Package ‘sftime’

October 14, 2022

Title Classes and Methods for Simple Feature Objects that Have a Time Column

Description Classes and methods for spatial objects that have a registered time column, in particular for irregular spatiotemporal data. The time column can be of any type, but needs to be ordinal. Regularly laid out spatiotemporal data (vector or raster data cubes) are handled by package ‘stars’.

Version 0.2-0

Depends sf (>= 1.0.7)

Imports methods

Suggests knitr, spacetime, rmarkdown, dplyr (>= 0.8-3), trajectories (>= 0.2.2), stars, ncmeta, tidyrr, ggplot2, magrittr, sp, rlang

License Apache License

Type Package

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.1.2


NeedsCompilation no

Author Henning Teickner [aut, cre, cph]
   (<https://orcid.org/0000-0002-3993-1182>),
   Edzer Pebesma [aut, cph] (<https://orcid.org/0000-0001-8049-7069>),
   Benedikt Graeler [aut, cph] (<https://orcid.org/0000-0001-5443-4304>)

Maintainer Henning Teickner <henning.teickner@uni-muenster.de>

Repository CRAN

Date/Publication 2022-03-17 08:50:01 UTC

R topics documented:

bind .................................................. 2
bind

Description

Bind rows (features) of sftime objects

Bind columns (variables) of sftime objects

Usage

```r
## S3 method for class 'sftime'
rbind(..., deparse.level = 1)

## S3 method for class 'sftime'
cbind(..., deparse.level = 1, sf_column_name = NULL, tc_column_name = NULL)
```

Arguments

- `...`: Objects to bind; note that for the `rbind` and `cbind` methods, all objects have to be of class `sftime`; see `dotsMethods`.
- `deparse.level`: An integer value; see `rbind`.
- `sf_column_name`: Character value; specifies the active geometry column; passed on to `st_sftime`.
- `tc_column_name`: Character value; specifies active time column; passed on to `st_sftime`.

Details

Both `rbind` and `cbind` have non-standard method dispatch (see `cbind`): the `rbind` or `cbind` method for `sftime` objects is only called when all arguments to be combined are of class `sftime`.

If you need to `cbind` e.g. a `data.frame` to an `sf`, use `data.frame` directly and use `st_sftime` on its result, or use `bind_cols`; see examples.
Value

`rbind` combines all `sftime` objects in ... row-wise and returns the combined `sftime` object.

`cbind` combines all `sftime` objects in ... column-wise and returns the combined `sftime` object. When called with multiple `sftime` objects warns about multiple time and geometry columns present when the time and geometry columns to use are not specified by using arguments `tc_column_name` and `sf_column_name`; see also `st_sftime`.

Examples

```r
# Simple examples

g1 <- st_sfc(st_point(1:2))
x1 <- st_sftime(a = 3, geometry = g1, time = Sys.time())

g2 <- st_sfc(st_point(c(4, 6)))
x2 <- st_sftime(a = 4, geometry = g2, time = Sys.time())

rbind(x1, x2) # works because both tc1 and tc2 have the same class

## Not run:
st_time(x2) <- 1
rbind(x1, x2) # error because both tc1 and tc2 do not have the same class

## End(Not run)

cbind(x1, x2)

if (require(dplyr))
  dplyr::bind_cols(x1, x2)

df <- data.frame(x = 3)
st_sftime(data.frame(x1, df))
```

Description

Geometric operations on pairs of simple feature geometry sets (including `sftime` objects)

Usage

```r
# S3 method for class 'sftime'
st_intersection(x, y, ...)
```
## S3 method for class 'sftime'
st_difference(x, y, ...)

## S3 method for class 'sftime'
st_sym_difference(x, y, ...)

**Arguments**

- **x**: object of class `sftime`, `sf`, `sfc` or `sfg`.
- **y**: object of class `sftime`, `sf`, `sfc` or `sfg`.
- **...**: See `geos_binary_ops`.

**Details**

**st_intersection**: When called with a missing `y`, the `sftime` method for `st_intersection` returns an `sftime` object with attributes taken from the contributing feature with lowest index; two fields are added:

- `n.overlaps`: The number of overlapping features in `x`.
- `origins`: A list-column with indexes of all overlapping features.

**st_difference**: When `st_difference` is called with a single argument, overlapping areas are erased from geometries that are indexed at greater numbers in the argument to `x`; geometries that are empty or contained fully inside geometries with higher priority are removed entirely.

**Value**

The intersection, difference or symmetric difference between two sets of geometries. The returned object has the same class as that of the first argument (`x`) with the non-empty geometries resulting from applying the operation to all geometry pairs in `x` and `y`. In case `x` is of class `sf` or `sftime`, the matching attributes of the original object(s) are added. The `sfc` geometry list-column returned carries an attribute `idx`, which is an `n`-by-2 matrix with every row the index of the corresponding entries of `x` and `y`, respectively.

