E Exercise Solutions

R. Heller · Date: 2026-05-09 · Last changed: 2026-10-31

This appendix provides suggested solutions for all chapter exercises. The code sketches below are meant to guide you toward a working answer; they are not the only valid approach. Because each solution references data built during the corresponding chapter, code chunks are **not evaluated** here — run them inside the chapter project after completing the worked example.

Chapter 1 — What Is Scientometrics?

1. Explore a different field. Swap the search term and fetch a fresh sample. Comparing the publication trend and top journals reveals how different fields vary in volume, growth rate, and journal concentration.

```
library(openalexR)
library(tidyverse)

works_ml ← fetch_openalex(
 entity = "works",
 search = "machine learning",
 from_publication_date = "2013-01-01",
 to_publication_date = "2023-12-31",
 count_only = FALSE,
 per_page = 200
) ▷ slice_sample(n = 500)

works_ml ▷
 count(publication_year) ▷
 ggplot(aes(publication_year, n)) +
 geom_col() +
 labs(title = "Machine-learning publications per year",
      x = "Year", y = "Works")
```

Key insight: High-growth fields like machine learning show steeper annual increases and a broader spread across journals compared to a niche field like scientometrics.

2. Uncited works. Filter to zero-citation records and examine the year distribution.

```
works ▷
 filter(cited_by_count == 0) ▷
 count(publication_year) ▷
 mutate(pct = n / sum(n)) ▷
 ggplot(aes(publication_year, pct)) +
 geom_col() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "Distribution of uncited works by year",
      x = "Year", y = "Share of uncited works")
```

Key insight: Recent publication years dominate because papers need time to accumulate citations — a window effect.

3. Open Access status. Compute yearly OA proportions to reveal the trend.

```
works |>
 mutate(year = publication_year) |>
 group_by(year) |>
 summarise(oa_share = mean(is_oa, na.rm = TRUE)) |>
 ggplot(aes(year, oa_share)) +
 geom_line() +
 geom_point() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "Open-access share over time",
      x = "Year", y = "OA proportion")
```

Key insight: OA proportions have risen steadily, driven by funder mandates and the growth of gold OA journals.

4. Growth curve. Plot annual counts on a log scale; linearity implies exponential growth.

```
works |>
 filter(publication_year >= 2000, publication_year <= 2023) |>
 count(publication_year) |>
 ggplot(aes(publication_year, n)) +
 geom_point() +
 geom_smooth(method = "lm") +
 scale_y_log10() +
 labs(title = "Publication growth (log scale)",
      x = "Year", y = "Works (log)")
```

Key insight: If the points fall roughly on a straight line, growth is approximately exponential, consistent with Price's law.

Chapter 2 — Theoretical Underpinnings

1. Lotka exponent. Fit a linear model in log-log space to estimate the exponent.

```
library(tidyverse)

author_prod <- works |>
 unnest(author) |>
 count(au_id, name = "n_papers")

freq_tbl <- author_prod |>
 count(n_papers, name = "n_authors")

fit <- lm(log(n_authors) ~ log(n_papers), data = freq_tbl)
coef(fit)  # slope ≈ -2 if Lotka holds

ggplot(freq_tbl, aes(n_papers, n_authors)) +
 geom_point() +
 scale_x_log10() +
 scale_y_log10() +
 geom_smooth(method = "lm", se = FALSE) +
 labs(title = "Lotka's law fit",
      x = "Papers per author (log)", y = "Number of authors (log)")
```

Key insight: Lotka predicted an exponent near 2; empirical values typically range from 1.5 to 3.5, depending on the field and time window.

2. Bradford zones. Divide journals into three zones of roughly equal article counts.

```
journal_counts ← works ▷
 count(so, sort = TRUE) ▷
 mutate(cum = cumsum(n),
        total = sum(n),
        zone = case_when(
         cum ⩽ total / 3 ~ "Core",
         cum ⩽ 2 * total / 3 ~ "Mid",
         TRUE ~ "Periphery"
        ))

journal_counts ▷ count(zone)
```

Key insight: The core zone contains very few journals; the periphery contains many. The ratio between successive zone sizes should approximate Bradford's multiplier.

3. Your field's Zipf distribution. Extract title words and plot rank vs. frequency on log-log axes.

```
library(tidytext)

word_freq ← works ▷
 select(id, display_name) ▷
 unnest_tokens(word, display_name) ▷
 anti_join(stop_words, by = "word") ▷
 count(word, sort = TRUE) ▷
 mutate(rank = row_number())

ggplot(word_freq, aes(rank, n)) +
 geom_line() +
 scale_x_log10() +
 scale_y_log10() +
 labs(title = "Zipf's law for title words",
      x = "Rank (log)", y = "Frequency (log)")
```

Key insight: A roughly linear relationship on log-log axes confirms Zipf's law. Domain-specific terms dominate the top ranks.

4. Matthew Effect simulation. Simulate preferential attachment and compare the resulting distribution to Lotka's law.

```r
set.seed(20260509)
n_authors <- 100
papers <- rep(1, n_authors)

for (step in seq_len(50)) {
 probs <- papers / sum(papers)
 winner <- sample(n_authors, 1, prob = probs)
 papers[winner] <- papers[winner] + 1
}

tibble(papers = papers) |>
 count(papers, name = "n_authors") |>
 ggplot(aes(papers, n_authors)) +
 geom_point() +
 scale_x_log10() +
 scale_y_log10() +
 labs(title = "Simulated preferential attachment",
      x = "Papers (log)", y = "Authors (log)")
```

Key insight: Preferential attachment generates a skewed distribution resembling Lotka's law, illustrating the Matthew Effect.

Chapter 3 — Ethics and Responsible Metrics

1. Leiden Manifesto audit. This is a qualitative exercise. Evaluate a real hiring policy against the ten principles.

```r
# No code required. Suggested structure:
# For each of the 10 Leiden Manifesto principles:
#   1. State the principle
#   2. Quote the relevant policy text (if any)
#   3. Assess: satisfied / partially satisfied / violated
#   4. Explain your reasoning
# Summarise overall compliance in a paragraph.
```

Key insight: Most policies rely heavily on the h-index or JIF, violating principles 6 (field variation) and 7 (qualitative portfolio assessment).

2. Self-citation across fields. Compare self-citation rates between two different fields.

```r
library(openalexR)
library(tidyverse)

fields ← c("bibliometrics", "genomics")
results ← map(fields, \(term) {
 w ← fetch_openalex(entity = "works", search = term,
               from_publication_date = "2020-01-01",
               to_publication_date = "2023-12-31") ▷
   slice_sample(n = 200)
 w ▷ mutate(field = term)
}) ▷ list_rbind()

# Self-citation proxy: papers citing other papers by the same first author
results ▷
 group_by(field) ▷
 summarise(mean_cites = mean(cited_by_count, na.rm = TRUE),
           median_cites = median(cited_by_count, na.rm = TRUE))
```

Key insight: Self-citation baselines differ by field. Higher self-citation in some fields is structural (smaller communities) rather than manipulative.

3. DORA compliance check. A research exercise using the DORA signatory database.

```r
# No code required. Visit https://sfdora.org/signers/
# Select three major funders and compare their grant evaluation
# criteria against DORA commitments:
#   - Do they avoid using JIF in evaluation?
#   - Do they assess research outputs on their own merit?
#   - Do they consider a broad range of impact measures?
```

Key insight: Many signatories still reference journal prestige in practice, revealing a gap between stated commitments and implementation.

4. Goodhart's Law thought experiment. Propose a new indicator and analyse potential gaming strategies.

```r
# Example indicator: "Replication Impact Score"
#   = fraction of a researcher's papers that have been independently replicated
#
# Potential gaming strategies:
#   - Publish trivially replicable results
#   - Arrange replication exchanges with collaborators
#   - Focus on small, easily replicated studies rather than ambitious work
#
# The numerator (replications) and denominator (total papers) are both gameable.
# This illustrates Goodhart's Law: once a measure becomes a target,
# it ceases to be a good measure.
```

Key insight: Any ratio-based metric can be gamed on both numerator and denominator. Indicator design must anticipate behavioural responses.

Chapter 4 — Data Sources Compared

1. Coverage gap analysis. Query the same DOIs in OpenAlex and Crossref and compare.

```
library(openalexR)
library(rcrossref)
library(tidyverse)

dois ← works ▷ slice_sample(n = 20) ▷ pull(doi) ▷ na.omit()

oa_results ← map(dois, \(d) {
 tryCatch(fetch_openalex(entity = "works", doi = d), error = \(e) NULL)
}) ▷ compact()
oa_found ← length(oa_results)

cr_results ← map(dois, \(d) {
 tryCatch(cr_works(d)$data, error = \(e) NULL)
}) ▷ compact()
cr_found ← length(cr_results)

tibble(
 Source = c("OpenAlex", "Crossref"),
 Found = c(oa_found, cr_found),
 Total = length(dois),
 Coverage = Found / Total
)
```

Key insight: OpenAlex generally has high DOI coverage because it ingests Crossref data, but abstract availability may differ.

2. Disciplinary bias. Compare indexed works across contrasting fields.

```
fields ← list(
 biology = "https://openalex.org/C86803240",
 philosophy = "https://openalex.org/C138885662"
)

coverage ← imap(fields, \(concept_id, name) {
 fetch_openalex(entity = "works", concept.id = concept_id,
           count_only = TRUE) ▷
   mutate(field = name)
}) ▷ list_rbind()

coverage
```

Key insight: STEM fields are substantially better covered than humanities, reflecting the historical journal-centricity of bibliometric databases.

3. Citation count concordance. Correlate citation counts from two sources.

```
library(rcrossref)

comparison ← works ▷
 slice_sample(n = 50) ▷
 filter(!is.na(doi)) ▷
 rowwise() ▷
 mutate(cr_cites = tryCatch(
   cr_works(doi)$data$is_referenced_by_count,
   error = \(e) NA_integer_
 )) ▷
 ungroup()

cor(comparison$cited_by_count, comparison$cr_cites,
    use = "complete.obs", method = "spearman")

ggplot(comparison, aes(cited_by_count, cr_cites)) +
 geom_point() +
 geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
 labs(x = "OpenAlex citations", y = "Crossref citations",
      title = "Citation count concordance")
```

Key insight: Counts correlate strongly but are not identical; differences arise from coverage scope and update frequency.

4. Preprint tracking. Check whether bioRxiv DOIs appear in OpenAlex with published links.

```
biorxiv_dois ← c(
 "10.1101/2023.01.01.123456",
 "10.1101/2023.02.15.234567"
 # ... add 8 more real bioRxiv DOIs
)

preprint_status ← map(biorxiv_dois, \(d) {
 tryCatch({
   w ← fetch_openalex(entity = "works", doi = d)
   tibble(doi = d, found = TRUE,
          has_published = !is.null(w$ids$doi))
 }, error = \(e) tibble(doi = d, found = FALSE, has_published = FALSE))
}) ▷ list_rbind()

mean(preprint_status$found)
mean(preprint_status$has_published)
```

Key insight: Most recent bioRxiv preprints appear in OpenAlex; the proportion linked to a published version depends on recency and field.

5. Source selection exercise. This is a qualitative exercise requiring no code.

```
# Recommended answer structure:
# 1. OpenAlex: free, broad coverage, good for large-scale analysis
#    - Limitation: may under-represent regional journals
# 2. African Journals Online (AJOL): specialised coverage of
#    Sub-Saharan African journals
# 3. Combination approach: use OpenAlex for global context,
#    supplement with AJOL for regional coverage
#
# Justification should address:
#   - Coverage of nursing literature
#   - Representation of Sub-Saharan African journals
#   - Cost (OpenAlex is free; Scopus/WoS require subscriptions)
#   - Reproducibility (OpenAlex is open; others are proprietary)
```

Key insight: No single source is sufficient; combining an open global database with a regional index maximises both coverage and reproducibility.

Chapter 5 — APIs and Packages

1. Fetch works by institution. Find your institution in OpenAlex and retrieve its works.

```
library(openalexR)
library(tidyverse)

# Step 1: find the institution
my_inst ← fetch_openalex(entity = "institutions",
                    search = "University of Example")
my_inst_id ← my_inst$id[1]

# Step 2: fetch works
inst_works ← fetch_openalex(entity = "works",
                      authorships.institutions.lineage = my_inst_id,
                      from_publication_date = "2020-01-01")

nrow(inst_works)
mean(!is.na(inst_works$doi))  # DOI coverage rate
```

Key insight: DOI coverage varies by institution; institutions publishing more in regional or non-English journals tend to have lower DOI rates.

2. Compare Crossref metadata. Retrieve the same work from both APIs and compare.

```
library(rcrossref)

sample_doi ← inst_works$doi[1]

oa_meta ← fetch_openalex(entity = "works", doi = sample_doi)
cr_meta ← cr_works(sample_doi)$data

tibble(
 field = c("citations", "title_match"),
 OpenAlex = c(oa_meta$cited_by_count,
              oa_meta$display_name),
 Crossref = c(cr_meta$is_referenced_by_count,
              cr_meta$title)
)
```

Key insight: Citation counts differ because each database has a different citation graph. Crossref counts only DOI-based citations; OpenAlex resolves more.

