Package 'searchAnalyzeR'

August 28, 2025

Type Package

Title Advanced Analytics and Testing Framework for Systematic Review Search Strategies

Version 0.1.0

Date 2025-08-12

Description Provides comprehensive analytics, reporting, and testing capabilities for systematic review search strategies. The package focuses on validating search performance, generating standardized 'PRISMA'-compliant reports, and ensuring reproducibility in evidence synthesis. Features include precision-recall analysis, cross-database performance comparison, benchmark validation against gold standards, sensitivity analysis, temporal coverage assessment, automated report generation, and statistical comparison of search strategies. Supports multiple export formats including 'CSV', 'Excel', 'RIS', 'BibTeX', and 'EndNote'. Includes tools for duplicate detection, search strategy optimization, cross-validation frameworks, meta-analysis of benchmark results, power analysis for study design, and reproducibility package creation. Optionally connects to 'PubMed' for direct database searching and real-time strategy comparison using the 'E-utilities' 'API'. Enhanced with bootstrap comparison methods, 'McNemar' test for strategy evaluation, and comprehensive visualization tools for performance assessment. Methods based on Manning et al. (2008) for information retrieval metrics, Moher et al. (2009) for 'PRISMA' guidelines, and Sampson et al. (2006) for systematic review search methodology.

License GPL-3

URL https://github.com/chaoliu-cl/searchAnalyzeR

BugReports https://github.com/chaoliu-cl/searchAnalyzeR/issues

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports R6 (>= 2.5.0), ggplot2 (>= 3.4.0), openxlsx (>= 4.2.5), stringdist (>= 0.9.10), lubridate (>= 1.9.0), digest (>= 0.6.31), stats, utils

Suggests testthat (>= 3.0.0), covr (>= 3.6.0), dplyr (>= 1.1.0), shiny (>= 1.7.0), shinydashboard (>= 0.7.2), flexdashboard (>= 0.6.0), plotly (>= 4.10.0), DT (>= 0.27), boot (>= 1.3-28), pwr (>= 1.3-0), progress (>= 1.2.2), rmarkdown (>= 2.20), knitr (>= 1.42), glue (>= 1.6.0), rentrez (>= 1.2.3), xml2 (>= 1.3.0), readr (>= 2.1.0), jsonlite (>= 1.8.0), revtools, tidyr, patchwork, metagear

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Chao Liu [aut, cre] (ORCID: <https://orcid.org/0000-0002-9979-8272>)

Maintainer Chao Liu <chaoliu@cedarville.edu>

Repository CRAN

Date/Publication 2025-08-28 13:10:02 UTC

Contents

`analysis_env` *Create a Temporary Analysis Environment*

Description

Creates a temporary environment for analysis that isolates objects from the global environment. This helps prevent memory leaks and allows for easy cleanup after analysis.

Usage

```
analysis_env(parent_env = parent.frame(), cleanup = TRUE)
```

Arguments

`parent_env`	Environment to use as parent (default: parent.frame())
`cleanup`	Logical, whether to automatically clean up on exit

Value

New environment for analysis

auto_detect_columns *Auto-detect Column Mappings*

Description

Auto-detect Column Mappings

Usage

```
auto_detect_columns(results)
```

Arguments

results	Data frame to analyze

Value

Named vector of column mappings

BenchmarkValidator *Benchmark Validation System*

Description

A comprehensive validation framework for testing search strategies against established benchmark datasets across multiple domains.

Details

The BenchmarkValidator class provides tools for:

- Cross-domain validation across medical, environmental, social science domains
- Sensitivity analysis for search parameters
- Statistical comparison of strategy performance
- Reproducible benchmark testing

Fields

benchmarks List of benchmark datasets with known relevant articles

Methods

new() Initialize a new BenchmarkValidator instance

validate_strategy(search_strategy, benchmark_name) Validate against specific benchmark

cross_domain_validation(search_strategy) Test across multiple domains

sensitivity_analysis(base_strategy, parameter_ranges) Parameter sensitivity testing

Public fields

`benchmarks` List of benchmark datasets

Methods

Public methods:

- `BenchmarkValidator$new()`
- `BenchmarkValidator$add_benchmark()`
- `BenchmarkValidator$validate_strategy()`
- `BenchmarkValidator$validate_single_benchmark()`
- `BenchmarkValidator$cross_domain_validation()`
- `BenchmarkValidator$sensitivity_analysis()`
- `BenchmarkValidator$clone()`

Method `new()`**:** Creates a new BenchmarkValidator instance and loads benchmark datasets. This method is called automatically when creating a new validator with `BenchmarkValidator$new()`.

Usage:

```
BenchmarkValidator$new()
```

Returns: No return value, called for side effects (loading benchmarks) Add a custom benchmark dataset

Method `add_benchmark()`**:**

Usage:

```
BenchmarkValidator$add_benchmark(name, corpus, relevant_ids)
```

Arguments:

`name` Name of the benchmark

`corpus` Data frame with article corpus

`relevant_ids` Vector of relevant article IDs

Returns: No return value, called for side effects Validate search strategy against benchmarks

Method `validate_strategy()`**:**

Usage:

```
BenchmarkValidator$validate_strategy(search_strategy, benchmark_name = "all")
```

Arguments:

`search_strategy` Search strategy object

`benchmark_name` Name of benchmark dataset

Returns: Validation results Validate against single benchmark (PUBLIC METHOD)

Method `validate_single_benchmark()`**:**

Usage:

```
BenchmarkValidator$validate_single_benchmark(search_strategy, benchmark_name)
```

Arguments:

search_strategy Search strategy object

benchmark_name Name of benchmark dataset

Returns: Validation results Cross-domain validation

Method `cross_domain_validation()`**:**

Usage:

```
BenchmarkValidator$cross_domain_validation(search_strategy)
```

Arguments:

search_strategy Search strategy object

Returns: Cross-domain validation results Sensitivity analysis for search parameters

Method `sensitivity_analysis()`**:**

Usage:

```
BenchmarkValidator$sensitivity_analysis(base_strategy, parameter_ranges)
```

Arguments:

base_strategy Base search strategy

parameter_ranges List of parameter ranges to test

Returns: Sensitivity analysis results

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BenchmarkValidator$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create validator
validator <- BenchmarkValidator$new()

# Check available benchmarks
print(names(validator$benchmarks))

# Define search strategy
strategy <- list(
  terms = c("systematic review", "meta-analysis"),
  databases = c("PubMed", "Embase")
)

# Create sample data for validation
sample_data <- data.frame(
  id = paste0("art", 1:20),
  title = paste("Article", 1:20),
  abstract = paste("Abstract", 1:20),
  source = "Journal",
  date = Sys.Date()
```

```
)

# Add custom benchmark
validator$add_benchmark("custom", sample_data, paste0("art", 1:5))

# Validate against custom benchmark
results <- validator$validate_strategy(strategy, "custom")
```

`bootstrap_compare` *Bootstrap Comparison of Search Strategies*

Description

Bootstrap Comparison of Search Strategies

Usage

```
bootstrap_compare(
  strategy1_results,
  strategy2_results,
  gold_standard,
  n_bootstrap = 1000
)
```

Arguments

`strategy1_results`	Results from first strategy
`strategy2_results`	Results from second strategy
`gold_standard`	Vector of relevant article IDs
`n_bootstrap`	Number of bootstrap samples

Value

Bootstrap comparison results

cache_manage *Manage Search Results Cache*

Description

Manages a cache of search results to avoid redundant database queries while keeping memory usage under control.

Usage

