Package 'rplanes'

July 17, 2024

Title Plausibility Analysis of Epidemiological Signals

Version 0.1.0

Description Provides functionality to prepare data and analyze plausibility of both forecasted and reported epidemiological signals. The functions implement a set of plausibility algorithms that are agnostic to geographic and time resolutions and are calculated independently then presented as a combined score.

License MIT + file LICENSE

URL https://signaturescience.github.io/rplanes/

Depends R (>= 2.10)

Imports dplyr, dtw, ecp, lubridate, magrittr, purrr, readr, rlang, stringr, tibble, tidyr, utils

Suggests DT, ggplot2, markdown, knitr, MMWRweek, rmarkdown, shiny, shinyjs, shinyWidgets, shinycssloaders, testthat (>= 3.0.0), tools

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author VP Nagraj [aut, cre] (<https://orcid.org/0000-0003-0060-566X>), Desiree Williams [aut], Amy Benefield [aut]

Maintainer VP Nagraj <nagraj@nagraj.net>

Repository CRAN

Date/Publication 2024-07-17 10:50:02 UTC

Contents

`check_incomplete` *Check completeness of seed and signal data*

Description

This unexported helper is used internally in valid_dates to optionally issue a warning for potential completeness of seed and signal data based on dates provided.

Usage

```
check_incomplete(seed_date, signal_date, resolution)
```

Arguments

`seed_date`	Last date available in seed object
`signal_date`	First date available in signal object
`resolution`	Character vector specifying the temporal resolution (e.g., "weeks", "months")

Value

Operates as side-effect and returns a `warning()` if there are the seed and signal dates combined indicate an incomplete week or month.

`create_sliding_windows_df` *Sliding windows*

Description

This unexported helper function is used within `plane_shape()` to generate sliding windows from a vector and return a data frame where each row is a subset (a sliding window) of a time series. The length of the each windowed time series (and therefore number of columns) is equal to "window_size". The number of windows is equal to `(length(vector) - window_size) + 1`. For example, given a time series of length 38 and a window size of length 4, then there will be 35 windowed time series (rows), with 4 time stamps each (columns).

Usage

```
create_sliding_windows_df(vector, window_size)
```

Arguments

`vector`	A numeric or integer vector that is the time series to be used to create sliding windows
`window_size`	An integer specifying the size (i.e., number of elements) of the windowed time series desired

Value

A `data.frame` where each row is a subset (a sliding window) of a time series.

`cutter` *Cut into categorical differences*

Description

This unexported helper function takes an input number for an observed difference and cuts it into a categorical description (e.g., "increase", "decrease", or "stable") of the change.

Usage

```
cutter(x, threshold = 1)
```

Arguments

x	Vector of length 1 with scaled difference to be categorized
threshold	Limit used to define the categorical differences; default is 1

Value

Character vector of length 1 with the categorical description of difference

---

epiweek_start *Epiweek start*

Description

This unexported helper identifies the date of the first day for the epiweek of the given date. The function is used internally inside of valid_dates.

Usage

```
epiweek_start(date)
```

Arguments

date	Date to be queried

Value

Date of the first day of the epiweek for the input date.

---

get_shapes *Determine shapes*

Description

This unexported helper function is used to identify the shape in the plane_shape() function's scaled difference ("sdiff") method.

Usage

```
get_shapes(input_data, window_size)
```

Arguments

input_data	A data frame containing at least two columns, one of which must be named "value" with the value assessed and another named "dates" with the date for the observed data
window_size	The number of of categorical differences used to define the shape

Value

A vector with the shapes identified. Each element of the vector will include a shape, which is a cluster of categorical differences (of the same size as the specified "window_size") collapsed with ";" (e.g., `c("decrease;stable;stable;stable","stable;stable;stable;increase","stable;stable;increase;incre`

---

is_forecast *Check forecast*

Description

This function checks if the object is of class `signal` and `forecast`.

Usage

```
is_forecast(x)
```

Arguments

x	Input object to be checked

Value

Logical as to whether or not the input object inherits the "signal" and "forecast" classes.

Examples

```
## get path to example forecast file
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
ex_forecast <- read_forecast(fp)
sig <- to_signal(ex_forecast, outcome="flu.admits", type="forecast", horizon=4, resolution="weeks")
is_forecast(sig)
```

---

is_observed *Check observed*

Description

This function checks if the object is of class `signal` and `observed`.

