Package 'santoku'

June 16, 2020

Type Package

Title A Versatile Cutting Tool

Version 0.4.1

Author David Hugh-Jones [aut, cre]

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 and times.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

Suggests covr,
testthat (>= 2.1.0),
knitr,
lubridate,
microbenchmark,
purrr,
rmarkdown,
scales,
withr

LinkingTo Rcpp

Imports Rcpp,
assertthat,
lifecycle,
vctrs

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

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

VignetteBuilder knitr

RdMacros lifecycle

R topics documented:

`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 or date-times.

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>

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-left-right *Left- or right-closed breaks*

Description

Questioning

Usage

```
brk_left(breaks)

brk_right(breaks)
```

Arguments

breaks	A numeric vector.

Details

These functions are in the "questioning" stage because they clash with the `left` argument to `chop()` and friends.

These functions override the `left` argument of `chop()`.

Value

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

Examples

```
chop(5:7, brk_left(5:7))

chop(5:7, brk_right(5:7))

chop(5:7, brk_left(5:7))
```

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 numeric 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.

Usage

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

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

Arguments

x	A numeric vector.
breaks	See below.
labels	See below.
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

breaks may be a numeric 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].

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 y with min(x) <= y <= max(x) are included in the default intervals. That is:

- 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] ...

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 in this context, to create a variety of data-dependent breaks.

`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 `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 = FALSE`, *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 they are extended on both ends the result will be e.g.
[-Inf, 1), [1, 3), [3, 5], (5, Inf]

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

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.

[x1, x2) ...‘

- If `left` is `FALSE` and `close_end` is `FALSE`, all breaks will look like ‘...(x1, x2]: R:x1,%20x2)%20...%60%0A*%20If [lbl_seq()]: R:lbl_seq() [3, 5]: R:3,%205 [3, 5]: R:3,%205

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

Value

A factor of the same length as `x`, representing the intervals containing the value of `x`.

See Also

cut

Other chopping functions: `chop_mean_sd()`, `chop_n()`, `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))
```

---

chop_mean_sd *Chop by standard deviations*

Description

Intervals of width 1 standard deviation are included on either side of the mean. The outermost pair of intervals will be shorter if sd is not a whole number.

Usage

```
chop_mean_sd(x, sd = 3, ...)

brk_mean_sd(sd = 3)
```

Arguments

x	A numeric vector.
sd	Positive number: include up to sd standard deviations.
...	Passed to chop().

Value

For chop_* functions, a factor of the same length as x.

See Also

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

Examples

```
chop_mean_sd(1:10)

chop(1:10, brk_mean_sd())
```

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)
```

Arguments

`x`	A numeric vector.
`n`	Integer: number of elements in each interval.
`...`	Passed to `chop()`.
`close_end`	Passed to `chop()`.

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

For chop_* functions, a factor of the same length as `x`.

See Also

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

Examples

```
table(chop_n(1:10, 5))

table(chop_n(1:10, 4))

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

chop_quantiles *Chop by quantiles*

Description

chop_quantiles chops data by quantiles. chop_equally chops data into equal-sized groups. 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, ...)

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

brk_quantiles(probs, ...)

brk_equally(groups)
```

Arguments

x	A numeric vector.
probs	A vector of probabilities for the quantiles.
...	Passed to chop(), or for brk_quantiles to stats::quantile().
left	Passed to chop().
close_end	Passed to chop().
groups	Number of groups.

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, 1) 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

For chop_* functions, a factor of the same length as x.

See Also

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

Examples

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

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

chop_deciles(1:10)

chop_equally(1:10, 5)

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

---

chop_width *Chop into equal-width intervals*

Description

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

Usage

```
chop_width(x, width, start, ...)

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

brk_width(width, start)

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

brk_evenly(intervals)
```

Arguments

`x`	A numeric vector.
`width`	Width of intervals.
`start`	Leftpoint of first interval. By default the lowest finite `x`.
`...`	Passed to `chop()`.
`intervals`	Integer: number of intervals to create.
`groups`	Do not use. **Deprecated**
`close_end`	Passed to `chop()`.

Details

Note that `chop_evenly` sets `close_end = TRUE` by default.

Value

For chop_* functions, a factor of the same length as `x`.

