



Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II.

doi: 10.18637/jss.v000.i00

cubble: An R Package for Structuring Spatio-temporal Data

H. Sherry Zhang
Monash University

Dianne Cook
Monash University

Ursula Laa
University of Natural Resources and Life Sciences

Nicolas Langrené
BNU-HKBU United International College

Patricia Menéndez
Monash University

Abstract

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 being flexible to explore the two aspects 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.

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

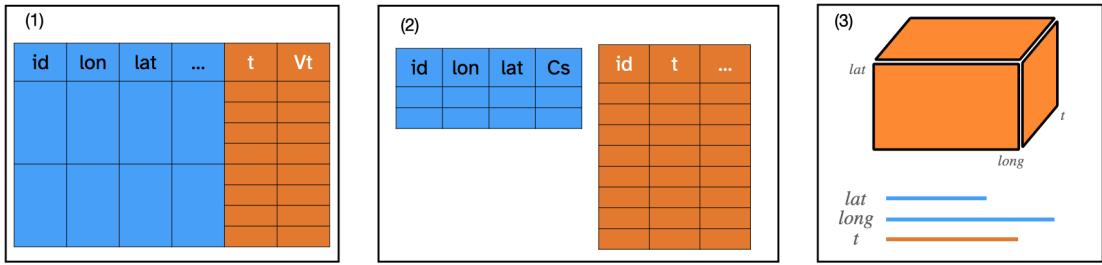


Figure 1: An illustration of different spatio-temporal data layouts: (1) a single table for both spatial and temporal variables; (2) separate tables for spatial and temporal variables; and (3) an array data structure that populates variables into a gridded longitude-latitude-time (hyper)cube.

1. Introduction

Spatio-temporal data involves variables observed in the space across time and this paper focuses on spatio-temporal vector data with fixed locations and regular time intervals. This type of data can naturally be considered as a data cube with three axes: spatial identifier, temporal identifier, and variables. In this cubic formation, space only occupies one axis, rather than two, to avoid hypercubes for multivariate spatio-temporal data. While conceptually the data format is relatively simple, the data objects coming to the analysts can vary in forms in the analysis. Figure 1 shows three examples of wild spatio-temporal data: 1) a single table that includes both the spatial and temporal variables; 2) separate tables for spatial and temporal variables; 3) array-based data, i.e. NetCDF (Network Common Data Form) data, which is commonly used in earth science, e.g. climatology, meteorology, and oceanography, among others, to organise multivariate data in a spatio-temporal grid.

While incorporating both spatial and temporal information enriches the data in the analysis, spatio-temporal data can sometimes be hard to work with given its two observational units in the variables. The single table format in Figure 1 is convenient for spatio-temporal operations that involve both types of variables, however, it makes duplicates of spatial variables and is not ideal when only looking at the spatial aspect of the data. The two tables format allows separate operations on spatial and temporal variables while analysts need to work with two data objects simultaneously and join them together into the single table format for spatio-temporal operations. Neither of the two formats is desirable at the same time for spatial, temporal, and spatio-temporal operations. This issue implies analysts need to constantly think about the data format and arrange variables into the proper shape before the actual operations, which can be a laborious step when switching between the two formats back and forth several times during the analysis.

In the R community, many data objects have been proposed to work with specific spatio-temporal tasks, while few can freely explore the data. **spacetime** (Pebesma 2012) is an attempt for such a data object, but the spatial and temporal classes it is built on have now been gradually replaced by more recent implementations. The **stars** (Pebesma 2021) package uses an array to wrangle spatio-temporal data, but the dense array it uses can quickly exhaust the memory for data in the case of irregular spatial grid. While there has not yet been a data object that makes it easy to wrangle spatio-temporal data, there is a strong need

for such tools given the ubiquitous spatial and temporal data collected in the society, i.e., climate observations from weather stations, air quality variables from air monitoring stations. The demand for such tools can also come from spatial data analysts wishing to incorporate temporal data in their analysis or vice versa.

This paper presents a new R package, **cubble**, to overcome the problem with organising variables in different levels when working with spatio-temporal data. 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 presents the main design and functionality of **cubble**. Section 3 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 4 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 5 concludes the paper.

2. Conceptual framework: spatio-temporal cube

Multivariate 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.

The cubical conceptual framework provides a generalisation to the operation can be done on the data for visualisation. A paper by [Bach, Dragicevic, Archambault, Hurter, and Carpendale \(2014\)](#) 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. Existing work and new challenges

what has been done and the new challenges

The **spacetime** paper ([Pebesma 2012](#)) has set the foundation for wrangling spatio-temporal data in R. It proposes four layouts (STF: full grid layout, STS: sparse grid layout, STI: irregular layout, and STT: trajectory) to represent spatio-temporal data. The implementation uses class **sp** to represent spatial data and class **xts** for temporal data. In recent years, new data structures have appeared to replace and improve on these two classes to represent spatial and temporal data. The package **sf** has adopted the simple feature access ([J.R. Herring 2011](#))

to represent spatial information using POINT, MULTIPOINT, LINESTRING, POLYGON, MULTILINESTRING, MULTIPOLYGON, and GEOMETRYCOLLECTION. For temporal data, the package **tsibble** adopts the long form to explicitly record the time stamp as a column rather than an attribute of the data. These changes in the spatial and temporal community have yet been reflected in the spatio-temporal data analysis. The [Pebesma \(2012\)](#) paper also summarises three formats spatio-temporal data are usually recorded: time-wide, space-wide, and long format. With the concept of tidy data ([Wickham 2014](#)), the long form has been increasingly considered as the tidy format. The main advantages of it over the wide format is that the latter treat either time or location value as variable names (Section 3.1 in [Wickham \(2014\)](#)), making them difficult to be wrangled. While temporal data can be benefited from a long format, it brings repetition of spatial variables in the combined spatio-temporal data and this can be a huge inefficiency for large MULTIPOLYGON over long time period. This suggests the need to look for other format for spatio-temporal data.

two challenges: multiple data sources + large spatio-temporal object In addition to this, there are more challenges in manipulation spatio-temporal data. Different variables are recorded and collected in multiple tables, for example, an areal data would map data for a collection of areas, geo-scientific data would record locations and other meta data, and the temporal variable indexed by both location and time are stored in the long form table. To work with these data, analysts need to link the geo-scientific data with the temporal variables and work out how they are contained in the area. This would require merging all three sources together and can often go wrong because of the slightest unmatched of linking variable in different sources. Another challenge analysts face when wrangling spatio-temporal data is with the simple feature access representation of spatial data. While they have become the standard for recording large and complex spatial geometry and precise operation on the spatial coordinates, with data analysis, the detail is more than enough. A data analysts don't necessarily need to have the every nitty gritty detail of the location and a map with sufficient information to outline the shape of the location would be enough for the propose of visualisation. With a large and detailed geometrical object, the map would take longer to render and analysts always need to first simplify it into a more manageable size. Also, while the simple feature access structure the coordinates into a list column, analysts would sometimes like to unpack it into two columns of longitude and latitude to work with. They should be provided with the option to do so.

