# Package 'ggsurveillance'

February 11, 2025

**Title** Tools for Outbreak Investigation/Infectious Disease Surveillance

**Version** 0.1.2

**Description** Create epicurves or epigantt charts in 'ggplot2'. Prepare data for visualisation or other reporting for infectious disease surveillance and outbreak investigation. Includes tidy functions to solve date based transformations for common reporting tasks, like (A) seasonal date alignment for respiratory disease surveillance, (B) date-based case binning based on specified time intervals like isoweek, epiweek, month and more, (C) automated detection and marking of the new year based on the date/datetime axis of the 'ggplot2'. An introduction on how to use epicurves can be found on the US CDC website (2012, <https://www.cdc.gov/training/quicklearns/epimode/index.html>).

**License** GPL (>= 3)

**URL** https://ggsurveillance.biostats.dev, https://github.com/biostats-dev/ggsurveillance

**BugReports** https://github.com/biostats-dev/ggsurveillance/issues

**Depends** R (>= 4.1.0)

**Imports** cli, dplyr, forcats, ggplot2, glue, ISOweek, lubridate, rlang, scales, stringr, tidyr, tidyselect, tsibble

**Suggests** Hmisc, knitr, outbreaks, rmarkdown, spelling, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-GB

**LazyData** true

**NeedsCompilation** no

**RoxygenNote** 7.3.2

**Author** Alexander Bartel [aut, cre] (<https://orcid.org/0000-0002-1280-6138>)

**Maintainer** Alexander Bartel <alexander.bartel@fu-berlin.de>

**Repository** CRAN
**Date/Publication** 2025-02-11 17:00:05 UTC

# Contents



---

`align_dates_seasonal` *Align dates for seasonal comparison*

---

## Description

Standardizes dates from multiple years to enable comparison of epidemic curves and visualization of seasonal patterns in infectious disease surveillance data. Commonly used for creating periodicity plots of respiratory diseases like influenza, RSV, or COVID-19.

## Usage

```
align_dates_seasonal(
  x,
  dates_from = NULL,
  date_resolution = c("week", "isoweek", "epiweek", "day", "month"),
  start = NULL,
  target_year = NULL,
  drop_leap_week = TRUE
)

align_and_bin_dates_seasonal(
  x,
  n = 1,
  dates_from,
  population = 1,
  fill_gaps = FALSE,
  date_resolution = c("week", "isoweek", "epiweek", "day", "month"),
  start = NULL,
  target_year = NULL,
  drop_leap_week = TRUE
)
```

## Arguments

| | |
|---|---|
| x | Either a data frame with a date column, or a date vector.<br>Supported date formats are date and datetime and also commonly used character strings:<br>• ISO dates "2024-03-09"<br>• Month "2024-03"<br>• Week "2024-W09" or "2024-W09-1" |
| dates_from | Column name containing the dates to align. Used when x is a data.frame. |
| date_resolution | Character string specifying the temporal resolution. One of:<br>• "week" or "isoweek" - Calendar weeks (ISO 8601), reporting weeks as used by the ECDC.<br>• "epiweek" - Epidemiological weeks (US CDC), i.e. ISO weeks with Sunday as week start.<br>• "month" - Calendar months<br>• "day" - Daily resolution |
| start | Numeric value indicating epidemic season start:<br>• For week/epiweek: week number (default: 28, approximately July)<br>• For month: month number (default: 7 for July)<br>• For day: day of year (default: 150, approximately June) |
| target_year | Numeric value for the reference year to align dates to. The default target year is the start of the most recent season in the data. This way the most recent dates stay unchanged. |
| drop_leap_week | If TRUE and date_resolution is week, isoweek or epiweek, leap weeks (week 53) are dropped if they are not in the most recent season. Disable if data should be returned. Dropping week 53 from historical data is the most common approach. Otherwise historical data for week 53 would map to week 52 if the target season has no leap week, resulting in a doubling of the case counts. |
| n | Numeric column with case counts. Supports quoted and unquoted column names. |
| population | A number or a numeric column with the population size. Used to calculate the incidence. |
| fill_gaps | Logical; If TRUE, gaps in the time series will be filled with 0 cases. |

## Details

This function helps create standardized epidemic curves by aligning surveillance data from different years. This enables:

- Comparison of disease patterns across multiple seasons
- Identification of typical seasonal trends
- Detection of unusual disease activity
- Assessment of current season against historical patterns

The alignment can be done at different temporal resolutions (daily, weekly, monthly) with customizable season start points to match different disease patterns or surveillance protocols.

