Package ‘cubble’

March 22, 2023

Title A Vector Spatio-Temporal Data Structure for Data Analysis
Version 0.2.1
Description A spatiotemperal data object in a relational data structure to separate the recording of time variant/invariant variables.
License MIT + file LICENSE
Language au
Encoding UTF-8
LazyData true
RoxygenNote 7.2.3
Imports cli, dplyr, geosphere, ggplot2, glue, lubridate, ncdf4, pillar, rlang, sf, stringr, tibble, tidyselect, tsibble, vctrs
Suggests rmarkdown, knitr, testthat (>= 3.0.0), ozmaps, GGally, ggforce, purrr, stars, units, leaflet, plotly, crosstalk, concaveman, colorspace, vdiffr
VignetteBuilder knitr
Depends R (>= 3.5.0)
Config/testthat/edition 3
URL https://github.com/huizezhang-sherry/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 <huize.zhang@monash.edu>
Repository CRAN
Date/Publication 2023-03-22 00:10:02 UTC
R topics documented:

add_missing_prct .................................................. 2
as_cubble ............................................................. 3
climate_aus .......................................................... 5
climate_flat ........................................................... 6
climate_subset ....................................................... 7
extract_var .......................................................... 7
face_spatial .......................................................... 8
face_temporal ......................................................... 9
fill_gaps.cubble_df ................................................ 9
form ................................................................. 10
geom_glyph .......................................................... 11
get_centroid .......................................................... 14
match_sites ........................................................... 15
prcp_aus .............................................................. 16
prep_edges ............................................................ 17
rename_key ............................................................ 17
river ................................................................. 18
simplify_sf ............................................................ 18
slice_head.cubble_df ............................................... 19
slice_nearby .......................................................... 20
strip_rowwise .......................................................... 21
switch_key .............................................................. 21
tmax_hist ............................................................. 22
unfold ................................................................. 22

Index 24

---

add_missing_prct  
Compute missing summary

Description

Compute missing summary

Usage

add_missing_prct(data, vars)

Arguments

data a cubble object
vars variables to compute percentage missing (support tidyselect)

Details

• add_missing_prct() computes the percentage of missing for the selected variables
Value

a cubble object with additional columns VAR_missing

Examples

climate_aus %>% add_missing_prct(vars = prcp)

---

**as_cubble**  
*The constructor for the cubble class*

**Description**

The constructor for the cubble class

**Usage**

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

```r
## S3 method for class 'list'  
as_cubble(data, key, index, coords, by = NULL, output = "auto-match", ...)
```

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

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

```r
## S3 method for class 'cubble_df'  
print(
  x,  
  width = NULL,  
  ...,  
  n_extra = NULL,  
  n = NULL,  
  max_extra_cols = NULL,  
  max_footer_lines = NULL
)
```

```r
## S3 method for class 'cubble_df'  
tbl_sum(x)
```

```r
is_cubble(data)
```

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

- **data**: the object to be created or tested as cubble
- **key**: the spatial identifier
- **index**: the time identifier
- **coords**: the coordinates that characterise the spatial dimension
- **...**: a list object to create new cubble
- **by**: only used in `as_cubble.list()` to specify the linking key between spatial and temporal data
- **output**: either "all" or "unmatch", whether to output all or a list of unmatched summary

- **x**, **width**, **n_extra**, **n**, **max_extra_cols**, **max_footer_lines**
  
  see pillar tbl-format.R

Value

- a cubble object
- a cubble object
- a cubble object
- a TRUE/FALSE predicate
- a cubble object

Examples

# Declaimer: to make the examples easier, here we first `climate_flat` into different classes and show how they can be casted into a cubble. This is to demonstrate if your data come in one of the classes, it can be directly cast into a cubble. By no mean you need to first transform your data into any of the following class and then cast it to cubble.

