

# Package ‘stevemisc’

October 22, 2021

**Type** Package

**Title** Steve's Miscellaneous Functions

**Version** 1.3.0

**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, broom.mixed, labelled, arm, parallel, purrr, tibble, dplyr, methods, lme4, rlang, forcats, stringr, httr, rmarkdown, tidyr

**Suggests** knitr, DBI, RSQLite, dbplyr

**RoxygenNote** 7.1.2

**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] (<<https://orcid.org/0000-0003-1883-3169>>)

**Maintainer** Steve Miller <[steven.v.miller@gmail.com](mailto:steven.v.miller@gmail.com)>

Repository CRAN

Date/Publication 2021-10-22 16:10:02 UTC

## R topics documented:

carrec	3
cor2data	4
corvectors	5
db_lselect	6
ess9_labelled	8
fct_reorg	9
fra_leaderyears	9
get_sims	10
get_var_info	12
gmy_dyadyears	13
jenny	14
linloess_plot	14
make_perclab	15
make_scale	16
mround	17
normal_dist	18
prepare_refs	19
print_refs	20
ps_btscs	21
ps_spells	22
p_z	23
r1sd	24
r2sd	25
rbnorm	26
revcode	27
sbtscs	28
show_ranef	29
smvrnorm	30
stevepubs	31
strategic_rivalries	32
studentt	33
tbl_df	34
theme_steve	35
usa_mids	36
%nin%	37

Index

39

---

 carrec

*Create multivariate data by permutation*


---

### 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()` function 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

<code>var</code>	numeric vector, character vector, or factor
<code>recodes</code>	character string of recode specifications: see below, but former Stata users will find this stuff familiar
<code>as_fac</code>	return a factor; default is TRUE if <code>var</code> is a factor, FALSE otherwise
<code>as_num</code>	if TRUE (which is the default) and <code>as.factor</code> 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.
<code>levels</code>	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.
<code>...</code>	optional, only to make the shortcut ( <code>carr</code> ) 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")
```

---

`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**

<code>cor</code>	A correlation matrix (of class <code>matrix</code> )
<code>n</code>	A number of observations to simulate
<code>seed</code>	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**

<code>.data</code>	a character vector of the tables in a relational database
<code>connection</code>	the name of the connection object
<code>vars</code>	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`)" <http://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	<i>Reorganize a factor after "re-leveling" it</i>
-----------	---

---

### 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")
```

---

fra_leaderyears	<i>French Leader-Years, 1874-2015</i>
-----------------	---------------------------------------

---

### 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

```
# 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)
```

---

`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

<code>.data</code>	a data frame
<code>x</code>	a column within the data frame
<code>...</code>	optional, only to make the shortcut ( <code>gvi</code> ) 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

`cocode1` the Correlates of War state code for Germany (255)

`cocode2` 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
```

---

`linloess_plot`*Compare Linear Smoother to LOESS Smoother for Your OLS Model*

---

**Description**

`linloess_plot()` provides a visual diagnostic of the linearity assumption of the OLS model. Provided an OLS model fit by `lm()` in base R, the function extracts the model frame and creates a faceted scatterplot. For each facet, a linear smoother and LOESS smoother are estimated over the points. Users who run this function can assess just how much the linear smoother and LOESS smoother diverge. The more they diverge, the more the user can determine how much the OLS model is a good fit as specified. The plot will also point to potential outliers that may need further consideration.

**Usage**

```
linloess_plot(mod, ...)
```

**Arguments**

mod	a fitted OLS model
...	optional parameters, passed to the scatterplot ( <code>geom_point()</code> ) component of this function. Useful if you want to make the smoothers more legible against the points.

**Details**

This function makes an implicit assumption that there is no variable in the regression formula with the name ".y".

**Value**

`linloess_plot()` returns a faceted scatterplot as a **ggplot2** object. The linear smoother is in solid blue (with blue standard error bands) and the LOESS smoother is a dashed black line (with gray/default standard error bands). You can add cosmetic features to it after the fact. The function may spit warnings to you related to the LOESS smoother, depending your data. I think these to be fine the extent to which this is really just a visual aid and an informal diagnostic for the linearity assumption.

**Author(s)**

Steven V. Miller

**Examples**

```
M1 <- lm(mpg ~ ., data=mtcars)
linloess_plot(M1)
linloess_plot(M1, color="black", pch=21)
```

---

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.
```

---

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)
```

---

normal_dist	<i>Make and annotate a normal distribution with <b>ggplot2</b></i>
-------------	--

---

### 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

<code>curvecolor</code>	What color should the curve itself be. Any <b>ggplot2</b> -recognized format should do here.
<code>fillcolor</code>	What color should the area underneath the curve be. Any <b>ggplot2</b> -recognized format should do here.
<code>fontfamily</code>	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")
```

---

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 <b>bib2df</b>
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)
```

## 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 = "latex",  
  csrepo = "https://raw.githubusercontent.com/citation-style-language/styles/master",  
  spit_out = TRUE,  
  delete_after = TRUE  
)
```

## Arguments

<code>bib</code>	a valid <code>.bib</code> entry
<code>csl</code>	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".
<code>toformat</code>	the output wanted by the user. Default is "latex".
<code>csrepo</code>	a directory of CSL files. Defaults to the one on Github.
<code>spit_out</code>	logical, defaults to TRUE. If TRUE, wraps ("spits out") formatted citations in a <code>writelines()</code> output for the console. If 'FALSE', returns a character vector.
<code>delete_after</code>	logical, defaults to TRUE. If TRUE, deletes CSL file when it's done. If FALSE, retains CSL for (potential) future use.

