Package: CiteSource (via r-universe)

June 17, 2026

Title Data-Driven Search Strategy Development and Evidence Synthesis Reporting

Version 1.0.0

Date 2026-06-17

Description Deduplicates bibliographic citations from multiple sources while preserving customizable metadata, supporting data-driven search strategy development and evidence synthesis reporting. Search results can be analyzed using plots and tables, and imported or exported in 'RIS' and 'CSV' formats. An interactive 'shiny' application is included for exploratory use.

License GPL (>= 3)

URL https://eshackathon.github.io/CiteSource/

BugReports https://github.com/ESHackathon/CiteSource/issues

Imports dplyr, DT, forcats, ggnewscale, ggplot2, glue, gt, igraph, parallelly, purrr, RecordLinkage, rlang, scales, stringr, tibble, tidyr, tidyselect, UpSetR, utf8

Suggests bslib, htmltools, jsonlite, knitr, plotly, progressr, rmarkdown, shiny, shinyalert, shinybusy, shinyjs, shinyWidgets, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Config/roxygen2/version 8.0.0

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

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

Date/Publication 2026-06-17 14:55:59 UTC

RemoteUrl https://github.com/eshackathon/citesource

RemoteRef HEAD

RemoteSha fa9f98753adcae5ec6ebe19327835efd8ea1f70d

Contents

CiteSource-package . . . 3
calculate_detailed_records . . . 4
calculate_initial_records . . . 5
calculate_phase_count . . . 6
calculate_phase_records . . . 7
calculate_record_counts . . . 8
citation_summary_table . . . 9
compare_sources . . . 10
count_unique . . . 11
create_detailed_record_table . . . 12
create_initial_record_table . . . 13
create_precision_sensitivity_table . . . 14
dedup_citations . . . 15
dedup_citations_add_manual . . . 16
dedup_citations_add_sources . . . 16
dedup_log . . . 18
detect_ . . . 19
export_bib . . . 19
export_csv . . . 20
export_dedup_candidates . . . 21
export_ris . . . 22
merge_columns . . . 23
parse_ . . . 24
plot_contributions . . . 24
plot_source_overlap_heatmap . . . 26
plot_source_overlap_upset . . . 27
read_citations . . . 30
record_counts . . . 31
record_level_table . . . 32
reimport_csv . . . 33
reimport_dedup_candidates . . . 34
reimport_ris . . . 35
runShiny . . . 36
synthesisr_read_refs . . . 37
write_bib . . . 38

Index **40**

CiteSource-package *CiteSource: A package to compare sources of citation records*

Description

The CiteSource package supports evidence aggregation by helping with the processing of results of various searches in different sources. It allows to deduplicate results while retaining meta-data on where those results were found and then enables users to compare the contribution of different sources.

Author(s)

Maintainer: Trevor Riley <tnriley@gmail.com> (ORCID)

Authors:

- Trevor Riley <tnriley@gmail.com> (ORCID)
- Kaitlyn Hair <kaitlyn.hair@ed.ac.uk> (ORCID)
- Lukas Wallrich <lukas.wallrich@gmail.com> (ORCID)
- Matthew Grainger <matthewjamesgrainger@gmail.com> (ORCID)
- Sarah Young <sarahy@andrew.cmu.edu> (ORCID)
- Chris Pritchard <chris.pritchard@ntu.ac.uk> (ORCID)
- Neal Haddaway <nealhaddaway@gmail.com> (ORCID)

Other contributors:

- Martin Westgate (Author of included synthesisr fragments) [copyright holder]
- Eliza Grames (Author of included synthesisr fragments) [copyright holder]
- Kaitlyn Hair (Author of included ASySD deduplication code) [copyright holder]
- CAMARADES Group (Authors of ASySD (github.com/camaradesuk/ASySD)) [copyright holder]

See Also

Useful links:

- https://eshackathon.github.io/CiteSource/
- Report bugs at https://github.com/ESHackathon/CiteSource/issues

`calculate_detailed_records` *Calculate Detailed Record Counts*

Description

This function processes a dataset and expands the 'cite_source' column, filters on user-specified labels (if provided), and calculates detailed counts such as the records imported, distinct records, unique records, non-unique records, and several percentage contributions for each citation source/method it also adds a total row summarizing these counts.

Usage

```
calculate_detailed_records(
  unique_citations,
  n_unique,
  labels_to_include = NULL
)
```

Arguments

`unique_citations`	A data frame containing unique citations. The data frame must include the columns `cite_source`, `cite_label`, and `duplicate_id`.
`n_unique`	A data frame containing counts of unique records, typically filtered by specific criteria (e.g., `cite_label == "search"`).
`labels_to_include`	An optional character vector of labels to filter the citations. If provided, only citations matching these labels will be included in the counts. if 'NULL' all labels are included. Default is 'NULL'.

Details

The function first checks if the required columns are present in the input data frames. It then expands the `cite_source` column, filters the data based on the provided labels (if any), and calculates various counts and percentages for each citation source. The function also adds a total row summarizing these counts across all sources.

Value

A data frame with detailed counts for each citation source, including:

- `Records Imported`: Total number of records imported.
- `Distinct Records`: Number of distinct records after deduplication.
- `Unique Records`: Number of unique records specific to a source.
- `Non-unique Records`: Number of records found in other sources.
- `Source Contribution %`: Percentage contribution of each source to the total distinct records.

- `Source Unique Contribution %`: Percentage contribution of each source to the total unique records.
- `Source Unique %`: Percentage of unique records within the distinct records for each source.

Examples

```
# Example usage with a sample dataset
unique_citations <- data.frame(
  cite_source = c("Source1, Source2", "Source2", "Source3"),
  cite_label = c("Label1", "Label2", "Label1"),
  duplicate_id = c(1, 2, 3)
)
n_unique <- data.frame(
  cite_source = c("Source1", "Source2", "Source3"),
  cite_label = c("search", "search", "search"),
  unique = c(10, 20, 30)
)
calculate_detailed_records(unique_citations, n_unique, labels_to_include = "search")
```

---

calculate_initial_records

Calculate Initial Records Unique Citations

Description

This function processes a dataset of unique citations, expands the `cite_source` column, filters based on user-specified labels (if provided), and then calculates the number of records imported and distinct records for each citation source. It also adds a total row summarizing these counts.

Usage

```
calculate_initial_records(unique_citations, labels_to_include = NULL)
```

Arguments

`unique_citations`
: A data frame containing the unique citations. It must contain the columns `cite_source`, `cite_label`, and `duplicate_id`.

`labels_to_include`
: An optional character vector of labels to filter the citations. If provided, only citations matching these labels will be included in the counts. Default is NULL, meaning no filtering will be applied.

Details

The function first checks if the required columns are present in the input data frame. It then expands the `cite_source` column to handle multiple sources listed in a single row and filters the dataset based on the provided labels (if any). The function calculates the number of records imported (total rows) and the number of distinct records (unique `duplicate_id` values) for each citation source. Finally, a total row is added to summarize the counts across all sources.

