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 functions provided in the **cubble** R package equip users with the capability to handle hierarchical spatial and temporal structures. 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 include 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, Pebesma (2012) summarises spatio-temporal data into three forms: time-wide, space-wide, and long formats. The associated package **spacetime** (Pebesma 2012) implements four spatio-temporal layouts (full grid, sparse grid, irregular, and trajectory) to handle different space and time combinations. The package **stars** (Pebesma 2021) has a new implementation to use dense arrays to represent spatio-temporal cubes. It also interfaces with the package **sf** (Pebesma 2018), commonly used for wrangling spatial data, and the **tidyverse** (Wickham *et al.* 2019) suite for general data wrangling and visualization in R.

Still, the data representation for spatio-temporal data can be further extended and there are two reasons for this. Firstly, the raw data sourced in the wild is less often presented in any one of the layouts above, and fitting the raw data into a data object can sometimes be difficult. More often, spatio-temporal data are collected in separate 2D tables and analysts need to assemble them into a whole piece before exploring the data. Examples of components of spatio-temporal data can be 1) areal data recording the shape of a collection of areas of interest; 2) geostatistical data storing the longitude and latitude coordinates of locations, typically also with other metadata related to the location, and; 3) temporal data of each location across time.

The other reason is about tidy data concepts (Wickham 2014) and how they should be applied to spatio-temporal data. According to the tidy data principles, data should be structured into 1) one row per observation, 2) one column per variable, and 3) one type of data per table. The long form data is preferred over wide data form given the downstream packages such as **dplyr** (Wickham *et al.* 2022) and **ggplot2** (Wickham 2016) for data wrangling and visualization. However, the long form can be inefficient to store feature geometries, especially for large multipolygons for hourly, daily or sub-daily periods over years, which are extensively collected and handled, for example in time series analysis. This poses the question of how to arrange spatial and temporal variables in a way that would make data wrangling, visualizing and analyzing spatio-temporal data easier.

This paper presents a new R package, **cubble**, which addresses the two issues mentioned above. In the package, a new class, also called cubble, is proposed 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 be applied to full grid, sparse grid, or irregular, but not trajectory, which is outside the scope of this work. 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 with hierarchical structure, 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 Australian weather station data and river level 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

2.1. The cubble object

[conceptual framework]

Spatio-temporal data can encompass data with various spatial and temporal characteristics and different data require different structures for wrangling and analysis: climate weather stations record data at fixed point location but may suffer from potential temporal data quality issue such as missing data for certain days. GPS data tracks unique point locations at different timestamps and is represented as trajectories. Satellite imageries capture snapshots of landscapes at selected time and is commonly structured as raster data. The spatio-temporal data cubble address are those collected at unique fixed locations while allowing for irregularity in the temporal dimension, such as the weather station data. This corresponds to the full space-time and sparse space-time layouts in the spacetime paper (Pebesma 2012).

The cubble class is an S3 class built on tibble that allows the spatio-temporal data to be wrangled in two forms: a nested/ spatialform and a long/temporal form. It consists of two subclasses:

- a nested/ spatial cubble is represented by the class `c("spatial_cubble_df", "cubble_df")`
- a long/ temporal cubble is represented by the class `c("temporal_cubble_df", "cubble_df")`

In a nested cubble, spaital variables are organised as columns and temporal variables are nested within a specialised `ts` column. The nested cubble object printed below contains weather records in three airport stations (the creation of a cubble object will be explained in Section 2.2). This toy data is a subset of a larger data `climate_aus` collected from Global Historical Climatology Network Daily (GHCND). It records three airport stations located in Melbourne, Australia and includes spatial variables such as station ID, longitude, latitude, elevation, station name, World Meteorology Organisation ID. The dataset contains temporal variables including precipitation, maximum and minimum temperature, which can be read from the cubble header:

```
R> cb_nested

# 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 long cubble, temporal variables are expanded in the long form and spatial variables are stored as a data attribute:

```
R> cb_long
```

```
# 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

A cubble object inherits the attributes from tibble (and its subclasses): `class`, `row.names`, and `names`. Additionally, it has three specialised attributes:

- `key`: the spatial identifier
- `index`: the temporal identifier
- `coords`: a pair of ordered coordinates associated with the location

Readers who are familiar with the `key` and `index` attributes from the `tsibble` package would already understand the two arguments. In cubble, the `key` attribute identifies the row in the nested cubble, and when combined with the `index` argument, it identifies the row in the long 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. It can be either an unprojected pair of longitude and latitude, or a projected easting and northing value. The `sf` package is used under teh hood to calculate the bounding box, displayed in the header of a nested cubble.

The long cubble has a special attribute called `spatial` to store the spatial variables, which includes all the variables from the nested cubble, except for the `ts` column. Shortcut functions are available to extract attributes, for example, `spatial()` for extracting spatial variables from the long cubble:

```
R> spatial(cb_long)

# 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

Creating from separate spatial and temporal tables

In many cases, spatio-temporal data arrive in separate tables for analysis. For example, in climate data, analysts may initially receive station data containing geographic location information, recorded variables and their recording periods. They can then query the temporal variables using the stations of interest to obtain the relevant temporal data. Alternatively, analyses may begin as purely spatial or temporal, and analysts may obtain additional temporal or spatial data to expand the result to spatio-temporal.

The function `make_cubble()` composes a `cubble` object from a spatial table (`spatial`) and a temporal table (`temporal`), along with three attributes introduced in the subsection 2.1: `key`, `index`, and `coords`. The following code creates the nested `cubble` object shown previously:

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

Coercing from foreign objects

Spatio-temporal data in other foreign objects can be coerced into a `cubble` object with the function `as_cubble()`. This includes casting from a `tibble` or `data.frame` with both spatial and temporal information, a NetCDF object, a `stars` object (Pebesma 2021), and a `sftime` object (Teickner *et al.* 2022). The two examples below show the casting from a tibble and a NetCDF object. The dataset `climate_flat` combines the spatial data, `stations`, with the temporal data, `meteo`, into a single tibble object. It can be coerced into a cubble using:

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

In R, there are several packages available for wrangling NetCDF data, including `ncdf4`, `RNetCDF`, and `tidync`. The code below converts a NetCDF object of class `ncdf4` into a cubble object:

```
R> path <- system.file("ncdf/era5-pressure.nc", package = "cubble")
R> raw <- ncdf4::nc_open(path)
R> as_cubble(raw)
```

```
# cubble:   key: id [26565], index: time, nested form
# spatial:  [113, -53, 153, -12], Missing CRS!
# temporal: time [dttm], q [dbl], z [dbl]
     id  long   lat ts
  <int> <dbl> <dbl> <list>
1     1  113    -12 <tibble [6 x 3]>
2     2  113.   -12 <tibble [6 x 3]>
3     3  114.   -12 <tibble [6 x 3]>
4     4  114.   -12 <tibble [6 x 3]>
5     5  114    -12 <tibble [6 x 3]>
# i 26,560 more rows
```

Sometimes, analysts may choose to read onlny a subset of the NetCDF data. In such cases, the arguments `vars`, `long_range` and `lat_range` can be used to subset the data based on variable and the grid resolution:

```
R> as_cubble(raw, vars = "q",
+            long_range = seq(-180, 180, 1), lat_range = seq(-90, 90, 1))
```

2.3. Functions and methods

Table 1 summarises the functions implemented in the **cubble** package and Table 2 details the methods implemented for each of the three cubble classes. The `cubble_df` class handles methods that behave consistently in both nested (`spatial_cubble_df`) and long cubble (`temproal_cubble_df`). When the method has distinct behavior for the nested cubble and temporal cubble, it is implemented separately for each subclass.

Table 1: Functions implemented in the **cubble** package

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: Methods implemented for the three cubble classes.

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

Specifically, the pair of verbs, `face_temporal()` and `face_spatial()`, pivot the cubble object between the spatial and temporal face of the multivariate spatio-temporal cube, as illustrated in Figure 1. The code below uses the function `face_temporal()` and `face_spatial()` to transform between a nested cubble (`cb_nested`) and a long one (`cb_long`), introduced earlier in subsection 2.1:

```
R> identical(face_temporal(cb_nested), cb_long)

[1] TRUE

R> identical(face_spatial(cb_long), cb_nested)

[1] TRUE
```

The pair of verbs are exact inverse and apply both functions on a cubble object will result in the object itself:

```
R> identical(face_spatial(face_temporal(cb_nested)), cb_nested)

[1] TRUE

R> identical(face_temporal(face_spatial(cb_long)), cb_long)

[1] TRUE
```

2.4. Compatibility with tsibble and sf

Analysts often have their preferred spatial or temporal data structure that they prefer to use for spatio-temporal analysis. For example, the `tbl_ts` class from the `tsibble` package (Wang *et al.* 2020) is commonly used in time series forecasting and similarly, the `sf` class (Pebesma 2018) is frequently used in spatial data science. In `cubble`, analysts have the flexibility to

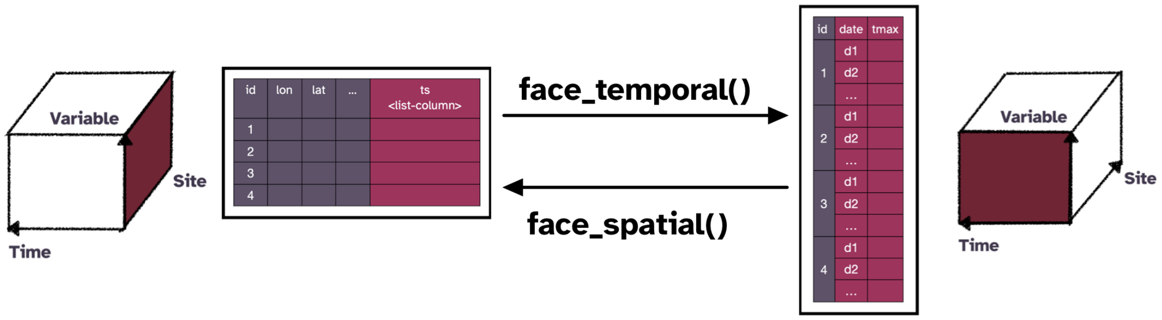

Figure 1: An illustration of the function `face_temporal()` and `face_spatial()`: `face_temporal()` converts a spatial cubble (neseted form) into a temporal cubble (long form) to focus on the temporal variables. Conversely, `face_spatial()` transforms a temporal cubble into a spatial one to emphasize spatial variables.

combine these two structures together by allowing the spatial component to also be an `sf` object and the temporal component to also be a `tsibble` object.

Using a tsibble for 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, i.e. `meteo_ts` in the example below. The `tsibble` class from the input will be carried over to the `cubble` object:

```
R> ts_nested <- make_cubble(
+   spatial = stations, temporal = meteo_ts, coords = c(long, lat))
R> (ts_long <- face_temporal(ts_nested))
```

```
# 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_long)
```

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

The long cubble shows [tsibble] in the header to indicate the object also being in a tbl_ts class (tsibble). Methods applies to the tbl_ts class can also be applied to the temporal cubble objects, for example, checking whether the data contain temporal gaps:

```
R> ts_long %>% has_gaps()
```

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

An existing cubble object can promote its temporal component to a tsibble object by applying make_temporal_tsibble(). The following code illustrates this with the object cb_long created in section 2.2 and the promoted cubble object is equal to the cubble object originally created from a tsibble object:

```
R> ts_long2 <- cb_long %>% make_temporal_tsibble()
R> identical(ts_long2, ts_long)
```

```
[1] TRUE
```

Using sf for the spatial component

Similarly, an sf object can be supplied as the spatial component to create a cubble object, with the coords argument being omitted. This opens up the possibility to represent fixed area with polygons or multipolygons (see Applications 4.1) and the coords argument will be calculated as the centroids of the (multi)polygons. The [sf] print in the cubble header suggest an spatial component being also a sf object:

```
R> (sf_nested <- 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_nested)
```

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

The `sf` functions applicable to a `cubble` object have been listed in Table 1 and the following code shows how to perform coordinate transformation with `st_transform` on a cubble object:

```
R> sf_nested %>% 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 counterpart to promote the spatial component in an existing `cubble` to be an `sf` object is `make_spatial_sf()`:

```
R> cb_nested %>% 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 structure and this section compares and contrasts **cubble** with other existing alternative, 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**, but **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 expensive to repeat, i.e. polygons or multipolygons. Consider the `climate_aus` data in the **cubble** package with 639 stations observed daily in a single year 2020. The created `sftime` object is approximately 14 times larger than the corresponding `cubble` object (118.24 MB vs. 8.52 MB).

3. Other features and considerations

3.1. Data fusion and matching

One common task when working with spatio-temporal data is to match nearby sites. For example, we may want to verify the location of an old list of stations with current stations, or we may want to match the data from different data sources. In cubble, spatial and temporal matching are performed using the functions `match_spatial()` and `match_temporal()`. The `match_spatial()` function calculates the spatial distance between observations in two cubble objects. Various distance measures are available (check `sf::st_distance`). Analysts can specify the number of matched groups to output using the `spatial_n_group` argument (default to 4 groups) and the number of matches per group using the `spatial_n_group`argument (default to 1, one-to-one matching). The syntax to use `match_spatial()` is:

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

The function `match_temporal()` calculates the similarity between time series within spatially matched groups. Two identifiers are required: one for separating each matched group (`match_id`) and one for separating the two data sources (`data_id`). The argument `temporal_by` uses the `by` syntax from dplyr's `*_join` to specify the temporal variables to match.

The similarity score between two time series is calculated using a matching function, which can be customised by the analysts. The matching function takes two time series as a list and returns a single numerical score. This allows for flexibility in using existing time series distance calculation implementation. By default, cubble implements a simple peak matching algorithm (`match_peak`) that counts the number of peaks in two time series that fall within a specified temporal window. The syntax to use `match_temporal()` is

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

3.2. Interactive graphics

The workflow with the `cubble` class works well with 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)) that is available in R with the package **crosstalk** (Cheng and Sievert 2021). Figure 2

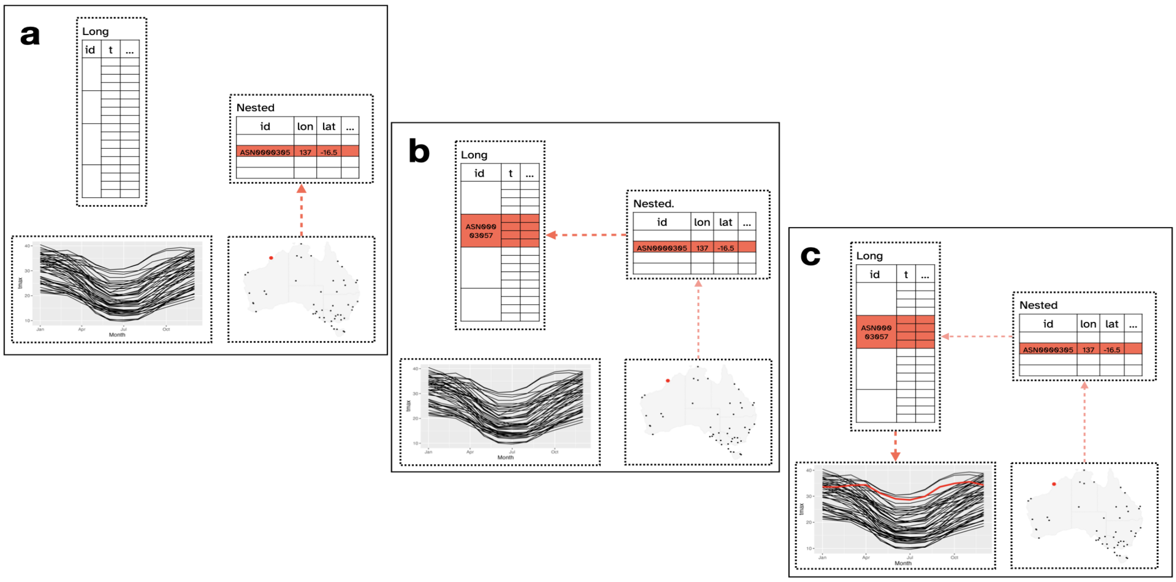

Figure 2: Linking between multiple plots. The line plots and the map are constructed from shared `crosstalk` objects (long and nested `cubble`). When a station is selected on the map (a), the corresponding row in the nested `cubble` will be activated. This will link to all the rows with the same id in the long `cubble` (b) and update the line plot (c).

illustrates how linking can be achieved between a map and multiple time series in a `cubble` object. The map (produced from the nested form) and time series (produced from the long form) are both shared `crosstalk` objects. When a user makes a selection on the map, the site is highlighted (a). This activates a row in the nested `cubble`, which is then communicated to the long `cubble` – all the observations with the same id (b) will be selected. The long `cubble` will then highlight the corresponding series in the time series plot (c).

Linking is also available starting from the time series plot, by selecting points. This will activate rows having the same id in the long `cubble`. The corresponding rows in the nested `cubble` are activated, and highlighted on the map. (An illustration can be found in the appendix.) Note that this type of linking, both from the map or the time series, is what Cook and Swayne (2007) would call categorical variable linking, where station id is the categorical variable.

3.3. Spatio-temporal transformations

Sometimes, we wish to communicate spatial and temporal information collectively through visualisation. This can be achieved through several graphical displays: one can make faceted maps across time, creating map animations, or constructing interactive graphics to link between map and time series plot. The concept of glyph maps was initially proposed in Wickham *et al.* (2012). The underlying idea is to transform the temporal coordinates into spatial coordinates so that time series plot can be displayed on the map.

The package `GGally` initially implement the glyph map. It uses `glyphs()` to calculate the axis transformation and then uses `geom_polygon()` to draw the map. In cubble, a ggproto implementation `geom_glyph()` performs the linear algebra internally as data transformation

. The `geom_glyph()` requires four aesthetics: `x_major`, `y_major`, `x_minor`, and `y_minor`. The major axes are the outer spatial coordinates and the minor axes are the inner/ temporal coordinates:

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

Some useful controls over the glyph map includes:

- polar coordinate glyph maps with `polar = TRUE`,
- adjust glyph size with arguments `width` and `height`,
- glyph scale to fixed (`global_rescale` defaults to `TRUE`) or free, and
- reference boxes and lines with `geom_glyph_box()` and `geom_glyph_line()`.

4. Applications

The five examples here are chosen to illustrate these aspects of the **cubble** package: creating a `cubble` object from two Coronavirus (COVID) data tables with the complication of differing location names, using spatial transformations to make a glyph map of seasonal temperature changes over years, matching river level data and weather station records for analysis of water supply, reading NetCDF format data to reproduce a climate reanalysis plot, and the workflow to create complex interactive linked plots.

4.1. Victoria COVID spatio-temporal incidence and spread

Since the start of the pandemic, the Victoria State Government in Australia has provided daily COVID counts by local government area (LGA). This data can be used to visualize COVID incidence and spread spatially, when combined with map polygon data available from the Australian Bureau of Statistics. These different sources need to be combined for the analysis, by matching the LGA names. Both the COVID count data (`covid`) and the LGA information (`lga`) are available in the **cubble** package with the `covid` data stored as a `tsibble` object, the `lga` data as an `sf` object.

The `make_cubble()` function is used to create a `cubble` object from the two spatial and temporal tables as described in Section 2.2. The `by` argument allows for mismatch of key names from the two tables in the `*_join()` `by` syntax:

```
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 warning message reports on the mismatches of location from both sides: there are LGAs in the COVID data that do not match with LGAs names in the spatial polygon data, and vice versa. The `make_cubble()` function prompts the use of `check_key()` to identify the mismatches:

```
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
[1] "No usual address (Vic.)"
[2] "Migratory - Offshore - Shipping (Vic.)"

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

The check result is a list with three elements: 1) matched pairs from both tables, 2) those pairs that can be potentially paired, and 3) others. Here the main mismatch comes from different encodings of two LGAs: Kingston and Latrobe. Analysts can then modify the input spatial and temporal data accordingly and create the cubble again:

```
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)) |>
+   filter(!lga %in% check_res$others$spatial)
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, -39.1591895, 149.976291, -33.9804256], WGS 84
# temporal: date [date], n [dbl], avg_7day [dbl]
  lga              long   lat                                      geometry ts
  <chr>           <dbl> <dbl>                            <MULTIPOLYGON [°]> <list>
1 Alpine (S)       147. -36.9 (((146.7258 -36.45922, 146.8033 -36.45139~ <tbl_ts>
2 Ararat (RC)      143. -37.5 (((143.1807 -37.73152, 143.1793 -37.73242~ <tbl_ts>
3 Ballarat (C)     144. -37.5 (((143.6622 -37.57241, 143.6686 -37.53844~ <tbl_ts>
4 Banyule (C)      145. -37.7 (((145.1357 -37.74091, 145.1331 -37.74281~ <tbl_ts>
5 Bass Coast (S)   146. -38.5 (((145.5207 -38.30667, 145.5418 -38.30938~ <tbl_ts>
# i 75 more rows
```

4.2. Australian historical maximum temperature

The GHCN provides daily climate measures from stations across the world. The data used here (`historical_tmax`) is a subset extracted using the package **rnoaa** (Chamberlain 2021), containing the records of maximum temperature for 237 Australian stations from ∞ through $-\infty$ and provides information also on the latitude, longitude and elevation of each of the stations. The goal of this example is to compare the monthly average maximum temperature between two periods, 1971-1975 and 2016-2020, for stations in Victoria and New South Wales (NSW), using a glyph map.

First, the stations need to be filtered to those in Victoria and NSW by using the station identifiers, stored within the 11 digits of the `id` variable entries. The country code is in the first 5 digits (Australia is represented by "ASN00") and the next 6 digits encode the station following the Australian Bureau of Meteorology (BOM) (Commonwealth of Australia 2022) coding protocols. The NSW stations correspond to entries in the range 46-75 and the Victorian stations to 76-90. Filtering Victoria and NSW stations is a *spatial operation* and hence uses the nested `cubble`:

```
R> tmax <- historical_tmax |>
+   filter(between(as.numeric(stringr::str_sub(id, 7, 8)), 46, 90))
```

Next, the monthly maximum average temperature is calculated for both periods. This is a *temporal operation* requiring a switch into the long `cubble` using the `face_temporal()` function. In addition, a new indicator for the two time periods of interest is created before the calculation of monthly averages:

```
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))
```

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 nested `cubble` to operate on the spatial component over time, and then, move back into the long `cubble` (to make the glyph map):

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

In order to create a glyph map displaying the monthly series (Figure 3), the spatial variables need to be unfolded with the temporal variables. The reason being that the major (`long`, `lat`) and minor (`month`, `tmax`) coordinates need to be on the same table to create the glyph map. The `geom_glyph()` function does both the transformation and the plotting.

```
R> ggplot() +
+   geom_sf(data = MAP_DATA, ...) +
+   geom_glyph(data = tmax,
+              aes(x_major = long, x_minor = month,
+                  y_major = lat, y_minor = tmax,
+                  group = interaction(id, group), color = group),
+              width = 1, height = 0.5) +
+   ...
```

Glyph maps work well to explore temporal patterns across spatial locations, particularly when the spatial locations are gridded. In this example, they are irregularly spaced, which can result in overlapping glyphs obscuring each other. To fix this, one could aggregate data from nearby stations. An example of this use is included in the Appendix.

4.3. River levels and rainfall in Victoria

One common task when working with spatio-temporal data is to match nearby sites. For example, we may want to verify the location of an old list of stations with current stations, or we may want to match the data from different data sources. In this example, we will introduce the spatial and temporal matching in cubble using an example on matching river level data with precipitation in Victoria, Australia.

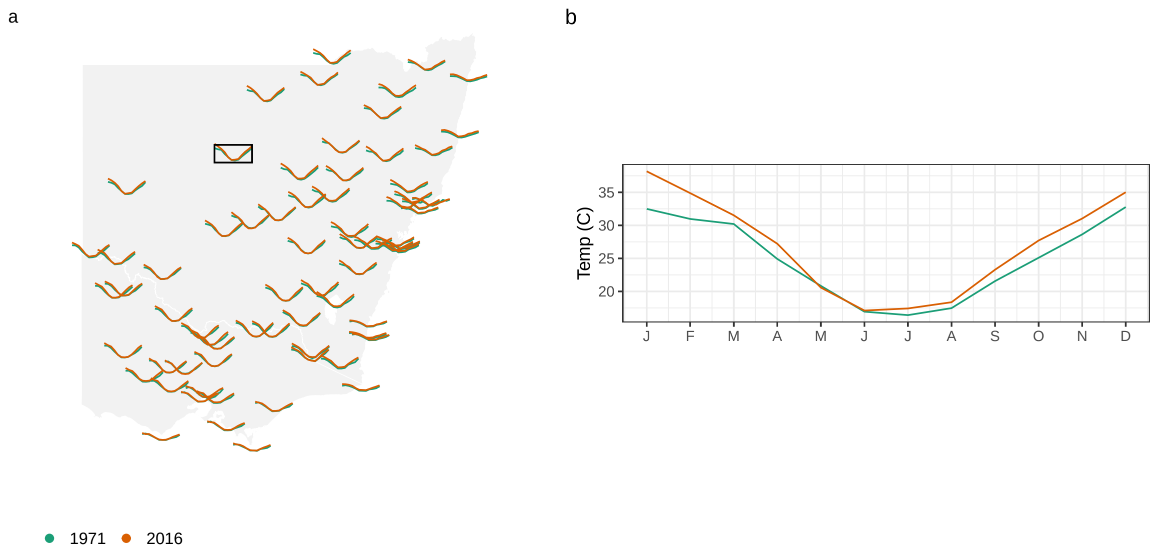

Figure 3: A glyph map of the monthly maximum average temperature for weather stations in Victoria and New South Wales (NSW) for the periods (1971-1975, 2016-2020). The corresponding average time series for the cobar station are display on the top left corner. From the glyph map we can observe that the monthly trend is similar for all locations (low in the winter, high in the summer), and small increased temperatures, particularly in late summer can be seen at most stations in NSW.

The water level data collected by the Bureau of Meteorology, can be compared with the precipitation since rainfall can directly impact water level in river. Both `climate_vic` and `river` are cubble objects, and we can obtain a summary of the 10 closest pairs between them:

```
R> res_sp <- match_spatial(climate_vic, 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 cubble objects by setting the argument `return_cubble = TRUE`. The output is be a list where each element is a paired cubble object. To combine all the results into a single cubble, you can use `bind_rows()`. In the case when a site in the second cubble (the `river` data here) is matched to two stations in the first cubble (`climate_vic` here), the binding may not be successful since cubble requires unique rows in the nested form. In the summary table above, the river station 226027 is matched to more than one weather station: ASN00085072 (group 3) and ASN00085298 (group 5). Similarly, the river station 230200 is matched in group 7 and 8). In such cases, you can either deselect one pair before combining, or work with the list output with the `purrr::map` syntax:

```
R> res_sp <- match_spatial(
+   climate_vic, river,
+   spatial_n_group = 10, return_cubble = TRUE)
R> str(res_sp, max.level = 0)

List of 10

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]
```

For temporal matching, we match the variable `Water_course_level` from the river data to `prcp` in the weather station data. The variable `group` and `types` identify the matching group and the two datasets:

```
R> (res_tm <- res_sp %>%
+   match_temporal(
+     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:

```
R> res_tm <- res_sp %>%
+   match_temporal(
+     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>
```

Figure 4 shows four matched pairs on the map (a) and standardized data as time series (b) with concurrent increasing.

4.4. ERA5: climate reanalysis data

Figure 5 reproduces the ERA5 data row of Figure 19 in Hersbach *et al.* (2020). Here we explain how this would be done using in the **cubble** package. The plots show that the southern polar vortex splits into two on 2002-09-26 and further splits 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).

The ERA5 data (Hersbach *et al.* 2020) provides hourly estimates across the Earth for atmospheric, land and oceanic climate variables. The data is available in the NetCDF format from the European Centre for Medium-Range Weather Forecasts (ECMWF). It can be directly downloaded from Copernicus Climate Data Store (CDS) (Copernicus Climate Change Service 2022) website or via the **ecmwfr** package (Hufkens *et al.* 2019). For the reproduction, we focus on the `era5-pressure` data, hourly pressure levels from 1970 to present, with the *specific humidity* and *geopotential*. The downloaded NetCDF data (`raw`) is first converted to a `cubble` object:

```
R> dt <- as_cubble(
+   raw, vars = c("q", "z"),
+   long_range = seq(-180, 180, 1), lat_range = seq(-88, 88, 1))
```

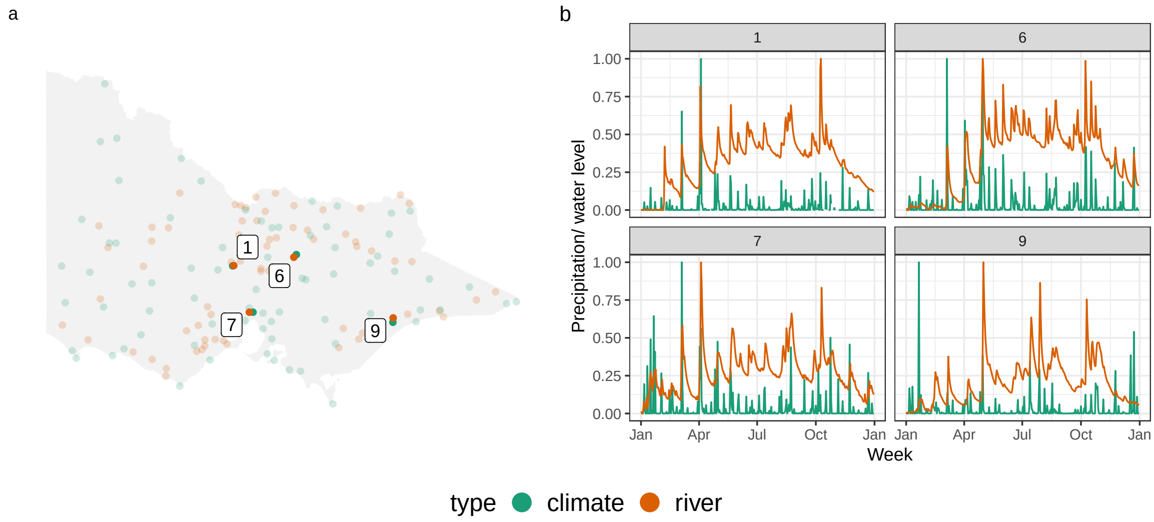

Figure 4: Weather stations and river gauges with matched pairs labelled on the map (a) and plotted across time (b). Precipitation and water level have been standardised between 0 and 1 to be displayed on the same scale. The water level reflects the increase in precipitation. The numbers (1, 7, 6, 9) indicate the group index derived from spatial matching, only those that were selectd by temporal matching are shown here.

Creating the plot requires making transformations on time, unfolding the data for computing the statistic of interest, which is plotted directly as a contour plot with ggplot. Code is provided to accomplish this in the supplementary material.

4.5. Interactive graphics

Interactive graphics can be useful because they make it possible to look at the data in multiple ways on-the-fly. This is especially important for spatio-temporal data, where we would like to interactively connect spatial and temporal displays. This example describes the process of using the **cubble** package with the **crosstalk** package to build an interactive display connecting a map of Australia, with ribbon plots of temperature range observed at the stations. The purpose is to explore the variation of monthly temperature range over the country. Figure 6 shows three snapshots of the interactivity.

The key steps are to convert both the nested and long forms of the data into shared `crosstalk` objects, and to plot these side-by-side. The two are linked by the station identifier.

```
clean <- climate_full |> ...

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

p1 <- nested |> ...
p2 <- long |> ...
```

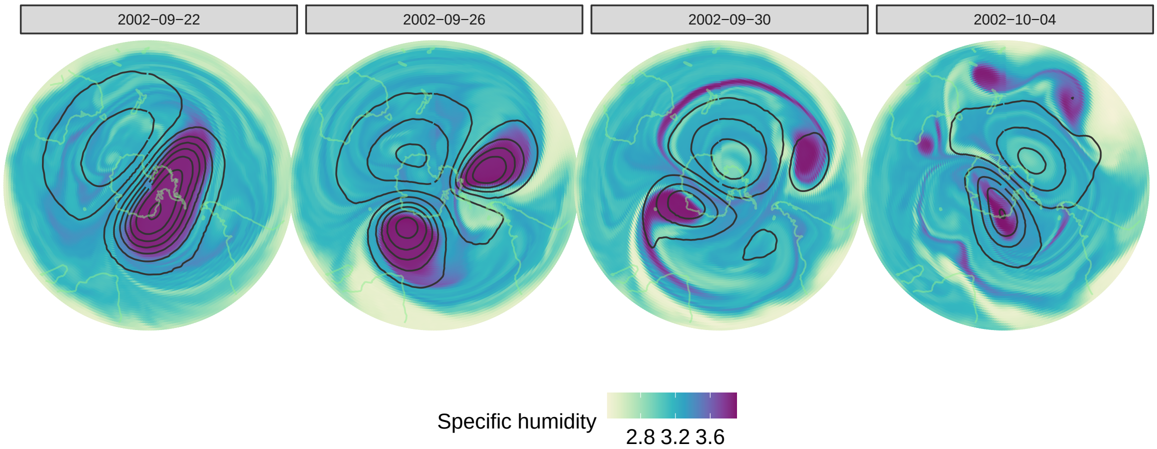

Figure 5: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020) to illustrate the break-up of sourthern polar vortex in late September and early October 2002. The polar vortex, signalled by the high specific humidity, splits into two on 2002-09-26 and further splits into four on 2002-10-04.

```
crosstalk::bscols(plotly::ggplotly(p1), plotly::ggplotly(p2), ...)
```

Plot (a) shows the initial state of the interactive display: all locations are shown as dots on the map, coloured by temperature range, and the right plot shows the ribbons representing maximum to minimum for all stations. In plot (b) the "Mount Elizabeth" station, which shows a high variance colour on the initial map, is selected on the map and this produces the ribbon on the right. In plot (c) the lowest temperature in August is selected, which is "Thredbo" station on the left map. It was surprising to us that this was not a station in Tasmania, so for comparison a station in Tasmania is selected on the map to show in relation to Thredbo. We can see that Thredbo has a bigger winter dip in temperature, and although Tasmania is cold generally, it's temperatures are more constant

5. Conclusion

This paper presents an R package **cubble** for organizing, manipulating and visualizing spatio-temporal data. The package introduces a new data class for spatio-temporal data, `cubble`, that connects the time invariant and varying variables and that allows the user to work with a nested and long form of the data. This work adds capabilities into the spatio-temporal practitioners toolbox to integrate it with a tidy data framework. The data structure and functions introduced in this package can be used and combined with existing spatial data analysis packages such as **sf**, data wrangling packages such as **dplyr**, and visualization packages such as **ggplot2**, **plotly** and **leaflet**.

Numerous examples are provided in the main text, appendix and package vignettes. These include creating and coercing data with mismatched names, handling hierarchical data, matching time series spatially and temporally, reproducing ERA5 plots from NetCDF data. Visualization of the `cubble` objects can be done with interactive graphic pipelines using **crosstalk**,

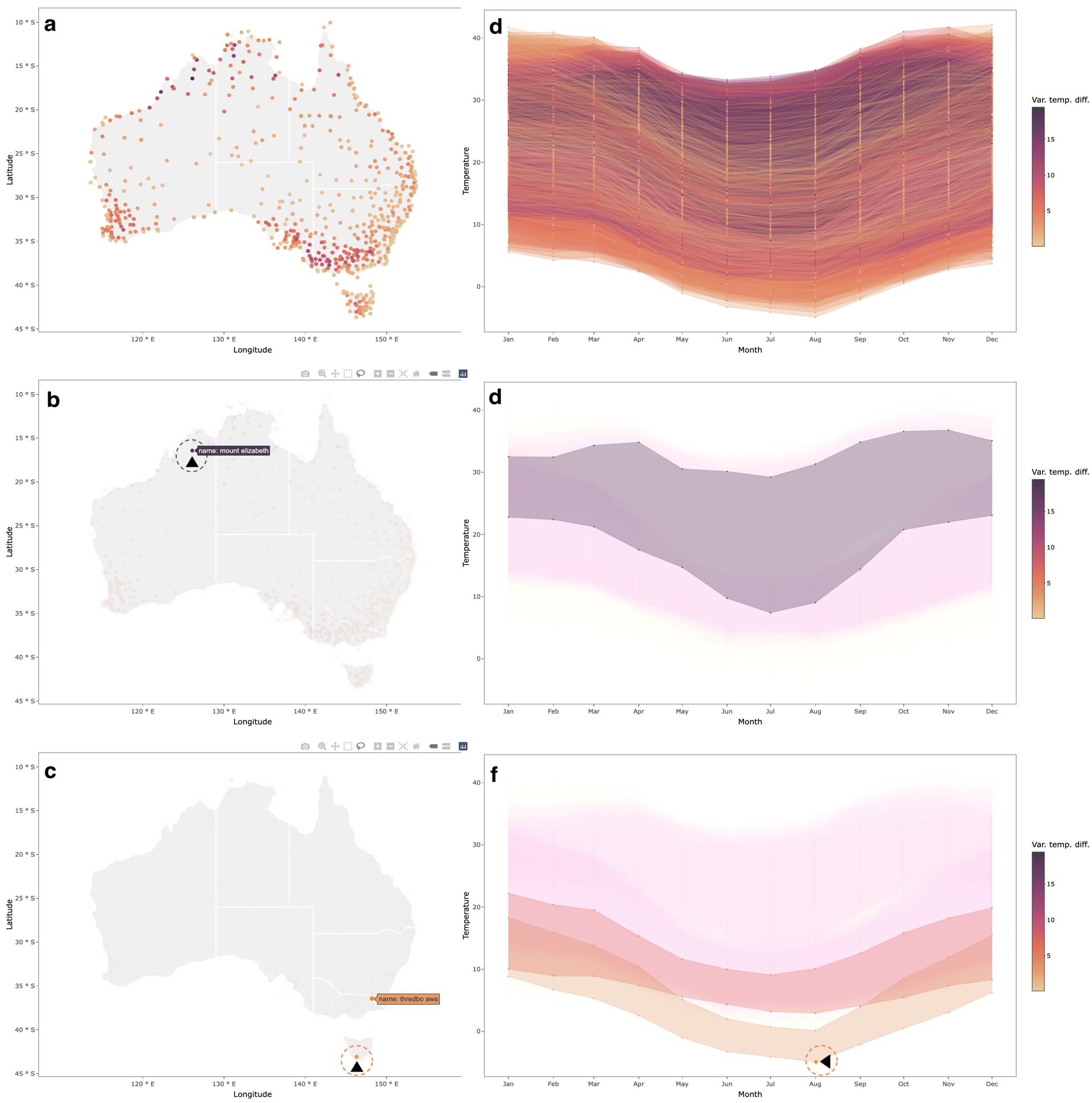

Figure 6: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a particular location on the map (here Mount Elizabeth) produces the plot in the second row. The maximum 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.

plotly and **leaflet**.

There are several possible future directions for the work. The data structure only described fixed spatial sites, and it could be useful to provide tools to accommodate moving coordinates as might be encountered in animal movement data. That could be achieved with a list-column for the location coordinates, and an additional form that these locations can be pivoted into, like the long form for temporal variables. For multiple measured variables, the **cubble** package could be integrated with dimension reduction methods, and dynamic graphics for multiple dimensions such as a tour (Wickham *et al.* 2011).

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`.

Chamberlain S (2021). ***rnoaa**: 'NOAA' Weather Data from R.* R package version 1.3.8, URL `https://CRAN.R-project.org/package=rnoaa`.

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`.

Commonwealth of Australia (2022). "Australia's Official Weather Forecasts & Weather Radar - Bureau of Meteorology." Online; accessed 24 June 2022, URL `http://www.bom.gov.au/`.

Cook D, Swayne D (2007). *Interactive and Dynamic Graphics for Data Analysis with examples using R and GGobi.* Springer-Verlag New York, New York. With contributions from Buja, A., Temple Lang, D., Hofmann, H., Wickham, H. and Lawrence, M. and additional data, R code and demo movies at `http://www.ggobi.org`.

Copernicus Climate Change Service (2022). "Climate Data Store." Online; accessed 24 June 2022, URL `https://cds.climate.copernicus.eu/#!/home`.

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.

Hufkens K, Stauffer R, Campitelli E (2019). "The **ecwmfr** Package: An Interface to ECMWF API Endpoints." URL `https://bluegreen-labs.github.io/ecmwfr/`.

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 (2018). "Simple Features for R: Standardized Support for Spatial Vector Data." *R Journal*, **10**(1), 439.

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`.

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 (2016). ***ggplot2**: Elegant Graphics for Data Analysis.* Springer-Verlag New York. ISBN 978-3-319-24277-4. URL `https://ggplot2.tidyverse.org`.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the **tidyverse**." *Journal of Open Source Software*, **4**(43), 1686. doi:10.21105/joss.01686.

Wickham H, Cook D, Hofmann H, Buja A (2011). "**tourr**: An R Package for Exploring Multivariate Data with Projections." *Journal of Statistical Software*, **40**(2). ISSN 1548-7660. URL http://www.jstatsoft.org/v40/i02/.

Wickham H, François R, Henry L, Müller K (2022). ***dplyr**: A Grammar of Data Manipulation*. R package version 1.0.8, URL https://CRAN.R-project.org/package=dplyr.

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