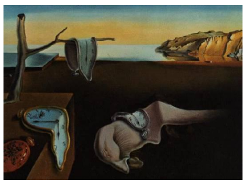


Journal of Statistical Software

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structure it uses is based on **sp** (Pebesma and Bivand 2005) and **xts** (Ryan and Ulrich 2020), both of which has been replaced by more recent implementations.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]

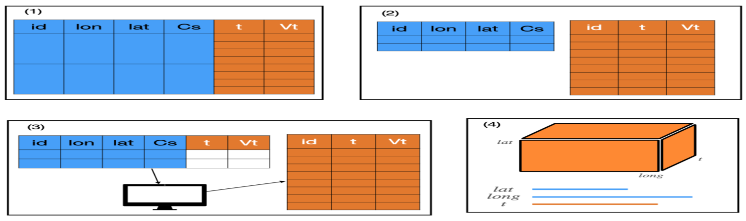


Figure 1: Illustration of incoming data formats for spatio-temporal data. (1) Data comes in as a single table; (2) Separate tables for spatial and temporal variables; (3) A single table with all the parameters used to query the database and a separate table for queried data; and (4) Cubical data in array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

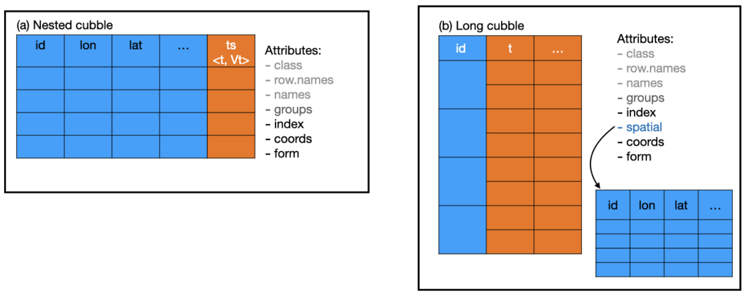


Figure 2: Illustration of nested and long cubble.

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.

3. Advanced features/ considerations

3.1. Hierarchical structure

Imposing a clustering structure can be thought of as building a hierarchical structure where stations are nested within clusters. This can be useful when there's intrinsic nesting structure in the data (i.e. country nested in the continent, county nested in the state) or there's some clustering. When we have access to both level, `switch_key()` is the function to re-structure the data as one cluster per row. Temporal observations from different stations while within the same cluster are bound in the nested column `ts`.

One thing we hope to do with the cluster is to find the coordinates of the centroid. These are variables variant to the station but invariant to the cluster and it would be nice to have a function that structure each cluster as a row. `switch_key()` is the function that does this: it lets you to specify a new key, say `cluster` and nests all spatial variables variant to `cluster` into a column. Spatial variables are all nested inside a new column `.val`. Temporal observations from different stations while within the same cluster are bound in the nested column `ts`.

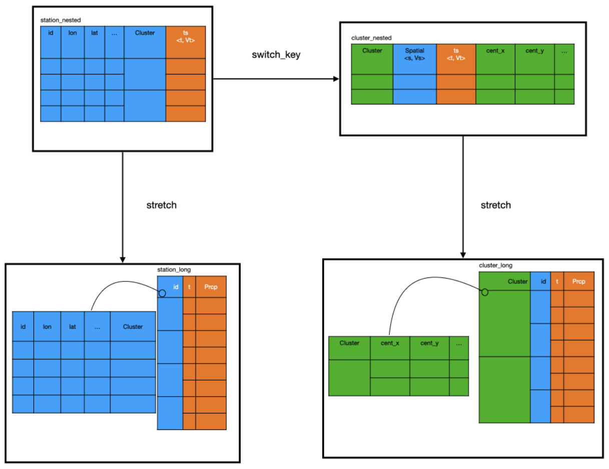


Figure 3: Hierarchical structure

This structure makes it easy to compute cluster level variable, for example, the convex hull and the centroid coordinate of each cluster. These can be amended into the nested form with function `get_centroid`.