Value

A data frame containing the counts of `Records Imported` and `Distinct Records` for each citation source. The data frame also includes a "Total" row summing the counts across all sources.

Examples

```
# Example usage with a sample dataset
unique_citations <- data.frame(
  cite_source = c("Source1", "Source2", "Source3"),
  cite_label = c("Label1", "Label2", "Label3"),
  duplicate_id = c(1, 2, 3)
)
calculate_initial_records(unique_citations)
```

`calculate_phase_count` *Calculate phase counts, precision, and recall*

Description

This function calculates counts for different phases and calculates precision and recall for each source based on unique citations and citations dataframe. The phases should be labeled as 'screened' and 'final' (case-insensitive) in the input dataframes. The function will give a warning if these labels are not present in the input dataframes.

Usage

```
calculate_phase_count(unique_citations, citations, db_colname)
```

Arguments

`unique_citations`	A dataframe containing unique citations with phase information. The phase information must be provided in a column named 'cite_label' in the dataframe.
`citations`	A dataframe containing all citations with phase information. The phase information must be provided in a column named 'cite_label' in the dataframe.
`db_colname`	The name of the column representing the source database.

Details

The function will give a warning if 'screened' and 'final' labels are not present in the 'cite_label' column of the input dataframes.

Value

A dataframe containing distinct counts, counts for different phases, precision, and recall for each source, as well as totals.

Examples

```
unique_citations <- data.frame(
db_source = c("Database1", "Database1", "Database2", "Database3", "Database3", "Database3"),
cite_label = c("screened", "final", "screened", "final", "screened", "final"),
duplicate_id = c(102, 102, 103, 103, 104, 104),
other_data = 1:6
)

citations <- data.frame(
db_source = c("Database1", "Database1", "Database1", "Database2", "Database2", "Database3"),
cite_label = c("screened", "final", "screened", "final", "screened", "final"),
other_data = 7:12
)

result <- calculate_phase_count(unique_citations, citations, "db_source")
result
```

`calculate_phase_records` *Calculate Phase Counts with Precision and Recall*

Description

This function calculates the distinct record counts, as well as screened and final record counts, for each citation source across different phases (e.g., "screened", "final"). It also calculates precision and recall metrics for each source.

Usage

```
calculate_phase_records(unique_citations, n_unique, db_colname)
```

Arguments

`unique_citations`	A data frame containing unique citations. It must include the columns `cite_source`, `cite_label`, and `duplicate_id`.
`n_unique`	A data frame containing counts of unique records. Typically filtered by specific criteria, such as `cite_label == "search"`.
`db_colname`	The name of the column representing the citation source in the `unique_citations` data frame.

Details

The function starts by calculating the total distinct records, as well as the total "screened" and "final" records across all sources. It then calculates distinct counts for each source, followed by counts for "screened" and "final" records. Finally, it calculates precision and recall metrics and adds a total row summarizing these counts across all sources.

Value

A data frame with phase counts and calculated precision and recall for each citation source, including:

- `Distinct Records`: The count of distinct records per source.
- `screened`: The count of records in the "screened" phase.
- `final`: The count of records in the "final" phase.
- `Precision`: The precision metric calculated as `final / Distinct Records`.
- `Recall`: The recall metric calculated as `final / Total final records`.

Examples

```
# Example usage with a sample dataset
unique_citations <- data.frame(
  cite_source = c("Source1", "Source2", "Source3"),
  cite_label = c("screened","screened", "final"),
  duplicate_id = c(1, 2, 3)
)
n_unique <- data.frame(
  cite_source = c("Source1", "Source2", "Source3"),
  unique = c(10, 20, 30)
)
calculate_phase_records(unique_citations, n_unique, "cite_source")
```

---

calculate_record_counts

Calculate record counts function Calculate and combine counts of distinct records, imported records, and unique records for each database

---

Description

This function calculates the counts of distinct records, records imported, and unique records for each database source. It combines these counts into one dataframe and calculates several ratios and percentages related to the unique and distinct counts. It also calculates the total for each count type.

Usage

```
calculate_record_counts(unique_citations, citations, n_unique, db_colname)
```

Arguments

`unique_citations`	Dataframe. The dataframe for calculating distinct records count.
`citations`	Dataframe. The dataframe for calculating records imported count.
`n_unique`	Dataframe. The dataframe for calculating unique records count.
`db_colname`	Character. The name of the column containing the database source information.

Value

A dataframe with counts of distinct records, imported records, and unique records for each source, including total counts and several calculated ratios and percentages.

Examples

```
unique_citations <- data.frame(
  db_source = c("Database1", "Database1", "Database2", "Database3", "Database3", "Database3"),
   other_data = 1:6
)

citations <- data.frame(
  db_source = c("Database1", "Database1", "Database1", "Database2", "Database2", "Database3"),
   other_data = 7:12
)

n_unique <- data.frame(
  cite_source = c("Database1", "Database2", "Database2", "Database3", "Database3", "Database3"),
 cite_label = c("search", "final", "search", "search", "search", "final"),
   unique = c(1, 0, 1, 1, 1, 0)
)

result <- calculate_record_counts(unique_citations, citations, n_unique, "db_source")
print(result)
```

---

citation_summary_table

Contribution summary table

Description

Create a summary table to show the contribution of each source and the overall performance of the search. For this to work, labels need to be used that contrast a "search" stage with one or more later stages.

Usage

```
citation_summary_table(
  citations,
  comparison_type = "sources",
  search_label = "search",
  screening_label = "final",
  top_n = NULL
)
```

Arguments

citations	A deduplicated tibble as returned by dedup_citations().
comparison_type	Either "sources" to summarise and assess sources or "strings" to consider strings.
search_label	One or multiple labels that identify initial search results (default: "search") - if multiple labels are provided, they are merged.
screening_label	One or multiple label that identify screened records (default: "final") - if multiple are provided, each is compared to the search stage.
top_n	Number of sources/strings to display, based on the number of total records they contributed at the search stage. Note that calculations and totals will still be based on all citations. Defaults to NULL, then all sources/strings are displayed.

Value

A tibble containing the contribution summary table, which shows the contribution of each source and the overall performance of the search

Examples

```
if (interactive()) {
# Load example data from the package
examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)

# Deduplicate citations and compare sources
unique_citations <- dedup_citations(examplecitations)

unique_citations |>
dplyr::filter(stringr::str_detect(cite_label, "final"))  |>
record_level_table(return = "DT")
citation_summary_table(unique_citations, screening_label = c("screened", "final"))
}
```

---

compare_sources *Compare duplicate citations across sources, labels, and strings*

Description

Compare duplicate citations across sources, labels, and strings

Usage

```
compare_sources(
  unique_data,
  comp_type = c("sources", "strings", "labels"),
  include_references = FALSE
)
```

Arguments

unique_data	from ASySD, merged unique rows with duplicate IDs
comp_type	Specify which fields are to be included. One or more of "sources", "strings" or "labels" - defaults to all.
include_references	Should bibliographic detail be included in return?

