Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

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

H. Sherry Zhang
Monash University

Dianne Cook
Monash University

Ursula Laa
University of Natural Resources and Life Sciences

Nicolas Langrené
BNU-HKBU
United International College

Patricia Menéndez
Monash University

Abstract

Multivariate spatio-temporal data refers to multiple measurements taken across space and time. For many analyses, spatial and time components can be separately studied: for example, to explore the temporal trend of one variable for a single spatial location, or to model the spatial distribution of one variable at a given time. However for some studies, it is important to analyze different aspects of the spatio-temporal data simultaneously, like for instance, temporal trends of multiple variables across locations. In order to facilitate the study of different portions or combinations of spatio-temporal data, we introduce a new class, `cubble`, with a suite of functions enabling easy slicing and dicing on different spatio-temporal components. The proposed `cubble` class ensures that all the components of the data are easy to access and manipulate while providing flexibility for data analysis. In addition, the **cubble** package facilitates visual and numerical explorations of the data while easing data wrangling and modelling. The `cubble` class and the tools implemented in the package are illustrated with different examples of Australian climate data.

Keywords: spatial, temporal, spatio-temporal, R, environmental data, exploratory data analysis.

1. Introduction

Spatio-temporal data has a spatial component referring to the location of each observation and a temporal component that is recorded at regular or irregular time intervals. It may also in-

clude multiple variables measured at each spatial and temporal values. With spatio-temporal data, one can fix the time to explore the spatial features of the data, fix the spatial location/s to explore temporal aspects, or dynamically explore the space and time simultaneously.

In order to computationally explore the spatial, temporal and spatio-temporal faces of such data, the data needs to be stored and represented under a specific data object that allows the user to query, group and dissect all the data faces.

The Comprehensive R Archive Network (CRAN) task view SpatioTemporal (Pebesma and Bivand 2022) gathers information about R packages designed for spatio-temporal data and it has a section on *Representing data* that lists existing spatio-temporal data representations used in R. Among them, the **spacetime** package (Pebesma 2012) implements four S4 classes to handle spatio-temporal data with different spatio-temporal layouts (full grid, sparse grid, irregular, and trajectory). The **stars** package (Pebesma 2021) implements an S3 class built from dense arrays.

However, the data representation implemented in those packages might present certain challenges when applying the principles of tidy data (Wickham 2014) for data analysis. The concept of tidy data is based on three principles regarding how data should be organized in tables to facilitate easier analysis: 1) one observation a row, 2) one variable a column, and 3) one type of observation a table. The third principle of tidy data is particularly relevant for spatio-temporal data since these data are naturally observed at different units: the spatial locations and the temporal units. While the tidyverse suite of R packages implements data wrangling and visualization tools primarily focused on working with single tables, there are not many tools available for handling relational data specifically for spatio-temporal data. This motivates a new design to organise spatio-temporal data in a way that would make data wrangling, visualizing and analyzing easier.

This paper presents a new R package, **cubble**, which implements a new cubble class to organize spatial and temporal variables as two forms of a single data object so that they can be wrangled separately or combined, while being kept synchronized. Among the four spacetime layouts in Pebesma (2012), the `cubble` class can handle the full grid layout and the sparse grid layout. The software is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=cubble`.

The rest of the paper is organized as follows: Section 2 presents the main design and functionality of the **cubble** package. Section 3 explains how the **cubble** package deals with more advanced considerations, including data matching and how the package fits with existing static and interactive visualization tools. Moreover we also illustrate how the **cubble** package deals with spatio-temporal data transformations. Section 4 uses primarily Australian weather station data as examples to demonstrate the use of the package. An example of how the **cubble** package handles Network Common Data Form (NetCDF) data is also provided. Section 5 discuss the paper contributions and future directions.

2. The cubble package

The cubble class includes two subclasses: the spatial cubble and the temporal cubble, which can be pivoted back and forth to focus on the two aspects of the spatio-temporal data, as illustrated in Figure 1. This section provides an overview of the **cubble** package, including the cubble class and its attributes, class creation and coercion, a summary of implemented

functionality, the compatibility with other spatial and temporal packages (**sf** and **tsibble**), and a comparison with other spatio-temporal packages (**stars** and **sftime**).

2.1. The cubble class

The cubble class is an S3 class built on tibble that allows the spatio-temporal data to be wrangled in two forms (subclasses):

- a spatial cubble with class `c("spatial_cubble_df", "cubble_df")`
- a temporal cubble with class `c("temporal_cubble_df", "cubble_df")`

In a spatial cubble object, spatial variables are organised as columns and temporal variables are nested within a specialised `ts` column. For example, the spatial cubble object, `cb_spatial` printed below, contains weather records of three airport stations from the Global Historical Climatology Network Daily (GHCND) database (Menne *et al.* 2012). In this case, the spatial cubble is convenient for wrangling the spatial variables:

```
R> cb_spatial
```

```
# cubble:   key: id [3], index: date, nested form
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], Missing CRS!
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name              wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl> <list>
1 ASN00086038  145. -37.7  78.4 essendon airport   95866 <tibble [10 x 4]>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble [10 x 4]>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble [10 x 4]>
```

In a temporal cubble, temporal variables are expanded in the long form and spatial variables are stored as a data attribute. The temporal cubble object, `cb_temporal`, contains the same spatio-temporal data as the spatial cubble object, `cb_spatial`, but in a structure that is easier for temporal analysis:

```
R> cb_temporal
```

```
# cubble:   key: id [3], index: date, long form
# temporal: 2020-01-01 -- 2020-01-10 [1D], no gaps
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  id          date        prcp  tmax  tmin
  <chr>       <date>     <dbl> <dbl> <dbl>
1 ASN00086038 2020-01-01     0  26.8  11
2 ASN00086038 2020-01-02     0  26.3  12.2
3 ASN00086038 2020-01-03     0  34.5  12.7
4 ASN00086038 2020-01-04     0  29.3  18.8
5 ASN00086038 2020-01-05    18  16.1  12.5
# i 25 more rows
```

The cubble attributes

Both cubble objects inherit tibble's attributes (which originates from data frames): `class`, `row.names`, and `names`. Additionally, both have three specialised attributes: `key`, `index`, and `coords`. `key` and `index` are used as introduced in the **tsibble** package (Wang *et al.* 2020). In cubble, the `key` attribute identifies the row in the spatial cubble (given the internal use of `tidyr::nest()` for nesting), and when combined with the `index` argument, it identifies the row in the temporal cubble. Currently, cubble only supports one variable as the key, and the accepted temporal classes for `index` includes the base R classes `Date`, `POSIXlt`, `POSIXct`, as well as tsibble's `yearmonth`, `yearweek`, and `yearquarter` classes. The `coords` attribute represents an ordered pair of coordinates that can be either an unprojected pair of longitude and latitude, or a projected easting and northing value. Moreover, temporal cubbles have a special attribute called `spatial` to store the spatial variables. Shortcut functions are available to extract attributes from the temporal cubble object, for example, `spatial()` for extracting spatial variables:

```
R> spatial(cb_temporal)
```

```
# A tibble: 3 x 6
  id           long   lat  elev name              wmo_id
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl>
1 ASN00086038  145. -37.7  78.4 essendon airport   95866
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870
3 ASN00086282  145. -37.7 113.  melbourne airport  94866
```

2.2. Creation and coercion

The spatial and temporal aspect of spatio-temporal data are often stored separately in the database. For climate data, analysts may initially receive station metadata and then query the time series based on the metadata. A (spatial) cubble object can be constructed from separate spatial and temporal tables using the function `make_cubble()`. The three attributes `key`, `index`, and `coords` need to be specified in creation. The following code creates a spatial cubble from its spatial component, `stations` and temporal component `meteo`:

```
R> make_cubble(spatial = stations, temporal = meteo,
+              key = id, index = date, coords = c(long, lat))
```

```
# cubble:   key: id [3], index: date, nested form
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], Missing CRS!
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name              wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl> <list>
1 ASN00086038  145. -37.7  78.4 essendon airport   95866 <tibble [10 x 4]>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble [10 x 4]>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble [10 x 4]>
```

Other R spatio-temporal objects can be coerced into a `cubble` object with the function `as_cubble()`. This includes a joined `tibble` or `data.frame` object, a NetCDF object, a `stars` object (Pebesma 2021), and a `sftime` object (Teickner *et al.* 2022). In the example below, the spatial cubble object is created from `climate_flat`, which combines the previous `stations` and `meteo` into a single tibble object:

