Package 'tmaptools'

July 18, 2019

Type Package

Title Thematic Map Tools

Version 2.0-2

Description
Set of tools for reading and processing spatial data. The aim is to supply the workflow to create thematic maps. This package also facilitates 'tmap', the package for visualizing thematic maps.

License GPL-3

Encoding UTF-8

LazyData true

Date 2019-07-13

Depends R (>= 3.0), methods

Imports sp, sf (>= 0.7-1), lwgeom (>= 0.1-4), units (>= 0.6-1), grid, raster (>= 2.7-15), rgdal, rgeos, classInt, KernSmooth, magrittr, RColorBrewer, viridisLite, stats, dichromat, XML

Suggests tmap (>= 2.0), rmapshaper, osmdata, OpenStreetMap, png, shiny, shinyjs

URL `https://github.com/mtennekes/tmaptools`

BugReports `https://github.com/mtennekes/tmaptools/issues`

RoxygenNote 6.1.1

NeedsCompilation no

Author Martijn Tennekes [aut, cre]

Maintainer Martijn Tennekes <mtennekes@gmail.com>

Repository CRAN

Date/Publication 2019-07-18 06:42:04 UTC

R topics documented:

`tmaptools-package` *Thematic Map Tools*

Description

This package offers a set of handy tool functions for reading and processing spatial data. The aim of these functions is to supply the workflow to create thematic maps, e.g. read shape files, set map projections, append data, calculate areas and distances, and query OpenStreetMap. The visualization of thematic maps can be done with the tmap package.

Details

This page provides a brief overview of all package functions.

Tool functions (shape)

approx_areas	Approximate area sizes of polygons
approx_distances	Approximate distances
bb	Create, extract or modify a bounding box
bb_poly	Convert bounding box to a polygon
get_asp_ratio	Get the aspect ratio of a shape object
get_IDs	Get ID values of a shape object
is_projected	Check if the map is projected
get_projection	Get the map projection

Tool functions (data)

append_data	Append a data frame to a shape object
calc_densities	Calculate density values

Tool functions (colors)

get_brewer_pal	Get and plot a (modified) Color Brewer palette
map_coloring	Find different colors for adjacent polygons
palette_explorer	Explore Color Brewer palettes

Spatial transformation functions

aggregate_map	Aggregate the units of a map
crop_shape	Crop shape objects
points_to_raster	Bin spatial points to a raster
poly_to_raster	Convert polygons to a raster
sbind	Bind shape objects
sample_dots	Sample dots from polygons
set_projection	Set the map projection
simplify_shape	Simplify a shape
smooth_map	Create a smooth map using a kernel density estimator
smooth_raster_cover	Create a smooth cover from a raster object

Input and output functions

geocode_OSM	Get a location from an address description
read_GPX	Read a GPX file
read_osm	Read Open Street Map data
read_shape	Read a shape object
rev_geocode_OSM	Get an address description from a location
write_shape	Write a shape object

Author(s)

Martijn Tennekes <mtennekes@gmail.com>

---

aggregate_map *Aggregate map (deprecated)*

Description

Aggregate spatial polygons, spatial lines or raster objects. For spatial polygons and lines, the units will be merged with the `by` variable. For rasters, the `fact` parameter determined how many rasters cells are aggregated both horizontally and vertically. Per data variable, an aggregation formula can be specified, by default mean for numeric and modal for categorical varaibles. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
aggregate_map(shp, by = NULL, fact = NULL, agg.fun = NULL,
  weights = NULL, na.rm = FALSE, ...)
```

Arguments

shp	shape object, which is one of 1. `SpatialPolygons(DataFrame)` 2. `SpatialLines(DataFrame)` 3. `SpatialGrid(DataFrame)` 4. `SpatialPixels(DataFrame)` 5. `RasterLayer,RasterStack,or RasterBrick` 6. `sf` object if it can be coerced to an `sp` object
by	variable by which polygons or lines are merged. Does not apply to raster objects.

`fact` number that specifies how many cells in both horizontal and vertical direction are merged. Only applied to raster objects.

`agg.fun` aggregation function(s). One of the following formats:

1. One function (name) by which all variables are aggregated.
2. A vector of two function names called `"num"` and `"cat"` that determine the functions by which numeric respectively categorical variables are aggregated. For instance `c(num="mean",cat="modal")`, which calculates the mean and mode for numeric respectively categorical variables.
3. A list where per variable the (names of the) function(s) are provided. The list names should correspond to the variable names.

These predefined functions can be used: `"mean"`, `"modal"`, `"first"`, and `"last"`.

`weights` name of a numeric variable in `shp`. The values serve as weights for the aggregation function. If provided, these values are passed on as second argument. Works with aggregation functions `"mean"` and `"modal"`. Use `"AREA"` for polygon area sizes.

`na.rm` passed on to the aggregation function(s) `agg.fun`.

`...` other arguments passed on to the aggregation function(s) `agg.fun`.

Details

This function is similar to `aggregate` from the `raster` package. However, the aggregation can be specified in more detail: weights can be used (e.g. polygon area sizes). Also, an aggregation function can be specified per variable or raster layer. It is also possible to specify a general function for numeric data and a function for categorical data.

By default, the data is not aggregated. In this case, this function is similar to `unionSpatialPolygons` from the `maptools` package. The only difference is way the aggregate-by variable is specified. When using `unionSpatialPolygons`, the values have to be assigned to `IDs` whereas when using `aggregate_map` the data variable name can be assigned to `by`.

The underlying functions of `aggregate_map` for `sp` objects are `gUnaryUnion`, `gUnionCascaded`, and `gLineMerge`. For `Raster` objects, the `aggregate` is used.

This function supports `sf` objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal. Alternatively, the `tidyverse` methods `group_by` and `summarize` can be used.

Value

A shape object, in the same format as `shp`

Examples

```
## Not run:
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(land)

    # original map
    qtm(land, raster="cover_cls")
```

```
    # map decreased by factor 4 for each dimension
    land4 <- aggregate_map(land, fact=4, agg.fun="modal")
    qtm(land4, raster="cover_cls")

    # map decreased by factor 8, where the variable trees is
    # aggregated with mean, min, and max
    land_trees <- aggregate_map(land, fact=8,
        agg.fun=list(trees="mean", trees="min", trees="max"))

    tm_shape(land_trees) +
     tm_raster(c("trees.1", "trees.2", "trees.3"), title="Trees (%)") +
     tm_facets(free.scales=FALSE) +
     tm_layout(panel.labels = c("mean", "min", "max"))

    data(NLD_muni, NLD_prov)

    # aggregate Dutch municipalities to provinces
    NLD_prov2 <- aggregate_map(NLD_muni, by="province",
        agg.fun = list(population="sum", origin_native="mean", origin_west="mean",
        origin_non_west="mean", name="modal"), weights = "population")

    # see original provinces data
    as.data.frame(NLD_prov)[, c("name", "population", "origin_native",
                                "origin_west", "origin_non_west")]

    # see aggregates data (the last column corresponds to the most populated municipalities)
    sf::st_set_geometry(NLD_prov2, NULL)

    # largest municipalities in area per province
    NLD_largest_muni <- aggregate_map(NLD_muni, by="province",
        agg.fun = list(name="modal"), weights = "AREA")

    sf::st_set_geometry(NLD_largest_muni, NULL)
}

## End(Not run)
```

`append_data` *Append data to a shape object (deprecated)*

Description

Data, in the format of a data.frame, is appended to a shape object. This is either done by a left join where keys are specified for both shape and data, or by fixed order. Under coverage (shape items that do not correspond to data records), over coverage (data records that do not correspond to shape items respectively) as well as the existence of duplicated key values are automatically checked and reported via console messages. With `under_coverage` and `over_coverage` the under and over coverage key values from the last `append_data` call can be retrieved. Tip: run `append_data` without assigning the result to check the coverage. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
append_data(shp, data, key.shp = NULL, key.data = NULL,
  ignore.duplicates = FALSE, ignore.na = FALSE,
  fixed.order = is.null(key.data) && is.null(key.shp))

under_coverage()

over_coverage()
```

Arguments

shp	shape object, which is one of 1. SpatialPolygons(DataFrame) 2. SpatialPoints(DataFrame) 3. SpatialLines(DataFrame) 4. SpatialGrid(DataFrame) 5. SpatialPixels(DataFrame) 6. sf object that can be coerced as one above
data	data.frame
key.shp	variable name of shp map data to be matched with key.data. If not specified, and fixed.order is FALSE, the ID's of the polygons/lines/points are taken.
key.data	variable name of data to be matched with key.shp. If not specified, and fixed.order is FALSE, the row names of data are taken.
ignore.duplicates	should duplicated keys in data be ignored? (FALSE by default)
ignore.na	should NA values in key.data and key.shp be ignored? (FALSE by default)
fixed.order	should the data be append in the same order as the shapes in shp?

