Package 'artma'

October 27, 2026

Title Automatic Replication Tools for Meta-Analysis

Version 0.5.2

Author Petr Čala [aut, cre],
Matyáš Tvrz [ctb] (cluster-robust standard errors for frequentist model averaging)

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, https://petrcala.github.io/artma/

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

Depends R (>= 4.4.0)

Imports box (>= 1.2.0), cli (>= 3.6.5), climenu (>= 0.2.0), ggplot2 (>= 3.4.0), lifecycle (>= 1.0.4), lmtest (>= 0.9-40), memoise (>= 2.0.1), parallel, Rcpp (>= 1.0.12), rlang (>= 1.1.6), sandwich (>= 3.1-0), withr (>= 3.0.2), yaml (>= 2.3.10)

LinkingTo Rcpp

Suggests AER (>= 1.2-10), bayesm (>= 3.1-5), BMS (>= 0.3.4), car (>= 3.1-0), corrplot (>= 0.92), fdrtool (>= 1.2.17), haven (>= 2.5.0), htmlwidgets (>= 1.6.0), jsonlite (>= 1.8.0), knitr (>= 1.50), lintr (>= 3.2.0), MAIVE (>= 0.2.4), mice (>= 3.16.0), NlcOptim (>= 0.6), plm (>= 2.6-3), plotly (>= 4.10.0), puniform (>= 0.2.7), quadprog (>= 1.5-8), ragg (>= 1.5.0), rddensity (>= 2.5), readxl (>= 1.4.0), RhpcBLASctl (>= 0.23-42), rjags (>= 4.15), rmarkdown (>= 2.30), RoBMA (>= 4.0.0), testthat (>= 3.2.3), writexl (>= 1.4.0)

VignetteBuilder knitr

Config/Needs/coverage covr, pkgload, xml2

Config/Needs/dev box.linters, devtools, languageserver, remotes

Config/Needs/e2e remotes

Config/Needs/lint box.linters, devtools, remotes

Config/Needs/release devtools, optparse, pkgbuild, remotes

Config/Needs/website pkgdown

Config/testthat/edition 3

Config/testthat/parallel TRUE

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

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-10-27 08:10:02 UTC

Contents

artma . . . 3
autonomy_get . . . 6
autonomy_is_full . . . 7
autonomy_is_set . . . 7
autonomy_set . . . 8
cli_install . . . 9
cli_run . . . 9
config_fix . . . 11
config_get . . . 11
config_overrides . . . 12
config_reset . . . 12
config_set . . . 13
data_preview . . . 13
methods_list . . . 15
options_copy . . . 16
options_create . . . 17
options_delete . . . 18
options_diff . . . 19
options_fix . . . 20
options_help . . . 21
options_list . . . 21
options_load . . . 22
options_modify . . . 23
options_open . . . 24
options_print_default_dir . . . 25

options_validate 25
print.artma_box_plot 26
print.artma_funnel_plot 27
print.artma_t_stat_histogram 27
report_render 28
results_dir 29
results_open 29
viz_get 30
viz_set 31
viz_themes 32

Index **33**

`artma` *Run meta-analysis with artma*

Description

Main entry point for the artma package. This function orchestrates the complete meta-analysis workflow: loading options, preparing data, and running specified analytical methods.

Usage

```
artma(
  data = NULL,
  methods = NULL,
  options = NULL,
  options_dir = NULL,
  open_results = FALSE,
  ...
)
```

Arguments

`data`	*[data.frame, optional]* Data frame to analyze. If `NULL`, data will be loaded from the options file (see `options` parameter). When provided, this data will be used directly, bypassing the data reading step.
`methods`	*[character, optional]* A character vector of method names to run. Use `"all"` to run all available methods. If `NULL` and running interactively, the session hub opens: a menu loop for picking and running methods, previewing data, and opening results across repeated runs. See `artma::methods_list()` for available methods.
`options`	*[character, optional]* Name of the options file (with or without `.yaml` extension) to use. If `NULL` and running interactively, the session hub opens without one: with no options files yet you are guided straight into creating your first, and otherwise a picker of the existing files opens. Backing out of either lands in the hub menu, whose options-file item selects or creates the file the session runs on.

`options_dir`	*[character, optional]* Directory containing the options file. If `NULL`, uses the default options directory.
`open_results`	*[logical, optional]* Whether to open the results directory after exporting results. Defaults to `FALSE`.
`...`	Additional arguments passed to the runtime methods.

