Review of the micromaps R package

Marcus W. Beck

June 26, 2015

Thanks for the opportunity to review the micromaps package. This is my first review of a software package and I honestly found it much more enjoyable than the standard journal review. I've read the JSS article and the Introduction Guide and have also reviewed the scripts that come with each. I've tried to review the content from the perspective of the end-user, focusing on ease of use in code execution and descriptions provided in the documentation. I also suggest several features that can be added or tweaks to existing functions that I think could improve the package. I realize that some of these suggestions could be considered 'drastic overhauls' to the underlying structure so I present them as discussion or ideas on further development. I'm more than happy to be a collaborator if you think any of these are worth pursuing.

General comments:

My main suggestion is to streamline the package by using a `SpatialPolygonsDataFrame` as sole input to the plotting functions, i.e., an all-inclusive spatial data object. I'm questioning the separation of data in the current format. I appreciate that separate objects can be used when lots of pre-processing is needed (i.e., polygon simplification, data summarization), and there are certainly many cases where this is appropriate, but it seems like a lot of the heavy lifting for more generic cases could be addressed under-the-hood. I think this could greatly improve the reach of the package by improving out-of-the-box functionality, while retaining the option to provide separate map/data objects when more attention is needed. Some related thoughts:

- The `create_map_table` function could be used within the plotting functions to skip a step. I assume this is needed outside of `mmplot`/`mmgroupedplot` to identify the 'id' column in cases where they are different between stat/map data. It's possible that the `create_map_table` function can be modified to search for the 'id' column automatically based on a simple matching with the columns in the stats input. Further, a `SpatialPolygonsDataFrame` object has spatial and statistical data already linked (right?) so this step could be avoided if it is the sole input.

- Stats data that are linked to the map data could be provided in raw format, with a user-supplied summary function to improve flexibility and reduce burden on the user, e.g., additional arguments passed to the plotting functions that call summary functions (see the next paragraph).

- Similarly, polygon simplification could be incorporated into the package, e.g., the raw spatial data in the `SpatialPolygonsDataFrame` could be used as input and simplified using functions in maptools or rgeos. Although I agree that this is probably easier in ArcMap, I'm certain that more generic cases can be effectively handled using R's capabilities.

I appreciate that the introduction guide provides instruction on creating new panel types, although most users will likely focus on the built-in features. I see two potential issues that limit the package from more effectively providing these tools. First, customization seems much too involved, which severely limits the number of users that can adopt the approach. Second, the built-in features rely too much on the format of the summarized input data. Creating a better balance between these options could be the focus of further package development. A limitation of the current package, as noted above, is that the input data must already be summarized. My suggestion is to modify `mmplot` (and `mmgroupedplot`) to work with raw data. These changes could allow the user to specify a pre-existing plot type on the fly (e.g., dot, bar) without having to summarize beforehand. This could also enable the use of arbitrary summary funcitons input, and potentially reduce the burden with custom maps. An example of this type of functionality is provided by the `stat_summary` function of ggplot2. The user has the option to summarize data with pre-existing functions (e.g., `mean_cl_boot`) or they can provide their own in the call to the `fun.data` argument.

The `mmgroupedplot` function is a really nice extension of `mmplot` - going from 1:1 to 1:many for the map:stats link is very intuitive. It would be good to create a function that produces the shapefile summary that's shown at the top of the grouped plots. The extra step on page 36 in the intro guide that adds the national summary could easily be automated.

I strongly encourage an object classification scheme for defining the functions (i.e., S3 or S4), particularly for the main functions. This makes it much easier to determine what object classes can be used as input. For example, the `create_map_table` function requires a `SpatialPolygonsDataFrame` object as input but this it not clear from the documentation - all that is specified is 'shapefile'. Defining an S3 method would create a generic and `SpatialPolygonsDataFrame` definition in the help file and will also return a sensible error if an unsupported object class is passed to the function. This style of documentation is really easy using the roxygen and devtools packages. For example, the generic function and the method for `SpatialPolygonsDataFrame` objects could be written in an R script like this:

```r
#' This is a function
#'
#' This is some more detail about the function
#'
#' @param input SpatialPolygonsDataFrame input
#' @param ... other arguments
#'
#' @export
mmplot <- function(input, ...) UseMethod(mmplot)

#' @rdname mmplot
#'
#' @export
#'
#' @S3method mmplot SpatialPolygonsDataFrame
mmplot.SpatialPolygonsDataFrame <- function(input, ...){

  # function stuff

}
```

Specific comments:

Related to the last point, the help file documentation could be improved in some places to clarify the exact object class for the arguments to each function. For example, The help file for `mmplot` indicates that `stat.data` is a 'table' but this really should say 'data.frame'.

This is more of a personal preference, but I'm not too crazy about a new graphics device being created each time a plotting function is executed. This can lead to rapid cluttering of the desktop.

Maybe return a warning or error if the colors argument in the plotting functions does not equal the length of groupings. Alternatively, the input colors could be passed to `colorRampPalette` to create a vector of colors equal in length to `grouping` based on an arbitrary number of color inputs. For example:

```r
colors <- c("red", "blue")
grouping <- 5

# send this to each perceptual group
percep_cols <- colorRampPalette(colors)(grouping)
```

I had to do a bit of searching to find the introduction guide as it's buried in the R library tree. Maybe include the introduction guide as a vignette in the next CRAN release. This way it will show up in searches and as a link on the CRAN web page. It can also be easily found with the following:

```r
vignette('micromap')
```

The introduction guide is helpful for showing the attributes that can be passed to the plotting function, although it may be good to include these in the help files. My go-to for looking up function arguments are the help files yet I couldn't find much of the specific documentation outside of the introduction guide.

The aspect ratio of maps changes with manual resizing of graphics device. This was caused by `scales = 'free_y'` in the `RankMaps` function. I proposed a change in a pull request to the EPA repo. Morever, there is no reason to have free y scales on the maps since each facet has the same bounding region.

Use with Shiny: I don't think there's anything that needs to be done to the structure of the package to work with Shiny. The 'reactivity' of Shiny tends to play well with most functions, so it's really more an issue of what you'd like to include in an app - what data, what map type to show, etc. It would also be much easier to create maps on the fly if the raw data are used as input to the plot functions. I'm interested in helping with app creation.