

# Package ‘timefully’

May 8, 2026

**Type** Package

**Title** Time-Series Management Made Easy

**Version** 0.1.0

**Description** Manage time-series data frames across time zones, resolutions, and date ranges, while filling gaps using weekday/hour patterns or simple fill helpers or plotting them interactively. It is designed to work seamlessly with the tidyverse and dygraphs environments.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Suggests** testthat (>= 3.0.0), rmarkdown

**Config/testthat/edition** 3

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1.0)

**Imports** dplyr, lubridate, rlang, purrr, dygraphs

**URL** <https://github.com/resourcefully-dev/timefully/>,  
<https://resourcefully-dev.github.io/timefully/>

**BugReports** <https://github.com/resourcefully-dev/timefully/issues>

**Language** en-US

**NeedsCompilation** no

**Author** Marc Cañigüeral [aut, cre] (ORCID:  
<<https://orcid.org/0000-0001-9724-5829>>)

**Maintainer** Marc Cañigüeral <marccanyigüeral@gmail.com>

**Repository** CRAN

**Date/Publication** 2025-12-10 20:40:02 UTC

## Contents

adapt_timeseries . . . . .	2
add_extra_days . . . . .	3
aggregate_timeseries . . . . .	4
change_timeseries_resolution . . . . .	5
change_timeseries_tzone . . . . .	6
check_timeseries_gaps . . . . .	6
convert_time_num_to_period . . . . .	7
date_to_timestamp . . . . .	8
fill_datetime . . . . .	8
fill_down_until . . . . .	9
fill_from_past . . . . .	10
fill_na . . . . .	10
get_datetime_seq . . . . .	11
get_timeseries_resolution . . . . .	12
get_timeseries_tzone . . . . .	13
get_time_resolution . . . . .	13
get_week_from_datetime . . . . .	14
get_week_total . . . . .	15
get_yearly_datetime_seq . . . . .	15
plot_ts . . . . .	16
tic . . . . .	17
toc . . . . .	18
to_hhmm . . . . .	18
<b>Index</b>	<b>19</b>

---

adapt_timeseries	<i>Adapt time-series dataframe to timezone, date range and fill gaps</i>
------------------	--

---

### Description

This function adapts the date range of a time series by reusing historical patterns based on the same weekday occurrence within the year and decimal hour of the day. It also can fill gaps in the data based on past data, so it is recommended to use it for time series with weekly or yearly patterns (so for example energy demand but not solar generation). It can also adapt the timezone of the time series, for example if the data was stored in UTC but corresponds to a different timezone.

### Usage

```
adapt_timeseries(dtf, start_date, end_date, tzone = NULL, fill_gaps = FALSE)
```

**Arguments**

dtf	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
start_date	Date, start date of the output datetime sequence
end_date	Date, end date of the output datetime sequence (included)
tzone	character, desired time-zone of the datetime sequence. If NULL, the timezone of dtf\$datetime is kept.
fill_gaps	boolean, whether to fill gaps based on same weekday and hour from past data (See fill_from_past function).

**Value**

tibble

**Examples**

```
# Example data set
print(dtf)

# Original date range
range(dtf$datetime)

dtf2 <- adapt_timeseries(
  dtf,
  start_date = as.Date("2021-01-01"),
  end_date = as.Date("2021-01-31"),
  tzone = "America/New_York",
  fill_gaps = FALSE
)

# New date range
range(dtf2$datetime)
```

---

add_extra_days	<i>Add an extra day at the beginning and the end of datetime sequence using the last and first day of the data</i>
----------------	--

---

**Description**

Add an extra day at the beginning and the end of datetime sequence using the last and first day of the data

**Usage**

```
add_extra_days(dtf)
```

**Arguments**

dtf                    data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric

**Value**

tibble

---

aggregate\_timeseries    *Aggregate multiple timeseries columns to a single one*

---

**Description**

The first column datetime will be kept.

**Usage**

```
aggregate_timeseries(dtf, varname, omit = NULL)
```

**Arguments**

dtf                    data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric

varname                character, name of the aggregation column

omit                    character, name of columns to not aggregate

**Value**

tibble

**Examples**

```
building_flows <- data.frame(  
  datetime = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:3 * 3600,  
  building1 = c(2.1, 2.5, 2.3, 2.0),  
  building2 = c(1.0, 1.1, 0.9, 1.2)  
)  
aggregate_timeseries(building_flows, varname = "total_building")
```

---

`change_timeseries_resolution`*Change time resolution of a time-series data frame*

---

**Description**

Change time resolution of a time-series data frame

**Usage**

```
change_timeseries_resolution(dtf, resolution, method)
```

**Arguments**

<code>dtf</code>	data.frame or tibble, first column of name <code>datetime</code> being of class <code>datetime</code> and rest of columns being numeric
<code>resolution</code>	integer, desired interval of minutes between two consecutive datetime values
<code>method</code>	character, being <code>interpolate</code> , <code>repeat</code> or <code>divide</code> if the resolution has to be increased, or <code>average</code> , <code>first</code> or <code>sum</code> if the resolution has to be decreased. See Examples for more information.

