# Package ‘simstudy’

January 20, 2022

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Simulation of Study Data</td>
</tr>
<tr>
<td>Version</td>
<td>0.4.0</td>
</tr>
<tr>
<td>Date</td>
<td>2022-01-18</td>
</tr>
</tbody>
</table>

**Description**  
Simulates data sets in order to explore modeling techniques or better understand data generating processes. The user specifies a set of relationships between covariates, and generates data based on these specifications. The final data sets can represent data from randomized control trials, repeated measure (longitudinal) designs, and cluster randomized trials. Missingness can be generated using various mechanisms (MCAR, MAR, NMAR).

**License**  
GPL-3

**URL**  
https://github.com/kgoldfeld/simstudy,
https://kgoldfeld.github.io/simstudy/,
https://kgoldfeld.github.io/simstudy/dev/

**BugReports**  
https://github.com/kgoldfeld/simstudy/issues

**Depends**  
R (>= 3.3.0)

**Imports**  
data.table, glue, methods, mvnfast, mvtnorm, Rcpp, backports

**Suggests**  
covr, dplyr, formatR, gee, ggplot2, grid, gridExtra, hedgehog, knitr, magrittr, Matrix, mgcv, ordinal, pracma, markdown, scales, splines, survival, testthat, gtsummary

**LinkingTo**  
Rcpp

**VignetteBuilder**  
knitr

**Encoding**  
UTF-8

**RoxygenNote**  
7.1.2

**NeedsCompilation**  
yes

**Author**  
Keith Goldfeld [aut, cre] (<https://orcid.org/0000-0002-0292-8780>), Jacob Wujciak-Jens [aut] (<https://orcid.org/0000-0002-7281-3989>)

**Maintainer**  
Keith Goldfeld <keith.goldfeld@nyulangone.org>

**Repository**  
CRAN

**Date/Publication**  
2022-01-20 16:00:02 UTC
R topics documented:

- `addColumns` ................................................. 3
- `addCondition` ............................................. 4
- `addCorData` ............................................... 5
- `addCorFlex` .............................................. 6
- `addCorGen` ............................................... 8
- `addMarkov` ............................................... 12
- `addMultiFac` ............................................. 13
- `addPeriods` .............................................. 14
- `betaGetShapes` ......................................... 16
- `defCondition` ........................................... 17
- `defData` ................................................ 18
- `defDataAdd` ............................................ 20
- `defMiss` ............................................... 21
- `defRead` ................................................ 22
- `defReadAdd` ........................................... 23
- `defReadCond` ........................................... 24
- `defRepeat` .............................................. 25
- `defRepeatAdd` ......................................... 27
- `defSurv` ............................................... 28
- `delColumns` ............................................. 29
- `distributions` .......................................... 30
- `gammaGetShapeRate` .................................... 31
- `genCatFormula` ......................................... 32
- `genCluster` ............................................ 33
- `genCorData` ........................................... 34
- `genCorFlex` ............................................ 35
- `genCorGen` ............................................ 36
- `genCorMat` ............................................. 38
- `genData` ............................................... 39
- `genDummy` ............................................. 40
- `genFactor` ............................................. 41
- `genFormula` ........................................... 42
- `genMarkov` ............................................ 43
- `genMiss` ............................................... 44
- `genMixFormula` ......................................... 45
- `genMultiFac` ........................................... 46
- `genNthEvent` .......................................... 47
- `genObs` ............................................... 48
- `genOrdCat` ............................................ 49
- `genSpline` ............................................. 52
- `genSurv` ............................................... 53
- `iccRE` ................................................ 54
- `mergeData` ............................................. 55
- `negbinomGetSizeProb` ................................... 56
- `simstudy-deprecated` ................................... 57
- `trimData` .............................................. 57
addColumns

Description

Add columns to existing data set

Usage

addColumns(dtDefs, dtOld, envir = parent.frame())

Arguments

dtDefs name of definitions for added columns
dtOld name of data table that is to be updated
envir Environment the data definitions are evaluated in. Defaults to base::parent.frame.

Value

an updated data.table that contains the added simulated data

Examples

# New data set

def <- defData(varname = "xNr", dist = "nonrandom", formula = 7, id = "idnum")
def <- defData(def, varname = "xUni", dist = "uniform", formula = "10;20")
dt <- genData(10, def)

# Add columns to dt

def2 <- defDataAdd(varname = "y1", formula = 10, variance = 3)
def2 <- defDataAdd(def2, varname = "y2", formula = .5, dist = "binary")
def2
dt <- addColumns(def2, dt)
dt
addCondition

Add a single column to existing data set based on a condition

Description
Add a single column to existing data set based on a condition

Usage
addCondition(condDefs, dtOld, newvar, envir = parent.frame())

Arguments
- condDefs: Name of definitions for added column
- dtOld: Name of data table that is to be updated
- newvar: Name of new column to add
- envir: Environment the data definitions are evaluated in. Defaults to base::parent.frame.

Value
An updated data.table that contains the added simulated data

Examples

# New data set
def <- defData(varname = "x", dist = "categorical", formula = ".33;.33")
def <- defData(def, varname = "y", dist = "uniform", formula = "-5;5")
dt <- genData(1000, def)

# Define conditions
defC <- defCondition(
  condition = "x == 1", formula = "5 + 2*y-.5*y^2",
  variance = 1, dist = "normal"
)
defC <- defCondition(defC,
  condition = "x == 2",
  formula = "3 - 3*y + y^2", variance = 2, dist = "normal"
)
defC <- defCondition(defC,
  condition = "x == 3",
  formula = "abs(y)", dist = "poisson"
)

# Add column
addCorData

```
dt <- addCondition(defC, dt, "NewVar")

# Plot data
library(ggplot2)

ggplot(data = dt, aes(x = y, y = NewVar, group = x)) +
  geom_point(aes(color = factor(x)))
```

---

**addCorData**

Add correlated data to existing data.table

**Description**

Add correlated data to existing data.table

**Usage**

```
addCorData(
  dtOld, idname, mu, sigma, corMatrix = NULL, rho, corstr = "ind", cnames = NULL
)
```

**Arguments**

- **dtOld**: Data table that is the new columns will be appended to.
- **idname**: Character name of id field, defaults to "id".
- **mu**: A vector of means. The length of mu must be nvars.
- **sigma**: Standard deviation of variables. If standard deviation differs for each variable, enter as a vector with the same length as the mean vector mu. If the standard deviation is constant across variables, a single value can be entered.
- **corMatrix**: Correlation matrix can be entered directly. It must be symmetrical and positive semi-definite. It is not a required field; if a matrix is not provided, then a structure and correlation coefficient rho must be specified.
- **rho**: Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
- **corstr**: Correlation structure of the variance-covariance matrix defined by sigma and rho. Options include "ind" for an independence structure, "cs" for a compound symmetry structure, and "ar1" for an autoregressive structure.
- **cnames**: Explicit column names. A single string with names separated by commas. If no string is provided, the default names will be V#, where # represents the column.
addCorFlex

Create multivariate (correlated) data - for general distributions

Value

The original data table with the additional correlated columns

Examples

def <- defData(varname = "xUni", dist = "uniform", formula = "10;20", id = "myID")
def <- defData(def, 
    varname = "xNorm", formula = "xUni * 2", dist = "normal", 
    variance = 8)
dt <- genData(250, def)
mu <- c(3, 8, 15)
sigma <- c(1, 2, 3)
dtAdd <- addCorData(dt, "myID", 
    mu = mu, sigma = sigma, 
    rho = .7, corstr = "cs"
) 
dtAdd

round(var(dtAdd[, .(V1, V2, V3)]), 3)
round(cor(dtAdd[, .(V1, V2, V3)]), 2)

dtAdd <- addCorData(dt, "myID", 
    mu = mu, sigma = sigma, 
    rho = .7, corstr = "ar1"
)
round(cor(dtAdd[, .(V1, V2, V3)]), 2)

corMat <- matrix(c(1, .2, .8, .2, 1, .6, .8, .6, 1), nrow = 3)
dtAdd <- addCorData(dt, "myID", 
    mu = mu, sigma = sigma, 
    corMatrix = corMat)
round(cor(dtAdd[, .(V1, V2, V3)]), 2)

addCorFlex

Create multivariate (correlated) data - for general distributions

Description

Create multivariate (correlated) data - for general distributions
addCorFlex

Usage

addCorFlex(
    dt,
    defs,
    rho = 0,
    tau = NULL,
    corstr = "cs",
    corMatrix = NULL,
    envir = parent.frame()
)

Arguments

dt     Data table that will be updated.
defs   Field definition table created by function defDataAdd.
rho    Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
tau    Correlation based on Kendall’s tau. If tau is specified, then it is used as the correlation even if rho is specified. If tau is NULL, then the specified value of rho is used, or rho defaults to 0.
corstr Correlation structure of the variance-covariance matrix defined by sigma and rho. Options include "cs" for a compound symmetry structure and "ar1" for an autoregressive structure. Defaults to "cs".
corMatrix Correlation matrix can be entered directly. It must be symmetrical and positive semi-definite. It is not a required field; if a matrix is not provided, then a structure and correlation coefficient rho must be specified.
envar Environment the data definitions are evaluated in. Defaults to base::parent.frame.

Value

data.table with added column(s) of correlated data

Examples

defC <- defData(
    varname = "nInds", formula = 50, dist = "noZeroPoisson",
    id = "idClust"
)