Value

dataframe with indicators of where a citation appears, with sources/labels/strings as columns

Examples

```
if (interactive()) {
  # Load example data from the package
 examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
  examplecitations <- readRDS(examplecitations_path)

  # Deduplicate citations and compare sources
  dedup_results <- dedup_citations(examplecitations)
  compare_sources(dedup_results, comp_type = "sources")
}
```

---

count_unique *Count number of unique and non-unique citations from different sources, labels, and strings*

Description

Count number of unique and non-unique citations from different sources, labels, and strings

Usage

```
count_unique(unique_data, include_references = FALSE)
```

Arguments

unique_data	from ASySD, merged unique rows with duplicate IDs
include_references	Should bibliographic detail be included in return?

Value

dataframe with indicators of where a citation appears, with source/label/string as column

Examples

```
# Load example data from the package
examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)

# Deduplicate citations
dedup_results <- dedup_citations(examplecitations)

# Count unique and non-unique citations
count_unique(dedup_results)
```

---

create_detailed_record_table

Create a Detailed Record Table

Description

This function generates a formatted summary table using the `gt` package, which displays detailed counts for each citation source. The table includes columns for the number of records imported, distinct records, unique records, non-unique records, and various contribution percentages. Data from the function calculate_detailed_records is pre-formatted for this table.

Usage

```
create_detailed_record_table(data)
```

Arguments

`data`	A data frame containing the detailed counts for each citation source. The data frame must include the following columns: • `Source`: The name of the citation source. • `Records Imported`: The total number of records imported from the source. • `Distinct Records`: The number of distinct records after deduplication within the source. • `Unique Records`: The number of records unique to that source. • `Non-unique Records`: The number of records found in at least one other source. • `Source Contribution %`: The percentage contribution of each source to the total distinct records. • `Source Unique Contribution %`: The percentage contribution of each source to the total unique records. • `Source Unique %`: The percentage of records from each source that were unique.

Value

A `gt` table object summarizing the detailed record counts for each citation source.

Examples

```
sample_data <- data.frame(
  Source = c("Source1", "Source2", "Total"),
  `Records Imported` = c(100, 150, 250),
  `Distinct Records` = c(90, 140, 230),
  `Unique Records` = c(50, 70, 120),
  `Non-unique Records` = c(40, 70, 110),
  `Source Contribution %` = c("39.1%", "60.9%", "100%"),
  `Source Unique Contribution %` = c("41.7%", "58.3%", "100%"),
  `Source Unique %` = c("55.6%", "50%", "52.2%"),
  check.names = FALSE
)
create_detailed_record_table(sample_data)
```

---

create_initial_record_table

Initial Record Table

Description

This function generates a formatted table displaying the record counts for each citation source, including the number of records imported and the distinct records after deduplication.

Usage

```
create_initial_record_table(data)
```

Arguments

data	A data frame containing the record counts for each citation source. It must include columns Source, Records_Imported, and Distinct_Records.

Details

The function checks if the input data frame is empty and returns an empty gt table if no data is present. Otherwise, it generates a formatted table with labeled columns and adds footnotes explaining the meaning of each column.

Value

A gt table object summarizing the record counts for each citation source.

Examples

```
sample_data <- data.frame(
  Source = c("Source1", "Source2", "Source3"),
  Records_Imported = c(100, 150, 250),
  Distinct_Records = c(90, 140, 230)
)
create_initial_record_table(sample_data)
```

`create_precision_sensitivity_table` *Count and Precision/Sensitivity Table*

Description

This function generates a formatted table that displays the precision and sensitivity (recall) metrics for each citation source, along with distinct records and phase-specific counts such as "screened" and "final".

Usage

```
create_precision_sensitivity_table(data)
```

Arguments

`data`	A data frame containing phase-specific counts and calculated metrics for each citation source. It must include columns such as `Source`, `Distinct_Records`, `final`, `Precision`, `Recall`, and optionally `screened`.

Details

The function first checks whether all values in the `screened` column are zero. If so, the column is removed from the table. The table is then generated using the `gt` package, with labeled columns and footnotes explaining the metrics.

Value

A `gt` table object summarizing the precision and sensitivity metrics for each citation source, with relevant footnotes and labels.

Examples

```
sample_data <- data.frame(
  Source = c("Source1", "Source2", "Total"),
  Distinct_Records = c(100, 150, 250),
  final = c(80, 120, 200),
  Precision = c(80.0, 80.0, 80.0),
  Recall = c(40.0, 60.0, 100.0),
  screened = c(90, 140, 230)
)
create_precision_sensitivity_table(sample_data)
```

dedup_citations *Deduplicate citations*

Description

Deduplicates citation data. Duplicates are assumed to be published in the same journal, so pre-prints vs. their published versions will not be merged.

Usage

```
dedup_citations(raw_citations, manual = FALSE, show_unknown_tags = FALSE)
```

Arguments

raw_citations	Citation dataframe with relevant columns
manual	logical. If TRUE, return the full result list including potential pairs for manual review. Default is FALSE.
show_unknown_tags	When a label, source, or other merged field is missing, show it as "unknown"? Default FALSE.

Value

When `manual = FALSE`: a dataframe of unique citations. When `manual = TRUE`: a list with `$unique` (unique citations), `$manual_dedup` (potential pairs for review), and `$auto_pairs` (pairs that were merged automatically - feed to `dedup_log()` together with confirmed manual pairs to build a full provenance log).

Examples

```
# Load example data from the package
examplecitations_path <- system.file("extdata", "examplecitations.rds",
                                     package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)

# Deduplicate citations
dedup_results <- dedup_citations(examplecitations)

# Return potential pairs for manual review
dedup_results_manual <- dedup_citations(examplecitations, manual = TRUE)
```

dedup_citations_add_manual

Add manually identified duplicate pairs to a deduplicated dataset

Description

Add manually identified duplicate pairs to a deduplicated dataset

Usage

```
dedup_citations_add_manual(unique_citations, additional_pairs)
```

Arguments

unique_citations	Unique citations returned by dedup_citations()
additional_pairs	Dataframe of manually confirmed duplicate pairs (a subset of the $manual_dedup output). If a result column is present, only rows where result == "match" are merged.

Value

Updated unique citations dataframe with manual duplicates merged.

Examples

```
# Load example data from the package
examplecitations_path <- system.file("extdata", "examplecitations.rds",
                                     package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)

# Deduplicate and retrieve manual pairs
dedup_results <- dedup_citations(examplecitations, manual = TRUE)
# (user reviews dedup_results$manual_dedup and sets result == "match" for true dups)
# final <- dedup_citations_add_manual(dedup_results$unique, dedup_results$manual_dedup)
```

dedup_citations_add_sources

Add new citations to a previously deduplicated set and re-deduplicate

Description

Adds further citations (e.g. an additional database search) to a set that was already deduplicated, and deduplicates the new records against both the existing set and each other - without discarding the work already done. Each existing unique record enters as a single row, so prior automatic and manual merge decisions are preserved; the new records are integrated and full provenance (the original record_ids behind every merged record) is carried through.

