Package ‘scales’

May 11, 2020

Title  Scale Functions for Visualization

Version  1.1.1

Description  Graphical scales map data to aesthetics, and provide methods for automatically determining breaks and labels for axes and legends.

License  MIT + file LICENSE


BugReports  https://github.com/r-lib/scales/issues

Depends  R (>= 3.2)

Imports  farver (>= 2.0.3), labeling, lifecycle, munsell (>= 0.5), R6, RColorBrewer, viridisLite

Suggests  bit64, covr, dichromat, ggplot2, hms (>= 0.5.0), testthat (>= 2.1.0)

Encoding  UTF-8

LazyLoad  yes

Roxygen  list(markdown = TRUE, r6 = FALSE)

RoxygenNote  7.1.0

R topics documented:

alpha .................................................. 3
area_pal ............................................. 3
asn_trans .......................................... 4
atanh_trans ........................................ 4
boxcox_trans ...................................... 4
breaks_extended ................................. 5
### R topics documented:

- `breaks_log` .................................................. 6
- `breaks_pretty` ............................................ 7
- `breaks_width` ............................................. 8
- `brewer_pal` ............................................... 8
- `col2hcl` .................................................. 9
- `colour_ramp` ............................................... 10
- `col_numeric` .............................................. 11
- `cscale` .................................................... 13
- `date_trans` ............................................... 14
- `dichromat_pal` .......................................... 14
- `div_gradient_pal` ....................................... 15
- `dscale` ................................................... 16
- `expand_range` ............................................ 16
- `exp_trans` ................................................ 17
- `gradient_n_pal` ......................................... 17
- `grey_pal` .................................................. 18
- `hms_trans` ................................................ 18
- `hue_pal` ................................................... 19
- `identity_pal` ............................................. 19
- `identity_trans` .......................................... 20
- `label_bytes` ............................................. 20
- `label_date` ................................................ 21
- `label_dollar` ............................................ 22
- `label_number` ............................................ 24
- `label_number_auto` .................................... 27
- `label_number_si` ....................................... 27
- `label_ordinal` ........................................... 28
- `label_parse` ............................................. 30
- `label_percent` .......................................... 31
- `label_pvalue` .......................................... 33
- `label_scientific` ....................................... 34
- `label_wrap` .............................................. 36
- `linetype_pal` ............................................. 37
- `log_trans` ................................................ 37
- `manual_pal` .............................................. 38
- `minor_breaks_width` .................................. 38
- `muted` ..................................................... 39
- `oob` ....................................................... 39
- `probability_trans` .................................... 41
- `Range` ..................................................... 42
- `reciprocal_trans` ...................................... 42
- `rescale` ................................................... 42
- `rescale_max` ............................................. 43
- `rescale_mid` ............................................. 44
- `rescale_none` ............................................ 45
- `rescale_pal` .............................................. 45
- `reverse_trans` .......................................... 45
- `seq_gradient_pal` ...................................... 46
- `shape_pal` ................................................ 46
- `sqrt_trans` ............................................... 47
- `time_trans` ............................................... 47
- `train_continuous` ...................................... 48
alpha

Modify colour transparency

Description
Vectorised in both colour and alpha.

Usage
alpha(colour, alpha = NA)

Arguments
colourcolour
alpha new alpha level in [0,1]. If alpha is NA, existing alpha values are preserved.

Examples
alpha("red", 0.1)
alpha(colours(), 0.5)
alpha("red", seq(0, 1, length.out = 10))
alpha(c("first" = "gold", "second" = "lightgray", "third" = "#cd7f32"), .5)

area_pal

Area palettes (continuous)

Description
Area palettes (continuous)

Usage
area_pal(range = c(1, 6))

abs_area(max)

Arguments
range Numeric vector of length two, giving range of possible sizes. Should be greater than 0.
max A number representing the maximum size.
asn_trans  

**Arc-sin square root transformation**

**Description**

This is the variance stabilising transformation for the binomial distribution.

**Usage**

asn_trans()

**Examples**

plot(asn_trans(), xlim = c(0, 1))

atanh_trans  

**Arc-tangent transformation**

**Description**

Arc-tangent transformation

**Usage**

atanh_trans()

**Examples**

plot(atanh_trans(), xlim = c(-1, 1))

boxcox_trans  

**Box-Cox & modulus transformations**

**Description**

The Box-Cox transformation is a flexible transformation, often used to transform data towards normality. The modulus transformation generalises Box-Cox to also work with negative values.

**Usage**

boxcox_trans(p, offset = 0)

modulus_trans(p, offset = 1)

**Arguments**

- **p**  
  Transformation exponent, $\lambda$.

- **offset**  
  Constant offset. 0 for Box-Cox type 1, otherwise any non-negative constant (Box-Cox type 2). modulus_trans() sets the default to 1.
Details

The Box-Cox power transformation (type 1) requires strictly positive values and takes the following form for \( y > 0 \):

\[
y^{(\lambda)} = \frac{y^\lambda - 1}{\lambda}
\]

When \( y = 0 \), the natural log transform is used.

The modulus transformation implements a generalisation of the Box-Cox transformation that works for data with both positive and negative values. The equation takes the following forms, when \( y \neq 0 \):

\[
y^{(\lambda)} = \text{sign}(y) \times (|y| + 1)^\lambda - 1
\]

and when \( y = 0 \):

\[
y^{(\lambda)} = \text{sign}(y) \times \ln(|y| + 1)
\]

References


See Also

\( yj\_trans() \)

Examples

```r
plot(boxcox_trans(-1), xlim = c(0, 10))
plot(boxcox_trans(0), xlim = c(0, 10))
plot(boxcox_trans(1), xlim = c(0, 10))
plot(boxcox_trans(2), xlim = c(0, 10))

plot(modulus_trans(-1), xlim = c(-10, 10))
plot(modulus_trans(0), xlim = c(-10, 10))
plot(modulus_trans(1), xlim = c(-10, 10))
plot(modulus_trans(2), xlim = c(-10, 10))
```

breaks_extended

Automatic breaks for numeric axes

Description

Uses Wilkinson’s extended breaks algorithm as implemented in the `labeling` package.

Usage

```r
breaks_extended(n = 5, ...)
```
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Desired number of breaks. You may get slightly more or fewer breaks than requested.</td>
</tr>
<tr>
<td>...</td>
<td>other arguments passed on to \texttt{labeling::extended()}</td>
</tr>
</tbody>
</table>

**References**


**Examples**

```r
demo_continuous(c(0, 10))
demo_continuous(c(0, 10), breaks = breaks_extended(3))
demo_continuous(c(0, 10), breaks = breaks_extended(10))
```

**Description**

This algorithm starts by looking for integer powers of base. If that doesn’t provide enough breaks, it then looks for additional intermediate breaks which are integer multiples of integer powers of base. If that fails (which it can for very small ranges), we fall back to \texttt{extended_breaks()}.

**Usage**

```r
breaks_log(n = 5, base = 10)
```

**Arguments**

- \texttt{n}: desired number of breaks
- \texttt{base}: base of logarithm to use

**Details**

The algorithm starts by looking for a set of integer powers of base that cover the range of the data. If that does not generate at least \( n - 2 \) breaks, we look for an integer between 1 and base that splits the interval approximately in half. For example, in the case of base = 10, this integer is 3 because \( \log_{10}(3) = 0.477 \). This leaves 2 intervals: \( c(1, 3) \) and \( c(3, 10) \). If we still need more breaks, we look for another integer that splits the largest remaining interval (on the log-scale) approximately in half. For base = 10, this is 5 because \( \log_{10}(5) = 0.699 \).

The generic algorithm starts with a set of integers \texttt{steps} containing only 1 and a set of candidate integers containing all integers larger than 1 and smaller than base. Then for each remaining candidate integer \( x \), the smallest interval (on the log-scale) in the vector \( \text{sort}(c(x, \text{steps}, \text{base})) \) is calculated. The candidate \( x \) which yields the largest minimal interval is added to \texttt{steps} and removed from the candidate set. This is repeated until either a sufficient number of breaks, \( \geq n - 2 \), are returned or all candidates have been used.
**breaks_pretty**

Pretty breaks for date/times

**Description**

Uses default R break algorithm as implemented in `pretty()`. This is primarily useful for date/times, as `extended_breaks()` should do a slightly better job for numeric scales.