# If the data is in a tibble:
climate_flat %>% as_cubble(key = id, index = date, coords = c(long, lat))

# If the spatial and temporal information are in two separate tables:
library(dplyr)
spatial <- climate_flat %>% select(id:wmo_id) %>% distinct()
temporal <- climate_flat %>% select(id, date: tmin) %>% filter(id != "ASN00009021")
as_cubble(data = list(spatial = spatial, temporal = temporal),
          key = id, index = date, coords = c(long, lat))

# If the data is already in a rowwise_df:
dt <- climate_flat %>%
tidyr::nest(ts = date:tmin) %>%
dplyr::rowwise()
dt %>% as_cubble(key = id, index = date, coords = c(long, lat))

# If the data is already in a tsibble, only need to supply `coords`
dt <- climate_flat %>% tsibble::as_tsibble(key = id, index = date)
dt %>% as_cubble(coords = c(long, lat))
# If the data is in netcdf:
path <- system.file("ncdf/era5-pressure.nc", package = "cubble")
raw <- ncdf4::nc_open(path)
dt <- as_cubble(raw, vars = c("q", "z"))

climatemel_aus  
Australia climate data - 639 stations

Description

Daily measure on precipitation (prcp) maximum temperature (tmax), and minimum temperature (tmin) in 2020 for 639 stations. stations and climate are the separate spatial and temporal objects while climatemel_aus is the combined cubble object.

Usage

climatemel_aus

Format

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

Details

<table>
<thead>
<tr>
<th>id</th>
<th>station id</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat</td>
<td>latitude of the station</td>
</tr>
<tr>
<td>long</td>
<td>longitude of the station</td>
</tr>
<tr>
<td>elev</td>
<td>elevation of the station</td>
</tr>
<tr>
<td>name</td>
<td>station name</td>
</tr>
<tr>
<td>wmo_id</td>
<td>the world meteorological organisation (WMO) station number</td>
</tr>
<tr>
<td>ts</td>
<td>a list-column that nests all the time-wise measures: date, prcp, tmax, and tmin</td>
</tr>
</tbody>
</table>

See Also

climate_subset climate_flat

Examples

climatemel_aus %>% face_temporal() %>% face_spatial()
**climate_flat**

Australia climate data - 5 stations

**Description**

Daily measure on precipitation (prcp) maximum temperature (tmax), and minimum temperature (tmin) in 2020 for 5 stations.

**Usage**

climate_flat

stations

climate

**Format**

A tibble object with 155 rows and 10 columns

- **id** station id
- **lat** latitude of the station
- **long** longitude of the station
- **elev** elevation of the station
- **name** station name
- **wmo_id** the world meteorological organisation (WMO) station number
- **date** the date that prcp, tmax, and tmin recorded
- **prcp** precipitation
- **tmax** maximum temperature
- **tmin** minimum temperature

An object of class tbl_df (inherits from tbl.data.frame) with 5 rows and 6 columns.
An object of class tbl_df (inherits from tbl.data.frame) with 1830 rows and 5 columns.

**See Also**

climate_aus climate_subset

**Examples**

climate_flat %>% as_cubble(key = id, index = date, coords = c(long, lat))
climate_subset

Australia climate data - 30 stations

Description
Daily measure on precipitation (prcp) maximum temperature (tmax), and minimum temperature (tmin) in 2020 for 30 stations.

Usage
climate_subset

Format
A cubble object

id  station id
lat  latitude of the station
long longitude of the station
elev elevation of the station
name station name
wmo_id the world meteorological organisation (WMO) station number
ts a list-column that nests all the time-wise measures: date, prcp, tmax, and tmin

See Also
climate_aus climate_flat

Examples
climate_subset %>% face_temporal()

extract_var

Functions to extract NetCDF dimension and variables

Description
Functions to extract NetCDF dimension and variables

Usage
extract_var(data, vars)
extract_longlat(data)
extract_time(data)
Arguments

data a NetCDF file read in from ncdf4::nc_open()

vars variables to read, see the variables in your data with names(data$var)

Value

extracted netcdf4 components

Description

face_spatial() turns a long cubble back into a nest cubble and can be seen as the inverse operation of face_temporal(). The nested cubble identifies each row by key and is suitable for operations whose output doesn’t involve a time index.

