# Package 'baselinenowcast'

May 7, 2026

**Title** Baseline Nowcasting for Right-Truncated Epidemiological Data

**Version** 0.2.0

**Description** Nowcasting right-truncated epidemiological data is critical for timely public health decision-making, as reporting delays can create misleading impressions of declining trends in recent data. This package provides nowcasting methods based on using empirical delay distributions and uncertainty from past performance. It is also designed to be used as a baseline method for developers of new nowcasting methods. For more details on the performance of the method(s) in this package applied to case studies of COVID-19 and norovirus, see our recent paper at <https://wellcomeopenresearch.org/articles/10-614>. The package supports standard data frame inputs with reference date, report date, and count columns, as well as the direct use of reporting triangles, and is compatible with 'epinowcast' objects. Alongside an opinionated default workflow, it has a low-level pipe-friendly modular interface, allowing context-specific workflows. It can accommodate a wide spectrum of reporting schedules, including mixed patterns of reference and reporting (daily-weekly, weekly-daily). It also supports sharing delay distributions and uncertainty estimates between strata, as well as custom uncertainty models and delay estimation methods.

**License** MIT + file LICENSE

**URL** https://github.com/epinowcast/baselinenowcast, https://baselinenowcast.epinowcast.org

**BugReports** https://github.com/epinowcast/baselinenowcast/issues

**Depends** R (>= 4.1.0)

**Imports** cli, checkmate, stats, utils, rlang, purrr

**Suggests** bookdown, ChainLadder, dplyr, tidyr, stringr, lubridate, readr, ggplot2, spelling, rmarkdown, testthat (>= 3.1.9), usethis, withr, knitr, zoo, glue

**Encoding** UTF-8

**Language** en-GB

**LazyData** true

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Config/Needs/hexsticker** hexSticker, sysfonts, ggplot2

**Config/Needs/website** r-lib/pkgdown, epinowcast/enwtheme

**NeedsCompilation** no

**Author** Kaitlyn Johnson [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-8011-0012>),
Emily Tyszka [aut] (ORCID: <https://orcid.org/0009-0005-6088-4017>),
Johannes Bracher [aut] (ORCID: <https://orcid.org/0000-0002-3777-1410>),
Sebastian Funk [aut] (ORCID: <https://orcid.org/0000-0002-2842-3406>),
Sam Abbott [aut] (ORCID: <https://orcid.org/0000-0001-8057-8037>),
Tim Taylor [ctb] (ORCID: <https://orcid.org/0000-0002-8587-7113>)

**Maintainer** Kaitlyn Johnson <kaitlyn.johnson@lshtm.ac.uk>

**Repository** CRAN

**Date/Publication** 2026-02-03 11:10:07 UTC

# Contents

---

`allocate_reference_times`

*Allocate training volume based on combination of defaults and user-specified values for training volume for delay and uncertainty estimation.*

---

## Description

Given the reporting triangle and optionally the user-specified scale factor on the max delay to be used as total reference times and the proportion of those reference times to be used for delay estimation, allocate reference times to the number used for delay estimation and the number used as retrospective nowcasts for uncertainty estimation.

This function implements an algorithm which:

- if the specified number of reference times (`scale_factor` x `max delay`) is less than or equal to the number of reference times available in the reporting triangle, split reference times between delay and uncertainty according to `prop_delay`, ensuring that the minimum requirements for delay and uncertainty estimation are met.
- if the specified number of reference times is greater than the number of reference times available in the reporting triangle, use all the reference times available and satisfy the minimum requirement for delay estimation and then split the remainder according to the specified `prop_delay`, ensuring that the minimum reference times for delay and uncertainty estimation are fulfilled.
- the function errors if the minimum requirements for delay and uncertainty estimation are not possible from the number of reference times in the reporting triangle.

## Usage

```
allocate_reference_times(
  reporting_triangle,
  scale_factor = 3,
  prop_delay = 0.5,
  n_min_retro_nowcasts = 2,
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `scale_factor` | Numeric value indicating the multiplicative factor on the maximum delay to be used for estimation of delay and uncertainty. Default is `3`. |
| `prop_delay` | Numeric value <1 indicating what proportion of all reference times in the reporting triangle to be used for delay estimation. Default is `0.5`. |
| `n_min_retro_nowcasts` | Integer indicating the minimum number of reference times needed for uncertainty estimation. Default is `2`. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

list of n_history_delay and n_retrospective_nowcasts

## See Also

High-level workflow wrapper functions `estimate_and_apply_delay()`, `estimate_and_apply_delays()`, `estimate_and_apply_uncertainty()`, `estimate_uncertainty_retro()`

## Examples

```
# Create a reporting triangle from package data
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 25)

# Use the defaults (scale_factor = 3, prop_delay = 0.5)
ref_time_allocation_default <- allocate_reference_times(rep_tri)
ref_time_allocation_default

# Modify to use less volume and redistribute
ref_time_allocation_alt <- allocate_reference_times(
  reporting_triangle = rep_tri,
  scale_factor = 2,
  prop_delay = 0.6
)
ref_time_allocation_alt
```

---

# apply_delay *Apply the delay to generate a point nowcast*

---

## Description

Generate a point estimate of a completed reporting square (or rectangle) from a reporting triangle that we want to complete with a nowcast and a delay PMF. Each element is computed by taking the product of the expected number of total cases assigned to a reference time $t$ and the proportion of those cases reported on delay $d$. The formula to obtain the expected number of total cases as a function of the reporting delay and previous observations was derived elsewhere. This code was adapted from code written (under an MIT license) by the Karlsruhe Institute of Technology RESPINOW German Hospitalization Nowcasting Hub. Modified from: https://github.com/KITmetricslab/RESPINOW-Hub/blob/7cce3ae2728116e8c8cc0e4ab29074462c24650e/code/baseline/functions.R#L55 #nolint

## Usage

```
apply_delay(reporting_triangle, delay_pmf, validate = TRUE)
```

### Arguments

| | |
|---|---|
| reporting_triangle | Matrix of the reporting triangle to be nowcasted, with rows representing the time points of reference and columns representing the delays |
| delay_pmf | Vector of delays assumed to be indexed starting at the first delay column in reporting_triangle. |
| validate | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

### Value

point_nowcast_matrix Matrix of the same number of rows and columns as the rep_mat_to_nowcast but with the missing values filled in as point estimates

### Examples

```
# Example 1: Standard usage with example dataset
delay_pmf <- estimate_delay(example_reporting_triangle)
point_nowcast_matrix <- apply_delay(
  reporting_triangle = example_reporting_triangle,
  delay_pmf = delay_pmf
)
print(point_nowcast_matrix)

# Example 2: Using delay PMF with negative entries from downward corrections
delay_pmf_negative <- c(0.7, 0.4, -0.15, 0.05)
nowcast_with_corrections <- apply_delay(
  reporting_triangle = example_downward_corr_rt,
  delay_pmf = delay_pmf_negative
)
# The nowcast includes negative predictions at delay 2,
# correctly reflecting expected downward corrections
print(nowcast_with_corrections)
```

---

## apply_reporting_structure

*Apply reporting structure to generate a single retrospective reporting triangle*

---

### Description

This function applies a reporting structure to a truncated reporting triangle by setting observations to NA row by row from the bottom up based on the specified structure. It is the singular version of apply_reporting_structures().

## Usage

```
apply_reporting_structure(
  truncated_reporting_triangle,
  structure = 1,
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| truncated_reporting_triangle | A single truncated reporting_triangle object. May or may not contain NAs. |
| structure | Integer or vector specifying the reporting structure. If integer, divides columns evenly by that integer (with last possibly truncated). If vector, the sum must not be greater than or equal to the number of columns. Default is 1 (standard triangular structure). |
| validate | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

A single retrospective reporting triangle matrix with NAs in the appropriate positions.

## See Also

Retrospective data generation functions apply_reporting_structures(), truncate_to_row(), truncate_to_rows()

## Examples

```
# Standard triangular structure (default)
rep_tri <- apply_reporting_structure(example_reporting_triangle)
rep_tri

# Ragged structure with 2 columns per delay period
rep_ragged <- apply_reporting_structure(example_reporting_triangle, 2)
rep_ragged

# Custom structure with explicit column counts
rep_custom <- apply_reporting_structure(example_reporting_triangle, c(1, 2))
rep_custom
```

---

# apply_reporting_structures

*Apply reporting structures to generate retrospective reporting triangles*

---

## Description

This function applies a reporting structure to a list of truncated reporting triangles by setting observations to NA row by row from the bottom up based on the specified structure. This generates the reporting triangles that would have been available at each retrospective time point. It operates on each element in the list in order (from most recent retrospective nowcast time to oldest retrospective nowcast time).

## Usage

```
apply_reporting_structures(
  truncated_reporting_triangles,
  structure = 1,
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| `truncated_reporting_triangles` | List of `n` truncated reporting triangle matrices with as many rows as available given the truncation. |
| `structure` | Integer or vector specifying the reporting structure. If integer, divides columns evenly by that integer (with last possibly truncated). If vector, the sum must not be greater than or equal to the number of columns. Default is 1 (standard triangular structure). |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

`reporting_triangles` List of retrospective reporting triangles, generated by removing the bottom right observations from the truncated reporting triangle matrices.

## See Also

Retrospective data generation functions `apply_reporting_structure()`, `truncate_to_row()`, `truncate_to_rows()`

## Examples

```
# Generate retrospective triangles from truncated triangles
trunc_rts <- truncate_to_rows(example_reporting_triangle, n = 2)
retro_rts <- apply_reporting_structures(trunc_rts)

# With custom structure
retro_rts_custom <- apply_reporting_structures(
  trunc_rts,
  structure = 2
)
retro_rts_custom
```

`as.data.frame.reporting_triangle` *Convert reporting_triangle to data.frame*

## Description

Convert reporting_triangle to data.frame

## Usage

```
## S3 method for class 'reporting_triangle'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | A reporting_triangle object to convert. |
| `row.names` | NULL or character vector giving row names for the data frame. Missing values are not allowed. |
| `optional` | Logical. If TRUE, setting row names and converting column names is optional. |
| `...` | Additional arguments to be passed to or from methods. |

## Value

A data.frame with columns reference_date, report_date, delay, count

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Convert reporting triangle to data frame
df <- as.data.frame(example_reporting_triangle)
head(df)
```

`as.matrix.reporting_triangle` *Convert reporting_triangle to plain matrix*

## Description

Returns a plain matrix representation of a reporting_triangle object, removing the reporting_triangle class and custom attributes while preserving row and column names.

## Usage

```
## S3 method for class 'reporting_triangle'
as.matrix(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A reporting_triangle object. |
| `...` | Additional arguments (not used). |

## Value

A plain matrix without reporting_triangle class or attributes.