```
cache_manage(
  operation,
  key = NULL,
  value = NULL,
  max_size = 500,
  max_items = 50
)
```

Arguments

operation	Operation to perform ("add", "get", "clear", "status")
key	Cache key (usually search query)
value	Value to cache (for "add" operation)
max_size	Maximum cache size in MB (default: 500)
max_items	Maximum number of items to cache (default: 50)

Value

Varies by operation

calc_ci *Calculate Confidence Intervals*

Description

Calculate Confidence Intervals

Usage

```
calc_ci(x, conf_level = 0.95, method = "normal")
```

Arguments

x	Numeric vector
conf_level	Confidence level (0-1)
method	Method for calculation ("normal", "bootstrap")

Value

List with lower and upper bounds

`calc_cosine` *Calculate Cosine Similarity*

Description

Calculate Cosine Similarity

Usage

```
calc_cosine(text1, text2)
```

Arguments

`text1`	First text string
`text2`	Second text string

Value

Cosine similarity score

`calc_coverage` *Calculate Coverage Metrics Across Databases*

Description

Calculate Coverage Metrics Across Databases

Usage

```
calc_coverage(results_by_database, gold_standard)
```

Arguments

`results_by_database`	List of result sets by database
`gold_standard`	Vector of relevant article IDs

Details

Calculates coverage metrics for each database and overall:

- **coverage_count**: Number of relevant articles found by each database
- **coverage_rate**: Proportion of relevant articles found by each database
- **unique_coverage**: Number of relevant articles found only by this database
- **total_coverage**: Overall proportion of relevant articles found by all databases
- **redundancy_rate**: Proportion of duplicate results across databases

Value

List containing coverage statistics

Examples

```
# Create sample data
results_db1 <- c("art1", "art2", "art3", "art4")
results_db2 <- c("art2", "art3", "art5", "art6")
results_by_db <- list("Database1" = results_db1, "Database2" = results_db2)
gold_standard <- c("art1", "art3", "art5", "art7", "art8")

coverage <- calc_coverage(results_by_db, gold_standard)
print(coverage$total_coverage)
```

---

calc_efficiency *Calculate Search Efficiency Metrics*

Description

Calculate Search Efficiency Metrics

Usage

```
calc_efficiency(search_time, results_count, relevant_count)
```

Arguments

search_time	Time taken to execute search (in seconds)
results_count	Number of results retrieved
relevant_count	Number of relevant results

Details

Calculates various efficiency metrics for search performance:

- **time_per_result**: Average time to retrieve each result
- **time_per_relevant**: Average time to retrieve each relevant result
- **relevant_ratio**: Proportion of results that are relevant
- **efficiency_score**: Overall efficiency combining time and relevance

Value

List containing efficiency metrics

Examples

```
efficiency <- calc_efficiency(search_time = 30, results_count = 100, relevant_count = 15)
print(paste("Efficiency score:", round(efficiency$efficiency_score, 4)))
```

---

calc_jaccard *Calculate Jaccard Similarity*

Description

Calculate Jaccard Similarity

Usage

```
calc_jaccard(text1, text2)
```

Arguments

text1	First text string
text2	Second text string

Value

Jaccard similarity score

---

calc_precision_recall *Calculate Precision and Recall Metrics*

Description

Calculate Precision and Recall Metrics

Usage

```
calc_precision_recall(retrieved, relevant, total_relevant = NULL)
```

Arguments

retrieved	Vector of retrieved article IDs
relevant	Vector of relevant article IDs (gold standard)
total_relevant	Total number of relevant articles in corpus

Details

Calculates standard information retrieval metrics:

- **Precision**: TP/(TP+FP) - proportion of retrieved articles that are relevant
- **Recall**: TP/(TP+FN) - proportion of relevant articles that were retrieved
- **F1 Score**: Harmonic mean of precision and recall
- **Number Needed to Read**: 1/precision - articles needed to read to find one relevant

where TP = True Positives, FP = False Positives, FN = False Negatives

References

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval.

Examples

```
retrieved_ids <- c("art1", "art2", "art3", "art4", "art5")
relevant_ids <- c("art1", "art3", "art6", "art7")
metrics <- calc_precision_recall(retrieved_ids, relevant_ids)
print(paste("Precision:", round(metrics$precision, 3)))
print(paste("Recall:", round(metrics$recall, 3)))
```

---

calc_sample_size *Power Analysis for Search Strategy Evaluation*

Description

Power Analysis for Search Strategy Evaluation

Usage

```
calc_sample_size(
  effect_size = 0.1,
  alpha = 0.05,
  power = 0.8,
  baseline_f1 = 0.7
)
```

Arguments

effect_size	Expected effect size (difference in F1 scores)
alpha	Significance level
power	Desired statistical power
baseline_f1	Baseline F1 score

Value

Required sample size

calc_search_stats *Calculate Search Result Statistics*

Description

Calculate Search Result Statistics

Usage

```
calc_search_stats(search_results)
```

Arguments

search_results	Data frame with search results

Value

List of summary statistics

calc_strategy_comparison *Calculate Strategy Comparison Metrics*

Description

Calculate Strategy Comparison Metrics

Usage

```
calc_strategy_comparison(strategy1_results, strategy2_results, gold_standard)
```

Arguments

strategy1_results	Vector of article IDs from strategy 1
strategy2_results	Vector of article IDs from strategy 2
gold_standard	Vector of relevant article IDs

Details

Compares two search strategies across multiple dimensions:

- **overlap_analysis**: Articles found by both, one, or neither strategy
- **performance_comparison**: Precision, recall, F1 for each strategy
- **complementarity**: How well strategies complement each other
- **efficiency_comparison**: Relative efficiency metrics

Value

List containing comparison metrics

Examples

```
strategy1 <- c("art1", "art2", "art3", "art4", "art5")
strategy2 <- c("art3", "art4", "art5", "art6", "art7")
gold_standard <- c("art1", "art3", "art5", "art8", "art9")

comparison <- calc_strategy_comparison(strategy1, strategy2, gold_standard)
print(comparison$overlap_analysis)
```

---

calc_temporal_coverage

Calculate Temporal Coverage Metrics

Description

Calculate Temporal Coverage Metrics

Usage

```
calc_temporal_coverage(search_results, target_date_range = NULL)
```

Arguments

search_results	Data frame with search results including date column
target_date_range	Vector of two dates defining the target time period

Details

Analyzes the temporal distribution of search results:

- **coverage_by_year**: Number of articles by publication year
- **target_period_coverage**: Proportion of results in target date range
- **temporal_gaps**: Years with no results in the target period
- **peak_years**: Years with highest number of results

Value

List containing temporal coverage statistics

Examples

```
# Create sample data
search_results <- data.frame(
  id = paste0("art", 1:20),
  date = seq(as.Date("2010-01-01"), as.Date("2023-12-31"), length.out = 20)
)
target_range <- c(as.Date("2015-01-01"), as.Date("2020-12-31"))

temporal_metrics <- calc_temporal_coverage(search_results, target_range)
print(temporal_metrics$target_period_coverage)
```

`calc_tes` *Calculate Term Effectiveness Score*

Description

Calculates a balanced effectiveness score for individual search terms using the harmonic mean of precision and coverage. This provides a single metric to evaluate how well each term performs in retrieving relevant articles.

Usage

