Package: mapStats (via r-universe)

September 7, 2024

Type Package

Title Geographic Display of Survey Data Statistics

Version 3.1

Date 2023-11-6

Description Automated calculation and visualization of survey data statistics on a color-coded (choropleth) map.

Depends R (>= 3.0.2), survey, lattice

License GPL (>= 2)

Imports RColorBrewer, Hmisc, classInt, sp, sf, colorspace, reshape2, ttutils

NeedsCompilation no

Author Samuel Ackerman [aut, cre]

Maintainer Samuel Ackerman <smackrmn@gmail.com>

Date/Publication 2023-11-10 07:20:02 UTC

Repository https://sam-data-guy-iam.r-universe.dev

RemoteUrl https://github.com/cran/mapStats

RemoteRef HEAD

RemoteSha d5431164e4834681bca1c6d88c38105d1e666dc1

Contents

mapStats-package *Geographic display of survey data statistics*

Description

mapStats will calculate statistics (mean, total, quantiles, variance, and standard deviation) for survey data variables, by geography level (e.g., state, county) and optional observation-level factor variables (e.g., survey year, eduction level, race). The statistics for each variable are then plotted on a shapefile with color codes (choropleth map). Easy control over visual elements such as cell censoring, color palettes, titles, plot layout, and variable renaming is enabled; see included demo(map_examples) for a full overview of options. Statistics may be calculated either weighted or unweighted. The function calcStats, which is called by mapStats, can also be used on its own to calculate variable statistics.

Details

Package:	mapStats
Type:	Package
Version:	3.1
Date:	2023-11-6
License:	GPL (>=2)

Author(s)

Samuel Ackerman Maintainer: Samuel Ackerman <smackrmn@gmail.com>

jiggleClass *Adjust class boundaries to protect from rounding errors*

Description

When using classIntervals to compute classes, occasionally there are rounding errors so that when the data is plotted and the class breaks are used for colors, for instance, the rounding error may cause a value to not be plotted with the right color, or to not be plotted at all. For this reason, we add a small value to each of the break points to accomodate a possible rounding error. This correction is negligible and should not affect plotting. When both the variable values and all the interval breaks are integer-valued (e.g. for sums of integer values), jiggleClass adjusts the interval breaks to be at the midpoints between interval endpoints. Additionally, in case ngroups is high, resulting in empty groups (even though the number of unique values is higher than ngroups), the function also eliminates the empty groups as part of the adjustment above. In case there is such a change, the palettes are also changed.

Usage

```
jiggleClass(x)
```

Arguments

x	an object of class `classIntervals` from the function `classIntervals`.

Value

an object of class `classIntervals`.

Examples

```
y <- 100*rnorm(50)

#compute class intervals using either right or left interval closure

class_list_right <- classInt::classIntervals(var=y, n=12, intervalClosure="right")
class_list_right$brks
class_list_left <- classInt::classIntervals(var=y, n=12, intervalClosure="left")
class_list_left$brks

#there should be a slight difference now between class breaks from before, but should
#have same number of observations per interval as before, and for both left and right closure

jiggleClass(x=class_list_right)
jiggleClass(x=class_list_left)


#example with discrete data, 7 groups but 9 unique values.
#classIntervals generates some empty intervals, so jiggleClass eliminates them and adjusts breaks
#in this example, with integer values, right/left closure matters more, and so the results
#will differ slightly depending on which is chosen
# here, both the observed values and the breaks under right and left are integers

y <- c(1, 1, 1, 1, 2, 3, 6, 7, 8, 9, 10, 10, 10, 10, 11)
class_list_right <- classInt::classIntervals(y, 7, intervalClosure="right")
class_list_right
#style: quantile
#  one of 28 possible partitions of this variable into 7 classes
#  [1,1]   (1,2]   (2,6]   (6,8]  (8,10] (10,10] (10,11]
#      4       1       2       2       1       4       1

class_list_left <- classInt::classIntervals(y, 7, intervalClosure="left")
class_list_left
#style: quantile
#  one of 28 possible partitions of this variable into 7 classes
#  [1,1)   [1,2)   [2,6)   [6,8)  [8,10) [10,10) [10,11]
#      0       4       2       2       2       0       5
```

```
#number of groups falls now for left closure, from 7 to 5, in this example

jiggleClass(x=class_list_right)
#style: quantile
#  one of 28 possible partitions of this variable into 7 classes
#  [0.5,1.5]   (1.5,2.5]   (2.5,6.5]   (6.5,8.5]   (8.5,9.5]  (9.5,10.5] (10.5,11.5]
#          4           1           2           2           1           4           1


jiggleClass(x=class_list_left)
#style: quantile
#  one of 70 possible partitions of this variable into 5 classes
# [0.5,1.5)  [1.5,4.5)  [4.5,7.5)  [7.5,9.5) [9.5,11.5]
#         4          2          2          2          5
```

---

mapStats *Calculate and plot survey statistics*

Description

mapStats computes statistics and quantiles of survey variable(s), with optional class variables, and displays them on a color-coded map. It calls function calcStats which is also usable outside of mapStats.

