Package 'cubble'

August 27, 2024

Title A Vector Spatio-Temporal Data Structure for Data Analysis

Version 1.0.0

Description A spatiotemperal data object in a relational data structure to separate the recording of time variant/ invariant variables. See the Journal of Statistical Software reference: <doi:10.18637/jss.v110.i07>.

License MIT + file LICENSE

Language en-US

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports cli, dplyr, ggplot2, glue, lubridate, ncdf4, rlang, sf, stringr, tibble, tidyr, tidyselect, tsibble, vctrs

Suggests spelling, rmarkdown, knitr, testthat (>= 3.0.0), ozmaps, GGally, ggrepel, ggforce, purrr, stars, units, leaflet, plotly, crosstalk, concaveman, colorspace, vdiffr, sftime, patchwork

VignetteBuilder knitr

Depends R (>= 3.5.0)

Config/testthat/edition 3

URL https://github.com/huizezhang-sherry/cubble, https://huizezhang-sherry.github.io/cubble/

BugReports https://github.com/huizezhang-sherry/cubble/issues

NeedsCompilation no

Author H. Sherry Zhang [aut, cre] (<https://orcid.org/0000-0002-7122-1463>), Dianne Cook [aut] (<https://orcid.org/0000-0002-3813-7155>), Ursula Laa [aut] (<https://orcid.org/0000-0002-0249-6439>), Nicolas Langrené [aut] (<https://orcid.org/0000-0001-7601-4618>), Patricia Menéndez [aut] (<https://orcid.org/0000-0003-0701-6315>)

Maintainer H. Sherry Zhang <huizezhangsh@gmail.com>

Repository CRAN

Date/Publication 2024-08-27 15:20:02 UTC

Contents

`arrange.temporal_cubble_df` `dplyr` *methods*

Description

Verbs supported for both nested and long cubble include: `dplyr::mutate()`, `dplyr::filter()`, `dplyr::arrange()`, `dplyr::select()`, `dplyr::group_by()`, `dplyr::ungroup()`, `dplyr::summarise()`,. `dplyr::rename()`, `dplyr::bind_cols()`, `dplyr::rowwise()`, `dplyr::slice_*()`, `dplyr::*_join()`, `dplyr::relocate()`, `dplyr::pull()`

Usage

```
## S3 method for class 'temporal_cubble_df'
arrange(.data, ...)

## S3 method for class 'spatial_cubble_df'
select(.data, ...)

## S3 method for class 'temporal_cubble_df'
select(.data, ...)

## S3 method for class 'spatial_cubble_df'
```

```
group_by(.data, ..., .add, .drop)

## S3 method for class 'temporal_cubble_df'
group_by(.data, ..., .add, .drop)

## S3 method for class 'spatial_cubble_df'
ungroup(x, ...)

## S3 method for class 'temporal_cubble_df'
ungroup(x, ...)

## S3 method for class 'spatial_cubble_df'
summarise(.data, ..., .by = NULL, .groups = NULL)

## S3 method for class 'temporal_cubble_df'
summarise(.data, ..., .by = key_vars(.data), .groups = NULL)

## S3 method for class 'spatial_cubble_df'
rename(.data, ...)

## S3 method for class 'temporal_cubble_df'
rename(.data, ...)

bind_rows.temporal_cubble_df(..., .id = NULL)

bind_cols.spatial_cubble_df(..., .name_repair)

bind_cols.temporal_cubble_df(..., .name_repair)

## S3 method for class 'spatial_cubble_df'
rowwise(data, ...)

## S3 method for class 'temporal_cubble_df'
rowwise(data, ...)

## S3 method for class 'cubble_df'
dplyr_col_modify(data, cols)

## S3 method for class 'spatial_cubble_df'
dplyr_row_slice(data, i, ...)

## S3 method for class 'temporal_cubble_df'
dplyr_row_slice(data, i, ...)

## S3 method for class 'spatial_cubble_df'
dplyr_reconstruct(data, template)

## S3 method for class 'temporal_cubble_df'
```

```
dplyr_reconstruct(data, template)

## S3 method for class 'spatial_cubble_df'
mutate(.data, ...)

## S3 method for class 'temporal_cubble_df'
mutate(.data, ...)

## S3 method for class 'spatial_cubble_df'
filter(.data, ...)

## S3 method for class 'spatial_cubble_df'
arrange(.data, ...)
```

