

Package: diseasenowcasting (via r-universe)

October 16, 2026

Title Nowcast Time Series of Epidemiological Cases

Version 2.4.0

Description Implements Bayesian nowcasting models for incident disease cases using a censored point-process approach. The reporting delay is modelled directly as a stochastic process, jointly with the epidemic dynamics, via a censored likelihood. Inference uses RTMB (CppAD autodiff + Laplace approximation). Supports arbitrary epidemic models (including HSGP, AR1, SIR), arbitrary delay families (including LogNormal, GenGamma, Dirichlet), stratified data, extreme-delay detection, and backtesting.

License MIT + file LICENSE

URL <https://rodrigozepeda.github.io/diseasenowcasting/>,
<https://github.com/RodrigoZepeda/diseasenowcasting>

BugReports <https://github.com/RodrigoZepeda/diseasenowcasting/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.2.0), tbl.now (>= 0.35.3)

Imports cli, dplyr, ggplot2, lifecycle, Matrix, methods, rlang, RTMB, RTMBdist, S7, scoringutils, stats, utils

Suggests future, generics, withr, testthat (>= 3.0.0), knitr, rmarkdown, scales, covr, patchwork, pkgdown, almanac, RTMBode

Additional_repositories <https://kaskr.r-universe.dev>,
<https://davisvaughan.r-universe.dev>,
<https://rodrigozepeda.r-universe.dev>

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libicu-dev

Repository <https://rodrigozepeda.r-universe.dev>

Date/Publication 2026-09-09 18:55:43 UTC

RemoteUrl <https://github.com/rodrigozepeda/diseasenowcasting>

RemoteRef HEAD

RemoteSha d90a3e36af7340c056f83e461c6147eb304077d5

Contents

auto_nowcast	3
backtest	6
best_model	8
best_model_name	8
best_score	9
comparison_scores	9
covid_colombia	10
cumulative_process	11
custom_delay	12
custom_epidemic	14
default_priors	16
delay_process	17
diseasenowcasting_workflows	18
dn_palette	19
epidemic_process	20
extreme_values	22
fit	22
fit_check	23
fix_param	24
infer_max_time	24
likelihood	25
load_nowcast	26
model	27
nowcast	28
nowcast_diagnostic	30
nowcast_twostage	31
parameters	32
prepare_data	33
priors	35
revision_delay	36
revision_process	38
sample	40
save_nowcast	41
selection_metric	42
selection_timings	42
summarise_nowcast_matrix	43
surprise	43

auto_nowcast

3

surprise.list

44

theme_diseasenowcasting

45

validate_custom_delay

45

validate_custom_epidemic

46

Index

48

auto_nowcast	<i>Automatically select and fit the best nowcasting model</i>
--------------	---

Description

Takes a `tbl_now` and **chooses a model for you**: it builds a grid of candidate models (epidemic process x reporting-delay family) sized to how much data you have, `backtest()`s them over several historical dates, converts the canonical backtest to a scoringutils forecast, keeps the best one, and refits it on the full data. The returned object is an ordinary `nowcast()` result (so `autoplot()`, `predict()`, etc. work), with the ranked scoreboard attached in its comparison slot.

Usage

```
auto_nowcast(  
  data,  
  metric = "wis",  
  relative_score = TRUE,  
  tie_break = c("epidemic_priority", "fastest"),  
  type = c("auto", "two_stage", "one_stage"),  
  sir = NULL,  
  ar = NULL,  
  hsgp = NULL,  
  delays = NULL,  
  likelihood = nb_likelihoood(),  
  models = NULL,  
  n_dates = 6L,  
  n_draws_select = 500L,  
  n_draws = 2000L,  
  K = 25L,  
  K_select = 10L,  
  min_ar = 15L,  
  min_hsgp = 30L,  
  now = NULL,  
  seed = sample.int(.Machine$integer.max, 1),  
  verbose = TRUE,  
  ...  
)
```

Arguments

data	A <code>tbl_now</code> object (<code>tbl.now::tbl_now()</code>).
metric	A single score column produced by <code>scoringutils::score()</code> to minimise. Default "wis".
relative_score	Logical. When TRUE (the default), select on the corresponding relative skill from <code>scoringutils::add_relative_skill()</code> rather than on the raw mean score. This makes comparisons fair when models are not all available for exactly the same targets.
tie_break	How to break effectively equal selection scores. "epidemic_priority" (default) prefers HSGP, then AR(1), then SIR, then a custom epidemic process. "fastest" prefers the candidate with the lowest median elapsed time per successful retrospective fit. The unused rule is applied second, followed by candidate-grid order for determinism.
type	Stage strategy used for <i>both</i> the backtest and the final fit: "auto" (default), "two_stage", or "one_stage" (see <code>nowcast()</code>).
sir, ar, hsgp	Optional epidemic-process components (e.g. <code>sir_epidemic(R0 = ...)</code>) carrying your priors. If supplied, that process is forced into the candidate grid; otherwise the plain constructor is used when the series length calls for it.
delays	A list of delay components to compare. Default: <code>list(lognormal_delay(), generalized_gamma_delay(), dirichlet_delay())</code> .
likelihood	Either a single likelihood used for every candidate (default <code>nb_likelihood()</code>), or a list of likelihoods to compare too, e.g. <code>list(nb_likelihood(), poisson_likelihood())</code> .
models	Optional <code>model()</code> object or list of them (e.g. carrying a <code>custom_delay()</code> / <code>custom_epidemic()</code>) appended to the candidate grid so they compete in the same backtest.
n_dates	Number of historical dates to backtest over (default 6).
n_draws_select	Posterior draws during the selection backtest (default 500 – kept small for speed).
n_draws	Posterior draws for the final fit of the winning model (default 2000).
K	Delay imputations for the final two-stage fit of the winning model (default 25).
K_select	Delay imputations during the selection backtest (default 10 – kept small for speed, like <code>n_draws_select</code>). The selection backtest fits the whole grid over many dates, so its cost scales with <code>K_select</code> ; ranking the candidates is robust to a coarser imputation than the final fit. Lower it (e.g. 5) for a long series where selection dominates the runtime.
min_ar, min_hsgp	Series-length thresholds (in event-times) at which AR(1) and HSGP become candidates (defaults 15 and 30).
now	As-of date for the final fit (default: the <code>tbl_now</code> 's now).
seed	RNG seed.
verbose	Print progress and the chosen model (default TRUE).
...	Passed through to <code>backtest()</code> and <code>nowcast()</code> (e.g. <code>temporal_effects</code>).

Details

Candidate epidemic processes are chosen by series length (`max_time`, the number of event-times): a process becomes a candidate as soon as the series is long enough to support it (SIR needs the least data, the HSGP the most) and is never dropped for being *too* long, so the comparison always spans every process the data can support. With the default thresholds:

- `max_time < min_ar -> compares {SIR};`
- `min_ar <= max_time < min_hsgp -> compares {SIR, AR(1)};`
- `max_time >= min_hsgp -> compares {SIR, AR(1), HSGP}.`

Any process you pass explicitly via `sir / ar / hsgp` is *always* included (regardless of length), which is how you make a prior compete: e.g. `pass sir = sir_epidemic(R0 = lognormal_prior(log(3), 0.2))` and the SIR candidate will use that `R0` prior throughout the comparison.

Robustness. A candidate that fails to converge on a backtest date simply drops out of the comparison there (it never aborts the search), and candidates are scored on the common set of dates where they all produced a forecast so a model cannot "win" on a lucky subset. The winner is then refit on the full data; if that refit fails, `auto_nowcast()` falls through to the next-best candidate (and so on), so it converges whenever any candidate would.

Candidate delays default to LogNormal, Generalized-Gamma and Dirichlet; override with `delays`.

Speed. The grid is backtested with a fast configuration (`n_draws_select` posterior draws over `n_dates` dates spread across the history); only the winning model is refit with the full `n_draws`. Backtesting is the expensive step – set a `future::plan()` (e.g. `future::plan(multisession)`) for parallel speed-up.

Value

A `diseasenowcasting` subclass of `tbl.now::tbl_nowcast` (as from `nowcast()`) for the selected model, with the model-selection scoreboard retained on the `diseasenowcasting` subclass and its native fit: `list(scores, chosen, metric, relative_score, tie_break, timings, max_time)`.

See Also

[diseasenowcasting_workflows](#), [nowcast\(\)](#), [backtest\(\)](#), [scoringutils::score\(\)](#), [scoringutils::add_relative_skill\(\)](#)

Examples

```
library(tbl.now)
data(denguedat)
# A short window keeps this example quick (auto_nowcast fits a whole grid):
dn <- subset(denguedat,
             onset_week >= as.Date("1990-06-01") & onset_week <= as.Date("1990-12-01"))
tn <- tbl_now(dn, event_date = onset_week, report_date = report_week,
             data_type = "linelist", verbose = FALSE)
# Backtesting the grid is the expensive step -- uncomment to run candidates
# in parallel (then restore sequential afterwards):
# future::plan(future::multisession, workers = 4)
# Compare a couple of delays; make the SIR candidate use a custom R0 prior:
nc <- auto_nowcast(tn,
                   sir = sir_epidemic(R0 = lognormal_prior(log(2), 0.3)),
```

```

      delays = list(lognormal_delay(), dirichlet_delay()),
      n_dates = 2, n_draws_select = 150, n_draws = 300,
      temporal_effects = "none")
# future::plan(future::sequential)
best_model_name(nc)  # the winning model's label
comparison_scores(nc) # the ranked scoreboard
best_score(nc)      # just the winner's row
selection_metric(nc) # which metric chose it
winner <- best_model(nc) # the model() object, to reuse elsewhere

```

backtest

Backtest one or more diseasenowcasting models

Description

`backtest()` translates native `model()` specifications into labelled `tbl.now::engine_diseasenowcasting()` specifications and delegates the full retrospective workflow to `tbl.now::nowcast_backtest()`. The returned object is therefore the common `nowcast_backtest` result used by `tbl.now` for tidying, forecast conversion, scoring, weighting, and ensembling.

