Package ‘tmaptools’

March 30, 2020

Type Package
Title Thematic Map Tools
Version 3.0
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 2020-03-30
Depends R (>= 3.0), methods
Imports sf (>= 0.9), lwgeom (>= 0.1-4), stars (>= 0.4-0), units (>= 0.6-1), grid, raster (>= 2.7-15), classInt, KernSmooth, magrittr, RColorBrewer, viridisLite, stats, dichromat, XML
Suggests tmap (>= 2.2), rmapshaper, osmdata, OpenStreetMap, png, shiny, shinyjs
URL https://github.com/mtennekes/tmaptools
BugReports https://github.com/mtennekes/tmaptools/issues
RoxygenNote 7.1.0
NeedsCompilation no
Author Martijn Tennekes [aut, cre]
Maintainer Martijn Tennekes <mtennekes@gmail.com>
Repository CRAN
Date/Publication 2020-03-30 15:00:02 UTC

R topics documented:

  tmaptools-package .................................................. 2
  approx_areas ......................................................... 3
### 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)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>approx_areas</td>
<td>Approximate area sizes of polygons</td>
</tr>
<tr>
<td>approx_distances</td>
<td>Approximate distances</td>
</tr>
<tr>
<td>bb</td>
<td>Create, extract or modify a bounding box</td>
</tr>
<tr>
<td>bb_poly</td>
<td>Convert bounding box to a polygon</td>
</tr>
<tr>
<td>get_asp_ratio</td>
<td>Get the aspect ratio of a shape object</td>
</tr>
</tbody>
</table>

#### Tool functions (colors)
approx_areas

approx_areas

get_brewer_pal
map_coloring
palette_explorer

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

Spatial transformation functions

crop_shape
simplify_shape

Crop shape objects
Simplify a shape

Input and output functions

geocode_OSM
read_GPX
read_osm
rev_geocode_OSM

Get a location from an address description
Read a GPX file
Read Open Street Map data
Get an address description from a location

Author(s)

Martijn Tennekes <mtennekes@gmail.com>

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)
approx_areas

Arguments

shp  
shape object, i.e., an \texttt{sf} or \texttt{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 \texttt{orig} for the coordinate units used by the shape \texttt{shp}.

total.area  
total area size of \texttt{shp} in number of target units (defined by \texttt{target}). Useful if the total area of the \texttt{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 \url{https://en.wikipedia.org/wiki/List_of_map_projections} for equal area world map projections.

Value

Numeric vector of area sizes (class \texttt{units}).

See Also

\texttt{approx_distances}

Examples

```r
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) {
```
approx_distances

```
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 %>% sf::st_transform(crs = 4326)) %>% 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 coordinates (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** and **y** is approximated.
- **y**: a pair of coordinates, vector of two. Only required when **x** is also a pair of coordinates.
- **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 specified, 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

```r
## 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 = 4326, target = "km")

  # Check distances in several projections
  sapply(c(3035, 28992, 4326), 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)
```
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.

projection projection to transform the bounding box to.

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 \texttt{raster::extent} object, which is a numeric vector of 4:
\texttt{xmin}, \texttt{xmax}, \texttt{ymin}, \texttt{ymax}. This representation used by the \texttt{raster} package.

Details

An existing bounding box (defined by \texttt{x}) can be modified as follows:

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

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

- Using the extension factor \texttt{ext}.
- Setting the center coordinates \texttt{cx} and \texttt{cy}, together with the \texttt{width} and \texttt{height}.
- Setting the \texttt{x} and \texttt{y} limits \texttt{xlim} and \texttt{ylim}

Value

bounding box (see argument \texttt{output})

See Also

\texttt{geocode\_OSM}

Examples

if (require(tmap) && packageVersion("tmap") >= "2.0") {
  ## load shapes
  data(NLD\_muni)
  data(World)

  ## get bounding box (similar to \texttt{sp}'s function \texttt{bbox})
  bb(NLD\_muni)

  ## extent it by factor 1.10
  bb(NLD\_muni, ext=1.10)

  ## convert to longlat
  bb(NLD\_muni, projection=4326)

  ## change existing bounding box
  bb(NLD\_muni, ext=1.5)
  bb(NLD\_muni, width=2, relative = \texttt{TRUE})
  bb(NLD\_muni, xlim=c(.25, .75), ylim=c(.25, .75), relative = \texttt{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

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

```r
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. 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 EPSG 4326.
calc_densities

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

```r
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 densities 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

```r
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 quality 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.

**Examples**

```r
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 <- sf::st_sf(data.frame(NLD_muni, NLD_muni_pop_per_km2))

  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**

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

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.
...

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

```r
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,
  geometry = c("point", "bbox"),
  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. If as.sf = TRUE (see below), as.data.frame will set to TRUE.

as.sf Return the output as sf object. If TRUE, return.first.only will be set to TRUE. Two geometry columns are added: bbox and point. The argument geometry determines which of them is set to the default geometry.

geometry When as.sf, this argument determines which column (bbox or point) is set as geometry column. Note that the geometry can be changed afterwards with st_set_geometry.

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
get_asp_ratio

Examples

```r
## Not run:
if (require(tmap)) {
  geocode_OSM("India")
  geocode_OSM("CBS Weg 1, Heerlen")
  geocode_OSM("CBS Weg 1, Heerlen", projection = 28992)

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, as.sf = TRUE)

  # 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

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

Arguments

- **x**: shape object (either `Spatial`, `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`. 
get_brewer_pal

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)
get_brewer_pal

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 categorical 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

```r
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_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.

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

```r
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) +
    tmBorders(lwd=2) +
  tm_text("name", shadow=TRUE) +
  tm_format("NLD", title="Dutch provinces and\nmunicipalities", bg.color="white")
}
```

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)

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", "routes", "tracks", "route_points", "track_points"),
  remove.empty.layers = TRUE,
  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 default, all those layers are read.

- **remove.empty.layers**
  should empty layers (i.e. with 0 features) be removed from the list?

- **as.sf**
  not used anymore

Details

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

Value

a list of sf objects, one for each layer

---

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, please 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.
**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**

```r
## 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="Purples")

  ### 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)
}
```
rev_geocode_OSM

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](http://wiki.openstreetmap.org/wiki/Nominatim_usage_policy)).

Usage

```r
rev_geocode_OSM(
  x,  
  y = NULL, 
  zoom = NULL, 
  projection = 4326, 
  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.
- **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 frame or a list with all attributes that are contained in the search result

See Also

gencode_OSM
Examples

```r
## 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 <- sf::st_sf(data.frame(five_cities, addresses))

  # 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)
```

---

**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`: an `sf` object.
- `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

```r
## 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)
```

Description

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

Arguments

- **lhs**: Left-hand side
- **rhs**: Right-hand side
Index

*Topic datasets
  palette_explorer, 18
*Topic densities
  calc_densities, 10
  %>%, 24
  approx_areas, 2, 3, 6, 11
  approx_distances, 2, 4, 5
  bb, 2, 5, 6, 9, 12–14, 20, 21
  bb_earth (bb_poly), 9
  bb_poly, 2, 9
  brewer.pal, 15
  brewer.pal.info, 18
  calc_densities, 10
  colortable, 21
  crop, 12
  crop_shape, 3, 11
  CRS, 13
  dichromat, 19
  display.brewer.pal, 16
  Extent, 7
  extent, 12
  geocode_OSM, 3, 7, 8, 12, 22
  get_asp_ratio, 2, 14
  get_brewer_pal, 3, 15, 19
  get_neighbours, 17
  get_proj4, 5, 13
  map_coloring, 3, 17
  ms_simplify, 23, 24
  openmap, 20, 21
  palette_explorer, 3, 15, 16, 18
  png, 15
  Raster, 7, 11, 12, 14
  raster, 21
  RColorBrewer, 19
  read_GPX, 3, 19
  read_osm, 3, 20
  rev_geocode_OSM, 3, 13, 22
  sf, 4, 7, 11–14, 17, 20, 22–24
  sfc, 9, 10
  simplify_shape, 3, 23
  sp, 4
  Spatial, 7, 11, 14
  SpatialPoints, 22
  SpatialPointsDataFrame, 13
  SpatialPolygons(DataFrame), 11, 17
  st_area, 3
  st_set_geometry, 13
  tmap, 14, 15
  tmap.pal.info (palette_explorer), 18
  tmaptools (tmaptools-package), 2
  tmaptools-package, 2
  units, 4