Package: epinowcast (via r-universe)

February 6, 2025

Title Flexible Hierarchical Nowcasting

Version 0.3.0.1000

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 (>= 4.1.0)

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

Suggests bookdown, dplyr, loo (>= 2.4.1), primarycensored, knitr, RcppEigen, rmarkdown, spelling, testthat (>= 3.1.9), usethis, vdiffr, withr

Remotes epiforecasts/scoringutils

VignetteBuilder knitr

Additional_repositories `https://stan-dev.r-universe.dev`

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 7.3.2

SystemRequirements CmdStan (>=2.29)

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

RemoteUrl https://github.com/epinowcast/epinowcast

RemoteRef HEAD

RemoteSha 7a9ebcd972d8098bce7fe1067f221c19721873af

Contents

add_max_observed_delay 4
add_pmfs 5
aggregate_rolling_sum 6
as_forecast_sample.epinowcast 7
as_string_formula 8
build_ord_obs 8
check_calendar_timestep 9
check_design_matrix_sparsity 10
check_group 11
check_group_date_unique 11
check_max_delay 12
check_module 13
check_modules_compatible 14
check_numeric_timestep 14
check_observation_indicator 15
check_quantiles 16
check_timestep 16
check_timestep_by_date 17
check_timestep_by_group 18
coerce_date 19
coerce_dt 20
construct_re 21
construct_rw 22
convolution_matrix 23
date_to_numeric_modulus 24
enw_add_cumulative 24
enw_add_cumulative_membership 25
enw_add_delay 26
enw_add_incidence 27
enw_add_latest_obs_to_nowcast 28

enw_add_max_reported 29
enw_add_metaobs_features 30
enw_add_pooling_effect 31
enw_aggregate_cumulative 32
enw_assign_group 34
enw_complete_dates 35
enw_construct_data 37
enw_design 39
enw_effects_metadata 40
enw_example 41
enw_expectation 42
enw_extend_date 43
enw_filter_reference_dates 44
enw_filter_report_dates 45
enw_fit_opts 46
enw_flag_observed_observations 48
enw_formula 48
enw_formula_as_data_list 50
enw_get_cache 51
enw_impute_na_observations 51
enw_incidence_to_linelist 52
enw_latest_data 53
enw_linelist_to_incidence 54
enw_manual_formula 55
enw_metadata 57
enw_metadata_delay 58
enw_missing 59
enw_missing_reference 60
enw_model 61
enw_nowcast_samples 62
enw_nowcast_summary 63
enw_obs 65
enw_one_hot_encode_feature 66
enw_pathfinder 66
enw_plot_nowcast_quantiles 68
enw_plot_obs 69
enw_plot_pp_quantiles 70
enw_plot_quantiles 70
enw_plot_theme 71
enw_posterior 72
enw_pp_summary 73
enw_preprocess_data 74
enw_priors_as_data_list 76
enw_quantiles_to_long 77
enw_reference 78
enw_reference_by_report 79
enw_replace_priors 80
enw_report 81

enw_reporting_triangle 82
enw_reporting_triangle_to_long 83
enw_reps_with_complete_refs 84
enw_sample 85
enw_score_nowcast 86
enw_set_cache 88
enw_simulate_missing_reference 89
enw_stan_to_r 90
enw_summarise_samples 91
enw_unset_cache 92
epinowcast 93
extract_obs_metadata 95
extract_sparse_matrix 96
germany_covid19_hosp 97
get_internal_timestep 98
is.Date 98
latest_obs_as_matrix 99
parse_formula 99
plot.epinowcast 100
re 101
remove_profiling 102
remove_rw_terms 103
rw 104
rw_terms 105
simulate_double_censored_pmf 106
split_formula_to_terms 107
stan_fns_as_string 107
subset_obs 108
summary.epinowcast 109
write_stan_files_no_profile 110

Index **112**

`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_pmfs()`, `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()`

`add_pmfs` *Add probability mass functions*

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_pmfs(pmfs)
```

Arguments

`pmfs`	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_pmfs(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_forecast_sample.epinowcast` *Convert an epinowcast object to a forecast_sample object*

Description

This function is used to convert an `epinowcast` as returned by `epinowcast()` object to a `forecast_sample` object which can be used for scoring using the `scoringutils` package.

Usage

```
## S3 method for class 'epinowcast'
as_forecast_sample(data, latest_obs, ...)
```

Arguments

`data`	An `epinowcast` nowcast object as returned by `epinowcast()`.
`latest_obs`	Latest observations to use for the true values must contain `confirm` and `observed` variables.
`...`	Additional arguments passed to `scoringutils::as_forecast_sample()`

Value

A `forecast_sample` object as returned by `scoringutils::as_forecast_sample()`

See Also

Other modelvalidation: `enw_score_nowcast()`

Examples

```
library(scoringutils)

nowcast <- enw_example("nowcast")
latest_obs <- enw_example("observations")
as_forecast_sample(nowcast, latest_obs)
```

`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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `split_formula_to_terms()`

Examples

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

`build_ord_obs` *Build the ord_obs* `data.table`.

Description

Build the ord_obs `data.table`.

Usage

```
build_ord_obs(obs, max_delay, internal_timestep, timestep, nowcast = NULL)
```

Arguments

`obs`	Observations as pulled from `nowcast$latest[[1]]`.
`max_delay`	Whole number representing the maximum delay in units of the timestep.
`internal_timestep`	The internal timestep in days.
`timestep`	The timestep to be used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
`nowcast`	If getting posterior samples, a data frame with a '.draws" column to get the draws from, as pulled from the fit attribute of a nowcast.

Value

A `data.table`.

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()`, `enw_summarise_samples()`, `subset_obs()`

`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_design_matrix_sparsity(), check_group(), check_group_date_unique(), check_max_delay(), check_module(), check_modules_compatible(), check_numeric_timestep(), check_observation_indicator(), check_quantiles(), check_timestep(), check_timestep_by_date(), check_timestep_by_group()

`check_design_matrix_sparsity` *Check design matrix sparsity*

Description

This function checks the sparsity of a design matrix and provides a recommendation if the matrix is considered sparse.

Usage