Details

This function supports sf objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal. Alternatively, the tidyverse method left_join can be used.

Value

Shape object with appended data. Tip: run append_data without assigning the result to check the coverage.

Examples

```
## Not run:
if (require(tmap)) {
    data(World)

    f <- tempfile()
```

```
    download.file("http://kejser.org/wp-content/uploads/2014/06/Country.csv", destfile = f)
    domain_codes <- read.table(f, header=TRUE, sep="|")
    unlink(f)

    domain_codes <- subset(domain_codes, select = c("Alpha3Code", "TopLevelDomain"))
    domain_codes$Alpha3Code <- toupper(domain_codes$Alpha3Code)

    World <- append_data(World, domain_codes, key.shp = "iso_a3", key.data = "Alpha3Code",
      ignore.na = TRUE)

    # codes in the data, but not in Europe:
    oc <- over_coverage()
    oc$value

    # Countries without appended data:
    uc <- under_coverage()

    current_mode <- tmap_mode("view")
    qtm(World[uc$id,], text="name")

    # plot the result
    qtm(World, text="TopLevelDomain")
    tmap_mode(current_mode)
}

## End(Not run)
```

`approx_areas` *Approximate area sizes of the shapes*

Description

Approximate the area sizes of the polygons in real-world area units (such as sq km or sq mi), proportional numbers, or normalized numbers. Also, the areas can be calibrated to a prespecified area total. This function is a convenient wrapper around `st_area`.

Usage

```
approx_areas(shp, target = "metric", total.area = NULL)
```

Arguments

`shp`	shape object, i.e., an `sf` or `sp` object.
`target`	target unit, one of
	`"prop"`: Proportional numbers. In other words, the sum of the area sizes equals one.
	`"norm"`: Normalized numbers. All area sizes are normalized to the largest area, of which the area size equals one.

"metric" **(default):** Output area sizes will be either "km" (kilometer) or "m" (meter) depending on the map scale

"imperial"**:** Output area sizes will be either "mi" (miles) or "ft" (feet) depending on the map scale

other: Predefined values are "km^2", "m^2", "mi^2", and "ft^2". Other values can be specified as well, in which case to is required).

These units are the output units. See orig for the coordinate units used by the shape shp.

total.area total area size of shp in number of target units (defined by target). Useful if the total area of the shp differs from a reference total area value. For "metric" and "imperial" units, please provide the total area in squared kilometers respectively miles.

Details

Note that the method of determining areas is an approximation, since it depends on the used projection and the level of detail of the shape object. Projections with equal-area property are highly recommended. See https://en.wikipedia.org/wiki/List_of_map_projections for equal area world map projections.

Value

Numeric vector of area sizes (class units).

See Also

approx_distances

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(NLD_muni)

    NLD_muni$area <- approx_areas(NLD_muni, total.area = 33893)

    tm_shape(NLD_muni) +
        tm_bubbles(size="area", title.size=expression("Area in " * km^2))


    # function that returns min, max, mean and sum of area values
    summary_areas <- function(x) {
        list(min_area=min(x),
             max_area=max(x),
             mean_area=mean(x),
             sum_area=sum(x))
    }

    # area of the polygons
    approx_areas(NLD_muni) %>% summary_areas()
```

```
    # area of the polygons, adjusted corrected for a specified total area size
    approx_areas(NLD_muni, total.area=33893) %>% summary_areas()

    # proportional area of the polygons
    approx_areas(NLD_muni, target = "prop") %>% summary_areas()

    # area in squared miles
    approx_areas(NLD_muni, target = "mi mi") %>% summary_areas()

    # area of the polygons when unprojected
    approx_areas(NLD_muni %>% set_projection(projection="longlat")) %>% summary_areas()
}
```

`approx_distances` *Approximate distances*

Description

Approximate distances between two points or across the horizontal and vertical centerlines of a bounding box.

Usage

```
approx_distances(x, y = NULL, projection = NULL, target = NULL)
```

Arguments

`x`	object that can be coerced to a bounding box with `bb`, or a pair of coordintes (vector of two). In the former case, the distance across the horizontal and vertical centerlines of the bounding box are approximated. In the latter case, `y` is also required; the distance between points `x` amd `y` is approximated.
`y`	a pair of coordintes, vector of two. Only required when `x` is also a pair of coordintes.
`projection`	projection code, needed in case `x` is a bounding box or when `x` and `y` are pairs of coordinates. See `get_proj4`
`target`	target unit, one of: `"m"`, `"km"`, `"mi"`, and `"ft"`.

Value

If `y` is specifyed, a list of two: unit and dist. Else, a list of three: unit, hdist (horizontal distance) and vdist (vertical distance).

See Also

`approx_areas`

Examples

```
## Not run:
if (require(tmap)) {
    data(NLD_prov)

    # North-South and East-West distances of the Netherlands
    approx_distances(NLD_prov)

    # Distance between Maastricht and Groningen
    p_maastricht <- geocode_OSM("Maastricht")$coords
    p_groningen <- geocode_OSM("Groningen")$coords
    approx_distances(p_maastricht, p_groningen, projection = "longlat", target = "km")

    # Check distances in several projections
    sapply(c("eck4", "utm31", "laea_Eur", "rd", "longlat"), function(projection) {
        p_maastricht <- geocode_OSM("Maastricht", projection = projection)$coords
        p_groningen <- geocode_OSM("Groningen", projection = projection)$coords
        approx_distances(p_maastricht, p_groningen, projection = projection)
    })
}

## End(Not run)
```

---

bb *Bounding box generator*

Description

Swiss army knife for bounding boxes. Modify an existing bounding box or create a new bounding box from scratch. See details.

Usage

```
bb(x = NA, ext = NULL, cx = NULL, cy = NULL, width = NULL,
  height = NULL, xlim = NULL, ylim = NULL, relative = FALSE,
  current.projection = NULL, projection = NULL, output = c("bbox",
  "matrix", "extent"))
```

Arguments

`x` One of the following:

- A shape (from class `Spatial`, `Raster`, or `sf` (simple features)).
- A bounding box (either 2 by 2 matrix or an `Extent` object).
- Open Street Map search query. The bounding is automatically generated by querying `x` from Open Street Map Nominatim. See `geocode_OSM` and `http://wiki.openstreetmap.org/wiki/Nominatim`.

If `x` is not specified, a bounding box can be created from scratch (see details).

`ext` Extension factor of the bounding box. If 1, the bounding box is unchanged. Values smaller than 1 reduces the bounding box, and values larger than 1 enlarges the bounding box. This argument is a shortcut for both `width` and `height` with `relative=TRUE`. If a negative value is specified, then the shortest side of the bounding box (so width or height) is extended with `ext`, and the longest side is extended with the same absolute value. This is especially useful for bounding boxes with very low or high aspect ratios.

`cx` center x coordinate

`cy` center y coordinate

`width` width of the bounding box. These are either absolute or relative (depending on the argument `relative`).

`height` height of the bounding box. These are either absolute or relative (depending on the argument `relative`).

`xlim` limits of the x-axis. These are either absolute or relative (depending on the argument `relative`).

`ylim` limits of the y-axis. See `xlim`.

`relative` boolean that determines whether relative values are used for `width`, `height`, `xlim` and `ylim` or absolute. If `x` is unspecified, `relative` is set to `"FALSE"`.

`current.projection` projection that corresponds to the bounding box specified by `x`. See `get_proj4` for options.

`projection` projection to transform the bounding box to. See `get_proj4` for options.

`output` output format of the bounding box, one of:

- `"bbox"` a `sf::bbox` object, which is a numeric vector of 4: xmin, ymin, xmax, ymax. This representation used by the `sf` package.
- `"matrix"` a 2 by 2 numeric matrix, where the rows correspond to x and y, and the columns to min and max. This representation used by the `sp` package.
- `"extent"` an `raster::extent` object, which is a numeric vector of 4: xmin, xmax, ymin, ymax. This representation used by the `raster` package.

Details

An existing bounding box (defined by `x`) can be modified as follows:

- Using the extension factor `ext`.
- Changing the width and height with `width` and `height`. The argument `relavitve` determines whether relative or absolute values are used.
- Setting the x and y limits. The argument `relavitve` determines whether relative or absolute values are used.

A new bounding box can be created from scratch as follows:

- Using the extension factor `ext`.
- Setting the center coorinates `cx` and `cy`, together with the `width` and `height`.
- Setting the x and y limits `xlim` and `ylim`

Value

bounding box (see argument `output`)

See Also

`geocode_OSM`

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {

    ## load shapes
    data(NLD_muni)
    data(World)

    ## get bounding box (similar to sp's function bbox)
    bb(NLD_muni)

    ## extent it by factor 1.10
    bb(NLD_muni, ext=1.10)

    ## convert to longlat
    bb(NLD_muni, projection="longlat")

    ## change existing bounding box
    bb(NLD_muni, ext=1.5)
    bb(NLD_muni, width=2, relative = TRUE)
    bb(NLD_muni, xlim=c(.25, .75), ylim=c(.25, .75), relative = TRUE)

}

## Not run:
if (require(tmap)) {
    bb("Limburg", projection = "rd")
    bb_italy <- bb("Italy", projection = "eck4")

    tm_shape(World, bbox=bb_italy) + tm_polygons()
    # shorter alternative: tm_shape(World, bbox="Italy") + tm_polygons()
}
## End(Not run)
```