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

# cubble:   key: id [3], index: date, nested form
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], Missing CRS!
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name              wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl> <list>
1 ASN00086038  145. -37.7  78.4 essendon airport   95866 <tibble [10 x 4]>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble [10 x 4]>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble [10 x 4]>
```

2.3. Functions and methods

The **cubble** package has several functions implemented for data wrangling and to facilitate data analysis as summarized in Table 1. In addition, for each of the three cubble classes there are number of methods implemented that facilitates the handling of the data as shown in Table 2. In particular, the `cubble_df` class handles methods that behave consistently in both spatial and temporal cubble. When the method works differently internally on the spatial and temporal cubble, it is implemented separately in `spatial_cubble_df` and `temporal_cubble_df`.

Table 1: An overview of functions implemented in the **cubble** package, categorised into base R, tidyverse, and cubble functions.

Category	Functions
base R	`[`, `[[<-`, `names<-`
tidyverse	`dplyr_row_slice`, `dplyr_col_modify`, `dplyr_reconstruct`, `select`, `mutate`, `arrange`, `filter`, `group_by`, `ungroup`, `summarise`, `select`, `slice`, `rowwise`, `rename`, `bind_rows`, `bind_cols`, `relocate`, `type_sum`, the slice family (`slice_head`, `slice_tail`, `slice_max`, `slice_min`, `slice_sample`) and the join family (`left_join`, `right_join`, `inner_join`, `full_join`, `anti_join`, `semi_join`)
cubble	`as_cubble`, `cubble`, `make_cubble`, `check_key`, `face_temporal`, `face_spatial`, `unfold`, `key`, `key_vars`, `key_data`, `index`, `index_var`, `coords`, `spatial`, `match_sites`, `match_spatial`, `match_temporal`, `geom_glyph`, `geom_glyph_box`, `geom_glyph_line`, `make_spatial_sf`, `make_temporal_tsibble`, `fill_gaps`, and `scan_gaps`

Table 2: An overview of the methods implemented in the three `cubble` classes. Methods are implemented in the `cubble_df` class when they behave consistently across the spatial and temporal cubble; otherwise, they are implemented separately.

Class	Methods
`cubble_df`	`[[<-, dplyr_col_modify, key_data, key_vars, key, print`
`spatial_cubble_df`	`[, names<-, tbl_sum, dplyr_reconstruct, dplyr_row_slice, face_spatial, face_temporal, unfold, arrange, rename, rowwise, group_by, ungroup, select, spatial, summarise, unfold, update_cubble`
`temporal_cubble_df`	`[, names<-, tbl_sum, arrange, dplyr_reconstruct, dplyr_row_slice, face_spatial, face_temporal, unfold, fill_gaps, group_by, ungroup, rename, rowwise, scan_gaps, select, spatial, summarise, tbl_sum, bind_rows, bind_cols, update_cubble`

The pair of cubble verbs, `face_temporal()` and `face_spatial()`, pivots the cubble object between its two forms or faces, as illustrated in Figure 1. The code applies `face_temporal()` on the spatial cubble, `cb_spatial`, introduced in Section 2.1 to get a temporal cubble:

```
R> face_temporal(cb_spatial)

# cubble:   key: id [3], index: date, long form
# temporal: 2020-01-01 -- 2020-01-10 [1D], no gaps
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  id          date        prcp  tmax  tmin
  <chr>       <date>     <dbl> <dbl> <dbl>
1 ASN00086038 2020-01-01     0  26.8  11
2 ASN00086038 2020-01-02     0  26.3  12.2
3 ASN00086038 2020-01-03     0  34.5  12.7
4 ASN00086038 2020-01-04     0  29.3  18.8
5 ASN00086038 2020-01-05    18  16.1  12.5
# i 25 more rows
```

Both verbs are the exact inverse of each other and apply both functions on a cubble object will result in the object itself:

```
R> face_spatial(face_temporal(cb_spatial))

# cubble:   key: id [3], index: date, nested form
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], Missing CRS!
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name              wmo_id ts
```

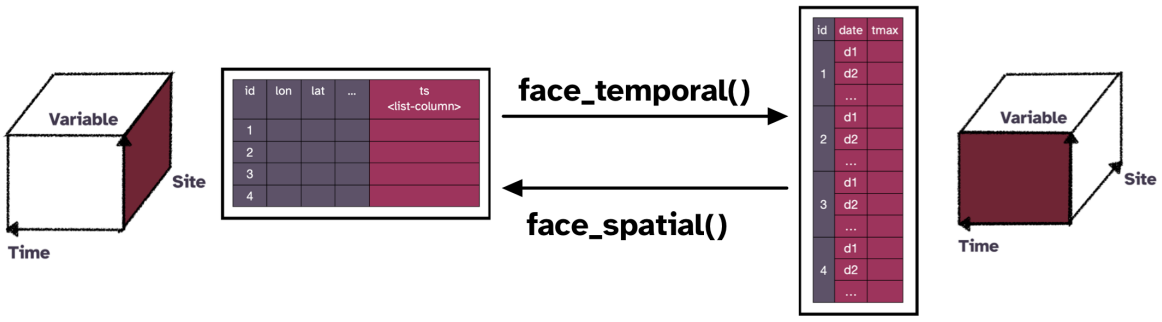

Figure 1: Illustration of main functions. To focus on he temporal variables `face_temporal()` converts a spatial cubble into a temporal cubble. To focus on the spatial variables `face_spatial()` transforms a temporal cubble into a spatial cubble. This pivoting makes it easy to separately do spatial or temporal analysis.

```
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl> <list>
1 ASN00086038  145. -37.7  78.4 essendon airport   95866 <tibble [10 x 4]>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble [10 x 4]>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble [10 x 4]>
```

To enable operations involve both spatial and temporal variables, the function `unfold` incorporates spatial variables into the temporal cubble. Below is an example to include the coordinate columns (`long` and `lat`) into `cb_temporal` to prepare the data for a glyph map transformation, which will be discussed in Section 3.3.

```
R> cb_temporal |> unfold(long, lat)

# cubble:   key: id [3], index: date, long form
# temporal: 2020-01-01 -- 2020-01-10 [1D], no gaps
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  id          date        prcp  tmax  tmin  long   lat
  <chr>       <date>     <dbl> <dbl> <dbl> <dbl> <dbl>
1 ASN00086038 2020-01-01     0  26.8  11    145. -37.7
2 ASN00086038 2020-01-02     0  26.3  12.2  145. -37.7
3 ASN00086038 2020-01-03     0  34.5  12.7  145. -37.7
4 ASN00086038 2020-01-04     0  29.3  18.8  145. -37.7
5 ASN00086038 2020-01-05    18  16.1  12.5  145. -37.7
# i 25 more rows
```

2.4. Compatibility with tsibble and sf

Analysts often have their preferred spatial or temporal data structure for spatial or temporal analysis, which they may wish to continue using for spatio-temporal analysis. With `cubble`, analysts can incorporate the `tsibble` class in a temporal cubble and the `sf` class in a spatial cubble.

Using a tsibble object as the temporal component

The `key` and `index` arguments in a `cubble` object corresponds to the tsibble counterparts and they can be safely omitted, if the temporal component is a tsibble object (`tbl_ts`). The `tsibble` class (`tbl_ts`) from the input will be carried over to the temporal cubble, indicated by the `[tsibble]` in the header and in the object class:

```
R> class(meteo_ts)

[1] "tbl_ts"     "tbl_df"     "tbl"        "data.frame"

R> ts_spatial <- make_cubble(
+   spatial = stations, temporal = meteo_ts, coords = c(long, lat))
R> (ts_temporal <- face_temporal(ts_spatial))

# cubble:   key: id [3], index: date, long form, [tsibble]
# temporal: 2020-01-01 -- 2020-01-10 [1D], no gaps
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  id          date        prcp  tmax  tmin
  <chr>       <date>     <dbl> <dbl> <dbl>
1 ASN00086038 2020-01-01     0  26.8  11
2 ASN00086038 2020-01-02     0  26.3  12.2
3 ASN00086038 2020-01-03     0  34.5  12.7
4 ASN00086038 2020-01-04     0  29.3  18.8
5 ASN00086038 2020-01-05    18  16.1  12.5
# i 25 more rows