```
calc_tes(term_analysis, score_name = "tes")
```

Arguments

`term_analysis`	Data frame from term_effectiveness() function
`score_name`	Name for the new score column (default: "tes")

Details

The Term Effectiveness Score (TES) is calculated as:

$$TES = 2 \times \frac{precision \times coverage}{precision + coverage}$$

Where:

- **Precision**: Proportion of retrieved articles that are relevant
- **Coverage**: Proportion of term-specific relevant articles that were retrieved

This differs from the traditional F1 score in that it uses **coverage** (term-specific relevance) rather than **recall** (overall strategy relevance).

Key Differences from F1 Score:

- **F1 Score**: Precision × Recall (strategy-level performance)
- **TES**: Precision × Coverage (term-level performance)
- **Recall**: Relevant articles found / All relevant articles
- **Coverage**: Relevant articles found / Term-specific relevant articles

Value

Data frame with added effectiveness score column

See Also

term_effectiveness for calculating term precision and coverage calc_precision_recall for strategy-level F1 scores

Examples

```
# Create sample term analysis
terms <- c("diabetes", "treatment", "clinical")
search_results <- data.frame(
  id = paste0("art", 1:20),
  title = paste("Study on", sample(terms, 20, replace = TRUE)),
  abstract = paste("Research about", sample(terms, 20, replace = TRUE))
)
gold_standard <- paste0("art", c(1, 3, 5, 7, 9))

# Analyze term effectiveness
term_analysis <- term_effectiveness(terms, search_results, gold_standard)

# Calculate effectiveness scores
term_scores <- calc_tes(term_analysis)
print(term_scores[order(term_scores$tes, decreasing = TRUE), ])
```

---

calc_text_sim *Calculate Text Similarity*

Description

Calculate Text Similarity

Usage

```
calc_text_sim(text1, text2, method = "jaccard")
```

Arguments

text1	First text string
text2	Second text string
method	Similarity method ("jaccard", "cosine", "jaro_winkler")

Value

Similarity score between 0 and 1

check_deps *Utility Functions for searchAnalyzeR Package*

Description

This file contains general utility functions used throughout the package.

Usage

```
check_deps(required_packages, install_missing = FALSE)
```

Arguments

required_packages	Character vector of required package names
install_missing	Logical, whether to suggest installing missing packages

Details

This function checks if required packages are installed using `requireNamespace` to check availability without loading packages. For CRAN compliance, this function does not automatically install packages.

Value

Logical vector indicating which packages are available

Examples

```
# Check if packages are available
required <- c("ggplot2", "dplyr")
availability <- check_deps(required)
print(availability)

# Get suggestions for missing packages
required_with_missing <- c("ggplot2", "dplyr", "nonexistent_package")
availability <- check_deps(required_with_missing, install_missing = TRUE)
print(availability)
```

chunk_process *Process Large Dataset in Chunks*

Description

Generic function to process a large dataset in manageable chunks to reduce memory usage.

Usage

```
chunk_process(data, chunk_size = 10000, fn, combine_fn = rbind, ...)
```

Arguments

data	Large data frame to process
chunk_size	Number of rows per chunk
fn	Function to apply to each chunk
combine_fn	Function to combine results from chunks
...	Additional arguments passed to fn

Value

Combined results after processing all chunks

clean_col_names *Clean Column Names*

Description

Clean Column Names

Usage

```
clean_col_names(names)
```

Arguments

names	Character vector of column names

Value

Cleaned column names

clean_text *Clean Text Fields*

Description

Clean Text Fields

Usage

```
clean_text(text)
```

Arguments

text	Character vector to clean

Value

Cleaned character vector

compare_strategies *Benchmark Testing Framework for Search Strategies*

Description

This file contains advanced benchmark testing capabilities including cross-validation, statistical testing, and performance comparison methods. Statistical Significance Testing for Search Performance

Usage

```
compare_strategies(
  strategy1_results,
  strategy2_results,
  gold_standard,
  test_type = "mcnemar",
  alpha = 0.05
)
```

Arguments

strategy1_results	Results from first search strategy
strategy2_results	Results from second search strategy
gold_standard	Vector of relevant article IDs
test_type	Type of statistical test ("mcnemar", "paired_t", "wilcoxon")
alpha	Significance level

Value

Statistical test results

`compare_terms` *Compare Terms Across Strategies*

Description

Compares the effectiveness of terms across multiple search strategies to identify which terms perform best in different contexts.

Usage

```
compare_terms(term_list, top_n = 5)
```

Arguments

`term_list`	Named list of term_analysis objects from different strategies
`top_n`	Number of top terms to compare (default: 5)

Details

This function:

- Calculates effectiveness scores for each strategy
- Identifies top terms in each strategy
- Creates a comparison matrix showing performance across strategies

Value

Data frame comparing term effectiveness across strategies

`complete_search_workflow`
Complete Search and Analysis Workflow

Description

Perform a complete workflow: search databases, analyze results, generate reports.

Usage

```
complete_search_workflow(
  search_terms,
  databases = "pubmed",
  gold_standard = NULL,
  max_results = 100,
  date_range = NULL,
  output_dir = NULL
)
```

Arguments

search_terms	Character vector of search terms
databases	Vector of databases to search
gold_standard	Optional vector of known relevant article IDs
max_results	Maximum results to retrieve
date_range	Optional date range for search
output_dir	Directory for reports (uses tempdir() by default)

Value

List containing search results, analysis, and report paths

Examples

```
# Complete workflow
results <- complete_search_workflow(
  search_terms = "diabetes treatment clinical trial",
  databases = "pubmed",
  max_results = 50,
  date_range = c("2022/01/01", "2023/12/31")
)

# View summary
print(results$summary)

# Access detailed metrics
print(results$analysis$metrics)
```

create_analysis_template *Create Analysis Template Script*

Description

Create Analysis Template Script

Usage

```
create_analysis_template(file_path)
```

Arguments

file_path	Output file path

create_data_dictionary *Create Data Dictionary*

Description

Create Data Dictionary

Usage

```
create_data_dictionary(file_path, search_results)
```

Arguments

file_path	Output file path
search_results	Search results data

create_data_package *Create Data Package for Sharing*

Description

Create Data Package for Sharing

Usage

```
create_data_package(
  search_results,
  analysis_results = NULL,
  output_dir = NULL,
  package_name = "search_analysis_package"
)
```

Arguments

search_results	Data frame with search results
analysis_results	List of analysis results
output_dir	Directory to create the package (defaults to tempdir())
package_name	Name of the package

Value

Path to created package directory

Examples

```
# Create sample data
search_results <- data.frame(
  id = paste0("art", 1:10),
  title = paste("Study", 1:10),
  abstract = paste("Abstract", 1:10),
  source = "Journal",
  date = Sys.Date(),
  stringsAsFactors = FALSE
)

# Create data package (writes to tempdir())
package_path <- create_data_package(search_results)
print(package_path)
```

create_package_manifest *Create Package Manifest*

Description

Create Package Manifest

Usage

```
create_package_manifest(package_dir)
```

Arguments

package_dir	Package directory

create_package_readme *Create Package README*

Description

Create Package README

Usage

```
create_package_readme(package_dir, search_results, analysis_results)
```

Arguments

package_dir	Package directory
search_results	Search results data
analysis_results	Analysis results

create_prisma *Create PRISMA Flow Diagram with Proper Spacing and Text Enclosure*

Description

Create PRISMA Flow Diagram with Proper Spacing and Text Enclosure

Usage

```
create_prisma(flow_data)
```

Arguments

flow_data	List containing PRISMA flow numbers

Value

ggplot object

create_progress_bar *Create Progress Bar for Long Operations*

Description

Create Progress Bar for Long Operations

Usage

```
create_progress_bar(total, format = "[:bar] :percent :elapsed")
```

Arguments

total	Total number of iterations
format	Progress bar format string

Value

Progress bar object

create_strategy *Create Default Search Strategy Template*

Description

Create Default Search Strategy Template

Usage