Usage

```
dedup_citations_add_sources(
  existing_citations,
  new_citations,
  manual = FALSE,
  show_unknown_tags = FALSE
)
```

Arguments

existing_citations	A previously deduplicated set (from dedup_citations(), reimport_csv() or reimport_ris()) - must contain a duplicate_id column.
new_citations	New raw citations to add, as returned by read_citations() (with cite_source / cite_label / cite_string).
manual	logical. If TRUE, return the full result list including $manual_dedup candidate pairs for review (see dedup_citations()). Default FALSE.
show_unknown_tags	When a label, source, or other merged field is missing, show it as "unknown"? Default FALSE.

Details

This is the incremental counterpart to running dedup_citations() on all sources from scratch and, for the same data, produces the same unique set.

Value

When manual = FALSE: a dataframe of unique citations across both sets. When manual = TRUE: a list with $unique, $manual_dedup and $auto_pairs (as in dedup_citations()). In both cases record_ids retains the original record IDs behind every merged record.

See Also

dedup_citations(), dedup_citations_add_manual()

Examples

```
if (interactive()) {
  existing <- dedup_citations(read_citations(old_files, cite_sources = old_srcs))
  new_raw  <- read_citations(new_files, cite_sources = new_srcs)
  combined <- dedup_citations_add_sources(existing, new_raw)
}
```

dedup_log *Build a provenance log of all merged duplicate pairs*

Description

Combines automatically merged pairs and user-confirmed manual pairs into a single tibble with a `method` column (`"auto"` / `"manual"`). Useful for reporting and auditing - e.g. as supplementary material for a systematic review.

Usage

```
dedup_log(dedup_result, confirmed_manual_pairs = NULL)
```

Arguments

`dedup_result`	List returned by `dedup_citations(manual = TRUE)` (must contain `$auto_pairs`; optionally `$manual_dedup`).
`confirmed_manual_pairs`	Optional dataframe of manual pairs the user confirmed as duplicates. Typically a subset of `dedup_result$manual_dedup`. If a `result` column is present, only rows where `result == "match"` are included.

Value

Tibble with columns `method`, `record_id1`, `record_id2`, and the common bibliographic fields (`title1/2`, `author1/2`, `year1/2`, `journal1/2`, `doi1/2`) when available.

Examples

```
examplecitations_path <- system.file("extdata", "examplecitations.rds",
                                     package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)
dedup_results <- dedup_citations(examplecitations, manual = TRUE)
# Log of just the auto-merged pairs
dedup_log(dedup_results)
# Or include user-confirmed manual pairs
# dedup_log(dedup_results, confirmed_manual_pairs = my_confirmed_pairs)
```

detect_ *Detect file formatting information*

Description

Bibliographic data can be stored in a number of different file types, meaning that detecting consistent attributes of those files is necessary if they are to be parsed accurately. These functions attempt to identify some of those key file attributes. Specifically, `detect_parser` determines which `parse_` function to use; `detect_delimiter` and `detect_lookup` identify different attributes of RIS files; and `detect_year` attempts to fill gaps in publication years from other information stored in a `data.frame`.

Usage

```
detect_parser(x)

detect_delimiter(x)

detect_lookup(tags)

detect_year(df)
```

Arguments

`x`	A character vector containing bibliographic data
`tags`	A character vector containing RIS tags.
`df`	a data.frame containing bibliographic data

Value

`detect_parser` and `detect_delimiter` return a length-1 character; `detect_year` returns a character vector listing estimated publication years; and `detect_lookup` returns a `data.frame`.

export_bib *Export deduplicated citations to .bib file*

Description

This function saves deduplicated citations as a BibTex file with sources, labels and strings included in the `note` field (if they were initially provided for any of the citations). Therefore, beware that **any** `note` **field that might be included in** `citations` **will be overwritten**. Also note that *existing files are overwritten without warning.*

Usage

```
export_bib(citations, filename, include = c("sources", "labels", "strings"))
```

Arguments

citations	Dataframe with unique citations, resulting from dedup_citations()
filename	Name (and path) of file, should end in .ris
include	Character. One or more of sources, labels or strings

Value

No return value, called for side effects. Saves deduplicated citations as a 'BibTeX' file to the specified location.

Examples

```
if (interactive()) {
  # Load example data from the package
 examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
  examplecitations <- readRDS(examplecitations_path)
  dedup_results <- dedup_citations(examplecitations, merge_citations = TRUE)
  export_bib(dedup_results$unique, tempfile(fileext = ".bib"), include = "sources")
}
```

---

export_csv *Export deduplicated citations with source data as CSV file*

Description

This function saves deduplicated citations as a CSV file for further analysis and/or reporting. Metadata can be separated into one column per source, label or string, which facilitates analysis. Note that *existing files are overwritten without warning.*

Usage

```
export_csv(
  unique_citations,
  filename,
  fields = "full",
  separate = NULL,
  trim_abstracts = 32000,
  manual_dedup_complete = FALSE
)
```

Arguments

unique_citations	Dataframe with unique citations, resulting from dedup_citations()
filename	Name (and path) of file, should end in .csv

`fields` Controls which columns are included. Use `"full"` (default) to export all columns (required for reimport into CiteSource via `reimport_csv()`); `"standard"` to export core bibliographic fields plus `cite_source`, `cite_label`, and `cite_string` (suitable for import into RELApp or other screening tools); or a character vector of column names for a custom selection. Note that exports other than `"full"` cannot be reimported into CiteSource.

`separate` Character vector indicating which (if any) of cite_source, cite_string and cite_label should be split into separate columns to facilitate further analysis.

`trim_abstracts` Some databases may return full-text that is misidentified as an abstract. This inflates file size and may lead to issues with Excel, which cannot deal with more than 32,000 characters per field. Therefore, the default is to trim very long abstracts to 32,000 characters. Set a lower number to reduce file size, or NULL to retain abstracts as they are.

`manual_dedup_complete` Logical. Records, in a `manual_dedup_complete` column, whether manual deduplication has been completed for this set (default `FALSE`). Set `TRUE` after confirming manual pairs with `dedup_citations_add_manual()`. This flag is read back by `reimport_csv()` and lets later steps know whether candidate pairs still need review. Only written when `fields = "full"`.

Value

No return value, called for side effects. Saves the deduplicated citations as a 'CSV' file to the specified location.

