colrow: Handling SimU, CR, and LU data

Pedro R. Andrade

28 August 2019

Abstract

This package has functions to handle Simulation Units (SimU), ColRow (CR), and Large Unit (LU) data. It shows examples on how to export them as shapefiles, create maps, as well as summarize and plot them over time.

Introduction

This package allows users to handle CR, SimU, and LU data. In order to start, it is necessary to install the package from GitHub.

```
devtools::install_github("pedro-andrade-inpe/colrow")
```

If the package will be installed in a Linux machine, it is necessary to manually install some dependencies. Please visit the webpage of the package in GitHub for more details.

Creating shapefiles for a given country

First load the package.

```
require(colrow)
```

```
## Loading required package: colrow
```

```
## colrow - Handling SimU, CR, and LU data
```

```
## Version 0.3-3 is now loaded
```

Download world data from Dropbox here and save them in a single directory. Create a variable in R to store this directory.

```
dataDir <- "c:/Users/pedro/Dropbox/colrow/"
```

The content of the directory should have the following files:

```
list.files(dataDir)
```

```
##  [1] "COLROW30.aux"     "COLROW30.dbf"     "COLROW30.sbn"
##  [4] "COLROW30.sbx"     "COLROW30.shp"     "COLROW30.shx"
##  [7] "g2006_2.dbf"      "g2006_2.shp"      "g2006_2.shp.xml"
## [10] "g2006_2.shx"      "SimU_all.dbf"     "SimU_all.prj"
## [13] "SimU_all.sbn"     "SimU_all.sbx"     "SimU_all.shp"
## [16] "SimU_all.shp.xml" "SimU_all.shx"     "version.txt"
## [19] "worldCR.dbf"      "worldCR.prj"      "worldCR.shp"
## [22] "worldCR.shx"      "worldLU.dbf"      "worldLU.prj"
## [25] "worldLU.shp"      "worldLU.shx"
```

To see the available countries in alphabetical order, use `getCountries()`, passing the data directory as argument. The code below shows the first ten countries.

```
colrow::getCountries(dataDir)[1:10]
```

```
##  [1] "Afghanistan"          "Aksai Chin"           "Albania"
##  [4] "Algeria"              "American Samoa"       "Andorra"
##  [7] "Angola"               "Anguilla"             "Antarctica"
## [10] "Antigua and Barbuda"
```

Then use `getLU()`, `getCR()`, and/or `getSimU()` to generate geospatial data for a given country. They get as first argument the country name written in the same way as one of the outputs of `getCountries()`. The second argument is the directory where the data downloaded from Dropbox was stored. Some of these functions might take time to execute, according to the selected arguments.

```
country <- "Brazil"

myLU   <- colrow::getLU(country, dataDir)
```

```
## Loading cached version of LU data for Brazil
```

```
myCR   <- colrow::getCR(country, dataDir)
```

```
## Loading cached version of CR data for Brazil
```

```
mySimU <- colrow::getSimU(country, dataDir)
```

```
## Reading all countries
## Selecting Brazil
## Reading all SimUs
## Subsetting SimUs
```

