Package 'metasurvey'

June 27, 2026

Title Reproducible Survey Data Processing with Step Pipelines

Version 0.0.24

URL https://metasurveyr.github.io/metasurvey/, https://github.com/metasurveyr/metasurvey

BugReports https://github.com/metasurveyr/metasurvey/issues

Description Provides a step-based pipeline for reproducible survey data processing, building on the 'survey' package for complex sampling designs. Supports rotating panels with bootstrap replicate weights, and provides a recipe system for sharing and reproducing data transformation workflows across survey editions.

License GPL (>= 3)

Imports data.table (>= 1.14.2), cli (>= 3.0.0), glue (>= 1.6.0), lifecycle (>= 1.0.0), jsonlite (>= 1.7.2), R6 (>= 2.5.0), survey (>= 4.2.1), methods

Suggests archive, convey, digest, httr2 (>= 1.0.0), haven, openxlsx, visNetwork (>= 2.0.9), roxygen2 (>= 7.1.2), testthat (>= 3.0.0), tibble (>= 3.1.3), dplyr (>= 1.0.7), knitr (>= 1.33), foreign (>= 0.8-81), rmarkdown (>= 2.11), parallel, rio (>= 0.5.27), here (>= 1.0.1), gt (>= 0.10.0), magrittr, shiny (>= 1.7.0), bslib (>= 0.5.0), bsicons (>= 0.1.0), htmltools (>= 0.5.0), xml2 (>= 1.3.0), eph, PNADcIBGE, ipumsr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1)

VignetteBuilder knitr

NeedsCompilation no

Author Mauro Loprete [aut, cre] (ORCID: <https://orcid.org/0000-0003-1560-0183>), Natalia da Silva [aut] (ORCID: <https://orcid.org/0000-0002-6031-7451>), Fabricio Machado [aut] (ORCID: <https://orcid.org/0009-0008-2653-9166>)

Maintainer Mauro Loprete <mauro.loprete@icloud.com>

Repository CRAN

Date/Publication 2026-06-27 17:20:02 UTC

Contents

add_category . . . 5
add_recipe . . . 6
add_replicate . . . 6
add_weight . . . 8
anda_download_microdata . . . 10
anda_variables . . . 11
api_comment_recipe . . . 12
api_comment_workflow . . . 12
api_delete_comment . . . 13
api_get_anda_variables . . . 14
api_get_recipe . . . 14
api_get_recipe_comments . . . 15
api_get_recipe_dependents . . . 16
api_get_recipe_stars . . . 16
api_get_workflow . . . 17
api_get_workflow_comments . . . 18
api_get_workflow_stars . . . 18
api_list_recipes . . . 19
api_list_workflows . . . 20
api_login . . . 21
api_logout . . . 22
api_me . . . 22
api_publish_recipe . . . 23
api_publish_workflow . . . 23
api_refresh_token . . . 24
api_register . . . 25
api_star_recipe . . . 26
api_star_workflow . . . 26
bake_recipes . . . 27
bake_steps . . . 28
cat_design . . . 29
cat_design_type . . . 30
certify_recipe . . . 31
configure_api . . . 32
default_categories . . . 32
evaluate_cv . . . 33
explore_recipes . . . 34
extract_surveys . . . 35
extract_time_pattern . . . 37
filter_recipes . . . 38
filter_workflows . . . 39

find_workflows_for_recipe . . . 40
get_backend . . . 40
get_data . . . 41
get_engine . . . 42
get_follow_up . . . 42
get_implantation . . . 44
get_metadata . . . 45
get_recipe . . . 46
get_steps . . . 48
get_workflow_backend . . . 49
group_dates . . . 50
harmonize . . . 50
has_design . . . 52
has_recipes . . . 53
has_steps . . . 54
is_baked . . . 54
lazy_default . . . 55
list_recipes . . . 56
list_workflows . . . 56
load_panel_survey . . . 57
load_survey . . . 59
load_survey_example . . . 61
parse_do_file . . . 62
parse_stata_labels . . . 63
PoolSurvey . . . 64
print.metasurvey_provenance . . . 65
print.metasurvey_provenance_diff . . . 66
print.Recipe . . . 67
print.RecipeWorkflow . . . 68
provenance . . . 69
provenance_diff . . . 70
provenance_to_json . . . 71
publish_recipe . . . 71
publish_workflow . . . 72
rank_recipes . . . 73
rank_workflows . . . 74
read_recipe . . . 74
read_workflow . . . 75
recipe . . . 76
Recipe-class . . . 78
RecipeCategory . . . 82
RecipeCertification . . . 84
RecipeUser . . . 86
RecipeWorkflow-class . . . 88
recipe_category . . . 92
recipe_certification . . . 93
recipe_user . . . 94
remove_category . . . 95

reproduce_workflow . . . 96
resolve_weight_spec . . . 97
RotativePanelSurvey . . . 98
save_recipe . . . 101
save_workflow . . . 102
search_recipes . . . 103
search_workflows . . . 104
set_backend . . . 104
set_data . . . 105
set_engine . . . 106
set_lazy_processing . . . 107
set_user_info . . . 107
set_use_copy . . . 108
set_version . . . 109
set_workflow_backend . . . 110
show_engines . . . 111
steps_to_recipe . . . 111
step_compute . . . 112
step_filter . . . 114
step_join . . . 115
step_recode . . . 117
step_remove . . . 119
step_rename . . . 121
step_validate . . . 122
Survey . . . 124
survey_empty . . . 129
survey_to_datatable . . . 130
survey_to_data_frame . . . 131
survey_to_tibble . . . 131
transpile_coverage . . . 132
transpile_stata . . . 133
transpile_stata_module . . . 134
use_copy_default . . . 135
validate_time_pattern . . . 136
view_graph . . . 136
workflow . . . 137
workflow_from_list . . . 139
workflow_table . . . 140

Index **142**

add_category *Add a category to a recipe*

Description

Pipe-friendly function to add a category to a Recipe object. Accepts either a category name (string) or a RecipeCategory object.

Usage

```
add_category(recipe, category, description = "")
```

Arguments

recipe	A Recipe object.
category	Character category name or RecipeCategory object.
description	Character. Description for the category (used when category is a string). Defaults to empty.

Value

The modified Recipe object (invisibly for piping).

See Also

remove_category, recipe_category, default_categories

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
r <- r |>
  add_category("labor_market", "Labor market indicators") |>
  add_category("income")
```

add_recipe *Add a recipe to a Survey*

Description

Tidy wrapper for `svy$add_recipe(recipe)`.

Usage

```
add_recipe(svy, recipe, bake = lazy_default())
```

Arguments

`svy`	Survey object
`recipe`	A Recipe object
`bake`	Logical; whether to bake immediately (default: lazy_default())

Value

The Survey object (invisibly), modified in place

See Also

Other recipes: `Recipe-class`, `bake_recipes()`, `explore_recipes()`, `get_recipe()`, `harmonize()`, `print.Recipe()`, `publish_recipe()`, `read_recipe()`, `recipe()`, `save_recipe()`, `steps_to_recipe()`

Examples

```
svy <- survey_empty(type = "ech", edition = "2023")
r <- recipe(
  name = "Example", user = "test",
  svy = svy, description = "Example"
)
svy <- add_recipe(svy, r)
```

add_replicate *Configure replicate weights for variance estimation*

Description

This function configures replicate weights (bootstrap, jackknife, etc.) that allow estimation of variance for complex statistics in surveys with complex sampling designs. It is essential for obtaining correct standard errors in population estimates.

Usage

```
add_replicate(
  weight,
  replicate_pattern,
  replicate_path = NULL,
  replicate_id = NULL,
  replicate_type
)
```

Arguments

`weight`	String with the name of the main weight variable in the survey (e.g., "pesoano", "pesomes")
`replicate_pattern`	String with regex pattern to identify replicate weight columns. Examples: "wr\d+" for columns wr1, wr2, etc.
`replicate_path`	Path to the file containing replicate weights. If NULL, assumes they are in the same main dataset
`replicate_id`	Named vector specifying how to join between the main dataset and replicate file. Format: c("main_var" = "replicate_var")
`replicate_type`	Type of replication used. Options: "bootstrap", "jackknife", "BRR" (Balanced Repeated Replication)

Details

Replicate weights are essential for:

- Correctly estimating variance in complex designs
- Calculating appropriate confidence intervals
- Obtaining reliable coefficients of variation
- Performing valid statistical tests

The regex pattern must exactly match the replicate weight column names in the file. For example, if columns are named "wr001", "wr002", etc., use the pattern "wr\\d+".

This function is typically used within `add_weight()` for more complex weight configurations.

Value

List with replicate configuration that will be used by the sampling design for variance estimation

See Also

`add_weight` for complete weight configuration `svrepdesign` for replicate design in survey package `load_survey` where this configuration is used

Other weights: `add_weight()`, `resolve_weight_spec()`

Examples

```
# Basic configuration with external file
annual_replicates <- add_replicate(
  weight = "pesoano",
  replicate_pattern = "wr\\d+",
  replicate_path = "bootstrap_weights_2023.xlsx",
  replicate_id = c("ID_HOGAR" = "ID"),
  replicate_type = "bootstrap"
)

# With replicates in same dataset
integrated_replicates <- add_replicate(
  weight = "main_weight",
  replicate_pattern = "rep_\\d{3}",
  replicate_type = "jackknife"
)

# Use within add_weight
weight_config <- add_weight(
  annual = add_replicate(
    weight = "pesoano",
    replicate_pattern = "wr\\d+",
    replicate_path = "bootstrap_annual.xlsx",
    replicate_id = c("numero" = "ID_HOGAR"),
    replicate_type = "bootstrap"
  ),
  monthly = "pesomes"
)
```

---

add_weight *Configure weights by periodicity for Survey objects*

Description

This function creates a weight structure that allows specifying different weight variables according to estimation periodicity. It is essential for proper functioning of workflows with multiple temporal estimation types.

Usage

```
add_weight(monthly = NULL, annual = NULL, quarterly = NULL, biannual = NULL)
```

Arguments

monthly	String with monthly weight variable name, or replicate list created with add_replicate for monthly weights
annual	String with annual weight variable name, or replicate list for annual weights
quarterly	String with quarterly weight variable name, or replicate list for quarterly weights
biannual	String with biannual weight variable name, or replicate list for biannual weights

Details

This function is fundamental for surveys that require different weights according to temporal estimation type. For example, Uruguay's ECH has specific weights for monthly, quarterly, and annual estimations.

Each parameter can be:

- A simple string with the weight variable name
- A replicate structure created with `add_replicate()` for bootstrap or jackknife estimations

Weights are automatically selected in `workflow()` according to the specified `estimation_type` parameter.

Value

Named list with weight configuration by periodicity, which will be used by `load_survey` and `workflow` to automatically select the appropriate weight

See Also

`add_replicate` to configure bootstrap/jackknife replicates `load_survey` where this configuration is used `workflow` that automatically selects weights

Other weights: `add_replicate()`, `resolve_weight_spec()`

Examples

```
# Basic configuration with simple weight variables
ech_weights <- add_weight(
  monthly = "pesomes",
  quarterly = "pesotri",
  annual = "pesoano"
)

# With bootstrap replicates for variance estimation
weights_with_replicates <- add_weight(
  monthly = add_replicate(
    weight = "pesomes",
    replicate_pattern = "wr\\d+",
    replicate_path = "monthly_replicate_weights.xlsx",
    replicate_id = c("ID_HOGAR" = "ID"),
    replicate_type = "bootstrap"
  ),
  annual = "pesoano"
)
```

`anda_download_microdata` *Download ECH microdata from ANDA5*

Description

[Experimental]

Downloads microdata files for a given ECH edition from INE Uruguay's ANDA5 catalog. Automatically accepts the terms of use, parses available resources, and downloads the appropriate file.

For editions >= 2022, ANDA provides separate files for implantation, monthly follow-ups, and bootstrap replicate weights. Use the `resource` parameter to select which file to download.

Usage

```
anda_download_microdata(
  edition,
  resource = "implantation",
  dest_dir = tempdir(),
  base_url = "https://www4.ine.gub.uy/Anda5"
)
```

Arguments

`edition`	Character year (e.g., "2023")
`resource`	Character type of resource to download. One of: **"implantation"** (default) Main implantation file. For editions < 2022, downloads the main microdata file. **"monthly"** Monthly follow-up files (editions >= 2022 only). Returns a character vector of paths, one per month. **"bootstrap_annual"** Annual bootstrap replicate weights. **"bootstrap_monthly"** Monthly bootstrap replicate weights. **"bootstrap_quarterly"** Quarterly bootstrap replicate weights. **"bootstrap_semestral"** Semestral bootstrap replicate weights. **"poverty"** Poverty line microdata (Microdatos_LP).
`dest_dir`	Character directory where to save files. Defaults to a temporary directory.
`base_url`	Character base URL of the ANDA5 instance

Value

Character path (or vector of paths for monthly) to the downloaded file(s), ready to pass to `load_survey()` or `data.table::fread()`.

See Also

Other anda: `anda_variables()`, `api_get_anda_variables()`

Examples

```
## Not run:
path <- anda_download_microdata("2023", resource = "implantation")
svy <- load_survey(path, svy_type = "ech", svy_edition = "2023")

## End(Not run)
```

`anda_variables` *Query ANDA variable metadata from the API*

Description

[Experimental]

Fetches variable metadata (labels, types, value labels) from the metasurvey API's ANDA endpoint.

Usage

```
anda_variables(survey_type = "ech", var_names = NULL)
```

Arguments

`survey_type`	Character survey type (default "ech")
`var_names`	Character vector of variable names to look up. If NULL, returns all variables for the survey type.

Value

A data.frame with columns: name, label, type

See Also

Other anda: `anda_download_microdata()`, `api_get_anda_variables()`

Examples

```
## Not run:
anda_variables("ech", c("pobpcoac", "e27"))

## End(Not run)
```

api_comment_recipe *Add a comment to a recipe*

Description

Post a text comment on a recipe. Requires authentication.

Usage

```
api_comment_recipe(id, text)
```

Arguments

id	Recipe ID
text	Comment text (max 2000 characters)

Value

Invisibly, the API response.

See Also

Other api-comments: api_comment_workflow(), api_delete_comment(), api_get_recipe_comments(), api_get_workflow_comments()

Examples

```
## Not run:
api_comment_recipe("r_1739654400_742", "Great recipe!")

## End(Not run)
```

api_comment_workflow *Add a comment to a workflow*

Description

Post a text comment on a workflow. Requires authentication.

Usage

```
api_comment_workflow(id, text)
```

Arguments

id	Workflow ID
text	Comment text (max 2000 characters)

Value

Invisibly, the API response.

See Also

Other api-comments: api_comment_recipe(), api_delete_comment(), api_get_recipe_comments(), api_get_workflow_comments()

Examples

```
## Not run:
api_comment_workflow("w_1739654400_123", "Very useful workflow!")

## End(Not run)
```

---

api_delete_comment *Delete a comment*

Description

Delete a comment by its ID. Only the comment author can delete it. Requires authentication.

Usage

```
api_delete_comment(comment_id)
```

Arguments

comment_id	Comment ID

Value

Invisibly, the API response.

See Also

Other api-comments: api_comment_recipe(), api_comment_workflow(), api_get_recipe_comments(), api_get_workflow_comments()

Examples

```
## Not run:
api_delete_comment("c_abc123")

## End(Not run)
```

api_get_anda_variables *Get ANDA variable metadata from the API*

Description

Get ANDA variable metadata from the API

Usage

```
api_get_anda_variables(survey_type = "ech", var_names = NULL)
```

Arguments

survey_type	Character survey type (default "ech")
var_names	Character vector of variable names. If NULL, returns all.

Value

A list of variable metadata objects

See Also

Other anda: anda_download_microdata(), anda_variables()

Examples

```
## Not run:
api_get_anda_variables("ech", c("pobpcoac", "e27"))

## End(Not run)
```

api_get_recipe *Get recipe(s) by ID*

Description

Retrieve one or more recipes from the API by their IDs.

Usage

```
api_get_recipe(id)
```

Arguments

id	Character vector of recipe ID(s). If length > 1, returns a list.

Value

A single Recipe object (or NULL) when `length(id) == 1`. A list of Recipe objects when `length(id) > 1` (NULLs are dropped).

See Also

Other api-recipes: `api_get_recipe_dependents()`, `api_list_recipes()`, `api_publish_recipe()`

Examples

```
## Not run:
api_get_recipe("r_1739654400_742")

## End(Not run)
```

`api_get_recipe_comments` *Get comments for a recipe*

Description

List all comments on a recipe, sorted by creation date.

Usage

```
api_get_recipe_comments(id)
```

Arguments

`id`	Recipe ID

Value

List of comment objects.

See Also

Other api-comments: `api_comment_recipe()`, `api_comment_workflow()`, `api_delete_comment()`, `api_get_workflow_comments()`

Examples

```
## Not run:
api_get_recipe_comments("r_1739654400_742")

## End(Not run)
```

api_get_recipe_dependents *Get recipes that depend on a recipe*

Description

Find all recipes whose `depends_on_recipes` field references the given recipe ID (backlinks).

Usage

```
api_get_recipe_dependents(id)
```

Arguments

`id`	Recipe ID

Value

List of recipe summaries (id, name, user).

See Also

Other api-recipes: `api_get_recipe()`, `api_list_recipes()`, `api_publish_recipe()`

Examples

```
## Not run:
api_get_recipe_dependents("r_1739654400_742")

## End(Not run)
```

api_get_recipe_stars *Get star summary for a recipe*

Description

Returns average rating, count, and the current user's rating (if authenticated).

Usage

```
api_get_recipe_stars(id)
```

Arguments

`id`	Recipe ID

Value

List with average, count, and optionally user_value.

See Also

Other api-stars: api_get_workflow_stars(), api_star_recipe(), api_star_workflow()

Examples

```
## Not run:
api_get_recipe_stars("r_1739654400_742")

## End(Not run)
```

---

api_get_workflow *Get a single workflow by ID*

Description

Retrieve a workflow from the API by its ID.

Usage

```
api_get_workflow(id)
```

Arguments

id	Workflow ID

Value

RecipeWorkflow object or NULL

See Also

Other api-workflows: api_list_workflows(), api_publish_workflow()

Examples

```
## Not run:
api_get_workflow("w_1739654400_123")

## End(Not run)
```

api_get_workflow_comments

Get comments for a workflow

Description

List all comments on a workflow, sorted by creation date.

Usage

```
api_get_workflow_comments(id)
```

Arguments

id	Workflow ID

Value

List of comment objects.

See Also

Other api-comments: api_comment_recipe(), api_comment_workflow(), api_delete_comment(), api_get_recipe_comments()

Examples

```
## Not run:
api_get_workflow_comments("w_1739654400_123")

## End(Not run)
```

api_get_workflow_stars

Get star summary for a workflow

Description

Returns average rating, count, and the current user's rating (if authenticated).

Usage

```
api_get_workflow_stars(id)
```

Arguments

id	Workflow ID

Value

List with `average`, `count`, and optionally `user_value`.

See Also

Other api-stars: `api_get_recipe_stars()`, `api_star_recipe()`, `api_star_workflow()`

Examples

```
## Not run:
api_get_workflow_stars("w_1739654400_123")

## End(Not run)
```

`api_list_recipes` *List recipes from API*

Description

Fetch recipes with optional search and filters.

Usage