Usage

```
mapStats(d = NULL, main.var, stat = c("mean", "quantile"), quantiles = c(0.5, 0.75),
         wt.var = NULL, wt.label = TRUE, d.geo.var, by.var = NULL,
         map.file = NULL, map.geo.var = d.geo.var, makeplot = TRUE, ngroups = 4,
         separate = 1 , cell.min = 0, paletteName = "Reds", colorVec = NULL,
         map.label = TRUE, map.label.names = map.geo.var, cex.label = 0.8,
        col.label = "black", titles = NULL, cex.title = 1, var.pretty = main.var,
         geo.pretty = map.geo.var, by.pretty = by.var, as.table = TRUE,
        sp_layout.pars = list(), between = list(y = 1), horizontal.fill = TRUE,
         plotbyvar = ifelse(separate == 1 & length(main.var) > 1, FALSE, TRUE),
         num.row = 1, num.col = 1, ...)

calcStats(d, main.var, d.geo.var, stat = c("mean", "quantile"), quantiles = c(0.5, 0.75),
          by.var = NULL, wt.var = NULL, cell.min = 0)
```

Arguments

d	a data frame containing the variables to be analyzed.
main.var	a vector of character strings of the name of the variables that statistics will be calculated for. Multiple variables are allowed.

`stat` a character vector of names of statistics to calculate. Valid names are "mean", "total", "quantile", "sd" (standard deviation), and "var" (variance). "Quantile" must be included for the quantiles specified to be calculated. Statistics are printed in the order given. For instance if `stat = c("total","quantile","mean")`, then the order will be total, then quantiles in order specified in argument `quantiles`, then the mean.

`quantiles` a numeric vector of quantiles to be calculated for each variable in variable `main.var`. The quantiles must be specified as decimals between 0 and 1. In order to be calculated, "quantile" must be specified as a statistic in the argument `stat`.

`wt.var` a character string of the name of the variable to be used as sample weights in calculating statistics. The default is NULL, meaning unweighted statistics will be calculated.

`wt.label` logical. Default is TRUE, in which case automatic titles will be followed by the string '(wtd.)' or '(unwtd.)' as appropriate, depending on whether weighted statistics were calculated. If FALSE no label will be added.

`d.geo.var` a character string specifying the variable name in the dataset that is the geographic unit to calculate statistics by. When using `calCstats` outside of `mapStats` without a mapping application, `d.geo.var` would be the first class variable, and additional ones can be specified in `by.var`.

`by.var` a vector of character strings specifying variable names in the dataset `d` to use as class variables. The order in which variables are specified will affect the order in which they are combined for calculations. The default is NULL, in which case statistics are calculated by each geographic unit (`d.geo.var`) only. The function will omit from analysis any class variables that have only one level over the entire dataset. However it is still possible that a given class variable will have only one value for one of the analysis variables if, say, multiple analysis variables are used.

`map.file` an object of class `SpatialPolygonsDataFrame` on which the statistics will be plotted.

`map.geo.var` a character string of the name of the geographic identifier in the data portion of `map.file`. The default is for it to be `d.geo.var`.The values of the geographic variables in the map and original dataset must be coded the same way for merging (i.e. the factor levels must be the same).

`makeplot` logical. Default is TRUE; if FALSE, plots will not be drawn. This option can be used to calculate statistics without an available shapefile.

`ngroups` a numeric vector of the number of levels for color plotting of variable statistics. If more than one number is specified, `ngroups` will be different in each plot. See details.

`separate` numeric (or logical TRUE or FALSE for legacy cases). Accepted values are 0,1,2,3. This parameter controls how plot color breaks are calculated. If `separate=0`, all variables and statistic combinations have the same color breaks. If `separate=3`, each variable and statistic combination plot has a potentially different color break. If `separate=1` (the default), then each statistic has a different color break, but this break is the same for the statistics acoss different variables. If `separate=2`, then each variable has a different color break, but this break is

the same for all statistics of that variable. In the legacy case of this parameter, TRUE results in 1 and FALSE results in 0.