`bb_poly` *Convert bounding box to a spatial polygon*

Description

Convert bounding box to a spatial (`sfc`) object . Useful for plotting (see example). The function `bb_earth` returns a spatial polygon of the 'boundaries' of the earth, which can also be done in other projections (if a feasible solution exists).

Usage

```
bb_poly(x, steps = 100, stepsize = NA, projection = NULL)

bb_earth(projection = NULL, stepsize = 1, earth.datum = 4326,
  bbx = c(-180, -90, 180, 90), buffer = 1e-06)
```

Arguments

x	object that can be coerced to a bounding box with bb
steps	number of intermediate points along the shortest edge of the bounding box. The number of intermediate points along the longest edge scales with the aspect ratio. These intermediate points are needed if the bounding box is plotted in another projection.
stepsize	stepsize in terms of coordinates (usually meters when the shape is projected and degrees of longlat coordinates are used). If specified, it overrules steps
projection	projection in which the coordinates of x are provided, see get_proj4. For bb_earth, projection is the projection in which the bounding box is returned (if possible).
earth.datum	Geodetic datum to determine the earth boundary. By default "WGS84", other frequently used datums are "NAD83" and "NAD27". Any other PROJ.4 character string can be used. See get_proj4.
bbx	boundig box of the earth in a vector of 4 values: min longitude, max longitude, min latitude, max latitude. By default c(-180,180,-90,90). If for some projection, a feasible solution does not exist, it may be wise to choose a smaller bbx, e.g. c(-180,180,-88,88). However, this is also automatically done with the next argument, buffer.
buffer	In order to determine feasible earth bounding boxes in other projections, a buffer is used to decrease the bounding box by a small margin (default 1e-06). This value is subtracted from each the bounding box coordinates. If it still does not result in a feasible bounding box, this procedure is repeated 5 times, where each time the buffer is multiplied by 10. Set buffer=0 to disable this procedure.

Value

sfc object

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(NLD_muni)

    current.mode <- tmap_mode("view")
    qtm(bb_poly(NLD_muni))

    # restore mode
    tmap_mode(current.mode)
}
```

`calc_densities` *Calculate densities*

Description

Transpose quantitative variables to densitiy variables, which are often needed for choroplets. For example, the colors of a population density map should correspond population density counts rather than absolute population numbers.

Usage

```
calc_densities(shp, var, target = "metric", total.area = NULL,
  suffix = NA, drop = TRUE)
```

Arguments

`shp`	a shape object, i.e., an `sf` object or a `SpatialPolygons(DataFrame)`
`var`	name(s) of a qualtity variable name contained in the `shp` data
`target`	the target unit, see `approx_areas`. Density values are calculated in `var/target^2`.
`total.area`	total area size of `shp` in number of target units (defined by `unit`), `approx_areas`.
`suffix`	character that is appended to the variable names. The resulting names are used as column names of the returned data.frame. By default, `_sq_<target>`, where target corresponds to the target unit, e.g. `_sq_km`
`drop`	boolean that determines whether an one-column data-frame should be returned as a vector

Value

Vector or data.frame (depending on whether `length(var)==1` with density values. This can be appended directly to the shape file with `append_data` with `fixed.order=TRUE`.

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(NLD_muni)

    NLD_muni_pop_per_km2 <- calc_densities(NLD_muni,
        target = "km km", var = c("pop_men", "pop_women"))
    NLD_muni <- append_data(NLD_muni, NLD_muni_pop_per_km2, fixed=TRUE)

    tm_shape(NLD_muni) +
     tm_polygons(c("pop_women_km^2", "pop_women_km^2"),
            title=expression("Population per " * km^2), style="quantile") +
    tm_facets(free.scales = FALSE) +
    tm_layout(panel.show = TRUE, panel.labels=c("Men", "Women"))
}
```

crop_shape *Crop shape object*

Description

Crop a shape object (from class Spatial, Raster, or sf). A shape file x is cropped, either by the bounding box of another shape y, or by y itself if it is a SpatialPolygons object and polygon = TRUE.

Usage

```
crop_shape(x, y, polygon = FALSE, ...)
```

Arguments

x	shape object, i.e. an object from class Spatial-class, Raster, or sf.
y	bounding box, an extent, or a shape object from which the bounding box is extracted (unless polygon is TRUE and x is a SpatialPolygons object).
polygon	should x be cropped by the polygon defined by y? If FALSE (default), x is cropped by the bounding box of x. Polygon cropping only works when x is a spatial object and y is a SpatialPolygons object.
...	arguments passed on to crop

Details

This function is similar to crop from the raster package. The main difference is that crop_shape also allows to crop using a polygon instead of a rectangle.

Value

cropped shape, in the same class as x

See Also

bb

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(World, NLD_muni, land, metro)

    land_NLD <- crop_shape(land, NLD_muni)

    qtm(land_NLD, raster="trees", style="natural")

    metro_Europe <- crop_shape(metro, World[World$continent == "Europe", ], polygon = TRUE)

    qtm(World) +
```

```
    tm_shape(metro_Europe) +
     tm_bubbles("pop2010", col="red", title.size="European cities") +
     tm_legend(frame=TRUE)
}
```

`geocode_OSM` *Geocodes a location using OpenStreetMap Nominatim*

Description

Geocodes a location (based on a search query) to coordinates and a bounding box. Similar to geocode from the ggmap package. It uses OpenStreetMap Nominatim. For processing large amount of queries, please read the usage policy (`http://wiki.openstreetmap.org/wiki/Nominatim_usage_policy`).

Usage

```
geocode_OSM(q, projection = NULL, return.first.only = TRUE,
  details = FALSE, as.data.frame = NA, as.sf = FALSE,
  server = "http://nominatim.openstreetmap.org")
```

Arguments

`q`	a character (vector) that specifies a search query. For instance `"India"` or `"CBS Weg 11,Heerlen,Netherlands"`.
`projection`	projection in which the coordinates and bounding box are returned. Either a `CRS` object or a character value. If it is a character, it can either be a `PROJ.4` character string or a shortcut. See `get_proj4` for a list of shortcut values. By default latitude longitude coordinates.
`return.first.only`	Only return the first result
`details`	provide output details, other than the point coordinates and bounding box
`as.data.frame`	Return the output as a `data.frame`. If `FALSE`, a list is returned with at least two items: `"coords"`, a vector containing the coordinates, and `"bbox"`, the corresponding bounding box. By default false, unless `q` contains multiple queries
`as.sf`	Return the output as `sf` object. If `TRUE`, `return.first.only` will be set to `TRUE`.
`server`	OpenStreetMap Nominatim server name. Could also be a local OSM Nominatim server.

Value

If `as.SPDF` then a `SpatialPointsDataFrame` is returned. Else, if `as.data.frame`, then a `data.frame` is returned, else a list.

See Also

rev_geocode_OSM, bb

Examples

```
## Not run:
if (require(tmap)) {
    geocode_OSM("India")
    geocode_OSM("CBS Weg 1, Heerlen")
    geocode_OSM("CBS Weg 1, Heerlen", projection = "rd")

    data(metro)

    # sample 5 cities from the metro dataset
    five_cities <- metro[sample(length(metro), 5), ]

    # obtain geocode locations from their long names
    five_cities_geocode <- geocode_OSM(five_cities$name_long)
    sp::coordinates(five_cities_geocode) <- ~lon+lat