dc <- genData(10, defC)
### Normal only
dc <- addCorData(dc,
    mu = c(0, 0, 0, 0), sigma = c(2, 2, 2, 2), rho = .2,
    corstr = "cs", cnames = c("a", "b", "c", "d"),
    idname = "idClust"
)
di <- genCluster(dc, "idClust", "nInds", "id")
```r

defI <- defDataAdd(
  varname = "A", formula = "-1 + a", variance = 3,
  dist = "normal"
)
defI <- defDataAdd(defI,
  varname = "B", formula = "4.5 + b", variance = .5,
  dist = "normal"
)
defI <- defDataAdd(defI,
  varname = "C", formula = "5*c", variance = 3,
  dist = "normal"
)
defI <- defDataAdd(defI,
  varname = "D", formula = "1.6 + d", variance = 1,
  dist = "normal"
)

#### Generate new data
di <- addCorFlex(di, defI, rho = 0.4, corstr = "cs")

# Check correlations by cluster
for (i in 1:nrow(dc)) {
  print(cor(di[idClust == i, list(A, B, C, D)]))
}

# Check global correlations - should not be as correlated
cor(di[, list(A, B, C, D)])

---

**addCorGen**  
Create multivariate (correlated) data - for general distributions

**Description**  
Create multivariate (correlated) data - for general distributions

**Usage**  
addCorGen(
  dtOld,  
nvars,  
idvar = "id",  
rho,  
costr,  
corMatrix = NULL,  
dist,  
param1,
  ...)
```
```r
param2 = NULL,
cnames = NULL,
method = "copula",
formSpec = NULL,
periodvar = "period"
)

Arguments

dtOld          If an existing data.table is specified, then wide will be set to TRUE and n will
               be set to the nrow(dt) without any warning or error.
nvars          Number of new variables to create for each id.
idvar          String variable name of column represents individual level id for correlated data.
rho            Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
corstr         Correlation structure of the variance-covariance matrix defined by sigma and
               rho. Options include "cs" for a compound symmetry structure and "ar1" for an
               autoregressive structure.
corMatrix      Correlation matrix can be entered directly. It must be symmetrical and posi-
               tive semi-definite. It is not a required field; if a matrix is not provided, then a
               structure and correlation coefficient rho must be specified.
dist           A string indicating "normal", "binary", "poisson" or "gamma".
param1         A string that represents the column in dtOld that contains the parameter for the
               mean of the distribution. In the case of the uniform distribution the column
               specifies the minimum.
param2         A string that represents the column in dtOld that contains a possible second
               parameter for the distribution. For the normal distribution, this will be the vari-
               ance; for the gamma distribution, this will be the dispersion; and for the uniform
               distribution, this will be the maximum.
cnames         Explicit column names. A single string with names separated by commas. If no
               string is provided, the default names will be V#, where # represents the column.
method         Two methods are available to generate correlated data. (1) "copula" uses the
               multivariate Gaussian copula method that is applied to all other distributions;
               this applies to all available distributions. (2) "ep" uses an algorithm developed
formSpec       The formula (as a string) that was used to generate the binary outcome in the
               ‘defDataAdd’ statement. This is only necessary when method "ep" is requested.
periodvar      A string value that indicates the name of the field that indexes the repeated
               measurement for an individual unit. The value defaults to "period".

Value

Original data.table with added column(s) of correlated data

References

Emrich LJ, Piedmonte MR. A Method for Generating High-Dimensional Multivariate Binary Vari-
Examples

# Wide example

def <- defData(varname = "xbase", formula = 5, variance = .4, dist = "gamma", id = "cid")
def <- defData(def, varname = "lambda", formula = ".5 + .1*xbase", dist = "nonrandom", link = "log")
def <- defData(def, varname = "p", formula = "-2 + .3*xbase", dist = "nonrandom", link = "logit")
dt <- genData(500, def)
dtX1 <- addCorGen(dtOld = dt, idvar = "cid", nvars = 3, rho = .7, corstr = "cs", dist = "poisson", param1 = "lambda")
dtX2 <- addCorGen(dtOld = dt, idvar = "cid", nvars = 4, rho = .4, corstr = "ar1", dist = "binary", param1 = "p")

# Long example

def <- defData(varname = "xbase", formula = 5, variance = .4, dist = "gamma", id = "cid")
def <- defData(def, "nperiods", formula = 3, dist = "noZeroPoisson")
def2 <- defDataAdd(varname = "lambda", formula = ".5+.5*period + .1*xbase", dist = "nonrandom", link = "log")
def2 <- defDataAdd(def2, varname = "p", formula = "-3+.2*period + .3*xbase", dist = "nonrandom", link = "logit")
def2 <- defDataAdd(def2, varname = "gammaMu", formula = ".2*period + .3*xbase", dist = "nonrandom", link = "log")
def2 <- defDataAdd(def2, varname = "gammaDis", formula = 1, dist = "nonrandom")
def2 <- defDataAdd(def2, varname = "normMu", formula = "5+period + .5*xbase", dist = "nonrandom")
def2 <- defDataAdd(def2, varname = "normVar", formula = 4, dist = "nonrandom")
def2 <- defDataAdd(def2, varname = "unifMin", formula = "5 + 2*period + .2*xbase", dist = "nonrandom")
```r
def2 <- defDataAdd(def2,
  varname = "unifMax",
  formula = "unifMin + 20", dist = "nonrandom"
)
dt <- genData(1000, def)
dtLong <- addPeriods(dt, idvars = "cid", nPeriods = 3)
dtLong <- addColumns(def2, dtLong)

# Poisson distribution
dtX3 <- addCorGen(dtOld = dtLong, idvar = "cid", nvars = 3, rho = .6, corstr = "cs",
  dist = "poisson", param1 = "lambda", cnames = "NewPois"
)
dtX3

# Binomial distribution - copula method
dtX4 <- addCorGen(dtOld = dtLong, idvar = "cid", nvars = 3, rho = .6, corstr = "cs",
  dist = "binary", param1 = "p", cnames = "NewBin"
)
dtX4

# Gamma distribution
dtX6 <- addCorGen(dtOld = dtLong, idvar = "cid", nvars = 3, rho = .6, corstr = "ar1",
  dist = "gamma", param1 = "gammaMu", param2 = "gammaDis",
  cnames = "NewGamma"
)
dtX6

# Normal distribution
dtX7 <- addCorGen(dtOld = dtLong, idvar = "cid", nvars = 3, rho = .6, corstr = "ar1",
  dist = "normal", param1 = "normMu", param2 = "normVar",
  cnames = "NewNorm"
)

# Binary outcome - ep method
probform <- "-2 + .3*period"
def1 <- defDataAdd(
  varname = "p", formula = probform,
  dist = "nonrandom", link = "logit"
)
```
dx <- genData(100)
dx <- addPeriods(dx, nPeriods = 4)
dx <- addColumns(def1, dx)

dg <- addCorGen(dx,
  nvars = 4,
  corMatrix = NULL, rho = .3, corstr = "cs",
  dist = "binary", param = "p",
  method = "ep", formSpec = probform,
  periodvar = "period"
)

addMarkov

Add Markov chain

Description
Generate a Markov chain for n individuals or units by specifying a transition matrix.

Usage
addMarkov(
  dd, 
  transMat, 
  chainLen, 
  wide = FALSE, 
  id = "id", 
  pername = "period", 
  varname = "state", 
  widePrefix = "S", 
  start0lab = NULL, 
  trimvalue = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd</td>
<td>data.table with a unique identifier</td>
</tr>
<tr>
<td>transMat</td>
<td>Square transition matrix where the sum of each row must equal 1. The dimensions of the matrix equal the number of possible states.</td>
</tr>
<tr>
<td>chainLen</td>
<td>Length of each chain that will be generated for each chain; minimum chain length is 2.</td>
</tr>
<tr>
<td>wide</td>
<td>Logical variable (TRUE or FALSE) indicating whether the resulting data table should be returned in wide or long format. The wide format includes all elements of a chain on a single row; the long format includes each element of a chain in its own row. The default is wide = FALSE, so the long format is returned by default.</td>
</tr>
</tbody>
</table>
id  Character string that represents name of "id" field. Defaults to "id".

pername  Character string that represents the variable name of the chain sequence in the long format. Defaults "period".

varname  Character string that represents the variable name of the state in the long format. Defaults to "state".

widePrefix  Character string that represents the variable name prefix for the state fields in the wide format. Defaults to "S".

start0lab  Character string that represents name of the integer field containing starting state (State 0) of the chain for each individual. If it is NULL, starting state defaults to 1. Default is NULL.

trimvalue  Integer value indicating end state. If trimvalue is not NULL, all records after the first instance of state = trimvalue will be deleted.

Value

A data table with n rows if in wide format, or n by chainLen rows if in long format.

Examples

def1 <- defData(varname = "x1", formula = 0, variance = 1)
def1 <- defData(def1, varname = "x2", formula = 0, variance = 1)
def1 <- defData(def1,
    varname = "S0", formula = ".6;.3;.1",
    dist = "categorical"
)

dd <- genData(20, def1)

# Transition matrix P

P <- t(matrix(c(
    0.7, 0.2, 0.1,
    0.5, 0.3, 0.2,
    0.0, 0.7, 0.3
),
    nrow = 3
))

d1 <- addMarkov(dd, P, chainLen = 3)
d2 <- addMarkov(dd, P, chainLen = 5, wide = TRUE)
d3 <- addMarkov(dd, P, chainLen = 5, wide = TRUE, start0lab = "S0")
d4 <- addMarkov(dd, P, chainLen = 5, start0lab = "S0", trimvalue = 3)
**Description**

Add multi-factorial data

**Usage**

```r
addMultiFac(dtOld, nFactors, levels = 2, coding = "dummy", colNames = NULL)
```

**Arguments**

- `dtOld` data.table that is to be modified
- `nFactors` Number of factors (columns) to generate.
- `levels` Vector or scalar. If a vector is specified, it must be the same length as `nFactors`. Each value of the vector represents the number of levels of each corresponding factor. If a scalar is specified, each factor will have the same number of levels. The default is 2 levels for each factor.
- `coding` String value to specify if "dummy" or "effect" coding is used. Defaults to "dummy".
- `colNames` A vector of strings, with a length of `nFactors`. The strings represent the name for each factor.

**Value**

A data.table that contains the added simulated data. Each new column contains an integer.

**Examples**

```r
defD <- defData(varname = "x", formula = 0, variance = 1)

DT <- genData(360, defD)
DT <- addMultiFac(DT, nFactors = 3, levels = c(2, 3, 3), colNames = c("A", "B", "C"))
DT
DT[, .N, keyby =.(A, B, C)]

DT <- genData(300, defD)
DT <- addMultiFac(DT, nFactors = 3, levels = 2)
DT[, .N, keyby =.(Var1, Var2, Var3)]
```

---

**addPeriods**

Create longitudinal/panel data

**Description**

Create longitudinal/panel data
**Usage**

```r
addPeriods(
  dtName,
  nPeriods = NULL,
  idvars = "id",
  timevars = NULL,
  timevarName = "timevar",
  timeid = "timeID",
  perName = "period"
)
```

**Arguments**

- `dtName`: Name of existing data table
- `nPeriods`: Number of time periods for each record
- `idvars`: Names of index variables (in a string vector) that will be repeated during each time period
- `timevars`: Names of time dependent variables. Defaults to NULL.
- `timevarName`: Name of new time dependent variable
- `timeid`: Variable name for new index field. Defaults to "timevar"
- `perName`: Variable name for period field. Defaults to "period"

**Value**

An updated data.table that has multiple rows per observation in `dtName`

**Examples**

```r
tdef <- defData(varname = "T", dist = "binary", formula = 0.5)
tdef <- defData(tdef, varname = "Y0", dist = "normal", formula = 10, variance = 1)
tdef <- defData(tdef, varname = "Y1", dist = "normal", formula = "Y0 + 5 + 5 * T", variance = 1)
tdef <- defData(tdef, varname = "Y2", dist = "normal", formula = "Y0 + 10 + 5 * T", variance = 1)

dtTrial <- genData(5, tdef)
dtTrial

dtTime <- addPeriods(dtTrial,
  nPeriods = 3, idvars = "id",
  timevars = c("Y0", "Y1", "Y2"), timevarName = "Y"
)
dtTime

# Varying # of periods and intervals - need to have variables
# called nCount and mInterval

def <- defData(varname = "xbase", dist = "normal", formula = 20, variance = 3)
def <- defData(def, varname = "nCount", dist = "noZeroPoisson", formula = 6)
def <- defData(def, varname = "mInterval", dist = "gamma", formula = 30, variance = .01)
```
def <- defData(def, varname = "vInterval", dist = "nonrandom", formula = .07)

dt <- genData(200, def)
dt[id %in% c(8, 121)]

dtPeriod <- addPeriods(dt)
dtPeriod[id %in% c(8, 121)] # View individuals 8 and 121 only

---

**betaGetShapes**

*Convert beta mean and precision parameters to two shape parameters*

**Description**

Convert beta mean and precision parameters to two shape parameters

**Usage**