Usage

```

backtest(
  data,
  models = diseasenowcasting::model(),
  dates = NULL,
  type = c("two_stage", "one_stage", "auto"),
  horizon = NULL,
  n_dates = 4L,
  n_draws = 1000L,
  K = 25L,
  np_spread = 1,
  seed = NULL,
  keep_draws = FALSE,
  on_error = c("warn", "abort"),
  verbose = TRUE,
  truth_axis = NULL,
  truth_type = NULL,
  quantile_levels = tbl.now::nowcast_quantile_levels(),
  ...
)

```

Arguments

`data` A `tbl.now::tbl_now` holding the full data, including observations that arrived after the retrospective nowcast dates.

models	A <code>model()</code> or list of models. Names on the list become the canonical method labels; unnamed models receive labels from their component names. Labels must be unique.
dates	Retrospective nowcast origins, passed as <code>now_dates</code> to <code>tbl.now::nowcast_backtest()</code> .
type	"two_stage", "one_stage", or "auto", passed to every diseasenowcasting engine.
horizon	Number of time units of hindsight used by <code>tbl.now</code> when <code>dates = NULL</code> . <code>NULL</code> uses 4 for ordinary data and the largest model settlement horizon for count-cumulative data (26 for its automatic model).
n_dates	Number of automatic retrospective origins. Ignored when <code>dates</code> is supplied. Default 4.
n_draws	Posterior draws per fit.
K, np_spread	Native two-stage fitting controls passed to every engine.
seed	Optional base seed. <code>tbl.now</code> derives a stable seed for each model/date fit from this value.
keep_draws	Whether the canonical backtest retains posterior draws.
on_error	Either "warn" to record and skip failed cells or "abort".
verbose	Whether to report progress.
truth_axis, truth_type	Canonical scoring truth controls passed to <code>tbl.now::nowcast_backtest()</code> . When both are <code>NULL</code> , they follow the native estimand: reported totals without a revision process, confirmed cases for confirmation/both modes, and still-standing cases for retraction-only mode.
quantile_levels	Quantile probabilities requested from every model.
...	Additional arguments passed to every <code>nowcast()</code> fit through its engine specification.

Value

A `tbl.now::nowcast_backtest` object.

See Also

`diseasenowcasting_workflows` for the native/common ownership boundary; `tbl.now::score_nowcast()`, `tbl.now::nowcast_weights()`, `tbl.now::nowcast_ensemble()`, `fit_check()`

best_model	<i>The winning <code>model()</code> object from a nowcast</i>
------------	---

Description

Returns the fitted nowcast's `model()` specification. For an `auto_nowcast()` result this is the **selected** model, so you can reuse it elsewhere, e.g. `nowcast(other_data, model = best_model(nc))` or pass it to `backtest()`.

Usage

```
best_model(nc)
```

Arguments

nc A nowcast_class (from `nowcast()` or `auto_nowcast()`).

Value

A model_class object.

See Also

`auto_nowcast()`, `best_model_name()`

best_model_name	<i>Name of the model chosen by <code>auto_nowcast()</code></i>
-----------------	--

Description

Name of the model chosen by `auto_nowcast()`

Usage

```
best_model_name(nc)
```

Arguments

nc A nowcast_class returned by `auto_nowcast()`.

Value

The winning model's label, a string of the form "epidemic/likelihood/delay" (e.g. "HSGP/nb/Dirichlet").

See Also

`auto_nowcast()`, `best_model()`, `comparison_scores()`

best_score	<i>The scoreboard row for the model <code>auto_nowcast()</code> chose</i>
------------	---

Description

The single `comparison_scores()` row belonging to the winning model, including its scoringutils metrics, selection score, and retrospective fit timing, rather than the whole table.

Usage

```
best_score(nc)
```

Arguments

`nc` A `nowcast_class` returned by `auto_nowcast()`.

Value

A one-row `data.frame`.

See Also

`auto_nowcast()`, `comparison_scores()`, `best_model_name()`

comparison_scores	<i>The model-selection scoreboard from <code>auto_nowcast()</code></i>
-------------------	--

Description

The ranked table of candidate models that `auto_nowcast()` backtested, one row per model, best-first by the selected raw or relative scoringutils metric. In addition to scoringutils metrics, it includes `selection_score`, median and total retrospective fit seconds, successful-fit count, epidemic priority, and original grid order.

Usage

```
comparison_scores(nc)
```

Arguments

`nc` A `nowcast_class` returned by `auto_nowcast()`.

Value

A `data.frame`, one row per candidate model.

See Also

`auto_nowcast()`, `best_score()`, `selection_timings()`, `scoringutils::score()`

covid_colombia

COVID-19 Notifications – Colombia 2020-2023

Description

Daily case counts of COVID-19 from Colombia's national epidemiological surveillance system (INS), aggregated by notification date, diagnosis date, and sex. Each row is a unique combination of these three variables together with the number of cases *n*.

Usage

`covid_colombia`

Format

A data frame with 35,501 rows and 4 variables:

notification_date Date. The event date – when the case was notified / symptom onset was recorded.

diagnosis_date Date. The report date – when the laboratory diagnosis was registered in the national system.

sex character. Biological sex of the case: "Female" or "Male".

n integer. Number of cases with this (notification_date, diagnosis_date, sex) combination.

Details

In the nowcasting context the **event date** is notification_date (when the case symptom onset was recorded in the system) and the **report date** is diagnosis_date (when the laboratory result was entered). The delay between the two reflects the time from symptom onset to laboratory confirmation and data entry – the quantity the nowcasting model estimates and corrects for.

The dataset covers the full first three years of the Colombian epidemic (2020-03-02 to 2023-03-03) and was used in the NobBS comparison study to benchmark the diseasenowcasting / diseasenowcast2 engine.

Source

Instituto Nacional de Salud (INS), Colombia. Data accessed via the SIVIGILA open-data platform and pre-processed for the diseasenowcasting benchmarking study.

Examples

```
# Build a stratified tbl_now (event = notification, report = diagnosis,
# strata = sex). Pipe through temporal effects for day-of-week covariates.
if (requireNamespace("tbl.now", quietly = TRUE)) {
  tn <- tbl.now::tbl_now(
    covid_colombia,
    event_date = notification_date,
    report_date = diagnosis_date,
    strata      = sex,
    case_count  = n,
    data_type   = "count-incidence",
    verbose     = FALSE
  )
  print(tn)
}
```

cumulative_process	<i>Count-cumulative observation process</i>
--------------------	---

Description

Configures models for revision streams that publish cumulative levels. The target is finite-horizon database retention $C_t(H)$, not biological truth. The retraction mechanism is the collapsed kernel $h_R(l) = \text{retraction_mass} * g_R(l)$; it does not separately identify a truth probability and a conditional revision-delay law.

Usage

```
cumulative_process(
  observation = c("hurdle_ztnb", "hurdle_ztpoisson", "cumulative"),
  retraction_delay = lognormal_delay(),
  settlement = 26L,
  retraction_mass = beta_prior(1.5, 20),
  movement_intercept = normal_prior(-1, 2),
  movement_age = normal_prior(0, 1),
  movement_previous = normal_prior(0, 1),
  magnitude_size = NULL
)
```

Arguments

observation	Observation composite likelihood. "cumulative" uses cumulative Poisson or negative-binomial marginals according to the model's likelihood . "hurdle_ztnb" uses signed hurdle updates with a zero-truncated-negative-binomial magnitude. "hurdle_ztpoisson" uses the corresponding zero-truncated-Poisson magnitude.
-------------	---

retraction_delay Parametric delay family for retraction ages 1:H. Lognormal, gamma, and generalized gamma are supported.

settlement Positive integer settlement horizon H, in model steps.

retraction_mass Prior or fixed value in $[0, 1]$ for the finite-horizon mass of h_R . A Beta prior is used by default.

movement_intercept, movement_age, movement_previous Priors or fixed values for the bounded movement-probability regression. Set movement_previous = 0 to disable previous-movement dependence.

magnitude_size Positive prior or fixed value for the ZTNB magnitude size. It is used only by "hurdle_ztnb".

Value

A cumulative_process_class object for model(cumulative =).

Examples

```
cumulative_process()
cumulative_process(observation = "cumulative", settlement = 52L)
cumulative_process(observation = "hurdle_ztpoisson", settlement = 6L)
```

custom_delay	<i>User-defined delay distribution</i>
--------------	--

Description

[Experimental]

Usage

```
custom_delay(
  cdf,
  log_cdf = function(theta) function(d) log(cdf(theta)(d)),
  log_survival = function(theta) function(d) log(1 - cdf(theta)(d)),
  priors = list(),
  name = "Custom",
  param_names = NULL,
  inits = NULL,
  num_delay_seasons = 1L,
  season_distribution = std_normal_prior()
)
```

Arguments

<code>cdf</code>	A function <code>cdf(theta)</code> returning a function of the delay <code>d</code> that evaluates the CDF $F(d)$ (values in $(0, 1)$). This is the only required piece.
<code>log_cdf</code>	Optional function <code>log_cdf(theta)</code> returning a function of <code>d</code> for $\log F(d)$. Defaults to <code>function(theta) function(d) log(cdf(theta)(d))</code> .
<code>log_survival</code>	Optional function <code>log_survival(theta)</code> returning a function of <code>d</code> for $\log(1 - F(d))$. Defaults to <code>function(theta) function(d) log(1 - cdf(theta)(d))</code> . Supplying it explicitly is recommended for heavy-tailed delays, where the default $\log(1 - F)$ loses precision as $F \rightarrow 1$ (e.g. for a Weibull, $\log_{\text{survival}} = -(d / \text{scale})^{\text{shape}}$ is exact and stable).
<code>priors</code>	A list with one element per parameter. Each element is either a <code>prior_class</code> object (free parameter, assigned that prior) or a length-1 numeric (parameter is fixed to that value and not estimated). The number of parameters is inferred from the length of this list (or from <code>param_names / inits</code> if <code>priors</code> is omitted).
<code>name</code>	A display name for the distribution (used in print output).
<code>param_names</code>	Optional character vector naming each parameter (used in diagnostics); its length sets the number of parameters if <code>priors</code> is empty.
<code>inits</code>	Numeric vector of initial values for the parameters (on the unconstrained scale passed to the functions); its length sets the number of parameters if <code>priors</code> and <code>param_names</code> are empty. Defaults to zeros.
<code>num_delay_seasons</code>	Number of periodic delay seasons. Default 1.
<code>season_distribution</code>	Prior for the delay-season effects.