Usage

```
is_observed(x)
```

Arguments

x	Input object to be checked

Value

Logical as to whether or not the input object inherits the "signal" and "observed" classes.

Examples

```
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
sig <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")
is_observed(sig)
```

---

month_start *Month start*

Description

This unexported helper identifies the date of the first day of the month for the given date. The function is used internally inside of valid_dates.

Usage

```
month_start(date)
```

Arguments

date	Date to be queried

Value

Date of the first day of the month for the input date.

---

plane_cover *Coverage component*

Description

This function evaluates whether or not the evaluated signal interval covers the last observed value. The interval used in this plausibility component is drawn from the upper and lower bounds of the forecasted prediction interval. As such, the only accepted signal format is forecast, which will include upper and lower bounds.

Usage

```
plane_cover(location, input, seed)
```

Arguments

location	Character vector with location code; the location must appear in input and seed
input	Input signal data to be scored; object must be one of forecast
seed	Prepared seed

Value

A list with the following values:

- **indicator**: Logical as to whether or not the last value falls outside of the interval (e.g., not in between lower and upper bounds of prediction interval) of the evaluated signal
- **last_value**: A vector with the last value recorded in the seed
- **bounds**: A list with a two elements corresponding to the upper and lower bounds of the evaluated signal interval

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_cover(location = "08", input = prepped_forecast, seed = prepped_seed)
plane_cover(location = "47", input = prepped_forecast, seed = prepped_seed)
```

---

plane_diff *Difference component*

Description

This function implements the point-to-point difference plausibility component. Differences in evaluated signals are calculated from input values iteratively subtracted from the previous values (i.e., for each x at time point i, the difference will be calculated as xi - xi-1). The plausibility analysis uses the evaluated differences to compare against the maximum difference observed and recorded in the seed.

Usage

```
plane_diff(location, input, seed)
```

Arguments

location	Character vector with location code; the location must appear in input and seed
input	Input signal data to be scored; object must be one of forecast or observed
seed	Prepared seed

Value

A list with the following values:

- **indicator**: Logical as to whether or not the absolute value of any of the evaluated differences exceeds the maximum difference
- **values**: A vector with the values assessed including the last value in seed concatenated with the evaluated signal values
- **evaluated_differences**: A vector with the consecutive differences for the values
- **maximum_difference**: A vector with one value for the maximum difference observed in seed

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_diff(location = "10", input = prepped_forecast, seed = prepped_seed)
plane_diff(location = "51", input = prepped_forecast, seed = prepped_seed)
```

---

plane_repeat *Repeat component*

Description

This function evaluates whether consecutive values in observations or forecasts are repeated a k number of times. This function takes in a forecast or observed object that is either from an observed dataset or forecast dataset. Note that if a signal is contant (i.e., the same value is repeated for all time points) then the repeat component will return FALSE.

Usage

```
plane_repeat(location, input, seed, tolerance = NULL, prepend = NULL)
```

Arguments

location	Character vector with location code; the location must appear in input and seed
input	Input signal data to be scored; object must be one of forecast or observed
seed	Prepared seed
tolerance	Integer value for the number of allowed repeats before flag is raised. Default is NULL and allowed repeats will be determined from seed.
prepend	Integer value for the number of values from seed to add before the evaluated signal. Default is NULL and the number of values will be determined from seed.

Value

A list with the following values:

- **indicator**: Logical as to whether or not the value is repeated sequentially k number of times.
- **repeats**: A tibble with repeating values found. If there are no repeats (i.e., indicator is FALSE) then the tibble will have 0 rows.