Details

The `artma()` function is the primary way to interact with the artma package. It handles the complete workflow:

1. **Options Loading**: Loads configuration from an options file (in the session hub, from the entry create-or-pick flow or the options-file menu item)
2. **Data Preparation**: Reads and prepares your data (unless `data` is provided)
3. **Method Execution**: Runs the specified analytical methods on your data
4. **Results**: Returns a structured list of results

Options Files:

Options files are YAML configuration files that store all settings for your analysis, including data paths, column mappings, method parameters, and output preferences. They ensure reproducibility and make it easy to manage multiple analysis configurations.

Methods:

Methods are analytical functions that perform specific meta-analysis tasks (e.g., funnel plots, Bayesian Model Averaging, effect size calculations). You can run multiple methods in a single call, and they will execute in a predefined order.

Data Parameter:

When `data` is provided, it bypasses the data reading step and uses your data frame directly. The data will still be preprocessed and validated according to your options configuration. This is useful when you already have data loaded in R or want to analyze data programmatically.

Parallel Execution:

Methods that do not depend on one another form a dependency layer and run concurrently in forked workers, so a run costs roughly the slowest method per layer rather than the sum of all of them. Each method's CLI output is captured and replayed in discovery order once the layer finishes, so the console reads like a sequential run.

Set `general.parallel` to `FALSE` in the options file to disable this. Execution also falls back to sequential automatically on Windows, on single-core machines, and in interactive sessions whose autonomy level still allows methods to prompt.

Every method receives its own L'Ecuyer-CMRG stream derived from the `general.seed` option and the method's name, so stochastic methods (bootstrap, MCMC) draw the same numbers whether the run was parallel or sequential, and regardless of which other methods ran alongside. Setting `general.seed` to `NA` derives the run seed from the session RNG instead, so calling `set.seed()` before `artma()` governs reproducibility the way it does for any stochastic R function.

Run Manifest:

Every run that saves results writes a `run.json` into its output directory: when it ran, the options file and data source behind it, the methods requested, run, skipped and failed, the effective seed, and the files it wrote. The file list is recorded as the files are written, so it describes this run rather than the accumulated contents of the directory, and it is what the HTML report uses to find each method's plots.

The manifest is overwritten on every run into the same output directory: it always describes the latest run, never a history. Runs driven by different options files already get their own output directory. To keep every run, set `output.run_subdirectories` to `TRUE`: each run then writes into `<output.dir>/runs/<timestamp>/`, with its own manifest, tables, graphics and report, and `artma::results_open()` opens the latest one.

Method Failures:

A method that throws an error does not abort the run. The failing method is skipped with a warning, the remaining methods still execute, and results from the methods that succeeded are exported as usual. A summary of successes and failures is printed at the end of the run. The run itself never signals an error because of a method failure; when every requested method fails, a final warning is emitted instead. Failed method names and their error messages are available in the `failed_methods` attribute of the returned list.

Value

[list] A named list containing results from each method, indexed by method name. The structure of each result depends on the specific method. Methods that fail are omitted from the list; their names and error messages are attached as the `failed_methods` attribute. The `run_info` attribute carries the run's identity: the methods requested, the effective seed, and the files each method wrote. A session hub call additionally attaches a `runs` attribute with one entry per run (methods, seed, timestamp, and the options changed since the previous run) and keeps the latest result per method across the session's runs.

See Also

- `artma::methods_list()` - List available methods
- `artma::options_create()` - Create a new options file
- `artma::prepare_data()` - Prepare data manually

Examples

```
## Not run:
# Interactive mode - will prompt for options and methods
results <- artma()

# Run specific methods with an options file
results <- artma(
  methods = c("funnel_plot", "bma", "fma"),
  options = "my_analysis.yaml"
)

# Run all methods
```

```
results <- artma(methods = "all", options = "my_analysis.yaml")

# Use data directly (bypasses file reading)
my_data <- data.frame(
  effect = c(0.5, 0.3, 0.7),
  se = c(0.1, 0.15, 0.12),
  study_id = c("Study A", "Study B", "Study C")
)
results <- artma(data = my_data, methods = "funnel_plot")

# Access results
funnel_result <- results$funnel_plot

## End(Not run)
```

`autonomy_get` *Get Autonomy Level*

Description

Get the current autonomy level. Autonomy controls how much user interaction is required during analysis. `interactive()` is the hard gate: non-interactive sessions never prompt, regardless of this setting.