Usage

face_spatial(data)

Arguments

data a long cubble object

Value

a cubble object in the nested form

Examples

cb_long <- climate_flat %>%
  as_cubble(key = id, index = date, coords = c(long, lat)) %>%
  face_temporal()

cb_long %>% face_spatial()
**face_temporal**

*Switch a cumble object into the long form*

**Description**

`face_temporal()` switches a cumble object into a long cumble, suitable for temporal operations. The long cumble uses the combination of key and index to identify each row and arranges each key as a separate group.

**Usage**

`face_temporal(data, col)`

**Arguments**

- **data**: a nested cumble object
- **col**: 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 cumble object in the nested form

**Examples**

```r
climate_flat %>%
  as_cumble(key = id, index = date, coords = c(long, lat)) %>%
  face_temporal()
```

**fill_gaps.cumble_df**

*tsibble methods implemented in cumble*

**Description**

See `fill_gaps`

**Usage**

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

**Arguments**

- `.data, ..., .full, .start, .end`
  - see tsibble documentation
Value

a cubble object

Description

Functions to extract cubble attributes

Usage

form(data)

is_long(data)

is_nested(data)

spatial(data)

key_vars(data)

key_data(data)

coords(data)

coord_x(data)

coord_y(data)

index(data)

Arguments

data an cubble object

Details

Apart from inheriting attributes names, row.names, and class from the underlying tibble, a cubble has its site identifier: key, temporal identifier, index, and spatial coordinate reference: coords.

If a cubble object is also a tsibble, then tsibble attributes (key, index, index2, interval) are also preserved and can be accessed via the relevant functions in the tsibble package. (NOT FULLY IMPLEMENTED)

Value

the name of cubble attributes
geom_glyph

Examples

```r
# extract attributes of a cubble object
form(climate_aus)
spatial(climate_aus) %>% head(5)
key_data(climate_aus) %>% head(5)
key_vars(climate_aus)
index(climate_aus)
coords(climate_aus)
coord_x(climate_aus)
coord_y(climate_aus)
```

Description

Create glyph map with ggplot2

Usage

```r
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
)
```

```r
geom_glyph_line(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...
)```
geom_glyph

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, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")
position  Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
... Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`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

```r
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)
```
get_centroid

Description

Find the centroid of cubble

find the convex hull that wraps around the cluster, make it a polygon, find the centroid of the polygon and finally, extract the x and y coordinate of each centroid:

Usage

get_centroid(data)

Arguments

data a cubble data object

Value

a cubble object
match_sites

Matching sites from two data sources

Description

The function includes both spatial and temporal matching. The spatial matching is based on the distance and the distance is calculated using the Vincenty formula assuming earth is sphere with a radius of 6371 km. The temporal matching first filters out the $n$ largest increases, determined by `temporal_n_highest`, in both datasets, constructs an interval of length `temporal_window` from one dataset and count the number that large increase from the other dataset falls into the interval constructed.

Usage

```r
match_sites(
major,
minor,
spatial_single_match = TRUE,
spatial_n_keep = 1,
spatial_dist_max = 10,
temporal_matching = TRUE,
temporal_by,
temporal_n_highest = 20,
temporal_independent,
temporal_window = 5,
temporal_min_match = 10
)
```

```r
match_spatial(
major,
minor,
spatial_single_match = TRUE,
spatial_n_keep = 1,
spatial_dist_max = 10
)
```

```r
match_postprocessing(major, minor, match_table)
```

```r
match_temporal(
major,
minor,
temporal_by,
temporal_n_highest = 20,
temporal_independent,
temporal_window = 5,
temporal_min_match = 10
)
```
Arguments

- **major**: The major dataset to match, every key in the major dataset will have a match, unless filtered by `dist_max`.
- **minor**: The dataset to match from.
- **spatial_single_match**: Whether each observation in the minor dataset is only allowed to be matched once, default to `TRUE`.
- **spatial_n_keep**: The number of matching to keep.
- **spatial_dist_max**: The maximum distance allowed between matched pair.
- **temporal_matching**: Whether to perform temporal matching.
- **temporal_by**: The variable used for temporal matching.
- **temporal_n_highest**: The number of highest peak used for temporal matching.
- **temporal_independent**: The dataset used to construct the temporal window, need to be the name of either major or minor.
- **temporal_window**: The temporal window allowed to fall in.
- **temporal_min_match**: The minimum number of peak matching for temporal matching.
- **match_table**: The spatial matching table.

Value

A cubble with matched pairs.

---

**prcp_aus**  
*Daily precipitation data from 2016 to 2020*

Description

Daily precipitation data from 2016 to 2020.

Usage

`prcp_aus`

Format

An object of class `cubble_df` (inherits from `rowwise_df`, `tbl_df`, `tbl`, `data.frame`) with 663 rows and 7 columns.
**prep_edges**

*A function to prepare edges data for tour display*

**Description**

A function to prepare edges data for tour display

**Usage**