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
## use defaults
plane_repeat(location = "12", input = prepped_forecast, seed = prepped_seed)
## set tolerated repeats to 2
plane_repeat(location = "12", input = prepped_forecast, seed = prepped_seed, tolerance = 2)

## use defaults
plane_repeat(location = "49", input = prepped_forecast, seed = prepped_seed)
## set number of values prepended for evaluation to 4
plane_repeat(location = "49", input = prepped_forecast, seed = prepped_seed, prepend = 4)
```

plane_score *Score PLANES components*

Description

This function wraps PLANES scoring for specified components across all locations in single step.

Usage

```
plane_score(input, seed, components = "all", args = NULL, weights = NULL)
```

Arguments

input	Input signal data to be scored; object must be one of forecast or observed
seed	Prepared seed
components	Character vector specifying component; must be either "all" or any combination of "cover", "diff", "taper", "trend", "repeat", "shape", and "zero"; default is "all" and will use all available components for the given signal
args	Named list of arguments for component functions. List elements must be named to match the given component and arguments passed as a nested list (e.g., args = list("trend" = list("sig_lvl" = 0.05))). Default is NULL and defaults for all components will be used
weights	Named vector with weights to be applied; default is NULL and all components will be equally weighted; if not NULL then the length of the vector must equal the number of components, with each component given a numeric weight (see Examples). Specified weights must be real numbers greater than or equal to 1.

Value

A list with scoring results for all locations.

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))

hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")
```

```
## run plane scoring with all components
plane_score(input = prepped_forecast, seed = prepped_seed)

## run plane scoring with select components
plane_score(input = prepped_forecast, seed = prepped_seed, components = c("cover","taper"))

## run plane scoring with all components and additional args
trend_args <- list("sig_lvl" = 0.05)
repeat_args <- list("prepend" = 4, "tolerance" = 8)
shape_args <- list("method" = "dtw")
comp_args <- list("trend" = trend_args, "repeat" = repeat_args, "shape" = shape_args)
plane_score(input = prepped_forecast, seed = prepped_seed, args = comp_args)

## run plane scoring with specific components and weights
comps <- c("cover", "taper", "diff")
wts <- c("cover" = 1.5, "taper" = 1, "diff" = 4)
plane_score(input = prepped_forecast, seed = prepped_seed, components = comps, weights = wts)
```

`plane_seed` *Create seed*

Description

This function wraps the seed_engine to operate across all locations in the input signal.

Usage

```
plane_seed(input, cut_date = NULL)
```

Arguments

`input`	Input signal data used for seeding; must be an observed signal object
`cut_date`	Maximum date (inclusive) for which seeding should be performed; default is `NULL` and the entire input will be used for seeding

Value

A named `list` of length *n*, where multiple elements corresponding to seed characteristics and metadata for each of the *n* locations are nested in independent lists.

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")
```

```
## prepare seed with no cut date
plane_seed(prepped_observed)

## prepare seed with cut date
plane_seed(prepped_observed, cut_date = "2022-10-29")
```

`plane_shape` *Shape component*

Description

This function identifies the shape of the trajectory for a forecasted signal to compare against existing shapes in seed data. If the shape is identified as novel, a flag is raised, and the signal is considered implausible. See the Details section for further information.

Usage