Examples

```
if (interactive()) {
  # Load example data from the package
 examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
  examplecitations <- readRDS(examplecitations_path)
  dedup_results <- dedup_citations(examplecitations, merge_citations = TRUE)
  export_csv(dedup_results, tempfile(fileext = ".csv"), separate = "cite_source")
  # Standard export for RELApp / screening tools (not reimportable into CiteSource):
  export_csv(dedup_results, tempfile(fileext = ".csv"), fields = "standard")
}
```

---

export_dedup_candidates

Export manual-review candidate pairs to a CSV file

Description

Saves the candidate duplicate pairs returned as the `$manual_dedup` element of `dedup_citations(manual = TRUE)` so that manual review can be completed later. Combine with `export_csv()` to defer manual deduplication: export the automatically deduplicated unique citations *and* these candidate pairs now, then re-import both later with `reimport_csv()` and `reimport_dedup_candidates()` to finish the review. Note that *existing files are overwritten without warning.*

Usage

```
export_dedup_candidates(manual_dedup, filename)
```

Arguments

manual_dedup	Data frame of candidate pairs, i.e. the $manual_dedup element of dedup_citations(manual = TRUE).
filename	Name (and path) of file, should end in .csv

Value

No return value, called for side effects. Saves the candidate pairs as a 'CSV' file to the specified location.

See Also

reimport_dedup_candidates(), dedup_citations_add_manual()

Examples

```
if (interactive()) {
 examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
  examplecitations <- readRDS(examplecitations_path)
  dedup_results <- dedup_citations(examplecitations, manual = TRUE)
  export_dedup_candidates(dedup_results$manual_dedup, tempfile(fileext = ".csv"))
}
```

---

export_ris *Export data frame to RIS file*

Description

This function saves a data frame as a RIS file with specified columns mapped to RIS fields. Note that *existing files are overwritten without warning.*

Usage

```
export_ris(
  citations,
  filename,
  source_field = "DB",
  label_field = "C7",
  string_field = "C8"
)
```

Arguments

citations	Dataframe to be exported to RIS file
filename	Name (and path) of file, should end in .ris
source_field	Field in citations representing the source. Default is "DB".
label_field	Field in citations representing the label. Default is "C7".
string_field	Field in citations representing additional string information. Default is "C8".

Value

No return value, called for side effects. Saves the citations as a 'RIS' file to the specified location.

Examples

```
if (interactive()) {
  # Load example data from the package
 examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
  examplecitations <- readRDS(examplecitations_path)
  dedup_results <- dedup_citations(examplecitations, merge_citations = TRUE)
  export_ris(dedup_results$unique, tempfile(fileext = ".ris"))
}
```

---

merge_columns *Bind two or more data frames with different columns*

Description

Takes two or more data.frames with different column names or different column orders and binds them to a single data.frame.

Usage

```
merge_columns(x, y)
```

Arguments

x	Either a data.frame or a list of data.frames.
y	A data.frame, optional if x is a list.

Value

Returns a single data.frame with all the input data frames merged.

parse_ *Parse bibliographic text in a variety of formats*

Description

Text in standard formats - such as imported via readLines - can be parsed using a variety of standard formats. Use detect_parser to determine which is the most appropriate parser for your situation.

Usage

```
parse_pubmed(x)

parse_ris(x, tag_naming = "best_guess")

parse_bibtex(x)

parse_csv(x)

parse_tsv(x)
```

Arguments

x	A character vector containing bibliographic information in ris format.
tag_naming	What format are ris tags in? Defaults to "best_guess" See synthesisr_read_refs for a list of accepted arguments.

Value

Returns an object of class bibliography (ris, bib, or pubmed formats) or data.frame (csv or tsv).

plot_contributions *Create a bar chart that compares source contributions over stages*

Description

Create a faceted plot that shows unique contributions and duplicated records across two metadata dimensions. Most typical use-case might be to show the contributions of each source across different screening stages.

Usage

```
plot_contributions(
  data,
  facets = cite_source,
  bars = cite_label,
  color = type,
  center = FALSE,
  bar_order = "keep",
  facet_order = "keep",
  color_order = "keep",
  totals_in_legend = FALSE
)
```

Arguments

data	A tibble with one hit per row, with variables indicating meta-data of interest.
facets	Variable in data used for facets (i.e. sub-plots). Defaults to source (i.e. cite_source). Specify NULL to refrain from faceting.
bars	Variable in data used for bars. Defaults to label (i.e. cite_label)
color	Color used to fill bars. Default to unique
center	Logical. Should one color be above and one below the axis?
bar_order	Character. Order of bars within each facet, any levels not specified will follow at the end. If "keep", then this is based on factor levels (or the first value) in the input data.
facet_order	Character. Order of facets. Any levels not specified will follow at the end.
color_order	Character. Order of values on the color scale.
totals_in_legend	Logical. Should totals be shown in legend (e.g. as Unique (N = 1234))

Value

A ggplot2 object showing source contributions as a faceted bar chart. The object can be further customized using ggplot2 functions or saved with ggsave.

Examples

```
data <- data.frame(
  article_id = 1:100,
  cite_source = sample(c("DB 1", "DB 2", "DB 3"), 100, replace = TRUE),
  cite_label = sample(c("2020", "2021", "2022"), 100, replace = TRUE),
  type = c("unique", "duplicated")[rbinom(100, 1, .7) + 1]
)

plot_contributions(data,
  center = TRUE, bar_order = c("2022", "2021", "2020"),
  color_order = c("unique", "duplicated")
)
```

`plot_source_overlap_heatmap` *Create a heatmap matrix showing the overlap between sources*

Description

Show overlap between different record sources, either by showing the number or the percentages of shared records between any pair of sources.

Usage

```
plot_source_overlap_heatmap(
  data,
  cells = "source",
  facets = NULL,
  plot_type = c("counts", "percentages"),
  sort_sources = TRUE,
  interactive = FALSE,
  show_labels = "auto",
  log_scale = FALSE
)
```

Arguments

`data`	A tibble with one record per row, an id column and then one column per source indicating whether the record was found in that source (usually obtained from `compare_sources()`)
`cells`	Variable to display in the cells. Should be 'source', 'label' or 'string'
`facets`	Variable in data used for facets (i.e. sub-plots). Should be NULL, 'source', 'label' or 'string'
`plot_type`	Either `counts` (number of shared records) or `percentages` (share of overlapping records).
`sort_sources`	Should sources be shown based on the number of records they contained? If FALSE, order of data is retained.
`interactive`	Should returned plot be interactive and enable user to export records underlying each field?
`show_labels`	Whether to show text labels in cells. `"auto"` (default) shows labels when there are 10 or fewer sources; `TRUE` always shows them; `FALSE` hides them.
`log_scale`	Should the fill colour scale be log-transformed? Useful when counts vary greatly across cells. Ignored when `plot_type = "percentages"`.

