the intuition of Itô's lemma
- distinguish physical and risk-neutral measures
- price a European call with Black–Scholes
- interpret payoff, price, delta, and volatility sensitivity

19.3 Packages and data

Base `stats`; optional `RQuantLib` for conventions and broader instruments. Foundations follow Black & Scholes (1973) and Merton (1973).

The complete tested script is `scripts/ch19_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Pricing formulas are conditional maps from inputs and assumptions to values. Sensitivity analysis is therefore as important as the point estimate.

19.4 Core ideas

Standard Brownian motion W_t starts at zero, has independent Gaussian increments $W_{t+h} - W_t \sim N(0, h)$, and continuous nowhere-differentiable paths. On a grid of size Δt, simulation accumulates $\sqrt{\Delta t} Z_j$. Refining the grid changes the path representation; it does not reveal a hidden smooth derivative.

Geometric Brownian motion satisfies $dS_t = \mu S_t dt + \sigma S_t dW_t$ and has solution

$$S_t = S_0 \exp[(\mu - \tfrac{1}{2}\sigma^2)t + \sigma W_t].$$

The $-\sigma^2/2$ term is the Itô correction. Itô's lemma is the stochastic counterpart to a chain rule and generates the second-derivative term because $(dW_t)^2$ contributes at order dt.

For a non-dividend-paying asset under Black–Scholes assumptions, risk-neutral valuation replaces the physical drift with the risk-free rate. A European call is

$$C = S_0 \Phi(d_1) - K e^{-rT} \Phi(d_2),$$

where $d_1 = [\log(S_0/K) + (r + \sigma^2/2)T]/(\sigma\sqrt{T})$ and $d_2 = d_1 - \sigma\sqrt{T}$. The value depends on tradable replication assumptions, not an estimate of the asset's expected return.

The terminal call payoff is $\max(S_T - K, 0)$; price is today's discounted expectation under a pricing measure, so payoff and price are not interchangeable. Delta $\Phi(d_1)$ is local price sensitivity to spot. Vega measures volatility sensitivity. Real markets violate constant volatility, continuous trading, and frictionless assumptions; implied-volatility smiles and hedging error are evidence of model risk.

19.5 Worked example 1: Build Brownian and GBM paths from the same shocks

```
set.seed(1919)
steps <- 252
dt <- 1 / steps
time <- seq(0, 1, length.out = steps + 1)
brownian <- c(0, cumsum(rnorm(steps, sd = sqrt(dt))))

gbm <- 100 * exp((0.06 - 0.22^2 / 2) * time +
         0.22 * brownian)
```

Result

Brownian motion can cross zero; GBM remains positive because it exponentiates the log-price process.

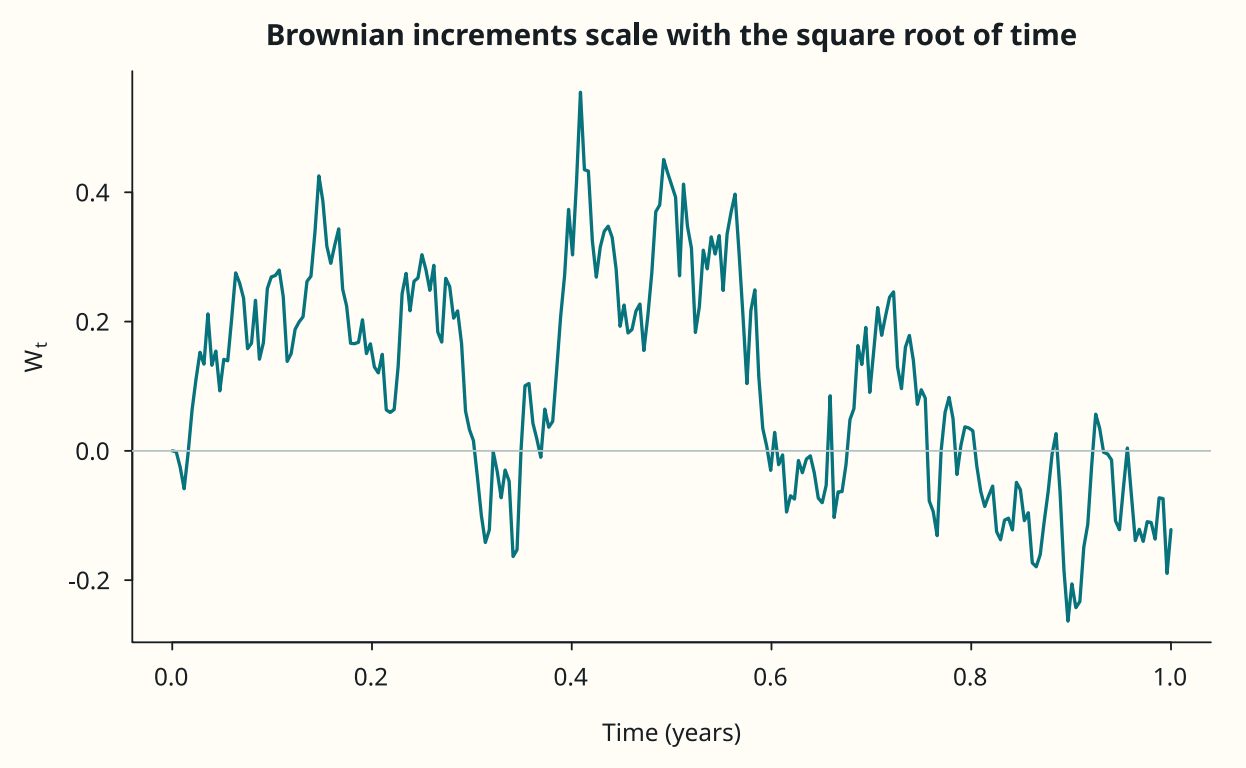

Figure 19.1: One simulated Brownian path over a year with 252 increments.

Interpretation. The same Brownian path drives both displays. A single simulated trajectory illustrates mechanics, not the full distribution of possible outcomes.

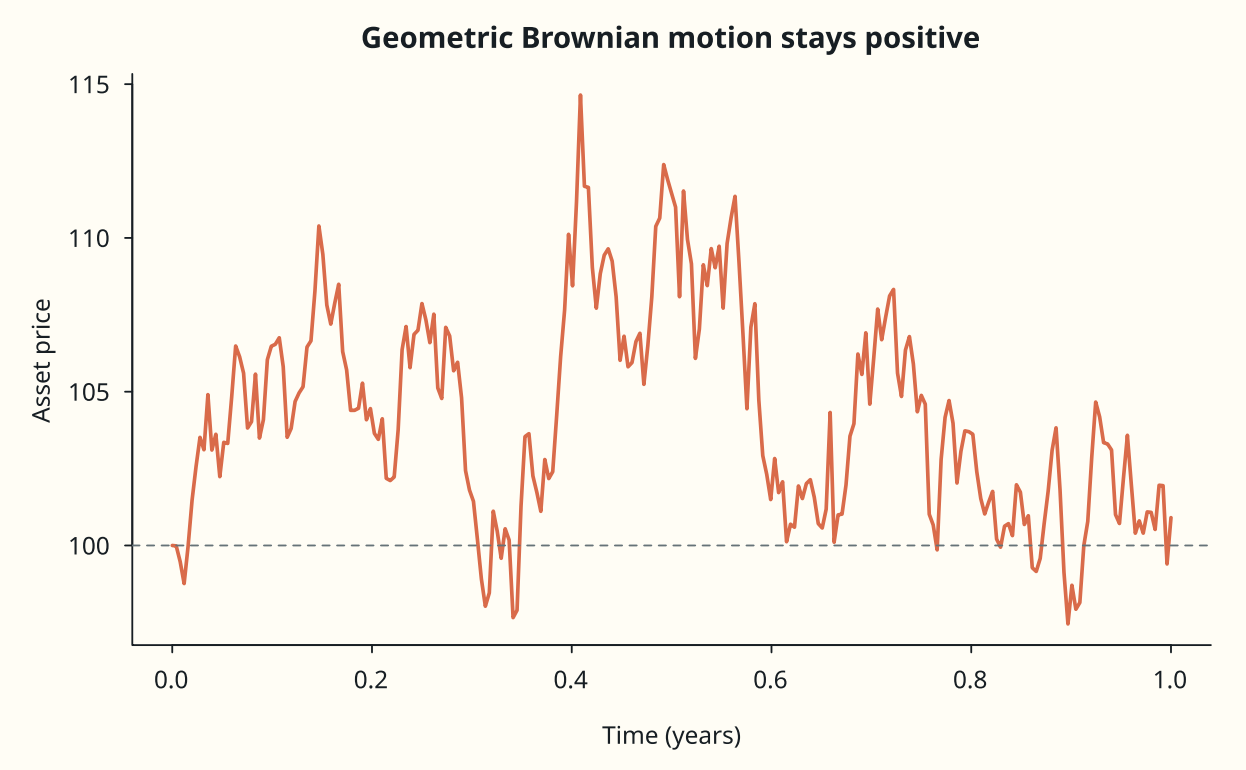

Figure 19.2: A geometric Brownian asset-price path driven by the same Brownian motion.

19.6 Worked example 2: Price and interpret an at-the-money call

```
price <- black_scholes_call(
  spot = 100, strike = 100,
  rate = 0.03, volatility = 0.20, maturity = 1
)
d1 <- (log(100 / 100) + (0.03 + 0.20^2 / 2)) / 0.20
delta <- pnorm(d1)
c(price = price, delta = delta)
```

Result

The call price is **$9.413** and delta is **0.5987** per spot dollar under the stated assumptions.

Interpretation. Delta is a local derivative, not the exercise probability. The risk-neutral probability $\Phi(d_2)$ is different, and neither is a physical forecast without further assumptions.

Common mistake

Do not mix annualized volatility, maturity units, and interest-rate units. A 20% volatility input is `0.20`, not `20`.

Practical tip

Test limiting cases: call value should rise with spot and volatility, fall with strike, stay nonnegative, and approach payoff near expiry.