Details

Allows any reporting-delay distribution that can be expressed with RTMB-traceable functions. You supply the cumulative distribution function `cdf`; `log_cdf` and `log_survival` are optional and default to the obvious transforms of `cdf`. Each is given as a function of the parameter vector `theta` that **returns a function of the delay `d`**:

```

cdf(theta)          -> function(d) returning F(d)
log_cdf(theta)      -> function(d) returning log F(d)
log_survival(theta) -> function(d) returning log(1 - F(d))

```

Inside the RTMB tape `theta` is an advector, so every operation must be AD-traceable (see the *RTMB traceability* section).

Value

A `custom_delay_class` object (a `delay_process_class`).

RTMB traceability

All operations inside `cdf` / `log_cdf` / `log_survival` must be differentiable under RTMB's CppAD tape:

- Allowed: `+`, `-`, `*`, `/`, `exp`, `log`, `sqrt`, `pnorm`, `pgamma`, `lgamma`, `abs`, `sum`, scalar multiplication, fixed-length for loops (length must not depend on parameter values).
- Not allowed: `if/else` on parameter values, `pmax/pmin` on AD types (use `(x + abs(x)) / 2` for `pmax(x, 0)`), external solvers, anything non-differentiable.

Use `validate_custom_delay()` to check your functions before fitting.

See Also

`validate_custom_delay()`, `delay_process`

Examples

```
# Weibull delay: theta = c(log_shape, log_scale).
# Only `cdf` is required; we also supply `log_survival` (exact in the tail).
weibull_cdf <- function(theta) {
  shape <- exp(theta[1]); scale <- exp(theta[2])
  function(d) 1 - exp(-(d / scale)^shape)
}
weibull_log_survival <- function(theta) {
  shape <- exp(theta[1]); scale <- exp(theta[2])
  function(d) -(d / scale)^shape
}
custom_delay(
  cdf          = weibull_cdf,
  log_survival = weibull_log_survival,
  priors       = list(normal_prior(0, 1), normal_prior(log(7), 1)),
  name        = "Weibull",
  param_names  = c("log_shape", "log_scale"),
  inits        = c(0, log(7))
)
```

custom_epidemic

User-defined epidemic process

Description

Lets you supply any RTMB-traceable function `intensity_fn(theta)` that returns the log expected-incidence trajectory `log_mean[n_time x n_strata]` as the latent epidemic process for the now-cast. This makes the framework epidemic-process agnostic: random walks, ODE models, regression surfaces, and anything else that can be written in AD-safe arithmetic are all valid.

Usage

```
custom_epidemic(
  intensity_fn,
  priors = list(),
  name = "Custom",
  param_names = character(0),
  inits = numeric(0)
)
```

Arguments

intensity_fn	A function function(theta) that takes a numeric parameter vector and returns a numeric matrix of dimensions [n_time x n_strata] containing log expected incidence (the full log_mean, not just a trend — include any intercept inside the function). Must use only RTMB-traceable operations: +, -, *, /, exp, log, sqrt, abs, sum, for loops of <i>fixed</i> length (not data-dependent). Never use if/else on parameter values, pmax/pmin on AD types, or external solvers. Call validate_custom_epidemic() to check traceability before fitting.
priors	A list with one element per parameter in theta. Each element is either a prior_class object such as normal_prior() (free parameter, estimated) or a single numeric scalar (fixed parameter, held constant during optimisation). The number of parameters is inferred from the length of this list (or from param_names / inits if priors is omitted). For time-varying processes (e.g. a random walk with one innovation per event-time), get the number of event-times with infer_max_time() first.
name	Character label shown in print and diagnostic output.
param_names	Character vector naming the parameters; its length sets the number of parameters if priors is empty. Defaults to "theta1", ...
inits	Numeric vector of starting values; its length sets the number of parameters if priors and param_names are empty. Defaults to zeros.

Details

[Experimental]

Value

A custom_epidemic_class object (subclass of epidemic_process_class).

See Also

[validate_custom_epidemic\(\)](#), [epidemic_process](#)

Examples

```
# Pure random walk on log-incidence (max_time = 20)
# Uses cumsum() — no [<- assignment needed, so fully AD-safe.
max_time <- 20L
```

```

rw_fn <- function(theta) {
  log_mu0 <- theta[1]
  sigma_rw <- exp(theta[2])
  eps <- theta[3:(2L + max_time)]
  lm <- log_mu0 + cumsum(sigma_rw * eps)
  matrix(lm, max_time, 1L)
}
# n_params is inferred (here 2 + max_time) from the priors list:
custom_epi <- custom_epidemic(
  rw_fn,
  priors = c(list(normal_prior(2, 1), normal_prior(-2, 0.5)),
             rep(list(std_normal_prior()), max_time)),
  name = "RandomWalk",
  param_names = c("log_mu0", "log_sigma", paste0("eps_", seq_len(max_time))),
  inits = c(2, -2, rep(0, max_time))
)

```

default_priors	<i>Build the default prior bundle for an RTMB nowcast model</i>
----------------	---

Description

Build the default prior bundle for an RTMB nowcast model

Usage

```
default_priors(mod, data = NULL, ...)
```

Arguments

mod	A <code>model()</code> object.
data	Optional prepared-data list from <code>prepare_data()</code> (used for the data-informed location/scale defaults). May also be a bare list with an <code>m</code> matrix.
...	Per-key overrides (e.g. <code>phi = lognormal_prior(log(20), 0.5)</code> , <code>delay_mu = normal_prior(log(5), 0.3)</code>).

Details

The default prior for each parameter is documented on the component constructor: see the **Default priors** section of [epidemic_process](#) (HSGP / AR(1) / SIR), [delay_process](#) (LogNormal / Gamma / GenGamma / Dirichlet) and [likelihood](#) (the NB overdispersion `phi`). To *see* what a set of priors implies for the epidemic curve before fitting, use `nowcast(prior_only = TRUE)`.

Value

A named list of prior specs.

See Also

[epidemic_process](#), [delay_process](#), [likelihood](#)

delay_process

*Delay distribution for the Bayesian Nowcast***Description**

Specify the reporting-delay distribution. Parameter slots accept a fixed numeric, a `prior_class` (e.g. `normal_prior()`), or `numeric(0)` for the default prior.

Usage

```
lognormal_delay(
  mu = numeric(0),
  sigma = numeric(0),
  num_delay_seasons = 1L,
  season_distribution = std_normal_prior()
)

gamma_delay(
  shape = numeric(0),
  rate = numeric(0),
  num_delay_seasons = 1L,
  season_distribution = std_normal_prior()
)

generalized_gamma_delay(
  mu = numeric(0),
  sigma = numeric(0),
  Q = numeric(0),
  num_delay_seasons = 1L,
  season_distribution = std_normal_prior()
)

dirichlet_delay(alpha = numeric(0), bins = numeric(0))
```

Arguments

<code>mu</code>	Log-mean intercept (<code>delay_mu</code>).
<code>sigma</code>	Log-scale / SD parameter > 0.
<code>num_delay_seasons</code>	Number of periodic delay seasons. Default 1.
<code>season_distribution</code>	Prior for the delay-season effects.
<code>shape, rate</code>	Gamma delay parameters (the shape slot is the log-mean, the rate slot the delay SD; see the original parameterisation).
<code>Q</code>	GenGamma shape (<code>delay_Q</code>); <code>Q = 0</code> recovers lognormal.
<code>alpha</code>	Dirichlet concentration (scalar broadcast to all bins).
<code>bins</code>	Dirichlet: number of explicit delay bins (geometric tail beyond).

Value

A `delay_process_class` object.

Default priors

When a prior argument is left empty, `default_priors()` supplies these defaults. The delay *mean* prior is **data-informed**: a `normal_prior()` on the log scale, centred at the log of the median observed delay.

- **LogNormal**: $\mu \sim \text{data-informed normal_prior}(\log \text{ median delay}); \sigma \sim \text{gamma_prior}(2, 2)$.
- **Gamma**: $\text{shape} \sim \text{data-informed normal_prior}(\dots); \text{rate} \sim \text{gamma_prior}(2, 2 / \text{sd})$ (data-informed SD).
- **Generalised Gamma**: $\mu \sim \text{data-informed normal_prior}(\dots); \sigma \sim \text{gamma_prior}(2, 0.1); Q(\text{shape}) \sim \text{normal_prior}(0, 0.5)$.
- **Dirichlet**: $\alpha \sim \text{a per-bin concentration vector, data-informed from the empirical delay pmf}(0.05 + (\text{bins}+1) * \text{pmf}), \text{ else rep}(1, \text{bins} + 1)$.

Examples

```
lognormal_delay()
lognormal_delay(mu = normal_prior(log(7), 0.5))
gamma_delay()
generalized_gamma_delay(Q = 1)
dirichlet_delay(alpha = 1, bins = 21)
```

diseasenowcasting_workflows

Native and cross-engine workflows

Description

diseasenowcasting owns the statistical model and RTMB fit; `tbl.now::tbl_nowcast` owns the common result grammar. The public result of `nowcast()` and `auto_nowcast()` therefore supports both layers without an explicit conversion.