## Value

A data frame with standardized date columns:

- `year`: Calendar year from original date
- `week/month/day`: Time unit based on chosen resolution
- `date_aligned`: Date standardized to target year
- `season`: Epidemic season identifier (e.g., "2023/24")
- `current_season`: Logical flag for most recent season

Binning also creates the columns:

- `n`: Sum of cases in bin
- `incidence`: Incidence calculated using n/population

## Examples

```
# Sesonal Visualization of Germany Influenza Surveillance Data
library(ggplot2)

influenza_germany |>
  align_dates_seasonal(
    dates_from = ReportingWeek, date_resolution = "epiweek", start = 28
  ) -> df_flu_aligned

ggplot(df_flu_aligned, aes(x = date_aligned, y = Incidence, color = season)) +
  geom_line() +
  facet_wrap(~AgeGroup) +
  theme_bw()
```

---

# create_agegroups *Create Age Groups from Numeric Values*

---

## Description

Creates age groups from numeric values using customizable break points and formatting options. The function allows for flexible formatting and customization of age group labels.

If a factor is returned, this factor includes factor levels of unobserved age groups. This allows for reproducible age groups, which can be used for joining data (e.g. adding age grouped population numbers for incidence calculation).

## Usage

```
create_agegroups(
  values,
  age_breaks = c(5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90),
  breaks_as_lower_bound = TRUE,
  first_group_format = "0-{x}",
  interval_format = "{x}-{y}",
  last_group_format = "{x}+",
  pad_numbers = FALSE,
  pad_with = "0",
  collapse_single_year_groups = FALSE,
  na_label = NA,
  return_factor = FALSE
)
```

## Arguments

| | |
|---|---|
| values | Numeric vector of ages to be grouped |
| age_breaks | Numeric vector of break points for age groups. Default: c(5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90) |
| breaks_as_lower_bound | Logical; if TRUE (default), breaks are treated as lower bounds of the intervals. If FALSE, as upper bounds. |
| first_group_format | Character string template for the first age group. Uses glue syntax. Default: "0-\{x\}", Other common styles: "<={x}", "<{x+1}" |
| interval_format | Character string template for intermediate age groups. Uses glue syntax. Default: "\{x\}-\{y\}", Other common styles: "{x} to {y}" |
| last_group_format | Character string template for the last age group. Uses glue syntax. Default: "\{x\}+", Other common styles: ">={x}",">{x-1}" |
| pad_numbers | Logical or numeric; if numeric, pad numbers up to the specified length (Tip: use 2). Not compatible with calculations within glue formats. Default: FALSE |
| pad_with | Character to use for padding numbers. Default: "0" |
| collapse_single_year_groups | Logical; if TRUE, groups spanning single years are collapsed. Default: FALSE |
| na_label | Label for NA values. If NA, keeps default NA handling. Default: NA |
| return_factor | Logical; if TRUE, returns a factor, if FALSE returns character vector. Default: FALSE |

## Value

Vector of age group labels (character or factor depending on return_factor)

## Examples

```
# Basic usage
create_agegroups(1:100)

# Custom formatting with upper bounds
create_agegroups(1:100,
  breaks_as_lower_bound = FALSE,
  interval_format = "{x} to {y}",
  first_group_format = "0 to {x}"
)

# Ages 1 to 5 are kept as numbers by collapsing single year groups
create_agegroups(1:10,
  age_breaks = c(1, 2, 3, 4, 5, 10),
  collapse_single_year_groups = TRUE
)
```

---

# geometric_mean *Compute a Geometric Mean*

---

## Description

The geometric mean is typically defined for strictly positive values. This function computes the geometric mean of a numeric vector, with the option to replace certain values (e.g., zeros, non-positive values, or values below a user-specified threshold) before computation.