```
betaGetShapes(mean, precision)
```

**Arguments**

- **mean**: The mean of a beta distribution
- **precision**: The precision parameter (phi) of a beta distribution

**Details**

In simstudy, users specify the beta distribution as a function of two parameters - a mean and precision, where $0 < \text{mean} < 1$ and precision $> 0$. In this case, the variance of the specified distribution is $(\text{mean})*(1-\text{mean})/(1+\text{precision})$. The base R function rbeta uses the two shape parameters to specify the beta distribution. This function converts the mean and precision into the shape1 and shape2 parameters.

**Value**

A list that includes the shape parameters of the beta distribution

**Examples**

```
set.seed(12345)
mean <- 0.3
precision <- 1.6
rs <- betaGetShapes(mean, precision)
c(rs$shape1, rs$shape2)
vec <- rbeta(1000, shape1 = rs$shape1, shape2 = rs$shape2)
(estMoments <- c(mean(vec), var(vec)))
(throryMoments <- c(mean, mean * (1 - mean) / (1 + precision)))
(throryMoments <- with(rs, c(
    shape1 / (shape1 + shape2),
    (shape1 * shape2) / ((shape1 + shape2)^2 * (1 + shape1 + shape2))
)))
```
defCondition

Add single row to definitions table of conditions that will be used to add data to an existing definitions table

Description

Add single row to definitions table of conditions that will be used to add data to an existing definitions table

Usage

defCondition(
  dtDefs = NULL,
  condition,
  formula,
  variance = 0,
  dist = "normal",
  link = "identity"
)

Arguments

dtDefs Name of definition table to be modified. Null if this is a new definition.
condition Formula specifying condition to be checked
formula An R expression for mean (string)
variance Number
dist Distribution. For possibilities, see details
link The link function for the mean, see details

Value

A data.table named dtName that is an updated data definitions table

See Also
distributions

Examples

# New data set

def <- defData(varname = "x", dist = "noZeroPoisson", formula = 5)
def <- defData(def, varname = "y", dist = "normal", formula = 0, variance = 9)
dt <- genData(10, def)

# Add columns to dt
```r
defC <- defCondition(
  condition = "x == 1", formula = "5 + 2*y",
  variance = 1, dist = "normal"
)
defC <- defCondition(defC,
  condition = "x <= 5 & x >= 2", formula = "3 - 2*y",
  variance = 1, dist = "normal"
)
defC <- defCondition(defC,
  condition = "x >= 6", formula = 1,
  variance = 1, dist = "normal"
)
defC

# Add conditional column with field name "z"

dt <- addCondition(defC, dt, "z")
dt
```

---

**defData**  
*Add single row to definitions table*

**Description**  
Add single row to definitions table

**Usage**
```r
defData(
  dtDefs = NULL,
  varname,
  formula,
  variance = 0,
  dist = "normal",
  link = "identity",
  id = "id"
)
```

**Arguments**
- `dtDefs` Definition data.table to be modified
- `varname` Name (string) of new variable
- `formula` An R expression for mean (string)
- `variance` Number
**defData**

- **dist**  
  Distribution. For possibilities, see details
- **link**  
  The link function for the mean, see details
- **id**  
  A string indicating the field name for the unique record identifier

**Details**

The possible data distributions are: normal, binary, binomial, poisson, noZeroPoisson, uniform, categorical, gamma, beta, nonrandom, uniformInt, negBinomial, exponential, mixture, trtAssign.

**Value**

A data.table named dtName that is an updated data definitions table

**See Also**

distributions

**Examples**

```r
extVar <- 2.3
def <- defData(varname = "xNr", dist = "nonrandom", formula = 7, id = "idnum")
def <- defData(def, varname = "xUni", dist = "uniform", formula = "10;20")
def <- defData(def,
  varname = "xNorm", formula = "xNr + xUni * 2", dist = "normal",
  variance = 8)
def <- defData(def,
  varname = "xPois", dist = "poisson", formula = "xNr - 0.2 * xUni",
  link = "log")
def <- defData(def, varname = "xCat", formula = "0.3;0.2;0.5", dist = "categorical")
def <- defData(def,
  varname = "xGamma", dist = "gamma", formula = "5+xCat",
  variance = 1, link = "log")
def <- defData(def,
  varname = "xBin", dist = "binary", formula = "-3 + xCat",
  link = "logit")
def <- defData(def,
  varname = "external", dist = "nonrandom",
  formula = "xBin * log(.extVar)"
)
def
```
defDataAdd  

Add single row to definitions table that will be used to add data to an existing data.table

Description
Add single row to definitions table that will be used to add data to an existing data.table

Usage
defDataAdd(
  dtDefs = NULL,
  varname,
  formula,
  variance = 0,
  dist = "normal",
  link = "identity"
)

Arguments
- dtDefs: Name of definition table to be modified. Null if this is a new definition.
- varname: Name (string) of new variable
- formula: An R expression for mean (string)
- variance: Number
- dist: Distribution. For possibilities, see details
- link: The link function for the mean, see details

Value
A data.table named dtName that is an updated data definitions table

See Also
[distributions]

Examples
# New data set

def <- defData(varname = "xNr", dist = "nonrandom", formula = 7, id = "idnum")
def <- defData(def, varname = "xUni", dist = "uniform", formula = "10;20")
dt <- genData(10, def)

# Add columns to dt
defMiss

```r
def2 <- defDataAdd(varname = "y1", formula = 10, variance = 3)
def2 <- defDataAdd(def2, varname = "y2", formula = .5, dist = "binary")
def2
dt <- addColumns(def2, dt)
dt
```

defMiss

*Definitions for missing data*

**Description**

Add single row to definitions table for missing data

**Usage**

```r
defMiss(
  dtDefs = NULL,
  varname,
  formula,
  logit.link = FALSE,
  baseline = FALSE,
  monotonic = FALSE
)
```

**Arguments**

- `dtDefs`: Definition data.table to be modified
- `varname`: Name of variable with missingness
- `formula`: Formula to describe pattern of missingness
- `logit.link`: Indicator set to TRUE when the probability of missingness is based on a logit model.
- `baseline`: Indicator is set to TRUE if the variable is a baseline measure and should be missing throughout an entire observation period. This is applicable to repeated measures/longitudinal data.
- `monotonic`: Indicator set to TRUE if missingness at time t is followed by missingness at all follow-up times > t.

**Value**

A data.table named dtName that is an updated data definitions table

**See Also**

`genMiss, genObs`
Examples

def1 <- defData(varname = "m", dist = "binary", formula = .5)
def1 <- defData(def1, "u", dist = "binary", formula = .5)
def1 <- defData(def1, "x1", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x2", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x3", dist = "normal", formula = "20*m + 20*u", variance = 2)

dtAct <- genData(1000, def1)
defM <- defMiss(varname = "x1", formula = .15, logit.link = FALSE)
defM <- defMiss(defM, varname = "x2", formula = "0.05 + m * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "x3", formula = "0.05 + u * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "u", formula = 1, logit.link = FALSE) # not observed
defM

# Generate missing data matrix

missMat <- genMiss(dtName = dtAct, missDefs = defM, idvars = "id")

# Generate observed data from actual data and missing data matrix

dtObs <- genObs(dtAct, missMat, idvars = "id")
dtObs

defRead

---

**Read external csv data set definitions**

---

Description

Read external csv data set definitions

Usage

defRead(filen, id = "id")

Arguments

- **filen**: String file name, including full path. Must be a csv file.
- **id**: string that includes name of id field. Defaults to "id"

Value

A data.table with data set definitions

See Also

[distributions]
Examples

```r
# Create temporary external "csv" file

test1 <- c(
  "varname,formula,variance,dist,link",
  "nr,7, 0,nonrandom,identity",
  "x1,.4, 0,binary,identity",
  "y1,nr + x1 * 2,8,normal,identity",
  "y2,nr - 0.2 * x1,0,poisson, log"
)

tfcsv <- tempfile()
writeLines(test1, tfcsv)

# Read external csv file stored in file "tfcsv"

defs <- defRead(tfcsv, id = "myID")
defs

unlink(tfcsv)

# Generate data based on external definition

genData(5, defs)
```

defReadAdd

Read external csv data set definitions for adding columns

Description

Read external csv data set definitions for adding columns

Usage

```r
defReadAdd(filen)
```

Arguments

- `filen` : String file name, including full path. Must be a csv file.

Value

A data.table with data set definitions

See Also

[distributions]
Examples

# Create temporary external "csv" files

test1 <- c(
  "varname,formula,variance,dist,link",
  "nr,7,0,nonrandom,identity"
)

tcsv1 <- tempfile()
writeLines(test1, tcsv1)

test2 <- c(
  "varname,formula,variance,dist,link",
  "x1,.4,0,binary,identity",
  "y1,nr + x1 * 2.8,normal,identity",
  "y2,nr - 0.2 * x1,0,poisson,log"
)

tcsv2 <- tempfile()
writeLines(test2, tcsv2)

# Generate data based on external definitions

defs <- defRead(tcsv1)
dt <- genData(5, defs)
dt

# Add additional data based on external definitions

defs2 <- defReadAdd(tcsv2)
dt <- addColumns(defs2, dt)
dt

unlink(tcsv1)
unlink(tcsv2)

---

defReadCond

Read external csv data set definitions for adding columns

Description

Read external csv data set definitions for adding columns

Usage

defReadCond(filen)

Arguments

filen String file name, including full path. Must be a csv file.
**Value**

A data.table with data set definitions

**See Also**

[distributions]

**Examples**

```r
# Create temporary external "csv" files
test1 <- c(
  "varname,formula,variance,dist,link",
  "x,0.3;0.4;0.3,0,categorical,identity"
)
tfcsv1 <- tempfile()
writeLines(test1, tfcsv1)

test2 <- c(
  "condition,formula,variance,dist,link",
  "x == 1, 0.4,0,binary,identity",
  "x == 2, 0.6,0,binary,identity",
  "x >= 3, 0.8,0,binary,identity"
)
tfcsv2 <- tempfile()
writeLines(test2, tfcsv2)