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Convert reporting_triangle to plain matrix
plain_mat <- as.matrix(example_downward_corr_rt)
class(plain_mat) # "matrix" "array"
```

assert_baselinenowcast_df *Assert validity of* baselinenowcast_df *objects*

## Description

Assert validity of baselinenowcast_df objects

## Usage

```
assert_baselinenowcast_df(data)
```

## Arguments

| | |
|---|---|
| data | A baselinenowcast_df object to check for validity. |

## Value

Returns NULL invisibly. Throws an error if validation fails.

## See Also

Main nowcasting interface functions baselinenowcast(), baselinenowcast.data.frame(), baselinenowcast.reporti
baselinenowcast_df-class, new_baselinenowcast_df()

## Examples

```
# Create a valid baselinenowcast_df object
valid_df <- data.frame(
  reference_date = as.Date("2024-01-01") + 0:4,
  pred_count = c(10, 15, 12, 18, 20),
  draw = 1,
  output_type = "point"
)
class(valid_df) <- c("baselinenowcast_df", "data.frame")

# Validate the object
assert_baselinenowcast_df(valid_df)
```

assert_reporting_triangle *Assert validity of* reporting_triangle *objects*

## Description

Assert validity of reporting_triangle objects

## Usage

```
assert_reporting_triangle(data, validate = TRUE)
```

## Arguments

| | |
|---|---|
| data | A reporting_triangle object to check for validity. |
| validate | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

Returns NULL invisibly. Throws an error if validation fails.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Validate an example reporting triangle
assert_reporting_triangle(example_reporting_triangle)
```

`as_ChainLadder_triangle` *Convert reporting_triangle to ChainLadder triangle format*

## Description

This function converts a reporting_triangle object to a triangle object from the ChainLadder package. ChainLadder is a mature package for claims reserving in general insurance that provides statistical methods for analysing reporting triangles, including the chain ladder technique, bootstrap methods, and diagnostic tools. Converting to ChainLadder format enables use of these specialized methods alongside baselinenowcast's nowcasting functionality.

## Usage

```
as_ChainLadder_triangle(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | A reporting_triangle object to convert. |
| `...` | Additional arguments passed to `ChainLadder::as.triangle()`. |

## Details

This function converts the reporting triangle to ChainLadder's triangle format using `ChainLadder::as.triangle()`. The ChainLadder package must be installed to use this function.

Once converted, you can use any ChainLadder methods such as:

- `ChainLadder::MackChainLadder()` for the Mack chain ladder method
- `ChainLadder::BootChainLadder()` for bootstrap chain ladder
- Standard plotting and summary methods

Note that some ChainLadder methods may require preprocessing for sparse triangles with many zeros, which can occur in syndromic surveillance data.

To convert back to a reporting_triangle object, use `as_reporting_triangle.triangle()`.

## Value

A ChainLadder triangle object (class "triangle" and "matrix"), with rows representing origin periods (reference dates) and columns representing development periods (delays).

## See Also

- `as_reporting_triangle.triangle()` for converting back

- ChainLadder package documentation

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Create a reporting triangle from synthetic NSSP data
data_as_of_df <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data = data_as_of_df)

# Convert to ChainLadder triangle format
cl_triangle <- as_ChainLadder_triangle(rep_tri)
print(cl_triangle)

# Visualize the reporting triangle structure
plot(cl_triangle)
```

---

# as_reporting_triangle *Create a* reporting_triangle *object*

---

## Description

Create a reporting_triangle object

## Usage

```
as_reporting_triangle(data, delays_unit = "days", ...)
```

## Arguments

| | |
|---|---|
| data | Data to be nowcasted. |
| delays_unit | Character string specifying the temporal granularity of the delays. Options are "days", "weeks", "months", "years". Default is "days". |
| ... | Additional arguments passed to methods. |

## Value

A reporting_triangle object

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Create a reporting triangle from a data.frame
data_as_of_df <- syn_nssp_df[
  syn_nssp_df$report_date <= "2026-04-01" &
    (syn_nssp_df$report_date - syn_nssp_df$reference_date) <= 25,
]
rt <- as_reporting_triangle(data = data_as_of_df)
rt
```

---

# as_reporting_triangle.data.frame

*Create a* reporting_triangle *object from a data.frame*

---

## Description

This method takes a data.frame containing case counts indexed by reference date and report date and creates a reporting_triangle object. See as_reporting_triangle.matrix() for other data input options.

## Usage

```
## S3 method for class 'data.frame'
as_reporting_triangle(
  data,
  delays_unit = "days",
  reference_date = "reference_date",
  report_date = "report_date",
  count = "count",
  ...
)
```

## Arguments

| | |
|---|---|
| data | Data.frame in a long tidy format with counts by reference date and report date. Must contain the following columns: . - Column of type date or character with the dates of the primary event occurrence (reference date). |

- Column of type `date` or character with the dates of report of the primary event (report_date).
- Column of numeric or integer indicating the new confirmed counts pertaining to that reference and report date (count). Additional columns can be included but will not be used. The input dataframe for this function must contain only a single strata, there can be no repeated reference dates and report dates.

`delays_unit` Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`.

`reference_date` Character string indicating the name of the column which represents the reference date, or the date of the primary event occurrence.

`report_date` Character string indicating the name of the column which represents the date the primary event was reported.

`count` Character string indicating the name of the column containing the number of incident cases on each reference and report date.

`...` Additional arguments not used.

## Value

A `reporting_triangle` object

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Filter to reasonable max_delay for faster example
data_as_of_df <- syn_nssp_df[
  syn_nssp_df$report_date <= "2026-04-01" &
    (syn_nssp_df$report_date - syn_nssp_df$reference_date) <= 25,
]
as_reporting_triangle(data = data_as_of_df)
```

`as_reporting_triangle.matrix` *Create a* `reporting_triangle` *from a matrix*

## Description

This method takes a matrix in the format of a reporting triangle, with rows as reference dates and columns as delays and elements as incident case counts and creates a reporting_triangle object. See `as_reporting_triangle.data.frame()` for creating from data frames.

## Usage

```
## S3 method for class 'matrix'
as_reporting_triangle(data, delays_unit = "days", reference_dates = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | Matrix of a reporting triangle where rows are reference times, columns are delays, and entries are the incident counts. The number of columns determines the maximum delay. |
| `delays_unit` | Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`. |
| `reference_dates` | Vector of Date objects or character strings indicating the reference dates corresponding to each row of the reporting triangle matrix (`data`). If NULL (default), dummy dates starting from 1900-01-01 are generated with spacing determined by `delays_unit`. |
| `...` | Additional arguments not used. |

## Value

A `reporting_triangle` object

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
rep_tri_mat <- matrix(
  c(
    1, 3, 5, 7, 9,
    4, 7, 8, 0, NA,
    9, 10, 0, NA, NA,
    3, 0, NA, NA, NA,
    6, NA, NA, NA, NA
  ),
  nrow = 5,
  byrow = TRUE
)

reference_dates <- seq(
  from = as.Date("2025-01-01"),
  to = as.Date("2025-01-05"),
  by = "day"
)

# max_delay is inferred from matrix dimensions (4 in this case)
rep_tri <- as_reporting_triangle(
  data = rep_tri_mat,
  reference_dates = reference_dates
)
rep_tri
```

---

# as_reporting_triangle.triangle

*Convert ChainLadder triangle to reporting_triangle format*

---

## Description

This S3 method converts a ChainLadder triangle object to a reporting_triangle object, enabling use of baselinenowcast's nowcasting methods.

## Usage

```
## S3 method for class 'triangle'
as_reporting_triangle(data, delays_unit = "days", reference_dates = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | A ChainLadder triangle object (class "triangle"). |
| `delays_unit` | Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`. |
| `reference_dates` | Vector of dates corresponding to the rows of the triangle. If not provided, will attempt to coerce row names to dates. If row names cannot be coerced to dates and this is not provided, an error will be raised. |

... Additional arguments passed to as_reporting_triangle.matrix().

## Details

This method converts a ChainLadder triangle back to baselinenowcast's reporting_triangle format. If reference_dates is not provided, the function will attempt to extract dates from the triangle's row names.

The ChainLadder package must be installed to use this function.

The conversion uses as_reporting_triangle.matrix() internally after extracting the matrix from the ChainLadder triangle object.

## Value

A reporting_triangle object. See reporting_triangle for details on the structure.

## See Also

- as_ChainLadder_triangle() for converting to ChainLadder
- as_reporting_triangle.matrix() for the underlying method
- as_reporting_triangle.data.frame() for creating from data frames

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Create a reporting triangle
data_as_of_df <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data = data_as_of_df)

# Convert to ChainLadder triangle
cl_triangle <- as_ChainLadder_triangle(rep_tri)

# Convert back to reporting_triangle (seamless round-trip)
# max_delay is inferred from the ChainLadder triangle dimensions
rep_tri_2 <- as_reporting_triangle(data = cl_triangle)
print(rep_tri_2)
```

baselinenowcast *Generate a nowcast*

## Description

This function ingests data to be nowcasted and generates a a baselinenowcast_df which contains a probabilistic or point estimate of the final case counts at each reference date in the `data`. See `baselinenowcast.reporting_triangle()` for details on the input requirements.

## Usage

```
baselinenowcast(
  data,
  scale_factor = 3,
  prop_delay = 0.5,
  output_type = c("samples", "point"),
  draws = 1000,
  uncertainty_model = fit_by_horizon,
  uncertainty_sampler = sample_nb,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | Data to be nowcasted |
| `scale_factor` | Numeric value indicating the multiplicative factor on the maximum delay to be used for estimation of delay and uncertainty. Default is `3`. |
| `prop_delay` | Numeric value <1 indicating what proportion of all reference times in the reporting triangle to be used for delay estimation. Default is `0.5`. |
| `output_type` | Character string indicating whether the output should be samples (`"samples"`) from the estimate with full uncertainty or whether to return the point estimate (`"point"`). Default is `"samples"`. If `"point"`estimates are specified, the minimum number of reference times needed is the number needed for delay estimation, otherwise, if `"samples"` are specified, at least 2 additional reference times are required for uncertainty estimation. |
| `draws` | Integer indicating the number of probabilistic draws to include if `output_type` is `"samples"`. Default is 1000. |
| `uncertainty_model` | Function that ingests a matrix of observations and a matrix of predictions and returns a vector that can be used to apply uncertainty using the same error model. Default is `fit_by_horizon` with arguments of `obs` matrix of observations and `pred` the matrix of predictions that fits each column (horizon) to a negative binomial observation model by default. The user can specify a different fitting model by replacing the `fit_model` argument in `fit_by_horizon`. |

uncertainty_sampler
: Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector.

...
: Additional arguments passed to methods.

## Value

Data.frame of class baselinenowcast_df

## See Also

Main nowcasting interface functions `assert_baselinenowcast_df()`, `baselinenowcast.data.frame()`, `baselinenowcast.reporting_triangle()`, `baselinenowcast_df-class`, `new_baselinenowcast_df()`

## Examples

```
# Generate a point nowcast from a reporting triangle
nowcast <- baselinenowcast(
  example_reporting_triangle,
  output_type = "point"
)
nowcast