3. Build a multi-year corpus. Fetch all works from a journal across multiple years.

```
journal_works ← fetch_openalex(
 entity = "works",
 primary_location.source.id = "S4210174107",  # example journal ID
 from_publication_date = "2015-01-01",
 to_publication_date = "2023-12-31"
)

journal_works ▷
 group_by(publication_year) ▷
 summarise(n = n(), median_cites = median(cited_by_count)) ▷
 pivot_longer(c(n, median_cites)) ▷
 ggplot(aes(publication_year, value)) +
 geom_line() +
 geom_point() +
 facet_wrap(~name, scales = "free_y") +
 labs(title = "Journal trend: publications and citations",
      x = "Year", y = NULL)
```

Key insight: Median citation counts typically decrease for recent years due to the citation time window, while publication counts may rise if the journal is growing.

Chapter 6 — Parsing Native Exports

1. Export and parse. Parse a BibTeX file from Google Scholar using `bib2df`.

```
library(bib2df)
library(tidyverse)

bib ← bib2df("scholar_export.bib")
ncol(bib)
map_int(bib, \(x) sum(!is.na(x))) ▷
 enframe("field", "populated") ▷
 arrange(desc(populated))
```

Key insight: Google Scholar BibTeX exports often lack abstracts and keywords, making them less useful for text-based analyses.

2. Format comparison. Parse the same records from CSV and RIS formats and compare fields.

```r
library(bibliometrix)

csv_data ← read_csv("scopus_export.csv")
ris_data ← convert2df("scopus_export.ris", dbsource = "scopus",
                      format = "plaintext")

cat("CSV columns:", ncol(csv_data), "\n")
cat("RIS columns:", ncol(ris_data), "\n")

# Compare overlapping fields
intersect(names(csv_data), names(ris_data))
```

Key insight: RIS files typically include structured reference data and keywords that CSV exports may omit; CSVs are easier to manipulate in R.

3. Standardization function. Write a reusable function to normalise columns.

```r
standardise_biblio ← function(df) {
 tibble(
   doi       = df$DI %||% NA_character_,
   title     = df$TI %||% df$Title %||% NA_character_,
   authors   = df$AU %||% NA_character_,
   year      = as.integer(df$PY %||% df$Year),
   journal   = df$SO %||% df$`Source title` %||% NA_character_,
   cited_by  = as.integer(df$TC %||% df$`Cited by` %||% 0L)
 )
}

# Usage:
# standardise_biblio(ris_data)
```

Key insight: A standardisation layer decouples downstream analysis from the input format, improving reproducibility.

Chapter 7 — Building Reproducible Corpora

1. DOI coverage by year. Compute the DOI coverage rate per publication year.

```r
library(tidyverse)

works ▷
 group_by(publication_year) ▷
 summarise(doi_rate = mean(!is.na(doi))) ▷
 ggplot(aes(publication_year, doi_rate)) +
 geom_line() +
 geom_point() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "DOI coverage by year", x = "Year", y = "Coverage")
```

Key insight: DOI coverage tends to improve for recent years as DOI assignment became standard practice after the mid-2000s.

2. Name variants. Detect potential author name duplicates.

```
library(stringdist)

author_names ← works ▷
 unnest(author) ▷
 distinct(au_id, au_display_name) ▷
 pull(au_display_name)

# Find pairs with low string distance
combos ← combn(author_names[1:200], 2, simplify = FALSE)
similar ← map(combos, \(pair) {
 d ← stringdist(pair[1], pair[2], method = "jw")
 if (d < 0.15) tibble(name1 = pair[1], name2 = pair[2], dist = d)
 else NULL
}) ▷ compact() ▷ list_rbind()

similar ▷ arrange(dist) ▷ head(20)
```

Key insight: Common heuristics include Jaro-Winkler distance, initial matching, and shared co-author overlap. OpenAlex author IDs help but are not always correct.

3. ROR lookup. Use the ROR API to resolve institutions.

```
library(httr2)

top_insts ← works ▷
 unnest(authorships) ▷
 unnest(institutions) ▷
 count(display_name, sort = TRUE) ▷
 head(5)

ror_lookup ← function(name) {
 resp ← request("https://api.ror.org/organizations") ▷
   req_url_query(query = name) ▷
   req_perform() ▷
   resp_body_json()
 if (length(resp$items) > 0)
   tibble(name = name, ror_id = resp$items[[1]]$id,
          ror_name = resp$items[[1]]$name)
 else
   tibble(name = name, ror_id = NA, ror_name = NA)
}

map(top_insts$display_name, ror_lookup) ▷ list_rbind()
```

Key insight: ROR provides stable, disambiguated identifiers; matching by name alone can produce false positives for common institutional names.

Chapter 8 — Working with Full Text

1. Extract and compare. Download a PDF, extract text, and compare the abstract.

```
library(pdftools)
library(tidyverse)

pdf_text_raw ← pdf_text("sample_paper.pdf")
extracted_abstract ← str_extract(
 paste(pdf_text_raw, collapse = "\n"),
 "(?i)abstract\\s*\\.?\\s*(.+?)(?=\\n\\s*(introduction|keywords|1\\.))",
)

# Compare with OpenAlex abstract
oa_abstract ← works ▷ filter(doi == "10.xxxx/example") ▷ pull(ab)

cat("Extracted:\n", str_trunc(extracted_abstract, 300), "\n")
cat("OpenAlex:\n", str_trunc(oa_abstract, 300), "\n")
```

Key insight: PDF extraction often introduces line-break artefacts and header/ footer noise. The OpenAlex abstract (from publisher metadata) is typically cleaner.

2. Section detection. Write a function to split extracted text into sections.

```
split_sections ← function(text) {
 lines ← str_split(text, "\n")[[1]]
 # Match numbered headings like "1. Introduction" or "2 Methods"
 heading_idx ← str_which(lines, "^\\s*\\d+\\.?\\s+[A-Z]")

 if (length(heading_idx) < 2) return(list(full_text = text))

 sections ← map2(heading_idx, c(heading_idx[-1] - 1, length(lines)),
                 \(start, end) {
   list(
     heading = str_trim(lines[start]),
     text = paste(lines[(start + 1):end], collapse = "\n")
   )
 })
 set_names(
   map_chr(sections, "text"),
   map_chr(sections, "heading")
 )
}

sections ← split_sections(paste(pdf_text_raw, collapse = "\n"))
names(sections)
```

Key insight: Heading detection is fragile; two-column layouts, supplementary materials, and non-standard numbering all break simple regex patterns.

3. Table extraction. Extract a table from a PDF using `tabulizer`.

```r
# tabulapdf needs Java (rJava); shown for reference, not executed.
# Install: install.packages("tabulapdf"); requires a working Java runtime.

tables <- extract_tables("sample_paper.pdf", pages = 3)
tbl <- as_tibble(tables[[1]], .name_repair = "unique")

# Typical cleaning steps:
tbl <- tbl |>
 row_to_names(row_number = 1) |>    # janitor::row_to_names
 mutate(across(where(is.character), str_trim)) |>
 filter(if_any(everything(), \(x) x != ""))
```

Key insight: Extracted tables almost always need header row promotion, whitespace trimming, and removal of empty or merged-cell artefact rows.

Chapter 9 — Descriptive Bibliometrics

1. Lotka's law. Fit Lotka's law using `bibliometrix`.

```r
library(bibliometrix)
library(tidyverse)

bib_df <- convert2df(works)  # or use the already-converted object
lotka_result <- lotka(biblioAnalysis(bib_df))

# Plot observed vs expected
plot(lotka_result)
```

Key insight: If the Kolmogorov-Smirnov test p-value is above 0.05, the observed distribution is consistent with the theoretical inverse-square law.

2. Bradford's law. Identify core, mid, and peripheral journal zones.

```r
library(openalexR)
library(bibliometrix)

ml_works <- fetch_openalex(entity = "works", search = "machine learning",
                     from_publication_date = "2020-01-01",
                     to_publication_date = "2023-12-31") |>
 slice_sample(n = 500)

bib_df <- convert2df(ml_works)
brad <- bradford(biblioAnalysis(bib_df))
brad$table |> head(20)
```

Key insight: The core zone typically contains very few journals that account for a disproportionate share of articles — Bradford's law in action.

3. Three-field exploration. Use country instead of author name on the left field.

```r
library(bibliometrix)

threeFieldsPlot(bib_df, fields = c("AU_CO", "SO", "DE"),
                n = c(10, 10, 10))
```

Key insight: Country-level Sankey diagrams reveal geographic specialisation patterns: certain countries cluster around specific journals and keywords.

4. Annual growth rate. Compute the compound annual growth rate (CAGR).

```
annual ← works ▷
 count(publication_year) ▷
 filter(publication_year ⩾ 2010, publication_year ⩽ 2023)

first_year ← annual$n[1]
last_year  ← annual$n[nrow(annual)]
n_years    ← nrow(annual) - 1

cagr ← (last_year / first_year)^(1 / n_years) - 1
cat(sprintf("CAGR: %.1f%%\n", cagr * 100))
```

Key insight: A positive CAGR indicates the field is growing; compare against the global average (~4-5% for all scientific literature) to assess relative growth.

Chapter 10 — Productivity and Impact Indicators

1. Career h-index. Compute h-index, g-index, and m-quotient for a specific author.

```
library(openalexR)
library(tidyverse)

author_works ← fetch_openalex(
 entity = "works",
 author.id = "A5023888391",  # replace with target author
 per_page = 200
)

cites ← sort(author_works$cited_by_count, decreasing = TRUE)

# h-index
h ← sum(cites ⩾ seq_along(cites))

# g-index
cum_cites ← cumsum(cites)
g_candidates ← which(cum_cites ⩾ seq_along(cites)^2)
g ← if (length(g_candidates)) max(g_candidates) else 0L

# m-quotient
career_years ← max(lubridate::year(author_works$publication_date)) -
               min(lubridate::year(author_works$publication_date)) + 1
m ← h / career_years

tibble(h_index = h, g_index = g, m_quotient = round(m, 2))
```

Key insight: The h-index rewards consistency, the g-index rewards highly cited papers, and the m-quotient adjusts for career length.

2. Self-citation sensitivity. Pseudocode for removing self-citations from the h-index.

```
# Pseudocode:
# 1. For each paper by the author, fetch its citing works
# 2. Check if any citing work shares the same author ID
# 3. Count only non-self-citations
# 4. Recompute h-index with adjusted citation counts
#
# Implementation sketch (expensive API calls):
# author_id ← "A5023888391"
# adjusted_cites ← map_int(author_works$id, \(work_id) {
#   citing ← fetch_openalex(entity = "works", cites = work_id)
#   citing ▷
#     unnest(author) ▷
#     filter(au_id ≠ author_id) ▷
#     distinct(id) ▷
#     nrow()
# })
# h_adjusted ← sum(sort(adjusted_cites, decreasing = TRUE) ⩾
#                  seq_along(adjusted_cites))
```

Key insight: Removing citations cannot increase the h-index. Its actual change requires verified author matches and citation records; no universal change is assumed.

3. Empirical resampling of sample ratios. Hold the empirical year reference fixed and resample its observed ratios.

```
set.seed(20260509)
ratios ← works_normalized$ncs[is.finite(works_normalized$ncs)]
stopifnot(length(ratios) > 1L)
ratio_boot ← replicate(1000, mean(sample(ratios, replace = TRUE)))
quantile(ratio_boot, c(0.025, 0.975))
```

Key insight: This interval describes resampling variation conditional on the chosen empirical reference. It ignores uncertainty in that reference and does not establish world-normalized impact or a population confidence interval.

4. Normalization and ranks. Compare the within-sample mean ratio and percentile selection share by year.

```
yearly_indicators ← works_with_pct ▷
 group_by(year) ▷
 summarise(
   mean_sample_ratio = mean(ncs, na.rm = TRUE),
   ranked_works = sum(!is.na(is_top10)),
   upper_percentile_share = mean(is_top10, na.rm = TRUE),
   .groups = "drop"
 )
yearly_indicators
```

Key insight: The mean ratio is one for each usable year by construction. Ties and small groups affect the percentile share. These sample summaries do not rank field-normalized impact; that requires a justified external reference.

Chapter 11 — Journal Metrics

1. Cohort sensitivity. Use the verified source table and helper from Chapter 11. This code performs a new provider request unless that exact query is already cached.

```
wider_cohort ← fetch_journal_works(journals$source_id[1], 2019, 2022)
summarise_journal_records(wider_cohort)
journal_summary ▷ filter(source_id == journals$source_id[1])
```

The summaries concern cumulative citations at retrieval. A wider publication cohort changes composition and citation exposure; the result need not move in one direction. CiteScore requires citations received within its specified four-year interval and eligible document types in both numerator and denominator. Changing a publication-date filter alone supplies neither requirement.

2. Descriptive association.

```
complete_pairs ← journal_summary ▷
  select(mean_cumulative_citations, median_recorded_references) ▷
  drop_na()
rho ← if (nrow(complete_pairs) ⩾ 3 &&
           n_distinct(complete_pairs$mean_cumulative_citations) > 1 &&
           n_distinct(complete_pairs$median_recorded_references) > 1) {
  cor(complete_pairs$mean_cumulative_citations,
      complete_pairs$median_recorded_references, method = "spearman")
} else NA_real_
rho
```

The coefficient describes the complete journal pairs only; insufficient or constant observations leave it unavailable. Its sign and magnitude depend on the retrieved data. It neither measures citation potential on the citing side nor demonstrates field normalisation or agreement with SNIP.

3. Missing-data denominator.

```
invented_records ← tibble(
  id = c("example-1", "example-2", "example-3"),
  cited_by_count = c(0, 20, NA_real_),
  referenced_works_count = c(10, NA_real_, 30)
)
summarise_journal_records(invented_records)
summarise_journal_records(mutate(invented_records, cited_by_count = NA_real_))
```

There are three records but two observed citation counts. Their sum is 20 and the observed-count mean is 10. Dividing 20 by all three records would give about 6.67 and implicitly count the missing value as zero. When all citation counts are missing, the observed denominator is zero and the sum, mean and median are reported as missing. A zero observed count is retained as a valid observation.

