Getting started with Rgis - V.0.1

Ivan Lizarazo
Faculty of Engineering,
Universidad Distrital,
Bogota,
Colombia
ilizarazo@udistrital.edu.co

August 2013

Abstract

This document provides examples and practicals tips for using *R*, the free software environment for statistical computing, as a GIS tool. After an overview of spatial data representation and spatial packages in R, the following topics are covered: (i) visualization of geographic data; (ii) accessing geoservices; and (iii) basic analysis of spatial data sets. As its title suggests, this document is just a starter guide. It was created using Sweave, a tool that allows to embed the *R* code in LaTeX.

1 Introduction

This tutorial requires a recent version of *R* software. As an example, this document's author used R version 2.15.3 "Security Blanket" installed in a Linux machine (Ubuntu 12.04). In order to reproduce these exercises previous installation of following R packages is required: *rgdal*, *sp*, *raster*,*rasterVIS*, *rworldmap*, *RgoogleMaps*.

This tutorial doesn't require any prior knowledge of *R* (though *R* users will likely feel at home quite quickly). Previous exposure to GIS concepts and operations, both vector and raster, is advantageous.

A number of web pages and documents have been consulted in order to grasp `Rgis` concepts. A lot of ideas and code have been borrowed from:

- Introduction to spatial data handling in R by Robert J. Hijmans (2013)
- rworldmap: A New R package for Mapping Global Data by Andy South (2011)
- RgoogleMaps package: Vignette: Plotting on Google Static Maps in R
- `http://geonames.r-forge.r-project.org/`
- `http://worldgrids.org/doku.php?id=wiki:functions`
- `http://gsoc2010r.wordpress.com/2010/06/10/rgeos-introduction`

2 Preliminary work

Start a work session in R. Then, set up your working directory and load basic packages using *R* command line client:

```
> ### use your own path ###
> setwd("~/ud/pdi avanzado/R/R as a GIS")
> ### Basic packages ###
> library(rgdal)          # geospatial data abstraction library
> library(sp)             # classes for spatial data
> library(raster)         # grids, rasters
> library(rasterVis)      # raster visualisation
> library(maptools)       # and their dependencies
```

3 Spatial objects and spatial packages in R

Spatial data in R are handled in complex object classes. The *sp* package defines a complete set of spatial classes, from points to lines to polygons (each possibly with attributes) to grids and pixels. *sp* classes have names that start with **Spatial**. The basic types are **SpatialPoints**, **SpatialLines**, **SpatialPolygons**, **SpatialGrid** (raster), and **SpatialPixels** (sparse raster). These classes only represent geometries. To also store attributes, extended classes are available such as **SpatialPolygonsDataFrame**, and **SpatialGridDataFrame**. The *raster* package extends the *sp* classes to include **Rasterlayer**, **RasterStack**, and **RasterBrick**, and provides tools for automatic tiling of raster objects too large to fit into memory.

In most cases, users do not create spatial objects with R code. Users probably read them from a file (e.g. a shapefile or a TIFF). Shapefiles, for example, can be read using function `readOGR()` in the `rgdal` package or function `shapefile()` in the `raster` package.

However, for the sake of illustration, let's create a few spatial objects:

```
> library(sp)
> lon=c(-73,-74.5,-72.3,-76.6)
> lat=c(7,4.5,11.3,5.7)
> # SpatialPoints
> sp1 <- SpatialPoints(cbind(lon, lat))
> class(sp1)
```

```
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

```
> str(sp1)
```

```
Formal class 'SpatialPoints' [package "sp"] with 3 slots
  ..@ coords      : num [1:4, 1:2] -73 -74.5 -72.3 -76.6 7 4.5 11.3 5.7
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "lon" "lat"
  ..@ bbox        : num [1:2, 1:2] -76.6 4.5 -72.3 11.3
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "lon" "lat"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

```
> # data frame
> df <- data.frame(precip=c(1442,1371,765,7480),
+ city=c("BUCARAMANGA","BOGOTA","MAICAO","QUIBDO"))
> class(df)
```

```
[1] "data.frame"
```

```
> str(df)
```

```
'data.frame':   4 obs. of  2 variables:
 $ precip: num  1442 1371 765 7480
 $ city  : Factor w/ 4 levels "BOGOTA","BUCARAMANGA",..: 2 1 3 4
```

```
> # SpatialPointsDataFrame
> sp2 <- SpatialPointsDataFrame(sp1, data=df)
> class(sp2)
```

```
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

