Package: ggsurveillance (via r-universe)

September 24, 2025

Title Tools for Outbreak Investigation/Infectious Disease Surveillance

Version 0.5.1

Description Create epicurves, epigantt charts, and diverging bar charts using 'ggplot2'. Prepare data for visualisation or other reporting for infectious disease surveillance and outbreak investigation (time series data). 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', (D) labelling of the last value of a time-series. 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 (>= 4.2.0)

Imports cli, dplyr, forcats, ggplot2 (>= 3.5.0), glue, ISOweek, legendry, lubridate, rlang, scales (>= 1.4.0), stringr, tidyr, tidyselect

Suggests ggrepel, Hmisc, knitr, outbreaks, plotly, rmarkdown, spelling, testthat (>= 3.0.0), vdiffr (>= 1.0.8)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

NeedsCompilation no

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/pak/sysreqs libicu-dev

Repository https://r-multiverse.r-universe.dev

Date/Publication 2025-07-02 10:23:44 UTC

RemoteUrl https://github.com/biostats-dev/ggsurveillance

RemoteRef 0.5.1

RemoteSha c3bd91cca84f21b55bc990ade592afc2a36a7b0d

Contents

`align_dates_seasonal` *Align dates for seasonal comparison*

Description

`align_dates_seasonal()` 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.

`align_and_bin_dates_seasonal()` is a convenience wrapper that first aligns the dates and then bins the data to calculate counts and incidence.

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

Arguments

`x`	Either a data frame with a date column, or a date vector. Supported date formats are `date` and `datetime` and also commonly used character strings: • ISO dates `"2024-03-09"` • Month `"2024-03"` • Week `"2024-W09"` or `"2024-W09-1"`
`dates_from`	Column name containing the dates to align and bin. Used when x is a data.frame.

`date_resolution`
: Character string specifying the temporal resolution. One of:
  - `"week"` or `"isoweek"` - Calendar weeks (ISO 8601), epidemiological reporting weeks as used by the ECDC.
  - `"epiweek"` - Epidemiological weeks as defined by the US CDC (weeks start on Sunday).
  - `"month"` - Calendar months
  - `"day"` - Daily resolution

`start`
: Numeric value indicating epidemic season start, i.e. the start and end of the new year interval:
  - For `week/epiweek`: week number (default: 28, approximately July)
  - For `month`: month number (default: 7 for July)
  - For `day`: day of year (default: 150, approximately June) If start is set to "1" the alignment is done for yearly comparison and the shift in dates for seasonality is skipped.

`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. Set to `FALSE` to retain leap weeks from all seasons. 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 (or weights). 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. Useful for ensuring complete time series without missing periods. Defaults to `FALSE`.

`.groups`
: See `dplyr::summarise()`.

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"), if `start = 1` this is the year only (e.g. 2023).
- `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

```
# Seasonal 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() +
  theme_mod_rotate_x_axis_labels_45()
```

`bin_by_date` *Aggregate data by time periods*

Description

Aggregates data by specified time periods (e.g., weeks, months) and calculates (weighted) counts. Incidence rates are also calculated using the provided population numbers.

This function is the core date binning engine used by `geom_epicurve()` and `stat_bin_date()` for creating epidemiological time series visualizations.

Usage

```
bin_by_date(
  x,
  dates_from,
  n = 1,
  population = 1,
  fill_gaps = FALSE,
  date_resolution = "week",
  week_start = 1,
  .groups = "drop"
)
```

Arguments

x	Either a data frame with a date column, or a date vector. Supported date formats are `date` and `datetime` and also commonly used character strings: • ISO dates `"2024-03-09"` • Month `"2024-03"` • Week `"2024-W09"` or `"2024-W09-1"`
dates_from	Column name containing the dates to bin. Used when x is a data.frame.
n	Numeric column with case counts (or weights). 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. Useful for ensuring complete time series without missing periods. Defaults to `FALSE`.
date_resolution	Character string specifying the time unit for date aggregation. Possible values include: `"hour"`, `"day"`, `"week"`, `"month"`, `"bimonth"`, `"season"`, `"quarter"`, `"halfyear"`, `"year"`. Special values: • `"isoweek"`: ISO week standard (week starts Monday, `week_start = 1`) • `"epiweek"`: US CDC epiweek standard (week starts Sunday, `week_start = 7`) • `"isoyear"`: ISO year (corresponding year of the ISO week, differs from year by 1-3 days) • `"epiyear"`: Epidemiological year (corresponding year of the epiweek, differs from year by 1-3 days) Defaults to `"week"`.
week_start	Integer specifying the start of the week (1 = Monday, 7 = Sunday). Only used when `date_resolution` involves weeks. Defaults to 1 (Monday). Overridden by `"isoweek"` (1) and `"epiweek"` (7) settings.
.groups	See `dplyr::summarise()`.

Details

The function performs several key operations:

1. **Date coercion**: Converts the date column to proper Date format
2. **Gap filling** (optional): Generates complete temporal sequences to fill missing time periods with zeros
3. **Date binning**: Rounds dates to the specified resolution using `lubridate::floor_date()`
4. **Weight and population handling**: Processes count weights and population denominators
5. **Aggregation**: Groups by binned dates and sums weights to get counts and incidence

Grouping behaviour: The function respects existing grouping in the input data frame.

