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 variables can be divided into time invariant ones that can be identified by the spatial identifier, or the time varying ones that cross-identified by the spatial and temporal identifiers. In this paper, a new data structure, `cubble`, is proposed for manipulating and visualising spatio-temporal data. The new data structure uses two forms to organise time invariant and varying variables so that the spatial and temporal dimension of the data can be manipulated separately while keep synchronised. Advanced considerations are given to illustrate data with hierarchical structure, data matching, and how cubble fits with the interactive graphics pipeline. Examples are given to analysing Australia meteorology data, river level data, and climate reanalysis (ERA5).

Keywords: spatio temporal, R, climate weather station, data analysis.

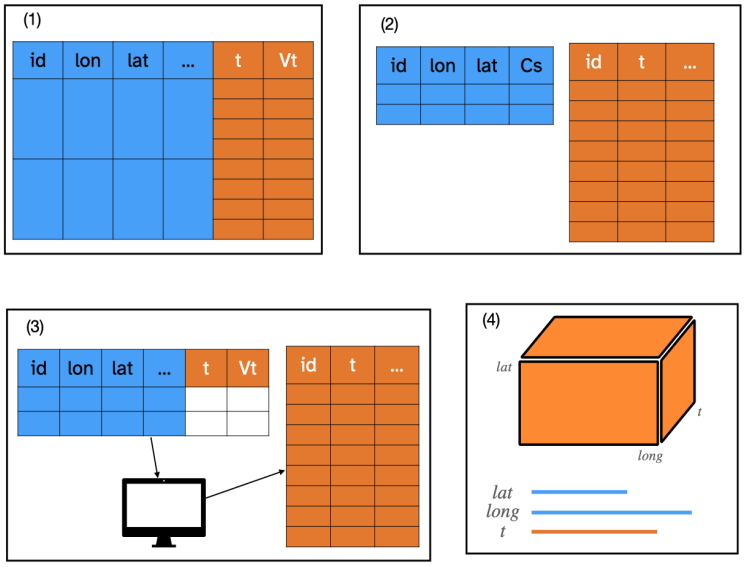

Figure 1: An illustration of different spatio-temporal data format: (1) a single table for all the variables; (2) separate tables for spatial and temporal variables; (3) a table of metadata used to query the database and a separate table for queried data; and (4) an array data structure of cube or hypercube.

1. Introduction

Spatio-temporal data concerns variables in the space across time. Elaborate on variable, space, and time. If the time interval is regular and each site has fixed point location, it can also be thought of as a data cube with three axes: spatial identifier, temporal identifier, and variable. In this cubical layout, variables can be those that are invariant to time, i.e. the longitude and latitude coordinates, or those that varying across time. While conceptually the data format is relatively simple, the actual data object that comes to the analysts can be in various forms in practice. Figure 1 shows four examples where the first one combines the time invariant and varying variables into a single table through duplicating the time invariant variables; 2) splits those variables into two separate sheets; 3) . . . ; and lastly 4) considers the array-based data, like 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.

think through a cube of long x lat x time vs. id x time x variable

more beginning like "working with such data different authors ..." Currently, there's various different spatio-temporal data structures in R includes: **spacetime** (Pebesma 2012), which proposes four space-time layouts: Full grid (STF), sparse grid(STS), irregular (STI), and trajectory (STT). **spacetime** is built on **sp** (Pebesma and Bivand 2005) and **xts** (Ryan and Ulrich 2020), both of which have been replaced by more recent implementations. **spatstat** (Baddeley and Turner 2005) implements a `ppp` class for point pattern data; and more recent,

stars (Pebesma 2021) implements a spatio-temporal for vector and raster spatio-temporal data.

In spatio-temporal data analysis, analysts will receive one of the various data formats depends on the data provider and need to manually rearrange the data to fit into one of the data object in R before starting any analysis on the data itself. Looking at spatial or temporal data analysis, data objects like **sf** (Pebesma 2018) and **tsibble** (Wang, Cook, and Hyndman 2020b) have smoothed the data analysis in these two domains while there has not yet been a spatio-temporal data structure that makes it easy to explore spatio-temporal data and this creates a gap in the software development. The requirement for such a tool is important given the ubiquity of spatio-temporal vector data in the wild: climate observations from weather stations, air quality variables from air monitoring stations, or any variables that are measured at fixed location across time.

This paper describes the implementation of a new spatio-temporal data structure: **cubble**. **Cubble** implements a data structure that uses two forms to manage the switch between spatial and temporal dimension. With this structure, users can manipulate the spatial or temporal dimension separately, while leaves the linking of two dimensions to **cubble**. 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 workflow of data manipulation in cubble. Section 3 explains how cubble deals with some advanced considerations including data with hierarchical structure, data matching, and how cubble fits with existing static and interactive visualisation tools. Section 4 gives examples of the features introduced in the previous two sections with climate and hydrology data. An example on how cubble handle NetCDF data is also provided. Section 5 concludes the paper.

2. The cubble package

In this section, the two formats cubble uses to arrange spatio-temporal data will be presented Then operations will be introduced, with short examples, to illustrate how to work with these two formats. Lastly, a subsection will be dedicated to how existing packages, in spatial and temporal analysis, fit in with cubble.

