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

the conceptual framework + nested and long form as per the design vignette

- diagram,
- two forms: long/ nested - spatial/ temporal
- attributes
- headers

The `cubble` class is an S3 class (Wickham 2019) built on the `tibble` class, specifically to organize spatio-temporal data. The `cubble` class uses an attribute "form", to arrange the spatial or temporal data components tidily. The form attribute can take a value of either "nested" or "long". The nested `cubble` is a subclass of rowwise `tibble` (`rowwise_df`). It arranges each spatial site in a row, and uses list columns to store the feature geometry and the temporal information. The long `cubble` is a subclass of grouped `tibble` (`grouped_df`), which expands the temporal information into the long form and stores the spatial information in a "spatial" attribute.

The arguments `key` and `index` follow the conventions in the **tsibble** package to describe the temporal order and multiple series while `coords` specifies the spatial location of each site.

```
R> cb_nested
```

```
# cubble:   key: id [3], index: date, nested form
# extent:   [144.8321, -37.98, 145.0964, -37.6655]
# temporal: 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>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble>
```

```
R> class(cb_nested)
```

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

Printing a `cubble` object provides some information about the data. Here `id` is the variable name to identify each location and there are five unique locations. The bounding box is `[115.97, -32.94, 133.55, -12.42]` and provides information about the coordinates in the data. The third row shows the name and type of all variables nested in the `ts` column. In this example, it includes `date [date], prcp [dbl], tmax [dbl], tmin [dbl]`.

```
R> cb_long
```

```
# cubble:  key: id [3], index: date, long form
# extent:  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(cb_long)
```

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

2.2. Creation and coercion

Creating from separate spatial and temporal table

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

```
# cubble:   key: id [3], index: date, nested form
# extent:   [144.8321, -37.98, 145.0964, -37.6655]
# temporal: 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>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble>
```

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

R> make_cubble(spatial = stations_sf, temporal = meteo_ts)

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

Coercing from foregin objects

Spatio-temporal data in other foreign objects can be coerced into cubble 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 shows the casting from a tibble and a netcdf object to a cubble object.

The dataset `climate_flat` joins the spatial data, `stations`, with the temporal data, `meteo`, into a single tibble object and it can be coerced into a cubble using:

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

# cubble:   key: id [3], index: date, nested form
# extent:   [144.8321, -37.98, 145.0964, -37.6655]
# temporal: 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>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble>
```

The NetCDF data is another format commonly used for storing spatio-temporal data. In R, packages for wrangling NetCDF data include a high-level R interface: **ncdf4** (Pierce 2019), a low-level interface that calls a C-interface: **RNetCDF** (Michna and Woods 2021), and a tidyverse implementation: **tidync** (Sumner 2020). The code below casts a netCDF object in the `ncdf4` class 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
# extent:   [113, -53, 153, -12]
# temporal: 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, one may want to read in a subset of the netcdf data and the argument `vars`, `long_range` and `lat_range` can be used to subset on the variable and the grid resolution:

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

We would recommend reducing to about 300 × 300 grid points for three daily variables in one year. A 300 by 300 spatial grid can be a bounding box of [100, -80, 180, 0] at 0.25 degree resolution or a global bounding box [-180, -90, 180, -90] at 1 degree resolution.

2.3. Functions and methods in cubble

Print all the methods with methods(class = "cubble_df")

dplyr methods

The **dplyr** package has many tools for wrangling tidy data, many of which are useful in the spatio-temporal analysis. The **cubble** package provides methods that support the use of the following operations in the **dplyr** package on both the nested and long forms: `mutate`, `filter`, `summarise`, `select`, `arrange`, `rename`, `left_join`, and the slice family (`slice_*`).

The created `cubble` is a subclass of the `rowwise_df` class where each row forms a group. All the temporal variables are nested in a list column, hence it is also called the nested `cubble`. The rowwise structure makes it simpler to operate on the list using the `mutate()` syntax, which is simpler than the `purr::map()` when working with a list column. For example, calculating the number of rainy days can be done by:

```
R> cb_nested |> mutate(rain_day = sum(ts$prcp != 0))
```

```
# cubble:   key: id [3], index: date, nested form
# extent:   [144.8321, -37.98, 145.0964, -37.6655]
# temporal: prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name          wmo_id rain_day ts
  <chr>       <dbl> <dbl> <dbl> <chr>          <dbl>    <int> <list>
```