See Also

brk_width-for-datetime

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

Examples

```
chop_width(1:10, 2)

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

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

chop_evenly(0:10, 5)
```

---

exactly *Syntactic sugar*

Description

`exactly` lets you write `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, left = TRUE, close_end = FALSE)
```

Arguments

x	A numeric vector.
breaks	Passed to chop().
labels	Passed to chop().
left	Passed to chop().
close_end	Passed to chop().

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

For chop_* functions, a factor of the same length as x.

See Also

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

Examples

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

knife *Deprecated*

Description

Soft-deprecated knife() is deprecated in favour of purrr::partial().

Usage

```
knife(...)
```

Arguments

...	Parameters for chop().

Value

A function.

`lbl_dash` *Label chopped intervals like 1 - 3, 4 - 5, ...*

Description

This label style is user-friendly, but doesn't distinguish between left- and right-closed intervals.

Usage

```
lbl_dash(symbol = " - ", raw = FALSE, fmt = NULL)
```

Arguments

`symbol`	String: symbol to use for the dash.
`raw`	Logical. Always use raw `breaks` in labels, rather than e.g. quantiles or standard deviations?
`fmt`	A format. Can be a string, passed into `base::sprintf()` or `format()` methods; or a one-argument formatting function.

Value

A vector of labels for `chop`, or a function that creates labels.

See Also

Other labelling functions: `lbl_discrete()`, `lbl_format()`, `lbl_intervals()`, `lbl_manual()`, `lbl_seq()`

Examples

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

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

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

Experimental

Usage

```
lbl_discrete(symbol = " - ", fmt = NULL)
```

Arguments

`symbol`	String: symbol to use for the dash.
`fmt`	A format. Can be a string, passed into `base::sprintf()` or `format()` methods; or a one-argument formatting function.

Details

`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".

No check is done that the data is discrete-valued. If it isn't, then these labels may be misleading. Here, discrete-valued means that if `x < y`, then `x <= y -1`.

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

Value

A vector of labels for chop, or a function that creates labels.

See Also

Other labelling functions: `lbl_dash()`, `lbl_format()`, `lbl_intervals()`, `lbl_manual()`, `lbl_seq()`

Examples

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

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

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

Description

This is useful when the left endpoint unambiguously indicates the interval.

Usage

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

Arguments

fmt	A format. Can be a string, passed into base::sprintf() or format() methods; or a one-argument formatting function.
raw	Logical. Always use raw breaks in labels, rather than e.g. quantiles or standard deviations?
left	Flag. Use left endpoint or right endpoint?

Value

A vector of labels for chop, or a function that creates labels.

Examples

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

lbl_format *Label chopped intervals with arbitrary formatting*

Description

Questioning

Usage

```
lbl_format(fmt, fmt1 = "%.3g", raw = FALSE)
```

Arguments

fmt	A format. Can be a string, passed into base::sprintf() or format() methods; or a one-argument formatting function.
fmt1	Format for breaks consisting of a single value.
raw	Logical. Always use raw breaks in labels, rather than e.g. quantiles or standard deviations?

Details

These labels let you format breaks arbitrarily, using either a string (passed to sprintf()) or a function.

If fmt is a function, it must accept two arguments, representing the left and right endpoints of each interval.

If breaks are non-numeric, you can only use "%s" in a string fmt. breaks will be converted to character in this case.

lbl_format() is in the "questioning" stage. As an alternative, consider using lbl_dash() or lbl_intervals() with the fmt argument.

Value

A vector of labels for chop, or a function that creates labels.

See Also

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

Examples

```
tab(1:10, c(1,3, 3, 7),
      label = lbl_format("%.3g to %.3g"))
tab(1:10, c(1,3, 3, 7),
      label = lbl_format("%.3g to %.3g", "Exactly %.3g"))

percent2 <- function (x, y) {
  sprintf("%.2f%% - %.2f%%", x*100, y*100)
}
tab(runif(100), c(0.25, 0.5, .75),
      labels = lbl_format(percent2))
```

---

lbl_intervals *Label chopped intervals using set notation*

Description

Label chopped intervals using set notation

Usage

```
lbl_intervals(raw = FALSE, fmt = NULL)
```

Arguments

raw	Logical. Always use raw breaks in labels, rather than e.g. quantiles or standard deviations?
fmt	A format. Can be a string, passed into base::sprintf() or format() methods; or a one-argument formatting function.

Details

Mathematical set notation is as follows:

- [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.

Value

A vector of labels for chop, or a function that creates labels.

See Also

Other labelling functions: lbl_dash(), lbl_discrete(), lbl_format(), lbl_manual(), 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	A format. Can be a string, passed into base::sprintf() or format() methods; or a one-argument formatting function.

Value

A vector of labels for chop, or a function that creates labels.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_format()`, `lbl_intervals()`, `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_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.

See Also

Other labelling functions: `lbl_dash()`, `lbl_discrete()`, `lbl_format()`, `lbl_intervals()`, `lbl_manual()`

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)"))
```

percent *Simple formatter*

Description

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)
```

tab *Tabulate data by intervals*

Description

These functions call their related chop_xxx function, and call `table()` on the result.

Usage

```
tab(...)

tab_width(...)

tab_evenly(...)

tab_n(...)

tab_mean_sd(...)
```

Arguments

...	Passed to chop

Value

A `table()`.

Examples

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

tab_mean_sd(1:10)
```

Index