Package 'santoku'

June 8, 2022

Type Package

Title A Versatile Cutting Tool

Version 0.8.0

Maintainer David Hugh-Jones <davidhughjones@gmail.com>

Description A tool for cutting data into intervals. Allows singleton intervals. Always includes the whole range of data by default. Flexible labelling. Convenience functions for cutting by quantiles etc. Handles dates, times, units and other vectors.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.0

Suggests bench, bit64, covr, haven, hms, knitr, lubridate, purrr, rmarkdown, scales, stringi, testthat (>= 2.1.0), units, withr, xts, zoo

LinkingTo Rcpp

Imports Rcpp, assertthat, glue, lifecycle, rlang, vctrs

URL https://github.com/hughjonesd/santoku, https://hughjonesd.github.io/santoku/

BugReports https://github.com/hughjonesd/santoku/issues

VignetteBuilder knitr

RdMacros lifecycle

NeedsCompilation yes

Author David Hugh-Jones [aut, cre], Daniel Possenriede [ctb]

Repository CRAN

Date/Publication 2022-06-08 18:00:02 UTC

R topics documented:

santoku-package . . . 2
breaks-class . . . 3
brk_default . . . 4
brk_manual . . . 4
brk_width-for-datetime . . . 5
chop . . . 6
chop_equally . . . 9
chop_evenly . . . 10
chop_mean_sd . . . 11
chop_n . . . 12
chop_pretty . . . 13
chop_proportions . . . 14
chop_quantiles . . . 15
chop_width . . . 16
exactly . . . 17
fillet . . . 18
lbl_dash . . . 19
lbl_discrete . . . 20
lbl_endpoints . . . 21
lbl_glue . . . 23
lbl_intervals . . . 25
lbl_manual . . . 26
lbl_midpoints . . . 27
lbl_seq . . . 28
non-standard-types . . . 29
percent . . . 30

Index **31**

`santoku-package` *A versatile cutting tool for R*

Description

santoku is a tool for cutting data into intervals. It provides the function `chop()`, which is similar to base R's `cut()` or `Hmisc::cut2()`. `chop(x, breaks)` takes a vector `x` and returns a factor of the same length, coding which interval each element of `x` falls into.

Details

Here are some advantages of santoku:

- By default, `chop()` always covers the whole range of the data, so you won't get unexpected `NA` values.

- Unlike `cut()` or `cut2()`, `chop()` can handle single values as well as intervals. For example, `chop(x, breaks = c(1, 2, 2, 3))` will create a separate factor level for values exactly equal to 2.
- Flexible and easy labelling.
- Convenience functions for creating quantile intervals, evenly-spaced intervals or equal-sized groups.
- Convenience functions to quickly tabulate chopped data.
- Can chop numbers, dates, date-times and other objects.

These advantages make santoku especially useful for exploratory analysis, where you may not know the range of your data in advance.

To get started, read the vignette:

```
vignette("santoku")
```

For more details, start with the documentation for `chop()`.

Author(s)

Maintainer: David Hugh-Jones <davidhughjones@gmail.com>

Other contributors:

- Daniel Possenriede <possenriede@gmail.com> [contributor]

See Also

Useful links:

- https://github.com/hughjonesd/santoku
- https://hughjonesd.github.io/santoku/
- Report bugs at https://github.com/hughjonesd/santoku/issues

---

`breaks-class` *Class representing a set of intervals*

Description

Class representing a set of intervals

Usage

```
## S3 method for class 'breaks'
format(x, ...)

## S3 method for class 'breaks'
print(x, ...)

is.breaks(x, ...)
```

Arguments

x	A breaks object
...	Unused

---

brk_default *Create a standard set of breaks*

Description

Create a standard set of breaks

Usage

```
brk_default(breaks)
```

Arguments

breaks	A numeric vector.

Value

A (function which returns an) object of class `breaks`.

Examples

```
chop(1:10, c(2, 5, 8))
chop(1:10, brk_default(c(2, 5, 8)))
```

---

brk_manual *Create a* `breaks` *object manually*

Description

Create a `breaks` object manually

Usage

```
brk_manual(breaks, left_vec)
```

Arguments

breaks	A vector, which must be sorted.
left_vec	A logical vector, the same length as `breaks`. Specifies whether each break is left-closed or right-closed.

Details

All breaks must be closed on exactly one side, like `..., x) [x, ...` (left-closed) or `..., x) [x, ...` (right-closed).

For example, if `breaks = 1:3` and `left = c(TRUE, FALSE, TRUE)`, then the resulting intervals are

```
T        F       T
[ 1,  2 ] ( 2, 3 )
```

Singleton breaks are created by repeating a number in `breaks`. Singletons must be closed on both sides, so if there is a repeated number at indices `i`, `i+1`, `left[i]` *must* be `TRUE` and `left[i+1]` must be `FALSE`.

Value

A (function which returns an) object of class `breaks`.

