Package 'jstor'

August 16, 2023

Title Read Data from JSTOR/DfR

Version 0.3.11

Description Functions and helpers to import metadata, ngrams and full-texts delivered by Data for Research by JSTOR.

Depends R (>= 3.1)

License GPL-3

Encoding UTF-8

Imports dplyr (>= 1.0.0), tidyr (>= 0.7.2), purrr (>= 0.2.4), xml2 (>= 1.2.0), magrittr, stringr (>= 1.3.0), readr (>= 2.0.0), tibble (>= 3.0.0), rlang (>= 0.2.0), furrr (>= 0.1.0), pryr, crayon, cli

Suggests testthat, covr, knitr, rmarkdown, future

BugReports https://github.com/ropensci/jstor/issues

URL https://github.com/ropensci/jstor, https://docs.ropensci.org/jstor/

RoxygenNote 7.2.2

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Thomas Klebel [aut, cre] (<https://orcid.org/0000-0002-7331-4751>)

Maintainer Thomas Klebel <t.klebel@hotmail.com>

Repository CRAN

Date/Publication 2023-08-16 13:14:39 UTC

R topics documented:

`jstor` *jstor: Read Data from JSTOR/DfR*

Description

The tool Data for Research (DfR) by JSTOR is a valuable source for citation analysis and text mining. `jstor` provides functions and suggests workflows for importing datasets from DfR.

Details

Please refer to the vignettes for information on how to use the package:

```
browseVignettes("jstor")
```

If you encounter any issues or have ideas for new features, please file an issue at https://github.com/ropensci/jstor/issues.

Author(s)

Thomas Klebel

jst_add_total_pages *Add total count of pages*

Description

This function adds a column with the total count of pages. It calls `jst_get_total_pages()` which does the main work.

Usage

```
jst_add_total_pages(meta_data, quietly = FALSE)
```

Arguments

`meta_data`	Data which was processed via `jst_get_article()`.
`quietly`	Should warnings from converting page ranges be suppressed?

Value

A `tibble`, as provided with in `meta_data`, with an additional column on total number of pages.

See Also

`jst_get_total_pages()`

jst_augment *Clean data from DfR*

Description

This function takes data from `jst_get_article()` and applies helper functions for cleaning the data.

Usage

```
jst_augment(meta_data, quietly = FALSE)
```

Arguments

`meta_data`	Data which was processed via `jst_get_article()`.
`quietly`	Should warnings from converting page ranges be suppressed?

Details

Data from DfR is inherently messy. For many examples see `vignette("known-quirks", package = "jstor")`. `jst_augment()` is a convenience function that tries to deal with a few common tasks to clean the data.

For journal articles, it calls `jst_clean_page()` to convert first and last page, `jst_unify_journal_id()` and `jst_add_total_pages()`.

Value

A cleaned tibble.

See Also

`jst_clean_page()` `jst_unify_journal_id()` `jst_add_total_pages()` `jst_get_total_pages()`

---

jst_clean_page *Clean a character vector of pages*

Description

This function tries to convert character vectors into integers. This function should not be called on page ranges.

Usage

```
jst_clean_page(page)
```

Arguments

page	A character vector for pages.

Value

An integer vector, cleaned and converted from the input vector.

Examples

```
jst_clean_page("2")

# anything that is not a digit gets removed
jst_clean_page("A2-")

# a weird format from the American Journal of Sociology is convered correctly
jst_clean_page("AJSv104p126")
# this is done by searching for "p", and if it is found, extracting the
# content after "p".
```

`jst_combine_outputs` *Combine outputs from converted files*

Description

`jst_combine_outputs()` helps you to manage the multitude of files you might receive after running `jst_import()` or `jst_import_zip()` with more than one batch.