**Examples**

```r
g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)), st_point(c(2, 1)), st_point(c(3, 1)))
tc <- Sys.time() + 1:5
x1 <- st_sftime(a = 1:5, g, time = tc)
x2 <- st_buffer(x1, dist = 1)

## intersection

# only x provided (no y)
plot(st_intersection(x2))

# with arguments x and y provided
plot(st_intersection(x2, x1))
```
geos_combine

## difference

# only x provided (no y)
plot(st_difference(x2))

# with arguments x and y provided
plot(st_difference(x2, x1))

## symmetric difference
plot(st_sym_difference(x1, x2))

describe_geos_combine

Combine or union feature geometries (including sftime objects)

Description

Combine or union feature geometries (including sftime objects)

Usage

## S3 method for class 'sftime'
st_union(x, y, ..., by_feature = FALSE, is_coverage = FALSE)

Arguments

x An object of class sftime, sf, sfc or sfg.
y An object of class sftime, sf, sfc or sfg (optional).
... See geos_combine.
by_feature See geos_combine.
is_coverage See geos_combine.

Details

See geos_combine.

Value

If y is missing, st_union(x) returns a single geometry with resolved boundaries, else the geometries for all unioned pairs of x[i] and y[j].
Examples

# union simple features in an sftime object

```r
library(sf)
library(tidyverse)

# create a MultiPoint object and add some time

g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)), st_point(c(2, 1)), st_point(c(3, 1)))
tc <- Sys.time() + 1:5
x <- st_sftime(a = 1:5, g, time = tc)

# only x provided (no y)
plot(st_union(st_buffer(x, dist = 1)))

# with arguments x and y provided
plot(st_union(st_buffer(x, dist = 1), st_buffer(x, dist = 0.5)), "a")
```

---

### plot.sftime

**Plots an sftime object**

#### Description

**plot.sftime**

#### Usage

```r
## S3 method for class 'sftime'
plot(x, y, ..., number = 6, tcuts)
```

#### Arguments

- `x`  
The *sftime* object to be plotted.
- `y`  
A character value; The variable name to be plotted; if missing, the first variable is plotted.
- `...`  
Additional arguments; Passed on to `plot.sf`.
- `number`  
A numeric value; The number of panels to be plotted, cannot be larger than the number of timestamps; ignored when `tcuts` is provided.
- `tcuts`  
predefined temporal ranges assigned to each map; if missing, will be determined as equal spans according to `number`.

#### Value

Returns NULL and creates as side effect a plot for `x`. 
print.sftime

Examples

```r
set.seed(123)
coords <- matrix(runif(100), ncol = 2)
g <- st_sfc(lapply(1:50, function(i) st_point(coords[i, ])))
sft <- st_sftime(a = 1:50, g, time = as.POSIXct("2020-09-01 00:00:00") + 0:49 * 3600 * 6)
plot(sft)
```

---

print.sftime  

**Prints an sftime object**

Description

Prints an sftime object

Usage

```r
## S3 method for class 'sftime'
print(x, ..., n = getOption("sf_max_print", default = 10))
```

Arguments

- `x`  
  An object of class sftime.

- `...`  
  Currently unused arguments, for compatibility.

- `n`  
  Numeric value; maximum number of printed elements.

Value

- `x` (invisible).

Examples

```r
g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)),
            st_point(c(2, 1)), st_point(c(3, 1)))
tc <- Sys.time() + 1:5
x <- st_sftime(a = 1:5, g, time = tc)
print(x)
print(x[0, 1])
```
st_as_sftime  

Convert a foreign object to an sftime object

Description

Convert a foreign object to an sftime object

Usage

```
st_as_sftime(x, ...)

## S3 method for class 'ST'
st_as_sftime(x, ...)

## S3 method for class 'Track'
st_as_sftime(x, ...)

## S3 method for class 'Tracks'
st_as_sftime(x, ...)

## S3 method for class 'TracksCollection'
st_as_sftime(x, ...)

## S3 method for class 'sftime'
st_as_sftime(x, ...)

## S3 method for class 'sf'
st_as_sftime(x, ..., time_column_name = NULL)

## S3 method for class 'stars'
st_as_sftime(x, ..., long = TRUE, time_column_name = NULL)

## S3 method for class 'data.frame'
st_as_sftime(
  x,
  ..., 
  agr = NA_agr_,
  coords,
  wkt,
  dim = "XYZ",
  remove = TRUE,
  na.fail = TRUE,
  sf_column_name = NULL,
  time_column_name = NULL,
  time_column_last = FALSE
)
```
Arguments

x  An object to be converted into an object of class \texttt{sftime}.

