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 27, 2022

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 analyse different aspects of the spatio-temporal data simultaneouly, like for instance instance, temporal trends of multiple variables across locations. In order to facilitate the study of different sections or combinations of spatio-temporal data, we introduce a new data structure, `cubble`, with a suite of functions enabling easy slicing and dicing on the different components spatio-temporal components, for visual and numerical explorations while facilitating data wrangling for modelling. The propose `cubble` structure ensures that all the components of the data are easy to access and manipulate while providing flexibility for data analysis. The `cubble` structure and the tools provided in the `cubble` R package equip users with the capability to handle hierarchical spatial and temporal structures. The tools are illustrated with examples of Australian climate data, merging climate and river level data sources, reproducing a climate reanalysis (ERA5), and creating interactive graphics.

Keywords spatial, temporal, spatio temporal, R, exploratory data analysis, environmental data, climate data

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 SpatioTemporal CRAN task view (Edzer Pebesma 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, E. Pebesma (2012) proposes four spatio-temporal layouts: full grid layout, sparse grid layout, irregular layout, and trajectory. These layouts are implemented in the **spacetime** (E. Pebesma 2012) package via the underlying spatial and temporal classes **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 data using spatio-temporal arrays. It interfaces with **sf** (E. J. Pebesma 2018), which adopts the simple features representation of spatial data, and **tidyverse** (Wickham et al. 2019), a suite of tools to wrangle and visualise data in R.

The paragraph above needs a bit more flesh

Still, the data representation for spatio-temporal data can be further improved and there are two reasons for this. Firstly, the raw data sourced in the wild is less often presented in any one of the representations above (Which representations - in the previous paragraph there are more than two?), 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) map data recording the shape of a collection of areas of interest; 2) geo-scientific data recording the longitude and latitude coordinates of locations in the area along with other metadata related to the location, and; 3) temporal variables indexed by both location and time.

I would aligned the examples of spatio-temporal data with those discuss in the st books. See Paula Moraga for example

The other reason is about tidy data concepts (Wickham 2014) and how they should be applied to spatio-temporal data. Tidy 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 given the downstream software i.e. **dplyr** (Wickham et al. 2022) and **ggplot2** (Wickham 2016) for data wrangling and visualisation. 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 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 analysing spatio-temporal data easier.

This paper presents a new R package, **cubble** which addresses the two issues mentioned above. 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 or combined while being kept synchronised. Among the four spacetime layouts in E. Pebesma (2012), **cubble** can be applied to full di layout, sparse grid layout, or irregular layout, but not trajectory, which is outside the scope of this work. The software is available from the Comprehensive R Archive Network (CRAN) at [CRAN link]. **At this point what is cubble is adding with respect the other packages it is not clear to me**

The rest of the paper is organized as follows: Section 2 introduces the proposed cube structure as a way to conceptualise multivariate 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 Australian weather station data and river level data as examples to demonstrate the use of **cubble**. An example of how **cubble** handles NetCDF data is also provided. Section 6 concludes the paper.

2 Conceptual framework: spatio-temporal cube

Ursula: I have rearranged this section somewhat, please check!

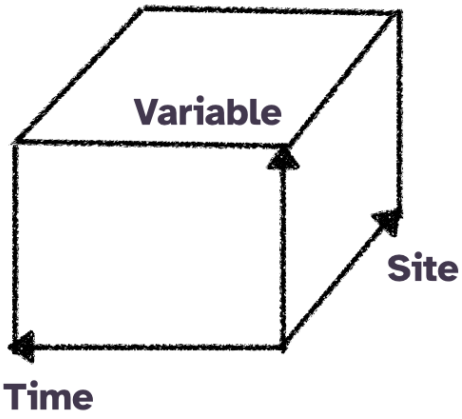

Figure 1: An illustration of the conceptualised spatio-temporal cube with three axes: time, site, and variable.

Spatio-temporal data can be conceptualised using a cubical data model with three axes, typically time, latitude and longitude. This abstraction can be useful for generalising operations and visualisation purposes: Lu, Appel, and Pebesma (2018) shows how array operations (select, scale, reduce, rearrange, and compute) can be mapped onto the cube; Bach et al. (2014) reviews 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 refers to "an abstract 2D substrate that is used to visualize data at a specific time." Despite its main focus being on temporal data, the mindset of abstracting out data representation to construct visualisation still applies to our spatio-temporal data manipulation and visualisation.

The most common data cube using the three axes (time, latitude, longitude) could be considered as taking snapshots of the space and stacking on time. Here we will define the spatio-temporal cube slightly differently with the three axes (time, site, variable), as illustrated in Figure 1. The time axis is the same in both versions, while the site axis now captures both latitude and longitude. Finally, variables are stacked on this space-time canvas, with one observation per site and time point. This notion is adopted to avoid using hypercubes when describing multivariate spatio-temporal data and is the workhorse behind **cubble**.

While a cubical data model is neat to view the spatio-temporal data conceptually, a 3D data representation, or an array, may not be convenient for data wrangling. There are two reasons for this: 1) While arrays are efficient for the computation on numerical values, spatio-temporal data contains more than just numerical variables. For example, wrangling on character strings and specific datetime classes is common for spatio-temporal data but may not be easily done with array objects. 2) The `mutate()` syntax for creating new variables is no longer valid since the operation can no longer define how a created 1D vector should be added to the array. Ursula: this seems too specific, you should be able to make the same argument without referencing the mutate function? How about the other functions from dplyr?, the same principle would apply right?