Value

A data frame with the following columns:

- A date column with the same name as `dates_from`, where values are binned to the start of the specified time period.
- `n`: Count of observations (sum of weights) for each time period
- `incidence`: Incidence rate calculated as `n / population` for each time period
- Any existing grouping variables are preserved

Examples

```
library(dplyr)

# Create sample data
outbreak_data <- data.frame(
  onset_date = as.Date("2024-12-10") + sample(0:100, 50, replace = TRUE),
  cases = sample(1:5, 50, replace = TRUE)
)

# Basic weekly binning
bin_by_date(outbreak_data, dates_from = onset_date)

# Weekly binning with case weights
bin_by_date(outbreak_data, onset_date, n = cases)

# Monthly binning
bin_by_date(outbreak_data, onset_date,
  date_resolution = "month"
)

# ISO week binning (Monday start)
bin_by_date(outbreak_data, onset_date,
  date_resolution = "isoweek"
) |>
  mutate(date_formatted = strftime(onset_date, "%G-W%V")) # Add correct date labels
```

```
# US CDC epiweek binning (Sunday start)
bin_by_date(outbreak_data, onset_date,
  date_resolution = "epiweek"
)

# With population data for incidence calculation
outbreak_data$population <- 10000
bin_by_date(outbreak_data, onset_date,
  n = cases,
  population = population
)
```

`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 define the the lower bounds of the intervals (e.g., a break at 5 starts the '5-9' group). If `FALSE`, breaks define the upper bound (e.g., a break at 5 ends the '0-5' group).

`first_group_format`
: Character string template for the first age group. Uses glue::glue syntax. The variable `x` represents the upper bound of the first interval. Default: `"0-{x}"`. Other common styles: `"<={x}"`, `"<{x+1}"`

`interval_format`
: Character string template for intermediate age groups. Uses glue::glue syntax. The variables `x` and `y` represent the lower and upper bounds of the interval, respectively. Default: `"{x}-{y}"`. Other common styles: `"{x} to {y}"`

`last_group_format`
: Character string template for the last age group. Uses glue::glue syntax. The variable `x` represents the lower bound of the last interval. 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 a single year (e.g., from `age_breaks = c(1, 2)`) are formatted as a single number (e.g., "1") instead of a range (e.g., "1-1"). 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"),
  warning = TRUE
)
```

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 `replace = "all"`, this value is also used as the threshold. 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.
`warning`	Disable warnings by setting it to `FALSE`. Defaults to `TRUE`.

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_bar_diverging *Create diverging bar charts, diverging area charts or other plots for opposing categorical data.*

Description

`geom_bar_diverging()` creates a diverging bar chart, i.e. stacked bars which are centred at 0. This is useful for visualizing contrasting categories like:

- case counts by contrasting categories like vaccination status or autochthonous (local) vs imported infections
- population pyramids
- likert scales for e.g. agreement (sentiment analysis)

- or any data with natural opposing groups.

`stat_diverging()` calculates the required statistics for diverging charts and can be used with different geoms. Used for easy labelling of diverging charts.

`geom_area_diverging()` creates a diverging area chart, for continuous data of opposing categories. x (or y) has to be continuous for this geom.

See `scale_x_continuous_diverging()`, `scale_y_continuous_diverging()` for the corresponding ggplot2 scales.

Usage

```
geom_bar_diverging(
  mapping = NULL,
  data = NULL,
  position = "identity",
  proportion = FALSE,
  neutral_cat = c("odd", "never", "NA", "force"),
  break_pos = NULL,
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_area_diverging(
  mapping = NULL,
  data = NULL,
  position = "identity",
  proportion = FALSE,
  neutral_cat = c("odd", "never", "NA", "force"),
  break_pos = NULL,
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_diverging(
  mapping = NULL,
  data = NULL,
  geom = "text",
  position = "identity",
  stacked = TRUE,
  proportion = FALSE,
  neutral_cat = c("odd", "never", "NA", "force"),
  break_pos = NULL,
  totals_by_direction = FALSE,
  nudge_label_outward = 0,
```

```
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). See the section Aesthetics below for more details.
data	The data to be displayed in this layer.
position	Position adjustment. For the geoms, categories will be stacked by default. Don't use position = "stack".
proportion	Logical. If TRUE, each stacked bar is normalized to 100%. Useful to plot or calculate the percentages of each category within each bar.
neutral_cat	How to handle the middle category for a odd number of factor levels. • "odd": If the number of factor levels is odd, the middle category is treated as neutral. • "never": For odd factor levels, the middle category is treated as positive. • "NA": observations with NA as category will be shown as the neutral category. By default the NA category will be in the middle (even number of levels) or the first category after the middle (odd number of levels). • "force": A neutral category is always shown. By default this will be middle (odd number of levels) or the first category after the middle (even number of levels).
break_pos	Only used for neutral_cat = c("never", "NA", "force"). Can either be a integer position or the name of a factor level. Depending on neutral_cat: • "never": The break_pos factor level will be the first category in the positive direction. • "NA": The neutral NA category will be inserted at the position specified by break_pos. For example, if break_pos = 3, the NA category will become the 3rd level, and the original 3rd level will be shifted to the 4th. • "force": break_pos determines the neutral category.
...	Other arguments passed on to layer.
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	stat_diverging(): The geometric object to use to display the data, e.g. "text" or "label".

`stacked` Logical. If `TRUE`, categories are stacked. Used instead of `position = "stack"` for specialized stacking logic.

`totals_by_direction` Logical. If `TRUE`, totals are calculated by direction. I.e. the total for the positive, negative and, if existent, neutral category.

`nudge_label_outward` Numeric. Relative value to nudge labels outward from `0`. Try `0.05`. Negative values nudge inward.

Value

A `ggplot2` geom layer that can be added to a plot.

Diverging bar charts

Diverging bar charts split categories into positive and negative directions based on factor level order. Categories in the first half of factor levels go in the negative direction, while categories in the second half go in the positive direction.

Aesthetics

Required aesthetics:

- `x` or `y`
- `diverging_groups`: Will default to `fill` if missing. A factor should be used for this aesthetic for best results. All factor levels defined will be used to determine positive, negative and neutral categories. Behaviour of the diverging bar charts can therefore be controlled by creating empty dummy factor levels.

Optional aesthetics:

- `weight`: Adjust the weight of observations. Can be used to pass case counts or incidences.