    # change to interactive mode
    current.mode <- tmap_mode("view")

    # plot metro coordinates in red and geocode coordinates in blue
    # zoom in to see the differences
    tm_shape(five_cities) +
     tm_dots(col = "blue") +
    tm_shape(five_cities_geocode) +
     tm_dots(col = "red")

    # restore current mode
    tmap_mode(current.mode)
}

## End(Not run)
```

---

get_asp_ratio *Get aspect ratio*

Description

Get the aspect ratio of a shape object, a tmap object, or a bounding box

Usage

```
get_asp_ratio(x, is.projected = NA, width = 700, height = 700,
  res = 100)
```

Arguments

x	shape object (either Spatial, a Raster, or an sf), a bounding box (that can be coerced by bb), or a tmap object.
is.projected	Logical that determined wether the coordinates of x are projected (TRUE) or longitude latitude coordinates (FALSE). By deafult, it is determined by the coordinates of x.
width	See details; only applicable if x is a tmap object.
height	See details; only applicable if x is a tmap object.
res	See details; only applicable if x is a tmap object.

Details

The arguments width, height, and res are passed on to png. If x is a tmap object, a temporarily png image is created to calculate the aspect ratio of a tmap object. The default size of this image is 700 by 700 pixels at 100 dpi.

Value

aspect ratio

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(World)

    get_asp_ratio(World)

    get_asp_ratio(bb(World))

    tm <- qtm(World)
    get_asp_ratio(tm)
}

## Not run:
    get_asp_ratio("Germany") #note: bb("Germany") uses geocode_OSM("Germany")

## End(Not run)
```

---

get_brewer_pal *Get and plot a (modified) Color Brewer palette*

Description

Get and plot a (modified) palette from Color Brewer. In addition to the base function brewer.pal, a palette can be created for any number of classes. The contrast of the palette can be adjusted for sequential and diverging palettes. For categorical palettes, intermediate colors can be generated. An interactive tool that uses this function is palette_explorer.

Usage

```
get_brewer_pal(palette, n = 5, contrast = NA, stretch = TRUE,
  plot = TRUE)
```

Arguments

palette	name of the color brewer palette. Run palette_explorer (or display.brewer.pal) for options.
n	number of colors
contrast	a vector of two numbers between 0 and 1 that defines the contrast range of the palette. Applicable to sequential and diverging palettes. For sequential palettes, 0 stands for the leftmost color and 1 the rightmost color. For instance, when contrast=c(.25,.75), then the palette ranges from 1/4 to 3/4 of the available color range. For diverging palettes, 0 stands for the middle color and 1 for both outer colors. If only one number is provided, the other number is set to 0. The default value depends on n. See details.
stretch	logical that determines whether intermediate colors are used for a cateogorical palette when n is greater than the number of available colors.
plot	should the palette be plot, or only returned? If TRUE the palette is silently returned.

Details

The default contrast of the palette depends on the number of colors, n, in the following way. The default contrast is maximal, so (0,1), when n = 9 for sequential palettes and n = 11 for diverging palettes. The default contrast values for smaller values of n can be extracted with some R magic: sapply(1:9,tmaptools:::default_contrast_seq) for sequential palettes and sapply(1:11,tmaptools:::default_contrast_div) for diverging palettes.

Value

vector of color values. It is silently returned when plot=TRUE.

See Also

palette_explorer

Examples

```
get_brewer_pal("Blues")
get_brewer_pal("Blues", contrast=c(.4, .8))
get_brewer_pal("Blues", contrast=c(0, 1))
get_brewer_pal("Blues", n=15, contrast=c(0, 1))

get_brewer_pal("RdYlGn")
get_brewer_pal("RdYlGn", n=11)
get_brewer_pal("RdYlGn", n=11, contrast=c(0, .4))
get_brewer_pal("RdYlGn", n=11, contrast=c(.4, 1))
```

```
get_brewer_pal("Set2", n = 12)
get_brewer_pal("Set2", n = 12, stretch = FALSE)
```

`get_IDs` *Get ID's of the shape items of sp objects*

Description

Get ID's of the shape items of sp objects. For polygons and lines, the ID attribute is used. For points, the coordinates are used.

Usage

```
get_IDs(shp)
```

Arguments

`shp`	shape object, which is one of 1. `SpatialPolygons(DataFrame)` 2. `SpatialPoints(DataFrame)` 3. `SpatialLines(DataFrame)`

Value

vector of ID's

`get_neighbours` *Get neighbours list from spatial objects*

Description

Get neighbours list from spatial objects. The output is similar to the function `poly2nb` of the `spdep` package, but uses `sf` instead of `sp`.

Usage

```
get_neighbours(x)
```

Arguments

`x`	a shape object, i.e., a `sf` object or a `SpatialPolygons(DataFrame)`.

Value

A list where the items correspond to the features. Each item is a vector of neighbours.

get_proj4 *Get a PROJ.4 character string*

Description

Get full PROJ.4 string from an existing PROJ.4 string, a shortcut, or a CRS object.

Usage

```
get_proj4(x, output = c("crs", "character", "epsg", "CRS"))
```

Arguments

x a projection. One of:

1. a PROJ.4 character string
2. a crs object
3. a CRS object
4. an EPSG code
5. one the following shortcuts codes:

"longlat" Not really a projection, but a plot of the longitude-latitude coordinates (WGS84 datum).

"wintri" Winkel Tripel (1921). Popular projection that is useful in world maps. It is the standard of world maps made by the National Geographic Society. Type: compromise

"robin" Robinson (1963). Another popular projection for world maps. Type: compromise

"eck4" Eckert IV (1906). Projection useful for world maps. Area sizes are preserved, which makes it particularly useful for truthful choropleths. Type: equal-area

"hd" Hobo-Dyer (2002). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

"gall" Gall (Peters) (1855). Another projection useful for world maps in which area sizes are preserved. Type: equal-area

"merc" Web Mercator. Projection in which shapes are locally preserved, a variant of the original Mercator (1569), used by Google Maps, Bing Maps, and OpenStreetMap. Areas close to the poles are inflated. Type: conformal

"utmXX(s)" Universal Transverse Mercator. Set of 60 projections where each projection is a traverse mercator optimized for a 6 degree longitude range. These ranges are called UTM zones. Zone 01 covers -180 to -174 degrees (West) and zone 60 174 to 180 east. Replace XX in the character string with the zone number. For southern hemisphere, add "s". So, for instance, the Netherlands is "utm31" and New Zealand is "utm59s"

"mill" Miller (1942). Projetion based on Mercator, in which poles are displayed. Type: compromise

"eqc0" Equirectangular (120). Projection in which distances along meridians are conserved. The equator is the standard parallel. Also known as Plate Carr\'ee. Type: equidistant

"eqc30" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 30 is the standard parallel. Type: equidistant

"eqc45" Equirectangular (120). Projection in which distances along meridians are conserved. The latitude of 45 is the standard parallel. Also known as Gall isographic. Type: equidistant

"laea_Eur" European Lambert Azimuthal Equal Area Projection. Similar to EPSG code 3035.

"laea_NA" North American Lambert Azimuthal Equal Area Projection. Known as SR-ORG:7314.

"rd" Rijksdriehoekstelsel. Triangulation coordinate system used in the Netherlands.

output — the output format of the projection, one of "character", "crs","epsg", or "CRS"

Value

see output

See Also

http://en.wikipedia.org/wiki/List_of_map_projections for a overview of projections. http://trac.osgeo.org/proj/ for the PROJ.4 project home page. An extensive list of PROJ.4 codes can be created with rgdal's make_EPSG.

---

is_projected *Is the shape projected?*

Description

Checks whether the shape is projected. Applicable to Spatial, Raster and sf objects. In case the projection is missing, it checks whether the coordinates are within -180/180 and -90/90 (if so, it returns FALSE).

Usage

```
is_projected(x)
```

Arguments

x — shape (from class Spatial, Raster, or sf), or projection (see get_proj4 for options)

Value

logical: TRUE if the shape is projected and FALSE otherwise.

---

map_coloring *Map coloring*

Description

Color the polygons of a map such that adjacent polygons have different colors

Usage

```
map_coloring(x, algorithm = "greedy", ncols = NA, minimize = FALSE,
  palette = NULL, contrast = 1)
```

Arguments

x	Either a shape (i.e. a sf or SpatialPolygons(DataFrame) object), or an adjacency list.
algorithm	currently, only "greedy" is implemented.
ncols	number of colors. By default it is 8 when palette is undefined. Else, it is set to the length of palette
minimize	logical that determines whether algorithm will search for a minimal number of colors. If FALSE, the ncols colors will be picked by a random procedure.
palette	color palette.
contrast	vector of two numbers that determine the range that is used for sequential and diverging palettes (applicable when auto.palette.mapping=TRUE). Both numbers should be between 0 and 1. The first number determines where the palette begins, and the second number where it ends. For sequential palettes, 0 means the brightest color, and 1 the darkest color. For diverging palettes, 0 means the middle color, and 1 both extremes. If only one number is provided, this number is interpreted as the endpoint (with 0 taken as the start).

