Appendix to 'cubble: An R Package for Organizing and Wrangling Multivariate Spatio-temporal Data'

Sherry Zhang, Dianne Cook, Ursula Laa, Nicolas Langrené, Patricia Menéndez

2022-06-14

This is the supplementary material for the main paper, containing an extended example following example 5.2 to highlight how **cubble** can be used to deal with data that has a hierarchical structure. It also describes the process to create linking plots. Furthermore, this appendix contains additional information about the data sources and the necessary code for extracting and preparing the data sets used in the paper with the goal to ensure reproducibility.

1 Extension of example 5.2: Australian precipitation pattern in 2020

In example 5.2, some overlapping of the glyphs occurred for a number of stations on the right hand side of the map in Figure 6. This is a problem when mapping time series or other glyphs corresponding to locations that are geographically closed on the map. In some cases it is better to display such information at an aggregated level by grouping the data adequately before exhibiting the information on a figure. The example below shows how can spatio-temporal data be organised at different hierarchical levels so that information can be grouped both temporal and spatially using `switch_key()`. The goal of this example is to highlight how easy is to move across the data hierarchy using **cubble**.

The data `climate_full`, also extracted from the GHCN, records daily precipitation and maximum/minimum temperature for 640 stations in Australia from 2016 to 2020. A simple k-means algorithm based on the distance matrix between stations is used to create 20 clusters. The data `station_nested` is a nested cubble with a cluster column indicating the group to which each station belongs. More advanced clustering algorithms can be used as well, as long as they provide a mapping from each station to a cluster.

```
station_nested <- climate_full %>% mutate(cluster = ...)
```

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

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

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

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

Long form cubble at both levels can be accessed through stretching the nested form. With access to both station and cluster-level cubbles, various plots can be made to understand the cluster. Figure 1 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.

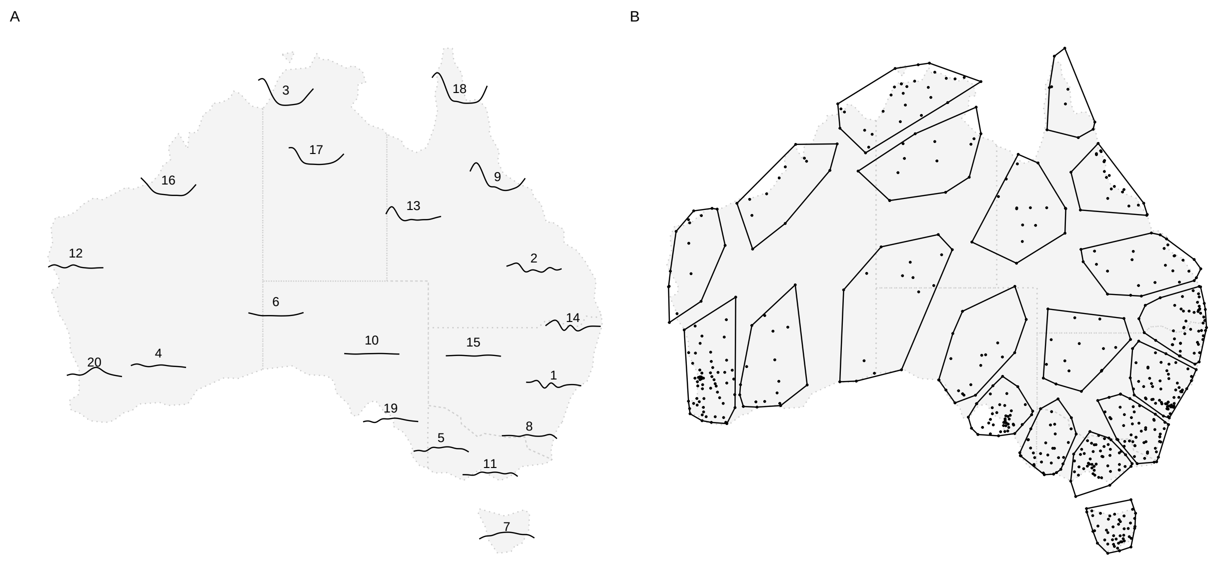

Figure 1: Profile of aggregated precipitation at 639 weather stations in Australia. Subplot A shows the glyph map of the weekly averaged precipitation of each cluster. The group number is printed in the middle of the y minor axis and can be used as a reference line to read the magnitude. Subplot B shows the station membership of each cluster.