R> class(ts_temporal)

[1] "temporal_cubble_df" "cubble_df"          "tbl_ts"
[4] "tbl_df"             "tbl"                "data.frame"
```

Methods applied to tsibble objects (`tbl_ts`) can also be applied to the temporal cubble objects, for example, checking whether the data contain temporal gaps:

```
R> ts_temporal |> has_gaps()

# A tibble: 3 x 2
  id          .gaps
  <chr>       <lgl>
1 ASN00086038 FALSE
2 ASN00086077 FALSE
3 ASN00086282 FALSE
```

The temporal component of a created temporal cubble can include class `tbl_ts` to also be a tsibble object `tsibble` object using `make_temporal_tsibble()`. See the code example below using the `cb_temporal` object, created in Section 2.2:

```
R> cb_temporal |> make_temporal_tsibble()

# cubble:   key: id [3], index: date, long form, [tsibble]
# temporal: 2020-01-01 -- 2020-01-10 [1D], no gaps
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  id          date        prcp  tmax  tmin
  <chr>       <date>     <dbl> <dbl> <dbl>
1 ASN00086038 2020-01-01     0  26.8  11
2 ASN00086038 2020-01-02     0  26.3  12.2
3 ASN00086038 2020-01-03     0  34.5  12.7
4 ASN00086038 2020-01-04     0  29.3  18.8
5 ASN00086038 2020-01-05    18  16.1  12.5
# i 25 more rows
```

Using an sf object as the spatial component

Similarly, the spatial component of a cubble object can be an `sf` object and if the `coords` argument is omitted, it will be calculated from the `sf` geometry. The sf status is signalled by the `[sf]` label in the cubble header:

```
R> (sf_spatial <- make_cubble(
+   spatial = stations_sf, temporal = meteo,
+   key = id, index = date))

# cubble:   key: id [3], index: date, nested form, [sf]
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], WGS 84
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           elev name   wmo_id  long   lat            geometry ts
  <chr>       <dbl> <chr>   <dbl> <dbl> <dbl>         <POINT [°]> <list>
1 ASN00086038  78.4 essen~  95866  145. -37.7 (144.9066 -37.7276) <tibble>
2 ASN00086077  12.1 moora~  94870  145. -38.0   (145.0964 -37.98) <tibble>
3 ASN00086282 113.  melbo~  94866  145. -37.7 (144.8321 -37.6655) <tibble>

R> class(sf_spatial)

[1] "spatial_cubble_df" "cubble_df"         "sf"
[4] "tbl_df"            "tbl"               "data.frame"
```

This allows applying functions from the `sf` package to a cubble object, for example, to handle coordinate transformation with `st_transform()`:

```
R> sf_spatial |> sf::st_transform(crs = "EPSG:3857")

# cubble:   key: id [3], index: date, nested form, [sf]
# spatial:  [16122635.6225205, -4576600.8687746, 16152057.3639371,
#   -4532279.35567565], WGS 84
```

```
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           elev name   wmo_id  long   lat            geometry ts
  <chr>       <dbl> <chr>   <dbl> <dbl> <dbl>         <POINT [°]> <list>
1 ASN00086038  78.4 essen~  95866  145. -37.7 (16130929 -4541016) <tibble>
2 ASN00086077  12.1 moora~  94870  145. -38.0 (16152057 -4576601) <tibble>
3 ASN00086282 113.  melbo~  94866  145. -37.7 (16122636 -4532279) <tibble>
```

The spatial component of a created cubble can also be an `sf` object using `make_spatial_sf()`:

```
R> cb_spatial |> make_spatial_sf()
```

```
# cubble:   key: id [3], index: date, nested form, [sf]
# spatial:  [144.8321, -37.98, 145.0964, -37.6655], WGS 84
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name   wmo_id ts                   geometry
  <chr>       <dbl> <dbl> <dbl> <chr>   <dbl> <list>           <POINT [°]>
1 ASN00086038  145. -37.7  78.4 essen~  95866 <tibble> (144.9066 -37.7276)
2 ASN00086077  145. -38.0  12.1 moora~  94870 <tibble>   (145.0964 -37.98)
3 ASN00086282  145. -37.7 113.  melbo~  94866 <tibble> (144.8321 -37.6655)
```

2.5. Comparison to other spatio-temporal classes

In R, there are other existing spatio-temporal data structures and this section compares and contrasts **cubble** with other existing alternatives, specifically **stars** and **sftime**. The **stars** package (Pebesma 2021) uses an array structure, as oppose to tibble, to represent multivariate spatio-temporal data. While both **stars** and **cubble** support vector and raster data, it is a matter of choice on which structure to use given the application. Analysts working on satellite imageries may prefer the array structure in **stars**, while others originally working with spatio-temporal data in 2D data frames may find **cubble** easier to adopt from their existing computing workflow.

The **sftime** package (Teickner *et al.* 2022) also builds from a tibble object and its focus is on handling irregular spatio-temporal data. This means **sftime** can also handle full space-time grids and sparse space-time layouts represented in **cubble**. However, **cubble** uses nesting to avoid storing spatial variables repetitively at each timestamp. This provides memory efficiency when data is observed frequent, i.e. daily or sub-daily, or the spatial geometry is computationally expensive to store repeatedly, i.e. polygons or multipolygons. Consider the `climate_aus` data in the **cubble** package with 639 stations observed daily throughout the year 2020. In that case, the `sftime` object is approximately 14 times larger than the corresponding `cubble` object (118 MB vs. 8.5 MB).

3. Other features and considerations

3.1. Data fusion and matching

Matching time series from two lists is a common task in spatio-temporal data analysis. In cubble, matching based on distance and time series features can be performed using the functions `match_spatial()` and `match_temporal()`. The `match_spatial()` function finds the matched pairs in two cubble objects based on distance between sites:

```
match_spatial(<cubble_obj1>, <cubble_obj2>, ...)
```

Two arguments are available to control the matching: the argument `spatial_n_group` specifies the number of paired groups to return and the argument `spatial_n_each` specifies the number of matches for each item in the first cubble object (default to 1 for one-to-one matching).

The function `match_temporal()` takes the outputs from spatial matching and calculates a similarity score of the time series between spatially matched pairs. The temporal matching requires two identifiers: one for separating each spatially matched group: `match_id` and one for separating the two data sources: `data_id`. Matching between different variables can be specified using the `temporal_by` argument, similar to the `by` syntax from dplyr's `*_join`.