```r
prep_edges(data, edges_col, color_col)
```

```r
## S3 method for class 'cubble_df'
prep_edges(data, edges_col, color_col = NULL)
```

```r
prep_data(data, cols)
```

```r
## S3 method for class 'cubble_df'
prep_data(data, cols = NULL)
```

**Arguments**

- `data`: a cubble object
- `edges_col`: the variable maps to edges colour
- `color_col`: the variable maps to point colour
- `cols`: the numerical column selected for a tour

**Value**

a list of edge linkage, edge color, and point color

---

**rename_key**

*Rename the key variable*

**Description**

Rename the key variable

**Usage**

```r
rename_key(data, ...)
```

**Arguments**

- `data`: a cubble
- `...`: argument passed to rename: NEW = OLD
**simplify_sf**

**Value**

a cubble object

---

<table>
<thead>
<tr>
<th>river</th>
<th>Australia river data</th>
</tr>
</thead>
</table>

**Description**

Australia river data

**Usage**

river

**Format**

An object of class cubble_df (inherits from rowwise_df, tbl_df, tbl, data.frame) with 71 rows and 5 columns.

---

**simplify_sf**

Find (multi)polygons with small area

**Description**

Find (multi)polygons with small area

**Usage**

simplify_sf(data, geom, area, point_threshold = 0.9, area_threshold = 0.9)

**Arguments**

data An sf object
gem The geometry column
area The area column if any
point_threshold The number of point threshold used to define small crumb
area_threshold The area size threshold used to define small crumb

**Value**

An sf object
	he data object with additional column crumb indicating whether the area is a "small crumb"
Slicing a cubble

Description

Slicing can be useful when the number of site is too large to be all visualised in a single plot. The slicing family in cubble wraps around the dplyr::slice() family to allow slicing from top and bottom, based on a variable, or in random.

Usage

```r
## S3 method for class 'cubble_df'
slice_head(.data, ..., n, prop, by)

## S3 method for class 'cubble_df'
slice_tail(.data, ..., n, prop, by)

## S3 method for class 'cubble_df'
slice_min(.data, order_by, ..., n, prop, by, with_ties, na_rm)

## S3 method for class 'cubble_df'
slice_max(.data, order_by, ..., n, prop, by, with_ties, na_rm)

## S3 method for class 'cubble_df'
slice_sample(.data, order_by, ..., n, prop, by, with_ties, na_rm)
```

Arguments

- `.data` a cubble object to slice
- `...`, `n`, `prop`, `by`
  - other arguments passed to the dplyr::slice()
- `order_by`, `with_ties`, `na_rm`
  - other arguments passed to the dplyr::slice()

Value

a cubble object

Examples

```r
# slice the first 50 stations from the top/ bottom
library(dplyr)
climate_aus |> slice_head(n = 50)
climate_aus |> slice_tail(n = 50)

# slice based on the max/ min of a variable
climate_aus |> slice_max(elev, n = 10)
```
climate_aus |> slice_min(lat, n = 10)

# random sample
climate_aus |> slice_sample(n = 10)

---

slice_nearby | Location-based slicing

Description

Location-based slicing

Usage

slice_nearby(data, coord, buffer, n)

Arguments

- **data**: the data to slice
- **coord**: the coordinate of used to slice nearby locations
- **buffer**: the buffer added to the coordinate for slicing
- **n**: the number of nearby points to slice, based on distance

Value

- a cubble object

Examples

# slice locations within 1 degree of (130E, 25S)
slice_nearby(climate_aus, coord = c(130, -25), buffer = 3)

# slice the 5 closest location to (130E, 25S)
slice_nearby(climate_aus, coord = c(130, -25), n = 5)
### strip_rowwise

**Description**

Remove the rowwise grouping of a cubble

**Usage**