## Usage

```
geometric_mean(
  x,
  na.rm = FALSE,
  replace_value = NULL,
  replace = c("all", "non-positive", "zero")
)
```

## Arguments

| | |
|---|---|
| x | A numeric or complex vector of values. |
| na.rm | Logical. If FALSE (default), the presence of zero or negative values triggers a warning and returns NA. If TRUE, such values (and any NA) are removed before computing the geometric mean. |
| replace_value | Numeric or NULL. The value used for replacement, depending on replace (e.g., a detection limit (LOD) or quantification limit (LOQ)). If NULL, no replacement is performed. For recommendations how to use, see details. |
| replace | Character string indicating which values to replace: |

`"all"` Replaces all values less than `replace_value` with `replace_value`. This is useful if you have a global threshold (such as a limit of detection) below which any measurement is replaced.

`"non-positive"` Replaces all non-positive values (`x <= 0`) with `replace_value`. This is helpful if zeros or negative values are known to be invalid or below a certain limit.

`"zero"` Replaces only exact zeros (`x == 0`) with `replace_value`. Useful if negative values should be treated as missing.

## Details

**Replacement Considerations**: The geometric mean is only defined for strictly positive numbers ($x > 0$). Despite this, the geometric mean can be useful for laboratory measurements which can contain 0 or negative values. If these values are treated as NA and are removed, this results in an upward bias due to missingness. To reduce this, values below the limit of detection (LOD) or limit of quantification (LOQ) are often replaced with the chosen limit, making this limit the practical lower limit of the measurement scale. This is therefore an often recommended approach.

There are also alternatives approaches, where values are replaced by either $\frac{LOD}{2}$ or $\frac{LOD}{\sqrt{2}}$ (or LOQ). These approaches create a gap in the distribution of values (e.g. no values for $\frac{LOD}{2} < x < LOD$) and should therefore be used with caution.

**If the replacement approach for values below LOD or LOQ has a material effect on the interpretation of the results, the values should be treated as statistically censored. In this case, proper statistical methods to handle (left) censored data should be used.**

When `replace_value` is provided, the function will *first* perform the specified replacements, then proceed with the geometric mean calculation. If no replacements are requested but zero or negative values remain and `na.rm = FALSE`, an `NA` will be returned with a warning.

## Value

A single numeric value representing the geometric mean of the processed vector `x`, or `NA` if the resulting vector is empty (e.g., if `na.rm = TRUE` removes all positive values) or if non-positive values exist when `na.rm = FALSE`.

## Examples

```
# Basic usage with no replacements:
x <- c(1, 2, 3, 4, 5)
geometric_mean(x)

# Replace all values < 0.5 with 0.5 (common in LOD scenarios):
x3 <- c(0.1, 0.2, 0.4, 1, 5)
geometric_mean(x3, replace_value = 0.5, replace = "all")

# Remove zero or negative values, since log(0) = -Inf and log(-1) = NaN
x4 <- c(-1, 0, 1, 2, 3)
geometric_mean(x4, na.rm = TRUE)
```

`geom_epicurve` *Create an epidemic curve plot or bin/count observations by date periods*

## Description

Creates a epicurve plot for visualizing epidemic case counts in outbreaks (epidemiological curves). An epicurve is a bar plot, where every case is outlined. `geom_epicurve` additionally provides date-based aggregation of cases (e.g. per week or month and many more).

- For week aggregation both isoweek (World + ECDC) and epiweek (US CDC) are supported.
- `stat_bin_date` and its alias `stat_date_count` provide date based binning only. After binning the by date, these stats behave like ggplot2::stat_count.