**Usage**

```r
breaks_pretty(n = 5, ...)
```

**Arguments**

- `n` Desired number of breaks. You may get slightly more or fewer breaks than requested.
- `...` other arguments passed on to `pretty()`

**Details**

`pretty_breaks()` is retired; use `breaks_pretty()` instead.

**Examples**

```r
one_month <- as.POSIXct(c("2020-05-01", "2020-06-01"))
demo_datetime(one_month)  
demo_datetime(one_month, breaks = breaks_pretty(2))
demo_datetime(one_month, breaks = breaks_pretty(4))

# Tightly spaced date breaks often need custom labels too
demo_datetime(one_month, breaks = breaks_pretty(12))
demo_datetime(one_month, breaks = breaks_pretty(12),
             labels = label_date_short())
```
**breaks_width**  
*Equally spaced breaks*

**Description**

Useful for numeric, date, and date-time scales.

**Usage**

`breaks_width(width, offset = 0)`

**Arguments**

- `width`  
Distance between each break. Either a number, or for date/times, a single string of the form "n unit", e.g. "1 month", "5 days". Unit can be of one "sec", "min", "hour", "day", "week", "month", "year".

- `offset`  
Use if you don’t want breaks to start at zero

**Examples**

```r
demo_continuous(c(0, 100))
demo_continuous(c(0, 100), breaks = breaks_width(10))
demo_continuous(c(0, 100), breaks = breaks_width(20, -4))
demo_continuous(c(0, 100), breaks = breaks_width(20, 4))
```

# This is also useful for dates
```r
one_month <- as.POSIXct(c("2020-05-01", "2020-06-01"))
demo_datetime(one_month)
demo_datetime(one_month, breaks = breaks_width("1 week"))
demo_datetime(one_month, breaks = breaks_width("5 days"))
```

# This is so useful that `scale_x_datetime()` has a shorthand:
```r
demo_datetime(one_month, date_breaks = "5 days")
```

# hms times also work
```r
one_hour <- hms::hms(hours = 0:1)
demo_time(one_hour)
demo_time(one_hour, breaks = breaks_width("15 min"))
demo_time(one_hour, breaks = breaks_width("600 sec"))
```

**brewer_pal**  
*Colour Brewer palette (discrete)*

**Description**

Colour Brewer palette (discrete)

**Usage**

`brewer_pal(type = "seq", palette = 1, direction = 1)`
Arguments

type One of seq (sequential), div (diverging) or qual (qualitative)
palette If a string, will use that named palette. If a number, will index into the list of palettes of appropriate type
direction Sets the order of colours in the scale. If 1, the default, colours are as output by RColorBrewer::brewer_pal(). If -1, the order of colours is reversed.

References

http://colorbrewer2.org

Examples

show_col(brewer_pal()$10))
show_col(brewer_pal("div")(5))
show_col(brewer_pal(palette = "Greens")(5))

# Can use with gradient_n to create a continous gradient
cols <- brewer_pal("div")(5)
show_col(gradients_pal(cols)(seq(0, 1, length.out = 30)))

col2hcl Modify standard R colour in hcl colour space.

Description

Transforms rgb to hcl, sets non-missing arguments and then backtransforms to rgb.

Usage

col2hcl(colour, h = NULL, c = NULL, l = NULL, alpha = NULL)

Arguments

colour character vector of colours to be modified
h Hue, [0, 360]
c Chroma, [0, 100]
l Luminance, [0, 100]
alpha Alpha, [0, 1].

Examples

reds <- rep("red", 6)
show_col(col2hcl(reds, h = seq(0, 180, length = 6)))
show_col(col2hcl(reds, c = seq(0, 80, length = 6)))
show_col(col2hcl(reds, l = seq(0, 100, length = 6)))
show_col(col2hcl(reds, alpha = seq(0, 1, length = 6)))
colour_ramp

\[ \text{Fast colour interpolation} \]

**Description**

Returns a function that maps the interval \([0,1]\) to a set of colours. Interpolation is performed in the CIELAB colour space. Similar to `colorRamp(space = 'Lab')`, but hundreds of times faster, and provides results in "#RRGGBB" (or "#RRGGBBAA") character form instead of RGB colour matrices.

**Usage**

```r
colour_ramp(colors, na.color = NA, alpha = TRUE)
```

**Arguments**

- **colors**: Colours to interpolate; must be a valid argument to `grDevices::col2rgb()`. This can be a character vector of "#RRGGBB" or "#RRGGBBAA", colour names from `grDevices::colors()`, or a positive integer that indexes into `grDevices::palette()`.
- **na.color**: The colour to map to NA values (for example, "#666666" for dark grey, or "#00000000" for transparent) and values outside of \([0,1]\). Can itself by NA, which will simply cause an NA to be inserted into the output.
- **alpha**: Whether to include alpha transparency channels in interpolation. If TRUE then the alpha information is included in the interpolation. The returned colours will be provided in "#RRGGBBAA" format when needed, i.e., in cases where the colour is not fully opaque, so that the "AA" part is not equal to "FF". Fully opaque colours will be returned in "#RRGGBB" format. If FALSE, the alpha information is discarded before interpolation and colours are always returned as "#RRGGBB".

**Value**

A function that takes a numeric vector and returns a character vector of the same length with RGB or RGBA hex colours.

**See Also**

`colorRamp`

**Examples**

```r
ramp <- colour_ramp(c("red", "green", "blue"))
show_col(ramp(seq(0, 1, length = 12)))
```
Description

Conveniently maps data values (numeric or factor/character) to colours according to a given palette, which can be provided in a variety of formats.

Usage

```r
col_numeric(  
  palette,  
  domain,  
  na.color = "#808080",  
  alpha = FALSE,  
  reverse = FALSE  
)

col_bin(  
  palette,  
  domain,  
  bins = 7,  
  pretty = TRUE,  
  na.color = "#808080",  
  alpha = FALSE,  
  reverse = FALSE,  
  right = FALSE  
)

col_quantile(  
  palette,  
  domain,  
  n = 4,  
  probs = seq(0, 1, length.out = n + 1),  
  na.color = "#808080",  
  alpha = FALSE,  
  reverse = FALSE,  
  right = FALSE  
)

col_factor(  
  palette,  
  domain,  
  levels = NULL,  
  ordered = FALSE,  
  na.color = "#808080",  
  alpha = FALSE,  
  reverse = FALSE  
)
```
Arguments

palette The colours or colour function that values will be mapped to
domain The possible values that can be mapped.
For col_numeric and col_bin, this can be a simple numeric range (e.g. c(0, 100));
col_quantile needs representative numeric data; and col_factor needs categorial data.
If NULL, then whenever the resulting colour function is called, the x value will represent the domain. This implies that if the function is invoked multiple times, the encoding between values and colours may not be consistent; if consistency is needed, you must provide a non-NULL domain.

na.color The colour to return for NA values. Note that na.color = NA is valid.
alpha Whether alpha channels should be respected or ignored. If TRUE then colors without explicit alpha information will be treated as fully opaque.
reverse Whether the colors (or color function) in palette should be used in reverse order. For example, if the default order of a palette goes from blue to green, then reverse = TRUE will result in the colors going from green to blue.
bins Either a numeric vector of two or more unique cut points or a single number (greater than or equal to 2) giving the number of intervals into which the domain values are to be cut.
pretty Whether to use the function pretty() to generate the bins when the argument bins is a single number. When pretty = TRUE, the actual number of bins may not be the number of bins you specified. When pretty = FALSE, seq() is used to generate the bins and the breaks may not be "pretty".
right parameter supplied to base::cut(). See Details
n Number of equal-size quantiles desired. For more precise control, use the probs argument instead.
probs See stats::quantile(). If provided, the n argument is ignored.
levels An alternate way of specifying levels; if specified, domain is ignored
ordered If TRUE and domain needs to be coerced to a factor, treat it as already in the correct order

Details

col_numeric is a simple linear mapping from continuous numeric data to an interpolated palette.
col_bin also maps continuous numeric data, but performs binning based on value (see the base::cut() function). col_bin defaults for the cut function are include.lowest = TRUE and right = FALSE.
col_quantile similarly bins numeric data, but via the stats::quantile() function.
col_factor maps factors to colours. If the palette is discrete and has a different number of colours than the number of factors, interpolation is used.
The palette argument can be any of the following:

1. A character vector of RGB or named colours. Examples: palette(), c("#000000","#0000FF","#FFFFFF"),
topo.colors(10)
2. The name of an RColorBrewer palette, e.g. "BuPu" or "Greens".
3. The full name of a viridis palette: "viridis", "magma", "inferno", or "plasma".
4. A function that receives a single value between 0 and 1 and returns a colour. Examples: colorRamp(c("#000000","#FFFFFF"),interpolate="spline").
Value

A function that takes a single parameter \( x \); when called with a vector of numbers (except for \text{col_factor}, which expects factors/characters), \#RRGGBB colour strings are returned (unless \( \text{alpha} = \text{TRUE} \) in which case \#RRGGBBAA may also be possible).

