Package 'artma'

April 25, 2025

Title Automatic Replication Tools for Meta-Analysis

Version 0.2.1

Author Petr Čala [aut, cre]

Maintainer Petr Čala <61505008@fsv.cuni.cz>

Description
Provides a unified and straightforward interface for performing a variety of meta-analysis methods directly from user data. Users can input a data frame, specify key parameters, and effortlessly execute and compare multiple common meta-analytic models. Designed for immediate usability, the package facilitates transparent, reproducible research without manual implementation of each analytical method. Ideal for researchers aiming for efficiency and reproducibility, it streamlines workflows from data preparation to results interpretation.

License GPL-3

URL https://github.com/PetrCala/artma

BugReports https://github.com/PetrCala/artma/issues

Depends R (>= 4.0.0)

Imports cli, glue, lifecycle, lintr, metafor, purrr, rlang, stringr, tidyverse, usethis, withr, yaml

Suggests box, box.linters, covr, devtools, fs, here, knitr, languageserver, mathjaxr, optparse, pkgbuild, remotes, rex, rmarkdown, roxygen2, testthat

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel TRUE

Config/testthat/start-first github-actions, release

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements JAGS >= 4.3.1 (https://mcmc-jags.sourceforge.io/)

NeedsCompilation no

Repository CRAN

Date/Publication 2025-04-25 13:50:06 UTC

Contents

`config.fix` *Fix the data config*

Description

Fix the data config.

Usage

```
config.fix(options_file_name = NULL, options_dir = NULL)
```

Arguments

`options_file_name`	*[character, optional]* The name of the options file to read the data config from. If `NULL` (default), the data config will be read from the `artma.data.config` option.
`options_dir`	*[character, optional]* The directory to read the options file from. If `NULL` (default), the current working directory will be used.

Value

[list] The fixed data config.

main *artma main*

Description

artma main

Usage

```
main(options = NULL, options_dir = NULL, FUN = NULL)
```

Arguments

options	*[character]* Name of the user options file to use.
options_dir	*[character]* Path to the directory that contains user options.
FUN	*[function]* The function to be called after the setup.

Value

[any] Depends on the main function definition.

methods.list *List methods*

Description

Print all runtime methods supported by artma into the console.

Usage

```
methods.list()
```

Value

NULL Prints the available methods into the console.

`options.copy` *Copy user options*

Description

Provide a name of a user options file to copy from, and a name of a file to copy to, and copy from the 'from' file to the 'to' file.

Usage

```
options.copy(
  options_file_name_from = NULL,
  options_file_name_to = NULL,
  options_dir = NULL,
  should_overwrite = NULL
)
```

Arguments

`options_file_name_from`
: *[character, optional]* Name of the options file to copy from. If not provided, the user will be prompted. Defaults to `NULL`.

`options_file_name_to`
: *[character, optional]* Name of the options file to copy to. If not provided, the user will be prompted. Defaults to `NULL`.

`options_dir`
: *[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to `NULL`.

`should_overwrite`
: *[logical, optional]* Whether to overwrite an existing file without asking. If `TRUE`, the file will be overwritten without prompting. If `FALSE`, the function will abort if the file already exists. If `NULL` (default), the user will be prompted.

Value

`NULL`

`options.create` *Create user options*

Description

Create a new user options file from an options template.

Usage

```
options.create(
  options_file_name = NULL,
  options_dir = NULL,
  template_path = NULL,
  user_input = list(),
  should_validate = TRUE,
  should_overwrite = FALSE,
  action_name = "creating"
)
```

Arguments

options_file_name	*[character]* Name of the new user options file, including the suffix.
options_dir	*[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to NULL.
template_path	*[character, optional]* Full path to the options template file.
user_input	*[list, optional]* A named list of user-supplied values for these options. If NULL or missing entries exist, the function will prompt the user via readline() (for required entries) or use defaults (for optional ones).
should_validate	*[logical, optional]* If TRUE, validate the new options file against the template. Defaults to TRUE.
should_overwrite	*[logical, optional]* If TRUE, overwrite the file if it already exists. Defaults to FALSE, in which case the user is prompted to confirm the overwrite.
action_name	*[character, optional]* A name for the action being performed. This is used for logging purposes. Defaults to "create". character Name of the newly created user options file as a character.