Usage

```
jst_combine_outputs(
  path,
  write_to_file = TRUE,
  out_path = NULL,
  overwrite = FALSE,
  clean_up = FALSE,
  warn = TRUE
)
```

Arguments

`path`	A path to a directory, containing .csv-files from `jst_import()` or `jst_import_zip()`, or a vector of files which are to be imported.
`write_to_file`	Should combined data be written to a file?
`out_path`	A directory where to write the combined files. If no directory is supplied and `write_to_file` is `TRUE`, the combined files are written to `path`.
`overwrite`	Should files be overwritten?
`clean_up`	Do you want to remove the original batch files? Use with caution.
`warn`	Should warnings be raised, if the file type cannot be determined?

Details

Splitting the output of `jst_import()` or `jst_import_zip()` might be done for multiple reasons, but in the end you possibly want to combine all outputs into one file/data.frame. This function makes a few assumptions in order to combine files:

- Files with similar names (except for trailing dashes with numbers) belong together and will be combined into one file.
- The names of the combined files can be determined from the original files. If you want to combine `foo-1.csv` and `foo-2.csv`, the combined file will be `combined_foo.csv`.
- The directory only contains files which were imported via `jst_import()` or `jst_import_zip()`. If the directory contains other `.csv` files, you should supply a character vector with paths to only those files, which you want to import.

Value

Either writes to disk, or returns a list with all combined files.

See Also

jst_re_import()

Examples

```
# set up a temporary directory
tmp <- tempdir()

# find multiple files
file_list <- rep(jst_example("article_with_references.xml"), 2)

# convert and write to file
jst_import(file_list, "article", out_path = tmp, .f = jst_get_article,
             n_batches = 2, show_progress = FALSE)

# combine outputs
jst_combine_outputs(tmp)
list.files(tmp, "csv")

## Not run:
# Trying to combine the files again raises an error.
jst_combine_outputs(tmp)

## End(Not run)

# this doesn't
jst_combine_outputs(tmp, overwrite = TRUE)

# we can remove the original files too
jst_combine_outputs(tmp, overwrite = TRUE, clean_up = TRUE)
list.files(tmp, "csv")
```

---

jst_define_import *Define an import specification*

Description

Define which parts of a zip file should be converted via which functions.

Usage

```
jst_define_import(...)
```

Arguments

...	Named arguments with bare function names.

Details

The function accepts the following names: article, book, report, pamphlet, ngram1, ngram2, ngram3. The corresponding files from a .zip-archive will be imported via the supplied functions.

Value

A specification of imports which is necessary for `jst_import_zip()`.

Examples

```
# articles will be imported via `jst_get_article()` and `jst_get_authors()`
jst_define_import(article = c(jst_get_article, jst_get_authors))

# define a specification for importing article metadata and unigrams (ngram1)
jst_define_import(article = jst_get_article,
                  ngram1 = jst_get_ngram)


# import all four types with one function each
jst_define_import(article = jst_get_article,
                  book = jst_get_book,
                  report = jst_get_book,
                  pamphlet = jst_get_article)

# import all four types with multiple functions
jst_define_import(article = c(jst_get_article, jst_get_authors, jst_get_references),
                  book = c(jst_get_book, jst_get_chapters),
                  report = jst_get_book,
                  pamphlet = jst_get_article)

# if you want to import chapters with authors, you can use an anonymous
# function

chapters_w_authors <- function(x) jst_get_chapters(x, authors = TRUE)
jst_define_import(book = chapters_w_authors)


## Not run:
# define imports
imports <- jst_define_import(article = c(jst_get_article, jst_get_authors))

# convert the files to .csv
jst_import_zip("my_archive.zip", out_file = "my_out_file",
                 import_spec = imports)

## End(Not run)
```

jst_example *Get path to jstor example*

Description

jstor includes several sample files for demonstration purposes. This helper makes them easy to access.

Usage

```
jst_example(path = NULL)
```

Arguments

path	Name of the example file. If NULL, the example files will be listed.

Details

The code for this function was adapted from the package readr.

Value

Either a character vector with the names of example files (if jst_example() is called without supplying an argument), or a character vector indicating the path to the example file.