```
check_design_matrix_sparsity(
  matrix,
  sparsity_threshold = 0.9,
  min_matrix_size = 50,
  name = "checked"
)
```

Arguments

matrix	A numeric matrix to be checked for sparsity.
sparsity_threshold	A numeric value between 0 and 1 indicating the threshold for considering a matrix sparse. Default is 0.9.
min_matrix_size	An integer indicating the minimum size of the matrix for which to perform the sparsity check. Default is 50.
name	A character string specifying the name of the design matrix. Default is "checked".

Value

This function is used for its side effect of providing an informational message if the matrix is sparse. It returns NULL invisibly.

See Also

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

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 `coerce_dtd` 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_design_matrix_sparsity()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

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 `coerce_dtd` in place, that contains `.group`, `reference_date`, and `report_date` columns.

See Also

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

---

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 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.
`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 `cum_coverage` for the majority of reference dates?
`warn_internal`	Should only be `TRUE` if this function is called internally by another `epinowcast` 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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

Examples

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

`check_module` *Check a model module contains the required components*

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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

---

check_modules_compatible

Check that model modules have compatible specifications

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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

---

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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

`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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

`check_quantiles` *Check required quantiles are present*

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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_timestep()`, `check_timestep_by_date()`, `check_timestep_by_group()`

`check_timestep` *Check timestep*

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

`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`.
`check_nrow`	Logical, if `TRUE`, checks if there are at least two observations. Default is `TRUE`. If `FALSE`, 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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `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

`obs`	Any of the types supported by `data.table::as.data.table()`.
`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 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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_group()`

---

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_design_matrix_sparsity()`, `check_group()`, `check_group_date_unique()`, `check_max_delay()`, `check_module()`, `check_modules_compatible()`, `check_numeric_timestep()`, `check_observation_indicator()`, `check_quantiles()`, `check_timestep()`, `check_timestep_by_date()`

`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.table`*s*

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 `data.table::as.IDate()`.
`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` *Constructs random effect terms*

Description

Constructs random effect terms

Usage

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

Arguments

`re`	A random effect as defined using `re()` which itself takes random effects specified in a model formula using the lme4 syntax.
`data`	A `data.frame` of observations used to define the random effects. Must contain the variables specified in the `re()` 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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `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

`rw`	A random walk term as defined by `rw()`.
`data`	A `data.frame` of observations used to define the random walk term. Must contain the time and grouping variables defined in the `rw()` 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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `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 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

`dist`	A vector of list of vectors describing the distribution to be convolved as a probability mass function.
`t`	Integer value indicating the number of time steps to convolve over.
`include_partial`	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

`dt`	A data.table.
`date_column`	A character string representing the name of the date column in dt.
`timestep`	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

`metaobs`	A `data.frame` with a column named `feature` that contains a numeric vector of values.
`feature`	The name of the column in `metaobs` that contains the numeric vector of values.
`copy`	Should `metaobs` 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` *Add a delay variable to the observations*

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

`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).
`timestep`	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
`copy`	Should obs 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()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_lon`

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` *Calculate incidence of new reports from cumulative reports*

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 `epinowcast()`.
`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

nowcast	A data.frame of nowcast output from enw_nowcast_summary().
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 enw_preprocess_data() or enw_latest_data().

Value

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

See Also

Functions used for postprocessing of model fits build_ord_obs(), enw_nowcast_samples(), enw_nowcast_summary(), enw_posterior(), enw_pp_summary(), enw_quantiles_to_long(), enw_summarise_samples(), subset_obs()

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()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_lo`

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

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`	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). This is the output of `enw_effects_metadata()`.
`var_name`	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'.
`finder_fn`	A function that will be used to find the effects that match the string. Defaults to `startsWith()`. This can be any function that takes a `character` as it's first argument (the `effects$effects` column) and then any other other arguments in `...` and returns a logical vector indicating whether the effects were matched.
`...`	Additional arguments to `finder_fn`. E.g. for the `finder_fn = startsWith` default, this should be `prefix = "somestring"`.

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

`obs`	An object coercible to a `data.table` (such as a `data.frame`) which must have a `new_confirm` numeric column, and `report_date` and `reference_date` date columns. The input must have a timestep of a day and be complete. See `enw_complete_dates()` for more information. If NA values are present in the `confirm` column then these will be set to zero before aggregation this may not be desirable if this missingness is meaningful.
`timestep`	The timestep to used. This can be a string ("day", "week", "month") or a numeric whole number representing the number of days.
`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`.
`min_reference_date`	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 `obs` object. Other sensible values would be the minimum report date in the `obs` object + 1 day if reporting is already weekly and you wish to ensure that the timestep of the output matches the reporting timestep.
`copy`	Should `obs` 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")
```

---

enw_assign_group *Assign a group to each row of a data.table*

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

obs	A `data.table` or `data.frame` without a `.group` column.
by	A character vector of column names to group by. Defaults to an empty vector.
copy	A logical; make a copy (default) of obs 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(), enw_metadata_delay(), enw_missing_reference(), enw_preprocess_data(), enw_reporting_triangle(), enw_reporting_triangle_to_lo

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 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.