# Generate probabilistic nowcast with samples
baselinenowcast(
  example_reporting_triangle,
  output_type = "samples",
  draws = 100
)
```

---

# baselinenowcast.data.frame

*Create a dataframe of nowcast results from a dataframe of cases indexed by reference date and report date*

---

## Description

This function ingests a data.frame with the number of incident cases indexed by reference date and report date for one or multiple strata, which define the unit of a single nowcast (e.g. age groups or locations). It returns a data.frame containing nowcasts by reference date for each strata, which are by default estimated independently. This function will by default estimate uncertainty using past retrospective nowcast errors and generate probabilistic nowcasts, which are samples from the predictive distribution of the estimated final case count at each reference date.

This function implements the full nowcasting workflow on multiple reporting triangles, generating estimates of the delay and uncertainty parameters for all strata using estimates from across strata if specified.

1. `estimate_delay()` - Estimate a delay PMF across strata if `strata_sharing` contains `"delay"`
2. `estimate_uncertainty_retro()` - Estimates uncertainty parameters across strata if `strata_sharing` contains `"uncertainty"`
3. `as_reporting_triangle()` - Generates a reporting triangle object from a data.frame
4. `baselinenowcast.reporting_triangle()` - Generates point or probabilistic nowcasts depending on `output_type` for each strata.

@detail See documentation for the arguments of this function which can be used to set the model specifications (things like number of reference times for delay and uncertainty estimation, the observation model, etc.). The function expects that each strata in the dataframe has the same maximum delay. If sharing estimates across all strata, the shared estimates will be made using the shared set of reference and report dates across strata.

## Usage

```
## S3 method for class 'data.frame'
baselinenowcast(
  data,
  scale_factor = 3,
  prop_delay = 0.5,
  output_type = c("samples", "point"),
  draws = 1000,
  uncertainty_model = fit_by_horizon,
  uncertainty_sampler = sample_nb,
  max_delay = NULL,
  delays_unit = "days",
  strata_cols = NULL,
  strata_sharing = "none",
  preprocess = preprocess_negative_values,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | Data.frame in a long tidy format with counts by reference date and report date for one or more strata. Must contain the following columns: - `reference_date`: Column of type `Date` containing the dates of the primary event occurrence.<br>• `report_date`: Column of type `Date` containing the dates of report of the primary event.<br>• `count`: Column of numeric or integer indicating the new confirmed counts pertaining to that reference and report date. Additional columns indicating the columns which set the unit of a single can be included. The user can specify these columns with the `strata_cols` argument, otherwise it will be assumed that the `data` contains only data for a single strata. |
| `scale_factor` | Numeric value indicating the multiplicative factor on the maximum delay to be used for estimation of delay and uncertainty. Default is `3`. |
| `prop_delay` | Numeric value <1 indicating what proportion of all reference times in the reporting triangle to be used for delay estimation. Default is `0.5`. |

`output_type` Character string indicating whether the output should be samples (`"samples"`) from the estimate with full uncertainty or whether to return the point estimate (`"point"`). Default is `"samples"`. If `"point"`estimates are specified, the minimum number of reference times needed is the number needed for delay estimation, otherwise, if `"samples"` are specified, at least 2 additional reference times are required for uncertainty estimation.

`draws` Integer indicating the number of probabilistic draws to include if `output_type` is `"samples"`. Default is 1000.

`uncertainty_model`
Function that ingests a matrix of observations and a matrix of predictions and returns a vector that can be used to apply uncertainty using the same error model. Default is `fit_by_horizon` with arguments of `obs` matrix of observations and `pred` the matrix of predictions that fits each column (horizon) to a negative binomial observation model by default. The user can specify a different fitting model by replacing the `fit_model` argument in `fit_by_horizon`.

`uncertainty_sampler`
Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector.

`max_delay` Maximum delay (in units of `delays_unit`) to include in the nowcast. If NULL (default), all delays in the data are used. If specified, only observations with delay <= max_delay are included.

`delays_unit` Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`.

`strata_cols` Vector of character strings indicating the names of the columns in `data` that determine how to stratify the data for nowcasting. The unique combinations of the entries in the `strata_cols` denote the unit of a single nowcast. Within a strata, there can be no repeated unique combinations of reference dates and report dates. Default is `NULL` which assumes that the data.frame being passed in represents a single strata (only one nowcast will be produced). All columns that are not part of the `strata_cols` will be removed.

`strata_sharing` Vector of character strings. Indicates if and what estimates should be shared for different nowcasting steps. Options are `"none"` for no sharing (each `strata_cols` is fully independent), `"delay"` for delay sharing and `"uncertainty"` for uncertainty sharing. Both `"delay"` and `"uncertainty"` can be passed at the same time.

`preprocess` Function to apply to the reporting triangle before estimation, or NULL to skip preprocessing. Default is `preprocess_negative_values()`, which handles negative values by redistributing them to earlier delays. Set to NULL if you want to preserve negative values. Custom preprocess functions must accept a `validate` parameter (defaults to TRUE) to enable validation optimisation in internal function chains.

`...` Additional arguments passed to `estimate_uncertainty()` and `sample_nowcast()`.

## Value

Data.frame of class baselinenowcast_df

## See Also

Main nowcasting interface functions assert_baselinenowcast_df(), baselinenowcast(), baselinenowcast.reportin
baselinenowcast_df-class, new_baselinenowcast_df()

## Examples

```
# Filter data to exclude most recent report dates and limit to 75
# reference dates
max_ref_date <- max(germany_covid19_hosp$reference_date)
min_ref_date <- max_ref_date - 74
covid_data_to_nowcast <- germany_covid19_hosp[
  germany_covid19_hosp$report_date < max_ref_date &
    germany_covid19_hosp$reference_date >= min_ref_date,
]
nowcasts_df <- baselinenowcast(covid_data_to_nowcast,
  max_delay = 25,
  strata_cols = c("age_group", "location"),
  draws = 100
)
nowcasts_df
```

---

# baselinenowcast.reporting_triangle

*Create a dataframe of nowcast results from a single reporting triangle*

---

## Description

This function ingests a single reporting_triangle object and generates a nowcast in the form of a baselinenowcast_df object.

This function implements a nowcasting workflow for a single reporting triangle:

1. allocate_reference_times() - Allocate the reference times used for delay and uncertainty estimation
2. estimate_delay() - Estimate a reporting delay PMF
3. apply_delay() - Generate a point nowcast using the delay PMF
4. estimate_and_apply_uncertainty() - Generate a probabilistic nowcast from a point nowcast and reporting triangle

This function will by default estimate the delay from the reporting_triangle and estimate uncertainty using past retrospective nowcast errors on that reporting_triangle to generate probabilistic nowcasts, which are samples from the predictive distribution of the estimated final case count at each reference date. Alternatives include passing in a separate delay_pmf or uncertainty_params. This method specifically computes a nowcast for a single reporting triangle. See documentation for the arguments of this function which can be used to set the model specifications (things like number of reference times for delay and uncertainty estimation, the observation model, etc.).

## Usage

```
## S3 method for class 'reporting_triangle'
baselinenowcast(
  data,
  scale_factor = 3,
  prop_delay = 0.5,
  output_type = c("samples", "point"),
  draws = 1000,
  uncertainty_model = fit_by_horizon,
  uncertainty_sampler = sample_nb,
  delay_pmf = NULL,
  uncertainty_params = NULL,
  preprocess = preprocess_negative_values,
  validate = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | reporting_triangle class object to be nowcasted. The matrix must contain missing observations in the form of NAs in order to generate an output from this function. |
| `scale_factor` | Numeric value indicating the multiplicative factor on the maximum delay to be used for estimation of delay and uncertainty. Default is `3`. |
| `prop_delay` | Numeric value <1 indicating what proportion of all reference times in the reporting triangle to be used for delay estimation. Default is `0.5`. |
| `output_type` | Character string indicating whether the output should be samples (`"samples"`) from the estimate with full uncertainty or whether to return the point estimate (`"point"`). Default is `"samples"`. If `"point"`estimates are specified, the minimum number of reference times needed is the number needed for delay estimation, otherwise, if `"samples"` are specified, at least 2 additional reference times are required for uncertainty estimation. |
| `draws` | Integer indicating the number of probabilistic draws to include if `output_type` is `"samples"`. Default is 1000. |
| `uncertainty_model` | Function that ingests a matrix of observations and a matrix of predictions and returns a vector that can be used to apply uncertainty using the same error model. Default is `fit_by_horizon` with arguments of `obs` matrix of observations and `pred` the matrix of predictions that fits each column (horizon) to a negative binomial observation model by default. The user can specify a different fitting model by replacing the `fit_model` argument in `fit_by_horizon`. |
| `uncertainty_sampler` | Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector. |

`delay_pmf` Vector of delays assumed to be indexed starting at the first delay column in the reporting triangle. Default is NULL, which will estimate the delay from the reporting triangle in `data`. See `estimate_delay()` for more details.

`uncertainty_params` Vector of uncertainty parameters ordered from horizon 1 to the maximum horizon. Default is `NULL`, which will result in computing the uncertainty parameters from the reporting triangle `data`. See `estimate_uncertainty()` for more details.

`preprocess` Function to apply to the reporting triangle before estimation, or NULL to skip preprocessing. Default is `preprocess_negative_values()`, which handles negative values by redistributing them to earlier delays. Set to NULL if you want to preserve negative values. Custom preprocess functions must accept a `validate` parameter (defaults to TRUE) to enable validation optimisation in internal function chains.

`validate` Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated.

`...` Additional arguments passed to `estimate_uncertainty()` and `sample_nowcast()`.

## Value

Data.frame of class `baselinenowcast_df`

## See Also

Main nowcasting interface functions `assert_baselinenowcast_df()`, `baselinenowcast()`, `baselinenowcast.data.fra`
`baselinenowcast_df-class`, `new_baselinenowcast_df()`

## Examples

```
# Filter to recent data and truncate to reasonable max_delay for faster
# example
data_as_of_df <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data = data_as_of_df) |>
  truncate_to_delay(max_delay = 25) |>
  tail(n = 40)
nowcast_df <- baselinenowcast(rep_tri, draws = 100)
nowcast_df
```

---

# baselinenowcast_df-class

*Nowcast Data.frame Object*

---

## Description

A `baselinenowcast_df` object which contains point or probabilistic nowcasts alongside reference dates and any additional metadata, in tidy data format. Nowcasts are presented aggregated across delays, by reference date.

## Value

A `baselinenowcast_df` object. This is a data.frame subclass containing nowcast results. See the Structure section for details on the required columns.

## Structure

A `baselinenowcast_df` is a data.frame with the following columns:

**reference_date** Dates corresponding to the reference times of the nowcast.

**pred_count** Numeric indicating the estimated total counts aggregated across delays at each reference date.

**draw** Integer indexing the sample from the probabilistic nowcast distribution. If `output_type = "point"`, this will be set to 1.

**output_type** Character string indicating whether the `pred_count` represents a probabilistic draw from the observation model indicated by `"samples"` or whether the `pred_count` is a point estimate indicated by `"point"`.

See the corresponding reporting_triangle and `baselinenowcast()` function for more details on the required inputs to generate the object.

## See Also

Main nowcasting interface functions `assert_baselinenowcast_df()`, `baselinenowcast()`, `baselinenowcast.data.fra`
`baselinenowcast.reporting_triangle()`, `new_baselinenowcast_df()`

---

# `combine_obs_with_pred` *Combine observed data with a single prediction draw*

---

## Description

Internally it sums observed counts from the reporting triangle by reference time and adds them to the predicted counts to form a single draw of the nowcast for the final counts by reference time.

## Usage