Value

The requested plot as a either a `ggplot2` object (when interactive = FALSE), which can then be further formatted or saved using `ggplot2::ggsave()`, or a `plotly` object when `interactive = TRUE`

Examples

```
data <- data.frame(
  article_id = 1:500,
  source__source1 = rbinom(500, 1, .5) == 1,
  source__source2 = rbinom(500, 1, .2) == 1,
  source__source3 = rbinom(500, 1, .1) == 1,
  source__source4 = rbinom(500, 1, .6) == 1,
  source__source5 = rbinom(500, 1, .7) == 1
)

plot_source_overlap_heatmap(data)
plot_source_overlap_heatmap(data, plot_type = "percentages")
```

`plot_source_overlap_upset` *Create an UpSetR upset plot showing the overlap between sources*

Description

Show records found in specific sets of sources to identify the unique contribution of each source and of any subsets

Usage

```
plot_source_overlap_upset(
  data,
  groups = "source",
  nsets = NULL,
  sets.x.label = "Number of records",
  mainbar.y.label = "Overlapping record count",
  order.by = c("freq", "degree"),
  ...
)
```

Arguments

`data`	A tibble with one record per row, an id column and then one column per source indicating whether the record was found in that source.
`groups`	Variable to use as groups. Should be 'source', 'label' or 'string' - defaults to source.
`nsets`	Number of sets to look at
`sets.x.label`	The x-axis label of the set size bar plot
`mainbar.y.label`	The y-axis label of the intersection size bar plot

`order.by` How the intersections in the matrix should be ordered by. Options include frequency (entered as "freq"), degree, or both in any order.

`...` Arguments passed on to `UpSetR::upset`

`nintersects` Number of intersections to plot. If set to NA, all intersections will be plotted.

`sets` Specific sets to look at (Include as combinations. Ex: c("Name1", "Name2"))

`keep.order` Keep sets in the order entered using the sets parameter. The default is FALSE, which orders the sets by their sizes.

`set.metadata` Metadata that offers insight to an attribute of the sets. Input should be a data frame where the first column is set names, and the remaining columns are attributes of those sets. To learn how to use this parameter it is highly suggested to view the set metadata vignette. The link can be found on the package's GitHub page.

`intersections` Specific intersections to include in plot entered as a list of lists. Ex: list(list("Set name1", "Set name2"), list("Set name1", "Set name3")). If data is entered into this parameter the only data shown on the UpSet plot will be the specific intersections listed.

`matrix.color` Color of the intersection points

`main.bar.color` Color of the main bar plot

`mainbar.y.max` The maximum y value of the intersection size bar plot scale. May be useful when aligning multiple UpSet plots horizontally.

`sets.bar.color` Color of set size bar plot

`plot.title` Title of UpSetR plot

`point.size` Size of points in matrix plot

`line.size` Width of lines in matrix plot

`mb.ratio` Ratio between matrix plot and main bar plot (Keep in terms of hundredths)

`expression` Expression to subset attributes of intersection or element query data. Enter as string (Ex: "ColName > 3")

`att.pos` Position of attribute plot. If NULL or "bottom" the plot will be at below UpSet plot. If "top" it will be above UpSet plot

`att.color` Color of attribute histogram bins or scatterplot points for unqueried data represented by main bars. Default set to color of main bars.

`decreasing` How the variables in order.by should be ordered. "freq" is decreasing (greatest to least) and "degree" is increasing (least to greatest)

`show.numbers` Show numbers of intersection sizes above bars

`number.angles` The angle of the numbers atop the intersection size bars

`number.colors` The colors of the numbers atop the intersection size bars

`group.by` How the data should be grouped ("degree" or "sets")

`cutoff` The number of intersections from each set (to cut off at) when aggregating by sets

`queries` Unified query of intersections, elements, and custom row functions. Entered as a list that contains a list of queries. query is the type of query being conducted. params are the parameters of the query (if any). color is the color of the points on the plot that will represent the query. If no

color is selected one will be provided automatically. active takes TRUE or FALSE, and if TRUE, it will overlay the bars present with the results from the query. If FALSE a tick mark will indicate the intersection size. See examples section on how to do this.

`query.legend` Position query legend on top or bottom of UpSet plot

`shade.color` Color of row shading in matrix

`shade.alpha` Transparency of shading in matrix

`matrix.dot.alpha` Transparency of the empty intersections points in the matrix

`empty.intersections` Additionally display empty sets up to nintersects

`color.pal` Color palette for attribute plots

`boxplot.summary` Boxplots representing the distribution of a selected attribute for each intersection. Select attributes by entering a character vector of attribute names (e.g. c("Name1", "Name2")). The maximum number of attributes that can be entered is 2.

`attribute.plots` Create custom ggplot using intersection data represented in the main bar plot. Prior to adding custom plots, the UpSet plot is set up in a 100 by 100 grid. The attribute.plots parameter takes a list that contains the number of rows that should be allocated for the custom plot, and a list of plots with specified positions. nrows is the number of rows the custom plots should take up. There is already 100 allocated for the custom plot. plots takes a list that contains a function that returns a custom ggplot and the x and y aesthetics for the function. ncols is the number of columns that your ggplots should take up. See examples for how to add custom ggplots.

`scale.intersections` The scale to be used for the intersection sizes. Options: "identity", "log10", "log2"

`scale.sets` The scale to be used for the set sizes. Options: "identity", "log10", "log2"

`text.scale` Numeric, value to scale the text sizes, applies to all axis labels, tick labels, and numbers above bar plot. Can be a universal scale, or a vector containing individual scales in the following format: c(intersection size title, intersection size tick labels, set size title, set size tick labels, set names, numbers above bars)

`set_size.angles` Numeric, angle to rotate the set size plot x-axis text

`set_size.show` Logical, display the set sizes on the set size bar chart

`set_size.numbers_size` If set_size.show is TRUE, adjust the size of the numbers

`set_size.scale_max` Increase the maximum of set size scale

Value

No return value, called for side effects. Renders an UpSet plot showing record overlap between sources to the current graphics device.

References

Conway, J. R., Lex, A., & Gehlenborg, N. (2017). UpSetR: an R package for the visualization of intersecting sets and their properties. Bioinformatics.

Examples

```
data <- data.frame(
  article_id = 1:500,
  source__source1 = rbinom(500, 1, .5) == 1,
  source__source2 = rbinom(500, 1, .2) == 1,
  source__source3 = rbinom(500, 1, .1) == 1,
  source__source4 = rbinom(500, 1, .6) == 1,
  source__source5 = rbinom(500, 1, .7) == 1
)

plot_source_overlap_upset(data)

# To start with the records shared among the greatest number of sources, use

plot_source_overlap_upset(data, decreasing = c(TRUE, TRUE))
```

---

read_citations *Import citations from file*

Description

This function imports RIS and Bibtex files with citations and merges them into one long tibble with one record per line.