`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 `enw_flag_observed_observations()` 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 `enw_obs()` 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_da`
`enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`,
`enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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 `enw_metadata_delay()`.
`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 `enw_aggregate_cumulative()` 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_da` `enw_filter_report_dates()`, `enw_flag_observed_observations()`, `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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 `no_contrasts = FALSE` then all categorical variables will use contrasts. If `no_contrasts = TRUE` then no categorical variables will use contrasts.
`sparse`	Logical, if TRUE return a sparse design matrix. Defaults to TRUE.
`...`	Arguments passed on to `stats::model.matrix` `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 `ewn_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   *Load a package example*

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"))
```

`enw_extend_date` *Extend a time series with additional dates*

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()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_lon`

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_observat`
`enw_latest_data()`, `enw_metadata()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_lon`

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,
  sparse_design = FALSE,
  ...
)
```

Arguments

`sampler`	A function that creates an object that be used to extract posterior samples from the specified model. By default this is `enw_sample()` which makes use of `cmdstanr::sample()`.
`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 `threads_per_chain` is greater than 1; enforced by `base::match.arg()`. Currently supported options: • "snapshots" which aggregates over report dates and groups (i.e the lowest level that observations are reported at), • "groups" which aggregates across user defined groups. Note that some model modules override this setting depending on model requirements. For example, the `enw_missing()` 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.
`sparse_design`	Logical, defaults to `FALSE`. Should a sparse design matrices be used for all design matrices. This reduces memory requirements and may reduce computation time when fitting models with very sparse design matrices (90% or more zeros).
`...`	Additional arguments to pass to the fitting function being used by `epinowcast()`. By default this will be `enw_sample()` and so `cmdstanr` 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()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`,
`enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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 lme4 syntax (see `re()`), and random walks defined using the `rw()` 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 sparely 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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `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 `enw_formula()`.
`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 an intercept is present).
- `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_pathfinder()`, `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` *Retrieve Stan cache location*

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_pathfinder()`, `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` *Impute NA observations*

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_observa enw_latest_data(), enw_metadata(), enw_metadata_delay(), enw_missing_reference(), enw_preprocess_data(), enw_reporting_triangle(), enw_reporting_triangle_to_long()

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 `data.table` (such as a `data.frame`) which must have a `new_confirm` column.
reference_date	A character string of the variable name to use for the `reference_date` in the line list. The default is "reference_date".
report_date	A character string of the variable name to use for the `report_date` 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` *Filter observations to the latest available reported*

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 `data.frame`; must have `report_date` and `reference_date` 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_observat` `enw_impute_na_observations()`, `enw_metadata()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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 is 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

`linelist`	An object coercible to a `data.table` (such as a `data.frame`) where each row represents a case. Must contain at least two date variables or variables that can be coerced to dates.
`reference_date`	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 `reference_date` is the date of symptom onset then the `new_confirm` will be the number of new cases reported (based on `report_date`) on each day that had onset on that day. The default is "reference_date".
`report_date`	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

`data`	A `data.frame` of observations. It must include all variables used in the supplied formula.
`fixed`	A character vector of fixed effects.
`random`	A character vector of random effects. Random effects specified here will be added to the fixed effects.
`custom_random`	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.
`no_contrasts`	Logical, defaults to `FALSE`. `TRUE` means that no variable uses contrast. Alternatively a character vector of variables can be supplied indicating which variables should not have contrasts.
`add_intercept`	Logical, defaults to `FALSE`. 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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `split_formula_to_terms()`

Examples

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

enw_metadata *Extract metadata from raw data*

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

`obs`	A `data.frame` or `data.table` with columns: `reference_date` and / or `report_date`; at least one must be provided, `.group`, a grouping column and a `date`, a Date column.
`target_date`	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_observat` `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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

`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.
`breaks`	Numeric, defaults to 4. The number of breaks to use when constructing a categorised version of numeric delays.
`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 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_observat`
`enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata()`, `enw_missing_reference()`, `enw_preprocess_data()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

Examples

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

`enw_missing` *Missing reference data model module*

Description

Missing reference data model module

Usage

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

Arguments

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

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_observat
enw_impute_na_observations(), enw_latest_data(), enw_metadata(), enw_metadata_delay(), enw_preprocess_data(), enw_reporting_triangle(), enw_reporting_triangle_to_long()

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 `cmdstanr::cmdstan_model()`.
`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 `threads_to_chain` 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 `cmdstanr` documentation. # nolint
`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 `include_paths`.

`stanc_options`	A list of options to pass to the `stanc_options` of `cmdstanr::cmdstan_model()`. By default nothing is passed but potentially users may wish to pass optimisation flags for example. See the documentation for `cmdstanr::cmdstan_model()` for further details.
`cpp_options`	A list of options to pass to the `cpp_options` of `cmdstanr::cmdstan_model()`. By default nothing is passed but potentially users may wish to pass optimisation flags for example. See the documentation for `cmdstanr::cmdstan_model()` 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 `cmdstanr::cmdstan_model()`.

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_pathfinder()`, `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 `enw_preprocess_data()` or `enw_latest_data()`.
`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 `build_ord_obs()`, `enw_add_latest_obs_to_nowcast()`, `enw_nowcast_summary()`, `enw_posterior()`, `enw_pp_summary()`, `enw_quantiles_to_long()`, `enw_summarise_samples()`, `subset_obs()`

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 `enw_preprocess_data()` or `enw_latest_data()`.
`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 `build_ord_obs()`, `enw_add_latest_obs_to_nowcast()`, `enw_nowcast_samples()`, `enw_posterior()`, `enw_pp_summary()`, `enw_quantiles_to_long()`, `enw_summarise_samples()`, `subset_obs()`

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 `base::match.arg()`. 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 `enw_preprocess_data()`. It can be generated using `flag_observation` in `enw_complete_dates()` or it can be created directly using `enw_flag_observed_observations()`. If either of these approaches are used then the variable will be name `.observed`. Default is `NULL`.
`data`	Output from `enw_preprocess_data()`.

