Package 'ggsurveillance'

March 2, 2025

Title Tools for Outbreak Investigation/Infectious Disease Surveillance

Version 0.2.0

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 2025-03-02 10:40:02 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. 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. Used when x is a data.frame.
date_resolution	Character string specifying the temporal resolution. One of: • `"week"` or `"isoweek"` - Calendar weeks (ISO 8601), reporting weeks as used by the ECDC. • `"epiweek"` - Epidemiological weeks (US CDC), i.e. ISO weeks with Sunday as week start. • `"month"` - Calendar months • `"day"` - Daily resolution
start	Numeric value indicating epidemic season start: • 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)
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"),
  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 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_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 = "identity",
  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 = "identity",
  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	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 the number of drawn rectangles.
position	Position adjustment. Currently supports "stack" for geom_epicurve().
date_resolution	Character string specifying the time unit for date aggregation. Set to NULL or NA for no date aggregation Possible values are: "day", "week", "month", "bimonth", "season", "quarter", "halfyear", "year". To special values enforce ISO or US week standard: • isoweek will force dadte_resolution = week and week_start = 1 (ISO and ECDC Standard) • 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). Only used when date_resolution includes weeks. Defaults to 1 (Monday). 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: • colour Colour of the outlines around cases. Disable with colour = NA. Defaults to "white". • 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

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".

... — Additional parameters:

- 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

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")
```

`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: • 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, 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

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")
```

`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: • `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.
limits	One of: • `NULL` to use the default scale range • 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()
```

---

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()
```

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