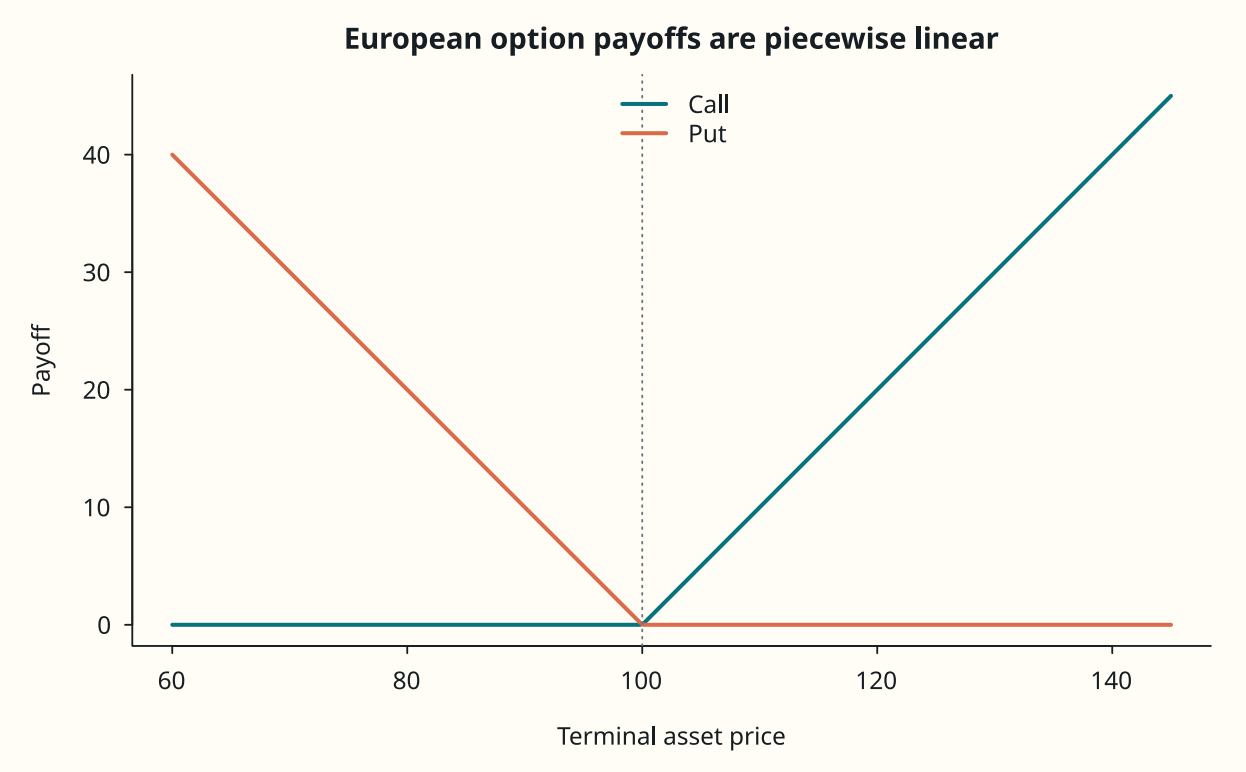

Figure 19.3: European call and put terminal payoffs around a strike of 100.

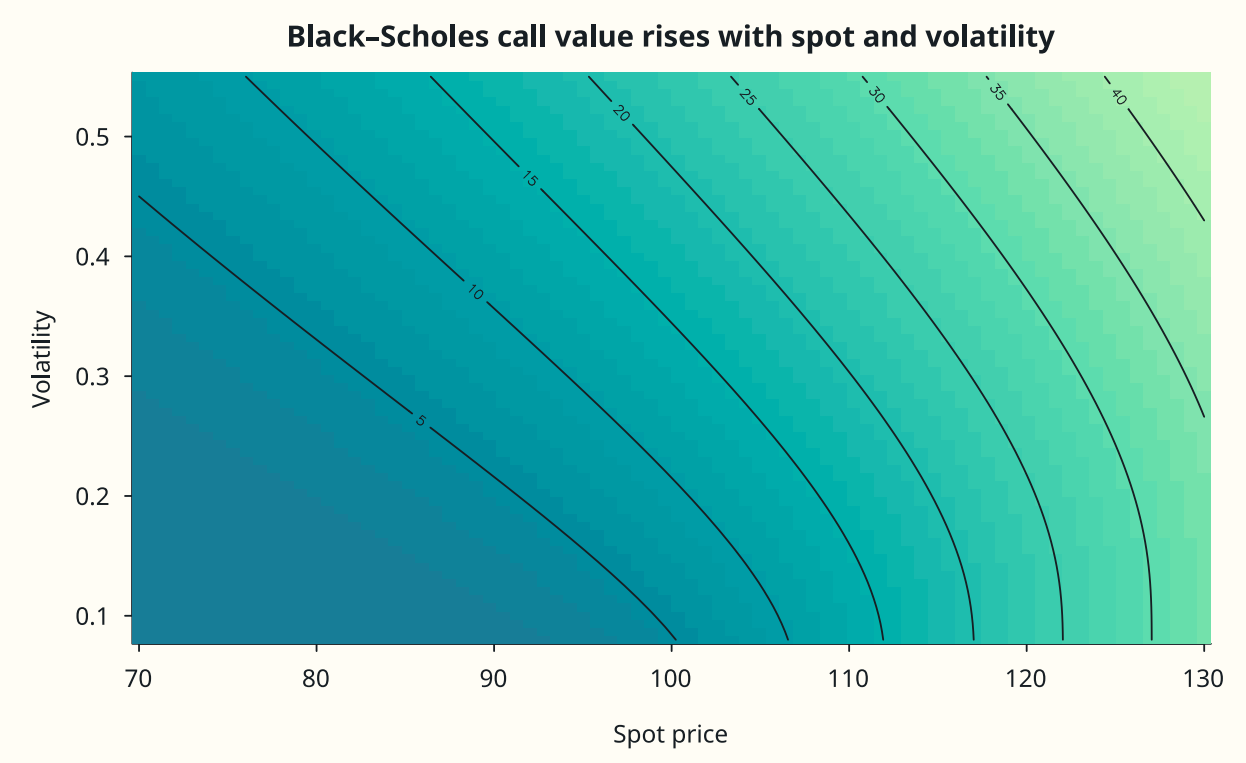

Figure 19.4: Black–Scholes call values across spot and volatility, with contour lines.

19.7 Guided practice

Calculate call prices for volatilities 10%, 20%, and 40% and maturities one month, six months, and one year. Explain the interaction without calling every change linear.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

19.8 Exercises

1. Verify put–call parity numerically.
2. Simulate 20,000 risk-neutral terminal prices and estimate the discounted call payoff; compare with Black–Scholes.
3. Approximate delta with a centred finite difference and compare with $\Phi(d_1)$.

19.9 Selected solutions

Exercise 1.

```
call <- black_scholes_call(100, 100, .03, .2, 1)
put <- call - 100 + 100 * exp(-.03)
call - put - (100 - 100 * exp(-.03))
```

Exercise 2.

```
h <- .01
fd_delta <- (black_scholes_call(100+h, 100, .03, .2, 1) -
     black_scholes_call(100-h, 100, .03, .2, 1)) / (2*h)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

19.10 Chapter summary

- Brownian increments scale with $\sqrt{dt}$.
- GBM remains positive and includes an Itô correction in log space.
- Black–Scholes prices under a risk-neutral measure and strong assumptions.
- Payoff, price, probability, and Greek sensitivity are distinct quantities.

19.11 Chapter cheat sheet

Table 19.1: Chapter 19 command cheat sheet.

Command or pattern	Purpose
`cumsum(rnorm(...))`	Brownian simulation
`exp()`	GBM transformation
`pnorm()`	normal CDF
`pmax()`	option payoff
`uniroot()`	implied-volatility inversion
`RQuantLib::EuropeanOption()`	library option valuation

CHAPTER 20

Market Risk: VaR, Expected Shortfall, and EVT

20.1 Motivation

Risk measures compress a loss distribution into decision quantities. Their apparent simplicity hides choices about horizon, confidence level, distribution, dependence, liquidity, and backtesting—choices that must be explicit.

20.2 Learning objectives

By the end of this chapter, you should be able to:

- define loss, VaR, and expected shortfall
- estimate risk by historical and parametric methods
- backtest exceedances without tuning on the test period
- use peaks-over-threshold EVT and diagnose thresholds
- communicate model risk and the limits of a one-number summary

20.3 Packages and data

Base R tested examples; optional `PerformanceAnalytics`, `rugarch`, and `evir`. Risk concepts follow Artzner et al. (1999) and current institutional guidance should be checked before regulatory use.

The complete tested script is `scripts/ch20_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

A risk number is actionable only when its loss definition, horizon, probability, model, and validation status are attached.

20.4 Core ideas

Let L be a loss, positive when the portfolio loses money. Value at Risk at probability p is the loss quantile $VaR_p = \inf\{l : P(L \leq l) \geq p\}$. It says little about severity beyond the threshold. Expected shortfall is the mean tail loss; for a continuous distribution, $ES_p = E[L \mid L \geq VaR_p]$. Always state horizon, probability, currency or return unit, position date, and whether losses are simulated or historical.

Historical simulation uses the empirical distribution and preserves observed nonlinearities, but assumes the selected past is representative and is limited in extreme-tail resolution. Variance–covariance methods are fast but depend on distribution and linearity assumptions. Filtered historical simulation first standardizes by a volatility model, resamples innovations, and applies a current volatility forecast. Monte Carlo can reprice nonlinear positions but adds model and computational choices.

Backtesting compares realized losses with forecasts made using only prior information. A correctly calibrated 95% VaR has about 5% exceptions over many independent forecast periods, but clustering also matters. Kupiec tests

unconditional coverage; Christoffersen-style tests add independence. Repeatedly changing a model in response to the same backtest contaminates the evaluation.

Extreme-value peaks-over-threshold modelling fits a generalized Pareto distribution to excesses $X - u$ above high threshold u. The threshold balances bias (too low) and variance (too high). Mean-excess and parameter-stability plots assist but do not automate the choice. Temporal clustering may require declustering or extremal-index methods. EVT extrapolation is especially uncertain, so sensitivity bands and scenario stress tests should accompany point estimates.

20.5 Worked example 1: Estimate historical VaR and expected shortfall

```
loss <- -100 * simulate_garch11(
  2600, omega = 0.000003,
  alpha = 0.07, beta = 0.91, seed = 2020
)$return
loss <- loss[201:length(loss)]

