Package 'fabricatr'

June 29, 2022

Type Package

Title Imagine Your Data Before You Collect It

Version 1.0.0

Description Helps you imagine your data before you collect it. Hierarchical data structures and correlated data can be easily simulated, either from random number generators or by resampling from existing data sources. This package is faster with 'data.table' and 'mvnfast' installed.

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

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

Depends R (>= 3.5.0)

Imports rlang (>= 1.0.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Suggests testthat, data.table, mvnfast, MASS, extraDistr

NeedsCompilation no

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

Maintainer Graeme Blair <graeme.blair@gmail.com>

Repository CRAN

Date/Publication 2022-06-29 12:50:02 UTC

R topics documented:

correlate . . . 2
cross_levels . . . 3
draw_binary_icc . . . 4
draw_discrete . . . 5
draw_likert . . . 8
draw_multivariate . . . 9
draw_normal_icc . . . 10
fabricate . . . 12
fabricatr . . . 14
join_using . . . 14
potential_outcomes . . . 15
recycle . . . 17
resample_data . . . 17
reveal_outcomes . . . 19
split_quantile . . . 20

Index **21**

`correlate` *Perform generation of a correlated random variable.*

Description

This function is EXPERIMENTAL, and we cannot guarantee its properties for all data structures. Be sure to diagnose your design and assess the distribution of your variables.

Usage

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

Arguments

`draw_handler`	The unquoted name of a function to generate data. Currently, `draw_binary`, `draw_binomial`, and `draw_count` are supported.
`...`	The arguments to draw_handler (e.g. `prob`, `mean`, etc.)
`given`	A vector that can be ordered; the reference distribution X that Y will be correlated with.
`rho`	A rank correlation coefficient between -1 and 1.

Details

In order to generate a random variable of a specific distribution based on another variable of any distribution and a correlation coefficient `rho`, we map the first, known variable into the standard normal space via affine transformation, generate the conditional distribution of the resulting variable as a standard normal, and then map that standard normal back to the target distribution. The result should ensure, in expectation, a rank-order correlation of `rho`.

Examples

```
# Generate a variable of interest
exam_score <- pmin(100, rnorm(n = 100, mean = 80, sd = 10))

# Generate a correlated variable using fabricatr variable generation
scholarship_offers <- correlate(given = exam_score, rho = 0.7,
                                draw_count, mean = 3)

# Generate a correlated variable using base R distributions
final_grade <- pmax(100, correlate(given = exam_score, rho = 0.7,
                                   rnorm, mean = 80, sd = 10))
```

`cross_levels` *Creates panel or cross-classified data*

Description

This function allows the user to create data structures that are paneled or cross-classified: where one level of observation draws simultaneously from two or many source levels. Common examples of panels include country-year data which have country-level and year-level characteristics.

Usage

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

link_levels(N = NULL, by = NULL, ...)
```

Arguments

`by`	The result of a call to `join_using()` which specifies how the cross-classified data will be created
`...`	A variable or series of variables to add to the resulting data frame after the cross-classified data is created.
`N`	The number of observations in the resulting data frame. If `N` is NULL or not provided, the join_using will be an "outer product" – merging each row of each provided data frame with each other data frame to make a full panel.

Details

By specifying the appropriate arguments in `join_using()` within the function call, it is possible to induce correlation in cross-classified data.

Value

data.frame

Examples

```
# Generate full panel data
panel <- fabricate(
 countries = add_level(N = 20, country_shock = runif(N, 1, 10)),
 years = add_level(N = 20, year_shock = runif(N, 1, 10), nest=FALSE),
 obs = cross_levels(by = join_using(countries, years), GDP_it = country_shock + year_shock)
)

# Include an "N" argument to allow for cross-classified
# data.
students <- fabricate(
 primary_school = add_level(N = 20, ps_quality = runif(N, 1, 10)),
 secondary_school = add_level(N = 15, ss_quality = runif(N, 1, 10), nest=FALSE),
 students = link_levels(N = 500, by = join_using(primary_school, secondary_school))
)
head(students)