## Usage

```
geom_epicurve(
  mapping = NULL,
  data = NULL,
  stat = "epicurve",
  position = "stack",
  date_resolution = NULL,
  week_start = getOption("lubridate.week.start", 1),
  width = NULL,
  relative.width = 1,
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_bin_date(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "stack",
  date_resolution = NULL,
  week_start = getOption("lubridate.week.start", 1),
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_date_count(
  mapping = NULL,
  data = NULL,
```

```
  geom = "line",
  position = "stack",
  date_resolution = NULL,
  week_start = getOption("lubridate.week.start", 1),
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| mapping | Set of aesthetic mappings created by aes. Commonly used mappings:<br>• **x or y**: date or datetime. Numeric is technically supported.<br>• **fill**: for colouring groups<br>• **weight**: if data is already aggregated (e.g. case counts) |
| data | The data frame containing the variables for the plot |
| stat | either "epicurve" for outlines around cases or "bin_date" for outlines around (fill) groups. For large numbers of cases please use "bin_date" to reduce number of drawn rectangles. |
| position | Position adjustment. Currently supports "stack". |
| date_resolution | Character string specifying the time unit for date aggregation. Set to NULL or NA for no date aggregation<br>Possible values are: "day", "week", "month", "bimonth", "season", "quarter", "halfyear", "year". To special values enforce ISO or US week standard:<br>• isoweek will force dadte_resolution = week and week_start = 1 (ISO and ECDC Standard)<br>• epiweek will force date_resolution = week and week_start = 7 (US CDC Standard) |
| week_start | Integer specifying the start of the week (1 = Monday, 7 = Sunday).<br>Only used when date_resolution includes weeks. Defaults to 1 (Monday).<br>For isoweek use week_start = 1 and for epiweek use week_start = 7. |
| width | Numeric value specifying the width of the bars. If NULL, calculated based on resolution and relative.width |
| relative.width | Numeric value between 0 and 1 adjusting the relative width of bars. Defaults to 1 |
| ... | Other arguments passed to layer. For example:<br>• colour Colour of the outlines around cases. Disable with colour = NA. Defaults to "white".<br>• linewidth Width of the case outlines. |
| na.rm | If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed. |

| | |
|---|---|
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. |
| inherit.aes | If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders(). |
| geom | The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. |

## Details

Epi Curves are a public health tool for outbreak investigation. For more details see the references.

## Value

A ggplot2 geom layer that can be added to a plot

## References


- Centers for Disease Control and Prevention. Quick-Learn Lesson: Using an Epi Curve to Determine Mode of Spread. USA. https://www.cdc.gov/training/quicklearns/epimode/
- Dicker, Richard C., Fátima Coronado, Denise Koo, and R. Gibson Parrish. 2006. Principles of Epidemiology in Public Health Practice; an Introduction to Applied Epidemiology and Biostatistics. 3rd ed. USA. https://stacks.cdc.gov/view/cdc/6914


## See Also

scale_y_cases_5er(), geom_vline_year()

## Examples

```
# Basic epicurve with dates
library(ggplot2)
set.seed(1)

plot_data_epicurve_imp <- data.frame(
  date = rep(as.Date("2023-12-01") + ((0:300) * 1), times = rpois(301, 0.5))
)

ggplot(plot_data_epicurve_imp, aes(x = date, weight = 2)) +
  geom_vline_year(year_break = "01-01", show.legend = TRUE) +
  geom_epicurve(date_resolution = "week") +
  labs(title = "Epicurve Example") +
  scale_y_cases_5er() +
 scale_x_date(date_breaks = "4 weeks", date_labels = "W%V'%g") + # Correct ISOWeek labels week'year
 coord_equal(ratio = 7) + # Use coord_equal for square boxes. 'ratio' are the days per week.
  theme_bw()

