Package: cellsynth (via r-universe)

October 30, 2026

Title Cell-Wise Synthetic Copies of De-Identified Tables

Version 0.1.0

Description Turns de-identified tables ('Excel' workbooks or delimited text files) into synthetic counterparts with the same layout, the same identifiers and the same data-entry quirks. Every non-identifier cell, missingness included, is censored once per sweep and redrawn by copying the raw value of a donor drawn from a classification and regression tree leaf fitted on the rest of the table, in the spirit of the sequential tree-based synthesis of Reiter (2005) ``Using CART to generate partially synthetic public use microdata'', Journal of Official Statistics 21(3):441-462, and Nowok, Raab and Dibben (2016) <doi:10.18637/jss.v074.i11>. Identifiers are kept so that tables stay joinable, and columns of other tables can condition the synthesis so cross-table associations survive a later join. Synthetic copies are intended for developing and testing analysis code; they are not an anonymisation method.

License MIT + file LICENSE

URL https://github.com/MartynK/cellsynth

BugReports https://github.com/MartynK/cellsynth/issues

Encoding UTF-8

Config/roxygen2/version 8.1.1

Imports openxlsx, readr, readxl, rpart, stats, tools, utils, withr

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Marton Kiss [aut, cre]

Maintainer Marton Kiss <mrkmarton@gmail.com>

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

Date/Publication 2026-10-30 19:33:14 UTC

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

RemoteRef HEAD

RemoteSha bfd35f6ad3e33b38b40a974bee526d62a2bcb776

Contents

apply_role_overrides . . . 2
compare_correlations . . . 3
compare_tables . . . 4
detect_nonsubject_rows . . . 5
infer_spec . . . 6
make_aux . . . 7
parse_date_chr . . . 8
parse_number_chr . . . 9
passthrough_result . . . 9
print.synth_result . . . 10
print.synth_source . . . 10
read_source . . . 11
read_spec . . . 12
row_disclosure . . . 13
source_sheets . . . 13
synthesize_file . . . 14
synthesize_table . . . 16
write_spec . . . 17
write_synthetic . . . 18

Index **20**

`apply_role_overrides` *Force roles onto a spec by column name or name pattern*

Description

Used by `infer_spec()` and by `synthesize_file()` (where it also applies to a reviewed spec read from disk, so identifiers named in the call always win). Names match the internal or the on-disk column name, ignoring case; patterns are regular expressions (perl, case-insensitive) matched against the on-disk name. The options `cellsynth.id`, `cellsynth.id_pattern`, `cellsynth.text` and `cellsynth.text_pattern` add session-wide defaults, e.g. `options(cellsynth.id_pattern = "^(pat|subj)")` once per session.

Usage

```
apply_role_overrides(
  spec,
  id = NULL,
  text = NULL,
```

```
  roles = NULL,
  id_pattern = NULL,
  text_pattern = NULL,
  warn_unknown = TRUE
)
```

Arguments

spec	A spec data.frame.
id, text	Column names to force to id / text.
roles	Named character vector, c(column = "role"), for any role.
id_pattern, text_pattern	Regular expressions on column names.
warn_unknown	Warn about names in id/text/roles that match no column.

Value

The spec with roles replaced and reason set to "user: ...".

Examples

```
df <- data.frame(PatKey = sprintf("%04d", 1:60), Visit_Note = "seen",
                 hb = round(seq(110, 160, length.out = 60)))
path <- file.path(tempdir(), "overrides.csv")
utils::write.csv(df, path, row.names = FALSE)
spec <- infer_spec(read_source(path))
spec <- apply_role_overrides(spec, id = "patkey", text_pattern = "note$")
spec[, c("orig_name", "role", "reason")]
```

---

compare_correlations *Mean absolute difference between correlation matrices*

Description

Over the numeric, date and binary categorical columns, Spearman correlations on pairwise complete rows.

Usage

```
compare_correlations(orig, syn, spec)
```

Arguments

orig, syn	Raw character data.frames in the same layout.
spec	The shared spec.

Value

list(mad = mean absolute difference, orig, syn matrices).

