Package: scimapR (via r-universe)

November 3, 2026

Type Package

Title Reproducible, Question-Driven, Embedding-Aware Science Mapping

Version 0.4.0.9000

Description A comprehensive toolkit for bibliometric and scientometric analysis, designed as a complement to the foundational 'bibliometrix' package (Aria & Cuccurullo, 2017, <doi:10.1016/j.joi.2017.08.007>). Provides unified ingestion from major bibliographic sources including 'OpenAlex', 'Crossref', 'Semantic Scholar', 'PubMed', 'arXiv', 'bioRxiv', 'ORCID', 'Overton', 'Unpaywall', 'Scopus', and 'Web of Science' exports; classical science mapping (co-citation, bibliographic coupling, co-authorship, co-word networks); modern citation indicators (CD index, Relative Citation Ratio, field-normalized scores, Uzzi novelty); embedding-based research cluster discovery; structured research-question objects with optional LLM-grounded screening; author trajectory analysis; equity and representation auditing with explicit confidence and method reporting; reproducible-by-construction corpus certificates enabling exact re-derivation of a corpus; retrieval-grounded conversational corpus exploration; and a comprehensive 'shiny' application with publication-ready figure and table export. Output is fully 'tidyverse'-compatible with tibble-based corpora, 'viridis'-themed 'ggplot2' visualizations, and 'tidygraph' network objects.

License MIT + file LICENSE

URL https://github.com/CTTIR/scimapR, https://cttir.github.io/scimapR/

BugReports https://github.com/CTTIR/scimapR/issues

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Imports cli, rlang (>= 1.1.0), tibble, dplyr, purrr, readr, xml2, jsonlite, httr2 (>= 1.0.0), tidygraph, ggplot2 (>= 3.4.0), viridisLite, fs, digest, lifecycle, stats, utils, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown, withr, httptest2, covr, pkgdown, bibliometrix, bibtex, openalexR, rcrossref, rentrez, reticulate, arrow, qs2, Rtsne, ragg, plotly, visNetwork, shiny (>= 1.8.0), bslib, DT, shinyWidgets, shinycssloaders, shinyjs, shinytest2, fontawesome, htmltools, ellmer, quarto, stringdist, memoise, cachem, Matrix, scales, patchwork, igraph, stringr, tidyr, yaml, ggraph, ggrepel, tidytext, stopwords, uwot, dbscan, openxlsx2, boot, magick, png, tidysynth

Config/testthat/edition 3

VignetteBuilder knitr

LazyData true

Config/roxygen2/version 8.0.0

Config/Needs/website CTTIR/themakR

Config/pak/sysreqs cmake libglpk-dev make libicu-dev libuv1-dev libxml2-dev libssl-dev libx11-dev

Repository https://cttir.r-universe.dev

Date/Publication 2026-11-03 09:56:39 UTC

RemoteUrl https://github.com/CTTIR/scimapR

RemoteRef HEAD

RemoteSha 97add3522668374a5fc4085fb3cb711b35b8f8b8

Contents

as_sm_corpus 6
autoplot.sm_corpus 7
is_sm_corpus 8
is_sm_question 9
is_sm_trajectory 9
print.sm_audit_summary 10
sm_affiliation_dict 11
sm_affiliation_match 12
sm_affiliation_methods 15
sm_affiliation_signals 15
sm_affiliation_summary 16
sm_affiliation_unmatched 17
sm_attribute_institution 18
sm_audit_funding 20
sm_audit_gender 21
sm_audit_geographic 23
sm_audit_oa 25
sm_author_trajectory 26

sm_bind_corpora . . . 27
sm_build_corpus . . . 28
sm_certificate . . . 29
sm_chat . . . 32
sm_chat_render . . . 34
sm_citation_maturity . . . 35
sm_citation_window . . . 36
sm_cite_corpus . . . 37
sm_cluster_evolution . . . 38
sm_cluster_hdbscan . . . 40
sm_cluster_kmeans . . . 41
sm_cluster_label . . . 42
sm_cluster_leiden . . . 45
sm_concept_vocabularies . . . 46
sm_corpus . . . 47
sm_corpus_for_question . . . 56
sm_corpus_from_tables . . . 57
sm_count . . . 59
sm_coverage_audit . . . 61
sm_coverage_breakdowns . . . 64
sm_dedupe . . . 65
sm_did . . . 66
sm_diff_corpora . . . 69
sm_embed_load . . . 70
sm_embed_save . . . 71
sm_embed_works . . . 71
sm_enrich_altmetric . . . 74
sm_enrich_concepts . . . 75
sm_enrich_openalex_impact . . . 76
sm_enrich_opencitations . . . 78
sm_enrich_orcid . . . 79
sm_enrich_retraction . . . 80
sm_enrich_ror . . . 81
sm_enrich_specter . . . 82
sm_enrich_unpaywall . . . 83
sm_example_corpus . . . 84
sm_example_db . . . 86
sm_example_files . . . 86
sm_export_covidence . . . 88
sm_export_csv . . . 88
sm_export_cytoscape . . . 89
sm_export_figure . . . 90
sm_export_gephi . . . 91
sm_export_quarto_report . . . 92
sm_export_rayyan . . . 93
sm_export_rds . . . 94
sm_export_table . . . 94
sm_export_vosviewer . . . 95

sm_export_zip . . . 96
sm_fetch_arxiv . . . 97
sm_fetch_biorxiv . . . 98
sm_fetch_crossref . . . 99
sm_fetch_openalex . . . 101
sm_fetch_orcid . . . 103
sm_fetch_overton . . . 104
sm_fetch_pubmed . . . 105
sm_fetch_semantic_scholar . . . 106
sm_field_clinical_trials . . . 107
sm_field_funder . . . 108
sm_field_pubmed_mesh . . . 109
sm_figure_manifest . . . 109
sm_filter_works . . . 110
sm_hash_corpus . . . 111
sm_import_rayyan . . . 112
sm_its . . . 113
sm_journal_in_index . . . 117
sm_link_references . . . 118
sm_lock . . . 119
sm_match_types . . . 120
sm_materialise . . . 121
sm_merge_screening_decisions . . . 122
sm_metric_collab_index . . . 123
sm_metric_disruption . . . 124
sm_metric_fnci . . . 125
sm_metric_g_index . . . 128
sm_metric_h_index . . . 129
sm_metric_m_index . . . 130
sm_metric_novelty . . . 131
sm_metric_rcr . . . 132
sm_metric_summary . . . 134
sm_network_citation . . . 136
sm_network_cocitation . . . 137
sm_network_collab . . . 138
sm_network_coupling . . . 139
sm_network_coword . . . 140
sm_network_semantic . . . 142
sm_oa_statuses . . . 143
sm_palette_qualitative . . . 144
sm_plot_bradford . . . 144
sm_plot_citation_network . . . 145
sm_plot_collab . . . 146
sm_plot_collab_turnover . . . 147
sm_plot_equity_dashboard . . . 148
sm_plot_evolution . . . 149
sm_plot_heaps . . . 150
sm_plot_landscape . . . 150

sm_plot_lotka . . . 151
sm_plot_production . . . 152
sm_plot_thematic_map . . . 153
sm_plot_top . . . 154
sm_plot_topic_pivots . . . 155
sm_plot_trajectory . . . 155
sm_provenance . . . 156
sm_query . . . 157
sm_question . . . 158
sm_read_auto . . . 160
sm_read_bib . . . 161
sm_read_cochrane . . . 163
sm_read_dimensions . . . 164
sm_read_endnote . . . 165
sm_read_lens . . . 167
sm_read_openalex_json . . . 168
sm_read_pubmed_xml . . . 169
sm_read_ris . . . 171
sm_read_scopus . . . 172
sm_read_wos . . . 174
sm_read_zotero . . . 176
sm_reconcile . . . 177
sm_refresh . . . 179
sm_register_input . . . 180
sm_researcher_profile . . . 182
sm_run_app . . . 183
sm_save_corpus . . . 183
sm_scale_color . . . 184
sm_screen_against_question . . . 185
sm_screen_prisma . . . 186
sm_screen_regex . . . 187
sm_screen_summary . . . 188
sm_self_citation . . . 189
sm_snapshot . . . 191
sm_staleness . . . 192
sm_summary_authors . . . 193
sm_summary_period . . . 193
sm_summary_sources . . . 194
sm_summary_works . . . 195
sm_synth . . . 196
sm_theme . . . 198
sm_to_bibliometrix . . . 198
sm_trajectory . . . 201
sm_validate . . . 202
validate_sm_corpus . . . 202

Index **204**

as_sm_corpus *Coerce objects to sm_corpus*

Description

Generic function to convert various data types to an sm_corpus object.

Usage

```
as_sm_corpus(x, ...)

## S3 method for class 'sm_corpus'
as_sm_corpus(x, ...)

## S3 method for class 'data.frame'
as_sm_corpus(x, source_label = "data.frame", ...)

## S3 method for class 'list'
as_sm_corpus(x, ...)

## S3 method for class 'bibliometrixDB'
as_sm_corpus(x, source_label = "bibliometrix-import", ...)
```

Arguments

x	An object to coerce.
...	Additional arguments passed to methods.
source_label	Label for provenance tracking. For a bibliometrix data frame, rows with DB = "ISI" are labelled "wos" unless source_label is given.

Value

An sm_corpus object.

bibliometrix data frames

as_sm_corpus() reads a data frame made by bibliometrix::convert2df() or sm_to_bibliometrix() (class bibliometrixDB), and the file readers with engine = "bibliometrix" use the same conversion:

- works from TI, AB (TI_raw and AB_raw where present, which keep the case of the file), PY, DT, TC, LA, DI and PM;
- authors and authorships from AU and AF, with one authorship row per author and address: the names in the brackets of C1 (C1raw where present) link authors to addresses. An address without names, as in older records, belongs to every author of the work; one whose names designate none of the authors belongs to the author of a single-author work and else to nobody. Authors are told apart by the full names of AF; an entry whose given names are only initials,

or that has no AF, belongs to the author with that surname and those initials when there is just one such author with a full name. RP gives is_corresponding: the authors it names before "(corresponding author)" or "(reprint author)", in any case, are TRUE on all their rows and the other authors of the work FALSE. Namesakes (the same surname and initials) are told apart by the address that RP gives with the name, and stay NA when it does not tell them apart; a work without RP, or whose RP designates none of its authors for certain, gets NA (unknown);

- sources from SO, SN, EI, J9 and JI (issn_l, issn, abbreviation, iso_abbreviation);
- references from CR, with the DOI of each item that has one;
- concepts from DE (DE_raw where present), ID, SC and WC, in the vocabularies author-keywords, the indexer vocabulary of the source, wos-subject-category and wos-category. ID is read as keywords-plus for Web of Science data (DB = "ISI"), mesh for PubMed, Dimensions, Lens and Cochrane, author-keywords for BibTeX and RIS files (where bibliometrix stores the keywords of the file in ID) and index-keywords otherwise;
- provenance with UT as source_id_external.

See Also

Other corpus: is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
corpus <- sm_example_corpus()
as_sm_corpus(corpus)
as_sm_corpus(corpus$works)
```

---

autoplot.sm_corpus *Autoplot for sm_corpus*

Description

Automatically generate a suitable plot for an sm_corpus based on the requested type.

Usage

```
## S3 method for class 'sm_corpus'
autoplot(
  object,
  type = c("production", "landscape", "thematic", "collab", "equity", "top", "lotka",
    "bradford"),
  ...
)
```

Arguments

object	An sm_corpus object.
type	Plot type.
...	Additional arguments passed to the plot function.

Value

A ggplot object.

See Also

Other plots: sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(), sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(), sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(), sm_theme()

Examples

```
corpus <- sm_example_corpus()
ggplot2::autoplot(corpus, type = "production")
```

---

is_sm_corpus *Test if an object is an sm_corpus*

Description

Test if an object is an sm_corpus

Usage

```
is_sm_corpus(x)
```

Arguments

x	An object to test.

Value

TRUE if x is an sm_corpus, FALSE otherwise.

See Also

Other corpus: as_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
corpus <- sm_example_corpus()
is_sm_corpus(corpus)
```

`is_sm_question` *Test if an object is an sm_question*

Description

Test if an object is an sm_question

Usage

```
is_sm_question(x)
```

Arguments

x	An object to test.

Value

Logical.

See Also

Other question: `sm_corpus_for_question()`, `sm_question()`, `sm_screen_against_question()`, `sm_screen_regex()`, `sm_screen_summary()`

`is_sm_trajectory` *Test if an object is an sm_trajectory*

Description

Test if an object is an sm_trajectory

Usage

```
is_sm_trajectory(x)
```

Arguments

x	An object to test.

Value

Logical.

See Also

Other trajectory: sm_author_trajectory(), sm_plot_collab_turnover(), sm_plot_topic_pivots(), sm_plot_trajectory(), sm_researcher_profile(), sm_trajectory

Examples

```
is_sm_trajectory(1)
```

---

print.sm_audit_summary

Combined equity audit summary

Description

Runs all available equity audits (geographic, gender, funding, OA status) on a corpus and returns a combined summary. Individual audit results can be passed as ... arguments to avoid re-computation; any missing audits are run with default parameters, except that the gender audit uses method = "manual" and audits that send web requests run only when online = TRUE.

Usage

```
## S3 method for class 'sm_audit_summary'
print(x, ...)

sm_audit_summary(corpus, ..., online = FALSE, call = rlang::caller_env())
```

Arguments

x	An audit object to print.
...	Optional pre-computed audit objects (e.g., sm_audit_geographic, sm_audit_gender, sm_audit_funding, sm_audit_oa). If provided, these are included directly rather than being re-run.
corpus	An sm_corpus object.
online	Logical. Whether audits that send web requests may run. Only the funding audit does: sm_audit_funding() looks up every DOI on Crossref. With the default FALSE, a funding audit is included only when one is passed in ...; otherwise it is reported as not run. Set TRUE to run it with its default Crossref source.
call	Caller environment for error reporting.

Value

An sm_audit_summary S3 object containing:

geographic An sm_audit_geographic result.

gender An sm_audit_gender result.

funding An sm_audit_funding result. A funding audit that was not run has no funders, source = "none" and coverage = NA.

oa An sm_audit_oa result.

overview A tibble with one row per audit: audit, coverage (NA for an audit that was not run) and top_finding.

See Also

Other audit: sm_audit_funding(), sm_audit_gender(), sm_audit_geographic(), sm_audit_oa()

Examples

```
corpus <- sm_example_corpus(n_works = 50, n_authors = 20)
# The funding audit sends batched Crossref requests, so it is not run
# unless online = TRUE.
summary_audit <- sm_audit_summary(corpus)
summary_audit$overview

# A precomputed audit is used as it is. This funding audit sends no
# request because its copy of the corpus has no DOIs.
no_doi <- corpus
no_doi$works$doi <- NA_character_
funding <- sm_audit_funding(no_doi)
summary_audit <- sm_audit_summary(corpus, funding)
print(summary_audit)
```

---

sm_affiliation_dict *Default affiliation-matching dictionary*

Description

A small, documented, user-overridable dictionary of institution name variants used by sm_affiliation_match(). It covers common military / medical institution variants (including German synonyms) plus email-domain fallbacks, and is designed to be extended by appending rows.

Usage

```
sm_affiliation_dict
```

Format

A tibble with three columns:

institution Canonical institution name (character).

pattern A case-insensitive Perl-compatible regular expression matched against affiliation strings (character).

email_domain Optional email domain used by the email-domain fallback, or NA (character).

Details

Patterns are case-insensitive PCRE with Unicode-aware \w, \b, \d and \s: sm_affiliation_match() matches them in Unicode mode by default, so a row added for a German, French or Scandinavian name can use these classes next to non-ASCII letters (see the Details of sm_affiliation_match(), including its unicode argument). In that mode the shipped row \bnato\b no longer matches "NATO" directly followed by an umlaut, as in a German compound word. The first pattern that matches wins, in row order, so add specific rows before generic ones.

The rows for "Bundeswehr Hospital" match its German name, its abbreviation, or "hospital" after "Bundeswehr" or after "German Armed Forces", "German Army" or "German Military", with at most one word in between. A bare "military hospital", "army hospital" or "armed forces hospital", which names hospitals of many countries, stays unmatched; up to scimapR 0.4.0 these phrases matched "Bundeswehr Hospital" wherever they appeared. The dictionary is a starting point: for an audit of your own institutions, pass a dictionary of them to sm_affiliation_match().

See Also

sm_affiliation_match(), sm_affiliation_unmatched()

Examples

```
sm_affiliation_dict
```

---

sm_affiliation_match *Match author affiliations to institutions*

Description

A tested, extensible institution matcher for the authorships table. It tags each authorship with the institution it belongs to, using a dictionary of name variants (multilingual and synonym-aware) and an optional email-domain fallback for records whose affiliation string is missing.

Because the matcher operates per authorship row, it naturally handles secondary / multiple affiliations per author (each authorship row is matched independently).

Usage

```
sm_affiliation_match(
  corpus,
  patterns = sm_affiliation_dict,
  fields = NULL,
  email_domain_fallback = TRUE,
  postcode_signal = FALSE,
  unicode = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
patterns	A dictionary of institution name variants. Either: • a named list mapping each canonical institution name to a character vector of case-insensitive regex variant patterns, or • a data frame with columns institution, pattern, and an optional email_domain (the form of the bundled sm_affiliation_dict). Defaults to sm_affiliation_dict.
fields	Character vector of authorships columns to search for affiliation text. Defaults to "raw_affiliation" (plus "email" if such a column exists). Values across multiple fields are concatenated per row.
email_domain_fallback	Logical; if TRUE (default) and an authorship has no pattern match but does have an email address (in an email column), match the email's domain against the dictionary's email_domain entries.
postcode_signal	Logical (default FALSE). When TRUE and the dictionary carries a postcode column, a postcode match is attempted as a last resort (lowest priority, after name tokens and email domains) so existing matches do not shift. Off by default.
unicode	Logical (default TRUE). Match the patterns in Unicode mode, so \w, \b, \d, \s and POSIX classes such as [[:alpha:]] know non-ASCII letters, digits and spaces (see Details). FALSE gives the ASCII-only classes of scimapR 0.4.0 and earlier, to reproduce earlier results.
call	Caller environment for error reporting.

Details

Dictionary patterns are case-insensitive Perl-compatible regular expressions (PCRE) with Unicode-aware \w, \b, \d and \s, matched against the affiliation text of each authorship. The rows are tried in dictionary order and the first pattern that matches wins, so list specific patterns before generic ones. sm_affiliation_unmatched() lists the most frequent affiliation strings that nothing matched.

In Unicode mode (unicode = TRUE, the default) each pattern is prefixed with the PCRE item (*UCP) unless its leading items already include it. Umlauts, accented letters and the sharp s are then word characters: \w consumes them, \b finds the edge of a word that starts or ends with one, and POSIX classes such as [[:alpha:]] are Unicode-aware too. In ASCII mode (unicode = FALSE) every non-ASCII letter is a non-word character, like punctuation: a pattern such as sanit\w+wesens cannot match the German word spelled with an a-umlaut, and \b finds no edge between a space and a word that starts with an umlaut. Unicode mode can therefore also lose matches. A pattern that used \W, \S, \D or \B to stand for a non-ASCII letter no longer matches it, and \b no longer finds an edge between an ASCII letter and a non-ASCII letter next to it: the shipped pattern \bnato\b does not match "NATO" directly followed by an umlaut, as in a German compound word. To keep ASCII semantics in one place, write the class out, for example [A-Za-z0-9_] instead of \w.

unicode = FALSE gives the matches of scimapR 0.4.0. sm_attribute_institution() calls sm_affiliation_match() with its defaults, and so matches in Unicode mode, when the corpus has no institution_match column; running sm_affiliation_match(unicode = FALSE) first gives its earlier results. sm_count(level

= "institution") no longer matches affiliations itself: it counts each authorship by its institution label, institution_match first, and by its raw affiliation string when it has none (see sm_count()).

The optional postcode column (used with postcode_signal = TRUE) is matched case-sensitively as a whole word, in the same mode: it is wrapped in \b, any run of whitespace in it matches any amount of whitespace (including none), and it is used as a regular expression without escaping.

To extend the dictionary, append rows to sm_affiliation_dict (or build your own data frame with the same columns) and pass it as patterns. For example rbind(sm_affiliation_dict, tibble::tibble(institution = "My Uni", pattern = "my university", email_domain = "myuni.edu")).

Value

The corpus with its authorships table gaining (or having updated) four columns:

institution_match Canonical institution name, or NA.

match_method "pattern", "email_domain", "postcode", or "none".

match_signal The signal that fired: "name_token", "email_domain", "postcode", or "none". Precedence (highest first): name token, email domain, postcode.

match_evidence The actual substring / domain / postcode that triggered the match (an audit trail), or NA.

Type-stable: a corpus with no authorships is returned unchanged with the four columns present and 0 rows. See sm_affiliation_summary() for a tidy breakdown.

See Also

sm_attribute_institution(), sm_affiliation_dict, sm_affiliation_unmatched()

Other affiliation: sm_affiliation_methods(), sm_affiliation_signals(), sm_affiliation_summary(), sm_affiliation_unmatched(), sm_attribute_institution()

Examples

```
corpus <- sm_example_corpus(n_works = 5, n_authors = 5)
corpus$authorships$raw_affiliation[1] <- "Bundeswehrkrankenhaus Berlin"
matched <- sm_affiliation_match(corpus)
matched$authorships$institution_match[1]

# In Unicode mode \w consumes the umlaut of a German unit name
corpus$authorships$raw_affiliation[2] <-
  "Zentrales Institut des Sanit\u00e4tswesens Nordstadt"
dict <- list("Central Medical Institute" = "sanit\\w+wesens")
matched <- sm_affiliation_match(corpus, patterns = dict)
matched$authorships$institution_match[2]
ascii <- sm_affiliation_match(corpus, patterns = dict, unicode = FALSE)
ascii$authorships$institution_match[2]
```

sm_affiliation_methods

Controlled vocabulary for affiliation match methods

Description

The set of values the `match_method` column of `sm_affiliation_match()` can take.

Usage

```
sm_affiliation_methods(describe = FALSE)
```

Arguments

`describe`	Logical; if `TRUE`, return a tibble of `level` + `description`, else the ordered character vector (default).

Value

A character vector or a tibble.

See Also

`sm_affiliation_signals()`

Other affiliation: `sm_affiliation_match()`, `sm_affiliation_signals()`, `sm_affiliation_summary()`, `sm_affiliation_unmatched()`, `sm_attribute_institution()`

Examples

```
sm_affiliation_methods()
```

sm_affiliation_signals

Controlled vocabulary for affiliation match signals

Description

The ordered set of values the `match_signal` column of `sm_affiliation_match()` / `sm_affiliation_summary()` can take. Using this helper (rather than hard-coding strings) means downstream filtering — e.g. separating institution-name matches from email-domain matches — cannot drift if the levels change.

Usage

```
sm_affiliation_signals(describe = FALSE)
```

Arguments

describe	Logical; if TRUE, return a tibble of level + a short description. If FALSE (default), return the ordered character vector of levels (suitable for factor(levels =)).

Value

A character vector (ordered, highest matching priority first) or a tibble of level/description.

See Also

sm_affiliation_match(), sm_affiliation_methods()

Other affiliation: sm_affiliation_match(), sm_affiliation_methods(), sm_affiliation_summary(), sm_affiliation_unmatched(), sm_attribute_institution()

Examples

```
sm_affiliation_signals()
sm_affiliation_signals(describe = TRUE)
```

---

sm_affiliation_summary

Summarise affiliation matches

Description

Tidy summary of the matches produced by sm_affiliation_match(), mirroring the audit-style summaries elsewhere in the package: counts of works and authorships flagged, broken down by institution and by match_signal.

Usage

```
sm_affiliation_summary(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus previously passed through sm_affiliation_match() (so its authorships carry institution_match and match_signal).
call	Caller environment for error reporting.

Value

A tibble with columns institution, match_signal (a factor; see sm_affiliation_signals()), n_authorships, n_works, and example_evidence (a representative matched-evidence string for that institution x signal — the audit trail), sorted by n_authorships descending. Type-stable: a 0-row tibble (with a warning) when no matches are present.

See Also

sm_affiliation_match()

Other affiliation: sm_affiliation_match(), sm_affiliation_methods(), sm_affiliation_signals(), sm_affiliation_unmatched(), sm_attribute_institution()

Examples

```
corpus <- sm_example_corpus(n_works = 5, n_authors = 5)
corpus$authorships$raw_affiliation[1] <- "Bundeswehrkrankenhaus Berlin"
corpus <- sm_affiliation_match(corpus)
sm_affiliation_summary(corpus)
```

---

sm_affiliation_unmatched

List affiliation strings that matched no institution

Description

Audits an affiliation dictionary. After sm_affiliation_match(), it lists the most frequent raw_affiliation strings of the authorships that no signal (name pattern, email domain or postcode) assigned to an institution, so the dictionary can be extended where that matters most.

Usage

```
sm_affiliation_unmatched(corpus, n = 25L, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus previously passed through sm_affiliation_match() (so its authorships carry institution_match).
n	Maximum number of strings to return: a positive whole number, or Inf for all of them. Default 25.
call	Caller environment for error reporting.

Value

A tibble with columns raw_affiliation (character), n_authorships (integer: unmatched authorships with that string) and n_works (integer: distinct works among them), at most n rows, sorted by n_authorships descending and then by raw_affiliation in code-point order. Authorships whose raw_affiliation is NA or blank are not listed. Type-stable: a 0-row tibble when every affiliation matched, and a 0-row tibble with a warning when the corpus has not been matched.

See Also

sm_affiliation_match(), sm_affiliation_summary()

Other affiliation: sm_affiliation_match(), sm_affiliation_methods(), sm_affiliation_signals(), sm_affiliation_summary(), sm_attribute_institution()

Examples

```
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
dict <- readr::read_csv(
  sm_example_files("example_realuse_affiliation_dict.csv"),
  col_types = "ccc"
)
matched <- sm_affiliation_match(fix, dict)
sm_affiliation_unmatched(matched, n = 5)
```

`sm_attribute_institution` *Attribute matched affiliations to a controlled institution vocabulary*

Description

Rolls institution matches (from `sm_affiliation_match()`) up to a controlled vocabulary, writing a normalised `institution_id` and `institution_name` onto the `authorships` table. Supports a ROR-backed vocabulary (via a user-supplied offline ROR table) or a `"custom"` vocabulary derived directly from the matched institution names.

Usage

```
sm_attribute_institution(
  corpus,
  vocabulary = c("ror", "custom"),
  ror_table = NULL,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus`. If it has no `institution_match` column, `sm_affiliation_match()` is run first with default settings, that is with the shipped sm_affiliation_dict; to attribute your own institutions, run it first with a dictionary of them.
`vocabulary`	`"ror"` (default) or `"custom"`.
`ror_table`	For `vocabulary = "ror"`, a data frame with columns `ror_id`, `name`, and `aliases` (aliases either a `;`-separated string or a list-column). Matching is case-insensitive against `name` and each alias, as well as against the `institution_match` value. A synthetic example ships at `system.file("extdata", "example_ror.csv", package = "scimapR")`.
`call`	Caller environment for error reporting.

Details

The institution_match column stays as it is. sm_count(level = "institution") labels each authorship by institution_match before institution_name and institution_id, so the roll-up to the vocabulary reaches the counts only for authorships without a match, and the "affiliation" breakdown of sm_coverage_audit() uses institution_name only when no authorship has a match. Up to scimapR 0.4.0, sm_count() counted by institution_id whenever any authorship had one. To count or break down by the vocabulary, put its names first, for example:

```
a <- corpus$authorships
corpus$authorships$institution_match <-
  dplyr::coalesce(a$institution_name, a$institution_match)
```

The ids written to matched rows are ROR ids or CUST: ids, which join corpus$institutions only where that table lists them, so functions that read the institutions table for these rows (such as sm_audit_geographic(by = "region")) find no entry for them. Unmatched rows keep their earlier ids and so their entries. Up to scimapR 0.4.0, unmatched rows lost their institution_id.

Value

The corpus with institution_id and institution_name set on the authorships rows that match the vocabulary (for "ror", institution_id holds the ROR id; for "custom", CUST: and the matched name). Unmatched rows keep the institution_id and institution_name they had, such as the OpenAlex institution ids of a fetched corpus, or NA – the function never errors on unmatched affiliations. Type-stable.

See Also

sm_affiliation_match()

Other affiliation: sm_affiliation_match(), sm_affiliation_methods(), sm_affiliation_signals(), sm_affiliation_summary(), sm_affiliation_unmatched()

Examples

```
ror <- utils::read.csv(
  system.file("extdata", "example_ror.csv", package = "scimapR"),
  stringsAsFactors = FALSE
)
corpus <- sm_example_corpus(n_works = 5, n_authors = 5)
corpus$authorships$raw_affiliation[1] <- "Charite Universitatsmedizin Berlin"
corpus <- sm_affiliation_match(corpus)
corpus <- sm_attribute_institution(corpus, vocabulary = "ror",
                                   ror_table = ror)
corpus$authorships$institution_name[1]
```

sm_audit_funding *Funding source audit*

Description

Audits the distribution of funding sources across works in a corpus. Funding metadata is extracted from either Crossref `funder` fields or OpenAlex `funders` and `awards` fields (also accepting legacy `grants`), depending on the `source` parameter.

The result includes funder counts, the proportion of works with known funding, and a concentration analysis.

Usage

```
sm_audit_funding(
  corpus,
  source = c("crossref", "openalex"),
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  call = rlang::caller_env()
)

## S3 method for class 'sm_audit_funding'
print(x, ...)
```

Arguments

`corpus`	An `sm_corpus` object.
`source`	Character. Where to look for funding data: `"crossref"` (uses DOIs to query Crossref funder metadata) or `"openalex"` (uses OpenAlex IDs, with DOI fallback).
`api_key`	OpenAlex API key, sent in the authorization header.
`mailto`	Contact address for Crossref requests.
`call`	Caller environment for error reporting.
`x`	An audit object to print.
`...`	Ignored.

Value

An `sm_audit_funding` S3 object containing:

funders Tibble with columns `funder_name`, `funder_doi`, `n_works`, `pct` (percentage of checked works).

coverage Proportion of checked works with at least one funder; `NA` if none could be checked, and 0 for an empty corpus.

n_works_total Total number of works in corpus.

n_works_funded Number of works with funding data.

n_works_checked Works returned by the source.

n_works_failed Works whose request failed; reported in a warning.

n_works_no_id Works without an identifier for the chosen source.

source The data source used.

See Also

Other audit: `print.sm_audit_summary()`, `sm_audit_gender()`, `sm_audit_geographic()`, `sm_audit_oa()`

Examples

```
# Works without a DOI need no Crossref lookup, so this runs offline and
# returns an empty audit.
corpus <- sm_example_corpus(n_works = 20, n_authors = 10)
corpus$works$doi <- NA_character_
funding <- sm_audit_funding(corpus, source = "crossref")
print(funding)

# Live lookup: batched Crossref requests (the example DOIs are invented,
# so no funders are found).
corpus <- sm_example_corpus(n_works = 3, n_authors = 3)
funding <- sm_audit_funding(corpus, source = "crossref")
print(funding)
```

---

sm_audit_gender *Gender representation audit*

Description

Audits the inferred gender distribution of authors in a corpus. Gender is inferred from first names using the specified method. Results include the overall distribution, authorship-position breakdown, and a coverage metric.

This function infers a binary gender proxy from names, which has well-documented limitations. See the Limitations section in the printed output and the details below.

Usage

```
sm_audit_gender(
  corpus,
  method = c("genderize", "ssa", "manual"),
  api_key = Sys.getenv("GENDERIZE_API_KEY"),
  cache_dir = tools::R_user_dir("scimapR", "cache"),
  call = rlang::caller_env()
)

## S3 method for class 'sm_audit_gender'
print(x, ...)
```

Arguments

corpus	An sm_corpus object.
method	Character. Gender inference method: • "genderize": Uses the genderize.io API (requires API key for high volume). • "ssa": Uses US Social Security Administration frequency tables (no API needed, but US-centric). • "manual": Uses pre-existing inferred_gender column in the authors table (no inference performed).
api_key	Character. API key for genderize.io. Read from the GENDERIZE_API_KEY environment variable by default.
cache_dir	Character. Directory for caching API results.
call	Caller environment for error reporting.
x	An audit object to print.
...	Ignored.

Details

Known limitations of automated gender inference:

- Name-based methods assign a binary gender proxy, which does not capture the full spectrum of gender identity.
- Accuracy varies dramatically by cultural context: names from East Asian, South Asian, and many African cultures are poorly served by Western-trained models.
- Non-binary, transgender, and gender-diverse individuals are systematically misclassified.
- The method cannot account for name changes, pen names, or initialised first names.

Value

An sm_audit_gender S3 object containing:

distribution Tibble with columns inferred_gender, count, pct.

by_position Tibble breaking down gender by authorship position: position_type ("first", "corresponding" for a corresponding author who is not first, else "middle/last"), inferred_gender, count and pct. Each work and author counts once, however many address rows the author has; corresponding authors are those with authorships$is_corresponding = TRUE (NA counts as not corresponding).

coverage Proportion of authors with an inferred gender.

method The method used.

n_names_failed Number of distinct names without an API answer; these are not cached and are retried on the next run.

confidence_summary Summary statistics for gender confidence scores.

See Also

Other audit: print.sm_audit_summary(), sm_audit_funding(), sm_audit_geographic(), sm_audit_oa()

Examples

```
corpus <- sm_example_corpus()
# Using manual method (no API call needed for examples):
gender_audit <- sm_audit_gender(corpus, method = "manual")
print(gender_audit)
```

---

sm_audit_geographic *Geographic representation audit*

Description

Audits the geographic distribution of a corpus, tabulating the representation of countries, regions, or World Bank income tiers. The analysis can be weighted by work count, total citations, first-authorship, or corresponding authorship. Each work counts once per distinct known group, with optional fractional credit.

The result includes a Gini coefficient for concentration and a coverage metric (proportion of works with known geography).

Usage

```
sm_audit_geographic(
  corpus,
  by = c("country", "region", "income_tier"),
  weight = c("count", "citations", "first-author", "corresponding"),
  method = c("full", "fractional"),
  call = rlang::caller_env()
)

## S3 method for class 'sm_audit_geographic'
print(x, ...)
```

Arguments

corpus	An sm_corpus object.
by	Character. Grouping variable: "country" (ISO 3166-1 alpha-2 codes from authorships), "region" (from institutions table), or "income_tier" (World Bank income classification from institutions).
weight	Character. How to weight each work: "count" (one per work and group, before applying method), "citations" (weighted by cited_by_count), "first-author" (only first-author affiliations), "corresponding" (only corresponding-author affiliations).

method	Counting rule: "full" gives each work one unit per distinct known group; "fractional" splits one unit equally among its known groups. Citation credit is split by the same shares.
call	Caller environment for error reporting.
x	An audit object to print.
...	Ignored.

Details

Coverage uses all corpus works as its denominator and counts works with at least one known group after the authorship filter. Missing geography is excluded from credit allocation; incomplete metadata can therefore change both coverage and the shares assigned to known groups.

Value

An sm_audit_geographic S3 object containing:

distribution Tibble with columns group, count, citations, pct, credit and weighted_citations. Counts and raw citations include each work once per group under both methods. Percentages and the Gini use credit, or weighted citations for citation weighting. Count percentages describe work-group pairs under full counting and works under fractional counting.

gini Gini coefficient of the distribution.

coverage Proportion of works with at least one known geography.

by The grouping variable used.

weight The weighting method used.

method The counting rule used.

x invisibly (print methods).

See Also

Other audit: print.sm_audit_summary(), sm_audit_funding(), sm_audit_gender(), sm_audit_oa()

Examples

```
corpus <- sm_example_corpus()
geo <- sm_audit_geographic(corpus)
print(geo)
```

sm_audit_oa *Open access status audit*

Description

Audits the open-access (OA) status distribution of works in a corpus. Uses the statuses of `sm_oa_statuses()`; every recognised status except `closed` is open. Missing and unrecognised values are unknown. The open percentage uses all works as the denominator and agrees with `sm_summary_works()` to one decimal place.

The result includes counts and percentages for each OA category, as well as a coverage metric.

Usage

```
sm_audit_oa(corpus, call = rlang::caller_env())

## S3 method for class 'sm_audit_oa'
print(x, ...)
```

Arguments

`corpus`	An `sm_corpus` object.
`call`	Caller environment for error reporting.
`x`	An audit object to print.
`...`	Ignored.

Value

An `sm_audit_oa` S3 object containing:

distribution Tibble with columns `oa_status`, `count`, `pct`, and logical `is_open` (`NA` for unknown statuses).

pct_open Percentage of works with an open status in `sm_oa_statuses()`.

coverage Proportion of works with a known OA status.

n_works Total number of works.

See Also

Other audit: `print.sm_audit_summary()`, `sm_audit_funding()`, `sm_audit_gender()`, `sm_audit_geographic()`

Examples

```
corpus <- sm_example_corpus()
oa <- sm_audit_oa(corpus)
print(oa)
```

`sm_author_trajectory` *Build an author trajectory analysis*

Description

Analyses the career trajectory of a single author within a corpus, partitioning their publication record into `n_periods` career stages. For each period, the function computes:

- **Career stages**: dominant topics (the three concept names found in most of the period's works) and publication volume.
- **Topic pivots**: how much the research focus shifted between periods, measured as the Jaccard distance of concept sets.
- **Collaborator turnover**: Jaccard similarity of co-author sets across consecutive periods, plus counts of new, kept, and lost collaborators.
- **Emerging collaborators**: co-authors who first appear in the most recent period.
- **H-index curve**: cumulative h-index at the end of each period.
- **Novelty curve**: average concept novelty per period, the mean number of distinct concept names per work (over the works with concepts).
- **Citation acceleration**: mean citations per period relative to the corpus-wide mean for the same years.

Usage

```
sm_author_trajectory(
  corpus,
  orcid = NULL,
  author_id = NULL,
  n_periods = 5L,
  call = rlang::caller_env()
)

## S3 method for class 'sm_trajectory'
print(x, ...)

## S3 method for class 'sm_trajectory'
format(x, ...)
```

Arguments

`corpus`	An `sm_corpus` object.
`orcid`	Character. ORCID of the author to analyse, with or without the `https://orcid.org/` prefix (both the argument and the corpus ORCIDs are compared without it). Either `orcid` or `author_id` must be supplied.
`author_id`	Character. Internal author ID. Either `orcid` or `author_id` must be supplied.
`n_periods`	Integer. Number of career periods to divide the publication span into. Default `5`.

call	Caller environment for error reporting.
x	An sm_trajectory object.
...	Ignored.

Value

An sm_trajectory S3 object.

- print(): x invisibly.
- format(): A single character string.

See Also

Other trajectory: is_sm_trajectory(), sm_plot_collab_turnover(), sm_plot_topic_pivots(), sm_plot_trajectory(), sm_researcher_profile(), sm_trajectory

Examples

```
corpus <- sm_example_corpus()
# Use the first author's ID
traj <- sm_author_trajectory(corpus, author_id = "A000000001")
print(traj)
```

---

sm_bind_corpora *Bind two corpora together*

Description

Combine two corpora, assigning new IDs to colliding records in the second input and rewriting their links in every table and embedding. This includes yearly citation counts (citations_by_year). The bound corpus keeps the metadata of corpus1, with the local input files (sm_register_input()) of both corpora, each file and content once.

Usage

```
sm_bind_corpora(corpus1, corpus2, mix_models = FALSE, verbose = TRUE)
```

Arguments

corpus1	An sm_corpus.
corpus2	An sm_corpus.
mix_models	Logical; bind embeddings that come from different models (or of which only one corpus records its model)? Defaults to FALSE. See Details.
verbose	Logical; report the number of re-keyed IDs and works whose normalised DOI also occurs in the first input?