Value

If palette is defined, a vector of colors is returned, otherwise a vector of color indices.

Examples

```
if (require(tmap) && packageVersion("tmap") >= "2.0") {
    data(World, metro)

    World$color <- map_coloring(World, palette="Pastel2")
    qtm(World, fill = "color")

    # map_coloring used indirectly: qtm(World, fill = "MAP_COLORS")
```

```
    data(NLD_prov, NLD_muni)
    tm_shape(NLD_prov) +
     tm_fill("name", legend.show = FALSE) +
    tm_shape(NLD_muni) +
     tm_polygons("MAP_COLORS", palette="Greys", alpha = .25) +
    tm_shape(NLD_prov) +
     tm_borders(lwd=2) +
     tm_text("name", shadow=TRUE) +
    tm_format("NLD", title="Dutch provinces and\nmunicipalities", bg.color="white")
}
```

`offset_line` *Create a double line or offset line (deprecated)*

Description

Create a double line or offset line. The double line can be useful for visualizing two-way tracks or emulating objects such as railway tracks. The offset line can be useful to prevent overlapping of spatial lines. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
offset_line(shp, offset)

double_line(shp, width, sides = "both")
```

Arguments

`shp`	SpatialLines(DataFrame)
`offset`	offset from the original lines
`width`	width between the left and righthand side
`sides`	character value that specifies which sides are selected: `"both"`, `"left"`, or `"right"`. The default is `"both"`. For the other two options, see also the shortcut function `offset_line`.

Details

This function supports `sf` objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal.

Value

A shape object, in the same format as `shp`

Examples

```
## Not run:
if (require(tmap)) {
### Demo to visualise the route of the Amstel Gold Race, a professional cycling race
tmpdir <- tempdir()
tmpfile <- tempfile()
download.file("http://www.gpstracks.nl/routes-fiets/f-limburg-amstel-gold-race-2014.zip",
  tmpfile, mode="wb")
unzip(tmpfile, exdir=tmpdir)

# read GPX file
AGR <- read_GPX(file.path(tmpdir, "f-limburg-amstel-gold-race-2014.gpx"))

# read OSM of Zuid-Limburg
Limburg_OSM <- read_osm(AGR$tracks, ext=1.05)

# change route part names
levels(AGR$tracks$name) <- paste(c("First", "Second", "Third", "Final"), "loop")
AGR$tracks_offset2 <- offset_line(AGR$tracks, offset=c(.0005,0,-.0005,-.001))

tm_shape(Limburg_OSM) +
tm_raster(saturation=.25) +
tm_shape(AGR$tracks_offset2) +
tm_lines(col = "name", lwd = 4, title.col="Amstel Gold Race", palette="Dark2") +
tm_shape(AGR$waypoints) +
tm_bubbles(size=.1, col="gold", border.col = "black") +
tm_text("name", size = .75, bg.color="white", bg.alpha=.25, auto.placement = .25) +
tm_legend(position=c("right", "top"), frame=TRUE, bg.color = "gold") +
tm_view(basemaps = "Esri.WorldTopoMap")

# TIP: also run the plot in viewing mode, enabled with tmap_mode("view")
}

## End(Not run)
```

---

palette_explorer *Explore color palettes*

Description

`palette_explorer()` starts an interactive tool shows all Color Brewer and viridis palettes, where the number of colors can be adjusted as well as the constrast range. Categorical (qualitative) palettes can be stretched when the number of colors exceeds the number of palette colors. Output code needed to get the desired color values is generated. Finally, all colors can be tested for color blindness. The data.frame `tmap.pal.info` is similar to `brewer.pal.info`, but extended with the color palettes from viridis.

Usage

```
palette_explorer()

tmap.pal.info
```

Format

An object of class `data.frame` with 40 rows and 4 columns.

References

http://www.color-blindness.com/types-of-color-blindness/

See Also

`get_brewer_pal`, `dichromat`, `RColorBrewer`

Examples

```
## Not run:
if (require(shiny) && require(shinyjs)) {
    palette_explorer()
}

## End(Not run)
```

---

points_to_raster *Bin spatial points to a raster (deprecated)*

Description

Bin spatial points to a raster. For each raster cell, the number of points are counted. Optionally, a factor variable can be specified by which the points are counts are split. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
points_to_raster(shp, nrow = NA, ncol = NA, N = 250000, by = NULL,
  to.Raster = NULL)
```

Arguments

`shp`	shape object. a `SpatialPoints(DataFrame)`, a `SpatialGrid(DataFrame)`, or an `sf` object that can be coerced as such.
`nrow`	number of raster rows. If `NA`, it is automatically determined by `N` and the aspect ratio of `shp`.
`ncol`	number of raster columns. If `NA`, it is automatically determined by `N` and the aspect ratio of `shp`.

N	preferred number of raster cells.
by	name of a data variable which should be a factor. The points are split and counted according to the levels of this factor.
to.Raster	not used anymore, since the output is always a raster as of version 2.0

Details

This function is a wrapper around rasterize.

This function supports sf objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal.

Value

a RasterBrick is returned when by is specified, and a RasterLayer when by is unspecified.

See Also

poly_to_raster

Examples

```
## Not run:
if (require(tmap)) {
    data(NLD_muni, NLD_prov)

    # sample points (each point represents 1000 people)
    NLD_muni_points <- sample_dots(NLD_muni, vars = "population",
        w=1000, convert2density = TRUE)

    # dot map
    tm_shape(NLD_muni_points) + tm_dots()

    # convert points to raster
    NLD_rst <- points_to_raster(NLD_muni_points, N = 1e4)

    # plot raster
    tm_shape(NLD_rst) +
     tm_raster() +
    tm_shape(NLD_prov) +
     tm_borders() +
    tm_format("NLD") + tm_style("grey")
}

## End(Not run)
```

poly_to_raster *Convert spatial polygons to a raster (deprecated)*

Description

Convert spatial polygons to a raster. The value of each raster cell will be the polygon ID number. Alternatively, if copy.data, the polygon data is appended to each raster cell.

Usage

```
poly_to_raster(shp, r = NULL, nrow = NA, ncol = NA, N = 250000,
  use.cover = FALSE, copy.data = FALSE, to.Raster = NULL, ...)
```

Arguments

shp	shape object. A SpatialPoints(DataFrame), a SpatialGrid(DataFrame), or an sf object that can be coerced as such.
r	Raster object. If not specified, it will be created from the bounding box of shp and the arguments N, nrow, and ncol.
nrow	number of raster rows. If NA, it is automatically determined by N and the aspect ratio of shp.
ncol	number of raster columns. If NA, it is automatically determined by N and the aspect ratio of shp.
N	preferred number of raster cells.
use.cover	logical; should the cover method be used? This method determines per raster cell which polygon has the highest cover fraction. This method is better, but very slow, since N times the number of polygons combinations are processed (using the getCover argument of rasterize). By default, when a raster cell is covered by multiple polygons, the last polygon is taken (see fun argment of rasterize))
copy.data	should the polygon data be appended to the raster? Only recommended when N is small.
to.Raster	not used anymore, since the "raster" output is always a RasterLayer as of version 2.0
...	arguments passed on to rasterize

Value

a RasterBrick is returned when by is specified, and a RasterLayer when by is unspecified

See Also

points_to_raster

Examples

```
## Not run:
if (require(tmap)) {
    data(NLD_muni)

    # choropleth of 65+ population percentages
    qtm(NLD_muni, fill="pop_65plus", format="NLD")

    # rasterized version
    NLD_rst <- poly_to_raster(NLD_muni, copy.data = TRUE)
    qtm(NLD_rst, raster="pop_65plus", format="NLD")
}

## End(Not run)
```

`read_GPX` *Read GPX file*

Description

Read a GPX file. By default, it reads all possible GPX layers, and only returns shapes for layers that have any features.