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20))
path <- file.path(tempdir(), "compare_correlations.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src)
syn <- src$raw
syn$value <- rev(syn$value)
compare_correlations(src$raw, syn, spec)
```

---

compare_tables *Compare original and synthetic tables column by column*

Description

For every modelled column: missing fraction in both; for numeric and date columns the mean, SD and median and the Kolmogorov-Smirnov distance; for categorical columns the total variation distance between label distributions. `self_match` is the share of subject cells equal to the same row's original cell; for categorical columns `chance_match` gives what a random redraw from the marginal would produce, so a large excess points at a column that copies its own row.

Usage

```
compare_tables(orig, syn, spec, rows = seq_len(nrow(orig)))
```

Arguments

`orig`, `syn`	Raw character data.frames in the same layout.
`spec`	The shared spec.
`rows`	Rows to compare (default: all).

Value

A data.frame, one row per modelled column.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20), group = rep(c("A", "B"), 30))
path <- file.path(tempdir(), "compare_tables.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src)
```

```
syn <- src$raw
syn$value <- rev(syn$value)
compare_tables(src$raw, syn, spec)
```

`detect_nonsubject_rows` *Detect rows that are not subjects (totals, site headers)*

Description

A row is non-subject when any identifier cell reads like a summary label (`TOTAL`, `Sum`, `Osszesen` with or without accent), or when the main identifier carries no digit while at least 90% of the others do (site-name header rows).

Usage

```
detect_nonsubject_rows(raw, spec)
```

Arguments

`raw`	Raw character data.frame.
`spec`	Its spec.

Value

Logical vector, `TRUE` for non-subject rows.

Examples

```
df <- data.frame(id = c(sprintf("AB%03d", 1:59), "TOTAL"), value = 1:60)
path <- file.path(tempdir(), "nonsubject_rows.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src, id = "id")
flagged <- detect_nonsubject_rows(src$raw, spec)
src$raw[flagged, , drop = FALSE]
```

infer_spec *Infer a column specification for a source table*

Description

Heuristics, in order: all-missing columns are `empty`; trial and EDC key names (`USUBJID`, `SUBJID`, `SITEID`, `STUDYID`, `RANDNO`, ...) are `id`; identifier-like names with near-unique values (or code-like values such as `AB123`) are `id`; a single value with no missing cells is `constant`; mostly-date columns are `date`; mostly-numeric columns are `numeric`, or `categorical` if they take few distinct values (coded 0/1/2 fields); remaining character columns are `categorical` when the label set is small and `text` otherwise.

Usage

```
infer_spec(
  src,
  id = NULL,
  text = NULL,
  roles = NULL,
  id_pattern = NULL,
  text_pattern = NULL,
  max_levels = 10,
  max_cat_levels = 60,
  warn_unknown = TRUE
)
```

Arguments

`src`	A `synth_source` from `read_source()`.
`id`, `text`	Column names (internal or on-disk, case-insensitive) to force to `id` or `text`, whatever the heuristics say.
`roles`	Named character vector of further role overrides.
`id_pattern`, `text_pattern`	Regular expressions on column names, e.g. `id_pattern = "^(pat|subj).*(id|no)$"`. See `apply_role_overrides()`, also for session-wide defaults via `options()`.
`max_levels`	Numeric columns with at most this many distinct values are treated as categorical.
`max_cat_levels`	Character columns with at most this many labels (and at most 10% of the observed rows) are categorical, otherwise text.
`warn_unknown`	Warn about override names that match no column.

Value

A data.frame, one row per column: `column`, `orig_name`, `role`, `n_obs`, `n_unique`, `miss_frac`, `num_frac`, `date_frac`, `reason`.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20), group = rep(c("A", "B"), 30))
path <- file.path(tempdir(), "infer_spec.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src, id = "id", roles = c(code = "categorical"))
spec[, c("column", "role", "reason")]
```

---

make_aux *Align columns of another table as auxiliary predictors*

Description

Auxiliary columns condition the synthesis (so cross-table associations survive a later join) but are never censored or written out.

Usage

```
make_aux(
  src,
  by,
  aux_src,
  aux_spec,
  aux_by,
  cols = NULL,
  key_fun = function(k) toupper(gsub("[^A-Za-z0-9]", "", k))
)
```

Arguments

src	The table being synthesised (synth_source).
by	Key column of src (internal or on-disk name).
aux_src	The other table (synth_source).
aux_spec	Spec of aux_src.
aux_by	Key column of aux_src.
cols	Columns of aux_src to use (default: all modelled ones).
key_fun	Function normalising keys before matching (e.g. dropping a hyphen so AB-123 matches AB123).

Value

list(raw, roles), rows aligned to src$raw (first match per key).

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60)
other <- data.frame(id = rev(df$id), code = rep(c(0, 1, NA), 20))
path <- file.path(tempdir(), "make_aux_main.csv")
aux_path <- file.path(tempdir(), "make_aux_other.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
utils::write.csv(other, aux_path, row.names = FALSE, na = "")
src <- read_source(path)
aux_src <- read_source(aux_path)
aux_spec <- infer_spec(aux_src)
aux <- make_aux(src, by = "id", aux_src = aux_src,
                aux_spec = aux_spec, aux_by = "id", cols = "code")
head(aux$raw)
```

---

parse_date_chr *Parse raw strings as dates (days since 1970-01-01)*

Description

Recognises ISO `yyyy-mm-dd` (with optional time), `yyyy.mm.dd`, `yyyy/mm/dd`, `dd.mm.yyyy` and, when `allow_serial = TRUE`, Excel date serials (numbers between 3000 and 73000, i.e. 1908 to 2099).

Usage

```
parse_date_chr(x, allow_serial = FALSE)
```

Arguments

`x`	Character vector.
`allow_serial`	Logical; treat plausible numbers as Excel serials.

Value

Numeric vector of days since 1970-01-01, `NA` if not a date.

Examples

```
x <- c("2020-01-15", "16.01.2020", "2020/01/17", "43848", NA)
days <- parse_date_chr(x)
as.Date(days, origin = "1970-01-01")
days <- parse_date_chr(x, allow_serial = TRUE)
as.Date(days, origin = "1970-01-01")
```

parse_number_chr *Parse raw strings as numbers, tolerating decimal commas*

Description

Parse raw strings as numbers, tolerating decimal commas

Usage

```
parse_number_chr(x)
```

Arguments

x	Character vector.

Value

Numeric vector, NA where a cell is not a number.

Examples

```
x <- c("1.5", "2,75", " 3 ", "1e2", "unknown", NA)
parsed <- parse_number_chr(x)
parsed
is.na(parsed)
data.frame(raw = x, numeric = parsed)
```

passthrough_result *A result that is not synthesised: copied, blanked or dropped*

Description

For sheets that are empty, have no modelled columns, or are too small to synthesise (fewer than 4 * minbucket subject rows): legends, codebooks, lookup tables, tiny pilot tables.

Usage

```
passthrough_result(src, spec, how = c("blank", "copy", "drop"), reason = "")
```

Arguments

src	A synth_source.
spec	Its spec.
how	"copy" (verbatim), "blank" (identifiers, constant and empty columns kept, every modelled cell emptied) or "drop" (left out of the output workbook).
reason	Recorded in settings$passthrough.

Value

A synth_result with settings$passthrough set.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:6), value = 1:6)
path <- file.path(tempdir(), "passthrough_result.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src)
res <- passthrough_result(src, spec, how = "blank", reason = "Tiny table")
res$synthetic
```

---

print.synth_result *Print a synth_result*

Description

Print a synth_result

Usage

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

Arguments

x	A synth_result.
...	Ignored.

Value

x, invisibly.

---

print.synth_source *Print a synth_source*

Description

Print a synth_source

Usage

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

Arguments

x	A synth_source.
...	Ignored.

Value

x, invisibly.

---

read_source *Read a source table (xlsx or csv) as raw character data*

Description

The first row is always taken as the header. Excel formulas are read as their cached values; cell formatting (colours, merged cells, number formats) is not read. CSV files are read with an auto-detected delimiter (comma, semicolon, tab or pipe) and encoding (Hungarian Excel exports are often semicolon-separated windows-1250).

Usage

```
read_source(
  path,
  sheet = NULL,
  guess_max = 1e+05,
  delim = NULL,
  encoding = NULL
)
```

Arguments

path	Path to an .xlsx/.xlsm/.xls or .csv/.txt file.
sheet	Sheet name or index for Excel files (default: first sheet).
guess_max	Rows used by readxl to guess the original column types, which are only used to restore cell types on writing.
delim, encoding	CSV only; NULL detects them.

Value

A synth_source list: raw (data.frame of character, internal unique names), orig_names (header as on disk, duplicates allowed), orig_class (readxl/readr guessed class per column), path, sheet, format, and for CSV delim and encoding.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20))
path <- file.path(tempdir(), "read_source.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
head(src$raw)
```

`read_spec` *Read a spec CSV written by* `write_spec()`

Description

Read a spec CSV written by `write_spec()`

Usage

```
read_spec(path)
```

Arguments

`path`	Path to the spec `.csv`.

Value

A spec data.frame.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60)
path <- file.path(tempdir(), "read_spec_source.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
spec <- infer_spec(read_source(path))
spec_path <- file.path(tempdir(), "read_spec.csv")
write_spec(spec, spec_path)
reviewed <- read_spec(spec_path)
reviewed[, c("column", "role")]
```

row_disclosure *Share of synthetic rows identical to their own original row*

Description

Compares the modelled columns only (identifiers are equal by design).

Usage

```
row_disclosure(orig, syn, spec, rows = seq_len(nrow(orig)), min_share = 0.9)
```

Arguments

orig, syn	Raw character data.frames in the same layout.
spec	The shared spec.
rows	Rows to compare (default: all).
min_share	Report also rows matching at least this share of cells.

Value

list(identical = share of rows fully identical, identical_unique = share of rows identical AND unique in the original (the disclosive kind), near = share of rows matching at least min_share of their cells, per_row = vector).

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20))
path <- file.path(tempdir(), "row_disclosure.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
spec <- infer_spec(src)
syn <- src$raw
syn$value[1:30] <- rev(syn$value[1:30])
row_disclosure(src$raw, syn, spec, min_share = 0.5)
```

source_sheets *Sheet names of an Excel workbook*

Description

Sheet names of an Excel workbook

Usage

```
source_sheets(path)
```

Arguments

path	Path to an Excel file.

Value

Character vector, in workbook order.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:6), value = 1:6)
path <- file.path(tempdir(), "source_sheets.xlsx")
sheets <- list(Subjects = df, Codes = data.frame(code = c(0, 1)))
openxlsx::write.xlsx(sheets, path, overwrite = TRUE)
source_sheets(path)
```

---

synthesize_file *Synthesise a file end to end*

Description

Reads the source (every sheet of a workbook unless `sheet` says otherwise), infers a spec per sheet (or reads the reviewed one), writes inferred specs for review, synthesises `m` independent copies and writes each as `<stem>_synth<i>.<ext>` (all sheets in one workbook, in the original order) with a run log.

Usage

```
synthesize_file(
  path,
  out_dir,
  sheet = NULL,
  spec_path = NULL,
  spec_dir = out_dir,
  id = NULL,
  text = NULL,
  id_pattern = NULL,
  text_pattern = NULL,
  m = 1,
  seed = 1,
  small_sheets = c("blank", "copy", "drop"),
  minbucket = 5,
  delim = NULL,
  encoding = NULL,
  ...
)
```

Arguments

path	Source .xlsx/.xlsm/.xls or .csv/.txt.
out_dir	Output folder.
sheet	Sheet names or indices to synthesise (default: all sheets). Sheets not listed are left out of the output.
spec_path	Spec CSV for a single-sheet source: read when it exists, written when it does not.
spec_dir	Folder for per-sheet specs (<stem>__<position>_<sheet>_spec.csv); default out_dir.
id, text	Column names forced to id / text (case-insensitive). They apply to inferred specs and also on top of reviewed specs read from disk, so identifiers named here always stay untouched.
id_pattern, text_pattern	Regular expressions on column names, same effect (see apply_role_overrides()).
m	Number of synthetic copies.
seed	Base seed; copy i, sheet k of K uses seed + K * (i - 1) + k - 1.
small_sheets	Policy for sheets too small to synthesise.
minbucket	Passed to synthesize_table() and used for the size check.
delim, encoding	CSV only, passed to read_source(); NULL detects.
...	Passed to synthesize_table().

Details

Review the spec CSVs after the first run: set role to id for every identifier the heuristics missed and to text for free text, then run again with the same spec_dir (or spec_path).

Sheets that cannot be synthesised are handled by small_sheets: empty sheets and sheets without modelled columns are copied as they are (they hold no modelled values); sheets with fewer than 4 * minbucket subject rows (legends, codebooks, lookup tables, tiny pilots) get small_sheets: "blank" (default; keeps header and identifiers, empties the rest), "copy" (verbatim, for codebooks you know hold no subject data) or "drop".

Value

list(results = list over copies of lists over sheets of synth_result, files, spec_paths).

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20), group = rep(c("A", "B"), 30))
path <- file.path(tempdir(), "synthesize_file.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
out <- synthesize_file(path, out_dir = file.path(tempdir(), "synth_output"),
                       m = 1, seed = 1, n_folds = 2, verbose = FALSE)
out$files
out$spec_paths
head(out$results[[1]][[1]]$synthetic)
```

`synthesize_table` *Synthesise one table*

Description

Synthesise one table

Usage

```
synthesize_table(
  src,
  spec = NULL,
  n_folds = 10,
  n_cycles = 1,
  censor_mode = c("cell", "row"),
  lock_threshold = 0.95,
  max_predictors = 25,
  minbucket = 5,
  cp = 1e-04,
  aux = NULL,
  aux_always = TRUE,
  nonsubject = c("blank", "keep"),
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`src`	A `synth_source` from `read_source()`.
`spec`	Column spec; inferred with `infer_spec()` when `NULL`.
`n_folds`	Number of folds; each round censors 1/`n_folds` of every column (default 10, i.e. 10%).
`n_cycles`	Full sweeps over all folds.
`censor_mode`	The privacy-utility dial. • `"cell"` (default): folds are drawn per column; each censored cell is redrawn given the row's other current values. Within-row and cross-table associations survive well, but a pair of columns that determine each other (death flag and death date) cannot change, so such attributes stay real per identifier. • `"row"`: whole rows are censored together and rebuilt column by column from already redrawn columns and `aux` only (sequential synthesis as in synthpop). Rows become fully synthetic; their link to the identifier survives only through `aux`.

lock_threshold	Cell mode only: columns whose screening association (largest absolute correlation between value, missingness and level signals) reaches this value form a lock group, share a fold and are redrawn in sequence without seeing each other's censored cells. This re-rolls coupled pairs (flag + date) that would otherwise stay real. NULL switches it off.
max_predictors	Predictors per target after screening.
minbucket, cp	rpart controls (minimum donor pool per leaf, complexity).
aux	Optional auxiliary predictors from make_aux().
aux_always	If TRUE (default) every aux column is offered to every target's tree in addition to the max_predictors screened ones; if FALSE aux columns compete in the screening like any other column.
nonsubject	What to do with non-subject rows (totals, site headers): "blank" keeps their identifiers and empties the rest, "keep" leaves them verbatim. They are never donors.
seed	Random seed.
verbose	Print progress.

Value

A synth_result: synthetic (raw character data.frame in the source layout), spec, log, nonsubject rows, settings, and the source metadata needed to write it out.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20), group = rep(c("A", "B"), 30))
path <- file.path(tempdir(), "synthesize_table.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
res <- synthesize_table(src, n_folds = 2, seed = 1, verbose = FALSE)
head(res$synthetic)
```

---

write_spec *Write a spec to CSV for review and editing*

Description

Only the role column is meant to be edited by hand; the rest documents why the heuristics chose it. The file holds column names and counts only, no cell values.

Usage

```
write_spec(spec, path)
```

Arguments

spec	A spec data.frame.
path	Output .csv path.

Value

path, invisibly.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60)
path <- file.path(tempdir(), "write_spec_source.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
spec <- infer_spec(read_source(path))
spec_path <- file.path(tempdir(), "write_spec.csv")
write_spec(spec, spec_path)
file.exists(spec_path)
```

---

write_synthetic *Write a synthetic table (or workbook) in the format of its source*

Description

Excel output keeps the original header (duplicate names included), the sheet names and order and, column by column, the cell type readxl saw in the source (numbers as numbers, dates as dates, everything else as text). Formulas, formatting and merged cells are not reproduced (values only). .xls sources are written as .xlsx. CSV output writes the raw strings with the original header and delimiter, in UTF-8.

Usage

```
write_synthetic(res, path)
```

Arguments

res	A synth_result, or a list of them (one per sheet, in order) for a workbook. Results passed through with "drop" are left out.
path	Output path.

Value

path, invisibly.

Examples

```
df <- data.frame(id = sprintf("AB%03d", 1:60), value = 1:60,
                 code = rep(c(0, 1, NA), 20))
path <- file.path(tempdir(), "write_synthetic_source.csv")
utils::write.csv(df, path, row.names = FALSE, na = "")
src <- read_source(path)
res <- synthesize_table(src, n_folds = 2, seed = 1, verbose = FALSE)
out_path <- file.path(tempdir(), "write_synthetic_output.csv")
write_synthetic(res, out_path)
file.exists(out_path)
```

Index

apply_role_overrides, 2
apply_role_overrides(), *6*, *15*

compare_correlations, 3
compare_tables, 4

detect_nonsubject_rows, 5

infer_spec, 6
infer_spec(), *2*, *16*

make_aux, 7
make_aux(), *17*

parse_date_chr, 8
parse_number_chr, 9
passthrough_result, 9
print.synth_result, 10
print.synth_source, 10

read_source, 11
read_source(), *6*, *15*, *16*
read_spec, 12
row_disclosure, 13

source_sheets, 13
synthesize_file, 14
synthesize_file(), *2*
synthesize_table, 16
synthesize_table(), *15*

write_spec, 17
write_spec(), *12*
write_synthetic, 18