4. Self-citation design.

Define the publication cohort and citation-receipt window first. Obtain the citing–cited work links within that scope and identify each citing work's journal using validated source IDs. Count each citation edge once. A journal self-citation share can then use within-journal citation edges in the numerator and all observed citation edges to the cohort in the denominator, with missing source identities reported separately. Distinguish a genuinely empty eligible edge set from a failed request; a zero denominator makes the share undefined.

An average of per-paper self-citation shares weights papers equally and generally differs from the edge-level share. Neither current cumulative counts nor references recorded for the cited papers identify the journals that cited them. No fixed self-citation threshold or empirical result follows from this chapter's data.

Chapter 12 — Author-Level Analysis

1. Profile comparison. Compare two researchers on multiple indicators.

```
library(openalexR)
library(tidyverse)

ids ← c("A5023888391", "A5048491430")  # two author IDs

profiles ← map(ids, \(aid) {
 w ← fetch_openalex(entity = "works", author.id = aid)
 cites ← sort(w$cited_by_count, decreasing = TRUE)
 tibble(
   author_id = aid,
   n_papers  = nrow(w),
   h_index   = sum(cites ⩾ seq_along(cites)),
   mean_cites = mean(w$cited_by_count)
 )
}) ▷ list_rbind()

profiles
```

Key insight: An author with many papers but a modest h-index publishes frequently but with less consistent impact; high mean citations suggest a few highly cited papers.

2. ORCID validation. Compare works fetched by OpenAlex ID vs. ORCID filtering.

```
oa_by_id ← fetch_openalex(entity = "works",
                     author.id = "A5023888391")

oa_by_orcid ← fetch_openalex(entity = "works",
                        author.orcid = "0000-0001-2345-6789")

cat("By ID:", nrow(oa_by_id), "works\n")
cat("By ORCID:", nrow(oa_by_orcid), "works\n")

# Find discrepancies
only_id ← setdiff(oa_by_id$id, oa_by_orcid$id)
only_orcid ← setdiff(oa_by_orcid$id, oa_by_id$id)
cat("Only in ID set:", length(only_id), "\n")
cat("Only in ORCID set:", length(only_orcid), "\n")
```

Key insight: Discrepancies arise from incomplete ORCID claiming, name disambiguation errors, and delayed metadata updates.

3. Name variant detection. Search for a common name and assess disambiguation.

```
wang_wei ← fetch_openalex(entity = "authors",
                     search = "Wang Wei")

nrow(wang_wei)  # many entities

wang_wei ▷
 select(id, display_name, works_count, cited_by_count) ▷
 arrange(desc(works_count)) ▷
 head(20)

# Heuristics for identifying the correct author:
# - Match institutional affiliation
# - Check ORCID linkage
# - Verify co-author overlap with known collaborators
# - Inspect topic/concept alignment
```

Key insight: Common names generate hundreds of author entities; institutional affiliation and ORCID are the most reliable disambiguation features.

4. Fractional productivity. Recompute productivity using fractional counting.

```
author_works ← fetch_openalex(entity = "works",
                         author.id = "A5023888391")

fractional ← author_works ▷
 mutate(n_authors = map_int(author, nrow),
        frac_credit = 1 / n_authors) ▷
 group_by(publication_year) ▷
 summarise(full_count = n(),
           frac_count = sum(frac_credit))

ggplot(fractional, aes(publication_year)) +
 geom_line(aes(y = full_count, colour = "Full")) +
 geom_line(aes(y = frac_count, colour = "Fractional")) +
 labs(title = "Full vs. fractional productivity",
      x = "Year", y = "Papers", colour = "Counting")
```

Key insight: Fractional counting penalises hyper-authored papers. Authors who collaborate broadly see a larger gap between full and fractional counts.

5. Self-citation ratio. Estimate the fraction of self-citations for an author.

```r
target_author ← "A5023888391"

# Sample a subset of the author's works
sample_works ← author_works ▷ slice_sample(n = 10)

self_cite_frac ← map_dbl(sample_works$id, \(wid) {
 citing ← tryCatch(
   fetch_openalex(entity = "works", cites = wid),
   error = \(e) tibble()
 )
 if (nrow(citing) == 0) return(NA_real_)
 citing ▷
   unnest(author) ▷
   summarise(has_self = any(au_id == target_author)) ▷
   pull(has_self) ▷
   mean()
})

cat(sprintf("Estimated self-citation rate: %.1f%%\n",
            mean(self_cite_frac, na.rm = TRUE) * 100))
```

Key insight: Self-citation rates of 5-15% are typical; rates above 25% may signal strategic self-citation behaviour.

Chapter 13 — Institutional and Country Analysis

1. Country-level analysis. Aggregate by country using full and fractional counts.

```r
library(tidyverse)

country_counts ← works ▷
 unnest(authorships) ▷
 unnest(countries) ▷
 rename(country = countries) ▷
 mutate(n_countries = n_distinct(country), .by = id) ▷
 summarise(
   full_count = n_distinct(id),
   frac_count = sum(1 / n_countries),
   .by = country
 ) ▷
 arrange(desc(full_count)) ▷
 head(10)

country_counts ▷
 mutate(ratio = full_count / frac_count) ▷
 ggplot(aes(reorder(country, full_count), ratio)) +
 geom_col() +
 coord_flip() +
 labs(title = "Full / fractional count ratio by country",
      x = NULL, y = "Ratio")
```

Key insight: Countries with high international collaboration rates (e.g., Switzerland, Singapore) show a larger ratio, meaning full counting inflates their apparent output.

2. PP(top 10%) by institution. Compute the proportion of papers in the top 10% most cited.

```
# Compute year-specific 90th percentile thresholds
thresholds ← works ▷
 group_by(publication_year) ▷
 summarise(p90 = quantile(cited_by_count, 0.9))

inst_top10 ← works ▷
 unnest(authorships) ▷
 unnest(institutions) ▷
 left_join(thresholds, by = "publication_year") ▷
 mutate(is_top10 = cited_by_count ⩾ p90) ▷
 summarise(
   pp_top10 = mean(is_top10),
   mncs = mean(cited_by_count),
   .by = display_name
 ) ▷
 filter(!is.na(display_name)) ▷
 arrange(desc(pp_top10))

ggplot(inst_top10 ▷ head(20), aes(mncs, pp_top10)) +
 geom_point() +
 ggrepel::geom_text_repel(aes(label = display_name), size = 3) +
 labs(x = "MNCS", y = "PP(top 10%)")
```

Key insight: PP(top 10%) and MNCS generally correlate but can diverge if an institution has a few extremely highly cited outliers.

3. Collaboration intensity. Relate the number of institutional affiliations per paper to counting method effects.

```
collab_intensity ← works ▷
 unnest(authorships) ▷
 unnest(institutions) ▷
 summarise(
   n_insts = n_distinct(display_name),
   .by = id
 )

inst_collab ← works ▷
 unnest(authorships) ▷
 unnest(institutions) ▷
 left_join(collab_intensity, by = "id") ▷
 summarise(
   mean_collab = mean(n_insts),
   full_count = n_distinct(id),
   frac_count = sum(1 / n_insts),
   ratio = full_count / frac_count,
   .by = display_name
 ) ▷
 filter(!is.na(display_name))

ggplot(inst_collab, aes(mean_collab, ratio)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 labs(x = "Mean affiliations per paper",
      y = "Full / fractional ratio",
      title = "Collaboration intensity vs. counting effect")
```

Key insight: There is a positive relationship — institutions with higher collaboration intensity are more inflated by full counting.

Chapter 14 — Temporal Analysis and Citation Aging

1. Half-life comparison. Compare citation half-lives between a mathematics and a biomedical journal.

```
library(openalexR)
library(tidyverse)

fetch_half_life ← function(journal_id, label) {
 w ← fetch_openalex(entity = "works",
               primary_location.source.id = journal_id,
               from_publication_date = "2015-01-01",
               to_publication_date = "2018-12-31")
 w ▷
   mutate(age = 2024 - publication_year) ▷
   group_by(age) ▷
   summarise(mean_cites = mean(cited_by_count)) ▷
   mutate(cum_share = cumsum(mean_cites) / sum(mean_cites),
          journal = label)
}

math_hl   ← fetch_half_life("S154589771", "Mathematics")
biomed_hl ← fetch_half_life("S49861241", "Biomedicine")

bind_rows(math_hl, biomed_hl) ▷
 ggplot(aes(age, cum_share, colour = journal)) +
 geom_line() +
 geom_hline(yintercept = 0.5, linetype = "dashed") +
 labs(title = "Citation half-life comparison",
      x = "Age (years)", y = "Cumulative citation share")
```

Key insight: Biomedical journals typically have shorter half-lives (3-5 years) than mathematics journals (8-12 years) because of faster publication cycles.

2. Price index computation. Compute the fraction of references less than 5 years old.

```
# For papers with reference data available
sample_papers ← works ▷
 filter(publication_year == 2020) ▷
 slice_sample(n = 20)

price_indices ← map_dbl(sample_papers$id, \(wid) {
 refs ← tryCatch(
   fetch_openalex(entity = "works", cited_by = wid),
   error = \(e) tibble()
 )
 if (nrow(refs) == 0) return(NA_real_)
 pub_year ← sample_papers$publication_year[sample_papers$id == wid]
 mean(refs$publication_year ⩾ (pub_year - 5), na.rm = TRUE)
})

cat(sprintf("Mean Price index: %.2f\n", mean(price_indices, na.rm = TRUE)))
```

Key insight: A high Price index indicates the field relies on recent work (fast-moving); a low Price index suggests reliance on older, established literature.

3. Sleeping beauty detection. Identify delayed-recognition papers.

```
sleeping ← works ▷
 filter(publication_year ⩽ 2018,
        cited_by_count ⩾ 20) ▷
 mutate(age = 2024 - publication_year)

# Approximate: papers with low early citation rate but high total
# (exact per-year citation data requires counts_by_year from OpenAlex)
sleeping_candidates ← sleeping ▷
 filter(
   map_lgl(counts_by_year, \(cy) {
     if (is.null(cy)) return(FALSE)
     early ← cy ▷ filter(year ⩽ publication_year + 3)
     sum(early$cited_by_count, na.rm = TRUE) < 2
   })
 )

sleeping_candidates ▷
 select(display_name, publication_year, cited_by_count) ▷
 arrange(desc(cited_by_count))
```

Key insight: Sleeping beauties are rare but occur across all fields; they often represent ahead-of-their-time ideas that gained traction only after a paradigm shift.

4. Median vs. mean aging. Replot aging curves using median citations.

```
aging_comparison ← works ▷
 mutate(age = 2024 - publication_year) ▷
 group_by(age) ▷
 summarise(
   mean_cites   = mean(cited_by_count),
   median_cites = median(cited_by_count)
 ) ▷
 pivot_longer(c(mean_cites, median_cites),
              names_to = "stat", values_to = "value")

ggplot(aging_comparison, aes(age, value, colour = stat)) +
 geom_line() +
 geom_point() +
 labs(title = "Mean vs. median citation aging",
      x = "Age (years)", y = "Citations")
```

Key insight: The median curve is lower and flatter because the mean is pulled up by highly cited outliers. Median is more robust for skewed distributions.

5. Publication-cohort sensitivity. Use the Scientometrics sample fetched in Chapter 14. Its publication dates span 2013–2018; the following cohorts end in 2018 and remain within that retrieved range. These are current cumulative citation summaries for subsets of a sample.

```
cohort_end ← 2018
cohort_lengths ← c(2, 3, 5)
cohort_summaries ← map_dfr(cohort_lengths, function(length_years) {
  cohort ← works_sciento ▷
    mutate(publication_year = lubridate::year(publication_date)) ▷
    filter(publication_year ⩾ cohort_end - length_years + 1,
           publication_year ⩽ cohort_end)
  observed ← cohort$cited_by_count[!is.na(cohort$cited_by_count)]
  tibble(cohort_years = length_years, n_records = nrow(cohort),
         n_citations_observed = length(observed),
         mean_cumulative_citations = if (length(observed)) mean(observed) else
NA_real_)
})
cohort_summaries

ggplot(cohort_summaries, aes(factor(cohort_years), mean_cumulative_citations)) +
  geom_col() +
  labs(x = "Publication-cohort length (years)",
       y = "Mean cumulative citations at retrieval",
       title = "Descriptive sensitivity to publication-cohort selection")
```

Changing the publication cohort changes the included papers and their citation exposure. The mean may rise or fall; it is not guaranteed to increase with cohort length. A publication-date filter does not restrict citation-receipt dates. These calculations therefore do not estimate JIF or compare fixed post-publication citation windows. Missing citation counts remain outside the observed denominator.

Chapter 15 — Co-authorship Networks

1. Fractional counting. Weight edges by 1/(k-1) where k is the number of authors.

```
library(igraph)
library(tidyverse)

edges_frac ← works ▷
 unnest(author) ▷
 group_by(id) ▷
 mutate(k = n()) ▷
 filter(k ⩾ 2) ▷
 reframe(
   pair = combn(au_id, 2, simplify = FALSE),
   weight = 1 / (k - 1)
 ) ▷
 unnest_wider(pair, names_sep = "_") ▷
 rename(from = pair_1, to = pair_2)

g_frac ← graph_from_data_frame(edges_frac, directed = FALSE)
E(g_frac)$weight ← edges_frac$weight

degree_frac ← strength(g_frac) ▷
 sort(decreasing = TRUE) ▷
 head(10)
degree_frac
```

Key insight: Fractional counting reduces the influence of mega-authored papers. Authors primarily involved in large teams may drop in rank.