2 Extension to section 5.5: Interactive graphics

Figure 2 in this section can also be made using `cubble` and `leaflet`, in which case the temperature range is displayed as a small subplot upon clicking on the map. This procedure involves first creating the popup plots from the long form cubble as a vector and then adding these plots to a leaflet map created from the nested cubble, with `leafpop::addPopupGraphs()`:

```r
# data pre-processing
clean <- climate_full %>% ...

# use the long form to create subplots for each station
df_id <- unique(clean$id)
p <- map(1:length(df_id), function(i){
  dt <- clean %>% filter(id == df_id[i])
  ggplot(dt) %>% ...
})

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

Figure 2 shows the same information as Figure 9 but using leaflet and popups.

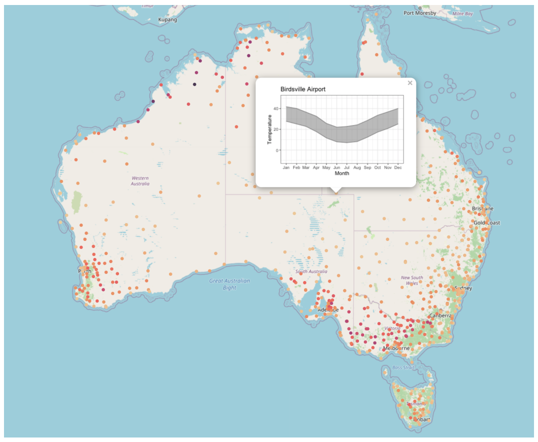

Figure 2: Same as Figure 11 except the temperature variation is now shown as a popup in the leaflet map.

3 Additional illustration on multiple linked plots

This figure is a supplement to Section 4.3 of the main paper, illustrating how linking from the time series plot to the map is achieved.

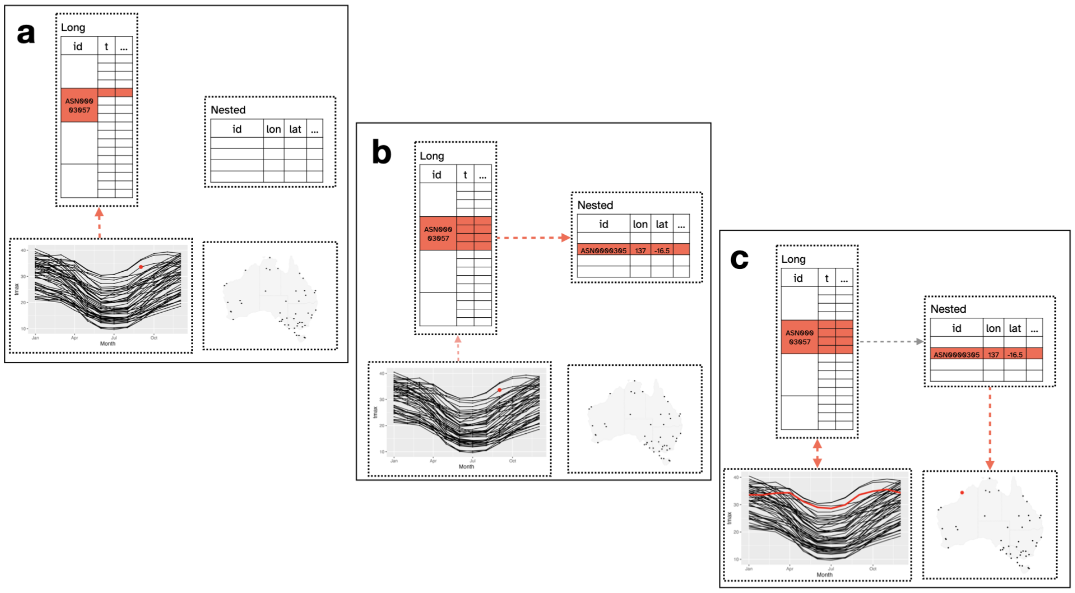

Figure 3: Linking between multiple plots. The line plots and the map are constructed from shared crosstalk objects (long and nested cubbles). When a point on the time series is selected, the corresponding row in the long cubble will be activated (a). This will link to all the rows with the same id in the long cubble and the row in the nested cubble with the same id (b). Both plots will be updated with the full line selected and the point highlighted on the map (c).

4 Scripts for creating the example data

This section contains the codes for extracting the data for the examples discussed in the main manuscript.

4.1 Historical maximum temperature

The script below presents the codes required to obtain the data `historical_tmax` used in the example of Section 5.2 *Australian historical maximum temperature*. The function `rnoaa::meteo_pull_monitors()` may take a while to query a large number of stations in the first time. A copy of the data is provided in the data folder of the paper repository at: https://github.com/huizezhang-sherry/paper-cubble.