```
create_strategy(terms, databases, date_range = NULL, filters = NULL)
```

Arguments

terms	Character vector of search terms
databases	Character vector of databases
date_range	Date vector of length 2 (start, end)
filters	List of additional filters

Value

Search strategy list

create_summary *Create Summary Statistics Table*

Description

Create Summary Statistics Table

Usage

```
create_summary(data, numeric_vars = NULL, categorical_vars = NULL)
```

Arguments

data	Data frame
numeric_vars	Character vector of numeric variable names
categorical_vars	Character vector of categorical variable names

Value

Summary statistics data frame

cv_strategy *Cross-Validation Framework for Search Strategies*

Description

Cross-Validation Framework for Search Strategies

Usage

```
cv_strategy(
  search_strategy,
  validation_corpus,
  gold_standard,
  k_folds = 5,
  stratified = TRUE
)
```

Arguments

search_strategy	Search strategy object
validation_corpus	Full corpus for validation
gold_standard	Vector of relevant article IDs
k_folds	Number of folds for cross-validation
stratified	Whether to use stratified sampling

Value

Cross-validation results

detect_doi_dupes *Detect DOI-based Duplicates*

Description

Detect DOI-based Duplicates

Usage

```
detect_doi_dupes(results)
```

Arguments

results	Data frame with search results

Value

Data frame with DOI duplicates marked

`detect_dupes` *Detect and Remove Duplicate Records*

Description

Detect and Remove Duplicate Records

Usage

```
detect_dupes(results, method = "exact", similarity_threshold = 0.85)
```

Arguments

`results`	Standardized search results data frame
`method`	Method for duplicate detection ("exact", "fuzzy", "doi")
`similarity_threshold`	Threshold for fuzzy matching (0-1)

Details

This function provides three methods for duplicate detection:

- **exact**: Matches on title and first 100 characters of abstract
- **fuzzy**: Uses Jaro-Winkler string distance for similarity matching
- **doi**: Matches based on cleaned DOI strings

For fuzzy matching, similarity_threshold should be between 0 and 1, where 1 means identical strings. A threshold of 0.85 typically works well for academic titles.

Value

Data frame with duplicates marked and removed

detect_exact_dupes *Detect Exact Duplicates*

Description

Detect Exact Duplicates

Usage

```
detect_exact_dupes(results)
```

Arguments

results	Data frame with search results

Value

Data frame with exact duplicates marked

detect_fuzzy_dupes *Detect Fuzzy Duplicates*

Description

Detect Fuzzy Duplicates

Usage

```
detect_fuzzy_dupes(results, threshold = 0.85)
```

Arguments

results	Data frame with search results
threshold	Similarity threshold

Value

Data frame with fuzzy duplicates marked

export_metrics *Export Analysis Metrics*

Description

Export Analysis Metrics

Usage

```
export_metrics(metrics, file_path, format = "xlsx")
```

Arguments

metrics	List of calculated metrics
file_path	Output file path
format	Export format ("csv", "xlsx", "json")

Value

File path of created file

Examples

```
# Create sample metrics
metrics <- list(
  basic = list(total_records = 100, unique_records = 95),
  precision_recall = list(precision = 0.8, recall = 0.6, f1_score = 0.69)
)

# Export metrics (writes to tempdir())
output_file <- export_metrics(metrics, file.path(tempdir(), "metrics.xlsx"))
print(output_file)
```

export_metrics_csv *Export Metrics to CSV*

Description

Export Metrics to CSV

Usage

```
export_metrics_csv(metrics, file_path)
```

Arguments

`metrics`	List of calculated metrics
`file_path`	Output file path

Value

File path of created file

`export_metrics_json` *Export Metrics to JSON*

Description

Export Metrics to JSON

Usage

```
export_metrics_json(metrics, file_path)
```

Arguments

`metrics`	List of calculated metrics
`file_path`	Output file path

Value

File path of created file

`export_metrics_xlsx` *Export Metrics to Excel*

Description

Export Metrics to Excel

Usage

```
export_metrics_xlsx(metrics, file_path)
```

Arguments

`metrics`	List of calculated metrics
`file_path`	Output file path

Value

File path of created file

`export_results` *Export Utilities for Search Analysis Results*

Description

This file contains functions for exporting search analysis results, reports, and data in various formats. Export Search Results to Multiple Formats

Usage

```
export_results(
  search_results,
  file_path = NULL,
  formats = c("csv", "xlsx"),
  include_metadata = TRUE
)
```

Arguments

`search_results`	Data frame with search results
`file_path`	Base file path (without extension). If NULL, uses tempdir()
`formats`	Vector of formats to export ("csv", "xlsx", "ris", "bibtex")
`include_metadata`	Logical, whether to include metadata sheets/files

Details

This function exports search results to multiple standard formats used in systematic reviews and reference management. Supported formats include:

- **CSV**: Comma-separated values for data analysis
- **Excel**: Multi-sheet workbook with metadata
- **RIS**: Reference Information Systems format for reference managers
- **BibTeX**: LaTeX bibliography format
- **EndNote**: Thomson Reuters EndNote format

Value

Vector of created file paths

Examples

```
# Create sample search results
search_results <- data.frame(
  id = paste0("article_", 1:5),
  title = paste("Sample Article", 1:5),
  abstract = paste("Abstract for article", 1:5),
  source = "Sample Journal",
  date = Sys.Date(),
  stringsAsFactors = FALSE
)

# Export to multiple formats (writes to tempdir())
output_files <- export_results(search_results, formats = c("csv", "xlsx"))
print(output_files)
```

---

export_to_bibtex *Export to BibTeX Format*

Description

Export to BibTeX Format

Usage

```
export_to_bibtex(search_results, file_path)
```

Arguments

search_results	Data frame with search results
file_path	Output file path

Value

File path of created file

---

export_to_csv *Export to CSV Format*

Description

Export to CSV Format

Usage

```
export_to_csv(search_results, file_path, include_metadata = TRUE)
```

Arguments

search_results	Data frame with search results
file_path	Output file path
include_metadata	Logical, whether to create metadata file

Value

File path of created file

---

export_to_endnote *Export to EndNote Format*

Description

Export to EndNote Format

Usage

```
export_to_endnote(search_results, file_path)
```

Arguments

search_results	Data frame with search results
file_path	Output file path

Value

File path of created file

---

export_to_ris *Export to RIS Format*

Description

Export to RIS Format

Usage

```
export_to_ris(search_results, file_path)
```

Arguments

search_results	Data frame with search results
file_path	Output file path

Value

File path of created file

`export_to_xlsx` *Export to Excel Format with Multiple Sheets*

Description

Export to Excel Format with Multiple Sheets

Usage

```
export_to_xlsx(search_results, file_path, include_metadata = TRUE)
```

Arguments

`search_results`	Data frame with search results
`file_path`	Output file path
`include_metadata`	Logical, whether to include metadata sheets

Value

File path of created file

`export_validation` *Export Validation Results*

Description

Export Validation Results

Usage

```
export_validation(validation_results, file_path, format = "xlsx")
```

Arguments

`validation_results`	Results from benchmark validation
`file_path`	Output file path
`format`	Export format ("xlsx", "csv", "json")

Value

File path of created file

Examples

```
# Create sample validation results
validation_results <- list(
  precision = 0.8,
  recall = 0.6,
  f1_score = 0.69,
  true_positives = 24,
  false_positives = 6,
  false_negatives = 16
)

