# ETC5543-Creative Activity - `hexmap` R package development

Ziyao(Billy) Wang

2022-10-28

## Table of contents



### Abstract

Conventional map visualization can provide an overview on the geographic information such as country, states, counties, etc. Moreover, it is a tool for exploratory data analysis as the map illustration allows geographic information to interact with statistical values like population or income level.

The purpose of this research project is to develop a new R package (`hexmap`) that can convert the geospatial polygons (i.e. geographical regions) into a hexagon grid automatically. This offers not only a better visualization of the geographical areas but also provides accurate

statistical values alongside insightful inference. In this report, I summarise relevant literature articles, software and R packages to help inform the construction of the `hexmap` package structure and workflow. The current development results and the related testing outcome will be discussed, followed by a prospect of future development direction.

## Background

Since this is a new package development, researching for relevant information can not only enhance the understanding of matter but also lighten a systematic process or direction on the development.

After the literature review and relevant research, we constructed a blueprint for this project. We listed out the essential key steps for the package. This included but was not limited to the identification and refinement of the input data, computation of a hexagonal grid for the geographic regions and one-to-one mapping of statistical information from the original geographic location to a hexagonal region. In each step, we examined potential approaches and adjusted the processes according to the output until the desired result. Those experimental testing for each step will be demonstrated along with the package (workflow) introduction.

### Academic reviews

*Malaysia Election Data Visualization Using Hexagon Tile Grid Map*

This article illustrates a way of using a hexagonal grid to present the Malaysia election result. It mainly uses `JavaScript` as the tool and constructs multiple files to output the final visualization. These four files represent four stages of processing.

The first `setting,json` file will set up the settings for the SVG elements, including hexagon radius, tooltip functionality and colours palette used. After setup, the `parliament.json` file will prepare each hexagon to represent each parliament by state (which has unique information such as parliament code and coordination of the hexagon). With the unique key (i.e. "parliament code"), it will find the election data related to the parliament in `election.json` (it contains the actual election data for each parliament, such as voting and party information), then populate the hexagons for each particular parliament area. Lastly, the `demographic.json` file will have both state and the corresponding parliament's demographic information, which will be incorporated with the previous three files (using `d3.json()` function).

The whole procedure provides a helpful idea of how to construct our package yet since this project is based on R, the actual function and coding scheme will be different.

Notes, this article is not a public resource, yet you can find it under "reference/Malaysia Election Data Visualization Using Hexagon Tile Grid Map" here.

*A Hexagon Tile Map Algorithm for Displaying Spatial Data*

By ***Stephanie Kobakian, Dianne Cook, and Earl Duncan***

This journal article demonstrates another way of using the combination of a hexagonal grid and map for geographic information visualization. In this paper, authors applied a non-contiguous hexagonal grid with multiple comparisons to other existing approaches, including the `contiguous cartogram`, `non-contiguous cartogram` and the `Dorling cartogram`. The result suggests building a separate algorithm in order to obtain the desired output.

This article provides a reference of a clear structure of the R package algorithm on a similar issue (using a hexagonal grid to visualize the map information) to our project (i.e. `hexmap`). But, the output result is different as our goal is to produce a contiguous hexagonal grid that may be followed with a hierarchical grouping process to summarise statistical values if needed.

There is a comprehensive user guide to instruct users in the `sugerbag` application, and the algorithm flowchart has been shown below:

### Relevant news releases

*538-2016 US Election Post*

Here is an example from the `Fivethirtyeight` News release on the 2016 US election result, which uses an Albers equal-area conic projection for visualization. The hexagonal grid map is using a pixel calculation process, the original geographic information has been processed as number of pixels then computing the actual pixels display based on the population of that region.

Although this visualization is done in `Python` which is different to R programming, it still provides some useful information on how to measure our desirable final output. The GitHub for this post can be found here.

*The Guardian - UK 2017 general election*

This news post shows another way of presenting the hexagonal map as it allows users to alter the location of each hexagon tile, yet this functionality may create mistake adjustments on the geographical information. Unfortunately, the post author hadn't disclosed information or source code of this visualization.

### Relevant Github repositories (including R packages)

tilegram

The author uses the US data to demonstrate the application of the functions in the package (a `JavaScript` one). At the start, it provides some existing tile-grams for user exploration (US map). Users can export the SVG files and the TopoJSON into a web application for