In cubble, data can represented in two formats: nested form and long form, and Figure 2 sketches the two forms with the associated attributes. The decision on which form to use in data manipulation 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.

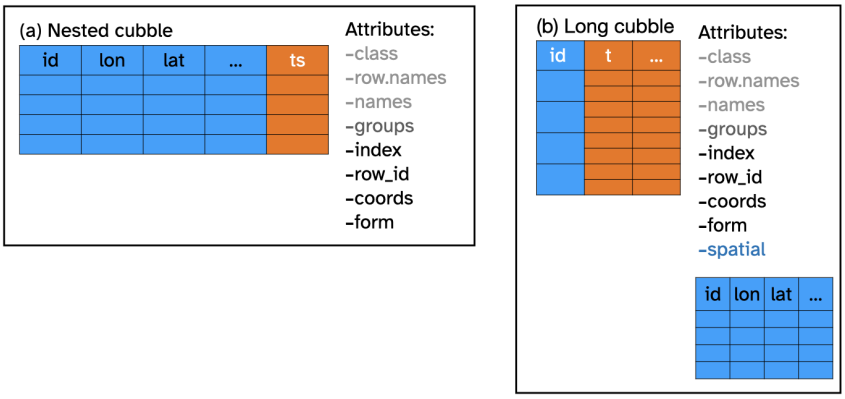

Figure 2: An illustration of the nested and long form in cubble. The nested form defines each site in a row and nests the time varying variables into a single column `ts`. The long form cubble uses `id` and `t` to identify each row and store the time invariant variables as an attribute, `spatial`.

2.1. Create a cubble in the nested form

To use functions in the `cubble` package, an analyst will need to first 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. The cubble created by default is in the nested form.

Before the formal examples in Section 4, each function introduced in this section will be demonstrated with a short example with a data that contains five Australia weather stations, a subset of hundreds of weather stations in Australia. This dataset contains spatial information of each station: station id, latitude, longitude, elevation, station name and World Meteorology Organisation ID and also daily temporal information: 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]`.

2.2. Stretch a nested cubble into the long form

The nested format is convenient for those operations whose output doesn't contain a time dimension, for those that are identified by both the spatial and temporal identifier, a long cubble is more suitable. Analysts will need to switch the nested cubble into a long cubble for such operation and `stretch()` is designed for this purpose: it first extracts all the spatial variables into a separate tibble and then unnests the `ts` column. This code switches 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]`. The spatial variables are still part of the data object but as a `spatial` attribute:

```
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
```

2.3. 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]>
```

2.4. Migrate spatial variables to a long cubble

Sometimes, analysts may need the spatial and temporal variable to be in the same table to apply some variable transformation. 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> <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 operation should generally be used in the last step of the analysis since it is a temporary operation, meaning meaning these added spatial variables will disappear if switched to the nested form and then switched back:

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

```
# cubble:  date, id [5]: long form
# bbox:    [115.97, -32.94, 133.55, -12.42]
# spatial: long [dbl], lat [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
```

2.5. Compatibility with existing packages

The last four subsections introduce operations specific to the `cubble` class and this section will demonstrate how the `cubble` class interacts with existing packages, specifically, `dplyr`, `tsibble`, `sf`)`s2`), and `netcdf4`.

dplyr

The `dplyr` package has provided many tools for common data wrangling tasks and these operations are useful in the spatio-temporal context. `cubble` provides its method for the following `dplyr` verbs to wrangle data 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`)

more work on groupby The grouping pair, `group_by` and `ungroup`, can only be used in the long form to define grouping on the time but not in the nested form.

tsibble

Tsibble is a temporal data structure that uses `index` and `key` to identify the time and different series. Cubble can be seen as a natural extension of tsibble for spatio-temporal data with an additional coordinates component. The use of two forms avoids the duplicates of spatial variables at each time stamp and hence, in the visualisation, avoid drawing duplicated points with the same coordinate. To cast a `tsibble` into a `cubble`, only the `coords` argument needs to be supplied:

```
R> # demonstrate with a tsibble created from climate_flat
R> raw <- climate_flat %>%
+   tsibble::as_tsibble(key = id, index = date)
R>
R> dt <-  raw %>%
+   cubble::as_cubble(coords = c(long, lat))
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
  <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 created nested form, each element of the list-column `ts` is of `tbl_ts` class and operations available to the tsibble class is still valid under cubble. For example, the code below calculate 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)))
```

```
# 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_t~      25.7    38.6
2 ASN00010311 -31.9  117. 179   york            94623 <tbl_t~      26.2    51.1
3 ASN00010614 -32.9  117. 338   narrogin        94627 <tbl_t~      23.7    45.4
4 ASN00014015 -12.4  131.  30.4 darwin airport  94120 <tbl_t~      33.1     3.02
5 ASN00015131 -17.6  134. 220   elliott         94236 <tbl_t~      34.6    24.7
```

sf and s2

As a spatial data object, `sf` introduces a simple feature geometry list-column (`sfc`) to the data frame and provides spatial operations on various geometry types (`POINT`, `LINESTRING`, `POLYGON`, `MULTIPOLYGON` etc). While these operations are valueable for spatio-temporal data analysis, an `sf` object usually don't contain temporal variables and hence can't be directly cast into a `cubble`. However, `cubble` can contain `sfc` columns in the nested form and spatial operations built on the `sfc` class are available. The following example shows how to create an `sfc` column of `POINT` type based on latitude and longitude in cubble. The state `MULTIPOLYGON` is then added based on spatial operation `st_within` before making a coordinate transformation.