```
match_temporal(
  <obj_from_match_spatial>,
  data_id = ... , match_id = ...,
  temporal_by = c("..." = "...")
)
```

The similarity score between two time series is calculated using a matching function, which can be customised by the analysts based on the time series feature relevant to match. The matching function takes two time series as a list and returns a single numerical score. By default, cubble uses a simple peak matching algorithm (`match_peak`) to count the number of peaks in two time series that fall within a specified temporal window.

3.2. Interactive graphics

The cubble workflow can be easily incorporated into an interactive graphics pipeline (e.g., Buja *et al.* (1988); Buja *et al.* (1996); Sutherland *et al.* (2000); Xie *et al.* (2014); Cheng *et al.* (2016)). This section describes the linking between a map and multiple time series in a `cubble` object using the package **crosstalk** (Cheng and Sievert 2021), as illustrated in Figure 2. The spatial and temporal cubble can be constructed as a shared crosstalk object to create a link between a map and a time series plot. For instance, when a user selects a location on the map as shown on panel (a), the corresponding site is highlighted. This selection activates a row in the spatial cubble, which is then connected to the temporal cubble, resulting in the selection of all observations with the same ID as depicted in panel (b). Consequently the temporal cubble highlights the corresponding series in the time series plot displayed in panel (c). The linking can also be initiated from the time series plot by selecting points on the time series graph. This action selects rows with the same ID in the temporal cubble and the corresponding row in the spatial cubble so that points can be highlight on the map.

3.3. Spatio-temporal transformations

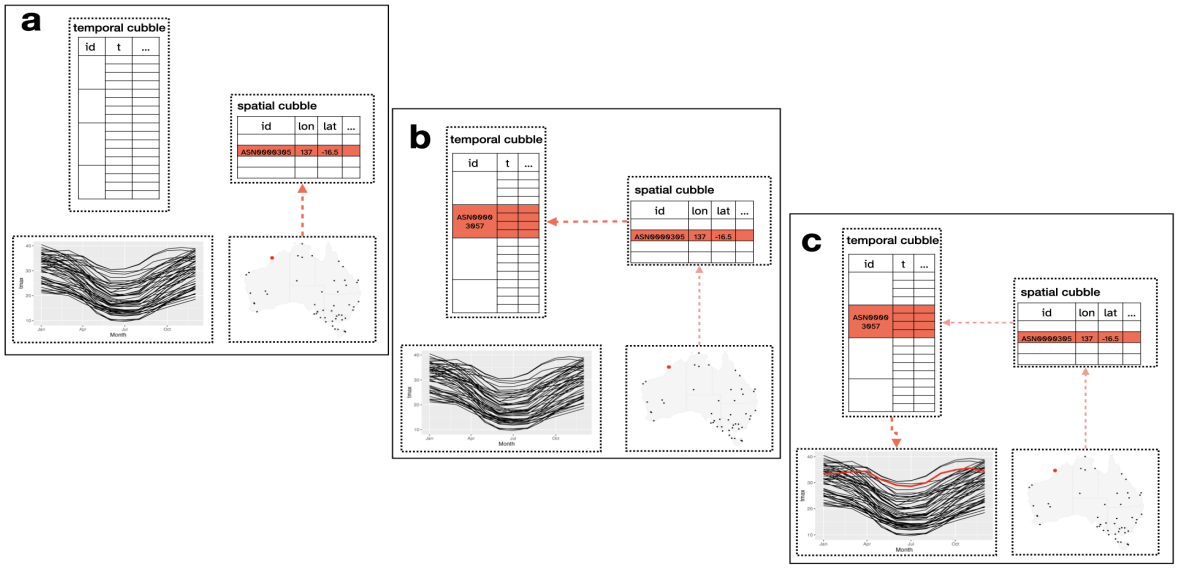

Figure 2: Linking between multiple plots is made possible by shared `crosstalk` objects. When a station is selected on the map (a), the corresponding row in the spatial `cubble` will be activated. This will activate the row with the same id in the temporal `cubble` (b) to trigger an update of the line plot (c). The cubble package makes linking between spatial and temporal plots easy.

Visualizing spatio-temporal data is important for exploring and understanding the data at hand, aiding in decision-making, and facilitating effective communication. To collectively visualize spatial and temporal information, several options are available such as: faceted maps across time, map animations, or interactive graphics that link maps and time series plots among others. Faceted maps and spatio-temporal animations, while commonly used, may be difficult to compare across time and space since users need to inspect multiple facets or frames to find changes in time or space. The glyph map (Wickham *et al.* 2012) resolves this issue by imposing the time series onto the map as a glyph through a coordinate transformation. The transformation uses linear algebra to convert the temporal coordinates (minor coordinates) into the spatial coordinates (major coordinates) and is implemented in the package `GGally` (Schloerke *et al.* 2021). The **cubble** package provides a ggproto implementation to create glyph maps, `geom_glyph()` and it takes four required aestheticsf : `x_major`, `y_major`, `x_minor`, and `y_minor`:

```
data |>
  ggplot() +
  geom_glyph(aes(x_major = ..., x_minor = ...,
                 y_major = ..., y_minor = ...))
```

Other useful controls to modify the glyph map that can be include are:

- the implementation of a polar coordinate glyph maps with `polar = TRUE`,
- the adjustment of the glyph size arguments using `width` and `height`,

- a transformation relative to the all series (`global_rescale` defaults to `TRUE`) or each single series, and
- the XX of the reference boxes and lines with `geom_glyph_box()` and `geom_glyph_line()`.

4. Applications

Five examples are chosen to illustrate different aspects of the **cubble** package: creating a `cubble` object from two Coronavirus (COVID) data tables with the challenge of having different location names, using spatial transformations to make a glyph map of seasonal temperature changes, matching river level data with weather station records to analyze water supply, reading NetCDF format data to replicate a climate reanalysis plot, and demonstrating the workflow to create complex interactive linked plots.

4.1. Victoria COVID spatio-temporal incidence and spread

Since the start of the COVID-19 pandemic, the Victoria State Government in Australia has been providing daily COVID-19 case counts per Local Government Area (LGA). This data can be combined with map polygon data, available from the Australian Bureau of Statistics (ABS), to visualize COVID-19 incidence and spread. The COVID-19 count data (`covid`) and the LGA information (`lga`) are available in the **cubble** package as a `tsibble` object and an `sf` object, respectively. This example illustrates the creation of a cubble object using real-world data, where analysts may deal with slightly varied encoding of the spatial units used across different institutes. Cubble can flag this discrepancies when creating the object and guide analysts to further investigate this issue.

The function `make_cubble()` is used to create a cubble object from separate spatial and temporal components. When the spatial identifier in the two data has different names, they can be handled by the `by` argument:

```
R> cb <- make_cubble(lga, covid, by = c("lga_name_2018" = "lga"))

Warning: st_centroid assumes attributes are constant over geometries

! Some sites in the spatial table don't have temporal information

! Some sites in the temporal table don't have spatial information

! Use `check_key()` to check on the unmatched key
The cubble is created only with sites having both spatial and
temporal information
```

The difference in LGA naming between both data sets triggers cubble to issue a warning, alerting the user to this discrepancy. The warning message suggests there are some differences between the LGA encoding used by Victoria government and ABS and prompts analysts to check the mismatch using `check_key()`, which takes the same inputs as `make_cubble()`, but returns a summary of key matches between the spatial and temporal input data.

```
R> (check_res <- check_key(
+   spatial = lga, temporal = covid,
+   by = c("lga_name_2018" = "lga")
+ ))

$paired
# A tibble: 78 x 2
  spatial        temporal
  <chr>          <chr>
1 Alpine (S)     Alpine (S)
2 Ararat (RC)    Ararat (RC)
3 Ballarat (C)   Ballarat (C)
4 Banyule (C)    Banyule (C)
5 Bass Coast (S) Bass Coast (S)
# i 73 more rows

$potential_pairs
# A tibble: 2 x 2
  spatial             temporal
  <chr>               <chr>
1 Kingston (C) (Vic.) Kingston (C)
2 Latrobe (C) (Vic.)  Latrobe (C)

$others
$others$spatial
character(0)

$others$temporal
[1] "Interstate" "Overseas"   "Unknown"
```

The result of the `check_key()` function is a list containing three elements: 1) matched keys from both tables, 2) potentially paired keys, and 3) others keys that can't be matched. Here, the main mismatch arises from the two LGAs: Kingston and Latrobe (Kingston is a LGA in both Victoria and South Australia and Latrobe is a LGA in both Victoria and Tasmania). Analysts can then reconcile the spatial and temporal data based on this check summary and recreate the cubble object:

```
R> 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))
R>
R> covid2 <- covid |> filter(!lga %in% check_res$others$temporal)
R>
R> (cb <- make_cubble(spatial = lga2, temporal = covid2))

# cubble:   key: lga [80], index: date, nested form, [sf]
# spatial:  [140.961682, -39.1339581, 149.976291, -33.9960517], WGS 84
```

```
# temporal: date [date], n [dbl], avg_7day [dbl]
  lga             long   lat                                    geometry ts
  <chr>          <dbl> <dbl>                                <GEOMETRY [°]> <list>
1 Alpine (S)      147. -36.9 POLYGON ((146.7258 -36.45922, 146.7198 -3~ <tbl_ts>
2 Ararat (RC)     143. -37.5 POLYGON ((143.1807 -37.73152, 143.0609 -3~ <tbl_ts>
3 Ballarat (C)    144. -37.5 POLYGON ((143.6622 -37.57241, 143.68 -37.~ <tbl_ts>
4 Banyule (C)     145. -37.7 POLYGON ((145.1357 -37.74091, 145.1437 -3~ <tbl_ts>
5 Bass Coast (S)  146. -38.5 MULTIPOLYGON (((145.5207 -38.30667, 145.5~ <tbl_ts>
# i 75 more rows
```

4.2. Australian historical maximum temperature

The Global Historical Climatology Network (GHCN) provides daily climate measures for stations worldwide. In the **cubble** package, the cubble object `historical_tmax` contains daily maximum temperature data for 75 stations in Australia, covering two periods: 1971-1975 and 2016-2020. This example uses dplyr verbs to wrangle a cubble object, which is pivoted between the spatial and temporal form during the analysis. Glyph maps are then created with ggplot2 to compare the changes in temperature between these two periods.

To prevent overlapping of weather stations on the map, stations are selected to ensure a minimum distance of 50km. Distance between stations can be calculated with `sf::st_distance()` after turning the spatial cubble to also be an sf object with `make_spatial_sf()`:

```
R> a <- historical_tmax |> make_spatial_sf() |> st_distance()
R> a[upper.tri(a, diag = TRUE)] <- 1e6
R>
R> (tmax <- historical_tmax |>
+   filter(rowSums(a < units::as_units(50, "km")) == 0))

# cubble:   key: id [54], index: date, nested form
# spatial:  [141.2652, -39.1297, 153.3633, -28.9786], Missing CRS!
# temporal: date [date], tmax [dbl]
  id           long   lat  elev name                     wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>                     <dbl> <list>
1 ASN00047016  141. -34.0    43 lake victoria storage     94692 <tibble>
2 ASN00047019  142. -32.4    61 menindee post office      94694 <tibble>
3 ASN00048015  147. -30.0   115 brewarrina hospital       95512 <tibble>
4 ASN00048027  146. -31.5   260 cobar mo                  94711 <tibble>
5 ASN00048031  149. -29.5   145 collarenebri (albert st)  95520 <tibble>
# i 49 more rows
```

Daily maximum temperature is then averaged into monthly series for each periods in the temporal cubble. The last step with `unfold()` moves the two coordinate columns (`long`, `lat`) into the temporal cubble, preparing the data for the glyph map:

```
R> (tmax <- tmax |>
+   face_temporal() |>
```

```
+   group_by(
+     yearmonth = tsibble::make_yearmonth(
+       year = ifelse(lubridate::year(date) > 2015, 2016, 1971),
+       month = lubridate::month(date))
+   )|>
+   summarise(tmax = mean(tmax, na.rm = TRUE)) |>
+   mutate(group = as.factor(lubridate::year(yearmonth)),
+          month = lubridate::month(yearmonth)) |>
+   unfold(long, lat))

# cubble:   key: id [54], index: yearmonth, long form
# temporal: 1971 Jan -- 2016 Dec [1M], has gaps!
# spatial:  long [dbl], lat [dbl], elev [dbl], name [chr], wmo_id [dbl]
  yearmonth id           tmax group month  long   lat
      <mth> <chr>       <dbl> <fct> <dbl> <dbl> <dbl>
1  1971 Jan ASN00047016  31.1 1971      1  141. -34.0
2  1971 Jan ASN00047019  33.1 1971      1  142. -32.4
3  1971 Jan ASN00048015  33.9 1971      1  147. -30.0
4  1971 Jan ASN00048027  32.5 1971      1  146. -31.5
5  1971 Jan ASN00048031  33.3 1971      1  149. -29.5
# i 1,276 more rows
```

A quick check on the number of observations for each location is made, revealing that there are several with less than 24 observations – these stations lack temperature values for some months. In this example, those stations are removed by switching to the spatial cubble to operate on the spatial component over time, and then, move back into the temporal cubble (to make the glyph map):

```
R> tmax <- tmax |>
+   face_spatial() |>
+   rowwise() |>
+   filter(nrow(ts) == 24) |>
+   face_temporal()
```

The following code creates the glyph map (a) in Figure 3 (additional codes are needed for highlighting the single station, Cobar and styling) and the glyph map (c) is produced similarly with the difference series, rather than the two original series, being plotted.

```
nsw_vic <- ozmaps::abs_ste |>
  filter(NAME %in% c("Victoria","New South Wales"))

tmax |>
  ggplot(aes(x_major = long, x_minor = month,
             y_major = lat, y_minor = tmax,
             group = interaction(id, group))) +
  geom_sf(data =  nsw_vic, ...,  inherit.aes = FALSE) +
```

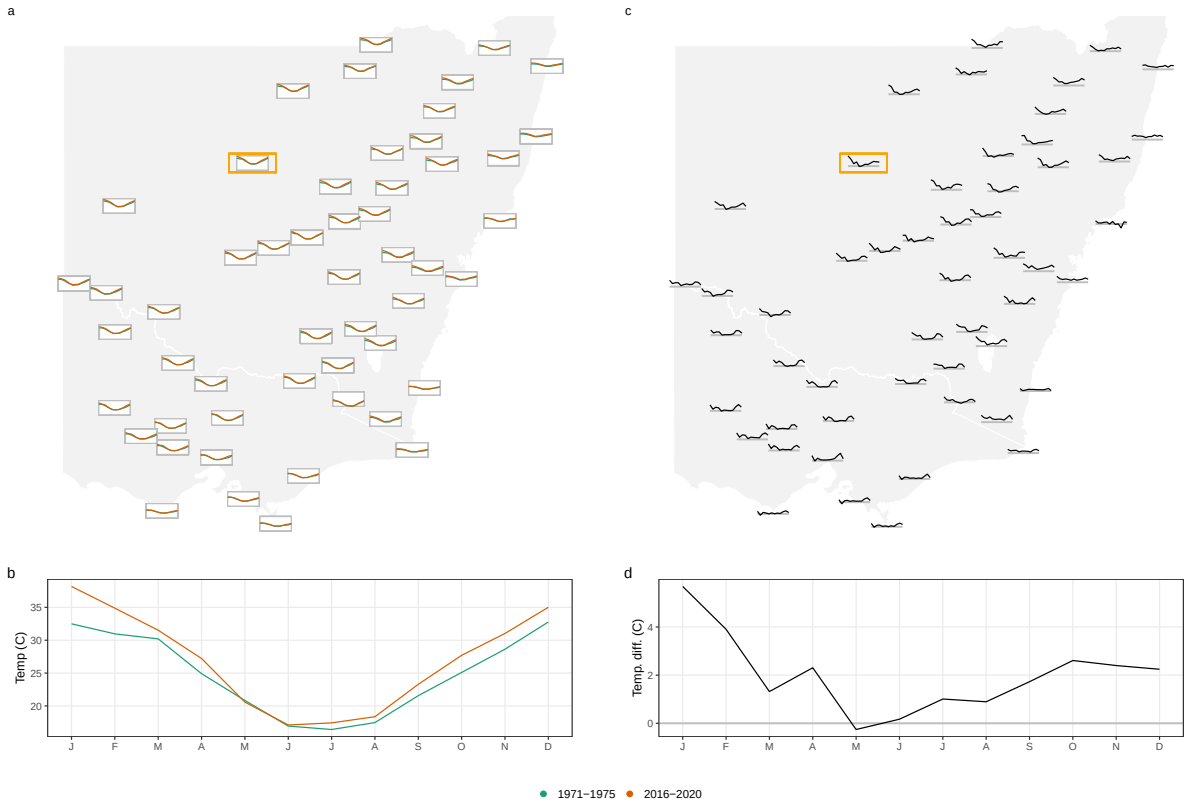

Figure 3: Glyph maps comparing temperature change between 1971-1975 and 2016-2020 for 54 stations in Victoria and New South Wales, Australia. Overlaid line plots show monthly temperature (a) where a hint of late summer warming can be seen. Transforming to temperature differences (c) shows pronounced changes between the two periods. The horizontal guideline marks zero difference. One station, Cobar, is highlighted in the glyph maps and shown separately (b, d). Here the late summer (Jan-Feb) warming pattern, which is more prevalent at inland locations, is clear.

```
  geom_glyph_box(width = 0.8, height = 0.3) +
  geom_glyph(aes(color = group), width = 0.8, height = 0.3) +
  ...
```

4.3. River levels and rainfall in Victoria

The Bureau of Meteorology collects water level data and they can be matched with the precipitation data from the climate weather stations. The data `river` from the **cubble** package contains water course level data for 71 river gauges collected in Victoria, Australia. Victoria weather station data can be subsetted from the `climate_aus` data in the **cubble** package. The example demonstrates the use of matching function introduced in Section 3.1 to detect river gauges that can reflects precipitation in Victoria.

