

# Package ‘posterior’

June 10, 2022

**Title** Tools for Working with Posterior Distributions

**Version** 1.2.2

**Date** 2022-06-09

**Description** Provides useful tools for both users and developers of packages for fitting Bayesian models or working with output from Bayesian models. The primary goals of the package are to:

- (a) Efficiently convert between many different useful formats of draws (samples) from posterior or prior distributions.
- (b) Provide consistent methods for operations commonly performed on draws, for example, subsetting, binding, or mutating draws.
- (c) Provide various summaries of draws in convenient formats.
- (d) Provide lightweight implementations of state of the art posterior inference diagnostics. References: Vehtari et al. (2021) [doi:10.1214/20-BA1221](https://doi.org/10.1214/20-BA1221).

**Depends** R (>= 3.2.0)

**Imports** abind, checkmate, rlang (>= 0.4.7), stats, tibble (>= 3.0.0), vctrs, tensorA, pillar, distributional, parallel, matrixStats

**Suggests** testthat (>= 2.1.0), caret (>= 6.0.84), gbm (>= 2.1.8), randomForest (>= 4.6.14), e1071 (>= 1.7.3), dplyr, tidyr, knitr, rmarkdown

**License** BSD\_3\_clause + file LICENSE

**Encoding** UTF-8

**URL** <https://mc-stan.org/posterior/>, <https://discourse.mc-stan.org/>

**BugReports** <https://github.com/stan-dev/posterior/issues>

**RoxygenNote** 7.1.2

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Paul-Christian Bürkner [aut, cre],  
Jonah Gabry [aut],  
Matthew Kay [aut],  
Aki Vehtari [aut],

Måns Magnusson [ctb],  
 Rok Češnovar [ctb],  
 Ben Lambert [ctb],  
 Ozan Adıgüzel [ctb],  
 Jacob Socolar [ctb]

**Maintainer** Paul-Christian Bürkner <paul.buerkner@gmail.com>

**Repository** CRAN

**Date/Publication** 2022-06-09 23:00:02 UTC

## R topics documented:

posterior-package . . . . .	3
as_rvar . . . . .	4
bind_draws . . . . .	5
chol.rvar . . . . .	6
diagnostics . . . . .	7
draws . . . . .	7
draws-index . . . . .	8
draws_array . . . . .	10
draws_df . . . . .	11
draws_list . . . . .	13
draws_matrix . . . . .	15
draws_of . . . . .	16
draws_rvars . . . . .	18
draws_summary . . . . .	19
ess_basic . . . . .	22
ess_bulk . . . . .	23
ess_mean . . . . .	25
ess_quantile . . . . .	26
ess_sd . . . . .	27
ess_tail . . . . .	28
example_draws . . . . .	30
extract_variable . . . . .	31
extract_variable_matrix . . . . .	32
is_rvar . . . . .	33
mcse_mean . . . . .	33
mcse_quantile . . . . .	34
mcse_sd . . . . .	36
merge_chains . . . . .	37
mutate_variables . . . . .	38
order_draws . . . . .	39
print.draws_array . . . . .	40
print.draws_df . . . . .	41
print.draws_list . . . . .	42
print.draws_matrix . . . . .	43
print.draws_rvars . . . . .	44
print.rvar . . . . .	45

quantile2 . . . . .	47
rdo . . . . .	48
rename_variables . . . . .	49
repair_draws . . . . .	50
resample_draws . . . . .	51
reserved_variables . . . . .	53
rfun . . . . .	54
rhat . . . . .	55
rhat_basic . . . . .	56
rstar . . . . .	58
rvar . . . . .	60
rvar-dist . . . . .	62
rvar-matmult . . . . .	63
rvar-summaries-over-draws . . . . .	64
rvar-summaries-within-draws . . . . .	67
rvar_apply . . . . .	69
rvar_is_finite . . . . .	70
rvar_rng . . . . .	71
split_chains . . . . .	73
subset_draws . . . . .	73
thin_draws . . . . .	76
weights.draws . . . . .	77
weight_draws . . . . .	78
[.draws_array . . . . .	79
[.draws_matrix . . . . .	80
<b>Index</b>	<b>81</b>

---

posterior-package      *Tools for working with posterior (and prior) distributions*

---

## Description

The **posterior** package is intended to provide useful tools for both users and developers of packages for fitting Bayesian models or working with output from Bayesian models. The primary goals of the package are to:

- Efficiently convert between many different useful formats of draws (samples) from posterior or prior distributions.
- Provide consistent methods for operations commonly performed on draws, for example, subsetting, binding, or mutating draws.
- Provide various summaries of draws in convenient formats.
- Provide lightweight implementations of state of the art posterior inference diagnostics.

## Package options

The following options are used to format and print `draws` objects, as in `print.draws_array()`, `print.draws_df()`, `print.draws_list()`, `print.draws_matrix()`, and `print.draws_rvars()`:

- `posterior.max_draws`: Maximum number of draws to print.
- `posterior.max_iterations`: Maximum number of iterations to print.
- `posterior.max_chains`: Maximum number of chains to print.
- `posterior.max_variables`: Maximum number of variables to print.

The following option is used to format and print `rvar` objects, as in `print.rvar()` and `print.draws_rvars()`:

- `posterior.rvar_summary`: What style of summary to display: "mean\_sd" displays  $\text{mean} \pm \text{sd}$ , "median\_mad" displays  $\text{median} \pm \text{mad}$ .

The following option is used to construct new `rvar` objects, as in `rfun()` and `rdo()`:

- `posterior.rvar_ndraws`: The number of draws used to construct new random variables when this number cannot be determined from existing arguments (e.g., other `rvars` passed to a function).

The following options are used to control warning messages:

- `posterior.warn_on_merge_chains`: (logical) Some operations will trigger an automatic merging of chains, for example, because chains do not match between two objects involved in a binary operation. Whether this causes a warning can be controlled by this option.

---

as\_rvar

*Coerce to a random variable*

---

## Description

Convert `x` to an `rvar` object.

## Usage

```
as_rvar(x, dim = NULL, dimnames = NULL, nchains = NULL)
```

## Arguments

<code>x</code>	(multiple options) An object that can be converted to an <code>rvar</code> , such as a vector, array, or an <code>rvar</code> itself.
<code>dim</code>	(integer vector) One or more integers giving the maximal indices in each dimension to override the dimensions of the <code>rvar</code> to be created (see <code>dim()</code> ). If <code>NULL</code> (the default), <code>dim</code> is determined by the input. <b>NOTE:</b> This argument controls the dimensions of the <code>rvar</code> , not the underlying array, so you cannot change the number of draws using this argument.
<code>dimnames</code>	(list) Character vectors giving the names in each dimension to override the names of the dimensions of the <code>rvar</code> to be created (see <code>dimnames()</code> ). If <code>NULL</code> (the default), this is determined by the input. <b>NOTE:</b> This argument controls the names of the dimensions of the <code>rvar</code> , not the underlying array.
<code>nchains</code>	(positive integer) The number of chains. The default is 1.

## Details

For objects that are already [rvars](#), returns them (with modified dimensions if `dim` is not `NULL`).

For numeric or logical vectors or arrays, returns an [rvar](#) with a single draw and the same dimensions as `x`. This is in contrast to the [rvar\(\)](#) constructor, which treats the first dimension of `x` as the draws dimension. As a result, [as\\_rvar\(\)](#) is useful for creating constants.

## Value

An object of class "rvar" representing a random variable.

## See Also

[rvar\(\)](#) to construct [rvars](#) directly. See [rdo\(\)](#), [rfun\(\)](#), and [rvar\\_rng\(\)](#) for higher-level interfaces for creating [rvars](#).

## Examples

```
# You can use as_rvar() to create "constant" rvars (having only one draw):
x <- as_rvar(1)
x

# Such constants can be of arbitrary shape:
as_rvar(1:4)
as_rvar(matrix(1:10, nrow = 5))
as_rvar(array(1:12, dim = c(2, 3, 2)))
```

---

bind\_draws

*Bind draws objects together*

---

## Description

Bind multiple [draws](#) objects together to form a single draws object.

## Usage

```
bind_draws(x, ...)

## S3 method for class 'draws_matrix'
bind_draws(x, ..., along = "variable")

## S3 method for class 'draws_array'
bind_draws(x, ..., along = "variable")

## S3 method for class 'draws_df'
bind_draws(x, ..., along = "variable")
```

```
## S3 method for class 'draws_list'
bind_draws(x, ..., along = "variable")

## S3 method for class 'draws_rvars'
bind_draws(x, ..., along = "variable")
```

### Arguments

**x** (draws) A [draws](#) object. The draws format of x will define the format of the returned draws object.

**...** (draws) Additional [draws](#) objects to bind to x.

**along** (string) The dimension along which draws objects should be bound together. Possible values are "variable" (the default), "chain", "iteration", and "draw". Not all options are supported for all input formats.

### Value

A draws object of the same class as x.

### Examples

```
x1 <- draws_matrix(alpha = rnorm(5), beta = rnorm(5))
x2 <- draws_matrix(alpha = rnorm(5), beta = rnorm(5))
ndraws(x1)
ndraws(x2)
x3 <- bind_draws(x1, x2, along = "draw")
ndraws(x3)

x4 <- draws_matrix(theta = rexp(5))
x5 <- bind_draws(x1, x4, along = "variable")
variables(x5)
```

---

chol.rvar

*Cholesky decomposition of random matrix*

---

### Description

Cholesky decomposition of an [rvar](#) containing a matrix.

### Usage

```
## S3 method for class 'rvar'
chol(x, ...)
```

**Arguments**

`x` (rvar) A 2-dimensional [rvar](#).  
`...` Additional parameters passed on to `chol.tensor()`

**Value**

An [rvar](#) containing the upper triangular factor of the Cholesky decomposition, i.e., the matrix  $R$  such that  $R'R = x$ .

---

diagnostics *List of available convergence diagnostics*

---

**Description**

A list of available diagnostics and links to their individual help pages.

**Details**

<b>Function</b>	<b>Description</b>
<a href="#">ess_basic()</a>	Basic version of effective sample size
<a href="#">ess_bulk()</a>	Bulk effective sample size
<a href="#">ess_tail()</a>	Tail effective sample size
<a href="#">ess_quantile()</a>	Effective sample sizes for quantiles
<a href="#">ess_sd()</a>	Effective sample sizes for standard deviations
<a href="#">mcse_mean()</a>	Monte Carlo standard error for the mean
<a href="#">mcse_quantile()</a>	Monte Carlo standard error for quantiles
<a href="#">mcse_sd()</a>	Monte Carlo standard error for standard deviations
<a href="#">rhat_basic()</a>	Basic version of Rhat
<a href="#">rhat()</a>	Improved, rank-based version of Rhat
<a href="#">rstar()</a>	R* diagnostic

**Value**

See individual functions for a description of return types.

---

draws *Transform to draws objects*

---

**Description**

Try to transform an R object to a format supported by the **posterior** package.

## Usage

```
as_draws(x, ...)
```

```
is_draws(x)
```

## Arguments

`x` (draws) A draws object or another R object for which the method is defined.  
`...` Arguments passed to individual methods (if applicable).

## Details

The class "draws" is the parent class of all supported formats, which also have their own subclasses of the form "draws\_{format}" (e.g. "draws\_array").

## Value

If possible, a draws object in the closest supported format to `x`. The formats are linked to in the **See Also** section below.

## See Also

Other formats: [draws\\_array\(\)](#), [draws\\_df\(\)](#), [draws\\_list\(\)](#), [draws\\_matrix\(\)](#), [draws\\_rvars\(\)](#)

## Examples

```
# create some random draws
x <- matrix(rnorm(30), nrow = 10)
colnames(x) <- c("a", "b", "c")
str(x)

# transform to a draws object
y <- as_draws(x)
str(y)

# remove the draws classes from the object
class(y) <- class(y)[-1:2]
str(y)
```

---

draws-index

*Index draws objects*

---

## Description

Index variables, iterations, chains, and draws.



**Usage**

```
variables(x, ...)  
  
variables(x) <- value  
  
iteration_ids(x)  
  
chain_ids(x)  
  
draw_ids(x)  
  
nvariables(x, ...)  
  
niterations(x)  
  
nchains(x)  
  
ndraws(x)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
...	Arguments passed to individual methods (if applicable).
value	(character vector) For <code>variables(x) &lt;- value</code> , the new variable names to use.

**Details**

The methods `variables()`, `iteration_ids()`, `chain_ids()`, and `draw_ids()` return vectors of all variables, iterations, chains, and draws, respectively. In contrast, the methods `nvariables()`, `niterations()`, `nchains()`, and `ndraws()` return the number of variables, iterations, chains, and draws, respectively.

`variables(x) <- value` allows you to modify the vector of variable names, similar to how `names(x) <- value` works for vectors and lists. For renaming specific variables, `rename_variables()` may offer a more convenient approach.

**Value**

For `variables()`, a character vector.

For `iteration_ids()`, `chain_ids()`, and `draw_ids()`, an integer vector.

For `niterations()`, `nchains()`, and `ndraws()`, a scalar integer.

**Examples**

```
x <- example_draws()  
  
variables(x)  
nvariables(x)  
variables(x) <- letters[1:nvariables(x)]
```

```
iteration_ids(x)
niterations(x)
```

```
chain_ids(x)
nchains(x)
```

```
draw_ids(x)
ndraws(x)
```

---

```
draws_array
```

```
The draws_array format
```

---

### Description

The `as_draws_array()` methods convert objects to the `draws_array` format. The `draws_array()` function creates an object of the `draws_array` format based on a set of numeric vectors. See **Details**.

### Usage

```
as_draws_array(x, ...)