Details

Work IDs shared by the inputs always get new values in corpus2. Author, source and institution IDs are shared only when their lookup rows agree on every common column, including missing and list values. Other collisions get new IDs above the numbers used by either input; external cited work IDs are reserved too. Shared DOIs remain separate records until sm_dedupe() is called. The merged corpus is unlocked.

Embeddings are bound when both corpora have them, with the same number of dimensions, and when they come from the same model as metadata$embedding records it (see sm_corpus()); the bound corpus keeps that record. Embeddings that record different models, or of which only one corpus records its model, are not comparable: the function stops (class scimapR_embedding_model_mismatch) unless mix_models = TRUE, which binds them and records the model as "mixed". Embeddings of two corpora that record no model, such as corpora embedded by earlier versions of scimapR, are bound as before, and the model of the bound rows stays unrecorded. When only one corpus has embeddings, or their widths differ, the bound corpus has none and warns with class scimapR_bind_embeddings_dropped.

Value

An sm_corpus object.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

---

sm_build_corpus *Build a corpus from multiple sources*

Description

Combine multiple file reads and/or API fetches into a single deduplicated sm_corpus.

Usage

```
sm_build_corpus(..., dedupe = TRUE, mix_models = FALSE, verbose = TRUE)
```

Arguments

...	One or more sm_corpus objects or file paths.
dedupe	Logical; deduplicate by normalised DOI after combining, keeping the most complete record and its child rows?
mix_models	Logical; bind embeddings that come from different models (or of which only one input records its model)? Passed to sm_bind_corpora(). Defaults to FALSE.
verbose	Logical; show progress messages?

Details

The inputs are bound in turn with sm_bind_corpora(), so its rules for embeddings apply: embeddings that record different models in metadata$embedding, or of which only one input records its model, stop the function (class scimapR_embedding_model_mismatch) unless mix_models = TRUE. Pass every file of a split export; the IDs stay unique.

Value

An sm_corpus object.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
c1 <- sm_example_corpus(n_works = 50, seed = 1)
c2 <- sm_example_corpus(n_works = 50, seed = 2)
combined <- sm_build_corpus(c1, c2, dedupe = TRUE)
```

---

sm_certificate *Create, rebuild from, and verify corpus certificates*

Description

A corpus certificate is a self-contained YAML document that records how a corpus was made and lets others check it: the corpus hash (sm_hash_corpus()) and the size of its tables, the steps recorded in its provenance (searches, file reads, enrichments and refreshes), a summary of its screening decisions, its embeddings and their model, and the local input files registered with sm_register_input(), each with its SHA-256 hash.

sm_certificate() creates a certificate from a corpus. sm_verify_certificate() compares a live corpus, and its local input files, against a certificate to detect divergence. sm_rebuild_from_cert() runs the recorded searches again (see "Rebuilding a corpus"); file reads and local inputs cannot be replayed, and verification checks them instead.

Each recorded step (queries) has a source, query, engine and kind: "api_query" for searches of a bibliographic API, "file" for records read from a local file (the file relative to the working directory when it lies below it, else its name), "enrichment" for steps such as sm_enrich_openalex_impact() that added fields to works already in the corpus, "refresh" for sm_refresh() and "manual" for synthetic records. The searches of sm_fetch_openalex(), sm_fetch_pubmed(), sm_fetch_crossref() and sm_corpus_for_question() also record their filter and n_max. The kind comes from the provenance column kind, else (for corpora and certificates written before kinds were recorded) from the source label, the query and the engine (see sm_provenance()).

Usage

```
sm_certificate(corpus, path = NULL, call = rlang::caller_env())

sm_rebuild_from_cert(cert, verbose = TRUE, call = rlang::caller_env())

sm_verify_certificate(
  corpus,
  cert,
  base_dir = NULL,
  call = rlang::caller_env()
)

## S3 method for class 'sm_certificate'
print(x, ...)

## S3 method for class 'sm_cert_verification'
print(x, ...)
```

Arguments

corpus	An sm_corpus object.
path	Character. Optional file path to write the certificate YAML. If NULL, the certificate is returned but not written to disk.
call	Caller environment for error reporting.
cert	A certificate object (list) or path to a YAML certificate file.
verbose	Logical. Print progress?
base_dir	NULL or the folder that the relative paths of the local inputs are relative to; looked in first. See "Local inputs".
x	An sm_certificate or sm_cert_verification object.
...	Ignored.

Details

Certificates write their timestamps (created, the corpus's build_date and last_refresh, and the registered_at of each local input) in UTC as ISO 8601 text, for example 2026-10-17T11:45:00Z, and read back both this form and the older one with a numeric offset. Printing a certificate shows created in UTC as well, so a certificate read back from YAML prints the same time as the one it was written from. The corpus timestamps may be stored as POSIXct, a Date, numeric seconds since 1970-01-01 UTC (as in corpora restored from tables) or ISO 8601 text. A missing or implausible value is written as NA.

embedding_info describes corpus$embeddings: its size (n_works, n_dimensions) and, from the record in corpus$metadata$embedding (see sm_corpus()), the model behind it: model, model_id, base_model_id, revision, base_revision, text, prefix, models, versions and created. method names how the embeddings were made: "sentence-transformers" or "adapters" (sm_embed_works()), "semantic-scholar" (sm_enrich_specter(), sm_fetch_semantic_scholar()) or "mixed" for rows of several models; it is "unknown" for embeddings without a record that fits them (such as

embeddings made by earlier versions of scimapR, or a matrix replaced by hand) and "none" when the corpus has no embeddings. In these two cases model and model_id say the same.

Value

For sm_certificate(): an sm_certificate S3 object (list), also written to path if provided. For sm_rebuild_from_cert(): an sm_corpus with the works of the replayed searches. For sm_verify_certificate(): an sm_cert_verification S3 object, a list with pass, checks, n_not_checked, corpus_hash (computed with the certificate's hash version), cert_hash, cert_created and hash_version.

x invisibly (print methods).

Local inputs

The local_inputs block lists the files of metadata$local_inputs (sm_register_input()) with their path, path_abs, root, basename, sha256, bytes, role, builder and registered_at. It is a tibble in the certificate object, with no rows when no file is registered, and a list with one entry per file in the YAML file.

sm_verify_certificate() adds one check per file, named input:<basename> (input:<path> when files share a name). It looks for a file with a relative path in base_dir, then in the folder of the certificate file (when cert is a path), then in the working directory, and last at path_abs; a file with an absolute path is looked for there and at path_abs. The first file found is hashed again, its check passes when the hash is the recorded one, and the check's location says where it was found. A file found nowhere is not checked, and the function warns (class scimapR_cert_inputs_missing), naming the files and the folders searched, because a verification run from the wrong folder would otherwise pass without a word.

Verification results

checks has one row per check, with the columns check, expected, actual (for a local input the first 16 characters of the hashes, or "missing"), pass, status and location. status (character) is "pass", "fail" or "not checked"; pass is TRUE, FALSE, or NA for a check that was not made. The overall pass is TRUE unless a check failed, so it can be TRUE while n_not_checked counts checks that were not made. The corpus hash is computed with the hash_version the certificate records, or with version 1 for certificates written before scimapR 0.5.0 (see sm_hash_corpus()).

Rebuilding a corpus

sm_rebuild_from_cert() replays the "api_query" steps of OpenAlex, PubMed and Crossref with the fetcher of their source (sm_fetch_openalex(), sm_fetch_pubmed() or sm_fetch_crossref()), the engine they were made with and the recorded filter and n_max. The searches of sm_corpus_for_question() record the client package that ran them (openalexR, rentrez or rcrossref) as their engine. A step without n_max, as in certificates of earlier versions, fetches up to the number of works of the certified corpus, with a message. The other steps (file reads, enrichments, refreshes, synthetic records and searches of other sources) and the local inputs cannot be replayed, and a message of class scimapR_cert_not_replayed names them. When a certificate has no search to replay but records file reads or local inputs, the function stops instead, with an error of class scimapR_cert_not_replayable that lists them, with the role, hash and builder of each local input. Without a search to replay, file

reads or local inputs (only synthetic records, enrichments, refreshes or searches of other sources, or no step at all), it returns an empty corpus.

The rebuilt corpus holds the works of the replayed searches, without duplicate DOIs and with new work ids, and their provenance; its other tables are empty. It shows what the recorded searches find today, so it does not pass sm_verify_certificate(): its hash and table sizes differ from those of the certified corpus.

See Also

Other reproducibility: sm_cite_corpus(), sm_diff_corpora(), sm_hash_corpus(), sm_provenance(), sm_register_input(), sm_snapshot()

Examples

```
corpus <- sm_example_corpus()
tracker <- tempfile(fileext = ".csv")
writeLines(c("doi,unit", "10.9999/example.1,Unit A"), tracker)
corpus <- sm_register_input(corpus, tracker, role = "publication tracker")
cert <- sm_certificate(corpus)
print(cert)
verification <- sm_verify_certificate(corpus, cert)
print(verification)

# a changed input file fails its check
cat("10.9999/example.2,Unit B\n", file = tracker, append = TRUE)
sm_verify_certificate(corpus, cert)$pass
```

---

sm_chat *Retrieval-grounded corpus chat*

Description

Ask a question of a corpus using retrieval-augmented generation (RAG). The function retrieves the most relevant works from the corpus using embeddings, TF-IDF, or a hybrid method, then sends them as context to a chat provider along with the user's question. The response includes inline citations keyed to work IDs. No hallucinated references are possible because citations are constrained to retrieved works.

Usage

```
sm_chat(
  corpus,
  question,
  provider = NULL,
  retrieve_n = 30L,
  retrieve_method = c("embedding", "tfidf", "hybrid"),
  cite = TRUE,
```

```
  max_tokens = lifecycle::deprecated(),
  temperature = lifecycle::deprecated(),
  call = rlang::caller_env()
)

## S3 method for class 'sm_chat_response'
print(x, ...)
```

Arguments

`corpus`	An `sm_corpus` object.
`question`	Character. The user's question in natural language.
`provider`	An ellmer chat provider object (e.g., from `ellmer::chat_openai()`). If `NULL`, errors informatively. Each request runs in a fresh copy: the object passed is unchanged, and its earlier turns and system prompt are not used.
`retrieve_n`	Integer. Number of works to retrieve as context. Default `30`.
`retrieve_method`	Character. Retrieval method: • `"embedding"`: cosine similarity on `corpus$embeddings`. • `"tfidf"`: TF-IDF on titles and abstracts. • `"hybrid"`: weighted average of both methods.
`cite`	Logical. If `TRUE`, the prompt asks the chat provider for inline citations using work IDs.
`max_tokens, temperature`	Deprecated; these arguments never affected requests. Set response parameters on the chat object instead.
`call`	Caller environment for error reporting.
`x`	An `sm_chat_response` object.
`...`	Ignored.

Value

An `sm_chat_response` S3 object with fields:

answer Character. The chat provider's response.

citations Tibble of cited work IDs and their details.

retrieved_works Tibble of the works sent as context.

prompt_hash SHA-256 hash of the full prompt for reproducibility.

model Character. The model used.

timestamp POSIXct. When the response was generated.

question The original question.

retrieve_method The retrieval method used.

See Also

Other chat: sm_chat_render()

Examples

```
## Not run:
# Requires ellmer package and chat-provider API access:
# response <- sm_chat(corpus, "What methods dominate?",
#                     provider = ellmer::chat_openai())

## End(Not run)
```

`sm_chat_render` *Render a chat response*

Description

Converts an sm_chat_response object into a formatted document string. The rendered output includes the question, answer with inline citations, and a reference list. Supported formats are Markdown, HTML, and Quarto.

Usage

```
sm_chat_render(
  response,
  format = c("markdown", "html", "quarto"),
  call = rlang::caller_env()
)
```

Arguments

response	An sm_chat_response object from sm_chat().
format	Character. Output format: "markdown", "html", or "quarto".
call	Caller environment for error reporting.

Value

A character string containing the rendered response. Printed to the console and returned invisibly.

See Also

Other chat: sm_chat()

Examples

```
# Requires a chat response object:
# rendered <- sm_chat_render(response, format = "markdown")
# cat(rendered)
```

`sm_citation_maturity` *Flag citation-immature recent years*

Description

Adds citation-maturity flags to the `works` table so that downstream trend and impact functions can shade or exclude recent, citation-immature years instead of treating their (artificially low) citation counts as final.

Works published within the most recent `lag` years – relative to the corpus's "as-of" date (`metadata$last_refresh` if present, else the maximum publication year) – are flagged provisional. `last_refresh` may be a POSIXct, a Date, numeric seconds since 1970-01-01 UTC (as in corpora restored from tables) or ISO 8601 text; its year is taken in UTC. A value that is not a plausible date (for example milliseconds stored as seconds) is ignored with a message.

Usage

```
sm_citation_maturity(corpus, lag = 2L, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus`.
`lag`	Integer number of recent years to treat as citation-immature (default `2`).
`call`	Caller environment for error reporting.

Value

The `corpus` with two logical columns added to `works`: `citation_mature` (`TRUE` when the year is mature) and `cnci_provisional` (the negation; `TRUE` when citation-based indicators should be treated as provisional). Type-stable: an empty corpus gains the columns with 0 rows.

See Also

`sm_its()`, `sm_metric_fnci()`

Other counting: `sm_citation_window()`, `sm_count()`, `sm_metric_summary()`

Examples

```
corpus <- sm_example_corpus(n_works = 20, seed = 1)
corpus <- sm_citation_maturity(corpus, lag = 2)
table(corpus$works$cnci_provisional)
```

`sm_citation_window` *Citations within a fixed window after publication*

Description

Counts each work's citations in its publication year and the `window` years after it, from the yearly citation counts of the corpus (`citations_by_year`, written by the OpenAlex readers and `sm_enrich_openalex_impact()`), and says whether the counts cover the whole window. Fixed windows compare old and recent works fairly, which `cited_by_count` (all citations up to now) does not.

Usage

```
sm_citation_window(corpus, window = 3L, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus`.
`window`	Number of years after the publication year, a whole number of 0 or more (default `3`). The window runs from the publication year through the publication year plus `window`, so it spans `window + 1` calendar years: `window = 3` counts the citations of a work from 2020 in 2020 to 2023.
`call`	Caller environment for error reporting.

Details

OpenAlex gives the citations of the ten most recent years only (its `counts_by_year`: the year of the counts and the nine before it), and leaves out the years without citations. A year missing from `citations_by_year` therefore counts 0 for a work that has yearly counts. The date of the counts is `works$citations_as_of` (the day OpenAlex last updated the record, or of the fetch); its year is still running, so it counts as partial. A corpus restored with `sm_corpus_from_tables()` may hold this date as a day count (days since 1970-01-01, as SQLite keeps dates) or as ISO 8601 text; both are read as the date they stand for.

A window is `complete` when the work has a publication year and yearly counts with a known date, its first year (the publication year) is no earlier than nine years before the year of the counts, and its last year is before the year of the counts. Otherwise `complete` is `FALSE` and `citations` is a lower bound: the citations the counts hold for the window years. A work without a publication year, or without yearly counts (no rows in `citations_by_year` and no `citations_as_of`), gets `citations = NA`.

The result depends only on the corpus, never on today's date, so it is the same in every session. A corpus without `citations_by_year` gives `NA` for every work, with one message (class `scimapR_no_citations_by_year`) that points to `sm_enrich_openalex_impact()`.

Value

A tibble with one row per work, in the order of `corpus$works`: `work_id` (character), `year` (integer, the publication year), `window` (integer), `citations` (integer) and `complete` (logical). A corpus without works gives a 0-row tibble with these columns.

See Also

sm_citation_maturity(), sm_enrich_openalex_impact()

Other counting: sm_citation_maturity(), sm_count(), sm_metric_summary()

Examples

```
works <- data.frame(
  work_id = c("W1", "W2", "W3"),
  title = c("Early work", "Recent work", "Unknown counts"),
  year = c(2018L, 2024L, 2019L),
  citations_as_of = as.Date(c("2026-10-17", "2026-10-17", NA))
)
yearly <- data.frame(
  work_id = c("W1", "W1", "W1", "W2"),
  year = c(2018L, 2019L, 2021L, 2025L),
  cited_by_count = c(2L, 5L, 4L, 3L)
)
corpus <- sm_corpus_from_tables(
  list(works = works, citations_by_year = yearly)
)
sm_citation_window(corpus, window = 3)
```

---

sm_cite_corpus *Generate citation block for a corpus*

Description

Produces a structured citation block for a corpus that can be included in manuscripts. The block documents the number of works, sources queried, date range, scimapR version, and corpus hash for reproducibility.

The build date is the UTC date of `metadata$build_date`, which may be a POSIXct, a Date, numeric seconds since 1970-01-01 UTC or ISO 8601 text; a missing or implausible value is shown as unknown.

Usage

```
sm_cite_corpus(
  corpus,
  style = c("text", "bibtex", "yaml"),
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
style	Character. Citation style: "text" for a plain-text paragraph, "bibtex" for a BibTeX entry, or "yaml" for a YAML block.
call	Caller environment for error reporting.

Value

A character string containing the formatted citation. Printed to the console and returned invisibly.

See Also

Other reproducibility: sm_certificate(), sm_diff_corpora(), sm_hash_corpus(), sm_provenance(), sm_register_input(), sm_snapshot()

Examples

```
corpus <- sm_example_corpus()
sm_cite_corpus(corpus)
```

---

sm_cluster_evolution *Track cluster evolution over time*

Description

Analyses how research clusters evolve across time windows. For each pair of consecutive windows, clusters are linked either by the overlap of the terms in their titles (link_by = "terms", the default) or, after one clustering of the whole corpus, by their shared cluster_id (link_by = "cluster_id").

Usage

```
sm_cluster_evolution(
  corpus,
  time_windows = NULL,
  link_threshold = 0.3,
  link_by = c("terms", "cluster_id"),
  noise = 0L,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object with a cluster_id column in corpus$works and a year column.
time_windows	A list of integer vectors defining year ranges for each window, or NULL (default). If NULL, windows are created automatically by splitting the year range into roughly equal periods.
link_threshold	Numeric between 0 and 1; minimum Jaccard similarity to link two clusters across time windows. Defaults to 0.3. Not used with link_by = "cluster_id".

`link_by` How clusters of consecutive windows are linked. `"terms"` (default) compares every pair of clusters by the Jaccard similarity of their title terms and keeps the pairs that reach `link_threshold`; use it when each window was clustered on its own. `"cluster_id"` follows a clustering of the whole corpus (for example from `sm_cluster_hdbscan()`): every cluster present in both windows gives one transition to itself.

`noise` The `cluster_id` that marks noise, the works no cluster claims: `0L` by default, as `sm_cluster_hdbscan()` assigns. With `link_by = "cluster_id"` noise gets no transition. `NULL` treats every id as a cluster; set it for clusterings that use another noise label or a real cluster 0. Not used with `link_by = "terms"`.

`call` Caller environment for error reporting.

Details

The function:

1. Splits works into time windows.
2. Within each window, identifies the cluster composition.
3. Between consecutive windows, links clusters. A work sits in exactly one window, so clusters cannot share works; `"terms"` therefore compares the terms of their titles and links the pairs whose Jaccard similarity reaches `link_threshold`, while `"cluster_id"` links each cluster to itself and reports the title-term Jaccard of its two windows without filtering.

With one clustering of the whole corpus, the same `cluster_id` in two windows is the continuity, and large clusters rarely reach the term threshold, so `"terms"` can report no transition at all. Use `link_by = "cluster_id"` there.

Value

A list with two components and the attribute `link_by` (the linking mode used):

`snapshots` A tibble with columns `window`, `cluster_id`, `n_works`, `top_terms` (a character string of representative terms) and `share` (the cluster's share of the window's clustered works, noise included, so the shares of a window sum to 1).

`transitions` A tibble with columns `from_window`, `to_window`, `from_cluster`, `to_cluster`, `jaccard` (the Jaccard similarity of the two clusters' title terms, rounded to 4 digits; 0 when neither has any) and `n_shared` (the number of shared title terms).

See Also

Other clustering: `sm_cluster_hdbscan()`, `sm_cluster_kmeans()`, `sm_cluster_label()`, `sm_cluster_leiden()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
corpus <- sm_cluster_kmeans(corpus, k = 5, reducer = "pca")
evo <- sm_cluster_evolution(corpus)
evo$snapshots
evo$transitions
```

```
# one clustering for the whole corpus: follow each cluster id
evo <- sm_cluster_evolution(corpus, link_by = "cluster_id")
evo$transitions
```

`sm_cluster_hdbscan` *HDBSCAN clustering of works*

Description

Clusters works using the HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) algorithm via `dbscan::hdbscan()`. Optionally reduces embedding dimensions first with UMAP or PCA.

Usage

```
sm_cluster_hdbscan(
  corpus,
  min_cluster_size = 15L,
  min_samples = NULL,
  reducer = c("umap", "pca", "none"),
  n_components = 5L,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object with embeddings.
`min_cluster_size`	Integer; minimum cluster size for HDBSCAN. Defaults to `15L`.
`min_samples`	Integer or `NULL`; minimum number of samples in a neighbourhood for a point to be a core point. If `NULL` (default), set equal to `min_cluster_size`.
`reducer`	Character; dimensionality reduction method to apply before clustering. One of `"umap"` (default), `"pca"`, or `"none"`.
`n_components`	Integer; number of dimensions to reduce to. Defaults to `5L`.
`call`	Caller environment for error reporting.

Details

Work IDs and embedding row names must be unique. Repeated IDs stop with class `scimapR_duplicate_ids`; use `sm_validate()` to inspect them.

Requires embeddings in `corpus$embeddings`. Compute them first with `sm_embed_works()` or load from cache with `sm_embed_load()`.

The reducer step helps HDBSCAN work in a lower-dimensional space where density estimation is more reliable.

UMAP requires its optional dependency only when reduction is needed. If the PCA dependency raises the known NULL-input error on older R, reduction retries with base SVD initialisation and reports `scimapR_umap_pca_fallback`. Other errors propagate unchanged.

Value

The input `corpus` with a `cluster_id` column added to `corpus$works`. Noise points (not assigned to any cluster) receive `cluster_id = 0L`. Works without a row in `corpus$embeddings` (such as works without text, see `sm_embed_works()`) get `NA`.

See Also

Other clustering: `sm_cluster_evolution()`, `sm_cluster_kmeans()`, `sm_cluster_label()`, `sm_cluster_leiden()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
corpus <- sm_cluster_hdbscan(corpus, min_cluster_size = 10L)
table(corpus$works$cluster_id)
```

---

sm_cluster_kmeans *K-means clustering of works*

Description

Clusters works using K-means clustering via `stats::kmeans()`. Optionally reduces embedding dimensions first with UMAP or PCA.

Usage

```
sm_cluster_kmeans(
  corpus,
  k,
  reducer = c("umap", "pca", "none"),
  n_components = 5L,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object with embeddings.
`k`	Integer; the number of clusters. Required.
`reducer`	Character; dimensionality reduction method to apply before clustering. One of `"umap"` (default), `"pca"`, or `"none"`.
`n_components`	Integer; number of dimensions to reduce to. Defaults to `5L`.
`call`	Caller environment for error reporting.

Details

Work IDs and embedding row names must be unique. Repeated IDs stop with class `scimapR_duplicate_ids`; use `sm_validate()` to inspect them.

Requires embeddings in `corpus$embeddings`. Compute them first with `sm_embed_works()` or load from cache with `sm_embed_load()`.

K-means is deterministic given a fixed random seed. Consider setting a seed before calling this function for reproducibility.

UMAP requires its optional dependency only when reduction is needed. If the PCA dependency raises the known NULL-input error on older R, reduction retries with base SVD initialisation and reports `scimapR_umap_pca_fallback`. Other errors propagate unchanged.

Value

The input `corpus` with a `cluster_id` column added to `corpus$works`. Works without a row in `corpus$embeddings` (such as works without text, see `sm_embed_works()`) get `NA`.

See Also

Other clustering: `sm_cluster_evolution()`, `sm_cluster_hdbscan()`, `sm_cluster_label()`, `sm_cluster_leiden()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
corpus <- sm_cluster_kmeans(corpus, k = 5, reducer = "pca")
table(corpus$works$cluster_id)
```

---

sm_cluster_label *Label clusters with representative terms*

Description

Assigns human-readable labels to clusters by extracting representative terms. Supports TF-IDF-based extraction (default), labels from the author and indexer keywords of the corpus, YAKE keyword extraction, or labels from a chat provider via the **ellmer** package.

Usage

```
sm_cluster_label(
  corpus,
  method = c("tfidf", "keywords", "yake", "llm"),
  n_terms = 5L,
  llm_provider = NULL,
  min_works = NULL,
  ngram = 1:2,
  stop_terms = NULL,
```

```
  noise = 0L,
  seed = NULL,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object with a cluster_id column in corpus$works.
method	Character; labelling method. One of "tfidf" (default), "keywords", "yake", or "llm".
n_terms	Integer; number of top terms to include in each cluster label. Defaults to 5L.
llm_provider	An **ellmer** chat provider object (e.g., ellmer::chat_openai()). Required when method = "llm", ignored otherwise. Each request uses a fresh copy; earlier turns and the object's system prompt are not used, and the object stays unchanged.
min_works	For "tfidf" and "keywords": the number of a cluster's works a term must occur in to enter its label. NULL (default) takes 3 or 5% of the cluster's works, whichever is more. When no term of a cluster reaches it, terms that occur in 2 of its works are used instead, as cluster_label_basis records.
ngram	For "tfidf": the lengths of the terms it scores, in words, as whole numbers from 1 to 3. The default 1:2 scores single words and pairs of adjacent words; 1 gives labels of single words. Two terms that overlap can be shown as the longer phrase they form (see Details). "keywords" uses it for the titles it falls back to.
stop_terms	Character vector of terms to leave out of the labels, such as generic words that every cluster shares; NULL (default) leaves out none. Each is compared with whole terms, ignoring case and the difference between singular and plural. Not used by "llm".
noise	The cluster_id that marks noise, the works no cluster claims: 0L by default, as sm_cluster_hdbscan() assigns. Noise gets no label and does not count for the labels of the other clusters. NULL labels every id as a cluster; set it for clusterings that use another noise label or a real cluster 0.
seed	Optional seed for sampling titles with a chat provider. The session random state is restored afterwards. NULL uses the session random stream as before. Used only by method = "llm".
call	Caller environment for error reporting.

Details

Noise and clusters without a label: works whose cluster_id is NA or noise get no label from any method, and the "tfidf" and "keywords" methods leave them out of the works that make a term common or rare in the corpus. A cluster whose works share no term, such as a cluster of a single work, gets no label either, with a message.

TF-IDF method: Splits the titles and abstracts of the works into terms of ngram words. A term's score in a cluster is the share of the cluster's works that contain it times log(N / n), where N is the number of clustered works and n the number of them that contain the term. As the score counts works, not occurrences, and a term must occur in at least min_works works of the cluster, a term that one paper repeats cannot name a cluster. The n_terms terms with the highest scores form the

label; ties go to the term that occurs more often in the cluster, then to the first in alphabetical order. A longer term takes the place of a shorter one it contains when it occurs in at least 40% of the shorter term's works ("cancer" gives way to "testicular cancer", and to "metastatic testicular cancer" only if that also occurs in 40% of the works with "cancer"), and a term contained in one already chosen is skipped. When a term overlaps one already chosen, as "heat illness" overlaps "exertional heat", the phrase they form ("exertional heat illness") counts as a longer term that contains both, also beyond the lengths in `ngram`, if it occurs in enough of the cluster's works to be a term of the label; so one phrase does not fill two places of a label.

Words of fewer than 3 characters, numbers and stop words are left out, and no term spans one of them, the end of a sentence or clause, or the border between title and abstract. The stop words of every work are the English ones of `tidytext::stop_words`, plus the Snowball list of the **stopwords** package for the work's own language in `works$language`, given as a name such as `"German"`, an ISO 639-1 code such as `"de"` or an ISO 639-2 code such as `"ger"`. Works without a language get the English list only, and the lists of other languages are never pooled, so German stop words such as "war" or "die" stay in English text. Without the **stopwords** package, or for a language it has no list for, only the English list is used, with a message.

Singular and plural count as one term: a final "ies" becomes "y", and a final "s" is dropped after a letter other than s, i, u, o, a and y, so "fractures" counts with "fracture" and "ticks" with "tick", while "diagnosis", "virus", "class" and "days" stay as they are. A possessive "'s" is dropped as well, and words such as "species", "series", "diabetes", "measles" or "aids" are kept as written, as are names of fields such as "logistics", "genetics", "genomics" or "pediatrics", which would otherwise count with the adjectives "logistic", "genetic", "genomic" or "pediatric". A label shows each term in the spelling the cluster's works use most often.

Keyword method: The same scoring on the author keywords and the indexer keywords (such as Keywords Plus or MeSH) in `corpus$concepts`, the families `"author_keyword"` and `"indexer_keyword"` of `sm_concept_vocabularies()`. Keywords are compared ignoring case, with hyphens read as spaces and plurals folded as above, and containment is checked on whole words; keywords that overlap form no phrase. A work counts once for each keyword, however many of its vocabularies give it. A keyword is shown as an author keyword spells it, or else in lower case. A cluster without a keyword in at least 2 of its works gets the TF-IDF terms of its titles instead; when the clustered works have no such keywords at all, every cluster does, with a message.

YAKE method: Uses a simplified YAKE-like scoring based on term frequency, position, and spread across works within each cluster, with the stop words of the TF-IDF method.

LLM method: Sends titles and abstracts of each cluster to an LLM with a prompt asking for a concise topical label. Requires the **ellmer** package and a configured provider.

Value

The input `corpus` with two columns added to `corpus$works`, in place of earlier ones:

`cluster_label` The terms of the work's cluster joined with `"; "`, the best first; `NA` for noise, for works without a cluster and for clusters that got no label.

`cluster_label_basis` How the label was made: `"tfidf"` or `"keywords"` when every term occurs in at least `min_works` of the cluster's works; `"tfidf (relaxed)"` or `"keywords (relaxed)"` when no term reached `min_works`, so that the terms occur in at least 2 works; `"tfidf (fallback)"` or `"tfidf (fallback, relaxed)"` when `method = "keywords"` found no keyword in 2 or more of the cluster's works and took the terms of its titles; `"yake"` or `"llm"` for these methods; `NA` where `cluster_label` is `NA`.

See Also

Other clustering: sm_cluster_evolution(), sm_cluster_hdbscan(), sm_cluster_kmeans(), sm_cluster_leiden()

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
corpus <- sm_cluster_kmeans(corpus, k = 5, reducer = "pca")
corpus <- sm_cluster_label(corpus, method = "tfidf", n_terms = 3L)
unique(corpus$works[, c("cluster_id", "cluster_label",
                        "cluster_label_basis")])

# keyword labels: the real-use example corpus grouped by journal category
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
fix$works$cluster_id <- match(fix$works$field, unique(fix$works$field))
fix <- sm_cluster_label(fix, method = "keywords", n_terms = 3L)
unique(fix$works[, c("field", "cluster_label", "cluster_label_basis")])
```

---

sm_cluster_leiden *Leiden community detection*

Description

Clusters works using the Leiden community detection algorithm via igraph::cluster_leiden(). If no network is provided, a semantic similarity network is built automatically from embeddings.

Usage

```
sm_cluster_leiden(
  corpus,
  network = NULL,
  resolution = 1,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
network	A tidygraph::tbl_graph or igraph::igraph object, or NULL. If NULL (default), a semantic similarity network is built via sm_network_semantic().
resolution	Numeric; resolution parameter for the Leiden algorithm. Higher values produce more (smaller) clusters. Defaults to 1.0.
call	Caller environment for error reporting.

Details

Work IDs and network node names must be unique. Repeated IDs stop with class `scimapR_duplicate_ids`; use `sm_validate()` to inspect them.

The Leiden algorithm (Traag et al., 2019) is a refinement of the Louvain algorithm that guarantees well-connected communities. It operates on the edge weights of the network.

When `network = NULL`, embeddings must be present in the corpus so that `sm_network_semantic()` can build a k-NN graph.

Value

The input `corpus` with a `cluster_id` column added to `corpus$works`. Works that are not nodes of the network, such as works without a row in `corpus$embeddings`, get `NA`.

See Also

Other clustering: `sm_cluster_evolution()`, `sm_cluster_hdbscan()`, `sm_cluster_kmeans()`, `sm_cluster_label()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
corpus <- sm_cluster_leiden(corpus, resolution = 1.0)
table(corpus$works$cluster_id)
```

---

sm_concept_vocabularies

Controlled vocabulary for concept vocabularies

Description

The canonical values of the `vocabulary` column of `corpus$concepts`. Readers write these names (for example `"author-keywords"` for author keywords and `"wos-category"` for the Web of Science categories), and the functions that select concepts by vocabulary, such as `sm_to_bibliometrix()` (DE) and `sm_network_coword()` (`field = "keywords"`), read them.

Each vocabulary belongs to a family:

- `author_keyword`: `author-keywords`, `manual-tags`;
- `indexer_keyword`: `keywords-plus`, `index-keywords`, `automatic-tags`, `mesh`;
- `category`: `wos-category` (WC, the Web of Science categories), `wos-subject-category` (SC, the research areas);
- `concept`: `openalex`, `arxiv`, `biorxiv`, `medrxiv`;
- `custom`: `custom`, for vocabularies of your own.

sm_corpus() and sm_corpus_from_tables() rename legacy and common spellings to these names, with a message: case, white space and the separators -, _, . and / do not matter, "keywords-author", "keyword" and "keywords" become "author-keywords", "medline" becomes "mesh", and the Web of Science field tags "DE", "ID", "WC" and "SC" name their vocabularies. Other values are kept and named in the message. The functions that select concepts by vocabulary read these spellings too, so saved corpora and tables assigned to corpus$concepts work as well. The column stays character.

Usage

```
sm_concept_vocabularies(describe = FALSE)
```

Arguments

describe	Logical; if TRUE, return a tibble with the factor columns level and family and a description. If FALSE (default), return the character vector of vocabularies.

Value

A character vector (grouped by family) or a tibble of level/family/description.

