Package 'contentanalysis'

March 7, 2026

Title Scientific Content and Citation Analysis from PDF Documents

Version 1.0.0

Description Provides comprehensive tools for extracting and analyzing scientific content from PDF documents, including citation extraction, reference matching, text analysis, and bibliometric indicators. Supports multi-column PDF layouts, 'CrossRef' API <https://www.crossref.org/documentation/retrieve-metadata/rest-api/> integration, and advanced citation parsing.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports base64enc (>= 0.1-3), dplyr (>= 1.1.0), httr2 (>= 0.2.0), igraph, jsonlite (>= 2.0.0), magrittr (>= 2.0.4), openalexR (>= 2.0.2), pdftools (>= 3.6.0), purrr (>= 1.1.0), stringr (>= 1.5.2), tibble (>= 3.3.0), tidyr (>= 1.3.0), tidytext (>= 0.4.3), visNetwork (>= 2.1.4)

Suggests knitr, plotly, RColorBrewer, rmarkdown, scales, stringdist, testthat (>= 3.0.0), mockery

URL https://github.com/massimoaria/contentanalysis,

BugReports https://github.com/massimoaria/contentanalysis/issues

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Massimo Aria [cre, aut, cph] (ORCID: <https://orcid.org/0000-0002-8517-9411>), Corrado Cuccurullo [aut] (ORCID: <https://orcid.org/0000-0002-7401-8575>)

Maintainer Massimo Aria <aria@unina.it>

Repository CRAN

Date/Publication 2026-03-07 13:20:02 UTC

Contents

`analyze_scientific_content` *Enhanced scientific content analysis with citation extraction*

Description

Comprehensive analysis of scientific documents including citation extraction, reference matching, text analysis, and bibliometric indicators.

Usage

```
analyze_scientific_content(
  text,
  doi = NULL,
  mailto = NULL,
  openalex_key = NULL,
 citation_type = c("all", "numeric_superscript", "numeric_bracketed", "author_year"),
  window_size = 10,
  min_word_length = 3,
  remove_stopwords = TRUE,
  language = "en",
  custom_stopwords = NULL,
  ngram_range = c(1, 3),
  parse_multiple_citations = TRUE,
  use_sections_for_citations = "auto",
  n_segments_citations = 10
)
```

Arguments

`text`	Character string or named list. Document text or text with sections.
`doi`	Character string or NULL. DOI for CrossRef reference retrieval.
`mailto`	Character string or NULL. Email for CrossRef API.
`openalex_key`	Character string or NULL. OpenAlex API key. If NULL (default), the function looks for the `OPENALEX_API_KEY` environment variable. Get a free key at `https://openalex.org/`.
`citation_type`	Character string. Type of citations to extract: • "all": Extract all citation types (default) • "numeric_superscript": Only numeric citations (brackets and superscript) + narrative • "numeric_bracketed": Only bracketed numeric citations + narrative • "author_year": Only author-year citations + narrative
`window_size`	Integer. Words before/after citations for context (default: 10).
`min_word_length`	Integer. Minimum word length for analysis (default: 3).
`remove_stopwords`	Logical. Remove stopwords (default: TRUE).
`language`	Character. Language for stopwords (default: "en").
`custom_stopwords`	Character vector. Additional stopwords.
`ngram_range`	Integer vector. N-gram range, e.g. c(1,3) (default: c(1,3)).
`parse_multiple_citations`	Logical. Parse complex citations (default: TRUE).
`use_sections_for_citations`	Logical or "auto". Use sections for mapping (default: "auto").
`n_segments_citations`	Integer. Segments if not using sections (default: 10).

Details

This function performs:

- Citation extraction (numbered, author-year, narrative, parenthetical)
- Reference parsing (from text or CrossRef API)

- Citation-reference matching
- Text analysis (word frequencies, n-grams)
- Citation context extraction
- Bibliometric indicators

The citation_type parameter filters which citation patterns to search for, reducing false positives. Narrative citations are always included as they are context-dependent.

Value

List with class "enhanced_scientific_content_analysis" containing:

- text_analytics: Basic statistics and word frequencies
- citations: All extracted citations with metadata
- citation_contexts: Citations with surrounding text
- citation_metrics: Citation type distribution, density, etc.
- citation_references_mapping: Matched citations to references
- parsed_references: Structured reference list
- word_frequencies: Word frequency table
- ngrams: N-gram frequency tables
- network_data: Citation co-occurrence data
- summary: Overall analysis summary

Examples

```
## Not run:
# For documents with numeric citations
doc <- pdf2txt_auto("paper.pdf", citation_type = "numeric_bracketed")
analysis <- analyze_scientific_content(
  doc,
  citation_type = "numeric_bracketed",
  doi = "10.xxxx/xxxxx",
  mailto = "your@email.com"
)

# For documents with author-year citations
doc <- pdf2txt_auto("paper.pdf", citation_type = "author_year")
analysis <- analyze_scientific_content(
  doc,
  citation_type = "author_year"
)

summary(analysis)
head(analysis$citations)
table(analysis$citation_metrics$type_distribution)

## End(Not run)
```

`calculate_readability_indices` *Calculate readability indices for text*

Description

Calculates multiple readability indices including Flesch-Kincaid Grade Level, Flesch Reading Ease, Automated Readability Index (ARI), and Gunning Fog Index.

Usage

```
calculate_readability_indices(text, detailed = FALSE)
```

Arguments

`text`	Character vector containing the text to analyze
`detailed`	Logical, if TRUE returns detailed statistics along with indices

Details

Formulas:

Flesch-Kincaid Grade Level:

$$0.39 \times \frac{words}{sentences} + 11.8 \times \frac{syllables}{words} - 15.59$$

Flesch Reading Ease:

$$206.835 - 1.015 \times \frac{words}{sentences} - 84.6 \times \frac{syllables}{words}$$

Automated Readability Index (ARI):

$$4.71 \times \frac{characters}{words} + 0.5 \times \frac{words}{sentences} - 21.43$$

Gunning Fog Index:

$$0.4 \times (\frac{words}{sentences} + 100 \times \frac{complex\_words}{words})$$

where complex words are those with 3 or more syllables.

Value

A tibble with the following columns:

- flesch_kincaid_grade: US grade level required to understand the text
- flesch_reading_ease: Score from 0-100 (higher = easier to read)
- automated_readability_index: ARI grade level

- gunning_fog_index: Grade level based on sentence length and complex words

If detailed = TRUE, also includes:

- n_sentences: Number of sentences
- n_words: Number of words
- n_syllables: Total syllables
- n_characters: Total characters
- n_complex_words: Words with 3+ syllables
- avg_sentence_length: Average words per sentence
- avg_syllables_per_word: Average syllables per word
- pct_complex_words: Percentage of complex words

Examples

```
## Not run:
# Simple text
text <- "The cat sat on the mat. It was a sunny day."
readability <- calculate_readability_indices(text)

# With detailed statistics
text2 <- "Reading is fun. Books open new worlds. They teach us many things."
readability_detailed <- calculate_readability_indices(text2, detailed = TRUE)

## End(Not run)
```

calculate_word_distribution

Calculate word distribution across text segments or sections

Description

Calculates the frequency of selected words/n-grams across document sections or equal-length segments.

Usage

```
calculate_word_distribution(
  text,
  selected_words,
  use_sections = "auto",
  n_segments = 10,
  remove_stopwords = TRUE,
  language = "en"
)
```

Arguments

`text`	Character string or named list. Document text or text with sections.
`selected_words`	Character vector. Words/n-grams to track.
`use_sections`	Logical or "auto". Use document sections if available (default: "auto").
`n_segments`	Integer. Number of segments if not using sections (default: 10).
`remove_stopwords`	Logical. Remove stopwords before analysis (default: TRUE).
`language`	Character. Language for stopwords (default: "en").

Details

The function:

- Automatically detects if sections are available
- Removes stopwords before creating n-grams (if requested)
- Supports unigrams, bigrams, trigrams, etc.
- Calculates both absolute and relative frequencies

Value

Tibble with columns:

- segment_id: Segment identifier
- segment_name: Section name or segment number
- segment_type: "section" or "equal_length"
- word: Word/n-gram
- count: Absolute frequency
- total_words: Total words in segment
- relative_frequency: Proportion of total words
- percentage: Percentage representation

Attributes include metadata about segmentation used.

Examples

```
## Not run:
doc <- pdf2txt_auto("paper.pdf")

# Track specific words across sections
words_to_track <- c("machine learning", "neural network", "accuracy")
dist <- calculate_word_distribution(doc, words_to_track)

# Use equal-length segments instead
dist <- calculate_word_distribution(doc, words_to_track,
                                    use_sections = FALSE,
                                    n_segments = 20)

## End(Not run)
```

`create_citation_network`

Create Citation Co-occurrence Network

Description

Creates an interactive network visualization of citation co-occurrences within a document. Citations that appear close to each other are connected, with the strength of the connection based on their distance (in characters). Nodes are colored by the document section where citations primarily appear.

Usage

```
create_citation_network(
  citation_analysis_results,
  max_distance = 1000,
  min_connections = 1,
  show_labels = TRUE
)
```

Arguments

`citation_analysis_results`
: A list object returned by citation analysis functions, containing at least two elements:
  - `network_data`: A data frame with columns `citation1`, `citation2`, and `distance` representing pairs of co-occurring citations
  - `citations`: A data frame with columns `citation_text_clean` and `section` containing citation text and section information
  - `section_colors`: A named vector of colors for each section

`max_distance`
: Numeric. Maximum distance (in characters) between citations to be considered connected. Default is 1000.

`min_connections`
: Integer. Minimum number of connections a citation must have to be included in the network. Default is 1.

`show_labels`
: Logical. Whether to show citation labels on the network nodes. Default is TRUE.

Details

The function creates a network where:

- **Nodes** represent unique citations
- **Node size** is proportional to the number of connections
- **Node color** indicates the primary section where the citation appears
- **Node border** is thicker (3px) for citations appearing in multiple sections

- **Edges** connect citations that co-occur within the specified distance
- **Edge width** decreases with distance (closer citations = thicker edges)
- **Edge color** indicates distance: red (<=300 chars), blue (<=600 chars), gray (>600 chars)

The network uses the Fruchterman-Reingold layout algorithm for optimal node positioning. Interactive features include zooming, panning, node dragging, and highlighting of nearest neighbors on hover.

Value

A `visNetwork` object representing the interactive citation network, or NULL if no valid network can be created. The returned object has an additional `stats` attribute containing:

- `n_nodes`: Number of nodes in the network
- `n_edges`: Number of edges in the network
- `avg_distance`: Average distance between connected citations
- `max_distance`: Maximum distance parameter used
- `section_distribution`: Distribution of citations across sections
- `multi_section_citations`: Citations appearing in multiple sections
- `section_colors`: Color mapping for sections

Examples

```
## Not run:
# Assuming you have citation_analysis_results from a previous analysis
network <- create_citation_network(
  citation_analysis_results,
  max_distance = 800,
  min_connections = 2,
  show_labels = TRUE
)

# Display the network
network

# Access network statistics
stats <- attr(network, "stats")
print(stats$n_nodes)
print(stats$section_distribution)

## End(Not run)
```

`describe_citation_clusters` *Describe Citation Clusters by Section Using Reference Title N-grams*

Description

Generates textual descriptions of citation clusters grouped by document section. For each section, the function extracts the most representative unigrams and bigrams from the titles of cited references, ranked by TF-IDF score. This helps understand the thematic focus of each section's bibliography.

Usage

```
describe_citation_clusters(
  citation_analysis_results,
  top_n = 10,
  min_word_length = 3,
  remove_stopwords = TRUE,
  ngram_range = c(1, 2),
  custom_stopwords = NULL
)
```

Arguments

`citation_analysis_results`
: A list object returned by `analyze_scientific_content()`, containing at least:
  - `citations`: A tibble with `citation_text_clean` and `section` columns
  - `citation_references_mapping`: A tibble linking citations to references with `citation_text_clean`, `ref_full_text`, and `matched_ref_id`
  - `parsed_references`: A tibble with `ref_id` and `ref_full_text`

`top_n`
: Integer. Number of top terms to return per section. Default is 10.

`min_word_length`
: Integer. Minimum character length for tokens to be included. Default is 3.

`remove_stopwords`
: Logical. Whether to remove English stopwords. Default is TRUE.

`ngram_range`
: Integer vector of length 2. Range of n-gram sizes to extract. Default is `c(1, 2)` for unigrams and bigrams.

`custom_stopwords`
: Character vector. Additional stopwords to remove beyond the default English stopwords. Default is NULL.

Details

The function works by:

1. Joining citations with their matched references, grouped by section

2. Extracting the title portion from each reference's full text
3. Tokenizing titles into unigrams and bigrams
4. Computing TF-IDF scores where each section is treated as a document
5. Selecting the top terms per section ranked by TF-IDF

Title extraction uses the pattern after `(YYYY).` up to the next period. If no year pattern is found, text after the first period is used as fallback.

Value

A list of class `"citation_cluster_description"` with:

- `cluster_descriptions`: A tibble with columns `section`, `ngram`, `n`, `tf`, `idf`, `tf_idf`, and `ngram_size` (1 or 2)
- `cluster_summary`: A tibble with one row per section containing `section`, `n_references`, and `top_terms` (comma-separated string)
- `cluster_references`: A tibble mapping `section` to `ref_full_text`

Examples

```
## Not run:
results <- analyze_scientific_content(pdf_path)
cluster_desc <- describe_citation_clusters(results)

# View top terms per section
cluster_desc$cluster_summary

# View detailed TF-IDF scores
cluster_desc$cluster_descriptions

## End(Not run)
```

`extract_doi_from_pdf` *Extract DOI from PDF Metadata (Legacy Function)*

Description

Legacy wrapper function for backward compatibility. Use extract_pdf_metadata() for more functionality.

Usage

```
extract_doi_from_pdf(pdf_path, return_all = FALSE)
```

Arguments

`pdf_path`	Character. Path to the PDF file.
`return_all`	Logical. If TRUE, returns all DOIs found.

Value

Character string with DOI or NA_character_.

See Also

`extract_pdf_metadata`

`extract_pdf_metadata` *Extract DOI and Metadata from PDF*

Description

This function extracts the Digital Object Identifier (DOI) and other metadata from a PDF file using pdftools::pdf_info(). It searches through all metadata fields including the XMP metadata XML.

Usage

```
extract_pdf_metadata(pdf_path, fields = "doi", return_all_dois = FALSE)
```

Arguments

`pdf_path`	Character. Path to the PDF file.
`fields`	Character vector. Metadata fields to extract. Options are: "doi", "title", "authors", "journal", "year", "all". Default is "doi".
`return_all_dois`	Logical. If TRUE, returns all DOIs found; if FALSE (default), returns only the first article DOI found (excluding journal ISSNs).

Details

The function searches for DOIs in:

- All fields in the keys list (prioritizing article DOI fields)
- The XMP metadata XML field

Journal DOIs/ISSNs (containing "(ISSN)" or from journal-specific fields) are automatically filtered out to return article DOIs.

For other metadata:

- Title: extracted from Title field or dc:title in XMP metadata
- Authors: extracted from dc:creator in XMP metadata or Author/Creator fields
- Journal: extracted from Subject, prism:publicationName in XMP metadata
- Year: extracted from prism:coverDate, created/modified dates (avoiding DOI patterns)

Common DOI prefixes are automatically removed. The function uses regex pattern matching to validate DOI format and extract structured data from XMP XML.

Value

If fields = "doi" (default), returns a character string with the DOI or NA_character_ if not found. If multiple fields are requested, returns a named list with the requested metadata. If return_all_dois = TRUE, the DOI element will be a character vector.

See Also

`pdf_info`

Examples

```
## Not run:
# Extract only DOI
doi <- extract_pdf_metadata("path/to/paper.pdf")

# Extract multiple metadata fields
meta <- extract_pdf_metadata("path/to/paper.pdf",
                             fields = c("doi", "title", "journal"))

# Extract all available metadata
meta <- extract_pdf_metadata("path/to/paper.pdf", fields = "all")

## End(Not run)
```

`gemini_content_ai` *Process Content with Google Gemini AI*

Description

Send images and/or documents to Google Gemini AI for content analysis and generation. The function supports multiple file types including images (PNG, JPG, etc.) and documents (PDF, TXT, HTML, CSV, RTF).

Usage

```
gemini_content_ai(
  image = NULL,
  docs = NULL,
  prompt = "Explain these images",
  model = "2.5-flash",
  image_type = "png",
  retry_503 = 5,
  api_key = NULL,
  outputSize = "medium"
)
```

Arguments

`image`	Character vector. Path(s) to image file(s) to be processed. Default is NULL.
`docs`	Character vector. Path(s) to document file(s) to be processed. Default is NULL.
`prompt`	Character. The prompt/instruction for the AI model. Default is "Explain these images".
`model`	Character. The Gemini model version to use. Default is "2.5-flash". Options include "2.5-flash", etc.
`image_type`	Character. The image MIME type. Default is "png".
`retry_503`	Integer. Number of retry attempts for HTTP 503 errors. Default is 5.
`api_key`	Character. Google Gemini API key. If NULL, uses the GEMINI_API_KEY environment variable.
`outputSize`	Character. Controls the maximum output tokens. Options are: • "small": 8,192 tokens • "medium": 16,384 tokens (default) • "large": 32,768 tokens • "huge": 131,072 tokens

Details

The function handles various error scenarios including:

- Missing or invalid files
- Invalid API keys (HTTP 400)
- Service unavailability (HTTP 503/429) with automatic retry
- File encoding errors

Supported document types: PDF, TXT, HTML, CSV, RTF

Value

Character vector containing the AI-generated response(s), or an error message string starting with "ERROR:" if the request fails.

Examples

```
## Not run:
# Process an image
result <- gemini_content_ai(
  image = "path/to/image.png",
  prompt = "Describe this image in detail"
)

# Process a PDF document
result <- gemini_content_ai(
  docs = "path/to/document.pdf",
  prompt = "Summarize this document",
  outputSize = "large"
)

# Process multiple images and documents
result <- gemini_content_ai(
  image = c("img1.png", "img2.png"),
  docs = c("doc1.pdf", "doc2.txt"),
  prompt = "Compare these materials"
)

## End(Not run)
```

get_crossref_references

Retrieve rich metadata from the CrossRef API for a given DOI

Description

Fetches a comprehensive set of metadata for a given DOI from the CrossRef API. The function parses the JSON response to extract main article details, author information, and the full reference list.

Usage

```
get_crossref_references(doi, mailto = NULL, output = "references")
```

Arguments

`doi`	Character string. The DOI (Digital Object Identifier) of the article.
`mailto`	Character string or NULL. An email address for polite API access, as recommended by CrossRef. This can lead to better service. Default is NULL.
`output`	Character string. The desired output content. c("all","metadata","authors","references").

Details

This function accesses the CrossRef REST API to retrieve structured metadata. It handles nested information like authors and references by parsing them into tidy data frames. Providing a `mailto` address in the user-agent is a best practice for API interaction.

Value

A list containing three main elements:

- **main_metadata**: A data frame with a single row containing key metadata about the article (e.g., DOI, title, journal, year, volume, issue).
- **authors**: A data frame listing all authors with their given name, family name, ORCID, and affiliation (if available).

- **references**: A data frame containing detailed information for each reference in the article (e.g., key, DOI, title, author, year).

Returns NULL if the DOI is not found or an error occurs.

Examples

```
## Not run:
metadata <- get_crossref_references(
  "10.1007/s11192-016-1948-8",
  mailto = "your.email@example.com",
  output = 'all'
)

# View main article metadata
print(metadata$main_metadata)

# View author information
head(metadata$authors)

# View reference list
head(metadata$references)

## End(Not run)
```

`get_example_paper` *Get path to example paper*

Description

Returns the path to the example paper included in the package source (available on GitHub but not in CRAN builds).

Usage

```
get_example_paper(example = "example_paper.pdf")
```

Arguments

`example`	Character string. Name of example file (default: "example_paper.pdf")

Value

Path to example file if it exists, otherwise downloads it from GitHub

Examples

```
## Not run:
paper_path <- get_example_paper()
doc <- pdf2txt_auto(paper_path, n_columns = 2)

## End(Not run)
```

`match_citations_to_references` *Match citations to references*

Description

Matches in-text citations to entries in the reference list using author-year matching with multiple disambiguation strategies.

Usage

```
match_citations_to_references(citations_df, references_df)
```

Arguments

`citations_df`	Data frame with citation information, must include: citation_id, citation_text, citation_text_clean, citation_type
`references_df`	Data frame with parsed references from parse_references_section()

Details

Matching algorithm:

1. Filter by exact year match
2. Match first author (exact, then fuzzy)
3. Disambiguate using second author or et al. heuristics

Match confidence levels include: high (exact first author + year), high_second_author (disambiguated with second author), medium_multiple_matches, medium_fuzzy, medium_etal_heuristic (various medium confidence scenarios), no_match_year, no_match_author, no_match_missing_info (no suitable reference found).

Value

Tibble with matched citations including columns:

- citation_id: Citation identifier
- citation_text: Original citation text
- citation_text_clean: Cleaned citation text

- citation_type: Type of citation
- cite_author: Extracted first author from citation
- cite_second_author: Second author (if present)
- cite_year: Extracted year
- cite_has_etal: Logical, contains "et al."
- matched_ref_id: ID of matched reference
- ref_full_text: Full text of matched reference
- ref_authors: Authors from reference
- ref_year: Year from reference
- match_confidence: Quality of match (high, medium, low, no_match)

Examples

```
## Not run:
matched <- match_citations_to_references(citations_df, references_df)
table(matched$match_confidence)

## End(Not run)
```

---

merge_text_chunks_named

Merge Text Chunks into Named Sections

Description

Takes a list of markdown text chunks and merges them into named sections. Each section name is extracted from the markdown header (# Title).

Usage

```
merge_text_chunks_named(
  text_chunks,
  remove_tables = TRUE,
  remove_figure_captions = TRUE
)
```

Arguments

`text_chunks`	A list of character strings with markdown text from sequential PDF chunks
`remove_tables`	Logical. If TRUE, removes all table content including captions. Default is FALSE.
`remove_figure_captions`	Logical. If TRUE, removes figure captions. Default is FALSE.

Value

A named character vector where:

- Names are section titles (without the # symbol)
- Values are complete section contents (including the title line)

`normalize_references_section` *Normalize references section formatting*

Description

Normalizes the References section to ensure each reference is separated by double newlines and internal line breaks are removed.

Usage

```
normalize_references_section(text_sections)
```

Arguments

`text_sections`	Named list from split_into_sections()

Details

Detects reference start patterns (Author, Initial.) and ensures consistent formatting with \n\n separators between references.

Value

Modified text_sections list with normalized References

Examples

```
## Not run:
sections <- split_into_sections(text, "paper.pdf")
sections <- normalize_references_section(sections)

## End(Not run)
```

parse_references_section

Parse references section from text

Description

Parses a references section into individual entries with extracted metadata including authors, year, and title information.

Usage

```
parse_references_section(references_text)
```

Arguments

`references_text`	Character string. Text of references section.

Value

Tibble with columns:

- ref_id: Unique reference identifier
- ref_full_text: Complete reference text
- ref_authors: Author string
- ref_year: Publication year
- ref_first_author: First author surname
- ref_first_author_normalized: Lowercase first author
- ref_second_author: Second author surname (if present)
- ref_second_author_normalized: Lowercase second author
- n_authors: Number of authors (99 = et al.)

Examples

```
## Not run:
refs_text <- doc$References
parsed_refs <- parse_references_section(refs_text)

## End(Not run)
```

pdf2txt_auto *Import PDF with Automatic Section Detection*

Description

High-level function that imports PDF files, extracts text while handling multi-column layouts, and optionally splits content into sections. Supports AI-enhanced text extraction using Google Gemini API and includes control over citation format conversion.

Usage

```
pdf2txt_auto(
  file,
  n_columns = NULL,
  preserve_structure = TRUE,
  sections = TRUE,
  normalize_refs = TRUE,
 citation_type = c("none", "numeric_superscript", "numeric_bracketed", "author_year"),
  enable_ai_support = FALSE,
  ai_model = "2.5-flash",
  api_key = NULL
)
```

Arguments

`file`	Character. Path to the PDF file to be processed.
`n_columns`	Integer or NULL. Number of columns in the PDF layout. Default is NULL (automatic detection).
`preserve_structure`	Logical. If TRUE, preserves paragraph structure and formatting. Default is TRUE.
`sections`	Logical. If TRUE, splits the document into sections based on headers. Default is TRUE.
`normalize_refs`	Logical. If TRUE, normalizes reference formatting in the document. Default is TRUE.
`citation_type`	Character. Type of citations used in the document. Options are: • `"none"`: No citation conversion (default) • `"numeric_superscript"`: Numeric citations in superscript format, will be converted to bracket notation • `"numeric_bracketed"`: Numeric citations already in brackets • `"author_year"`: Author-year citations (e.g., Smith, 2020) This parameter helps avoid false positives in citation detection. Only specify `"numeric_superscript"` if your document uses superscript numbers for citations.

`enable_ai_support`	Logical. If TRUE, enables AI-enhanced text extraction using Google Gemini API. Default is FALSE.
`ai_model`	Character. The Gemini model version to use for AI processing. Default is "2.5-flash". See `process_large_pdf` for available models.
`api_key`	Character or NULL. Google Gemini API key. If NULL, the function attempts to read from the `GEMINI_API_KEY` environment variable.

Details

The function attempts multiple extraction methods:

1. First tries multi-column extraction with `pdf2txt_multicolumn_safe`
2. Falls back to standard `pdftools::pdf_text` if the first method fails
3. Optionally applies AI-enhanced extraction if `enable_ai_support = TRUE`

When AI support is enabled and successful, the function:

- Processes the PDF using `process_large_pdf`
- Merges text chunks and converts to appropriate format
- Preserves References/Bibliography section from standard extraction
- Returns AI-processed content with improved formatting

Citation conversion is applied based on the `citation_type` parameter to standardize reference markers throughout the document.

Value

If `sections = TRUE`, returns a named list where:

- The first element `Full_text` contains the complete document text
- Subsequent elements contain individual sections (Introduction, Methods, etc.)

If `sections = FALSE`, returns a character string with the full document text. Returns NA if extraction fails.

Note

- AI support requires a valid Google Gemini API key
- AI processing may take longer but provides better text extraction quality
- The function automatically handles hyphenation and line breaks
- Multi-column layouts are detected and processed appropriately

See Also

`pdf2txt_multicolumn_safe` for multi-column extraction, `process_large_pdf` for AI-enhanced processing, `split_into_sections` for section detection

Examples

```
## Not run:
# Basic import with automatic section detection
doc <- pdf2txt_auto("paper.pdf")

# Import with superscript citation conversion
doc <- pdf2txt_auto(
  "paper.pdf",
  citation_type = "numeric_superscript"
)

# Import with AI-enhanced extraction
doc <- pdf2txt_auto(
  "paper.pdf",
  enable_ai_support = TRUE,
  ai_model = "2.5-flash",
  api_key = Sys.getenv("GEMINI_API_KEY")
)

# Import paper with author-year citations (no conversion)
doc <- pdf2txt_auto(
  "paper.pdf",
  citation_type = "author_year"
)

# Simple text extraction without sections or citation processing
text <- pdf2txt_auto(
  "paper.pdf",
  sections = FALSE,
  citation_type = "none"
)

# Access specific sections
introduction <- doc$Introduction
methods <- doc$Methods

## End(Not run)
```

`pdf2txt_multicolumn_safe` *Extract text from multi-column PDF with structure preservation*

Description

Extracts text from PDF files handling multi-column layouts, with options for structure preservation and automatic column detection. This version includes post-processing to convert superscript citation numbers based on the specified citation type.

Usage

```
pdf2txt_multicolumn_safe(
  file,
  n_columns = NULL,
  column_threshold = NULL,
  preserve_structure = TRUE,
 citation_type = c("none", "numeric_superscript", "numeric_bracketed", "author_year")
)
```

Arguments

`file`	Character string. Path to the PDF file.
`n_columns`	Integer or NULL. Number of columns to detect. If NULL, attempts automatic detection. Default is NULL.
`column_threshold`	Numeric or NULL. X-coordinate threshold for column separation. If NULL and n_columns is NULL, calculated automatically.
`preserve_structure`	Logical. If TRUE, preserves paragraph breaks and section structure. If FALSE, returns continuous text. Default is TRUE.
`citation_type`	Character string. Type of citations in the document: • "numeric_superscript": Numeric citations in superscript (converted to dplyr::n) • "numeric_bracketed": Numeric citations already in brackets dplyr::n (no conversion) • "author_year": Author-year citations like (Smith, 2020) (no conversion) • "none": No citation conversion. Default is "none".

Details

This function uses `pdftools::pdf_data()` for precise text extraction with spatial coordinates. It handles:

- Multi-column layouts (2+ columns)
- Section detection and paragraph preservation
- Hyphenation removal
- Title and heading identification
- Superscript citation number conversion (only if citation_type = "numeric_superscript")

If `pdf_data()` fails, falls back to `pdftools::pdf_text()`.

Value

Character string with extracted text.

Examples

```
## Not run:
# Extract from 2-column paper with superscript citations
text <- pdf2txt_multicolumn_safe("paper.pdf", n_columns = 2,
                                  citation_type = "numeric_superscript")

# Extract paper with author-year citations (no conversion)
text <- pdf2txt_multicolumn_safe("paper.pdf", citation_type = "author_year")

## End(Not run)
```

`plot_citation_clusters` *Plot Citation Cluster Descriptions*

Description

Creates interactive plotly visualizations that describe citation clusters thematically, complementing the interactive network from `create_citation_network()`.

Usage

```
plot_citation_clusters(cluster_description, section_colors = NULL, top_n = 10)
```

Arguments

`cluster_description`	An object of class `"citation_cluster_description"` returned by `describe_citation_clusters()`.
`section_colors`	Optional named vector of hex colors for sections (e.g., from `citation_analysis_results$section_co` If NULL, colors are auto-generated via an internal palette.
`top_n`	Integer. Maximum number of terms per section to display. Re-slices if the input has more. Default is 10.

Details

The three plots provide complementary views of the citation clusters:

- **TF-IDF bars** show which terms are most distinctive for each section
- **Heatmap** reveals which terms are unique to a section versus shared
- **References per section** provides an overview of bibliographic density

Value

A list of class "`citation_cluster_plots`" with three plotly objects:

- `tfidf_bars`: Horizontal bar chart of top TF-IDF terms per section, with one subplot row per section sharing the x-axis
- `tfidf_heatmap`: Heatmap of terms vs sections showing TF-IDF intensity
- `references_per_section`: Bar chart of reference counts per section

Returns NULL with a warning if `cluster_description` is NULL.

Examples

```
## Not run:
results <- analyze_scientific_content(pdf_path)
cluster_desc <- describe_citation_clusters(results)
plots <- plot_citation_clusters(cluster_desc,
  section_colors = results$section_colors
)

# Display individual plots
plots$tfidf_bars
plots$tfidf_heatmap
plots$references_per_section

## End(Not run)
```

`plot_word_distribution` *Create interactive word distribution plot*

Description

Creates an interactive plotly visualization of word frequencies across document segments or sections.

Usage

```
plot_word_distribution(
  word_distribution_data,
  plot_type = "line",
  smooth = FALSE,
  show_points = TRUE,
  colors = NULL
)
```

Arguments

`word_distribution_data`	Tibble from calculate_word_distribution()
`plot_type`	Character. "line" or "area" (default: "line").
`smooth`	Logical. Apply smoothing to lines (default: FALSE).
`show_points`	Logical. Show data points on lines (default: TRUE).
`colors`	Character vector. Custom colors for words (optional).

Details

The plot shows:

- X-axis: Document sections or segments
- Y-axis: Relative frequency (percentage)
- Each word as a separate line/area
- Hover information with exact values

Value

A plotly object with interactive visualization.

Examples

```
## Not run:
dist <- calculate_word_distribution(doc, c("method", "result", "conclusion"))
plot_word_distribution(dist, plot_type = "line", show_points = TRUE)

# Area plot with custom colors
plot_word_distribution(dist, plot_type = "area",
                       colors = c("#FF6B6B", "#4ECDC4", "#45B7D1"))

## End(Not run)
```

`process_large_pdf` *Process Large PDF Documents with Google Gemini AI*

Description

Split a large PDF into chunks and process each chunk with Google Gemini AI to extract and format text content. Particularly useful for PDFs that exceed the token limit of a single API request.

Usage

```
process_large_pdf(
  pdf_path,
  api_key,
  pages_per_chunk = 4,
  model = c("2.5-flash", "2.5-flash-lite")
)
```

Arguments

`pdf_path`	Character. Path to the PDF file to be processed.
`api_key`	Character. Google Gemini API key.
`pages_per_chunk`	Integer. Number of pages to include in each chunk. Default is 4. Lower values may help with very dense documents or API rate limits.
`model`	Character. The Gemini model version to use. Options are: "2.5-flash" and "2.5-flash-lite. Default is "2.5-flash".

Details

The function performs the following steps:

1. Validates input parameters and PDF file
2. Splits the PDF into chunks based on `pages_per_chunk`
3. Processes each chunk sequentially with Gemini AI
4. Extracts text while:
   - Removing repeated headers
   - Maintaining hierarchical structure
   - Preserving reference numbers in bracket notation
   - Formatting output as markdown
   - Handling sections that span multiple chunks
5. Returns a list of extracted text, one element per chunk

The function includes comprehensive error handling for:

- Invalid or missing PDF files
- Missing or invalid API keys
- PDF processing errors
- Gemini AI service errors
- File system operations

Rate limiting: The function includes a 1-second delay between chunks to respect API rate limits.

Value

List of character vectors, one element per chunk, containing the extracted and formatted text in markdown format. Returns NULL if processing fails.

Note

- Requires `pdftools` package for PDF manipulation
- Uses temporary files that are automatically cleaned up
- Progress messages are printed for each chunk
- All warnings are captured and reported

See Also

`gemini_content_ai` for the underlying AI processing function

Examples

```
## Not run:
# Process a large PDF with default settings
result <- process_large_pdf(
  pdf_path = "large_document.pdf",
  api_key = Sys.getenv("GEMINI_API_KEY")
)

# Process with smaller chunks and specific model
result <- process_large_pdf(
  pdf_path = "very_large_document.pdf",
  api_key = Sys.getenv("GEMINI_API_KEY"),
  pages_per_chunk = 3,
  model = "2.5-flash"
)

# Combine all chunks into single text
if (!is.null(result)) {
  full_text <- paste(unlist(result), collapse = "\n\n")
}

## End(Not run)
```

`readability_multiple` *Calculate readability indices for multiple texts*

Description

Vectorized version that calculates readability indices for multiple texts.

Usage

```
readability_multiple(texts, detailed = FALSE, text_id = NULL)
```

Arguments

`texts`	Character vector containing texts to analyze
`detailed`	Logical, if TRUE returns detailed statistics along with indices
`text_id`	Optional character vector with identifiers for each text

Value

A tibble with one row per text, including text_id if provided

Examples

```
## Not run:
texts <- c("First text here.", "Second text is longer and more complex.")
ids <- c("doc1", "doc2")
readability_multiple(texts, text_id = ids, detailed = TRUE)

## End(Not run)
```

`remove_all_tables` *Remove All Types of Tables (Markdown and Plain Text)*

Description

Remove All Types of Tables (Markdown and Plain Text)

Usage

```
remove_all_tables(text)
```

Arguments

`text`	Character string containing text with tables

Value

Character string with all tables and table captions removed

`remove_code_blocks` *Remove Markdown Code Block Markers*

Description

Remove Markdown Code Block Markers

Usage

```
remove_code_blocks(text)
```

Arguments

`text`	Character string containing markdown text

Value

Character string with `markdown and ` markers removed

`remove_figure_caps` *Remove Figure Captions*

Description

Remove Figure Captions

Usage

```
remove_figure_caps(text)
```

Arguments

`text`	Character string containing markdown text

Value

Character string with figure captions removed

`split_into_sections` *Split document text into sections*

Description

Splits extracted text into logical sections (Introduction, Methods, Results, etc.) using either the PDF's table of contents or common academic section patterns.

Usage

```
split_into_sections(text, file_path = NULL)
```

Arguments

`text`	Character string. Full text of the document.
`file_path`	Character string or NULL. Path to PDF file for TOC extraction. If NULL, uses common section names. Default is NULL.

Details

The function attempts to:

1. Extract section names from PDF table of contents
2. Fall back to common academic section names if TOC unavailable
3. Match section headers in text using regex patterns
4. Handle duplicate section names

Common sections searched: Abstract, Introduction, Methods, Results, Discussion, Conclusion, References, etc.

Value

Named list where each element is a section's text. Always includes "Full_text" element with complete document.

Examples

```
## Not run:
text <- pdf2txt_auto("paper.pdf", sections = FALSE)
sections <- split_into_sections(text, file_path = "paper.pdf")
names(sections)

## End(Not run)
```

Index