Package: autocodebook (via r-universe)

June 8, 2026

Title Automatic Codebook and Tracking for 'Spark' and 'dplyr' Pipelines

Version 0.1.0

Description Wraps 'dplyr' verbs (mutate, summarise, filter) to automatically capture variable metadata (type, source columns, categories, and source code), producing a codebook and eligibility tracking table with zero manual documentation. Works with both 'sparklyr' (tbl_spark) and local data frames. Adds big-data optimizations (caching, assume-unique counting, checkpointing) and a standardized report module with an eligibility flowchart, editable codebook export (HTML, DOCX, XLSX), and cross-sectional or longitudinal variable inspection. The eligibility flowchart follows the CONSORT statement (Schulz, Altman and Moher (2010) <doi:10.1136/bmj.c332>) and the reporting of observational cohort studies follows the STROBE recommendations (von Elm and others (2007) <doi:10.1371/journal.pmed.0040296>).

License MIT + file LICENSE

URL https://github.com/patriciafortesm/autocodebook

BugReports https://github.com/patriciafortesm/autocodebook/issues

Encoding UTF-8

Imports dplyr (>= 1.1.0), rlang (>= 1.0.0), tibble, gt, grid

Suggests sparklyr, dbplyr, testthat (>= 3.0.0), tidyplots, ggplot2, patchwork, rmarkdown, knitr, officer, flextable, openxlsx, scales, rvg, devEMF, svglite

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Patricia Fortes C. de Macedo [aut, cre]

Maintainer Patricia Fortes C. de Macedo <macedopatriciafortes@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-08 18:00:19 UTC

RemoteUrl https://github.com/cran/autocodebook

RemoteRef HEAD

RemoteSha 355ee9bcca4ed3ffe381a6baa2b63faedaeba670

Contents

auto_filter . . . 2
auto_mutate . . . 3
auto_summarise . . . 4
cb_checkpoint . . . 5
cb_export . . . 5
cb_get . . . 6
cb_init . . . 6
cb_register . . . 7
cb_render . . . 8
cb_reset . . . 8
cb_set_default_cache . . . 9
cb_set_verbose . . . 9
flow_diagram . . . 10
flow_diagram_export . . . 11
flow_get . . . 12
flow_reset . . . 13
flow_table . . . 13
generate_report . . . 14
track_export . . . 16
track_get . . . 16
track_outcomes . . . 17
track_render . . . 17
track_reset . . . 18
track_split . . . 18
track_step . . . 19

Index **20**

`auto_filter` *Filter with automatic tracking*

Description

Works exactly like `dplyr::filter()`, but also logs a tracking step recording how many unique IDs remain after the filter.

Usage

```
auto_filter(
  .data,
  step = "",
  description = "",
  ...,
  cache = NULL,
  assume_unique = FALSE
)
```

Arguments

.data	A Spark DataFrame or local data frame.
step	Character label for this filtering step.
description	Character description of the filter.
...	Filter conditions, same syntax as dplyr::filter().
cache	Logical or NULL (named-only). If TRUE, materializes the result with cb_checkpoint() after filtering - useful in long Spark pipelines. If NULL, falls back to the session default (set via cb_init() or cb_set_default_cache()). Default: NULL.
assume_unique	Logical (named-only). Passed to track_step(). Set TRUE only when you are certain the ID column has no duplicates at this stage. Default: FALSE.

Details

The signature mirrors v0.1.0 for full backward compatibility: step and description come first (so existing positional calls keep working), then ... for the filter conditions, and finally the new big-data options (cache, assume_unique) which **must be passed by name**.

Value

The filtered data frame.

---

auto_mutate *Mutate with automatic codebook registration*

Description

Works exactly like dplyr::mutate(), but also captures each expression and registers the resulting variable in the codebook. Type, source columns, categories, and source code are inferred automatically - you only need to provide human-readable labels.

Usage

```
auto_mutate(.data, labels = list(), block = "", ...)
```

Arguments

.data	A Spark DataFrame (tbl_spark) or local data frame.
labels	Named list mapping variable names to labels (descriptions). Variables not in this list get their own name as label.
block	Optional character label for the pipeline block/section (e.g. "Demographic variables"). Groups variables in the codebook.
...	Named expressions, same syntax as dplyr::mutate().

Value

The transformed data frame (same class as input).

---

auto_summarise *Summarise with automatic codebook registration*

Description

Works exactly like dplyr::summarise(), but also captures each expression and registers the resulting variable in the codebook.