Arguments

...	In group_by(), variables or computations to group by. Computations are always done on the ungrouped data frame. To perform computations on the grouped data, you need to use a separate mutate() step before the group_by(). Computations are not allowed in nest_by(). In ungroup(), variables to remove from the grouping.
.add	When FALSE, the default, group_by() will override existing groups. To add to the existing groups, use .add = TRUE. This argument was previously called add, but that prevented creating a new grouping variable called add, and conflicts with our naming conventions.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE. See group_by_drop_default() for details.
x	A tbl()
.by	**[Experimental]** <tidy-select> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to group_by(). For details and examples, see ?dplyr_by.
.groups	**[Experimental]** Grouping structure of the result. • "drop_last": dropping the last level of grouping. This was the only supported option before version 1.0.0. • "drop": All levels of grouping are dropped. • "keep": Same grouping structure as .data. • "rowwise": Each row is its own group. When .groups is not specified, it is chosen based on the number of rows of the results: • If all the results have 1 row, you get "drop_last". • If the number of rows varies, you get "keep" (note that returning a variable number of rows was deprecated in favor of reframe(), which also unconditionally drops all levels of grouping).

	In addition, a message informs you of that choice, unless the result is ungrouped, the option "dplyr.summarise.inform" is set to FALSE, or when summarise() is called from a function in a package.
.id	The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.
.name_repair	One of "unique", "universal", or "check_unique". See vctrs::vec_as_names() for the meaning of these options.
data, .data	a cubble object of class spatial_cubble_df or temporal_cubble_df
cols	A named list used to modify columns. A NULL value should remove an existing column.
i	A numeric or logical vector that indexes the rows of data.
template	Template data frame to use for restoring attributes.

Details

You may find not all the verbs have a verb.spatial_cubble_df or verb.temporal_cubble_df implemented. These verbs call the dplyr extending trios: dplyr_row_slice, dplyr_col_modify, and dplyr_reconstruct under the hood. See https://dplyr.tidyverse.org/reference/dplyr_extending.html

Examples

```
library(dplyr)
cb_nested <- climate_mel
cb_long <- face_temporal(climate_mel)

# filter - currently filter.spatial_cubble_df, dply_row_slice
cb_nested |> filter(elev > 40)
cb_long |> filter(prcp > 0)

# mutate - curerntly mutate.spatial_cubble_df, dply_col_modify
cb_nested |> mutate(elev2 = elev + 10)
cb_long |> mutate(prcp2 = prcp + 10)

# arrange - currently arrange.spatial_cubble_df, arrange.temporal_cubble_df
cb_nested |> arrange(wmo_id)
cb_long |> arrange(prcp)

# summarise - summarise.spatial_cubble_df,  summarise.temporal_cubble_df
cb_long |>
  group_by(first_5 = ifelse(lubridate::day(date) <=5, 1, 2 )) |>
  summarise(tmax = mean(tmax))
cb_long |>
  mutate(first_5 = ifelse(lubridate::day(date) <=5, 1, 2)) |>
  summarise(t = mean(tmax), .by = first_5)

# select -  select.spatial_cubble_df,  select.temporal_cubble_df
cb_nested |> select(name)
cb_nested |> select(-id, -name)
```

```
cb_long |> select(prcp)
cb_long |> select(-prcp, -date)

# rename - rename.spatial_cubble_df, rename.temporal_cubble_df
cb_nested |> rename(elev2 = elev)
cb_long |> rename(prcp2 = prcp)
# rename on key attributes
cb_nested |> rename(id2 = id)
cb_long |> rename(date2 = date)

# join - mutate_join - dplyr_reconstruct()
# join - filter_join - dplyr_row_slice()
df1 <- cb_nested |> as_tibble() |> select(id, name) |> head(2)
nested <- cb_nested |> select(-name)
nested |> left_join(df1, by = "id")
nested |> right_join(df1, by = "id")
nested |> inner_join(df1, by = "id")
nested |> full_join(df1, by = "id")
nested |> anti_join(df1, by = "id")

# bind_rows - dplyr_reconstruct, bind_rows.temporal_cubble_df
df1 <- cb_nested |> head(1)
df2 <- cb_nested |> tail(2)
bind_rows(df1, df2)
df1 <- cb_long |> head(10)
df2 <- cb_long |> tail(20)
bind_rows(df1, df2)

# relocate - dplyr_col_select, dplyr_col_select
cb_nested |> relocate(ts, .before = name)
cb_nested |> face_temporal() |> relocate(tmin)

# slice - all the slice_* uses dplyr::slice(), which uses dplyr_row_slice()
cb_nested |> slice_head(n = 2)
cb_nested |> slice_tail(n = 2)
cb_nested |> slice_max(elev)
cb_nested |> slice_min(elev)
cb_nested |> slice_sample(n = 2)

# rowwise - rowwise.spatial_cubble_df, rowwise.temporal_cuble_df
cb_nested |> rowwise()
cb_long |> rowwise()

# group_by & ungroup -
(res <- cb_nested |> mutate(group1 = c(1, 1, 2)) |> group_by(group1))
res |> ungroup()
(res2 <- res |> face_temporal())
res2 |> ungroup()
res2 |> mutate(first_5 = ifelse(lubridate::day(date) <= 5, 1, 6)) |>
  group_by(first_5)
```

