

Package: stanpop (via r-universe)

September 22, 2026

Title Stan-based Poll of Polls Models

Version 1.0.1

Description Stan Models and code to run Bayesian poll of polls models.

License MIT + file LICENSE

Additional_repositories <https://mc-stan.org/r-packages/>

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports checkmate, digest, dplyr, ggplot2, lubridate, rlang, rstan,
tibble, yaml

Suggests abind, bayesplot, BH, cmdstanr, matrixStats, git2r,
posterior, RcppEigen, testthat (>= 2.1.0)

Remotes stan-dev/cmdstanr

Config/pak/sysreqs cmake make

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

Date/Publication 2026-09-22 13:00:08 UTC

RemoteUrl <https://github.com/MansMeg/stanpop>

RemoteRef HEAD

RemoteSha 458636a6c859f55d9378bb37ef7de4940485babd

Contents

assert_dates_in_time_line	4
assert_poll_data_in_time_line	4
assert_poll_data_in_time_line_using_time_scale	5
assert_stan_data_model	5
complete_poll_info	7
compute_g_scale_default	8
compute_gamma_parameters	8

compute_prediction_error	9
create_constrained_names_to_unconstrained_idx_map	9
create_unpar_one_to_one_map	10
curate_swedish_polls	10
dates_in_time_line	11
diagnose_extract_min_eigen_value_Omega	11
divergent_transitions_percentiles	12
elpd_known_state	12
existing_parties	13
extract	13
extract_poll_of_polls_refit_arguments	14
extract_pop_election_period	14
extract_pop_empirical_posterior_mean_x_change	15
format_election_period_to_string	16
format_obs_x_to_string	16
geom_known_state	17
geom_latent_state	17
geom_polls_time_weights	18
geom_stan_polls_data	19
german_elections	19
german_polls	20
get_adaptation_info	20
get_first_poll_as_simplex	21
get_known_state_lookback_windows	21
get_known_state_periods	22
get_last_evaluation_info	23
get_latent_state_for_dates	23
get_model_diagnostics	24
get_model_model_arguments	24
get_ndraws	25
get_polls_next_known_state_index	25
get_polls_time_points_between_known_states	26
get_polls_time_points_since_last_known_state	27
get_sampler_params	27
get_stan_date	28
get_stancode	28
get_time_line	29
get_time_points	29
get_time_points_from_time_line	30
house_levels	30
house_map	31
incorrect_draws	31
latent_state	31
latent_state_abind	32
latent_state_apply	32
latent_state_FUN	33
latent_state_percentiles	33
latent_state_subset	34

load_pop	34
log_prob	35
logMeanExp	35
model_config	36
model_dates_data_frame	36
parameter_names	37
parameter_names_from_unconstrained_index	37
parse_obs_x	38
pd_test	38
plot.polls_data	39
plot_parameters_areas	39
plot_parameters_bayesplot	40
plot_poll_of_polls	41
poll_id	42
poll_of_polls	43
polls_data	46
polls_time_reweight	47
recompile_stanfit	47
refit_poll_of_polls	48
remove_file_extension	49
remove_known_states	50
rerun_poll_of_polls	50
Rhat_is_above_m_by_parameter_block	51
sample_polls_y	51
save_pop	52
setup_latent_time_ranges	52
simulate_polls	53
spanish_elections	53
stan_data_add_missing	54
stan_polls_data	54
stan_polls_data_model	58
standard_error	61
stanpop	61
subset.polls_data	62
subset_latent_state_dates	62
subset_publish_dates	63
summarize_polls_time_weights	63
swedish_elections	64
swedish_party_time_ranges	64
swedish_polls	65
swedish_polls_curated	65
time_line	66
time_line_expand	67
time_range	67
time_ranges_are_disjoint	68
time_scale_dates	68
validate_backend_sample_arguments	69
x_predict	69

4

assert_poll_data_in_time_line

x_test 70

y_predict 70

Index 72

assert_dates_in_time_line

Assert that the dates are available within the time_line

Description

Assert that the dates are available within the time_line

Usage

assert_dates_in_time_line(dates, tl)

Arguments

- datesDates to check
- tlA [time_line] object to check against

assert_poll_data_in_time_line

Assert that the poll_data object dates are available within the time_line

Description

Assert that the poll_data object dates are available within the time_line

Usage

assert_poll_data_in_time_line(pd, tl)

Arguments

- pd[polls_data] to check if part of [time_line]
- tl[time_line] to check against

```
assert_poll_data_in_time_line_using_time_scale
```

Assert that the poll_data object dates are available within the [time_line] using time_scale

Description

The assertion creates a [time_line] using the [time_scale]

Usage

```
assert_poll_data_in_time_line_using_time_scale(pd, ts)
```

```
assert_poll_data_in_time_line_using_time_range(pd, ts, tr)
```

Arguments

pd	[polls_data] to check if part of [time_line]
ts	a time_scale to use
tr	a [time_range]

See Also

[time_range()]

```
assert_stan_data_model
```

Check Stan Data for specific model

Description

Check Stan Data for specific model

Usage

```
assert_stan_data_model(x)
```

```
## S3 method for class 'model2'
assert_stan_data_model(x)
```

```
## S3 method for class 'model3'
assert_stan_data_model(x)
```

```
## S3 method for class 'model5'
assert_stan_data_model(x)
```

```
## S3 method for class 'model6'  
assert_stan_data_model(x)  
  
## S3 method for class 'model6b'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8a'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8a1'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8a3'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8c'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8b'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8d'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8e'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8f'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8g'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8i'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8k'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8l'  
assert_stan_data_model(x)  
  
## S3 method for class 'model8m'  
assert_stan_data_model(x)  
  
## S3 method for class 'model2'  
assert_stan_data_model(x)
```

```
## S3 method for class 'model5'
assert_stan_data_model(x)

## S3 method for class 'model3'
assert_stan_data_model(x)

## S3 method for class 'model4'
assert_stan_data_model(x)

## S3 method for class 'model9'
assert_stan_data_model(x)
```

Arguments

x a [poll_of_polls] object

complete_poll_info *Return indicator on polls without missing values*

Description

Return indicator on polls without missing values

Usage

```
complete_poll_info(x)

complete_poll_data(x)

complete_poll_y(x)

complete_polls(x)
```

Arguments

x a [polls_data] object.

Details

[complete_poll_info()] indicate polls with complete poll_info.
 [complete_poll_data()] and [complete_poll_y()] indicate polls with complete poll data (or y)
 [complete_polls()] indicate polls with complete poll data and info

Value

a boolean vector

`compute_g_scale_default`*Compute the default value of g_scale*

Description

Compute the default value of g_scale

Usage

```
compute_g_scale_default(x)
```

Arguments

x a list with g and next_known_state_index

`compute_gamma_parameters`*Compute gamma parameters with same expectation and variances as a half-normal with parameter sigma*

Description

Compute gamma parameters with same expectation and variances as a half-normal with parameter sigma

Usage

```
compute_gamma_parameters(sigma, n = 1e+06)
```

Arguments

sigma the sigma parameter in a half-normal
n how many samples to approximate the half-normal

Value

a list with parameters a and b

compute_prediction_error

Compute prediction error for a poll_of_polls model

Description

Compute prediction error for a poll_of_polls model

Usage

```
compute_prediction_error(x, predict_state, type = "elpd")
```

Arguments

x	a [poll_of_polls] model
predict_state	the true value to predict
type	error measurement to compute

create_constrained_names_to_unconstrained_idx_map

Create a map between constrained parameter names and unconstrained parameter indecies

Description

Go through all unconstrained indecies and return all parameters that changes due to the constrained parameter.

Note that in a state space model a unconstrated parameter can change a whole series due to the model formulation.

Usage

```
create_constrained_names_to_unconstrained_idx_map(x, return_first = FALSE)
```

Arguments

x	a poll of polls object
return_first	return the first character element (the unconstrained, non-transformed parameter)

`create_unpar_one_to_one_map`*Semi-automatic construction of parameters for specific models*

Description

Semi-automatic construction of parameters for specific models

Usage

```
create_unpar_one_to_one_map(x)
```

Arguments

`x` a poll of polls object

Details

Create the names of the unconstrained parameters from specific models

`curate_swedish_polls` *Curate the Swedish polls dataset*

Description

Curate the Swedish polls dataset

Usage

```
curate_swedish_polls(  
  x,  
  parties = c("M", "L", "C", "KD", "S", "V", "MP", "SD", "FI")  
)
```

Arguments

`x` a SwedishPolls data.frame
`parties` parties to curate

dates_in_time_line	<i>What dates are available within the time_line</i>
--------------------	--

Description

What dates are available within the time_line

Usage

```
dates_in_time_line(dates, tl)
```

Arguments

dates	Dates to check
tl	A [time_line] object to check against

diagnose_extract_min_eigen_value_Omega	<i>Extract minimal eigen value per iteration</i>
--	--

Description

Extract minimal eigen value per iteration

Usage

```
diagnose_extract_min_eigen_value_Omega(x)
```

Arguments

x	a [poll_of_polls] object
---	--------------------------

divergent_transitions_percentiles

Extract the percentiles of the divergent transistions

Description

Extract the percentiles of the divergent transistions

Usage

```
divergent_transitions_percentiles(x)
```

Arguments

x a [poll_of_polls] object

Value

a matrix of the percentiles of the divergent transitions

elpd_known_state

Compute the elpd, percentiles and rmse for a true [known_state]

Description

Compute the elpd, percentiles and rmse for a true [known_state]

Usage

```
elpd_known_state(x, known_state)
```

```
## S3 method for class 'pop_model8f1'
```

```
elpd_known_state(x, known_state)
```

```
## S3 method for class 'pop_model8f2'
```

```
elpd_known_state(x, known_state)
```

```
rmse_known_state(x, known_state)
```

```
percentile_known_state(x, known_state)
```

Arguments

x a [poll_of_polls] object

known_state a [known_state] object

Details

Incorrect draws has been removed

existing_parties	<i>Return parties existing in a given model range</i>
------------------	---

Description

Return parties existing in a given model range

Usage

```
existing_parties(y, ltr, mtr)
```

Arguments

y	a character vector of party names
ltr	the [time_range] of the latent series
mtr	the [time_range] of the model

extract	<i>Extract parameters from the stan object</i>
---------	--

Description

Extract parameters from the stan object

Usage

```
extract(object, ...)

## S3 method for class 'poll_of_polls'
extract(object, ...)
```

Arguments

object	an object to extract parameter draws from
...	arguments supplied to rstan::extract.

```
extract_poll_of_polls_refit_arguments
```

Extract refit-ready constructor arguments from a poll_of_polls object

Description

‘extract_poll_of_polls_refit_arguments()’ returns the named constructor arguments for [poll_of_polls()] that are stored in a [poll_of_polls] object. Unlike ‘x\$input_args’, the returned list excludes backend sampler arguments from ‘...’, which are returned separately by [extract_poll_of_polls_sample_arguments()].

