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
CSIRO Data61

Patricia Menéndez
Monash University

Abstract

The abstract of the article.

Keywords: spatio-temporal data, R.

1. Introduction

Many data structures have been proposed for spatial (**sf** by Pebesma (2018)) and temporal (**tsibble** by Wang, Cook, and Hyndman (2020)) data in the R community, while less has been done for spatio-temporal data. The lack of such tools could potentially because analysts usually treat the spatial and temporal dimension of the data separately, without realising the need to create a new data structure. While this approach follows the third tidy data principal (Wickham 2014) (*Each type of observational unit forms a table*), analysts always need to manually join results from different observational units or combining multiple tables into one for downstream analysis. This additional step doesn't add new operations into the data but can be error prone.

Currently, available spatio-temporal data structure in R includes: **spacetime** (Pebesma 2012), which proposes four space-time layouts: Full grid (STF), sparse grid(STS), irregular (STI), and trajectory (STT). The data structure it uses is based on **sp** (Pebesma and Bivand 2005) and **xts** (Ryan and Ulrich 2020), both of which has been replaced by more recent implementations.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 array with the dplyr's data cube structure **cubelyr** (Wickham 2020) as its backend. While these implementations either store spatial and temporal variables all in a single table, hence duplicate the spatial variables for each temporal unit; or split them into two separate tables that has the problem of manually joining, mentioned in the previously. None of these packages enjoy both the benefits of being able to separate manipulation in the two dimensions while also keep the data object as a whole. This create 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: the Ireland wind data from **gstat** is an classic example data that splits variables into spatial (`wind.loc`) and temporal (`wind`) dimension; Bureau of Meteorology (BoM) provides climate observations that are widely applied in agriculture and ecology study; air pollution data.

This paper describes the implementation of a new spatio-temporal data structure: **cubble**. **cubble** implements a relational 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 will be divided as follows: [complete when the paper structure is more solid]

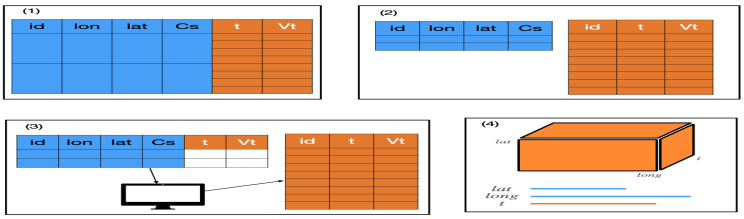

Figure 1: Illustration of incoming data formats for spatio-temporal data. (1) Data comes in as a single table; (2) Separate tables for spatial and temporal variables; (3) A single table with all the parameters used to query the database and a separate table for queried data; and (4) Cubical data in array or NetCDF format.

2. The cubble package

Spatio-temporal data usually come in various forms and Figure 1 shows four examples of this. No matter which form the data is in, these formats share some common components that characterise spatio-temporal data. A spatial identifier (`id` in the diagram) is the unique identifier of each site. The temporal identifier (`t` in the diagram) prescribes the time stamp each site is recorded. Coordinates, comprising of latitude and longitude (`lon` and `lat` in the diagram), locates each site on the map. These identifiers will be the building blocks for the data structure introduced below. Other variables in the data can be categorised into two groups: spatial variables that are invariant at each time stamp for every site, i.e. the name or code of the weather station and temporal variables that varies with time.

In a cubble, there are two forms: nested form and long form, and Figure 2 sketches the two forms along with the associated attributes. The decision on which form to use is output-oriented, meaning analysts need to first think about whether the output of a particular 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 site and this type of operation can be a pure manipulation of spatial variables, or a summary of temporal variables by site (i.e. the output of counting the number of raining day is only identified by sites and hence should be performed with the nested form). Underneath the nested form, a cubble is built from a row-wise dataframe (`rowwise_df`) where each site forms a separate group. This structure simplifies the calculation that involves temporal variables by avoiding the use of `map` syntax when working with list-column.

For those operations whose output involves both a spatial and temporal dimension, long form should be used. The long form is identified by both the spatial and temporal identifier and adopts a grouped dataframe (`grouped_df`) to forms each site as a group. Spatial variables are stored separately in a **tibble** 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.

2.1. Create a cubble in the nested form

To use functionalities from cubble, data analysts first need to create a cubble. `as_cubble` create a **cubble** by supplying the three key components: `key` as the spatial identifier; `index`

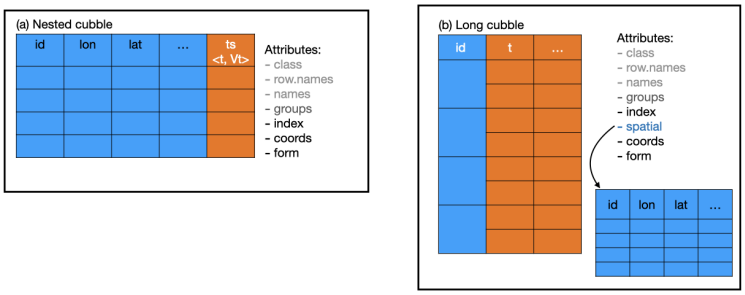

Figure 2: Illustration of nested and long cubble.