See Also

sm_to_bibliometrix(), sm_network_coword(), sm_read_wos()

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
sm_concept_vocabularies()
sm_concept_vocabularies(describe = TRUE)

corpus <- sm_example_corpus(n_works = 5, with_embeddings = FALSE)
table(corpus$concepts$vocabulary)
```

---

sm_corpus *Create an sm_corpus object*

Description

Build a typed, tibble-of-tibbles corpus container for bibliometric analysis. The corpus is the central data structure in scimapR, holding works, authors, authorships, institutions, sources, references, concepts, embeddings, provenance, screening decisions, and metadata.

Usage

```
sm_corpus(
  works,
  authors = NULL,
  authorships = NULL,
  institutions = NULL,
  sources = NULL,
  references = NULL,
  concepts = NULL,
  embeddings = NULL,
  provenance = NULL,
  screening = NULL,
  metadata = list(),
  citations_by_year = NULL
)

## S3 method for class 'sm_corpus'
x[i, ...]

## S3 method for class 'sm_corpus'
length(x)

## S3 method for class 'sm_corpus'
dim(x)

## S3 method for class 'sm_corpus'
as_tibble(x, ...)

## S3 method for class 'sm_corpus'
as.data.frame(x, ...)

## S3 method for class 'sm_corpus'
print(x, ...)

## S3 method for class 'sm_corpus'
format(x, ...)

## S3 method for class 'sm_corpus'
summary(object, ...)

## S3 method for class 'sm_corpus'
str(object, ...)
```

Arguments

works	A tibble of works (publications). See Details for schema. Common spellings of `oa_status` are renamed using `sm_oa_statuses()`, with a message; unrecognised values are kept and reported as unknown.

authors	A tibble of authors. If NULL, empty.
authorships	A tibble linking works to authors. If NULL, empty.
institutions	A tibble of institutions. If NULL, empty.
sources	A tibble of publication sources/journals. If NULL, empty.
references	A tibble of cited references. If NULL, empty.
concepts	A tibble of concepts/keywords. If NULL, empty. Legacy and common spellings in its vocabulary column (such as "keywords-author") are renamed to the names of sm_concept_vocabularies(), with a message. Values that name no known vocabulary are kept as written and listed in the same message, each time a corpus is built from them; name a vocabulary of your own "custom".
embeddings	A numeric matrix of work embeddings, or NULL.
provenance	A tibble tracking data lineage. If NULL, empty.
screening	A tibble of screening decisions. If NULL, empty.
metadata	A list of corpus-level metadata.
citations_by_year	NULL (default: the corpus has no yearly citation counts) or a tibble of yearly citation counts: work_id, year and cited_by_count, one row per work and year. Whole numbers are stored as integers. See Details.
x, object	An sm_corpus object.
i	Row index for subsetting.
...	Ignored.

Details

IDs are unique within a corpus; sm_bind_corpora() gives the second corpus new IDs where they collide.

A corpus is a list of tables that share identifiers: work_id links every table to works, author_id links authorships to authors, institution_id links them to institutions and source_id links works to sources. The tables below distinguish the standard schema columns (as in the empty tables) from optional columns that particular readers and functions write or read. A schema value may be NA. sm_corpus() adds missing schema columns to works (mostly NA); it takes other supplied tables as given. validate_sm_corpus() checks the required list elements, that works is a tibble with work_id, that metadata is a list, and the yearly citation schema below. It does not require or check every standard column of the other tables. Use sm_validate() for further data checks.

works: one row per work. Standard schema:

Column	Type	Content
work_id	character	Identifier, unique within the corpus
doi	character	DOI, lower case, without https://doi.org/
title, abstract	character	Title and abstract
year	integer	Publication year
type	character	Document type, as the source names it
source_id	character	The journal or other source (sources)

cited_by_count	integer	Citations, as the source counts them
oa_status	character	One of sm_oa_statuses(), or NA
language	character	Language
pmid, arxiv_id, openalex_id	character	Other identifiers
is_retracted	logical	Retraction flag
retraction_date	Date	Date of the retraction
last_refreshed	POSIXct	When the record was read or fetched

Optional: quartile, jif_quartile or sjr_quartile (journal quartiles such as "Q1", for sm_its(outcome = "share_q1")); numeric impact columns cnci, fwci, fnci, rcr, cnci_value, ncs, mncs and field_citation_ratio (searched in that order by sm_its() and sm_metric_summary()); field (for sm_metric_fnci(classification = "manual")); citation_mature and cnci_provisional (sm_citation_maturity()); cluster_id (the clustering functions); cluster_label and cluster_label_basis, a cluster's label and how it was made (sm_cluster_label()). The OpenAlex readers and sm_enrich_openalex_impact() write these impact columns:

Column	Type	Content
fwci	double	Field-weighted citation impact from OpenAlex
citation_percentile	double	OpenAlex citation percentile, 0 to 1
is_top_10pct	logical	Among the 10% most cited of field and year
is_top_1pct	logical	Among the 1% most cited of field and year
citations_as_of	Date	Date of these values and the yearly counts

The openalexR engine of sm_fetch_openalex() has no percentile, so its citation_percentile, is_top_10pct and is_top_1pct are NA. citations_as_of is the day of the OpenAlex record's updated_date, or of the fetch when the record has none; sm_read_openalex_json() leaves it NA for records without updated_date. sm_citation_window() reads it to tell complete citation windows from partial ones. A corpus restored with sm_corpus_from_tables() can hold it as a day count (days since 1970-01-01, as SQLite keeps dates) or as ISO 8601 text; sm_citation_window() and sm_enrich_openalex_impact() read both as the date they stand for.

authors: one row per author. Standard schema:

Column	Type	Content
author_id	character	Identifier, unique within the corpus
orcid	character	ORCID iD
display_name	character	Name
display_name_alternatives	list of character	Other spellings
inferred_gender, gender_method	character	Gender and its method
gender_confidence	double	Confidence of inferred_gender

display_name_alternatives holds, for example, the names as a Web of Science file writes them, and sm_audit_gender(method = "manual") reads the gender columns. Optional: none so far.

authorships: one row per work, author and affiliation, so an author with several addresses in a work has several rows, which share position and is_corresponding. Standard schema:

Column	Type	Content
work_id, author_id	character	The work and the author
position	integer	Place in the author list, from 1 (NA: unknown)
is_corresponding	logical	Corresponding author: see below
institution_id	character	The institution (institutions)
raw_affiliation	character	The address as the source writes it
country_code	character	ISO 3166-1 alpha-2 country code

is_corresponding is TRUE for the authors the source names as corresponding, FALSE for the other authors of such a work and NA when the source does not say who the corresponding author is. sm_read_wos() and as_sm_corpus() read it from the RP field, sm_read_openalex_json() and sm_fetch_openalex() from OpenAlex (a work whose authors are all marked false takes its corresponding authors from corresponding_author_ids, and counts as unknown when that lists none of them), sm_read_dimensions() from its Corresponding Authors column and sm_fetch_biorxiv() from the bioRxiv API; the other readers and fetchers write NA. Up to scimapR 0.4.0 they flagged the first author instead. A name that fits several authors of a work equally (namesakes with the same surname and initials) and that the source does not tell apart leaves them NA. Functions that select corresponding authors take TRUE only, and the leadership outcome of sm_its() leaves works whose corresponding author is unknown out of its shares.

Optional: institution_match, match_method, match_signal and match_evidence (sm_affiliation_match()); institution_name (sm_attribute_institution()); email (exported as EM by sm_to_bibliometrix()).

institutions: one row per institution. Standard schema:

Column	Type	Content
institution_id	character	Identifier, as in authorships
ror	character	ROR id
display_name	character	Name
country_code	character	ISO 3166-1 alpha-2 country code
region, income_tier	character	World region and income group
type	character	Kind of institution, such as education or healthcare

Optional: none so far.

sources: one row per journal or other source. Standard schema:

Column	Type	Content
source_id	character	Identifier, as in works
issn_l	character	Linking ISSN
issn	list of character	All ISSNs of the source
display_name	character	Name
type	character	Kind of source, such as journal
is_oa	logical	Open-access source
publisher, publisher_country	character	Publisher and its country

Optional: abbreviation and iso_abbreviation (character; the Web of Science abbreviations J9 and JI, written by sm_read_wos() and the bibliometrix conversion and exported by sm_to_bibliometrix()).

`references`: one row per cited reference. Standard schema:

Column	Type	Content
`work_id`	character	The citing work
`ref_index`	integer	Place in the work's reference list
`cited_work_id`	character	Cited work in `works`; see linking below
`cited_doi`	character	DOI of the cited work
`cited_raw`	character	The reference as the source writes it

Reference linking: readers, fetchers, `sm_bind_corpora()`, `sm_dedupe()` and `sm_link_references()` set `cited_work_id` from `cited_doi` or an OpenAlex id in `cited_raw` when the cited work is in `works`.

Optional: none so far.

`concepts`: one row per keyword, category or concept of a work. Standard schema:

Column	Type	Content
`work_id`	character	The work
`concept_id`	character	Identifier in its vocabulary
`concept_name`	character	The keyword, category or concept
`level`	integer	Level in the hierarchy of the vocabulary
`score`	double	Weight given by the source
`vocabulary`	character	One of `sm_concept_vocabularies()`

For MeSH headings `score` is 1 for a major topic and 0 otherwise. Optional: none so far.

`embeddings`: `NULL` or a numeric matrix with one row per embedded work, named by its `work_id`, and one column per dimension. Works may have no row: works without text to embed (`sm_embed_works()`), works missing from a cache (`sm_embed_load()`) or works for which a service has no vector (`sm_enrich_specter()`). The clustering functions give them no cluster (`NA`), and `sm_network_semantic()` and `sm_plot_landscape()` leave them out. `metadata$embedding` records the model (see below).

`provenance`: one row per work and source. Standard schema:

Column	Type	Content
`work_id`	character	The work
`source`	character	Where the record comes from, such as `"wos"`
`source_id_external`	character	Its identifier at the source
`fetch_date`	POSIXct	When it was read or fetched
`query`	character	The query or file it came from
`engine`	character	The parser or client that read it
`scimapR_version`	character	The scimapR version that read it
`prompt_hash`	character	Hash of a prompt behind the record

`source_id_external` is, for example, the Web of Science accession number (UT); `prompt_hash` is `NA` for records that no prompt produced. Optional:

Column	Type	Content
kind	character	The kind of step that wrote the row (see below)
filter	character	The filter of a search, else NA
n_max	integer	The maximum number of works of a search

kind is "api_query" for the searches of sm_fetch_openalex(), sm_fetch_pubmed(), sm_fetch_crossref() and sm_corpus_for_question(), which also write filter and n_max; "file" for the file readers, whose query holds the file relative to the working directory when it lies below it, else its name; and "enrichment" for the enrichers, such as sm_enrich_openalex_impact(), whose rows have the source "openalex_impact". sm_certificate() records the kind of each step, inferred from the source, query and engine for rows without kind, and sm_rebuild_from_cert() replays only searches, never a file read or an enrichment.

screening: one row per screening decision. Standard schema:

Column	Type	Content
work_id	character	The work
stage	character	Stage, such as "title-abstract" or "full-text"
decision	character	"include", "exclude" or "uncertain"
reason	character	Reason for the decision
confidence	double	Confidence of the decision
source	character	Who or what decided
decided_at	POSIXct	When

Optional: model records the chat-provider model for the decision. The legacy decision "unclear" is read as "uncertain".

citations_by_year: optional (NULL when the corpus has no yearly citation counts); one row per work and citation year, as the OpenAlex readers, sm_enrich_openalex_impact() and sm_corpus_from_tables() write it:

Column	Type	Content
work_id	character	The cited work
year	integer	The year of the citations
cited_by_count	integer	Citations received in that year, 0 or more

work_id and year may not be missing, and each pair appears once. OpenAlex gives the counts of the ten most recent years and leaves out the years without citations, so a missing year counts 0 for a work with yearly counts (see works$citations_as_of and sm_citation_window()). Subsetting keeps the rows of the kept works, sm_bind_corpora() binds them and sm_hash_corpus() includes them when there are any.

metrics: an optional legacy table read by the impact resolver when no impact column in works has values. It is not created by sm_corpus() or included in the standard empty schema:

Column	Type	Content

work_id	character	The work
Impact column	numeric	One of the impact names above, or value

The first named impact column, in the same order as for works and then value, is used when it contains a nonmissing value. Prefer impact columns in works for new corpora.

metadata: a list. Standard entries unless marked optional:

Entry	Type	Content
scimapR_version	character	The scimapR version that built the corpus
build_date	POSIXct	When the corpus was built
corpus_hash	character	Hash of the content (sm_hash_corpus())
is_locked	logical	Whether sm_lock() locked the corpus
last_refresh	timestamp	Last refresh without a failed request
n_identified	integer	Optional count before search deduplication
question_id	character	Id of its sm_question()
fetch_incomplete	list	Optional failed-fetch report (see below)

n_identified includes records removed by question-search deduplication or its result limit.

last_refresh advances only when sm_refresh() refreshed at least one work and no request failed. A recovered partial fetch has optional fetch_incomplete (list) with source, n_fetched, n_expected and reason; see the fetchers' Failed requests sections.

question_id is set by sm_corpus_for_question(). Optional entries locked_at (POSIXct), lock_reason (character) and lock_hash (character) record when, why and which content sm_lock() locked. Subsets remain locked but discard lock_hash. Other optional entries include those added by readers, such as source_file and reader; hash_version (integer), the algorithm of corpus_hash (sm_hash_corpus()), which sm_snapshot() writes (a snapshot without it used version 1); and local_inputs, a tibble of the local files the corpus was built from, with their SHA-256 hashes, which sm_register_input() and sm_corpus_from_tables() (inputs) write, sm_certificate() records and sm_verify_certificate() checks (see sm_register_input() for its columns). local_inputs is NULL until inputs are registered or supplied. build_date and last_refresh may also be numbers of seconds since 1970-01-01 UTC, as corpora restored from tables store them, ISO 8601 text or dates; sm_certificate() and the citation-maturity cut-offs read all of these. print() also reads these forms for last_refresh and shows YYYY-MM-DD HH:MM UTC. NULL and a single missing value print as "never"; unsupported values and vectors use format().

Optional: embedding, the record of the model behind embeddings, a list that sm_embed_works(), sm_enrich_specter(), sm_fetch_semantic_scholar(), sm_embed_load() (from the cache of sm_embed_save()) and sm_bind_corpora() write and sm_certificate() reports:

Entry	Type	Content
model	character	Short model name, "mixed" for several
model_id	character	Hugging Face id or Semantic Scholar field
base_model_id	character	Base model of an adapter, else NA
revision	character	Commit of the model snapshot, or NA
base_revision	character	Commit of the base model, or NA

backend	character	The Python package or the service used
text	character	The text embedded, such as "title_abstract"
prefix	character	Input prefix, else ""
versions	list	Versions of the Python packages used
models	character	The model ids behind the rows
n_works, n_dimensions	integer	Size of the matrix when made
created	POSIXct	When the record was written (UTC)

model_id is the Hugging Face id of the model (for "specter2" that of the adapter, whose base model is base_model_id) or the Semantic Scholar field of the vectors ("embedding.specter_v2" or "embedding.specter_v1"), whose revision is the model version that Semantic Scholar reports. backend is "sentence-transformers", "adapters", "semantic-scholar" or "mixed"; a mix lists its models in models. prefix is "query: " for "multilingual-e5". A record whose n_dimensions differs from the width of embeddings, or whose n_works is smaller than its number of rows, no longer describes the matrix and counts as no record. Embeddings without a record, such as those made by earlier versions of scimapR, have a model that is not recorded: sm_certificate() reports it as "unknown", and the functions that combine embeddings combine them with recorded ones only when asked to (mix_models).

Value

An sm_corpus S3 object.

- [: An sm_corpus with the selected works and only the authors, institutions and sources they refer to. Counts and the corpus hash describe the subset, including when all works are selected.
- length(): Number of works (integer).
- dim(): Integer vector of length 2 (rows, columns of works table).
- as_tibble(): The works tibble.
- as.data.frame(): The works table as a data frame.
- print(): x invisibly.
- format(): A single character string.
- summary(): A named list of summary statistics.
- str(): object invisibly.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
corpus <- sm_corpus(
  works = tibble::tibble(
    work_id = "W000000001",
    doi = "10.1234/example",
    title = "Example Work",
    abstract = "An example abstract.",
    year = 2024L,
    type = "journal-article",
    source_id = NA_character_,
    cited_by_count = 0L,
    oa_status = "closed",
    language = "en",
    pmid = NA_character_,
    arxiv_id = NA_character_,
    openalex_id = NA_character_,
    is_retracted = FALSE,
    retraction_date = NA_real_,
    last_refreshed = Sys.time()
  )
)
print(corpus)
```

---

sm_corpus_for_question

Build a corpus from a research question

Description

Takes a structured `sm_question()` object, translates its fields into database-specific query strings, fetches results from the requested bibliographic sources, deduplicates on DOI, and returns an `sm_corpus` with `metadata$question_id` linked to the question.

This function requires the appropriate API-client packages to be installed for each source (`openalexR` for OpenAlex, `rcrossref` for Crossref, `rentrez` for PubMed).

Usage

```
sm_corpus_for_question(
  question,
  sources = c("pubmed", "openalex", "crossref"),
  n_max = 1000L,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

question	An sm_question object created with sm_question().
sources	Character vector of sources to query. Subset of "pubmed", "openalex", "crossref".
n_max	Integer. Maximum total number of works to retrieve (across all sources, before deduplication). Default 1000.
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus with metadata$question_id set to the question's content-hash ID. Its provenance records each search (query, n_max and, as engine, the client package that ran it), so that sm_rebuild_from_cert() can run it again.

See Also

Other question: is_sm_question(), sm_question(), sm_screen_against_question(), sm_screen_regex(), sm_screen_summary()

Examples

```
q <- sm_question(
  text = "Does immunotherapy improve survival in melanoma?",
  framework = "PICO",
  population = "melanoma",
  intervention = "immunotherapy",
  outcome = "survival"
)
# Would query live APIs:
# corpus <- sm_corpus_for_question(q)
```

---

sm_corpus_from_tables *Construct an sm_corpus from a relational set of tables*

Description

A documented, validating constructor that builds an sm_corpus from a named list of data frames – making non-OpenAlex/WoS tabular sources first-class and side-stepping format-specific parsers. Required columns are validated against the sm_corpus schema, column types are coerced (with informative messages), absent optional tables are filled with correctly-typed 0-row tibbles, and the result is validated before being returned. Legacy and common spellings of concept vocabularies (such as "keywords-author" or "Author Keywords") are renamed to the names of sm_concept_vocabularies(), with a message. Values that name no known vocabulary are kept as written and listed in the same message, each time a corpus is built from them; name a vocabulary of your own "custom".

This is the recommended ingestion path for arbitrary tabular sources.

Usage

```
sm_corpus_from_tables(
  tables,
  schema = NULL,
  .coerce = TRUE,
  metadata = list(),
  inputs = NULL,
  call = rlang::caller_env()
)
```

Arguments

tables	A named list of data frames. Recognised names are works, authors, authorships, institutions, sources, references, concepts, provenance, screening and citations_by_year (yearly citation counts: work_id, year, cited_by_count; see sm_corpus()). works is required (and must have, or be coercible to, a work_id column; if absent, work ids are generated). A corpus built without a citations_by_year table has no such element.
schema	Optional named list of empty template tibbles overriding the built-in schema (advanced use). Defaults to the standard sm_corpus schema.
.coerce	Logical (default TRUE); coerce supplied columns to the schema's types. When FALSE, columns are used as-is (missing columns are still filled with typed NA, and concept vocabularies are still renamed).
metadata	Optional list of corpus-level metadata.
inputs	NULL (default) or the local files the tables come from, such as a database snapshot or a publication tracker, registered with sm_register_input() after the corpus is built: a character vector of paths, whose names (where given) become the role of each file, or a data frame with a path column and optional role and builder columns. Files already listed in metadata$local_inputs are kept, and a file registered again replaces its row. sm_certificate() records them with their SHA-256 hashes.
call	Caller environment for error reporting.

Value

A validated sm_corpus object.

See Also

sm_corpus(), as_sm_corpus()

Other reporting: sm_figure_manifest()

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

Examples

```
works <- data.frame(
  work_id = c("W1", "W2"),
  title = c("First", "Second"),
  year = c("2020", "2021"),   # character -> coerced to integer
  doi = c("10.1/a", "10.1/b")
)
authorships <- data.frame(
  work_id = c("W1", "W1", "W2"),
  author_id = c("A1", "A2", "A1"),
  position = c(1, 2, 1)
)
corpus <- sm_corpus_from_tables(list(works = works,
                                     authorships = authorships))
corpus

# record the file the tables were read from
snapshot <- tempfile(fileext = ".csv")
utils::write.csv(works, snapshot, row.names = FALSE)
corpus <- sm_corpus_from_tables(
  list(works = utils::read.csv(snapshot)),
  inputs = c("source table" = snapshot)
)
corpus$metadata$local_inputs[, c("basename", "role", "sha256")]
```

---

sm_count *Full and fractional output / impact counting*

Description

Attributes publication output (and citation impact) to entities using either full or fractional counting. Reviewers routinely ask for fractional counts so that a single multi-author / multi-institution paper is not counted in full for every contributor.

Usage

```
sm_count(
  corpus,
  method = c("full", "fractional"),
  level = c("institution", "author", "source"),
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus.
method	"full" (default) gives each entity credit 1 per work it appears on. "fractional" splits each work's single unit of credit equally among the distinct entities on that work.

level	Entity level: "institution" (default), "author", or "source".
call	Caller environment for error reporting.

Details

The fractional rule is the standard equal-share rule: a work with k distinct entities contributes $1/k$ of credit (and $1/k$ of its citations) to each. For level = "source" each work has a single source, so fractional and full counts coincide.

For level = "institution" each authorship row is labelled on its own, by the first of these authorships columns that has a value in that row: institution_match (from sm_affiliation_match()), institution_name, institution_id (named by corpus$institutions$display_name when that table lists the id) and raw_affiliation. A row labelled through institution_id has that id as entity_id; every other row has its label as both entity_id and entity_name. Rows labelled by their raw affiliation string are counted un-disambiguated, with a warning, because sm_count() applies no affiliation dictionary itself: run sm_affiliation_match() with a dictionary of your institutions first to count them. The order is the one sm_coverage_audit() uses for its affiliation levels, which however takes one column for all rows and falls back to country codes instead of raw strings. Up to scimapR 0.4.0, sm_count() used one column for every row (the first of institution_id, institution_match and raw_affiliation with any value), so rows without a value in it were left out, and it matched raw strings against the shipped sm_affiliation_dict before counting them.

sm_attribute_institution() keeps institution_match, so its roll-up of the matches to a controlled vocabulary (ROR ids and names in institution_id and institution_name) is counted only for rows without a match; up to scimapR 0.4.0 the counts followed institution_id. To count by the vocabulary, put its names first:

```
a <- corpus$authorships
corpus$authorships$institution_match <-
  dplyr::coalesce(a$institution_name, a$institution_match)
```

Value

A tibble, one row per entity, sorted by credit descending:

entity_id Entity identifier.

entity_name Human-readable name (falls back to entity_id).

n_works Number of distinct works the entity appears on (this is the full count and is identical for both methods).

credit Output credit: n_works under "full"; the sum of per-work fractional shares under "fractional".

weighted_citations Citation impact attributed to the entity: summed cited_by_count under "full"; summed fractional-share weighted cited_by_count under "fractional".

Type-stable: an empty/inapplicable corpus returns a 0-row tibble with these columns.

See Also

sm_metric_summary(), sm_affiliation_match()

Other counting: sm_citation_maturity(), sm_citation_window(), sm_metric_summary()

Examples

```
corpus <- sm_example_corpus(n_works = 30, seed = 1)
sm_count(corpus, method = "fractional", level = "author")

# Institutions: match the affiliation strings with a dictionary of your
# institutions first, then count
a <- corpus$authorships
corpus$authorships$raw_affiliation <- ifelse(
  a$position == 1L,
  "Unit North, Northtown",
  "Department of Surgery, Unit South"
)
dict <- list("Unit North" = "unit north", "Unit South" = "unit south")
matched <- sm_affiliation_match(corpus, patterns = dict)
sm_count(matched, method = "fractional", level = "institution")
```

---

sm_coverage_audit *Audit corpus coverage against a ground-truth reference*

Description

Computes recall, precision, and F1 of an `sm_corpus` against an external ground-truth `reference` (a manual tracker, an ORCID works set, an institutional-repository export, or another `sm_corpus`). This is the core primitive for a coverage / completeness audit: "how much of what we *should* have did we actually capture, and how much of what we captured is real?"

Matching is by normalised DOI with a fuzzy-title fallback (see `sm_reconcile()` for the shared matching engine). Full match provenance is retained so every decision can be inspected.

Usage

```
sm_coverage_audit(
  corpus,
  reference,
  by = NULL,
  match = c("doi_then_title", "doi", "title"),
  threshold = 0.9,
  index_table = NULL,
  call = rlang::caller_env()
)

## S3 method for class 'sm_coverage'
print(x, ...)

## S3 method for class 'sm_coverage'
summary(object, ...)

## S3 method for class 'sm_coverage'
autoplot(object, dim = NULL, ...)
```

Arguments

corpus	An sm_corpus object (the corpus under audit).
reference	The ground truth: an sm_corpus or a data frame with at least a DOI and/or title column (column names matched case-insensitively against doi/di and title/ti/display_name, the year against year/py/publication_year). A Date or date-time year column gives its calendar year, and a year outside 1000-2999 (such as a day count) counts as missing. Rows are identified by the first of the columns ut (Web of Science accession numbers), id, work_id, reference_id and ref_id whose values are unique and never missing, and otherwise by their position (R000001, ...), with a warning when such a column had to be passed over. In a data frame with Web of Science field tags (such as DI, TI, PY or UT) a column named ID holds the Keywords Plus and is never the identifier.
by	Optional character vector of breakdown dimensions. A name is either one of the named dimensions "year", "source" and "affiliation", written in lower case (see Details), or a column: of the reference data frame (matched case-insensitively) or of the works table of an sm_corpus (the exact name first, then case-insensitively). Each name is resolved on both sides; a name found on one side only gives levels whose columns for the other side are NA. A name that is neither a named dimension nor a column on either side is an error listing the available columns. The breakdowns label each dimension with the name as given.
match	Matching strategy: "doi_then_title" (default), "doi", or "title".
threshold	Minimum Jaro-Winkler title similarity ([0, 1]) to accept a title match. Default 0.9. When the optional stringdist package is not installed, the title fallback degrades to normalised exact matching. Titles only match between records published at most one year apart (when both years are known) and never between two records with different DOIs.
index_table	Optional journal index master list (same contract as sm_journal_in_index(); document expected columns there). When supplied, the audit additionally reports, per corpus record, whether its journal is **indexable** – so "is this journal in the index" and "is this record captured" are assessed together. When NULL (default) the result is identical to before.
call	Caller environment for error reporting.
x	An sm_coverage object.
...	Ignored.
object	An sm_coverage object.
dim	Which breakdown dimension to plot (defaults to the first available). If no breakdowns exist, a recall/precision summary bar is drawn.

Details

Named dimensions. "year" uses the publication years. "source" uses the source names of an sm_corpus (sources$display_name, or works$source_id when the corpus has no source table) and, in a data frame, the first of the columns source, source_title, journal, so and source_id, in this order (names matched case-insensitively); a source_id column holds ids, whose levels do

not join the names of the corpus side. "affiliation" uses, in a data frame, the first of the columns affiliation, affiliations, institution, country and country_code, in this order. A data-frame cell holds one value, so an item reported by two units takes two rows; a list column gives each of its values. Missing and blank values have no level. A named dimension that neither side can resolve gives a warning and no breakdown rows. Only "affiliation" joins a tracker's units with the institutions of the corpus: by = "institution" names the column itself, which the corpus side has only if works has a column of that name.

Affiliation levels of a corpus. For an sm_corpus (the corpus under audit, or a reference given as one), "affiliation" labels the authorships from the first of these authorships columns that has a value in at least one row: institution_match (from sm_affiliation_match()), institution_name (from sm_attribute_institution()), institution_id (named by corpus$institutions$display_name when that table lists the id, else the id itself) and, as a last resort, country_code. One column serves every row, so the levels of a side are all of one kind, and an authorship without a value in that column has no level. A message names the column used on each side, and a warning says when a side falls back to country codes, whose levels do not join institution names. Because sm_attribute_institution() runs sm_affiliation_match() first, institution_name is used only when institution_match is absent or empty; the Details of sm_attribute_institution() show how to break the coverage down by its vocabulary instead. sm_count(level = "institution") reads the same columns in the same order, but row by row, and counts the raw affiliation string where none has a value instead of falling back to countries; up to scimapR 0.4.0 it used, for every row, the first of institution_id, institution_match and the raw strings (matched with sm_affiliation_dict) that had a value in any row.

Full counting. A record belongs to every level of its values, so a work with authors at two institutions counts in both slices, and the per-level counts of a dimension can add up to more than n_corpus or n_reference. The overall recall and precision count each record once.

Value

An sm_coverage S3 object (a list) with components:

recall Reference records found in the corpus / total reference records. Each reference row is looked up on its own, so rows that report the same item (one paper reported by two units) are all found.

precision Matched corpus records / total corpus records. Each corpus record pairs with at most one reference row.

f1 Harmonic mean of recall and precision.

n_corpus, n_reference Integer record counts.

n_matched Reference records found in the corpus (the numerator of recall).

n_corpus_matched Corpus records paired with a reference record (the numerator of precision); equal to n_matched unless reference rows share a corpus record.

n_corpus_only, n_reference_only Unmatched counts.

matches Tibble with one row per corpus record: corpus_id, reference_id, match_type ("doi"/"title"/"none"), match_score. When index_table is supplied, also issn, indexable (logical), and indexed_title.

reference_matches Tibble with one row per reference record: reference_id, corpus_id (the corpus record it was found in), match_type and match_score; the recall-side counterpart of matches.

corpus_only Tibble of corpus records absent from the reference.

reference_only Tibble of reference records absent from the corpus.

breakdowns A single flat tibble (`dimension`, `level`, `n_reference`, `n_matched`, `recall`) across all by dimensions, from the reference side. Use `sm_coverage_breakdowns()` to access/filter it, also with the corpus-side counts and precision.

breakdowns_nested The legacy named list of per-dimension tibbles (`slice`, `n_reference`, `n_matched`, `recall`). Retained for one release; prefer `breakdowns`.

indexability When `index_table` is supplied, a summary tibble of indexable vs non-indexable record counts; otherwise `NULL`.

`print` returns `x` invisibly.

`summary` returns a one-row tibble of headline metrics: `recall`, `precision`, `f1`, `n_corpus`, `n_reference`, `n_matched`, `n_corpus_only`, `n_reference_only` and `n_corpus_matched`, so that `n_matched` + `n_reference_only` gives `n_reference` and `n_corpus_matched` + `n_corpus_only` gives `n_corpus`.

`autoplot` returns a `ggplot` object.

See Also

`sm_reconcile()`, `sm_journal_in_index()`, `sm_coverage_breakdowns()`, `sm_affiliation_match()`

Other coverage: `sm_coverage_breakdowns()`, `sm_journal_in_index()`, `sm_match_types()`, `sm_reconcile()`

Examples

```
corpus <- sm_example_corpus(n_works = 30, seed = 1)
# Pretend the reference is the corpus minus a few works, plus an extra one
ref <- corpus$works[1:25, c("work_id", "doi", "title", "year")]
cov <- sm_coverage_audit(corpus, ref, by = "year")
cov

# Any column of the reference or of the works is a dimension too
ref$unit <- rep(c("Unit A", "Unit B"), length.out = nrow(ref))
cov <- sm_coverage_audit(corpus, ref, by = c("unit", "type"), match = "doi")
sm_coverage_breakdowns(cov)
```

`sm_coverage_breakdowns`

Coverage breakdowns as a flat tibble

Description

Accessor returning the per-dimension coverage breakdowns of an `sm_coverage_audit()` result as a single flat (long) tibble, optionally filtered to one dimension.

Usage

```
sm_coverage_breakdowns(x, dimension = NULL, tidy = TRUE)
```

Arguments

x	An sm_coverage object.
dimension	Optional dimension name (e.g. "year") to filter to.
tidy	Logical (default TRUE). The stable long contract: returns columns dimension, level, n_reference, n_matched, recall, n_corpus, precision, f1. With tidy = FALSE, the legacy recall-only shape (dimension, level, n_reference, n_matched, recall) is returned for backward compatibility.

Details

This accessor honours the package's accessor return-type stability contract (see scimapR-stability): with tidy = TRUE the column set is guaranteed and will only change via a lifecycle deprecation.

The recall columns (n_reference, n_matched, recall) come from the reference side and the precision columns (n_corpus, precision) from the corpus side, joined on (dimension, level). A level found on the reference side only has n_corpus, precision and f1 NA; a level found on the corpus side only has n_reference, n_matched, recall and f1 NA. Under full counting (see the Details of sm_coverage_audit()) the per-level counts of a dimension can add up to more than the record totals. tidy = FALSE, print() and autoplot() show the same reference-side counts.

Value

A tibble (see tidy). Type-stable: a 0-row tibble with the documented columns when there are no breakdowns. Slices present on only one side (corpus vs reference) carry NA in the other side's counts/metric.

See Also

sm_coverage_audit(), scimapR-stability

Other coverage: sm_coverage_audit(), sm_journal_in_index(), sm_match_types(), sm_reconcile()

Examples

```
corpus <- sm_example_corpus(n_works = 30, seed = 1)
ref <- corpus$works[1:25, c("work_id", "doi", "title", "year")]
cov <- sm_coverage_audit(corpus, ref, by = "year")
sm_coverage_breakdowns(cov, dimension = "year")
```

---

sm_dedupe *Deduplicate corpus works by selected columns*

Description

Remove duplicate works, preferring the record with more complete data. Ties keep the first record. Kept records stay in their input order, with only their child rows and referenced lookup records.

Usage

```
sm_dedupe(corpus, by = "doi", verbose = TRUE)
```

Arguments

corpus	An sm_corpus.
by	Non-empty character vector of works columns that together identify duplicates. Default "doi". DOIs and titles are normalised, other character values are trimmed, and other types compare as stored. A missing or empty value in any key column never counts as a duplicate.
verbose	Logical; show progress?

Value

An sm_corpus object.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate(), validate_sm_corpus()

---

sm_did *Difference-in-differences for a treated vs control institution set*

Description

Fits a difference-in-differences (DiD) model comparing the yearly outcome series of a treated institution set against a control set, before and after intervention_year. Group membership is resolved via the institution-attribution columns from sm_affiliation_match() / sm_attribute_institution().

Usage

```
sm_did(
  corpus,
  treated,
  control,
  intervention_year,
  outcome = c("count", "share_q1", "cnci", "leadership"),
  family = NULL,
  lag = NULL,
  institution_aware = FALSE,
  lead_role = c("corresponding", "first", "last", "first_or_last", "any_lead"),
  shared = c("keep", "exclude", "lead"),
  min_cell = NULL,
  call = rlang::caller_env()
```

```
)

## S3 method for class 'sm_did'
print(x, ...)

## S3 method for class 'sm_did'
summary(object, ...)

## S3 method for class 'sm_did'
autoplot(object, ...)
```

Arguments

corpus	An sm_corpus with institution-attribution columns.
treated, control	Character vectors of institution labels (matched against institution_name / institution_match / institution_id). The two may not share a label.
intervention_year	Year the intervention takes effect: one whole number such as 2020, or text holding one.
outcome	One of "count", "share_q1", "cnci", "leadership" (see sm_its()).
family	Optional GLM family; defaults to Gaussian (DiD is an additive contrast). Override for, e.g., a Poisson count model; a Poisson fit warns when the counts are overdispersed.
lag	Citation-maturity lag in years (default 2), as in sm_its(): for "cnci" the most recent lag years are citation-immature and excluded.
institution_aware	For outcome = "leadership": if TRUE, each group's share counts the works whose author in the role lead_role belongs to the group's own institutions (the treated series uses institutions = treated, the control series institutions = control, as in sm_its()). FALSE (default) counts the works with a corresponding author.
lead_role	The author role that leads a work, as in sm_its(), for outcome = "leadership" and for shared = "lead". For the leadership outcome, roles other than "corresponding" need institution_aware = TRUE.
shared	What to do with works that belong to both groups, such as works co-authored by a treated and a control institution: "keep" (default) keeps them in both series, "exclude" leaves them out of both, and "lead" puts each in the series of the group whose institution holds its author in the role lead_role (a work led from both groups or from neither leaves both series).
min_cell	For outcome = "cnci": NULL (default) or a positive integer, passed to sm_metric_fnci() when the impact is derived from citation counts, as in sm_its().
call	Caller environment for error reporting.
x	An sm_did object.
...	Ignored.
object	An sm_did object.

Details

Each group's yearly series is built as in sm_its(). For "count" a group's years without works count 0, over every year from the first to the last publication year of the corpus. For the other outcomes a year in which a group has works but no usable value is dropped from that group's series, with a warning, and listed in dropped_years. For "cnci" the citation-immature years are then excluded from both series, by the same rule as in sm_its(), and listed in provisional_years; up to scimapR 0.4.0 sm_did() kept them. Each group needs 4 or more years, with at least one before intervention_year and one from it on; otherwise sm_did() aborts. The confidence interval uses the reference distribution of the p-value (t with the residual degrees of freedom for the default Gaussian family); no correction for autocorrelation is applied.

A work belongs to a group when one of its authorships carries one of the group's labels, so a work co-authored by both groups belongs to both. Such shared works pull the two series together and the estimate toward 0. sm_did() warns when there are shared works and shared is not set, and records them in shared_works. For "leadership" the denominators leave out the works whose lead is unknown, as in sm_its().

Value

An sm_did S3 object with components: model, did_estimate (a one-row tibble: estimate, std.error, conf.low, conf.high, p.value), series (group-by-year tibble: year, group, value, n_works), dropped_years (tibble: group, year; the group-years left out for lack of a usable value), provisional_years (integer; the citation-immature years excluded for "cnci"), shared_works (tibble: work_id, series; the works of both groups and where they went: "both", "treated", "control" or "none"), shared (the policy), leadership (the leadership definition, as in sm_its(), with institutions the labels of each group when institution_aware; NULL for the other outcomes), impact_source (where the impact values of "cnci" come from, as in sm_its(); NA for the other outcomes), and metadata (outcome, intervention_year, family, lag).

print returns x invisibly.

summary returns the DiD estimate tibble.

autoplot returns a ggplot (parallel trends + divergence).

See Also

sm_its(), sm_synth()

Other causal: sm_its(), sm_synth()

Examples

```
corpus <- sm_example_corpus(n_works = 120, seed = 1)
# Tag two institution groups for illustration: each work belongs to one
a <- corpus$authorships
group <- rep(c("Inst A", "Inst B"), length.out = nrow(corpus$works))
corpus$authorships$institution_name <-
  group[match(a$work_id, corpus$works$work_id)]
did <- sm_did(corpus, treated = "Inst A", control = "Inst B",
              intervention_year = 2020, outcome = "count")
did
```

`sm_diff_corpora` *Compare two corpora*

Description

[Superseded]

Produces a detailed comparison between two `sm_corpus` objects, reporting works added, removed, and changed; author differences; reference differences; and screening changes. This is useful for auditing how a corpus evolved between snapshots or after a refresh.

This function compares strictly by internal `work_id`. For content-based reconciliation across corpora that do not share identifiers (matching by normalised DOI with a fuzzy title fallback), use `sm_reconcile()`.

Usage

```
sm_diff_corpora(corpus1, corpus2, call = rlang::caller_env())

## S3 method for class 'sm_corpus_diff'
print(x, ...)
```

Arguments

`corpus1`	An `sm_corpus` object (the "before" state).
`corpus2`	An `sm_corpus` object (the "after" state).
`call`	Caller environment for error reporting.
`x`	An `sm_corpus_diff` object.
`...`	Ignored.

Value

An `sm_corpus_diff` S3 object (a list) with components:

added Tibble of works in `corpus2` but not `corpus1`.

removed Tibble of works in `corpus1` but not `corpus2`.

changed Tibble of works present in both but with differing fields.

summary A one-row summary tibble with counts.

hash1 Hash of `corpus1`.

hash2 Hash of `corpus2`.

See Also

`sm_reconcile()`

Other reproducibility: `sm_certificate()`, `sm_cite_corpus()`, `sm_hash_corpus()`, `sm_provenance()`, `sm_register_input()`, `sm_snapshot()`

Examples

```
c1 <- sm_example_corpus(seed = 1L)
c2 <- sm_example_corpus(seed = 2L)
d <- sm_diff_corpora(c1, c2)
print(d)
```

`sm_embed_load` *Load embeddings from disk*

Description

Loads a previously saved embeddings matrix from an RDS file and attaches it to the corpus. Only embeddings for works present in the corpus are loaded; mismatches are reported.

The loaded embeddings replace any the corpus had, and so does the record of their model (`corpus$metadata$embedding`, see `sm_corpus()`): the cache's record when it describes the cached matrix, else none, so the model of embeddings from an older cache or a bare matrix counts as not recorded (`"unknown"` in `sm_certificate()`).

Usage

```
sm_embed_load(corpus, path, call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object.
`path`	Character; file path to read embeddings from (an `.rds` file created by `sm_embed_save()`).
`call`	Caller environment for error reporting.

Value

The input `corpus` with `corpus$embeddings` and `corpus$metadata$embedding` updated. Works without a cached row get no row, so the clustering functions give them no cluster (`NA`).

See Also

Other embedding: `sm_embed_save()`, `sm_embed_works()`

Examples

```
## Not run:
corpus <- sm_example_corpus(with_embeddings = FALSE)
corpus <- sm_embed_load(corpus, "my_embeddings.rds")
dim(corpus$embeddings)

## End(Not run)
```

sm_embed_save *Save embeddings to disk*

Description

Saves the embeddings matrix from an sm_corpus to an RDS file on disk. This allows caching expensive embedding computations for later reuse with sm_embed_load(). The cache keeps the record of the model behind the embeddings (corpus$metadata$embedding, see sm_corpus()) when the corpus has one.

Usage

```
sm_embed_save(corpus, path, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object with a non-NULL embeddings matrix.
path	Character; file path to write the embeddings to. Should end in .rds.
call	Caller environment for error reporting.

Value

The input corpus (invisibly).

See Also

Other embedding: sm_embed_load(), sm_embed_works()

Examples

```
## Not run:
corpus <- sm_example_corpus(with_embeddings = TRUE)
sm_embed_save(corpus, "my_embeddings.rds")

## End(Not run)
```

sm_embed_works *Compute work embeddings using transformer models*

Description

Generates dense vector embeddings for works in an sm_corpus using pretrained transformer models in Python. The default model, specter2, loads the SPECTER2 adapter with the Python package adapters; the other models load with sentence-transformers. Requires **reticulate** and a Python installation with the packages the chosen model needs.