```
combine_obs_with_pred(
  predicted_counts,
  reporting_triangle,
  ref_time_aggregator = identity,
  delay_aggregator = function(x) rowSums(x, na.rm = TRUE)
)
```

## Arguments

`predicted_counts`
: Vector of predicted counts at each reference date. Note that if using a reference time or delay aggregator function, this is assumed to have already been aggregated.

`reporting_triangle`
: A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data).

`ref_time_aggregator`
: Function that operates along the rows (reference times) of the retrospective point nowcast matrix before it has been aggregated across columns (delays). Default is `identity` which does not aggregate across reference times.

`delay_aggregator`
: Function that operates along the columns (delays) of the retrospective point nowcast matrix after it has been aggregated across reference times. Default is `function(x) rowSums(x, na.rm = TRUE)`.

## Value

A vector of predicted counts at each reference date, for all reference dates in the input `reporting_triangle` (or fewer if using `ref_time_aggregator`)

## See Also

Probabilistic nowcast generation functions `sample_nb()`, `sample_nowcast()`, `sample_nowcasts()`, `sample_prediction()`, `sample_predictions()`

## Examples

```
# Use example data
reporting_triangle <- apply_reporting_structure(example_reporting_triangle)
pred_counts <- c(10, 20, 30, 40)
combine_obs_with_pred(pred_counts, reporting_triangle)

# Example with rolling sum
if (requireNamespace("zoo", quietly = TRUE)) {
  combine_obs_with_pred(pred_counts,
    reporting_triangle,
    ref_time_aggregator = function(x) zoo::rollsum(x, k = 2, align = "right")
  )
}
```

`estimate_and_apply_delay` *Estimate and apply delay from a reporting triangle*

## Description

This function generates a point nowcast by estimating a delay distribution from the reporting triangle and applying it to complete the triangle. If a delay distribution is specified, this will be used to generate the nowcast, otherwise, a delay distribution will be estimated from the `reporting_triangle`.

## Usage

```
estimate_and_apply_delay(
  reporting_triangle,
  n = nrow(reporting_triangle),
  delay_pmf = NULL,
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `n` | Integer indicating the number of reference times (observations) to be used in the estimate of the reporting delay, always starting from the most recent reporting delay. The default is to use the whole reporting triangle, so `nrow(reporting_triangle)`. |
| `delay_pmf` | Vector of delays assumed to be indexed starting at the first delay column in `reporting_triangle`. Default is `NULL`, which will estimate a delay from the `reporting_triangle`. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

`pt_nowcast_matrix` A `reporting_triangle` object of point nowcasts with the same structure as the input

## See Also

High-level workflow wrapper functions `allocate_reference_times()`, `estimate_and_apply_delays()`, `estimate_and_apply_uncertainty()`, `estimate_uncertainty_retro()`

## Examples

```
# Estimate and apply delay using default parameters
pt_nowcast_matrix <- estimate_and_apply_delay(
  reporting_triangle = example_reporting_triangle
)
pt_nowcast_matrix

# Use downward correction example with specific rows for delay estimation
pt_nowcast_matrix <- estimate_and_apply_delay(
  reporting_triangle = example_downward_corr_rt,
  n = 5
)
pt_nowcast_matrix

# Provide a pre-computed delay PMF
delay_pmf <- estimate_delay(
  reporting_triangle = example_reporting_triangle
)
pt_nowcast_matrix <- estimate_and_apply_delay(
  reporting_triangle = example_reporting_triangle,
  delay_pmf = delay_pmf
)
pt_nowcast_matrix
```

---

# estimate_and_apply_delays

*Estimate and apply delays to generate retrospective nowcasts*

---

## Description

This function ingests a list of incomplete reporting triangles and generates a list of point nowcast matrices, based on the delay estimated in each triangle or the corresponding delay passed in. It uses the specified n number of reference times to estimate the delay in each retrospective reporting triangle.

## Usage

```
estimate_and_apply_delays(
  retro_reporting_triangles,
  n = min(sapply(retro_reporting_triangles, nrow)),
  delay_pmf = NULL,
  validate = TRUE
)
```

## Arguments

`retro_reporting_triangles`
: List of reporting triangles to generate nowcasts for. Typically created by apply_reporting_structures(

| | |
|---|---|
| n | Integer indicating the number of reference times (number of rows) to use to estimate the delay distribution for each reporting triangle. Default is the minimum of the number of rows of all the matrices in retro_reporting_triangles. |
| delay_pmf | Vector or list of vectors of delays assumed to be indexed starting at the first delay column in each of the matrices in retro_reporting_triangles. If a list, must be of the same length as retro_reporting_triangles, with elements aligning. Default is NULL. |
| validate | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

point_nowcast_matrices List of the same number of elements as the input retro_reporting_triangles but with each reporting triangle filled in based on the delay estimated in that reporting triangle.

## See Also

High-level workflow wrapper functions allocate_reference_times(), estimate_and_apply_delay(), estimate_and_apply_uncertainty(), estimate_uncertainty_retro()

## Examples

```
# Generate retrospective nowcasts using larger triangle
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 25) |>
  tail(n = 50)
trunc_rts <- truncate_to_rows(rep_tri, n = 2)
retro_rts <- apply_reporting_structures(trunc_rts)
retro_pt_nowcast_mat_list <- estimate_and_apply_delays(retro_rts, n = 30)
retro_pt_nowcast_mat_list[1:2]

# Using a pre-computed delay PMF
delay <- estimate_delay(rep_tri, n = 30)
retro_pt_nowcast_mat_list <- estimate_and_apply_delays(
  retro_rts,
  n = 30,
  delay_pmf = delay
)
retro_pt_nowcast_mat_list[1:2]
```

---

# estimate_and_apply_uncertainty

*Estimate and apply uncertainty to a point nowcast matrix*

---

## Description

Generates probabilistic nowcasts by estimating uncertainty parameters from retrospective nowcasts and applying them to a point nowcast matrix.

This function combines:

1. `estimate_uncertainty_retro()` - Estimates uncertainty parameters using retrospective nowcasts
2. `sample_nowcasts()` - Applies uncertainty to generate draws

To obtain estimates of uncertainty parameters, use `estimate_uncertainty_retro()`. For full control over individual steps (e.g., custom matrix preparation, alternative aggregation), use the low-level functions (`truncate_to_rows()`, `apply_reporting_structures()`, `estimate_and_apply_delays()`, `estimate_uncertainty()`) directly.

## Usage

```
estimate_and_apply_uncertainty(
  point_nowcast_matrix,
  reporting_triangle,
  n_history_delay,
  n_retrospective_nowcasts,
  structure = get_reporting_structure(reporting_triangle),
  draws = 1000,
  delay_pmf = NULL,
  uncertainty_model = fit_by_horizon,
  uncertainty_sampler = sample_nb,
  validate = TRUE,
  ...
)
```

## Arguments

`point_nowcast_matrix`
: Matrix of point nowcast predictions and observations, with rows representing the reference times and columns representing the delays.

`reporting_triangle`
: A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data).

`n_history_delay`
: Integer indicating the number of reference times (observations) to be used in the estimate of the reporting delay, always starting from the most recent reporting delay.

`n_retrospective_nowcasts`
: Integer indicating the number of retrospective nowcast times to use for uncertainty estimation.

`structure` Integer or vector specifying the reporting structure. If integer, divides columns evenly by that integer (with last possibly truncated). If vector, the sum must not be greater than or equal to the number of columns. Default is 1 (standard triangular structure).

`draws` Integer indicating the number of draws of the predicted nowcast vector to generate. Default is `1000`.

`delay_pmf` Vector or list of vectors of delays assumed to be indexed starting at the first delay column in each of the matrices in `retro_reporting_triangles`. If a list, must be of the same length as `retro_reporting_triangles`, with elements aligning. Default is `NULL`.

`uncertainty_model`
Function that ingests a matrix of observations and a matrix of predictions and returns a vector that can be used to apply uncertainty using the same error model. Default is `fit_by_horizon` with arguments of `obs` matrix of observations and `pred` the matrix of predictions that fits each column (horizon) to a negative binomial observation model by default. The user can specify a different fitting model by replacing the `fit_model` argument in `fit_by_horizon`.

`uncertainty_sampler`
Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector.

`validate` Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated.

`...` Additional arguments to `estimate_uncertainty()` and `sample_prediction()`.

## Value

`nowcast_draws_df` Dataframe containing draws of combined observations and probabilistic predictions at each reference time.

## See Also

High-level workflow wrapper functions `allocate_reference_times()`, `estimate_and_apply_delay()`, `estimate_and_apply_delays()`, `estimate_uncertainty_retro()`

## Examples

```
# Use package data truncated to appropriate size
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
triangle <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 25)

pt_nowcast_matrix <- estimate_and_apply_delay(
  reporting_triangle = triangle,
  n = 75
)
# Use 75 reference times for delay estimation and 40 for uncertainty
```

```
nowcast_draws_df <- estimate_and_apply_uncertainty(
  pt_nowcast_matrix,
  triangle,
  n_history_delay = 75,
  n_retrospective_nowcasts = 40,
  draws = 100
)
head(nowcast_draws_df)
```

---

`estimate_delay` *Estimate a delay distribution from a reporting triangle*

---

## Description

Provides an estimate of the reporting delay as a function of the delay, based on the reporting triangle and the number of reference date observations to be used in the estimation. This point estimate of the delay is computed empirically, using an iterative algorithm starting from the most recent observations. Use `truncate_to_delay()` if you want to limit the maximum delay before estimation. This code was adapted from code written (under an MIT license) by the Karlsruhe Institute of Technology RESPINOW German Hospitalization Nowcasting Hub. Modified from: https://github.com/KITmetricslab/RESPINOW-Hub/blob/7cce3ae2728116e8c8cc0e4ab29074462c24650e/code/baseline/functions.R#L55 #nolint

## Usage

```
estimate_delay(
  reporting_triangle,
  n = nrow(reporting_triangle),
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `n` | Integer indicating the number of reference times (observations) to be used in the estimate of the reporting delay, always starting from the most recent reporting delay. The default is to use the whole reporting triangle, so `nrow(reporting_triangle)`. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

Vector indexed at 0 of length `ncol(reporting_triangle)` with columns indicating the point estimate of the empirical probability mass on each delay.

## See Also

Delay distribution estimation functions `preprocess_negative_values()`

## Examples

```
# Example 1: Standard usage
delay_pmf <- estimate_delay(
  reporting_triangle = example_reporting_triangle
)
delay_pmf

# Example 2: Using data with downward corrections (negatives preserved)
# Low-level functions process triangles as-is without preprocessing
delay_pmf_negative <- estimate_delay(
  reporting_triangle = example_downward_corr_rt,
  n = 5
)
delay_pmf_negative

# Example 3: Preprocess explicitly before estimation if needed
preprocessed_triangle <- preprocess_negative_values(example_downward_corr_rt)
delay_pmf_preprocessed <- estimate_delay(
  reporting_triangle = preprocessed_triangle,
  n = 5
)
delay_pmf_preprocessed
```

---

# estimate_uncertainty *Estimate uncertainty parameters*

---

## Description

This function ingests a list of point nowcast matrices and a corresponding list of truncated reporting matrices and uses both to estimate a vector of uncertainty parameters from the observations and estimates at each horizon, starting at 0 up until the max delay number of horizons.

## Usage