‘extract_poll_of_polls_input_arguments()’ is retained as a backward-compatible alias and returns the same refit-ready constructor arguments.

Usage

```
extract_poll_of_polls_refit_arguments(x)
```

```
extract_poll_of_polls_input_arguments(x)
```

```
extract_poll_of_polls_sample_arguments(x)
```

Arguments

x A pop object.

Value

‘extract_poll_of_polls_refit_arguments()’ and ‘extract_poll_of_polls_input_arguments()’ return a list with the named [poll_of_polls()] constructor arguments, excluding sampler arguments from ‘...’.

‘extract_poll_of_polls_sample_arguments()’ returns the sampler arguments that were supplied through ‘...’ when the model was fit.

```
extract_pop_election_period
```

Deprecated: extract election periods from a poll_of_polls object

Description

Deprecated. This helper encodes an older fixed-length election-period marker. New code should instead use [get_known_state_periods()] or [get_known_state_lookback_windows()] and apply any report-specific window naming or presentation downstream.

Usage

```
extract_pop_election_period(pop, election_period_length = 6)
```

Arguments

pop a poll of polls object

election_period_length the number of time points that make up the election period

Details

Kept for backward compatibility. It still returns a tibble with the election period as a boolean per time point.

See Also

[get_known_state_periods()], [get_known_state_lookback_windows()]

extract_pop_empirical_posterior_mean_x_change

Extract mean x change from a poll_of_polls object

Description

Extract mean x change from a poll_of_polls object

Usage

```
extract_pop_empirical_posterior_mean_x_change(pop, type)
```

Arguments

pop a poll of polls object

type type of

Details

Extracts the difference in the latent state between time points, either as the [diff]erence $x[t]-x[t-1]$ or the [ratio] $x[t]/x[t-1]$. Here $t=1$ is set to NA.

format_election_period_to_string

Format election period lists to string for printout

Description

Format election period lists to string for printout

Usage

```
format_election_period_to_string(x, period_marker = "--", collapse = NULL)
```

Arguments

x	a poll of polls object
period_marker	character used to combine election period dates to an election period
collapse	should the election periods be collapsed to a character of length one (NULL if not), then the string used to collapse the character vector elements.

format_obs_x_to_string

Format obs_x lists to string for printout

Description

Format obs_x lists to string for printout

Usage

```
format_obs_x_to_string(x, collapse = NULL)
```

Arguments

x	a poll of polls object
collapse	should the obs_x characters be collapsed to a character of length one (NULL if not), then the string used to collapse the character vector elements.

geom_known_state	<i>Visualize a known state in a poll_of_polls ggplot</i>
------------------	--

Description

Visualize a known state in a poll_of_polls ggplot

Usage

```
geom_known_state(x, y, ...)
```

Arguments

x	a known_states data.frame
y	the specific variable to visualize
...	further arguments to geom_line and geom_point

geom_latent_state	<i>Add a [latent_state] geom to a ggplot</i>
-------------------	--

Description

Add a [latent_state] geom to a ggplot

Usage

```
geom_latent_state(  
  x,  
  median = TRUE,  
  intervals = c(0.9, 0.75, 0.5),  
  latent_state_colour = "darkgrey",  
  ...  
)
```

Arguments

x	a [latent_state] object
median	should the median be plotted?
intervals	what intervals should be visualized
latent_state_colour	the color of the latent state
...	further arguments to [geom_ribbon] and [geom_line].

`geom_polls_time_weights`*Compute a time period weight tibble*

Description

Constructs a tibble with the weight of each day of a poll. The tibble can then be aggregated to the right level time scale. The first day of each year, month and week is used to set a date for each time point.

Usage

```
geom_polls_time_weights(x, y, ...)  
  
polls_time_weights(x, ...)  
  
## S3 method for class 'polls_data'  
polls_time_weights(x, week_start = 1, ...)  
  
## S3 method for class 'stan_polls_data'  
polls_time_weights(x, ...)  
  
polls_time_weights(x) <- value  
  
## S3 replacement method for class 'polls_data'  
polls_time_weights(x) <- value
```

Arguments

<code>x</code>	a [polls_data] object.
<code>y</code>	a vector to plot for the [polls_time_weights] object.
<code>...</code>	further arguments to [geom_point]
<code>week_start</code>	day on which week starts following ISO conventions - 1 means Monday, 7 means Sunday (default).
<code>value</code>	a [polls_time_weights] object.

Value

a [polls_time_weights] object (tibble)

geom_stan_polls_data *Visualize a stan_data object*

Description

Visualize a stan_data object

Usage

```
geom_stan_polls_data(x, ...)
```

Arguments

x a [stan_polls_data] object
... further arguments supplied to [geom_polls_time_weights()].

See Also

stan_polls_data

german_elections *German election data*

Description

German election data

Usage

```
german_elections
```

Format

A data frame with 4 elections, 2005-2017

Source

<http://www.wikipedia.com>

german_polls	<i>German polls data</i>
--------------	--------------------------

Description

German polls data

Usage

```
german_polls
german_polls_curated
```

Format

A data frame with 4609 polls and 6 parties:
 An object of class `data.frame` with 4606 rows and 15 columns.

Source

<http://www.wahlrecht.de/umfragen/>

get_adaptation_info	<i>Get adaptation info</i>
---------------------	----------------------------

Description

Extract the adaptation info (step size and diagonal invers mass matrix) from a `[poll_of_polls]` object

Usage

```
get_adaptation_info(x, as.list = FALSE, ...)
```

Arguments

<code>x</code>	a <code>[poll_of_polls]</code>
<code>as.list</code>	return an R list. Defaults to <code>[FALSE]</code> .
<code>...</code>	further arguemt past to <code>[rstan::get_adaptation_info]</code>

```
get_first_poll_as_simplex
```

The first poll is the poll with the earliest starting date

Description

The first poll is the poll with the earliest starting date

Usage

```
get_first_poll_as_simplex(x)
```

Arguments

x a stan_data object

```
get_known_state_lookback_windows
```

Return known-state lookback windows for a poll_of_polls object

Description

Return one lookback window per usable known-state evaluation date.

Usage

```
get_known_state_lookback_windows(pop, window = lubridate::period(months = 6))
```

Arguments

pop a [poll_of_polls] object
 window a scalar lubridate [Period] defining how far back each lookback window should start.

Details

This helper builds directly on [get_known_state_periods()] so lookback windows use the same stable set of usable evaluation dates. The ‘window’ argument must be a scalar lubridate [Period], for example ‘lubridate::period(months = 6)’ or ‘lubridate::days(180)’.

Lookback window semantics are:

1. ‘window_to’ is the current ‘evaluation_date’, 2. ‘window_from’ is ‘evaluation_date’ shifted backward by ‘window’, and 3. the returned rows retain the matching period metadata.

Calendar-aware month arithmetic is computed with ‘lubridate::%m-%’ so month-based lookback windows behave consistently around month ends.

Value

A tibble with one row per usable known-state date and the columns 'window_index', 'evaluation_date', 'window_from', 'window_to', 'period_index', 'previous_evaluation_date', 'period_from', 'period_to', and 'is_last'.

`get_known_state_periods`*Return known-state evaluation periods for a poll_of_polls object*

Description

Return one general evaluation period per usable known-state date.

Usage

```
get_known_state_periods(pop)
```

Arguments

`pop` a [poll_of_polls] object

Details

This function extract stable period definition. A usable evaluation date is a non-missing known-state date on or before the model end date. Returned dates are sorted and deduplicated before periods are constructed.

The period semantics are intentionally conservative:

1. 'period_to' is the current 'evaluation_date',
2. 'period_from' is the previous 'evaluation_date',
- and 3. the first period gets 'NA' for 'previous_evaluation_date' and 'period_from'.

Value

A tibble with one row per usable known-state date and the columns 'period_index', 'evaluation_date', 'previous_evaluation_date', 'period_from', 'period_to', and 'is_last'.

`get_last_evaluation_info`*Return the latest evaluation window for a poll_of_polls object*

Description

Return the model end date together with the latest known-state date that can be used for evaluation.

Usage

```
get_last_evaluation_info(pop)
```

Arguments

pop a [poll_of_polls] object

Details

This helper exists to extract definition of the "last evaluation window" instead of recomputing it ad hoc. In practice, reports often need to know:

1. when the fitted latent series ends, 2. what the most recent usable known-state date is, and 3. how many days separate those two dates.

Value

A one-row tibble with 'model_time_to', 'evaluation_date', and 'gap_days'. When there is no known state on or before the model end date, 'evaluation_date' and 'gap_days' are returned as missing values.