```
> str(sp2)
```

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 4 obs. of  2 variables:
  .. ..$ precip: num [1:4] 1442 1371 765 7480
  .. ..$ city  : Factor w/ 4 levels "BOGOTA","BUCARAMANGA",..: 2 1 3 4
  ..@ coords.nrs : num(0)
  ..@ coords     : num [1:4, 1:2] -73 -74.5 -72.3 -76.6 7 4.5 11.3 5.7
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : NULL
  .. .. ..$ : chr [1:2] "lon" "lat"
  ..@ bbox       : num [1:2, 1:2] -76.6 4.5 -72.3 11.3
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "lon" "lat"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

```r
> #
> # Spatial Polygons
> lon <- c(-75, -72, -72, -75)
> lat <- c(7.5, 7.5, 4.25, 4.25)
> coord <- cbind(lon, lat)
> # close the ring of the polygon
> coord <- rbind(coord, coord[1,])
> poly <- SpatialPolygons(list( Polygons(list(Polygon(coord)), 1)))
> str(poly)
```

```
Formal class 'SpatialPolygons' [package "sp"] with 4 slots
  ..@ polygons   :List of 1
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] -73.5 5.88
  .. .. .. .. .. .. ..@ area   : num 9.75
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:5, 1:2] -75 -72 -72 -75 -75 7.5 7.5 4.25 4.25 7.5
  .. .. .. .. .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. .. .. .. .. ..$ : NULL
  .. .. .. .. .. .. .. .. ..$ : chr [1:2] "lon" "lat"
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] -73.5 5.88
  .. .. .. ..@ ID       : chr "1"
  .. .. .. ..@ area     : num 9.75
  ..@ plotOrder  : int 1
  ..@ bbox       : num [1:2, 1:2] -75 4.25 -72 7.5
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr NA
```

```r
> class(poly)
```

```
[1] "SpatialPolygons"
attr(,"package")
[1] "sp"
```

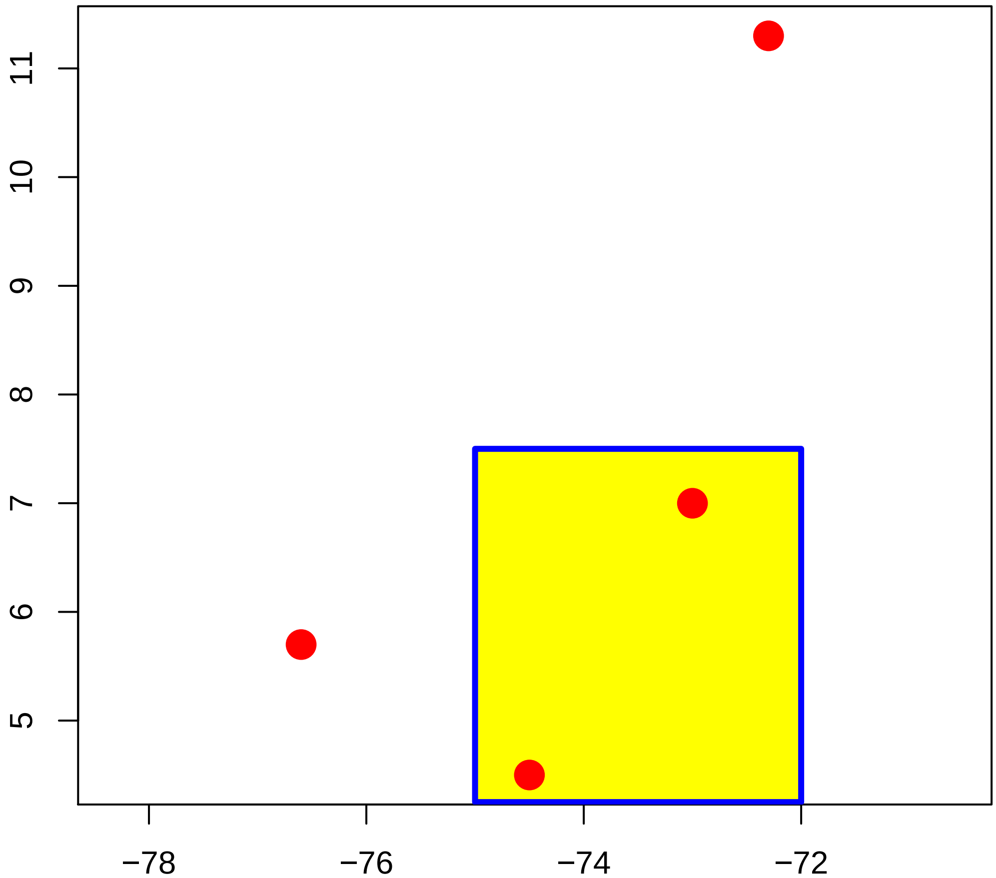
```r
> bbox(poly)
```