# Export validation results (writes to tempdir())
output_file <- export_validation(
  validation_results,
  file.path(tempdir(), "validation.xlsx")
)
print(output_file)
```

---

export_validation_csv *Export Validation Results to CSV*

Description

Export Validation Results to CSV

Usage

```
export_validation_csv(validation_results, file_path)
```

Arguments

validation_results	Validation results
file_path	Output file path

Value

File path of created file

`export_validation_json` *Export Validation Results to JSON*

Description

Export Validation Results to JSON

Usage

```
export_validation_json(validation_results, file_path)
```

Arguments

`validation_results`	Validation results
`file_path`	Output file path

Value

File path of created file

`export_validation_xlsx` *Export Validation Results to Excel*

Description

Export Validation Results to Excel

Usage

```
export_validation_xlsx(validation_results, file_path)
```

Arguments

`validation_results`	Validation results
`file_path`	Output file path

Value

File path of created file

extract_screening *Extract Screening Data Structure*

Description

Extract Screening Data Structure

Usage

```
extract_screening(search_results, screening_decisions = NULL)
```

Arguments

search_results	Combined search results
screening_decisions	Optional data frame with screening decisions

Value

Data frame with screening structure for PRISMA

find_top_terms *Find Top Performing Terms*

Description

Identifies the top-performing search terms based on their effectiveness scores and optionally creates highlighted visualizations.

Usage

```
find_top_terms(
  term_analysis,
  n = 3,
  score_col = "tes",
  plot = TRUE,
  plot_type = "precision_only"
)
```

Arguments

term_analysis	Data frame from term_effectiveness() function
n	Number of top terms to identify (default: 3)
score_col	Name of the score column to use for ranking (default: "tes")
plot	Whether to create a highlighted plot (default: TRUE)
plot_type	Type of plot for highlighting ("precision_only", "coverage_only", "precision_coverage")

Details

This function:

1. Calculates effectiveness scores if not already present
2. Identifies the top N performing terms
3. Optionally creates a visualization highlighting these terms

Value

List containing top terms and optionally a highlighted plot

---

format_numbers *Format Numbers for Display*

Description

Format Numbers for Display

Usage

```
format_numbers(x, digits = 3, percent = FALSE)
```

Arguments

x	Numeric vector
digits	Number of decimal places
percent	Logical, whether to format as percentage

Value

Formatted character vector

---

gen_repro_seed *Generate Reproducible Random Seed*

Description

Generate Reproducible Random Seed

Usage

```
gen_repro_seed(base_string = "searchAnalyzeR")
```

Arguments

base_string	Base string for seed generation

Details

This function generates a reproducible seed based on a string input. It does not set the seed automatically - users should call set.seed() themselves if they want to use the generated seed.

Value

Integer seed value

Examples

```
# Generate a seed value
seed_value <- gen_repro_seed("my_analysis")

# User can choose to set it

set.seed(seed_value)
sample(1:10, 3)
```

---

get_pkg_versions *Extract Package Version Information*

Description

Extract Package Version Information

Usage

```
get_pkg_versions(
  packages = c("searchAnalyzeR", "ggplot2", "lubridate", "openxlsx")
)
```

Arguments

packages	Character vector of package names

Value

Data frame with package version information

is_empty *Check if Object is Empty*

Description

Check if Object is Empty

Usage

```
is_empty(x)
```

Arguments

x	Object to check

Value

Logical indicating if object is empty

mem_cleanup *Clean up Search Analysis Objects to Free Memory*

Description

Removes intermediate and temporary objects created during analysis to free memory. This is particularly useful for large-scale analyses.

Usage

```
mem_cleanup(keep_results = TRUE, verbose = TRUE, env = parent.frame())
```

Arguments

keep_results	Logical, whether to keep final results
verbose	Logical, whether to print memory freed information
env	Environment to clean (defaults to parent.frame())

Value

Amount of memory freed in MB

mem_monitor *Monitor Memory Usage During Function Execution*

Description

Wraps a function call with memory usage monitoring, reporting memory usage before, during, and after execution.

Usage

```
mem_monitor(fn, interval = 1, ...)
```

Arguments

fn	Function to execute
interval	Time interval in seconds for memory checks during execution
...	Arguments passed to fn

Value

Result of fn with memory usage statistics as an attribute

mem_usage *Get Current Memory Usage*

Description

Reports the current memory usage of the R session.

Usage

```
mem_usage(units = "MB", include_gc = FALSE)
```

Arguments

units	Units for reporting memory usage ("MB", "GB", or "KB")
include_gc	Logical, whether to run garbage collection before measuring

Value

Named list with memory usage information

merge_results *Merge Search Results from Multiple Sources*

Description

Merge Search Results from Multiple Sources

Usage

```
merge_results(result_list, deduplicate = TRUE, dedup_method = "exact")
```

Arguments

result_list	List of standardized search result data frames
deduplicate	Logical, whether to remove duplicates
dedup_method	Method for duplicate detection

Value

Combined and deduplicated data frame

meta_analyze *Meta-Analysis of Benchmark Results*

Description

Meta-Analysis of Benchmark Results

Usage

```
meta_analyze(benchmark_results, strategy_name, metric = "f1_score")
```

Arguments

benchmark_results	List of benchmark result objects
strategy_name	Name of strategy to analyze across benchmarks
metric	Metric to meta-analyze ("precision", "recall", "f1_score")

Value

Meta-analysis results

opt_df *Memory-Efficient Data Frame*

Description

Converts a data frame to a memory-efficient format by optimizing column types.

Usage

```
opt_df(df, compress_strings = FALSE, verbose = TRUE)
```

Arguments

df	Data frame to optimize
compress_strings	Logical, whether to convert character columns to factors
verbose	Logical, whether to print memory savings information

Value

Memory-efficient version of the input data frame

plot_db_performance *Create Database Performance Comparison*

Description

Create Database Performance Comparison

Usage

```
plot_db_performance(results_by_database, gold_standard = NULL)
```

Arguments

results_by_database	List of result sets by database
gold_standard	Vector of relevant article IDs

Value

ggplot object

plot_keyword_eff *Create Keyword Effectiveness Analysis Plot*

Description

Create Keyword Effectiveness Analysis Plot

Usage

```
plot_keyword_eff(search_results, search_terms, gold_standard = NULL)
```

Arguments

search_results	Data frame with search results
search_terms	Vector of search terms
gold_standard	Vector of relevant article IDs

Value

ggplot object

plot_overview *Visualization Functions for Search Strategy Analysis*

Description

This file contains all visualization functions used by the SearchAnalyzer class and other components of the searchAnalyzeR package. Create Overview Performance Plot

Usage

```
plot_overview(metrics)
```

Arguments

metrics	List of calculated metrics from SearchAnalyzer

Details

Creates a focused overview plot displaying the core search performance metrics:

- Precision: Proportion of retrieved articles that are relevant
- Recall: Proportion of relevant articles that were retrieved
- F1 Score: Harmonic mean of precision and recall

The plot uses color coding to distinguish between metric types and displays exact values on top of each bar.

Value

ggplot object showing key performance indicators

See Also

plot_pr_curve, plot_temporal

Examples

```
# Assume you have calculated metrics
metrics <- list(
  precision_recall = list(precision = 0.8, recall = 0.6, f1_score = 0.69)
)

overview_plot <- plot_overview(metrics)
print(overview_plot)
```

---

plot_pr_curve *Create Precision-Recall Curve*

Description

Create Precision-Recall Curve

Usage

```
plot_pr_curve(retrieved, relevant, thresholds = seq(0, 1, 0.05))
```

Arguments

retrieved	Vector of retrieved article IDs
relevant	Vector of relevant article IDs
thresholds	Vector of threshold values

Value

ggplot object

plot_sensitivity *Create Sensitivity Analysis Heatmap*

Description

Create Sensitivity Analysis Heatmap

Usage