as the temporal identifier; and a vector of `coords` in the order of longitude first and then latitude. The naming of `key` and `index` follows the convention in the **tsibble** package. The cubble created by default is in the nested form. Below is an example of creating a cubble:

```
# cubble:   id [5]: nested form
# bbox:     [115.97, -32.94, 133.55, -12.42]- check gap on long and lat
# 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, bbox, and also the name of variable nested in the `ts` column. In this example, each site is identifier is `id` and the number in the bracket means there are 5 unique `id` in this dataset. The bbox in the second row gives the range of the coordinates. The temporal variables are all nested in the `ts` column, but it could be useful to know the name these variables. The third row in the cubble header shows these names and in this example this includes: precipitation, `prcp`, maximum temperature, `tmax`, and minimum temperature, `tmin`.

2.2. Stretch a nested cubble into the long form

The long cubble is suitable to manipulate the time dimension of the data. The function `stretch()` switches the nested cubble into the long cubble by first extracts all the spatial variables into a separate tibble and store in the `spatial` attribute and then unnests the `ts` column:

```
# cubble:  date, id [5]: long form
# bbox:    [115.97, -32.94, 133.55, -12.42]- check gap on long and lat
# 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 here that the third line in the header now shows the name of spatial variables rather than the temporal variables.

2.3. Tamp a long cubble back to the nested form

Manipulation on the spatial and temporal dimension can be an iterative process. Many times, analysts will need to go back and forth between the nested and long cubble. The `stretch()` function introduced in the previous section switches a nested cubble into a long cubble and function `tamp()` is its inverse function to switch a long cubble back to the nested cubble:

```
# cubble:   id [5]: nested form
# bbox:     [115.97, -32.94, 133.55, -12.42]- check gap on long and lat
# 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

As a final data output for modelling or visualisation, spatio-temporal data is usually expected to be in a single table. Function `migrate()` moves the spatial variables from the `spatial` attribute into the long cubble:

```
# cubble:  date, id [5]: long form
# bbox:    [115.97, -32.94, 133.55, -12.42]- check gap on long and lat
```