var95 <- historical_var(loss, 0.95)
es95 <- historical_es(loss, 0.95)
c(VaR = var95, ES = es95)
```

Result

Historical 95% VaR is **2.037% loss** and ES is **2.775% loss**. The latter is larger because it averages the selected tail.

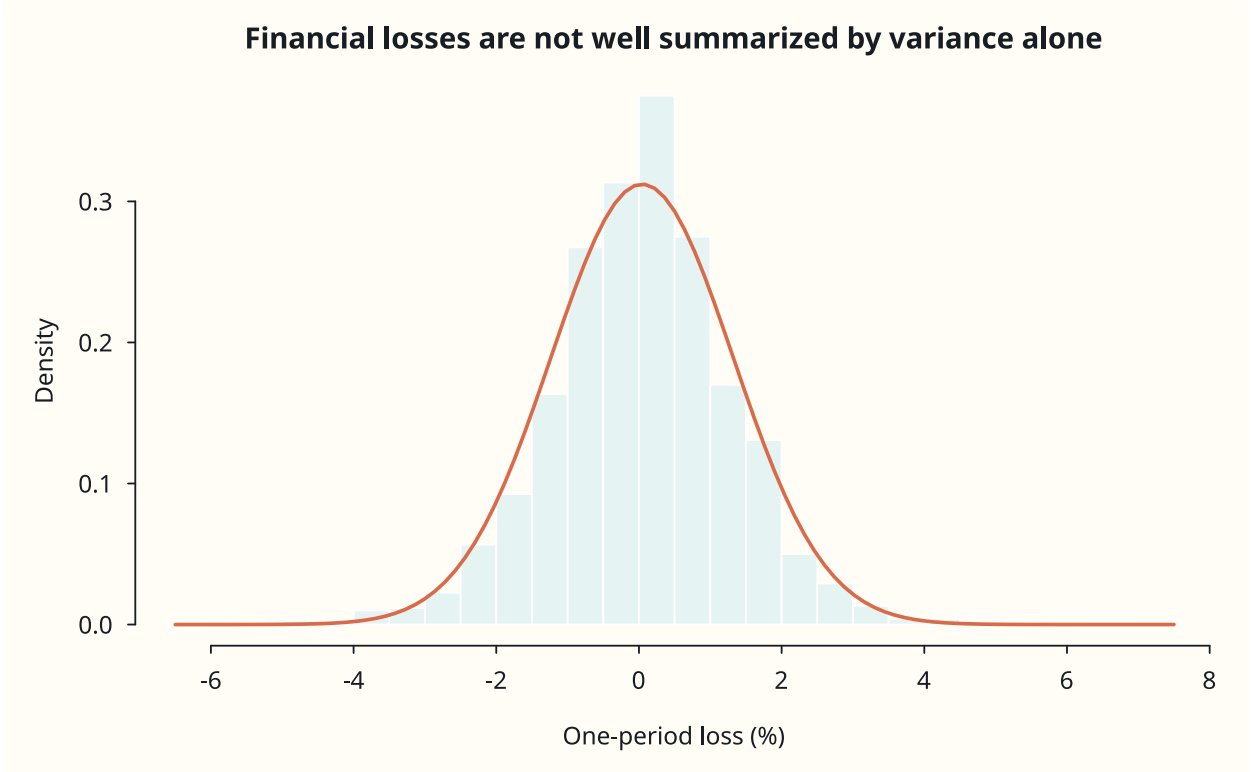

Figure 20.1: Simulated GARCH loss distribution with a same-mean-and-SD normal density.

Interpretation. These are simulated one-period returns, not a claim about a real portfolio. The normal overlay

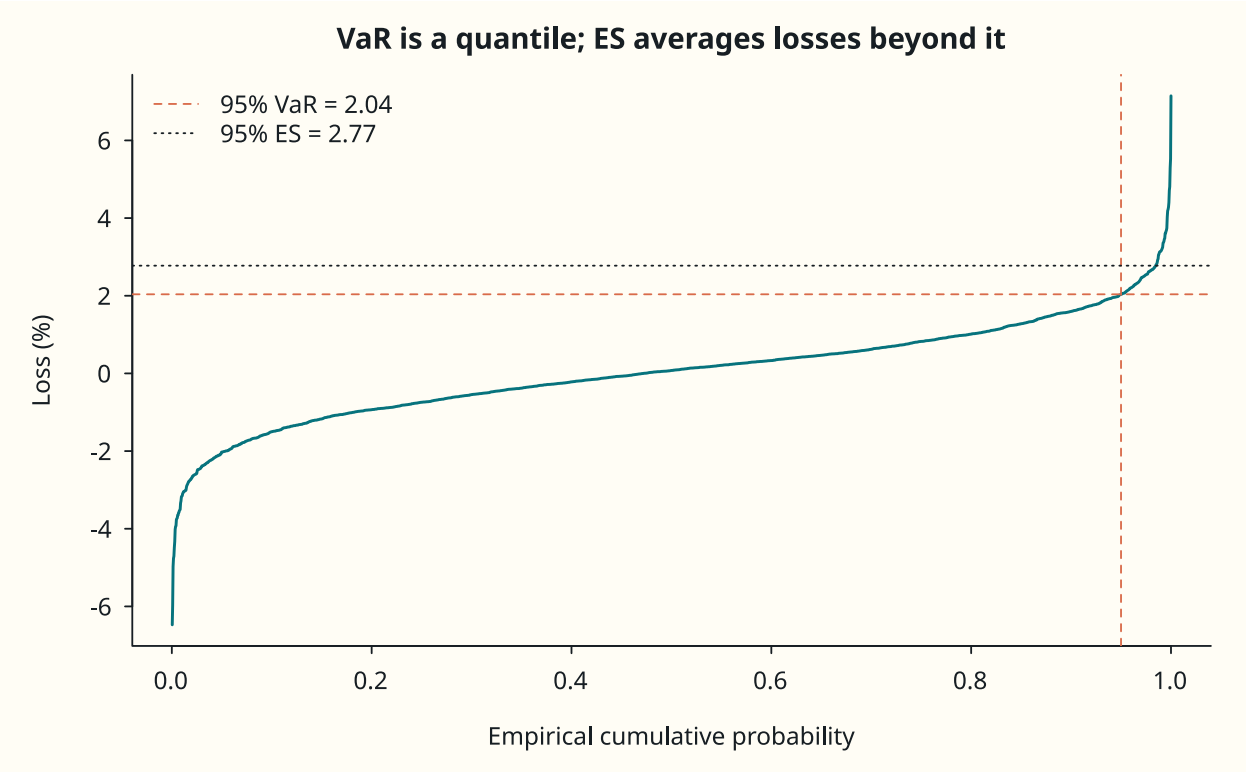

Figure 20.2: Empirical loss quantiles with historical 95% VaR and expected shortfall marked.

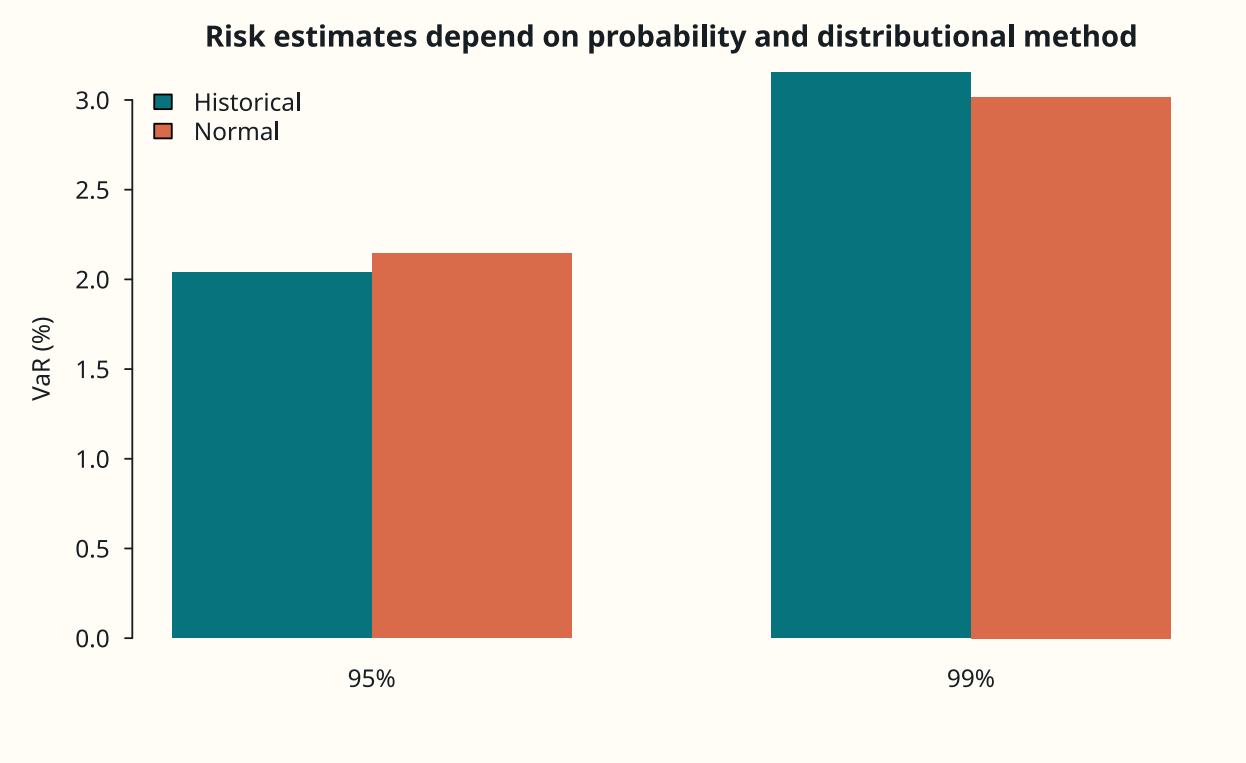

Figure 20.3: Historical and normal VaR estimates at 95% and 99% probabilities.

misses aspects of the GARCH mixture distribution.

20.6 Worked example 2: Backtest a rolling VaR and inspect the EVT threshold

```
window <- 250
rolling_var <- rep(NA_real_, length(loss))
for (i in seq.int(window + 1, length(loss))) {
  rolling_var[i] <- quantile(
    loss[(i - window):(i - 1)], 0.95,
    names = FALSE
  )
}
breach <- loss > rolling_var
mean(breach, na.rm = TRUE)
```

Result

The rolling exception rate is **5.30%**. At the full-sample 95% threshold there are **120 exceedances** for EVT analysis.

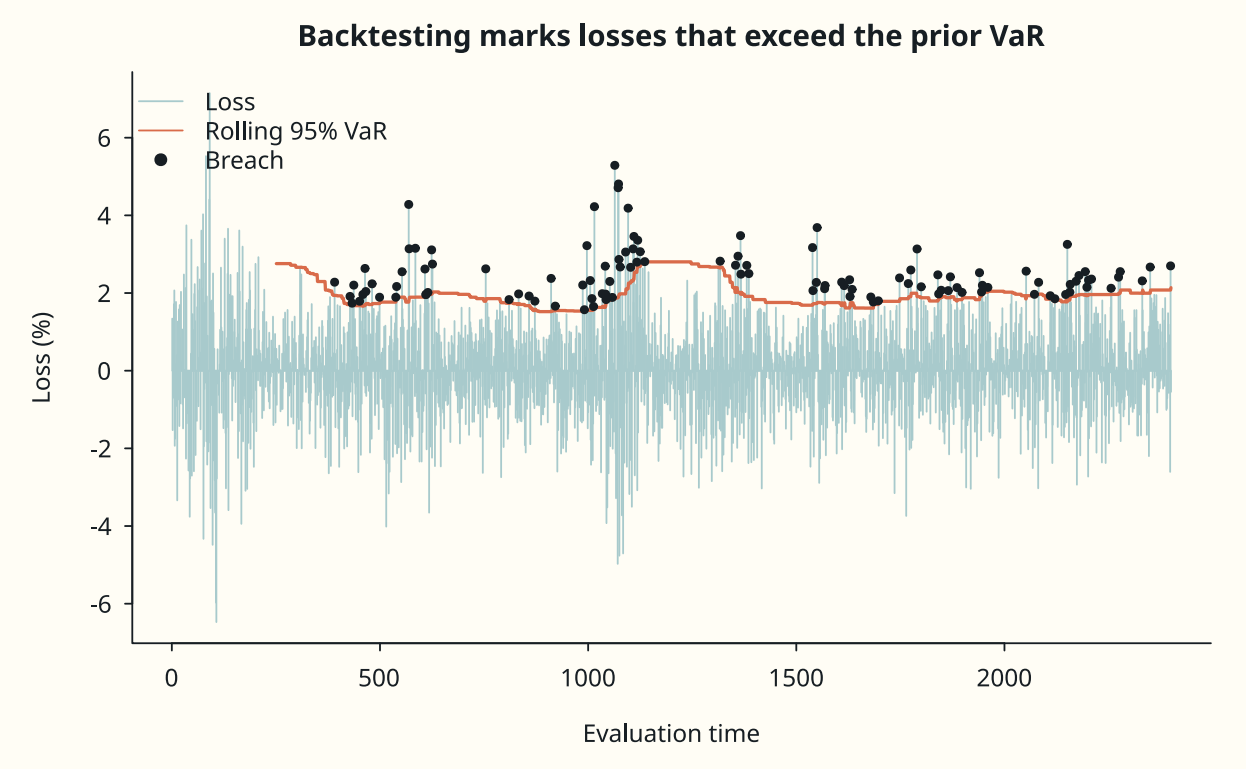

Figure 20.4: Simulated losses, prior 250-observation historical VaR forecasts, and exceptions.

Interpretation. A rate near 5% is necessary but not sufficient: exception clustering and the model-selection process also matter. The threshold plot is a diagnostic, not proof that a generalized Pareto tail is correct.

Common mistake

Do not change loss signs halfway through a workflow. If losses are positive, upper quantiles measure risk; if returns are used directly, lower quantiles require a sign conversion.

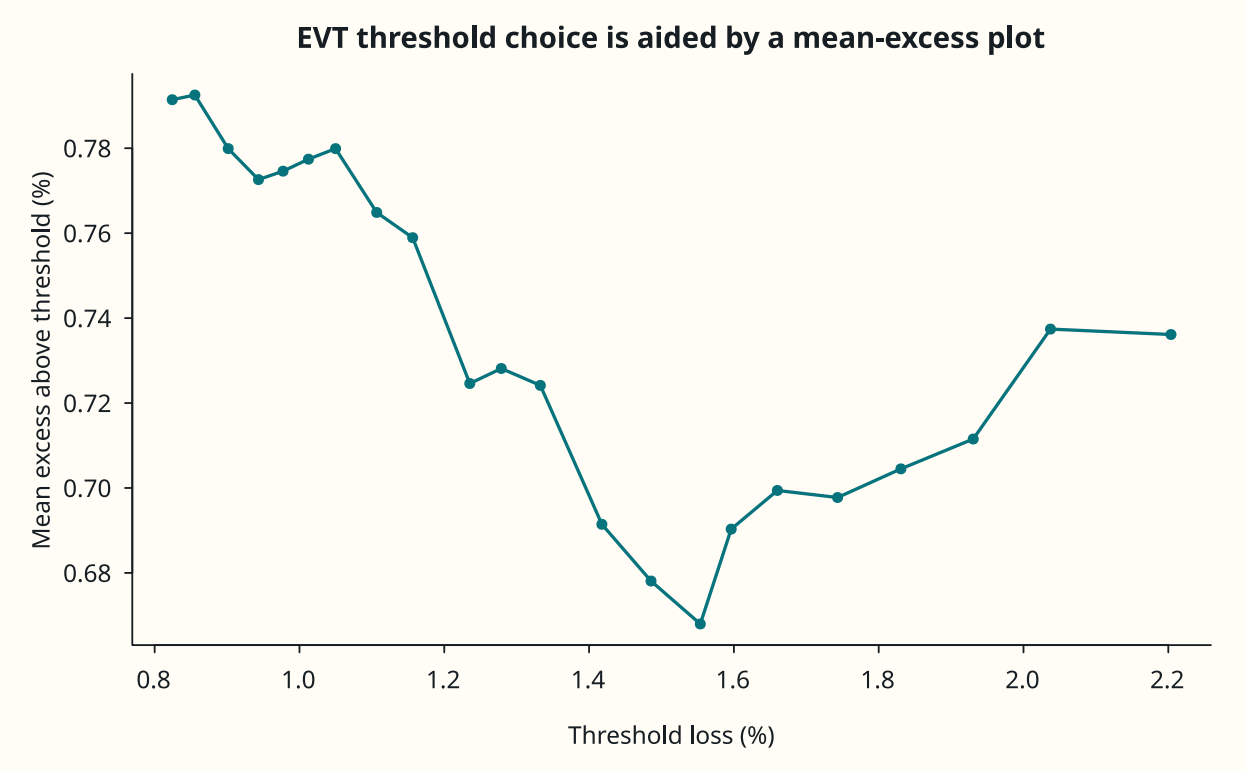

Figure 20.5: Mean excess above candidate loss thresholds from the simulated sample.

Practical tip

Write unit tests for monotonicity: 99% VaR should not be below 95% VaR on the same loss sample, and ES should not be below VaR for a continuous upper tail.

20.7 Guided practice

Repeat the rolling backtest at 99%. Compare expected and observed exception counts, then inspect whether the few exceptions cluster.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

20.8 Exercises

1. Compute historical 99% VaR and ES.
2. Implement normal-theory VaR and explain why its ES needs a different formula.
3. Fit mean excess over thresholds from the 80th to 97th percentile and identify a plausible stable region with caveats.

20.9 Selected solutions

Exercise 1.