Usage

```
sm_embed_works(
  corpus,
 model = c("specter2", "scincl", "scibert", "minilm-l6", "mpnet", "multilingual-mpnet",
    "multilingual-e5"),
  text = c("title_abstract", "title", "abstract"),
  batch_size = 32L,
  cache_dir = tools::R_user_dir("scimapR", "cache"),
  python = NULL,
  install_deps = FALSE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
model	Character; the embedding model to use. One of "specter2" (default, optimised for scientific text), "scincl", "scibert", "minilm-l6" and "mpnet", which are trained on English text, or "multilingual-mpnet" and "multilingual-e5", which place works of many languages in one space. See Details.
text	Character; which text field(s) to embed. One of "title_abstract" (default, title and abstract together), "title", or "abstract".
batch_size	Integer; batch size for the transformer model. Defaults to 32L.
cache_dir	Character; directory to cache downloaded models. Defaults to the scimapR user cache directory.
python	Character or NULL; path to the Python executable. If NULL, uses **reticulate**'s default Python.
install_deps	Logical; if TRUE, install the Python packages the chosen model needs with pip first: torch and adapters (which brings transformers) for "specter2", sentence-transformers for the other models. Defaults to FALSE.
verbose	Logical; if TRUE, print progress messages.
call	Caller environment for error reporting.

Details

Models and their Hugging Face models:

- "specter2": allenai/specter2, an adapter on allenai/specter2_base;
- "scincl": malteos/scincl;
- "scibert": allenai/scibert_scivocab_uncased;
- "minilm-l6": sentence-transformers/all-MiniLM-L6-v2;
- "mpnet": sentence-transformers/all-mpnet-base-v2;
- "multilingual-mpnet": sentence-transformers/paraphrase-multilingual-mpnet-base-v2, trained on more than 50 languages;

- "multilingual-e5": intfloat/multilingual-e5-base, trained on about 100 languages.

The first five are trained on English text. "specter2" loads with the Python package adapters, the others with sentence-transformers.

"specter2" follows its model card: the base model with the proximity adapter, title and abstract joined by the separator token [SEP] without spaces, at most 512 tokens, and the first token of the last layer as the embedding. It needs the Python packages torch, transformers and adapters; the other models need sentence-transformers. A missing package stops the function with an error naming it, and so does a model that does not load; it never falls back to another model.

The other models join title and abstract with a space. For "multilingual-e5" every text starts with "query: ", the prefix its model card gives for clustering and for using the embeddings as features; its embeddings have unit length. "multilingual-mpnet" reads at most 128 tokens, so it sees the title and the start of a long abstract only. "scibert" has no sentence-transformers configuration, so sentence-transformers averages its token vectors, with a warning.

When more than 10% of the works with a known language (works$language: names such as "German", ISO 639-1 codes such as "de" or ISO 639-2 codes such as "ger") are not in English and an English-only model is chosen, a message (class scimapR_embed_multilingual) suggests a multilingual model; it changes nothing else.

Works without text to embed (with "title_abstract" neither a title nor an abstract, with "title" or "abstract" no such field; blank text counts as none) would all get the same vector and become each other's nearest neighbours. They are left out, with one warning (class scimapR_embed_no_text) giving their number: they get no row in corpus$embeddings, so the clustering functions give them no cluster (NA), and sm_network_semantic() and sm_plot_landscape() leave them out, as they do for works missing from a cache loaded with sm_embed_load().

corpus$metadata$embedding records what produced the embeddings: the model, its Hugging Face id and, when the local model snapshot in cache_dir shows it, its revision, the text used, the input prefix and the versions of the Python packages (see sm_corpus()). sm_certificate() reports it in embedding_info.

Python and its packages are soft dependencies (**reticulate** is in Suggests). If they are unavailable, the function aborts with an informative message.

Value

The input corpus with corpus$embeddings replaced by a numeric matrix with one row per embedded work and one column per dimension, and with corpus$metadata$embedding recording the model. Row names are the work_ids of the embedded works: every work that has text to embed (see Details).

See Also

Other embedding: sm_embed_load(), sm_embed_save()

Examples

```
## Not run:
corpus <- sm_example_corpus(with_embeddings = FALSE)
corpus <- sm_embed_works(corpus, model = "minilm-l6")
dim(corpus$embeddings)
```

```
corpus$metadata$embedding$model_id

# a corpus with works in several languages
corpus <- sm_embed_works(corpus, model = "multilingual-mpnet")

## End(Not run)
```

`sm_enrich_altmetric` *Enrich corpus with Altmetric attention data*

Description

Look up DOIs in the corpus via the Altmetric API and add attention scores, Mendeley reader counts, and social media metrics to the `works` table.

This enricher is idempotent: re-running it updates existing Altmetric data rather than duplicating it.

Usage

```
sm_enrich_altmetric(
  corpus,
  api_key = Sys.getenv("ALTMETRIC_API_KEY"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus` object.
`api_key`	Altmetric API key. Read from `ALTMETRIC_API_KEY` env var. The free tier (no key) is used when empty.
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object with added `altmetric_score`, `mendeley_readers`, `twitter_count`, and `news_count` columns in the `works` table, plus new provenance rows.

See Also

Other enrichers: `sm_enrich_concepts()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_orcid()`, `sm_enrich_retraction()`, `sm_enrich_ror()`, `sm_enrich_specter()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_altmetric(corpus)

## End(Not run)
```

`sm_enrich_concepts` *Enrich corpus with concepts from OpenAlex or MeSH*

Description

Look up works in the corpus via OpenAlex or PubMed to retrieve associated concepts, topics, or MeSH terms and add them to the `concepts` table.

This enricher is idempotent: re-running it updates existing concept data rather than duplicating rows.

Usage

```
sm_enrich_concepts(
  corpus,
  source = c("openalex", "mesh"),
  verbose = TRUE,
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus` object.
`source`	One of `"openalex"` (fetch OpenAlex concepts via DOI) or `"mesh"` (fetch MeSH terms via PubMed PMID; the concept `score` is 1 for a major topic of the article and 0 otherwise).
`verbose`	Print progress messages?
`api_key`	OpenAlex API key, sent in the authorization header.
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object with updated `concepts` table and new provenance rows for enriched works. Failed OpenAlex lookups warn and stop at a rate limit, preserving earlier successful results.

See Also

Other enrichers: `sm_enrich_altmetric()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_orcid()`, `sm_enrich_retraction()`, `sm_enrich_ror()`, `sm_enrich_specter()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_concepts(corpus, source = "openalex")

## End(Not run)
```

`sm_enrich_openalex_impact` *Add OpenAlex impact fields to any corpus*

Description

Looks the works of a corpus up on OpenAlex by identifier and adds OpenAlex's impact fields: the field-weighted citation impact (`fwci`), the citation percentile (`citation_percentile`, 0 to 1, with the flags `is_top_10pct` and `is_top_1pct`), the date of these values (`citations_as_of`) and the yearly citation counts (the corpus element `citations_by_year`). The corpus can come from any source, such as Web of Science files or `sm_corpus_from_tables()`; only identifiers are sent.

Usage

```
sm_enrich_openalex_impact(
  corpus,
  by = c("openalex_id", "doi"),
  batch_size = 50L,
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus`.
`by`	The identifiers to look works up by, in order of preference: one or both of `"openalex_id"` and `"doi"`. The default tries the OpenAlex id first and the DOI for works without one, as in corpora from Web of Science files, whose `openalex_id` is `NA`.
`batch_size`	Number of identifiers per request, from 1 to 100 (default `50`). Larger values are lowered to 100 with a message.
`api_key`	OpenAlex API key. Read from the `OPENALEX_API_KEY` environment variable by default; see "API key and rate limits".
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Details

Each work is looked up by the first identifier of by that it has: its OpenAlex id (`works$openalex_id`, shortened to the `W` number) or its DOI (normalised on both sides, so `https://doi.org/` prefixes and case do not matter). Works without either are counted and left as they are. A DOI that contains `,` or `|` cannot be written into an OpenAlex filter; such works are left out and counted too. The identifiers are sent in batches of `batch_size` (at most 100, the number of values an OpenAlex filter takes), with only the fields needed (`select`). When OpenAlex has several records for one DOI, the record with the most yearly citations is used, then the one with the lowest OpenAlex id.

The fields are joined with `sm_materialise()` (`overwrite = TRUE`). All five values of a work found come from its OpenAlex record, `NA` where the record has none, so a new run replaces the values of an earlier run or of another source, and the values, their date (`citations_as_of`) and the yearly counts of a work always belong to one record. The works not found keep their values. Only these five `works` columns change; `cited_by_count` and the other columns keep the values of their source. The `citations_by_year` rows of every work found are replaced by the counts OpenAlex has now; the rows of the other works stay. `metadata$last_refresh` is not changed.

Every work found gets a provenance row with the source `"openalex_impact"`, its OpenAlex id in `source_id_external`, a short description of the call in `query` and the kind `"enrichment"`, so `sm_rebuild_from_cert()` never replays the step as a search. Neither the API key nor any request URL or filter is stored.

On HTTP 429 (the request budget is used up) the function stops at once and joins nothing, not even the batches fetched before: a partial enrichment would compare works with OpenAlex values against works without them. The error (class `scimapR_openalex_rate_limit`) names the reset time, the remaining budget OpenAlex reports and the key setting. Any other failed request stops it too (class `scimapR_openalex_failed`).

Value

The `corpus` with the `works` columns `fwci`, `citation_percentile`, `is_top_10pct`, `is_top_1pct` and `citations_as_of` (see `sm_corpus()`), the element `citations_by_year` and a provenance row for each work found. The columns and the element are added even when OpenAlex has none of the works looked up (with `NA` and no rows where the corpus had none). A corpus without works, or without works that have an identifier, is returned unchanged.

API key and rate limits

OpenAlex expects a free API key. Requests without one share one small daily budget with everyone on the same network address; once it is used up, OpenAlex answers HTTP 429 (too many requests) until the budget resets at midnight UTC. Set the key in the `OPENALEX_API_KEY` environment variable (for example in `~/.Renviron`) or pass it as `api_key`. It is sent in the `Authorization` header, never in the URL, and it is stored nowhere in the corpus.

On HTTP 429 the function stops at once with an error of class `scimapR_openalex_rate_limit` instead of waiting: the message gives the reset time and the remaining budget that OpenAlex reports (the fields `reset_at` and `remaining` of the condition) and says whether a key was sent. Only a wait of up to two seconds (OpenAlex also limits requests per second) is waited out, once. Other failed requests stop with class `scimapR_openalex_failed`; both classes inherit from `scimapR_openalex_error`. With `engine = "openalexR"` the **openalexR** package (version 3.1.1 or later) sends the requests: it

gets the key too and its own retries are switched off, but it does not pass on the response headers, so the reset time and the budget are not known.

See Also

sm_fetch_openalex(), sm_citation_window()

Other enrichers: sm_enrich_altmetric(), sm_enrich_concepts(), sm_enrich_opencitations(), sm_enrich_orcid(), sm_enrich_retraction(), sm_enrich_ror(), sm_enrich_specter(), sm_enrich_unpaywall()

Examples

```
## Not run:
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
fix <- sm_enrich_openalex_impact(fix, by = "doi")
sm_citation_window(fix, window = 3)

## End(Not run)
```

---

sm_enrich_opencitations

Enrich corpus with OpenCitations citation data

Description

Look up DOIs using the OpenCitations Index API and add outgoing references. DOI and OpenAlex identifiers link these references to works in the corpus. The separate incoming citation count replaces `works$cited_by_count`.

This enricher is idempotent: re-running it updates existing citation data rather than duplicating it.

Usage

```
sm_enrich_opencitations(corpus, verbose = TRUE, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object with updated cited_by_count in the works table, outgoing references linked where possible, and new provenance rows for works with usable responses.

See Also

Other enrichers: sm_enrich_altmetric(), sm_enrich_concepts(), sm_enrich_openalex_impact(), sm_enrich_orcid(), sm_enrich_retraction(), sm_enrich_ror(), sm_enrich_specter(), sm_enrich_unpaywall()

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_opencitations(corpus)

## End(Not run)
```

`sm_enrich_orcid` *Enrich corpus authors with ORCID data*

Description

Look up authors in the corpus who have ORCID identifiers and fetch additional profile information from the ORCID Public API. Updates the `authors` table with display name alternatives and the `authorships` table with affiliation data.

This enricher is idempotent: re-running it updates existing ORCID data rather than duplicating it.

Usage

```
sm_enrich_orcid(corpus, verbose = TRUE, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus` object.
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object with updated `authors` and `authorships` tables, plus new provenance rows.

See Also

Other enrichers: `sm_enrich_altmetric()`, `sm_enrich_concepts()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_retraction()`, `sm_enrich_ror()`, `sm_enrich_specter()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_orcid(corpus)

## End(Not run)
```

`sm_enrich_retraction` *Enrich corpus with retraction data*

Description

Check works in the corpus against the Retraction Watch database (via the Crossref update notices and OpenAlex) to identify retracted publications.

Updates the `is_retracted` and `retraction_date` columns in the `works` table.

This enricher is idempotent: re-running it updates existing retraction data rather than duplicating it.

Usage

```
sm_enrich_retraction(
  corpus,
  verbose = TRUE,
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus` object.
`verbose`	Print progress messages?
`api_key`	OpenAlex API key, sent in the authorization header.
`mailto`	Contact address sent only to Crossref.
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object with updated `is_retracted` and `retraction_date` columns in the `works` table, plus new provenance rows for checked works only. Failed requests warn; when no work could be checked they error without changing the corpus. Missing records are reported separately. Existing retraction flags are never cleared.

See Also

Other enrichers: `sm_enrich_altmetric()`, `sm_enrich_concepts()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_orcid()`, `sm_enrich_ror()`, `sm_enrich_specter()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_retraction(corpus)

## End(Not run)
```

`sm_enrich_ror` *Enrich corpus institutions with ROR data*

Description

Look up institutions in the corpus via the Research Organization Registry (ROR) API and enrich the `institutions` table with standardized metadata including country, region, type, and income tier.

Works by matching raw affiliation strings against the ROR affiliation matching API, or by directly resolving existing ROR identifiers.

This enricher is idempotent: re-running it updates existing ROR data rather than duplicating it.

Usage

```
sm_enrich_ror(corpus, verbose = TRUE, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus` object.
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object with updated `institutions` and `authorships` tables, plus new provenance rows.

See Also

Other enrichers: `sm_enrich_altmetric()`, `sm_enrich_concepts()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_orcid()`, `sm_enrich_retraction()`, `sm_enrich_specter()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_ror(corpus)

## End(Not run)
```

`sm_enrich_specter` *Enrich corpus with SPECTER embeddings*

Description

Fetch pre-computed SPECTER v2 embeddings from the Semantic Scholar API for works in the corpus and attach them as the `embeddings` matrix.

SPECTER embeddings are 768-dimensional vectors useful for computing document similarity, clustering, and visualisation. No Python installation is required – this function retrieves pre-computed vectors from the API.

This enricher is idempotent: works that already have embeddings from the same source are skipped, so a second call fills only the gaps.

Usage

```
sm_enrich_specter(
  corpus,
  api_key = Sys.getenv("SEMANTIC_SCHOLAR_API_KEY"),
  replace = FALSE,
  mix_models = FALSE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus` object.
`api_key`	Semantic Scholar API key. Read from `SEMANTIC_SCHOLAR_API_KEY` env var by default.
`replace`	Logical; replace existing embeddings of another model (or of an unrecorded one) by SPECTER v2 vectors? Defaults to `FALSE`. See Details.
`mix_models`	Logical; keep existing embeddings of another model (or of an unrecorded one) and add SPECTER v2 vectors for the other works? Defaults to `FALSE`. See Details.
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Details

The function asks Semantic Scholar for its SPECTER v2 vectors (the API field `embedding.specter_v2`; the API's default is SPECTER v1) and records them in `corpus$metadata$embedding` (see `sm_corpus()`) with the model id `"embedding.specter_v2"` and, as `revision`, the model version that Semantic Scholar reports. Semantic Scholar does not say which revision of the model or which text it used, so these vectors count as a model of their own, also next to embeddings from `sm_embed_works(model = "specter2")`.

A corpus that already has embeddings of another model, or of a model that is not recorded (such as embeddings made by earlier versions of scimapR), would get a matrix whose rows come from two models, and similarities between such rows mean nothing. The function therefore stops (class `scimapR_embedding_model_mismatch`) unless you choose: `replace = TRUE` discards the existing embeddings, with a message, and fetches SPECTER v2 vectors for every work with a DOI; `mix_models = TRUE` keeps them and adds SPECTER v2 vectors for the works without an embedding, and the record then names the model `"mixed"`. A mix needs embeddings of the same width.

Requests go out in batches of 100 DOIs. A batch that Semantic Scholar answers with HTTP 429 (too many requests) is tried once more after a short wait; a batch that still fails is skipped, and one warning (class `scimapR_specter_failed`) gives the number of failed batches.

Value

An `sm_corpus` object with an `embeddings` matrix, its record in `metadata$embedding`, plus new provenance rows of the kind `"enrichment"`, which `sm_rebuild_from_cert()` does not replay.

See Also

Other enrichers: `sm_enrich_altmetric()`, `sm_enrich_concepts()`, `sm_enrich_openalex_impact()`, `sm_enrich_opencitations()`, `sm_enrich_orcid()`, `sm_enrich_retraction()`, `sm_enrich_ror()`, `sm_enrich_unpaywall()`

Examples

```
## Not run:
corpus <- sm_example_corpus(with_embeddings = FALSE)
corpus <- sm_enrich_specter(corpus)
corpus$metadata$embedding$model_id

# a corpus with embeddings of another model
corpus <- sm_embed_works(corpus, model = "mpnet")
corpus <- sm_enrich_specter(corpus, replace = TRUE)

## End(Not run)
```

---

sm_enrich_unpaywall *Enrich corpus with Unpaywall open-access data*

Description

Look up DOIs in the corpus via the Unpaywall API and add open-access status and best OA URL to the `works` table.

This enricher is idempotent: re-running it updates existing OA data rather than duplicating it. Requires a `mailto` address for API access.

Usage

```
sm_enrich_unpaywall(
  corpus,
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
mailto	Email address for the Unpaywall API. Read from SCIMAPR_MAILTO env var by default.
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object with updated oa_status and oa_url columns in the works table, plus new provenance rows.

See Also

Other enrichers: sm_enrich_altmetric(), sm_enrich_concepts(), sm_enrich_openalex_impact(), sm_enrich_opencitations(), sm_enrich_orcid(), sm_enrich_retraction(), sm_enrich_ror(), sm_enrich_specter()

Examples

```
## Not run:
corpus <- sm_example_corpus()
corpus <- sm_enrich_unpaywall(corpus)

## End(Not run)
```

---

sm_example_corpus *Generate a synthetic example corpus*

Description

Creates a realistic synthetic corpus for examples, testing, and demonstration. Works have plausible titles, DOIs, years, and citations. Authors are clustered into collaborating groups.

Usage

```
sm_example_corpus(
  n_works = 200L,
  n_authors = 80L,
  year_range = c(2015L, 2024L),
  n_clusters = 5L,
  with_embeddings = TRUE,
  with_screening = FALSE,
  with_trajectory_seed = TRUE,
  seed = 42L
)
```

Arguments

`n_works`	Number of works to generate.
`n_authors`	Number of authors to generate.
`year_range`	Two-element integer vector of year range.
`n_clusters`	Number of topic clusters.
`with_embeddings`	Logical; generate random embeddings?
`with_screening`	Logical; generate screening decisions?
`with_trajectory_seed`	Logical; create a prolific author for trajectory demonstration?
`seed`	Random seed for reproducibility. `NULL` chooses a fresh seed. The session random-number state is restored after generation.

Value

An `sm_corpus` object.

See Also

Other example: `sm_example_files()`

Examples

```
corpus <- sm_example_corpus()
print(corpus)
```

sm_example_db *Bundled example corpus*

Description

A pre-built synthetic corpus with 200 works, 80 authors, 5 clusters, and embeddings. Created with `sm_example_corpus(seed = 42)`.

Usage

```
sm_example_db
```

Format

An `sm_corpus` S3 object.

Source

Synthetically generated.

Examples

```
data(sm_example_db)
print(sm_example_db)
```

sm_example_files *Get paths to example data files*

Description

Returns the path to bundled example bibliographic files in the package's `inst/extdata` directory.

Usage

```
sm_example_files(name = NULL)
```

Arguments

`name`	Name of the example file (without path). If `NULL`, lists all available files.

Value

A character string path, or a character vector of available files.

Real-use fixture

Three files form a synthetic fixture shaped like a corpus that was built from a Web of Science export of three military-medical units and their civilian partners. Everything in it is invented: units, cities, people, journals, titles and DOIs (`10.9999/...`). The script `data-raw/realuse-fixture.R` in the package sources rebuilds the files.

`example_realuse_corpus.rds` An sm_corpus built with `sm_corpus_from_tables()`: 150 works from 2016 to 2026, with authors, Web of Science style addresses (ten of them German strings with non-ASCII letters), one corresponding author per work, sources, Web of Science style cited references and keyword and category concepts. Extra `works` columns are `quartile` (`NA` for every 2026 work), `fwci`, `field` (the first Web of Science category), `volume`, `first_page` and `citations_as_of`; `work_id` holds the Web of Science accession number. `citations_by_year` holds yearly citation counts shaped like OpenAlex's, dated 2026-10-17 (`citations_as_of`) for the works with a DOI: the years 2017 to 2026, without the years that have no citations (see `sm_citation_window()`). `metadata$last_refresh` is a numeric time stamp in seconds since 1970-01-01 UTC.

`example_realuse_reference.csv` A reference list in the style of a publication tracker, with columns `id`, `doi`, `title`, `year`, `institution` (the reporting unit) and `category` (a publication class code). It holds 120 of the corpus works and 40 items that are not in the corpus.

`example_realuse_affiliation_dict.csv` An affiliation dictionary in the sm_affiliation_dict format (`institution`, `pattern`, `email_domain`) for `sm_affiliation_match()`: institutes first, the hospital patterns anchored on its city next, a generic fallback last. Its institution names are the `institution` values of the reference list.

The CSV files are UTF-8 encoded.

See Also

Other example: `sm_example_corpus()`

Examples

```
sm_example_files()
sm_example_files("example.bib")

# The real-use fixture
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
ref <- readr::read_csv(sm_example_files("example_realuse_reference.csv"),
                       col_types = "cccicc")
dict <- readr::read_csv(
  sm_example_files("example_realuse_affiliation_dict.csv"),
  col_types = "ccc"
)
matched <- sm_affiliation_match(fix, dict)
table(matched$authorships$institution_match, useNA = "ifany")
```

sm_export_covidence *Export corpus for Covidence*

Description

Write corpus works in RIS format for Covidence import.

Usage

```
sm_export_covidence(corpus, path)
```

Arguments

corpus	An sm_corpus.
path	Output file path.

Value

path invisibly.

See Also

Other screening: sm_export_rayyan(), sm_import_rayyan(), sm_merge_screening_decisions(), sm_screen_prisma()

sm_export_csv *Export corpus tables as CSV files*

Description

Write each corpus table as a separate CSV file in a directory, the yearly citation counts (citations_by_year.csv) too when the corpus has them.

Usage

```
sm_export_csv(corpus, dir, ...)
```

Arguments

corpus	An sm_corpus object.
dir	Output directory.
...	Additional arguments passed to readr::write_csv().

Value

dir invisibly.

See Also

Other export: sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_rds(), sm_export_table(), sm_export_vosviewer(), sm_export_zip()

Examples

```
corpus <- sm_example_corpus(n_works = 10)
dir <- tempdir()
sm_export_csv(corpus, dir)
```

---

sm_export_cytoscape *Export network for Cytoscape (JSON)*

Description

Write network to Cytoscape JSON format.

Usage

```
sm_export_cytoscape(network, path)
```

Arguments

network	A tbl_graph object.
path	Output file path.

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_rds(), sm_export_table(), sm_export_vosviewer(), sm_export_zip()

Examples

```
corpus <- sm_example_corpus()
net <- sm_network_cocitation(corpus)
sm_export_cytoscape(net, tempfile(fileext = ".json"))
```

`sm_export_figure` *Export a plot as a publication-ready figure*

Description

Save a ggplot to disk in multiple formats and resolutions. By default, raster formats are saved at both 300 and 600 dpi.

Usage

```
sm_export_figure(
  plot,
  path,
  format = c("png", "pdf", "svg", "tiff", "eps"),
  dpi = c(300, 600),
  width = 7,
  height = 5,
  units = c("in", "cm", "mm"),
  background = c("white", "transparent"),
  multi_dpi = TRUE
)
```

Arguments

`plot`	A ggplot object.
`path`	Output file path (extension determines format if `format` not specified).
`format`	Output format.
`dpi`	Resolution for raster formats.
`width`	Plot width.
`height`	Plot height.
`units`	Size units.
`background`	Background colour.
`multi_dpi`	Logical; write both 300 and 600 dpi versions for raster formats?

Value

Character vector of paths actually written, invisibly.

See Also

Other export: `sm_export_csv()`, `sm_export_cytoscape()`, `sm_export_gephi()`, `sm_export_quarto_report()`, `sm_export_rds()`, `sm_export_table()`, `sm_export_vosviewer()`, `sm_export_zip()`

Examples

```
corpus <- sm_example_corpus(n_works = 20)
p <- sm_plot_production(corpus)
path <- tempfile(fileext = ".png")
sm_export_figure(p, path, multi_dpi = FALSE)
```

`sm_export_gephi` *Export network for Gephi (GEXF format)*

Description

Write a tidygraph network to GEXF XML format for use with Gephi. Labels are escaped as XML, and directed networks retain their direction.

Usage

```
sm_export_gephi(network, path)
```

Arguments

`network`	A `tbl_graph` object.
`path`	Output file path (should end in `.gexf`).

Value

`path` invisibly.

See Also

Other export: `sm_export_csv()`, `sm_export_cytoscape()`, `sm_export_figure()`, `sm_export_quarto_report()`, `sm_export_rds()`, `sm_export_table()`, `sm_export_vosviewer()`, `sm_export_zip()`

Examples

```
corpus <- sm_example_corpus()
net <- sm_network_cocitation(corpus)
sm_export_gephi(net, tempfile(fileext = ".gexf"))
```

sm_export_quarto_report *Export a Quarto report*

Description

Generate a comprehensive bibliometric report from a corpus using a Quarto template.

Usage

```
sm_export_quarto_report(
  corpus,
  path,
  template = c("standard", "minimal", "thesis"),
  include_chat = FALSE,
  include_audit = TRUE,
  include_trajectory = FALSE
)
```

Arguments

corpus	An sm_corpus.
path	Output path for the rendered report.
template	Report template.
include_chat	Logical; include chat session?
include_audit	Logical; include equity audit?
include_trajectory	Logical; include trajectory analysis?

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_rds(), sm_export_table(), sm_export_vosviewer(), sm_export_zip()

Examples

```
corpus <- sm_example_corpus()
sm_export_quarto_report(corpus, tempfile(fileext = ".html"))
```

`sm_export_rayyan` *Export corpus for Rayyan*

Description

Write corpus works in RIS format for import into Rayyan. Work IDs appear in the RIS ID field and a note; missing fields are omitted.

Usage

```
sm_export_rayyan(corpus, path)
```

Arguments

`corpus`	An `sm_corpus`.
`path`	Output file path.

Details

Keep the ID or the scimapR work_id note when exporting decisions.

Value

`path` invisibly.

See Also

Other screening: `sm_export_covidence()`, `sm_import_rayyan()`, `sm_merge_screening_decisions()`, `sm_screen_prisma()`

Examples

```
path <- tempfile(fileext = ".ris")
sm_export_rayyan(sm_example_corpus(n_works = 2), path)
readLines(path, n = 3)
unlink(path)
```

sm_export_rds *Export corpus as RDS*

Description

Save the full corpus including embeddings as a compressed RDS file.

Usage

```
sm_export_rds(corpus, path, compress = "xz")
```

Arguments

corpus	An sm_corpus object.
path	Output file path.
compress	Compression type. Default "xz".

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_table(), sm_export_vosviewer(), sm_export_zip()

Examples

```
corpus <- sm_example_corpus(n_works = 10)
path <- tempfile(fileext = ".rds")
sm_export_rds(corpus, path)
```

sm_export_table *Export a table as formatted XLSX or CSV*

Description

Save a tibble or data frame with publication-ready formatting. XLSX output includes bold headers, frozen top row, autofilter, and optional title/caption/source note.

Usage

```
sm_export_table(
  table,
  path,
  format = c("xlsx", "csv", "tsv"),
  title = NULL,
  caption = NULL,
  source_note = NULL,
  style = c("scimapr", "minimal", "publication")
)
```

Arguments

table	A data frame or tibble.
path	Output file path.
format	Output format.
title	Optional title row (merged, larger font).
caption	Optional caption row beneath header.
source_note	Optional source note in footer.
style	Formatting style.

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_rds(), sm_export_vosviewer(), sm_export_zip()

Examples

```
dat <- data.frame(Author = "Smith J", Works = 10L, Citations = 150L)
path <- tempfile(fileext = ".xlsx")
sm_export_table(dat, path)
```

---

sm_export_vosviewer *Export network for VOSviewer*

Description

Write a headerless network file with integer node ids and positive link strengths, plus a tab-separated map file with id and label columns. Missing weights become 1. Directed edges are written as supplied, but VOSviewer treats the links as undirected.

Usage

```
sm_export_vosviewer(network, path, map_path = NULL)
```

Arguments

network	A tbl_graph object.
path	Output file path.
map_path	Map file path. By default, _map is inserted before the extension of path, or appended if path has no extension.

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_rds(), sm_export_table(), sm_export_zip()

Examples

```
corpus <- sm_example_corpus()
net <- sm_network_cocitation(corpus)
sm_export_vosviewer(net, tempfile(fileext = ".tsv"))
# The matching map file is written beside the network file.
```

---

sm_export_zip *Export corpus as self-contained ZIP bundle*

Description

Build a ZIP archive containing the corpus RDS, certificate, figures, tables, and an auto-generated README. This is the "send everything to a collaborator" workflow.

Usage

```
sm_export_zip(
  corpus,
  path,
  include = c("rds", "certificate", "tables"),
  figure_formats = c("png"),
  figure_dpi = c(300),
  report_template = "standard"
)
```

Arguments

corpus	An sm_corpus object.
path	Output ZIP file path.
include	Components to include in the bundle.
figure_formats	Figure output formats.
figure_dpi	Figure resolutions.
report_template	Quarto report template.

Value

path invisibly.

See Also

Other export: sm_export_csv(), sm_export_cytoscape(), sm_export_figure(), sm_export_gephi(), sm_export_quarto_report(), sm_export_rds(), sm_export_table(), sm_export_vosviewer()

Examples

```
corpus <- sm_example_corpus(n_works = 10)
path <- tempfile(fileext = ".zip")
sm_export_zip(corpus, path, include = c("rds", "certificate"))
```

---

sm_fetch_arxiv *Fetch works from arXiv*

Description

Query the arXiv API (Atom feed) for preprints and return the results as an sm_corpus.

Uses start/max_results pagination to retrieve up to n_max results. The arXiv API is free and requires no authentication.

Usage

```
sm_fetch_arxiv(query, n_max = 200L, verbose = TRUE, call = rlang::caller_env())
```

Arguments

query	arXiv search query string using the arXiv query syntax. Supports all:, ti:, au:, abs:, cat:, etc.
n_max	Maximum number of results to return (default 200).
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

See Also

Other fetchers: sm_fetch_biorxiv(), sm_fetch_crossref(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_pubmed(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_arxiv(query = "all:bibliometrics", n_max = 10)
print(corpus)

## End(Not run)
```

---

sm_fetch_biorxiv *Fetch works from bioRxiv/medRxiv*

Description

Query the bioRxiv or medRxiv API for preprints and return the results as an sm_corpus.

The API provides date-range-based content retrieval. If query is supplied, results are filtered locally by title/abstract matching.

Usage

```
sm_fetch_biorxiv(
  query = NULL,
  server = c("biorxiv", "medrxiv"),
  from_date = NULL,
  to_date = NULL,
  n_max = 200L,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

query	Optional search string for local filtering of results.
server	One of "biorxiv" or "medrxiv".
from_date	Start date in "YYYY-MM-DD" format. Defaults to 30 days ago.
to_date	End date in "YYYY-MM-DD" format. Defaults to today.
n_max	Maximum number of results to return (default 200).
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Failed requests

Native requests retry transient server and network failures, and short announced rate-limit waits. Long waits abort immediately. If a later page fails, the error inherits from scimapR_fetch_incomplete and holds the earlier parsed works in rlang::last_error()$partial. Their metadata records fetch_incomplete with the source, received and expected counts, and failure reason. If those pages cannot be parsed, partial is NULL. The default remains an error; an incomplete fetch is never returned as an ordinary successful result.

See Also

Other fetchers: sm_fetch_arxiv(), sm_fetch_crossref(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_pubmed(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_biorxiv(
  from_date = "2024-01-01",
  to_date = "2024-01-07",
  n_max = 10
)
print(corpus)

## End(Not run)
```

---

sm_fetch_crossref *Fetch works from Crossref*

Description

Query the Crossref API for scholarly works and return the results as an sm_corpus.

Uses cursor pagination to retrieve up to n_max works. Providing a mailto address enables the polite pool (faster rate limits).

Usage

```
sm_fetch_crossref(
  query = NULL,
  filter = NULL,
  n_max = 200L,
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  engine = c("native", "rcrossref", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

query	Free-text search query. If NULL, filter must be supplied.
filter	Crossref filter syntax, e.g. "from-pub-date:2020,type:journal-article".
n_max	Maximum number of works to return (default 200).
mailto	Email address for the polite pool. Read from SCIMAPR_MAILTO env var by default.
engine	One of "native" (built-in httr2 client), "rcrossref" (use the rcrossref package), or "auto" (use rcrossref if available, otherwise native).
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object. Its provenance records the search (query, filter, n_max and engine), so that sm_rebuild_from_cert() can run it again.

Failed requests

Native requests retry transient server and network failures, and short announced rate-limit waits. Long waits abort immediately. If a later page fails, the error inherits from scimapR_fetch_incomplete and holds the earlier parsed works in rlang::last_error()$partial. Their metadata records fetch_incomplete with the source, received and expected counts, and failure reason. If those pages cannot be parsed, partial is NULL. The default remains an error; an incomplete fetch is never returned as an ordinary successful result.

See Also

Other fetchers: sm_fetch_arxiv(), sm_fetch_biorxiv(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_pubmed(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_crossref(query = "bibliometrics", n_max = 10)
print(corpus)

## End(Not run)
```

`sm_fetch_openalex` *Fetch works from OpenAlex*

Description

Query the OpenAlex API for scholarly works and return the results as an `sm_corpus`.

Supports both free-text search (`query`) and structured filter syntax (`filter`). Uses cursor-based pagination to retrieve up to `n_max` works. Set an OpenAlex API key (see "API key and rate limits").

Besides `cited_by_count`, the works keep OpenAlex's field-weighted citation impact (`fwci`), its citation percentile (`citation_percentile`, 0 to 1, with the flags `is_top_10pct` and `is_top_1pct`) and the date of these values (`citations_as_of`), and the corpus gets the yearly citation counts as `citations_by_year` (see `sm_corpus()` and `sm_citation_window()`). The openalexR engine has no percentile, so its three percentile columns are `NA`. To add these fields to a corpus from another source, use `sm_enrich_openalex_impact()`.

Usage

```
sm_fetch_openalex(
  query = NULL,
  filter = NULL,
  n_max = 200L,
  per_page = 200L,
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  engine = c("native", "openalexR", "auto"),
  batch_size = 50L,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`query`	Free-text search query passed to the `search` parameter. If `NULL`, `filter` must be supplied.
`filter`	A character string using OpenAlex filter syntax, e.g. `"from_publication_date:2020-01-01,type:jou`
`n_max`	Maximum number of works to return (default 200).
`per_page`	Number of results per page (max 200).
`mailto`	Contact address sent with each request as the `mailto` parameter. Read from the `SCIMAPR_MAILTO` environment variable by default. OpenAlex no longer gives such requests a larger budget (the API key does), and the openalexR engine does not send it.
`api_key`	OpenAlex API key. Read from the `OPENALEX_API_KEY` environment variable by default; see "API key and rate limits".

engine	One of "native" (built-in httr2 client), "openalexR" (use the openalexR package), or "auto" (use openalexR if version 3.1.1 or later is installed, otherwise native).
batch_size	Integer; maximum number of |-joined values allowed in a single OpenAlex filter clause before the request is automatically split into multiple batched requests. OpenAlex limits the number of values in an OR filter and the overall URL length, so long DOI (or other ID) lists are chunked, fetched per batch, and row-bound with de-duplication. Default 50, which stays comfortably under the API's documented limit of 100.
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object. Its provenance records the search (query, filter, n_max and engine), so that sm_rebuild_from_cert() can run it again.

API key and rate limits

OpenAlex expects a free API key. Requests without one share one small daily budget with everyone on the same network address; once it is used up, OpenAlex answers HTTP 429 (too many requests) until the budget resets at midnight UTC. Set the key in the OPENALEX_API_KEY environment variable (for example in ~/.Renviron) or pass it as api_key. It is sent in the Authorization header, never in the URL, and it is stored nowhere in the corpus.

On HTTP 429 the function stops at once with an error of class scimapR_openalex_rate_limit instead of waiting: the message gives the reset time and the remaining budget that OpenAlex reports (the fields reset_at and remaining of the condition) and says whether a key was sent. Only a wait of up to two seconds (OpenAlex also limits requests per second) is waited out, once. Other failed requests stop with class scimapR_openalex_failed; both classes inherit from scimapR_openalex_error. With engine = "openalexR" the **openalexR** package (version 3.1.1 or later) sends the requests: it gets the key too and its own retries are switched off, but it does not pass on the response headers, so the reset time and the budget are not known.

Failed requests

Native requests retry transient server and network failures, and short announced rate-limit waits. Long waits abort immediately. If a later page fails, the error inherits from scimapR_fetch_incomplete and holds the earlier parsed works in rlang::last_error()$partial. Their metadata records fetch_incomplete with the source, received and expected counts, and failure reason. If those pages cannot be parsed, partial is NULL. The default remains an error; an incomplete fetch is never returned as an ordinary successful result.

See Also

sm_enrich_openalex_impact(), sm_citation_window()

Other fetchers: sm_fetch_arxiv(), sm_fetch_biorxiv(), sm_fetch_crossref(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_pubmed(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_openalex(query = "bibliometrics", n_max = 10)
print(corpus)

## End(Not run)
```

`sm_fetch_orcid` *Fetch works from ORCID*

Description

Retrieve works associated with an ORCID identifier via the ORCID Public API and return the results as an `sm_corpus`.

This fetches work summaries from the public API endpoint `https://pub.orcid.org/v3.0/{orcid}/works`.

Usage

```
sm_fetch_orcid(orcid, verbose = TRUE, call = rlang::caller_env())
```

Arguments

`orcid`	An ORCID identifier (e.g., `"0000-0001-8006-9742"`).
`verbose`	Print progress messages?
`call`	Caller environment for error reporting.

Value

An `sm_corpus` object.

See Also

Other fetchers: `sm_fetch_arxiv()`, `sm_fetch_biorxiv()`, `sm_fetch_crossref()`, `sm_fetch_openalex()`, `sm_fetch_overton()`, `sm_fetch_pubmed()`, `sm_fetch_semantic_scholar()`

Examples

```
## Not run:
corpus <- sm_fetch_orcid("0000-0001-8006-9742")
print(corpus)

## End(Not run)
```

sm_fetch_overton *Fetch policy citations from Overton*

Description

Search the Overton API for policy document citations of scholarly works and return the results as an sm_corpus.

Requires an Overton API key (set via the OVERTON_API_KEY environment variable).

Usage

```
sm_fetch_overton(
  query,
  api_key = Sys.getenv("OVERTON_API_KEY"),
  n_max = 100L,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

query	Search query string.
api_key	Overton API key. Read from OVERTON_API_KEY env var.
n_max	Maximum number of results to return (default 100).
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

See Also

Other fetchers: sm_fetch_arxiv(), sm_fetch_biorxiv(), sm_fetch_crossref(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_pubmed(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_overton(query = "climate change policy", n_max = 10)
print(corpus)

## End(Not run)
```

sm_fetch_pubmed *Fetch works from PubMed*

Description

Query PubMed via the NCBI E-utilities (esearch + efetch) and return the results as an sm_corpus.

Uses retmax/retstart pagination to retrieve up to n_max records. Providing an NCBI API key allows up to 10 requests/second instead of 3.

MeSH headings become concepts in the "mesh" vocabulary. Their score is 1 for a major topic of the article (MajorTopicYN on the descriptor or one of its qualifiers) and 0 otherwise.

Usage

```
sm_fetch_pubmed(
  query,
  n_max = 200L,
  api_key = Sys.getenv("NCBI_API_KEY"),
  engine = c("native", "rentrez", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

query	PubMed search query string (required).
n_max	Maximum number of records to return (default 200).
api_key	NCBI API key. Read from NCBI_API_KEY env var by default.
engine	One of "native" (built-in httr2 client), "rentrez" (use the rentrez package), or "auto" (use rentrez if available, otherwise native).
verbose	Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object. Its provenance records the search (query, n_max and engine), so that sm_rebuild_from_cert() can run it again.

Failed requests

Native requests retry transient server and network failures, and short announced rate-limit waits. Long waits abort immediately. If a later page fails, the error inherits from scimapR_fetch_incomplete and holds the earlier parsed works in rlang::last_error()$partial. Their metadata records fetch_incomplete with the source, received and expected counts, and failure reason. If those pages cannot be parsed, partial is NULL. The default remains an error; an incomplete fetch is never returned as an ordinary successful result.