```
estimate_uncertainty(
  point_nowcast_matrices,
  truncated_reporting_triangles,
  retro_reporting_triangles,
  n = length(point_nowcast_matrices),
  uncertainty_model = fit_by_horizon,
  ref_time_aggregator = identity,
  delay_aggregator = function(x) rowSums(x, na.rm = TRUE),
  validate = TRUE
)
```

## Arguments

`point_nowcast_matrices`
: List of point nowcast matrices where rows represent reference time points and columns represent delays.

`truncated_reporting_triangles`
: List of truncated reporting matrices, containing all observations as of the latest reference time. Elements of list are paired with elements of `point_nowcast_matrices`.

`retro_reporting_triangles`
: List of `n` truncated reporting triangle matrices with as many rows as available given the truncation.

`n`
: Integer indicating the number of reporting matrices to use to estimate the uncertainty parameters.

`uncertainty_model`
: Function that ingests a matrix of observations and a matrix of predictions and returns a vector that can be used to apply uncertainty using the same error model. Default is `fit_by_horizon` with arguments of `obs` matrix of observations and `pred` the matrix of predictions that fits each column (horizon) to a negative binomial observation model by default. The user can specify a different fitting model by replacing the `fit_model` argument in `fit_by_horizon`.

`ref_time_aggregator`
: Function that operates along the rows (reference times) of the retrospective point nowcast matrix before it has been aggregated across columns (delays). Default is `identity` which does not aggregate across reference times.

`delay_aggregator`
: Function that operates along the columns (delays) of the retrospective point nowcast matrix after it has been aggregated across reference times. Default is `function(x) rowSums(x, na.rm = TRUE)`.

`validate`
: Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated.

## Value

`uncertainty_params` Vector of length of the number of horizons, with each element representing the estimate of the uncertainty parameter for each horizon. The specific parameter type depends on the chosen error model.

## See Also

Observation error estimation functions `fit_by_horizon()`, `fit_nb()`

## Examples

```
# Use example data to create reporting triangle
data_as_of_df <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data = data_as_of_df)

# Create retrospective nowcasts
trunc_rts <- truncate_to_rows(rep_tri, n = 2)
```

```
retro_rts <- apply_reporting_structures(trunc_rts)
retro_nowcasts <- estimate_and_apply_delays(retro_rts)

# Estimate dispersion parameters using default negative binomial model
disp_params <- estimate_uncertainty(
  point_nowcast_matrices = retro_nowcasts,
  truncated_reporting_triangles = trunc_rts,
  retro_reporting_triangles = retro_rts
)
disp_params

# Estimate dispersion parameters from rolling sum on the reference times
if (requireNamespace("zoo", quietly = TRUE)) {
  disp_params_agg <- estimate_uncertainty(
    point_nowcast_matrices = retro_nowcasts,
    truncated_reporting_triangles = trunc_rts,
    retro_reporting_triangles = retro_rts,
    ref_time_aggregator = function(x) zoo::rollsum(x, k = 2, align = "right")
  )
  disp_params_agg
}
```

---

`estimate_uncertainty_retro` *Estimate uncertainty parameters using retrospective nowcasts*

---

## Description

Estimates uncertainty parameters for nowcasting by creating a series of retrospective datasets from the input reporting triangle, generating point nowcasts for those datasets, and calibrating uncertainty parameters based on retrospective nowcast performance.

This function chains the retrospective nowcasting workflow:

1. `truncate_to_rows()` - Create retrospective snapshots
2. `apply_reporting_structures()` - Generate retrospective reporting triangles
3. `estimate_and_apply_delays()` - Generate point nowcasts
4. `estimate_uncertainty()` - Estimate uncertainty parameters

For full probabilistic nowcasts (uncertainty estimation + sampling), use `estimate_and_apply_uncertainty()`.

For more control over individual steps (e.g., custom matrix preparation, alternative aggregation), use the low-level functions directly.

## Usage

```
estimate_uncertainty_retro(
  reporting_triangle,
  n_history_delay,
  n_retrospective_nowcasts,
```

```
  structure = get_reporting_structure(reporting_triangle),
  delay_pmf = NULL,
  validate = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `n_history_delay` | Integer indicating the number of reference times (observations) to be used in the estimate of the reporting delay, always starting from the most recent reporting delay. |
| `n_retrospective_nowcasts` | Integer indicating the number of retrospective nowcast times to use for uncertainty estimation. |
| `structure` | Integer or vector specifying the reporting structure. If integer, divides columns evenly by that integer (with last possibly truncated). If vector, the sum must not be greater than or equal to the number of columns. Default is 1 (standard triangular structure). |
| `delay_pmf` | Vector or list of vectors of delays assumed to be indexed starting at the first delay column in each of the matrices in `retro_reporting_triangles`. If a list, must be of the same length as `retro_reporting_triangles`, with elements aligning. Default is `NULL`. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |
| `...` | Additional arguments passed to `estimate_uncertainty()`. |

## Value

A numeric vector of uncertainty parameters with length equal to one less than the number of columns in the reporting triangle, with each element representing the estimate of the uncertainty parameter for each horizon. Returns NULL if insufficient data is available for estimation.

## See Also

High-level workflow wrapper functions `allocate_reference_times()`, `estimate_and_apply_delay()`, `estimate_and_apply_delays()`, `estimate_and_apply_uncertainty()`

## Examples

```
# Create a reporting triangle from syn_nssp_df
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
```

```
  truncate_to_delay(max_delay = 25)

uncertainty_params <- estimate_uncertainty_retro(
  rep_tri,
  n_history_delay = 30,
  n_retrospective_nowcasts = 10
)
uncertainty_params
```

---

`example_downward_corr_rt` *Example reporting triangle with downward corrections*

---

## Description

A reporting_triangle object demonstrating how to handle systematic downward corrections in reporting data. This represents a realistic case where data quality reviews at delay 2 consistently identify false positives or reclassify cases, producing negative values at that specific delay.

## Usage

```
example_downward_corr_rt
```

## Format

A reporting_triangle object with 8 reference dates and 4 delays:

**reporting_triangle_matrix** 8x4 matrix with negative values at delay 2

**reference_dates** 8 dates starting from 2024-01-01

**delays_unit** "days"

## Details

Use this example to understand:

- How to work with negative corrections in delay distributions
- The impact of preprocessing negative values on delay estimation
- How PMFs and CDFs behave with systematic downward corrections

## See Also

- example_reporting_triangle for a clean example without corrections
- `baselinenowcast.reporting_triangle()` specifically the `preprocess` argument for a description of how to remove negative values if desired.

Example datasets `example_reporting_triangle`, `germany_covid19_hosp`, `syn_nssp_df`, `syn_nssp_line_list`

## Examples

```
# View the example triangle with downward corrections
example_downward_corr_rt

# Estimate delay with and without preprocessing
delay_raw <- estimate_delay(example_downward_corr_rt, n = 5)
delay_processed <- estimate_delay(
  preprocess_negative_values(example_downward_corr_rt),
  n = 5
)

# Compare the resulting PMFs
delay_raw
delay_processed
```

---

# example_reporting_triangle

*Simple example reporting triangle for demonstrations*

---

## Description

A basic reporting_triangle object demonstrating standard structure with complete early reference times and progressively incomplete recent times. Useful for simple examples and tests.

## Usage

```
example_reporting_triangle
```

## Format

A reporting_triangle object with 5 reference dates and 4 delays:

**reporting_triangle_matrix** 5x4 matrix with counts

**reference_dates** 5 dates starting from 2024-01-01

**delays_unit** "days"

## Details

This is a simple, clean example without complications like negative values or unusual structures. Ideal for:

- Package examples demonstrating basic functionality
- Unit tests for standard cases
- Vignettes introducing nowcasting concepts

Use example_downward_corr_rt for examples with data quality corrections.

## See Also

- example_downward_corr_rt for downward corrections example
- as_reporting_triangle() to create reporting triangles

Example datasets example_downward_corr_rt, germany_covid19_hosp, syn_nssp_df, syn_nssp_line_list

## Examples

```
# View the example triangle
example_reporting_triangle

# Use in nowcasting - requires complete rows for delay estimation
estimate_delay(example_reporting_triangle, n = 6)
```

---

# fit_by_horizon *Helper function that fits its each column of the matrix (horizon) to an observation model.*

---

## Description

Helper function that fits its each column of the matrix (horizon) to an observation model.

## Usage

```
fit_by_horizon(obs, pred, fit_model = fit_nb)
```

## Arguments

| | |
|---|---|
| obs | Matrix or vector of observations. |
| pred | Matrix or vector of predictions. |
| fit_model | Function that ingests observations and expectations and returns uncertainty parameters, default is fit_nb. |

## Value

Vector of uncertainty parameters of the same length as the number of columns in the obs matrix.

## See Also

Observation error estimation functions estimate_uncertainty(), fit_nb()

## Examples

```
obs <- matrix(
  c(
    5, 6, 2,
    1, 4, 2,
    8, 4, 2
  ),
  nrow = 3,
  byrow = TRUE
)
pred <- matrix(
  c(
    4.2, 5.2, 1.8,
    0.7, 3.5, 3.4,
    7.3, 4.1, 1.2
  ),
  nrow = 3,
  byrow = TRUE
)
disp <- fit_by_horizon(obs = obs, pred = pred)
disp
```

---

# fit_nb *Fit a negative binomial to a vector of observations and expectations*

---

## Description

Takes in a vector of observations and a vector of expectations and performs a MLE estimator to estimate the dispersion parameter of a negative binomial. This code was adapted from code written (under an MIT license) by the Karlsruhe Institute of Technology RESPINOW German Hospitalization Nowcasting Hub. Modified from: https://github.com/KITmetricslab/RESPINOW-Hub/blob/7fab4dce7b559c3076ab643cf22048cb5fb84cc2/code/baseline/functions.R#L404 #nolint

## Usage

```
fit_nb(x, mu)
```

## Arguments

| | |
|---|---|
| x | Vector of observed values. |
| mu | Vector of expected values. |

## Value

the maximum likelihood estimate of the dispersion

## See Also

Observation error estimation functions estimate_uncertainty(), fit_by_horizon()

## Examples

```
obs <- c(4, 8, 10)
pred <- c(3.1, 7.2, 11)
disp <- fit_nb(obs, pred)
disp
```

---

`germany_covid19_hosp` *Incident COVID-19 hospitalisations indexed by the date of positive test (reference date) and report date from Germany in 2021 and 2022.*

---

## Description

Incident COVID-19 hospitalisations indexed by the date of positive test (reference date) and report date from Germany in 2021 and 2022.

## Usage

```
germany_covid19_hosp
```

## Format

A data.frame with 140,630 rows and 6 columns.

**reference_date** Date of first positive COVID-19 test formatted in ISO8601 standards as YYYY-MM-DD.

**location** Character string indicating the location of the case counts

**age_group** Character string indicating the age group of the case counts.

**delay** Integer specifying the delay, in days, between the reference date and the report date

**count** Integer indicating the number of cases indexed by reference and report date.

**report_date** Date of case report, formatted in ISO8601 standards as YYYY-MM-DD.

## Source

This data comes directly from the preprocessed data in the German COVID-19 Nowcast Hub from https://github.com/KITmetricslab/hospitalization-nowcast-hub/blob/main/data-truth/COVID-19/COVID-19_hospitalizations_preprocessed.csv. #nolint It contains incident case counts by age group in Germany.

