Package 'fabricatr'

October 21, 2026

Title Tidyverse-First Data Fabrication

Version 2.0.0

Description Helps you imagine your data before you collect it. Builds a data frame one column at a time, with hierarchical levels, cross-classified levels, potential outcomes, and draws with a target intra-cluster correlation, so that a research design can be simulated and diagnosed before any data exist. Version 2.0 is a rewrite of the same public API on 'dplyr', 'tibble', 'purrr', and 'rlang'.

URL https://declaredesign.org/r/fabricatr/, https://github.com/DeclareDesign/fabricatr

BugReports https://github.com/DeclareDesign/fabricatr/issues

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports dplyr (>= 1.1.0), tibble, purrr, rlang, stats, utils

Suggests knitr, MASS, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Alexander Coppock [aut, cre] (ORCID: <https://orcid.org/0000-0002-5733-2386>), Graeme Blair [aut] (ORCID: <https://orcid.org/0000-0001-9164-2102>), Jasper Cooper [aut] (ORCID: <https://orcid.org/0000-0002-8639-3188>), Macartan Humphreys [aut] (ORCID: <https://orcid.org/0000-0001-7029-2326>), Aaron Rudkin [aut], Neal Fultz [aut], David C. Hall [ctb] (ORCID: <https://orcid.org/0000-0002-2193-0480>)

Maintainer Alexander Coppock <acoppock@gmail.com>

Repository CRAN

Date/Publication 2026-10-21 16:30:07 UTC

Contents

add_level . . . 2
ALL . . . 3
correlate . . . 4
cross_levels . . . 5
declare_level . . . 5
draw_binary . . . 6
draw_binary_icc . . . 7
draw_binomial . . . 8
draw_categorical . . . 9
draw_count . . . 10
draw_likert . . . 11
draw_multivariate . . . 12
draw_normal_ar . . . 13
draw_normal_icc . . . 14
draw_ordered . . . 15
draw_quantile . . . 16
fabricate . . . 17
fabricate_with_dots . . . 18
import_level . . . 19
join_using . . . 21
link_levels . . . 21
modify_level . . . 22
nest_level . . . 24
potential_outcomes . . . 24
recycle . . . 26
resample_data . . . 26
reveal_outcomes . . . 27
split_quantile . . . 28

Index **30**

`add_level` *Add a level, nested in the one before it if there is one*

Description

The first `add_level()` in a `fabricate()` call creates `N` rows, since there is nothing above it to nest in. A later one nests: it creates `N` rows inside each row of the level before it, exactly as `nest_level` does, and the two can be used interchangeably there. For independent (non-nested) levels used in `cross_levels` or `link_levels`, use `declare_level`.

Usage

```
add_level(N, ...)
```

Arguments

N	Number of rows to create.
...	Column expressions evaluated sequentially. N is available as a scalar integer.

Value

A fabricatr_level object, meaningful only as a named argument to fabricate. It contributes N rows, an ID column named after the argument it is assigned to, and one column per expression in .... Any level nested below it repeats these rows.

Examples

```
fabricate(
  villages = add_level(N = 20, income = rnorm(N)),
  citizens = nest_level(N = 5, Y = income + rnorm(N))
)
```

ALL *Keep every unit at a resampling level*

Description

Use ALL in the N argument of resample_data to keep every unit at a given hierarchical level while still resampling inner levels. Leaving a level out of N does the same thing: N = c(clusters = 2) and N = c(clusters = 2, units = ALL) return the same frame.

Usage

```
ALL
```

Format

A single integer used as a sentinel. Its value carries no meaning and should not be relied on or arithmetic done with it; write ALL itself wherever a level is to pass through whole.

Examples

```
clustered <- fabricate(
  clusters = add_level(N = 4, gdp = rnorm(N)),
  units    = nest_level(N = 2, Y = gdp + rnorm(N))
)
# Resample 2 clusters, keeping every unit inside the ones drawn
resample_data(clustered, N = c(clusters = 2, units = ALL))
```

`correlate` *Generate a variable correlated with an existing variable*

Description

Uses a Gaussian copula to produce a draw from `draw_handler` whose rank correlation with `given` is approximately `rho`. Works with any `draw_*` function that accepts a `quantile_y` argument, and with base R random-number generators (e.g. `rnorm`, `rpois`).

Usage

```
correlate(draw_handler, ..., given, rho)
```

Arguments

`draw_handler`	Unquoted function name. Either one of the three `draw_*` functions that take a `quantile_y` argument (`draw_binary`, `draw_binomial`, `draw_count`) or a base R `r*` generator with a `q*` counterpart (`rnorm`, `rpois`, `rbinom`, and the rest of the **stats** family). `draw_ordered`, `draw_likert`, `draw_categorical`, `draw_quantile`, and the two ICC draws take no quantile argument and are refused with a message saying so.
`...`	Arguments forwarded to `draw_handler` (e.g. `prob`, `mean`).
`given`	Reference vector; the new variable will be rank-correlated with this.
`rho`	Correlation of the Gaussian copula, in $[-1, 1]$. It is not the realized Spearman correlation, which is smaller by the copula's own relation, $(6/\pi)\arcsin(\rho/2)$: `rho = 0.7` lands at 0.68 and `rho = 0.3` at 0.288. The realized Pearson correlation is `rho`. fabricatr 1.x behaves identically.

Value

A double vector of length `length(given)`. Note that the type is double whatever `draw_handler` is: `draw_binary`, `draw_binomial`, and `draw_count` each return an integer vector when called directly and a double one when routed through here, because the copula reaches them through a quantile function. The values are unaffected. An `NA` in `given` yields an `NA` in the result at that position.