```
plot_sensitivity(sensitivity_results)
```

Arguments

sensitivity_results	Results from sensitivity analysis

Value

ggplot object

plot_temporal *Create Temporal Coverage Plot*

Description

Create Temporal Coverage Plot

Usage

```
plot_temporal(search_results, gold_standard = NULL)
```

Arguments

search_results	Data frame with search results including date column
gold_standard	Vector of relevant article IDs

Value

ggplot object

plot_term_effectiveness *Plot Term Effectiveness Results*

Description

Plot Term Effectiveness Results

Usage

```
plot_term_effectiveness(
  term_analysis,
  plot_type = "precision_coverage",
  highlight_terms = NULL,
  title_override = NULL,
  show_values = TRUE
)
```

Arguments

term_analysis	Result from term_effectiveness function
plot_type	Type of plot to create ("precision_coverage", "counts", "comparison", "precision_only", "coverage_only")
highlight_terms	Optional character vector of terms to highlight
title_override	Optional custom title for the plot
show_values	Logical, whether to show values on bars/points (default: TRUE)

Details

This function creates visualizations of term effectiveness results with enhanced options for creating individual, clean plots. New plot types include "precision_only" and "coverage_only" for focused analysis.

Value

A ggplot object if ggplot2 is available, otherwise NULL with a message

Examples

```
# Create sample data for demonstration
search_results <- data.frame(
  id = paste0("art", 1:10),
  title = c("Diabetes treatment", "Clinical trial", "Diabetes study",
            "Treatment options", "New therapy", "Glucose control",
            "Insulin therapy", "Management of diabetes", "Clinical study",
            "Therapy comparison"),
```

```
  abstract = c("This study examines diabetes treatments.",
               "A clinical trial on new treatments.",
               "Diabetes research findings.",
               "Comparison of treatment options.",
               "Novel therapy approach.",
               "Methods to control glucose levels.",
               "Insulin therapy effectiveness.",
               "Managing diabetes effectively.",
               "Clinical research protocols.",
               "Comparing therapy approaches.")
)

# Define search terms and gold standard
terms <- c("diabetes", "treatment", "clinical", "therapy")
gold_standard <- c("art1", "art3", "art7", "art8")

# First analyze term effectiveness
term_metrics <- term_effectiveness(terms, search_results, gold_standard)

# Create individual plots
precision_plot <- plot_term_effectiveness(term_metrics, "precision_only")
coverage_plot <- plot_term_effectiveness(term_metrics, "coverage_only")
bubble_plot <- plot_term_effectiveness(term_metrics, "precision_coverage")
```

`print.term_comparison` *Print Method for Term Comparison*

Description

Print Method for Term Comparison

Usage

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

Arguments

`x`	A term_comparison object
`...`	Further arguments passed to or from other methods

Value

Invisibly returns the input object

print.term_effectiveness *Print Method for term_effectiveness Objects*

Description

Print Method for term_effectiveness Objects

Usage

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

Arguments

x	A term_effectiveness object
...	Further arguments passed to or from other methods

Value

Invisibly returns the input object

PRISMAReporter *PRISMA-Compliant Report Generator*

Description

A comprehensive reporting system for generating PRISMA-compliant reports from systematic review search analyses.

Details

The PRISMAReporter class provides tools for:

- Generating comprehensive search strategy reports
- Creating PRISMA flow diagrams
- Documenting search strategies
- Exporting reports in multiple formats (HTML, PDF, Word)

Methods

`new()` Initialize a new PRISMAReporter instance

`generate_report(search_analysis, output_format, template_type)` Generate comprehensive search strategy report

`generate_prisma_diagram(screening_data)` Generate PRISMA flow diagram

`document_search_strategy(search_strategy)` Generate search strategy documentation

Methods

Public methods:

- PRISMAReporter$new()
- PRISMAReporter$generate_report()
- PRISMAReporter$generate_prisma_diagram()
- PRISMAReporter$document_search_strategy()
- PRISMAReporter$clone()

Method `new()`**:** Creates a new PRISMAReporter instance for generating PRISMA-compliant reports. Sets up the necessary template paths and configuration.

Usage:

```
PRISMAReporter$new()
```

Returns: No return value, called for side effects (initialization) Generate comprehensive search strategy report

Method `generate_report()`**:**

Usage:

```
PRISMAReporter$generate_report(
  search_analysis,
  output_format = "html",
  template_type = "comprehensive"
)
```

Arguments:

`search_analysis` SearchAnalyzer object

`output_format` Output format ("html", "pdf", "word")

`template_type` Type of report template

Returns: Path to generated report Generate PRISMA flow diagram

Method `generate_prisma_diagram()`**:**

Usage:

```
PRISMAReporter$generate_prisma_diagram(screening_data)
```

Arguments:

`screening_data` Data frame with screening results

Returns: ggplot object Generate search strategy documentation

Method `document_search_strategy()`**:**

Usage:

```
PRISMAReporter$document_search_strategy(search_strategy)
```

Arguments:

`search_strategy` Search strategy object

Returns: Formatted documentation

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PRISMAReporter$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# Create reporter
reporter <- PRISMAReporter$new()

# Create sample search strategy for documentation
search_strategy <- list(
  terms = c("systematic review", "meta-analysis", "evidence synthesis"),
  databases = c("PubMed", "Embase", "Cochrane"),
  date_range = as.Date(c("2020-01-01", "2023-12-31")),
  filters = list(language = "English", study_type = "RCT")
)

# Generate search strategy documentation
strategy_docs <- reporter$document_search_strategy(search_strategy)
print(strategy_docs)

# Create sample screening data for PRISMA diagram
screening_data <- data.frame(
  id = 1:100,
  duplicate = c(rep(FALSE, 80), rep(TRUE, 20)),
  title_abstract_screened = c(rep(TRUE, 80), rep(FALSE, 20)),
  full_text_eligible = c(rep(TRUE, 25), rep(FALSE, 75)),
  included = c(rep(TRUE, 15), rep(FALSE, 85)),
  excluded_title_abstract = c(rep(FALSE, 25), rep(TRUE, 55), rep(FALSE, 20)),
  excluded_full_text = c(rep(FALSE, 15), rep(TRUE, 10), rep(FALSE, 75))
)

# Generate PRISMA diagram
prisma_plot <- reporter$generate_prisma_diagram(screening_data)
print("PRISMA diagram created successfully")
```

PubMedConnector *PubMed Database Connector*

Description

A class for connecting to and searching PubMed database directly, then formatting results for analysis with searchAnalyzeR.

Details

This module provides functionality to search PubMed directly and integrate the results with search-AnalyzeR's analysis capabilities. PubMed Search Interface

This class uses the rentrez package to interface with NCBI's E-utilities to search PubMed and retrieve article metadata. Results are automatically formatted for use with SearchAnalyzer. If rentrez is not available, it provides simulated data for demonstration purposes.