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_pathfinder

Fit a CmdStan model using the pathfinder algorithm

Description

For more information on the pathfinder algorithm see the CmdStan documentation. # nolint

Usage

```
enw_pathfinder(
  data,
  model = epinowcast::enw_model(),
  diagnostics = TRUE,
  init = NULL,
  ...
)
```

Arguments

`data`	A list of data as produced by model modules (for example `enw_expectation()`, `enw_obs()`, etc.) and as required for use the `model` being used.
`model`	A `cmdstanr` model object as loaded by `enw_model()` or as supplied by the user.
`diagnostics`	Logical, defaults to `TRUE`. Should fitting diagnostics be returned as a `data.frame`.
`init`	A list of initial values or a function to generate initial values. If not provided, the model will attempt to generate initial values
`...`	Additional parameters to be passed to `CmdStanModel$pathfinder()`.

Details

Note that the `threads_per_chain` argument is renamed to `num_threads` to match the `CmdStanModel$pathfinder()` method.

This fitting method is faster but more approximate than the NUTS sampler used in `enw_sample()` and as such is recommended for use in exploratory analysis and model development.

Value

A data.table containing the fit, data, and fit_args. If diagnostics is TRUE, it also includes the run_time column with the timing information.

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()`, `write_stan_files_no_profile()`

Examples

```
pobs <- enw_example("preprocessed")

nowcast <- epinowcast(pobs,
 expectation = enw_expectation(~1, data = pobs),
 fit = enw_fit_opts(enw_pathfinder, pp = TRUE),
 obs = enw_obs(family = "poisson", data = pobs),
)

summary(nowcast)
```

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 `enw_plot_pp_quantiles()`.

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 `ggplot2::aes()` 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 `enw_plot_nowcast_quantiles()` and `enw_plot_pp_quantiles()` 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 `ggplot2::aes()` 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` *Package plot theme*

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` *Summarise the posterior*

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

fit	A cmdstanr fit object.
variables	A character vector of variables to return posterior summaries for. By default summaries for all parameters are returned.
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.
...	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 build_ord_obs(), enw_add_latest_obs_to_nowcast(), enw_nowcast_samples(), enw_nowcast_summary(), enw_pp_summary(), enw_quantiles_to_long(), enw_summarise_samples(), subset_obs()

Examples

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

`enw_pp_summary` *Posterior predictive summary*

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

`fit`	A `cmdstanr` fit object.
`diff_obs`	A `data.frame` of observed data with at least a date variable `reference_date`, and a grouping variable `.group`.
`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.table summarising the posterior predictions.

See Also

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

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

obs	A `data.frame` containing at least the following variables: `reference_date` (index date of interest), `report_date` (report date for observations), `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. 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 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. 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).

`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 `enw_aggregate_cumulative()` to aggregate your data to the desired timestep.
`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 `epinowcast()`.
`...`	Other arguments to `enw_add_metaobs_features()`, e.g. `holidays`, which sets commonly used metadata (e.g. day of week, days since start of time series)
`copy`	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.
- `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_construct_data()`, `enw_extend_date()`, `enw_filter_delay()`, `enw_filter_reference_dates()`, `enw_filter_report_dates()`, `enw_flag_observed_observat` `enw_impute_na_observations()`, `enw_latest_data()`, `enw_metadata()`, `enw_metadata_delay()`, `enw_missing_reference()`, `enw_reporting_triangle()`, `enw_reporting_triangle_to_long()`

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_pathfinder()`, `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 `build_ord_obs()`, `enw_add_latest_obs_to_nowcast()`, `enw_nowcast_samples()`, `enw_nowcast_summary()`, `enw_posterior()`, `enw_pp_summary()`, `enw_summarise_samples(`
`subset_obs()`

Examples

```
fit <- enw_example("nowcast")
posterior <- enw_posterior(fit$fit[[1]], var = "expr_lelatent_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 `enw_formula()`) describing the parametric reference date delay model. This can use features defined by report date as defined in `metareference` as produced by `enw_preprocess_data()`. 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 `~ 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 `enw_formula()`) 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 `metareference` and `metadelay` as produced by `enw_preprocess_data()`. 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. `~ 0 + (1 | delay)`).
`data`	Output from `enw_preprocess_data()`.

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

`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.
`custom_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. Priors in this data.frame replace the default priors. Note that currently vectorised prior names (i.e those of the form `variable[n]` will be treated as `variable`).

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_pathfinder()`, `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 `enw_preprocess_data()`. 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 `enw_preprocess_data()`.

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_observat
enw_impute_na_observations(), enw_latest_data(), enw_metadata(), enw_metadata_delay(), 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_observat
enw_impute_na_observations(), enw_latest_data(), enw_metadata(), enw_metadata_delay(), 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 `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.
`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` *Fit a CmdStan model using NUTS*

Description

Fit a CmdStan model using NUTS

Usage

```
enw_sample(
  data,
  model = epinowcast::enw_model(),
  init = NULL,
  init_method = c("prior", "pathfinder"),
  init_method_args = list(),
  diagnostics = TRUE,
  ...
)
```

Arguments

`data`	A list of data as produced by model modules (for example `enw_expectation()`, `enw_obs()`, etc.) and as required for use the `model` being used.
`model`	A `cmdstanr` model object as loaded by `enw_model()` or as supplied by the user.
`init`	A list of initial values or a function to generate initial values. If not provided, the model will attempt to generate initial values
`init_method`	The method to use for initializing the model. Defaults to "prior" which samples initial values from the prior. "pathfinder", which uses the pathfinder algorithm (`enw_pathfinder()`) to initialize the model.
`init_method_args`	A list of additional arguments to pass to the initialization method.
`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_pathfinder()`, `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()`

Examples