```
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_ste
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 airpo~  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   narrogin      94627 <tibble~ (117.1797 -32.9342)
4 ASN00014015 -12.4  131.  30.4 darwin airp~  94120 <tibble~ (130.8925 -12.4239)
5 ASN00015131 -17.6  134. 220   elliott       94236 <tibble~ (133.5407 -17.5521)
# ... with 1 more variable: state <MULTIPOLYGON [m]>
```

An **s2** `lnglat` vector can similarly be created in cubble for using `s2`-prefixed functions:

```
R> library(s2)
R> # western australia map
R> wa <- ozmaps::abs_ste %>% 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"))
```

Memory limit with NetCDF data in cubble depends on longitude grid point x latitude grid point x time grid point x number of variable. Cubble can handle slightly more than 300 x 300 (longitude x longitude) grid points for 3 variables in one year and spatial grid can be reduced to trade for longer time period 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 grid 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
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))
```

3. Other features and considerations

3.1. 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

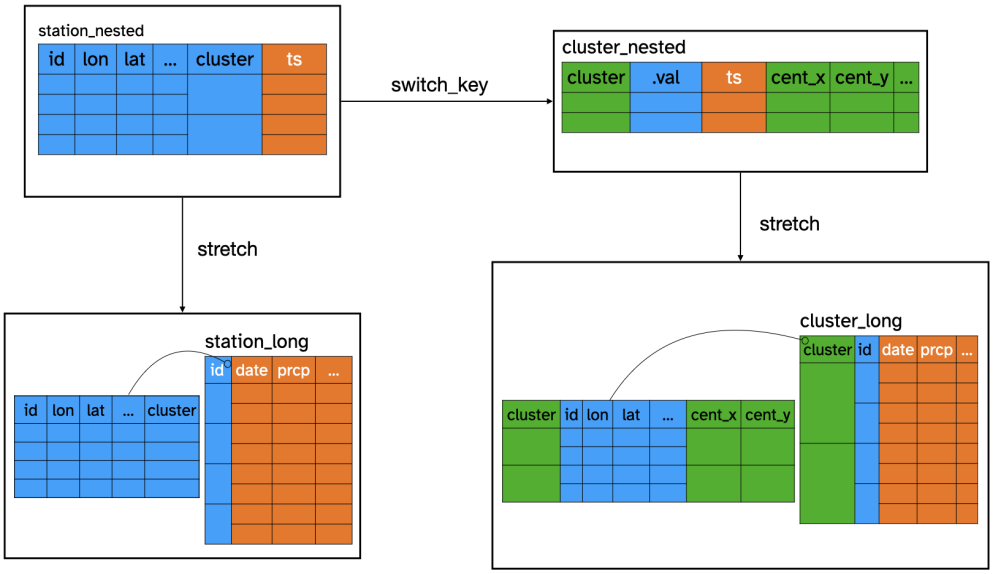

Figure 3: An illustration of the original and cluster level cubble in the nested form long form for hierarchical structure data. `switch_key()` changes the station level cubble into a cluster level cubble and both can be stretched into the long form.

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`.

3.2. Data fusion and matching

more beginning Cubble provides functionality for matching sites from two data sources. To be a valid pair of matching, the matched pair from different data sources need to:

- be spatially close to each other, and
- have similar temporal movement

`match_sites()` matches data based on these two criteria: it 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