as_cubble *Coerce foreign objects into a cubble object*

Description

Coerce foreign objects into a cubble object

Usage

```
as_cubble(data, key, index, coords, ...)

## S3 method for class 'data.frame'
as_cubble(data, key, index, coords, ...)

## S3 method for class 'tbl_df'
as_cubble(data, key, index, coords, crs, dimensions, ...)

## S3 method for class 'sf'
as_cubble(data, key, index, ...)

## S3 method for class 'ncdf4'
as_cubble(
  data,
  key,
  index,
  coords,
  vars,
  lat_range = NULL,
  long_range = NULL,
  ...
)

## S3 method for class 'stars'
as_cubble(data, key, index, coords, ...)

## S3 method for class 'sftime'
as_cubble(data, key, index, coords, ...)
```

Arguments

data	an object to be converted into an cubble object. Currently support objects of classes tibble, ncdf4, stars, and sftime.
key	a character (symbol), the spatial identifier, see make_cubble()
index	a character (symbol), the temporal identifier, see make_cubble().
coords	a vector of character (symbol) of length 2, see make_cubble().
...	other arguments.

crs — used in as_cubble.tbl_df() to set the crs. the data to read in as_cubble.netcdf().

dimensions — used when creating a cubble from a stars object

vars — a vector of variables to read in (with quote), used in as_cubble.netcdf() to select the variable to read in.

lat_range, long_range — in the syntax of seq(FROM, TO, BY) to downsample

Value

a cubble object

Examples

```
climate_flat |> as_cubble(key = id, index = date, coords = c(long, lat))

# only need `coords` if create from a tsibble
dt <- climate_flat |>  tsibble::as_tsibble(key = id, index = date)
dt |>  as_cubble(coords = c(long, lat))

# netcdf
path <- system.file("ncdf/era5-pressure.nc", package = "cubble")
raw <- ncdf4::nc_open(path)
dt <- as_cubble(raw)
# subset degree
dt <- as_cubble(raw,vars = c("q", "z"),
                long_range = seq(113, 153, 3),
                lat_range = seq(-53, -12, 3))

## Not run:
# stars - take a few seconds to run
tif <- system.file("tif/L7_ETMs.tif", package = "stars")
x <-  stars::read_stars(tif)
x |> as_cubble(index = band)

## End(Not run)

# don't have to supply coords if create from a sftime
dt <- climate_flat |>
  sf::st_as_sf(coords = c("long", "lat"), crs = sf::st_crs("OGC:CRS84")) |>
  sftime::st_as_sftime()
dt |> as_cubble(key = id, index = date)
```

---

check_key *Check on key when create cubble from two components (spatial/temporal)*

Description

When creating a cubble from separate spatial and temporal component, `make_cubble()` will informed users about potential disagreement of the key values in the two datasets (some sites appear in one table but not the other). This function summarises the key values into those match, potentially can be matched, and can't be matched.

Usage

```
check_key(spatial, temporal, by = NULL)
```

Arguments

`spatial`	a tibble object or an sf object, the spatial component containing the `key` and `coords` variable (`coords` can be automatically created from an `sf` object if not supplied).
`temporal`	a tibble object or a tsibble object, the temporal component containing the `key` and `index` variable.
`by`	in the syntax of the by argument in `dplyr::left_join()`, used in `make_cubble()` when the key variable has different names in the `spatial` and `temporal` data.

