Package ‘padr’

July 16, 2026

Type Package

Title Quickly Get Datetime Data Ready for Analysis

Version 0.7.0

Description Transforms datetime data into a format ready for analysis.
It offers two core functionalities; aggregating data to a higher level interval (thicken) and imputing records where observations were absent (pad).

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports Rcpp, dplyr (>= 1.0.0), lubridate, rlang

Suggests ggplot2, testthat, knitr, rmarkdown, lazyeval, tidyr, data.table

RoxygenNote 7.3.3

LinkingTo Rcpp

VignetteBuilder knitr

URL https://edwinth.github.io/padr/, https://github.com/EdwinTh/padr

BugReports https://github.com/EdwinTh/padr/issues

ByteCompile true

NeedsCompilation yes

Author Edwin Thoen [aut, cre]

Maintainer Edwin Thoen <edwinthoen@gmail.com>

Repository CRAN

Date/Publication 2026-07-16 14:30:02 UTC

Contents

center_interval 2
closest_weekday 3
coffee 4
emergency 4
fill_by_function 5
fill_by_prevalent 5
fill_by_value 6
format_interval 7
get_interval 8
pad 9
pad_cust 11
pad_int 12
span_around 13
span_date 14
span_time 15
subset_span 16
thicken 17
thicken_cust 19

Index **21**

`center_interval` *Shift to the middle of each interval*

Description

After thickening all the values are either shifted to the first or the last value of their interval. This function creates a vector from `x`, with the values shifted to the (approximate) center of the interval. This can give a more accurate picture of the aggregated data when plotting.

Usage

```
center_interval(x, shift = c("up", "down"), interval = NULL)
```

Arguments

`x`	A vector of class `Date`, `POSIXct` or `POSIXlt`.
`shift`	"up" or "down".
`interval`	The interval to be used for centering. If `NULL`, `get_interval` will be applied on `x`.

Details

The interval will be translated to number of days when `x` is of class `Date`, or number of seconds when `x` is of class `POSIXt`. For months and quarters this will be the average length of the interval. The translated units divided by two will be added by or subtracted from each value of `x`.

Value

Vector of the same class as x, with the values shifted to the (approximate) center.

Examples

```
plot_set <- emergency |>
  thicken("hour", "h") |>
  dplyr::count(h) |>
  head(24)

ggplot2::ggplot(plot_set, ggplot2::aes(h, n)) +
  ggplot2::geom_col()

plot_set |>
  dplyr::mutate(h_center = center_interval(h)) |>
  ggplot2::ggplot(ggplot2::aes(h_center, n)) +
  ggplot2::geom_col()
```

---

closest_weekday *Retrieve the closest given weekday*

Description

Find the closest instance of the requested weekday to `min(x)`. Helper function for `thicken` with the interval "week", when the user desires the start day of the weeks to be different from Sundays.

Usage

```
closest_weekday(x, wday = 1, direction = c("down", "up"))
```

Arguments

x	A vector of class `Date`, `POSIXct`, or `POSIXlt`.
wday	Integer in the range 0-6 specifying the desired weekday start (0 = Sun, 1 = Mon, 2 = Tue, 3 = Wed, 4 = Thu, 5 = Fri, 6 = Sat).
direction	The first desired weekday before ("down") or after ("up") the first day in x.

Value

Object of class `Date`, reflecting the closest desired weekday to x.

Examples

```
closest_weekday(coffee$time_stamp)
closest_weekday(coffee$time_stamp, 5)
closest_weekday(coffee$time_stamp, 1, direction = "up")
closest_weekday(coffee$time_stamp, 5, direction = "up")
```

coffee *Coffee Data Set*

Description

Made-up data set for demonstrating `padr`.

Usage

```
coffee
```

Format

A data frame with 4 rows and 2 variables:

time_stamp YYYY-MM-DD HH:MM:SS

amount Amount spent on coffee

emergency *Emergency Calls for Montgomery County, PA*

Description

The emergency calls coming in at Montgomery County, PA since 2015-12-10. Data set was created at 2016-10-17 16:15:40 CEST from the API and contains events until 2016-10-17 09:47:03 EST. From the original set the columns desc and e are not included.