```
R> climate_vic <- climate_aus |>
+   filter(between(as.numeric(substr(id, 7, 8)), 76, 90)) |>
```

```
+   mutate(type = "climate")
R> river <- cubble::river |> mutate(type = "river")
```

Spatial match on the site location is first performed since rainfall can directly impact water level in the nearby river. With `match_spatial()`, we can obtain a summary of the 10 closest pairs:

```
R> res_sp <- match_spatial(df1 = climate_vic, df2 = river,
+                          spatial_n_group = 10)
R> print(res_sp, n = 20)

# A tibble: 10 x 4
   from        to      dist group
   <chr>       <chr>    [m] <int>
 1 ASN00088051 406213 1838.     1
 2 ASN00084145 222201 2185.     2
 3 ASN00085072 226027 3282.     3
 4 ASN00080015 406704 4034.     4
 5 ASN00085298 226027 4207.     5
 6 ASN00082042 405234 6153.     6
 7 ASN00086038 230200 6167.     7
 8 ASN00086282 230200 6928.     8
 9 ASN00085279 224217 7431.     9
10 ASN00080091 406756 7460.    10
```

The result can also be returned as a list of matched cubbles, by setting the argument `return_cubble = TRUE`. All the results can be combined into a single cubble using `bind_rows()`, after excluding the two pairs where a river station is matched to more than one weather stations (river station 226027 is matched twice in group 3 and 5 and similarly for station 230200 in group 7 and 8).

```
R> res_sp <- match_spatial(
+   df1 = climate_vic, df2 = river,
+   spatial_n_group = 10, return_cubble = TRUE)
R> (res_sp <- res_sp[-c(5, 8)] |> bind_rows())

# cubble:   key: id [16], index: date, nested form, [sf]
# spatial:  [144.5203, -38.144913, 148.4667, -36.128657], WGS 84
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id      long   lat  elev name  wmo_id ts       type              geometry
  <chr>  <dbl> <dbl> <dbl> <chr>  <dbl> <list>   <chr>          <POINT [°]>
1 ASN00~  145. -37.0 290   rede~  94859 <tibble> clim~  (144.5203 -37.0194)
2 406213  145. -37.0  NA   CAMP~     NA <tibble> river (144.5403 -37.01512)
3 ASN00~  148. -37.7  62.7 orbo~  95918 <tibble> clim~  (148.4667 -37.6922)
4 222201  148. -37.7  NA   SNOW~     NA <tibble> river  (148.451 -37.70739)
5 ASN00~  147. -38.1   4.6 east~  94907 <tibble> clim~  (147.1322 -38.1156)
# i 11 more rows
# i 2 more variables: group <int>, dist [m]
```

To match the water level series with precipitation, the function `match_temporal()` is used with the variable `group` and `types` identifying the matching group and the two data sources:

```
R> (res_tm <- match_temporal(data = res_sp,
+                            data_id = type, match_id = group,
+                            temporal_by = c("prcp" = "Water_course_level")))

# A tibble: 8 x 2
  group match_res
  <int>     <dbl>
1     1        30
2     2         5
3     3        14
4     4        20
5     6        23
# i 3 more rows
```

Similarly, the cubble output can be returned using the argument `return_cubble = TRUE`. Here we select the four pairs with the highest number of matching peaks and show them on the map (a) and as standardized series (b) in Figure 4.

```
R> res_tm <- match_temporal(data =  res_sp,
+                           data_id = type, match_id = group,
+                           temporal_by = c("prcp" = "Water_course_level"),
+                           return_cubble = TRUE)
R> (res_tm <- res_tm |> bind_rows() |> filter(group %in% c(1, 7, 6, 9)))

# cubble:   key: id [8], index: date, nested form, [sf]
# spatial:  [144.5203, -37.8817, 147.572223, -36.8472], WGS 84
# temporal: date [date], matched [dbl]
  id          long   lat  elev name  wmo_id type              geometry group
  <chr>      <dbl> <dbl> <dbl> <chr>  <dbl> <chr>          <POINT [°]> <int>
1 ASN00088~  145. -37.0 290   rede~  94859 clim~  (144.5203 -37.0194)     1
2 406213     145. -37.0  NA   CAMP~     NA river (144.5403 -37.01512)     1
3 ASN00082~  146. -36.8 502   stra~  95843 clim~  (145.7308 -36.8472)     6
4 405234     146. -36.9  NA   SEVE~     NA river (145.6828 -36.88701)     6
5 ASN00086~  145. -37.7  78.4 esse~  95866 clim~  (144.9066 -37.7276)     7
# i 3 more rows
# i 3 more variables: dist [m], ts <list>, match_res <dbl>
```

4.4. ERA5: climate reanalysis data

The ERA5 reanalysis (Hersbach *et al.* 2020) provides hourly estimates of atmospheric, land and oceanic climate variables on a global scale and is available in the NetCDF format from Copernicus Climate Data Store (CDS). This example demonstrates a case of analysing raster spatio-temporal data using cubble, replicating Figure 19 from the Hersbach *et al.* (2020)

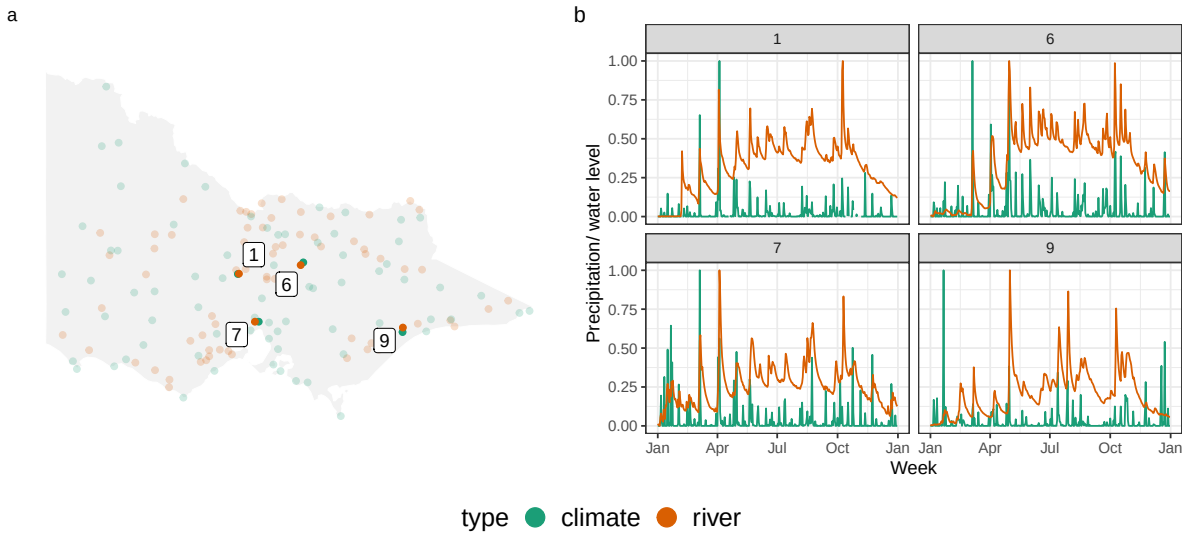

Figure 4: Example of matching weather stations and river gauges. These four stations show on the map (a) and time (b) would be considered to be matching. Precipitation and water level have been standardised between 0 and 1 to be displayed in the same scale in (b). The peaks in the time series roughly match, and would reflect percipitation increasing water levels.

paper. The plot shows the southern polar vortex splitting into two on 2002-09-26, and further splitting into four on 2002-10-04. Further explanation of why this is interesting can be found in the figure source, and also in Simmons *et al.* (2020) and Simmons *et al.* (2005).

A `ncdf4` object (Pierce 2019) can be converted into a cubble using `as_cubble()` and the NetCDF data can be subsetted with arguments `vars`, `long_range` and `lat_range`. In this example, the variables q (specific humidity) and z (geopotential) are read in and the coordinates are subsetted to every degree in longitude and latitude:

```
R> raw <- ncdf4::nc_open(here::here("data/era5-pressure.nc"))
R> (dt <- as_cubble(
+   raw, vars = c("q", "z"),
+   long_range = seq(-180, 180, 1), lat_range = seq(-88, -15, 1)))