Examples

```
lbrks <- brk_manual(1:3, rep(TRUE, 3))
chop(1:3, lbrks, extend = FALSE)

rbrks <- brk_manual(1:3, rep(FALSE, 3))
chop(1:3, rbrks, extend = FALSE)

brks_singleton <- brk_manual(
      c(1,    2,    2,     3),
      c(TRUE, TRUE, FALSE, TRUE))

chop(1:3, brks_singleton, extend = FALSE)
```

`brk_width-for-datetime`

Equal-width intervals for dates or datetimes

Description

`brk_width()` can be used with time interval classes from base R or the `lubridate` package.

Usage

```
## S3 method for class 'Duration'
brk_width(width, start)
```

Arguments

`width`	A scalar difftime, Period or Duration object.
`start`	A scalar of class Date or POSIXct. Can be omitted.

Details

If `width` is a Period, `lubridate::add_with_rollback()` is used to calculate the widths. This can be useful for e.g. calendar months.

Examples

```
if (requireNamespace("lubridate")) {
  year2001 <- as.Date("2001-01-01") + 0:364
  tab_width(year2001, months(1),
        labels = lbl_discrete(" to ", fmt = "%e %b %y"))
}
```

---

chop *Cut data into intervals*

Description

`chop()` cuts `x` into intervals. It returns a `factor` of the same length as `x`, representing which interval contains each element of `x`. `kiru()` is an alias for `chop`. `tab()` calls `chop()` and returns a contingency `table()` from the result.

Usage

```
chop(
  x,
  breaks,
  labels = lbl_intervals(),
  extend = NULL,
  left = TRUE,
  close_end = FALSE,
  drop = TRUE
)

kiru(
  x,
  breaks,
  labels = lbl_intervals(),
  extend = NULL,
  left = TRUE,
  close_end = FALSE,
  drop = TRUE
)

tab(
  x,
```

```
  breaks,
  labels = lbl_intervals(),
  extend = NULL,
  left = TRUE,
  close_end = FALSE,
  drop = TRUE
)
```

Arguments

x	A vector.
breaks	A numeric vector of cut-points or a function to create cut-points from x.
labels	A character vector of labels or a function to create labels.
extend	Logical. Extend breaks to +/-Inf?
left	Logical. Left-closed breaks?
close_end	Logical. Close last break at right? (If left is FALSE, close first break at left?)
drop	Logical. Drop unused levels from the result?

Details

x may be a numeric vector, or more generally, any vector which can be compared with < and == (see Ops). In particular Date and date-time objects are supported. Character vectors are supported with a warning.

Breaks:

breaks may be a vector or a function.

If it is a vector, breaks gives the break endpoints. Repeated values create singleton intervals. For example breaks = c(1, 3, 3, 5) creates 3 intervals: [1, 3), {3} and (3, 5].

If breaks is a function, it is called with the x, extend, left and close_end arguments, and should return an object of class breaks. Use brk_* functions to create a variety of data-dependent breaks.

Options for breaks:

By default, left-closed intervals are created. If left is FALSE, right- closed intervals are created.

If close_end is TRUE the end break will be closed at both ends, ensuring that all values x with min(breaks) <= x <= max(breaks) are included in the default intervals.

Using mathematical set notation:

- If left is TRUE and close_end is TRUE, breaks will look like [x1, x2), [x2, x3) ... [x_n-1, x_n].
- If left is FALSE and close_end is TRUE, breaks will look like [x1, x2], (x2, x3] ... (x_n-1, x_n].
- If left is TRUE and close_end is FALSE, all breaks will look like ...[x1, x2) ....
- If left is FALSE and close_end is FALSE, all breaks will look like ...(x1, x2] ....

Extending intervals:

If extend is TRUE, intervals will be extended to [-Inf,min(breaks)) and (max(breaks), Inf].

If `extend` is `NULL` (the default), intervals will be extended to `[min(x), min(breaks))` and `(max(breaks), max(x)]`, *only* if necessary – i.e. if `min(x) < min(breaks)` and `max(x) > max(breaks)` respectively.

Extending intervals, either by `extend = NULL` or `extend = TRUE`, *always* leaves the central, non-extended intervals unchanged. In particular, `close_end` applies to the central intervals, not to the extended ones. For example, if `breaks = c(1, 3, 5)` and `close_end = TRUE`, the resulting breaks will be

`[1, 3), [3, 5]`

and if `extend = TRUE` the result will be

`[-Inf, 1), [1, 3), [3, 5], (5, Inf]`

Labels:

`labels` may be a character vector. It should have the same length as the number of intervals. Alternatively, use a `lbl_*` function such as `lbl_seq()`.

If `labels` is `NULL`, then integer codes will be returned instead of a factor.

Miscellaneous:

`NA` values in `x`, and values which are outside the extended endpoints, return `NA`.

`kiru()` is a synonym for `chop()`. If you load `{tidyr}`, you can use it to avoid confusion with `tidyr::chop()`.

Note that `chop()`, like all of R, uses binary arithmetic. Thus, numbers may not be exactly equal to what you think they should be. There is an example below.