## Default S3 method:
as_draws_array(x, ...)

## S3 method for class 'draws_array'
as_draws_array(x, ...)

## S3 method for class 'draws_matrix'
as_draws_array(x, ...)

## S3 method for class 'draws_df'
as_draws_array(x, ...)

## S3 method for class 'draws_list'
as_draws_array(x, ...)

## S3 method for class 'draws_rvars'
as_draws_array(x, ...)

## S3 method for class 'mcmc'
as_draws_array(x, ...)

## S3 method for class 'mcmc.list'
as_draws_array(x, ...)
```

```
draws_array(..., .nchains = 1)
```

```
is_draws_array(x)
```

### Arguments

`x` An object to convert to a `draws_array` object.

`...` For `as_draws_array()`: Arguments passed to individual methods (if applicable). For `draws_array()`: Named arguments containing numeric vectors each defining a separate variable.

`.nchains` (positive integer) The number of chains. The default is 1.

### Details

Objects of class "draws\_array" are 3-D arrays with dimensions "iteration", "chain", and "variable". See **Examples**.

### Value

A `draws_array` object, which has classes `c("draws_array", "draws", "array")`.

### See Also

Other formats: [draws\\_df\(\)](#), [draws\\_list\(\)](#), [draws\\_matrix\(\)](#), [draws\\_rvars\(\)](#), [draws](#)

### Examples

```
x1 <- as_draws_array(example_draws())
class(x1)
print(x1)
str(x1)

x2 <- draws_array(a = rnorm(10), b = rnorm(10), c = 1)
class(x2)
print(x2)
str(x2)
```

---

draws\_df

*The draws\_df format*

---

### Description

The `as_draws_df()` methods convert objects to the `draws_df` format. The `draws_df()` function creates an object of the `draws_df` format based on a set of numeric vectors. See **Details**.

**Usage**

```

as_draws_df(x, ...)

## Default S3 method:
as_draws_df(x, ...)

## S3 method for class 'data.frame'
as_draws_df(x, ...)

## S3 method for class 'draws_df'
as_draws_df(x, ...)

## S3 method for class 'draws_matrix'
as_draws_df(x, ...)

## S3 method for class 'draws_array'
as_draws_df(x, ...)

## S3 method for class 'draws_list'
as_draws_df(x, ...)

## S3 method for class 'draws_rvars'
as_draws_df(x, ...)

## S3 method for class 'mcmc'
as_draws_df(x, ...)

## S3 method for class 'mcmc.list'
as_draws_df(x, ...)

draws_df(..., .nchains = 1)

is_draws_df(x)

```

**Arguments**

x	An object to convert to a draws_df object.
...	For as_draws_df(): Arguments passed to individual methods (if applicable). For draws_df(): Named arguments containing numeric vectors each defining a separate variable.
.nchains	(positive integer) The number of chains. The default is 1.

**Details**

Objects of class "draws\_df" are [tibble](#) data frames. They have one column per variable as well as additional metadata columns ".iteration", ".chain", and ".draw". The difference between the ".iteration" and ".draw" columns is that the former is relative to the MCMC chain while the latter ignores the chain information and has all unique values. See **Examples**.

If a data.frame-like object is supplied to `as_draws_df` that contains columns named `".iteration"` or `".chain"`, they will be treated as iteration and chain indices, respectively. See **Examples**.

### Value

A `draws_df` object, which has classes `c("draws_df", "draws", class(tibble::tibble()))`.

### See Also

Other formats: [draws\\_array\(\)](#), [draws\\_list\(\)](#), [draws\\_matrix\(\)](#), [draws\\_rvars\(\)](#), [draws](#)

### Examples

```
x1 <- as_draws_df(example_draws())
class(x1)
print(x1)
str(x1)

x2 <- draws_df(a = rnorm(10), b = rnorm(10), c = 1)
class(x2)
print(x2)
str(x2)

# the difference between iteration and draw is clearer when contrasting
# the head and tail of the data frame
print(head(x1, reserved = TRUE, max_variables = 2))
print(tail(x1, reserved = TRUE, max_variables = 2))

# manually supply chain information
xnew <- data.frame(mu = rnorm(10), .chain = rep(1:2, each = 5))
xnew <- as_draws_df(xnew)
print(xnew)
```

---

draws\_list

*The draws\_list format*

---

### Description

The `as_draws_list()` methods convert objects to the `draws_list` format. The `draws_list()` function creates an object of the `draws_list` format based on a set of numeric vectors. See **Details**.

### Usage

```
as_draws_list(x, ...)

## Default S3 method:
as_draws_list(x, ...)
```

```

## S3 method for class 'draws_list'
as_draws_list(x, ...)

## S3 method for class 'draws_matrix'
as_draws_list(x, ...)

## S3 method for class 'draws_array'
as_draws_list(x, ...)

## S3 method for class 'draws_df'
as_draws_list(x, ...)

## S3 method for class 'draws_rvars'
as_draws_list(x, ...)

## S3 method for class 'mcmc'
as_draws_list(x, ...)

## S3 method for class 'mcmc.list'
as_draws_list(x, ...)

draws_list(..., .nchains = 1)

is_draws_list(x)

```

### Arguments

<code>x</code>	An object to convert to a <code>draws_list</code> object.
<code>...</code>	For <code>as_draws_list()</code> : Arguments passed to individual methods (if applicable). For <code>draws_list()</code> : Named arguments containing numeric vectors each defining a separate variable.
<code>.nchains</code>	(positive integer) The number of chains. The default is 1.

### Details

Objects of class "draws\_list" are lists with one element per MCMC chain. Each of these elements is itself a named list of numeric vectors with one vector per variable. The length of each vector is equal to the number of saved iterations per chain. See **Examples**.

### Value

A `draws_list` object, which has classes `c("draws_list", "draws", "list")`.

### See Also

Other formats: [draws\\_array\(\)](#), [draws\\_df\(\)](#), [draws\\_matrix\(\)](#), [draws\\_rvars\(\)](#), [draws](#)

## Examples

```
x1 <- as_draws_list(example_draws())
class(x1)
print(x1)
str(x1)

x2 <- draws_list(a = rnorm(10), b = rnorm(10), c = 1)
class(x2)
print(x2)
str(x2)
```

---

draws_matrix	<i>The draws_matrix format</i>
--------------	--------------------------------

---

## Description

The `as_draws_matrix()` methods convert objects to the `draws_matrix` format. The `draws_matrix()` function creates an object of the `draws_matrix` format based on a set of numeric vectors. See **Details**.

## Usage

```
as_draws_matrix(x, ...)

## Default S3 method:
as_draws_matrix(x, ...)

## S3 method for class 'draws_matrix'
as_draws_matrix(x, ...)

## S3 method for class 'draws_array'
as_draws_matrix(x, ...)

## S3 method for class 'draws_df'
as_draws_matrix(x, ...)

## S3 method for class 'draws_list'
as_draws_matrix(x, ...)

## S3 method for class 'draws_rvars'
as_draws_matrix(x, ...)

## S3 method for class 'mcmc'
as_draws_matrix(x, ...)

## S3 method for class 'mcmc.list'
as_draws_matrix(x, ...)
```

```
draws_matrix(..., .nchains = 1)
```

```
is_draws_matrix(x)
```

### Arguments

**x** An object to convert to a `draws_matrix` object.

**...** For `as_draws_matrix()`: Arguments passed to individual methods (if applicable). For `draws_matrix()`: Named arguments containing numeric vectors each defining a separate variable.

**.nchains** (positive integer) The number of chains. The default is 1.

### Details

Objects of class "draws\_matrix" are matrices (2-D arrays) with dimensions "draw" and "variable". See **Examples**.

### Value

A `draws_matrix` object, which has classes `c("draws_matrix", "draws", "matrix")`.

### See Also

Other formats: [draws\\_array\(\)](#), [draws\\_df\(\)](#), [draws\\_list\(\)](#), [draws\\_rvars\(\)](#), [draws](#)

### Examples

```
x1 <- as_draws_matrix(example_draws())
class(x1)
print(x1)
str(x1)

x2 <- draws_matrix(a = rnorm(10), b = rnorm(10), c = 1)
class(x2)
print(x2)
str(x2)
```

---

draws\_of

*Get/set array of draws underlying a random variable*

---

### Description

Gets/sets the array-representation that backs an [rvar](#). Should be used rarely.



**Usage**

```
draws_of(x, with_chains = FALSE)
```

```
draws_of(x, with_chains = FALSE) <- value
```

**Arguments**

**x** (rvar) An [rvar](#) object.

**with\_chains** (logical) Should the array of draws include a dimension for chains? If FALSE (the default), chains are not included and the array has dimension  $c(\text{ndraws}(x), \text{dim}(x))$ . If TRUE, chains are included and the array has dimension  $c(\text{niterations}(x), \text{nchains}(x), \text{dim}(x))$ .

**value** (array) An array of values to use as the backing array of **x**.

**Details**

While [rvars](#) implement fast versions of basic math operations (including [matrix multiplication](#)), sometimes you may need to bypass the [rvar](#) abstraction to do what you need to do more efficiently. `draws_of()` allows you to get / set the underlying array of draws in order to do that.

[rvars](#) represent draws internally using arrays of arbitrary dimension, which is returned by `draws_of(x)` and can be set using `draws_of(x) <- value`. The **first** dimension of these arrays is the index of the draws. If `with_chains = TRUE`, then the dimensions of the returned array are modified so that the first dimension is the index of the iterations and the second dimension is the index of the chains.

**Value**

If `with_chains = FALSE`, an array with dimensions  $c(\text{ndraws}(x), \text{dim}(x))$ .

If `with_chains = TRUE`, an array with dimensions  $c(\text{niterations}(x), \text{nchains}(x), \text{dim}(x))$ .

**Examples**

```
x <- rvar(1:10, nchains = 2)
x

# draws_of() without arguments will return the array of draws without
# chain information (first dimension is draw)
draws_of(x)

# draws_of() with with_chains = TRUE will reshape the returned array to
# include chain information in the second dimension
draws_of(x, with_chains = TRUE)

# you can also set draws using draws_of(). When with_chains = FALSE the
# existing chain information will be retained ...
draws_of(x) <- 2:11
x

# when with_chains = TRUE the chain information will be set by the
```

```
# second dimension of the assigned array
draws_of(x, with_chains = TRUE) <- array(2:11, dim = c(2,5))
x
```

---

draws\_rvars

*The draws\_rvars format*


---

## Description

The `as_draws_rvars()` methods convert objects to the `draws_rvars` format. The `draws_rvars()` function creates an object of the `draws_rvars` format based on a set of numeric vectors. See **Details**.

## Usage

```
as_draws_rvars(x, ...)

## Default S3 method:
as_draws_rvars(x, ...)

## S3 method for class 'draws_rvars'
as_draws_rvars(x, ...)

## S3 method for class 'list'
as_draws_rvars(x, ...)

## S3 method for class 'draws_matrix'
as_draws_rvars(x, ...)

## S3 method for class 'draws_array'
as_draws_rvars(x, ...)

## S3 method for class 'draws_df'
as_draws_rvars(x, ...)

## S3 method for class 'draws_list'
as_draws_rvars(x, ...)

## S3 method for class 'mcmc'
as_draws_rvars(x, ...)

## S3 method for class 'mcmc.list'
as_draws_rvars(x, ...)

draws_rvars(..., .nchains = 1)

is_draws_rvars(x)
```

**Arguments**

<code>x</code>	An object to convert to a <code>draws_rvars</code> object.
<code>...</code>	For <code>as_draws_rvars()</code> : Arguments passed to individual methods (if applicable). For <code>draws_rvars()</code> : Named arguments containing numeric vectors each defining a separate variable.
<code>.nchains</code>	(positive integer) The number of chains. The default is 1.

**Details**

Objects of class "draws\_rvars" are lists of `rvar` objects. See **Examples**.

**Value**

A `draws_rvars` object, which has classes `c("draws_rvars", "draws", "list")`.

**See Also**

Other formats: [draws\\_array\(\)](#), [draws\\_df\(\)](#), [draws\\_list\(\)](#), [draws\\_matrix\(\)](#), [draws](#)

**Examples**

```
x1 <- as_draws_rvars(example_draws())
class(x1)
print(x1)
str(x1)

x2 <- draws_rvars(a = rnorm(10), b = rnorm(10), c = 1)
class(x2)
print(x2)
str(x2)
```

---

draws\_summary

*Summaries of draws objects*


---

**Description**

The `summarise_draws()` (and `summarize_draws()`) methods provide a quick way to get a table of summary statistics and diagnostics. These methods will convert an object to a `draws` object if it isn't already. For convenience, a `summary()` method for `draws` and `rvar` objects are also provided as an alias for `summarise_draws()` if the input object is a `draws` or `rvar` object.

**Usage**

```

summarise_draws(.x, ...)

summarize_draws(.x, ...)

## S3 method for class 'draws'
summarise_draws(.x, ..., .args = list(), .cores = 1)

## S3 method for class 'draws'
summary(object, ...)

## S3 method for class 'rvar'
summarise_draws(.x, ...)

## S3 method for class 'rvar'
summary(object, ...)

default_summary_measures()

default_convergence_measures()

default_mcse_measures()

```

**Arguments**

<code>.x, object</code>	(draws) A draws object or one coercible to a draws object.
<code>...</code>	Name-value pairs of summary or <a href="#">diagnostic</a> functions. The provided names will be used as the names of the columns in the result <i>unless</i> the function returns a named vector, in which case the latter names are used. The functions can be specified in any format supported by <a href="#">as_function()</a> . See <b>Examples</b> .
<code>.args</code>	(named list) Optional arguments passed to the summary functions.
<code>.cores</code>	(positive integer) The number of cores to use for computing summaries for different variables in parallel. Coerced to integer if possible, otherwise errors. The default is <code>.cores = 1</code> , in which case no parallelization is implemented. By default, a socket cluster is used on Windows and forks otherwise.

**Details**

The default summary functions used are the ones specified by `default_summary_measures()` and `default_convergence_measures()`:

```
default_summary_measures()
```

- [mean\(\)](#)
- [median\(\)](#)
- [sd\(\)](#)
- [mad\(\)](#)

- [quantile2\(\)](#)

`default_convergence_measures()`

- [rhat\(\)](#)
- [ess\\_bulk\(\)](#)
- [ess\\_tail\(\)](#)

The `var()` function should not be used to compute variances due to its inconsistent behavior with matrices. Instead, please use `distributional::variance()`.

### Value

The `summarise_draws()` methods return a [tibble](#) data frame. The first column ("variable") contains the variable names and the remaining columns contain summary statistics and diagnostics.

The functions `default_summary_measures()`, `default_convergence_measures()`, and `default_mcse_measures()` return character vectors of names of the default measures.

### See Also

[diagnostics](#) for a list of available diagnostics and links to their individual help pages.

### Examples

```
x <- example_draws("eight_schools")
class(x)
str(x)

summarise_draws(x)
summarise_draws(x, "mean", "median")
summarise_draws(x, mean, mcse = mcse_mean)
summarise_draws(x, ~quantile(.x, probs = c(0.4, 0.6)))

# using default_*_measures()
summarise_draws(x, default_summary_measures())
summarise_draws(x, default_convergence_measures())
summarise_draws(x, default_mcse_measures())

# compute variance of variables
summarise_draws(x, var = distributional::variance)

# illustrate use of '.args'
ws <- rexp(ndraws(x))
summarise_draws(x, weighted.mean, .args = list(w = ws))
```

---

 ess\_basic

*Basic version of the effective sample size*


---

## Description

Compute the basic effective sample size (ESS) estimate for a single variable as described in Gelman et al. (2013) with some changes according to Vehtari et al. (2021). For practical applications, we strongly recommend the improved ESS convergence diagnostics implemented in [ess\\_bulk\(\)](#) and [ess\\_tail\(\)](#). See Vehtari (2021) for an in-depth comparison of different effective sample size estimators.

## Usage

```
ess_basic(x, ...)