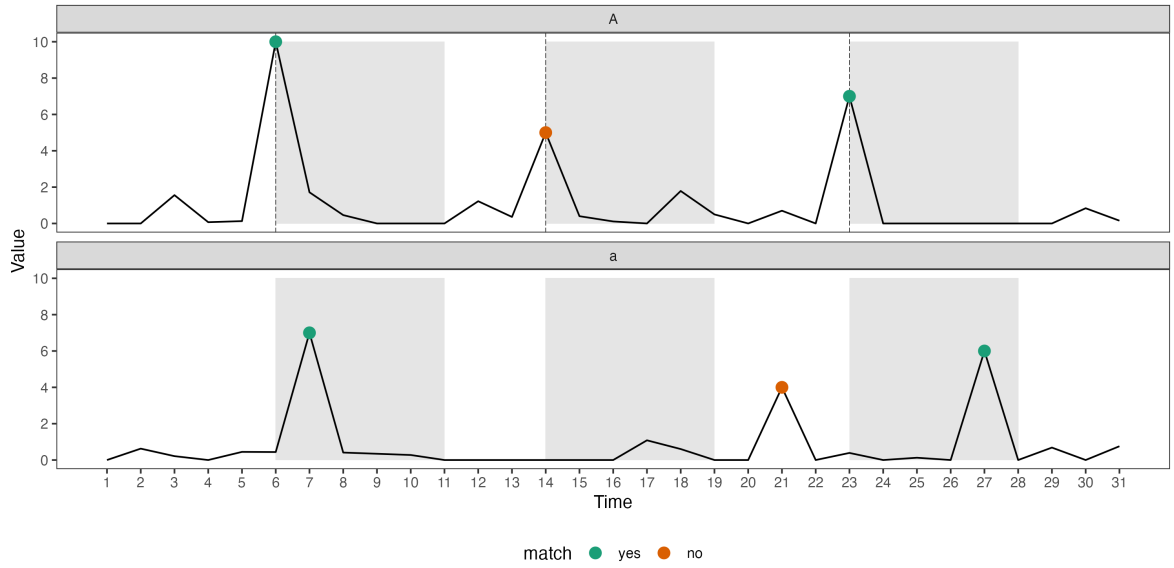

Figure 4: An illustration of temporal matching in cubble. Three highest peaks are identified in each series and intervals are constructed on series `A`. Two peaks in series `a` fall into the intervals and hence the two series are considered to have 2 matches.

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

This functionality can be seen as a matching exercise (Stuart 2010; McIntosh, Jenkins, White, Barnard, Thomson, Dolby, Simpson, Streicher, Kleinman, Ragan *et al.* 2018) in the spatio-temporal domain to pair up nearby observations. It can also be considered as a form of data fusion (Cocchi 2019), where data from multiple sources are combined together.

3.3. Interactive graphics

Cubble fits in naturally with the interactive graphic pipeline discussed in the literature (Buja, Asimov, and Hurley 1988; Buja, Cook, and Swayne 1996; Sutherland, 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).

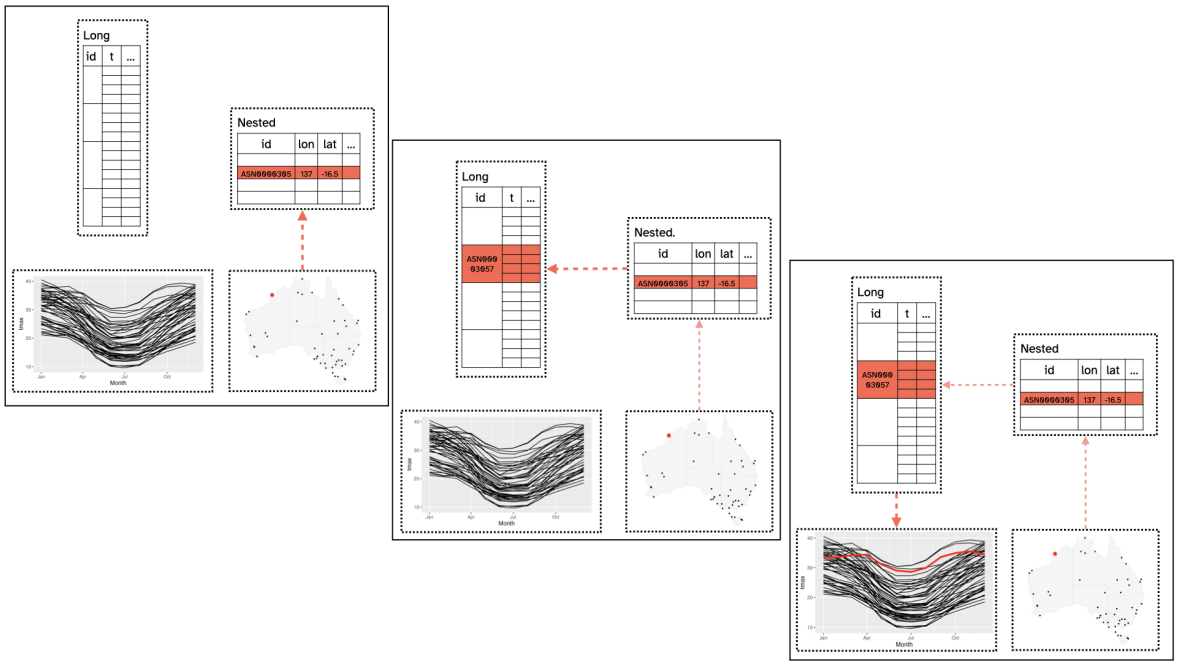

Figure 5: An illustration of the data model under interactive graphics with cubble. The line plot and the map is made separately with the long and nested cubble. When a station is selected on the map (left), the corresponding row in the nested cubble will be activated. This will link to all the rows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

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.

3.4. 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 2020a) 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()`.