Value

`chop()` returns a `factor` of the same length as `x`, representing the intervals containing the value of `x`.

`tab()` returns a contingency `table()`.

See Also

`base::cut()`, `non-standard-types` for chopping objects that aren't numbers.

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_mean_sd()`, `chop_n()`, `chop_proportions()`, `chop_quantiles()`, `chop_width()`, `fillet()`

Examples

```
chop(1:3, 2)

chop(1:10, c(2, 5, 8))

chop(1:10, c(2, 5, 8), extend = FALSE)

chop(1:10, c(2, 5, 5, 8))

chop(1:10, c(2, 5, 8), left = FALSE)

chop(1:10, c(2, 5, 8), close_end = TRUE)
```

```
chop(1:10, brk_quantiles(c(0.25, 0.75)))

chop(1:10, c(2, 5, 8), labels = lbl_dash())

# floating point inaccuracy:
chop(0.3/3, c(0, 0.1, 0.1, 1), labels = c("< 0.1", "0.1", "> 0.1"))

tab(1:10, c(2, 5, 8))
```

---

chop_equally *Chop equal-sized groups*

Description

chop_equally() chops x into groups with an equal number of elements.

Usage

```
chop_equally(
  x,
  groups,
  ...,
  labels = lbl_intervals(raw = TRUE),
  left = is.numeric(x),
  close_end = TRUE
)

brk_equally(groups)

tab_equally(x, groups, ..., left = is.numeric(x), close_end = TRUE)
```

Arguments

x	A vector.
groups	Number of groups.
...	Passed to chop().
labels	A character vector of labels or a function to create labels.
left	Logical. Left-closed breaks?
close_end	Logical. Close last break at right? (If left is FALSE, close first break at left?)

Value

chop_* functions return a factor of the same length as x.
brk_* functions return a function to create breaks.
tab_* functions return a contingency table().

See Also

Other chopping functions: chop_evenly(), chop_mean_sd(), chop_n(), chop_proportions(), chop_quantiles(), chop_width(), chop(), fillet()

Examples

```
chop_equally(1:10, 5)
```

---

chop_evenly *Chop into equal-width intervals*

Description

chop_evenly() chops x into intervals intervals of equal width.

Usage

```
chop_evenly(x, intervals, ..., close_end = TRUE)

brk_evenly(intervals)

tab_evenly(x, intervals, ..., close_end = TRUE)
```

Arguments

x	A vector.
intervals	Integer: number of intervals to create.
...	Passed to chop().
close_end	Logical. Close last break at right? (If left is FALSE, close first break at left?)

Details

chop_evenly() sets close_end = TRUE by default.

Value

chop_* functions return a factor of the same length as x.
brk_* functions return a function to create breaks.
tab_* functions return a contingency table().

See Also

Other chopping functions: chop_equally(), chop_mean_sd(), chop_n(), chop_proportions(), chop_quantiles(), chop_width(), chop(), fillet()

Examples

```
chop_evenly(0:10, 5)
```

`chop_mean_sd` *Chop by standard deviations*

Description

Intervals are measured in standard deviations on either side of the mean.

Usage

```
chop_mean_sd(x, sds = 1:3, ..., sd = deprecated())

brk_mean_sd(sds = 1:3, sd = deprecated())

tab_mean_sd(x, sds = 1:3, ...)
```

Arguments

`x`	A vector.
`sds`	Positive numeric vector of standard deviations.
`...`	Passed to `chop()`.
`sd`	**[Deprecated]**

Details

In version 0.7.0, these functions changed to specifying `sds` as a vector. To chop 1, 2 and 3 standard deviations around the mean, write `chop_mean_sd(x, sds = 1:3)` instead of `chop_mean_sd(x, sd = 3)`.

Value

`chop_*` functions return a `factor` of the same length as `x`.
`brk_*` functions return a `function` to create `breaks`.
`tab_*` functions return a contingency `table()`.

See Also

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_n()`, `chop_proportions()`, `chop_quantiles()`, `chop_width()`, `chop()`, `fillet()`

Examples

```
chop_mean_sd(1:10)

chop(1:10, brk_mean_sd())

tab_mean_sd(1:10)
```

---

chop_n *Chop into fixed-sized groups*

Description

`chop_n()` creates intervals containing a fixed number of elements. One interval may have fewer elements.

Usage

```
chop_n(x, n, ..., close_end = TRUE)

brk_n(n)

tab_n(x, n, ..., close_end = TRUE)
```

Arguments

`x`	A vector.
`n`	Integer: number of elements in each interval.
`...`	Passed to `chop()`.
`close_end`	Logical. Close last break at right? (If `left` is `FALSE`, close first break at left?)

Details

Note that `chop_n()` sets `close_end = TRUE` by default.

Groups may be larger than `n`, if there are too many duplicated elements in `x`. If so, a warning is given.