Examples

```r
pal <- \text{col_bin("Greens", domain = 0:100)}
\text{show_col(pal(sort(runif(10, 60, 100))))}
```

# Exponential distribution, mapped continuously
```
\text{show_col(col_numeric("Blues", domain = NULL)(sort(rexp(16))))}
```

# Exponential distribution, mapped by interval
```
\text{show_col(col_bin("Blues", domain = NULL, bins = 4)(sort(rexp(16))))}
```

# Exponential distribution, mapped by quantile
```
\text{show_col(col_quanitle("Blues", domain = NULL)(sort(rexp(16))))}
```

# Categorical data; by default, the values being coloured span the gamut...
```
\text{show_col(col_factor("RdY1Bu", domain = NULL)(LETTERS[1:5]))}
```

# ...unless the data is a factor, without droplevels...
```
\text{show_col(col_factor("RdY1Bu", domain = NULL)(factor(LETTERS[1:5], levels=LETTERS)))}
```

# ...or the domain is stated explicitly.
```
\text{show_col(col_factor("RdY1Bu", levels = LETTERS)(LETTERS[1:5]))}
```

---

cscale

Continuous scale

Description

Continuous scale

Usage

```
cscale(x, palette, na.value = NA_real_, trans = identity_trans())
```

Arguments

\( x \) vector of continuous values to scale

\text{palette} palette to use.

Built in palettes:
- \text{area_pal, brewer_pal, dichromat_pal, div_gradient_pal, gradient_n_pal, grey_pal, hue_pal, identity_pal, linetype_pal, manual_pal, rescale_pal, seq_gradient_pal, shape_pal, viridis_pal}

\text{na.value} value to use for missing values

\text{trans} transformation object describing the how to transform the raw data prior to scaling. Defaults to the identity transformation which leaves the data unchanged.

Built in transformations:
- \text{asn_trans, atanh_trans, boxcox_trans, date_trans, exp_trans, hms_trans, identity_trans, log10_trans, log1p_trans, log2_trans, log_trans, logit_trans, modulus_trans, probability_trans, probit_trans, pseudo_log_trans, reciprocal_trans, reverse_trans, sqrt_trans, time_trans, yj_trans}.
**Examples**

```r
with(mtcars, plot(disp, mpg, cex = cscale(hp, rescale_pal())))
with(mtcars, plot(disp, mpg, cex = cscale(hp, rescale_pal(),
    trans = sqrt_trans())))
with(mtcars, plot(disp, mpg, cex = cscale(hp, area_pal())))
with(mtcars, plot(disp, mpg, pch = 20, cex = 5,
    col = cscale(hp, seq_gradient_pal("grey80", "black"))))
```

---

**dichromat_pal**  
*Dichromat (colour-blind) palette (discrete)*

**Description**

Dichromat (colour-blind) palette (discrete)

**Usage**

```r
dichromat_pal(name)
```

**Arguments**

- `name`  
  Name of colour palette. One of:  
  BbluegreyB,  
  BpurplegreyB,  
  Bluetogreen,  
  LightBluetoDarkBlue,  
  SteppedSequential,  
  Categorical,  
  GreentoMagenta,  
  BrownToBlack,  
  BrownToBlue,  
  BluetoDarkOrange,  
  BluetoOrange,  
  DarkRedtoBlue,  
  LightRedtoBlue,  
  BluetoGreen,  
  BluetoGray,  
  BluetoOrangeRed,  
  BluetoOrange,  
  BlueGreyRed,  
  BluetoOrangeGray
Examples

if (requireNamespace("dichromat", quietly = TRUE)) {
  show_col(dichromat_pal("BluetoOrange.10")(10))
  show_col(dichromat_pal("BluetoOrange.10")(5))

  # Can use with gradient_n to create a continous gradient
  cols <- dichromat_pal("DarkRedtoBlue.12")(12)
  show_col(gradient_n_pal(cols)(seq(0, 1, length.out = 30)))
}

---

**div_gradient_pal**

Diverging colour gradient (continuous).

Description

Diverging colour gradient (continuous).

Usage

```
div_gradient_pal(
  low = mns1("10B 4/6"),
  mid = mns1("N 8/0"),
  high = mns1("10R 4/6"),
  space = "Lab"
)
```

Arguments

- **low**: colour for low end of gradient.
- **mid**: colour for mid point
- **high**: colour for high end of gradient.
- **space**: colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

Examples

```
x <- seq(-1, 1, length.out = 100)
r <- sort(outer(x^2, x^2, "+"))
image(r, col = div_gradient_pal()(seq(0, 1, length.out = 12)))
image(r, col = div_gradient_pal()(seq(0, 1, length.out = 30)))
image(r, col = div_gradient_pal()(seq(0, 1, length.out = 100)))

library(munsell)
image(r, col = div_gradient_pal(low =
  mns1(complement("10R 4/6"), fix = TRUE))(seq(0, 1, length = 100)))
```
dscale  

**Discrete scale**

**Description**

Discrete scale

**Usage**

```r
dscale(x, palette, na.value = NA)
```

**Arguments**

- `x` vector of discrete values to scale
- `palette` aesthetic palette to use
- `na.value` aesthetic to use for missing values

**Examples**

```r
with(mtcars, plot(disp, mpg, pch = 20, cex = 3,
    col = dscale(factor(cyl), brewer_pal() )))
```

---

**expand_range**

*Expand a range with a multiplicative or additive constant*

**Description**

Expand a range with a multiplicative or additive constant

**Usage**

```r
expand_range(range, mul = 0, add = 0, zero_width = 1)
```

**Arguments**

- `range` range of data, numeric vector of length 2
- `mul` multiplicative constant
- `add` additive constant
- `zero_width` distance to use if range has zero width
exp_trans

Description
Exponential transformation (inverse of log transformation)

Usage
exp_trans(base = exp(1))

Arguments
base Base of logarithm

Examples
plot(exp_trans(0.5), xlim = c(-2, 2))
plot(exp_trans(1), xlim = c(-2, 2))
plot(exp_trans(2), xlim = c(-2, 2))
plot(exp_trans(), xlim = c(-2, 2))

gradient_n_pal Arbitrary colour gradient palette (continuous)

Description
Arbitrary colour gradient palette (continuous)

Usage
gradient_n_pal(colours, values = NULL, space = "Lab")

Arguments
colours vector of colours
values if colours should not be evenly positioned along the gradient this vector gives
the position (between 0 and 1) for each colour in the colours vector. See
rescale() for a convenience function to map an arbitrary range to between
0 and 1.
space colour space in which to calculate gradient. Must be "Lab" - other values are
deprecated.
grey_pal
Grey scale palette (discrete)

Description
Grey scale palette (discrete)

Usage
grey_pal(start = 0.2, end = 0.8)

Arguments
- start: grey value at low end of palette
- end: grey value at high end of palette

See Also
seq_gradient_pal() for continuous version

Examples
show_col(grey_pal)(25)
show_col(grey_pal(0, 1)(25))

hms_trans
Transformation for times (class hms)

Description
Transformation for times (class hms)

Usage
hms_trans()

Examples
if (require("hms")) {
  hms <- round(runif(10) * 86400)
  t <- hms_trans()
  t$transform(hms)
  t$inverse(t$transform(hms))
  t$breaks(hms)
}
**hue_pal**

**Hue palette (discrete)**

**Description**

Hue palette (discrete)

**Usage**

```r
hue_pal(h = c(0, 360) + 15, c = 100, l = 65, h.start = 0, direction = 1)
```

**Arguments**

- `h`: range of hues to use, in [0, 360]
- `c`: chroma (intensity of colour), maximum value varies depending on combination of hue and luminance.
- `l`: luminance (lightness), in [0, 100]
- `h.start`: hue to start at
direction: direction to travel around the colour wheel, 1 = clockwise, -1 = counter-clockwise

**Examples**

```r
show_col(hue_pal() (4))
show_col(hue_pal() (9))
show_col(hue_pal(l = 90) (9))
show_col(hue_pal(l = 30) (9))

show_col(hue_pal() (9))
show_col(hue_pal(direction = -1) (9))

show_col(hue_pal() (9))
show_col(hue_pal(h = c(0, 90)) (9))
show_col(hue_pal(h = c(90, 180)) (9))
show_col(hue_pal(h = c(180, 270)) (9))
show_col(hue_pal(h = c(270, 360)) (9))
```

**identity_pal**

**Identity palette**

**Description**

Leaves values unchanged - useful when the data is already scaled.

