

# Package: epinowcast (via r-universe)

July 11, 2024

**Title** Flexible Hierarchical Nowcasting

**Version** 0.3.0

**Description** Tools to enable flexible and efficient hierarchical nowcasting of right-truncated epidemiological time-series using a semi-mechanistic Bayesian model with support for a range of reporting and generative processes. Nowcasting, in this context, is gaining situational awareness using currently available observations and the reporting patterns of historical observations. This can be useful when tracking the spread of infectious disease in real-time: without nowcasting, changes in trends can be obfuscated by partial reporting or their detection may be delayed due to the use of simpler methods like truncation. While the package has been designed with epidemiological applications in mind, it could be applied to any set of right-truncated time-series count data.

**License** MIT + file LICENSE

**URL** <https://package.epinowcast.org>,  
<https://github.com/epinowcast/epinowcast/>

**BugReports** <https://github.com/epinowcast/epinowcast/issues/>

**Depends** R (>= 3.6.0)

**Imports** cli, cmdstanr, data.table, lme4, lubridate, ggplot2,  
posterior, purrr, rlang, scales, lifecycle

**Suggests** bookdown, loo (>= 2.4.1), knitr, RcppEigen, rmarkdown,  
scoringutils, spelling, testthat (>= 3.1.9), usethis, vdiff,  
withr

**VignetteBuilder** knitr

**Remotes** stan-dev/cmdstanr, stan-dev/loo

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

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

**Config/testthat/edition** 3

**Encoding** UTF-8

**Language** en-GB  
**LazyData** true  
**Roxygen** list(markdown = TRUE)  
**RoxygenNote** 7.2.3  
**SystemRequirements** CmdStan (>=2.29)  
**Repository** <https://epinowcast.r-universe.dev>  
**RemoteUrl** <https://github.com/epinowcast/epinowcast>  
**RemoteRef** v0.3.0  
**RemoteSha** 8a9cda94d0af59dfdbd5ba662167d329eb9707a3

## Contents

add_max_observed_delay . . . . .	4
add_pmfs . . . . .	5
aggregate_rolling_sum . . . . .	6
as_string_formula . . . . .	7
check_calendar_timestep . . . . .	7
check_group . . . . .	8
check_group_date_unique . . . . .	9
check_max_delay . . . . .	9
check_module . . . . .	11
check_modules_compatible . . . . .	11
check_numeric_timestep . . . . .	12
check_observation_indicator . . . . .	12
check_quantiles . . . . .	13
check_timestep . . . . .	14
check_timestep_by_date . . . . .	15
check_timestep_by_group . . . . .	15
coerce_date . . . . .	16
coerce_dt . . . . .	17
construct_re . . . . .	18
construct_rw . . . . .	19
convolution_matrix . . . . .	20
date_to_numeric_modulus . . . . .	21
enw_add_cumulative . . . . .	22
enw_add_cumulative_membership . . . . .	23
enw_add_delay . . . . .	24
enw_add_incidence . . . . .	25
enw_add_latest_obs_to_nowcast . . . . .	26
enw_add_max_reported . . . . .	27
enw_add_metaobs_features . . . . .	28
enw_add_pooling_effect . . . . .	29
enw_aggregate_cumulative . . . . .	30
enw_assign_group . . . . .	32
enw_complete_dates . . . . .	33

enw_construct_data . . . . .	35
enw_design . . . . .	37
enw_effects_metadata . . . . .	38
enw_example . . . . .	39
enw_expectation . . . . .	40
enw_extend_date . . . . .	41
enw_filter_reference_dates . . . . .	42
enw_filter_report_dates . . . . .	43
enw_fit_opts . . . . .	44
enw_flag_observed_observations . . . . .	46
enw_formula . . . . .	46
enw_formula_as_data_list . . . . .	48
enw_get_cache . . . . .	49
enw_impute_na_observations . . . . .	49
enw_incidence_to_linelist . . . . .	50
enw_latest_data . . . . .	51
enw_linelist_to_incidence . . . . .	52
enw_manual_formula . . . . .	53
enw_metadata . . . . .	55
enw_metadata_delay . . . . .	56
enw_missing . . . . .	57
enw_missing_reference . . . . .	58
enw_model . . . . .	59
enw_nowcast_samples . . . . .	60
enw_nowcast_summary . . . . .	61
enw_obs . . . . .	62
enw_one_hot_encode_feature . . . . .	63
enw_plot_nowcast_quantiles . . . . .	64
enw_plot_obs . . . . .	65
enw_plot_pp_quantiles . . . . .	66
enw_plot_quantiles . . . . .	66
enw_plot_theme . . . . .	67
enw_posterior . . . . .	68
enw_pp_summary . . . . .	69
enw_preprocess_data . . . . .	70
enw_priors_as_data_list . . . . .	72
enw_quantiles_to_long . . . . .	73
enw_reference . . . . .	74
enw_reference_by_report . . . . .	75
enw_replace_priors . . . . .	76
enw_report . . . . .	77
enw_reporting_triangle . . . . .	78
enw_reporting_triangle_to_long . . . . .	79
enw_reps_with_complete_refs . . . . .	80
enw_sample . . . . .	81
enw_score_nowcast . . . . .	81
enw_set_cache . . . . .	83
enw_simulate_missing_reference . . . . .	84

enw_stan_to_r . . . . .	85
enw_summarise_samples . . . . .	87
enw_unset_cache . . . . .	88
epinowcast . . . . .	88
extract_obs_metadata . . . . .	91
extract_sparse_matrix . . . . .	92
germany_covid19_hosp . . . . .	93
get_internal_timestep . . . . .	93
is.Date . . . . .	94
latest_obs_as_matrix . . . . .	94
parse_formula . . . . .	95
plot.epinowcast . . . . .	96
re . . . . .	97
remove_profiling . . . . .	98
remove_rw_terms . . . . .	98
rw . . . . .	99
rw_terms . . . . .	100
simulate_double_censored_pmf . . . . .	101
split_formula_to_terms . . . . .	102
stan_fns_as_string . . . . .	103
summary.epinowcast . . . . .	103
write_stan_files_no_profile . . . . .	105

---

**Index** **106**


---

 add\_max\_observed\_delay

*Add maximum observed delay*

---

**Description**

This function calculates and adds the maximum observed delay for each group and reference date in the provided dataset. It first checks the validity of the observation indicator and then computes the maximum delay. If an observation indicator is provided, it further adjusts the maximum observed delay for unobserved data to be negative 1 (indicating no maximum observed).

**Usage**

```
add_max_observed_delay(new_confirm, observation_indicator = NULL)
```

**Arguments**

**new\_confirm** A data.table containing the columns: "reference\_date", "delay", ".group", "new\_confirm", and "max\_obs\_delay". As produced by [enw\\_preprocess\\_data\(\)](#) in the new\_confirm output with the addition of the "max\_obs\_delay" column as produced by [add\\_max\\_observed\\_delay\(\)](#).

**observation\_indicator** A character string specifying the column name in new\_confirm that indicates whether an observation is observed or not. This column should be a logical vector. If NULL (default), all observations are considered observed.

**Value**

A `data.table` with the original columns of `new_confirm` and an additional "max\_obs\_delay" column representing the maximum observed delay for each group and reference date. If an observation indicator is provided, unobserved data will have a "max\_obs\_delay" value of -1.

**See Also**

Helper functions for model modules `add_pmf()`, `convolution_matrix()`, `enw_reference_by_report()`, `enw_reps_with_complete_refs()`, `extract_obs_metadata()`, `extract_sparse_matrix()`, `latest_obs_as_matrix()`, `simulate_double_censored_pmf()`

---

<code>add_pmf</code>	<i>Add probability mass functions</i>
----------------------	---------------------------------------

---

**Description**

This function allows the addition of probability mass functions (PMFs) to produce a new PMF. This is useful for example in the context of reporting delays where the PMF of the sum of two Poisson distributions is the convolution of the PMFs.

**Usage**

```
add_pmf(pmf)
```

**Arguments**

`pmf` A list of vectors describing the probability mass functions to

**Value**

A vector describing the probability mass function of the sum of the

**See Also**

Helper functions for model modules `add_max_observed_delay()`, `convolution_matrix()`, `enw_reference_by_report()`, `enw_reps_with_complete_refs()`, `extract_obs_metadata()`, `extract_sparse_matrix()`, `latest_obs_as_matrix()`, `simulate_double_censored_pmf()`

**Examples**

```
# Sample and analytical PMFs for two Poisson distributions
x <- rpois(10000, 5)
xpmf <- dpois(0:20, 5)
y <- rpois(10000, 7)
ypmf <- dpois(0:20, 7)
# Add sampled Poisson distributions up to get combined distribution
z <- x + y
# Analytical convolution of PMFs
```

```
conv_pmf <- add_pmf(list(xpmf, ypmf))
conv_cdf <- cumsum(conv_pmf)
# Empirical convolution of PMFs
cdf <- ecdf(z)(0:42)
# Compare sampled and analytical CDFs
plot(conv_cdf)
lines(cdf, col = "black")
```

---

aggregate\_rolling\_sum *Internal function to perform rolling sum aggregation*

---

### Description

This function takes a `data.table` and applies a rolling sum over a given timestep, aggregating by specified columns. It's particularly useful for aggregating observations over certain periods.

### Usage

```
aggregate_rolling_sum(dt, internal_timestep, by = NULL)
```

### Arguments

`dt` A `data.table` to be aggregated.

`internal_timestep` An integer indicating the period over which to aggregate.

`by` A character vector specifying the columns to aggregate by.

### Value

A modified `data.table` with aggregated observations.

### See Also

Utility functions [coerce\\_date\(\)](#), [coerce\\_dt\(\)](#), [date\\_to\\_numeric\\_modulus\(\)](#), [get\\_internal\\_timestep\(\)](#), [is.Date\(\)](#), [stan\\_fns\\_as\\_string\(\)](#)

---

as\_string\_formula      *Converts formulas to strings*

---

**Description**

Converts formulas to strings

**Usage**

```
as_string_formula(formula)
```

**Arguments**

formula      A model formula that may use standard fixed effects, random effects using [lme4](#) syntax (see [re\(\)](#)), and random walks defined using the [rw\(\)](#) helper function.

**Value**

A character string of the supplied formula

**See Also**

Functions used to help convert formulas into model designs [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_formula\(\)](#), [enw\\_manual\\_formula\(\)](#), [parse\\_formula\(\)](#), [remove\\_rw\\_terms\(\)](#), [re\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#), [split\\_formula\\_to\\_terms\(\)](#)

**Examples**

```
epinowcast:::as_string_formula(~ 1 + age_group)
```

---

check\_calendar\_timestep  
*Check calendar timestep*

---

**Description**

This function verifies if the difference in calendar dates in the provided observations corresponds to the provided timestep of "month".

**Usage**

```
check_calendar_timestep(dates, date_var, exact = TRUE)
```

**Arguments**

dates	Vector of Date class representing dates.
date_var	The variable in obs representing dates.
exact	Logical, if TRUE, checks if all differences exactly match the timestep. If FALSE, checks if the sum of the differences modulo the timestep equals zero. Default is TRUE.

**Value**

This function is used for its side effect of stopping if the check fails. If the check passes, the function returns invisibly.

**See Also**

Functions used for checking inputs [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check\_group

*Check observations for reserved grouping variables*

---

**Description**

Check observations for reserved grouping variables

**Usage**

```
check_group(obs)
```

**Arguments**

obs	An object that will be coerced in place, that does not contain .group, .old_group, or .new_group. These are reserved names.
-----	---

**Value**

The obs object, which will be modifiable in place.

**See Also**

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)



---

`check_group_date_unique`

*Check observations for uniqueness of grouping variables with respect to reference\_date and report\_date*

---

### Description

This function checks that the input data is stratified by `reference_date`, `report_date`, and `.group`. It does this by counting the number of observations for each combination of these variables, and throwing a warning if any combination has more than one observation.

### Usage

```
check_group_date_unique(obs)
```

### Arguments

`obs` An object that will be coerced in place, that contains `.group`, `reference_date`, and `report_date` columns.

### See Also

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

`check_max_delay`

*Check appropriateness of maximum delay*

---

### Description

Check if maximum delay specified by the user is long enough and raise potential warnings. This is achieved by computing the share of reference dates where the cumulative case count is below some aspired coverage.

### Usage

```
check_max_delay(  
  data,  
  max_delay = data$max_delay,  
  cum_coverage = 0.8,  
  maxdelay_quantile_outlier = 0.97,  
  warn = TRUE,  
  warn_internal = FALSE  
)
```

**Arguments**

data	Output from <code>enw_preprocess_data()</code> .
max_delay	The maximum number of days to model in the delay distribution. Must be an integer greater than or equal to 1. Observations with delays larger than the maximum delay will be dropped. If the specified maximum delay is too short, nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Consider what delays are realistic for your application, and when in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by <code>epinowcast</code> . If it does, your maximum delay may still be too short. Note that delays are zero indexed and so include the reference date and <code>max_delay - 1</code> other days (i.e. a <code>max_delay</code> of 1 corresponds to no delay). You can use <code>check_max_delay()</code> to check the coverage of a delay distribution for different maximum delays.
cum_coverage	The aspired percentage of cases that the maximum delay should cover. Defaults to 0.8 (80%).
maxdelay_quantile_outlier	Only reference dates sufficiently far in the past, determined based on the maximum observed delay, are included (see details). Instead of the overall maximum observed delay, a quantile of the maximum observed delay over all reference dates is used. This is more robust against outliers. Defaults to 0.97 (97%).
warn	Should a warning be issued if the cumulative case count is below <code>cum_coverage</code> for the majority of reference dates?
warn_internal	Should only be TRUE if this function is called internally by another <code>epinowcast</code> function. Then, warnings are adjusted to avoid confusing the user.

**Details**

The coverage is with respect to the maximum observed case count for the corresponding reference date. As the maximum observed case count is likely smaller than the true overall case count for not yet fully observed reference dates (due to right truncation), only reference dates that are more than the maximum observed delay ago are included. Still, because we can only use the maximum observed delay, not the unknown true maximum delay, the computed coverage values should be interpreted with care, as they are only proxies for the true coverage.

**Value**

A data.table with the share of reference dates where the cumulative case count is below `cum_coverage`, stratified by group.

**See Also**

Functions used for checking inputs `check_calendar_timestep()`, `check_group_date_unique()`, `check_group()`, `check_modules_compatible()`, `check_module()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep_by_date()`, `check_timestep_by_group()`, `check_timestep()`

**Examples**

```
pobs <- enw_example(type = "preprocessed_observations")
check_max_delay(pobs, max_delay = 20, cum_coverage = 0.8)
```

---

check_module	<i>Check a model module contains the required components</i>
--------------	--

---

**Description**

Check a model module contains the required components

**Usage**

```
check_module(module)
```

**Arguments**

module            A model module. For example [enw\\_expectation\(\)](#).

**See Also**

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check_modules_compatible	<i>Check that model modules have compatible specifications</i>
--------------------------	--

---

**Description**

Check that model modules have compatible specifications

**Usage**

```
check_modules_compatible(modules)
```

**Arguments**

modules            A list of model modules.

**See Also**

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check\_numeric\_timestep

*Check Numeric Timestep*

---

### Description

This function verifies if the difference in numeric dates in the provided observations corresponds to the provided timestep.

### Usage

```
check_numeric_timestep(dates, date_var, timestep, exact = TRUE)
```

### Arguments

dates	Vector of Date class representing dates.
date_var	The variable in obs representing dates.
timestep	Numeric timestep for date difference.
exact	Logical, if TRUE, checks if all differences exactly match the timestep. If FALSE, checks if the sum of the differences modulo the timestep equals zero. Default is TRUE.

### Value

This function is used for its side effect of stopping if the check fails. If the check passes, the function returns invisibly.

### See Also

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check\_observation\_indicator

*Check observation indicator*

---

### Description

This function verifies if the observation\_indicator within the provided new\_confirm observations is logical. The check is performed to ensure that the observation\_indicator is of the correct type.

### Usage

```
check_observation_indicator(new_confirm, observation_indicator = NULL)
```

**Arguments**

`new_confirm` A data frame containing the observations to be checked.

`observation_indicator` A character string specifying the column name in `new_confirm` that represents the observation indicator. This column should be of logical type. If NULL, no check is performed.

**Value**

This function is used for its side effect of checking the observation indicator in `new_confirm`. If the check passes, the function returns invisibly. Otherwise, it stops and returns an error message.

**See Also**

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_quantiles\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check_quantiles	<i>Check required quantiles are present</i>
-----------------	---

---

**Description**

Check required quantiles are present

**Usage**

```
check_quantiles(posterior, req_probs = c(0.5, 0.95, 0.2, 0.8))
```

**Arguments**

`posterior` A `data.table` that will be [coerce\\_dt\(\)](#)d in place; must contain quantiles identified using the q5 naming scheme.

`req_probs` A numeric vector of required probabilities. Default: `c(0.5, 0.95, 0.2, 0.8)`.

**See Also**

Functions used for checking inputs [check\\_calendar\\_timestep\(\)](#), [check\\_group\\_date\\_unique\(\)](#), [check\\_group\(\)](#), [check\\_max\\_delay\(\)](#), [check\\_modules\\_compatible\(\)](#), [check\\_module\(\)](#), [check\\_numeric\\_timestep\(\)](#), [check\\_observation\\_indicator\(\)](#), [check\\_timestep\\_by\\_date\(\)](#), [check\\_timestep\\_by\\_group\(\)](#), [check\\_timestep\(\)](#)

---

check_timestep	<i>Check timestep</i>
----------------	-----------------------

---

### Description

This function verifies if the difference in dates in the provided observations corresponds to the provided timestep. If the `exact` argument is set to `TRUE`, the function checks if all differences exactly match the timestep; otherwise, it checks if the sum of the differences modulo the timestep equals zero. If the check fails, the function stops and returns an error message.

### Usage

```
check_timestep(
  obs,
  date_var,
  timestep = "day",
  exact = TRUE,
  check_nrow = TRUE
)
```

### Arguments

<code>obs</code>	Any of the types supported by <code>data.table::as.data.table()</code> .
<code>date_var</code>	The variable in <code>obs</code> representing dates.
<code>timestep</code>	The timestep to use. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
<code>exact</code>	Logical, if <code>TRUE</code> , checks if all differences exactly match the timestep. If <code>FALSE</code> , checks if the sum of the differences modulo the timestep equals zero. Default is <code>TRUE</code> .
<code>check_nrow</code>	Logical, if <code>TRUE</code> , checks if there are at least two observations. Default is <code>TRUE</code> . If <code>FALSE</code> , the function returns invisibly if there is only one observation.

### Value

This function is used for its side effect of stopping if the check fails. If the check passes, the function returns invisibly.

### See Also

Functions used for checking inputs `check_calendar_timestep()`, `check_group_date_unique()`, `check_group()`, `check_max_delay()`, `check_modules_compatible()`, `check_module()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep_by_date()`, `check_timestep_by_group()`

---

 check\_timestep\_by\_date

*Check timestep by date*


---

### Description

This function verifies if the difference in dates within each date in the provided observations corresponds to the provided timestep. This check is performed for both `report_date` and `reference_date` and for each group in `obs`.

### Usage

```
check_timestep_by_date(obs, timestep = "day", exact = TRUE)
```

### Arguments

<code>obs</code>	Any of the types supported by <code>data.table::as.data.table()</code> .
<code>timestep</code>	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
<code>exact</code>	Logical, if TRUE, checks if all differences exactly match the timestep. If FALSE, checks if the sum of the differences modulo the timestep equals zero. Default is TRUE.

### Value

This function is used for its side effect of checking the timestep by date in `obs`. If the check passes for all dates, the function returns invisibly. Otherwise, it stops and returns an error message.

### See Also

Functions used for checking inputs `check_calendar_timestep()`, `check_group_date_unique()`, `check_group()`, `check_max_delay()`, `check_modules_compatible()`, `check_module()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep_by_group()`, `check_timestep()`

---

 check\_timestep\_by\_group

*Check timestep by group*


---

### Description

This function verifies if the difference in dates within each group in the provided observations corresponds to the provided timestep. This check is performed for the specified `date_var` and for each group in `obs`.

**Usage**

```
check_timestep_by_group(obs, date_var, timestep = "day", exact = TRUE)
```

**Arguments**

**obs** Any of the types supported by `data.table::as.data.table()`.

**date\_var** The variable in obs representing dates.

**timestep** The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.

**exact** Logical, if TRUE, checks if all differences exactly match the timestep. If FALSE, checks if the sum of the differences modulo the timestep equals zero. Default is TRUE.

**Value**

This function is used for its side effect of checking the timestep by group in obs. If the check passes for all groups, the function returns invisibly. Otherwise, it stops and returns an error message.