Usage

```
emergency
```

Format

A data frame with 120450 rows and 6 variables:

lat Latitude from Google maps, based on the address

lng Longitude from Google maps, based on the address

zip Zipcode from Google, when possible

title Title, emergency category

time_stamp YYYY-MM-DD HH:MM:SS

twp Township

fill_by_function — *Fill missing values by a function of the nonmissings*

Description

For each specified column in x replace the missing values by a function of the nonmissing values.

Usage

```
fill_by_function(x, ..., fun = mean)
```

Arguments

x	A data frame.
...	The unquoted column names of the variables that should be filled.
fun	The function to apply on the nonmissing values.

Value

x with the altered columns.

Examples

```
x <- seq(as.Date("2016-01-01"), by = "day", length.out = 366)
x <- x[sample(1:366, 200)] |> sort()
x_df <- data.frame(
  x = x,
  y1 = runif(200, 10, 20) |> round(),
  y2 = runif(200, 1, 50) |> round()
)
x_df |>
  pad() |>
  fill_by_function(y1, y2)
x_df |>
  pad() |>
  fill_by_function(y1, y2, fun = median)
```

fill_by_prevalent — *Fill missing values by the most prevalent nonmissing value*

Description

For each specified column in x replace the missing values by the most prevalent nonmissing value.

Usage

```
fill_by_prevalent(x, ...)
```

Arguments

x	A data frame.
...	The unquoted column names of the variables that should be filled.

Value

x with the altered columns.

Examples

```
x <- seq(as.Date("2016-01-01"), by = "day", length.out = 366)
x <- x[sample(1:366, 200)] |> sort()
x_df <- data.frame(
  x = x,
  y1 = rep(letters[1:3], c(80, 70, 50)) |> sample(),
  y2 = rep(letters[2:5], c(60, 80, 40, 20)) |> sample()
)
x_df |>
  pad() |>
  fill_by_prevalent(y1, y2)
```

---

fill_by_value *Fill missing values by a single value*

Description

Replace all missing values in the specified columns by the same value.

Usage

```
fill_by_value(x, ..., value = 0)
```

Arguments

x	A data frame.
...	The unquoted column names of the variables that should be filled.
value	The value to replace the missing values by.

Value

x with the altered columns.

Examples

```
x <- seq(as.Date("2016-01-01"), by = "day", length.out = 366)
x <- x[sample(1:366, 200)] |> sort()
x_df <- data.frame(
  x = x,
  y1 = runif(200, 10, 20) |> round(),
  y2 = runif(200, 1, 50) |> round(),
  y3 = runif(200, 20, 40) |> round(),
  y4 = sample(letters[1:5], 200, replace = TRUE)
)
x_padded <- x_df |> pad()
x_padded |> fill_by_value(y1)
x_df |>
  pad() |>
  fill_by_value(y1, y2, value = 42)
```

---

format_interval *Make a period character vector*

Description

After applying `thicken` all the observations of a period are mapped to a single time point. This function will convert a datetime variable to a character vector that reflects the period, instead of a single time point. `strftime` is used to format the start and the end of the interval.

Usage

```
format_interval(
  x,
  start_format = "%Y-%m-%d",
  end_format = start_format,
  sep = " ",
  end_offset = 0,
  units_to_last = NULL
)
```

Arguments

`x`	A vector of class `Date`, `POSIXct` or `POSIXlt`, of which the values are unique.
`start_format`	String to format the start values of each period, to be used in `strftime`.
`end_format`	String to format the end values of each period, to be used in `strftime`.
`sep`	Character string that separates the `start_format` and the `end_format`.
`end_offset`	Units in days if `x` is `Date`, or in seconds if `x` is `POSIXct` or `POSIXlt`. Will be subtracted from the end of each period. If 0, the end of the previous period is equal to the start of the next.
`units_to_last`	To determine the formatting of the last value in `x`, the length of the last period has to be specified. If `NULL` the function guesses based on the interval of `x`. Specify in days when `x` is `Date`, or in seconds when `x` is `POSIXct` or `POSIXlt`.

Details

The end of the periods will be determined by the next unique value in `x`. It does so without regarding the interval of `x`. If a specific interval is desired, `thicken` and / or `pad` should first be applied to create an equally spaced datetime variable.

Value

A character vector showing the interval.

Examples

```
plot_set <- emergency |>
  head(500) |>
  thicken("hour", "h") |>
  dplyr::count(h)

# this will show the data on the full hour
ggplot2::ggplot(plot_set, ggplot2::aes(h, n)) +
  ggplot2::geom_col()

# adding a character to indicate the hours of the interval.
plot_set |>
  dplyr::mutate(h_int = format_interval(h, "%H", sep = "-"))
```

`get_interval` *Get the interval of a datetime variable*

Description

The interval is the highest datetime unit that can explain all instances of a variable of class `Date`, class `POSIXct`, or class `POSIXct`. This function will determine what the interval of the variable is.

Usage

```
get_interval(x)
```

Arguments

`x`	A variable of class of class `Date` or of class `POSIXt`.

Details

See `vignette("padr")` for more information on intervals.

Value

A character string indicating the interval of `x`.

Examples

```
x_month <- seq(as.Date("2016-01-01"), as.Date("2016-05-01"), by = "month")
get_interval(x_month)

x_sec <- seq(as.POSIXct("2016-01-01 00:00:00"), length.out = 100, by = "sec")
get_interval(x_sec)
get_interval(x_sec[seq(0, length(x_sec), by = 5)])
```

---

pad *Pad the datetime column of a data frame*

Description

pad will fill the gaps in incomplete datetime variables, by figuring out what the interval of the data is and what instances are missing. It will insert a record for each of the missing time points. For all other variables in the data frame a missing value will be inserted at the padded rows.

Usage

```
pad(
  x,
  interval = NULL,
  start_val = NULL,
  end_val = NULL,
  by = NULL,
  group = NULL,
  break_above = 1
)
```

Arguments

x	A data frame containing at least one variable of class Date, POSIXct or POSIXlt.
interval	The interval of the returned datetime variable. Any character string that would be accepted by seq.Date() or seq.POSIXt. The only exceptions is "DSTday", which is not accepted. pad will take care of daylight savings time when regular "day" is used. When NULL the the interval will be equal to the interval of the datetime variable. When specified it can only be lower than the interval and step size of the input data. See Details.
start_val	An object of class Date, POSIXct or POSIXlt that specifies the start of the returned datetime variable. If NULL it will use the lowest value of the input variable.
end_val	An object of class Date, POSIXct or POSIXlt that specifies the end of returned datetime variable. If NULL it will use the highest value of the input variable.
by	Optional character vector of length 1 specifying the column name of the variable to be used for padding. Only needs to be specified when x contains multiple variables of class Date, POSIXct or POSIXlt.

group — Optional character vector that specifies the grouping variable(s). Padding will take place within the different groups. When interval is not specified, it will be determined applying `get_interval` on the datetime variable as a whole, ignoring groups (see last example). When `x` is a grouped `tibble` this will be used for grouping.

break_above — Numeric value that indicates the number of rows in millions above which the function will break. Safety net for situations where the interval is different than expected and padding yields a very large dataframe, possibly overflowing memory.

Details

The interval of a datetime variable is the time unit at which the observations occur. The eight intervals in `padr` are from high to low `year`, `quarter`, `month`, `week`, `day`, `hour`, `min`, and `sec`. Since `padr` v.0.3.0 the interval is no longer limited to be of a single unit. (Intervals like 5 minutes, 6 hours, 10 days are possible). `pad` will figure out the interval of the input variable and the step size, and will fill the gaps for the instances that would be expected from the interval and step size, but are missing in the input data. Note that when `start_val` and/or `end_val` are specified, they are concatenated with the datetime variable before the interval is determined.