Value

a list with three elements: 1) paired: a tibble of paired ID from spatial and temporal data, 2) potential_pairs: a tibble of pairs that could potentially match from both datasets, 3) others: other key values that can't be matched in a list: others$temporal and others$spatial

Examples

```
check_key(stations, meteo)

# make_cubble() will prompt to use check_key if there are key mis-match:
colnames(lga) <- c("lga", "geometry")
cb <- make_cubble(spatial = lga, temporal = covid)
(check_res <- check_key(lga, covid))
make_cubble(spatial = lga, temporal = covid, potential_match = check_res)
```

`climate_aus` *Australia climate data*

Description

`climate_aus`: daily measure on precipitation (`prcp`), maximum temperature (`tmax`), and minimum temperature (`tmin`) in 2020 for 639 stations. `historical_tmax`: daily maximum temperature (`tmax`) for 75 stations in Victoria and New South Wales for two periods: 1971-1975 and 2016-2020.

Usage

```
climate_aus

historical_tmax
```

Format

An object of class `spatial_cubble_df` (inherits from `cubble_df`, `tbl_df`, `tbl`, `data.frame`) with 639 rows and 7 columns.

An object of class `spatial_cubble_df` (inherits from `cubble_df`, `tbl_df`, `tbl`, `data.frame`) with 75 rows and 7 columns.

Details

id station ID, "ASN000" are international paddings, the next two digits (digit 8-9) indicates the states the station is in: Western Australia: 01-13, Northern Territory: 14-15, South Australia: 16-26, Queensland: 27-45, New South Wales: 46-75, Victoria: 76-90, Tasmania: 91-99. See http://www.bom.gov.au/climate/cdo/about/site-num.shtml

lat latitude of the stations, in degree

long longitude of the stations, in degree

elev elevation of the stations

name station name

wmo_id the world meteorological organisation (WMO) station number

ts For `climate_aus`: date, prcp, tmax, and tmin, for `historical_tmax`: date and tmax

Examples

```
climate_aus |> face_temporal() |> face_spatial()
```

---

covid *Daily COVID count data (in* `tsibble`*) and Victoria LGA (in* `sf`*)*

Description

Daily COVID count data (`covid`) from 2022-01-01 to 2020-03-23 in a tsibble object (`date`, `lga`, `n`, and `avg_7day`). Victoria Local Government Area (LGA) spatial geometry in an sf object (`lga_name_2018` and `geometry`)

Usage

```
covid

lga
```

Format

An object of class `tbl_ts` (inherits from `tbl_df`, `tbl`, `data.frame`) with 6806 rows and 4 columns.

An object of class `sf` (inherits from `data.frame`) with 80 rows and 2 columns.

Details

date date object, from 2022-01-01 to 2020-03-23

lga Victoria Local Government Area (LGA) in Australia

n COVID-19 case count

avg_7day rolling mean of `n` in a 7 day window. Calculate with `mutate(avg_7day = slider::slide_dbl(n, mean, .before = 6))`

lga_name_2018 LGA encoding by Australia Bureau of Statistics, slightly differ from the encoding used by the Department of Health in the `covid` data

geometry multipolygon geometry of each LGA

Examples

```
library(sf)
library(dplyr)
# prompt msg on the key mismatch between the two datasets
make_cubble(lga, covid, by = c("lga_name_2018" = "lga"))
check_res <- check_key(lga, covid, by = c("lga_name_2018" = "lga"))

# fix mismatch
lga2 <- lga |>
  rename(lga = lga_name_2018) |>
  mutate(lga = ifelse(lga == "Kingston (C) (Vic.)", "Kingston (C)", lga),
         lga = ifelse(lga == "Latrobe (C) (Vic.)", "Latrobe (C)", lga)) |>
  filter(!lga %in% check_res$others$spatial)
covid2 <- covid |> filter(!lga %in% check_res$others$temporal)

make_cubble(spatial = lga2, temporal = covid2)
```

---

cubble *Create a cubble object*

Description

Create a cubble object

Usage

```
cubble(..., key, index, coords)

make_cubble(
  spatial,
```

```
  temporal,
  by = NULL,
  key,
  index,
  coords,
  potential_match = NULL,
  key_use = "temporal"
)
```