```
plane_shape(location, input, seed, method = "sdiff")
```

Arguments

`location`	Character vector with location code; the location must appear in input and seed
`input`	Input signal data to be scored; object must be one of forecast
`seed`	Prepared seed
`method`	The method for determining shapes; must be one of "sdiff" or "dtw" (see Details); default is "sdiff"

Details

The approach for determining shapes can be customized by the user with the `plane_shape()` "method" argument. The two methods available are "sdiff" (default) and "dtw". Compared with "sdiff", the "dtw" method has been shown to have a higher sensitivity, lower specificity, and much greater computational cost in some circumstances. The "sdiff" method is recommended if computational efficiency is a concern.

The "sdiff" method will use consecutive scaled differences to construct shapes. The algorithm operates in three steps:

1. The prepared seed data is combined with forecasted point estimates and each point-to-point difference is calculated.
2. The differences are centered and scaled, then cut into categories. Differences greater than or equal to one standard deviation above the mean of differences are considered an "increase". Differences less than or equal to one standard deviation below the mean of differences are considered a "decrease". All other differences are considered "stable".
3. The categorical differences are then combined into windows of equal size to the forecasted horizon. Collectively these combined categorical differences create a "shape" (e.g., "increase;stable;stable;decrease").

4. Lastly, the algorithm compares the shape for the forecast to all of the shapes observed. If the shape assessed has not been previously observed in the time series then a flag is raised and the indicator returned is `TRUE`.

The "dtw" method uses a Dynamic Time Warping (DTW) algorithm to identify shapes within the seed data and then compares the shape of the forecast input signal to the observed shapes. This is done in three broad steps:

1. The prepared seed data is divided into a set of sliding windows with a step size of one, each representing a section of the overall time series. The length of these windows is determined by the horizon length of the input data signal (e.g., 2 weeks). For example, if the seed data was a vector, `c(1, 2, 3, 4, 5)`, and the horizon length was 2, then the sliding windows for the observed seed data would be: `c(1, 2)`, `c(2, 3)`, `c(3, 4)`, and `c(4, 5)`. Each sliding window is a subset of the total trajectory shape of the observed data.
2. Shape-based DTW distances are calculated for every 1x1 combination of the observed sliding windows and are stored in a distance matrix. These distances calibrate the function for identifying outlying shapes in forecast data. The algorithm finds the minimum distances for each windowed time series to use as a baseline for "observed distances" between chunks of the larger observed time series. The maximum of those minimum distances across the observed time series is set as the threshold. If the minimum of the forecast:observed distance matrix is greater than the threshold, then the forecast is inferred to be unfamiliar (i.e., a novel shape).
3. Next, the algorithm calculates the shape-based DTW distances between the forecast signal (including the point estimate, lower, and upper bounds) and every observed sliding window. If the distance between the forecast and any observed sliding window is less than or equal to the threshold defined above, then this shape is not novel and no flag is raised (indicator is `FALSE`).

Value

A `list` with the following values:

- **indicator**: Logical as to whether or not the the shape of the evaluated signal is novel (`TRUE` if shape is novel, `FALSE` if a familiar shape exists in the seed)

References

Toni Giorgino. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. Journal of Statistical Software, 31(7), 1-24. doi:10.18637/jss.v031.i07

Tormene, P.; Giorgino, T.; Quaglini, S. & Stefanelli, M. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. Artif Intell Med, 2009, 45, 11-34. doi:10.1016/j.artmed.2008.11.007

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))

tmp_hosp <-
 hosp %>%
 dplyr::select(date, location, flu.admits) %>%
 dplyr::mutate(date = as.Date(date))
```

```
prepped_observed <- to_signal(tmp_hosp,
                              outcome = "flu.admits",
                              type = "observed",
                              resolution = "weeks")
## read in example forecast and prep forecast signal
prepped_forecast <- read_forecast(system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv",
                                              package = "rplanes")) %>%
   to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_shape(location = "37", input = prepped_forecast, seed = prepped_seed)

## run plane component with DTW method
plane_shape(location = "37", input = prepped_forecast, seed = prepped_seed, method = "dtw")
```

`plane_taper` *Taper component*

Description

This function evaluates whether or not the evaluated signal interval tapers (i.e., decreases in width) as horizons progress. The interval used in this plausibility component is drawn from the upper and lower bounds of the forecasted prediction interval. As such, the only accepted signal format is forecast, which will include upper and lower bounds.

Usage

```
plane_taper(location, input, seed)
```

Arguments

`location`	Character vector with location code; the location must appear in input and seed
`input`	Input signal data to be scored; object must be one of forecast
`seed`	Prepared seed

Value

A `list` with the following values:

- **indicator**: Logical as to whether or not the prediction interval width tapers with advancing horizons
- **widths**: Consecutive interval widths for forecasted data

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_taper(location = "19", input = prepped_forecast, seed = prepped_seed)
plane_taper(location = "44", input = prepped_forecast, seed = prepped_seed)
```

`plane_trend` *Trend component*

Description

This function identifies any change points in the forecast data or in the final observed data point. Change points are identified by any significant change in magnitude or direction of the slope of the time series.

Usage