`get_latent_state_for_dates`*Get the latent state for specific dates*

Description

Get the latent state for specific dates

Usage

```
get_latent_state_for_dates(x, date)
```

Arguments

x a [poll_of_polls] or [latent_state] object
date a [date] vector

`get_model_diagnostics` *Get the model diagnostics used in a [poll_of_polls] object*

Description

Get the model diagnostics used in a [poll_of_polls] object

Usage

```
get_model_diagnostics(x)
```

Arguments

`x` a [poll_of_polls] object

Details

`no_divergent_transitions`: The number of draws with divergent transitions
`no_max_treedepth`: The number of draws where the max treedepth was reached
`no_low_bfmi_chains`: The number of chains with low E-BFMI
`no_Rhat_above_1_1`: The number of parameters with an Rhat above 1.1
`no_Rhat_is_NA`: The number of parameters without an Rhat value
`mean_no_leapfrog_steps`: The mean number of leapfrog steps taken per iteration/draw
`mean_chain_step_size`: The mean stepsize over the chains
`mean_chain_inv_mass_matrix_min_max`: The mean max and min of the invers mass matrix over the chains
`mean_chain_warmup_time`: the mean warmup time in seconds
`mean_chain_sampling_time`: the mean warmup time in seconds

Value

a list with diagnostic values

`get_model_model_arguments`
Get the model arguments used in a [poll_of_polls] object

Description

Get the model arguments used in a [poll_of_polls] object

Usage

```
get_model_model_arguments(x, all = TRUE)
```

Arguments

`x` a [poll_of_polls] object
`all` Should all model arguments be returned, or just the one that were set?

Value

a list with model arguments

get_ndraws	<i>Get the number of draws/samples in a [poll_of_polls] object</i>
------------	--

Description

Get the number of draws/samples in a [poll_of_polls] object

Usage

```
get_ndraws(x)
```

```
ndraws(x)
```

Arguments

x	a [poll_of_polls] object
---	--------------------------

get_polls_next_known_state_index	<i>Get the next known state index</i>
----------------------------------	---------------------------------------

Description

Get the next known state index

Usage

```
get_polls_next_known_state_index(x, known_state, type = "collection_midpoint")
```

```
get_time_line_next_known_state_index(time_line, known_state)
```

```
get_dates_next_known_state_index(dates, known_state)
```

Arguments

x	a [polls_data] object
known_state	a [known_state] object
type	how is the next state index defined. Default is based on [collection_period] dates.
time_line	time line to get next index for
dates	Dates to get next index for

Details

The function returns the index for the next known state. Polls without a next unknown state, gets index $[\text{nrow}(\text{known_state}) + 1]$.

```
get_polls_time_points_between_known_states
```

Get the number of time points between known state

Description

Get the number of time points between known state

Usage

```
get_polls_time_points_between_known_states(
  x,
  known_state,
  tl,
  type = "collection_midpoint"
)

get_time_line_time_points_between_known_states(tl, known_state)

get_dates_time_points_between_known_states(dates, known_state, tl)
```

Arguments

x	a [polls_data] object
known_state	a [known_state] object
tl	a [time_line] object to get the number of time points between known state
type	how is the next state index defined. Default is based on [collection_period] dates.
dates	Dates to get the number of time points between known states

Details

The function returns the number of time points between the known states.

get_polls_time_points_since_last_known_state

Get the number of time points since the last known state

Description

Get the number of time points since the last known state

Usage

```
get_polls_time_points_since_last_known_state(
  x,
  known_state,
  tl,
  type = "collection_midpoint"
)
```

```
get_time_line_time_points_since_last_known_state(tl, known_state)
```

```
get_dates_time_points_since_last_known_state(dates, known_state, tl)
```

Arguments

x	a [polls_data] object
known_state	a [known_state] object
tl	a [time_line] object to get the number of time points since last known state
type	how is the next state index defined. Default is based on [collection_period] dates.
dates	Dates to get the number of time points since the last known state

Details

The function returns the number of time points since the last known state.

get_sampler_params *Get Sampler Diagnostics*

Description

Get Sampler Diagnostics

Usage

```
get_sampler_params(object, ...)

## S3 method for class 'poll_of_polls'
get_sampler_params(object, ...)
```

Arguments

- object a [poll_of_polls] object to extract stan samples from
- ... arguments further supplied to [rstan::get_sampler_params()]

See Also

rstan::get_sampler_params

get_stan_date	<i>Extract the date when Stan was run</i>
---------------	---

Description

Extract the date when Stan was run

Usage

```
get_stan_date(object)
```

Arguments

- object a [poll_of_polls] object

get_stancode	<i>Get Stan Code from a pop object</i>
--------------	--

Description

Get Stan Code from a pop object

Usage

```
get_stancode(object, ...)

## S3 method for class 'poll_of_polls'
get_stancode(object, ...)
```

Arguments

object	a [poll_of_polls] object to extract stan code from
...	arguments further supplied to [rstan::get_stancode()]

See Also

rstan::get_stancode

get_time_line	<i>Get time line from polls or time range</i>
---------------	---

Description

Get time line from polls or time range

Usage

```
get_time_line(x, time_range, time_scale)
```

Arguments

x	a [polls_data] object
time_range	a time range object
time_scale	a time_scale

get_time_points	<i>Get the time points for specific dates</i>
-----------------	---

Description

Get the time points for specific dates

Usage

```
get_time_points(x, dates)

## S3 method for class 'time_line'
get_time_points(x, dates)

## S3 method for class 'poll_of_polls'
get_time_points(x, dates)
```

Arguments

x	a [time_line] or [poll_of_polls]
dates	a [date] vector

```
get_time_points_from_time_line
```

Get time points from a time line object

Description

Look up one index value from the [daily] table of a [time_line] object for each date in [dates]. This is typically used to convert calendar dates to latent time-point indices such as [time_line_t].

Usage

```
get_time_points_from_time_line(dates, tl, return_variable = "time_line_t")
```

```
get_total_time_points_from_time_line(tl)
```

Arguments

dates	a vector with dates to convert to time points
tl	a [time_line] object
return_variable	the daily variable (time point type) to return

```
house_levels
```

Get the levels/houses from a [poll_of_polls] object

Description

The house levels are extracted in the same order as h_i in the stan data object.

Usage

```
house_levels(x)
```

```
party_levels(x)
```

Arguments

x	a [poll_of_polls] object
---	--------------------------

house_map	<i>Map between houses and house names</i>
-----------	---

Description

Map between houses and house names

Usage

```
house_map
```

Format

A data.frame with house, house name and country.:

incorrect_draws	<i>What samples are correct?</i>
-----------------	----------------------------------

Description

What samples are correct?

Usage

```
incorrect_draws(x)
```

Arguments

x	a [poll_of_polls] object
---	--------------------------

latent_state	<i>Create a latent state object</i>
--------------	-------------------------------------

Description

Create a latent state object

Usage

```
latent_state(x, time_line = NULL, ...)

## S3 method for class 'poll_of_polls'
latent_state(x, time_line = NULL, ...)

## S3 method for class 'stanfit'
latent_state(x, time_line, dimnames, ...)
```

Arguments

x	an object to convert to a [latent_state]
time_line	a [time_line] object
...	further arguments to methods
dimnames	The dimnames attribute for the latent state array

latent_state_abind	<i>Bind latent_state objects</i>
--------------------	----------------------------------

Description

Bind latent_state objects

Usage

```
latent_state_abind(x, y, along, ...)
```

Arguments

x	a [latent_state] object
y	a [latent_state] object
along	the dimension to bind the latent states. See [abind::abind] for details.
...	further arguments to [abind::abind]

Value

a [latent_state] object

latent_state_apply	<i>Apply latent_state objects</i>
--------------------	-----------------------------------

Description

Apply latent_state objects

Usage

```
latent_state_apply(x, MARGIN, FUN, ...)
```

Arguments

x	a [latent_state] object to subset
MARGIN	margins to apply over
FUN	function to apply
...	further arguments to base::apply

Value

a [latent_state] object

latent_state_FUN	<i>Compute an arbitrary function of latent states</i>
------------------	---

Description

Compute an arbitrary function of latent states

Usage

```
latent_state_FUN(x, FUN, var_name)

## S3 method for class 'poll_of_polls'
latent_state_FUN(x, FUN, var_name)

## S3 method for class 'latent_state'
latent_state_FUN(x, FUN, var_name)
```

Arguments

x	a latent state
FUN	an arbitrary function
var_name	the variable name to use

latent_state_percentiles	<i>Compute functions of the latent state</i>
--------------------------	--

Description

Compute functions of the latent state

Usage

```
latent_state_percentiles(x, percentiles)

latent_state_mean(x)

latent_state_draw(x, draw_idx)

latent_state_sd(x)

latent_state_mean_dates(x, dates)
```

Arguments

x	a [latent_state]
percentiles	percentiles to compute
draw_idx	a draw index from the posterior
dates	a vector of date values

latent_state_subset	<i>Subset latent_state objects</i>
---------------------	------------------------------------

Description

Subset latent_state objects

Usage

```
latent_state_subset(x, iterations = NULL, t = NULL, categories = NULL)
```

Arguments

x	a [latent_state] object to subset
iterations	subset iterations
t	subset time points (t)
categories	subset categories

Value

a [latent_state] object

load_pop	<i>Load a poll_of_polls object saved with save_pop</i>
----------	--

Description

Read a wrapped '.rds' file created by [save_pop()] and return the stored [poll_of_polls] object. Files not created by [save_pop()] are rejected so the loader can rely on the wrapper metadata and avoid silently accepting arbitrary serialized objects.

