cubble An R Package for Structuring Spatio-temporal Data

A Preprint

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

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

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

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

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

April 20, 2022

Abstract

Spatio-temporal data refer to measurements taken across space and time. In practice, spatio-temporal data can be decomposed into a spatial and temporal component: at one time, we would select a spatial location and inspect the temporal trend; at other time, we might select one or multiple time value(s) and explore the spatial distribution. Ideally, we could make multiple maps and multiple time series to explore these together, however, doing all of these actions is complicated when data arrive fragmented in multiple objects. To make it easy to do all these tasks, ideally spatial and temporal variables are in a single data object that we can slice and dice in different ways to conduct different visualisations. This work suggests a new data structure, `cubble`, that uses a nested form and a long form to organise spatial and temporal variables in a single data object. The new data structure ensures that data in the two forms are synchronised while been flexible to explore the two aspect of spatio-temporal data. It also provides capability for handling data with hierarchical structure, matching data from multiple sources, constructing interactive graphics, and performing spatio-temporal transformation. The paper will demonstrate **cubble** with examples using Australian climate weather stations, river level data, and climate reanalysis (ERA5) data.

***K*eywords** 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, a temporal component that is recorded at regular or irregular time intervals, and may have multiple variables measured at each spatial and temporal value. The analysis of spatio-temporal data can fix the time to explore the spatial distribution, fix the spatial location to explore temporal trends, or dynamically explore the space and time simultaneously. For these tasks to be carried out easily, there is a need for a spatio-temporal data representation that underpins various operations.

The *Representing data* section of the SpatioTemproal CRAN task view (Edzer Pebesma 2022) lists existing spatio-temporal data representation. Among them, E. Pebesma (2012) proposes four layouts: full grid layout, sparse grid layout, irregular layout, and trajectory. These layouts are implemented in the **spacetime** (E. Pebesma 2012) package with the underlying spatial and temporal class: **sp** (E. Pebesma and Bivand 2005) and **xts** (Ryan and Ulrich 2020). The **stars** (E. Pebesma 2021) package has been a recent implementation to handle both raster and vector spatio-temporal data using array. It depends on the newer spatial package **sf** (E. J. Pebesma 2018) and interfaces with the widely-used tidyverse verbs.

[till here] One issue with wrangling spatio-temporal data is that the raw data does not naturally structured in any one of the representation above. Also, in data collection, spatio-temporal data is less often collected as a whole piece but fragmented spatial and temporal data for analysts to ensemble. For example, map data records the shape of a collection of area of interest is provided by official government website; geo-scientific data records the longitude and latitude coordinate of locations in the area along with other meta data relates to the location; and temporal variables indexed by both location and time. For spatio-temporal data analysis, merging all these sources can be challenging and joining these tables can go wrong because of the slightest unmatched of the linking variable from different sources.

Another issue is how the concept of tidy data (Wickham 2014) should be applied to spatio-temporal data. The tidy data concept advocates the data to be shaped 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 given the downstream software i.e. **dplyr** and **ggplot2** for data wrangling and visualisation. However, the long form can be inefficient to store feature geometries, especially for large multipolygon over long time period. This poses the question on how to arrange spatial and temporal variables that makes analysing spatio-temporal data easier.

This paper presents a new R package, **cubble**, to [overcome the problem] with organising variables in different levels when working with spatio-temporal data. The data structure focuses on connecting to the upstream data collection and is suitable for temporal data collected at the fixed location over time. Among the four spacetime layouts in E. Pebesma (2012), it can be full grid layout, sparse grid layout, or irregular layout, but trajectory is not within the scope of this work. In the package a new data structure, also called `cubble`, is proposed to organise spatial and temporal variables as two forms of a single data object so that they can be wrangled separately while synchronised as a whole piece. With this new data object, analysts can spend less time writing codes to organise spatial and temporal variables and focus more on the exploratory data analysis itself. The software is available from the Comprehensive R Archive Network (CRAN) at [CRAN link].

The rest of the paper is divided as follows: Section 2 discusses the spatio-temporal cube to conceptualise spatio-temporal data. Section 3 presents the main design and functionality of **cubble**. Section 4 explains how cubble deals with more advanced considerations including data with hierarchical structure, data matching, how cubble fits with existing static and interactive visualisation tools, and spatio-temporal data transformation. Section 5 uses Australia weather station data and river level data as examples to demonstrate the use of **cubble**. An example on how **cubble** handles NetCDF data is also provided. Section 6 concludes the paper.

2 Conceptual framework: spatio-temporal cube

Spatio-temporal data can be conceptualised using a cubical data model with reference to variables, time and space. This naturally motivates the usage of array to represent and store spatio-temporal data, as evident by satellite imageries, large climate models, among others. The work by Lu, Appel, and Pebesma (2018) has discussed the suitability of this representation through array algebra and practical spatio-temporal tasks. SZ: is this too direct?

The cubical conceptual framework provides a generalisation to the operation can be done on the data for visualisation. A paper by Bach et al. (2014) has reviewed the temporal data visualisation based on space-time

cube operations. Notice that the term space-time cube in their article "does not need to involve spatial data", but referring to "an abstract 2D substrate that is used to visualize data at a specific time". Despite its main focus is on temporal data, the mindset of abstracting out data representation to construct visualisation is applicable to our spatio-temporal data manipulation and visualisation.

3 The cubble package

This section will first introduce the core functions in the cubble class: `as_cubble()`, `face_spatial()`, `face_temporal()`, and `unfold()` from subsection 3.1 to 3.4. The next subsection addresses why bother a new class before subsection 3.6 finishing on the compatibility of cubble with existing packages in spatial and temporal analysis.

Each core function is introduced with a short example accompanied and the data `climate_flat` is used throughout this section. `climate_flat` contains five weather stations in Australia with spatial information of each station: station id, latitude, longitude, elevation, station name and World Meteorology Organisation ID and also daily temporal information of date, maximum and minimum temperature and precipitation for 2020. Below prints out the first five rows of `climate_flat`:

```
## # A tibble: 1,830 x 10
##   id            lat  long  elev name         wmo_id date        prcp  tmax  tmin
##   <chr>       <dbl> <dbl> <dbl> <chr>         <dbl> <date>     <dbl> <dbl> <dbl>
## 1 ASN00009021 -31.9  116.  15.4 perth airpo~  94610 2020-01-01     0  31.9  15.3
## 2 ASN00009021 -31.9  116.  15.4 perth airpo~  94610 2020-01-02     0  24.9  16.4
## 3 ASN00009021 -31.9  116.  15.4 perth airpo~  94610 2020-01-03     6  23.2  13
## 4 ASN00009021 -31.9  116.  15.4 perth airpo~  94610 2020-01-04     0  28.4  12.4
## 5 ASN00009021 -31.9  116.  15.4 perth airpo~  94610 2020-01-05     0  35.3  11.6
## # ... with 1,825 more rows
```

3.1 Create a cubble in the nested form

As described in the introduction, spatio-temporal data can come in various formats and shapes. A cubble can be created from various raw data, including tibble and its variates multiple tables (an example provided in section 5.1), and netcdf (detailed in section 3.6.4). The function `as_cubble()` is used to create a cubble with three additional components: `key` as the spatial identifier; `index` as the temporal identifier; and a vector of `coords` in the order of longitude and latitude. Argument `key` and `index` follow the wording in **tsibble** to define temporal order and multiple series while `coords` is a space specific argument to locate each site in two columns. The code below creates a cubble out of `climate_flat` with `id` as the key, `date` as the index, and `c(long, lat)` as the coordinates:

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

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name           wmo_id ts
##   <chr>       <dbl> <dbl> <dbl> <chr>           <dbl> <list>
## 1 ASN00009021 -31.9  116.  15.4 perth airport   94610 <tibble [366 x 4]>
## 2 ASN00010311 -31.9  117. 179   york            94623 <tibble [366 x 4]>
## 3 ASN00010614 -32.9  117. 338   narrogin        94627 <tibble [366 x 4]>
## 4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tibble [366 x 4]>
## 5 ASN00015131 -17.6  134. 220   elliott         94236 <tibble [366 x 4]>
```

The cubble header provides some information about this data. `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 the names of variable nested in the `ts` column, along with its type, are `date [date], prcp [dbl], tmax [dbl], tmin [dbl]`.

The created cubble is built from a `rowwise_df` class where each row forms a group and 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 calculate on the list using `mutate()` and avoid the `map()` when working with list column. For example calculating the number of raining day can be done by:

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name           wmo_id ts       rain_day
##   <chr>       <dbl> <dbl> <dbl> <chr>           <dbl> <list>      <int>
## 1 ASN00009021 -31.9  116.  15.4 perth airport   94610 <tibble>      104
## 2 ASN00010311 -31.9  117. 179   york            94623 <tibble>       89
## 3 ASN00010614 -32.9  117. 338   narrogin        94627 <tibble>       90
## 4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tibble>      106
## 5 ASN00015131 -17.6  134. 220   elliott         94236 <tibble>       63
```

3.2 Switch from the nested cubble into the long form

The nested format is convenient for those operations whose output does not contain a time dimension. For those outputs that are cross-identified by the spatial and temporal identifier, a long cubble needs to be used. The function `face_temporal()` switches the cubble from the nested form to the long form and the first row in Figure 1 sketches the function in a diagram. This code shows how to switch the cubble just created into its long form:

```
cubble_long <- cubble_nested |> face_temporal()
cubble_long
```

```
## # cubble:  date, id [5]: long form
## # bbox:    [115.97, -32.94, 133.55, -12.42]
## # spatial: lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]
##   id          date        prcp  tmax  tmin
##   <chr>       <date>     <dbl> <dbl> <dbl>
## 1 ASN00009021 2020-01-01     0  31.9  15.3
## 2 ASN00009021 2020-01-02     0  24.9  16.4
## 3 ASN00009021 2020-01-03     6  23.2  13
## 4 ASN00009021 2020-01-04     0  28.4  12.4
## 5 ASN00009021 2020-01-05     0  35.3  11.6
## # ... with 1,825 more rows
```

Notice that the third line in the header now shows the name and type of spatial variables: `lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]` and these variables are now stored as a `spatial` attribute of the data:

```
attr(cubble_long, "spatial")
```

```
## # A tibble: 5 x 6
##   id            lat  long  elev name           wmo_id
##   <chr>       <dbl> <dbl> <dbl> <chr>           <dbl>
## 1 ASN00009021 -31.9  116.  15.4 perth airport   94610
## 2 ASN00010311 -31.9  117. 179   york            94623
## 3 ASN00010614 -32.9  117. 338   narrogin        94627
## 4 ASN00014015 -12.4  131.  30.4 darwin airport  94120
## 5 ASN00015131 -17.6  134. 220   elliott         94236
```

The long form is built form class `groupped_df` where all the observations from the same sites forms a group.

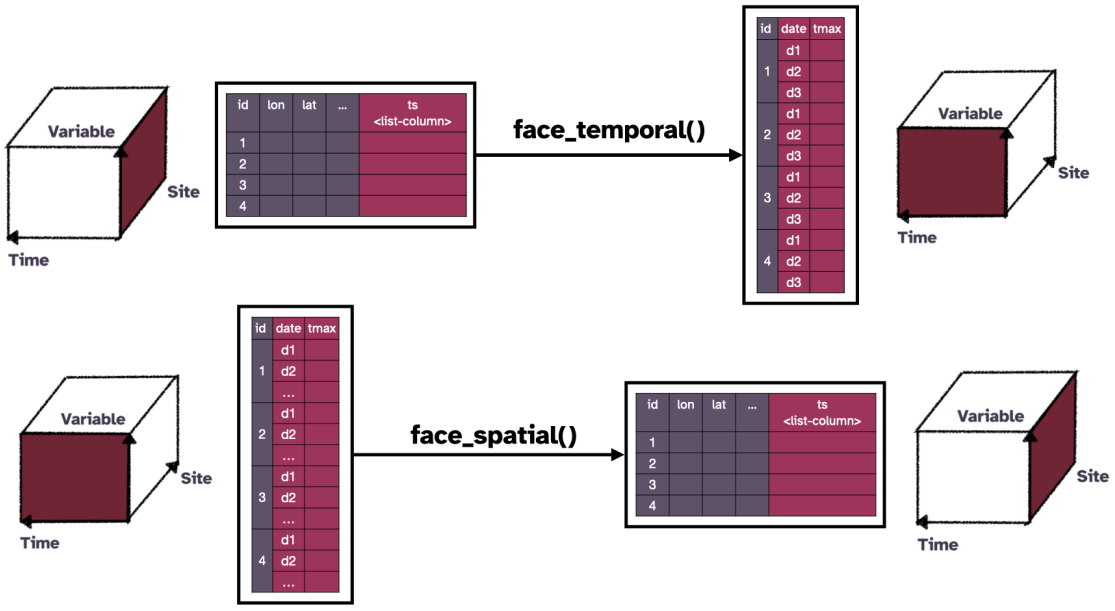

Figure 1: An illustration of function `face_temporal` and `face_spatial` in cubble. In the first row, `face_temporal` switches a cubble from the nested form into the long form and the focus has switched from the spatial aspect (the side face) to the temporal aspect (the front face). In the second row,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

3.3 Switch a long cubble back to the nested form

Manipulation on the spatial and temporal dimension can be an iterative process and analysts may need to go back and forth between the nested and long cubble. `face_spatial()`, the inverse of `face_temporal()`, switches the long cubble back to the nested form, as sketched in the second row in Figure 1:

```
cubble_back <- cubble_long |> face_spatial()
cubble_back
```

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name           wmo_id ts
##   <chr>       <dbl> <dbl> <dbl> <chr>           <dbl> <list>
## 1 ASN00009021 -31.9  116.  15.4 perth airport   94610 <tibble [366 x 4]>
## 2 ASN00010311 -31.9  117. 179   york            94623 <tibble [366 x 4]>
## 3 ASN00010614 -32.9  117. 338   narrogin        94627 <tibble [366 x 4]>
## 4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tibble [366 x 4]>
## 5 ASN00015131 -17.6  134. 220   elliott         94236 <tibble [366 x 4]>
```

3.4 Flatten spatial variables into the long cubble

Sometimes, analysts may need to apply some variable transformation that involves both the spatial and temporal variable. An example of this is the transformation of temporal variables into the spatial dimension in glyph maps, which will be elaborated in section 4.4. This type of operations 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:

```
cubble_unfold <- cubble_long |> unfold(long, lat)
cubble_unfold
```

```
## # cubble:  date, id [5]: long form
## # bbox:    [115.97, -32.94, 133.55, -12.42]
## # spatial: lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]
##   id          date        prcp  tmax  tmin  long   lat
##   <chr>       <date>     <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ASN00009021 2020-01-01     0  31.9  15.3  116. -31.9
## 2 ASN00009021 2020-01-02     0  24.9  16.4  116. -31.9
## 3 ASN00009021 2020-01-03     6  23.2  13    116. -31.9
## 4 ASN00009021 2020-01-04     0  28.4  12.4  116. -31.9
## 5 ASN00009021 2020-01-05     0  35.3  11.6  116. -31.9
## # ... with 1,825 more rows
```

This function should generally be used in the last step of the analysis since it is a temporary operation, meaning these added spatial variables are not stored in the long form and will disappear if switched to the nested form and then switched back:

```
cubble_unfold |> face_spatial() |> face_temporal()
```

```
## # cubble:  date, id [5]: long form
## # bbox:    [115.97, -32.94, 133.55, -12.42]
## # spatial: lat [dbl], long [dbl], elev [dbl], name [chr], wmo_id [dbl]
##   id          date        prcp  tmax  tmin
##   <chr>       <date>     <dbl> <dbl> <dbl>
## 1 ASN00009021 2020-01-01     0  31.9  15.3
## 2 ASN00009021 2020-01-02     0  24.9  16.4
## 3 ASN00009021 2020-01-03     6  23.2  13
## 4 ASN00009021 2020-01-04     0  28.4  12.4
## 5 ASN00009021 2020-01-05     0  35.3  11.6
## # ... with 1,825 more rows
```

3.5 Why not just use `dplyr::nest_by()`

It may naturally raise the question that why bother a new data structure since the nested form can already be created by the **dplyr** function `nest_by()`? While `nest_by()` and `unnest()` can mimic the nested form in cubble, cubble can be seen as an attempt to arrange variables from different observational units into a single object, specifically for spatio-temporal data. While `nest_by()` solves this problem on the spatial side, it doesn't solve the temporal side. For temporal data, a combined representation with spatial variables is useful for spatio-temporal operations but its duplication can be a burden to operate on. Also, spatial variables are dropped when summarised and you will need to either specified all of them in the `group_by()` before `summarise`, or add a joining after `summarise()`.

3.6 Compatibility with existing packages

The previous four subsections have introduced operations specific to the `cubble` class and this section will demonstrate how the `cubble` class interacts with existing packages commonly used in spatial and temporal analysis, specifically, `dplyr`, `tsibble`, `sf` (`s2`), and `netcdf4`.

3.6.1 dplyr

The `dplyr` package has provided many tools for data wrangling tasks and these operations are useful in the spatio-temporal context. `cubble` provides methods that support the following `dplyr` verbs in both the nested and long form:

> `mutate`, `filter`, `summarise`, `select`, `arrange`, `rename`, `left_join`, and the slice family (`slice_head`, `slice_tail`, `slice_sample`, `slice_min`, `slice_max`)

3.6.2 tsibble

`tsibble` is a temporal data structure that uses `index` and `key` to identify the time and different series. `cubble` can be seen as following the same vein as `tsibble` for spatio-temporal data. This makes it easy to cast a `tsibble` into a `cubble` as only the `coords` argument needs to be supplied:

```
# example with a tsibble created from climate_flat
raw <- climate_flat |> tsibble::as_tsibble(key = id, index = date)
dt <-  raw |> cubble::as_cubble(coords = c(long, lat))
dt
```

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name           wmo_id ts
##   <chr>       <dbl> <dbl> <dbl> <chr>           <dbl> <list>
## 1 ASN00009021 -31.9  116.  15.4 perth airport   94610 <tbl_ts [366 x 4]>
## 2 ASN00010311 -31.9  117. 179   york            94623 <tbl_ts [366 x 4]>
## 3 ASN00010614 -32.9  117. 338   narrogin        94627 <tbl_ts [366 x 4]>
## 4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tbl_ts [366 x 4]>
## 5 ASN00015131 -17.6  134. 220   elliott         94236 <tbl_ts [366 x 4]>
```

In the nested cubble created, each element in the list-column `ts` is of `tbl_ts` class and operations available to the tsibble class is still valid under cubble. For example, the code below calculates two features of the maximum temperature:

```
# add station-based features in the nested form.
dt |> mutate(fabletools::features(ts, tmax, list(tmax_mean = mean, tmax_var = var)))
```

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name          wmo_id ts       tmax_mean tmax_var
##   <chr>       <dbl> <dbl> <dbl> <chr>          <dbl> <list>       <dbl>    <dbl>
## 1 ASN00009021 -31.9  116.  15.4 perth airport  94610 <tbl_ts>      25.7    38.6
## 2 ASN00010311 -31.9  117. 179   york           94623 <tbl_ts>      26.2    51.1
## 3 ASN00010614 -32.9  117. 338   narrogin       94627 <tbl_ts>      23.7    45.4
## 4 ASN00014015 -12.4  131.  30.4 darwin airpo~  94120 <tbl_ts>      33.1     3.02
## 5 ASN00015131 -17.6  134. 220   elliott        94236 <tbl_ts>      34.6    24.7
```

3.6.3 sf and s2

As a spatial data object, `sf` creates a simple feature geometry list-column (`sfc`) in the data frame to provide spatial operations on various geometry types (`POINT`, `LINESTRING`, `POLYGON`, `MULTIPOLYGON`, etc). These spatial operations are also valuable for spatio-temporal data analysis, but an `sf` object *usually* does not contain temporal variables. This means `sf` cannot be directly cast into a `cubble`, however, `cubble` does support `sfc` columns in the nested form and spatial operations applied to the `sfc` column in `sf` can still be applied to the `sfc` column in a cubble. The following example shows how to create an `sfc` column of `POINT` type from latitude and longitude in cubble. Then `sf::st_within` is used to add the state `MULTIPOLYGON` of each weather station before a coordinate transformation is made.

```
library(sf)
# create a cubble
cb <- climate_flat |>
  cubble::as_cubble(key = id, index = date, coords = c(long, lat))

aus <- ozmaps::abs_ste

dt <- cb |>
  mutate(
```

```
    # create `sfc` column based on long and lat
    ll = st_sfc(
      purrr::map2(long, lat, ~st_point(c(.x, .y))),
      crs = st_crs(aus)),

    # append state multi-polygon based on the `sfc` created
    state = aus$geometry[st_within(ll, aus, sparse = FALSE)],

    # adopt a different projection: lambert conformal conic (EPSG:3112)
    state = st_transform(state, crs = "EPSG:3112")
    )

dt
```

```
## # cubble:   id [5]: nested form
## # bbox:     [115.97, -32.94, 133.55, -12.42]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name   wmo_id ts                       ll
##   <chr>       <dbl> <dbl> <dbl> <chr>   <dbl> <list>           <POINT [°]>
## 1 ASN00009021 -31.9  116.  15.4 perth~  94610 <tibble> (115.9764 -31.9275)
## 2 ASN00010311 -31.9  117. 179   york    94623 <tibble>  (116.765 -31.8997)
## 3 ASN00010614 -32.9  117. 338   narro~  94627 <tibble> (117.1797 -32.9342)
## 4 ASN00014015 -12.4  131.  30.4 darwi~  94120 <tibble> (130.8925 -12.4239)
## 5 ASN00015131 -17.6  134. 220   ellio~  94236 <tibble> (133.5407 -17.5521)
## # ... with 1 more variable: state <MULTIPOLYGON [m]>
```

An **s2** `lnglat` vector can similarly be created as an `sfc` in cubble before using any `s2`-prefixed function:

```
library(s2)
# Western Australia map
wa <- ozmaps::abs_ste |> filter(NAME == "Western Australia")

# mutate a `s2_lnglat` vector on `cb` created in the last chunk
cb |>
  mutate(ll = s2_lnglat(long, lat)) |>
  filter(s2_within(ll, wa))
```

```
## # cubble:   id [3]: nested form
## # bbox:     [115.97, -32.94, 117.18, -31.89]
## # temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
##   id            lat  long  elev name          wmo_id ts                  ll
##   <chr>       <dbl> <dbl> <dbl> <chr>          <dbl> <list>             <s2_lng>
## 1 ASN00009021 -31.9  116.  15.4 perth airport  94610 <tibble [366 x 4]> (115.97~
## 2 ASN00010311 -31.9  117. 179   york           94623 <tibble [366 x 4]> (116.76~
## 3 ASN00010614 -32.9  117. 338   narrogin       94627 <tibble [366 x 4]> (117.17~
```

3.6.4 netcdf

NetCDF data has two main components: *dimension* for defining the spatio-temporal grid (longitude, latitude, and time) and *variable* that populates the defined grid. Attributes are usually associated with dimension and variable in the NetCDF format data and a metadata convention for climate and forecast has been designed to standardise the format of the attributes. A few packages in R exists for manipulating NetCDF data and this includes a high-level R interface: **ncdf4** (Pierce 2019), a low-level interface that calls C-interface: **RNetCDF** (Michna and Woods 2021, 2013), and a tidyverse implementation: **tidync** (Sumner 2020).

Cubble provides an `as_cubble()` method to coerce the `ncdf4` class from the **ncdf4** package into a `cubble`. It maps each combination of longitude and latitude into an `id` as the `key`:

```
# read in the .nc file as a ncdf4 class
raw <- ncdf4::nc_open(here::here("data/era5-pressure.nc"))

# convert the variable q and z in the ncdf4 into a cubble
dt <- as_cubble(raw, vars = c("q", "z"))
```

The memory limit with NetCDF data in cubble depends on the number of longitude grid points times the number of latitude grid points times the number of time grid points times the number of variables. Cubble can handle slightly more than 300 x 300 (longitude x longitude) grid points for 3 variables in one year. The spatial grid can be reduced to trade for longer time periods and more variables. 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. Subsetting longitude and latitude grids is available through `long_range` and `lat_range` if the NetCDF file has finer resolution than needed.

```
# Assume my_ncdf has bounding box of [-180, -90, 180, -90]
# at 0.25 degree resolution and subset it to have
# 1 degree resolution:
dt <- as_cubble(my_ncdf, vars = c("q", "z"),
                long_range = seq(-180, 180, 1),
                lat_range = seq(-90, 90, 1))
```

4 Other features and considerations

4.1 Hierarchical structure

Spatial locations can have grouping structure either inherent to the data or formed by clustering. Rather than analysing variables in the site level, summarised variables in the cluster level can give a crisper picture of local areas. In cubble, `switch_key()` can be used to create a new level of grouping of spatial locations by specifying a clustering variable. Figure 2 illustrates the relationship of cubbles at station and cluster level, in both the long and nested form. By specifying `cluster_nested <- station_nested %>% switch_key(key = cluster)`, the cubble re-defines the cubble key from `id` in `station_nested` to `cluster` in `cluster_nested`. All the spatial variables variant to `cluster` are now nested into a `.val` column and cluster level variables can be computed in the same fashion as station level variables in `station_nested`.

4.2 Data fusion and matching

One task that may interest analysts in spatio-temporal data is to find how similar the time series from nearby sites are. This can be seen as a matching problem (Stuart 2010; McIntosh et al. 2018) that pairs up similar similar time series in nearby locations or a data fusing exercise that merges data collected from different sources (Cocchi 2019). `match_sites()` in **cubble** provides a simple algorithm for this task. The algorithm first matches the two data sources spatially through computing the pairwise distance on latitude and longitude. Pairs that pass the spatial matching are then matched temporally through computing the number of matched peaks within a fixed length window. Figure 3 illustrates this temporal matching in more details. Given two series `A` and `a`, 3 peaks have been picked in each series. An interval, with default length of 5, is constructed for each peak in series `A` and the peaks in series `a` are tested against whether they fall into the any of the intervals. In this illustration, there are 2 matches for these two series. Several arguments are available in `match_sites()` to fine-tune the matching:

- `spatial_n_keep`: the number of spatial match for each site to keep
- `spatial_dist_max`: the maximum distance allowed for a matched pair
- `temporal_n_highest`: the number of peak used - 3 in the example above
- `temporal_window`: the length of the interval - 5 in the example above
- `temporal_min_match`: the minimum number of matched peak for a valid matched pair

4.3 Interactive graphics

Cubble fits in naturally with the interactive graphic pipeline discussed in the literature (Buja, Asimov, and Hurley 1988; Buja, Cook, and Swayne 1996; Sutherland et al. 2000; Xie, Hofmann, and Cheng 2014; Cheng,

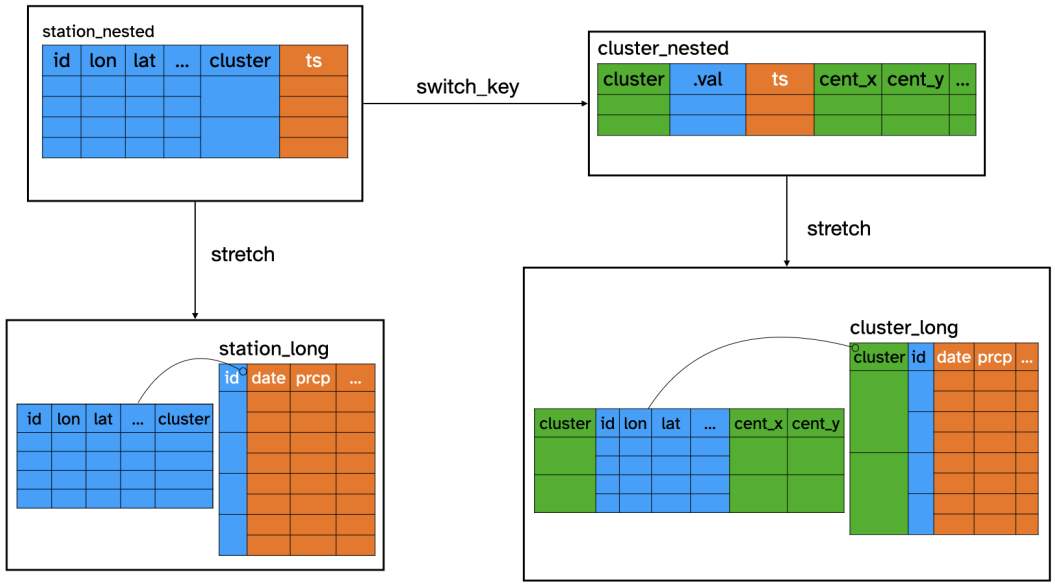

Figure 2: An illustration of the original and cluster level cubble in the nested form long form for hierarchical structure data. `switch_key()` changes the station level cubble into a cluster level cubble and both can be stretched into the long form.

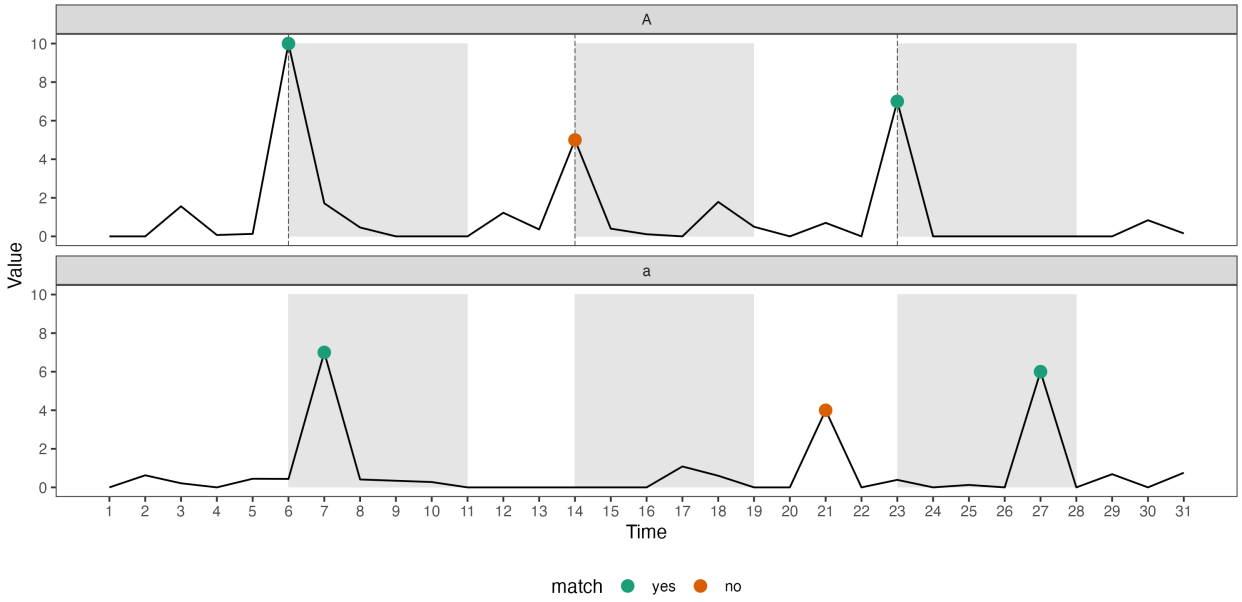

Figure 3: An illustration of temporal matching in cubble. Three highest peaks are identified in each series and intervals are constructed on series `A`. Two peaks in series `a` fall into the intervals and hence the two series are considered to have 2 matches.

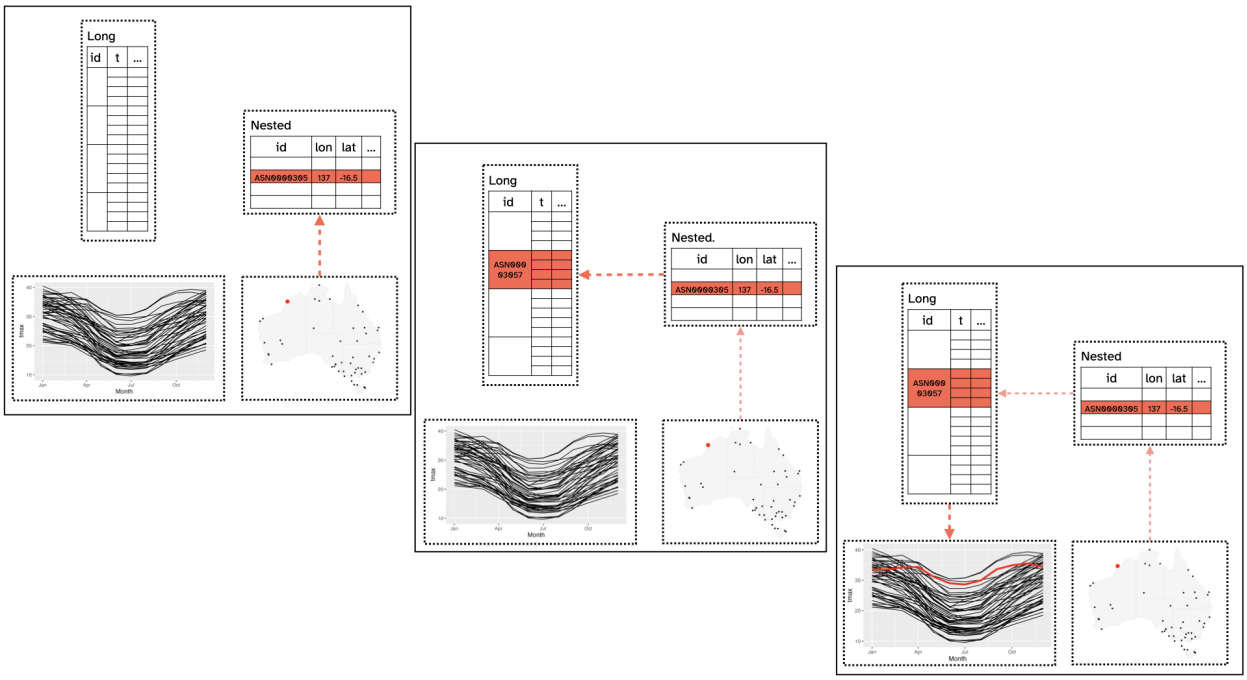

Figure 4: An illustration of the data model under interactive graphics with cubble. The line plot and the map is made separately with the long and nested cubble. When a station is selected on the map (left), the corresponding row in the nested cubble will be activated. This will link to all the rows with the same id in the long cubble (middle) and update the line plot (right).

Cook, and Hofmann 2016). Diagram 4 illustrates how linking works from the map to the time series in cubble. The map and time series plot is associated with the nested or long cubble, respectively, and when a user action is captured on the map, the site will be activated in the nested cubble (left). The nested cubble will communicate to the long cubble to activate all the observations with the same `id` (middle). The long cubble will then highlight the activated series in the time series plot (right).

The linking is also available from the time series plot to the map. The selection(s) on the time series is through selecting the point(s) on the time series and once a point is selected, it will be activated in the long cubble. All the observations that share the same `id` are then activated and this includes other points in the same time series in the long cubble and the corresponding observation of site in the nested cubble. These activated observations will then being reflected in the updated plots and Diagram 12 in the Appendix illustrates this process.

4.4 Spatio-temporal transformations

Spatio-temporal transformation is a useful technique to extract information form spatio-temporal data. Glyph maps (Wickham et al. 2012) transform the time coordinates into the space coordinates to plot the time series of different locations on the map. Calendar plots (Wang, Cook, and Hyndman 2020) reconstructs time into calendar-based grid to discover weekday and weekend pattern. Projection, or linear combination, of variables summarises multivariate information into lower dimension to further digest. This section elaborates on the glyph map.

In R, **GGally** implements 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 (Equation 1 and 2 in Wickham et al. (2012)). The data can then be piped into `ggplot` to create the glyph map as:

```
library(ggplot2)
gly <- glyphs(data,
              x_major = ..., x_minor = ...,
              y_major = ..., y_minor = ..., ...)
```

```
ggplot(gly, aes(gx, gy, group = gid)) +
  geom_path()
```

A re-implementation of the glyph map as a ggproto, `GeomGlyph`, has been made in the **cubble** package and now the glyph map can be created with `geom_glyph()`:

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

Some useful controls over the glyph map is also available in the `geom_glyph()` implementation. Polar glyph map can be specify as a parameter, `polar = TRUE`. in the `geom_glyph()`, along with `width` and `height` in either absolute or relative value. Global and local scale can be controlled by the parameter `global_rescale`. which default to `TRUE` for global scaling. Reference box and line can be added with separate `geom_glyph_box()` and `geom_glyph_line()`.

5 Examples

5.1 Victoria covid data

This example illustrate casting different sources into a cubble object. Victoria government of Australia provides daily COVID case number by date, source, and local government area (LGA). This data can be useful for visualising COVID spread when combined with LGA map information, which is available from Australian Bureau of Statistics. Variable `covid` and `lga` show the first 5 rows of these two data:

```
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
```

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

```
## Simple feature collection with 5 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 142.3643 ymin: -38.67876 xmax: 147.386 ymax: -36.39269
## Geodetic CRS:  WGS 84
##   lga_code_2018            lga state_code_2016 state_name_2016 areasqkm_2018
## 1         20110     Alpine (S)               2        Victoria     4788.1568
## 2         20260    Ararat (RC)               2        Victoria     4211.1171
## 3         20570   Ballarat (C)               2        Victoria      739.0321
## 4         20660    Banyule (C)               2        Victoria       62.5402
## 5         20740 Bass Coast (S)               2        Victoria      865.8095
##   cent_long  cent_lat                       geometry
## 1  146.9742 -36.85357 MULTIPOLYGON (((146.7258 -3...
## 2  142.8432 -37.47271 MULTIPOLYGON (((143.1807 -3...
## 3  143.7815 -37.49286 MULTIPOLYGON (((143.6622 -3...
## 4  145.0851 -37.73043 MULTIPOLYGON (((145.1357 -3...
## 5  145.5581 -38.50730 MULTIPOLYGON (((145.5207 -3...
```

To cast these two data into a cubble object, supply them as a list with name `spatial` and `temporal` and other arguments (`key`, `index`, and `coords`). `as_cubble()` will automatically check the match of spatial and temporal information in the two datasets and warnings will emit if a location has missing spatial or temporal information.

```
cb <- as_cubble(list(spatial = lga, temporal = covid),
                key = lga, index = date, coords = c(cent_long, cent_lat))
```

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

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

## ! Use argument `output = "unmatch"` to check on the unmatched key
```

Here, some locations have been detected with this issue and you can use `output = "unmatch"` to check on these locations:

```
pair <- as_cubble(list(spatial = lga, temporal = covid),
                key = lga, index = date, coords = c(cent_long, cent_lat),
                output = "unmatch")

pair
```

```
## $paired
## # A tibble: 2 x 2
##   spatial             temporal
##   <chr>               <chr>
## 1 Kingston (C) (Vic.) Kingston (C)
## 2 Latrobe (C) (Vic.)  Latrobe (C)
##
## $others
## $others$temporal
## [1] "Interstate"       "Overseas"         "Queenscliffe (B)" "Unknown"
##
## $others$spatial
## character(0)
```

cubble will make an attempt to pair the unmatched sites as well as showing unpaired locations. These can be useful information to help clean up the data.

```
lga <- lga %>%
  mutate(lga = ifelse(lga == "Kingston (C) (Vic.)", "Kingston (C)", lga),
         lga = ifelse(lga == "Latrobe (C) (Vic.)", "Latrobe (C)", lga)) %>%
  filter(!lga %in% pair$others$spatial)

covid <- covid %>% filter(!lga %in% pair$others$temporal)
```

After cleaning, you can cast the data into cubble with the same syntax without warnings:

```
cb <- as_cubble(data = list(spatial = lga, temporal = covid),
                key = lga, index = date, coords = c(cent_long, cent_lat))
```

5.2 Australia historical maximum temperature

Global Historical Climatology Network (GHCN) provides daily climate measures from stations across the world. The dataset `weatherdata::historical_tmax` extracts the maximum temperature for 236 Australian stations from GHCN with starting from year 1969. `weatherdata::historical_tmax` is already in a cubble, with `id` as the key, `date` as the index, and `c(longitude, latitude)` as the coordinates. This example compares the maximum temperature in two periods: 1971 - 1975 and 2016 - 2020 for stations in Victoria and New South Wales.

Stations in the two states can be subsetted by the station number: Australia GHCN station number starts with "ASN00" and followed by the Bureau of Meteorology (BOM) station number, where the 2nd and 3rd digit (7th and 8th in the GHCN number) define the state of the station. New South Wales stations start from 46 to 75 and Victoria stations follow from 76 to 90. Filtering Victoria and New South Wales stations is an operation in the spatial dimension and hence uses the nested form:

```
tmax <- weatherdata::historical_tmax |>
  filter(between(stringr::str_sub(id, 7, 8), 46, 90))
```

Filtering for the period 1971 ~ 1975 and 2016 ~ 2020 is an operation on the time dimension and the nested cubble needs to be switched to the long cubble by `stretch()`:

```
tmax <- tmax |>
  face_temporal() |>
  filter(lubridate::year(date) %in% c(1971:1975, 2016:2020))
```

A monthly average is used for both periods to smooth the maximum temperature and it is again an operation on the time dimension:

```
tmax <- tmax |>
  group_by(month = lubridate::month(date),
           group = as.factor(ifelse(lubridate::year(date) > 2015,
                                    "2016 ~ 2020", "1971 ~ 1975"))) |>
  summarise(tmax = mean(tmax, na.rm = TRUE))
```

A few stations don't have records during 1971 - 1975 and further investigation shows that while the first and last year of each series is recorded, the missing years in this period is not reported. These stations are filtered out by examining whether the summarised time series has a total of 24 months. The long cubble needs to be switched to the nested form for this spatial operation using `face_spatial()`:

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

Lastly, to create a glyph map, both the major (`longitude`, `latitude`) and minor (`month`, `tmax`) coordinates need to be in the same table. Spatial variables can be moved to the long form with `migrate()`:

```
tmax <- tmax |> face_temporal() |> unfold(latitude, longitude)
```

`tmax` can then be supplied to `geom_glyph()` for the glyph map in Figure 5 with a station inset on the top left corner:

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

ggplot() +
  geom_sf(data = nsw_vic,
          fill = "transparent", color = "grey", linetype = "dotted") +
  geom_glyph(data = tmax,
             aes(x_major = longitude, x_minor = month,
                 y_major = latitude, y_minor = tmax,
                 group = interaction(id, group), color = group),
             width = 1, height = 0.5) +
  ...
```

5.3 Australia precipitation pattern in 2020

In the previous example, there has already been some overlapping of the glyphs for a few stations near (151E, 34S) and (152E, 33S) and this will be a problem when mapping more stations in the national level. Aggregation can be helpful to group series into clusters before visualising the cluster with glyph map. This example shows how to organise data at both level with `switch_key()`.

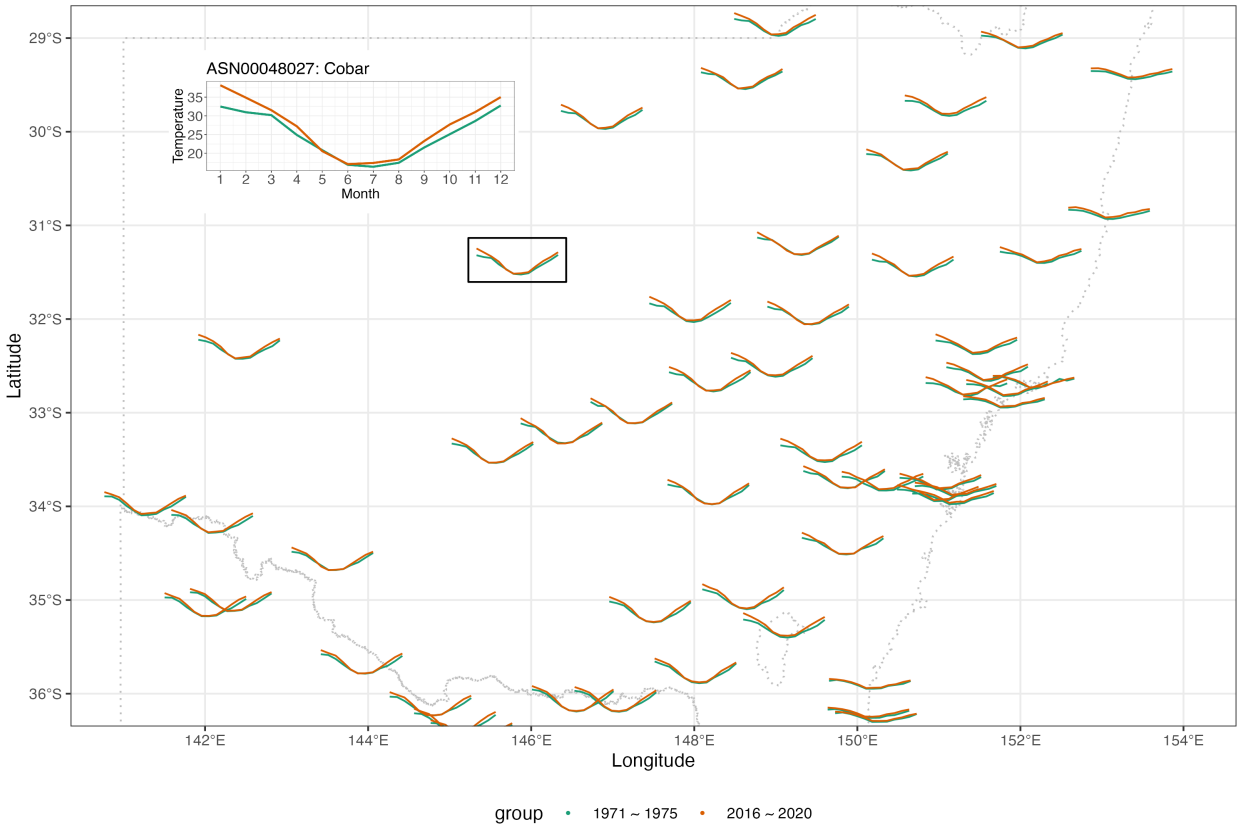

Figure 5: A glyph map of the average maximum temperature by month of Victoria and New South Wales weather stations in Australia. On the top left corner is an insetted plot of station Cobar highlighted in the black box.

Figure 6: Profile of aggregated precipitation from 639 weather stations in Australia. Subplot A shows the glyph map of the weekly averaged precipitation of each cluster. The group number of printed in the middle of y minor axis and can be used as a reference line to read the magnitude. Subplot B shows the station membership of each cluster.

`weatherdata::climate_full`, also extracted from GHCN, records daily precipitation and maximum/ minimum temperature for 639 stations in Australia from 2016 to 2020. A simple kmean algorithm based on the distance matrix is used to create 20 clusters. This creates `station_nested` as a station level nested cubble with a cluster column indicating the group each station belongs to. More complex algorithms can be used for other problem, as long as there is a mapping from each station to a cluster.

```
station_nested <- weatherdata::climate_full |>
  mutate(cluster = ...)
```

To create a group level cubble, use `switch_key()` with the new key variable, `cluster`:

```
cluster_nested <- station_nested |> switch_key(cluster)
```

With the group level cubble, `get_centroid()` is useful to compute the centroid of each cluster, which will be used as the major axis for the glyph map later:

```
cluster_nested <- cluster_nested |> get_centroid()
```

Long form cubble at both levels can be accessed through stretching the nested form and with access to both station and cluster level cubbles, various plots can be made to understand the cluster. Figure 6 shows two example plots that can be made with this data: subplot A is a glyph map made with the cluster level cubble in the long form and subplot B inspects the station membership of each cluster using the station level cubble in the nested form.

5.4 River level data in Victoria water gauges

Bureau of Meteorology collects water data from river gauges and this includes variables: electrical conductivity, turbidity, water course discharge, water course level, and water temperature. In particular, water level will interactive with precipitation from the climate data since rainfall will raise the water level in the river. Figure 7 gives the location of available weather station and water gauges in Victoria.

From the map, a few water gauges and weather stations are close to each other and the fluctuation of the water level could be matched up with precipitation measured by the climate station. As introduced in

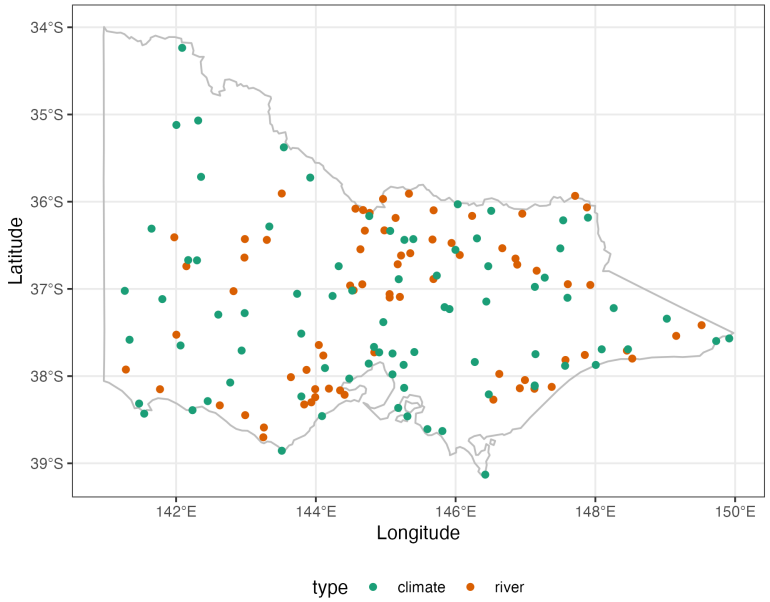

Figure 7: Location of weather stations and river gauges in Victoria, Australia.

Section 3.2, `match_sites()` can be used to match one source of data with another source in a cubble. Here `Water_course_level` in `river` will be matched to `prcp` in `climate` in 2020. The two datasets need to be specified as the first two arguments and the variable to match can be specified in `temporal_by` using the `by` syntax in `join`. `temporal_independent` controls the variable used to construct the interval and the goal here is to see if precipitation will be reflected by the water level in the river. This puts precipitation `prcp`, as the independent. Given there is one year worth of data, the number of peak (`temporal_n_highest`) to consider is slightly raised from a default 20 to 30 and `temporal_min_match` is raised accordingly To return all the pairs of the match, `temporal_min_match` can be set to 0.

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

The output from matching is also a cubble, with additional column `dist` and `group` produced from spatial matching and `n_match` from temporal matching.

```
## # cubble:   id [8]: nested form
## # bbox:     [144.52, -37.73, 146.06, -36.55]
## # temporal: date [date], matched_var [dbl]
##   id          name                  lat  long type   dist group ts       n_match
##   <chr>       <chr>                <dbl> <dbl> <chr> <dbl> <int> <list>     <int>
## 1 405234      SEVEN CREEKS @ D/S~ -36.9  146. river  6.15     5 <tibble>      21
## 2 404207      HOLLAND CREEK @ KE~ -36.6  146. river  8.54    10 <tibble>      21
## 3 ASN00082042 strathbogie         -36.8  146. clim~  6.15     5 <tibble>      21
## 4 ASN00082170 benalla airport     -36.6  146. clim~  8.54    10 <tibble>      21
## 5 230200      MARIBYRNONG RIVER ~ -37.7  145. river  6.17     6 <tibble>      19
## 6 ASN00086038 essendon airport    -37.7  145. clim~  6.17     6 <tibble>      19
## 7 406213      CAMPASPE RIVER @ R~ -37.0  145. river  1.84     1 <tibble>      18
## 8 ASN00088051 redesdale           -37.0  145. clim~  1.84     1 <tibble>      18
```

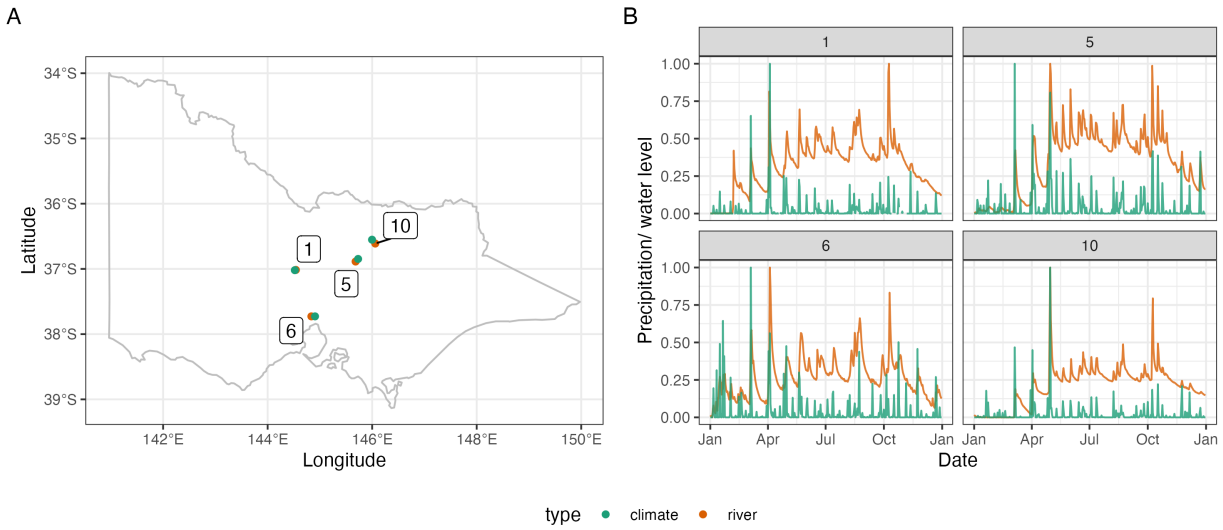

Figure 8: Matched weather stations and river gauges on the map (A) and across time (B). Precipitation and water level have been standardised between 0 and 1 to be displayed on the same scale. The increases in the precipitation is reflected by the water level.

Figure 8 plots the matched pairs on the map or to view the matched series. There are four pairs of matches, which all locates in the middle Victoria and the concurrent increase of precipitation and water level can be observed.

5.5 ERA5: climate reanalysis data

ERA5 data (Hersbach et al. 2020) is the latest reanalysis of global atmosphere, land surface, and ocean waves from 1950 onwards and is available in the NetCDF format from European Centre for Medium-Range Weather Forecasts (ECMWF). The data can be directly downloaded from Copernicus Climate Data Store (CDS) website or programmatically via an R package **ecmwfr** (Hufkens, Stauffer, and Campitelli 2019). The `era5-pressure` data contains variable *specific humidity* and *geopotential* on the 10 hPa pressure level on four dates: 2002-09-22, 2002-09-26, 2002-09-30, and 2002-10-04. Once downloaded, the data can be read into a cubble as:

```
raw <- ncdf4::nc_open(here::here("data/era5-pressure.nc"))
dt <- as_cubble(raw, vars = c("q", "z"))
```

Figure 9 reproduces the ERA5 data row of Figure 19 in Hersbach et al. (2020). It shows the southern polar vortex splits into two on 2002-09-26 and further splits into four on 2002-10-04 in the stratosphere. Readers interested in the analysis of this figure can refer to Hersbach et al. (2020), Simmons et al. (2020) and Simmons et al. (2005) for more details.

5.6 Interative graphic

With spatio-temporal data, users may wish to make plots to learn the spatial distribution of a variable, or to find patterns, such as trend or seasonality, in the time series. Combining this two types of plot with interactivity let users to link between points on the map and the corresponding time series to explore the spatial and temporal dimension of the data simultaneously. Below is an example that describes the process of building an interactive graphic with **cubble** and **crosstalk** The example explores the variation of monthly temperature range with `weatherdata::climate_full` data.

The temperature range is calculated as the difference between `tmax` and `tmin` and its monthly average over 2016 - 2020 is taken before calculating the variance. A `SharedData` object is constructed for each form of

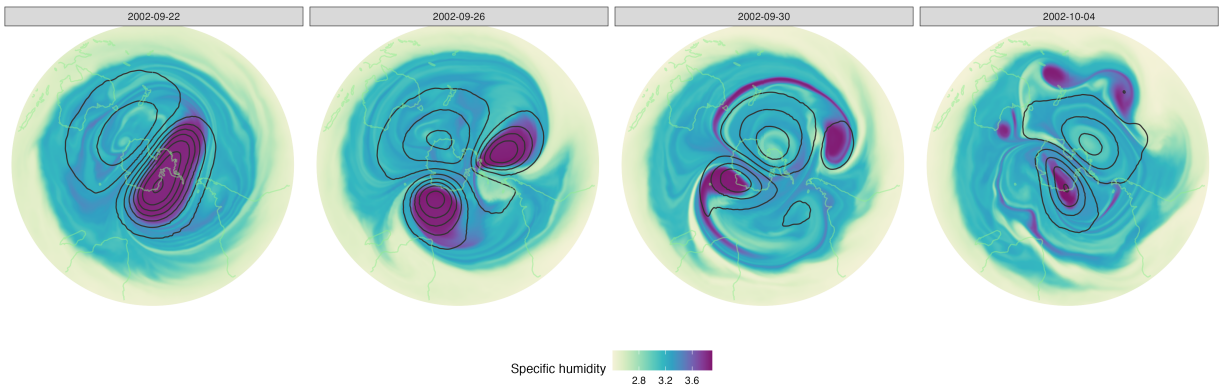

Figure 9: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020).

the cubble and the same `group` argument ensures the cross-linking of the two forms via the common `id` column. The spatial map and time series plot are then made with each `SharedData` objects separately. In this example, stations on the Australia map, made from the nested form, are coloured by the calculated variance and a ribbon band is constructed using the long form cubble to show the maximum and minimum temperature of each station across month. With a different dataset, users are free to calculate any per station measure in the nested form or to make any time-wise summary of the data in the long form to customise the spatial or temporal view. The cross-linking between the two plots is always safeguarded by the shared `id` column embedded in the cubble structure. Below is the pseudo code that outlines the process to construct an interactive graphic described above:

```
# data pre-processing
clean <- weatherdata::climate_full |> ...

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

# create the spatial and temporal view each with a ShareData instance
p1 <- nested |> ...
p2 <- long |> ...

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

In Figure 10, the first row shows the initial view of the interactive graphic. On the map, most regions in Australia have low variance of temperature range while the north-west coastline, bottom of South Australia, and Victoria stands out with larger monthly changes. In the second row, Mount Elizabeth is selected on the map given its high variance colour on the initial map and this links to the ribbon on the right. The third row the lowest temperature in August and this corresponds to Thredbo AWS in the Victoria and New South Wales border. Another station in the Tasmania island is selected on the map to cross compare with Thredbo AWS.

This plot can also be made using **cubble** and **leaflet** where the temperature range can be displayed as a small subplot upon clicking on the map. This would require first creating the popup plots from the long form cubble as a vector and then add these plots to a leaflet map created from the nested cubble, with `leafpop::addPopupGraphs()`:

```
# data pre-processing
clean <- weatherdata::climate_full |> ...

# use the long form to create subplots for each station
df_id <- unique(clean$id)
```

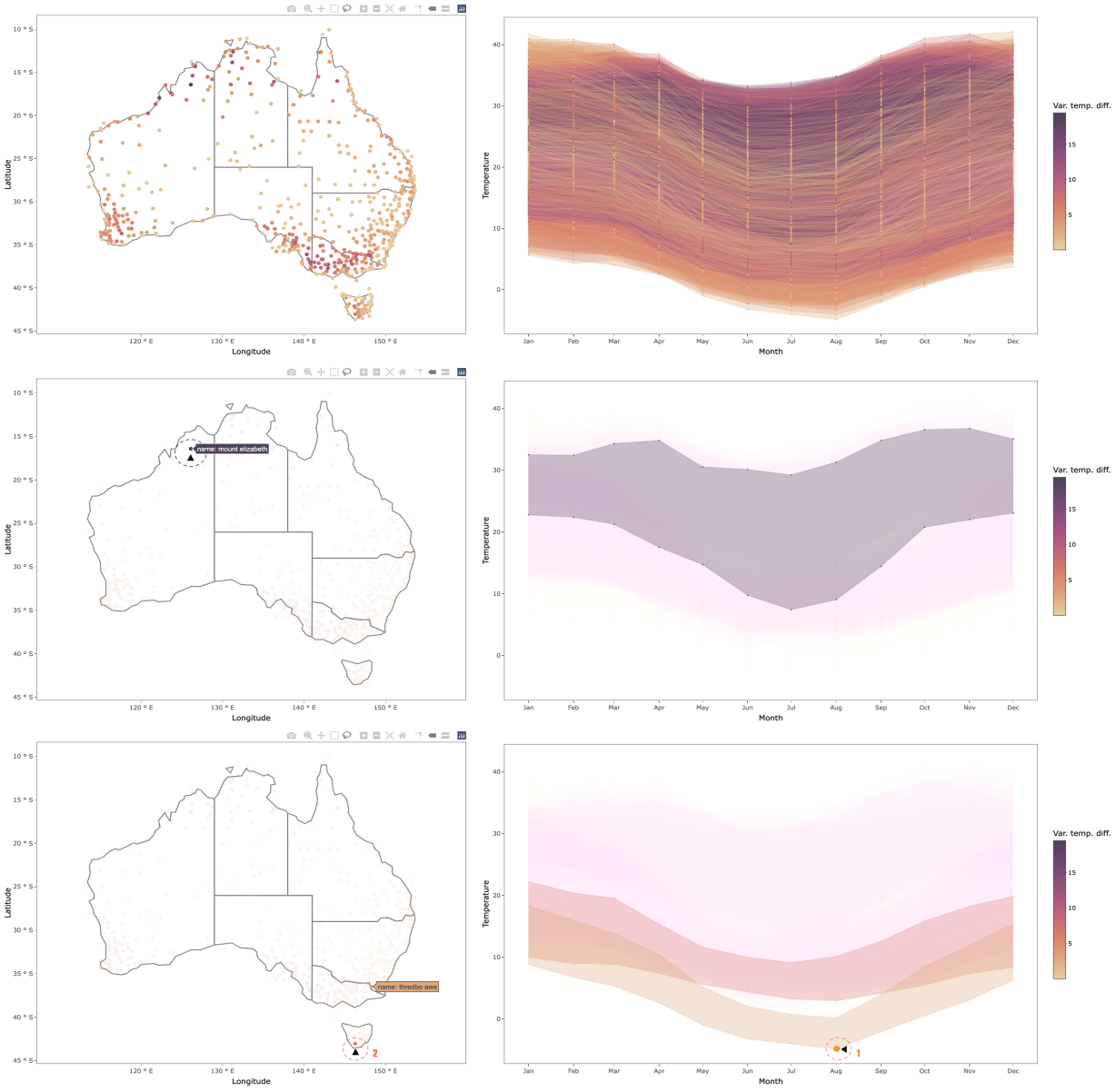

Figure 10: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a location with high variance on the map produces the plot in the second row. The maximum nad minimum temperature is shown using a ribbon. The bottom row first selects the lowest temperature in Auguest in the seasonal display. A location in the Tasmania Island is then selected to compare the temperature variation with Thredbo AWS.

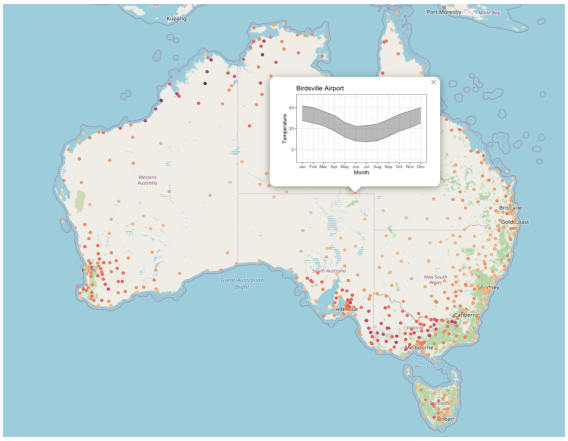

Figure 11: Same as Figure 11 with thetemperature variation shown as a popup in the leaflet map.

```
p <- map(1:length(df_id), function(i){
  dt <- clean |> filter(id == df_id[i])
  ggplot(dt) |> ...
})

# create nested form leaflet map with temperature band as subplots
nested <- face_spatial(clean)
leaflet(nested) |>
  addTiles() |>
  addCircleMarkers(group = "a", ...) |>
  leafpop::addPopupGraphs(graph = p, ...)
```

Figure 11 shows Figure 10 made made with leaflet and popups (Appelhans and Detsch 2021).

6 Conclusion

This paper describes an R package **cubble** for manipulating and visualising spatio-temporal data. A new data structure, `cubble` that builds from the `rowwise_df` and `grouped_df` class in the tidyverse ecosystem, is proposed to connect the time invariant and varying variables in the spatio-temporal data. This design frees the data analysts from spending time on organising variables of different observational units. The data structure is also flexible to the techniques and packages analysts use to analyse the data, for example, in the matching example in section 4.3, users are free to use algorithms from another package to cluster stations.

Further development and maintenance of the package involves responding to changes in the tidyverse packages that **cubble** imports, in particular, **tibble**, **tidyr**, and **dplyr**. In the spatial aspect, the simple feature representation wraps spatial coordinates in a list column, for some complex spatial operations, while sometimes, analysts may want to unpack them into longitude and latitude columns to work from. Another extension to **cubble** is to build a smooth transition between the coordinate columns and simple feature geometry column.

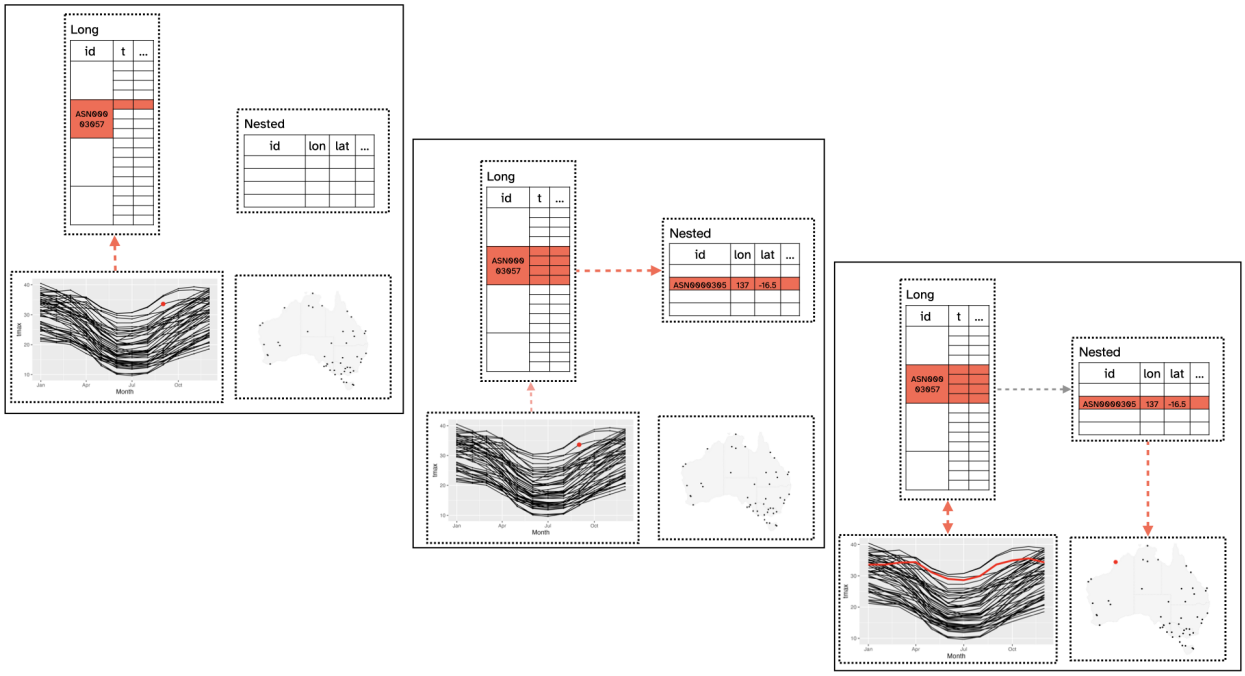

Figure 12: An illustration of the data model under interactive graphics with cubble. When a point on the time series is selected, the corresponding row in the long cubble will be activated. This will link to all the rows with the same id in the long cubble and the row in the nested cubble with the smae id (middle). Both plots will be updated with the full line selected and the point highlighted on the map (right).

7 Acknowledgement

This work is funded by the 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 **knitr** (Xie 2015) and **rmarkdown** (Xie, Allaire, and Grolemund 2018) in R. The source code for reproducing this paper can be found at: `https://github.com/huizezhang-sherry/paper-cubble`.

8 Appendix

Appelhans, Tim, and Florian Detsch. 2021. ***leafpop**: Include Tables, Images and Graphs in Leaflet Pop-Ups*. https://CRAN.R-project.org/package=leafpop.

Bach, Benjamin, Pierre Dragicevic, Dragicevic Archambault, Christophe Hurter, and Sheelagh Carpendale. 2014. "A Review of Temporal Data Visualizations Based on Space-Time Cube Operations." *Eurographics Conference on Visualization*, 19. https://hal.inria.fr/hal-01006140/.

Buja, Andreas, Daniel Asimov, and Catherine Hurley. 1988. "Elements of a Viewing Pipeline." *Dynamic Graphics Statistics*, 277.

Buja, Andreas, Dianne Cook, and Deborah F Swayne. 1996. "Interactive High-Dimensional Data Visualization." *Journal of Computational and Graphical Statistics* 5 (1): 78–99. https://doi.org/10.2307/1390754.

Cheng, Xiaoyue, Dianne Cook, and Heike Hofmann. 2016. "Enabling Interactivity on Displays of Multivariate Time Series and Longitudinal Data." *Journal of Computational and Graphical Statistics* 25 (4): 1057–76. https://doi.org/10.1080/10618600.2015.1105749.

Cocchi, Marina. 2019. *Data Fusion Methodology and Applications*. Elsevier.

Edzer Pebesma, Roger Bivand. 2022. "CRAN Task View: Handling and Analyzing Spatio-Temporal Data." https://CRAN.R-project.org/view=SpatioTemporal.

Hersbach, Hans, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, et al. 2020. "The Era5 Global Reanalysis." *Quarterly Journal of the Royal Meteorological Society* 146 (730): 1999–2049.

Hufkens, Koen, Reto Stauffer, and Elio Campitelli. 2019. "The **ecwmfr** Package: An Interface to ECMWF API Endpoints." https://doi.org/10.5281/zenodo.2647541.

Lu, Meng, Marius Appel, and Edzer Pebesma. 2018. "Multidimensional Arrays for Analysing Geoscientific Data." *ISPRS International Journal of Geo-Information* 7 (8): 313. https://doi.org/10.3390/ijgi7080313.

McIntosh, Avery I, Helen E Jenkins, Laura F White, Marinus Barnard, Dana R Thomson, Tania Dolby, John Simpson, 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): e1002638.

Michna, Pavel, and Milton Woods. 2013. "**RNetCDF**: A Package for Reading and Writing NetCDF Datasets." *The R Journal* 5 (2): 29–36.

———. 2021. ***RNetCDF**: Interface to 'NetCDF' Datasets*. https://CRAN.R-project.org/package=RNetCDF.

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

———. 2021. ***stars**: Spatiotemporal Arrays, Raster and Vector Data Cubes*. https://CRAN.R-project.org/package=stars.

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

Pebesma, Edzer, and Roger S Bivand. 2005. "S Classes and Methods for Spatial Data: The **sp** Package." *R News* 5 (2): 9–13.

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

Ryan, Jeffrey A., and Joshua M. Ulrich. 2020. ***xts**: eXtensible Time Series*. https://CRAN.R-project.org/package=xts.

Simmons, Adrian, Mariano Hortal, Graeme Kelly, Anthony McNally, Agathe Untch, and Sakari Uppala. 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): 668–89. https://doi.org/10.1175/JAS-3322.1.

Simmons, Adrian, Cornel Soci, Julien Nicolas, Bill Bell, P. Berrisford, Rossana Dragani, Johannes Flemming, et al. 2020. "Global Stratospheric Temperature Bias and Other Stratospheric Aspects of Era5 and Era5.1," no. 859 (January). https://doi.org/10.21957/rcxqfmg0.

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

Sumner, Michael. 2020. ***tidync**: A Tidy Approach to 'NetCDF' Data Exploration and Extraction*. https://CRAN.R-project.org/package=tidync.

Sutherland, Peter, Anthony Rossini, Thomas Lumley, Nicholas Lewin-Koh, Julie Dickerson, Zach Cox, and Dianne Cook. 2000. "**Orca**: A Visualization Toolkit for High-Dimensional Data." *Journal of Computational and Graphical Statistics* 9 (3): 509–29. https://www.tandfonline.com/doi/abs/10.1080/10618600.2000.10474896.

Wang, Earo, Dianne Cook, and Rob J Hyndman. 2020. "Calendar-Based Graphics for Visualizing People's Daily Schedules." *Journal of Computational and Graphical Statistics* 29 (3): 490–502.

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

Wickham, Hadley, Heike Hofmann, Charlotte Wickham, and Dianne Cook. 2012. "Glyph-Maps for Visually Exploring Temporal Patterns in Climate Data and Models." *Environmetrics* 23 (5): 382–93.

Xie, Yihui. 2015. *Dynamic Documents with R and **knitr***. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.name/knitr/.

Xie, Yihui, J. J. Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. Boca Raton, Florida: Chapman; Hall/CRC. https://bookdown.org/yihui/rmarkdown.

Xie, Yihui, Heike Hofmann, and Xiaoyue Cheng. 2014. "Reactive Programming for Interactive Graphics." *Statistical Science* 29 (2): 201–13. https://doi.org/10.1214/14-STS477.