Calculated stats

The following calculated stats can be used further in aes:

- `after_stat(count)`
- `after_stat(prop)`: Proportion of the category within the stacked bar.
- `after_stat(sign)`: Direction of the category. Either `-1`, `0` or `+1`

See Also

`scale_x_continuous_diverging()`, `scale_y_continuous_diverging()`

Examples

```
# Basic example with geom_bar_diverging
library(ggplot2)
library(dplyr)
library(tidyr)

set.seed(123)
df_6cat <- data.frame(matrix(sample(1:6, 600, replace = TRUE), ncol = 6)) |>
  mutate_all(~ ordered(., labels = c("+++", "++", "+", "-", "--", "---"))) |>
  pivot_longer(cols = everything())

ggplot(df_6cat, aes(y = name, fill = value)) +
  geom_bar_diverging() + # Bars
  stat_diverging() + # Labels
  scale_x_continuous_diverging() + # Scale
  theme_classic()

ggplot(df_6cat, aes(y = name, fill = value)) +
  geom_bar_diverging() + # Bars
 stat_diverging(totals_by_direction = TRUE, nudge_label_outward = 0.05) + # Totals as Label
  scale_x_continuous_diverging() + # Scale
  theme_classic()

# Population pyramid
population_german_states |>
  filter(state %in% c("Berlin", "Mecklenburg-Vorpommern"), age < 90) |>
  ggplot(aes(y = age, fill = sex, weight = n)) +
  geom_bar_diverging(width = 1) +
  geom_vline(xintercept = 0) +
  scale_x_continuous_diverging(n.breaks = 10) +
  facet_wrap(~state, scales = "free_x") +
  theme_bw()

# Vaccination status: set neutral category
set.seed(456)
cases_vacc <- data.frame(year = 2017:2025) |>
  rowwise() |>
 mutate(vacc = list(sample(1:4, 100, prob = (4:1)^(1 - 0.2 * (year - 2017)), replace = TRUE))) |>
  unnest(vacc) |>
  mutate(
    year = as.factor(year),
    "Vaccination Status" = ordered(vacc,
      labels = c("Fully Vaccinated", "Partially Vaccinated", "Unknown", "Unvaccinated")
    )
  )

ggplot(cases_vacc, aes(y = year, fill = `Vaccination Status`)) +
  geom_vline(xintercept = 0) +
  geom_bar_diverging(proportion = TRUE, neutral_cat = "force", break_pos = "Unknown") +
  stat_diverging(
    size = 3, proportion = TRUE, neutral_cat = "force", break_pos = "Unknown",
    totals_by_direction = TRUE, nudge_label_outward = 0.05
```

```
  ) +
  scale_x_continuous_diverging(labels = scales::label_percent(), n.breaks = 10) +
  scale_y_discrete_reverse() +
  ggtitle("Proportion of vaccinated cases by year") +
  theme_classic() +
  theme_mod_legend_bottom()
```

`geom_col_range` *Create a ranged bar chart*

Description

Creates a bar chart with explicitly defined ranges.

Usage

```
geom_col_range(
  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: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. 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. 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`	Defaults to "identity".
`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.

Aesthetics

Required aesthetics:

- Either `x` or `y`
- Either `xmin` and `xmax` or `ymin` and `ymax`

Examples

```
# Basic example
library(ggplot2)
df <- data.frame(x = 1:3, ymin = -1:-3, ymax = 1:3)
ggplot(df, aes(x = x, ymin = ymin, ymax = ymax)) +
  geom_col_range()
```

`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) using bin_by_date.

- 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 with bin_by_date, these stats behave like ggplot2::stat_count.
- `geom_epicurve_text` adds text labels to cases on epicurve plots.
- `geom_epicurve_point` adds points/shapes to cases on epicurve plots.

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 = "identity",
```

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

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

geom_epicurve_text(
  mapping = NULL,
  data = NULL,
  stat = "epicurve",
  vjust = 0.5,
  date_resolution = NULL,
  week_start = getOption("lubridate.week.start", 1),
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_epicurve_point(
  mapping = NULL,
  data = NULL,
  stat = "epicurve",
  vjust = 0.5,
  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: • **x or y**: date or datetime. Numeric is technically supported. • **fill**: for colouring groups. • **weight**: if data is already aggregated (e.g., case counts).
data	The data frame containing the variables for the plot
stat	For the geoms, use "epicurve" (default) to outline individual cases, or "bin_date" to aggregate data by group. For large datasets, "bin_date" is recommended for better performance by drawing less rectangles.
position	Position adjustment. Currently supports "stack" for geom_epicurve().
date_resolution	Character string specifying the time unit for date aggregation. If NULL (default), no date binning is performed. Possible values include: "hour", "day", "week", "month", "bimonth", "season", "quarter", "halfyear", "year". Special values: • "isoweek": ISO week standard (week starts Monday, week_start = 1) • "epiweek": US CDC epiweek standard (week starts Sunday, week_start = 7) • "isoyear": ISO year (corresponding year of the ISO week, differs from year by 1-3 days) • "epiyear": Epidemiological year (corresponding year of the epiweek, differs from year by 1-3 days) Defaults to NULL, i.e. no binning.
week_start	Integer specifying the start of the week (1 = Monday, 7 = Sunday). Only used when date_resolution involves weeks. Defaults to 1 (Monday). Overridden by "isoweek" (1) and "epiweek" (7) settings.
width	Numeric value specifying the width of the bars. If NULL, calculated based on date_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: • **colour**: Colour of the outlines around cases. Disable with colour = NA. Defaults to "white". • **linewidth**: Width of the case outlines. For geom_epicurve_text() additional geom_text arguments are supported: • **fontface**: Font face for text labels: one of "plain", "bold", "italic", "bold.italic". • **family**: The font family. • **size**: The font size.
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.
fill_gaps	Logical; If TRUE, gaps in the time series will be filled with a count of 0. Often needed for line charts.
vjust	Vertical justification of the text or shape. Value between 0 and 1. Used by geom_epicurve_text and geom_epicurve_point to control vertical positioning within the case rectangles. Defaults to 0.5 (center).