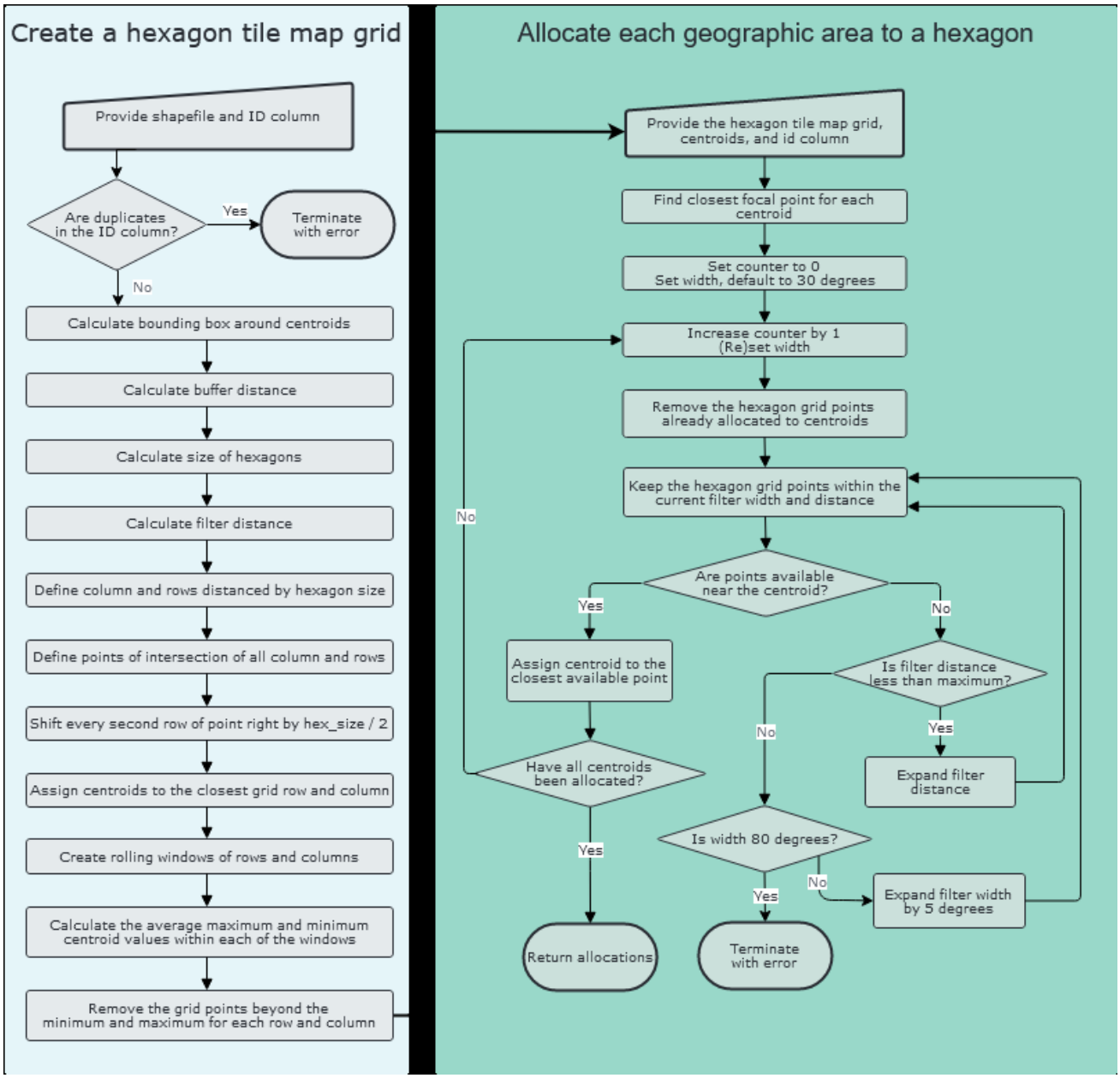



Figure 1: Algorithm flowchart - `sugerbag`

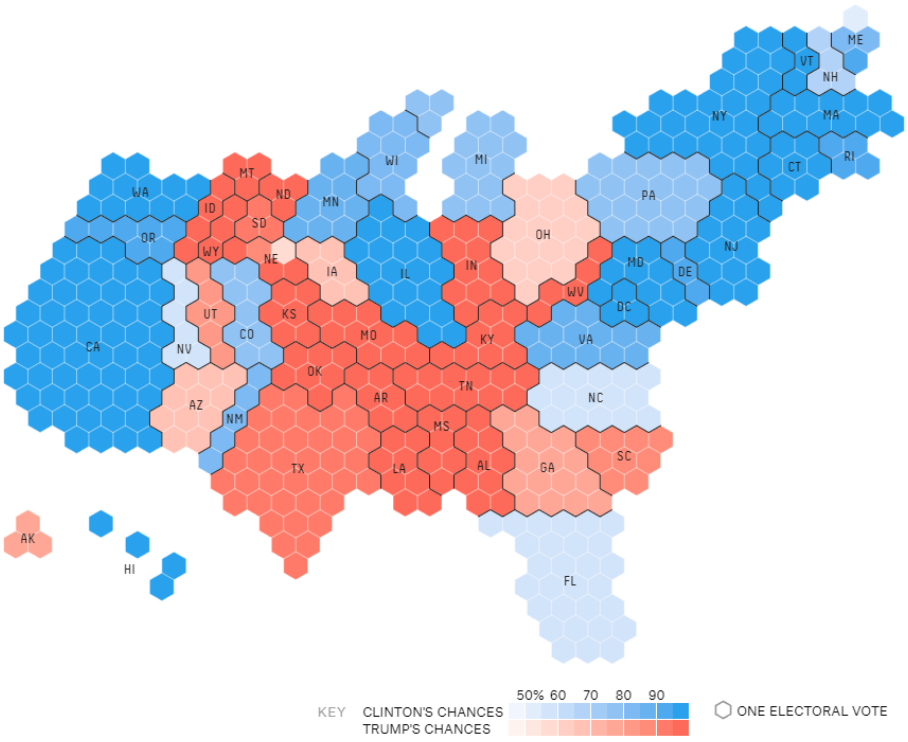



Figure 2: 2016 US election - `Fivethirtyeight`

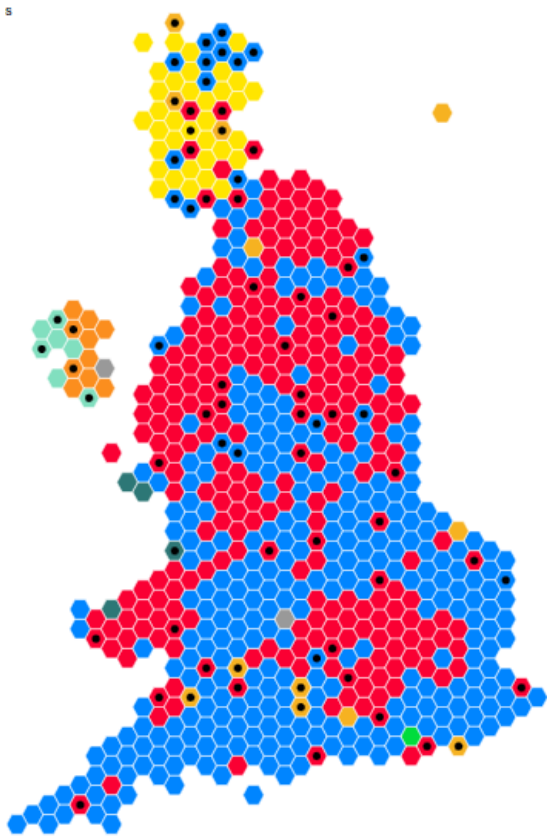


Figure 3: 2017 UK general election - `The Guardian`

customized display. Users can generate new tilegrams (customized ones) by selecting from a few prepared data sets or inputting their CSV (in a specified format using US FIPS codes). It will start from a conventional geographic map and then resize the regions to the selected data. Meanwhile, users can adjust the number of values for each tile manually or using the `Resolution` slider. (Need to alter the `delta` to `0` to make reasonable tilegrams for inferring.)

