



## tmap: Thematic Maps in R

Martijn Tennekes  
Statistics Netherlands

---

### Abstract

Thematic maps show spatial distributions. The theme refers to the phenomena that is shown, which is often demographical, social, cultural, or economic. The best known thematic map type is the choropleth, in which regions are colored according to the distribution of a data variable. The R package **tmap** offers a coherent plotting system for thematic maps that is based on the *layered grammar of graphics*. Thematic maps are created by stacking layers, where per layer, data can be mapped to one or more aesthetics. It is also possible to generate small multiples. Thematic maps can be further embellished by configuring the map layout and by adding map attributes, such as a scale bar and a compass. Besides plotting thematic maps on the graphics device, they can also be made interactive as an HTML widget. In addition, the R package **tmaptools** contains several convenient functions for reading and processing spatial data.

*Keywords:* thematic maps, spatial data, R.

---

## 1. Introduction

Visualization is important in statistics and data science. Without looking at the data, it is difficult to understand the underlying phenomena, to unveil anomalies, and, moreover, to extract valuable knowledge (Tufte 1983). Software tools to visually explore, analyze, and present data are therefore greatly needed. The R environment (R Core Team 2017) and its extension packages contain many possibilities to craft elegant and clarifying graphics, most notably package **ggplot2** (Wickham 2009). Although it is possible to achieve great looking geospatial visualizations in R, the required programming is often complicated and the available functions are often limited in terms of features and flexibility. Therefore, we introduce the **tmap** package (Tennekes 2018a), which offers a flexible, easy to use, and layer-based approach to crafting thematic maps. Package **tmap** is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=tmap>.

Thematic maps are geographic maps in which spatial data distributions are shown. Although

they are widely used for publication purposes, which is no surprise due to their visual appeal and recognizability, they also proved to be successful for the discovery and exploration of spatial data (Friendly 1995). One of the most illustrative examples is the dot map that physician John Snow used to locate the source of the cholera outbreak in London in 1854 (Snow 1855). Spatial data analysis is also key in many of today's global issues related to geography, for instance those concerning climate change (IPCC 2014), health (WHO 2016), and sustainable development (UN 2016). Visual exploration and presentation of the underlying spatial data contributes to acquire knowledge about these issues and share this with the general public.

### 1.1. Layered grammar of thematic maps

Thematic maps are often characterized and named by the specific technique by which they are created (Slocum, MacMaster, Kessler, and Howard 2009; Tyner 2010), such as a choropleth or a dot map. Although this may be a good starting point to study thematic cartography, other characterizations provide a more systematic approach that provide a better understanding of the underlying structure of thematic maps (Bertin 1983; Kraak and Ormeling 2010). The grammar of graphics (Wilkinson 2005) is an extension of this approach and can be applied to any statistical graphic. The layered grammar of graphics (LGG) is a further refinement and has been implemented in the R package `ggplot2` (Wickham 2009, 2010). In this section, we introduce the *layered grammar of thematic maps* (LGTm), which is a variation of the LGG that is tailored to spatial data visualization and which has been implemented in package `tmap`.

First, we enumerate the most commonly used thematic map types (Slocum *et al.* 2009; Tyner 2010), which will give an impression of the large diversity of thematic maps. Subsequently, we show how thematic maps can also be described by the LGTM.

**Choropleth:** Areal regions, such as countries or municipalities, are filled with colors that represent a variable which is either a density or a ratio. The usage of class intervals encourages the readability of the data values. A good starting point is to use five or seven classes based on natural breaks (MacEachren 1994) or quantiles (Brewer and Pickle 2002).

**Categorical map:** Areal regions are filled with colors that represent a categorical data variable. Categorical maps are often confused with choropleths.

**Proportional symbol map:** Data points are encoded by symbols that are scaled in proportion to a variable (Slocum *et al.* 2009). Bubbles are the most widely used symbols. The corresponding map is called a *bubble map*.

**Isopleth or contour map:** Contour lines are drawn through equally valued data points. Since data points are often sampled, these lines often have to be approximated according to a two-dimensional kernel density estimator. This method is also useful when the data distribution is not smooth, which may result in an occlusion of spiky contour lines. Smoothing spatial data with a two-dimensional kernel density estimator may result in smooth contour lines that provide a better understanding in the overall patterns of the data distribution. Isopleths are commonly used in elevation and weather maps.

**Dasymetric map:** Similar to the choropleth, except that the polygons are not predefined areal areas, but defined by levels of the variable that is encoded by color. The advantage

over the choropleth is that the dasymetric map portrays the underlying data distributions in a better way (Langford and Unwin 1994). The dasymetric map can be seen as a hybrid between the choropleth and the isopleth.

**Dot map:** Data points are shown as dots, like the cholera occurrences in John Snow’s map. Although the dot map can technically be seen as a bubble map with bubbles of fixed, very small, size, they are conceptually different. The dot map is based on spatial points rather than areas. Generally, dot maps contain much more objects than bubble maps.

**Raster map:** Square grid cells are colored according to the values of a data variable. Raster maps are especially popular in geology and ecology, where gridded land cover and land use data are often visualized (Fisher, Comber, and Wadsworth 2005).

**Cartogram:** Areal regions are distorted such that the obtained area sizes are proportional to a quantitative variable (Gastner and Newman 2004; Nusrat and Kobourov 2016). In a contiguous area cartogram, the shapes of the polygons are distorted, where the neighborhood relationships between them are preserved as much as possible. In a non-contiguous cartogram, the polygons are only resized. Categorical maps and choropleths of ratio variables are often misleading, since these maps do not take the population distribution into account properly. Densely populated regions are often underemphasized by their small areas. The project <http://www.worldmapper.org/> contains a couple of illustrative examples, for instance a distorted world map in which the areas of the countries are proportional to their population sizes (Barford and Dorling 2008).

The LGG is a tool to describe a statistical graphic. Data variables are mapped to graphical variables, such as position, size, color, and shape. These graphical variables are called aesthetics. A graphic consists of one or more layers, for example, a scatter plot with a smoothed line. Each layer is defined by a geometry and optionally a statistical transformation. For the scatter plot, the geometry is a point where no statistical transformation is applied, i.e., each data record corresponds to a point in the graphic. For the smoothed line, the geometry is a line and the statistical transformation is a smoothing function. A scale is defined for each aesthetic, for instance, a linear scale for the position aesthetics and a sequential color scale for the color of the points. The coordinate system determines how the position aesthetics are mapped. Finally, small multiples are graphics that are created for different subsets of the data. These small multiples, which are also called facets, trellis charts, or lattice charts, are usually placed on a rectangular grid.

Although the LGG can be directly applied to thematic maps, it can be tailored to thematic maps in order to make it more practical, especially regarding the coordinates. Whereas the coordinates of a scatter plot are determined by mapped data, the coordinates of a bubble map are determined by geographic locations. In the context of geospatial visualization, the coordinate system is called the *coordinate reference system*, which is usually referred to as the map projection. It determines how longitude-latitude coordinates are projected from the earth surface to a two-dimensional plane (Slocum *et al.* 2009; Tyner 2010). We introduce the LGTM to accommodate this specific focus on geospatial visualization.

The LGTM is built up as follows:

**Shape:** We refer to a *shape* as an object that consists of the following components:

**Coordinates and topology:** The coordinates and topology are required to define spatial units. Four basic types can be distinguished: polygons, lines, points, and raster cells. A spatial unit may consist of multiple polygons or lines. This is the case when areal regions are non-contiguous, for instance a country with islands.

**Data:** The format of the data is tabular, where the rows correspond to the spatial units and the columns to the data variables.

**Map projection:** The map projection defines how longitude-latitude coordinates are projected.

**Bounding box:** The bounding box determines which part of the map is shown.

**Spatial transformation:** The spatial transformation determines whether and how the shape is transformed. It may be related to the coordinates/topology, to the data, or both. Three common examples are (1) a cartogram, where polygons are transformed according to the values of a numeric data variable, (2) a two-dimensional kernel density map, where a two-dimensional kernel density estimator is applied to a shape consisting of points, and (3) a raster map, where polygons or spatial points are converted to a spatial raster, along with the corresponding data.

**One or more layers:** Each layer consists of the following components:

**Geometry:** The geometry determines the spatial geometry that is shown, for instance polygons, symbols, or text labels. The availability of the geometries depends on the type of spatial units in the shape object. Obviously, polygon units can be visualized with a polygon geometry, but also with a symbol geography by using the polygon centroids as coordinates. The reverse is not possible, i.e., point units cannot be visualized with a polygon geometry.

**Mapping:** The mapping specifies which data variables are mapped to the available aesthetics. The set of aesthetics depends on the chosen geometry. For example, the available aesthetics for symbols are color, size, and symbol shape.

**Scaling:** A scale determines how the data values are translated to graphical attribute values. For the most common aesthetic in thematic maps, color, the scale determines what color palette is used and how the data values are mapped to this palette.

**Faceting:** Small multiples are created by subsetting the data, or by specifying more than one data variable for one aesthetic.

**Map attribute layers:** Additional components can be added to the thematic map, for instance a map compass and coordinate grid lines.

A shape object and one or more layers form a *group*. When multiple groups are stacked, only one map projection and bounding box can be used. By default, these will be taken from the shape specified in the first group.

A thematic map of the world that has been created with package **tmap** is depicted in Figure 1 to illustrate the LGTM. It shows the relation between level of income per country and the distribution of emerging metropolitan areas. While the metropolitan areas in high income countries have an annual growth rate that is less than two percent, which is lower than the

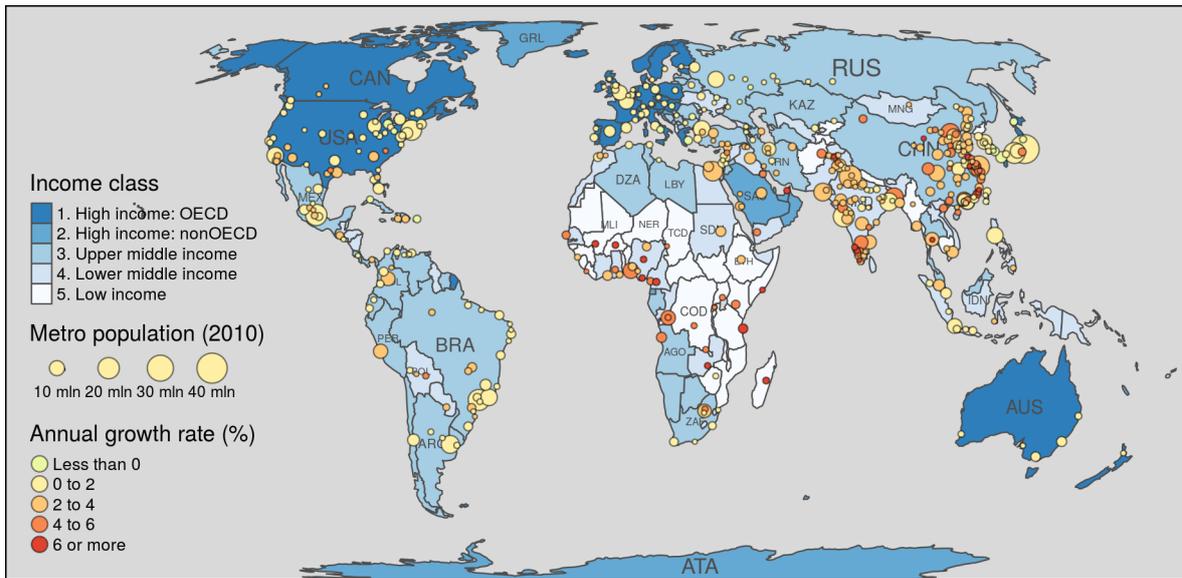


Figure 1: World map created with package **tmap**. The bottom layer is a choropleth of income class on country level. The top layer is a bubble map of metropolitan areas where the size of the bubbles corresponds to population and the color to the annual growth rate.

world population growth rate of 2.7 percent, metropolitan areas in lower income countries grow rapidly, especially in Asia but also western and middle Africa.

The LGTM description of the map in Figure 1 is the following. It contains two groups. The first one uses a shape with country polygons. These polygons are also drawn with the polygon geometry where income data is mapped to the fill color aesthetic using a sequential color palette of blues. The second group is specified with a shape of spatial points that correspond to the geographic centers of the metropolitan areas. For the second group, a bubble geometry is used, where population is mapped to size and annual growth rate to color, using a diverging color palette from green via yellow to red.

Although the map in Figure 1 may aesthetically look pretty and does contain a lot of information, there is one note of caution from a methodological point of view. The bubble map and the choropleth interfere with each other perceptually. For instance, the yellow bubbles may be perceived differently against a dark blue background than against a light blue background. In color theory, this phenomenon is known as simultaneous contrast (Whittle 2003). Although this may still be a useful map, especially for illustrating the **tmap** package, it is better to have a bubble map with a uniform background to analyze the distribution of metropolitan areas.

## 1.2. The **tmap** package

The aim of the **tmap** package is to provide R users with an elegant and flexible way to make thematic maps, requiring a minimum amount of code. All of the thematic map types enumerated in Section 1.1 can be applied with package **tmap**. However, package **tmap** is not a collection of stand-alone functions, one for each thematic map type, but an implementation of the LGTM. The syntax of **tmap** resembles the syntax of **ggplot2**.