```
api_list_recipes(
  search = NULL,
  survey_type = NULL,
  topic = NULL,
  certification = NULL,
  user = NULL,
  limit = 50,
  offset = 0
)
```

Arguments

`search`	Text search (matches name/description)
`survey_type`	Filter by survey type (e.g., `"ech"`)
`topic`	Filter by topic
`certification`	Filter by certification level
`user`	Filter by author email
`limit`	Maximum results (default 50)
`offset`	Skip first N results (default 0)

Value

List of Recipe objects

See Also

Other api-recipes: api_get_recipe(), api_get_recipe_dependents(), api_publish_recipe()

Examples

```
## Not run:
configure_api("https://metasurvey-api.example.com")
recipes <- api_list_recipes(survey_type = "ech")

## End(Not run)
```

`api_list_workflows` *List workflows from API*

Description

Fetch workflows with optional search and filters.

Usage

```
api_list_workflows(
  search = NULL,
  survey_type = NULL,
  recipe_id = NULL,
  user = NULL,
  limit = 50,
  offset = 0
)
```

Arguments

search	Text search
survey_type	Filter by survey type
recipe_id	Filter workflows that reference this recipe
user	Filter by author email
limit	Maximum results
offset	Skip first N results

Value

List of RecipeWorkflow objects

See Also

Other api-workflows: api_get_workflow(), api_publish_workflow()

Examples

```
## Not run:
api_list_workflows(survey_type = "ech")

## End(Not run)
```

`api_login` *Login*

Description

Authenticate with the metasurvey API. On success the JWT token is stored automatically.

Usage

```
api_login(email, password)
```

Arguments

`email`	Email address
`password`	Password

Value

Invisibly, the API response.

See Also

Other api-auth: `api_logout()`, `api_me()`, `api_refresh_token()`, `api_register()`, `configure_api()`

Examples

```
## Not run:
api_login("ana@example.com", "s3cret")

## End(Not run)
```

api_logout *Logout*

Description

Clear the stored API token from memory and the environment.

Usage

```
api_logout()
```

Value

Invisibly, NULL.

See Also

Other api-auth: `api_login()`, `api_me()`, `api_refresh_token()`, `api_register()`, `configure_api()`

Examples

```
api_logout()
```

api_me *Get current user profile*

Description

Returns profile info for the currently authenticated user.

Usage

```
api_me()
```

Value

List with user fields (name, email, user_type, etc.)

See Also

Other api-auth: `api_login()`, `api_logout()`, `api_refresh_token()`, `api_register()`, `configure_api()`

Examples

```
## Not run:
api_me()

## End(Not run)
```

api_publish_recipe *Publish a recipe*

Description

Publish a Recipe object to the API. Requires authentication (call `api_login()` first).

Usage

```
api_publish_recipe(recipe)
```

Arguments

`recipe`	A Recipe object

Value

Invisibly, the API response with the assigned ID.

See Also

Other api-recipes: `api_get_recipe()`, `api_get_recipe_dependents()`, `api_list_recipes()`

Examples

```
## Not run:
api_publish_recipe(my_recipe)

## End(Not run)
```

api_publish_workflow *Publish a workflow*

Description

Publish a RecipeWorkflow object to the API. Requires authentication.

Usage

```
api_publish_workflow(workflow)
```

Arguments

`workflow`	A RecipeWorkflow object

Value

Invisibly, the API response.

See Also

Other api-workflows: `api_get_workflow()`, `api_list_workflows()`

Examples

```
## Not run:
api_publish_workflow(my_workflow)

## End(Not run)
```

`api_refresh_token` *Refresh JWT token*

Description

Request a new JWT token using the current (still valid) token. The new token is stored automatically. This is called internally by `api_request()` when the current token is close to expiry (within 5 minutes).

Usage

```
api_refresh_token()
```

Value

The new token string (invisibly), or NULL if refresh fails.

See Also

Other api-auth: `api_login()`, `api_logout()`, `api_me()`, `api_register()`, `configure_api()`

Examples

```
## Not run:
api_refresh_token()

## End(Not run)
```

`api_register` *Register a new user*

Description

Create an account on the metasurvey API. On success the JWT token is stored automatically via `options(metasurvey.api_token)`.

Usage

```
api_register(
  name,
  email,
  password,
  user_type = "individual",
  institution = NULL
)
```

Arguments

`name`	Display name
`email`	Email address
`password`	Password
`user_type`	One of `"individual"`, `"institutional_member"`, `"institution"`
`institution`	Institution name (required for `"institutional_member"`)

Value

Invisibly, the API response (list with `ok`, `token`, `user`).

See Also

Other api-auth: `api_login()`, `api_logout()`, `api_me()`, `api_refresh_token()`, `configure_api()`

Examples

```
## Not run:
configure_api("https://metasurvey-api.example.com")
api_register("Ana Garcia", "ana@example.com", "s3cret")

## End(Not run)
```

api_star_recipe *Rate a recipe*

Description

Give a star rating (1-5) to a recipe. Requires authentication. Each user can have one rating per recipe (upserts on subsequent calls).

Usage

```
api_star_recipe(id, value)
```

Arguments

id	Recipe ID
value	Integer rating from 1 to 5

Value

Invisibly, the API response.

See Also

Other api-stars: api_get_recipe_stars(), api_get_workflow_stars(), api_star_workflow()

Examples

```
## Not run:
api_star_recipe("r_1739654400_742", 5)

## End(Not run)
```

api_star_workflow *Rate a workflow*

Description

Give a star rating (1-5) to a workflow. Requires authentication.

Usage

```
api_star_workflow(id, value)
```

Arguments

id	Workflow ID
value	Integer rating from 1 to 5

Value

Invisibly, the API response.

See Also

Other api-stars: api_get_recipe_stars(), api_get_workflow_stars(), api_star_recipe()

Examples

```
## Not run:
api_star_workflow("w_1739654400_123", 4)

## End(Not run)
```

`bake_recipes` *Bake recipes*

Description

Bake recipes

Usage

```
bake_recipes(svy)
```

Arguments

svy	Survey object

Value

Survey object with all recipes applied

See Also

Other recipes: Recipe-class, add_recipe(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), publish_recipe(), read_recipe(), recipe(), save_recipe(), steps_to_recipe()

Examples

```
dt <- data.table::data.table(id = 1:20, x = rnorm(20), w = runif(20, 0.5, 2))
svy <- Survey$new(
  data = dt, edition = "2023", type = "demo",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
r <- recipe(
  name = "Demo", user = "test", svy = svy,
  description = "Demo recipe"
```

```
)
svy <- add_recipe(svy, r)
processed <- bake_recipes(svy)
```

`bake_steps` *Execute all pending steps*

Description

Iterates over all pending (lazy) steps attached to a Survey or RotativePanelSurvey and executes them sequentially, mutating the underlying data.table. Each step is validated before execution (checks that required variables exist).

Usage

```
bake_steps(svy)
```

Arguments

`svy`	A `Survey` or `RotativePanelSurvey` object with pending steps

Details

Steps are executed in the order they were added. Each step's expressions can reference variables created by previous steps.

For RotativePanelSurvey objects, steps are applied to both the implantation and all follow-up surveys.

Value

The same object with all steps materialized in the data and each step marked as `bake = TRUE`.

See Also

Other steps: `get_steps()`, `step_compute()`, `step_filter()`, `step_join()`, `step_recode()`, `step_remove()`, `step_rename()`, `step_validate()`, `view_graph()`

Examples

```
dt <- data.table::data.table(id = 1:5, age = c(15, 30, 45, 50, 70), w = 1)
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy <- step_compute(svy, age2 = age * 2)
svy <- bake_steps(svy)
get_data(svy)
```

`cat_design` *Display survey design information*

Description

Pretty-prints the sampling design configuration for each estimation type in a Survey object, showing PSU, strata, weights, and other design elements in a color-coded, readable format.

Usage

```
cat_design(self)
```

Arguments

`self`	Survey object containing design information

Details

This function displays design information including:

- Primary Sampling Units (PSU/clusters)
- Stratification variables
- Weight variables for each estimation type
- Finite Population Correction (FPC) if used
- Calibration formulas if applied
- Overall design type classification

Output is color-coded for better readability in supporting terminals.

Value

Invisibly returns NULL; called for side effect of printing design info

See Also

`cat_design_type` for design type classification

Other survey-objects: `Survey`, `cat_design_type()`, `get_data()`, `get_metadata()`, `has_design()`, `has_recipes()`, `has_steps()`, `is_baked()`, `set_data()`, `survey_empty()`, `survey_to_data_frame()`, `survey_to_datatable()`, `survey_to_tibble()`

Examples

```
dt <- data.table::data.table(id = 1:20, x = rnorm(20), w = runif(20, 0.5, 2))
svy <- Survey$new(
  data = dt, edition = "2023", type = "demo",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
cat_design(svy)
```

`cat_design_type` *cat_design_type*

Description

Cast design type from survey

Usage

```
cat_design_type(self, design_name)
```

Arguments

`self`	Object of class Survey
`design_name`	Name of design

Value

Character string describing the design type, or "None".

See Also

Other survey-objects: `Survey`, `cat_design()`, `get_data()`, `get_metadata()`, `has_design()`, `has_recipes()`, `has_steps()`, `is_baked()`, `set_data()`, `survey_empty()`, `survey_to_data_frame()`, `survey_to_datatable()`, `survey_to_tibble()`

Examples

```
dt <- data.table::data.table(id = 1:20, x = rnorm(20), w = runif(20, 0.5, 2))
svy <- Survey$new(
  data = dt, edition = "2023", type = "demo",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
svy$ensure_design()
cat_design_type(svy, "annual")
```

`certify_recipe` *Certify a recipe*

Description

Pipe-friendly function to certify a Recipe at a given quality level.

Usage

```
certify_recipe(recipe, user, level)
```

Arguments

`recipe`	A Recipe object.
`user`	RecipeUser who is certifying.
`level`	Character. Certification level: `"reviewed"` or `"official"`.

Value

The modified Recipe object.

See Also

`recipe_certification`, `recipe_user`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `default_categories()`, `filter_recipes()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_category()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
inst <- recipe_user("IECON", type = "institution")
r <- r |> certify_recipe(inst, "official")
```

configure_api *Configure metasurvey API*

Description

Set API base URL and optionally load stored credentials. The URL can also be set via the METASURVEY_API_URL environment variable, and the token via METASURVEY_TOKEN.

Usage

```
configure_api(url)
```

Arguments

url	API base URL (e.g., "https://metasurvey-api.example.com")

Value

Invisibly, the previous URL (for restoring).

See Also

Other api-auth: api_login(), api_logout(), api_me(), api_refresh_token(), api_register()

Examples

```
configure_api(url = "https://metasurvey-api.example.com")
```

default_categories *Default recipe categories*

Description

Returns a list of standard built-in categories for recipe classification.

Usage

```
default_categories()
```

Value

List of RecipeCategory objects

See Also

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
cats <- default_categories()
vapply(cats, function(c) c$name, character(1))
```

---

evaluate_cv *Evaluate estimation with Coefficient of Variation*

Description

Evaluate estimation with Coefficient of Variation

Usage

```
evaluate_cv(cv)
```

Arguments

cv	Numeric coefficient of variation value.

Value

Character string with the quality category (e.g. "Excellent", "Good").

See Also

Other workflows: RecipeWorkflow-class, print.RecipeWorkflow(), publish_workflow(), read_workflow(), reproduce_workflow(), save_workflow(), workflow(), workflow_from_list(), workflow_table()

Examples

```
evaluate_cv(3) # "Excellent"
evaluate_cv(12) # "Good"
evaluate_cv(30) # "Use with caution"
```

`explore_recipes` *Launch the Recipe Explorer Shiny App*

Description

Opens an interactive web application to explore, search, and browse metasurvey recipes with visual documentation cards. Supports user registration and login via MongoDB Atlas.

Usage

```
explore_recipes(port = NULL, host = "127.0.0.1", launch.browser = TRUE)
```

Arguments

`port`	Integer port number, or NULL for automatic.
`host`	Character. The host to listen on. Defaults to `"127.0.0.1"` for local use. Set to `"0.0.0.0"` for server deployments (Railway, etc.).
`launch.browser`	Logical. Open the app in a browser?

Value

NULL (called for side effect of launching the app).

See Also

Other recipes: `Recipe-class`, `add_recipe()`, `bake_recipes()`, `get_recipe()`, `harmonize()`, `print.Recipe()`, `publish_recipe()`, `read_recipe()`, `recipe()`, `save_recipe()`, `steps_to_recipe()`

Examples

```
## Not run:
# Local / RStudio viewer
explore_recipes()

# Server deployment (Railway, Docker, etc.)
explore_recipes(host = "0.0.0.0", port = 3838, launch.browser = FALSE)

## End(Not run)
```

`extract_surveys` *Extract surveys by periodicity from a rotating panel*

Description

Extracts subsets of surveys from a RotativePanelSurvey object based on temporal criteria. Allows obtaining surveys for different types of analysis (monthly, quarterly, annual) respecting the rotating panel's temporal structure.

Usage

```
extract_surveys(
  RotativePanelSurvey,
  index = NULL,
  monthly = NULL,
  annual = NULL,
  quarterly = NULL,
  biannual = NULL,
  use.parallel = FALSE
)
```

Arguments

`RotativePanelSurvey`	A `RotativePanelSurvey` object containing the rotating panel surveys organized temporally
`index`	Integer vector specifying survey indices to extract. If a single value, returns that survey; if a vector, returns a list
`monthly`	Integer vector specifying which months to extract for monthly analysis (1-12)
`annual`	Integer vector specifying which years to extract for annual analysis
`quarterly`	Integer vector specifying which quarters to extract for quarterly analysis (1-4)
`biannual`	Integer vector specifying which semesters to extract for biannual analysis (1-2)
`use.parallel`	Logical indicating whether to use parallel processing for intensive operations. Default FALSE

Details

This function is essential for working with rotating panels because:

- Enables periodicity-based analysis: Extract data for different types of temporal estimations
- Preserves temporal structure: Respects temporal relationships between different panel waves
- Optimizes memory: Only loads surveys needed for the analysis
- Facilitates comparisons: Extract specific periods for comparative analysis
- Supports parallelization: For operations with large data volumes

Extraction criteria are interpreted according to survey frequency:

- For monthly ECH: monthly=c(1,3,6) extracts January, March and June
- For annual analysis: annual=1 typically extracts the first available year
- For quarterly analysis: quarterly=c(1,4) extracts Q1 and Q4

If no criteria are specified, the function returns the implantation survey with a warning.

Value

A list of `Survey` objects matching the specified criteria, or a single `Survey` object if a single index is specified

See Also

`load_panel_survey` for loading rotating panels `get_implantation` for obtaining implantation data `get_follow_up` for obtaining follow-up data `workflow` for using extracted surveys in analysis

Other panel-surveys: `PoolSurvey`, `RotativePanelSurvey`, `get_follow_up()`, `get_implantation()`

Examples

```
## Not run:
# Load rotating panel
panel_ech <- load_panel_survey(
  path = "ech_panel_2023.dta",
  svy_type = "ech_panel",
  svy_edition = "2023"
)

# Extract specific monthly surveys
ech_q1 <- extract_surveys(
  panel_ech,
  monthly = c(1, 2, 3) # January, February, March
)

# Extract by index
ech_first <- extract_surveys(panel_ech, index = 1)
ech_several <- extract_surveys(panel_ech, index = c(1, 3, 6))

# Quarterly analysis
ech_Q1_Q4 <- extract_surveys(
  panel_ech,
  quarterly = c(1, 4)
)

# Annual analysis (typically all surveys for the year)
ech_annual <- extract_surveys(
  panel_ech,
  annual = 1
)
```

```
# With parallel processing for large volumes
ech_full <- extract_surveys(
  panel_ech,
  monthly = 1:12,
  use.parallel = TRUE
)

# Use in workflow
results <- workflow(
  survey = extract_surveys(panel_ech, quarterly = c(1, 2)),
  svymean(~unemployed, na.rm = TRUE),
  estimation_type = "quarterly"
)

## End(Not run)
```

`extract_time_pattern` *Extract time pattern*

Description

Extract time pattern

Usage

```
extract_time_pattern(svy_edition)
```

Arguments

`svy_edition`	Survey edition string (e.g. "2023", "2023-06", "2023_Q1").

Value

List with components: periodicity, year, month (when applicable).

See Also

Other survey-loading: `group_dates()`, `load_panel_survey()`, `load_survey()`, `load_survey_example()`, `validate_time_pattern()`

Examples

```
# Annual edition
extract_time_pattern("2023")

# Monthly edition
extract_time_pattern("2023-06")
```

`filter_recipes` *Filter recipes by criteria*

Description

Filter recipes in the active backend by survey type, edition, category, or certification level.

Usage

```
filter_recipes(
  survey_type = NULL,
  edition = NULL,
  category = NULL,
  certification_level = NULL,
  topic = NULL
)
```

Arguments

`survey_type`	Character survey type or NULL.
`edition`	Character edition or NULL.
`category`	Character category name or NULL.
`certification_level`	Character certification level or NULL.
`topic`	Character topic name or NULL.

Value

List of matching Recipe objects.

See Also

`search_recipes`, `rank_recipes`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_category()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
ech_recipes <- filter_recipes(survey_type = "ech")
length(ech_recipes)
```

`filter_workflows` *Filter workflows by criteria*

Description

Filter workflows in the active backend by survey type, edition, recipe ID, or certification level.

Usage

```
filter_workflows(
  survey_type = NULL,
  edition = NULL,
  recipe_id = NULL,
  certification_level = NULL
)
```

Arguments

`survey_type`	Character survey type or `NULL` (default `NULL`).
`edition`	Character edition or `NULL` (default `NULL`).
`recipe_id`	Character recipe ID or `NULL` (default `NULL`). Find workflows using this recipe.
`certification_level`	Character certification level or `NULL` (default `NULL`).

Value

List of matching RecipeWorkflow objects.

See Also

`search_workflows`, `find_workflows_for_recipe`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_recipes()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_category()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
ech_wf <- filter_workflows(survey_type = "ech")
length(ech_wf)
```

find_workflows_for_recipe *Find workflows that use a specific recipe*

Description

Cross-reference query: find all workflows that reference a given recipe ID.

Usage

```
find_workflows_for_recipe(recipe_id)
```

Arguments

recipe_id	Character recipe ID to search for.

Value

List of RecipeWorkflow objects that reference this recipe.

See Also

filter_workflows

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
wfs <- find_workflows_for_recipe("recipe_001")
length(wfs)
```

get_backend *Get recipe backend*

Description

Returns the currently configured recipe backend. Defaults to "api" if not configured.

Usage

```
get_backend()
```

Value

RecipeBackend object

See Also

Other backends: get_workflow_backend(), set_backend(), set_workflow_backend()

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
backend <- get_backend()
backend
```

---

get_data *get_data*

Description

Get data from survey

Usage

```
get_data(svy)
```

Arguments

svy	Survey object

Value

A data.table (or data.frame) containing the survey microdata.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
head(get_data(svy))
```

get_engine *Get the current survey data engine*

Description

Retrieves the currently configured engine for loading surveys from system options or environment variables.

Usage

```
get_engine()
```

Value

Character vector with the name of the configured engine.

See Also

Other options: `lazy_default()`, `set_engine()`, `set_lazy_processing()`, `set_use_copy()`, `show_engines()`, `use_copy_default()`

Examples

```
get_engine()
```

get_follow_up *Get follow-up surveys from a rotating panel*

Description

Extracts one or more follow-up surveys (waves after the implantation) from a RotativePanelSurvey object. Follow-up surveys represent subsequent data collections and are essential for longitudinal and temporal change analysis.

Usage

```
get_follow_up(
  RotativePanelSurvey,
  index = seq_along(RotativePanelSurvey$follow_up)
)
```

Arguments

`RotativePanelSurvey`	A `RotativePanelSurvey` object from which to extract the follow-up surveys
`index`	Integer vector specifying which follow-up surveys to extract. Defaults to all available (1:length(follow_up)). Can be a single index or a vector of indices

Details

Follow-up surveys are fundamental in rotating panels because:

- Enable longitudinal analysis: Track the same units over time
- Capture temporal changes: Evolution of economic, social, and demographic variables
- Maintain representativeness: Each wave preserves population representativeness through controlled rotation
- Optimize resources: Reuse information from previous waves to reduce collection costs
- Facilitate comparisons: Consistent temporal structure for trend analysis

In rotating panels like ECH:

- Each follow-up wave covers a specific period (monthly/quarterly)
- Units rotate gradually maintaining temporal overlap
- Indices correspond to the chronological collection order
- Each follow-up maintains methodological consistency with implantation

Value

A list of `Survey` objects corresponding to the specified follow-up surveys. If a single index is specified, returns a list with one element

See Also

`get_implantation` for obtaining the implantation survey `extract_surveys` for extracting surveys by temporal criteria `load_panel_survey` for loading rotating panels `workflow` for analysis with follow-up surveys

Other panel-surveys: `PoolSurvey`, `RotativePanelSurvey`, `extract_surveys()`, `get_implantation()`

Examples

```
impl <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
fu1 <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023_01", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
fu2 <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023_02", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
panel <- RotativePanelSurvey$new(
  implantation = impl, follow_up = list(fu1, fu2),
  type = "test", default_engine = "data.table",
```

```
  steps = list(), recipes = list(), workflows = list(), design = NULL
)
get_follow_up(panel, index = 1)
get_follow_up(panel)
```

`get_implantation` *Get implantation survey from a rotating panel*

Description

Extracts the implantation (baseline) survey from a RotativePanelSurvey object. The implantation survey represents the first data collection wave and is essential for establishing the baseline and structural characteristics of the panel.

Usage

```
get_implantation(RotativePanelSurvey)
```

Arguments

`RotativePanelSurvey`
: A `RotativePanelSurvey` object from which to extract the implantation survey

Details

The implantation survey is special in a rotating panel because:

- Establishes the baseline: Defines initial characteristics of all panel units
- Contains the full sample: Includes all units that will participate in the different panel waves
- Defines temporal structure: Establishes rotation and follow-up patterns
- Configures metadata: Contains information about periodicity, key variables, and stratification
- Serves as tracking reference: Basis for unit tracking in subsequent waves

This function is essential for analysis requiring:

- Temporal comparisons from the baseline
- Analysis of the complete panel structure
- Configuration of longitudinal models
- Evaluation of sampling design quality

Value

A `Survey` object containing the implantation survey with all its metadata, data, and design configuration

See Also

get_follow_up for obtaining follow-up surveys extract_surveys for extracting multiple surveys by criteria load_panel_survey for loading rotating panels workflow for analysis with the implantation survey

Other panel-surveys: PoolSurvey, RotativePanelSurvey, extract_surveys(), get_follow_up()

Examples

```
impl <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
fu <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023_01", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
panel <- RotativePanelSurvey$new(
  implantation = impl, follow_up = list(fu),
  type = "test", default_engine = "data.table",
  steps = list(), recipes = list(), workflows = list(), design = NULL
)
get_implantation(panel)
```

---

get_metadata *get_metadata*

Description

Get metadata from survey

Usage

```
get_metadata(self)
```

Arguments

self	Object of class Survey

Value

NULL (called for side effect: prints metadata to console).

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
get_metadata(svy)
```

`get_recipe` *Get recipe from repository or API*

Description

This function retrieves data transformation recipes from the metasurvey repository or API, based on specific criteria such as survey type, edition, and topic. It is the primary way to access predefined and community-validated recipes.

Usage

```
get_recipe(
  svy_type = NULL,
  svy_edition = NULL,
  topic = NULL,
  allowMultiple = TRUE
)
```

Arguments

`svy_type`	String specifying the survey type. Examples: "ech", "eaii", "eai", "eph"
`svy_edition`	String specifying the survey edition. Supported formats: "YYYY", "YYYYMM", "YYYY-YYYY"
`topic`	String specifying the recipe topic. Examples: "labor_market", "poverty", "income", "demographics"
`allowMultiple`	Logical indicating whether multiple recipes are allowed. If FALSE and multiple matches exist, returns the most recent one

Details

This function is essential for:

- Accessing official recipes: Get validated and maintained recipes by specialized teams
- Reproducibility: Ensure different users apply the same standard transformations
- Automation: Integrate recipes into automatic pipelines

- Collaboration: Share methodologies between teams and organizations
- Versioning: Access different recipe versions according to edition

The function queries the metasurvey API to retrieve recipes. **Internet connection is required**. If the API is unavailable or you need to work offline:

Working Offline:

- Don't call `get_recipe()` - work directly with steps
- Set `options(metasurvey.skip_recipes = TRUE)` to disable API calls
- Load recipes from local files using `read_recipe()`
- Create custom recipes with `recipe()`

Search criteria are combined with AND operator, so all specified criteria must match for a recipe to be returned.

Value

`Recipe` object or list of `Recipe` objects according to the specified criteria and the value of `allowMultiple`

See Also

`recipe` to create custom recipes `save_recipe` to save recipes locally `read_recipe` to read recipes from file `publish_recipe` to publish recipes to the repository `load_survey` where recipes are used

Other recipes: `Recipe-class`, `add_recipe()`, `bake_recipes()`, `explore_recipes()`, `harmonize()`, `print.Recipe()`, `publish_recipe()`, `read_recipe()`, `recipe()`, `save_recipe()`, `steps_to_recipe()`

Examples

```
## Not run:
# Get specific recipe for ECH 2023
ech_recipe <- get_recipe(
  svy_type = "ech",
  svy_edition = "2023"
)

# Recipe for specific topic
labor_recipe <- get_recipe(
  svy_type = "ech",
  svy_edition = "2023",
  topic = "labor_market"
)

# Allow multiple recipes
available_recipes <- get_recipe(
  svy_type = "eaii",
  svy_edition = "2019-2021",
  allowMultiple = TRUE
)

# Use recipe in load_survey
```

```
ech_with_recipe <- load_survey(
  path = "ech_2023.dta",
  svy_type = "ech",
  svy_edition = "2023",
  recipes = get_recipe("ech", "2023"),
  bake = TRUE
)

# Working offline - don't use recipes
ech_offline <- load_survey(
  path = "ech_2023.dta",
  svy_type = "ech",
  svy_edition = "2023",
  svy_weight = add_weight(annual = "PESOANO")
)

# Disable recipe API globally
options(metasurvey.skip_recipes = TRUE)
# Now get_recipe() will return NULL with a warning

# For year ranges
panel_recipe <- get_recipe(
  svy_type = "ech_panel",
  svy_edition = "2020-2023"
)

## End(Not run)
```

`get_steps` *get_steps*

Description

Get steps from survey

Usage

```
get_steps(svy)
```

Arguments

svy	Survey object

Value

List of Step objects

See Also

Other steps: bake_steps(), step_compute(), step_filter(), step_join(), step_recode(), step_remove(), step_rename(), step_validate(), view_graph()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy <- step_compute(svy, age2 = age * 2)
get_steps(svy) # list of Step objects
```

---

get_workflow_backend *Get workflow backend*

Description

Returns the currently configured workflow backend. Defaults to "local" if not configured.

Usage

```
get_workflow_backend()
```

Value

WorkflowBackend object

See Also

Other backends: get_backend(), set_backend(), set_workflow_backend()

Examples

```
backend <- get_workflow_backend()
```

group_dates *Group dates*

Description

Group dates

Usage

```
group_dates(dates, type = c("monthly", "quarterly", "biannual"))
```

Arguments

dates	Vector of Date objects.
type	Grouping type: "monthly", "quarterly", or "biannual".

Value

Integer vector of group indices (e.g. 1-12 for monthly, 1-4 for quarterly).

See Also

Other survey-loading: extract_time_pattern(), load_panel_survey(), load_survey(), load_survey_example(), validate_time_pattern()

Examples

```
dates <- as.Date(c(
  "2023-01-15", "2023-04-20",
  "2023-07-10", "2023-11-05"
))
group_dates(dates, "quarterly")
group_dates(dates, "biannual")
```

harmonize *Harmonize surveys using registry recipes*

Description

For each survey in the list, fetches matching recipes from the active backend (by survey type and edition), sorts them by dependency order, applies them via bake_recipes, and assembles the results into a PoolSurvey ready for workflow.

Usage

```
harmonize(
  surveys,
  survey_type = NULL,
  topic = NULL,
  category = NULL,
  certification_level = NULL,
  grouping = "annual",
  group_name = "series",
  .verbose = getOption("metasurvey.verbose", TRUE)
)
```

Arguments

surveys	List of Survey objects to harmonize.
survey_type	Character. Override survey type for recipe lookup. If NULL (default), uses each survey's own type field.
topic	Character. Filter recipes by topic (e.g. "compatibilizada"). Default NULL (no filter).
category	Character. Optional category filter for recipes.
certification_level	Character. Optional certification level filter (e.g. "official", "reviewed").
grouping	Character. Time hierarchy for the PoolSurvey structure. One of "annual" (default), "quarterly", "monthly", "biannual".
group_name	Character. Name of the group in the PoolSurvey structure (default "series").
.verbose	Logical. Print progress messages (default uses metasurvey.verbose option).

Details

The function performs the following steps for each survey:

1. Calls filter_recipes with the survey's type and edition (plus optional category/certification filters).
2. Sorts matching recipes using topological sort on depends_on_recipes.
3. Attaches recipes via add_recipe and applies them with bake_recipes.

If no recipes are found for a survey, it is included in the pool unchanged (with a warning).

Value

A PoolSurvey object containing the harmonized surveys, structured as list(<grouping> = list(<group_name> = list(svy1, svy2, ...))).

See Also

filter_recipes, bake_recipes, PoolSurvey, workflow

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), print.Recipe(), publish_recipe(), read_recipe(), recipe(), save_recipe(), steps_to_recipe()

Examples

```
# Set up a local backend with a temp file
old <- set_backend("local", path = tempfile(fileext = ".json"))

# Create a small survey and a recipe
dt <- data.table::data.table(
  id = 1:10, x = 1:10, w = rep(1, 10)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
r <- Recipe$new(
  name = "demo", edition = "2023", survey_type = "test",
  default_engine = "data.table", depends_on = list(),
  user = "demo", description = "demo recipe",
  steps = list("step_compute(., z = x * 2)"),
  id = "demo_recipe"
)
publish_recipe(r)

pool <- harmonize(list(svy), .verbose = FALSE)

# Restore backend
options(metasurvey.backend = old)
```

---

has_design *Check if survey has a design*

Description

Check if survey has a design

Usage

```
has_design(svy)
```

Arguments

svy	A Survey object.

Value

Logical.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
svy <- survey_empty(type = "test", edition = "2023")
has_design(svy) # FALSE
```

---

has_recipes *Check if survey has recipes*

Description

Check if survey has recipes

Usage

```
has_recipes(svy)
```

Arguments

svy	A Survey or RotativePanelSurvey object.

Value

Logical.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
svy <- survey_empty(type = "test", edition = "2023")
has_recipes(svy) # FALSE
```

has_steps *Check if survey has steps*

Description

Check if survey has steps

Usage

```
has_steps(svy)
```

Arguments

svy	A Survey or RotativePanelSurvey object.

Value

Logical.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
svy <- survey_empty(type = "test", edition = "2023")
has_steps(svy) # FALSE
```

is_baked *Check if all steps are baked*

Description

Returns TRUE when every step attached to the survey has been executed (bake == TRUE), or when there are no steps.

Usage

```
is_baked(svy)
```

Arguments

svy	A Survey or RotativePanelSurvey object.

Value

Logical.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
svy <- survey_empty(type = "test", edition = "2023")
is_baked(svy) # TRUE (no steps)
```

---

lazy_default *Lazy processing*

Description

Lazy processing

Usage

```
lazy_default()
```

Value

Logical indicating the current lazy processing setting.

See Also

Other options: get_engine(), set_engine(), set_lazy_processing(), set_use_copy(), show_engines(), use_copy_default()

Examples

```
# Check current lazy processing default
lazy_default()
```

list_recipes *List all recipes*

Description

List all recipes from the active backend.

Usage

```
list_recipes()
```

Value

List of all Recipe objects.

See Also

search_recipes, filter_recipes

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
all <- list_recipes()
length(all)
```

list_workflows *List all workflows*

Description

List all workflows from the active workflow backend.

Usage

```
list_workflows()
```

Value

List of all RecipeWorkflow objects.

See Also

search_workflows, filter_workflows

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
all <- list_workflows()
length(all)
```

---

load_panel_survey *Read panel survey files from different formats and create a RotativePanelSurvey object*

Description

Read panel survey files from different formats and create a RotativePanelSurvey object

Usage

```
load_panel_survey(
  path_implantation,
  path_follow_up,
  svy_type,
  svy_weight_implantation,
  svy_weight_follow_up,
  svy_strata = NULL,
  ...
)
```

Arguments

path_implantation	Survey implantation path, file can be in different formats, csv, xtsx, dta, sav and rds
path_follow_up	Path with all the needed files with only survey valid files but also can be character vector with path files.
svy_type	String with the survey type, supported types; "ech" (Encuensta Continua de Hogares, Uruguay), "eph" (Encuesta Permanente de Hogares, Argentina), "eai" (Encuesta de Actividades de Innovacion, Uruguay)

svy_weight_implantation
: List with survey implantation weights information specifing periodicity and the name of the weight variable. Recomended to use the helper function add_weight().

svy_weight_follow_up
: List with survey follow_up weights information specifing periodicity and the name of the weight variable. Recomended to use the helper function add_weight().

svy_strata
: Stratification variable name (character or NULL). Passed to Survey$new(strata = ...).

...
: Further arguments to be passed to load_panel_survey

Value

RotativePanelSurvey object

See Also

Other survey-loading: `extract_time_pattern()`, `group_dates()`, `load_survey()`, `load_survey_example()`, `validate_time_pattern()`

Examples

```
## Not run:
# example code
path_dir <- here::here("example-data", "ech", "ech_2023")
ech_2023 <- load_panel_survey(
  path_implantation = file.path(
    path_dir,
    "ECH_implantacion_2023.csv"
  ),
  path_follow_up = file.path(
    path_dir,
    "seguimiento"
  ),
  svy_type = "ECH_2023",
  svy_weight_implantation = add_weight(
    annual = "W_ANO"
  ),
  svy_weight_follow_up = add_weight(
    monthly = add_replicate(
      "W",
      replicate_path = file.path(
        path_dir,
        c(
          "Pesos replicados Bootstrap mensuales enero_junio 2023",
          "Pesos replicados Bootstrap mensuales julio_diciembre 2023"
        ),
        c(
          "Pesos replicados mensuales enero_junio 2023",
          "Pesos replicados mensuales Julio_diciembre 2023"
        )
      ),
```

```
      replicate_id = c("ID" = "ID"),
      replicate_pattern = "wr[0-9]+",
      replicate_type = "bootstrap"
    )
  )
)

## End(Not run)
## Not run:
# Example of loading a panel survey
panel_survey <- load_panel_survey(
  path_implantation = "path/to/implantation.csv",
  path_follow_up = "path/to/follow_up",
  svy_type = "ech",
  svy_weight_implantation = add_weight(annual = "w_ano"),
  svy_weight_follow_up = add_weight(monthly = "w_monthly")
)
print(panel_survey)

## End(Not run)
```

`load_survey` *Load survey from file and create Survey object*

Description

This function reads survey files in multiple formats and creates a Survey object with all necessary metadata for subsequent analysis. Supports various survey types with specific configurations for each one.

Usage

```
load_survey(
  path = NULL,
  svy_type = NULL,
  svy_edition = NULL,
  svy_weight = NULL,
  svy_psu = NULL,
  svy_strata = NULL,
  ...,
  bake = FALSE,
  recipes = NULL
)
```

Arguments

`path`	Path to the survey file. Supports multiple formats: csv, xlsx, dta (Stata), sav (SPSS), rds (R). If NULL, survey arguments must be specified to create an empty object

svy_type	Survey type as string. Supported types: • "ech": Encuesta Continua de Hogares (Uruguay) • "eph": Encuesta Permanente de Hogares (Argentina) • "eai": Encuesta de Actividades de Innovación (Uruguay) • "eaii": Encuesta de Actividades de Innovación e I+D (Uruguay)
svy_edition	Survey edition as string. Supports different temporal patterns: • "YYYY": Year (e.g., "2023") • "YYYYMM" or "MMYYYY": Year-month (e.g., "202301" or "012023") • "YYYY-YYYY": Year range (e.g., "2020-2022")
svy_weight	List with weight information specifying periodicity and weight variable name. Use helper function add_weight
svy_psu	Primary sampling unit (PSU) variable as string
svy_strata	Stratification variable name as string (optional). Used in survey::svydesign() for stratified sampling designs.
...	Additional arguments passed to specific reading functions
bake	Logical indicating whether recipes are processed automatically when loading data. Defaults to FALSE
recipes	Recipe object obtained with get_recipe. If bake=TRUE, these recipes are applied automatically

Details

The function automatically detects file format and uses the appropriate reader. For each survey type, it applies specific configurations such as standard variables, data types, and validations.

When bake=TRUE is specified, recipes are applied immediately after loading the data, creating an analysis-ready object.

If no path is provided, an empty Survey object is created that can be used to build step pipelines without initial data.

Value

Survey object with structure:

- data: Survey data
- metadata: Information about type, edition, weights
- steps: History of applied transformations
- recipes: Available recipes
- design: Sample design information

See Also

add_weight to specify weights get_recipe to get available recipes load_survey_example to load example data load_panel_survey for panel surveys

Other survey-loading: extract_time_pattern(), group_dates(), load_panel_survey(), load_survey_example(), validate_time_pattern()

Examples

```
## Not run:
# Load ECH 2023 with recipes
ech_2023 <- load_survey(
  path = "data/ech_2023.csv",
  svy_type = "ech",
  svy_edition = "2023",
  svy_weight = add_weight(annual = "pesoano"),
  recipes = get_recipe("ech", "2023"),
  bake = TRUE
)

# Load monthly survey
ech_january <- load_survey(
  path = "data/ech_202301.dta",
  svy_type = "ech",
  svy_edition = "202301",
  svy_weight = add_weight(monthly = "pesomes")
)

# Create empty object for pipeline
pipeline <- load_survey(
  svy_type = "ech",
  svy_edition = "2023"
) %>%
  step_compute(new_var = operation)

# With included example data
ech_example <- load_survey(
  path = load_survey_example("ech", "ech_2022"),
  svy_type = "ech",
  svy_edition = "2022",
  svy_weight = add_weight(annual = "pesoano")
)

## End(Not run)
```

`load_survey_example` *Load survey example data*

Description

Downloads and loads example survey data from the metasurvey data repository. This function provides access to sample datasets for testing and demonstration purposes, including ECH (Continuous Household Survey) and other survey types.

Usage

```
load_survey_example(svy_type, svy_edition)
```

Arguments

svy_type	Character string specifying the survey type (e.g., "ech")
svy_edition	Character string specifying the survey edition/year (e.g., "2023")

Details

This function downloads example data from the official metasurvey data repository on GitHub. The data is cached locally in a temporary file to avoid repeated downloads in the same session.

Available survey types and editions can be found at: https://github.com/metasurveyr/metasurvey_data

Value

Character string with the path to the downloaded CSV file containing the example survey data

See Also

load_survey for loading the downloaded data

Other survey-loading: extract_time_pattern(), group_dates(), load_panel_survey(), load_survey(), validate_time_pattern()

Examples

```
## Not run:
# Load ECH 2023 example data
ech_path <- load_survey_example("ech", "2023")

# Use with load_survey
ech_data <- load_survey(
  path = load_survey_example("ech", "2023"),
  svy_type = "ech",
  svy_edition = "2023"
)

## End(Not run)
```

---

parse_do_file *Parse a STATA .do file into structured commands*

Description

Reads a .do file and returns a list of parsed command objects. Handles comment stripping, line continuation, loop expansion, and command tokenization.

Usage

```
parse_do_file(do_file, encoding = "latin1")
```

Arguments

do_file	Path to a STATA .do file
encoding	File encoding (default "latin1" for legacy STATA files)

Value

A list of StataCommand lists, each with fields: cmd, args, if_clause, options, raw_line, line_num, capture

See Also

Other transpiler: parse_stata_labels(), transpile_coverage(), transpile_stata(), transpile_stata_module()

Examples

```
tf <- tempfile(fileext = ".do")
writeLines(c("gen age2 = edad^2", "replace sexo = 1 if sexo == ."), tf)
cmds <- parse_do_file(tf)
length(cmds)
cmds[[1]]$cmd
```

---

parse_stata_labels *Parse STATA label commands from source lines*

Description

Extracts variable labels, value label definitions, and value label assignments from label commands.

Usage

```
parse_stata_labels(lines)
```

Arguments

lines	Character vector of source lines (already comment-stripped)

Value

A list with var_labels (named list) and val_labels (named list of named lists)

See Also

Other transpiler: parse_do_file(), transpile_coverage(), transpile_stata(), transpile_stata_module()

Examples

```
lines <- c(
  'label variable edad "Age in years"',
  'label define sexo_lbl 1 "Male" 2 "Female"',
  "label values sexo sexo_lbl"
)
labels <- parse_stata_labels(lines)
labels$var_labels
labels$val_labels
```

`PoolSurvey` *PoolSurvey Class*

Description

This class represents a collection of surveys grouped by specific periods (e.g., monthly, quarterly, annual). It provides methods to access and manipulate the grouped surveys.

Value

An object of class `PoolSurvey`.

Public fields

`surveys` A list containing the grouped surveys.

Methods

Public methods:

- `PoolSurvey$new()`
- `PoolSurvey$get_surveys()`
- `PoolSurvey$print()`
- `PoolSurvey$clone()`

Method `new()`**:** Initializes a new instance of the PoolSurvey class.

Usage:

```
PoolSurvey$new(surveys)
```

Arguments:

`surveys` A list containing the grouped surveys.

Method `get_surveys()`**:** Retrieves surveys for a specific period.

Usage:

```
PoolSurvey$get_surveys(period = NULL)
```

Arguments:

`period` A string specifying the period to retrieve (e.g., "monthly", "quarterly").

Returns: A list of surveys for the specified period.

Method `print()`**:** Prints metadata about the PoolSurvey object.

Usage:

```
PoolSurvey$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PoolSurvey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other panel-surveys: `RotativePanelSurvey`, `extract_surveys()`, `get_follow_up()`, `get_implantation()`

Examples

```
s1 <- Survey$new(
  data = data.table::data.table(id = 1:3, w = 1),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
s2 <- Survey$new(
  data = data.table::data.table(id = 4:6, w = 1),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
pool <- PoolSurvey$new(list(annual = list("group1" = list(s1, s2))))
```

---

print.metasurvey_provenance

Print provenance information

Description

Print provenance information

Usage

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

Arguments

x	A metasurvey_provenance list.
...	Additional arguments (unused).

Value

Invisibly returns x.

See Also

Other provenance: print.metasurvey_provenance_diff(), provenance(), provenance_diff(), provenance_to_json()

Examples

```
s <- survey_empty("ech", "2023")
s <- set_data(s, data.table::data.table(age = 18:65))
s <- step_compute(s, age2 = age * 2)
s <- bake_steps(s)
print(provenance(s))
```

---

print.metasurvey_provenance_diff

Print provenance diff

Description

Print provenance diff

Usage

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

Arguments

x	A metasurvey_provenance_diff list.
...	Additional arguments (unused).

Value

Invisibly returns x.

See Also

Other provenance: print.metasurvey_provenance(), provenance(), provenance_diff(), provenance_to_json()

Examples

```
s1 <- survey_empty("ech", "2022")
s1 <- set_data(s1, data.table::data.table(age = 18:65))
s1 <- step_compute(s1, age2 = age * 2)
s1 <- bake_steps(s1)

s2 <- survey_empty("ech", "2023")
s2 <- set_data(s2, data.table::data.table(age = 20:70))
s2 <- step_compute(s2, age2 = age * 2)
s2 <- bake_steps(s2)

diff_result <- provenance_diff(provenance(s1), provenance(s2))
print(diff_result)
```

---

print.Recipe *Print method for Recipe objects*

Description

Displays a formatted recipe card showing metadata, required variables, pipeline steps, and produced variables.

Usage

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

Arguments

x	A Recipe object
...	Additional arguments (currently unused)

Value

Invisibly returns the Recipe object

See Also

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), publish_recipe(), read_recipe(), recipe(), save_recipe(), steps_to_recipe()

Examples

```
rec <- Recipe$new(
  id = "r1", name = "Example", user = "tester",
  edition = "2023", survey_type = "test",
  default_engine = "data.table", depends_on = list(),
  description = "Demo recipe", steps = list()
)
print(rec)
```

`print.RecipeWorkflow` *Print method for RecipeWorkflow objects*

Description

Print method for RecipeWorkflow objects

Usage

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

Arguments

`x`	A RecipeWorkflow object
`...`	Additional arguments (unused)

Value

Invisibly returns the object

See Also

Other workflows: `RecipeWorkflow-class`, `evaluate_cv()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `save_workflow()`, `workflow()`, `workflow_from_list()`, `workflow_table()`

Examples

```
wf <- RecipeWorkflow$new(
  id = "w1", name = "Example Workflow", user = "tester",
  edition = "2023", survey_type = "test",
  recipe_ids = "r1",
  calls = list(), description = "Demo"
)
print(wf)
```

provenance *Get provenance from a survey or workflow result*

Description

Returns the provenance metadata recording the full data lineage: source file, step history with row counts, and environment info.

Usage

```
provenance(x, ...)

## S3 method for class 'Survey'
provenance(x, ...)

## S3 method for class 'data.table'
provenance(x, ...)

## Default S3 method:
provenance(x, ...)
```

Arguments

x	A Survey object or a data.table from workflow().
...	Additional arguments (unused).

Value

A metasurvey_provenance list, or NULL if no provenance is available.

See Also

Other provenance: print.metasurvey_provenance(), print.metasurvey_provenance_diff(), provenance_diff(), provenance_to_json()

Examples

```
svy <- Survey$new(
  data = data.table::data.table(id = 1:10, age = 20:29, w = 1),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
provenance(svy)
```

`provenance_diff` *Compare two provenance objects*

Description

Shows differences between two provenance records, useful for comparing processing across survey editions.

Usage

```
provenance_diff(prov1, prov2)
```

Arguments

`prov1`	First provenance list.
`prov2`	Second provenance list.

Value

A `metasurvey_provenance_diff` list with detected differences.

See Also

Other provenance: `print.metasurvey_provenance()`, `print.metasurvey_provenance_diff()`, `provenance()`, `provenance_to_json()`

Examples

```
svy1 <- Survey$new(
  data = data.table::data.table(id = 1:5, w = rep(1, 5)),
  edition = "2023", type = "test",
  engine = "data.table", weight = add_weight(annual = "w")
)
svy2 <- Survey$new(
  data = data.table::data.table(id = 1:5, w = rep(2, 5)),
  edition = "2024", type = "test",
  engine = "data.table", weight = add_weight(annual = "w")
)
provenance_diff(provenance(svy1), provenance(svy2))
```

provenance_to_json *Export provenance to JSON*

Description

Serializes a provenance object to JSON format, optionally writing to a file.

Usage

```
provenance_to_json(prov, path = NULL)
```

Arguments

prov	A metasurvey_provenance list.
path	File path to write JSON. If NULL, returns the JSON string.

Value

JSON string (invisibly if path is provided).

See Also

Other provenance: print.metasurvey_provenance(), print.metasurvey_provenance_diff(), provenance(), provenance_diff()

Examples

```
svy <- Survey$new(
  data = data.table::data.table(id = 1:5, w = rep(1, 5)),
  edition = "2023", type = "test",
  engine = "data.table", weight = add_weight(annual = "w")
)
prov <- provenance(svy)
provenance_to_json(prov)
```

publish_recipe *Publish Recipe*

Description

Publishes a Recipe object to the active backend (local JSON registry or remote API).

Usage

```
publish_recipe(recipe)
```

Arguments

recipe	A Recipe object.

Value

The Recipe object (invisibly).

See Also

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), read_recipe(), recipe(), save_recipe(), steps_to_recipe()

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
publish_recipe(r)
length(list_recipes())
```

---

publish_workflow *Publish a workflow to the active backend*

Description

Publishes a RecipeWorkflow to the configured workflow backend.

Usage

```
publish_workflow(wf)
```

Arguments

wf	A RecipeWorkflow object.

Value

NULL (called for side effect).

See Also

set_workflow_backend, RecipeWorkflow

Other workflows: RecipeWorkflow-class, evaluate_cv(), print.RecipeWorkflow(), read_workflow(), reproduce_workflow(), save_workflow(), workflow(), workflow_from_list(), workflow_table()

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
wf <- RecipeWorkflow$new(
  name = "Example", description = "Test",
  survey_type = "ech", edition = "2023",
  recipe_ids = "r_001", estimation_type = "svymean"
)
publish_workflow(wf)
```

`rank_recipes` *Rank recipes by downloads*

Description

Get the top recipes ranked by download count from the active backend.

Usage

```
rank_recipes(n = NULL)
```

Arguments

`n`	Integer. Maximum number of recipes to return, or NULL for all.

Value

List of Recipe objects sorted by downloads (descending).

See Also

`search_recipes`, `filter_recipes`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_recipes()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_workflows()`, `recipe_category()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
top10 <- rank_recipes(n = 10)
```

rank_workflows *Rank workflows by downloads*

Description

Get the top workflows ranked by download count.

Usage

```
rank_workflows(n = NULL)
```

Arguments

n	Integer or NULL (default NULL). Maximum number to return, or NULL for all.

Value

List of RecipeWorkflow objects sorted by downloads.

See Also

search_workflows, filter_workflows

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
top5 <- rank_workflows(n = 5)
length(top5)
```

read_recipe *Read Recipe*

Description

Reads a Recipe object from a JSON file.

Usage

```
read_recipe(file)
```

Arguments

file	A character string specifying the file path.

Details

This function reads a JSON file and decodes it into a Recipe object.

Value

A Recipe object.

See Also

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), publish_recipe(), recipe(), save_recipe(), steps_to_recipe()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
f <- tempfile(fileext = ".json")
save_recipe(r, f)
r2 <- read_recipe(f)
r2
```

---

read_workflow *Read a RecipeWorkflow from a JSON file*

Description

Read a RecipeWorkflow from a JSON file

Usage

```
read_workflow(file)
```

Arguments

file	Character file path

Value

A RecipeWorkflow object

See Also

Other workflows: RecipeWorkflow-class, evaluate_cv(), print.RecipeWorkflow(), publish_workflow(), reproduce_workflow(), save_workflow(), workflow(), workflow_from_list(), workflow_table()

Examples

```
wf <- RecipeWorkflow$new(
  name = "Example", description = "Test",
  survey_type = "ech", edition = "2023",
  recipe_ids = "r_001", estimation_type = "svymean"
)
f <- tempfile(fileext = ".json")
save_workflow(wf, f)
wf2 <- read_workflow(f)
```

---

recipe *Create a survey data transformation recipe*

Description

Creates a Recipe object that encapsulates a sequence of data transformations that can be applied to surveys in a reproducible manner. Recipes allow documenting, sharing, and reusing data processing workflows.

Usage

```
recipe(...)
```

Arguments

...	Required metadata and optional steps. Required parameters: • name: Descriptive name for the recipe • user: User/author creating the recipe • svy: Base Survey object (use survey_empty() for generic recipes) • description: Detailed description of the recipe's purpose Optional parameters include data transformation steps.

Details

Recipes are essential for:

- Reproducibility: Ensure transformations are applied consistently
- Documentation: Keep a record of what transformations are performed and why
- Collaboration: Share workflows between users and teams
- Versioning: Maintain different processing versions for different editions
- Automation: Apply complex transformations automatically

Steps included in the recipe can be any combination of step_compute, step_recode, or other transformation steps.

Recipes can be saved with save_recipe(), loaded with read_recipe(), and applied automatically with bake_recipes().

Value

A Recipe object containing metadata, transformation steps, dependency information, and default engine configuration.

See Also

Recipe for class definition save_recipe to save recipes read_recipe to load recipes get_recipe to retrieve recipes from repository bake_recipes to apply recipes to data

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), publish_recipe(), read_recipe(), save_recipe(), steps_to_recipe()

Examples

```
# Basic recipe without steps
r <- recipe(
  name = "Basic ECH Indicators",
  user = "Analyst",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Basic labor indicators for ECH 2023"
)
r


# Recipe with steps using local data
dt <- data.table::data.table(
  id = 1:50, age = sample(18:65, 50, TRUE),
  income = runif(50, 1000, 5000), w = runif(50, 0.5, 2)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "demo",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
svy <- svy |>
  step_compute(income_cat = ifelse(income > 3000, "high", "low")) |>
  step_recode(age_group, age < 30 ~ "young", .default = "adult")
r2 <- recipe(
  name = "Demo", user = "test", svy = svy,
  description = "Demo recipe", steps = get_steps(svy)
)
r2
```

`Recipe-class` *Recipe R6 class*

Description

Recipe R6 class

Recipe R6 class

Format

An R6 class generator (R6ClassGenerator)

Details

R6 class representing a reproducible data transformation recipe for surveys. It encapsulates metadata, declared dependencies, and a list of transformation steps to be applied to a Survey object.

Value

An object of class `Recipe`.

Methods

$new(name, edition, survey_type, default_engine, depends_on, user, description, steps, id, doi, topic) Class constructor.

$doc() Auto-generate documentation from recipe steps. Returns a list with metadata, input_variables, output_variables, and pipeline information.

$validate(svy) Validate that a survey object has all required input variables.

Public fields

`name` Descriptive name of the recipe (character).

`edition` Target edition/period (character or Date).

`survey_type` Survey type (character), e.g., "ech", "eaii".

`default_engine` Default evaluation engine (character).

`depends_on` Vector/list of dependencies declared by the steps.

`user` Author/owner (character).

`description` Recipe description (character).

`id` Unique identifier (character/numeric).

`steps` List of step calls that make up the workflow.

`doi` DOI or external identifier (character|NULL).

`bake` Logical flag indicating whether it has been applied.

`topic` Recipe topic (character|NULL).

`step_objects` List of Step R6 objects (list|NULL), used for documentation generation.

`categories` List of RecipeCategory objects for classification.

`downloads` Integer download/usage count.

`certification` RecipeCertification object (default community).

`user_info` RecipeUser object or NULL.

`version` Recipe version string.

`depends_on_recipes` List of recipe IDs that must be applied before this one.

`data_source` List with S3 bucket info (s3_bucket, s3_prefix, file_pattern, provider) or NULL.

`labels` List with variable and value labels (var_labels, val_labels) or NULL.

Methods

Public methods:

- `Recipe$new()`
- `Recipe$increment_downloads()`
- `Recipe$certify()`
- `Recipe$add_category()`
- `Recipe$remove_category()`
- `Recipe$to_list()`
- `Recipe$doc()`
- `Recipe$validate()`
- `Recipe$clone()`

Method `new()`**:** Create a Recipe object

Usage:

```
Recipe$new(
  name,
  edition,
  survey_type,
  default_engine,
  depends_on,
  user,
  description,
  steps,
  id,
  doi = NULL,
  topic = NULL,
  step_objects = NULL,
  cached_doc = NULL,
  categories = list(),
  downloads = 0L,
  certification = NULL,
  user_info = NULL,
  version = "1.0.0",
```

```
  depends_on_recipes = list(),
  data_source = NULL,
  labels = NULL
)
```

Arguments:

`name` Descriptive name of the recipe (character)
`edition` Target edition/period (character or Date)
`survey_type` Survey type (character), e.g., "ech", "eaii"
`default_engine` Default evaluation engine (character)
`depends_on` Vector or list of declared dependencies
`user` Author or owner of the recipe (character)
`description` Detailed description of the recipe (character)
`steps` List of step calls that make up the workflow
`id` Unique identifier (character or numeric)
`doi` DOI or external identifier (character or NULL)
`topic` Recipe topic (character or NULL)
`step_objects` List of Step R6 objects (optional, used for doc generation)
`cached_doc` Pre-computed documentation (optional, used when loading from JSON)
`categories` List of RecipeCategory objects (optional)
`downloads` Integer download count (default 0)
`certification` RecipeCertification object (optional, default community)
`user_info` RecipeUser object (optional)
`version` Recipe version string (default "1.0.0")
`depends_on_recipes` List of recipe IDs that must be applied before this one (optional)
`data_source` List with S3 bucket info (optional)
`labels` List with var_labels and val_labels (optional)

Method `increment_downloads()`**:** Increment the download counter

Usage:

```
Recipe$increment_downloads()
```

Method `certify()`**:** Certify the recipe at a given level

Usage:

```
Recipe$certify(user, level)
```

Arguments:

`user` RecipeUser who is certifying
`level` Character certification level ("reviewed" or "official")

Method `add_category()`**:** Add a category to the recipe

Usage:

```
Recipe$add_category(category)
```

Arguments:

`category` RecipeCategory to add

Method `remove_category()`**:** Remove a category by name

Usage:

```
Recipe$remove_category(name)
```

Arguments:

`name` Character category name to remove

Method `to_list()`**:** Serialize Recipe to a plain list suitable for JSON/API publishing. Steps are encoded as character strings via deparse().

Usage:

```
Recipe$to_list()
```

Returns: A named list with all recipe fields.

Method `doc()`**:** Auto-generate documentation from recipe steps

Usage:

```
Recipe$doc()
```

Returns: A list with metadata, input_variables, output_variables, and pipeline information

Method `validate()`**:** Validate that a survey has all required input variables

Usage:

```
Recipe$validate(svy)
```

Arguments:

`svy` A Survey object

Returns: TRUE if valid, otherwise stops with error listing missing variables

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Recipe$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`recipe`, `save_recipe`, `read_recipe`, `bake_recipes`

Other recipes: `add_recipe()`, `bake_recipes()`, `explore_recipes()`, `get_recipe()`, `harmonize()`, `print.Recipe()`, `publish_recipe()`, `read_recipe()`, `recipe()`, `save_recipe()`, `steps_to_recipe()`

Examples

```
# Use the recipe() constructor:
svy <- survey_empty(type = "ech", edition = "2023")
r <- recipe(
  name = "Example", user = "Test", svy = svy,
  description = "Example recipe"
)
```

RecipeCategory *RecipeCategory*

Description

Standardized taxonomy for classifying recipes by domain. Supports hierarchical categories with parent-child relationships.

Value

An object of class `RecipeCategory`.

Methods

$new(name, description, parent) Constructor for creating a new category

$is_subcategory_of(ancestor_name) Check if this category is a subcategory of another

$get_path() Get full hierarchical path

$equals(other) Check equality by name

$to_list() Serialize to list for JSON

$print(...) Print category information

$from_list(lst) Class method to reconstruct from list (see details)

Public fields

`name` Character. Category identifier.

`description` Character. Human-readable description.

`parent` RecipeCategory or NULL. Parent category for hierarchy.

Methods

Public methods:

- `RecipeCategory$new()`
- `RecipeCategory$is_subcategory_of()`
- `RecipeCategory$get_path()`
- `RecipeCategory$equals()`
- `RecipeCategory$to_list()`
- `RecipeCategory$print()`
- `RecipeCategory$from_list()`
- `RecipeCategory$clone()`

Method `new()`**:** Create a new RecipeCategory

Usage:

```
RecipeCategory$new(name, description, parent = NULL)
```

Arguments:
`name` Character. Category identifier (non-empty string).
`description` Character. Description of the category.
`parent` RecipeCategory or NULL. Parent category.

Method `is_subcategory_of()`**:** Check if this category is a subcategory of another
Usage:
```
RecipeCategory$is_subcategory_of(ancestor_name)
```
Arguments:
`ancestor_name` Character. Name of the potential ancestor category.
Returns: Logical

Method `get_path()`**:** Get full hierarchical path
Usage:
```
RecipeCategory$get_path()
```
Returns: Character string with slash-separated path

Method `equals()`**:** Check equality by name
Usage:
```
RecipeCategory$equals(other)
```
Arguments:
`other` RecipeCategory to compare with.
Returns: Logical

Method `to_list()`**:** Serialize to list for JSON
Usage:
```
RecipeCategory$to_list()
```
Returns: List representation

Method `print()`**:** Print category
Usage:
```
RecipeCategory$print(...)
```
Arguments:
`...` Additional arguments (not used)

Method `from_list()`**:** Deserialize a RecipeCategory from a list
Usage:
```
RecipeCategory$from_list(lst)
```
Arguments:
`lst` List with name, description, parent fields, or NULL
Returns: RecipeCategory object or NULL

Method `clone()`**:** The objects of this class are cloneable with this method.
Usage:
```
RecipeCategory$clone(deep = FALSE)
```
Arguments:
`deep` Whether to make a deep clone.

See Also

Other tidy-api: RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
# Use recipe_category() for the public API:
cat <- recipe_category(
  "economics", "Economic indicators"
)
sub <- recipe_category(
  "labor_market", "Labor market",
  parent = "economics"
)
```

RecipeCertification *RecipeCertification*

Description

Quality certification for recipes with three tiers: community (default), reviewed (peer-reviewed by institutional member), and official (certified by institution).

Value

An object of class `RecipeCertification`.

Public fields

`level` Character. Certification level.

`certified_by` RecipeUser or NULL. The certifying user.

`certified_at` POSIXct. Timestamp of certification.

`notes` Character or NULL. Additional notes.

Methods

Public methods:

- RecipeCertification$new()
- RecipeCertification$numeric_level()
- RecipeCertification$is_at_least()
- RecipeCertification$to_list()
- RecipeCertification$print()

- RecipeCertification$clone()

Method `new()`**:** Create a new RecipeCertification

Usage:

```
RecipeCertification$new(
  level,
  certified_by = NULL,
  notes = NULL,
  certified_at = NULL
)
```

Arguments:

`level` Character. One of "community", "reviewed", "official".

`certified_by` RecipeUser or NULL. Required for reviewed/official.

`notes` Character or NULL. Additional notes.

`certified_at` POSIXct or NULL. Auto-set if NULL.

Method `numeric_level()`**:** Get numeric level for ordering (1=community, 2=reviewed, 3=official)

Usage:

```
RecipeCertification$numeric_level()
```

Returns: Integer

Method `is_at_least()`**:** Check if certification is at least a given level

Usage:

```
RecipeCertification$is_at_least(level)
```

Arguments:

`level` Character. Level to compare against.

Returns: Logical

Method `to_list()`**:** Serialize to list for JSON

Usage:

```
RecipeCertification$to_list()
```

Returns: List representation

Method `print()`**:** Print certification badge

Usage:

```
RecipeCertification$print(...)
```

Arguments:

`...` Additional arguments (not used)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RecipeCertification$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other tidy-api: RecipeCategory, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
# Use recipe_certification() for the public API:
cert <- recipe_certification()
inst <- recipe_user("IECON", type = "institution")
official <- recipe_certification("official", certified_by = inst)
```

RecipeUser *RecipeUser*

Description

User identity for the recipe ecosystem. Supports three account types: individual, institutional_member, and institution.

Value

An object of class RecipeUser.

Public fields

`name` Character. User or institution name.

`email` Character or NULL. Email address.

`user_type` Character. One of "individual", "institutional_member", "institution".

`affiliation` Character or NULL. Organizational affiliation.

`institution` RecipeUser or NULL. Parent institution (for institutional_member).

`url` Character or NULL. Institution URL.

`verified` Logical. Whether the account is verified.

`review_status` Character. One of "approved", "pending", "rejected".

Methods

Public methods:

- RecipeUser$new()
- RecipeUser$trust_level()
- RecipeUser$can_certify()
- RecipeUser$to_list()

- RecipeUser$print()
- RecipeUser$clone()

Method `new()`**:** Create a new RecipeUser

Usage:

```
RecipeUser$new(
  name,
  user_type,
  email = NULL,
  affiliation = NULL,
  institution = NULL,
  url = NULL,
  verified = FALSE,
  review_status = "approved"
)
```

Arguments:

`name` Character. User or institution name.

`user_type` Character. One of "individual", "institutional_member", "institution".

`email` Character or NULL. Email address.

`affiliation` Character or NULL. Organizational affiliation.

`institution` RecipeUser or NULL. Parent institution for institutional_member.

`url` Character or NULL. Institution URL.

`verified` Logical. Whether account is verified.

`review_status` Character. "approved", "pending", or "rejected".

Method `trust_level()`**:** Get trust level (1=individual, 2=member, 3=institution)

Usage:

```
RecipeUser$trust_level()
```

Returns: Integer trust level

Method `can_certify()`**:** Check if user can certify at a given level

Usage:

```
RecipeUser$can_certify(level)
```

Arguments:

`level` Character. Certification level ("reviewed" or "official").

Returns: Logical

Method `to_list()`**:** Serialize to list for JSON

Usage:

```
RecipeUser$to_list()
```

Returns: List representation

Method `print()`**:** Print user card

Usage:

`RecipeUser$print(...)`

Arguments:

`...` Additional arguments (not used)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RecipeUser$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_recipes()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_category()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
# Use recipe_user() for the public API:
user <- recipe_user("Juan Perez", email = "juan@example.com")
inst <- recipe_user("IECON", type = "institution")
member <- recipe_user(
  "Maria",
  type = "institutional_member",
  institution = inst
)
```

RecipeWorkflow-class *RecipeWorkflow R6 class*

Description

RecipeWorkflow R6 class

RecipeWorkflow R6 class

Format

An R6 class generator (R6ClassGenerator)

Details

R6 class representing a publishable workflow that captures statistical estimations applied to survey data. Workflows reference the recipes they use and document the estimation calls made.

Value

An object of class `RecipeWorkflow`.

Methods

$new(...) Class constructor.

$doc() Generate documentation for the workflow.

$to_list() Serialize to a plain list for JSON export.

$increment_downloads() Increment the download counter.

$add_category(category) Add a category.

$certify(user, level) Certify the workflow.

Public fields

`id` Unique identifier (character).

`name` Descriptive name (character).

`description` Workflow description (character).

`user` Author/owner (character).

`user_info` RecipeUser object or NULL.

`survey_type` Survey type (character).

`edition` Survey edition (character).

`estimation_type` Character vector of estimation types used.

`recipe_ids` Character vector of recipe IDs referenced.

`calls` List of deparsed call strings.

`call_metadata` List of lists with type, formula, by, description fields.

`categories` List of RecipeCategory objects.

`downloads` Integer download count.

`certification` RecipeCertification object.

`version` Version string.

`doi` DOI or external identifier (character|NULL).

`created_at` Creation timestamp (character).

`weight_spec` Named list with weight configuration per periodicity (list|NULL).

Methods

Public methods:

- `RecipeWorkflow$new()`
- `RecipeWorkflow$doc()`
- `RecipeWorkflow$to_list()`
- `RecipeWorkflow$increment_downloads()`
- `RecipeWorkflow$certify()`

- RecipeWorkflow$add_category()
- RecipeWorkflow$remove_category()
- RecipeWorkflow$clone()

Method `new()`**:** Create a RecipeWorkflow object

Usage:

```
RecipeWorkflow$new(
  id = NULL,
  name,
  description = "",
  user = "Unknown",
  user_info = NULL,
  survey_type = "Unknown",
  edition = "Unknown",
  estimation_type = character(0),
  recipe_ids = character(0),
  calls = list(),
  call_metadata = list(),
  categories = list(),
  downloads = 0L,
  certification = NULL,
  version = "1.0.0",
  doi = NULL,
  created_at = NULL,
  weight_spec = NULL
)
```

Arguments:

`id` Unique identifier

`name` Descriptive name

`description` Workflow description

`user` Author name

`user_info` RecipeUser object or NULL

`survey_type` Survey type

`edition` Survey edition

`estimation_type` Character vector of estimation types

`recipe_ids` Character vector of recipe IDs

`calls` List of deparsed call strings

`call_metadata` List of call metadata lists

`categories` List of RecipeCategory objects

`downloads` Integer download count

`certification` RecipeCertification or NULL

`version` Version string

`doi` DOI or NULL

`created_at` Timestamp string or NULL (auto-generated)

`weight_spec` Named list with weight configuration per periodicity

Method `doc()`**:** Generate documentation for this workflow

Usage:

```
RecipeWorkflow$doc()
```

Returns: List with meta, recipe_ids, estimations, and estimation_types

Method `to_list()`**:** Serialize to a plain list for JSON export

Usage:

```
RecipeWorkflow$to_list()
```

Returns: A list suitable for jsonlite::write_json

Method `increment_downloads()`**:** Increment the download counter

Usage:

```
RecipeWorkflow$increment_downloads()
```

Method `certify()`**:** Certify the workflow at a given level

Usage:

```
RecipeWorkflow$certify(user, level)
```

Arguments:

`user` RecipeUser who is certifying

`level` Character certification level

Method `add_category()`**:** Add a category to the workflow

Usage:

```
RecipeWorkflow$add_category(category)
```

Arguments:

`category` RecipeCategory to add

Method `remove_category()`**:** Remove a category by name

Usage:

```
RecipeWorkflow$remove_category(name)
```

Arguments:

`name` Character category name to remove

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RecipeWorkflow$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`save_workflow`, `read_workflow`, `workflow`

Other workflows: `evaluate_cv()`, `print.RecipeWorkflow()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `save_workflow()`, `workflow()`, `workflow_from_list()`, `workflow_table()`

Examples

```
wf <- RecipeWorkflow$new(
  name = "Labor workflow", description = "Unemployment rate",
  user = "test", survey_type = "ech", edition = "2023",
  estimation_type = "annual", recipe_ids = "r_001",
  calls = list("svymean(~desocupado, na.rm = TRUE)")
)
```

`recipe_category` *Create a recipe category*

Description

Creates a `RecipeCategory` object for classifying recipes.

Usage

```
recipe_category(name, description = "", parent = NULL)
```

Arguments

`name`	Character. Category identifier (e.g. `"labor_market"`).
`description`	Character. Human-readable description. Defaults to empty.
`parent`	RecipeCategory object or character parent category name (default `NULL`). If a string is provided, it creates a parent category with that name.

Value

A `RecipeCategory` object.

See Also

`RecipeCategory`, `add_category`, `default_categories`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_recipes()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_certification()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
cat <- recipe_category("labor_market", "Labor market indicators")

# With parent hierarchy
sub <- recipe_category(
  "employment", "Employment stats",
  parent = "labor_market"
)
```

`recipe_certification` *Create a recipe certification*

Description

Creates a `RecipeCertification` object. Typically you would use `certify_recipe` to certify a recipe in a pipeline instead.

Usage

```
recipe_certification(level = "community", certified_by = NULL, notes = NULL)
```

Arguments

`level`	Character. One of `"community"` (default), `"reviewed"`, or `"official"`.
`certified_by`	RecipeUser or `NULL` (default `NULL`). Required for reviewed/official.
`notes`	Character or `NULL` (default `NULL`). Additional notes.

Value

A `RecipeCertification` object.

See Also

`RecipeCertification`, `certify_recipe`

Other tidy-api: `RecipeCategory`, `RecipeCertification`, `RecipeUser`, `add_category()`, `certify_recipe()`, `default_categories()`, `filter_recipes()`, `filter_workflows()`, `find_workflows_for_recipe()`, `list_recipes()`, `list_workflows()`, `rank_recipes()`, `rank_workflows()`, `recipe_category()`, `recipe_user()`, `remove_category()`, `search_recipes()`, `search_workflows()`, `set_user_info()`, `set_version()`

Examples

```
# Default community certification
cert <- recipe_certification()

# Official certification
inst <- recipe_user("IECON", type = "institution")
cert <- recipe_certification("official", certified_by = inst)
```

`recipe_user` *Create a recipe user*

Description

Creates a `RecipeUser` object with a simple functional interface.

Usage

```
recipe_user(
  name,
  type = "individual",
  email = NULL,
  affiliation = NULL,
  institution = NULL,
  url = NULL,
  verified = FALSE
)
```

Arguments

`name`	Character. User or institution name.
`type`	Character. One of `"individual"` (default), `"institutional_member"`, or `"institution"`.
`email`	Character or `NULL` (default `NULL`). Email address.
`affiliation`	Character or `NULL` (default `NULL`). Organizational affiliation.
`institution`	RecipeUser object or character institution name. Required for `"institutional_member"` type. If a string is provided, it creates an institution user with that name automatically.
`url`	Character or `NULL` (default `NULL`). Institution URL.
`verified`	Logical (default `FALSE`). Whether the account is verified.

Value

A `RecipeUser` object.

See Also

RecipeUser, set_user_info, certify_recipe

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), remove_category(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
# Individual user
user <- recipe_user("Juan Perez", email = "juan@example.com")

# Institution
inst <- recipe_user(
  "Instituto de Economia",
  type = "institution", verified = TRUE
)

# Member linked to institution
member <- recipe_user(
  "Maria",
  type = "institutional_member",
  institution = inst
)

# Member with institution name shortcut
member2 <- recipe_user(
  "Pedro",
  type = "institutional_member",
  institution = "IECON"
)
```

---

remove_category *Remove a category from a recipe*

Description

Pipe-friendly function to remove a category from a Recipe by name.

Usage

```
remove_category(recipe, name)
```

Arguments

recipe	A Recipe object.
name	Character. Category name to remove.

Value

The modified Recipe object.

See Also

add_category

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), search_recipes(), search_workflows(), set_user_info(), set_version()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
r <- r |>
  add_category("labor_market") |>
  remove_category("labor_market")
```

---

reproduce_workflow *Reproduce a workflow from its published specification*

Description

Given a RecipeWorkflow (typically fetched from the registry), downloads the data, resolves the weight configuration, fetches referenced recipes, and returns a Survey object ready for `workflow()` estimation.

Usage

```
reproduce_workflow(wf, data_path = NULL, dest_dir = tempdir())
```

Arguments

wf	RecipeWorkflow object
data_path	Character path to survey microdata. If NULL, attempts to download from ANDA for ECH surveys.
dest_dir	Character directory for downloaded files

Value

Survey object with recipes applied and weight configuration set

See Also

Other workflows: RecipeWorkflow-class, evaluate_cv(), print.RecipeWorkflow(), publish_workflow(), read_workflow(), save_workflow(), workflow(), workflow_from_list(), workflow_table()

Examples

```
## Not run:
wf <- api_get_workflow("w_123")
svy <- reproduce_workflow(wf)

## End(Not run)
```

---

resolve_weight_spec *Resolve a portable weight specification to a usable weight configuration*

Description

Converts the portable weight_spec from a RecipeWorkflow back into the format expected by load_survey() and add_weight(). For replicate weights with ANDA sources, automatically downloads the replicate file.

Usage

```
resolve_weight_spec(weight_spec, dest_dir = tempdir())
```

Arguments

weight_spec	Named list from RecipeWorkflow$weight_spec
dest_dir	Character directory for downloaded files (default: tempdir())

Value

Named list compatible with add_weight() output

See Also

Other weights: add_replicate(), add_weight()

Examples

```
## Not run:
wf <- api_get_workflow("w_123")
weight <- resolve_weight_spec(wf$weight_spec)

## End(Not run)
```

RotativePanelSurvey *RotativePanelSurvey Class*

Description

This class represents a rotative panel survey, which includes implantation and follow-up surveys. It provides methods to access and manipulate survey data, steps, recipes, workflows, and designs.

Value

An object of class `RotativePanelSurvey`.

Public fields

`implantation` A survey object representing the implantation survey.

`follow_up` A list of survey objects representing the follow-up surveys.

`type` A string indicating the type of the survey.

`default_engine` A string specifying the default engine used for processing.

`steps` A list of steps applied to the survey.

`recipes` A list of recipes associated with the survey.

`workflows` A list of workflows associated with the survey.

`design` A design object for the survey.

`periodicity` A list containing the periodicity of the implantation and follow-up surveys.

Methods

Public methods:

- `RotativePanelSurvey$new()`
- `RotativePanelSurvey$get_implantation()`
- `RotativePanelSurvey$get_follow_up()`
- `RotativePanelSurvey$get_type()`
- `RotativePanelSurvey$get_default_engine()`
- `RotativePanelSurvey$get_steps()`
- `RotativePanelSurvey$get_recipes()`
- `RotativePanelSurvey$get_workflows()`
- `RotativePanelSurvey$get_design()`
- `RotativePanelSurvey$print()`
- `RotativePanelSurvey$clone()`

Method `new()`**:** Initializes a new instance of the RotativePanelSurvey class.

Usage:

```
RotativePanelSurvey$new(
  implantation,
  follow_up,
  type,
  default_engine,
  steps,
  recipes,
  workflows,
  design
)
```

Arguments:

`implantation` A survey object representing the implantation survey.

`follow_up` A list of survey objects representing the follow-up surveys.

`type` A string indicating the type of the survey.

`default_engine` A string specifying the default engine used for processing.

`steps` A list of steps applied to the survey.

`recipes` A list of recipes associated with the survey.

`workflows` A list of workflows associated with the survey.

`design` A design object for the survey.

Method `get_implantation()`**:** Retrieves the implantation survey.

Usage:

```
RotativePanelSurvey$get_implantation()
```

Returns: A survey object representing the implantation survey.

Method `get_follow_up()`**:** Retrieves the follow-up surveys.

Usage:

```
RotativePanelSurvey$get_follow_up(
  index = length(self$follow_up),
  monthly = NULL,
  quarterly = NULL,
  semiannual = NULL,
  annual = NULL
)
```

Arguments:

`index` An integer specifying the index of the follow-up survey to retrieve.

`monthly` A vector of integers specifying monthly intervals.

`quarterly` A vector of integers specifying quarterly intervals.

`semiannual` A vector of integers specifying semiannual intervals.

`annual` A vector of integers specifying annual intervals.

Returns: A list of follow-up surveys matching the specified criteria.

Method `get_type()`**:** Retrieves the type of the survey.

Usage:

```
RotativePanelSurvey$get_type()
```

Returns: A string indicating the type of the survey.

Method `get_default_engine()`**:** Retrieves the default engine used for processing.

Usage:

```
RotativePanelSurvey$get_default_engine()
```

Returns: A string specifying the default engine.

Method `get_steps()`**:** Retrieves the steps applied to the survey.

Usage:

```
RotativePanelSurvey$get_steps()
```

Returns: A list containing the steps for the implantation and follow-up surveys.

Method `get_recipes()`**:** Retrieves the recipes associated with the survey.

Usage:

```
RotativePanelSurvey$get_recipes()
```

Returns: A list of recipes.

Method `get_workflows()`**:** Retrieves the workflows associated with the survey.

Usage:

```
RotativePanelSurvey$get_workflows()
```

Returns: A list of workflows.

Method `get_design()`**:** Retrieves the design object for the survey.

Usage:

```
RotativePanelSurvey$get_design()
```

Returns: A design object.

Method `print()`**:** Prints metadata about the RotativePanelSurvey object.

Usage:

```
RotativePanelSurvey$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RotativePanelSurvey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other panel-surveys: `PoolSurvey`, `extract_surveys()`, `get_follow_up()`, `get_implantation()`

Examples

```
impl <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023", type = "ech", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
fu1 <- Survey$new(
  data = data.table::data.table(id = 1:5, w = 1),
  edition = "2023_01", type = "ech", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
panel <- RotativePanelSurvey$new(
  implantation = impl, follow_up = list(fu1),
  type = "ech", default_engine = "data.table",
  steps = list(), recipes = list(), workflows = list(), design = NULL
)
```

---

save_recipe *Save Recipe*

Description

Saves a Recipe object to a file in JSON format.

Usage

```
save_recipe(recipe, file)
```

Arguments

recipe	A Recipe object.
file	A character string specifying the file path.

Details

This function encodes the Recipe object and writes it to a JSON file.

Value

NULL.

See Also

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), publish_recipe(), read_recipe(), recipe(), steps_to_recipe()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
f <- tempfile(fileext = ".json")
save_recipe(r, f)
```

`save_workflow` *Save a RecipeWorkflow to a JSON file*

Description

Save a RecipeWorkflow to a JSON file

Usage

```
save_workflow(wf, file)
```

Arguments

`wf`	A RecipeWorkflow object
`file`	Character file path

Value

NULL (called for side-effect)

See Also

Other workflows: `RecipeWorkflow-class`, `evaluate_cv()`, `print.RecipeWorkflow()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `workflow()`, `workflow_from_list()`, `workflow_table()`

Examples

```
wf <- RecipeWorkflow$new(
  name = "Example", description = "Test",
  survey_type = "ech", edition = "2023",
  recipe_ids = "r_001", estimation_type = "svymean"
)
f <- tempfile(fileext = ".json")
save_workflow(wf, f)
```

search_recipes *Search recipes*

Description

Search for recipes by name or description in the active backend.

Usage

```
search_recipes(query)
```

Arguments

query	Character search string.

Value

List of matching Recipe objects.

See Also

filter_recipes, rank_recipes, set_backend

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_workflows(), set_user_info(), set_version()

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
r <- recipe(
  name = "Labor Market", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Labor market indicators"
)
publish_recipe(r)
results <- search_recipes("labor")
length(results)
```

search_workflows *Search workflows*

Description

Search for workflows by name or description in the active workflow backend.

Usage

```
search_workflows(query)
```

Arguments

query	Character search string.

Value

List of matching RecipeWorkflow objects.

See Also

filter_workflows, rank_workflows

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), set_user_info(), set_version()

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
results <- search_workflows("labor market")
length(results)
```

set_backend *Set recipe backend*

Description

Configure the active recipe backend via options.

Usage

```
set_backend(type, path = NULL)
```

Arguments

type	Character. "local" or "api" (also accepts "mongo" for backward compat).
path	Character. File path for local backend.

Value

Invisibly, the RecipeBackend object created.

See Also

Other backends: get_backend(), get_workflow_backend(), set_workflow_backend()

Examples

```
set_backend("local", path = tempfile(fileext = ".json"))
```

---

set_data *Set data on a Survey*

Description

Tidy wrapper for svy$set_data(data).

Usage

```
set_data(svy, data, .copy = FALSE)
```

Arguments

svy	Survey object
data	A data.frame or data.table with survey microdata
.copy	Logical; if TRUE, clone the Survey before modifying (default FALSE)

Value

The Survey object (invisibly). If .copy=TRUE, returns a new clone.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(id = 1:5, x = rnorm(5), w = rep(1, 5))
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
new_dt <- data.table::data.table(id = 1:3, x = rnorm(3), w = rep(1, 3))
svy <- set_data(svy, new_dt)
```

---

set_engine *Configure the survey data engine*

Description

Configures the engine to be used for loading surveys. Checks if the provided engine is supported, sets the default engine if none is specified, and generates a message indicating the configured engine. If the engine is not supported, it throws an error.

Usage

```
set_engine(.engine = show_engines())
```

Arguments

.engine	Character vector with the name of the engine to configure. By default, the engine returned by the show_engines() function is used.

Value

Invisibly, the previous engine name (for restoring).

See Also

Other options: get_engine(), lazy_default(), set_lazy_processing(), set_use_copy(), show_engines(), use_copy_default()

Examples

```
old <- set_engine("data.table")
get_engine()
```

set_lazy_processing *Set lazy processing*

Description

Set lazy processing

Usage

```
set_lazy_processing(lazy)
```

Arguments

lazy	Logical. If TRUE, steps are deferred until bake_steps() is called.

Value

Invisibly, the previous value.

See Also

Other options: get_engine(), lazy_default(), set_engine(), set_use_copy(), show_engines(), use_copy_default()

Examples

```
old <- lazy_default()
set_lazy_processing(FALSE)
lazy_default() # now FALSE
set_lazy_processing(old) # restore
```

set_user_info *Set user info on a recipe*

Description

Pipe-friendly function to assign a RecipeUser to a Recipe.

Usage

```
set_user_info(recipe, user)
```

Arguments

recipe	A Recipe object.
user	A RecipeUser object.

Value

The modified Recipe object.

See Also

recipe_user

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_version()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
user <- recipe_user("Juan Perez", email = "juan@example.com")
r <- r |> set_user_info(user)
```

---

set_use_copy *Set data copy option*

Description

Configures whether survey operations should create copies of the data or modify existing data in place. This setting affects memory usage and performance across the metasurvey package.

Usage

```
set_use_copy(use_copy)
```

Arguments

use_copy	Logical value: TRUE to create data copies (safer), FALSE to modify data in place (more efficient)

Details

Setting use_copy affects all subsequent survey operations:

- TRUE (default): Operations create data copies, preserving original data
- FALSE: Operations modify data in place, reducing memory usage

Use FALSE for large datasets where memory is a concern, but ensure you don't need the original data after operations.

Value

Invisibly, the previous value (for restoring).

See Also

use_copy_default to check current setting

Other options: get_engine(), lazy_default(), set_engine(), set_lazy_processing(), show_engines(), use_copy_default()

Examples

```
# Set to use copies (default behavior)
set_use_copy(TRUE)
use_copy_default()

# Set to modify in place for better performance
set_use_copy(FALSE)
use_copy_default()

# Reset to default
set_use_copy(TRUE)
```

---

set_version *Set version on a recipe*

Description

Pipe-friendly function to set the version string on a Recipe.

Usage

```
set_version(recipe, version)
```

Arguments

recipe	A Recipe object.
version	Character version string (e.g. "2.0.0").

Value

The modified Recipe object.

See Also

Other tidy-api: RecipeCategory, RecipeCertification, RecipeUser, add_category(), certify_recipe(), default_categories(), filter_recipes(), filter_workflows(), find_workflows_for_recipe(), list_recipes(), list_workflows(), rank_recipes(), rank_workflows(), recipe_category(), recipe_certification(), recipe_user(), remove_category(), search_recipes(), search_workflows(), set_user_info()

Examples

```
r <- recipe(
  name = "Example", user = "Test",
  svy = survey_empty(type = "ech", edition = "2023"),
  description = "Example recipe"
)
r <- r |> set_version("2.0.0")
```

`set_workflow_backend` *Set workflow backend*

Description

Configure the active workflow backend via options.

Usage

```
set_workflow_backend(type, path = NULL)
```

Arguments

`type`	Character. "local" or "api" (also accepts "mongo" for backward compat).
`path`	Character. File path for local backend.

Value

Invisibly, the WorkflowBackend object created.

See Also

Other backends: `get_backend()`, `get_workflow_backend()`, `set_backend()`

Examples

```
set_workflow_backend("local", path = tempfile(fileext = ".json"))
```

show_engines *List available survey data engines*

Description

Returns a character vector of available engines that can be used for loading surveys.

Usage

```
show_engines()
```

Value

Character vector with the names of the available engines.

See Also

Other options: `get_engine()`, `lazy_default()`, `set_engine()`, `set_lazy_processing()`, `set_use_copy()`, `use_copy_default()`

Examples

```
show_engines()
```

steps_to_recipe *Convert a list of steps to a recipe*

Description

Convert a list of steps to a recipe

Usage

```
steps_to_recipe(
  name,
  user,
  svy = survey_empty(type = "eaii", edition = "2019-2021"),
  description,
  steps,
  doi = NULL,
  topic = NULL
)
```

Arguments

name	A character string with the name of the recipe
user	A character string with the user of the recipe
svy	A Survey object
description	A character string with the description of the recipe
steps	A list with the steps of the recipe
doi	A character string with the DOI of the recipe
topic	A character string with the topic of the recipe

Value

A Recipe object

See Also

Other recipes: Recipe-class, add_recipe(), bake_recipes(), explore_recipes(), get_recipe(), harmonize(), print.Recipe(), publish_recipe(), read_recipe(), recipe(), save_recipe()

Examples

```
dt <- data.table::data.table(
  id = 1:20, age = sample(18:65, 20, TRUE),
  w = runif(20, 0.5, 2)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "demo",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
svy <- step_compute(svy, age2 = age^2)
my_recipe <- steps_to_recipe(
  name = "age_vars", user = "analyst",
  svy = svy, description = "Age-derived variables",
  steps = get_steps(svy)
)
my_recipe
```

---

step_compute *Create computation steps for survey variables*

Description

This function uses optimized expression evaluation with automatic dependency detection and error prevention. All computations are validated before execution.

Usage

```
step_compute(
  svy = NULL,
  ...,
  .by = NULL,
  .copy = use_copy_default(),
  comment = "Compute step",
  .level = "auto",
  use_copy = deprecated()
)
```

Arguments

svy	A Survey or RotativePanelSurvey object. If NULL, creates a step that can be applied later using the pipe operator (%>%)
...	Computation expressions with automatic optimization. Names are assigned using new_var = expression
.by	Vector of variables to group computations by. The system automatically validates these variables exist before execution
.copy	Logical indicating whether to create a copy of the object before applying transformations. Defaults to use_copy_default()
comment	Descriptive text for the step for documentation and traceability. Compatible with Markdown syntax. Defaults to "Compute step"
.level	For RotativePanelSurvey objects (default "auto"), specifies the level where computations are applied: "implantation", "follow_up", "quarter", "month", or "auto"
use_copy	**[Deprecated]** Use .copy instead.

Details

Lazy evaluation (default): By default, steps are recorded but **not executed** until bake_steps() is called. This allows building a full pipeline before materializing any changes. Set options(metasurvey.lazy_processing = FALSE) to apply steps immediately.

Expression processing: Expressions are evaluated using data.table's := operator. Variable dependencies are detected automatically via all.vars(). Missing variables are caught before execution.

Grouped computations: Use .by to compute aggregated values (e.g., group means) that are automatically joined back to the data.

For RotativePanelSurvey objects, .level controls where computations are applied:

- "auto" (default): applies to both implantation and follow-ups
- "implantation": household/dwelling level only
- "follow_up": individual/person level only

Value

Same type of input object (Survey or RotativePanelSurvey) with new computed variables and the step added to the history

See Also

step_recode for categorical recodings bake_steps to execute all pending steps

Other steps: bake_steps(), get_steps(), step_filter(), step_join(), step_recode(), step_remove(), step_rename(), step_validate(), view_graph()

Examples

```
# Basic computation
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60), w = 1
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy <- svy |> step_compute(age_squared = age^2, comment = "Age squared")
svy <- bake_steps(svy)
get_data(svy)


# ECH example: labor indicator
# ech <- ech |>
#   step_compute(
#     unemployed = ifelse(POBPCOAC %in% 3:5, 1, 0),
#     comment = "Unemployment indicator")
```

---

step_filter *Filter rows from survey data*

Description

Creates a step that filters (subsets) rows from the survey data based on logical conditions. Multiple conditions are combined with AND.

Usage

```
step_filter(
  svy,
  ...,
  .by = NULL,
  .copy = use_copy_default(),
  comment = "Filter step",
  .level = "auto"
)
```

Arguments

svy	A Survey or RotativePanelSurvey object.
...	Logical expressions evaluated against the data. Each must return a logical vector. Multiple conditions are combined with AND.
.by	Optional grouping variable(s) for within-group filtering.
.copy	Whether to operate on a copy (default: use_copy_default()).
comment	Descriptive text for the step (default "Filter step").
.level	For RotativePanelSurvey, the level to apply (default "auto"): "implantation", "follow_up", or "auto" (both).

Details

Lazy evaluation (default): Like all steps, filter is recorded but **not executed** until bake_steps() is called.

Value

The survey object with rows filtered and the step recorded.

See Also

Other steps: bake_steps(), get_steps(), step_compute(), step_join(), step_recode(), step_remove(), step_rename(), step_validate(), view_graph()

Examples

```
svy <- Survey$new(
  data = data.table::data.table(
    id = 1:10, age = c(15, 25, 35, 45, 55, 65, 75, 20, 30, 40), w = 1
  ),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
svy <- svy |>
  step_filter(age >= 18) |>
  bake_steps()
nrow(get_data(svy))
```

---

step_join *Join external data into survey (step)*

Description

Creates a step that joins additional data into a Survey or RotativePanelSurvey.

Usage

```
step_join(
  svy,
  x,
  by = NULL,
  type = c("left", "inner", "right", "full"),
  suffixes = c("", ".y"),
  .copy = use_copy_default(),
  comment = "Join step",
  use_copy = deprecated(),
  lazy = lazy_default(),
  record = TRUE
)
```

Arguments

svy	A Survey or RotativePanelSurvey object. If NULL, returns a step call
x	A data.frame/data.table or a Survey to join into svy
by	Character vector of join keys. Named vector for different names between svy and x (names are keys in svy, values are keys in x). If NULL, tries to infer common column names
type	Join type: "left" (default), "inner", "right", or "full"
suffixes	Length-2 character vector of suffixes for conflicting columns from svy and x respectively. Defaults to c("", ".y")
.copy	Whether to operate on a copy (default: use_copy_default())
comment	Optional description for the step (default "Join step").
use_copy	**[Deprecated]** Use .copy instead.
lazy	Internal. Whether to delay execution (default lazy_default()).
record	Internal. Whether to record the step (default TRUE).

Details

Lazy evaluation (default): By default, steps are recorded but **not executed** until bake_steps() is called.

Supports left, inner, right, and full joins. Allows named by mapping (e.g., c("id" = "code")) or a simple character vector. Conflicting column names are resolved by appending suffixes to the right-hand side columns.

Value

Modified survey object with the join recorded as a step (and applied immediately when baked). For RotativePanelSurvey, the join is applied to implantation and every follow_up survey.

See Also

Other steps: bake_steps(), get_steps(), step_compute(), step_filter(), step_recode(), step_remove(), step_rename(), step_validate(), view_graph()

Examples

```
# With data.frame
s <- Survey$new(
  data = data.table::data.table(id = 1:3, w = 1, a = c("x", "y", "z")),
  edition = "2023", type = "ech", psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
info <- data.frame(id = c(1, 2), b = c(10, 20))
s2 <- step_join(s, info, by = "id", type = "left")
s2 <- bake_steps(s2)

# With another Survey
s_right <- Survey$new(
  data = data.table::data.table(id = c(2, 3), b = c(200, 300), w2 = 1),
  edition = "2023", type = "ech", psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w2")
)
s3 <- step_join(s, s_right, by = c("id" = "id"), type = "inner")
s3 <- bake_steps(s3)
```

---

step_recode *Create recoding steps for categorical variables*

Description

This function uses optimized expression evaluation for all recoding conditions. All conditional expressions are validated and optimized for efficient execution.

Usage

```
step_recode(
  svy,
  new_var,
  ...,
  .default = NA_character_,
  .name_step = NULL,
  ordered = FALSE,
  .copy = use_copy_default(),
  comment = "Recode step",
  .to_factor = FALSE,
  .level = "auto",
  use_copy = deprecated()
)
```

Arguments

svy	A Survey or RotativePanelSurvey object. If NULL, creates a step that can be applied later using the pipe operator (%>%)
new_var	Name of the new variable to create (unquoted)
...	Sequence of two-sided formulas defining recoding rules. Left-hand side (LHS) is a conditional expression, right-hand side (RHS) defines the replacement value. Format: condition ~ value
.default	Default value assigned when no condition is met. Defaults to NA_character_
.name_step	**[Deprecated]** Custom name for the step in the history. Now auto-generated from the variable name. Use comment for user-facing documentation instead.
ordered	Logical indicating whether the new variable should be an ordered factor. Defaults to FALSE
.copy	Logical indicating whether to create a copy of the object before applying transformations. Defaults to use_copy_default()
comment	Descriptive text for the step for documentation and traceability. Compatible with Markdown syntax. Defaults to "Recode step"
.to_factor	Logical indicating whether the new variable should be converted to a factor. Defaults to FALSE
.level	For RotativePanelSurvey objects (default "auto"), specifies the level where recoding is applied: "implantation", "follow_up", "quarter", "month", or "auto"
use_copy	**[Deprecated]** Use .copy instead.

Details

Lazy evaluation (default): By default, steps are recorded but **not executed** until bake_steps() is called. This allows building a full pipeline before materializing any changes.

Condition evaluation: Conditions are two-sided formulas evaluated in order. The first matching condition determines the assigned value. If no condition matches, .default is used.

Condition examples:

- Simple: variable == 1 ~ "Yes"
- Complex: age >= 18 & income > 12000 ~ "High"
- Vectorized: variable %in% c(1,2,3) ~ "Group A"
- Logical: !is.na(education) ~ "Has education"

Value

Same type of input object (Survey or RotativePanelSurvey) with the new recoded variable and the step added to the history

See Also

step_compute for more complex calculations bake_steps to execute all pending steps get_steps to view step history

Other steps: bake_steps(), get_steps(), step_compute(), step_filter(), step_join(), step_remove(), step_rename(), step_validate(), view_graph()

Examples

```
# Basic recode: categorize ages
dt <- data.table::data.table(
  id = 1:6, age = c(10, 25, 45, 60, 70, 80), w = 1
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
svy <- svy |>
  step_recode(
    age_group,
    age < 18 ~ "Under 18",
    age >= 18 & age < 65 ~ "Working age",
    age >= 65 ~ "Senior",
    .default = "Unknown"
  )
svy <- bake_steps(svy)
get_data(svy)


# ECH example: labor force status
# ech <- ech |>
#   step_recode(labor_status,
#     POBPCOAC == 2 ~ "Employed",
#     POBPCOAC %in% 3:5 ~ "Unemployed",
#     .default = "Missing")
```

---

step_remove *Remove variables from survey data (step)*

Description

Creates a step that removes one or more variables from the survey data when baked.

Usage

```
step_remove(
  svy,
  ...,
  vars = NULL,
  .copy = use_copy_default(),
  comment = "Remove variables",
  use_copy = deprecated(),
  lazy = lazy_default(),
  record = TRUE
)
```

Arguments

svy	A Survey or RotativePanelSurvey object
...	Unquoted variable names to remove, or a character vector
vars	Character vector of variable names to remove. Alternative to ... for programmatic use.
.copy	Whether to operate on a copy (default: use_copy_default())
comment	Descriptive text for the step for documentation and traceability (default "Remove variables").
use_copy	**[Deprecated]** Use .copy instead.
lazy	Internal. Whether to delay execution (default lazy_default()).
record	Internal. Whether to record the step (default TRUE).

Details

Lazy evaluation (default): By default, steps are recorded but **not executed** until bake_steps() is called.

Variables can be specified in two ways:

- **Unquoted names:** step_remove(svy, age, income)
- **Character vector:** step_remove(svy, vars = c("age", "income"))

Variables that don't exist in the data produce a warning (not an error), allowing pipelines to be robust to missing columns.

Value

Survey object with the specified variables removed (or queued for removal).

See Also

Other steps: bake_steps(), get_steps(), step_compute(), step_filter(), step_join(), step_recode(), step_rename(), step_validate(), view_graph()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy2 <- step_remove(svy, age)
svy2 <- bake_steps(svy2)
"age" %in% names(get_data(svy2)) # FALSE
```

`step_rename` *Rename variables in survey data (step)*

Description

Creates a step that renames variables in the survey data when baked.

Usage

```
step_rename(
  svy,
  ...,
  mapping = NULL,
  .copy = use_copy_default(),
  comment = "Rename variables",
  use_copy = deprecated(),
  lazy = lazy_default(),
  record = TRUE
)
```

Arguments

`svy`	A Survey or RotativePanelSurvey object
`...`	Pairs in the form `new_name = old_name` (unquoted).
`mapping`	A named character vector of the form `c(new_name = "old_name")`. Alternative to `...` for programmatic use.
`.copy`	Whether to operate on a copy (default: `use_copy_default()`)
`comment`	Descriptive text for the step for documentation and traceability (default `"Rename variables"`).
`use_copy`	**[Deprecated]** Use `.copy` instead.
`lazy`	Internal. Whether to delay execution (default `lazy_default()`).
`record`	Internal. Whether to record the step (default `TRUE`).

Details

Lazy evaluation (default): By default, steps are recorded but **not executed** until `bake_steps()` is called.

Variables can be renamed in two ways:

- **Unquoted pairs:** `step_rename(svy, new_name = old_name)`
- **Named character vector:** `step_rename(svy, mapping = c(new_name = "old_name"))`

Variables that don't exist in the data cause an error, unlike `step_remove()` which issues a warning.

Value

Survey object with the specified variables renamed (or queued for renaming).

See Also

Other steps: `bake_steps()`, `get_steps()`, `step_compute()`, `step_filter()`, `step_join()`, `step_recode()`, `step_remove()`, `step_validate()`, `view_graph()`

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy2 <- step_rename(svy, edad = age)
svy2 <- bake_steps(svy2)
"edad" %in% names(get_data(svy2)) # TRUE
```

---

step_validate *Validate data during the step pipeline*

Description

Creates a non-mutating step that checks data invariants when `bake_steps` is called. Each check is a logical expression evaluated row-wise against the survey data. If any row fails a check, the pipeline stops (or warns).

Usage

```
step_validate(
  svy,
  ...,
  .action = c("stop", "warn"),
  .min_n = NULL,
  .copy = use_copy_default(),
  comment = "Validate step"
)
```

Arguments

svy	A Survey or RotativePanelSurvey object
...	Logical expressions evaluated against the data. Each must return a logical vector with one value per row. Named expressions use the name in error messages; unnamed expressions use the deparsed code. Examples: `income > 0`, `!is.na(age)`, `sex %in% c(1, 2)`.
.action	What to do when a check fails: `"stop"` (default) raises an error, `"warn"` issues a warning and continues.
.min_n	Minimum number of rows required. Checked before row-level expressions.
.copy	Whether to operate on a copy (default: `use_copy_default()`)
comment	Descriptive text for the step for documentation and traceability (default `"Validate step"`).

Details

Lazy evaluation (default): Like all steps, validation checks are recorded but **not executed** until `bake_steps` is called. This means `step_validate` can reference variables created by preceding `step_compute` calls.

The validate step does **not** modify the data in any way. It only inspects the current state of the data.table and raises an error or warning if any check fails.

Value

The survey object with a validate step recorded (no data mutation).

See Also

Other steps: `bake_steps()`, `get_steps()`, `step_compute()`, `step_filter()`, `step_join()`, `step_recode()`, `step_remove()`, `step_rename()`, `view_graph()`

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  income = c(1000, 2000, 3000, 4000, 5000), w = 1
)
svy <- Survey$new(
```

```
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)

# Validate that all ages are positive and income is not NA
svy <- svy |>
  step_validate(age > 0, !is.na(income), .min_n = 3) |>
  bake_steps()
```

`Survey` *Survey R6 class*

Description

Survey R6 class

Survey R6 class

Format

An R6 class generator (R6ClassGenerator)

Details

R6 class that encapsulates survey data, metadata (type, edition, periodicity), sampling design (simple/replicate), steps/recipes/workflows, and utilities to manage them.

Only copies the data; reuses design objects and other metadata. Much faster than clone(deep=TRUE) but design objects are shared.

Value

R6 class generator for Survey.

Main methods

$new(data, edition, type, psu, strata, engine, weight, design = NULL, steps = NULL, recipes = list()) Constructor.

$set_data(data) Set data.

$set_edition(edition) Set edition.

$set_type(type) Set type.

$set_weight(weight) Set weight specification.

$print() Print summarized metadata.

$add_step(step) Add a step and invalidate design.

$add_recipe(recipe) Add a recipe (validates type and dependencies).

$add_workflow(workflow) Add a workflow.

$bake() Apply recipes and return updated Survey.

$ensure_design() Lazily initialize the sampling design.

$update_design() Update design variables with current data.

$shallow_clone() Efficient copy (shares design, copies data).

Public fields

`data` Survey data (data.frame/data.table).

`edition` Reference edition or period.

`type` Survey type, e.g., "ech" (character).

`periodicity` Periodicity detected by validate_time_pattern.

`default_engine` Default engine (character).

`weight` List with weight specifications per estimation type.

`steps` List of steps applied to the survey.

`recipes` List of Recipe objects associated.

`workflows` List of workflows.

`design` List of survey design objects (survey/surveyrep).

`psu` Primary Sampling Unit specification (formula or character).

`strata` Stratification variable name (character or NULL).

`design_initialized` Logical flag for lazy design initialization.

`provenance` Data lineage metadata (see `provenance()`).

Active bindings

`design_active` Deprecated. Use `ensure_design()` instead.

Methods

Public methods:

- `Survey$new()`
- `Survey$get_data()`
- `Survey$get_edition()`
- `Survey$get_type()`
- `Survey$set_data()`
- `Survey$set_edition()`
- `Survey$set_type()`
- `Survey$set_weight()`
- `Survey$print()`
- `Survey$add_step()`
- `Survey$add_recipe()`
- `Survey$add_workflow()`
- `Survey$bake()`

- Survey$head()
- Survey$str()
- Survey$set_design()
- Survey$ensure_design()
- Survey$update_design()
- Survey$shallow_clone()
- Survey$clone()

Method `new()`**:** Create a Survey object

Usage:

```
Survey$new(
  data,
  edition,
  type,
  psu = NULL,
  strata = NULL,
  engine,
  weight,
  design = NULL,
  steps = NULL,
  recipes = list()
)
```

Arguments:

`data` Survey data

`edition` Edition or period

`type` Survey type (character)

`psu` PSU variable or formula (optional)

`strata` Stratification variable name (optional)

`engine` Default engine

`weight` Weight specification(s) per estimation type

`design` Pre-built design (optional)

`steps` Initial steps list (optional)

`recipes` List of Recipe (optional)

Method `get_data()`**:** Return the underlying data

Usage:

```
Survey$get_data()
```

Method `get_edition()`**:** Return the survey edition/period

Usage:

```
Survey$get_edition()
```

Method `get_type()`**:** Return the survey type identifier

Usage:

```
Survey$get_type()
```

Method `set_data()`: Set the underlying data

Usage:

```
Survey$set_data(data)
```

Arguments:

`data` New survey data

Method `set_edition()`: Set the survey edition/period

Usage:

```
Survey$set_edition(edition)
```

Arguments:

`edition` New edition or period

Method `set_type()`: Set the survey type

Usage:

```
Survey$set_type(type)
```

Arguments:

`type` New type identifier

Method `set_weight()`: Set weight specification(s) per estimation type

Usage:

```
Survey$set_weight(weight)
```

Arguments:

`weight` Weight specification list or character

Method `print()`: Print summarized metadata to console

Usage:

```
Survey$print()
```

Method `add_step()`: Add a step and invalidate design

Usage:

```
Survey$add_step(step)
```

Arguments:

`step` Step object

Method `add_recipe()`: Add a recipe

Usage:

```
Survey$add_recipe(recipe, bake = lazy_default())
```

Arguments:

`recipe` Recipe object

`bake` Whether to bake lazily (internal flag)

Method `add_workflow()`: Add a workflow to the survey

Usage:

```
Survey$add_workflow(workflow)
```

Arguments:

`workflow` Workflow object

Method `bake()`**:** Apply recipes and return updated Survey

Usage:

```
Survey$bake()
```

Method `head()`**:** Return the head of the underlying data

Usage:

```
Survey$head()
```

Method `str()`**:** Display the structure of the underlying data

Usage:

```
Survey$str()
```

Method `set_design()`**:** Set the survey design object

Usage:

```
Survey$set_design(design)
```

Arguments:

`design` Survey design object or list

Method `ensure_design()`**:** Ensure survey design is initialized (lazy initialization)

Usage:

```
Survey$ensure_design()
```

Returns: Invisibly returns self

Method `update_design()`**:** Update design variables using current data and weight

Usage:

```
Survey$update_design()
```

Method `shallow_clone()`**:** Create a shallow copy of the Survey (optimized for performance)

Usage:

```
Survey$shallow_clone()
```

Returns: New Survey object with copied data but shared design

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Survey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

survey_empty, bake_recipes, cat_design, cat_recipes

Other survey-objects: cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(id = 1:5, x = rnorm(5), w = rep(1, 5))
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
svy
```

`survey_empty` *survey_empty*

Description

Create an empty survey

Usage

```
survey_empty(
  edition = NULL,
  type = NULL,
  weight = NULL,
  engine = NULL,
  psu = NULL,
  strata = NULL
)
```

Arguments

`edition`	Edition of survey
`type`	Type of survey
`weight`	Weight of survey
`engine`	Engine of survey
`psu`	PSU variable or formula (optional)
`strata`	Stratification variable name (optional)

Value

Survey object

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_to_data_frame(), survey_to_datatable(), survey_to_tibble()

Examples

```
empty <- survey_empty(edition = "2023", type = "test")
empty_typed <- survey_empty(edition = "2023", type = "ech")
```

---

survey_to_datatable *Convert survey to data.table*

Description

Extracts the survey microdata as a data.table.

Usage

```
survey_to_datatable(svy)

survey_to_data.table(svy)
```

Arguments

svy	Survey object.

Value

A data.table.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
result <- survey_to_datatable(svy)
data.table::is.data.table(result) # TRUE
```

survey_to_data_frame *survey_to_data_frame*

Description

Convert survey to data.frame

Usage

```
survey_to_data_frame(svy)
```

Arguments

svy	Survey object

Value

data.frame

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_datatable(), survey_to_tibble()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
df <- survey_to_data_frame(svy)
class(df) # "data.frame"
```

survey_to_tibble *survey_to_tibble*

Description

Convert survey to tibble

Usage

```
survey_to_tibble(svy)
```

Arguments

svy	Survey object

Value

A tibble (tbl_df) containing the survey data.

See Also

Other survey-objects: Survey, cat_design(), cat_design_type(), get_data(), get_metadata(), has_design(), has_recipes(), has_steps(), is_baked(), set_data(), survey_empty(), survey_to_data_frame(), survey_to_datatable()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table", weight = add_weight(annual = "w")
)
tbl <- survey_to_tibble(svy)
class(tbl)
```

---

transpile_coverage *Analyze transpilation coverage for STATA do-files*

Description

[Experimental]

Reports what percentage of commands in a .do file (or directory of files) can be automatically transpiled vs require manual review.

Usage

```
transpile_coverage(path, recursive = TRUE)
```

Arguments

path	Path to a .do file or directory of .do files
recursive	If TRUE and path is a directory, search subdirectories

Value

A data.frame with columns: file, total_commands, translated, skipped, manual_review, coverage_pct

See Also

Other transpiler: parse_do_file(), parse_stata_labels(), transpile_stata(), transpile_stata_module()

Examples

```
tf <- tempfile(fileext = ".do")
writeLines(c("gen x = 1", "replace x = 2 if y == 3", "drop z"), tf)
transpile_coverage(tf)
```

---

transpile_stata *Transpile a STATA .do file to metasurvey steps*

Description

[Experimental]

Parses a STATA .do file and translates its commands into metasurvey step call strings suitable for use in Recipe objects.

Usage

```
transpile_stata(do_file, survey_type = "ech", user = "iecon", strict = FALSE)
```

Arguments

do_file	Path to a STATA .do file
survey_type	Survey type (default "ech")
user	Author name for the recipe
strict	If TRUE, stops on untranslatable commands; if FALSE, inserts MANUAL_REVIEW comments as warnings

Value

A list with:

- steps: character vector of step call strings
- labels: list with var_labels and val_labels (if label commands found)
- warnings: character vector of MANUAL_REVIEW items
- stats: list with command counts

See Also

Other transpiler: parse_do_file(), parse_stata_labels(), transpile_coverage(), transpile_stata_module()

Examples

```
tf <- tempfile(fileext = ".do")
writeLines(c("gen age2 = edad^2", "replace sexo = 1 if sexo == ."), tf)
result <- transpile_stata(tf)
result$steps
result$stats
```

`transpile_stata_module`

Transpile and group do-files by thematic module

Description

[Experimental]

Processes all do-files in a year directory and groups them into thematic Recipe objects (demographics, income, etc.).

Usage

```
transpile_stata_module(year_dir, year, user = "iecon", output_dir = NULL)
```

Arguments

`year_dir`	Path to a year directory (e.g., "do_files_iecon/2022")
`year`	Year of the edition (character or numeric)
`user`	Author name
`output_dir`	Directory to write JSON recipes (NULL = no file output)

Value

A named list of Recipe objects, one per thematic module

See Also

Other transpiler: parse_do_file(), parse_stata_labels(), transpile_coverage(), transpile_stata()

Examples

```
## Not run:
# Requires a directory of .do files organized by year
recipes <- transpile_stata_module("do_files_iecon/2022", year = 2022)
names(recipes)

## End(Not run)
```

use_copy_default *Get data copy option*

Description

Retrieves the current setting for the use_copy option, which controls whether survey operations create copies of the data or modify in place.

Usage

```
use_copy_default()
```

Details

The use_copy option affects memory usage and performance:

- TRUE: Creates copies, safer but uses more memory
- FALSE: Modifies in place, more efficient but requires caution

Value

Logical value indicating whether to use data copies (TRUE) or modify data in place (FALSE). Default is TRUE.

See Also

set_use_copy to change the setting

Other options: get_engine(), lazy_default(), set_engine(), set_lazy_processing(), set_use_copy(), show_engines()

Examples

```
# Check current setting
current_setting <- use_copy_default()
print(current_setting)
```

validate_time_pattern *Validate time pattern*

Description

Validate time pattern

Usage

```
validate_time_pattern(svy_type = NULL, svy_edition = NULL)
```

Arguments

svy_type	Survey type (e.g. "ech").
svy_edition	Survey edition string (e.g. "2023", "2023-06").

Value

List with components: svy_type, svy_edition (parsed), svy_periodicity.

See Also

Other survey-loading: extract_time_pattern(), group_dates(), load_panel_survey(), load_survey(), load_survey_example()

Examples

```
validate_time_pattern(svy_type = "ech", svy_edition = "2023")
validate_time_pattern(
  svy_type = "ech", svy_edition = "2023-06"
)
```

view_graph *View graph*

Description

View graph

Usage

```
view_graph(svy, init_step = "Load survey")
```

Arguments

svy	Survey object
init_step	Initial step label (default: "Load survey")

Value

A visNetwork interactive graph of the survey processing steps.

See Also

Other steps: bake_steps(), get_steps(), step_compute(), step_filter(), step_join(), step_recode(), step_remove(), step_rename(), step_validate()

Examples

```
dt <- data.table::data.table(
  id = 1:5, age = c(25, 30, 45, 50, 60),
  w = rep(1, 5)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "ech",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
svy <- step_compute(svy, age2 = age * 2)
view_graph(svy)
```

---

workflow *Execute estimation workflow for surveys*

Description

This function executes a sequence of statistical estimations on Survey objects, applying functions from the R survey package with appropriate metadata. Automatically handles different survey types and periodicities.

Usage

```
workflow(svy, ..., estimation_type = "monthly", level = 0.95)
```

Arguments

svy	A **list** of Survey objects, or a PoolSurvey. Even for a single survey, wrap it in `list()`: `workflow(svy = list(my_survey), ...)`. Must contain properly configured sample design.
...	Calls to survey package functions (such as `svymean`, `svytotal`, `svyratio`, etc.) that will be executed sequentially
estimation_type	Type of estimation (default `"monthly"`) that determines which weight to use. Options: `"monthly"`, `"quarterly"`, `"annual"`, or vector with multiple types
level	Confidence level for the interval (default `0.95`). Passed to confint. Can also be set per-call inside the estimation function (e.g., `svymean(~x, na.rm = TRUE, level = 0.90)`), which overrides the `workflow()` default for that estimation.

Details

The function automatically selects the appropriate sample design according to the specified `estimation_type`. For each Survey in the input list, it executes all functions specified in `...` and combines the results.

Supported estimation types:

- "monthly": Monthly estimations
- "quarterly": Quarterly estimations
- "annual": Annual estimations

For PoolSurvey objects, it uses a specialized methodology that handles pooling of multiple surveys.

Value

`data.table` with results from all estimations, including columns:

- `stat`: Estimation call and variable name
- `variable`: Variable name (for filtering)
- `value`: Point estimate
- `se`: Standard error
- `cv`: Coefficient of variation (proportion)
- `confint_lower`: Lower bound of confidence interval
- `confint_upper`: Upper bound of confidence interval
- `evaluate`: CV quality label from `evaluate_cv` (e.g. "Excellent", "Good", "Use with caution")

For `svyby` estimations, grouping variables (e.g. `region`, `sexo`) appear as additional columns.

See Also

`svymean` for population means `svytotal` for population totals `svyratio` for ratios `svyby` for domain estimations `PoolSurvey` for survey pooling

Other workflows: `RecipeWorkflow-class`, `evaluate_cv()`, `print.RecipeWorkflow()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `save_workflow()`, `workflow_from_list()`, `workflow_table()`

Examples

```
# Simple estimation
dt <- data.table::data.table(
  x = rnorm(100), g = sample(c("a", "b"), 100, TRUE),
  w = rep(1, 100)
)
svy <- Survey$new(
  data = dt, edition = "2023", type = "test",
  psu = NULL, engine = "data.table",
  weight = add_weight(annual = "w")
)
result <- workflow(
  svy = list(svy),
```

```
  survey::svymean(~x, na.rm = TRUE),
  estimation_type = "annual"
)

# Domain estimation with svyby
result_by <- workflow(
  svy = list(svy),
  survey::svyby(~x, ~g, survey::svymean, na.rm = TRUE),
  estimation_type = "annual"
)

# Custom confidence level (90%) for all estimations
result_90 <- workflow(
  svy = list(svy),
  survey::svymean(~x, na.rm = TRUE),
  estimation_type = "annual",
  level = 0.90
)

# Per-call confidence level (overrides workflow default)
result_mixed <- workflow(
  svy = list(svy),
  survey::svymean(~x, na.rm = TRUE, level = 0.80),
  estimation_type = "annual"
)
```

`workflow_from_list` *Construct a RecipeWorkflow from a plain list*

Description

Construct a RecipeWorkflow from a plain list

Usage

```
workflow_from_list(lst)
```

Arguments

`lst`	A list (typically from JSON) with workflow fields

Value

A RecipeWorkflow object

See Also

Other workflows: `RecipeWorkflow-class`, `evaluate_cv()`, `print.RecipeWorkflow()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `save_workflow()`, `workflow()`, `workflow_table()`

Examples

```
lst <- list(
  name = "example", user = "test", survey_type = "ech",
  edition = "2023", estimation_type = "svymean"
)
wf <- workflow_from_list(lst)
```

`workflow_table` *Create publication-quality table from workflow results*

Description

Formats a `workflow()` result as a gt table with confidence intervals, CV quality classification, and provenance-based source notes.

Usage

```
workflow_table(
  result,
  ci = 0.95,
  digits = 2,
  compare_by = NULL,
  show_cv = TRUE,
  show_se = TRUE,
  title = NULL,
  subtitle = NULL,
  source_note = TRUE,
  locale = "en",
  theme = "publication"
)
```

Arguments

`result`	A `data.table` from `workflow()`.
`ci`	Confidence level for intervals (default 0.95). Set to `NULL` to hide confidence intervals.
`digits`	Number of decimal places (default 2).
`compare_by`	Column name to pivot for side-by-side comparison (e.g., `"survey_edition"`).
`show_cv`	Logical; show CV column with quality classification.
`show_se`	Logical; show SE column.
`title`	Table title. Auto-generated if `NULL`.
`subtitle`	Table subtitle. Auto-generated if `NULL`.
`source_note`	Logical; show provenance footer.
`locale`	Locale for number formatting (`"en"` or `"es"`).
`theme`	Table theme: `"publication"` (clean) or `"minimal"`.

Details

CV quality classification follows INE/CEPAL standards:

- Excellent: CV < 5\
- Very good: 5-10\
- Good: 10-15\
- Acceptable: 15-25\
- Use with caution: 25-35\
- Do not publish: >= 35\

Value

A `gt_tbl` object. Export via `gt::gtsave()` to .html, .docx, .pdf, .png, or .rtx. Falls back to `knitr::kable()` if gt is not installed.

See Also

Other workflows: `RecipeWorkflow-class`, `evaluate_cv()`, `print.RecipeWorkflow()`, `publish_workflow()`, `read_workflow()`, `reproduce_workflow()`, `save_workflow()`, `workflow()`, `workflow_from_list()`

Examples

```
svy <- Survey$new(
  data = data.table::data.table(
    x = rnorm(100), g = sample(c("a", "b"), 100, TRUE), w = rep(1, 100)
  ),
  edition = "2023", type = "test", psu = NULL,
  engine = "data.table", weight = add_weight(annual = "w")
)
result <- workflow(
  list(svy), survey::svymean(~x, na.rm = TRUE),
  estimation_type = "annual"
)

if (requireNamespace("gt", quietly = TRUE)) {
  workflow_table(result)
}
```

Index

∗ **anda**
anda_download_microdata, 10
anda_variables, 11
api_get_anda_variables, 14
∗ **api-auth**
api_login, 21
api_logout, 22
api_me, 22
api_refresh_token, 24
api_register, 25
configure_api, 32
∗ **api-comments**
api_comment_recipe, 12
api_comment_workflow, 12
api_delete_comment, 13
api_get_recipe_comments, 15
api_get_workflow_comments, 18
∗ **api-recipes**
api_get_recipe, 14
api_get_recipe_dependents, 16
api_list_recipes, 19
api_publish_recipe, 23
∗ **api-stars**
api_get_recipe_stars, 16
api_get_workflow_stars, 18
api_star_recipe, 26
api_star_workflow, 26
∗ **api-workflows**
api_get_workflow, 17
api_list_workflows, 20
api_publish_workflow, 23
∗ **backends**
get_backend, 40
get_workflow_backend, 49
set_backend, 104
set_workflow_backend, 110
∗ **engine**
get_engine, 42
set_engine, 106
∗ **options**
get_engine, 42
lazy_default, 55
set_engine, 106
set_lazy_processing, 107
set_use_copy, 108
show_engines, 111
use_copy_default, 135
∗ **panel-surveys**
extract_surveys, 35
get_follow_up, 42
get_implantation, 44
PoolSurvey, 64
RotativePanelSurvey, 98
∗ **panel-survey**
extract_surveys, 35
get_follow_up, 42
get_implantation, 44
PoolSurvey, 64
RotativePanelSurvey, 98
∗ **preprocessing**
load_panel_survey, 57
load_survey, 59
∗ **provenance**
print.metasurvey_provenance, 65
print.metasurvey_provenance_diff, 66
provenance, 69
provenance_diff, 70
provenance_to_json, 71
∗ **recipes**
add_recipe, 6
bake_recipes, 27
explore_recipes, 34
get_recipe, 46
harmonize, 50
print.Recipe, 67
publish_recipe, 71
read_recipe, 74

recipe, 76
Recipe-class, 78
save_recipe, 101
steps_to_recipe, 111
* **recipe**
print.Recipe, 67
recipe, 76
Recipe-class, 78
steps_to_recipe, 111
* **steps**
bake_steps, 28
get_steps, 48
step_compute, 112
step_filter, 114
step_join, 115
step_recode, 117
step_remove, 119
step_rename, 121
step_validate, 122
view_graph, 136
* **step**
get_steps, 48
step_compute, 112
step_join, 115
step_recode, 117
steps_to_recipe, 111
view_graph, 136
* **survey-loading**
extract_time_pattern, 37
group_dates, 50
load_panel_survey, 57
load_survey, 59
load_survey_example, 61
validate_time_pattern, 136
* **survey-objects**
cat_design, 29
cat_design_type, 30
get_data, 41
get_metadata, 45
has_design, 52
has_recipes, 53
has_steps, 54
is_baked, 54
set_data, 105
Survey, 124
survey_empty, 129
survey_to_data_frame, 131
survey_to_datatable, 130
survey_to_tibble, 131
* **survey**
bake_recipes, 27
cat_design_type, 30
get_data, 41
get_metadata, 45
get_steps, 48
steps_to_recipe, 111
Survey, 124
survey_empty, 129
survey_to_data_frame, 131
survey_to_datatable, 130
survey_to_tibble, 131
view_graph, 136
workflow, 137
* **tidy-api**
add_category, 5
certify_recipe, 31
default_categories, 32
filter_recipes, 38
filter_workflows, 39
find_workflows_for_recipe, 40
list_recipes, 56
list_workflows, 56
rank_recipes, 73
rank_workflows, 74
recipe_category, 92
recipe_certification, 93
recipe_user, 94
RecipeCategory, 82
RecipeCertification, 84
RecipeUser, 86
remove_category, 95
search_recipes, 103
search_workflows, 104
set_user_info, 107
set_version, 109
* **transpiler**
parse_do_file, 62
parse_stata_labels, 63
transpile_coverage, 132
transpile_stata, 133
transpile_stata_module, 134
* **utils**
add_replicate, 6
add_weight, 8
evaluate_cv, 33
extract_time_pattern, 37

get_recipe, 46
group_dates, 50
lazy_default, 55
load_survey_example, 61
publish_recipe, 71
read_recipe, 74
save_recipe, 101
set_lazy_processing, 107
set_use_copy, 108
validate_time_pattern, 136
∗ **weights**
add_replicate, 6
add_weight, 8
resolve_weight_spec, 97
∗ **workflows**
evaluate_cv, 33
print.RecipeWorkflow, 68
publish_workflow, 72
read_workflow, 75
RecipeWorkflow-class, 88
reproduce_workflow, 96
save_workflow, 102
workflow, 137
workflow_from_list, 139
workflow_table, 140
∗ **workflow**
print.RecipeWorkflow, 68
read_workflow, 75
save_workflow, 102

add_category, 5, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
add_recipe, 6, *27*, *34*, *47*, *51*, *67*, *72*, *75*, *77*, *81*, *101*, *112*
add_replicate, 6, *8, 9*, *97*
add_weight, *7*, 8, *60*, *97*
anda_download_microdata, 10, *11*, *14*
anda_variables, *10*, 11, *14*
api_comment_recipe, 12, *13*, *15*, *18*
api_comment_workflow, *12*, 12, *13*, *15*, *18*
api_delete_comment, *12, 13*, 13, *15*, *18*
api_get_anda_variables, *10, 11*, 14
api_get_recipe, 14, *16*, *20*, *23*
api_get_recipe_comments, *12, 13*, 15, *18*
api_get_recipe_dependents, *15*, 16, *20*, *23*
api_get_recipe_stars, 16, *19*, *26, 27*
api_get_workflow, 17, *20*, *24*
api_get_workflow_comments, *12, 13*, *15*, 18
api_get_workflow_stars, *17*, 18, *26, 27*
api_list_recipes, *15, 16*, 19, *23*
api_list_workflows, *17*, 20, *24*
api_login, 21, *22*, *24, 25*, *32*
api_logout, *21, 22*, 22, *24, 25*, *32*
api_me, *21, 22*, 22, *24, 25*, *32*
api_publish_recipe, *15, 16*, *20*, 23
api_publish_workflow, *17*, *20*, 23
api_refresh_token, *21, 22*, 24, *25*, *32*
api_register, *21, 22*, *24*, 25, *32*
api_star_recipe, *17*, *19*, 26, *27*
api_star_workflow, *17*, *19*, *26*, 26

bake_recipes, *6*, 27, *34*, *47*, *50, 51*, *67*, *72*, *75*, *77*, *81*, *101*, *112*, *129*
bake_steps, 28, *49*, *114–116*, *119, 120*, *122, 123*, *137*
bake_steps(), *113*, *115, 116*, *118*, *120*, *122*

cat_design, 29, *30*, *41*, *45*, *53–55*, *105*, *129–132*
cat_design_type, *29*, 30, *41*, *45*, *53–55*, *105*, *129–132*
cat_recipes, *129*
certify_recipe, *5*, 31, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
configure_api, *21, 22*, *24, 25*, 32
confint, *137*

default_categories, *5*, *31*, 32, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*

evaluate_cv, 33, *68*, *72*, *76*, *91*, *97*, *102*, *138, 139*, *141*
explore_recipes, *6*, *27*, 34, *47*, *51*, *67*, *72*, *75*, *77*, *81*, *101*, *112*
extract_surveys, 35, *43*, *45*, *65*, *100*
extract_time_pattern, 37, *50*, *58*, *60*, *62*, *136*

filter_recipes, *5*, *31*, *33*, 38, *39, 40*, *51*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
filter_workflows, *5*, *31*, *33*, *38*, 39, *40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*

find_workflows_for_recipe, *5*, *31*, *33*, *38, 39*, 40, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*

get_backend, 40, *49*, *105*, *110*
get_data, *29, 30*, 41, *45*, *53–55*, *105*, *129–132*
get_engine, 42, *55*, *106, 107*, *109*, *111*, *135*
get_follow_up, *36*, 42, *45*, *65*, *100*
get_implantation, *36*, *43*, 44, *65*, *100*
get_metadata, *29, 30*, *41*, 45, *53–55*, *105*, *129–132*
get_recipe, *6*, *27*, *34*, 46, *51*, *60*, *67*, *72*, *75*, *77*, *81*, *101*, *112*
get_steps, *28*, 48, *114–116*, *119, 120*, *122, 123*, *137*
get_workflow_backend, *41*, 49, *105*, *110*
group_dates, *37*, 50, *58*, *60*, *62*, *136*

harmonize, *6*, *27*, *34*, *47*, 50, *67*, *72*, *75*, *77*, *81*, *101*, *112*
has_design, *29, 30*, *41*, *45*, 52, *53–55*, *105*, *129–132*
has_recipes, *29, 30*, *41*, *45*, *53*, 53, *54, 55*, *105*, *129–132*
has_steps, *29, 30*, *41*, *45*, *53*, 54, *55*, *105*, *129–132*

is_baked, *29, 30*, *41*, *45*, *53, 54*, 54, *105*, *129–132*

knitr::kable(), *141*

lazy_default, *42*, 55, *106, 107*, *109*, *111*, *135*
list_recipes, *5*, *31*, *33*, *38–40*, 56, *57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
list_workflows, *5*, *31*, *33*, *38–40*, *56*, 56, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
load_panel_survey, *36, 37*, *43*, *45*, *50*, 57, *60*, *62*, *136*
load_survey, *7*, *9*, *37*, *47*, *50*, *58*, 59, *62*, *136*
load_survey_example, *37*, *50*, *58*, *60*, 61, *136*

parse_do_file, 62, *63*, *133, 134*
parse_stata_labels, *63*, 63, *133, 134*
PoolSurvey, *36*, *43*, *45*, *50, 51*, 64, *100*, *138*
print.metasurvey_provenance, 65, *66*, *69–71*
print.metasurvey_provenance_diff, *66*, 66, *69–71*
print.Recipe, *6*, *27*, *34*, *47*, *51*, 67, *72*, *75*, *77*, *81*, *101*, *112*
print.RecipeWorkflow, *33*, 68, *72*, *76*, *91*, *97*, *102*, *138, 139*, *141*
provenance, *66*, 69, *70, 71*
provenance(), *125*
provenance_diff, *66*, *69*, 70, *71*
provenance_to_json, *66*, *69, 70*, 71
publish_recipe, *6*, *27*, *34*, *47*, *51*, *67*, 71, *75*, *77*, *81*, *101*, *112*
publish_workflow, *33*, *68*, 72, *76*, *91*, *97*, *102*, *138, 139*, *141*

rank_recipes, *5*, *31*, *33*, *38–40*, *56, 57*, 73, *74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
rank_workflows, *5*, *31*, *33*, *38–40*, *56, 57*, *73*, 74, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
read_recipe, *6*, *27*, *34*, *47*, *51*, *67*, *72*, 74, *77*, *81*, *101*, *112*
read_workflow, *33*, *68*, *72*, 75, *91*, *97*, *102*, *138, 139*, *141*
Recipe, *77*, *125*
Recipe *(*Recipe-class*)*, 78
recipe, *6*, *27*, *34*, *47*, *51*, *67*, *72*, *75*, 76, *81*, *101*, *112*
Recipe-class, 78
recipe_category, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, 92, *93*, *95, 96*, *103, 104*, *108, 109*
recipe_certification, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92*, 93, *95, 96*, *103, 104*, *108, 109*
recipe_user, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, 94, *96*, *103, 104*, *108, 109*
RecipeCategory, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, 82, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
RecipeCertification, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, 84, *88*, *92, 93*, *95, 96*, *103, 104*, *108, 109*
RecipeUser, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, 86, *92–96*, *103, 104*, *107–109*
RecipeWorkflow, *72*

RecipeWorkflow *(RecipeWorkflow-class)*, 88
RecipeWorkflow-class, 88
remove_category, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95*, 95, *103, 104*, *108, 109*
reproduce_workflow, *33*, *68*, *72*, *76*, *91*, 96, *102*, *138, 139*, *141*
resolve_weight_spec, *7*, *9*, 97
RotativePanelSurvey, *36*, *43*, *45*, *65*, 98, *115*

save_recipe, *6*, *27*, *34*, *47*, *51*, *67*, *72*, *75*, *77*, *81*, 101, *112*
save_workflow, *33*, *68*, *72*, *76*, *91*, *97*, 102, *138, 139*, *141*
search_recipes, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, 103, *104*, *108, 109*
search_workflows, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103*, 104, *108, 109*
set_backend, *41*, *49*, *103*, 104, *110*
set_data, *29, 30*, *41*, *45*, *53–55*, 105, *129–132*
set_engine, *42*, *55*, 106, *107*, *109*, *111*, *135*
set_lazy_processing, *42*, *55*, *106*, 107, *109*, *111*, *135*
set_use_copy, *42*, *55*, *106, 107*, 108, *111*, *135*
set_user_info, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, 107, *109*
set_version, *5*, *31*, *33*, *38–40*, *56, 57*, *73, 74*, *84*, *86*, *88*, *92, 93*, *95, 96*, *103, 104*, *108*, 109
set_workflow_backend, *41*, *49*, *72*, *105*, 110
show_engines, *42*, *55*, *106, 107*, *109*, 111, *135*
step_compute, *28*, *49*, 112, *115, 116*, *119, 120*, *122, 123*, *137*
step_filter, *28*, *49*, *114*, 114, *116*, *119, 120*, *122, 123*, *137*
step_join, *28*, *49*, *114, 115*, 115, *119, 120*, *122, 123*, *137*
step_recode, *28*, *49*, *114–116*, 117, *120*, *122, 123*, *137*
step_remove, *28*, *49*, *114–116*, *119*, 119, *122, 123*, *137*
step_remove(), *122*
step_rename, *28*, *49*, *114–116*, *119, 120*, 121, *123*, *137*
step_validate, *28*, *49*, *114–116*, *119, 120*, *122*, 122, *137*
steps_to_recipe, *6*, *27*, *34*, *47*, *51*, *67*, *72*, *75*, *77*, *81*, *101*, 111
Survey, *29, 30*, *41*, *45*, *51*, *53–55*, *69*, *105*, *115*, 124, *130–132*
survey::svydesign(), *60*
survey_empty, *29, 30*, *41*, *45*, *53–55*, *105*, *129*, 129, *130–132*
survey_to_data.table *(survey_to_datatable)*, 130
survey_to_data_frame, *29, 30*, *41*, *45*, *53–55*, *105*, *129, 130*, 131, *132*
survey_to_datatable, *29, 30*, *41*, *45*, *53–55*, *105*, *129, 130*, 130, *131, 132*
survey_to_tibble, *29, 30*, *41*, *45*, *53–55*, *105*, *129–131*, 131
svrepdesign, *7*
svyby, *138*
svymean, *138*
svyratio, *138*
svytotal, *138*

tbl_df, *132*
transpile_coverage, *63*, 132, *134*
transpile_stata, *63*, *133*, 133, *134*
transpile_stata_module, *63*, *133, 134*, 134

use_copy_default, *42*, *55*, *106, 107*, *109*, *111*, 135
use_copy_default(), *115*

validate_time_pattern, *37*, *50*, *58*, *60*, *62*, 136
view_graph, *28*, *49*, *114–116*, *119, 120*, *122, 123*, 136

workflow, *9*, *33*, *36*, *43*, *45*, *50, 51*, *68*, *72*, *76*, *91*, *97*, *102*, 137, *139*, *141*
workflow(), *69*, *140*
workflow_from_list, *33*, *68*, *72*, *76*, *91*, *97*, *102*, *138*, 139, *141*
workflow_table, *33*, *68*, *72*, *76*, *91*, *97*, *102*, *138, 139*, 140