Usage

```
auto_summarise(.data, labels = list(), block = "", ..., .groups = "drop")
```

Arguments

.data	A Spark DataFrame (tbl_spark) or local data frame.
labels	Named list mapping variable names to labels (descriptions). Variables not in this list get their own name as label.
block	Optional character label for the pipeline block/section (e.g. "Demographic variables"). Groups variables in the codebook.
...	Named expressions, same syntax as dplyr::mutate().
.groups	Grouping behavior after summarise. Default: "drop".

Value

The summarised data frame.

cb_checkpoint *Checkpoint a Spark DataFrame*

Description

Forces materialization of a lazy Spark plan. Useful in long pipelines where query plans get too deep and the optimizer starts re-computing upstream steps. For local data frames, this is a no-op.

Usage

```
cb_checkpoint(sdf, name = NULL, mode = c("memory", "disk", "register"))
```

Arguments

sdf	A Spark DataFrame (tbl_spark) or local data frame.
name	Optional. Name to register the checkpoint under (Spark only). If NULL, a temporary name is generated.
mode	Character. One of "memory" (cache in memory, fastest), "disk" (sdf_checkpoint via disk, more durable), or "register" (just register as temp table without caching). Default: "memory".

Value

The (possibly materialized) data frame.

cb_export *Export codebook to file*

Description

Supports multiple formats based on file extension:

- .html - rendered gt table (presentation)
- .csv - raw tibble (programmatic reuse)
- .docx - editable Word table (paper supplements, presentations)
- .xlsx - editable spreadsheet with filters

Usage

```
cb_export(path, variables = NULL, ...)
```

Arguments

path	File path. Extension determines format. There is no default: the destination must be supplied explicitly (e.g. a file under tempdir() or a directory chosen by the user).
variables	Optional character vector. If provided, exports only these variables (in the given order). Default: all.
...	Additional arguments passed to cb_render() for HTML (e.g. show_code).

Details

For .docx and .xlsx, you can pass variables = c(...) to export only a subset (useful for paper supplements / presentations).

Value

Invisible path.

---

cb_get *Get the current codebook as a tibble*

Description

Get the current codebook as a tibble

Usage

```
cb_get()
```

Value

A tibble with all registered variables.

---

cb_init *Initialize autocodebook session*

Description

Resets the codebook and tracking logs and sets the ID column used for counting unique individuals in track_step().

Usage

```
cb_init(id_col = "id", verbose = FALSE, default_cache = FALSE)
```

Arguments

id_col	Character. Name of the unique identifier column. Default: "id".
verbose	Logical. If TRUE, prints diagnostic messages from track_step(), auto_filter(), and cb_checkpoint(). Default: FALSE (matches v0.1.0 behavior - silent).
default_cache	Logical. If TRUE, big-data verbs cache intermediate results in Spark by default. Can be overridden per-call. Default: FALSE.

Value

Invisible NULL.

Examples

```
cb_init(id_col = "id_cidacs_pop100_v2")
cb_init(id_col = "id", verbose = TRUE, default_cache = TRUE)
```

---

cb_register *Manually register a variable in the codebook*

Description

Use when auto_mutate/auto_summarise doesn't apply - e.g. variables created via window_order + row_number in a separate pipeline step.

Usage

```
cb_register(
  var,
  label,
  type = NULL,
  source = "",
  categories = "",
  code = "",
  block = ""
)
```

Arguments

var	Variable name (character).
label	Human-readable description.
type	Optional. If NULL, defaults to "character".
source	Optional. Source column(s).
categories	Optional. Category descriptions.
code	Optional. Code that generated the variable.
block	Optional. Pipeline block label.

Value

Invisible NULL.

---

cb_render *Render the codebook as a gt table*

Description

Render the codebook as a gt table

Usage

```
cb_render(group_by_block = TRUE, show_code = TRUE)
```

Arguments

group_by_block	Logical. If TRUE and blocks are defined, groups rows by block. Default: TRUE.
show_code	Logical. Show the "code" column? Default: TRUE.

Value

A gt object.

---

cb_reset *Reset the codebook (clear all entries)*

Description

Reset the codebook (clear all entries)

Usage

```
cb_reset()
```

Value

Invisible NULL.

cb_set_default_cache *Toggle default caching for big-data verbs*

Description

Toggle default caching for big-data verbs

Usage

```
cb_set_default_cache(default_cache = TRUE)
```

Arguments

default_cache	Logical.

Value

Invisible previous value.

cb_set_verbose *Toggle verbose diagnostic messages*

Description

Controls whether track_step(), auto_filter() and cb_checkpoint() print diagnostic messages (n removed, elapsed time, etc.).

Usage

```
cb_set_verbose(verbose = TRUE)
```

Arguments

verbose	Logical.