```
var99 <- historical_var(loss, .99)
es99 <- historical_es(loss, .99)
```

```
c(var99 = var99, es99 = es99)
```

Exercise 2.

```
normal_var95 <- mean(loss) + sd(loss) * qnorm(.95)
normal_es95 <- mean(loss) + sd(loss) * dnorm(qnorm(.95)) / (1 - .95)
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

20.10 Chapter summary

- VaR is a quantile; ES measures average tail severity.
- Historical, parametric, filtered, and simulation methods embed different assumptions.
- Backtests must use prior-only forecasts and assess coverage plus clustering.
- EVT threshold choice and extrapolation require stability and sensitivity analysis.

20.11 Chapter cheat sheet

Table 20.1: Chapter 20 command cheat sheet.

Command or pattern	Purpose
`quantile(loss,p)`	historical VaR
`mean(loss >= VaR)`	exception rate
`PerformanceAnalytics::VaR()`	portfolio VaR
`PerformanceAnalytics::ES()`	expected shortfall
`evir::gpd()`	GPD tail fit
`rugarch::ugarchroll()`	rolling filtered forecasts

CHAPTER 21

Multivariate Time Series and Cointegration

21.1 Motivation

Economic and financial variables interact. VAR models trace joint dynamics, Granger tests evaluate incremental predictability, cointegration represents stable long-run combinations, and multivariate volatility models describe changing co-risk.

21.2 Learning objectives

By the end of this chapter, you should be able to:

- inspect cross-correlations without assigning causality
- estimate and diagnose a VAR
- interpret Granger-causality tests and impulse responses
- distinguish cointegration from ordinary correlation
- compare CCC, DCC, and BEKK volatility structures

21.3 Packages and data

Base R tested VAR calculations; optional `vars` 1.6-1, `urca`, `tsDyn`, `rmgarch`, and `MTS`. Foundations build on Engle & Granger (1987) and modern VAR practice.

The complete tested script is `scripts/ch21_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

Multivariate models can improve forecasts and risk estimates, but identification claims require restrictions beyond predictive fit.

21.4 Core ideas

A VAR(p) treats every component of y_t as endogenous:

$$y_t = c + A_1 y_{t-1} + \cdots + A_p y_{t-p} + \varepsilon_t.$$

Each equation can be estimated by OLS when regressors are shared. Lag selection balances dynamics and parameters; residual serial correlation, stability roots, and changing covariance must be checked. Forecasts iterate the system jointly.

Variable x Granger-causes y if past x improves prediction of y conditional on included past information. This is incremental predictability, not philosophical or intervention causality. Results depend on lag length, transformations, information set, breaks, and measurement timing. Cross-correlation is still more descriptive and can be distorted by common trends.

An impulse response traces the fitted system after a specified innovation. Innovations are contemporaneously correlated, so identifying a structural shock requires restrictions. Cholesky ordering is convenient but order-dependent; generalized responses or substantive structural restrictions offer alternatives. Confidence bands should reflect parameter uncertainty.

Two $I(1)$ series are cointegrated if a nonzero linear combination is stationary. A vector error-correction model separates short-run changes from adjustment toward the long-run relation. Engle–Granger is simple for one relation; Johansen methods allow multiple relations. Spurious regression between trending levels can show high R^2 without cointegration.

For conditional covariance H_t, constant conditional correlation (CCC) combines univariate variances with a fixed correlation matrix. Dynamic conditional correlation (DCC) lets correlation evolve parsimoniously. BEKK parameterizes covariance dynamics to preserve positive definiteness but scales heavily. Evaluate covariance forecasts against portfolio or hedging losses, not only likelihood.

21.5 Worked example 1: Describe lead–lag association and a VAR response

```
inflation <- scale(macro$inflation_yoy_percent)[, 1]
rate <- scale(macro$policy_rate_percent)[, 1]
cc <- ccf(inflation, rate, lag.max = 18)

stationary_data <- cbind(
  inflation_change = diff(inflation),
  policy_change = diff(rate)
)
var_model <- fit_var1(stationary_data)
irf <- var1_irf(var_model, impulse = 2, response = 1, horizon = 18)
```

Result

The largest absolute sample cross-correlation is **0.829**. The impulse response is based on a one-standard-deviation policy innovation in a compact VAR(1).

Interpretation. Neither display identifies monetary-policy causality. The simple response uses a particular covariance-based shock and omits other macroeconomic variables and real-time data revisions.

21.6 Worked example 2: Simulate and recover a cointegrating relation

```
set.seed(2121)
x <- cumsum(rnorm(220, sd = 0.8))
stationary_spread <- arima.sim(list(ar = 0.55), n = 220, sd = 0.5)
y <- 4 + 1.25 * x + stationary_spread

cointegration_fit <- lm(y ~ x)
coef(cointegration_fit)["x"]
acf(residuals(cointegration_fit))
```

Result

The estimated cointegrating slope is **1.244**, close to the generating value 1.25; the residual spread visibly returns toward zero.

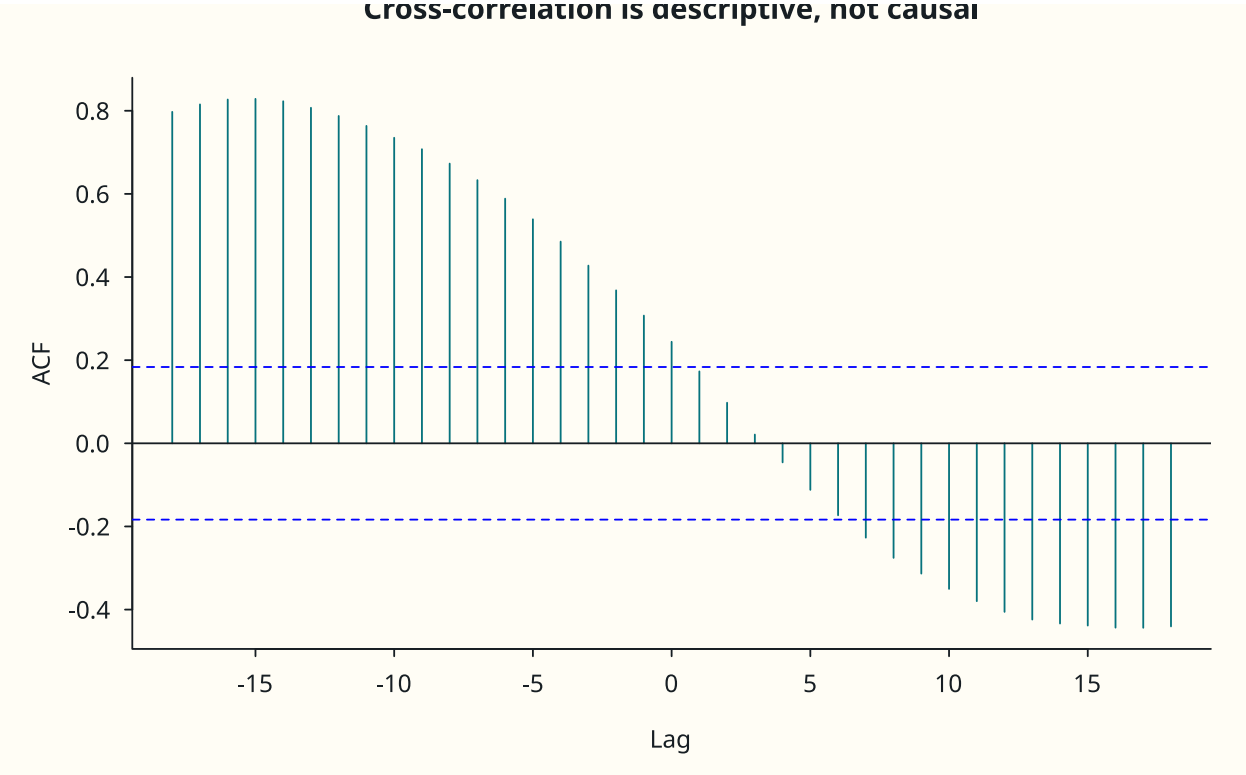

Figure 21.1: Cross-correlation function for standardized Canadian inflation and policy rate.