The following code is the core part of the code to reproduce the map depicted in Figure 1

and illustrates the basics of the LGTM and its implementation in **tmap**.

```
R> tm_shape(World) + tm_polygons(col = "income_grp") +
+   tm_text(text = "iso_a3", size = "AREA") +
+   tm_shape(metro) + tm_bubbles(size = "pop2010", col = "growth")
```

The objects `World` and `metro` are the shape objects of world country borders and metropolitan area centers respectively. The functions `tm_polygons`, `tm_text`, and `tm_bubbles` define the layers. The geometry follows from the function names, in this example polygons, text, and bubbles respectively. The mappings are specified by the arguments, where the argument names correspond to the aesthetics and the values to the mapped data variables, except for "AREA", which is a fixed value for polygon area size. The scales are not specified in this code, hence default scale settings are used. The **tmap** package facilitates two modes to create thematic maps: the "plot" mode is used to create static maps and the "view" mode to create interactive maps. The code to create the maps in both modes is identical. The syntax of **tmap** will be discussed in more detail in Section 3. The complete code to reproduce the map depicted in Figure 1 will be provided in Section 4.1.

The **tmap** package stands on the shoulders of the following R packages.

**sp** (Pebesma and Bivand 2005; Bivand, Pebesma, and Gómez-Rubio 2008): This package contains classes and methods for spatial data.

**raster** (Hijmans 2017): This package contain classes and methods for rastered spatial data.

**rgdal** (Bivand, Keitt, and Rowlingson 2017): Reading and writing ESRI shape files is done with this package, as well as map projections via the **PROJ.4** library (Warmerdam and Evenden 2015).

**rgeos** (Bivand and Rundel 2017): All geometric processing is carried out by this package.

**classInt** (Bivand 2017): This package facilitates class intervals, which are used to map data variables to classes.

**RColorBrewer** (Neuwirth 2014): All used color schemes have been proposed by Brewer, Hatchard, and Harrower (2003) and have been implemented in this package.

**grid** (R Core Team 2017): This package provides the graphics system that **tmap** relies on.

**leaflet** (Cheng, Karambelkar, and Xie 2017): This package is an R interface to the JavaScript library **Leaflet** (Agafonkin 2017) for interactive maps.

The **tmap** package also uses **OpenStreetMap** (Fellows 2016) and **osmar** (Eugster and Schlesinger 2013) for obtaining OpenStreetMap data. For facilitating interactive maps, **tmap** not only uses **leaflet**, but also **mapview** (Appelhans, Detsch, Reudenbach, and Woellauer 2017) for interactive small multiples, **htmlwidgets** (Vaidyanathan, Xie, Allaire, Cheng, and Russell 2016) for further processing interactive maps, and **htmltools** (RStudio Inc. 2017) for processing HTML. Finally, the package **spdep** (Bivand, Hauke, and Kossowski 2013; Bivand and Piras 2015) is used to find neighboring polygons.

The supplementary R package **tmaptools** (Tennekes 2018b) contains several convenient functions to read and process spatial data. Although there are already many R packages for these

tasks (Bivand *et al.* 2008), the aim of **tmaptools** is to supply the workflow for creating thematic maps that should be sufficient for most users, e.g., reading a shapefile, appending external data, and exploring color palettes. These functions are further discussed in Section 3.4. Many of these functions are also used by **tmap**. However, the user does not need to use **tmaptools** explicitly to make thematic maps with **tmap**.

## 2. Related software

Dedicated desktop geographic information system (GIS) software is supposedly the most straightforward choice to analyze and visualize spatial data for most users. **ArcGIS** (ESRI 2016) is considered as the industry standard of desktop GIS software, whereas **QGIS** (QGIS Development Team 2016) and **GRASS** (GRASS Development Team 2016) are widely used free and open source alternatives (Steiniger and Bocher 2009; GIS Geography 2016). Desktop GIS software packages provide easy to use graphical user interfaces and contain standard, ready to use tools for processing and analyzing spatial data. The programming languages R and Python (Van Rossum *et al.* 2011) offer more freedom to analyze spatial data and also provide functionalities to visualize spatial data (Bivand *et al.* 2008; Westra 2015). However, a major drawback in comparison to desktop GIS software is that the learning curve is steep for users without any programming experience. Also, the ability to visualize spatial data with R and Python is considered moderate in comparison to desktop GIS software, because the workflow to create geographic visualizations is often cumbersome, the created visualizations are usually not interactive, and the ability to design and embellish thematic maps is often limited. However, the aim of the **tmap** package is to tackle these shortcomings.

Another programming language besides R and Python that is useful to create geospatial visualizations is JavaScript, especially with the libraries **D3** (Villarreal 2014) and **Leaflet** (Agafonkin 2017). With these libraries, maps can be made interactive and embedded in web pages. However, users will have to know the basics of JavaScript, HTML, and CSS. Moreover, JavaScript is less suitable for exploratory spatial data visualization, since data processing is usually more cumbersome than with R or Python.

There are a couple of other ways by which thematic maps can be created in R. The **sp** package contains a plotting method for spatial objects using the generic plot function. Creating thematic maps in this way is completely do-it-yourself regarding data processing, statistical transformation, and map layout. It usually takes some effort and requires many lines of code to create nice-looking thematic maps (Lovelace 2015).

Thematic maps can also be created with package **ggplot2**. Although the result can be very satisfactory, there are two obstacles to overcome. First, shape objects have to be converted to **data.frames** with the function **fortify**, which is prohibitive for data processing purposes since the rows in the obtained **data.frames** no longer correspond to statistical units anymore, which are represented by polygons, points, lines, or raster cells. Second, the layout is not optimized for maps. For instance, coordinate grid panels are visualized by default, and the legend is shown outside of the map viewport, while in many maps there is space for it in at least one corner.

The R packages **rworldmap** (South 2011), **GISTools** (Brunsdon and Chen 2014), and **choroplethr** (Lamstein and Johnson 2017) provide high level functions that are tailored to plot spatial data. However, the scope of features that these packages offer is limited. For instance

they do not support multiple layers and small multiples. They are also less flexible regarding the map layout.

The **maps** package provides low level plotting functions that are similar to those of **sp**, so users have to create class intervals and color palettes by themselves. Moreover, only shapes included in the package itself can be used.

Another R package that can be used to create thematic maps is **cartography** (Giraud and Lambert 2016). Like **tmap**, it offers a layer-based approach and it is very flexible regarding the map layout. This package uses base graphics, whereas **tmap** uses **grid**-based graphics. For each implemented thematic map type, which are the choropleth, the proportional symbol map, the categorical map, the flow map, and the discontinuities map<sup>1</sup>, a map can be created with a stand-alone function. The maps can be stacked as layers by setting the **add** argument of the functions that create the add-on maps to **TRUE**. However, it assumes that the map projections of the shapes are identical, since otherwise the add-on maps will be misplaced. In comparison to **tmap**, **cartography** does not support small multiples nor interactive maps.

Several R packages use maps from Google Maps or OpenStreetMap as base layer to plot spatial data. Maps made with **ggplot2** can be enhanced by adding such as base layer with the **ggmap** package (Kahle and Wickham 2013; Lovelace 2015). Alternatively, static thematic maps with a base layer can be created with package **RgoogleMaps**, which facilitates small multiples when combined with the **loa** package (Loecher and Ropkins 2015).

There are three downsides to using basemaps for thematic maps. First, since basemaps are usually projected according to the web Mercator projection, the created thematic maps are also limited to this projection. Although the web Mercator projection is useful for navigation purposes, it is often not the best projection for static thematic maps (Slocum *et al.* 2009). In particular, thematic maps that deal with densities, such as choropleths and dot maps, are best suited with map projections where areas are preserved. Second, for many thematic map types, detailed and colorful basemaps can be distracting, especially in combination with space-filling thematic map types like choropleths. Moreover, colored basemaps may influence our color perception of the map layers (Whittle 2003). Grayscale maps, which are used by **ggmap**, have less interference with the thematic map drawn on top, and are therefore better suited. Third, the thematic map layer may occlude the content of the basemap tiles. This problem can be partly solved by using semi-transparent layers. Another solution is to use interactive maps where the layers can be easily turned on and off.

For interactive maps, **tmap** uses the R package **leaflet**, an interface for the JavaScript library **Leaflet** (Agafonkin 2017). Interactive thematic maps can also directly be made with the R package **leaflet**. The main advantage of using **leaflet** directly is that it offers more control to design the maps. However, the plotting functions are of lower level than those of **tmap** and therefore require more user code. The R package **mapview** also provides high level functions for the R package **leaflet**. In comparison to the interactive output of **tmap**, **mapview** has some extra interactive tools, such as a function to compare two raster shapes and a function to view a raster shape as a three-dimensional cube interactively. Both **mapview** and **tmap** are useful for spatial data exploration, although the emphasis is a little different. The **mapview** package offers more features regarding interactive mapping, such as a home button per layer and popup charts. Its function to view small multiples interactively, **latticeView**, has been

---

<sup>1</sup>A discontinuity map shows borders between neighboring polygons. The line widths of these borders reflect the ratio between data values of neighboring polygons.

imported by **tmap**. On the other hand, **tmap** offers a more systematic approach to create thematic maps. One of the design principles of **tmap** is that the interactive maps look similar as their static counterparts, so that users are able to switch between interactive and static mapping without additional effort.

Interactive maps can also be created in R with the **plotGoogleMaps** package (Kilibarda and Bajat 2012). Spatial data can be plotted as layers in Google Maps, which is opened in a browser. With **plotGoogleMaps**, spatio-temporal data can be visualized interactively as small multiples. The main advantage of using Google Maps as mapping service is that it is very popular and therefore familiar to most people. However, a downside is that Google Maps have a fixed style that is often less suitable for thematic maps. OpenStreetMap, on the other hand, has many third-party tile servers, of which some are well suited for thematic maps because of the use of neutral color schemes, in particular **CARTO** (CartoDB, Inc. 2017). Interactive maps made with **plotGoogleMaps** can only be opened in a browser, whereas the R packages **leaflet**, **mapview**, and **tmap** create HTML widgets (Vaidyanathan *et al.* 2016), which can also be opened in the viewer pane in **RStudio** (RStudio Team 2017).

### 3. Syntax

The syntax of **tmap** is briefly discussed in this section and will be illustrated with examples in Section 4. The method of stacking layers will be referred to as the *main plotting method*. Thematic maps can also be easily made with one function, **qtm**, which stands for quick thematic map. Drawing maps with this function will be referred to as the *quick plotting method*.

The methods are interchangeable, so each map created with the main plotting method can also be created with the quick plotting method and vice versa. As a rule of thumb, **qtm** is recommended for quick plotting, whereas the main plotting method is recommended for plotting maps that require more control.

#### 3.1. Plotting and interactive viewing

The **tmap** package contains two drawing modes: `plot` and `view`.

In the default mode, `plot`, thematic maps are drawn in the graphics device. It encompasses all features that will be described in Sections 3.2 and 3.3, and in more detail in the package documentation. The output of the "plot" mode can be saved with the function `save_tmap` to static formats, such as JPG, PNG, SVG, and PDF.

In the interactive viewing mode, thematic maps are shown as interactive HTML widgets (Vaidyanathan *et al.* 2016). Within **RStudio**, these are shown in its viewer pane, and otherwise, in a web browser. Users are able to zoom, pan, and click on the spatial units. Therefore, the interactive mode is especially useful to explore dense spatial data. However, there are a few downsides in comparison to the plotting mode. First of all, maps will be projected to the web Mercator projection, since the vast majority of basemap tiles have been rendered accordingly. In this projection, areas are not preserved, which makes it less suitable for thematic maps. Second, the options to embellish the map layout and to add map attributes are limited. Third and last, although small multiples can be shown interactively, the number of small multiples in "view" mode is restricted to four (by default) in order to prevent slow interaction.



Figure 2: Forecasted metropolitan population sizes.

Interactive maps can be exported to standalone HTML files with `save_tmap`. It is also possible to embed the HTML widgets in dynamic documents made with `rmarkdown` (Allaire, Cheng, Xie, McPherson, Chang, Allen, Wickham, Atkins, Hyndman, and Arslan 2017).

The mode is stored in a global option named `tmap.mode`. It can be changed with the function `tmap_mode`, which is to be called with either "plot" or "view". For convenience, the function `tmap`, called without arguments, is used to toggle between the two modes.

### 3.2. Main plotting method

In R, a shape is usually an object of class 'Spatial' or 'Raster' from packages `sp` and `raster` respectively. Simple features are the next generation class of spatial objects and are currently being implemented in the `sf` package (Pebesma 2017). These 'Spatial', 'Raster', and 'sf' objects contain the four elements described in Section 1.1, namely the coordinates/topology, the data, the map projection, and the bounding box.

The main plotting method is applied in the following code. It produces the map illustrated in Figure 2 where the metropolitan areas, contained in the shape object `metro`, are displayed as bubbles that are sized according to the data variable `pop2030`, which contains the forecasted population numbers for the year 2030. The layout of the map is configured with predefined settings for world maps. Furthermore, a color scheme based on cobalt blue is applied.

```
R> data("metro", package = "tmap")
R> tm_shape(metro) + tm_bubbles(size = "pop2030") + tm_format_World() +
+   tm_style_cobalt()
```

We define an *element* as a building block to create thematic maps. Element functions, whose names have the prefix `tm_`, are stacked with the `+` operator. The most fundamental element function of the main plotting method, `tm_shape`, specifies the shape, the desired map projection, and bounding box (Table 1). The map projection can take characters specified according to the **PROJ.4** syntax (Warmerdam and Evenden 2015), EPSG projection numbers (IOGP 2015), or shortcut characters for commonly used map projections, such as "longlat" and "wintri" for longitude-latitude coordinates<sup>2</sup> and the Winkel tripel projection respectively.