Use the native layer for modelling

Use `model()`, `nowcast()`, and `auto_nowcast()` when you need this package's epidemic, delay, likelihood, revision, or cumulative-process components. Use `fit_check()` and `nowcast_diagnostic()` for RTMB-specific convergence and fit diagnostics. These functions automatically unwrap the retained native fit from the common result.

`backtest()` is a native model-specification convenience, not a second evaluation system: it translates each `model()` into a `tbl.now::engine_diseasenowcasting()` and returns `tbl.now::nowcast_backtest()` directly.

Use the common layer for results and comparison

Use `tbl.now::run_nowcast()` with `tbl.now::engine_diseasenowcasting()` when diseasenowcasting is one of several engines. Once a fit exists, use the common methods for `tidy()`, `autoplot()`, predictive scoring, forecast conversion, ensembling, and persistence.

In particular, a backtest can be passed directly to `scoringutils::as_forecast_quantile()`, `scoringutils::as_forecast_point()`, or, when draws were retained, `scoringutils::as_forecast_sample()`. Relative WIS and additional scores then come from `scoringutils::score()` and `scoringutils::add_relative_skill()` rather than a package-local scorer.

Examples

```
## Not run:
fit <- nowcast(data, model = model())
autoplot(fit)           # common result plot
fit_check(fit)          # native RTMB diagnostics

bt <- backtest(data, models = list(default = model()))
relative <- bt |>
  scoringutils::as_forecast_quantile() |>
  scoringutils::score() |>
  scoringutils::add_relative_skill(metric = "wis")

## End(Not run)
```

dn_palette

diseasenowcasting colour palette

Description

Returns hex colour codes matching the package's visual identity (derived from the hex-sticker logo: dark charcoal background, sage green, burnt orange accent).

Usage

```
dn_palette(n = NULL)
```

Arguments

`n` Number of colours to return (1-8). If `NULL`, returns all 8.

Value

A named character vector of hex colour codes.

epidemic_process

Epidemic process for the Bayesian Nowcast

Description

Specify the latent epidemic process. Parameter slots accept a fixed numeric, a `prior_class`, or `numeric(0)` for the default prior. The log-incidence mean intercept is inherited from the likelihood (`mu`).

Usage

```
hsGP_epidemic(
  alpha = numeric(0),
  ell = numeric(0),
  gp_kernel = "matern32",
  gp_basis = "dirichlet",
  num_basis = 0,
  tmax_model = 0
)

ar1_epidemic(phi = numeric(0), sigma = numeric(0), error = numeric(0))

sir_epidemic(
  R0 = numeric(0),
  gamma = numeric(0),
  N_eff = numeric(0),
  N_pop = 10000,
  use_beta_rw_trend = TRUE
)
```

Arguments

<code>alpha</code>	HSGP GP amplitude prior (> 0).
<code>ell</code>	HSGP GP length-scale prior (> 0).
<code>gp_kernel</code>	HSGP kernel: "sq_exp", "matern32" (default), "matern52".
<code>gp_basis</code>	HSGP eigenbasis: "dirichlet"/"sine" (default) or "neumann"/"cosine".
<code>num_basis</code>	HSGP basis count; <code>numeric(0)/0</code> = auto from series length.
<code>tmax_model</code>	HSGP time normalisation; 0 = auto (newest point at the right boundary).
<code>phi</code>	AR(1) autocorrelation prior in $(-1, 1)$.
<code>sigma</code>	AR(1) innovation SD prior (> 0).
<code>error</code>	AR(1) standardised innovation prior.
<code>R0</code>	SIR basic reproduction number prior (> 0).
<code>gamma</code>	SIR recovery rate prior in $(0, 1)$.
<code>N_eff</code>	SIR effective susceptible fraction prior in $(0, 1)$.

`N_pop` SIR total population (default 10000).
`use_beta_rw_trend` SIR: beta follows an AR(1) walk if TRUE (default).

Value

An epidemic_process_class object.

Default priors

When a prior argument is left empty, `default_priors()` supplies these defaults (see also `now-cast(prior_only = TRUE)` to visualise them):

HSGP (hsgp_epidemic):

- alpha (GP amplitude): `half_normal_prior(0, 1)`
- ell (GP length-scale): `inv_gamma_prior(3, 1)`

AR(1) (ar1_epidemic):

- phi (autocorrelation): `std_normal_prior()`
- sigma (innovation SD): `exponential_prior(100)`
- innovations: `std_normal_prior()`

SIR (sir_epidemic):

- R_0 : `lognormal_prior(log(2), 0.5)`
- gamma (recovery rate): `lognormal_prior(log(1/5), 0.5)`
- N_{eff} (susceptible fraction): `beta_prior(2, 5)`

The log-incidence intercept comes from the likelihood (`mu`), defaulting to a data-informed `normal_prior()` centred at the log median daily count.

Examples

```
hsgp_epidemic()
hsgp_epidemic(gp_kernel = "sq_exp", num_basis = 20)
ar1_epidemic(phi = 0.9)
sir_epidemic(R0 = 2.5, use_beta_rw_trend = FALSE)
```

extreme_values	<i>Surprising (extreme) values flagged during the last update()</i>
----------------	---

Description

Returns a tidy data.frame with one row per flagged surprise – currently the reporting delays that arrived later than the model expects (surprise = "delay", direction = "long") – together with their tail probability and the level used. Returns NULL when nothing was flagged (or surprise was not computed).

Usage

```
extreme_values(nc)
```

Arguments

nc A nowcast_class object returned by `update()`.

Value

A data.frame of flagged surprises, or NULL.

fit	<i>Fit a nowcast model with the RTMB engine</i>
-----	---

Description

Optimises the negative log-posterior built from model + data. For delay_only data this fits the reporting-delay process alone (no epidemic); the joint epidemic fit is added in later phases.

Usage

```
fit(
  model,
  data,
  priors = NULL,
  init = NULL,
  control = list(iter.max = 500, eval.max = 1000, rel.tol = 1e-09),
  warn = TRUE
)
```

Arguments

model	A <code>model()</code> object.
data	Prepared-data list from <code>prepare_data()</code> .
priors	Optional prior bundle; defaults to <code>default_priors()</code> .
init	Optional named init list.
control	nlminb control list.
warn	If TRUE, warn when the returned joint fit does not pass the optimizer adequacy checks. Internal warm-start and imputation fits set this to FALSE and report only diagnostics for the fits that affect the result.

Value

A list with par (named estimates), obj, opt, data, priors, model, convergence, and (delay-only) delay_mu / delay_sigma.

fit_check	<i>Check RTMB optimizer diagnostics for a fitted nowcast</i>
-----------	--

Description

Predictive accuracy belongs to `tbl.now::score_nowcast()` and `tbl.now::nowcast_backtest()`. `fit_check()` deliberately reports only diagnostics specific to the RTMB optimization performed by `diseasenowcasting`.

Usage

```
fit_check(object, warn = TRUE)
```

Arguments

object	A result from <code>nowcast()</code> or <code>auto_nowcast()</code> . Native <code>diseasenowcasting</code> operations unwrap the common result automatically.
warn	If TRUE, warn when any retained fit fails the common optimizer adequacy predicate: finite objective and derivatives, optimizer code zero, box-constrained KKT residual, positive-definite curvature on the locally free subspace, and a quadratic objective-gap estimate no larger than 0.01 .

Value

A data frame with one row per retained RTMB fit and columns fit, rung, convergence, objective, raw and projected gradients, quadratic objective gap, Hessian status, any Laplace-precision regularization used for prediction, overall status, and diagnostic reasons.

See Also

[diseasenowcasting_workflows](#) for the distinction between native fit diagnostics and predictive scoring; `nowcast_diagnostic()`, `tbl.now::score_nowcast()`, `tbl.now::nowcast_backtest()`

fix_param	<i>Hard-fix a parameter in a prior bundle (treat as data, drop from estimation)</i>
-----------	---

Description

Hard-fix a parameter in a prior bundle (treat as data, drop from estimation)

Usage

```
fix_param(priors, key, value)
```

Arguments

priors	A prior bundle from default_priors() .
key	Parameter key to fix.
value	Fixed numeric value.

Value

The modified prior bundle.

infer_max_time	<i>Infer max_time for custom epidemic processes</i>
----------------	---

Description

Infers max_time from a model and a tbl_now. max_time is the number of event-time points the model spans: the epidemic process runs from $t = 0$ to $t = \text{max_time} - 1$.

Usage

```
infer_max_time(data, model = diseasenowcasting::model())
```

Arguments

data	A tbl_now object.
model	A model() object (default model()). Only its dimensions matter here, so any model works as a placeholder.

Details

This function is useful for building custom epidemic processes whose [intensity function](#) loops over time and therefore needs to know the number of event-times *before* it is written (e.g. an SIR recursion or a random walk with one innovation per time point).

Value

The number of event-time points in the model (an integer).

Examples

```
library(tbl.now)
library(dplyr)
mpox_pooled <- mpoxdat |>
  ungroup() |>
  summarise(n = sum(n), .by = c(dx_date, dx_report_date))
mpox_tn <- tbl_now(mpox_pooled,
  event_date = dx_date,
  report_date = dx_report_date,
  case_count = n,
  data_type = "count-incidence",
  verbose = FALSE)

# The result is 316: the model spans from time t = 0 (min event_num)
# to time t = 315 (max report_num).
infer_max_time(mpox_tn)
```

likelihood

*Likelihood for the Bayesian Nowcast***Description**

Count observation model for the (truncation-corrected) case counts.

Usage

```
poisson_likelihood(mu = numeric(0))

nb_likelihood(mu = numeric(0), phi = lognormal_prior(log(20), 0.5))
```

Arguments

mu	Log-scale mean intercept prior (or fixed numeric).
phi	Negative-binomial overdispersion prior (or fixed numeric); NB only. Defaults to <code>lognormal_prior(log(20), 0.5)</code> . This is the <i>only</i> place to set the overdispersion prior — <code>nowcast()</code> reads it from the model and does not accept its own phi argument.

Value

A `likelihood_class` object.

Examples

```
poisson_likelihood()
nb_likelihood()
# Wider overdispersion (heavier-tailed counts) -- set via the likelihood:
nb_likelihood(phi = lognormal_prior(log(5), 0.5))
```

load_nowcast	<i>Load a nowcast saved with save_nowcast()</i>
--------------	---

Description

Restores the common `tbl_nowcast` result from a bundle written by [save_nowcast\(\)](#). The native fit is restored into `@fit`. The result works with [predict\(\)](#), [autoplot\(\)](#), [coef\(\)](#), [parameters\(\)](#), [mean\(\)/median\(\)/quantile\(\)](#) straight away (sampling from the stored Laplace mode + precision). To re-fit it – on the same or new data – pass the loaded object’s model to [nowcast\(\)](#) (the saved `tbl_now` is in the data slot).

Usage

```
load_nowcast(file, rebuild = FALSE)
```

Arguments

<code>file</code>	Path to a <code>.rds</code> bundle written by save_nowcast() .
<code>rebuild</code>	If TRUE, also re-tape the RTMB objective for each fit (no re-optimization). Needed only for <code>use_random</code> marginal fits or to inspect the live tape; custom delays/epidemics require <code>library(RTMB)</code> . Default FALSE – the stored mode + precision already drive predict() .

Value

A `diseasenowcasting` subclass of `tbl.now::tbl_nowcast`.

See Also

[save_nowcast\(\)](#)

model

Bayesian Nowcast Model

Description

Combines a likelihood, an epidemic process, a reporting-delay distribution, and an optional revision process into a model object. Arguments are positional: likelihood, epidemic process, reporting delay, then revision process. Any argument can be omitted by naming the later components.

Usage

```
model(
  likelihood = nb_likelihood(),
  epidemic = hsgp_epidemic(),
  delay = lognormal_delay(),
  revision = no_revision(),
  covariate_prior = std_normal_prior(),
  strata_pooling = "independent",
  cumulative = no_cumulative()
)
```

Arguments

likelihood	A <code>likelihood_class</code> (<code>poisson_likelihood()</code> / <code>nb_likelihood()</code>). Default: <code>nb_likelihood()</code> .
epidemic	An <code>epidemic_process_class</code> . Default: <code>hsgp_epidemic()</code> .
delay	A <code>delay_process_class</code> . Default: <code>lognormal_delay()</code> .
revision	A <code>revision_process_class</code> (<code>revision_process()</code>) for report-level confirmation/retraction outcomes in linelist or count-incidence data. Default: <code>inert</code> . Count-cumulative revisions use the separate cumulative component.
covariate_prior	A <code>prior_class</code> applied to all covariate coefficients. Default: <code>std_normal_prior()</code> .
strata_pooling	"independent" (default) fits fully separate intercepts per stratum. "hierarchical" pools intercepts via a shared mean and a half-normal prior on the between-stratum SD: $\mu_0^{(s)} = \mu_{\text{global}} + \tau \cdot \delta^{(s)}$, $\delta^{(s)} \sim \mathcal{N}(0, 1)$, $\tau \sim \text{HalfNormal}(0, 1)$. Only relevant when <code>num_strata > 1</code> .
cumulative	Dedicated count-cumulative observation configuration from <code>cumulative_process()</code> . It is inert by default for linelist and count-incidence data; count-cumulative data use the hurdle-ZTNB default unless configured explicitly.

Value

A `model_class` object.

Examples

```
model()
model(poisson_likelihood(), hsgp_epidemic(gp_kernel = "matern52"))
model(nb_likelihood(), ar1_epidemic(), lognormal_delay())
model(nb_likelihood(), ar1_epidemic(), lognormal_delay(),
      revision_process())
model(nb_likelihood(), hsgp_epidemic(), lognormal_delay(),
      strata_pooling = "hierarchical")
```

nowcast

Fit a nowcast model to censored reporting data

Description

The main entry point. Takes a `tbl_now` (from the `tbl.now` package) and a `model()`, fits the latent-epidemic + reporting-delay model as of a given date, and returns the common `tbl.now::tbl_nowcast` result. Predictive draws and quantiles are materialised in that result; the untouched native fit is retained in `@fit` for `predict()`, `mean()/median()/quantile()`, `coef()`, and model-specific diagnostics.

Usage

```
nowcast(
  data,
  model = diseasenowcasting::model(),
  type = c("two_stage", "one_stage", "auto"),
  now = NULL,
  K = 25L,
  n_draws = 2000L,
  delay_window = 120L,
  np_spread = 1,
  floor_mu = 0.08,
  floor_sig_frac = 0.08,
  temporal_effects = "auto",
  prior_only = FALSE,
  quantile_levels = tbl.now::nowcast_quantile_levels(),
  seed = sample.int(.Machine$integer.max, 1),
  ...
)
```

Arguments

`data` A `tbl_now` object (`tbl.now::tbl_now()`).

`model` A `model()` object. Default: `model()` (NB + HSGP + Dirichlet).

type	"two_stage" (default; reporting-delay imputation pooling), "one_stage" (a single joint fit), or "auto" (per reporting-delay family: Dirichlet one-stage, all other delays two-stage – the better choice for each in our experiments).
now	As-of date; only events/reports up to now are used. Default: <code>tbl.now::get_now(data)</code> , falling back to the latest report date.
K	Number of delay imputations for the two-stage path.
n_draws	Default number of posterior draws used by <code>predict()</code> and the latent-incidence summaries.
delay_window	Recent window length for the parametric Stage-1 delay fit.
np_spread	Dirichlet simplex imputation covariance inflation (default 1).
floor_mu, floor_sig_frac	Imputation-spread floors (parametric families).
temporal_effects	Controls automatic seasonal / day-of-week covariates. "auto" (default) adds sensible effects based on the data's time unit (weekly -> 52-period seasonality; daily -> day-of-week + 52-period seasonality; monthly -> 12-period seasonality) only if the <code>tbl.now</code> does not already carry computed temporal effects . Use "none" (or "None") to disable, or pre-attach your own effects to the <code>tbl.now</code> with <code>tbl.now::add_temporal_effects() + tbl.now::compute_temporal_effects()</code> .
prior_only	If TRUE, ignore the likelihood and draw the epidemic parameters from their priors only, returning the prior-predictive latent incidence. Useful for understanding what a prior implies <i>before</i> seeing data (e.g. how the SIR R_0 prior or the AR(1) ϕ prior reshapes the epidemic). The result uses the same common grammar as an ordinary fit, so <code>predict()</code> / <code>autoplot()</code> / <code>median()</code> / <code>quantile()</code> all work; data only supplies the time grid. Default FALSE.
quantile_levels	Probabilities at which to summarise the predictive draws in the returned <code>tbl.now::tbl_nowcast</code> .
seed	Optional RNG seed (imputation draws).
...	Passed to <code>prepare_data()</code> (e.g. <code>gp_boundary_frac</code>).

Value

A `diseasenowcasting` subclass of `tbl.now::tbl_nowcast`. The native fitted model is retained in `@fit`; `diseasenowcasting` operations unwrap it automatically. The `@fit_diagnostics` property (also available at `@metadata$diseasenowcasting$fit_diagnostics`) records the resolved fitting stage, imputation retention, optimizer adequacy, curvature, and any Laplace-precision regularization used for prediction.

Revision processes

A revision process – reports that are later **confirmed** or **retracted** – is a row-level mechanism for linelist and count-incidence data. It is detected from the data, not requested by an argument. `nowcast()` attaches one when the `tbl.now` carries `revision_date` / `revision_type` (see `tbl.now::add_revision_date()`). The mode (`confirmation_only` / `retraction_only` / `both`) is read from `unique(revision_type)` over the full data, so it is stable across as-of dates. Configure `p` and the lag with `model(revision = revision_process(...))`, which always wins over the

detected default; assert the mode with `revision_process(mode = )`. Revision-date censoring is likewise data metadata: set `is_censored_revision` when constructing the `tbl_now`. There is no `nowcast()` column-name argument for it. Count-cumulative revisions instead use `cumulative_process()` and do not estimate a separate revision probability `p`.

One-stage and two-stage revision

With `type = "one_stage"`, the epidemic process, event-to-report delay, report-to-revision delay, and revision probability `p` are estimated in one joint objective. With `type = "two_stage"`, Stage 1 estimates the event-to-report delay and draws `K` imputations from its Laplace approximation. Each Stage-2 fit conditions on one imputed reporting-delay distribution and jointly estimates the epidemic process, revision delay, and `p`. Posterior draws within a Stage-2 fit propagate revision uncertainty; pooling across the fits additionally propagates reporting-delay uncertainty. This is the same stepwise boundary used by the original event-to-report model: `two_stage` separates the reporting process from the downstream model, while the revision block remains downstream of a report.

Overdispersion (phi)

The negative-binomial overdispersion prior is **not** an argument of `nowcast()`. Set it on the likelihood instead, e.g. `model(nb_likelihood(phi = lognormal_prior(log(5), 0.5)), ...)`. The default `nb_likelihood()` already uses `lognormal_prior(log(20), 0.5)`.

See Also

[diseasenowcasting_workflows](#) for when to use native modelling operations versus the shared `tbl_now` result workflow; `backtest()` and `auto_nowcast()` for retrospective comparison and automatic selection.

Examples

```
if (requireNamespace("tbl.now", quietly = TRUE)) {
  # data <- tbl.now::tbl_now(my_linelist, event_date = onset, report_date = reported)
  # nc <- nowcast(data, model(nb_likelihood(), hsgp_epidemic(), lognormal_delay()))
  # predict(nc); median(nc); coef(nc)
}
```

nowcast_diagnostic	<i>Three-panel diagnostic plot for a fitted nowcast</i>
--------------------	---

Description

Produces three stacked panels: (1) reporting-delay histogram with the fitted density, (2) smoothed latent incidence λ with 50/90% CI, and (3) posterior-predictive nowcast with observed counts. Requires the `patchwork` package for a combined output.

Usage

```
nowcast_diagnostic(
  object,
  n_draws = NULL,
  seed = sample.int(.Machine$integer.max, 1),
  previous_times = 30
)
```

Arguments

<code>object</code>	A <code>nowcast_class</code> object.
<code>n_draws</code>	Number of posterior draws (default: <code>min(object@n_draws, 500)</code>).
<code>seed</code>	Optional RNG seed.
<code>previous_times</code>	Number of most recent event-times to display in panels 2 (smoothed incidence) and 3 (nowcast): only the last <code>previous_times</code> dates (including the nowcast target) are shown, as in <code>autoplot()</code> . Panel 1 (the delay distribution) always uses all delays. Default 30; use <code>Inf</code> (or <code>NULL</code>) to show every event-time.

Value

A patchwork combined plot (or a named list of three ggplots).

<code>nowcast_twostage</code>	<i>Two-stage multiple-imputation nowcast</i>
-------------------------------	--

Description

Two-stage multiple-imputation nowcast

Usage

```
nowcast_twostage(
  model,
  m,
  X = NULL,
  d_star = NULL,
  max_time = NULL,
  target = NULL,
  delay_window = 120L,
  K = 25L,
  floor_mu = 0.08,
  floor_sig_frac = 0.08,
  np_spread = 1,
  n_draws_per = 200L,
  phi = lognormal_prior(log(20), 0.5),
  probs = c(0.025, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.975),
  seed = sample.int(.Machine$integer.max, 1)
)
```

Arguments

<code>model</code>	A <code>model()</code> object (LogNormal delay).
<code>m</code>	Observation matrix [event_time, count, delay, strata...].
<code>X</code>	Optional covariate matrix (max_time rows).
<code>d_star</code>	Optional max-observable-delay vector.
<code>max_time</code>	Time-window length; defaults to <code>max(m[, 1])</code> .
<code>target</code>	Event-time to nowcast (default newest).
<code>delay_window</code>	Recent window length for the Stage-1 delay fit.
<code>K</code>	Number of delay imputations.
<code>floor_mu</code>	Floor on the log-mean imputation SD (parametric families).
<code>floor_sig_frac</code>	Floor on the delay-SD imputation SD (fraction of sigma).
<code>np_spread</code>	Dirichlet only: covariance-inflation factor for the simplex imputation (samples delay_logits from the Stage-1 Laplace posterior with covariance scaled by np_spread). Default 1 (the raw, well-informed full-series posterior); values > 1 widen the simplex spread.
<code>n_draws_per</code>	Posterior nowcast draws per imputation.
<code>phi</code>	NB overdispersion prior (default <code>lognormal_prior(log(20), 0.5)</code>).
<code>probs</code>	Quantile probabilities to report.
<code>seed</code>	Optional RNG seed.

Value

A list with quantiles, median, pooled draws, the rung used ("multi", "anchored", or "onestage"), `n_samp` (imputations pooled), and `fit_diagnostics`. The latter records the requested and retained imputation counts plus warm, Stage-1, fallback, exclusion, and retained-fit diagnostics.

parameters	<i>Parameter estimates from a fitted nowcast</i>
------------	--

Description

Returns all estimated parameters as a long data frame with credible intervals derived from the Laplace approximation posterior precision matrix.

Usage

```
parameters(x, conf.level = 0.95, ...)
```

Arguments

<code>x</code>	A <code>nowcast_class</code> object.
<code>conf.level</code>	Credible level for the interval (default 0.95).
<code>...</code>	Unused.

Details

`tidy()` on a nowcast gives you the **nowcast** – the predicted counts, via `tbl.now`'s method for the broom generic. This function gives you the **parameters**. They are different questions, and this package used to answer the second one under the first one's name.

Value

A data.frame with columns `term`, `estimate`, `std.error`, `conf.low`, `conf.high`, `type`.

Examples

```
if (requireNamespace("tbl.now", quietly = TRUE)) {
  # nc <- nowcast(data, model())
  # parameters(nc)
}
```

```
prepare_data
```

```
Prepare data for the RTMB nowcast engine
```

Description

Prepare data for the RTMB nowcast engine

Usage

```
prepare_data(
  model,
  m,
  m_censored = NULL,
  X = NULL,
  d_star = NULL,
  delay_only = FALSE,
  max_time = NULL,
  num_strata = NULL,
  gp_L = 1.5,
  gp_boundary_frac = 0.62,
  ar_sigma_max = 1,
  is_confirmation = FALSE,
  cumulative_levels = NULL,
  cumulative_previous_nonzero = NULL,
  cumulative_settlement = NULL,
  retraction = NULL,
  resolution_mode = 0L,
  ...
)
```

Arguments

<code>model</code>	A <code>model()</code> object.
<code>m</code>	Observation matrix: columns [event_time, count, delay, strata...], delays 1-indexed (single stratum supported in this version).
<code>m_censored</code>	Optional censored-observation matrix (same layout).
<code>X</code>	Optional covariate matrix (max_time rows, P columns).
<code>d_star</code>	Optional max-observable-delay vector; if NULL, computed as <code>rev(seq_len(max_time)) - 1</code> .
<code>delay_only</code>	If TRUE, only the delay process is prepared/fit.
<code>max_time</code>	Time-window length; defaults to <code>max(m[, 1])</code> .
<code>num_strata</code>	Number of stratum cells (the K-way product of the strata levels). If NULL, inferred from column 4 of <code>m</code> . The likelihood is summed over all max_time x num_strata (time, stratum) cells; 1 is unstratified.
<code>gp_L</code>	HSGP boundary factor (> 1). Default 1.5.
<code>gp_boundary_frac</code>	Fraction of the HSGP domain placed left of the data. Default 0.62.
<code>ar_sigma_max</code>	Upper bound on the AR/beta RW innovation SD. Default 1.
<code>is_confirmation</code>	Deprecated legacy switch. TRUE now errors; use the dedicated <code>count_cumulative</code> model component and cumulative arguments.
<code>cumulative_levels</code>	Optional cumulative levels aligned row-for-row with <code>m</code> ; used only by the dedicated count-cumulative composites.
<code>cumulative_previous_nonzero</code>	Optional indicators, aligned with <code>m</code> , that the preceding signed update was non-zero.
<code>cumulative_settlement</code>	Optional positive integer settlement horizon H. It is required for count-cumulative preparation.
<code>retraction</code>	Optional list of linelist-retraction sufficient statistics from <code>.linelist_retraction_stats()</code> . When supplied, the engine carries the cure-model observation block (see <code>31_retraction_likelihood.R</code>). Default NULL.
<code>resolution_mode</code>	0L when the resolution observed is a RETRACTION (the default), 1L when it is a CONFIRMATION. Sets the support of the resolution lag and what a resolved row means for the nowcast target.
<code>...</code>	Reserved.

Value

A named list of engine inputs.

priors

Priors for model parameters

Description

Specify a prior distribution for any estimated parameter. The result is a `prior_class` object that can be passed to any parameter slot in `poisson_likelihood()`, `nb_likelihood()`, the delay process constructors, or the epidemic process constructors.

Usage

```
std_normal_prior()

normal_prior(mean, sd)

cauchy_prior(location, scale)

student_t_prior(df, location, scale)

double_exponential_prior(mean, sd)

flat_prior()

positive_flat_prior()

half_std_normal_prior()

half_normal_prior(location, scale)

half_cauchy_prior(location, scale)

half_student_t_prior(df, scale)

half_double_exponential_prior(mean, sd)

gamma_prior(shape, rate)

weibull_prior(shape, scale)

inv_gamma_prior(shape, scale)

lognormal_prior(meanlog, sdlog)

chi_square_prior(df)

exponential_prior(rate)
```

```
logistic_prior(location, scale)
```

```
beta_prior(alpha, beta)
```

Arguments

mean, location	Location parameter.
sd, scale	Scale parameter (must be > 0).
df	Degrees of freedom (must be > 0).
shape	Shape parameter (must be > 0).
rate	Rate parameter (must be > 0).
meanlog	Mean on the log scale (lognormal).
sdlog	SD on the log scale (lognormal, must be > 0).
alpha, beta	Beta distribution shape parameters (must be > 0).

Value

A prior_class object.

Examples

```
normal_prior(0, 1)
gamma_prior(2, 0.1)
exponential_prior(1)
flat_prior()
```

revision_delay	<i>Revision-lag distributions</i>
----------------	-----------------------------------

Description

The distribution g_C of the **revision lag**: how long after a case is reported its result comes back. Pass one to [revision_process\(\)](#) as revision_delay.

Usage

```
lognormal_revision(mu = numeric(0), sigma = numeric(0))
```

```
gamma_revision(shape = numeric(0), rate = numeric(0))
```

```
generalized_gamma_revision(mu = numeric(0), sigma = numeric(0), Q = numeric(0))
```

```
dirichlet_revision(alpha = numeric(0), bins = numeric(0))
```

Arguments

mu	Log-mean intercept (delay_mu).
sigma	Log-scale / SD parameter > 0.
shape, rate	Gamma delay parameters (the shape slot is the log-mean, the rate slot the delay SD; see the original parameterisation).
Q	GenGamma shape (delay_Q); Q = 0 recovers lognormal.
alpha	Dirichlet concentration (scalar broadcast to all bins).
bins	Dirichlet: number of explicit delay bins (geometric tail beyond).

Details

These are aliases of the corresponding [delay_process](#) constructors – a revision lag is an ordinary non-negative delay, only measured from the *report* rather than from the event – so the parameters, priors and **[Experimental]** behaviour are identical. The one difference is the support, and it depends on the mode: under *retraction_only* the lag lives on {1, 2, ...} (a retraction lands strictly after the report it withdraws, so a case retracted in the same period is dropped by [nowcast\(\)](#) – it was never visible in any data vintage), while under *confirmation_only* and both it lives on {0, 1, ...}, since a test can come back the day it was ordered.

Value

A `delay_process_class` object, for the `revision_delay` slot of [revision_process\(\)](#).

Which one to use

`dirichlet_revision()` is the safest default when the counts are large. The correction applied to a pending report of age j is $\rho(j) = p / (p + (1 - p) * (1 - G_C(j)))$, so at high counts a *shape* error in g_C biases the nowcast by more than its Monte-Carlo noise: on a COVID series of ~8000 cases/day a lognormal g_C fitted to a $1 + \text{Poisson}(2)$ lag left a ~0.9% bias and lost nominal coverage, while the Dirichlet recovered ρ to four decimals. At low counts the parametric families are fine and estimate fewer parameters.

See Also

[revision_process\(\)](#), [delay_process](#), [nowcast\(\)](#)

Examples

```
revision_process(lognormal_revision())
revision_process(dirichlet_revision(bins = 10))

# Held-fixed revision lag, e.g. from an external study
revision_process(gamma_revision(shape = log(3), rate = 2))
```

revision_process	<i>Revision process: reports that are later confirmed or retracted</i>
------------------	--

Description

A report is rarely a case outright. It is provisional, and **resolved exactly once**: a test comes back, and the report is either *confirmed* (a real case) or *retracted* (removed from the register). A revision process models that second step, so the nowcast targets the settled count rather than the raw report count.

Usage

```
revision_process(
  revision_delay = lognormal_delay(),
  p = numeric(0),
  stratified_p = FALSE,
  mode = c("auto", "confirmation_only", "retraction_only", "both")
)
```

Arguments

- | | |
|----------------|--|
| revision_delay | A <code>delay_process_class</code> describing the revision lag <code>g_C</code> (report to result). Any delay family works; the <code>*_revision()</code> constructors (lognormal_revision() and friends) are aliases that read more naturally in this slot. Default a short <code>lognormal</code> . |
| p | Probability that a report resolves positive – that it is a real case and is never retracted. Either a <code>prior_class</code> (estimated under that prior) or a single numeric in $(0, 1]$ (held fixed). Left unset (the default) the behaviour depends on the data, because the two cases identify <code>p</code> through the report-level cure block: <ul style="list-style-type: none"> • linelist / count-incidence – a weak data-informed Beta. The cure block pins <code>p</code> directly: a report standing unresolved for a long time is evidence about the cure fraction, so the likelihood is informative and the prior only has to keep <code>p</code> on the interval. |
| stratified_p | If TRUE, estimate a separate <code>p</code> per stratum instead of one shared value. Only meaningful for per-report revision data with more than one stratum; the revision lag <code>g_C</code> stays shared either way (it is usually a property of the verification workflow, whereas <code>p</code> reflects how often a given group is misclassified). Each stratum's <code>p</code> gets the same prior. Default FALSE – with sparse strata the shared <code>p</code> is safer. |
| mode | Which outcomes the data record. "auto" (the default) infers it from the <code>revision_type</code> column; the others assert it. See Modes . |

Details

Attach it with `model(revision = revision_process(...))`. [nowcast\(\)](#) switches it on automatically when the data carry it – see **Detection** below.

Value

A `revision_process_class` object, for `model(revision = )`.

Modes

What differs between surveillance systems is only which resolutions get recorded, and all three possibilities share one likelihood:

	<code>retraction_only</code>	<code>confirmation_only</code>	<code>both</code>
outcomes recorded	the negatives	the positives	both signs
a missing outcome means	not retracted yet	not confirmed yet	not resolved yet
target	reports never retracted	reports eventually confirmed	reports resolving positive
lag support	{1, 2, ...}	{0, 1, ...}	{0, 1, ...}

The lag support differs only because a *retraction* in the same period as its report describes a case never visible in any data vintage, whereas a test coming back the day it was ordered is ordinary.

In `mode = "both"`, `revision_delay` is one shared law for positive and negative resolutions. This first prototype deliberately does not fit separate competing-risk lag laws. In a one-outcome mode, the same argument denotes the lag for the outcome that is recorded: confirmation or retraction respectively.

"auto" reads `unique(revision_type)` over the **full** data, not the as-of view, so the mode is a stable property of the data source and does not flip between backtest dates.

Detection

`nowcast()` attaches a revision process when the `tbl_now` carries `revision_date / revision_type` (see `tbl.now::add_revision_date()`).

Count-cumulative data instead use `cumulative_process()`, whose primitive retraction object is the finite-age kernel `h_R`; it does not estimate `p`.

Count-cumulative data

Do not use this component for an aggregate cumulative stream. Configure its down-revisions with `cumulative_process()`. Without individual report outcomes, `p` and a conditional revision-delay law are not separately identified.

Default priors

`revision_delay` inherits the default priors of its delay family (see `delay_process`). `p` is described under its argument above. At `p = 1` the revision layer is inert and the model is the ordinary count model.

See Also

`model()`, `revision_delay`, `delay_process`, `nowcast()`

Examples

```
# Results come back on one shared timescale, whatever the answer:
revision_process(lognormal_revision())

# Attach to a model:
model(nb_likelihood(), hsgp_epidemic(), lognormal_delay(),
      revision = revision_process(
        revision_delay = dirichlet_revision(bins = 10),
        p               = beta_prior(20, 3),
        stratified_p    = TRUE))
```

sample	<i>Draw random samples from a prior (or fall back to <code>base::sample()</code>)</i>
--------	---

Description

A generic that draws from a `prior_class` with the appropriate random number generator. For **any other object** (numeric, character, factor, Date, list, ...) it dispatches to `base::sample()`, so `sample()` keeps its usual base behaviour outside the package.

Usage

```
sample(object, size, ...)

## S7 method for class <any>
sample(object, size, ...)

## S7 method for class <diseasenowcasting::prior_class>
sample(object, size, ...)
```

Arguments

object	A <code>prior_class</code> , or any object accepted by <code>base::sample()</code> .
size	Number of draws.
...	Passed through to the underlying sampler.

Value

For a prior, a numeric vector of length `size`; otherwise whatever `base::sample()` returns.

Examples

```
sample(normal_prior(log(7), 0.5), 10)
sample(gamma_prior(2, 0.1), 10)
sample(as.Date("2020-01-01") + 0:9, 3) # falls back to base::sample()
```

save_nowcast	<i>Save a fitted nowcast to disk</i>
--------------	--------------------------------------

Description

Serializes a `nowcast()` (or `auto_nowcast()`) result to a single `.rds` bundle that can be restored later with `load_nowcast()`. The RTMB autodiff tape is **not** saved (its external pointers cannot be serialized); instead the fitted parameters and the Laplace mode + precision are stored, which is all `predict()` needs. The input `tbl_now` and the `model()` specification are saved too, so the loaded object can also be re-fit.

Usage

```
save_nowcast(object, file)
```

Arguments

object	A result from <code>nowcast()</code> or <code>auto_nowcast()</code> . The native fit stored in <code>object@fit</code> is serialized automatically.
file	Path to write (a single <code>.rds</code> file).

Value

file, invisibly.

See Also

[load_nowcast\(\)](#)

Examples

```
if (requireNamespace("tbl.now", quietly = TRUE)) {
  library(tbl.now)
  data(denguedat)
  dn <- subset(denguedat, onset_week <= as.Date("1990-12-01") &
               report_week <= as.Date("1990-12-01"))
  tn <- tbl_now(dn, event_date = onset_week, report_date = report_week,
               data_type = "linelist", verbose = FALSE)
  nc <- nowcast(tn, type = "one_stage", n_draws = 200)
  f <- tempfile(fileext = ".rds")
  save_nowcast(nc, f)
  nc2 <- load_nowcast(f)
  predict(nc2, summary = TRUE)      # works without the original RTMB tape
}
```

selection_metric	<i>The metric <code>auto_nowcast()</code> used to pick the winner</i>
------------------	---

Description

The metric `auto_nowcast()` used to pick the winner

Usage

```
selection_metric(nc)
```

Arguments

nc A `nowcast_class` returned by `auto_nowcast()`.

Value

The scoringutils score-column name supplied to `auto_nowcast()`. Consult the result's `relative_score` comparison field to determine whether its relative skill was used.

See Also

[auto_nowcast\(\)](#), [comparison_scores\(\)](#)

selection_timings	<i>Fitting times recorded by <code>auto_nowcast()</code></i>
-------------------	--

Description

Fitting times recorded by `auto_nowcast()`

Usage

```
selection_timings(nc)
```

Arguments

nc A result returned by `auto_nowcast()`.

Value

A list with `backtest` (one row per attempted retrospective fit), `refit` (the full-data refit attempts), and `total_seconds` for the complete automatic-selection call.

See Also

[auto_nowcast\(\)](#), [comparison_scores\(\)](#)

summarise_nowcast_matrix

Quantile-table summary of a pooled nowcast draws matrix

Description

Matches diseasesnowcast2's `.summarise_nowcast_matrix()` exactly: one row per event (`.event_num` 0-indexed, ascending) with columns mean, median, sd, mad, q2.5, q5, q10, q25, q50, q75, q90, q95, q97.5.

Usage

```
summarise_nowcast_matrix(draws_matrix)
```

Arguments

`draws_matrix` Pooled draws matrix [`n_draws` x `max_time`].

Value

A data.frame in the scoring-pipeline format.

surprise

Compute surprise scores for new observations

Description

Given a fitted nowcast and new data that arrived *after* the fit, compute how surprising the new data is under the fitted posterior.

Two types of surprise are supported (select via the `type` argument):

- "count": How surprising is a new case count at a given event time
- "delay": How surprising is a report with a very long delay
- "both": Compute both (requires appropriate `new_data` format).