# Induce a correlation structure in cross-classified data by providing
# rho.
students <- fabricate(
 primary_school = add_level(N = 20, ps_quality = runif(N, 1, 10)),
 secondary_school = add_level(N = 15, ss_quality = runif(N, 1, 10), nest=FALSE),
 students = link_levels(N = 500, by = join_using(ps_quality, ss_quality, rho = 0.5))
)
cor(students$ps_quality, students$ss_quality)
```

`draw_binary_icc` *Draw binary data with fixed intra-cluster correlation.*

Description

Data is generated to ensure inter-cluster correlation 0, intra-cluster correlation in expectation ICC. Algorithm taken from Hossein, Akhtar. "ICCbin: An R Package Facilitating Clustered Binary Data Generation, and Estimation of Intracluster Correlation Coefficient (ICC) for Binary Data".

Usage

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

Arguments

`prob`	A number or vector of numbers, one probability per cluster. If none is provided, will default to 0.5.
`N`	(Optional) A number indicating the number of observations to be generated. Must be equal to length(clusters) if provided.
`clusters`	A vector of factors or items that can be coerced to clusters; the length will determine the length of the generated data.

ICC — A number indicating the desired ICC, if none is provided the default ICC will be 0.

Value

A vector of binary numbers corresponding to the observations from the supplied cluster IDs.

Examples

```
# Divide units into clusters
clusters = rep(1:5, 10)

# Default probability 0.5, default ICC 0
draw_binary_icc(clusters = clusters)

# Specify probability or ICC
corr_draw = draw_binary_icc(prob = 0.5, clusters = clusters, ICC = 0.5)

# Verify ICC of data.
summary(lm(corr_draw ~ as.factor(clusters)))$r.squared
```

---

draw_discrete — *Draw discrete variables including binary, binomial count, poisson count, ordered, and categorical*

Description

Drawing discrete data based on probabilities or latent traits is a common task that can be cumbersome. Each function in our discrete drawing set creates a different type of discrete data: `draw_binary` creates binary 0/1 data, `draw_binomial` creates binomial data (repeated trial binary data), `draw_categorical` creates categorical data, `draw_ordered` transforms latent data into observed ordered categories, `draw_count` creates count data (poisson-distributed).

Usage

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

draw_categorical(
  prob = link(latent),
  N = NULL,
```

```
  latent = NULL,
  link = "identity",
  category_labels = NULL
)

draw_ordered(
  x = link(latent),
  breaks = c(-1, 0, 1),
  break_labels = NULL,
  N = length(x),
  latent = NULL,
  strict = FALSE,
  link = "identity"
)

draw_count(
  mean = link(latent),
  N = length(mean),
  latent = NULL,
  link = "identity",
  quantile_y = NULL
)

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

draw_quantile(type, N)
```

Arguments

prob	A number or vector of numbers representing the probability for binary or binomial outcomes; or a number, vector, or matrix of numbers representing probabilities for categorical outcomes. If you supply a link function, these underlying probabilities will be transformed.
trials	for `draw_binomial`, the number of trials for each observation
N	number of units to draw. Defaults to the length of the vector of probabilities or latent data you provided.
latent	If the user provides a link argument other than identity, they should provide the variable `latent` rather than `prob` or `mean`
link	link function between the latent variable and the probability of a positive outcome, e.g. "logit", "probit", or "identity". For the "identity" link, the latent variable must be a probability.

quantile_y	A vector of quantiles; if provided, rather than drawing stochastically from the distribution of interest, data will be drawn at exactly those quantiles.
category_labels	vector of labels for the categories produced by draw_categorical. If provided, must be equal to the number of categories provided in the prob argument.
x	for draw_ordered, the latent data for each observation.
breaks	vector of breaks to cut a latent outcome into ordered categories with draw_ordered
break_labels	vector of labels for the breaks to cut a latent outcome into ordered categories with draw_ordered. (Optional)
strict	Logical indicating whether values outside the provided breaks should be coded as NA. Defaults to FALSE, in which case effectively additional breaks are added between -Inf and the lowest break and between the highest break and Inf.
mean	for draw_count, the mean number of count units for each observation
type	The number of buckets to split data into. For a median split, enter 2; for terciles, enter 3; for quartiles, enter 4; for quintiles, 5; for deciles, 10.

Details

For variables with intra-cluster correlations, see draw_binary_icc and draw_normal_icc

Value

A vector of data in accordance with the specification; generally numeric but for some functions, including draw_ordered and draw_categorical, may be factor if labels are provided.

Examples

```
# Drawing binary values (success or failure, treatment assignment)
fabricate(N = 3,
   p = c(0, .5, 1),
   binary = draw_binary(prob = p))

# Drawing binary values with probit link (transforming continuous data
# into a probability range).
fabricate(N = 3,
   x = 10 * rnorm(N),
   binary = draw_binary(latent = x, link = "probit"))

# Repeated trials: `draw_binomial`
fabricate(N = 3,
   p = c(0, .5, 1),
   binomial = draw_binomial(prob = p, trials = 10))

# Ordered data: transforming latent data into observed, ordinal data.
# useful for survey responses.
fabricate(N = 3,
   x = 5 * rnorm(N),
   ordered = draw_ordered(x = x,
```

```
                          breaks = c(-Inf, -1, 1, Inf)))

# Providing break labels for latent data.
fabricate(N = 3,
   x = 5 * rnorm(N),
   ordered = draw_ordered(x = x,
                          breaks = c(-Inf, -1, 1, Inf),
                          break_labels = c("Not at all concerned",
                                           "Somewhat concerned",
                                           "Very concerned")))


# Count data: useful for rates of occurrences over time.
fabricate(N = 5,
   x = c(0, 5, 25, 50, 100),
   theft_rate = draw_count(mean=x))

# Categorical data: useful for demographic data.
fabricate(N = 6, p1 = runif(N), p2 = runif(N), p3 = runif(N),
          cat = draw_categorical(cbind(p1, p2, p3)))
```

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

Description

Recode a latent variable into a Likert response variable

Usage

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

Arguments

`x`	a numeric variable considered to be "latent"
`min`	the minimum value of the latent variable
`max`	the maximum value of the latent variable
`bins`	the number of Likert scale values. The latent variable will be cut into equally sized bins as in seq(min, max, length.out = bins + 1)

breaks	A vector of breaks. This option is useful for settings in which equally-sized breaks are inappropriate
labels	An optional vector of labels. If labels are provided, the resulting output will be a factor.

Examples

```
x <- 1:100

draw_likert(x, min = 0, max = 100, bins = 7)
draw_likert(x, breaks = c(-1, 10, 100))
```

---

draw_multivariate *Draw multivariate random variables*

Description

Draw multivariate random variables

Usage

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

Arguments

formula	Formula describing the multivariate draw. The lefthand side is the names or prefix and the right-hand side is the multivariate draw function call, such as mvrnorm from the MASS library or rmnom from the extraDistr library.
sep	Separator string between prefix and variable number. Only used when a single character string is provided and multiple variables created.

Value

tibble

Examples

```
library(MASS)

# draw from multivariate normal distribution
dat <-
  draw_multivariate(c(Y_1, Y_2) ~ mvrnorm(
    n = 500,
    mu = c(0, 0),
    Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2)
```

```
  ))


cor(dat)


# equivalently, you can provide a prefix for the variable names
# (easier if you have many variables)
draw_multivariate(Y ~ mvrnorm(
  n = 5,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2)
))

# within fabricate
fabricate(
  N = 100,
  draw_multivariate(c(Y_1, Y_2) ~ mvrnorm(
    n = N,
    mu = c(0, 0),
    Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2)
  ))
)

# You can also write the following, which works but gives less control over the names
fabricate(N = 100,
Y = mvrnorm(
  n = N,
  mu = c(0, 0),
  Sigma = matrix(c(1, 0.5, 0.5, 1), 2, 2)
))
```