Examples

```
score  <- rnorm(100, mean = 75, sd = 10)
offers <- correlate(draw_count, mean = 3, given = score, rho = 0.6)
cor(score, offers, method = "spearman")

# Works with base R generators too
y2 <- correlate(rnorm, mean = 0, sd = 1, given = score, rho = -0.5)
```

cross_levels *Create a full Cartesian product of declared levels*

Description

Produces all combinations of the specified levels (equivalent to SQL CROSS JOIN). Use `link_levels` to sample N rows from the product with an optional correlation structure.

Usage

```
cross_levels(.by, ...)
```

Arguments

`.by`	Character vector of level names to cross (must have been created by `add_level` or `declare_level` in the same `fabricate` call).
`...`	Additional column expressions evaluated after crossing.

Value

A `fabricatr_level` object, meaningful only as a named argument to `fabricate`. It contributes one row per combination of the levels named in `.by`, so the frame has as many rows as their sizes multiplied together, carrying every column of those levels plus an ID column of its own. The first level in `.by` varies fastest.

Examples

```
fabricate(
  countries = declare_level(N = 10, gdp = runif(N, 1, 10)),
  years     = declare_level(N = 5, shock = runif(N, 0, 1)),
  obs       = cross_levels(.by = c("countries", "years"), Y = gdp + shock)
)
```

declare_level *Declare an independent level for cross-classification*

Description

Creates N rows as a standalone data frame registered for use by `cross_levels` or `link_levels`. Unlike `add_level`, a `declare_level` call does not nest into any existing hierarchy: it starts fresh. Use this when you need two independent populations to cross (e.g., countries and years for panel data).

Usage

```
declare_level(N, ...)
```

Arguments

N	Number of rows to create.
...	Column expressions evaluated sequentially. N is available as a scalar integer.

Value

A fabricatr_level object, meaningful only as a named argument to fabricate. It builds N rows and an ID column named after the argument it is assigned to, and holds them aside: the rows reach the fabricated frame only when a later cross_levels or link_levels names this level in .by. Declaring a level that nothing crosses or links contributes nothing to the result.

Examples

```
fabricate(
  countries = declare_level(N = 20, gdp = runif(N, 1, 10)),
  years     = declare_level(N = 10, shock = runif(N, 1, 5)),
  obs       = cross_levels(
    .by = c("countries", "years"),
    GDP_it = gdp + shock
  )
)
```

---

draw_binary *Draw binary (0/1) outcomes*

Description

Draw binary (0/1) outcomes

Usage

```
draw_binary(
  prob = apply_link(latent, link),
  N = length(prob),
  link = "identity",
  latent = NULL,
  quantile_y = NULL
)
```

Arguments

prob	Probability of success. Scalar or per-unit vector. Already on the probability scale, so a non-identity link has nothing to do to it: supplying both is an error naming latent.
N	Number of draws. Defaults to length(prob).

link	One of "identity" (default), "logit", "logistic", "probit", or a function, which is applied to latent as it stands. Any other name is an error, as it is in fabricatr 1.x; "logistic" is a 2.0 synonym for "logit" that 1.x does not accept. The link acts on latent only.
latent	Latent variable on an unbounded scale, mapped to the probability scale by link. Supply this rather than prob whenever link is not the identity.
quantile_y	Optional quantile vector for deterministic draws (used by correlate).

Value

An integer vector of 0s and 1s, of length N, which defaults to length(prob).

Examples

```
fabricate(N = 6, p = c(0, 0.2, 0.4, 0.6, 0.8, 1),
          Y = draw_binary(prob = p))

# Logit link: supply a latent continuous variable
fabricate(N = 100, x = rnorm(N), Y = draw_binary(latent = x, link = "logit"))
```

---

draw_binary_icc *Draw binary data with a target intra-cluster correlation*

Description

Uses the mixture-of-Bernoullis model: with probability $\sqrt{ICC}$ each unit takes the cluster-level draw; otherwise it takes an independent draw. This produces an expected ICC of the specified value.

Usage

```
draw_binary_icc(clusters, prob = 0.5, ICC = 0, N = NULL)
```

Arguments

clusters	Vector of cluster IDs.
prob	Cluster success probability. Scalar or per-cluster vector. Default 0.5.
ICC	Target ICC. Default 0 (independent draws).
N	Optional; must equal length(clusters).

Value

An integer vector of 0s and 1s of the same length as clusters, in the order the clusters were given.

Examples

```
clusters <- rep(1:10, each = 20)
y <- draw_binary_icc(clusters = clusters, prob = 0.5, ICC = 0.3)
summary(lm(y ~ factor(clusters)))$r.squared  # approx 0.3
```

`draw_binomial` *Draw binomial counts*

Description

Draw binomial counts

Usage

```
draw_binomial(
  prob = apply_link(latent, link),
  trials = 1L,
  N = length(prob),
  link = "identity",
  latent = NULL,
  quantile_y = NULL
)
```

Arguments

`prob`	Probability of success per trial, already on the probability scale. Supplying it with a non-identity `link` is an error naming `latent`.
`trials`	Number of trials per observation (scalar or vector).
`N`	Number of observations. Defaults to `length(prob)`.
`link`	One of `"identity"` (default), `"logit"`, `"logistic"`, `"probit"`, or a function applied to `latent`. Any other name is an error.
`latent`	Latent variable on an unbounded scale, mapped to the probability scale by `link`.
`quantile_y`	Optional quantile vector for `correlate`.

Value

An integer vector of successes out of `trials`, of length `N`.

Examples

```
fabricate(N = 4, p = c(0.1, 0.3, 0.7, 0.9),
          Y = draw_binomial(prob = p, trials = 10))
```

`draw_categorical` *Draw categorical outcomes*

Description

Draws from a multinomial distribution. Supply a probability matrix (one row per observation, one column per category) or a single probability vector shared across all observations.

Usage

```
draw_categorical(prob, N = NULL, labels = NULL, category_labels = NULL)
```

Arguments

`prob`	Probability specification. Either: • A numeric vector of category probabilities (same for all observations; `N` must be supplied), or • A matrix with `N` rows and one column per category. Rows need not sum to 1; they are normalised internally.
`N`	Number of observations. Required when `prob` is a vector.
`labels`	Optional character vector of category labels (length must equal the number of categories). When supplied, returns a factor. The categories are nominal, so the factor is unordered, as it is in fabricatr 1.x.
`category_labels`	fabricatr 1.x's name for `labels`. Accepted with a warning.

Value

An integer vector of category indices `1`, `2`, and so on, one per row of `prob`, or an unordered factor with those `labels` when they are supplied.

Examples

```
# Shared probabilities across all units
fabricate(N = 100, cat = draw_categorical(prob = c(0.2, 0.5, 0.3), N = N))

# Per-unit probability matrix
fabricate(N = 4, p1 = runif(N), p2 = runif(N), p3 = runif(N),
          cat = draw_categorical(prob = cbind(p1, p2, p3)))

# With labels -> factor
fabricate(N = 100,
          edu = draw_categorical(prob = c(0.3, 0.5, 0.2), N = N,
                                 labels = c("low", "medium", "high")))
```

draw_count *Draw Poisson count data*

Description

Draw Poisson count data

Usage

```
draw_count(
  mean = apply_link(latent, link),
  N = length(mean),
  link = "identity",
  latent = NULL,
  quantile_y = NULL,
  dispersion = 0
)
```

Arguments

mean	Mean count (lambda). Scalar or per-unit vector, already the rate the draw uses. Supplying it with a non-identity `link` is an error naming `latent`, because `mean = log(3)` with `link = "log"` reads as a request to exponentiate and no range check can tell it apart from a rate of 1.099.
N	Number of draws. Defaults to `length(mean)`.
link	`"identity"` (default) or `"log"`, or a function applied to `latent`. `"log"` exponentiates `latent` to get the rate. Any other name is an error. fabricatr 1.x accepts no link here at all, answering "Count data does not accept link functions".
latent	Latent variable on the log scale, exponentiated by `link = "log"` to give the rate.
quantile_y	Optional quantile vector for `correlate`.
dispersion	Overdispersion, a single non-negative number. The default 0 is the Poisson, whose variance equals its mean. A positive value draws from the negative binomial with the same mean and variance `mean + dispersion * mean^2`. It is the reciprocal of the `theta` that `MASS::glm.nb()` reports, and the `alpha` of Stata's `nbreg`.

Value

An integer vector of non-negative counts, of length `N`.

Examples

```
fabricate(N = 5, rate = c(0, 1, 5, 10, 50),
          Y = draw_count(mean = rate))

# same mean, variance 3 + 0.5 * 3^2 = 7.5 rather than 3
```

```
y <- draw_count(mean = 3, N = 10000, dispersion = 0.5)
c(mean(y), var(y))
```

`draw_likert` *Recode a latent variable into a Likert response*

Description

A convenience wrapper around `draw_ordered` that constructs equally spaced breaks between `min` and `max`.

Usage

```
draw_likert(
  x,
  min = NULL,
  max = NULL,
  bins = NULL,
  breaks = NULL,
  labels = NULL
)
```

Arguments

`x`	Latent numeric variable.
`min`, `max`	Range the latent variable is cut on, rather than a filter applied to it: a value outside `[min, max]` joins the outermost bin. For 1.0.2's behaviour, where such a value became `NA`, call `draw_ordered(x, breaks = seq(min, max, length.out = bins + 1), strict = TRUE)`.
`bins`	Number of Likert categories.
`breaks`	Manual interior cut-points, read as `draw_ordered()` reads them, so `length(breaks) + 1` categories. An alternative to `min`, `max`, and `bins`. 1.0.2 read the same vector as the full set of bin boundaries; the `strict = TRUE` call above returns that answer.
`labels`	Optional category labels.

Value

An integer vector of category indices, one per element of `x`, or an ordered factor with those `labels` when they are supplied.

Examples

```
fabricate(N = 100, x = rnorm(N),
          likert = draw_likert(x, min = -3, max = 3, bins = 5))
```

`draw_multivariate` *Draw multivariate random variables*

Description

A formula interface for functions that return a matrix of draws (e.g. `MASS::mvrnorm`, `extraDistr::rmnom`). The left-hand side of the formula names the resulting columns; the right-hand side is any expression that returns an N-by-k matrix.

Usage

```
draw_multivariate(formula, sep = "_")
```

Arguments

`formula`	A two-sided formula. LHS: either a single bare name (used as a column prefix, producing `name_1`, `name_2`, ...) or a `c(name1, name2, ...)` call giving explicit column names. RHS: any expression returning an N-by-k numeric matrix.
`sep`	Separator between prefix and index when LHS is a single name. Default `"_"`.

Details

When called inside `fabricate()` without a name on the left of `=`, the resulting columns are appended directly to the data frame.

Value

A tibble with one column per name on the left-hand side of the formula, in that order, and one row per row of the matrix the right-hand side returns.