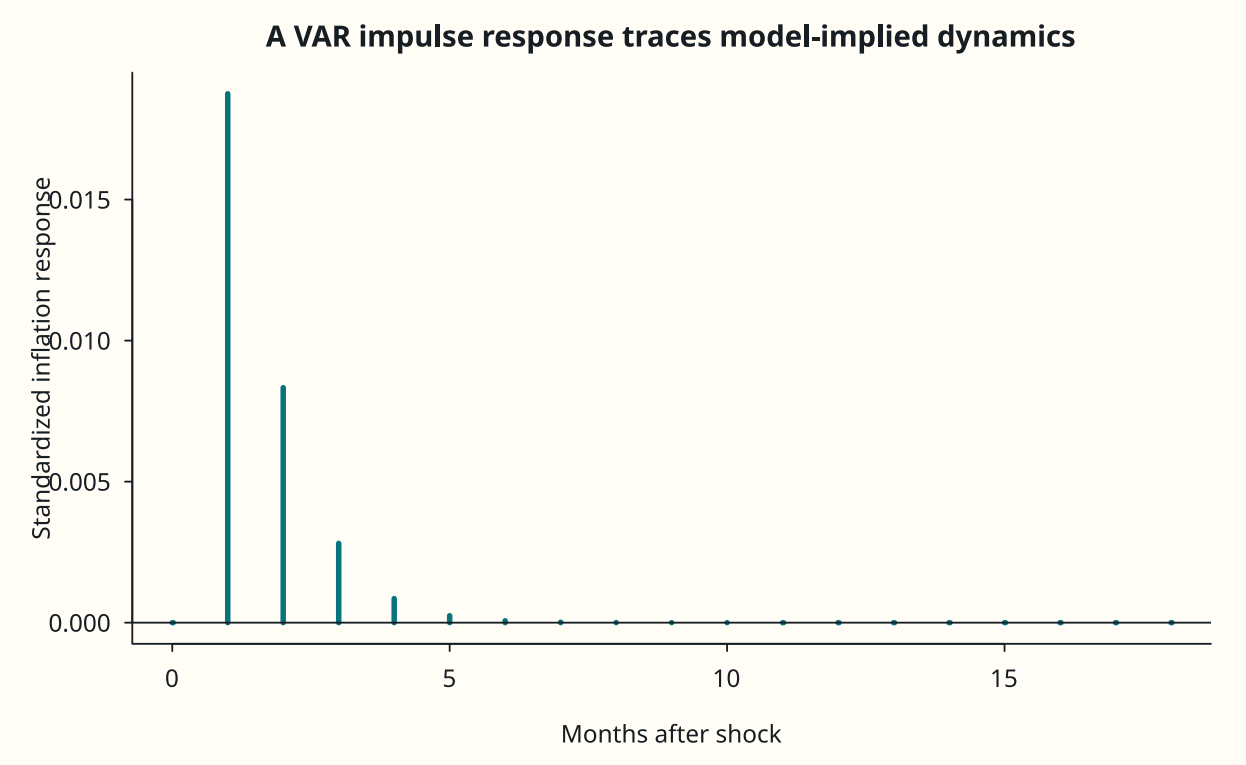

Figure 21.2: VAR-implied standardized inflation response after a policy-change innovation.

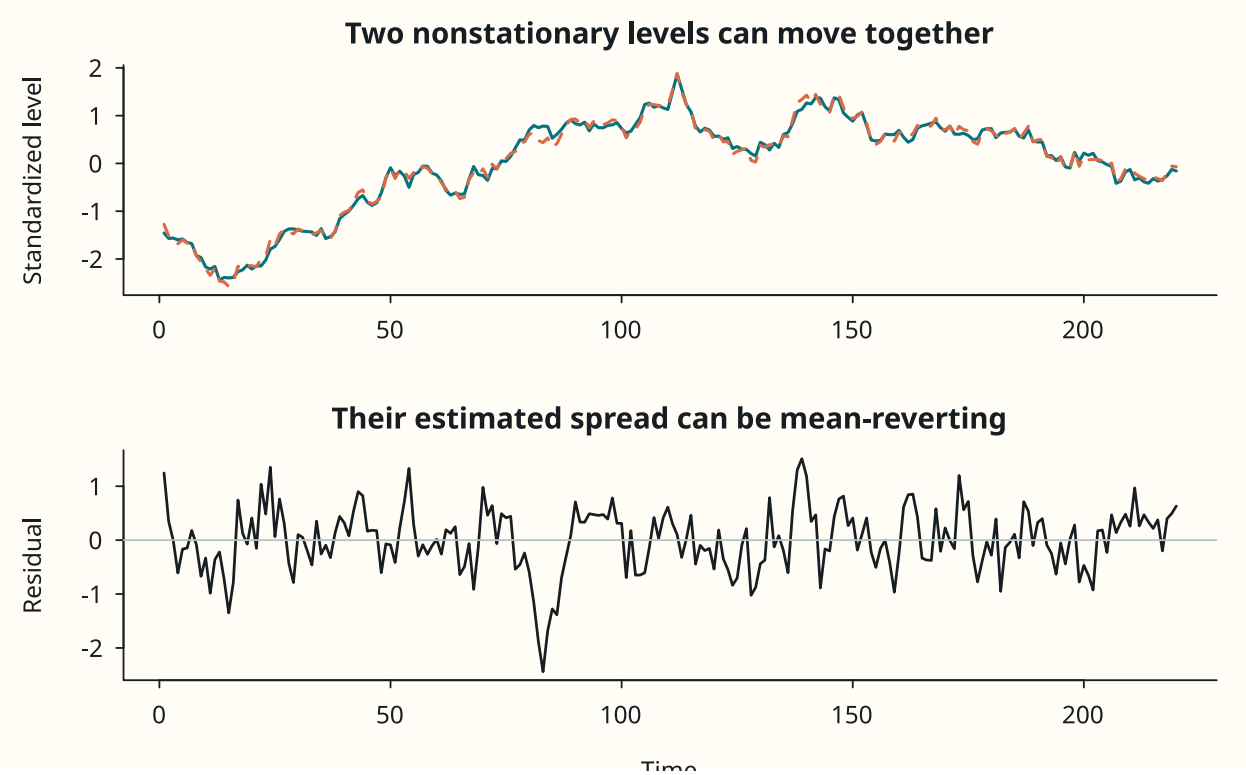

Figure 21.3: Two simulated nonstationary levels and their estimated mean-reverting cointegration residual.

Interpretation. This controlled simulation illustrates the definition. With observed data, both integration order and residual stationarity require formal and graphical assessment.

Common mistake

Do not run OLS on two trending levels and interpret a high R^2 as a relationship before checking integration and cointegration.

Practical tip

For every impulse-response plot, label the shock variable, shock size, identification rule, response units, horizon, and uncertainty method.

21.7 Guided practice

Reverse the ordering of the two variables in a Cholesky-identified VAR and compare responses. Explain what changes and why ordering is an assumption.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

21.8 Exercises

1. Fit VAR(1) and VAR(2) candidates and compare AIC plus residual serial correlation.
2. Test whether the simulated cointegration residual behaves more stationarily than either level.
3. Describe when DCC is preferable to estimating every covariance coefficient freely.

21.9 Selected solutions

Exercise 1.

```
# With the vars package:
fit1 <- vars::VAR(stationary_data, p = 1, type = "const")
fit2 <- vars::VAR(stationary_data, p = 2, type = "const")
vars::serial.test(fit1); vars::serial.test(fit2)
```

Exercise 2.

```
acf(x); acf(y); acf(residuals(cointegration_fit))
# Add ADF/KPSS tests with deterministic terms chosen explicitly.
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

21.10 Chapter summary

- VAR models joint lag dynamics among endogenous variables.
- Granger causality is conditional predictive content, not intervention causality.
- Impulse responses require shock identification.
- Cointegration creates stationary combinations of nonstationary levels.
- Multivariate volatility balances flexibility, positive definiteness, and scalability.

21.11 Chapter cheat sheet

Table 21.1: Chapter 21 command cheat sheet.

Command or pattern	Purpose
`ccf()`	cross-correlation
`vars::VAR()`	vector autoregression
`vars::causality()`	Granger test
`vars::irf()`	impulse response
`urca::ca.jo()`	Johansen cointegration
`rmgarch::dccfit()`	DCC-GARCH estimation

CHAPTER 22

Case Study: Forecasting Canadian Inflation

22.1 Motivation

The final chapter assembles the complete workflow: source documentation, validation, exploration, time-aware feature construction, benchmarks, ARIMA forecasting, diagnostics, holdout scoring, and responsible communication.

22.2 Learning objectives

By the end of this chapter, you should be able to:

- write a forecast question with target, horizon, and information set
- build a frozen and documented modelling table
- compare transparent benchmark and ARIMA forecasts
- score a chronological holdout with RMSE and MAE
- write a decision-ready conclusion with limitations and reproducibility metadata

22.3 Packages and data

Base `stats`, with `dplyr`, `ggplot2`, `forecast`, or `fable` as maintainable extensions. Official inputs are Statistics Canada and Bank of Canada data frozen on 2026-08-11.

The complete tested script is `scripts/ch22_examples.R`. Figures and numerical results were regenerated from the frozen inputs under R 4.6.0; the validation record is in `outputs/validation_summary.csv`.

Why this matters

An end-to-end case study exposes dependencies that isolated examples hide: definitions affect transformations, transformations affect models, and evaluation design affects conclusions.

22.4 Core ideas

The task is to forecast monthly Canadian year-over-year CPI inflation, using information in the frozen table, over an 18-month holdout. The target definition, vintage, horizon, and scoring rule are fixed before fitting. This is an educational pseudo-real-time exercise because the CPI column is the latest revised vintage rather than a sequence of historical vintages.

The modelling table passes schema, missingness, uniqueness, order, and range checks. The dashboard reveals a sharp inflation rise and decline, policy tightening after inflation increased, and exchange-rate variation. Those features warn against causal interpretations and suggest possible structural change. With only 114 months, a parsimonious model is preferable to a large predictor set.

Three candidates are compared: a last-observation naive forecast, a constant mean of the latest 36 training months, and ARIMA(1,1,1). All use the identical training endpoint and horizon. RMSE weights large errors quadratically; MAE weights absolute errors linearly. A model's rank can depend on the loss aligned with the decision.

The case study is not a production Bank of Canada forecast. It lacks real-time vintages, survey expectations, commodity prices, output-gap measures, release calendars, parameter-uncertainty integration, and scenario analysis. Model selection used the same sample that demonstrates the workflow, so reported performance is illustrative. A production system would use rolling origins, monitored drift, approval controls, and scheduled refresh with data-vintage retention.

22.5 Worked example 1: Validate and visualize the modelling table

