Package: stevemisc (via r-universe)

November 23, 2026

Type Package

Title Steve's Miscellaneous Functions

Version 2.0.0

Depends R (>= 3.6.0), stats

Description These are miscellaneous functions that I find useful for my research and teaching. The contents include themes for plots, functions for simulating quantities of interest from regression models, functions for simulating various forms of fake data for instructional/research purposes, and many more. All told, the functions provided here are broadly useful for data organization, data presentation, data recoding, and data simulation.

License GPL (>= 2)

BugReports https://github.com/svmiller/stevemisc/issues

Encoding UTF-8

LazyData true

Imports ggplot2 (>= 3.3.0), magrittr, labelled, arm, parallel, purrr, tibble, dplyr, methods, lme4, rlang, forcats, stringr, httr, rmarkdown, tidyr

Suggests knitr, DBI, RSQLite, dbplyr, lifecycle

RoxygenNote 7.3.3

NeedsCompilation no

Author Steve Miller [aut, cre], Ben Bolker [ctb], Dave Armstrong [ctb], John Fox [ctb], Winston Chang [ctb], Brian Ripley [ctb], Bill Venables [ctb], Pascal van Kooten [ctb], Gerko Vink [ctb], Paul Williamson [ctb], Andreas Beger [ctb] (ORCID: <https://orcid.org/0000-0003-1883-3169>), Vincent Arel-Bundock [ctb] (ORCID: <https://orcid.org/0000-0003-2042-7063>), Grant McDermott [ctb] (ORCID: <https://orcid.org/0000-0001-7883-8573>), Hadley Wickham [ctb] (ORCID: <https://orcid.org/0000-0003-4757-117X>), Teucher Andy [ctb], Miller Jason [ctb]

Maintainer Steve Miller <steve@svmiller.com>

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

Date/Publication 2026-11-23 15:00:41 UTC

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

RemoteRef HEAD

RemoteSha e1c9fe90d2e272a1ccf37a3feb4733cc116b9119

Contents

%nin% . . . 3
binred_plot . . . 4
carrec . . . 5
center_at . . . 6
charitable_contributions . . . 9
cor2data . . . 10
corvectors . . . 11
db_lselect . . . 12
ess9_labelled . . . 14
fct_reorg . . . 15
filter_refs . . . 15
fr_plot . . . 16
fra_leaderyears . . . 17
get_sims . . . 18
get_var_info . . . 20
gmy_dyadyears . . . 21
jenny . . . 22
make_perclab . . . 22
make_scale . . . 23
map_quiz . . . 24
mround . . . 24
msl.brms . . . 25
n2w . . . 27
normal_dist . . . 28
p_z . . . 29
prepare_refs . . . 30
print_refs . . . 31
ps_btscs . . . 32
ps_spells . . . 33
r1sd . . . 34
r2sd . . . 35
rbnorm . . . 36
rd_plot . . . 37
revcode . . . 38
sbayesboot . . . 39
sbtscs . . . 40
show_ranef . . . 41

smvrnorm 42
stevepubs 43
strategic_rivalries 44
studentt 45
tbl_df 46
theme_steve_web 46
usa_mids 48
wls 49
wom 50

Index **51**

`%nin%` *Find Non-Matching Elements*

Description

`%nin%` finds non-matching elements in a given vector. It is the negation of `%in%`.

Usage

```
a %nin% b
```

Arguments

a	a vector (character, factor, or numeric)
b	a vector (character, factor, or numeric)

Details

This is a simple negation of `%in%`. I use it mostly for columns in a data frame.

Value

`%nin%` finds non-matching elements and returns one of two things, depending on the use. For two simple vectors, it will report what matches and what does not. For comparing a vector within a data frame, it has the effect of reporting the rows in the data frame that do not match the supplied (second) vector.

Examples

```
library(tibble)
library(dplyr)

# Watch this subset stuff

dat <- tibble(x = seq(1:10), d = rnorm(10))
filter(dat, x %nin% c(3, 6, 9))
```

`binred_plot` *Generate a Binned-Residual Plot from a Fitted Generalized Linear Model*

Description

`binred_plot()` provides a diagnostic of the fit of the generalized linear model by "binning" the fitted and residual values from the model and showing where they may fall outside 95% error bounds.

Usage

```
binred_plot(model, nbins, plot = TRUE)
```

Arguments

`model`	a fitted GLM model, assuming link is "logit"
`nbins`	number of "bins" for the calculation. Defaults to the rounded square root of the number of observations in the model in the absence of a user-specified override here.
`plot`	logical, defaults to TRUE. If TRUE, the function plots the binned residuals. If FALSE, the function returns a data frame of the binned residuals.

Details

The number of bins the user wants is arbitrary. Gelman and Hill (2007) say that, for larger data sets (n >= 100), the number of bins should be the rounded-down square root of the number of observations from the model. For models with a number of observations between 10 and 100, the number of bins should be 10. For models with fewer than 10 observations, the number of bins should be the rounded-down number of observations (divided by 2). The default is the rounded square root of the number of observations in the model. Be smart about what you want here.

Value

`binred_plot()` returns a plot as a **ggplot2** object, as a default. The *y*-axis is the mean residuals of the particular bin. The *x*-axis is the mean fitted values from the bin. Error bounds are 95%. A LOESS smoother is overlaid as a solid blue line.

If `plot = FALSE`, the function returns a data frame of the binned residuals and a summary about whether the residuals are in the error bounds.

Author(s)

Steven V. Miller

Examples

```
set.seed(8675309)
n <- 500
x <- rnorm(n)
a <- -.5
b <- .75
p <- plogis(a + b*x)
y <- rbinom(n, size = 1, prob = p)

M1 <- glm(y ~ x, family=binomial(link='logit'))

binred_plot(M1)
```

---

carrec *Recode a Variable*

Description

This recodes a numeric vector, character vector, or factor according to fairly simple recode specifications that former Stata users will appreciate. Yes, this is taken from John Fox's `recode()` unction in his **car** package. I'm going with `carrec()` (i.e. shorthand for `car::recode()`, phonetically here: "car-wreck") for this package, with an additional shorthand of `carr` that does the same thing.

The goal here is to minimize the number of function clashes with multiple packages that I use in my workflow. For example: **car**, **dplyr**, and **Hmisc** all have `recode()` functions. I rely on the **car** package just for this function, but it conflicts with some other **tidyverse** functions that are vital to my workflow.

Usage

```
carrec(var, recodes, as_fac, as_num = TRUE, levels)

carr(...)
```

Arguments

`var`	numeric vector, character vector, or factor
`recodes`	character string of recode specifications: see below, but former Stata users will find this stuff familiar
`as_fac`	return a factor; default is `TRUE` if `var` is a factor, `FALSE` otherwise
`as_num`	if `TRUE` (which is the default) and `as.factor` is `FALSE`, the result will be coerced to a numeric if all values in the result are numeric. This should be what you want in 99% of applications for regression analysis.
`levels`	an optional argument specifying the order of the levels in the returned factor; the default is to use the sort order of the level names.
`...`	optional, only to make the shortcut (`carr()`) work

Details

Recode specifications appear in a character string, separated by semicolons (see the examples below), of the form input=output. If an input value satisfies more than one specification, then the first (from left to right) applies. If no specification is satisfied, then the input value is carried over to the result. NA is allowed on input and output.

Value

`carrec()` returns a vector, recoded to the specifications of the user. `carr()` is a simple shortcut for`carrec()`.

Author(s)

John Fox

References

Fox, J. and Weisberg, S. (2019). *An R Companion to Applied Regression*, Third Edition, Sage.

Examples

```
x <- seq(1,10)
carrec(x,"0=0;1:2=1;3:5=2;6:10=3")
```

`center_at` *Scoped Helper Verbs*

Description

Scoped helper verbs included in this R Documentation file allow for targeted commands on specified columns. They also rename the ensuing output to conform to my preferred style. The commands here are multiple and explained in the details section below.

Usage

```
center_at(data, x, prefix = "c", na = TRUE, .by = NULL)

diff_at(data, x, o = 1, prefix = "d", .by = NULL)

group_mean_center_at(
  data,
  x,
  mean_prefix = "mean",
  prefix = "b",
  na = TRUE,
  .by
```

```
)

lag_at(data, x, prefix = "l", o = 1, .by = NULL)

log_at(data, x, prefix = "ln", plus_1 = FALSE)

mean_at(data, x, prefix = "mean", na = TRUE, .by = NULL)

r1sd_at(data, x, prefix = "s", na = TRUE, .by = NULL)

r2sd_at(data, x, prefix = "z", na = TRUE, .by = NULL)

rewb_at(data, x, w_prefix = "w", b_prefix = "b", .by)
```

Arguments

data	a data frame
x	a vector, likely in your data frame
prefix	Allows the user to rename the prefix of the new variables. Each function has defaults (see details section).
na	a logical about whether missing values should be ignored in the creation of means and re-scaled variables. Defaults to TRUE (i.e. pass over/remove missing observations). Not applicable to diff_at, lag_at, and log_at.
.by	a selection of columns by which to group the operation. Defaults to NULL. This will eventually become a standard feature of the functions as this operator moves beyond the experimental in **dplyr**. The argument is not applicable to log_at (why would it be) and is optional for all functions except group_mean_center_at. group_mean_center_at must have something specified for grouped mean-centering.
o	The order of lags for calculating differences or lags in diff_at or lag_at. Applicable only to these functions.
mean_prefix	Applicable only to group_mean_center_at. Specifies the prefix of the (assumed) total population mean variables. Default is "mean", though the user can change this as they see fit.
plus_1	Applicable only to log_at. If TRUE, adds 1 to the variables prior to log transformation. If FALSE, performs logarithmic transformation on variables no matter whether 0 occurs (i.e. 0s will come back as -Inf). Defaults to FALSE.
w_prefix	Applicable only to rewb_at, but specifies prefix for so-called "within" variables created by this procedure. Defaults to "w".
b_prefix	Applicable only to rewb_at, but specifies prefix for so-called "between" variables created by this procedure. Defaults to "b".

