Package ‘santoku’

January 24, 2020

Type Package
Title A Versatile Cutting Tool
Version 0.3.0
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.
License MIT + file LICENSE
Encoding UTF-8
LazyData true
RoxygenNote 7.0.2
Suggests covr, testthat (>= 2.1.0), knitr, microbenchmark, rmarkdown
LinkingTo Rcpp
Imports Rcpp, assertthat
URL https://github.com/hughjonesd/santoku,
    https://hughjonesd.github.io/santoku/
BugReports https://github.com/hughjonesd/santoku/issues
VignetteBuilder knitr
NeedsCompilation yes
Repository CRAN
Date/Publication 2020-01-24 17:10:02 UTC

R topics documented:

breaks-class ..................................................... 2
brk-left-right ................................................... 3
brk_manual ......................................................... 4
brk_mean_sd ....................................................... 5
breaks-class

Class representing a set of intervals

Description

Class representing a set of intervals

Usage

```r
## 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**

Left- or right-closed breaks

**Usage**

```r
brk_left(breaks, close_end = TRUE)

brk_right(breaks, close_end = TRUE)
```

**Arguments**

- `breaks`: A numeric vector or a function.
- `close_end`: Logical: close the rightmost endpoint (`brk_left()`) / leftmost endpoint (`brk_right()`)?

**Details**

`brk_left` and `brk_right` can be used to wrap another `brk_*` function.

**Value**

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

**Examples**

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

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

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

# wrapping another `brk_*` function:
chop(1:10, brk_right(brk_quantiles(1:3/4)))
```
Create a breaks object manually

Usage

brk_manual(breaks, left)

Arguments

breaks A numeric vector which must be sorted.
left A logical vector, the same length as breaks. Is break left-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

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 2]</td>
<td>(2, 3)</td>
<td></td>
</tr>
</tbody>
</table>

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_mean_sd**  

**Chop by standard deviations**

**Description**

Chop by standard deviations.

**Usage**

```r
brk_mean_sd(sd = 3)
```

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

**Arguments**

- `sd`  
  Whole number: include sd standard deviations on each side of the mean.

- `x`  
  A numeric vector.

- `...`  
  Passed to `chop`.

**Value**

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

**See Also**

Other chopping functions: `brk_n()`, `brk_quantiles()`, `brk_width()`, `chop()`, `fillet()`

**Examples**

```r
chop_mean_sd(1:10)
```

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

---

**brk_n**  

**Chop into fixed-sized groups**

**Description**

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

**Usage**

```r
brk_n(n)
```

```r
chop_n(x, n, ...)
```
Arguments

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

Value

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

See Also

Other chopping functions: `brk_mean_sd()`, `brk_quantiles()`, `brk_width()`, `chop()`, `fillet()`

Examples

```r
table(chop_n(1:10, 5))
table(chop_n(1:10, 4))
```

---

brk_quantiles  
Chop by quantiles

Description

`chopping_by_quants` chops data by quantiles. `chop_equality` chops data into equal-sized groups. `chopping_by_deciles` is a convenience shortcut and chops into deciles.

Usage

- `brk_quantiles(probs, ...)`
- `chop_quantiles(x, probs, ...)`
- `chop_deciles(x, ...)`
- `chop_equally(x, groups, ...)`

Arguments

- `probs` A vector of probabilities for the quantiles.
- `...` Arguments passed to `quantile()`.
- `x` A numeric vector.
- `groups` Number of groups.
Value

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

See Also

Other chopping functions: brk_mean_sd(), brk_n(), brk_width(), chop(), fillet()

Examples

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

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

c chop_deciles(1:10)

c chop_equally(1:10, 5)

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

---

**brk_width**  
*Chop into equal-width intervals*

Description

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

Usage

```r
brk_width(width, start)
```

```r
brk_evenly(groups)
```

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

```r
c chop_evenly(x, groups, ...)
```

Arguments

- `width` Width of intervals.
- `start` Leftpoint of first interval. By default the lowest finite x.
- `groups` Integer: number of intervals to create.
- `x` A numeric vector.
- `...` Passed to chop.
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

\[
\text{chop}(x, \text{breaks}, \text{labels}, \text{extend} = \text{NULL}, \text{drop} = \text{TRUE})
\]

\[
\text{kiru}(x, \text{breaks}, \text{labels}, \text{extend} = \text{NULL}, \text{drop} = \text{TRUE})
\]

Arguments

- \( x \) A numeric vector.
- \( \text{breaks} \) See below.
- \( \text{labels} \) See below.
- \( \text{extend} \) Logical. Extend breaks to +/-Inf?
- \( \text{drop} \) 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]. Default breaks are left-closed except for the last interval; for right-closed breaks see `brk_right()`.

If breaks is a function it is called with a single argument, x, and returns an object of class 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.

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.