```
macro <- read.csv("data/canada_macro_monthly.csv")
macro$date <- as.Date(macro$date)

required <- c("date", "cpi_all_items_2002_100",
      "inflation_yoy_percent", "policy_rate_percent",
      "usd_cad_monthly_average")
stopifnot(setequal(names(macro), required),
  !anyNA(macro), !anyDuplicated(macro$date),
  all(diff(macro$date) > 0))
```

Result

All checks pass for **114 months**, January 2017–June 2026. The frozen latest inflation value is **2.798%**.

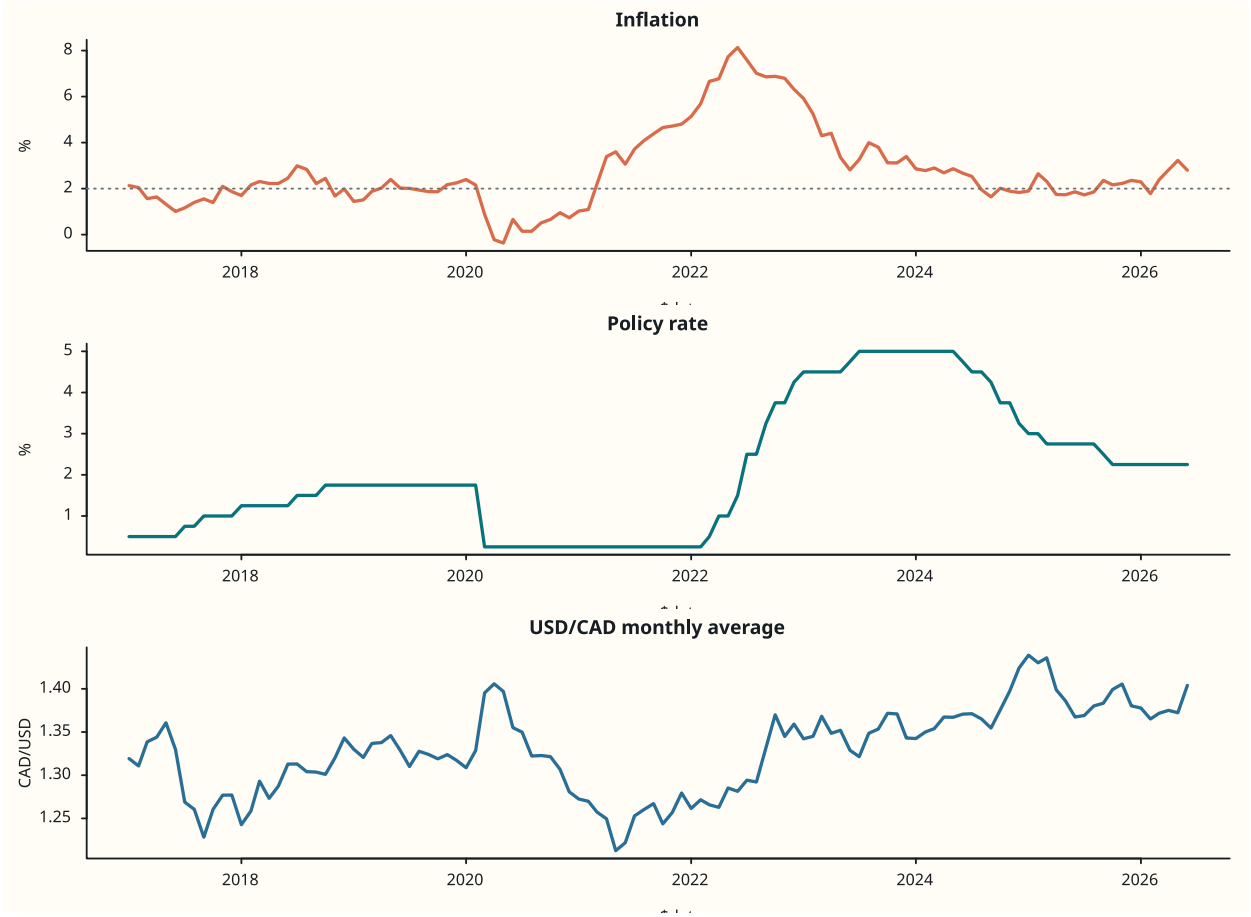

Figure 22.1: Aligned panels for Canadian inflation, policy rate, and USD/CAD in the frozen case-study table.

Interpretation. A passing check shows the table satisfies declared mechanical conditions; it does not establish that

every source value is economically correct. Provenance and spot checks remain necessary.

22.6 Worked example 2: Compare three forecasts on one holdout

```
holdout <- 18
train <- head(macro$inflation_yoy_percent, -holdout)
actual <- tail(macro$inflation_yoy_percent, holdout)

naive <- rep(tail(train, 1), holdout)
mean36 <- rep(mean(tail(train, 36)), holdout)
fit <- arima(train, order = c(1, 1, 1), method = "ML")
arima_fc <- as.numeric(predict(fit, n.ahead = holdout)$pred)

rbind(
  Naive = c(RMSE = rmse(actual, naive), MAE = mae(actual, naive)),
  Mean36 = c(RMSE = rmse(actual, mean36), MAE = mae(actual, mean36)),
  ARIMA = c(RMSE = rmse(actual, arima_fc), MAE = mae(actual, arima_fc))
)
```

Result

ARIMA has the lowest holdout RMSE, **0.581 percentage points**, and its MAE is **0.437 points**.

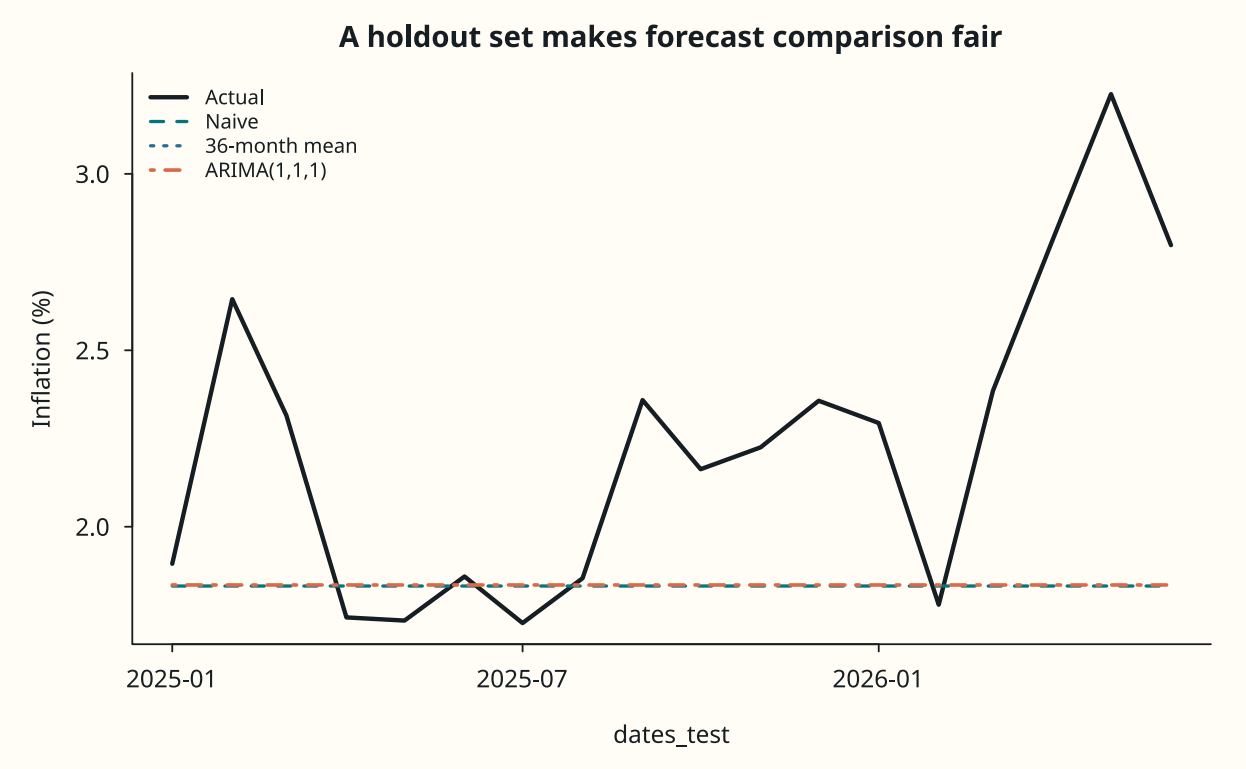

Figure 22.2: Actual holdout inflation and naive, 36-month-mean, and ARIMA forecasts.

Interpretation. The result supports ARIMA for this one exercise, not universally. A different origin, vintage, horizon, or loss can change the ranking.

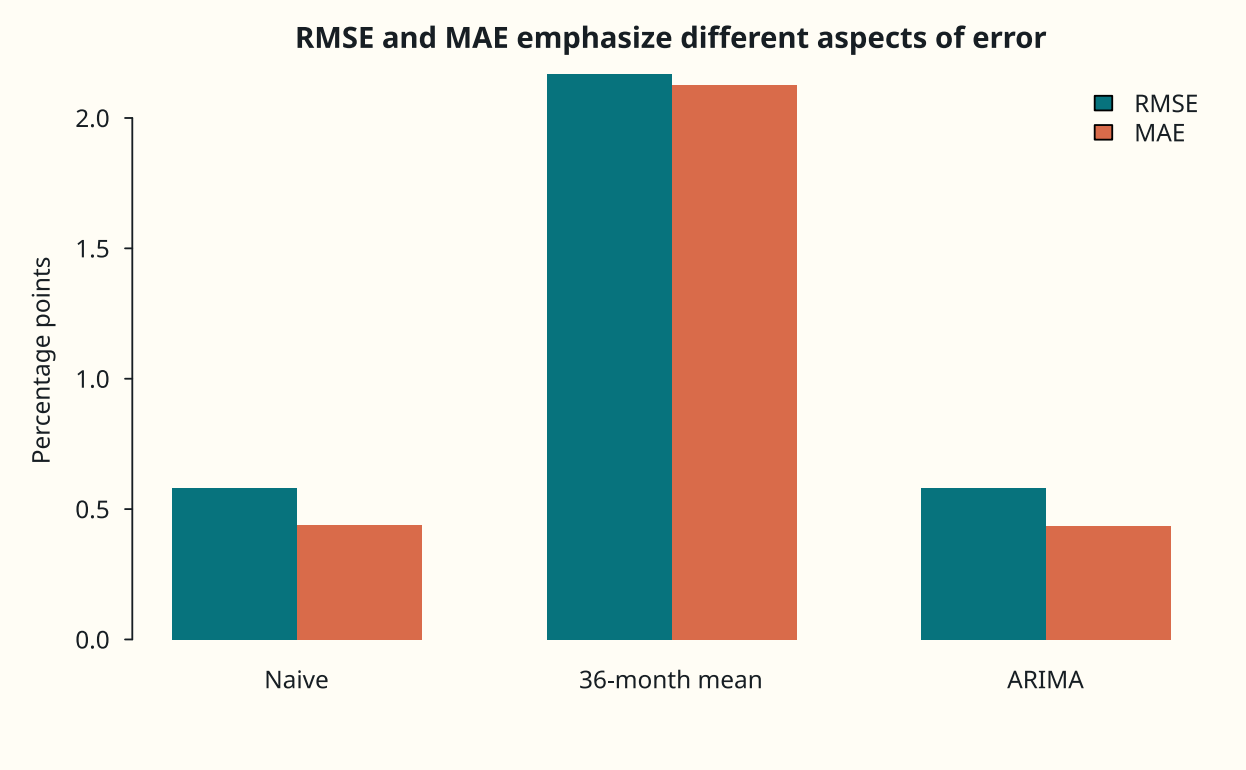

Figure 22.3: RMSE and MAE for the three forecast candidates on the same 18-month holdout.

Common mistake

Do not select the best model on the holdout and then report that same score as untouched final evidence. Preserve a final test period or use nested/rolling evaluation.

Practical tip

Write the limitations paragraph before the executive conclusion. It naturally calibrates the claim and prevents a benchmark win from becoming an exaggerated recommendation.

22.7 Guided practice

Move the forecast origin back six months and repeat the comparison. Record whether the winner changes and what this says about evaluation variance.

Before looking at a solution, write down the expected object type, unit, and one validation check. Run the complete chapter script after your change to ensure earlier outputs still reproduce.

22.8 Exercises

1. Add a seasonal-naive benchmark and justify whether it is appropriate for year-over-year inflation.
2. Design a rolling-origin evaluation table with origin, horizon, model, actual, forecast, and error columns.
3. Draft a 150-word forecast memo stating target, horizon, best model, error, uncertainty, and at least three limitations.

22.9 Selected solutions

Exercise 1.