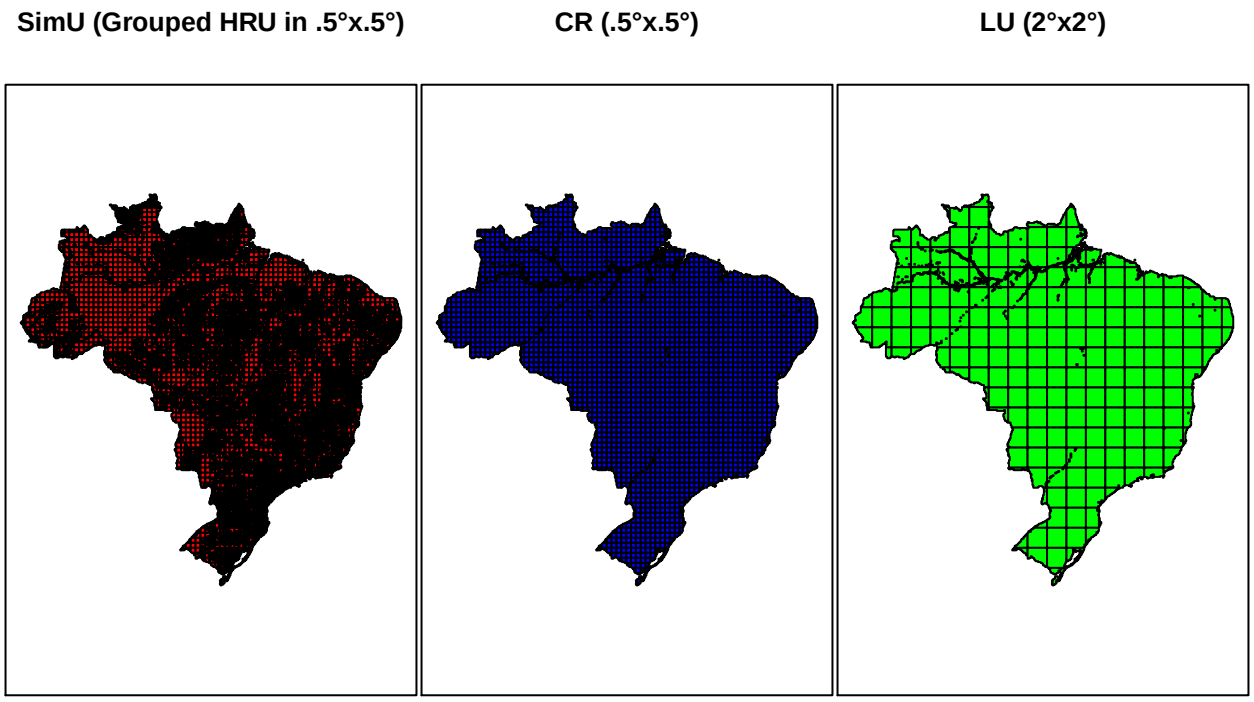
```
## Warning in st_is_longlat(x): bounding box has potentially an invalid value
## range for longlat data
```

```
## Joining SimUs
```

To see that everything is correct, plot the data.

```
par(mfrow = c(1, 3), mar = c(5, 0.1, 5, 0.1))

plot(sf::st_geometry(mySimU), main = "SimU (Grouped HRU in .5°x.5°)", col = "red"); box()
plot(sf::st_geometry(myCR),   main = "CR (.5°x.5°)", col = "blue"); box()
plot(sf::st_geometry(myLU),   main = "LU (2°x2°)", col ="green"); box()
```



Finally, save the data to be used by the processing functions. Each of these files has a column named `ID` that will be used to match the objects stored in CSV files. Once these files are saved, it is not necessary to create them again every time outputs are processed.

```
sf::write_sf(myLU, paste0(country, "LU.shp"))
sf::write_sf(myCR, paste0(country, "CR.shp"))
sf::write_sf(mySimU, paste0(country, "SimU.shp"))
```

Several countries together

The same functions above can be used to get several countries together. Just use the names of the countries as a string vector for the first argument. For example, the code below gets LU data for all nine countries of the Amazon basin.

```
amazon <- c("Brazil", "Peru", "Colombia", "Venezuela",
            "Ecuador", "Bolivia", "Guyana",
            "Suriname", "French Guiana")

lu <- colrow::getLU(amazon, dataDir)
```

```
## Loading cached version of LU data for Brazil
## Loading cached version of LU data for Peru
## Loading cached version of LU data for Colombia
## Loading cached version of LU data for Venezuela
## Loading cached version of LU data for Ecuador
## Loading cached version of LU data for Bolivia
## Loading cached version of LU data for Guyana
```

```
## Loading cached version of LU data for Suriname
## Loading cached version of LU data for French Guiana
```

To save the file, just use `write_sf()`:

```
sf::write_sf(lu, "amazonLU.shp")
```