## Default S3 method:
ess_basic(x, split = TRUE, ...)

## S3 method for class 'rvar'
ess_basic(x, split = TRUE, ...)
```

## Arguments

<code>x</code>	(multiple options) One of: <ul style="list-style-type: none"> <li>• A matrix of draws for a single variable (iterations x chains). See <a href="#">extract_variable_matrix()</a>.</li> <li>• An <a href="#">rvar</a>.</li> </ul>
<code>...</code>	Arguments passed to individual methods (if applicable).
<code>split</code>	(logical) Should the estimate be computed on split chains? The default is TRUE.

## Value

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#), returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function.

## References

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari and Donald B. Rubin (2013). *Bayesian Data Analysis, Third Edition*. Chapman and Hall/CRC.

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

Aki Vehtari (2021). Comparison of MCMC effective sample size estimators. Retrieved from [https://avehtari.github.io/rhat\\_ess/ess\\_comparison.html](https://avehtari.github.io/rhat_ess/ess_comparison.html)

## See Also

Other diagnostics: [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [mcse\\_sd\(\)](#), [rhat\\_basic\(\)](#), [rhat\(\)](#), [rstar\(\)](#)

## Examples

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_basic(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
ess_basic(d$Sigma)
```

---

ess\_bulk

*Bulk effective sample size (bulk-ESS)*

---

## Description

Compute a bulk effective sample size estimate (bulk-ESS) for a single variable. Bulk-ESS is useful as a diagnostic for the sampling efficiency in the bulk of the posterior. It is defined as the effective sample size for rank normalized values using split chains. For the tail effective sample size see [ess\\_tail\(\)](#). See Vehtari (2021) for an in-depth comparison of different effective sample size estimators.

## Usage

```
ess_bulk(x, ...)

## Default S3 method:
ess_bulk(x, ...)

## S3 method for class 'rvar'
ess_bulk(x, ...)
```

## Arguments

- x (multiple options) One of:
- A matrix of draws for a single variable (iterations x chains). See `extract_variable_matrix()`.
  - An `rvar`.
- ... Arguments passed to individual methods (if applicable).

## Value

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an `rvar`, returns an array of the same dimensions as the `rvar`, where each element is equal to the value that would be returned by passing the draws array for that element of the `rvar` to this function.

## References

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

Aki Vehtari (2021). Comparison of MCMC effective sample size estimators. Retrieved from [https://avehtari.github.io/rhat\\_ess/ess\\_comparison.html](https://avehtari.github.io/rhat_ess/ess_comparison.html)

## See Also

Other diagnostics: `ess_basic()`, `ess_quantile()`, `ess_sd()`, `ess_tail()`, `mcse_mean()`, `mcse_quantile()`, `mcse_sd()`, `rhat_basic()`, `rhat()`, `rstar()`

## Examples

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_bulk(mu)
```

```
d <- as_draws_rvars(example_draws("multi_normal"))
ess_bulk(d$Sigma)
```



---

ess_mean	<i>Effective sample size for the mean</i>
----------	---

---

### Description

Compute an effective sample size estimate for a mean (expectation) estimate of a single variable.

### Usage

```
ess_mean(x, ...)

## S3 method for class 'rvar'
ess_mean(x, ...)
```

### Arguments

**x** (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
- An [rvar](#).

**...** Arguments passed to individual methods (if applicable).

### Value

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#), returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function.

### References

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari and Donald B. Rubin (2013). *Bayesian Data Analysis, Third Edition*. Chapman and Hall/CRC.

### Examples

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_mean(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
ess_mean(d$Sigma)
```

ess\_quantile

*Effective sample sizes for quantiles***Description**

Compute effective sample size estimates for quantile estimates of a single variable.

**Usage**

```
ess_quantile(x, probs = c(0.05, 0.95), ...)

## Default S3 method:
ess_quantile(x, probs = c(0.05, 0.95), names = TRUE, ...)

## S3 method for class 'rvar'
ess_quantile(x, probs = c(0.05, 0.95), names = TRUE, ...)

ess_median(x, ...)

## Default S3 method:
ess_mean(x, ...)
```

**Arguments**

x	(multiple options) One of: <ul style="list-style-type: none"> <li>• A matrix of draws for a single variable (iterations x chains). See <a href="#">extract_variable_matrix()</a>.</li> <li>• An <a href="#">rvar</a>.</li> </ul>
probs	(numeric vector) Probabilities in $[0, 1]$ .
...	Arguments passed to individual methods (if applicable).
names	(logical) Should the result have a names attribute? The default is TRUE, but use FALSE for improved speed if there are many values in probs.

**Value**

If the input is an array, returns a numeric vector with one element per quantile. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be a vector of (numeric) NA values. Also, if all draws of a variable are the same (constant), the returned output will be a vector of (numeric) NA values as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#) and `length(probs) == 1`, returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function. If `length(probs) > 1`, the first dimension of the result indexes the input probabilities; i.e. the result has dimension `c(length(probs), dim(x))`.

## References

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

## See Also

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [mcse\\_sd\(\)](#), [rhat\\_basic\(\)](#), [rhat\(\)](#), [rstar\(\)](#)

## Examples

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_quantile(mu, probs = c(0.1, 0.9))

d <- as_draws_rvars(example_draws("multi_normal"))
ess_quantile(d$mu, probs = c(0.1, 0.9))
```

---

 ess\_sd

*Effective sample size for the standard deviation*


---

## Description

Compute an effective sample size estimate for the standard deviation (SD) estimate of a single variable. This is defined as the effective sample size estimate for the absolute deviation from mean.

## Usage

```
ess_sd(x, ...)

## Default S3 method:
ess_sd(x, ...)

## S3 method for class 'rvar'
ess_sd(x, ...)
```

## Arguments

**x** (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
- An [rvar](#).

**...** Arguments passed to individual methods (if applicable).

**Value**

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an `rvar`, returns an array of the same dimensions as the `rvar`, where each element is equal to the value that would be returned by passing the draws array for that element of the `rvar` to this function.

**References**

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

**See Also**

Other diagnostics: `ess_basic()`, `ess_bulk()`, `ess_quantile()`, `ess_tail()`, `mcse_mean()`, `mcse_quantile()`, `mcse_sd()`, `rhat_basic()`, `rhat()`, `rstar()`

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_sd(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
ess_sd(d$Sigma)
```

---

ess\_tail

*Tail effective sample size (tail-ESS)*


---

**Description**

Compute a tail effective sample size estimate (tail-ESS) for a single variable. Tail-ESS is useful as a diagnostic for the sampling efficiency in the tails of the posterior. It is defined as the minimum of the effective sample sizes for 5% and 95% quantiles. For the bulk effective sample size see `ess_bulk()`. See Vehtari (2021) for an in-depth comparison of different effective sample size estimators.

**Usage**

```
ess_tail(x, ...)

## Default S3 method:
ess_tail(x, ...)

## S3 method for class 'rvar'
ess_tail(x, ...)
```

**Arguments**

**x** (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See `extract_variable_matrix()`.
- An `rvar`.

**...** Arguments passed to individual methods (if applicable).

**Value**

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an `rvar`, returns an array of the same dimensions as the `rvar`, where each element is equal to the value that would be returned by passing the draws array for that element of the `rvar` to this function.

**References**

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

Aki Vehtari (2021). Comparison of MCMC effective sample size estimators. Retrieved from [https://avehtari.github.io/rhat\\_ess/ess\\_comparison.html](https://avehtari.github.io/rhat_ess/ess_comparison.html)

**See Also**

Other diagnostics: `ess_basic()`, `ess_bulk()`, `ess_quantile()`, `ess_sd()`, `mcse_mean()`, `mcse_quantile()`, `mcse_sd()`, `rhat_basic()`, `rhat()`, `rstar()`

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
ess_tail(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
```

```
ess_tail(d$Sigma)
```

---

example_draws	<i>Example draws objects</i>
---------------	------------------------------

---

## Description

Objects for use in examples, vignettes, and tests.

## Usage

```
example_draws(example = "eight_schools")
```

## Arguments

`example` (string) The name of the example draws object. See **Details** for available options.

## Details

The following example draws objects are available.

**eight\_schools**: A `draws_array` object with 100 iterations from each of 4 Markov chains obtained by fitting the eight schools model described in Gelman et al. (2013) with **Stan**. The variables are:

- `mu`: Overall mean of the eight schools
- `tau`: Standard deviation between schools
- `theta`: Individual means of each of the eight schools

**multi\_normal**: A `draws_array` object with 100 iterations from each of the 4 Markov chains obtained by fitting a 3-dimensional multivariate normal model to 100 simulated observations. The variables are:

- `mu`: Mean parameter vector of length 3
- `Sigma`: Covariance matrix of dimension 3 x 3

## Value

A draws object.

## Note

These objects are only intended to be used in demonstrations and tests. They contain fewer iterations and chains than recommended for performing actual inference.

## References

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari and Donald B. Rubin (2013). Bayesian Data Analysis, Third Edition. Chapman and Hall/CRC.

**Examples**

```
draws_eight_schools <- example_draws("eight_schools")
summarise_draws(draws_eight_schools)

draws_multi_normal <- example_draws("multi_normal")
summarise_draws(draws_multi_normal)
```

---

extract_variable	<i>Extract draws of a single variable</i>
------------------	---

---

**Description**

Extract a vector of draws of a single variable.

**Usage**

```
extract_variable(x, variable, ...)

## Default S3 method:
extract_variable(x, variable, ...)

## S3 method for class 'draws'
extract_variable(x, variable, ...)

## S3 method for class 'draws_rvars'
extract_variable(x, variable, ...)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
variable	(string) The name of the variable to extract.
...	Arguments passed to individual methods (if applicable).

**Value**

A numeric vector of length equal to the number of draws.

**Examples**

```
x <- example_draws()
mu <- extract_variable(x, variable = "mu")
str(mu)
```

---

`extract_variable_matrix`*Extract matrix of a single variable*

---

### Description

Extract an iterations x chains matrix of draws of a single variable. This is primarily used for convergence diagnostic functions such as `rhat()`.

### Usage

```
extract_variable_matrix(x, variable, ...)  
  
## Default S3 method:  
extract_variable_matrix(x, variable, ...)  
  
## S3 method for class 'draws'  
extract_variable_matrix(x, variable, ...)  
  
## S3 method for class 'draws_rvars'  
extract_variable_matrix(x, variable, ...)
```

### Arguments

<code>x</code>	(draws) A draws object or another R object for which the method is defined.
<code>variable</code>	(string) The name of the variable to extract.
<code>...</code>	Arguments passed to individual methods (if applicable).

### Value

A matrix with dimension iterations x chains.

### Examples

```
x <- example_draws()  
mu <- extract_variable_matrix(x, variable = "mu")  
dim(mu)  
rhat(mu)
```



---

is_rvar	<i>Is x a random variable?</i>
---------	--------------------------------

---

**Description**

Test if x is an [rvar](#).

**Usage**

```
is_rvar(x)
```

**Arguments**

x (any object) An object to test.

**Value**

TRUE if x is an [rvar](#), FALSE otherwise.

**See Also**

[as\\_rvar\(\)](#) to convert objects to rvars.

---

mcse_mean	<i>Monte Carlo standard error for the mean</i>
-----------	--

---

**Description**

Compute the Monte Carlo standard error for the mean (expectation) of a single variable.

**Usage**

```
mcse_mean(x, ...)
```

```
## Default S3 method:
```

```
mcse_mean(x, ...)
```

```
## S3 method for class 'rvar'
```

```
mcse_mean(x, ...)
```

**Arguments**

x (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
- An [rvar](#).

... Arguments passed to individual methods (if applicable).

**Value**

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an `rvar`, returns an array of the same dimensions as the `rvar`, where each element is equal to the value that would be returned by passing the draws array for that element of the `rvar` to this function.

**References**

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari and Donald B. Rubin (2013). *Bayesian Data Analysis, Third Edition*. Chapman and Hall/CRC.

**See Also**

Other diagnostics: `ess_basic()`, `ess_bulk()`, `ess_quantile()`, `ess_sd()`, `ess_tail()`, `mcse_quantile()`, `mcse_sd()`, `rhat_basic()`, `rhat()`, `rstar()`

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
mcse_mean(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
mcse_mean(d$Sigma)
```

---

mcse\_quantile

*Monte Carlo standard error for quantiles*

---

**Description**

Compute Monte Carlo standard errors for quantile estimates of a single variable.

**Usage**

```
mcse_quantile(x, probs = c(0.05, 0.95), ...)

## Default S3 method:
mcse_quantile(x, probs = c(0.05, 0.95), names = TRUE, ...)

## S3 method for class 'rvar'
mcse_quantile(x, probs = c(0.05, 0.95), names = TRUE, ...)

mcse_median(x, ...)
```

**Arguments**

x	(multiple options) One of: <ul style="list-style-type: none"> <li>• A matrix of draws for a single variable (iterations x chains). See <a href="#">extract_variable_matrix()</a>.</li> <li>• An <a href="#">rvar</a>.</li> </ul>
probs	(numeric vector) Probabilities in $[\emptyset, 1]$ .
...	Arguments passed to individual methods (if applicable).
names	(logical) Should the result have a names attribute? The default is TRUE, but use FALSE for improved speed if there are many values in probs.

**Value**

If the input is an array, returns a numeric vector with one element per quantile. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be a vector of (numeric) NA values. Also, if all draws of a variable are the same (constant), the returned output will be a vector of (numeric) NA values as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#) and `length(probs) == 1`, returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function. If `length(probs) > 1`, the first dimension of the result indexes the input probabilities; i.e. the result has dimension `c(length(probs), dim(x))`.

**References**

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

**See Also**

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_sd\(\)](#), [rhat\\_basic\(\)](#), [rhat\(\)](#), [rstar\(\)](#)

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
mcse_quantile(mu, probs = c(0.1, 0.9))

d <- as_draws_rvars(example_draws("multi_normal"))
mcse_quantile(d$mu)
```

---

 mcse\_sd

*Monte Carlo standard error for the standard deviation*


---

## Description

Compute the Monte Carlo standard error for the standard deviation (SD) of a single variable without assuming normality using moments of moments and first order Taylor series approximation (Kenney and Keeping, 1951, p. 141).