After the input of data and initial adjustment, users can move in a single, range or a specific region with a highlight of each region on the sidebar area, or remove or add a tile on the map. As for the statistical accuracy of the display, it is summarized an indicator **"delta"**. If it is positive, it represents the region that has too many tiles than it should have and vice versa. **Delta** is the difference between the calculated proposed number of tiles of each region based on the population (`500,000 per tile`) and the number of tiles in each region's population from the data set.

This package illustrates another way of constructing a hexagonal map with a range of user customization functionalists, which is illuminating yet may not be suitable as this is not the main goal of our project.

`geogrid`

This `R` package turns geospatial polygons like states, counties or local authorities into regular or hexagonal grids automatically. It uses the `Hungarian algorithm` as the core algorithm in determining the allocation of statistical values on the hexagonal grid. This aims to minimize the total distance between the centroid of every original geography and its new centroid on the grid.

This algorithm has also been used in our package functions (`tile_allocate`) when assigning statistical values on the hexagonal map correctly.

`tilemaps`

This tilemaps `R` package implements an algorithm for generating hexagonal or square maps, in which each region is represented by a single tile of the same shape and size. The algorithm implemented in this package was proposed by Graham McNeill and Scott Hale in the paper "Generating Tile Maps" (2017).

It shows a reasonable output that we expected in our project, yet there are several limitations in this package. Firstly, the algorithm can only produce a single tile (either hexagonal or square) for each area, and the allocation of tiles to the geographical locations may not be accurate if there are multiple sub-areas for that particular region such as suburbs in each city or state. Secondly, this package can only output a contiguous cartogram and have to manually allocate the separate geographical locations alongside the main map. Thus, it is a useful reference yet not directly related to our project as we wish to visualize the statistical values on a sub-level like electoral in each state.

`cartogram`

This R package can construct a continuous area cartogram, non-contiguous area cartogram, and non-overlapping circles cartogram. The main inspiration is the improvement of input data quality for statistical values allocation by distortion on the original map according to a weighting variable. This enlightens us to impose a data pre-processing on input data before implementing computation for the next step. Thus, we designed the input data refinement as the initial step to not only improve the computation efficiency but also enhance the allocation accuracy for statistical values.

## Motivation

Map illustration takes a vital role in exploratory data analysis (EDA), as it not only provides statistical information visualization but also adds geographical information to allow insightful analysis based on a specific region or area. While a map is delivering important information to users, sometimes it conveys misleading information.

With an example illustration on ABC News release on the 2022 Australian election result, it inspires an R package development to automate the conversion of conventional map visualization to a hexagonal grid for an accurate and insightful statistical inference. Therefore, `hexmap` will reshape the geographical area into multiple hexagons and allocate the statistics properly thus concluding reasonable and proper inference with an automatic process.

## Package development - `hexmap` Tour

### Package Installation

```
install.packages("remotes")
remotes::install_github("numbats/hexmap")
```

### Briefing for `hexmap`

The package will automate the conversion of **spatial polygons into hexagonal grids**, and allow users to better visualize statistical values associated with each geographic region. The sample data here is the 2022 Australian Election Data from AEC. As for input data type, it should be a Simple Features (sf) object.

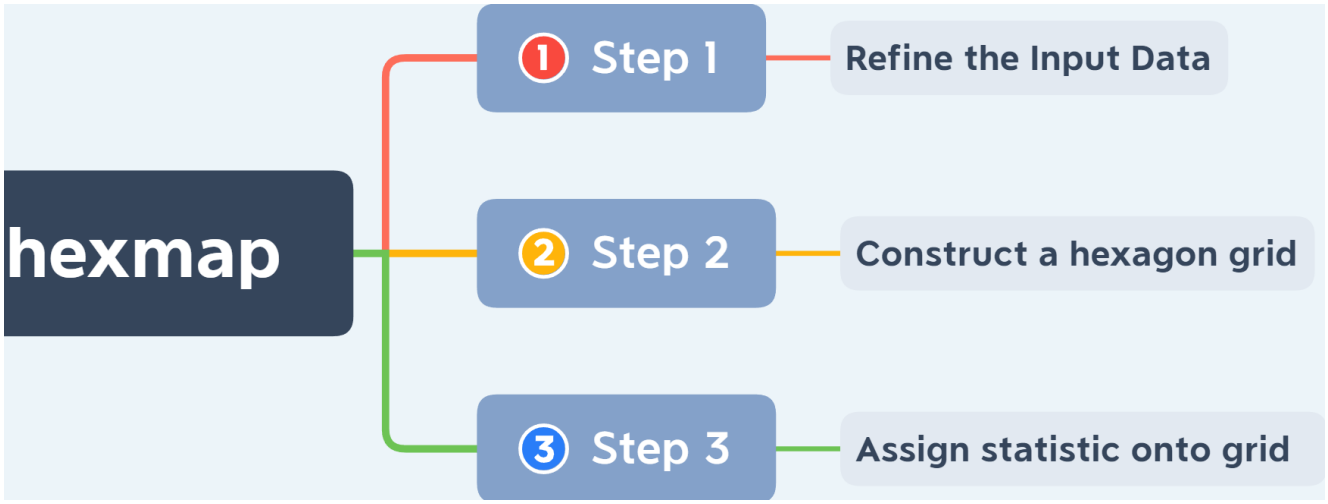



Figure 4: Flowchart - `hexmap`

### General workflow

#### Step 1 - Refining the input data

There are multiple reasons for implementing a refinement on input data. Firstly, this will **speed up** the computation of later processes (i.e. **steps 2 & 3**). Secondly, refined data is expected to perform a better assignment of statistics on the hexagonal grid. More importantly, the current development feedback suggests this initial data pre-processing has a significant effect on allocating statistical information onto the geographic map correctly. Thus, the refinement of input data is vital for desirable output.