<sup>2</sup>The shortcut "longlat" corresponds to the longitude-latitude definition using the WGS84 geodetic datum.

Function	Argument	Description	Spatial types
<code>tm_shape</code>	<code>shp</code>	shape object	polygons points lines raster cells
	<code>projection</code>	projection code	
	<code>bbox</code>	bounding box	

Table 1: Overview of the function `tm_shape`.

The function `tm_shape` should always be preceded by one or more layers, which are created by the functions provided in Table 2. Each of these functions specifies the geometry, mapping, and scaling component of the LGTM. An aesthetic can take a constant value, a data variable name, or a vector consisting of values or variable names. Variable names are specified through standard evaluation, i.e., by character strings, as opposed to non-standard evaluation, which is used by `ggplot2`. If a data variable is provided, the scale is automatically configured according to the values of this variable, but can be adjusted with several arguments. For instance, the main scaling arguments for a color aesthetic are color palette, the preferred number of classes, and a style to create classes. Also, for each aesthetic, except for the text labels, a legend is automatically created. If a vector of variable names is provided, small multiples are created, which will be explained further below.

Notice that the five base layer functions listed in Table 2 correspond to the basic spatial unit types, except for the text labels. In other words, spatial polygons are drawn with `tm_polygons`, spatial points with `tm_symbols`, spatial lines with `tm_lines`, and a spatial raster with `tm_raster`. Each derived layer function is composed of one or two base layer functions with different default settings.

The base layer `tm_symbols` accepts three types of symbol shapes: character symbols, icons, and ‘`grob`’ objects. Icons can be created from PNG images with the function `tmap_icons`. Visualizations created with the `grid` package can be exported to ‘`grob`’ objects. This is not only useful for creating vector based icons, but also for charts created with the `ggplot2` package. The result is a map with inset charts, which can be scaled with the `size` aesthetic. An example is included in the documentation page of `tm_symbols`.

One `tm_shape` element together with one or more layers form a *group*. Multiple groups can be stacked. This is particularly useful when multiple shapes are used, which is illustrated in the examples of Section 4. Shapes may have different map projections and different bounding boxes. In that case, the map projection and bounding box of the *master* shape are used in the map. By default, the master is the shape of the first group. If one of the shapes is a raster, then that shape is the master shape. Any shape can be set to the master shape with the argument `is.master` of the function `tm_shape`.

Functions related to the spatial transformation, the second component of the LGTM, are not implemented as element functions, but as stand-alone functions in `tmaptools`. The main reason for this is that these functions are often time consuming and require manual configuration. It is therefore recommended to apply the statistical transformation prior to the plotting phase.

Small multiples can be created in three ways. First, a vector of constant values or data

Function	Geometry	Accepted spatial units	Aesthetics	
<i>Base layer functions</i>				
<code>tm_polygons</code>	polygons	polygons	<code>col</code>	fill color
<code>tm_symbols</code>	symbols	points, polygons, and lines	<code>size</code>	symbol size
			<code>col</code>	symbol color
			<code>shape</code>	symbol shape
<code>tm_lines</code>	lines	lines	<code>col</code>	line color
			<code>lwd</code>	line width
<code>tm_raster</code>	raster	raster	<code>col</code>	raster color
<code>tm_text</code>	text	points, polygons, and lines	<code>text</code>	text labels
			<code>size</code>	text size
			<code>col</code>	text color
<i>Derived layer functions</i>				
<code>tm_fill</code>	polygons (fill)	polygons	see <code>tm_polygons</code>	
<code>tm_borders</code>	polygons (borders)	polygons	none	
<code>tm_bubbles</code>	bubbles	points, polygons, and lines	see <code>tm_symbols</code>	
<code>tm_squares</code>	squares	points, polygons, and lines	see <code>tm_symbols</code>	
<code>tm_dots</code>	dots	points, polygons, and lines	see <code>tm_symbols</code>	
<code>tm_rgb</code>	RGB image	raster	none	
<code>tm_markers</code>	marker symbols with text labels	points, polygons, and lines	see <code>tm_symbols</code> and <code>tm_text</code>	
<code>tm_iso</code>	contour lines with text labels	lines	see <code>tm_lines</code> and <code>tm_text</code>	

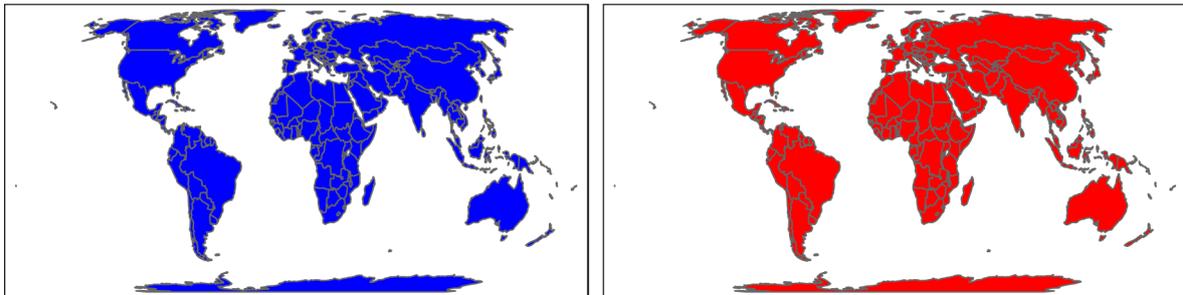
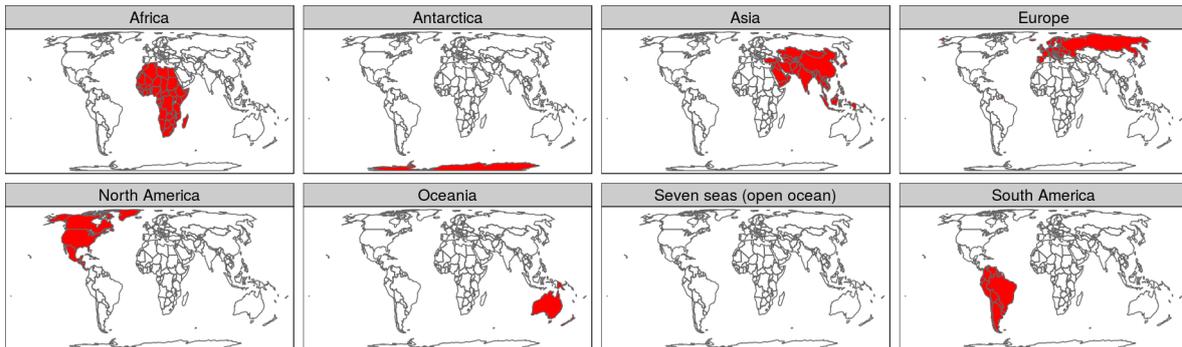
Table 2: Layer elements of package **tmap**.

Figure 3: Small multiples created by assigning two constant values for one aesthetic.

variable names can be assigned to one of the aesthetics listed in Table 2. This generates a small multiple for each constant value or data variable. An example is the following line of code, which creates Figure 3. Constant color values are assigned to the fill aesthetic of `tm_polygons`. In the example of Section 4.3, we illustrate this for multiple data variables.

```
R> tm_shape(World) + tm_polygons(col = c("blue", "red"))
```

The second method to create small multiples is by assigning the names of one or two data variables to the `by` argument of `tm_facets`. In that case, the data is split according to these variables. The following line of code creates small multiples in which countries are colored by

Figure 4: Small multiples created by specifying the `by` argument of `tm_facets`.

Function	Description	Main arguments
<code>tm_facets</code>	small multiples	<code>by</code> group by variable <code>free.coords</code> should each map have its own coordinate ranges? <code>free.scales</code> should aesthetic scales be free for each map? <code>nrow</code> and <code>ncol</code> number of rows and columns

Table 3: Element that configures small multiples.

continent (see Figure 4). We also illustrate this method in Section 4.5.

```
R> tm_shape(World) + tm_polygons(col = "red") +
+   tm_facets(by = "continent", free.coords = FALSE)
```

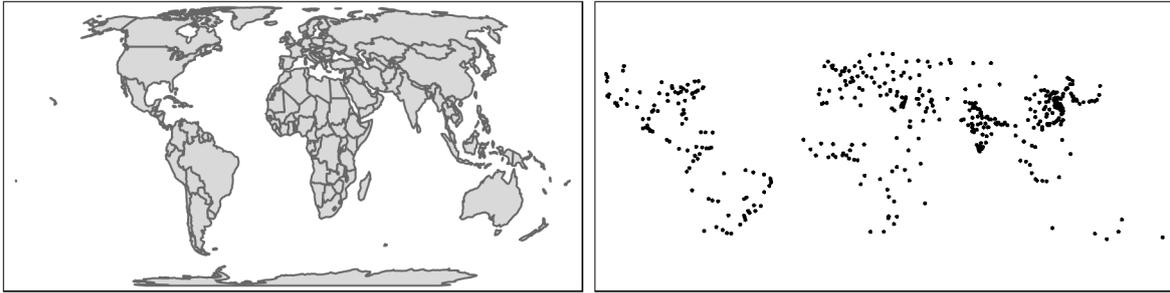
For these first two methods, the function `tm_facets` can be used to further specify the small multiples (see Table 3). By setting `free.coords = FALSE`, the coordinate range will be identical for all small multiples. The argument `free.scales` is used to specify whether the aesthetic scales should be determined per small multiple. By default, this argument is set to `TRUE` for the first method and `FALSE` for the second method. This can also be specified for each aesthetic separately.

The third and final method to create small multiples is by creating several independent maps and use the function `tmap_arrange` to arrange them in a grid layout. This is illustrated in the following code chunk, which produces Figure 5. The objects `tm1` and `tm2` are ‘`tmap`’ objects, which can also be used to export maps. This will be described in Section 3.5.

```
R> tm1 <- tm_shape(World) + tm_polygons()
R> tm2 <- tm_shape(metro) + tm_dots()
R> tmap_arrange(tm1, tm2)
```

The map can be embellished with map attributes. The attribute element functions are listed in Table 4. The usage of map attributes is illustrated in Section 4.2

Elements that are related to the map layout are listed in Table 5. The function `tm_layout` is the main function which contains all arguments regarding the layout in plotting mode. Arguments that are legend related can also be set with `tm_legend`.

Figure 5: Small multiples created with `tmap_arrange`.

Function	Description	Main arguments
<code>tm_grid</code>	coordinate grid lines	<code>x</code> and <code>y</code> position of grid lines <code>projection</code> projection of the grid lines
<code>tm_credits</code>	credits text	<code>text</code> credits text
<code>tm_scale_bar</code>	scale bar	<code>breaks</code> scale bar breaks
<code>tm_compass</code>	map compass	<code>type</code> compass type
<code>tm_logo</code>	logo(s)	<code>file</code> filename or URL
<code>tm_xlab</code> and <code>tm_ylab</code>	axis labels	<code>text</code> label text

Table 4: Map attributes.

The wrapper functions `tm_format_X`, with `X` being the name of a predefined format, specify position related arguments, such as margins and scale, and positions of map items such as title, legend, and map attributes. The optimal values of these arguments may depend on the geometry of the shape. The default format, which is specified by the default values of the position related arguments of `tm_layout`, is supposed to be of general purpose, where the legend position is automatically determined based on the coordinates of the shape. A couple of these predefined formats are included in the **tmap** package, for instance `tm_format_World`, and users are able to specify their own.

The arguments that are specified with the `tm_style_X` wrapper functions, with `X` being the name of a predefined style, do not relate to position and are therefore shape independent. The default style is "white", which refers to the white background. Other styles are "natural" with natural colors, "col\_blind" with color-blind friendly color palettes, and "classic", which emulates old-style maps. The function `style_catalogue` generates a catalogue of all available styles. The default style is defined in the global option "`tmap.style`", and can be get and set with the function `tmap_style`.

The function `tm_view` contains arguments that are specific for the viewing mode. Most importantly, the `alpha` controls the transparency of all layers specified by the functions listed in Table 2. The optimal value is a trade-off between the clarity of the layers and the readability of the underlying basemap. The default basemap depends on the style. For "white", the default style, a neutral, light gray basemap from **CARTO** (CartoDB, Inc. 2017) is used.

Function	Description	Main arguments	
<code>tm_layout</code>	all layout options	<code>title</code>	map title
<code>tm_legend*</code>	legend options	<code>position</code>	legend position
<code>tm_format_X*</code>	formatting options	<code>inner.margins</code>	inner margins
		<code>outer.margins</code>	outer margins
		<code>asp</code>	aspect ratio
		<code>scale</code>	global size multiplier of all text, bubbles, and lines
<code>tm_style_X*</code>	styling options	<code>bg.color</code>	background color
		<code>aes.color</code>	default colors for constant aesthetics
		<code>aes.palette</code>	default color palettes for mapped aesthetics
		<code>basemaps</code>	basemaps (for viewing mode only)
<code>tm_view</code>	interactive options	<code>alpha</code>	overall transparency

Table 5: Layout related elements of `tmap`. The functions marked with \* are wrappers of `tm_layout`.

