Package: growthrates (via r-universe)

November 18, 2024

Title Estimate Incidence, Proportions and Exponential Growth Rates

Version 0.2.0

Description Simple statistical models and visualisations for calculating the incidence, proportion, exponential growth rate, and reproduction number of infectious disease case timeseries. This toolkit was largely developed during the COVID-19 pandemic.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Suggests downloader, fs, knitr, rappdirs, rmarkdown, here, jsonlite, readr, stringr, testthat (>= 3.0.0), devtools, patchwork, memoise

Remotes github::bristol-vaccine-centre/interfacer

VignetteBuilder knitr

Imports dplyr, magrittr, nnet, rlang, stats, tibble, tidyr, locfit, lubridate, scales, tidyselect, ggplot2, glue, purrr, EpiEstim, interfacer (>= 0.2.0), ragg

URL https://bristol-vaccine-centre.github.io/growthrates/, https://github.com/bristol-vaccine-centre/growthrates, https://doi.org/10.5281/zenodo.7242761

Config/testthat/edition 3

Depends R (>= 2.10)

LazyData true

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libxml2-dev

Repository https://bristol-vaccine-centre.r-universe.dev

RemoteUrl https://github.com/bristol-vaccine-centre/growthrates

RemoteRef 0.2.0

RemoteSha 334d6f64505eb3ee7382489327a311f1302de3a9

Contents

`as.Date.time_period` *Convert time period to dates*

Description

Convert time period to dates

Usage

```
## S3 method for class 'time_period'
as.Date(x, ...)

## S3 method for class 'time_period'
as.POSIXct(x, ...)
```

Arguments

`x`	a time_period
`...`	not used

Value

a vector of dates representing the start of each of the input `time_period` entries

Functions

- `as.POSIXct(time_period)`: Convert to a vector of POSIXct

as.time_period *Convert to a time period class*

Description

Time periods are just a zero based numeric representation of dates with a time unit baked in. This allows variable length periods (e.g. days or weeks), and fractional days to be represented in a consistent(ish) way

Usage

```
as.time_period(x, unit = NULL, start_date = NULL, anchor = NULL, ...)

## S3 method for class 'time_period'
c(..., recursive = F)

## S3 method for class 'time_period'
x[...]

## S3 replacement method for class 'time_period'
x[...] <- value

## S3 method for class 'time_period'
x[[...]]

## S3 replacement method for class 'time_period'
x[[...]] <- value

is.time_period(x)

## S3 method for class 'time_period'
print(x, ...)
```

Arguments

x	a vector of numbers (may be integer or real) or a time_period
unit	the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another `time_period`. If `x` is a `time_period`, and the unit is different then from that of `x` this will return a new `time_period` using the new units.
start_date	the zero time date as something that can be coerced to a date. If the `x` input is already a `time_period` and this is different to its `start_date` then it will be recalibrated to use the new start date.
anchor	only relevant is `x` is a vector of dates and `start_date` is not specified, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in `x` it will find a reference date for the time-series. If this is `NULL` and `start_date` is also `NULL` it will fall back to `getOption("day_zero","2019-12-29")`

`...`	used for subtype implementations
`recursive`	concatenate recursively
`value`	the value

Value

a `time_period` class, consisting of a vector of numbers, with attributes for time period and `start_date`

Functions

- `c(time_period)`: Combine `time_period`
- `[`: Subset a `time_period`
- `` `[`(time_period) <- value ``: Assign values to a subset of a `time_period`
- `[[`: Get a value in a `time_period`
- `` `[[`(time_period) <- value ``: Assign a value in a `time_period`
- `is.time_period()`: Check is a `time_period`
- `print(time_period)`: Print a time_period

Examples

```
# 100 weeks from 2020-01-01

tmp = as.time_period(0:100, 7, "2020-01-01")
as.Date(tmp)

range(tmp)
min(tmp)
tmp2 = as.integer(as.Date(tmp))
# testthat::expect_true(all(na.omit(tmp2-lag(tmp2)) == 7))

tmp2 = as.time_period(0:23, 1/24, "2020-01-01")
as.POSIXct(tmp2)

# convert timeseries to new "unit"
tmp = as.time_period(0:100, 7, "2020-01-01")
tmp2 = as.time_period(tmp,1)
testthat::expect_equal(as.numeric(tmp2), 0:100*7)
```

`breaks_log1p` *A scales breaks generator for log1p scales*

Description

A scales breaks generator for log1p scales

Usage

```
breaks_log1p(n = 5, base = 10)
```

Arguments

`n`	the number of breaks
`base`	the base for the breaks

Value

a function for ggplot scale breaks

Examples

```
ggplot2::ggplot(ggplot2::diamonds, ggplot2::aes(x=price))+
  ggplot2::geom_density()+
  ggplot2::scale_x_continuous(trans="log1p", breaks=breaks_log1p())
```

`covid_infectivity_profile` *The covid_infectivity_profile dataframe structure specification*

Description

The covid_infectivity_profile dataframe structure specification

Format

A dataframe containing the following columns:

- boot (anything) - a bootstrap identifier
- time (positive_double) - the end of the time period (in days)
- probability (proportion) - the probability of infection between previous time period until `time`

Must be grouped by: boot (exactly).

A default value is defined.

`cut_date` *Places a set of dates within a regular time series*

Description

The counterpart to date_seq_dates(). Take an original set of data and place it within a regular time series where the periodicity of the time series may be expressed as numbers of days, weeks, months quarters, or years, and the periods are defined by an anchoring date, day of the week or by reference to the start or end of the input dates. This can either return the periods as dates or factors (e.g. for plotting) or as a `time_period` for analysis that relies on a numeric representation of the date or duration from the anchor.

Usage

```
cut_date(
  dates,
  unit,
  anchor = "start",
  output = c("date", "factor", "time_period"),
  dfmt = "%d/%b/%y",
  ifmt = "{start} - {end}",
  ...
)
```

Arguments

`dates`	a set of dates
`unit`	a period e.g. "1 week"
`anchor`	one of a date, "start" or "end" or a weekday name e.g. "mon" this will always be one of the start of the time periods we are cutting into
`output`	return the result as either a "date" (the default), an ordered "factor" with the date ranges as a label, or as a "time_period". The result is named with labels referring to the
`dfmt`	the `strptime` format for the dates in the labels
`ifmt`	a `sprintf` format for the period label containing `%s` exactly twice.
`...`	ignored

Value

a set of dates, times or a factor level, representing the start of the period the date falls into, where the period is defined by the duration and the anchor

Examples

```
dates = as.Date(c("2020-01-01","2020-02-01","2020-01-15","2020-02-03",NA))
fs = growthrates::date_seq(dates, "2 days")
dates - cut_date(dates, "2 days")
cut_date(dates,unit="2 days", output="time_period")

# A weekly set of dates:
dates2 = Sys.Date() + floor(stats::runif(50,max=10))*7

# in this specific situation the final date is not truncated because the
# input data is seen as an exact match for the whole output period.
cut_date(dates2, "1 week", "sun", output="factor")
cut_date(dates2, dfmt = "%d/%b", output="factor", unit = "2 weeks", anchor="sun")
```

`date_seq` *Create the full sequence of values in a vector*

Description

This is useful if you want to fill in missing values that should have been observed but weren't. For example, date_seq(c(1, 2, 4, 6), 1) will return 1:6.