4. Examples

4.1. 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_ste %>%
+   filter(NAME %in% c("Victoria", "New South Wales"))
R>
R> ggplot() +
+   geom_sf(data = nsw_vic,
+           fill = "transparent", color = "grey", linetype = "dotted") +
+   geom_glyph(data = tmax,
+              aes(x_major = longitude, x_minor = month,
+                  y_major = latitude, y_minor = tmax,
+                  group = interaction(id, group), color = group),
+              width = 1, height = 0.5) +
+   ...
```

4.2. Australia precipitation pattern in 2020

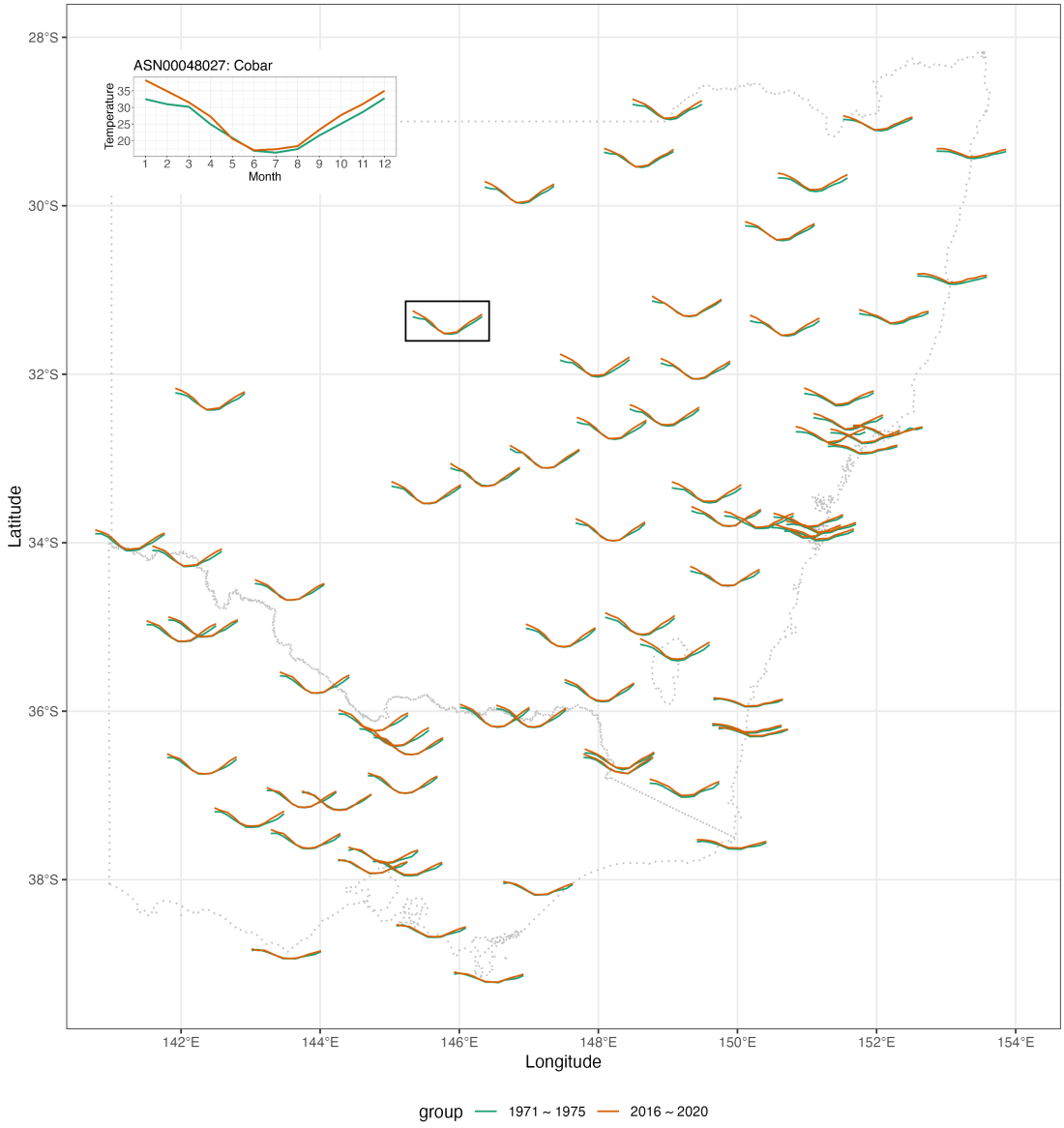

Figure 6: A glyph map of the average maximum temperature by month of Victoria and New South Wales weather stations in Australia. On the top left corner is an insetted plot of station Cobar highlighted in the black box.

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.

4.3. 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.

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

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

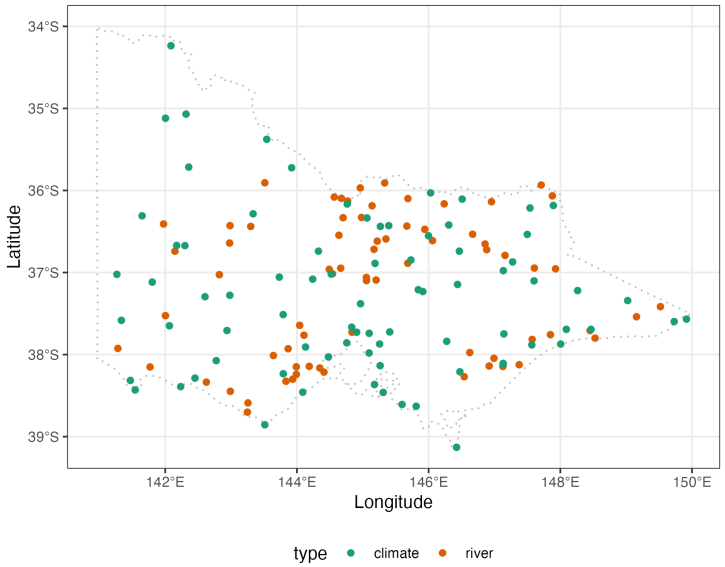

Figure 8: Location of weather stations and river gauges in Victoria, Australia.

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~ -36.9  146. river   6.15     5 <tibble ~      21
2 404207      HOLLAND CREEK @ ~ -36.6  146. river   8.54    10 <tibble ~      21
3 ASN00082042 strathbogie       -36.8  146. clima~  6.15     5 <tibble ~      21
4 ASN00082170 benalla airport   -36.6  146. clima~  8.54    10 <tibble ~      21
5 230200      MARIBYRNONG RIVE~ -37.7  145. river   6.17     6 <tibble ~      19
6 ASN00086038 essendon airport  -37.7  145. clima~  6.17     6 <tibble ~      19
7 406213      CAMPASPE RIVER @~ -37.0  145. river   1.84     1 <tibble ~      18
8 ASN00088051 redesdale         -37.0  145. clima~  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.

4.4. 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 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"))
```