See Also

Other fetchers: sm_fetch_arxiv(), sm_fetch_biorxiv(), sm_fetch_crossref(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_semantic_scholar()

Examples

```
## Not run:
corpus <- sm_fetch_pubmed(query = "bibliometrics[tiab]", n_max = 10)
print(corpus)

## End(Not run)
```

---

sm_fetch_semantic_scholar

Fetch works from Semantic Scholar

Description

Query the Semantic Scholar Academic Graph API for papers and return the results as an sm_corpus.

Supports both keyword search (query) and batch paper lookup (paper_ids). Optionally fetches pre-computed SPECTER embeddings.

Usage

```
sm_fetch_semantic_scholar(
  query = NULL,
  paper_ids = NULL,
  n_max = 100L,
  api_key = Sys.getenv("SEMANTIC_SCHOLAR_API_KEY"),
  include_embeddings = FALSE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

query	Free-text search query. If NULL, paper_ids must be supplied.
paper_ids	Character vector of Semantic Scholar paper IDs, DOIs (prefixed DOI:), arXiv IDs (prefixed ARXIV:), or PMIDs (prefixed PMID:). If NULL, query must be supplied.
n_max	Maximum number of papers to return (default 100).
api_key	Semantic Scholar API key. Read from SEMANTIC_SCHOLAR_API_KEY env var by default.

include_embeddings
: Logical; if TRUE, request SPECTER embeddings for each paper and attach as an embedding matrix. These are the API's default SPECTER v1 vectors, recorded in metadata$embedding with the model id "embedding.specter_v1" (see sm_corpus()); sm_enrich_specter() fetches SPECTER v2 vectors.

verbose
: Print progress messages?

call
: Caller environment for error reporting.

Value

An sm_corpus object.

Failed requests

Native requests retry transient server and network failures, and short announced rate-limit waits. Long waits abort immediately. If a later page fails, the error inherits from scimapR_fetch_incomplete and holds the earlier parsed works in rlang::last_error()$partial. Their metadata records fetch_incomplete with the source, received and expected counts, and failure reason. If those pages cannot be parsed, partial is NULL. The default remains an error; an incomplete fetch is never returned as an ordinary successful result.

See Also

Other fetchers: sm_fetch_arxiv(), sm_fetch_biorxiv(), sm_fetch_crossref(), sm_fetch_openalex(), sm_fetch_orcid(), sm_fetch_overton(), sm_fetch_pubmed()

Examples

```
## Not run:
corpus <- sm_fetch_semantic_scholar(query = "bibliometrics", n_max = 10)
print(corpus)

## End(Not run)
```

---

sm_field_clinical_trials

Link corpus works to clinical trials

Description

Identify works in the corpus that are linked to clinical trials based on PMIDs or DOIs.

Usage

```
sm_field_clinical_trials(corpus, ..., call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus.
...	Additional arguments (currently unused).
call	Caller environment.

Value

A tibble with work_id and trial-related information.

See Also

Other field-helpers: sm_field_funder(), sm_field_pubmed_mesh()

Examples

```
corpus <- sm_example_corpus()
sm_field_clinical_trials(corpus)
```

---

sm_field_funder *Extract funding information*

Description

Summarise funding sources from corpus metadata.

Usage

```
sm_field_funder(
  corpus,
  source = c("crossref", "openalex"),
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus.
source	Data source for funding info.
call	Caller environment.

Value

A tibble of funders with counts.

See Also

Other field-helpers: sm_field_clinical_trials(), sm_field_pubmed_mesh()

Examples

```
corpus <- sm_example_corpus()
sm_field_funder(corpus)
```

`sm_field_pubmed_mesh` *Extract MeSH terms from corpus*

Description

Summarise MeSH (Medical Subject Headings) terms from the concepts table.

Usage

```
sm_field_pubmed_mesh(corpus, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus`.
`call`	Caller environment.

Value

A tibble with `mesh_term`, `count`, and `proportion`.

See Also

Other field-helpers: `sm_field_clinical_trials()`, `sm_field_funder()`

Examples

```
corpus <- sm_example_corpus()
sm_field_pubmed_mesh(corpus)
```

`sm_figure_manifest` *Build a figure caption and alt-text manifest*

Description

Scans a directory of figures and produces a tidy manifest of captions, alt-text, and (where readable) pixel dimensions and DPI. This replaces the hand-maintained `figure_captions.R` scripts that accompany many manuscripts.

Captions and alt-text are pulled from a sidecar file if one is present (`captions.csv`, `captions.yml`, or `captions.yaml` in `dir`), otherwise the columns are left empty for the user to fill in.

Usage

```
sm_figure_manifest(
  dir,
  pattern = NULL,
  output = NULL,
  call = rlang::caller_env()
)
```

Arguments

dir	Directory to scan.
pattern	Optional regular expression to filter file names. Defaults to common figure extensions (png, pdf, svg, tif, tiff, jpg, jpeg, eps).
output	Optional path to write the manifest to. A .csv extension writes CSV; .yml/.yaml writes YAML (requires the optional **yaml** package).
call	Caller environment for error reporting.

Value

A tibble with one row per figure file and columns file, caption, alt_text, width, height, dpi. Pixel dimensions / DPI are read from the image where feasible (using the optional **magick** or **png** packages) and are NA otherwise. Type-stable: a missing or empty directory returns a 0-row tibble (with a cli warning), never an error.

See Also

Other reporting: sm_corpus_from_tables()

Examples

```
d <- withr::local_tempdir()
ggplot2::ggsave(file.path(d, "fig1.png"),
                ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
                  ggplot2::geom_point(),
                width = 4, height = 3, dpi = 150)
sm_figure_manifest(d)
```

---

sm_filter_works *Filter works in a corpus*

Description

Subset a corpus by filtering works on year, type, OA status, or custom expressions.

Usage

```
sm_filter_works(corpus, ..., year_range = NULL, types = NULL, oa_only = FALSE)
```

Arguments

corpus	An sm_corpus object.
...	Filtering expressions passed to dplyr::filter(), evaluated in the context of the works tibble.
year_range	Optional two-element integer vector for year filtering.
types	Optional character vector of document types to keep.
oa_only	Logical; keep every status in sm_oa_statuses() except closed, including recognised export spellings. Missing and unrecognised statuses are dropped.

Value

An sm_corpus with the filtered subset.

See Also

Other filters: sm_query()

Examples

```
corpus <- sm_example_corpus()
filtered <- sm_filter_works(corpus, year_range = c(2020, 2024))
nrow(filtered$works)
```

---

sm_hash_corpus *Hash a corpus for reproducibility*

Description

Computes a content-addressable SHA-256 hash of the corpus. The hash covers the works (without last_refreshed), authors, authorships, institutions, sources, references, concepts, screening decisions (without decided_at) and embeddings, and the yearly citation counts (citations_by_year) when the corpus has any, so a corpus without them hashes as it did before the element existed.

Two corpora with identical scientific content will produce the same hash, regardless of when they were built and of the order of their rows and columns.

The hash leaves out the provenance table, which records when each record was fetched, and the metadata (timestamps, the lock, the embedding model and the local input files of sm_register_input()): changing them does not change the hash. sm_verify_certificate() checks the local input files on their own.

Usage

```
sm_hash_corpus(corpus, version = 2L)
```

Arguments

corpus	An sm_corpus object.
version	The hash algorithm: 2 (default) or 1 (scimapR 0.4.0 and earlier). See Details.

Details

version selects the algorithm. Version 2, the default from scimapR 0.5.0 on, hashes canonical UTF-8 text, independently of R's binary serialisation. Table columns are sorted by name and rows by their text values, using radix ordering. Missing values are distinct from the string "NA". Numbers use 15 significant digits, signed zero becomes zero, dates use ISO dates and timestamps use UTC with six decimal places for seconds. List cells use JSON. Column classes are included in the text. Embeddings are sorted by work id and rounded to eight decimal places. These rules make version 2 independent of the R version, platform, time zone and collation locale. Hashes of version 2 therefore differ from the hashes that earlier versions of scimapR computed for the same content. Version 1 is the algorithm of these earlier versions. Certificates (sm_certificate()) and snapshots (sm_snapshot()) record the version they used as hash_version; those written before it existed are checked with version 1. Version 1 uses R's binary serialisation and depends on the R version: older certificates and snapshots remain verifiable on the R version that wrote them.

Value

A single character string containing the hex-encoded SHA-256 hash.

See Also

Other reproducibility: sm_certificate(), sm_cite_corpus(), sm_diff_corpora(), sm_provenance(), sm_register_input(), sm_snapshot()

Examples

```
corpus <- sm_example_corpus()
sm_hash_corpus(corpus)

# the order of the columns does not matter
reordered <- corpus
reordered$works <- reordered$works[, rev(names(reordered$works))]
identical(sm_hash_corpus(reordered), sm_hash_corpus(corpus))
```

---

sm_import_rayyan *Import screening decisions from Rayyan*

Description

Read Rayyan CSV export and parse screening decisions.

Usage

```
sm_import_rayyan(path, corpus = NULL, id_col = NULL)
```

Arguments

path	Path to Rayyan CSV export.
corpus	Optional corpus for DOI matching and ID validation.
id_col	Optional name of the CSV column containing work IDs.

Details

IDs come from the named column, otherwise from the scimapR work_id note, or from unique matching corpus DOIs. Decisions and reasons come from their columns or Rayyan inclusion and exclusion notes. Disagreeing reviewers give an uncertain decision.

Value

One screening row per CSV record. Unmatched work IDs remain missing and warn. Decisions use include, exclude and uncertain.

See Also

Other screening: sm_export_covidence(), sm_export_rayyan(), sm_merge_screening_decisions(), sm_screen_prisma()

Examples

```
path <- tempfile(fileext = ".csv")
utils::write.csv(data.frame(id = "paper-a", decision = "Included"),
                 path, row.names = FALSE)
sm_import_rayyan(path, id_col = "id")
unlink(path)
```

---

sm_its *Interrupted time series for a corpus outcome*

Description

A turnkey interrupted time series (ITS): aggregates the chosen outcome to a yearly series, fits a segmented regression with a level shift and a slope change at intervention_year, and derives the counterfactual (the pre-intervention trend projected forward).

Usage

```
sm_its(
  corpus,
  intervention_year,
  outcome = c("count", "share_q1", "cnci", "leadership"),
  family = NULL,
  lag = NULL,
```

```
  institutions = NULL,
  lead_role = c("corresponding", "first", "last", "first_or_last", "any_lead"),
  min_cell = NULL,
  call = rlang::caller_env()
)

## S3 method for class 'sm_its'
print(x, ...)

## S3 method for class 'sm_its'
summary(object, ...)

## S3 method for class 'sm_its'
autoplot(object, ...)
```

Arguments

corpus	An sm_corpus.
intervention_year	Year at which the intervention takes effect: one whole number such as 2020, or text holding one.
outcome	One of "count", "share_q1", "cnci", "leadership". The expected source columns are: "count" Works per year (uses works$year). "share_q1" Share of Q1-journal works among the works with a quartile; needs a journal-quartile column on works (quartile, jif_quartile, or sjr_quartile). "cnci" Mean field-normalised citation impact per year. Impact is resolved from the first populated of works$cnci, works$fwci (OpenAlex; see sm_enrich_openalex_impact the other impact columns fnci, rcr, cnci_value, ncs, mncs and field_citation_ratio, a corpus$metrics table (cnci/fnci/value keyed by work_id), or works$cited_by_count (from which FNCI is derived via sm_metric_fnci(), with min_cell). If none is populated, an informative error (class scimapR_no_impact_data) names the columns inspected. sm_metric_summary() (metric = "cnci") resolves impact in the same order, and the result names the source used (impact_source). Works without a finite impact value are left out of the yearly mean. "leadership" Share of works led through the author role lead_role (by default the corresponding author, authorships$is_corresponding), by one of institutions when they are given; see Details.
family	Optional GLM family (a family object, generator function, or name). Defaults to Poisson for "count" and Gaussian otherwise; document your choice when overriding. A Poisson fit warns when the counts are overdispersed; "quasipoisson" then scales the standard errors.
lag	Citation-maturity lag in years (default 2). For citation-based outcomes ("cnci"), the most recent lag years are citation-immature and are excluded from the fit; this is reported in the print method.

institutions	For outcome = "leadership": NULL (default) or a character vector of focal institution labels. A work then counts as led when an authorship in the role lead_role belongs to one of them, that is when its institution_match, institution_name or institution_id, or the name of that id in corpus$institutions, is one of the labels (run sm_affiliation_match() first). A label that no authorship carries gives a warning naming the nearest labels; when none is found, sm_its() aborts.
lead_role	For outcome = "leadership": the author role that leads a work. "corresponding" (default) is an author with is_corresponding = TRUE, "first" and "last" are the authors with the lowest and the highest position of the work, "first_or_last" either, and "any_lead" the first, last or corresponding author. Roles other than "corresponding" need institutions.
min_cell	For outcome = "cnci": NULL (default) or a positive integer, passed to sm_metric_fnci() when the impact is derived from works$cited_by_count, so that works in field-year cells with fewer works fall back to larger baselines. Not used when the corpus carries stored impact values.
call	Caller environment for error reporting.
x	An sm_its object.
...	Ignored.
object	An sm_its object.

Details

The yearly series is built first. For "count" it runs over every year from the first to the last publication year of the corpus, and a year without works counts 0, so filter the corpus to the analysis window beforehand. For the other outcomes a year with works but no usable value is dropped with a warning and listed in dropped_years: no work of the year has a journal quartile ("share_q1") or a finite impact value ("cnci"). For "cnci" the citation-immature years are excluded after that and listed in provisional_years, so no year is in both.

For "leadership" the share of a year is over the works whose lead is known, and that number of works weights the year in the fit. A work whose authorships all have is_corresponding = NA (unknown, as the readers of sources without corresponding-author data record it; see sm_corpus()) leaves the denominator for lead_role = "corresponding", with or without institutions; up to scimapR 0.4.0 it counted as not led. So does a work without authorships, and for the position roles a work whose positions are all NA. For "any_lead" an unknown corresponding author counts as not leading, so the work stays in the denominator. A year in which no work has a known lead is dropped with a warning, and sm_its() aborts when no work has one. With institutions the denominator is every work of the year with a known lead, not only the works with an authorship at the focal institutions.

A focal label is compared with every label column of an authorship row (institution_match, institution_name, institution_id and the name of that id), as group membership in sm_did() and sm_synth() is, so a row with institution_match "A" and institution_name "B" belongs to both "A" and "B". sm_count() instead labels each row by the first of these columns with a value. The print method names the column that labels the institutions of the corpus (leadership$label_column).

At least 4 years must remain after these exclusions, with 2 or more before intervention_year and 2 or more from it on, so that the pre-trend, the level change and the slope change can all be

estimated; otherwise sm_its() aborts. With only 4 years the 4 coefficients fit the series exactly: a family that estimates its dispersion (Gaussian, quasi families) then has no residual degrees of freedom, so its standard errors, intervals and p-values are NaN and sm_its() warns. Families that fix the dispersion, such as the Poisson default for "count", keep their standard errors.

The confidence intervals are Wald intervals on the reference distribution of the p-values: t with the residual degrees of freedom when the family estimates its dispersion (Gaussian and quasi families), normal when the family fixes it (Poisson, binomial). The standard errors treat the yearly values as independent; no correction for autocorrelation is applied.

Value

An sm_its S3 object with components: model (the fitted glm), coefficients (tidy tibble: term, estimate, std.error, conf.low, conf.high, statistic, p.value), series (tibble: year, observed, fitted, counterfactual), and metadata (outcome, intervention_year, family, lag, provisional_years, dropped_years, leadership). dropped_years (integer) lists the years left out for lack of a usable value, provisional_years (integer) the citation-immature years excluded after them. leadership holds the definition used for outcome = "leadership", a list of lead_role, institution_aware (FALSE), institutions (the focal labels or NULL) and label_column (the column that labels the institutions of the corpus, institution_match, institution_name or institution_id, the first with a value; NA without institutions), and is NULL for the other outcomes. impact_source (character) names where the impact values of outcome = "cnci" come from, such as "works$fwci (OpenAlex)" or "works$cited_by_count (field-normalised)"; NA for the other outcomes.

print returns x invisibly.

summary returns the tidy coefficient tibble.

autoplot returns a ggplot.

See Also

sm_did(), sm_citation_maturity()

Other causal: sm_did(), sm_synth()

Examples

```
corpus <- sm_example_corpus(n_works = 200, seed = 1)
its <- sm_its(corpus, intervention_year = 2020, outcome = "count")
its

ggplot2::autoplot(its)


# share of works whose corresponding author is at a focal unit
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
dict <- readr::read_csv(
  sm_example_files("example_realuse_affiliation_dict.csv"),
  col_types = "ccc"
)
fix <- sm_affiliation_match(fix, dict)
lead <- sm_its(fix, intervention_year = 2021, outcome = "leadership",
               institutions = "Armed Forces Hospital Northtown")
```

```
lead$series
```

`sm_journal_in_index` *Verify journal source coverage against an index by ISSN*

Description

Checks whether each supplied ISSN appears in a journal master list (e.g. the Web of Science Master Journal List, a Scopus source list, or a DOAJ export). Matching is by normalised ISSN and checks both print and electronic ISSNs.

This function is deliberately offline and data-driven: it never scrapes or hardcodes proprietary lists. You supply the master list via `reference_list`, so the function is fully testable and reproducible.

Usage

```
sm_journal_in_index(
  issn,
  index = c("wos", "scopus", "doaj"),
  reference_list = NULL,
  call = rlang::caller_env()
)
```

Arguments

`issn`	Character vector of ISSNs to check. Hyphenation and case are normalised automatically (e.g. `"15452786"` and `"1545-2786"` are equivalent; a lowercase `x` check digit is upper-cased).
`index`	Which index the master list represents: `"wos"` (default), `"scopus"`, or `"doaj"`. This is a label echoed in the output; the actual matching is performed entirely against `reference_list`.
`reference_list`	A data frame master list. Expected columns (matched case-insensitively): **title** Journal title (aliases: `title`, `journal`, `journal_title`, `source_title`, `display_name`). **ISSN columns** One or more of `issn`, `issn_print`, `print_issn`, `issn_electronic`, `issn_online`, `eissn`, `issn_l`. Columns whose name indicates "electronic"/"online"/"eissn" are reported as `"electronic"`; print columns as `"print"`; a bare `issn`/`issn_l` column as `"unknown"`. A small synthetic example list ships at `system.file("extdata", "example_journal_index.csv", package = "scimapR")`.
`call`	Caller environment for error reporting.

Value

A tibble with one row per input `issn` and columns: `issn` (the normalised input), `index`, `in_index` (logical), `matched_title` (character or `NA`), `matched_issn_type` (`"print"`/`"electronic"`/`"unknown"`/`NA`). Type-stable: an empty `issn` vector returns a 0-row tibble with these columns.

See Also

sm_coverage_audit()

Other coverage: sm_coverage_audit(), sm_coverage_breakdowns(), sm_match_types(), sm_reconcile()

Examples

```
ref <- utils::read.csv(
  system.file("extdata", "example_journal_index.csv", package = "scimapR"),
  stringsAsFactors = FALSE
)
sm_journal_in_index(c("1078-8956", "1546-170X", "9999-9999"),
                    index = "doaj", reference_list = ref)
```

---

sm_link_references *Link cited references to works in a corpus*

Description

Resolves cited DOIs and OpenAlex identifiers against the works in the corpus. Use this for corpora built from tables or saved before references were linked automatically. No network request is made.

Usage

```
sm_link_references(
  corpus,
  by = c("doi", "openalex_id"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
by	Identifiers to match, in priority order: "doi" and "openalex_id" by default.
verbose	Report the numbers of linked and unresolved references.
call	Caller environment for error reporting.

Details

Links to existing corpus works are preserved. Missing or dangling links are resolved by a normalised DOI in cited_doi or a Web of Science DOI field in cited_raw, including bracketed DOI lists. OpenAlex matching accepts a whole short work id or its URL in cited_raw. A match must identify exactly one work and must not point to the citing work itself. The first identifier type in by giving a unique, non-self match wins. A dangling link without a match is retained. The source identifiers and provenance are unchanged. Locked corpora must first be unlocked.

Readers and fetchers that build references, sm_bind_corpora() and sm_dedupe() link automatically. sm_corpus(), sm_corpus_from_tables(), new_sm_corpus(), sm_materialise(), subsetting and sm_example_corpus() take the supplied links as given.

Value

The corpus with `references$cited_work_id` filled where possible.

See Also

Other corpus: `as_sm_corpus()`, `is_sm_corpus()`, `sm_bind_corpora()`, `sm_build_corpus()`, `sm_concept_vocabularies()`, `sm_corpus()`, `sm_corpus_from_tables()`, `sm_dedupe()`, `sm_materialise()`, `sm_oa_statuses()`, `sm_save_corpus()`, `sm_validate()`, `validate_sm_corpus()`

Examples

```
corpus <- readRDS(sm_example_files("example_realuse_corpus.rds"))
linked <- sm_link_references(corpus)
sum(linked$references$cited_work_id %in% linked$works$work_id)
```

---

sm_lock *Lock or unlock a corpus*

Description

Locking a corpus prevents mutation by `sm_refresh()` and other modifying operations. This is useful for archival or when you want to guarantee reproducibility after generating a `sm_certificate()`.

`sm_lock()` sets `metadata$is_locked = TRUE` with an optional reason. `sm_unlock()` reverses the lock but requires explicit confirmation to prevent accidental unlocking of archived corpora.

Usage

```
sm_lock(corpus, reason = NULL, call = rlang::caller_env())

sm_unlock(corpus, confirm = FALSE, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus` object.
`reason`	Optional character string documenting why the corpus was locked (e.g., `"submitted for review"`, `"certificate generated"`).
`call`	Caller environment for error reporting.
`confirm`	Logical. Must be `TRUE` to actually unlock. This safeguard prevents accidental unlocking of archived corpora.

Details

Merging, deduplication, materialisation, citation maturity, enrichment, embedding changes, clustering, labelling, affiliation matching and screening refuse locked inputs, as do refreshing and input registration. Subsets remain locked but lose their stored lock hash. Merged corpora are unlocked; unlock the inputs explicitly before merging.

The lock records a content hash. `sm_certificate()` warns and `sm_validate()` reports when direct edits changed that content. Printing shows only the lock status and does not hash the corpus. Earlier locked corpora without a stored hash are not checked.

Value

A modified `sm_corpus` with updated lock status.

See Also

Other refresh: `sm_refresh()`, `sm_staleness()`

Examples

```
corpus <- sm_example_corpus()
locked <- sm_lock(corpus, reason = "archival snapshot")
locked$metadata$is_locked

unlocked <- sm_unlock(locked, confirm = TRUE)
unlocked$metadata$is_locked
```

---

sm_match_types *Controlled vocabulary for coverage match types*

Description

The set of values the `match_type` column of `sm_coverage_audit()`'s `matches` tibble (and `sm_reconcile()`'s) can take.

Usage

```
sm_match_types(describe = FALSE)
```

Arguments

`describe`	Logical; if `TRUE`, return a tibble of `level` + `description`, else the ordered character vector (default).

Value

A character vector or a tibble.

See Also

sm_coverage_audit(), sm_reconcile()

Other coverage: sm_coverage_audit(), sm_coverage_breakdowns(), sm_journal_in_index(), sm_reconcile()

Examples

```
sm_match_types()
```

---

sm_materialise *Materialise cached enrichment into a corpus*

Description

Joins cached enrichment data into the matching sm_corpus sub-tibbles by their key columns, returning an updated, schema-valid sm_corpus. This replaces hand-written cache-to-corpus joins (which are easy to get wrong — e.g. a bind_rows() that coerces a NULL element to a logical column).

Usage

```
sm_materialise(
  corpus,
  sources,
  .by = NULL,
  overwrite = FALSE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus.
sources	Either a named list whose names are corpus sub-tables (works, authors, authorships, sources, institutions, references, concepts, ...) and whose elements are tibbles or paths to cached .rds/.parquet files; or a single directory path containing <table>.rds / <table>.parquet files.
.by	Optional named list mapping table name to its join key column(s). Missing or empty key values never match. Defaults: works, provenance and screening use work_id; authors use author_id; institutions use institution_id; sources use source_id; authorships use (work_id, author_id); references use (work_id, ref_index) (formerly (work_id, cited_work_id)); concepts use (work_id, concept_id). For keywords, use .by = list(concepts = c("work_id", "vocabulary", "concept_name")).
overwrite	Logical (default FALSE). When FALSE, enrichment only fills NA cells of overlapping columns; populated cells are never overwritten. When TRUE, non-NA enrichment values win.
call	Caller environment for error reporting.

Details

Missing key columns warn and skip the source. Enrichment rows with a missing key value are dropped with `scimapR_materialise_na_keys`; corpus rows with missing keys keep their values and produce a `scimapR_materialise_unmatched_rows` message. A repeated complete key in `works`, `authors`, `institutions`, `sources`, `references` or `concepts` aborts with `scimapR_materialise_key_not_unique`. The group keys in `authorships`, `provenance` and `screening` can repeat: every matching row receives the enrichment. Duplicate enrichment keys warn and keep the first. Concept vocabularies used as keys are normalised on both sides before matching. After a merge into `concepts`, legacy spellings in its `vocabulary` column are renamed to the names of `sm_concept_vocabularies()`, without a message.

Value

An updated, validated `sm_corpus` with the enrichment columns merged into the relevant sub-tables. New columns are added; existing rows are preserved (this is a column-enrichment join, not a row append).

See Also

`sm_corpus_from_tables()`

Other corpus: `as_sm_corpus()`, `is_sm_corpus()`, `sm_bind_corpora()`, `sm_build_corpus()`, `sm_concept_vocabularies()`, `sm_corpus()`, `sm_corpus_from_tables()`, `sm_dedupe()`, `sm_link_references()`, `sm_oa_statuses()`, `sm_save_corpus()`, `sm_validate()`, `validate_sm_corpus()`

Examples

```
corpus <- sm_example_corpus(n_works = 10, seed = 1)
metrics <- tibble::tibble(work_id = corpus$works$work_id,
                          cnci = runif(10, 0.5, 2))
corpus2 <- sm_materialise(corpus, sources = list(works = metrics))
"cnci" %in% names(corpus2$works)
```

---

sm_merge_screening_decisions

Merge external screening decisions into corpus

Description

Add external screening decisions (from Rayyan, Covidence, or manual spreadsheet) into the corpus screening table.

Usage

```
sm_merge_screening_decisions(corpus, decisions)
```

Arguments

corpus	An sm_corpus.
decisions	A tibble with screening decisions.

Value

An sm_corpus with updated screening.

See Also

Other screening: sm_export_covidence(), sm_export_rayyan(), sm_import_rayyan(), sm_screen_prisma()

---

sm_metric_collab_index

Calculate collaboration index

Description

Computes several collaboration indicators for the corpus, including the Collaboration Index (CI), mean authors per paper, proportion of multi-authored papers, and international collaboration rate.

Usage

```
sm_metric_collab_index(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object with a populated authorships table.
call	Caller environment for error reporting.

Details

The Collaboration Index (CI) for a set of works is typically defined as the mean number of authors per paper. This function returns per-work data, from which aggregate CI can be computed by the user. Country and institution counts are NA when no non-blank value is known. With one known country, is_international is FALSE, even if some affiliations are missing. Institutions are counted by institution_id only; names and affiliation text are not matched.

Value

A tibble with one row per work and columns:

work_id The work identifier.

n_authors Number of authors on the work.

n_countries Distinct known countries; NA if none are known.

n_institutions Distinct known institution_id values; NA if none are known. Affiliation match labels are not counted.

`is_international` Logical; `TRUE` if more than one country is known; `FALSE` if exactly one is known, and `NA` if none are known. Missing codes can hide international collaboration.

`is_multi_authored` Logical; `TRUE` if more than one author.

See Also

Other metrics: `sm_metric_disruption()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_m_index()`, `sm_metric_novelty()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
collab <- sm_metric_collab_index(corpus)
head(collab)
# Aggregate collaboration index:
mean(collab$n_authors)
```

---

sm_metric_disruption *Calculate the CD (disruption) index*

Description

Computes the CD index (Funk & Owen-Smith, 2017) for each work in the corpus. The CD index measures how disruptive a work is by comparing the citation patterns of papers that cite the focal work versus those that cite its references.

Usage

```
sm_metric_disruption(corpus, call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object with a populated `references` table.
`call`	Caller environment for error reporting.

Details

The CD index is defined as:

$$CD = \frac{n_i - n_j}{n_i + n_j + n_k}$$

Where for a focal work f:

n_i Number of works that cite f but do NOT cite any of f's references (disruptive citations).

n_j Number of works that cite BOTH f and at least one of f's references (consolidating citations).

n_k Number of works that cite at least one of f's references but do NOT cite f (ignored-but-referencing citations).

A CD index near +1 indicates the focal work is highly disruptive: its citers tend to ignore its references. A CD index near -1 indicates the focal work consolidates existing ideas.

Only citations between works inside the corpus count. Repeated links and self-references are ignored. The focal work itself is excluded from n_k, as are works published in an earlier year when both years are known. Same-year and undated works remain eligible: publication year is the available time resolution. Citers of the focal work always contribute to n_i or n_j. Works without linked references or without any internal citers receive `NA`.

Value

A tibble with columns `work_id` and `cd_index`. The CD index ranges from -1 (consolidating) to +1 (disruptive). Works without sufficient citation data receive `NA`.

References

Funk, R. J., & Owen-Smith, J. (2017). A Dynamic Network Measure of Technological Change. *Management Science*, 63(3), 791–817. doi:10.1287/mnsc.2015.2366

See Also

`sm_link_references()`

Other metrics: `sm_metric_collab_index()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_m_index()`, `sm_metric_novelty()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
cd <- sm_metric_disruption(corpus)
head(cd)
```

---

sm_metric_fnci *Calculate Field-Normalized Citation Impact*

Description

Computes the Field-Normalized Citation Impact (FNCI) for each work in the corpus. FNCI normalises a work's citation count by the average citation count of all works in the same field and publication year.

Usage

```
sm_metric_fnci(
  corpus,
  classification = c("openalex_concepts", "mesh", "manual"),
  min_cell = NULL,
  by_type = FALSE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
classification	Character; the classification system used to assign works to fields. One of "openalex_concepts" (default), "mesh", or "manual".
min_cell	NULL (default) or a positive integer. When set, a baseline needs at least min_cell works with a known citation count; works in smaller cells fall back to larger baselines (see Details), and the result gains the columns n_cell and baseline.
by_type	Logical. If TRUE, baseline cells are also split by document type, normalised from works$type: articles and reviews get cells of their own and all other types get fnci = NA. Adds the column doc_type. Default FALSE.
call	Caller environment for error reporting.

Details

FNCI is defined as:

$$FNCI_i = \frac{C_i}{\bar{C}_{f,y}}$$

Where C_i is the citation count of work i and $\bar{C}_{f,y}$ is the mean citation count of all works in field f published in year y.

An FNCI of 1.0 means the work is cited at the corpus average of its baseline cell (by default its field and year). Values > 1.0 indicate above-average impact.

For "openalex_concepts", the highest-scoring level-0 concept from the concepts table is used. For "mesh", MeSH headings are used: concepts of the "mesh" vocabulary of sm_concept_vocabularies(), also when spelled "MEDLINE" (requires MeSH data in concepts; see below). For "manual", a field column must already exist in corpus$works.

Value

A tibble with one row per work and the columns work_id, field, year, cited_by_count, field_mean (the baseline: the mean citation count of the work's cell) and fnci. The column set depends on the arguments: by_type = TRUE adds doc_type ("article", "review" or NA), and min_cell adds n_cell (integer; the number of works with a known citation count in the baseline cell) and baseline (a factor with the levels field_year, field_year_pm1, year and none; the level the baseline comes from). An empty corpus gives a 0-row tibble with the same columns.

Small cells

A cell with few works gives an unstable baseline, and a work alone in its cell gets FNCI = 1 by construction. When `min_cell` is `NULL` and more than 25% of the works with a citation count sit in cells with fewer than 10 such works, a message (class `scimapR_fnci_small_cells`) gives that share. `sm_its()`, `sm_did()` and `sm_synth()` (`outcome = "cnci"`) and `sm_metric_summary()` (`metric = "cnci"`) show it too when they derive impact from citation counts; their own `min_cell` argument is passed on here.

Fallback baselines

With `min_cell` set, each work gets the most specific baseline that has at least `min_cell` works with a known citation count, the work itself included:

1. `field_year`: its field and publication year;
2. `field_year_pm1`: its field in the pooled years y - 1 to y + 1 (truncated at the edges of the corpus);
3. `year`: all works of its publication year.

If even the year has fewer than `min_cell` such works, `fnci` and `field_mean` are `NA`, `baseline` is `"none"` and `n_cell` gives the size of the year cell. Works without a field (`"unclassified"`) start at the `year` level, and works without a year get `NA` throughout.

In this mode `field` may list several categories separated by `;`, as the Web of Science WC field does. A work belongs to the cell of each of its categories. For each category the most specific level that reaches `min_cell` is found, and the work's baseline is the average of the category baselines at the most specific level any of them reached: `baseline` names that level and `n_cell` is the smallest of the averaged cells. The baselines are averaged ($C_i/mean(e_k)$), not the ratios ($mean(C_i/e_k)$). `field` keeps the original string. Without `min_cell` the string is one field, as before.

Document types

With `by_type = TRUE`, `works$type` is normalised: lower-cased, its first `;`-separated part, trimmed. `"article"` covers article, journal-article, jour and research-article; `"review"` covers review and review article. Works of any other or a missing type get `fnci = NA` (and `baseline = NA`), with one message (class `scimapR_fnci_other_types`). The type stays part of the cell at every fallback level. The PubMed and Semantic Scholar fetchers record every work as `"journal-article"`, so their reviews count as articles.

MeSH fields

MeSH check tags (Humans, Animals, Male, Female, Pregnancy, the age groups, common laboratory and farm animals, and the historical periods) describe who or what was studied, and common descriptors of study design, statistics and laboratory material (such as Retrospective Studies, Risk Factors, Treatment Outcome or Cells, Cultured) describe how it was studied. Neither is ever used as a field.

`sm_fetch_pubmed()`, `sm_read_pubmed_xml()` and `sm_enrich_concepts()` (`source = "mesh"`) record each heading's major-topic flag as its score: 1 for a major topic of the article (starred in PubMed) and 0 otherwise. A work gets its highest-scoring heading; among several, the one that most works of the corpus carry (ties in alphabetical order). A work whose headings carry no scores,

as with MeSH terms from other sources, gets its heading that is most frequent in the corpus, with one message (class `scimapR_fnci_mesh_unscored`). A work with no other headings is unclassified.

References

Waltman, L., van Eck, N. J., van Leeuwen, T. N., Visser, M. S., & van Raan, A. F. J. (2011). Towards a New Crown Indicator: Some Theoretical Considerations. *Journal of Informetrics*, 5(1), 37–47. doi:10.1016/j.joi.2010.08.001

See Also

Other metrics: `sm_metric_collab_index()`, `sm_metric_disruption()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_m_index()`, `sm_metric_novelty()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
fnci <- sm_metric_fnci(corpus, classification = "openalex_concepts")
head(fnci)

# Fall back to larger baselines for cells with fewer than 5 works, with
# articles and reviews normalised separately
fnci <- sm_metric_fnci(corpus, min_cell = 5, by_type = TRUE)
table(fnci$baseline, useNA = "ifany")
```

---

sm_metric_g_index *Calculate g-index*

Description

Computes the g-index for entities at the specified level. The g-index is the largest g such that the top g works have at least g^2 citations in total (Egghe, 2006).

Usage

```
sm_metric_g_index(
  corpus,
  level = c("author", "institution", "source", "country"),
  self_corrected = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object.
`level`	Character; the entity level. Defaults to `"author"`.
`self_corrected`	Logical (default `FALSE`); remove self-citations (`sm_self_citation()`) before computing the index. Author/institution only.
`call`	Caller environment for error reporting.

Value

A tibble with columns for the entity ID/name and g_index.

See Also

sm_self_citation()

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_self_citation(), sm_summary_authors(), sm_summary_period(), sm_summary_sources(), sm_summary_works()

Examples

```
corpus <- sm_example_corpus()
sm_metric_g_index(corpus, level = "author")
```

---

sm_metric_h_index *Calculate h-index*

Description

Computes the h-index for entities at the specified level. An entity has h-index h if h of its works have at least h citations each.

Usage

```
sm_metric_h_index(
  corpus,
  level = c("author", "institution", "source", "country"),
  self_corrected = FALSE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
level	Character; the entity level. One of "author" (default), "institution", "source", or "country".
self_corrected	Logical (default FALSE). When TRUE, self-citations identified by sm_self_citation() are removed before computing the index (each work's citation count is reduced by the entity's internal self-citations to it, floored at 0). Only available for "author" and "institution" levels. The corrected index is always <= the uncorrected one.
call	Caller environment for error reporting.

Details

Self-correction uses the corpus's internal reference network (no API calls): citations counted against a work are reduced by those coming from works that share the entity. Because the network is internal to the corpus, this is a lower-bound correction on the global `cited_by_count`.

Value

A tibble with columns for the entity ID/name and `h_index`.

See Also

`sm_self_citation()`

Other metrics: `sm_metric_collab_index()`, `sm_metric_disruption()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_m_index()`, `sm_metric_novelty()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
sm_metric_h_index(corpus, level = "author")
```

---

sm_metric_m_index *Calculate m-index (m-quotient)*

Description

Computes the m-index (m-quotient) for entities at the specified level. The m-index equals the h-index divided by the number of years since the entity's first publication (Hirsch, 2005).

Usage

```
sm_metric_m_index(
  corpus,
  level = c("author", "institution", "source", "country"),
  self_corrected = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object.
`level`	Character; the entity level. Defaults to `"author"`.
`self_corrected`	Logical (default `FALSE`); remove self-citations (`sm_self_citation()`) before computing the index. Author/institution only.
`call`	Caller environment for error reporting.

Value

A tibble with columns for the entity ID/name, `h_index`, `first_year`, `career_years`, and `m_index`.

See Also

`sm_self_citation()`

Other metrics: `sm_metric_collab_index()`, `sm_metric_disruption()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_novelty()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
sm_metric_m_index(corpus, level = "author")
```

---

sm_metric_novelty *Calculate Uzzi novelty score*

Description

Computes the Uzzi et al. (2013) novelty score for each work based on atypical combinations of cited journals. A work is novel if its reference list contains unusual journal pairings – combinations that are rarely seen together in the broader literature.

Usage

```
sm_metric_novelty(corpus, call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object with populated `references` and `works` tables. Works and their references should have `source_id` assignments.
`call`	Caller environment for error reporting.

Details

The algorithm:

1. For each work, identify the journals (sources) of its cited references.
2. Enumerate all pairwise journal combinations in each reference list.
3. Compute the observed frequency of each journal pair across all works.
4. Compute the expected frequency under independence (product of marginal frequencies).
5. The novelty score for a work is the median of $-\log_{10}(observed/expected)$ across all its journal pairs. High values indicate atypical combinations.

Works with fewer than 2 references with known journals receive `NA`.

Value

A tibble with columns `work_id` and `novelty`. Higher values indicate more novel (atypical) journal combinations. Works with insufficient data receive `NA`.

References

Uzzi, B., Mukherjee, S., Stringer, M., & Jones, B. (2013). Atypical Combinations and Scientific Impact. *Science*, 342(6157), 468–472. doi:10.1126/science.1240474

See Also

`sm_link_references()`

Other metrics: `sm_metric_collab_index()`, `sm_metric_disruption()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_m_index()`, `sm_metric_rcr()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
nov <- sm_metric_novelty(corpus)
head(nov)
```

---

sm_metric_rcr *Calculate Relative Citation Ratio*

Description

Computes the Relative Citation Ratio (RCR) for each work in the corpus. RCR normalises citation counts by the expected citation rate of works in the same field and publication year, providing a field-normalised measure of scientific influence.