**Value**

tibble

**Examples**

```
fifteen_min <- data.frame(  
  datetime = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:7 * 900,  
  load = c(10, 12, 14, 16, 14, 12, 10, 8)  
)  
change_timeseries_resolution(  
  fifteen_min,  
  resolution = 60,  
  method = "average"  
)
```

change\_timeseries\_tzone

*Adapt the timezone of a time series dataframe*

---

### Description

The timezone of the `datetime` column is changed while keeping the same date time sequence. This is useful when the time series data is known to be in a different timezone. If you just want the same time series in a different timezone, use `lubridate::force_tz` function instead.

### Usage

```
change_timeseries_tzone(dtf, tzone = "Europe/Amsterdam")
```

### Arguments

<code>dtf</code>	data.frame or tibble, first column of name <code>datetime</code> being of class <code>datetime</code> and rest of columns being numeric
<code>tzone</code>	character, desired time-zone of the <code>datetime</code> sequence

### Value

tibble

### Examples

```
# Example data set
get_timeseries_tzone(dtf)
range(dtf$datetime)

# Change timezone
new_dtf <- change_timeseries_tzone(dtf, tzone = "Europe/Paris")
get_timeseries_tzone(new_dtf)
range(new_dtf$datetime)
```

---

check\_timeseries\_gaps *Check if there are any gaps in the datetime sequence*

---

### Description

This means all rows a part from "datetime" will be NA. Note that `timefully` considers a full datetime sequence when days are complete.

### Usage

```
check_timeseries_gaps(dtf)
```

**Arguments**

`dtf` data.frame or tibble, first column of name `datetime` being of class `datetime` and rest of columns being numeric

**Value**

tibble

**Examples**

```
# Sample just some hours
dtf_gaps <- dtf[c(1:3, 7:10), ]

# Note that the full day is provided
check_timeseries_gaps(
  dtf_gaps
)
```

---

`convert_time_num_to_period`

*Convert numeric time value to a datetime period (hour-based)*

---

**Description**

Convert numeric time value to a datetime period (hour-based)

**Usage**

```
convert_time_num_to_period(time_num)
```

**Arguments**

`time_num` Numeric time value (hour-based)

**Value**

`lubridate::period` vector with hours and minutes corresponding to the numeric input.

**Examples**

```
convert_time_num_to_period(1.5)
convert_time_num_to_period(c(0.25, 2))
```

---

date\_to\_timestamp      *Convert date or datetime value to timestamp number*

---

### Description

Convert date or datetime value to timestamp number

### Usage

```
date_to_timestamp(date, tzzone = "Europe/Paris", milliseconds = TRUE)
```

### Arguments

date	date or datetime value
tzzone	character, time-zone of the current time
milliseconds	logical, whether the timestamp is in milliseconds or seconds

### Value

numeric

### Examples

```
date_to_timestamp(as.Date("2024-01-01"))
date_to_timestamp(as.POSIXct("2024-01-01 08:00:00", tz = "UTC"), milliseconds = FALSE)
```

---

fill\_datetime      *Fill NA values of a datetime sequence vector*

---

### Description

Fill NA values of a datetime sequence vector

### Usage

```
fill_datetime(dttm)
```

### Arguments

dttm	datetime sequence vector
------	--------------------------

### Value

filled datetime sequence vector

**Examples**

```
incomplete_seq <- as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:4 * 3600
incomplete_seq[c(2, 3)] <- NA
fill_datetime(incomplete_seq)
```

---

fill_down_until	<i>Fill down tibble columns until a maximum number of time slots</i>
-----------------	--

---

**Description**

Fill down tibble columns until a maximum number of time slots

**Usage**

```
fill_down_until(dtf, varnames, max_timeslots = 1)
```

**Arguments**

dtf	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
varnames	character or vector of characters, column names with NA values
max_timeslots	integer, maximum number of time slots to fill

**Value**

tibble

**Examples**

```
down_data <- data.frame(
  datetime = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:5 * 3600,
  temperature = c(15, 15, NA, NA, NA, 16)
)
fill_down_until(down_data, "temperature", max_timeslots = 2)
```

---

fill_from_past	<i>Fill from past values</i>
----------------	------------------------------

---

**Description**

If back index ( NA index - back) is lower than zero then the it is filled with the first value of the data frame. If the value in the back index is also NA, it iterates backwards until finding a non-NA value.