## See Also

Example datasets `example_downward_corr_rt`, `example_reporting_triangle`, `syn_nssp_df`, `syn_nssp_line_list`

`get_delays_from_dates` *Compute delays between report dates and reference dates*

## Description

Computes delays between report dates and reference dates using the specified time unit. This is the inverse operation of `get_report_dates()`.

## Usage

```
get_delays_from_dates(report_dates, reference_dates, delays_unit)
```

## Arguments

| | |
|---|---|
| `report_dates` | Date vector of report dates. |
| `reference_dates` | Date vector of reference dates. |
| `delays_unit` | Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`. |

## Value

Numeric vector of delays.

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Compute delays in days
ref_date <- as.Date("2024-01-01")
report_date <- as.Date("2024-01-08")
get_delays_from_dates(report_date, ref_date, "days") # 7

# Compute delays in weeks
report_date_weeks <- as.Date("2024-01-15")
get_delays_from_dates(report_date_weeks, ref_date, "weeks") # 2
```

## get_delays_unit *Get delays unit from a reporting triangle*

### Description

Get delays unit from a reporting triangle

### Usage

```
get_delays_unit(x)
```

### Arguments

| | |
|---|---|
| x | A reporting_triangle object. |

### Value

Character string indicating the delays unit.

### See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

### Examples

```
delays_unit <- get_delays_unit(example_reporting_triangle)
delays_unit
```

## get_max_delay *Get maximum delay from reporting_triangle*

### Description

Get maximum delay from reporting_triangle

### Usage

```
get_max_delay(x, non_zero = FALSE)
```

### Arguments

| | |
|---|---|
| x | A reporting_triangle object |
| non_zero | Logical. If TRUE, returns the maximum delay where at least one observation is non-zero. Useful for identifying the actual extent of the delay distribution. Default FALSE. |

### Value

Maximum delay (integer), or -1 if all zero when non_zero = TRUE

### See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

### Examples

```
# Get maximum delay from triangle structure
max_delay <- get_max_delay(example_reporting_triangle)
max_delay

# Get maximum delay with non-zero observations
max_delay_nz <- get_max_delay(example_reporting_triangle, non_zero = TRUE)
max_delay_nz
```

---

## get_mean_delay *Get mean delay for each row of reporting_triangle*

---

### Description

Get mean delay for each row of reporting_triangle

### Usage

```
get_mean_delay(x)
```

### Arguments

| | |
|---|---|
| x | A reporting_triangle object |

## Value

Vector of mean delays for each reference date (numeric)

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
mean_delays <- get_mean_delay(example_reporting_triangle)
mean_delays
```

---

# get_quantile_delay *Get quantile delay for each row of reporting_triangle*

---

## Description

Get quantile delay for each row of reporting_triangle

## Usage

```
get_quantile_delay(x, p = 0.99)
```

## Arguments

| | |
|---|---|
| x | A reporting_triangle object |
| p | Numeric value between 0 and 1 indicating the quantile to compute. For example, p = 0.99 returns the delay at which 99% of cases have been reported. Default is 0.99. |

## Value

Vector of quantile delays for each reference date (integer). Returns NA for rows with no observations.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Get 99th percentile delay for each reference date
quantile_delays_99 <- get_quantile_delay(example_reporting_triangle)
quantile_delays_99

# Get median delay
median_delays <- get_quantile_delay(example_reporting_triangle, p = 0.5)
median_delays
```

---

# get_reference_dates *Get reference dates from reporting_triangle*

---

## Description

Get reference dates from reporting_triangle

## Usage

```
get_reference_dates(x)
```

## Arguments

| | |
|---|---|
| x | A reporting_triangle object |

## Value

Vector of Date objects

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
ref_dates <- get_reference_dates(example_reporting_triangle)
head(ref_dates)
```

---

`get_reporting_structure` *Get reporting structure from a reporting triangle*

---

## Description

Returns an integer or vector specifying the reporting structure, which indicates how the reporting triangle is organized. This structure tells `apply_reporting_structure()` how to create new reporting triangles with the same reporting pattern.

## Usage

```
get_reporting_structure(reporting_triangle)
```

## Arguments

`reporting_triangle`
: A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data).

## Value

Integer or vector specifying the reporting structure. If integer, divides columns evenly by that integer (with last possibly truncated). If vector, the sum must not be greater than or equal to the number of columns. Default is 1 (standard triangular structure). If there are no NAs, will return 0.

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Get structure from example triangle
structure <- get_reporting_structure(example_reporting_triangle)
structure
```

`get_report_dates` *Compute report dates from reference dates and delays*

## Description

Adds delays to reference dates using unit-aware date arithmetic.

## Usage

```
get_report_dates(reference_dates, delays, delays_unit)
```

## Arguments

| | |
|---|---|
| `reference_dates` | Date vector of reference dates. |
| `delays` | Numeric vector of delays. |
| `delays_unit` | Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`. |

## Value

Date vector of report dates.

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Compute report dates with days
ref_date <- as.Date("2024-01-01")
get_report_dates(ref_date, 7, "days") # 2024-01-08

# Compute report dates with weeks
get_report_dates(ref_date, 2, "weeks") # 2024-01-15
```

head.reporting_triangle *Get first rows of a reporting_triangle*

## Description

Get first rows of a reporting_triangle

## Usage

```
## S3 method for class 'reporting_triangle'
head(x, n = 6L, ...)
```

## Arguments

| | |
|---|---|
| x | A reporting_triangle object. |
| n | Integer indicating the number of rows to return. Default is 6. |
| ... | Additional arguments (not currently used). |

## Value

First rows as a reporting_triangle.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Get first 3 rows
head(example_reporting_triangle, n = 3)
```

## is_reporting_triangle *Check if an object is a reporting_triangle*

### Description

Check if an object is a reporting_triangle

### Usage

```
is_reporting_triangle(x)
```

### Arguments

| | |
|---|---|
| x | An object to check. |

### Value

Logical indicating whether the object is a reporting_triangle.

### See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## new_baselinenowcast_df *Combine data from a nowcast dataframe, strata, and reference dates*

### Description

Combines data from a nowcast dataframe, a named list of the strata associated with the nowcast dataframe, and a vector of reference dates corresponding to the time column in the `baselinenowcast_df`

### Usage

```
new_baselinenowcast_df(baselinenowcast_df, reference_dates, output_type)
```

### Arguments

`baselinenowcast_df`
: Data.frame containing information for multiple draws with columns for the reference time (`time`), the predicted counts (`pred_count`), and the draw number (`draw`).

`reference_dates`
: Vector of reference dates corresponding to the reference times in the `baselinenowcast_df`.

`output_type`
: Character string indicating whether the output should be samples (`"samples"`) from the estimate with full uncertainty or whether to return the point estimate (`"point"`). Default is `"samples"`. If `"point"`estimates are specified, the minimum number of reference times needed is the number needed for delay estimation, otherwise, if `"samples"` are specified, at least 2 additional reference times are required for uncertainty estimation.

### Value

An object of class `baselinenowcast_df`

### See Also

Main nowcasting interface functions `assert_baselinenowcast_df()`, `baselinenowcast()`, `baselinenowcast.data.fra`
`baselinenowcast.reporting_triangle()`, `baselinenowcast_df-class`

---

## `new_reporting_triangle` *Class constructor for* `reporting_triangle` *objects*

---

### Description

Creates a new reporting_triangle object from a matrix.

### Usage

```
new_reporting_triangle(reporting_triangle_matrix, reference_dates, delays_unit)
```

### Arguments

`reporting_triangle_matrix`
: Matrix of reporting triangle where rows are reference times, columns are delays, and entries are incident counts.

`reference_dates`
: Vector of Date objects indicating the reference dates corresponding to each row of the matrix.

`delays_unit`
: Character string specifying the temporal granularity of the delays. Options are `"days"`, `"weeks"`, `"months"`, `"years"`. Default is `"days"`.

## Value

An object of class reporting_triangle

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

---

# preprocess_negative_values

*Preprocess negative values in the reporting triangle*

---

## Description

Takes in a reporting triangle and returns it with negative values of reporting handled by redistributing them to earlier delays (from longer delay to shorter). This is useful when dealing with reporting corrections that can result in negative incremental counts.

When negative values are detected, they are set to zero and the negative amount is subtracted from the count at the next earlier delay (moving from right to left in each row). This process continues until either the negative value is fully absorbed or the first delay is reached.

This code was adapted from code written (under an MIT license) by the Karlsruhe Institute of Technology RESPINOW German Hospitalization Nowcasting Hub. Modified from https://github.com/KITmetricslab/RESPINOW-Hub/blob/main/code/baseline/functions.R #nolint

## Usage

```
preprocess_negative_values(reporting_triangle, validate = TRUE)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Details

Use this function when:

- Your data contains reporting corrections that result in negative counts
- You want to preserve the total count while handling negatives
- You need a delay distribution that sums to 1 or a CDF that is weakly increasing

Do not use this function when:

- Your data naturally has negative PMF entries (e.g., from differencing)
- You want to preserve the original structure including negatives
- You are working with corrections that should be reflected as negative probabilities

## Value

A reporting_triangle object with negative values handled via redistribution to earlier delays.

## See Also

Delay distribution estimation functions `estimate_delay()`

## Examples

```
# Using example dataset with negative values from corrections
# Preprocess to handle negatives
preprocessed <- preprocess_negative_values(example_downward_corr_rt)
preprocessed
```

---

# print.reporting_triangle

*Print a reporting_triangle object*

---

## Description

Print a reporting_triangle object

## Usage

```
## S3 method for class 'reporting_triangle'
print(x, n_rows = 10, n_cols = 10, ...)
```

### Arguments

| | |
|---|---|
| x | A reporting_triangle object to print. |
| n_rows | Maximum number of rows to display. If the triangle has more rows, only the last n_rows rows are shown. Default is 10. Set to NULL to display all rows. |
| n_cols | Maximum number of columns to display. If the triangle has more columns, only the first n_cols columns are shown. Default is 10. Set to NULL to display all columns. |
| ... | Additional arguments passed to print methods. |

### Value

Invisibly returns the object.

### See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

---

## reporting_triangle-class

*Reporting Triangle Object*

---

### Description

A reporting_triangle object contains the data and metadata needed for nowcasting.

### Value

A reporting_triangle object. This is a matrix subclass containing case counts indexed by reference date (rows) and delay (columns). See the Structure section for details on the object format.

### Structure

A reporting_triangle is a matrix with class c("reporting_triangle", "matrix"):

- Rows: Reference dates
- Columns: Delays (0, 1, 2, ...)
- Entries: Incident cases at each reference date and delay
- Row names: Reference dates as character

- Column names: Delays as character

Attributes:

- `delays_unit`: Character ("days", "weeks", "months", "years")

Reference dates are stored as row names and can be extracted using `get_reference_dates()`. The maximum delay can be obtained using `get_max_delay()`. The structure can be computed using `get_reporting_structure()`. See the corresponding `as_reporting_triangle.matrix()` and `as_reporting_triangle.data.frame()` functions for more details on the required input formats to generate the object.