Usage

```
load_pop(file)
```

Arguments

file	Path to a file previously created by [save_pop()].
------	--

Value

A [poll_of_polls] object.

log_prob

Stan Function for pop objects

Description

Stan Function for pop objects

Usage

```
log_prob(object, ...)

## S3 method for class 'poll_of_polls'
log_prob(object, ...)

## Default S3 method:
log_prob(object, ...)

get_num_upars(object, ...)

## S3 method for class 'poll_of_polls'
get_num_upars(object, ...)

## Default S3 method:
get_num_upars(object, ...)

get_num_pars(object)
```

Arguments

object	an object
...	arguments supplied to the rstan function

logMeanExp

Extract the data when Stan was run

Description

Extract the data when Stan was run

Usage

```
logMeanExp(x)
```

Arguments

x a numeric vector of logged values

model_config	<i>Set default values and assert arguments for a model configuration</i>
--------------	--

Description

Set default values and assert arguments for a model configuration

Usage

```
model_config(model, x = NULL, stan_data = NULL)
```

```
model_arguments(model)
```

Arguments

model	the model for which to set the model arguments
x	a list with model arguments (the model config) set by the user
stan_data	a list with the stan data used for the model

Details

With model arguments, the arguments that are set by the user for each poll_of_polls stan model. This differs between models. See [model_arguments()] for the model arguments used by each model.

model_dates_data_frame	<i>Return a data.frame with dates from a list of pop objects</i>
------------------------	--

Description

Return a data.frame with dates from a list of pop objects

Usage

```
model_dates_data_frame(x)
```

Arguments

x a list of pop objects

parameter_names	<i>Extract the parameter names and parameter counts from a poll_of_polls object</i>
-----------------	---

Description

Extract the parameter names and parameter counts from a poll_of_polls object

Usage

```
parameter_names(x, rm_idx = FALSE)
extract_parameter_names(x, rm_idx = FALSE)
parameter_block_names(x)
parameters_counts(x)
get_parameter_names(x, rm_idx = FALSE)
```

Arguments

x	a [poll_of_polls] object
rm_idx	remove parameter indecies (e.g. [1], [1,1], [1,1,1]) from the parameter names. Default is FALSE.

parameter_names_from_unconstrained_index	<i>Get parameter names from unconstrained space</i>
--	---

Description

The function returns the name and indecies of all parameters on the constrained space that are affected by the unconstrained parameter.

Usage

```
parameter_names_from_unconstrained_index(stanfit, up_idx)
parameter_block_names_from_unconstrained_index(stanfit, up_idx)
parameter_unique_names_from_unconstrained_index(stanfit, up_idx)
```

Arguments

stanfit a [stanfit] object to extract parameter names from
up_idx the index of parameters at the constrained space to extract names for

Value

a [data.frame] with one row per parameter affected by the unconstrained

parse_obs_x	<i>Parse a data frame of observed x (t distributed)</i>
-------------	---

Description

Parse a data frame of observed x (t distributed)

Usage

parse_obs_x(hyper_parameters, time_line, y_name)

Arguments

hyper_parameters a list that contains the slot obs_x with the dataframe
time_line the time_line to use to parse dates to time points
y_name the different values used to parse obs_x\$y to integers

pd_test	<i>A subset of swe_polls stored as a [polls_data] object</i>
---------	--

Description

A subset of swe_polls stored as a [polls_data] object

Usage

pd_test

Format

a [polls_data] object with 1007 polls.

Details

All polls from [swe_polls] without missing data during 2010-01-01 to 2019-12-31. See data-raw folder for exact construction.

plot.polls_data	<i>Plot a [polls_data] object</i>
-----------------	-----------------------------------

Description

Plot a [polls_data] object

Usage

```
## S3 method for class 'polls_data'
plot(x, y = NULL, publish_date = TRUE, collection_period = TRUE, ...)

geom_publish_date(x, y, ...)

geom_collection_period(x, y, ...)
```

Arguments

x	a [polls_data] object
y	a variable name in the [polls_data] object.
publish_date	visualize publication dates.
collection_period	visualize collection period.
...	further arguments to [geom_collection_period]

plot_parameters_areas	<i>Plot posterior distributions of parameters</i>
-----------------------	---

Description

Plot posterior distributions of parameters

Usage

```
plot_parameters_areas(
  x,
  params,
  params_plot_name = NULL,
  title = "Posterior distributions",
  ...
)

plot_parameters_mcmc_hex(
  x,
```

```

    params,
    params_plot_name = NULL,
    title = "Posterior distributions",
    ...
)

plot_parameters_mcmc_scatter(
  x,
  params,
  params_plot_name = NULL,
  title = "Posterior distributions",
  ...
)

plot_parameters_mcmc_intervals(
  x,
  params,
  params_plot_name = NULL,
  title = "Posterior distributions",
  ...
)

## S3 method for class 'poll_of_polls'
pairs(x, ...)

traceplot(x, ...)

## S3 method for class 'poll_of_polls'
traceplot(x, pars = NULL, inc_warmup = FALSE, ...)

```

Arguments

<code>x</code>	a [poll_of_polls] object
<code>params</code>	parameters to plot (see <code>parameter_names()</code> too see all parameters in model)
<code>params_plot_name</code>	names to use for parameters in plot
<code>title</code>	The title of the plot.
<code>...</code>	further arguments sent to <code>bayesplot::mcmc_areas</code> , <code>bayesplot::mcmc_hex</code> , etc.
<code>pars</code>	optional parameter names to include in the traceplot.
<code>inc_warmup</code>	should warmup draws be included when constructing the traceplot input?

plot_parameters_bayesplot

Plot posterior distributions of parameters

Description

Plot posterior distributions of parameters

Usage

```
plot_parameters_bayesplot(
  x,
  bayesplot_FUN,
  params,
  params_plot_name = NULL,
  ...
)
```

Arguments

x	a [poll_of_polls] object
bayesplot_FUN	a plotting function from the bayesplot package
params	parameters to plot (see parameter_names() too see all parameters in model)
params_plot_name	names to use for parameters in plot
...	further arguments sent to bayesplot::mcmc_area

plot_poll_of_polls	<i>Plot a [poll_of_polls] object</i>
--------------------	--------------------------------------

Description

Plot a [poll_of_polls] object

Usage

```
plot_poll_of_polls(
  x,
  y = NULL,
  from = NULL,
  to = NULL,
  publish_date = TRUE,
  collection_period = FALSE,
  shift_latent_days = 0,
  house = NULL,
  include_latent_state = TRUE,
  latent_date_position = c("period_start", "period_end"),
  ...
)

## S3 method for class 'poll_of_polls'
```

```
plot(x, ...)
```

```
geom_pop_line(x, y, ...)
```

Arguments

x	a [poll_of_polls] object
y	column to visualize.
from	plot from [date]
to	plot to [date]
publish_date	visualize publication dates.
collection_period	visualize collection period.
shift_latent_days	Shift the latent series right, this number of days.
house	only plot the following polling houses
include_latent_state	include latent state in plot
latent_date_position	Where to display latent-state dates. "period_start" uses the latent anchor/start dates stored in the timeline. "period_end" uses the last daily date mapped to each latent time point, which is aware of mixed time-scale timelines.
...	further arguments to [geom_line]

poll_id	<i>Get and set elements of a [data_polls] object</i>
---------	--

Description

Get and set elements of a [data_polls] object

Usage

```
poll_id(x)
```

```
poll_ids(x)
```

```
poll_id(x) <- value
```

```
poll_ids(x) <- value
```

```
start_date(x)
```

```
start_dates(x)
```

```
start_date(x) <- value
start_dates(x) <- value
end_date(x)
end_dates(x)
end_date(x) <- value
end_dates(x) <- value
publish_date(x)
publish_dates(x)
publish_date(x) <- value
publish_dates(x) <- value
n(x)
collection_midpoint_dates(x)
houses(x)
n(x) <- value
y(x)
y(x) <- value
```

Arguments

x	a [data_polls] object.
value	a value to set.

poll_of_polls	<i>Fit a poll of polls object</i>
---------------	-----------------------------------

Description

Fit a poll of polls object

Usage

```
poll_of_polls(
  y,
  model,
  polls_data,
  time_scale,
  time_scale_overrides = NULL,
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  hyper_parameters = NULL,
  slow_scales = NULL,
  backend = "rstan",
  compile_args = NULL,
  ...,
  cache_dir = file.path(tempdir(), "pop_cache")
)
```

Arguments

<code>y</code>	character vector indicating poll variable to use
<code>model</code>	poll of polls model to use (or a path to a stan model)
<code>polls_data</code>	a [polls_data] object
<code>time_scale</code>	the time scale to use, [day], [week], or [month].
<code>time_scale_overrides</code>	a [data.frame] with columns [from], [to], and [time_scale] that override the default [time_scale] for inclusive date ranges in the latent state. The [from] and [to] bounds are inclusive and override ranges must not overlap.
<code>known_state</code>	a [data.frame] with variables [date] and [x] indicating the known states for certain dates.
<code>model_time_range</code>	a [time_range] object that describe the period used for the latent state. Default is the [time_range] of the [polls_data] object.
<code>latent_time_ranges</code>	time ranges of the latent state of individual [y]s
<code>hyper_parameters</code>	hyperparameters to supply direct to the model
<code>slow_scales</code>	a vector of [Date]s that indicate breaks (right-inclusive) for a slower moving time scale.
<code>backend</code>	Stan backend to use. Supported values are [rstan] and [cmdstanr].
<code>compile_args</code>	additional arguments passed to [cmdstanr::cmdstan_model()] when 'backend = "cmdstanr"'. Ignored for 'backend = "rstan"'.
<code>...</code>	further arguments passed directly to the backend sampler. These go to [rstan::stan()] when 'backend = "rstan"' and to 'CmdStanModel\$sample()' when 'backend = "cmdstanr"'.

`cache_dir` directory to cache model. Default is cache in `tempdir()`. [NULL], no cache. Cached objects are stored with `[save_pop()]` in the same wrapped on-disk format used by the public save/load API. Existing cache files at the same hash are replaced when it reaches the save step.

Details

The `[input_args]` slot contain all input arguments except polls data and known state that are stored in the original object instead. The `[stan_arguments]` slot contains the backend sampling arguments supplied through `'...'`, except for the `'data'` argument which is stored in the `[stan_data]` slot. The `[compile_arguments]` slot contains the CmdStanR compilation arguments. The `[warm_start_state]` slot caches the reusable warm-start payload extracted from the fitted Stan object so future refits do not need to recover it from the backend fit after serialization.