... Further arguments passed to methods.

time_column_name  A character value; name of the active time column. In case there is more than one and 
time_column_name is \texttt{NULL}, the first one is taken.

long  A logical value; See \texttt{st_as_sf}. Typically, long should be set to \texttt{TRUE} since time 
information typically is a dimension of a \texttt{stars} object.

agr  A character vector; see details section of \texttt{st_sf}.

coords In case of point data: names or numbers of the numeric columns holding coor-
dinates.

wkt The name or number of the character column that holds WKT encoded geome-
tries.

dim Passed on to \texttt{st_point} (only when argument coords is given).

remove A logical value; when coords or wkt is given, remove these columns from code-
data.frame?

na.fail A logical value; if \texttt{TRUE}, raise an error if coordinates contain missing values.

sf_column_name  A character value; name of the active list-column with simple feature geome-
tries; in case there is more than one and \texttt{sf_column_name} is \texttt{NULL}, the first one 
is taken.

time_column_last  A logical value; if \texttt{TRUE}, the active time column is always put last, otherwise 
column order is left unmodified. If both sfc_last and time_column_last are 
\texttt{TRUE}, the active time column is put last.

Value

x converted to an \texttt{sftime} object.

\texttt{st_as_sftime.Tracks} furthermore adds a column track_name with the names of the tracks slot 
of the input \texttt{Tracks} object.

\texttt{st_as_sftime.TracksCollection} furthermore adds the columns tracks_name with the names of 
the \texttt{tracksCollection} slot and track_name with the names of the tracks slot of the input \texttt{Tracks} 
object.

Examples

# modified from spacetime:
library(sp)
library(spacetime)

sp <- cbind(x = c(0,0,1), y = c(0,1,1))
row.names(sp) <- paste("point", 1:nrow(sp), sep="")
sp <- SpatialPoints(sp)
time <- as.POSIXct("2010-08-05") + 3600 * (10:12)
x <- STI(sp, time)
st_as_sftime(x)

# convert a Track object from package trajectories to an sftime object
library(trajectories)
x1_Track <- trajectories::rTrack(n = 100)
x1_Track@data$speed <- sort(rnorm(length(x1_Track)))
x1_sftime <- st_as_sftime(x1_Track)

# convert a Tracks object from package trajectories to an sftime object
x2_Tracks <- trajectories::rTracks(m = 6)
x2_sftime <- st_as_sftime(x2_Tracks)

# convert a TracksCollection object from package trajectories to an sftime object
x3_TracksCollection <- trajectories::rTracksCollection(p = 2, m = 3, n = 50)
x3_sftime <- st_as_sftime(x3_TracksCollection)

# convert an sftime object to an sftime object
st_as_sftime(x3_sftime)

# convert an sf object to an sftime object
g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)),
            st_point(c(2, 1)), st_point(c(3, 1)))
x4_sf <- st_sf(a = 1:5, g, time = Sys.time() + 1:5)
x4_sftime <- st_as_sftime(x4_sf)

# convert a Tracks object from package trajectories to an sftime object
x5_stars <- stars::read_stars(system.file("nc/bcsd_obs_1999.nc", package = "stars"))
x5_sftime <- st_as_sftime(x5_stars, time_column_name = "time")

# this requires some thought to not accidentally drop time dimensions. For
# example, setting `merge = TRUE` will drop the time dimension and thus throw
# an error:
## Not run:
x5_sftime <- st_as_sftime(x5_stars, merge = TRUE, time_column_name = "time")
## End(Not run)

# convert a data frame to an sftime object
x5_df <-
data.frame(a = 1:5, g, time = Sys.time() + 1:5, stringsAsFactors = FALSE)
x5_sftime <- st_as_sftime(x5_df)

---

st_cast  

**Cast geometry to another type: either simplify, or cast explicitly**

**Description**

Cast geometry to another type: either simplify, or cast explicitly
**Usage**

```r
## S3 method for class 'sftime'
st_cast(x, to, ..., warn = TRUE, do_split = TRUE)
```

**Arguments**

- `x`: An object of class `sftime`.
- `to`: character; target type, if missing, simplification is tried; when `x` is of type `sfg` (i.e., a single geometry) then `to` needs to be specified.
- `...`: ignored
- `warn`: logical; if TRUE, warn if attributes are assigned to sub-geometries
- `do_split`: logical; if TRUE, allow splitting of geometries in sub-geometries

**Value**

`x` with changed geometry type.

**Examples**

```r
# cast from POINT to LINestring

g <- st_sfc(st_point(1:2), st_point(c(2, 4)))
time <- Sys.time()
x <- st_sftime(a = 3:4, g, time = time) %>%
dplyr::group_by(time) %>%
dplyr::summarize(do_union = TRUE) %>%
st_cast(to = "LINestring")
```

---

**st_crop.sftime**  
*Crop an sftime object to a specific rectangle*

**Description**

Crop an `sftime` object to a specific rectangle

**Usage**

```r
## S3 method for class 'sftime'
st_crop(x, y, ...)
```

**Arguments**

- `x`: An object of class `sftime`.
- `y`: A numeric vector with named elements `xmin`, `ymin`, `xmax` and `ymax`, or an object of class `bbox`, or an object for which there is an `st_bbox` method to convert it to a `bbox` object.
- `...`: Additional arguments; Ignored.
Details

See `st_crop`.