**See Also**

Functions used for checking inputs `check_calendar_timestep()`, `check_group_date_unique()`, `check_group()`, `check_max_delay()`, `check_modules_compatible()`, `check_module()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep_by_date()`, `check_timestep()`

---

coerce\_date

*Coerce Dates*

---

**Description**

Provides consistent coercion of inputs to `IDate` with error handling

**Usage**

```
coerce_date(dates = NULL)
```

**Arguments**

**dates** A vector-like input, which the function attempts to coerce via `data.table::as.IDate()`. Defaults to NULL.

**Details**

If any of the elements of dates cannot be coerced, this function will result in an error, indicating all indices which cannot be coerced to `IDate`.

Internal methods of `epinowcast` assume dates are represented as `IDate`.



**Value**

An [IDate](#) vector.

**See Also**

Utility functions [aggregate\\_rolling\\_sum\(\)](#), [coerce\\_dt\(\)](#), [date\\_to\\_numeric\\_modulus\(\)](#), [get\\_internal\\_timestep\(\)](#), [is.Date\(\)](#), [stan\\_fns\\_as\\_string\(\)](#)

**Examples**

```
# works
coerce_date(c("2020-05-28", "2020-05-29"))
# does not, indicates index 2 is problem
tryCatch(
  coerce_date(c("2020-05-28", "2020-o5-29")),
  error = function(e) {
    print(e)
  }
)
```

---

coerce\_dt

*Coerce data tables*


---

**Description**

Provides consistent coercion of inputs to [data.table](#) with error handling, column checking, and optional selection.

**Usage**

```
coerce_dt(
  data,
  select = NULL,
  required_cols = select,
  forbidden_cols = NULL,
  group = FALSE,
  dates = FALSE,
  copy = TRUE,
  msg_required = "The following columns are required: ",
  msg_forbidden = "The following columns are forbidden: "
)
```

**Arguments**

**data** Any of the types supported by [data.table::as.data.table\(\)](#)

**select** An optional character vector of columns to return; *unchecked* n.b. it is an error to include ".group"; use group argument for that

required_cols	An optional character vector of required columns
forbidden_cols	An optional character vector of forbidden columns
group	A logical; ensure the presence of a .group column?
dates	A logical; ensure the presence of report_date and reference_date? If TRUE (default), those columns will be coerced with <code>data.table::as.IDate()</code> .
copy	A logical; if TRUE (default), a new data.table is returned
msg_required	A character string; for required_cols-related error message
msg_forbidden	A character string; for forbidden_cols-related error message

### Details

This function provides a single-point function for getting a "local" version of data provided by the user, in the internally used data.table format. It also enables selectively copying versus not, as well as checking for the presence and/or absence of various columns.

While it is intended to address garbage in from the *user*, it does not generally attempt to address garbage in from the *developer* - e.g. if asking for overlapping required and forbidden columns (though that will lead to an always-error condition).

### Value

A data.table; the returned object will be a copy, unless copy = FALSE, in which case modifications are made in-place

### See Also

Utility functions `aggregate_rolling_sum()`, `coerce_date()`, `date_to_numeric_modulus()`, `get_internal_timestep()`, `is.Date()`, `stan_fns_as_string()`

---

construct_re	<i>Constructs random effect terms</i>
--------------	---------------------------------------

---

### Description

Constructs random effect terms

### Usage

```
construct_re(re, data)
```

### Arguments

re	A random effect as defined using <code>re()</code> which itself takes random effects specified in a model formula using the <code>lme4</code> syntax.
data	A data.frame of observations used to define the random effects. Must contain the variables specified in the <code>re()</code> term.

**Value**

A list containing the transformed data ("data"), fixed effects terms ("terms") and a `data.frame` specifying the random effect structure between these terms (effects). Note that if the specified random effect was not a factor it will have been converted into one.

**See Also**

Functions used to help convert formulas into model designs `as_string_formula()`, `construct_rw()`, `enw_formula()`, `enw_manual_formula()`, `parse_formula()`, `remove_rw_terms()`, `re()`, `rw_terms()`, `rw()`, `split_formula_to_terms()`

**Examples**

```
# Simple examples
form <- epinowcast:::parse_formula(~ 1 + (1 | day_of_week))
data <- enw_example("prepr")$metareference[[1]]
random_effect <- re(form$random[[1]])
epinowcast:::construct_re(random_effect, data)

# A more complex example
form <- epinowcast:::parse_formula(
  ~ 1 + disp + (1 + gear | cyl) + (0 + wt | am)
)
random_effect <- re(form$random[[1]])
epinowcast:::construct_re(random_effect, mtcars)

random_effect2 <- re(form$random[[2]])
epinowcast:::construct_re(random_effect2, mtcars)
```

---

 construct\_rw

*Constructs random walk terms*


---

**Description**

This function takes random walks as defined by `rw()`, produces the required additional variables (denoted using a "c" prefix and constructed using `enw_add_cumulative_membership()`), and then returns the extended `data.frame` along with the new fixed effects and the random effect structure.

**Usage**

```
construct_rw(rw, data)
```

**Arguments**

<code>rw</code>	A random walk term as defined by <code>rw()</code> .
<code>data</code>	A <code>data.frame</code> of observations used to define the random walk term. Must contain the time and grouping variables defined in the <code>rw()</code> term specified.

**Value**

A list containing the following:

- `data`: The input `data.frame` with the addition of the new variables required by the specified random walk. These are added using `enw_add_cumulative_membership()`. `-terms`: A character vector of new fixed effects terms to add to a model formula.
- `effects`: A `data.frame` describing the random effect structure of the new effects.

**See Also**

Functions used to help convert formulas into model designs `as_string_formula()`, `construct_re()`, `enw_formula()`, `enw_manual_formula()`, `parse_formula()`, `remove_rw_terms()`, `re()`, `rw_terms()`, `rw()`, `split_formula_to_terms()`

**Examples**

```
data <- enw_example("preproc")$metareference[[1]]
epinowcast:::construct_rw(rw(week), data)
epinowcast:::construct_rw(rw(week, day_of_week), data)
```

---

`convolution_matrix`      *Construct a convolution matrix*

---

**Description**

This function allows the construction of convolution matrices which can be combined with a vector of primary events to produce a vector of secondary events for example in the form of a renewal equation or to simulate reporting delays. Time-varying delays are supported as well as distribution padding (to allow for use in renewal equation like approaches).

**Usage**

```
convolution_matrix(dist, t, include_partial = FALSE)
```

**Arguments**

<code>dist</code>	A vector of list of vectors describing the distribution to be convolved as a probability mass function.
<code>t</code>	Integer value indicating the number of time steps to convolve over.
<code>include_partial</code>	Logical, defaults to FALSE. If TRUE, the convolution include partially complete secondary events.

**Value**

A matrix with each column indicating a primary event and each row indicating a secondary event.

**See Also**

Helper functions for model modules [add\\_max\\_observed\\_delay\(\)](#), [add\\_pmfs\(\)](#), [enw\\_reference\\_by\\_report\(\)](#), [enw\\_reps\\_with\\_complete\\_refs\(\)](#), [extract\\_obs\\_metadata\(\)](#), [extract\\_sparse\\_matrix\(\)](#), [latest\\_obs\\_as\\_matrix\(\)](#), [simulate\\_double\\_censored\\_pmf\(\)](#)

**Examples**

```
# Simple convolution matrix with a static distribution
convolution_matrix(c(1, 2, 3), 10)
# Include partially reported convolutions
convolution_matrix(c(1, 2, 3), 10, include_partial = TRUE)
# Use a list of distributions
convolution_matrix(rep(list(c(1, 2, 3)), 10), 10)
# Use a time-varying list of distributions
convolution_matrix(c(rep(list(c(1, 2, 3)), 10), list(c(4, 5, 6))), 11)
```

---

date\_to\_numeric\_modulus

*Convert date column to numeric and calculate its modulus with given timestep.*

---

**Description**

This function processes a date column in a `data.table`, converting it to a numeric representation and then computing the modulus with the provided timestep.

**Usage**

```
date_to_numeric_modulus(dt, date_column, timestep)
```

**Arguments**

<code>dt</code>	A <code>data.table</code> .
<code>date_column</code>	A character string representing the name of the date column in <code>dt</code> .
<code>timestep</code>	An integer representing the internal timestep.

**Value**

A modified `data.table` with two new columns: one for the numeric representation of the date minus the minimum date and another for its modulus with the timestep.

**See Also**

Utility functions [aggregate\\_rolling\\_sum\(\)](#), [coerce\\_date\(\)](#), [coerce\\_dt\(\)](#), [get\\_internal\\_timestep\(\)](#), [is.Date\(\)](#), [stan\\_fns\\_as\\_string\(\)](#)

---

enw\_add\_cumulative      *Calculate cumulative reported cases from incidence of new reports*

---

### Description

Calculate cumulative reported cases from incidence of new reports

### Usage

```
enw_add_cumulative(obs, by = NULL, copy = TRUE)
```

### Arguments

obs	A data.frame containing at least the following variables: reference date (index date of interest), report_date (report date for observations), and new_confirm (incident observations by reference and report date).
by	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling
copy	Should obs be copied (default) or modified in place?

### Value

The input data.frame with a new variable confirm.

### See Also

Data converters [enw\\_add\\_incidence\(\)](#), [enw\\_aggregate\\_cumulative\(\)](#), [enw\\_cumulative\\_to\\_incidence\(\)](#), [enw\\_incidence\\_to\\_cumulative\(\)](#), [enw\\_incidence\\_to\\_linelist\(\)](#), [enw\\_linelist\\_to\\_incidence\(\)](#)

### Examples

```
# Default reconstruct incidence
dt <- germany_covid19_hosp[location == "DE"][age_group == "00+"]
dt <- enw_add_incidence(dt)
dt <- dt[, confirm := NULL]
enw_add_cumulative(dt)

# Make use of maximum reported to calculate empirical daily reporting
enw_add_cumulative(dt)
```

---

`enw_add_cumulative_membership`*Add a cumulative membership effect to a data.frame*

---

## Description

This function adds a cumulative membership effect to a data frame. This is useful for specifying models such as random walks (using `rw()`) where these features can be used in the design matrix with the appropriate formula. Supports grouping via the optional `.group` column. Note that cumulative membership is indexed to start with zero (i.e. the first observation is assigned a cumulative membership of zero).

## Usage

```
enw_add_cumulative_membership(metaobs, feature, copy = TRUE)
```

## Arguments

<code>metaobs</code>	A data.frame with a column named <code>feature</code> that contains a numeric vector of values.
<code>feature</code>	The name of the column in <code>metaobs</code> that contains the numeric vector of values.
<code>copy</code>	Should <code>metaobs</code> be copied (default) or modified in place?

## Value

A data.frame with a new columns `cfeature$` that contain the cumulative membership effect for each value of `feature`. For example if the original `feature` was `week` (with numeric entries 1, 2, 3) then the new columns will be `cweek1`, `cweek2`, and `cweek3`.

## See Also

Functions used to formulate models [enw\\_add\\_pooling\\_effect\(\)](#), [enw\\_design\(\)](#), [enw\\_effects\\_metadata\(\)](#), [enw\\_one\\_hot\\_encode\\_feature\(\)](#)

## Examples

```
metaobs <- data.frame(week = 1:2)
enw_add_cumulative_membership(metaobs, "week")

metaobs <- data.frame(week = 1:3, .group = c(1,1,2))
enw_add_cumulative_membership(metaobs, "week")
```

---

enw_add_delay	<i>Add a delay variable to the observations</i>
---------------	---

---

## Description

This helper function takes a `data.frame` or `data.table` of observations and adds the delay (numeric, in days) between `reference_date` and `report_date` for each observation.

## Usage

```
enw_add_delay(obs, timestep = "day", copy = TRUE)
```

## Arguments

<code>obs</code>	A <code>data.frame</code> containing at least the following variables: <code>reference_date</code> (index date of interest), <code>report_date</code> (report date for observations), and <code>confirm</code> (cumulative observations by reference and report date).
<code>timestep</code>	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
<code>copy</code>	Should <code>obs</code> be copied (default) or modified in place?

## Value

A `data.table` of observations with a new column `delay`.

## See Also

Preprocessing functions [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

## Examples

```
obs <- data.frame(report_date = as.Date("2021-01-01") + -2:0)
obs$reference_date <- as.Date("2021-01-01")
enw_add_delay(obs)
```



---

enw_add_incidence	<i>Calculate incidence of new reports from cumulative reports</i>
-------------------	---

---

### Description

Calculate incidence of new reports from cumulative reports

### Usage

```
enw_add_incidence(obs, set_negatives_to_zero = TRUE, by = NULL, copy = TRUE)
```

### Arguments

obs	A data.frame containing at least the following variables: reference date (index date of interest), report_date (report date for observations), and confirm (cumulative observations by reference and report date).
set_negatives_to_zero	Logical, defaults to TRUE. Should negative counts (for calculated incidence of observations) be set to zero? Currently downstream modelling does not support negative counts and so setting must be TRUE if intending to use <a href="#">epinowcast()</a> .
by	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling
copy	Should obs be copied (default) or modified in place?

### Value

The input data.frame with a new variable new\_confirm. If max\_confirm is present in the data.frame, then the proportion reported on each day (prop\_reported) will also be added.

### See Also

Data converters [enw\\_add\\_cumulative\(\)](#), [enw\\_aggregate\\_cumulative\(\)](#), [enw\\_cumulative\\_to\\_incidence\(\)](#), [enw\\_incidence\\_to\\_cumulative\(\)](#), [enw\\_incidence\\_to\\_linelist\(\)](#), [enw\\_linelist\\_to\\_incidence\(\)](#)

### Examples

```
# Default reconstruct incidence
dt <- germany_covid19_hosp[location == "DE"][age_group == "00+"]
enw_add_incidence(dt)

# Make use of maximum reported to calculate empirical daily reporting
dt <- enw_add_max_reported(dt)
enw_add_incidence(dt)
```

---

`enw_add_latest_obs_to_nowcast`*Add latest observations to nowcast output*

---

## Description

Add the latest observations to the nowcast output. This is useful for plotting the nowcast against the latest observations.

## Usage

```
enw_add_latest_obs_to_nowcast(nowcast, obs)
```

## Arguments

<code>nowcast</code>	A data.frame of nowcast output from <a href="#">enw_nowcast_summary()</a> .
<code>obs</code>	An observation data.frame containing <code>reference_date</code> columns of the same length as the number of rows in the posterior and the most up to date observation for each date. This is used to align the posterior with the observations. The easiest source of this data is the output of latest output of <a href="#">enw_preprocess_data()</a> or <a href="#">enw_latest_data()</a> .

## Value

A data.frame of nowcast output with the latest observations added.

## See Also

Functions used for postprocessing of model fits [enw\\_nowcast\\_samples\(\)](#), [enw\\_nowcast\\_summary\(\)](#), [enw\\_posterior\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_quantiles\\_to\\_long\(\)](#), [enw\\_summarise\\_samples\(\)](#)

## Examples

```
fit <- enw_example("nowcast")
obs <- enw_example("obs")
nowcast <- summary(fit, type = "nowcast")
enw_add_latest_obs_to_nowcast(nowcast, obs)
```

---

enw\_add\_max\_reported *Add the maximum number of reported cases for each reference\_date*

---

## Description

This is a helper function which adds the maximum (in the sense of latest observed) number of reported cases for each reference\_date and computes the proportion of already reported cases for each combination of reference\_date and report\_date.

## Usage

```
enw_add_max_reported(obs, copy = TRUE)
```

## Arguments

obs	A data.frame containing at least the following variables: reference_date (index date of interest), report_date (report date for observations), and confirm (cumulative observations by reference and report date).
copy	Should obs be copied (default) or modified in place?

## Value

A data.table with new columns max\_confirm and cum\_prop\_reported. max\_confirm is the maximum number of cases reported for a certain reference\_date. cum\_prop\_reported is the proportion of cases for a certain reference\_date that are reported until a given report\_day, relative to all cases so far observed for this reference\_date.

## See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

## Examples

```
obs <- data.frame(report_date = as.Date("2021-01-01") + 0:2)
obs$reference_date <- as.Date("2021-01-01")
obs$confirm <- 1:3
enw_add_max_reported(obs)
```

---

 enw\_add\_metaobs\_features

*Add common metadata variables*


---

## Description

If not already present, annotates time series data with metadata commonly used in models: day of week, and days, weeks, and months since start of time series.

## Usage

```
enw_add_metaobs_features(
  metaobs,
  holidays = NULL,
  holidays_to = "Sunday",
  datecol = "date"
)
```

## Arguments

metaobs	Raw data, coercible via <code>data.table::as.data.table()</code> . Coerced object must have <code>Dates</code> column corresponding to <code>datecol</code> name.
holidays	a (potentially empty) vector of dates (or input coercible to such; see <code>coerce_date()</code> ). The <code>day_of_week</code> column will be set to <code>holidays_to</code> for these dates.
holidays_to	A character string to assign to holidays, when <code>holidays</code> argument non-empty. Replaces the <code>day_of_week</code> column value
datecol	The column in <code>metaobs</code> corresponding to pertinent dates.

## Details

Effects models often need to include covariates for time-based features, such as day of the week (e.g. to reflect different care-seeking and/or reporting behaviour).

This function is called from within `enw_preprocess_data()` to systematically annotate `metaobs` with these commonly used metadata, if not already present.

However, it can also be used directly on other data.

## Value

A copy of the `metaobs` input, with additional columns:

- `day_of_week`, a factor of values as output from `weekdays()` and possibly as `holiday_to` if distinct from `weekdays` values
- `day`, numeric, 0 based from start of time series
- `week`, numeric, 0 based from start of time series
- `month`, numeric, 0 based from start of time series

**See Also**

Preprocessing functions `enw_add_delay()`, `enw_add_max_reported()`, `enw_assign_group()`, `enw_complete_dates()`, `enw_construct_data()`, `enw_extend_date()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata_delay()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle_to_long()`, `enw_reporting_triangle()`

**Examples**

```
# make some example date
nat_germany_hosp <- subset(
  germany_covid19_hosp,
  location == "DE" & age_group == "80+"
)[1:40]

basemeta <- enw_add_metaobs_features(
  nat_germany_hosp,
  datecol = "report_date"
)
basemeta

# with holidays - n.b.: holidays not found are silently ignored
holidaymeta <- enw_add_metaobs_features(
  nat_germany_hosp,
  datecol = "report_date",
  holidays = c(
    "2021-04-04", "2021-04-05",
    "2021-05-01", "2021-05-13",
    "2021-05-24"
  ),
  holidays_to = "Holiday"
)
holidaymeta
subset(holidaymeta, day_of_week == "Holiday")
```

---

`enw_add_pooling_effect`

*Add a pooling effect to model design metadata*

---

**Description**

This function adds a pooling effect to the metadata returned by `enw_effects_metadata()`. It does this updating the fixed column to 0 for the effects that match the string argument and adding a new column `var_name` that is 1 for the effects that match the string argument and 0 otherwise.

**Usage**

```
enw_add_pooling_effect(effects, var_name = "sd", finder_fn = startsWith, ...)
```

**Arguments**

effects	<p>A data.table with the following columns:</p> <ul style="list-style-type: none"> <li>• effects: the name of the effect</li> <li>• fixed: a logical indicating whether the effect is fixed (1) or random (0).</li> </ul> <p>This is the output of <code>enw_effects_metadata()</code>.</p>
var_name	<p>The name of the new column that will be added to the effects data.table. This column will be 1 for the effects that match the string and 0 otherwise. Defaults to 'sd'.</p>
finder_fn	<p>A function that will be used to find the effects that match the string. Defaults to <code>startsWith()</code>. This can be any function that takes a character as its first argument (the effects\$effects column) and then any other other arguments in ... and returns a logical vector indicating whether the effects were matched.</p>
...	<p>Additional arguments to finder_fn. E.g. for the finder_fn = startsWith default, this should be prefix = "somestring".</p>

**Value**

A data.table with the following columns:

- effects: the name of the effect
- fixed: a logical indicating whether the effect is fixed (1) or random (0).
- Argument supplied to var\_name: a logical indicating whether the effect should be pooled (1) or not (0).

**See Also**

Functions used to formulate models `enw_add_cumulative_membership()`, `enw_design()`, `enw_effects_metadata()`, `enw_one_hot_encode_feature()`

**Examples**

```
data <- data.frame(a = 1:3, b = as.character(1:3), c = c(1,1,2))
design <- enw_design(a ~ b + c, data)$design
effects <- enw_effects_metadata(design)
enw_add_pooling_effect(effects, prefix = "b")
```

---

enw\_aggregate\_cumulative

*Aggregate observations over a given timestep for both report and reference dates.*

---

## Description

This function aggregates observations over a specified timestep, ensuring alignment on the same day of week for report and reference dates. It is useful for aggregating data to a weekly timestep, for example which may be desirable if testing using a weekly timestep or if you are very concerned about runtime. Note that the start of the timestep will be determined by `min_date` + a single timestep (i.e. the first timestep will be "2022-10-23" if the minimum reference date is "2022-10-16").