# cubble:   key: id [26640], index: time, nested form
# spatial:  [-180, -88, 179, -15], Missing CRS!
# temporal: time [date], q [dbl], z [dbl]
     id  long   lat ts
  <int> <dbl> <dbl> <list>
1     1  -180   -15 <tibble [8 x 3]>
2     2  -179   -15 <tibble [8 x 3]>
3     3  -178   -15 <tibble [8 x 3]>
4     4  -177   -15 <tibble [8 x 3]>
5     5  -176   -15 <tibble [8 x 3]>
# i 26,635 more rows
```

Once the NetCDF data is coerced into a cubble object, subsequent analysis can be conducted

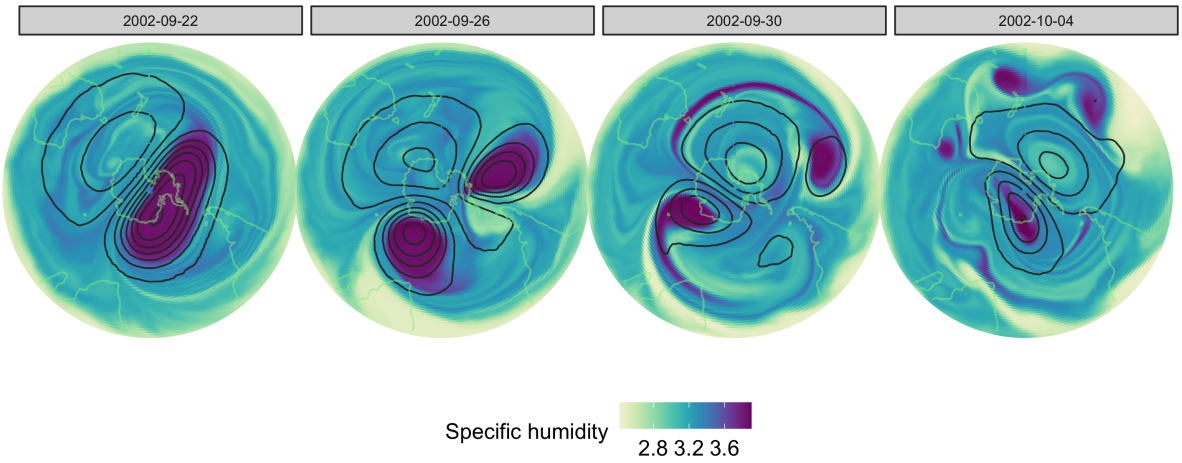

Figure 5: An example illustrating that cubble can be used to readily reproduce common spatiotemporal analyses. This plot of ERA5 reanalysis (Fig. 19, Hersbach et al, 2020) shows the break-up of the southern polar vortex in late September and early October 2002. The polar vortex, signalled by the high specific humidity, splits into two on 2002-09-26 and further splits into four on 2002-10-04.

to filter on the date of interest, scale the variable specific humidity and create visualisation in ggplot to reproduce the ERA5 plot. A snippet of code to create Figure 5 is provided below with additional codes needed to style the plot.

```
res <- dt |>
  face_temporal() |>
  filter(lubridate::date(time) %in%
           as.Date(c("2002-09-22", "2002-09-26",
                     "2002-09-30", "2002-10-04"))) |>
  unfold(long, lat) |>
  mutate(q = q* 10^6)

con <- rnaturalearth::ne_coastline("small", returnclass = "sf")
box <- st_bbox(c(xmin = -180, ymin = -90, xmax = 180, ymax = -15),
               crs = st_crs(con))
country <- con |>
  st_geometry() |>
  st_crop(box) |>
  st_cast("MULTILINESTRING")

res |>
  ggplot() +
  geom_point(aes(x = long, y = lat, color = q)) +
  geom_contour(data = res, aes(x = long, y = lat, z = z), ...) +
  geom_sf(data = country, ...) +
  ...
```

4.5. Australian temperature range

Interactive graphics can be especially useful for spatio-temporal data because they make it possible to look at the data in multiple ways on-the-fly. The last example describes the process of using cubble with the **crosstalk** package to build an interactive display connecting a map of Australia, with ribbon plots of temperature range observed at the stations in 2020. The purpose is to explore the variation of monthly temperature range over the country.

We will first summarise the daily data in `climate_aus` into monthly average and calculate the variance of the temperature difference, which will be used to color the temperature band later.

```
clean <- climate_aus |>
  face_temporal() |>
  mutate(month = lubridate::month(date)) |>
  group_by(month) |>
  summarise(
    tmax = mean(tmax, na.rm = TRUE),
    tmin = mean(tmin, na.rm = TRUE),
    diff = mean(tmax - tmin, na.rm = TRUE)
  ) |>
  face_spatial() |>
  rowwise() |>
  mutate(temp_diff_var = var(ts$diff, na.rm =TRUE))
```

The spatial and temporal cubble are then created into shared `crosstalk` objects, plotted as ggplots, and combined together using `crosstalk::bscols()`:

```
sd_spatial <- clean |> SharedData$new(~id, group = "cubble")

sd_temporal <- clean |>
  face_temporal() |>
  SharedData$new(~id, group = "cubble")