A newer implementation The **stars** has been an implementation to represent spatio-temporal data with array. While it is efficient to represent spatio-temporal data in an array, there are situation where array [is not the best solution]. mutate in the array is not clearly define (how to add a column vector). Also, it doesn't solve the problem with multiple data sources as it may take a large amount of effort to shape multivariate spatio-temporal data into high-dimensional cube before the analysis starts and they should be provided with option to work with 2D tables.

4. The cubble package

This section will present two formats that cubble uses to arrange spatio-temporal data. The main cubble functions will then be introduced to illustrate how to work with these two formats in short examples. Lastly, a subsection will be dedicated to how existing packages, in spatial and temporal analysis, fit in with cubble.

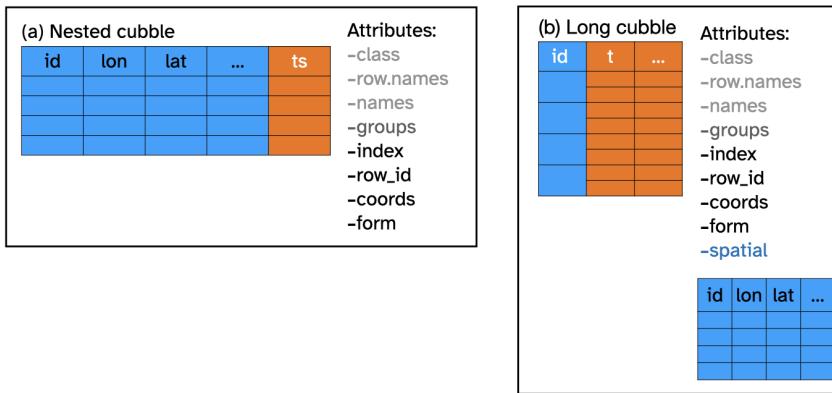


Figure 2: An illustration of the nested and long form in **cubble**. The nested form defines each site in a row and nests the time varying variables into a single column **ts**. The long form cubble uses **id** and **t** to identify each row and stores the time invariant variables as an attribute, **spatial**.

In **cubble**, a single data object can be arranged into two forms: nested form and long form. Figure 2 sketches the two forms with the associated attributes. The decision on which form to use in data wrangling is output-oriented, meaning analysts need to first think about whether the output of an operation is identified only by the spatial identifier, or a combination of spatial and temporal identifier. The nested cubble is suitable for working with operations that are only identified by the site and this type of operation can be a pure manipulation of time invariant variables, or an operation that summarises time varying variables into site. Underneath the nested form, a cubble is built from a **rowwise_df** class where each site forms a separate group. This structure simplifies the calculation that involves temporal variables by avoiding the use of **purrr::map()** syntax when working with list-column.

For those operations whose output involves both a spatial and temporal identifier, long form should be used. The long form uses a **grouped_df** class to forms all the time of a site as a group. Time invariant variables are stored separately as an special attribute of the long cubble. This design avoids repeating the spatial variables at each time stamp while not dropping information from spatial variables.

4.1. Create a cubble in the nested form

To use functions in the **cubble** package, an analyst will first need to turn the data object into a **cubble** class. **as_cubble()** does this through supplying the three key components: **key** as the spatial identifier; **index** as the temporal identifier; and a vector of **coords** in the order of longitude and latitude. The use of **key** and **index** follows the design in the **tsibble** package and the cubble created by default is in the nested form.

Before the formal examples in Section 6, each function introduced in this section will be accompanied by a small example. The data used for the small examples is a subset of a larger climate weather station data used in the formal examples and has been simplified to include only five weather stations. It contains 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. `climate_flat` stores this data in format 1) in Figure 1 and 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:

```
R> cubble_nested <- cubble::climate_flat %>%
+   as_cubble(key = id, index = date, coords = c(long, lat))
R> 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]>
```

There are a few information in the cubble header: the name of the key variable: `id` and its unique number: 5, the bounding box: `[115.97, -32.94, 133.55, -12.42]`, and also the name of variable nested in the `ts` column with its type: `date [date], prcp [dbl], tmax [dbl], tmin [dbl]`.

4.2. Stretch a 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 `stretch()` is designed to switch the cubble from the nested form to the long form. This code shows how to switch the nested cubble just created into its long form:

```
R> cubble_long <- cubble_nested %>% stretch()
R> 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:

```
R> 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
```

4.3. Tamp 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. `tamp()`, an inverse of `stretch()`, switches a long cubble to a nested cubble:

```
R> cubble_back <- cubble_long %>% tamp()
R> 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]>
```

4.4. Migrate spatial variables to a 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 5.4). Cubble allows this operation through `migrate()`, which moves the specified spatial variables into the long cubble:

```
R> cubble_mig <- cubble_long %>% migrate(long, lat)
R> cubble_mig
```

```
# 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>
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:

```
R> cubble_mig %>% temp() %>% stretch()

# 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
```

4.5. 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`.

