Package 'fabricatr'

January 25, 2018

Type Package

Title Imagine Your Data Before You Collect It

Version 0.2.0

Date 2018-01-24

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 http://fabricatr.declaredesign.org, https://github.com/DeclareDesign/fabricatr

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

Depends R (>= 3.3.0)

Imports rlang

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 6.0.1

Suggests testthat, dplyr, knitr, rmarkdown, data.table, mvnfast, diagram

VignetteBuilder knitr

NeedsCompilation no

Author Graeme Blair [aut, cre], Jasper Cooper [aut], Alexander Coppock [aut], Macartan Humphreys [aut], Aaron Rudkin [aut], Neal Fultz [ctb]

Maintainer Graeme Blair <graeme.blair@ucla.edu>

Repository CRAN

Date/Publication 2018-01-25 13:18:20 UTC

R topics documented:

`add_level` *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

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

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

modify_level(N = NULL, ...)

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

Arguments

`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.
`...`	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.
`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.

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).
ID_label	(optional) variable name for ID variable, e.g. citizen_ID

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(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
modified_multi_level_df <- fabricate(
```

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

# 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(ps_quality, ss_quality, rho = 0.5),
                         student_quality = ps_quality + 3*ss_quality + rnorm(N)))
```

`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()` 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 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()` 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(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(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(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_binomial — *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). `draw_likert` is an alias to `draw_ordered` that pre-specifies break labels and offers default breaks appropriate for a likert survey question.

Usage

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

draw_categorical(prob, N = NULL, link = "identity")

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

draw_count(mean, N = length(mean), link = "identity")

draw_binary(prob, N = length(prob), link = "identity")

draw_likert(x, type = 7, breaks = NULL, N = length(x),
  link = "identity")
```

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.
link	link function between the latent variable and the probability of a postiive outcome, e.g. "logit", "probit", or "identity". For the "identity" link, the latent variable must be a probability.
x	for ‘draw_ordered‘ or ‘draw_likert‘, the latent data for each observation.
breaks	vector of breaks to cut a latent outcome into ordered categories with ‘draw_ordered‘ or ‘draw_likert‘
break_labels	vector of labels for the breaks to cut a latent outcome into ordered categories with ‘draw_ordered‘.
mean	for ‘draw_count‘, the mean number of count units for each observation
type	Type of Likert scale data for ‘draw_likert‘. Valid options are 4, 5, and 7. Type corresponds to the number of categories in the Likert scale.

Details

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

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(prob = 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")))

# Likert data: often used for survey data
fabricate(N = 10,
          support_free_college = draw_likert(x = rnorm(N),
                                             type = 5))

# 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_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, 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.
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)

# 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
```

---

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` *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(..., 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.

`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` without replacement. Users can also specify more complex resampling strategies to resample hierarchical data.

Usage

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

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).

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))
```

Index