Value

x cropped using y.

Examples

```r
# modified from sf:
box <- c(xmin = 0, ymin = 0, xmax = 1, ymax = 1)
pol <- sf::st_sfc(sf::st_buffer(sf::st_point(c(0.5, 0.5)), 0.6))
pol_sftime <- st_sftime(a = 1, geom = pol, time = Sys.time() + 1:2 * 1000)
pol_sftime_cropped <- sf::st_crop(pol_sftime, sf::st_bbox(box))
class(pol_sftime_cropped)
plot(pol_sftime_cropped)
```

**st_geometry**  
*Drops the geometry column of sftime objects*

Description

Drops the geometry column of an `sftime` object. This will also drop the `sftime` class attribute and `time_column` attribute.

Usage

```r
## S3 method for class 'sftime'
st_drop_geometry(x, ...)
```

Arguments

- **x**  
  An `sftime` object.
- **...**  
  Ignored

Value

x without geometry column and without `sftime` and `sf` class.

Examples

```r
# dropping the geometry column will also drop the `sftime` class:
g <- st_sfc(st_point(1:2))
time <- Sys.time()
x <- st_sftime(a = 3, g, time = time)
st_drop_geometry(x)
```
st_join  

**Spatial join, spatial filter for sftime objects**

**Description**

Spatial join, spatial filter for sftime objects

**Usage**

```r
## S3 method for class 'sftime'
st_join(
  x,
  y,
  join = st_intersects,
  ...,  
  suffix = c(".x", ".y"),
  left = TRUE,
  largest = FALSE
)

## S3 method for class 'sftime'
st_filter(x, y, ..., .predicate = st_intersects)
```

**Arguments**

- `x`  
  An object of class sftime or sf.
- `y`  
  An object of class sftime or sf.
- `join`  
  A geometry predicate function with the same profile as `st_intersects`; see details.
- `...`  
  for `st_join`: arguments passed on to the `join` function or `st_intersection` when `largest` is `TRUE`; for `st_filter` arguments passed on to the `.predicate` function, e.g. `prepared`, or a pattern for `st_relate`.
- `suffix`  
  length 2 character vector; see `merge`.
- `left`  
  logical; if TRUE return the left join, otherwise an inner join; see details. see also `left_join`.
- `largest`  
  logical; if TRUE, return `x` features augmented with the fields of `y` that have the largest overlap with each of the features of `x`; see https://github.com/r-spatial/sf/issues/578.
- `.predicate`  
  A geometry predicate function with the same profile as `st_intersects`; see details.

**Details**

Alternative values for argument `join` are:

- `st_contains_properly`

• st_contains
• st_covered_by
• st_covers
• st_crosses
• st_disjoint
• st_equal_exact
• st_equals
• st_is_within_distance
• st_nearest_feature
• st_overlaps
• st_touches
• st_within
• any user-defined function of the same profile as the above

A left join returns all records of the x object with y fields for non-matched records filled with NA values; an inner join returns only records that spatially match.

Value

An object of class sftime, joined based on geometry.

Examples

g1 <- st_sfc(st_point(c(1,1)), st_point(c(2,2)), st_point(c(3,3)))
x1 <- st_sftime(a = 1:3, geometry = g1, time = Sys.time())

g2 <- st_sfc(st_point(c(10,10)), st_point(c(2,2)), st_point(c(2,2)), st_point(c(3,3)))
x2 <- st_sftime(a = 11:14, geometry = g2, time = Sys.time())

## st_join
# left spatial join with st_intersects
st_join(x1, x2)

# inner spatial join with st_intersects
st_join(x1, x2, left = FALSE)

## st_filter
st_filter(x1, x2)
st_filter(x2, x1)
Construct an sftime object from all its components

Arguments

... Column elements to be binded into an sftime object or a single list or data.frame with such columns. At least one of these columns shall be a geometry list-column of class sfc and one shall be a time column (to be specified with time_column_name).

agr A character vector; see details below.

row.names row.names for the created sf object.

stringsAsFactors A logical value; see st_read.

crs Coordinate reference system, something suitable as input to st_crs.

precision A numeric value; see st_as_binary.

sf_column_name A character value; name of the active list-column with simple feature geometries; in case there is more than one and sf_column_name is NULL, the first one is taken.
time_column_name
A character value; name of the active time column. In case time_column_name
is NULL, the first POSIXct column is taken. If there is no POSIXct column, the
first Date column is taken.

check_ring_dir
A logical value; see st_read.

sfc_last
A logical value; if TRUE, sfc columns are always put last, otherwise column
order is left unmodified.

time_column_last
A logical value; if TRUE, the active time column is always put last, otherwise
column order is left unmodified. If both sfc_last and time_column_last are
TRUE, the active time column is put last.

x
An object of class sf.

i
Record selection, see [.data.frame

j
Variable selection, see [.data.frame

drop
A logical value, default FALSE; if TRUE drop the geometry column and return a
data.frame, else make the geometry sticky and return an sf object.

op
A function; geometrical binary predicate function to apply when i is a simple
feature object.

value
An object to insert into x.

Details
See also [.data.frame; for [.sftime ... arguments are passed to op.

Value

st_sftime: An object of class sftime.

Returned objects for subsetting functions: [.sf will return a data.frame or vector if the geome-
try column (of class sfc) is dropped (drop=TRUE), an sfc object if only the geometry column is
selected, and otherwise return an sftime object.

Examples

## construction with an sfc object
library(sf)
g <- st_sfc(st_point(1:2))
tc <- Sys.time()
st_sftime(a = 3, g, time = tc)

## construction with an sf object
## Not run:
st_sftime(st_sf(a = 3, g), time = tc)
# error, because if ... contains a data.frame-like object, no other objects
# may be passed through ... . Instead, add the time column before.

## End(Not run)
st_sftime(st_sf(a = 3, g, time = tc))
## Subsetting

g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)),
            st_point(c(2, 1)), st_point(c(3, 1)))
tc <- Sys.time() + 1:5
x <- st_sftime(a = 1:5, g, time = tc)

# rows
x[1,]
class(x[1,])
x[x$a < 3,]
class(x[x$a < 3,])

# columns
x[, 1]
class(x[, 1])  # drops time column as for ordinary data.frame subsetting,
               # keeps geometry column of sf object
x[, 3]
class(x[, 3])  # keeps time column because it is explicitly selected,
               # keeps geometry column of sf object, returns an sftime object
x[, 3, drop = TRUE]
class(x[, 3, drop = TRUE])  # if the geometry column is dropped, not only the
                           # sf class is dropped, but also the sftime class

x["a"]
class(x["a"])  # Time columns are not sticky: If a column is selected by a
               # character vector and this does not contain the active time column, the time
               # column is dropped.

x[c("a", "time")]
class(x[c("a", "time")])  # keeps the time column

# with sf or sftime object
pol = st_sfc(st_polygon(list(cbind(c(0,2,2,0,0),c(0,0,2,2,0)))))
h = st_sf(r = 5, pol)

x[h,]
class(x[h,])  # returns sftime object
h[x,]
class(h[x,])  # returns sf object

## Assigning values to columns

# assigning new values to a non-time column
x[["a"]]<- 5:1
class(x)

# assigning allowed new values to the time column
x[["time"]]<- Sys.time() + 1:5
class(x)

# assigning new values to the time column which invalidate the time column
x[["time"]]<-list(letters[1:2])
class(x)

# assigning new values with `$`
x$time<-Sys.time()+1:5
class(x)

---

st_time  Get, set, or replace time information

Description
Get, set, or replace time information

Usage

st_time(obj, ...)
st_time(x, ...) <- value

## S3 method for class 'sftime'
st_time(obj, ...)

## S3 replacement method for class 'sf'
st_time(x, ..., time_column_name = "time") <- value

## S3 replacement method for class 'sftime'
st_time(x, ...) <- value

st_set_time(x, value, ...)
st_drop_time(x)

Arguments

obj      An object of class sftime.
...      Additional arguments; Ignored.
x        An object of class sftime or sf.
value    An object for which is_sortable returns TRUE or an object of class character, or NULL.
time_column_name    Character value; The name of the column to set as active time column in x.
Details

In case value is character and x is of class sftime, the active time column (as indicated by attribute time_column) is set to x[[value]].

The replacement function applied to sftime objects will overwrite the active time column, if value is NULL, it will remove it and coerce x to an sftime object.

st_drop_time drops the time column of its argument, and reclasses it accordingly.

Value

st_time returns the content of the active time column of an sftime object. Assigning an object for which is_sortable returns TRUE to an sf object creates an sftime object. Assigning an object for which is_sortable returns TRUE to an sftime object replaces the active time column by this object.

Examples

```r
# from sftime object
g <- st_sfc(st_point(1:2))
time <- Sys.time()
x <- st_sftime(a = 3, g, time = time)
st_time(x)

## assign a vector with time information

# to sf object
x <- st_sf(a = 3, g)
st_time(x) <- time
x

# to sftime object
x <- st_sftime(a = 3, g, time = time)
st_time(x) <- Sys.time()

## remove time column from sftime object
st_time(x) <- NULL

## pipe-friendly

# assign time column to sf object
x <- st_sf(a = 3, g)
x <- st_set_time(x, time)

# remove time column from sftime object
st_set_time(x, NULL)

## drop time column and class

# same as x <- st_set_time(x, NULL)
st_drop_time(x)
```
Description

'tidyverse' methods for sftime objects. Geometries are sticky, use `as.data.frame` to let `dplyr`'s own methods drop them. Use these methods without the `.sftime` suffix and after loading the 'tidyverse' package with the generic (or after loading package 'tidyverse').

Usage

```r
inner_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
left_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
right_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
full_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
semi_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
anti_join.sftime(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) 
filter.sftime(.data, ..., .dots) 
arrange.sftime(.data, ..., .dots) 
group_by.sftime(.data, ..., add = FALSE) 
ungroup.sftime(.data, ...) 
rowwise.sftime(.data, ...) 
mutate.sftime(.data, ..., .dots) 
transmute.sftime(.data, ..., .dots) 
select.sftime(.data, ...) 
rename.sftime(.data, ...) 
slice.sftime(.data, ..., .dots) 
summarise.sftime(.data, ..., .dots, do_union = TRUE, is_coverage = FALSE) 
summarize.sftime(.data, ..., .dots, do_union = TRUE, is_coverage = FALSE)
```
distinct.sftime(.data, ..., .keep_all = FALSE)

## S3 method for class 'sftime'
gather(
  data,
  key,
  value,
  ..., 
  na.rm = FALSE,
  convert = FALSE,
  factor_key = FALSE
)

## S3 method for class 'sftime'
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = NULL,
  names_transform = NULL,
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = NULL,
  values_transform = NULL,
  ...
)

## S3 method for class 'sftime'
spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

sample_n.sftime(
  tbl,
  size,
  replace = FALSE,
  weight = NULL,
  .env = parent.frame()
)

sample_frac.sftime(
  tbl,
  size = 1,
  replace = FALSE,
  weight = NULL,
  .env = parent.frame()
## S3 method for class 'sftime'
nest(.data, ...)

## S3 method for class 'sftime'
unnest(data, ..., .preserve = NULL)

## S3 method for class 'sftime'
separate(
  data,
  col,
  into,
  sep = "\^[[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  extra = "warn",
  fill = "warn",
  ...
)

## S3 method for class 'sftime'
unite(data, col, ..., sep = "_", remove = TRUE)

## S3 method for class 'sftime'
separate_rows(data, ..., sep = "\^[[:alnum:]]+", convert = FALSE)

### Arguments

- **x**: An object of class `sftime`.
- **y**: A pair of data frames, data frame extensions (e.g. a tibble), or lazy data frames (e.g. from dbplyr or dtplyr). See `Methods`, below, for more details.
- **by**: A character vector of variables to join by.
  - If `NULL`, the default, `*_join()` will perform a natural join, using all variables in common across `x` and `y`. A message lists the variables so that you can check they're correct; suppress the message by supplying `by` explicitly.
  - To join by different variables on `x` and `y`, use a named vector. For example, `by = c("a" = "b")` will match `x$a` to `y$b`.
  - To join by multiple variables, use a vector with length > 1. For example, `by = c("a", "b")` will match `x$a` to `y$a` and `x$b` to `y$b`. Use a named vector to match different variables in `x` and `y`. For example, `by = c("a" = "b", "c" = "d")` will match `x$a` to `y$b` and `x$c` to `y$d`.
  - To perform a cross-join, generating all combinations of `x` and `y`, use `by = character()`.
- **copy**: If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same src as `x`. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
- **suffix**: If there are non-joined duplicate variables in `x` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
other arguments
.data An object of class stime.
.dots see corresponding function in package dplyr
add see corresponding function in dplyr
do_union logical; in case summary does not create a geometry column, should geometries be created by unioning using st_union, or simply by combining using st_combine? Using st_union resolves internal boundaries, but in case of unioning points, this will likely change the order of the points; see Details.
is_coverage logical; if do_union is TRUE, use an optimized algorithm for features that form a polygonal coverage (have no overlaps)
.keep_all see corresponding function in dplyr
data see original function docs
key see original function docs
value see original function docs
na.rm see original function docs
convert see separate_rows
factor_key see original function docs
cols <tidy-select> Columns to pivot into longer format.
names_to A string specifying the name of the column to create from the data stored in the column names of data.
Can be a character vector, creating multiple columns, if names_sep or names_pattern is provided. In this case, there are two special values you can take advantage of:
- NA will discard that component of the name.
- .value indicates that component of the name defines the name of the column containing the cell values, overriding values_to.
names_prefix A regular expression used to remove matching text from the start of each variable name.
names_sep If names_to contains multiple values, these arguments control how the column name is broken up.
names_sep takes the same specification as separate(), and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).
names_pattern takes the same specification as extract(), a regular expression containing matching groups (\()). If these arguments do not give you enough control, use pivot_longer_spec() to create a spec object and process manually as needed.
names_pattern If names_to contains multiple values, these arguments control how the column name is broken up.
names_sep takes the same specification as separate(), and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).
names_pattern takes the same specification as extract(), a regular expression containing matching groups (()).  
If these arguments do not give you enough control, use pivot_longer_spec() to create a spec object and process manually as needed.

names_ptypes A list of column name-prototype pairs. A prototype (or ptype for short) is a zero-length vector (like integer() or numeric()) that defines the type, class, and attributes of a vector. Use these arguments to confirm that the created columns are the types that you expect.

If not specified, the type of the columns generated from names_to will be character, and the type of the variables generated from values_to will be the common type of the input columns used to generate them.

names_transform A list of column name-function pairs. Use these arguments if you need to change the type of specific columns. For example, names_transform = list(week = as.integer) would convert a character week variable to an integer.

names_repair What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See vctrs::vec_as_names() for more options.

values_to A string specifying the name of the column to create from the data stored in cell values. If names_to is a character containing the special .value sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

values_drop_na If TRUE, will drop rows that contain only NAs in the value_to column. This effectively converts explicit missing values to implicit missing values, and should generally be used only when missing values in data were created by its structure.

values_ptypes A list of column name-prototype pairs. A prototype (or ptype for short) is a zero-length vector (like integer() or numeric()) that defines the type, class, and attributes of a vector. Use these arguments to confirm that the created columns are the types that you expect.

If not specified, the type of the columns generated from names_to will be character, and the type of the variables generated from values_to will be the common type of the input columns used to generate them.

values_transform A list of column name-function pairs. Use these arguments if you need to change the type of specific columns. For example, names_transform = list(week = as.integer) would convert a character week variable to an integer.
weight  see original function docs
.env  see original function docs
.preserve  see unnest
col  see separate
into  see separate
remove  see separate
extra  see separate

Value

- For _join methods: An object of class sftime representing the joining result of x and y. See mutate-joins.
- For filter: See filter.
- For arrange: See arrange.
- For group_by and ungroup: A grouped sftime object. See arrange.
- For rowwise: An sftime object. See rowwise.
- For mutate and transmute: See mutate.
- For select: See select. If the active time column is not explicitly selected, a sf object is returned.
- For rename: See rename.
- For slice: See slice.
- For summarize and summarise: See summarise.
- For distinct: See distinct.
- For gather: See gather.

Examples

g1 <- st_sfc(st_point(1:2), st_point(c(5, 8)), st_point(c(2, 9)))
x1 <- st_sftime(a = 1:3, geometry = g1, time = Sys.time())

g2 <- st_sfc(st_point(c(4, 6)), st_point(c(4, 6)), st_point(c(4, 6)))
x2 <- st_sftime(a = 2:4, geometry = g2, time = Sys.time())

library(dplyr)

## inner_join
inner_join(x1, as.data.frame(x2), by = "a") # note: the active time column is # time.x and the active geometry column geometry.x

inner_join(x2, as.data.frame(x1), by = "a")

## left_join
left_join(x1, as.data.frame(x2), by = "a")

left_join(x2, as.data.frame(x1), by = "a")
## right_join
right_join(x1, as.data.frame(x2), by = "a")
right_join(x2, as.data.frame(x1), by = "a")

## full_join
full_join(x1, as.data.frame(x2), by = "a")
full_join(x2, as.data.frame(x1), by = "a")

## semi_join
semi_join(x1, as.data.frame(x2), by = "a")
semi_join(x2, as.data.frame(x1), by = "a")

## anti_join
anti_join(x1, as.data.frame(x2), by = "a")
anti_join(x2, as.data.frame(x1), by = "a")

## filter
filter(x1, a <= 2)

## arrange
arrange(x1, dplyr::desc(a))

## group_by
group_by(x1, time)

## ungroup
ungroup(group_by(x1, time))

## rowwise
x1 %>%
    mutate(a1 = 5:7) %>%
    rowwise() %>%
    mutate(a2 = mean(a, a1))

## mutate
x1 %>%
    mutate(a1 = 5:7)

## transmute
x1 %>%
    transmute(a1 = 5:7)

## select
x1 %>%
    select(-time) %>%
    select(geometry)

## rename
x1 %>%
  rename(a1 = a)

## slice
x1 %>%
slice(1:2)

## summarise
x1 %>%
  summarise(time = mean(time))

x1 %>%
  summarize(time = mean(time))

## distinct
x1 %>%
distinct(geometry)

## gather
library(tidyr)

x1 %>%
  mutate(a1 = 5:7) %>%
  gather(key = "variable", value = "value", a, a1)

## pivot_longer
x1 %>%
  mutate(a1 = 5:7) %>%
  pivot_longer(cols = c("a", "a1"), names_to = "variable", values_to = "value")

## spread
x1 %>%
  mutate(a1 = 5:7) %>%
  gather(key = "variable", value = "value", a, a1) %>%
  spread(key = "variable", value = "value")

## sample_n
set.seed(234)

x1 %>%
  sample_n(size = 10, replace = TRUE)

## sample_frac
x1 %>%
  sample_frac(size = 10, replace = TRUE) %>%
  sample_frac(size = 0.1, replace = FALSE)

## nest
x1 %>%
nest(a1 = -time)

## unnest
x1 %>%
  mutate(a1 = list(1, c(1, 2), 5)) %>%
  unnest(a1)
transform.sftime

## separate
x1 %>%
  mutate(x = c(NA, "a.b", "a.d")) %>%
  separate(x, c("A", "B"))

## unite
x1 %>%
  mutate(x = c(NA, "a.b", "a.d")) %>%
  separate(x, c("A", "B")) %>%
  unite(x, c("A", "B"))

## separate_rows
x1 %>%
  mutate(z = c("1", "2,3,4", "5,6")) %>%
  separate_rows(z, convert = TRUE)

---

**transform.sftime**

*Transform method for sftime objects*

### Description

Can be used to create or modify attribute variables; for transforming geometries see `st_transform`, and all other functions starting with `st_`.

### Usage

```r
## S3 method for class 'sftime'
transform(~_data~, ...)
```

### Arguments

- **_data**  
  An object of class `sftime`.

- **...**  
  Further arguments of the form `new_variable=expression`

### Value

_`_data (an sftime object) with modified attribute values (columns).

### Examples

```r
# create an sftime object
g <- st_sfc(st_point(c(1, 2)), st_point(c(1, 3)), st_point(c(2, 3)),
            st_point(c(2, 1)), st_point(c(3, 1)))
x <-
  data.frame(a = 1:5, g, time = Sys.time() + 1:5, stringsAsFactors = FALSE)
x_sftime <- st_as_sftime(x)
x_sftime
```
# modify values in column a
transform(x_sftime, a = rev(a))
Index

[.data.frame, 16
[.sftime(st_sftime), 15
[[<-.sftime(st_sftime), 15
$<-.sftime(st_sftime), 15

anti_join.sftime(tidyverse), 20
arrange, 25
arrange.sftime(tidyverse), 20
as.data.frame, 20

bind, 2
bind_cols, 2

cbind, 2
cbind.sftime(bind), 2

data.frame, 2
Date, 16
distinct, 25
distinct.sftime(tidyverse), 20
dotsMethods, 2

eextract(), 23, 24

filter, 25
filter.sftime(tidyverse), 20
full_join.sftime(tidyverse), 20

gather, 25
gather.sftime(tidyverse), 20
group_by.sftime(tidyverse), 20

inner_join.sftime(tidyverse), 20
is_sortable, 18, 19

left_join, 13
left_join.sftime(tidyverse), 20
merge, 13

mutate, 25
mutate.sftime(tidyverse), 20

nest.sftime(tidyverse), 20

pivot_longer.sftime(tidyverse), 20
plot(plot.sftime), 6
plot.sf, 6
plot.sftime, 6
POSIXct, 16
print.sftime, 7

rbind, 2
rbind.sftime(bind), 2
rename, 25
rename.sftime(tidyverse), 20
right_join.sftime(tidyverse), 20
rowwise, 25
rowwise.sftime(tidyverse), 20

sample_frac.sftime(tidyverse), 20
sample_n.sftime(tidyverse), 20
select, 25
select.sftime(tidyverse), 20
semi_join.sftime(tidyverse), 20
separate, 25
separate(), 23
separate.sftime(tidyverse), 20
separate_rows, 23, 24
separate_rows.sftime(tidyverse), 20
slice, 25
slice.sftime(tidyverse), 20
spread.sftime(tidyverse), 20
st_as_binary, 15
st_as_sf, 9
st_as_sftime, 8
st_bbox, 11
st_cast, 10
st_combine, 23
st_contains, 14
st_contains_properly, 13
st_covered_by, 14
st_covers, 14
st_crop, 12
st_crop.sftime, 11
st_crosses, 14
st_crs, 15
st_difference.sftime (geos_binary_ops), 3
st_disjoint, 14
st_drop_geometry.sftime (st_geometry), 12
st_drop_time (st_time), 18
st_equals, 14
st_equals_exact, 14
st_filter.sftime (st_join), 13
st_geometry, 12
st_intersection.sftime
  (geos_binary_ops), 3
st_intersects, 13
st_is_within_distance, 14
st_join, 13
st_nearest_feature, 14
st_overlaps, 14
st_point, 9
st_read, 15, 16
st_relate, 13
st_set_time (st_time), 18
st_sf, 9
st_sftime, 2, 3, 15
st_sym_difference.sftime
  (geos_binary_ops), 3
st_time, 18
st_time<- (st_time), 18
st_touches, 14
st_transform, 28
st_union, 23
st_union.sftime (geos_combine), 5
st_within, 14
summarise, 25
summarise.sftime (tidyverse), 20
summarize.sftime (tidyverse), 20
tidyverse, 20
transform.sftime, 28
transmute.sftime (tidyverse), 20
ungroup.sftime (tidyverse), 20
unite.sftime (tidyverse), 20
unnest, 25
unnest.sftime (tidyverse), 20
vctrs::vec_as_names(), 24