2. Temporal slicing. Build yearly networks and track structural evolution.

```
yearly_stats ← works ▷
 group_by(publication_year) ▷
 group_map(\(df, key) {
   edges ← df ▷
     unnest(author) ▷
     group_by(id) ▷
     filter(n() ⩾ 2) ▷
     reframe(pair = combn(au_id, 2, simplify = FALSE)) ▷
     unnest_wider(pair, names_sep = "_")
   if (nrow(edges) == 0) return(NULL)
   g ← graph_from_data_frame(edges, directed = FALSE)
   tibble(
     year = key$publication_year,
     n_nodes = vcount(g),
     n_components = components(g)$no,
     mean_degree = mean(degree(g))
   )
 }) ▷ list_rbind()

ggplot(yearly_stats, aes(year, mean_degree)) +
 geom_line() +
 geom_point() +
 labs(title = "Mean degree over time", x = "Year", y = "Mean degree")
```

Key insight: Mean degree typically increases over time as collaboration becomes more common; the number of components may decrease as the field becomes more connected.

3. Betweenness vs. degree. Compare centrality measures for the giant component.

```
g ← build_coauth_graph(works)
gc ← induced_subgraph(g, components(g)$membership ==
                         which.max(components(g)$csize))

centrality ← tibble(
 author = V(gc)$name,
 deg = degree(gc),
 btw = betweenness(gc, normalized = TRUE)
)

ggplot(centrality, aes(deg, btw)) +
 geom_point(alpha = 0.4) +
 labs(x = "Degree", y = "Betweenness (normalised)",
      title = "Betweenness vs. degree in co-authorship network")
```

Key insight: High-betweenness, low-degree nodes are “bridge” authors who connect otherwise separate groups — they may be interdisciplinary researchers.

4. Resolution parameter. Vary the Leiden resolution and compare community counts.

```
library(igraph)

resolutions ← c(0.5, 1.0, 2.0)

res_results ← map(resolutions, \(r) {
 comm ← cluster_leiden(gc, resolution = r,
                       objective_function = "modularity")
 tibble(resolution = r,
        n_communities = length(comm),
        modularity = modularity(comm))
})

list_rbind(res_results)
```

Key insight: Higher resolution produces more communities. The optimal value depends on the application — lower resolution reveals broad thematic clusters, higher resolution reveals tight collaborative teams.

Chapter 16 — Co-citation and Bibliographic Coupling

1. Cosine normalisation. Normalise co-citation strengths by the geometric mean of individual citation counts.

```
library(igraph)
library(tidyverse)

# Assume cocit_matrix is the raw co-citation matrix
diag_sqrt ← sqrt(diag(cocit_matrix))
cosine_matrix ← cocit_matrix / outer(diag_sqrt, diag_sqrt)
diag(cosine_matrix) ← 0

g_cosine ← graph_from_adjacency_matrix(cosine_matrix,
                                       mode = "undirected",
                                       weighted = TRUE)

# Compare top pairs
top_pairs_raw ← E(g_raw)[order(-E(g_raw)$weight)][1:10]
top_pairs_cos ← E(g_cosine)[order(-E(g_cosine)$weight)][1:10]
```

Key insight: Cosine normalisation down-weights pairs that are individually highly cited, surfacing topically related pairs that raw counts miss.

2. Temporal co-citation. Split the corpus into two periods and build separate networks.

```
periods ← list(
 early = works ▷ filter(publication_year %in% 2020:2021),
 late  = works ▷ filter(publication_year %in% 2022:2023)
)

period_graphs ← map(periods, \(df) {
 # Build co-citation network for this period's references
 refs ← df ▷ unnest(referenced_works)
 ref_pairs ← refs ▷
   group_by(id) ▷
   filter(n() ⩾ 2) ▷
   reframe(pair = combn(referenced_works, 2, simplify = FALSE)) ▷
   unnest_wider(pair, names_sep = "_")

 ref_pairs ▷
   count(pair_1, pair_2, name = "weight") ▷
   graph_from_data_frame(directed = FALSE)
})

# Find stable pairs: edges present in both periods
shared_edges ← E(period_graphs$early) %in% E(period_graphs$late)
```

Key insight: Stable co-citation pairs represent the intellectual core of the field; new pairs emerging in the later period may signal research-front shifts.

3. Bibliographic coupling by year. Track modularity and fragmentation of the intellectual structure.

```
yearly_bc ← works ▷
 group_by(publication_year) ▷
 group_map(\(df, key) {
   refs ← df ▷ unnest(referenced_works)
   pairs ← refs ▷
     group_by(referenced_works) ▷
     filter(n() ⩾ 2) ▷
     reframe(pair = combn(id, 2, simplify = FALSE)) ▷
     unnest_wider(pair, names_sep = "_") ▷
     count(pair_1, pair_2, name = "weight")
   if (nrow(pairs) == 0) return(NULL)
   g ← graph_from_data_frame(pairs, directed = FALSE)
   comm ← cluster_leiden(g, resolution = 1.0)
   tibble(year = key$publication_year,
          n_components = components(g)$no,
          modularity = modularity(comm))
 }) ▷ list_rbind()

ggplot(yearly_bc, aes(year, modularity)) +
 geom_line() +
 labs(title = "Bibliographic coupling modularity over time")
```

Key insight: Decreasing modularity suggests convergence across subfields; increasing modularity suggests specialisation and fragmentation.

4. Identifying research fronts. Label communities using top title words.

```r
library(tidytext)

bc_comm ← cluster_leiden(g_bc, resolution = 1.0)
V(g_bc)$community ← membership(bc_comm)

front_labels ← works ▷
 filter(id %in% V(g_bc)$name) ▷
 left_join(
   tibble(id = V(g_bc)$name, community = V(g_bc)$community),
   by = "id"
 ) ▷
 unnest_tokens(word, display_name) ▷
 anti_join(stop_words, by = "word") ▷
 count(community, word, sort = TRUE) ▷
 group_by(community) ▷
 slice_head(n = 3) ▷
 summarise(label = paste(word, collapse = ", "))

front_labels
```

Key insight: Community labels based on frequent title words typically correspond to recognisable subfields or research themes.

Chapter 17 — Co-word and Keyword Co-occurrence

1. Author keywords vs. concepts. Build and compare two co-occurrence networks.

```r
library(igraph)
library(tidyverse)

# Author keywords network
kw_pairs ← works ▷
 unnest(keywords) ▷
 group_by(id) ▷
 filter(n() ⩾ 2) ▷
 reframe(pair = combn(keyword, 2, simplify = FALSE)) ▷
 unnest_wider(pair, names_sep = "_") ▷
 count(pair_1, pair_2, name = "weight", sort = TRUE)

g_kw ← graph_from_data_frame(kw_pairs, directed = FALSE)

# OpenAlex concepts network
concept_pairs ← works ▷
 unnest(concepts) ▷
 group_by(id) ▷
 filter(n() ⩾ 2) ▷
 reframe(pair = combn(display_name, 2, simplify = FALSE)) ▷
 unnest_wider(pair, names_sep = "_") ▷
 count(pair_1, pair_2, name = "weight", sort = TRUE)

g_concept ← graph_from_data_frame(concept_pairs, directed = FALSE)

cat("Author KW network:", vcount(g_kw), "nodes,",
    ecount(g_kw), "edges\n")
cat("Concept network:", vcount(g_concept), "nodes,",
    ecount(g_concept), "edges\n")
```

Key insight: Author keywords are more specific but inconsistent; OpenAlex concepts are more standardised but coarser.

2. Temporal evolution. Track the top keywords by degree over time.

```
yearly_top_kw ← works ▷
 unnest(keywords) ▷
 group_by(publication_year) ▷
 group_map(\(df, key) {
   pairs ← df ▷
     group_by(id) ▷
     filter(n() ⩾ 2) ▷
     reframe(pair = combn(keyword, 2, simplify = FALSE)) ▷
     unnest_wider(pair, names_sep = "_")
   if (nrow(pairs) == 0) return(NULL)
   g ← graph_from_data_frame(pairs, directed = FALSE)
   tibble(
     year = key$publication_year,
     keyword = names(sort(degree(g), decreasing = TRUE)[1:5]),
     degree = sort(degree(g), decreasing = TRUE)[1:5]
   )
 }) ▷ list_rbind()

ggplot(yearly_top_kw, aes(year, degree, colour = keyword)) +
 geom_line() +
 labs(title = "Top keywords by degree over time")
```

Key insight: Some keywords are perennially central; others emerge or fade, signalling evolving research interests.

3. Strategic diagram. Plot clusters on a density-centrality plane.

```
comm ← cluster_leiden(g_kw, resolution = 1.0)
V(g_kw)$community ← membership(comm)

strategic ← map(unique(membership(comm)), \(c) {
 nodes ← V(g_kw)[V(g_kw)$community == c]
 subg ← induced_subgraph(g_kw, nodes)
 tibble(
   community = c,
   density = edge_density(subg),
   centrality = mean(strength(g_kw, v = nodes))
 )
}) ▷ list_rbind()

ggplot(strategic, aes(centrality, density,
                      label = community)) +
 geom_point(size = 3) +
 geom_text(nudge_y = 0.01) +
 geom_hline(yintercept = median(strategic$density),
            linetype = "dashed") +
 geom_vline(xintercept = median(strategic$centrality),
            linetype = "dashed") +
 labs(title = "Strategic diagram",
      x = "Centrality (external)", y = "Density (internal)")
```

Key insight: Upper-right quadrant clusters are core themes (high density and centrality); lower-left clusters are peripheral or emerging topics.

4. Normalisation comparison. Compare raw and association-strength-normalised networks.

```
# Association strength: w_ij / (w_i * w_j / 2m)
total_weight ← sum(E(g_kw)$weight)
node_strength ← strength(g_kw)

assoc_edges ← as_data_frame(g_kw) ▷
 mutate(
   assoc = weight / (node_strength[from] * node_strength[to] /
                     (2 * total_weight))
 )

g_assoc ← graph_from_data_frame(assoc_edges, directed = FALSE)
E(g_assoc)$weight ← assoc_edges$assoc

# Compare top-centrality keywords
top_raw ← names(sort(strength(g_kw), decreasing = TRUE))[1:10]
top_assoc ← names(sort(strength(g_assoc), decreasing = TRUE))[1:10]

tibble(raw = top_raw, normalised = top_assoc)
```

Key insight: Normalisation reduces the dominance of high-frequency generic terms, elevating more specific and meaningful keyword pairs.

Chapter 18 — Community Detection and Backbone Extraction

1. Alpha sensitivity. Run the disparity filter across multiple alpha values.

```
library(igraph)
library(tidyverse)

alphas ← c(0.01, 0.05, 0.10, 0.20)

alpha_results ← map(alphas, \(a) {
 # Disparity filter: keep edge if p_ij < alpha
 el ← as_data_frame(g) ▷
   mutate(
     s_from = strength(g)[from],
     k_from = degree(g)[from],
     p_ij = (1 - weight / s_from)^(k_from - 1)
   ) ▷
   filter(p_ij < a)

 g_bb ← graph_from_data_frame(el, directed = FALSE)
 comm ← cluster_leiden(g_bb, resolution = 1.0)
 tibble(alpha = a,
        edges = ecount(g_bb),
        communities = length(comm),
        connected = components(g_bb)$no == 1)
}) ▷ list_rbind()

alpha_results
```

Key insight: Very low alpha values remove too many edges and disconnect the network; very high values retain noise. Typical values of 0.05-0.10 balance signal and connectivity.

2. Threshold vs. disparity. Compare two backbone extraction methods using NMI.

```
library(igraph)

# Threshold filtering: keep edges above median weight
threshold ← median(E(g)$weight)
g_thresh ← subgraph.edges(g, which(E(g)$weight > threshold))
comm_thresh ← cluster_leiden(g_thresh, resolution = 1.0)

# Disparity filtering (from exercise 1)
comm_disp ← cluster_leiden(g_bb, resolution = 1.0)

# NMI comparison on shared nodes
shared ← intersect(V(g_thresh)$name, V(g_bb)$name)
m1 ← membership(comm_thresh)[shared]
m2 ← membership(comm_disp)[shared]

compare(m1, m2, method = "nmi")
```

Key insight: The disparity filter preserves locally significant edges (important for low-degree nodes) while the threshold filter favours globally strong connections. NMI values below 0.7 indicate meaningfully different community structures.

3. Hierarchical structure. Check whether communities at different resolutions are nested.

```
resolutions ← c(0.5, 1.0, 2.0)

memberships ← map(resolutions, \(r) {
 comm ← cluster_leiden(g_bb, resolution = r)
 tibble(node = V(g_bb)$name, community = membership(comm),
        resolution = r)
}) ▷ list_rbind()

# Check nesting: at resolution 2.0, are communities subsets of
# resolution 0.5 communities?
nesting ← memberships ▷
 pivot_wider(names_from = resolution, values_from = community,
             names_prefix = "res_") ▷
 count(res_0.5, res_2) ▷
 group_by(res_2) ▷
 mutate(is_nested = n_distinct(res_0.5) == 1)

mean(nesting$is_nested)  # proportion of fine communities nested
```

Key insight: Perfectly nested hierarchies are rare in real networks. Partial nesting suggests a hierarchical thematic structure with some cross- cutting topics.

