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

Motivation

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 separately, without realising the need to create a new data structure for spatio-temporal data. While this approach fulfills the third tidy data principal to store each type of observational unit in its own table (Wickham 2014), analysts will always need to manually join the station summary of temporal results into the spatial table or combine the two tables into one for downstream analysis. This additional step doesn't add new operations but can be error prone.

Existing packages

Currently, available spatio-temporal data structure in R includes: `spacetime`(Pebesma 2012), which has proposed four space-time layouts: Full grid (STF), sparse grid(STS), irregular (STI), and trajectory (STT). The data structure it uses are based on `sp` (Pebesma and Bivand 2005) for spatial and `xts`(Ryan and Ulrich 2020) for temporal, both of which has been replaced by more recent implementations. `spatstat` (Baddeley and Turner 2005) implements a `ppp` class for point pattern data. More recent package `stars` (Pebesma 2021) uses a spatio-temporal array to store the data and the array structure has its influence from `cubelyr` (Wickham 2020), a dplyr data cube backend. None of these packages provide [. . . .] and this motivates a new spatio-temporal data structure that can smoothly switch between the spatial and temporal table for manipulation of each dimension, while also take care of combining the two dimension together. The requirement for such a tool is important given the ubiquity of spatio-temporal vector data in the wild. Example of this includes [house price of a city or county], climate station data from Beureau of Meteorology (BoM), the Ireland wind data from `gstat`, air pollution data.

Our new data structure for spatio-temporal data

In this paper, a new spatio-temporal data structure, `cubble`, is introduced. `cubble` implements a relational data structure that splits the data into its spatial and temporal component. With this structure, users can separately manipulate the spaital or temporal component of the data while keep the two dimension linked. The software is available from the Comprehensive R Archive Network (CRAN) at [CRAN link].

Section division

The rest of the paper will be divided as follows: Section 2 reviews the existing data structure for spatio, temporal, and spatio-temporal data. Section 3 presents a new data structure for spatio-temporal data: cubble. Then the paper introduces the workflow of data manipulation and visualisation with the cubble structure in Section 4. Section 5 gives some examples on how common spatial and temporal manipulations are performed with cubble and how static and interactive visualisation help to understand climate and [. . .] data.

Criteria of a data structure we want

With recent development in the R community, `sf` (Pebesma 2018) and `tsibble`(Wang *et al.* 2020) have replaced `sp` and `xts` to be the convention structure for spatial and temporal data. One reason for their popularity is its integration with tidyverse ecosystem, making them intuitive and easy to adopt. For spatio-temporal data, we hope to build a data structure has the following features:

1) A data structure that handles spatial and temporal dimension in a relational structure. This derives from the 3rd tidy data principal.
2) An intuitive and easy to use interface that fits into the tidyverse ecosystem, and
3) Compatibility with the latest spatial and temporal data structure. This would give users the flexibility to use the work in existing packages.

- In this article, we consider spatio-temporal vector data, which are recorded in a fixed interval.

point to common feature: spatial level variables + time level variables relates to Table 13 in Tidy data paper
Usually, this type of spatio-temporal data does not necessarily come in as a single table. Tidy data principle (Wickham 2014) prescribes each type of observational unit to form a table. This would suggest two tables to store the data, one for spatial-level and one for temporal-level. The Table 13 in Wickham (2014) presents a structure like this and the author argues the lack of tools to work with relational data like such.

My proposal
Recent software development in R has proposed several relational data structure: `tidygraph`(Pedersen 2020) for graph manipulation, `dm` (Schieferdecker, Müller, and Bergant 2021) for relational data model, while spatio-temporal data could benefit from having its own relational data structure that identifies the essential spati0-temporal element: site identifier, time identifier, geographic coordinates.

2. Existing data structure for spatio and temporal data

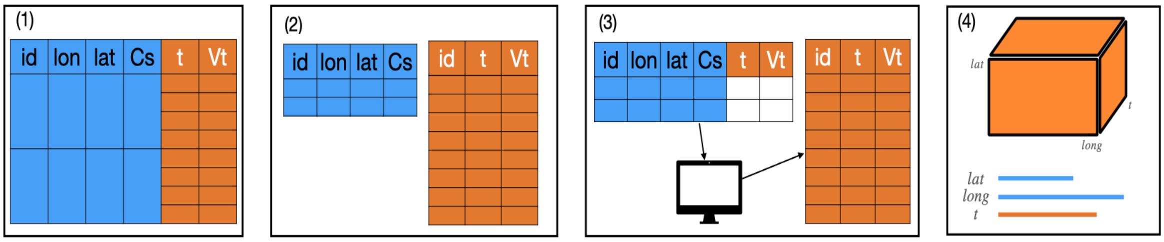

Figure 1: Illustration of incoming data formats for spatio-temporal data. (1) Data comes in as a single table; (2) Separate tables for spatial and temporal variables; (3) A single table with all the parameters used to query the database and a separate table for queried data; and (4) Cubical data in array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

3. Cubble data structure

3.1. Spatio-temporal data in the wild

Spatio-temporal data don't usually come to the analysts in various forms. Figure 1 groups them into four categories. Despite whichever form the data is in, all the forms have a group identifier, `key`, a time column, `t`, and variables that either various across group, i.e. `lon`, `lat`, C_s, or across time, i.e. V_t. These identifiers will be the building blocks for the data structure introduced below.

3.2. A new data structure: cubble

A cubble, in essence, wires the spatial and temporal dimension into a single object while provides two forms for manipulation each dimension separately.

In the nest form, Figure 2a), a cubble:

- defines each group in a row,
- displays the group-related variables in columns, and
- nests all the time-related variables into a column called `ts`.

In the long form, Figure 2b), a cubble:

- defines the combination of group and timestamp as a row
- displays time-related variables, along with the group identifier in columns, and
- stores group-related variables as a attributes called `spatial`.

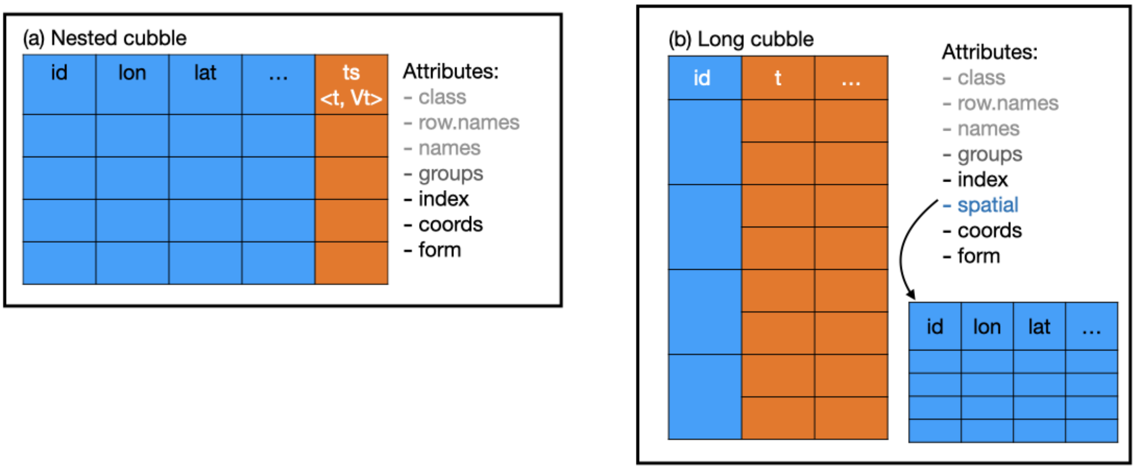

Figure 2: Illustration of nested and long cubble.

3.3. Create a cubble in the nested form

To create a cubble, you can use `as_cubble` with proper supply of `key`, `index`, and `coords`. [`key` defines whether a variable is spatial or temperal by looking at whether it varies across `key`. For example, `lat` is a spatial variable since for one `id` there's only one `lat` value. `index` and `coords` prescribes the temporal index and spaital coordinates used in the data. These are common variables that should present in the point pattern spatio-temporal data.] The cubble created by default is in the nested form.