# Generate data based on external definitions
defs <- defRead(tfcsv1)
dt <- genData(2000, defs)
dt

defsCond <- defReadCond(tfcsv2)
dt <- addCondition(defsCond, dt, "y")
dt

dt[, mean(y), keyby = x]

unlink(tfcsv1)
unlink(tfcsv2)
```

**defRepeat**

Add multiple (similar) rows to definitions table
Description

Add multiple (similar) rows to definitions table

Usage

defRepeat(
  dtDefs = NULL,
  nVars,
  prefix,
  formula,
  variance = 0,
  dist = "normal",
  link = "identity",
  id = "id"
)

Arguments

dtDefs Definition data.table to be modified
nVars Number of new variables to define
prefix Prefix (character) for new variables
formula An R expression for mean (string)
variance Number or formula
dist Distribution. For possibilities, see details
link The link function for the mean, see details
id A string indicating the field name for the unique record identifier

Details

The possible data distributions are: 'r paste0(.getDists(),collapse = ", ")'.

Value

A data.table named dtName that is an updated data definitions table

See Also

[distributions]

Examples

def <- defRepeat(
  nVars = 4, prefix = "g", formula = "1/3;1/3;1/3",
  variance = 0, dist = "categorical"
)
def <- defData(def, varname = "a", formula = "1;1", dist = "trtAssign")
def <- defRepeat(def, 8, "b", formula = "5 + a", variance = 3, dist = "normal")
def <- defData(def, "y", formula = "0.10", dist = "binary")
defRepeatAdd

Add multiple (similar) rows to definitions table that will be used to add data to an existing data.table

Description
Add multiple (similar) rows to definitions table that will be used to add data to an existing data.table

Usage

defRepeatAdd(
  dtDefs = NULL,
  nVars,
  prefix,
  formula,
  variance = 0,
  dist = "normal",
  link = "identity",
  id = "id"
)

Arguments

dtDefs Definition data.table to be modified
nVars Number of new variables to define
prefix Prefix (character) for new variables
formula An R expression for mean (string)
variance Number or formula
dist Distribution. For possibilities, see details
link The link function for the mean, see details
id A string indicating the field name for the unique record identifier

Details
The possible data distributions are: ‘r paste0(.getDists(),collapse = ", ")’.

Value
A data.table named dtName that is an updated data definitions table

See Also
[distributions]
Examples

```r
def <- defRepeatAdd(
  nVars = 4, prefix = "g", formula = "1/3;1/3;1/3",
  variance = 0, dist = "categorical"
)
def <- defDataAdd(def, varname = "a", formula = "1;1", dist = "trtAssign")
def <- defRepeatAdd(def, 8, "b", formula = "5 + a", variance = 3, dist = "normal")
def <- defDataAdd(def, "y", formula = "0.10", dist = "binary")
def
```

---

**defSurv**  
*Add single row to survival definitions*

**Description**

Add single row to survival definitions

**Usage**

```r
defSurv(dtDefs = NULL, varname, formula = 0, scale, shape = 1)
```

**Arguments**

- `dtDefs`: Definition data.table to be modified  
- `varname`: Variable name  
- `formula`: Covariates predicting survival  
- `scale`: Scale parameter for the Weibull distribution.  
- `shape`: The shape of the Weibull distribution. Shape = 1 for an exponential distribution

**Value**

A data.table named dtName that is an updated data definitions table

**Examples**

```r
# Baseline data definitions
def <- defData(varname = "x1", formula = .5, dist = "binary")
def <- defData(def, varname = "x2", formula = .5, dist = "binary")
def <- defData(def, varname = "grp", formula = .5, dist = "binary")

# Survival data definitions
sdef <- defSurv(
  varname = "survTime", formula = "1.5*x1",
  scale = "grp*50 + (1-grp)*25", shape = "grp*1 + (1-grp)*1.5"
)
```
sdef <- defSurv(sdef, varname = "censorTime", scale = 80, shape = 1)

sdef

# Baseline data definitions
dtSurv <- genData(300, def)

# Add survival times
dtSurv <- genSurv(dtSurv, sdef)

head(dtSurv)

delColumns

Delete columns from existing data set

Description
Delete columns from existing data set

Usage
delColumns(dtOld, vars)

Arguments
dtOld Name of data table that is to be updated.
vars Vector of column names (as strings).

Value
An updated data.table without vars.

Examples
# New data set
def <- defData(varname = "x", dist = "noZeroPoisson", formula = 7, id = "idnum")
def <- defData(def, varname = "xUni", dist = "uniformInt", formula = "x-3;x+3")
dt <- genData(10, def)
dt

# Delete column
dt <- delColumns(dt, "x")
dt
Distributions

This help file describes the distributions used for data creation in simstudy.

Arguments

- **formula**: Desired mean as a Number or an R expression for mean as a String. Variables defined via `defData()` and variables within the parent environment (prefixed with `..`) can be used within the formula. Functions from the parent environment can be used without a prefix.

- **variance**: Number. Default is 0.

- **link**: String identifying the link function to be used. Default is `identity`.

Details

For details about the statistical distributions please see `stats::distributions`, any non-statistical distributions will be explained below. Required variables and expected pattern for each distribution can be found in this table:

<table>
<thead>
<tr>
<th>name</th>
<th>formula</th>
<th>format</th>
<th>variance</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta</td>
<td>mean</td>
<td>String or Number</td>
<td>dispersion value</td>
<td>identity or log</td>
</tr>
<tr>
<td>binary</td>
<td>probability for 1</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>binomial</td>
<td>probability of success</td>
<td>String or Number</td>
<td>number of trials</td>
<td>identity or log</td>
</tr>
<tr>
<td>categorical</td>
<td>probabilities</td>
<td>p_1;p_2;...;p_n</td>
<td>category labels: a;b;c , 50;130;20</td>
<td>identity or log</td>
</tr>
<tr>
<td>exponential</td>
<td>mean (lambda)</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>gamma</td>
<td>mean</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>mixture</td>
<td>formula</td>
<td>x_1</td>
<td>p_1 + x_2</td>
<td>p_2 ... x_n</td>
</tr>
<tr>
<td>negBinomial</td>
<td>mean</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>nonrandom</td>
<td>formula</td>
<td>String or Number</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>normal</td>
<td>mean</td>
<td>String or Number</td>
<td>variance</td>
<td>NA</td>
</tr>
<tr>
<td>noZeroPoisson</td>
<td>mean</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>poisson</td>
<td>mean</td>
<td>String or Number</td>
<td>NA</td>
<td>identity or log</td>
</tr>
<tr>
<td>trtAssign</td>
<td>ratio</td>
<td>r_1;r_2;...;r_n</td>
<td>stratification</td>
<td>NA</td>
</tr>
<tr>
<td>uniform</td>
<td>range</td>
<td>from:to</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>uniformInt</td>
<td>range</td>
<td>from:to</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Mixture

The mixture distribution makes it possible to mix to previously defined distributions/variables. Each variable that should be part of the new distribution x_1,...,X_n is assigned a probability p_1,...,p_n. For more information see rdatagen.net.
gammaGetShapeRate

Examples

ext_var <- 2.9
def <- defData(varname = "external", formula = "3 + log(.ext_var)", variance = .5)
def
genData(5, def)

---

gammaGetShapeRate  Convert gamma mean and dispersion parameters to shape and rate parameters

Description

Convert gamma mean and dispersion parameters to shape and rate parameters

Usage

gammaGetShapeRate(mean, dispersion)

Arguments

mean  The mean of a gamma distribution
dispersion  The dispersion parameter of a gamma distribution

Details

In simstudy, users specify the gamma distribution as a function of two parameters - a mean and dispersion. In this case, the variance of the specified distribution is (mean^2)*dispersion. The base R function rgamma uses the shape and rate parameters to specify the gamma distribution. This function converts the mean and dispersion into the shape and rate.

Value

A list that includes the shape and rate parameters of the gamma distribution

Examples

set.seed(12345)
mean <- 5
dispersion <- 1.5
rs <- gammaGetShapeRate(mean, dispersion)
c(rs$shape, rs$rate)
vec <- rgamma(1000, shape = rs$shape, rate = rs$rate)
(estMoments <- c(mean(vec), var(vec)))
(theoryMoments <- c(mean, mean^2 * dispersion))
(theoryMoments <- c(rs$shape / rs$rate, rs$shape / rs$rate^2))
genCatFormula  

Generate Categorical Formula

Description

Create a semi-colon delimited string of probabilities to be used to define categorical data.

Usage

\[
\text{genCatFormula}(\ldots, n = 0)
\]

Arguments

\[
\ldots \quad \text{one or more numeric values to be concatenated, delimited by ";".}
\]
\[
n \quad \text{Number of probabilities (categories) to be generated - all with equal probability.}
\]

Details

The function accepts a number of probabilities or a value of \( n \), but not both.

If probabilities are passed, the string that is returned depends on the nature of those probabilities. If the sum of the probabilities is less than 1, an additional category is created with the probability \( 1 - \text{sum(provided probabilities)} \). If the sum of the probabilities is equal to 1, then the number of categories is set to the number of probabilities provided. If the sum of the probabilities exceeds one (and there is more than one probability), the probabilities are standardized by dividing by the sum of the probabilities provided.

If \( n \) is provided, \( n \) probabilities are included in the string, each with a probability equal to \( 1/n \).

Value

String with multinomial probabilities.

Examples

\[
\text{genCatFormula}(0.25, 0.25, 0.50)
\]
\[
\text{genCatFormula}(1/3, 1/2)
\]
\[
\text{genCatFormula}(1, 2, 3)
\]
\[
\text{genCatFormula}(n = 5)
\]
Simulate clustered data

Description

Simulate data set that is one level down in a multilevel data context. The level "2" data set must contain a field that specifies the number of individual records in a particular cluster.

Usage

genCluster(dtClust, cLevelVar, numIndsVar, level1ID, allLevel2 = TRUE)

Arguments

dtClust Name of existing data set that contains the level "2" data
cLevelVar Variable name (string) of cluster id in dtClust
numIndsVar Variable name (string) of number of observations per cluster in dtClust. Can also be a single integer value that will be used for all clusters.
level1ID Name of id field in new level "1" data set
allLevel2 Indicator: if set to TRUE (default), the returned data set includes all of the Level 2 data columns. If FALSE, the returned data set only includes the Levels 1 and 2 ids.

Value

A simulated data table with level "1" data

Examples

gen.school <- defData(
  varname = "s0", dist = "normal",
  formula = 0, variance = 3, id = "idSchool"
)

# add nClasses

gen.school <- defData(gen.school,
  varname = "nClasses",
  dist = "noZeroPoisson", formula = 3)

dtSchool <- genData(3, gen.school) 

# genCluster

dtClass <- genCluster(dtSchool, 
  cLevelVar = "idSchool", 
  numIndsVar = "nClasses", level1ID = "idClass"
)

dtClass <- genCluster(dtSchool,
cLevelVar = "idSchool",
numIndsVar = 3, level1ID = "idClass"
}
dtClass

---

**genCorData**

Create correlated data

**Description**

Create correlated data

**Usage**

```r
genCorData(
  n,
  mu,
  sigma,
  corMatrix = NULL,
  rho,
  corstr = "ind",
  cnames = NULL,
  idname = "id"
)
```

**Arguments**

- `n` Number of observations
- `mu` A vector of means. The length of mu must be nvars.
- `sigma` Standard deviation of variables. If standard deviation differs for each variable, enter as a vector with the same length as the mean vector mu. If the standard deviation is constant across variables, as single value can be entered.
- `corMatrix` Correlation matrix can be entered directly. It must be symmetrical and positive semi-definite. It is not a required field; if a matrix is not provided, then a structure and correlation coefficient rho must be specified.
- `rho` Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
- `corstr` Correlation structure of the variance-covariance matrix defined by sigma and rho. Options include "ind" for an independence structure, "cs" for a compound symmetry structure, and "ar1" for an autoregressive structure.
- `cnames` Explicit column names. A single string with names separated by commas. If no string is provided, the default names will be V#, where # represents the column.
- `idname` The name of the index id name. Defaults to "id."

**Value**

A data.table with n rows and the k + 1 columns, where k is the number of means in the vector mu.
Examples