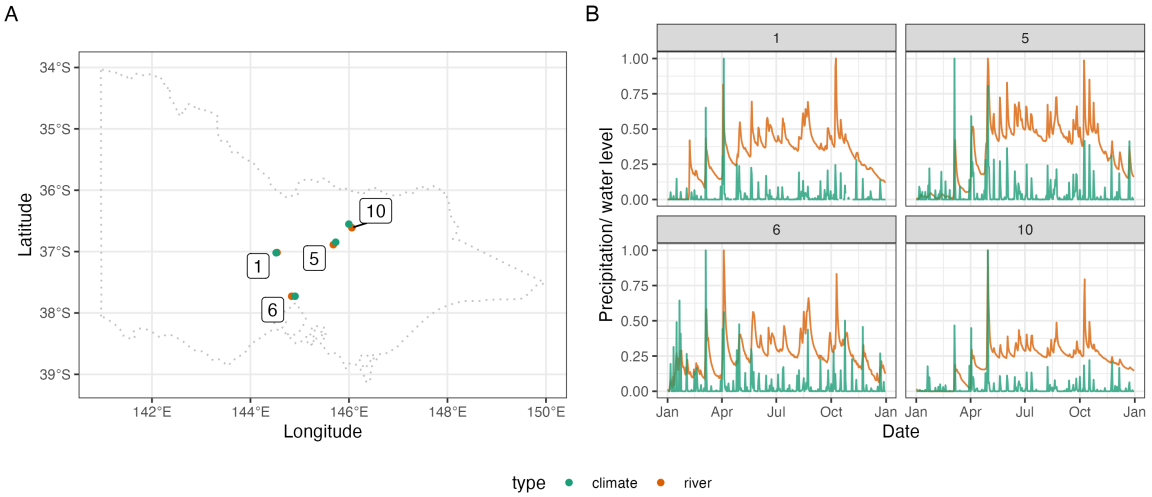

Figure 9: Matched weather stations and river gauges on the map (A) and across time (B). Precipitation and water level have been standardised between 0 and 1 to be displayed on the same scale. The increases in the precipitation is reflected by the water level.

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.

4.5. 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.

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

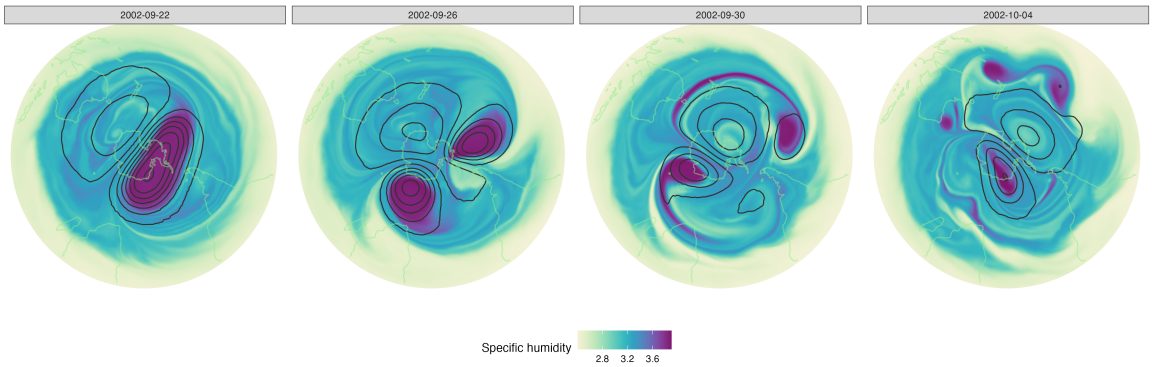

Figure 10: A reproduction of the second row (ERA5 data) of Figure 19 in Hersbach et al (2020).