Details

center_at is a wrapper for mutate_at and rename_at from **dplyr**. It takes supplied vectors and effectively centers them from the mean. It then renames these new variables to have a prefix of c_. The default prefix ("c") can be changed by way of an argument in the function.

diff_at is a wrapper for mutate and across from **dplyr**. It takes supplied vectors and creates differences from the previous value recorded above it. It then renames these new variables to have a prefix of d_ (in the case of a first difference), or something like d2_ in the case of second differences, or d3_ in the case of third differences (and so on). The exact prefix depends on the o argument, which communicates the order of lags you want. It defaults to 1. The default prefix ("d") can be changed by way of an argument in the function, though the naming convention will omit a numerical prefix for first differences.

group_mean_center_at is a wrapper for mutate and across in **dplyr**. It takes supplied vectors and centers an (assumed) group mean of the variables from an (assumed) total population mean of the variables provided to it. It then returns the new variables with a prefix, whose default is b_. This prefix communicates, if you will, a kind of "between" variable in the panel model context, in juxtaposition to "within" variables in the panel model context.

lag_at is a wrapper for mutate and across from **dplyr**. It takes supplied vector(s) and creates lag variables from them. These new variables have a prefix of l[o]_ where o corresponds to the order of the lag (specified by an argument in the function, which defaults to 1). This default prefix ("l") can be changed by way of an another argument in the function.

log_at is a wrapper for mutate and across from **dplyr**. It takes supplied vectors and creates a variable that takes a natural logarithmic transformation of them. It then renames these new variables to have a prefix of ln_. This default prefix ("ln") can be changed by way of an argument in the function. Users can optionally specify that they want to add 1 to the vector before taking its natural logarithm, which is a popular thing to do when positive reals have naturally occurring zeroes.

mean_at is a wrapper for mutate and across from **dplyr**. It takes supplied vectors and creates a variable communicating the mean of the variable. It then renames these new variables to have a prefix of mean_. This default prefix ("mean") can be changed by way of an argument in the function.

r1sd_at is a wrapper for mutate and across from **dplyr**. It both rescales the supplied vectors to new vectors and renames the vectors to each have a prefix of s_. Note the rescaling here is just by one standard deviation and not two. The default prefix ("s") can be changed by way of an argument in the function.

r2sd_at is a wrapper for mutate and across from **dplyr**. It both rescales the supplied vectors to new vectors and renames the vectors to each have a prefix of z_. Note the rescaling here is by two standard deviations and not one. The default prefix ("z") can be changed by way of an argument in the function.

rewb_at is a wrapper for routines done by mean_at, group_mean_center_at, and center_at in this package. It implicitly assumes the data are a panel and runs these three routines in order to create so-called "between" and "within" variables for a "random effects, within-between" analysis. Means are calculated based on available data, so there is no na argument available in this function. The function will fail in the presence of variables in the data matching those that this routine wants to create.

All functions, except for lag_at, will fail in the absence of a character vector of a length of one. They are intended to work across multiple columns instead of just one. If you are wanting to create one new variable, you should think about using some other **dplyr** verb on its own.

Value

The function returns a set of new vectors in a data frame after performing relevant functions. The new vectors have distinct prefixes corresponding with the action performed on them.

Examples

```
set.seed(8675309)
Example <- data.frame(category = c(rep("A", 5),
                                   rep("B", 5),
                                   rep("C", 5)),
                      x = runif(15), y = runif(15),
                      z = sample(1:20, 15, replace=TRUE))

my_vars <- c("x", "y", "z")
center_at(Example, my_vars)

diff_at(Example, my_vars)

diff_at(Example, my_vars, o=3)

lag_at(Example, my_vars)

lag_at(Example, my_vars, o=3)

log_at(Example, my_vars)

log_at(Example, my_vars, plus_1 = TRUE)

mean_at(Example, my_vars)

r1sd_at(Example, my_vars)

r2sd_at(Example, my_vars)
```

---

charitable_contributions

Charitable Contributions Panel Data

Description

This is a toy panel data set on charitable contributions across 10 years for 47 taxpayers. It's useful for illustrating the estimation of panel models.

Usage

```
charitable_contributions
```

Format

A data frame with 470 observations on the following 8 variables.

`subject` a numeric identifier for the subject

`time` a numeric time identifier, as a simple integer from 1 to 10

`charity` the sum of cash and other property contributions, excluding carry-overs from previous years

`income` adjusted gross income

`price` 1 minus the marginal income tax rate, which is defined on income prior to contributions

`age` a dummy variable that equals 1 if the respondent is over 64, 0 otherwise

`ms` a dummy variable that equals 1 if the respondent is married, 0 otherwise

`deps` the number of claimed dependents, as an integer

Details

Frees (2003) is the nominal source for these data, as they appear as toy data sets for use in his book. He in turn cites Banerjee and Frees (1995), though this citation may have been meant for a 1997 article in *Journal of the American Statistical Association*. The actual source for these data as I obtained them is Gujarati (2012). The underlying source of the raw data are supposedly the 1979-1988 *Statistics of Income* Panel of Individual Tax Returns. Given the opacity of the data, and its temporal limitations, these data should only be used for illustration and not inference.

The charitable contributions variable and income variables are very clearly log-transformed. Banerjee and Price (1997) seem to imply the price variable is as well.

`cor2data` *Simulate Data from Correlation Matrix*

Description

A function to simulate data from a correlation matrix. This is useful for illustrating some theoretical properties of regressions when population parameters are known and set in advance.

Usage

```
cor2data(cor, n, seed)
```

Arguments

`cor`	A correlation matrix (of class `matrix`)
`n`	A number of observations to simulate
`seed`	An optional parameter to set a seed. Omitting this generates new simulations every time.

Value

`cor2data()` returns a data frame where all observations are simulated from a standard normal distribution, but with those pre-set correlations.

Author(s)

Steven V. Miller

Examples

```
vars <- c("control", "treat", "instr", "e")
Correlations <- matrix(cbind(1, 0.001, 0.001, 0.001,
                             0.001, 1, 0.85, -0.5,
                            0.001, 0.85, 1, 0.001,
                           0.001, -0.5, 0.001, 1),nrow=4)

rownames(Correlations) <- colnames(Correlations) <- vars

cor2data(Correlations, 1000, 8675309)
```

`corvectors` *Create multivariate data by permutation*

Description

`corvectors()` is a function to obtain a multivariate dataset by specifying the relation between those specified variables.

Usage

```
corvectors(
  data,
  corm,
  tol = 0.005,
  conv = 10000,
  cores = 2,
  splitsize = 1000,
  verbose = FALSE,
  seed
)
```

Arguments

`data`	a data matrix containing the data
`corm`	A value containing the desired correlation or a vector or data matrix containing the desired correlations
`tol`	A single value or a vector of tolerances with length `ncol(data) - 1`. The default is 0.005
`conv`	The maximum iterations allowed. Defaults to 1000.
`cores`	The number of cores to be used for parallel computing
`splitsize`	The size to use for splitting the data
`verbose`	Logical statement. Default is FALSE
`seed`	An optional seed to set

Details

This is liberally copy-pasted from van Kooten and Vink's wonderful-but-no-longer-supported **correlate** package. They call it `correlate()` in their package, but I opt for `corvectors()` here.

Value

`corvectors()` returns a matrix given the specified multivariate relation.

Author(s)

Pascal van Kooten and Gerko Vink

Examples

```
## Not run:
set.seed(8675309)
library(tibble)
# bivariate example, start with zero correlation
as_tibble(data.frame(corvectors(replicate(2, rnorm(100)), .5)))

# multivariate example
as_tibble(data.frame(corvectors(replicate(4, rnorm(100)), c(.5, .6, .7))))


## End(Not run)
```

---

db_lselect *Lazily select variables from multiple tables in a relational database*

Description

`db_lselect()` allows you to select variables from multiple tables in an SQL database. It returns a lazy query that combines all the variables together into one data frame (as a `tibble`). The user can choose to run `collect()` after this query if they see fit.