4. Weighted vs. unweighted community detection. Remove weights and compare results.

```
g_unweighted ← g_bb
E(g_unweighted)$weight ← 1

comm_weighted   ← cluster_leiden(g_bb, resolution = 1.0)
comm_unweighted ← cluster_leiden(g_unweighted, resolution = 1.0)

compare(membership(comm_weighted), membership(comm_unweighted),
        method = "nmi")

cat("Weighted communities:", length(comm_weighted), "\n")
cat("Unweighted communities:", length(comm_unweighted), "\n")
```

Key insight: Removing weights often increases the number of communities because weak connections that helped merge groups are no longer distinguishable from strong ones.

Chapter 19 — Science Mapping and Overlay Maps

1. Multi-institution overlay. Overlay two institutions on the same base map.

```
library(tidyverse)
library(ggrepel)

inst_ids ← c("I123456789", "I987654321")

profiles ← map(inst_ids, \(iid) {
 w ← fetch_openalex(entity = "works",
              authorships.institutions.lineage = iid,
              from_publication_date = "2020-01-01")
 w ▷
   unnest(concepts) ▷
   count(display_name, name = "n_works") ▷
   mutate(inst = iid)
}) ▷ list_rbind()

ggplot(profiles, aes(display_name, n_works, fill = inst)) +
 geom_col(position = "dodge") +
 coord_flip() +
 labs(title = "Research profile overlay")
```

Key insight: Overlapping areas indicate shared research strengths; divergent areas reveal institutional specialisation.

2. Temporal overlay. Compare the same institution's profile across two time periods.

```r
periods ← list(
 early = c("2018-01-01", "2019-12-31"),
 late  = c("2022-01-01", "2023-12-31")
)

temporal_profiles ← imap(periods, \(dates, label) {
 fetch_openalex(entity = "works",
          authorships.institutions.lineage = "I123456789",
          from_publication_date = dates[1],
          to_publication_date   = dates[2]) ▷
   unnest(concepts) ▷
   count(display_name, name = "n_works") ▷
   mutate(period = label)
}) ▷ list_rbind()

temporal_profiles ▷
 pivot_wider(names_from = period, values_from = n_works,
             values_fill = 0) ▷
 mutate(shift = late - early) ▷
 arrange(desc(abs(shift))) ▷
 head(15)
```

Key insight: Positive shifts indicate growing research areas; negative shifts suggest declining focus or strategic reorientation.

3. Country-level overlay. Compare a small and a large country's publication profiles.

```r
countries ← c("SG", "US")  # Singapore vs. United States

country_profiles ← map(countries, \(cc) {
 fetch_openalex(entity = "works",
          authorships.countries = cc,
          from_publication_date = "2022-01-01",
          to_publication_date = "2022-12-31") ▷
   slice_sample(n = 500) ▷
   unnest(concepts) ▷
   count(display_name, name = "n_works") ▷
   mutate(share = n_works / sum(n_works),
          country = cc)
}) ▷ list_rbind()

country_profiles ▷
 pivot_wider(names_from = country, values_from = share,
             values_fill = 0) ▷
 ggplot(aes(US, SG)) +
 geom_point() +
 geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
 ggrepel::geom_text_repel(aes(label = display_name), size = 2) +
 labs(title = "Country profile: Singapore vs. US")
```

Key insight: Small countries tend to show concentrated profiles (above the diagonal in selected areas); large countries are more evenly distributed.

Chapter 20 — Network Visualization and Interoperability

1. Layout comparison. Apply five layouts and compare quality and speed.

```
library(igraph)
library(ggraph)
library(tidyverse)

layouts <- c("fr", "kk", "drl", "graphopt", "lgl")

layout_results <- map(layouts, \(algo) {
 t <- system.time({
   p <- ggraph(gc, layout = algo) +
     geom_edge_link(alpha = 0.1) +
     geom_node_point(aes(colour = factor(community)), size = 1) +
     labs(title = algo) +
     theme_void()
 })
 list(plot = p, time = t["elapsed"], layout = algo)
})

# Print timing
map(layout_results, \(x) tibble(layout = x$layout, secs = x$time)) |>
 list_rbind()
```

Key insight: Fruchterman-Reingold is a good default; Kamada-Kawai is better for small networks; DRL scales to large graphs but can produce chaotic layouts.

2. Interactive filtering. Create a filterable `visNetwork` visualisation.

```
library(visNetwork)

nodes_df <- tibble(
 id = V(gc)$name,
 label = V(gc)$name,
 group = as.character(V(gc)$community)
)

edges_df <- as_data_frame(gc) |>
 select(from, to)

visNetwork(nodes_df, edges_df) |>
 visOptions(selectedBy = "group",
            highlightNearest = TRUE) |>
 visPhysics(stabilization = FALSE)
```

Key insight: Interactive filtering lets users focus on specific communities and explore inter-community bridges.

3. Gephi round-trip. Export to GraphML for use in Gephi.

```
library(igraph)

# Export
write_graph(gc, "network.graphml", format = "graphml")

# After Gephi processing, compare:
# - Force Atlas 2 layout often reveals community separation better
# - R/ggraph offers more reproducible, scriptable figures
# - Gephi is better for interactive exploration and publication
#   quality figures via SVG/PDF export
```

Key insight: Gephi's Force Atlas 2 often produces cleaner community separation; R workflows are more reproducible and automatable.

4. Colour-blind check. Test colour distinguishability with increasing community counts.

```
library(ggraph)
library(viridis)

n_communities ← c(3, 5, 10)

plots ← map(n_communities, \(k) {
 comm ← cluster_leiden(gc, resolution = k / 5)
 V(gc)$comm_k ← membership(comm)

 ggraph(gc, layout = "fr") +
   geom_edge_link(alpha = 0.05) +
   geom_node_point(aes(colour = factor(comm_k)), size = 2) +
   scale_colour_viridis_d(option = "D") +
   labs(title = paste(k, "communities")) +
   theme_void() +
   theme(legend.position = "none")
})

# Use colorBlindness package or online simulator to test
# library(colorBlindness)
# cvdPlot(plots[[3]])
```

Key insight: Viridis palettes remain distinguishable up to about 8 categories; beyond that, consider shape or faceting as supplementary channels.

Chapter 21 — Text Mining Bibliographic Corpora

1. Bigram analysis. Build a DFM from bigrams.

```
library(quanteda)
library(tidyverse)

corp ← corpus(works, text_field = "ab")
toks ← tokens(corp, remove_punct = TRUE, remove_numbers = TRUE) ▷
 tokens_tolower() ▷
 tokens_remove(stopwords("en")) ▷
 tokens_ngrams(n = 2)

dfm_bi ← dfm(toks) ▷ dfm_trim(min_termfreq = 5)

topfeatures(dfm_bi, 20)
```

Key insight: Bigrams capture meaningful phrases like "machine_learning" or "citation_analysis" that single words miss.

2. Lexical diversity. Compute the type-token ratio (TTR) per year.

```r
library(quanteda)
library(quanteda.textstats)

ttr_by_year ← works ▷
 filter(!is.na(ab)) ▷
 group_by(publication_year) ▷
 summarise(
   all_text = paste(ab, collapse = " "),
   .groups = "drop"
 ) ▷
 mutate(
   n_types  = map_int(all_text, \(t) {
     toks ← tokens(t, remove_punct = TRUE) ▷ tokens_tolower()
     ntype(dfm(toks))
   }),
   n_tokens = map_int(all_text, \(t) {
     toks ← tokens(t, remove_punct = TRUE)
     ntoken(toks)
   }),
   ttr = n_types / n_tokens
 )

ggplot(ttr_by_year, aes(publication_year, ttr)) +
 geom_line() +
 labs(title = "Lexical diversity over time",
      x = "Year", y = "Type-token ratio")
```

Key insight: Declining TTR may indicate vocabulary standardisation; increasing TTR suggests diversifying terminology.

3. Keyness analysis. Identify terms that distinguish a specific year.

```r
library(quanteda)
library(quanteda.textstats)

corp ← corpus(works ▷ filter(!is.na(ab)),
              text_field = "ab",
              docid_field = "id")
docvars(corp, "year") ← works$publication_year[!is.na(works$ab)]

dfm_all ← dfm(tokens(corp, remove_punct = TRUE) ▷
              tokens_tolower() ▷
              tokens_remove(stopwords("en")))

# Compare 2023 vs. all other years
dfm_grouped ← dfm_group(dfm_all, groups = ifelse(
 docvars(dfm_all, "year") == 2023, "2023", "other"
))

keyness ← textstat_keyness(dfm_grouped, target = "2023")
textplot_keyness(keyness, n = 20)
```

Key insight: Keyness analysis surfaces terms unique to a period. For 2023, terms related to LLMs and AI may dominate.

4. Preprocessing sensitivity. Compare analyses with and without stemming.

```
toks_nostem ← tokens(corp, remove_punct = TRUE) ▷
 tokens_tolower() ▷
 tokens_remove(stopwords("en"))

toks_stem ← tokens_wordstem(toks_nostem)

top_nostem ← topfeatures(dfm(toks_nostem), 20)
top_stem   ← topfeatures(dfm(toks_stem), 20)

tibble(
 rank = 1:20,
 unstemmed = names(top_nostem),
 stemmed = names(top_stem)
)
```

Key insight: Stemming merges variants (e.g., “citations” and “citation”) but can produce unintelligible stems. For keyword-based analyses, unstemmed terms are often more interpretable.

Chapter 22 — Topic Modeling

1. K selection. Fit LDA models with multiple values of K and compare coherence.

```
library(topicmodels)
library(tidyverse)
set.seed(20260509)

ks ← c(5, 8, 12, 15)

models ← map(ks, \(k) {
 LDA(dtm, k = k, method = "Gibbs",
     control = list(seed = 20260509, iter = 1000))
})

# Compute coherence (using top-10 terms per topic)
coherence_scores ← map_dbl(models, \(mod) {
 terms ← terms(mod, 10)
 # Simple PMI-based coherence approximation
 mean(map_dbl(seq_len(ncol(terms)), \(t) {
   top_terms ← terms[, t]
   pairs ← combn(top_terms, 2, simplify = FALSE)
   mean(map_dbl(pairs, \(p) {
     log((slam::col_sums(dtm[, p[1]] > 0 & dtm[, p[2]] > 0) + 1) /
         slam::col_sums(dtm[, p[1]] > 0))
   }))
 }))
})

tibble(k = ks, coherence = coherence_scores) ▷
 ggplot(aes(k, coherence)) +
 geom_line() +
 geom_point() +
 labs(title = "Topic coherence by K", x = "K", y = "Coherence")
```

Key insight: The best K balances coherence (higher is better) and interpretability. There is rarely a single “correct” K.

2. STM with journal covariate. Fit a Structural Topic Model with journal as a prevalence covariate.

```
library(stm)
set.seed(20260509)

# Prepare data from two journals
two_journals ← works ▷
 filter(so %in% c("Scientometrics", "PLOS ONE"), !is.na(ab))

processed ← textProcessor(two_journals$ab,
                          metadata = two_journals)
prepped ← prepDocuments(processed$documents,
                        processed$vocab,
                        processed$meta)

stm_fit ← stm(prepped$documents, prepped$vocab,
              K = 10,
              prevalence = ~ so,
              data = prepped$meta,
              seed = 20260509)

effect ← estimateEffect(1:10 ~ so, stm_fit, metadata = prepped$meta)
plot(effect, covariate = "so", topics = 1:10)
```

Key insight: Topics with large journal effects represent disciplinary specialisation; shared topics indicate cross-cutting themes.

3. Topic labelling. Read top documents per topic to assign labels.

```
library(topicmodels)
library(tidyverse)

best_model ← models[[which.max(coherence_scores)]]
gamma ← posterior(best_model)$topics

# For each topic, find the top-5 documents
topic_labels ← map(seq_len(ncol(gamma)), \(t) {
 top_docs ← order(gamma[, t], decreasing = TRUE)[1:5]
 titles ← works$display_name[top_docs]
 tibble(topic = t,
        top_terms = paste(terms(best_model, 5)[, t], collapse = ", "),
        sample_titles = paste(titles, collapse = " | "))
}) ▷ list_rbind()

topic_labels
# Now assign human-readable labels based on inspection
```

Key insight: Automated topic labels from top terms are a starting point; reading representative documents is essential for meaningful labels.

4. LDA vs. STM comparison. Compare topics from both models.

```
# Extract top terms for each model
lda_terms ← terms(best_model, 10)
stm_terms ← labelTopics(stm_fit, n = 10)$prob

# Compute Jaccard similarity between matched topics
jaccard ← function(a, b) {
 length(intersect(a, b)) / length(union(a, b))
}

# Best-match each LDA topic to an STM topic
similarities ← map_dbl(seq_len(ncol(lda_terms)), \(i) {
 max(map_dbl(seq_len(nrow(stm_terms)), \(j) {
   jaccard(lda_terms[, i], stm_terms[j, ])
 }))
})

mean(similarities)
```

Key insight: STM topics may differ because the year covariate allows topic prevalence to shift over time, producing more temporally coherent topics.

Chapter 23 — Keyword Extraction

1. Precision and recall. Evaluate RAKE keywords against author keywords.

```
library(udpipe)
library(tidyverse)

sample_papers ← works ▷
 filter(!is.na(ab), map_lgl(keywords, \(k) length(k) > 0)) ▷
 slice_sample(n = 20)

eval_results ← map(seq_len(nrow(sample_papers)), \(i) {
 paper ← sample_papers[i, ]
 rake_kw ← keywords_rake(
   data.frame(doc_id = paper$id,
              sentence = paper$ab,
              stringsAsFactors = FALSE),
   term = "sentence", group = "doc_id"
 ) ▷ head(10) ▷ pull(keyword) ▷ tolower()

 author_kw ← tolower(unlist(paper$keywords))

 tibble(
   precision = mean(rake_kw %in% author_kw),
   recall = mean(author_kw %in% rake_kw)
 )
}) ▷ list_rbind()

eval_results ▷
 summarise(across(everything(), list(mean = mean, sd = sd)))
```

Key insight: RAKE precision is typically low (0.1-0.3) because algorithms and authors conceptualise keywords differently. Recall is also limited because author keywords often include broad terms not in the abstract.