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
```

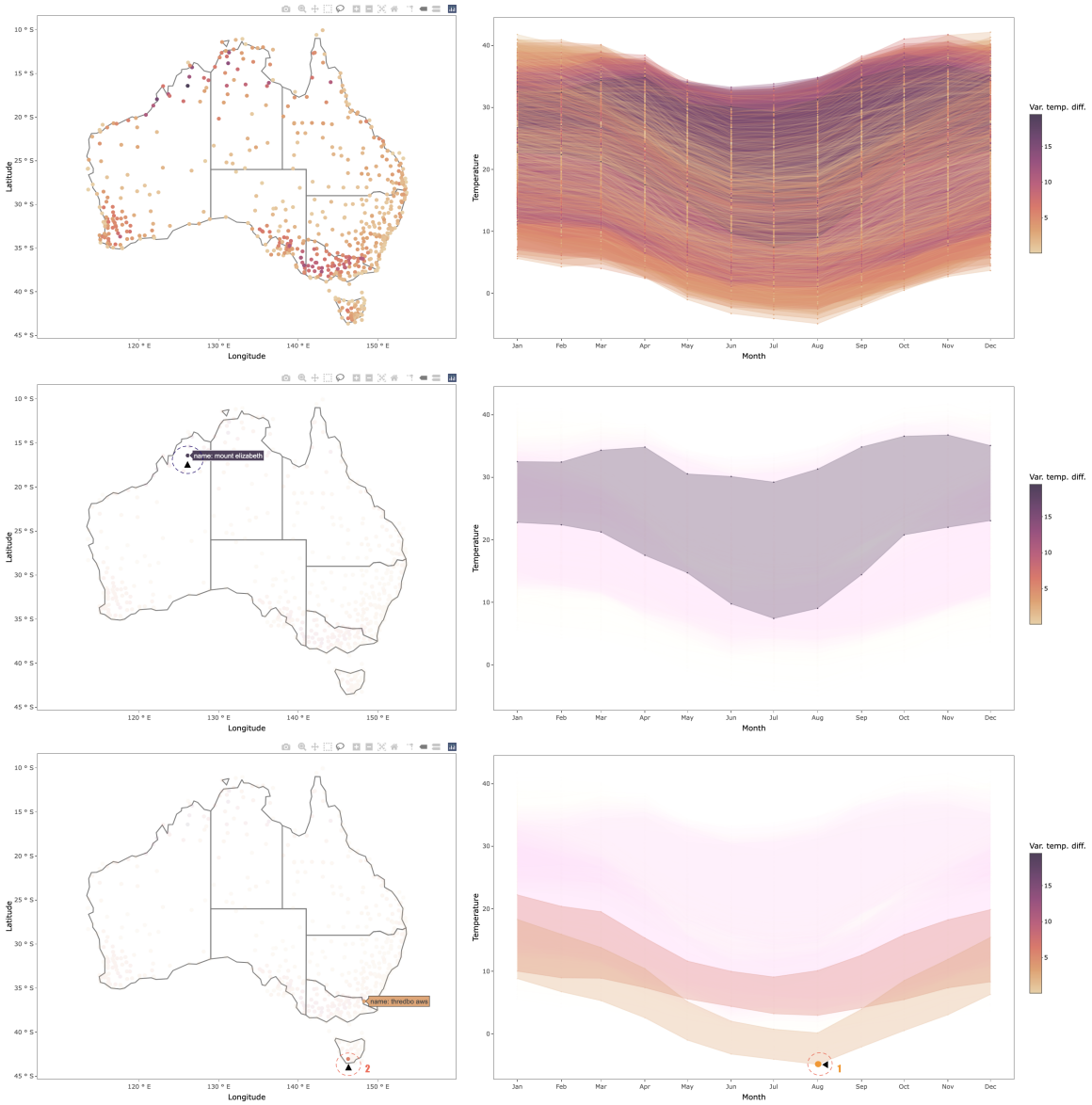

Figure 11: Exploring temperature variation using linking of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiles. Selecting a location with high variance on the map produces the plot in the second row. The maximum nad minimum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

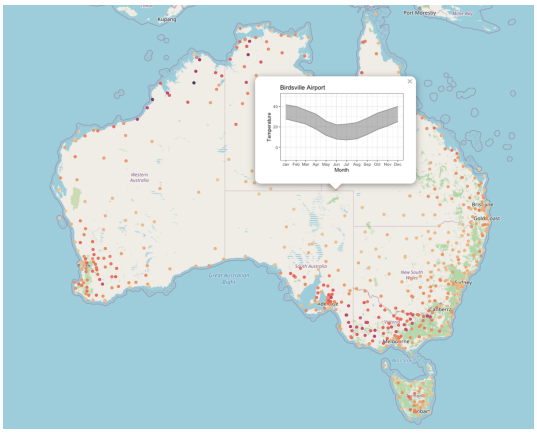

Figure 12: Same as Figure 11 with thetemperature variation shown as a popup in the leaflet map.

```
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).

5. 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.