### 3.3. Quick plotting method

The function `qtm` stands for quick thematic map. It is a wrapper of the main plotting method. The map shown in Figure 2 can also be created with the quick plotting method using the following line of code.

```
R> qtm(metro, symbols.size = "pop2030", format = "World", style = "cobalt")
```

Its first argument is a shape. In the main plotting method, it is defined with `tm_shape`. If none of the other arguments are specified, in this example `qtm(metro)`, then a simple map of the shape is drawn. This is comparable to the generic `plot` function implemented in packages `sp` and `raster` for ‘Spatial’ and ‘Raster’ objects respectively, but with a different layout.

The other important arguments of `qtm` are the aesthetics from the base layer functions listed in Table 2 and the `by` argument from `tm_facets` that can be used to define small multiples. In the code above, `symbols.size` is the only specified aesthetic. All other arguments are passed on to the element functions listed in Tables 1, 2, 3, and 5. In order to assign the arguments to the correct element functions, the argument names can be prefixed with the element function name excluding `tm_`. For instance, the argument `symbols.alpha` controls the alpha transparency of the symbols.

Note that only one shape object can be assigned to `qtm` and each of the layers only once. Therefore, `qtm` is actually one group of one or more layers. It is possible to stack multiple `qtm` function calls with the `+` operator. The function `qtm` can also be used in conjunction with element functions listed in Tables 2, 3, 4 and 5.

In interactive mode, the `qtm` function can also be called without arguments, or with a search query. In these cases, the basemaps are shown interactively. When a search query is provided, a place marker is added to the map at the coordinates found by the OpenStreetMap Nominatim search engine.

### 3.4. Reading and processing spatial data

The **tmaptools** package contains several functions for reading and processing spatial data, of which the most important ones are listed in Table 6. These functions should be sufficient to supply the workflow that is needed to create thematic maps in most common cases. Most of these functions are convenient wrappers around functions from underlying packages.

OpenStreetMap (OSM) data exists in two formats: vectors and rasters. Typically, vectors are used when analyzing OSM data and rasters when plotting OSM data as a background layer. Both can be read with `read_osm`. It requires a bounding box or a shape object from which the bounding box is extracted. Note that reading OSM rasters with `read_osm` is only useful in "plot" mode, since in "view" mode, OSM rasters are already shown as basemaps. The function `read_osm` requires the suggested R package **OpenStreetMap**, which requires Java to be installed.

The function `bb` can be seen as a Swiss army knife regarding bounding boxes. It can retrieve and modify existing bounding boxes, for instance with the extension factor argument `ext`, and reproject them. Furthermore, it allows users to create new bounding boxes from scratch, either using coordinates or by querying OpenStreetMap Nominatim. For the latter, the function `geocode_OSM` is used.

Often, spatial data is not yet contained in a shape object. In that case, the data can be appended to a shape object with the function `append_data`. In database terminology, a left join is executed, i.e., matched data will be appended to the corresponding spatial units where missing values will be used for spatial units without match. Both the under coverage and over coverage of the match can be retrieved. Under coverage is the subset of spatial units for which no data records could be found. Over coverage is the subset of data for which no spatial units could be found. The function `append_data` will be illustrated in Section 4.4.

The **tmap** package uses color palettes from ColorBrewer (Brewer *et al.* 2003; Neuwirth 2014) by default. A graphical user interface that enables users to explore these palettes can be started with the function `palette_explorer`. Users are able to change the number of colors and the contrast range. Moreover, they can test the palettes for color blindness.

The **tmaptools** package contains a couple of spatial transformation functions. The functions `set_projection` and `crop_shape` can be used to set the map projection and crop the shape by a bounding box respectively. Both map projection and bounding box can also be specified directly in the plot code as arguments of `tm_shape`. With `crop_shape`, it is also possible to crop a shape by a polygon instead of a rectangle. The function `simplify_shape` reduces the number of coordinates used to describe polygons or lines. This is especially useful for shapes with high detailed natural boundaries, such as coastlines or rivers, that may become too occluded when they are plot in low resolution. The function `smooth_map` creates a smooth map by estimating a two-dimensional kernel density from the spatial data distribution of the input shape. The output of this function can be used to create two-dimensional kernel density maps or smoothed raster maps. The two-dimensional kernel densities are calculated with package **KernSmooth** package (Wand 2015).

For cartograms, we suggest to use the **cartogram** package (Jeworutzki 2016). Contiguous and non-contiguous cartogram shapes can be created with the functions `cartogram` and `nc_cartogram` respectively. These shapes can be visualized with the **tmap** package using the standard approach, as described in Sections 3.2 and 3.3.

Function	Description	Main arguments	
<i>Reading spatial data</i>			
<code>read_shape</code>	read ESRI shape files	<code>file</code>	file name
<code>read_osm</code>	read OpenStreetMap data	<code>x</code>	bounding box or shape object
<code>geocode_OSM</code>	find location	<code>q</code>	location search query
<code>rev_geocode_OSM</code>	find address	<code>x</code>	'SpatialPoints' object or $x$ coordinates
		<code>y</code>	$y$ coordinates
<i>Tool functions</i>			
<code>bb</code>	bounding box generator	<code>x</code>	shape object, bounding box, or location search query
<code>append_data</code>	append data to shape object	<code>shp</code> <code>data</code>	shape object data to append
<code>approx_areas</code>	approximate polygon areas	<code>shp</code>	shape object
<code>palette_explorer</code>	explore color palettes		
<i>Spatial transformation functions</i>			
<code>set_projection</code>	set map projection or reproject map	<code>shp</code> <code>projection</code>	'tmap' object projection code
<code>crop_shape</code>	crop a shape object	<code>x</code> <code>y</code>	shape object bounding box or shape object
<code>simplify_shape</code>	simplify a shape object	<code>polygon</code> <code>shp</code>	crop by $y$ as polygon? shape object (polygons or lines)
<code>aggregate_map</code>	aggregate a map	<code>fact</code> <code>shp</code> <code>by</code> <code>fact</code>	simplification factor shape object (except spatial points) group variable name raster aggregation factor
<code>smooth_map</code>	create a smooth map using two-dimensional kernel density estimations	<code>shp</code>	shape object (except spatial lines)
<code>cartogram</code> and <code>nc_cartogram</code>	create a cartogram (from the <b>cartogram</b> package)	<code>shp</code> <code>weight</code>	polygon shape object name of a numeric data variable

Table 6: Functions to read and process spatial data.

### 3.5. Output

A plot created with **tmap** silently returns an object of class 'tmap'. This object can be used to redraw the map, possibly in the other mode, or export the map. Table 7 lists the export possibilities.

Function	Description	Main arguments	
<code>print</code>	plot or view	<code>tm</code>	'tmap' object
<code>last_map</code>	redraw the last map		
<code>tmap_leaflet</code>	obtain leaflet object	<code>x</code>	'tmap' object
<code>save_tmap</code>	save map	<code>tm</code>	'tmap' object
		<code>width</code>	width
		<code>height</code>	height
<code>animation_tmap</code>	animation (GIF or MPEG)	<code>tm</code>	'tmap' object that creates multiple plots

Table 7: Output functions.

The `print` function draws the map from a 'tmap' object, either static or interactive, based on the chosen mode. In "plot" mode, the map is shown in the graphics device. The layout of the map is configured for the original size and aspect ratio of the graphics device window. Therefore, it is recommended to set the proper size of the graphics device window prior to plotting the map. The function `last_map` can be used to redraw the last map in a resized window.

In "view" mode, an HTML widget is printed. This widget can also be obtained explicitly with the function `tmap_leaflet` and further processed with functions from the R package `leaflet`.

Maps made with `tmap` can easily be saved into the usual formats such as JPG, PNG, and PDF with `save_tmap`. If only one of the dimensions `width` and `height` is specified, the other will be set automatically based on the aspect ratio of plot, which depends on the aspect ratio of the shape, the margins, and the layout of small multiples. The `save_tmap` function can also export interactive maps to stand-alone HTML pages. When the 'tmap' object is not specified, the last map is taken, retrieved with `last_map()`. The type of output does not depend on the current mode, but on the file extension.

When the free software tool **ImageMagick** ([ImageMagick Studio LLC 2017](#)) is installed, animations can be made with `animation_tmap`. This is particularly useful to visualize a time series of spatial data. The main argument of `animation_tmap` is a 'tmap' object that results in multiple plots. This can be achieved by generating small multiples, where each multiple is plot on a separate page, which can be done by using the argument `along` of `tm_facets` or by setting both `nrow` and `ncol` of `tm_facets` to one.

## 4. Examples

In this section, the syntax described in the previous section is illustrated with a set of examples that cover most of the features of `tmap`. Both the main and the quick plotting methods will be applied. Preprocessed shape objects that are contained in package `tmap` are used in the examples in Sections 4.1, 4.2, and 4.3. We illustrate how package `tmertools` can be used to read and process spatial data from external sources in Sections 4.4 and 4.5. Here, we also show how to go from exploratory to explanatory data visualization. More examples can be found in the package documentation.

The following code is required to load the `tmap` and `tmertools` packages.

```
R> library("tmap")
R> library("tmtools")
```

#### 4.1. Metropolitan areas

The code required to create the map depicted in Figure 1 is the following.

```
R> data("World", "metro", package = "tmap")
R> metro$growth <- (metro$pop2020 - metro$pop2010)/(metro$pop2010 * 10) * 100
R> tm_shape(World) + tm_polygons("income_grp", palette = "-Blues",
+   title = "Income class", contrast = 0.7, border.col = "gray30",
+   id = "name") +
+   tm_text("iso_a3", size = "AREA", col = "gray30", root = 3) +
+   tm_shape(metro) + tm_bubbles("pop2010", col = "growth",
+   border.col = "black", border.alpha = 0.5,
+   breaks = c(-Inf, 0, 2, 4, 6, Inf), palette = "-RdYlGn",
+   title.size = "Metro population (2010)",
+   title.col = "Annual growth rate (%)", id = "name",
+   popup.vars = c("pop2010", "pop2020", "growth")) +
+   tm_format_World() + tm_style_gray()
```

First, two shapes that are included in the **tmap** package are loaded. The shape `World` is a ‘SpatialPolygonsDataFrame’ of the countries of the world. The shape `metro` is a ‘SpatialPointsDataFrame’ object containing a population time series for metropolitan areas. In the second line of code, the annual growth rate percentage is calculated.

The first group of the plot call defines the choropleth and adds the country codes using the `World` shape. A bubble map is drawn on top of that using the `metro` shape.

In this example, `World` serves as the master shape, so its projection, the equal-area Eckert IV projection, is used in this map. This map can also be drawn in longitude-latitude (WGS84) coordinates by letting the `metro` shape be the master, which can be done by setting `is.master = TRUE`. However, in that case, also the bounding box will be taken from the `metro` shape, which will result in a map that is cropped too much. Better is to reproject the `World` shape to longitude-latitude coordinates by replacing `tm_shape(World)` with `tm_shape(World, projection = "longlat")`.

Figure 6 shows the map in "view" mode. The code to produce this map is the following. After the mode is set to "view" with `tmap`, the map is redrawn with the function `last_map`.

```
R> ttm()
```

```
tmap mode set to interactive viewing
```

```
R> last_map()
```

Text size will be constant in view mode.

Set `tm_view(text.size.variable = TRUE)` to enable text size variables.

Legend for symbol sizes not available in view mode.

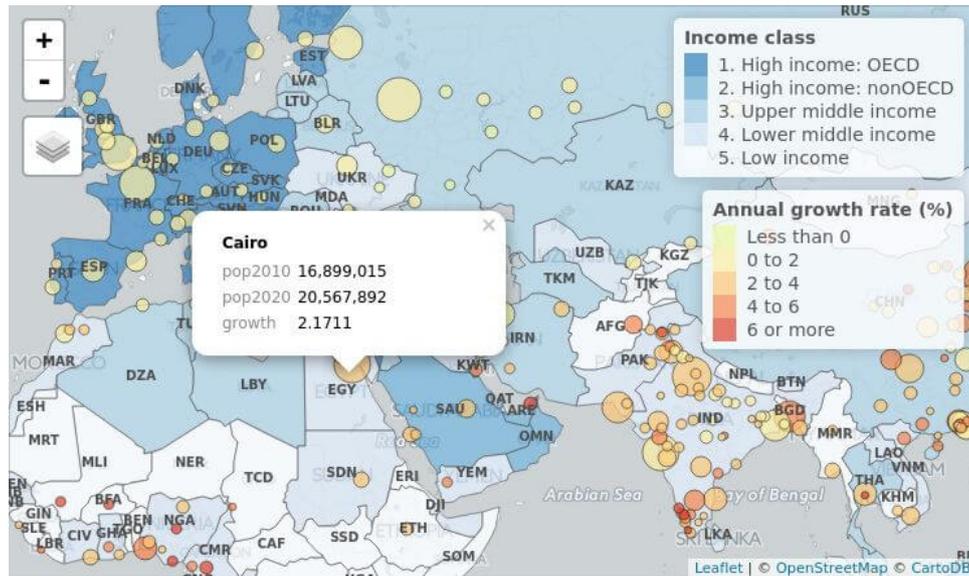


Figure 6: Screenshot of the interactive "view" mode of the map depicted in Figure 1.