Usage

```
sm_metric_rcr(corpus, baseline = "field_year", call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object.
`baseline`	Character; the normalisation baseline. Currently only `"field_year"` is supported, which normalises by field (derived from concepts) and publication year.
`call`	Caller environment for error reporting.

Details

The RCR is calculated as:

$$RCR = \frac{C_i}{E_i}$$

Where:

C_i The citation count of work i.

E_i The expected citation count, computed as the mean citation count of works in the same field-year group.

Field assignment uses the top-level concept (level 0) from the concepts table. Works without concept assignments are grouped into an "unclassified" field.

An RCR of 1.0 means the work is cited at the average rate for its field-year. Values above 1.0 indicate above-average impact.

Value

A tibble with columns `work_id`, `cited_by_count`, `expected_rate`, and `rcr`. Works without sufficient data receive `NA` for `rcr`.

References

Hutchins, B. I., Yuan, X., Anderson, J. M., & Santangelo, G. M. (2016). Relative Citation Ratio (RCR): A New Metric That Uses Citation Rates to Measure Influence at the Article Level. *PLOS Biology*, 14(9), e1002541. doi:10.1371/journal.pbio.1002541

See Also

Other metrics: `sm_metric_collab_index()`, `sm_metric_disruption()`, `sm_metric_fnci()`, `sm_metric_g_index()`, `sm_metric_h_index()`, `sm_metric_m_index()`, `sm_metric_novelty()`, `sm_self_citation()`, `sm_summary_authors()`, `sm_summary_period()`, `sm_summary_sources()`, `sm_summary_works()`

Examples

```
corpus <- sm_example_corpus()
rcr <- sm_metric_rcr(corpus)
head(rcr)
```

`sm_metric_summary` *Robust summary of a heavy-tailed impact metric*

Description

Summarises a work-level impact metric robustly. Citation-based metrics are heavy-tailed, so the mean is a poor central estimate; this function reports the median with a bootstrap confidence interval and the proportion of papers in the top citation fraction of a chosen reference, alongside the mean. Without an external reference, `pp_top10` equals `top_pct` by construction and does not measure impact relative to other publications.

Usage

```
sm_metric_summary(
  corpus,
  metric = c("citations", "cnci", "rcr", "fwci"),
  robust = TRUE,
  n_boot = 2000L,
  conf = 0.95,
  top_pct = 0.1,
  seed = NULL,
  min_cell = NULL,
  top_reference = NULL,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus`.
`metric`	Which work-level metric to summarise: `"citations"` (`cited_by_count`), `"cnci"` (field-normalised citation impact, resolved as in `sm_its()`: a stored impact column such as `works$cnci` or `works$fwci`, a `corpus$metrics` table, else the FNCI that `sm_metric_fnci()` derives from citation counts), `"rcr"` (`sm_metric_rcr()`) or `"fwci"` (OpenAlex's field-weighted citation impact, `works$fwci`; see `sm_enrich_openalex_impact(`
`robust`	Logical (default `TRUE`). When `TRUE`, report the median with a bootstrap CI and `pp_top10`. When `FALSE`, report only `n`, `mean`, and `median` (no resampling).
`n_boot`	Number of bootstrap resamples for the median CI (default `2000`).
`conf`	Confidence level for the bootstrap interval (default `0.95`).
`top_pct`	Single number strictly between 0 and 1: the top fraction of the reference distribution (default `0.1`). Ties share credit fractionally. The output column remains named `pp_top10` for any fraction.
`seed`	Optional integer seed for reproducible bootstrap resampling. When supplied, the RNG state is saved and restored so the call has no global side effect (mirroring scimapR's reproducibility guarantees).

min_cell	For metric = "cnci": NULL (default) or a positive integer, passed to sm_metric_fnci() when the impact is derived from citation counts, as in sm_its().
top_reference	Reference for pp_top10: NULL uses the corpus itself; a numeric vector supplies citation counts (at least two after dropping NA); a data frame supplies numeric cited_by_count and optionally year and/or field, matched on keys present in both it and works. Missing keys or unmatched cells are excluded with a message; no matching works gives NA. Reference counts must be finite and non-negative. Data-frame cells may contain a single reference work. "openalex" uses is_top_10pct for top_pct = 0.1 or is_top_1pct for 0.01, averaging non-missing flags; see sm_enrich_openalex_impact().
call	Caller environment for error reporting.

Details

The bootstrap uses base-R resampling by default; if the optional **boot** package is installed it is used instead (percentile interval). pp_top10 remains citation-based for every metric; top_reference changes neither the metric summary nor the bootstrap. Tied citation counts share the part of their reference block in the top fraction equally, following Waltman and Schreiber (2013). Using the corpus itself gives top_pct exactly before rounding. An external reference from comparable fields and years, or OpenAlex flags, makes this an informative share.

For "cnci" and "fwci" only works with a finite value count, and for "cnci" only works of corpus$works (a corpus$metrics table may hold others). metric = "fwci" aborts (class scimapR_no_fwci) when the works have no fwci value.

Value

A one-row tibble: metric, n, mean, median, and – when robust = TRUE – median_ci_low, median_ci_high, pp_top10, n_boot. Type-stable: an empty corpus, or for "cnci" a corpus without usable impact data, returns a one-row tibble with n = 0 and NA statistics. The attribute impact_source (character) names where the values come from: for "cnci" the same label as the impact_source of sm_its() (such as "works$fwci (OpenAlex)"), for "fwci" "works$fwci (OpenAlex)", for "citations" "works$cited_by_count" and for "rcr" "sm_metric_rcr()"; it is NA on the n = 0 row. For "cnci" and "fwci" a message (class scimapR_impact_source) names the source too.

References

Waltman, L., & Schreiber, M. (2013). On the calculation of percentile-based bibliometric indicators. Journal of the American Society for Information Science and Technology, 64(2), 372–379.

See Also

sm_summary_works(), sm_count()

Other counting: sm_citation_maturity(), sm_citation_window(), sm_count()

Examples

```
corpus <- sm_example_corpus(n_works = 100, seed = 1)
sm_metric_summary(corpus, metric = "citations", seed = 1)
```

```
# the real-use fixture carries OpenAlex's FWCI, which "cnci" then uses
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
s <- sm_metric_summary(fix, metric = "cnci", robust = FALSE)
attr(s, "impact_source")
```

`sm_network_citation` *Build a direct citation network*

Description

Constructs a directed citation network from the references table of an `sm_corpus`. Each node represents a work and a directed edge runs from the citing work to the cited work. Only works present in the corpus (i.e. both the citing and cited work exist in `corpus$works`) are included as nodes unless external references are also present via `cited_work_id`.

Usage

```
sm_network_citation(corpus, call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object. Must contain a non-empty `references` table with columns `work_id` and `cited_work_id`.
`call`	Caller environment for error reporting.

Details

Empty input (zero works or zero references) returns an empty directed `tbl_graph` with a single `name` column on nodes and no edges. Each work counts a cited item once: repeated links and self-loops are removed. References with no link to a corpus work trigger a warning when none is linked; use `sm_link_references()` to resolve identifiers first.

Value

A tidygraph::tbl_graph object (directed). Node data contains `name` (the work ID) plus any columns from `corpus$works`. Edge data contains `from`, `to`, and any additional reference metadata.

See Also

`sm_link_references()`

Other networks: `sm_network_cocitation()`, `sm_network_collab()`, `sm_network_coupling()`, `sm_network_coword()`, `sm_network_semantic()`

Examples

```
corpus <- sm_example_corpus()
g <- sm_network_citation(corpus)
g
```

`sm_network_cocitation` *Build a co-citation network*

Description

Constructs an undirected co-citation network. Two works (A and B) are connected by an edge if they are both cited by a common third work. The edge weight equals the number of works that co-cite A and B.

Usage

```
sm_network_cocitation(
  corpus,
  min_weight = 2L,
  linked_only = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object with a populated `references` table.
`min_weight`	Integer; minimum co-citation count to retain an edge. Defaults to `2L` to reduce noise.
`linked_only`	Use only references with a `cited_work_id`. Defaults to `FALSE`, which includes cited items outside the corpus.
`call`	Caller environment for error reporting.

Details

The algorithm:

1. For every citing work, enumerate all pairs of cited works.
2. Count how often each pair co-occurs.
3. Filter by `min_weight`. Each work counts a cited item once; self-references are excluded. Items are keyed by `cited_work_id`, else normalised DOI, else OpenAlex id, else upper-case reference text with whitespace squished and final periods removed. Identifier kinds are kept separate, even when labels coincide. `linked_only = TRUE` restores the earlier behaviour and warns when none points to corpus works.

Empty input (zero works, zero references, or no pairs above the threshold) returns an empty undirected `tbl_graph`.

Value

A tidygraph::tbl_graph object (undirected). Nodes carry `name` (work ID for corpus works, else DOI, OpenAlex id or original reference text), `in_corpus`, and columns from `corpus$works` (missing for external items). Edges carry a `weight` column representing co-citation frequency.

See Also

sm_link_references()

Other networks: sm_network_citation(), sm_network_collab(), sm_network_coupling(), sm_network_coword(), sm_network_semantic()

Examples

```
corpus <- sm_example_corpus()
g <- sm_network_cocitation(corpus, min_weight = 2L)
g
```

---

sm_network_collab *Build a collaboration network*

Description

Constructs an undirected collaboration (co-authorship) network. Nodes represent entities at the chosen `level` (authors, institutions, or countries) and an edge connects two entities if they co-authored at least one work. Edge weight equals the number of co-authored works.

Usage

```
sm_network_collab(
  corpus,
  level = c("author", "institution", "country"),
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object with a populated `authorships` table.
`level`	Character; the entity level for network nodes. One of `"author"` (default), `"institution"`, or `"country"`.
`call`	Caller environment for error reporting.

Details

For `level = "author"`, the `author_id` column from `authorships` is used and author metadata is joined from `corpus$authors`.

For `level = "institution"`, the `institution_id` column is used and institution metadata is joined from `corpus$institutions`.

For `level = "country"`, the `country_code` column is used.

Empty input returns an empty undirected `tbl_graph`.

Value

A tidygraph::tbl_graph object (undirected). Nodes carry `name` (entity ID or label) and, for `"author"` level, columns from `corpus$authors`. Edges carry a `weight` column.

See Also

Other networks: `sm_network_citation()`, `sm_network_cocitation()`, `sm_network_coupling()`, `sm_network_coword()`, `sm_network_semantic()`

Examples

```
corpus <- sm_example_corpus()
g <- sm_network_collab(corpus, level = "author")
g
```

---

sm_network_coupling *Build a bibliographic coupling network*

Description

Constructs an undirected bibliographic coupling network. Two works are connected if they share at least `min_shared` cited references. The edge weight equals the number of shared references.

Usage

```
sm_network_coupling(
  corpus,
  min_shared = 1L,
  linked_only = FALSE,
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An sm_corpus object with a populated `references` table.
`min_shared`	Integer; minimum number of shared references to create an edge. Defaults to `1L`.
`linked_only`	Use only references with a `cited_work_id`. Defaults to `FALSE`, which also counts shared references outside the corpus.
`call`	Caller environment for error reporting.

Details

Bibliographic coupling is the mirror of co-citation: two works are coupled if they cite the same references, suggesting intellectual similarity. Each work counts a cited item once; self-references are excluded. Items are keyed by `cited_work_id`, else normalised DOI, else OpenAlex id, else upper-case reference text with whitespace squished and final periods removed. Identifier kinds are kept separate. `linked_only = TRUE` restores the earlier behaviour and warns when none points to corpus works.

Empty input (zero works, zero references, or no pairs above the threshold) returns an empty undirected `tbl_graph`.

Value

A tidygraph::tbl_graph object (undirected). Nodes carry `name` (work ID) and columns from `corpus$works`. Edges carry a `weight` column representing the number of shared references.

See Also

`sm_link_references()`

Other networks: `sm_network_citation()`, `sm_network_cocitation()`, `sm_network_collab()`, `sm_network_coword()`, `sm_network_semantic()`

Examples

```
corpus <- sm_example_corpus()
g <- sm_network_coupling(corpus, min_shared = 3L)
g
```

`sm_network_coword` *Build a co-word (co-occurrence) network*

Description

Constructs an undirected co-word network. Nodes represent terms and an edge connects two terms if they co-occur within the same work. Edge weight equals the number of works in which both terms appear.

Usage

```
sm_network_coword(
  corpus,
  field = c("concepts", "title", "abstract", "keywords"),
  ngram = 1L,
  min_freq = 5L,
  stopwords = NULL,
  vocabulary = NULL,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
field	Character; the text field to extract terms from. One of "concepts" (default, uses corpus$concepts), "title", "abstract", or "keywords".
ngram	Integer; for free-text fields ("title", "abstract"), the n-gram size to tokenise into. Defaults to 1L (unigrams).
min_freq	Integer; minimum document frequency a term must have to be included. Defaults to 5L.
stopwords	Character vector of additional stopwords to remove when tokenising free text. Combined with tidytext::stop_words. Set to NULL (default) for no additional stopwords.
vocabulary	For field = "keywords" or "concepts", the concept vocabularies to take terms from: names of vocabularies or of families of sm_concept_vocabularies(), such as "author-keywords" or "indexer_keyword". NULL (default) means the author keywords for "keywords" (see Details) and every vocabulary for "concepts". Must be NULL for "title" and "abstract", whose terms come from the text.
call	Caller environment for error reporting.

Details

When field = "concepts", terms come from the concept_name column in corpus$concepts (no tokenisation needed), of every vocabulary or of those named in vocabulary.

When field = "keywords", terms are whole keywords from corpus$concepts, not tokenised. By default these are the author keywords (the author_keyword family: author-keywords, manual-tags); a corpus without any falls back to the indexer keywords (the indexer_keyword family: Keywords Plus, Scopus index keywords, Zotero automatic tags and MeSH) with a message, and a corpus with neither gives an empty network with a message. The fallback applies to the whole corpus, not per work.

When field is "title" or "abstract", the text is tokenised with tidytext::unnest_tokens() using the chosen ngram size and stopwords are removed.

Terms are lower-cased, and terms below min_freq are dropped. Empty input returns an empty undirected tbl_graph.

Value

A tidygraph::tbl_graph object (undirected). Nodes carry name (the term) and freq (document frequency). Edges carry a weight column (co-occurrence count).

See Also

Other networks: sm_network_citation(), sm_network_cocitation(), sm_network_collab(), sm_network_coupling(), sm_network_semantic()

Examples

```
corpus <- sm_example_corpus()
g <- sm_network_coword(corpus, field = "concepts", min_freq = 5L)
g

# Keyword networks from a Web of Science file
path <- tempfile(fileext = ".txt")
writeLines(c(
  "PT J", "AU Doe, J", "TI First work",
  "DE heat illness; recruits; cooling", "ID HYPERTHERMIA; EXERCISE",
  "PY 2021", "UT WOS:000000000000001", "ER",
  "PT J", "AU Roe, R", "TI Second work",
  "DE heat illness; recruits", "ID HYPERTHERMIA; EXERCISE",
  "PY 2022", "UT WOS:000000000000002", "ER", "EF"
), path)
wos <- sm_read_wos(path, verbose = FALSE)
sm_network_coword(wos, field = "keywords", min_freq = 2L)
sm_network_coword(wos, field = "keywords", vocabulary = "keywords-plus",
                  min_freq = 2L)
```

`sm_network_semantic` *Build a semantic similarity network*

Description

Constructs an undirected k-nearest-neighbour network based on work embeddings. Each work is connected to its k nearest neighbours in embedding space (cosine similarity). The edge weight is the cosine similarity between the connected works.

Usage

```
sm_network_semantic(corpus, k = 10L, call = rlang::caller_env())
```

Arguments

`corpus`	An sm_corpus object. Must contain an `embeddings` matrix (see `sm_embed_works()`).
`k`	Integer; number of nearest neighbours per work. Defaults to `10L`.
`call`	Caller environment for error reporting.

Details

Work IDs and embedding row names must be unique. Repeated IDs stop with class `scimapR_duplicate_ids`; use `sm_validate()` to inspect them.

Embeddings must be present in `corpus$embeddings` (a numeric matrix with row names matching `work_id`). Use `sm_embed_works()` to compute them. Works without a row, such as works without text, are not nodes.

The function computes cosine similarity via matrix multiplication on L2-normalised vectors and selects the top-k neighbours for each work.

Empty input or missing embeddings returns an empty undirected `tbl_graph`.

Value

A tidygraph::tbl_graph object (undirected). Nodes carry `name` (work ID) and columns from `corpus$works`. Edges carry a `weight` column (cosine similarity, between 0 and 1).

See Also

Other networks: `sm_network_citation()`, `sm_network_cocitation()`, `sm_network_collab()`, `sm_network_coupling()`, `sm_network_coword()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE)
g <- sm_network_semantic(corpus, k = 5L)
g
```

---

sm_oa_statuses *Controlled vocabulary for open-access statuses*

Description

Canonical values for `works$oa_status`. All levels except `closed` are open; missing and unrecognised values are unknown. Status definitions follow the OpenAlex taxonomy (see the link below), with `open` for access without a known colour. When an export names several statuses, the package uses the first recognised level in the order below: diamond, gold, hybrid, bronze, green, open, closed.

`sm_corpus()` and `sm_corpus_from_tables()` normalise common spellings with a message. Consumers also recognise those spellings in saved corpora. The column remains character, and unknown strings are kept as written.

Usage

```
sm_oa_statuses(describe = FALSE)
```

Arguments

`describe`	Logical; if `TRUE`, return a tibble with a factor `level`, logical `is_open` and character `description`; otherwise the character vector of statuses in precedence order.

Value

A character vector or a tibble of status descriptions.

See Also

`sm_audit_oa()`, `sm_filter_works()`, https://help.openalex.org/data/works/open-access/

Other corpus: `as_sm_corpus()`, `is_sm_corpus()`, `sm_bind_corpora()`, `sm_build_corpus()`, `sm_concept_vocabularies()`, `sm_corpus()`, `sm_corpus_from_tables()`, `sm_dedupe()`, `sm_link_references()`, `sm_materialise()`, `sm_save_corpus()`, `sm_validate()`, `validate_sm_corpus()`

Examples

```
sm_oa_statuses()
sm_oa_statuses(describe = TRUE)
```

`sm_palette_qualitative` *Qualitative viridis palette*

Description

Qualitative viridis palette

Usage

```
sm_palette_qualitative(n, option = "D")
```

Arguments

`n`	Number of colours.
`option`	Viridis palette option.

Value

A character vector of hex colours.

See Also

Other plots: `autoplot.sm_corpus()`, `sm_plot_bradford()`, `sm_plot_citation_network()`, `sm_plot_collab()`, `sm_plot_equity_dashboard()`, `sm_plot_evolution()`, `sm_plot_heaps()`, `sm_plot_landscape()`, `sm_plot_lotka()`, `sm_plot_production()`, `sm_plot_thematic_map()`, `sm_plot_top()`, `sm_scale_color()`, `sm_theme()`

`sm_plot_bradford` *Plot Bradford's law*

Description

Source productivity distribution following Bradford's law of scattering.

Usage

```
sm_plot_bradford(corpus, dark = FALSE, ...)
```

Arguments

corpus	An sm_corpus.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_citation_network(), sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(), sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(), sm_theme()

---

sm_plot_citation_network

Plot citation network

Description

Visualise the citation network of top-cited works.

Usage

```
sm_plot_citation_network(
  corpus,
  top_n = 50L,
  dark = FALSE,
  precompute = FALSE,
  max_nodes = NULL,
  ...
)
```

Arguments

corpus	An sm_corpus.
top_n	Number of top works to include.
dark	Logical; dark mode?
precompute	Logical (default FALSE). When TRUE, the graph layout is computed eagerly and a plain self-contained ggplot (materialised coordinates) is returned, instead of a lazy ggraph object. Use this when embedding the plot in an RMarkdown/knitr/callr/workflowr document: the heavy layout runs once here rather than in the (often memory-limited) render subprocess, where large ggraph objects can crash the harness.

max_nodes	Optional integer node cap. NULL (default) keeps all nodes so existing renders are unchanged; set it to downsample large graphs to the highest-degree nodes (opt-in, with a cli message).
...	Additional arguments.

Value

A ggplot object (a ggraph plot when precompute = FALSE, a plain ggplot when precompute = TRUE).

Large graphs

For big networks, prefer precompute = TRUE and save the returned object (e.g. with saveRDS()); printing it in a document then re-renders cheaply without recomputing the layout. See the networks section of the getting- started vignette.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(), sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(), sm_theme()

Examples

```
corpus <- sm_example_corpus()
sm_plot_citation_network(corpus)
```

---

sm_plot_collab *Plot collaboration map*

Description

Visualise international or institutional collaboration patterns.

Usage

```
sm_plot_collab(
  corpus,
  level = c("country", "institution", "author"),
  top_n = 20L,
  dark = FALSE,
  precompute = FALSE,
  max_nodes = NULL,
  ...
)
```

Arguments

corpus	An sm_corpus.
level	Collaboration level.
top_n	Number of top entities to include.
dark	Logical; dark mode?
precompute	Logical (default FALSE). When TRUE, return a plain ggplot with the layout computed eagerly (see sm_plot_citation_network() for why this matters when knitting large graphs).
max_nodes	Optional integer node cap. NULL (default) keeps all nodes; set it to downsample large graphs to the highest-degree nodes (opt-in, with a cli message).
...	Additional arguments.

Value

A ggplot object (ggraph when precompute = FALSE, plain ggplot when precompute = TRUE).

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(), sm_plot_landscape(),
sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(),
sm_theme()

Examples

```
corpus <- sm_example_corpus()
sm_plot_collab(corpus, level = "country")
```

---

sm_plot_collab_turnover

Plot collaborator turnover

Description

Visualise collaborator retention and turnover across career periods.

Usage

```
sm_plot_collab_turnover(traj, dark = FALSE, ...)
```

Arguments

traj	An sm_trajectory object.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object.

See Also

Other trajectory: is_sm_trajectory(), sm_author_trajectory(), sm_plot_topic_pivots(), sm_plot_trajectory(), sm_researcher_profile(), sm_trajectory

Examples

```
corpus <- sm_example_corpus()
traj <- sm_author_trajectory(corpus, author_id = corpus$authors$author_id[1])
sm_plot_collab_turnover(traj)
```

---

sm_plot_equity_dashboard

Plot equity dashboard

Description

Four-panel visualization: geographic distribution, gender mix, funding sources, and OA evolution. Uses viridis throughout.

Usage

```
sm_plot_equity_dashboard(corpus, dark = FALSE, ...)
```

Arguments

corpus	An sm_corpus.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object composed via patchwork.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(, sm_plot_collab(), sm_plot_evolution(), sm_plot_heaps(), sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(), sm_theme()

Examples

```
corpus <- sm_example_corpus()
sm_plot_equity_dashboard(corpus)
```

`sm_plot_evolution` *Plot topic evolution over time*

Description

Visualise how research topics/clusters evolve across time periods.

Usage

```
sm_plot_evolution(corpus, time_var = "year", dark = FALSE, ...)
```

Arguments

`corpus`	An `sm_corpus` with clusters and years.
`time_var`	Column name for time variable. Default `"year"`.
`dark`	Logical; dark mode?
`...`	Additional arguments.

Value

A `ggplot` object.

See Also

Other plots: `autoplot.sm_corpus()`, `sm_palette_qualitative()`, `sm_plot_bradford()`, `sm_plot_citation_network(`
`sm_plot_collab()`, `sm_plot_equity_dashboard()`, `sm_plot_heaps()`, `sm_plot_landscape()`,
`sm_plot_lotka()`, `sm_plot_production()`, `sm_plot_thematic_map()`, `sm_plot_top()`, `sm_scale_color()`,
`sm_theme()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE, seed = 42)
corpus <- sm_cluster_hdbscan(corpus, min_cluster_size = 10)
sm_plot_evolution(corpus)
```

sm_plot_heaps *Plot Heaps' law*

Description

Vocabulary growth curve over accumulated documents.

Usage

```
sm_plot_heaps(corpus, dark = FALSE, ...)
```

Arguments

corpus	An sm_corpus.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_landscape(),
sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(),
sm_theme()

sm_plot_landscape *Plot research landscape*

Description

UMAP/t-SNE/PCA scatter plot of work embeddings, coloured by cluster.

Usage

```
sm_plot_landscape(
  corpus,
  color_by = "cluster_id",
  reducer = c("umap", "tsne", "pca"),
  n_components = 2L,
  dark = FALSE,
  ...
)
```

Arguments

corpus	An sm_corpus with embeddings.
color_by	Column name in works to colour by. Default "cluster_id".
reducer	Dimensionality reduction method.
n_components	Number of output dimensions (2 for plotting).
dark	Logical; dark mode?
...	Additional arguments.

Details

UMAP retains its Euclidean metric and automatic thread settings. If its PCA dependency raises the known older-R NULL-input error, it retries with base SVD initialisation and reports scimapR_umap_pca_fallback. Other errors propagate unchanged.

Value

A ggplot object with one point per row of corpus$embeddings, so works without an embedding are not shown.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(),
sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(),
sm_theme()

Examples

```
corpus <- sm_example_corpus()
corpus <- sm_cluster_hdbscan(corpus, min_cluster_size = 10)
sm_plot_landscape(corpus)
```

---

sm_plot_lotka *Plot Lotka's law*

Description

Frequency distribution of author productivity compared to Lotka's law.

Usage

```
sm_plot_lotka(corpus, dark = FALSE, ...)
```

Arguments

corpus	An sm_corpus.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(),
sm_plot_landscape(), sm_plot_production(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(),
sm_theme()

---

sm_plot_production *Plot annual scientific production*

Description

Bar chart of publication counts over time.

Usage

```
sm_plot_production(corpus, by = c("year", "month"), dark = FALSE, ...)
```

Arguments

corpus	An sm_corpus object.
by	Time unit: "year" or "month".
dark	Logical; dark mode?
...	Additional arguments (currently unused).

Value

A ggplot object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(),
sm_plot_landscape(), sm_plot_lotka(), sm_plot_thematic_map(), sm_plot_top(), sm_scale_color(),
sm_theme()

Examples

```
corpus <- sm_example_corpus()
sm_plot_production(corpus)
```

`sm_plot_thematic_map` *Plot thematic map (Callon centrality-density)*

Description

Strategic diagram plotting clusters by centrality (external links) and density (internal links), following Callon et al.

Usage

```
sm_plot_thematic_map(corpus, by = "cluster_id", dark = FALSE, ...)
```

Arguments

`corpus`	An `sm_corpus` with clusters.
`by`	Grouping variable. Default `"cluster_id"`.
`dark`	Logical; dark mode?
`...`	Additional arguments.

Value

A `ggplot` object.

See Also

Other plots: `autoplot.sm_corpus()`, `sm_palette_qualitative()`, `sm_plot_bradford()`, `sm_plot_citation_network()`, `sm_plot_collab()`, `sm_plot_equity_dashboard()`, `sm_plot_evolution()`, `sm_plot_heaps()`, `sm_plot_landscape()`, `sm_plot_lotka()`, `sm_plot_production()`, `sm_plot_top()`, `sm_scale_color()`, `sm_theme()`

Examples

```
corpus <- sm_example_corpus(with_embeddings = TRUE, seed = 42)
corpus <- sm_cluster_hdbscan(corpus, min_cluster_size = 10)
sm_plot_thematic_map(corpus)
```

`sm_plot_top` *Plot top entities*

Description

Horizontal bar chart of top authors, sources, institutions, or countries.

Usage

```
sm_plot_top(
  corpus,
  level = c("authors", "sources", "institutions", "countries"),
  n = 20L,
  metric = c("count", "citations"),
  dark = FALSE,
  ...
)
```

Arguments

`corpus`	An `sm_corpus`.
`level`	Entity level.
`n`	Number of top entities to show.
`metric`	Ranking metric.
`dark`	Logical; dark mode?
`...`	Additional arguments.

Value

A `ggplot` object.

See Also

Other plots: `autoplot.sm_corpus()`, `sm_palette_qualitative()`, `sm_plot_bradford()`, `sm_plot_citation_network(`
`sm_plot_collab()`, `sm_plot_equity_dashboard()`, `sm_plot_evolution()`, `sm_plot_heaps()`,
`sm_plot_landscape()`, `sm_plot_lotka()`, `sm_plot_production()`, `sm_plot_thematic_map()`,
`sm_scale_color()`, `sm_theme()`

Examples

```
corpus <- sm_example_corpus()
sm_plot_top(corpus, level = "authors", n = 10)
```

sm_plot_topic_pivots *Plot topic pivots*

Description

Vertical timeline showing when an author's research focus shifted.

Usage

```
sm_plot_topic_pivots(traj, dark = FALSE, ...)
```

Arguments

traj	An sm_trajectory object.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object.

See Also

Other trajectory: is_sm_trajectory(), sm_author_trajectory(), sm_plot_collab_turnover(), sm_plot_trajectory(), sm_researcher_profile(), sm_trajectory

Examples

```
corpus <- sm_example_corpus()
traj <- sm_author_trajectory(corpus, author_id = corpus$authors$author_id[1])
sm_plot_topic_pivots(traj)
```

sm_plot_trajectory *Plot author trajectory*

Description

Multi-panel visualization of an author's career trajectory.

Usage

```
sm_plot_trajectory(traj, dark = FALSE, ...)
```

Arguments

traj	An sm_trajectory object.
dark	Logical; dark mode?
...	Additional arguments.

Value

A ggplot object composed via patchwork.

See Also

Other trajectory: is_sm_trajectory(), sm_author_trajectory(), sm_plot_collab_turnover(), sm_plot_topic_pivots(), sm_researcher_profile(), sm_trajectory

Examples

```
corpus <- sm_example_corpus()
traj <- sm_author_trajectory(corpus, author_id = corpus$authors$author_id[1])
sm_plot_trajectory(traj)
```

---

sm_provenance *Get corpus provenance*

Description

Returns the full provenance table from a corpus, documenting the data lineage of every work: where it was fetched from, when, with which query, and which version of scimapR performed the ingestion.

Usage

```
sm_provenance(corpus)
```

Arguments

corpus	An sm_corpus object.

Value

A tibble with columns work_id, source, source_id_external, fetch_date, query, engine, scimapR_version, and prompt_hash, and the optional columns kind (the kind of step: "api_query", "file", "enrichment", "refresh" or "manual", see sm_certificate(); NA for rows that do not record one), filter and n_max (the filter and the maximum number of works of a search, which sm_rebuild_from_cert() uses to replay it). File readers record the file in query relative to the working directory when it lies below it, else by its name.

See Also

Other reproducibility: sm_certificate(), sm_cite_corpus(), sm_diff_corpora(), sm_hash_corpus(), sm_register_input(), sm_snapshot()

Examples

```
corpus <- sm_example_corpus()
sm_provenance(corpus)
```

`sm_query` *Search works by text query*

Description

Full-text search across titles and abstracts in a corpus.

Usage

```
sm_query(corpus, query, fields = c("title", "abstract"), ignore_case = TRUE)
```

Arguments

`corpus`	An `sm_corpus` object.
`query`	A search string (regex supported).
`fields`	Which fields to search. Default `c("title", "abstract")`.
`ignore_case`	Logical; case-insensitive search?

Value

An `sm_corpus` with matching works.

See Also

Other filters: sm_filter_works()

Examples

```
corpus <- sm_example_corpus()
results <- sm_query(corpus, "transcriptomics")
```

`sm_question` *Create a structured research question*

Description

Constructs an `sm_question` S3 object encoding a structured research question using a recognised framework (PICO, PECO, PICOS, SPIDER, or free-form). The object carries all structured fields, auto-generated query strings suitable for bibliographic databases, and a prompt template for LLM-based screening.

The question receives a content-addressable ID (SHA-256 hash of all input fields), ensuring that identical questions always produce the same identifier regardless of when they are created.

Usage

```
sm_question(
  text,
  framework = c("PICO", "PECO", "PICOS", "SPIDER", "free"),
  population = NULL,
  intervention = NULL,
  comparison = NULL,
  outcome = NULL,
  exposure = NULL,
  design = NULL,
  timeframe = NULL,
  include_terms = NULL,
  exclude_terms = NULL,
  languages = "en",
  notes = NULL
)

## S3 method for class 'sm_question'
print(x, ...)

## S3 method for class 'sm_question'
format(x, ...)
```

Arguments

`text`	Character. The full research question in natural language.
`framework`	Character. The structuring framework. One of `"PICO"` (Population, Intervention, Comparison, Outcome), `"PECO"` (Population, Exposure, Comparison, Outcome), `"PICOS"` (adds Study design), `"SPIDER"` (Sample, Phenomenon of Interest, Design, Evaluation, Research type), or `"free"` for unstructured questions.
`population`	Character. The population or sample under study.
`intervention`	Character. The intervention or phenomenon of interest.

comparison	Character. The comparator or control condition.
outcome	Character. The outcome(s) of interest.
exposure	Character. The exposure (for PECO framework).
design	Character. Study design (for PICOS/SPIDER frameworks).
timeframe	Character. Optional time restriction.
include_terms	Character vector. Additional terms that must appear in relevant works.
exclude_terms	Character vector. Terms whose presence disqualifies a work.
languages	Character vector. ISO 639-1 language codes to restrict the search. Default "en".
notes	Character. Free-text notes about the question.
x	An sm_question object.
...	Ignored.

Value

An sm_question S3 object with fields:

id Content-hash identifier.

text The research question text.

framework The structuring framework.

created POSIXct creation timestamp.

population, intervention, comparison, outcome, exposure, design, timeframe Structured facets (may be NULL).

include_terms, exclude_terms Term filters.

languages Language filter.

notes Free-text notes.

query_strings Named list of database-specific query strings.

prompt_template A prompt template for LLM screening.

- print(): x invisibly.
- format(): A single character string.

See Also

Other question: is_sm_question(), sm_corpus_for_question(), sm_screen_against_question(), sm_screen_regex(), sm_screen_summary()

Examples

```
q <- sm_question(
  text = "Does immunotherapy improve survival in metastatic melanoma?",
  framework = "PICO",
  population = "adults with metastatic melanoma",
  intervention = "immune checkpoint inhibitors",
  comparison = "chemotherapy",
  outcome = "overall survival"
)
print(q)
```

`sm_read_auto` *Auto-detect bibliographic file format and read*

Description

Detect the bibliographic file format from the file extension and content, then dispatch to the appropriate reader function.

Usage

```
sm_read_auto(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a bibliographic file.
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`engine`	Character scalar. One of `"native"` (built-in parser), `"bibliometrix"` (delegate to `bibliometrix::convert2df()`), or `"auto"` (try bibliometrix first, fall back to native). Passed through to the selected reader. Ignored for formats without engine support (OpenAlex JSON, Zotero, EndNote XML).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

Format detection proceeds in two stages:

1. **Extension-based**: `.bib` (BibTeX), `.ris` (RIS), `.json`/`.jsonl` (OpenAlex JSON), `.xml` (PubMed XML or EndNote XML).
2. **Content-based**: For `.csv`, `.tsv`, and `.txt` files, the first few lines are inspected for format-specific signatures:
   - WoS plaintext: begins with `FN` or `PT` tags
   - WoS tab-delimited: header contains PT, AU, TI and UT; also detected in `.tab` files and files without an extension
   - Scopus CSV: contains `EID` column header

- Lens CSV: contains `Lens ID` column header
- Dimensions CSV: contains `Dimensions ID` or `PubYear` header
- Cochrane CSV: contains `Cochrane` in header or record-like structure
- Zotero CSV: contains `Key` and `Item Type` columns
- RIS-format content in non-`.ris` files

For XML files, the parsed root distinguishes PubMed XML (`PubmedArticleSet`, `PubmedArticle`, `PubmedBookArticleSet`) from EndNote XML (`xml/records` or `records`). Unknown roots are unsupported; malformed XML raises a file-reading error.

For `.nbib` and `.enw`, the first 20 lines determine the content: RIS records are accepted, but MEDLINE text (`PMID-` in `.nbib`, `.txt` or files without extensions) and Refer text (`%` tags in `.enw`) are not supported. Export MEDLINE records as PubMed XML, or use `sm_fetch_pubmed()`; export Refer libraries as RIS or EndNote XML. A leading UTF-8 BOM is ignored for text format detection.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: `sm_read_bib()`, `sm_read_cochrane()`, `sm_read_dimensions()`, `sm_read_endnote()`, `sm_read_lens()`, `sm_read_openalex_json()`, `sm_read_pubmed_xml()`, `sm_read_ris()`, `sm_read_scopus()`, `sm_read_wos()`, `sm_read_zotero()`

Examples

```
## Not run:
corpus <- sm_read_auto("references.bib")
corpus <- sm_read_auto("exported_data.csv")

## End(Not run)
```

---

sm_read_bib *Read BibTeX files*

Description

Parse standard BibTeX (`.bib`) files into an `sm_corpus` object. Extracts entry type, title, author, year, DOI, journal, abstract, volume, pages, and keywords from each `@type{key, ...}` entry.

Usage

```
sm_read_bib(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
```

```
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to a .bib file.
encoding	Character scalar. File encoding (default "UTF-8").
engine	Character scalar. One of "native" (built-in parser), "bibliometrix" (delegate to bibliometrix::convert2df()), or "auto" (try bibliometrix first, fall back to native).
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser follows the BibTeX format specification as described in the original BibTeX documentation (Patashnik, 1988). Field values may be enclosed in braces or double-quotes and may span multiple lines. String concatenation with # is not supported; @string and @preamble entries are skipped.

Protective braces are removed from titles, journals, abstracts, keywords and DOIs; escaped braces remain literal. Common LaTeX accent commands (\", \', \^, \~, \=, \., \c, \v, \u, \H, \k, \r, and backslash followed by a grave accent) decode to precomposed letters where available, else the bare letter. Letter commands \ss, \o, \O, \aa, \AA, \ae, \AE, \oe, \OE, \l, \L and \i also decode, as do \&, \%, \$, \# and _. DOI cleaning only removes braces and decodes those five escapes. Ties (~), -- and math expressions remain as written. Whitespace outside math is collapsed and trimmed.

Only a lowercase and outside braces separates authors; a comma outside braces flips a name from surname-first to given-name-first. Braced corporate names stay together, and a trailing and others is retained. Quoted field values end only at a quote outside braces. The first isolated four-digit year (1000–2999) in year is used, falling back to the BibLaTeX date field when no such year is present.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_openalex_json(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_scopus(), sm_read_wos(), sm_read_zotero()

Examples

```
## Not run:
corpus <- sm_read_bib("references.bib")
corpus$works

## End(Not run)
```

`sm_read_cochrane` *Read Cochrane Library export files*

Description

Parse Cochrane Library (CDSR) export files into an `sm_corpus` object. Supports both CSV/tab-delimited exports and RIS-format exports from the Cochrane Library search interface.

Usage

```
sm_read_cochrane(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a Cochrane export file (`.csv`, `.tsv`, or `.ris`).
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`engine`	Character scalar. One of `"native"` (built-in parser), `"bibliometrix"` (delegate to `bibliometrix::convert2df()`), or `"auto"` (try bibliometrix first, fall back to native).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The Cochrane Library exports records in CSV or RIS format. The CSV export typically contains columns such as: Record Number (or #), Authors, Title, Source, Year, DOI, Abstract. RIS exports follow the standard RIS specification and are delegated to the RIS parser.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: `sm_read_auto()`, `sm_read_bib()`, `sm_read_dimensions()`, `sm_read_endnote()`, `sm_read_lens()`, `sm_read_openalex_json()`, `sm_read_pubmed_xml()`, `sm_read_ris()`, `sm_read_scopus()`, `sm_read_wos()`, `sm_read_zotero()`

Examples

```
## Not run:
corpus <- sm_read_cochrane("cochrane_export.csv")
corpus$works

## End(Not run)
```

---

sm_read_dimensions *Read Dimensions CSV files*

Description

Parse Dimensions CSV export files into an `sm_corpus` object. Handles the standard Dimensions analytics platform CSV export format.