For `'model8m10'`, `'hyper_parameters'` can include `'structural_bridge_type'`, `'structural_bridge_window'`, `'structural_bridge_x_target_path'`, `'structural_bridge_alpha_week'`, `'structural_bridge_sigma_scale'`, and `'structural_bridge_epsilon'`. Type 0 means no bridge, type 1 or `"x_drift"` enables a state-dependent x-scale drift for selected parties, and type 2 or `"constant_gain_pull"` enables a constant-gain convex pull toward the selected-party structural vote-share path. `'structural_bridge_window'` supplies one global inclusive calendar date range for both x-scale drift and bridge-specific sigma scaling. The bridge window is inclusive in calendar time; the bridge prior is applied to latent transitions after `'from'` and through `'to'`. The `'from'` and `'to'` dates must be exact latent anchor dates; use `'time_scale_overrides'` when bridge endpoints need to be represented exactly on a coarser latent grid. Known states and zero-day steps are forced inactive. For `"x_drift"`, `'from_x'` and `'to_x'` define the total drift size, `'to_x - from_x'`; they are not an attractor endpoint. For example, if the sampled party vote share at the bridge start is 0.028 and `'from_x = 0.025'`, `'to_x = 0.043'`, the bridge adds 0.018, so absent innovations the drift points toward 0.046, not 0.043. For `"constant_gain_pull"`, `'from_x'` and `'to_x'` define the structural target path and `'structural_bridge_alpha_week'` is the weekly gap-closing fraction. The derived `'structural_bridge_x_target_t'` is a full latent-grid target path and may include target values on inactive rows such as the bridge origin; `'structural_bridge_active_t'` determines where the bridge prior is applied. Sigma scaling must be strictly positive, applies only during active bridge steps, and scales eta coordinates, not vote-share points directly.

Example `'model8m10'` bridge setup:

```
time_scale_overrides <- tibble::tibble(
  from = as.Date("2026-06-04"),
  to = as.Date("2026-09-13"),
  time_scale = "day"
)

hyper_parameters <- list(
  structural_bridge_window = c(
    as.Date("2026-06-04"),
    as.Date("2026-09-13")
  ),
  structural_bridge_x_target_path = data.frame(
    y = "L",
    from_x = 0.025,
```

```

    to_x = 0.043
  ),
  structural_bridge_sigma_scale = c(
    M = 1, L = 0.5, C = 1, KD = 1,
    S = 1, V = 1, MP = 1, SD = 1
  )
)

```

polls_data

Create a [polls_data] object

Description

Create a [polls_data] object

Usage

```

polls_data(y, house, publish_date, start_date, end_date, n, poll_id = NULL)

as.polls_data(x, ...)

## S3 method for class 'polls_data'
x[i, keep_poll_id = TRUE, ...]

```

Arguments

y	a named [data.frame] with variables of interest to estimate over time.
house	a factor variable indicating polling house (of same length as rows in [y])
publish_date	the date of when the poll was published
start_date	the start date of the data collection period of each poll
end_date	the end date of the data collection period of each poll
n	the (integer) sample size of each poll
poll_id	the id of each poll. Deafult is [row.names] of [y].
x	an object to convert to a [polls_data] object.
...	further arguments to methods.
i	row in [polls_data] to select.
keep_poll_id	Should the [poll_id]

Examples

```
data(swedish_polls_curated)
pd <- polls_data(y = swedish_polls_curated[,3:11],
                 house = swedish_polls_curated$Company,
                 publish_date = swedish_polls_curated$PublDate,
                 start_date = swedish_polls_curated$collectPeriodFrom,
                 end_date = swedish_polls_curated$collectPeriodTo,
                 n = swedish_polls_curated$n)
```

`polls_time_reweight` *Reweights the weights based on function*

Description

Reweights the weights based on function

Usage

```
polls_time_reweight(x, poll_ids, FUN, ...)
```

Arguments

<code>x</code>	a [polls_time_weights] to reweight.
<code>poll_ids</code>	a character vector with poll_ids to reweight.
<code>FUN</code>	function used to reweight
<code>...</code>	further arguments to [FUN].

Details

The function [FUN] is used to compute the weights for integer values 0,...,T_i, where T_i is the length of the period for poll i. The values from poll i is then normalized to sum to 1.

`recompile_stanfit` *Recompile a poll_of_polls object*

Description

The function takes a [poll_of_polls] object and recompile the code on the data

Usage

```
recompile_stanfit(x)
```

Arguments

`x` a [poll_of_polls] object

Value

a recompiled [stanfit] object based on the model and data of the [poll_of_polls] object

`refit_poll_of_polls` *Refit a poll_of_polls object*

Description

`'refit_poll_of_polls()'` reconstructs a call to `[poll_of_polls()]` from an existing [poll_of_polls] object. The public API mirrors the practical refit use case: optionally replace `'polls_data'`, optionally choose refit-specific settings such as `'backend'`, `'compile_args'`, `'cache_dir'`, and `'warm_start'`, and use `'...'` only for backend sampler arguments.

Backend sampler arguments inherit from `'x'` and can be overridden or removed via `'...'`. Named `'NULL'` values remove stored sampler arguments.

Warm starts are filled in automatically unless you override them in `'warm_start'`. Omitted `'warm_start'` elements are taken from the last draw and sampler state of `'x'`. When `'polls_data'` changes, reused warm-start values are validated against the refit model's parameter dimensions before sampling begins.

Structural model inputs other than `'polls_data'` are always inherited from `'x'`; use `[poll_of_polls()]` directly to change `'y'`, `'model'`, `'time_scale'`, `'time_scale_overrides'`, `'known_state'`, `'model_time_range'`, `'latent_time_ranges'`, `'hyper_parameters'`, or `'slow_scales'`.

Usage

```
refit_poll_of_polls(
  x,
  polls_data = x$polls_data,
  backend = "cmdstanr",
  compile_args = x$compile_arguments,
  cache_dir = x$cache_dir,
  warm_start = list(),
  ...
)

reestimate_poll_of_polls(
  x,
  polls_data = x$polls_data,
  backend = "cmdstanr",
  compile_args = x$compile_arguments,
  cache_dir = x$cache_dir,
  warm_start = list(),
  ...
)
```

Arguments

<code>x</code>	Existing [poll_of_polls] object to refit.
<code>polls_data</code>	a [polls_data] object. Defaults to <code>'x\$polls_data'</code> .
<code>backend</code>	Stan backend to use for the refit. Only <code>'backend = "cmdstanr"'</code> is currently supported because RStan does not accept a supplied mass matrix. The argument remains explicit so a future version can extend backend support without changing the public API.
<code>compile_args</code>	additional arguments passed to <code>[cmdstanr::cmdstan_model()]</code> when <code>'backend = "cmdstanr"'</code> . Defaults to <code>'x\$compile_arguments'</code> .
<code>cache_dir</code>	directory to cache model. Defaults to <code>'x\$cache_dir'</code> .
<code>warm_start</code>	A named list controlling warm-start initialization. Supported elements are <code>'init'</code> , <code>'init_mode'</code> , <code>'inv_metric'</code> , <code>'metric_type'</code> , and <code>'step_size'</code> . Omitted elements use the automatic defaults extracted from <code>'x'</code> . Set an element to <code>'NULL'</code> to disable that default warm-start component. When supplied, <code>'init_mode'</code> must be one of <code>"last"</code> or <code>"random"</code> . The default is <code>"last"</code> , which reuses the final draw from each stored chain and therefore requires the refit to use the same number of chains as the stored fit. For example, <code>'warm_start = list(inv_metric = NULL)'</code> reuses the last draw but does not reuse the inverse metric.
<code>...</code>	Named backend sampler arguments supplied to <code>'CmdStanModel\$sample()'</code> . For <code>'backend = "cmdstanr"'</code> , these keep their usual CmdStanR meaning. If arguments are omitted, they are inherited from the stored arguments in <code>'x'</code> . Warm-start controls such as <code>'init'</code> , <code>'init_mode'</code> , <code>'inv_metric'</code> , <code>'metric_type'</code> , and <code>'step_size'</code> must be supplied through <code>'warm_start'</code> , not <code>'...'</code> .

Value

A refitted [poll_of_polls] object.

`remove_file_extension` *Remove file extension*

Description

Remove file extension

Usage

```
remove_file_extension(x)
```

Arguments

<code>x</code>	a string with a file name
----------------	---------------------------

remove_known_states	<i>function for removing known states outside timeline</i>
---------------------	--

Description

function for removing known states outside timeline

Usage

```
remove_known_states(
    psd,
    known_state,
    model_time_range = NULL,
    time_scale = "week"
)
```

Arguments

psd	[polls_data]
known_state	data.frame containg known result needs date (date of the known data)
model_time_range	time range of where model is active
time_scale	what scale is the data at

rerun_poll_of_polls	<i>Rerun a poll_of_polls object without warm-start reuse</i>
---------------------	--

Description

‘rerun_poll_of_polls()’ is a backward-compatible wrapper around [refit_poll_of_polls()]. It uses the refit implementation but disables automatic warm-start reuse by default, so the rerun behaves like a plain re-estimation unless you call [refit_poll_of_polls()] directly.

All arguments in ‘...’ are forwarded to [refit_poll_of_polls()] except ‘warm_start’, which is intentionally not supported here.

Usage

```
rerun_poll_of_polls(x, ...)
```

Arguments

x	Existing [poll_of_polls] object to rerun.
...	Arguments forwarded to [refit_poll_of_polls()], such as ‘polls_data’, ‘backend’, ‘compile_args’, ‘cache_dir’, and backend sampler arguments.

Value

A rerun [poll_of_polls] object.