Usage

```
autonomy_get()
```

Value

[character or NULL] The current autonomy level ("ask_more", "balanced", or "autonomous"), or NULL if not set.

Examples

```
## Not run:
# Get current autonomy level
level <- autonomy_get()
print(level)

## End(Not run)
```

autonomy_is_full *Check if Fully Autonomous*

Description

Check if the package is running in fully autonomous mode, i.e. the autonomy level is set to "autonomous", or the session is non-interactive (where prompts never happen regardless of the configured level).

Usage

```
autonomy_is_full()
```

Value

[logical] TRUE if fully autonomous, FALSE otherwise.

Examples

```
## Not run:
if (autonomy_is_full()) {
  message("Running in fully autonomous mode")
}

## End(Not run)
```

autonomy_is_set *Check if Autonomy Level is Set*

Description

Check if the autonomy level has been configured.

Usage

```
autonomy_is_set()
```

Value

[logical] TRUE if the autonomy level is set, FALSE otherwise.

Examples

```
## Not run:
if (!autonomy_is_set()) {
  message("Autonomy level not configured")
}

## End(Not run)
```

`autonomy_set` *Set Autonomy Level*

Description

Set the autonomy level for the current session. This setting controls how much user interaction is required during analysis.

Usage

```
autonomy_set(level)
```

Arguments

`level`	*[character]* The autonomy level to set. • "ask_more": Prompt for most decisions, including non-critical ones. • "balanced": Prompt for important decisions only. • "autonomous" (default): Minimal prompts; use defaults and auto-detection for most decisions. Legacy numeric levels (1-5) are still accepted and translated, with a warning (1-2 -> "ask_more", 3 -> "balanced", 4-5 -> "autonomous").

Value

`NULL` (invisible)

Examples

```
## Not run:
# Set to fully autonomous mode
autonomy_set("autonomous")

# Set to balanced mode
autonomy_set("balanced")

## End(Not run)
```

cli_install *Install the artma CLI launcher*

Description

Copy the launcher shipped inside the installed package to a directory on your PATH, so `artma <command>` works from any shell. The launcher relies solely on the installed package; it resolves nothing from its own location.

The no-install alternative is to call the CLI through Rscript directly: `Rscript -e 'artma::cli_run()' run --options my`

In interactive sessions the copy is confirmed via a menu. In non-interactive sessions it requires `force = TRUE` and otherwise aborts. It never writes inside `R.home()` and never elevates privileges.

Usage

```
cli_install(dir = "~/.local/bin", force = FALSE)
```

Arguments

dir	*[character, optional]* Directory to install the launcher into. Defaults to `"~/.local/bin"`.
force	*[logical, optional]* Required to install in a non-interactive session. Ignored (the menu governs) when interactive. Defaults to FALSE.

Value

[character] Path to the installed launcher (invisibly).

Examples

```
## Not run:
artma::cli_install()
artma::cli_install(dir = "~/bin", force = TRUE)

## End(Not run)
```

cli_run *Run the artma command-line interface*

Description

Scriptable entry point behind the artma launcher and `Rscript -e 'artma::cli_run()'`. It parses an argument vector, dispatches to the matching public function (`artma()`, `methods_list()`, `options_validate()` / `options_create()` / `options_list()`, or the package version) and returns an exit code. No analysis logic lives here: each subcommand is a thin translation to the existing API.

The function never calls `quit()`; it returns the exit code invisibly so it can be driven in-process by tests. The launcher script turns that code into a process exit status.

Exit codes:

- `0` success (including `--help`).
- `1` an R error was raised while dispatching (message printed to stderr).
- `2` a usage error (unknown command, unknown flag, malformed value); usage is printed to stderr.

Subcommands: `run`, `methods`, `options` (with sub-actions `validate`, `create`, `list`) and `version`. Every subcommand accepts `--help`.

Flags for `run` become an in-session `options()` overlay applied around the `artma()` call, so the user's YAML options file is never mutated: `--data` sets `artma.data.source_path`, `--output-dir` sets `artma.output.dir`, `--verbose` sets `artma.verbose`, `--no-cache` sets `artma.cache.use_cache` to `FALSE`, and `--report` sets `artma.output.report` to `TRUE`. `--options`, `--options-dir` and `--methods` are forwarded as `artma()` arguments.