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: `brk_mean_sd()`, `brk_n()`, `brk_quantiles()`, `brk_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, 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))
**Description**

exactly lets you write \( \text{chop}(x, \text{c}(1, \text{exactly}(2), 3)) \). This is the same as \( \text{chop}(x, \text{c}(1, 2, 2, 3)) \) but conveys your intent more clearly.

**Usage**

\( \text{exactly}(x) \)

**Arguments**

- \( x \)  
  A numeric vector.

**Value**

The same as \( \text{rep}(x, \text{each} = 2) \).

**Examples**

\[
\text{chop}(1:10, \text{c}(2, \text{exactly}(5), 8))
\]

# same:
\[
\text{chop}(1:10, \text{c}(2, 5, 5, 8))
\]

---

**Description**

Chop data precisely (for programmers)

**Usage**

\( \text{fillet}(x, \text{breaks}, \text{labels}) \)

**Arguments**

- \( x \)  
  A numeric vector.
- \( \text{breaks} \)  
  See below.
- \( \text{labels} \)  
  See below.
knife

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

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

See Also

Other chopping functions: brk_mean_sd(), brk_n(), brk_quantiles(), brk_width(), chop()

Examples

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

knife Create a chopping function

Description

knife() returns a one-argument function for chopping data.

Usage

knife(...)  

Arguments

...  

Parameters for chop().

Value

A function.

Examples

chop_six <- knife(breaks = -2:2, labels = lbl_seq("A."))
chop_six(rnorm(10))
chop_six(rnorm(10))
**lbl_dash**

*Labels like 1 - 3, 4 - 5,...*

**Description**

Labels like 1 - 3, 4 - 5,...

**Usage**

```r
lbl_dash(symbol = " - ", raw = FALSE)
```

**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.

**Value**

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

**See Also**

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

**Examples**

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

---

**lbl_format**

*Labels using breaks, with arbitrary formatting*

**Description**

Labels using breaks, with arbitrary formatting

**Usage**

```r
lbl_format(fmt, fmt1 = "%s", raw = FALSE)
```
lbl_intervals

Arguments

fmt A `sprintf()`-style format.
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

If `raw = FALSE`, breaks will be preformatted as strings before being passed to `sprintf()`, so only "%s" should be used in format strings.

Value

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

See Also

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

Examples

```r
tab(1:10, c(1, 3, 3, 7), label = lbl_format("%s to %s\n"))
tab(1:10, c(1, 3, 3, 7), label = lbl_format("%s to %s", "Exactly %s\n"))
```

---

lbl_intervals  
Labels using set notation

Description

Labels using set notation

Usage

```r
lbl_intervals(raw = FALSE)
```

Arguments

raw Logical. Always use raw breaks in labels, rather than e.g. quantiles or standard deviations.

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_format()`, `lbl_manual()`, `lbl_seq()`

Examples

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

lbl_manual
Label manually in 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 `sprintf()`-style format.

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_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 sequentially*

**Description**

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

**Usage**

```r
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_format()`, `lbl_intervals()`, `lbl_manual()`

**Examples**

```r
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)"))
```
Description
These functions call their related chop_xxx function, and call \texttt{table()} on the result.

Usage
\begin{verbatim}
tab(...)  
tab_width(...)  
tab_evenly(...)  
tab_n(...)  
tab_mean_sd(...)  
\end{verbatim}

Arguments
\begin{verbatim}
... Passed to chop  
\end{verbatim}

Value
A \texttt{table()}.  

Examples
\begin{verbatim}
tab(1:10, c(2, 5, 8))  
tab_mean_sd(1:10)  
\end{verbatim}
Index

breaks-class, 2
brk-left-right, 3
brk_evenly (brk_width), 7
brk_left (brk-left-right), 3
brk_manual, 4
brk_mean_sd, 5, 6–9, 11
brk_n, 5, 7–9, 11
brk_quantiles, 5, 6, 6, 8, 9, 11
brk_right (brk-left-right), 3
brk_right(), 9
brk_width, 5–7, 7, 9, 11

chop, 5–8, 8, 11
chop(), 11
chop_deciles (brk_quantiles), 6
chop_equally (brk_quantiles), 6
chop_evenly (brk_width), 7
chop_mean_sd (brk_mean_sd), 5
chop_n (brk_n), 5
chop_quantiles (brk_quantiles), 6
chop_width (brk_width), 7

exactly, 10

factor, 9, 11
fillet, 5–9, 10
format.breaks (breaks-class), 2

is.breaks (breaks-class), 2

kiru (chop), 8
knife, 11

lbl_dash, 12, 13–15
lbl_format, 12, 12, 14, 15
lbl_intervals, 12, 13, 13, 14, 15
lbl_manual, 12–14, 14, 15
lbl_seq, 12–14, 15
lbl_seq(), 9

print.breaks (breaks-class), 2

quantile(), 6

sprintf(), 13, 14

tab, 16
tab_evenly (tab), 16
tab_mean_sd (tab), 16
tab_n (tab), 16
tab_width (tab), 16
table(), 16