3 The cubble package

This section will first introduce the core functions in the cubble class: `as_cubble()`, `face_spatial()`, `face_temporal()`, and `unfold()` in subsection 3.1 - 3.4. The next subsection addresses the necessity of creating a new class. Subsection 3.6 will then show the compatibility of cubble with existing packages in spatial and temporal analysis.

Each core **cubble** function is introduced and accompanied with a short example using the data `climate_flat` throughout this section. `climate_flat` contains five weather stations in Australia with spatial information of each station: station id, latitude, longitude, elevation, station name, World Meteorology Organisation ID and also information on the record's date (recorded daily) together with the maximum and minimum temperature and precipitation for 2020. The first five rows of `climate_flat` are shown below:

```
## # 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

Spatio-temporal data can come in various formats and shapes and a cubble can be created from various types of raw data. This includes tibble and its variates in multiple tables (an example is 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 inputs: `key` as the spatial identifier; `index` as the temporal identifier; and a vector of `coords` in the order (longitude, latitude). The arguments `key` and `index` follow the wording in **tsibble** to describe the temporal order and multiple series while `coords` specifies the spatial location of each site. 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]`. The third row shows the name and type of all variables nested in the `ts` column and here it includes `date [date], prcp [dbl], tmax [dbl], tmin [dbl]`.

The resulting cubble is built from a `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 calculate on the list using the `mutate()` syntax, which is simpler than the `map()` when working with list column. For example, calculating the number of raining days can be done by:

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

```
## # 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
```

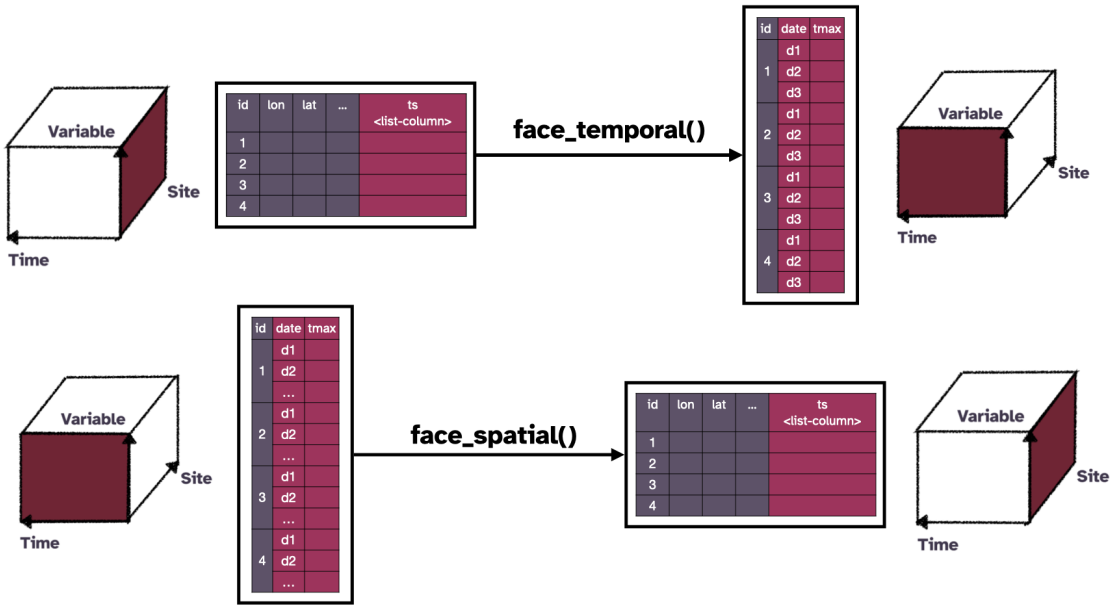

Figure 2: An illustration of function `face_temporal` and `face_spatial` in cubble. In the first row, `face_temporal` switches a cubble from the nested form into the long form and the focus has switched from the spatial aspect (the side face) to the temporal aspect (the front face). In the second row, `face_spatial` switches a cubble back to the nested form from the long form and shifts focus back to the spatial aspect.

3.2 Change focus by facing the time-variables

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`). **cubble** also provides a long form, which expands the `ts` column and temporarily "hides" the spatial variables. The function `face_temporal()` is used to switch a nested cubble into a long one and the first row in Figure 2 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 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
```

The first line in the header now shows the cubble is in its long form and the third line has been changed to show 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 class `groupped_df` where all the observations from the same sites form a group.

3.3 Change focus back to the site-variable face

Wrangling spatio-temporal data can be 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 2 illustrates the function and below is an example on the climate data:

```
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]>
```

```
identical(cubble_nested, cubble_back)
```

```
## [1] FALSE
```

3.4 Unfold spatial variables into the long cubble

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, which will be elaborated in section 4.4. 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:

```
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 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()`?

Ursula: I feel like this doesn't get the message accross, the main point seems to be that with cubble we can switch and never have to add information back in "by-hand" (with some join function), maybe make the point without refering to specific functions first and then explain the example?

Some readers may raise the question why a new data structure rather than using `dplyr::nest_by()` on the combined data to directly create a list-column. While `nest_by()` and its inverse `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()` can recreate the nested form, it cannot create a clean slate for temporal variables as used in the long form. For temporal variables, carrying the duplicated spatial variables in the combined form is not convenient to work with. This is because these spatial variables will be lost after `summarise()` and one will need to subset the spatial variables again from the original data and make a left join, which tends to create unnecessary intermediate products in the workflow.