## 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>.

## Value

`print_refs()` takes a `.bib` entry and returns a formatted reference 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	<i>Create "peace years" or "spells" by cross-sectional unit, more generally</i>
----------	---

---

**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`.” <http://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

<code>data</code>	the data set with which you are working
<code>event</code>	some event (0, 1) for which you want spells
<code>tvar</code>	the time variable (e.g. a year)
<code>csunit</code>	the cross-sectional unit (e.g. a dyad or leader)
<code>time_type</code>	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.
<code>ongoing</code>	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)
```

---

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)
```

---

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

<code>x</code>	a numeric vector
<code>na</code>	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)
```

```
r2sd_at(data, x)
```

**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)
data	a data frame

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

---

revcode	<i>Reverse code a numeric variable</i>
---------	--

---

## 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(.))
```

---

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`." <http://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(data, grp, reorder = TRUE)
```

## Arguments

data	a fitted mixed model with random intercepts
grp	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**

<code>n</code>	the number of observations to simulate
<code>mu</code>	a vector of means
<code>sigma</code>	a positive-definite symmetric matrix specifying the covariance matrix of the variables.
<code>tol</code>	tolerance (relative to largest variance) for numerical lack of positive-definiteness in <code>sigma</code> .
<code>empirical</code>	logical. If true, <code>mu</code> and <code>sigma</code> specify the empirical not population mean and covariance matrix.
<code>eispack</code>	logical. values other than <code>FALSE</code> result in an error
<code>seed</code>	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	<i>The Student-t Distribution (Location-Scale)</i>
----------	--

---

**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

*Steve's Preferred **ggplot2** Themes and Assorted Stuff*


---

## Description

theme\_steve() was a preferred theme of mine a few years ago. It is basically theme\_bw() from **ggplot2** theme, but with me tweaking a few things. I've since moved to theme\_steve\_web() for most things now, prominently on my website. It incorporates the "Open Sans" and "Titillium Web" fonts that I like so much. post\_bg() is for changing the backgrounds on plots to better match my website for posts that I write. 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(...)
```

```
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() depends on having the fonts installed on your end. It's ultimately optional for you to have them.

## Value

post\_bg() takes a **ggplot2** plot and changes the background to have a color of "#fdfdfd". theme\_steve() takes a **ggplot2** plot and formats it to approximate theme\_bw() from **ggplot2**, but with some other tweaks. 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 Text" 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() +
  labs(title = "A ggplot2 Plot from the Motor Trend Car Road Tests Data",
        subtitle = "We've all seen this plot over a hundred times.",
        caption = "Data: ?mtcars in {datasets} in base R.")

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
- `cocode1` the Correlates of War state code for the United States (2)
- `cocode2` 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.

---

<code>%nin%</code>	<i>Find Non-Matching Elements</i>
--------------------	-----------------------------------

---

**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))
```

# Index

- \* **datasets**
  - ess9\_labelled, 8
  - fra\_leaderyears, 9
  - gmy\_dyadyears, 13
  - stevepubs, 31
  - strategic\_rivalries, 32
  - usa\_mids, 36
- %nin%, 37
- carr (carrec), 3
- carrec, 3
- cor2data, 4
- corvectors, 5
- db\_lselect, 6
- dst (studentt), 33
- ess9\_labelled, 8
- fct\_reorg, 9
- fra\_leaderyears, 9
- get\_sims, 10
- get\_var\_info, 12
- gmy\_dyadyears, 13
- gvi (get\_var\_info), 12
- jenny, 14
- linloess\_plot, 14
- make\_perclab, 15
- make\_scale, 16
- mround, 17
- normal\_dist, 18
- p\_z, 23
- post\_bg (theme\_steve), 35
- prepare\_refs, 19
- print\_refs, 20
- ps\_btscs, 21
- ps\_spells, 22
- pst (studentt), 33
- qst (studentt), 33
- r1sd, 24
- r2sd, 25
- r2sd\_at (r2sd), 25
- rbnorm, 26
- revcode, 27
- rst (studentt), 33
- sbtscs, 28
- show\_ranef, 29
- smvrnorm, 30
- stevepubs, 31
- strategic\_rivalries, 32
- studentt, 33
- tbl\_df, 34
- TDist, 34
- theme\_steve, 35
- theme\_steve\_font (theme\_steve), 35
- theme\_steve\_ms (theme\_steve), 35
- theme\_steve\_web (theme\_steve), 35
- to\_tbl (tbl\_df), 34
- usa\_mids, 36