```
seasonal_naive <- tail(train, 12)[seq_len(holdout) %% 12 + 1]
# Check indexing and target definition before using this benchmark.
```

Exercise 2.

```
score <- function(actual, forecast) {
  c(RMSE = sqrt(mean((actual - forecast)^2)),
    MAE = mean(abs(actual - forecast)))
}
```

Solutions show one defensible route, not the only valid program. Confirm object dimensions and units, and explain any changed modelling choices.

22.10 Chapter summary

- Forecast questions require a target, horizon, origin, information set, and loss.
- Data validation and provenance precede model fitting.
- Benchmarks and identical holdouts make comparisons interpretable.
- One holdout result is evidence with sampling and model-selection uncertainty.
- A responsible conclusion states scope, limitations, and reproducibility metadata.

22.11 Chapter cheat sheet

Table 22.1: Chapter 22 command cheat sheet.

Command or pattern	Purpose
`stopifnot()`	validation gate
`head(x,-h)`	training split
`tail(x,h)`	holdout split
`arima()`	candidate model
`predict()`	forecast
`rmse()/mae()`	holdout scoring

Part IV

Appendices

CHAPTER A

Glossary

ACF. Autocorrelation function; correlation of a series with lagged versions of itself.

ARCH. Autoregressive conditional heteroskedasticity; variance driven by past squared innovations.

ARIMA. Autoregressive integrated moving-average model.

Backtest. Evaluation of forecasts or risk estimates using outcomes unavailable at forecast time.

Bootstrap. Approximation to a sampling distribution by resampling under a stated design.

Brownian motion. Continuous stochastic process with independent Gaussian increments.

Call option. Right, not obligation, to buy an underlying asset at a strike by specified terms.

Cointegration. Stationary linear combination of nonstationary variables.

Conditional mean. Expected value given the specified information set.

Conditional variance. Variance given the specified information set.

Cross-validation. Repeated train/assessment splits; for time series the order must be respected.

Data leakage. Use of information that would not be available at the prediction origin.

Data frame. Rectangular R object whose equal-length columns may have different types.

Delta. Local option-price sensitivity to the underlying spot price.

Dynamic regression. Regression whose errors follow a time-series process.

Expected shortfall. Mean loss in the tail beyond a specified probability threshold.

Factor. R class for finite categorical levels, optionally ordered.

Forecast horizon. Number of periods from origin to target.

GARCH. Generalized ARCH; variance recursion with past squared innovations and past variance.

GBM. Geometric Brownian motion; lognormal continuous-time price model.

Granger causality. Incremental predictive content of past variables conditional on an information set.

Holdout. Observations reserved from fitting for honest evaluation.

Implied volatility. Volatility input that makes an option-pricing model match a market price.

Innovation. New one-step-ahead model error after conditioning on past information.

Itô's lemma. Stochastic change-of-variables rule including a second-derivative term.

Join key. Variable or variables used to match table rows.

Log return. Difference in log price; additive across time.

MAE. Mean absolute error.

Missing value. Unknown or unavailable value represented by `NA` in R.

PACF. Partial autocorrelation after controlling for intervening lags.

Prediction interval. Model-based range intended to cover a future observation at a stated rate.

Realized variance. Sum of squared intraday returns over a period.

Regime. Region of a nonlinear model with its own dynamics.

Reproducibility. Ability to regenerate results from documented code, data, and environment.

Residual. Observed value minus fitted value; an estimate or proxy for an innovation.

RMSE. Root mean squared error.

Seasonality. Systematic variation tied to position in a fixed cycle.

Simple return. Price ratio minus one; compounds multiplicatively.

Stationarity. Stability of probabilistic properties under time shifts, in the specified sense.

STL. Seasonal-trend decomposition using LOESS.

Threshold. Value that separates regimes or defines exceedances.

Unit root. Autoregressive root of one, producing persistent stochastic trend.

VaR. Loss quantile at a stated horizon and probability.

VAR. Vector autoregression for joint lag dynamics.

Vector. One-dimensional R object whose atomic elements share a type.

Vega. Option-price sensitivity to volatility.

Volatility. Standard deviation, usually conditional or measured over a stated horizon.

White noise. Zero-mean, constant-variance, serially uncorrelated process.

CHAPTER B

Command index

Commands are alphabetized by their displayed form. Chapter cheat sheets provide the local context and complete scripts show tested use.

Command	Where used and purpose
`%in%`	Ch. 2: membership test
`<-`	Ch. 1: assign a value
`acf()`	Ch. 11: sample autocorrelation
`acf(r^2)`	Ch. 15: volatility-dependence diagnostic
`aes()`	Ch. 8: map variables
`AIC()`	Ch. 12: penalized fit comparison
`anyDuplicated()`	Ch. 3: validate key uniqueness; Ch. 5: duplicate-key check
`arima()`	Ch. 11: estimate ARMA/ARIMA; Ch. 12: fit ARIMA; Ch. 22: candidate model
`arima.sim()`	Ch. 11: simulate ARIMA process
`as.Date()`	Ch. 3: parse calendar dates
`Box.test()`	Ch. 11: portmanteau residual test; Ch. 12: residual autocorrelation test
`c()`	Ch. 2: combine values
`ccf()`	Ch. 21: cross-correlation
`coef()`	Ch. 9: extract coefficients
`colSums()`	Ch. 5: column totals
`confint()`	Ch. 9: confidence intervals
`cumprod(1+r)`	Ch. 14: wealth index
`cumsum(rnorm(...))`	Ch. 19: Brownian simulation
`cut()`	Ch. 3: bin numeric or date values
`data.frame()`	Ch. 3: construct a rectangular table
`density()`	Ch. 7: smoothed distribution
`diff()`	Ch. 10: first or seasonal difference
`diff(log(x))`	Ch. 14: log returns
`diff(timestamp)`	Ch. 18: event durations
`diff(x, lag=12)`	Ch. 13: seasonal difference
`ecdf()`	Ch. 7: empirical cumulative distribution
`evir::gpd()`	Ch. 20: GPD tail fit
`exp()`	Ch. 19: GBM transformation
`facet_wrap()`	Ch. 8: small multiples
`factor()`	Ch. 3: encode categories
`factor(month)`	Ch. 13: season indicators
`file.path()`	Ch. 4: portable path construction
`filter()`	Ch. 6: keep rows
`forecast::Arima()`	Ch. 12: extended ARIMA interface
`forecast::Arima(xreg=)`	Ch. 13: regression with ARIMA errors
`frequency()`	Ch. 10: seasonal frequency
`function()`	Ch. 4: define a reusable rule
`geom_line()`	Ch. 8: ordered paths
`geom_point()`	Ch. 8: scatterplot marks
`ggplot()`	Ch. 8: initialize a graphic
`group_by()`	Ch. 6: define groups
`head()/tail()`	Ch. 1: inspect boundary rows
`head(x,-h)`	Ch. 22: training split

Command	Where used and purpose
`highfrequency::rCov()`	Ch. 17: realized covariance
`hist()`	Ch. 7: binned distribution
`ifelse()`	Ch. 18: threshold rule
`is.na()`	Ch. 2: identify missing values; Ch. 5: missingness indicator
`labs()`	Ch. 8: titles and units
`lag()`	Ch. 10: time shift; Ch. 13: distributed-lag feature
`left_join()`	Ch. 6: add matched columns
`lm()`	Ch. 9: fit linear regression; Ch. 13: mean regression
`MASS::polr()`	Ch. 18: ordered logit/probit
`mean()/median()`	Ch. 7: centre
`mean(loss >= VaR)`	Ch. 20: exception rate
`merge()`	Ch. 14: calendar alignment
`model='apARCH'`	Ch. 16: APARCH specification
`model='eGARCH'`	Ch. 16: EGARCH specification
`mutate()`	Ch. 6: create or change columns
`newsimpact()`	Ch. 16: fitted impact curve
`nnet::nnet()`	Ch. 18: neural network
`pacf()`	Ch. 11: partial autocorrelation
`PerformanceAnalytics::ES()`	Ch. 20: expected shortfall
`PerformanceAnalytics::Return.calculate()`	Ch. 14: return transformation
`PerformanceAnalytics::VaR()`	Ch. 20: portfolio VaR
`pmax()`	Ch. 19: option payoff
`pnorm()`	Ch. 18: ordered-probit probabilities; Ch. 19: normal CDF
`predict()`	Ch. 12: forecast and standard errors; Ch. 22: forecast
`quantile()`	Ch. 7: ordered cut points
`quantile(loss,p)`	Ch. 20: historical VaR
`quantmod::getSymbols()`	Ch. 14: provider retrieval
`read.csv()`	Ch. 1: import a CSV file
`readr::read_csv()`	Ch. 5: typed CSV import
`readxl::read_excel()`	Ch. 5: spreadsheet import
`replicate()`	Ch. 9: repeat a computation
`residuals()`	Ch. 9: model residuals; Ch. 12: extract innovations
`residuals(..., standardize=TRUE)`	Ch. 15: standardized residuals
`rmgarch::dccfit()`	Ch. 21: DCC-GARCH estimation
`rmse()/mae()`	Ch. 22: holdout scoring
`rowsum()`	Ch. 17: aggregate intraday returns
`RQuantLib::EuropeanOption()`	Ch. 19: library option valuation
`rugarch::ugarchfit()`	Ch. 15: GARCH estimation
`rugarch::ugarchforecast()`	Ch. 15: variance forecast
`rugarch::ugarchroll()`	Ch. 20: rolling filtered forecasts
`rugarch::ugarchspec()`	Ch. 15: model specification
`rugarch::ugarchspec(model='gjrGARCH')`	Ch. 16: GJR specification
`sample()`	Ch. 9: resample values
`sd()/IQR()`	Ch. 7: spread
`select()`	Ch. 6: keep columns
`seq()`	Ch. 2: construct an index sequence
`sessionInfo()`	Ch. 4: record runtime versions
`set.seed()`	Ch. 4: reproduce random draws
`sigma()`	Ch. 15: conditional volatility
`sin()/cos()`	Ch. 13: Fourier terms
`sqrt(k) * sd(r)`	Ch. 17: annualized volatility
`stl()`	Ch. 10: seasonal-trend decomposition
`stochvol::svsample()`	Ch. 16: stochastic-volatility sampling
`stopifnot()`	Ch. 4: assert input conditions; Ch. 5: validation assertion; Ch. 22: validation gate

Command	Where used and purpose
`str()`	Ch. 1: inspect structure and types; Ch. 3: show column classes
`sum(r^2)`	Ch. 17: realized variance
`summarise()`	Ch. 6: reduce rows
`summary()`	Ch. 1: quick numerical diagnostics
`tail(x,h)`	Ch. 22: holdout split
`ts()`	Ch. 10: regular time-series object
`tsDyn::setar()`	Ch. 18: threshold autoregression
`TTR::runSD()`	Ch. 17: rolling SD
`typeof()`	Ch. 2: inspect storage type
`uniroot()`	Ch. 19: implied-volatility inversion
`urca::ca.jo()`	Ch. 21: Johansen cointegration
`urca::ur.df()`	Ch. 11: augmented Dickey–Fuller test
`variance.targeting=TRUE`	Ch. 16: variance-targeting option
`vars::causality()`	Ch. 21: Granger test
`vars::irf()`	Ch. 21: impulse response
`vars::VAR()`	Ch. 21: vector autoregression
`which.max()`	Ch. 1: locate a maximum
`window()`	Ch. 10: time interval subset
`x[condition]`	Ch. 2: logical subset
`xts::xts()`	Ch. 14: dated market series
`zoo::rollapply()`	Ch. 17: rolling window
`|>`	Ch. 4: native pipe

CHAPTER C

Data sources and dictionary

C.1 Frozen Canadian macro dataset

`data/canada_macro_monthly.csv` contains 114 monthly observations from January 2017 through June 2026. It was frozen on 2026-08-11 so every worked example remains reproducible.

Variable	Unit	Definition and transformation
`date`	month	First day used to label the observation month.
`cpi_all_items_2002_100`	index	Canada all-items CPI, 2002=100.
`inflation_yoy_percent`	percent	$100(CPI_t/CPI_{t-12} - 1)$.
`policy_rate_percent`	percent	Last available target overnight rate in the month.
`usd_cad_monthly_average`	CAD per USD	Mean of available daily Bank of Canada averages.

C.2 Official sources

CPI is Statistics Canada table 18-10-0004-01, Canada, All-items, vector `v41690973` (Statistics Canada, 2026). It is reused under the Statistics Canada Open Licence. Policy rate is Bank of Canada Valet series `V39079`; exchange rate is `FXUSDCAD` (Bank of Canada, 2026).

`scripts/prepare_frozen_data.py` retrieves and transforms the official sources. `data/DATA_DICTIONARY.md` and `data/SOURCES.md` preserve field definitions, URLs, attribution, and rebuild instructions. A future rebuild may differ because official data can be revised; the included CSV is canonical for Version 1.0.

CHAPTER D

Packages and reproducibility

D.1 Tested environment

All chapter scripts were executed in R 4.6.0 (2026-04-24). The tidy-data examples used `dplyr` 1.2.1 and `ggplot2` 4.0.3. Base R performs the time-series, simulation, risk, VAR, and option-pricing calculations so core examples remain transparent.

D.2 Current package set for extended work

Task	Package version checked for Version 1.0
Tidy workflow	`tidyverse` 2.0.0; `dplyr` 1.2.1; `tidyr` 1.3.2
Graphics	`ggplot2` 4.0.3; `patchwork` 1.3.2
Dates and import	`lubridate` 1.9.5; `readr` 2.2.0; `readxl` 1.5.0
Dated financial data	`xts` 0.14.2; `zoo` 1.9-0; `quantmod` 0.4.29
Forecasting	`forecast` 9.0.2; `fable` 0.5.0
Unit roots and VAR	`tseries` 0.10-62; `urca` 1.3-4; `vars` 1.6-1
Volatility	`rugarch` 1.5-6
Performance/risk	`PerformanceAnalytics` 2.1.0

Versions were checked against current repositories on 2026-08-11; they are a tested publication record, not a demand to pin all future analysis forever. Use a project library such as `renv` when exact package restoration is required.

D.3 Regenerate and validate