Examples

```
jst_example()
jst_example("article_with_references.xml")
```

jst_get_article *Extract meta information for articles*

Description

jst_get_article() extracts meta-data from JSTOR-XML files for journal articles.

Usage

```
jst_get_article(file_path)
```

Arguments

file_path	A .xml-file for a journal-article.

Value

A `tibble` containing the extracted meta-data with the following columns:

- file_name *(chr)*: The file_name of the original .xml-file. Can be used for joining with other parts (authors, references, footnotes, full-texts).
- journal_doi *(chr)*: A registered identifier for the journal.
- journal_jcode *(chr)*: A identifier for the journal like "amerjsoci" for the "American Journal of Sociology".
- journal_pub_id *(chr)*: Similar to journal_jcode. Most of the time either one is present.
- journal_title *(chr)*: The title of the journal.
- article_doi *(chr)*: A registered unique identifier for the article.
- article_jcode *(chr)*: A unique identifier for the article (not a DOI).
- article_pub_id *(chr)*: Infrequent, either part of the DOI or the article_jcode.
- article_type *(chr)*: The type of article (research-article, book-review, etc.).
- article_title *(chr)*: The title of the article.
- volume *(chr)*: The volume the article was published in.
- issue *(chr)*: The issue the article was published in.
- language *(chr)*: The language of the article.
- pub_day *(chr)*: Publication day, if specified.
- pub_month *(chr)*: Publication month, if specified.
- pub_year *(int)*: Year of publication.
- first_page *(int)*: Page number for the first page of the article.
- last_page *(int)*: Page number for the last page of the article.
- page_range *(chr)*: The range of pages for the article.

A note about publication dates: always the first entry is being extracted, which should correspond to the oldest date, in case there is more than one date.

Examples

```
jst_get_article(jst_example("article_with_references.xml"))
```

---

jst_get_authors *Extract author information*

Description

`jst_get_authors()` extracts information about authors from JSTOR-XML files.

Usage

```
jst_get_authors(file_path)
```

Arguments

file_path	A .xml-file from JSTOR containing meta-data.

Details

The function returns a tibble with the following six columns:

- *prefix*: in case there was a prefix to the name, like "Dr.".
- *given_name*: The author's given name, like "Albert".
- *surname*: The author's surname like "Einstein".
- *string_name*: In some cases data the name is not available in separate fields, but just as a complete string: "Albert Einstein".
- *suffix*: a suffix to the name, like "Jr.".
- *author_number*: The authors are enumerated in the order they appear in the data.

Value

A tibble containing the extracted authors. All empty fields are NA_character.

Examples

```
jst_get_authors(jst_example("article_with_references.xml"))
```

---

jst_get_book *Extract meta information for books*

Description

jst_get_book() extracts meta-data from JSTOR-XML files for book chapters.

Usage

```
jst_get_book(file_path)
```

Arguments

file_path	A .xml-file for a book or research report.

Value

A tibble containing the extracted meta-data with the following columns:

- file_name *(chr)*: The filename of the original .xml-file. Can be used for joining with other data for the same file.
- discipline *(chr)*: The discipline from the discipline names used on JSTOR.
- book_id *(chr)*: The book id of type "jstor", which is not a registered DOI.

- book_title *(chr)*: The title of the book.
- book_subtitle *(chr)*: The subtitle of the book.
- pub_day *(int)*: Publication day, if specified.
- pub_month *(int)*: Publication month, if specified.
- pub_year *(int)*: Year of publication.
- isbn *(chr)*: One or more entries for the book's ISBN. If two or more, separated by `"; "`.
- publisher_name *(chr)*: The name of the publisher.
- publisher_loc *(chr)*: The location of the publisher.
- n_pages *(int)*: The number of pages.
- language *(chr)*: The language of the book.

A note about publication dates: always the first entry is being extracted, which should correspond to the oldest date, in case there is more than one date.

Examples

```
jst_get_book(jst_example("book.xml"))
```

`jst_get_chapters` *Extract information on book chapters*

Description

`jst_get_chapters()` extracts meta-data from JSTOR-XML files for book chapters.

