

# Package: tmaptools (via r-universe)

January 6, 2025

**Type** Package

**Title** Thematic Map Tools

**Version** 3.2

**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** 2025-01-03

**Depends** R (>= 3.5), methods

**Imports** sf (>= 0.9.2), lwgeom (>= 0.1-4), stars (>= 0.4-1), units (>= 0.6-1), grid, magrittr, RColorBrewer, viridisLite, stats, dichromat, XML

**Suggests** tmap, cols4all, rmapshaper, osmdata, OpenStreetMap, raster, png, shiny, shinyjs

**URL** <https://github.com/r-tmap/tmaptools>,  
<https://r-tmap.github.io/tmaptools/>

**BugReports** <https://github.com/r-tmap/tmaptools/issues>

**RoxygenNote** 7.3.2

**Config/pak/sysreqs** libgdal-dev gdal-bin libgeos-dev libxml2-dev  
libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

**Repository** <https://r-tmap.r-universe.dev>

**RemoteUrl** <https://github.com/r-tmap/tmaptools>

**RemoteRef** HEAD

**RemoteSha** 4d2385f77904c8d99bd9c467a15256ef7fc36cfe

## Contents

tmaptools-package	2
approx_areas	4
approx_distances	5
bb	7
bb_poly	9
calc_densities	11
crop_shape	12
geocode_OSM	13
get_asp_ratio	15
get_brewer_pal	16
get_neighbours	17
map_coloring	18
palette_explorer	19
read_GPX	20
read_osm	21
rev_geocode_OSM	22
simplify_shape	24
%>%	25
<b>Index</b>	<b>26</b>

---

tmaptools-package	<i>Thematic Map Tools</i>
-------------------	---------------------------

---

### 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)

<a href="#">approx_areas</a>	Approximate area sizes of polygons
<a href="#">approx_distances</a>	Approximate distances
<a href="#">bb</a>	Create, extract or modify a bounding box
<a href="#">bb_poly</a>	Convert bounding box to a polygon
<a href="#">get_asp_ratio</a>	Get the aspect ratio of a shape object

---

**Tool functions (colors)**

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

---

**Spatial transformation functions**

<code>crop_shape</code>	Crop shape objects
<code>simplify_shape</code>	Simplify a shape

---

**Input and output functions**

<code>geocode_OSM</code>	Get a location from an address description
<code>read_GPX</code>	Read a GPX file
<code>read_osm</code>	Read Open Street Map data
<code>rev_geocode_OSM</code>	Get an address description from a location

---

**Author(s)**

**Maintainer:** Martijn Tennekes <mtennekes@gmail.com>

**See Also**

Useful links:

- <https://github.com/r-tmap/tmertools>
- <https://r-tmap.github.io/tmertools/>
- Report bugs at <https://github.com/r-tmap/tmertools/issues>

---

approx_areas	<i>Approximate area sizes of the shapes</i>
--------------	---

---

### 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 <code>sf</code> or <code>sp</code> object.
target	target unit, one of <p>"prop": Proportional numbers. In other words, the sum of the area sizes equals one.</p> <p>"norm": Normalized numbers. All area sizes are normalized to the largest area, of which the area size equals one.</p> <p>"metric" (<b>default</b>): Output area sizes will be either "km" (kilometer) or "m" (meter) depending on the map scale</p> <p>"imperial": Output area sizes will be either "mi" (miles) or "ft" (feet) depending on the map scale</p> <p><b>other</b>: Predefined values are "km^2", "m^2", "mi^2", and "ft^2". Other values can be specified as well, in which case <code>total.area</code> is required).</p> <p>These units are the output units. See <code>orig</code> for the coordinate units used by the shape <code>shp</code>.</p>
total.area	total area size of <code>shp</code> in number of target units (defined by <code>target</code> ). Useful if the total area of the <code>shp</code> 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](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") >= "3.99") {
  data(NLD_muni)

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

  tm_shape(NLD_muni) +
    tm_bubbles(size="area",
              size.legend = tm_legend(title = 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 %>% 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 <code>bb</code> , 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 <a href="#">get_proj4</a>
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**

```
## 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)
```

---

**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,
  asp.limit = NULL,
  current.projection = NULL,
  projection = NULL,
  output = c("bbox", "matrix", "extent")
)
```

**Arguments**

- |     |   |
|-----|---|
| x   | <p>One of the following:</p> <ul style="list-style-type: none"> <li>• A shape from class <code>sf</code>, <code>stars</code>, <code>sp</code>, or <code>raster</code>.</li> <li>• A bounding box (<code>st_bbox</code>, <code>Extent</code> (raster package, which will no longer be supported in the future versions), numeric vector of 4 (default order: <code>xmin</code>, <code>ymin</code>, <code>xmax</code>, <code>ymax</code>), or a 2x2 matrix).</li> <li>• Open Street Map search query. The bounding is automatically generated by querying <code>x</code> from Open Street Map Nominatim. See <a href="#">geocode_OSM</a> and <a href="https://wiki.openstreetmap.org/wiki/Nominatim/">https://wiki.openstreetmap.org/wiki/Nominatim/</a>.</li> </ul> <p>If <code>x</code> is not specified, a bounding box can be created from scratch (see details).</p> |
| ext | <p>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 <code>width</code> and <code>height</code> with <code>relative=TRUE</code>. If a negative value is specified, then the shortest side of the bounding box (so <code>width</code> or <code>height</code>) is extended with <code>ext</code>, 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.</p>  |
| cx  | <p>center x coordinate</p>  |

<code>cy</code>	center y coordinate
<code>width</code>	width of the bounding box. These are either absolute or relative (depending on the argument <code>relative</code> ).
<code>height</code>	height of the bounding box. These are either absolute or relative (depending on the argument <code>relative</code> ).
<code>xlim</code>	limits of the x-axis. These are either absolute or relative (depending on the argument <code>relative</code> ).
<code>ylim</code>	limits of the y-axis. See <code>xlim</code> .
<code>relative</code>	boolean that determines whether relative values are used for <code>width</code> , <code>height</code> , <code>xlim</code> and <code>ylim</code> or absolute. If <code>x</code> is unspecified, <code>relative</code> is set to "FALSE".
<code>asp.limit</code>	maximum aspect ratio, which is width/height. Number greater than or equal to 1. For landscape bounding boxes, <code>1/asp.limit</code> will be used. The returned bounding box will have an aspect ratio between <code>1/asp.limit</code> and <code>asp.limit</code> .
<code>current.projection</code>	projection that corresponds to the bounding box specified by <code>x</code> .
<code>projection</code>	projection to transform the bounding box to.
<code>output</code>	output format of the bounding box, one of: <ul style="list-style-type: none"> <li>• "bbox" a <code>sf::bbox</code> object, which is a numeric vector of 4: <code>xmin</code>, <code>ymin</code>, <code>xmax</code>, <code>ymax</code>. This representation used by the <code>sf</code> package.</li> <li>• "matrix" a 2 by 2 numeric matrix, where the rows correspond to <code>x</code> and <code>y</code>, and the columns to <code>min</code> and <code>max</code>. This representation used by the <code>sp</code> package.</li> <li>• "extent" an <code>raster::extent</code> object, which is a numeric vector of 4: <code>xmin</code>, <code>xmax</code>, <code>ymin</code>, <code>ymax</code>. This representation used by the <code>raster</code> package.</li> </ul>

## 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 `relative` determines whether relative or absolute values are used.
- Setting the `x` and `y` limits. The argument `relative` 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 coordinates `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=4326)  
  
  ## 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 = 28992)  
  bb_italy <- bb("Italy", projection = "+proj=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 <code>bb</code>
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 <code>bb_earth</code> , 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.
bbx	boundig box of the earth in a vector of 4 values: min longitude, max longitude, min latitude, max latitude. By default <code>c(-180, 180, -90, 90)</code> . If for some projection, a feasible solution does not exist, it may be wise to choose a smaller bbx, e.g. <code>c(-180, 180, -88, 88)</code> . However, this is also automatically done with the next argument, <code>buffer</code> .
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 <code>1e-06</code> ). 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 <code>buffer=0</code> 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 density variables, which are often needed for choropleths. 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 <a href="#">sf</a> object or a <code>SpatialPolygons(DataFrame)</code> from the <code>sp</code> package.
var	name(s) of a quality variable name contained in the shp data
target	the target unit, see <a href="#">approx_areas</a> . Density values are calculated in $\text{var}/\text{target}^2$ .
total.area	total area size of shp in number of target units (defined by unit), <a href="#">approx_areas</a> .
suffix	character that is appended to the variable names. The resulting names are used as column names of the returned data.frame. By default, <code>_sq_&lt;target&gt;</code> , where target corresponds to the target unit, e.g. <code>_sq_km</code>
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**

```

if (require(tmap) && packageVersion("tmap") >= "3.99") {
  data(NLD_muni)

  NLD_muni_pop_per_km2 <- calc_densities(NLD_muni,
    target = "km km", var = c("population", "dwelling_total"))
  NLD_muni <- sf::st_sf(data.frame(NLD_muni, NLD_muni_pop_per_km2))

  tm_shape(NLD_muni) +
  tm_polygons(c("population_km.2", "dwelling_total_km.2"),
    fill.legend = list(tm_legend(expression("Population per " * km^2)), tm_legend(expression("Dwellings per " * km^2))),
  tm_facets(free.scales = TRUE) +
  tm_layout(panel.show = FALSE)
}

```

---

`crop_shape`*Crop shape object*

---

**Description**

Crop a shape object (from class `sf`, `stars`, `sp`, or `raster`). 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**

<code>x</code>	shape object, i.e. an object from class <code>sf</code> , <code>stars</code> , <code>sp</code> , or <code>raster</code> .
<code>y</code>	bounding box, an <code>st_bbox</code> , <code>extent</code> ( <code>raster</code> package), or a shape object from which the bounding box is extracted (unless <code>polygon</code> is <code>TRUE</code> and <code>x</code> is an <code>sf</code> object).
<code>polygon</code>	should <code>x</code> be cropped by the polygon defined by <code>y</code> ? If <code>FALSE</code> (default), <code>x</code> is cropped by the bounding box of <code>x</code> . Polygon cropping only works when <code>x</code> is a spatial object and <code>y</code> is a <code>SpatialPolygons</code> object.
<code>...</code>	not used anymore

**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") >= "3.99") {
  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",
             size.legend = tm_legend("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 (<https://operations.osmfoundation.org/policies/nominatim/>).

**Usage**

```

geocode_OSM(
  q,
  projection = NULL,
  return.first.only = TRUE,
  keep.unfound = FALSE,
  details = FALSE,
  as.data.frame = NA,
  as.sf = FALSE,
  geometry = c("point", "bbox"),
  server = "https://nominatim.openstreetmap.org"
)

```

**Arguments**

<code>q</code>	a character (vector) that specifies a search query. For instance "India" or "CBS Weg 11, Heerlen, Netherlands".
<code>projection</code>	projection in which the coordinates and bounding box are returned. See <a href="#">st_crs</a> for details. By default latitude longitude coordinates (EPSG 4326).
<code>return.first.only</code>	Only return the first result
<code>keep.unfound</code>	Keep list items / data.frame rows with NAs for unfound search terms. By default FALSE
<code>details</code>	provide output details, other than the point coordinates and bounding box
<code>as.data.frame</code>	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 <code>q</code> contains multiple queries. If <code>as.sf = TRUE</code> (see below), <code>as.data.frame</code> will set to TRUE.
<code>as.sf</code>	Return the output as <code>sf</code> object. If TRUE, <code>return.first.only</code> will be set to TRUE. Two geometry columns are added: <code>bbox</code> and <code>point</code> . The argument <code>geometry</code> determines which of them is set to the default geometry.
<code>geometry</code>	When <code>as.sf</code> , this argument determines which column ( <code>bbox</code> or <code>point</code> ) is set as geometry column. Note that the geometry can be changed afterwards with <a href="#">st_set_geometry</a> .
<code>server</code>	OpenStreetMap Nominatim server name. Could also be a local OSM Nominatim server.

**Value**

If `as.sf` then a `sf` object 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 = 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

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

### Arguments

x	A shape from class <a href="#">sf</a> , <a href="#">stars</a> , <a href="#">sp</a> , or <a href="#">Raster</a> , a bounding box (that can be coerced by <a href="#">bb</a> ), or a <a href="#">tmap</a> 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 <a href="#">tmap</a> object.
height	See details; only applicable if x is a <a href="#">tmap</a> object.
res	See details; only applicable if x is a <a href="#">tmap</a> 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 <a href="#">palette_explorer</a> or see <a href="#">brewer.pal</a> 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 <code>contrast=c(.25, .75)</code> , 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**

```
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)` (`sp` package).

**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 <code>sf</code> or <code>SpatialPolygons(DataFrame)</code> ( <code>sp</code> package) object), or an adjacency list.
algorithm	currently, only "greedy" is implemented.
ncols	number of colors. By default it is 8 when <code>palette</code> is undefined. Else, it is set to the length of <code>palette</code>
minimize	logical that determines whether <code>algorithm</code> will search for a minimal number of colors. If <code>FALSE</code> , the <code>ncols</code> 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 <code>auto.palette.mapping=TRUE</code> ). 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") >= "3.99") {
  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",
         fill.legend = tm_legend_hide()) +
  tm_shape(NLD_muni) +
  tm_polygons("MAP_COLORS",
             fill_alpha = .25,
             fill.scale = tm_scale(values = "brewer.greys")) +
  tm_shape(NLD_prov) +
  tm_borders(lwd=2) +
  tm_text("name", options = opt_tm_text(shadow = TRUE)) +
  tm_title("Dutch provinces and\nmunicipalities", bg.color="white")
}

```

---

palette\_explorer

*Explore color palettes (deprecated)*


---

**Description**

This interactive tool has become deprecated and will not be maintained anymore. Please use [c4a\\_gui](#) instead.

**Usage**

```
palette_explorer()
```

```
tmap.pal.info
```

**Format**

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

**Details**

`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 contrast 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.

**References**

<https://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)
```

---

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", "routes", "tracks", "route_points", "track_points"),
  remove.empty.layers = TRUE,
  as.sf = TRUE
)
```

**Arguments**

<code>file</code>	a GPX filename (including directory)
<code>layers</code>	vector of GPX layers. Possible options are "waypoints", "tracks", "routes", "track_points", "route_points". By default, all those layers are read.
<code>remove.empty.layers</code>	should empty layers (i.e. with 0 features) be removed from the list?
<code>as.sf</code>	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	<i>Read Open Street Map data</i>
----------	----------------------------------

---

### 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,
  ...
)
```

### Arguments

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

### 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="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)

}

## End(Not run)
```

---

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 (<https://operations.osmfoundation.org/policies/nominatim/>).

## Usage

```
rev_geocode_OSM(
  x,
  y = NULL,
  zoom = NULL,
```

```

    projection = 4326,
    as.data.frame = NA,
    server = "https://nominatim.openstreetmap.org"
  )

```

### Arguments

x	x coordinate(s), or a spatial points object ( <a href="#">sf</a> or <a href="#">SpatialPoints</a> )
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

[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 <- 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	<i>Simplify shape</i>
----------------	-----------------------

---

### 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 <i>sf</i> or <i>sfc</i> object.
fact	simplification factor, number between 0 and 1 (default is 0.1)
keep.units	prevent small polygon features from disappearing at high simplification (default FALSE)
keep.subunits	should multipart polygons be converted to singlepart polygons? This prevents small shapes from disappearing during simplification if keep.units = TRUE. Default FALSE
...	other arguments passed on to the underlying function <a href="#">ms_simplify</a> (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)
```

---

%>%

*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

# Index

- \* **GIS**
  - tmertools-package, 2
- \* **datasets**
  - palette\_explorer, 19
- \* **densities**
  - calc\_densities, 11
- \* **spatial data**
  - tmertools-package, 2
- \* **thematic maps**
  - tmertools-package, 2
- %>%, 25
- approx\_areas, 2, 4, 6, 11
- approx\_distances, 2, 5, 5
- bb, 2, 6, 7, 10, 13–15, 21
- bb\_earth (bb\_poly), 9
- bb\_poly, 2, 9
- brewer.pal, 16
- brewer.pal.info, 19
- c4a\_gui, 19
- calc\_densities, 11
- colortable, 21
- crop\_shape, 3, 12
- dichromat, 20
- geocode\_OSM, 3, 7, 9, 13, 23
- get\_asp\_ratio, 2, 15
- get\_brewer\_pal, 3, 16, 20
- get\_neighbours, 17
- get\_proj4, 6
- map\_coloring, 3, 18
- ms\_simplify, 24
- openmap, 21
- palette\_explorer, 3, 16, 17, 19
- png, 15
- raster, 21
- RColorBrewer, 20
- read\_GPX, 3, 20
- read\_osm, 3, 21
- rev\_geocode\_OSM, 3, 14, 22
- sf, 4, 7, 8, 11, 12, 14, 15, 17, 18, 20, 23, 24
- sfc, 9, 10, 24
- simplify\_shape, 3, 24
- SpatialPoints, 23
- st\_area, 4
- st\_bbox, 7, 12
- st\_crs, 14
- st\_set\_geometry, 14
- stars, 7, 12, 15
- tmap, 15
- tmap.pal.info (palette\_explorer), 19
- tmertools (tmertools-package), 2
- tmertools-package, 2
- units, 4