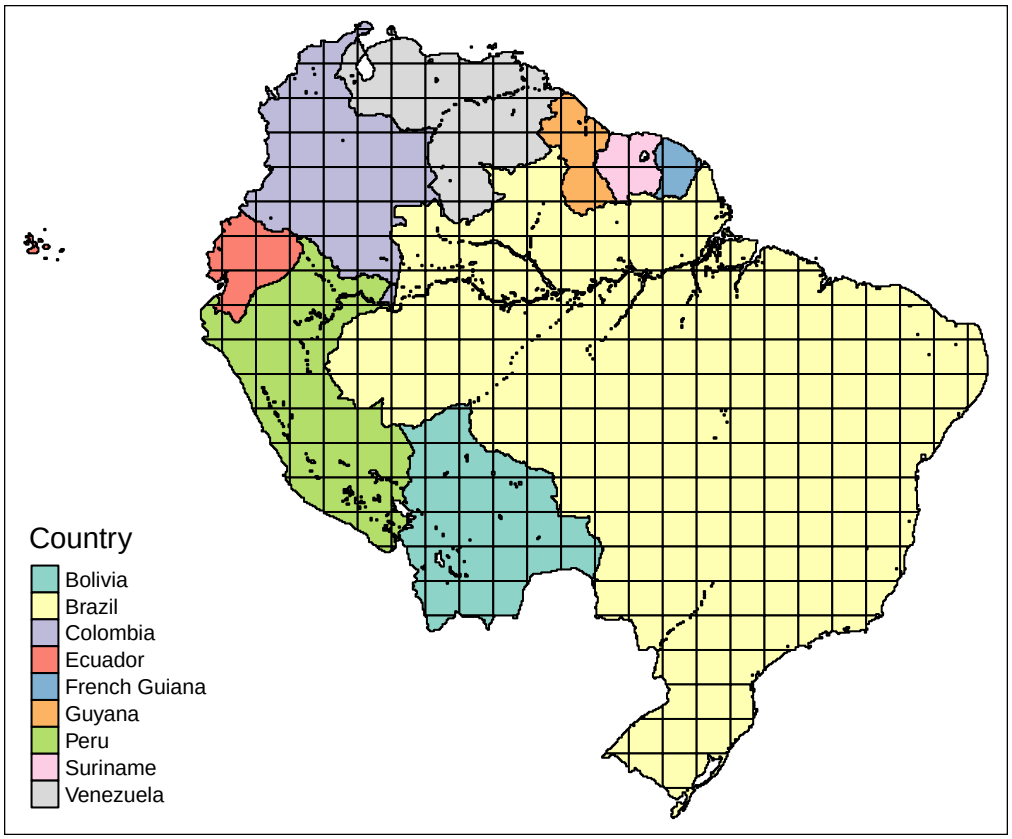
Then we can plot them together. This and several examples of plotting in this tutorial use `tmap` package.

```
require(tmap)
```

```
## Loading required package: tmap
```

```
tm_shape(lu) +
  tm_fill(col = "Country") +
  tm_borders(lwd = 1, col = "black")
```

```
## Linking to GEOS 3.6.1, GDAL 2.2.3, PROJ 4.9.3
```



It is important to note that, when working with more than one neighbor countries, there can exist two or more LU (or SimU, or CR) objects with the same ID, but belonging to different countries. In order to deal with that, colrow package returns also the name of the country. For example, there are two objects LU08515, one belonging to Brazil and the other to Peru:

```
lu %>%
  dplyr::filter(ID == "LU08515") %>%
  as.data.frame() %>%
  dplyr::select(Country)
```

```
##   Country
```

```
## 1  Brazil
## 2    Peru
```

Processing outputs

To process a CSV file, it is necessary to choose the created shapefile that matches the representation used in the CSV. Then, it is necessary to define the attribute names, as the CSV files do not store a header. These names must have one attribute called `ID` (to match the geometries) and another called `VALUE` (with the output value). Function `processFile()` gets the shapefile, the CSV file, and the description of the attributes as arguments. You can use `colrow::attrs()` to avoid quotes.

```
csvfile <- system.file("extdata/scenarios/FC/Land_Compare3_FC.csv", package = "colrow")
attributes <- colrow::attrs(COUNTRY, ID, ALTI, SLP, SOIL, USE, SCENARIO, YEAR, VALUE)

result <- colrow::processFile("BrazilCR.shp", csvfile, attributes)
```

```
## Reading shapefile: BrazilCR.shp
## Reading data file: C:/Users/pedro/Documents/R/win-library/3.6/colrow/extdata/scenarios/FC/Land_Compa
```

```
## 6 objects belong to the shapefile but not to the csv file: CR273185, CR273186, CR274185, CR274186, C
```

```
## Ignored attributes: COUNTRY, ALTI, SLP, SOIL, SCENARIO
## Attributes to be joined: USE, YEAR
## USE: CrpLnd, PriFor, NatLnd, ForReg, GrsLnd, MngFor, PltFor
## YEAR: 2000, 2010, 2020, 2030, 2040, 2050
## Spreading the data
## 40 attributes were created
## Merging the data
## Replacing 34340 NA values by zero
```

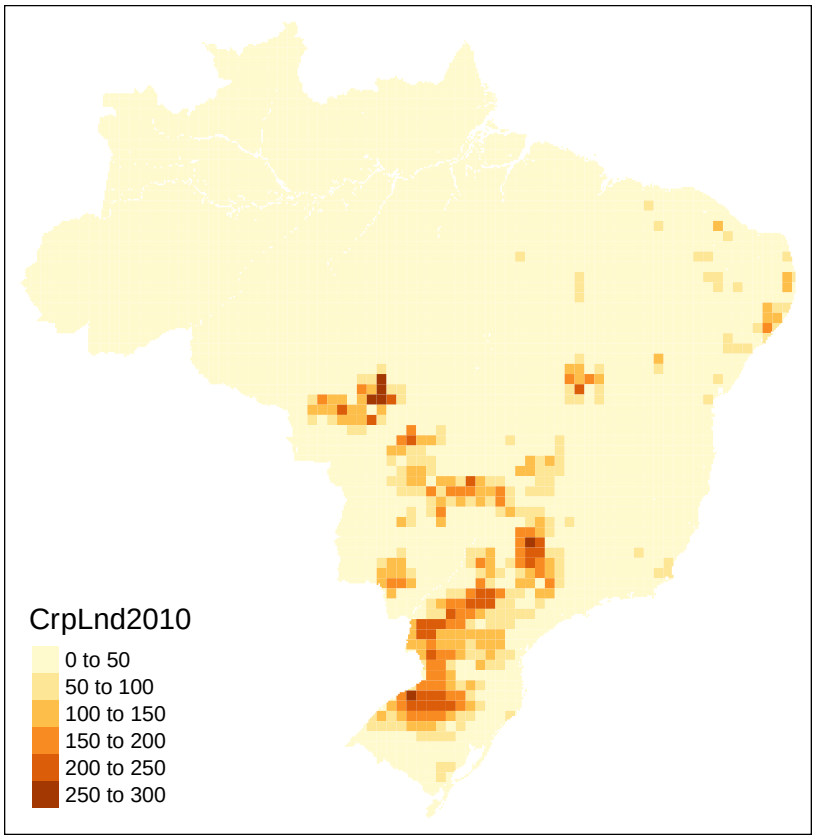
This function automatically ignores all attributes that have only one value (`COUNTRY`, `ALTI`, `SLP`, `SOIL` and `SCENARIO` above). The other attributes (except `ID` and `VALUE`) compose the created attribute names (`USE` and `YEAR` above). They are concatenated according to their order in the CSV file. See the names of the attributes below.

```
names(result)
```

```
##  [1] "ID"         "Country"    "geometry"   "CrpLnd2000" "CrpLnd2010"
##  [6] "CrpLnd2020" "CrpLnd2030" "CrpLnd2040" "CrpLnd2050" "ForReg2030"
## [11] "ForReg2040" "ForReg2050" "GrsLnd2000" "GrsLnd2010" "GrsLnd2020"
## [16] "GrsLnd2030" "GrsLnd2040" "GrsLnd2050" "MngFor2000" "MngFor2010"
## [21] "MngFor2020" "MngFor2030" "MngFor2040" "MngFor2050" "NatLnd2000"
## [26] "NatLnd2010" "NatLnd2020" "NatLnd2030" "NatLnd2040" "NatLnd2050"
## [31] "PltFor2000" "PltFor2010" "PltFor2020" "PltFor2030" "PltFor2040"
## [36] "PltFor2050" "PriFor2000" "PriFor2010" "PriFor2020" "PriFor2030"
## [41] "PriFor2040" "PriFor2050"
```

It is possible to plot a map selecting a given attribute using tmap package:

```
tm_shape(result) +
  tm_fill(col = "CrpLnd2010")
```



It is also possible to use `processFile()` to save the output when processing a file by adding a fourth argument with the output file name. In this case, this function does not return anything.

```
colrow::processFile(
  "BrazilCR.shp",
  csvfile,
  attributes,
  "brazilLandCompare.shp"
)
```

```
## Reading shapefile: BrazilCR.shp
## Reading data file: C:/Users/pedro/Documents/R/win-library/3.6/colrow/extdata/scenarios/FC/Land_Compa
## 6 objects belong to the shapefile but not to the csv file: CR273185, CR273186, CR274185, CR274186, C
## Ignored attributes: COUNTRY, ALTI, SLP, SOIL, SCENARIO
## Attributes to be joined: USE, YEAR
## USE: CrpLnd, PriFor, NatLnd, ForReg, GrsLnd, MngFor, PltFor
## YEAR: 2000, 2010, 2020, 2030, 2040, 2050
## Spreading the data
## 40 attributes were created
## Merging the data
## Replacing 34340 NA values by zero
## Renaming attributes according to convertList
## No attribute has more than 10 characters
## Writing output file: brazilLandCompare.shp
```

Note that shapefiles have limits: 255 or less columns, ten or less characters in the attribute names. When

working with large attribute names, it is necessary to simplify the names. In the example below, seven names will be replaced by shorter names. For instance, `CrpLnd` will be replaced by `cr` in the attribute names.

```
convert <- list(
  CrpLnd = "cr", PriFor = "pr",
  NatLnd = "nl", ForReg = "fr",
  GrsLnd = "gl", MngFor = "mf",
  PltFor = "pl"
)
```

If you want to simplify the years to two characters, just run:

```
for(year in paste(seq(2000, 2050, 10))) # from 2000, 2010, ..., 2050
  convert[[year]] = substr(year, 3, 4) # to 00, 10, ..., 50

unlist(convert)
```

```
## CrpLnd PriFor NatLnd ForReg GrsLnd MngFor PltFor   2000   2010   2020
##   "cr"   "pr"   "nl"   "fr"   "gl"   "mf"   "pl"   "00"   "10"   "20"
##   2030   2040   2050
##   "30"   "40"   "50"
```

Finally, these simplifications can be used as fifth argument to `processFile()`.

```
colrow::processFile(
  "BrazilCR.shp",
  csvfile,
  attributes,
  "brazilOutput.shp",
  convert
)
```

```
## Reading shapefile: BrazilCR.shp
## Reading data file: C:/Users/pedro/Documents/R/win-library/3.6/colrow/extdata/scenarios/FC/Land_Compa
## 6 objects belong to the shapefile but not to the csv file: CR273185, CR273186, CR274185, CR274186, C
## Ignored attributes: COUNTRY, ALTI, SLP, SOIL, SCENARIO
## Attributes to be joined: USE, YEAR
## USE: CrpLnd, PriFor, NatLnd, ForReg, GrsLnd, MngFor, PltFor
## YEAR: 2000, 2010, 2020, 2030, 2040, 2050
## Spreading the data
## 40 attributes were created
## Merging the data
## Replacing 34340 NA values by zero
## Renaming attributes according to convertList
## No attribute has more than 10 characters
## Writing output file: brazilOutput.shp
```

Sometimes there are more than one `VALUE` per unique `ID`, for each unique combination of attributes. In this case, it is necessary to transform all the values into one single result. As default, `processFile()` sums all the values, but it is possible to use any other function passed as argument, such as `mean()`. Note that `sum()` is used for absolute values and `mean()` for averages. To use another function, use argument `aggregate`, passing a function as value, as shown in the example below.

```
result <- colrow::processFile(
  "BrazilCR.shp",
  system.file("extdata/csv/YIELD_COMPARE2.CSV", package = "colrow"),
  colrow::attrs(ID, CROP, ScenYear, VALUE),
```

```
  aggregate = sum # default value, could be omitted
)
```

```
## Reading shapefile: BrazilCR.shp
## Reading data file: C:/Users/pedro/Documents/R/win-library/3.6/colrow/extdata/csv/YIELD_COMPARE2.CSV
## 177 objects belong to the shapefile but not to the csv file: CR214192, CR214193, CR218189, CR220180,
## Ignored attributes:
## Attributes to be joined: CROP, ScenYear
## CROP: Corn, Rice
## ScenYear: 2000, 2010
## Spreading the data
## 5 attributes were created
## Merging the data
## Replacing 1322 NA values by zero
```

Plotting outputs

To plot the outputs created above, first we read the Brazilian Biomes to be drawn on top of the ColRows. The shapefile with this data is stored in file `br_biomes.shp` available within `colrow` package. We will use CR data from `brazilLandCompare.shp`, created previously in this tutorial.

```
brazil <- sf::read_sf("brazilLandCompare.shp")
biomes <- system.file("extdata/shape", "br_biomes.shp", package = "colrow") %>% sf::read_sf()
```

The next step is to select a palette from the functions availables in `RColorBrewer`, using `brewer.pal()` (see the description of all colors available here), and to choose the cuts to represent the intervals of each slice. In the code below, as we have seven cuts, we need to define six colors.

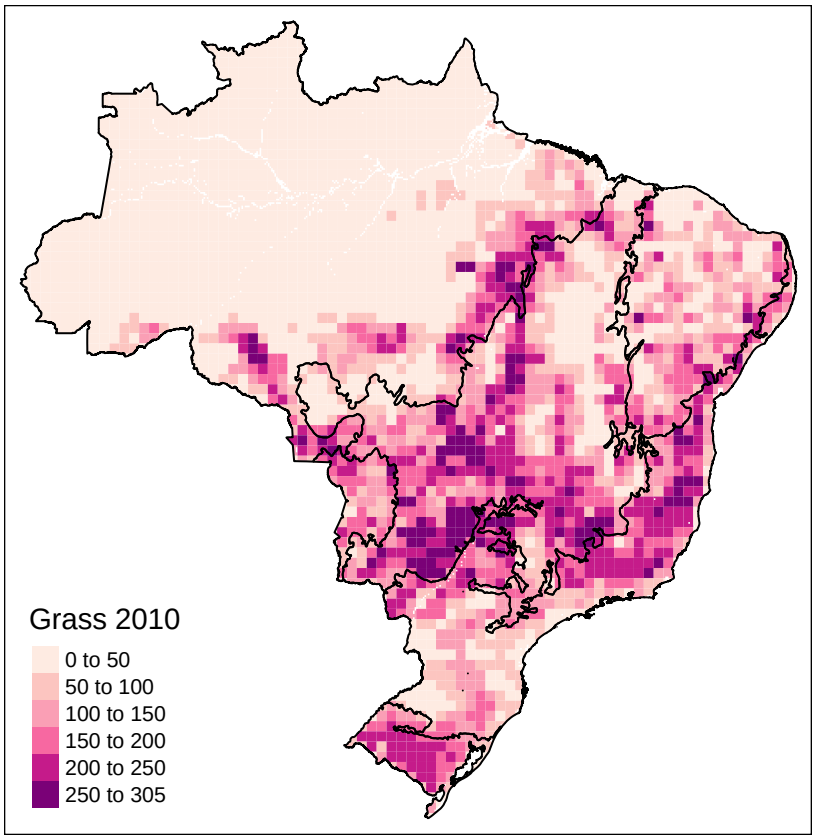
```
max(brazil$GrsLnd2010)
```

```
## [1] 303.33
```

```
cuts <- c(0, 50, 100, 150, 200, 250, 305)
rdPu <- RColorBrewer::brewer.pal(6, "RdPu")
```

Finally, use `tm_shape()`, `tm_fill()`, and `tm_borders()` to draw the map.

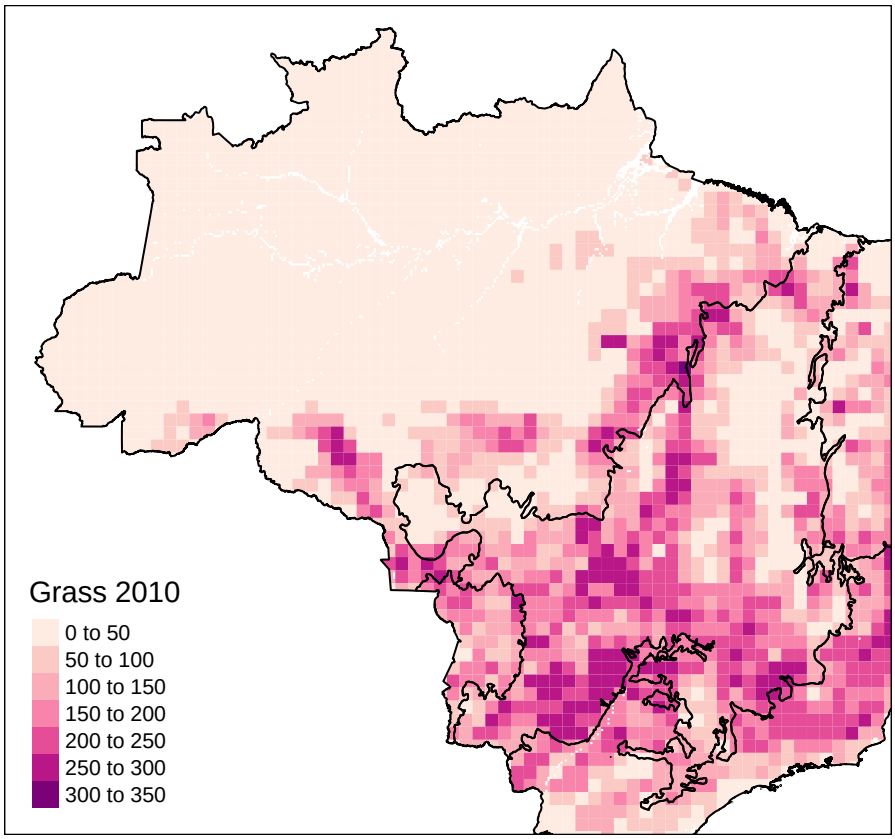
```
tm_shape(brazil) +
  tm_fill(col = "GrsLnd2010", palette = rdPu, breaks = cuts, title = "Grass 2010") +
  tm_shape(biomes) +
  tm_borders(lwd = 1, col = "black")
```



If only part of the map is necessary, it is possible to zoom in the map to a given region by selecting a `bbox` to the `tm_shape()`:

```
amazCerradoBox <- c(xmin = -74.5, xmax = -41.4, ymin = -25, ymax = 5.5) %>%
  sf::st_bbox(crs = st_crs(4326))

tm_shape(brazil, bbox = amazCerradoBox) +
  tm_fill(col = "GrsLnd2010", palette = rdPu, title = "Grass 2010") +
  tm_shape(biomes) +
  tm_borders(lwd = 1, col = "black")
```



To save this or any other plot created within R into a file using tmap package, it is necessary to store the plot into a variable and then call `tmap_save()`. For example, the following code creates variable `result` storing the same map created above and then saves it into file `amz-cerrado.png` in the current directory.

```
result <- tm_shape(brazil, bbox = amazCerradoBox) +
  tm_fill(col = "GrsLnd2010", palette = rdPu, title = "Grass 2010") +
  tm_shape(biomes) +
  tm_borders(lwd = 1, col = "black")

tmap_save(result, "amaz-cerrado.png")
```

```
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
```

```
## Map saved to C:\Users\pedro\github\colrow\vignettes\amaz-cerrado.png
```

```
## Resolution: 2171.821 by 2030.554 pixels
```

```
## Size: 7.239403 by 6.768513 inches (300 dpi)
```

It is also possible to create several maps using `plotAll()`. This function automatically uses all the values of several attributes with the same prefix in order to define a common legend, with the same maximum and minimum values. The following code creates six pdf files, each one plotting grassland, from 2000 to 2050.

```
require(tmap)

plotAll(brazil, "GrsLnd", palette = "RdPu", title = "Grass", additional =
  tm_shape(biomes) +
  tm_borders(lwd = 1, col = "black")
```

```
)
```

```
## Processing attribute 'GrsLnd2000'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2000.pdf
## Size: 6.902778 by 7.083333 inches
## Processing attribute 'GrsLnd2010'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2010.pdf
## Size: 6.902778 by 7.083333 inches
## Processing attribute 'GrsLnd2020'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2020.pdf
## Size: 6.902778 by 7.083333 inches
## Processing attribute 'GrsLnd2030'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2030.pdf
## Size: 6.902778 by 7.083333 inches
## Processing attribute 'GrsLnd2040'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2040.pdf
## Size: 6.902778 by 7.083333 inches
## Processing attribute 'GrsLnd2050'
## Warning in png(tmp, width = width, height = height, res = res): 'width=7,
## height=7' are unlikely values in pixels
## Map saved to C:\Users\pedro\github\colrow\vignettes\GrsLnd2050.pdf
## Size: 6.902778 by 7.083333 inches
```

Handling CSV files

You can handle CSV files directly to create charts or to compute values without creating shapefiles. First read data as a tibble using `readCSV()`:

```
data <- colrow::readCSV(
  system.file("extdata/scenarios/FC/Land_Compare3_FC.csv", package = "colrow"),
  colrow::attrs(COUNTRY, ID, ALTICLASS, SLPCLASS, SOILCLASS, USE, SCENARIO, YEAR, VALUE)
)
```

To get the unique values per class, just use `unique()`:

```
unique(data$USE)
```

```
## [1] "CrpLnd" "PriFor" "NatLnd" "ForReg" "GrsLnd" "MngFor" "PltFor"
```

```
unique(data$YEAR)
```

```
## [1] 2000 2010 2020 2030 2040 2050
```

To check if the sum of all land use areas are the same for each year use some `dplyr` functions:

```
result <- data %>%
  dplyr::group_by(YEAR) %>%
  dplyr::summarise(VALUE = sum(VALUE))

result
```

```
## # A tibble: 6 x 2
##    YEAR   VALUE
##   <dbl>   <dbl>
## 1  2000 584619.
## 2  2010 584619.
## 3  2020 584619.
## 4  2030 584618.
## 5  2040 584618.
## 6  2050 584618.
```

Then let us now compute the total of each land use area per year and rename the attributes in order to use them directly in the plot.

```
result <- data %>%
  dplyr::group_by(YEAR, USE) %>%
  dplyr::summarise(VALUE = sum(VALUE)) %>%  # sum by year/use
  dplyr::mutate(Year = YEAR, Use = USE) %>% # rename variables
  dplyr::mutate(Use = dplyr::recode(Use,    # rename attributes
    CrpLnd = "Crop Land",
    ForReg = "Forest Regrowth",
    GrsLnd = "Grass Land",
    MngFor = "Managed Forest",
    NatLnd = "Natural Land",
    PltFor = "Planted Forest",
    PriFor = "Primary Forest"))

result
```

```
## # A tibble: 39 x 5
## # Groups:   YEAR [6]
##     YEAR USE      VALUE  Year Use
##    <dbl> <chr>    <dbl> <dbl> <chr>
##  1  2000 CrpLnd  39983.  2000 Crop Land
##  2  2000 GrsLnd 214941.  2000 Grass Land
##  3  2000 MngFor  41955.  2000 Managed Forest
##  4  2000 NatLnd  73001.  2000 Natural Land
##  5  2000 PltFor   6005.  2000 Planted Forest
##  6  2000 PriFor 208736.  2000 Primary Forest
##  7  2010 CrpLnd  60109.  2010 Crop Land
##  8  2010 GrsLnd 240326.  2010 Grass Land
##  9  2010 MngFor  45416.  2010 Managed Forest
```

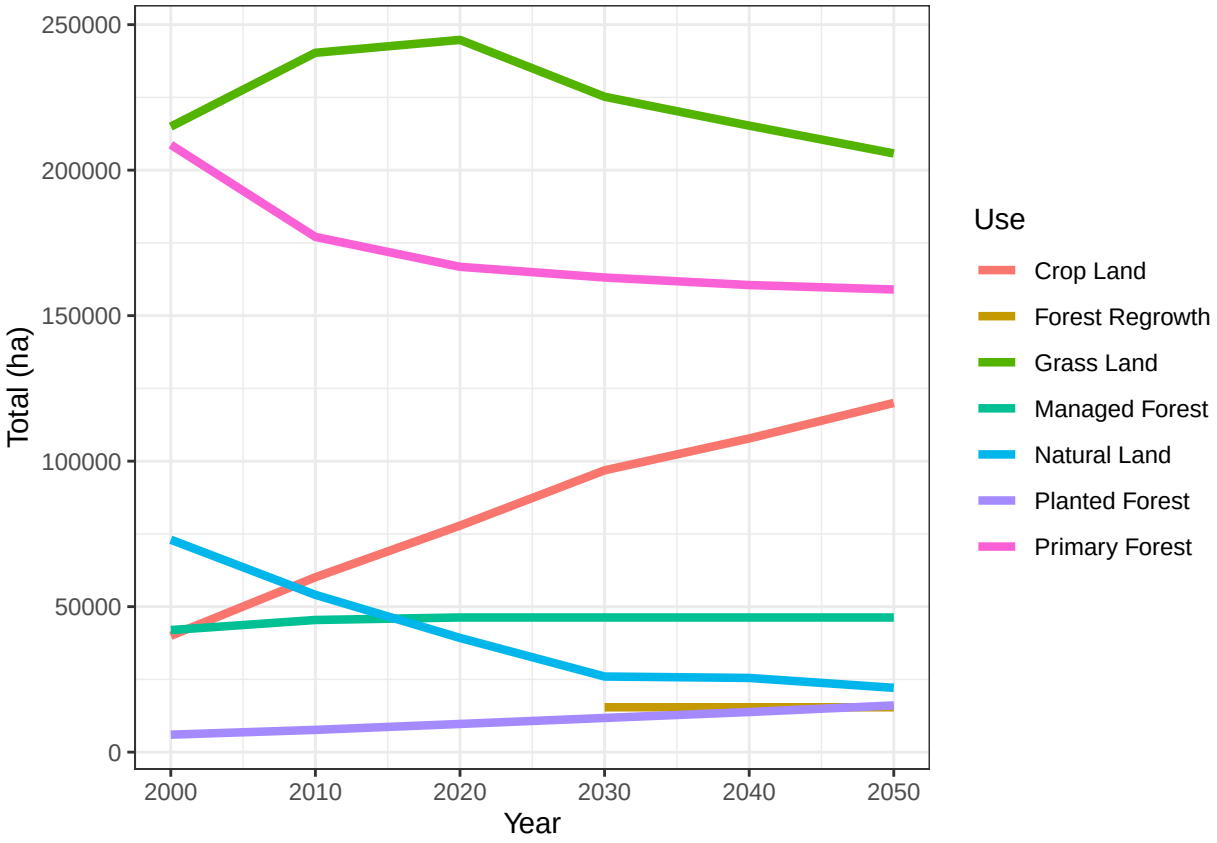
```
## 10  2010 NatLnd  54057.  2010 Natural Land
## # ... with 29 more rows
```

Finally, just use `ggplot()` to draw the evolution of each land use area per year.

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
ggplot(result) +
  aes(x = Year, y = VALUE, colour = Use) +
  geom_line(lwd = 1.5) +
  theme_bw() +
  ylab("Total (ha)")
```



Final remarks

If you have any suggestions or want to report an error, please visit the GitHup page of the package here.