`draw_normal_icc` *Draw normal data with fixed intra-cluster correlation.*

Description

Data is generated to ensure inter-cluster correlation 0, intra-cluster correlation in expectation ICC. The data generating process used in this function is specified at the following URL: https://stats.stackexchange.com/questions/263451/create-synthetic-data-with-a-given-intraclass-correlation-

Usage

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

```
  total_sd = NULL,
  ICC = NULL
)
```

Arguments

mean	A number or vector of numbers, one mean per cluster. If none is provided, will default to 0.
N	(Optional) A number indicating the number of observations to be generated. Must be equal to length(clusters) if provided.
clusters	A vector of factors or items that can be coerced to clusters; the length will determine the length of the generated data.
sd	A number or vector of numbers, indicating the standard deviation of each cluster's error terms – standard deviation within a cluster (default 1)
sd_between	A number or vector of numbers, indicating the standard deviation between clusters.
total_sd	A number indicating the total sd of the resulting variable. May only be specified if ICC is specified and sd and sd_between are not.
ICC	A number indicating the desired ICC.

Details

The typical use for this function is for a user to provide an ICC and, optionally, a set of within-cluster standard deviations, sd. If the user does not provide sd, the default value is 1. These arguments imply a fixed between-cluster standard deviation.

An alternate mode for the function is to provide between-cluster standard deviations, sd_between, and an ICC. These arguments imply a fixed within-cluster standard deviation.

If users provide all three of ICC, sd_between, and sd, the function will warn the user and use the provided standard deviations for generating the data.

Value

A vector of numbers corresponding to the observations from the supplied cluster IDs.

Examples

```
# Divide observations into clusters
clusters = rep(1:5, 10)

# Default: unit variance within each cluster
draw_normal_icc(clusters = clusters, ICC = 0.5)

# Alternatively, you can specify characteristics:
draw_normal_icc(mean = 10, clusters = clusters, sd = 3, ICC = 0.3)

# Can specify between-cluster standard deviation instead:
draw_normal_icc(clusters = clusters, sd_between = 4, ICC = 0.2)
```

```
# Can specify total SD instead:
total_sd_draw = draw_normal_icc(clusters = clusters, ICC = 0.5, total_sd = 3)
sd(total_sd_draw)

# Verify that ICC generated is accurate
corr_draw = draw_normal_icc(clusters = clusters, ICC = 0.4)
summary(lm(corr_draw ~ as.factor(clusters)))$r.squared
```

`fabricate` *Fabricate data*

Description

`fabricate` helps you simulate a dataset before you collect it. You can either start with your own data and add simulated variables to it (by passing `data` to `fabricate()`) or start from scratch by defining `N`. Create hierarchical data with multiple levels of data such as citizens within cities within states using `add_level()` or modify existing hierarchical data using `modify_level()`. You can use any R function to create each variable. Use `cross_levels()` and `link_levels()` to make more complex designs such as panel or cross-classified data.

Usage

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

add_level(N = NULL, ..., nest = TRUE)

modify_level(..., by = NULL)

nest_level(N = NULL, ...)
```

Arguments

`...`	Variable or level-generating arguments, such as `my_var = rnorm(N)`. For `fabricate`, you may also pass `add_level()` or `modify_level()` arguments, which define a level of a multi-level dataset. See examples.
`data`	(optional) user-provided data that forms the basis of the fabrication, e.g. you can add variables to existing data. Provide either `N` or `data` (`N` is the number of rows of the data if `data` is provided). If `data` and `N` are not provided, fabricatr will try to interpret the first un-named argument as either `data` or `N` based on type.
`N`	(optional) number of units to draw. If provided as `fabricate(N = 5)`, this determines the number of units in the single-level data. If provided in `add_level`, e.g. `fabricate(cities = add_level(N = 5))`, `N` determines the number of units in a specific level of a hierarchical dataset.
`ID_label`	(optional) variable name for ID variable, e.g. citizen_ID. Set to NA to suppress the creation of an ID variable.

nest (Default TRUE) Boolean determining whether data in an `add_level()` call will be nested under the current working data frame or create a separate hierarchy of levels. See our vignette for cross-classified, non-nested data for details.

by (optional) quoted name of variable `modify_level` uses to split-modify-combine data by.

Details

We also provide several built-in options to easily create variables, including `draw_binary`, `draw_count`, `draw_likert`, and intra-cluster correlated variables `draw_binary_icc` and `draw_normal_icc`

Value

data.frame

See Also

`link_levels`

Examples