**Usage**

```
fill_from_past(dtf, varnames, back = 24)
```

**Arguments**

dtf	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
varnames	character or vector of characters, column names with NA values
back	integer, number of indices (rows) to go back and get the filling value

**Value**

tibble or data.frame

**Examples**

```
past_data <- data.frame(  
  datetime = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:3 * 3600,  
  consumption = c(1.2, NA, NA, 2.5)  
)  
fill_from_past(past_data, "consumption", back = 1)
```

---

fill_na	<i>Fill gaps with a specific value</i>
---------	--

---

**Description**

This is useful when the gaps in a numeric timeseries can be filled with the same number (e.g. zero)

**Usage**

```
fill_na(dtf, varnames, with = 0)
```

**Arguments**

dtf	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
varnames	character or vector of characters, column names with NA values
with	numeric, value to fill NA values

**Value**

tibble or data.frame

**Examples**

```
past_data <- data.frame(  
  datetime = as.POSIXct("2024-01-01 00:00:00", tz = "UTC") + 0:3 * 3600,  
  consumption = c(1.2, NA, NA, 2.5)  
)  
fill_na(past_data, "consumption", with = 0)
```

---

get\_datetime\_seq      *Date time sequence with time zone and resolution*

---

**Description**

Date time sequence with time zone and resolution

**Usage**

```
get_datetime_seq(start_date, end_date, tzone, resolution)
```

**Arguments**

start_date	Date, start date of the output datetime sequence
end_date	Date, end date of the output datetime sequence (included)
tzone	character, desired time-zone of the datetime sequence
resolution	integer, interval of minutes between two consecutive datetime values

**Value**

vector of datetime values

**Examples**

```
get_datetime_seq(  
  start_date = as.Date("2024-01-01"),  
  end_date = as.Date("2024-01-03"),  
  tzone = "UTC",  
  resolution = 120  
)
```

---

```
get_timeseries_resolution
```

*Return the time resolution of a time series dataframe*

---

**Description**

Return the time resolution of a time series dataframe

**Usage**

```
get_timeseries_resolution(dtf, units = "mins")
```

**Arguments**

dtf	data.frame or tibble, first column of name <code>datetime</code> being of class <code>datetime</code> and rest of columns being numeric
units	character being one of "auto", "secs", "mins", "hours", "days" and "weeks"

**Value**

numeric

**Examples**

```
get_timeseries_resolution(dtf, units = "mins")
```

---

get\_timeseries\_tzone *Get the time zone of a time series dataframe*

---

**Description**

Get the time zone of a time series dataframe

**Usage**

```
get_timeseries_tzone(dtf)
```

**Arguments**

dtf                    data.frame or tibble, first column of name `datetime` being of class `datetime` and rest of columns being numeric

**Value**

character

**Examples**

```
get_timeseries_tzone(dtf)
```

---

get\_time\_resolution *Return the time resolution of a datetime sequence*

---

**Description**

Return the time resolution of a datetime sequence

**Usage**

```
get_time_resolution(dttm_seq, units = "mins")
```

**Arguments**

dttm\_seq            datetime sequence  
units                character being one of "auto", "secs", "mins", "hours", "days" and "weeks"

**Value**

numeric

**Examples**

```
seq_15m <- as.POSIXct(  
  c("2024-01-01 00:00:00", "2024-01-01 00:15:00", "2024-01-01 00:30:00"),  
  tz = "UTC"  
)  
get_time_resolution(seq_15m, units = "mins")
```

---

get\_week\_from\_datetime

*Week date from datetime value*

---

**Description**

Week date from datetime value

**Usage**

```
get_week_from_datetime(dttm)
```

**Arguments**

dttm            datetime vector

**Value**

date vector

**Examples**

```
dttm <- as.POSIXct(  
  c("2024-01-01 08:00:00", "2024-01-02 09:00:00", "2024-01-08 10:00:00"),  
  tz = "UTC"  
)  
get_week_from_datetime(dttm)
```

---

get_week_total	<i>Summarise dataframe with weekly total column values</i>
----------------	--

---

**Description**

Converts the numeric columns of a time-series data frame to total values per week (sum). Note that if the input values are in power units (e.g., kW), the output values will be in energy units (e.g., kWh).