```
     min   max
x -75.00 -72.0
y   4.25   7.5
```

```r
> proj4string(poly)
```

```
[1] NA
```

```r
> #
> plot(sp2, axes=TRUE)
> plot(poly, border='blue', col='yellow', lwd=3, add=TRUE)
> points(sp2, col='red', pch=20, cex=3)
```



All spatial objects have 2 slots or components in common: a bounding box in *sp* (or an extent in *raster*) and a coordinate reference system (CRS). All useful spatial classes have additional slots: lists of coordinates for points, coordinates of vertices for polygons, descriptions of dimensions and a matrix of values for rasters, etc. The `str()` function returns the structure of an object.

The bounding box of an object can be computed on the fly using the `bbox()` function. The coordinate reference system must either be read or set explicitly on input spatial objects, and defaults to NA. Then, because all spatial operations require objects in the same CRS, if the result of a spatial operation is a spatial object, it inherits the same CRS as the input objects.

CRS uses proj4 strings to define projections coordinate reference systems. If you know such parameters, you can use them: the `proj4string()` function on the left of an assignment statement can be used to set the CRS of a spatial object. However, when the `readOGR()` and `readGDAL()` functions read a file with projection information (.prj for a shapefile, embedded in .img files, etc.), the resulting R object has the correct CRS proj4string. `spTransform()` reprojects spatial objects; the target CRS is usually the CRS of another R spatial object, and can be set accordingly.