**Usage**

```r
identity_pal()
```
identity_trans  
*Identity transformation (do nothing)*

**Description**

Identity transformation (do nothing)

**Usage**

`identity_trans()`

**Examples**

```r
plot(identity_trans(), xlim = c(-1, 1))
```

---

**label_bytes**  
*Label bytes (1 kb, 2 MB, etc)*

**Description**

Scale bytes into human friendly units. Can use either SI units (e.g. kB = 1000 bytes) or binary units (e.g. kiB = 1024 bytes). See Units of Information on Wikipedia for more details.

**Usage**

`label_bytes(units = "auto_si", accuracy = 1, ...)`

**Arguments**

- **units**: Unit to use. Should either one of:
  - `auto_si` or `auto_binary` to automatically pick the most appropriate unit for each value.

- **accuracy**: A number to round to. Use (e.g.) 0.01 to show 2 decimal places of precision. If NULL, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values.

- **...**: Other arguments passed on to `number()`

**Value**

A labeller function that takes a numeric vector of breaks and returns a character vector of labels.
See Also

Other labels for continuous scales: `label_dollar()`, `label_number_auto()`, `label_number_si()`, `label_ordinal()`, `label_parse()`, `label_percent()`, `label_pvalue()`, `label_scientific()

Other labels for log scales: `label_number_si()`, `label_scientific()

Examples

demo_continuous(c(1, 1e6))

demo_continuous(c(1, 1e6), label = label_bytes())

# Force all to use same units

demo_continuous(c(1, 1e6), label = label_bytes("kB"))

# Auto units are particularly nice on log scales

demo_log10(c(1, 1e6))

demo_log10(c(1, 1e7), label = label_bytes())

# You can also use binary units where a megabyte is defined as

# (1024) ^ 2 bytes rather than (1000) ^ 2. You’ll need to override
# the default breaks to make this more informative.

demo_continuous(c(1, 1024^2), label = label_bytes("auto_binary"))

demo_continuous(c(1, 1024^2),

   breaks = breaks_width(250 * 1024),
   label = label_bytes("auto_binary")
)

label_date() and label_time() label date/times using date/time format strings. label_date_short() automatically constructs a short format string sufficient to uniquely identify labels. It’s inspired by matplotlib’s ConciseDateFormatter, but uses a slightly different approach: ConciseDateFormatter formats “firsts” (e.g. first day of month, first day of day) specially; date_short() formats changes (e.g. new month, new year) specially.

Usage

c

date_format("%Y-%m-%d", tz = "UTC")

date_time("%Y-%m-%d", tz = "UTC")

time_format("%Y-%m-%d", tz = "UTC")

Arguments

format  For date_format() and time_format() a date/time format string using standard POSIX specification. See strftime() for details.

For date_short() a character vector of length 4 giving the format components to use for year, month, day, and hour respectively.

tz  a time zone name, see timezones(). Defaults to UTC

sep  Separator to use when combining date formats into a single string.

Value

All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value.

Labelling functions are designed to be used with the labels argument of ggplot2 scales. The examples demonstrate their use with x scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface

date_format() and time_format() are retired; please use label_date() and label_time() instead.

Examples

date_range <- function(start, days) {
  start <- as.POSIXct(start)
  c(start, start + days * 24 * 60 * 60)
}
two_months <- date_range("2020-05-01", 60)
demo_datetime(two_months)
demo_datetime(two_months, labels = date_format("%m/%d"))
  # ggplot2 provides a short-hand:
demo_datetime(two_months, date_labels = "%m/%d")

  # An alternative labelling system is label_date_short()
demo_datetime(two_months, date_breaks = "7 days", labels = label_date_short())
  # This is particularly effective for dense labels
one_year <- date_range("2020-05-01", 365)
demo_datetime(one_year, date_breaks = "month")
demo_datetime(one_year, date_breaks = "month", labels = label_date_short())

label_dollar  Label currencies ($100, $2.50, etc)

Description

Format numbers as currency, rounding values to dollars or cents using a convenient heuristic.
Usage

```r
label_dollar(
  accuracy = NULL,
  scale = 1,
  prefix = "$",
  suffix = "",
  big.mark = ",",
  decimal.mark = ".",
  trim = TRUE,
  largest_with_cents = 1e+05,
  negative_parens = FALSE,
  ...
)
```

dollar_format(
  accuracy = NULL,
  scale = 1,
  prefix = "$",
  suffix = "",
  big.mark = ",",
  decimal.mark = ".",
  trim = TRUE,
  largest_with_cents = 1e+05,
  negative_parens = FALSE,
  ...
)

dollar(
  x,
  accuracy = NULL,
  scale = 1,
  prefix = "$",
  suffix = "",
  big.mark = ",",
  decimal.mark = ".",
  trim = TRUE,
  largest_with_cents = 1e+05,
  negative_parens = FALSE,
  ...
)
```

Arguments

- **accuracy, largest_with_cents**
  Number to round to. If NULL, the default, values will be rounded to the nearest integer, unless any of the values has non-zero fractional component (e.g. cents) and the largest value is less than largest_with_cents which by default is 100,000.

- **scale**
  A scaling factor: x will be multiplied by scale before formating. This is useful if the underlying data is very small or very large.

- **prefix, suffix**
  Symbols to display before and after value.

- **big.mark**
  Character used between every 3 digits to separate thousands.
The character to be used to indicate the numeric decimal point.
trim Logical, if FALSE, values are right-justified to a common width (see base::format()).
negative_parens Display negative using parentheses?
... Other arguments passed on to base::format().
x A numeric vector

Value
All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value.

Labelling functions are designed to be used with the labels argument of ggplot2 scales. The examples demonstrate their use with x scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface
dollar() and format_dollar() are retired; please use label_dollar() instead.

See Also
Other labels for continuous scales: label_bytes(), label_number_auto(), label_number_si(), label_ordinal(), label_parse(), label_percent(), label_pvalue(), label_scientific()

Examples
demo_continuous(c(0, 1), labels = label_dollar())
demo_continuous(c(100, 100), labels = label_dollar())

# Customise currency display with prefix and suffix
demo_continuous(c(1, 100), labels = label_dollar(prefix = "USD "))
euro <- dollar_format(
  prefix = ",",
  suffix = "\u20ac",
  big.mark = ",",
  decimal.mark = ","
)
demo_continuous(c(1000, 1100), labels = euro)

# Use negative_parens = TRUE for finance style display
demo_continuous(c(-100, 100), labels = label_dollar(negative_parens = TRUE))
Usage

```r
label_number(
  accuracy = NULL,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = "",
  decimal.mark = ".",
  trim = TRUE,
  ...)

label_comma(
  accuracy = NULL,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = ",",
  decimal.mark = ",",
  trim = TRUE,
  digits,
  ...)

comma(
  x,
  accuracy = NULL,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = ",",
  decimal.mark = ",",
  trim = TRUE,
  digits,
  ...)

number_format(
  accuracy = NULL,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = ",",
  decimal.mark = ".",
  trim = TRUE,
  ...)