```
# operation on the long form w/o cubble
# cubble workflow
dt |>
  as_cubble(...) |>
  face_temporal() |>
  group_by() |>
  summarise() |>
  unfold(...)

# tidyverse workflow
temporal <- dt |>
  group_by(...) |>
  summarise(...)

spatial <- dt |> select(....)

spatial |> left_join(temporal)
```

3.6 Compatibility with existing packages

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 are still valid within the 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). When creating a cubble an `sf` object can be supplied as the spatial table. Once included in a cubble, methods for the `sfc` class can be applied in the nested form. An example of this is shown in 5.1, which also handles the case when the site identifiers in the two tables do not match exactly. A spatial data object with `s2` vector can also be the input for the spatial table.

3.6.4 netcdf

NetCDF data is another format that is commonly used for storing spatio-temporal data. It 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 dimensions and variables 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 exist for manipulating NetCDF data and these include a high-level R interface: **ncdf4** (Pierce 2019), a low-level interface that calls a 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 × the number of latitude grid points × the number of time grid points × the number of variables. Cubble can handle slightly more than 300 × 300 (longitude × longitude) 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. The spatial grid can also be subsetted to trade for longer time periods and more variables through `long_range` and `lat_range` if the NetCDF file has finer resolution than needed:

```
# Assume my_ncdf has a 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 structures either inherent in the data or obtained 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 3 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 redefines the cubble key from the `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 problem can be seen as a matching problem (Stuart 2010; McIntosh et al. 2018) that pairs up 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 by 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 4 illustrates this temporal matching in more details. Given two series `A` and `a`, three peaks have been picked in each series. An interval, with default length of five, is constructed for each peak in series `A` and the peaks in series `a` are tested against whether they fall into any of the intervals. In this illustration, there are two 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 peaks 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 peaks for a valid matched pair

4.3 Interactive graphics

The linking structure of two forms in cubble fits 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, Cook, and Hofmann 2016). Diagram 5 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

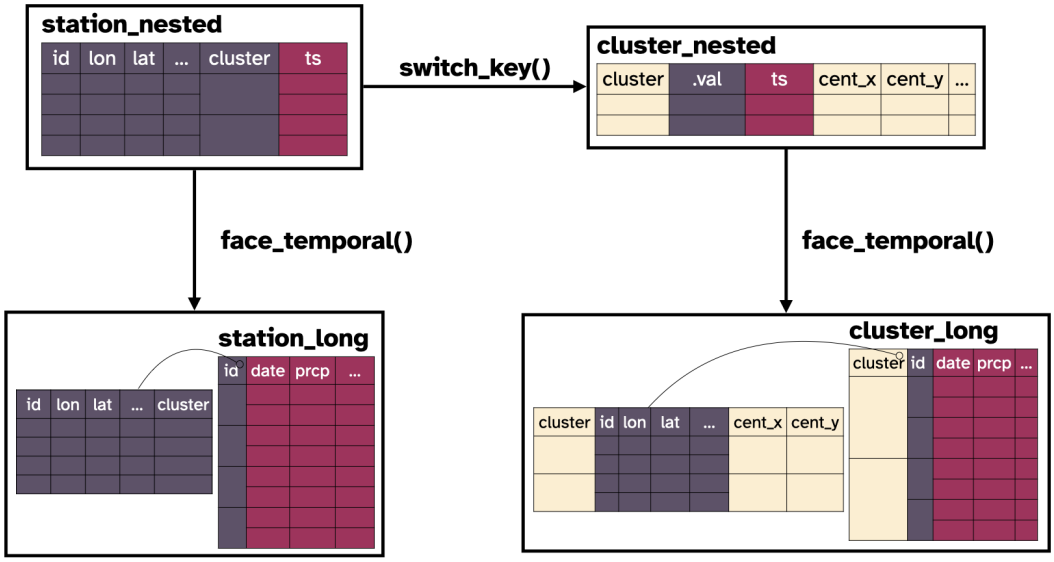

Figure 3: An illustration of the original and cluster level cubble in the nested form long form for hierarchical structure data. `switch_key()` changes the station level cubble into a cluster level cubble and both can be stretched into the long form.