# Categorical epicurve
```

```
library(tidyr)
library(outbreaks)

sars_canada_2003 |> # SARS dataset from outbreaks
  pivot_longer(starts_with("cases"), names_prefix = "cases_", names_to = "origin") |>
  ggplot(aes(x = date, weight = value, fill = origin)) +
  geom_epicurve(date_resolution = "week") +
  scale_x_date(date_labels = "W%V'%g", date_breaks = "2 weeks") +
  scale_y_cases_5er() +
  theme_classic()
```

---

`geom_epigantt` *Epi Gantt Chart: Visualize Epidemiological Time Intervals*

---

## Description

Various ways of representing a vertical interval defined by `y`, `xmin` and `xmax`. Each case draws a single graphical object.

## Usage

```
geom_epigantt(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

## Arguments

| | |
|---|---|
| `mapping` | Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping. |
| `data` | The data to be displayed in this layer. There are three options:<br>If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.<br>A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.<br>A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`). |

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used the override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

## Value

A ggplot2 geom layer that can be added to a plot

---

# geom_vline_year *Automatically create lines at the turn of every year*

---

## Description

Determines turn of year dates based on the range of either the x or y axis of the ggplot.

- geom_vline_year() draws vertical lines at the turn of each year
- geom_hline_year() draws horizontal lines at the turn of each year

## Usage

```
geom_vline_year(
  mapping = NULL,
  position = "identity",
  year_break = "01-01",
  just = -0.5,
  ...,
  show.legend = NA
)

geom_hline_year(
  mapping = NULL,
  position = "identity",
  year_break = "01-01",
  just = -0.5,
  ...,
  show.legend = NA
)
```

## Arguments

mapping — Mapping created using ggplot2::aes(). Can be used to add the lines to the legend. E.g. aes(linetype = 'End of Year'). Cannot access data specified in ggplot2::ggplot(). Panels created by ggplot2::facet_wrap() or ggplot2::facet_grid() are available with aes(linetype = PANEL).

position — Position adjustment, either as a string, or the result of a call to a position adjustment function.

| | |
|---|---|
| year_break | String specifying the month and day of the year break ("MM-DD"). Defaults to: "01-01" for January 1. |
| just | Numeric offset in days (justification). Shifts the lines from the year break date. Defaults to -0.5, which shifts the line by half a day so if falls in the middle between December 31 and January 1. |
| ... | Other arguments passed to layer. For example:<br>• colour Colour of the line. Try: colour = "grey50"<br>• linetype Linetype. Try: linetype = "dashed" or linetype = "dotted"<br>• linewidth Width of the line.<br>• alpha Transparency of the line. used to set an aesthetic to a fixed value, like colour = "grey25" or linetype = 2. |
| show.legend | logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. |

## Value

A ggplot2 layer that can be added to a plot.

## See Also

geom_epicurve(), ggplot2::geom_vline()

## Examples

```
library(ggplot2)
set.seed(1)

plot_data_epicurve_imp <- data.frame(
  date = rep(as.Date("2023-12-01") + ((0:300) * 1), times = rpois(301, 0.5))
)

ggplot(plot_data_epicurve_imp, aes(x = date, weight = 2)) +
  geom_epicurve(date_resolution = "week") +
  geom_vline_year(year_break = "01-01", show.legend = TRUE) +
  labs(title = "Epicurve Example") +
  scale_y_cases_5er() +
 scale_x_date(date_breaks = "4 weeks", date_labels = "W%V'%g") + # Correct ISOWeek labels week'year
  theme_bw()
```

---

# influenza_germany *Germany Influenza (FLU) Surveillance data*

---

## Description

A subset of the weekly German influenza surveillance data from January 2020 to January 2025.

## Usage

```
influenza_germany
```

## Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1037 rows and 4 columns.

## Details

A data frame with 1,037 rows and 4 columns:

**ReportingWeek** Reporting Week in "2024-W03" format

**AgeGroup** Age groups: `00+` for all and `00-14`, `15-59` and `60+` for age stratified cases.

**Cases** Weekly case count

**Incidence** Calculated weekly incidence

## Source

License CC-BY 4.0: Robert Koch-Institut (2025): Laborbestätigte Influenzafälle in Deutschland. Dataset. Zenodo. DOI:10.5281/zenodo.14619502. https://github.com/robert-koch-institut/Influenzafaelle_in_Deutschland

---

# scale_y_cases_5er *Continuous x-axis and y-axis scale for (case) counts*

---

## Description

A continuous ggplot scale for count data with sane defaults for breaks. It uses `base::pretty()` to increase the default number of breaks and prefers 5er breaks. Additionally, the first tick (i.e. zero) is aligned to the lower left corner.