Rows with missing values in the datetime variables will be retained. However, they will be moved to the end of the returned data frame.

Value

The data frame `x` with the datetime variable padded. All non-grouping variables in the data frame will have missing values at the rows that are padded. The result will always be sorted on the datetime variable. If `group` is not `NULL` result is sorted on grouping variable(s) first, then on the datetime variable.

Examples

```
simple_df <- data.frame(
  day = as.Date(c("2016-04-01", "2016-04-03")),
  some_value = c(3, 4)
)
pad(simple_df)
pad(simple_df, interval = "day")

month <- seq(as.Date("2016-04-01"), as.Date("2017-04-01"),
  by = "month"
)[c(1, 4, 5, 7, 9, 10, 13)]
month_df <- data.frame(
  month = month,
  y = runif(length(month), 10, 20) |> round()
)
# forward fill the padded values with tidyr's fill
month_df |>
  pad() |>
  tidyr::fill(y)
```

```
# or fill all y with 0
month_df |>
  pad() |>
  fill_by_value(y)

# padding a data.frame on group level
day_var <- seq(as.Date("2016-01-01"), length.out = 12, by = "month")
x_df_grp <- data.frame(
  grp1 = rep(LETTERS[1:3], each = 4),
  grp2 = letters[1:2],
  y = runif(12, 10, 20) |> round(0),
  date = sample(day_var, 12, TRUE)
) |>
  dplyr::arrange(grp1, grp2, date)

# pad by one grouping var
x_df_grp |> pad(group = "grp1")

# alternatively you `dplyr::group_by` can be used
x_df_grp |> dplyr::group_by(grp1) |> pad()

# pad by two groups vars
x_df_grp |> pad(group = c("grp1", "grp2"), interval = "month")

# Using group argument the interval is determined over all the observations,
# ignoring the groups.
x <- data.frame(
  dt_var = as.Date(c(
    "2017-01-01", "2017-03-01", "2017-05-01",
    "2017-01-01", "2017-02-01", "2017-04-01"
  )),
  id = rep(1:2, each = 3), val = round(rnorm(6))
)
pad(x, group = "id")
# applying pad with do, interval is determined individual for each group
x |>
  dplyr::group_by(id) |>
  dplyr::do(pad(.))
```

`pad_cust` *Pad with a custom spanning*

Description

Pad the datetime variable after `thicken_cust` is applied, using the same spanning.

Usage

```
pad_cust(x, spanned, by = NULL, group = NULL, drop_last_spanned = TRUE)
```

Arguments

- `x` — A data frame containing at least one datetime variable of class `Date`, `POSIXct` or `POSIXlt`.
- `spanned` — A datetime vector to which the the datetime variable in `x` should be mapped. See `subset_span` for quickly spanning unequally spaced variables.
- `by` — Only needs to be specified when `x` contains multiple variables of class `Date`, `POSIXct` or `POSIXlt`.
- `group` — Optional character vector that specifies the grouping variable(s). Padding will take place within the different group values.
- `drop_last_spanned` — Logical, indicating whether to drop the last value from `spanned`. The spanned is typically around the datetime variable. This would create an empty last record when padding. Setting to `TRUE` will drop the last value in `spanned` and will not create an empty last record in this situation.

Value

The data frame `x` with the datetime column padded.

Examples

```
# analysis of traffic accidents in traffic jam hours and other hours.
accidents <- emergency |> dplyr::filter(title == "Traffic: VEHICLE ACCIDENT -")
spanning <- span_time("20151210 16", "20161017 17", tz = "EST") |>
  subset_span(list(hour = c(6, 9, 16, 19)))
thicken_cust(accidents, spanning, "period") |>
  dplyr::count(period) |>
  pad_cust(spanning)
```

`pad_int` *Pad the integer column of a data frame*

Description

`pad_int` fills the gaps in incomplete integer variables. It will insert a record for each of the missing value. For all other variables in the data frame a missing value will be inserted at the padded rows.