Usage

```
sm_read_dimensions(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a Dimensions CSV file.
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`engine`	Character scalar. One of `"native"` (built-in parser), `"bibliometrix"` (delegate to `bibliometrix::convert2df()`), or `"auto"` (try bibliometrix first, fall back to native).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser reads the Dimensions CSV export using `readr::read_csv()`. Dimensions CSV files may begin with informational rows before the header; the parser skips lines that do not look like the header. Key columns matched (case-insensitive): Title, DOI, Authors, Source title (or Source Title), PubYear (or Publication Year), Times cited, Document Type (or Publication Type), Abstract, Dimensions ID, MeSH terms, Author Keywords, ISSN and Corresponding Authors.

The authors that the Corresponding Authors column names (;-separated, an affiliation in parentheses after a name is ignored) get `authorships$is_corresponding = TRUE` and the other authors of the record `FALSE`; names are compared by surname and given names or initials. A name that fits several authors equally, such as "Wang, Y" for Wang Yan and Wang Yu, leaves them `NA`. Without that column, or when it names none of the record's authors for certain, `is_corresponding` is `NA` (unknown).

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: `sm_read_auto()`, `sm_read_bib()`, `sm_read_cochrane()`, `sm_read_endnote()`, `sm_read_lens()`, `sm_read_openalex_json()`, `sm_read_pubmed_xml()`, `sm_read_ris()`, `sm_read_scopus()`, `sm_read_wos()`, `sm_read_zotero()`

Examples

```
## Not run:
corpus <- sm_read_dimensions("dimensions_export.csv")
corpus$works

## End(Not run)
```

---

sm_read_endnote *Read EndNote XML export files*

Description

Parse EndNote XML export files into an `sm_corpus` object. Uses xml2 to extract record metadata from the EndNote XML format.

Usage

```
sm_read_endnote(
  path,
  encoding = "UTF-8",
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to an EndNote XML file.
encoding	Character scalar. File encoding (default "UTF-8").
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The parser uses xml2::read_xml() to read the EndNote XML format. Each xml/records/record element (or records/record) is processed. The following fields are extracted:

- titles/title/style for the title
- contributors/authors/author/style for author names
- dates/year/style for year
- periodical/full-title/style or secondary-title/style for journal
- electronic-resource-num/style for DOI
- abstract/style for abstract
- ref-type attribute for document type
- keywords/keyword/style for keywords
- accession-num/style for record identifier

No bibliometrix engine is available since EndNote XML is not directly supported as a format by bibliometrix::convert2df().

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_lens(), sm_read_openalex_json(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_scopus(), sm_read_wos(), sm_read_zotero()

Examples

```
## Not run:
corpus <- sm_read_endnote("library.xml")
corpus$works

## End(Not run)
```

`sm_read_lens` *Read Lens.org CSV files*

Description

Parse Lens.org scholarly CSV export files into an `sm_corpus` object. Handles the standard Lens scholarly export format with columns for Lens ID, Title, Authors, DOI, Source Title, Abstract, and more.

Usage

```
sm_read_lens(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a Lens CSV file.
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`engine`	Character scalar. One of `"native"` (built-in parser), `"bibliometrix"` (delegate to `bibliometrix::convert2df()`), or `"auto"` (try bibliometrix first, fall back to native).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser reads the Lens.org scholarly CSV export using `readr::read_csv()`. Key columns matched (case-insensitive): Lens ID, Title, Date Published, Year Published (or Publication Year), DOI, Source Title, Authors, Abstract, Citing Works Count (or Scholarly Citations Count), Document Type (or Publication Type), Open Access Colour (or Color), Open Access Status, Is Open Access, or exactly Open Access (in that priority order; never a licence), MeSH Terms, Keywords, Source ISSN, Language.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: `sm_read_auto()`, `sm_read_bib()`, `sm_read_cochrane()`, `sm_read_dimensions()`, `sm_read_endnote()`, `sm_read_openalex_json()`, `sm_read_pubmed_xml()`, `sm_read_ris()`, `sm_read_scopus()`, `sm_read_wos()`, `sm_read_zotero()`

Examples

```
## Not run:
corpus <- sm_read_lens("lens_export.csv")
corpus$works

## End(Not run)
```

---

sm_read_openalex_json *Read OpenAlex JSON files*

Description

Parse OpenAlex JSON export files into an `sm_corpus` object. Handles JSON arrays of works, a saved API response (an object whose `results` hold the works, compact or pretty-printed), a single work object and newline-delimited JSON (JSONL, one work or one response per line). Extracts fields from the OpenAlex works schema including id, doi, title, publication year, type, cited_by_count, authorships, host venue, concepts, open access status, and referenced works.

The works also keep OpenAlex's field-weighted citation impact (`fwci`), its citation percentile (`citation_percentile`, 0 to 1, with the flags `is_top_10pct` and `is_top_1pct`) and the day of the record's `updated_date` (`citations_as_of`, `NA` when the file has none), all `NA` where a record lacks them, and the corpus gets the yearly citation counts (`counts_by_year`) as `citations_by_year` (see `sm_corpus()`). Each authorship's `raw_affiliation` holds its `raw_affiliation_strings` (joined with `"; "`), else the `raw_affiliation_string` of its `affiliations`, else the names of its institutions. Works are linked to their source (`works$source_id`), and sources are told apart by their OpenAlex id, else by their name.

Usage

```
sm_read_openalex_json(
  path,
  encoding = "UTF-8",
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to a .json or .jsonl file.
encoding	Character scalar. File encoding (default "UTF-8").
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The parser reads the OpenAlex works JSON schema as documented at https://developers.openalex.org/api-entities/works/work-object. The whole file is parsed first; only when that fails is it read as newline-delimited JSON, one value per line. No bibliometrix engine is available since OpenAlex is not a shared bibliometric export format.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

sm_fetch_openalex(), sm_enrich_openalex_impact()

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_scopus(), sm_read_wos(), sm_read_zotero()

Examples

```
path <- system.file("extdata", "example_openalex_inverted.json",
                    package = "scimapR")
corpus <- sm_read_openalex_json(path, verbose = FALSE)
corpus$works[c("title", "year", "cited_by_count")]
```

---

sm_read_pubmed_xml *Read PubMed XML files*

Description

Parse PubMed XML export files (NCBI DTD format) into an sm_corpus object. Extracts article metadata from PubmedArticle nodes including PMID, title, abstract, journal, authors, MeSH terms, DOI, and publication type.

Usage

```
sm_read_pubmed_xml(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to a PubMed XML file.
encoding	Character scalar. File encoding (default "UTF-8").
engine	Character scalar. One of "native" (built-in parser using xml2), "bibliometrix" (delegate to bibliometrix::convert2df()), or "auto" (try bibliometrix first, fall back to native).
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser uses xml2::read_xml() to parse the NCBI PubMed XML DTD. Each PubmedArticle element is processed to extract:

- MedlineCitation/PMID for the PubMed identifier
- MedlineCitation/Article/ArticleTitle for the title
- MedlineCitation/Article/Abstract/AbstractText for the abstract (multiple sections are concatenated)
- MedlineCitation/Article/Journal/Title and /ISSN for the journal
- MedlineCitation/Article/Journal/JournalIssue/PubDate/Year for year
- MedlineCitation/Article/AuthorList/Author for authors
- MedlineCitation/MeshHeadingList/MeshHeading for MeSH terms; the concept score is 1 for a major topic of the article (MajorTopicYN on the descriptor or one of its qualifiers) and 0 otherwise
- PubmedData/ArticleIdList/ArticleId[@IdType='doi'] for DOI
- MedlineCitation/Article/PublicationTypeList for document type

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_openalex_json(), sm_read_ris(), sm_read_scopus(), sm_read_wos(), sm_read_zotero()

Examples

```
## Not run:
corpus <- sm_read_pubmed_xml("pubmed_result.xml")
corpus$works

## End(Not run)
```

---

sm_read_ris *Read RIS files*

Description

Parse RIS (Research Information Systems) tag-value format files into an sm_corpus object. Supports the standard RIS specification with two-letter tags followed by two spaces, a dash, and a space.

Usage

```
sm_read_ris(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to a .ris file.
encoding	Character scalar. File encoding (default "UTF-8").
engine	Character scalar. One of "native" (built-in parser), "bibliometrix" (delegate to bibliometrix::convert2df()), or "auto" (try bibliometrix first, fall back to native).
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser follows the RIS format specification (Thomson Reuters). Each record starts with `TY  -` and ends with `ER  -`. Tags are two uppercase letters followed by two spaces, a hyphen, and a space. Repeating tags (AU, KW) generate multiple values per record.

Publication year comes from the first usable year in `PY`, `Y1`, then `DA`. Bare years and dates such as `2019/03/01/` or `2020///` are accepted: the first isolated four-digit year from 1000 through 2999 is used. `Y2` is never a fallback because it often holds an access or secondary date. Unrecognised dates yield `NA` without a coercion warning.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: `sm_read_auto()`, `sm_read_bib()`, `sm_read_cochrane()`, `sm_read_dimensions()`, `sm_read_endnote()`, `sm_read_lens()`, `sm_read_openalex_json()`, `sm_read_pubmed_xml()`, `sm_read_scopus()`, `sm_read_wos()`, `sm_read_zotero()`

Examples

```
## Not run:
corpus <- sm_read_ris("references.ris")
corpus$works

## End(Not run)
```

---

sm_read_scopus *Read Scopus CSV files*

Description

Parse Scopus CSV export files into an `sm_corpus` object. Handles the standard Scopus CSV format with columns for Authors, Title, Year, Source title, DOI, Abstract, and more.

Usage

```
sm_read_scopus(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

path	Character scalar. Path to a Scopus CSV file.
encoding	Character scalar. File encoding (default "UTF-8").
engine	Character scalar. One of "native" (built-in parser), "bibliometrix" (delegate to bibliometrix::convert2df()), or "auto" (try bibliometrix first, fall back to native).
verbose	Logical. Print progress messages?
call	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser reads the Scopus CSV export using readr::read_csv(). Scopus CSV column names may vary between export versions; the parser matches on known column name patterns. Key columns: Authors, Title, Year, Source title, DOI, Abstract, Document Type, Cited by, Author Keywords, Index Keywords, ISSN, Language of Original Document, EID.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_openalex_json(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_wos(), sm_read_zotero()

Examples

```
## Not run:
corpus <- sm_read_scopus("scopus.csv")
corpus$works

## End(Not run)
```

`sm_read_wos` *Read Web of Science export files*

Description

Parse Web of Science (WoS) plaintext or tab-delimited export files into an `sm_corpus` object, preserving authors, fields and cited references.

Usage

```
sm_read_wos(
  path,
  encoding = "UTF-8",
  engine = c("native", "bibliometrix", "auto"),
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a plaintext or tab-delimited export.
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`engine`	Character scalar. One of `"native"` (built-in parser), `"bibliometrix"` (delegate to `bibliometrix::convert2df()`), or `"auto"` (try bibliometrix first, fall back to native).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The native parser follows the Web of Science Core Collection export format. Tab-delimited files have a header containing PT, AU, TI and UT and one record per line. Cells are unquoted, so quotes in titles are preserved. Both layouts share field, author and reference-linking rules. Tab-delimited input always uses the native path with `engine = "native"` or `"auto"`; `engine = "bibliometrix"` rejects it. UTF-8 BOMs are accepted; UTF-16 exports can be read with `encoding = "UTF-16LE"`.

A file whose records all lack a title, accession number and author is an error. Such records among readable ones are dropped with one warning, including when `verbose = FALSE`. Files without any PT record still return an empty corpus.

Each record begins with `PT` (publication type) and ends with `ER`. Field tags are two uppercase letters followed by a single space. Continuation lines begin with three spaces. Tags parsed: AU or AF (authors), TI (title), SO (source), SN and EI (ISSN and electronic ISSN), J9 and JI (source abbreviations), AB (abstract), DI (DOI), PY (year), DT (document type), LA (language), TC (times cited),

UT (accession number, kept in provenance), PM (PubMed ID), C1 (addresses), RP (corresponding authors), CR (cited references), DE (author keywords), ID (Keywords Plus), WC (Web of Science categories) and SC (research areas). Other tags are not read.

Each distinct SO becomes a row of sources, with issn_l from SN (EI when SN is missing), issn holding SN and EI, and the abbreviations J9 and JI in the columns abbreviation and iso_abbreviation.

DE, ID, WC and SC become concepts, one row per ;-separated term, in the vocabularies "author-keywords", "keywords-plus", "wos-category" and "wos-subject-category" of sm_concept_vocabularies(). Terms keep the case of the file.

Author names are stored in the AU form, such as "J Smith" for "Smith, J", and authors$display_name_alternatives keeps the name as the file writes it, which tells a surname of several words such as "Garcia Lopez" from the initials.

Both parsers write one authorship row per author and address. The names in the brackets of C1 link authors to addresses: each name belongs to the author with the same full name (from AF), else to the one with the same surname and initials, else to the one with the same surname and first initial, so a short surname such as "He" never takes the address of "Chen". An address that C1 lists without names, as older records do, belongs to every author of the record, so each of them gets a row for it. An address whose names designate none of the authors belongs to the author of a single-author record and else to nobody.

The corresponding authors come from RP, which names them before "(corresponding author)" ("(reprint author)" in older files), in any case, with each name and its address on a line of its own or on one line. The authors these names designate, matched like the names of C1, get is_corresponding = TRUE on all their rows and the other authors FALSE. When a name fits several authors equally (namesakes with the same surname and initials), the one whose C1 addresses include the address RP gives with the name is meant; if that does not tell them apart, they get NA. A record without RP, or whose RP designates none of its authors for certain, gets NA (unknown) for all its authors; the native parser reports the number of such records when verbose = TRUE.

The "bibliometrix" engine (also used by "auto" when bibliometrix can read the file) converts the data frame of bibliometrix::convert2df() as as_sm_corpus() does: it reads the same tags into sources, references and concepts, tells authors apart by the full names of AF and links C1 and RP to the authors by the same rule.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_openalex_json(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_scopus(), sm_read_zotero()

Examples

```
## Not run:
corpus <- sm_read_wos("savedrecs.txt")
corpus$works
```

```
## End(Not run)
```

`sm_read_zotero` *Read Zotero CSV export files*

Description

Parse Zotero CSV export files into an `sm_corpus` object. Handles the standard Zotero CSV export format with columns for Key, Title, Author, Publication Year, DOI, Publication Title, Abstract, and Item Type.

Usage

```
sm_read_zotero(
  path,
  encoding = "UTF-8",
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

`path`	Character scalar. Path to a Zotero CSV file.
`encoding`	Character scalar. File encoding (default `"UTF-8"`).
`verbose`	Logical. Print progress messages?
`call`	Caller environment for error reporting.

Value

An sm_corpus object.

Implementation

The parser reads the Zotero CSV export format using `readr::read_csv()`. Zotero CSV exports have a standard set of column names. Key columns matched (case-insensitive): Key, Title, Author, Publication Year, DOI, Publication Title, Abstract Note, Item Type, ISSN, Language, Manual Tags, Automatic Tags. No bibliometrix engine is available since Zotero CSV is not a format supported by `bibliometrix::convert2df()`.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959–975. doi:10.1016/j.joi.2017.08.007

See Also

Other readers: sm_read_auto(), sm_read_bib(), sm_read_cochrane(), sm_read_dimensions(), sm_read_endnote(), sm_read_lens(), sm_read_openalex_json(), sm_read_pubmed_xml(), sm_read_ris(), sm_read_scopus(), sm_read_wos()

Examples

```
## Not run:
corpus <- sm_read_zotero("zotero_export.csv")
corpus$works

## End(Not run)
```

---

sm_reconcile *Reconcile two corpora by DOI and title*

Description

Computes a symmetric set difference between two corpora (or coercible data frames) using the shared DOI-then-title matching engine. This generalises sm_diff_corpora(), which compares strictly by internal work_id: sm_reconcile() instead matches on *content* (normalised DOI with a fuzzy title fallback), so it works across corpora from different sources that do not share identifiers.

Usage

```
sm_reconcile(
  corpus_a,
  corpus_b,
  match = c("doi_then_title", "doi", "title"),
  threshold = 0.9,
  call = rlang::caller_env()
)

## S3 method for class 'sm_reconciliation'
print(x, ...)

## S3 method for class 'sm_reconciliation'
summary(object, ...)

## S3 method for class 'sm_reconciliation'
autoplot(object, ...)
```

Arguments

corpus_a, corpus_b
: An sm_corpus or a data frame with DOI and/or title columns (see sm_coverage_audit() for accepted column aliases and for how the rows of a data frame are identified).

match	Matching strategy: "doi_then_title" (default), "doi", or "title".
threshold	Minimum Jaro-Winkler title similarity ([0, 1]) to accept a title match (default 0.9). Titles only match between records published at most one year apart (when both years are known) and never between two records with different DOIs.
call	Caller environment for error reporting.
x	An sm_reconciliation object.
...	Ignored.
object	An sm_reconciliation object.

Value

An sm_reconciliation S3 object with components:

in_both Tibble of matched pairs: a_id, b_id, title, match_type, match_score.

only_a Tibble of corpus_a records absent from corpus_b (id, doi, title, year).

only_b Tibble of corpus_b records absent from corpus_a.

matches Match provenance tibble (a_id, b_id, match_type, match_score), the same shape as sm_coverage_audit()'s.

summary One-row tibble with counts.

print returns x invisibly.

summary returns the one-row summary tibble.

autoplot returns a ggplot set-size bar chart.

See Also

sm_coverage_audit(), sm_diff_corpora()

Other coverage: sm_coverage_audit(), sm_coverage_breakdowns(), sm_journal_in_index(), sm_match_types()

Examples

```
a <- sm_example_corpus(n_works = 20, seed = 1)
b <- sm_example_corpus(n_works = 20, seed = 1)[5:20]
rec <- sm_reconcile(a, b)
rec
```

`sm_refresh` *Refresh stale corpus data*

Description

Re-fetches metadata for works whose `last_refreshed` timestamp is older than `max_age_days`. For each stale work, the original provenance source is consulted: OpenAlex supplies citation counts and open-access status; Crossref supplies citation counts. Other sources remain stale. Failed requests, missing identifiers and records not returned are reported. Only works with returned updates receive a new timestamp and provenance row. `metadata$last_refresh` advances only when at least one work was refreshed and none failed. Enrichment rows never select a refresh source.

The operation is **idempotent**: running it twice in succession will not re-fetch works that were just refreshed. The function **refuses to operate on a locked corpus**; use `sm_unlock()` first.

Usage

```
sm_refresh(
  corpus,
  max_age_days = 30,
  what = c("citations", "oa_status"),
  sources = NULL,
  verbose = TRUE,
  api_key = Sys.getenv("OPENALEX_API_KEY"),
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  call = rlang::caller_env()
)
```

Arguments

`corpus`	An `sm_corpus` object.
`max_age_days`	Numeric. Works last refreshed more than this many days ago will be re-fetched. Default `30`.
`what`	Character vector of data aspects to refresh. One or more of `"citations"` and `"oa_status"` (the default). `"new_refs"` and `"altmetric"` are accepted but unsupported: they warn and are ignored, or error when no implemented aspect remains.
`sources`	Optional character vector restricting which provenance sources to re-query (e.g., `"openalex"`, `"crossref"`). If `NULL` (the default), the original source for each work is used.
`verbose`	Logical. Print progress messages?
`api_key`	OpenAlex API key, sent only in the authorization header.
`mailto`	Contact address for Crossref requests.
`call`	Caller environment for error reporting.

Value

A refreshed sm_corpus with updated fields and appended provenance rows.

See Also

Other refresh: sm_lock(), sm_staleness()

Examples

```
corpus <- sm_example_corpus()
# In practice, this would re-fetch from APIs:
# refreshed <- sm_refresh(corpus, max_age_days = 7)
```

---

sm_register_input *Register a local input file of a corpus*

Description

Record a local file that a corpus was built from, such as a database snapshot, an institutional export, a publication tracker or an affiliation dictionary, together with its SHA-256 hash. sm_certificate() lists the registered files in its local_inputs block, and sm_verify_certificate() checks that each of them still has the hash recorded here. sm_corpus_from_tables() registers the files given as its inputs.

Usage

```
sm_register_input(
  corpus,
  path,
  role = NULL,
  builder = NULL,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object. A locked corpus (sm_lock()) cannot be changed.
path	Path of the file.
role	NULL or a string: what the file is.
builder	NULL or a string: the code that read the file.
call	Caller environment for error reporting.

Details

The file is hashed as a stream (`digest::digest(path, algo = "sha256", file = TRUE)`), so large snapshots are not read into memory. The function adds one row to `corpus$metadata$local_inputs`, a tibble with these columns:

Column	Type	Content
`path`	character	The path, relative or as given (see below)
`path_abs`	character	The absolute path of the file
`root`	character	The working directory at registration
`basename`	character	The file name
`sha256`	character	SHA-256 hash of the content
`bytes`	double	Size in bytes
`role`	character	What the file is, such as `"dictionary"`
`builder`	character	The code that read it
`registered_at`	POSIXct	When it was registered (UTC, whole seconds)

`path` is relative to the working directory when the file lies below it, and as given otherwise. `role` names the file's part in the corpus, such as `"source database"` or `"publication tracker"`, and `builder` the code that read it, such as `"R/02_corpus.R::build_corpus"`.

Registering the same file again replaces its row; paths are compared after they are made absolute, so `"data/x.csv"` and `"./data/x.csv"` name the same file. A corpus without registered files has no such element; `sm_certificate()` then writes an empty block.

Local inputs are metadata: they do not enter the corpus hash (`sm_hash_corpus()`), and `sm_rebuild_from_cert()` cannot replay them. A certificate records `path_abs` and `root` as well as `path`, so that `sm_verify_certificate()` finds the files on the same computer from any folder; these absolute paths show the folder names of that computer.

Value

The corpus with the file recorded in `corpus$metadata$local_inputs`.

See Also

`sm_certificate()`, `sm_verify_certificate()`

Other reproducibility: `sm_certificate()`, `sm_cite_corpus()`, `sm_diff_corpora()`, `sm_hash_corpus()`, `sm_provenance()`, `sm_snapshot()`

Examples

```
corpus <- sm_example_corpus(n_works = 10)
tracker <- tempfile(fileext = ".csv")
writeLines(c("doi,unit", "10.9999/example.1,Unit A"), tracker)
corpus <- sm_register_input(corpus, tracker,
                            role = "publication tracker",
                            builder = "R/01_import.R::read_tracker")
corpus$metadata$local_inputs
```

`sm_researcher_profile` *Build a researcher profile*

Description

One-shot workflow: fetch all works for a researcher by ORCID, enrich, embed, and build a trajectory analysis.

Usage

```
sm_researcher_profile(
  orcid,
  sources = c("openalex"),
  enrich = c("oa_status"),
  embed = FALSE,
  trajectory = TRUE,
  mailto = Sys.getenv("SCIMAPR_MAILTO"),
  n_max = 200L
)
```

Arguments

`orcid`	ORCID identifier.
`sources`	API sources to query.
`enrich`	Enrichment steps to run.
`embed`	Logical; compute embeddings?
`trajectory`	Logical; build trajectory analysis?
`mailto`	Contact address sent to OpenAlex (see `sm_fetch_openalex()`).
`n_max`	Maximum number of works to fetch (default `200`, as in `sm_fetch_openalex()`). A researcher with more works gets a profile of the first `n_max` only, so raise it for long careers.

Value

An `sm_corpus` with trajectory attached as attribute.

See Also

Other trajectory: `is_sm_trajectory()`, `sm_author_trajectory()`, `sm_plot_collab_turnover()`, `sm_plot_topic_pivots()`, `sm_plot_trajectory()`, `sm_trajectory`

Examples

```
## Not run:
# profile <- sm_researcher_profile("0000-0001-8006-9742")

## End(Not run)
```

sm_run_app *Launch the scimapR Shiny application*

Description

Launch a comprehensive 13-tab interactive explorer for bibliometric analysis. The app provides auto/light/dark theming, publication-ready export in multiple formats, and viridis colour palettes throughout.

Usage

```
sm_run_app(corpus = NULL, launch.browser = TRUE, ...)
```

Arguments

corpus	An sm_corpus to load on startup. If NULL, loads sm_example_corpus() so users can explore without their own data.
launch.browser	Passed to shiny::runApp().
...	Additional arguments passed to shiny::runApp().

Value

Invisible NULL. Launches a Shiny application.

Examples

```
if (interactive()) {
  sm_run_app()
  # sm_run_app(my_corpus)
}
```

sm_save_corpus *Save and load an sm_corpus*

Description

Persist a corpus to disk as RDS and reload it.

Usage

```
sm_save_corpus(corpus, path, compress = "xz")

sm_load_corpus(path)
```

Arguments

corpus	An sm_corpus object.
path	File path for saving.
compress	Compression type passed to saveRDS().

Value

For sm_save_corpus, the path invisibly. For sm_load_corpus, an sm_corpus object.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_validate(), validate_sm_corpus()

Examples

```
corpus <- sm_example_corpus(n_works = 10)
path <- tempfile(fileext = ".rds")
sm_save_corpus(corpus, path)
loaded <- sm_load_corpus(path)
nrow(loaded$works)
```

---

sm_scale_color *Viridis colour scale for scimapR*

Description

Wraps viridis with sensible defaults for scimapR plots.

Usage

```
sm_scale_color(option = "viridis", discrete = TRUE, ...)

sm_scale_fill(option = "viridis", discrete = TRUE, ...)
```

Arguments

option	Viridis palette option. One of "viridis", "magma", "plasma", "cividis", "inferno", "mako", "rocket", "turbo".
discrete	Logical; discrete or continuous scale?
...	Additional arguments passed to the viridis scale function.

Value

A ggplot2 scale object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(),
sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(),
sm_plot_top(), sm_theme()

---

sm_screen_against_question

Screen corpus against a research question

Description

sm_screen_against_question() uses a chat provider to screen each work in a corpus against a structured research question. For each work, the provider is asked to classify the title/abstract as "include", "exclude", or "uncertain", with a confidence score and brief reason. Results are written to the corpus's screening table.

sm_screen_regex() provides a deterministic fallback using regular-expression matching on titles and abstracts.

sm_screen_summary() returns a count summary of screening decisions by stage.

Usage

```
sm_screen_against_question(
  corpus,
  question,
  stages = c("title-abstract", "full-text"),
  llm = NULL,
  batch_size = 10L,
  include_uncertain = TRUE,
  verbose = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
question	An sm_question object.
stages	Character vector of screening stages to run. One or more of "title-abstract" and "full-text", run in that order. The full-text stage still screens titles and abstracts using its instruction: no full texts are fetched.
llm	An ellmer chat provider object (e.g., from ellmer::chat_openai()). Required: no default provider is created. Titles and abstracts are sent to this provider. Each request uses a fresh copy: the object is unchanged, and its system prompt and earlier turns are not used.
batch_size	Integer. Number of works to send per request.

include_uncertain
: Logical. If TRUE, works classified as "uncertain" by the latest successful title-abstract decision are carried forward. Failed provider rows are retried and never carried forward.

verbose
: Logical. Print progress?

call
: Caller environment for error reporting.

Value

A modified sm_corpus with updated screening table, including the provider model on each new row. Failed batches warn and are retried on the next run, replacing their earlier failed rows.

See Also

Other question: is_sm_question(), sm_corpus_for_question(), sm_question(), sm_screen_regex(), sm_screen_summary()

Examples

```
corpus <- sm_example_corpus()
q <- sm_question(
  text = "spatial transcriptomics in cancer",
  framework = "free"
)
# Deterministic regex screening (no provider needed):
screened <- sm_screen_regex(
  corpus, include_terms = c("spatial", "transcriptom")
)
sm_screen_summary(screened)
```

---

sm_screen_prisma *Generate PRISMA flow diagram*

Description

Create a PRISMA 2020-compatible flow diagram from corpus screening data.

Usage

```
sm_screen_prisma(
  corpus,
  decisions = NULL,
  stages = c("identification", "title-abstract", "full-text", "inclusion"),
  n_identified = NULL
)
```

Arguments

corpus	An sm_corpus with screening decisions.
decisions	Optional external decisions tibble.
stages	Stage names in flow order. Defaults to identification, title-abstract, full-text and inclusion. Only current works still in the flow count; each stage uses the latest decision per work.
n_identified	Whole count before deduplication and other removals, at least the corpus size. Defaults to metadata$n_identified when set, otherwise the corpus size. Ignored without an identification stage.

Value

A list with $counts (tibble) and $plot (ggplot).

See Also

Other screening: sm_export_covidence(), sm_export_rayyan(), sm_import_rayyan(), sm_merge_screening_decisio

Examples

```
corpus <- sm_example_corpus(with_screening = TRUE)
prisma <- sm_screen_prisma(corpus)
prisma$counts
```

---

sm_screen_regex *Screen corpus using regex matching*

Description

A deterministic screening method that uses regular expression matching on work titles and abstracts. Works matching any include_terms (case-insensitive) and none of the exclude_terms are classified as "include"; works matching an exclude term are "exclude"; works matching no include term are "exclude".

Usage

```
sm_screen_regex(
  corpus,
  include_terms,
  exclude_terms = NULL,
  unicode = TRUE,
  call = rlang::caller_env()
)
```

Arguments

corpus	An sm_corpus object.
include_terms	Character vector of regex patterns. A work must match at least one to be included.
exclude_terms	Character vector of regex patterns. A work matching any is excluded. Default NULL (no exclusions).
unicode	Logical (default TRUE). Match the terms in Unicode mode, so \w, \b, \d, \s and POSIX classes know non-ASCII letters, digits and spaces, as in German or French titles. FALSE gives ASCII-only classes.
call	Caller environment for error reporting.

Details

Terms are Perl-compatible regular expressions (PCRE), matched as written and case-insensitively against the title and abstract of each work. In Unicode mode (the default) each term is prefixed with the PCRE item (*UCP) unless its leading items already include it, so \b finds the edge of a word that starts with an umlaut and \w consumes one. A term can therefore also stop matching: one that used \W, \S, \D or \B to stand for a non-ASCII letter no longer matches it, and \b no longer finds an edge between an ASCII letter and a non-ASCII letter next to it, as inside a German compound word. Write [A-Za-z0-9_] instead of \w for an ASCII-only class.

Value

A modified sm_corpus with updated screening table.

See Also

Other question: is_sm_question(), sm_corpus_for_question(), sm_question(), sm_screen_against_question(), sm_screen_summary()

Examples

```
corpus <- sm_example_corpus(n_works = 20, n_authors = 5)
screened <- sm_screen_regex(corpus,
                            include_terms = c("spatial", "transcriptom"),
                            exclude_terms = "\\bmouse\\b")
sm_screen_summary(screened)
```

---

sm_screen_summary *Summarise screening decisions*

Description

Returns a summary tibble of screening decisions by stage, showing the count of included, excluded, and uncertain works at each stage.

Usage

```
sm_screen_summary(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with columns stage, decision, n, and pct.

See Also

Other question: is_sm_question(), sm_corpus_for_question(), sm_question(), sm_screen_against_question(), sm_screen_regex()

---

sm_self_citation *Compute self-citation from corpus reference lists*

Description

Identifies self-citations from the references already in the corpus — no per-citation API calls. A citation from a citing work to a cited work is a self-citation at the chosen level when the citing and cited works share an author (or institution). This is the quota-light "reference overlap" method.

Usage

```
sm_self_citation(
  corpus,
  level = c("author", "institution"),
  method = c("reference_overlap"),
  call = rlang::caller_env()
)

## S3 method for class 'sm_self_citation'
print(x, ...)

## S3 method for class 'sm_self_citation'
summary(object, ...)
```

Arguments

corpus	An sm_corpus with a populated references sub-tibble whose cited_work_id links to corpus works.
level	"author" (default) or "institution".
method	Self-citation definition; currently only "reference_overlap".
call	Caller environment for error reporting.
x	An sm_self_citation object.
...	Ignored.
object	An sm_self_citation object.

Value

An sm_self_citation S3 object (a list) with components:

by_entity Tibble: entity_id, n_citations_received (internal citations to the entity's works), n_self_citations, self_citation_share.

by_work Tibble: cited_work_id, n_citations_received, n_self_citations, self_citation_share (per cited work).

provenance Tibble: citing_work_id, cited_work_id, and shared_author_id (author level) or shared_institution_id (institution level) — the evidence behind each self-citation.

Type-stable: when references is absent/empty the components are 0-row tibbles with the documented columns, returned after a cli::cli_warn (the function never spins).

print returns x invisibly.

summary returns the by_entity tibble.

See Also

sm_metric_h_index(), sm_link_references()

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_g_index(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_summary_authors(), sm_summary_period(), sm_summary_sources(), sm_summary_works()

Examples

```
corpus <- sm_example_corpus(n_works = 40, seed = 1)
sc <- sm_self_citation(corpus, level = "author")
head(sc$by_entity)
```

sm_snapshot *Save and load corpus snapshots*

Description

sm_snapshot() serializes an sm_corpus to disk as a compressed RDS file, embedding the corpus hash as part of the filename for traceability.

sm_snapshot_load() reads a previously saved snapshot and validates that the loaded object is a well-formed sm_corpus.

The snapshot stores the corpus hash (sm_hash_corpus()) in metadata$corpus_hash and the version of the hash algorithm in metadata$hash_version. sm_snapshot_load() recomputes the hash with that version, or with version 1 for snapshots written before scimapR 0.5.0, and warns when it differs.

Usage

```
sm_snapshot(
  corpus,
  path = NULL,
  compress = c("xz", "gzip", "bzip2", "none"),
  call = rlang::caller_env()
)

sm_snapshot_load(path, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
path	Character. File path for the snapshot. If NULL, a path is generated in the current working directory using the corpus hash.
compress	Character. Compression method passed to saveRDS(). One of "xz" (default, best compression), "gzip", "bzip2", or "none".
call	Caller environment for error reporting.

Value

For sm_snapshot(): the file path (invisibly). For sm_snapshot_load(): an sm_corpus object.

See Also

Other reproducibility: sm_certificate(), sm_cite_corpus(), sm_diff_corpora(), sm_hash_corpus(), sm_provenance(), sm_register_input()

Examples

```
corpus <- sm_example_corpus()
path <- tempfile(fileext = ".rds")
sm_snapshot(corpus, path = path)

loaded <- sm_snapshot_load(path)
identical(nrow(corpus$works), nrow(loaded$works))
```

`sm_staleness` *Check corpus staleness*

Description

Inspects each work in a corpus and reports how long ago it was last refreshed. Works with `last_refreshed` older than 30 days are flagged as stale. This is useful before calling `sm_refresh()` to preview which works would be updated.

Usage

```
sm_staleness(corpus, threshold_days = 30, call = rlang::caller_env())
```

Arguments

`corpus`	An `sm_corpus` object.
`threshold_days`	Numeric. Number of days after which a work is considered stale. Defaults to 30.
`call`	Caller environment for error reporting.

Value

A tibble with columns:

work_id Work identifier.

last_refreshed POSIXct timestamp of last refresh.

age_days Number of days since last refresh.

is_stale Logical; `TRUE` if `age_days > threshold_days`.

See Also

Other refresh: `sm_lock()`, `sm_refresh()`

Examples

```
corpus <- sm_example_corpus()
sm_staleness(corpus)
```

sm_summary_authors *Summary statistics for authors*

Description

Computes aggregate summary statistics for authors in the corpus.

Usage

```
sm_summary_authors(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with one row containing: n_authors, n_with_orcid, pct_orcid, mean_works_per_author, median_works_per_author, max_works_per_author, mean_authors_per_work, single_author_pct. The works of an author and the authors of a work are counted as distinct pairs of work_id and author_id, so an author with several address rows in a work counts once, as in sm_metric_collab_index().

See Also

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_g_index(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_self_citation(), sm_summary_period(), sm_summary_sources(), sm_summary_works()

Examples

```
corpus <- sm_example_corpus()
sm_summary_authors(corpus)
```

sm_summary_period *Summary statistics by publication period*

Description

Computes per-year summary statistics for works in the corpus.

Usage

```
sm_summary_period(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with one row per year and columns: year, n_works, mean_citations, median_citations, total_citations, n_oa, pct_oa, n_authors (unique authors active in that year), mean_authors_per_work (distinct authors per work, so an author with several address rows counts once).

Open-access percentages use sm_oa_statuses() and all works in each year as the denominator, agreeing with sm_audit_oa() to one decimal.

See Also

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_g_index(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_self_citation(), sm_summary_authors(), sm_summary_sources(), sm_summary_works()

Examples

```
corpus <- sm_example_corpus()
sm_summary_period(corpus)
```

---

sm_summary_sources *Summary statistics for sources (journals)*

Description

Computes aggregate summary statistics for publication sources in the corpus.

Usage

```
sm_summary_sources(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with one row containing: n_sources, n_oa_sources, pct_oa_sources, n_publishers, top_source (source with most works), top_source_n.

See Also

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_g_index(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_self_citation(), sm_summary_authors(), sm_summary_period(), sm_summary_works()

Examples

```
corpus <- sm_example_corpus()
sm_summary_sources(corpus)
```

---

sm_summary_works *Summary statistics for works*

Description

Computes aggregate summary statistics for all works in the corpus.