Usage

```
date_seq(x, period, ...)
```

Arguments

`x`	a numeric or date vector
`period`	Gap between each observation. The existing data will be checked to ensure that it is actually of this periodicity.
`...`	for subtype methods

Value

a vector of the same type as the input

Examples

```
date_seq(c(1, 2, 4, 5, 10), 1)
```

`date_seq.Date` *Expand a date vector to the full range of possible dates*

Description

Derive from a vector of observation dates, a complete ordered sequence of periods in a regular time series, where the length of the periods is specified, as a number od days, weeks, years etc. E.g. this can convert a random set of dates to a ordered complete list of 1 week intervals (or 2 month intervals) spanning the same range as the dates. This has some interesting problems regarding where to put breaks within a month or week. Often this is either based on a specific date (e.g. yearly periods starting at 2020-01-01) or a day of week (e.g. 2 weekly periods staring on a sunday) or maybe relative to the input time series (weekly ending on the last date of the data). There is also a problem when we consider data that may have incomplete starting and end periods, which may not be comparable to other periods, and we may need to exclude these from the result.

Usage

```
## S3 method for class 'Date'
date_seq(x, period = .day_interval(x), anchor = "start", complete = FALSE, ...)
```

Arguments

`x`	a vector of dates, possibly including NA values
`period`	the gap between observations as a number of days or as a natural language definition of the period such as "1 week", '2 weeks', '1 month', etc. If not given this will be derived from the dates.
`anchor`	defines a day that appears in the sequence (if it were to extend that far). Given either as a date, or "start", "end" or a day of the week, e.g. "mon".
`complete`	truncate incomplete start and end periods
`...`	ignored

Value

a vector of dates for regular periods between the minimum and maximum of dates, with the boundaries defined by the anchor.

Examples

```
date_seq(as.Date(c("2020-01-01","2020-02-01","2020-01-15","2020-02-01",NA)), "2 days")
```

`date_seq.numeric` *Create the full sequence of values in a vector*

Description

This is useful if you want to fill in missing values that should have been observed but weren't. For example, date_seq(c(1, 2, 4, 6), 1) will return 1:6.

Usage

```
## S3 method for class 'numeric'
date_seq(x, period = 1, tol = 1e-06, ...)
```

Arguments

`x`	a numeric or date vector
`period`	Gap between each observation. The existing data will be checked to ensure that it is actually of this periodicity.
`tol`	Numerical tolerance for checking periodicity.
`...`	for subtype methods

Value

a vector of the same type as the input

Examples

```
date_seq(c(1, 2, 4, 5, 10), 1)
```

`date_seq.time_period` *Expand a* `time_period` *vector to the full range of possible times*

Description

Derive from a vector of observation `time_periods`, a complete ordered sequence of periods in a regular time series, where the length of the periods is specified, as a number of days, weeks, years etc. E.g. this can convert a random set of times to a ordered complete list of 1 week intervals (or 2 month intervals) spanning the same range as the dates. This has some interesting problems regarding where to put breaks within a month or week. Often this is either based on a specific date (e.g. yearly periods starting at 2020-01-01) or a day of week (e.g. 2 weekly periods staring on a sunday) or maybe relative to the input time series (weekly ending on the last date of the data). There is also a problem when we consider data that may have incomplete starting and end periods, which may not be comparable to other periods, and we may need to exclude these from the result.

Usage

```
## S3 method for class 'time_period'
date_seq(x, period = attributes(x)$unit, complete = FALSE, ...)
```

Arguments

`x`	a time period vector
`period`	the gap between observations as a number of days or as a natural language definition of the period such as "1 week", '2 weeks', '1 month', etc. If not given this will be derived from the dates.
`complete`	truncate incomplete start and end periods
`...`	ignored

Value

a vector of `time_periods` for regular periods between the minimum and maximum of dates, with the boundaries defined by the anchor.

Examples

```
tmp = as.time_period(c(0,10,100), 7, "2020-01-01")
date_seq(tmp, "7 days")
```

`date_to_time` *Convert a set of dates to numeric timepoints*

Description

Using a start_date and a unit specification

Usage

```
date_to_time(
  dates,
  unit = .day_interval(dates),
  start_date = getOption("day_zero", "2019-12-29")
)
```

Arguments

`dates`	a vector of dates to convert
`unit`	a specification of the unit of the resulting time series. Will be determined from periodicity of dates if not specified. If another `time_period` is given as the unit then the
`start_date`	the origin of the conversion. Defaults to the beginning of the COVID pandemic

Value

a vector of class `time_period`

Examples

```
times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
dates = time_to_date(times)
```

`doubling_time` *Doubling time from growth rate*

Description

The unit of doubling times is always days.

Usage

```
doubling_time(x, ...)
```

Arguments

`x` a dataframe calculated from either proportion or incidence growth rate calculations:

e.g. A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

No mandatory groupings.

No default value.

OR

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

No mandatory groupings.

No default value.

`...` not used

Value

the same dataframe with additional columns for doubling time or relative doubling time plus confidence intervals.

Examples

```
growthrates::england_covid %>%
  growthrates::poisson_locfit_model(window=21) %>%
  growthrates::doubling_time() %>%
  dplyr::glimpse()
```

`england_consensus_growth_rate`

The SPI-M-O England consensus growth rate

Description

SPI-M-O used a range of different statistical and mechanistic models to produce estimates of the growth rate of the epidemic from various data sources (including with an early version of `growthrates`).

Usage

```
data(england_consensus_growth_rate)
```

Format

A dataframe containing the following columns:

- date (date) - the date of the estimate
- low (numeric) - the lower published estimate of the growth rate
- high (numeric) - the higher published estimate of the growth rate

No mandatory groupings.

No default value.

111 rows and 3 columns

`england_consensus_rt` *The SPI-M-O England consensus reproduction number*

Description

SPI-M-O used a range of different statistical and mechanistic models to produce estimates of the reproduction number of the epidemic from various data sources.

Usage

```
data(england_consensus_rt)
```

Format

A dataframe containing the following columns:

- date (date) - the date of the estimate
- low (numeric) - the lower published estimate of the reproduction number
- high (numeric) - the higher published estimate of the reproduction number

No mandatory groupings.

No default value.

113 rows and 3 columns

`england_covid` *Daily COVID-19 case counts by age group in England*

Description

A dataset of the daily count of covid cases by age group in England downloaded from the UKHSA coronavirus API, and formatted for use in `growthrates`. A denominator is calculated which is the overall positive count for all age groups. This data set can be used to calculate group-wise incidence and absolute growth rates and group wise proportions and relative growth rates.

Usage

```
data(england_covid)
```

Format

A dataframe containing the following columns:

- date (as.Date) - the date column
- class (enum(`00_04,05_09,10_14,15_19,20_24,25_29,30_34,35_39,40_44,45_49,50_54,55_59,60_64,65_69,70_74,75`
  - the class column
- count (numeric) - the test positives for each age group
- denom (numeric) - the test positives for all age groups
- time (as.time_period) - the time column

Must be grouped by: class (and other groupings allowed).

No default value.

26790 rows and 5 columns

`england_covid_pcr_positivity` *England COVID-19 PCR test positivity*

Description

The `coronavirus.gov.uk` dashboard published tests conducted and positive results as separate data sets for a range of geographies. In this case the data is combined with testing rate as denominator, and positives as count for England.