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`)

tsibble

`tsibble` is a temporal data structure that uses `index` and `key` to identify the time and different series. `cubicle` can be seen as a natural extension of `tsibble` for spatio-temporal data with an additional coordinates component and using two forms to arrange variables. This makes it easy to cast a `tsibble` into a `cubicle` as only the `coords` argument needs to be supplied:

```
R> # example with a tsibble created from climate_flat
R> raw <- climate_flat %>%
+   tsibble::as_tsibble(key = id, index = date)
R>
R> dt <- raw %>%
+   cubicle::as_cubicle(coords = c(long, lat))
R> dt

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

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

# cubicle: 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
```

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.

```
R> library(sf)
R> # create a cubble
R> cb <- climate_flat %>%
+   cubble::as_cubble(key = id, index = date, coords = c(long, lat))
R>
R> aus <- ozmaps::abs_st
R>
R> 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")
+   )
R>
R> 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:

```
R> library(s2)
R> # Western Australia map
R> wa <- ozmaps::abs_st %>% filter(NAME == "Western Australia")
R>
R> # mutate a `s2_lnglat` vector on `cb` created in the last chunk
R> 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~
```

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:

```
R> # read in the .nc file as a ncdf4 class
R> raw <- ncdf4::nc_open(here::here("data/era5-pressure.nc"))
R>
R> # convert the variable q and z in the ncdf4 into a cubble
R> 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.

```
R> # Assume my_ncdf has bounding box of [-180, -90, 180, -90]
R> # at 0.25 degree resolution and subset it to have
```

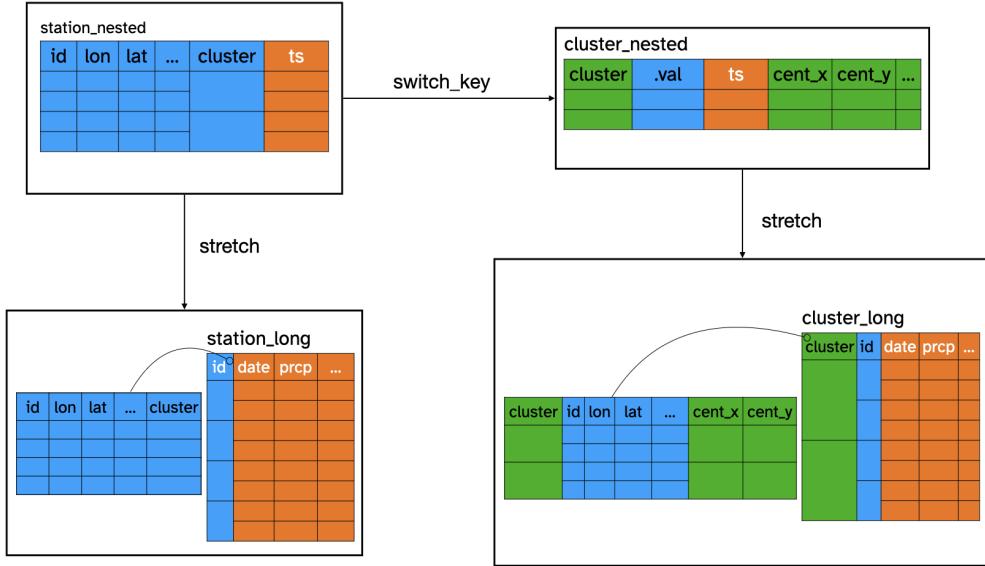


Figure 3: An illustration of the original and cluster level cubble in the nested form long form for hierarchical structure data. **switch_key()** changes the station level cubble into a cluster level cubble and both can be stretched into the long form.

```
R> # 1 degree resolution:
R> dt <- as_cubble(my_ncdf, vars = c("q", "z"),
+                     long_range = seq(-180, 180, 1),
+                     lat_range = seq(-90, 90, 1))
```

5. Other features and considerations

5.1. 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 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 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**.

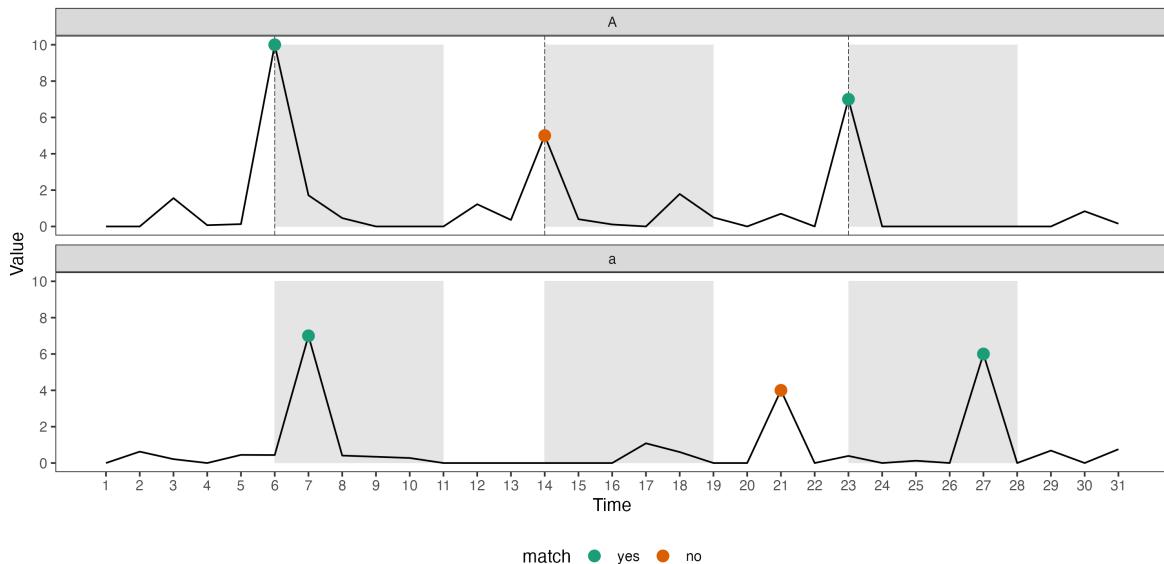


Figure 4: An illustration of temporal matching in **cubble**. Three highest peaks are identified in each series and intervals are constructed on series A. Two peaks in series a fall into the intervals and hence the two series are considered to have 2 matches.