In `--json` mode stdout carries only a JSON run manifest (`methods_run`, `methods_skipped` with reasons, `output_dir`, `exported_files`, `seed`, `package_version`), read back from the `run.json` the run wrote; all human-readable output is routed to stderr.

Usage

```
cli_run(args = commandArgs(trailingOnly = TRUE))
```

Arguments

`args`	*[character, optional]* The argument vector to parse. Defaults to `commandArgs(trailingOnly = TRUE)`.

Value

[integer] The exit code (invisibly).

Examples

```
## Not run:
# List available methods
artma::cli_run("methods")

# Run two methods against an options file, emitting a JSON manifest
artma::cli_run(c(
  "run", "--options", "my_analysis.yaml",
  "--methods", "funnel_plot,effect_summary_stats", "--json"
))

## End(Not run)
```

config_fix *Fix the data config*

Description

Regenerate the data config from the dataframe, clearing all overrides.

Usage

```
config_fix(options_file_name = NULL, options_dir = NULL)
```

Arguments

options_file_name	*[character, optional]* The name of the options file. If NULL (default), the user will be prompted interactively.
options_dir	*[character, optional]* The directory containing options files. If NULL (default), the default directory is used.

Value

[list] The fixed data config.

config_get *Get the resolved data config*

Description

Returns the fully-resolved data config (base defaults merged with sparse overrides). If var_name is provided, returns only that variable's config entry.

Usage

```
config_get(var_name = NULL, options_file_name = NULL, options_dir = NULL)
```

Arguments

var_name	*[character, optional]* A specific variable name to retrieve. If NULL (default), returns the entire config.
options_file_name	*[character, optional]* The name of the options file. If NULL (default), the user will be prompted interactively.
options_dir	*[character, optional]* The directory containing options files. If NULL (default), the default directory is used.

Value

[list] The fully-resolved data config (or a single entry).

`config_overrides` *View sparse config overrides*

Description

Returns only the sparse overrides that are actually persisted in the options file – i.e., only non-default field values.

Usage

```
config_overrides(options_file_name = NULL, options_dir = NULL)
```

Arguments

`options_file_name`	*[character, optional]* The name of the options file. If `NULL` (default), the user will be prompted interactively.
`options_dir`	*[character, optional]* The directory containing options files. If `NULL` (default), the default directory is used.

Value

[list] The sparse overrides (only non-default values).

`config_reset` *Reset variable config to defaults*

Description

Removes all overrides for a specific variable (or all variables), resetting them to auto-detected defaults.

Usage

```
config_reset(var_name = NULL, options_file_name = NULL, options_dir = NULL)
```

Arguments

`var_name`	*[character, optional]* The variable name to reset. If `NULL` (default), resets all overrides.
`options_file_name`	*[character, optional]* The name of the options file. If `NULL` (default), the user will be prompted interactively.
`options_dir`	*[character, optional]* The directory containing options files. If `NULL` (default), the default directory is used.

Value

[list] The updated fully-resolved data config (invisibly).

`config_set` *Set per-variable config overrides*

Description

Sets specific config fields for a variable. Only non-default values are persisted to the options file.

Usage

```
config_set(var_name, ..., options_file_name = NULL, options_dir = NULL)
```

Arguments

`var_name`	*[character]* The variable name to configure.
`...`	Named arguments for config fields to set (e.g., `bma = TRUE, bma_to_log = TRUE`). Column mappings are set the same way: `source_name = "study_name"` maps the variable to that column in the data file, and `drop_conflicting_raw = TRUE` keeps such a mapping while dropping a different raw column that occupies the standard name.
`options_file_name`	*[character, optional]* The name of the options file. If `NULL` (default), the user will be prompted interactively.
`options_dir`	*[character, optional]* The directory containing options files. If `NULL` (default), the default directory is used.

Value

[list] The updated fully-resolved data config (invisibly).

`data_preview` *Preview data*

Description

Open a data frame in R's viewer. Data can be supplied as a file path, a data frame, or loaded from an options file (with the same prompt flow as `artma()` when no options are given).