```r
mu <- c(3, 8, 15)
sigma <- c(1, 2, 3)
corMat <- matrix(c(1, .2, .8, .2, 1, .6, .8, .6, 1), nrow = 3)
dtcor1 <- genCorData(1000, mu = mu, sigma = sigma, rho = .7, corstr = "cs")
dtcor2 <- genCorData(1000, mu = mu, sigma = sigma, corMatrix = corMat)
dtcor1
dtcor2
round(var(dtcor1[, .(V1, V2, V3)]), 3)
round(cor(dtcor1[, .(V1, V2, V3)]), 2)
round(var(dtcor2[, .(V1, V2, V3)]), 3)
round(cor(dtcor2[, .(V1, V2, V3)]), 2)
```

---

**genCorFlex**

Create multivariate (correlated) data - for general distributions

**Usage**

```r
genCorFlex(n, defs, rho = 0, tau = NULL, corstr = "cs", corMatrix = NULL)
```

**Arguments**

- `n`: Number of observations
- `defs`: Field definition table created by function `defData`. All definitions must be scalar. Definition specifies distribution, mean, and variance, with all caveats for each of the distributions. (See `defData`).
- `rho`: Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
- `tau`: Correlation based on Kendall's tau. If tau is specified, then it is used as the correlation even if rho is specified. If tau is NULL, then the specified value of rho is used, or rho defaults to 0.
- `corstr`: Correlation structure of the variance-covariance matrix defined by sigma and rho. Options include "cs" for a compound symmetry structure and "ar1" for an autoregressive structure. Defaults to "cs".
- `corMatrix`: Correlation matrix can be entered directly. It must be symmetrical and positive semi-definite. It is not a required field; if a matrix is not provided, then a structure and correlation coefficient rho must be specified. This is only used if tau is not specified.
Value
data.table with added column(s) of correlated data

Examples
def <- defData(varname = "xNorm", formula = 0, variance = 4, dist = "normal")
def <- defData(def, varname = "xGamma1", formula = 15, variance = 2, dist = "gamma")
def <- defData(def, varname = "xBin", formula = 0.5, dist = "binary")
def <- defData(def, varname = "xUnif1", formula = "0;10", dist = "uniform")
def <- defData(def, varname = "xPois", formula = 15, dist = "poisson")
def <- defData(def, varname = "xUnif2", formula = "23;28", dist = "uniform")
def <- defData(def, varname = "xUnif3", formula = "100;150", dist = "uniform")
def <- defData(def, varname = "xGamma2", formula = 150, variance = 0.003, dist = "gamma")
def <- defData(def, varname = "xNegBin", formula = 5, variance = .8, dist = "negBinomial")

dt <- genCorFlex(1000, def, tau = 0.3, corstr = "cs")
cor(dt[, -"id"])
cor(dt[, -"id"], method = "kendall")
var(dt[, -"id"])
apply(dt[, -"id"], 2, mean)
Arguments

n  Number of observations
nvars  Number of variables
params1  A single vector specifying the mean of the distribution. The vector is of length 1
if the mean is the same across all observations, otherwise the vector is of length
nvars. In the case of the uniform distribution the vector specifies the minimum.

params2  A single vector specifying a possible second parameter for the distribution. For
the normal distribution, this will be the variance; for the gamma distribution,
this will be the dispersion; and for the uniform distribution, this will be the max-
imum. The vector is of length 1 if the mean is the same across all observations,
otherwise the vector is of length nvars.

dist  A string indicating "binary", "poisson" or "gamma", "normal", or "uniform".
rho  Correlation coefficient, -1 <= rho <= 1. Use if corMatrix is not provided.
corstr  Correlation structure of the variance-covariance matrix defined by sigma and
rho. Options include "cs" for a compound symmetry structure and "ar1" for an
autoregressive structure.
corMatrix  Correlation matrix can be entered directly. It must be symmetrical and posi-
tive semi-definite. It is not a required field; if a matrix is not provided, then a
structure and correlation coefficient rho must be specified.
wide  The layout of the returned file - if wide = TRUE, all new correlated variables
will be returned in a single record, if wide = FALSE, each new variable will be
its own record (i.e. the data will be in long form). Defaults to FALSE.
cnames  Explicit column names. A single string with names separated by commas. If no
string is provided, the default names will be V#, where # represents the column.
method  Two methods are available to generate correlated data. (1) "copula" uses the
multivariate Gaussian copula method that is applied to all other distributions;
this applies to all available distributions. (2) "ep" uses an algorithm developed

idname  Character value that specifies the name of the id variable.

Value

data.table with added column(s) of correlated data

References

Emrich LJ, Piedmonte MR. A Method for Generating High-Dimensional Multivariate Binary Vari-

Examples

```r
set.seed(23432)
1 <- c(8, 10, 12)
genCorGen(1000, nvars = 3, params1 = 1, dist = "poisson", rho = .7, corstr = "cs")
genCorGen(1000, nvars = 3, params1 = 5, dist = "poisson", rho = .7, corstr = "cs")
```
genCorGen(1000, nvars = 3, params1 = 1, dist = "poisson", rho = .7, corstr = "cs", wide = TRUE)
genCorGen(1000, nvars = 3, params1 = 5, dist = "poisson", rho = .7, corstr = "cs", wide = TRUE)

genCorGen(1000, nvars = 3, params1 = 1, dist = "poisson", rho = .7, corstr = "cs",
cnames = "new_var"
)
genCorGen(1000, nvars = 3, params1 = 1, dist = "poisson", rho = .7, corstr = "cs",
wide = TRUE, cnames = "a, b, c"
)
genCorGen(1000, nvars = 3, params1 = c(.3, .5, .7), dist = "binary", rho = .3, corstr = "cs")
genCorGen(1000, nvars = 3, params1 = c(1, 1, 1), dist = "gamma", rho = .3,
corstr = "cs", wide = TRUE
)
genCorGen(1000, nvars = 3, params1 = c(.3, .5, .7), dist = "binary",
corMatrix = genCorMat(3), method = "ep"
)
genCorGen(1000, nvars = 3, params1 = c(.3, .5, .7), dist = "binary",
corMatrix = genCorMat(3), method = "copula"
)

---

**genCorMat**

Create a correlation matrix

---

**Description**

Create a correlation matrix

**Usage**

`genCorMat(nvars, cors = NULL)`

**Arguments**

- `nvars` number of rows and columns (i.e. number of variables) for correlation matrix
- `cors` vector of correlations.

**Details**

If the vector `cors` is not specified, a random correlation matrix is generated with no assumptions. If the vector is provided, it should be interpreted as the lower triangle of the correlation matrix, and is specified by reading down the columns. For example, if CM is the correlation matrix and `nvars = 3`, then `CM[2,1] = cors[1]`, `CM[3,1] = cors[2]`, and `CM[3,2] = cors[3]`. 
Value

correlation matrix of size nvars x nvars

Examples

genCorMat(3, c(.3, -.2, .1))
genCorMat(3)

genCorMat(5, c(.3, -.2, .1, .2, .5, .2, -.1, .3, .1, .2))
genCorMat(5)

Description

Calling function to simulate data

Usage

genData(n, dtDefs = NULL, id = "id", envir = parent.frame())

Arguments

n
the number of observations required in the data set.
dtDefs
name of definitions data.table/data.frame. If no definitions are provided a data set with ids only is generated.
id
The string defining the id of the record. Will override previously set id name with a warning (unless the old value is 'id'). If the id attribute in dtDefs is NULL will default to 'id'.
envir
Environment the data definitions are evaluated in. Defaults to base::parent.frame.

Value

A data.table that contains the simulated data.

Examples

genData(5)
genData(5, id = "grpID")

def <- defData(
    varname = "xNr", dist = "nonrandom", formula = 7,
    id = "idnum"
)
def <- defData(def,
    varname = "xUni", dist = "uniform",
    formula = "10;20"
)
genDummy <- genDummy(def,
  varname = "xNorm", formula = "xNr + xUni * 2",
  dist = "normal", variance = 8)
)
  def <- defData(def,
        varname = "xPois", dist = "poisson",
  formula = "xNr - 0.2 * xUni", link = "log"
  )
  def <- defData(def,
        varname = "xCat", formula = "0.3;0.2;0.5",
  dist = "categorical"
  )
  def <- defData(def,
        varname = "xGamma", dist = "gamma", formula = "5+xCat",
  variance = 1, link = "log"
  )
  def <- defData(def,
        varname = "xBin", dist = "binary", formula = "-3 + xCat",
  link = "logit"
  )
  def

  genData(5, def)

---

genDummy Create dummy variables from a factor or integer variable

**Description**

Create dummy variables from a factor or integer variable

**Usage**

```
  genDummy(dtName, varname, sep = ".", replace = FALSE)
```

**Arguments**

- **dtName**: Data table with column
- **varname**: Name of factor
- **sep**: Character to be used in creating new name for dummy fields. Valid characters include all letters and ".". Will default to ".". If an invalid character is provided, it will be replaced by default.
- **replace**: If replace is set to TRUE (defaults to FALSE) the field referenced varname will be removed.
Examples

# First example:

def <- defData(varname = "cat", formula = ".2;.3;.5", dist = "categorical")
def <- defData(def, varname = "x", formula = 5, variance = 2)
dx <- genData(200, def)
dx

dx <- genFactor(dx, "cat", labels = c("one", "two", "three"), replace = TRUE)
dx <- genDummy(dx, varname = "fcat", sep = "_")
dx

# Second example:
dx <- genData(15)
dx <- trtAssign(dtName = dx, 3, grpName = "arm")
dx <- genDummy(dx, varname = "arm")
dx

---

**genFactor**

Create factor variable from an existing (non-double) variable

---

**Description**

Create factor variable from an existing (non-double) variable

**Usage**

genFactor(dtName, varname, labels = NULL, prefix = "f", replace = FALSE)

**Arguments**

dtName Data table with columns.

varname Name of field(s) to be converted.

labels Factor level labels. If not provided, the generated factor levels will be used as the labels. Can be a vector (if only one new factor or all factors have the same labels) or a list of character vectors of the same length as varname.

prefix By default, the new field name will be a concatenation of "f" and the old field name. A prefix string can be provided.

replace If replace is set to TRUE (defaults to FALSE) the field referenced varname will be removed.
Examples

# First example:

```r
def <- defData(varname = "cat", formula = ".2;.3;.5", dist = "categorical")
def <- defData(def, varname = "x", formula = 5, variance = 2)

dx <- genData(200, def)
dx

dx <- genFactor(dx, "cat", labels = c("one", "two", "three"))
dx
```

# Second example:

```r
dx <- genData(10)
dx <- trtAssign(dtName = dx, 2, grpName = "studyArm")
dx <- genFactor(dx, varname = "studyArm", labels = c("control", "treatment"), prefix = "t_")
dx
```

---

**genFormula**

 Generate a linear formula

Description

Formulas for additive linear models can be generated with specified coefficient values and variable names.

Usage