comma_format(
  accuracy = NULL,
  scale = 1,
  prefix = "",
  ...)
```
suffix = ",",
big.mark = ",",
decimal.mark = ".",
trim = TRUE,
digits,
...)

Arguments

accuracy  A number to round to. Use (e.g.) 0.01 to show 2 decimal places of precision. If NULL, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
scale  A scaling factor: x will be multiplied by scale before formatting. This is useful if the underlying data is very small or very large.
prefix, suffix  Symbols to display before and after value.
big.mark  Character used between every 3 digits to separate thousands.
decimal.mark  The character to be used to indicate the numeric decimal point.
trim  Logical, if FALSE, values are right-justified to a common width (see base::format()).
digits  Deprecated, use accuracy instead.
x  A numeric vector to format.

Value

All label() functions return a “labelling” function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value. Labelling functions are designed to be used with the labels argument of ggplot2 scales. The examples demonstrate their use with x scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface

class_format(), comma_format(), and comma() are retired; please use label_number() and label_comma() instead.

Examples

demo_continuous(c(-1e6, 1e6))
demo_continuous(c(-1e6, 1e6), labels = label_number())
demo_continuous(c(-1e6, 1e6), labels = label_comma())

# Use scale to rescale very small or large numbers to generate more readable labels
demo_continuous(c(0, 1e6), labels = label_number())
demo_continuous(c(0, 1e6), labels = label_number(scale = 1 / 1e3))
demo_continuous(c(0, 1e-6), labels = label_number())
demo_continuous(c(0, 1e-6), labels = label_number(scale = 1e6))

# You can use prefix and suffix for other types of display
demo_continuous(c(32, 212), label = label_number(suffix = "\u00b0F"))
demo_continuous(c(0, 100), label = label_number(suffix = "\u00b0C"))
**label_number_auto**  
*Label numbers, avoiding scientific notation where possible*

**Description**

Switches between `number_format()` and `scientific_format()` based on a set of heuristics designed to automatically generate useful labels across a wide range of inputs.

**Usage**

```
label_number_auto()
```

**See Also**

Other labels for continuous scales: `label_bytes()`, `label_dollar()`, `label_number_si()`, `label_ordinal()`, `label_parse()`, `label_percent()`, `label_pvalue()`, `label_scientific()`

**Examples**

```
# Very small and very large numbers get scientific notation
demo_continuous(c(0, 1e-6), labels = label_number_auto())
demo_continuous(c(0, 1e9), labels = label_number_auto())

# Other ranges get the numbers printed in full
demo_continuous(c(0, 1e-3), labels = label_number_auto())
demo_continuous(c(0, 1), labels = label_number_auto())
demo_continuous(c(0, 1e3), labels = label_number_auto())
demo_continuous(c(0, 1e6), labels = label_number_auto())

# Transformation is applied individually so you get as little scientific notation as possible
demo_log10(c(1, 1e7), labels = label_number_auto())
```

**label_number_si**  
*Label numbers with SI prefixes (2k, 1M, 5T etc)*

**Description**

`number_si()` automatically scales and labels with the best SI prefix, "K" for values $\geq 10^3$, "M" for $\geq 10^6$, "B" for $\geq 10^9$, and "T" for $\geq 10^{12}$.

**Usage**

```
label_number_si(accuracy = 1, unit = NULL, sep = NULL, ...)
```
Arguments

accuracy  A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.

unit  Optional units specifier.

sep  Separator between number and SI unit. Defaults to `" "` if `units` is supplied, and `""` if not.

...  Other arguments passed on to `base::format()`. 

Value

All `label_()` functions return a "labelling" function, i.e. a function that takes a vector `x` and returns a character vector of length (`length(x)`) giving a label for each input value.

Labelling functions are designed to be used with the `labels` argument of `ggplot2` scales. The examples demonstrate their use with `x` scales, but they work similarly for all scales, including those that generate legends rather than axes.

See Also

Other labels for continuous scales: `label_bytes()`, `label_dollar()`, `label_number_auto()`, `label_ordinal()`, `label_parse()`, `label_percent()`, `label_pvalue()`, `label_scientific()`

Other labels for log scales: `label_bytes()`, `label_scientific()`

Examples

demo_continuous(c(1, 1e9), label = label_number_si())
demo_continuous(c(1, 5000), label = label_number_si(unit = "g"))
demo_continuous(c(1, 1000), label = label_number_si(unit = "m"))

demo_log10(c(1, 1e9), breaks = log_breaks(10), labels = label_number_si())

---

**label_ordinal**  
Label ordinal numbers (1st, 2nd, 3rd, etc)

**Description**

Round values to integers and then display as ordinal values (e.g. 1st, 2nd, 3rd). Built-in rules are provided for English, French, and Spanish.

**Usage**

```
label_ordinal(
  prefix = "",
  suffix = "",
  big.mark = " ",
  rules = ordinal_english(),
  ...
)
```
label_ordinal

    ordinal_english()
    ordinal_french(gender = c("masculin", "feminin"), plural = FALSE)
    ordinal_spanish()

    ordinal_format(
        prefix = "",
        suffix = "",
        big.mark = " ",
        rules = ordinal_english(),
        ...
    )

    ordinal(
        x,
        prefix = "",
        suffix = "",
        big.mark = " ",
        rules = ordinal_english(),
        ...
    )

Arguments

    prefix, suffix  Symbols to display before and after value.
    big.mark        Character used between every 3 digits to separate thousands.
    rules           Named list of regular expressions, matched in order. Name gives suffix, and
                    value specifies which numbers to match.
    ...             Other arguments passed on to base::format().
    gender          Masculin or feminin gender for French ordinal.
    plural          Plural or singular for French ordinal.
    x               A numeric vector to format.

Value

    All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns
    a character vector of length(x) giving a label for each input value.

    Labelling functions are designed to be used with the labels argument of ggplot2 scales. The
    examples demonstrate their use with x scales, but they work similarly for all scales, including those
    that generate legends rather than axes.

Old interface

    ordinal() and format_ordinal() are retired; please use label_ordinal() instead.

See Also

    Other labels for continuous scales: label_bytes(), label_dollar(), label_number_auto(),
    label_number_si(), label_parse(), label_percent(), label_pvalue(), label_scientific()
Examples

demo_continuous(c(1, 5))
demo_continuous(c(1, 5), labels = label_ordinal())
demo_continuous(c(1, 5), labels = label_ordinal(rules = ordinal_french()))

# The rules are just a set of regular expressions that are applied in turn
ordinal_french()
ordinal_english()

# Note that ordinal rounds values, so you may need to adjust the breaks too
demo_continuous(c(1, 10))
demo_continuous(c(1, 10), labels = label_ordinal())
demo_continuous(c(1, 10),
  labels = label_ordinal(),
  breaks = breaks_width(2))
)

label_parse                Label with mathematical annotations

Description

label_parse() produces expression from strings by parsing them; label_math() constructs expressions by replacing the pronoun .x with each string.

Usage

label_parse()

label_math(expr = 10^.x, format = force)

parse_format()

math_format(expr = 10^.x, format = force)

Arguments

expr                expression to use
format               another format function to apply prior to mathematical transformation - this makes it easier to use floating point numbers in mathematical expressions.

Value

All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value.

Labelling functions are designed to be used with the labels argument of ggplot2 scales. The examples demonstrate their use with x scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface

parse_format() and math_format() was retired; please use label_parse() and label_math() instead.
See Also

plotmath for the details of mathematical formatting in R.

Other labels for continuous scales: label_bytes(), label_dollar(), label_number_auto(), label_number_si(), label_ordinal(), label_percent(), label_pvalue(), label_scientific()

Other labels for discrete scales: label_wrap()

Examples

# Use label_parse() with discrete scales
greek <- c("alpha", "beta", "gamma")
demo_discrete(greek)
demo_discrete(greek, labels = label_parse())

# Use label_math() with continuous scales
demo_continuous(c(1, 5))
demo_continuous(c(1, 5), labels = label_math(alpha[x]))

label_percent  Label percentages (2.5%, 50%, etc)

Description

Label percentages (2.5%, 50%, etc)

Usage

label_percent(
  accuracy = NULL,
  scale = 100,
  prefix = "",
  suffix = "%",
  big.mark = " ",
  decimal.mark = ".",
  trim = TRUE,
...
)

percent_format(
  accuracy = NULL,
  scale = 100,
  prefix = "",
  suffix = "%",
  big.mark = " ",
  decimal.mark = ".",
  trim = TRUE,
...
)

percent(
  x,
  accuracy = NULL,
Arguments

accuracy  A number to round to. Use (e.g.) 0.01 to show 2 decimal places of precision. If
NULL, the default, uses a heuristic that should ensure breaks have the minimum
number of digits needed to show the difference between adjacent values.
Applied to rescaled data.
scale     A scaling factor: x will be multiplied by scale before formatting. This is useful
if the underlying data is very small or very large.
prefix    Symbols to display before and after value.
suffix    Symbols to display before and after value.
big.mark  Character used between every 3 digits to separate thousands.
decimal.mark  The character to be used to indicate the numeric decimal point.
trim      Logical, if FALSE, values are right-justified to a common width (see base::format()).
...       Other arguments passed on to base::format().
x         A numeric vector to format.

Value

All label_( ) functions return a “labelling” function, i.e. a function that takes a vector x and returns
a character vector of length(x) giving a label for each input value.
Labelling functions are designed to be used with the labels argument of ggplot2 scales. The
examples demonstrate their use with x scales, but they work similarly for all scales, including those
that generate legends rather than axes.

Old interface

percent() and percent_format() are retired; please use label_percent() instead.

See Also

Other labels for continuous scales: label_bytes(), label_dollar(), label_number_auto(),
label_number_si(), label_ordinal(), label_parse(), label_pvalue(), label_scientific()

Examples

demo_continuous(c(0, 1))
demo_continuous(c(0, 1), labels = label_percent())

# Use prefix and suffix to create your own variants
french_percent <- label_percent(
  decimal.mark = "",
  suffix = "\%")
demo_continuous(c(0, .01), labels = french_percent)
**label_pvalue**

*Label p-values (e.g. <0.001, 0.25, p >= 0.99)*

**Description**

Formatter for p-values, using "<" and ">" for p-values close to 0 and 1.

**Usage**