Arguments

...	a set of name-value pairs to create a cubble, need to include the key, index, and coords variables.
key	a character (or symbol), the spatial identifier. See the Key section in tsibble::as_tsibble()
index	a character (or symbol), the temporal identifier. Currently support base R classes Date, POSIXlt, POSIXct and tsibble's tsibble::yearmonth(), tsibble::yearweek(), and tsibble::yearquarter() class. See the Index section in tsibble::as_tsibble()
coords	a vector of character (or symbol) of length two, in the order of longitude first and then latitude, the argument can be omitted if created from an sf and its subclasses. In case the sf geometry column is not POINT, coords will be the centroid coordinates.
spatial	a tibble object or an sf object, the spatial component containing the key and coords variable (coords can be automatically created from an sf object if not supplied).
temporal	a tibble object or a tsibble object, the temporal component containing the key and index variable.
by	in the syntax of the by argument in dplyr::left_join(), used in make_cubble() when the key variable has different names in the spatial and temporal data.
potential_match	a key_tbl object from check_key(). When unmatched key values appear in spatial and temporal data, make_cubble will prompt the user to use check_key() for checking. This argument allow the check result to be parsed back to make_cubble to also match the potential_pairs found by the check.
key_use	a character of either "spatial" or "temporal". When potential_math is activated, this argument specifies which key column in the potential match to use. Default to "temporal".

Value

a cubble object

Examples

```
cubble(
  id = rep(c("perth", "melbourne", "sydney"), each = 3),
  date = rep(as.Date("2020-01-01") + 0:2, times = 3),
  long = rep(c(115.86, 144.96, 151.21), each = 3),
```

```
  lat = rep(c(-31.95, -37.81, -33.87), each = 3),
  value = rnorm(n = 9),
  key = id, index = date, coords = c(long, lat)
  )

# stations and climate are in-built data in cubble
make_cubble(spatial = stations, temporal = meteo,
            key = id, index = date, coords = c(long, lat))
```

---

face_temporal *Pivot a cubble object between the nested/long (spatial/temporal) form*

Description

While face_temporal() switches a cubble object into a long cubble, suitable for temporal operations, face_spatial() turns a long cubble back into a nest cubble for spatial operations. The two operations are exact inverse.

Usage

```
face_temporal(data, col)

## S3 method for class 'temporal_cubble_df'
face_temporal(data, col)

## S3 method for class 'spatial_cubble_df'
face_temporal(data, col)

face_spatial(data)

## S3 method for class 'spatial_cubble_df'
face_spatial(data)

## S3 method for class 'temporal_cubble_df'
face_spatial(data)
```

Arguments

data	a cubble object
col	a character (or a symbol), the list column to be expanded, col is required to be specified if there are more than one list column and the list column name is not ts.

Value

a cubble object

References

cubble: An R Package for Organizing and Wrangling Multivariate Spatio-Temporal Data

Examples

```
cb_long <- climate_mel |> face_temporal()
cb_back <- cb_long |> face_spatial()
identical(climate_mel, cb_back)
```

`fill_gaps.temporal_cubble_df` *Gap-filling on the temporal component of a cubble object*

Description

Gap-filling on the temporal component of a cubble object

Usage

```
## S3 method for class 'temporal_cubble_df'
fill_gaps(.data, ..., .full = FALSE, .start = NULL, .end = NULL)

## S3 method for class 'temporal_cubble_df'
scan_gaps(.data, ...)

## S3 method for class 'temporal_cubble_df'
index_by(.data, ...)
```

Arguments

`.data`	A tsibble.
`...`	A set of name-value pairs. The values provided will only replace missing values that were marked as "implicit", and will leave previously existing `NA` untouched. • empty: filled with default `NA`. • filled by values or functions.
`.full`	• `FALSE` inserts `NA` for each keyed unit within its own period. • `TRUE` fills `NA` over the entire time span of the data (a.k.a. fully balanced panel). • `start()` pad `NA` to the same starting point (i.e. `min(<index>)`) across units. • `end()` pad `NA` to the same ending point (i.e. `max(<index>)`) across units.
`.start, .end`	Set custom starting/ending time that allows to expand the existing time spans.

Value

a cubble object

Examples

```
library(tsibble)
climate_aus |> face_temporal() |> fill_gaps()
climate_aus |> face_temporal() |> scan_gaps()
```

---

geom_glyph *Create glyph map with ggplot2*

Description

Create glyph map with ggplot2

Usage

```
geom_glyph(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  x_major = NULL,
  x_minor = NULL,
  y_major = NULL,
  y_minor = NULL,
  x_scale = identity,
  y_scale = identity,
  polar = FALSE,
  width = ggplot2::rel(2.1),
  height = ggplot2::rel(1.8),
  global_rescale = TRUE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_glyph_line(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  x_major = NULL,
  x_minor = NULL,
  y_major = NULL,
  y_minor = NULL,
  polar = FALSE,
  width = ggplot2::rel(2.1),
  height = ggplot2::rel(2.1),
```

```
  show.legend = NA,
  inherit.aes = TRUE
)

geom_glyph_box(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  x_major = NULL,
  x_minor = NULL,
  y_major = NULL,
  y_minor = NULL,
  polar = FALSE,
  width = ggplot2::rel(2.1),
  height = ggplot2::rel(2.1),
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`x_major`, `x_minor`, `y_major`, `y_minor`
The name of the variable (as a string) for the major and minor x and y axes. Together, each unique combination of `x_major` and `y_major` specifies a grid cell.

`y_scale`, `x_scale`
The scaling function to be applied to each set of minor values within a grid cell. Defaults to `identity` so that no scaling is performed.

`polar` A logical of length 1, specifying whether the glyphs should be drawn in polar coordinates. Defaults to `FALSE`.

`height`, `width` The height and width of each glyph. Defaults to 95% of the `resolution` of the data. Specify the width absolutely by supplying a numeric vector of length 1, or relative to the resolution of the data by using `rel`.

`global_rescale` Whether rescale is performed globally or on each individual glyph.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Value

a ggplot object

Examples

```
print_p <- GGally::print_if_interactive

library(ggplot2)
# basic glyph map with reference line and box---------------
p <- ggplot(data = GGally::nasa,
       aes(x_major = long, x_minor = day,
           y_major = lat, y_minor = surftemp)) +
  geom_glyph_box() +
  geom_glyph_line() +
  geom_glyph() +
  theme_bw()
print_p(p)

# rescale on each individual glyph ---------------
p <- ggplot(data = GGally::nasa,
       aes(x_major = long, x_minor = day,
           y_major = lat, y_minor = surftemp)) +
  geom_glyph(global_rescale = FALSE)
print_p(p)

# adjust width and height with relative & absolute value ---------------
p <- ggplot() +
  geom_glyph(data = GGally::nasa,
             aes(x_major = long, x_minor = day,
                 y_major = lat, y_minor = surftemp),
                 width = rel(0.8), height = 1) +
   theme_bw()
print_p(p)

# apply a re-scaling on Y and use polar coordinate
p <-
  GGally::nasa |>
  ggplot(aes(x_major = long, x_minor = day,
             y_major = lat, y_minor = ozone)) +
    geom_glyph_box(fill=NA) +
    geom_glyph_line() +
    geom_glyph(y_scale = GGally::range01, polar = TRUE)
print_p(p)
```

is_cubble *Predicate functions on the object class*

Description

Predicate functions on the object class

Usage

```
is_cubble(data)

is_cubble_spatial(data)

is_cubble_temporal(data)

is_sf(data)

is_tsibble(data)
```

Arguments

data	an object to test for the class

Value

a logical value of TRUE/FALSE

Examples

```
is_cubble(stations)
is_cubble(meteo)
is_cubble(climate_flat)
is_cubble(climate_mel)
is_cubble(climate_aus)
is_cubble_spatial(climate_aus)
is_cubble_temporal(climate_aus)
```

key_vars.cubble_df *Extract cubble attributes*

Description

Extract cubble attributes

Usage

```
## S3 method for class 'cubble_df'
key_vars(x)

## S3 method for class 'cubble_df'
key(x)

## S3 method for class 'cubble_df'
key_data(.data)

coords(data)

spatial(data)

## S3 method for class 'spatial_cubble_df'
spatial(data)

## S3 method for class 'temporal_cubble_df'
spatial(data)

index(data)

index_var(data)
```

Arguments

x, .data, data	a cubble object

Examples

```
library(tsibble)
key(climate_mel)
key_vars(climate_mel)
key_data(climate_mel)
cubble::index(climate_mel)
cubble::index_var(climate_mel)
coords(climate_mel)
spatial(climate_mel)
```

---

make_spatial_sf *Update the spatial cubble to include the sf class*

Description

add geometry list column to cubble_df object

Usage

```
make_spatial_sf(x, sfc = NULL, crs, silent = FALSE)
```

Arguments

x	object of class spatial_cubble_df
sfc	object of class sfc (see package sf)
crs	object of class crs (see package sf); if missing 'OGC:CRS84' is assumed (WGS84) and a message is emitted
silent	logical; suppress message?

See Also

make_temporal_tsibble

Examples

```
climate_mel |> make_spatial_sf()
```

---

make_temporal_tsibble *Update the temporal cubble to include the tsibble class (*tbl_ts*)*

Description

Update the temporal cubble to include the tsibble class (tbl_ts)

Usage

```
make_temporal_tsibble(x)
```

Arguments

x	object of class temporal_cubble_df

Examples

```
climate_mel |> face_temporal() |> make_temporal_tsibble()
```

match_sites *Match stations in two cubbles by spatial distance/ temporal similarity*

Description

The spatial matching is calculated using `sf::st_distance()` with different distance (in meter or degree) available depending on the coordinate reference system and parameter (`which` and `par`). The temporal matching is based on a temporal matching function (`temporal_match_fn`) that can be customised.

Usage

```
match_sites(
  df1,
  df2,
  crs = sf::st_crs("OGC:CRS84"),
  which = NULL,
  par = 0,
  spatial_n_each = 1,
  spatial_n_group = 4,
  data_id,
  match_id,
  temporal_matching = TRUE,
  temporal_by,
  temporal_match_fn = match_peak,
  temporal_n_highest = 20,
  temporal_window = 5,
  ...
)

match_spatial(
  df1,
  df2,
  crs = sf::st_crs("OGC:CRS84"),
  which = NULL,
  par = 0,
  spatial_n_each = 1,
  spatial_n_group = 4,
  return_cubble = FALSE
)

match_temporal(
  data,
  data_id,
  match_id = NULL,
  temporal_by,
  return_cubble = FALSE,
```

```
  temporal_match_fn = match_peak,
  temporal_n_highest = 30,
  temporal_window = 5,
  ...
)
```

Arguments

df1, df2	the two cubble objects to match
crs	a crs object from sf::st_crs()
which	character; for Cartesian coordinates only: one of Euclidean, Hausdorff or Frechet; for geodetic coordinates, great circle distances are computed; see details
par	for which equal to Hausdorff or Frechet, optionally use a value between 0 and 1 to densify the geometry
spatial_n_each	integer, the number of matched "station" in df2 for each df1 record
spatial_n_group	integer, the number of matched group (pair) return
data_id	a character (or symbol), the variable differentiates df1 and df2
match_id	a character (or symbol), the variable differentiate each group of match
temporal_matching	logical, whether to match temporally
temporal_by	in the by syntax in dplyr::*_join(), the variables to match temporally in df1 and df2.
temporal_match_fn	character, the function name on how two time series should be matched
temporal_n_highest	numeric, the number of highest peak used for temporal matching in match_peak
temporal_window	The temporal window allowed in match_peak
...	parameters passing to temporal match
return_cubble	logical (default to false), whether to return the cubble object or a matching summary table
data	the resulting cubble object from spatial matching (with return_cubble = TRUE in spatial matching)

Examples

```
library(dplyr)
climate_aus <- mutate(climate_aus, type = "climate")
match_spatial(climate_aus, river)
# turn with different distance calculation:
match_spatial(climate_aus, river, which = "Hausdorff")
# tune the number of matches in each group
match_spatial(climate_aus, river, spatial_n_each = 5, spatial_n_group = 2)
```

```
a1 <- match_spatial(climate_aus, river, return_cubble = TRUE) |> bind_rows()
match_temporal(a1, data_id = type, match_id = group,
               temporal_by = c("prcp" = "Water_course_level"))
```

`print.cubble_df` *Print methods*

Description

Print methods

Usage

```
## S3 method for class 'cubble_df'
print(x, width = NULL, ...)

## S3 method for class 'spatial_cubble_df'
tbl_sum(x)

## S3 method for class 'temporal_cubble_df'
tbl_sum(x)
```

Arguments

`x`	any R object (conceptually); typically numeric.
`width`	`default` method: the *minimum* field width or `NULL` or `0` for no restriction. `AsIs` method: the *maximum* field width for non-character objects. `NULL` corresponds to the default `12`.
`...`	further arguments passed to or from other methods.

Examples

```
climate_mel # a nested/spatial cubble
face_temporal(climate_mel) # a long/temporal cubble
```

`river` *Australia river data*

Description

Australia river data

Usage

```
river
```

Format

An object of class spatial_cubble_df (inherits from cubble_df, tbl_df, tbl, data.frame) with 71 rows and 6 columns.

Examples

```
river
```

---

stations *Toy climate data*

Description

Daily measure (2020-01-01 to 2020-01-10) on precipitation (prcp), maximum temperature (tmax), and minimum temperature (tmin) for 3 melbourne airport stations. stations is the spatial component, (stations_sf as an sf object), meteo has the temporal component (meteo_ts as a tsibble object), climate_flat has both in a single joined table, and climate_mel is the cubble object. See climate_aus on the full dataset.

Usage

```
stations

stations_sf

meteo

meteo_ts

climate_flat

climate_mel
```

Format

An object of class tbl_df (inherits from tbl, data.frame) with 3 rows and 6 columns.

An object of class sf (inherits from tbl_df, tbl, data.frame) with 3 rows and 5 columns.

An object of class tbl_df (inherits from tbl, data.frame) with 30 rows and 5 columns.

An object of class tbl_ts (inherits from tbl_df, tbl, data.frame) with 30 rows and 5 columns.

An object of class tbl_df (inherits from tbl, data.frame) with 30 rows and 10 columns.

An object of class spatial_cubble_df (inherits from cubble_df, tbl_df, tbl, data.frame) with 3 rows and 7 columns.

See Also

climate_aus

Examples

```
cb <- make_cubble(
  spatial = stations, temporal = meteo,
  key = id, index = date, coords = c(long, lat)
)
identical(cb, climate_mel)
cb2 <- climate_flat |>
  as_cubble(key = id, index = date, coords = c(long, lat))
identical(cb, climate_mel)
```

---

unfold *Augment spatial component into the long (temporal) form*

Description

Some spatio-temporal transformation, i.e. glyph maps, uses both spatial and temporal variables. `unfold()` allows you to temporarily moves spatial variables into the long form for these transformations.

Usage

```
unfold(data, ...)

## S3 method for class 'spatial_cubble_df'
unfold(data, ...)

## S3 method for class 'temporal_cubble_df'
unfold(data, ...)
```

Arguments

`data`	a long cubble object
`...`	spatial variables to move into the long form, support tidyselect syntax

Value

a cubble object in the long form

Examples

```
climate_mel |> face_temporal() |> unfold(long, lat)
climate_mel |> face_temporal() |> unfold(dplyr::starts_with("l"))
```

update_cubble *Temporary update cubble if the sf class take precedent of cubble classes*

Description

When the data is already a cubble object but need update on attributes

Usage

```
update_cubble(data, key, index, coords, ...)

## S3 method for class 'spatial_cubble_df'
update_cubble(data, key = NULL, index = NULL, coords = NULL, ...)
```

Arguments

data, key, index, coords, ...	see make_cubble

[.spatial_cubble_df *Accessors to a cubble object*

Description

Accessors to a cubble object

Usage

```
## S3 method for class 'spatial_cubble_df'
data[i, j, drop = FALSE]

## S3 method for class 'temporal_cubble_df'
data[i, j, drop = FALSE]

## S3 replacement method for class 'spatial_cubble_df'
names(x) <- value

## S3 replacement method for class 'temporal_cubble_df'
names(x) <- value

## S3 replacement method for class 'cubble_df'
x[[i]] <- value
```

Arguments

data	an object of class spatial_cubble_df or temporal_cubble_df
i, j	row and column selector
drop	logical. If TRUE the result is coerced to the lowest possible dimension. The default is to drop if only one column is left, but **not** to drop if only one row is left.
x	data frame.
value	a suitable replacement value: it will be repeated a whole number of times if necessary and it may be coerced: see the Coercion section. If NULL, deletes the column if a single column is selected.

Details

For nested cubbles, [will return a cubble object if the key variable, thecoords variables, and the ts column all present. If the cubble object is also an sf object, the sticky select behavior on the sf column will preserve. For long cubbles, [will return a cubble object if the key and index variable both present. When a cubble can't be created and the data is not an sf class, [will always return a tibble, even with single index selection.

Examples

```
climate_mel[c(1:3, 7)] # a nested cubble
make_spatial_sf(climate_mel)[1:3] # an sf

long <- climate_mel |> face_temporal()
long[1:3] # a long cubble

climate_mel[1:3] # tibble
long[2:5] # tibble
climate_mel[1] # still tibble
long[1] # and still tibble
```

Index