## Usage

```
mcse_sd(x, ...)
```

```
## Default S3 method:
mcse_sd(x, ...)
```

```
## S3 method for class 'rvar'
mcse_sd(x, ...)
```

## Arguments

`x` (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
- An [rvar](#).

`...` Arguments passed to individual methods (if applicable).

## Value

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#), returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function.

## References

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

J. F. Kenney & E. S. Keeping (1951). *Mathematics of Statistics, Vol. II*.

**See Also**

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [rhat\\_basic\(\)](#), [rhat\(\)](#), [rstar\(\)](#)

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
mcse_sd(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
mcse_sd(d$Sigma)
```

---

merge_chains	<i>Merge chains of draws objects</i>
--------------	--------------------------------------

---

**Description**

Merge chains of [draws](#) objects into a single chain. Some operations will trigger an automatic merging of chains, for example, because chains do not match between two objects involved in a binary operation. By default, no warning will be issued when this happens but you can activate one via `options(posterior.warn_on_merge_chains = TRUE)`.

**Usage**

```
merge_chains(x, ...)
```

## S3 method for class 'draws\_matrix'

```
merge_chains(x, ...)
```

## S3 method for class 'draws\_array'

```
merge_chains(x, ...)
```

## S3 method for class 'draws\_df'

```
merge_chains(x, ...)
```

## S3 method for class 'draws\_list'

```
merge_chains(x, ...)
```

## S3 method for class 'rvar'

```
merge_chains(x, ...)
```

## S3 method for class 'draws\_rvars'

```
merge_chains(x, ...)
```

**Arguments**

`x` (draws) A draws object or another R object for which the method is defined.  
`...` Arguments passed to individual methods (if applicable).

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- example_draws()

# draws_array with 4 chains, 100 iters each
str(x)

# draws_array with 1 chain of 400 iterations
str(merge_chains(x))
```

---

mutate\_variables      *Mutate variables in draws objects*

---

**Description**

Mutate variables in a [draws](#) object.

**Usage**

```
mutate_variables(.x, ...)
```

## S3 method for class 'draws\_matrix'  
mutate\_variables(.x, ...)

## S3 method for class 'draws\_array'  
mutate\_variables(.x, ...)

## S3 method for class 'draws\_df'  
mutate\_variables(.x, ...)

## S3 method for class 'draws\_list'  
mutate\_variables(.x, ...)

## S3 method for class 'draws\_rvars'  
mutate\_variables(.x, ...)

**Arguments**

`.x` (draws) A [draws](#) object.

`...` Name-value pairs of expressions, each with either length 1 or the same length as in the entire input (i.e., number of iterations or draws). The name of each argument will be the name of a new variable, and the value will be its corresponding value. Use a NULL value in `mutate_variables` to drop a variable. New variables overwrite existing variables of the same name.

**Details**

In order to mutate variables in [draws\\_matrix](#) and [draws\\_array](#) objects, they are transformed to [draws\\_df](#) objects first and then transformed back after mutation. As those transformations are quite expensive for larger number of draws, we recommend using `mutate_variables` on [draws\\_df](#) and [draws\\_list](#) objects if speed is an issue.

In [draws\\_rvars](#) objects, the output of each expression in `...` is coerced to an [rvar](#) object if it is not already one using `as_rvar()`.

**Value**

Returns a [draws](#) object of the same format as `.x`, with variables mutated according to the expressions provided in `...`

**See Also**

[variables](#), [rename\\_variables](#)

**Examples**

```
x <- as_draws_df(example_draws())
x <- subset(x, variable = c("mu", "tau"))

mutate_variables(x, tau2 = tau^2)
mutate_variables(x, scale = 1.96 * tau, lower = mu - scale)
```

---

order\_draws

*Order draws objects*

---

**Description**

Order [draws](#) objects according to iteration and chain number. By default, draws objects are ordered but subsetting or extracting parts of them may leave them in an unordered state.

**Usage**

```
order_draws(x, ...)  
  
## S3 method for class 'draws_matrix'  
order_draws(x, ...)  
  
## S3 method for class 'draws_array'  
order_draws(x, ...)  
  
## S3 method for class 'draws_df'  
order_draws(x, ...)  
  
## S3 method for class 'draws_list'  
order_draws(x, ...)  
  
## S3 method for class 'draws_rvars'  
order_draws(x, ...)  
  
## S3 method for class 'rvar'  
order_draws(x, ...)
```

**Arguments**

x (draws) A draws object or another R object for which the method is defined.  
... Arguments passed to individual methods (if applicable).

**Value**

A draws object of the same class as x.

**See Also**

[repair\\_draws\(\)](#)

**Examples**

```
x <- as_draws_array(example_draws())  
dimnames(x[10:5, 4:3, ])  
dimnames(order_draws(x[10:5, 4:3, ]))
```

---

print.draws\_array      *Print draws\_array objects*

---

**Description**

Pretty printing for [draws\\_array](#) objects.



**Usage**

```
## S3 method for class 'draws_array'
print(
  x,
  digits = 2,
  max_iterations = getOption("posterior.max_iterations", 5),
  max_chains = getOption("posterior.max_chains", 8),
  max_variables = getOption("posterior.max_variables", 4),
  reserved = FALSE,
  ...
)
```

**Arguments**

<code>x</code>	(draws) A draws object or another R object for which the method is defined.
<code>digits</code>	(nonnegative integer) The minimum number of significant digits to print.
<code>max_iterations</code>	(positive integer) The maximum number of iterations to print. Can be controlled globally via the "posterior.max_iterations" <a href="#">option</a> .
<code>max_chains</code>	(positive integer) The maximum number of chains to print. Can be controlled globally via the "posterior.max_chains" <a href="#">option</a> .
<code>max_variables</code>	(positive integer) The maximum number of variables to print. Can be controlled globally via the "posterior.max_variables" <a href="#">option</a> .
<code>reserved</code>	(logical) Should reserved variables be included in the output? Defaults to FALSE. See <a href="#">reserved_variables</a> for an overview of currently reserved variable names.
<code>...</code>	Further arguments passed to the underlying <code>print()</code> methods.

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- as_draws_array(example_draws())
print(x)
```

---

<code>print.draws_df</code>	<i>Print draws_df objects</i>
-----------------------------	-------------------------------

---

**Description**

Pretty printing for `draws_df` objects.

**Usage**

```
## S3 method for class 'draws_df'
print(
  x,
  digits = 2,
  max_draws = getOption("posterior.max_draws", 10),
  max_variables = getOption("posterior.max_variables", 8),
  reserved = FALSE,
  ...
)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
digits	(nonnegative integer) The minimum number of significant digits to print.
max_draws	(positive integer) The maximum number of draws to print. Can be controlled globally via the "posterior.max_draws" <a href="#">option</a> .
max_variables	(positive integer) The maximum number of variables to print. Can be controlled globally via the "posterior.max_variables" <a href="#">option</a> .
reserved	(logical) Should reserved variables be included in the output? Defaults to FALSE. See <a href="#">reserved_variables</a> for an overview of currently reserved variable names.
...	Further arguments passed to the underlying <code>print()</code> methods.

**Value**

A draws object of the same class as x.

**Examples**

```
x <- as_draws_df(example_draws())
print(x)
```

---

print.draws_list	<i>Print draws_list objects</i>
------------------	---------------------------------

---

**Description**

Pretty printing for [draws\\_list](#) objects.

**Usage**

```
## S3 method for class 'draws_list'
print(
  x,
  digits = 2,
  max_iterations = getOption("posterior.max_iterations", 10),
  max_chains = getOption("posterior.max_chains", 2),
  max_variables = getOption("posterior.max_variables", 4),
  reserved = FALSE,
  ...
)
```

**Arguments**

<code>x</code>	(draws) A draws object or another R object for which the method is defined.
<code>digits</code>	(nonnegative integer) The minimum number of significant digits to print.
<code>max_iterations</code>	(positive integer) The maximum number of iterations to print. Can be controlled globally via the "posterior.max_iterations" <a href="#">option</a> .
<code>max_chains</code>	(positive integer) The maximum number of chains to print. Can be controlled globally via the "posterior.max_chains" <a href="#">option</a> .
<code>max_variables</code>	(positive integer) The maximum number of variables to print. Can be controlled globally via the "posterior.max_variables" <a href="#">option</a> .
<code>reserved</code>	(logical) Should reserved variables be included in the output? Defaults to FALSE. See <a href="#">reserved_variables</a> for an overview of currently reserved variable names.
<code>...</code>	Further arguments passed to the underlying <code>print()</code> methods.

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- as_draws_list(example_draws())
print(x)
```

---

`print.draws_matrix`     *Print draws\_matrix objects*

---

**Description**

Pretty printing for `draws_matrix` objects.

**Usage**

```
## S3 method for class 'draws_matrix'
print(
  x,
  digits = 2,
  max_draws = getOption("posterior.max_draws", 10),
  max_variables = getOption("posterior.max_variables", 8),
  reserved = FALSE,
  ...
)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
digits	(nonnegative integer) The minimum number of significant digits to print.
max_draws	(positive integer) The maximum number of draws to print. Can be controlled globally via the "posterior.max_draws" <a href="#">option</a> .
max_variables	(positive integer) The maximum number of variables to print. Can be controlled globally via the "posterior.max_variables" <a href="#">option</a> .
reserved	(logical) Should reserved variables be included in the output? Defaults to FALSE. See <a href="#">reserved_variables</a> for an overview of currently reserved variable names.
...	Further arguments passed to the underlying <code>print()</code> methods.

**Value**

A draws object of the same class as x.

**Examples**

```
x <- as_draws_matrix(example_draws())
print(x)
```

---

print.draws\_rvars      *Print draws\_rvars objects*

---

**Description**

Pretty printing for [draws\\_rvars](#) objects.

**Usage**

```
## S3 method for class 'draws_rvars'
print(
  x,
  digits = 2,
  max_variables = getOption("posterior.max_variables", 8),
  summary = getOption("posterior.rvar_summary", "mean_sd"),
  reserved = FALSE,
  ...
)
```

**Arguments**

<code>x</code>	(draws) A draws object or another R object for which the method is defined.
<code>digits</code>	(nonnegative integer) The minimum number of significant digits to print.
<code>max_variables</code>	(positive integer) The maximum number of variables to print. Can be controlled globally via the "posterior.max_variables" <a href="#">option</a> .
<code>summary</code>	(string) The style of summary to display: "mean_sd" displays mean±sd, "median_mad" displays median±mad. If NULL, <code>getOption("posterior.rvar_summary")</code> is used (default "mean_sd").
<code>reserved</code>	(logical) Should reserved variables be included in the output? Defaults to FALSE. See <a href="#">reserved_variables</a> for an overview of currently reserved variable names.
<code>...</code>	Further arguments passed to the underlying <code>print()</code> methods.

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- as_draws_rvars(example_draws())
print(x)
```

---

print.rvar

*Print or format a random variable*

---

**Description**

Printing and formatting methods for [rvars](#).

**Usage**

```
## S3 method for class 'rvar'
print(x, ..., summary = NULL, digits = 2, color = TRUE)

## S3 method for class 'rvar'
format(x, ..., summary = NULL, digits = 2, color = FALSE)

## S3 method for class 'rvar'
str(
  object,
  ...,
  summary = NULL,
  vec.len = NULL,
  indent.str = paste(rep.int(" ", max(0, nest.lev + 1)), collapse = ".."),
  nest.lev = 0,
  give.attr = TRUE
)
```

**Arguments**

x, object	(rvar) The <a href="#">rvar</a> to print.
...	Further arguments passed to the underlying <a href="#">print()</a> methods.
summary	(string) The style of summary to display: "mean_sd" displays mean±sd, "median_mad" displays median±mad. If NULL, <code>getOption("posterior.rvar.summary")</code> is used (default "mean_sd").
digits	(nonnegative integer) The minimum number of significant digits to print.
color	(logical) Whether or not to use color when formatting the output. If TRUE, the <a href="#">pillar::style_num()</a> functions may be used to produce strings containing control sequences to produce colored output on the terminal.
vec.len	(nonnegative integer) How many 'first few' elements are displayed of each vector. If NULL, defaults to <code>getOption("str")\$vec.len</code> , which defaults to 4.
indent.str	(string) The indentation string to use.
nest.lev	(nonnegative integer) Current nesting level in the recursive calls to <code>str()</code> .
give.attr	(logical) If TRUE (default), show attributes as sub structures.

**Details**

`print()` and `str()` print out [rvar](#) objects by summarizing each element in the random variable with either its mean±sd or median±mad, depending on the value of `summary`. Both functions use the `format()` implementation for [rvar](#) objects under the hood, which returns a character vector in the mean±sd or median±mad form.

**Value**

For `print()`, an invisible version of the input object.

For `str()`, nothing; i.e. `invisible(NULL)`.

For `format()`, a character vector of the same dimensions as `x` where each entry is of the form "mean±sd" or "median±mad", depending on the value of `summary`.

### Examples

```
set.seed(5678)
x = rbind(
  cbind(rvar(rnorm(1000, 1)), rvar(rnorm(1000, 2))),
  cbind(rvar(rnorm(1000, 3)), rvar(rnorm(1000, 4)))
)

print(x)
print(x, summary = "median_mad")

str(x)

format(x)
```

---

quantile2

*Compute Quantiles*

---

### Description

Compute quantiles of a sample and return them in a format consistent with other summary functions in the **posterior** package.

### Usage

```
quantile2(x, probs = c(0.05, 0.95), na.rm = FALSE, ...)

## Default S3 method:
quantile2(x, probs = c(0.05, 0.95), na.rm = FALSE, names = TRUE, ...)

## S3 method for class 'rvar'
quantile2(x, probs = c(0.05, 0.95), na.rm = FALSE, names = TRUE, ...)
```

### Arguments

<code>x</code>	(multiple options) One of: <ul style="list-style-type: none"> <li>• A matrix of draws for a single variable (iterations x chains). See <code>extract_variable_matrix()</code>.</li> <li>• An <code>rvar</code>.</li> </ul>
<code>probs</code>	(numeric vector) Probabilities in $[0, 1]$ .
<code>na.rm</code>	(logical) Should NA and NaN values be removed from <code>x</code> prior to computing quantiles? The default is FALSE.
<code>...</code>	Arguments passed to individual methods (if applicable) and then on to <code>stats::quantile()</code> .
<code>names</code>	(logical) Should the result have a <code>names</code> attribute? The default is TRUE, but use FALSE for improved speed if there are many values in <code>probs</code> .

**Value**

A numeric vector of length `length(probs)`. If `names = TRUE`, it has a `names` attribute with names like "q5", "q95", etc, based on the values of `probs`.

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
quantile2(mu)
```

---

rdo

---

*Execute expressions of random variables*


---

**Description**

Execute (nearly) arbitrary R expressions that may include `rvars`, producing a new `rvar`.

**Usage**

```
rdo(expr, dim = NULL, ndraws = NULL)
```

**Arguments**

<code>expr</code>	(expression) A bare expression that can (optionally) contain <code>rvars</code> . The expression supports <a href="#">quasiquote</a> .
<code>dim</code>	(integer vector) One or more integers giving the maximal indices in each dimension to override the dimensions of the <code>rvar</code> to be created (see <code>dim()</code> ). If <code>NULL</code> (the default), <code>dim</code> is determined by the input. <b>NOTE:</b> This argument controls the dimensions of the <code>rvar</code> , not the underlying array, so you cannot change the number of draws using this argument.
<code>ndraws</code>	(positive integer) The number of draws used to construct new random variables if no <code>rvars</code> are supplied in <code>expr</code> . If <code>NULL</code> , <code>getOption("posterior.rvar.ndraws")</code> is used (default 4000). If <code>expr</code> contains <code>rvars</code> , the number of draws in the provided <code>rvars</code> is used instead of the value of this argument.

**Details**

This function evaluates `expr` possibly multiple times, once for each draw of the `rvars` it contains, then returns a new `rvar` representing the output of those expressions. To identify `rvars`, `rdo()` searches the calling environment for any variables named in `expr` for which `is_rvar()` evaluates to `TRUE`. If `expr` contains no `rvars`, then it will be executed `ndraws` times and an `rvar` with that many draws returned.

`rdo()` is not necessarily *fast* (in fact in some cases it may be very slow), but it has the advantage of allowing a nearly arbitrary R expression to be executed against `rvars` simply by wrapping it with `rdo( ... )`. This makes it especially useful as a prototyping tool. If you create code with `rdo()` and it is unacceptably slow for your application, consider rewriting it using math operations directly on `rvars` (which should be fast), using `rvar_rng()`, and/or using operations directly on the arrays that back the `rvars` (via `draws_of()`).



**Value**

An [rvar](#).

**See Also**

Other rfun: [rfun\(\)](#), [rvar\\_rng\(\)](#)

**Examples**

```
mu <- rdo(rnorm(10, mean = 1:10, sd = 1))
sigma <- rdo(rgamma(1, shape = 1, rate = 1))
x <- rdo(rnorm(10, mu, sigma))
x
```

---

rename_variables	<i>Rename variables in draws objects</i>
------------------	--

---

**Description**

Rename variables in a [draws](#) object.

**Usage**

```
rename_variables(.x, ...)

## S3 method for class 'draws'
rename_variables(.x, ...)
```

**Arguments**

<code>.x</code>	(draws) A <a href="#">draws</a> object.
<code>...</code>	One or more expressions, separated by commas, indicating the variables to rename. The variable names can be unquoted ( <code>new_name = old_name</code> ) or quoted ( <code>"new_name" = "old_name"</code> ). For non-scalar variables, all elements can be renamed together ( <code>"new_name" = "old_name"</code> ) or they can be renamed individually ( <code>"new_name[1]" = "old_name[1]"</code> ).

**Value**

Returns a [draws](#) object of the same format as `.x`, with variables renamed according to the expressions provided in `...`

**See Also**

[variables](#), [mutate\\_variables](#)

**Examples**

```
x <- as_draws_df(example_draws())
variables(x)

x <- rename_variables(x, mean = mu, sigma = tau)
variables(x)

x <- rename_variables(x, b = `theta[1]`) # or b = "theta[1]"
variables(x)

# rename all elements of 'theta' at once
x <- rename_variables(x, alpha = theta)
variables(x)
```

---

 repair\_draws

*Repair indices of draws objects*


---

**Description**

Repair indices of draws objects so that iterations, chains, and draws are continuously and consistently numbered.

**Usage**

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'draws\_matrix'

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'draws\_array'

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'draws\_df'

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'draws\_list'

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'draws\_rvars'

```
repair_draws(x, order = TRUE, ...)
```

## S3 method for class 'rvar'

```
repair_draws(x, order = TRUE, ...)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
order	(logical) Should draws be ordered (via <code>order_draws()</code> ) before repairing indices? Defaults to TRUE.
...	Arguments passed to individual methods (if applicable).

**Value**

A draws object of the same class as x.

**See Also**

[order\\_draws\(\)](#)

**Examples**

```
x <- as_draws_array(example_draws())
(x <- x[10:5, 3:4, ])
repair_draws(x)
```

---

resample\_draws

*Resample draws objects*


---

**Description**

Resample [draws](#) objects according to provided weights, for example weights obtained through importance sampling.

**Usage**

```
resample_draws(x, ...)
```

```
## S3 method for class 'draws'
```

```
resample_draws(x, weights = NULL, method = "stratified", ndraws = NULL, ...)
```

**Arguments**

x	(draws) A draws object or another R object for which the method is defined.
...	Arguments passed to individual methods (if applicable).
weights	(numeric vector) A vector of positive weights of length <code>ndraws(x)</code> . The weights will be internally normalized. If <code>weights</code> is not specified, an attempt will be made to extract any weights already stored in the draws object (via <a href="#">weight_draws()</a> ). How exactly the weights influence the resampling depends on the method argument.
method	(string) The resampling method to use:

- "simple": simple random resampling with replacement
- "simple\_no\_replace": simple random resampling without replacement
- "stratified": stratified resampling with replacement
- "deterministic": deterministic resampling with replacement

Currently, "stratified" is the default as it has comparably low variance and bias with respect to ideal resampling. The latter would sample perfectly proportional to the weights, but this is not possible in practice due to the finite number of draws available. For more details about resampling methods, see Kitagawa (1996).

`ndraws` (positive integer) The number of draws to be returned. By default `ndraws` is set internally to the total number of draws in `x` if sensible.

### Details

Upon usage of `resample_draws()`, chains will automatically be merged due to subsetting of individual draws (see [subset\\_draws](#) for details). Also, weights stored in the draws object will be removed in the process, as resampling invalidates existing weights.

### Value

A draws object of the same class as `x`.

### References

Kitagawa, G., Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear ' State Space Models, *Journal of Computational and Graphical Statistics*, 5(1):1-25, 1996.

### See Also

[resample\\_draws\(\)](#)

### Examples

```
x <- as_draws_df(example_draws())

# random weights for just for demonstration
w <- runif(ndraws(x), 0, 10)

# use default stratified sampling
x_rs <- resample_draws(x, weights = w)
summarise_draws(x_rs, default_summary_measures())

# use simple random sampling
x_rs <- resample_draws(x, weights = w, method = "simple")
summarise_draws(x_rs, default_summary_measures())
```

---

reserved_variables	<i>Reserved variables</i>
--------------------	---------------------------

---

## Description

Get names of reserved variables from objects in the **posterior** package.

## Usage

```
reserved_variables(x, ...)  
  
## Default S3 method:  
reserved_variables(x, ...)  
  
## S3 method for class 'draws_matrix'  
reserved_variables(x, ...)  
  
## S3 method for class 'draws_array'  
reserved_variables(x, ...)  
  
## S3 method for class 'draws_df'  
reserved_variables(x, ...)  
  
## S3 method for class 'draws_list'  
reserved_variables(x, ...)  
  
## S3 method for class 'draws_rvars'  
reserved_variables(x, ...)
```

## Arguments

x (draws) A draws object or another R object for which the method is defined.  
... Arguments passed to individual methods (if applicable).

## Details

reserved\_variables() returns the names of reserved variables in use by an object.

The following variables names are currently reserved for special use cases in all [draws](#) formats:

- `.log_weight`: Log weights per draw (see [weight\\_draws](#)).

Further, specific for the [draws\\_df](#) format, there are three additional reserved variables:

- `.chain`: Chain index per draw
- `.iteration`: Iteration index within each chain
- `.draw`: Draw index across chains

More reserved variables may be added in the future.

**Value**

A character vector of reserved variables used in `x`.

**Examples**

```
x <- example_draws()
reserved_variables(x)

# if we add weights, the \.log_weight\ reserved variable is used
x <- weight_draws(x, rexp(ndraws(x)))
reserved_variables(x)
```

---

 rfun

---

*Create functions of random variables*


---

**Description**

Function that create functions that can accept and/or produce `rvars`.

**Usage**

```
rfun(.f, rvar_args = NULL, ndraws = NULL)
```

**Arguments**

<code>.f</code>	(multiple options) A function to turn into a function that accepts and/or produces random variables: <ul style="list-style-type: none"> <li>• A function</li> <li>• A one-sided formula that can be parsed by <code>rlang::as_function()</code></li> </ul>
<code>rvar_args</code>	(character vector) The names of the arguments of <code>.f</code> that should be allowed to accept <code>rvars</code> as arguments. If <code>NULL</code> (the default), all arguments to <code>.f</code> are turned into arguments that accept <code>rvars</code> .
<code>ndraws</code>	(positive integer). The number of draws used to construct new random variables if no <code>rvars</code> are supplied as arguments to the returned function. If <code>NULL</code> , <code>getOption("posterior.rvar.ndraws")</code> is used (default 4000). If any arguments to the returned function contain <code>rvars</code> , the number of draws in the provided <code>rvars</code> is used instead of the value of this argument.

**Details**

This function wraps an existing function (`.f`) such that it returns `rvars` containing whatever type of data `.f` would normally return.

The returned function, when called, executes `.f` possibly multiple times, once for each draw of the `rvars` passed to it, then returns a new `rvar` representing the output of those function evaluations.

If the arguments contain no `rvars`, then `.f` will be executed `ndraws` times and an `rvar` with that many draws returned.

Functions created by `rfun()` are not necessarily *fast* (in fact in some cases they may be very slow), but they have the advantage of allowing a nearly arbitrary R functions to be executed against `rvars` simply by wrapping them with `rfun()`. This makes it especially useful as a prototyping tool. If you create code with `rfun()` and it is unacceptably slow for your application, consider rewriting it using math operations directly on `rvars` (which should be fast), using `rvar_rng()`, and/or using operations directly on the arrays that back the `rvars` (via `draws_of()`).

## Value

A function with the same argument specification as `.f`, but which can accept and return `rvars`.

## See Also

Other `rfun`: `rdo()`, `rvar_rng()`

## Examples

```
rvar_norm <- rfun(rnorm)
rvar_gamma <- rfun(rgamma)

mu <- rvar_norm(10, mean = 1:10, sd = 1)
sigma <- rvar_gamma(1, shape = 1, rate = 1)
x <- rvar_norm(10, mu, sigma)
x
```

---

rhat

*Rhat convergence diagnostic*

---

## Description

Compute the Rhat convergence diagnostic for a single variable as the maximum of rank normalized split-Rhat and rank normalized folded-split-Rhat as proposed in Vehtari et al. (2021).

## Usage

```
rhat(x, ...)
```

## Default S3 method:

```
rhat(x, ...)
```

## S3 method for class 'rvar'

```
rhat(x, ...)
```

**Arguments**

- x (multiple options) One of:
- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
  - An [rvar](#).
- ... Arguments passed to individual methods (if applicable).

**Value**

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#), returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function.

**References**

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

**See Also**

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [mcse\\_sd\(\)](#), [rhat\\_basic\(\)](#), [rstar\(\)](#)

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
rhat(mu)

d <- as_draws_rvars(example_draws("multi_normal"))
rhat(d$Sigma)
```

---

rhat\_basic

*Basic version of the Rhat convergence diagnostic*


---

**Description**

Compute the basic Rhat convergence diagnostic for a single variable as described in Gelman et al. (2013) with some changes according to Vehtari et al. (2021). For practical applications, we strongly recommend the improved Rhat convergence diagnostic implemented in [rhat\(\)](#).



**Usage**

```
rhat_basic(x, ...)

## Default S3 method:
rhat_basic(x, split = TRUE, ...)

## S3 method for class 'rvar'
rhat_basic(x, split = TRUE, ...)
```

**Arguments**

**x** (multiple options) One of:

- A matrix of draws for a single variable (iterations x chains). See [extract\\_variable\\_matrix\(\)](#).
- An [rvar](#).

**...** Arguments passed to individual methods (if applicable).

**split** (logical) Should the estimate be computed on split chains? The default is TRUE.

**Value**

If the input is an array, returns a single numeric value. If any of the draws is non-finite, that is, NA, NaN, Inf, or -Inf, the returned output will be (numeric) NA. Also, if all draws within any of the chains of a variable are the same (constant), the returned output will be (numeric) NA as well. The reason for the latter is that, for constant draws, we cannot distinguish between variables that are supposed to be constant (e.g., a diagonal element of a correlation matrix is always 1) or variables that just happened to be constant because of a failure of convergence or other problems in the sampling process.

If the input is an [rvar](#), returns an array of the same dimensions as the [rvar](#), where each element is equal to the value that would be returned by passing the draws array for that element of the [rvar](#) to this function.

**References**

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari and Donald B. Rubin (2013). *Bayesian Data Analysis, Third Edition*. Chapman and Hall/CRC.

Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Data Analysis*. 16(2), 667–718. doi:10.1214/20-BA1221

**See Also**

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [mcse\\_sd\(\)](#), [rhat\(\)](#), [rstar\(\)](#)

**Examples**

```
mu <- extract_variable_matrix(example_draws(), "mu")
rhat_basic(mu)
```

```
d <- as_draws_rvars(example_draws("multi_normal"))
rhat_basic(d$Sigma)
```

---

rstar

*Calculate R\* convergence diagnostic*


---

### Description

The `rstar()` function generates a measure of convergence for MCMC draws based on whether it is possible to determine the Markov chain that generated a draw with probability greater than chance. To do so, it fits a machine learning classifier to a training set of MCMC draws and evaluates its predictive accuracy on a testing set: giving the ratio of accuracy to predicting a chain uniformly at random.

### Usage

```
rstar(
  x,
  split = TRUE,
  uncertainty = FALSE,
  method = "rf",
  hyperparameters = NULL,
  training_proportion = 0.7,
  nsimulations = 1000,
  ...
)
```

### Arguments

<code>x</code>	(draws) A <code>draws_df</code> object or one coercible to a <code>draws_df</code> object.
<code>split</code>	(logical) Should the estimate be computed on split chains? The default is <code>TRUE</code> .
<code>uncertainty</code>	(logical). Indicates whether to provide a vector of $R^*$ values representing uncertainty in the calculated value (if <code>TRUE</code> ) or a single value (if <code>FALSE</code> ). The default is <code>TRUE</code> .
<code>method</code>	(string) The machine learning classifier to use (must be available in the <code>caret</code> package). The default is <code>"rf"</code> , which calls the random forest classifier.
<code>hyperparameters</code>	(named list) Hyperparameter settings passed to the classifier. The default for the random forest classifier ( <code>method = "rf"</code> ) is <code>list(mtry = floor(sqrt(nvariables(x))))</code> . The default for the gradient-based model ( <code>method = "gbm"</code> ) is <code>list(interaction.depth = 3, n.trees = 50, shrinkage = 0.1, n.minobsinnode = 10)</code> .
<code>training_proportion</code>	(positive real) The proportion (in $(0, 1)$ ) of iterations in used to train the classifier. The default is <code>0.7</code> .

`nsimulations` (positive integer) The number of  $R^*$  values in the returned vector if uncertainty is TRUE. The default is 1000.

... Other arguments passed to `caret::train()`.

### Details

The `rstar()` function provides a measure of MCMC convergence based on whether it is possible to determine the chain that generated a particular draw with a probability greater than chance. To do so, it fits a machine learning classifier to a subset of the original MCMC draws (the training set) and evaluates its predictive accuracy on the remaining draws (the testing set). If predictive accuracy exceeds chance (i.e. predicting the chain that generated a draw uniformly at random), the diagnostic measure  $R^*$  will be above 1, indicating that convergence has yet to occur. This statistic is recently developed, and it is currently unclear what is a reasonable threshold for diagnosing convergence.

The statistic,  $R^*$ , is stochastic, meaning that each time the test is run, unless the random seed is fixed, it will generally produce a different result. To minimize the implications of this stochasticity, it is recommended to repeatedly run this function to calculate a distribution of  $R^*$ ; alternatively, an approximation to this distribution can be obtained by setting `uncertainty = TRUE`, although this approximation of uncertainty will generally have a lower mean.

By default, a random forest classifier is used (`method = "rf"`), which tends to perform best for target distributions of around 4 dimensions and above. For lower dimensional targets, gradient boosted models (called via `method = "gbm"`) tend to have a higher classification accuracy. On a given MCMC sample, it is recommended to try both of these classifiers.

### Value

A numeric vector of length 1 (by default) or length `nsimulations` (if `uncertainty = TRUE`).

### References

Ben Lambert, Aki Vehtari (2020)  $R^*$ : A robust MCMC convergence diagnostic with uncertainty using gradient-boosted machines. *arXiv preprint* arXiv:2003.07900.

### See Also

Other diagnostics: [ess\\_basic\(\)](#), [ess\\_bulk\(\)](#), [ess\\_quantile\(\)](#), [ess\\_sd\(\)](#), [ess\\_tail\(\)](#), [mcse\\_mean\(\)](#), [mcse\\_quantile\(\)](#), [mcse\\_sd\(\)](#), [rhat\\_basic\(\)](#), [rhat\(\)](#)

### Examples

```
if (require("caret", quietly = TRUE) && require("randomForest", quietly = TRUE)) {
  x <- example_draws("eight_schools")
  print(rstar(x))
  print(rstar(x, split = FALSE))
  print(rstar(x, method = "gbm"))
  # can pass additional arguments to methods
  print(rstar(x, method = "gbm", verbose = FALSE))

  # with uncertainty, returns a vector of R* values
  hist(rstar(x, uncertainty = TRUE))
}
```

```

hist(rstar(x, uncertainty = TRUE, nsimulations = 100))

# can use other classification methods in caret library
print(rstar(x, method = "knn"))
}

```

---

rvar

*Random variables of arbitrary dimension*


---

### Description

Random variables backed by arrays of arbitrary dimension

### Usage

```

rvar(
  x = double(),
  dim = NULL,
  dimnames = NULL,
  nchains = 1L,
  with_chains = FALSE
)

```

### Arguments

- |          |  |
|----------|--|
| x        | (multiple options) The object to convert to an rvar: <ul style="list-style-type: none"> <li>• A vector of draws from a distribution.</li> <li>• An array where the first dimension represents draws from a distribution. The resulting <code>rvar</code> will have dimension <code>dim(x)[-1]</code>; that is, everything except the first dimension is used for the shape of the variable, and the first dimension is used to index draws from the distribution (see <b>Examples</b>). Optionally, if <code>with_chains == TRUE</code>, the first dimension indexes the iteration and the second dimension indexes the chain (see <code>with_chains</code>).</li> </ul> |
| dim      | (integer vector) One or more integers giving the maximal indices in each dimension to override the dimensions of the <code>rvar</code> to be created (see <code>dim()</code> ). If <code>NULL</code> (the default), <code>dim</code> is determined by the input. <b>NOTE:</b> This argument controls the dimensions of the <code>rvar</code> , not the underlying array, so you cannot change the number of draws using this argument.   |
| dimnames | (list) Character vectors giving the names in each dimension to override the names of the dimensions of the <code>rvar</code> to be created (see <code>dimnames()</code> ). If <code>NULL</code> (the default), this is determined by the input. <b>NOTE:</b> This argument controls the names of the dimensions of the <code>rvar</code> , not the underlying array.   |
| nchains  | (positive integer) The number of chains. The default is 1.   |

`with_chains` (logical) Does `x` include a dimension for chains? If `FALSE` (the default), chains are not included, the first dimension of the input array should index draws, and the `nchains` argument can be used to determine the number of chains. If `TRUE`, the `nchains` argument is ignored and the second dimension of `x` is used to index chains. Internally, the array will be converted to a format without the chain index.

## Details

The "rvar" class internally represents random variables as arrays of arbitrary dimension, where the first dimension is used to index draws from the distribution. Most mathematical operators and functions are supported, including efficient matrix multiplication and vector and array-style indexing. The intent is that an rvar works as closely as possible to how a base vector/matrix/array does, with a few differences:

- The default behavior when subsetting is not to drop extra dimensions (i.e. the default `drop` argument for `[]` is `FALSE`, not `TRUE`).
- Rather than base R-style recycling, rvars use a limited form of broadcasting: if an operation is being performed on two vectors with different size of the same dimension, the smaller vector will be recycled up to the size of the larger one along that dimension so long as it has size 1.

For functions that expect base numeric arrays and for which rvars cannot be used directly as arguments, you can use `rfunc()` or `rdo()` to translate your code into code that executes across draws from one or more random variables and returns a random variable as output. Typically `rdo()` offers the most straightforward translation.

As `rfunc()` and `rdo()` incur some performance cost, you can also operate directly on the underlying array using the `draws_of()` function. To re-use existing random number generator functions to efficiently create rvars, use `rvar_rng()`.

## Value

An object of class "rvar" representing a random variable.

## See Also

`as_rvar()` to convert objects to rvars. See `rdo()`, `rfunc()`, and `rvar_rng()` for higher-level interfaces for creating rvars.

## Examples

```
set.seed(1234)

# To create a "scalar" `rvar`, pass a one-dimensional array or a vector
# whose length (here `4000`) is the desired number of draws:
x <- rvar(rnorm(4000, mean = 1, sd = 1))
x

# Create random vectors by adding an additional dimension:
n <- 4 # length of output vector
```

```

x <- rvar(array(rnorm(4000 * n, mean = rep(1:n, each = 4000), sd = 1), dim = c(4000, n)))
x

# Create a random matrix:
rows <- 4
cols <- 3
x <- rvar(array(rnorm(4000 * rows * cols, mean = 1, sd = 1), dim = c(4000, rows, cols)))
x

# If the input sample comes from multiple chains, we can indicate that using the
# nchains argument (here, 1000 draws each from 4 chains):
x <- rvar(rnorm(4000, mean = 1, sd = 1), nchains = 4)
x

# Or if the input sample has chain information as its second dimension, we can
# use with_chains to create the rvar
x <- rvar(array(rnorm(4000, mean = 1, sd = 1), dim = c(1000, 4)), with_chains = TRUE)
x

```

---

rvar-dist

*Density, CDF, and quantile functions of random variables*


---

## Description

The probability density function (`density()`), cumulative distribution function (`cdf()`), and quantile function / inverse CDF (`quantile()`) of an `rvar`.

## Usage

```

## S3 method for class 'rvar'
density(x, at, ...)

## S3 method for class 'rvar'
cdf(x, q, ...)

## S3 method for class 'rvar'
quantile(x, probs, ...)

```

## Arguments

<code>x</code>	( <code>rvar</code> ) An <code>rvar</code> object.
<code>...</code>	Additional arguments passed onto underlying methods: <ul style="list-style-type: none"> <li>• For <code>density()</code>, these are passed to <code>stats::density()</code>.</li> <li>• For <code>cdf()</code>, these are ignored.</li> <li>• For <code>quantile()</code>, these are passed to <code>stats::quantile()</code>.</li> </ul>
<code>q, at</code>	(numeric vector) One or more quantiles.
<code>probs</code>	(numeric vector) One or more probabilities in $[\theta, 1]$ .

**Value**

If `x` is a scalar `rvar`, returns a vector of the same length as the input (`q`, `at`, or `probs`) containing values from the corresponding function of the given `rvar`.

If `x` has length greater than 1, returns an array with dimensions `c(length(y), dim(x))` where `y` is `q`, `at`, or `probs`, where each `result[i, ...]` is the value of the corresponding function, `f(y[i])`, for the corresponding cell in the input array, `x[...]`.

**Examples**

```
set.seed(1234)
x = rvar(rnorm(100))

density(x, seq(-2, 2, length.out = 10))
cdf(x, seq(-2, 2, length.out = 10))
quantile(x, ppoints(10))

x2 = c(rvar(rnorm(100, mean = -0.5)), rvar(rnorm(100, mean = 0.5)))
density(x2, seq(-2, 2, length.out = 10))
cdf(x2, seq(-2, 2, length.out = 10))
quantile(x2, ppoints(10))
```

---

rvar-matmult

*Matrix multiplication of random variables*


---

**Description**

Matrix multiplication of random variables.

**Usage**

```
x %**% y
```

**Arguments**

`x` (multiple options) The object to be postmultiplied by `y`:

- An `rvar`
- A `numeric` vector or matrix
- A `logical` vector or matrix

If a vector is used, it is treated as a *row* vector.

`y` (multiple options) The object to be premultiplied by `x`:

- An `rvar`
- A `numeric` vector or matrix
- A `logical` vector or matrix

If a vector is used, it is treated as a *column* vector.

**Details**

If  $x$  or  $y$  are vectors, they are converted into matrices prior to multiplication, with  $x$  converted to a row vector and  $y$  to a column vector. Numerics and logicals can be multiplied by `rvars` and are broadcasted across all draws of the `rvar` argument. Tensor multiplication is used to efficiently multiply matrices across draws, so if either  $x$  or  $y$  is an `rvar`,  $x \%*\% y$  will be much faster than `rdo(x \%*\% y)`.

Because `rvar` is an S3 class and S3 classes cannot properly override `%*%`, `rvars` use `%**%` for matrix multiplication.

**Value**

An `rvar` representing the matrix product of  $x$  and  $y$ .

**Examples**

```
# d has mu (mean vector of length 3) and Sigma (3x3 covariance matrix)
d <- as_draws_rvars(example_draws("multi_normal"))
d$Sigma

# trivial example: multiplication by a non-random matrix
d$Sigma %**% diag(1:3)

# Decompose Sigma into R s.t. R'R = Sigma ...
R <- chol(d$Sigma)
# ... and recreate Sigma using matrix multiplication
t(R) %**% R
```

---

rvar-summaries-over-draws

*Summaries of random variables within array elements, over draws*

---

**Description**

Compute summaries within elements of an `rvar` and over draws of each element, producing an array of the same shape as the input random variable (except in the case of `range()`, see **Details**).

**Usage**

```
E(x, ...)

## S3 method for class 'rvar'
mean(x, ...)

Pr(x, ...)
```



```
## Default S3 method:
Pr(x, ...)

## S3 method for class 'logical'
Pr(x, ...)

## S3 method for class 'rvar'
Pr(x, ...)

## S3 method for class 'rvar'
median(x, ...)

## S3 method for class 'rvar'
min(x, ...)

## S3 method for class 'rvar'
max(x, ...)

## S3 method for class 'rvar'
sum(x, ...)

## S3 method for class 'rvar'
prod(x, ...)

## S3 method for class 'rvar'
all(x, ...)

## S3 method for class 'rvar'
any(x, ...)

## S3 method for class 'rvar'
Summary(...)

## S3 method for class 'rvar'
variance(x, ...)

var(x, ...)

## Default S3 method:
var(x, ...)

## S3 method for class 'rvar'
var(x, ...)

sd(x, ...)

## Default S3 method:
sd(x, ...)
```

```

## S3 method for class 'rvar'
sd(x, ...)

mad(x, ...)

## Default S3 method:
mad(x, ...)

## S3 method for class 'rvar'
mad(x, ...)

## S3 method for class 'rvar'
range(x, ...)

## S3 method for class 'rvar'
is.finite(x)

## S3 method for class 'rvar'
is.infinite(x)

## S3 method for class 'rvar'
is.nan(x)

## S3 method for class 'rvar'
is.na(x)

```

### Arguments

x	(rvar) An <a href="#">rvar</a> .
...	Further arguments passed to underlying functions (e.g., <code>base::mean()</code> or <code>base::median()</code> ), such as <code>na.rm</code> .

### Details

Summaries include expectations (`E()` or `mean()`), probabilities (`Pr()`), medians (`median()`), spread (`var()`, `variance()`, `sd()`, `mad()`), sums and products (`sum()`, `prod()`), extrema and ranges (`min()`, `max()`, `range()`), logical summaries (`all()`, `any()`), and special value predicates (`is.finite()`, `is.infinite()`, `is.nan()`, `is.na()`).

Unless otherwise stated, these functions return a numeric array with the same shape (same dimensions) as the input [rvar](#), `x`.

`range(x)` returns an array with dimensions `c(2, dim(x))`, where the last dimension contains the minimum and maximum values.

`is.infinite(x)`, `is.nan(x)`, and `is.na(x)` return logical arrays, where each element is TRUE if **any** draws in its corresponding element in `x` match the predicate. Each elements in the result of `is.finite(x)` is TRUE if **all** draws in the corresponding element in `x` are finite.

Both `E()`, `mean()`, and `Pr()` return the means of each element in the input. `Pr()` additionally checks that the provided [rvar](#) is a logical variable (hence, taking its expectation results in a probability).

For consistency, `E()` and `Pr()` are also defined for base arrays so that they can be used as summary functions in `summarise_draws()`.

### Value

A numeric or logical vector with the same dimensions as the given random variable, where each entry in the vector is the mean, median, or variance of the corresponding entry in `x`.

### See Also

[rvar-summaries-within-draws](#) for summary functions within draws. [rvar-dist](#) for density, CDF, and quantile functions of random variables.

Other rvar-summaries: [rvar-summaries-within-draws](#), [rvar\\_is\\_finite\(\)](#)

### Examples

```
set.seed(5678)
x = rvar_rng(rnorm, 4, mean = 1:4, sd = 2)

# These should all be ~ c(1, 2, 3, 4)
E(x)
mean(x)
median(x)

# This ...
Pr(x < 1.5)
# ... should be about the same as this:
pnorm(1.5, mean = 1:4, sd = 2)
```

---

rvar-summaries-within-draws

*Summaries of random variables over array elements, within draws*

---

### Description

Compute summaries of random variables over array elements and within draws, producing a new random variable of length 1 (except in the case of `rvar_range()`, see **Details**).

### Usage

```
rvar_mean(..., na.rm = FALSE)
```

```
rvar_median(..., na.rm = FALSE)
```

```
rvar_sum(..., na.rm = FALSE)
```

```

rvar_prod(..., na.rm = FALSE)
rvar_min(..., na.rm = FALSE)
rvar_max(..., na.rm = FALSE)
rvar_sd(..., na.rm = FALSE)
rvar_var(..., na.rm = FALSE)
rvar_mad(..., constant = 1.4826, na.rm = FALSE)
rvar_range(..., na.rm = FALSE)
rvar_quantile(..., probs, names = FALSE, na.rm = FALSE)
rvar_all(..., na.rm = FALSE)
rvar_any(..., na.rm = FALSE)

```

### Arguments

...	(rvar) One or more <a href="#">rvars</a> .
na.rm	(logical) Should NAs be removed from the input before summaries are computed? The default is FALSE.
constant	(scalar real) For <code>rvar_mad()</code> , a scale factor for computing the median absolute deviation. See the details of <code>stats::mad()</code> for the justification for the default value.
probs	(numeric vector) For <code>rvar_quantile()</code> , probabilities in $[\theta, 1]$ .
names	(logical) For <code>rvar_quantile()</code> , if TRUE, the result has a <code>names</code> attribute.

### Details

These functions compute statistics within each draw of the random variable. For summaries over draws (such as expectations), see [rvar-summaries-over-draws](#).

Each function defined here corresponds to the base function of the same name without the `rvar_` prefix (e.g., `rvar_mean()` calls `mean()` under the hood, etc).

### Value

An [rvar](#) of length 1 (for `range()`, length 2; for `quantile()`, length equal to `length(probs)`) with the same number of draws as the input `rvar(s)` containing the summary statistic computed within each draw of the input `rvar(s)`.

### See Also

[rvar-summaries-over-draws](#) for summary functions across draws (e.g. expectations). [rvar-dist](#) for density, CDF, and quantile functions of random variables.

Other rvar-summaries: [rvar-summaries-over-draws](#), [rvar\\_is\\_finite\(\)](#)

## Examples

```
set.seed(5678)
x = rvar_rng(rnorm, 4, mean = 1:4, sd = 2)

# These will give similar results to mean(1:4),
# median(1:4), sum(1:4), prod(1:4), etc
rvar_mean(x)
rvar_median(x)
rvar_sum(x)
rvar_prod(x)
rvar_range(x)
rvar_quantile(x, probs = c(0.25, 0.5, 0.75), names = TRUE)
```

---

rvar_apply	<i>Random variable resulting from a function applied over margins of an array or random variable</i>
------------	--

---

## Description

Returns an [rvar](#) obtained by applying a function to margins of an array or [rvar](#). Acts like `apply()`, except that the function supplied (`.f`) should return an [rvar](#), and the final result is always an [rvar](#).

## Usage

```
rvar_apply(.x, .margin, .f, ...)
```

## Arguments

<code>.x</code>	An array or an <a href="#">rvar</a> .
<code>.margin</code>	(multiple options) The subscripts which the function will be applied over: <ul style="list-style-type: none"> <li>• An integer vector. E.g., for a matrix 1 indicates rows, 2 indicates columns, <code>c(1, 2)</code> indicates rows and columns.</li> <li>• A character vector of dimension names if <code>.x</code> has named dimensions.</li> </ul>
<code>.f</code>	(function) The function to be applied. The function <code>.f</code> must return an <a href="#">rvar</a> and the dimensions of the result of <code>.f</code> applied to each margin of <code>.x</code> must be able to be broadcasted to a common shape (otherwise the resulting <a href="#">rvar</a> cannot be simplified). See <b>Details</b> .
<code>...</code>	Optional arguments passed to <code>.f</code> .

## Details

This function acts much like `apply()`, except that the function passed to it (`.f`) must return `rvars`, and the result is simplified into an `rvar`. Unlike `apply()`, it also keeps the dimensions of the returned values along each margin, rather than simplifying each margin to a vector, and if the results of `.f` do not all have the same dimensions, it applies the `rvar` broadcasting rules to bind results together rather than using vector recycling.

If you wish to apply functions over `rvars` where the result is not intended to be simplified into an `rvar`, you can use the standard `apply()`, `lapply()`, `sapply()`, or `vapply()` functions.

## Value

An `rvar`.

If the result of each call to `.f` returns an `rvar` of dimension `d` after being broadcast to a common shape, then `rvar_apply()` returns an `rvar` of dimension `c(d, dim(.x)[.margin])`. If the last dimension of the result would be 1, it is dropped (other dimensions equal to 1 are retained). If `d` is  $\emptyset$ , the result has length  $\emptyset$  but not necessarily the 'correct' dimension.

## See Also

`as_rvar()` to convert objects to `rvars`. See `rdo()`, `rfunc()`, and `rvar_rng()` for higher-level interfaces for creating `rvars`.

## Examples

```
set.seed(3456)
x <- rvar_rng(rnorm, 24, mean = 1:24)
dim(x) <- c(2,3,4)

# we can find the distributions of marginal means of the above array
# using rvar_mean along with rvar_apply
rvar_apply(x, 1, rvar_mean)
rvar_apply(x, 2:3, rvar_mean)
```

---

rvar\_is\_finite

*Special value predicates for random variables*

---

## Description

Compute special value predicates (checking for finite / infinite values, NaN, and NA) on all draws within a random variable, returning a random variable.

**Usage**

```
rvar_is_finite(x)
rvar_is_infinite(x)
rvar_is_nan(x)
rvar_is_na(x)
```

**Arguments**

x (rvar) An [rvar](#).

**Details**

These functions return a new [rvar](#) that is the result of applying `is.finite()`, `is.infinite()`, `is.nan()`, or `is.na()` to every draw in the input random variable.

**Value**

A logical [rvar](#) of the same length as the input.

**See Also**

[rvar-summaries-over-draws](#) for summary functions across draws, including implementations of `is.finite()`, `is.infinite()`, `is.nan()`, and `is.na()` for rvars.

Other rvar-summaries: [rvar-summaries-over-draws](#), [rvar-summaries-within-draws](#)

**Examples**

```
x <- rvar(c(1, Inf, -Inf, NaN, NA))
x

rvar_is_finite(x)
rvar_is_infinite(x)
rvar_is_nan(x)
rvar_is_na(x)
```

---

rvar\_rng

*Create random variables from existing random number generators*

---

**Description**

Specialized alternative to `rdo()` or `rfun()` for creating [rvars](#) from existing random-number generator functions (such as `rnorm()`, `rbinom()`, etc).

**Usage**

```
rvar_rng(.f, n, ..., ndraws = NULL)
```

**Arguments**

<code>.f</code>	(function) A function (or string naming a function) representing a random-number generating function that follows the pattern of base random number generators (like <code>rnorm()</code> , <code>rbinom()</code> , etc). It must: <ul style="list-style-type: none"> <li>• Have a first argument, <code>n</code>, giving the number of draws to take from the distribution</li> <li>• Have vectorized parameter arguments</li> <li>• Return a single vector of length <code>n</code></li> </ul>
<code>n</code>	(positive integer) The length of the output <code>rvar</code> vector ( <b>not</b> the number of draws).
<code>...</code>	Arguments passed to <code>.f</code> . These arguments may include <code>rvars</code> , so long as they are vectors only (no multidimensional <code>rvars</code> are allowed).
<code>ndraws</code>	(positive integer) The number of draws used to construct the returned random variable if no <code>rvars</code> are supplied in <code>...</code> . If <code>NULL</code> , <code>getOption("posterior.rvar.ndraws")</code> is used (default 4000). If <code>...</code> contains <code>rvars</code> , the number of draws in the provided <code>rvars</code> is used instead of the value of this argument.

**Details**

This function unwraps the arrays underlying the input `rvars` in `...` and then passes them to `.f`, relying on the vectorization of `.f` to evaluate it across draws from the input `rvars`. This is why the arguments of `.f` **must** be vectorized. It asks for `n` times the number of draws in the input `rvars` (or `ndraws` if none are given) draws from the random number generator `.f`, then reshapes the output from `.f` into an `rvar` with length `n`.

`rvar_rng()` is a fast alternative to `rdo()` or `rfunc()`, but you **must** ensure that `.f` satisfies the preconditions described above for the result to be correct. Most base random number generators satisfy these conditions. It is advisable to test against `rdo()` or `rfunc()` (which should be correct, but slower) if you are uncertain.

**Value**

A single-dimensional `rvar` of length `n`.

**See Also**

Other `rfunc`: `rdo()`, `rfunc()`

**Examples**

```
mu <- rvar_rng(rnorm, 10, mean = 1:10, sd = 1)
sigma <- rvar_rng(rgamma, 1, shape = 1, rate = 1)
x <- rvar_rng(rnorm, 10, mu, sigma)
x
```



---

`split_chains`*Split Chains*

---

**Description**

Split chains by halving the number of iterations per chain and doubling the number of chains.

**Usage**

```
split_chains(x, ...)
```

**Arguments**

`x` (draws) A draws object or another R object for which the method is defined.  
`...` Arguments passed to individual methods (if applicable).

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- example_draws()  
niterations(x)  
nchains(x)
```

```
x <- split_chains(x)  
niterations(x)  
nchains(x)
```

---

`subset_draws`*Subset draws objects*

---

**Description**

Subset [draws](#) objects by variables, iterations, chains, and draws indices.

**Usage**

```
subset_draws(x, ...)  
  
## S3 method for class 'draws_matrix'  
subset_draws(  
  x,  
  variable = NULL,  
  iteration = NULL,  
  chain = NULL,  
  draw = NULL,  
  regex = FALSE,  
  unique = TRUE,  
  ...  
)  
  
## S3 method for class 'draws_array'  
subset_draws(  
  x,  
  variable = NULL,  
  iteration = NULL,  
  chain = NULL,  
  draw = NULL,  
  regex = FALSE,  
  unique = TRUE,  
  ...  
)  
  
## S3 method for class 'draws_df'  
subset_draws(  
  x,  
  variable = NULL,  
  iteration = NULL,  
  chain = NULL,  
  draw = NULL,  
  regex = FALSE,  
  unique = TRUE,  
  ...  
)  
  
## S3 method for class 'draws_list'  
subset_draws(  
  x,  
  variable = NULL,  
  iteration = NULL,  
  chain = NULL,  
  draw = NULL,  
  regex = FALSE,  
  unique = TRUE,
```

```

    ...
  )

## S3 method for class 'draws_rvars'
subset_draws(
  x,
  variable = NULL,
  iteration = NULL,
  chain = NULL,
  draw = NULL,
  regex = FALSE,
  unique = TRUE,
  ...
)

## S3 method for class 'draws'
subset(x, ...)
```

### Arguments

x	(draws) A draws object or another R object for which the method is defined.
...	Arguments passed to individual methods (if applicable).
variable	(character vector) The variables to select. All elements of non-scalar variables can be selected at once.
iteration	(integer vector) The iteration indices to select.
chain	(integer vector) The chain indices to select.
draw	(integer vector) The draw indices to be select. Subsetting draw indices will lead to an automatic merging of chains via <a href="#">merge_chains</a> .
regex	(logical) Should variable should be treated as a (vector of) regular expressions? Any variable in x matching at least one of the regular expressions will be selected. Defaults to FALSE.
unique	(logical) Should duplicated selection of chains, iterations, or draws be allowed? If TRUE (the default) only unique chains, iterations, and draws are selected regardless of how often they appear in the respective selecting arguments.

### Details

To ensure that multiple consecutive subsetting operations work correctly, `subset()` *repairs* the draws object before and after subsetting.

### Value

A draws object of the same class as x.

**Examples**

```
x <- example_draws()
subset_draws(x, variable = c("mu", "tau"))
subset_draws(x, chain = 2)
subset_draws(x, iteration = 5:10, chain = 3:4)

# extract the first chain twice
subset_draws(x, chain = c(1, 1), unique = FALSE)

# extract all elements of 'theta'
subset_draws(x, variable = "theta")
```

---

thin\_draws

*Thin draws objects*


---

**Description**

Thin [draws](#) objects to reduce their size and autocorrelation in the chains.

**Usage**

```
thin_draws(x, thin, ...)