```
library(tidyverse)
library(cubble)
all_stations <- rnoaa::ghcnd_stations() %>%
  filter(str_starts(id, "ASN")) %>% # Australian stations start wiht "ASN"
  filter(last_year >= 2020) %>%
  mutate(wmo_id = as.numeric(wmo_id), name = str_to_lower(name)) %>%
  select(-state, -gsn_flag) %>%
  select(id, longitude, latitude, elevation, name,
         wmo_id, element, first_year, last_year) %>%
  rename(long = longitude, lat = latitude, elev = elevation)

tmax_stations <- all_stations %>%
  filter(element == "TMAX", first_year < 1970, !is.na(wmo_id))

raw_tmax <- all_stations %>%
  rowwise() %>%
  mutate(ts = list(rnoaa::meteo_pull_monitors(
    monitors = id, var = "TMAX",
    date_min = glue::glue("{first_year}-01-01"),
    date_max = glue::glue("{last_year}-12-31")
    ) %>%
      select(-id)
    )
  )

historical_tmax <- raw_tmax %>%
  select(-element) %>%
  unnest(ts) %>%
  mutate(tmax = tmax/10) %>%
  filter(lubridate::year(date) %in% c(1971: 1975, 2016:2020)) %>%
  as_cubble(index = date, key = id, coords = c(long, lat))

save(historical_tmax, file = here::here("data/historical_tmax.rda"))
```

4.2 Australian 2016-2020 climate data

The data `climate_full`, used in the examples in Sections 5.3, 5.4, and 5.5 of the main paper and in **??** here, can be obtained in a similar fashion with a slight change on the selected variable and date parameter in `rnoaa::meteo_pull_monitors()` as shown below. The full script is provided below and a copy of the data is also available in the data folder of the paper GitHub repository linked above.

```
# all the Australian stations have all of the three PRCP, TMAX, and TMIN recorded
aus_stations <- all_stations %>%
  filter(element %in% c("PRCP", "TMAX", "TMIN")) %>%
  nest(element: last_year) %>%
  rowwise() %>%
  filter(nrow(data) == 3) %>%
  select(-data)
```

```r
aus_climate_raw <- aus_stations %>%
  rowwise() %>%
  mutate(ts = list(
    rnoaa::meteo_pull_monitors(
      monitors = id, var = c("PRCP", "TMAX", "TMIN"),
      date_min = "2016-01-01", date_max = "2020-12-31"
      ) %>%
      select(-id)
    )
  )

climate_full <- aus_climate_raw %>%
  unnest(ts) %>%
  mutate(tmax = tmax/10, tmin = tmin/10) %>%
  cubble::as_cubble(key = id, index = date, coords = c(long, lat))

save(climate_full, file = here::here("data/climate_full.rda"))
```