```
pobs <- enw_example("preprocessed")

nowcast <- epinowcast(pobs,
 expectation = enw_expectation(~1, data = pobs),
 fit = enw_fit_opts(enw_sample, pp = TRUE),
 obs = enw_obs(family = "poisson", data = pobs),
)

summary(nowcast)

# Use pathfinder initialization
nowcast_pathfinder <- epinowcast(pobs,
 expectation = enw_expectation(~1, data = pobs),
 fit = enw_fit_opts(enw_sample, pp = TRUE, init_method = "pathfinder"),
 obs = enw_obs(family = "poisson", data = pobs),
)

summary(nowcast_pathfinder)
```

enw_score_nowcast *Evaluate nowcasts using proper scoring rules*

Description

[Deprecated]

This function is deprecated in favour of using `as_forecast_sample.epinowcast()` with `scoringutils::score()`. See the documentation for the `scoringutils` package for more details on 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 `summary.epinowcast()`, when used on the output of `epinowcast()`.
`latest_obs`	A `data.frame` of the latest available observations as produced by `enw_latest_data()` or otherwise.

`log`	Logical, defaults to FALSE. 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 FALSE. Should input nowcasts be checked for consistency with the scoringutils package.
`round_to`	Integer defaults to 3. Number of digits to round scoring output to.
`...`	Arguments passed on to `scoringutils::score` `forecast` A forecast object (a validated data.table with predicted and observed values). `metrics` A named list of scoring functions. Names will be used as column names in the output. See `get_metrics()` for more information on the default metrics used. See the *Customising metrics* section below for information on how to pass custom arguments to scoring functions.

Value

A `data.table` as returned by `scoringutils::score()`.

See Also

Other modelvalidation: `as_forecast_sample.epinowcast()`

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_pathfinder()`, `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")
)
```

`enw_stan_to_r` *Expose* `epinowcast` *stan functions in R*

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 `epinowcast()` package.
`global`	A logical value indicating whether to expose the functions globally. Defaults to `TRUE`. Passed to the `expose_functions()` method of cmdstanr::CmdStanModel.
`verbose`	Logical, defaults to `TRUE`. Should verbose messages be shown.
`...`	Additional arguments passed to `enw_model`.

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_pathfinder(), 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 build_ord_obs(), enw_add_latest_obs_to_nowcast(), enw_nowcast_samples(), enw_nowcast_summary(), enw_posterior(), enw_pp_summary(), enw_quantiles_to_long(
subset_obs()

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 *Unset Stan cache location*

Description

Optionally removes the `enw_cache_location` environment variable from the user .Renviron 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 `.Renviron` file using `readRenviron("~/.Renviron")`.

Usage

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

Arguments

type	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 `.Renviron` 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_pathfinder(), 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 — *Nowcast using partially observed data*

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 `enw_preprocess_data()`.
`reference`	The reference date indexed reporting process model specification as defined using `enw_reference()`.
`report`	The report date indexed reporting process model specification as defined using `enw_report()`.
`expectation`	The expectation model specification as defined using `enw_expectation()`. 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 `rw(week, by = .group)`)).
`missing`	The missing reference date model specification as defined using `enw_missing()`. By default this is set to not be used.
`obs`	The observation model as defined by `enw_obs()`. Observations are also processed within this function for use in modelling.

`fit`	Model fit options as defined using `enw_fit_opts()`. This includes the sampler function to use (with the package default being `enw_sample()`), whether or now a nowcast should be used, etc. See `enw_fit_opts()` for further details.
`model`	The model to use within `fit`. By default this uses `enw_model()`.
`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. 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` *Check an object is a Date*

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` *Convert latest observed data to a matrix*

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` *Parse a formula into components*

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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`, `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` *Plot method for epinowcast*

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 `epinowcast()`.
`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 `base::match.arg()`. Currently supported options: • "nowcast" which plots the nowcast for each dataset along with latest available observed data using `enw_plot_nowcast_quantiles()`, • "posterior_prediction" which plots observations reported at the time against simulated observations from the model using `enw_plot_pp_quantiles()`.
`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 `lme4::findbars()` when a random effect is defined using the lme4 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()`, `rw_terms()`, `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` *Remove profiling statements from a character vector representing stan code*

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_pathfinder()`, `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` *Remove random walk terms from a formula object*

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()`, `rw_terms()`, `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()`, `re()`, `remove_rw_terms()`, `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()`, `re()`, `remove_rw_terms()`, `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

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

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 lme4 syntax (see `re()`), and random walks defined using the `rw()` 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()`, `re()`, `remove_rw_terms()`, `rw()`, `rw_terms()`

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 `epinowcast()` 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()`

---

subset_obs — *Subset observations data table for either modelled dates or not-modelled earlier dates.*

Description

Subset observations data table for either modelled dates or not-modelled earlier dates.

Usage

```
subset_obs(ord_obs, max_delay, internal_timestep, reference_subset)
```

Arguments

`ord_obs`	The observations `data.table` to be subset, as pulled from the result of calling `epinowcast()` and coerced to a data table.
`max_delay`	Whole number representing the maximum delay in units of the timestep.
`internal_timestep`	A numeric value representing the number of days in the timestep, e.g. 7 when the timesteps are weeks.
`reference_subset`	String giving a relational operator to subset ord_obs by reference date; e.g. > to keep the modelled reference dates from after the max_delay.

Value

A `data.frame` subset for the desired observations

See Also

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

`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 `epinowcast()`.
`type`	Character string indicating the summary to return; enforced by `base::match.arg()`. Supported options are: • "nowcast" which summarises nowcast posterior with `enw_nowcast_summary()`, • "nowcast_samples" which samples latest with `enw_nowcast_samples()`, • "fit" which returns the summarised `cmdstanr` fit with `enw_posterior()`, • "posterior_prediction" which returns summarised posterior predictions for the observations after fitting using `enw_pp_summary()`.
`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

`stan_file`	The path to a .stan file containing a Stan program.
`include_paths`	Paths to directories where Stan should look for files specified in #include directives in the Stan program.
`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 `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_pathfinder()`, `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, 9
check_design_matrix_sparsity, 10
check_group, 11
check_group_date_unique, 11
check_max_delay, 12
check_module, 13
check_modules_compatible, 14
check_numeric_timestep, 14
check_observation_indicator, 15
check_quantiles, 16
check_timestep, 16
check_timestep_by_date, 17
check_timestep_by_group, 18
∗ **dataconverters**
enw_add_cumulative, 24
enw_add_incidence, 27
enw_aggregate_cumulative, 32
enw_incidence_to_linelist, 52
enw_linelist_to_incidence, 54
∗ **datasets**
germany_covid19_hosp, 97
∗ **data**
enw_example, 41
germany_covid19_hosp, 97
∗ **epinowcast**
epinowcast, 93
plot.epinowcast, 100
summary.epinowcast, 109
∗ **formulatools**
as_string_formula, 8
construct_re, 21
construct_rw, 22
enw_formula, 48
enw_manual_formula, 55
parse_formula, 99
re, 101
remove_rw_terms, 103
rw, 104
rw_terms, 105
split_formula_to_terms, 107
∗ **modeldesign**
enw_add_cumulative_membership, 25
enw_add_pooling_effect, 31
enw_design, 39
enw_effects_metadata, 40
enw_one_hot_encode_feature, 66
∗ **modelmodulehelpers**
add_max_observed_delay, 4
add_pmfs, 5
convolution_matrix, 23
enw_reference_by_report, 79
enw_reps_with_complete_refs, 84
extract_obs_metadata, 95
extract_sparse_matrix, 96
latest_obs_as_matrix, 99
simulate_double_censored_pmf, 106
∗ **modelmodules**
enw_expectation, 42
enw_fit_opts, 46
enw_missing, 59
enw_obs, 65
enw_reference, 78
enw_report, 81
∗ **modeltools**
enw_formula_as_data_list, 50
enw_get_cache, 51
enw_model, 61
enw_pathfinder, 66
enw_priors_as_data_list, 76
enw_replace_priors, 80
enw_sample, 85
enw_set_cache, 88
enw_stan_to_r, 90
enw_unset_cache, 92
remove_profiling, 102
write_stan_files_no_profile, 110
∗ **modelvalidation**

as_forecast_sample.epinowcast, 7
enw_score_nowcast, 86
∗ **plot**
enw_plot_nowcast_quantiles, 68
enw_plot_obs, 69
enw_plot_pp_quantiles, 70
enw_plot_quantiles, 70
enw_plot_theme, 71
plot.epinowcast, 100
∗ **postprocess**
build_ord_obs, 8
enw_add_latest_obs_to_nowcast, 28
enw_nowcast_samples, 62
enw_nowcast_summary, 63
enw_posterior, 72
enw_pp_summary, 73
enw_quantiles_to_long, 77
enw_summarise_samples, 91
subset_obs, 108
∗ **preprocess**
enw_add_delay, 26
enw_add_max_reported, 29
enw_add_metaobs_features, 30
enw_assign_group, 34
enw_complete_dates, 35
enw_construct_data, 37
enw_extend_date, 43
enw_filter_reference_dates, 44
enw_filter_report_dates, 45
enw_flag_observed_observations, 48
enw_impute_na_observations, 51
enw_latest_data, 53
enw_metadata, 57
enw_metadata_delay, 58
enw_missing_reference, 60
enw_preprocess_data, 74
enw_reporting_triangle, 82
enw_reporting_triangle_to_long, 83
∗ **simulate**
enw_simulate_missing_reference, 89
∗ **utils**
aggregate_rolling_sum, 6
coerce_date, 19
coerce_dt, 20
date_to_numeric_modulus, 24
get_internal_timestep, 98
is.Date, 98
stan_fns_as_string, 107

add_max_observed_delay, 4, *5*, *23*, *80*, *84*, *96*, *97*, *99*, *106*
add_max_observed_delay(), *5*, *95*
add_pmfs, *5*, 5, *23*, *80*, *84*, *96*, *97*, *99*, *106*
aggregate_rolling_sum, 6, *19*, *21*, *24*, *98*, *99*, *108*
as.integer(), *75*
as_forecast_sample.epinowcast, 7, *87*
as_forecast_sample.epinowcast(), *86*
as_string_formula, 8, *21*, *22*, *49*, *56*, *100*, *102–105*, *107*

base::match.arg(), *47*, *65*, *101*, *104*, *109*
build_ord_obs, 8, *28*, *63*, *64*, *72*, *73*, *77*, *92*, *109*

check_calendar_timestep, 9, *10–19*
check_design_matrix_sparsity, *10*, 10, *11–19*
check_group, *10*, 11, *12–19*
check_group_date_unique, *10*, *11*, 11, *13–19*
check_max_delay, *10–12*, 12, *14–19*
check_max_delay(), *12*, *35*, *58*, *75*, *80*, *84*
check_module, *10–13*, 13, *14–19*
check_modules_compatible, *10–14*, 14, *15–19*
check_numeric_timestep, *10–14*, 14, *16–19*
check_observation_indicator, *10–15*, 15, *16–19*
check_quantiles, *10–16*, 16, *17–19*
check_timestep, *10–16*, 16, *18*, *19*
check_timestep_by_date, *10–17*, 17, *19*
check_timestep_by_group, *10–18*, 18
cmdstanr::cmdstan_model(), *61*, *62*
cmdstanr::CmdStanModel, *90*
cmdstanr::sample(), *47*
coerce_date, *6*, 19, *21*, *24*, *98*, *99*, *108*
coerce_date(), *30*
coerce_dt, *6*, *19*, 20, *24*, *98*, *99*, *108*
coerce_dt(), *16*
construct_re, *8*, 21, *22*, *49*, *56*, *100*, *102–105*, *107*
construct_rw, *8*, *21*, 22, *49*, *56*, *100*, *102–105*, *107*
construct_rw(), *104*
convolution_matrix, *5*, 23, *80*, *84*, *96*, *97*, *99*, *106*

data.table, *20*
data.table::as.data.table(), *17, 18*, *20*, *30*
data.table::as.IDate(), *19, 20*
Date, *57*
date_to_numeric_modulus, *6*, *19*, *21*, 24, *98, 99*, *108*
Dates, *30*

enw_add_cumulative, 24, *28*, *34*, *53*, *55*
enw_add_cumulative_membership, 25, *32*, *39, 40*, *66*
enw_add_cumulative_membership(), *22*
enw_add_delay, 26, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_add_incidence, *25*, 27, *34*, *53*, *55*
enw_add_incidence(), *60*, *82*
enw_add_latest_obs_to_nowcast, *9*, 28, *63, 64*, *72, 73*, *77*, *92*, *109*
enw_add_max_reported, *27*, 29, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_add_metaobs_features, *27*, *29*, 30, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_add_metaobs_features(), *43*, *75*
enw_add_pooling_effect, *26*, 31, *39, 40*, *66*
enw_aggregate_cumulative, *25*, *28*, 32, *53*, *55*
enw_aggregate_cumulative(), *38*, *75*
enw_assign_group, *27*, *29*, *31*, 34, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_complete_dates, *27*, *29*, *31*, *34*, 35, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_complete_dates(), *33*, *65*, *74*
enw_construct_data, *27*, *29*, *31*, *34*, *36*, 37, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_cumulative_to_incidence, *25*, *28*, *34*, *53*, *55*
enw_design, *26*, *32*, 39, *40*, *66*
enw_effects_metadata, *26*, *32*, *39*, 40, *66*
enw_effects_metadata(), *31, 32*
enw_example, 41, *97*
enw_expectation, 42, *47*, *59*, *65*, *78*, *82*
enw_expectation(), *14*, *67*, *85*, *93*, *96, 97*
enw_extend_date, *27*, *29*, *31*, *34*, *36*, *38*, 43, *45, 46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_filter_delay, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_filter_reference_dates, *27*, *29*, *31*, *34*, *36*, *38*, *44*, 44, *46*, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_filter_reference_dates(), *45*
enw_filter_report_dates, *27*, *29*, *31*, *34*, *36*, *38*, *44, 45*, 45, *48*, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_filter_report_dates(), *44*
enw_fit_opts, *43*, 46, *59*, *65*, *78*, *82*
enw_fit_opts(), *94*
enw_flag_observed_observations, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, 48, *52*, *54*, *57*, *59, 60*, *76*, *83*
enw_flag_observed_observations(), *36*, *65*
enw_formula, *8*, *21, 22*, 48, *56*, *100*, *102–105*, *107*
enw_formula(), *42*, *50*, *55*, *59*, *78*, *81*, *104*
enw_formula_as_data_list, 50, *51*, *62*, *67*, *77*, *81*, *85*, *88*, *91, 92*, *102*, *111*
enw_get_cache, *51*, 51, *62*, *67*, *77*, *81*, *85*, *88*, *91, 92*, *102*, *111*
enw_impute_na_observations, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, 51, *54*, *57*, *59, 60*, *76*, *83*
enw_incidence_to_cumulative, *25*, *28*, *34*, *53*, *55*
enw_incidence_to_linelist, *25*, *28*, *34*, 52, *55*
enw_latest_data, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, 53, *57*, *59, 60*, *76*, *83*
enw_latest_data(), *28*, *41*, *63, 64*, *86*
enw_linelist_to_incidence, *25*, *28*, *34*, *53*, 54
enw_manual_formula, *8*, *21, 22*, *49*, 55, *100*, *102–105*, *107*
enw_metadata, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, 57, *59, 60*, *76*, *83*
enw_metadata_delay, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, 58, *60*, *76*, *83*
enw_metadata_delay(), *37, 38*, *75*
enw_missing, *43*, *47*, 59, *65*, *78*, *82*
enw_missing(), *47*, *93*
enw_missing_reference, *27*, *29*, *31*, *34*, *36*, *38*, *44–46*, *48*, *52*, *54*, *57*, *59*, 60, *76*, *83*
enw_model, *51*, 61, *67*, *77*, *81*, *85*, *88*, *91, 92*, *102*, *111*