5.2. 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, Jenkins, White, Barnard, Thomson, Dolby, Simpson, Streicher, Kleinman, Ragan *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 4 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

5.3. Interactive graphics

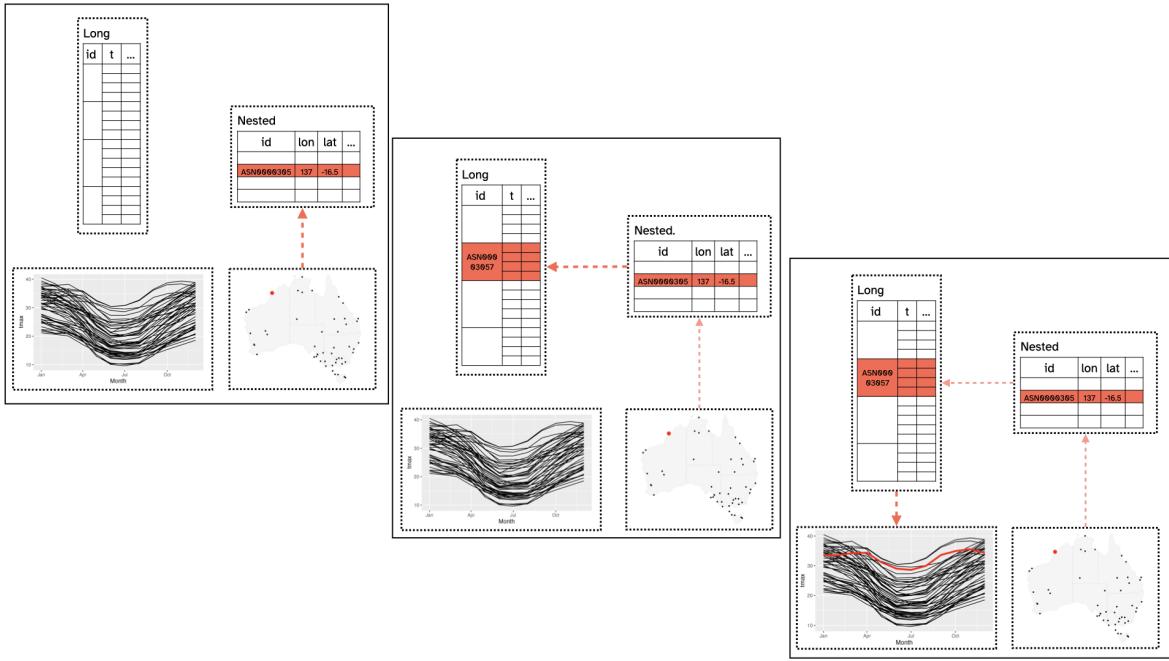


Figure 5: An illustration of the data model under interactive graphics with cubble. The line plot and the map is made separately with the long and nested cubble. When a station is selected on the map (left), the corresponding row in the nested cubble will be activated. This will link to all the rows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

Cubble fits in naturally with the interactive graphic pipeline discussed in the literature (Buja, Asimov, and Hurley 1988; Buja, Cook, and Swayne 1996; Sutherland, Rossini, Lumley, Lewin-Koh, Dickerson, Cox, and Cook 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 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 13 in the Appendix illustrates this process.

5.4. Spatio-temporal transformations

Spatio-temporal transformation is a useful technique to extract information form spatio-temporal data. Glyph maps (Wickham, Hofmann, Wickham, and Cook 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:

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

A 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()`:

```
R> 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()`.

6. Examples

6.1. 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:

```
R> 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()`:

```
R> tmax <- tmax %>%
+   stretch() %>%
+   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:

```
R> 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 `tamp()`:

```
R> tmax <- tmax %>% tamp() %>% 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()`:

```
R> tmax <- tmax %>% stretch() %>% migrate(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:

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

```
+   geom_glyph(data = tmax,
+               aes(x_major = longitude, x_minor = month,
+                    y_major = latitude, y_minor = tmax,
+                    group = interaction(id, group), color = group),
+               width = 1, height = 0.5) +
+   ...
```

6.2. 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()`.

`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.

```
R> station_nested <- weatherdata::climate_full %>%
+   mutate(cluster = ...)
```

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

```
R> 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:

```
R> 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.

6.3. 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 8 gives the location of available weather station and water gauges in Victoria.