`cell.min` numeric. Indicates the minimum number of observations in a cell combination of class variables of `d.geo.var` and `by.var`. If there are fewer than that, the statistic will be NA in that combination, and if plotted, the geographic unit will be white and not used in calculating the color key.

`paletteName` a character vector containing names of color palettes for the `RColorBrewer` function `brewer.pal`. See details below for valid names. The default is to use these palettes for coloring, in which case `ngroups` will be restricted to between 3 and 9 levels, since there are at most 9 levels in `RColorBrewer` palettes. This is a good simple option. User-provided palettes can be used instead by specifying the argument `colorVec` to override this option. See details below.

`colorVec` a list where each element is vector of ordered colors; they should be ordered from light to dark for a sequential palette. These override the use of `RColorBrewer` through the `paletteName` argument. See the demo for an example of using HCL sequential palettes from the `colorspace` package. Use of the `colorVec` argument will override a value provided for `ngroups`. See details below.

`map.label` logical. Default is TRUE; if FALSE, names of the geographic regions will not be labeled on the map outputs.

`map.label.names`
a character string naming the vector from the `map.file@data` data.frame to use to label the map. The default is to use `map.geo.var`.

`cex.label` numeric. Character expansion for the labels to be printed.

`col.label` color of the label text to be printed. Default is black.

`titles` a character string of length equal to the number of statistics to be plotted, in order. Replaces the default plot titles. The default is NULL, in which case the plot titles are automatically generated. See details below.

`cex.title` numeric. Character expansion for the plot titles.

`var.pretty` a character string used to name the analysis variables `main.var` in the default plot titles. The default is to use `main.var` as the name in titles.

`geo.pretty` a character string used to name the geographic variable in the default panel strip labels. The default is to use `map.geo.var` as the name labels.

`by.pretty` a character string used to name the by-variables (optional class variables) in the default panel strip labels. The default is to use `by.var` as the name labels.

`as.table` logical. Default is TRUE, meaning the panels will be displayed in ascending order of `by.var` (top to bottom).

`sp_layout.pars` a list. This contains additional parameters to be plotted on each panel. See details section below and explanation of `sp.layout` in `spplot`. An example is provided in the demo file.

`between` list. A `lattice` argument for parameters for spacing between panels.

`horizontal.fill`
logical. Default is TRUE, meaning that given the plot arrangement specified with `num.row` and `num.col`, plots will be plotted in order left to right then down. FALSE means they will be plotted going down first and then left to right.

	The user may need to use the optional `lattice` `layout` argument to control the layout of panels within a single plot to make sure the plots print with enough space, and `par.strip.text` to control the size of panel strip fonts. Examples are shown in the demo file.
`plotbyvar`	logical. If TRUE plots will be grouped by variable, otherwise by statistic.
`num.row`	numeric. To print multiple statistics on one page, indicate the number of rows for panel arrangement. Under the default, one statistic is printed per page.
`num.col`	numeric. To print multiple statistics on one page, indicate the number of columns for panel arrangement. Under the default, one statistic is printed per page.
`...`	Further arguments, usually lattice plot arguments.

Details

`paletteName` should contain one or more names of a sequential color palette in R from the `RColorBrewer` package. These are: Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples RdPu Reds YlGn YlGnBu YlOrBr YlOrRd. The argument `ngroups` for this option should contain values between 3 and 9 since sequential color palettes have at most nine levels. The `style` argument from `classIntervals` can be included to change the method for calculating breaks (the default is by quantiles).

The default titles for the plots will be "(stat) of (variable) by (geography)", followed by either "(unwtd.)" or "(wtd.)", as appropriate. Using the `wt.label` argument controls the appearance of the weight label in the titles. Providing a value for the `titles` argument will override the default titles. This can be used, for instance, as shown below, to display percentages for a binary variable by calculating the mean of an indicator variable and specifying titles that indicate the percent is displayed.

If `quantiles` are 0 (minimum), 0.50 (median), or 1 (maximum), the statistics in the titles will be named "Minimum", "Median", and "Maximum" instead of "Q0", "Q50" or "Q100".