Usage

```
pad_int(x, by, start_val = NULL, end_val = NULL, group = NULL, step = 1)
```

Arguments

x	A data frame.
by	The column to be padded.
start_val	The first value of the returned variable. If NULL it will use the lowest value of the input variable.
end_val	The last value of the returned variable. If NULL it will use the highest value of the input variable.
group	Optional character vector that specifies the grouping variable(s). Padding will take place within the different group values.
step	The step size of the returned variable.

Value

The data frame x with the specified variable padded. All non-grouping variables in the data frame will have missing values at the rows that are padded.

Examples

```
int_df <- data.frame(
  x = c(2005, 2007, 2008, 2011),
  val = c(3, 2, 6, 3)
)
pad_int(int_df, "x")
pad_int(int_df, "x", start_val = 2006, end_val = 2013)

int_df2 <- data.frame(x = c(2005, 2015), val = c(3, 4))
pad_int(int_df2, "x", step = 2)
pad_int(int_df2, "x", step = 5)

int_df3 <- data.frame(
  x = c(2005, 2006, 2008, 2006, 2007, 2009),
  g = rep(LETTERS[1:2], each = 3),
  val = c(6, 6, 3, 5, 4, 3)
)
pad_int(int_df3, "x", group = "g")
pad_int(int_df3, "x", group = "g", start_val = 2005, end_val = 2009)
```

---

span_around *Span an equally spaced vector around a datetime variable*

Description

Span a vector of specified interval around a variable of class Date, POSIXct, or POSIXlt..

Usage

```
span_around(x, interval, start_shift = NULL, end_shift = start_shift)
```

Arguments

x	A vector of class Date, POSIXct, or POSIXlt.
interval	Character, specifying the desired interval.
start_shift	Character, indicating the time to shift back from the first observation.
end_shift	Character, indicating the time to shift forward from the last observation.

Details

Note that use of the start_shift and end_shift arguments change the entire spanning when they are not in line with the interval. It is not checked for.

Value

A datetime vector, with the first observation smaller or equal than min(x) and the last observation larger or equal than max(x). Spaces between points are equal to interval.

Examples

```
span_around(coffee$time_stamp, "hour")
span_around(coffee$time_stamp, "hour", end_shift = "2 hour")
span_around(coffee$time_stamp, "2 day")
span_around(coffee$time_stamp, "2 day", start_shift = "2 day")
span_around(emergency$time_stamp, "week")
span_around(emergency$time_stamp, "2 month")
```

---

span_date *Wrapper around* seq.Date.

Description

Quickly create a sequence of dates from minimal specifications.

Usage

```
span_date(from, to = NULL, len_out = NULL, by = NULL)
```

Arguments

from	Integer or character of length 4 (yyyy), 6 (yyyymm), or 8 (yyymmdd). Indicating the start value of the sequence.
to	Integer or character of length 4 (yyyy), 6 (yyyymm), or 8 (yyymmdd). Optional.
len_out	The desired length of the sequence. Optional.
by	The desired interval. Optional.

Details

Minimal specification of dates, sets unspecified date parts to default values. These are 01 for both month and day.

In addition to `from`, either `to` or `len_out` must be specified. If `by` is not specified, `span_date` will set the interval to the highest of the specified date parts in either `from` or `to`. For example, if they are 2011 and 2015 it will be "year", if they are 2011 and 201501 it will be "month".

Value

An object of class Date.

Examples

```
# using "to" argument
span_date(2011, 2015)
span_date(201101, 201501)
span_date(2011, 2015, by = "month")
span_date(2011, 201501)
span_date(20111225, 2012)

# using "len_out" argument
span_date(2011, len_out = 4)
span_date(201101, len_out = 4)
span_date(20110101, len_out = 4)
span_date(20110101, len_out = 4, by = "month")
```

---

span_time *Wrapper around* `seq.POSIXct`.