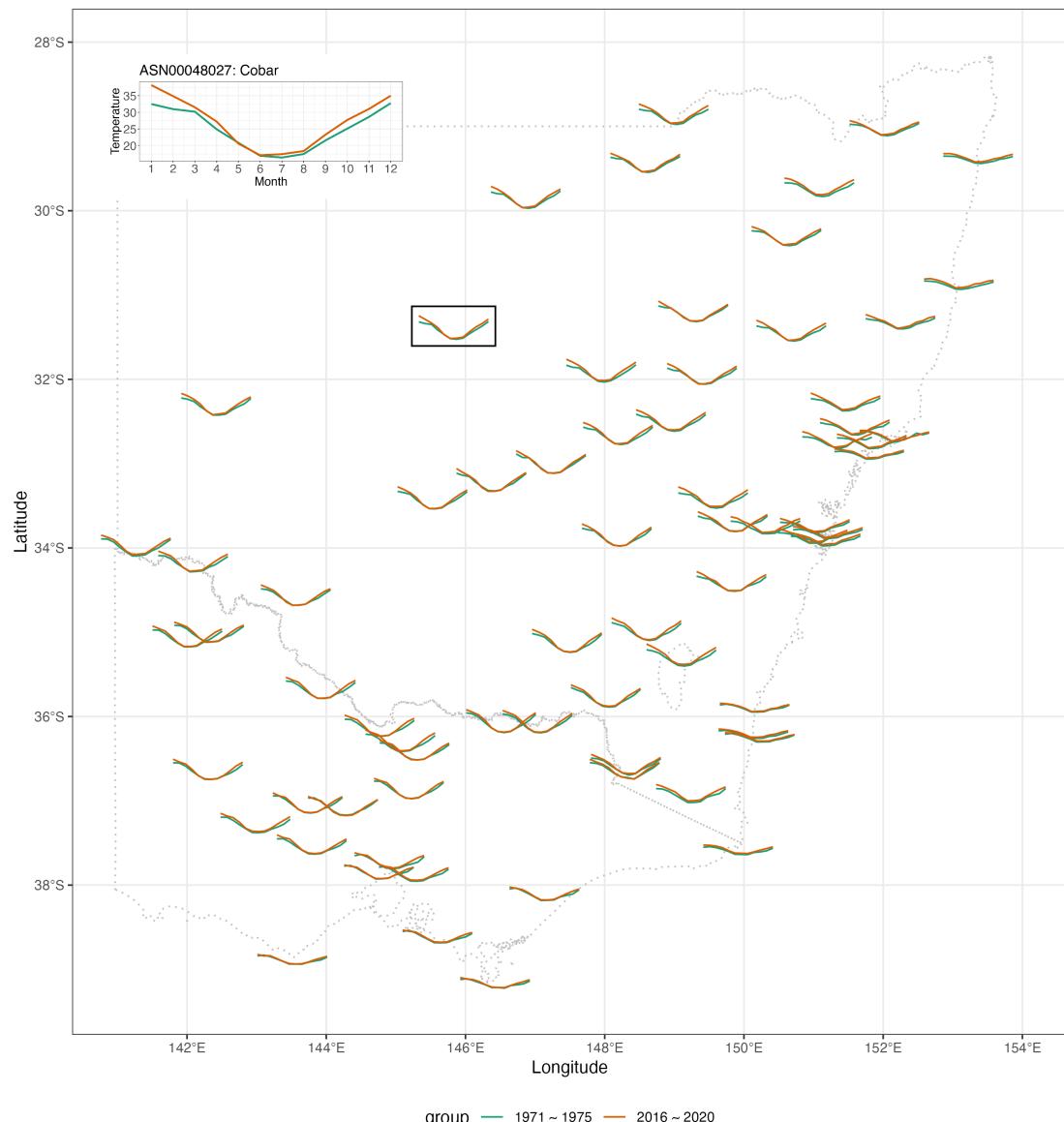


Figure 6: A glyph map of the average maximum temperature by month of Victoria and New South Wales weather stations in Australia. On the top left corner is an insetted plot of station Cobar highlighted in the black box.

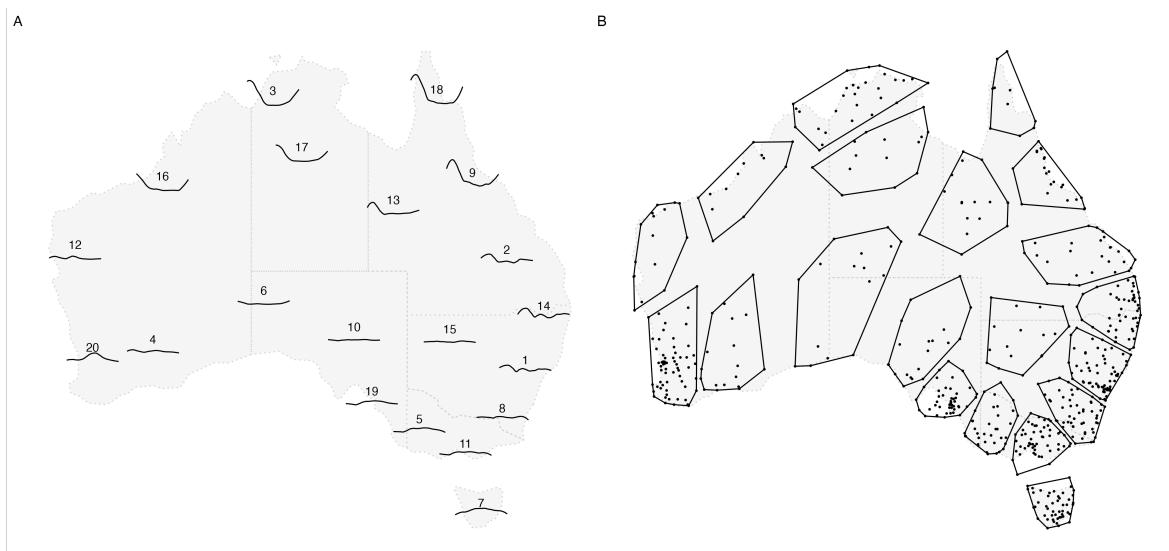


Figure 7: Profile of aggregated precipitation from 639 weather stations in Australia. Subplot A shows the glyph map of the weekly averaged precipitation of each cluster. The group number printed in the middle of y minor axis and can be used as a reference line to read the magnitude. Subplot B shows the station membership of each cluster.

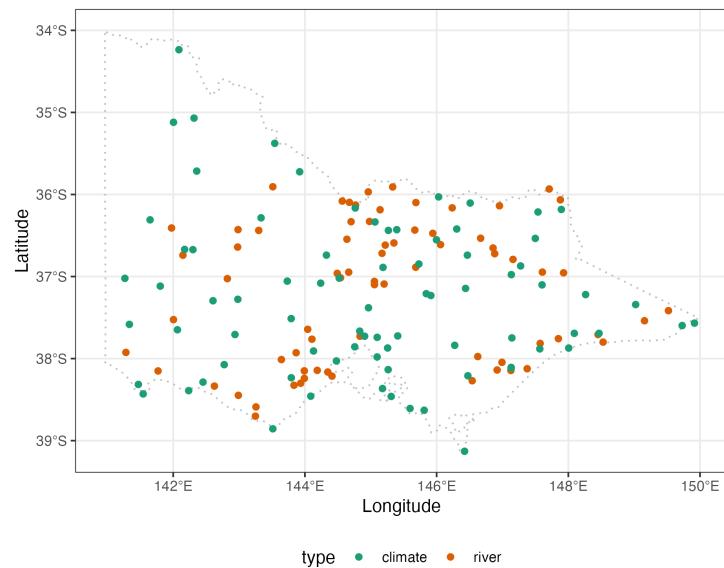


Figure 8: Location of weather stations and river gauges in Victoria, Australia.