```
# Draw a single-level dataset with a covariate
building_df <- fabricate(
  N = 100,
  height_ft = runif(N, 300, 800)
)
head(building_df)

# Start with existing data instead
building_modified <- fabricate(
  data = building_df,
  rent = rnorm(N, mean = height_ft * 100, sd = height_ft * 30)
)

# Draw a two-level hierarchical dataset
# containing cities within regions
multi_level_df <- fabricate(
 regions = add_level(N = 5),
 cities = add_level(N = 2, pollution = rnorm(N, mean = 5)))
head(multi_level_df)

# Start with existing data and add a nested level:
company_df <- fabricate(
 data = building_df,
 company_id = add_level(N=10, is_headquarters = sample(c(0, 1), N, replace=TRUE))
)

# Start with existing data and add variables to hierarchical data
# at levels which are already present in the existing data.
# Note: do not provide N when adding variables to an existing level
```

```
fabricate(
  data = multi_level_df,
  regions = modify_level(watershed = sample(c(0, 1), N, replace = TRUE)),
  cities = modify_level(runoff = rnorm(N))
)

# fabricatr can add variables that are higher-level summaries of lower-level
# variables via a split-modify-combine logic and the \code{by} argument

multi_level_df <-
 fabricate(
   regions = add_level(N = 5, elevation = rnorm(N)),
   cities = add_level(N = 2, pollution = rnorm(N, mean = 5)),
   cities = modify_level(by = "regions", regional_pollution = mean(pollution))
 )

# fabricatr can also make panel or cross-classified data. For more
# information about syntax for this functionality please read our vignette
# or check documentation for \code{link_levels}:
cross_classified <- fabricate(
  primary_schools = add_level(N = 50, ps_quality = runif(N, 0, 10)),
  secondary_schools = add_level(N = 100, ss_quality = runif(N, 0, 10), nest=FALSE),
  students = link_levels(N = 2000,
                         by = join_using(ps_quality, ss_quality, rho = 0.5),
                         student_quality = ps_quality + 3*ss_quality + rnorm(N)))
```

`fabricatr` *fabricatr package*

Description

fabricatr helps you imagine your data before you collect it. Hierarchical data structures and correlated data can be easily simulated, either from random number generators or by resampling from existing data sources.

`join_using` *Helper function handling specification of which variables to join a cross-classified data on, and what kind of correlation structure needed. Correlation structures can only be provided if the underlying call is a* `link_levels()` *call.*

Description

Helper function handling specification of which variables to join a cross-classified data on, and what kind of correlation structure needed. Correlation structures can only be provided if the underlying call is a `link_levels()` call.

Usage

```
join_using(..., rho = 0, sigma = NULL)
```

Arguments

...	A series of two or more variable names, unquoted, to join on in order to create cross-classified data.
rho	A fixed (Spearman's rank) correlation coefficient between the variables being joined on: note that if it is not possible to make a correlation matrix from this coefficient (e.g. if you are joining on three or more variables and rho is negative) then the `cross_levels()` call will fail. Do not provide `rho` if making panel data.
sigma	A matrix with dimensions equal to the number of variables you are joining on, specifying the correlation for the resulting joined data. Only one of rho and sigma should be provided. Do not provide `sigma` if making panel data.

Examples