Description

Quickly create a sequence of datetimes from minimal specifications.

Usage

```
span_time(from, to = NULL, len_out = NULL, by = NULL, tz = "UTC")
```

Arguments

`from`	Integer or character of length 4 (yyyy), 6 (yyyymm), or 8 (yyymmdd). Character of length 11 (yyyymmdd hh), 13 (yyyymmdd hhmm), or 15 (yyyymmdd hhmmss). Indicating the start value of the sequence.
`to`	Integer or character of length 4 (yyyy), 6 (yyyymm), or 8 (yyymmdd). Character of length 11 (yyyymmdd hh), 13 (yyyymmdd hhmm), or 15 (yyyymmdd hhmmss). Indicating the end value of the sequence. Optional.
`len_out`	The desired length of the sequence. Optional.
`by`	The desired interval. Optional.
`tz`	The desired timezone.

Details

Minimal specification of datetimes, sets unspecified date parts to default values. These are 01 for both month and day and 00 for hour, minute, and second.

In addition to `from`, either `to` or `length` must be specified. If the `by` is not specified, `span_time` will set the interval to the highest of the specified datetime parts in either `from` or `to`. For example, if they are "20160103 01" and "20160108 05" it will be "hour", if they are "2011" and "20110101 021823" it will be "second".

Value

An object of class POSIXct.

Examples

```
# using to
span_time(2011, 2013)
span_time("2011", "2013")
span_time(2011, 201301)
span_time(2011, 20130101)
span_time(2011, "20110101 0023")
span_time(2011, "20110101 002300")

# using len_out
span_time(2011, len_out = 3)
span_time("2011", len_out = 3)
span_time(2011, len_out = 10, by = "month")
span_time(2011, len_out = 10, by = "day")
span_time(2011, len_out = 10, by = "hour")
span_time("20110101 00", len_out = 10)
span_time("20110101 002300", len_out = 10)
```

---

subset_span *Subset a spanned datetime vector*

Description

Take a `Date`, `POSIXct`, or `POSIXlt` vector and subset it by a pattern of date and/or time parts.

Usage

```
subset_span(spanned, pattern_list)
```

Arguments

`spanned`	A vector of class `Date`, `POSIXct`, or `POSIXlt`.
`pattern_list`	A list with the desired pattern for each of the following datetime parts: year, mon, mday, wday, hour, min, sec.

Details

For subsetting weekdays, they run from 0 (Sunday) to 6 (Saturday).

Value

Vector of the same class as `spanned`, containing all the data points in `spanned` that meets the requirements in `pattern_list`.

Examples

```
date_span <- span_date(20170701, len_out = 100)
subset_span(date_span, list(wday = 1:5))

time_span <- span_time("20170101 00", 201702)
subset_span(time_span, list(hour = 7:17))
subset_span(time_span, list(hour = c(10, 16), mday = seq(5, 30, 5)))
```

`thicken` *Add a variable of a higher interval to a data frame*

Description

Take the datetime variable in a data frame and map this to a variable of a higher interval. The mapping is added to the data frame in a new variable.

Usage

```
thicken(
  x,
  interval,
  colname = NULL,
  rounding = c("down", "up"),
  by = NULL,
  start_val = NULL,
  drop = FALSE,
  ties_to_earlier = FALSE
)
```

Arguments

`x`	A data frame containing at least one datetime variable of class `Date`, `POSIXct` or `POSIXlt`.
`interval`	The interval of the added datetime variable. Any character string that would be accepted by `seq.Date` or `seq.POSIXt`. It can only be higher than the interval and step size of the input data.
`colname`	The column name of the added variable. If `NULL` it will be the name of the original datetime variable with the interval name added to it (including the unit), separated by underscores.

rounding — Should a value in the input datetime variable be mapped to the closest value that is lower (down) or that is higher (up) than itself.

by — Only needs to be specified when x contains multiple variables of class Date, POSIXct or POSIXlt. Indicates which to use for thickening.

start_val — By default the first instance of interval that is lower than the lowest value of the input datetime variable, with all time units on default value. Specify start_val as an offset if you want the range to be nonstandard.

drop — Should the original datetime variable be dropped from the returned data frame? Defaults to FALSE.

ties_to_earlier — By default when the original datetime observations is tied with a value in the added datetime variable, it is assigned to the current value when rounding is down or to the next value when rounding is up. When TRUE the ties will be assigned to the previous observation of the new variable instead.