Text labels in thematic maps often cause occlusion. Common ways to prevent this are using different font sizes, hiding less important text labels, and changing placement of the text labels. These options can be enabled by using the specification of `tm_text`. In the code chunk that reproduces Figure 1, the polygon area sizes are used to scale the text labels, where `size.lowerbound`, by default 0.4, determines which labels are shown. Alternatively, the arguments `auto.placement` and `remove.overlap` can be used to change the placement of the text labels. However, in "view" mode, occlusion is usually less of a problem due to the ability to zoom in. Therefore, text sizes are kept constant in "view" mode by default, as shown in Figure 6.

The Javascript library **Leaflet** does not support legend items for bubble sizes yet. However, by clicking on a bubble, its data values are shown in a small pop-up window. The argument `id` of the layer element functions `tm_polygons` and `tm_bubbles` are both specified with "name", which is the data variable that contains the names of the countries and the metropolitan areas. These names are included in the pop-up text as titles.

The size of the bubbles are not fixed, but scale proportionally with the zoom level. This behavior can be changed by setting `tm_view`'s argument `bubble.size.fixed` to `TRUE`.

Both the choropleth and the bubble map can be created quickly with `qtm` as the following two lines of code show.

```
R> qtm(World, fill = "income_grp", text = "iso_a3", text.size = "AREA")
R> qtm(metro, symbols.size = "pop2010", symbols.col = "growth")
```

The map depicted in Figure 1 can also be reproduced with `qtm`. To do this, the two function calls above need to be extended with the remaining arguments, e.g., `fill.palette = "-Blues"` in the first line, and to be stacked with the `+` operator. However, for readability issues, we only recommend `qtm` for quick plotting.

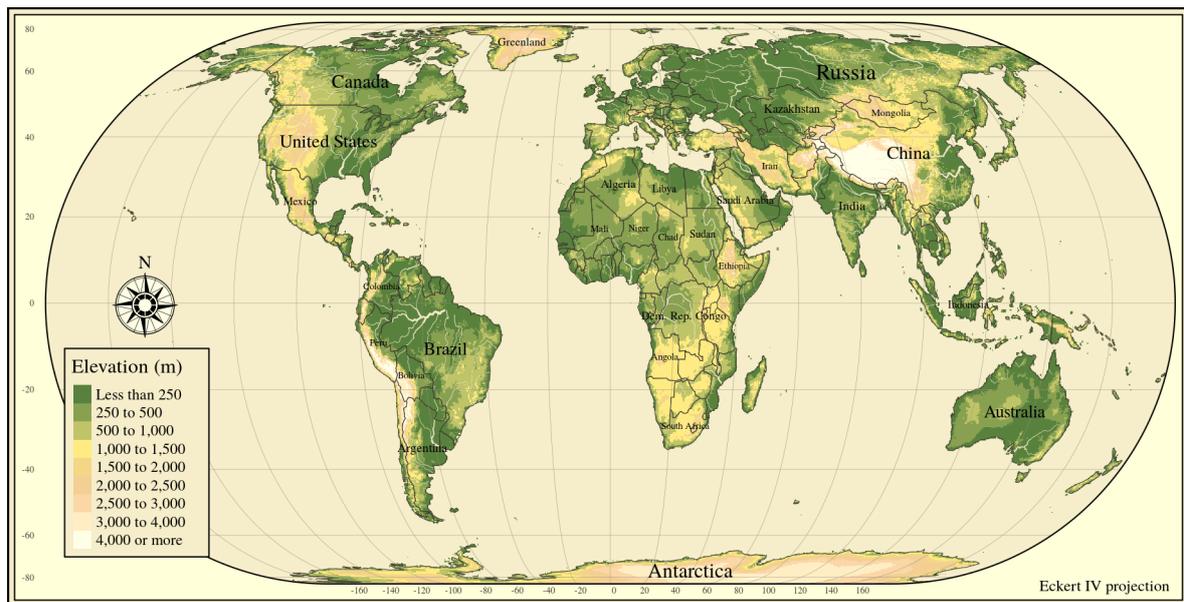


Figure 7: Classic world map, consisting of gridded elevation data, rivers, country borders, and country names.

## 4.2. Classic world map

Figure 7 illustrates that the **tmap** package is not limited to maps for statistical purposes. A classic styled world map is shown with elevation data, country borders and names, the largest world rivers, projected grid lines and a map compass.

```
R> data("World", "land", "rivers", package = "tmap")
R> tm_shape(land) + tm_raster("elevation", breaks = c(-Inf, 250, 500, 1000,
+ 1500, 2000, 2500, 3000, 4000, Inf), palette = terrain.colors(9),
+ title = "Elevation (m)", auto.palette.mapping = FALSE) +
+ tm_shape(rivers) +
+ tm_lines("lightblue", lwd = "strokewd", scale = 1.5,
+ legend.lwd.show = FALSE) + tm_shape(World, is.master = TRUE) +
+ tm_borders("gray20", lwd = .5) +
+ tm_grid(projection = "longlat", labels.size = 0.4, lwd = 0.25) +
+ tm_text("name", size = "AREA") +
+ tm_compass(position = c(0.08, 0.45), color.light = "gray90",
+ size = 3) +
+ tm_credits("Eckert IV projection", position = c("RIGHT", "BOTTOM")) +
+ tm_layout(inner.margins = c(0.04, 0.04, 0.03, 0.02),
+ earth.boundary = TRUE) +
+ tm_style_classic(bg.color = "lightblue", space.color = "gray90") +
+ tm_legend(position = c("left", "bottom"), frame = TRUE,
+ bg.color = "lightblue")
```

The ‘SpatialGridDataFrame’ object `land` is reprojected from longitude-latitude coordinates

to the Eckert IV projection, since `is.master = TRUE` in the specification of the shape object `World`.

A quick map of elevation data and country borders can be created with the following line of code.

```
R> qtm(land, raster = "elevation") + qtm(World, fill = NULL)
```

The color classes are by default based on rounded equidistant breaks. Since there are also negative elevation numbers, that is below mean sea level, a diverging color palette is used in this case.

### 4.3. Two world views

Two thematic world maps are depicted in Figure 8 as small multiples. The map at the top shows the happy planet index across countries and the map at the bottom shows the gross domestic product (GDP) per capita. The following code is needed to produce these maps.

```
R> data("World", package = "tmap")
R> tm_shape(World, projection = "robin") +
+   tm_polygons(c("HPI", "gdp_cap_est"),
+   auto.palette.mapping = c(FALSE, TRUE),
+   palette = list("RdYlGn", "Purples"),
+   style = c("pretty", "fixed"), n = 7,
+   breaks = list(NULL, c(0, 500, 2000, 5000, 10000, 25000, 50000, Inf)),
+   title = c("Happy Planet Index", "GDP per capita")) +
+   tm_credits(c("", "Robinson projection"),
+   position = c("RIGHT", "BOTTOM")) +
+   tm_format_World(inner.margins = 0.02, frame = FALSE) +
+   tm_style_natural(earth.boundary = c(-180, 180, -87, 87)) +
+   tm_legend(position = c("left", "bottom"), bg.color = "gray95",
+   frame = TRUE)
```

By assigning two data variables to an aesthetic, in this case polygon fill color, two small multiples are generated. The call of the function `tm_polygons` shows how other, non-aesthetic, arguments can be specified per small multiple. Arguments that normally take a single value can also take a vector of values, one per small multiple. Likewise, arguments that normally take a vector can also take a list of vectors, one per small multiple.

The arrangement of the small multiples in terms of rows and columns is determined automatically based on the aspect ratios of the map and the graphics device, but can be changed with the function `tm_facets`.

Small multiples can be created with `qtm` in a similar way. The following line of code generates two small multiples of the same data variables, using default values for the non-aesthetic arguments.

```
R> qtm(World, fill = c("HPI", "gdp_cap_est"), style = "natural")
```

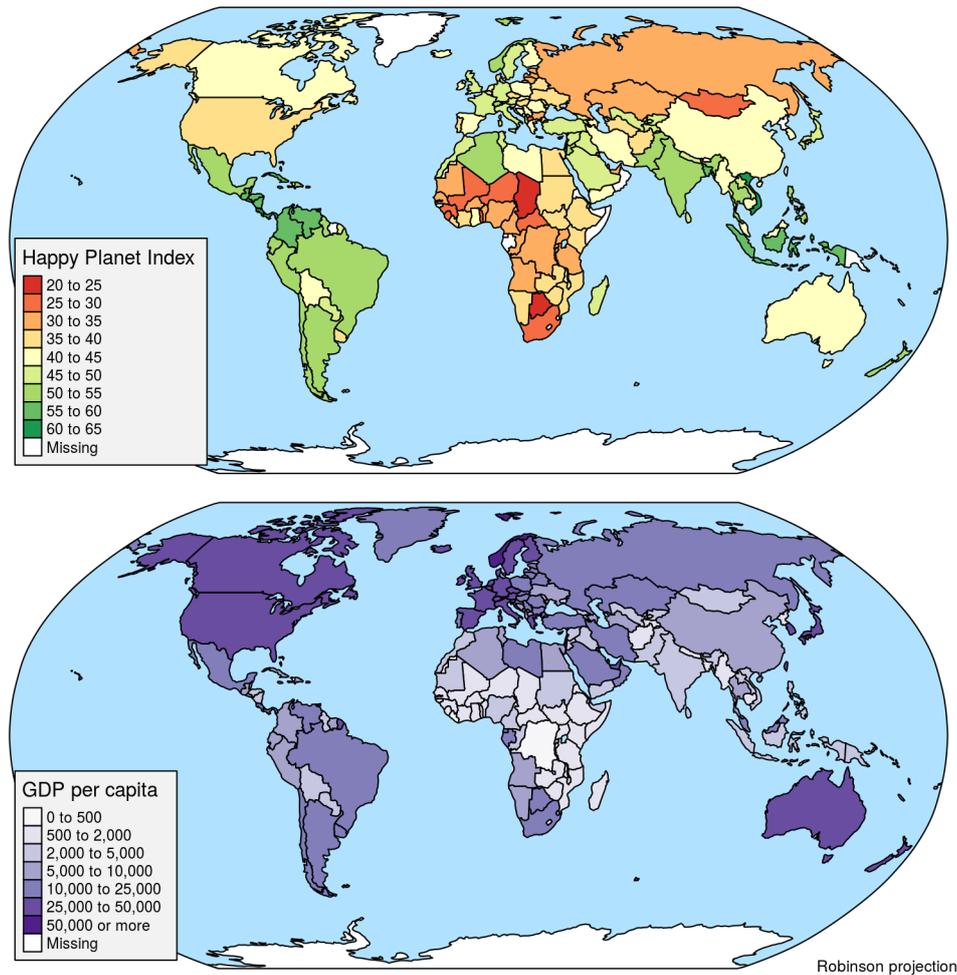


Figure 8: Two choropleths on country level. The top one shows the happy planet index and the bottom one shows the GDP per capita.

#### 4.4. Obesity in the United States

In this example, we will explore obesity in the United States. Percentages of adult obesity per county have been published in the Food Environment Atlas (ERS and USDA 2017). This data is assigned to the object `FEA`, which is a `data.frame`. The shape with county borders for 2010 (US Census Bureau 2016) has been downloaded manually and assigned to the ‘SpatialPolygonsDataFrame’ object `US`. The package `tigris` (Walker 2016) provides a convenient interface for shapes published by the US Census Bureau that are more recent.

First, we plot the `US` shape to ensure it is the correct shape. The result is shown in Figure 9.

```
R> qtm(US)
```

Note that some of the islands of Alaska are drawn at the right-hand side of the map, because they have positive longitude values. The map with coordinates grid lines can be shown with `qtm(US) + tm_grid()`.

When the data is separate from the shape, which is the case in this example, the first thing

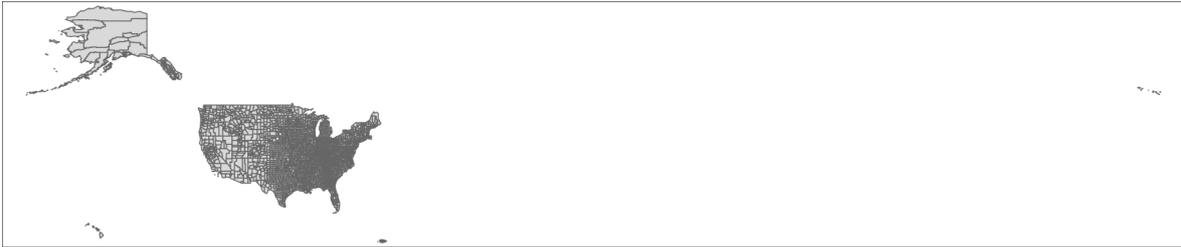


Figure 9: Shape of the counties of the United States.

to do is to append the data to the shape file. We use the function `append_data` where the data is matched by Federal Information Processing Standard (FIPS) code.

```
R> US <- append_data(US, FEA, key.shp = "FIPS", key.data = "FIPS",
+   ignore.duplicates = TRUE)
```

```
Data contains duplicated keys: 08014
Over coverage: 34 out of 3255 data records were not appended. Run
over_coverage() to get the corresponding data row numbers and key values.
```

It turns out that there are two identical data records for county 08014 (Broomfield, Colorado). Without setting `ignore.duplicates` to `TRUE`, the duplicated FIPS codes would have caused an error message. In this example, there are 34 unmatched data records, which can be identified as follows.

```
R> unmatched_data <- over_coverage()
R> str(unmatched_data)
```

```
List of 4
 $ result: chr "Over coverage: 34 out of 3255 data records were not
   appended."
 $ call  : chr [1:2] "append_data(shp = US, data = FEA, key.shp = \"FIPS\",
   key.data = \"FIPS\", \" \" ignore.duplicates = TRUE)"
 $ id    : int [1:34] 68 90 93 94 99 339 559 1663 2931 2959 ...
 $ value : chr [1:34] "02010" "02201" "02231" "02232" ...
```

The list item `id` contains the row numbers of the unmatched data records. Inspection of these records with `View(FEA[unmatched_data$id, ])` reveals that the corresponding values of the variable of interest, `PCT_DIABETES_ADULTS10`, are missing. Hence, the over coverage can be safely ignored.