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(break_type = "week") +
  geom_epicurve(date_resolution = "week") +
  labs(title = "Epicurve Example") +
  scale_y_cases_5er() +
  # Correct ISOWeek labels for week-year
  scale_x_date(date_breaks = "4 weeks", date_labels = "W%V'%g") +
 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

Creates Epi Gantt charts, which are specialized timeline visualizations used in outbreak investigations to track potential exposure periods and identify transmission patterns. They are particularly useful for:

- Hospital outbreak investigations to visualize patient movements between wards
- Identifying potential transmission events by showing when cases were in the same location
- Visualizing common exposure times using overlapping exposure time intervals

The chart displays time intervals as horizontal bars, typically with one row per case/patient. Different colours can be used to represent different locations (e.g., hospital wards) or exposure types. Additional points or markers can show important events like symptom onset or test dates.

`geom_epigantt()` will adjust the linewidth depending on the number of cases.

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. Must include: • y: Case/patient identifier • xmin: Start date/time of interval • xmax: End date/time of interval • Optional: colour or fill for different locations/categories
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). 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. 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	A ggplot2 stat. Defaults to "identity".
position	A ggplot2 position. Defaults to "identity".
...	Other arguments passed to ggplot2::layer(). The following parameters are specific to geom_epigantt: • linewidth: Set width of bars directly, disables auto-scaling if set. • lw_scaling_factor: Scaling factor for auto-width calculation. The linewidth is calculated as lw_scaling_factor/number_of_rows (default: 90) • lw_min: Minimum auto-scaled line width cutoff (default: 1) • lw_max: Maximum auto-scaled line width cutoff (default: 8)
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

See Also

theme_mod_legend_bottom()

Examples

```
library(dplyr)
library(tidyr)
library(ggplot2)

# Transform hospital outbreak line list to long format
linelist_hospital_outbreak |>
  pivot_longer(
    cols = starts_with("ward"),
    names_to = c(".value", "num"),
    names_pattern = "ward_(name|start_of_stay|end_of_stay)_([0-9]+)",
    values_drop_na = TRUE
  ) -> df_stays_long

linelist_hospital_outbreak |>
  pivot_longer(cols = starts_with("pathogen"), values_to = "date") -> df_detections_long

# Create Epi Gantt chart showing ward stays and test dates
ggplot(df_stays_long) +
  geom_epigantt(aes(y = Patient, xmin = start_of_stay, xmax = end_of_stay, color = name)) +
  geom_point(aes(y = Patient, x = date, shape = "Date of pathogen detection"),
    data = df_detections_long
  ) +
  scale_y_discrete_reverse() +
  theme_bw() +
  theme_mod_legend_bottom()
```

`geom_label_last_value, stat_last_value`
Add labels or points to the last value of a line chart

Description

Creates a label, point or any geom at the last point of a line (highest x value). This is useful for line charts where you want to identify each line at its endpoint, write the last value of a time series at the endpoint or just add a point at the end of a `geom_line`. This functions also nudges the last value relative to the length of the x-axis. The function automatically positions the label slightly to the right of the last point. There are 5 functions:

- `stat_last_value()`: The core statistical transformation that identifies the last point of a line (e.g. last date of the time series).
- `geom_label_last_value()`: Adds the last y value or a custom label after the last observation using `geom_label`.
- `geom_text_last_value()`: Adds the last y value or a custom text after the last observation using `geom_text`.
- `geom_label_last_value_repel()`: Adds non-overlapping labels with `geom_label_repel`.
- `geom_text_last_value_repel()`: Adds non-overlapping text with `geom_text_repel`.

Usage

```
stat_last_value(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  nudge_rel = 0,
  nudge_add = 0,
  expand_rel = 0,
  expand_add = 0,
  labeller = NULL,
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_label_last_value(
  mapping = NULL,
  data = NULL,
  stat = "last_value",
  position = "identity",
  nudge_rel = 0.015,
  nudge_add = 0,
  expand_rel = 0.05,
  expand_add = 0,
  labeller = NULL,
  hjust = 0,
  ...,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

geom_text_last_value(
  mapping = NULL,
  data = NULL,
  stat = "last_value",
  position = "identity",
  nudge_rel = 0.015,
  nudge_add = 0,
  expand_rel = 0.035,
  expand_add = 0,
  labeller = NULL,
  hjust = 0,
  ...,
  na.rm = FALSE,
  show.legend = FALSE,
```

```
  inherit.aes = TRUE
)

geom_label_last_value_repel(
  mapping = NULL,
  data = NULL,
  stat = "last_value_repel",
  position = "identity",
  nudge_rel = 0.03,
  nudge_add = 0,
  expand_rel = 0.05,
  expand_add = 0,
  labeller = NULL,
  hjust = 0,
  direction = "y",
  min.segment.length = 0.5,
  ...,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

geom_text_last_value_repel(
  mapping = NULL,
  data = NULL,
  stat = "last_value_repel",
  position = "identity",
  nudge_rel = 0.015,
  nudge_add = 0,
  expand_rel = 0.035,
  expand_add = 0,
  labeller = NULL,
  hjust = 0,
  direction = "y",
  min.segment.length = 0.5,
  ...,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes. Commonly used mappings: • **x**: position on x-axis • **y**: position on y-axis • **label**: text to display (defaults to the last y value)
data	The data frame containing the variables for the plot

`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.
`position`	Position adjustment. Defaults to "identity"
`nudge_rel`	Numeric value specifying how far to nudge the label to the right, relative to the range of the x-values of the data. Defaults to 0.015 (1.5% of axis width) for labels.
`nudge_add`	Numeric value specifying an absolute amount to nudge the label (in units of the x-axis).
`expand_rel`	Numeric value specifying how far to expand the axis limits, relative to the range of the x-values of the data. This can be used to create room for longer text/labels. For repel functions this has to be large enough to place the text to achieve good results.
`expand_add`	Numeric value specifying an absolute amount to expand the axis limits (in units of the x-axis).
`labeller`	Label function to format the last value. E.g. `scales::label_percent()`, `scales::label_number()`, `scales::label_dictionary()`.
`...`	Other arguments passed to `geom_label`, `geom_text`, `geom_label_repel` or `geom_text_repel`.
`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()`.
`stat`	The statistical transformation to use on the data. Defaults to "last_value"
`hjust`	Horizontal text alignment. Defaults to left aligned (0).
`direction`	Direction in which to repel the labels. See `geom_text_repel`.
`min.segment.length`	Minimum length of the leader line segments. See `geom_text_repel`.

Details

The following calculated stats can be used further in aes:

- `after_stat(x0)`: the highest x value
- `after_stat(y)`: the y value of the observation with the highest x value.
- `after_stat(label_formatted)`: the formatted y value using the `labeller`.

Value

A `ggplot2` layer that can be added to a plot

Examples