```
plane_trend(location, input, seed, sig_lvl = 0.1)
```

Arguments

`location`	Character vector with location code; the location must appear in input and seed
`input`	Input signal data to be scored; object must be forecast
`seed`	Prepared seed
`sig_lvl`	The significance level at which to identify change points (between zero and one); default is `0.1`

Details

This function uses e.divisive(), which implements a hierarchical divisive algorithm to identify change points based on distances between segments (calculated using equations 3 and 5 in Matteson and James, 2014; the larger the distance, the more likely a change point). Then a permutation test is used to calculate an approximate p-value.

The input to `e.divisive()` is transformed using differencing (i.e., `diff(x)` instead of the raw data, `x`). This slightly changes the way that change points are identified, as the index aligns with the gap

between points rather than the points themselves. Instead of identifying a change point based on the change in size between two points, it identifies change points based on the change in the change itself. The dataframe below illustrates the difference between `x` and `diff(x)`:

Index	x	diff(x)
1	3	6
2	9	0
3	9	28
4	37	37
5	74	1
6	75	0
7	75	0

Given this data, `e.divisive(x)` would identify index 5 (74) as the change point, because there was a jump of +37 between index 4 and 5. But `e.divisive(diff(x))` would pick both index 3 (28) and 5 (1), because there was a jump of +28 from index 2 and 3, and there was a jump of -36 between index 4 and 5.

Internally, the trend function uses an extra argument to `e.divisive()` for `min.size = 2`, which requires a gap of at least 2 points between detecting change points. This can indirectly increase the significance level or decrease the number of change points identified.

Value

A `list` with the following values:

- **indicator**: Logical as to whether or not the any forecast data or the final observed data point are a significant change point
- **output**: An n x 7 tibble. The length of the forecast plus the observed data determine the length of n. The columns are:
  - **Location**: A character vector with the location code
  - **Index**: An integer index of all observed and forecast data
  - **Date**: The dates corresponding to all observed and forecast data (formatted as date)
  - **Value**: The incidence of all observed and forecast data (e.g., hospitalization rates)
  - **Type**: Indicates whether the data row is observed or forecast data
  - **Changepoint**: Logical identifying any change point (whether in observed or forecast data). A TRUE is returned if any point is determined a change point based on the user defined significance level (sig_lvl).
  - **Flagged**: Logical indicating whether or not the change point was flagged. Change points are only flagged if they are in the forecast data or are the final observed data point. A TRUE is returned if the Changepoint is TRUE and is a final observed data point or any forecast point.
- **flagged_dates**: The date of any flagged change point(s). If there are none, NA is returned

References

Matteson, D. S., & James, N. A. (2014). A nonparametric approach for multiple change point analysis of multivariate data. Journal of the American Statistical Association, 109(505), 334–345. https://doi.org/10.1080/01621459.2013.849605

Matteson DS, James NA (2013). "A Nonparametric Approach for Multiple Change Point Analysis of Multivariate Data." ArXiv e-prints. To appear in the Journal of the American Statistical Association, 1306.4933.

Gandy, A. (2009) "Sequential implementation of Monte Carlo tests with uniformly bounded resampling risk." Journal of the American Statistical Association.

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
tmp_hosp <-
  hosp %>%
  dplyr::select(date, location, flu.admits) %>%
  dplyr::mutate(date = as.Date(date))

prepped_observed <- to_signal(tmp_hosp, outcome = "flu.admits",
                              type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
prepped_forecast <- read_forecast(system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv",
                                              package = "rplanes")) %>%
   to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_trend(location = "05", input = prepped_forecast, seed = prepped_seed, sig_lvl = .2)
## change location
plane_trend(location = "09", input = prepped_forecast, seed = prepped_seed, sig_lvl = .2)
## change sig_lvl
plane_trend(location = "06", input = prepped_forecast, seed = prepped_seed, sig_lvl = .05)
```

`plane_zero` *Zero component*

Description

This function checks for the presence of any value(s) equal to zero in the evaluated signal. If there are any zeros found, then the function assesses whether or not any zeros have been observed in the seed for the given location. If so, the function will consider the evaluated zero plausible and no flag will be raised (i.e., indicator returned as `FALSE`). If not, the function will consider the evaluated zero implausible and a flag will be raised (i.e., indicator returned as `TRUE`).

Usage

```
plane_zero(location, input, seed)
```

Arguments

location	Character vector with location code; the location must appear in input and seed
input	Input signal data to be scored; object must be one of forecast or observed
seed	Prepared seed

Value

A list with the following values:

- **indicator**: Logical as to whether or not there are zeros in evaluated signal but not in seed data

Examples

```
## read in example observed data and prep observed signal
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
hosp$date <- as.Date(hosp$date, format = "%Y-%m-%d")
prepped_observed <- to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

## read in example forecast and prep forecast signal
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
prepped_forecast <- read_forecast(fp) %>%
  to_signal(., outcome = "flu.admits", type = "forecast", horizon = 4)

## prepare seed with cut date
prepped_seed <- plane_seed(prepped_observed, cut_date = "2022-10-29")

## run plane component
plane_zero(location = "10", input = prepped_forecast, seed = prepped_seed)
plane_zero(location = "51", input = prepped_forecast, seed = prepped_seed)
```

---

q_boundary *Quantile boundary*

Description

This unexported helper generates a vector of lower bound, median, and upper bound for the prediction interval of specified width. The function is used internally inside of read_forecast.