2. Bigram keywords. Score bigrams by TF-IDF.

```
library(tidytext)
library(tidyverse)

bigram_tfidf ← works ▷
 filter(!is.na(ab)) ▷
 unnest_tokens(bigram, ab, token = "ngrams", n = 2) ▷
 separate(bigram, c("word1", "word2"), sep = " ") ▷
 filter(!word1 %in% stop_words$word,
        !word2 %in% stop_words$word) ▷
 unite(bigram, word1, word2, sep = " ") ▷
 count(id, bigram) ▷
 bind_tf_idf(bigram, id, n) ▷
 group_by(bigram) ▷
 summarise(mean_tfidf = mean(tf_idf)) ▷
 arrange(desc(mean_tfidf))

bigram_tfidf ▷ head(20)
```

Key insight: Multi-word terms like “citation analysis” and “research evaluation” are invisible to unigram methods but highly informative.

3. Keyword enrichment. Use extracted keywords to build a co-word network for keyword-poor papers.

```
library(igraph)
library(udpipe)

# Papers without author keywords
no_kw_papers ← works ▷
 filter(map_lgl(keywords, \(k) length(k) == 0), !is.na(ab))

# Extract keywords via RAKE
extracted ← map(seq_len(nrow(no_kw_papers)), \(i) {
 kw ← keywords_rake(
   data.frame(doc_id = no_kw_papers$id[i],
              sentence = no_kw_papers$ab[i]),
   term = "sentence", group = "doc_id"
 ) ▷ head(5) ▷ pull(keyword)
 tibble(id = no_kw_papers$id[i], keyword = kw)
}) ▷ list_rbind()

# Build co-word network
pairs ← extracted ▷
 group_by(id) ▷
 filter(n() ⩾ 2) ▷
 reframe(pair = combn(keyword, 2, simplify = FALSE)) ▷
 unnest_wider(pair, names_sep = "_") ▷
 count(pair_1, pair_2, sort = TRUE)

g_enriched ← graph_from_data_frame(pairs, directed = FALSE)
```

Key insight: Extracted keywords produce a topical structure comparable to author keywords, enabling co-word analysis even for papers without manual keyword assignments.

Chapter 24 — Embeddings for Scientometrics

1. Weighted averaging. Use TF-IDF weights for document embedding computation.

```
library(word2vec)
library(tidytext)
library(tidyverse)
library(umap)

# Compute TF-IDF weights
tfidf ← works ▷
 filter(!is.na(ab)) ▷
 unnest_tokens(word, ab) ▷
 anti_join(stop_words, by = "word") ▷
 count(id, word) ▷
 bind_tf_idf(word, id, n)

# Get word vectors
wv ← predict(w2v_model, newdata = unique(tfidf$word), type = "embedding")

# Weighted average per document
doc_embeddings ← tfidf ▷
 inner_join(
   as_tibble(wv, rownames = "word"),
   by = "word"
 ) ▷
 group_by(id) ▷
 summarise(across(starts_with("V"), \(x) weighted.mean(x, tf_idf)))

# UMAP
umap_result ← umap(as.matrix(doc_embeddings ▷ select(-id)))
```

Key insight: TF-IDF weighting down-weights common words, often producing tighter, more meaningful clusters in the UMAP projection.

2. K-means clustering. Apply k-means to embeddings and compare with LDA topics.

```
set.seed(20260509)

embedding_matrix ← as.matrix(doc_embeddings ▷ select(-id))

km_results ← map(c(5, 8, 12), \(k) {
 km ← kmeans(embedding_matrix, centers = k, nstart = 25)
 tibble(k = k,
        cluster = km$cluster,
        id = doc_embeddings$id)
}) ▷ list_rbind()

# Compare k=8 clusters with LDA topics (K=8)
# Assuming lda_topics is a vector of topic assignments
km8 ← km_results ▷ filter(k == 8) ▷ pull(cluster)
compare_nmi ← igraph::compare(km8, lda_topics, method = "nmi")
cat(sprintf("NMI between k-means and LDA: %.3f\n", compare_nmi))
```

Key insight: Moderate NMI (0.3-0.6) is typical; the two methods capture overlapping but not identical structure because they model different aspects of the data.

3. Pre-trained embeddings. Use SPECTER embeddings via reticulate.

```
library(reticulate)

# Requires Python with transformers installed
transformers ← import("transformers")
torch ← import("torch")

tokenizer ← transformers$AutoTokenizer$from_pretrained(
 "allenai/specter2"
)
model ← transformers$AutoModel$from_pretrained(
 "allenai/specter2"
)

abstracts ← works$ab[!is.na(works$ab)][1:100]
inputs ← tokenizer(abstracts, padding = TRUE, truncation = TRUE,
                   return_tensors = "pt", max_length = 512L)
with(torch$no_grad(), {
 outputs ← model(inputs)
})
specter_emb ← outputs$last_hidden_state$mean(dim = 1L)$numpy()
```

Key insight: Pre-trained scientific embeddings like SPECTER capture domain semantics better than generic word2vec, especially for similarity search.

4. Analogy test. Test analogical reasoning in the word2vec model.

```
library(word2vec)

# Test: "journal" - "article" + "book" ≈ ?
analogy ← predict(w2v_model,
                  newdata = c("journal", "article", "book"),
                  type = "nearest", top_n = 5)

# Alternative approach using vector arithmetic
wv ← as.matrix(w2v_model)
if (all(c("journal", "article", "book") %in% rownames(wv))) {
 target ← wv["journal", ] - wv["article", ] + wv["book", ]
 sims ← word2vec_similarity(
   matrix(target, nrow = 1), wv
 )
 sort(sims[1, ], decreasing = TRUE)[1:5]
}
```

Key insight: Domain-specific models may capture field-specific analogies (e.g., “h-index” - “author” + “journal” near “impact factor”) but fail on generic analogies, reflecting the training corpus.

Chapter 25 — Detecting Emerging Topics

1. Relative burst detection. Use proportional frequencies instead of absolute counts.

```
library(tidyverse)

# Compute keyword proportions by year
kw_yearly ← works ▷
 unnest(keywords) ▷
 count(publication_year, keyword) ▷
 group_by(publication_year) ▷
 mutate(proportion = n / sum(n)) ▷
 ungroup()

# Apply Kleinberg burst detection on proportions
bursts_relative ← kw_yearly ▷
 group_by(keyword) ▷
 filter(n() ⩾ 3) ▷  # need multiple years
 summarise(
   burst = list(kleinberg_bursts(proportion, publication_year)),
   .groups = "drop"
 ) ▷
 unnest(burst) ▷
 arrange(desc(strength))

bursts_relative ▷ head(20)
```

Key insight: Relative frequencies reduce bias toward high-volume years; some terms that burst in absolute counts may be merely riding overall growth.

2. Emergence by journal. Compare emerging keywords across two journals.

```
journals ← c("Scientometrics", "PLOS ONE")

journal_bursts ← map(journals, \(j) {
 kw_j ← works ▷
   filter(so == j) ▷
   unnest(keywords) ▷
   count(publication_year, keyword) ▷
   group_by(keyword) ▷
   filter(n() ⩾ 3) ▷
   summarise(
     burst = list(kleinberg_bursts(n, publication_year))
   ) ▷
   unnest(burst) ▷
   mutate(journal = j)
}) ▷ list_rbind()

# Compare top bursting terms
journal_bursts ▷
 group_by(journal) ▷
 arrange(desc(strength)) ▷
 slice_head(n = 10) ▷
 select(journal, keyword, strength)
```

Key insight: Different journals serve different research communities; emerging terms may overlap only for broad methodological advances.

3. Novelty scoring. Compute document novelty using cosine distance to prior-year centroids.

```
library(quanteda)
library(tidyverse)

dfm_yearly ← dfm(tokens(corpus(works, text_field = "ab"),
                        remove_punct = TRUE) ▷
                  tokens_tolower() ▷
                  tokens_remove(stopwords("en"))) ▷
 dfm_tfidf()

years ← sort(unique(works$publication_year))

novelty ← map(years[-1], \(yr) {
 prior ← dfm_subset(dfm_yearly, publication_year < yr)
 current ← dfm_subset(dfm_yearly, publication_year == yr)
 if (nrow(prior) == 0 || nrow(current) == 0) return(NULL)

 centroid ← colMeans(as.matrix(prior))
 current_mat ← as.matrix(current)

 cos_dist ← 1 - (current_mat %*% centroid) /
   (sqrt(rowSums(current_mat^2)) * sqrt(sum(centroid^2)))

 tibble(year = yr, novelty = as.numeric(cos_dist))
}) ▷ list_rbind()

novelty ▷
 group_by(year) ▷
 summarise(mean_novelty = mean(novelty, na.rm = TRUE)) ▷
 ggplot(aes(year, mean_novelty)) +
 geom_line() +
 labs(title = "Document novelty over time",
      x = "Year", y = "Mean cosine distance to prior centroid")
```

Key insight: Increasing novelty scores suggest the field is exploring new territory; stable scores indicate incremental progress within established paradigms.

4. Validation. Manually validate burst-detected emerging terms.

```
# Select 5 terms identified as "emerging"
emerging_terms ← bursts_relative ▷
 arrange(desc(strength)) ▷
 head(5) ▷
 pull(keyword)

# For each term, search for corroborating evidence:
# - Is there a review article about this topic?
# - Was it a conference theme in recent years?
# - Does Google Trends show a rising interest?

# Create a validation table
validation ← tibble(
 term = emerging_terms,
 review_found = c(TRUE, TRUE, FALSE, TRUE, FALSE),
 conference_theme = c(TRUE, FALSE, FALSE, TRUE, TRUE),
 google_trend_rising = c(TRUE, TRUE, TRUE, FALSE, TRUE),
 validated = review_found | conference_theme
)

cat(sprintf("False positive rate: %.0f%%\n",
            (1 - mean(validation$validated)) * 100))
```

Key insight: Burst detection has moderate precision; triangulating with qualitative sources reduces false positives and increases confidence.

Chapter 26 — Altmetrics

1. OA citation advantage. Compare citations for OA vs. closed-access papers across journals.

```
library(openalexR)
library(tidyverse)

journal_ids ← c("S4210174107", "S137773608")

oa_advantage ← map(journal_ids, \(jid) {
 w ← fetch_openalex(entity = "works",
                primary_location.source.id = jid,
                from_publication_date = "2019-01-01",
                to_publication_date = "2022-12-31")
 w ▷
   group_by(is_oa) ▷
   summarise(
     n = n(),
     mean_cites = mean(cited_by_count),
     median_cites = median(cited_by_count)
   ) ▷
   mutate(journal = jid)
}) ▷ list_rbind()

oa_advantage
```

Key insight: The OA citation advantage exists in most fields but varies in magnitude. Self-selection (better papers go OA) may partially explain the effect.

2. Temporal attention. Track citation accumulation over time for early-attention papers.

```
recent_works ← works ▷
 filter(publication_year == 2022, !is.null(counts_by_year))

attention_trajectory ← recent_works ▷
 unnest(counts_by_year) ▷
 group_by(id) ▷
 arrange(year) ▷
 mutate(cum_cites = cumsum(cited_by_count)) ▷
 ungroup()

# Classify: high vs. low early attention (first year citations)
early_cites ← attention_trajectory ▷
 filter(year == 2022) ▷
 mutate(early_attention = ifelse(cited_by_count ⩾ 5,
                                 "High", "Low"))

attention_trajectory ▷
 left_join(early_cites ▷ select(id, early_attention), by = "id") ▷
 group_by(early_attention, year) ▷
 summarise(mean_cum = mean(cum_cites, na.rm = TRUE)) ▷
 ggplot(aes(year, mean_cum, colour = early_attention)) +
 geom_line() +
 labs(title = "Citation accumulation by early attention")
```

Key insight: Papers with high early attention tend to sustain their advantage, consistent with cumulative advantage (Matthew Effect) in citations.

3. Cross-field comparison. Compare citation distributions and reference list lengths across fields.

```
biomed ← fetch_openalex(entity = "works",
                   primary_location.source.id = "S49861241",
                   from_publication_date = "2020-01-01") ▷
 mutate(field = "Biomedicine") ▷
 slice_sample(n = 200)

math ← fetch_openalex(entity = "works",
                 primary_location.source.id = "S154589771",
                 from_publication_date = "2020-01-01") ▷
 mutate(field = "Mathematics") ▷
 slice_sample(n = 200)

comparison ← bind_rows(biomed, math)

comparison ▷
 group_by(field) ▷
 summarise(
   mean_cites = mean(cited_by_count),
   median_cites = median(cited_by_count),
   mean_refs = mean(referenced_works_count, na.rm = TRUE)
 )

ggplot(comparison, aes(cited_by_count, fill = field)) +
 geom_histogram(position = "dodge", bins = 30) +
 scale_x_log10() +
 labs(title = "Citation distributions by field")
```

Key insight: Biomedical papers have higher citation counts and longer reference lists than mathematics papers. This is why raw citation counts cannot be compared across fields.