Value

NULL

---

options.delete *Delete user options*

Description

Provide a name of a user options file to delete, and delete that file.

Usage

```
options.delete(
  options_file_name = NULL,
  options_dir = NULL,
  skip_confirmation = FALSE
)
```

Arguments

options_file_name	*[character, optional]* Name of the options file to delete. If not provided, the user will be prompted. Defaults to NULL.
options_dir	*[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to NULL.
skip_confirmation	*[boolean, optional]* If passed as TRUE, the user will not be prompted for deletion confirmation. Defaults to FALSE.

Value

NULL

---

options.fix *Fix user options file*

Description

Fix a user options file by setting the default values for missing options.

Usage

```
options.fix(
  options_file_name = NULL,
  options_dir = NULL,
  template_path = NULL,
  force_default_overwrites = TRUE
)
```

Arguments

options_file_name	*[character, optional]* Name of the options file to fix, including the .yaml suffix. Defaults to NULL.
options_dir	*[character, optional]* Path to the folder in which to look for user options files. Defaults to NULL.
template_path	*[character, optional]* Path to the options template file. Defaults to NULL.

force_default_overwrites
: *[logical, optional]* If set to TRUE, the function will overwrite the existing options file with the default values. Defaults to TRUE.

Details

The function will attempt to load the user options file and validate it. If any errors are found, the function will attempt to fix them by setting the default values for the missing options.

Value

NULL Fixes the user options file.

---

options.help *Options Help*

Description

Prints information for each requested option (or all options if options is NULL).

Usage

```
options.help(options = NULL, template_path = NULL)
```

Arguments

options	*[character, optional]* A single option name (dot-separated) or a character vector thereof. If NULL, prints **all** options from" the template.
template_path	*[character, optional]* Path to the template YAML file. Defaults to PATHS$FILE_OPTIONS_TEMPLATE.

Value

Invisibly returns NULL, printing the requested information to the console.

---

options.list *List available user options*

Description

Retrieves the list of the existing options files and returns their names as a character vector. By default, this retrieves the names of the files including the yaml suffix, but can be modified to retrieve options verbose names instead.

Usage

```
options.list(options_dir = NULL, should_return_verbose_names = FALSE)
```

Arguments

`options_dir` *[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to `NULL`.

`should_return_verbose_names`
[logical, optional] If set to TRUE, the custom names of each of the options files are read and returned instead of file names. Defaults to FALSE.

Value

[vector, character] A character vector with the names of the options available.

`options.load` *Load user options*

Description

Load user options by their name and return them as a list.

Usage

```
options.load(
  options_file_name = NULL,
  options_dir = NULL,
  create_options_if_null = TRUE,
  load_with_prefix = TRUE,
  template_path = NULL,
  should_validate = TRUE,
  should_set_to_namespace = FALSE,
  should_add_temp_options = FALSE,
  should_return = TRUE
)
```

Arguments

`options_file_name`
[character, optional] Name of the options to load, including the .yaml suffix. Defaults to `NULL`.

`options_dir` *[character, optional]* Path to the folder in which to look for user options files. Defaults to `NULL`.

`create_options_if_null`
[logical, optional] If set to TRUE and the options file name is set to NULL, the function will prompt the user to create a new options file. Defaults to TRUE.

`load_with_prefix`
[logical, optional] Whether the options should be loaded with the package prefix. Defaults to TRUE.