An interactive choropleth of adult obesity can be created with the following code. The produced map, depicted in Figure 10, is already useful for data exploration and, with some minor improvements, also for publication. These improvements include the adjustment of the legend title and the addition of thick state borders. In the remaining of the example, we illustrate these improvements for the "plot" mode, but they can also be applied in "view" mode. Adjustments that are specific to the "view" mode, such as the basemap specification and the overall alpha transparency of the layers on top, can be set with `tm_view`.

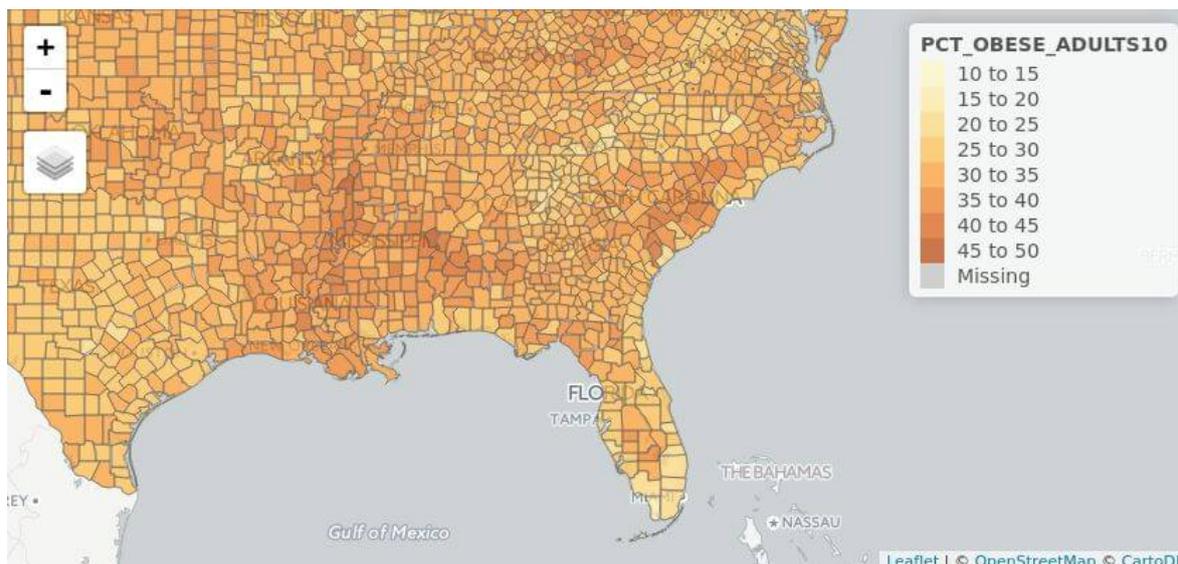


Figure 10: Interactive choropleth of adult obesity percentages per county.

```
R> ttm()
```

```
tmap mode set to interactive viewing
```

```
R> qtm(US, fill = "PCT_OBESE_ADULTS10")
```

Static maps of the United States usually contain insets of Alaska and Hawaii. To accomplish this, the county polygons of Alaska and Hawaii need to be separated from the contiguous United States. In the following code chunk, we create three shapes. The shape `US_cont` consists of county polygons of the contiguous United States. The shapes `US_AK` and `US_HI` represent the counties of Alaska and Hawaii respectively. Furthermore, we simplify each shape in order to reduce the level of detail, which is locally very high, especially along the coastlines. The simplification factor of 0.2 is chosen by trial and error. For these processing steps, we use the pipe operator `%>%`, which is imported from the **magrittr** package (Bache and Wickham 2014).

```
R> US_cont <- US %>% subset(!STATE %in% c("02", "15", "72")) %>%
+   simplify_shape(0.2)
R> US_AK <- US %>% subset(STATE == "02") %>% simplify_shape(0.2)
R> US_HI <- US %>% subset(STATE == "15") %>% simplify_shape(0.2)
```

Thick borders of aggregated regions, in this example the states, usually improve the readability of choropleths. We create a shape of state polygons by aggregating the county polygons of the contiguous United States.

```
R> US_states <- US_cont %>% aggregate_map(by = "STATE")
```

The choropleth of the contiguous United States is created as follows. The result is shown in Figure 11.

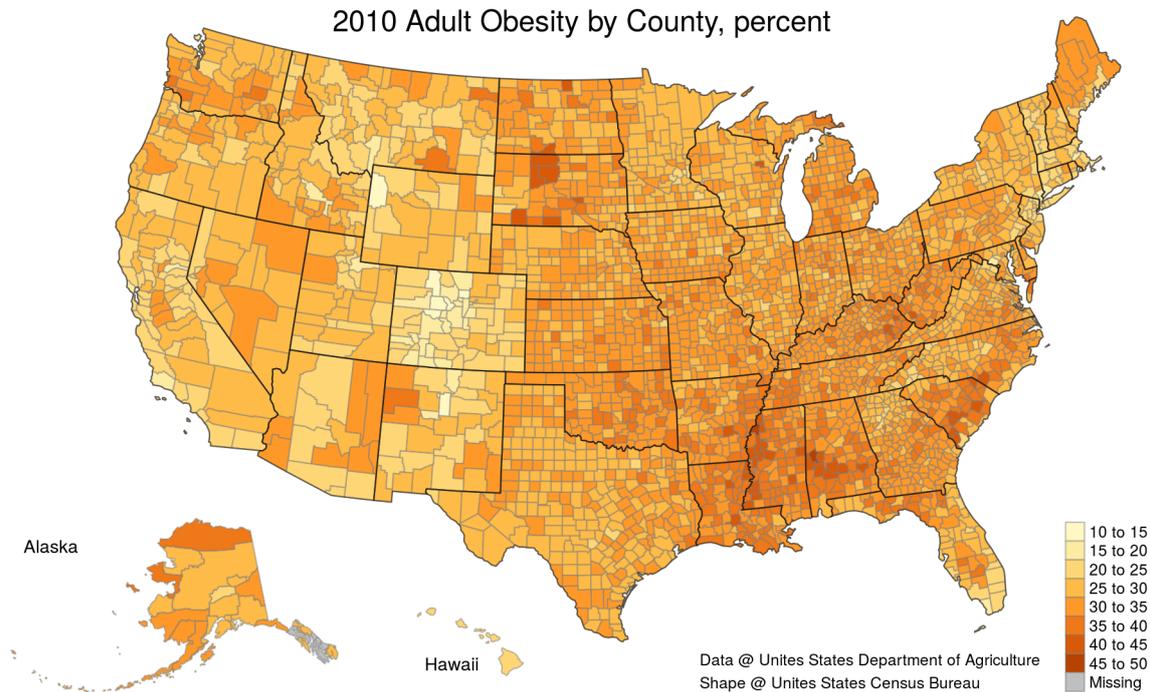


Figure 11: Choropleth of the United States on county level with additional state borders. Alaska and Hawaii are placed as map insets.

```
R> m_cont <- tm_shape(US_cont, projection = 2163) +
+   tm_polygons("PCT_OBESE_ADULTS10", title = "", showNA = TRUE,
+     border.col = "gray50", border.alpha = .5) + tm_shape(US_states) +
+   tm_borders(lwd = 1, col = "black", alpha = .5) +
+   tm_credits(paste0("Data @ Unites States Department of Agriculture\n",
+     "Shape @ Unites States Census Bureau"),
+     position = c("right", "bottom")) +
+   tm_layout(title = "2010 Adult Obesity by County, percent",
+     title.position = c("center", "top"),
+     legend.position = c("right", "bottom"), frame = FALSE,
+     inner.margins = c(0.1, 0.1, 0.05, 0.05))
```

Note that the choropleth of contiguous United States is not plotted yet, but assigned to the variable `m_cont`, which is a ‘tmap’ object. Likewise, the maps of Alaska and Hawaii are created and assigned to the variables `m_AK` and `m_HI` respectively. For these two maps, we define viewports in which the insets will be printed.

```
R> library("grid")
R> vp_AK <- viewport(x = 0.15, y = 0.15, width = 0.3, height = 0.3)
R> vp_HI <- viewport(x = 0.4, y = 0.1, width = 0.2, height = 0.1)
```

The following code chunk shows how these maps are printed. The map of contiguous United States is plotted in the normal way. The maps of Alaska and Hawaii are printed as insets in

the viewports `vp_AK` and `vp_HI` respectively.

```
R> tmap_mode("plot")
```

```
tmap mode set to plotting
```

```
R> m_cont
```

```
R> print(m_AK, vp = vp_AK)
```

```
R> print(m_HI, vp = vp_HI)
```

This map is exported to a PNG file in the following code chunk. The width is set to 6.125 inches, equal to the text width of this journal. The height is not specified. Therefore, it is configured automatically based on the width and the aspect ratio of the map. The `scale` argument determines the overall size of all map objects that are scalable, such as the font size and the line width. Finally, the map insets and the corresponding viewports are specified with the arguments `insets_tm` and `insets_vp` respectively.

```
R> save_tmap(m_cont, "US_adult_obesity.png", scale = 0.7, width = 6.125,
+   insets_tm = list(m_AK, m_HI), insets_vp = list(vp_AK, vp_HI))
```

#### 4.5. Crimes in Greater London

Data about crime in England, Wales, and Northern Ireland have been made publicly available by [Police UK \(2016\)](#). In this example, we will explore the crimes that are registered by the City of London Police and the Metropolitan Police Service during October 2015. The crime data set, which has been downloaded manually from <https://data.police.uk/data>, is available as a supplementary file of this publication.

The data set `crimes` is a `data.frame` that contains latitude, longitude, and crime type, among other variables. With the following code, a `'SpatialPointsDataFrame'` is created. In the second code line, records of crimes with unknown location are removed. Next, the `crimes` object is converted to a `'SpatialPointsDataFrame'` by assigning the coordinate variables. Finally, the map projection is set to the British National Grid, which corresponds to EPSG code 27700 ([IOGP 2015](#)).

```
R> library("sp")
```

```
R> crimes <- crimes[!is.na(crimes$Longitude) & !is.na(crimes$Latitude), ]
```

```
R> coordinates(crimes) <- ~ Longitude + Latitude
```

```
R> crimes <- set_projection(crimes, current_projection = "longlat",
```

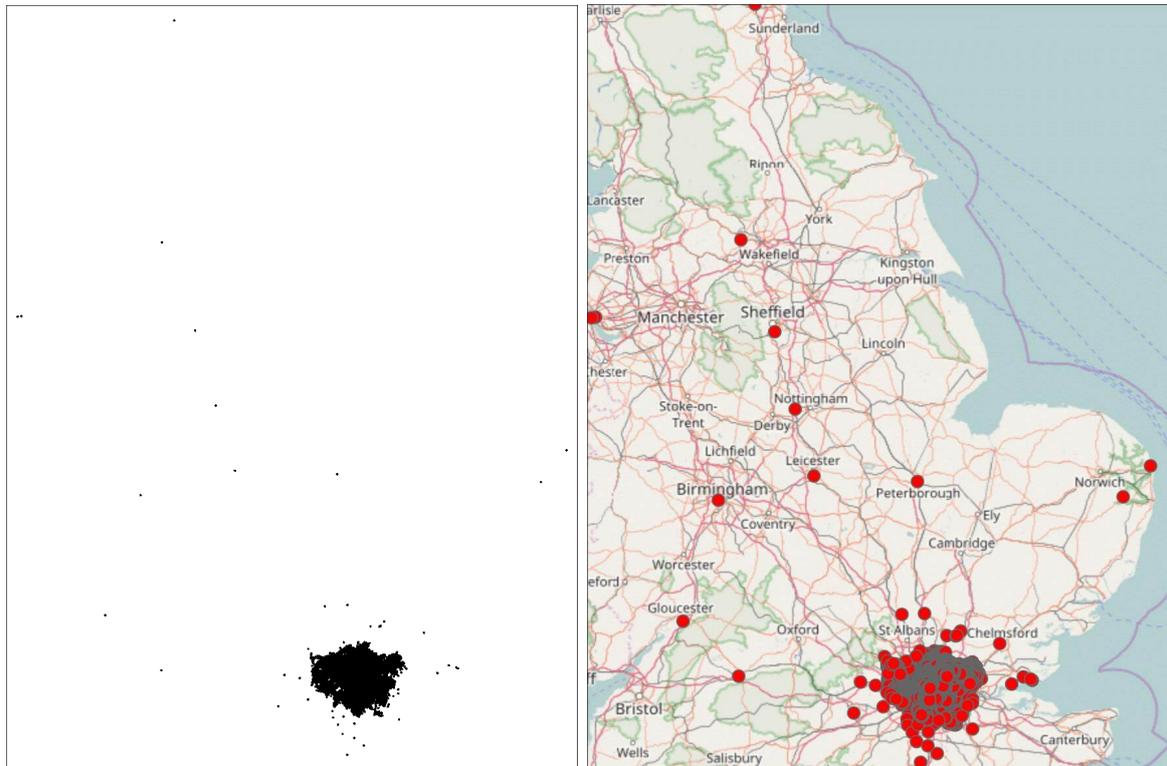
```
+   projection = 27700)
```

A first glance at the data can be done with `qtm`.

```
R> tmap_mode("plot")
```

```
tmap mode set to plotting
```

```
R> qtm(crimes)
```



(a) Dot map of crime locations.