Usage

```
q_boundary(pi_width)
```

Arguments

pi_width	Interval width as an integer

Value

Vector of quantiles corresponding to lower and upper bounds centered on median.

---

read_forecast *Read in forecast file*

Description

This function reads a probabilistic ("quantile") forecast csv file and prepares it for the to_signal function and downstream plausibility analysis. The quantile forecast file can be either a "legacy" or "hubverse" format (see Details for more information). The object returned is a `tibble` with summarized forecast data (i.e., prediction interval) for each location and horizon in the original file.

Usage

```
read_forecast(file, pi_width = 95, format = "legacy")
```

Arguments

file	Path to csv file containing quantile forecasts
pi_width	Width of prediction interval as integer; default `95` corresponds to 95% prediction interval
format	Format of the probabilistic format file; must be one of `"legacy"` or `"hubverse"` (see Details for more information); default is `"legacy"`

Details

The probabilistic forecast format has been used by multiple forecasting hubs. In general, this format includes one row per combination of quantile, location, target, and horizon. At each row the forecasted value is provided. The specific format, including columns required, has changed over time. This function accommodates the "legacy" as well as more recent "hubverse" formats. For more details on specific columns and see the links in the References.

Value

A `tibble` with the following columns:

- **location**: Geographic unit such as FIPS code
- **date**: Date corresponding the forecast horizon
- **horizon**: Forecast horizon
- **lower**: Lower limit of the prediction interval for the forecast
- **point**: Point estimate for the forecast
- **upper**: Upper limit of the prediction interval for the forecast

References

Hubverse: https://hubdocs.readthedocs.io/en/latest/user-guide/model-output.html

Legacy: https://github.com/cdcepi/Flusight-forecast-data/tree/master/data-forecasts#forecast-file-format

Examples

```
## read in example forecast and prep forecast signal (legacy format)
fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
read_forecast(fp)

fp2 <- system.file("extdata/forecast/2023-11-04-SigSci-TSENS.csv", package = "rplanes")
read_forecast(fp2, format = "hubverse")
```

---

resolve_resolution *Resolve resolution*

Description

This helper function uses argument matching to resolve the resolution from input. The function also handles casing. This will allow, for example, an input resolution of "daily" or "day" to be resolved to "days".

Usage

```
resolve_resolution(resolution)
```

Arguments

resolution	Character vector specifying the temporal resolution (e.g., "days", "weeks", "months")

Value

If the resolution matches to "days", "weeks", or "months" then the match will be returned. If not, the function will throw an error.

rplanes_explorer rplanes *explorer app launcher*

Description

The rplanes explorer app allows a user to interactively upload their own data (or view an internal example) to explore the plausibility analysis functionality.

Usage

```
rplanes_explorer(...)
```

Arguments

...	Additional arguments to be passed to shiny::runApp

Value

This function operates as a side-effect and starts the rplanes Shiny app.

Examples

```
## Not run:
# Launch the explorer app
rplanes_explorer(host = "0.0.0.0",
                 launch.browser = TRUE,
                 port = 80)

## End(Not run)
```

seed_engine *Seed engine*

Description

This helper function is used inside of plane_seed to evaluate characteristics of observed data to use for downstream plausibility analysis.

Usage

```
seed_engine(input, location, cut_date = NULL)
```

Arguments

input	Input signal data used for seeding; must be an observed signal object
location	Character vector with location code
cut_date	Maximum date (inclusive) for which seeding should be performed; default is NULL and the entire input will be used for seeding

Value

A `list` of length 1 with multiple elements corresponding to seed characteristics and metadata for the given location.

`to_chunk` *Chunk a vector*

Description

This unexported helper function creates a list with contents of a vector spit into chunks. The user can specify how large each chunk should be with the "size" argument.

Usage

```
to_chunk(x, size)
```

Arguments

`x`	Vector to be split into chunks as large as the "size" specified
`size`	Width of the chunks for "x" vector

Value

A list with as many elements as the number of chunks created. Each element will include vector with a length equal to the "size" specified.