**Usage**

```
get_week_total(dtf)
```

**Arguments**

dtf	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
-----	---

**Value**

tibble

**Examples**

```
get_week_total(dtf[1:100, ])
```

---

get_yearly_datetime_seq	<i>Yearly date time sequence with time zone and resolution</i>
-------------------------	--

---

**Description**

Yearly date time sequence with time zone and resolution

**Usage**

```
get_yearly_datetime_seq(year, tzone, resolution)
```

**Arguments**

year	integer, year of the datetime sequence
tzone	character, desired time-zone of the datetime sequence
resolution	integer, interval of minutes between two consecutive datetime values

**Value**

vector of datetime values

**Examples**

```
get_yearly_datetime_seq(  
  year = 2024,  
  tzone = "UTC",  
  resolution = 60  
)
```

---

plot\_ts

*Interactive plot for time-series tibbles*

---

**Description**

First column of the df tibble must be a datetime or date variable. The rest of columns must be numeric of the same units. This functions makes use of dygraphs package to generate an HTML dygraphs plot.

**Usage**

```
plot_ts(  
  df,  
  title = NULL,  
  xlab = NULL,  
  ylab = NULL,  
  legend_show = "auto",  
  legend_width = 250,  
  group = NULL,  
  width = NULL,  
  height = NULL,  
  ...  
)
```

**Arguments**

df	data.frame or tibble, first column of name datetime being of class datetime and rest of columns being numeric
title	character, title of the plot (accepts HTML code)
xlab	character, X axis label (accepts HTML code)
ylab	character, Y axis label (accepts HTML code)

legend_show	character, when to display the legend. Specify "always" to always show the legend. Specify "onmouseover" to only display it when a user mouses over the chart. Specify "follow" to have the legend show as overlay to the chart which follows the mouse. The default behavior is "auto", which results in "always" when more than one series is plotted and "onmouseover" when only a single series is plotted.
legend_width	integer, width (in pixels) of the div which shows the legend.
group	character, dygraphs group to associate this plot with. The x-axis zoom level of dygraphs plots within a group is automatically synchronized.
width	Width in pixels (optional, defaults to automatic sizing)
height	Height in pixels (optional, defaults to automatic sizing)
...	extra arguments to pass to dygraphs::dyOptions function.

**Value**

dygraph

**Examples**

```
plot_ts(dtf, ylab = "kW", legend_show = "onmouseover")
```

---

 tic

*Time difference start function*

---

**Description**

Use this function together with toc() to control time spent by functions

**Usage**

```
tic()
```

**Value**

numeric

---

toc	<i>Time difference end function</i>
-----	-------------------------------------

---

**Description**

Use this function together with `tic()` to control time spent by functions

**Usage**

```
toc(units = "secs", digits = 2)
```

**Arguments**

units	character, one of "auto", "secs", "mins", "hours", "days" and "weeks"
digits	integer, number of decimals

**Value**

numeric

---

to_hhmm	<i>Convert a number of minutes in string format "HH:MM"</i>
---------	---

---

**Description**

Convert a number of minutes in string format "HH:MM"

**Usage**

```
to_hhmm(mins)
```

**Arguments**

mins	integer, number of minutes (from 0 to 1439)
------	---

**Value**

character

**Examples**

```
to_hhmm(75)
```

# Index

[adapt\\_timeseries](#), 2  
[add\\_extra\\_days](#), 3  
[aggregate\\_timeseries](#), 4  
  
[change\\_timeseries\\_resolution](#), 5  
[change\\_timeseries\\_tzone](#), 6  
[check\\_timeseries\\_gaps](#), 6  
[convert\\_time\\_num\\_to\\_period](#), 7  
  
[date\\_to\\_timestamp](#), 8  
  
[fill\\_datetime](#), 8  
[fill\\_down\\_until](#), 9  
[fill\\_from\\_past](#), 10  
[fill\\_na](#), 10  
  
[get\\_datetime\\_seq](#), 11  
[get\\_time\\_resolution](#), 13  
[get\\_timeseries\\_resolution](#), 12  
[get\\_timeseries\\_tzone](#), 13  
[get\\_week\\_from\\_datetime](#), 14  
[get\\_week\\_total](#), 15  
[get\\_yearly\\_datetime\\_seq](#), 15  
  
[plot\\_ts](#), 16  
  
[tic](#), 17  
[to\\_hhmm](#), 18  
[toc](#), 18