Usage

```
data_preview(
  data = NULL,
  options = NULL,
  options_dir = NULL,
  preprocess = TRUE
)
```

Arguments

data	*[character, data.frame, optional]* Either NULL, a length-one character path to a data file, or a data frame. If NULL, data is loaded from the options file (you will be prompted to select or create one in interactive mode).
options	*[character, optional]* Name of the options file (with or without .yaml extension). If NULL and options are required, you will be prompted in interactive mode.
options_dir	*[character, optional]* Directory containing the options file. If NULL, uses the default options directory.
preprocess	*[logical, optional]* If TRUE (default), data is run through the full pipeline (read, preprocess, compute) so the viewer shows what runtime methods receive. If FALSE, only the raw file read (for a path) or the given data frame is shown, without options-dependent standardization or preprocessing.

Details

Three data sources are supported:

- **Path**: pass a length-one character path to a data file. With preprocess = FALSE, the file is read without loading options (raw read). With preprocess = TRUE, options are loaded and the full pipeline is applied.
- **Data frame**: pass a data frame. With preprocess = FALSE, it is viewed as-is. With preprocess = TRUE, options are loaded and preprocess + compute are applied before viewing.
- **NULL**: data comes from the chosen options file (same "select or create options file" flow as artma() with empty arguments). With preprocess = TRUE (default), the full pipeline (read, preprocess, compute) is run. With preprocess = FALSE, only the data as read from file (with column standardization from options) is shown, without preprocessing or computed columns.

In non-interactive mode, when data is NULL and options is NULL, no viewer is shown (consistent with artma()).

Value

Invisible NULL. Opens the data in the standard R viewer (utils::View()). When no viewer is available (a non-interactive session, or a Unix session with no reachable display), the first rows are printed instead.

See Also

- artma - Run meta-analysis methods
- prepare_data() - Prepare data manually
- options_load - Load options

Examples

```
## Not run:
# Preview data from options file (prompts for file if NULL)
data_preview(options = "my_analysis.yaml")

# Preview raw file without loading options
data_preview("/path/to/data.csv", preprocess = FALSE)

# Preview preprocessed data from a path (uses options for standardization)
data_preview("/path/to/data.csv", options = "my_analysis.yaml")

# Preview a data frame as-is
data_preview(mtcars, preprocess = FALSE)

## End(Not run)
```

---

methods_list *List methods*

Description

Print an overview of the runtime methods supported by artma, and return it as a data frame. The table is built from the metadata each method registers with `register_runtime_method()`: what it does, which data columns it needs, which methods run before it, which optional packages it uses (and whether they are installed), and whether it is opt-in (left out of `methods = "all"`).

Pass a data frame to `available_for` to turn the table into a pre-flight check: each method's required columns are matched against that data, so a method that would be skipped is visible before the run rather than after.

Usage

```
methods_list(available_for = NULL)
```

Arguments

`available_for`	*[data.frame, optional]* Data to check each method's required columns against. When supplied, the returned frame gains a `missing_columns` column and an `available` flag. Defaults to `NULL`.

Value

[data.frame] Invisibly, one row per method, with the `method`, `description`, `required_columns`, `depends_on`, `suggests`, `missing_packages`, `installed`, and `opt_in` columns (plus `missing_columns` and `available` when `available_for` is supplied). Multi-value columns are comma-separated strings.

Examples

```
## Not run:
artma::methods_list()
artma::methods_list(available_for = my_data)

## End(Not run)
```

`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, using either flat dotted-path names (e.g. `list("data.source_path" = "...")`) or nested lists that mirror the YAML structure (e.g. `list(data = list(source_path = "..."))`). When overwriting an existing file, a partial edit to a list-type option (e.g. one entry of `data.columns`) is merged into the existing value instead of replacing it. 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. `options_remove()` is an alias.

Usage

```
options_delete(
  options_file_name = NULL,
  options_dir = NULL,
  skip_confirmation = FALSE
)

options_remove(
  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_diff` *Diff two user options files*

Description

Compares two user options files, printing the options whose values differ between them, followed by each file's deviations from the template defaults.

Usage

```
options_diff(
  options_file_name_a = NULL,
  options_file_name_b = NULL,
  options_dir = NULL,
  template_path = NULL
)
```

Arguments

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

`options_file_name_b`
: *[character, optional]* Name of the second options file to compare. 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`.

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

Details

List-typed options (such as `data.columns`) are compared entry by entry, so the diff names the individual column mappings that differ rather than reporting the whole store as changed.

Value

[list] Invisibly, a list with the compared `files`, a `differences` data frame, and a `deviations` list holding one data frame of template-default deviations per file.

`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), a group name, or a character vector thereof. If `NULL`, prints **all** options from the template as a tree.
`template_path`	*[character, optional]* Path to the template YAML file. Defaults to `PATHS$FILE_OPTIONS_TEMPLATE`.

Details

Called without arguments, the whole option tree is printed, grouped by top-level section, one line per option carrying its name, type and default. Named options are printed in full, with their help text. A name that matches a group rather than a single option (e.g. `"methods.bma"`, or `"methods"`) expands to every option underneath it.

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,
  details = FALSE,
  template_path = NULL
)
```

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.