`template_path` *[character, optional]* Path to the template YAML file. Defaults to `NULL`.

should_validate
: *[logical, optional]* Whether the options should be validated after loading. Defaults to TRUE.

should_set_to_namespace
: *[logical, optional]* Whether the options should be set in the options() namespace. Defaults to TRUE.

should_add_temp_options
: *[logical, optional]* Whether the options should be added to the temporary options. Defaults to FALSE.

should_return
: *[logical, optional]* Whether the function should return the list of options. Defaults to FALSE.

Details

In case the options name is not passed, the function will attempt to load the current options configuration. If none is found, it will then attempt to load the default options. If that fails too, an error is raised.

Value

[list|NULL] The loaded options as a list or NULL.

---

options.modify *Modify User Options*

Description

Modify an existing user options file with new values.

Usage

```
options.modify(
  options_file_name = NULL,
  options_dir = NULL,
  template_path = NULL,
  user_input = list(),
  should_validate = TRUE
)
```

Arguments

options_file_name
: *[character, optional]* Name of the user options file to modify, including the suffix.

options_dir
: *[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to NULL.

template_path
: *[character, optional]* Full path to the options template file. Defaults to NULL.

user_input *[list, optional]* A named list of user-supplied values for these options. If NULL or missing entries exist, the function will prompt the user via `readline()` (for required entries) or use defaults (for optional ones).

should_validate *[logical, optional]* If TRUE, validate the modified options file against the template. Defaults to TRUE.

Value

NULL

`options.open` *Options Open*

Description

Open an options file for editing. Must be run interactively.

Usage

```
options.open(options_file_name = NULL, options_dir = NULL)
```

Arguments

options_file_name *[character, optional]* Name of the user options file to modify, including the suffix.

options_dir *[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to NULL.

Value

NULL Opens the file for editing

`options.print_default_dir` *Print default user options directory*

Description

Prints the full path to the directory where user options are stored by default

Usage

```
options.print_default_dir(...)
```

Arguments

...	*[any]* Additional arguments.

Value

NULL Prints the default directory to console.

`options.validate` *Validate a user options file against an options template.*

Description

This function reads a YAML template and an options file, flattens both structures, and then checks that:

- Every option defined in the template is present in the options file.
- The value for each option is of the correct type.
- (Optionally) It warns about extra options in the file that are not defined in the template.

Usage

```
options.validate(
  options_file_name = NULL,
  options_dir = NULL,
  should_flag_redundant = FALSE,
  template_path = NULL,
  failure_action = "abort_verbose",
  verbose = TRUE
)
```

Arguments

options_file_name	*[character]* Name of the user options file to validate, including the suffix.
options_dir	*[character, optional]* Full path to the folder that contains user options files. If not provided, the default folder is chosen. Defaults to NULL.
should_flag_redundant	*[logical, optional]* If TRUE, warn the user about any extraneous options (i.e., options not defined in the options template, such as custom options that the user might have added). Defaults to FALSE.
template_path	*[character, optional]* Full path to the options template file. Defaults to NULL.
failure_action	*[character]* Action to take if validation fails. Can be one of: 'abort_verbose', 'abort_quiet', 'return_errors_verbose', 'return_errors_quiet'. Defaults to 'abort_verbose'.
verbose	*[logical, optional]* If TRUE, print additional information about the validation process. Defaults to TRUE. list Invisibly returns a list of error messages (empty if no errors).

Details

For each problem found (missing option or type mismatch), an error message is printed.

Value

[list] The validation errors

`run` *Run artma*

Description

Run artma with the specified methods and options.

Usage

```
run(methods = NULL, options_file_name = NULL, options_dir = NULL)
```

Arguments

`methods`	*[character, optional]* A character vector of the methods to invoke. Defaults to NULL.
`options_file_name`	*[character, optional]* The name of the options file to use. Defaults to NULL.
`options_dir`	*[character, optional]* The directory containing the options file. Defaults to NULL.

Value

[list] Results of the invocations, indexed by method names.

Index