Chapter 27 — Open Science Indicators

1. Cross-journal OA comparison. Compare OA rates across three journals in different disciplines.

```
library(openalexR)
library(tidyverse)

journals ← list(
 biology = "S49861241",
 physics = "S137773608",
 sociology = "S64468286"
)

oa_rates ← imap(journals, \(jid, field) {
 w ← fetch_openalex(entity = "works",
                primary_location.source.id = jid,
                from_publication_date = "2020-01-01",
                to_publication_date = "2023-12-31")
 w ▷
   summarise(oa_rate = mean(is_oa, na.rm = TRUE),
             gold_oa = mean(oa_status == "gold", na.rm = TRUE)) ▷
   mutate(field = field)
}) ▷ list_rbind()

oa_rates
```

Key insight: Biology tends to have the highest gold OA rate (driven by PLOS, eLife, etc.), while sociology and mathematics lag behind.

2. Temporal OA growth. Track the year-over-year OA growth rate.

```
oa_growth ← works ▷
 filter(is_oa) ▷
 count(publication_year, name = "n_oa") ▷
 mutate(yoy_growth = (n_oa / lag(n_oa)) - 1)

ggplot(oa_growth, aes(publication_year, yoy_growth)) +
 geom_col() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "Year-over-year OA growth",
      x = "Year", y = "Growth rate")
```

Key insight: OA growth accelerated around 2020, likely driven by COVID-19 mandates and Plan S implementation.

3. OA and collaboration. Test whether author count predicts OA status.

```
works ▷
 mutate(n_authors = map_int(author, nrow)) ▷
 group_by(n_authors = pmin(n_authors, 10)) ▷
 summarise(oa_rate = mean(is_oa, na.rm = TRUE), n = n()) ▷
 ggplot(aes(n_authors, oa_rate)) +
 geom_col() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "OA rate by number of authors",
      x = "Authors (capped at 10)", y = "OA rate")
```

Key insight: Larger teams may have more funding to cover APCs. International collaborations also increase OA probability because at least one funder may mandate it.

Chapter 28 — Gender and Equity Analysis

1. Denominator comparison. Use the explicitly synthetic `lookup_records` from Chapter 28.

```
all_records ← lookup_records ▷
  count(lookup_category) ▷
  mutate(share = n / sum(n))
matched_only ← lookup_records ▷
  filter(lookup_category ≠ "Unmatched") ▷
  count(lookup_category) ▷
  mutate(share = n / sum(n))
all_records
matched_only
```

The fixture has 12 records: four A, two B and six Unmatched. Among all records, A is one third and B one sixth; among matched records, A is two thirds and B one third. None of these arbitrary category shares measures gender representation.

2. Position audit.

```
position_data ▷ count(author_position)
position_summary ▷
  group_by(author_position) ▷
  summarise(proportion_sum = sum(pct), records = sum(n), .groups = "drop")
```

The invented lists produce four first, four last, three middle and one single-author record. Each position's proportions sum to one because unmatched records remain included. Recorded position does not establish a contribution role.

3. Study design.

Specify the scientific question and population before defining an analysis. A real study would need a defensible demographic variable and its provenance, a consent/data-governance plan, categories appropriate to that question, missing-data and validation procedures, and evidence about authorship conventions. The fixture supplies none of those demographic observations, nor real identity linkage, publication years or citation outcomes. Changing a name dictionary cannot recover them. No citation-gap or gender-equity conclusion follows from this exercise.

Chapter 29 — Retraction Analysis

1. Retraction rate by field. Compare retraction rates across fields, normalised by corpus size.

```
library(openalexR)
library(tidyverse)

fields ← c("biomedicine", "computer science")

retraction_rates ← map(fields, \(f) {
 all_works ← fetch_openalex(entity = "works", search = f,
                       from_publication_date = "2015-01-01",
                       count_only = TRUE)
 retracted ← fetch_openalex(entity = "works", search = f,
                       type = "retraction",
                       from_publication_date = "2015-01-01",
                       count_only = TRUE)
 tibble(
   field = f,
   total = all_works$count,
   retracted = retracted$count,
   rate_per_1000 = retracted$count / all_works$count * 1000
 )
}) ▷ list_rbind()

retraction_rates
```

Key insight: Retraction rates are typically higher in biomedical fields (1-2 per 1,000) than in computer science, partly reflecting different scrutiny levels and reproducibility challenges.

2. Citation trajectory. Examine when a retracted paper's citations were received.

```
retracted_paper ← fetch_openalex(entity = "works",
                            doi = "10.xxxx/retracted-example")

if (!is.null(retracted_paper$counts_by_year)) {
 retracted_paper ▷
   unnest(counts_by_year) ▷
   ggplot(aes(year, cited_by_count)) +
   geom_col() +
   geom_vline(xintercept = 2020, linetype = "dashed",
              colour = "red") +  # retraction year
   labs(title = "Citation trajectory of a retracted paper",
        x = "Year", y = "Citations received")
}
```

Key insight: Many retracted papers continue to be cited after retraction, indicating that the retraction notice does not always reach citing authors.

3. Retraction-citation correlation. Test whether pre-retraction citation count predicts post-retraction citations.

```
# Assuming a set of retracted papers with known retraction years
retracted_set ← retracted_works ▷
 unnest(counts_by_year) ▷
 mutate(period = ifelse(year < retraction_year, "pre", "post")) ▷
 group_by(id, period) ▷
 summarise(cites = sum(cited_by_count), .groups = "drop") ▷
 pivot_wider(names_from = period, values_from = cites,
             values_fill = 0)

cor(retracted_set$pre, retracted_set$post, method = "spearman")

ggplot(retracted_set, aes(pre, post)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 labs(x = "Pre-retraction citations",
      y = "Post-retraction citations",
      title = "Pre- vs. post-retraction citation counts")
```

Key insight: A positive correlation suggests that highly visible papers continue to be cited after retraction — the "retraction paradox."

Chapter 30 — Funding Analysis

1. Funder co-occurrence. Identify frequently co-occurring funder pairs.

```
library(tidyverse)

funder_pairs ← inner_join(
  select(funder_data, work_id, funder_a = funder_id),
  select(funder_data, work_id, funder_b = funder_id),
  by = "work_id", relationship = "many-to-many"
) ▷
  filter(funder_a < funder_b) ▷
  distinct(work_id, funder_a, funder_b) ▷
  count(funder_a, funder_b, sort = TRUE)

funder_pairs ▷ head(10)
```

This uses the cleaned, observed funder IDs from Chapter 30 and counts each pair once per sampled work. An empty result means no co-funding pair was observed in this sample; it is not evidence that the funders never collaborate.

2. Funder portfolios. Prepare a bounded comparison for two funders observed in the sample.

```r
library(tidyverse)

funder_ids <- head(funder_counts$funder_id, 2)
if (length(funder_ids) < 2L) {
  cat("A two-funder comparison is unavailable in this sample.\n")
} else {
  portfolio_sample <- funder_data |>
    filter(funder_id %in% funder_ids) |>
    select(work_id, funder_id) |>
    distinct() |>
    left_join(select(works, id, display_name, abstract),
              by = c("work_id" = "id"))
  portfolio_sample |> count(funder_id, name = "n_sampled_works")
}
```

Apply the text-processing workflow from Chapter 20 to the available titles and abstracts, reporting text coverage per funder. This corpus contains only the sampled journal's papers, so it is not either funder's full portfolio. Completing the exercise for whole portfolios requires a documented retrieval for validated funder IDs, the same publication window, and verified funding metadata coverage. Do not substitute invented identifiers or infer strategic priorities from a small journal sample.

3. Temporal trends. Track funding acknowledgement coverage over time.

```r
funding_trend <- funding_works |>
 mutate(publication_year = lubridate::year(publication_date)) |>
 group_by(publication_year) |>
 summarise(n_sampled = n(),
           n_with_funder = sum(has_funder_metadata),
           funding_rate = mean(has_funder_metadata), .groups = "drop")

ggplot(funding_trend, aes(publication_year, funding_rate)) +
 geom_line() +
 geom_point() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "Observed funder metadata by publication year in the sample",
      x = "Publication year", y = "Share with observed funder IDs")
```

The rate describes metadata observed at retrieval, grouped by publication year. It does not measure the prevalence of financial support or establish how the database's coverage changed historically. Keep the annual sample denominators and report sparse groups and missing publication years.

Chapter 31 — Patent-Paper Linkages

1. Scholarly-citation comparison. Use two journals represented in the Chapter 31 sample. Keep the common sample threshold when comparing them, and report the publication years and observed sample sizes. These are descriptive sample summaries, not journal rankings.

```r
library(tidyverse)

selected_journals ← works_citations ▷
  filter(!is.na(source_display_name)) ▷
  count(source_display_name, sort = TRUE) ▷
  slice_head(n = 2) ▷
  pull(source_display_name)

stopifnot(length(selected_journals) == 2)
common_p90 ← quantile(works_citations$cited_by_count, 0.9, na.rm = TRUE)

works_citations ▷
  filter(source_display_name %in% selected_journals) ▷
  group_by(source_display_name, year) ▷
  summarise(
    n_sampled = n(),
    n_citation_observed = sum(!is.na(cited_by_count)),
    median_cites = median(cited_by_count, na.rm = TRUE),
    mean_cites = mean(cited_by_count, na.rm = TRUE),
    proportion_at_or_above_common_p90 =
      mean(cited_by_count ⩾ common_p90, na.rm = TRUE),
    .groups = "drop"
  )
```

Record the original retrieval date from the data provenance; a cached result's render date is not its retrieval date. Document field classifications separately before attributing differences to fields: the worked sample does not supply a field classification for this comparison. The common threshold is the sample's 90th percentile of current cumulative scholarly citations; ties may put more than 10% of works at or above it. A separate 90th percentile for each journal would put approximately 10% of each journal above its own threshold by construction. None of these quantities measures patent citations or predicts technological impact.

2. Linkage data specification. The linkage table needs actual source records, with at least these fields:

Field	Purpose
`paper_id`	Stable identifier for the matched scholarly work, such as a DOI or OpenAlex work ID.
`patent_id`	Identifier for the citing patent document, including the patent office.
`patent_family_id`	Documented family identifier for deduplicating related patent documents.
`paper_publication_date`	Starting date for a clearly defined paper-to-patent interval.
`citation_date`	Date associated with the citation in the source, with its meaning documented; if only patent publication date is available, identify it as that proxy.
`source` and `retrieved_at`	Patent-reference source, version or snapshot, and retrieval date.
`raw_reference`	Original reference text or source record identifier for verification.

Field	Purpose
`match_status`	Verified, ambiguous or unmatched, with the matching method and review evidence retained.

Keep ambiguous and unmatched references in the source table. Review candidate matches against the original bibliographic record, and report matching coverage before restricting an analysis to verified links. Missing identifiers or dates must remain missing rather than being inferred from scholarly citation counts. No patent-reference dataset is supplied by the worked example, so it cannot yield an observed linkage rate or lag distribution.

3. Institutional patent linkage. Define an eligible publication set for each institution using the same years, document types, affiliation rule and data coverage. Deduplicate publications by paper identifier, and specify how shared publications are credited. Use a common patent observation cutoff and, where possible, equal follow-up time for each publication cohort. Count a paper as observed patent-cited only when at least one verified patent-family link falls in that window; repeated documents within a family must not inflate linkage counts.

The observed linked-paper proportion is the number of distinct eligible papers with a verified in-window patent-family link divided by the number of eligible papers. Report the numerator, denominator, source coverage, unmatched and ambiguous records, and sensitivity to matching and follow-up choices. A paper without a link in the observed source is not proof that no patent cites it.

Without the independently validated patent-reference data, the missing inputs are the citing patent and family identifiers, reference matches, dates and coverage information. Report the analysis as unavailable. The scholarly-citation sample and institutional top-decile shares cannot replace those inputs.

Chapter 32 — Causal Inference in Scientometrics

1. Interaction sensitivity. Use the complete sampled records in `cohort_data` from Chapter 32.

```
unadjusted ← lm(cited_by_count ~ is_oa * recent_cohort, data = cohort_data)
adjusted ← lm(cited_by_count ~ is_oa * recent_cohort + referenced_works_count,
              data = cohort_data)
broom::tidy(unadjusted)
broom::tidy(adjusted)
```

The interaction coefficients describe associations in two conditional-mean specifications. A difference between them indicates specification sensitivity; it does not identify an OA treatment effect or by itself establish the size or direction of omitted-variable bias.

2. Boundary sensitivity.

```r
alternative ← cohort_data ▷ mutate(cohort_2018 = year ⩾ 2018)
alternative_model ← lm(cited_by_count ~ is_oa * cohort_2018 +
                         referenced_works_count, data = alternative)
broom::tidy(alternative_model)
```

There is no identified intervention at either date. This changes an arbitrary publication-cohort partition, so it is a descriptive sensitivity check, not a placebo intervention test. Neither a significant nor a non-significant interaction establishes parallel counterfactual trends; these data contain different cohorts' cumulative citations at retrieval, not annual outcomes around a policy change.

3. Comparison audit.

```r
matched ▷ count(control_id, name = "times_used") ▷ arrange(desc(times_used))
nrow(oa_papers) - nrow(matched)
```

The original within-year draw can reuse a closed-access paper for multiple OA papers. OA papers from years lacking a closed-access candidate are omitted. Such selection changes the comparison population and does not establish exchangeability. An actual policy study must first define treatment assignment and timing, a target causal effect, outcome measurement over comparable exposure periods, and a defensible comparison design. It also needs relevant pre-treatment covariates and an explicit account of identification and inference assumptions.