Usage

```
data(england_covid_pcr_positivity)
```

Format

A dataframe containing the following columns:

- date (date) - a daily time series
- time (as.time_period) - the time column
- count (numeric) - test positives in England on that day
- denom (numeric) - total tests conducted on that day

No mandatory groupings.

No default value.

1413 rows and 4 columns

`england_covid_proportion`
England COVID by age group for ascertainment

Description

An age group stratified dataset from

Usage

```
data(england_covid_proportion)
```

Format

A dataframe containing the following columns:

- class (character) - the age group
- date (date) - the start date of a week
- count (numeric) - the count of COVID positives
- denom (numeric) - the number of COVID tests performed
- population (numeric) - the size of the population at this age group
- time (as.time_period) - the time column (weekly)

Must be grouped by: class (and other groupings allowed).

No default value.

1050 rows and 6 columns

Details

- the coronavirus.gov.uk site for positive cases aggregated to 10 year age groups and by weekly time.
- NHS test and trace date which reported regional by age group testing effort aggregated to country level.
- ONS 2021 census population aggregated to 10 year age groups.

`england_demographics` *England demographics*

Description

Population counts by 5 year age group for England only from the 2021 census.

Usage

```
data(england_demographics)
```

Format

A dataframe containing the following columns:

- class (enum(`00_04,05_09,10_14,15_19,20_24,25_29,30_34,35_39,40_44,45_49,50_54,55_59,60_64,65_69,70_74,75_`
  - the class column
- population (numeric) - the population count column
- baseline_proportion (numeric) - the baseline proportion is the proportion this age group makes up of the total.

Must be grouped by: class (and other groupings allowed).

No default value.

19 rows and 3 columns

Source

https://www.ons.gov.uk/file?uri=/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/pop

`england_events` *Key dated in the COVID-19 response in England*

Description

This includes mainly the dates of lockdowns, releases from social distancing measures and the dates that new variants were first detected.

Usage

```
data(england_events)
```

Format

A dataframe containing the following columns:

- label (character) - the event label
- start (date) - the event start date
- end (date) - the (optional) event end date

No mandatory groupings.

No default value.

13 rows and 3 columns

`england_nhs_app` *NHS COVID-19 app data*

Description

check-in (social activity) and alerts (self isolation instruction) data from the NHS COVID-19 app, aggregated to country level on a week by week basis.

Usage

```
data(england_nhs_app)
```

Format

A dataframe containing the following columns:

- date (date) - the start date of the week
- alerts (integer) - the count of self-isolation alerts
- visits (integer) - the number of venue check-ins representing visits to social venues.
- time (as.time_period) - the time column

No mandatory groupings.

No default value.

137 rows and 4 columns

`england_ons_infection_survey` *The england_ons_infection_survey dataset*

Description

The COVID-19 ONS infection survey took a random sample of the population and provides an estimate of the prevalence of COVID-19 that is supposedly free from ascertainment bias.

Usage

```
data(england_ons_infection_survey)
```

Format

A dataframe containing the following columns:

- date (date) - the date column
- geography (character) - the geography column
- proportion.0.5 (numeric) - the median proportion of people in the region testing positive for COVID-19
- proportion.0.025 (numeric) - the lower CI of the proportion of people in the region testing positive for COVID-19
- proportion.0.975 (numeric) - the upper CI of the proportion of people in the region testing positive for COVID-19
- denom (integer) - the sample size on which this estimate was made (daily rate inferred from weekly sample sizes.)
- time (as.time_period) - the time column

No mandatory groupings.

No default value.

9820 rows and 7 columns

Details

The data is available here: https://www.ons.gov.uk/file?uri=/peoplepopulationandcommunity/healthandsocialcare/conditions

`england_variants` *Counts of COVID-19 variants*

Description

Data from the COG-UK and Sanger centre sequencing programme. The data were made available through the Welcome foundation at Lower tier local authority level, and is weekly timeseries of counts per variant. Variants were assigned using the tree structure of the Pango lineage. Different sub-lineages are aggregated to the major WHO variants of concern.

Usage

```
data(england_variants)
```

Format

A dataframe containing the following columns:

- date (date) - the end date of the week
- time (as.time_period) - the time column
- class (enum(`Other`,`Alpha (B.1.1.7)`,`Delta (B.1.617.2)`,`Delta (AY.4)`,`Omicron (Other)`,`Omicron (BA.2)`,`Omicron (BA.4)`,`Omicron (BA.5)`,`XBB (Other)`,`Kraken (XBB.1.5)`,`Arcturus (XBB.1.16)`,`Eris (EG.5.1)`)) - the class column
- who_class (enum(`Other`,`Alpha`,`Delta`,`Omicron`,`Kraken`,`Arcturus`,`Eris`)) - the who_class column
- count (numeric) - the weekly count column
- denom (numeric) - the number of sequences performed in that week

Must be grouped by: class (and other groupings allowed).

No default value.

479 rows and 6 columns

`fdmy` *Format date as dmy*

Description

Format date as dmy

Usage

```
fdmy(date)
```

Arguments

`date`	a date to convert

Value

the formatted date

Examples

```
fdmy(Sys.Date())
```

`geom_events` *Add time series event markers to a timeseries plot.*

Description

The x axis must be a date.

Usage

```
geom_events(
  events = i_events,
  event_label_size = 7,
  event_label_colour = "black",
  event_label_angle = -30,
  event_line_colour = "grey50",
  event_fill_colour = "grey50",
  hide_labels = FALSE,
  guide_axis = ggplot2::derive(),
  ...
)
```

Arguments

`events` Significant events or time spans

A dataframe containing the following columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

No mandatory groupings.

A default value is defined.

`event_label_size`
how big to make the event label

`event_label_colour`
the event label colour

`event_label_angle`
: the event label colour

`event_line_colour`
: the event line colour

`event_fill_colour`
: the event area fill

`hide_labels`
: do not show labels at all

`guide_axis`
: a guide axis configuration for the labels (see ggplot2::guide_axis and ggplot2::dup_axis). This can be used to specify a position amongst other things.

`...`
: Arguments passed on to `ggplot2::scale_x_date`

`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.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the breaks specified by `date_breaks`
- A `Date`/`POSIXct` vector giving positions of breaks
- A function that takes the limits as input and returns breaks as output

`date_breaks` A string giving the distance between breaks like "2 weeks", or "10 years". If both `breaks` and `date_breaks` are specified, `date_breaks` wins. Valid specifications are 'sec', 'min', 'hour', 'day', 'week', 'month' or 'year', optionally followed by 's'.

`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.

`date_labels` A string giving the formatting specification for the labels. Codes are defined in `strftime()`. If both `labels` and `date_labels` are specified, `date_labels` wins.

`minor_breaks` One of:

- `NULL` for no breaks
- `waiver()` for the breaks specified by `date_minor_breaks`
- A `Date`/`POSIXct` vector giving positions of minor breaks
- A function that takes the limits as input and returns minor breaks as output

`date_minor_breaks` A string giving the distance between minor breaks like "2 weeks", or "10 years". If both `minor_breaks` and `date_minor_breaks` are specified, `date_minor_breaks` wins. Valid specifications are 'sec', 'min', 'hour', 'day', 'week', 'month' or 'year', optionally followed by 's'.

`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()`).