Rhat_is_above_m_by_parameter_block
Extract more detailed diagnostic information

Description

Extract more detailed diagnostic information

Usage

```
Rhat_is_above_m_by_parameter_block(x, m = 1.1)
```

```
Rhat_is_above_m(x, m = 1.1)
```

```
Rhat_is_NA_by_parameter_block(x)
```

```
Rhat_is_NA(x)
```

Arguments

x	a [poll_of_polls] object
m	a limit to extract Rhat values above

sample_polls_y	<i>Sample a dependent variable based on the polls</i>
----------------	---

Description

Sample a dependent variable based on the polls

Usage

```
sample_polls_y(x, true_ls, tl, week_start = 1L)
```

Arguments

x	a [polls_data] object
true_ls	a numeric vector of the length as the [time_line]
tl	a [time_line] object.
week_start	the integer day starting the week (1 = Monday, 7 = Sunday).

Details

The [y] is sampled based on the [time_weights] and the [true_ls].

save_pop	<i>Save a poll_of_polls object</i>
----------	------------------------------------

Description

Save a [poll_of_polls] object in stanpop's wrapped on-disk format. The file contains a small meta-data wrapper around the original object so [load_pop()] can verify that the file was created by [save_pop()] rather than by a plain [base::saveRDS()] call. The same wrapped format is also used internally for cached objects created by [poll_of_polls()] when 'cache_dir' is not [NULL].

Usage

```
save_pop(x, file, compress = "xz", overwrite = FALSE)
```

Arguments

x	A [poll_of_polls] object.
file	Path to the output '.rds' file.
compress	Compression passed to [base::saveRDS()].
overwrite	Should an existing file at 'file' be overwritten?

Value

Invisibly returns 'x'.

setup_latent_time_ranges	<i>Construct a list of [time_range]s</i>
--------------------------	--

Description

Construct a list of [time_range]s

Usage

```
setup_latent_time_ranges(x, y, mtr, check_dates = TRUE)
```

Arguments

x	a list with a character vector with either "from" or "to"
y	all categories as a character vector
mtr	a model [time_range] that will be used as default
check_dates	assert that the dates of [x] is in the [mtr] interval

simulate_polls	<i>Simulate Polls</i>
----------------	-----------------------

Description

Simulate [npolls] polls for a given latent series [x], where the time slace is defined by [time_scale] and starting from [start_date]. Polling periods are randomly sampled from a a [polls_data] object.

Usage

```
simulate_polls(
  x,
  pd,
  npolls,
  time_scale = "week",
  start_date = as.Date("2010-01-04")
)
```

Arguments

x	a latent series to simulate polls for. Either a numeric vector or a data.frame.
pd	a [polls_data] object to sample poll periods and sample sizes from.
npolls	The total number of polls to simulate
time_scale	The time_scale used, [day], [week], [month] or [year].
start_date	The starting date for the polls.

spanish_elections	<i>Spanish election data</i>
-------------------	------------------------------

Description

Spanish election data

Usage

```
spanish_elections
```

Format

A data frame with 6 elections, 2008-2019

Source

<http://www.wikipedia.com>

`stan_data_add_missing` *Adds missing data through NA to stan data*

Description

Adds missing data through NA to stan data

Usage

```
stan_data_add_missing(stan_data)
```

Arguments

`stan_data` contatning \$y which can include missing in the form of na

`stan_polls_data` *Generate Stan Data from [polls_data] objects*

Description

Generate Stan Data from [polls_data] objects

Usage

```
stan_polls_data(
  x,
  y_name,
  model,
  time_scale = "week",
  time_scale_overrides = NULL,
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  hyper_parameters = NULL,
  slow_scales = NULL
)

stan_polls_data_model2(x, y_name, time_scale = "week", model_time_range = NULL)

stan_polls_data_model3(x, y_name, time_scale = "week", model_time_range = NULL)

stan_polls_data_model5(
  x,
  y_name,
  time_scale = "week",
```

```
        known_state = NULL,  
        model_time_range = NULL  
    )  
  
    stan_polls_data_model6(  
        x,  
        y_name,  
        time_scale = "week",  
        known_state = NULL,  
        model_time_range = NULL  
    )  
  
    stan_polls_data_model6b(  
        x,  
        y_name,  
        time_scale = "week",  
        known_state = NULL,  
        model_time_range = NULL,  
        latent_time_ranges = NULL  
    )  
  
    stan_polls_data_model7(  
        x,  
        y_name,  
        time_scale = "week",  
        known_state = NULL,  
        model_time_range = NULL,  
        latent_time_ranges = NULL  
    )  
  
    stan_polls_data_model8a(  
        x,  
        y_name,  
        time_scale = "week",  
        known_state = NULL,  
        model_time_range = NULL,  
        latent_time_ranges = NULL,  
        hyper_parameters = NULL  
    )  
  
    stan_polls_data_model8a1(  
        x,  
        y_name,  
        time_scale = "week",  
        known_state = NULL,  
        model_time_range = NULL,  
        latent_time_ranges = NULL,  
        hyper_parameters = NULL
```

```

)

stan_polls_data_model18b(
  x,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  hyper_parameters = NULL,
  slow_scales = NULL
)

stan_polls_data_model18c(
  x,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  hyper_parameters = NULL,
  slow_scales = NULL
)

stan_polls_data_model18d(
  x,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  hyper_parameters = NULL,
  slow_scales = NULL,
  model
)

```

Arguments

<code>x</code>	a [polls_data] object
<code>y_name</code>	a character vector indicating y variables in polls object.
<code>model</code>	model to get data for.
<code>time_scale</code>	to use.
<code>time_scale_overrides</code>	a [data.frame] with columns [from], [to], and [time_scale] that override the default [time_scale] for inclusive date ranges in the latent state. The [from] and [to] bounds are inclusive and override ranges must not overlap.
<code>known_state</code>	known time points in the latent state

model_time_range	the time range to model (for example for extrapolation)
latent_time_ranges	the time range of the latent state
hyper_parameters	a list with hyper parameters supplied to the model.
slow_scales	a vector of [Date]s that indicate breaks (right-inclusive) for a slower moving time scale. Example: If only 2010-01-15 is used, all dates up to and including 2010-01-15, will have s=1, Dates after 2010-01-15 will have s=2.

Details

For ‘model8m10’, ‘hyper_parameters’ can include ‘structural_bridge_type’, ‘structural_bridge_window’, ‘structural_bridge_x_target_path’, ‘structural_bridge_alpha_week’, ‘structural_bridge_sigma_scale’, and ‘structural_bridge_epsilon’. ‘structural_bridge_type = 0’ means no bridge, while ‘structural_bridge_type = 1’ or “x_drift” enables a state-dependent x-scale drift for selected parties. ‘structural_bridge_type = 2’ or “constant_gain_pull” enables a constant-gain convex pull toward the structural vote-share path for selected parties. ‘structural_bridge_window’ supplies one global inclusive calendar date range for both x-scale drift and bridge-specific sigma scaling. The high-level ‘structural_bridge_x_target_path’ data frame should contain ‘y’, ‘from_x’, and ‘to_x’; party names are matched through ‘y_name’. For “x_drift”, ‘from_x’ and ‘to_x’ define the total x-scale drift size, ‘to_x - from_x’, which is allocated over active bridge steps in proportion to ‘delta_days_t’. They are not an attractor endpoint. For example, if the sampled party vote share at the bridge start is 0.028 and ‘from_x = 0.025’, ‘to_x = 0.043’, the bridge adds 0.018, so absent innovations the drift points toward 0.046, not 0.043. For “constant_gain_pull”, ‘from_x’ and ‘to_x’ define the selected-party structural target path, and ‘structural_bridge_alpha_week’ is the weekly gap-closing fraction. A half-life of ‘H’ days corresponds to $1 - 2^{(-7 / H)}$. The derived low-level ‘structural_bridge_x_target_t’ is a full target-path matrix on the latent time grid. It can contain target values at rows where ‘structural_bridge_active_t == 0’, especially the bridge origin. ‘structural_bridge_active_t’ is the source of truth for where the bridge prior is applied in the state equation. Since the state transition is from ‘t - 1’ into ‘t’, the bridge origin usually has ‘active_t = 0’, while dates after the origin and up to the target date have ‘active_t = 1’. For example:

date	active_t	x_target_t[L]
2026-06-08	0	0.026
2026-06-09	1	0.0262
...		
2026-09-13	1	0.044

The bridge window is inclusive in calendar time; the bridge prior is applied to latent transitions after ‘from’ and through ‘to’. The ‘from’ and ‘to’ dates must be exact latent anchor dates in ‘time_line\$time_line\$date’; use ‘time_scale_overrides’ when bridge endpoints need to be represented exactly on a coarser latent grid. Known states and zero-day steps are forced inactive. ‘structural_bridge_sigma_scale’ is ordered by ‘y_name’ when named, must be strictly positive, applies only during active bridge steps, and scales eta-coordinate innovations rather than vote-share points directly.