After we have got `cluster_nested`, spatial and temporal data at both levels can be easily obtained. Figure 3 illustrate the relationship between the long and nested form cubble at both site and cluster level. More description on this. Start with the original `station_nested`, `stretch()` expands the `ts` column with each station (`id`) forming a group and attach variables invariant to `id` as an attribute. `switch_key()` changes the `key` from `id` to `cluster` and nests all the spatial variables that variant to `cluster`. `stretch()` `cluster_nested` will store variables that are invariant to `cluster` as a tibble in the attribute.

this fit into the remaining data pipeline.

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 `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.

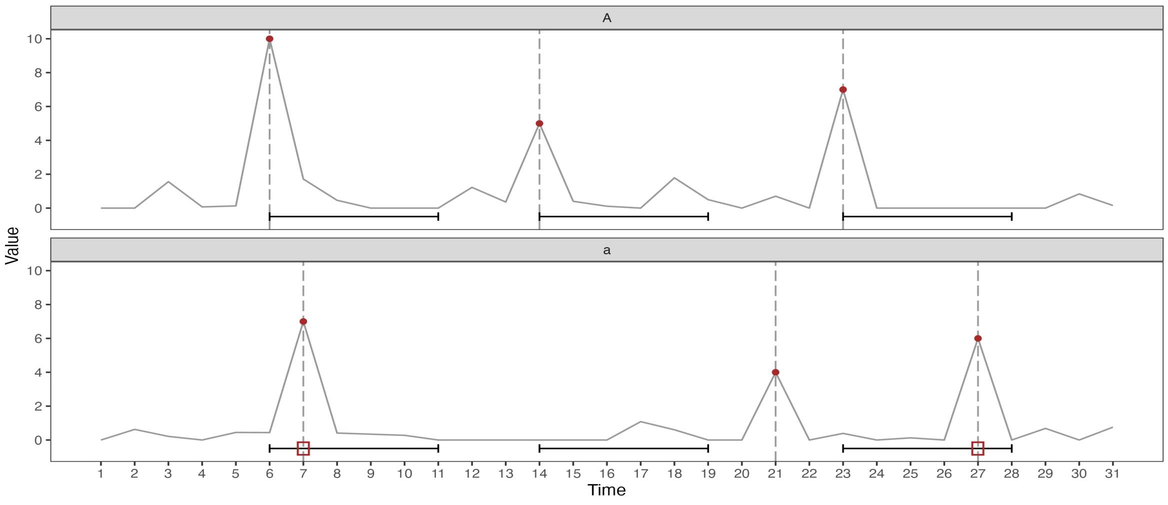


Figure 4: sdfasdf

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 activated observations will then being reflected in the updated plots and Diagram 11 in the Appendix illustrates this process.

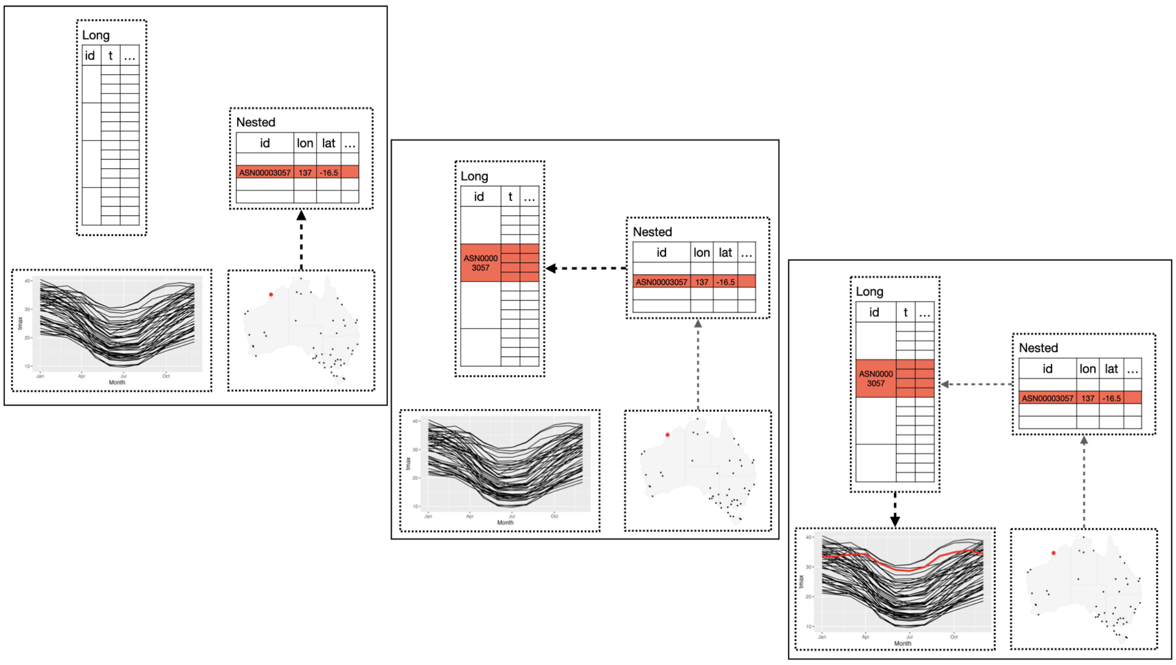


Figure 5: demon interactivity

4. Examples

4.1. Australia precipitation pattern in 2020

Basic manipulation

Bureau of Meteorology (BoM) collects climate variables. This variables can be openly queried via `rnoaa`. `weatherdata` provides a readily available copy of several data. `weatherdata::historical_tmax` provides historical maximum temperature recorded for Australian stations with the earliest dating back to 1859 and this can be used to understand the change of maximum temperature from a past period to nowadays.

Figure 6.

```
R> tmax <- weatherdata::historical_tmax %>%
+   # filter out VIC and NSW stations
+   filter(between(stringr::str_sub(id, 7, 8), 46, 90)) %>%
+
+   # filter out observations in between 1971-1975 and 2016 - 2020
+   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 into monthly mean for both periods
+   summarise(tmax = mean(tmax, na.rm = TRUE)) %>%
+
+   # filter out stations that have complete records in both periods
+   tamp() %>%
+   filter(nrow(ts) == 24) %>%
+
+   # migrate longitude and latitude to the long form for making the glyph map
+   stretch() %>%
+   migrate(latitude, longitude)
R>
R> gly <- glyphs(tmax, "longitude", "month", "latitude", "tmax",
+                height = 0.6, width = 1)
R>
R> nsw_vic <- ozmaps::abs_ste %>% filter(NAME %in% c("Victoria", "New South Wales"))
R>
R> ggplot() +
+   geom_sf(data = nsw_vic, fill = "transparent") +
+   geom_path(data = gly,
+             aes(gx, gy, group = interaction(id, group), color = group)) +
+   scale_color_brewer(palette = "Dark2") +
+   theme_bw() +
+   coord_sf(xlim = c(140, 155)) +
+   theme(legend.position = "bottom")
```

Aggregation

In figure 6, there are a few stations around Sydney (151E, 34S) and New castle (152E, 33S). Aggregating them into averages or other statistics would give a better picture. This can be done with cubble through a hierarchical structure, where the cluster indicator is one level above the individual station.

In Figure 7

The `weatherdata::climate_full` data has daily climate data of 639 Australia stations from 2016 to 2020.

calcluate distance matrix As an example to illustrate here, a kmean clustering algorithm based on the distance matrix is used and the number of centres is set to 20. More complex algorithms can also be used for more complex problem, as long as a mapping from each station id to the cluster id can be constructed.

4.2. River level data in Victria water gauges

The water level data comes from Bureau of Meteorology and has a copy in `weatherdata`. Here we extract the water course level and add a column annotate this data of type `river`. For the rainfall data, we will still use the `weatherdata::climate_full`, filtering for Victorian stations in 2020 should be pretty familiar by now. Again, we first look at where these stations are on the map first:

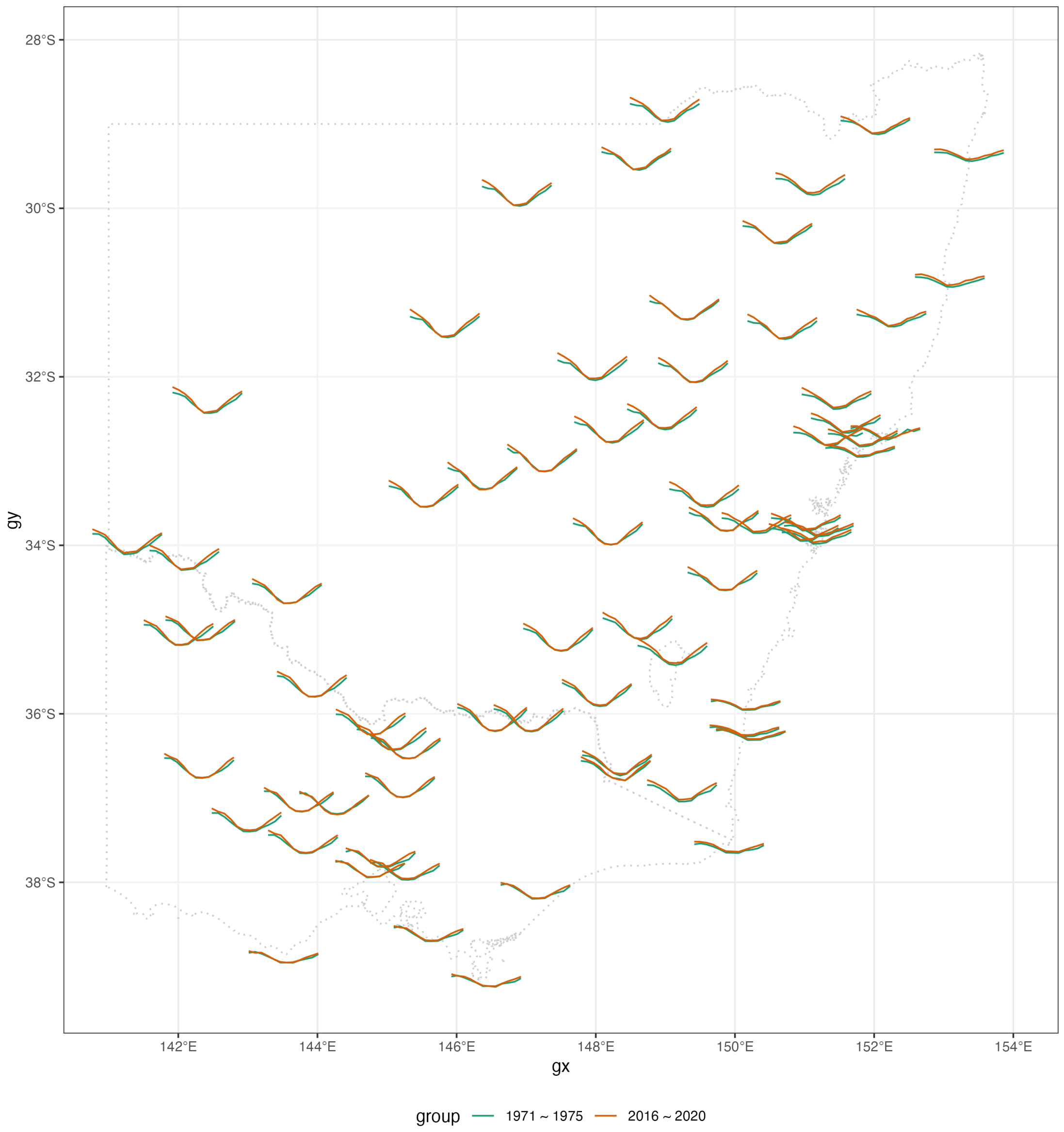


Figure 6: asdfasd

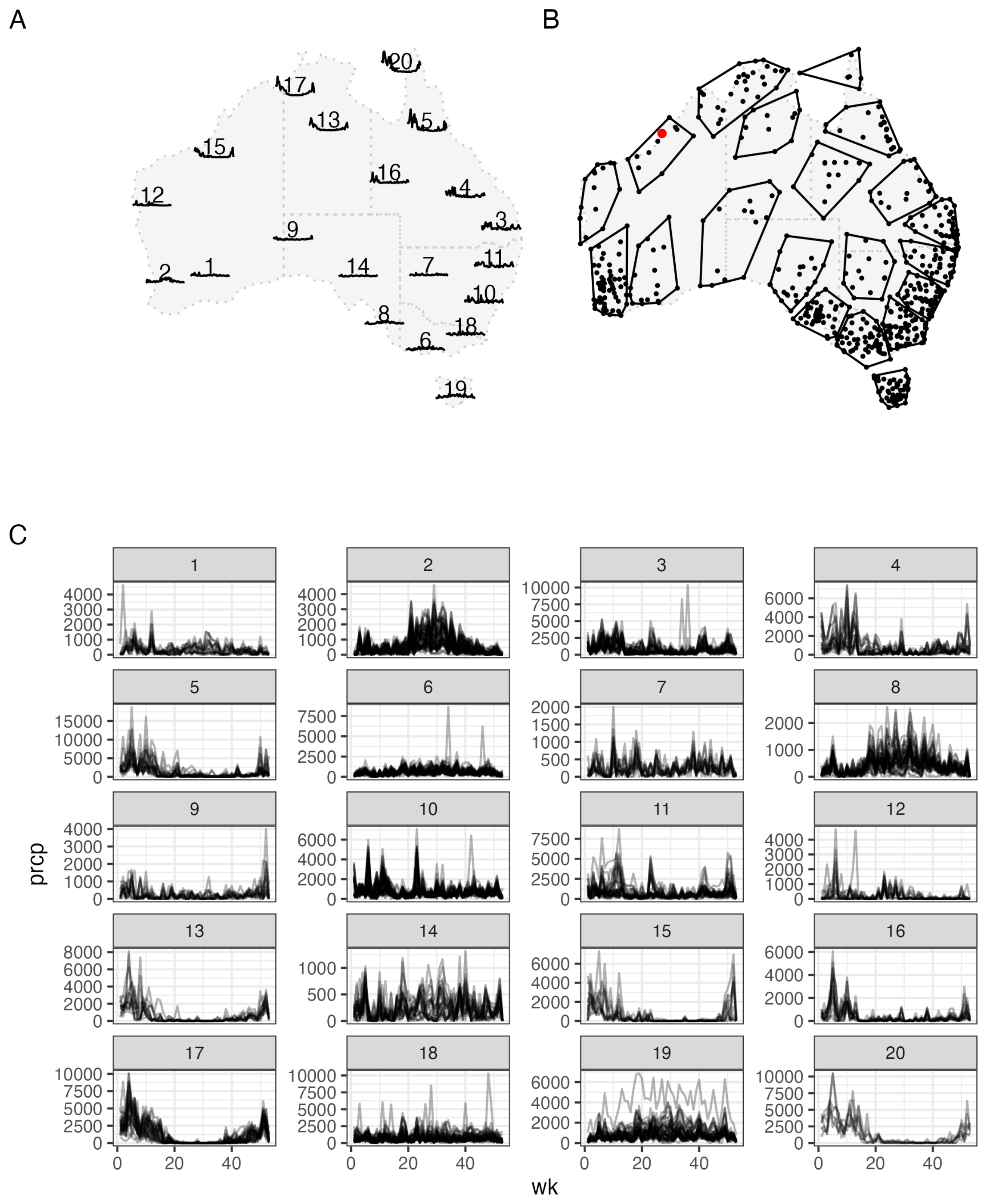
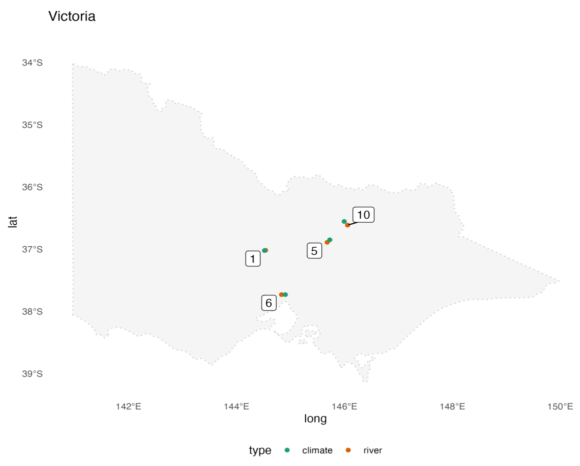


Figure 7: sdfasdf



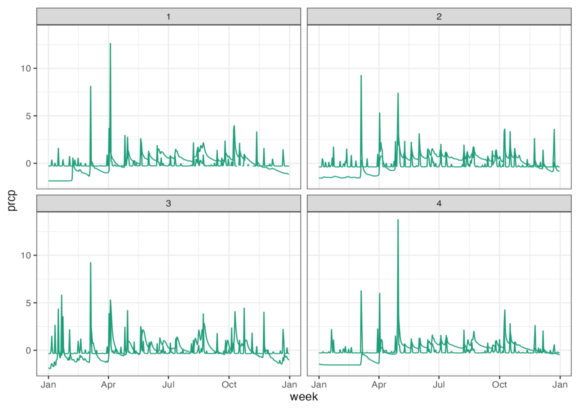


Figure 8: asdfasd

Now we use `match_sites()` to first pair the weather stations with the river gauges spatially and then apply the temporal matching on `prcp`. We will construct the interval based on peaks in `climate` since we would expect a lag effect for precipitation to flow into the river and cause a raise in river level, hence `temporal_independent = climate`. We select the 30 highest peak from the series to construct the match by setting `temporal_n_highest = 30`. This is a tuning parameter and you can start with 10% of the points of one series (here we have daily data for a year, 10% is roughly 30 points). `temporal_min_match` filters out pairs don't have enough match and to return all the pairs, set `temporal_min_match` to `0`.“

The output from temporal matching is also a cubble, with additional column `.dist` and `.group` inherent from spatial matching and `n_match` for the number of matched temporal peaks. Then you can use this output to plot the location of match or to look at the series:

4.3. ERA5: climate reanalysis data

4.4. Interative graphic with cubble

With a larger dataset, interactive graphics can be helpful to explore these data from multiple perspective. Two examples are presented here to show how interactivity can be used to explore the climate data across Australia.

Figure 9 Figure 10

5. Conclusion

6. Appendix

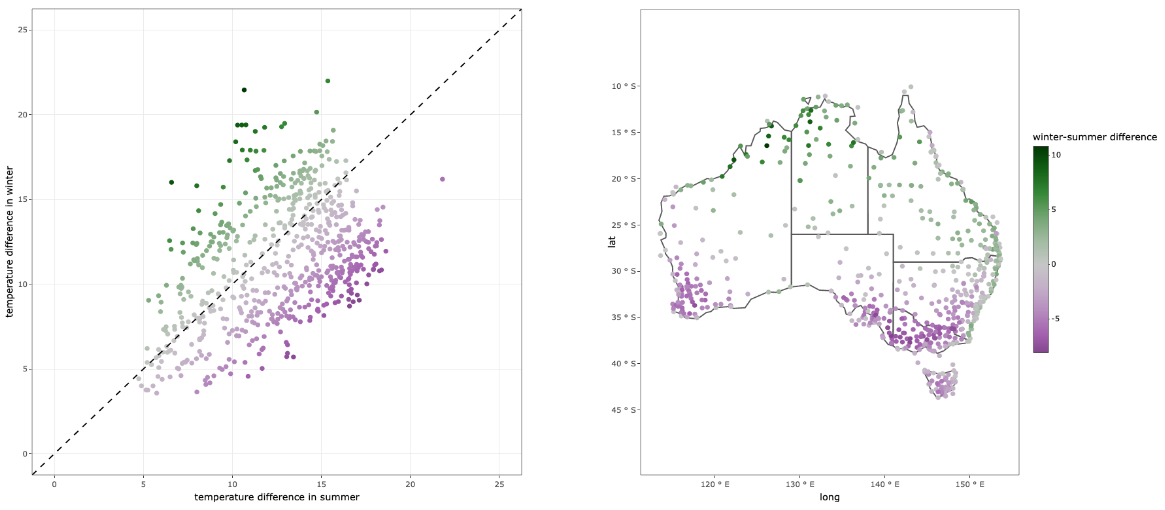


Figure 9: sdfasd

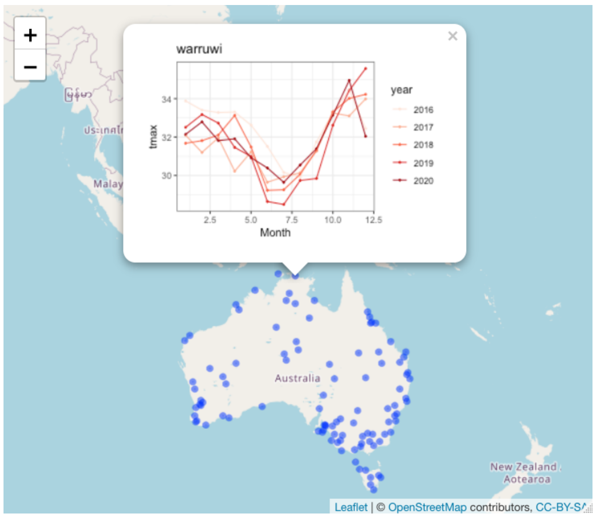


Figure 10: sdfasd

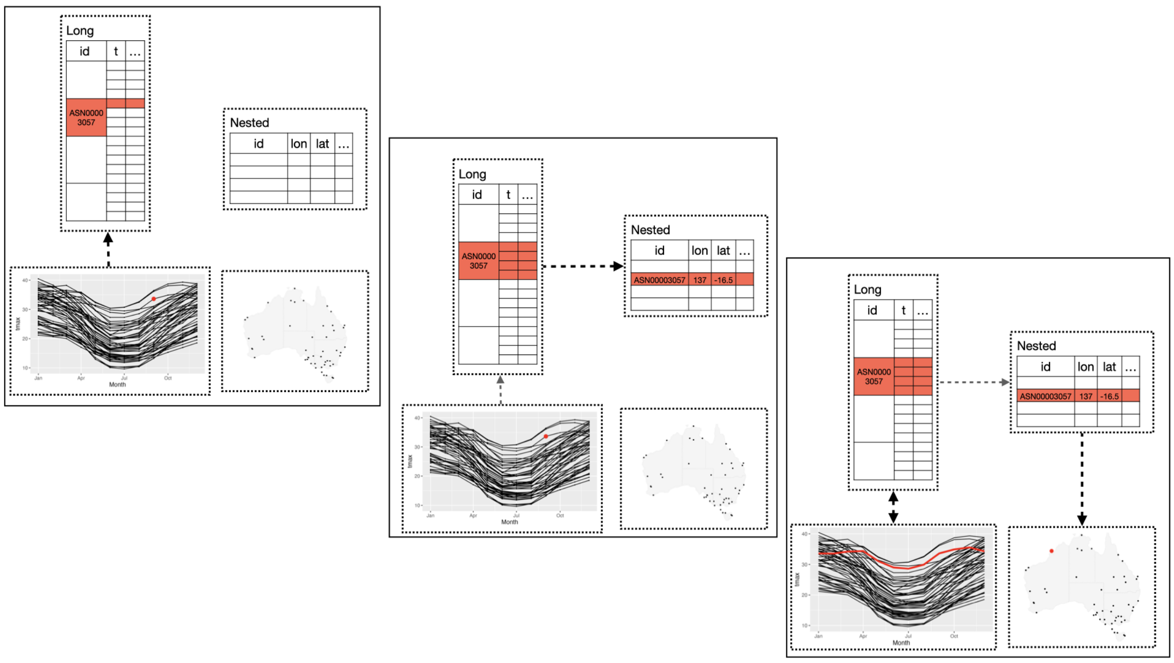


Figure 11: demon interactivity

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`.

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`.

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