`details` *[logical, optional]* If set to TRUE, a `data.frame` describing each options file is returned instead of a character vector: the file name, its resolved `data.source_path`, when the file was last modified, when it was last run, and how many of its options deviate from the template defaults. The last run time is read from the file's output directory, and is `NA` when the file has never produced any results. Defaults to FALSE.

`template_path` *[character, optional]* Full path to the options template file, used to resolve the defaults when `details = TRUE`. Defaults to `NULL`.

Value

[vector, character | data.frame] A character vector with the names of the options available, or a `data.frame` when `details = TRUE`.

`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_add_temp_options = FALSE,
  should_return = TRUE
)
```

Arguments

`options_file_name`
[character, optional] Name of the options to load. The `.yaml` suffix is appended automatically if missing. 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_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, using either flat dotted-path names (e.g. `list("data.source_path" = "...")`, matching what `options_load(load_with_prefix = FALSE)` returns) or nested lists that mirror the YAML structure (e.g. `list(data = list(source_path = "..."))`); both are flattened against the template before merging, so a partial edit to a list-type option (e.g. one entry of `data.columns`) is merged into the existing value instead of replacing it. 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. The editor is resolved from: (1) cli.editor option, (2) VISUAL/EDITOR env vars, or (3) system default file handler.

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"
)
```

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'. `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

`print.artma_box_plot` *Print method for box_plot results*

Description

Print method for box_plot results

Usage

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

Arguments

`x`	An artma_box_plot object
`...`	Additional arguments (ignored)

Value

x invisibly

print.artma_funnel_plot *Print method for funnel_plot results*

Description

Print method for funnel_plot results

Usage

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

Arguments

x	An artma_funnel_plot object
...	Additional arguments (ignored)

Value

x invisibly

print.artma_t_stat_histogram *Print method for t_stat_histogram results*

Description

Print method for t_stat_histogram results

Usage

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

Arguments

x	An artma_t_stat_histogram object
...	Additional arguments (ignored)

Value

x invisibly

`report_render` *Render an HTML report of a run's results*

Description

Turn the results of an `artma()` run into a single self-contained HTML file: a metadata header, a table of contents, and one section per method with its tables and plots. The file has no external dependencies (all styling is inline and every plot is embedded as a base64 PNG), so it can be shared on its own.

The report can also be produced automatically at the end of every run by setting the `output.report` option to `TRUE`.

Usage

```
report_render(results, output_file = NULL, open = interactive())
```

Arguments

`results`	*[list]* The named list returned by `artma()`, one `new_method_result` per method.
`output_file`	*[character, optional]* Absolute path of the HTML file to write. When `NULL` (default), the report is written as `report.html` inside the most recent run's resolved output directory.
`open`	*[logical, optional]* Whether to open the report in a browser after writing it. Only ever honoured in an interactive session. Defaults to `interactive()`.

Details

Plots come from the `run.json` manifest a run leaves in its output directory: it records which files each method wrote, so the report embeds exactly those. A results directory without a manifest (written before manifests existed, or by a run with `output.save_results` off) yields a report with tables only.

Value

[character] The path of the written report file (invisibly).

Examples

```
## Not run:
res <- artma(methods = c("funnel_plot", "effect_summary_stats"), options = "my.yaml")
report_render(res)

# Write to a specific location and open it
report_render(res, output_file = "~/analysis/report.html", open = TRUE)

## End(Not run)
```

results_dir *Get Results Directory Path*

Description

Returns the resolved path to the output directory where analysis results (tables, graphics) are saved. The path is printed and returned invisibly. When called without arguments, tries to use the most recently exported directory without prompting for an options file.