There are multiple ways of simplifying input data size by using different methods.

Here is the example result on data downsizing by using both `st_simplify()` from `sf` and functions from `sfheaders` and `rmapshaper`, using sample data `abs_ced` from `ozmaps`.

```
# simplifying regions based on a distance argument,
## but large distances can completely remove polygons.
inputdata %>% sf::st_simplify(dTolerance = 3000)
```

```
# simplifying regions to a proportion of their original vertices
inputdata %>%
  sfheaders::sf_remove_holes() %>%
  rmapshaper::ms_simplify(keep = 0.0001, keep_shapes = TRUE)
 # keep_shapes: Prevent small polygon features
 ## from disappearing at high simplification
```

| Object | Size | Unit |
|---|---|---|
| Original Data | 2824360 | bytes |
| st_simplify | 302624 | bytes |
| ms_simplify | 141912 | bytes |

Figure 5: Input data size reduction

For the input data refinement, we are currently testing functions from `cartogram` to distort original geographic map by a weighting variable. Here, we only use the contiguous cartogram function `cartogram_cont`, as our desired output should be a contiguous hexagonal map.

**Optimal result**: Applying a hierarchical structure, that is using one group variable like "states" and then calculating the number of electoral in each state to capture the "weight" for each state followed with a proper distortion to ensure the next step (i.e. "step 2") results in a faster and less complicated computation.

Through multiple ways of testing, the result from this function is not optimal or desirable, hence we need to develop further based on the current result in future (detail discussion on "Future directions" section).

```
cartogram::cartogram_cont("input data", weight = "weight variable", itermax = 15)
# weight: Name of the weighting variable in x
# itermax: Maximum iterations for the cartogram transformation
```

### Step 2 - Determining the number of hexagons

Function `hex_grid` (from `map_grid`) will compute the proper number of hexagons for the geographic region. This function applies a reverse computation procedure (based on `st_make_grid` function from `sf`). The key value to determine the number of hexagons for the map is the **width** of each hexagon as we use a regular (unified size) hexagon. (See detailed derivation of this function in Appendix "hexagon size".)

To show the effect of distortion from "step 1", the example uses two different input data (but the same original `sf` object). There is an obvious difference in the layout of these hexagonal grids, and the distorted one shows a more reasonable hexagon grids location than the normal one's.

```
# The input data for this function can be any forms of `sf` objects.
hexmap::hex_grid(object, n_tiles = 100)
```

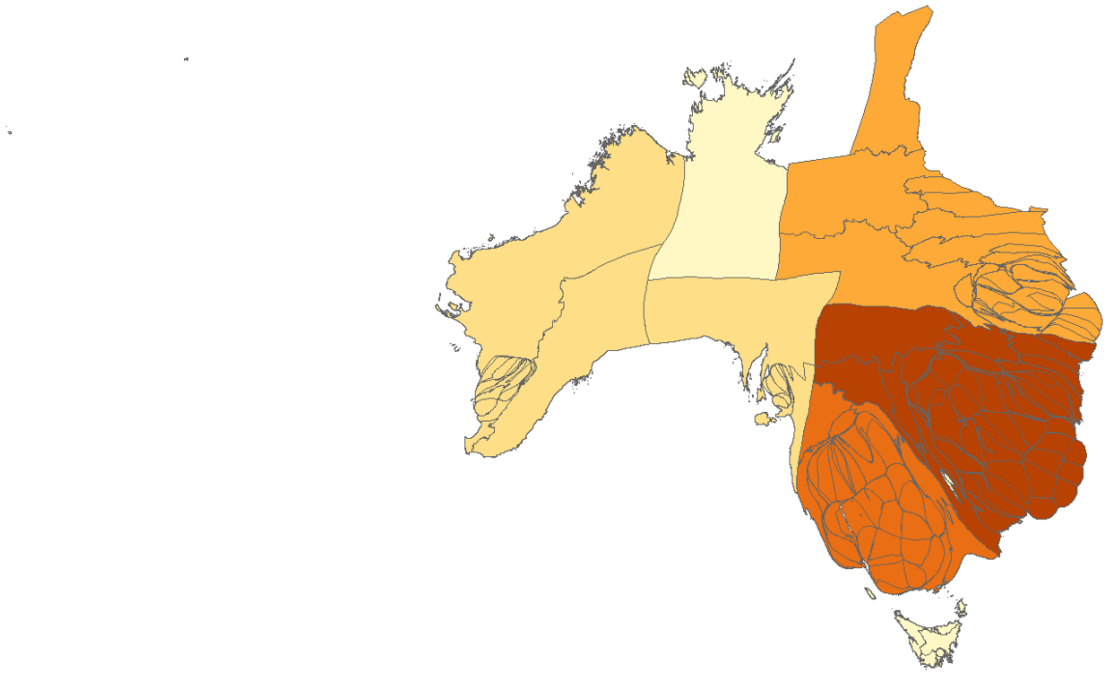


Figure 6: Distortion by `cartogram`

```
# object: input data (i.e. sf object)
# n_tiles: wanted number of hexagons
```

### Step 3 - Allocating statistical values on hexagons

In this step, it starts the assignment of the statistical information (from original input data) onto the hexagonal grid (produced by "step 2") using an algorithm (currently the "Hungarian Algorithm").

Here, the `tile_allocate` function from `hexmap` will allocate the statistical information on the hexagon grid.

Most likely, the assignment of hexagons will be measured by some sort of **distance matrix** to find the "optimal" allocation. (Yet, an actual optimal allocation may lead to a "*NP-hard problem*" for optimization). Therefore, we choose to pick the best result from a proper algorithm instead of pursuing an optimization on minimizing the distance between the centroids (of the hexagonal grid and the original map).