Usage

```
db_lselect(.data, connection, vars)
```

Arguments

`.data`	a character vector of the tables in a relational database
`connection`	the name of the connection object
`vars`	the variables (entered as class "character") to select from the tables in the database

Details

This is a wrapper function in which **purrr** and **dplyr** are doing the heavy lifting. The tables in the database are declared as a character (or character vector). The variables to select are also declared as a character (or character vector), which are then wrapped in a `one_of()` function within `select()` in **dplyr**.

Value

Assuming a particular structure to the database, the function returns a combined table including all the requested variables from all the tables listed in the `data` character vector. The returned table will have other attributes inherited from how **dplyr** interfaces with SQL, allowing the user to extract some information about the query (e.g. through `show_query()`).

References

Miller, Steven V. 2020. "Clever Uses of Relational (SQL) Databases to Store Your Wider Data (with Some Assistance from `dplyr` and `purrr`)" https://svmiller.com/blog/2020/11/smarter-ways-to-store-your-wide

Examples

```
library(DBI)
library(RSQLite)
library(dplyr)
library(dbplyr)
set.seed(8675309)

A <- data.frame(uid = c(1:10),
                a = rnorm(10),
                b = sample(letters, 10),
                c = rbinom(10, 1, .5))

B <- data.frame(uid = c(11:20),
                a = rnorm(10),
                b = sample(letters, 10),
                c = rbinom(10, 1, .5))

C <- data.frame(uid = c(21:30), a = rnorm(10),
                b = sample(letters, 10),
                c = rbinom(10, 1, .5),
                d = rnorm(10))

con <- dbConnect(SQLite(), ":memory:")

copy_to(con, A, "A",
        temporary=FALSE)

copy_to(con, B, "B",
        temporary=FALSE)

copy_to(con, C, "C",
```

```
        temporary=FALSE)

# This returns no warning because columns "a" and "b" are in all tables
c("A", "B", "C") %>% db_lselect(con, c("uid", "a", "b"))

# This returns two warnings because column "d" is not in 2 of 3 tables.
# ^ this is by design. It'll inform the user about data availability.
c("A", "B", "C") %>% db_lselect(con, c("uid", "a", "b", "d"))
dbDisconnect(con)
```

`ess9_labelled` *Some Labeled Data in the European Social Survey (Round 9)*

Description

These are data to illustrate labeled data and how to process them with `get_var_info()` in this package.

Usage

```
ess9_labelled
```

Format

A data frame with 109 observations on the following 4 variables.

`essround` a numeric constant

`edition` another numeric constant

`cntry` a character vector (with label) for the country in the data

`netusoft` a numeric vector (with label) for self-reported internet consumption of a respondent

Details

Data are condensed summaries from the raw data. They amount to every unique combination of country and self-reported internet consumption. The data are here to illustrate the `get_var_info()` function in this package.

fct_reorg *Reorganize a factor after "re-leveling" it*

Description

`fct_reorg()` is a **forcats** hack that reorganizes a factor after re-leveling it. It has been situationally useful in my coefficient plots over the years.

Usage

```
fct_reorg(fac, ...)
```

Arguments

`fac`	a character or factor vector
`...`	optional parameters to be supplied to **forcats** functions.

Details

Solution comes by way of this issue on Github: https://github.com/tidyverse/forcats/issues/45

Value

This function takes a character or factor vector and first re-levels it before re-coding certain values. The end result is a factor.

Examples

```
x<-factor(c("a","b","c"))
fct_reorg(x, B="b", C="c")
```

filter_refs *Filter a Data Frame of Citations and Return the Entries as a Character*

Description

`filter_refs()` is a convenience function I wrote for filtering a data frame of citations returning the entries as a valid `.bib` entry (as a character vector). I wrote this for more easily passing on citations to the `print_refs()` function also included in this package.

Usage

```
filter_refs(bibdat, criteria, type = "bibtexkey")
```

Arguments

bibdat	a data frame of citations, like the one created by the **bib2df** package
criteria	criteria, specified as a character vector, by which to filter the data frame of citations
type	the particular type of citation entry on which to filter. Defaults to "bibtexkey" (which filters based on a column of unique citation keys). When `type == "year"`, the function filters on a character vector of years.

Details

`filter_refs()` assumes some familiarity with `BibTeX`, `.bib` entries, and depends on the **bib2df** package.

Value

`filter_refs()` takes a data frame of citations, like the one created by the **bib2df** package, and returns a character vector (amounting to a valid `.bib` entry) of citations the user wants. This can then be easily passed to the `print_refs()` function also included in this package.

Examples

```
# Based on `stevepubs` configuration, filter on `BIBTEXKEY` where
# the citation key matches one of these.
filter_refs(stevepubs, c("miller2017etst", "miller2017etjc", "miller2013tdpi"))

# Based on `stevepubs` configuration, filter on `YEAR` where
# the publication year is 2017, 2018, 2019, 2020, or 2021.
filter_refs(stevepubs, c(2017:2021), type = "year")
```

---

fr_plot *Fitted-Residual Plot for Linear Models*

Description

`fr_plot()` provides a visual diagnostic of the linearity assumption and the homoskedasticity assumption of the linear model. Provided an OLS model fit by `lm()` in base R, the function extracts the fitted values and the residuals of the model. Then, it creates a scatterplot of them. By definition, the line of best fit through this scatterplot is flat at 0. A LOESS smoother, drawn over the top of this scatterplot, provides a visual assessment of whether this relationship that is true by definition is a reasonable summary of the relationship between the fitted values and the residuals. The plot allows for a visual assessment of the variation of the residuals across the range of the fitted values as well. The function may be used for diagnostic purposes.

Usage

```
fr_plot(mod, se = TRUE)
```

Arguments

mod	a fitted linear model
se	logical, defaults to TRUE. If TRUE, includes a band for the standard error around the LOESS smoother line. If FALSE, band is suppressed for presentation.

Details

The user can always add **ggplot2** elements on top of this for greater legibility/clarity.

This function is targeted toward students I teach as a kind of convenience function when teaching linear models.

This function right now will issue a stop if the model provided is not of type "lm" generated by the `lm()` function in base R.

Value

`fr_plot()` returns a scatterplot of the fitted values and residuals of a linear model, as a **ggplot2** object. The relationship between the fitted values and the residuals is flat at 0 by definition. A LOESS smoother offers another route to an interpretation of how one might describe this relationship.

Author(s)

Steven V. Miller

Examples

```
set.seed(8675309)
x <- rnorm(400)
y <- 5 + x + rnorm(400)

M1 <- lm(y ~ x)

fr_plot(M1)
```

---

fra_leaderyears *French Leader-Years, 1874-2015*

Description

These are data generated in **peacesciencer** for all French leader-years from 1874 to 2015. I'm going to use these data for stress-testing the calculation of so-called "peace spells" for data that are decidedly imbalanced, as these are.

Usage

```
fra_leaderyears
```

Format

A data frame with 255 observations on the following 10 variables.

`obsid` the unique observation ID in the Archigos data

`ccode` the Correlates of War state code for France (220)

`leader` a name—typically last name—for the leader

`year` an observation year for the leader

`startdate` the start date for the leader's period in office

`enddate` the end date for the leader's period in office

`gmlmidongoing` was there an ongoing inter-state dispute for the leader?

`gmlmidonset` was there a new inter-state dispute onset for the leader?

`gmlmidongoing_init` was there an ongoing inter-state dispute for the leader that the leader initiated?

`gmlmidonset_init` was there a new inter-state dispute onset for the leader that the leader initiated?

Details

Data are generated in the development version (scheduled release of v. 0.7) of **peacesciencer**. Conflict data come from the GML MID data (v. 2.2.1). Leader data come from Archigos (v. 4.1).

References

Goemans, Henk E., Kristian Skrede Gleditsch, and Giacomo Chiozza. 2009. "Introducing Archigos: A Dataset of Political Leaders" *Journal of Peace Research* 46(2): 269–83.

Gibler, Douglas M., Steven V. Miller, and Erin K. Little. 2016. "An Analysis of the Militarized Interstate Dispute (MID) Dataset, 1816-2001." International Studies Quarterly 60(4): 719-730.

`get_sims` *Get Simulations from a Model Object (with New Data)*

Description

`get_sims()` is a function to simulate quantities of interest from a multivariate normal distribution for "new data" from a regression model.

Usage

```
get_sims(model, newdata, nsim, seed)
```

Arguments

`model`	a model object
`newdata`	A data frame on some quantities of interest to be simulated
`nsim`	Number of simulations to be run
`seed`	An optional seed to set

Details

This (should) be a flexible function that takes a `merMod` object (estimated from **lme4**, **blme**, etc.) or a `lm` or `glm` object and generates some quantities of interest when paired with new data of observations of interest. Of note: I've really only tested this function with linear models, generalized linear models, and their mixed model equivalents. For mixed models, this approach does not offer support for the incorporation of the random effects or the random slopes. It's just for the fixed effects, which is typically what most people want anyway. Users who want to better incorporate the random intercepts or slope could find that support in the **merTools** package.