Usage

```
read_GPX(file, layers = c("waypoints", "tracks", "routes",
  "track_points", "route_points"), as.sf = TRUE)
```

Arguments

`file`	a GPX filename (including directory)
`layers`	vector of GPX layers. Possible options are `"waypoints"`, `"tracks"`, `"routes"`, `"track_points"`, `"route_points"`. By dedault, all those layers are read.
`as.sf`	should `sf` objects be returned? By default `TRUE`

Details

Note that this function returns `sf` objects, but still uses methods from sp and rgdal internally.

Value

for each defiend layer, a shape is returned (only if the layer has any features). If only one layer is defined, the corresponding shape is returned. If more than one layer is defined, a list of shape objects, one for each layer, is returned.

Examples

```
## Not run:
if (require(tmap)) {
### Demo to visualise the route of the Amstel Gold Race, a professional cycling race
tmpdir <- tempdir()
tmpfile <- tempfile()
download.file("http://www.gpstracks.nl/routes-fiets/f-limburg-amstel-gold-race-2014.zip",
  tmpfile, mode="wb")
unzip(tmpfile, exdir=tmpdir)

# read GPX file
AGR <- read_GPX(file.path(tmpdir, "f-limburg-amstel-gold-race-2014.gpx"))

# read OSM of Zuid-Limburg
Limburg_OSM <- read_osm(AGR$tracks, ext=1.05)

# change route part names
levels(AGR$tracks$name) <- paste(c("First", "Second", "Third", "Final"), "loop")
AGR$tracks_offset2 <- offset_line(AGR$tracks, offset=c(.0005,0,-.0005,-.001))

tm_shape(Limburg_OSM) +
tm_raster(saturation=.25) +
tm_shape(AGR$tracks_offset2) +
tm_lines(col = "name", lwd = 4, title.col="Amstel Gold Race", palette="Dark2") +
tm_shape(AGR$waypoints) +
tm_bubbles(size=.1, col="gold", border.col = "black") +
tm_text("name", size = .75, bg.color="white", bg.alpha=.25, auto.placement = .25) +
tm_legend(position=c("right", "top"), frame=TRUE, bg.color = "gold") +
tm_view(basemaps = "Esri.WorldTopoMap")
}

## End(Not run)
```

---

read_osm *Read Open Street Map data*

Description

Read Open Street Map data. OSM tiles are read and returned as a spatial raster. Vectorized OSM data is not supported anymore (see details).

Usage

```
read_osm(x, zoom = NULL, type = "osm", minNumTiles = NULL,
  mergeTiles = NULL, use.colortable = FALSE, raster, ...)
```

Arguments

x	object that can be coerced to a bounding box with bb (e.g. an existing bounding box or a shape). In the first case, other arguments can be passed on to bb (see ...). If an existing bounding box is specified in projected coordinates, plesae specify current.projection.
zoom	passed on to openmap. Only applicable when raster=TRUE.
type	tile provider, by default "osm", which corresponds to OpenStreetMap Mapnik. See openmap for options. Only applicable when raster=TRUE.
minNumTiles	passed on to openmap Only applicable when raster=TRUE.
mergeTiles	passed on to openmap Only applicable when raster=TRUE.
use.colortable	should the colors of the returned raster object be stored in a colortable? If FALSE, a RasterStack is returned with three layers that correspond to the red, green and blue values betweeen 0 and 255.
raster	deprecated
...	arguments passed on to bb.

Details

As of version 2.0, read_osm cannot be used to read vectorized OSM data anymore. The reason is that the package that was used under the hood, osmar, has some limitations and is not actively maintained anymore. Therefore, we recommend the package osmdata. Since this package is very user-friendly, there was no reason to use read_osm as a wrapper for reading vectorized OSM data.

Value

The output of read_osm is a raster object.

Examples

```
## Not run:
if (require(tmap)) {
    #### Choropleth with OSM background

    # load Netherlands shape
    data(NLD_muni)

    # read OSM raster data
    osm_NLD <- read_osm(NLD_muni, ext=1.1)

    # plot with regular tmap functions
    tm_shape(osm_NLD) +
     tm_rgb() +
    tm_shape(NLD_muni) +
     tm_polygons("population", convert2density=TRUE, style="kmeans", alpha=.7, palette="Purp

    #### A close look at the building of Statistics Netherlands in Heerlen
```

```
    # create a bounding box around the CBS (Statistics Netherlands) building
    CBS_bb <- bb("CBS Weg 11, Heerlen", width=.003, height=.002)

    # read Microsoft Bing satellite and OpenCycleMap OSM layers
    CBS_osm1 <- read_osm(CBS_bb, type="bing")
    CBS_osm2 <- read_osm(CBS_bb, type="opencyclemap")

    # plot OSM raster data
    qtm(CBS_osm1)
    qtm(CBS_osm2)

}

## End(Not run)
```

`read_shape` *Read shape file (deprecated)*

Description

Read an ESRI shape file. Optionally, set the current projection if it is missing.

Usage

```
read_shape(file, current.projection = NULL, as.sf = TRUE, ...)
```

Arguments

`file`	a shape file name (including directory)
`current.projection`	the current projection of the shape object, if it is missing in the shape file. See `get_proj4` for options. Use `set_projection` to reproject the shape object.
`as.sf`	should the shape be returned as an `sf` object?
`...`	other parameters, such as `stringsAsFactors`, are passed on to `readOGR`

Details

This function is a convenient wrapper of rgdal's `readOGR`. It is possible to set the current projection, if it is undefined in the shape file. If a reprojection is required, use `set_projection`.

For the Netherlands: often, the Dutch Rijksdriehoekstelsel (Dutch National Grid) projection is provided in the shape file without proper datum shift parameters to wgs84. This functions automatically adds these parameters. See `http://www.qgis.nl/2011/12/05/epsg28992-of-rijksdriehoekstelsel-v` (in Dutch) for details.

Value

shape object from class `Spatial` or `sf` if `as.sf` = `TRUE`

rev_geocode_OSM *Reverse geocodes a location using OpenStreetMap Nominatim*

Description

Reverse geocodes a location (based on spatial coordinates) to an address. It uses OpenStreetMap Nominatim. For processing large amount of queries, please read the usage policy (http://wiki.openstreetmap.org/wiki/Nominatim_usage_policy).

Usage

```
rev_geocode_OSM(x, y = NULL, zoom = NULL, projection = NULL,
  as.data.frame = NA, server = "http://nominatim.openstreetmap.org")
```

Arguments

x	x coordinate(s), or a spatial points object (sf or SpatialPoints)
y	y coordinate(s)
zoom	zoom level
projection	projection in which the coordinates x and y are provided. Either a CRS object or a character value. If it is a character, it can either be a PROJ.4 character string or a shortcut. See get_proj4 for a list of shortcut values. By default latitude longitude coordinates.
as.data.frame	return as data.frame (TRUE) or list (FALSE). By default a list, unless multiple coordinates are provided.
server	OpenStreetMap Nominatim server name. Could also be a local OSM Nominatim server.

Value

A data frmame or a list with all atributes that are contained in the search result

See Also

geocode_OSM

Examples

```
## Not run:
if (require(tmap)) {
    data(metro)

    # sample five cities from metro dataset
    set.seed(1234)
    five_cities <- metro[sample(length(metro), 5), ]
```

```
    # obtain reverse geocode address information
    addresses <- rev_geocode_OSM(five_cities, zoom = 6)
    five_cities <- append_data(five_cities, addresses, fixed.order = TRUE)

    # change to interactive mode
    current.mode <- tmap_mode("view")
    tm_shape(five_cities) +
     tm_markers(text="name")

    # restore current mode
    tmap_mode(current.mode)
}

## End(Not run)
```

`sample_dots` *Sample dots from spatial polygons (deprecated)*

Description

Sample dots from spatial polygons according to a spatial distribution of a population. The population may consist of classes. The output, an sf object containing spatial points, can be used to create a dot map (see `tm_dots`), where the dots are colored according to the classes. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
sample_dots(shp, vars = NULL, convert2density = FALSE, nrow = NA,
  ncol = NA, N = 250000, npop = NA, n = 10000, w = NA,
  shp.id = NULL, var.name = "class", var.labels = vars,
  target = "metric", randomize = TRUE, output = c("points", "grid"),
  orig = NULL, to = NULL, ...)
```

Arguments

`shp`	A shape object, more specifically, a `SpatialPolygonsDataFrame` or an `sf` object that can be coerced as such.
`vars`	Names of one or more variables that are contained in `shp`. If `vars` is not provided, the dots are sampled uniformly. If `vars` consists of one variable name, the dots are sampled according to the distribution of the corresponding variable. If `vars` consist of more than one variable names, then the dots are sampled according to the distributions of those variables. A categorical variable is added that contains the distrubtion classes (see `var.name`).
`convert2density`	Should the variables be converted to density values? Density values are used for the sampling algorithm, so use `TRUE` when the values are absolute counts.
`nrow`	Number of grid rows

ncol	Number of grid colums
N	Number of grid points
npop	Population total. If NA, it is recontructed from the data. If density values are specified, the population total is approximated using the polygon areas (see also target, orig and to).
n	Number of sampled dots
w	Number of population units per dot. It is the population total divided by n. If specified, n is calculated accordingly.
shp.id	Name of the variable of shp that contains the polygon identifying numbers or names.
var.name	Name of the variable that will be created to store the classes. The classes are defined by vars, and the labels can be configured with var.labels.
var.labels	Labels of the classes (see var.name).
target	target unit, see approx_areas
randomize	should the order of sampled dots be randomized? The dots are sampled class-wise (specified by vars). If this order is not randomized (so if randomize=FALSE), then the dots from the last class will be drawn on top, which may introduce a perception bias. By default randomize=TRUE, so the sampled dots are randomized to prevent this bias.
output	format of the output: use "points" for spatial points, and "grid" for a spatial grid.
orig	not used anymore as of version 2.0
to	not used anymore as of version 2.0
...	other arguments passed on to calc_densities and approx_areas

Details

This function supports sf objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal. Alternatively, st_sample can be used.

Value

A shape object, in the same format as shp

Examples

```
## Not run:
if (require(tmap)) {
    data(World)
    World_dots <- sample_dots(World, vars="pop_est_dens", nrow=200, ncol=400, w=1e6)

    tm_shape(World_dots) + tm_dots(size = .02, jitter=.1) +
     tm_layout("One dot represents one million people", title.position = c("right", "bottom"
}

## End(Not run)
```

sbind *Combine shape objects (from sp) (deprecated)*

Description

Combine shape objects from sp into one shape object. It works analogous to `rbind`.

Usage

```
sbind(...)
```

Arguments

`...`	shape objects. Each shape object is one of 1. `SpatialPolygons(DataFrame)` 2. `SpatialPoints(DataFrame)` 3. `SpatialLines(DataFrame)`

Value

shape object

set_projection *Set and get the map projection*

Description

The function `set_projection` sets the projection of a shape file. It is a convenient wrapper of `st_transform` (or `st_transform_proj`, see details) and `projectRaster` with shortcuts for commonly used projections. The projection can also be set directly in the plot call with `tm_shape`. This function is also used to set the current projection information if this is missing. The function `get_projection` is used to get the projection information.

Usage

```
set_projection(shp, projection = NA, current.projection = NA,
  overwrite.current.projection = FALSE)

get_projection(shp, guess.longlat = FALSE, output = c("character",
  "crs", "epsg", "CRS"))
```

Arguments

shp	shape object, which is an object from a class defined by the sf, sp, or raster package.
projection	new projection. See get_proj4 for options. This argument is only used to transform the shp. Use current.projection to specify the current projection of shp.
current.projection	the current projection of shp. See get_proj4 for possible options. Only use this if the current projection is missing or wrong.
overwrite.current.projection	logical that determines whether the current projection is overwritten if it already has a projection that is different.
guess.longlat	if TRUE, it checks if the coordinates are within -180/180 and -90/90, and if so, it returns the WGS84 longlat projection (which is get_proj4("longlat")).
output	output format of the projection. One of "character", "crs" (from sf package), "epsg" or "CRS" (from sp/rgdal package)

Details

For sf objects, set_projection first tries to use sf::st_transform, which uses the GDAL API. For some projections, most notably Winkel Tripel ("wintri"), is doesn't work. In these cases, set_projection will use lwgeom::st_transform_proj, which uses the PROJ.4 API.

For raster objects, the projection method is based on the type of data. For numeric layers, the bilinear method is used, and for categorical layers the nearest neighbor. See projectRaster for details.

Value

set_projection returns a (transformed) shape object with updated projection information. get_projection returns the PROJ.4 character string of shp.

---

simplify_shape *Simplify shape*

Description

Simplify a shape consisting of polygons or lines. This can be useful for shapes that are too detailed for visualization, especially along natural borders such as coastlines and rivers. The number of coordinates is reduced.

Usage

```
simplify_shape(shp, fact = 0.1, keep.units = FALSE,
  keep.subunits = FALSE, ...)
```

Arguments

shp	a SpatialPolygons(DataFrame) or a SpatialLines(DataFrame), or an sf object that can be coerced to one of them.
fact	simplification factor, number between 0 and 1 (default is 0.1)
keep.units	d
keep.subunits	d
...	other arguments passed on to the underlying function ms_simplify (except for the arguments input, keep, keep_shapes and explode)

Details

This function is a wrapper of ms_simplify. In addition, the data is preserved. Also sf objects are supported.

Value

sf object

Examples

```
## Not run:
if (require(tmap)) {
    data(World)

    # show different simplification factors
    tm1 <- qtm(World %>% simplify_shape(fact = 0.05), title="Simplify 0.05")
    tm2 <- qtm(World %>% simplify_shape(fact = 0.1), title="Simplify 0.1")
    tm3 <- qtm(World %>% simplify_shape(fact = 0.2), title="Simplify 0.2")
    tm4 <- qtm(World %>% simplify_shape(fact = 0.5), title="Simplify 0.5")
    tmap_arrange(tm1, tm2, tm3, tm4)

    # show different options for keeping smaller (sub)units
    tm5 <- qtm(World %>% simplify_shape(keep.units = TRUE, keep.subunits = TRUE),
        title="Keep units and subunits")
    tm6 <- qtm(World %>% simplify_shape(keep.units = TRUE, keep.subunits = FALSE),
        title="Keep units, ignore small subunits")
    tm7 <- qtm(World %>% simplify_shape(keep.units = FALSE),
        title="Ignore small units and subunits")
    tmap_arrange(tm5, tm6, tm7)
}

## End(Not run)
```

smooth_map *Create a smooth map (deprecated)*

Description

Create a smooth map from a shape object. A 2D kernel density estimator is applied to the shape, which can be a spatial points, polygons, or raster object. Various format are returned: a smooth raster, contour lines, and polygons. The covered area can be specified, i.e., the area outside of it is extracted from the output. Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
smooth_map(shp, var = NULL, nrow = NA, ncol = NA, N = 250000,
  unit = "km", unit.size = 1000, smooth.raster = TRUE, nlevels = 5,
  style = ifelse(is.null(breaks), "pretty", "fixed"), breaks = NULL,
  bandwidth = NA, threshold = 0, cover.type = NA, cover = NULL,
  cover.threshold = 0.6, weight = 1, extracting.method = "full",
  buffer.width = NA, to.Raster = NULL)
```

Arguments

`shp`	shape object of class `Spatial`, `Raster`, or `sf`. Spatial points, polygons, and grids are supported. Spatial lines are not.
`var`	variable name. Not needed for `SpatialPoints`. If missing, the first variable name is taken. For polygons, the variable should contain densities, not absolute numbers.
`nrow`	number of rows in the raster that is used to smooth the shape object. Only applicable if shp is not a `SpatialGrid(DataFrame)` or `Raster`
`ncol`	number of rows in the raster that is used to smooth the shape object. Only applicable if shp is not a `SpatialGrid(DataFrame)` or `Raster`
`N`	preferred number of points in the raster that is used to smooth the shape object. Only applicable if shp is not a `SpatialGrid(DataFrame)` or `Raster`
`unit`	unit specification. Needed when calculating density values. When set to `NA`, the densities values are based on the dimensions of the raster (defined by `nrow` and `ncol`). See also `unit.size`.
`unit.size`	size of the unit in terms of coordinate units. The coordinate system of many projections is approximately in meters while thematic maps typically range many kilometers, so by default `unit="km"` and `unit.size=1000` (meaning 1 kilometer equals 1000 coordinate units).
`smooth.raster`	logical that determines whether 2D kernel density smoothing is applied to the raster shape object. Not applicable when `shp` is a `SpatialPoints` object (since it already requires a 2D kernel density estimator). Other spatial objects are converted to a raster, which is smoothed when `smooth.raster=TRUE`.

nlevels — preferred number of levels

style — method to cut the color scale: e.g. "fixed", "equal", "pretty", "quantile", or "kmeans". See the details in classIntervals.

breaks — in case style=="fixed", breaks should be specified

bandwidth — single numeric value or vector of two numeric values that specifiy the bandwidth of the kernal density estimator. By default, it is 1/50th of the shortest side in units (specified with unit.size).