To emphasize the importance of data refinement can impact on the allocation result (i.e. assignment accuracy), here illustrates an example of using two different input data sets. The

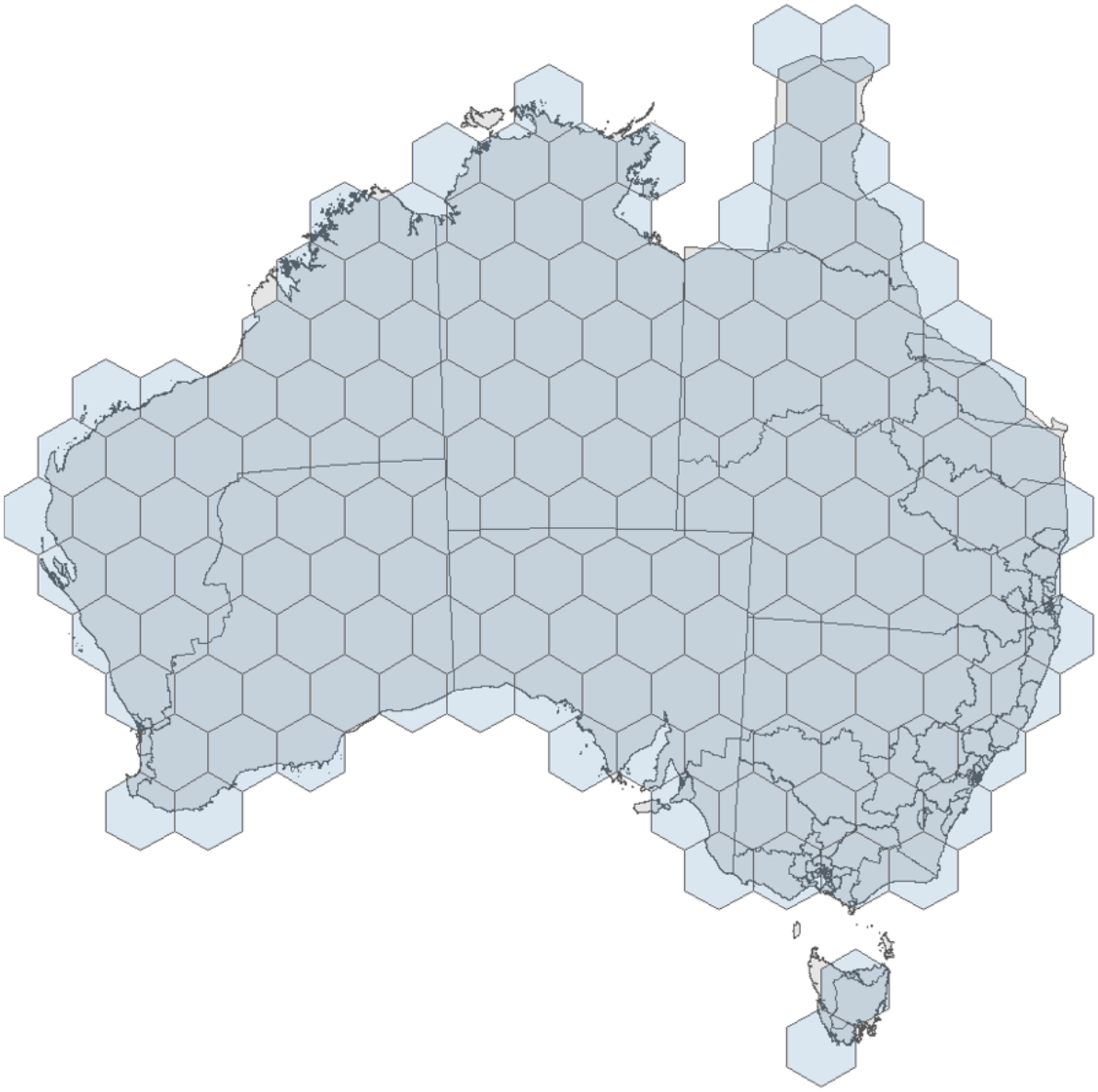