Usage

```
read_citations(
  files = NULL,
  cite_sources = NULL,
  cite_strings = NULL,
  cite_labels = NULL,
  metadata = NULL,
  verbose = TRUE,
  tag_naming = "best_guess",
  only_key_fields = TRUE
)
```

Arguments

files	One or multiple RIS or Bibtex files with citations. Should be .bib or .ris files
cite_sources	The origin of the citation files (e.g. "Scopus", "WOS", "Medline") - vector with one value per file, defaults to file names.
cite_strings	Optional. The search string used (or another grouping to analyse) - vector with one value per file
cite_labels	Optional. An additional label per file, for instance the stage of search - vector with one value per file

metadata	A tibble with file names and metadata for each file. Can be specified as an *alternative* to files, cite_sources, cite_strings and cite_labels.
verbose	Should number of reference and allocation of labels be reported?
tag_naming	Either a length-1 character stating how should ris tags be replaced (see details for a list of options), or an object inheriting from class `data.frame` containing user-defined replacement tags.
only_key_fields	Should only key fields (e.g., those used by CiteCourse) be imported? If FALSE, all RIS data is retained. Can also be a character vector of field names to retain (after they have been renamed by the import function) in addition to the essential ones.

Value

A tibble with one row per citation

Examples

```
if (interactive()) {
  # Import only key fields from the RIS files
  read_citations(c("res.ris", "res.bib"),
    cite_sources = c("CINAHL", "MEDLINE"),
    cite_strings = c("Search1", "Search2"),
    cite_labels = c("raw", "screened"),
    only_key_fields = TRUE
  )

  # or equivalently
  metadata_tbl_key_fields <- tibble::tribble(
   ~files,     ~cite_sources, ~cite_strings, ~cite_labels, ~only_key_fields,
  "res.ris",  "CINAHL",      "Search1",     "raw",        TRUE,
  "res.bib",  "MEDLINE",     "Search2",     "screened",   TRUE
  )

  read_citations(metadata = metadata_tbl_key_fields)
}
```

---

record_counts *Record counts function Calculate and combine counts of distinct records and imported records for each database*

Description

This function calculates the counts of distinct records and records imported for each database source. It combines these counts into one dataframe and calculates the total for each count type.

Usage

```
record_counts(unique_citations, citations, db_colname)
```

Arguments

unique_citations	Dataframe. The dataframe for calculating distinct records count.
citations	Dataframe. The dataframe for calculating records imported count.
db_colname	Character. The name of the column containing the database source information.

Value

A dataframe with counts of distinct records and imported records for each source, including total counts.

Examples

```
# Create synthetic data for example
unique_citations <- data.frame(
  title = paste("Article", 1:10),
  db_source = sample(c("Database 1", "Database 2", "Database 3"), 10, replace = TRUE),
  stringsAsFactors = FALSE
)

citations <- data.frame(
  title = paste("Article", 1:20),
  db_source = sample(c("Database 1", "Database 2", "Database 3"), 20, replace = TRUE),
  stringsAsFactors = FALSE
)

# Use the synthetic data with the function
result <- record_counts(unique_citations, citations, "db_source")
result
```

---

record_level_table *Record-level table*

Description

Creates a per-record table that shows which sources (and/or labels/strings) each item was found in.

Usage

```
record_level_table(
  citations,
  include = "sources",
  include_empty = TRUE,
  return = c("tibble", "DT"),
  indicator_presence = NULL,
  indicator_absence = NULL
)
```

Arguments

citations	A deduplicated tibble as returned by `dedup_citations()`.
include	Which metadata should be included in the table? Defaults to 'sources', can be replaced or expanded with 'labels' and/or 'strings'
include_empty	Should records with empty metadata (e.g., no information on 'sources') be included in the table? Defaults to FALSE.
return	Either a `tibble` that can be exported, e.g. as a csv, or a DataTable (`DT`) that allows for interactive exploration. Note that the DataTable allows users to download a .csv file; in that file, presence and absence is always indicated as TRUE and FALSE to prevent issues with character encodings.
indicator_presence	How should it be indicated that a value is present in a source/label/string? Defaults to TRUE in tibbles and a tickmark in DT tables
indicator_absence	How should it be indicated that a value is *not* present in a source/label/string? Defaults to FALSE in tibbles and a cross in DT tables

Value

A tibble or DataTable containing the per-record table that shows which sources (and/or labels/strings) each item was found in.

Examples

```
# Load example data from the package
examplecitations_path <- system.file("extdata", "examplecitations.rds", package = "CiteSource")
examplecitations <- readRDS(examplecitations_path)

# Deduplicate citations and compare sources
unique_citations <- dedup_citations(examplecitations)

unique_citations |>
dplyr::filter(stringr::str_detect(cite_label, "final"))  |>
record_level_table(return = "DT")
```

---

reimport_csv *Reimport a CSV-file exported from CiteSource*

Description

This function reimports a csv file that was tagged and deduplicated by CiteSource. It allows to continue with further analyses without repeating that step, and also allows users to make any manual corrections to tagging or deduplication. Note that this function only works on CSV files that were written with `export_csv(..., separate = NULL)`

Usage

```
reimport_csv(filename)
```

Arguments

filename	Name (and path) of CSV file to be reimported, should end in .csv

Value

A data frame containing the imported citation data if all required columns are present.

Examples

```
if (interactive()) {
  citations <- reimport_csv("path/to/citations.csv")
}
```

---

reimport_dedup_candidates

Reimport manual-review candidate pairs exported from CiteSource

---

Description

Reads a CSV of candidate duplicate pairs previously written by `export_dedup_candidates()` (i.e. the `$manual_dedup` element of `dedup_citations(manual = TRUE)`). This supports a deferred workflow: run automatic deduplication now, export both the unique citations and the candidate pairs, and complete the manual review later after re-importing.

Usage

```
reimport_dedup_candidates(filename)
```

Arguments

filename	Name (and path) of the candidate-pairs CSV, should end in .csv

Details

After review, set the `result` column to `"match"` for confirmed duplicates and pass the result, together with the reimported unique citations, to `dedup_citations_add_manual()`.

Value

A data frame of candidate pairs with `duplicate_id.x` / `duplicate_id.y` read as character (matching the unique citations from `reimport_csv()`), ready for review and `dedup_citations_add_manual()`.

See Also

export_dedup_candidates(), dedup_citations_add_manual()

Examples

```
if (interactive()) {
  candidates <- reimport_dedup_candidates("path/to/candidates.csv")
  # mark confirmed duplicates, then merge into the reimported unique set
  candidates$result <- ifelse(candidates$result == "match", "match", "no_match")
  final <- dedup_citations_add_manual(reimport_csv("unique.csv"), candidates)
}
```

---

reimport_ris *Reimport a RIS-file exported from CiteSource*

Description

This function reimports a RIS file that was tagged and deduplicated by CiteSource. It allows to continue with further analyses without repeating that step, and also allows users to make any manual corrections to tagging or deduplication. The function can also be used to replace the import step (for instance if tags are to be added to individual citations rather than entire files) - in this case, just call `dedup_citations()` after the import.

Usage

```
reimport_ris(
  filename = "citations.ris",
  source_field = "DB",
  label_field = "C7",
  string_field = "C8",
  duplicate_id_field = "C1",
  record_id_field = "C2",
  tag_naming = "ris_synthesisr",
  verbose = TRUE
)
```

Arguments

`filename`	Name (and path) of RIS file to be reimported, should end in .ris
`source_field`	Character. Which RIS field should cite_sources be read from? NULL to set to missing
`label_field`	Character. Which RIS field should cite_labels be read from? NULL to set to missing
`string_field`	Character. Which RIS field should cite_strings be read from? NULL to set to missing

duplicate_id_field
: Character. Which RIS field should duplicate IDs be read from? NULL to recreate based on row number (note that neither duplicate nor record IDs directly affect CiteSource analyses - they can only allow you to connect processed data with raw data)

record_id_field
: Character. Which RIS field should record IDs be read from? NULL to recreate based on row number

tag_naming
: Synthesisr option specifying how RIS tags should be replaced with names. This should not be changed when using this function to reimport a file exported from CiteSource. If you import your own RIS, check `names(CiteSource:::synthesisr_code_lookup)` and select any of the options that start with `ris_`

verbose
: Should confirmation message be displayed?

Details

Note that this functions defaults' are based on those in `export_ris()` so that these functions can easily be combined.

Value

A data frame containing the reimported citation data, with 'CiteSource' metadata columns (cite_source, cite_label, cite_string, duplicate_id, record_ids) restored from the 'RIS' fields.

Examples

```
if (interactive()) {
  dedup_results <- dedup_citations(citations, merge_citations = TRUE)
  tmp <- tempfile(fileext = ".ris")
  export_ris(dedup_results$unique, tmp)
  unique_citations2 <- reimport_ris(tmp)
}
```

---

runShiny *A wrapper function to run Shiny Apps from* CiteSource.

Description

Running this function will launch the CiteSource shiny app

Usage

```
runShiny(app = "CiteSource", offer_install = interactive())
```

Arguments

app
: Defaults to CiteSource - possibly other apps will be included in the future

offer_install
: Should user be prompted to install required packages if they are missing?

Value

CiteSource shiny app

Examples

```
if (interactive()) {
  # To run the CiteSource Shiny app:
  runShiny()
}
```

`synthesisr_read_refs` *Import bibliographic search results*

Description

Imports common bibliographic reference formats (i.e. .bib, .ris, or .txt).

Usage

```
synthesisr_read_refs(
  filename,
  tag_naming = "best_guess",
  return_df = TRUE,
  verbose = FALSE,
  select_fields = NULL
)

read_ref(
  filename,
  tag_naming = "best_guess",
  return_df = TRUE,
  verbose = FALSE,
  select_fields = NULL
)
```

Arguments

`filename`	A path to a filename or vector of filenames containing search results to import.
`tag_naming`	Either a length-1 character stating how should ris tags be replaced (see details for a list of options), or an object inheriting from class `data.frame` containing user-defined replacement tags.
`return_df`	If TRUE (default), returns a data.frame; if FALSE, returns a list.
`verbose`	If TRUE, prints status updates (defaults to FALSE).
`select_fields`	Character vector of fields to be retained. If NULL, all fields from the RIS file are returned

Details

The default for argument `tag_naming` is `"best_guess"`, which estimates what database has been used for ris tag replacement, then fills any gaps with generic tags. Any tags missing from the database (i.e. `code_lookup`) are passed unchanged. Other options are to use tags from Web of Science (`"wos"`), Scopus (`"scopus"`), Ovid (`"ovid"`) or Academic Search Premier (`"asp"`). If a `data.frame` is given, then it must contain two columns: `"code"` listing the original tags in the source document, and `"field"` listing the replacement column/tag names. The `data.frame` may optionally include a third column named `"order"`, which specifies the order of columns in the resulting `data.frame`; otherwise this will be taken as the row order. Finally, passing `"none"` to `replace_tags` suppresses tag replacement.

Value

Returns a data.frame or list of assembled search results.

Functions

- `read_ref()`: Import a single file

---

write_bib *Export data to a bibliographic format*

Description

This function exports data.frames containing bibliographic information to either a .ris or .bib file.

Usage

```
write_bib(x)

write_ris(x, tag_naming = "synthesisr")

write_refs(x, format = "ris", tag_naming = "synthesisr", file = FALSE)
```

Arguments

`x`	Either a data.frame containing bibliographic information or an object of class bibliography.
`tag_naming`	what naming convention should be used to write RIS files? See details for options.
`format`	What format should the data be exported as? Options are ris or bib.
`file`	Either logical indicating whether a file should be written (defaulting to FALSE), or a character giving the name of the file to be written.

Value

Returns a character vector containing bibliographic information in the specified format if `file` is FALSE, or saves output to a file if TRUE.

Functions

- `write_bib()`: Format a bib file for export
- `write_ris()`: Format a ris file for export

Index

calculate_detailed_records, 4
calculate_initial_records, 5
calculate_phase_count, 6
calculate_phase_records, 7
calculate_record_counts, 8
citation_summary_table, 9
CiteSource *(*CiteSource-package*)*, 3
CiteSource-package, 3
compare_sources, 10
count_unique, 11
create_detailed_record_table, 12
create_initial_record_table, 13
create_precision_sensitivity_table, 14

dedup_citations, 15
dedup_citations(), *16, 17*
dedup_citations_add_manual, 16
dedup_citations_add_manual(), *17*, *21, 22*, *34, 35*
dedup_citations_add_sources, 16
dedup_log, 18
dedup_log(), *15*
detect_, 19
detect_delimiter *(*detect_*)*, 19
detect_lookup *(*detect_*)*, 19
detect_parser, *24*
detect_parser *(*detect_*)*, 19
detect_year *(*detect_*)*, 19

export_bib, 19
export_csv, 20
export_csv(), *21*
export_dedup_candidates, 21
export_dedup_candidates(), *34, 35*
export_ris, 22

ggplot2::ggsave(), *26*
ggsave, *25*

merge_columns, 23

parse_, *19*, 24
parse_bibtex *(*parse_*)*, 24
parse_csv *(*parse_*)*, 24
parse_pubmed *(*parse_*)*, 24
parse_ris *(*parse_*)*, 24
parse_tsv *(*parse_*)*, 24
plot_contributions, 24
plot_source_overlap_heatmap, 26
plot_source_overlap_upset, 27

read_citations, 30
read_citations(), *17*
read_ref *(*synthesisr_read_refs*)*, 37
readLines, *24*
record_counts, 31
record_level_table, 32
reimport_csv, 33
reimport_csv(), *17*, *21*, *34*
reimport_dedup_candidates, 34
reimport_dedup_candidates(), *21, 22*
reimport_ris, 35
reimport_ris(), *17*
run_shiny *(*runShiny*)*, 36
runShiny, 36

synthesisr_read_refs, *24*, 37

UpSetR::upset, *28*

write_bib, 38
write_refs *(*write_bib*)*, 38
write_ris *(*write_bib*)*, 38