Value

Invisible previous value.

`flow_diagram` *Draw the eligibility flow as a CONSORT-style flowchart*

Description

Renders a publication-ready CONSORT-style diagram (a 'ggplot' object) from the information captured during the pipeline. The layout is computed automatically, so no manual positioning is needed:

Usage

```
flow_diagram(
  title = "Eligibility flowchart",
  show_exclusions = TRUE,
  box_fill = "white",
  border_col = "grey25",
  text_size = 3.5
)
```

Arguments

`title`	Plot title. Default: "Eligibility flowchart".
`show_exclusions`	Logical. Show the side box listing the linear exclusion steps? Default: TRUE.
`box_fill`	Fill colour of the boxes. Default: "white".
`border_col`	Border colour of the boxes. Default: "grey25".
`text_size`	Base text size for the box labels. Default: 3.5.

Details

- A vertical trunk built from the linear eligibility steps (recorded by `auto_filter()` / `track_step()`): the cohort baseline at the top, a single side box listing every exclusion and its count, and the eligible cohort below.
- One column of boxes per subgroup leaf of the flow tree (built with `track_split()`), shown side by side under the eligible cohort.
- Outcome boxes (added with `track_outcomes()`) stacked beneath each subgroup column, one row per outcome, with the event count and percent.

Either part is optional: with only linear steps you get the trunk; with only a split tree you get the cohort box and its subgroup columns. The function reads the current session state and writes nothing to disk.

Requires the 'ggplot2' package (a soft dependency).

Value

A 'ggplot' object, or invisible NULL if there is nothing to draw.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  cb_init(id_col = "id_indiv")
  df <- data.frame(
    id_indiv  = sprintf("ID%04d", 1:400),
    sgm       = sample(c(0L, 1L), 400, replace = TRUE, prob = c(0.95, 0.05)),
    self_harm = sample(c(0L, 1L), 400, replace = TRUE, prob = c(0.97, 0.03))
  )
  df <- track_split(df, by = "sgm", label = "SGM status",
                    value_labels = c("0" = "Non-SGM", "1" = "SGM"))
  track_outcomes(df, vars = "self_harm",
                 labels = list(self_harm = "Self-harm"))
  flow_diagram()
}
```

---

flow_diagram_export *Save the eligibility flowchart to a file*

Description

Renders flow_diagram() and writes it to disk. The output format is taken from the file extension, so the same call can produce a raster image, a vector image, or an editable office document:

Usage

```
flow_diagram_export(path, width = NULL, height = NULL, dpi = 300, ...)
```

Arguments

path	File path. The extension determines the format.
width, height	Plot size in inches. If NULL (default), sensible values are derived from the number of subgroups and outcomes.
dpi	Resolution for raster formats. Default: 300.
...	Further arguments passed to flow_diagram() (e.g. title, box_fill, text_size).

Details

- .png, .jpg/.jpeg, .tiff, .bmp - raster image.
- .pdf, .svg, .eps - vector image (editable in Inkscape / Illustrator; .svg needs the 'svglite' package).
- .emf - Windows vector metafile, editable in Word once inserted (needs the 'devEMF' package).
- .docx - a Word document with the flowchart embedded as a vector image (editable after ungrouping; needs the 'officer' package, plus 'devEMF' for the vector version, otherwise a raster image is used).

- `.pptx` - a PowerPoint slide where every box and label is a native, fully editable shape (needs the 'rvg' and 'officer' packages).

There is no default path: the destination must be supplied explicitly (e.g. a file under `tempdir()` or a directory chosen by the user).

Value

Invisible path, or invisible NULL if there is nothing to draw.

Examples

```
# Wrapped in \donttest because writing the image invokes a graphics
# device and may take more than 5 seconds; it writes only to tempdir().
if (requireNamespace("ggplot2", quietly = TRUE)) {
  cb_init(id_col = "id_indiv")
  df <- data.frame(
    id_indiv = sprintf("ID%03d", 1:100),
    g        = sample(c(0L, 1L), 100, replace = TRUE),
    y        = sample(c(0L, 1L), 100, replace = TRUE)
  )
  df <- track_split(df, by = "g", value_labels = c("0" = "A", "1" = "B"))
  track_outcomes(df, vars = "y", labels = list(y = "Outcome"))
  # Raster (no extra packages needed); written to tempdir() and cleaned up:
  out <- file.path(tempdir(), "flow.png")
  flow_diagram_export(out)
  unlink(out)
}
```

`flow_get` *Get the current flow tree (raw structure)*

Description

Returns the internal flow representation. Mostly for debugging / programmatic access. For a tidy table, use `flow_table()`.

Usage