```
1 ASN00086038  145. -37.7  78.4 essendon air~  95866        3 <tibble>
2 ASN00086077  145. -38.0  12.1 moorabbin ai~  94870        3 <tibble>
3 ASN00086282  145. -37.7 113.  melbourne ai~  94866        3 <tibble>
```

cubble specific methods

The three cubble specific functions are `face_temporal()`, `face_spatial()`, and `unfold`.

The nested form can be used for those operations where the output is only indexed by the spatial identifier (`key`), but becomes inadequate when outputs need both a spatial and a temporal identifier (`key` and `index`). The `cubble` class also provides a long form, which expands the `ts` column and stores the spatial variables as an attribute. The function `face_temporal()` is used to switch from the nested `cubble` into the long one. The first row in Figure 1 illustrates this operation where the focus of the cube now changes from the site-variable face to the time-variable face. This code switches the `cubble` object just created into its long form:

```
R> (cb_long <- cb_nested |> face_temporal())

# cubble:  key: id [3], index: date, long form
# extent:  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 first line in the header now shows it in the long form and the third line has been changed to display the name and type of spatial variables: `lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]`. Unlike the nested form, the long `cubble` is built from a `grouped_df` class where all the observations from the same site form a group.

Wrangling spatio-temporal data can be seen as an iterative process in the spatial and temporal dimensions. Switching the focus back to the site-variable face can be accomplished by the function `face_spatial()`, which is the inverse of `face_temporal()`. The second row of Figure 1 illustrates the function, which is used as follows:

```
R> (cb_back <- cb_long |> face_spatial())

# cubble:   key: id [3], index: date, nested form
# extent:   [144.8321, -37.98, 145.0964, -37.6655]
# temporal: prcp [dbl], tmax [dbl], tmin [dbl]
  id           long   lat  elev name              wmo_id ts
  <chr>       <dbl> <dbl> <dbl> <chr>              <dbl> <list>
```

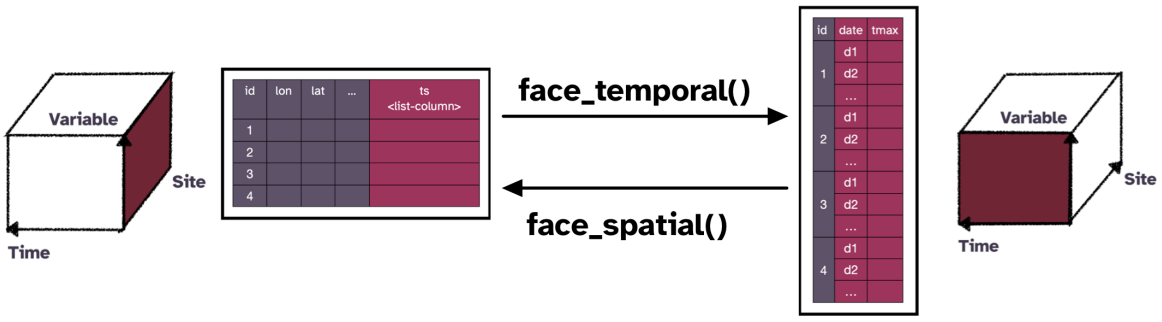

Figure 1: An illustration of function `face_temporal` and `face_spatial`. In the first row, `face_temporal` switches a `cubble` object from the nested form into the long form and the focus has switched from the spatial aspect (the side face) to the temporal aspect (the front face). In the second row, `face_spatial` switches a `cubble` object back to the nested form from the long form and shifts focus back to the spatial aspect.

```
1 ASN00086038  145. -37.7  78.4 essendon airport   95866 <tibble>
2 ASN00086077  145. -38.0  12.1 moorabbin airport  94870 <tibble>
3 ASN00086282  145. -37.7 113.  melbourne airport  94866 <tibble>
```

```
R> #identical(cb_nested, cb_back)
```

Sometimes, analysts may need to apply some variable transformation that involves both the spatial and temporal variables. An example of this is the transformation of temporal variables into the spatial dimension in glyph maps (Wickham *et al.* 2012). (How to make glyph maps will be explained in Section 3.4, and are illustrated in the second example.) This type of operation can be seen as flattening, or *unfolding*, the cube into a 2D data frame. Here the function `unfold()` moves the spatial variables `long` and `lat` into the long `cubble`:

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

```
# cubble:  key: id [3], index: date, long form
# extent:  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 other packages

tsibble

The tsibble class is a subclass of tibble where the index and key components are used to store temporal and strata information, that makes working with temporal data cognitively efficient. A cubble object can use the tsibble class to store the temporal information, and effectively utilize the specialist time series operations in the **tsibble** package. A tsibble object can also be casted into a cubble object through supplying the coordinate information in the argument coords:

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

When a nested cubble is created, each element in the list-column ts is in the tsibble class (labelled tbl_ts) and operations available to the tsibble class are still valid on these elements. For example, the code below calculates two time series features (mean and variance) of maximum temperature, utilizing the tsibble syntax in the cubble object:

```
R> cb %>% face_temporal() %>% tsibble::has_gaps()

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

sf (s2)

add an example on changing CRS?

The sf class is also a subclass of tibble with a specialized feature geometry list-column (sfc) to store different geometry types (POINT, LINESTRING, POLYGON, MULTIPOLYGON, etc). The package **sf** provides functions that operate efficiently on this spatial information. A cubble object can store spatial information in the sf class. Methods for the sfc class can be applied in the nested form of the cubble object. An illustration is in Section 4.1. The spatial information can also be stored as an s2 vector in a cubble object.

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

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

R> #cb %>% sf::st_transform(crs = "EPSG:3857")
```

2.5. Comparison to other spatio-temporal classes

Some readers may question why a new data structure is needed rather than directly creating a list-column on the combined data using `dplyr::nest_by()`. The reason is that the `cubble` object is specifically designed to utilize the spatio-temporal structure when arranging observations in a single object. Moreover, it enables easy pivoting between purely spatial, purely temporal, or unfolded into a combined form.

From the initial **spacetime** (Pebesma 2012) package. New packages, such as **stars** (Pebesma 2021) and **sftime** (Teickner *et al.* 2022), have [. . .]. This section compares and contrast the cubble data structure with other existing alternative, namely **stars** and **sftime**.

The `stars` package creates a `stars` object from an array and a dimension object. In many cases, i.e. satellite images, this is convenient, however, there are also many spatio-temporal data are formatted in 2D data frames.

create a dimension object, populate the data into the array

- array in R allows for character,
- specific datetime class

make the example simpler

- `temporal$var1 %>% matrix(byrow = TRUE, nrow = 2, ncol = 5)`

This happens when different stations have different start date and require other software to reshape the data correctly.

The approach taken by the **sftime** package is to combine the spatial and temporal variables together into a joint table. [for sf objects with daily observation, memory efficiency]

Consider the climate data used in the **cubble** package, `climate_aus`. It contains daily precipitation, minimum and maximum temperature for 639 weather stations across Australia in 2020. The converted sftime object is times the size of the cubble object:

```
R> #climate_aus %>% sftime::st_as_sftime() %>% object.size()
R> climate_aus %>% object.size()

8523040 bytes
```

3. Other features and considerations

3.1. Hierarchical structure

Spatial locations can have grouping structures either inherent to the data e.g., state within country or obtained during the analysis e.g., cluster id. In this case, it can be useful to summarize variables at various levels of the hierarchy. The function `switch_key()` can be used to change the grouping level of spatial locations. The diagram in Figure 2 shows how this function can be used to switch the grouping from station ids to cluster ids. The result

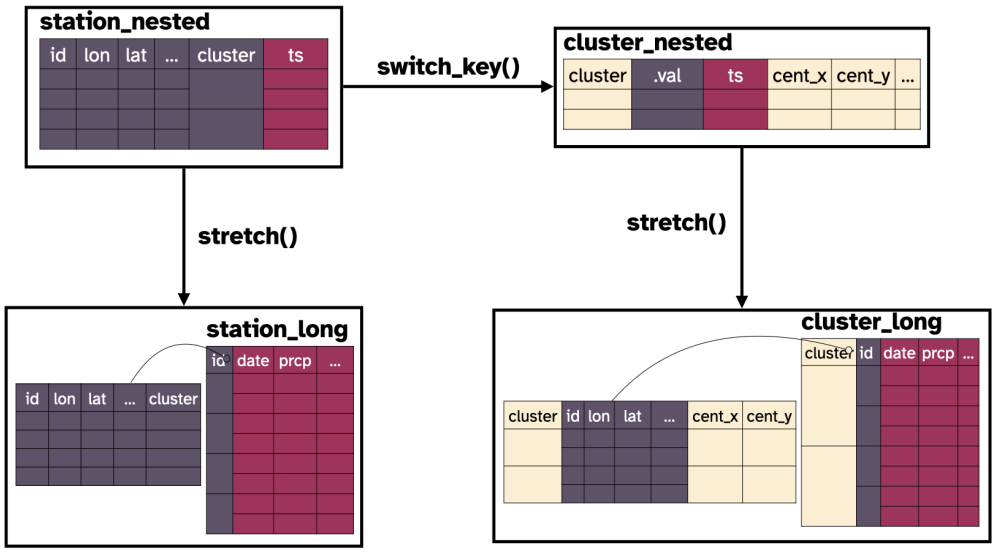

Figure 2: Hierarchical spatial structure can be handled using `switch_key()`, to create summaries based on any level. Here the switch is between the station id and a cluster id. Once the change is made the data can be stretched into the long form.

can also be stretched into long form. By specifying `cluster_nested <- station_nested %>% switch_key(key = cluster)`, the `cubble` object redefines its `key` from the `id` column in `station_nested` to the `cluster` column in `cluster_nested`. All the spatial variables belonging to the `cluster` column are now nested into a `.val` column which allows for summarizing based on cluster.

3.2. Data fusion and matching

One task that may interest spatio-temporal analysts is combining data collected at nearby but not exactly the same sites, for example, weather station measured rainfall and river levels. This can be considered to be a matching problem (Stuart 2010; McIntosh *et al.* 2018) to pair similar time series from nearby locations, or even a data fusion exercise that merges data collected from different sources (Cocchi 2019). The function `match_sites()` in the **cubble** package provides a simple algorithm for this task. The algorithm first matches spatially by computing the pairwise distance on latitude and longitude. Then it matches temporally by computing the number of matched peaks within a fixed length moving window. Figure 3 illustrates this temporal matching. In the two series, A and a, three peaks have been identified in each. An interval, of fixed length, is constructed for each peak in series A, while the peaks in series a are tested against whether they fall into any of the intervals. Here two out of three peaks match. Options for `match_sites()` are:

- `spatial_n_keep`: the number of spatial match for each site to keep;
- `spatial_dist_max`: the maximum distance allowed for a matched pair;

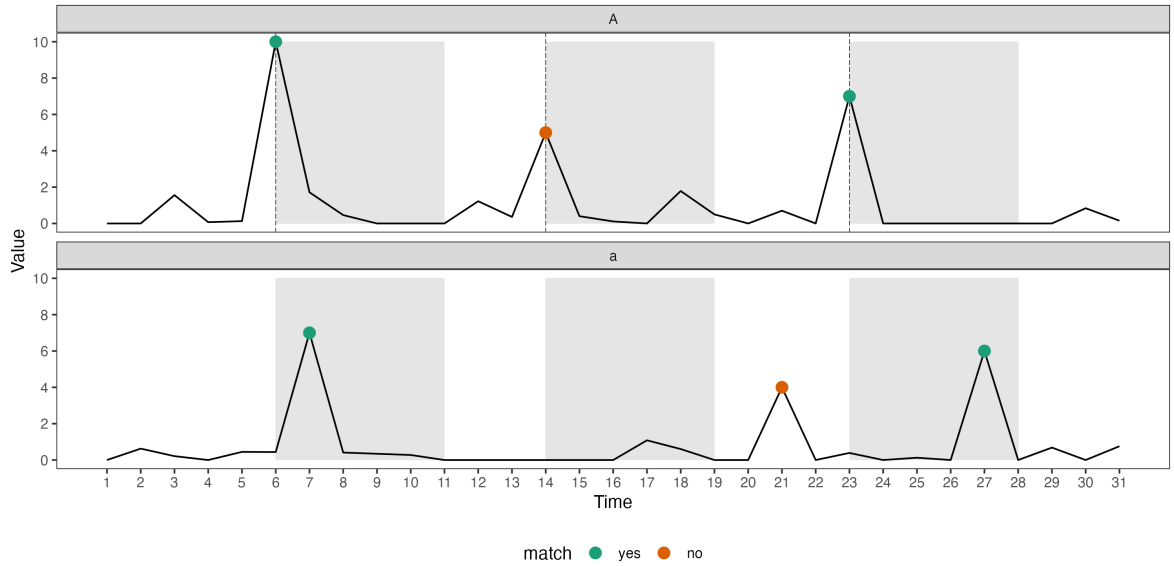

Figure 3: An illustration of temporal matching in the **cubble** package. Three highest peaks are identified in each series and intervals are constructed on series `A`. Two peaks in series `a` fall into the intervals and hence the two series are considered to have two matches.

- `temporal_n_highest`: the number of peaks used - 3 in the example above;
- `temporal_window`: the length of the interval - 5 in the example above; and
- `temporal_min_match`: the minimum number of matched peaks for a valid matched pair.

3.3. 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 4 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.

3.4. Spatio-temporal transformations

Spatio-temporal data lends itself to a range of transformations. Glyph maps (Wickham *et al.*

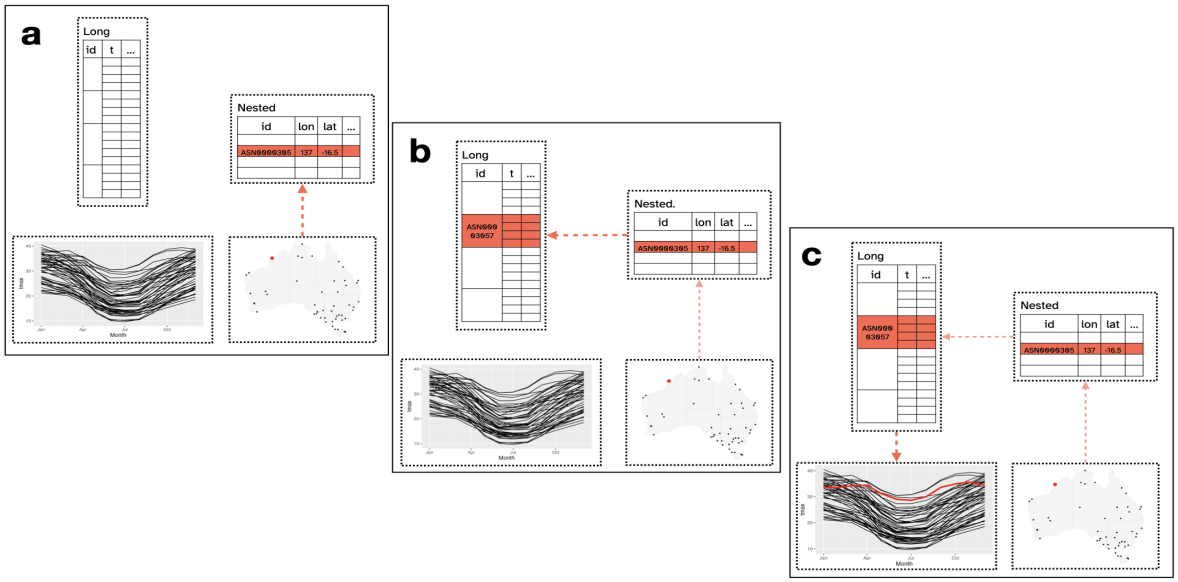

Figure 4: Linking between multiple plots. The line plots and the map are constructed from shared `crosstalk` objects (long and nested `cubble`). When a station is selected on the map (a), the corresponding row in the nested `cubble` will be activated. This will link to all the rows with the same id in the long `cubble` (b) and update the line plot (c).

2012) transform the measured variable and time coordinates into microplots at the spatial locations. Calendar plots (Wang *et al.* 2020) deconstruct time to produce plots of variables in a calendar format. Summarizing multiple variables is commonly done using projections, or linear combinations. Here we elaborate on the transformations made to produce a glyph map.

The package **GGally** (Schloerke *et al.* 2021) has implemented glyph maps through the `glyphs()` function. The function constructs a `data.frame` with calculated position (`gx`, `gy`, `gid`) of each point on the time series using linear algebra (Equations 1 and 2 in Wickham *et al.* (2012)). The data can then be piped into ggplot to create the glyph map as:

```
R> library("ggplot2")
R> gly <- glyphs(data,
+                x_major = ..., x_minor = ...,
+                y_major = ..., y_minor = ..., ...)
R>
R> ggplot(gly, aes(gx, gy, group = gid)) +
+   geom_path()
```

A new implementation of the glyph map as a ggproto, `GeomGlyph`, has been made in the **cubble** package so that a glyph map can be created with `geom_glyph()`:

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

An example using a glyph map is shown in Section 4.2.

Some useful controls over the glyph map are also available in the `geom_glyph()` implementation. Polar coordinate glyph maps are specified using `polar = TRUE`, and arguments `width` and `height` can be specified in either absolute or relative value. Global and local scale is specified with `global_rescale`, which defaults to `TRUE`. Reference boxes and lines can be added with separate `geom_glyph_box()` and `geom_glyph_line()` lines.

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, aggregating information spatially to explore precipitation patterns, 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. (There is an additional example and figures in the Appendix, and more examples in the package vignettes.)

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. Here is how to do this with the **cubble** package, including how to handle mismatches arising from different names of the same LGAs in the two tables. The COVID data is store in a csv file and looks like:

```
R> covid |> head(5)
```

```
# A tsibble: 5 x 5 [1D]
# Key:       lga [1]
# Groups:    lga, source [1]
  date       lga        source                             n roll_mean
  <date>     <chr>      <chr>                          <int>     <dbl>
1 2022-01-01 Alpine (S) Contact with a confirmed case     1        NA
2 2022-01-02 Alpine (S) Contact with a confirmed case     2        NA
3 2022-01-03 Alpine (S) Contact with a confirmed case     4        NA
4 2022-01-04 Alpine (S) Contact with a confirmed case     4        NA
5 2022-01-05 Alpine (S) Contact with a confirmed case     2        NA
```

and the spatial polygons are an ESRI shapefile as follows:

```
R> lga |> head(5)
```

```
Simple feature collection with 5 features and 1 field
Geometry type: MULTIPOLYGON
```

```
Dimension:     XY
Bounding box:  xmin: 142.3535 ymin: -38.67876 xmax: 147.3909 ymax: -36.39269
Geodetic CRS:  WGS 84
               lga                       geometry
132     Alpine (S) MULTIPOLYGON (((146.7258 -3...
133    Ararat (RC) MULTIPOLYGON (((143.1807 -3...
134   Ballarat (C) MULTIPOLYGON (((143.6622 -3...
135    Banyule (C) MULTIPOLYGON (((145.1357 -3...
136 Bass Coast (S) MULTIPOLYGON (((145.5207 -3...
```

The function `make_cubble()` is used to create a `cubble` object from the two spatial and temporal tables, and requires specifying the arguments `key`, `index`, and `coords` (as described in Section 2.2). It will automatically try to match the sites in both tables using the location names and will show a warning message when there are mismatches, as shown below:

```
R> cb <- make_cubble(spatial = lga, temporal = covid)

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
```

It can be seen that there are two-way mismatches – LGAs in the COVID data that do not match with LGAs names in the spatial polygon data, and vice versa. The mismatches can be identified by using the function `check_key`:

```
R> (check_res <- check_key(spatial = lga, temporal = covid))

$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"
```

With this information both tables can be fixed, to create the desired `cubble` object, as follows:

```
R> lga2 <- lga |>
+   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]
# extent:   [140.961682, -39.1591895, 149.976291, -33.9804256]
# temporal: source [chr], n [int], roll_mean [dbl]
  lga             long   lat                       geometry ts
  <chr>          <dbl> <dbl>             <MULTIPOLYGON [°]> <list>
1 Alpine (S)      147. -36.9 (((146.7258 -36.45922, 146.80~ <gropd_ts>
2 Ararat (RC)     143. -37.5 (((143.1807 -37.73152, 143.17~ <gropd_ts>
3 Ballarat (C)    144. -37.5 (((143.6622 -37.57241, 143.66~ <gropd_ts>
4 Banyule (C)     145. -37.7 (((145.1357 -37.74091, 145.13~ <gropd_ts>
5 Bass Coast (S)  146. -38.5 (((145.5207 -38.30667, 145.54~ <gropd_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))
```

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

```
R> tmax |>
+   face_spatial() |>
+   mutate(n = nrow(ts)) |>
+   arrange(n) |>
+   pull(n) |>
+   head(10)

 [1] 12 12 12 13 19 19 20 24 24 24

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

In order to create a glyph map displaying the monthly series (Figure 5), 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.

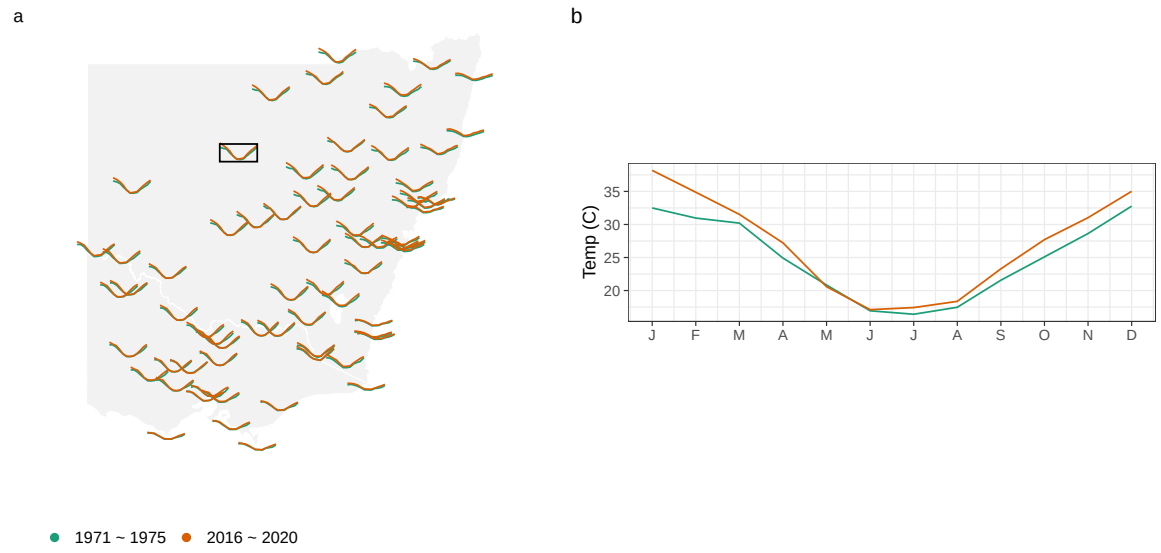

Figure 5: A glyph map of the monthly maximum average temperature for weather stations in Victoria and New South Wales (NSW) for the periods (1971-1975, 2016-2020). The corresponding average time series for the cobar station are display on the top left corner. From the glyph map we can observe that the 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.

```
R> nsw_vic <- ozmaps::abs_ste |>
+   filter(NAME %in% c("Victoria", "New South Wales"))
R>
R> ggplot() +
+   geom_sf(data = nsw_vic,
+           fill = "transparent", color = "grey",
+           linetype = "dotted") +
+   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

River level and rainfall data for the same areas should have some similarity. Here we examine the river gauge data (`Water_course_level`) from the Bureau of Meteorology (Commonwealth

of Australia 2022) in relation to weather station rainfall from NOAA's climate data (`climate`). The goal is to match water gauges with nearby weather stations, spatially and temporally, using the `match_sites()` function.

This function requires passing the major and minor data sets used for matching, in this case those are `river` and `climate`. The variables used for the temporal matching are `Water_course_level` from the `river` data set and `prcp` in the climate data set. The rest of the arguments, as explained in Section 3.2, correspond to the maximum and minimum number of peaks in the time series to be matched. In this example those are set to be a maximum of 30 and a minimum of 15 (approximately 2 and 1 per month).

```
R> res <- match_sites(
+   river, climate,
+   temporal_by = c("Water_course_level" = "prcp"),
+   temporal_independent = "prcp",
+   temporal_n_highest = 30,
+   temporal_min_match = 15,
+ )
```

This function returns a `cubble` object, with additional columns: `dist` storing the distance between matched stations, `group` summarizing spatial matching, and `n_match` showing the temporal matching.

Figure **??** shows four matched pairs on the map (a) and standardized data as time series (b). The expected concurrent increase in precipitation and water level can be seen clearly.

4.4. ERA5: climate reanalysis data

Figure 6 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))
```

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.

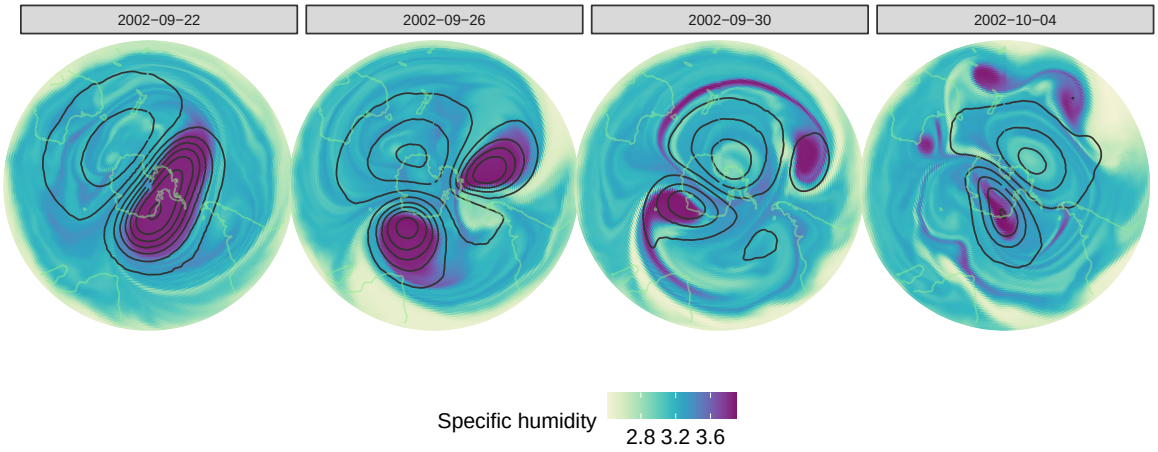

Figure 6: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020) to illustrate the break-up of sourthern polar vortex in late September and early October 2002. The polar vortex, signalled by the high specific humidity, splits into two on 2002-09-26 and further splits into four on 2002-10-04.

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 7 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 |> ...

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**, **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.

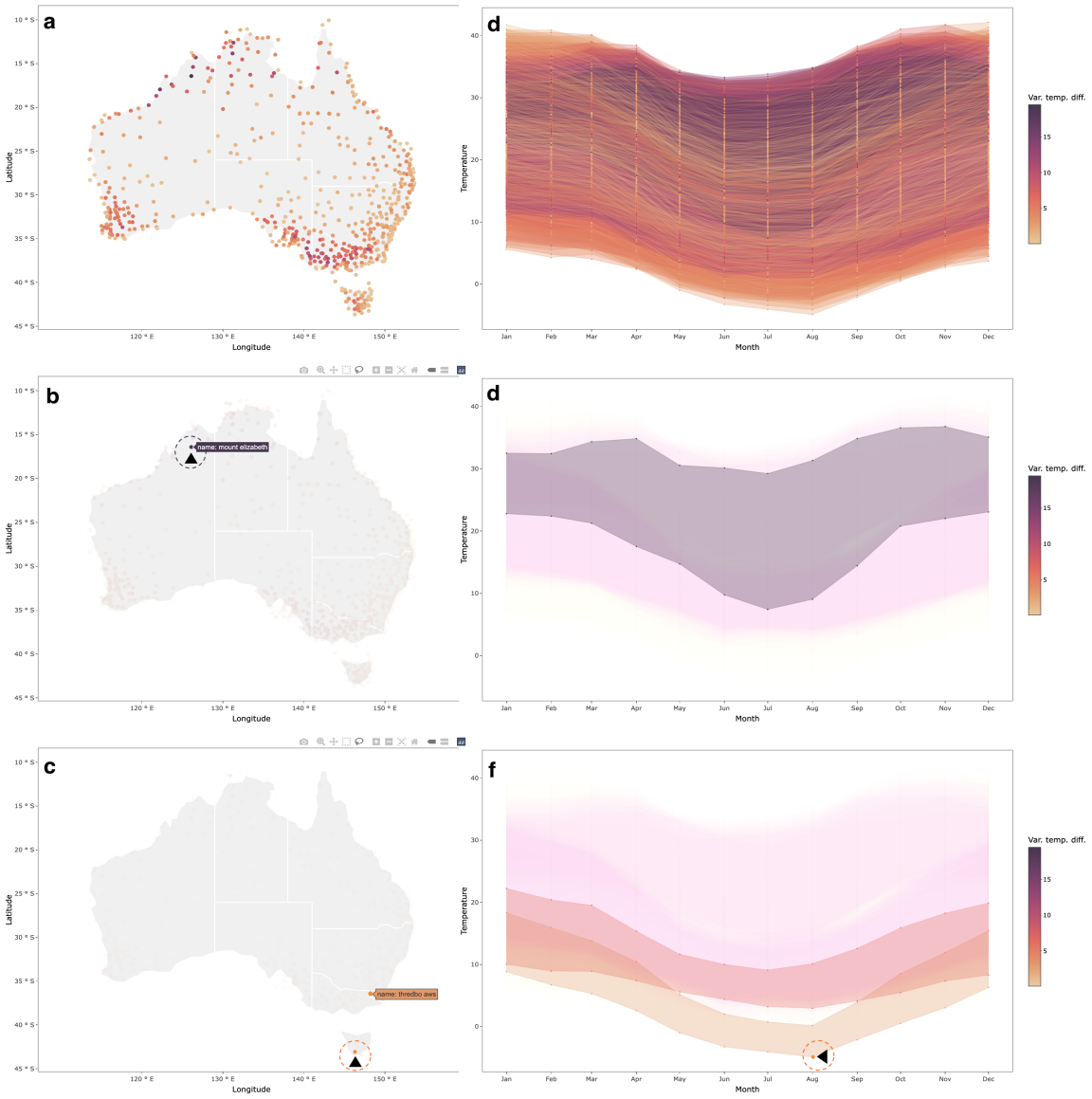

Figure 7: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a particular location on the map (here Mount Elizabeth) produces the plot in the second row. The maximum 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.

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.

Cocchi M (2019). *Data Fusion Methodology and Applications*. Elsevier.

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/.

McIntosh AI, Jenkins HE, White LF, Barnard M, Thomson DR, Dolby T, Simpson J, Streicher EM, Kleinman MB, Ragan EJ, *et al.* (2018). "Using Routinely Collected Laboratory Data to Identify High Rifampicin-Resistant Tuberculosis Burden Communities in the Western Cape Province, South Africa: A Retrospective Spatiotemporal Analysis." *PLoS Medicine*, **15**(8).

Michna P, Woods M (2021). ***RNetCDF**: Interface to 'NetCDF' Datasets*. R package version 2.5-2, URL https://CRAN.R-project.org/package=RNetCDF.

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.

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.

Stuart EA (2010). "Matching Methods for Causal Inference: A Review and a Look Forward." *Statistical Science*, **25**(1), 1.

Sumner M (2020). ***tidync**: A Tidy Approach to NetCDF Data Exploration and Extraction*. R package version 0.2.4, URL https://CRAN.R-project.org/package=tidync.

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). "Calendar-based Graphics for Visualizing People's Daily Schedules." *Journal of Computational and Graphical Statistics*, **29**(3), 490–502.

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 (2019). *Advanced R*. CRC press. ISBN 978-0815384571. URL https://adv-r.hadley.nz/rcpp.html.

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