Usage

```
jst_get_chapters(file_path, authors = FALSE)
```

Arguments

`file_path`	The path to a `.xml`-file for a book or research report.
`authors`	Extracting the authors is an expensive operation which makes the function ~3 times slower, depending on the number of chapters and the number of authors. Defaults to `FALSE`. Use `authors = TRUE` to import the authors too.

Details

Currently, `jst_get_chapters()` is quite a lot slower than most of the other functions. It is roughly 10 times slower than `jst_get_book`, depending on the number of chapters to extract.

Value

A `tibble` containing the extracted meta-data with the following columns:

- book_id *(chr)*: The book id of type "jstor", which is not a registered DOI.
- file_name *(chr)*: The filename of the original .xml-file. Can be used for joining with other data for the same file.
- part_id *(chr)*: The id of the part.
- part_label *(chr)*: A label for the part, if specified.
- part_title *(chr)*: The title of the part.
- part_subtitle *(chr)*: The subtitle of the part, if specified.
- authors *(list)*: A list-column with information on the authors. Can be unnested with `tidyr::unnest()`. See the examples and `jst_get_authors()`.
- abstract *(chr)*: The abstract to the part.
- part_first_page *(chr)*: The page where the part begins.

Examples

```
# extract parts without authors
jst_get_chapters(jst_example("book.xml"))

# import authors too
parts <- jst_get_chapters(jst_example("book.xml"), authors = TRUE)
parts

tidyr::unnest(parts)
```

---

jst_get_file_name *Extract the basename of a path*

Description

This helper simply extracts the basename of a path and removes the extension, e.g. `foo/bar.txt` is shortened to `bar`.

Usage

```
jst_get_file_name(file_path)
```

Arguments

`file_path`	A path to a file

Value

A character vector, containing the basename of the file without an extension.

jst_get_footnotes *Extract all footnotes*

Description

This function extracts the content of fn-group from journal-articles.

Usage

```
jst_get_footnotes(file_path)
```

Arguments

file_path	The path to the .xml-file from which footnotes should be extracted.

Details

The fn-group usually contains footnotes corresponding to the article. However, since footnotes are currently not fully supported by DfR, there is no comprehensive documentation on the different variants. jstor therefore extracts the content of fn-group exactly as it appears in the data. Because of this, there might be other content present than footnotes.

In order to get all available information on citation data, you might need to combine jst_get_footnotes() with jst_get_references().

Value

A tibble containing the content from fn-group (usually the footnotes). If there were no footnotes, NA_character is returned for the column footnotes.

Examples

```
jst_get_footnotes(jst_example("article_with_footnotes.xml"))
```

jst_get_full_text *Import full-text*

Description

This function imports the full_text contents of a JSTOR-article.

Usage

```
jst_get_full_text(filename)
```

Arguments

filename	The path to the file.

Value

A `tibble`, containing the file-path as id, the full content of the file, and the encoding which was used to read it.

`jst_get_journal_overview` *Get table with information on journals*

Description

Download most recent or display cached version of data on journals.

Usage

```
jst_get_journal_overview(most_recent = FALSE, quiet = FALSE)
```

Arguments

`most_recent`	Should the most recent version be downloaded from DfR? (Currently disabled due to changes on the JSTOR-servers).
`quiet`	Should status messages about the download be printed?

Details

When analysing your sample of articles from DfR, it might be helpful to have some context about the journals in your sample. This function provides a `tibble` with various information like the full name of the journal, the short version of the name (sometimes referred to as `JCODE`), dates on where the first and last (available) issues were published, etc.

The data on journals might change. Therefore this function provides two sources of data: a cached version which gets updated with every release, and the ability to pull the most recent version directly from DfR (this had to be temporarily disabled.)

The cached version was updated on 2020-04-03.

Value

A `tibble` with various information about journals.

Examples

```
# use the function without arguments to get a tibble from disk
jst_get_journal_overview()

## Not run:
# download the most recent version from DfR
jst_get_journal_overview(most_recent = TRUE)

## End(Not run)
```

jst_get_ngram *Read ngram data*

Description

Read in data on ngrams via `readr::read_tsv()`.

Usage

```
jst_get_ngram(file)
```

Arguments

`file`	A path to a file or a zip location from `jst_subset_ngrams()`.

Details

This function is mainly useful when it is used in together with jst_import_zip, where you can use it to specify reading in ngram data.

Value

A `tibble::tibble()` with two columns:

- *ngram*: the ngram term (unigram, bigram, trigram)
- *n*: an integer for the number of times the term occurred in the original file

jst_get_references *Extract all references*

Description

This function extracts the content of `ref-list` from the `xml`-file.

Usage

```
jst_get_references(file_path, parse_refs = FALSE)
```

Arguments

`file_path`	The path to the `.xml`-file from which references should be extracted.
`parse_refs`	Should references be parsed, if available?

Details

This content may contain references or endnotes, depending on how the article used citations. Since references are currently not fully supported by DfR, there is no comprehensive documentation on the different variants. `jstor` therefore extracts the content of `ref-list` exactly as it appears in the data. Because of this, there might be other content present than references.

In order to get all available information on citation data, you might need to combine `jst_get_references()` with `jst_get_footnotes()`.

For newer `xml`-files, there would be the option to extract single elements like authors, title or date of the source, but this is not yet implemented.

In general, the implementation is not as fast as `jst_get_article()` - articles with many references slow the process down.

Value

A `tibble` with the following columns:

- `file_name`: the identifier for the article the references come from.
- `ref_title`: the title of the references sections.
- `ref_authors`: a string of authors. Several authors are separated with `;`.
- `ref_editors`: a string of editors, if available.
- `ref_collab`: a field that may contain information on the authors, if authors are not available.
- `ref_item_title`: the title of the cited entry. For books this is often empty, with the title being in `ref_source`.
- `ref_year`: a year, often the article's publication year, but not always.
- `ref_source`: the source of the cited entry. For books often the title of the book, for articles the publisher of the journal.
- `ref_volume`: the volume of the journal article.
- `ref_first_page`: the first page of the article/chapter.
- `ref_last_page`: the last page of the article/chapter.
- `ref_publisher`: For books the publisher, for articles often missing.
- `ref_publication_type`: Known types: `book`, `journal`, `web`, `other`.
- `ref_unparsed`: The full references entry in unparsed form.

Examples

```
jst_get_references(jst_example("article_with_references.xml"))

# import parsed references
jst_get_references(
  jst_example("parsed_references.xml"),
  parse_refs = TRUE
)
```

`jst_get_total_pages` *Calculate total pages*

Description

This function is a simple helper to calculate the total number of pages of an article.

Usage

```
jst_get_total_pages(first_page, last_page, page_range, quietly = FALSE)
```

Arguments

`first_page`	The first page of an article (numeric).
`last_page`	The last page of an article (numeric).
`page_range`	The page range of an article (character).
`quietly`	Sometimes page ranges contain roman numerals like `xiv`. These are not recognized, return `NA` and raise a warning. If set to `TRUE`, this warning not raised.

Details

This function deals with four cases:

- if all three arguments are missing, NA is returned.
- if page_range is supplied, the number of pages is calculated from it.
- if only the first page is supplied, NA is returned.
- if first and last page are supplied, the number of pages is calculated as `last_page - first_page + 1`.

The algorithm to parse page ranges works as follows: A typical page range is `1-10, 200` where the article starts at page 1, ends at page 10, and has an erratum at page 200. For this case, the range is calculated as `range + single_page`, as in `(10 - 1 + 1) + 1 = 11`. Sometimes multiple ranges are given: `1-10, 11-20`. For those cases all ranges are summed: `(10 - 1 + 1) + (20 - 11 + 1) = 20`. Another specification for multiple ranges is `1-10+11-20`, which is treated similarly.

Value

A vector with the calculated total pages.

Examples

```
# calculate pages from first and last page
first_pages <- sample(30:50, 10)
last_pages <- first_pages + sample(5:20, 10)
page_ranges <- rep(NA_character_, 10)

jst_get_total_pages(first_pages, last_pages, page_ranges)
```

```
# get pages from page range
jst_get_total_pages(NA_real_, NA_real_, "51 - 70")
jst_get_total_pages(NA_real_, NA_real_, "51 - 70, 350")
jst_get_total_pages(NA_real_, NA_real_, "350, 51 - 70")
jst_get_total_pages(NA_real_, NA_real_, "51 - 70, 80-100")
jst_get_total_pages(NA_real_, NA_real_, "51-70+350")
```

`jst_import` *Wrapper for file import*

Description

This function applies an import function to a list of `xml`-files or a .zip-archive in case of `jst_import_zip` and saves the output in batches of `.csv`-files to disk.

Usage

```
jst_import(
  in_paths,
  out_file,
  out_path = NULL,
  .f,
  col_names = TRUE,
  n_batches = NULL,
  files_per_batch = NULL,
  show_progress = TRUE
)

jst_import_zip(
  zip_archive,
  import_spec,
  out_file,
  out_path = NULL,
  col_names = TRUE,
  n_batches = NULL,
  files_per_batch = NULL,
  show_progress = TRUE,
  rows = NULL
)
```

Arguments

`in_paths`	A character vector to the `xml`-files which should be imported
`out_file`	Name of files to export to. Each batch gets appended by an increasing number.
`out_path`	Path to export files to (combined with filename).

.f	Function to use for import. Can be one of jst_get_article, jst_get_authors, jst_get_references, jst_get_footnotes, jst_get_book or jst_get_chapter.
col_names	Should column names be written to file? Defaults to TRUE.
n_batches	Number of batches, defaults to 1.
files_per_batch	Number of files for each batch. Can be used instead of n_batches, but not in conjunction.
show_progress	Displays a progress bar for each batch, if the session is interactive.
zip_archive	A path to a .zip-archive from DfR
import_spec	A specification from jst_define_import for which parts of a .zip-archive should be imported via which functions.
rows	Mainly used for testing, to decrease the number of files which are imported (i.e. 1:100).

Details

Along the way, we wrap three functions, which make the process of converting many files easier:

- purrr::safely()
- furrr::future_map()
- readr::write_csv()

When using one of the find_* functions, there should usually be no errors. To avoid the whole computation to fail in the unlikely event that an error occurs, we use safely() which let's us continue the process, and catch the error along the way.

If you have many files to import, you might benefit from executing the function in parallel. We use futures for this to give you maximum flexibility. By default the code is executed sequentially. If you want to run it in parallel, simply call future::plan() with future::multisession() as an argument before running jst_import or jst_import_zip.

After importing all files, they are written to disk with readr::write_csv().

Since you might run out of memory when importing a large quantity of files, you can split up the files to import into batches. Each batch is being treated separately, therefore for each batch multiple processes from future::multisession() are spawned, if you added this plan. For this reason, it is not recommended to have very small batches, as there is an overhead for starting and ending the processes. On the other hand, the batches should not be too large, to not exceed memory limitations. A value of 10000 to 20000 for files_per_batch should work fine on most machines. If the session is interactive and show_progress is TRUE, a progress bar is displayed for each batch.

Value

Writes .csv-files to disk.

See Also

jst_combine_outputs()

Examples

```
## Not run:
# read from file list --------
# find all files
meta_files <- list.files(pattern = "xml", full.names = TRUE)

# import them via `jst_get_article`
jst_import(meta_files, out_file = "imported_metadata", .f = jst_get_article,
           files_per_batch = 25000)

# do the same, but in parallel
library(future)
plan(multiprocess)
jst_import(meta_files, out_file = "imported_metadata", .f = jst_get_article,
           files_per_batch = 25000)

# read from zip archive ------
# define imports
imports <- jst_define_import(article = c(jst_get_article, jst_get_authors))

# convert the files to .csv
jst_import_zip("my_archive.zip", out_file = "my_out_file",
                 import_spec = imports)

## End(Not run)
```

`jst_preview_zip` *Preview content of zip files*

Description

This function gives you a quick preview about what a .zip-file from DfR contains.

Usage

```
jst_preview_zip(zip_archive)
```

Arguments

`zip_archive`	A path to a .zip-file from DfR

Value

The function returns a tibble with three columns:

- *type*: metadata or some form of ngram
- *meta_type*: which type of metadata (book_chapter, journal article, ...)
- *n*: a count for each category

Examples

```
jst_preview_zip(jst_example("pseudo_dfr.zip"))
```

`jst_re_import` *Re-import files*

Description

`jst_re_import()` lets you re-import a file which was exported via `jst_import()` or `jst_import_zip()`.

Usage

```
jst_re_import(file, warn = TRUE)
```

Arguments

`file`	A path to a .csv file.
`warn`	Should warnings be emitted, if the type of file cannot be determined?

Details

When attempting to re-import, a heuristic is applied. If the file has column names which match the names from any of the `find_*` functions, the file is read with the corresponding specifications. If no column names are recognized, files are recognized based on the number of columns. Since both references and footnotes have only two columns, the first line is inspected for either `"Referenc...|Bilbio...|Endnote..."` or `"Footnote..."`. In case there is still no match, the file is read with `readr::read_csv()` with `guess_max = 5000` and a warning is raised.

Value

A `tibble`, with the columns determined based on heuristics applied to the input file.

See Also

`jst_combine_outputs()`

jst_subset_ngrams *Define a subset of ngrams*

Description

This function helps in defining a subset of ngram files which should be imported, since importing all ngrams at once can be very expensive (in terms of cpu and memory).

Usage

```
jst_subset_ngrams(zip_archives, ngram_type, selection, by = file_name)
```

Arguments

zip_archives	A character vector of one or multiple zip-files.
ngram_type	One of "ngram1", "ngram2" or "ngram3"
selection	A data.frame with the articles/books which are to be selected.
by	A column name for matching.

Value

A list of zip-locations which can be read via jst_get_ngram().

Examples

```
# create sample output
tmp <- tempdir()
jst_import_zip(jst_example("pseudo_dfr.zip"),
               import_spec = jst_define_import(book = jst_get_book),
               out_file = "test", out_path = tmp)

# re-import as our selection for which we would like to import ngrams
selection <- jst_re_import(file.path(tmp,
                                     "test_book_chapter_jst_get_book-1.csv"))

# get location of file
zip_loc <- jst_subset_ngrams(jst_example("pseudo_dfr.zip"), "ngram1",
                             selection)

# import ngram
jst_get_ngram(zip_loc[[1]])
unlink(tmp)
```

`jst_unify_journal_id` *Unify journal IDs*

Description

This function is a simple wrapper to unify journal ids.

Usage

```
jst_unify_journal_id(meta_data, remove_cols = TRUE)
```

Arguments

`meta_data`	Data which was processed via `jst_get_article()`.
`remove_cols`	Should the original columns be removed after unifying?

Details

Date on journal ids can be found in three columns: `journal_pub_id`, `journal_jcode` and `journal_doi`. From my experience, most of the time the relevant dat ais present in `journal_pub_id` or `journal_jcode`, with `journal_jcode` being to most common identifier. This function takes the value from `journal_pub_id`, and if it is missing, that from `journal_jcode`. `journal_doi` is currently disregarded.

Value

A modified `tibble`.

A modified tibble.

Examples

```
article <- jst_get_article(jst_example("article_with_references.xml"))

jst_unify_journal_id(article)


# per default, original columns with data on the journal are removed
library(dplyr)

jst_unify_journal_id(article) %>%
  select(contains("journal")) %>%
  names()

# you can keep them by setting `remove_cols` to `FALSE`
jst_unify_journal_id(article, remove_cols = FALSE) %>%
  select(contains("journal")) %>%
  names()
```

Index