`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.

`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.

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

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

Value

a set of geoms for a timeseries.

`germany_covid` *Weekly COVID-19 case counts by age group in Germany*

Description

A dataset of the weekly count of covid cases by age group in Germany downloaded from the Robert Koch Institute Survstat service, and formatted for use in growth rates. A denominator is calculated which is the overall positive count for all age groups. This data set can be used to calculate group-wise incidence and absolute growth rates and group wise proportions and relative growth rates.

Usage

```
data(germany_covid)
```

Format

A dataframe containing the following columns:

- class (enum(`0-14`,`15-19`,`20-24`,`25-29`,`30-39`,`40-49`,`50-59`,`60-69`,`70-79`,`80+`,`Unknown`, .ordered=TRUE)) - the age group
- date (as.Date) - the date column
- count (integer) - the test positives for each age group
- time (as.time_period) - the time column
- denom (integer) - the test positives for all age groups

Must be grouped by: class (and other groupings allowed).

No default value.

2070 rows and 6 columns

`germany_demographics` *Germany demographics*

Description

Derived from the Robert Koch Survstat service by comparing counts and incidence rates.

Usage

```
data(germany_demographics)
```

Format

A dataframe containing the following columns:

- class (enum(`0-14`,`15-19`,`20-24`,`25-29`,`30-39`,`40-49`,`50-59`,`60-69`,`70-79`,`80+`, .ordered=TRUE)) - the class column
- population (integer) - the population column

Must be grouped by: class (and other groupings allowed).

No default value.

10 rows and 2 columns

is.Date — *Check whether vector is a date*

Description

Check whether vector is a date

Usage

```
is.Date(x)
```

Arguments

`x`	a vector to check

Value

TRUE if dates, FALSE otherwise

Examples

```
is.Date(Sys.Date())
```

labels.time_period — *Label a time period*

Description

Create a set of labels for a time period based on the start and duration of the period. The format is configurable using the start and end dates and the `dfmt` and `ifmt` parameters, however if the time period has names then these are used in preference.

Usage

```
## S3 method for class 'time_period'
labels(
  object,
  ...,
  dfmt = "%d/%b",
  ifmt = "{start} - {end}",
  na.value = "Unknown"
)
```

Arguments

`object`	a set of decimal times as a time_period
`...`	not used
`dfmt`	a `strptime` format specification for the format of the date
`ifmt`	a glue spec referring to `start` and `end` of the period as a formatted date
`na.value`	a label for `NA` times

Value

a set of character labels for the time

Examples

```
eg = as.time_period(Sys.Date()+0:10*7, anchor="start")
labels(eg)
labels(eg, ifmt="{start}", dfmt="%d/%b/%y")
labels(eg, ifmt="until {end}", dfmt="%d %b %Y")

# labels retained in constructor:
eg2 = Sys.Date()+0:10*7
names(eg2) = paste0("week ",0:10)
labels(eg2)
labels(as.time_period(eg2, anchor="start"))
```

`logit_trans` *logit scale*

Description

Perform logit scaling with correct axis formatting. To not be used directly but with ggplot (e.g. ggplot2::scale_y_continuous(trans = "logit"))

Usage

```
logit_trans(n = 5, ...)
```

Arguments

`n`	the number of breaks
`...`	not used, for compatibility

Value

A scales object

Examples

```
library(ggplot2)
library(tibble)

tibble::tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
 ggplot2::ggplot(aes(fold_change , pvalue)) +
 ggplot2::geom_point() +
 ggplot2::scale_y_continuous(trans = "logit")
```

`max_date` *The maximum of a set of dates*

Description

`max.Date` returns an integer and `-Inf` for a set of `NA` dates. This is usually inconvenient.

Usage

```
max_date(x, ...)
```

Arguments

`x`	a vector of dates
`...`	ignored

Value

a date. ‘0001-01-01“ if there is no well defined minimum.

Examples

```
max_date(NA)
```

`min_date` *The minimum of a set of dates*

Description

`min.Date` returns an integer and `Inf` for a set of `NA` dates. This is usually inconvenient.

Usage

```
min_date(x, ...)
```

Arguments

`x`	a vector of dates
`...`	ignored

Value

a date. `9999-12-31` if there is no well defined minimum.

Examples

```
min_date(NA)
```

`multinomial_nnet_model` *Multinomial time-series model.*

Description

Takes a list of times, classes and counts, e.g. a COGUK variant like data set with time, (multinomial) class (e.g. variant) and count being the count in that time period. Fits a quadratic B-spline on time to the proportion of the data using `nnet::multinom`, with approx one degree of freedom per class and per `window` units of the time series

Usage

```
multinomial_nnet_model(
  d = i_multinomial_input,
  ...,
  window = 14,
  frequency = "1 day",
  predict = TRUE
)
```

Arguments

`d`	Multiclass count input
`...`	not used and present to allow proportion model to be used in a group_modify
`window`	a number of data points between knots, smaller values result in less smoothing, large value in more.
`frequency`	the density of the output estimates.
`predict`	result a prediction. If false we return the model.

Value

a new dataframe with `time` (as a time period), `class`, and `proportion.0.5`, or a model object

Examples

```
if (FALSE) {
  # not run due to long running
  tmp = growthrates::england_covid %>%
    dplyr::filter(date > "2022-01-01") %>%
    growthrates::multinomial_nnet_model(window=21) %>%
    dplyr::glimpse()
}
```

`normalise_incidence` *Calculate a normalised incidence rate per capita*

Description

This assumes positive disease counts are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population size allows us to compare groups.

Usage

```
normalise_incidence(
  modelled = i_timeseries,
  ...,
  population_unit = 1e+05,
  normalise_time = FALSE
)
```

Arguments

`modelled`	Model output from processing the `raw` dataframe with something like `poission_locfit_model` A dataframe containing the following columns: • time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' No mandatory groupings. No default value.
`...`	not used
`population_unit`	what population unit do you want the incidence in e.g. per 100K
`normalise_time`	The default behaviour for incidence is to keep it in the same time units as the input data. If this parameter is set to `TRUE` the incidence rates are calculated per year. If given as a lubridate period string e.g. "1 day" then the incidence is calculated over that time period.

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.per_capita.fit (double) - an estimate of the incidence per capita rate on a log scale
- incidence.per_capita.se.fit (double) - the standard error of the incidence per capita rate estimate on a log scale
- incidence.per_capita.0.025 (positive_double) - lower confidence limit of the incidence per capita rate (true scale)
- incidence.per_capita.0.5 (positive_double) - median estimate of the incidence per capita rate (true scale)
- incidence.per_capita.0.975 (positive_double) - upper confidence limit of the incidence per capita rate (true scale)
- population_unit (double) - The population unit on which the per capita incidence rate is calculated

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::poisson_locfit_model(window=21) %>%
  growthrates::normalise_incidence(growthrates::england_demographics) %>%
  dplyr::glimpse()
```

normalise_incidence.incidence

Calculate a normalised incidence rate per capita

Description

This assumes positive disease counts are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population size allows us to compare groups.

Usage