## Usage

```
enw_aggregate_cumulative(  
  obs,  
  timestep = "day",  
  by = NULL,  
  min_reference_date = min(obs$reference_date, na.rm = TRUE),  
  copy = TRUE  
)
```

## Arguments

- |                                 |  |
|---------------------------------|--|
| <code>obs</code>                | An object coercible to a <code>data.table</code> (such as a <code>data.frame</code> ) which must have a <code>new_confirm</code> numeric column, and <code>report_date</code> and <code>reference_date</code> date columns. The input must have a timestep of a day and be complete. See <a href="#">enw_complete_dates()</a> for more information. If NA values are present in the <code>confirm</code> column then these will be set to zero before aggregation this may not be desirable if this missingness is meaningful.               |
| <code>timestep</code>           | The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.   |
| <code>by</code>                 | A character vector of variables to also aggregate by (i.e. as well as using the <code>reference_date</code> and <code>report_date</code> ). If not supplied then the function will aggregate by just the <code>reference_date</code> and <code>report_date</code> .  |
| <code>min_reference_date</code> | The minimum reference date to start the aggregation from. Note that the timestep will start from the minimum reference date + a single time step (i.e. the first timestep will be "2022-10-23" if the minimum reference date is "2022-10-16"). The default is the minimum reference date in the <code>obs</code> object. Other sensible values would be the minimum report date in the <code>obs</code> object + 1 day if reporting is already weekly and you wish to ensure that the timestep of the output matches the reporting timestep. |
| <code>copy</code>               | Should <code>obs</code> be copied (default) or modified in place?  |

## Value

A `data.table` with aggregated observations.

**See Also**

Data converters [enw\\_add\\_cumulative\(\)](#), [enw\\_add\\_incidence\(\)](#), [enw\\_cumulative\\_to\\_incidence\(\)](#), [enw\\_incidence\\_to\\_cumulative\(\)](#), [enw\\_incidence\\_to\\_linelist\(\)](#), [enw\\_linelist\\_to\\_incidence\(\)](#)

**Examples**

```
nat_hosp <- germany_covid19_hosp[location == "DE"][age_group == "00+"]
enw_aggregate_cumulative(nat_hosp, timestep = "week")
```

---

<code>enw_assign_group</code>	<i>Assign a group to each row of a data.table</i>
-------------------------------	---

---

**Description**

Assign a group to each row of a `data.table`. If `by` is specified, then each unique combination of the columns in `by` will be assigned a unique group. If `by` is not specified, then all rows will be assigned to the same group.

**Usage**

```
enw_assign_group(obs, by = NULL, copy = TRUE)
```

**Arguments**

<code>obs</code>	A <code>data.table</code> or <code>data.frame</code> without a <code>.group</code> column.
<code>by</code>	A character vector of column names to group by. Defaults to an empty vector.
<code>copy</code>	A logical; make a copy (default) of <code>obs</code> or modify it in place?

**Value**

A `data.table` with a `.group` column added ordered by `.group` and the existing key of `obs`.

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

**Examples**

```
obs <- data.frame(x = 1:3, y = 1:3)
enw_assign_group(obs)
enw_assign_group(obs, by = "x")
```



---

enw\_complete\_dates      *Complete missing reference and report dates*

---

## Description

Ensures that all reference and report dates are present for all groups based on the maximum and minimum dates found in the data. This function may be of use to users when preprocessing their data. In general all features that you may consider using as grouping variables or as covariates need to be included in the by variable.

## Usage

```
enw_complete_dates(  
  obs,  
  by = NULL,  
  max_delay,  
  min_date = min(obs$reference_date, na.rm = TRUE),  
  max_date = max(obs$report_date, na.rm = TRUE),  
  timestep = "day",  
  missing_reference = TRUE,  
  completion_beyond_max_report = FALSE,  
  flag_observation = FALSE  
)
```

## Arguments

- |           |  |
|-----------|--|
| obs       | A data.frame containing at least the following variables: reference date (index date of interest), report_date (report date for observations), and confirm (cumulative observations by reference and report date).   |
| by        | A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling   |
| max_delay | The maximum number of days to model in the delay distribution. Must be an integer greater than or equal to 1. Observations with delays larger than the maximum delay will be dropped. If the specified maximum delay is too short, nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Consider what delays are realistic for your application, and when in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by epinowcast. If it does, your maximum delay may still be too short. Note that delays are zero indexed and so include the reference date and max_delay - 1 other days (i.e. a max_delay of 1 corresponds to no delay). You can use <a href="#">check_max_delay()</a> to check the coverage of a delay distribution for different maximum delays. |

min_date	The minimum date to include in the data. Defaults to the minimum reference date found in the data.
max_date	The maximum date to include in the data. Defaults to the maximum report date found in the data.
timestep	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
missing_reference	Logical, should entries for cases with missing reference date be completed as well?, Default: TRUE
completion_beyond_max_report	Logical, should entries be completed beyond the maximum date found in the data? Default: FALSE
flag_observation	Logical, should observations that have been imputed as missing be flagged as not observed?. Makes use of <a href="#">enw_flag_observed_observations()</a> to add a .observed logical vector which indicates if observations have been imputed. This vector can then be passed to the observation_indicator argument of <a href="#">enw_obs()</a> to control if these observations are used in the likelihood. Default: FALSE

### Value

A data.table with completed entries for all combinations of reference dates, groups and possible report dates.

### See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

### Examples

```
obs <- data.frame(
  report_date = c("2021-10-01", "2021-10-03"), reference_date = "2021-10-01",
  confirm = 1
)
enw_complete_dates(obs)

# Allow completion beyond the maximum date found in the data
enw_complete_dates(obs, completion_beyond_max_report = TRUE, max_delay = 10)
```

---

enw\_construct\_data      *Construct preprocessed data*


---

## Description

This function is used internally by [enw\\_preprocess\\_data\(\)](#) to combine various pieces of processed observed data into a single object. It is exposed to the user in order to allow for modular data preprocessing though this is not currently recommended. See documentation and code of [enw\\_preprocess\\_data\(\)](#) for more on the expected inputs.

## Usage

```
enw_construct_data(
  obs,
  new_confirm,
  latest,
  missing_reference,
  reporting_triangle,
  metareport,
  metareference,
  metadelay,
  max_delay,
  timestep,
  by
)
```

## Arguments

obs	Observations with the addition of empirical reporting proportions and and restricted to the specified maximum delay.
new_confirm	Incidence of notifications by reference and report date. Empirical reporting distributions are also added.
latest	The latest available observations.
missing_reference	A data.frame of reported observations that are missing the reference date.
reporting_triangle	Incident observations by report and reference date in the standard reporting triangle matrix format.
metareport	Metadata for report dates.
metareference	Metadata reference dates derived from observations.
metadelay	Metadata for reporting delays produced using <a href="#">enw_metadata_delay()</a> .
max_delay	Maximum delay to be modelled by epinowcast.

timestep	The timestep to used in the process model (i.e. the reference date model). This can be a string ("day", "week", "month") or a numeric whole number representing the number of days. If your data does not have this timestep then you may wish to make use of <a href="#">enw_aggregate_cumulative()</a> to aggregate your data to the desired timestep.
by	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling

### Value

A data.table containing processed observations as a series of nested data.frames as well as variables containing metadata. These are:

- obs: (observations with the addition of empirical reporting proportions and restricted to the specified maximum delay).
- new\_confirm: Incidence of notifications by reference and report date. Empirical reporting distributions are also added.
- latest: The latest available observations.
- missing\_reference: Observations missing reference dates.
- reporting\_triangle: Incident observations by report and reference date in the standard reporting triangle matrix format.
- metareference: Metadata reference dates derived from observations.
- metrareport: Metadata for report dates.
- metadelay: Metadata for reporting delays produced using [enw\\_metadata\\_delay\(\)](#).
- max\_delay: Maximum delay to be modelled by epinowcast.
- time: Numeric, number of timepoints in the data.
- snapshots: Numeric, number of available data snapshots to use for nowcasting.
- groups: Numeric, Number of groups/strata in the supplied observations (set using by).
- max\_date: The maximum available report date.

### See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

### Examples

```
pobs <- enw_example("preprocessed")
enw_construct_data(
  obs = pobs$obs[[1]],
  new_confirm = pobs$new_confirm[[1]],
  latest = pobs$latest[[1]],
```

```

missing_reference = pobs$missing_reference[[1]],
reporting_triangle = pobs$reporting_triangle[[1]],
metareport = pobs$metareport[[1]],
metareference = pobs$metareference[[1]],
metadelay = pobs$metadelay[[1]],
max_delay = pobs$max_delay,
timestep = pobs$timestep[[1]],
by = c()
)

```

---

enw\_design

*A helper function to construct a design matrix from a formula*


---

## Description

This function is a wrapper around `stats::model.matrix()` that can optionally return a sparse design matrix defined as the unique number of rows in the design matrix and an index vector that allows the full design matrix to be reconstructed. This is useful for models that have many repeated rows in the design matrix and that are computationally expensive to fit. This function also allows for the specification of contrasts for categorical variables.

## Usage

```
enw_design(formula, data, no_contrasts = FALSE, sparse = TRUE, ...)
```

## Arguments

formula	An R formula.
data	A data.frame containing the variables in the formula.
no_contrasts	A vector of variable names that should not be converted to contrasts. If <code>no_contrasts = FALSE</code> then all categorical variables will use contrasts. If <code>no_contrasts = TRUE</code> then no categorical variables will use contrasts.
sparse	Logical, if TRUE return a sparse design matrix. Defaults to TRUE.
...	Arguments passed on to <code>stats::model.matrix</code> object an object of an appropriate class. For the default method, a model formula or a terms object.

## Value

A list containing the formula, the design matrix, and the index.

## See Also

Functions used to formulate models `enw_add_cumulative_membership()`, `enw_add_pooling_effect()`, `enw_effects_metadata()`, `enw_one_hot_encode_feature()`

## Examples

```
data <- data.frame(a = 1:3, b = as.character(1:3), c = c(1,1,2))
enw_design(a ~ b + c, data)
enw_design(a ~ b + c, data, no_contrasts = TRUE)
enw_design(a ~ b + c, data, no_contrasts = c("b"))
enw_design(a ~ c, data, sparse = TRUE)
enw_design(a ~ c, data, sparse = FALSE)
```

---

enw\_effects\_metadata *Extracts metadata from a design matrix*

---

## Description

This function extracts metadata from a design matrix and returns a `data.table` with the following columns:

- `effects`: the name of the effect
- `fixed`: a logical indicating whether the effect is fixed (1) or random (0).

It automatically drops the intercept (defined as "(Intercept)").

This function is useful for constructing a model design object for random effects when used in combination with `enw_add_pooling_effect`.

## Usage

```
enw_effects_metadata(design)
```

## Arguments

`design` A design matrix as returned by `stats::model.matrix()`.

## Value

A `data.table` with the following columns:

- `effects`: the name of the effect
- `fixed`: a logical indicating whether the effect is fixed (1) or random (0)

## See Also

Functions used to formulate models `enw_add_cumulative_membership()`, `enw_add_pooling_effect()`, `enw_design()`, `enw_one_hot_encode_feature()`

## Examples

```
data <- data.frame(a = 1:3, b = as.character(1:3), c = c(1,1,2))
design <- enw_design(a ~ b + c, data)$design
enw_effects_metadata(design)
```

---

enw_example	<i>Load a package example</i>
-------------	-------------------------------

---

## Description

Loads examples of nowcasts produced using example scripts. Used to streamline examples, in package tests and to enable users to explore package functionality without needing to install `cmdstanr`.

## Usage

```
enw_example(  
  type = c("nowcast", "preprocessed_observations", "observations", "script")  
)
```

## Arguments

`type` A character string indicating the example to load. Supported options are

- "nowcast", for `epinowcast()` applied to `germany_covid19_hosp`
- "preprocessed\_observations", for `enw_preprocess_data()` applied to `germany_covid19_hosp`
- "observations", for `enw_latest_data()` applied to `germany_covid19_hosp`
- "script", the code used to generate these examples.

## Value

Depending on `type`, a `data.table` of the requested output OR the file name(s) to generate these outputs (`type = "script"`)

## See Also

Package data sets `germany_covid19_hosp`

## Examples

```
# Load the nowcast  
enw_example(type = "nowcast")  
  
# Load the preprocessed observations  
enw_example(type = "preprocessed_observations")  
  
# Load the latest observations  
enw_example(type = "observations")  
  
# Load the script used to generate these examples  
# Optionally source this script to regenerate the example  
readLines(enw_example(type = "script"))
```

---

enw\_expectation      *Expectation model module*

---

## Description

Expectation model module

## Usage

```
enw_expectation(
  r = ~0 + (1 | day:.group),
  generation_time = 1,
  observation = ~1,
  latent_reporting_delay = 1,
  data,
  ...
)
```

## Arguments

- r** A formula (as implemented in [enw\\_formula\(\)](#)) describing the generative process used for expected incidence. This can use features defined by reference date as defined in [metareference](#) as produced by [enw\\_preprocess\\_data\(\)](#). By default this is set to use a daily random effect by group. This parameterisation is highly flexible and so may not be the most appropriate choice when data is sparsely reported or reporting delays are substantially. These settings an alternative could be a group specific weekly random walk (specified as `rw(week, by = .group)`).
- generation\_time** A numeric vector that sums to 1 and defaults to 1. Describes the weighting to apply to previous generations (i.e as part of a renewal equation). When set to 1 (the default) this corresponds to modelling the daily growth rate.
- observation** A formula (as implemented in [enw\\_formula\(\)](#)) describing the modifiers used to adjust expected observations. This can use features defined by reference date as defined in [metareference](#) as produced by [enw\\_preprocess\\_data\(\)](#). By default no modifiers are used but a common choice might be to adjust for the day of the week. Note as the baseline is no modification an intercept is always used and it is set to 0.
- latent\_reporting\_delay** A numeric vector that defaults to 1. Describes the weighting to apply to past and current latent expected observations (from most recent to least). This can be used both to convolve based on some assumed reporting delay and to rescale observations (by multiplying a probability mass function by some fraction) to account ascertainment etc. A list of PMFs can be provided to allow for time-varying PMFs. This should be the same length as the modelled time period plus the length of the generation time if supplied.



`data` Output from `enw_preprocess_data()`.

`...` Additional parameters passed to `enw_add_metaobs_features()`. The same arguments as passed to `enw_preprocess_data()` should be used here.

### Value

A list containing the supplied formulas, data passed into a list describing the models, a `data.frame` describing the priors used, and a function that takes the output data and priors and returns a function that can be used to sample from a tightened version of the prior distribution.

### See Also

Model modules `enw_fit_opts()`, `enw_missing()`, `enw_obs()`, `enw_reference()`, `enw_report()`

### Examples

```
enw_expectation(data = enw_example("preprocessed"))
```

---

<code>enw_extend_date</code>	<i>Extend a time series with additional dates</i>
------------------------------	---

---

### Description

Extend a time series with additional dates. This is useful when extending the report dates of a time series to include future dates for nowcasting purposes or to include additional dates for backcasting when using a renewal process as the expectation model.

### Usage

```
enw_extend_date(
  metaobs,
  days = 20,
  direction = c("end", "start"),
  timestep = "day"
)
```

### Arguments

`metaobs` A `data.frame` with a date column.

`days` Number of days to add to the time series. Defaults to 20.

`direction` Should new dates be added at the beginning or end of the data. Default is "end" with "start" also available.

`timestep` The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.

## Value

A `data.table` with the same columns as `metaobs` but with additional rows for each date in the range of `date` to `date + days` (or `date - days` if `direction = "start"`). An additional variable `observed` is added with a value of `FALSE` for all new dates and `TRUE` for all existing dates.

## See Also

Preprocessing functions `enw_add_delay()`, `enw_add_max_reported()`, `enw_add_metaobs_features()`, `enw_assign_group()`, `enw_complete_dates()`, `enw_construct_data()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata_delay()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle_to_long()`, `enw_reporting_triangle()`

## Examples

```
metaobs <- data.frame(date = as.Date("2021-01-01") + 0:4)
enw_extend_date(metaobs, days = 2)
enw_extend_date(metaobs, days = 2, direction = "start")
```

---

`enw_filter_reference_dates`

*Filter by reference dates*

---

## Description

This is a helper function which allows users to filter datasets by reference date. This is useful, for example, when evaluating nowcast performance against fully observed data. Users may wish to combine this function with `enw_filter_report_dates()`. Note that by definition it is assumed that report dates must be equal or greater than the corresponding reference date (i.e a report cannot happen before the event being reported occurs). This means that this function will also filter out any report dates that are earlier than their corresponding reference date.

## Usage

```
enw_filter_reference_dates(  
  obs,  
  earliest_date,  
  include_days,  
  latest_date,  
  remove_days  
)
```

**Arguments**

obs	A data.frame; must have report_date and reference_date columns.
earliest_date	earliest reference date to include in the data set
include_days	if earliest_date is not given, the number of reference dates to include, ending with the latest reference date included (determined by latest_date or remove_days).
latest_date	Date, the latest reference date to include in the returned dataset.
remove_days	Integer, if latest_date is not given, the number of reference dates to remove, starting from the latest date included.

**Value**

A data.table filtered by report date

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

**Examples**

```
# Filter by date
enw_filter_reference_dates(
  germany_covid19_hosp,
  earliest_date = "2021-09-01",
  latest_date = "2021-10-01"
)
#
# Filter by days
enw_filter_reference_dates(
  germany_covid19_hosp,
  include_days = 10, remove_days = 10
)
```

---

enw\_filter\_report\_dates

*Filter by report dates*

---

**Description**

This is a helper function which allows users to create truncated data sets at past time points from a given larger data set. This is useful when evaluating nowcast performance against fully observed data. Users may wish to combine this function with [enw\\_filter\\_reference\\_dates\(\)](#).

**Usage**

```
enw_filter_report_dates(obs, latest_date, remove_days)
```

**Arguments**

obs	A data.frame; must have report_date and reference_date columns.
latest_date	Date, the latest report date to include in the returned dataset.
remove_days	Integer, if latest_date is not given, the number of report dates to remove, starting from the latest date included.

**Value**

A data.table filtered by report date

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

**Examples**

```
# Filter by date
enw_filter_report_dates(germany_covid19_hosp, latest_date = "2021-09-01")

# Filter by days
enw_filter_report_dates(germany_covid19_hosp, remove_days = 10)
```

---

enw\_fit\_opts

*Format model fitting options for use with stan*


---

**Description**

Format model fitting options for use with stan

**Usage**

```
enw_fit_opts(
  sampler = epinowcast::enw_sample,
  nowcast = TRUE,
  pp = FALSE,
  likelihood = TRUE,
  likelihood_aggregation = c("snapshots", "groups"),
  threads_per_chain = 1L,
```

```

    debug = FALSE,
    output_loglik = FALSE,
    ...
)

```

## Arguments

sampler	A function that creates an object that be used to extract posterior samples from the specified model. By default this is <code>enw_sample()</code> which makes use of <code>cmdstanr::sample()</code> .
nowcast	Logical, defaults to TRUE. Should a nowcast be made using posterior predictions of the unobserved future reported notifications.
pp	Logical, defaults to FALSE. Should posterior predictions be made for observed data. Useful for evaluating the performance of the model.
likelihood	Logical, defaults to TRUE. Should the likelihood be included in the model
likelihood_aggregation	Character string, aggregation over which stratify the likelihood when <code>threads_per_chain</code> is greater than 1; enforced by <code>base::match.arg()</code> . Currently supported options: <ul style="list-style-type: none"> <li>• "snapshots" which aggregates over report dates and groups (i.e the lowest level that observations are reported at),</li> <li>• "groups" which aggregates across user defined groups.</li> </ul> Note that some model modules override this setting depending on model requirements. For example, the <code>enw_missing()</code> module model forces "groups" option. Generally, Users should typically want the default "snapshots" aggregation.
threads_per_chain	Integer, defaults to 1. The number of threads to use within each MCMC chain. If this is greater than 1 then components of the likelihood will be calculated in parallel within each chain.
debug	Logical, defaults to FALSE. Should within model debug information be returned.
output_loglik	Logical, defaults to FALSE. Should the log-likelihood be output. Disabling this will speed up fitting if evaluating the model fit is not required.
...	Additional arguments to pass to the fitting function being used by <code>epinowcast()</code> . By default this will be <code>enw_sample()</code> and so <code>cmdstanr</code> options should be used.

## Value

A list containing the specified sampler function, data as a list specifying the fitting options to use, and additional arguments to pass to the sampler function when it is called.

## See Also

Model modules `enw_expectation()`, `enw_missing()`, `enw_obs()`, `enw_reference()`, `enw_report()`

## Examples

```

# Default options along with settings to pass to enw_sample
enw_fit_opts(iter_sampling = 1000, iter_warmup = 1000)

```

---

enw\_flag\_observed\_observations  
*Flag observed observations*

---

### Description

Flags observations based on the 'confirm' column. If the '.observed' column does not exist, it is created. Observations are flagged as observed (TRUE) if 'confirm' is not NA.

### Usage

```
enw_flag_observed_observations(obs, copy = TRUE)
```

### Arguments

obs                    A data.frame with at least a have confirm column.  
copy                    A logical; if TRUE (the default) creates a copy; otherwise, modifies obs in place.

### Value

A data.table with an additional column '.observed' indicating observed observations.

### See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

### Examples

```
dt <- data.frame(id = 1:3, confirm = c(NA, 1, 2))
enw_flag_observed_observations(dt)
```

---

enw\_formula                    *Define a model using a formula interface*

---

### Description

This function allows models to be defined using a flexible formula interface that supports fixed effects, random effects (using [lme4](#) syntax). Note that the returned fixed effects design matrix is sparse and so the index supplied is required to link observations to the appropriate design matrix row.

**Usage**

```
enw_formula(formula, data, sparse = TRUE)
```

**Arguments**

formula	A model formula that may use standard fixed effects, random effects using <a href="#">lme4</a> syntax (see <a href="#">re()</a> ), and random walks defined using the <a href="#">rw()</a> helper function.
data	A data.frame of observations. It must include all variables used in the supplied formula.
sparse	Logical, defaults to TRUE. Should the fixed effects design matrix be sparsely defined.

**Value**

A list containing the following:

- formula: The user supplied formula
- parsed\_formula: The formula as parsed by [parse\\_formula\(\)](#)
- extended\_formula: The flattened version of the formula with both user supplied terms and terms added for the user supplied complex model components.
- fixed: A list containing the fixed effect formula, sparse design matrix, and the index linking the design matrix with observations.
- random: A list containing the random effect formula, sparse design matrix, and the index linking the design matrix with random effects.

**See Also**

Functions used to help convert formulas into model designs [as\\_string\\_formula\(\)](#), [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_manual\\_formula\(\)](#), [parse\\_formula\(\)](#), [remove\\_rw\\_terms\(\)](#), [re\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#), [split\\_formula\\_to\\_terms\(\)](#)

**Examples**

```
# Use meta data for references dates from the Germany COVID-19
# hospitalisation data.
obs <- enw_filter_report_dates(
  germany_covid19_hosp[location == "DE"],
  remove_days = 40
)
obs <- enw_filter_reference_dates(obs, include_days = 40)
pobs <- enw_preprocess_data(
  obs, by = c("age_group", "location"), max_delay = 20
)
data <- pobs$metareference[[1]]

# Model with fixed effects for age group
enw_formula(~ 1 + age_group, data)

# Model with random effects for age group
```

```

enw_formula(~ 1 + (1 | age_group), data)

# Model with a random effect for age group and a random walk
enw_formula(~ 1 + (1 | age_group) + rw(week), data)

# Model defined without a sparse fixed effects design matrix
enw_formula(~1, data[1:20, ])

# Model using an interaction in the right hand side of a random effect
# to specify an independent random effect per strata.
enw_formula(~ (1 + day | week:month), data = data)

```

---

```
enw_formula_as_data_list
```

*Format formula data for use with stan*

---

## Description

Format formula data for use with stan

## Usage

```
enw_formula_as_data_list(formula, prefix, drop_intercept = FALSE)
```

## Arguments

formula	The output of <a href="#">enw_formula()</a> .
prefix	A character string indicating variable label to use as a prefix.
drop_intercept	Logical, defaults to FALSE. Should the intercept be included as a fixed effect or excluded. This is used internally in model modules where an intercept must be present/absent.

## Value

A list defining the model formula. This includes:

- `prefix_fintercept`: Is an intercept present for the fixed effects design matrix.
- `prefix_fdesign`: The fixed effects design matrix
- `prefix_fnrow`: The number of rows of the fixed design matrix
- `prefix_findex`: The index linking design matrix rows to observations
- `prefix_fnindex`: The length of the index
- `prefix_fncol`: The number of columns (i.e effects) in the fixed effect design matrix (minus 1 if `drop_intercept = TRUE`).
- `prefix_rdesign`: The random effects design matrix
- `prefix_rncol`: The number of columns (i.e random effects) in the random effect design matrix (minus 1 as the intercept is dropped).



**See Also**

Functions used to help convert models into the format required for stan [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

**Examples**

```
f <- enw_formula(~ 1 + (1 | cyl), mtcars)
enw_formula_as_data_list(f, "mtcars")

# A missing formula produces the default list
enw_formula_as_data_list(prefix = "missing")
```

---

enw_get_cache	<i>Retrieve Stan cache location</i>
---------------	-------------------------------------

---

**Description**

Retrieves the user set cache location for Stan models. This path can be set through the `enw_cache_location` function call. If no environmental variable is available the output from `tempdir()` will be returned.

**Usage**

```
enw_get_cache()
```

**Value**

A string representing the file path for the cache location

**See Also**

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

---

enw_impute_na_observations	<i>Impute NA observations</i>
----------------------------	-------------------------------

---

**Description**

Imputes NA values in the 'confirm' column. NA values are replaced with the last available observation or 0.

**Usage**

```
enw_impute_na_observations(obs, by = NULL, copy = TRUE)
```

**Arguments**

`obs` A `data.frame` with at least 'confirm' and 'reference\_date' columns.  
`by` A character vector of column names to group by. Defaults to an empty vector.  
`copy` A logical; if TRUE (the default) creates a copy; otherwise, modifies `obs` in place.

**Value**

A `data.table` with imputed 'confirm' column where NA values have been replaced with zero.

**See Also**

Preprocessing functions `enw_add_delay()`, `enw_add_max_reported()`, `enw_add_metaobs_features()`, `enw_assign_group()`, `enw_complete_dates()`, `enw_construct_data()`, `enw_extend_date()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_latest_data()`, `enw_metadata_delay()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle_to_long()`, `enw_reporting_triangle()`

**Examples**

```
dt <- data.frame(
  id = 1:3, confirm = c(NA, 1, 2),
  reference_date = as.Date("2021-01-01")
)
enw_impute_na_observations(dt)
```

---

enw\_incidence\_to\_linelist

*Convert Aggregate Counts (Incidence) to a Line List*

---

**Description**

This function takes a `data.table` of aggregate counts or something coercible to a `data.table` (such as a `data.frame`) and converts it to a line list where each row represents a case.

**Usage**

```
enw_incidence_to_linelist(
  obs,
  reference_date = "reference_date",
  report_date = "report_date"
)
```

**Arguments**

obs	An object coercible to a <code>data.table</code> (such as a <code>data.frame</code> ) which must have a <code>new_confirm</code> column.
reference_date	A character string of the variable name to use for the <code>reference_date</code> in the line list. The default is "reference_date".
report_date	A character string of the variable name to use for the <code>report_date</code> in the line list. The default is "report_date".

**Value**

A `data.table` with the following variables: `id`, `reference_date`, `report_date`, and any other variables in the `obs` object. Rows in `obs` will be duplicated based on the `new_confirm` column. `reference_date` and `report_date` may be renamed if `reference_date` and `report_date` are supplied.

**See Also**

Data converters [enw\\_add\\_cumulative\(\)](#), [enw\\_add\\_incidence\(\)](#), [enw\\_aggregate\\_cumulative\(\)](#), [enw\\_cumulative\\_to\\_incidence\(\)](#), [enw\\_incidence\\_to\\_cumulative\(\)](#), [enw\\_linelist\\_to\\_incidence\(\)](#)

**Examples**

```
incidence <- enw_add_incidence(germany_covid19_hosp)
incidence <- enw_filter_reference_dates(
  incidence[location == "DE"], include_days = 10
)
enw_incidence_to_linelist(incidence, reference_date = "onset_date")
```

---

enw_latest_data	<i>Filter observations to the latest available reported</i>
-----------------	---

---

**Description**

Filter observations for the latest available reported data for each reference date. Note this is not the same as filtering for the maximum report date in all cases as data may only be updated up to some maximum number of days.

**Usage**

```
enw_latest_data(obs)
```

**Arguments**

obs	A <code>data.frame</code> ; must have <code>report_date</code> and <code>reference_date</code> columns.
-----	---

**Value**

A `data.table` of observations filtered for the latest available data for each reference date.

**See Also**

Preprocessing functions `enw_add_delay()`, `enw_add_max_reported()`, `enw_add_metaobs_features()`, `enw_assign_group()`, `enw_complete_dates()`, `enw_construct_data()`, `enw_extend_date()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_metadata_delay()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle_to_long()`, `enw_reporting_triangle()`

**Examples**

```
# Filter for latest reported data
enw_latest_data(germany_covid19_hosp)
```

---

```
enw_linelist_to_incidence
```

*Convert a Line List to Aggregate Counts (Incidence)*

---

**Description**

This function takes a line list (i.e. tabular data where each row represents a case) and aggregates to a count (`new_confirm`) of cases by user-specified `reference_dates` and `report_dates`. This enables the use of `enw_preprocess_data()` and other `epinowcast()` preprocessing functions.

**Usage**

```
enw_linelist_to_incidence(
  linelist,
  reference_date = "reference_date",
  report_date = "report_date",
  by = NULL,
  max_delay,
  completion_beyond_max_report = FALSE,
  copy = TRUE
)
```

**Arguments**

<code>linelist</code>	An object coercible to a <code>data.table</code> (such as a <code>data.frame</code> ) where each row represents a case. Must contain at least two date variables or variables that can be coerced to dates.
<code>reference_date</code>	A date or a variable that can be coerced to a date that represents the date of interest for the case. For example, if the <code>reference_date</code> is the date of symptom onset then the <code>new_confirm</code> will be the number of new cases reported (based on <code>report_date</code> ) on each day that had onset on that day. The default is "reference_date".
<code>report_date</code>	A date or a variable that can be coerced to a date that represents the date the case was reported. The default is "report_date".

by	A character vector of variables to also aggregate by (i.e. as well as using the reference_date and report_date). If not supplied then the function will aggregate by just the reference_date and report_date.
max_delay	The maximum number of days between the reference_date and the report_date. If not supplied then the function will use the maximum number of days between the reference_date and the report_date in the linelist. If the max_delay is less than the maximum number of days between the reference_date and the report_date in the linelist then the function will use this value instead and inform the user.
completion_beyond_max_report	Logical, should entries be completed beyond the maximum date found in the data? Default: FALSE
copy	Should obs be copied (default) or modified in place?

**Value**

A data.table with the following variables: reference\_date, report\_date, new\_confirm, confirm, delay, and any variables specified in by.

**See Also**

Data converters [enw\\_add\\_cumulative\(\)](#), [enw\\_add\\_incidence\(\)](#), [enw\\_aggregate\\_cumulative\(\)](#), [enw\\_cumulative\\_to\\_incidence\(\)](#), [enw\\_incidence\\_to\\_cumulative\(\)](#), [enw\\_incidence\\_to\\_linelist\(\)](#)

**Examples**

```
linelist <- data.frame(
  onset_date = as.Date(c("2021-01-02", "2021-01-03", "2021-01-02")),
  report_date = as.Date(c("2021-01-03", "2021-01-05", "2021-01-04"))
)
enw_linelist_to_incidence(linelist, reference_date = "onset_date")

# Specify a custom maximum delay and allow completion beyond the maximum
# observed delay
enw_linelist_to_incidence(
  linelist, reference_date = "onset_date", max_delay = 5,
  completion_beyond_max_report = TRUE
)
```

---

enw\_manual\_formula      *Define a model manually using fixed and random effects*

---

**Description**

For most typical use cases [enw\\_formula\(\)](#) should provide sufficient flexibility to allow models to be defined. However, there may be some instances where more manual model specification is required. This function supports this by allowing the user to supply vectors of fixed, random, and customised random effects (where they are not first treated as fixed effect terms). Prior to 1.0.0 this was the main interface for specifying models and it is still used internally to handle some parts of the model specification process.

**Usage**

```
enw_manual_formula(
  data,
  fixed = NULL,
  random = NULL,
  custom_random = NULL,
  no_contrasts = FALSE,
  add_intercept = TRUE
)
```

**Arguments**

<code>data</code>	A <code>data.frame</code> of observations. It must include all variables used in the supplied formula.
<code>fixed</code>	A character vector of fixed effects.
<code>random</code>	A character vector of random effects. Random effects specified here will be added to the fixed effects.
<code>custom_random</code>	A vector of random effects. Random effects added here will not be added to the vector of fixed effects. This can be used to random effects for fixed effects that only have a partial name match.
<code>no_contrasts</code>	Logical, defaults to <code>FALSE</code> . <code>TRUE</code> means that no variable uses contrast. Alternatively a character vector of variables can be supplied indicating which variables should not have contrasts.
<code>add_intercept</code>	Logical, defaults to <code>FALSE</code> . Should an intercept be added to the fixed effects.

**Value**

A list specifying the fixed effects (formula, design matrix, and design matrix index), and random effects (formula and design matrix).

**See Also**

Functions used to help convert formulas into model designs [as\\_string\\_formula\(\)](#), [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_formula\(\)](#), [parse\\_formula\(\)](#), [remove\\_rw\\_terms\(\)](#), [re\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#), [split\\_formula\\_to\\_terms\(\)](#)

**Examples**

```
data <- enw_example("prep")$metareference[[1]]
enw_manual_formula(data, fixed = "week", random = "day_of_week")
```

---

enw_metadata	<i>Extract metadata from raw data</i>
--------------	---------------------------------------

---

## Description

Extract metadata from raw data, either by reference or by report date. For the target date chosen (reference or report), `confirm`, `max_confirm``, and `cum_prop_reported`` are dropped and the first observation for each group and date is retained.

## Usage

```
enw_metadata(obs, target_date = c("reference_date", "report_date"))
```

## Arguments

<code>obs</code>	A <code>data.frame</code> or <code>data.table</code> with columns: <code>reference_date</code> and/or <code>report_date</code> ; at least one must be provided, <code>.group</code> , a grouping column and a date, a <a href="#">Date</a> column.
<code>target_date</code>	A character string, either "reference_date" or "report_date". The column corresponding to this string will be used as the target date for metadata extraction.

## Value

A `data.table` with columns:

- `date`, a [Date](#) column
- `.group`, a grouping column

and the first observation for each group and date. The `data.table` is sorted by `.group` and date.

## See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

## Examples

```
obs <- data.frame(
  reference_date = as.Date("2021-01-01"),
  report_date = as.Date("2022-01-01"), x = 1:10
)
enw_metadata(obs, target_date = "reference_date")
```

---

enw\_metadata\_delay      *Calculate reporting delay metadata for a given maximum delay*

---

### Description

Calculate delay metadata based on the supplied maximum delay and independent of other metadata or date indexing. These data are meant to be used in conjunction with metadata on the date of reference. Users can build additional features with this `data.frame` or regenerate it using this function in the output of `enw_preprocess_data()`.

### Usage

```
enw_metadata_delay(max_delay = 20, breaks = 4, timestep = "day")
```

### Arguments

<code>max_delay</code>	The maximum number of days to model in the delay distribution. Must be an integer greater than or equal to 1. Observations with delays larger than the maximum delay will be dropped. If the specified maximum delay is too short, nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Consider what delays are realistic for your application, and when in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by <code>epinowcast</code> . If it does, your maximum delay may still be too short. Note that delays are zero indexed and so include the reference date and <code>max_delay - 1</code> other days (i.e. a <code>max_delay</code> of 1 corresponds to no delay). You can use <code>check_max_delay()</code> to check the coverage of a delay distribution for different maximum delays.
<code>breaks</code>	Numeric, defaults to 4. The number of breaks to use when constructing a categorised version of numeric delays.
<code>timestep</code>	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.

### Value

A `data.frame` of delay metadata. This includes:

- `delay`: The numeric delay from reference date to report.
- `delay_cat`: The categorised delay. This may be useful for model building.
- `delay_week`: The numeric week since the delay was reported. This again may be useful for model building.
- `delay_head`: A logical variable defining if the delay is in the lower 25% of the potential delays. This may be particularly useful when building models that assume a parametric distribution in order to increase the weight of the head of the reporting distribution in a pragmatic way.



- `delay_tail`: A logical variable defining if the delay is in the upper 75% of the potential delays. This may be particularly useful when building models that assume a parametric distribution in order to increase the weight of the tail of the reporting distribution in a pragmatic way.

### See Also

Preprocessing functions `enw_add_delay()`, `enw_add_max_reported()`, `enw_add_metaobs_features()`, `enw_assign_group()`, `enw_complete_dates()`, `enw_construct_data()`, `enw_extend_date()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle_to_long()`, `enw_reporting_triangle()`

### Examples

```
enw_metadata_delay(max_delay = 20, breaks = 4)
```

---

enw_missing	<i>Missing reference data model module</i>
-------------	--

---

### Description

Missing reference data model module

### Usage

```
enw_missing(formula = ~1, data)
```

### Arguments

formula	A formula (as implemented in <code>enw_formula()</code> ) describing the missing data proportion on the logit scale by reference date. This can use features defined by reference date as defined in <code>metareference</code> as produced by <code>enw_preprocess_data()</code> . " <code>~0</code> " implies no model is required. Otherwise an intercept is always needed
data	Output from <code>enw_preprocess_data()</code> .

### Value

A list containing the supplied formulas, data passed into a list describing the models, a `data.frame` describing the priors used, and a function that takes the output data and priors and returns a function that can be used to sample from a tightened version of the prior distribution.

### See Also

Model modules `enw_expectation()`, `enw_fit_opts()`, `enw_obs()`, `enw_reference()`, `enw_report()`

**Examples**

```
# Missingness model with a fixed intercept only
enw_missing(data = enw_example("preprocessed"))

# No missingness model specified
enw_missing(~0, data = enw_example("preprocessed"))
```

---

enw\_missing\_reference *Extract reports with missing reference dates*

---

**Description**

Returns reports with missing reference dates as well as calculating the proportion of reports for a given reference date that were missing.

**Usage**

```
enw_missing_reference(obs)
```

**Arguments**

`obs` A data.frame as produced by `enw_add_incidence()`. Must contain the following variables: `report_date`, `reference_date`, `.group`, and `confirm`, and `new_confirm`.

**Value**

A data.table of missing counts and proportions by report date and group.

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

**Examples**

```
obs <- data.frame(
  report_date = c("2021-10-01", "2021-10-03"), reference_date = "2021-10-01",
  confirm = 1
)
obs <- rbind(
  obs,
  data.frame(report_date = "2021-10-04", reference_date = NA, confirm = 4)
)
obs <- enw_complete_dates(obs)
```

```

obs <- enw_assign_group(obs)
obs <- enw_add_incidence(obs)
enw_missing_reference(obs)

```

---

enw\_model

*Load and compile the nowcasting model*


---

## Description

Load and compile the nowcasting model

## Usage

```

enw_model(
  model = system.file("stan", "epinowcast.stan", package = "epinowcast"),
  include = system.file("stan", package = "epinowcast"),
  compile = TRUE,
  threads = TRUE,
  profile = FALSE,
  target_dir = epinowcast::enw_get_cache(),
  stanc_options = list(),
  cpp_options = list(),
  verbose = TRUE,
  ...
)

```

## Arguments

model	A character string indicating the path to the model. If not supplied the package default model is used.
include	A character string specifying the path to any stan files to include in the model. If missing the package default is used.
compile	Logical, defaults to TRUE. Should the model be loaded and compiled using <code>cmdstanr::cmdstan_model()</code> .
threads	Logical, defaults to TRUE. Should the model compile with support for multi-thread support in chain. Note that setting this will produce a warning that <code>threads_to_chain</code> is set and ignored. Changing this to FALSE is not expected to yield any performance benefits even when not using multithreading and thus not recommended.
profile	Logical, defaults to FALSE. Should the model be profiled? For more on profiling see the <a href="#">cmdstanr documentation</a> . <code># nolint</code>
target_dir	The path to a directory in which the manipulated .stan files without profiling statements should be stored. To avoid overriding of the original .stan files, this should be different from the directory of the original model and the <code>include_paths</code> .

stanc_options	A list of options to pass to the stanc_options of <code>cmdstanr::cmdstan_model()</code> . By default nothing is passed but potentially users may wish to pass optimisation flags for example. See the documentation for <code>cmdstanr::cmdstan_model()</code> for further details.
cpp_options	A list of options to pass to the cpp_options of <code>cmdstanr::cmdstan_model()</code> . By default nothing is passed but potentially users may wish to pass optimisation flags for example. See the documentation for <code>cmdstanr::cmdstan_model()</code> for further details. Note that the threads argument replaces stan_threads.
verbose	Logical, defaults to TRUE. Should verbose messages be shown.
...	Additional arguments passed to <code>cmdstanr::cmdstan_model()</code> .

**Value**

A cmdstanr model.

**See Also**

Functions used to help convert models into the format required for stan `enw_formula_as_data_list()`, `enw_get_cache()`, `enw_priors_as_data_list()`, `enw_replace_priors()`, `enw_sample()`, `enw_set_cache()`, `enw_stan_to_r()`, `enw_unset_cache()`, `remove_profiling()`, `write_stan_files_no_profile()`

**Examples**

```
mod <- enw_model()
```

---

enw\_nowcast\_samples *Extract posterior samples for the nowcast prediction*

---

**Description**

A generic wrapper around `posterior::draws_df()` with opinionated defaults to extract the posterior samples for the nowcast ("pp\_inf\_obs" from the stan code). The functionality of this function can be used directly on the output of `epinowcast()` using the supplied `summary.epinowcast()` method.

**Usage**

```
enw_nowcast_samples(fit, obs, max_delay = NULL, timestep = "day")
```

**Arguments**

fit	A cmdstanr fit object.
obs	An observation data.frame containing reference_date columns of the same length as the number of rows in the posterior and the most up to date observation for each date. This is used to align the posterior with the observations. The easiest source of this data is the output of latest output of <code>enw_preprocess_data()</code> or <code>enw_latest_data()</code> .

max_delay	Maximum delay to which nowcasts should be summarised. Must be equal (default) or larger than the modelled maximum delay. If it is larger, then nowcasts for unmodelled dates are added by assuming that case counts beyond the modelled maximum delay are fully observed.
timestep	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.

### Value

A data.frame of posterior samples for the nowcast prediction. This uses observed data where available and the posterior prediction where not.

### See Also

Functions used for postprocessing of model fits [enw\\_add\\_latest\\_obs\\_to\\_nowcast\(\)](#), [enw\\_nowcast\\_summary\(\)](#), [enw\\_posterior\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_quantiles\\_to\\_long\(\)](#), [enw\\_summarise\\_samples\(\)](#)

### Examples

```
fit <- enw_example("nowcast")
enw_nowcast_samples(
  fit$fit[[1]],
  fit$latest[[1]],
  fit$max_delay,
  "day"
)
```

---

enw\_nowcast\_summary     *Summarise the posterior nowcast prediction*

---

### Description

A generic wrapper around [enw\\_posterior\(\)](#) with opinionated defaults to extract the posterior prediction for the nowcast ("pp\_inf\_obs" from the stan code). The functionality of this function can be used directly on the output of [epinowcast\(\)](#) using the supplied [summary.epinowcast\(\)](#) method.

### Usage

```
enw_nowcast_summary(
  fit,
  obs,
  max_delay = NULL,
  timestep = "day",
  probs = c(0.05, 0.2, 0.35, 0.5, 0.65, 0.8, 0.95)
)
```

**Arguments**

fit	A cmdstanr fit object.
obs	An observation data.frame containing reference_date columns of the same length as the number of rows in the posterior and the most up to date observation for each date. This is used to align the posterior with the observations. The easiest source of this data is the output of latest output of <a href="#">enw_preprocess_data()</a> or <a href="#">enw_latest_data()</a> .
max_delay	Maximum delay to which nowcasts should be summarised. Must be equal (default) or larger than the modelled maximum delay. If it is larger, then nowcasts for unmodelled dates are added by assuming that case counts beyond the modelled maximum delay are fully observed.
timestep	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
probs	A vector of numeric probabilities to produce quantile summaries for. By default these are the 5%, 20%, 80%, and 95% quantiles which are also the minimum set required for plotting functions to work.

**Value**

A data.frame summarising the model posterior nowcast prediction. This uses observed data where available and the posterior prediction where not.

**See Also**

[summary.epinowcast\(\)](#)

Functions used for postprocessing of model fits [enw\\_add\\_latest\\_obs\\_to\\_nowcast\(\)](#), [enw\\_nowcast\\_samples\(\)](#), [enw\\_posterior\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_quantiles\\_to\\_long\(\)](#), [enw\\_summarise\\_samples\(\)](#)

**Examples**

```
fit <- enw_example("nowcast")
enw_nowcast_summary(
  fit$fit[[1]],
  fit$latest[[1]],
  fit$max_delay
)
```

---

enw\_obs

*Setup observation model and data*


---

**Description**

Setup observation model and data

**Usage**

```
enw_obs(family = c("negbin", "poisson"), observation_indicator = NULL, data)
```

**Arguments**

family	Character string, the observation model to use in the likelihood; enforced by <code>base::match.arg()</code> . By default this is a negative binomial ("negbin") with Poisson ("poisson") also being available. Support for additional observation models is planned, please open an issue with suggestions.
observation_indicator	A character string, the name of the column in the data that indicates whether an observation is observed or not (using a logical variable) and therefore whether or not it should be used in the likelihood. This variable should be present in the data input to <code>enw_preprocess_data()</code> . It can be generated using <code>flag_observation</code> in <code>enw_complete_dates()</code> or it can be created directly using <code>enw_flag_observed_observations()</code> . If either of these approaches are used then the variable will be name <code>.observed</code> . Default is NULL.
data	Output from <code>enw_preprocess_data()</code> .

**Value**

A list as required by stan.

**See Also**

Model modules `enw_expectation()`, `enw_fit_opts()`, `enw_missing()`, `enw_reference()`, `enw_report()`

**Examples**

```
enw_obs(data = enw_example("preprocessed"))
```

---

enw\_one\_hot\_encode\_feature

*One-hot encode a variable and column-bind it to the original data.table*

---

**Description**

This function takes a data.frame and a categorical variable, performs one-hot encoding, and column-binds the encoded variables back to the data.frame.

**Usage**

```
enw_one_hot_encode_feature(metaobs, feature, contrasts = FALSE)
```

**Arguments**

metaobs	A data.frame containing the data to be encoded.
feature	The name of the categorical variable to one-hot encode as a character string.
contrasts	Logical. If TRUE, create one-hot encoded variables with contrasts; if FALSE, create them without contrasts. Defaults to FALSE.

**See Also**

Functions used to formulate models [enw\\_add\\_cumulative\\_membership\(\)](#), [enw\\_add\\_pooling\\_effect\(\)](#), [enw\\_design\(\)](#), [enw\\_effects\\_metadata\(\)](#)

**Examples**

```
metaobs <- data.frame(week = 1:2)
enw_one_hot_encode_feature(metaobs, "week")
enw_one_hot_encode_feature(metaobs, "week", contrasts = TRUE)

metaobs <- data.frame(week = 1:6)
enw_one_hot_encode_feature(metaobs, "week")
enw_one_hot_encode_feature(metaobs, "week", contrasts = TRUE)
```

---

```
enw_plot_nowcast_quantiles
      Plot nowcast quantiles
```

---

**Description**

Plot nowcast quantiles

**Usage**

```
enw_plot_nowcast_quantiles(nowcast, latest_obs = NULL, log = FALSE, ...)
```

**Arguments**

nowcast	A data.frame of summarised posterior nowcast estimates containing at least a confirm count column and a reference_date date variable.
latest_obs	A data.frame of observed data containing at least a confirm count variable and the same date variable as in the main data.frame used for plotting.
log	Logical, defaults to FALSE. Should counts be plot on the log scale.
...	Additional arguments passed to <a href="#">enw_plot_pp_quantiles()</a> .

**Value**

A ggplot2 plot.

**See Also**

Plotting functions [enw\\_plot\\_obs\(\)](#), [enw\\_plot\\_pp\\_quantiles\(\)](#), [enw\\_plot\\_quantiles\(\)](#), [enw\\_plot\\_theme\(\)](#), [plot.epinowcast\(\)](#)

**Examples**

```
nowcast <- enw_example("nowcast")
nowcast <- summary(nowcast, probs = c(0.05, 0.2, 0.8, 0.95))
enw_plot_nowcast_quantiles(nowcast)
```



---

enw\_plot\_obs                      *Generic quantile plot*

---

### Description

Generic quantile plot

### Usage

```
enw_plot_obs(obs, latest_obs = NULL, log = TRUE, ...)
```

### Arguments

obs	A data.frame of summarised posterior estimates containing at least a confirm count column and a date variable
latest_obs	A data.frame of observed data containing at least a confirm count variable and the same date variable as in the main data.frame used for plotting.
log	Logical, defaults to FALSE. Should counts be plot on the log scale.
...	Additional arguments passed to <code>ggplot2::aes()</code> must at least specify the x date variable.

### Value

A ggplot2 plot.

### See Also

Plotting functions `enw_plot_nowcast_quantiles()`, `enw_plot_pp_quantiles()`, `enw_plot_quantiles()`, `enw_plot_theme()`, `plot.epinowcast()`

### Examples

```
nowcast <- enw_example("nowcast")
obs <- enw_example("obs")

# Plot observed data by reference date
enw_plot_obs(obs, x = reference_date)

# Plot observed data by reference date with more recent data
enw_plot_obs(nowcast$latest[[1]], obs, x = reference_date)
```

---

enw\_plot\_pp\_quantiles *Plot posterior prediction quantiles*

---

**Description**

Plot posterior prediction quantiles

**Usage**

```
enw_plot_pp_quantiles(pp, log = FALSE, ...)
```

**Arguments**

`pp` A data.frame of summarised posterior predictions estimates containing at least a confirm count column and a report\_date date variable.

`log` Logical, defaults to FALSE. Should counts be plot on the log scale.

`...` Additional arguments passed to `enw_plot_pp_quantiles()`.

**Value**

A ggplot2 plot.

**See Also**

Plotting functions `enw_plot_nowcast_quantiles()`, `enw_plot_obs()`, `enw_plot_quantiles()`, `enw_plot_theme()`, `plot.epinowcast()`

**Examples**

```
nowcast <- enw_example("nowcast")
nowcast <- summary(
  nowcast, type = "posterior_prediction", probs = c(0.05, 0.2, 0.8, 0.95)
)
enw_plot_pp_quantiles(nowcast) +
  ggplot2::facet_wrap(ggplot2::vars(reference_date), scales = "free")
```

---

enw\_plot\_quantiles *Generic quantile plot*

---

**Description**

Generic quantile plot

**Usage**

```
enw_plot_quantiles(posterior, latest_obs = NULL, log = FALSE, ...)
```

**Arguments**

posterior	A data.frame of summarised posterior estimates containing at least a confirm count column a date variable, quantile estimates for the 5%, 20%, 80%, and 95% quantiles and the mean and median. This function is wrapped in <a href="#">enw_plot_nowcast_quantiles()</a> and <a href="#">enw_plot_pp_quantiles()</a> with sensible default labels.
latest_obs	A data.frame of observed data containing at least a confirm count variable and the same date variable as in the main data.frame used for plotting.
log	Logical, defaults to FALSE. Should counts be plot on the log scale.
...	Additional arguments passed to <a href="#">ggplot2::aes()</a> must at least specify the x date variable.

**Value**

A ggplot2 plot.

**See Also**

[enw\\_plot\\_nowcast\\_quantiles\(\)](#), [enw\\_plot\\_pp\\_quantiles\(\)](#)

Plotting functions [enw\\_plot\\_nowcast\\_quantiles\(\)](#), [enw\\_plot\\_obs\(\)](#), [enw\\_plot\\_pp\\_quantiles\(\)](#), [enw\\_plot\\_theme\(\)](#), [plot.epinowcast\(\)](#)

**Examples**

```
nowcast <- enw_example("nowcast")
nowcast <- summary(nowcast, probs = c(0.05, 0.2, 0.8, 0.95))
enw_plot_quantiles(nowcast, x = reference_date)
```

---

enw_plot_theme	<i>Package plot theme</i>
----------------	---------------------------

---

**Description**

Package plot theme

**Usage**

```
enw_plot_theme(plot)
```

**Arguments**

plot	ggplot2 plot object.
------	----------------------

**Value**

ggplot2 plot object.

**See Also**

Plotting functions [enw\\_plot\\_nowcast\\_quantiles\(\)](#), [enw\\_plot\\_obs\(\)](#), [enw\\_plot\\_pp\\_quantiles\(\)](#), [enw\\_plot\\_quantiles\(\)](#), [plot.epinowcast\(\)](#)

---

enw_posterior	<i>Summarise the posterior</i>
---------------	--------------------------------

---

**Description**

A generic wrapper around `posterior::summarise_draws()` with opinionated defaults.

**Usage**

```
enw_posterior(fit, variables = NULL, probs = c(0.05, 0.2, 0.8, 0.95), ...)
```

**Arguments**

<code>fit</code>	A cmdstanr fit object.
<code>variables</code>	A character vector of variables to return posterior summaries for. By default summaries for all parameters are returned.
<code>probs</code>	A vector of numeric probabilities to produce quantile summaries for. By default these are the 5%, 20%, 80%, and 95% quantiles which are also the minimum set required for plotting functions to work.
<code>...</code>	Additional arguments that may be passed but will not be used.

**Value**

A data.frame summarising the model posterior.

**See Also**

Functions used for postprocessing of model fits [enw\\_add\\_latest\\_obs\\_to\\_nowcast\(\)](#), [enw\\_nowcast\\_samples\(\)](#), [enw\\_nowcast\\_summary\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_quantiles\\_to\\_long\(\)](#), [enw\\_summarise\\_samples\(\)](#)

**Examples**

```
fit <- enw_example("nowcast")
enw_posterior(fit$fit[[1]], variables = "expr_beta")
```

---

enw_pp_summary	<i>Posterior predictive summary</i>
----------------	-------------------------------------

---

### Description

This function summarises posterior predictives for observed data (by report and reference date). The functionality of this function can be used directly on the output of `epinowcast()` using the supplied `summary.epinowcast()` method.

### Usage

```
enw_pp_summary(fit, diff_obs, probs = c(0.05, 0.2, 0.35, 0.5, 0.65, 0.8, 0.95))
```

### Arguments

<code>fit</code>	A cmdstanr fit object.
<code>diff_obs</code>	A data.frame of observed data with at least a date variable <code>reference_date</code> , and a grouping variable <code>.group</code> .
<code>probs</code>	A vector of numeric probabilities to produce quantile summaries for. By default these are the 5%, 20%, 80%, and 95% quantiles which are also the minimum set required for plotting functions to work.

### Value

A data.table summarising the posterior predictions.

### See Also

Functions used for postprocessing of model fits `enw_add_latest_obs_to_nowcast()`, `enw_nowcast_samples()`, `enw_nowcast_summary()`, `enw_posterior()`, `enw_quantiles_to_long()`, `enw_summarise_samples()`

### Examples

```
fit <- enw_example("nowcast")
enw_pp_summary(fit$fit[[1]], fit$new_confirm[[1]], probs = c(0.5))
```

---

enw\_preprocess\_data    *Preprocess observations*

---

## Description

This function preprocesses raw observations under the assumption they are reported as cumulative counts by a reference and report date and is used to assign groups. It also constructs data objects used by visualisation and modelling functions including the observed empirical probability of a report on a given day, the cumulative probability of report, the latest available observations, incidence of observations, and metadata about the date of reference and report (used to construct models). This function wraps other preprocessing functions that may be instead used individually if required. Note that internally reports beyond the user specified delay are dropped for modelling purposes with the `cum_prop_reported` and `max_confirm` variables allowing the user to check the impact this may have (if `cum_prop_reported` is significantly below 1 a longer `max_delay` may be appropriate). Also note that if missing reference or report dates are suspected to occur in your data then these need to be completed with `enw_complete_dates()`.

## Usage

```
enw_preprocess_data(
  obs,
  by = NULL,
  max_delay,
  timestep = "day",
  set_negatives_to_zero = TRUE,
  ...,
  copy = TRUE
)
```

## Arguments

<code>obs</code>	A data.frame containing at least the following variables: <code>reference_date</code> (index date of interest), <code>report_date</code> (report date for observations), <code>confirm</code> (cumulative observations by reference and report date).
<code>by</code>	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling
<code>max_delay</code>	The maximum number of days to model in the delay distribution. If not specified the maximum observed delay is assumed to be the true maximum delay in the model. Otherwise, an integer greater than or equal to 1 can be specified. Observations with delays larger than the maximum delay will be dropped. If the specified maximum delay is too short, nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Steps to take to determine the maximum delay:

- Consider what is realistic and relevant for your application.
- Check the proportion of observations reported (`prop_reported`) by delay in the `new_confirm` output of `enw_preprocess_obs`.
- Use `check_max_delay()` to check the coverage of a candidate `max_delay`.
- If in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by `epinowcast`. If it does, your maximum delay may still be too short.

Note that delays are zero indexed and so include the reference date and `max_delay - 1` other days (i.e. a `max_delay` of 1 corresponds to no delay).

<code>timestep</code>	The timestep to used in the process model (i.e. the reference date model). This can be a string ("day", "week", "month") or a numeric whole number representing the number of days. If your data does not have this timestep then you may wish to make use of <code>enw_aggregate_cumulative()</code> to aggregate your data to the desired timestep.
<code>set_negatives_to_zero</code>	Logical, defaults to TRUE. Should negative counts (for calculated incidence of observations) be set to zero? Currently downstream modelling does not support negative counts and so setting must be TRUE if intending to use <code>epinowcast()</code> .
<code>...</code>	Other arguments to <code>enw_add_metaobs_features()</code> , e.g. <code>holidays</code> , which sets commonly used metadata (e.g. day of week, days since start of time series)
<code>copy</code>	A logical; if TRUE (the default) creates a copy; otherwise, modifies obs in place.

## Details

If `max_delay` is numeric, it will be internally coerced to integer using `as.integer()`.

## Value

A `data.table` containing processed observations as a series of nested `data.frames` as well as variables containing metadata. These are:

- `obs`: (observations with the addition of empirical reporting proportions and restricted to the specified maximum delay).
- `new_confirm`: Incidence of notifications by reference and report date. Empirical reporting distributions are also added.
- `latest`: The latest available observations.
- `missing_reference`: Observations missing reference dates.
- `reporting_triangle`: Incident observations by report and reference date in the standard reporting triangle matrix format.
- `metareference`: Metadata reference dates derived from observations.
- `metrereport`: Metadata for report dates.
- `metadelay`: Metadata for reporting delays produced using `enw_metadata_delay()`.
- `max_delay`: Maximum delay to be modelled by `epinowcast`.
- `time`: Numeric, number of timepoints in the data.

- snapshots: Numeric, number of available data snapshots to use for nowcasting.
- groups: Numeric, Number of groups/strata in the supplied observations (set using by).
- max\_date: The maximum available report date.

### See Also

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

### Examples

```
library(data.table)

# Filter example hospitalisation data to be national and over all ages
nat_germany_hosp <- germany_covid19_hosp[location == "DE"]
nat_germany_hosp <- nat_germany_hosp[age_group == "00+"]

# Preprocess with default settings
pobs <- enw_preprocess_data(nat_germany_hosp)
pobs
```

---

```
enw_priors_as_data_list
```

*Convert prior data.frame to list*

---

### Description

Converts priors defined in a `data.frame` into a list format for use by `stan`. In addition it adds `"_p"` to all variable names in order too allow them to be distinguished from their standard usage within modelling code.

### Usage

```
enw_priors_as_data_list(priors)
```

### Arguments

`priors` A `data.frame` with the following variables: `variable`, `mean`, `sd` describing normal priors. Priors in the appropriate format are returned by [enw\\_reference\(\)](#) as well as by other similar model specification functions.

### Value

A named list with each entry specifying a prior as a length two vector (specifying the mean and standard deviation of the prior).



**See Also**

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

**Examples**

```
priors <- data.frame(variable = "x", mean = 1, sd = 2)
enw_priors_as_data_list(priors)
```

---

`enw_quantiles_to_long` *Convert summarised quantiles from wide to long format*

---

**Description**

Convert summarised quantiles from wide to long format

**Usage**

```
enw_quantiles_to_long(posterior)
```

**Arguments**

`posterior` A data.frame as output by [enw\\_posterior\(\)](#).

**Value**

A data.frame of quantiles in long format.

**See Also**

Functions used for postprocessing of model fits [enw\\_add\\_latest\\_obs\\_to\\_nowcast\(\)](#), [enw\\_nowcast\\_samples\(\)](#), [enw\\_nowcast\\_summary\(\)](#), [enw\\_posterior\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_summarise\\_samples\(\)](#)

**Examples**

```
fit <- enw_example("nowcast")
posterior <- enw_posterior(fit$fit[[1]], var = "expr_llatent_int[1,1]")
enw_quantiles_to_long(posterior)
```

---

enw\_reference                      *Reference date logit hazard reporting model module*

---

## Description

Reference date logit hazard reporting model module

## Usage

```
enw_reference(
  parametric = ~1,
  distribution = c("lognormal", "none", "exponential", "gamma", "loglogistic"),
  non_parametric = ~0,
  data
)
```

## Arguments

parametric	A formula (as implemented in <a href="#">enw_formula()</a> ) describing the parametric reference date delay model. This can use features defined by report date as defined in <a href="#">metareference</a> as produced by <a href="#">enw_preprocess_data()</a> . Note that this formula will be applied to all summary statistics of the chosen parametric distribution but each summary parameter will have separate effects. Use $\sim 0$ to not use a parametric model.
distribution	A character vector describing the parametric delay distribution to use. Current options are: "none", "lognormal", "gamma", "exponential", and "loglogistic", with the default being "lognormal".
non_parametric	A formula (as implemented in <a href="#">enw_formula()</a> ) describing the non-parametric logit hazard model. This can use features defined by reference date and by delay. It draws on a linked data.frame using <a href="#">metareference</a> and <a href="#">metadelay</a> as produced by <a href="#">enw_preprocess_data()</a> . When an effect per delay is specified this approximates the cox proportional hazard model in discrete time with a single strata. When used in conjunction with a parametric model it likely makes sense to disable the intercept in order to make the joint model identifiable (i.e. $\sim 0 + (1   \text{delay})$ ).
data	Output from <a href="#">enw_preprocess_data()</a> .

## Value

A list containing the supplied formulas, data passed into a list describing the models, a data.frame describing the priors used, and a function that takes the output data and priors and returns a function that can be used to sample from a tightened version of the prior distribution.

## See Also

Model modules [enw\\_expectation\(\)](#), [enw\\_fit\\_opts\(\)](#), [enw\\_missing\(\)](#), [enw\\_obs\(\)](#), [enw\\_report\(\)](#)

**Examples**

```

# Parametric model with a lognormal distribution
enw_reference(
  parametric = ~1, distribution = "lognormal",
  data = enw_example("preprocessed")
)

# Non-parametric model with a random effect per delay
enw_reference(
  parametric = ~ 0, non_parametric = ~ 1 + (1 | delay),
  data = enw_example("preprocessed")
)

# Combined parametric and non-parametric model
enw_reference(
  parametric = ~ 1, non_parametric = ~ 0 + (1 | delay_cat),
  data = enw_example("preprocessed")
)

```

---

enw\_reference\_by\_report

*Construct a lookup of references dates by report*


---

**Description**

Construct a lookup of references dates by report

**Usage**

```

enw_reference_by_report(
  missing_reference,
  reps_with_complete_refs,
  metareference,
  max_delay
)

```

**Arguments**

**missing\_reference** missing\_reference data.frame output from [enw\\_preprocess\\_data\(\)](#).

**reps\_with\_complete\_refs** A data.frame of report dates with complete (i.e fully reported) reference dates as produced using [enw\\_reps\\_with\\_complete\\_refs\(\)](#).

**metareference** metareference data.frame output from [enw\\_preprocess\\_data\(\)](#).

**max\_delay** The maximum number of days to model in the delay distribution. Must be an integer greater than or equal to 1. Observations with delays larger then the maximum delay will be dropped. If the specified maximum delay is too short,

nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Consider what delays are realistic for your application, and when in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by `epinowcast`. If it does, your maximum delay may still be too short. Note that delays are zero indexed and so include the reference date and `max_delay - 1` other days (i.e. a `max_delay` of 1 corresponds to no delay). You can use `check_max_delay()` to check the coverage of a delay distribution for different maximum delays.

### Value

A wide `data.frame` with each row being a complete report date and' the columns being the observation index for each reporting delay

### See Also

Helper functions for model modules `add_max_observed_delay()`, `add_pmfs()`, `convolution_matrix()`, `enw_reps_with_complete_refs()`, `extract_obs_metadata()`, `extract_sparse_matrix()`, `latest_obs_as_matrix()`, `simulate_double_censored_pmf()`

---

`enw_replace_priors`      *Replace default priors with user specified priors*

---

### Description

This function is used internally by `epinowcast` to replace default model priors with users specified ones (restricted to normal priors with specified mean and standard deviations). A common use would be extracting the posterior from a previous `epinowcast()` run (using `summary(nowcast, type = fit)`) and using this a prior.

### Usage

```
enw_replace_priors(priors, custom_priors)
```

### Arguments

<code>priors</code>	A <code>data.frame</code> with the following variables: <code>variable</code> , <code>mean</code> , <code>sd</code> describing normal priors. Priors in the appropriate format are returned by <code>enw_reference()</code> as well as by other similar model specification functions.
<code>custom_priors</code>	A <code>data.frame</code> with the following variables: <code>variable</code> , <code>mean</code> , <code>sd</code> describing normal priors. Priors in the appropriate format are returned by <code>enw_reference()</code> as well as by other similar model specification functions. Priors in this <code>data.frame</code> replace the default priors. Note that currently vectorised prior names (i.e those of the form <code>variable[n]</code> will be treated as <code>variable</code> ).

**Value**

A data.table of prior definitions (variable, mean and sd).

**See Also**

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

**Examples**

```
# Update priors from a data.frame
priors <- data.frame(variable = c("x", "y"), mean = c(1, 2), sd = c(1, 2))
custom_priors <- data.frame(variable = "x[1]", mean = 10, sd = 2)
enw_replace_priors(priors, custom_priors)

# Update priors from a previous model fit
default_priors <- enw_reference(
  distribution = "lognormal",
  data = enw_example("preprocessed"),
)$priors
print(default_priors)

fit_priors <- summary(
  enw_example("nowcast"), type = "fit",
  variables = c("refp_mean_int", "refp_sd_int", "sqrt_phi")
)
fit_priors

enw_replace_priors(default_priors, fit_priors)
```

---

enw\_report

*Report date logit hazard reporting model module*


---

**Description**

Report date logit hazard reporting model module

**Usage**

```
enw_report(non_parametric = ~0, structural = ~0, data)
```

**Arguments**

**non\_parametric** A formula (as implemented in [enw\\_formula\(\)](#)) describing the non-parametric logit hazard model. This can use features defined by report date as defined in [metareport](#) as produced by [enw\\_preprocess\\_data\(\)](#). Note that the intercept for this model is set to 0 as it should be used for specifying report date related hazards vs time invariant hazards which should instead be modelled using the **non\_parametric** argument of [enw\\_reference\(\)](#)

structural	A formula with fixed effects and using only binary variables, and factors describing the known reporting structure (i.e weekday only reporting). The base case (i.e the first factor entry) should describe the dates for which reporting is possible. Internally dates with a non-zero element in the design matrix have their hazard set to 0. This can use features defined by report date as defined in metareport as produced by <code>enw_preprocess_data()</code> . Note that the intercept for this model is set to 0 in order to allow all dates without other structural reasons to not be reported to be reported. Note that this feature is not yet available to users.
data	Output from <code>enw_preprocess_data()</code> .

**Value**

A list containing the supplied formulas, data passed into a list describing the models, a `data.frame` describing the priors used, and a function that takes the output data and priors and returns a function that can be used to sample from a tightened version of the prior distribution.

**See Also**

Model modules `enw_expectation()`, `enw_fit_opts()`, `enw_missing()`, `enw_obs()`, `enw_reference()`

**Examples**

```
enw_report(data = enw_example("preprocessed"))
```

---

```
enw_reporting_triangle
```

*Construct the reporting triangle*

---

**Description**

Constructs the reporting triangle with each row representing a reference date and columns being observations by report date

**Usage**

```
enw_reporting_triangle(obs)
```

**Arguments**

`obs` A `data.frame` as produced by `enw_add_incidence()`. Must contain the following variables: `reference_date`, `.group`, `delay`.

**Value**

A `data.frame` with each row being a reference date, and columns being observations by reporting delay.

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\\_to\\_long\(\)](#)

**Examples**

```
obs <- enw_example("preprocessed")$new_confirm
enw_reporting_triangle(obs)
```

---

```
enw_reporting_triangle_to_long
```

*Recast the reporting triangle from wide to long format*

---

**Description**

Recast the reporting triangle from wide to long format

**Usage**

```
enw_reporting_triangle_to_long(obs)
```

**Arguments**

obs                    A data.frame in the format produced by [enw\\_reporting\\_triangle\(\)](#).

**Value**

A long format reporting triangle as a data.frame with additional variables new\_confirm and delay.

**See Also**

Preprocessing functions [enw\\_add\\_delay\(\)](#), [enw\\_add\\_max\\_reported\(\)](#), [enw\\_add\\_metaobs\\_features\(\)](#), [enw\\_assign\\_group\(\)](#), [enw\\_complete\\_dates\(\)](#), [enw\\_construct\\_data\(\)](#), [enw\\_extend\\_date\(\)](#), [enw\\_filter\\_delay\(\)](#), [enw\\_filter\\_reference\\_dates\(\)](#), [enw\\_filter\\_report\\_dates\(\)](#), [enw\\_flag\\_observed\\_observations\(\)](#), [enw\\_impute\\_na\\_observations\(\)](#), [enw\\_latest\\_data\(\)](#), [enw\\_metadata\\_delay\(\)](#), [enw\\_metadata\(\)](#), [enw\\_missing\\_reference\(\)](#), [enw\\_preprocess\\_data\(\)](#), [enw\\_reporting\\_triangle\(\)](#)

**Examples**

```
obs <- enw_example("preprocessed")$new_confirm
rt <- enw_reporting_triangle(obs)
enw_reporting_triangle_to_long(rt)
```

---

 enw\_reps\_with\_complete\_refs

*Identify report dates with complete (i.e up to the maximum delay) reference dates*

---

### Description

Identify report dates with complete (i.e up to the maximum delay) reference dates

### Usage

```
enw_reps_with_complete_refs(new_confirm, max_delay, by = NULL, copy = TRUE)
```

### Arguments

new_confirm	new_confirm data.frame output from <a href="#">enw_preprocess_data()</a> .
max_delay	The maximum number of days to model in the delay distribution. Must be an integer greater than or equal to 1. Observations with delays larger than the maximum delay will be dropped. If the specified maximum delay is too short, nowcasts can be biased as important parts of the true delay distribution are cut off. At the same time, computational cost scales non-linearly with this setting, so you want the maximum delay to be as long as necessary, but not much longer. Consider what delays are realistic for your application, and when in doubt, check if increasing the maximum delay noticeably changes the delay distribution or nowcasts as estimated by <code>epinowcast</code> . If it does, your maximum delay may still be too short. Note that delays are zero indexed and so include the reference date and <code>max_delay - 1</code> other days (i.e. a <code>max_delay</code> of 1 corresponds to no delay). You can use <a href="#">check_max_delay()</a> to check the coverage of a delay distribution for different maximum delays.
by	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling
copy	A logical; if TRUE (the default) creates a copy; otherwise, modifies obs in place.

### Value

A data.frame containing a `report_date` variable, and grouping variables specified for report dates that have complete reporting.

### See Also

Helper functions for model modules [add\\_max\\_observed\\_delay\(\)](#), [add\\_pmfs\(\)](#), [convolution\\_matrix\(\)](#), [enw\\_reference\\_by\\_report\(\)](#), [extract\\_obs\\_metadata\(\)](#), [extract\\_sparse\\_matrix\(\)](#), [latest\\_obs\\_as\\_matrix\(\)](#), [simulate\\_double\\_censored\\_pmf\(\)](#)



---

enw_sample	<i>Fit a CmdStan model using NUTS</i>
------------	---------------------------------------

---

**Description**

Fit a CmdStan model using NUTS

**Usage**

```
enw_sample(data, model = epinowcast::enw_model(), diagnostics = TRUE, ...)
```

**Arguments**

data	A list of data as produced by model modules (for example <a href="#">enw_expectation()</a> , <a href="#">enw_obs()</a> , etc.) and as required for use the model being used.
model	A cmdstanr model object as loaded by <a href="#">enw_model()</a> or as supplied by the user.
diagnostics	Logical, defaults to TRUE. Should fitting diagnostics be returned as a data.frame.
...	Additional parameters passed to the sample method of cmdstanr.

**Value**

A data.frame containing the cmdstanr fit, the input data, the fitting arguments, and optionally summary diagnostics.

**See Also**

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

---

enw_score_nowcast	<i>Evaluate nowcasts using proper scoring rules</i>
-------------------	---

---

**Description**

Acts as a wrapper to [scoringutils::score\(\)](#). In particular, handling filtering nowcast summary output and linking this output to observed data. See the documentation for the [scoringutils](#) package for more on forecast scoring.

**Usage**

```
enw_score_nowcast(
  nowcast,
  latest_obs,
  log = FALSE,
  check = FALSE,
  round_to = 3,
  ...
)
```

**Arguments**

nowcast	A posterior nowcast or posterior prediction as returned by <code>summary.epinowcast()</code> , when used on the output of <code>epinowcast()</code> .
latest_obs	A <code>data.frame</code> of the latest available observations as produced by <code>enw_latest_data()</code> or otherwise.
log	Logical, defaults to <code>FALSE</code> . Should scores be calculated on the log scale (with a 0.01 shift) for both observations and nowcasts. Scoring in this way can be thought of as a relative score vs the more usual absolute measure. It may be useful when targets are on very different scales or when the forecaster is more interested in good all round performance versus good performance for targets with large values.
check	Logical, defaults to <code>FALSE</code> . Should <code>scoringutils::check_forecasts()</code> be used to check input nowcasts.
round_to	Integer defaults to 3. Number of digits to round scoring output to.
...	Arguments passed on to <code>scoringutils::score</code>
data	A <code>data.frame</code> or <code>data.table</code> with the predictions and observations. For scoring using <code>score()</code> , the following columns need to be present: <ul style="list-style-type: none"> <li>• <code>true_value</code> - the true observed values</li> <li>• <code>prediction</code> - predictions or predictive samples for one true value. (You only don't need to provide a prediction column if you want to score quantile forecasts in a wide range format.)</li> </ul> For scoring integer and continuous forecasts a <code>sample</code> column is needed: <ul style="list-style-type: none"> <li>• <code>sample</code> - an index to identify the predictive samples in the prediction column generated by one model for one true value. Only necessary for continuous and integer forecasts, not for binary predictions.</li> </ul> For scoring predictions in a quantile-format forecast you should provide a column called <code>quantile</code> : <ul style="list-style-type: none"> <li>• <code>quantile</code>: quantile to which the prediction corresponds</li> </ul> In addition a <code>model</code> column is suggested and if not present this will be flagged and added to the input data with all forecasts assigned as an "unspecified model". You can check the format of your data using <code>check_forecasts()</code> and there are examples for each format ( <code>example_quantile</code> , <code>example_continuous</code> , <code>example_integer</code> , and <code>example_binary</code> ).

metrics the metrics you want to have in the output. If NULL (the default), all available metrics will be computed. For a list of available metrics see [available\\_metrics\(\)](#), or check the [metrics](#) data set.

## Value

A data.table as returned by [scoringutils::score\(\)](#).

## Examples

```
library(data.table)
library(scoringutils)

# Summarise example nowcast
nowcast <- enw_example("nowcast")
summarised_nowcast <- summary(nowcast)

# Load latest available observations
obs <- enw_example("observations")

# Keep the last 7 days of data
obs <- obs[reference_date > (max(reference_date) - 7)]

# score on the absolute scale
scores <- enw_score_nowcast(summarised_nowcast, obs)
summarise_scores(scores, by = "location")

# score overall on a log scale
log_scores <- enw_score_nowcast(summarised_nowcast, obs, log = TRUE)
summarise_scores(log_scores, by = "location")
```

---

enw\_set\_cache

*Set caching location for Stan models*

---

## Description

This function allows the user to set a cache location for Stan models rather than a temporary directory. This can reduce the need for model compilation on every new model run across sessions or within a session. For R version 4.0.0 and above, it's recommended to use the persistent cache as shown in the example.

## Usage

```
enw_set_cache(path, type = c("session", "persistent", "all"))
```

**Arguments**

path	A valid filepath representing the desired cache location. If the directory does not exist it will be created.
type	A character string specifying the cache type. It can be one of "session", "persistent", or "all". Default is "session". "session" sets the cache for the current session, "persistent" writes the cache location to the user's .Renviron file, and "all" does both.

**Value**

The string of the filepath set.

**See Also**

Functions used to help convert models into the format required for stan `enw_formula_as_data_list()`, `enw_get_cache()`, `enw_model()`, `enw_priors_as_data_list()`, `enw_replace_priors()`, `enw_sample()`, `enw_stan_to_r()`, `enw_unset_cache()`, `remove_profiling()`, `write_stan_files_no_profile()`

**Examples**

```
# Set to local directory
my_enw_cache <- enw_set_cache(file.path(tempdir(), "test"))
enw_get_cache()
## Not run:
# Use the package cache in R >= 4.0
if (R.version.string >= "4.0.0") {
  enw_set_cache(
    tools::R_user_dir(package = "epinowcast", "cache"), type = "all"
  )
}

## End(Not run)
```

---

enw\_simulate\_missing\_reference

*Simulate observations with a missing reference date.*

---

**Description**

A simple binomial simulator of missing data by reference date using simulated or observed data as an input. This function may be used to validate missing data models, as part of examples and case studies, or to explore the implications of missing data for your use case.

**Usage**

```
enw_simulate_missing_reference(obs, proportion = 0.2, by = NULL)
```

**Arguments**

obs	A data.frame containing at least the following variables: reference date (index date of interest), report_date (report date for observations), and confirm (cumulative observations by reference and report date).
proportion	Numeric, the proportion of observations that are missing a reference date, indexed by reference date. Currently only a fixed proportion are supported and this defaults to 0.2.
by	A character vector describing the stratification of observations. This defaults to no grouping. This should be used when modelling multiple time series in order to identify them for downstream modelling

**Value**

A data.table of the same format as the input but with a simulated proportion of observations now having a missing reference date.

**Examples**

```
# Load and filter germany hospitalisations
nat_germany_hosp <- subset(
  germany_covid19_hosp, location == "DE" & age_group == "00+"
)
nat_germany_hosp <- enw_filter_report_dates(
  nat_germany_hosp,
  latest_date = "2021-08-01"
)

# Make sure observations are complete
nat_germany_hosp <- enw_complete_dates(
  nat_germany_hosp,
  by = c("location", "age_group"), missing_reference = FALSE
)

# Simulate
enw_simulate_missing_reference(
  nat_germany_hosp,
  proportion = 0.35, by = c("location", "age_group")
)
```

**Description**

This function facilitates the exposure of Stan functions from the `epinowcast` package in R. It utilizes the `expose_functions()` method of `cmdstanr::CmdStanModel` or this purpose. This function is useful for developers and contributors to the `epinowcast` package, as well as for users interested in exploring and prototyping with model functionalities.

**Usage**

```
enw_stan_to_r(
  files = list.files(include),
  include = system.file("stan", "functions", package = "epinowcast"),
  global = TRUE,
  verbose = TRUE,
  ...
)
```

**Arguments**

files	A character vector specifying the names of Stan files to be exposed. These must be in the include directory. Defaults to all Stan files in the include directory. Note that the following files contain overloaded functions and cannot be exposed: "delay_lpmf.stan", "allocate_observed_obs.stan", "obs_lpmf.stan", and "effects_priors_lp.stan".
include	A character string specifying the directory containing Stan files. Defaults to the 'stan/functions' directory of the <code>epinowcast()</code> package.
global	A logical value indicating whether to expose the functions globally. Defaults to TRUE. Passed to the <code>expose_functions()</code> method of <code>cmdstan::CmdStanModel</code> .
verbose	Logical, defaults to TRUE. Should verbose messages be shown.
...	Additional arguments passed to <code>enw_model</code> .

**Value**

An object of class `CmdStanModel` with functions from the model exposed for use in R.

**See Also**

Functions used to help convert models into the format required for stan `enw_formula_as_data_list()`, `enw_get_cache()`, `enw_model()`, `enw_priors_as_data_list()`, `enw_replace_priors()`, `enw_sample()`, `enw_set_cache()`, `enw_unset_cache()`, `remove_profiling()`, `write_stan_files_no_profile()`

**Examples**

```
# Compile functions in stan/functions/hazard.stan
stan_functions <- enw_stan_to_r("hazard.stan")
# These functions can now be used in R
stan_functions$functions$prob_to_hazard(c(0.5, 0.1, 0.1))
# or exposed globally and used directly
prob_to_hazard(c(0.5, 0.1, 0.1))
```

---

enw\_summarise\_samples *Summarise posterior samples*

---

## Description

This function summarises posterior samples for arbitrary strata. It optionally holds out the observed data (variables that are not ".draw", ".iteration", ".sample", ".chain" ) joins this to the summarised posterior.

## Usage

```
enw_summarise_samples(  
  samples,  
  probs = c(0.05, 0.2, 0.35, 0.5, 0.65, 0.8, 0.95),  
  by = c("reference_date", ".group"),  
  link_with_obs = TRUE  
)
```

## Arguments

samples	A data.frame of posterior samples with at least a numeric sample variable.
probs	A vector of numeric probabilities to produce quantile summaries for. By default these are the 5%, 20%, 80%, and 95% quantiles which are also the minimum set required for plotting functions to work.
by	A character vector of variables to summarise by. Defaults to c("reference_date", ".group").
link_with_obs	Logical, should the observed data be linked to the posterior summary? This is useful for plotting the posterior against the observed data. Defaults to TRUE.

## Value

A data.frame summarising the posterior samples.

## See Also

Functions used for postprocessing of model fits [enw\\_add\\_latest\\_obs\\_to\\_nowcast\(\)](#), [enw\\_nowcast\\_samples\(\)](#), [enw\\_nowcast\\_summary\(\)](#), [enw\\_posterior\(\)](#), [enw\\_pp\\_summary\(\)](#), [enw\\_quantiles\\_to\\_long\(\)](#)

## Examples

```
fit <- enw_example("nowcast")  
samples <- summary(fit, type = "nowcast_sample")  
enw_summarise_samples(samples, probs = c(0.05, 0.5, 0.95))
```

---

enw_unset_cache	<i>Unset Stan cache location</i>
-----------------	----------------------------------

---

### Description

Optionally removes the `enw_cache_location` environment variable from the user `.Renvirom` file and/or removes it from the local environment. If you unset the local cache and want to switch back to using the persistent cache, you can reload the `.Renvirom` file using `readRenvirom("~/ .Renvirom")`.

### Usage

```
enw_unset_cache(type = c("session", "persistent", "all"))
```

### Arguments

<code>type</code>	A character string specifying the type of cache to unset. It can be one of "session", "persistent", or "all". Default is "session". "session" unsets the cache for the current session, "persistent" removes the cache location from the user's <code>.Renvirom</code> file, and "all" does all options.
-------------------	--

### Value

The prior cache location, if it existed otherwise NULL.

### See Also

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [remove\\_profiling\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

### Examples

```
enw_unset_cache()
```

---

epinowcast	<i>Nowcast using partially observed data</i>
------------	--

---

### Description

Provides a user friendly interface around package functionality to produce a nowcast from observed preprocessed data, and a series of user defined models. By default a model that assumes a fixed parametric reporting distribution with a flexible expectation model is used. Explore the individual model components for additional documentation and see the package case studies for example model specifications for different tasks.



**Usage**

```
epinowcast(
  data,
  reference = epinowcast::enw_reference(parametric = ~1, distribution = "lognormal",
    non_parametric = ~0, data = data),
  report = epinowcast::enw_report(non_parametric = ~0, structural = ~0, data = data),
  expectation = epinowcast::enw_expectation(r = ~0 + (1 | day:.group), generation_time =
    1, observation = ~1, latent_reporting_delay = 1, data = data),
  missing = epinowcast::enw_missing(formula = ~0, data = data),
  obs = epinowcast::enw_obs(family = "negbin", data = data),
  fit = epinowcast::enw_fit_opts(sampler = epinowcast::enw_sample, nowcast = TRUE, pp =
    FALSE, likelihood = TRUE, debug = FALSE, output_loglik = FALSE),
  model = epinowcast::enw_model(),
  priors,
  ...
)
```

**Arguments**

data	Output from <a href="#">enw_preprocess_data()</a> .
reference	The reference date indexed reporting process model specification as defined using <a href="#">enw_reference()</a> .
report	The report date indexed reporting process model specification as defined using <a href="#">enw_report()</a> .
expectation	The expectation model specification as defined using <a href="#">enw_expectation()</a> . By default this is set to be a highly flexible random effect by reference date for each group and thus weakly informed. Depending on your context (and in particular the density of data reporting) other choices that enforce more assumptions may be more appropriate (for example a weekly random walk (specified using <code>rw(week, by = .group)</code> )).
missing	The missing reference date model specification as defined using <a href="#">enw_missing()</a> . By default this is set to not be used.
obs	The observation model as defined by <a href="#">enw_obs()</a> . Observations are also processed within this function for use in modelling.
fit	Model fit options as defined using <a href="#">enw_fit_opts()</a> . This includes the sampler function to use (with the package default being <a href="#">enw_sample()</a> ), whether or now a nowcast should be used, etc. See <a href="#">enw_fit_opts()</a> for further details.
model	The model to use within fit. By default this uses <a href="#">enw_model()</a> .
priors	A data.frame with the following variables: variable, mean, sd describing normal priors. Priors in the appropriate format are returned by <a href="#">enw_reference()</a> as well as by other similar model specification functions. Priors in this data.frame replace the default priors specified by each model component.
...	Additional model modules to pass to model. User modules may be used but currently require the supplied model to be adapted.

**Value**

A object of the class "epinowcast" which inherits from `enw_preprocess_data()` and `data.table`, and combines the input data, priors, and output from the sampler specified in `enw_fit_opts()`.

**See Also**

Other epinowcast: `plot.epinowcast()`, `summary.epinowcast()`

**Examples**

```
# Load data.table and ggplot2
library(data.table)
library(ggplot2)

# Use 2 cores
options(mc.cores = 2)
# Load and filter germany hospitalisations
nat_germany_hosp <-
  germany_covid19_hosp[location == "DE"][age_group == "00+"]
nat_germany_hosp <- enw_filter_report_dates(
  nat_germany_hosp,
  latest_date = "2021-10-01"
)
# Make sure observations are complete
nat_germany_hosp <- enw_complete_dates(
  nat_germany_hosp,
  by = c("location", "age_group")
)
# Make a retrospective dataset
retro_nat_germany <- enw_filter_report_dates(
  nat_germany_hosp,
  remove_days = 40
)
retro_nat_germany <- enw_filter_reference_dates(
  retro_nat_germany,
  include_days = 40
)
# Get latest observations for the same time period
latest_obs <- enw_latest_data(nat_germany_hosp)
latest_obs <- enw_filter_reference_dates(
  latest_obs,
  remove_days = 40, include_days = 20
)
# Preprocess observations (note this maximum delay is likely too short)
pobs <- enw_preprocess_data(retro_nat_germany, max_delay = 20)
# Fit the default nowcast model and produce a nowcast
# Note that we have reduced samples for this example to reduce runtimes
nowcast <- epinowcast(pobs,
  fit = enw_fit_opts(
    save_warmup = FALSE, pp = TRUE,
    chains = 2, iter_warmup = 500, iter_sampling = 500
  )
)
```

```

)
nowcast
# plot the nowcast vs latest available observations
plot(nowcast, latest_obs = latest_obs)

# plot posterior predictions for the delay distribution by date
plot(nowcast, type = "posterior") +
  facet_wrap(vars(reference_date), scale = "free")

```

---

extract\_obs\_metadata *Extract observation metadata*

---

## Description

This function extracts metadata from the provided dataset to be used in the observation model.

## Usage

```
extract_obs_metadata(new_confirm, observation_indicator = NULL)
```

## Arguments

**new\_confirm** A data.table containing the columns: "reference\_date", "delay", ".group", "new\_confirm", and "max\_obs\_delay". As produced by [enw\\_preprocess\\_data\(\)](#) in the new\_confirm output with the addition of the "max\_obs\_delay" column as produced by [add\\_max\\_observed\\_delay\(\)](#).

**observation\_indicator** A character string specifying the column name in new\_confirm that indicates whether an observation is observed or not. This column should be a logical vector. If NULL (default), all observations are considered observed.

## Value

A list containing:

- st: time index of each snapshot (snapshot time).
- ts: snapshot index by time and group.
- sl: number of reported observations per snapshot (snapshot length).
- csl: cumulative version of sl.
- lsl: number of consecutive reported observations per snapshot accounting for missing data.
- clsl: cumulative version of lsl.
- nsl: number of observed observations per snapshot (snapshot length).
- cnsl: cumulative version of nsl.
- sg: group index of each snapshot (snapshot group).

**See Also**

Helper functions for model modules [add\\_max\\_observed\\_delay\(\)](#), [add\\_pmfs\(\)](#), [convolution\\_matrix\(\)](#), [enw\\_reference\\_by\\_report\(\)](#), [enw\\_reps\\_with\\_complete\\_refs\(\)](#), [extract\\_sparse\\_matrix\(\)](#), [latest\\_obs\\_as\\_matrix\(\)](#), [simulate\\_double\\_censored\\_pmf\(\)](#)

---

extract\_sparse\_matrix *Extract sparse matrix elements*

---

**Description**

This helper function allows the extraction of a sparse matrix from a matrix using a similar approach to that implemented in [rstan::extract\\_sparse\\_parts\(\)](#) and returns these elements in a named list for use in stan. This function is used in the construction of the expectation model (see [enw\\_expectation\(\)](#)).

**Usage**

```
extract_sparse_matrix(mat, prefix = "")
```

**Arguments**

mat	A matrix to extract the sparse matrix from.
prefix	A character string to prefix the names of the returned list.

**Value**

A list representing the sparse matrix, containing:

- nw: Count of non-zero elements in mat.
- w: Vector of non-zero elements in mat. Equivalent to the numeric values from mat excluding zeros.
- nv: Length of v.
- v: Vector of row indices corresponding to each non-zero element in w. Indicates the row location in mat for each non-zero value.
- nu: Length of u.
- u: Vector indicating the starting indices in w for non-zero elements of each row in mat. Helps identify the partition of w into different rows of mat.

**See Also**

[enw\\_expectation\(\)](#)

Helper functions for model modules [add\\_max\\_observed\\_delay\(\)](#), [add\\_pmfs\(\)](#), [convolution\\_matrix\(\)](#), [enw\\_reference\\_by\\_report\(\)](#), [enw\\_reps\\_with\\_complete\\_refs\(\)](#), [extract\\_obs\\_metadata\(\)](#), [latest\\_obs\\_as\\_matrix\(\)](#), [simulate\\_double\\_censored\\_pmf\(\)](#)

**Examples**

```
mat <- matrix(1:12, nrow = 4)
mat[2, 2] <- 0
mat[3, 1] <- 0
extract_sparse_matrix(mat)
```

---

germany\_covid19\_hosp *Hospitalisations in Germany by date of report and reference*

---

**Description**

Hospitalisations in Germany by date of report and reference

**Usage**

```
germany_covid19_hosp
```

**Format**

An object of class `data.table` (inherits from `data.frame`) with 1536885 rows and 5 columns.

**Value**

A `data.table`

**See Also**

Package data sets [enw\\_example\(\)](#)

---

`get_internal_timestep` *Get internal timestep*

---

**Description**

This function converts the string representation of the timestep to its corresponding numeric value or returns the numeric input (if it is a whole number). For "day", "week", it returns 1 and 7 respectively. For "month", it returns "month" as months are not a fixed number of days. If the input is a numeric whole number, it is returned as is.

**Usage**

```
get_internal_timestep(timestep)
```

**Arguments**

`timestep` The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.

**Value**

A numeric value representing the number of days for "day" and "week", "month" for "month", or the input value if it is a numeric whole number.

**See Also**

Utility functions [aggregate\\_rolling\\_sum\(\)](#), [coerce\\_date\(\)](#), [coerce\\_dt\(\)](#), [date\\_to\\_numeric\\_modulus\(\)](#), [is.Date\(\)](#), [stan\\_fns\\_as\\_string\(\)](#)

---

is.Date	<i>Check an object is a Date</i>
---------	----------------------------------

---

**Description**

Checks that an object is a date

**Usage**

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

**Arguments**

x                    An object

**Value**

A logical

**See Also**

Utility functions [aggregate\\_rolling\\_sum\(\)](#), [coerce\\_date\(\)](#), [coerce\\_dt\(\)](#), [date\\_to\\_numeric\\_modulus\(\)](#), [get\\_internal\\_timestep\(\)](#), [stan\\_fns\\_as\\_string\(\)](#)

---

latest_obs_as_matrix	<i>Convert latest observed data to a matrix</i>
----------------------	---

---

**Description**

Convert latest observed data to a matrix

**Usage**

```
latest_obs_as_matrix(latest)
```

**Arguments**

latest                latest data.frame output from [enw\\_preprocess\\_data\(\)](#).

**Value**

A matrix with each column being a group and each row a reference date

**See Also**

Helper functions for model modules `add_max_observed_delay()`, `add_pmfs()`, `convolution_matrix()`, `enw_reference_by_report()`, `enw_reps_with_complete_refs()`, `extract_obs_metadata()`, `extract_sparse_matrix()`, `simulate_double_censored_pmf()`

---

parse_formula	<i>Parse a formula into components</i>
---------------	--

---

**Description**

This function uses a series internal functions to break an input formula into its component parts each of which can then be handled separately. Currently supported components are fixed effects, `lme4` style random effects, and random walks using the `rw()` helper function.

**Usage**

```
parse_formula(formula)
```

**Arguments**

`formula` A model formula that may use standard fixed effects, random effects using `lme4` syntax (see `re()`), and random walks defined using the `rw()` helper function.

**Value**

A list of formula components. These currently include:

- `fixed`: A character vector of fixed effect terms
- `random`: A list of of `lme4` style random effects
- `rw`: A character vector of `rw()` random walk terms.

**Reference**

The random walk functions used internally by this function were adapted from code written by J Scott (under an MIT license) as part of the `epidemia` package (<https://github.com/ImperialCollegeLondon/epidemia/>).

**See Also**

Functions used to help convert formulas into model designs `as_string_formula()`, `construct_re()`, `construct_rw()`, `enw_formula()`, `enw_manual_formula()`, `remove_rw_terms()`, `re()`, `rw_terms()`, `rw()`, `split_formula_to_terms()`

**Examples**

```
epinowcast:::parse_formula(~ 1 + age_group + location)

epinowcast:::parse_formula(~ 1 + age_group + (1 | location))

epinowcast:::parse_formula(~ 1 + (age_group | location))

epinowcast:::parse_formula(~ 1 + (1 | location) + rw(week, location))
```

---

plot.epinowcast	<i>Plot method for epinowcast</i>
-----------------	-----------------------------------

---

**Description**

plot method for class "epinowcast".

**Usage**

```
## S3 method for class 'epinowcast'
plot(
  x,
  latest_obs = NULL,
  type = c("nowcast", "posterior_prediction"),
  log = FALSE,
  ...
)
```

**Arguments**

x	A data.table of output as produced by <code>epinowcast()</code> .
latest_obs	A data.frame of observed data which may be passed to lower level methods.
type	Character string indicating the plot required; enforced by <code>base::match.arg()</code> . Currently supported options: <ul style="list-style-type: none"> <li>"nowcast" which plots the nowcast for each dataset along with latest available observed data using <code>enw_plot_nowcast_quantiles()</code>,</li> <li>"posterior_prediction" which plots observations reported at the time against simulated observations from the model using <code>enw_plot_pp_quantiles()</code>.</li> </ul>
log	Logical, defaults to FALSE. Should counts be plot on the log scale.
...	Additional arguments to the plot function specified by type.

**Value**

ggplot2 object



**See Also**

Other `epinowcast`: [epinowcast\(\)](#), [summary.epinowcast\(\)](#)

Plotting functions [enw\\_plot\\_nowcast\\_quantiles\(\)](#), [enw\\_plot\\_obs\(\)](#), [enw\\_plot\\_pp\\_quantiles\(\)](#), [enw\\_plot\\_quantiles\(\)](#), [enw\\_plot\\_theme\(\)](#)

**Examples**

```
nowcast <- enw_example("nowcast")
latest_obs <- enw_example("obs")

# Plot nowcast
plot(nowcast, latest_obs = latest_obs, type = "nowcast")

# Plot posterior predictions by reference date
plot(nowcast, type = "posterior_prediction") +
  ggplot2::facet_wrap(ggplot2::vars(reference_date), scales = "free")
```

---

re

*Defines random effect terms using the lme4 syntax*


---

**Description**

Defines random effect terms using the `lme4` syntax

**Usage**

```
re(formula)
```

**Arguments**

formula	A random effect as returned by <a href="#">lme4::findbars()</a> when a random effect is defined using the <code>lme4</code> syntax in formula. Currently only simplified random effects (i.e LHS   RHS) are supported.
---------	--

**Value**

A list defining the fixed and random effects of the specified random effect

**See Also**

Functions used to help convert formulas into model designs [as\\_string\\_formula\(\)](#), [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_formula\(\)](#), [enw\\_manual\\_formula\(\)](#), [parse\\_formula\(\)](#), [remove\\_rw\\_terms\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#), [split\\_formula\\_to\\_terms\(\)](#)

**Examples**

```
form <- epinowcast:::parse_formula(~ 1 + (1 | age_group))
re(form$random[[1]])

form <- epinowcast:::parse_formula(~ 1 + (location | age_group))
re(form$random[[1]])
```

---

remove_profiling	<i>Remove profiling statements from a character vector representing stan code</i>
------------------	---

---

**Description**

Remove profiling statements from a character vector representing stan code

**Usage**

```
remove_profiling(s)
```

**Arguments**

s                      Character vector representing stan code

**Value**

A character vector of the stan code without profiling statements

**See Also**

Functions used to help convert models into the format required for stan [enw\\_formula\\_as\\_data\\_list\(\)](#), [enw\\_get\\_cache\(\)](#), [enw\\_model\(\)](#), [enw\\_priors\\_as\\_data\\_list\(\)](#), [enw\\_replace\\_priors\(\)](#), [enw\\_sample\(\)](#), [enw\\_set\\_cache\(\)](#), [enw\\_stan\\_to\\_r\(\)](#), [enw\\_unset\\_cache\(\)](#), [write\\_stan\\_files\\_no\\_profile\(\)](#)

---

remove_rw_terms	<i>Remove random walk terms from a formula object</i>
-----------------	---

---

**Description**

This function removes random walk terms denoted using `rw()` from a formula so that they can be processed on their own.

**Usage**

```
remove_rw_terms(formula)
```

**Arguments**

`formula` A model formula that may use standard fixed effects, random effects using [lme4](#) syntax (see [re\(\)](#)), and random walks defined using the [rw\(\)](#) helper function.

**Value**

A formula object with the random walk terms removed.

**Reference**

This function was adapted from code written by J Scott (under an MIT license) as part of the `epidemia` package (<https://github.com/ImperialCollegeLondon/epidemia/>).

**See Also**

Functions used to help convert formulas into model designs [as\\_string\\_formula\(\)](#), [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_formula\(\)](#), [enw\\_manual\\_formula\(\)](#), [parse\\_formula\(\)](#), [re\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#), [split\\_formula\\_to\\_terms\(\)](#)

**Examples**

```
epinowcast:::remove_rw_terms(~ 1 + age_group + location)

epinowcast:::remove_rw_terms(~ 1 + age_group + location + rw(week, location))
```

---

 rw

*Adds random walks with Gaussian steps to the model.*

---

**Description**

A call to `rw()` can be used in the 'formula' argument of model construction functions in the `epinowcast` package such as [enw\\_formula\(\)](#). Does not evaluate arguments but instead simply passes information for use in model construction.

**Usage**

```
rw(time, by, type = c("independent", "dependent"))
```

**Arguments**

`time` Defines the random walk time period.

`by` Defines the grouping parameter used for the random walk. If not specified no grouping is used. Currently this is limited to a single variable.

`type` Character string, how standard deviation of grouped random walks is estimated: "independent", or "dependent" across groups; enforced by [base::match.arg\(\)](#).

**Value**

A list defining the time frame, group, and type with class "enw\_rw\_term" that can be interpreted by `construct_rw()`.

**See Also**

Functions used to help convert formulas into model designs `as_string_formula()`, `construct_re()`, `construct_rw()`, `enw_formula()`, `enw_manual_formula()`, `parse_formula()`, `remove_rw_terms()`, `re()`, `rw_terms()`, `split_formula_to_terms()`

**Examples**

```
rw(time)

rw(time, location)

rw(time, location, type = "dependent")
```

---

`rw_terms`*Finds random walk terms in a formula object*

---

**Description**

This function extracts random walk terms denoted using `rw()` from a formula so that they can be processed on their own.

**Usage**

```
rw_terms(formula)
```

**Arguments**

`formula` A model formula that may use standard fixed effects, random effects using `lme4` syntax (see `re()`), and random walks defined using the `rw()` helper function.

**Value**

A character vector containing the random walk terms that have been identified in the supplied formula.

**Reference**

This function was adapted from code written by J Scott (under an MIT license) as part of the `epidemia` package (<https://github.com/ImperialCollegeLondon/epidemia/>).

**See Also**

Functions used to help convert formulas into model designs `as_string_formula()`, `construct_re()`, `construct_rw()`, `enw_formula()`, `enw_manual_formula()`, `parse_formula()`, `remove_rw_terms()`, `re()`, `rw()`, `split_formula_to_terms()`

**Examples**

```
epinowcast:::rw_terms(~ 1 + age_group + location)
```

```
epinowcast:::rw_terms(~ 1 + age_group + location + rw(week, location))
```

---

```
simulate_double_censored_pmf
```

*Simulate daily double censored PMF*

---

**Description**

This function simulates the probability mass function of a daily double-censored process. The process involves two distributions: a primary distribution which represents the censoring process for the primary event and another distribution (which is offset by the primary).

**Usage**

```
simulate_double_censored_pmf(
  max,
  fun_primary = stats::runif,
  primary_args = list(),
  fun_dist = stats::rlnorm,
  dist_args = list(...),
  n = 1e+06,
  ...
)
```

**Arguments**

<code>max</code>	Maximum value for the computed CDF. If not specified, the maximum value is the maximum simulated delay.
<code>fun_primary</code>	Primary distribution function (default is <code>runif</code> ).
<code>primary_args</code>	List of additional arguments to be passed to the primary distribution function.
<code>fun_dist</code>	Distribution function to be added to the primary (default is <code>rlnorm</code> ).
<code>dist_args</code>	List of additional arguments to be passed to the distribution function.
<code>n</code>	Number of simulations (default is <code>1e6</code> ).
<code>...</code>	Additional arguments to be passed to the distribution function. This is an alternative to <code>dist_args</code> .

**Value**

A numeric vector representing the PMF.

**See Also**

Helper functions for model modules [add\\_max\\_observed\\_delay\(\)](#), [add\\_pmfs\(\)](#), [convolution\\_matrix\(\)](#), [enw\\_reference\\_by\\_report\(\)](#), [enw\\_reps\\_with\\_complete\\_refs\(\)](#), [extract\\_obs\\_metadata\(\)](#), [extract\\_sparse\\_matrix\(\)](#), [latest\\_obs\\_as\\_matrix\(\)](#)

**Examples**

```
simulate_double_censored_pmf(10, meanlog = 0, sdlog = 1)
```

---

```
split_formula_to_terms
```

*Split formula into individual terms*

---

**Description**

Split formula into individual terms

**Usage**

```
split_formula_to_terms(formula)
```

**Arguments**

formula	A model formula that may use standard fixed effects, random effects using <a href="#">lme4</a> syntax (see <a href="#">re()</a> ), and random walks defined using the <a href="#">rw()</a> helper function.
---------	---

**Value**

A character vector of formula terms

**See Also**

Functions used to help convert formulas into model designs [as\\_string\\_formula\(\)](#), [construct\\_re\(\)](#), [construct\\_rw\(\)](#), [enw\\_formula\(\)](#), [enw\\_manual\\_formula\(\)](#), [parse\\_formula\(\)](#), [remove\\_rw\\_terms\(\)](#), [re\(\)](#), [rw\\_terms\(\)](#), [rw\(\)](#)

**Examples**

```
epinowcast:::split_formula_to_terms(~ 1 + age_group + location)
```

---

stan\_fns\_as\_string      *Read in a stan function file as a character string*

---

**Description**

Read in a stan function file as a character string

**Usage**

```
stan_fns_as_string(files, include)
```

**Arguments**

files	A character vector specifying the names of Stan files to be exposed. These must be in the include directory. Defaults to all Stan files in the include directory. Note that the following files contain overloaded functions and cannot be exposed: "delay_lpmf.stan", "allocate_observed_obs.stan", "obs_lpmf.stan", and "effects_priors_lp.stan".
include	A character string specifying the directory containing Stan files. Defaults to the 'stan/functions' directory of the <code>epinowcast()</code> package.

**Value**

A character string in the of stan functions.

**See Also**

Utility functions `aggregate_rolling_sum()`, `coerce_date()`, `coerce_dt()`, `date_to_numeric_modulus()`, `get_internal_timestep()`, `is.Date()`

---

summary.epinowcast      *Summary method for epinowcast*

---

**Description**

summary method for class "epinowcast".

**Usage**

```
## S3 method for class 'epinowcast'
summary(
  object,
  type = c("nowcast", "nowcast_samples", "fit", "posterior_prediction"),
  max_delay = object$max_delay,
  ...
)
```

**Arguments**

object	A data.table output from <code>epinowcast()</code> .
type	Character string indicating the summary to return; enforced by <code>base::match.arg()</code> . Supported options are: <ul style="list-style-type: none"> <li>• "nowcast" which summarises nowcast posterior with <code>enw_nowcast_summary()</code>,</li> <li>• "nowcast_samples" which samples latest with <code>enw_nowcast_samples()</code>,</li> <li>• "fit" which returns the summarised cmdstanr fit with <code>enw_posterior()</code>,</li> <li>• "posterior_prediction" which returns summarised posterior predictions for the observations after fitting using <code>enw_pp_summary()</code>.</li> </ul>
max_delay	Maximum delay to which nowcasts should be summarised. Must be equal (default) or larger than the modelled maximum delay. If it is larger, then nowcasts for unmodelled dates are added by assuming that case counts beyond the modelled maximum delay are fully observed.
...	Additional arguments passed to summary specified by type.

**Value**

A summary data.frame

**See Also**

summary.epinowcast

Other epinowcast: `epinowcast()`, `plot.epinowcast()`

**Examples**

```
nowcast <- enw_example("nowcast")

# Summarise nowcast posterior
summary(nowcast, type = "nowcast")

# Nowcast posterior samples
summary(nowcast, type = "nowcast_samples")

# Nowcast model fit
summary(nowcast, type = "fit")

# Posterior predictions
summary(nowcast, type = "posterior_prediction")
```



---

`write_stan_files_no_profile`

*Write copies of the .stan files of a Stan model and its #include files with all profiling statements removed.*

---

### Description

Write copies of the .stan files of a Stan model and its #include files with all profiling statements removed.

### Usage

```
write_stan_files_no_profile(  
  stan_file,  
  include_paths = NULL,  
  target_dir = epinowcast::enw_get_cache()  
)
```

### Arguments

<code>stan_file</code>	The path to a .stan file containing a Stan program.
<code>include_paths</code>	Paths to directories where Stan should look for files specified in #include directives in the Stan program.
<code>target_dir</code>	The path to a directory in which the manipulated .stan files without profiling statements should be stored. To avoid overriding of the original .stan files, this should be different from the directory of the original model and the include_paths.

### Value

A list containing the path to the .stan file without profiling statements and the include\_paths for the included .stan files without profiling statements

### See Also

Functions used to help convert models into the format required for stan `enw_formula_as_data_list()`, `enw_get_cache()`, `enw_model()`, `enw_priors_as_data_list()`, `enw_replace_priors()`, `enw_sample()`, `enw_set_cache()`, `enw_stan_to_r()`, `enw_unset_cache()`, `remove_profiling()`

# Index

- \* **check**
  - check\_calendar\_timestep, 7
  - check\_group, 8
  - check\_group\_date\_unique, 9
  - check\_max\_delay, 9
  - check\_module, 11
  - check\_modules\_compatible, 11
  - check\_numeric\_timestep, 12
  - check\_observation\_indicator, 12
  - check\_quantiles, 13
  - check\_timestep, 14
  - check\_timestep\_by\_date, 15
  - check\_timestep\_by\_group, 15
- \* **dataconverters**
  - enw\_add\_cumulative, 22
  - enw\_add\_incidence, 25
  - enw\_aggregate\_cumulative, 30
  - enw\_incidence\_to\_linelist, 50
  - enw\_linelist\_to\_incidence, 52
- \* **datasets**
  - germany\_covid19\_hosp, 93
- \* **data**
  - enw\_example, 39
  - germany\_covid19\_hosp, 93
- \* **epinowcast**
  - epinowcast, 88
  - plot.epinowcast, 96
  - summary.epinowcast, 103
- \* **formulatools**
  - as\_string\_formula, 7
  - construct\_re, 18
  - construct\_rw, 19
  - enw\_formula, 46
  - enw\_manual\_formula, 53
  - parse\_formula, 95
  - re, 97
  - remove\_rw\_terms, 98
  - rw, 99
  - rw\_terms, 100
  - split\_formula\_to\_terms, 102
- \* **modeldesign**
  - enw\_add\_cumulative\_membership, 23
  - enw\_add\_pooling\_effect, 29
  - enw\_design, 37
  - enw\_effects\_metadata, 38
  - enw\_one\_hot\_encode\_feature, 63
- \* **modelmodulehelpers**
  - add\_max\_observed\_delay, 4
  - add\_pmf, 5
  - convolution\_matrix, 20
  - enw\_reference\_by\_report, 75
  - enw\_reps\_with\_complete\_refs, 80
  - extract\_obs\_metadata, 91
  - extract\_sparse\_matrix, 92
  - latest\_obs\_as\_matrix, 94
  - simulate\_double\_censored\_pmf, 101
- \* **modelmodules**
  - enw\_expectation, 40
  - enw\_fit\_opts, 44
  - enw\_missing, 57
  - enw\_obs, 62
  - enw\_reference, 74
  - enw\_report, 77
- \* **modeltools**
  - enw\_formula\_as\_data\_list, 48
  - enw\_get\_cache, 49
  - enw\_model, 59
  - enw\_priors\_as\_data\_list, 72
  - enw\_replace\_priors, 76
  - enw\_sample, 81
  - enw\_set\_cache, 83
  - enw\_stan\_to\_r, 85
  - enw\_unset\_cache, 88
  - remove\_profiling, 98
  - write\_stan\_files\_no\_profile, 105
- \* **modelvalidation**
  - enw\_score\_nowcast, 81
- \* **plot**

- enw\_plot\_nowcast\_quantiles, 64
  - enw\_plot\_obs, 65
  - enw\_plot\_pp\_quantiles, 66
  - enw\_plot\_quantiles, 66
  - enw\_plot\_theme, 67
  - plot.epinowcast, 96
  - \* **postprocess**
    - enw\_add\_latest\_obs\_to\_nowcast, 26
    - enw\_nowcast\_samples, 60
    - enw\_nowcast\_summary, 61
    - enw\_posterior, 68
    - enw\_pp\_summary, 69
    - enw\_quantiles\_to\_long, 73
    - enw\_summarise\_samples, 87
  - \* **preprocess**
    - enw\_add\_delay, 24
    - enw\_add\_max\_reported, 27
    - enw\_add\_metaobs\_features, 28
    - enw\_assign\_group, 32
    - enw\_complete\_dates, 33
    - enw\_construct\_data, 35
    - enw\_extend\_date, 41
    - enw\_filter\_reference\_dates, 42
    - enw\_filter\_report\_dates, 43
    - enw\_flag\_observed\_observations, 46
    - enw\_impute\_na\_observations, 49
    - enw\_latest\_data, 51
    - enw\_metadata, 55
    - enw\_metadata\_delay, 56
    - enw\_missing\_reference, 58
    - enw\_preprocess\_data, 70
    - enw\_reporting\_triangle, 78
    - enw\_reporting\_triangle\_to\_long, 79
  - \* **simulate**
    - enw\_simulate\_missing\_reference, 84
  - \* **utils**
    - aggregate\_rolling\_sum, 6
    - coerce\_date, 16
    - coerce\_dt, 17
    - date\_to\_numeric\_modulus, 21
    - get\_internal\_timestep, 93
    - is.Date, 94
    - stan\_fns\_as\_string, 103
- add\_max\_observed\_delay, 4, 5, 21, 76, 80, 92, 95, 102
  - add\_max\_observed\_delay(), 4, 91
  - add\_pmfs, 5, 5, 21, 76, 80, 92, 95, 102
  - aggregate\_rolling\_sum, 6, 17, 18, 21, 94, 103
  - as.integer(), 71
  - as\_string\_formula, 7, 19, 20, 47, 54, 95, 97, 99–102
  - available\_metrics(), 83
  - base::match.arg(), 45, 63, 96, 99, 104
  - check\_calendar\_timestep, 7, 8–16
  - check\_forecasts(), 82
  - check\_group, 8, 8, 9–16
  - check\_group\_date\_unique, 8, 9, 10–16
  - check\_max\_delay, 8, 9, 9, 11–16
  - check\_max\_delay(), 10, 33, 56, 71, 76, 80
  - check\_module, 8–11, 11, 12–16
  - check\_modules\_compatible, 8–11, 11, 12–16
  - check\_numeric\_timestep, 8–11, 12, 13–16
  - check\_observation\_indicator, 8–12, 12, 13–16
  - check\_quantiles, 8–13, 13, 14–16
  - check\_timestep, 8–13, 14, 15, 16
  - check\_timestep\_by\_date, 8–14, 15, 16
  - check\_timestep\_by\_group, 8–15, 15
  - cmdstanr::cmdstan\_model(), 59, 60
  - cmdstanr::CmdStanModel, 85, 86
  - cmdstanr::sample(), 45
  - coerce\_date, 6, 16, 18, 21, 94, 103
  - coerce\_date(), 28
  - coerce\_dt, 6, 17, 17, 21, 94, 103
  - coerce\_dt(), 13
  - construct\_re, 7, 18, 20, 47, 54, 95, 97, 99–102
  - construct\_rw, 7, 19, 19, 47, 54, 95, 97, 99–102
  - construct\_rw(), 100
  - convolution\_matrix, 5, 20, 76, 80, 92, 95, 102
  - data.table, 17
  - data.table::as.data.table(), 14–17, 28
  - data.table::as.IDate(), 16, 18
  - Date, 55
  - date\_to\_numeric\_modulus, 6, 17, 18, 21, 94, 103
  - Dates, 28
  - enw\_add\_cumulative, 22, 25, 32, 51, 53

- enw\_add\_cumulative\_membership, 23, 30, 37, 38, 64
- enw\_add\_cumulative\_membership(), 19, 20
- enw\_add\_delay, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_add\_incidence, 22, 25, 32, 51, 53
- enw\_add\_incidence(), 58, 78
- enw\_add\_latest\_obs\_to\_nowcast, 26, 61, 62, 68, 69, 73, 87
- enw\_add\_max\_reported, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_add\_metaobs\_features, 24, 27, 28, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_add\_metaobs\_features(), 41, 71
- enw\_add\_pooling\_effect, 23, 29, 37, 38, 64
- enw\_aggregate\_cumulative, 22, 25, 30, 51, 53
- enw\_aggregate\_cumulative(), 36, 71
- enw\_assign\_group, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_complete\_dates, 24, 27, 29, 32, 33, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_complete\_dates(), 31, 63, 70
- enw\_construct\_data, 24, 27, 29, 32, 34, 35, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_cumulative\_to\_incidence, 22, 25, 32, 51, 53
- enw\_design, 23, 30, 37, 38, 64
- enw\_effects\_metadata, 23, 30, 37, 38, 64
- enw\_effects\_metadata(), 29, 30
- enw\_example, 39, 93
- enw\_expectation, 40, 45, 57, 63, 74, 78
- enw\_expectation(), 11, 81, 89, 92
- enw\_extend\_date, 24, 27, 29, 32, 34, 36, 41, 43, 44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_filter\_delay, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_filter\_reference\_dates, 24, 27, 29, 32, 34, 36, 42, 42, 44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_filter\_reference\_dates(), 43
- enw\_filter\_report\_dates, 24, 27, 29, 32, 34, 36, 42, 43, 43, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_filter\_report\_dates(), 42
- enw\_fit\_opts, 41, 44, 57, 63, 74, 78
- enw\_fit\_opts(), 89
- enw\_flag\_observed\_observations, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_flag\_observed\_observations(), 34, 63
- enw\_formula, 7, 19, 20, 46, 54, 95, 97, 99–102
- enw\_formula(), 40, 48, 53, 57, 74, 77, 99
- enw\_formula\_as\_data\_list, 48, 49, 60, 73, 77, 81, 84, 86, 88, 98, 105
- enw\_get\_cache, 49, 49, 60, 73, 77, 81, 84, 86, 88, 98, 105
- enw\_impute\_na\_observations, 24, 27, 29, 32, 34, 36, 42–44, 46, 49, 52, 55, 57, 58, 72, 79
- enw\_incidence\_to\_cumulative, 22, 25, 32, 51, 53
- enw\_incidence\_to\_linelist, 22, 25, 32, 50, 53
- enw\_latest\_data, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 51, 55, 57, 58, 72, 79
- enw\_latest\_data(), 26, 39, 60, 62, 82
- enw\_linelist\_to\_incidence, 22, 25, 32, 51, 52
- enw\_manual\_formula, 7, 19, 20, 47, 53, 95, 97, 99–102
- enw\_metadata, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_metadata\_delay, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 56, 58, 72, 79
- enw\_metadata\_delay(), 35, 36, 71
- enw\_missing, 41, 45, 57, 63, 74, 78
- enw\_missing(), 45, 89
- enw\_missing\_reference, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79
- enw\_model, 49, 59, 73, 77, 81, 84, 86, 88, 98, 105
- enw\_model(), 81, 89
- enw\_nowcast\_samples, 26, 60, 62, 68, 69, 73, 87
- enw\_nowcast\_samples(), 104
- enw\_nowcast\_summary, 26, 61, 61, 68, 69, 73, 87
- enw\_nowcast\_summary(), 26, 104
- enw\_obs, 41, 45, 57, 62, 74, 78
- enw\_obs(), 34, 81, 89
- enw\_one\_hot\_encode\_feature, 23, 30, 37,

- 38, 63
- enw\_plot\_nowcast\_quantiles, 64, 65–68, 97
- enw\_plot\_nowcast\_quantiles(), 67, 96
- enw\_plot\_obs, 64, 65, 66–68, 97
- enw\_plot\_pp\_quantiles, 64, 65, 66, 67, 68, 97
- enw\_plot\_pp\_quantiles(), 64, 66, 67, 96
- enw\_plot\_quantiles, 64–66, 66, 68, 97
- enw\_plot\_theme, 64–67, 67, 97
- enw\_posterior, 26, 61, 62, 68, 69, 73, 87
- enw\_posterior(), 61, 73, 104
- enw\_pp\_summary, 26, 61, 62, 68, 69, 73, 87
- enw\_pp\_summary(), 104
- enw\_preprocess\_data, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 70, 79
- enw\_preprocess\_data(), 4, 10, 26, 28, 35, 39–41, 52, 56, 57, 60, 62, 63, 74, 75, 77, 78, 80, 89–91, 94
- enw\_priors\_as\_data\_list, 49, 60, 72, 77, 81, 84, 86, 88, 98, 105
- enw\_quantiles\_to\_long, 26, 61, 62, 68, 69, 73, 87
- enw\_reference, 41, 45, 57, 63, 74, 78
- enw\_reference(), 72, 76, 77, 89
- enw\_reference\_by\_report, 5, 21, 75, 80, 92, 95, 102
- enw\_replace\_priors, 49, 60, 73, 76, 81, 84, 86, 88, 98, 105
- enw\_report, 41, 45, 57, 63, 74, 77
- enw\_report(), 89
- enw\_reporting\_triangle, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 78, 79
- enw\_reporting\_triangle(), 79
- enw\_reporting\_triangle\_to\_long, 24, 27, 29, 32, 34, 36, 42–44, 46, 50, 52, 55, 57, 58, 72, 79, 79
- enw\_reps\_with\_complete\_refs, 5, 21, 76, 80, 92, 95, 102
- enw\_reps\_with\_complete\_refs(), 75
- enw\_sample, 49, 60, 73, 77, 81, 84, 86, 88, 98, 105
- enw\_sample(), 45, 89
- enw\_score\_nowcast, 81
- enw\_set\_cache, 49, 60, 73, 77, 81, 83, 86, 88, 98, 105
- enw\_simulate\_missing\_reference, 84
- enw\_stan\_to\_r, 49, 60, 73, 77, 81, 84, 85, 88, 98, 105
- enw\_summarise\_samples, 26, 61, 62, 68, 69, 73, 87
- enw\_unset\_cache, 49, 60, 73, 77, 81, 84, 86, 88, 98, 105
- epinowcast, 16, 85, 88, 97, 104
- epinowcast(), 25, 39, 45, 52, 60, 61, 69, 71, 76, 82, 86, 96, 103, 104
- example\_binary, 82
- example\_continuous, 82
- example\_integer, 82
- example\_quantile, 82
- expose\_functions(), 85, 86
- extract\_obs\_metadata, 5, 21, 76, 80, 91, 92, 95, 102
- extract\_sparse\_matrix, 5, 21, 76, 80, 92, 92, 95, 102
- formula, 37
- germany\_covid19\_hosp, 39, 93
- get\_internal\_timestep, 6, 17, 18, 21, 93, 94, 103
- ggplot2::aes(), 65, 67
- IDate, 16, 17
- is.Date, 6, 17, 18, 21, 94, 94, 103
- latest\_obs\_as\_matrix, 5, 21, 76, 80, 92, 94, 102
- lme4, 7, 18, 46, 47, 95, 97, 99, 100, 102
- lme4::findbars(), 97
- metrics, 83
- parse\_formula, 7, 19, 20, 47, 54, 95, 97, 99–102
- parse\_formula(), 47
- plot.epinowcast, 64–68, 90, 96, 104
- posterior::draws\_df(), 60
- posterior::summarise\_draws(), 68
- re, 7, 19, 20, 47, 54, 95, 97, 99–102
- re(), 7, 18, 47, 95, 99, 100, 102
- remove\_profiling, 49, 60, 73, 77, 81, 84, 86, 88, 98, 105
- remove\_rw\_terms, 7, 19, 20, 47, 54, 95, 97, 98, 100–102
- rstan::extract\_sparse\_parts(), 92

`rw`, 7, 19, 20, 47, 54, 95, 97, 99, 99, 101, 102  
`rw()`, 7, 19, 23, 47, 95, 98–100, 102  
`rw_terms`, 7, 19, 20, 47, 54, 95, 97, 99, 100,  
100, 102

`score()`, 82  
`scoringutils::check_forecasts()`, 82  
`scoringutils::score`, 82  
`scoringutils::score()`, 81, 83  
`simulate_double_censored_pmf`, 5, 21, 76,  
80, 92, 95, 101  
`split_formula_to_terms`, 7, 19, 20, 47, 54,  
95, 97, 99–101, 102  
`stan_fns_as_string`, 6, 17, 18, 21, 94, 103  
`startsWith()`, 30  
`stats::model.matrix`, 37  
`stats::model.matrix()`, 37, 38  
`summary.epinowcast`, 90, 97, 103  
`summary.epinowcast()`, 60–62, 69, 82

`tempdir()`, 49  
`terms`, 37

`weekdays()`, 28  
`write_stan_files_no_profile`, 49, 60, 73,  
77, 81, 84, 86, 88, 98, 105