Parameter `separate` affects given values of `colorVec`, `paletteName`, and `ngroups`. Say there are 2 variables and 3 statistics (mean, median, and variance) to calculate for each. If any of the 3 parameters above are of length 1, (e.g. `paletteName="Reds"`), these will be used for all plots. If multiple values are given for any of the 3 parameters, these should be done with `separate` in mind. For instance, if `separate=0` (same color breaks for all plots), then the first element of each of `colorVec`, `paletteName`, and `ngroups` will be used. If `separate=1` (e.g. all of the means) have the same breaks, then the user may want to specify 3 different color palettes, for example. In this case, if `ngroups=c(3,4,5,6)`, for instance, only the first 3 values will be used since there are only 3 statistics. If `separate=2`, then only up to 2 palettes, for instance, are used. If `separate=3`, then up to 6 values for each parameter will be used. See the demo file for examples.

The `lattice` `layout` argument can be used to control the placement of panels within a graph, especially if multiple plots are done on a page.

`sp_layout.pars` must itself be a list, even if its contents are lists also. This allows overplotting of more than one object. For instance, say you had a shapefile `areas` to be colored blue, and a vector of strings `labels1` that had x-coordinates `xplaces` and y-coordinates `yplaces` to overlay on the plot. Create objects `areas_overlay= list("sp.polygons", areas, fill="blue")`, and `labels_overlay= list("panel.text", labels1, xplaces, yplaces)`. Then set argument `sp_layout.pars= list(areas_overlay, labels_overlay)`. Even if you only wanted to overlay with `areas`, you would still need to enclose it in another list, for example `sp_layout.pars= list(areas_overlay)`. An example is shown in the demo file.

Value

`mapStats` and `calcStats` return an object of class "list"

`summary.stats` a list containing the calculated statistics matrices, as well as a frequency count matrix

with attribute

`variable` the name of the variable

Note

Please see the included demo file `map_examples` for examples on controlling formatting, coloring, and other customizable options, as well as more examples

Author(s)

Samuel Ackerman

See Also

The `survey` package function `svyby` is used to calculate variable statistics, and `spplot` plots the map. See the supplied demo `map_examples` for a fuller set of examples illustrating the range of parameter options.

Examples

```
#More complex examples with formatting are shown in the map_examples
#demo for the package

#create synthetic survey dataset from internal function

surveydata <- synthetic_US_dataset()


#load map shapefile
usMap <- sf::as_Spatial(sf::st_read(system.file("shapes/usMap.shp", package="mapStats")[1]))


#Calculate weighted mean of variable income by state and year.  Display using red
#sequential color palette with 4 groups.  In the titles, rename 'income'
#by 'household income'.

# save result in an object to suppress output

res <- mapStats(d=surveydata, main.var="income", d.geo.var="state", by.var="year",
    wt.var="obs_weight", map.file=usMap, stat="mean", ngroups=4,
    paletteName="Reds", var.pretty="household income", map.label=TRUE)


#Calculate the weighted mean and 40th and 50th quantiles of the variable income
#by state, survey year, and education. Use three color palettes
```

```
res <- mapStats(d=surveydata, main.var="income", d.geo.var="state",
    by.var=c("year","educ"), wt.var="obs_weight", map.file=usMap,
    stat=c("mean","quantile"), quantiles=c(.4, .5), ngroups=6,
    paletteName=c("Reds","Greens","Blues"),
    var.pretty="household income", map.label=TRUE, num.col=1,
    num.row=2, layout=c(2,1), cex.label=.5)


#To calculate percentages of class variables, create an indicator variable, calculate
#its mean, and override the default titles to say you are calculating the percentage.
#Here we illustrate by calculating the percent of respondents by state that have income
#above $20,000.

res <- mapStats(d=surveydata, main.var="income_ge30k", wt.var="obs_weight",
    map.file=usMap, d.geo.var="state", map.geo.var="STATE_ABBR",
    paletteName="Reds", stat=c("mean"),
    titles="Percent of respondents with income at least $30,000")

#calculating statistics using the functions outside of mapStats

res <- calcStats(d=surveydata, main.var="income", stat="mean",
     d.geo.var="state", by.var=c("year", "educ"),
     wt.var="obs_weight")
```

---

usMap *Shapefile of US state boundaries*

Description

Examples are provided using the usMap shapefile with a synthetic dataset.

Source

https://www.cdc.gov/brfss/gis/gis_maps.htm

Examples

```
#read the shapefile
#display first five rows of shapefile dataset
#plot the shapefile boundaries

usMap <- sf::as_Spatial(sf::st_read(system.file("shapes/usMap.shp", package="mapStats")[1]))
head(usMap@data)
# plot the outlines of the shapefile only
sp::plot(usMap)
# plot map colored according to one of the variables in @data
```

```
sp::spplot(usMap, zcol="A187_1")
```

Index