p1 <- sd_spatial |> ggplot() + ...
p2 <- sd_temporal |> ggplot() + ...
crosstalk::bscols(plotly::ggplotly(p1), plotly::ggplotly(p2), ...)
```

Figure 6 shows three snapshots of the interactivity. Plot (a) shows the initial state of the interactive display: all locations are shown as dots on the map, coloured by the temperature range, and the right plot shows the ribbons representing maximum to minimum for all stations. In plot (b) the station shows a high variance on the initial map, the "Mount Elizabeth" station, is selected and this produces the ribbon on the right. In plot (c) the lowest temperature in August is selected on the left map and this corresponds to the "Thredbo" station in the mountain area in Victoria and New South Wales. This station is compared to a station in the Tasmania island, the southernmost island of the country, selected on the map.

5. Conclusion

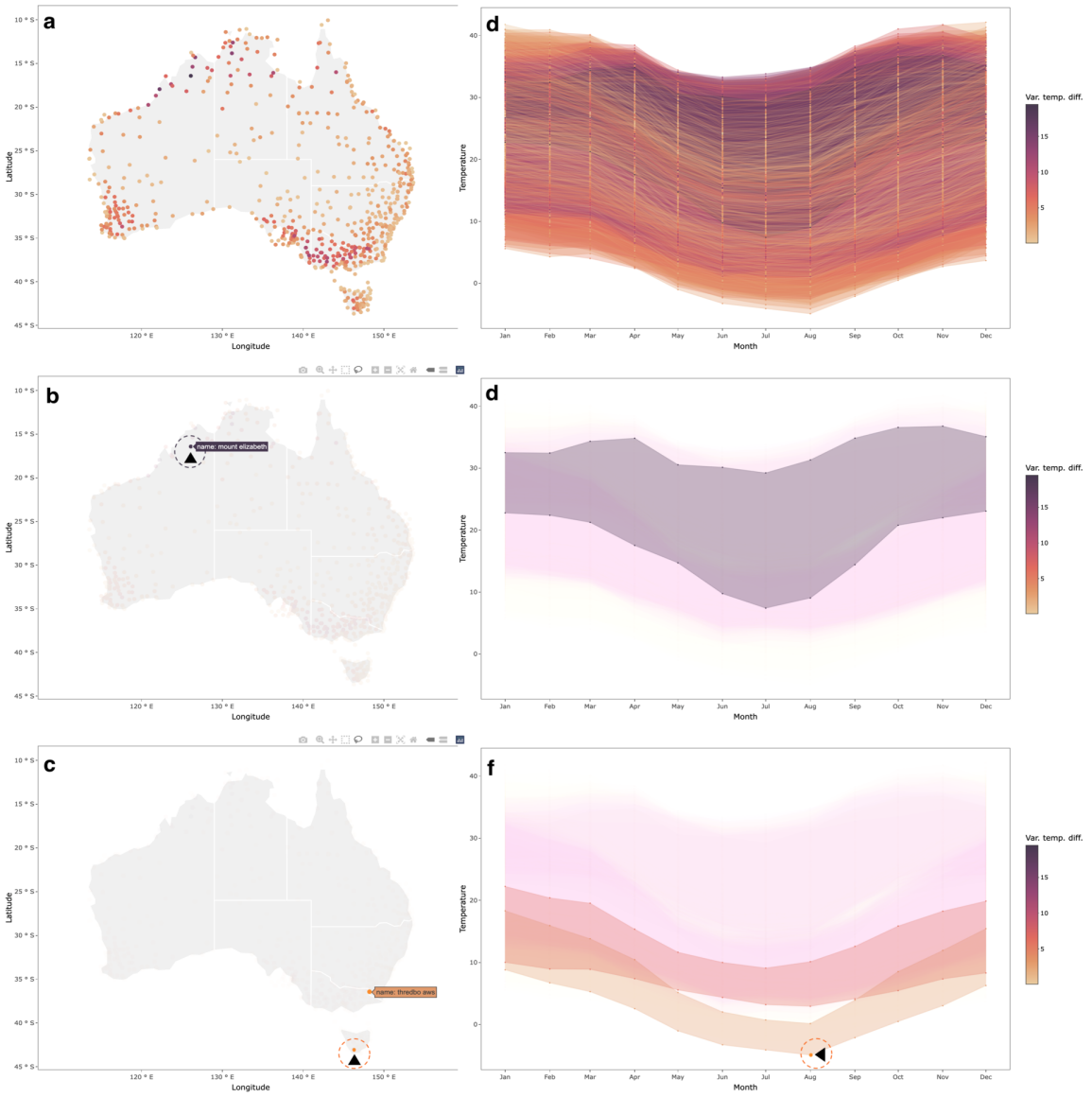

Figure 6: Illustration of using cubble for interactive graphics. Here we explore temperature variation by linking a map and a seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a particular location on the map (here Mount Elizabeth) produces the plot in the second row. The maximum and minimum temperatures are shown using a ribbon. The bottom row first selects the lowest temperature in August in the seasonal display, which highlights the corresponding station on the map (Thredbo). Another station, located in the Tasmania Island, is then selected to compare its temperature variation with the Thredbo station.

This paper presents the R package **cubble** for organizing, wrangling and visualizing spatio-temporal data. The package introduces a new data structure, `cubble`, consisting of two subclass, spatial cubble and a temporal cubble, to organise spatio-temporal data in two different formats within the tidy data framework. The data structure and functions introduced in the package can be used and combined with existing tools for data wrangling, spatial and temporal data analysis, and visualization.

The paper includes plenty examples to illustrate the utility of **cubble** as a data structure for spatio-temporal analysis. These examples cover different tasks of a typical analysis workflow: handling data with spatial and temporal misalignment, matching data from multiple sources, and creating both static and interactive spatio-temporal visualisation. For future directions, other commonly-used spatial or temporal data structures can be integrated into the **cubble** package to extend analysts' familiar spatial and temporal toolkit to spatio-temporal.

6. Acknowledgement

This work is funded by a Commonwealth Scientific and Industrial Research Organisation (CSIRO) Data61 Scholarship and started while Nicolas Langrené was affiliated with CSIRO's Data61. The article is created using the package **knitr** (Xie 2015) and **rmarkdown** (Xie *et al.* 2018) in R with the `rticles::jss_article` template. The source code for reproducing this paper can be found at: https://github.com/huizezhang-sherry/paper-cubble.

References

Buja A, Asimov D, Hurley C (1988). "Elements of A Viewing Pipeline." *Dynamic Graphics Statistics*, p. 277.

Buja A, Cook D, Swayne DF (1996). "Interactive High-dimensional Data Visualization." *Journal of Computational and Graphical Statistics*, **5**(1), 78–99. URL https://doi.org/10.2307/1390754.

Cheng J, Sievert C (2021). ***crosstalk**: Inter-Widget Interactivity for HTML Widgets*. R package version 1.1.1, URL https://CRAN.R-project.org/package=crosstalk.

Cheng X, Cook D, Hofmann H (2016). "Enabling Interactivity on Displays of Multivariate Time Series and Longitudinal Data." *Journal of Computational and Graphical Statistics*, **25**(4), 1057–1076. URL https://doi.org/10.1080/10618600.2015.1105749.

Hersbach H, Bell B, Berrisford P, Hirahara S, Horányi A, Muñoz-Sabater J, Nicolas J, Peubey C, Radu R, Schepers D, *et al.* (2020). "The ERA5 Global Reanalysis." *Quarterly Journal of the Royal Meteorological Society*, **146**(730), 1999–2049.

Menne MJ, Durre I, Vose RS, Gleason BE, Houston TG (2012). "An overview of the global historical climatology network-daily database." *Journal of atmospheric and oceanic technology*, **29**(7), 897–910.

Pebesma E (2012). "**spacetime**: Spatio-Temporal Data in R." *Journal of Statistical Software*, **51**(7), 1–30. URL https://doi.org/10.18637/jss.v051.i07.

Pebesma E (2021). ***stars**: Spatiotemporal Arrays, Raster and Vector Data Cubes*. R package version 0.5-2, URL https://CRAN.R-project.org/package=stars.

Pebesma E, Bivand R (2022). "CRAN Task View: Handling and Analyzing Spatio-Temporal Data." Version 2022-03-07, URL https://CRAN.R-project.org/view=SpatioTemporal.

Pierce D (2019). ***ncdf4**: Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files*. R package version 1.17, URL https://CRAN.R-project.org/package=ncdf4.

Schloerke B, Cook D, Larmarange J, Briatte F, Marbach M, Thoen E, Elberg A, Crowley J (2021). ***GGally**: Extension to **ggplot2***. R package version 2.1.2, URL https://CRAN.R-project.org/package=GGally.

Simmons A, Hortal M, Kelly G, McNally A, Untch A, Uppala S (2005). "ECMWF Analyses and Forecasts of Stratospheric Winter Polar Vortex Breakup: September 2002 in the Southern Hemisphere and Related Events." *Journal of the Atmospheric Sciences*, **62**(3), 668 – 689. doi:10.1175/JAS-3322.1. URL https://journals.ametsoc.org/view/journals/atsc/62/3/jas-3322.1.xml.

Simmons A, Soci C, Nicolas J, Bell B, Berrisford P, Dragani R, Flemming J, Haimberger L, Healy S, Hersbach H, Horányi A, Inness A, Munoz-Sabater J, Radu R, Schepers D (2020). "Global Stratospheric Temperature Bias and Other Stratospheric Aspects of ERA5 and ERA5.1." (859). doi:10.21957/rcxqfmg0. URL https://www.ecmwf.int/node/19362.

Sutherland P, Rossini A, Lumley T, Lewin-Koh N, Dickerson J, Cox Z, Cook D (2000). "**Orca**: A Visualization Toolkit for High-dimensional Data." *Journal of Computational and Graphical Statistics*, **9**(3), 509–529. URL https://www.tandfonline.com/doi/abs/10.1080/10618600.2000.10474896.

Teickner H, Pebesma E, Graeler B (2022). ***sftime**: Classes and Methods for Simple Feature Objects that Have a Time Column*. Https://r-spatial.github.io/sftime/, https://github.com/r-spatial/sftime.

Wang E, Cook D, Hyndman RJ (2020). "A New Tidy Data Structure to Support Exploration and Modeling of Temporal Data." *Journal of Computational and Graphical Statistics*, **29**(3), 466–478. URL https://doi.org/10.1080/10618600.2019.1695624.

Wickham H (2014). "Tidy Data." *Journal of Statistical Software*, **59**(10), 1–23. URL https://doi.org/10.18637/jss.v059.i10.

Wickham H, Hofmann H, Wickham C, Cook D (2012). "Glyph-Maps for Visually Exploring Temporal Patterns in Climate Data and Models." *Environmetrics*, **23**(5), 382–393. doi:10.1002/env.2152.

Xie Y (2015). *Dynamic Documents with R and **knitr***. 2nd edition. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1498716963, URL https://yihui.name/knitr/.

Xie Y, Allaire J, Grolemund G (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1138359338, URL https://bookdown.org/yihui/rmarkdown.

Xie Y, Hofmann H, Cheng X (2014). "Reactive Programming for Interactive Graphics." *Statistical Science*, **29**(2), 201 – 213. URL https://doi.org/10.1214/14-STS477.

Affiliation:

H. Sherry Zhang
Monash University
21 Chancellors Walk, Clayton VIC 3800 Australia
E-mail: huize.zhang@monash.edu

Dianne Cook
Monash University
21 Chancellors Walk, Clayton VIC 3800 Australia
E-mail: dicook@monash.edu

Ursula Laa
University of Natural
Resources and Life Sciences
Gregor-Mendel-Straße 33, 1180 Wien, Austria
E-mail: ursula.laa@boku.ac.at

Nicolas Langrené
BNU-HKBU
United International College
2000 Jintong Road, Tangjiawan, Zhuhai, Guangdong Province, China
E-mail: nicolaslangrene@uic.edu.cn

Patricia Menéndez
Monash University
21 Chancellors Walk, Clayton VIC 3800 Australia
E-mail: patricia.menendez@monash.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd