

Package: cohort (via r-universe)

September 26, 2026

Title Build Analytic Cohorts with Full Provenance

Version 2026.8.21

Description A framework for building analytic cohorts with full provenance. Supports branched cohort trees, per-step exclusion logging with human-readable reasons, schema validation at branch boundaries, cached derived artifacts on each cohort, and CONSORT diagram generation. The framework cleanly separates cohort definition from analysis: it produces analytic data tables that downstream packages can consume. A shared-base storage strategy keeps memory flat as the cohort tree grows.

License MIT + file LICENSE

URL <https://www.rwhite.no/cohort/>,
<https://github.com/raubreywhite/cohort>

BugReports <https://github.com/raubreywhite/cohort/issues>

Encoding UTF-8

Config/Needs/website raubreywhite/rwtemplate

Depends R (>= 3.5.0)

Imports data.table, digest, grDevices, grid, gridExtra, R6, tools,
utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Repository <https://raubreywhite.r-universe.dev>

Date/Publication 2026-08-26 16:17:36 UTC

RemoteUrl <https://github.com/raubreywhite/cohort>

RemoteRef HEAD

RemoteSha 862e6548b2c20c37a6b6f2f633ff7957e7fba68a

Contents

CohortPipeline	2
Index	11

CohortPipeline	<i>R6 Class for Cohort Construction with Provenance</i>
----------------	---

Description

CohortPipeline builds analytic cohorts as a tree of named branches with full exclusion provenance. Each branch derives from a parent branch. It applies a sequence of named exclusion rules to that parent. The class records every exclusion. Each record holds the reason, the predicate that produced it, and the number of subjects affected. The object can therefore drive a CONSORT diagram. It is also the auditable record of how the analytic dataset was constructed.

Cohort construction stays strictly upstream of analysis. The class produces analytic data tables that downstream code can consume. See `vignette("cohort", package = "cohort")` for a worked example.

Storage strategy

A CohortPipeline stores one shared base data table. For each branch it also stores a small integer status vector, with one entry per row. The vector identifies which rows are included, and which step excluded the other rows. Branching is therefore $O(n)$ in the number of rows of the base table. Branching never copies the data values, so deep cohort trees stay flat in memory.

Freeze rule

A cohort becomes **frozen** the first time either:

1. another cohort branches from it (via `$new_cohort(from = X)`), or
2. an artifact is set on it (via `$set_artifact(from = X)`).

After a cohort freezes, `$exclude_and_track()` on it errors. The rule guarantees that a cohort's name maps to exactly one definition forever. Once children depend on a cohort, its exclusion list is fixed. Any cached artifact stays consistent with the included rows that produced it. The practical workflow is: apply all exclusions on a cohort, then branch from it or attach artifacts. Multi-way forks are unaffected. You can branch a frozen cohort as many times as you like.

Mutation contract

- `CohortPipeline$new(dt)` makes a defensive copy of `dt` once. The class never modifies the user's data table.
- `$get_included(cohort)` returns an independent copy. The caller may mutate it freely. The change does not affect any other cohort or the shared base table.
- `$set_artifact()` always passes an independent copy of the data table to the callback. The callback may mutate it freely.
- `$get_everyone(cohort)` returns an independent copy with a `.cohort_status` column reconstructed from the branch's status vector.

Public API

- `CohortPipeline$new(dt, cache_file, label)` – construct a pipeline with a shared base table installed as the root cohort. With `cache_file`, restore from a prior run if the file exists.
- `$new_cohort(name, from, label)` – branch from an existing cohort.
- `$exclude_and_track(branch, reason, expr_str)` – apply a string-form predicate and log the exclusion.
- `$set_artifact(name, from, fn, argset)` – cache a derived object on a cohort. `fn` may be `function(dt, sib)` or `function(dt, sib, argset)`.
- `$get_included(cohort)` – included rows of a cohort.
- `$get_everyone(cohort)` – full-cohort view with a reconstructed `.cohort_status` column.
- `$get_artifact(cohort, name)` – retrieve a cached artifact.
- `$n_included(cohort), $n_total()` – row counts.
- `$list_cohorts(), $list_artifacts(cohort), $list_schemas()` – inventories.
- `$declare_schema(branch, schema, from), $validate()` – column contracts.
- `$consort()` – long-form exclusion log across all branches.
- `$draw_consort_panels(panels, file)` – render CONSORT diagrams.
- `$save(file), $invalidate(cohort, artifact)` – incremental cache persistence and manual cache invalidation.
- `$print()` – concise text summary of the cohort tree.

Predicate strings

You pass exclusion predicates as strings (`expr_str`). The class parses each string with `parse(text = expr_str)`. It evaluates the string against the included subset of the base table. A predicate can therefore assume that earlier exclusions already removed invalid rows. The class treats NA predicate results as FALSE, so it keeps those rows. The exclusion log stores the original string verbatim. Cohort definitions therefore stay serializable and auditable.

Methods

Public methods:

- `CohortPipeline$new()`
- `CohortPipeline$declare_schema()`
- `CohortPipeline$validate()`
- `CohortPipeline$list_schemas()`
- `CohortPipeline$get_schemas()`
- `CohortPipeline$new_cohort()`
- `CohortPipeline$exclude_and_track()`
- `CohortPipeline$set_artifact()`
- `CohortPipeline$get_included()`
- `CohortPipeline$get_everyone()`
- `CohortPipeline$get_artifact()`
- `CohortPipeline$n_included()`
- `CohortPipeline$n_total()`

- `CohortPipeline$list_cohorts()`
- `CohortPipeline$list_artifacts()`
- `CohortPipeline$consort()`
- `CohortPipeline$draw_consort_panels()`
- `CohortPipeline$plot()`
- `CohortPipeline$print()`
- `CohortPipeline$save()`
- `CohortPipeline$invalidate()`

`CohortPipeline$new()`: Create a new CohortPipeline. If `cache_file` is set and the file exists, the constructor restores the pipeline from that snapshot. The constructor then uses `dt` only as a sanity check. Its dimensions and column names must match the cached base table. Otherwise the constructor installs `dt` as the root cohort.

Usage:

```
CohortPipeline$new(
  dt = NULL,
  cache_file = NULL,
  label = NULL,
  auto_validate = FALSE
)
```

Arguments:

`dt` A `data.table` to install as the root cohort. Required on cold construction; optional on warm cache load.

`cache_file` Optional character path. When you supply a path, the pipeline enables an incremental cache. If the file exists, the constructor restores the pipeline from it. Subsequent operations then replay the recorded log on cache hits, and only divergent steps recompute. If the file does not exist, the constructor builds fresh state, and `$save()` writes to this path. In a script, put `on.exit(cp$save(), add = TRUE)` near the top.

`label` Optional character. Display label for the root cohort (used in CONSORT diagrams and `$list_cohorts()`). Defaults to "Cohort participants". A warm cache load refreshes the label silently, so you may change the label between runs.

`auto_validate` Logical. When TRUE, the pipeline calls `$validate()` after every `$new_cohort()` and `$set_artifact()` call. A schema mismatch then stops at the failure site. It does not accumulate until the next manual `$validate()`. Defaults to FALSE.

Returns: A new CohortPipeline instance.

`CohortPipeline$declare_schema()`: Declare a column-type / level / NA contract for a branch. Validation runs only when you call `$validate()`. With `auto_validate = TRUE` at construction, the pipeline calls `$validate()` for you.

Usage:

```
CohortPipeline$declare_schema(branch, schema = NULL, from = NULL)
```

Arguments:

`branch` Character. Branch name to attach the schema to.

`schema` Named list. Each element describes one column with fields:

- type: one of "integer", "numeric", "factor", "logical", "Date", "character".
- levels (factor only): expected levels() vector.
- na: if FALSE, the column MUST contain no NAs.

from Optional character. If you supply from, the new schema starts as a copy of the schema attached to from. The entries in schema then merge on top.

Returns: The pipeline (invisibly).

CohortPipeline\$validate(): Validate every declared schema against the included rows of its branch. Throws one error that lists every mismatch found.

Usage:

```
CohortPipeline$validate()
```

Returns: The pipeline (invisibly), if validation passes.

CohortPipeline\$list_schemas(): Tabulate the names and column counts of all declared schemas.

Usage:

```
CohortPipeline$list_schemas()
```

Returns: A data.table with columns branch and n_cols.

CohortPipeline\$get_schemas(): Return the raw schema list for inspection.

Usage:

```
CohortPipeline$get_schemas()
```

Returns: A named list of schemas.

CohortPipeline\$new_cohort(): Create a new cohort branched from an existing cohort. The new cohort starts identical to its parent at the moment of branching. Later exclusions in the parent do not propagate to the child.

Usage:

```
CohortPipeline$new_cohort(name, from, label = NULL)
```

Arguments:

name Character. Name of the new cohort.

from Character. Name of the parent cohort.

label Optional character. Display label for the cohort (used in CONSORT diagrams and \$list_cohorts()); defaults to name. A cache replay may refresh it silently.

Returns: The pipeline (invisibly).

CohortPipeline\$exclude_and_track(): Apply an exclusion predicate to a cohort. Record the result on the exclusion log. The method evaluates the predicate against the included subset of the base table. It excludes every row where the predicate evaluates to TRUE, with the supplied reason. The method treats NA predicate results as FALSE.

Usage:

```
CohortPipeline$exclude_and_track(branch, reason, expr_str)
```

Arguments:

`branch` Character. Cohort to apply the exclusion to.
`reason` Character. Human-readable reason recorded on the log.
`expr_str` Character. R expression as a string (parsed with `parse(text = ...)`). For example
`"is.na(age) | age < 18"`.

Returns: The pipeline (invisibly).

`CohortPipeline$set_artifact()`: Compute and cache a derived artifact on a cohort.

`fn` may have either the legacy 2-argument signature `function(dt, sib)` or the 3-argument signature `function(dt, sib, argset)`. The 3-argument form pairs with the `argset` parameter to make the cache contract explicit. The cache key is `(name, from, body(fn), argset)`. The artifact recomputes only when one of those changes. With the 2-argument form, the method invokes `fn` normally, but `argset` does not join the cache key. Use the 2-argument form for one-off scripts. You **SHOULD NOT** use it together with `cache_file`.

The cache key uses `body(fn)` literally. If `fn` calls a helper that you change, the cache cannot detect the change. Either put the helper's output or a version tag in `argset`, or call `$invalidate()` to force a recompute.

Usage:

```
CohortPipeline$set_artifact(name, from, fn, argset = NULL)
```

Arguments:

`name` Character. Artifact name (must be unique on the cohort).

`from` Character. Cohort to attach the artifact to.

`fn` Function with signature `function(dt, sib)` or `function(dt, sib, argset)`. The return value becomes the artifact.

`argset` Optional named list. Explicit data dependencies of `fn` (e.g. `list(outcomes = cfg$outcomes)`); participates in the cache key. Use the 3-argument `fn` signature to read these out.

Returns: The pipeline (invisibly).

`CohortPipeline$get_included()`: Return an independent copy of the included rows of a cohort. You may mutate the returned `data.table` freely. The change does not affect the shared base table or any other cohort.

Usage:

```
CohortPipeline$get_included(cohort)
```

Arguments:

`cohort` Character. Cohort name.

Returns: A `data.table`.

`CohortPipeline$get_everyone()`: Return a copy of the full base table with a `.cohort_status` column reconstructed from this branch's exclusion history. Included rows carry the label "included". Excluded rows carry the reason of the first exclusion that caught them.

Usage:

```
CohortPipeline$get_everyone(cohort)
```

Arguments:

`cohort` Character. Cohort name.

Returns: A `data.table` of the same height as the base table, with one extra column `.cohort_status`.

`CohortPipeline$get_artifact()`: Retrieve a cached artifact from a cohort.

Usage:

`CohortPipeline$get_artifact(cohort, name)`

Arguments:

`cohort` Character. Cohort name.

`name` Character. Artifact name.

Returns: The cached artifact (any type).

`CohortPipeline$n_included()`: Number of included rows in a cohort.

Usage:

`CohortPipeline$n_included(cohort)`

Arguments:

`cohort` Character. Cohort name.

Returns: Integer.

`CohortPipeline$n_total()`: Total number of rows in the shared base table.

Usage:

`CohortPipeline$n_total()`

Returns: Integer.

`CohortPipeline$list_cohorts()`: Tabulate every cohort with its parent, sizes and number of own exclusion steps and artifacts.

Usage:

`CohortPipeline$list_cohorts()`

Returns: A `data.table` with one row per cohort and columns `name`, `parent`, `n_total`, `n_included`, `n_excluded`, `n_own_steps`, `n_artifacts`, `frozen`. An empty pipeline returns a zero-row table carrying those same columns.

`CohortPipeline$list_artifacts()`: Names of cached artifacts attached to a cohort.

Usage:

`CohortPipeline$list_artifacts(cohort)`

Arguments:

`cohort` Character. Cohort name.

Returns: Character vector. `character(0)` when the cohort has no artifacts.

`CohortPipeline$consort()`: Long-form table of exclusion log entries across all cohorts. Each cohort contributes only its own exclusion steps. A step that a cohort inherits from its parent at branch time appears under the parent only. The table does not duplicate it.

Usage:

`CohortPipeline$consort()`

Returns: A data.table with columns branch, parent, step, reason, expr_str, n_excluded, n_remaining.

CohortPipeline\$draw_consort_panels(): Render one or more CONSORT panels for cohort flows. Each panel walks a sequence of cohort names. It lumps the named cohorts' exclusion steps into bullet blocks.

Most users want \$plot() instead. \$plot() auto-discovers every root-to-leaf path in the tree and lays them out automatically. \$draw_consort_panels() is the manual route for custom layouts and labels.

Usage:

```
CohortPipeline$draw_consort_panels(
  panels,
  file = NULL,
  ncol = NULL,
  width = NULL,
  height = NULL,
  text_width = 40,
  title_fontsize = 14
)
```

Arguments:

panels A named list. Each element takes one of two forms:

- a character vector of cohort names, which is the panel's main flow; or
- a list with a flow component (character) and an optional side_branches component. side_branches is a named character vector of identity-only branches that merge into the spine.

file Optional character path. If you supply a path, the method writes the rendered plot to a .pdf or .png file. Otherwise the method draws the plot on the active device.

ncol Optional integer. Number of panels per row.

width, height Optional numeric (inches). File dimensions.

text_width Integer. Wrap width for box text.

title_fontsize Numeric. Title fontsize for each panel.

Returns: A list of grobs (invisibly).

CohortPipeline\$plot(): Plot a CONSORT diagram of the cohort tree.

With no arguments, plots one panel per cohort. Each panel walks the root-to-cohort path automatically. It uses cohort names as box labels. With one or more cohort names, plots only those.

This is the default convenience entry point. Use \$draw_consort_panels() for custom labels or layouts.

Usage:

```
CohortPipeline$plot(
  cohorts = NULL,
  file = NULL,
  ncol = NULL,
  width = NULL,
  height = NULL,
```

```

    text_width = 40,
    title_fontsize = 14
  )

```

Arguments:

cohorts Optional character vector of cohort names. If you omit it, the method plots every cohort.

file Optional .pdf/.png path. If you supply a path, the method writes the plot to that file. Otherwise the method draws the plot on the active device.

ncol, width, height, text_width, title_fontsize Optional layout overrides; see `$draw_consort_panels()`.

Returns: A list of grobs (invisibly).

CohortPipeline\$print(): Concise text summary of the cohort tree, exclusion counts, and attached artifacts.

Usage:

```
CohortPipeline$print(...)
```

Arguments:

... Unused.

Returns: The pipeline (invisibly).

CohortPipeline\$save(): Persist the pipeline to its `cache_file` (set at construction). The next `CohortPipeline$new(dt, cache_file = ...)` with the same file restores the saved state. Re-issued operations then replay from the cache. Only divergent operations recompute. The method is idempotent beyond the file write.

Usage:

```
CohortPipeline$save(file = NULL)
```

Arguments:

file Optional override for the cache file path.

Returns: The pipeline (invisibly).

CohortPipeline\$invalidate(): Manually invalidate a cached cohort (drops the cohort and every descendant) or a single artifact. Use this method when you change a helper function that a `set_artifact` fn calls. The cache key (`body(fn)` plus `argset`) cannot detect that change automatically.

Usage:

```
CohortPipeline$invalidate(cohort, artifact = NULL)
```

Arguments:

cohort Character. Cohort to invalidate.

artifact Optional character. If you supply a name, the method drops only that artifact, and any artifacts declared after it on the same cohort.

Returns: The pipeline (invisibly).

See Also

`vignette("cohort")` for a worked example that branches a cohort, caches derived artifacts, declares a column schema and draws a CONSORT diagram.

Examples

```
library(data.table)
d <- data.table(
  id = 1:10,
  age = c(17, 22, 35, NA, 41, 28, 19, 16, 67, 50),
  sex = c("F", "M", "F", "F", NA, "M", "M", "F", "F", "M")
)

cp <- CohortPipeline$new(d)

# Root-level exclusions on the shared base
cp$exclude_and_track("root", "Missing sex",      "is.na(sex)")
cp$exclude_and_track("root", "Missing age",       "is.na(age)")
cp$exclude_and_track("root", "Under 18",          "age < 18")

# Branch into an "adults_female" cohort
cp$new_cohort("adults_female", from = "root")
cp$exclude_and_track("adults_female", "Not female", "sex != 'F'")

# Cache a derived artifact on the cohort
cp$set_artifact("mean_age", from = "adults_female",
  fn = function(dt, sib) mean(dt$age))

cp$list_cohorts()
cp$consort()
cp$get_artifact("adults_female", "mean_age")
```

Index

CohortPipeline, [2](#)