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 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.

```
R> 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 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.

6.4. ERA5: climate reanalysis data

ERA5 data (Hersbach, Bell, Berrisford, Hirahara, Horányi, Muñoz-Sabater, Nicolas, Peubey, Radu, Schepers *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

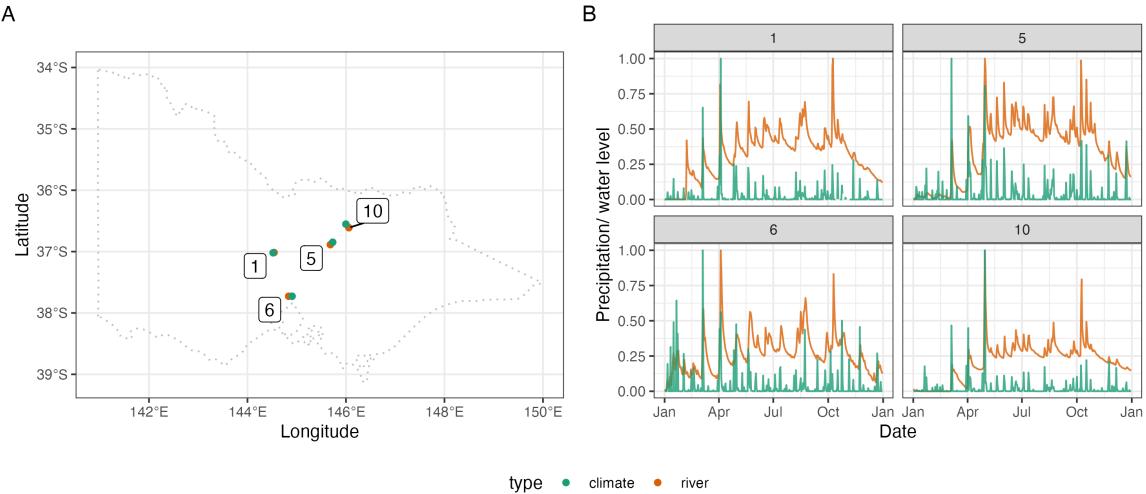


Figure 9: Matched weather stations and river gauges on the map (A) and across time (B). Precipitation and water level have been standardised between 0 and 1 to be displayed on the same scale. The increases in the precipitation is reflected by the water level.

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:

```
R> raw <- ncdf4::nc_open(here::here("data/era5-pressure.nc"))
R> 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, Soci, Nicolas, Bell, Berrisford, Dragani, Flemming, Haimberger, Healy, Hersbach, Horányi, Inness, Munoz-Sabater, Radu, and Schepers (2020) and Simmons, Hortal, Kelly, McNally, Untch, and Uppala (2005) for more details.

6.5. Interative graphic with **cubble**

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.

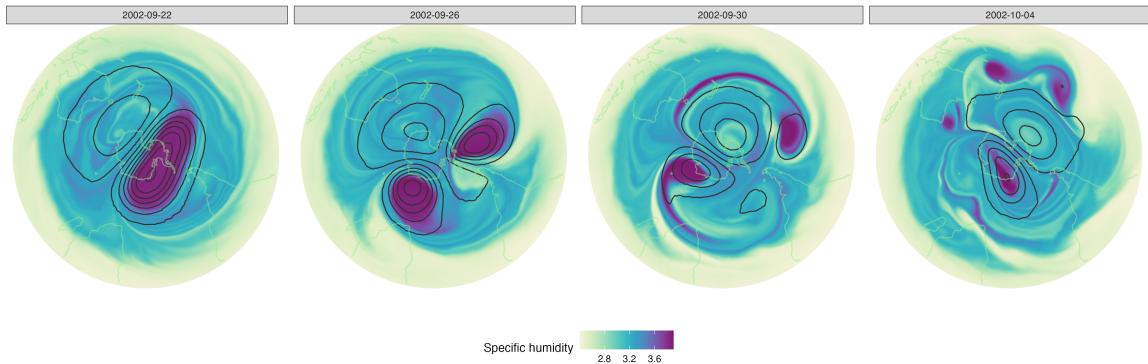


Figure 10: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020).

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` 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:

```
R> # data pre-processing
R> clean <- weatherdata::climate_full %>% ...
R>
R> # created SharedData instance for crosstalk
R> nested <- clean %>% SharedData$new(~id, group = "cubble")
R> long <- stretch(clean) %>% SharedData$new(~id, group = "cubble")
R>
R> # create the spatial and temporal view each with a ShareData instance
R> p1 <- nested %>% ...
R> p2 <- long %>% ...
R>
R> # Combine p1 and p2
R> crosstalk::bscols(plotly::ggplotly(p1), plotly::ggplotly(p2), ...)
```

In Figure 11, 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()**:

```
R> # data pre-processing
R> clean <- weatherdata::climate_full %>% ...
R>
R> # use the long form to create subplots for each station
R> df_id <- unique(clean$id)
R> p <- map(1:length(df_id), function(i){
+   dt <- clean %>% filter(id == df_id[i])
+   ggplot(dt) %>% ...
+ })
R>
R> # create nested form leaflet map with temperature band as subplots
R> nested <- tamp(clean)
R> leaflet(nested) %>%
+   addTiles() %>%
+   addCircleMarkers(group = "a", ...) %>%
+   leafpop::addPopupGraphs(graph = p, ...)
```

Figure 12 shows Figure 11 made made with leaflet and popups (Appelhans and Detsch 2021).

7. 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**. Another area for further development is to extend **cubble** to other spatial objects other than points.