Details

When the datetime variable contains missing values, they are left in place in the dataframe. The added column with the new datetime variable, will have a missing values for these rows as well.

See vignette("padr") for more information on thicken. See vignette("padr_implementation") for detailed information on daylight savings time, different timezones, and the implementation of thicken.

Value

The data frame x with the variable added to it.

Examples

```
x_hour <- seq(lubridate::ymd_hms("20160302 000000"),
  by = "hour",
  length.out = 200
)
some_df <- data.frame(x_hour = x_hour)
thicken(some_df, "week")
thicken(some_df, "month")
thicken(some_df, "day", start_val = lubridate::ymd_hms("20160301 120000"))

x_df <- data.frame(
  x = seq(lubridate::ymd(20130101), by = "day", length.out = 1000) |>
    sample(500),
  y = runif(500, 10, 50) |> round()
) |>
  dplyr::arrange(x)

# get the max per month
x_df |>
  thicken("month") |>
  dplyr::group_by(x_month) |>
  dplyr::summarise(y_max = max(y))
```

```
# get the average per week, but you want your week to start on Mondays
# instead of Sundays
x_df |>
  thicken("week",
    start_val = closest_weekday(x_df$x, 2)
  ) |>
  dplyr::group_by(x_week) |>
  dplyr::summarise(y_avg = mean(y))

# rounding up instead of down
x <- data.frame(dt = lubridate::ymd_hms(
  "20171021 160000",
  "20171021 163100"
))
thicken(x, interval = "hour", rounding = "up")
thicken(x, interval = "hour", rounding = "up", ties_to_earlier = TRUE)
```

`thicken_cust` *Thicken with a custom spanning*

Description

Like `thicken`, it will find the datetime variable in `x` and add a variable of a higher periodicity to it. However, the variable to which to map the observation is provided by the user. This enables mapping to time points that are unequally spaced.

Usage

```
thicken_cust(x, spanned, colname, by = NULL, drop = FALSE)
```

Arguments

`x`	A data frame containing at least one datetime variable of class `Date`, `POSIXct` or `POSIXlt`.
`spanned`	A datetime vector to which the the datetime variable in `x` should be mapped.
`colname`	Character, the column name of the added variable.
`by`	Only needs to be specified when `x` contains multiple variables of class `Date`, `POSIXct` or `POSIXlt`. Indicates which to use for thickening.
`drop`	Should the original datetime variable be dropped from the returned data frame? Defaults to `FALSE`.

Details

Only rounding down is available for custom thickening.

Value

The data frame `x` with the variable added to it.

Examples

```
# analysis of traffic accidents in traffic jam hours and other hours.
accidents <- emergency |> dplyr::filter(title == "Traffic: VEHICLE ACCIDENT -")
spanning <- span_time("20151210 16", "20161017 17", tz = "EST") |>
  subset_span(list(hour = c(6, 9, 16, 19)))
thicken_cust(accidents, spanning, "period") |>
  dplyr::count(period) |>
  pad_cust(spanning)
```

Index

∗ **datasets**
    coffee, 4
    emergency, 4

center_interval, 2
closest_weekday, 3
coffee, 4

emergency, 4

fill_by_function, 5
fill_by_prevalent, 5
fill_by_value, 6
format_interval, 7

get_interval, 8

pad, 9
pad_cust, 11
pad_int, 12

span_around, 13
span_date, 14
span_time, 15
subset_span, 16

thicken, 17
thicken_cust, 19