enw_model(), 67, 85, 94
enw_nowcast_samples, 9, 28, 62, 64, 72, 73, 77, 92, 109
enw_nowcast_samples(), 109
enw_nowcast_summary, 9, 28, 63, 63, 72, 73, 77, 92, 109
enw_nowcast_summary(), 28, 109
enw_obs, 43, 47, 59, 65, 78, 82
enw_obs(), 36, 67, 85, 93
enw_one_hot_encode_feature, 26, 32, 39, 40, 66
enw_pathfinder, 51, 62, 66, 77, 81, 85, 88, 91, 92, 102, 111
enw_pathfinder(), 85
enw_plot_nowcast_quantiles, 68, 69–72, 101
enw_plot_nowcast_quantiles(), 71, 101
enw_plot_obs, 68, 69, 70–72, 101
enw_plot_pp_quantiles, 68, 69, 70, 71, 72, 101
enw_plot_pp_quantiles(), 68, 70, 71, 101
enw_plot_quantiles, 68–70, 70, 72, 101
enw_plot_theme, 68–71, 71, 101
enw_posterior, 9, 28, 63, 64, 72, 73, 77, 92, 109
enw_posterior(), 63, 77, 109
enw_pp_summary, 9, 28, 63, 64, 72, 73, 77, 92, 109
enw_pp_summary(), 109
enw_preprocess_data, 27, 29, 31, 34, 36, 38, 44–46, 48, 52, 54, 57, 59, 60, 74, 83
enw_preprocess_data(), 5, 12, 28, 30, 37, 41–43, 54, 58, 59, 63–65, 78, 79, 81, 82, 84, 93–95, 99
enw_priors_as_data_list, 51, 62, 67, 76, 81, 85, 88, 91, 92, 102, 111
enw_quantiles_to_long, 9, 28, 63, 64, 72, 73, 77, 92, 109
enw_reference, 43, 47, 59, 65, 78, 82
enw_reference(), 76, 80, 81, 93, 94
enw_reference_by_report, 5, 23, 79, 84, 96, 97, 99, 106
enw_replace_priors, 51, 62, 67, 77, 80, 85, 88, 91, 92, 102, 111
enw_report, 43, 47, 59, 65, 78, 81
enw_report(), 93
enw_reporting_triangle, 27, 29, 31, 34, 36, 38, 44–46, 48, 52, 54, 57, 59, 60, 76, 82, 83
enw_reporting_triangle(), 83
enw_reporting_triangle_to_long, 27, 29, 31, 34, 36, 38, 44–46, 48, 52, 54, 57, 59, 60, 76, 83, 83
enw_reps_with_complete_refs, 5, 23, 80, 84, 96, 97, 99, 106
enw_reps_with_complete_refs(), 79
enw_sample, 51, 62, 67, 77, 81, 85, 88, 91, 92, 102, 111
enw_sample(), 47, 67, 94
enw_score_nowcast, 7, 86
enw_set_cache, 51, 62, 67, 77, 81, 85, 88, 91, 92, 102, 111
enw_simulate_missing_reference, 89
enw_stan_to_r, 51, 62, 67, 77, 81, 85, 88, 90, 92, 102, 111
enw_summarise_samples, 9, 28, 63, 64, 72, 73, 77, 91, 109
enw_unset_cache, 51, 62, 67, 77, 81, 85, 88, 91, 92, 102, 111
epinowcast, 19, 90, 93, 101, 109
epinowcast(), 7, 27, 41, 47, 54, 62, 63, 73, 75, 80, 86, 90, 101, 108, 109
expose_functions(), 90
extract_obs_metadata, 5, 23, 80, 84, 95, 97, 99, 106
extract_sparse_matrix, 5, 23, 80, 84, 96, 96, 99, 106

formula, 39

germany_covid19_hosp, 41, 97
get_internal_timestep, 6, 19, 21, 24, 98, 99, 108
get_metrics(), 87
ggplot2::aes(), 69, 71

IDate, 19
is.Date, 6, 19, 21, 24, 98, 98, 108

latest_obs_as_matrix, 5, 23, 80, 84, 96, 97, 99, 106
lme4, 8, 21, 48, 49, 99, 100, 102, 103, 105, 107
lme4::findbars(), 102

parse_formula, 8, 21, 22, 49, 56, 99, 102–105, 107
parse_formula(), 49

plot.epinowcast, *68–72*, *94*, 100, *109*
posterior::draws_df(), *62*
posterior::summarise_draws(), *72*

re, *8*, *21, 22*, *49*, *56*, *100*, 101, *103–105*, *107*
re(), *8*, *21*, *49*, *99*, *103*, *105*, *107*
remove_profiling, *51*, *62*, *67*, *77*, *81*, *85*, *88*, *91*, *92*, 102, *111*
remove_rw_terms, *8*, *21, 22*, *49*, *56*, *100*, *102*, 103, *104, 105*, *107*
rstan::extract_sparse_parts(), *96*
rw, *8*, *21, 22*, *49*, *56*, *100*, *102, 103*, 104, *105*, *107*
rw(), *8*, *22*, *25*, *49*, *99, 100*, *103*, *105*, *107*
rw_terms, *8*, *21, 22*, *49*, *56*, *100*, *102–104*, 105, *107*

scoringutils::as_forecast_sample(), *7*
scoringutils::score, *87*
scoringutils::score(), *86, 87*
simulate_double_censored_pmf, *5*, *23*, *80*, *84*, *96, 97*, *99*, 106
split_formula_to_terms, *8*, *21, 22*, *49*, *56*, *100*, *102–105*, 107
stan_fns_as_string, *6*, *19*, *21*, *24*, *98, 99*, 107
startsWith(), *32*
stats::model.matrix, *39*
stats::model.matrix(), *39, 40*
subset_obs, *9*, *28*, *63, 64*, *72, 73*, *77*, *92*, 108
summary.epinowcast, *94*, *101*, 109
summary.epinowcast(), *62–64*, *73*, *86*

tempdir(), *51*
terms, *39*

weekdays(), *30*
write_stan_files_no_profile, *51*, *62*, *67*, *77*, *81*, *85*, *88*, *91, 92*, *102*, 110