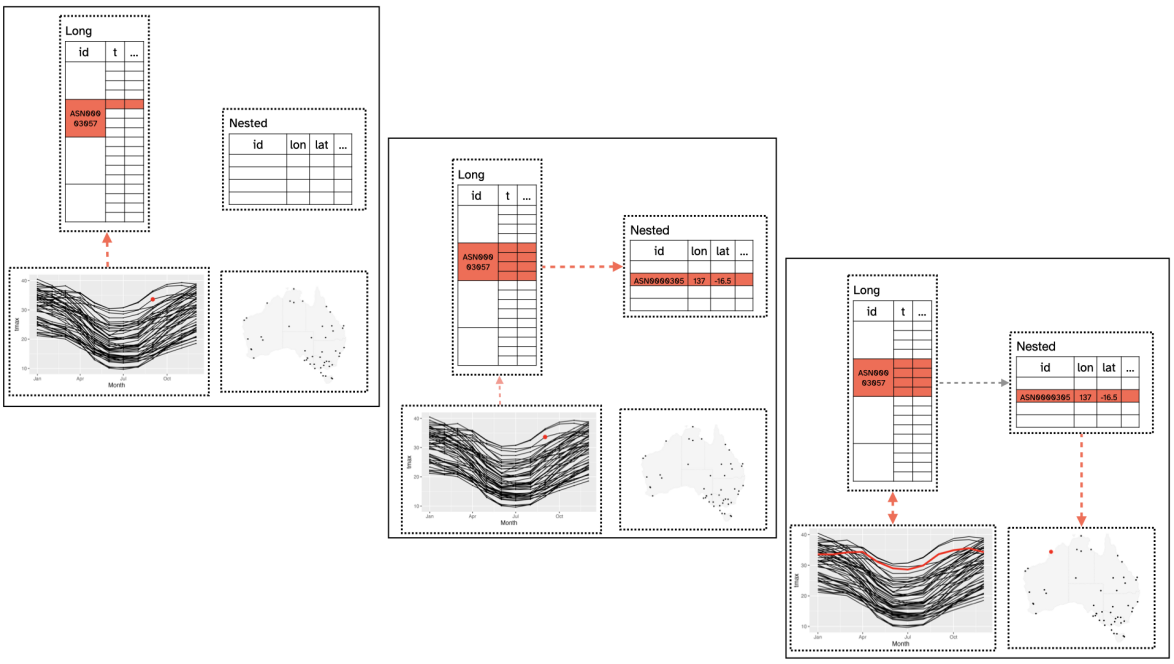

Figure 13: An illustration of the data model under interactive graphics with cubble. When a point on the time series is selected, the corresponding row in the long cubble will be activated. This will link to all the rows with the same id in the long cubble and the row in the nested cubble with the smae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

6. Acknowledgement

This work is funded by the Commonwealth Scientific and Industrial Research Organisation (CSIRO) Data61 Scholarship and started while Nicolas Langrené was affiliated with CSIRO's Data61. The article is created using **knitr** (Xie 2015) and **rmarkdown** (Xie, Allaire, and Grolemund 2018) in R. The source code for reproducing this paper can be found at: `https://github.com/huizezhang-sherry/paper-cubble`.

7. Appendix

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.

Baddeley A, Turner R (2005). "**Spatstat**: An R Package for Analyzing Spatial Point Patterns." *Journal of Statistical Software*, **12**(6), 1–42. URL https://doi.org/10.18637/jss.v012.i06.

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. URL https://bluegreen-labs.github.io/ecmwfr/.

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

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

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

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.

Ryan JA, Ulrich JM (2020). ***xts**: eXtensible Time Series*. R package version 0.12.1, URL https://CRAN.R-project.org/package=xts.

Simmons A, Hortal M, Kelly G, McNally A, Untch A, Uppala S (2005). "ECMWF Analyses and Forecasts of Stratospheric Winter Polar Vortex Breakup: September 2002 in the Southern Hemisphere and Related Events." *Journal of the Atmospheric Sciences*, **62**(3), 668 – 689. doi:10.1175/JAS-3322.1. URL https://journals.ametsoc.org/view/journals/atsc/62/3/jas-3322.1.xml.

Simmons A, Soci C, Nicolas J, Bell B, Berrisford P, Dragani R, Flemming J, Haimberger L, Healy S, Hersbach H, Horányi A, Inness A, Munoz-Sabater J, Radu R, Schepers D (2020). "Global Stratospheric Temperature Bias and Other Stratospheric Aspects of ERA5 and ERA5.1." (859). doi:10.21957/rcxqfmg0. URL https://www.ecmwf.int/node/19362.

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

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

Sutherland P, Rossini A, Lumley T, Lewin-Koh N, Dickerson J, Cox Z, Cook D (2000). "**Orca**: A Visualization Toolkit for High-dimensional Data." *Journal of Computational and Graphical Statistics*, **9**(3), 509–529. URL https://www.tandfonline.com/doi/abs/10.1080/10618600.2000.10474896.

Wang E, Cook D, Hyndman RJ (2020a). "Calendar-based Graphics for Visualizing People's Daily Schedules." *Journal of Computational and Graphical Statistics*, **29**(3), 490–502.

Wang E, Cook D, Hyndman RJ (2020b). "A New Tidy Data Structure to Support Exploration and Modeling of Temporal Data." *Journal of Computational and Graphical Statistics*, **29**(3), 466–478. URL https://doi.org/10.1080/10618600.2019.1695624.

Wickham H, 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. URL https://doi.org/10.1214/14-STS477.

Affiliation:

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

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

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

Nicolas Langrené
BNU-HKBU United International College
2000 Jintong Road, Tangjiawan, Zhuhai, Guangdong Province, China
E-mail: nicolas@langrene.com

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd