Code example

Code to reproduce Fig. 2 in "Open-source tools in R for landscape ecology"

Code created by MHKH, JN, JS and LJG

Contact: mhk.hesselbarth@gmail.com

Setup

First, we need to load all required packages used in this R script. In case you are missing a package, it can be install from CRAN using e.g., install.packages("raster"). By loading the packages, we make sure all functions from them can be used in the script.

```
library(ggplot2)
library(ggspatial)
library(raster)
library(rnaturalearth)
library(sf)
library(tmap)
```

Download data

This function downloads data and saves it as a RasterLayer. The 'name' argument allows to specify a source to download data from the worldclim database. The 'lon' and 'lat' argument specify the SRTM tile for which the data is downloaded, and the 'res' the resolution of the data in minutes of a degree. The 'var' argument specifies which kind of data is downloaded, in this case precipitation data, and lastly the path on the local disk where the data is saved can be set.

```
precipitation <- raster::getData(name = "worldclim", lon = 3, lat = 38,
                                 res = 0.5, var = "prec", path = "R/")
```

Pre-process data

The previous downloaded object is a RasterStack in which each layer corresponds to the precipitation in a month (Jan - Dec). To get the annual precipitation, we can use the calc function which applies basic mathematical function to each layer of the RasterStack. By getting the sum of all layers for each cell, we get the annual precipitation.

```
precipitation_sum <- raster::calc(x = precipitation, fun = sum)
```

Next, we need to ensure that all spatial data has the same coordinate reference system. To achieve this we reproject the downloaded data to the CRS epsg:21781. This returns a warning due to the Proj4 definition, which can be ignored at this point.

```
precipitation_sum <- raster::projectRaster(precipitation_sum, crs = "EPSG:21781")
```

```
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum CH1903 in Proj4 definition
```

Next, we download administration boundaries of Switzerland as vector data from the naturalearth database. Again, we need to specify the resolution of the data (called 'scale' here). Additionally, the function allows to select the return type and we select a sf vector object.

```
switzerland <- rnaturalearth::ne_countries(scale = 50,
                                           country = "switzerland",
                                           returnclass = "sf")
```

To plot the precipitation data together with the administration boundaries, we also need to reproject the boundaries to the same coordinate reference system epsg:21781.

```
switzerland <- sf::st_transform(x = switzerland, crs = "EPSG:21781")
```

Because the precipitation data includes not just Switzerland but whole Europe, we want to crop the raster data by including only cells that are within the the administration boundaries of Switzerland. However, we add a buffer of 5000m around the boundaries.

```
precipitation_sum_ch <- raster::crop(x = precipitation_sum,
                                     y = sf::st_buffer(x = switzerland, dist = 5000))
```

The precipitation data is continuous, however, we want to plot discrete classes. To classify the continuous values into 5 classes, we can use the cut function and simply specify the number of breaks.

```
precipitation_class_ch <- raster::cut(precipitation_sum_ch, breaks = 5)
```

We are going to use the same colors for all maps set by hex color codes. There are many homepages to find such codes e.g., https://htmlcolorcodes.com. We need one color for each class i.e. 5 different colors.

```
colors <- c("#440154FF", "#3B528BFF", "#21908CFF", "#5DC863FF", "#FDE725FF")
```

Create maps

To create a map in base R we first plot the raster with precipitation classes, setting the colors of the 5 classes to the previously set hex codes. Next, we are going to add the vector data to the plot by plotting the sf object and setting the 'add' argument to TRUE which just adds them to the already present map. For better visualization, we add the vector data twice using white and black colors.

```
# pdf(file = "R/Figures/base_plot.pdf")

plot(precipitation_class_ch, col = colors)
plot(st_geometry(switzerland), add = TRUE, lwd = 4, border = "white")
plot(st_geometry(switzerland), add = TRUE)

# dev.off()
```

We can create the same map as a ggplot2 map. Therefore, we first specify which data to use for the plot and then add subsequently the different so-called geoms. This allows to first plot the raster and then add the vector data. Lastly, we can modify some parts of the map as the axis labels or the general appearance (theme). For more information about ggplot2 in general, please see https://ggplot2-book.org

```
plot_gg <- ggplot(raster::as.data.frame(precipitation_class_ch, xy = TRUE)) +
    geom_raster(aes(x = x, y = y, fill = factor(layer))) +
    geom_sf(data = switzerland, fill = NA, col = "white") +
    scale_fill_manual(name = "Precipitation\nclassified", values = colors) +
    coord_sf() +
    annotation_scale(location = "br") +
    annotation_north_arrow(location = "tr", which_north = "true") +
    labs(x = "", y = "") +
    theme_classic() +
    theme(legend.position = "bottom")

# ggplot2::ggsave(filename = "R/Figures/ggplot2.pdf")
```

Creating the map with tmap is comparable to ggplot2. First, we need to specify the data to use and subsequently we can add elements to the map. This includes the raster data, the vector data, a scale bar, a legend (besides other options). For more information about tmap in general, please see https://cran.r-project.org/web/packages/tmap/vignettes/tmap-getstarted.html

```
plot_tm <- tm_shape(precipitation_class_ch) +
    tm_graticules() +
    tm_raster(title = "Precipitation\nclassified",
              style = "cat",
              palette = colors,
              legend.is.portrait = FALSE) +
    tm_shape(switzerland) +
    tm_borders(lwd = 2, col = "white") +
    tm_scale_bar(breaks = c(0, 25, 50),
                 bg.color = "white") +
    tm_compass(position = c("right", "top"),
               bg.color = "white") +
    tm_layout(legend.outside = TRUE,
              legend.outside.position = "bottom")

# tmap::tmap_save(plot_tm, "R/Figures/tmap.pdf")
```