Examples

```
if (requireNamespace("MASS", quietly = TRUE)) {
  S <- matrix(c(1, 0.6, 0.6, 1), 2, 2)
  draw_multivariate(c(Y1, Y2) ~ MASS::mvrnorm(n = 100, mu = c(0, 0), Sigma = S))
}

# Inside fabricate (unnamed -> columns appended automatically)
if (requireNamespace("MASS", quietly = TRUE)) {
  S <- matrix(c(1, 0.4, 0.4, 1), 2, 2)
  fabricate(N = 50,
            draw_multivariate(c(X, Y) ~ MASS::mvrnorm(n = N, mu = c(0, 0), Sigma = S)))
}
```

`draw_normal_ar` *Draw normally distributed data with serial correlation within clusters*

Description

Draws a first-order autoregressive (AR(1)) process separately inside each cluster, the error structure of a panel in which a unit's shocks persist from one period to the next. Within a cluster, rows are ordered by `time`; the first value is a draw from the stationary distribution and each later one is

$$e_t = r e_{t-1} + \sqrt{1 - r^2}\,\sigma u_t, \quad r = \rho^{\Delta t},$$

where Δt is the gap in `time` since the unit's previous observation and u_t is standard normal. Because the process starts at its stationary distribution, every period has standard deviation `sd` and there is no burn-in. Two observations of the same unit k periods apart have correlation ρ^k. With equally spaced periods this is the ordinary AR(1), and with gaps it is the continuous-time version, so an unbalanced panel needs no special handling. This is the process that `nlme::corAR1()` and `nlme::corCAR1()` assume.

Usage

```
draw_normal_ar(clusters, time, rho, mean = 0, sd = 1)
```

Arguments

`clusters`	Vector of unit IDs. Each unit gets its own independent series.
`time`	Vector giving each row's position in its unit's series. Numeric, a `Date` (read in days, so `rho` is the correlation from one day to the next), or character or factor values that read as numbers, which is what `cross_levels` produces for a level's ID. A factor is read by its labels, never by the order of its levels. A unit may not have two rows at the same time.
`rho`	Correlation between consecutive periods, strictly between -1 and 1. A negative `rho` needs whole-number gaps in `time`, since $\rho^{\Delta t}$ is undefined otherwise.
`mean`	Mean, a scalar or one value per row. Default 0.
`sd`	Standard deviation of the process in every period. Default 1. This is the marginal standard deviation, the spread of the values in any one period, so changing `rho` changes persistence and leaves the variance alone. `stats::arima.sim()` takes the standard deviation of each period's new shock instead, which is `sd * sqrt(1 - rho^2)` here: the same number there gives a process `1 / sqrt(1 - rho^2)` times wider, 2.3 times at `rho = 0.9`.

Details

The result is a latent normal variable. For a binary or count outcome with the same persistence, pass it on: as `latent` to `draw_binary` with a probit link, or as `quantile_y = pnorm(e)` to `draw_count` and the other `draw_*` functions. The serial correlation of the outcome is then weaker than `rho`, as with `correlate`.

The function draws persistent errors, not state dependence: a model in which Y_t depends on Y_{t-1} itself is a different model.

Value

A double vector of the same length as `clusters`, in the order the rows were given.

Examples

```
panel <- fabricate(
  units = declare_level(N = 50, u = rnorm(N)),
  periods = declare_level(N = 10),
  obs = cross_levels(
    .by = c("units", "periods"),
    e = draw_normal_ar(clusters = units, time = periods, rho = 0.7),
    Y = u + e
  )
)

# the lag-1 correlation of e within unit is close to 0.7
panel <- panel[order(panel$units, as.numeric(panel$periods)), ]
same_unit <- panel$units[-1] == panel$units[-nrow(panel)]
cor(panel$e[-1][same_unit], panel$e[-nrow(panel)][same_unit])
```

---

draw_normal_icc *Draw normally distributed data with a target intra-cluster correlation*

Description

Uses a two-component variance model: each observation's value is the sum of a cluster-level draw (between-cluster variance) and an individual-level draw (within-cluster variance). The ICC equals $\sigma_b^2/(\sigma_b^2 + \sigma_w^2)$.

Usage

```
draw_normal_icc(
  clusters,
  ICC = NULL,
  mean = 0,
  sd = NULL,
  sd_between = NULL,
  total_sd = NULL,
  N = NULL
)
```

Arguments

`clusters`	Vector of cluster IDs (factor, integer, or character).
`ICC`	Target intra-cluster correlation, in $[0, 1]$. The endpoints are the degenerate cases rather than errors: at 0 the cluster variable does no work, and at 1 every unit in a cluster takes the same value.

mean	Grand mean or per-cluster mean vector. Default 0. May be given per cluster or already expanded to one value per unit.
sd	Within-cluster standard deviation.
sd_between	Between-cluster standard deviation.
total_sd	Overall standard deviation, $\sqrt{\sigma_b^2 + \sigma_w^2}$. This is a parameter of the distribution, not a rescaling of the draw, so the realised sd() varies from draw to draw as any other sample statistic does. With ICC, it gives $\sigma_{between} = s\sqrt{ICC}$ and $\sigma_{within} = s\sqrt{1 - ICC}$. fabricatr 1.x instead rescales the finished vector so that its sample standard deviation is exactly total_sd every time; see the package vignette for why this diverges.
N	Optional; must equal length(clusters) when provided.

Details

Supply any two of ICC, sd (within-cluster), sd_between (between-cluster), and total_sd. The other two follow from $\sigma_{total}^2 = \sigma_b^2 + \sigma_w^2$. Supplying ICC alone is shorthand for sd = 1.

Value

A double vector of the same length as clusters, in the order the clusters were given.

Examples

```
clusters <- rep(1:10, each = 20)
y <- draw_normal_icc(clusters = clusters, ICC = 0.4)
summary(lm(y ~ factor(clusters)))$r.squared  # approx 0.4
```

---

draw_ordered *Draw ordered categorical outcomes from a latent variable*

Description

Cuts a continuous latent variable at the supplied breaks to produce ordered discrete categories. A natural companion to latent-variable specifications of survey responses.

Usage

```
draw_ordered(
  x = latent,
  breaks,
  labels = NULL,
  N = length(x),
  strict = FALSE,
  latent = NULL,
  link = "identity",
  break_labels = NULL
)
```

Arguments

x	Latent continuous variable (numeric vector).
breaks	Numeric vector of cut-points in ascending order. The vector should span the range of x; values outside the range are placed in the outermost categories unless strict = TRUE. An infinite endpoint bounds the scale rather than cutting it, so c(-Inf, 0, Inf) gives the same two categories as 0.
labels	Optional character vector of category labels, one per category. That is length(breaks) + 1 for interior cut-points, one fewer for each infinite endpoint, and length(breaks) - 1 when strict = TRUE. When supplied, returns an ordered factor.
N	Must equal length(x) if it is given at all, and is an error otherwise. It cannot change how many values come back, which is always length(x): unlike draw_binary() and the rest of the family, draw_ordered() has never recycled to N, in either version. 1.x checks N against a recycling rule it does not apply.
strict	If TRUE, observations outside breaks are coded NA instead of being placed in the outermost category.
latent	Alias for x (kept for compatibility).
link	Must be "identity", the only link this function has: breaks cut the latent variable on the scale it arrives. Anything else is an error, as it is in fabricatr 1.x.
break_labels	fabricatr 1.x's name for labels. Accepted with a warning.

Value

An integer vector of category indices, one per element of x, or an ordered factor with those labels when they are supplied. With strict = TRUE, values outside breaks are NA.

Examples

```
fabricate(N = 200, x = rnorm(N),
          resp = draw_ordered(x, breaks = c(-1, 0, 1),
                              labels = c("disagree", "neutral",
                                         "agree", "strongly agree")))
```

---

draw_quantile *Draw quantile bucket assignments*

Description

Draws N new bucket assignments, type equally sized buckets in random order, by cutting N uniform draws with split_quantile. To bucket a variable you already have, use split_quantile on it directly.

Usage

```
draw_quantile(type, N)
```

Arguments

type	Number of buckets.
N	Number of observations.

Value

A factor of length N with levels 1 through type, unordered for the reason given under split_quantile.

Examples

```
draw_quantile(type = 4, N = 100)
```

---

fabricate *Fabricate a data frame*

Description

The main entry point for fabricatr. Builds a data frame column by column, supporting hierarchical levels, multi-column outputs, and starting from existing data.

Usage

```
fabricate(..., N = NULL, ID_label = "ID", data = NULL)
```

Arguments

...	Column definitions or level calls (add_level, nest_level, declare_level, import_level, cross_levels, link_levels, modify_level, potential_outcomes, draw_multivariate). N must be supplied by name.
N	Number of rows. Made available as a scalar integer N inside all column expressions; it does not appear as a column in the output unless you write it explicitly (e.g. n_obs = N).
ID_label	Name of the unit ID column created alongside N. Defaults to "ID"; set to NA to suppress it. Ignored when data is supplied or when the data frame is built from levels (each level's ID column is named after the level).
data	Optional existing data frame to start from.

Value

A tibble with one row per unit of the lowest level and columns in the order they were declared. Each level contributes an ID column named after the argument it was assigned to, placed before that level's own columns; a flat call contributes one named by ID_label, and ID_label = NA suppresses it. A call that starts from data adds no ID column at all. A nested level's IDs are unique across the whole frame rather than restarting inside each parent, so nesting 2 citizens in each of 3 villages numbers the citizens "1" through "6".

ID column type

ID columns are zero-padded character strings, as in fabricatr 1.x, so that an ID is a label rather than a quantity. The padding keeps character sort order matching numeric order, and the character type keeps a cluster ID from being read as a linear term by a model formula. The padding width is set by the number of units at that level, so a frame can hold `clusters` running `"1"` to `"4"` alongside `units` running `"01"` to `"12"`.

Counting the rows a level is building

`N` names the number of rows the current level is building, so `rnorm(N)` draws one value per row. `n()` answers the same question as a function call and is available in every declaration. Two things make it worth reaching for. It cannot be confused with a parameter of the design: `N` is a name, so a design that also has an `N` in the workspace can leave a reader unsure which one an expression reads, and `redesign()` rebinds names, where `n()` always means the rows in hand. And a data set carrying a column called `n` keeps it: `sum(n)` reads the column and `n()` reads the count.

Examples

```
# Flat fabrication
fabricate(N = 100, Y = rnorm(N), X = rbinom(N, 1, 0.5))

# n() is the number of rows the level is building
fabricate(N = 10, Y = rnorm(n()))

# Hierarchical: 20 villages, 5 citizens each
fabricate(
  villages = add_level(N = 20, v_income = rnorm(N)),
  citizens = nest_level(N = 5, income = v_income + rnorm(N))
)

# Starting from existing data
df <- data.frame(x = 1:10)
fabricate(data = df, y = x ^ 2)
```

`fabricate_with_dots` *Fabricate from a list of quosures*

Description

The programmatic entry point, for a caller that has already captured the expressions it wants fabricated. [fabricate()] captures its `...` with `enquos()`, which such a caller cannot use: splicing captured quosures back in with `!!!` renders them as formulas, and `fabricate()` then captures those, one level too deep. `fabricate_with_dots()` takes the captured list as it stands and builds the frame [fabricate()] would have built from the same expressions.

Usage

```
fabricate_with_dots(data = NULL, dots, ID_label = "ID")
```

Arguments

data	Optional existing data frame to start from, as in [fabricate()].
dots	A named list of quosures, such as [rlang::quos()] or [rlang::enquos()] returns. A quosure named N sets the number of rows and does not become a column, exactly as fabricate(N =) does. Every other element needs a name.
ID_label	Name of the unit ID column, or NA to suppress it. See [fabricate()].

Details

DeclareDesign is the caller this exists for. Its declare_model(), declare_measurement(), declare_assignment() and declare_sampling() each capture their own arguments, so that a design carries its expressions as a value rather than as a view onto the environment it was written in, and then pass the captured list here.

Value

A tibble, the same one [fabricate()] returns for the same expressions.

See Also

[fabricate()], which is what to call when you hold the expressions rather than their captures.

Examples

```
dots <- rlang::quos(N = 10, Y = rnorm(N), X = Y > 0)
fabricate_with_dots(dots = dots)

# The same frame without the row id, which is how a step that adds columns
# to data already in hand calls it.
fabricate_with_dots(dots = rlang::quos(Y2 = Y * 2),
                    data = fabricate_with_dots(dots = dots),
                    ID_label = NA)
```

---

import_level *Import an existing data frame as a level*

Description

Brings a data frame into a fabricate call as a level of its own, so that real data can be crossed or linked with other real data. fabricate(data =) starts from one frame and builds down from it; import_level makes each frame a named level, which is the thing cross_levels and link_levels work on.

Usage

```
import_level(data, ..., .id = NULL)
```

Arguments

`data`	A data frame. Its rows are the level's units.
`...`	Column expressions evaluated against the imported columns, as in any other level. `N` is the number of rows imported.
`.id`	Optional string naming the column of `data` that already identifies its rows. Left off, a column named after the level is used when the data has one, the way a join matches on a shared name, and otherwise fabricatr numbers the rows itself.

Value

A `fabricatr_level` object, meaningful only as a named argument to `fabricate`. It contributes the imported rows, an ID column named after the argument it is assigned to, and one column per expression in `....` Like `declare_level`, it stands alone rather than nesting into the frame in hand, so it can be named in a later `cross_levels` or `link_levels`.

Which column is the ID

A level's ID column is named after the level, and `import_level` only decides what goes in it. Three cases, in order:

- `.id = "pid"` uses that column of the imported data, renamed to the level's name. It has to exist, and its values have to be distinct and non-missing, because a level's rows are its units.
- Left off, a column already named after the level is used as it stands: `individuals = import_level(df)` where `df` has an `individuals` column keeps that column rather than putting a second ID beside it.
- Otherwise the rows are numbered, as `add_level` numbers them.

A column called `N` is refused. `N` is the number of rows the level is building in every expression it evaluates, so the two cannot both be in view; rename the column before importing it, or use `fabricate(data = )`, which evaluates nothing at a level and keeps it.

An imported ID keeps its own type and values, where a fabricated one is a zero-padded character string. That is the point of importing it: the column still matches the same key in the data it came from. It also means an integer key stays an integer, so a model formula reading it will treat it as a quantity unless you make it a factor.

Examples

```
individuals <- data.frame(individuals = c("ann", "bob", "cyd"),
                          ind_shock = c(-0.4, 0.1, 0.8))
periods <- data.frame(year = 2020:2022, period_shock = c(0.2, -0.1, 0.3))

# Each frame keeps its own key: `individuals` by name, `year` by `.id`
fabricate(
  individuals = import_level(individuals),
  period      = import_level(periods, .id = "year"),
  obs         = cross_levels(
    .by = c("individuals", "period"),
    Y = ind_shock + period_shock + rnorm(N)
```

```
  )
)

# Imported units with fabricated units nested inside them
fabricate(
  villages = import_level(data.frame(villages = c("v1", "v2"),
                                     v_income = c(10, 12))),
  citizens = nest_level(N = 3, income = v_income + rnorm(N))
)
```

`join_using` *Name the levels to cross or link*

Description

fabricatr 1.x's helper for 'by = join_using(countries, years)'. In 2.0 the levels are named directly, '.by = c("countries", "years")', and this is kept so designs written for 1.x run unchanged. It returns exactly that character vector, so it is also harmless in front of '.by'.

Usage

```
join_using(...)
```

Arguments

`...`	Bare level names, or strings.

Value

A character vector of level names.

Examples

```
join_using(countries, years)
```

`link_levels` *Sample N rows from a Cartesian product with optional correlation*

Description

Draws N rows from the cross-product of the specified levels. When `rho` or `sigma` is non-zero, row assignments are correlated via a Gaussian copula so that units with high values on one level's variable tend to be paired with units with high values on the other level's variable.

Usage

```
link_levels(N, .by, ..., rho = 0, sigma = NULL)
```

Arguments

N	Number of rows to sample from the product.
.by	Character vector of level names, usually two.
...	Additional column expressions evaluated after linking.
rho	Scalar correlation of the Gaussian copula that links the levels' row assignments (default 0 = independent). It is not the realized rank correlation, which is smaller by the copula's own relation, $(6/\pi)\arcsin(\rho/2)$: `rho = 0.7` lands at 0.68 and `rho = 0.3` at 0.283. Ignored if `sigma` is provided. Comes after `...`, so a column called `r` or `s` is a column and not a partial match.
sigma	Square correlation matrix (dimension = `length(.by)`).

Details

The correlated draw takes one code path, so a given seed produces the same data on every machine. fabricatr 1.x switches to `mvnfast::rmvn()` when that package is installed, and passes it a core count, so its numbers move when either changes. This is the second and last place where 2.0 can give different numbers than 1.x, and it only differs when `mvnfast` is installed.

Value

A `fabricatr_level` object, meaningful only as a named argument to `fabricate`. It contributes `N` rows drawn from the same product `cross_levels` would build, with replacement, so a combination can appear more than once and others not at all, and `N` may exceed the size of the product. Columns are as for `cross_levels`.

Examples

```
fabricate(
  primary   = declare_level(N = 20, p_quality = runif(N, 1, 10)),
  secondary = declare_level(N = 15, s_quality = runif(N, 1, 10)),
  students  = link_levels(
    N = 200, .by = c("primary", "secondary"), rho = 0.5,
    score = p_quality + s_quality + rnorm(N)
  )
)
```

---

modify_level *Modify columns of an existing level, or of the rows in hand*

Description

Adds or overwrites columns, as `dplyr::mutate` does. Named after a level that already exists, it works at that level: the expressions are evaluated once per unit of the level, `N` is the number of units, and the result is written back to every row of each unit. Left unnamed, it works on the rows in hand, with an optional `.by` for split-apply-combine on a grouping column.

Usage

```
modify_level(..., .by = NULL)
```

Arguments

`...`	Column expressions to add or overwrite.
`.by`	Optional character string: column name to group by before evaluating expressions.

Details

On a flat frame, unnamed `modify_level(..., .by = "g")` gives the same columns as `group_by(g) |> mutate(...) |> ungroup()`. Its use is that it runs inside the `fabricate()` call, so it is part of a DeclareDesign `declare_model()` step rather than a pipe after it, and that only it can work at a named level.

Value

A `fabricatr_level` object, meaningful only as a named or unnamed argument to `fabricate`. It adds or overwrites columns and changes no rows: the frame it returns has the same number of rows as the one it received.

Which columns a level can see

Inside `regions = modify_level(...)`, the columns in view are the ones that take a single value per region: those built at the regions level or above it. A column built at a level nested inside regions varies within a region, so it is out of view, as it is in fabricatr 1.x. To summarise such a column per region, leave the level name off and group instead: `modify_level(mean_b = mean(b), .by = "regions")`.

Examples

```
# At a named level: one draw per region, written to every city in it
fabricate(
  regions = add_level(N = 3, a = 1:3),
  cities  = add_level(N = 2, b = rnorm(N)),
  regions = modify_level(z = rnorm(N), a2 = a * 2)
)

# On the rows in hand, grouped
fabricate(
  N = 50,
  cluster = sample(1:5, N, replace = TRUE),
  Y = rnorm(N),
  modify_level(cluster_mean = mean(Y), .by = "cluster")
)
```

nest_level *Nest a level in the one before it (same as a later add_level())*

Description

For each row in the current data frame, creates N child rows. This is what add_level does after the first level of a call; nest_level() is the spelling that says so. Parent columns are replicated across children. N may be a scalar (same count for every parent) or a vector of length nrow of the parent (variable children per parent).

Usage

```
nest_level(N, ...)
```

Arguments

N	Rows per parent. Scalar or per-parent vector.
...	Column expressions. Note that the N visible inside these expressions is the total number of rows the level creates, not the per-parent count in the N argument: nesting 5 citizens in each of 20 villages makes N equal to 100 here. The behaviour matches fabricatr 1.x, and it is what makes rnorm(N) draw independently for every village rather than drawing five values and reusing them.

Value

A fabricatr_level object, meaningful only as a named argument to fabricate. It replaces each row of the current frame with N child rows, repeating the parent's columns down them, and adds an ID column named after the argument it is assigned to. That ID is unique across the whole frame rather than restarting within each parent.

Examples

```
fabricate(
  villages = add_level(N = 20, v_income = rnorm(N)),
  citizens = nest_level(N = 5, income = v_income + rnorm(N))
)
```

potential_outcomes *Build potential outcomes columns*

Description

For each combination of condition values, evaluates the RHS of x with the assignment variable(s) set to those values, producing one column per condition. The column names follow the pattern <outcome>_<var>_<value>.

Usage

```
potential_outcomes(x, conditions = list(Z = c(0, 1)), sep = "_")
```

Arguments

x	A two-sided formula. LHS: outcome variable name. RHS: expression for the potential outcome (may reference the assignment variable and other columns).
conditions	Named list mapping assignment variable name(s) to their possible values. Default `list(Z = c(0, 1))`.
sep	Column-name separator. Default `"_"`. Give `reveal_outcomes()` the same `sep`.

Details

When called inside `fabricate()` without a left-hand name (i.e. as an unnamed argument), the resulting columns are appended automatically.

Value

A data frame with one column per combination of the values in `conditions`, named `<outcome><sep><var><sep><value>` and, with more than one assignment variable, carrying one such pair per variable: `Y_Z1_0_Z2_0`, `Y_Z1_1_Z2_0`, `Y_Z1_0_Z2_1`, `Y_Z1_1_Z2_1`, the first variable varying fastest. The names are kept as written even where they are not syntactic R names, so a negative condition gives `Y_Z_-1` and a condition `"b c"` gives `Y_Z_b c`; refer to such a column with backticks or `.data`. Called inside `fabricate()` without a name, these columns are appended to the frame.

Examples

```
fabricate(
  N = 10, U = rnorm(N),
  potential_outcomes(Y ~ 0.5 * Z + U)
)

# Three conditions
fabricate(
  N = 10, U = rnorm(N),
  potential_outcomes(Y ~ Z * 0.5 + U, conditions = list(Z = 0:2))
)

# Multi-arm factorial
fabricate(
  N = 10, U = rnorm(N),
  potential_outcomes(Y ~ 0.3 * Z1 + 0.5 * Z2 + U,
                     conditions = list(Z1 = 0:1, Z2 = 0:1))
)
```

recycle *Recycle a vector to the length of the level being built*

Description

fabricatr 1.x's helper for filling a level with a short vector. 2.0 recycles automatically whenever the vector's length divides the level evenly, so this is only ever a no-op made explicit, and is kept so designs written for 1.x run unchanged.

Usage

```
recycle(x, .N = NULL)
```

Arguments

x	A vector.
.N	Length to recycle to. Left off inside a level.

Details

Left off, '.N' is the 'N' of the level being built. 1.x found it by walking the call stack; a 2.0 column expression is evaluated against a data mask instead, so 'N' is read out of the calling frame, which is that mask.

Value

'x', recycled to length '.N'.

Examples

```
fabricate(villages = add_level(N = 6, arm = recycle(c("a", "b", "c"))))
recycle(c("a", "b"), .N = 6)
```

resample_data *Resample a data frame, with optional hierarchical cluster resampling*

Description

For simple bootstrap resampling, call `resample_data(data)` with no other arguments. For cluster resampling, supply `N` and `ID_labels`. For multi-level hierarchical resampling, supply vectors for both.

Usage

```
resample_data(data, N, ID_labels = NULL, unique_labels = FALSE)
```

Arguments

data	A data frame.
N	Resample size. May be: • Missing: resample nrow(data) units (standard bootstrap). • A single unnamed number, with no ID_labels: resample that many rows with replacement. • A named integer vector: names are cluster ID column names (from outermost to innermost), values are the number of units to resample at each level. Use the constant ALL to pass through all units at a level. • An unnamed integer vector: same as above but supply ID_labels separately.
ID_labels	Character vector of cluster ID column names (outermost to innermost). Required when N is unnamed.
unique_labels	Logical. When TRUE, adds a <ID>_unique column with globally unique labels after resampling (useful for multi-level bootstraps where the same cluster is drawn multiple times). Default FALSE.

Value

A tibble of the resampled rows: nrow(data) of them for the default bootstrap, N for an unnamed scalar N, and for a hierarchical call the sizes requested at each level multiplied together. ID columns are carried over from the rows that were drawn rather than regenerated, so a cluster drawn twice appears twice under one ID; pass unique_labels = TRUE to add an <ID>_unique column that tells the copies apart.

Examples

```
# Simple bootstrap
df <- fabricate(N = 50, Y = rnorm(N))
boot <- resample_data(df)
nrow(boot)

# Cluster bootstrap: resample 10 of 20 clusters
clustered <- fabricate(
  clusters = add_level(N = 20, gdp = rnorm(N)),
  units    = nest_level(N = 5, Y = gdp + rnorm(N))
)
resample_data(clustered, N = c(clusters = 10))
```

---

reveal_outcomes *Reveal observed outcomes from potential outcomes columns*

Description

Implements the switching equation: for each unit, selects the potential outcome column corresponding to that unit's realized assignment.

Usage

```
reveal_outcomes(x, sep = "_")
```

Arguments

x	A formula of the form `outcome ~ assignment` or `outcome ~ Z1 + Z2` for factorial assignments. The outcome name and assignment variable(s) must match columns already in the data (typically created by `potential_outcomes`).
sep	Column-name separator. Default `"_"`. It must match the `sep` given to `potential_outcomes()`.

Value

A vector with one element per row, holding each unit's outcome under the assignment it actually received, and carrying the type of the potential outcome columns it reads. A factor reveals as a factor, keeping its levels and their order, and a `Date` as a `Date`. Where the conditions do not agree on one set of factor levels there are none to keep, so the result is `character` and a warning says so.

Examples

```
dat <- fabricate(
  N = 10, U = rnorm(N),
  potential_outcomes(Y ~ 0.5 * Z + U)
)
fabricate(
  data = dat,
  Z = rbinom(N, 1, 0.5),
  Y = reveal_outcomes(Y ~ Z)
)

# A separator other than "_" is given to both
fabricate(
  N = 10, U = rnorm(N),
  potential_outcomes(Y ~ 0.5 * Z + U, sep = "."),
  Z = rbinom(N, 1, 0.5),
  Y = reveal_outcomes(Y ~ Z, sep = ".")
)
```

`split_quantile` *Split data into quantile buckets*

Description

Cuts a numeric vector you already have into `type` equally sized buckets at its sample quantiles, so bucket `1` holds the lowest values. To draw new bucket assignments without a variable to cut, use `draw_quantile`.

Usage

```
split_quantile(x, type)
```

Arguments

x	Numeric vector.
type	Number of buckets (2 = median split, 3 = terciles, etc.).

Value

A factor with levels 1 through type, one per element of x. It is deliberately *unordered*, as in fabricatr 1.x: an ordered factor would make lm fit polynomial contrasts (q.L, q.Q, q.C) where the same script under 1.x gets treatment contrasts (q2, q3, q4), and report different coefficients without saying why.

Examples

```
split_quantile(rnorm(100), type = 4)
```

Index

add_level, 2, *24*
ALL, 3

correlate, 4, *13*
cross_levels, 5, *13*

declare_level, 5
draw_binary, 6, *13*
draw_binary_icc, 7
draw_binomial, 8
draw_categorical, 9
draw_count, 10, *13*
draw_likert, 11
draw_multivariate, 12
draw_normal_ar, 13
draw_normal_icc, 14
draw_ordered, 15
draw_quantile, 16, *28*

fabricate, 17
fabricate_with_dots, 18

import_level, 19

join_using, 21

link_levels, 21

modify_level, 22

nest_level, *2*, 24

potential_outcomes, 24

recycle, 26
resample_data, 26
reveal_outcomes, 27

split_quantile, *16, 17*, 28