While `sp` package is well suited for managing vector objects, the `raster` package focuses on raster objects. A RasterLayer object represents single-layer raster data. A RasterLayer object stores fundamental parameters that describe itself, e.g. number of columns and rows, the coordinates of its spatial extent, and the CRS. In addition, a RasterLayer can store information about the file in which the raster cell values are stored (if such a file exists). A RasterLayer can also hold the raster cell values in memory. Multiple layers can be represented by a RasterStack and by a RasterBrick. While these are very similar objects, a RasterStack can refer to different files (all with the same extent and resolution), whereas a RasterBrick can only point to a single file.

Let's create a raster object from scratch.

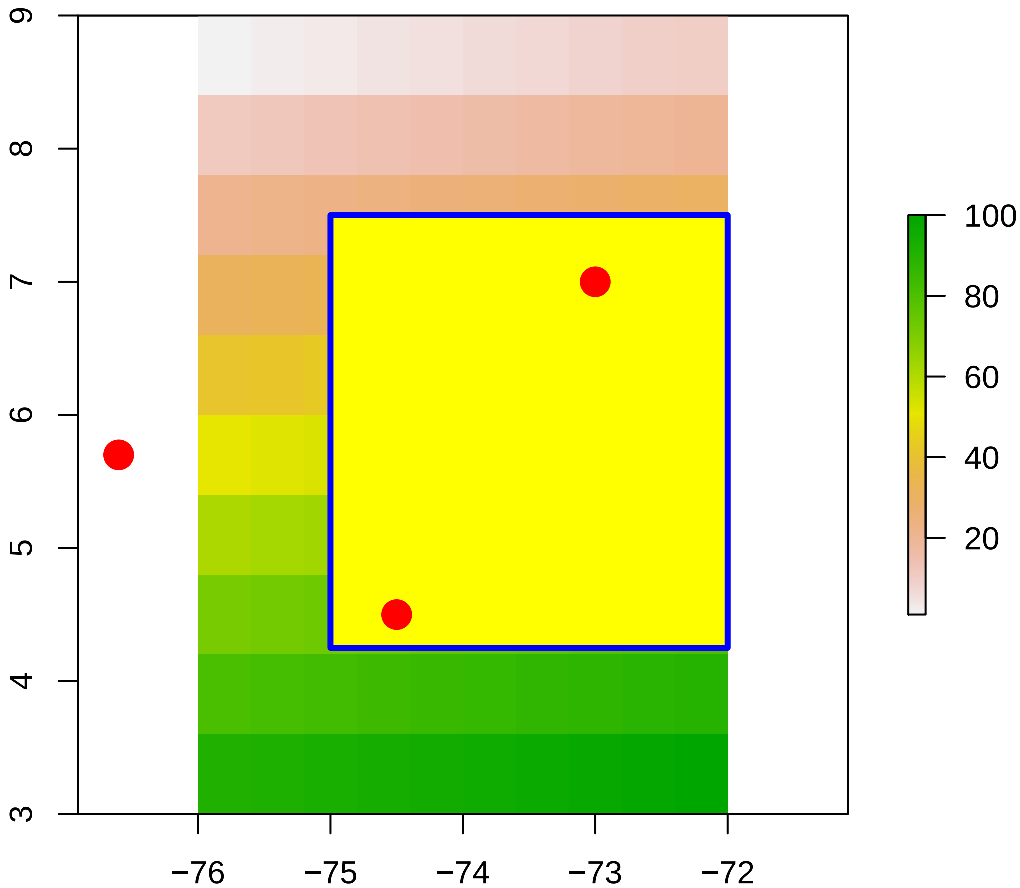
```
> library(raster)
> # create empty RasterLayer
> r1 <- raster(ncol=10, nrow=10, xmx=-72, xmn=-76, ymn=3, ymx=9)
> r1
```

```
class       : RasterLayer
dimensions  : 10, 10, 100  (nrow, ncol, ncell)
resolution  : 0.4, 0.6  (x, y)
extent      : -76, -72, 3, 9  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
> # assign values
> r1[] <- 1:ncell(r1)
> r1
```

```
class       : RasterLayer
dimensions  : 10, 10, 100  (nrow, ncol, ncell)
resolution  : 0.4, 0.6  (x, y)
extent      : -76, -72, 3, 9  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 1, 100  (min, max)
```