## Usage

```
scale_y_cases_5er(
  name = waiver(),
  n = 8,
  n.min = 5,
  u5.bias = 4,
  expand = NULL,
  labels = waiver(),
  limits = NULL,
  oob = scales::censor,
  na.value = NA_real_,
  transform = "identity",
  position = "left",
  sec.axis = waiver(),
```

```
  guide = waiver(),
  ...
)

scale_x_cases_5er(
  name = waiver(),
  n = 8,
  n.min = 5,
  u5.bias = 4,
  expand = NULL,
  labels = waiver(),
  limits = NULL,
  oob = scales::censor,
  na.value = NA_real_,
  transform = "identity",
  position = "bottom",
  sec.axis = waiver(),
  guide = waiver(),
  ...
)
```

## Arguments

| | |
|---|---|
| `name` | The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted. |
| `n` | Target number of breaks passed to `base::pretty()`. Defaults to 8. |
| `n.min` | Minimum number of breaks passed to `base::pretty()`. Defaults to 5. |
| `u5.bias` | The "5-bias" parameter passed to `base::pretty()`; higher values push the breaks more strongly toward multiples of 5. Defaults to 4. |
| `expand` | Uses own expansion logic. Use `expand = waiver()` to restore ggplot defaults or `ggplot2::expansion()` to modify |
| `labels` | One of:<br>• `NULL` for no labels<br>• `waiver()` for the default labels computed by the transformation object<br>• A character vector giving labels (must be same length as `breaks`)<br>• An expression vector (must be the same length as breaks). See ?plotmath for details.<br>• A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation. |
| `limits` | One of:<br>• `NULL` to use the default scale range<br>• A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum |

- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`oob` One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with `NA`.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

`na.value` Missing values will be replaced with this value.

`transform` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `transform_<name>`. If transformations require arguments, you can call them from the scales package, e.g. `scales::transform_boxcox(p = 2)`. You can create your own transformation with `scales::new_transform()`.

`position` For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

`sec.axis` `sec_axis()` is used to specify a secondary axis.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`...` Additional arguments passed on to `base::pretty()`.

## Value

A ggplot2 scale object that can be added to a plot.

## See Also

`geom_epicurve()`, `ggplot2::scale_y_continuous()`, `base::pretty()`

## Examples

```
library(ggplot2)

data <- data.frame(date = as.Date("2024-01-01") + 0:30)
ggplot(data, aes(x = date)) +
  geom_epicurve(date_resolution = "week") +
  scale_y_cases_5er()
```

uncount *Duplicate rows according to a weighting variable*

## Description

uncount() is provided by the tidyr package, and re-exported by ggsurveillance. See tidyr::uncount() for more details.

uncount() and its alias expand_counts() are complements of dplyr::count(): they take a data frame with a column of frequencies and duplicate each row according to those frequencies.

## Usage

```
uncount(data, weights, ..., .remove = TRUE, .id = NULL)

expand_counts(data, weights, ..., .remove = TRUE, .id = NULL)
```

## Arguments

| | |
|---|---|
| data | A data frame, tibble, or grouped tibble. |
| weights | A vector of weights. Evaluated in the context of data; supports quasiquotation. |
| ... | Additional arguments passed on to methods. |
| .remove | If TRUE, and weights is the name of a column in data, then this column is removed. |
| .id | Supply a string to create a new variable which gives a unique identifier for each created row. |

## Value

A data.frame with rows duplicated according to weights.

## Examples

```
df <- data.frame(x = c("a", "b"), n = c(2, 3))
df |> uncount(n)
# Or equivalently:
df |> expand_counts(n)
```

# Index