```r
genFormula(coefs, vars)
```

Arguments

- **coefs**: A numerical vector that contains the values of the coefficients. If `length(coefs) == length(vars)`, then no intercept is assumed. Otherwise, an intercept is assumed.
- **vars**: A vector of strings that specify the names of the explanatory variables in the equation.

Value

A string that represents the desired formula
genMarkov

Examples

genFormula(c(.5, 2, 4), c("A", "B", "C"))
genFormula(c(.5, 2, 4), c("A", "B"))

changeX <- c(7, 10)
genFormula(c(.5, 2, changeX[1]), c("A", "B"))
genFormula(c(.5, 2, changeX[2]), c("A", "B"))
genFormula(c(.5, 2, changeX[2]), c("A", "B", "C"))

ewForm <- genFormula(c(-2, 1), c("A"))
def1 <- defData(varname = "A", formula = 0, variance = 3, dist = "normal")
def1 <- defData(def1, varname = "B", formula = newForm, dist = "binary", link = "logit")

set.seed(2001)
dt <- genData(500, def1)
summary(glm(B ~ A, data = dt, family = binomial))

---

**genMarkov**

*Generate Markov chain*

Description

Generate a Markov chain for \( n \) individuals or units by specifying a transition matrix.

Usage

```r
genMarkov(
  n,
  transMat,
  chainLen,
  wide = FALSE,
  id = "id",
  pername = "period",
  varname = "state",
  widePrefix = "S",
  trimvalue = NULL
)
```

Arguments

- \( n \) : number of individual chains to generate
- \( \text{transMat} \) : Square transition matrix where the sum of each row must equal 1. The dimensions of the matrix equal the number of possible states.
- \( \text{chainLen} \) : Length of each chain that will be generated for each chain; minimum chain length is 2.
Logical variable (TRUE or FALSE) indicating whether the resulting data table should be returned in wide or long format. The wide format includes all elements of a chain on a single row; the long format includes each element of a chain in its own row. The default is wide = FALSE, so the long format is returned by default.

Character string that represents name of "id" field. Defaults to "id".

Character string that represents the variable name of the chain sequence in the long format. Defaults to "period".

Character string that represents the variable name of the state in the long format. Defaults to "state".

Character string that represents the variable name prefix for the state fields in the wide format. Defaults to "S".

Integer value indicating end state. If trimvalue is not NULL, all records after the first instance of state = trimvalue will be deleted.

A data table with n rows if in wide format, or n by chainLen rows if in long format.

# Transition matrix P

```r
P <- t(matrix(c(0.7, 0.2, 0.1,
                 0.5, 0.3, 0.2,
                 0.0, 0.1, 0.9),
                 nrow = 3, ncol = 3))
```

d1 <- genMarkov(n = 10, transMat = P, chainLen = 5)
d2 <- genMarkov(n = 10, transMat = P, chainLen = 5, wide = TRUE)
d3 <- genMarkov(n = 10, transMat = P, chainLen = 5,
                pername = "seq", varname = "health",
                trimvalue = 3)

---

### genMiss

**Generate missing data**

**Description**

Generate missing data

**Usage**

```r
genMiss(dtName, missDefs, idvars, repeated = FALSE, periodvar = "period")
```
**Arguments**

- `dtName`: Name of complete data set
- `missDefs`: Definitions of missingness
- `idvars`: Index variables
- `repeated`: Indicator for longitudinal data
- `periodvar`: Name of variable that contains period

**Value**

Missing data matrix indexed by idvars (and period if relevant)

**See Also**

`defMiss, genObs`

**Examples**

```r
def1 <- defData(varname = "m", dist = "binary", formula = .5)
def1 <- defData(def1, "u", dist = "binary", formula = .5)
def1 <- defData(def1, "x1", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x2", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x3", dist = "normal", formula = "20*m + 20*u", variance = 2)
dtAct <- genData(1000, def1)

defM <- defMiss(varname = "x1", formula = .15, logit.link = FALSE)
defM <- defMiss(defM, varname = "x2", formula = ".05 + m * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "x3", formula = ".05 + u * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "u", formula = 1, logit.link = FALSE) # not observed
defM

# Generate missing data matrix

missMat <- genMiss(dtAct, defM, idvars = "id")
missMat

# Generate observed data from actual data and missing data matrix

dtObs <- genObs(dtAct, missMat, idvars = "id")
dtObs
```

---

**Description**

Generates a mixture formula from a vector of variable names and an optional vector of probabilities.
Usage

genMixFormula(vars, probs = NULL, varLength = NULL)

Arguments

vars Character vector/list of variable names.
probs Numeric vector/list of probabilities. Has to be same length as vars or NULL. Probabilities will be normalized if the sum to > 1.
varLength If vars is of length one and varLength is set to any integer > 0, vars will be interpreted as array of length varLength and all elements will used in sequence.

Value

The mixture formula as a string.

Examples

genMixFormula(c("a", "b[.i]", "c"))
genMixFormula(c("a", "b", "c"), c(.2, .5, .3))

# Shorthand to use external vectors/lists
genMixFormula(".arr", varLength = 5)

genMultiFac Generate multi-factorial data

description

Generate multi-factorial data

Usage

genMultiFac(
    nFactors,
    each,
    levels = 2,
    coding = "dummy",
    colNames = NULL,
    idName = "id"
)
**genNthEvent**

Generate event data using longitudinal data, and restrict output to time until the nth event.

**Description**

Generate event data using longitudinal data, and restrict output to time until the nth event.

**Usage**

```r
genNthEvent(dtName, defEvent, nEvents = 1, perName = "period", id = "id")
```

**Arguments**

- `dtName`: name of existing data table
- `defEvent`: data definition table (created with `defDataAdd`) that determines the event generating process.
- `nEvents`: maximum number of events that will be generated (the nth event).
- `perName`: variable name for period field. Defaults to "period".
- `id`: string representing name of the id field in table specified by `dtName`
Value

data.table that stops after "nEvents" are reached.

Examples

defD <- defData(
  varname = "effect", formula = 0, variance = 1,
  dist = "normal"
)
defE <- defDataAdd(
  varname = "died", formula = "-2.5 + 0.3*period + effect",
  dist = "binary", link = "logit"
)
d <- genData(1000, defD)
d <- addPeriods(d, 10)
dx <- genNthEvent(d, defEvent = defE, nEvents = 3)

---

genObs Create an observed data set that includes missing data

Description

Create an observed data set that includes missing data

Usage

genObs(dtName, dtMiss, idvars)

Arguments

dtName Name of complete data set
dtMiss Name of missing data matrix
idvars Index variables that cannot be missing

Value

A data table that represents observed data, including missing data

See Also

defMiss, genMiss
Examples

def1 <- defData(varname = "m", dist = "binary", formula = .5)
def1 <- defData(def1, "u", dist = "binary", formula = .5)
def1 <- defData(def1, "x1", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x2", dist = "normal", formula = "20*m + 20*u", variance = 2)
def1 <- defData(def1, "x3", dist = "normal", formula = "20*m + 20*u", variance = 2)
dtAct <- genData(1000, def1)

defM <- defMiss(varname = "x1", formula = .15, logit.link = FALSE)
defM <- defMiss(defM, varname = "x2", formula = "0.05 + m * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "x3", formula = "0.05 + u * 0.25", logit.link = FALSE)
defM <- defMiss(defM, varname = "u", formula = 1, logit.link = FALSE) # not observed
defM

# Generate missing data matrix
missMat <- genMiss(dtAct, defM, idvars = "id")
missMat

# Generate observed data from actual data and missing data matrix

dtObs <- genObs(dtAct, missMat, idvars = "id")
dtObs

---

**genOrdCat**

Generate ordinal categorical data

**Description**

Ordinal categorical data is added to an existing data set. Correlations can be added via correlation matrix or rho and corstr.

**Usage**

genOrdCat(
  dtName,
  adjVar = NULL,
  baseprobs,
  catVar = "cat",
  asFactor = TRUE,
  idname = "id",
  prefix = "grp",
  rho = 0,
  corstr = "ind",
  corMatrix = NULL,
  npVar = NULL,
  npAdj = NULL
)
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtName</td>
<td>Name of complete data set</td>
</tr>
<tr>
<td>adjVar</td>
<td>Adjustment variable name in dtName - determines logistic shift. This is specified assuming a cumulative logit link.</td>
</tr>
<tr>
<td>baseprobs</td>
<td>Baseline probability expressed as a vector or matrix of probabilities. The values (per row) must sum to &lt;= 1. If ( \text{rowSums(baseprobs)} &lt; 1 ), an additional category is added with probability ( 1 - \text{rowSums(baseprobs)} ). The number of rows represents the number of new categorical variables. The number of columns represents the number of possible responses - if an particular category has fewer possible responses, assign zero probability to non-relevant columns.</td>
</tr>
<tr>
<td>catVar</td>
<td>Name of the new categorical field. Defaults to &quot;cat&quot;. Can be a character vector with a name for each new variable defined via baseprobs. Will be overridden by prefix if more than one variable is defined and ( \text{length(catVar)} == 1 ).</td>
</tr>
<tr>
<td>asFactor</td>
<td>If ( \text{asFactor} == \text{TRUE} ) (default), new field is returned as a factor. If ( \text{asFactor} == \text{FALSE} ), new field is returned as an integer.</td>
</tr>
<tr>
<td>idname</td>
<td>Name of the id column in dtName.</td>
</tr>
<tr>
<td>prefix</td>
<td>A string. The names of the new variables will be a concatenation of the prefix and a sequence of integers indicating the variable number.</td>
</tr>
<tr>
<td>rho</td>
<td>Correlation coefficient, (-1 &lt; \rho &lt; 1). Use if corMatrix is not provided.</td>
</tr>
<tr>
<td>corstr</td>
<td>Correlation structure of the variance-covariance matrix defined by sigma and rho. Options include &quot;ind&quot; for an independence structure, &quot;cs&quot; for a compound symmetry structure, and &quot;ar1&quot; for an autoregressive structure.</td>
</tr>
<tr>
<td>corMatrix</td>
<td>Correlation matrix can be entered directly. It must be symmetrical and positive definite. It is not a required field; if a matrix is not provided, then a structure and correlation coefficient rho must be specified. (The matrix created via rho and corstr must also be positive definite.)</td>
</tr>
<tr>
<td>npVar</td>
<td>Vector of variable names that indicate which variables are to violate the proportionality assumption.</td>
</tr>
<tr>
<td>npAdj</td>
<td>Matrix with a row for each npVar and a column for each category. Each value represents the deviation from the proportional odds assumption on the logistic scale.</td>
</tr>
</tbody>
</table>

### Value

Original data.table with added categorical field.

### Examples