Figure 7: normal choropleth

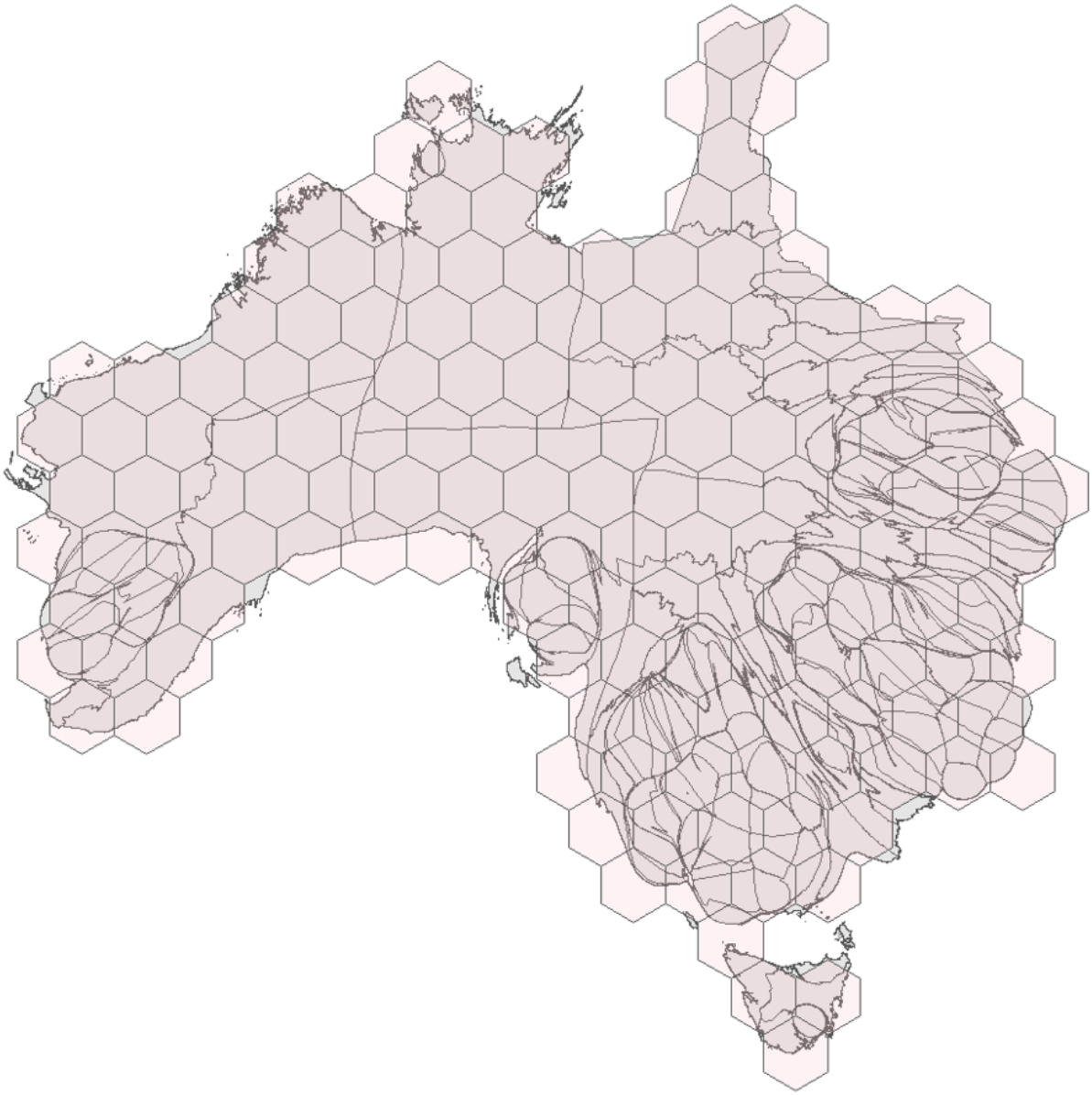


Figure 8: distorted cartogram

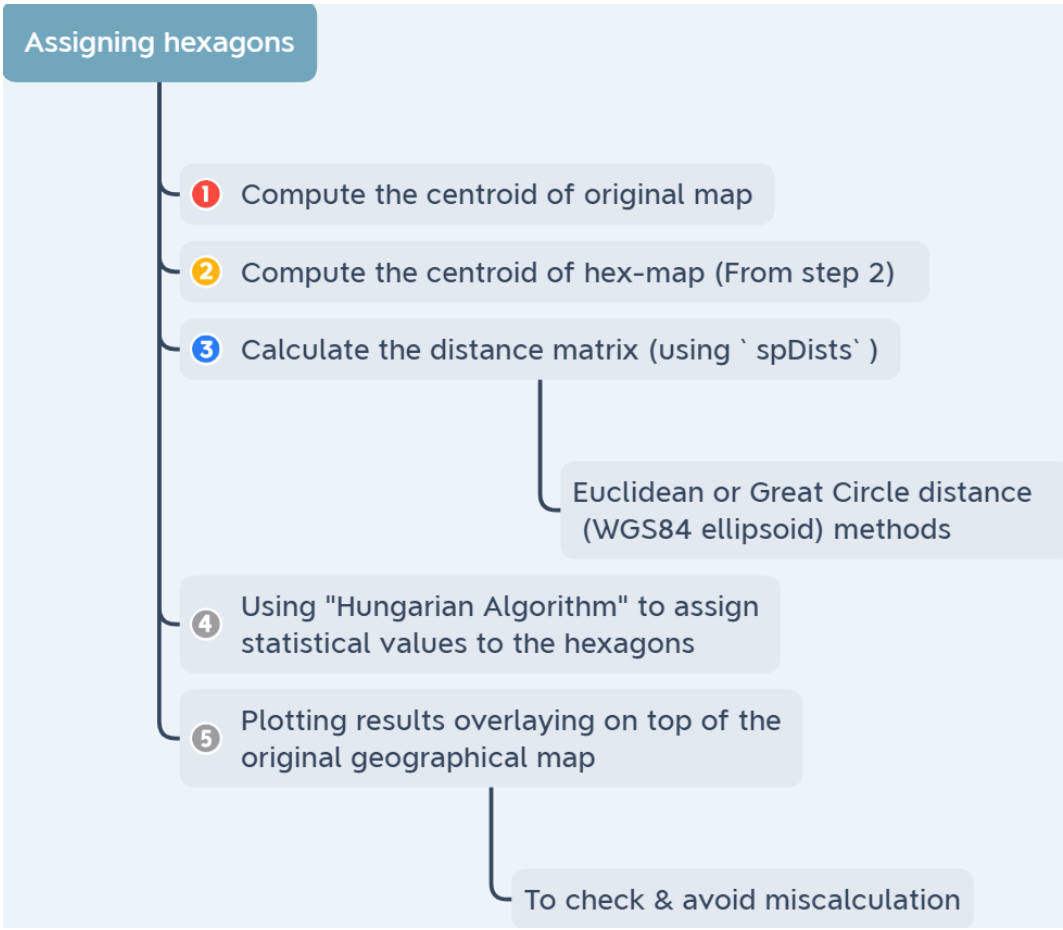
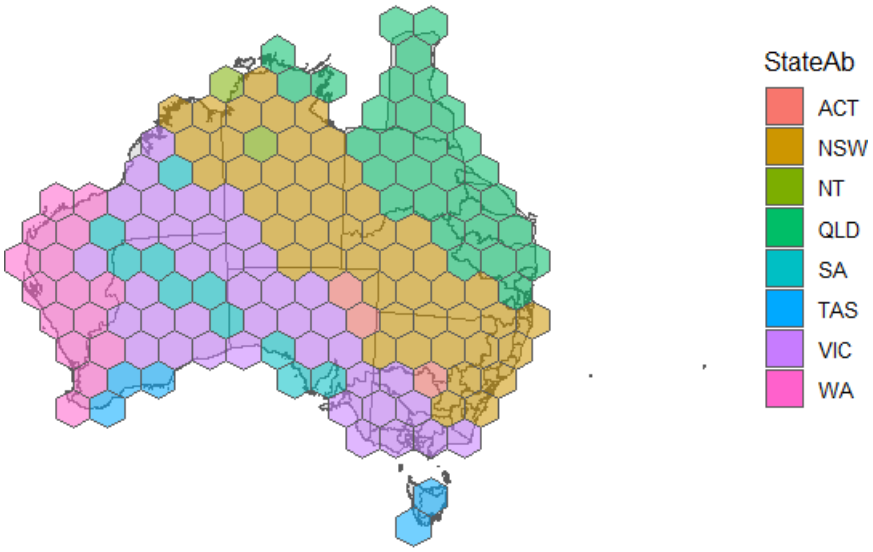



Figure 9: Flowchart - Allocating statistical values on hexagonal map

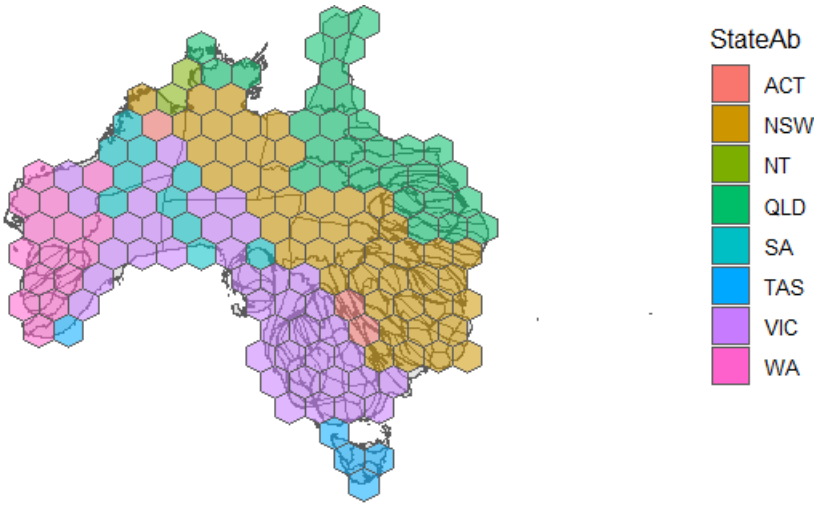
result indicates the statistical value allocation accuracy for a distorted cartogram is better than the normal map one.

```
# object: input data (i.e. sf object)
# tile: output from `hex_grid` function (i.e. a sfc object)c
hexmap::tile_allocate(object, tile)
```

- Allocation of statistical values on a **normal choropleth**




- Allocation of statistical values on a **distorted cartogram**



### Future directions

For **step 1**, the refinement of the input data can be modified better to improve the information allocation accuracy in **step 3**. Instead of using functions from the `cartogram` package, writing a new function to distort or refine the input data that can result in a more effective pre-processing of the data for later steps.

Regarding the construction of the hexagonal grid in **step 2**, the current application (an approximation of computing the number of hexagons for the grid) is reasonable and accurate enough for the current stage of development. The optimal goal will be determining the exact width of individual hexagons when constructing the grid.

The current allocation of information on hexagonal grid accuracy in **step 3** is not desirable, since there are multiple incorrect assignments for each geographic region. Through reconsidering the function (i.e. `map_allocate/tile_allocate`) and consultation (to experts), it suggests a better pre-processing of the input data (i.e. **step 1**) will improve the result significantly. Furthermore, searching for alternative algorithms may also provide additional insights into this allocation problem.

## Conclusion & Learning outcome

This R package development project is still undergoing development and I will continue work with my supervisors on this project for future improvement and modification. Welcome to leave any helpful suggestions or report issues here. Detail of code testing and report-related material will be on my_project repository to keep the actual package repository neat for future development.

This research project experience has provided me not only a chance to apply theoretical knowledge into practice but also introduce the world of package development to me. Through a vast amount of code testing and method trials, they have offered the opportunity to explore various aspects of the R community and not limit me to purely coding but other research skills.

During the project development time, supervisors illustrated a clear academic research outline to assist me in identifying the relevant information efficiently. After the literature review, the discussion on practical coding pinpointed the key difficulties and tasks, hence I could understand the directions and process of the future development process. Moreover, learning how to use GitHub **"issue"** offered an efficient platform for communication and debugging any issues during the development such as sharing my testing results with supervisors and modifying for improvement or adjustment under their instructions. Therefore, this experience not only enhances my R coding and debugging skills but also introduces a systematic way of researching and R package development.

***Acknowledgement***

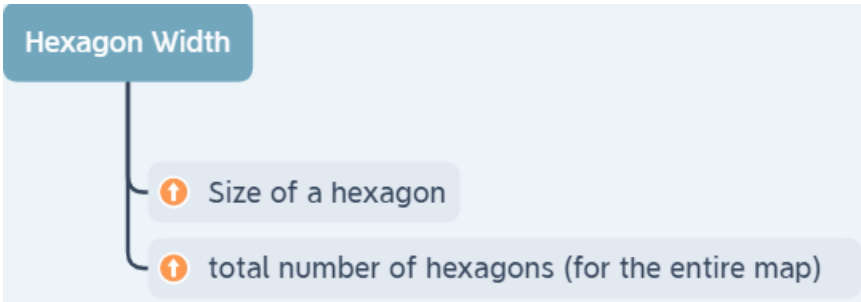
Great thanks for the guidance and supervision of **Mitchell O'Hara-Wild** and **Emi Tanaka**.

## Appendix

### hexagon size

This function (`map_tile/hex_grid`) is a process of reverse computation based on the source code of `st_make_grid` function from `sf` package. It applies a mathematics approximation to the size of the individual hexagon.

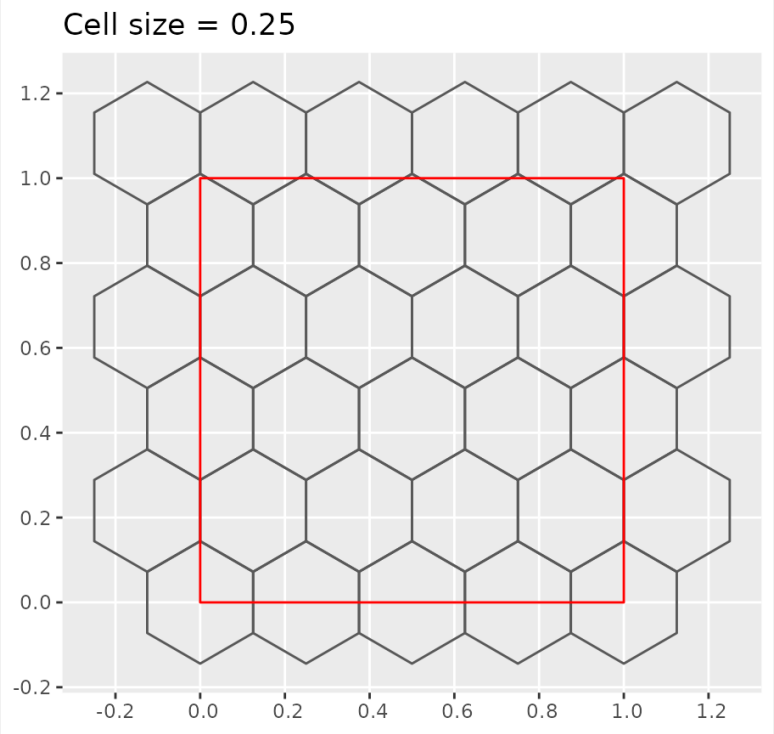
- The size of each hexagon is controlled by the "width" (assuming for regular hexagons).

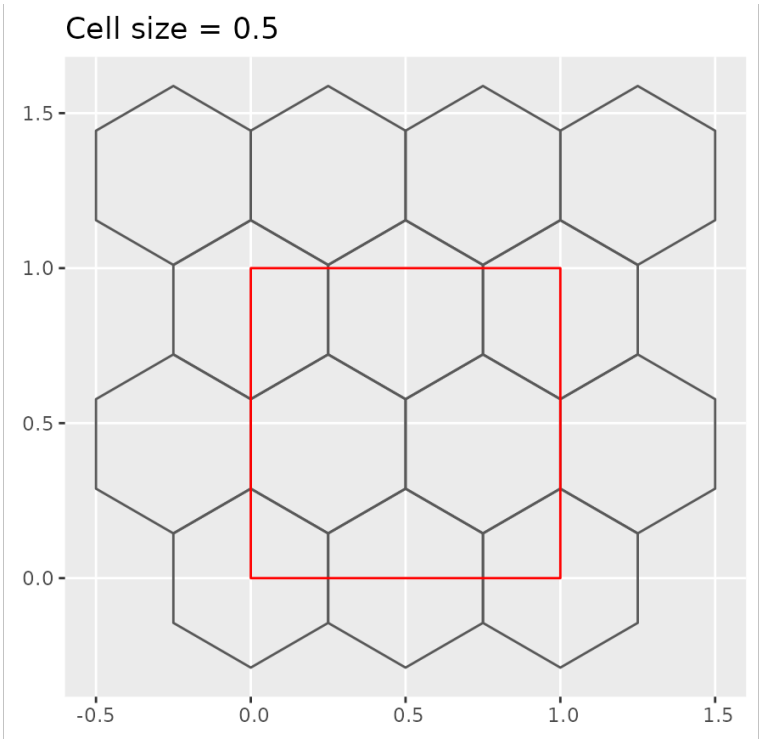



- Observe **"cell size" (i.e. width)**
- **Notes**: The number of hexagons in **odd and even** rows are different.
- Brief explanation on the "approximation" mathematics

```
# measures for the map
## Calculate the width of map
map_width <- diff(unname(st_bbox(object))[c(1, 3)])
## Calculate the height of map
map_height <- diff(unname(st_bbox(object))[c(2, 4)])
## Calculate the area of map
map_area <- st_area(st_as_sfc(st_bbox(object)))
## Calculate the map width and height ratio
map_ratio <- map_width / map_height


## Total land area
```


Cell size = 0.25
1.2
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Cell size = 0.5
1.5
1.0
0.5
0.0
-0.5
0.0
0.5
1.0
1.5

$$\frac{map\_hex}{map\_ratio} = \frac{n\_t}{land\_ratio} \nabla \cdot \frac{map\_width}{map\_height}$$
$$= \frac{n\_t \times map\_height}{land\_ratio \times map\_width}$$
$$hex\_height = \frac{map\_height}{\sqrt{map\_hex / map\_ratio}}$$
$$hex\_width = \frac{map\_height}{(1.5/\sqrt{3})}$$

```
land_area <- sum(st_area(object))
## Compute land to map ratio
land_ratio <- as.numeric(land_area / map_area)

# Number of hexagons to tile map such that ~n_tiles hexagons overlap land
map_hex <- n_tiles / land_ratio

# Size of hexagons
hex_height <- map_height / sqrt(map_hex / map_ratio)
hex_width <- hex_height / (1.5 / sqrt(3))
```

## References


[1] JJ Allaire. *quarto: R Interface to 'Quarto' Markdown Publishing System*. R package version 1.2. 2022. URL: https://CRAN.R-project.org/package=quarto.

[2] Joseph Bailey. *geogrid: Turn Geospatial Polygons into Regular or Hexagonal Grids*. R package version 0.1.1. 2022. URL: https://github.com/jbaileyh/geogrid.

[3] Sebastian Jeworutzki. *cartogram: Create Cartograms with R*. R package version 0.2.2. 2022. URL: https://github.com/sjewo/cartogram.

[4] Edzer Pebesma. "Simple Features for R: Standardized Support for Spatial Vector Data". In: *The R Journal* 10.1 (2018), pp. 439–446. DOI: 10.32614/RJ-2018-009. URL: https://doi.org/10.32614/RJ-2018-009.

[5] Emi Tanaka. *hexmap: Convert a map into a hex map*. https://github.com/emitanaka/hexmap, https://numbats.github.io/hexmap/. 2022.

[6] Hadley Wickham et al. *dplyr: A Grammar of Data Manipulation*. R package version 1.0.9. 2022. URL: https://CRAN.R-project.org/package=dplyr.