Usage

```
surprise(
  object,
  new_data,
  type = c("both", "count", "delay"),
  level = 0.99,
  n_draws = 500L,
  seed = sample.int(.Machine$integer.max, 1)
)
```

Arguments

object	A nowcast_class object (or fit() result).
new_data	For type = "count": a data.frame with columns event_index (0-indexed) and count (observed new total at that time). For type = "delay": a data.frame with columns delay (numeric delay values) and optionally weight (counts with that delay, default 1). For type = "both": supply both sets of columns.
type	Which surprise type(s) to compute. One of "count", "delay", "both" (default "both").
level	Credible level used to flag surprises (default 0.99). A count is flagged when it falls in the outer $(1 - \text{level})/2$ tail of the posterior predictive (too high or too low); a delay is flagged when $P(D \geq d)$ or $P(D \leq d)$ is below $1 - \text{level}$ (surprisingly long or short).
n_draws	Number of posterior draws (default 500).
seed	Optional RNG seed.

Value

A list with:

- \$count_surprise (data.frame): one row per event time with event_index, observed, posterior_mean, lpd, ppp_right, ppp_left, relative_surprise.
- \$delay_surprise (data.frame): one row per delay value with delay, tail_prob (= $1 - G_D(\text{delay})$), lpd, relative_surprise.

surprise.list	<i>Surprise score on a raw fit() result</i>
---------------	---

Description

Surprise score on a raw fit() result

Usage

```
## S3 method for class 'list'
surprise(
  object,
  new_data,
  type = c("both", "count", "delay"),
  level = 0.99,
  n_draws = 500L,
  seed = sample.int(.Machine$integer.max, 1)
)
```

Arguments

object	A list returned by <code>fit()</code> .
new_data	See <code>surprise()</code> .
type	See <code>surprise()</code> .
level	See <code>surprise()</code> .
n_draws	See <code>surprise()</code> .
seed	See <code>surprise()</code> .

 theme_diseasenowcasting

ggplot2 theme matching the diseasenowcasting visual identity

Description

ggplot2 theme matching the diseasenowcasting visual identity

Usage

```
theme_diseasenowcasting(base_size = 11)
```

Arguments

base_size	Base font size.
-----------	-----------------

 validate_custom_delay *Validate a custom delay distribution for RTMB traceability*

Description

[Experimental]

Usage

```
validate_custom_delay(delay, test_theta = NULL, test_delays = 1:14)
```

Arguments

delay	A custom_delay_class object from <code>custom_delay()</code> .
test_theta	Optional numeric vector of length n_params to use as the test point. Defaults to the inits slot.
test_delays	Integer vector of delay values to evaluate (default 1:14).

Details

Tests the user-supplied `cdf / log_cdf / log_survival` on a dummy numeric vector, checking that `obj$fn()` and `obj$gr()` are finite. Emits a clear error (including common causes) on failure.

Value

Invisibly returns TRUE on success.

Examples

```
# Custom components tape USER functions, so RTMB must be attached
# (it is kept in Imports, not Depends, so attach it yourself):
library(RTMB)
weibull_cdf <- function(theta) {
  shape <- exp(theta[1]); scale <- exp(theta[2])
  function(d) 1 - exp(-(d / scale)^shape)
}
dly <- custom_delay(weibull_cdf,
  priors = list(normal_prior(0, 1), normal_prior(log(7), 1)),
  name = "Weibull", inits = c(0, log(7)))
validate_custom_delay(dly)
```

validate_custom_epidemic

Validate a user-defined epidemic process for RTMB traceability

Description

[Experimental]

Usage

```
validate_custom_epidemic(epidemic, test_theta = NULL)
```

Arguments

<code>epidemic</code>	A <code>custom_epidemic_class</code> object from <code>custom_epidemic()</code> .
<code>test_theta</code>	Optional numeric vector of length <code>n_params</code> to use as the test point. Defaults to <code>epidemic@inits</code> .

Details

Passes `intensity_fn` through `RTMB::MakeADFun` at the supplied (or default) initial values and confirms that the objective value and gradient are both finite. Emits an informative error if the function is not AD-safe.

Value

`epidemic`, invisibly. Emits a success message if the check passes.

Examples

```
# Custom components tape USER functions, so RTMB must be attached
# (it is kept in Imports, not Depends, so attach it yourself):
library(RTMB)
max_time <- 15L
rw_fn <- function(theta)
  matrix(theta[1] + cumsum(exp(theta[2]) * theta[3:(2L + max_time)]), max_time, 1L)
custom_epi <- custom_epidemic(
  rw_fn,
  priors = c(list(normal_prior(2, 1), normal_prior(-2, 0.5)),
    rep(list(std_normal_prior()), max_time)),
  inits = c(2, -2, rep(0, max_time))
)
validate_custom_epidemic(custom_epi)
```

Index

* datasets

- covid_colombia, 10
- ar1_epidemic (epidemic_process), 20
- auto_nowcast, 3
- auto_nowcast(), 8–10, 18, 23, 30, 41, 42
- autoplot(), 26, 31
- backtest, 6
- backtest(), 3–5, 8, 18, 30
- base::sample(), 40
- best_model, 8
- best_model(), 8
- best_model_name, 8
- best_model_name(), 8, 9
- best_score, 9
- best_score(), 10
- beta_prior (priors), 35
- cauchy_prior (priors), 35
- chi_square_prior (priors), 35
- coef(), 26, 28
- comparison_scores, 9
- comparison_scores(), 8, 9, 42
- covid_colombia, 10
- cumulative_process, 11
- cumulative_process(), 27, 30, 39
- custom_delay, 12
- custom_delay(), 4, 45
- custom_epidemic, 14
- custom_epidemic(), 4, 46
- default_priors, 16
- default_priors(), 18, 21, 23, 24
- delay_process, 14, 16, 17, 37, 39
- dirichlet_delay (delay_process), 17
- dirichlet_revision (revision_delay), 36
- diseasenowcasting_workflows, 5, 7, 18, 23, 30
- dn_palette, 19

- double_exponential_prior (priors), 35
- epidemic_process, 15, 16, 20
- exponential_prior (priors), 35
- extreme_values, 22
- fit, 22
- fit(), 45
- fit_check, 23
- fit_check(), 7, 18
- fix_param, 24
- flat_prior (priors), 35
- gamma_delay (delay_process), 17
- gamma_prior (priors), 35
- gamma_revision (revision_delay), 36
- generalized_gamma_delay (delay_process), 17
- generalized_gamma_revision (revision_delay), 36
- half_cauchy_prior (priors), 35
- half_double_exponential_prior (priors), 35
- half_normal_prior (priors), 35
- half_std_normal_prior (priors), 35
- half_student_t_prior (priors), 35
- hsgp_epidemic (epidemic_process), 20
- hsgp_epidemic(), 27
- infer_max_time, 24
- infer_max_time(), 15
- intensity function, 24
- inv_gamma_prior (priors), 35
- likelihood, 11, 16, 25
- load_nowcast, 26
- load_nowcast(), 41
- logistic_prior (priors), 35
- lognormal_delay (delay_process), 17
- lognormal_delay(), 27

[lognormal_prior \(priors\), 35](#)
[lognormal_revision \(revision_delay\), 36](#)
[lognormal_revision\(\), 38](#)

[mean\(\), 26, 28](#)
[median\(\), 26, 28](#)
[model, 27](#)
[model\(\), 4, 6–8, 16, 18, 23, 24, 28, 32, 34, 39](#)

[nb_likelihood \(likelihood\), 25](#)
[nb_likelihood\(\), 27, 35](#)
[normal_prior \(priors\), 35](#)
[normal_prior\(\), 15, 17](#)
[nowcast, 28](#)
[nowcast\(\), 3–5, 7, 8, 18, 23, 25, 26, 37–39, 41](#)
[nowcast\(prior_only = TRUE\), 16, 21](#)
[nowcast_diagnostic, 30](#)
[nowcast_diagnostic\(\), 18, 23](#)
[nowcast_twostage, 31](#)

[parameters, 32](#)
[parameters\(\), 26](#)
[poisson_likelihood \(likelihood\), 25](#)
[poisson_likelihood\(\), 27, 35](#)
[positive_flat_prior \(priors\), 35](#)
[predict\(\), 26, 28, 29, 41](#)
[prepare_data, 33](#)
[prepare_data\(\), 16, 23, 29](#)
[priors, 35](#)

[quantile\(\), 26, 28](#)

[revision_delay, 36, 39](#)
[revision_process, 38](#)
[revision_process\(\), 27, 36, 37](#)

[sample, 40](#)
[save_nowcast, 41](#)
[save_nowcast\(\), 26](#)
[scoringutils::add_relative_skill\(\), 4, 5, 19](#)
[scoringutils::as_forecast_point\(\), 19](#)
[scoringutils::as_forecast_quantile\(\), 19](#)
[scoringutils::as_forecast_sample\(\), 19](#)
[scoringutils::score\(\), 4, 5, 10, 19](#)
[selection_metric, 42](#)
[selection_timings, 42](#)
[selection_timings\(\), 10](#)
[sir_epidemic \(epidemic_process\), 20](#)

[std_normal_prior \(priors\), 35](#)
[std_normal_prior\(\), 27](#)
[student_t_prior \(priors\), 35](#)
[summarise_nowcast_matrix, 43](#)
[surprise, 43](#)
[surprise\(\), 45](#)
[surprise.list, 44](#)

[tbl.now::engine_diseasenowcasting\(\), 6, 18, 19](#)
[tbl.now::nowcast_backtest, 7](#)
[tbl.now::nowcast_backtest\(\), 6, 7, 18, 23](#)
[tbl.now::nowcast_ensemble\(\), 7](#)
[tbl.now::nowcast_weights\(\), 7](#)
[tbl.now::run_nowcast\(\), 19](#)
[tbl.now::score_nowcast\(\), 7, 23](#)
[tbl.now::tbl_now, 6](#)
[tbl.now::tbl_nowcast, 5, 18, 26, 28, 29](#)
[theme_diseasenowcasting, 45](#)

[update\(\), 22](#)

[validate_custom_delay, 45](#)
[validate_custom_delay\(\), 14](#)
[validate_custom_epidemic, 46](#)
[validate_custom_epidemic\(\), 15](#)

[weibull_prior \(priors\), 35](#)