```
normalise_incidence.incidence(
  modelled = i_incidence_model,
  pop = i_population_data,
  ...,
  population_unit = 1e+05,
  normalise_time = FALSE
)
```

Arguments

`modelled`	Model output from processing the `raw` dataframe with something like `poission_locfit_model` A dataframe containing the following columns: • time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘ • incidence.fit (double) - an estimate of the incidence rate on a log scale • incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale • incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) • incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale) • incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale) No mandatory groupings. No default value.
`pop`	The population data must be grouped in the same way as `modelled`. A dataframe containing the following columns: • population (positive_integer) - Size of population No mandatory groupings. No default value.
`...`	not used
`population_unit`	what population unit do you want the incidence in e.g. per 100K
`normalise_time`	The default behaviour for incidence is to keep it in the same time units as the input data. If this parameter is set to `TRUE` the incidence rates are calculated per year. If given as a lubridate period string e.g. "1 day" then the incidence is calculated over that time period.

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.per_capita.fit (double) - an estimate of the incidence per capita rate on a log scale
- incidence.per_capita.se.fit (double) - the standard error of the incidence per capita rate estimate on a log scale
- incidence.per_capita.0.025 (positive_double) - lower confidence limit of the incidence per capita rate (true scale)
- incidence.per_capita.0.5 (positive_double) - median estimate of the incidence per capita rate (true scale)
- incidence.per_capita.0.975 (positive_double) - upper confidence limit of the incidence per capita rate (true scale)

- population_unit (double) - The population unit on which the per capita incidence rate is calculated

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::poisson_locfit_model(window=21) %>%
  growthrates::normalise_incidence(growthrates::england_demographics) %>%
  dplyr::glimpse()
```

---

normalise_incidence.proportion

Calculate a normalised incidence rate per capita

Description

This assumes positive disease counts are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population size allows us to compare groups.

Usage

```
normalise_incidence.proportion(
  modelled = i_proportion_model,
  ...,
  population_unit = 1e+05,
  normalise_time = FALSE
)
```

Arguments

`modelled` Model output from processing the `raw` dataframe with something like `poission_locfit_model`

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)

No mandatory groupings.

No default value.

`...`	not used
`population_unit`	what population unit do you want the incidence in e.g. per 100K
`normalise_time`	The default behaviour for incidence is to keep it in the same time units as the input data. If this parameter is set to `TRUE` the incidence rates are calculated per year. If given as a lubridate period string e.g. "1 day" then the incidence is calculated over that time period.

Details

This scales a proportion model by the population unit to make it comparable to an incidence model.

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.per_capita.fit (double) - an estimate of the incidence per capita rate on a log scale
- incidence.per_capita.se.fit (double) - the standard error of the incidence per capita rate estimate on a log scale
- incidence.per_capita.0.025 (positive_double) - lower confidence limit of the incidence per capita rate (true scale)
- incidence.per_capita.0.5 (positive_double) - median estimate of the incidence per capita rate (true scale)
- incidence.per_capita.0.975 (positive_double) - upper confidence limit of the incidence per capita rate (true scale)
- population_unit (double) - The population unit on which the per capita incidence rate is calculated

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::poisson_locfit_model(window=21) %>%
  growthrates::normalise_incidence(growthrates::england_demographics) %>%
  dplyr::glimpse()
```

`normalise_proportion` *Calculate a normalised risk ration from proportions*

Description

This assumes case distribution proportions are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population proportion allows us to compare groups.

Usage

```
normalise_proportion(
  modelled = i_proportion_model,
  base = i_baseline_proportion_data,
  ...
)
```

Arguments

`modelled` Model output from processing the `raw` dataframe with something like `proportion_locfit_model`

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)

No mandatory groupings.

No default value.

`base` The baseline data must be grouped in the same way as `modelled`.

A dataframe containing the following columns:

- baseline_proportion (proportion) - Size of population

No mandatory groupings.

No default value.

`...` not used

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- risk_ratio.0.025 (positive_double) - lower confidence limit of the excess risk ratio for a population group
- risk_ratio.0.5 (positive_double) - median estimate of the excess risk ratio for a population group
- risk_ratio.0.975 (positive_double) - upper confidence limit of the excess risk ratio for a population group
- baseline_proportion (proportion) - The population baseline risk from which the excess risk ratio is based

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::proportion_locfit_model(window=21) %>%
  growthrates::normalise_proportion(growthrates::england_demographics) %>%
  dplyr::glimpse()

plot_growth_phase(tmp)
```

`plot_growth_phase` *Plot an incidence or proportion vs. growth phase diagram*

Description

Plot an incidence or proportion vs. growth phase diagram

Usage

```
plot_growth_phase(
  modelled = i_timestamped,
  timepoints = NULL,
  duration = max(dplyr::count(modelled)$n),
  interval = 7,
 mapping = if (interfacer::is_col_present(modelled, class)) ggplot2::aes(colour = class)
    else ggplot2::aes(),
  cis = TRUE,
  ...
)
```

Arguments

`modelled` Either:

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

No mandatory groupings.

No default value.

OR:

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)

- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

No mandatory groupings.

No default value.

`timepoints`	timepoints (as `Date` or `time_period` vector) of dates to plot phase diagrams. If multiple this will result in a sequence of plots as facets. If `NULL` (the default) it will be the last time point in the series
`duration`	the length of the growth rate phase trail
`interval`	the length of time between markers on the phase plot
`mapping`	a `ggplot2::aes()` mapping
`cis`	should the phases be marked with confidence intervals?
`...`	Arguments passed on to `geom_events`

`events` Significant events or time spans

A dataframe containing the following columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

No mandatory groupings.

A default value is defined.

Value

a ggplot timeseries

Examples

```
# example code

tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))

tmp_pop = growthrates::england_demographics %>%
  dplyr::ungroup() %>%
  dplyr::summarise(population = sum(population))

# If the incidence is normalised by population
tmp2 = tmp %>%
  poisson_locfit_model() %>%
  normalise_incidence(tmp_pop)
```

```
timepoints = as.Date(c("Lockdown 1" = "2020-03-30", "Lockdown 2" = "2020-12-31"))

plot_growth_phase(tmp2, timepoints, duration=108)
```

plot_growth_rate *Growth rate timeseries diagram*

Description

Growth rate timeseries diagram

Usage

```
plot_growth_rate(
  modelled = i_timeseries,
  ...,
 mapping = if (interfacer::is_col_present(modelled, class)) ggplot2::aes(colour = class)
    else ggplot2::aes(),
  events = i_events
)
```

Arguments

`modelled` Either:

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

No mandatory groupings.

No default value.

OR:

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

No mandatory groupings.

No default value.

`...` Arguments passed on to `geom_events`

`mapping` a `ggplot2::aes` mapping. Most importantly setting the `colour` to something if there are multiple incidence time series in the plot

`events` Significant events or time spans

A dataframe containing the following columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

No mandatory groupings.

A default value is defined.

Value

a ggplot timeseries

Examples

```
# example code
tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))

tmp_pop = growthrates::england_demographics %>%
  dplyr::ungroup() %>%
  dplyr::summarise(population = sum(population))
```

```
# If the incidence is normalised by population
tmp2 = tmp %>%
  poisson_locfit_model() %>%
  normalise_incidence(tmp_pop)

# Default pdf device doesn't support unicode
plot_growth_rate(tmp2,colour="blue")

tmp3 = growthrates::england_covid %>%
  proportion_locfit_model()

# Default pdf device doesn't support unicode
plot_growth_rate(tmp3)
```

---

plot_incidence *Plot an incidence timeseries*

Description

Plot an incidence timeseries

Usage

```
plot_incidence(
  modelled = i_incidence_model,
  raw = i_incidence_data,
  ...,
  mapping = if (interfacer::is_col_present(modelled, class)) ggplot2::aes(colour = class)
    else ggplot2::aes(),
  events = i_events
)
```

Arguments

`modelled` An optional estimate of the incidence time series. If `modelled` is missing then it is estimated from `raw` using a `poisson_locfit_model`. In this case parameters `window` and `deg` may be supplied to control the fit.

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)

- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)

No mandatory groupings.

No default value.

`modelled` can also be the output from `normalise_incidence` in which case the plot uses the per capita rates calculated by that function

`raw` The raw count data

A dataframe containing the following columns:

- count (positive_integer) - Positive case counts associated with the specified timeframe
- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

No mandatory groupings.

No default value.

`...` Arguments passed on to `geom_events`, `poisson_locfit_model`

`window` a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)

`deg` polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - (defaults to `1`)

`frequency` the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)

`mapping` a `ggplot2::aes` mapping. Most importantly setting the `colour` to something if there are multiple incidence timeseries in the plot

`events` Significant events or time spans

A dataframe containing the following columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

No mandatory groupings.

A default value is defined.

Value

a ggplot object

Examples

```
# example code

tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))

tmp_pop = growthrates::england_demographics %>%
  dplyr::ungroup() %>%
  dplyr::summarise(population = sum(population))

# If the incidence is normalised by population
tmp2 = tmp %>%
  poisson_locfit_model() %>%
  normalise_incidence(tmp_pop)

plot_incidence(tmp2,tmp %>% dplyr::cross_join(tmp_pop),colour="blue",size=0.25)
```

`plot_multinomial` *Plot a multinomial proportions mode*

Description

Plot a multinomial proportions mode

Usage

```
plot_multinomial(
  modelled = i_multinomial_proportion_model,
  ...,
  mapping = ggplot2::aes(fill = class),
  events = i_events,
  normalise = FALSE
)
```

Arguments

`modelled` the multinomial count data

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- class (factor) - A factor specifying the type of observation. This will be things like variant, or serotype, for a multinomial model. Any missing data points are ignored.
- proportion.0.5 (proportion) - median estimate of proportion (true scale)

Must be grouped by: class (exactly).

No default value.

`...`	Arguments passed on to `geom_events`
`mapping`	a `ggplot2::aes` mapping. Most importantly setting the `colour` to something if there are multiple incidence timeseries in the plot
`events`	Significant events or time spans A dataframe containing the following columns: • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event No mandatory groupings. A default value is defined.
`normalise`	make sure the probabilities add up to one - this can be a bad idea if you know you may have missing values.

Value

a ggplot

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::proportion_locfit_model(window=21) %>%
  dplyr::glimpse()

plot_multinomial(tmp, normalise=TRUE)+
  ggplot2::scale_fill_viridis_d()
```

`plot_proportion` *Plot a proportions timeseries*

Description

Plot a proportions timeseries

Usage

```
plot_proportion(
  modelled = i_proportion_model,
  raw = i_proportion_data,
  ...,
  mapping = if (interfacer::is_col_present(modelled, class)) ggplot2::aes(colour = class)
    else ggplot2::aes(),
  events = i_events
)
```

Arguments

`modelled`
: Proportion model estimates

  A dataframe containing the following columns:

  - time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘
  - proportion.fit (double) - an estimate of the proportion on a logit scale
  - proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
  - proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
  - proportion.0.5 (proportion) - median estimate of proportion (true scale)
  - proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)

  No mandatory groupings.

  No default value.

`raw`
: Raw count data

  A dataframe containing the following columns:

  - denom (positive_integer) - Total test counts associated with the specified timeframe
  - count (positive_integer) - Positive case counts associated with the specified timeframe
  - time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘

  No mandatory groupings.

  No default value.

`...`
: Arguments passed on to `geom_events`, `proportion_locfit_model`

  `window` a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)

  `deg` polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - (defaults to `1`)

  `frequency` the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)

`mapping`
: a `ggplot2::aes` mapping. Most importantly setting the `colour` to something if there are multiple incidence timeseries in the plot

`events`
: Significant events or time spans

  A dataframe containing the following columns:

  - label (character) - the event label
  - start (date) - the start date, or the date of the event
  - end (date) - the end date or NA if a single event

  No mandatory groupings.

  A default value is defined.

Value

a ggplot object

Examples

```
tmp = growthrates::england_covid %>%
  growthrates::proportion_locfit_model(window=21) %>%
  dplyr::glimpse()

plot_proportion(tmp)+
  ggplot2::scale_fill_viridis_d(aesthetics = c("fill","colour"))
```

`plot_rt` *Reproduction number timeseries diagram*

Description

Reproduction number timeseries diagram

Usage

```
plot_rt(
  modelled = i_reproduction_number,
  ...,
  mapping = if (interfacer::is_col_present(modelled, class)) ggplot2::aes(colour = class)
    else ggplot2::aes(),
  events = i_events
)
```

Arguments

`modelled` the modelled Rt estimate

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

No mandatory groupings.

No default value.

`...` Arguments passed on to `geom_events`

`mapping`	a `ggplot2::aes` mapping. Most importantly setting the `colour` to something if there are multiple incidence time series in the plot
`events`	Significant events or time spans A dataframe containing the following columns: • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event No mandatory groupings. A default value is defined.

Value

a ggplot timeseries

Examples

```
# example code
tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))
if (FALSE) {

  tmp2 = tmp %>%
    poisson_locfit_model() %>%
    rt_from_growth_rate()

  # comparing RT from growth rates with England consensus Rt:
  plot_rt(tmp2,colour="blue")+
   geom_errorbar(data=england_consensus_rt, mapping=aes(x=date-21,ymin=low,ymax=high),colour="red")

}
```

`poisson_glm_model` *Poisson time-series model.*

Description

This uses a generalised linear model to fit a quasi-poisson model with a time varying rate as a natural cubic spline with approx one degree of freedom per `window` units of the time series.

Usage

```
poisson_glm_model(d = i_incidence_input, ..., window = 14, frequency = "1 day")
```

Arguments

`d`	Count model input A dataframe containing the following columns: • count (positive_integer) - Positive case counts associated with the specified timeframe • time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘ Ungrouped. No default value.
`...`	not used and present to allow proportion model to be used in a `group_modify`
`window`	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)
`frequency`	the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
 growthrates::poisson_glm_model(window=21) %>%
 dplyr::glimpse()
```

`poisson_locfit_model` *Poisson time-series model.*

Description

Takes a list of times and counts and fits a quasi-poisson model fitted with a log link function to count data using local regression using the package `locfit`.

Usage

```
poisson_locfit_model(
  d = i_incidence_input,
  ...,
  window = 14,
  deg = 1,
  frequency = "1 day",
  predict = TRUE
)
```

Arguments

`d`	input data A dataframe containing the following columns: • count (positive_integer) - Positive case counts associated with the specified timeframe • time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘ Ungrouped. No default value.
`...`	not used and present to allow proportion model to be used in a `group_modify`
`window`	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)
`deg`	polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - (defaults to `1`)
`frequency`	the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)
`predict`	result is a prediction dataframe. If false we return the `locfit` models (advanced). - (defaults to `TRUE`)

Details

This results is an incidence rate estimate plus an absolute exponential growth rate estimate both based on the time unit of the input data (e.g. for daily data the rate will be cases per day and the growth rate will be daily).

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

No mandatory groupings.

No default value.

Examples

```
growthrates::england_covid %>%
  growthrates::poisson_locfit_model(window=21) %>%
  dplyr::glimpse()
```

`proportion_glm_model` *Binomial time-series model.*

Description

This uses a generalised linear model to fit a quasi-binomial model with a time varying rate as a natural cubic spline with approx one degree of freedom per `window` units of the time series.

Usage

```
proportion_glm_model(
  d = i_proportion_input,
  ...,
  window = 14,
  frequency = "1 day"
)
```

Arguments

`d`	Proportion model input A dataframe containing the following columns: • denom (positive_integer) - Total test counts associated with the specified timeframe • count (positive_integer) - Positive case counts associated with the specified timeframe • time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘ Ungrouped. No default value.
`...`	not used and present to allow proportion model to be used in a `group_modify`
`window`	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)
`frequency`	the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)

No mandatory groupings.

No default value.

Examples

```
# TODO: find out cause of the warnings
# "observations with zero weight not used for calculating dispersion"
suppressWarnings(
  growthrates::england_covid %>%
   growthrates::proportion_glm_model(window=21) %>%
   dplyr::glimpse()
)
```

`proportion_locfit_model` *A binomial proportion estimate and associated exponential growth rate*

Description

takes a list of times, counts and a denominator and fits a quasi-binomial model using a logit link function to proportion data using local regression using the package `locfit`.

Usage

```
proportion_locfit_model(
  d = i_proportion_input,
  ...,
  window = 14,
  deg = 1,
  frequency = "1 day",
  predict = TRUE
)
```

Arguments

`d` the input

A dataframe containing the following columns:

- denom (positive_integer) - Total test counts associated with the specified timeframe
- count (positive_integer) - Positive case counts associated with the specified timeframe
- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘

Ungrouped.

No default value.

`...` not used and present to allow proportion model to be used in a `group_modify`

`window`	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - (defaults to `14`)
`deg`	polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - (defaults to `1`)
`frequency`	the density of the output estimates as a time period such as `7 days` or `2 weeks`. - (defaults to `"1 day"`)
`predict`	result is a prediction dataframe. If false we return the `locfit` models (advanced). - (defaults to `TRUE`)

Details

This expects d to contain one combination of:

- `time` and `count` and `denom` columns - e.g. all tests conducted.

This results is a one versus others comparison binomial proportion estimate plus a relative growth rate estimate specifying how much quicker this is growing compared to the growth of the denominator.

The denominator maybe the sum of all subgroups `denom = sum(count)`, e.g. in the situation where there are multiple variants of a disease circulating. In which case the relative growth is that of the subgroup compared to the overall. You can make this a one-versus-others comparison by making the denominator exclude the current item (e.g. `denom = sum(count)-count`).

The denominator can also be used to express the size of the population tested. This gives us a relative growth rate that is different in essence to the previous and may be a better estimate of the true growth rate in the situation where testing effort is variable, or capacity saturated.

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

No mandatory groupings.

No default value.

Examples

```
growthrates::england_covid %>%
 growthrates::proportion_locfit_model(window=21) %>%
 dplyr::glimpse()
```

`reband_discrete` *Reband any discrete distribution*

Description

e.g. age banded population, or a discrete probability distribution e.g. a serial interval distribution.

Usage

```
reband_discrete(
  x,
  y,
  xout,
  xlim = c(0, NA),
  ytotal = c(0, sum(y)),
  digits = 0,
  labelling = c("positive_integer", "inclusive", "exclusive"),
  sep = "-"
)
```

Arguments

`x`	a set of upper limits of bands, e.g. for age: 0-14;15-64;65-79;80+ is 15,65,80,NA
`y`	a set of quantities for each band e.g. population figures
`xout`	a set of new upper limits
`xlim`	Upper and lower limits for x. if the last band is e.g 80+ in the input and we want to know the 85+ band in the output some kind of maximum upper limit is needed to interpolate to.
`ytotal`	upper and lower limits for y. If the interpolation values fall outside of x then the in and max limits of y are given by this.
`digits`	if the `xout` value is continuous then how many significant figures to put in the labels
`labelling`	are the `xout` values interpretable as an `inclusive` upper limit, or an `exclusive` upper limit, or as an upper limit of an 'positive_integer" quantity
`sep`	seperator for names e.g. `18-24` or `18 to 24`

Value

a rebanded set of discrete values, guaranteed to sum to the same as `y`

Examples

```
ul = stringr::str_extract(england_demographics$class, "_([0-9]+)",group = 1) %>%
  as.numeric()

tmp = reband_discrete(
  ul, england_demographics$population,
  c(5,10,15,40,80), xlim=c(0,120))

tmp

sum(tmp)
sum(england_demographics$population)
```

`rt_epiestim` *EpiEstim reproduction number*

Description

Calculate a reproduction number estimate from incidence data using the EpiEstim library and an empirical generation time distribution. This uses resampling to transmit uncertainty in generation time estimates. This is quite slow for each time series depending on the number of bootstraps and samples in the infectivity profile.

Usage

```
rt_epiestim(
  df = i_incidence_input,
  ip = i_infectivity_profile,
  bootstraps = 2000,
  window = 14,
  mean_prior = 1,
  std_prior = 2,
  ...
)
```

Arguments

`df` Count data. Extra groups are allowed.

A dataframe containing the following columns:

- count (positive_integer) - Positive case counts associated with the specified timeframe
- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period‘

Ungrouped.

No default value.

`ip`	infectivity profile A dataframe containing the following columns: • boot (anything) - a bootstrap identifier • time (positive_double) - the end of the time period (in days) • probability (proportion) - the probability of infection between previous time period until ‘time‘ Must be grouped by: boot (exactly). A default value is defined.
`bootstraps`	• the number of bootstraps to take to calculate for each point.
`window`	• the width of the epiestim window
`mean_prior`	the prior for the $R_t$ estimate. When sample size is low the $R_t$ estimate will revert to this prior. In EpiEstim the default is a high number to allow detection of insufficient data but this tends to create anomalies in the early part of infection timeseries. A possible value is $R_0$ but in fact this also will be a poor choice for the value of $R_t$ when case numbers drop to a low value.
`std_prior`	the prior for the $R_t$ SD.
`...`	not used

Details

This will calculate a reproduction number for each group in the input dataframe.