```
> # plot
> plot(r1)
> # add polygon and points
> plot(poly, border='blue', col='yellow', lwd=3, add=TRUE)
> points(sp2, col='red', pch=20, cex=3)
> #
```



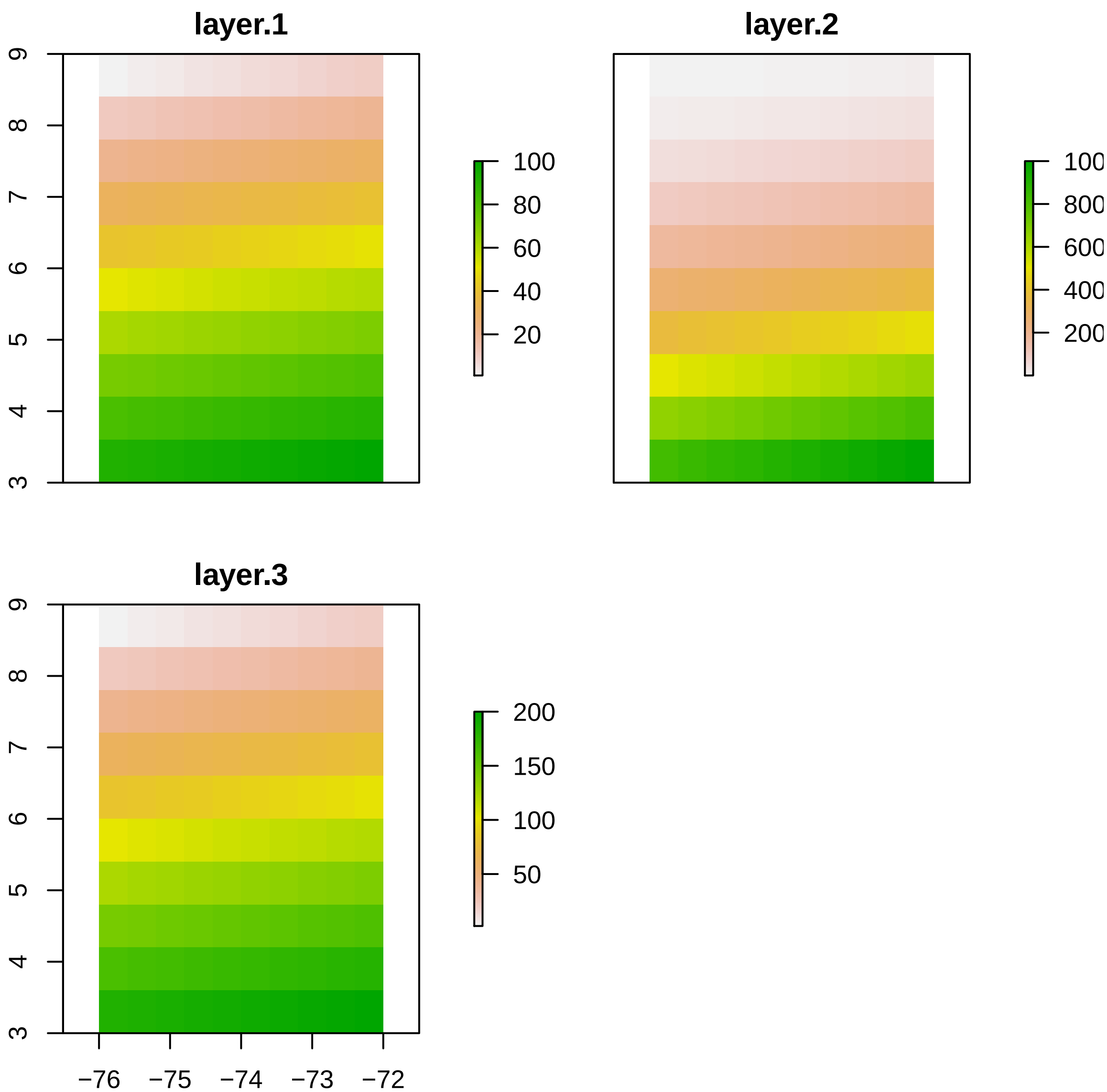
Let's make a RasterStack from multiple layers:

```
> r2 <- r1 * r1
> r3 <- 2 * r1
> s <- stack(r1, r2, r3)
> s
```

```
class       : RasterStack
dimensions  : 10, 10, 100, 3  (nrow, ncol, ncell, nlayers)
resolution  : 0.4, 0.6  (x, y)
extent      : -76, -72, 3, 9  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
names        : layer.1, layer.2, layer.3
min values   :       1,       1,       2
max values   :     100,   10000,     200
```