Value

`get_sims()` returns a data frame (as a `tibble`) with the quantities of interest and identifying information about the particular simulation number.

Author(s)

Steven V. Miller

Examples

```
## Not run:
# Note: these models are dumb, but they illustrate how it works.

M1 <- lm(mpg ~ hp, mtcars)
# Note: this function requires the DV to appear somewhere, anywhere in the "new data"
newdat <- data.frame(mpg = 0,
                     hp = c(mean(mtcars$hp) - sd(mtcars$hp),
                            mean(mtcars$hp),
                            mean(mtcars$hp) + sd(mtcars$hp)))

get_sims(M1, newdat, 100, 8675309)

# Note: this is likely a dumb model, but illustrates how it works.
mtcars$mpgd <- ifelse(mtcars$mpg > 25, 1, 0)

M2 <- glm(mpgd ~ hp, mtcars, family=binomial(link="logit"))

# Again: this function requires the DV to be somewhere, anywhere in the "new data"
newdat$mpgd <- 0

# Note: the simulations are returned on their original "link". Here, that's a "logit"
# You can adjust that accordingly. `plogis(y)` will convert those to probabilities.
get_sims(M2, newdat, 100, 8675309)

library(lme4)
M3 <- lmer(mpg ~ hp + (1 | cyl), mtcars)

# Random effects are not required here since we're passing over them.
get_sims(M3, newdat, 100, 8675309)

## End(Not run)
```

get_var_info *Get a small data frame of the variable label and values.*

Description

get_var_info() allows you to peek at your labelled data, extracting a given column's variable labels. The intended use here is mostly "peeking" for the purpose of recoding column's in the absence of a codebook or other form of documentation. gvi() is a shortcut for this function.

Usage

```
get_var_info(.data, x)

gvi(...)
```

Arguments

.data	a data frame
x	a column within the data frame
...	optional, only to make the shortcut (gvi) work

Details

This function leans on var_label() and val_label() in the labelled package, which is a dependency for this package. The function is designed to be used in a "pipe."

Value

If the column in the data frame is not labelled, the function returns a message communicating the absence of labels. If the column in the data frame is labelled, the function returns a small data frame communicating the var_label() output (var), the (often but not always) numeric "code" coinciding with with the label (code), and the "label" attached to it (label).

Examples

```
library(tibble)
library(dplyr)
library(magrittr)

ess9_labelled %>% get_var_info(netusoft) # works, as intended
ess9_labelled %>% get_var_info(cntry) # works, as intended
ess9_labelled %>% get_var_info(ess9round) # barks at you; data are not labelled
```

gmy_dyadyears *German Dyad-Years, 1816-2020*

Description

These are data generated in **peacesciencer** for all German (and Prussian) dyad-years from 1816 to 2020. These are going to be useful in stress-testing what "peace spell" calculations may look like when there is a huge gap in between years. In the Correlates of War context, Germany disappears from the international system from 1945 to 1990. It'll also serve as a nice test for making sure spell calculations don't misbehave in the context of missing data. In this application, there are no data for disputes between 2011 and 2020, but the dyad-years include 2011 to 2020.

Usage

```
gmy_dyadyears
```

Format

A data frame with 11174 observations on the following 6 variables.

`dyad` a unique identifier for the dyad

`ccode1` the Correlates of War state code for Germany (255)

`ccode2` the Correlates of War state code for the other state in the dyad

`year` an observation year for the dyad

`gmlmidongoing` was there an ongoing inter-state dispute in the dyad-year?

`gmlmidonset` was there a new inter-state dispute onset in the dyad-year

Details

Data are generated in the development version (scheduled release of v. 0.7) of **peacesciencer**. Conflict data come from the GML MID data (v. 2.2.1).

References

Gibler, Douglas M., Steven V. Miller, and Erin K. Little. 2016. "An Analysis of the Militarized Interstate Dispute (MID) Dataset, 1816-2001." International Studies Quarterly 60(4): 719-730.

jenny *Set the Only Reproducible Seed That Matters*

Description

`jenny()` sets a reproducible seed of 8675309. It is the only reproducible seed you should use.

Usage

```
jenny(x = 8675309)
```

Arguments

x	a vector

Details

`jenny()` comes with some additional perks if you have the **emo** package installed. The package is optional.

Value

When x is not specified or is 8675309, the function sets a reproducible seed of 8675309 and returns a nice message congratulating you for it. If x is not 8675309, the function sets no reproducible seed and gently admonishes you for wasting its time.

Examples

```
jenny() # will work and reward you for it
jenny(12345) # will not work and will result in a stern message
```

make_perclab *Make Percentage Label for Proportion and Add Percentage Sign*

Description

`make_perclab()` takes a proportion, multiplies it by 100, optionally rounds it, and pastes a percentage sign next to it.

Usage

```
make_perclab(x, d = 2)
```

Arguments

x	a numeric vector
d	digits to round. Defaults to 2.

Details

This function is useful if you're modeling proportions in something like a bar chart (for which proportions are more flexible) but want to label each bar as a percentage. The function here is mostly cosmetic.

Value

The function takes a proportion, multiplies it by 100, (optionally) rounds it to a set decimal point, and pastes a percentage sign next to it.

Examples

```
x <- runif(100)
make_perclab(x)
```

---

make_scale *Rescale Vector to Arbitrary Minimum and Maximum*

Description

`make_scale()` will rescale any vector to have a user-defined minimum and maximum.

Usage

```
make_scale(x, minim, maxim)
```

Arguments

`x`	a numeric vector
`minim`	a desired numeric minimum
`maxim`	a desired numeric maximum

Details

This function is useful if you wanted to do some kind of minimum-maximum rescaling of a variable on some given scale, prominently rescaling to a minimum of 0 and a maximum of 1 (thinking ahead to a regression). The function is flexible enough for any minimum or maximum.

Value

The function takes a numeric vector and returns a rescaled version of it with the observed (desired) minimum, the observed (desired) maximum, and rescaled values between both extremes.

Examples

```
x <- runif(100, 1, 100)
make_scale(x, 2, 5) # works
make_scale(x, 5, 2) # results in message
make_scale(x, 0, 1) # probably why you're using this.
```

map_quiz *Map Quiz Wrong Guesses Across Five Intro to IR Courses*

Description

This is a simple data set that records every wrong guess for map quiz assignments I gave in my intro to IR class at Clemson University across five semesters.

Usage

```
map_quiz
```

Format

A data frame with 1772 observations on the following 8 variables.

`class` an ordered factor of the semester in which the wrong guess was recorded by a student. Levels include "Spring 2018", "Fall 2018", "Spring 2019", "Fall 2019", and "Spring 2020."

`students` the number of students in the class taking the map quiz.

`region` the region map on which the country was located. Values include "Europe", "Africa", "Asia", "Latin America", and "MENA." "MENA" is short for "Middle East and North Africa."

`country` the country I asked the student to correctly identify

`guess` the country that was the actual state incorrectly guessed by the student

`ccode1` the Correlates of War state code for the state I wanted the student to identify in `country`.

`ccode2` the Correlates of War state code for the state that is the wrong guess for the state in `guess`

`mindist` the minimum distance (in kilometers) between `country` and `guess`

Details

Students can always not make a guess and be wrong, which explains the `NA`s in the data. Students were given five separate numbered maps and prompted to identify 10 countries each on them. The maps never changed across five semesters, nor did the prompts. Use these data as you see fit. Obviously, FERPA considerations mean I can't share anything else of potential value here.

mround *Multiply a Number by 100 and Round It (By Default: 2)*

Description

`mround()` is a convenience function I wrote for my annotating bar charts that I make. Assuming a proportion variable, `mround()` will multiply each value by 100 and round it for presentation. By default, it rounds to two. The user can adjust this.

Usage

```
mround(x, d = 2)
```

Arguments

x	a numeric vector
d	the number of decimal points to which the user wants to round. If this is not set, it rounds to two decimal points.

Details

This is a sister function of `make_perclab()` in the same package. This, however, won't add a percentage sign.

Value

The function takes a numeric vector, multiplies it by 100, rounds it (to two digits by default), and returns it to the user.

Examples

```
x <- runif(100)
mround(x)
mround(x, 2) # same as above
mround(x, 3)
```

---

msl.brms *Create Custom* **modelsummary** *List from a Model Fitted in* **brms**

Description

`msl.brms()` is a convenience function for preparing a model fitted in **brms** for presentation in **modelsummary** using its custom "modelsummary_list" feature.

Usage

```
msl.brms(mod, robust = FALSE)
```

Arguments

mod	a model fitted in **brms**
robust	logical, defaults to FALSE. If FALSE, the "tidy" estimates are the mean of the simulated coefficients and the standard deviation of those coefficients. If TRUE, the "tidy" estimates are the median and median absolute deviation.