```
strip_rowwise(data)
```

**Arguments**

- `data` [a cubble object](#)

**Value**

[a cubble object](#)

**Examples**

```r
code
library(dplyr)
climate_aus |> mutate(.id = row_number())
climate_aus |> strip_rowwise() |> mutate(.id = row_number())
```

### switch_key

**Description**

`switch_key()` allows you select a new variable in the data to become the key. This can be used to create hierarchical data where one variable is nested in another.

**Usage**

```
switch_key(data, key)
```

**Arguments**

- `data` [a cubble object](#), can be either long or nested cubble
- `key` [the new key](#)
Value

a cubble object

Examples

```
library(ggplot2)
library(dplyr)
# create an artificial cluster for stations
set.seed(1234)
cb <- climate_flat %>%
  as_cubble(key = id, index = date, coords = c(long, lat)) %>%
  mutate(cluster = sample(1:3, 1))

# switch the key to cluster
cb_hier <- cb %>% switch_key(cluster)
```

tmax_hist  

*Victoria and Tasmania daily maximum temperature for 1970 - 1975 and 2016 - 2020*

Description

Victoria and Tasmania daily maximum temperature for 1970 - 1975 and 2016 - 2020

Usage

```
tmax_hist
```

Format

An object of class `cubble_df` (inherits from `rowwise_df, tbl_df, tbl, data.frame`) with 39 rows and 7 columns.

```
unfold  

*Move spatial variables into the long 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, ...)
```
**Arguments**

- **data**
  - a long cubble object
- ... spatial variables to move into the long form

**Value**

- a cubble object in the long form

**Examples**

```r
cb <- climate_flat |> 
  as_cubble(key = id, index = date, coords = c(long, lat)) |>
  face_temporal()

# unfold long and lat
cb_mig <- cb |> unfold(long, lat)

# unfold is not memorised by cubble:
# if you switch to the nested cubble and then switch back,
# long and lat will not be preserved
cb_mig |> face_spatial() |> face_temporal()
```
Index

* datasets
  climate_aus, 5
  climate_flat, 6
  climate_subset, 7
  prcp_aus, 16
  river, 18
  tmax_hist, 22
add_missing_prct, 2
aes(), 12
as_cubble, 3
borders(), 13
climate (climate_flat), 6
climate_aus, 5
climate_flat, 6
climate_subset, 7
coord_x (form), 10
coord_y (form), 10
coords (form), 10
cubble (as_cubble), 3
dplyr::slice(), 19
extract_longlat (extract_var), 7
extract_time (extract_var), 7
extract_var, 7
face Spatial, 8
face_temporal, 9
fill_gaps, 9
fill_gaps.cubble_df, 9
form, 10
fortify(), 12
geom_glyph, 11
geom_glyph_box (geom_glyph), 11
geom_glyph_line (geom_glyph), 11
gt get_centroid, 14
ggplot(), 12
identity, 13
index (form), 10
is_cubble (as_cubble), 3
is_long (form), 10
is_nested (form), 10
key_data (form), 10
key vars (form), 10
layer(), 13
match_postprocessing (match_sites), 15
match_sites, 15
match_spatial (match_sites), 15
match_temporal (match_sites), 15
prcp_aus, 16
prep_data (prep_edges), 17
prep_edges, 17
print.cubble_df (as_cubble), 3
rel, 13
rename_key, 17
resolution, 13
river, 18
simplify_sf, 18
slice_head.cubble_df, 19
slice_max.cubble_df
  (slice_head.cubble_df), 19
slice_min.cubble_df
  (slice_head.cubble_df), 19
slice_nearby, 20
slice_sample.cubble_df
  (slice_head.cubble_df), 19
slice_tail.cubble_df
  (slice_head.cubble_df), 19
spatial (form), 10
stations (climate_flat), 6
strip_rowwise, 21
switch_key, 21
INDEX

tbl_sum.cubble_df (as_cubble), 3
	tmax_hist, 22

unfold, 22