The Paper

The paper describes a new data structure `cubble` and associated workflows. In a nutshell `cubble` subsumes two data structures:

- `nested` which is indexed by spatial location and keeps the time-series data in a list column
- `long` which is indexed by time and keeps the spatial data in a hidden "spatial" attribute.

The functionality of the package consists of

- data transformation functions: `face_temporal`,`face_spacial` and `unfold`
- data matching (merging) functions: `match_sites`, `match_spatial` and `match_temporal`

The package and the proposed data structures could certainly constitute a valuable base for the spatio-temporal statistics tooling. Unfortunately the paper does not clearly describe the datastructure, nor the associated functionality. I had to read the package vignettes, run the attached code and even look into the package's code in order to fully understand the data structure.

Given the small size of the package, the paper feels a bit on the heavy side. Most of the code in the paper is not self-contained which makes it rather difficult to follow. Examples at the end of the paper don't elucidate the inner workings of the package but rather advertise the functionality of other, mostly visualization, packages.

More fundamentally, the design of the package has an immediately apparent drawback. The `cubble` class is used to represent two incompatible data types - long and nested. Internally, the two data types are distinguished by the internal attribute "form". In other words, the authors use an attribute of an object to mimic the functionality of a sub-class.

A more natural implementation would be start with two distinct data structures, both sub-classes of the abstract `cubble_df` class, `c("spatial_cubble_df", "cubble_df")` and `c("spatial_temporal_df", "cubble_df")`. Such a generic implementation would allow for natural extension of the functionality trough dedicated methods. Currently an ever-present check on "form" attribute is necessary throughout the code base. To make matters worse, `as_cubble` currently produces yet another data structure - a list of paired matches, as can be seen from example 5.1 in the paper.

Relatedly, the rationale for the parallel naming convention `long/nested` and `temporal/spatial` is not entirely clear. It seems to me that the semantically unambiguous `temporal/spatial` could be used throughout without ambiguity, thus resolving the terminological redundancy.

As `cubble` is a new package and no reverse dependencies yet exist, I would suggest rewriting the class dispatch mechanism before pursuing with the paper publication. My recommendation would be a combination of the following:

- Rework the definitional parts (sections 2 and 3) by following more closely the "design" vignette.
- Describe at a glance the core functions which operate on the class and their motivation.
- Rework section 4 by making it more concise and provide links to sections in supplementary material or dedicated vignettes.
- Rework examples section 5 into "Applications". Briefly describe the applications and provide and refer to dedicated and self-contained vignettes.

In conclusion, I believe that the package is worth supporting and publicizing. I recommend the paper for publication, but after a major rewrite and ideally after the aforementioned class refactoring of the package.

The Package

The package is generally badly documented. The meaning of the arguments is almost never clear from the documentation alone. For example the documentation of `as_cubble` states:

```
    data: the object to be created or tested as cubble
     key: the spatial identifier
   index: the time identifier
```

```
coords: the coordinates that characterise the spatial dimension
   ...: a list object to create new cubble
```

- What objects are supported as input data?
- What is the assumption of the "nestedness" of the input data.
- What is the accepted data type of `key`, `index` and `coords`?

The user would need to visit the documentation of `tsibble` in order to understand the meaning of the key and index. But the relationship to `tsibble` is not even mentioned in the docs.

The documentation of the `key` and `index` is not perfect in `tsible` either, but sheds some more light on their role:

```
    key: Variable(s) that uniquely determine time indices. 'NULL' for
         empty key, and 'c()' for multiple variables. It works with
         tidy selector (e.g. 'dplyr::starts_with()').
  index: A variable to specify the time index variable.
```

The doc states:

```
The constructor for the cubble class
```

If it's the constructor then the convention is to name it `cubble()`. `as_xyz` is the converter which translates between different data types. For example, if the proper sub-classing would be used, `as_spatial_cubble` and `as_temporal_cubble` would be more standard names rather than the `face_spacial` and `face_temporal` respectively.

Further Suggestions

- Consider not throwing when `face_spacial` is applied to the spatial object. This would allow for generic code where the "form" of the input the data is not known or does not matter.

```
nested <- climate_flat |>
  as_cubble(key = id, index = date, coords = c(long, lat))
class(nested)

face_spatial(nested) ## Error!
```

- It would be useful to print a few rows of the spatial attribute of the "long" object.
- Section 3.6 is missing `tidyr` package which is especially relevant here given its nesting and unnesting operations https://tidyr.tidyverse.org/articles/nest.html
- More comments inlined in the pdf.