```r
label_pvalue(
  accuracy = 0.001,
  decimal.mark = ".",
  prefix = NULL,
  add_p = FALSE
)
```

```r
pvalue_format(
  accuracy = 0.001,
  decimal.mark = ".",
  prefix = NULL,
  add_p = FALSE
)
```

```r
pvalue(x, accuracy = 0.001, decimal.mark = ".", prefix = NULL, add_p = FALSE)
```

**Arguments**

- **accuracy**: A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.

- **decimal.mark**: The character to be used to indicate the numeric decimal point.

- **prefix**: A character vector of length 3 giving the prefixes to put in front of numbers. The default values are `c("H", "B", "p")` if `add_p` is `TRUE` and `c("p<", "p="", "p>")` if `FALSE`.

- **add_p**: Add "p=" before the value?

- **x**: A numeric vector to format.

**Value**

All `label_()` functions return a "labelling" function, i.e. a function that takes a vector `x` and returns a character vector of length(`x`) giving a label for each input value.

Labelling functions are designed to be used with the `labels` argument of `ggplot2` scales. The examples demonstrate their use with `x` scales, but they work similarly for all scales, including those that generate legends rather than axes.

**Old interface**

`pvalue()` and `pvalue_dollar()` are retired; please use `label_pvalue()` instead.
See Also

Other labels for continuous scales: `label_bytes()`, `label_dollar()`, `label_number_auto()`, `label_number_si()`, `label_percent()`, `label_parse()`, `label_scientific()`,

Examples

demo_continuous(c(0, 1))
demo_continuous(c(0, 1), labels = label_pvalue())
demo_continuous(c(0, 1), labels = label_pvalue(accuracy = 0.1))
demo_continuous(c(0, 1), labels = label_pvalue(add_p = TRUE))

# Or provide your own prefixes
prefix <- c("p < ", "p = ", "p > ")
demo_continuous(c(0, 1), labels = label_pvalue(prefix = prefix))

label_scientific  
Label numbers with scientific notation (e.g. 1e05, 1.5e-02)

Description

Label numbers with scientific notation (e.g. 1e05, 1.5e-02)

Usage

label_scientific(
  digits = 3,
  scale = 1,
  prefix = "",
  suffix = "",
  decimal.mark = ".",
  trim = TRUE,
  ...
)

scientific_format(
  digits = 3,
  scale = 1,
  prefix = "",
  suffix = "",
  decimal.mark = ".",
  trim = TRUE,
  ...
)

scientific(
  x,
  digits = 3,
  scale = 1,
  prefix = "",
  suffix = "",
  decimal.mark = ".",
  ...)
Arguments

digits Number of digits to show before exponent.
scale A scaling factor: x will be multiplied by scale before formatting. This is useful if the underlying data is very small or very large.

prefix, suffix Symbols to display before and after value.
decimal.mark The character to be used to indicate the numeric decimal point.
trim Logical, if FALSE, values are right-justified to a common width (see base::format()).

... Other arguments passed on to base::format().
x A numeric vector to format.

Value

All label_() functions return a "labelling" function, i.e. a function that takes a vector x and returns a character vector of length(x) giving a label for each input value.

Labelling functions are designed to be used with the labels argument of ggplot2 scales. The examples demonstrate their use with x scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface

scientific_format() and scientific() are retired; please use label_scientific().

See Also

Other labels for continuous scales: label_bytes(), label_dollar(), label_number_auto(), label_number_si(), label_ordinal(), label_parse(), label_percent(), label_pvalue()

Other labels for log scales: label_bytes(), label_number_si()

Examples

demo_continuous(c(1, 10))
demo_continuous(c(1, 10), labels = label_scientific())
demo_continuous(c(1, 10), labels = label_scientific(digits = 3))
demo_log10(c(1, 1e9))
label_wrap

Label strings by wrapping across multiple lines

Description

Uses `strwrap()` to split long labels across multiple lines.

Usage

```r
label_wrap(width)
wrap_format(width)
```

Arguments

- `width` Number of characters per line.

Value

All `label_()` functions return a "labelling" function, i.e. a function that takes a vector `x` and returns a character vector of `length(x)` giving a label for each input value.

Labelling functions are designed to be used with the `labels` argument of ggplot2 scales. The examples demonstrate their use with `x` scales, but they work similarly for all scales, including those that generate legends rather than axes.

Old interface

`wrap_format()` is retired; please use `label_format()` instead.

See Also

Other labels for discrete scales: `label_parse()`

Examples

```r
x <- c("this is a long label", "this is another long label", "this a label this is even longer")
demo_discrete(x)
demo_discrete(x, labels = label_wrap(10))
demo_discrete(x, labels = label_wrap(20))
```
linetype_pal  Line type palette (discrete)

Description

Based on a set supplied by Richard Pearson, University of Manchester

Usage

linetype_pal()

log_trans  Log transformations

Description

• log_trans(): log(x)
• log1p(): log(x + 1)
• pseudo_log_trans(): smoothly transition to linear scale around 0.

Usage

log_trans(base = exp(1))

log10_trans()

log2_trans()

log1p_trans()

pseudo_log_trans(sigma = 1, base = exp(1))

Arguments

base       base of logarithm
sigma     Scaling factor for the linear part of pseudo-log transformation.

Examples

plot(log2_trans(), xlim = c(0, 5))
plot(log_trans(), xlim = c(0, 5))
plot(log10_trans(), xlim = c(0, 5))

plot(log_trans(), xlim = c(0, 2))
plot(log1p_trans(), xlim = c(-1, 1))

# The pseudo-log is defined for all real numbers
plot(pseudo_log_trans(), xlim = c(-5, 5))
lines(log_trans(), xlim = c(0, 5), col = "red")
# For large positive numbers it's very close to log
plot(pseudo_log_trans(), xlim = c(1, 20))
lines(log_trans(), xlim = c(1, 20), col = "red")

---

### **manual_pal**

#### Manual palette (discrete)

**Description**

Manual palette (discrete)

**Usage**

`manual_pal(values)`

**Arguments**

- `values` vector of values to be used as a palette.

### **minor_breaks_width**

#### Minor breaks

**Description**

Generate minor breaks between major breaks either spaced with a fixed width, or having a fixed number.

**Usage**

`minor_breaks_width(width, offset)`

- `minor_breaks_n(n)`

**Arguments**

- `width` Distance between each break. Either a number, or for date/times, a single string of the form "n unit", e.g. "1 month", "5 days". Unit can be of one "sec", "min", "hour", "day", "week", "month", "year".
- `offset` Use if you don't want breaks to start at zero
- `n` number of breaks

**Examples**

```r
demo_log10(c(1, 1e6))
if (FALSE) {
  # Requires https://github.com/tidyverse/ggplot2/pull/3591
  demo_log10(c(1, 1e6), minor_breaks = minor_breaks_n(10))
}
```
muted

Mute standard colour

Description
Mute standard colour

Usage
muted(colour, l = 30, c = 70)

Arguments
- colour: character vector of colours to modify
- l: new luminance
- c: new chroma

Examples
muted("red")
muted("blue")
show_col(c("red", "blue", muted("red"), muted("blue")))

oob
Out of bounds handling

Description
This set of functions modify data values outside a given range. The oob_*() functions are designed to be passed as the oob argument of ggplot2 continuous and binned scales, with oob_discard being an exception.

These functions affect out of bounds values in the following ways:

- oob_censor() replaces out of bounds values with NAs. This is the default oob argument for continuous scales.
- oob_censor_any() acts like oob_censor(), but also replaces infinite values with NAs.
- oob_squish() replaces out of bounds values with the nearest limit. This is the default oob argument for binned scales.
- oob_squish_any() acts like oob_squish(), but also replaces infinite values with the nearest limit.
- oob_squish_infinite() only replaces infinite values by the nearest limit.
- oob_keep() does not adjust out of bounds values. In position scales, behaves as zooming limits without data removal.
- oob_discard() removes out of bounds values from the input. Not suitable for ggplot2 scales.
Usage