Value

`chop_*` functions return a `factor` of the same length as `x`.
`brk_*` functions return a `function` to create `breaks`.
`tab_*` functions return a contingency `table()`.

See Also

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_mean_sd()`, `chop_proportions()`, `chop_quantiles()`, `chop_width()`, `chop()`, `fillet()`

Examples

```
chop_n(1:10, 5)

# too many duplicates
x <- rep(1:2, each = 3)
chop_n(x, 2)

tab_n(1:10, 5)

# fewer elements in one group
tab_n(1:10, 4)
```

---

chop_pretty *Chop using pretty breakpoints*

Description

chop_pretty() uses base::pretty() to calculate breakpoints which are 1, 2 or 5 times a power of 10. These look nice in graphs.

Usage

```
chop_pretty(x, n = 5, ...)

brk_pretty(n = 5, ...)

tab_pretty(x, n = 5, ...)
```

Arguments

x	A vector.
n	Positive integer passed to base::pretty(). How many intervals to chop into?
...	Passed to chop() by chop_pretty() and tab_pretty(); passed to base::pretty() by brk_pretty().

Details

base::pretty() tries to return n+1 breakpoints, i.e. n intervals, but note that this is not guaranteed. There are methods for Date and POSIXct objects.

For fine-grained control over base::pretty() parameters, use chop(x, brk_pretty(...)).

Value

chop_* functions return a factor of the same length as x.
brk_* functions return a function to create breaks.
tab_* functions return a contingency table().

Examples

```
chop_pretty(1:10)

chop(1:10, brk_pretty(n = 5, high.u.bias = 0))

tab_pretty(1:10)
```

`chop_proportions` *Chop into proportions of the range of x*

Description

`chop_proportions()` chops `x` into `proportions` of its range, excluding infinite values.

Usage

```
chop_proportions(x, proportions, ..., labels = lbl_intervals(raw = TRUE))

brk_proportions(proportions)

tab_proportions(x, proportions, ...)
```

Arguments

`x`	A vector.
`proportions`	Numeric vector between 0 and 1: proportions of x's range
`...`	Passed to `chop()`.
`labels`	A character vector of labels or a function to create labels.

Details

By default, labels show the raw numeric endpoints. To label intervals by the proportions, use `labels = lbl_intervals(raw = FALSE)`.

Value

`chop_*` functions return a `factor` of the same length as `x`.
`brk_*` functions return a `function` to create `breaks`.
`tab_*` functions return a contingency `table()`.

See Also

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_mean_sd()`, `chop_n()`, `chop_quantiles()`, `chop_width()`, `chop()`, `fillet()`

Examples

```
chop_proportions(0:10, c(0.2, 0.8))
```

`chop_quantiles` *Chop by quantiles*

Description

`chop_quantiles()` chops data by quantiles. `chop_deciles()` is a convenience shortcut and chops into deciles.

Usage

```
chop_quantiles(x, probs, ..., left = is.numeric(x), close_end = TRUE)

chop_deciles(x, ...)

brk_quantiles(probs, ...)

tab_quantiles(x, probs, ..., left = is.numeric(x), close_end = TRUE)

tab_deciles(x, ...)
```

Arguments

`x`	A vector.
`probs`	A vector of probabilities for the quantiles.
`...`	Passed to `chop()`, or for `brk_quantiles()` to `stats::quantile()`.
`left`	Logical. Left-closed breaks?
`close_end`	Logical. Close last break at right? (If `left` is `FALSE`, close first break at left?)

Details

Note that these functions set `close_end = TRUE` by default. This helps ensure that e.g. `chop_quantiles(x, c(0, 1/3, 2/3,` will split the data into three equal-sized groups.

For non-numeric `x`, `left` is set to `FALSE` by default. This works better for calculating "type 1" quantiles, since they round down. See `stats::quantile()`.

Value

`chop_*` functions return a `factor` of the same length as `x`.
`brk_*` functions return a `function` to create `breaks`.
`tab_*` functions return a contingency `table()`.

See Also

Other chopping functions: chop_equally(), chop_evenly(), chop_mean_sd(), chop_n(), chop_proportions(), chop_width(), chop(), fillet()

Examples

```
chop_quantiles(1:10, 1:3/4)

chop(1:10, brk_quantiles(1:3/4))

chop_deciles(1:10)

# to label by the quantiles themselves:
chop_quantiles(1:10, 1:3/4, lbl_intervals(raw = TRUE))

set.seed(42)
tab_quantiles(rnorm(100), probs = 1:3/4, label = lbl_intervals(raw = TRUE))
```

---

chop_width *Chop into fixed-width intervals*

Description

chop_width() chops x into intervals of fixed width.

Usage

```
chop_width(x, width, start, ..., left = sign(width) > 0)

brk_width(width, start)

## Default S3 method:
brk_width(width, start)

tab_width(x, width, start, ..., left = sign(width) > 0)
```

Arguments

x	A vector.
width	Width of intervals.
start	Starting point for intervals. By default the smallest finite x (largest if width is negative).
...	Passed to chop().
left	Logical. Left-closed breaks?

Details

If `width` is negative, `chop_width()` sets `left = FALSE` and intervals will go downwards from `start`.

Value

`chop_*` functions return a `factor` of the same length as `x`.
`brk_*` functions return a `function` to create `breaks`.
`tab_*` functions return a contingency `table()`.

See Also

brk_width-for-datetime

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_mean_sd()`, `chop_n()`, `chop_proportions()`, `chop_quantiles()`, `chop()`, `fillet()`

Examples

```
chop_width(1:10, 2)

chop_width(1:10, 2, start = 0)

chop_width(1:9, -2)

chop(1:10, brk_width(2, 0))

tab_width(1:10, 2, start = 0)
```

---

exactly *Define singleton intervals explicitly*

Description

`exactly()` duplicates its input. It lets you define singleton intervals like this: `chop(x, c(1, exactly(2), 3))`. This is the same as `chop(x, c(1, 2, 2, 3))` but conveys your intent more clearly.

Usage

```
exactly(x)
```

Arguments

`x`	A numeric vector.

Value

The same as `rep(x, each = 2)`.

Examples

```
chop(1:10, c(2, exactly(5), 8))

# same:
chop(1:10, c(2, 5, 5, 8))
```

`fillet` *Chop data precisely (for programmers)*

Description

Chop data precisely (for programmers)

Usage

```
fillet(x, breaks, labels = lbl_intervals(), left = TRUE, close_end = FALSE)
```

Arguments

`x`	A vector.
`breaks`	A numeric vector of cut-points or a function to create cut-points from `x`.
`labels`	A character vector of labels or a function to create labels.
`left`	Logical. Left-closed breaks?
`close_end`	Logical. Close last break at right? (If `left` is `FALSE`, close first break at left?)

Details

`fillet()` calls `chop()` with `extend = FALSE` and `drop = FALSE`. This ensures that you get only the `breaks` and `labels` you ask for. When programming, consider using `fillet()` instead of `chop()`.

Value

`fillet()` returns a `factor` of the same length as `x`, representing the intervals containing the value of `x`.

See Also

Other chopping functions: `chop_equally()`, `chop_evenly()`, `chop_mean_sd()`, `chop_n()`, `chop_proportions()`, `chop_quantiles()`, `chop_width()`, `chop()`

Examples

```
fillet(1:10, c(2, 5, 8))
```

`lbl_dash` *Label chopped intervals like 1-4, 4-5, ...*

Description

This label style is user-friendly, but doesn't distinguish between left- and right-closed intervals. It's good for continuous data where you don't expect points to be exactly on the breaks.

Usage

```
lbl_dash(
  symbol = em_dash(),
  fmt = NULL,
  single = "{l}",
  first = NULL,
  last = NULL,
  raw = FALSE
)
```

Arguments

`symbol`	String: symbol to use for the dash.
`fmt`	String or function. A format for break endpoints.
`single`	Glue string: label for singleton intervals. See `lbl_glue()` for details.
`first`	Glue string: override label for the first category. Write e.g. `first = "<{r}"` to create a label like `"<18"`. See `lbl_glue()` for details.
`last`	String: override label for the last category. Write e.g. `last = ">{l}"` to create a label like `">65"`. See `lbl_glue()` for details.
`raw`	Logical. Always use raw `breaks` in labels, rather than e.g. quantiles or standard deviations?

Details

If you don't want unicode output, use `lbl_dash("-")`.

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_discrete()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_intervals()`, `lbl_manual()`, `lbl_midpoints()`, `lbl_seq()`

Examples

```
chop(1:10, c(2, 5, 8), lbl_dash())

chop(1:10, c(2, 5, 8), lbl_dash(" to ", fmt = "%.1f"))

chop(1:10, c(2, 5, 8), lbl_dash(first = "<{r}"))

pretty <- function (x) prettyNum(x, big.mark = ",", digits = 1)
chop(runif(10) * 10000, c(3000, 7000), lbl_dash(" to ", fmt = pretty))
```

---

lbl_discrete *Label discrete data*

Description

`lbl_discrete()` creates labels for discrete data, such as integers. For example, breaks `c(1, 3, 4, 6, 7)` are labelled: `"1-2"`, `"3"`, `"4-5"`, `"6-7"`.

Usage

```
lbl_discrete(
  symbol = em_dash(),
  unit = 1,
  fmt = NULL,
  single = NULL,
  first = NULL,
  last = NULL
)
```

Arguments

`symbol`	String: symbol to use for the dash.
`unit`	Minimum difference between distinct values of data. For integers, 1.
`fmt`	String or function. A format for break endpoints.
`single`	Glue string: label for singleton intervals. See `lbl_glue()` for details.
`first`	Glue string: override label for the first category. Write e.g. `first = "<{r}"` to create a label like `"<18"`. See `lbl_glue()` for details.
`last`	String: override label for the last category. Write e.g. `last = ">{l}"` to create a label like `">65"`. See `lbl_glue()` for details.

Details

No check is done that the data are discrete-valued. If they are not, then these labels may be misleading. Here, discrete-valued means that if `x < y`, then `x <= y - unit`.

Be aware that Date objects may have non-integer values. See Date.

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_dash()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_intervals()`, `lbl_manual()`, `lbl_midpoints()`, `lbl_seq()`

Examples

```
tab(1:7, c(1, 3, 5), lbl_discrete())

tab(1:7, c(3, 5), lbl_discrete(first = "<= {r}"))

tab(1:7 * 1000, c(1, 3, 5) * 1000, lbl_discrete(unit = 1000))

# Misleading labels for non-integer data
chop(2.5, c(1, 3, 5), lbl_discrete())
```

---

lbl_endpoints *Label chopped intervals by their left or right endpoints*

Description

This is useful when the left endpoint unambiguously indicates the interval. In other cases it may give errors due to duplicate labels.

Usage

```
lbl_endpoints(
  left = TRUE,
  fmt = NULL,
  single = NULL,
  first = NULL,
  last = NULL,
  raw = FALSE
)

lbl_endpoint(fmt = NULL, raw = FALSE, left = TRUE)
```

Arguments

left	Flag. Use left endpoint or right endpoint?
fmt	String or function. A format for break endpoints.
single	Glue string: label for singleton intervals. See lbl_glue() for details.
first	Glue string: override label for the first category. Write e.g. first = "<{r}" to create a label like "<18". See lbl_glue() for details.
last	String: override label for the last category. Write e.g. last = ">{l}" to create a label like ">65". See lbl_glue() for details.
raw	Logical. Always use raw breaks in labels, rather than e.g. quantiles or standard deviations?

Details

lbl_endpoint() is deprecated. Do not use it.

Value

A function that creates a vector of labels.

Formatting endpoints

If fmt is not NULL then it is used to format the endpoints. If fmt is a string then numeric endpoints will be formatted by sprintf(fmt, breaks); other endpoints, e.g. Date objects, will be formatted by format(breaks, fmt).

If fmt is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the {scales} package, e.g. scales::label_comma().

See Also

Other labelling functions: lbl_dash(), lbl_discrete(), lbl_glue(), lbl_intervals(), lbl_manual(), lbl_midpoints(), lbl_seq()

Examples

```
chop(1:10, c(2, 5, 8), lbl_endpoints(left = TRUE))
chop(1:10, c(2, 5, 8), lbl_endpoints(left = FALSE))
if (requireNamespace("lubridate")) {
  tab_width(
          as.Date("2000-01-01") + 0:365,
         months(1),
         labels = lbl_endpoints(fmt = "%b")
       )
}
```

`lbl_glue` *Label chopped intervals using the glue package*

Description

Use `"{l}"` and `"{r}"` to show the left and right endpoints of the intervals.

Usage

```
lbl_glue(
  label,
  fmt = NULL,
  single = NULL,
  first = NULL,
  last = NULL,
  raw = FALSE,
  ...
)
```

Arguments

`label`	A glue string passed to `glue::glue()`.
`fmt`	String or function. A format for break endpoints.
`single`	Glue string: label for singleton intervals. See `lbl_glue()` for details.
`first`	Glue string: override label for the first category. Write e.g. `first = "<{r}"` to create a label like `"<18"`. See `lbl_glue()` for details.
`last`	String: override label for the last category. Write e.g. `last = ">{l}"` to create a label like `">65"`. See `lbl_glue()` for details.
`raw`	Logical. Always use raw `breaks` in labels, rather than e.g. quantiles or standard deviations?
`...`	Further arguments passed to `glue::glue()`.

Details

The following variables are available in the glue string:

- `l` is a character vector of left endpoints of intervals.
- `r` is a character vector of right endpoints of intervals.
- `l_closed` is a logical vector. Elements are `TRUE` when the left endpoint is closed.
- `r_closed` is a logical vector, `TRUE` when the right endpoint is closed.

Endpoints will be formatted by `fmt` before being passed to `glue()`.

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_endpoints()`, `lbl_intervals()`, `lbl_manual()`, `lbl_midpoints()`, `lbl_seq()`

Examples

```
tab(1:10, c(1, 3, 3, 7),
    label = lbl_glue("{l} to {r}", single = "Exactly {l}"))

tab(1:10 * 1000, c(1, 3, 5, 7) * 1000,
    label = lbl_glue("{l}-{r}", fmt = function(x) prettyNum(x, big.mark=',')))

# reproducing lbl_intervals():
interval_left <- "{ifelse(l_closed, '[', '(')}"
interval_right <- "{ifelse(r_closed, ']', ')')}"
glue_string <- paste0(interval_left, "{l}", ", ", "{r}", interval_right)
tab(1:10, c(1, 3, 3, 7), label = lbl_glue(glue_string, single = "{{{l}}}"))
```

`lbl_intervals` *Label chopped intervals using set notation*

Description

These labels are the most exact, since they show you whether intervals are "closed" or "open", i.e. whether they include their endpoints.

Usage

```
lbl_intervals(
  fmt = NULL,
  single = "{{{l}}}",
  first = NULL,
  last = NULL,
  raw = FALSE
)
```

Arguments

`fmt`	String or function. A format for break endpoints.
`single`	Glue string: label for singleton intervals. See `lbl_glue()` for details.
`first`	Glue string: override label for the first category. Write e.g. `first = "<{r}"` to create a label like `"<18"`. See `lbl_glue()` for details.
`last`	String: override label for the last category. Write e.g. `last = ">{l}"` to create a label like `">65"`. See `lbl_glue()` for details.
`raw`	Logical. Always use raw `breaks` in labels, rather than e.g. quantiles or standard deviations?

Details

Mathematical set notation looks like this:

- `[a, b]`: all numbers `x` where `a <= x <= b`;
- `(a, b)`: all numbers where `a < x < b`;
- `[a, b)`: all numbers where `a <= x < b`;
- `(a, b]`: all numbers where `a < x <= b`;
- `{a}`: just the number `a` exactly.

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_manual()`, `lbl_midpoints()`, `lbl_seq()`

Examples

```
tab(-10:10, c(-3, 0, 0, 3),
      labels = lbl_intervals())

tab_evenly(runif(20), 10,
      labels = lbl_intervals(fmt = percent))
```

---

lbl_manual *Label chopped intervals in a user-defined sequence*

Description

`lbl_manual()` uses an arbitrary sequence to label intervals. If the sequence is too short, it will be pasted with itself and repeated.

Usage

```
lbl_manual(sequence, fmt = "%s")
```

Arguments

`sequence`	A character vector of labels.
`fmt`	String or function. A format for break endpoints.

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_intervals()`, `lbl_midpoints()`, `lbl_seq()`

Examples

```
chop(1:10, c(2, 5, 8), lbl_manual(c("w", "x", "y", "z")))

# if labels need repeating:
chop(1:10, 1:10, lbl_manual(c("x", "y", "z")))
```

---

lbl_midpoints *Label chopped intervals by their midpoints*

Description

This uses the midpoint of each interval for its label.

Usage

```
lbl_midpoints(
  fmt = NULL,
  single = NULL,
  first = NULL,
  last = NULL,
  raw = FALSE
)
```

Arguments

`fmt`	String or function. A format for break endpoints.
`single`	Glue string: label for singleton intervals. See `lbl_glue()` for details.
`first`	Glue string: override label for the first category. Write e.g. `first = "<{r}"` to create a label like `"<18"`. See `lbl_glue()` for details.
`last`	String: override label for the last category. Write e.g. `last = ">{l}"` to create a label like `">65"`. See `lbl_glue()` for details.
`raw`	Logical. Always use raw `breaks` in labels, rather than e.g. quantiles or standard deviations?

Value

A function that creates a vector of labels.

Formatting endpoints

If `fmt` is not `NULL` then it is used to format the endpoints. If `fmt` is a string then numeric endpoints will be formatted by `sprintf(fmt, breaks)`; other endpoints, e.g. Date objects, will be formatted by `format(breaks, fmt)`.

If `fmt` is a function, it should take a vector of numbers (or other objects that can be used as breaks) and return a character vector. It may be helpful to use functions from the `{scales}` package, e.g. `scales::label_comma()`.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_intervals()`, `lbl_manual()`, `lbl_seq()`

Examples

```
chop(1:10, c(2, 5, 8), lbl_midpoints())
```

---

lbl_seq *Label chopped intervals in sequence*

Description

`lbl_seq()` labels intervals sequentially, using numbers or letters.

Usage

```
lbl_seq(start = "a")
```

Arguments

`start`	String. A template for the sequence. See below.

Details

`start` shows the first element of the sequence. It must contain exactly *one* character out of the set "a", "A", "i", "I" or "1". For later elements:

- "a" will be replaced by "a", "b", "c", ...
- "A" will be replaced by "A", "B", "C", ...
- "i" will be replaced by lower-case Roman numerals "i", "ii", "iii", ...
- "I" will be replaced by upper-case Roman numerals "I", "II", "III", ...
- "1" will be replaced by numbers "1", "2", "3", ...

Other characters will be retained as-is.

Value

A function that creates a vector of labels.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_endpoints()`, `lbl_glue()`, `lbl_intervals()`, `lbl_manual()`, `lbl_midpoints()`

Examples

```
chop(1:10, c(2, 5, 8), lbl_seq())

chop(1:10, c(2, 5, 8), lbl_seq("i."))

chop(1:10, c(2, 5, 8), lbl_seq("(A)"))
```

---

non-standard-types *Tips for chopping non-standard types*

Description

Santoku can handle many non-standard types.

Details

- If objects can be compared using <, == etc. then they should be choppable.
- Objects which can't be converted to numeric are handled within R code, which may be slower.
- Character `x` and `breaks` are chopped with a warning.
- If `x` and `breaks` are not the same type, they should be able to be cast to the same type, usually using `vctrs::vec_cast_common()`.
- Not all chopping operations make sense, for example, `chop_mean_sd()` on a character vector.
- For indexed objects such as `stats::ts()` objects, indices will be dropped from the result.
- If you get errors, try setting `extend = FALSE` (but also file a bug report).
- To request support for a type, open an issue on Github.

See Also

brk-width-for-Datetime

`percent` *Simple percentage formatter*

Description

`percent()` formats `x` as a percentage. For a wider range of formatters, consider the scales package.

Usage

```
percent(x)
```

Arguments

`x`	Numeric values.

Value

`x` formatted as a percent.

Examples

```
percent(0.5)
```

Index

∗ **chopping functions**
  chop, 6
  chop_equally, 9
  chop_evenly, 10
  chop_mean_sd, 11
  chop_n, 12
  chop_proportions, 14
  chop_quantiles, 15
  chop_width, 16
  fillet, 18
∗ **labelling functions**
  lbl_dash, 19
  lbl_discrete, 20
  lbl_endpoints, 21
  lbl_glue, 23
  lbl_intervals, 25
  lbl_manual, 26
  lbl_midpoints, 27
  lbl_seq, 28

base::cut(), *8*
base::pretty(), *13*
breaks-class, 3
brk_default, 4
brk_equally *(*chop_equally*)*, 9
brk_evenly *(*chop_evenly*)*, 10
brk_manual, 4
brk_mean_sd *(*chop_mean_sd*)*, 11
brk_n *(*chop_n*)*, 12
brk_pretty *(*chop_pretty*)*, 13
brk_proportions *(*chop_proportions*)*, 14
brk_quantiles *(*chop_quantiles*)*, 15
brk_width *(*chop_width*)*, 16
brk_width-for-datetime, 5, *17*
brk_width.Duration
        *(*brk_width-for-datetime*)*, 5

chop, 6, *10–12*, *14*, *16–18*
chop(), *2*, *3*, *9–16*, *18*
chop_deciles *(*chop_quantiles*)*, 15
chop_equally, *8*, 9, *10–12*, *14*, *16–18*
chop_evenly, *8*, *10*, 10, *11*, *12*, *14*, *16–18*
chop_mean_sd, *8*, *10*, 11, *12*, *14*, *16–18*
chop_mean_sd(), *29*
chop_n, *8*, *10*, *11*, 12, *14*, *16–18*
chop_pretty, 13
chop_proportions, *8*, *10–12*, 14, *16–18*
chop_quantiles, *8*, *10–12*, *14*, 15, *17*, *18*
chop_width, *8*, *10–12*, *14*, *16*, 16, *18*
cut(), *2*

Date, *5*, *7*, *21*
date-time, *7*
difftime, *5*
Duration, *5*

exactly, 17

factor, *6*, *8–15*, *17*, *18*
fillet, *8*, *10–12*, *14*, *16*, *17*, 18
format.breaks *(*breaks-class*)*, 3
function, *9–15*, *17*

glue::glue(), *23*

is.breaks *(*breaks-class*)*, 3

kiru *(*chop*)*, 6

lbl_dash, 19, *21*, *22*, *24*, *26–29*
lbl_discrete, *20*, 20, *22*, *24*, *26–29*
lbl_endpoint *(*lbl_endpoints*)*, 21
lbl_endpoints, *20*, *21*, 21, *24*, *26–29*
lbl_glue, *20–22*, 23, *26–29*
lbl_glue(), *19*, *20*, *22*, *23*, *25*, *27*
lbl_intervals, *20–22*, *24*, 25, *27–29*
lbl_manual, *20–22*, *24*, *26*, 26, *28*, *29*
lbl_midpoints, *20–22*, *24*, *26*, *27*, 27, *29*
lbl_seq, *20–22*, *24*, *26–28*, 28
lbl_seq(), *8*
lubridate::add_with_rollback(), *6*

mathematical set notation, *7*

non-standard-types, 29

Ops, *7*

percent, 30
Period, *5*
POSIXct, *5*
print.breaks *(*breaks-class*)*, 3

santoku *(*santoku-package*)*, 2
santoku-package, 2
scales::label_comma(), *19*, *21*, *22*, *24*, *26–28*
stats::quantile(), *15*
stats::ts(), *29*

tab *(*chop*)*, 6
tab_deciles *(*chop_quantiles*)*, 15
tab_equally *(*chop_equally*)*, 9
tab_evenly *(*chop_evenly*)*, 10
tab_mean_sd *(*chop_mean_sd*)*, 11
tab_n *(*chop_n*)*, 12
tab_pretty *(*chop_pretty*)*, 13
tab_proportions *(*chop_proportions*)*, 14
tab_quantiles *(*chop_quantiles*)*, 15
tab_width *(*chop_width*)*, 16
table(), *6*, *8–15*, *17*

vctrs::vec_cast_common(), *29*