Methods

`new()` Initialize a new PubMedConnector instance

`search(query, max_results, date_range)` Search PubMed database

`get_details(pmids)` Get detailed information for specific PMIDs

`format_for_analysis()` Format results for SearchAnalyzer

Public fields

`last_search_results` Raw results from last search

`formatted_results` Formatted results ready for analysis

`search_metadata` Metadata about the last search

`use_simulation` Flag indicating if simulation mode is active

Methods

Public methods:

- `PubMedConnector$new()`
- `PubMedConnector$search()`
- `PubMedConnector$get_details()`
- `PubMedConnector$format_for_analysis()`
- `PubMedConnector$get_search_summary()`
- `PubMedConnector$clone()`

Method `new()`**:** Initialize a new PubMedConnector instance

Usage:

```
PubMedConnector$new()
```

Returns: No return value, called for side effects Search PubMed database

Method `search()`**:**

Usage:

```
PubMedConnector$search(
  query,
  max_results = 100,
  date_range = NULL,
  retmode = "xml"
)
```

Arguments:

`query` PubMed search query string

`max_results` Maximum number of results to retrieve (default: 100)

`date_range` Optional date range as c("YYYY/MM/DD", "YYYY/MM/DD")

`retmode` Return mode ("xml" or "text")

Returns: Number of results found Get detailed information for specific PMIDs

Method `get_details()`**:**

Usage:

```
PubMedConnector$get_details(pmids, retmode = "xml")
```

Arguments:

`pmids` Vector of PubMed IDs

`retmode` Return mode ("xml" or "text")

Returns: Detailed article information Format results for SearchAnalyzer

Method `format_for_analysis()`**:**

Usage:

```
PubMedConnector$format_for_analysis()
```

Returns: Data frame formatted for searchAnalyzeR analysis Get search summary

Method `get_search_summary()`**:**

Usage:

```
PubMedConnector$get_search_summary()
```

Returns: List with search summary information

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PubMedConnector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create PubMed connector
pubmed <- PubMedConnector$new()

# Search for diabetes studies
results <- pubmed$search(
  query = "diabetes[Title/Abstract] AND clinical trial[Publication Type]",
  max_results = 100,
  date_range = c("2020/01/01", "2023/12/31")
)

# Format for analysis
search_data <- pubmed$format_for_analysis()
```

```
# Use with SearchAnalyzer
analyzer <- SearchAnalyzer$new(search_data)
metrics <- analyzer$calculate_metrics()
```

`rename_columns` *Rename Columns Based on Mapping*

Description

Rename Columns Based on Mapping

Usage

```
rename_columns(df, mapping)
```

Arguments

`df`	Data frame to rename
`mapping`	Named vector of column mappings

Value

Data frame with renamed columns

`ReproducibilityManager` *Search Reproducibility Manager*

Description

A comprehensive system for managing and validating the reproducibility of systematic review search strategies and analyses.

Details

The ReproducibilityManager class provides tools for:

- Creating reproducible search packages
- Validating reproducibility of existing packages
- Generating audit trails
- Ensuring transparency and reproducibility in evidence synthesis

Methods

`new()` Initialize a new ReproducibilityManager instance

`create_repro_package(search_strategy, results, analysis_config)` Create reproducible search package

`validate_repro(package_path)` Validate reproducibility of existing package

`gen_audit_trail(search_analysis)` Generate audit trail

Methods

Public methods:

- `ReproducibilityManager$new()`
- `ReproducibilityManager$create_repro_package()`
- `ReproducibilityManager$validate_repro()`
- `ReproducibilityManager$gen_audit_trail()`
- `ReproducibilityManager$clone()`

Method `new()`**:** Creates a new ReproducibilityManager instance for managing search reproducibility. Sets up necessary configurations and validates system requirements.

Usage:

```
ReproducibilityManager$new()
```

Returns: No return value, called for side effects (initialization) Create reproducible search package

Method `create_repro_package()`**:**

Usage:

```
ReproducibilityManager$create_repro_package(
  search_strategy,
  results,
  analysis_config
)
```

Arguments:

`search_strategy` Search strategy object

`results` Search results

`analysis_config` Analysis configuration

Returns: Path to reproducibility package Validate reproducibility of existing package

Method `validate_repro()`**:**

Usage:

```
ReproducibilityManager$validate_repro(package_path)
```

Arguments:

`package_path` Path to reproducibility package

Returns: Validation results Generate audit trail

Method `gen_audit_trail()`**:**

Usage:

```
ReproducibilityManager$gen_audit_trail(search_analysis)
```

Arguments:

`search_analysis` SearchAnalyzer object

Returns: Audit trail object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReproducibilityManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create reproducibility manager
manager <- ReproducibilityManager$new()

# Create sample search strategy
search_strategy <- list(
  terms = c("systematic review", "meta-analysis"),
  databases = c("PubMed", "Embase"),
  timestamp = Sys.time(),
  date_range = as.Date(c("2020-01-01", "2023-12-31"))
)

# Create sample search results
search_results <- data.frame(
  id = paste0("article_", 1:20),
  title = paste("Research Study", 1:20),
  abstract = paste("Abstract for study", 1:20),
  source = "Journal of Research",
  date = Sys.Date() - sample(1:365, 20, replace = TRUE),
  stringsAsFactors = FALSE
)

# Create sample analysis configuration
analysis_config <- list(
  gold_standard = paste0("article_", sample(1:20, 5)),
  method = "precision_recall",
  parameters = list(threshold = 0.8)
)

# Create reproducible package (writes to tempdir())
package_path <- manager$create_repro_package(
  search_strategy = search_strategy,
  results = search_results,
  analysis_config = analysis_config
)
```

```
print(paste("Package created at:", package_path))

# Generate audit trail (create mock analyzer object for demonstration)
mock_analysis <- list(
  search_results = search_results,
  metadata = list(timestamp = Sys.time())
)
class(mock_analysis) <- "mock_analyzer"

audit_trail <- manager$gen_audit_trail(mock_analysis)
print("Audit trail generated successfully")
```

`run_benchmarks` *Benchmark Suite Execution*

Description

Benchmark Suite Execution

Usage

```
run_benchmarks(
  search_strategies,
  benchmark_datasets,
  metrics_to_calculate = c("precision", "recall", "f1", "efficiency")
)
```

Arguments

`search_strategies`	List of search strategy objects
`benchmark_datasets`	List of benchmark datasets
`metrics_to_calculate`	Vector of metrics to calculate

Value

Comprehensive benchmark results

safe_divide *Safe Division Function*

Description

Safe Division Function

Usage

```
safe_divide(numerator, denominator, default_value = 0)
```

Arguments

numerator	Numerator value
denominator	Denominator value
default_value	Value to return if denominator is 0

Value

Division result or default value

safe_list_to_df *Convert List to Data Frame Safely*

Description

Convert List to Data Frame Safely

Usage

```
safe_list_to_df(x)
```

Arguments

x	List to convert

Value

Data frame or NULL if conversion fails

`SearchAnalyzer` *Search Strategy Analytics Engine*

Description

The SearchAnalyzer class provides a comprehensive framework for analyzing the performance of systematic review search strategies. It calculates precision, recall, and other performance metrics, generates visualizations, and supports validation against gold standard datasets.

Details

Core class for analyzing systematic review search strategies

This R6 class encapsulates all functionality needed for search strategy analysis. Key capabilities include:

- Performance metric calculation (precision, recall, F1, efficiency)
- Temporal and database coverage analysis
- Visualization generation for reports
- Gold standard validation

Methods

`new(search_results, gold_standard, search_strategy)` Initialize analyzer

`calculate_metrics()` Calculate comprehensive performance metrics

`visualize_performance(type)` Generate performance visualizations

Public fields

`search_results` Data frame containing search results

`gold_standard` Reference set of relevant articles

`metadata` Search strategy metadata

Methods

Public methods:

- `SearchAnalyzer$new()`
- `SearchAnalyzer$calculate_metrics()`
- `SearchAnalyzer$visualize_performance()`
- `SearchAnalyzer$clone()`

Method `new()`**:** Initialize the analyzer with search results and optional gold standard.

Usage:

```
SearchAnalyzer$new(
  search_results,
  gold_standard = NULL,
  search_strategy = NULL
)
```

Arguments:

`search_results` Data frame with search results

`gold_standard` Vector of known relevant article IDs

`search_strategy` List containing search parameters

Returns: No return value, called for side effects Calculate comprehensive performance metrics

Method `calculate_metrics()`**:**

Usage:

```
SearchAnalyzer$calculate_metrics()
```

Returns: List of performance metrics Generate performance visualization

Method `visualize_performance()`**:**

Usage:

```
SearchAnalyzer$visualize_performance(type = "overview")
```

Arguments:

`type` Type of visualization

Returns: ggplot object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SearchAnalyzer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`search_multiple_databases` *Search Multiple Databases*

Description

Search multiple databases and combine results for comprehensive analysis.

Usage

```
search_multiple_databases(
  search_strategy,
  databases = c("pubmed"),
  max_results_per_db = 100
)
```

Arguments

search_strategy	List containing search parameters
databases	Vector of databases to search ("pubmed", "pmc", etc.)
max_results_per_db	Maximum results per database

Value

Combined search results from all databases

Examples

```
# Define search strategy
strategy <- list(
  terms = "diabetes AND treatment",
  date_range = c("2020/01/01", "2023/12/31"),
  max_results = 50
)

# Search multiple databases
results <- search_multiple_databases(
  search_strategy = strategy,
  databases = c("pubmed"),
  max_results_per_db = 100
)

# Analyze results
analyzer <- SearchAnalyzer$new(results)
metrics <- analyzer$calculate_metrics()
```

---

search_pubmed *Search PubMed and Retrieve Articles*

Description

Searches PubMed using the provided search terms and retrieves article metadata in a format compatible with searchAnalyzeR analysis functions.

Usage

```
search_pubmed(
  search_terms,
  max_results = 200,
  date_range = NULL,
  language = "English"
)
```

Arguments

search_terms	Character vector of search terms to use in PubMed query
max_results	Maximum number of results to retrieve (default: 200)
date_range	Optional date range as c("YYYY-MM-DD", "YYYY-MM-DD")
language	Optional language filter (default: "English")

Details

This function connects to PubMed using the rentrez package (if available) or provides simulated data if the package is not installed. Results are returned as a standardized data frame ready for use with SearchAnalyzer.

Value

Data frame containing standardized search results

Examples

```
# Search for diabetes clinical trials
results <- search_pubmed(
  search_terms = c("diabetes", "clinical trial"),
  max_results = 100,
  date_range = c("2020-01-01", "2023-12-31")
)

# Use with SearchAnalyzer
analyzer <- SearchAnalyzer$new(results)
metrics <- analyzer$calculate_metrics()
```

---

simulate_search_execution

Simulate Search Strategy Execution

Description

Simulate Search Strategy Execution

Usage

```
simulate_search_execution(strategy, corpus)
```

Arguments

strategy	Search strategy object
corpus	Data frame with article corpus

Value

Vector of retrieved article IDs

---

standardize_date *Standardize Date Formats*

Description

Standardize Date Formats

Usage

```
standardize_date(dates)
```

Arguments

dates	Character or Date vector

Value

Date vector

---

std_cochrane_results *Standardize Cochrane Results*

Description

Standardize Cochrane Results

Usage

```
std_cochrane_results(results)
```

Arguments

results	Data frame with Cochrane results

Value

Standardized data frame

std_embase_results *Standardize Embase Results*

Description

Standardize Embase Results

Usage

```
std_embase_results(results)
```

Arguments

results	Data frame with Embase results

Value

Standardized data frame

std_generic_results *Standardize Generic Results*

Description

Standardize Generic Results

Usage

```
std_generic_results(results)
```

Arguments

results	Data frame with generic results

Value

Standardized data frame

std_pubmed_results *Standardize PubMed Results*

Description

Standardize PubMed Results

Usage

```
std_pubmed_results(results)
```

Arguments

results	Data frame with PubMed results

Value

Standardized data frame

std_scopus_results *Standardize Scopus Results*

Description

Standardize Scopus Results

Usage

```
std_scopus_results(results)
```

Arguments

results	Data frame with Scopus results

Value

Standardized data frame

std_search_results *Standardize Search Results Format*

Description

Standardize Search Results Format

Usage

```
std_search_results(results, source_format = "generic")
```

Arguments

results	Data frame with search results
source_format	Character indicating the source format

Value

Standardized data frame

std_wos_results *Standardize Web of Science Results*

Description

Standardize Web of Science Results

Usage

```
std_wos_results(results)
```

Arguments

results	Data frame with Web of Science results

Value

Standardized data frame

stream_file *Stream Process Large Files*

Description

Stream Process Large Files

Usage

```
stream_file(
  file_path,
  process_fn,
  chunk_size = 10000,
  skip = 0,
  max_lines = NULL,
  progress = TRUE
)
```

Arguments

file_path	Path to the file to process
process_fn	Function to process each chunk/line
chunk_size	Number of lines to read at once
skip	Number of lines to skip at beginning of file
max_lines	Maximum number of lines to process (NULL = all)
progress	Logical, whether to show progress

Value

Result of processing

term_effectiveness *Analyze Term Effectiveness in Search Results*

Description

Analyzes the effectiveness of individual search terms by calculating precision, coverage, and other relevant metrics for each term. This provides insight into which terms are most effective at retrieving relevant articles.

Usage

```
term_effectiveness(
  terms,
  search_results,
  gold_standard = NULL,
  text_fields = c("title", "abstract")
)
```

Arguments

terms	Character vector of search terms to analyze
search_results	Data frame with search results
gold_standard	Optional vector of relevant article IDs
text_fields	Character vector of column names to search for terms (default: c("title", "abstract"))

Details

For each term, this function calculates:

- Number of articles containing the term
- Number of relevant articles containing the term (if gold_standard provided)
- Precision (proportion of retrieved articles that are relevant)
- Coverage (proportion of relevant articles retrieved by the term)

Value

Data frame with term effectiveness metrics

Examples

```
# Create sample data
search_results <- data.frame(
  id = paste0("art", 1:10),
  title = c("Diabetes treatment", "Clinical trial", "Diabetes study",
            "Treatment options", "New therapy", "Glucose control",
            "Insulin therapy", "Management of diabetes", "Clinical study",
            "Therapy comparison"),
  abstract = c("This study examines diabetes treatments.",
               "A clinical trial on new treatments.",
               "Diabetes research findings.",
               "Comparison of treatment options.",
               "Novel therapy approach.",
               "Methods to control glucose levels.",
               "Insulin therapy effectiveness.",
               "Managing diabetes effectively.",
               "Clinical research protocols.",
               "Comparing therapy approaches.")
)
```

```
# Define search terms
terms <- c("diabetes", "treatment", "clinical", "therapy")

# Define gold standard (relevant articles)
gold_standard <- c("art1", "art3", "art7", "art8")

# Analyze term effectiveness
term_metrics <- term_effectiveness(terms, search_results, gold_standard)
print(term_metrics)
```

---

validate_date_range *Validate Date Range*

Description

Validate Date Range

Usage

```
validate_date_range(date_range, allow_future = TRUE)
```

Arguments

date_range	Date vector of length 2
allow_future	Logical, whether future dates are allowed

Value

Logical indicating if valid

---

validate_strategy *Validate Search Strategy Object*

Description

Validate Search Strategy Object

Usage

```
validate_strategy(search_strategy)
```

Arguments

search_strategy	Search strategy object to validate

Value

Logical indicating if valid, with warnings for issues

Index