Details

Bayesians typically recoil at the thought of reporting *p*-values for presentation even as reviewers ask to see these. The *p*-value in question is the "probability of direction" of an effect's existence. Makowski et al. (2019) is a worthwhile read on this procedure. The function calculates what they propose but doesn't require **bayestestR** as a dependency.

Right now, the only easy goodness of fit parameter to return is the number of observations. The output of this function is easily customized ex post by the user.

The function scrubs the b_ prefix in the terms, which is is a convention of the modeling procedure in **brms**.

Value

`msl.brms()` takes a model fitted by **brms** and prepares a custom **modelsummary** list for formatting with that package's `modelsummary()` function.

References

Makowski, Dominique, Mattan S. Ben-Shachar, S. H. Annabel Chen, & Daniel Lüdecke. 2019. "Indices of Effect Existence and Significance in the Bayesian Framework." *Frontiers in Psychology* (10)2767.

Examples

```
## Not run:
# this would take too much time to test...
library(stevedata)
library(brms)

# This is typically how I do it. 4 chains, 4 cores. backend is cmdstanr.
M1 <- brm(gc ~ ool + cvlhs + vl + fl + m + pd, data = Mitchell68,
backend = 'cmdstanr', cores = 4, chains = 4)

MSL <- msl.brms(M1)

MSL

# You can customize this output if you know how {modelsummary} does things.
# For example:

MSL$glance$r.squared <- as.vector(bayes_R2(M1, robust = TRUE))[1]
MSL$glance$adj.r.squared <- as.vector(loo_R2(M1, robust = TRUE))[1]


## End(Not run)
```

n2w *Convert Number (Integer) to Word*

Description

n2w() is a convenience function that converts a positive number (and one assumes an integer) to an English word or set of words.

Usage

```
n2w(x)
```

Arguments

x	a number (one assumes: an integer)

Details

n2w() assumes but does not compel the use of integers. Positive reals are rounded down to the nearest integer.

The function issues a stop in cases where the number provided is negative or a decimal value between 0 (with precision) and 1. Returns of 0 and infinity are still valid.

This function has been passed from multiple sources, including John Fox, Andy Teucher, and an "AJH." I copied it from Jason Miller (no relation) and made some minor edits to it.

Value

n2w() takes a number (one assumes: an integer) and returns that number as an English word (or set of words).

Author(s)

John Fox, Andy Teucher, "AJH", Jason Miller, Steven V. Miller

Examples

```
n2w(3)
n2w(13)
n2w(30)
n2w(8675309)
```

normal_dist *Make and annotate a normal distribution with* **ggplot2**

Description

`normal_dist()` is a convenience function for making a plot of a normal distribution with annotated areas underneath the normal curve.

Usage

```
normal_dist(curvecolor, fillcolor, fontfamily)
```

Arguments

`curvecolor`	What color should the curve itself be. Any **ggplot2**-recognized format should do here.
`fillcolor`	What color should the area underneath the curve be. Any **ggplot2**-recognized format should do here.
`fontfamily`	Font family for labeling areas underneath the curve. OPTIONAL. You can omit this if you'd like.

Details

The normal distribution is a standard normal distribution with a mean of 0 and a standard deviation of 1.

Value

The function returns a fancy plot of a normal distribution annotated with areas underneath the hood. Note that whatever color is supplied in `fillcolor` is automatically lightened for areas further from the center of the curve.

Examples

```
library(stevemisc)
normal_dist("blue","red")
normal_dist("purple","orange")
```

p_z *Convert the p-value you want to the z-value it actually is*

Description

I *loathe* how statistical instruction privileges obtaining a magical p-value by reference to an area underneath the standard normal curve, only to botch what the actual z-value is corresponding to the magical p-value. This simple function converts the p-value you want (typically .05, thanks to R.A. Fisher) to the z-value it actually is for the kind of claims we typically make in inferential statistics. If we're going to do inference the wrong way, let's at least get the z-value right.

Usage

```
p_z(x, ts = TRUE)
```

Arguments

x	a numeric vector (one or multiple) between 0 or 1
ts	a logical, defaults to TRUE. If TRUE, returns two-sided critical z-value. If FALSE, the function returns a one-sided critical z-value.

Details

p_z() takes a p-value of interest and converts it, with precision, to the z-value it actually is. The function takes a vector and returns a vector. The function assumes you're doing something akin to calculating a confidence interval or testing a regression coefficient against a null hypothesis of zero. This means the default output is a two-sided critical z-value. We're taught to use two-sided z-values when we're agnostic about the direction of the effect or statistic of interest, which is, to be frank, hilarious given how most research is typically done.

Value

This function takes a numeric vector, corresponding to the p-value you want, and returns a numeric vector coinciding with the z-value you want under the standard normal distribution. For example, the z-value corresponding with the magic number of .05 (the conventional cutoff for assessing statistical significance) is not 1.96, it's something like 1.959964 (rounding to the default six decimal points).

Examples

```
library(stevemisc)

p_z(.05)
p_z(c(.001, .01, .05, .1))
p_z(.05, ts=FALSE)
p_z(c(.001, .01, .05, .1), ts=FALSE)
```

prepare_refs *Prepare* **bib2df** *Data Frame for Formatting to Various Outputs*

Description

prepare_refs does some last-minute formatting of a data frame created by **bib2df** so that it can be formatted nicely to various outputs.

Usage

```
prepare_refs(bib2df_refs, toformat = "plain")
```

Arguments

bib2df_refs	a data frame created by **bib2df**
toformat	what type of output you are ultimately going to want from print_refs() . Default is "plain".

Details

The function is designed to work more generally in the absence of various fields. Assume, for example, that your data frame has no BOOK field. The function uses the one_of() wrapper to work around this. The "warning" returned by the function is more of a message. This function may be expanded as I think of more use cases.

Value

print_refs() does some last-minute formatting to a data frame created by **bib2df** so that rendering in R Markdown is a little easier and less code-heavy.

See Also

print_refs() for formatting a .bib references to various outputs.

Examples

```
prepare_refs(stevepubs)
```

print_refs *Print and Format* .bib *Entries as References*

Description

print_refs() is a convenience function I found and edited that will allow a user to print and format .bib entries as if they were references. This function is useful if you want to load a .bib entry or set of entries and print them in the middle of a document in R Markdown.

Usage

```
print_refs(
  bib,
  csl = "american-political-science-association.csl",
  toformat = "markdown_strict",
 cslrepo = "https://raw.githubusercontent.com/citation-style-language/styles/master",
  spit_out = TRUE,
  delete_after = TRUE,
  scrub_bibitem = FALSE
)
```

Arguments

bib	a valid .bib entry
csl	a CSL file, matching one available on the Github repository, that the user wants to format the references. Default is "american-political-science-association.csl".
toformat	the output wanted by the user. Default is "markdown_strict".
cslrepo	a directory of CSL files. Defaults to the one on Github.
spit_out	logical, defaults to TRUE. If TRUE, wraps ("spits out") formatted citations in a writeLines() output for the console. If FALSE, returns a character vector.
delete_after	logical, defaults to TRUE. If TRUE, deletes CSL file when it's done. If FALSE, retains CSL for (potential) future use.
scrub_bibitem	logical, defaults to FALSE. If TRUE, function leans on str_replace_all() in {stringr} to remove the use of "bibitem" in LaTeX. Only applicable to cases where toformat is "LaTeX".

Details

print_refs() assumes an active internet connection in the absence of the appropriate CSL file in the working directory. The citation style language (CSL) file supplied by the user must match a file in the massive Github repository of CSL files. Users interested in potential outputs should read more about Pandoc (https://pandoc.org/MANUAL.html). The Github repository of CSL files is available here: https://github.com/citation-style-language/styles.

I have an older version of the APSA CSL file that I prefer to use available on my Github. If you want to use it dynamically, set csl to "american-political-science-association-2015.csl" and set cslrepo to "https://raw.githubusercontent.com/svmiller/hopplock/refs/heads/main". The file is available on my Github here: https://github.com/svmiller/hopplock.

Value

print_refs() takes a .bib entry and returns the requested formatted reference or references from it.

Examples

```
example <- "@Book{vasquez2009twp, Title = {The War Puzzle Revisited},
Author = {Vasquez, John A}, Publisher = {New York, NY: Cambridge University Press},
Year = {2009}}"

print_refs(example)
```

---

ps_btscs *Create "peace years" or "spells" by cross-sectional unit, more generally*

Description

ps_btscs() allows you to create spells ("peace years" in the international conflict context) between observations of some event. This will allow the researcher to better model temporal dependence in binary time-series cross-section ("BTSCS") models. It is an improvement on sbtscs() (included in this package) by its ability to more flexibly work with data that have lots of NAs that bracket the observed event data. It is used in the peacesciencer package.

Usage

```
ps_btscs(data, event, tvar, csunit, pad_ts = FALSE)
```

Arguments

data	the data set with which you are working
event	some event (0, 1) for which you want spells or peace years
tvar	the time variable (e.g. a year)
csunit	the cross-sectional unit (likely a dyad if you're doing boilerplate international conflict stuff)
pad_ts	should time-series be filled when panels are unbalanced/have gaps? Defaults to FALSE.