```r
# Ordinal Categorical Data ----
def1 <- defData(
  varname = "male",
  formula = 0.45, dist = "binary", id = "idG"
)
def1 <- defData(def1,
  varname = "z",
  asFactor = TRUE, corMatrix = corMat2,
  corstr = "cs", rho = 0.5)
```
genOrdCat

formula = "1.2*male", dist = "nonrandom"
)
def1

## Generate data

set.seed(20)

dx <- genData(1000, def1)

probs <- c(0.40, 0.25, 0.15)

dx <- genOrdCat(dx, 
adjVar = "z", idname = "idG", baseprobs = probs, 
catVar = "grp"
)
dx

# Correlated Ordinal Categorical Data ----

baseprobs <- matrix(c(
  0.2, 0.1, 0.1, 0.6, 
  0.7, 0.2, 0.1, 0, 
  0.5, 0.2, 0.3, 0, 
  0.4, 0.2, 0.4, 0, 
  0.6, 0.2, 0.2, 0
), nrow = 5, byrow = TRUE
)

set.seed(333)

dT <- genData(1000)

dX <- genOrdCat(dT, 
adjVar = NULL, baseprobs = baseprobs, 
prefix = "q", rho = .125, corstr = "cs", asFactor = FALSE
)
dx

dM <- data.table::melt(dX, id.vars = "id")
dProp <- dM[, prop.table(table(value)), by = variable]
dProp[, response := c(1:4, 1:3, 1:3, 1:3, 1:3)]
data.table::dcast(dProp, variable ~ response, 
  value.var = "V1", fill = 0
)

# proportional odds assumption violated

d1 <- defData(varname = "rx", formula = "1;1", dist = "trtAssign")
d1 <- defData(d1, varname = "z", formula = "0 - 1.2*rx", dist = "nonrandom")

dd <- genData(1000, d1)
```r
dn <- genOrdCat(
  dtName = dd, adjVar = "z",
  baseprobs = baseprobs,
  npVar = "rx", npAdj = npAdj
)
```

---

**Description**

Generate spline curves

**Usage**

```r
genSpline(
  dt,
  newvar,
  predictor,
  theta,
  knots = c(0.25, 0.5, 0.75),
  degree = 3,
  newrange = NULL,
  noise.var = 0
)
```

**Arguments**

- `dt` data.table that will be modified
- `newvar` Name of new variable to be created
- `predictor` Name of field in old data.table that is predicting new value
- `theta` A vector or matrix of values between 0 and 1. Each column of the matrix represents the weights/coefficients that will be applied to the basis functions determined by the knots and degree. Each column of theta represents a separate spline curve.
- `knots` A vector of values between 0 and 1, specifying quantile cut-points for splines. Defaults to c(0.25, 0.50, 0.75).
- `degree` Integer specifying polynomial degree of curvature.
- `newrange` Range of the spline function, specified as a string with two values separated by a semi-colon. The first value represents the minimum, and the second value represents the maximum. Defaults to NULL, which sets the range to be between 0 and 1.
noise.var  Add to normally distributed noise to observation - where mean is value of spline curve.

Value

A modified data.table with an added column named newvar.

Examples

ddef <- defData(varname = "age", formula = "0;1", dist = "uniform")
theta1 <- c(0.1, 0.8, 0.6, 0.4, 0.6, 0.9, 0.9)
knots <- c(0.25, 0.5, 0.75)
viewSplines(knots = knots, theta = theta1, degree = 3)
set.seed(234)
dt <- genData(1000, ddef)
dt <- genSpline(
  dt = dt, newvar = "weight",
  predictor = "age", theta = theta1,
  knots = knots, degree = 3,
  noise.var = .025
)

dt

genSurv  Generate survival data

Description

Survival data is added to an existing data set.

Usage

genSurv(dtName, survDefs, digits = 3)

Arguments

dtName  Name of complete data set
survDefs  Definitions of survival
digits  Number of digits for rounding

Value

Original matrix with survival time
Examples

# Baseline data definitions

def <- defData(varname = "x1", formula = .5, dist = "binary")
def <- defData(def, varname = "x2", formula = .5, dist = "binary")
def <- defData(def, varname = "grp", formula = .5, dist = "binary")

# Survival data definitions

sdef <- defSurv(
  varname = "survTime", formula = "1.5*x1",
  scale = "grp*50 + (1-grp)*25", shape = "grp*1 + (1-grp)*1.5"
)
sdef <- defSurv(sdef, varname = "censorTime", scale = 80, shape = 1)
sdef

# Baseline data definitions
dtSurv <- genData(300, def)

# Add survival times
dtSurv <- genSurv(dtSurv, sdef)

head(dtSurv)

---

iccRE

Generate variance for random effects that produce desired intra-class coefficients (ICCs) for clustered data.

Description

Generate variance for random effects that produce desired intra-class coefficients (ICCs) for clustered data.

Usage

iccRE(ICC, dist, varTotal = NULL, varWithin = NULL, lambda = NULL, disp = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICC</td>
<td>Vector of values between 0 and 1 that represent the target ICC levels</td>
</tr>
<tr>
<td>dist</td>
<td>The distribution that describes the outcome data at the individual level. Possible distributions include &quot;normal&quot;, &quot;binary&quot;, &quot;poisson&quot;, or &quot;gamma&quot;</td>
</tr>
<tr>
<td>varTotal</td>
<td>Numeric value that represents the total variation for a normally distributed model. If &quot;normal&quot; distribution is specified, either varTotal or varWithin must be specified, but not both.</td>
</tr>
</tbody>
</table>
mergeData

varWithin Numeric value that represents the variation within a cluster for a normally distributed model. If "normal" distribution is specified, either varTotal or varWithin must be specified, but not both.

lambda Numeric value that represents the grand mean. Must be specified when distribution is "poisson" or "negative binomial".

disp Numeric value that represents the dispersion parameter that is used to define a gamma or negative binomial distribution with a log link. Must be specified when distribution is "gamma".

Value
A vector of values that represents the variances of random effects at the cluster level that correspond to the ICC vector.

Examples

targetICC <- seq(0.05, 0.20, by = .01)

iccRE(targetICC, "poisson", lambda = 30)

iccRE(targetICC, "binary")

iccRE(targetICC, "normal", varTotal = 100)

iccRE(targetICC, "normal", varWithin = 100)

iccRE(targetICC, "gamma", disp = .5)

iccRE(targetICC, "negBinomial", lambda = 40, disp = .5)

mergeData

Merge two data tables

Description
Merge two data tables

Usage
mergeData(dt1, dt2, idvars)

Arguments

dt1 Name of first data.table
dt2 Name of second data.table
idvars Vector of string names to merge on
A new data table that merges dt2 with dt1

Examples

```r
def1 <- defData(varname = "x", formula = 0, variance = 1)
def1 <- defData(varname = "xcat", formula = "0.3;0.2", dist = "categorical")
def2 <- defData(varname = "yBin", formula = 0.5, dist = "binary", id = "xcat")
def2 <- defData(def2, varname = "yNorm", formula = 5, variance = 2)
dt1 <- genData(20, def1)
dt2 <- genData(3, def2)
dtMerge <- mergeData(dt1, dt2, "xcat")
dtMerge
```

---

**negbinomGetSizeProb**

*Convert negative binomial mean and dispersion parameters to size and prob parameters*

**Description**

Convert negative binomial mean and dispersion parameters to size and prob parameters

**Usage**

```r
negbinomGetSizeProb(mean, dispersion)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>The mean of a gamma distribution</td>
</tr>
<tr>
<td>dispersion</td>
<td>The dispersion parameter of a gamma distribution</td>
</tr>
</tbody>
</table>

**Details**

In simstudy, users specify the negative binomial distribution as a function of two parameters - a mean and dispersion. In this case, the variance of the specified distribution is mean + (mean^2)*dispersion. The base R function rbinom uses the size and prob parameters to specify the negative binomial distribution. This function converts the mean and dispersion into the size and probability parameters.

**Value**

A list that includes the size and prob parameters of the neg binom distribution
Examples

```r
set.seed(12345)
mean <- 5
dispersion <- 0.5
sp <- negbinomGetSizeProb(mean, dispersion)
c(sp$size, sp$prob)
vec <- rnbinom(1000, size = sp$size, prob = sp$prob)
(estMoments <- c(mean(vec), var(vec)))
(theoryMoments <- c(mean, mean + mean^2 * dispersion))
(theoryMoments <- c(sp$size * (1 - sp$prob) / sp$prob, sp$size * (1 - sp$prob) / sp$prob^2))
```

---

**simstudy-deprecated**  
*Deprecated functions in simstudy*

**Description**

These functions are provided for compatibility with older versions of simstudy only, and will be defunct in the future.

**Details**

- **genCorOrdCat**: This function is deprecated, and will be removed in the future. Use `genOrdCat` with `asFactor = FALSE` instead.
- **catProbs**: This function is deprecated, and will be removed in the future. Use `genCatFormula` with the same functionality instead.

---

**trimData**  
*Trim longitudinal data file once an event has occurred*

**Description**

Trim longitudinal data file once an event has occurred

**Usage**

```r
trimData(dtOld, seqvar, eventvar, idvar = "id")
```

**Arguments**

- `dtOld`: name of data table to be trimmed
- `seqvar`: string referencing column that indexes the sequence or period
- `eventvar`: string referencing event data column
- `idvar`: string referencing id column
trtAssign

Value

an updated data.table removes all rows following the first event for each individual

Examples

eDef <- defDataAdd(varname = "e", formula = "u==4", dist = "nonrandom")

P <- t(matrix(c(
  0.4, 0.3, 0.2, 0.1,
  0.0, 0.4, 0.3, 0.3,
  0.0, 0.0, 0.5, 0.5,
  0.0, 0.0, 0.0, 1.0
),
nrow = 4
))

dp <- genMarkov(
  n = 100, transMat = P,
  chainLen = 8, id = "id",
  pername = "period",
  varname = "u"
)

dp <- addColumns(eDef, dp)

dp <- trimData(dp, seqvar = "period", eventvar = "e", idvar = "id")

dp

trtAssign Assign treatment

Description

Assign treatment

Usage

trtAssign(
  dtName,
  nTrt = 2,
  balanced = TRUE,
  strata = NULL,
  grpName = "trtGrp",
  ratio = NULL
)
**Arguments**

- `dtName`: data table
- `nTrt`: number of treatment groups
- `balanced`: indicator for treatment assignment process
- `strata`: vector of strings representing stratifying variables
- `grpName`: string representing variable name for treatment or exposure group
- `ratio`: vector of values indicating relative proportion of group assignment

**Value**

An integer (group) ranging from 1 to length of the probability vector

**See Also**

- `trtObserve`

**Examples**

```r
dt <- genData(15)

dt1 <- trtAssign(dt, nTrt = 3, balanced = TRUE)
dt1[, .N, keyby = trtGrp]

dt2 <- trtAssign(dt, nTrt = 3, balanced = FALSE)
dt2[, .N, keyby = trtGrp]

def <- defData(varname = "male", formula = .4, dist = "binary")
dt <- genData(1000, def)
dt
dt3 <- trtAssign(dt, nTrt = 5, strata = "male", balanced = TRUE, grpName = "Group")
dt3
dt3[, .N, keyby = .(male, Group)]
dt3[, .N, keyby = .(Group)]

dt4 <- trtAssign(dt, nTrt = 5, strata = "male", balanced = FALSE, grpName = "Group")
dt4[, .N, keyby = .(male, Group)]
dt4[, .N, keyby = .(Group)]

dt5 <- trtAssign(dt, nTrt = 5, balanced = TRUE, grpName = "Group")
dt5[, .N, keyby = .(male, Group)]
dt5[, .N, keyby = .(Group)]

dt6 <- trtAssign(dt, nTrt = 3, ratio = c(1, 2, 2), grpName = "Group")
dt6[, .N, keyby = .(Group)]
```
**trtObserve**  
*Observed exposure or treatment*

**Description**

Observed exposure or treatment

**Usage**

trtObserve(dt, formulas, logit.link = FALSE, grpName = "trtGrp")

**Arguments**

- **dt**: data table
- **formulas**: collection of formulas that determine probabilities
- **logit.link**: indicator that specifies link. If TRUE, then logit link is used. If FALSE, the identity link is used.
- **grpName**: character string representing name of treatment/exposure group variable

**Value**

An integer (group) ranging from 1 to length of the probability vector

**See Also**

trtAssign

**Examples**

def <- defData(varname = "male", dist = "binary", formula = .5, id = "cid")
def <- defData(def, varname = "over65", dist = "binary", formula = "-1.7 + .8*male", link = "logit")
def <- defData(def, varname = "baseDBP", dist = "normal", formula = 70, variance = 40)
dtstudy <- genData(1000, def)
dtstudy

formula1 <- c("-2 + 2*male - .5*over65", "-1 + 2*male + .5*over65")
dtObs <- trtObserve(dtstudy, formulas = formula1, logit.link = TRUE, grpName = "exposure")
dtObs

# Check actual distributions

dtObs[, .(pctMale = round(mean(male), 2)), keyby = exposure]
dtObs[, .(pctMale = round(mean(over65), 2)), keyby = exposure]
dtSum <- dtObs[, .N, keyby = .(male, over65, exposure)]
dtSum[, grpPct := round(N / sum(N), 2), keyby = .(male, over65)]
dtSum
trtStepWedge

Assign treatment for stepped-wedge design

Description

Assign treatment for stepped-wedge design

Usage

trtStepWedge(
  dtName,
  clustID,
  nWaves,
  lenWaves,
  startPer,
  perName = "period",
  grpName = "rx",
  lag = 0,
  xrName = "xr"
)

Arguments

dtName    data table
clustID   string representing name of column of cluster level ids
nWaves    number of treatment waves
lenWaves  the number of periods between waves
startPer  the starting period of the first wave
perName   string representing name of column of time periods
grpName   string representing variable name for treatment or exposure group
lag       integer representing length of transition period
xrName    string representing name of the field that indicates whether the cluster status is in transition status

Value

A data.table with the added treatment assignment

See Also

trtObserve  trtAssign
Examples

defc <- defData(
    varname = "ceffect", formula = 0, variance = 0.10,
    dist = "normal", id = "cluster"
)
defc <- defData(defc, "m", formula = 10, dist = "nonrandom")

# Will generate 3 waves of 4 clusters each - starting 2, 5, and 8
dc <- genData(12, defc)
dp <- addPeriods(dc, 12, "cluster")
dp <- trtStepWedge(dp, "cluster",
    nWaves = 3,
    lenWaves = 3, startPer = 2
)
dp

dp <- addPeriods(dc, 12, "cluster")
dp <- trtStepWedge(dp, "cluster",
    nWaves = 2,
    lenWaves = 1, startPer = 4, lag = 3
)
dp

updateDef

Update definition table

Description

Updates row definition table created by function defData or defRead. (For tables created using
defDataAdd and defReadAdd use updateDefAdd.) Does not modify in-place.

Usage

updateDef(
    dtDefs,
    changevar,
    newformula = NULL,
    newvariance = NULL,
    newdist = NULL,
    newlink = NULL,
    remove = FALSE
)

Arguments

dtDefs Definition table that will be modified
changevar Name of field definition that will be changed
newformula  New formula definition (defaults to NULL)
newvariance New variance specification (defaults to NULL)
newdist    New distribution definition (defaults to NULL)
newlink    New link specification (defaults to NULL)
remove     If set to TRUE, remove ‘changevar’ from definition (defaults to FALSE).

Value
The updated data definition table.

Examples

# Example 1

defs <- defData(varname = "x", formula = 0, variance = 3, dist = "normal")
defs <- defData(defs, varname = "y", formula = "2 + 3*x", variance = 1, dist = "normal")
defs <- defData(defs, varname = "z", formula = "4 + 3*x - 2*y", variance = 1, dist = "normal")
defs
updateDef(dtDefs = defs, changevar = "y", newformula = "x + 5", newvariance = 2)
updateDef(dtDefs = defs, changevar = "z", newdist = "poisson", newlink = "log")

# Example 2

defs <- defData(varname = "w", formula = 0, variance = 3, dist = "normal")
defs <- defData(defs, varname = "x", formula = "1 + w", variance = 1, dist = "normal")
defs <- defData(defs, varname = "z", formula = 4, variance = 1, dist = "normal")
defs
updateDef(dtDefs = defs, changevar = "x", remove = TRUE)
updateDef(dtDefs = defs, changevar = "z", remove = TRUE)

# No changes to original definition:
defs

updateDefAdd

Update definition table

Description

Updates row definition table created by functions defDataAdd and defReadAdd. (For tables created using defData or defRead use updateDef.)
Usage

updateDefAdd(
  dtDefs,
  changevar,
  newformula = NULL,
  newvariance = NULL,
  newdist = NULL,
  newlink = NULL,
  remove = FALSE
)

Arguments

dtDefs            Definition table that will be modified
changevar         Name of field definition that will be changed
newformula        New formula definition (defaults to NULL)
newvariance       New variance specification (defaults to NULL)
newdist           New distribution definition (defaults to NULL)
newlink           New link specification (defaults to NULL)
remove            If set to TRUE, remove definition (defaults to FALSE)

Value

A string that represents the desired formula

Examples

# Define original data
defs <- defData(varname = "w", formula = 0, variance = 3, dist = "normal")
defs <- defData(defs, varname = "x", formula = "1 + w", variance = 1, dist = "normal")
defs <- defData(defs, varname = "z", formula = 4, variance = 1, dist = "normal")

# Define additional columns
defsA <- defDataAdd(varname = "a", formula = "w + x + z", variance = 2, dist = "normal")

set.seed(2001)
dt <- genData(10, defs)
dt <- addColumns(defsA, dt)
dt

# Modify definition of additional column
defsA <- updateDefAdd(dtDefs = defsA, changevar = "a", newformula = "w+z", newvariance = 1)

set.seed(2001)
dt <- genData(10, defs)
viewBasis

Plot basis spline functions

Description

Plot basis spline functions

Usage

viewBasis(knots, degree)

Arguments

knots A vector of values between 0 and 1, specifying cut-points for splines
degree Integer specifying degree of curvature.

Value

A ggplot object that contains a plot of the basis functions. In total, there will be length(knots) + 
degree + 1 functions plotted.

Examples

knots <- c(0.25, 0.50, 0.75)
viewBasis(knots, degree = 1)

knots <- c(0.25, 0.50, 0.75)
viewBasis(knots, degree = 2)

knots <- c(0.25, 0.50, 0.75)
viewBasis(knots, degree = 3)

viewSplines

Plot spline curves

Description

Plot spline curves

Usage

viewSplines(knots, degree, theta)
viewSplines

Arguments

knots A vector of values between 0 and 1, specifying cut-points for splines
degree Integer specifying degree of curvature.
theta A vector or matrix of values between 0 and 1. Each column of the matrix represents the weights/coefficients that will be applied to the basis functions determined by the knots and degree. Each column of theta represents a separate spline curve.

Value

A ggplot object that contains a plot of the spline curves. The number of spline curves in the plot will equal the number of columns in the matrix (or it will equal 1 if theta is a vector).

Examples

knots <- c(0.25, 0.5, 0.75)
theta1 <- c(0.1, 0.8, 0.4, 0.9, 0.2, 1.0)

viewSplines(knots, degree = 2, theta1)

theta2 <- matrix(c(
  0.1, 0.2, 0.4, 0.9, 0.2, 0.3,
  0.1, 0.3, 0.8, 1.0, 0.9,
  0.1, 0.4, 0.3, 0.8, 0.7, 0.5,
  0.1, 0.9, 0.8, 0.2, 0.1, 0.6
),
  ncol = 4
)

viewSplines(knots, degree = 2, theta2)
Index

* categorical
  genOrdCat, 49
* condition
  addCondition, 4
defCondition, 17
defRead, 22
* correlated
  addCorData, 5
  addCorFlex, 6
  addCorGen, 8
genCorData, 34
genCorFlex, 35
genCorGen, 36
genCorMat, 38
genOrdCat, 49
* define_data
  defCondition, 17
defData, 18
defDataAdd, 20
defRead, 22
defReadAdd, 23
defReadCond, 24
defRepeat, 25
defRepeatAdd, 27
defSurv, 28
updateDef, 62
updateDefAdd, 63
* generate_data
  addColumns, 3
  addCondition, 4
  addMarkov, 12
  addMultiFac, 13
genData, 39
genDummy, 40
genFactor, 41
genFormula, 42
genMarkov, 43
genMultiFac, 46
genOrdCat, 49
genSpline, 52
genSurv, 53
* group_data
  addPeriods, 14
genCluster, 33
genNthEvent, 47
trtAssign, 58
trtObserve, 60
trtStepWedge, 61
* missing
  defMiss, 21
genMiss, 44
genObs, 48
* splines
  genSpline, 52
  viewBasis, 65
  viewSplines, 65
* utility
  betaGetShapes, 16
delColumns, 29
gammaGetShapeRate, 31
genCatFormula, 32
genMixFormula, 45
iccRE, 54
mergeData, 55
negbinomGetSizeProb, 56
trimData, 57
updateDef, 62
updateDefAdd, 63
viewBasis, 65
viewSplines, 65

addColumns, 3
addCondition, 4
addCorData, 5
addCorFlex, 6
addCorGen, 8
addMarkov, 12
addMultiFac, 13
addPeriods, 14
base::parent.frame, 3, 4, 7, 39
beta(distributions), 30
betaGetShapes, 16
binary (distributions), 30
binomial (distributions), 30
categorical (distributions), 30
catProbs, 57
defCondition, 17
defData, 18
defData(), 30
defDataAdd, 20
defMiss, 21, 45, 48
defRead, 22
defReadAdd, 23
defReadCond, 24
defRepeat, 25
defRepeatAdd, 27
defSurv, 28
delColumns, 29
distributions, 17, 19, 30
exponential (distributions), 30
gamma (distributions), 30
gammaGetShapeRate, 31
genCatFormula, 32, 57
genCluster, 33
genCorData, 34
genCorFlex, 35
genCorGen, 36
genCorMat, 38
genCorOrdCat, 57
genData, 39
genDummy, 40
genFactor, 41
genFormula, 42
genMarkov, 43
genMiss, 21, 44, 48
genMixFormula, 45
genMultiFac, 46
genNthEvent, 47
genObs, 21, 45, 48
genOrdCat, 49, 57
genSpline, 52
genSurv, 53
iccRE, 54
mergeData, 55
mixture (distributions), 30
negbinomGetSizeProb, 56
negBinomial (distributions), 30
nonrandom (distributions), 30
normal (distributions), 30
noZeroPoisson (distributions), 30
poisson (distributions), 30
simstudy-deprecated, 57
stats::distributions, 30
trimData, 57
trtAssign, 58, 60, 61
trtObserve, 59, 60, 61
trtStepWedge, 61
uniform (distributions), 30
updateDef, 62
updateDefAdd, 63
viewBasis, 65
viewSplines, 65