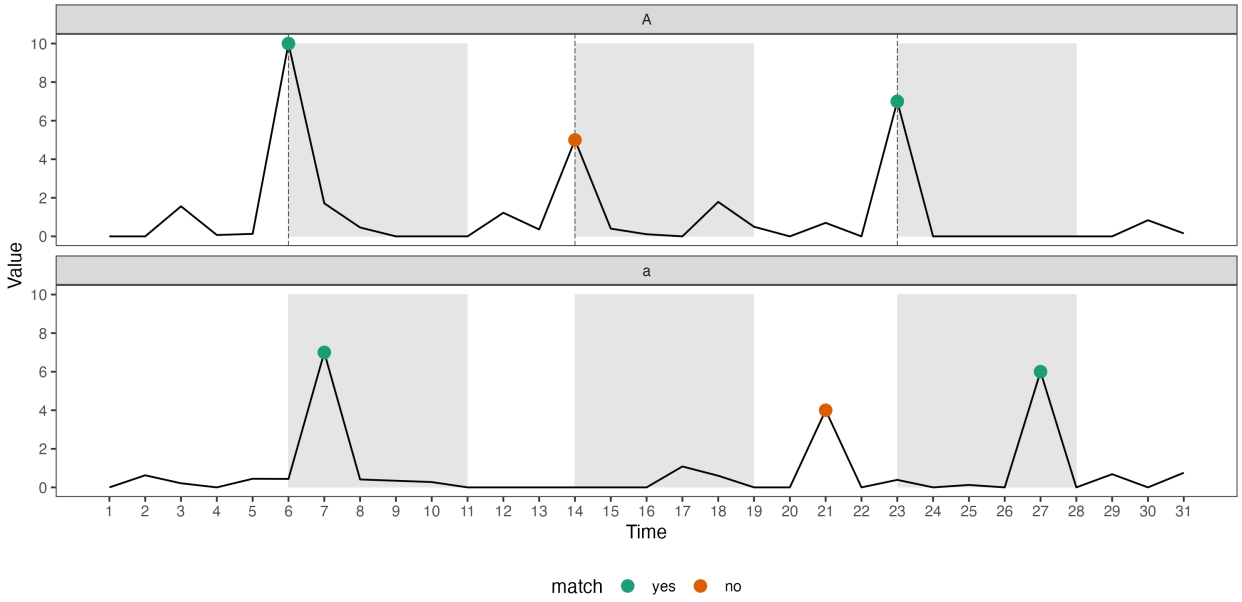

Figure 4: An illustration of temporal matching in cubble. Three highest peaks are identified in each series and intervals are constructed on series `A`. Two peaks in series `a` fall into the intervals and hence the two series are considered to have two matches.

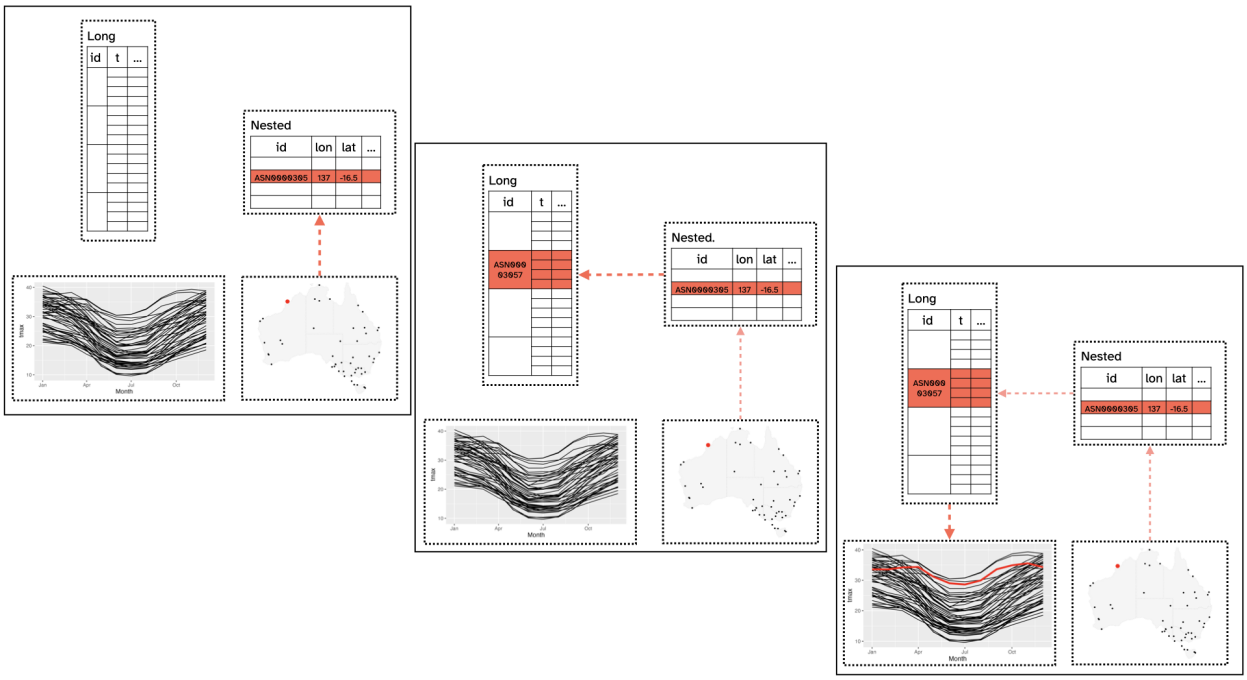

Figure 5: An illustration of the data model under interactive graphics with cubble. The line plot and the map is made separately with the long and nested cubble. When a station is selected on the map (left), the corresponding row in the nested cubble will be activated. This will link to all the rows with the same id in the long cubble (middle) and update the line plot (right).

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 be reflected in the updated plots and Diagram 13 in the Appendix illustrates this process.