## S3 method for class 'draws'
thin_draws(x, thin, ...)
```

**Arguments**

`x` (draws) A draws object or another R object for which the method is defined.  
`thin` (positive integer) The period for selecting draws.  
`...` Arguments passed to individual methods (if applicable).

**Value**

A draws object of the same class as `x`.

**Examples**

```
x <- example_draws()
niterations(x)

x <- thin_draws(x, thin = 5)
niterations(x)
```

---

weights.draws	<i>Extract Weights from Draws Objects</i>
---------------	---

---

## Description

Extract weights from [draws](#) objects, with one weight per draw. See [weight\\_draws](#) for details how to add weights to [draws](#) objects.

## Usage

```
## S3 method for class 'draws'  
weights(object, log = FALSE, normalize = TRUE, ...)
```

## Arguments

object	(draws) A <a href="#">draws</a> object.
log	(logical) Should the weights be returned on the log scale? Defaults to FALSE.
normalize	(logical) Should the weights be normalized to sum to 1 on the standard scale? Defaults to TRUE.
...	Arguments passed to individual methods (if applicable).

## Value

A vector of weights, with one weight per draw.

## See Also

[weight\\_draws](#), [resample\\_draws](#)

## Examples

```
x <- example_draws()  
  
# sample some random weights for illustration  
wts <- rexp(ndraws(x))  
head(wts)  
  
# add weights  
x <- weight_draws(x, weights = wts)  
  
# extract weights  
head(weights(x)) # defaults to normalized weights  
head(weights(x, normalize=FALSE)) # recover original weights  
head(weights(x, log=TRUE)) # get normalized log-weights  
  
# add weights which are already on the log scale  
log_wts <- log(wts)  
head(log_wts)
```

```
x <- weight_draws(x, weights = log_wts, log = TRUE)
# extract weights
head(weights(x))
head(weights(x, log=TRUE, normalize = FALSE)) # recover original log_wts
```

---

weight\_draws

*Weight draws objects*


---

## Description

Add weights to [draws](#) objects, with one weight per draw, for use in subsequent weighting operations. For reasons of numerical accuracy, weights are stored in the form of unnormalized log-weights (in a variable called `.log_weight`). See [weights.draws\(\)](#) for details how to extract weights from draws objects.

## Usage

```
weight_draws(x, weights, ...)

## S3 method for class 'draws_matrix'
weight_draws(x, weights, log = FALSE, ...)

## S3 method for class 'draws_array'
weight_draws(x, weights, log = FALSE, ...)

## S3 method for class 'draws_df'
weight_draws(x, weights, log = FALSE, ...)

## S3 method for class 'draws_list'
weight_draws(x, weights, log = FALSE, ...)

## S3 method for class 'draws_rvars'
weight_draws(x, weights, log = FALSE, ...)
```

## Arguments

<code>x</code>	(draws) A draws object or another <b>R</b> object for which the method is defined.
<code>weights</code>	(numeric vector) A vector of weights of length <code>ndraws(x)</code> . Weights will be internally stored on the log scale (in a variable called <code>.log_weight</code> ) and will not be normalized, but normalized (non-log) weights can be returned via the <a href="#">weights.draws()</a> method later.
<code>...</code>	Arguments passed to individual methods (if applicable).
<code>log</code>	(logical) Are the weights passed already on the log scale? The default is <code>FALSE</code> , that is, expecting weights to be on the standard (non-log) scale.

**Value**

A draws object of the same class as x.

**See Also**

[weights.draws\(\)](#), [resample\\_draws\(\)](#)

**Examples**

```
x <- example_draws()

# sample some random weights for illustration
wts <- rexp(ndraws(x))
head(wts)

# add weights
x <- weight_draws(x, weights = wts)

# extract weights
head(weights(x)) # defaults to normalized weights
head(weights(x, normalize=FALSE)) # recover original weights
head(weights(x, log=TRUE)) # get normalized log-weights

# add weights which are already on the log scale
log_wts <- log(wts)
head(log_wts)

x <- weight_draws(x, weights = log_wts, log = TRUE)
# extract weights
head(weights(x))
head(weights(x, log=TRUE, normalize = FALSE)) # recover original log_wts
```

---

[.draws\_array

*Extract parts of a draws\_array object*

---

**Description**

Extract parts of a draws\_array object. They are strictly defined as arrays of 3 dimensions (iteration x chain x variable) so dropping any of the dimensions breaks the expected structure of the object. Accordingly, no dropping of dimensions is done by default even if the extracted slices are of length 1. If drop is manually set to TRUE and any of the dimensions is actually dropped, this will lead to dropping the "draws\_array" class as well.

**Usage**

```
## S3 method for class 'draws_array'
x[i, j, ..., drop = FALSE]
```

**Arguments**

`x, i, j, ..., drop`

Same as in the default extraction method but with `drop` being set to `FALSE` by default.

**Value**

An object of class `"draws_array"` unless any of the dimensions was dropped during the extraction.

---

[.draws\_matrix      *Extract parts of a draws\_matrix object*

---

**Description**

Extract parts of a `draws_matrix` object. They are strictly defined as matrices (draws x variable) so dropping any of the dimensions breaks the expected structure of the object. Accordingly, no dropping of dimensions is done by default even if the extracted slices are of length 1. If `drop` is manually set to `TRUE` and any of the dimensions is actually dropped, this will lead to dropping the `"draws_matrix"` class as well.

**Usage**

```
## S3 method for class 'draws_matrix'
x[i, j, ..., drop = FALSE]
```

**Arguments**

`x, i, j, ..., drop`

Same as in the default extraction method but with `drop` being set to `FALSE` by default.

**Value**

An object of class `"draws_matrix"` unless any of the dimensions was dropped during the extraction.



# Index

- \* **diagnostics**
  - ess\_basic, 22
  - ess\_bulk, 23
  - ess\_quantile, 26
  - ess\_sd, 27
  - ess\_tail, 28
  - mcse\_mean, 33
  - mcse\_quantile, 34
  - mcse\_sd, 36
  - rhat, 55
  - rhat\_basic, 56
  - rstar, 58
- \* **formats**
  - draws, 7
  - draws\_array, 10
  - draws\_df, 11
  - draws\_list, 13
  - draws\_matrix, 15
  - draws\_rvars, 18
- \* **rfunc**
  - rdo, 48
  - rfunc, 54
  - rvar\_rng, 71
- \* **rvar-summaries**
  - rvar-summaries-over-draws, 64
  - rvar-summaries-within-draws, 67
  - rvar\_is\_finite, 70
- [.draws\_array, 79
- [.draws\_matrix, 80
- \*\*\* (rvar-matmult), 63
- all.rvar (rvar-summaries-over-draws), 64
- any.rvar (rvar-summaries-over-draws), 64
- as\_draws (draws), 7
- as\_draws\_array (draws\_array), 10
- as\_draws\_df (draws\_df), 11
- as\_draws\_list (draws\_list), 13
- as\_draws\_matrix (draws\_matrix), 15
- as\_draws\_rvars (draws\_rvars), 18
- as\_function(), 20
- as\_rvar, 4
- as\_rvar(), 33, 61, 70
- bind\_draws, 5
- cdf.rvar (rvar-dist), 62
- chain\_ids (draws-index), 8
- chol.rvar, 6
- convergence (diagnostics), 7
- default\_convergence\_measures (draws\_summary), 19
- default\_mcse\_measures (draws\_summary), 19
- default\_summary\_measures (draws\_summary), 19
- density.rvar (rvar-dist), 62
- diagnostic, 20
- diagnostics, 7, 21
- dim(), 4, 48, 60
- dimnames(), 4, 60
- draw\_ids (draws-index), 8
- draws, 4–6, 7, 11, 13, 14, 16, 19, 37–39, 49, 51, 53, 73, 76–78
- draws-index, 8
- draws\_array, 8, 10, 13, 14, 16, 19, 30, 39, 40
- draws\_df, 8, 11, 11, 14, 16, 19, 39, 41, 53, 58
- draws\_list, 8, 11, 13, 13, 16, 19, 39, 42
- draws\_matrix, 8, 11, 13, 14, 15, 19, 39, 43
- draws\_of, 16
- draws\_of(), 48, 55, 61
- draws\_of<- (draws\_of), 16
- draws\_rvars, 8, 11, 13, 14, 16, 18, 39, 44
- draws\_summary, 19
- E (rvar-summaries-over-draws), 64
- ess\_basic, 22, 24, 27–29, 34, 35, 37, 56, 57, 59
- ess\_basic(), 7
- ess\_bulk, 23, 23, 27–29, 34, 35, 37, 56, 57, 59

- ess\_bulk(), 7, 21, 22, 28
- ess\_mean, 25
- ess\_mean.default (ess\_quantile), 26
- ess\_median (ess\_quantile), 26
- ess\_quantile, 23, 24, 26, 28, 29, 34, 35, 37, 56, 57, 59
- ess\_quantile(), 7
- ess\_sd, 23, 24, 27, 27, 29, 34, 35, 37, 56, 57, 59
- ess\_sd(), 7
- ess\_tail, 23, 24, 27, 28, 28, 34, 35, 37, 56, 57, 59
- ess\_tail(), 7, 21–23
- example\_draws, 30
- extract\_variable, 31
- extract\_variable\_matrix, 32
- extract\_variable\_matrix(), 22, 24–27, 29, 33, 35, 36, 47, 56, 57
  
- format.rvar (print.rvar), 45
  
- is.finite.rvar (rvar-summaries-over-draws), 64
- is.infinite.rvar (rvar-summaries-over-draws), 64
- is.na.rvar (rvar-summaries-over-draws), 64
- is.nan.rvar (rvar-summaries-over-draws), 64
- is\_draws (draws), 7
- is\_draws\_array (draws\_array), 10
- is\_draws\_df (draws\_df), 11
- is\_draws\_list (draws\_list), 13
- is\_draws\_matrix (draws\_matrix), 15
- is\_draws\_rvars (draws\_rvars), 18
- is\_rvar, 33
- is\_rvar(), 48
- iteration\_ids (draws-index), 8
  
- logical, 63
  
- mad (rvar-summaries-over-draws), 64
- mad(), 20
- matrix multiplication, 17
- max.rvar (rvar-summaries-over-draws), 64
- mcse\_mean, 23, 24, 27–29, 33, 35, 37, 56, 57, 59
- mcse\_mean(), 7
- mcse\_median (mcse\_quantile), 34
- mcse\_quantile, 23, 24, 27–29, 34, 34, 37, 56, 57, 59
- mcse\_quantile(), 7
- mcse\_sd, 23, 24, 27–29, 34, 35, 36, 56, 57, 59
- mcse\_sd(), 7
- mean(), 20
- mean.rvar (rvar-summaries-over-draws), 64
- median(), 20
- median.rvar (rvar-summaries-over-draws), 64
- merge\_chains, 37, 75
- min.rvar (rvar-summaries-over-draws), 64
- mutate\_variables, 38, 49
  
- names, 48
- nchains (draws-index), 8
- ndraws (draws-index), 8
- niterations (draws-index), 8
- numeric, 63
- nvariables (draws-index), 8
  
- option, 41–45
- order\_draws, 39
- order\_draws(), 51
  
- pillar::style\_num(), 46
- posterior (posterior-package), 3
- posterior-package, 3
- Pr (rvar-summaries-over-draws), 64
- print(), 41–46
- print.draws\_array, 40
- print.draws\_df, 41
- print.draws\_list, 42
- print.draws\_matrix, 43
- print.draws\_rvars, 44
- print.rvar, 45
- prod.rvar (rvar-summaries-over-draws), 64
  
- quantile.rvar (rvar-dist), 62
- quantile2, 47
- quantile2(), 21
- quasiquoteation, 48
  
- range.rvar (rvar-summaries-over-draws), 64
- rdo, 48, 55, 72
- rdo(), 5, 61, 70

- rename\_variables, 39, 49
- rename\_variables(), 9
- repair\_draws, 50
- repair\_draws(), 40
- repairs, 75
- resample\_draws, 51, 77
- resample\_draws(), 52, 79
- reserved\_variables, 41–45, 53
- rfunc, 49, 54, 72
- rfunc(), 5, 61, 70
- rhat, 23, 24, 27–29, 34, 35, 37, 55, 57, 59
- rhat(), 7, 21, 32, 56
- rhat\_basic, 23, 24, 27–29, 34, 35, 37, 56, 56, 59
- rhat\_basic(), 7
- rlang::as\_function(), 54
- rstar, 23, 24, 27–29, 34, 35, 37, 56, 57, 58
- rstar(), 7
- rvar, 4–7, 16, 17, 19, 22, 24–29, 33–36, 39, 45–49, 54–57, 60, 60, 62–64, 66, 68–72
- rvar(), 5
- rvar-dist, 62, 67, 68
- rvar-matmult, 63
- rvar-summaries-over-draws, 64, 68, 71
- rvar-summaries-within-draws, 67, 67
- rvar\_all (rvar-summaries-within-draws), 67
- rvar\_any (rvar-summaries-within-draws), 67
- rvar\_apply, 69
- rvar\_is\_finite, 67, 69, 70
- rvar\_is\_infinite (rvar\_is\_finite), 70
- rvar\_is\_na (rvar\_is\_finite), 70
- rvar\_is\_nan (rvar\_is\_finite), 70
- rvar\_mad (rvar-summaries-within-draws), 67
- rvar\_max (rvar-summaries-within-draws), 67
- rvar\_mean (rvar-summaries-within-draws), 67
- rvar\_median (rvar-summaries-within-draws), 67
- rvar\_min (rvar-summaries-within-draws), 67
- rvar\_prod (rvar-summaries-within-draws), 67
- rvar\_quantile (rvar-summaries-within-draws), 67
- rvar\_range (rvar-summaries-within-draws), 67
- rvar\_rng, 49, 55, 71
- rvar\_rng(), 5, 48, 55, 61, 70
- rvar\_sd (rvar-summaries-within-draws), 67
- rvar\_sum (rvar-summaries-within-draws), 67
- rvar\_var (rvar-summaries-within-draws), 67
- sd (rvar-summaries-over-draws), 64
- sd(), 20
- split\_chains, 73
- stats::density(), 62
- stats::quantile(), 47, 62
- str.rvar (print.rvar), 45
- subset.draws (subset\_draws), 73
- subset\_draws, 52, 73
- sum.rvar (rvar-summaries-over-draws), 64
- summarise\_draws (draws\_summary), 19
- summarize\_draws (draws\_summary), 19
- summary(), 19
- summary.draws (draws\_summary), 19
- Summary.rvar (rvar-summaries-over-draws), 64
- summary.rvar (draws\_summary), 19
- thin (thin\_draws), 76
- thin\_draws, 76
- tibble, 12, 21
- var (rvar-summaries-over-draws), 64
- variables, 39, 49
- variables (draws-index), 8
- variables<- (draws-index), 8
- variance.rvar (rvar-summaries-over-draws), 64
- weight\_draws, 53, 77, 78
- weight\_draws(), 51
- weights.draws, 77
- weights.draws(), 78, 79