Chapter 33 — Reproducibility, RRIDs, and Badges

1. DAS detection. Screen available abstracts for candidate data/code availability mentions, then manually validate candidates against full texts and linked resources.

```r
library(tidyverse)

das_keywords ← c("data availability", "code availability",
                 "github\\.com", "zenodo", "figshare",
                 "data are available", "code is available")

das_pattern ← paste(das_keywords, collapse = "|")

works ▷
 filter(!is.na(abstract), nzchar(trimws(abstract))) ▷
 mutate(candidate_mention = str_detect(tolower(abstract), das_pattern)) ▷
 summarise(
   total = n(),
   candidate_mentions = sum(candidate_mention),
   candidate_rate = mean(candidate_mention)
 )
```

Key insight: Abstract screening can miss statements and produce false positives, so the match rate is neither a validated DAS rate nor a guaranteed lower bound. Inspect full texts and validate precision/recall on an annotated sample.

2. Cross-journal comparison. Compare metadata completeness and OA rates across journals.

```
journal_ids ← c("S148561398", "S205292342", "S4210197613")

journal_quality ← map(journal_ids, \(jid) {
 w ← fetch_openalex(entity = "works",
                primary_location.source.id = jid,
                from_publication_date = "2020-01-01",
                options = list(sample = 200, seed = 42))
 stopifnot(is.data.frame(w), nrow(w) > 0)
 tibble(
   journal = jid,
   abstract_rate = mean(!is.na(w$abstract) & nzchar(trimws(w$abstract))),
   oa_observed = sum(!is.na(w$oa_status)),
   oa_rate = mean(w$oa_status ≠ "closed", na.rm = TRUE),
   doi_rate = mean(!is.na(w$doi))
 )
}) ▷ list_rbind()

journal_quality
```

Key insight: Access status and metadata completeness describe different properties. Inspect actual output and coverage; no association is assumed.

3. Temporal trends. Track recorded access status and abstract coverage by publication cohort.

```
metadata_trends ← works_repro ▷
 group_by(year) ▷
 summarise(
   abstract_rate = mean(has_abstract),
   oa_rate = mean(oa_status ≠ "closed", na.rm = TRUE),
   .groups = "drop"
 ) ▷
 pivot_longer(-year, names_to = "metric",
              values_to = "value")

ggplot(metadata_trends, aes(year, value,
                            colour = metric)) +
 geom_line() +
 scale_y_continuous(labels = scales::percent) +
 labs(title = "Access status and metadata coverage by cohort",
      x = "Year", y = "Proportion")
```

Key insight: These are current recorded properties of publication cohorts, not adoption histories or measures of data/code sharing and reproducibility.

Chapter 34 — Reproducible Pipelines

1. Build a pipeline. Create a `_targets.R` file for a bibliometric workflow.

```r
library(targets)

# _targets.R content:
tar_option_set(packages = c("openalexR", "tidyverse"))

list(
 tar_target(raw_data, {
   fetch_openalex(entity = "works", search = "bibliometrics",
             from_publication_date = "2020-01-01")
 }),

 tar_target(clean_data, {
   raw_data |>
     filter(!is.na(doi), !is.na(publication_year)) |>
     select(id, doi, display_name, publication_year,
            cited_by_count, is_oa)
 }),

 tar_target(summary_stats, {
   clean_data |>
     group_by(publication_year) |>
     summarise(n = n(), mean_cites = mean(cited_by_count))
 }),

 tar_target(trend_plot, {
   ggplot(summary_stats, aes(publication_year, n)) +
     geom_col() +
     labs(title = "Publication trend", x = "Year", y = "Works")
 })
)

# Run with: tar_make()
# Verify: tar_visnetwork()
```

Key insight: Changing the summary function only re-runs `summary_stats` and `trend_plot`, not `raw_data` or `clean_data` — this is the power of dependency-based pipelines.

2. renv snapshot. Demonstrate dependency management with renv.

```r
# Step 1: Initialise renv
# renv::init()

# Step 2: Install a package
# install.packages("glue")
# library(glue)

# Step 3: Snapshot
# renv::snapshot()
# This creates/updates renv.lock

# Step 4: Simulate loss --- remove the package
# remove.packages("glue")
# library(glue)  # Error: package not found

# Step 5: Restore
# renv::restore()
# library(glue)  # Works again
```

Key insight: renv creates a project-local library and a lockfile, ensuring exact reproducibility of package versions across machines and time.

3. Branching. Use dynamic branching to analyse multiple journals in parallel.

```
library(targets)

# _targets.R with dynamic branching:
tar_option_set(packages = c("openalexR", "tidyverse"))

list(
 tar_target(journal_ids,
            c("S4210174107", "S137773608", "S49861241")),

 tar_target(journal_data,
   fetch_openalex(entity = "works",
             primary_location.source.id = journal_ids,
             from_publication_date = "2020-01-01"),
   pattern = map(journal_ids)
 ),

 tar_target(journal_summary,
   journal_data |>
     summarise(
       journal = first(so),
       n = n(),
       mean_cites = mean(cited_by_count)
     ),
   pattern = map(journal_data)
 )
)
```

Key insight: Dynamic branching avoids duplicating target definitions; each journal is processed independently and can run in parallel with `tar_make_future()`.

Chapter 35 — Interactive Dashboards

1. Multi-page dashboard. Create a Quarto dashboard with two pages.

```
# Save as dashboard.qmd:
# ---
# title: "Bibliometric Dashboard"
# format: dashboard
# ---
#
# # Overview
#
# ## Row
#
# ```{r}
# library(plotly)
# plot_ly(summary_data, x = ~year, y = ~n, type = "bar")
# ```
#
# ## Row
#
# ```{r}
# plot_ly(works, x = ~cited_by_count, type = "histogram")
# ```
#
# # Details
#
# ## Row
#
# ```{r}
# library(DT)
# datatable(works ▷ select(display_name, publication_year,
#                          cited_by_count, is_oa))
# ```
```

Key insight: Quarto dashboards use page-level headings (`#`) for navigation tabs and row/column-level headings (`##`) for layout.

2. Linked brushing. Use plotly selection to filter a DT table.

```
library(plotly)
library(crosstalk)
library(DT)

shared_data ← SharedData$new(works)

bscols(
 plot_ly(shared_data, x = ~publication_year,
         y = ~cited_by_count, type = "scatter",
         mode = "markers") ▷
   layout(dragmode = "select"),

 datatable(shared_data,
           options = list(pageLength = 10),
           selection = "multiple")
)
```

Key insight: `crosstalk::SharedData` enables linked brushing between any htmlwidget that supports it, without needing Shiny.

3. Custom tooltips. Add informative hover text to a plotly chart.

```
library(plotly)
library(tidyverse)

tooltip_data ← works ▷
 group_by(publication_year) ▷
 summarise(n = n(), mean_cites = round(mean(cited_by_count), 1))

plot_ly(tooltip_data,
       x = ~publication_year,
       y = ~n,
       type = "bar",
       text = ~paste0("Year: ", publication_year,
                      "<br>Papers: ", n,
                      "<br>Mean cites: ", mean_cites),
       hoverinfo = "text")
```

Key insight: Custom tooltips using `text` and `hoverinfo = "text"` provide context-rich information on hover, improving dashboard usability.

Chapter 36 — Parameterized Quarto Reports

1. Institutional template. Create a parameterized report for institutional profiles.

```
# Save as inst_profile.qmd:
# ---
# title: "Institutional Profile"
# params:
#   inst_id: "I123456789"
#   inst_name: "University of Example"
#   start_year: 2020
#   end_year: 2023
# ---
#
# ```{r}
# library(openalexR)
# library(tidyverse)
#
# works ← fetch_openalex(
#   entity = "works",
#   authorships.institutions.lineage = params$inst_id,
#   from_publication_date = paste0(params$start_year, "-01-01"),
#   to_publication_date = paste0(params$end_year, "-12-31")
# )
#
# cat(sprintf("Total publications: %d\n", nrow(works)))
# ```
#
# ## Publication Trend
# ```{r}
# works ▷
#   count(publication_year) ▷
#   ggplot(aes(publication_year, n)) +
#   geom_col() +
#   labs(title = params$inst_name)
# ```
#
# ## Citation Distribution
# ```{r}
# ggplot(works, aes(cited_by_count)) +
#   geom_histogram(bins = 30) +
#   scale_x_log10()
# ```
#
# ## Top Authors
# ```{r}
# works ▷ unnest(author) ▷
#   count(au_display_name, sort = TRUE) ▷
#   head(10)
# ```
```

Key insight: Parameters make the same template reusable for any institution; batch rendering generates a portfolio of reports efficiently.

2. Batch rendering. Render the template for multiple journals.

```r
library(quarto)

journals <- list(
 list(journal_id = "S4210174107", journal_name = "Scientometrics"),
 list(journal_id = "S137773608", journal_name = "Nature Physics"),
 list(journal_id = "S49861241", journal_name = "PLOS ONE")
)

walk(journals, \(j) {
 quarto_render(
   "journal_profile.qmd",
   execute_params = list(
     journal_id = j$journal_id,
     journal_name = j$journal_name,
     start_year = 2020,
     end_year = 2023
   ),
   output_file = paste0("report_", j$journal_name, ".html")
 )
})
```

Key insight: Batch rendering with `quarto_render()` and parameterised templates scales to hundreds of reports with minimal code.

3. Conditional content. Add a warning when the sample is too small.

```r
# In the parameterized .qmd file:
#
# ```{r}
# #| results: asis
# if (nrow(works) < 50) {
#   cat("<div class="callout callout-warn">\n")
#   cat("This sample contains fewer than 50 papers. ")
#   cat("The citation distribution may not be reliable.\n")
#   cat(":::\n")
# } else {
#   ggplot(works, aes(cited_by_count)) +
#     geom_histogram(bins = 30) +
#     scale_x_log10() +
#     labs(title = "Citation distribution")
# }
# ```
```

Key insight: Conditional logic with `results: asis` enables dynamic content switching — essential for robust automated reports that handle edge cases.

Chapter 37 — Building a Bibliometric Shiny App

1. Add network tab. Extend the app with a co-authorship network visualisation.

```
library(shiny)
library(visNetwork)
library(igraph)
library(tidyverse)

# Add to UI:
# tabPanel("Network",
#   visNetworkOutput("coauth_network", height = "600px")
# )

# Add to server:
# output$coauth_network ← renderVisNetwork({
#   req(data())
#   edges ← data() ▷
#     unnest(author) ▷
#     group_by(id) ▷
#     filter(n() ⩾ 2, n() ⩽ 10) ▷
#     reframe(pair = combn(au_display_name, 2, simplify = FALSE)) ▷
#     unnest_wider(pair, names_sep = "_") ▷
#     count(pair_1, pair_2, name = "weight") ▷
#     rename(from = pair_1, to = pair_2)
#
#   g ← graph_from_data_frame(edges, directed = FALSE)
#   comm ← cluster_leiden(g, resolution = 1.0)
#
#   nodes ← tibble(
#     id = V(g)$name, label = V(g)$name,
#     group = as.character(membership(comm))
#   )
#
#   visNetwork(nodes, edges) ▷
#     visOptions(highlightNearest = TRUE, selectedBy = "group")
# })
```

Key insight: Limiting the network to papers with 2-10 authors avoids overwhelming the visualisation with hyper-authored papers.

2. Caching. Implement server-side caching for repeated queries.

```
library(shiny)
library(memoise)

# Create a cached version of the data fetching function
fetch_cached ← memoise(function(journal_id, start_date, end_date) {
 fetch_openalex(
   entity = "works",
   primary_location.source.id = journal_id,
   from_publication_date = start_date,
   to_publication_date = end_date
 )
}, cache = cachem::cache_disk("shiny_cache"))

# In the server:
# data ← reactive({
#   fetch_cached(
#     input$journal_id,
#     paste0(input$start_year, "-01-01"),
#     paste0(input$end_year, "-12-31")
#   )
# })
```

Key insight: Disk-backed caching persists across sessions, so the same query never hits the API twice. Use `cachem::cache_disk()` for production apps.

3. Deployment. Deploy to shinyapps.io.

```
# Step 1: Install rsconnect
# install.packages("rsconnect")

# Step 2: Configure account
# rsconnect::setAccountInfo(
#   name = "your-account",
#   token = "your-token",
#   secret = "your-secret"
# )

# Step 3: Deploy
# rsconnect::deployApp(
#   appDir = ".",
#   appName = "bibliometric-explorer",
#   appTitle = "Bibliometric Explorer"
# )

# Step 4: Test with different inputs
# - Try different journal IDs
# - Test with narrow and wide date ranges
# - Verify that caching works (second load should be instant)
```

Key insight: Test with a variety of inputs before deploying; edge cases (empty results, very large corpora) should be handled gracefully with validation messages.

4. biblioshiny comparison. Run biblioshiny and compare features.

```
library(bibliometrix)
# biblioshiny()

# Feature comparison table:
# | Feature              | biblioshiny | Custom app |
# |----------------------|-------------|------------|
# | Data sources         | WoS, Scopus | OpenAlex   |
# | Network analysis     | Yes         | Basic      |
# | Topic analysis       | Yes         | No         |
# | Custom indicators    | Limited     | Full       |
# | Parameterization     | No          | Yes        |
# | Deployment           | Local only  | shinyapps  |
# | Reproducibility      | GUI-based   | Code-based |
#
# biblioshiny is comprehensive but not easily customisable;
# a custom app trades breadth for tailored functionality.
```

Key insight: biblioshiny is excellent for exploration but lacks reproducibility and customisation. A custom app is better for production workflows with specific analytical requirements.