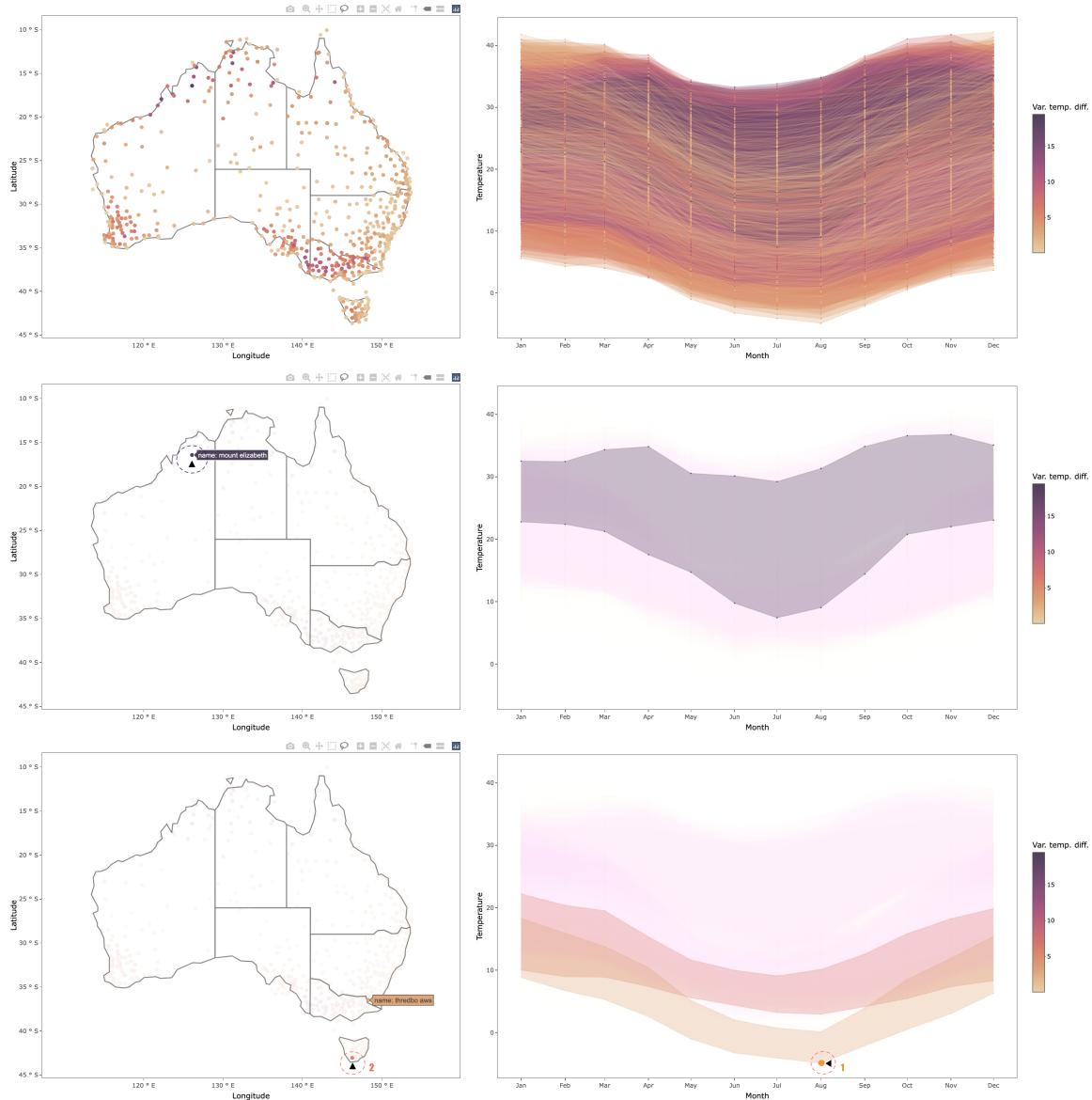


Figure 11: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a location with high variance on the map produces the plot in the second row. The maximum nad minimum 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.

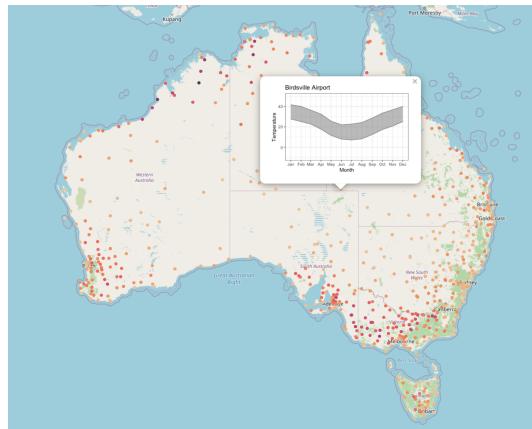


Figure 12: Same as Figure 11 with the temperature variation shown as a popup in the leaflet map.

8. 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/huizehang-sherry/paper-cubble>.

9. Appendix

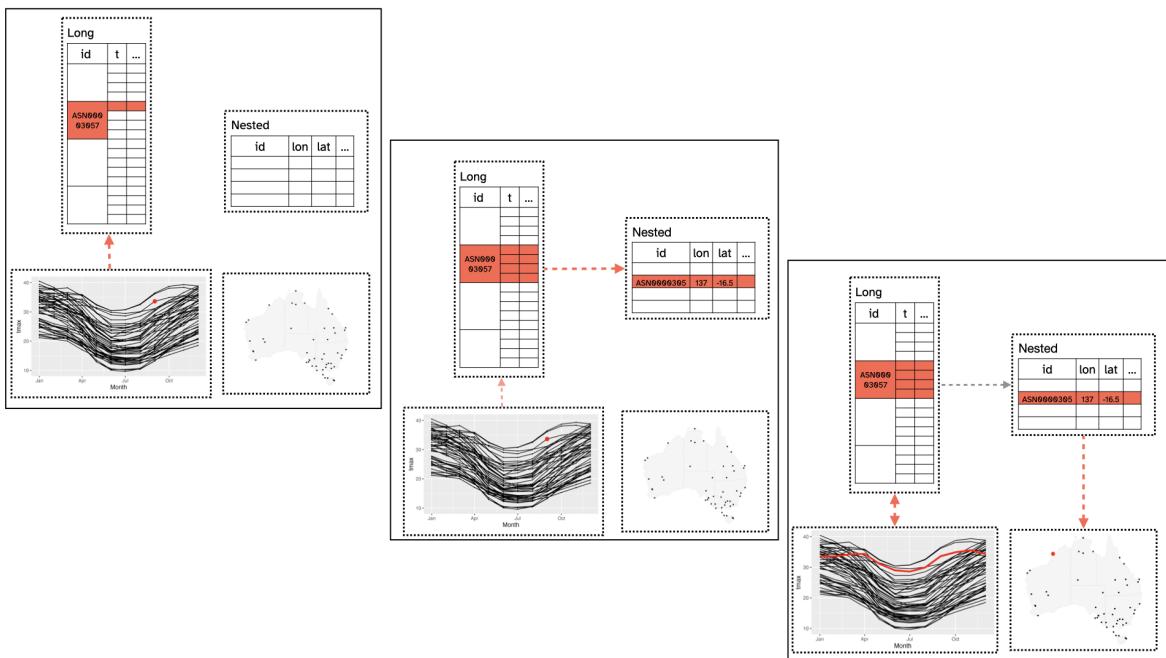


Figure 13: An illustration of the data model under interactive graphics with cubble. When a point on the time series is selected, the corresponding row in the long cubble will be activated. This will link to all the rows with the same id in the long cubble and the row in the nested cubble with the same id (middle). Both plots will be updated with the full line selected and the point highlighted on the map (right).