Usage

```
results_dir(options = NULL, options_dir = NULL)
```

Arguments

options	*[character, optional]* Name of the options file (with or without `.yaml` extension). If `NULL`, the function first checks for a recent export marker. If no marker is found and running interactively, you will be prompted to select an options file.
options_dir	*[character, optional]* Directory containing the options file. If `NULL`, uses the default options directory.

Value

[character] The resolved output directory path (invisibly).

Examples

```
## Not run:
# Get the most recent results directory
results_dir()

# Get results dir for a specific options file
results_dir(options = "my_analysis.yaml")

## End(Not run)
```

results_open *Open Results Directory*

Description

Opens the output directory in the system file browser (Finder on macOS, Explorer on Windows, or the default file manager on Linux). When called without arguments, tries to open the most recently exported results directory without prompting for an options file.

Usage

```
results_open(options = NULL, options_dir = NULL, use_last = TRUE)
```

Arguments

options	*[character, optional]* Name of the options file (with or without .yaml extension). If NULL, the function first checks for a recent export marker. If no marker is found and running interactively, you will be prompted to select an options file.
options_dir	*[character, optional]* Directory containing the options file. If NULL, uses the default options directory.
use_last	*[logical]* If TRUE (default) and no options/options_dir are provided, automatically open the most recently exported results directory. Set to FALSE to always resolve via the options file.

Value

[character] The resolved output directory path (invisibly).

Examples

```
## Not run:
# Open the most recent results (no prompt if a recent export exists)
results_open()

# Force options-based resolution (will prompt if needed)
results_open(use_last = FALSE)

# Open results for a specific options file
results_open(options = "my_analysis.yaml")

## End(Not run)
```

---

viz_get *Get Visualization Settings*

Description

Get the current visualization settings. Returns all settings as a list, or a single setting by name.

Usage

```
viz_get(option = NULL)
```

Arguments

option	*[character, optional]* Name of a specific option to retrieve. One of: "theme", "export_graphics", "export_path", "graph_scale". If NULL (default), returns all options as a named list.

Value

A named list of all visualization settings, or a single setting value.

Examples

```
## Not run:
# Get all visualization settings
viz_get()

# Get just the current theme
viz_get("theme")

# Get export path
viz_get("export_path")

## End(Not run)
```

`viz_set` *Set Visualization Settings*

Description

Set visualization options for the current session. Only provided arguments are changed; others remain unchanged.

Usage

```
viz_set(
  theme = NULL,
  export_graphics = NULL,
  export_path = NULL,
  graph_scale = NULL
)
```

Arguments

`theme`	*[character, optional]* Color theme. Use `viz_themes()` to see available themes.
`export_graphics`	*[logical, optional]* If TRUE, export plots to files.
`export_path`	*[character, optional]* Directory path for exported plots.
`graph_scale`	*[numeric, optional]* Scaling factor for exported graphics. Values > 1 increase resolution.

Value

Previous settings (invisibly), enabling easy restoration.

Examples

```
## Not run:
# Change theme
viz_set(theme = "purple")

# Enable export with custom path
viz_set(export_graphics = TRUE, export_path = "./output/plots")

# Save and restore settings
prev <- viz_set(theme = "red")
# ... do work ...
do.call(viz_set, prev)

## End(Not run)
```

`viz_themes` *List Available Themes*

Description

Get the names of all available visualization themes.

Usage

```
viz_themes()
```

Value

[character] Vector of valid theme names.

Examples

```
## Not run:
viz_themes()
# [1] "blue" "yellow" "green" "red" "purple"

## End(Not run)
```

Index

artma, 3, *15*
artma(), *28*
autonomy_get, 6
autonomy_is_full, 7
autonomy_is_set, 7
autonomy_set, 8

cli_install, 9
cli_run, 9
config_fix, 11
config_get, 11
config_overrides, 12
config_reset, 12
config_set, 13

data_preview, 13

methods_list, 15

options_copy, 16
options_create, 17
options_delete, 18
options_diff, 19
options_fix, 20
options_help, 21
options_list, 21
options_load, *15*, 22
options_modify, 23
options_open, 24
options_print_default_dir, 25
options_remove *(*options_delete*)*, 18
options_validate, 25

print.artma_box_plot, 26
print.artma_funnel_plot, 27
print.artma_t_stat_histogram, 27

report_render, 28
results_dir, 29
results_open, 29

viz_get, 30
viz_set, 31
viz_themes, 32