`to_signal` *Create signal object*

Description

This function creates an object of the S3 class "signal". The user can conditionally specify either a "forecast" or "observed" signal.

Usage

```
to_signal(
  input,
  outcome,
  type = "observed",
  resolution = "weeks",
  horizon = NULL
)
```

Arguments

`input`	Data to be converted to signal; see "Details" for more information
`outcome`	Name of the outcome column in the input data
`type`	Signal type; must be one of "observed" or "forecast"; default is "observed"
`resolution`	The temporal resolution of the signal; data can be aggregated daily, weekly, or monthly; default is `"weeks"`; see "Details" for more information
`horizon`	Number of time steps ahead for forecast signals; only used if `type="forecast"`; default is `NULL`

Details

The input signal data may be either "observed" or "forecast" data. Depending on the type, the input data must conform to certain format prior to submission. In both cases, the data must be passed as a data frame.

For "observed" data the data frame must at minimum include columns for **location** (geographic unit such as FIPS code) and **date** (date of reported value; must be `date` class). The data should also include a column that contains the outcome (e.g., case count).

For "forecast" data the data frame must include columns for **location** (geographic unit such as FIPS code), **date** (date corresponding to forecast horizon; must be `date` class or character formatted as 'YYYY-MM-DD'), **horizon** (forecast horizon), **lower** (the lower limit of the prediction interval for the forecast), **point** (the point estimate for the forecast), and **upper** (the upper limit of the prediction interval for the forecast). Note that the read_forecast function returns data in this format.

The input data must at the daily, weekly, or monthly resolution. The "resolution" parameter is designed to use string matching. This allows flexibility for the user, such that, for example, an input of "day", "days", or "daily" would all resolve to a resolution of *days*. The same rules apply for designating weekly or monthly resolution.

Value

An object of the class `signal`. The object will have a second class of either `observed` or `forecast` depending on the value passed to the "type" argument.

Examples

```
hosp <- read.csv(system.file("extdata/observed/hdgov_hosp_weekly.csv", package = "rplanes"))
to_signal(hosp, outcome = "flu.admits", type = "observed", resolution = "weeks")

fp <- system.file("extdata/forecast/2022-10-31-SigSci-TSENS.csv", package = "rplanes")
ex_forecast <- read_forecast(fp)
to_signal(ex_forecast, outcome = "flu.admits", type = "forecast", horizon = 4, resolution = "weeks")
```

valid_dates *Validate dates*

Description

This function validates that there are no gaps or overlaps between dates specified in the "seed_date" and "signal_date". During plausibility component analyses, the function is called to validate the seed against the evaluated signal.

Usage

```
valid_dates(seed_date, signal_date, resolution, warn_incomplete = FALSE)
```

Arguments

seed_date	Last date available in seed object
signal_date	First date available in signal object
resolution	Character vector specifying the temporal resolution (e.g., "days", "weeks", "months")
warn_incomplete	Logical as to whether or not the validation should warn for completeness of seed and signal; default is FALSE

Value

The validation will return with a stop() if there is an overlap or gap between seed and signal dates. Otherwise the function will invisibly return TRUE indicating that the date span is valid.

Examples

```
seed_date <- as.Date("2023-03-08")
signal_date <- as.Date("2023-03-15")
valid_dates(seed_date = seed_date, signal_date = signal_date, resolution="weeks")
x <- try(valid_dates(seed_date = seed_date,
                         signal_date = signal_date,
                         resolution="days"), silent=TRUE)
x
x <- try(valid_dates(seed_date = seed_date,
                         signal_date = signal_date,
                         resolution="months"), silent=TRUE)
x
```

valid_location *Validate location*

Description

This unexported helper is used inside of the individual plausibility component functions (e.g., `plane_diff()`) to validate that the location specified appears in both the input signal and seed and that the location has as many values as other locations in the seed.

Usage

```
valid_location(location, input, seed)
```

Arguments

`location`	Character vector with location code; the location must appear in input and seed
`input`	Input signal data to be scored; object must be forecast
`seed`	Prepared seed

Value

The validation will return with a `stop()` if the location is not found in the seed or input signal. Otherwise the function will invisibly return `TRUE` indicating that the location is valid.

Index