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

# 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]>
```

In the `cubble` header, you can read the key variable + number, bbox, and also the name of variable nested in the `ts` column. Here in this example, the temporal variables are precipitation, `prcp`, maximum temporature, `tmax`, and minimum temperature, `tmin`.

In this structure, the nested form is suitable for manipulating the spatial dimension of the data and this would include manipulating variables that 1) are inherently spatial and 2) summarise the temporal variables by `key`. Under the hood, the nested cubble is built on top of the rowwise dataframe (`rowwise_df`). This is design to simplify the code when working with the temporal variables - they are nested into the list-column `ts`.

3.4. Stretch a nested cubble into the long form

The verb `stretch()` switch the cubble from the nested form into a long form. Under the hood, it first extracts the spatial variables into a separate tibble to store in the attribute `spatial` and then unnests the `ts` column to show the temporal content:

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

# 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
 6 ASN00009021 2020-01-06     0  34.8  13.1
 7 ASN00009021 2020-01-07     0  32.8  15.1
 8 ASN00009021 2020-01-08     0  30.4  17.4
 9 ASN00009021 2020-01-09     0  28.7  17.3
10 ASN00009021 2020-01-10     0  32.6  15.8
# ... with 1,820 more rows
```

Notice here that the third line in the header is changed to reflect the spatial variables stored. This is a format suitable for computing time-wise variables.

3.5. Tamp a long cubble back to the nested form

Manipulation on the spatial and temporal dimension can be an iterative process. Many times, we may decide to go back to the nested form after some temporal manipulation. The verb to switch a long cubble back to the nested form is `tamp()`:

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

# 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]>
```

3.6. Migrate spatial variables to a long cubble

As an output to be supplied to further visualisation or modelling, analysts would usually like the spatial and temporal variables to be in the same table. `migrate()` moves the spatial variables in the attribute `spatial` into the long form cubble.

```
R> (cubble_long %>% migrate(long, lat))
```

```
# 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
 6 ASN00009021 2020-01-06     0  34.8  13.1  116. -31.9
 7 ASN00009021 2020-01-07     0  32.8  15.1  116. -31.9
 8 ASN00009021 2020-01-08     0  30.4  17.4  116. -31.9
 9 ASN00009021 2020-01-09     0  28.7  17.3  116. -31.9
10 ASN00009021 2020-01-10     0  32.6  15.8  116. -31.9
# ... with 1,820 more rows
```

3.7. Support on hierarchical structure

`switch_key()`

3.8. Integrating into a tidy workflow

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: `mutate`, `filter`, `summarise`, `select`, `arrange`, `rename`, `left_join`, `group_by`, `ungroup`, and the slice family (`slice_head`, `slice_tail`, `slice_sample`, `slice_min` and `slice_max`) into a table.

`cubble` is also compatible with `tsibble` in the sense that the original list-column can be a `tbl_ts` object. Duplicates and gaps shoudl 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.

4. Examples

4.1. Australia precipitation pattern in 2020

Forming a cubble + basic tidyverse verbs - Vig 2 Aggregation - Vig 4

4.2. Matching precipitation and river level in Victria water gauges

Matching - Vig 3

5. Conclusion

6. Old stuff

Many spatial and spatio-temporal data structures have been developed by the R-spatial team for both raster and vector spatial data. For vector spatial data, which is the focus of this paper, `sf` (?) represents spatial vector information with simple features: points, lines, polygons and their multiples. Various `st_` function are designed to manipulate these features based on their geometric relationships. For spatio-temporal data, `stars` (Pebesma 2021) can represent both raster and vector data using multi-dimensional array. However, the underlying array structure can be difficult to operate for data analysts who are more familiar with a flat 2D data frame structure used by the tidyverse ecosystem.

In the temporal aspect, the `tsibble` (?) structure and its tidyverts ecosystem have provided a [. . .] workflow to work with temporal data. In a tsibble structure, temporal data is characterised by `index` and `key` where `index` is the temporal identifier and `key` is the identifier for multiple series, which could be used as a spatio identifier. However, a tsibble object, by construction, always requires the `index` in its structure. This makes it less appealing for spatio-temporal data since the output of calculated spatio-specific variables (i.e. features of each series) don't have the time dimension. Analysts will either need to have an additional step to join this output to the original tsibble or operate with variables stored in two separate objects. In addition, the long form structure of a tsibble object means spatio variables (i.e. longitude, latitude, and features of each series if joined back to the tsibble) of each spatio identifier will be repetitively recorded at each timestamp. This repetition is unnecessary and would inflate the object size for long series.

7. A new data structure for spatio-temporal data

The main difficulty and challenge

The main difficulty in visualising this type of data is to show information in both space and time dimension with the proper level of details without information overflow. This would sometimes require aggregating the time dimension into the proper level or slicing the data into a reasonable number of subset for display. In this sense, a data structure that regulates the manipulation spatio-temporal data will benefit the analysis workflow. While many implementations focus on manipulating and visualising pure spatial or temporal data, there are not sufficient tools to deal with spatio-temporal data. The purpose of this paper is to introduce a spatio-temporal vector data structure for data analysis in R.

To work with spatio-temporal data, analysts can choose to either work separately on each dimension or join the two sets together, however, each approach has its own problem: While is is natural to work separately on each sheet (since spatial and temporal operations usually don't overlap), analysts will need to manually keep the other data frame up to date. For example, the following pseudo code illustrates the scenario where once the spatial dataset is filtered for those within Victoria, the temporal dataset needs to be manually updated to reflect this spatial filter.

```
R> spatial_new <- spatial %>% filter(SITES_IN_VICTORIA)
R> temporal_new <- temporal %>% filter(id %in% spatial_new$id)
```

If analysts choose to join the spatial and temporal data together, the joined dataset could be too large since each spatial variable will be repeated at each time stamp for each site. Also, recordings of the site ID from different data sources can be slightly different from each other, causing a painful checking and cleaning of site IDs before the join.

8. Create a cubble

The creation of a cubble requires the site identifier (`key`), as well as the spatial (`coords`) and temporal (`index`) identifier. `climate_flat` is already a tibble and it uses `id` to identify each station, `date` as the time identifier, and `c(long, lat)` as the spatial identifier. To create a cubble for this data, use:

```
R> climate_flat %>% as_cubble(key = id, index = date, coords = c(long, lat))

# 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]>
```

Most of the time, spatio-temporal data doesn't come into this form and analysts need to query the climate variables based on station metadata. This is also a problem illustrated in Section 3.5 in @tidydata. Here we provide a structured way to query this data based on the row-wise operator and nested list. For this type of task, one can structure a metadata into a tibble and use row-wise operator to query the climate variables into a nested list. As an example here we demonstrate the workflow to find the 5 closest stations to Melbourne. We first create a station data frame with the 5 target stations.

```
# A tibble: 5 x 8
  id            lat  long  elev name                   wmo_id  dist city
  <chr>       <dbl> <dbl> <dbl> <chr>                   <dbl> <dbl> <chr>
1 ASN00086038 -37.7  145.  78.4 essendon airport        95866  10.8 melbourne
2 ASN00086282 -37.7  145. 113.  melbourne airport       94866  20.1 melbourne
3 ASN00086077 -38.0  145.  12.1 moorabbin airport       94870  21.9 melbourne
4 ASN00088162 -37.4  145. 528.  wallan (kilmore gap)    94860  48.1 melbourne
5 ASN00087113 -38.0  144.  10.6 avalon airport          94854  48.8 melbourne
```

We can query the climate information into a nested list named `ts` for each station with the `rowwise()` operator. To create a cubble, supply the same identifiers as with the first example.

```
R> sydmel_climate <- stations %>%
+   rowwise() %>%
+   mutate(ts = list(meteo_pull_monitors(id,
+                                        date_min = "2020-01-01",
+                                        date_max = "2020-12-31",
+                                        var = c("PRCP", "TMAX", "TMIN")) %>%
+                    select(-id))) %>%
+   as_cubble(key = id, index = date, coords = c(long, lat))
```

```
# cubble:   id [5]: nested form
# bbox:     [144.47, -38.03, 145.1, -37.38]
# temporal: date [date], prcp [dbl], tmax [dbl], tmin [dbl]
  id            lat  long  elev name                  wmo_id  dist city   ts
  <chr>       <dbl> <dbl> <dbl> <chr>                  <dbl> <dbl> <chr>  <list>
1 ASN00086038 -37.7  145.  78.4 essendon airport       95866  10.8 melbo~ <tibbl~
2 ASN00086282 -37.7  145. 113.  melbourne airport      94866  20.1 melbo~ <tibbl~
3 ASN00086077 -38.0  145.  12.1 moorabbin airport      94870  21.9 melbo~ <tibbl~
4 ASN00088162 -37.4  145. 528.  wallan (kilmore gap)  94860  48.1 melbo~ <tibbl~
5 ASN00087113 -38.0  144.  10.6 avalon airport         94854  48.8 melbo~ <tibbl~
```

Below are the how the nested and long form look like for Australia climate data, which records daily precipitation, maximum and minimum temperature for 55 stations across Australia from 2015- 2020. Notice that each station forms a group in both forms and specifically, the nested and long form have a underlying `rowwise_df` and `grouped_df` respectively.

With a cubic framework on mind, different types of manipulation with cubble can be thought of as slicing the cube in various way. The table below shows how some `dplyr` verbs are mapped into the operation in a cubble. With the existing grouping on the station, additional groupping can be added with `group_by` and removed with `ungrouped`. [talk about why it is useful]

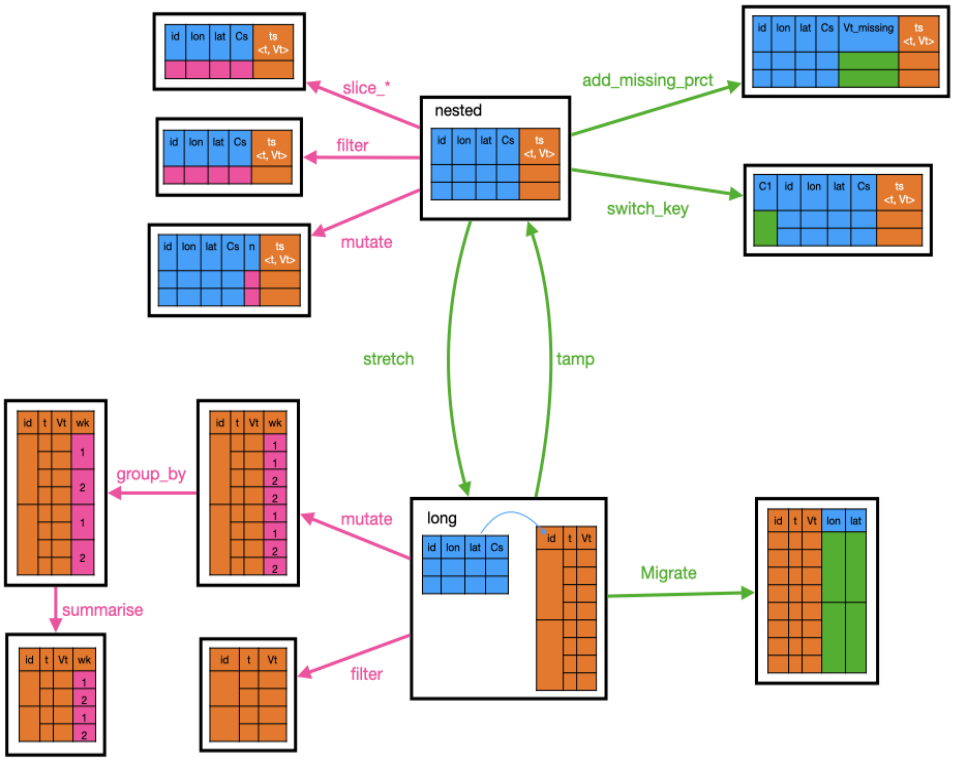

Figure 3: Cubble operations

8.1. Cubble operations

Basics

- `stretch`: nest to long form
- `tamp`: long to nest form
- `migrate`: move selected spatial variables to the long form.
- `add_dscrb_prct`: summary stats for missingness

dplyr compatibility:

- mutate, filter, summarise, select, arrange
- group and ungroup: group_by, ungroup
- slice family

Combine two cubbles

- match river and weather gauges data

- involve combining two cubbles
- join operations combine the two together by appending more rows but what we really want is to bind rows.
- bind rows also doesn't work since we want to bind only when there' s a matching????
- introduce bind_join

Hierarchical structure in cubble

- hierarchical is common.
- Given examples.
- Essence: switch between different levels
- introduce `switch_key`

9. Examples

Daily climate data (prcp, tmax, and tmin) from RNOAA - lots of stations across Australia

An exploratory data analysis questions: What's the climate profile look like in Australia

- General features: Any general trend/ fluctuation in prcp, tmax, and tmin?
- Local features: Any station stands out from the crowd?

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`.

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.

Pedersen TL (2020). *tidygraph: A Tidy API for Graph Manipulation*. R package version 1.2.0, URL `https://CRAN.R-project.org/package=tidygraph`.

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

Schieferdecker T, Müller K, Bergant D (2021). *dm: Relational Data Models.* R package version 0.2.5, URL `https://CRAN.R-project.org/package=dm`.

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. `doi:10.1080/10618600.2019.1695624`. 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`.

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