## Working with reporting triangles

Reporting triangle objects provide:

**Inspection and display:**

- `print()`: Informative display with metadata
- `summary()`: Statistics including completion, delays, and zeros
- `head()`, `tail()`: Extract first or last rows
- Standard matrix operations: `rowSums()`, `colSums()`

**Subsetting and modification:**

- `[` and `[<-`: Extract or assign values with automatic validation
- Subsetting preserves class and attributes when result is a matrix

**Package functions:**

- `estimate_and_apply_delay()`: Estimate delay and generate point nowcast
- `estimate_delay()`: Extract delay distribution from triangle
- `apply_delay()`: Apply delay distribution for nowcasting
- `truncate_to_row()`: Remove most recent rows
- `preprocess_negative_values()`: Handle reporting corrections

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `summary.reporting_triangle()`, `tail.reporting_triangle()`, `truncate_to_delay()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Create a reporting triangle from data
data <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data = data)

# Use with low-level functions
filled <- estimate_and_apply_delay(rep_tri)
delay_pmf <- estimate_delay(rep_tri)
nowcast <- apply_delay(rep_tri, delay_pmf)

# Direct matrix operations
total_by_date <- rowSums(rep_tri, na.rm = TRUE)
total_by_delay <- colSums(rep_tri, na.rm = TRUE)

# Subsetting and inspection
recent <- tail(rep_tri, n = 10)
summary(rep_tri)
```

---

# sample_nb *Sample from negative binomial model given a set of predictions*

---

## Description

Sample from negative binomial model given a set of predictions

## Usage

```
sample_nb(pred, uncertainty_params)
```

## Arguments

| | |
|---|---|
| pred | Vector of predictions. |
| uncertainty_params | Vector of uncertainty parameters. |

## Value

`sampled_pred` Object of the same dimensions as `pred` representing a single draw from the negative binomial distribution with the specified `uncertainty params`.

## See Also

Probabilistic nowcast generation functions `combine_obs_with_pred()`, `sample_nowcast()`, `sample_nowcasts()`, `sample_prediction()`, `sample_predictions()`

## Examples

```
pred <- c(3.2, 4.6)
sampled_preds <- sample_nb(pred,
  uncertainty_params = c(50, 100)
)
sampled_preds
```

---

`sample_nowcast` *Generate a single draw of a nowcast combining observed and predicted values*

---

## Description

Generate a single draw of a nowcast combining observed and predicted values

## Usage

```
sample_nowcast(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  uncertainty_sampler = sample_nb,
  ref_time_aggregator = identity,
  delay_aggregator = function(x) rowSums(x, na.rm = TRUE)
)
```

## Arguments

`point_nowcast_matrix`
: Matrix of point nowcast predictions and observations, with rows representing the reference times and columns representing the delays.

`reporting_triangle`
: A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data).

`uncertainty_params`
: Vector of uncertainty parameters ordered from horizon 1 to the maximum horizon. Note that these will be reversed internally to match the ordering of the `point_nowcast_matrix` (where a horizon of 1 is the last entry).

`uncertainty_sampler`
: Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector.

`ref_time_aggregator`
: Function that operates along the rows (reference times) of the retrospective point nowcast matrix before it has been aggregated across columns (delays). Default is `identity` which does not aggregate across reference times.

`delay_aggregator`
: Function that operates along the columns (delays) of the retrospective point nowcast matrix after it has been aggregated across reference times. Default is `function(x) rowSums(x, na.rm = TRUE)`.

## Value

Vector of predicted counts at each reference date based on combining the observed counts and the predicted counts for the unobserved elements. Returns values for all reference dates in the input `reporting_triangle` (or fewer if using `ref_time_aggregator`).

## See Also

Probabilistic nowcast generation functions `combine_obs_with_pred()`, `sample_nb()`, `sample_nowcasts()`, `sample_prediction()`, `sample_predictions()`

## Examples

```
# Generate point nowcast and uncertainty params from example data
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 5) |>
  tail(n = 10)
point_nowcast_matrix <- estimate_and_apply_delay(rep_tri, n = 10)
reporting_triangle <- apply_reporting_structure(rep_tri)
uncertainty_params <- estimate_uncertainty_retro(
  rep_tri,
  n_history_delay = 8,
  n_retrospective_nowcasts = 2
)
nowcast_draw <- sample_nowcast(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params
)
nowcast_draw
```

---

# sample_nowcasts *Generate multiple draws of a nowcast combining observed and predicted values*

---

## Description

Generate multiple draws of a nowcast combining observed and predicted values

## Usage

```
sample_nowcasts(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  draws = 1000,
  ...
)
```

## Arguments

| | |
|---|---|
| point_nowcast_matrix | Matrix of point nowcast predictions and observations, with rows representing the reference times and columns representing the delays. |
| reporting_triangle | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| uncertainty_params | Vector of uncertainty parameters ordered from horizon 1 to the maximum horizon. Note that these will be reversed internally to match the ordering of the point_nowcast_matrix (where a horizon of 1 is the last entry). |
| draws | Integer indicating the number of draws of the predicted nowcast vector to generate. Default is 1000. |
| ... | Additional arguments passed to sample_nowcast. |

## Value

Dataframe containing information for multiple draws with columns for the reference date (reference_date), the predicted counts (pred_count), and the draw number (draw). Returns predictions for all reference dates in the input reporting_triangle (or fewer if using ref_time_aggregator).

## See Also

Probabilistic nowcast generation functions combine_obs_with_pred(), sample_nb(), sample_nowcast(), sample_prediction(), sample_predictions()

## Examples

```
# Generate point nowcast and uncertainty params from example data
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 5) |>
  tail(n = 10)
point_nowcast_matrix <- estimate_and_apply_delay(rep_tri, n = 10)
reporting_triangle <- apply_reporting_structure(rep_tri)
uncertainty_params <- estimate_uncertainty_retro(
  rep_tri,
```

```
  n_history_delay = 8,
  n_retrospective_nowcasts = 2
)
nowcast_draws <- sample_nowcasts(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  draws = 5
)
nowcast_draws
```

---

`sample_prediction` *Get a draw of only the predicted elements of the nowcast vector*

---

## Description

Get a draw of only the predicted elements of the nowcast vector

## Usage

```
sample_prediction(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  uncertainty_sampler = sample_nb,
  ref_time_aggregator = identity,
  delay_aggregator = function(x) rowSums(x, na.rm = TRUE)
)
```

## Arguments

`point_nowcast_matrix`
: Matrix of point nowcast predictions and observations, with rows representing the reference times and columns representing the delays.

`reporting_triangle`
: A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data).

`uncertainty_params`
: Vector of uncertainty parameters ordered from horizon 1 to the maximum horizon. Note that these will be reversed internally to match the ordering of the `point_nowcast_matrix` (where a horizon of 1 is the last entry).

`uncertainty_sampler`
: Function that ingests a vector or matrix of predictions and a vector of uncertainty parameters and generates draws from the observation model. Default is `sample_nb` which expects arguments `pred` for the vector of predictions and uncertainty parameters for the corresponding vector of uncertainty parameters, and draws from a negative binomial for each element of the vector.

`ref_time_aggregator`
: Function that operates along the rows (reference times) of the retrospective point nowcast matrix before it has been aggregated across columns (delays). Default is `identity` which does not aggregate across reference times.

`delay_aggregator`
: Function that operates along the columns (delays) of the retrospective point nowcast matrix after it has been aggregated across reference times. Default is `function(x) rowSums(x, na.rm = TRUE)`.

## Value

Matrix of predicted draws at each reference date, for all reference dates in the input `point_nowcast_matrix` (or fewer if using `ref_time_aggregator`).

## See Also

Probabilistic nowcast generation functions `combine_obs_with_pred()`, `sample_nb()`, `sample_nowcast()`, `sample_nowcasts()`, `sample_predictions()`

## Examples

```
# Generate point nowcast and uncertainty params from example data
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 5) |>
  tail(n = 10)
point_nowcast_matrix <- estimate_and_apply_delay(rep_tri, n = 10)
reporting_triangle <- apply_reporting_structure(rep_tri)
uncertainty_params <- estimate_uncertainty_retro(
  rep_tri,
  n_history_delay = 8,
  n_retrospective_nowcasts = 2
)
nowcast_pred_draw <- sample_prediction(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params
)
nowcast_pred_draw

# Get draws on the rolling sum
if (requireNamespace("zoo", quietly = TRUE)) {
  nowcast_pred_draw_agg <- sample_prediction(
    point_nowcast_matrix,
    reporting_triangle,
    uncertainty_params,
    ref_time_aggregator = function(x) zoo::rollsum(x, k = 2, align = "right")
  )
  nowcast_pred_draw_agg
}
```

`sample_predictions` *Get a dataframe of multiple draws of only the predicted elements of the nowcast vector*

## Description

Get a dataframe of multiple draws of only the predicted elements of the nowcast vector

## Usage

```
sample_predictions(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  draws = 1000,
  ...
)
```

## Arguments

| | |
|---|---|
| `point_nowcast_matrix` | Matrix of point nowcast predictions and observations, with rows representing the reference times and columns representing the delays. |
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `uncertainty_params` | Vector of uncertainty parameters ordered from horizon 1 to the maximum horizon. Note that these will be reversed internally to match the ordering of the `point_nowcast_matrix` (where a horizon of 1 is the last entry). |
| `draws` | Integer indicating the number of draws of the predicted nowcast vector to generate. Default is `1000`. |
| `...` | Additional arguments passed to `sample_prediction`. |

## Value

Dataframe containing the predicted point nowcast vectors indexed by predicted count (`pred_count`), reference date (`reference_date`), and the draw index (`draw`). Returns predictions for all reference dates in the input `reporting_triangle` (or fewer if using `ref_time_aggregator`).

## See Also

Probabilistic nowcast generation functions `combine_obs_with_pred()`, `sample_nb()`, `sample_nowcast()`, `sample_nowcasts()`, `sample_prediction()`

## Examples

```
# Generate point nowcast and uncertainty params from example data
data_as_of <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
rep_tri <- as_reporting_triangle(data_as_of) |>
  truncate_to_delay(max_delay = 5) |>
  tail(n = 10)
point_nowcast_matrix <- estimate_and_apply_delay(rep_tri, n = 10)
reporting_triangle <- apply_reporting_structure(rep_tri)
uncertainty_params <- estimate_uncertainty_retro(
  rep_tri,
  n_history_delay = 8,
  n_retrospective_nowcasts = 2
)
nowcast_pred_draws <- sample_predictions(
  point_nowcast_matrix,
  reporting_triangle,
  uncertainty_params,
  draws = 5
)
nowcast_pred_draws

# Get nowcast pred draws over rolling sum
if (requireNamespace("zoo", quietly = TRUE)) {
  nowcast_pred_draws_rolling_df <- sample_predictions(
    point_nowcast_matrix,
    reporting_triangle,
    uncertainty_params,
    draws = 5,
    ref_time_aggregator = function(x) zoo::rollsum(x, k = 2, align = "right")
  )
  nowcast_pred_draws_rolling_df
}
```

---

`summary.reporting_triangle` *Summarize a reporting_triangle object*

---

## Description

Summarize a reporting_triangle object

## Usage

```
## S3 method for class 'reporting_triangle'
summary(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | A reporting_triangle object to summarize. |
| `...` | Additional arguments not used. |