```
flow_get()
```

Value

A list describing the flow tree.

flow_reset *Reset the flow tree*

Description

Clears the CONSORT flow tree. Called automatically by `cb_init()`.

Usage

```
flow_reset()
```

Value

Invisible NULL.

flow_table *Flow tree as a tidy table*

Description

Flattens the CONSORT flow tree into a publication-friendly data frame. One row per leaf x outcome. Split levels become named columns (using their labels), values are mapped through value_labels, and percentages are formatted as readable strings.

Usage

```
flow_table()
```

Value

A tibble.

generate_report *Generate a standardized report from the current session*

Description

Produces an HTML report combining the eligibility flowchart, the codebook, and a per-variable inspection panel. Supports two inspection modes:

Usage

```
generate_report(
  data,
  type = c("cross_sectional", "longitudinal"),
  id_var = NULL,
  time_var = NULL,
  variables = NULL,
  labels = NULL,
  treat_as_categorical = NULL,
  output_html,
  output_dir = NULL,
  export_codebook_editable = TRUE,
  cache_data = TRUE,
  title = NULL,
  n_bins = 30,
  top_n_cat = 20
)
```

Arguments

data	A Spark DataFrame (tbl_spark) or local data frame.
type	One of "cross_sectional" or "longitudinal".
id_var	Character. Name of the ID column. For longitudinal, mandatory. For cross_sectional, used to skip the ID column in inspection.
time_var	Character or NULL. Name of the time/wave column. Used in longitudinal to compute missingness-over-time. Default: NULL.
variables	Optional character vector. If provided, inspects only these variables. Default: NULL (all except id_var/time_var).
labels	Optional named list (variable -> label). If NULL, uses labels from the codebook when available.
treat_as_categorical	Character vector of variable names to treat as categorical even when their R class is numeric or integer. Useful for coded variables (e.g. cod_sexo stored as 1L/2L, cod_raca stored as integer). For these variables, the report uses bar charts and proportion-by-time stacked plots instead of histograms / median+IQR. Default: NULL.

output_html	File path for the HTML output. There is no default: the destination must be supplied explicitly (e.g. a file under tempdir() or a directory chosen by the user).
output_dir	Optional directory for ancillary files (codebook.xlsx, codebook.docx, etc.). If NULL, derived from output_html.
export_codebook_editable	Logical. Also export codebook as .docx and .xlsx in output_dir. Default: TRUE.
cache_data	Logical. If TRUE and data is a tbl_spark, persists the dataset once before the report aggregations, then releases it on exit. No-op for local data frames. Default: TRUE.
title	Optional title for the report.
n_bins	Number of bins for numeric histograms. Default: 30.
top_n_cat	Max categories shown in categorical plots. Default: 20.

Details

- cross_sectional: one plot per variable (histogram / bar / time).
- longitudinal: three plots per variable (global distribution, intra-ID variation, missingness by time) plus a meta plot of observations per ID.

All aggregations happen in Spark/dplyr; only small summaries are collected.

Value

Invisible list with paths to all generated files.

Examples

```
# Rendering the HTML report needs rmarkdown + pandoc and a few plotting
# packages (all in Suggests); it also takes more than 5 seconds, so the
# example is wrapped in \donttest and writes only to tempdir().
if (requireNamespace("rmarkdown", quietly = TRUE) &&
    requireNamespace("knitr", quietly = TRUE) &&
    requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("patchwork", quietly = TRUE) &&
    requireNamespace("scales", quietly = TRUE) &&
    rmarkdown::pandoc_available()) {

  cb_init(id_col = "id_indiv")
  df_baseline <- data.frame(
    id_indiv = sprintf("ID%03d", 1:50),
    cod_sexo = sample(c(1L, 2L), 50, replace = TRUE),
    idade    = sample(18:80, 50, replace = TRUE)
  )

  # Write to a dedicated subdir of tempdir() and clean everything up after:
  out_dir <- file.path(tempdir(), "autocodebook_report_demo")
  generate_report(df_baseline, type = "cross_sectional",
                  id_var = "id_indiv",
```

```
                  treat_as_categorical = "cod_sexo",
                  output_html = file.path(out_dir, "report_baseline.html"))
  unlink(out_dir, recursive = TRUE)
}
```

`track_export` *Export tracking table to file*

Description

Supports `.html`, `.csv`, `.docx`, and `.xlsx`.

Usage

