Package 'artma'

May 7, 2026

Title Automatic Replication Tools for Meta-Analysis

Version 0.3.3

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 (>= 3.6.5), climenu (>= 0.1.3), ggplot2 (>= 3.4.0), ggtext (>= 0.1.2), lintr (>= 3.2.0), lmtest (>= 0.9-40), memoise (>= 2.0.1), metafor (>= 4.8-0), NlcOptim (>= 0.6), plm (>= 2.6-3), 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), BMS (>= 0.3.4), box (>= 1.2.0), box.linters (>= 0.10.6), covr (>= 3.6.4), devtools (>= 2.4.5), fdrtool (>= 1.2.17), fs (>= 1.6.6), here (>= 1.0.2), ivmodel (>= 1.9.0), knitr (>= 1.50), languageserver (>= 0.3.16), MAIVE (>= 0.1.10), mathjaxr (>= 1.8-0), mice (>= 3.16.0), optparse (>= 1.7.5), pkgbuild (>= 1.4.8), quadprog (>= 1.5-8), rddensity (>= 2.5), remotes (>= 2.5.0), rex (>= 1.2.1), rmarkdown (>= 2.30), roxygen2 (>= 7.3.3), testthat (>= 3.2.3)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel TRUE

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

Encoding UTF-8

RoxygenNote 7.3.3

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-02-11 13:20:02 UTC

Contents

`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`, an interactive menu will prompt you to select methods. 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, you will be prompted to create or select an options file.
`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 (or prompts for creation in interactive mode)
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.

Value

[list] A named list containing results from each method, indexed by method name. The structure of each result depends on the specific method.

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.describe` *Get Autonomy Level Description*

Description

Get a human-readable description of an autonomy level.

Usage

```
autonomy.describe(level = NULL)
```

Arguments

`level`	*[integer, optional]* The autonomy level (1-5). If NULL, describes the current level.

Value

[character] A description of the autonomy level.

Examples

```
## Not run:
# Get description of current level
desc <- autonomy.describe()
print(desc)

# Get description of a specific level
desc <- autonomy.describe(5)
print(desc)

## End(Not run)
```

autonomy.get *Get Autonomy Level*

Description

Get the current autonomy level. Autonomy controls how much user interaction is required during analysis. Higher levels mean less user interaction and more automatic decision-making.

Usage

```
autonomy.get()
```

Value

[integer or NULL] The current autonomy level (1-5), 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 (level 5).

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.levels *List Available Autonomy Levels*

Description

Get information about all available autonomy levels.

Usage

```
autonomy.levels()
```

Value

[list] A list of autonomy level definitions.

Examples

```
## Not run:
levels <- autonomy.levels()
print(levels)

## 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`	*[integer]* The autonomy level to set (1-5). • 1 (Minimal): Maximum user control - prompt for all optional decisions • 2 (Low): Frequent prompts - ask for most non-critical decisions • 3 (Medium): Balanced - prompt for important decisions only • 4 (High): Mostly autonomous - minimal prompts for critical decisions only (default) • 5 (Full): Fully autonomous - no prompts, use all defaults and auto-detection

Value

`NULL` (invisible)

Examples

```
## Not run:
# Set to fully autonomous mode
autonomy.set(5)

# Set to balanced mode
autonomy.set(3)

## 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`).
`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()`).

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 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. 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.remove *Remove user options*

Description

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

Usage

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

Arguments

options_file_name	*[character, optional]* Name of the options file to remove. 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.

Details

This function is an alias for options.delete and behaves identically.

Value

NULL

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

---

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