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))

if (FALSE) {
  # not run due to long running
  tmp2 = tmp %>% rt_epiestim()
}
```

`rt_from_growth_rate` *Wallinga-Lipsitch reproduction number*

Description

Calculate a reproduction number estimate from growth rate using the Wallinga 2007 estimation using empirical generation time distribution. This uses resampling to transmit uncertainty in growth rate estimates

Usage

```
rt_from_growth_rate(
  df = i_growth_rate,
  ip = i_infectivity_profile,
  bootstraps = 2000
)
```

Arguments

`df` Growth rate estimates

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

No mandatory groupings.

No default value.

`ip` Infectivity profile

A dataframe containing the following columns:

- boot (anything) - a bootstrap identifier
- time (positive_double) - the end of the time period (in days)
- probability (proportion) - the probability of infection between previous time period until 'time'

Must be grouped by: boot (exactly).

A default value is defined.

`bootstraps`

- the number of bootstraps to take to calculate for each point.

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

No mandatory groupings.

No default value.

Examples

```
tmp = growthrates::england_covid %>%
  time_aggregate(count=sum(count))


if (FALSE) {
  # not run
  tmp2 = tmp %>%
    poisson_locfit_model() %>%
    rt_from_growth_rate()
}
```

`rt_from_incidence` *Reproduction number from modelled incidence*

Description

Calculate a reproduction number estimate from growth rate using the methods described in the vignette "Estimating the reproduction number from modelled incidence" and using an empirical generation time distribution.

Usage

```
rt_from_incidence(df = i_incidence_model, ip = i_infectivity_profile)
```

Arguments

`df` Count data

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)

No mandatory groupings.

No default value.

`ip` Infectivity profile

A dataframe containing the following columns:

- boot (anything) - a bootstrap identifier
- time (positive_double) - the end of the time period (in days)
- probability (proportion) - the probability of infection between previous time period until 'time'

Must be grouped by: boot (exactly).

A default value is defined.

Value

A dataframe containing the following columns:

- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

No mandatory groupings.

No default value.

Examples

```
df = growthrates::england_covid %>%
  time_aggregate(count=sum(count)) %>%
    poisson_locfit_model()


if (FALSE) {
  # not run
  tmp2 = df %>% rt_from_incidence()
}
```

`scale_y_log1p` *A log1p y scale*

Description

A log1p y scale

Usage

```
scale_y_log1p(..., n = 5, base = 10, dp = 0)
```

Arguments

`...`	Other arguments passed on to `scale_(x|y)_continuous()`
`n`	the number of major breaks
`base`	the base for the logarithm
`dp`	decimal points

Value

a ggplot scale

`scale_y_logit` *A logit y scale*

Description

A logit y scale

Usage

```
scale_y_logit(...)
```

Arguments

`...`	Other arguments passed on to `scale_(x|y)_continuous()`

Value

a ggplot scale

`time_aggregate` *Aggregate time series data preserving the time series*

Description

Aggregate time series data preserving the time series

Usage

```
time_aggregate(
  df = i_timestamped,
  ...,
  .groups = NULL,
  .cols = NULL,
  .fns = NULL
)
```

Arguments

`df`	an optionally grouped time series. Grouping should not include the time column. The grouping works differently from `dplyr::summarise` in that the last level of non-time groups is lost in this operation, so the subgroup you wish to aggregate should be included in the grouping.
`...`	A set of `dplyr::summarise` statements, or additional parameters for `.fns`
`.groups`	as per `dplyr::summarise`
`.cols`	Optional tidyselect column specification for `dplyr::across`. if `.fns` is given and the `.cols` parameter is not specified then the columns to summarise are automatically identified. In doing this any `Date` columns are dropped. If this in not what you want then `.cols` or `...` must be given
`.fns`	Optional a set of function specifications as per `dplyr::across`

Value

the summarised time series preserving the `time` column, and with the grouping structure involving one fewer levels that the input

Examples

```
growthrates::england_covid %>%
  time_aggregate(count = sum(count), denom = sum(denom)) %>%
  dplyr::glimpse()

growthrates::england_covid %>%
  time_aggregate(.fns=mean) %>%
  dplyr::glimpse()
```

`time_summarise` *Summarise data from a line list to a time-series of counts.*

Description

This principally is designed to take a record of single events and produce a summary time-series count of events by group, class and date. The default behaviour is to guess the cadence of the input data and summarise the event line list to a (set of) regular time-series counts for use in incidence and growth rate estimates.

Usage

```
time_summarise(
  df = i_dated,
  unit,
  anchor = "start",
  rectangular = FALSE,
  ...,
  .fill = list(count = 0)
)
```

Arguments

`df`	a line list of data you want to summarise, optionally grouped. If this is grouped then each group is treated independently. The remaining columns must contain a `date` column and may contain a `class` column. If a `count` column is present the counts will be summed, otherwise each individual row will be counted as a single event (as a linelist)
`unit`	a period e.g. "1 week"
`anchor`	one of a date, "start" or "end" or a weekday name e.g. "mon" this will always be one of the start of the time periods we are cutting into
`rectangular`	should the resulting time series be the same length for all groups. This is only the case if you can be sure that your data is complete for all subgroups, otherwise missing data will be treated as zero counts. This is important if leading and trailing missing data in one subgroup can be due to a reporting delay in that subgroup, in which case a rectangular time series will erroneously fill in zero counts for this missing data.

`...`	a spec for a dplyr::summary(...) - optional, and if not provided a `count = dplyr::n()` or a `count = sum(count)` is performed.
`.fill`	a list similar to tidyr::complete for values to fill variables with

Details

If the data is given with a `class` column the time series are interpreted as having a denominator, consisting of all the different classes within a time period. This may be subtypes (e.g. variants, serotypes) or markers for test positivity. In either case the resulting time series will have counts for all classes and denominators for the combination.

There is flexibility for other kinds of summarisation if the raw data is not count based (e.g. means of continuous variables) but in this case a the `slider` package is usually going to be better, as time summarise will only look at non overlapping time periods with fixed lengths.

There is another use case where an existing timeseries on a particular frequency is aggregated to another less frequent basis (e.g. moving from a daily timeseries to a weekly one). In this case the input will contain a `count` column. In this mode no checks are made that the more frequent events are all present before summarisation so the result may include different numbers of input periods (e.g. going from weeks to months may be 4 or 5 weeks in each month)

Value

The output depends on whether or not the input was grouped and had a `class` column. The most detailed output will be:

A dataframe containing the following columns:

- denom (positive_integer) - Total test counts associated with the specified timeframe
- count (positive_integer) - Positive case counts associated with the specified timeframe
- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`

No mandatory groupings.

No default value.

or a more minimal output if the input is only a plain list of dated events:

A dataframe containing the following columns:

- count (positive_integer) - Positive case counts associated with the specified timeframe
- time (as.time_period + group_unique) - A (usually complete) set of singular observations per unit time as a `time_period`

No mandatory groupings.

No default value.

time_to_date *Convert a set of timepoints to dates*

Description

Convert a set of timepoints to dates

Usage

```
time_to_date(
  timepoints,
  unit = attr(timepoints, "unit"),
  start_date = attr(timepoints, "start_date")
)
```

Arguments

`timepoints`	a set of numeric time points
`unit`	the period / unit of the time points, which will be extracted from timepoints if possible
`start_date`	the zero day of the time series, will be extracted from timepoints if possible

Value

a vector of dates

Examples

```
times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
dates = time_to_date(times)
```

wallinga_lipsitch *Calculate the reproduction number from a growth rate estimate and an infectivity profile*

Description

Calculate the reproduction number from a growth rate estimate and an infectivity profile

Usage

```
wallinga_lipsitch(r, y, a = 1:length(y))
```

Arguments

`r`	a growth rate (may be a vector)
`y`	an empirical infectivity profile as a probability vector, starting at `P(0<t,a[1])`
`a`	the end time of the estimate (defaults to single days).

Value

a reproduction number estimate based on `r`

Examples

```
wallinga_lipsitch(r=seq(-0.1,0.1,length.out=9), y=dgamma(1:50, 5,2))
```

Index