References

- Appelhans T, Detsch F (2021). *leafpop: Include Tables, Images and Graphs in Leaflet Pop-Ups*. R package version 0.1.0, URL <https://CRAN.R-project.org/package=leafpop>.
- Bach B, Dragicevic P, Archambault D, Hurter C, Carpendale S (2014). “A Review of Temporal Data Visualizations Based on Space-Time Cube Operations.” *Eurographics conference on visualization*, p. 19. URL <https://hal.inria.fr/hal-01006140/>.
- Buja A, Asimov D, Hurley C (1988). “Elements of A Viewing Pipeline.” *Dynamic Graphics Statistics*, p. 277.
- Buja A, Cook D, Swayne DF (1996). “Interactive High-dimensional Data Visualization.” *Journal of Computational and Graphical Statistics*, **5**(1), 78–99. URL <https://doi.org/10.2307/1390754>.
- Cheng X, Cook D, Hofmann H (2016). “Enabling Interactivity on Displays of Multivariate Time Series and Longitudinal Data.” *Journal of Computational and Graphical Statistics*, **25**(4), 1057–1076. URL <https://doi.org/10.1080/10618600.2015.1105749>.
- Cocchi M (2019). *Data Fusion Methodology and Applications*. Elsevier.
- Hersbach H, Bell B, Berrisford P, Hirahara S, Horányi A, Muñoz-Sabater J, Nicolas J, Peubey C, Radu R, Schepers D, et al. (2020). “The ERA5 Global Reanalysis.” *Quarterly Journal of the Royal Meteorological Society*, **146**(730), 1999–2049.
- Hufkens K, Stauffer R, Campitelli E (2019). “The **ecwmfr** package: An Interface to ECMWF API endpoints.” doi:[10.5281/zenodo.2647541](https://doi.org/10.5281/zenodo.2647541). URL <https://bluegreen-labs.github.io/ecmwfr/>.
- JR Herring (2011). “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture.” *Technical report*, Open Geospatial Consortium Inc. URL http://portal.opengeospatial.org/files/?artifact_id=25355.
- Lu M, Appel M, Pebesma E (2018). “Multidimensional Arrays for Analysing Geoscientific Data.” *ISPRS International Journal of Geo-Information*, **7**(8), 313. ISSN 2220-9964. doi: [10.3390/ijgi7080313](https://doi.org/10.3390/ijgi7080313). URL <http://www.mdpi.com/2220-9964/7/8/313>.
- McIntosh AI, Jenkins HE, White LF, Barnard M, Thomson DR, Dolby T, Simpson J, Streicher EM, Kleinman MB, Ragan EJ, et al. (2018). “Using Routinely Collected Laboratory Data to Identify High Rifampicin-Resistant Tuberculosis Burden Communities in the Western Cape Province, South Africa: A Retrospective Spatiotemporal Analysis.” *PLoS Medicine*, **15**(8), e1002638.
- Michna P, Woods M (2013). “**RNetCDF**: A package for Reading and Writing NetCDF Datasets.” *The R Journal*, **5**(2), 29–36.
- Michna P, Woods M (2021). *RNetCDF: Interface to 'NetCDF' Datasets*. R package version 2.5-2, URL <https://CRAN.R-project.org/package=RNetCDF>.

- Pebesma E (2012). “**spacetime**: Spatio-Temporal Data in R.” *Journal of Statistical Software*, **51**(7), 1–30. URL <https://doi.org/10.18637/jss.v051.i07>.
- Pebesma E (2021). **stars**: *Spatiotemporal Arrays, Raster and Vector Data Cubes*. R package version 0.5-2, URL <https://CRAN.R-project.org/package=stars>.
- Pierce D (2019). **ncdf4**: *Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files*. R package version 1.17, URL <https://CRAN.R-project.org/package=ncdf4>.
- Simmons A, Hortal M, Kelly G, McNally A, Untch A, Uppala S (2005). “ECMWF Analyses and Forecasts of Stratospheric Winter Polar Vortex Breakup: September 2002 in the Southern Hemisphere and Related Events.” *Journal of the Atmospheric Sciences*, **62**(3), 668 – 689. doi:[10.1175/JAS-3322.1](https://doi.org/10.1175/JAS-3322.1). URL <https://journals.ametsoc.org/view/journals/atsc/62/3/jas-3322.1.xml>.
- Simmons A, Soci C, Nicolas J, Bell B, Berrisford P, Dragani R, Flemming J, Haimberger L, Healy S, Hersbach H, Horányi A, Inness A, Munoz-Sabater J, Radu R, Schepers D (2020). “Global Stratospheric Temperature Bias and Other Stratospheric Aspects of ERA5 and ERA5.1.” (859). doi:[10.21957/rcxqfmgo](https://doi.org/10.21957/rcxqfmgo). URL <https://www.ecmwf.int/node/19362>.
- Stuart EA (2010). “Matching Methods for Causal Inference: A Review and a Look Forward.” *Statistical Science*, **25**(1), 1.
- Sumner M (2020). **tidync**: *A Tidy Approach to 'NetCDF' Data Exploration and Extraction*. R package version 0.2.4, URL <https://CRAN.R-project.org/package=tidync>.
- Sutherland P, Rossini A, Lumley T, Lewin-Koh N, Dickerson J, Cox Z, Cook D (2000). “**Orca**: A Visualization Toolkit for High-dimensional Data.” *Journal of Computational and Graphical Statistics*, **9**(3), 509–529. URL <https://www.tandfonline.com/doi/abs/10.1080/10618600.2000.10474896>.
- Wang E, Cook D, Hyndman RJ (2020). “Calendar-based Graphics for Visualizing People’s Daily Schedules.” *Journal of Computational and Graphical Statistics*, **29**(3), 490–502.
- Wickham H (2014). “Tidy Data.” *Journal of Statistical Software*, **59**(10), 1–23. URL <https://doi.org/10.18637/jss.v059.i10>.
- Wickham H, Hofmann H, Wickham C, Cook D (2012). “Glyph-Maps for Visually Exploring Temporal Patterns in Climate Data and Models.” *Environmetrics*, **23**(5), 382–393.
- Xie Y (2015). *Dynamic Documents with R and knitr*. 2nd edition. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1498716963, URL <https://yihui.name/knitr/>.
- Xie Y, Allaire J, Grolemund G (2018). *R Markdown: The Definitive Guide*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1138359338, URL <https://bookdown.org/yihui/rmarkdown>.
- Xie Y, Hofmann H, Cheng X (2014). “Reactive Programming for Interactive Graphics.” *Statistical Science*, **29**(2), 201 – 213. doi:[10.1214/14-STS477](https://doi.org/10.1214/14-STS477). URL <https://doi.org/10.1214/14-STS477>.

Affiliation:

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

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

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

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

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