Details

This function is derived from sbtscs(). See documentation there for more information.

Value

ps_btscs() takes a data frame and returns the data frame with a new variable named spell.

Author(s)

David A. Armstrong, Steven V. Miller

References

Armstrong, Dave. 2016. "**DAMisc**: Dave Armstrong's Miscellaneous Functions." *R package version 1.4-3*.

Miller, Steven V. 2017. "Quickly Create Peace Years for BTSCS Models with `sbtscs` in `stevemisc`." https://svmiller.com/blog/2017/06/quickly-create-peace-years-for-btscs-models-with-stevemisc/

Examples

```
library(dplyr)
library(stevemisc)
data(usa_mids)

# notice: no quotes
ps_btscs(usa_mids, midongoing, year, dyad)
```

`ps_spells` *Create "spells" by cross-sectional unit, even more generally*

Description

`ps_spells()` allows you to create spells ("peace years" in the international conflict context) between observations of some event. This will allow the researcher to better model temporal dependence in binary time-series cross-section ("BTSCS") models. The function is one of three in this package, and the contents of this function are partly ported from the `add_duration()` function in the **spduration** package. That function, unlike the other two I offer here, works much better where panels are decidedly imbalanced.

Usage

```
ps_spells(data, event, tvar, csunit, time_type = "year", ongoing = FALSE)
```

Arguments

`data`	the data set with which you are working
`event`	some event (0, 1) for which you want spells
`tvar`	the time variable (e.g. a year)
`csunit`	the cross-sectional unit (e.g. a dyad or leader)
`time_type`	what type of time-unit are the data? Right now, this will only work with years but support for months and days are forthcoming. Don't do anything with this argument just yet.
`ongoing`	If `TRUE`, successive 1s are considered ongoing events and treated as `NA` after the first 1. If `FALSE`, successive 1s are all treated as failures. Defaults to `FALSE`.

Details

This function is derived from `add_duration()` in the **spduration** package. See documentation there for more information. I thank Andreas Beger for the blessing to port parts of it here.

Value

`ps_spells()` takes a data frame and returns the data frame with a new variable named `spell`.

Author(s)

Andreas Beger, Steven V. Miller

References

Beger, Andreas, Daina Chiba, Daniel W. Hill, Jr, Nils W. Metternich, Shahryar Minhas and Michael D. Ward. 2018. "**spduration**: Split-Population and Duration (Cure) Regression." *R package version 0.17.1*.

Examples

```
One <- ps_btscs(usa_mids, midongoing, year, dyad)
Two <- ps_spells(usa_mids, midongoing, year, dyad)
identical(One, Two)
```

---

r1sd *Scale a vector by one standard deviation*

Description

`r1sd` allows you to rescale a numeric vector such that the ensuing output has a mean of 0 and a standard deviation of 1.

Usage

```
r1sd(x, na = TRUE)
```

Arguments

`x`	a numeric vector
`na`	what to do with NAs in the vector. Defaults to TRUE (i.e. passes over the missing observations)

Details

This is a convenience function since the default `rescale()` function has some additional weirdness that is not welcome for my use cases. By default, `na.rm` is set to TRUE.

Value

The function returns a numeric vector rescaled with a mean of 0 and a standard deviation of 1.

Examples

```
x <- rnorm(100)
r1sd(x)
```

`r2sd` *Scale a vector (or vectors) by two standard deviations*

Description

`r2sd` allows you to rescale a numeric vector such that the ensuing output has a mean of 0 and a standard deviation of .5. `r2sd_at` is a wrapper for `mutate_at` and `rename_at` from **dplyr**. It both rescales the supplied vectors to new vectors and renames the vectors to each have a prefix of `z_`.

Usage

```
r2sd(x, na = TRUE)
```

Arguments

`x`	a vector, likely in your data frame
`na`	what to do with NAs in the vector. Defaults to TRUE (i.e. passes over the missing observations)

Details

By default, `na.rm` is set to TRUE. If you have missing data, the function will just pass over them.

Gelman (2008) argues that rescaling by two standard deviations puts regression inputs on roughly the same scale no matter their original scale. This allows for some honest, if preliminary, assessment of relative effect sizes from the regression output. This does that, but without requiring the `rescale` function from **arm**. I'm trying to reduce the packages on which my workflow relies.

Importantly, I tend to rescale only the ordinal and interval inputs and leave the binary inputs as 0/1. So, my `r2sd` function doesn't have any of the fancier if-else statements that Gelman's `rescale` function has.

Value

The function returns a numeric vector rescaled with a mean of 0 and a standard deviation of .5.

References

Gelman, Andrew. 2008. "Scaling Regression Inputs by Dividing by Two Standard Deviations." *Statistics in Medicine* 27: 2865–2873.

Examples

```
x <- rnorm(100)
r2sd(x)

r2sd_at(mtcars, c("mpg", "hp", "disp"))
```

---

rbnorm *Bounded Normal (Really: Scaled Beta) Distribution*

Description

`rbnorm()` is a function to randomly generate values from a bounded normal (really: a scaled beta) distribution with specified mean, standard deviation, and upper/lower bounds. I use this function to randomly generate data that we treat as interval for sake of getting means and standard deviations, but have discernible bounds (and even skew) to teach students about things like random sampling and central limit theorem.

Usage

```
rbnorm(n, mean, sd, lowerbound, upperbound, round = FALSE, seed)
```

Arguments

`n`	the number of observations to simulate
`mean`	a mean to approximate
`sd`	a standard deviation to approximate
`lowerbound`	a lower bound for the data to be generated
`upperbound`	an upper bound for the data to be generated
`round`	whether to round the values to whole integers. Defaults to FALSE
`seed`	set an optional seed

Details

I call it "bounded normal" when it's really a beta distribution. I'm aware of this. I took much of this code from somewhere. I forget where.

Value

The function returns a vector of simulated data approximating the user-specified conditions.

Examples

```
library(tibble)

tibble(x = rbnorm(10000, 57, 14, 0, 100))
tibble(x = rbnorm(10000, 57, 14, 0, 100, round = TRUE))
tibble(x = rbnorm(10000, 57, 14, 0, 100, seed = 8675309))
```

`rd_plot` *Residual Density Plot for Linear Models*

Description

`rd_plot()` provides a visual diagnostic of the normality assumption of the linear model. Provided an OLS model fit by `lm()` in base R, the function extracts the residuals of the model and creates a density plot of those residuals (solid black line) against a standard normal distribution with a mean of 0 and a standard deviation matching the standard deviation of the residuals from the model. The function may be used for diagnostic purposes.

Usage

```
rd_plot(mod, ...)
```

Arguments

`mod`	a fitted linear model
`...`	optional arguments, intended for adjusting the line width.

Details

The user can always add **ggplot2** elements on top of this for greater legibility/clarity. For example, density plots can be finicky about making observations appear where they don't. Perhaps adjusting the scale of `x` ad hoc, after the fact, may be warranted.

The goal of this function is to emphasize that in many real world applications, the normality assumption of the residuals is never held but can often be reasonably approximated upon visual inspection.

Value

`rd_plot()` returns a density plot as a **ggplot2** object. A density plot of the actual residuals is a solid black line. A stylized normal distribution matching the description of the residuals is the blue dashed line.

Author(s)

Steven V. Miller

Examples

```
M1 <- lm(mpg ~ hp + disp, data = mtcars)

rd_plot(M1)
rd_plot(M1, linewidth = 1.1)
```

revcode *Reverse code a numeric variable*

Description

revcode allows you to reverse code a numeric variable. If, say, you have a Likert item that has values of 1, 2, 3, 4, and 5, the function inverts the scale so that 1 = 5, 2 = 4, 3 = 3, 4 = 2, and 5 = 1.

Usage

```
revcode(x)
```

Arguments

x	a numeric vector

Details

This function passes over NAs you may have in your variable. It does assume, reasonably might I add, that the observed values include both the minimum and the maximum. This is usually the case in a discrete ordered-categorical variable (like a Likert item). It also assumes that the numeric vector supplied to it contains all possible values and that the minimum observed value is 1. This is usually a safe assumption in survey data where the variable of interest is ordinal (either on a 1:4 scale, or 1:5 scale, or 1:10 scale). No matter, use the function with that in mind.

Value

The function returns a numeric vector that reverse codes the the numeric vector that was supplied to it.

Examples

```
data.frame(x1 = rep(c(1:7, NA), 2),
      x2 = c(1:10, 1:4, NA, NA),
     x3 = rep(c(1:4), 4)) -> example_data

library(dplyr)
library(magrittr)

example_data %>% mutate_at(vars("x1", "x2", "x3"), ~revcode(.))
```

sbayesboot *Bootstrap a Regression Model, the Bayesian Way*

Description

sbayesboot() performs a Bayesian bootstrap of a regression model.