Usage

```
sm_summary_works(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with one row containing summary statistics: n_works, year_min, year_max, year_span, mean_citations, median_citations, total_citations, n_types (unique document types), n_sources (unique journals/sources), n_oa (open access works), pct_oa, n_retracted, n_languages.

Open-access percentages use sm_oa_statuses() and all works as the denominator, agreeing with sm_audit_oa() to one decimal place.

See Also

Other metrics: sm_metric_collab_index(), sm_metric_disruption(), sm_metric_fnci(), sm_metric_g_index(), sm_metric_h_index(), sm_metric_m_index(), sm_metric_novelty(), sm_metric_rcr(), sm_self_citation(), sm_summary_authors(), sm_summary_period(), sm_summary_sources()

Examples

```
corpus <- sm_example_corpus()
sm_summary_works(corpus)
```

sm_synth *Synthetic control for a treated institution*

Description

Builds the unit-by-time outcome panel for a treated institution against a donor pool and estimates a synthetic control using the optional **tidysynth** package. The dependency is kept in Suggests; when it is not installed an informative error explains how to install it.

Usage

```
sm_synth(
  corpus,
  treated,
  donors,
  intervention_year,
  outcome = c("count", "share_q1", "cnci", "leadership"),
  lag = NULL,
  institution_aware = FALSE,
  lead_role = c("corresponding", "first", "last", "first_or_last", "any_lead"),
  min_cell = NULL,
  call = rlang::caller_env()
)

## S3 method for class 'sm_synth'
print(x, ...)
```

Arguments

corpus	An sm_corpus with institution-attribution columns.
treated	A single institution label (the treated unit).
donors	Character vector of donor-pool institution labels.
intervention_year	Year the intervention takes effect: one whole number such as 2020, or text holding one.
outcome	One of "count", "share_q1", "cnci", "leadership".
lag	Citation-maturity lag in years (default 2), as in sm_its(): for "cnci" the most recent lag years are citation-immature and excluded from the panel.
institution_aware	For outcome = "leadership": if TRUE, each unit's share counts the works whose author in the role lead_role belongs to the unit itself (its label is the focal institution, as institutions in sm_its()); the treated unit and every donor use their own label. FALSE (default) counts the works with a corresponding author.
lead_role	For outcome = "leadership": the author role that leads a work, as in sm_its(). Roles other than "corresponding" need institution_aware = TRUE.

min_cell	For outcome = "cnci": NULL (default) or a positive integer, passed to sm_metric_fnci() when the impact is derived from citation counts, as in sm_its().
call	Caller environment for error reporting.
x	An sm_synth object.
...	Ignored.

Details

Each unit's yearly series is built as in sm_its(), and every unit must contain works. For "count" a unit's years without works count 0, over every year from the first to the last publication year of the corpus, so no year is dropped. For the other outcomes the panel stays balanced: a year in which any unit has no works or no usable value is dropped for every unit, with a warning, and listed in dropped_years. For "cnci" the citation-immature years are then excluded for every unit, by the same rule as in sm_its(), and listed in provisional_years; up to scimapR 0.4.0 sm_synth() kept them. At least 4 panel years must remain, 2 or more before intervention_year and 1 or more from it on; otherwise sm_synth() aborts. For "leadership" the shares leave out the works whose lead is unknown, as in sm_its().

Value

An sm_synth S3 object wrapping the initialised tidysynth object (synth), the long panel (panel: unit, year, value), dropped_years (tibble: unit, year; the unit-years without a usable value, whose years were dropped for every unit), provisional_years (integer; the citation-immature years excluded for "cnci"), leadership (the leadership definition, as in sm_its(), with institutions the unit labels when institution_aware; NULL for the other outcomes), impact_source (where the impact values of "cnci" come from, as in sm_its(); NA for the other outcomes), and metadata.

print returns x invisibly.

See Also

sm_did()

Other causal: sm_did(), sm_its()

Examples

```
corpus <- sm_example_corpus(n_works = 300, seed = 1)
corpus$authorships$institution_name <- sample(
  c("A", "B", "C", "D"), nrow(corpus$authorships), replace = TRUE)
synth <- sm_synth(corpus, treated = "A", donors = c("B", "C", "D"),
                  intervention_year = 2020, outcome = "count")
```

sm_theme *scimapR plot theme*

Description

A restrained, publication-ready ggplot2 theme with viridis defaults.

Usage

```
sm_theme(base_size = 11, base_family = "", dark = FALSE, ...)
```

Arguments

base_size	Base font size in points.
base_family	Base font family.
dark	Logical; use dark mode?
...	Additional arguments passed to ggplot2::theme().

Value

A ggplot2::theme object.

See Also

Other plots: autoplot.sm_corpus(), sm_palette_qualitative(), sm_plot_bradford(), sm_plot_citation_network(
sm_plot_collab(), sm_plot_equity_dashboard(), sm_plot_evolution(), sm_plot_heaps(),
sm_plot_landscape(), sm_plot_lotka(), sm_plot_production(), sm_plot_thematic_map(),
sm_plot_top(), sm_scale_color()

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) + geom_point() + sm_theme()
```

sm_to_bibliometrix *Convert sm_corpus to bibliometrix format*

Description

Converts an sm_corpus to the data frame format used by bibliometrix::biblioAnalysis() and the other functions of bibliometrix: one row per work and one column per Web of Science field tag. This enables using bibliometrix's analysis functions on corpora built with scimapR.

Usage

```
sm_to_bibliometrix(
  corpus,
  keep_extra = TRUE,
  db = c("auto", "ISI", "scimapR"),
  ...
)
```

Arguments

corpus	An sm_corpus object.
keep_extra	Logical. If TRUE (default), the columns without a field tag follow the tag columns, as in earlier versions: the other works columns (such as work_id and source_id), country_code and SO_ISSN. FALSE returns the tag columns only.
db	Value of the DB column. "auto" (default) writes "ISI" when every work comes from Web of Science and "scimapR" otherwise; "ISI" and "scimapR" write that value. See Details.
...	Additional arguments (currently unused).

Details

Field tags:

- AU: authors as SURNAME INITIALS, AF: full names as SURNAME, GIVEN NAMES; one entry per author of the work, in position order and separated by ;, from authors$display_name through authorships. An author with several address rows appears once; authors without a name are left out.
- TI, AB: works$title, works$abstract; TI_raw and AB_raw hold them in their stored case, as bibliometrix::convert2df() does.
- SO, J9, JI: sources$display_name, and the optional columns sources$abbreviation and sources$iso_abbreviation.
- SN, EI: sources$issn_l (else the first value of sources$issn) and the next distinct value of sources$issn.
- C1: one segment per distinct address of the work, [AUTHOR A; AUTHOR B] ADDRESS, with the authors (AF form) in position order, joined by ; ; from authorships$raw_affiliation.
- RP: one segment per corresponding author (authorships$is_corresponding), SURNAME, INITIALS (CORRESPONDIN with the author's first address, joined by ; .
- EM: the e-mail addresses of the optional column authorships$email.
- CR: references$cited_raw in ref_index order, separated by ;; NR: the number of items in CR.
- DE: author keywords, see below; DE_raw holds them in their stored case. ID: the concepts of the indexer_keyword family of sm_concept_vocabularies() (such as Keywords Plus or MeSH). WC: the vocabulary wos-category (Web of Science categories); SC: the vocabulary wos-subject-category (research areas). Terms are joined by ; .
- DT, LA, PY, TC, DI, PM: works$type, language, year, cited_by_count, doi and pmid.
- UT: the Web of Science accession number, provenance$source_id_external of the work's first provenance row with source == "wos", else a work_id that starts with WOS:.

- DB: see below. SR: the short reference FIRST AUTHOR, YEAR, J9 (with SO where J9 is missing), made unique by appending -a, -b, ... to repeats; SR_FULL keeps it without the suffixes.

With keep_extra = TRUE, country_code holds the country codes of authorships$country_code, one per authorship row (author and address), and SO_ISSN holds sources$issn_l. The export has no AU_CO column, as data from bibliometrix::convert2df() has none: bibliometrix::metaTagExtraction(M, "AU_CO") adds the country names that bibliometrix reads from C1 and uses for its country analyses.

Conventions:

Like bibliometrix::convert2df() for Web of Science files, the export upper-cases AU, AF, TI, SO, J9, JI, AB, C1, RP, EM, CR (including the DOIs in it), DE, ID, WC, SC, DT and LA. DI, UT, SN, EI and PM are written as stored, so DI keeps the lower-case DOIs of the scimapR readers. Upper-casing uses toupper(), whose result for non-ASCII letters depends on the locale.

Names: a display_name with a comma is read as Surname, Given names. Otherwise an entry of authors$display_name_alternatives in that form that agrees with the name gives the surname, so a surname of several words such as "Garcia Lopez" stays whole; sm_read_wos() keeps the names of the file there. Else initials at the end of the name (single letters, or capitals with periods, as in "Chen W." from sm_read_scopus()) follow the surname, and otherwise the last word is the surname, together with the lower-case particles before it ("Anna van der Berg" gives VAN DER BERG A). Particles never become initials. Without such a spelling, a surname of several words without particles ("Maria Garcia Lopez") keeps only its last word, and a name such as "SMITH JA" is read as given names followed by the surname. The given names of AF are the longest spelling at hand, the initials of AU the shortest. Corpora read by sm_read_wos() store the AU form of a name (such as "J Smith"), so their AF holds initials.

CR items are written in the Web of Science form SURNAME INITIALS, YEAR, SOURCE, ...: an item Surname, IN, YEAR, ... (with a comma after the surname, as some exports write it) becomes SURNAME IN, YEAR, ... Other items, such as titles or OpenAlex IDs, are only upper-cased. Because ; separates the items of AU, AF, C1, RP, CR, DE, ID, WC and SC, a ; inside an item becomes ,.

DE holds the author keywords of each work: the concepts of the author_keyword family of sm_concept_vocabularies() (author-keywords, manual-tags). A work without author keywords gets its OpenAlex concepts (openalex) instead, so corpora built from OpenAlex keep their DE; other vocabularies (Keywords Plus, MeSH, categories, arXiv and bioRxiv subjects) never enter DE. A work with neither has DE = NA.

DB: with db = "auto", a corpus whose works all come from Web of Science (each has a provenance row with source == "wos", or a work_id that starts with WOS:) gets "ISI", the value bibliometrix::convert2df() writes for Web of Science files, so that bibliometrix::histNetwork() and bibliometrix::rpys() accept it. Other corpora get "scimapR", because bibliometrix reads "ISI" data as Web of Science records; db = "ISI" declares a corpus whose CR items are in Web of Science form anyway.

RP follows authorships$is_corresponding: it names the authors flagged TRUE and is NA for a work without one. sm_read_wos() takes the corresponding authors from the RP field of the file, sm_read_openalex_json() and sm_fetch_openalex() from the OpenAlex records, sm_read_dimensions() from the Corresponding Authors column, sm_fetch_biorxiv() from the bioRxiv API and as_sm_corpus() from RP. The other readers and fetchers record the corresponding author as unknown (NA), so their corpora export RP = NA. Corpora built from tables keep the flags of their tables.

as_sm_corpus() reads the result back: the round trip keeps the works, their DOIs and years, the authors of each work, their addresses and the corresponding authors.

Value

A data frame with class c("bibliometrixDB", "data.frame"), one row per work in the order of corpus$works, with SR as row names. Every tag column is present, NA where the corpus holds no data.

References

Aria, M. & Cuccurullo, C. (2017). bibliometrix: An R-tool for comprehensive science mapping analysis. *Journal of Informetrics*, 11(4), 959-975. doi:10.1016/j.joi.2017.08.007

Examples

```
corpus <- sm_example_corpus(n_works = 10, n_authors = 5,
                            with_embeddings = FALSE)
M <- sm_to_bibliometrix(corpus, keep_extra = FALSE)
class(M)
M[1:3, c("AU", "PY", "SO", "DB")]

# a corpus from Web of Science records
fix <- readRDS(sm_example_files("example_realuse_corpus.rds"))
M <- sm_to_bibliometrix(fix)
unique(M$DB)
M$C1[1]

if (requireNamespace("bibliometrix", quietly = TRUE)) {
  results <- bibliometrix::biblioAnalysis(M, sep = ";")
}
```

---

sm_trajectory *sm_trajectory S3 class*

Description

An sm_trajectory object encodes the career trajectory of a single author across a corpus, including career stages, topic pivots, collaborator turnover, emerging collaborators, h-index curve, novelty curve, and citation acceleration.

The object is created by sm_author_trajectory() and should not normally be constructed directly.

See Also

Other trajectory: is_sm_trajectory(), sm_author_trajectory(), sm_plot_collab_turnover(), sm_plot_topic_pivots(), sm_plot_trajectory(), sm_researcher_profile()

sm_validate *Validate corpus integrity*

Description

Check referential integrity across all corpus tables and report issues. Checks work, author, source and institution links, duplicate lookup IDs, and duplicate or orphan embedding row names. A locked corpus with a stored hash is also checked for changed content.

Usage

```
sm_validate(corpus, call = rlang::caller_env())
```

Arguments

corpus	An sm_corpus object.
call	Caller environment for error reporting.

Value

A tibble with table, issue and n_affected, or a 0-row tibble with the same columns if clean. Missing links count distinct unresolved IDs; duplicates count surplus rows.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), validate_sm_corpus()

Examples

```
corpus <- sm_example_corpus()
issues <- sm_validate(corpus)
nrow(issues)
```

validate_sm_corpus *Validate an sm_corpus object*

Description

Check that an sm_corpus has valid structure and consistent IDs.

Usage

```
validate_sm_corpus(x, call = rlang::caller_env())
```

Arguments

x	An sm_corpus object.
call	Caller environment for error reporting.

Value

x invisibly if valid; throws an error otherwise.

See Also

Other corpus: as_sm_corpus(), is_sm_corpus(), sm_bind_corpora(), sm_build_corpus(), sm_concept_vocabularies(), sm_corpus(), sm_corpus_from_tables(), sm_dedupe(), sm_link_references(), sm_materialise(), sm_oa_statuses(), sm_save_corpus(), sm_validate()

Index

∗ **affiliation**
  sm_affiliation_match, 12
  sm_affiliation_methods, 15
  sm_affiliation_signals, 15
  sm_affiliation_summary, 16
  sm_affiliation_unmatched, 17
  sm_attribute_institution, 18
∗ **audit**
  print.sm_audit_summary, 10
  sm_audit_funding, 20
  sm_audit_gender, 21
  sm_audit_geographic, 23
  sm_audit_oa, 25
∗ **causal**
  sm_did, 66
  sm_its, 113
  sm_synth, 196
∗ **chat**
  sm_chat, 32
  sm_chat_render, 34
∗ **clustering**
  sm_cluster_evolution, 38
  sm_cluster_hdbscan, 40
  sm_cluster_kmeans, 41
  sm_cluster_label, 42
  sm_cluster_leiden, 45
∗ **corpus**
  as_sm_corpus, 6
  is_sm_corpus, 8
  sm_bind_corpora, 27
  sm_build_corpus, 28
  sm_concept_vocabularies, 46
  sm_corpus, 47
  sm_corpus_from_tables, 57
  sm_dedupe, 65
  sm_link_references, 118
  sm_materialise, 121
  sm_oa_statuses, 143
  sm_save_corpus, 183
  sm_validate, 202
  validate_sm_corpus, 202
∗ **counting**
  sm_citation_maturity, 35
  sm_citation_window, 36
  sm_count, 59
  sm_metric_summary, 134
∗ **coverage**
  sm_coverage_audit, 61
  sm_coverage_breakdowns, 64
  sm_journal_in_index, 117
  sm_match_types, 120
  sm_reconcile, 177
∗ **datasets**
  sm_affiliation_dict, 11
  sm_example_db, 86
∗ **embedding**
  sm_embed_load, 70
  sm_embed_save, 71
  sm_embed_works, 71
∗ **enrichers**
  sm_enrich_altmetric, 74
  sm_enrich_concepts, 75
  sm_enrich_openalex_impact, 76
  sm_enrich_opencitations, 78
  sm_enrich_orcid, 79
  sm_enrich_retraction, 80
  sm_enrich_ror, 81
  sm_enrich_specter, 82
  sm_enrich_unpaywall, 83
∗ **example**
  sm_example_corpus, 84
  sm_example_files, 86
∗ **export**
  sm_export_csv, 88
  sm_export_cytoscape, 89
  sm_export_figure, 90
  sm_export_gephi, 91
  sm_export_quarto_report, 92

sm_export_rds, 94
sm_export_table, 94
sm_export_vosviewer, 95
sm_export_zip, 96
* **fetchers**
sm_fetch_arxiv, 97
sm_fetch_biorxiv, 98
sm_fetch_crossref, 99
sm_fetch_openalex, 101
sm_fetch_orcid, 103
sm_fetch_overton, 104
sm_fetch_pubmed, 105
sm_fetch_semantic_scholar, 106
* **field-helpers**
sm_field_clinical_trials, 107
sm_field_funder, 108
sm_field_pubmed_mesh, 109
* **filters**
sm_filter_works, 110
sm_query, 157
* **interop**
sm_to_bibliometrix, 198
* **metrics**
sm_metric_collab_index, 123
sm_metric_disruption, 124
sm_metric_fnci, 125
sm_metric_g_index, 128
sm_metric_h_index, 129
sm_metric_m_index, 130
sm_metric_novelty, 131
sm_metric_rcr, 132
sm_self_citation, 189
sm_summary_authors, 193
sm_summary_period, 193
sm_summary_sources, 194
sm_summary_works, 195
* **networks**
sm_network_citation, 136
sm_network_cocitation, 137
sm_network_collab, 138
sm_network_coupling, 139
sm_network_coword, 140
sm_network_semantic, 142
* **plots**
autoplot.sm_corpus, 7
sm_palette_qualitative, 144
sm_plot_bradford, 144
sm_plot_citation_network, 145
sm_plot_collab, 146
sm_plot_equity_dashboard, 148
sm_plot_evolution, 149
sm_plot_heaps, 150
sm_plot_landscape, 150
sm_plot_lotka, 151
sm_plot_production, 152
sm_plot_thematic_map, 153
sm_plot_top, 154
sm_scale_color, 184
sm_theme, 198
* **question**
is_sm_question, 9
sm_corpus_for_question, 56
sm_question, 158
sm_screen_against_question, 185
sm_screen_regex, 187
sm_screen_summary, 188
* **readers**
sm_read_auto, 160
sm_read_bib, 161
sm_read_cochrane, 163
sm_read_dimensions, 164
sm_read_endnote, 165
sm_read_lens, 167
sm_read_openalex_json, 168
sm_read_pubmed_xml, 169
sm_read_ris, 171
sm_read_scopus, 172
sm_read_wos, 174
sm_read_zotero, 176
* **refresh**
sm_lock, 119
sm_refresh, 179
sm_staleness, 192
* **reporting**
sm_corpus_from_tables, 57
sm_figure_manifest, 109
* **reproducibility**
sm_certificate, 29
sm_cite_corpus, 37
sm_diff_corpora, 69
sm_hash_corpus, 111
sm_provenance, 156
sm_register_input, 180
sm_snapshot, 191
* **screening**
sm_export_covidence, 88

sm_export_rayyan, 93
sm_import_rayyan, 112
sm_merge_screening_decisions, 122
sm_screen_prisma, 186
∗ **trajectory**
is_sm_trajectory, 9
sm_author_trajectory, 26
sm_plot_collab_turnover, 147
sm_plot_topic_pivots, 155
sm_plot_trajectory, 155
sm_researcher_profile, 182
sm_trajectory, 201
[.sm_corpus *(sm_corpus)*, 47

as.data.frame.sm_corpus *(sm_corpus)*, 47
as_sm_corpus, 6
as_sm_corpus(), *8*, *28*, *29*, *47*, *51*, *55*, *58*, *66*, *119*, *122*, *143*, *175*, *184*, *200–203*
as_tibble.sm_corpus *(sm_corpus)*, 47
autoplot.sm_corpus, 7
autoplot.sm_corpus(), *144–154*, *185*, *198*
autoplot.sm_coverage *(sm_coverage_audit)*, 61
autoplot.sm_did *(sm_did)*, 66
autoplot.sm_its *(sm_its)*, 113
autoplot.sm_reconciliation *(sm_reconcile)*, 177

dbscan::hdbscan(), *40*
dim.sm_corpus *(sm_corpus)*, 47
dplyr::filter(), *111*

format.sm_corpus *(sm_corpus)*, 47
format.sm_question *(sm_question)*, 158
format.sm_trajectory *(sm_author_trajectory)*, 26

ggplot2::theme(), *198*

igraph::cluster_leiden(), *45*
igraph::igraph, *45*
is_sm_corpus, 8
is_sm_corpus(), *7*, *28*, *29*, *47*, *55*, *58*, *66*, *119*, *122*, *143*, *184*, *202*, *203*
is_sm_question, 9
is_sm_question(), *57*, *159*, *186*, *188*, *189*
is_sm_trajectory, 9
is_sm_trajectory(), *27*, *148*, *155*, *156*, *182*, *201*

length.sm_corpus *(sm_corpus)*, 47

print.sm_audit_funding *(sm_audit_funding)*, 20
print.sm_audit_gender *(sm_audit_gender)*, 21
print.sm_audit_geographic *(sm_audit_geographic)*, 23
print.sm_audit_oa *(sm_audit_oa)*, 25
print.sm_audit_summary, 10
print.sm_audit_summary(), *21*, *23–25*
print.sm_cert_verification *(sm_certificate)*, 29
print.sm_certificate *(sm_certificate)*, 29
print.sm_chat_response *(sm_chat)*, 32
print.sm_corpus *(sm_corpus)*, 47
print.sm_corpus_diff *(sm_diff_corpora)*, 69
print.sm_coverage *(sm_coverage_audit)*, 61
print.sm_did *(sm_did)*, 66
print.sm_its *(sm_its)*, 113
print.sm_question *(sm_question)*, 158
print.sm_reconciliation *(sm_reconcile)*, 177
print.sm_self_citation *(sm_self_citation)*, 189
print.sm_synth *(sm_synth)*, 196
print.sm_trajectory *(sm_author_trajectory)*, 26

readr::read_csv(), *165*, *167*, *173*, *176*
readr::write_csv(), *88*

saveRDS(), *146*, *184*, *191*
scimapR-stability, *65*
shiny::runApp(), *183*
sm_affiliation_dict, 11, *13*, *14*, *18*, *60*, *63*, *87*
sm_affiliation_match, 12
sm_affiliation_match(), *11*, *12*, *15–19*, *51*, *60*, *63*, *64*, *66*, *87*, *115*
sm_affiliation_methods, 15
sm_affiliation_methods(), *14*, *16*, *17*, *19*
sm_affiliation_signals, 15
sm_affiliation_signals(), *14–17*, *19*
sm_affiliation_summary, 16
sm_affiliation_summary(), *14–17*, *19*

sm_affiliation_unmatched, 17
sm_affiliation_unmatched(), *12–17*, *19*
sm_attribute_institution, 18
sm_attribute_institution(), *13–17*, *51*, *60*, *63*, *66*
sm_audit_funding, 20
sm_audit_funding(), *10, 11*, *23–25*
sm_audit_gender, 21
sm_audit_gender(), *11*, *21*, *24, 25*
sm_audit_geographic, 23
sm_audit_geographic(), *11*, *21*, *23*, *25*
sm_audit_oa, 25
sm_audit_oa(), *11*, *21*, *23, 24*, *143*, *194, 195*
sm_audit_summary *(*print.sm_audit_summary*)*, 10
sm_author_trajectory, 26
sm_author_trajectory(), *10*, *148*, *155, 156*, *182*, *201*
sm_bind_corpora, 27
sm_bind_corpora(), *7, 8*, *28, 29*, *47*, *49*, *52–55*, *58*, *66*, *118, 119*, *122*, *143*, *184*, *202, 203*
sm_build_corpus, 28
sm_build_corpus(), *7, 8*, *28*, *47*, *55*, *58*, *66*, *119*, *122*, *143*, *184*, *202, 203*
sm_certificate, 29
sm_certificate(), *38*, *53–55*, *58*, *69, 70*, *73*, *112*, *119, 120*, *156, 157*, *180, 181*, *191*
sm_chat, 32
sm_chat(), *34*
sm_chat_render, 34
sm_chat_render(), *34*
sm_citation_maturity, 35
sm_citation_maturity(), *37*, *50*, *60*, *116*, *135*
sm_citation_window, 36
sm_citation_window(), *35*, *50*, *53*, *60*, *78*, *87*, *101, 102*, *135*
sm_cite_corpus, 37
sm_cite_corpus(), *32*, *69*, *112*, *157*, *181*, *191*
sm_cluster_evolution, 38
sm_cluster_evolution(), *41, 42*, *45, 46*
sm_cluster_hdbscan, 40
sm_cluster_hdbscan(), *39*, *42, 43*, *45, 46*
sm_cluster_kmeans, 41
sm_cluster_kmeans(), *39*, *41*, *45, 46*
sm_cluster_label, 42
sm_cluster_label(), *39*, *41, 42*, *46*, *50*
sm_cluster_leiden, 45
sm_cluster_leiden(), *39*, *41, 42*, *45*
sm_concept_vocabularies, 46
sm_concept_vocabularies(), *7, 8*, *28, 29*, *44*, *49*, *52*, *55*, *57, 58*, *66*, *119*, *122*, *126*, *141*, *143*, *175*, *184*, *199, 200*, *202, 203*
sm_corpus, *38*, *40, 41*, *43*, *45*, 47, *70–72*, *87*, *118*, *123, 124*, *126*, *128–132*, *136–139*, *141, 142*, *160*, *162, 163*, *165–167*, *169–171*, *173, 174*, *176*, *193–195*
sm_corpus(), *7, 8*, *28–30*, *47*, *58*, *66*, *70, 71*, *73*, *77*, *82*, *101*, *107*, *115*, *118, 119*, *122*, *143*, *168*, *184*, *202, 203*
sm_corpus_for_question, 56
sm_corpus_for_question(), *9*, *29*, *31*, *53, 54*, *159*, *186*, *188, 189*
sm_corpus_from_tables, 57
sm_corpus_from_tables(), *7, 8*, *28, 29*, *36*, *47*, *50*, *53–55*, *66*, *76*, *87*, *110*, *118, 119*, *122*, *143*, *180*, *184*, *202, 203*
sm_count, 59
sm_count(), *14*, *19*, *35*, *37*, *115*, *135*
sm_coverage_audit, 61
sm_coverage_audit(), *19*, *60*, *64, 65*, *118*, *120, 121*, *177, 178*
sm_coverage_breakdowns, 64
sm_coverage_breakdowns(), *64*, *118*, *121*, *178*
sm_dedupe, 65
sm_dedupe(), *7, 8*, *28, 29*, *47*, *52*, *55*, *58*, *118, 119*, *122*, *143*, *184*, *202, 203*
sm_did, 66
sm_did(), *115, 116*, *127*, *197*
sm_diff_corpora, 69
sm_diff_corpora(), *32*, *38*, *112*, *157*, *177, 178*, *181*, *191*
sm_embed_load, 70
sm_embed_load(), *40*, *42*, *52*, *54*, *71*, *73*
sm_embed_save, 71
sm_embed_save(), *54*, *70*, *73*
sm_embed_works, 71
sm_embed_works(), *30*, *40–42*, *52*, *54*, *70, 71*, *142*
sm_enrich_altmetric, 74
sm_enrich_altmetric(), *75*, *78–81*, *83, 84*

sm_enrich_concepts, 75
sm_enrich_concepts(), *74*, *78–81*, *83, 84*, *127*
sm_enrich_openalex_impact, 76
sm_enrich_openalex_impact(), *29*, *36, 37*, *50*, *53*, *74, 75*, *78–81*, *83, 84*, *101, 102*, *114*, *134, 135*, *169*
sm_enrich_opencitations, 78
sm_enrich_opencitations(), *74, 75*, *78–81*, *83, 84*
sm_enrich_orcid, 79
sm_enrich_orcid(), *74, 75*, *78*, *80, 81*, *83, 84*
sm_enrich_retraction, 80
sm_enrich_retraction(), *74, 75*, *78, 79*, *81*, *83, 84*
sm_enrich_ror, 81
sm_enrich_ror(), *74, 75*, *78–80*, *83, 84*
sm_enrich_specter, 82
sm_enrich_specter(), *30*, *52*, *54*, *74, 75*, *78–81*, *84*, *107*
sm_enrich_unpaywall, 83
sm_enrich_unpaywall(), *74, 75*, *78–81*, *83*
sm_example_corpus, 84
sm_example_corpus(), *87*, *118*
sm_example_db, 86
sm_example_files, 86
sm_example_files(), *85*
sm_export_covidence, 88
sm_export_covidence(), *93*, *113*, *123*, *187*
sm_export_csv, 88
sm_export_csv(), *89–92*, *94–97*
sm_export_cytoscape, 89
sm_export_cytoscape(), *89–92*, *94–97*
sm_export_figure, 90
sm_export_figure(), *89*, *91, 92*, *94–97*
sm_export_gephi, 91
sm_export_gephi(), *89, 90*, *92*, *94–97*
sm_export_quarto_report, 92
sm_export_quarto_report(), *89–91*, *94–97*
sm_export_rayyan, 93
sm_export_rayyan(), *88*, *113*, *123*, *187*
sm_export_rds, 94
sm_export_rds(), *89–92*, *95–97*
sm_export_table, 94
sm_export_table(), *89–92*, *94*, *96, 97*
sm_export_vosviewer, 95
sm_export_vosviewer(), *89–92*, *94, 95*, *97*
sm_export_zip, 96
sm_export_zip(), *89–92*, *94–96*
sm_fetch_arxiv, 97
sm_fetch_arxiv(), *99, 100*, *102–104*, *106, 107*
sm_fetch_biorxiv, 98
sm_fetch_biorxiv(), *51*, *98*, *100*, *102–104*, *106, 107*, *200*
sm_fetch_crossref, 99
sm_fetch_crossref(), *29*, *31*, *53*, *98, 99*, *102–104*, *106, 107*
sm_fetch_openalex, 101
sm_fetch_openalex(), *29*, *31*, *50, 51*, *53*, *78*, *98–100*, *103, 104*, *106, 107*, *169*, *182*, *200*
sm_fetch_orcid, 103
sm_fetch_orcid(), *98–100*, *102*, *104*, *106, 107*
sm_fetch_overton, 104
sm_fetch_overton(), *98–100*, *102, 103*, *106, 107*
sm_fetch_pubmed, 105
sm_fetch_pubmed(), *29*, *31*, *53*, *98–100*, *102–104*, *107*, *127*, *161*
sm_fetch_semantic_scholar, 106
sm_fetch_semantic_scholar(), *30*, *54*, *98–100*, *102–104*, *106*
sm_field_clinical_trials, 107
sm_field_clinical_trials(), *108, 109*
sm_field_funder, 108
sm_field_funder(), *108, 109*
sm_field_pubmed_mesh, 109
sm_field_pubmed_mesh(), *108*
sm_figure_manifest, 109
sm_figure_manifest(), *58*
sm_filter_works, 110
sm_filter_works(), *143*, *157*
sm_hash_corpus, 111
sm_hash_corpus(), *29*, *31, 32*, *38*, *53, 54*, *69*, *157*, *181*, *191*
sm_import_rayyan, 112
sm_import_rayyan(), *88*, *93*, *123*, *187*
sm_its, 113
sm_its(), *35*, *50, 51*, *67, 68*, *127*, *134, 135*, *196, 197*
sm_journal_in_index, 117
sm_journal_in_index(), *62*, *64, 65*, *121*, *178*
sm_link_references, 118

sm_link_references(), *7, 8*, *28, 29*, *47*, *52*, *55*, *58*, *66*, *122*, *125*, *132*, *136*, *138*, *140*, *143*, *184*, *190*, *202, 203*
sm_load_corpus *(*sm_save_corpus*)*, 183
sm_lock, 119
sm_lock(), *54*, *180*, *192*
sm_match_types, 120
sm_match_types(), *64, 65*, *118*, *178*
sm_materialise, 121
sm_materialise(), *7, 8*, *28, 29*, *47*, *55*, *58*, *66*, *77*, *118, 119*, *143*, *184*, *202, 203*
sm_merge_screening_decisions, 122
sm_merge_screening_decisions(), *88*, *93*, *113*, *187*
sm_metric_collab_index, 123
sm_metric_collab_index(), *125*, *128–133*, *190*, *193–195*
sm_metric_disruption, 124
sm_metric_disruption(), *124*, *128–133*, *190*, *193–195*
sm_metric_fnci, 125
sm_metric_fnci(), *35*, *67*, *114, 115*, *124, 125*, *129–135*, *190*, *193–195*, *197*
sm_metric_g_index, 128
sm_metric_g_index(), *124, 125*, *128*, *130–133*, *190*, *193–195*
sm_metric_h_index, 129
sm_metric_h_index(), *124, 125*, *128, 129*, *131–133*, *190*, *193–195*
sm_metric_m_index, 130
sm_metric_m_index(), *124, 125*, *128–130*, *132, 133*, *190*, *193–195*
sm_metric_novelty, 131
sm_metric_novelty(), *124, 125*, *128–131*, *133*, *190*, *193–195*
sm_metric_rcr, 132
sm_metric_rcr(), *124, 125*, *128–132*, *134*, *190*, *193–195*
sm_metric_summary, 134
sm_metric_summary(), *35*, *37*, *50*, *60*, *114*, *127*
sm_network_citation, 136
sm_network_citation(), *138–141*, *143*
sm_network_cocitation, 137
sm_network_cocitation(), *136*, *139–141*, *143*
sm_network_collab, 138
sm_network_collab(), *136*, *138*, *140, 141*, *143*
sm_network_coupling, 139
sm_network_coupling(), *136*, *138, 139*, *141*, *143*
sm_network_coword, 140
sm_network_coword(), *46, 47*, *136*, *138–140*, *143*
sm_network_semantic, 142
sm_network_semantic(), *45, 46*, *52*, *73*, *136*, *138–141*
sm_oa_statuses, 143
sm_oa_statuses(), *7, 8*, *25*, *28, 29*, *47, 48*, *55*, *58*, *66*, *111*, *119*, *122*, *184*, *194, 195*, *202, 203*
sm_palette_qualitative, 144
sm_palette_qualitative(), *8*, *145–154*, *185*, *198*
sm_plot_bradford, 144
sm_plot_bradford(), *8*, *144*, *146–154*, *185*, *198*
sm_plot_citation_network, 145
sm_plot_citation_network(), *8*, *144, 145*, *147–154*, *185*, *198*
sm_plot_collab, 146
sm_plot_collab(), *8*, *144–146*, *148–154*, *185*, *198*
sm_plot_collab_turnover, 147
sm_plot_collab_turnover(), *10*, *27*, *155, 156*, *182*, *201*
sm_plot_equity_dashboard, 148
sm_plot_equity_dashboard(), *8*, *144–147*, *149–154*, *185*, *198*
sm_plot_evolution, 149
sm_plot_evolution(), *8*, *144–148*, *150–154*, *185*, *198*
sm_plot_heaps, 150
sm_plot_heaps(), *8*, *144–149*, *151–154*, *185*, *198*
sm_plot_landscape, 150
sm_plot_landscape(), *8*, *52*, *73*, *144–150*, *152–154*, *185*, *198*
sm_plot_lotka, 151
sm_plot_lotka(), *8*, *144–154*, *185*, *198*
sm_plot_production, 152
sm_plot_production(), *8*, *144–154*, *185*, *198*
sm_plot_thematic_map, 153
sm_plot_thematic_map(), *8*, *144–152*, *154*,

185, *198*
sm_plot_top, 154
sm_plot_top(), *8*, *144–153*, *185*, *198*
sm_plot_topic_pivots, 155
sm_plot_topic_pivots(), *10*, *27*, *148*, *156*, *182*, *201*
sm_plot_trajectory, 155
sm_plot_trajectory(), *10*, *27*, *148*, *155*, *182*, *201*
sm_provenance, 156
sm_provenance(), *29*, *32*, *38*, *69*, *112*, *181*, *191*
sm_query, 157
sm_query(), *111*
sm_question, 158
sm_question(), *9*, *54*, *56, 57*, *186*, *188, 189*
sm_read_auto, 160
sm_read_auto(), *162*, *164–166*, *168, 169*, *171–173*, *175*, *177*
sm_read_bib, 161
sm_read_bib(), *161*, *164–166*, *168, 169*, *171–173*, *175*, *177*
sm_read_cochrane, 163
sm_read_cochrane(), *161, 162*, *165, 166*, *168, 169*, *171–173*, *175*, *177*
sm_read_dimensions, 164
sm_read_dimensions(), *51*, *161, 162*, *164*, *166*, *168, 169*, *171–173*, *175*, *177*, *200*
sm_read_endnote, 165
sm_read_endnote(), *161, 162*, *164, 165*, *168, 169*, *171–173*, *175*, *177*
sm_read_lens, 167
sm_read_lens(), *161, 162*, *164–166*, *169*, *171–173*, *175*, *177*
sm_read_openalex_json, 168
sm_read_openalex_json(), *50, 51*, *161, 162*, *164–166*, *168*, *171–173*, *175*, *177*, *200*
sm_read_pubmed_xml, 169
sm_read_pubmed_xml(), *127*, *161, 162*, *164–166*, *168, 169*, *172, 173*, *175*, *177*
sm_read_ris, 171
sm_read_ris(), *161, 162*, *164–166*, *168, 169*, *171*, *173*, *175*, *177*
sm_read_scopus, 172
sm_read_scopus(), *161, 162*, *164–166*, *168, 169*, *171, 172*, *175*, *177*, *200*
sm_read_wos, 174
sm_read_wos(), *47*, *51*, *161, 162*, *164–166*, *168, 169*, *171–173*, *177*, *200*
sm_read_zotero, 176
sm_read_zotero(), *161, 162*, *164–166*, *168, 169*, *171–173*, *175*
sm_rebuild_from_cert *(*sm_certificate*)*, 29
sm_rebuild_from_cert(), *53*, *57*, *77*, *83*, *100*, *102*, *105*, *156*, *181*
sm_reconcile, 177
sm_reconcile(), *61*, *64, 65*, *69*, *118*, *120, 121*
sm_refresh, 179
sm_refresh(), *29*, *54*, *119, 120*, *192*
sm_register_input, 180
sm_register_input(), *27*, *29*, *31, 32*, *38*, *54*, *58*, *69*, *111, 112*, *157*, *191*
sm_researcher_profile, 182
sm_researcher_profile(), *10*, *27*, *148*, *155, 156*, *201*
sm_run_app, 183
sm_save_corpus, 183
sm_save_corpus(), *7, 8*, *28, 29*, *47*, *55*, *58*, *66*, *119*, *122*, *143*, *202, 203*
sm_scale_color, 184
sm_scale_color(), *8*, *144–154*, *198*
sm_scale_fill *(*sm_scale_color*)*, 184
sm_screen_against_question, 185
sm_screen_against_question(), *9*, *57*, *159*, *188, 189*
sm_screen_prisma, 186
sm_screen_prisma(), *88*, *93*, *113*, *123*
sm_screen_regex, 187
sm_screen_regex(), *9*, *57*, *159*, *186*, *189*
sm_screen_summary, 188
sm_screen_summary(), *9*, *57*, *159*, *186*, *188*
sm_self_citation, 189
sm_self_citation(), *124, 125*, *128–133*, *193–195*
sm_snapshot, 191
sm_snapshot(), *32*, *38*, *54*, *69*, *112*, *157*, *181*
sm_snapshot_load *(*sm_snapshot*)*, 191
sm_staleness, 192
sm_staleness(), *120*, *180*
sm_summary_authors, 193
sm_summary_authors(), *124, 125*, *128–133*, *190*, *194, 195*

sm_summary_period, 193
sm_summary_period(), *124, 125*, *128–133*, *190*, *193*, *195*
sm_summary_sources, 194
sm_summary_sources(), *124, 125*, *128–133*, *190*, *193–195*
sm_summary_works, 195
sm_summary_works(), *25*, *124, 125*, *128–133*, *135*, *190*, *193–195*
sm_synth, 196
sm_synth(), *68*, *115, 116*, *127*
sm_theme, 198
sm_theme(), *8*, *144–154*, *185*
sm_to_bibliometrix, 198
sm_to_bibliometrix(), *6*, *46, 47*, *51*
sm_trajectory, *10*, *27*, *148*, *155, 156*, *182*, 201
sm_unlock *(*sm_lock*)*, 119
sm_unlock(), *179*
sm_validate, 202
sm_validate(), *7, 8*, *28, 29*, *40*, *42*, *46, 47*, *49*, *55*, *58*, *66*, *119, 120*, *122*, *142, 143*, *184*, *203*
sm_verify_certificate *(*sm_certificate*)*, 29
sm_verify_certificate(), *32*, *54*, *111*, *180, 181*
stats::kmeans(), *41*
str.sm_corpus *(*sm_corpus*)*, 47
summary.sm_corpus *(*sm_corpus*)*, 47
summary.sm_coverage *(*sm_coverage_audit*)*, 61
summary.sm_did *(*sm_did*)*, 66
summary.sm_its *(*sm_its*)*, 113
summary.sm_reconciliation *(*sm_reconcile*)*, 177
summary.sm_self_citation *(*sm_self_citation*)*, 189

tidygraph::tbl_graph, *45*, *136, 137*, *139–141*, *143*
tidytext::stop_words, *141*
tidytext::unnest_tokens(), *141*
toupper(), *200*

validate_sm_corpus, 202
validate_sm_corpus(), *7, 8*, *28, 29*, *47*, *49*, *55*, *58*, *66*, *119*, *122*, *143*, *184*, *202*

xml2, *165*, *170*
xml2::read_xml(), *166*, *170*