```
Rscript scripts/run_all_examples.R
bash scripts/build_book.sh
```

The first command regenerates 51 SVG figures, 47 worked-result rows, `outputs/session_info.txt`, and a validation summary. The second converts vector figures for PDF and runs Pandoc/XeLaTeX. No script changes the working directory, installs packages, or downloads live market data during rendering.

CHAPTER E

Accessibility and figure descriptions

The PDF uses high-contrast navy/teal/coral styling, redundant line types where series overlap, scalable vector figures, descriptive captions, running headers, and bookmarks. Source `.qmd` files carry `fig-alt` text. The table below provides a plain-text description for every figure and can be used when republishing to HTML or another tagged format.

Figure file	Text alternative
`ch01-first-canadian-analysis`	Canadian year-over-year inflation in the final 36 months of the frozen teaching data; the dashed line is a 2% reference, not a forecast.
`ch01-analysis-workflow`	Six connected stages of a reproducible data analysis: question, import, tidy, explore, model, and communicate.
`ch02-vectorized-transformations`	Two indexed paths constructed with vectorized arithmetic; both begin at 100 but encode different transformations.
`ch02-logical-subsetting`	Monthly inflation shown as vertical marks, with observations above 4% highlighted.
`ch03-data-structure-map`	Schematic of the Canadian data frame showing 114 rows and the type of each column.
`ch03-factor-counts`	Observation counts for the three ordered economic-era factor levels.
`ch04-function-contract`	Future value over 0 to 20 years using the validated compounding function.
`ch04-project-paths`	Project root connected to data, scripts, and figures through relative paths.
`ch05-missingness-map`	Binary missingness map for the constructed cleaning example; coral cells mark missing values.
`ch05-cleaning-before-after`	Constructed policy-rate data with an impossible 99% value flagged for investigation.
`ch06-grouped-summary`	Mean inflation by economic era with plus or minus one within-era standard deviation.
`ch06-filter-mutate-summarise`	Four common dplyr verbs linked in a readable transformation pipeline.
`ch07-histogram-density`	Histogram and kernel density estimate of Canadian year-over-year inflation.
`ch07-boxplots-by-era`	Boxplots of Canadian inflation for three analyst-defined economic eras.
`ch08-small-multiples`	Three vertically aligned Canadian macroeconomic time series with separate y-scales.
`ch08-honest-scales`	The same annual inflation means shown with a zero and a truncated baseline.
`ch09-sampling-distribution`	Bootstrap distribution of means from 2,500 samples of 24 months, with a normal approximation and full-sample mean.
`ch09-linear-regression`	Scatterplot of Canadian inflation and the policy rate with an ordinary least-squares line.
`ch10-canadian-time-series`	Canadian CPI level and twelve-month inflation shown in aligned panels.
`ch10-stl-decomposition`	Robust STL decomposition of log Canadian CPI into data, seasonal, trend, and remainder panels.
`ch11-white-noise-and-ar`	Simulated white-noise and stationary AR(1) paths using the same time length.

Figure file	Text alternative
ch11-acf-pacf	Sample ACF and PACF of the simulated AR(1) series.
ch12-arima-forecast	Training data, actual holdout inflation, ARIMA forecasts, and model-based 95% intervals.
ch12-residual-diagnostics	ARIMA residuals over time and their sample autocorrelation function.
ch13-seasonal-subseries	Monthly CPI paths aligned by calendar month, one line per year.
ch13-differencing-comparison	Monthly first differences and twelve-month seasonal differences of log CPI.
ch14-policy-and-exchange-rate	Bank of Canada policy rate and USD/CAD monthly average shown with separate, labelled axes.
ch14-simple-and-log-returns	Simple versus log monthly USD/CAD returns with a 45-degree reference line.
ch15-garch-clustering	Simulated GARCH returns and their latent conditional standard deviation.
ch15-squared-return-acf	ACFs of simulated GARCH returns and squared returns.
ch16-news-impact-curve	Symmetric GARCH and asymmetric GJR news-impact curves across positive and negative shocks.
ch16-symmetric-asymmetric-volatility	Conditional volatility paths from symmetric GARCH and GJR recursions driven by the same standardized innovations.
ch17-rolling-fx-volatility	Twelve-month rolling annualized volatility of monthly USD/CAD log returns.
ch17-realized-volatility-sampling	Realized volatility estimates across intraday sampling intervals with a simulation target.
ch18-threshold-autoregression	Lag plot of a two-regime TAR simulation with different fitted slopes below and above zero.
ch18-ordered-probabilities	Low, middle, and high ordered-probit probabilities across a continuous predictor.
ch19-brownian-motion	One simulated Brownian path over a year with 252 increments.
ch19-geometric-brownian-motion	A geometric Brownian asset-price path driven by the same Brownian motion.
ch19-option-payoffs	European call and put terminal payoffs around a strike of 100.
ch19-black-scholes-sensitivity	Black–Scholes call values across spot and volatility, with contour lines.
ch20-loss-distribution	Simulated GARCH loss distribution with a same-mean-and-SD normal density.
ch20-var-and-expected-shortfall	Empirical loss quantiles with historical 95% VaR and expected shortfall marked.
ch20-var-method-comparison	Historical and normal VaR estimates at 95% and 99% probabilities.
ch20-var-backtest	Simulated losses, prior 250-observation historical VaR forecasts, and exceptions.
ch20-evt-mean-excess	Mean excess above candidate loss thresholds from the simulated sample.
ch21-cross-correlation	Cross-correlation function for standardized Canadian inflation and policy rate.
ch21-var-impulse-response	VAR-implied standardized inflation response after a policy-change innovation.
ch21-cointegration-spread	Two simulated nonstationary levels and their estimated mean-reverting cointegration residual.
ch22-canadian-macro-dashboard	Aligned panels for Canadian inflation, policy rate, and USD/CAD in the frozen case-study table.

Figure file	Text alternative
`ch22-forecast-comparison`	Actual holdout inflation and naive, 36-month-mean, and ARIMA forecasts.
`ch22-forecast-errors`	RMSE and MAE for the three forecast candidates on the same 18-month holdout.

CHAPTER F

Modernization notes

The supplied 2013 lecture PDFs were treated as a topic inventory, not copy-ready prose. Version 1.0 replaces provider-dependent examples, machine-specific paths, session-side installation, deprecated syntax, and unverified pseudo-code with a reproducible project and tested R.

Major changes include a tidyverse-first data workflow; official Canadian CPI, policy-rate, and FX sources; |> rather than hidden state; time-aware holdouts; residual and volatility diagnostics; explicit innovation distributions; VaR plus expected shortfall and backtesting; modern GJR/EGARCH/APARCH and stochastic-volatility framing; shock-identification warnings for VARs; and accessible, source-generated figures. A detailed file-by-file account appears in `MODERNIZATION_REPORT.md` in the source package.

References

Artzner, P., Delbaen, F., Eber, J.-M., & Heath, D. (1999). Coherent measures of risk. *Mathematical Finance*, *9*(3), 203–228. https://doi.org/10.1111/1467-9965.00068

Bank of Canada. (2026). *Valet API: Target for the overnight rate and USD/CAD daily exchange rate*. https://www.bankofcanada.ca/valet/docs

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, *81*(3), 637–654. https://doi.org/10.1086/260062

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327. https://doi.org/10.1016/0304-4076(86)90063-1

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, *50*(4), 987–1007. https://doi.org/10.2307/1912773

Engle, R. F., & Granger, C. W. J. (1987). Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, *55*(2), 251–276. https://doi.org/10.2307/1913236

Glosten, L. R., Jagannathan, R., & Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. *Journal of Finance*, *48*(5), 1779–1801. https://doi.org/10.1111/j.1540-6261.1993.tb05128.x

Merton, R. C. (1973). Theory of rational option pricing. *Bell Journal of Economics and Management Science*, *4*(1), 141–183. https://doi.org/10.2307/3003143

Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, *59*(2), 347–370. https://doi.org/10.2307/2938260

Statistics Canada. (2026). *Consumer price index, monthly, not seasonally adjusted, table 18-10-0004-01*. Government of Canada. https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=1810000401