Usage

```
sbayesboot(object, reps = 1000L, seed, cluster = NULL, ...)
```

Arguments

object	a regression model object
reps	how many bootstrap replicates the user wants. Defaults to 1000
seed	set an optional seed for reproducibility
cluster	an optional cluster for calibrating the weights
...	optional arguments

Details

The code underpinning sbayesboot() is largely derived from code provided by Grant McDermott and Vincent Arel-Bundock. My approach here takes the flexibility of McDermott's model-agnostic code (along with the ease of specifying clusters) and combines it with Arel-Bundock's update() approach to the actual bootstrapping. I may have screwed something up, so feel free to point to cases where I did screw up.

Value

sbayesboot() takes a fitted regression model and returns a matrix of bootstrapped coefficients (with intercept). These could be easily converted to a data frame for ease of summary.

Author(s)

Grant McDermott, Vincent Arel-Bundock

Examples

```
M1 <- lm(mpg ~ disp + wt + hp, mtcars)

# Default options

BB1 <- sbayesboot(M1)

# Cluster bootstrap on cylinder variable

BB2 <- sbayesboot(M1, cluster=~cyl)
```

sbtscs *Create "peace years" or "spells" by cross-sectional unit*

Description

`sbtscs()` allows you to create spells ("peace years" in the international conflict context) between observations of some event. This will allow the researcher to better model temporal dependence in binary time-series cross-section ("BTSCS") models.

Usage

```
sbtscs(data, event, tvar, csunit, pad_ts = FALSE)
```

Arguments

`data`	the data set with which you are working
`event`	some event (0, 1) for which you want spells or peace years
`tvar`	the time variable (e.g. a year)
`csunit`	the cross-sectional unit (likely a dyad if you're doing boilerplate international conflict stuff)
`pad_ts`	should time-series be filled when panels are unbalanced/have gaps? Defaults to FALSE.

Details

I should confess outright, and it should be obvious to anyone who looks at the code, that I liberally copy from Dave Armstrong's `btscs()` function in the **DAMisc** package. I offer two such improvements. One, the `btscs()` function chokes when a large number of cross-sectional units have no recorded "event." I don't know why this happens but it does. Further, "tidying" up the code by leaning on **dplyr** substantially speeds up computation. Incidentally, this concerns the same cross-sectional units with no recorded events that can choke the `btscs()` function in large numbers.

Value

`sbtscs()` takes a data frame and returns the data frame with a new variable named `spell`.

Author(s)

David A. Armstrong, Steven V. Miller

References

Armstrong, Dave. 2016. "**DAMisc**: Dave Armstrong's Miscellaneous Functions." *R package version 1.4-3*.

Miller, Steven V. 2017. "Quickly Create Peace Years for BTSCS Models with `sbtscs` in `stevemisc`." https://svmiller.com/blog/2017/06/quickly-create-peace-years-for-btscs-models-with-stevemisc/

Examples

```
## Not run:
library(dplyr)
library(stevemisc)
data(usa_mids)

# notice: no quotes
sbtscs(usa_mids, midongoing, year, dyad)

## End(Not run)
```

---

show_ranef *Get a caterpillar plot of random effects from a mixed model*

Description

show_ranef() allows a user estimating a mixed model to quickly plot the random intercepts (with conditional variances) of a given random effect in a mixed model. In cases where there is a random slope over the intercept, the function plots the random slope as another caterpillar plot (as another facet)

Usage

```
show_ranef(model, group, reorder = TRUE)
```

Arguments

model	a fitted mixed model with random intercepts
group	What random intercept/slopes do you want to see as a caterpillar plot? Declare it as a character
reorder	optional argument. DEFAULT is TRUE, which "re-orders" the intercepts by the original value in the data. If FALSE, the ensuing caterpillar plot defaults to a default method of ordering the levels of the random effect by their estimated conditional mode.

Details

This function is a simple wrapper in which broom.mixed and, obviously ggplot2 are doing the heavy lifting.

Value

show_ranef() returns a caterpillar plot of the random intercepts from a given mixed model. If broom.mixed::augment() can process it, this function should work just fine.

Author(s)

Steven V. Miller

Examples

```
library(lme4)
library(stevemisc)
data(sleepstudy)

M1 <- lmer(Reaction ~ Days + (Days | Subject), data=sleepstudy)
show_ranef(M1, "Subject")
show_ranef(M1, "Subject", reorder=FALSE)
```

`smvrnorm` *Simulate from a Multivariate Normal Distribution*

Description

`smvrnorm()` simulates data from a multivariate normal distribution.

Usage

```
smvrnorm(
  n = 1,
  mu,
  sigma,
  tol = 1e-06,
  empirical = FALSE,
  eispack = FALSE,
  seed
)
```

Arguments

`n`	the number of observations to simulate
`mu`	a vector of means
`sigma`	a positive-definite symmetric matrix specifying the covariance matrix of the variables.
`tol`	tolerance (relative to largest variance) for numerical lack of positive-definiteness in `sigma`.
`empirical`	logical. If true, `mu` and `sigma` specify the empirical not population mean and covariance matrix.
`eispack`	logical. values other than FALSE result in an error
`seed`	set an optional seed

Details

This is a simple port and rename of `mvrnorm()` from the **MASS** package. I elect to plagiarize/port it because the **MASS** package conflicts with a lot of things in my workflow, especially `select()`. This is useful for "informal Bayes" approaches to generating quantities of interest from a regression model.

Value

The function returns simulated data from a multivariate normal distribution.

References

B. D. Ripley (1987) *Stochastic Simulation.* Wiley. Page 98.

Examples

```
M1 <- lm(mpg ~ disp + cyl, mtcars)

smvrnorm(100, coef(M1), vcov(M1))
```

---

stevepubs *An Incomplete List of My Publications, All of Which You Should Cite*

Description

These are data on my publications, barring a few things like book reviews and some forthcoming pieces. I use these data to illustrate the `print_refs()` function. You should cite my publications more.

Usage

```
stevepubs
```

Format

A data frame with the following 14 variables.

`CATEGORY` the entry type

`BIBTEXKEY` the unique entry key

`AUTHOR` a list of authors for this entry

`BOOKTITLE` the book title, if appropriate

`JOURNAL` the journal title, if appropriate

`NUMBER` the journal volume number, if appropriate

`PAGES` the range of page numbers, if appropriate

`PUBLISHER` the book publisher, if appropriate

TITLE the title of the publication

VOLUME the journal volume number, if appropriate

YEAR the year of publication, as a character. Publications with no year are assumed to be forthcoming

DOI a DOI, if I entered one

Details

Cite my publications more, you goons. *Extremely Smokey Bear voice* Only YOU can jack my h-index to infinity.

`strategic_rivalries` *Strategic Rivalries, 1494-2010*

Description

A simple summary of all strategic (inter-state) rivalries from Thompson and Dreyer (2012).

Usage

```
data("strategic_rivalries")
```

Format

A data frame with 197 observations on the following 10 variables.

`rivalryno` a numeric vector for the rivalry number

`rivalryname` a character vector for the rivalry name

`sidea` a character vector for the first country in the rivalry

`sideb` a character vector for the second country in the rivalry

`styear` a numeric vector for the start year of the rivalry

`endyear` a numeric vector for the end year of the rivalry

`region` a character vector for the region of the rivalry, per Thompson and Dreyer (2012)

`type1` a character vector for the primary type of the rivalry (spatial, positional, ideological, or interventionary)

`type2` a character vector for the secondary type of the rivalry, if applicable (spatial, positional, ideological, or interventionary)

`type3` a character vector for the tertiary type of the rivalry, if applicable (spatial, positional, ideological, or interventionary)

Details

Information gathered from the appendix of Thompson and Dreyer (2012). Ongoing rivalries are right-bound at 2010, the date of publication for Thompson and Dreyer's handbook. Users are free to change this if they like.

References

Thompson, William R. and David Dreyer. 2012. Handbook of International Rivalries. CQ Press.

Examples

```
data(strategic_rivalries)
```

`studentt` *The Student-t Distribution (Location-Scale)*

Description

These are density, distribution function, quantile function and random generation for the Student-t distribution with location `mu`, scale `sigma`, and degrees of freedom `df`. Base R gives you the so-called "standard" Student-t distribution, with just the varying degrees of freedom. This generalizes that standard Student-t to the three-parameter version.

Usage

```
dst(x, df, mu, sigma)

pst(q, df, mu, sigma)

qst(p, df, mu, sigma)

rst(n, df, mu, sigma)
```

Arguments

`x, q`	a vector of quantiles
`df`	a vector of degrees of freedom
`mu`	a vector for the location value
`sigma`	a vector of scale values
`p`	Vector of probabilities.
`n`	Number of samples to draw from the distribution.

Details

This is a simple hack taken from Wikipedia. It's an itch I've been wanting to scratch for a while. I can probably generalize this outward to allow the tail and log stuff, but I wrote this mostly for the random number generation. Right now, I haven't written this to account for the fact that sigma should be non-negative, but that's on the user to know that (for now).

Value

`dst()` returns the density. `pst()` returns the distribution function. `qst()` returns the quantile function. `rst()` returns random numbers.