```
# 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
```

In this workflow described above, data objects come into cubble in the nested form, then various operations on the spatial and temporal dimension can go back and forth between the nested and long form, and finally, the data will come out of cubble in the long form for further modelling or visualisation.

Building from an underlying `tbl_df` structure, it is natural to implement methods available in `dplyr` to `cubble`. Supported methods in the `cubble` with `dplyr` generics includes:

basics	**mutate**, **filter**, **summarise**, **select**, **arrange**, **rename**, **left_join**
grouping	**group_by**, **ungroup**
slice family	**slice_head**, **slice_tail**, **slice_sample**, **slice_min** and **slice_max**

cubble is also compatible with **tsibble** in the sense that the original list-column can be a `tbl_ts` object. Duplicates and gaps should be first checked before structuring the data into a cubble. If the input data is a **tsibble** object, the long form cubble is also a **tsibble** where users can directly apply time series operations.

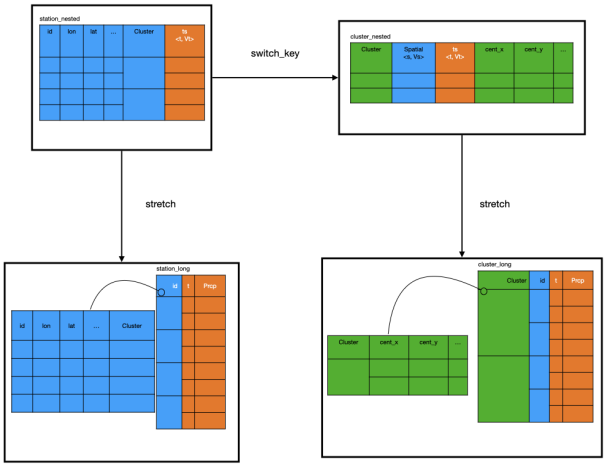

Figure 3: Relationship between the station and cluster level cubble in both long and nested form.

3. Advanced features/ considerations

3.1. Hierarchical structure

Spatial locations can have further grouping structure either in nature (i.e. countries are nested in continents), or calculated by some clustering algorithms. Rather than investigate the read from individual locations, group character summarised from multiple locations can give a clearer picture of the local dynamic. In cubble, `switch_key()` can be used to create a new level of grouping of spatial location by specifying a clustering variable. Figure 3 illustrates the relationship of cubbles at site 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`. Following the same principal of `stretch`, the long form cluster level cubble expands the time series variables with both indices: `cluster` and `id`. All the cluster level variables are stored separately for recovering the nested from from the long form.

3.2. Data fusion and matching

Temporal matching checks how spatially matched pairs align temporally. We use the following chart to illustrate how the temporal matching works:

For each spatially matched pair, say `A` and `a`, we first find the largest `n` points in each series, colored in brown points here. Here we use the largest three but you can tune this number by

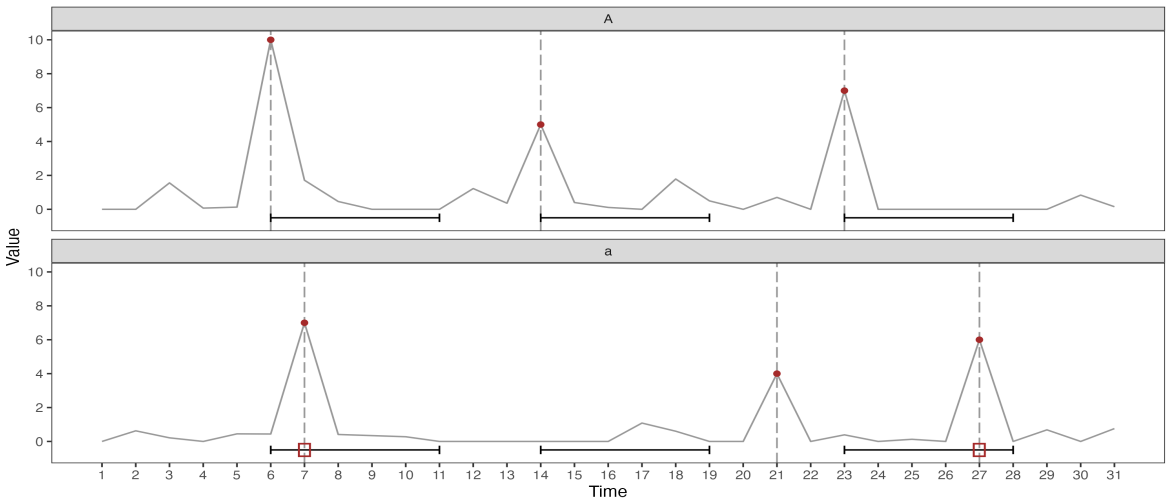

Figure 4: sdfasdf

`temporal_n_highest`. Then we construct the interval of the largest points from one series and see how many points, from the other series, fall into the intervals. The series used to construct the interval is controlled by `temporal_independent` and the window size by `temporal_window` with a default of 5.

In this illustration, we construct the interval based on series `A` and two of the three peaks from `a` falls into this interval at Time 7 and 27.

There's another mandatory argument that hasn't been introduced above: `temporal_var_to_match`. This argument controls the variable to match and it needs to appear in both the `major` and `minor` set. In the water level matching example, we match the variable `Water_course_level` from `river` to `prcp` from `climate`, hence need to manually rename one of them to match the other, here we rename `Water_course_level` to `prcp` in `river`:

Figure 4

3.3. Interactive graphics

The cubble structure 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 with the two forms in a cubble, where a time series plot is created with the long cubble and a map is created with the nested cubble. When a user action is captured from the map, the site will be activated in the nested cubble. Then, the nested cubble will communicate to the long cubble to activate all the observations with the same `id`. The long cubble will then highlight the activated series in the time series plot.

The linking is also available from the time series plot to the map. The selection on the time series is through selecting the point 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`, either in the long and nested cubble, are then activated. This includes other points in the same time series in the long cubble and the corresponding observation of site in the nested cubble. These

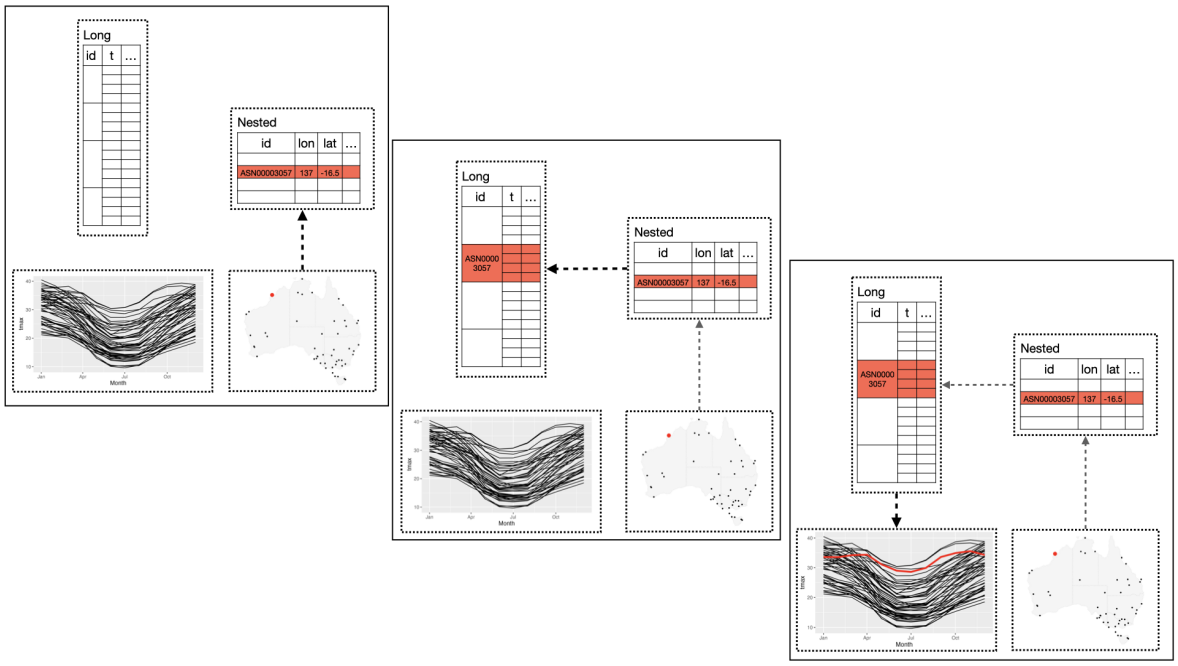

Figure 5: demon interactivity

activated observations will then being reflected in the updated plots and Diagram 12 in the Appendix illustrates this process.

3.4. Glyph map

Glyph map (Wickham, Hofmann, Wickham, and Cook 2012) plots the time series as single glyph on the map. In R, `GGally` implements the glyph map through the `glyphs()` function, which outputs a data frame with calculated position (`gx`, `gy`, `gid`) of each point on the time series given the major and minor xy variable 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:

```
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()
```

While this calculation can be seen as part of the `setup_data()` in a `ggproto`. A re-implementation of the glyph map as `GeomGlyph` has been made in `cubble` to create the glyph map with `geom_glyph`:

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

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 also be controlled by the parameter `global_rescale`, which default to `TRUE`. 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 and the dataset `weatherdata::historical_tmax` extracts the historical maximum temperature recorded for 236 Australian stations. The data `historical_tmax` is already presented as a cubble, with `id` as the key, `date` as the index, and `c("longitude", "latitude")` as the coordinates. Other variables include `elevation`, `name`, `wmo_id`, `first_year`, and `last_year` in the nested form and `tmax` in the long form. 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 on 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) denote the state a station belongs to. New South Wales stations start from 46 to 75 and Victoria stations then follow from 76 to 90. Extracting Victoria and New South Wales stations is a filter operation in the spatial dimension and hence is operated in the nested form:

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

The five year window is chosen to remove the effect of a particular year and the historical period of 1971 - 1975 is used since all the stations have records from 1970. This following chunk filters on records in the two study periods and summarises the maximum temperature into monthly measure. `stretch()` is used to convert the nested form cubble into the long form for these operations:

```
R> tmax <- tmax %>%
+   stretch() %>%
+   filter(lubridate::year(date) %in% c(1971:1975, 2016:2020)) %>%
+   mutate(month = lubridate::month(date),
+          group = as.factor(ifelse(lubridate::year(date) > 2015,
+                                   "2016 ~ 2020", "1971 ~ 1975"))) %>%
+   group_by(month, group) %>%
+   summarise(tmax = mean(tmax, na.rm = TRUE))
```

A data quality issue with GHCN data is that while the first and last year of each series is provided, years missing in this period is not reported. There are a few stations which don't have records during 1971 - 1975 and these stations are filtered out by examining whether the summarised `tmax` has a total of 24 months. This is again a station-wise operation and is operated in the nested form, which is switched to from the long form by `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)
```

Figure 6 shows the glyph map made with the data after the wangling above. One issue with this map is that the similar pattern shown from a few nearby stations around Sydney (151E, 34S) and New castle (152E, 33S) can be distracting. Aggregation is a useful technique to observe the general pattern of a collection of series and will be the topic of the next example.

4.2. Australia precipitation pattern in 2020

With recent climate, there are 639 stations in Australia that records daily maximum and minimum temperature and precipitation. While it would be too much series to show on a glyph map, stations close to each other should have similar precipitation pattern. Aggregation can be helpful here to cluster series into groups and visualise them with glyph map.

A simple kmean algorithm is used based on the distance matrix to cluster the stations into 20 groups. This creates `station_nested` as a station level nested cubble with a cluster column indicating which group each station belongs to. More complex algorithms can be used for more complex problem, as long as a mapping from each station id to the cluster id can be constructed.

```
# cubble:   id [639]: nested form
# bbox:     [113.53, -43.66, 153.64, -10.05]
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id            lat  long  elev name             wmo_id ts                cluster
  <chr>       <dbl> <dbl> <dbl> <chr>             <dbl> <list>             <int>
1 ASN00001006 -15.5  128.   3.8 wyndham aero      95214 <tibble [1,827 ~       3
2 ASN00001007 -13.8  126.   6   troughton island  94102 <tibble [1,827 ~       3
3 ASN00001018 -16.4  126. 546   mount elizabeth   94211 <tibble [1,460 ~      16
4 ASN00001019 -14.3  127.  23   kalumburu         94100 <tibble [1,827 ~       3
5 ASN00001020 -14.1  126.  51   truscott          95101 <tibble [1,827 ~       3
# ... with 634 more rows
```

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, it is useful to compute the centroid of each cluster as the location of group level glyph map. `get_centroid()` is a shortcut for doing so or you can manually find the convex hull, its centroid, and extract the longitude and latitude:

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

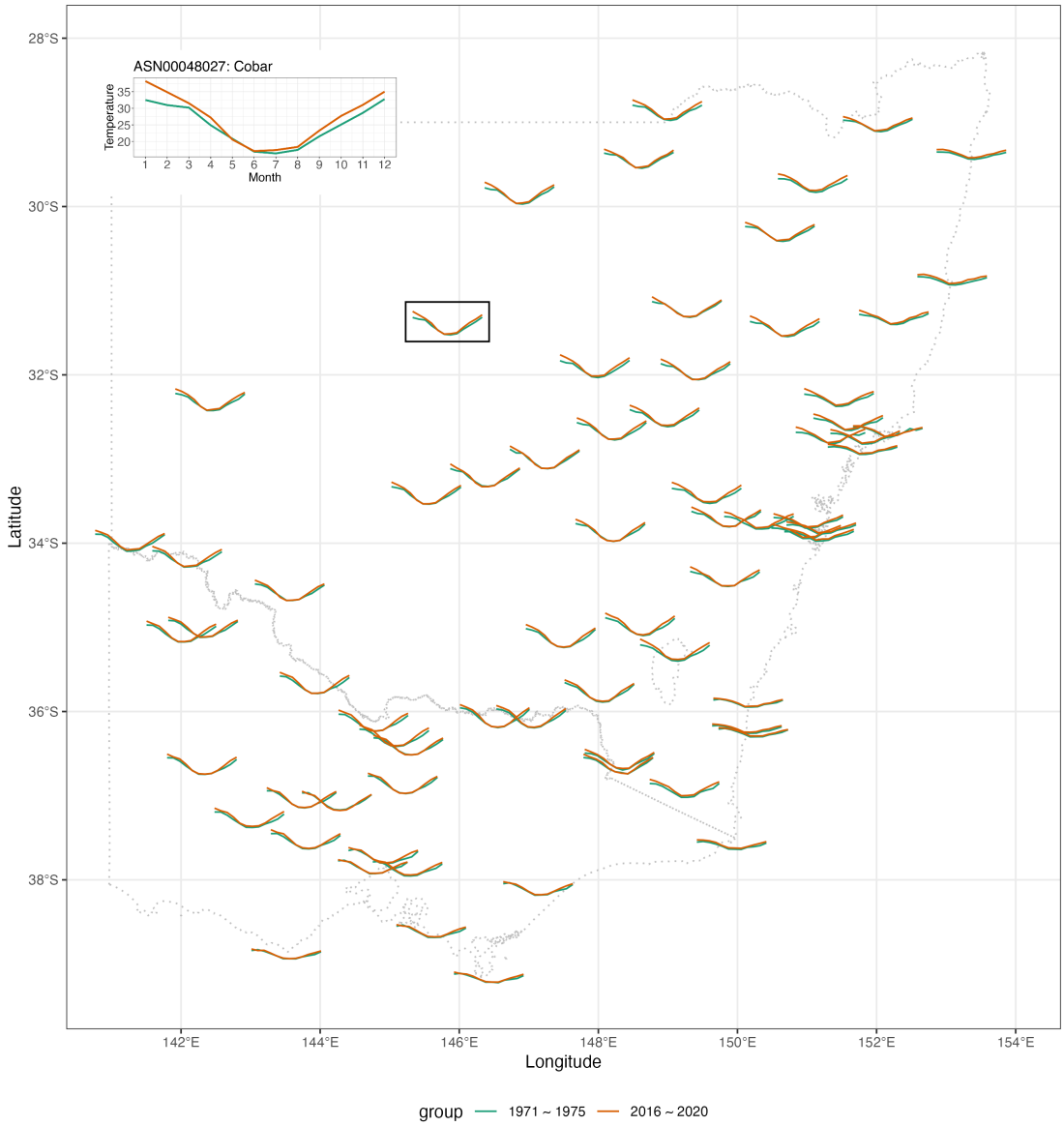

Figure 6: Glyph map of the mean maximum temperature by month for Victoria and New South Wales weather stations. On the top left corner is a detailed legend for station Cobar highlighted in the black box. Compared to 1971 - 1975, the period 2016 - 2020 sees an increase of mean maximum temperature in spring and summer in Australia (end and beginning of the year). A larger increase during the first three months of the year is observed at stations with a lower latitude.

Figure 7: Aggregated precipitation in Australia.

Long form cubble at both levels can be access through stretching the nested form. Here weekly averages of the precipitation is calculated:

```
R> station_long <- station_nested %>%
+   stretch(ts) %>%
+   mutate(wk = lubridate::week(date)) %>%
+   group_by(wk) %>%
+   summarise(prcp = sum(prcp, na.rm = TRUE))
R>
R> cluster_long <- cluster_nested %>%
+   stretch(ts) %>%
+   mutate(wk = lubridate::week(date)) %>%
+   group_by(wk) %>%
+   summarise(prcp = mean(prcp, na.rm = TRUE)) %>%
+   migrate(cent_long, cent_lat)
```

With the four cubbles: `station_nested`, `station_long`, `cluster_nested`, and `cluster_long`, all the information at both levels are accessible. This allows us to make glyph map on the cluster level, looking at station composition of each cluster, and plot the individual series to check for consistency within groups. Figure 7 composes of the three plots together to show the precipitation in Australia.

4.3. River level data in Victria water gauges

Water data is collected by the Bureau of Meteorology. This includes ... water gauges, variables collected. In particular, water level will interactive with the climate variable precipitation as rains fall into the river will raise the water level. Figure 8 shows the available weather station and water gauges in Victoria, Melbourne.

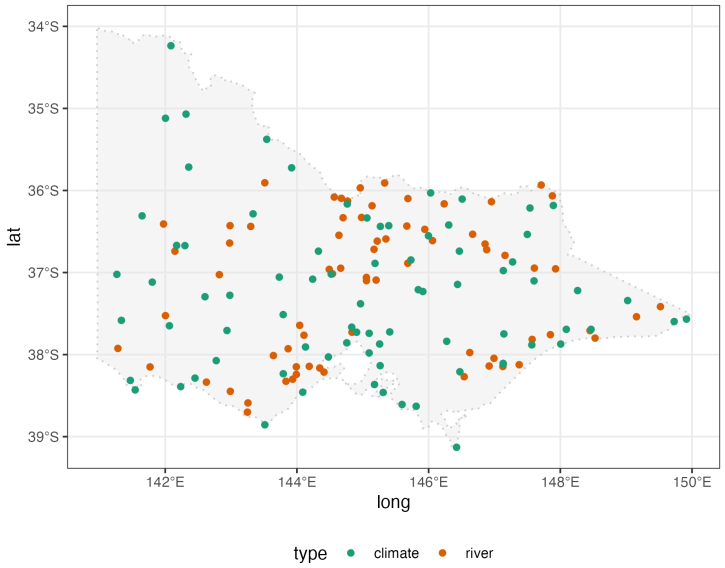

Figure 8: asdfasd

```
R> climate <- weatherdata::climate_full %>%
+   filter(between(stringr::str_sub(id, 7, 8), 76, 90)) %>%
+   stretch() %>%
+   filter(lubridate::year(date) == 2020) %>%
+   tamp() %>%
+   mutate(type = "climate")
R>
R> river <- weatherdata::water %>%
+   stretch() %>%
+   select(date, Water_course_level) %>%
+   rename(prcp = Water_course_level) %>%
+   tamp() %>%
+   mutate(type = "river")
```

From the map, a few water gauges and weather stations are close to each other and we may be able to observe a spontaneous increase in the precipitation and water level from the measures, potential with a little delay in the river gauges. `match_sites()` matches one source of data with another source in both spatial and temporal dimension in a cubble.

```
R> res <- match_sites(river, climate,
+                     temporal_var_to_match = prcp,
+                     temporal_independent = climate,
+                     temporal_n_highest = 30,
+                     temporal_min_match = 15)
```

The output from temporal matching is also a cubble, with additional column `dist` and `group` produced from spatial matching and `n_match` from the number of peak matched temporally.

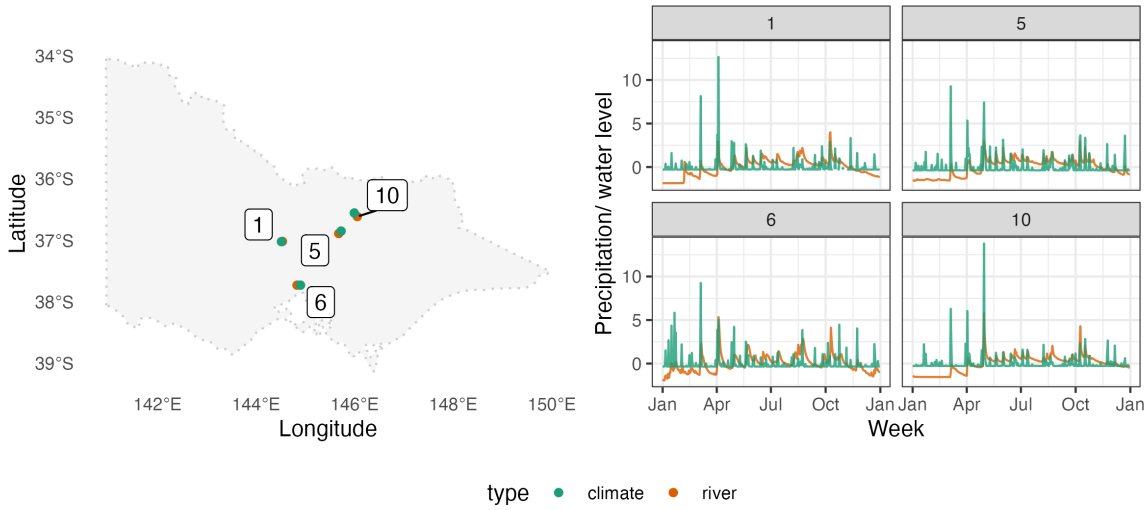

Figure 9: asdfasd

```
# cubble:   id [8]: nested form
# bbox:     [144.52, -37.73, 146.06, -36.55]
# temporal: date [date], prcp [dbl]
  id          name                lat  long ts         type   dist group n_match
  <chr>       <chr>             <dbl> <dbl> <list>     <chr> <dbl> <int>   <int>
1 405234      SEVEN CREEKS @ D~ -36.9  146. <tibble [~ river  6.15     5      21
2 ASN00082042 strathbogie       -36.8  146. <tibble [~ clim~  6.15     5      21
3 404207      HOLLAND CREEK @ ~ -36.6  146. <tibble [~ river  8.54    10      21
4 ASN00082170 benalla airport   -36.6  146. <tibble [~ clim~  8.54    10      21
5 230200      MARIBYRNONG RIVE~ -37.7  145. <tibble [~ river  6.17     6      19
# ... with 3 more rows
```

Then plots can be made to inspect the matched pairs on the map or to view the co-movement of the two series:

With the chosen parameter, four pairs of stations stands out, 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

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 their corresponding time series and 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`.

Starting with the original data, some pre-processing may be required to summarise the data before the visualisation. This example explores the variation of monthly temperature range across 639 weather stations in Australia and with `weatherdata::climate_full`, daily temperature range is first calculated as the difference between `tmax` and `tmin`. The three daily variables are then averaged into month over 2016 - 2020 in the long form. Variance of the temperature difference is then calculated for each station in the nested form. Then, 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. And the cross-linking 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 10, the first row shows the initial view of the interactive graphic. Overall, most regions in Australia have low variance of temperature range across different months 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 dark colour and this links to the ribbon on the right where there is a larger temperature range is presented in Australian winter (June to August). The third row selects the Grampian station in Victoria and the linked ribbon shows an opposite wider range in Australian summer period (December to February). The last row selects point with the lowest minimum temperature in August in

the ribbon plot and surprisingly, this links to the Thredbo Airport in the Victoria and New South Wales border, rather than somewhere in the Tasmania island!

With leaflet popup, 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 11 shows the same content as Figure 10 but made with leaflet and popups.

5. Conclusion

6. Appendix

References

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.

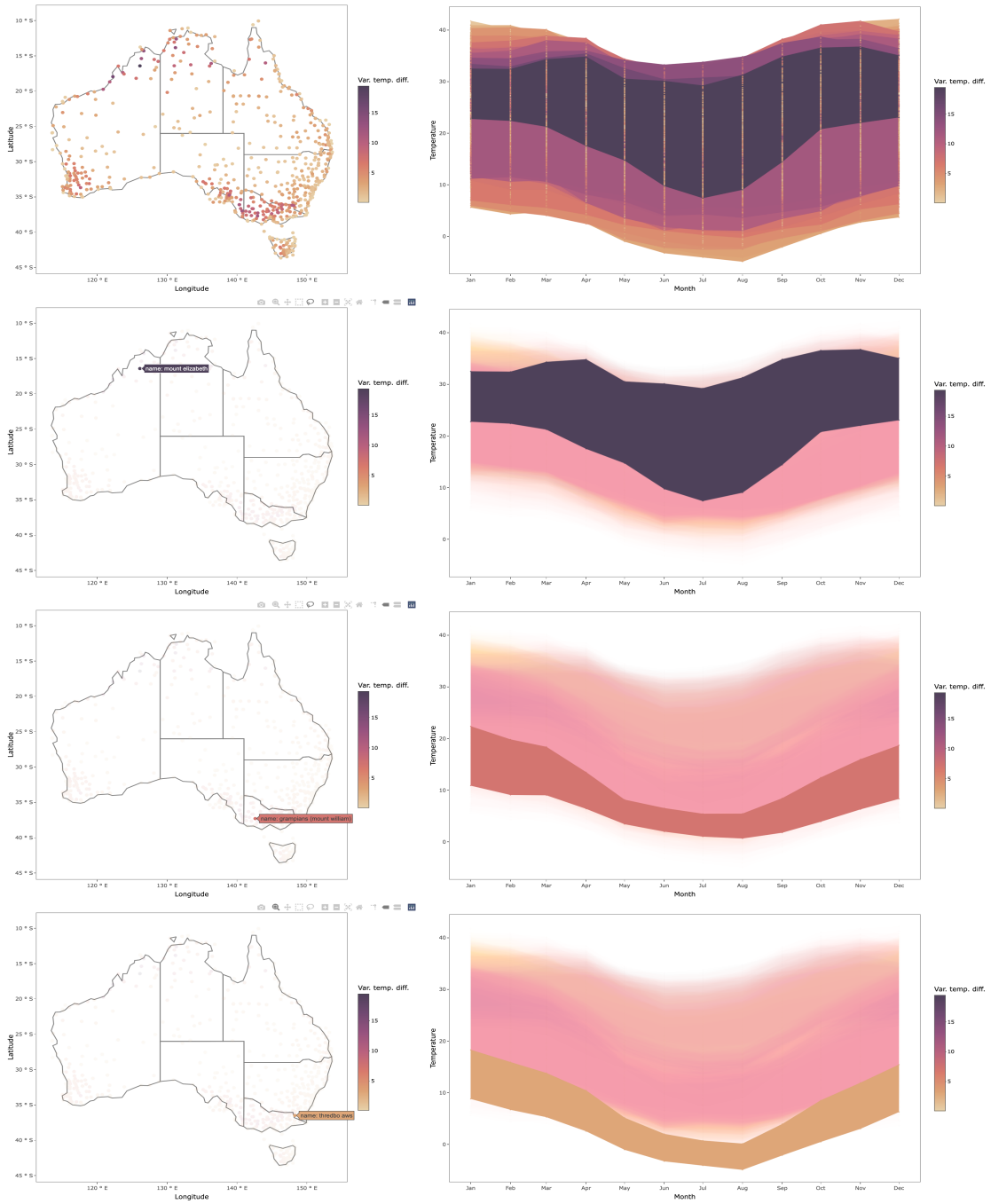

Figure 10: Exploring temperature variation using linkning of a map and seasonal display. Each row is a screen dump of the process. The top row shows all locations and all temperature profiels. Selecting a location with high variance on the map produces the plot in the second row. The maximum nad minimum temperature is shown using a ribbon.

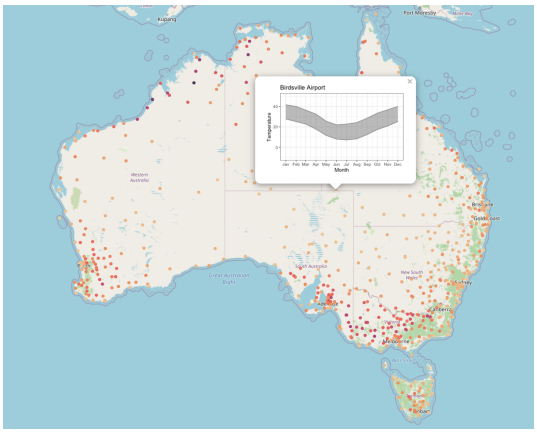

Figure 11: Screenshots of variance of monthly temperature range from 2016 to 2020 in Australia. Upon clicking a single station on the leaftlet up, the temperature band will be shown as a subplot in the popup box.

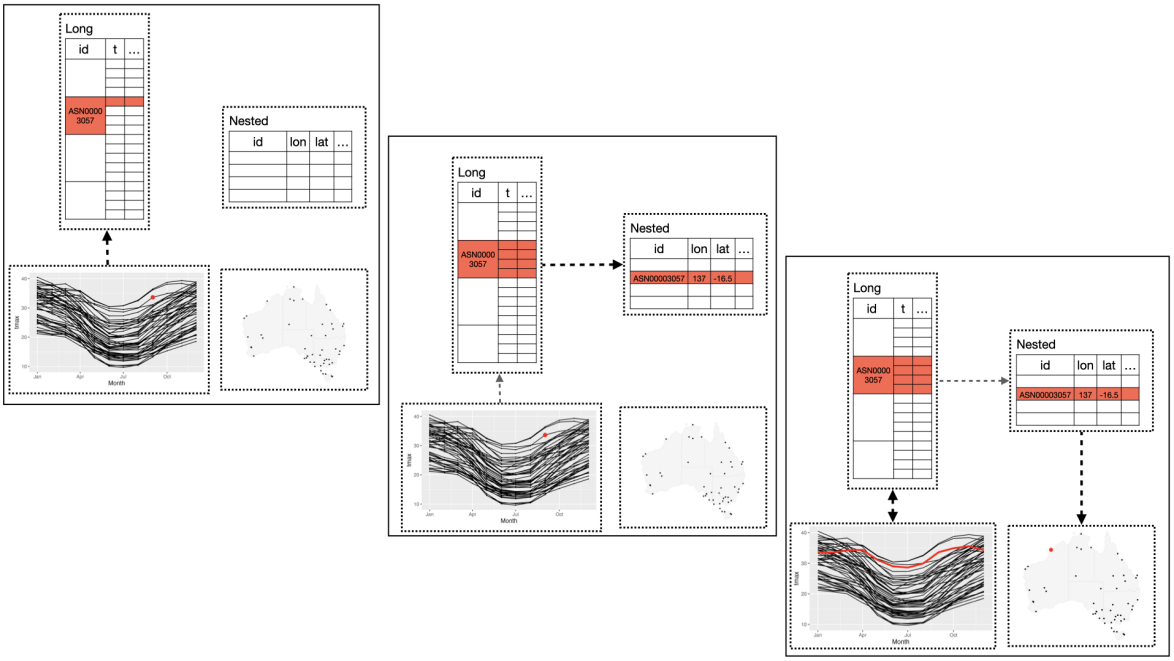

Figure 12: demon interactivity

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.

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.

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

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

Wickham H (2020). *cubelyr: A Data Cube 'dplyr' Backend*. R package version 1.0.1, URL https://CRAN.R-project.org/package=cubelyr.

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, Hofmann H, Cheng X (2014). "Reactive programming for interactive graphics." *Statistical Science*, pp. 201–213. URL https://doi.org/10.1214/14-STS477.

Affiliation:

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