Package: metaextractoR (via r-universe)

July 5, 2026

Type Package

Title Data Extraction for Meta-Analysis with Large Language Models

Version 0.1.0

Description Use Large Language Models (LLMs) to assist with data extraction for meta-analysis. This package incorporates three modular Shiny apps to implement a human-in-the-loop framework. (1) a manual extraction interface for abstracts, (2) refining prompt engineering and model selection, and (3) validation of LLM-generated outputs. These apps enable researchers to iteratively collaborate with LLMs, with each abstract undergoing double data extraction—once manually by a human researcher and once independently by an LLM-assisted process—to emulate the double extraction process recommended by international standards. Notably, the package runs fully on local machines, with no need for API setup or external data transfer, maximising data privacy and accessibility. Robust logging features further enhance transparency and reproducibility by recording all prompt iterations and outputs.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.5)

Imports ellmer, shiny, tidyr, shinyFiles, cli, dplyr, fs, methods, DT, bslib, shinyjs

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://danyangdai.github.io/metaextractoR/

BugReports https://github.com/DanyangDai/metaextractoR/issues

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev zlib1g-dev

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

Date/Publication 2026-04-07 22:22:39 UTC

RemoteUrl https://github.com/DanyangDai/metaextractoR

RemoteRef HEAD

RemoteSha 229c8547ffc83f73d418a860254f066023d9ec0d

Contents

`abstracts` *Sample abstracts*

Description

This dataset includes sample abstracts downloaded from one of my Covidence project. Pre-processing was done by running the clean_name function from janitor package.

Usage

```
abstracts

app_2

app_3

testing_stage_0_data

training_stage_0_data
```

Format

A data frame with 20 rows and 16 variables:

title Title of the manuscript

authors Authors

abstract The column that contains all abstracts

published_year Year of the publication

published_month Month of the publication

journal Journal's name

volume Volume number

issue Issue number

pages Page number

accession_number Accession number

doi doi number

ref Reference

covidence_number Covidence number

study Study

notes Notes

tags Tags

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 10 rows and 15 columns.

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 40 rows and 16 columns.

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 40 rows and 16 columns.

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 10 rows and 16 columns.

Source

Created for demonstration purposes

`add_predefined_vars` *Pre-processing functions before shinyapps*

Description

Pre-processing functions and saving intermediate data

Usage

```
add_predefined_vars(data, list_vars)
```

Arguments

`data`	a csv file contains abstract information. This could be the csv file downloaded from covidence
`list_vars`	a vector of data elements you want to extract. i.e. c("no_participants,"no_female","..")

Value

new dataset with additional empty columns

Examples

```
add_predefined_vars(abstracts,
                    c("no_participants", "no_aki", "age_mean", "age_sd"))
```

`glance_manual_app` *Launch the first shinyapp for to look into what variables are available in the abstract.*

Description

This function launches a Shiny app that displays each abstract alongside the data elements you want to extract. The app provides an interactive interface for manually entering this information, helping to fill out the _manual column generated by the ‘add_predefined_vars()‘ function in the training set.

Usage

```
glance_manual_app()
```

Value

Runs a shiny app.

Examples

```
if (interactive()) {
 glance_manual_app()
}
```

`manual_validation_app` *manual_validation_app*

Description

This Shiny app allows you to manually review the results generated by the ‘process_with_ollama()‘ function. By uploading a testing dataset with LLM-extracted values, you can systematically check and validate each field filled out by the model.

Usage

```
manual_validation_app()
```

Value

Runs a shiny app.

Examples

```
if (interactive()) {
  manual_validation_app()
}
```

`process_with_ollama` *Process the abstract with a large language model*

Description

A function that batch process data extraction from text.

Usage

```
process_with_ollama(
  input,
  model = "llama3.1:8b",
  type_abstract,
  i,
  abstract_col
)
```

Arguments

`input`	A data frame contains abstracts and variables we want to extract. If you are not sure how the data should look like, please see the example dataset abstract. This data should be the testing data saved using the 'save_training_data()' function.
`model`	Large Language Model name you want to use i.e. "llama3.1:8b". If you are not sure what LLMs have been installed locally, please use the function: ellmer:::ollama_models() to find out what LLMs are available.
`type_abstract`	is created using the type_object function from the ellmer package. Objects represent a collection of named values and are created with type_object(). Objects can contain any number of scalars, arrays, and other objects. They are similar to named lists in R. If still not sure, please look up the documentation of ellmer::type_object().
`i`	number of abstracts you want to process at once.
`abstract_col`	column name for the column contains abstract.

Value

Returns a data.frame with results and time.

Examples

```
library(ellmer)
type_abstract <- type_object("Some key information from abstract.",
  no_patients_llm = type_integer("Find the total number of patients
                                 included in this study.",
                                 required = FALSE),
  no_AKI_llm = type_integer("Number of Acute Kidney Injury (AKI) patients
                            included in this study.",
                            required = FALSE),
  per_AKI_llm = type_number("Percentage of Acute Kidney Injury ",
                            required = FALSE),
  ICU_llm = type_boolean("Included only Intensive Care Unit (ICU) patients.",
                         required = FALSE),
  start_date = type_string("The starting date of the study written in
                           YYYY-MM-DD format",
                           required = FALSE),
  end_date = type_string("The end date of the study written in YYYY-MM-DD
                         format",
                         required = FALSE),
  age_mean_llm = type_number("Find the average age of the study cohort",
                             required = FALSE),
  age_median_llm = type_number("Find the median age of the study cohort",
                               required = FALSE)
)
## Not run:
process_with_ollama(abstracts,
                    type_abstract = type_abstract,
                    i = 1,
                    abstract_col = "Abstract")

## End(Not run)
```

---

prompt_engineering_app

prompt_engineering_app

Description

This Shiny app is designed for model testing and prompt engineering. It uses the training set—with manually extracted values from ‘glance_manual_app()‘ —as the reference for training and evaluation. When run, the function automatically creates a log_files folder in your current working directory. After you exit the prompt_engineering_app, a CSV file is saved in this folder, containing details of each prompt tested, results, runtime, model type, and accuracy. This log supports model selection, documentation, and reproducibility. The chosen model and prompt can be used for the ‘process_with_ollama()‘ function.

Usage

```
prompt_engineering_app()
```

Value

Runs a shiny app.

Examples

```
if (interactive()) {
 prompt_engineering_app()
}
```

`save_testing_data` *save_testing_data*

Description

This function will save the testing abstract data with empty columns to the processed data folder. This csv file will be used in the shinyapp 2 The data will be stored in metaextroctor_process_data

Usage

```
save_testing_data(testing_abs)
```

Arguments

`testing_abs`	training abstracts including the variables you want to extract.

Value

a csv file saved in the metaextroctor_process_data file named training_stage_0_data.csv

---

`save_training_data` *save_training_data*

Description

This function will save the training abstract data with empty columns to the processed data folder. This csv file will be used in the shinyapp 1 The data will be stored in metaextroctor_process_data

Usage

```
save_training_data(training_abs)
```

Arguments

`training_abs`	training abstracts including the variables you want to extract.

Value

a csv file saved in the metaextroctor_process_data file named training_stage_0_data.csv

`separate_training` *Separates a data into training and testing datasets*

Description

This function separates the abstracts into training and testing sets.

Usage

```
separate_training(data, percentage = 0.1)
```

Arguments

`data`	The csv file contains abstracts with
`percentage`	percentage of separation training sets. If percentage set to be 0.1, 10% of the data will be the training set and 90% of the data will be testing set.

Value

A list with two the train and test data.frames.

Examples

```
separate_training(abstracts, percentage = 0.4)
```

Index