```
panels <- fabricate(
  countries = add_level(N = 150, country_fe = runif(N, 1, 10)),
  years = add_level(N = 25, year_shock = runif(N, 1, 10), nest = FALSE),
  obs = cross_levels(
    by = join_using(countries, years),
    new_variable = country_fe + year_shock + rnorm(N, 0, 2)
  )
)

schools_data <- fabricate(
  primary_schools = add_level(N = 20, ps_quality = runif(N, 1, 10)),
  secondary_schools = add_level(
    N = 15,
    ss_quality = runif(N, 1, 10),
    nest = FALSE),
  students = link_levels(
    N = 1500,
    by = join_using(primary_schools, secondary_schools),
    SAT_score = 800 + 13 * ps_quality + 26 * ss_quality + rnorm(N, 0, 50)
  )
)
```

---

potential_outcomes *Build potential outcomes variables*

Description

Function to draw multiple potential outcomes, one for each condition that an assignment variable can be set to.

Usage

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

Arguments

x	Formula describing the potential outcomes with the outcome name on the left hand side and the expression describing the potential outcomes on the right hand side, e.g. `Y ~ 0.1 * Z + rnorm(N)` (this would draw two potential outcomes columns by default, named Y_Z_0 and Y_Z_1).
conditions	A list of conditions for each assignment variable. Defaults to `list(Z = c(0, 1))`.
sep	Separator inserted between the outcome name and the assignment variable name used to construct the potential outcome variable names, defaults to "_".

Examples

```
fabricate(
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z + U)
)

# equivalently,

fabricate(
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z + U,
                     conditions = list(Z = c(0, 1)))
)

fabricate(
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z + U,
                     conditions = list(Z = c(1, 2, 3)))
)

fabricate(
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z1 + 0.3 * Z2 + 0.5 * Z1 * Z2 + U,
                     conditions = list(Z1 = c(0, 1),
                                       Z2 = c(0, 1)))
)
```

recycle *Expands data to a given length through recycling.*

Description

This function is a helper function designed call `rep_len` to expand the length of a data vector, but which can dynamically retrieve N from the surrounding level call for use in fabricatr.

Usage

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

Arguments

`x`	Data to recycle into length `N`
`.N`	the length to recycle the data to, typically provided implicitly by a or fabricate call wrapped around the function call.

Value

A vector of data padded to length `N`

Examples

```
fabricate(
  N = 15,
  month = recycle(month.abb)
)
```

resample_data *Resample data, including hierarchical data*

Description

This function allows you to resample any data frame. The default mode performs a single resample of size `N` with replacement. Users can also specify more complex resampling strategies to resample hierarchical data.

Usage

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

Arguments

data	A data.frame, usually provided by the user.
N	The number of sample observations to return. If N is a single scalar and no labels are provided, N will specify the number of unit observations to resample. If N is named, or if the ID_labels argument is specified (in which case, both N and ID_labels should be the same length), then the units resampled will be values of the levels resampled (this is useful for, e.g., cluster resampling). If N is the constant ALL for any level, all units of this level will be transparently passed through to the next level of resampling.
ID_labels	A character vector of the variables that indicate the data hierarchy, from highest to lowest (i.e., from cities to citizens).
unique_labels	A boolean, defaulting to FALSE. If TRUE, fabricatr will created an extra data frame column depicting a unique version of the ID_label variable resampled on, called <ID_label>_unique.

Value

A data.frame

Examples

```
# Resample a dataset of size N without any hierarchy
baseline_survey <- fabricate(N = 50, Y_pre = rnorm(N))
bootstrapped_data <- resample_data(baseline_survey)

# Specify a fixed number of observations to return
baseline_survey <- fabricate(N = 50, Y_pre = rnorm(N))
bootstrapped_data <- resample_data(baseline_survey, N = 100)

# Resample by a single level of a hierarchical dataset (e.g. resampling
# clusters of observations): N specifies a number of clusters to return

clustered_survey <- fabricate(
  clusters = add_level(N=25),
  cities = add_level(N=round(runif(25, 1, 5)),
                     population=runif(n = N, min=50000, max=1000000))
)

cluster_resample <- resample_data(clustered_survey, N = 5, ID_labels = "clusters")

# Alternatively, pass the level to resample as a name:
cluster_resample_2 <- resample_data(clustered_survey, N=c(clusters = 5))

# Resample a hierarchical dataset on multiple levels
my_data <-
fabricate(
  cities = add_level(N = 20, elevation = runif(n = N, min = 1000, max = 2000)),
  citizens = add_level(N = 30, age = runif(n = N, min = 18, max = 85))
)
```

```
# Specify the levels you wish to resample:
my_data_2 <- resample_data(my_data, N = c(3, 5),
                           ID_labels = c("cities", "citizens"))

# To resample every unit at a given level, use the ALL constant
# This example will resample 10 citizens at each of the cities:

passthrough_resample_data <- resample_data(my_data, N = c(cities=ALL, citizens=10))

# To ensure a column with unique labels (for example, to calculate block-level
# statistics irrespective of sample choices), use the unique_labels=TRUE
# argument -- this will produce new columns with unique labels.

unique_resample <- resample_data(my_data, N = c(cities=5), unique_labels = TRUE)
```

`reveal_outcomes` *Reveal outcomes*

Description

Implements a generalized switching equation. Reveals observed outcomes from multiple potential outcomes variables and an assignment variable.

Usage

```
reveal_outcomes(x)
```

Arguments

`x`	A formula with the outcome name on the left hand side and assignment variables on the right hand side (e.g., `Y ~ Z`).

Examples

```
dat <- fabricate(
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z + U)
)

fabricate(
  data = dat,
  Z = rbinom(N, 1, prob = 0.5),
  Y = reveal_outcomes(Y ~ Z)
)

fabricate(
```

```
  N = 10,
  U = rnorm(N),
  potential_outcomes(Y ~ 0.1 * Z1 + 0.3 * Z2 + 0.5 * Z1 * Z2 + U,
                     conditions = list(Z1 = c(0, 1),
                                       Z2 = c(0, 1))),
  Z1 = rbinom(N, 1, prob = 0.5),
  Z2 = rbinom(N, 1, prob = 0.5),
  Y = reveal_outcomes(Y ~ Z1 + Z2)
)
```

`split_quantile` *Split data into quantile buckets (e.g. terciles, quartiles, quantiles, deciles).*

Description

Survey data is often presented in aggregated, depersonalized form, which can involve binning underlying data into quantile buckets; for example, rather than reporting underlying income, a survey might report income by decile. `split_quantile` can automatically produce this split using any data `x` and any number of splits 'type.

Usage

```
split_quantile(x = NULL, type = NULL)
```

Arguments

`x`	A vector of any type that can be ordered – i.e. numeric or factor where factor levels are ordered.
`type`	The number of buckets to split data into. For a median split, enter 2; for terciles, enter 3; for quartiles, enter 4; for quintiles, 5; for deciles, 10.

Examples

```
# Divide this arbitrary data set in 3.
data_input <- rnorm(n = 100)
split_quantile(x = data_input, type = 3)
```

Index

add_level *(*fabricate*)*, 12

correlate, 2
cross_levels, 3

draw_binary, *13*
draw_binary *(*draw_discrete*)*, 5
draw_binary_icc, 4, *7*, *13*
draw_binomial *(*draw_discrete*)*, 5
draw_categorical *(*draw_discrete*)*, 5
draw_count, *13*
draw_count *(*draw_discrete*)*, 5
draw_discrete, 5
draw_likert, 8, *13*
draw_multivariate, 9
draw_normal_icc, *7*, 10, *13*
draw_ordered *(*draw_discrete*)*, 5
draw_quantile *(*draw_discrete*)*, 5

fabricate, 12
fabricatr, 14

join_using, 14

link_levels, *13*
link_levels *(*cross_levels*)*, 3

modify_level *(*fabricate*)*, 12

nest_level *(*fabricate*)*, 12

potential_outcomes, 15

recycle, 17
resample_data, 17
reveal_outcomes, 19

split_quantile, 20