```
# Basic example with last value labels
library(ggplot2)

ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  geom_text_last_value()

# Percentages
ggplot(economics, aes(x = date, y = unemploy / pop)) +
  geom_line() +
  geom_label_last_value(labeller = scales::label_percent(accuracy = 0.1))

# Multiple lines with custom labels
ggplot(economics_long, aes(x = date, y = value, color = variable)) +
  geom_line() +
  stat_last_value() + # Add a point at the end
  geom_label_last_value_repel(aes(label = variable),
    expand_rel = 0.1, nudge_rel = 0.05
  ) +
  scale_y_log10() +
  theme_mod_disable_legend()
```

---

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,
  year_break = "01-01",
  break_type = c("day", "week", "isoweek", "epiweek"),
  just = NULL,
  ...,
  show.legend = NA
)

geom_hline_year(
  mapping = NULL,
  year_break = "01-01",
  break_type = c("day", "week", "isoweek", "epiweek"),
```

```
  just = NULL,
  ...,
  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).
year_break	String specifying the month and day ("MM-DD") or week ("W01") of the year break . Defaults to: "01-01" for January 1. "Week" and "MM-DD" are converted automatically based on a leap year (366 days) which starts on Monday.
break_type	String specifying the type of break to use. Options are: • "day" (default): Line drawn based on the specified day for each visible year. • "week" or "isoweek": Line drawn based on the Monday of the specified week for each visible year. (e.g., "W01" for new year or "W40" for start of influenza season) • "epiweek": same as week, but the line is drawn one day earlier (Sunday).
just	Numeric offset in days (justification). Shifts the lines from the year break date. Defaults to -0.5 for day, which shifts the line by half a day so it falls between December 31 and January 1 by default. Defaults to -3.5 (i.e. half a week) for week, isoweek and epiweek.
...	Other arguments passed to layer. For example: • colour Colour of the line. Try: colour = "grey50" • linetype Linetype. Try: linetype = "dashed" or linetype = "dotted" • linewidth Width of the line. • alpha Transparency of the line. used to set an aesthetic to a fixed value.
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))
)

# Break type day
ggplot(plot_data_epicurve_imp, aes(x = date, weight = 2)) +
  geom_epicurve(date_resolution = "week") +
  geom_vline_year() +
  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()

# Break type week
ggplot(plot_data_epicurve_imp, aes(x = date, weight = 2)) +
  geom_epicurve(date_resolution = "week") +
  geom_vline_year(break_type = "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
  theme_bw()
```

`guide_axis_nested_date` *Nested axis guide for date scales*

Description

A specialized axis guide for date scales that creates nested axis labels by automatically detecting hierarchical patterns in date labels (e.g., separating day-month from year components). This guide is particularly useful for time series data, where the axis can get crowded when showing the full dates. This is similar to the date scale from Excel.

Usage

```
guide_axis_nested_date(
  sep = "[^[:alnum:]]+",
  regular_key = "auto",
  type = "bracket",
  mode = "simple",
  pad_date = NULL,
  oob = "none",
  ...
)
```

Arguments

`sep`	A regular expression pattern used to split axis labels into hierarchical components. Default is `"[^[:alnum:]]+"` which splits on non-alphanumeric characters.

regular_key	Default is "auto", which generates the nested axis based on the date labels and the separator above. This option can be used to provide your own specification for the nested key. See legendry::key_standard()
type	The visual type of nested axis guide to create. Options include: • "bracket" (default): Creates bracket-style nested labels • "fence": Creates fence-style nested labels (like Excel) • "box": Creates box-style nested labels
mode	Processing mode for the guide. Default is "simple". Currently, this is the only supported mode.
pad_date	Numeric value controlling the padding around date levels, i.e. extending the length of the bracket or box or for correctly positioning the fences. If NULL (default), automatically sets to 0.5 for "fence" type and 0.25 for other types.
oob	How to handle out-of-bounds values of the scale labels. Default is "none". Another option is "squish", but this can result in overlapping labels.
...	Additional arguments passed to legendry::guide_axis_nested().

Value

A nested axis guide object that can be used with ggplot2::scale_x_date() etc. or ggplot2::guides().

See Also

legendry::guide_axis_nested()

Examples

```
library(ggplot2)

# Create sample epidemic curve data
epi_data <- data.frame(
  date = rep(as.Date("2023-12-15") + 0:100, times = rpois(101, 2))
)

ggplot(epi_data, aes(x = date)) +
  geom_epicurve(date_resolution = "week") +
  scale_x_date(
    date_breaks = "2 weeks", date_labels = "%d-%b-%Y",
    guide = guide_axis_nested_date()
  )

# Using fence type with ISO week labels
ggplot(epi_data, aes(x = date)) +
  geom_epicurve(date_resolution = "week") +
  scale_x_date(
    date_breaks = "2 weeks", date_labels = "W%V.%G",
    guide = guide_axis_nested_date(type = "fence")
  )

# Using box type with custom padding
```

```
ggplot(epi_data, aes(x = date)) +
  geom_epicurve(date_resolution = "month") +
  scale_x_date(
    date_breaks = "1 month", date_labels = "%b.%Y",
    guide = guide_axis_nested_date(type = "box", pad_date = 0.3)
  )

# Custom separator for different label formats
ggplot(epi_data, aes(x = date)) +
  geom_epicurve(date_resolution = "week") +
  scale_x_date(
    date_breaks = "1 week", date_labels = "%d-%b-%Y",
    guide = guide_axis_nested_date(type = "bracket", sep = "-")
  )

# Datetime example with fence type
datetime_data <- data.frame(
  datetime = rep(as.POSIXct("2024-02-05 01:00:00") + 0:50 * 3600,
    times = rpois(51, 3)
  )
)

ggplot(datetime_data, aes(x = datetime)) +
  geom_epicurve(date_resolution = "2 hours") +
  scale_x_datetime(
    date_breaks = "6 hours", date_labels = "%Hh %e.%b",
    limits = c(as.POSIXct("2024-02-04 22:00:00"), NA),
    guide = guide_axis_nested_date()
  )
```

`influenza_germany` *German Influenza (FLU) Surveillance data*

Description

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

Usage

```
influenza_germany
```

Format

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

Examples

```
library(ggplot2)

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

ggplot(df_flu_aligned, aes(x = date_aligned, y = Incidence, color = season)) +
  geom_line() +
  facet_wrap(~AgeGroup) +
  theme_bw() +
  theme_mod_rotate_x_axis_labels_45()
```

---

label_date *Date labeller*

Description

Re-export from the scales package.

- Can be used to overwrite the default locale of date labels.
- `label_date_short()` only labels part of the dates, when they change, i.e. year is only labelled when the year changes.
- See `scales::label_date()` and `scales::label_date_short()` for more details.

Usage

```
label_date(format = "%Y-%m-%d", tz = "UTC", locale = NULL)

label_date_short(
  format = c("%Y", "%b", "%d", "%H:%M"),
  sep = "\n",
  leading = "0",
  tz = "UTC",
  locale = NULL
)
```

Arguments

format	For label_date() and label_time() a date/time format string using standard POSIX specification. See strptime() for details. For label_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
locale	Locale to use when for day and month names. The default uses the current locale. Setting this argument requires stringi, and you can see a complete list of supported locales with stringi::stri_locale_list().
sep	Separator to use when combining date formats into a single string.
leading	A string to replace leading zeroes with. Can be "" to disable leading characters or "\u2007" for figure-spaces.

Value

A character vector of formatted dates.

Examples

```
library(tidyr)
library(outbreaks)
library(ggplot2)

# Change locale of date labels to Italian
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(labels = label_date("%B %Y", locale = "it"), date_breaks = "1 month") +
  scale_y_cases_5er() +
  theme_classic()

# label_date_short()
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(labels = label_date_short(), date_breaks = "1 week") +
  scale_y_cases_5er() +
  theme_classic()
```

---

label_power10 *Format numbers as power-of-10 R expressions*

Description

Creates a labeller function that formats numbers in scientific notation using power-of-10 R expressions (e.g., 2.5×10^3 or 5×10^6). Useful for axis labels in ggplot2 when dealing with large numbers or when you want to emphasize the order of magnitude.

Usage

```
label_power10(
  decimal.mark = NULL,
  digits = 3,
  scale = 1,
  prefix = "",
  suffix = "",
  magnitude_only = FALSE,
  ...
)
```

Arguments

`decimal.mark`	Character used as decimal separator. If `NULL` (default), retrieves the setting from [`scales::number_options()`].
`digits`	Number of significant digits to show in the mantissa.
`scale`	Scaling factor multiplied to the input values. Default is `1`.
`prefix`	Character string to prepend to each label. Default is `""`.
`suffix`	Character string to append to each label. Default is `""`.
`magnitude_only`	Logical. If `TRUE`, shows only the power-of-10 part (e.g., 10^5 instead of 1×10^5). Default is `FALSE`.
`...`	Additional arguments passed to `scales::scientific()`.

Details

The function converts numbers to scientific notation and then formats them as mathematical expressions using the R expression syntax:

- For exponent 0: returns the mantissa as-is (e.g., 5.5)
- For exponent 1: it omits the exponent (e.g., 1.5×10)
- For other exponents: everything is shown (e.g., 1.5×10^3)

When `magnitude_only = TRUE`:

- For exponent 0: returns 1
- For exponent 1: returns 10
- For other exponents (positive or negative): returns $10^{exponent}$

The function handles negative numbers by preserving the sign and supports custom decimal marks, prefixes, and suffixes.

Value

A label function that takes a numeric vector and returns an expression vector suitable for use as axis labels in ggplot2.

Examples

```
library(ggplot2)

# Basic usage with default settings
label_power10()(c(1000, 10000, 100000, -1000))

# Use in ggplot2
ggplot(
  data.frame(x = 1:5, y = c(1, 50000, 75000, 100000, 200000)),
  aes(x, y)
) +
  geom_point() +
  scale_y_continuous(labels = label_power10())

# Use in ggplot2 with options
ggplot(
  data.frame(x = 1:5, y = c(1, 50000, 75000, 100000, 200000)),
  aes(x, y)
) +
  geom_point() +
 scale_y_continuous(labels = label_power10(decimal.mark = ",", digits = 2, suffix = " CFU"))

# Magnitude only for cleaner labels with log scales
ggplot(
  data.frame(x = 1:5, y = c(1000, 10000, 100000, 1000000, 10000000)),
  aes(x, y)
) +
  geom_point() +
  scale_y_log10(labels = label_power10(magnitude_only = TRUE))
```

---

label_skip *Skip labels on an axis*

Description

Creates a labeller function that removes every n-th label on an ggplot2 axis. Useful for reducing overlapping labels while keeping the major ticks.

Usage

```
label_skip(n = 2, start = c("left", "right"), labeller = NULL)
```

Arguments

n	Integer. Display every nth label. Default is 2.
start	Where to start the pattern. Either "left" for first tick (default), "right" for last tick, or an integer position (i.e. 1 for first tick, 2 for second tick, etc.).
labeller	Optional function to transform labels before applying skip pattern. For example label_date(). For more complex labeller combinations use scales::compose_label().

Value

A function that takes a vector of labels and returns a vector with skipped labels replaced by empty strings.

Examples

```
library(ggplot2)
# Default skip labels
ggplot(mtcars, aes(x = mpg, y = wt)) +
  geom_point() +
  scale_x_continuous(labels = label_skip())

# Skip date labels, while keep ticks
ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line() +
  scale_x_date(
    date_breaks = "2 years",
    labels = label_skip(start = "right", labeller = label_date(format = "%Y"))
  ) +
  theme_bw()
```

---

linelist_hospital_outbreak

Line list of a fictional hospital outbreak (Data)

Description

This hospital outbreak is inspired by typical hospital outbreaks with resistant 4MRGN bacterial pathogens. These outbreaks start silent, since they are not initially apparent from the symptoms of the patient.

Usage

```
linelist_hospital_outbreak
```

Format

A data frame with 8 rows and 9 columns:

- `Patient` - Patient ID (0-7)
- `ward_name_1` - Name of first ward where patient stayed
- `ward_start_of_stay_1` - Start date of stay in first ward
- `ward_end_of_stay_1` - End date of stay in first ward
- `ward_name_2` - Name of second ward where patient stayed (if applicable)
- `ward_start_of_stay_2` - Start date of stay in second ward (if applicable)
- `ward_end_of_stay_2` - End date of stay in second ward (if applicable)
- `pathogen_detection_1` - Date of first positive pathogen test
- `pathogen_detection_2` - Date of second positive pathogen test (if applicable)

Patient details:

- Patient 0: Index case (ICU), infected early on but detected June 30, 2024
- Patient 1-2: ICU patients, found during initial screening
- Patient 3: Case who moved from ICU to general ward prior to the detection of patient 0, potentially linking both outbreak clusters. Detected during extended case search
- Patient 4-6: General ward cases, found after Patient 3's detection
- Patient 7: General ward case, detected post-discharge by GP, who notified the hospital

Examples

```
library(dplyr)
library(tidyr)
library(ggplot2)

# Transform hospital outbreak line list to long format
linelist_hospital_outbreak |>
  pivot_longer(
    cols = starts_with("ward"),
    names_to = c(".value", "num"),
    names_pattern = "ward_(name|start_of_stay|end_of_stay)_([0-9]+)",
    values_drop_na = TRUE
  ) -> df_stays_long

linelist_hospital_outbreak |>
  pivot_longer(cols = starts_with("pathogen"), values_to = "date") -> df_detections_long

# Create Epi Gantt chart showing ward stays and test dates
ggplot(df_stays_long) +
  geom_epigantt(aes(y = Patient, xmin = start_of_stay, xmax = end_of_stay, color = name)) +
  geom_point(aes(y = Patient, x = date, shape = "Date of pathogen detection"),
    data = df_detections_long
  ) +
  scale_y_discrete_reverse() +
```

```
  theme_bw() +
  theme(legend.position = "bottom")
```

`population_german_states`
Population of the German states (2023)

Description

German Population data by state in 2023

Usage

```
population_german_states
```

Format

A data frame with 2912 rows and 5 columns:

reporting_date Date: Always "2023-12-31"

state Character: Name of the German state

age Numeric: Age from 0 to 89. Age 90 includes "90 and above"

sex Factor: "female" or "male"

n Numeric: Population size

Source

© Statistisches Bundesamt (Destatis), Genesis-Online, 2025: Bevölkerung: Bundesländer, Stichtag, Geschlecht, Altersjahre (12411-0013). Data licence Germany (dl-de/by-2-0) https://www-genesis.destatis.de/datenbank/online/statistic/12411/table/12411-0013

Examples

```
# Population pyramid
library(ggplot2)
library(dplyr)
population_german_states |>
  filter(age < 90) |>
  ggplot(aes(y = age, fill = sex, weight = n)) +
  geom_bar_diverging(width = 1) +
  geom_vline(xintercept = 0) +
  scale_x_continuous_diverging() +
  facet_wrap(~state, scales = "free_x") +
  theme_bw(base_size = 8) +
  theme_mod_legend_top()
```

scale_continuous_diverging *Diverging continuous scales for diverging bar charts with symmetrical limits*

Description

These scales automatically create symmetrical limits around a centre point (zero by default). They're useful for diverging continuous variables where the visual encoding should be balanced around a center point, such as positive and negative values. They are intended to be used with `geom_bar_diverging()`, `geom_area_diverging()` and `stat_diverging()`.

Usage

```
scale_x_continuous_diverging(
  name = waiver(),
  limits = waiver(),
  labels = NULL,
  transform = "identity",
  ...,
  breaks = waiver(),
  n.breaks = NULL,
  expand = waiver(),
  position = "bottom"
)

scale_y_continuous_diverging(
  name = waiver(),
  limits = NULL,
  labels = NULL,
  transform = "identity",
  ...,
  breaks = waiver(),
  n.breaks = NULL,
  expand = waiver(),
  position = "left"
)
```

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.
`limits`	Numeric vector of length two providing limits of the scale. If `waiver()` (the default), limits are automatically computed to be symmetrical around zero. Use `NULL` for default `ggplot2` limits.

labels	Either `waiver()`, a character vector or a function that takes the breaks as input and returns labels as output. By default, absolute values are displayed or passed to the label function.
transform	Defaults to "identity". Use "reverse" to invert the scale. Especially useful to flip the direction of diverging bar charts.
...	Other arguments passed on to `scale_(x|y)_continuous()`
breaks	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.
n.breaks	An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.
expand	For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.
position	For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

Value

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

See Also

`geom_bar_diverging()`, `geom_area_diverging()`, `stat_diverging()`

Examples

```
library(ggplot2)

# Create sample data with positive and negative values
df <- data.frame(
  x = c(-5, -2, 0, 3, 7),
  y = c(2, -1, 0, -3, 5)
)

# Basic usage
ggplot(df, aes(x, y)) +
  geom_point() +
  scale_x_continuous_diverging() +
```

```
scale_y_continuous_diverging()
```

`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,
  min.n = 5,
  u5.bias = 4,
  expand = NULL,
  limits = c(0, NA),
  labels = waiver(),
  oob = scales::censor,
  na.value = NA_real_,
  transform = "identity",
  position = "left",
  sec.axis = waiver(),
  guide = waiver(),
  ...
)

scale_x_cases_5er(
  name = waiver(),
  n = 8,
  min.n = 5,
  u5.bias = 4,
  expand = NULL,
  limits = c(0, NA),
  labels = waiver(),
  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.
min.n	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
limits	The lower limit defaults to 0 and the upper limits is chosen based on the data. This is the recommended approach for visualizing case numbers and incidences, i.e. the scale starts at 0 and is only positive. To use the default `ggplot2` limits use `limits = NULL`.
labels	One of: • `NULL` for no labels • `waiver()` for the default labels computed by the transformation object • A character vector giving labels (must be same length as `breaks`) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.
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(), theme_mod_remove_minor_grid_y()

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() +
  theme_mod_remove_minor_grid_y()
```

---

scale_y_discrete_reverse

Reversed discrete scale for 'ggplot2'

Description

scale_y_discrete_reverse() and scale_x_discrete_reverse() are standard discrete 'ggplot2' scales with a reversed order of values. Since the ggplot2 coordinate system starts with 0 in the lower left corner, factors on the y-axis are sorted is descending order by default (i.e. alphabetically from Z to A). With this scale the the y-axis will start with the first factor level at the top or with alphabetically correctly ordered values

Usage

```
scale_y_discrete_reverse(
  name = waiver(),
  limits = NULL,
  ...,
  expand = waiver(),
  position = "left"
)

scale_x_discrete_reverse(
  name = waiver(),
  limits = NULL,
  ...,
  expand = waiver(),
  position = "bottom"
)
```

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.
limits	Can be either NULL which uses the default reversed scale values or a character vector which will be reversed.
...	Arguments passed on to `ggplot2::discrete_scale()`
expand	For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.
position	For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

Value

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

See Also

`geom_epigantt()`, `ggplot2::scale_y_discrete()`

Examples

```
library(ggplot2)

# Create sample data
df <- data.frame(
  category = factor(c("A", "B", "C", "D")),
  value = c(10, 5, 8, 3)
)

# Basic plot with reversed y-axis
ggplot(df, aes(x = value, y = category)) +
  geom_col() +
  scale_y_discrete_reverse()
```

---

theme_mod_disable_legend, theme_mod_legend_position

Quickly adjust the legend position

Description

Convenience functions to control the legend position for ggplot2. Has to be called after setting the theme.

Usage

```
theme_mod_disable_legend()

theme_mod_legend_position(
  position = c("top", "bottom", "left", "right", "none", "inside"),
  position.inside = NULL
)

theme_mod_legend_top()

theme_mod_legend_bottom()

theme_mod_legend_left()

theme_mod_legend_right()

theme_mod_remove_legend_title()
```

Arguments

`position`	Position of the ggplot2 legend. Options are `("top", "bottom", "left", "right", "none", "inside"`
`position.inside`	Coordinates for the legend inside the plot. If set overwrites `position` to `inside`.

Value

Changes the `legend.position` of the `ggplot2::theme()`.

---

theme_mod_remove_minor_grid

Quickly remove the minor lines of the panel grid

Description

`theme_mod_remove_minor_grid()`, `theme_mod_remove_minor_grid_x()`, `theme_mod_remove_minor_grid_y()` are convenience functions remove the minor lines of the panel grid. Has to be called after setting the theme.

Usage

```
theme_mod_remove_minor_grid()

theme_mod_remove_minor_grid_y()

theme_mod_remove_minor_grid_x()

theme_mod_remove_panel_grid()
```

Value

Changes the `panel.grid.minor` of the `ggplot2::theme()`.

`theme_mod_rotate_axis_labels` *Rotate axis labels*

Description

Rotate axis labels by 90°, 45° or any angle. Has to be called after setting the theme.

Usage

```
theme_mod_rotate_x_axis_labels(
  angle = 90,
  margin_top = 2,
  vjust = 0.4,
  hjust = 0,
  ...
)

theme_mod_rotate_x_axis_labels_90(angle = 90, ...)

theme_mod_rotate_x_axis_labels_45(angle = 45, ...)

theme_mod_rotate_x_axis_labels_30(angle = 30, ...)

theme_mod_rotate_x_axis_labels_60(angle = 60, ...)

theme_mod_rotate_y_axis_labels(angle = 90, hjust = 0.5, vjust = 0, ...)
```

Arguments

`angle`	Angle of rotation. Should be between 10 and 90 degrees.
`margin_top`	Used to move the tick labels downwards to prevent text intersecting the x-axis. Increase for angled multiline text (e.g. 5 for two lines at 45°).
`hjust, vjust`	Text justification within the rotated text element. Just ignore.
`...`	Arguments passed to `theme_mod_rotate_x_axis_labels` and `ggplot2::element_text()`.

Value

Changes the rotation of the axis labels by modifying the `axis.text` of the `ggplot2::theme()`.

uncount, expand_counts

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