```
track_export(path, show_elapsed = FALSE)
```

Arguments

`path`	File path. There is no default: the destination must be supplied explicitly (e.g. a file under `tempdir()` or a directory chosen by the user).
`show_elapsed`	Logical. Include elapsed_s column? Default: FALSE.

Value

Invisible path.

`track_get` *Get the current tracking log as a tibble*

Description

Get the current tracking log as a tibble

Usage

```
track_get()
```

Value

A tibble with all tracking steps.

track_outcomes *Attach outcome counts to the current leaves (CONSORT flowchart)*

Description

Adds one or more outcome variables, counted within each current leaf (combination of all splits so far). Outcomes are stacked, not branched. Each outcome is treated as binary: counts how many individuals have value == 1 (or TRUE), plus the percentage within the leaf.

Usage

```
track_outcomes(sdf, vars, labels = NULL)
```

Arguments

sdf	A Spark DataFrame or local data frame.
vars	Character vector of outcome column names (binary 0/1 or logical).
labels	Optional named list (var -> label).

Value

sdf unchanged (for piping).

track_render *Render the tracking log as a gt table*

Description

Render the tracking log as a gt table

Usage

```
track_render(show_elapsed = FALSE)
```

Arguments

show_elapsed	Logical. Show the elapsed_s column if present? Default: FALSE.

Value

A gt object.

track_reset *Reset the tracking log*

Description

Reset the tracking log

Usage

```
track_reset()
```

Value

Invisible NULL.

track_split *Split the cohort into branches by a column (CONSORT flowchart)*

Description

Adds one branching level to the flow tree. The cohort is divided by the distinct values of `by`. Chain multiple `track_split()` calls to create nested branches (e.g. exposure then mediator). Passes the data through unchanged, so it fits in a `%>%` pipeline.

Usage

```
track_split(sdf, by, label = NULL, value_labels = NULL, max_levels = 3L)
```

Arguments

`sdf`	A Spark DataFrame or local data frame.
`by`	Character. Column name to split by. Its distinct values become the branches. NA values are grouped as "(NA)".
`label`	Optional character. A human-readable name for this split level (e.g. "Exposure: drought"). Defaults to `by`.
`value_labels`	Optional named character vector mapping raw values to readable labels, e.g. `c("0" = "Sem seca", "1" = "Com seca")`. If not given, the function tries factor levels / labelled attributes on the column; failing that, uses the raw value.
`max_levels`	Integer. Safety cap on nesting depth. Default 3.

Value

`sdf` unchanged (for piping).

Examples

```
cb_init(id_col = "id_indiv")
df <- data.frame(
  id_indiv     = sprintf("ID%03d", 1:100),
  exposto_seca = sample(c(0L, 1L), 100, replace = TRUE),
  migrou       = sample(c(0L, 1L), 100, replace = TRUE),
  obito_dcv    = sample(c(0L, 1L), 100, replace = TRUE)
)
df <- track_split(df, by = "exposto_seca", label = "Exposure: drought",
                  value_labels = c("0" = "No drought", "1" = "Drought"))
df <- track_split(df, by = "migrou", label = "Mediator: migration",
                  value_labels = c("0" = "Did not migrate", "1" = "Migrated"))
track_outcomes(df, vars = "obito_dcv", labels = list(obito_dcv = "CVD death"))
flow_table()
```

---

track_step *Record a tracking step*

Description

Counts unique individuals in the current data and logs the step. Works with both tbl_spark and local data frames.

Usage

```
track_step(sdf, step_label, description = "", assume_unique = FALSE)
```

Arguments

sdf	A Spark DataFrame or local data frame.
step_label	Short label for the step.
description	Longer description.
assume_unique	Logical. If TRUE, skips the `distinct()` call when counting (use only when you are certain the ID column has no duplicates at this stage - e.g. after a deduplication step). Default: FALSE.

Value

Invisible integer: number of unique IDs.

Index

auto_filter, 2
auto_filter(), *10*
auto_mutate, 3
auto_summarise, 4

cb_checkpoint, 5
cb_export, 5
cb_get, 6
cb_init, 6
cb_register, 7
cb_render, 8
cb_reset, 8
cb_set_default_cache, 9
cb_set_verbose, 9

dplyr::filter(), *2*
dplyr::mutate(), *3*
dplyr::summarise(), *4*

flow_diagram, 10
flow_diagram(), *11*
flow_diagram_export, 11
flow_get, 12
flow_reset, 13
flow_table, 13

generate_report, 14

tempdir(), *6*, *12*, *15, 16*
track_export, 16
track_get, 16
track_outcomes, 17
track_outcomes(), *10*
track_render, 17
track_reset, 18
track_split, 18
track_split(), *10*
track_step, 19
track_step(), *10*