## Value

Invisibly returns the object.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, tail.reporting_triangle, truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Display summary statistics
summary(example_reporting_triangle)
```

---

syn_nssp_df — *A synthetic dataset containing the number of incident cases indexed by reference date and report date. While data of this form could be from any source, this data is meant to represent the output of pre-processing the syn_nssp_line_list dataset, which is a synthetic patient-level line list data from the United State's National Syndromic Surveillance System (NSSP).*

---

## Description

A synthetic dataset containing the number of incident cases indexed by reference date and report date. While data of this form could be from any source, this data is meant to represent the output of pre-processing the syn_nssp_line_list dataset, which is a synthetic patient-level line list data from the United State's National Syndromic Surveillance System (NSSP).

## Usage

```
syn_nssp_df
```

## Format

A data.frame with 3795 rows and 3 columns.

**reference_date** Date the primary event occurred (e.g. date of hospital admissions, specimen collection date, symptom onset), formatted in ISO8601 standards as YYYY-MM-DD.

**report_date** Date the event was reported into the surveillance system, formatted as YYYY-MM-DD.

**count** Number of incident events (e.g. cases) occurring on the specified reference date and reported on the report date.

### Source

Created for package demonstration, made to look like the output after preprocessing the line-list data to obtain the number of incidence cases of a specific syndromic surveillance definition, indexed by the date of admission (reference date) and the date of the diagnoses being reported to the surveillance system (report date) (e.g. a reporting triangle in long format).

### See Also

Example datasets `example_downward_corr_rt`, `example_reporting_triangle`, `germany_covid19_hosp`, `syn_nssp_line_list`

---

`syn_nssp_line_list` *A synthetic dataset resembling line-list (each row is a patient) data from the United States' National Syndromic Surveillance System (NSSP) accessed via the Essence platform. All entries are synthetic, formatted to look as close to the real raw data as possible.*

---

### Description

For an example of how to produce a nowcast from this data, see `vignette("nssp_nowcast")`.

### Usage

```
syn_nssp_line_list
```

### Format

A data.frame with 25 rows and 8 columns.

**C_Processed_BioSenseID** Unique identifier for each patient

**CCDDParsed** Character string indicating primary symptoms and corresponding diagnoses codes.

**DischargeDiagnosisMDTUpdates** Character string formatted as a dictionary with indices and corresponding time stamps formatted as YYYY-MM-DD HH:MM:SS.

**DischargeDiagnosisUpdates** Character string formatted as a dictionary with indices and corresponding diagnoses codes pertaining to this diagnosis associated with that event.

**HasBeenAdmitted** Numeric indicating whether the patient was admitted (0 for no, 1 for admission).

**C_Visit_Date_Time** Date-time indicating the time stamp of the the patient registering in the emergency department, in YYYY-MM-DD HH:MM:SS format.

**c_race** Character string indicating the race/ethnicity of the patient.

**sex** Character string indicating the sex of the patient.

### Source

Created for package demonstration to provide an example of how to pre-process this dataset to obtain a reporting triangle. This is made to look like the data that one would pull directly an API to access patient-level line-list data.

## See Also

Example datasets example_downward_corr_rt, example_reporting_triangle, germany_covid19_hosp, syn_nssp_df

---

`tail.reporting_triangle` *Get last rows of a reporting_triangle*

---

## Description

Get last rows of a reporting_triangle

## Usage

```
## S3 method for class 'reporting_triangle'
tail(x, n = 6L, ...)
```

## Arguments

| | |
|---|---|
| x | A reporting_triangle object. |
| n | Integer indicating the number of rows to return. Default is 6. |
| ... | Additional arguments (not currently used). |

## Value

Last rows as a reporting_triangle.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triar truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

## Examples

```
# Get last 3 rows
tail(example_reporting_triangle, n = 3)
```

`truncate_to_delay` *Truncate reporting triangle to a specific maximum delay*

## Description

Creates a new reporting_triangle with columns filtered to include only delays from 0 to the specified maximum delay. This is useful when you want to limit the delay distribution used for estimation.

## Usage

```
truncate_to_delay(x, max_delay)
```

## Arguments

| | |
|---|---|
| `x` | A reporting_triangle object. |
| `max_delay` | Integer specifying the maximum delay to retain. Must be between 0 and the current maximum delay of the triangle. |

## Value

A new reporting_triangle object with delays 0 through max_delay.

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triar` `tail.reporting_triangle()`, `truncate_to_quantile()`, `validate_reporting_triangle()`

## Examples

```
# Truncate to delays 0-2
rt_short <- truncate_to_delay(example_downward_corr_rt, max_delay = 2)
get_max_delay(rt_short) # Returns 2
```

`truncate_to_quantile` *Truncate reporting_triangle to quantile-based maximum delay*

## Description

Automatically determines an appropriate maximum delay based on when a specified proportion of cases have been reported (CDF cutoff). This is useful for reducing computational burden when most cases are reported within a shorter delay window.

## Usage

```
truncate_to_quantile(x, p = 0.99)
```

## Arguments

| | |
|---|---|
| x | A reporting_triangle object |
| p | Numeric value between 0 and 1 indicating the quantile cutoff. For example, p = 0.99 truncates to the delay at which 99% of cases have been reported. Default is 0.99. |

## Value

A reporting_triangle object truncated to the maximum quantile delay, or the original object if no truncation is needed

## See Also

Reporting triangle construction and validation `[.reporting_triangle()`, `[<-.reporting_triangle()`, `as.data.frame.reporting_triangle()`, `as.matrix.reporting_triangle()`, `as_ChainLadder_triangle()`, `as_reporting_triangle()`, `as_reporting_triangle.data.frame()`, `as_reporting_triangle.matrix()`, `as_reporting_triangle.triangle()`, `assert_reporting_triangle()`, `get_delays_from_dates()`, `get_delays_unit()`, `get_max_delay()`, `get_mean_delay()`, `get_quantile_delay()`, `get_reference_dates()`, `get_report_dates()`, `get_reporting_structure()`, `head.reporting_triangle()`, `is_reporting_triangle()`, `new_reporting_triangle()`, `print.reporting_triangle()`, `reporting_triangle-class`, `summary.reporting_triar` `tail.reporting_triangle()`, `truncate_to_delay()`, `validate_reporting_triangle()`

## Examples

```
data_as_of_df <- syn_nssp_df[syn_nssp_df$report_date <= "2026-04-01", ]
# Create triangle, max_delay is automatically computed
rep_tri <- suppressMessages(as_reporting_triangle(data = data_as_of_df))

# Check the maximum delay in the triangle
ncol(rep_tri)

# Truncate to 99th percentile of reporting
rep_tri_trunc <- truncate_to_quantile(rep_tri, p = 0.99)
ncol(rep_tri_trunc)
```

```
# More aggressive truncation
rep_tri_trunc90 <- truncate_to_quantile(rep_tri, p = 0.90)
ncol(rep_tri_trunc90)
```

---

`truncate_to_row` *Truncate reporting triangle by removing a specified number of the last rows*

---

## Description

Removes the last `t` rows from a reporting triangle to simulate what would have been observed at an earlier reference time.

## Usage

```
truncate_to_row(reporting_triangle, t, validate = TRUE)
```

## Arguments

| | |
|---|---|
| `reporting_triangle` | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| `t` | Integer indicating the number of timepoints to truncate off the bottom of the original reporting triangle. |
| `validate` | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

`trunc_rep_tri` A `reporting_triangle` object with `t` fewer rows than the input. The class and metadata are preserved with updated reference dates.

## See Also

Retrospective data generation functions `apply_reporting_structure()`, `apply_reporting_structures()`, `truncate_to_rows()`

## Examples

```
# Generate single truncated triangle
trunc_rep_tri <- truncate_to_row(example_reporting_triangle, t = 1)
trunc_rep_tri
```

truncate_to_rows *Truncate reporting triangle by removing bottom rows*

## Description

Generates a list of retrospective reporting triangles by successively removing rows from the bottom of the original triangle. Each truncated triangle represents what would have been observed at an earlier reference time. This function truncates row(s) of the reporting triangle, removing the most recent observations (starting from the bottom of the reporting triangle).

## Usage

```
truncate_to_rows(
  reporting_triangle,
  n = nrow(reporting_triangle) - sum(is.na(rowSums(reporting_triangle))) - 1,
  validate = TRUE
)
```

## Arguments

| | |
|---|---|
| reporting_triangle | A reporting_triangle object with rows representing reference times and columns representing delays. Can be a reporting matrix or incomplete reporting matrix. Can also be a ragged reporting triangle, where multiple columns are reported for the same row (e.g., weekly reporting of daily data). |
| n | Integer indicating the number of retrospective truncated triangles to be generated, always starting from the most recent reference time. Default is to generate truncated matrices for each row up until there are insufficient rows to generate nowcasts from, where the minimum requirement is one more than the number of horizon rows (rows containing NAs). |
| validate | Logical. If TRUE (default), validates the object. Set to FALSE only when called from functions that already validated. |

## Value

trunc_rep_tri_list List of n truncated reporting_triangle objects with as many rows as available given the truncation, and the same number of columns as the input reporting_triangle.

## See Also

Retrospective data generation functions apply_reporting_structure(), apply_reporting_structures(), truncate_to_row()

## Examples

```
# Generate multiple truncated triangles
truncated_rts <- truncate_to_rows(example_reporting_triangle, n = 2)
truncated_rts[1:2]
```

## validate_reporting_triangle *Validate a reporting_triangle object*

### Description

Validate a reporting_triangle object

### Usage

```
validate_reporting_triangle(data)
```

### Arguments

| | |
|---|---|
| data | A reporting_triangle object to validate |

### Value

The validated object (invisibly) or throws error

### See Also

Reporting triangle construction and validation [.reporting_triangle(), [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triar tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile()

## [.reporting_triangle *Subset reporting_triangle objects*

### Description

Extract or replace parts of a reporting_triangle object while preserving its attributes.

### Usage

```
## S3 method for class 'reporting_triangle'
x[...]
```

### Arguments

| | |
|---|---|
| x | A reporting_triangle object |
| ... | Indices for subsetting |

## Value

A reporting_triangle object with the subset data

## See Also

Reporting triangle construction and validation [<-.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

---

# [<-.reporting_triangle

*Subset assignment for reporting_triangle objects*

---

## Description

Assignment method that allows modification of reporting_triangle values while validating the result remains a valid reporting_triangle.

## Usage

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

## Arguments

| | |
|---|---|
| x | A reporting_triangle object. |
| ... | Row and column indices for assignment. |
| value | Values to assign. |

## Value

The modified reporting_triangle object.

## See Also

Reporting triangle construction and validation [.reporting_triangle(), as.data.frame.reporting_triangle(), as.matrix.reporting_triangle(), as_ChainLadder_triangle(), as_reporting_triangle(), as_reporting_triangle.data.frame(), as_reporting_triangle.matrix(), as_reporting_triangle.triangle(), assert_reporting_triangle(), get_delays_from_dates(), get_delays_unit(), get_max_delay(), get_mean_delay(), get_quantile_delay(), get_reference_dates(), get_report_dates(), get_reporting_structure(), head.reporting_triangle(), is_reporting_triangle(), new_reporting_triangle(), print.reporting_triangle(), reporting_triangle-class, summary.reporting_triangle(), tail.reporting_triangle(), truncate_to_delay(), truncate_to_quantile(), validate_reporting_triangle()

# Index