```r
oob_censor(x, range = c(0, 1), only.finite = TRUE)
oob_censor_any(x, range = c(0, 1))
oob_discard(x, range = c(0, 1))
oob_squish(x, range = c(0, 1), only.finite = TRUE)
oob_squish_any(x, range = c(0, 1))
oob_squish_infinite(x, range = c(0, 1))
oob_keep(x, range = c(0, 1))
censor(x, range = c(0, 1), only.finite = TRUE)
discard(x, range = c(0, 1))
squish(x, range = c(0, 1), only.finite = TRUE)
squish_infinite(x, range = c(0, 1))
```

Arguments

- **x**: A numeric vector of values to modify.
- **range**: A numeric vector of length two giving the minimum and maximum limit of the desired output range respectively.
- **only.finite**: A logical of length one. When TRUE, only finite values are altered. When FALSE, also infinite values are altered.

Details

The `oob_censor_any()` and `oob_squish_any()` functions are the same as `oob_censor()` and `oob_squish()` with the `only.finite` argument set to FALSE.

Replacing position values with NAs, as `oob_censor()` does, will typically lead to removal of those datapoints in ggplot.

Setting ggplot coordinate limits is equivalent to using `oob_keep()` in position scales.

Value

Most `oob_()` functions return a vector of numerical values of the same length as the `x` argument, wherein out of bounds values have been modified. Only `oob_discard()` returns a vector of less than or of equal length to the `x` argument.

Old interface

`censor()`, `squish()`, `squish_infinite()` and `discard()` are no longer recommended; please use `oob_censor()`, `oob_squish()`, `oob_squish_infinite()` and `oob_discard()` instead.
probability_trans

Description

Probability transformation

Usage

probability_trans(distribution, ...)

logit_trans()

probit_trans()

Arguments

distribution probability distribution. Should be standard R abbreviation so that "p" + distribution is a valid probability density function, and "q" + distribution is a valid quantile function.

... other arguments passed on to distribution and quantile functions

Examples

plot(logit_trans(), xlim = c(0, 1))
plot(probit_trans(), xlim = c(0, 1))
**Mutable ranges**

**Description**

Mutable ranges have a two methods (`train` and `reset`), and make it possible to build up complete ranges with multiple passes.

**reciprocal_trans**

**Reciprocal transformation**

**Description**

Reciprocal transformation

**Usage**

```r
reciprocal_trans()
```

**Examples**

```r
plot(reciprocal_trans(), xlim = c(0, 1))
```

**rescale**

**Rescale continuous vector to have specified minimum and maximum**

**Description**

Rescale continuous vector to have specified minimum and maximum

**Usage**

```r
rescale(x, to, from, ...)
```

```r
## S3 method for class 'numeric'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
```

```r
## S3 method for class 'dist'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
```

```r
## S3 method for class 'logical'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
```

```r
## S3 method for class 'POSIXt'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
```

```r
## S3 method for class 'Date'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE, finite = TRUE), ...)
```

```r
## S3 method for class 'integer64'
rescale(x, to = c(0, 1), from = range(x, na.rm = TRUE), ...)
```
`rescale_max`  

**Arguments**

- `x` : continuous vector of values to manipulate.  
- `to` : output range (numeric vector of length two)  
- `from` : input range (vector of length two). If not given, is calculated from the range of `x`  
- `...` : other arguments passed on to methods

**Examples**

```r
rescale(1:100)
rescale(runif(50))
rescale(1)
```

---

**Description**

Rescale numeric vector to have specified maximum

**Usage**

```r
rescale_max(x, to = c(0, 1), from = range(x, na.rm = TRUE))
```

**Arguments**

- `x` : numeric vector of values to manipulate.  
- `to` : output range (numeric vector of length two)  
- `from` : input range (numeric vector of length two). If not given, is calculated from the range of `x`

**Examples**

```r
rescale_max(1:100)
rescale_max(runif(50))
rescale_max(1)
```
Rescale vector to have specified minimum, midpoint, and maximum

Arguments

- `x` vector of values to manipulate.
- `to` output range (numeric vector of length two)
- `from` input range (vector of length two). If not given, is calculated from the range of `x`
- `mid` mid-point of input range
- `...` other arguments passed on to methods

Examples

```
rescale_mid(1:100, mid = 50.5)
rescale_mid(runif(50), mid = 0.5)
rescale_mid(1)
```
rescale_none

Don’t perform rescaling

Description
Don’t perform rescaling

Usage
rescale_none(x, ...)

Arguments
x numeric vector of values to manipulate.
... all other arguments ignored

Examples
rescale_none(1:100)

rescale_pal
Rescale palette (continuous)

Description
Just rescales the input to the specific output range. Useful for alpha, size, and continuous position.

Usage
rescale_pal(range = c(0.1, 1))

Arguments
range Numeric vector of length two, giving range of possible values. Should be between 0 and 1.

reverse_trans
Reverse transformation

Description
Reverse transformation

Usage
reverse_trans()

Examples
plot(reverse_trans(), xlim = c(-1, 1))
seq_gradient_pal  Sequential colour gradient palette (continuous)

Description
Sequential colour gradient palette (continuous)

Usage
seq_gradient_pal(low = mns1("10B 4/6"), high = mns1("10R 4/6"), space = "Lab")

Arguments
low  colour for low end of gradient.
high  colour for high end of gradient.
space  colour space in which to calculate gradient. Must be "Lab" - other values are deprecated.

Examples
x <- seq(0, 1, length.out = 25)
show_col(seq_gradient_pal(x))
show_col(seq_gradient_pal("white", "black"))(x)

library(munsell)
show_col(seq_gradient_pal("white", mns1("10R 4/6"))(x))

shape_pal  Shape palette (discrete)

Description
Shape palette (discrete)

Usage
shape_pal(solid = TRUE)

Arguments
solid  should shapes be solid or not?
**sqrt_trans**

**Square-root transformation**

**Description**

This is the variance stabilising transformation for the Poisson distribution.

**Usage**

\[
\text{sqrt\_trans()}
\]

**Examples**

\[
\text{plot(sqrt\_trans(), xlim = c(0, 5))}
\]

**time_trans**

**Transformation for date-times (class POSIXt)**

**Description**

Transformation for date-times (class POSIXt)

**Usage**

\[
\text{time\_trans(tz = NULL)}
\]

**Arguments**

- **tz**: Optionally supply the time zone. If NULL, the default, the time zone will be extracted from first input with a non-null tz.

**Examples**

\[
\begin{align*}
\text{hours } & \leftarrow \text{seq(ISOdate(2000,3,20, tz = ""), by = "hour", length.out = 10)} \\
\text{t } & \leftarrow \text{time\_trans()} \\
\text{t}\$\text{transform(hours)} \\
\text{t}\$\text{inverse(t}\$\text{transform(hours))} \\
\text{t}\$\text{format(t}\$\text{breaks(range(hours)))}
\end{align*}
\]
train_continuous

Train (update) a continuous scale

Description

Strips attributes and always returns a numeric vector

Usage

train_continuous(new, existing = NULL)

Arguments

- new: New data to add to scale
- existing: Optional existing scale to update

train_discrete

Train (update) a discrete scale

Description

Train (update) a discrete scale

Usage

train_discrete(new, existing = NULL, drop = FALSE, na.rm = FALSE)

Arguments

- new: New data to add to scale
- existing: Optional existing scale to update
- drop: TRUE, will drop factor levels not associated with data
- na.rm: If TRUE, will remove missing values
**viridis_pal**  

**Viridis palette**  

**Description**  

Viridis palette  

**Usage**  

`viridis_pal(alpha = 1, begin = 0, end = 1, direction = 1, option = "D")`  

**Arguments**  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| alpha    | The alpha transparency, a number in [0,1], see argument alpha in `hsv`.
| begin    | The (corrected) hue in [0,1] at which the viridis colormap begins. |
| end      | The (corrected) hue in [0,1] at which the viridis colormap ends. |
| direction| Sets the order of colors in the scale. If 1, the default, colors are ordered from darkest to lightest. If -1, the order of colors is reversed. |
| option   | A character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E"). |

**References**  

[https://bids.github.io/colormap/](https://bids.github.io/colormap/)

**Examples**  