(b) Dot map with added basemap layer.

Figure 12: Dot maps of the crimes data.

The result, a dot map, is shown in Figure 12(a). This map tells that the vast majority of the dots are clustered, whereas the other dots can be marked as outliers. In order to locate the outliers, the same map can be executed in viewing mode with `tmap_mode("view")` and `last_map()`. However, since there are over 80,000 points, the interactive viewer will probably be slow. Another approach is to add a basemap in plotting mode. A basemap from OpenStreetMap can be loaded with `read_osm`, as the following code shows. The bounding box of the basemap is set to be identical to the bounding box of `crimes`, but extended with 5 percent. The function `bb` can be used to obtain this bounding box explicitly. The produced map is depicted in Figure 12(b).

```
R> crimes_osm <- read_osm(crimes, ext = 1.05)
R> qtm(crimes_osm) + qtm(crimes, symbols.col = "red", symbols.size = 0.5)
```

It may be interesting to find out what types of crime were committed outside London. The following code produces the map in Figure 13. The basemap is shown with 50 percent transparency to better highlight the bubbles. From this map, it is clear that most of these outlier crimes are related to violence and sexual offenses.

```
R> qtm(crimes_osm, raster.alpha = 0.5) +
+   qtm(crimes, symbols.col = "Crime.type", symbols.size = 0.5) +
+   tm_legend(outside = TRUE)
```



Figure 13: Map of crimes colored by type.

In the remaining of this example, we will focus on the crimes committed in Greater London. Therefore, we crop the `crimes` object in the following way.

```
R> library("rnaturalearth")
R> regions <- ne_download(scale = "large", type = "states",
+   category = "cultural")
R> london <- regions[which(regions$region == "Greater London"), ]
R> london <- set_projection(london, projection = 27700)
R> crimes_london <- crop_shape(crimes, london, polygon = TRUE)
```

The object `london` is a ‘SpatialPolygonsDataFrame’ of the districts within Greater London. This object is obtained with the package `rnaturalearth` (South 2017). The function `crop_shape` returns the intersection of the first two arguments if the argument `polygon` is set to `TRUE`. In this case, the intersection is taken of the crime locations and the outer borders of Greater London.

A quick plot of `crimes_london` with `qtm` still shows a large amount of occlusion. This can be prevented by applying an alpha transparency to the points. In the following code, we apply ten percent alpha transparency. As a result, individual dots are hard to see, but the overall



Figure 14: Dot map of crimes in Greater London.

data distribution is more clear. The borders of the districts of Greater London are added for geographical reference.

```
R> qtm(crimes_london, dots.alpha = 0.1) + tm_shape(london) +
+   tm_borders()
```

This map, which is shown in Figure 14, clearly illustrates the distribution of crimes in Greater London. However, it is hard to infer the actual density values. This can be done by showing two-dimensional kernel densities instead of dots. The bandwidth of the kernel density estimator determines the smoothness of the kernel and is set to half a kilometer in this case.

```
R> crime_densities <- smooth_map(crimes_london, bandwidth = 0.5,
+   breaks = c(0, 50, 100, 250, 500, 1000), cover = london)
```

The variable `crime_densities` is a list consisting of a spatial raster of smoothed density values, contour lines, and spatial polygons that represent areas with uniform density values. The latter is used to draw the map depicted in Figure 15. This map is made with publication in mind. Besides the borders of the districts of Greater London, the Thames is also drawn as an additional layer since it is a main natural border between the districts. Spatial data

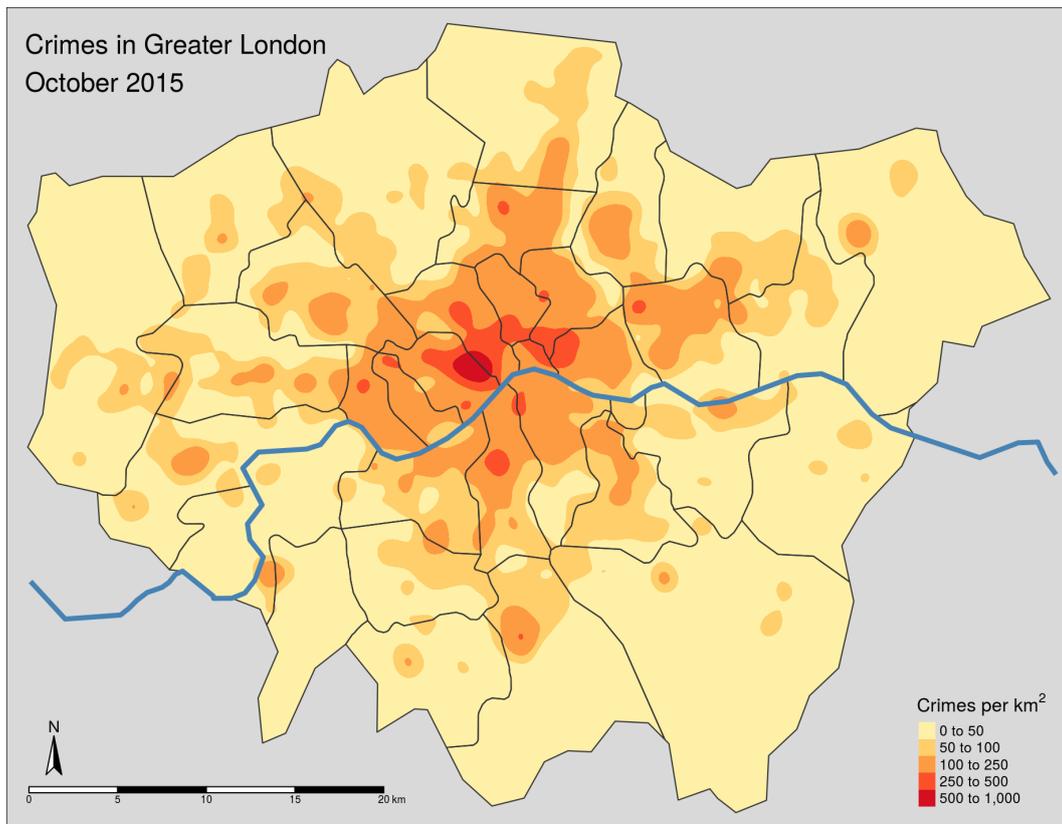


Figure 15: Kernel density map of crimes in Greater London.

of the Thames is downloaded via the package `rnaturalearth` and cropped using `crop_shape`. With the default specification `polygon = FALSE`, the function `crop_shape` determines the intersection of the first object and the bounding box of the second object. Therefore, the Thames is also drawn outside the borders of Greater London.

```
R> rivers <- ne_download(scale = "large", type = "rivers_lake_centerlines",
+   category = "physical")
R> thames <- crop_shape(rivers, london)
```

The following code is used to produce this map in Figure 15. A map compass and scale bar are added to embellish the map. Further, a gray background is chosen, because of the contrast with the yellow areas.

```
R> tm_shape(crime_densities$polygons) + tm_fill(col = "level",
+   palette = "YlOrRd", title = expression("Crimes per " * km^2)) +
+   tm_shape(london) + tm_borders() + tm_shape(thames) +
+   tm_lines(col = "steelblue", lwd = 4) +
+   tm_compass(position = c("left", "bottom")) +
+   tm_scale_bar(position = c("left", "bottom")) +
+   tm_style_gray(title = "Crimes in Greater London\nOctober 2015")
```

The crimes are classified into fourteen types as Figure 13 already showed. Next, we will analyze the distribution of crimes per type in the City of London. In the following code, the



Figure 16: Small multiples that show the crimes per type in the City of London.

district of the City of London is selected from `london` and assigned to a new object of class `'SpatialPolygonsDataFrame'` called `london_city`. An OpenStreetMap basemap is obtained from Stamen-Watercolor, a tile server with artistically designed basemaps. The subset of crimes, that are located in the City of London, are assigned to `crimes_city`.

```
R> london_city <- london[london$name == "City", ]
R> london_osm <- read_osm(london_city, type = "stamen-watercolor", zoom = 13)
R> crimes_city <- crop_shape(crimes_london, london_city, polygon = TRUE)
```

The following code shows how to create small multiples in which the crimes are split by type. The produced plot is shown in Figure 16.

```
R> qtm(london_osm) +
+   qtm(crimes_city, dots.size = 0.2, by = "Crime.type", free.coords = FALSE)
```

The `"view"` mode can be used to explore the crime data in depth. Figure 17 shows a screenshot

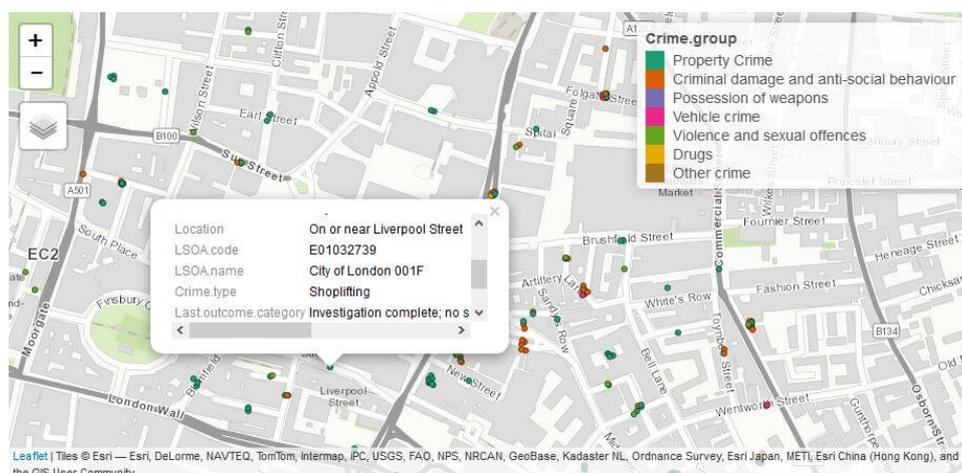


Figure 17: Screenshot of an interactive map that shows the crimes colored by group type.

of the widget. The dots are colored by type, classified into seven groups. Jittering is enabled since many crimes have the exact same location.

```
R> tmap_mode("view")
R> tm_shape(crimes_city) + tm_dots(jitter = 0.2, col = "Crime.group",
+   palette = "Dark2", popup.vars = TRUE) +
+   tm_view(alpha = 1, basemaps = "Esri.WorldTopoMap")
```

This interactive map can be saved to a stand-alone HTML page with `save_tmap`.

```
R> save_tmap(filename = "index.html")
```

## 5. Conclusions and future directions

The **tmap** package can be used to create thematic maps of many kinds. It offers a flexible, layered approach to build thematic maps from scratch. Users have a lot of freedom in several aspects of mapping. First, there are many possibilities to map spatial data to the aesthetics of the layer functions. Users are able to create their own class intervals, use their own color palettes, and customize the corresponding legends. Second, users have a lot of possibilities to format and style maps. Third, the produced maps can be easily exported to different formats with full control over important settings, such as the aspect ratio and the overall scale. Finally, users are able to switch instantly between static plotting and interactive viewing.

Although one of the main objectives of **tmap** is to have a large degree of flexibility and customization, it is also important that the default values are suitable in most cases, since users often need to be able to create thematic maps fast, especially in the phase of data exploration. To accommodate this need, the default format and style are chosen as general as possible. Also, the amount of code to create a minimal working thematic map, for instance a choropleth, is low, especially with the function `qtm`.

A couple of features are not implemented yet in "view" mode. The legend is not yet available for the aesthetics regarding bubbles, dots, lines, and text labels. From the map attributes

listed in Table 4, only the scale bar and the grid lines are supported in "view" mode. Other interactive mapping features, especially those implemented by the R packages **leaflet** and **mapview**, could also be used by **tmap** in future versions.

Another aim for future development is the implementation of the flow map. Arcs between spatial units, points or polygons, represent a certain flow between them. This type of map is especially useful to visualize migration. Flow maps with a low number of arcs that are located between more or less uniformly distributed points are probably easy to implement. However, for a high number of flows, or when spatial points lie close to each other, the implementation is hard, since it often requires routing and partial clustering of arcs (Buchin, Speckmann, and Verbeek 2011).

Currently, packages **tmap** and **tmaptools** are based on the classes and methods from the **sp**, **rgdal**, and **rgeos** packages for shapes other than rasters. The class of simple features, implemented in the R package **sf**, can be seen as the successor of the class of 'Spatial' objects. This format is similar to GeoJSON, a popular format for shapes, especially for web-based applications, and also used in **PostGIS**, a spatial database (Holl and Plum 2009), and **ArcGIS**. Both **tmap** and **tmaptools** support 'sf' objects. However, 'sf' objects are still coerced to 'Spatial' objects under the hood. For future development, it will be worthwhile to use the 'sf' classes and the corresponding methods directly.