4.4 Spatio-temporal transformations

Different types of transformations can be useful to extract new information from spatio-temporal data. Glyph maps (Wickham et al. 2012) transform the time coordinates into space coordinates to plot the time series of different locations on the map. Calendar plots (Wang, Cook, and Hyndman 2020) reconstruct time into a 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 reimplementation 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 are also available in the `geom_glyph()` implementation. Polar glyph map can be specified 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

The Victoria State Government in Australia provides daily COVID case numbers by date, source, and local government area (LGA). This data can be used to visualise COVID spread when combined with map information on LGA, available from the Australian Bureau of Statistics. The first five rows of both data are printed below:

```
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...
```

Cubble can be created with separate spatial and temporal tables in a list, with other arguments (`key`, `index`, and `coords`) introduced in section 3.1. `as_cubble()` will automatically check the matching of sites in both tables and emit a warning 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 attempt to pair the unmatched sites as well as those that cannot be paired. These can be helpful to clean up the data before creating the cubble again:

```
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)

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

5.2 Australian historical maximum temperature

The 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 the GHCN 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: the Australian GHCN station numbers start with "ASN00" and are followed by the Bureau of Meteorology (BOM) station number. The second and third digits of the BOM station number (7th and 8th in the GHCN number) define the state of the station; the range 46-75 corresponds to New South Wales stations and the range 76-90 corresponds to Victorian stations. Filtering Victoria and New South Wales stations is a spatial operation and hence uses the nested form:

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

Filtering for the time periods 1971-1975 and 2016-2020 is a time operation and the nested cubble needs to be switched to the long cubble form 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, which is another time operation:

```
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 do not have records during the time period 1971-1975 and further investigation shows that while the first and last year of each series is recorded, the missing years in this period are not reported. These stations are filtered out by examining whether the summarised time series has 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 `unfold()`:

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

`tmax` can then be supplied to `geom_glyph()` for the glyph map in Figure 6 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), which is a problem when mapping more stations at the national level. Aggregation can be helpful in grouping series into clusters before visualising the clusters with a glyph map. This example shows how to organise data at both levels with `switch_key()`.

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

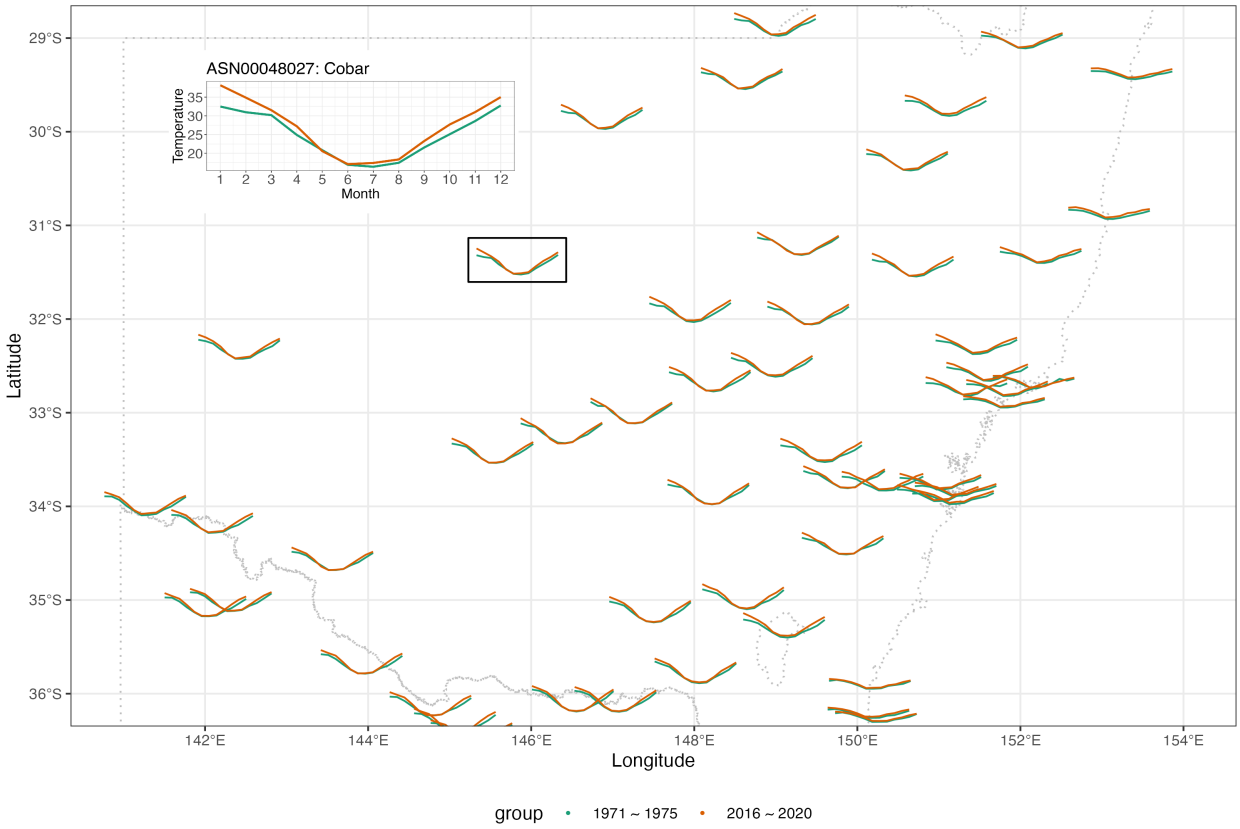

Figure 6: A glyph map of the average maximum temperature by month of Victoria and New South Wales weather stations in Australia. On the top left corner is an insetted plot of station Cobar highlighted in the black box.

Figure 7: 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 is printed in the middle of the y minor axis and can be used as a reference line to read the magnitude. Subplot B shows the station membership of each 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 7 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

The Bureau of Meteorology collects water data from river gauges. The collected variables include: electrical conductivity, turbidity, watercourse discharge, watercourse level, and water temperature. In particular, water level will interact with precipitation from the climate data since rainfall will raise the water level in the river. Figure 8 gives the location of available weather stations and water gauges in Victoria.

From the map, one can see that a few water gauges and weather stations are close to each other and the fluctuation of the water level could be matched up with the precipitation measured by the nearby climate station. As introduced in Section 4.2, `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 the water level in the river will reflect precipitation. This puts precipitation `prcp`, as the independent variable. Given there is one year's worth of data, the number of peaks

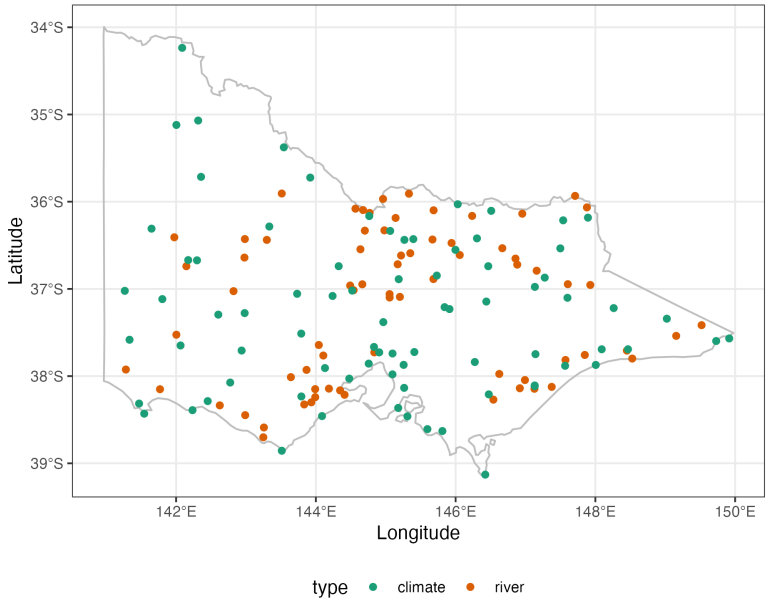

Figure 8: Location of weather stations and river gauges in Victoria, Australia.

(`temporal_n_highest`) to consider is slightly raised from a default of 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 9 plots the matched pairs on the map. There are four pairs of matches, all located in the middle of Victoria and the expected concurrent increase in precipitation and water level can be observed.

5.5 ERA5: climate reanalysis data

The ERA5 dataset (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 the European Centre for Medium-Range

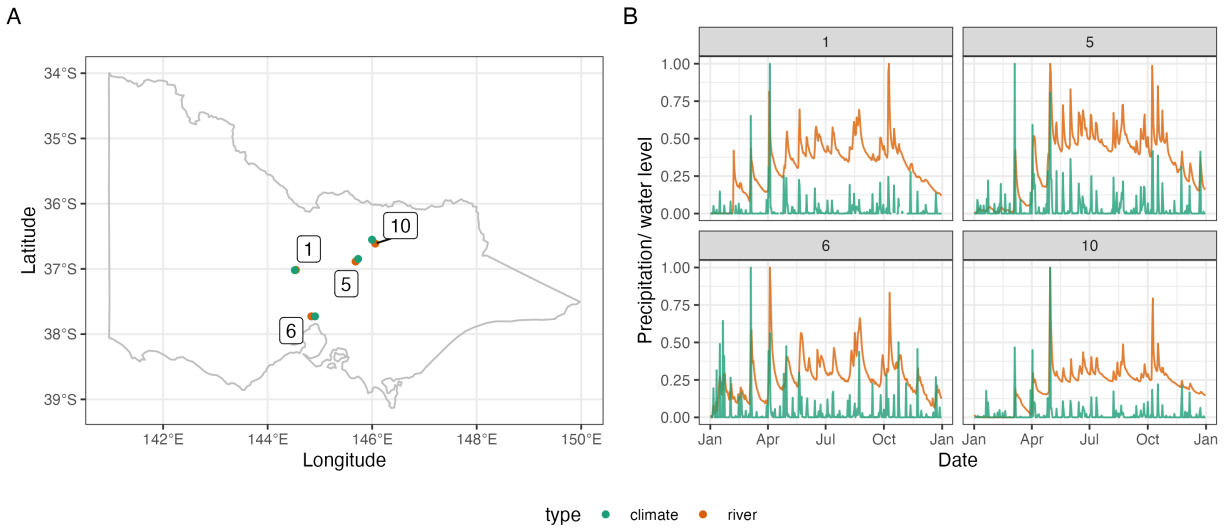

Figure 9: Matched weather stations and river gauges on the map (A) and across time (B). Precipitation and water level have been standardised between 0 and 1 to be displayed on the same scale. The water level reflects the increase in precipitation.

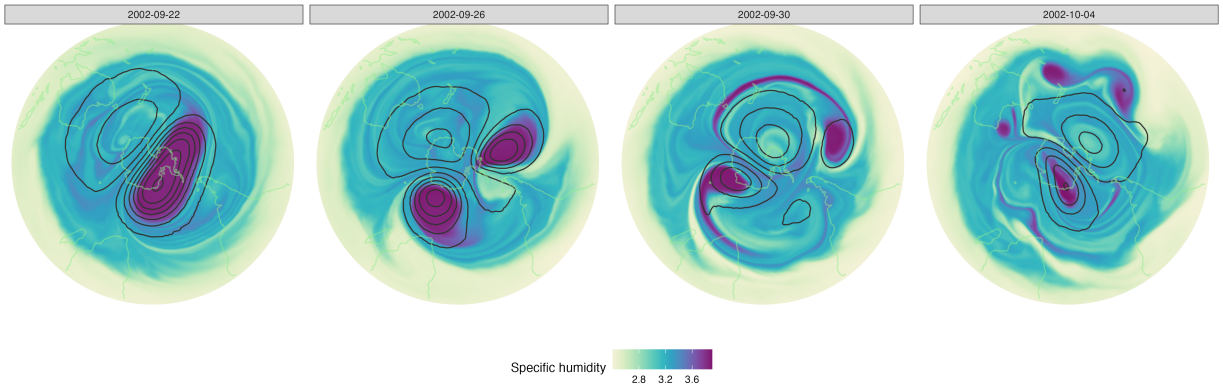

Figure 10: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020).

Weather Forecasts (ECMWF). The data can be directly downloaded from Copernicus Climate Data Store (CDS) website or programmatically via the R package **ecmwfr** (Hufkens, Stauffer, and Campitelli 2019). The `era5-pressure` data contains the *specific humidity* and *geopotential* variables 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 10 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 Interactive 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 these two types of plots with interactivity lets users link between points on the map and the corresponding time series to explore the spatial and temporal dimensions 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 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` object separately. In this example, stations on the Australia map, made from the nested form, are coloured by the calculated variance. A ribbon band is constructed using the long form cubble to show each station's maximum and minimum temperature across month. With a different dataset, users can calculate other per station measure in the nested form or make other 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 11, the first row shows the initial view of the interactive graphic. Most regions in Australia have low variance of temperature range, while the north-west coastline, bottom of South Australia, and Victoria stand 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 selects the lowest temperature in August and this corresponds to Thredbo AWS on 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 is displayed as a small subplot upon clicking on the map. The procedure involves first creating the popup plots from the long form cubble as a vector and then adding 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)
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() |>
```

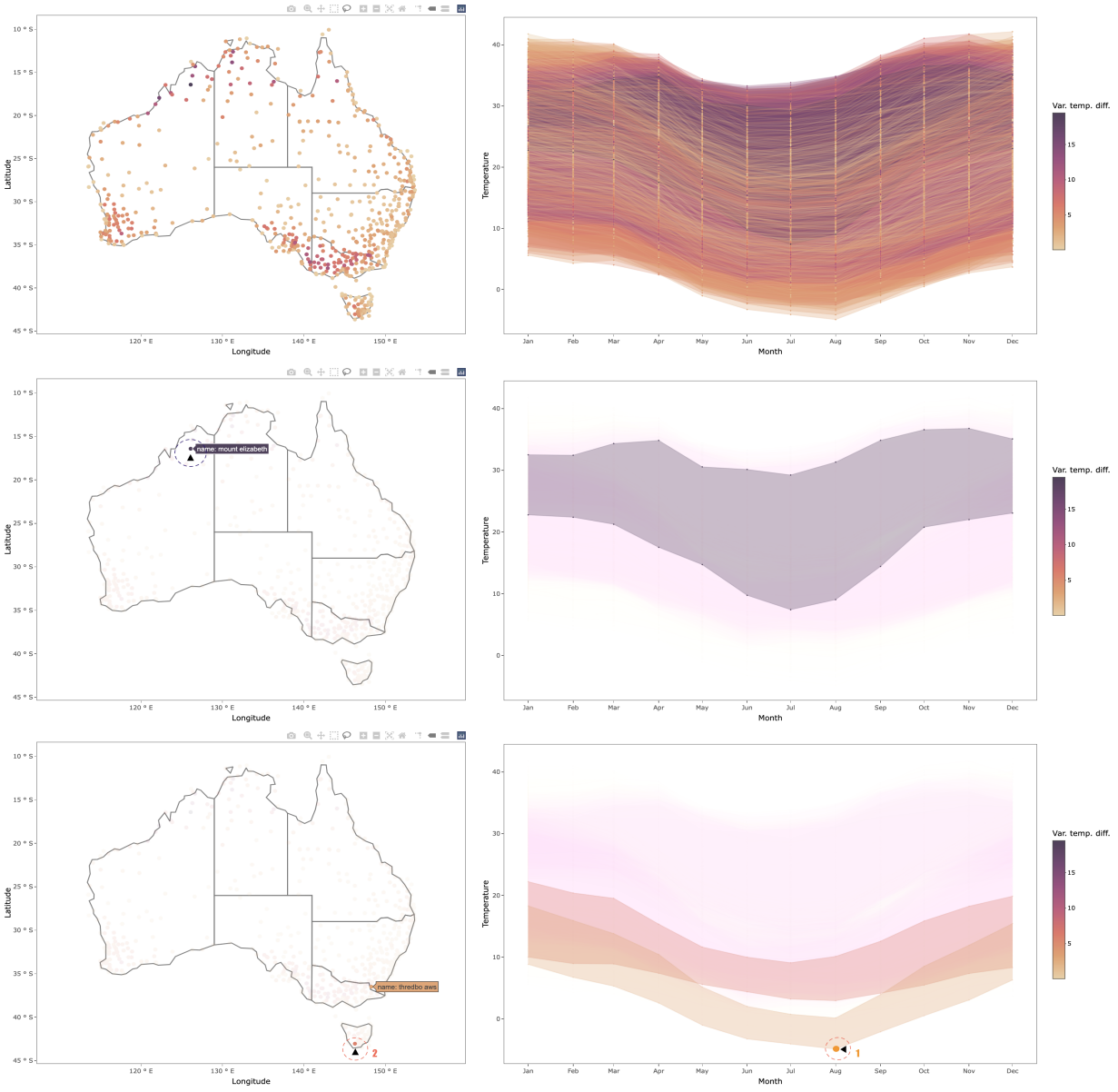

Figure 11: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a location with high variance on the map produces the plot in the second row. The maximum and minimum temperature is shown using a ribbon. The bottom row first selects the lowest temperature in August in the seasonal display. A location in the Tasmania Island is then selected to compare the temperature variation with Thredbo AWS.

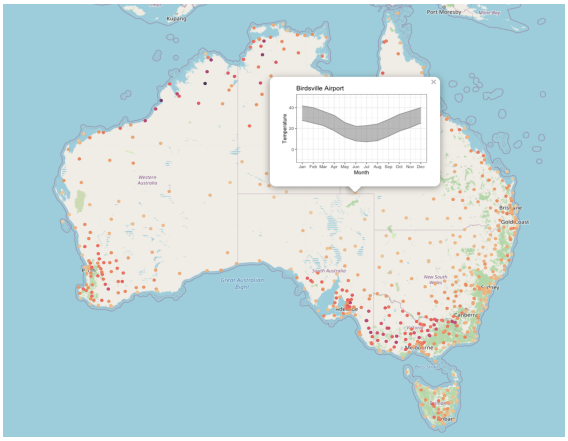

Figure 12: Same as Figure 11 with the temperature variation shown as a popup in the leaflet map.

```
addCircleMarkers(group = "a", ...) |>
leafpop::addPopupGraphs(graph = p, ...)
```

Figure 12 shows Figure 11 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.

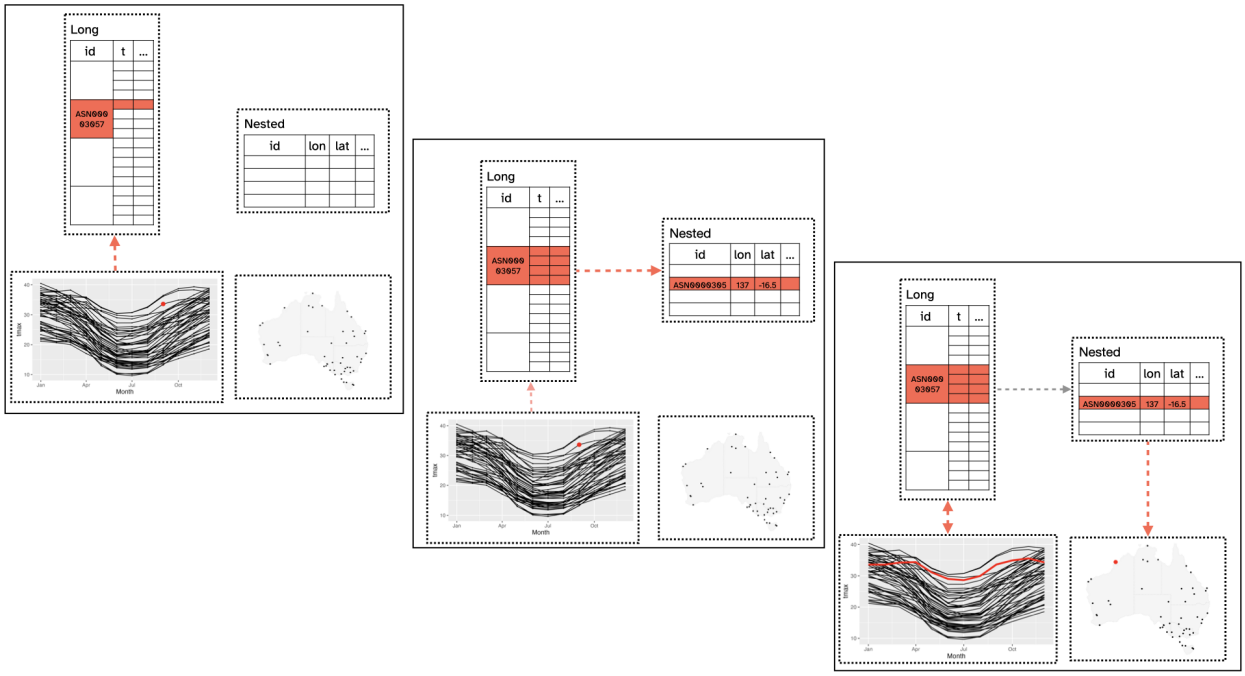

Figure 13: An illustration of the data model under interactive graphics with cubble. When a point on the time series is selected, the corresponding row in the long cubble will be activated. This will link to all the rows with the same id in the long cubble and the row in the nested cubble with the same 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 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 **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.

———. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

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.