```r  
show_col(viridis_pal())
show_col(viridis_pal(direction = -1))
show_col(viridis_pal(begin = 0.2, end = 0.8))
show_col(viridis_pal(option = "plasma"))
```

---

**yj_trans**  

**Yeo-Johnson transformation**  

**Description**  

The Yeo-Johnson transformation is a flexible transformation that is similar to Box-Cox, `boxcox_trans()`, but does not require input values to be greater than zero.  

**Usage**  

`yj_trans(p)`  

**Arguments**  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Transformation exponent, λ.</td>
</tr>
</tbody>
</table>
Zero Range

Details

The transformation takes one of four forms depending on the values of \( y \) and \( \lambda \).

- \( y \geq 0 \) and \( \lambda \neq 0 \): \( y(\lambda) = \frac{(y+1)^{\lambda}-1}{\lambda} \)
- \( y \geq 0 \) and \( \lambda = 0 \): \( y(\lambda) = \ln(y + 1) \)
- \( y < 0 \) and \( \lambda \neq 2 \): \( y(\lambda) = -\frac{(\bar{y}+1)(2-\lambda)-1}{2-\lambda} \)
- \( y < 0 \) and \( \lambda = 2 \): \( y(\lambda) = -\ln(-y + 1) \)

References


Examples

```
pplot(yj_trans(-1), xlim = c(-10, 10))
pplot(yj_trans(0), xlim = c(-10, 10))
pplot(yj_trans(1), xlim = c(-10, 10))
pplot(yj_trans(2), xlim = c(-10, 10))
```

zero_range

Determine if range of vector is close to zero, with a specified tolerance

Description

The machine epsilon is the difference between 1.0 and the next number that can be represented by the machine. By default, this function uses epsilon * 1000 as the tolerance. First it scales the values so that they have a mean of 1, and then it checks if the difference between them is larger than the tolerance.

Usage

```
zero_range(x, tol = 1000 * .Machine$double.eps)
```

Arguments

- \( x \) numeric range: vector of length 2
- \( tol \) A value specifying the tolerance.

Value

logical TRUE if the relative difference of the endpoints of the range are not distinguishable from 0.
Examples

eps <- .Machine$double.eps
zero_range(c(1, 1 + eps))  # TRUE
zero_range(c(1, 1 + 99 * eps))  # TRUE
zero_range(c(1, 1 + 1001 * eps))  # FALSE - Crossed the tol threshold
zero_range(c(1, 1 + 2 * eps), tol = eps)  # FALSE - Changed tol

# Scaling up or down all the values has no effect since the values
# are rescaled to 1 before checking against tol
zero_range(100000 * c(1, 1 + eps))  # TRUE
zero_range(100000 * c(1, 1 + 1001 * eps))  # FALSE
zero_range(0.00001 * c(1, 1 + eps))  # TRUE
zero_range(0.00001 * c(1, 1 + 1001 * eps))  # FALSE

# NA values
zero_range(c(1, NA))  # NA
zero_range(c(1, NaN))  # NA

# Infinite values
zero_range(c(1, Inf))  # FALSE
zero_range(c(-Inf, Inf))  # FALSE
zero_range(c(Inf, Inf))  # TRUE
Index

*Topic manip
  rescale, 42

abs_area(area_pal), 3
alpha, 3
area_pal, 3, 13
asn_trans, 4, 13
atanh_trans, 4, 13
base::cutHI, 12
base::formatHI, 24, 26, 28, 29, 32, 35
boxcox_trans, 4, 13
boxcox_transHI, 49
breaks_extended, 5
breaks_log, 6
breaks_pretty, 7
breaks_width, 8
brewer_pal, 8, 13
censor (oob), 39
col2hcl, 9
col_bin(col_numeric), 11
col_factor (col_numeric), 11
col_numeric, 11
col_quantile (col_numeric), 11
colorRamp, 10
colour_ramp, 10
comma (label_number), 24
coma_format (label_number), 24
ContinuousRange (Range), 42
cscale, 13
date_format (label_date), 21
date_trans, 13, 14
dichromat_pal, 13, 15
discard (oob), 39
DiscreteRange (Range), 42
div_gradient_pal, 13, 15
dollar (label_dollar), 22
dollar_format (label_dollar), 22
dsacle, 16

exp_trans, 13, 17
expand_range, 16
extended_breaks (breaks_extended), 5

extended_breaks(), 6, 7
gradient_n_pal, 13, 17
grDevices::col2rgb(), 10
grDevices::colors(), 10
grDevices::palette(), 10
grey_pal, 13, 18

hms_trans, 13, 18
hsb, 49
hue_pal, 13, 19

identity_pal, 13, 19
identity_trans, 13, 20

label_bytes, 20, 24, 27–29, 31, 32, 34, 35
label_comma (label_number), 24
label_date, 21
label_date_short (label_date), 21
label_dollar, 21, 22, 27–29, 31, 32, 34, 35
label_math (label_parse), 30
label_number, 24
label_number_auto, 21, 24, 27, 28, 29, 31, 32, 34, 35
label_number_si, 21, 24, 27, 29, 31, 32, 34, 35
label_ordinal, 21, 24, 27, 29, 31, 32, 34, 35
label_parse, 21, 24, 27–29, 30, 32, 34–36
label_percent, 21, 24, 27–29, 31, 33, 34, 35
label_pvalue, 21, 24, 27–29, 31, 32, 33, 35
label_scientific, 21, 24, 27–29, 31, 32, 34, 35
label_time (label_date), 21
label_wrap, 31, 36
labeling::extended(), 6
linetype_pal, 13, 37
log10_trans, 13
log10_trans (log_trans), 37
log1p_trans, 13
log1p_trans (log_trans), 37
log2_trans, 13
log2_trans (log_trans), 37
log_breaks (breaks_log), 6

52
INDEX

log_trans, 13, 37
logit_trans, 13
logit_trans (probability_trans), 41

manual_pal, 13, 38
math_format (label_parse), 30
minor_breaks_n (minor_breaks_width), 38
minor_breaks_width, 38
modulus_trans, 13
modulus_trans (boxcox_trans), 4
muted, 39

number(), 20
number_format (label_number), 24

oob, 39
oob_censor (oob), 39
oob_censor_any (oob), 39
oob_discard (oob), 39
oob_keep (oob), 39
oob_squish (oob), 39
oob_squish_any (oob), 39
oob_squish_infinite (oob), 39
ordinal (label_ordinal), 28
ordinal_english (label_ordinal), 28
ordinal_format (label_ordinal), 28
ordinal_french (label_ordinal), 28
ordinal_spanish (label_ordinal), 28

parse_format (label_parse), 30
percent (label_percent), 31
percent_format (label_percent), 31
plotmath, 31
pretty(), 7, 12
pretty_breaks (breaks_pretty), 7
probability_trans, 13, 41
probit_trans, 13
probit_trans (probability_trans), 41
pseudo_log_trans, 13
pseudo_log_trans (log_trans), 37
pvalue (label_pvalue), 33
pvalue_format (label_pvalue), 33

Range, 42
RCColorBrewer::brewer_pal(), 9
reciprcal_trans, 13, 42
rescale, 42
rescale(), 17
rescale_max, 43
rescale_mid, 44
rescale_none, 45
rescale_pal, 13, 45

reverse_trans, 13, 45
scientific, 24
scientific (label_scientific), 34
scientific_format (label_scientific), 34
scientific_format(), 27
seq(), 12
seq_gradient_pal, 13, 46
seq_gradient_pal(), 18
shape_pal, 13, 46
sqrt_trans, 13, 47
squish (oob), 39
squish_infinite (oob), 39
stats::quantile(), 12
strptime(), 22
strwrap(), 36
time_format (label_date), 21
time_trans, 13, 47
timezones(), 22
train_continuous, 48
train_discrete, 48

viridis_pal, 13, 49
wrap_format (label_wrap), 36
yj_trans, 13, 49
yj_trans(), 5
zero_range, 50