threshold — threshold value when a 2D kernel density is applied. Density values below this threshold will be set to NA. Only applicable when shp is a SpatialPoints or smooth.raster=TRUE.

cover.type — character value that specifies the type of raster cover, in other words, how the boundaries are specified. Options: "original" uses the same boundaries as shp (default for polygons), "smooth" calculates a smooth boundary based on the 2D kernal density (determined by smooth_raster_cover), "rect" uses the bounding box of shp as boundaries (default for spatial points and grids).

cover — SpatialPolygons shape that determines the covered area in which the contour lines are placed. If specified, cover.type is ignored.

cover.threshold — numeric value between 0 and 1 that determines which part of the estimated 2D kernal density is returned as cover. Only applicable when cover.type="smooth".

weight — single number that specifies the weight of a single point. Only applicable if shp is a SpatialPoints object.

extracting.method — Method of how coordinates are extracted from the kernel density polygons. Options are: "full" (default), "grid", and "single". See details. For the slowest method "full", extract is used. For "grid", points on a grid layout are selected that intersect with the polygon. For "simple", a simple point is generated with gPointOnSurface.

buffer.width — Buffer width of the iso lines to cut kernel density polygons. Should be small enough to let the polygons touch each other without space in between. However, too low values may cause geometric errors.

to.Raster — not used anymore, since the "raster" output is always a RasterLayer as of version 2.0

Details

For the estimation of the 2D kernal density, code is borrowed from bkde2D. This implemention is slightly different: bkde2D takes point coordinates and applies linear binning, whereas in this function, the data is already binned, with values 1 if the values of var are not missing and 0 if values of var are missing.

This function supports sf objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal.

Value

List with the following items:

`"raster"` A smooth raster, which is either a `SpatialGridDataFrame` or a `RasterLayer` (see `to.Raster`)

`"iso"` Contour lines, which is an `sf` object of spatial lines.

`"polygons"` Kernel density polygons, which is an `sf` object of spatial polygons

`"bbox"` Bounding box of the used raster

`"ncol"` Number of rows in the raster

`"nrow"` Number of columns in the raster

Examples

```
## Not run:
if (require(tmap)) {
    # set mode to plotting mode
    current.mode <- tmap_mode("plot")

    ####################################
    ## Already smoothed raster
    ####################################
    vol <- raster::raster(t(volcano[, ncol(volcano):1]), xmn=0, xmx=870, ymn=0, ymx=610)
    vol_smooth <- smooth_map(vol, smooth.raster = FALSE, nlevels = 10)

    tm_shape(vol_smooth$polygons) +
     tm_fill("level", palette=terrain.colors(11), title="Elevation") +
     tm_shape(vol_smooth$iso) +
     tm_iso(col = "black", size = .7, fontcolor="black") +
     tm_layout("Maunga Whau volcano (Auckland)", title.position=c("left", "bottom"),
         inner.margins=0) +
     tm_legend(width=.13, position=c("right", "top"), bg.color="gray80", frame = TRUE)


    ####################################
    ## Smooth polygons
    ####################################
    data(NLD_muni)

    NLD_muni$population_dens <- as.vector(calc_densities(NLD_muni, "population"))

    qtm(NLD_muni, fill="population_dens")

    NLD_smooth <- smooth_map(NLD_muni, var = "population_dens")

    qtm(NLD_smooth$raster, style="grey")
    qtm(NLD_smooth$polygons, format="NLD")

    ####################################
    ## Smooth points
    ####################################
```

```
# Approximate world population density as spatial points, one for each 1 million people,
# in the following way. Each metropolitan area of x million people will be represented
# by x dots. The remaining population per country will be represented by dots that are
# sampled across the country.
create_dot_per_1mln_people <- function() {
 data(World, metro)
 metro_eck <- set_projection(metro, projection = "eck4")

 # aggregate metropolitan population per country
 metro_per_country <- tapply(metro_eck$pop2010, INDEX = list(metro_eck$iso_a3), FUN=sum)
 metro_per_country_in_World <- metro_per_country[names(metro_per_country) %in% World$isc

 # assign to World shape
 World$pop_metro <- 0
 World$pop_metro[match(names(metro_per_country_in_World), World$iso_a3)] <-
 metro_per_country_in_World

 # define population density other than metropolitan areas
 World$pop_est_dens_non_metro <- (World$pop_est - World$pop_metro) / World$area

 # generate dots for metropolitan areas (1 dot = 1mln people)
 repeats <- pmax(1, metro_eck$pop2010 %/% 1e6)
 ids <- unlist(mapply(rep, 1:nrow(metro_eck), repeats, SIMPLIFY = FALSE))

 metro_dots <- metro_eck[ids, ]


 # sample population dots from non-metropolitan areas (1 dot = 1mln people)
 World_pop <- sample_dots(World, vars="pop_est_dens_non_metro", w = 1e6,
  npop = 7.3e9 - nrow(metro_dots)*1e6)

 # combine

 c(st_geometry(World_pop), st_geometry(metro_dots))
}

World_1mln_dots <- create_dot_per_1mln_people()


# dot map
tm_shape(World_1mln_dots) + tm_dots()

# create smooth map
World_list <- smooth_map(World_1mln_dots, cover = World, weight=1e6)

# plot smooth raster map
qtm(World_list$raster, style="grey")

# plot smooth raster map
qtm(World, bbox="India") + qtm(World_list$iso)

# plot kernel density map
```

```
    qtm(World_list$polygons, style="grey", format="World")

    ####################################
    ## Smooth raster
    ####################################
    data(land)

    land_smooth <- smooth_map(land, var="trees", cover.type = "smooth")

    qtm(land, raster="trees")
    qtm(land_smooth$raster)
    qtm(land_smooth$polygons, format="World", style="grey")

    # reset current mode
    tmap_mode(current.mode)
}

## End(Not run)
```

`smooth_raster_cover` *Get a smoothed cover of a raster object (deprecated)*

Description

Get a smoothed cover of a raster object. From all non-missing values of a raster object, a 2D kernal density is applied. The output is a `sf` object of spatial polygons. Used by `smooth_map`. Note that this function supports `sf` objects, but still uses sp-based methods (see details). Note that this function supports `sf` objects, but still uses sp-based methods (see details).

Usage

```
smooth_raster_cover(shp, var = NULL, bandwidth = NA, threshold = 0.6,
  output = "polygons")
```

Arguments

`shp`	raster object, from either `SpatialGrid(DataFrame)` or `Raster` class.
`var`	name of the variable from which missing values are flagged. If unspecified, the first variable will be taken.
`bandwidth`	single numeric value or vector of two numeric values that specifiy the bandwidth of the kernal density estimator. See details.
`threshold`	numeric value between 0 and 1 that determines which part of the estimated 2D kernal density is returned as cover.
`output`	class of the returned object. One of: `polygons` (`sf` object), `lines` (`sf` object), or a `raster`. A vector of class names results in a list of output objects.

Details

For the estimation of the 2D kernal density, code is borrowed from `bkde2D`. This implemention is slightly different: `bkde2D` takes point coordinates and applies linear binning, whereas in this function, the data is already binned, with values 1 if the values of `var` are not missing and 0 if values of `var` are missing.

This function supports `sf` objects, but still uses sp-based methods, from the packages sp, rgeos, and/or rgdal.

`tmaptools-deprecated` *Deprecated functions in* `tmaptools`

Description

These functions are based on the `sp` package and are not supported anymore. They have been migrated to `https://github.com/mtennekes/oldtmaptools`

Details

`append_data`, `aggregate_map`, `double_line`, `points_to_raster`, `poly_to_raster`, `sample_dots`, `sbind`, `smooth_map`, `smooth_raster_cover`, `read_shape`, `write_shape`

`write_shape` *Write shape file (deprecated)*

Description

Write a shape object to an ESRI shape file.

Usage

```
write_shape(shp, file)
```

Arguments

`shp`	a shape object
`file`	file name (including directory)

Details

This function is a convenient wrapper of rgdal's `writeOGR`.

`%>%` *Pipe operator*

Description

The pipe operator from magrittr, `%>%`, can also be used in functions from `tmaptools`.

Arguments

`lhs`	Left-hand side
`rhs`	Right-hand side

Examples

```
## Not run:
if (require(tmap)) {
    data(land, World)

    current.mode <- tmap_mode("view")

    land %>%
        crop_shape(World[World$name=="China",], polygon = TRUE) %>%
        tm_shape() +
        tm_raster("cover_cls")

    tmap_mode(current.mode)
}

## End(Not run)
```