## Acknowledgments

I would like to thank Edwin de Jonge, Roger Bivand, Tim Appelhans and the anonymous referees for their useful and constructive comments on earlier versions of this paper and the software. Furthermore, thanks to all users of the **tmap** package, in particular Joel Gombin, Kent Russell, Sebastian Jeworutzki, Richard Zijdemann, Carsten Behring, Melanie Bacou, Mario Nowak, Egge-Jan Pollé, Robin Lovelace, Andy South, and Edzer Pebesma for providing me with useful feedback and bug reports, and for sharing ideas for extra features and future development.

## References

- Agafonkin V (2017). **Leaflet: An Open-Source JavaScript Library for Mobile-Friendly Interactive Maps**. URL <http://leafletjs.com>.
- Allaire JJ, Cheng J, Xie Y, McPherson J, Chang W, Allen J, Wickham H, Atkins A, Hyndman R, Arslan R (2017). **rmarkdown: Dynamic Documents for R**. R package version 1.5, URL <https://CRAN.R-project.org/package=rmarkdown>.
- Appelhans T, Detsch F, Reudenbach C, Woellauer S (2017). **mapview: Interactive Viewing of Spatial Objects in R**. R package version 2.0.1, URL <https://CRAN.R-project.org/package=mapview>.
- Bache SM, Wickham H (2014). **magrittr: A Forward-Pipe Operator for R**. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.

- Barford A, Dorling D (2008). “Telling an Old Story with New Maps.” In M Turner, M Dodge, M McDerby (eds.), *Geographic Visualization: Concepts, Tools and Applications*, pp. 67–107. John Wiley & Sons.
- Bertin J (1983). *Semiology of Graphics*. University of Wisconsin Press.
- Bivand RS (2017). **classInt**: Choose Univariate Class Intervals. R package version 0.1-24, URL <https://CRAN.R-project.org/package=classInt>.
- Bivand RS, Hauke J, Kossowski T (2013). “Computing the Jacobian in Gaussian Spatial Autoregressive Models: An Illustrated Comparison of Available Methods.” *Geographical Analysis*, **45**(2), 150–179. doi:10.1111/gean.12008.
- Bivand RS, Keitt TH, Rowlingson B (2017). **rgdal**: Bindings for the Geospatial Data Abstraction Library. R package version 1.2-7, URL <https://CRAN.R-project.org/package=rgdal>.
- Bivand RS, Pebesma EJ, Gómez-Rubio V (2008). *Applied Spatial Data Analysis with R*. Springer-Verlag, New York. doi:10.1007/978-1-4614-7618-4.
- Bivand RS, Piras G (2015). “Comparing Implementations of Estimation Methods for Spatial Econometrics.” *Journal of Statistical Software*, **63**(18), 1–36. doi:10.18637/jss.v063.i18.
- Bivand RS, Rundel C (2017). **rgeos**: Interface to Geometry Engine – Open Source (GEOS). R package version 0.3-23, URL <https://CRAN.R-project.org/package=rgeos>.
- Brewer CA, Hatchard GW, Harrower MA (2003). “ColorBrewer in Print: A Catalog of Color Schemes for Maps.” *Cartography and Geographic Information Science*, **30**(1), 5–32. doi:10.1559/152304003100010929.
- Brewer CA, Pickle L (2002). “Evaluation of Methods for Classifying Epidemiological Data on Choropleth Maps in Series.” *The Annals of the Association of American Geographers*, **92**(4), 662–681. doi:10.1111/1467-8306.00310.
- Brunsdon C, Chen H (2014). **GISTools**: Some Further GIS Capabilities for R. R package version 0.7-4, URL <https://CRAN.R-project.org/package=GISTools>.
- Buchin K, Speckmann B, Verbeek K (2011). “Flow Map Layout via Spiral Trees.” *IEEE Transactions on Visualization and Computer Graphics*, **17**(12), 2536–2544. doi:10.1109/tvcg.2011.202.
- CartoDB, Inc (2017). **CARTO**. CartoDB, Inc., New York. URL <https://carto.com/>.
- Cheng J, Karambelkar B, Xie Y (2017). **leaflet**: Create Interactive Web Maps with the JavaScript Leaflet Library. R package version 1.1.0, URL <https://CRAN.R-project.org/package=leaflet>.
- ERS, USDA (2017). *Food Environment Atlas*. Economic Research Service, U.S. Department of Agriculture. URL <http://www.ers.usda.gov/data-products/food-environment-atlas.aspx>.

- ESRI (2016). **ArcGIS Desktop: Release 10.4**. Environmental Systems Research Institute, Redlands.
- Eugster MJA, Schlesinger T (2013). “**osmar**: OpenStreetMap and R.” *The R Journal*, **5**(1), 53–63.
- Fellows I (2016). **OpenStreetMap**: *Access to Open Street Map Raster Images*. R package version 0.3.3, using the **JMapView** library by Jan Peter Stotz, URL <https://CRAN.R-project.org/package=OpenStreetMap>.
- Fisher PF, Comber AJ, Wadsworth RA (2005). “Land Use and Land Cover: Contradiction or Complement.” In PF Fisher, D Unwin (eds.), *Re-Presenting GIS*, pp. 85–98. John Wiley & Sons.
- Friendly M (1995). “Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization.” In *13th International Conference on Database and Expert Systems Applications (DEXA 2002)*, pp. 59–66. Aix en Provence.
- Gastner MT, Newman MEJ (2004). “Diffusion-Based Method for Producing Density-Equalizing Maps.” *Proceedings of the National Academy of Sciences of the United States of America*, **101**(20), 7499–7504. doi:10.1073/pnas.0400280101.
- Giraud T, Lambert N (2016). “**cartography**: Create and Integrate Maps in Your R Workflow.” *The Journal of Open Source Software*, **1**(4), 1–2. doi:10.21105/joss.00054.
- GIS Geography (2016). “Mapping Out the GIS Software Landscape.” Published on 2016-07-28, URL <http://gisgeography.com/mapping-out-gis-software-landscape/>.
- GRASS** Development Team (2016). *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.0*. Open Source Geospatial Foundation. URL <http://grass.osgeo.org/>.
- Hijmans RJ (2017). **raster**: *Geographic Data Analysis and Modeling*. R package version 2.6-7, URL <https://CRAN.R-project.org/package=raster>.
- Holl S, Plum H (2009). “**PostGIS**.” *GeoInformatics*, **3**, 34–36.
- ImageMagick** Studio LLC (2017). **ImageMagick**: *Convert, Edit, and Compose Images*. Version 7.0.6-1, URL <http://www.ImageMagick.org/>.
- IOGP (2015). “EPSG Geodetic Parameter Dataset.” URL <http://www.epsg.org/>.
- IPCC (2014). *Climate Change 2014: Synthesis Report*. Intergovernmental Panel on Climate Change, Geneva.
- Jeworutzki S (2016). **cartogram**: *Create Cartograms with R*. R package version 0.0.2, URL <https://CRAN.R-project.org/package=cartogram>.
- Kahle D, Wickham H (2013). “**ggmap**: Spatial Visualization with **ggplot2**.” *The R Journal*, **5**(1), 144–161.
- Kilibarda M, Bajat B (2012). “**plotGoogleMaps**: The R-Based Web-Mapping Tool for Thematic Spatial Data.” *GEOMATICA*, **66**(1), 37–49. doi:10.5623/cig2012-007.

- Kraak MJ, Ormeling FJ (2010). *Cartography: Visualization of Spatial Data*. Pearson Prentice Hall.
- Lamstein A, Johnson BP (2017). **choroplethr**: *Simplify the Creation of Choropleth Maps in R*. R package version 3.6.1, URL <https://CRAN.R-project.org/package=choroplethr>.
- Langford M, Unwin DJ (1994). “Generating and Mapping Population Density Surfaces within a Geographical Information System.” *The Cartographic Journal*, **31**(1), 21–26. doi:10.1179/000870494787073718.
- Loecher M, Ropkins K (2015). “**RgoogleMaps** and **loa**: Unleashing R Graphics Power on Map Tiles.” *Journal of Statistical Software*, **63**(4), 1–18. doi:10.18637/jss.v063.i04.
- Lovelace R (2015). *Creating Maps in R*. Online tutorial, URL <https://github.com/Robinlovelace/Creating-maps-in-R>.
- MacEachren AM (1994). *Some Truth with Maps: A Primer on Symbolization and Design*. Association of American Geographers, Washington, DC.
- Neuwirth E (2014). **RColorBrewer**: *ColorBrewer Palettes*. R package version 1.1-2, URL <https://CRAN.R-project.org/package=RColorBrewer>.
- Nusrat S, Kobourov SG (2016). “The State of the Art in Cartograms.” *Computer Graphics Forum*, **35**(3), 619–642. doi:10.1111/cgf.12932.
- Pebesma E (2017). **sf**: *Simple Features for R*. R package version 0.4-1, URL <https://CRAN.R-project.org/package=sf>.
- Pebesma EJ, Bivand RS (2005). “Classes and Methods for Spatial Data in R.” *R News*, **5**(2), 9–13.
- Police UK (2016). “Crime and Policing in England, Wales and Northern Ireland.” URL <https://data.police.uk/>.
- QGIS** Development Team (2016). **QGIS** *Geographic Information System*. Open Source Geospatial Foundation. URL <http://qgis.osgeo.org>.
- R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- RStudio** Inc (2017). **htmltools**: *Tools for HTML*. R package version 0.3.6, URL <https://CRAN.R-project.org/package=htmltools>.
- RStudio** Team (2017). **RStudio**: *Integrated Development Environment for R*. **RStudio**, Inc., Boston, MA. URL <https://www.RStudio.com/>.
- Slocum TA, MacMaster RB, Kessler FC, Howard HH (2009). *Thematic Cartography and Geovisualization*. 3rd edition. Pearson Prentice Hall.
- Snow J (1855). “On the Mode of Communication of Cholera.” London.
- South A (2011). “**rworldmap**: A New R package for Mapping Global Data.” *The R Journal*, **3**(1), 35–43.

- South A (2017). **rnaturalearth**: *World Map Data from Natural Earth*. R package version 0.1.0, URL <https://CRAN.R-project.org/package=rnaturalearth>.
- Steiniger S, Bocher E (2009). “An Overview on Current Free and Open Source Desktop GIS Developments.” *International Journal of Geographical Information Science*, **23**(10), 1345–1370. doi:10.1080/13658810802634956.
- Tennekes M (2018a). **tmap**: *Thematic Maps*. R package version 1.11-2, URL <https://CRAN.R-project.org/package=tmap>.
- Tennekes M (2018b). **tmertools**: *Thematic Map Tools*. R package version 1.2-4, URL <https://CRAN.R-project.org/package=tmertools>.
- Tufte ER (1983). *The Visual Display of Quantitative Information*. Graphics Press.
- Tyner JA (2010). *Principles of Map Design*. Guilford Press.
- UN (2016). *Global Sustainable Development Report 2016*. United Nations, New York.
- US Census Bureau (2016). *Topologically Integrated Geographic Encoding and Referencing (TIGER)*. United States Census Bureau. URL <https://www.census.gov/geography/maps-data/data/tiger.html>.
- Vaidyanathan R, Xie Y, Allaire JJ, Cheng J, Russell K (2016). **htmlwidgets**: *HTML Widgets for R*. R package version 0.8, URL <https://CRAN.R-project.org/package=htmlwidgets>.
- Van Rossum G, et al. (2011). *Python Programming Language*. URL <http://www.python.org/>.
- Villarreal O (2014). *Learning D3.js Mapping*. Community Experience Distilled. Packt Publishing.
- Walker K (2016). “**tigris**: An R Package to Access and Work with Geographic Data from the US Census Bureau.” *The R Journal*, **8**(2), 231–242.
- Wand M (2015). **KernSmooth**: *Functions for Kernel Smoothing Supporting Wand & Jones (1995)*. R package version 2.23-15, URL <https://CRAN.R-project.org/package=KernSmooth>.
- Warmerdam F, Evenden G (2015). “**PROJ.4** – Cartographic Projections Library, version 4.9.2.” URL <http://trac.osgeo.org/proj/>.
- Westra E (2015). *Python Geospatial Analysis Essentials*. Packt Publishing.
- Whittle P (2003). “Contrast Colours.” In R Mausfeld, D Heyer (eds.), *Colour Perception: Mind and the Physical World*, pp. 115–138. Oxford University Press.
- WHO (2016). *World Health Statistics 2016*. World Health Organization, Geneva.
- Wickham H (2009). **ggplot2**: *Elegant Graphics for Data Analysis*. Springer-Verlag. URL <http://ggplot2.org>.
- Wickham H (2010). “A Layered Grammar of Graphics.” *Journal of Computational and Graphical Statistics*, **19**(1), 3–28. doi:10.1198/jcgs.2009.07098.

Wilkinson L (2005). *The Grammar of Graphics*. Statistics and Computing. Springer-Verlag.

**Affiliation:**

Martijn Tennekes  
Statistics Netherlands  
CBS-weg 11  
6412 EX Heerlen, The Netherlands  
E-mail: [mtennekes@gmail.com](mailto:mtennekes@gmail.com)  
URL: <https://github.com/mtennekes/tmap>