See Also

TDist

`tbl_df` *Convert data frame to an object of class "tibble"*

Description

`tbl_df()` ensures legacy compatibility with some of my scripts since the function is deprecated in **dplyr**. `to_tbl()` also added for fun.

Usage

```
tbl_df(...)

to_tbl(...)
```

Arguments

`...`	optional parameters, but don't put anything here. It's just there to quell CRAN checks.

Value

This function takes a data frame and turns it into a tibble.

Examples

```
tbl_df(mtcars)
tbl_df(iris)
```

`theme_steve_web` *Legacy functions for Steve's Preferred* **ggplot2** *Themes and Assorted Stuff*

Description

`theme_steve()`, now in **stevethemes**, was a preferred theme of mine a few years ago. It was basically `theme_bw()` from **ggplot2** theme, but with me tweaking a few things. I then moved to `theme_steve_web()` for most things now, prominently on my website. This theme incorporates the "Open Sans" and "Titillium Web" fonts that I like so much. `post_bg()` is a legacy function for changing the backgrounds on plots to better match what was the background color on my website. `theme_steve_ms()` is for `LaTeX` manuscripts that use the `cochineal` font package. `theme_steve_font()` is for any purpose, allowing you to supply your own font.

Usage

```
theme_steve_web(...)

post_bg(...)

theme_steve_ms(axis_face = "italic", caption_face = "italic", ...)

theme_steve_font(axis_face = "italic", caption_face = "italic", font, ...)
```

Arguments

...	optional stuff, but don't put anything in here. You won't need it.
axis_face	font face ("plain", "italic", "bold", "bold.italic"). Optional, defaults to "italic". Applicable only to theme_steve_ms().
caption_face	font face ("plain", "italic", "bold", "bold.italic"). Optional, defaults to "italic". Applicable only to theme_steve_ms().
font	font family for the plot. Applicable only to theme_steve_font().

Details

theme_steve_web() and theme_steve_ms() both explicitly depend on having the fonts installed on your end. It's ultimately optional for you to have them but the use of these functions imply them. All functions that remain here should be understood as "legacy" functions that will no longer be maintained or updated. The **stevethemes** package will have all my **ggplot2** elements going forward.

Value

post_bg() takes a **ggplot2** plot and changes the background to have a color of "#fdfdfd". theme_steve_web() extends theme_steve() to add custom fonts, notably "Open Sans" and "Titillium Web". In all cases, these functions take a **ggplot2** plot and return another **ggplot2** plot, but with some cosmetic changes. theme_steve_ms() takes a **ggplot2** plot and overlays "Crimson Pro" fonts, which is the basis of the cochineal font package in LaTeX. theme_steve_font() takes a **ggplot2** plot and overlays a font of your choosing.

See Also

ggplot2::theme

Examples

```
## Not run:
library(ggplot2)


ggplot(mtcars, aes(x = mpg, y = hp)) +
  geom_point() + theme_steve_web() +
  labs(title = "A ggplot2 Plot from the Motor Trend Car Road Tests Data",
  subtitle = "Notice the prettier fonts, if you have them.",
```

```
  caption = "Data: ?mtcars in {datasets} in base R.")

ggplot(mtcars, aes(x = mpg, y = hp)) +
  geom_point() + theme_steve_web() +
  post_bg() +
  labs(title = "A ggplot2 Plot from the Motor Trend Car Road Tests Data",
  subtitle = "Notice the slight change in background color",
  caption = "Data: ?mtcars in {datasets} in base R.")

ggplot(mtcars, aes(x = mpg, y = hp)) +
  geom_point() + theme_steve_ms() +
  labs(title = "A ggplot2 Plot from the Motor Trend Car Road Tests Data",
  subtitle = "Notice the fonts will match the 'cochineal' font package in LaTeX.",
  caption = "Data: ?mtcars in {datasets} in base R.")

ggplot(mtcars, aes(x = mpg, y = hp)) +
  geom_point() + theme_steve_font(font = "Comic Sans MS") +
  labs(title = "A ggplot2 Plot from the Motor Trend Car Road Tests Data",
  subtitle = "Notice that this will look ridiculous",
  caption = "Data: ?mtcars in {datasets} in base R.")

## End(Not run)
```

`usa_mids` *United States Militarized Interstate Disputes (MIDs)*

Description

This is a non-directed dyad-year data set for militarized interstate disputes involving the United States. I created these to illustrate the `sbtscs()` function.

Usage

```
usa_mids
```

Format

A data frame with 14586 observations on the following 6 variables.

`dyad` a unique identifier for the dyad

`ccode1` the Correlates of War state code for the United States (2)

`ccode2` the Correlates of War state code for the other state in the dyad

`year` an observation year for the dyad

`midongoing` was there an ongoing inter-state dispute in the dyad-year?

`midonset` was there a new inter-state dispute onset in the dyad-year

Details

Data were generated some time ago. Rare cases where there were multiple disputes ongoing in a given dyad-year were first whittled by isolating 1) unique dispute onsets. Thereafter, the data select the 2) highest fatality, then 3) the highest hostility level, and then 4) the longer dispute, until 5) just picking whichever one came first. There are no duplicate non-directed dyad-year observations.

References

Gibler, Douglas M., Steven V. Miller, and Erin K. Little. 2016. "An Analysis of the Militarized Interstate Dispute (MID) Dataset, 1816-2001." International Studies Quarterly 60(4): 719-730.

`wls` *Get Weighted Least Squares of Your OLS Model*

Description

`wls()` takes an OLS model and re-estimates it using a weighted least squares approach. Weighted least squares is often a "textbook" approach to dealing with the presence of heteroskedastic standard errors, for which the weighted least squares estimates are compared to the OLS estimates of uncertainty to check for consistency or potential inferential implications.

Usage

```
wls(mod)
```

Arguments

`mod`	a fitted OLS model

Details

The function *should* be robust to potential model specification oddities (e.g. polynomials and fixed effects). It also should perform nicely in the presence of missing data, if and only if `na.action = na.exclude` is supplied first to the offending OLS model supplied to the function for a weighted least squares re-estimation.

Value

`wls()` returns a new model object that is a weighted least squares re-estimation of the OLS model supplied to it.

Author(s)

Steven V. Miller

Examples

```
M1 <- lm(mpg ~ ., data=mtcars)
M2 <- wls(M1)

summary(M2)
```

---

wom *Generate Week of the Month from a Date*

Description

`wom()` is a convenience function I use for constructing calendars in **ggplot2**. It takes a date and returns, as a numeric vector, the week of the month for the date given to it.

Usage

```
wom(x)
```

Arguments

x	a date

Details

`wom()` assumes Sunday is the start of the week. This can assuredly be customized later in this function, but right now the assumption is Sunday is the start of the week (and not Monday, as it might be in other contexts).

Value

`wom()` is a convenience function I use for constructing calendars in **ggplot2**. It takes a date and returns, as a numeric vector, the week of the month for the date given to it.

Examples

```
wom(as.Date("2022-01-01"))

wom(Sys.Date())
```

Index

∗ **datasets**
    charitable_contributions, 9
    ess9_labelled, 14
    fra_leaderyears, 17
    gmy_dyadyears, 21
    map_quiz, 24
    stevepubs, 43
    strategic_rivalries, 44
    usa_mids, 48
%nin%, 3

binred_plot, 4

carr *(carrec)*, 5
carrec, 5
center_at, 6
charitable_contributions, 9
cor2data, 10
corvectors, 11

db_lselect, 12
diff_at *(center_at)*, 6
dst *(studentt)*, 45

ess9_labelled, 14

fct_reorg, 15
filter_refs, 15
fr_plot, 16
fra_leaderyears, 17

get_sims, 18
get_var_info, 20
ggplot2::theme, *47*
gmy_dyadyears, 21
group_mean_center_at *(center_at)*, 6
gvi *(get_var_info)*, 20

jenny, 22

lag_at *(center_at)*, 6
log_at *(center_at)*, 6

make_perclab, 22
make_scale, 23
map_quiz, 24
mean_at *(center_at)*, 6
mround, 24
msl.brms, 25

n2w, 27
normal_dist, 28

p_z, 29
post_bg *(theme_steve_web)*, 46
prepare_refs, 30
print_refs, 31
ps_btscs, 32
ps_spells, 33
pst *(studentt)*, 45

qst *(studentt)*, 45

r1sd, 34
r1sd_at *(center_at)*, 6
r2sd, 35
r2sd_at *(center_at)*, 6
rbnorm, 36
rd_plot, 37
revcode, 38
rewb_at *(center_at)*, 6
rst *(studentt)*, 45

sbayesboot, 39
sbtscs, 40
show_ranef, 41
smvrnorm, 42
stevepubs, 43
strategic_rivalries, 44
studentt, 45

tbl_df, 46

TDist, *46*
theme_steve_font *(*theme_steve_web*)*, 46
theme_steve_ms *(*theme_steve_web*)*, 46
theme_steve_web, 46
to_tbl *(*tbl_df*)*, 46

usa_mids, 48

wls, 49
wom, 50