Example ‘model8m10’ bridge setup:

```

time_scale_overrides <- tibble::tibble(
  from = as.Date("2026-06-04"),
  to = as.Date("2026-09-13"),
  time_scale = "day"
)

hyper_parameters <- list(
  structural_bridge_window = c(
    as.Date("2026-06-04"),
    as.Date("2026-09-13")
  ),
  structural_bridge_x_target_path = data.frame(
    y = "L",
    from_x = 0.025,
    to_x = 0.043
  ),
  structural_bridge_sigma_scale = c(
    M = 1, L = 0.5, C = 1, KD = 1,
    S = 1, V = 1, MP = 1, SD = 1
  )
)

```

The returned object keeps the existing [stan_data] and [time_line] fields unchanged for current models. A parallel future path is attached in [stan_data_with_overrides] and [time_line_with_overrides], built using [time_line_with_overrides()].

stan_polls_data_model *Compute the posterior predictive distribution of polls*

Description

Compute the posterior predictive distribution of polls

Usage

```

stan_polls_data_model(
  spd,
  psd,
  y_name,
  time_scale,
  known_state,
  model_time_range,
  latent_time_ranges,
  ...
)

## S3 method for class 'model2'

```

```
stan_polls_data_model(  
  spd,  
  psd,  
  y_name,  
  time_scale = "week",  
  known_state = NULL,  
  model_time_range = NULL,  
  ...  
)  
  
## S3 method for class 'model3'  
stan_polls_data_model(  
  spd,  
  psd,  
  y_name,  
  time_scale = "week",  
  known_state = NULL,  
  model_time_range = NULL,  
  ...  
)  
  
## S3 method for class 'model4'  
stan_polls_data_model(  
  spd,  
  psd,  
  y_name,  
  time_scale = "week",  
  known_state = NULL,  
  model_time_range = NULL,  
  ...  
)  
  
## S3 method for class 'model5'  
stan_polls_data_model(  
  spd,  
  psd,  
  y_name,  
  time_scale = "week",  
  known_state = NULL,  
  model_time_range = NULL,  
  ...  
)  
  
## S3 method for class 'model6'  
stan_polls_data_model(  
  spd,  
  psd,  
  y_name,
```

```
    time_scale = "week",
    known_state = NULL,
    model_time_range = NULL,
    ...
)

## S3 method for class 'model6b'
stan_polls_data_model(
  spd,
  psd,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  ...
)

## S3 method for class 'model6c'
stan_polls_data_model(
  spd,
  psd,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  ...
)

## S3 method for class 'model7'
stan_polls_data_model(
  spd,
  psd,
  y_name,
  time_scale = "week",
  known_state = NULL,
  model_time_range = NULL,
  latent_time_ranges = NULL,
  ...
)

## S3 method for class 'model9'
stan_polls_data_model(
  spd,
  psd,
  y_name,
  time_scale = "week",
```

```

    known_state = NULL,
    model_time_range = NULL,
    latent_time_ranges = NULL,
    ...
  )

```

Arguments

spd	a [stan_polls] objects
psd	a [poll_of_polls] object
y_name	names of interested parties
time_scale	one what scale is the data
known_state	states where we know the data exactly
model_time_range	what is the time range where the model is active
latent_time_ranges	not certain
...	extra arguments

standard_error	<i>Compute standard errors for poll estimates</i>
----------------	---

Description

Compute standard errors for poll estimates

Usage

```
standard_error(x, y, type = "binomial")
```

Arguments

x	a [polls_data] object
y	variable to compute standrad error for
type	type of standard error estimator

stanpop	<i>stanpop: Package to run Stan for poll of polls models.</i>
---------	---

Description

A package to run poll of polls models and Bayesian election prediction models

Author(s)

Maintainer: Mans Magnusson <mons.magnusson@gmail.com>

subset.polls_data	<i>Subset a [polls_data] object.</i>
-------------------	--------------------------------------

Description

Subset a [polls_data] object.

Usage

```
## S3 method for class 'polls_data'
subset(x, subset, select, drop = FALSE, ...)
```

Arguments

x	a [polls_data]
subset, select, drop, ...	
	see [subset()]

subset_latent_state_dates	<i>Subset a latent state based on a time range</i>
---------------------------	--

Description

Subset a latent state based on a time range

Usage

```
subset_latent_state_dates(x, from = NULL, to = NULL, ...)

subset_time_line_dates(x, from = NULL, to = NULL, ...)
```

Arguments

x	a [latent_state] object
from	subset from [date]
to	subset to [date]
...	Currently not in use.

subset_publish_dates *Subset a [polls_data] object based on a specific [time_range]*

Description

Subset a [polls_data] object. * [subset_publish_dates()] subset the data with the polls that has a publication date within the interval. * [subset_dates()] subset the data with the polls that has all non-[NA] dates in the interval.

Usage

```
subset_publish_dates(x, from = NULL, to = NULL, ...)
```

```
subset_dates(x, from = NULL, to = NULL, ...)
```

Arguments

x	a [polls_data] object
from	a the earliest date to include polls for
to	a the latest date to include polls for
...	further arguments

summarize_polls_time_weights
 Summarize a polls weight object by a time line

Description

Summarize a polls weight object by a time line

Usage

```
summarize_polls_time_weights(ptw, tl)
```

Arguments

ptw	a [polls_time_weights] object.
tl	a [time_line] object.

Value

a [polls_time_weights] object.

swedish_elections	<i>Swedish election data</i>
-------------------	------------------------------

Description

Swedish election data

Usage

```
swedish_elections
```

Format

A data frame with 4 elections, 2006-2018

Source

<http://www.wikipedia.com>

swedish_party_time_ranges	<i>Party time ranges</i>
---------------------------	--------------------------

Description

Party time ranges

Usage

```
swedish_party_time_ranges
```

```
german_party_time_ranges
```

```
spanish_party_time_ranges
```

Format

A named list (parties) with starting and ending dates.

An object of class `list` of length 1.

An object of class `list` of length 2.

Details

Time ranges for political parties

swedish_polls	<i>Swedish polls data</i>
---------------	---------------------------

Description

Swedish polls data

Usage

```
swedish_polls
```

Format

A data frame with 1565 polls and 9 parties:

Source

<http://www.github.com/MansMeg/SwedishPolls>

swedish_polls_curated	<i>Spanish polls data</i>
-----------------------	---------------------------

Description

Spanish polls data

Usage

```
swedish_polls_curated
```

```
spanish_polls
```

```
spanish_polls_curated
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 1660 rows and 18 columns.

A data frame with 1852 polls and 7 parties:

An object of class `data.frame` with 1345 rows and 16 columns.

Source

<http://www.wikipedia.com>

time_line	<i>Construct a time_line</i>
-----------	------------------------------

Description

Construct a timeline to use based on the [time_scale] of interest.

Usage

```
time_line(
  x,
  time_scale,
  ...,
  week_start = getOption("lubridate.week.start", 1)
)

## S3 method for class 'numeric'
time_line(
  x,
  time_scale,
  start_date,
  ...,
  week_start = getOption("lubridate.week.start", 1)
)

## S3 method for class 'data.frame'
time_line(
  x,
  time_scale,
  start_date,
  ...,
  week_start = getOption("lubridate.week.start", 1)
)
```

Arguments

x	a vector of the same length as the [time_line], a [polls_data] object, or a [time_range] object.
time_scale	The time_scale, [day], [week] or [month].
...	Further arguments to methods.
week_start	What is week starting day (1 = Monday, Default or 7 = Sunday).
start_date	The starting date of the [time_line].

Details

The timeline is created based on full months, week or days. If a [start_date] is supplied, length(x) no of days/weeks/months units are added, including the day/week/month of [start_date]. Hence, the number of total days of the [time_line] will change, based on the weekday of [start_date].

If [time_range] is supplied, the [time_line] will include every day/week/month that overlaps the closed interval defined by [time_range], including partial periods at both boundaries.

Value

a [time_line] object. A list with two data frames, one for a [daily] [time_line] and one with the [time_line] for the chosen [time_scale]. Both [data.frames] consist of a [date] variable and a [t] index column.

time_line_expand	<i>Expand a [time_line] object to a new time range but keep the same time point index.</i>
------------------	--

Description

Rebuild a [time_line] on a larger [time_range] while shifting the new daily and latent indices so that dates already present in [tl] keep the same [t] and [time_line_t] values as before.

Usage

```
time_line_expand(tl, time_range)
```

Arguments

tl	a time_line to expand
time_range	a new time range to expand to

time_range	<i>Time range</i>
------------	-------------------

Description

Time range

Usage

```
time_range(x, ...)

time_range(x) <- value

time_range_polls(x, ...)
```

Arguments

x	an object to compute the time range for.
...	further arguments supplied to method.
value	a Date vector of length 2 with names [from] and [to].

Details

Compute the time range for an object with [time_range] or compute the time range of the polls in the object with [time_range_polls].

time_ranges_are_disjoint

Check if two time ranges are disjoint (not overlapping)

Description

Check if two time ranges are disjoint (not overlapping)

Usage

```
time_ranges_are_disjoint(tr1, tr2)
```

Arguments

tr1	first [time_range]
tr2	second [time_range]

time_scale_dates

Convert from days to time scale

Description

Convert each date in [x] to the start date of its corresponding [time_scale] bucket/latent slot. For example, dates can be mapped to the same week start or month start when constructing a latent time line.

Usage

```
time_scale_dates(
  x,
  time_scale,
  week_start = getOption("lubridate.week.start", 1)
)
```

Arguments

x	a vector of dates to convert to a specific [time_scale]
time_scale	to use.
week_start	Day on which week starts following ISO conventions - 1 means Monday (default), 7 means Sunday. You can set lubridate.week.start option to control this parameter globally.

 validate_backend_sample_arguments

Validate backend-specific sampler arguments

Description

Checks sampler arguments before a backend fit starts. This rejects common argument names from the other Stan backend so callers supply arguments in the style expected by the selected backend.

Usage

```
validate_backend_sample_arguments(backend, sample_arguments)
```

Arguments

backend	Stan backend to validate against. Supported values are [rstan] and [cmdstanr].
sample_arguments	A list of arguments intended for the backend sampler.

Value

Invisibly returns 'TRUE' when validation succeeds.

 x_predict

Compute the posterior predictive distribution of polls

Description

Compute the posterior predictive distribution of polls

Usage

```
x_predict(x, y = NULL, t = NULL, ...)
```

```
## S3 method for class 'pop_model6b'
```

```
x_predict(x, y = NULL, t = NULL, ...)
```

```
## S3 method for class 'pop_model6c'
```

```
x_predict(x, y = NULL, t = NULL, ...)
```

Arguments

x	a [poll_of_polls] object
y	a vector of categories to compute the predicted distribution for
t	a vector of time points to compute the predictive for for the latent state. If [NULL], the whole latent state is returned. If [t] exist in latent state, the latent state is returned.
...	Currently not in use.

x_test	<i>Four true latent states used in experiments</i>
--------	--

Description

Four true latent states used in experiments

Usage

```
x_test
```

Format

A list of length four with numeric vectors of length 100.

Details

See data-raw folder for exact construction.

y_predict	<i>Compute the posterior predictive distribution of polls</i>
-----------	---

Description

Compute the posterior predictive distribution of polls

Usage

```
y_predict(x, y = NULL, poll_ids = NULL, ...)

poll_predictive_distribution(x, y = NULL, poll_ids = NULL, ...)

## S3 method for class 'pop_model6b'
y_predict(x, y = NULL, poll_ids = NULL, ...)

## S3 method for class 'pop_model6c'
y_predict(x, y = NULL, poll_ids = NULL, ...)
```

Arguments

<i>x</i>	a [poll_of_polls] object
<i>y</i>	a vector of categories to compute the predicted distribution for
<i>poll_ids</i>	a vector of poll ids to compute the predictive for. If [NULL] all are computed.
<i>...</i>	Currently not used.

Index

* datasets

- german_elections, 19
- german_polls, 20
- house_map, 31
- pd_test, 38
- spanish_elections, 53
- swedish_elections, 64
- swedish_party_time_ranges, 64
- swedish_polls, 65
- swedish_polls_curated, 65
- x_test, 70
- [.polls_data (polls_data), 46
- as.polls_data (polls_data), 46
- assert_dates_in_time_line, 4
- assert_poll_data_in_time_line, 4
- assert_poll_data_in_time_line_using_time_range, 5
 - (assert_poll_data_in_time_line_using_time_scale), 5
- assert_poll_data_in_time_line_using_time_scale, 5
- assert_stan_data_model, 5
- collection_midpoint_dates (poll_id), 42
- complete_poll_data
 - (complete_poll_info), 7
- complete_poll_info, 7
- complete_poll_y (complete_poll_info), 7
- complete_polls (complete_poll_info), 7
- compute_g_scale_default, 8
- compute_gamma_parameters, 8
- compute_prediction_error, 9
- create_constrained_names_to_unconstrained_index_map, 9
- create_unpar_one_to_one_map, 10
- curate_swedish_polls, 10
- dates_in_time_line, 11
- diagnose_extract_min_eigen_value_Omega, 11
- divergent_transitions_percentiles, 12
- elpd_known_state, 12
- end_date (poll_id), 42
- end_date<- (poll_id), 42
- end_dates (poll_id), 42
- end_dates<- (poll_id), 42
- existing_parties, 13
- extract, 13
- extract_parameter_names
 - (parameter_names), 37
- extract_poll_of_polls_input_arguments
 - (extract_poll_of_polls_refit_arguments), 14
- extract_poll_of_polls_refit_arguments, 14
- extract_poll_of_polls_sample_arguments
 - (extract_poll_of_polls_refit_arguments), 14
- extract_pop_election_period, 14
- extract_pop_empirical_posterior_mean_x_change, 15
- format_election_period_to_string, 16
- format_obs_x_to_string, 16
- geom_collection_period
 - (plot.polls_data), 39
- geom_known_state, 17
- geom_latent_state, 17
- geom_polls_time_weights, 18
- geom_pop_line (plot_poll_of_polls), 41
- geom_publish_date (plot.polls_data), 39
- geom_stan_polls_data, 19
- german_elections, 19
- german_party_time_ranges
 - (swedish_party_time_ranges), 64
- german_polls, 20
- german_polls_curated (german_polls), 20
- get_adaptation_info, 20

get_dates_next_known_state_index
 (get_polls_next_known_state_index),
 25
 get_dates_time_points_between_known_states
 (get_polls_time_points_between_known_states),
 26
 get_dates_time_points_since_last_known_state
 (get_polls_time_points_since_last_known_state),
 27
 get_first_poll_as_simplex, 21
 get_known_state_lookback_windows, 21
 get_known_state_periods, 22
 get_last_evaluation_info, 23
 get_latent_state_for_dates, 23
 get_model_diagnostics, 24
 get_model_model_arguments, 24
 get_ndraws, 25
 get_num_pars (log_prob), 35
 get_num_upars (log_prob), 35
 get_parameter_names (parameter_names),
 37
 get_polls_next_known_state_index, 25
 get_polls_time_points_between_known_states,
 26
 get_polls_time_points_since_last_known_state,
 27
 get_sampler_params, 27
 get_stan_date, 28
 get_stancode, 28
 get_time_line, 29
 get_time_line_next_known_state_index
 (get_polls_next_known_state_index),
 25
 get_time_line_time_points_between_known_states
 (get_polls_time_points_between_known_states),
 26
 get_time_line_time_points_since_last_known_state
 (get_polls_time_points_since_last_known_state),
 27
 get_time_points, 29
 get_time_points_from_time_line, 30
 get_total_time_points_from_time_line
 (get_time_points_from_time_line),
 30
 house_levels, 30
 house_map, 31
 houses (poll_id), 42
 incorrect_draws, 31
 latent_state, 31
 latent_state_abind, 32
 latent_state_apply, 32
 latent_state_draw
 (latent_state_percentiles), 33
 latent_state_FUN, 33
 latent_state_mean
 (latent_state_percentiles), 33
 latent_state_mean_dates
 (latent_state_percentiles), 33
 latent_state_percentiles, 33
 latent_state_sd
 (latent_state_percentiles), 33
 latent_state_subset, 34
 load_pop, 34
 log_prob, 35
 logMeanExp, 35
 model_arguments (model_config), 36
 model_config, 36
 model_dates_data_frame, 36
 n (poll_id), 42
 n<- (poll_id), 42
 ndraws (get_ndraws), 25
 pairs.poll_of_polls
 (plot_parameters_areas), 39
 parameter_block_names
 (parameter_names), 37
 parameter_block_names_from_unconstrained_index
 (parameter_names_from_unconstrained_index),
 37
 parameter_names, 37
 parameter_names_from_unconstrained_index,
 37
 parameter_unique_names_from_unconstrained_index
 (parameter_names_from_unconstrained_index),
 37
 parameters_counts (parameter_names), 37
 parse_obs_x, 38
 party_levels (house_levels), 30
 pd_test, 38
 percentile_known_state
 (elpd_known_state), 12
 plot.poll_of_polls
 (plot_poll_of_polls), 41

plot.polls_data, 39
 plot_parameters_areas, 39
 plot_parameters_bayesplot, 40
 plot_parameters_mcmc_hex
 (plot_parameters_areas), 39
 plot_parameters_mcmc_intervals
 (plot_parameters_areas), 39
 plot_parameters_mcmc_scatter
 (plot_parameters_areas), 39
 plot_poll_of_polls, 41
 poll_id, 42
 poll_id<- (poll_id), 42
 poll_ids (poll_id), 42
 poll_ids<- (poll_id), 42
 poll_of_polls, 43
 poll_predictive_distribution
 (y_predict), 70
 polls_data, 46
 polls_time_reweight, 47
 polls_time_weights
 (geom_polls_time_weights), 18
 polls_time_weights<-
 (geom_polls_time_weights), 18
 publish_date (poll_id), 42
 publish_date<- (poll_id), 42
 publish_dates (poll_id), 42
 publish_dates<- (poll_id), 42

 recompile_stanfit, 47
 reestimate_poll_of_polls
 (refit_poll_of_polls), 48
 refit_poll_of_polls, 48
 remove_file_extension, 49
 remove_known_states, 50
 rerun_poll_of_polls, 50
 Rhat_is_above_m
 (Rhat_is_above_m_by_parameter_block),
 51
 Rhat_is_above_m_by_parameter_block, 51
 Rhat_is_NA
 (Rhat_is_above_m_by_parameter_block),
 51
 Rhat_is_NA_by_parameter_block
 (Rhat_is_above_m_by_parameter_block),
 51
 rmse_known_state (elpd_known_state), 12

 sample_polls_y, 51
 save_pop, 52

 setup_latent_time_ranges, 52
 simulate_polls, 53
 spanish_elections, 53
 spanish_party_time_ranges
 (swedish_party_time_ranges), 64
 spanish_polls (swedish_polls_curated),
 65
 spanish_polls_curated
 (swedish_polls_curated), 65
 stan_data_add_missing, 54
 stan_polls_data, 54
 stan_polls_data_model, 58
 stan_polls_data_model2
 (stan_polls_data), 54
 stan_polls_data_model3
 (stan_polls_data), 54
 stan_polls_data_model5
 (stan_polls_data), 54
 stan_polls_data_model6
 (stan_polls_data), 54
 stan_polls_data_model6b
 (stan_polls_data), 54
 stan_polls_data_model7
 (stan_polls_data), 54
 stan_polls_data_model8a
 (stan_polls_data), 54
 stan_polls_data_model8a1
 (stan_polls_data), 54
 stan_polls_data_model8b
 (stan_polls_data), 54
 stan_polls_data_model8c
 (stan_polls_data), 54
 stan_polls_data_model8d
 (stan_polls_data), 54
 standard_error, 61
 stanpop, 61
 stanpop-package (stanpop), 61
 start_date (poll_id), 42
 start_date<- (poll_id), 42
 start_dates (poll_id), 42
 start_dates<- (poll_id), 42
 subset.polls_data, 62
 subset_dates (subset_publish_dates), 63
 subset_latent_state_dates, 62
 subset_publish_dates, 63
 subset_time_line_dates
 (subset_latent_state_dates), 62
 summarize_polls_time_weights, 63

swedish_elections, [64](#)
swedish_party_time_ranges, [64](#)
swedish_polls, [65](#)
swedish_polls_curated, [65](#)

time_line, [66](#)
time_line_expand, [67](#)
time_range, [67](#)
time_range<- (time_range), [67](#)
time_range_polls (time_range), [67](#)
time_ranges_are_disjoint, [68](#)
time_scale_dates, [68](#)
traceplot (plot_parameters_areas), [39](#)

validate_backend_sample_arguments, [69](#)

x_predict, [69](#)
x_test, [70](#)

y (poll_id), [42](#)
y<- (poll_id), [42](#)
y_predict, [70](#)