```
> plot(s)
```



4 Visualization of geographic data

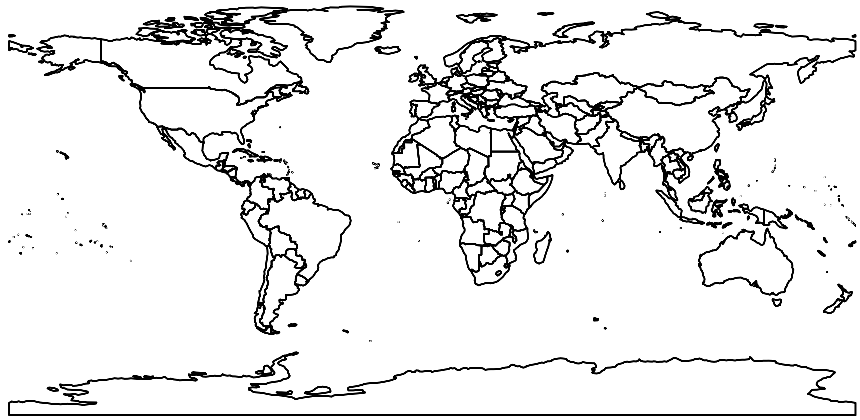
4.1 Mapping data using rworldmap

rworldmap is a package available on CRAN for mapping and visualization of global data. *rworldmap* has three core functions:

- **jointCountryData2map()** joins user country data referenced by country names or codes to a map to enable plotting
- **mapCountryData()** plots a map of country data
- **mapGriddedData()** plots a map of gridded data

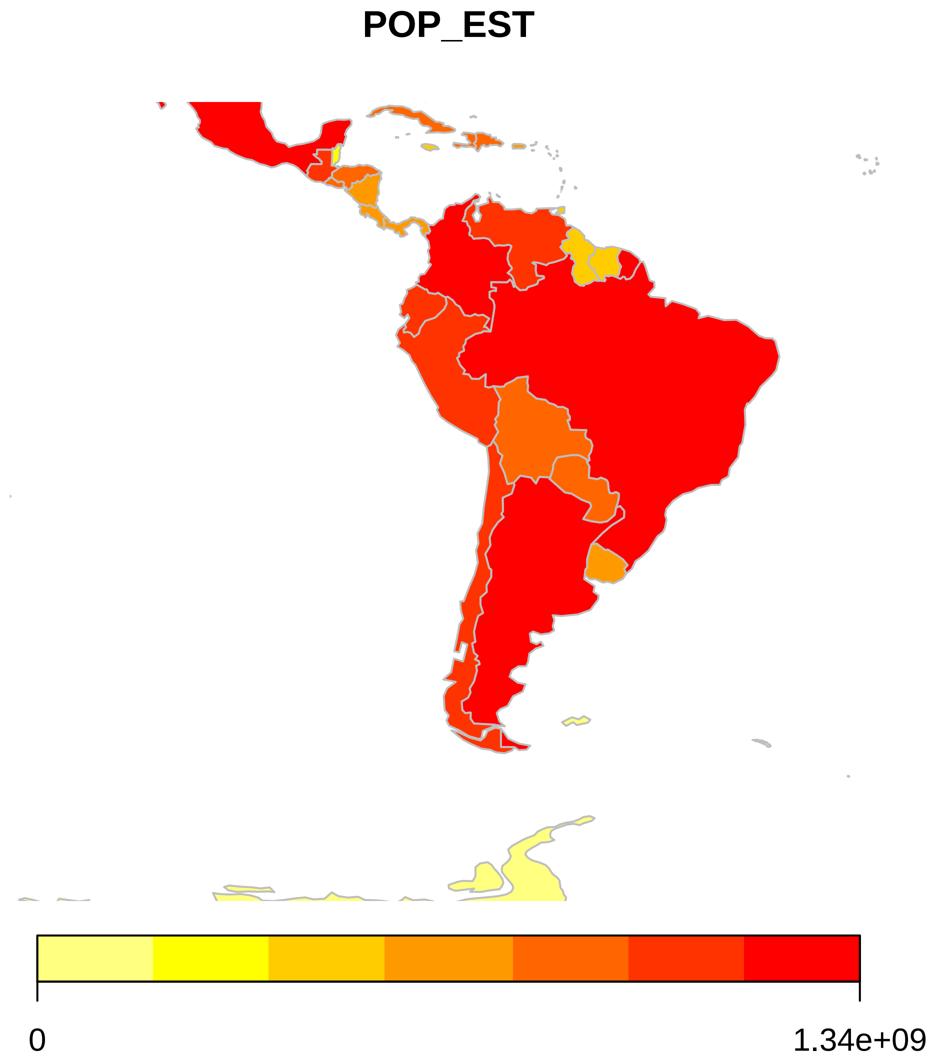
As for our exercise we will conduct three steps: (a) load the rworldmap package; (b) get the self-contained worldwide countries dataset, an object of type *SpatialPolygonsDataFrame*, at a desired spatial resolution; and (c) plot the corresponding map:

```
> library(rworldmap)
> # examples:
> newmap <- getMap(resolution = "coarse")  # different resolutions available
> plot(newmap, main="Hello world!")
```



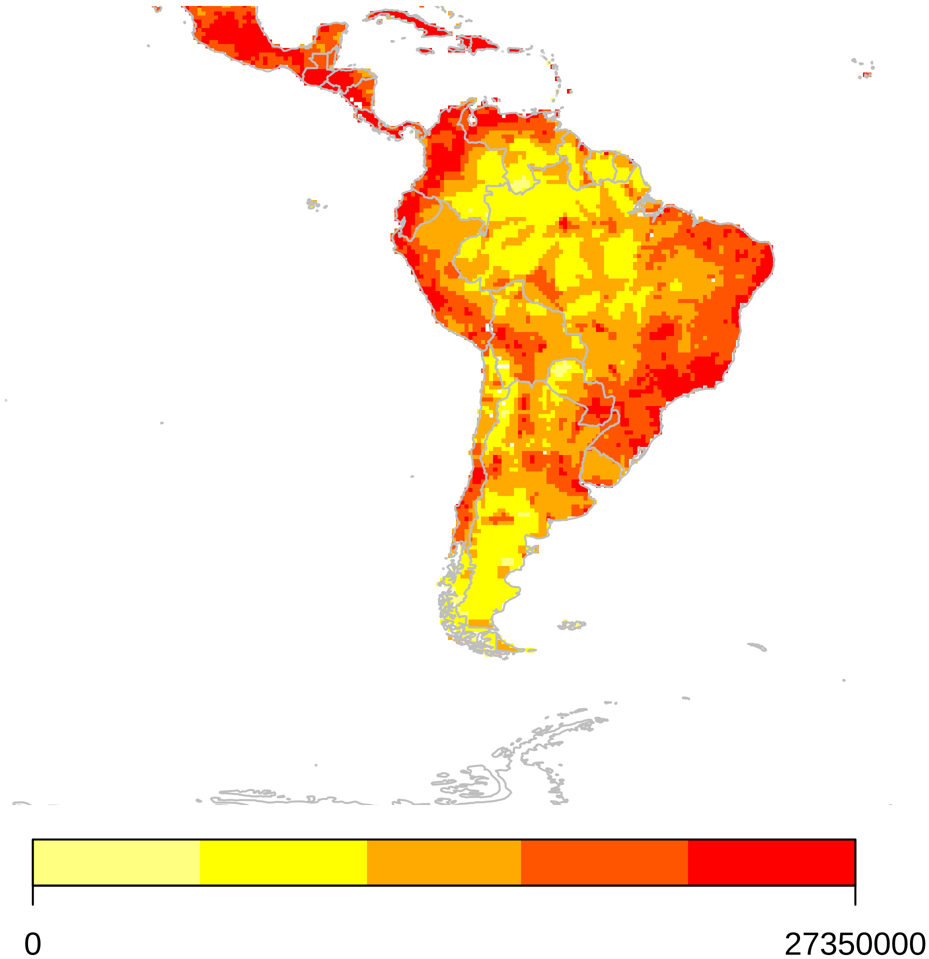
It is possible to map a subset of countries and show estimated population:

```
> mapCountryData(mapRegion = "latin america")
```



It is also possible to visualize gridded data using the *SpatialGridDataFrame* included in the *worldmap* package:

```
> data(gridExData)
> # mapDevice()
> mapGriddedData(mapRegion = "latin america")
```



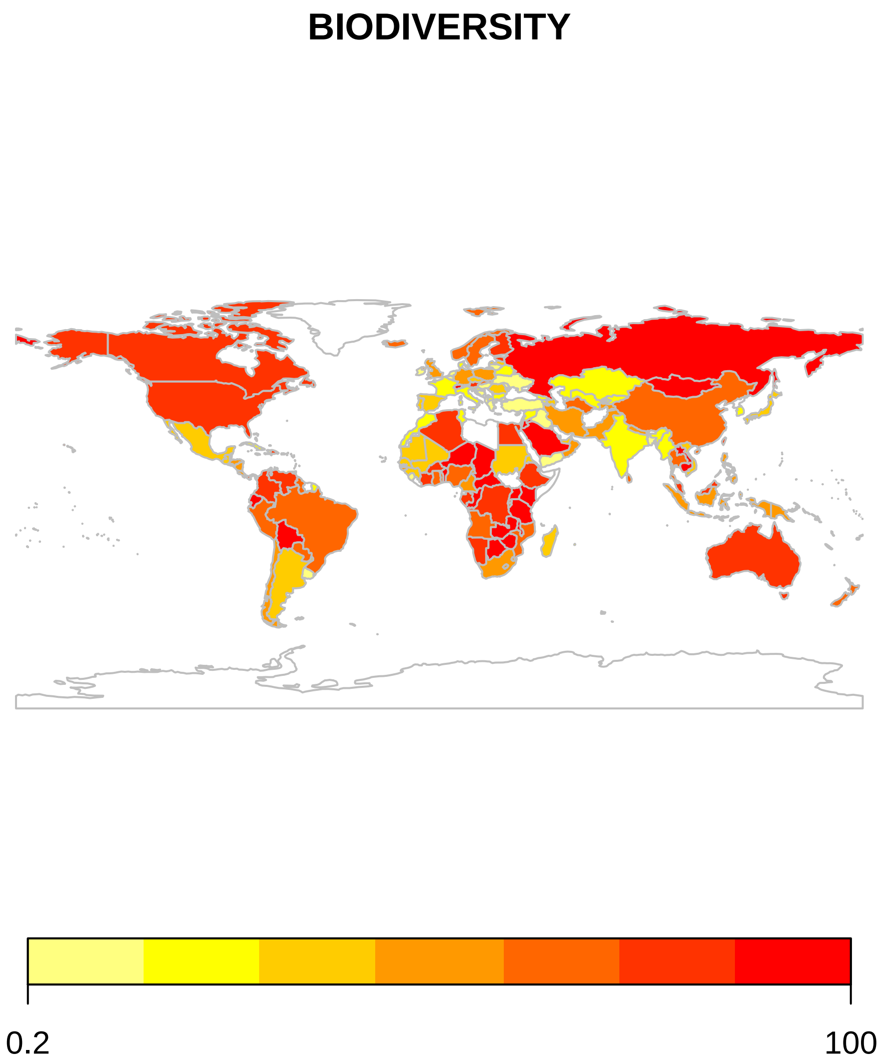
To map anything other than the default map, **mapCountryData()** requieres an object of class *SpatialPolygonsDataFrame* and a specification of the name of the column containing the data ato plot. This code allows plotting biodiversity categories:

```
> data(countryExData)
> sPDF <- joinCountryData2Map(countryExData, joinCode="ISO3",
+ nameJoinColumn="ISO3V10")
```

```
149 codes from your data successfully matched countries in the map
0 codes from your data failed to match with a country code in the map
```

```
94 codes from the map weren't represented in your data
```

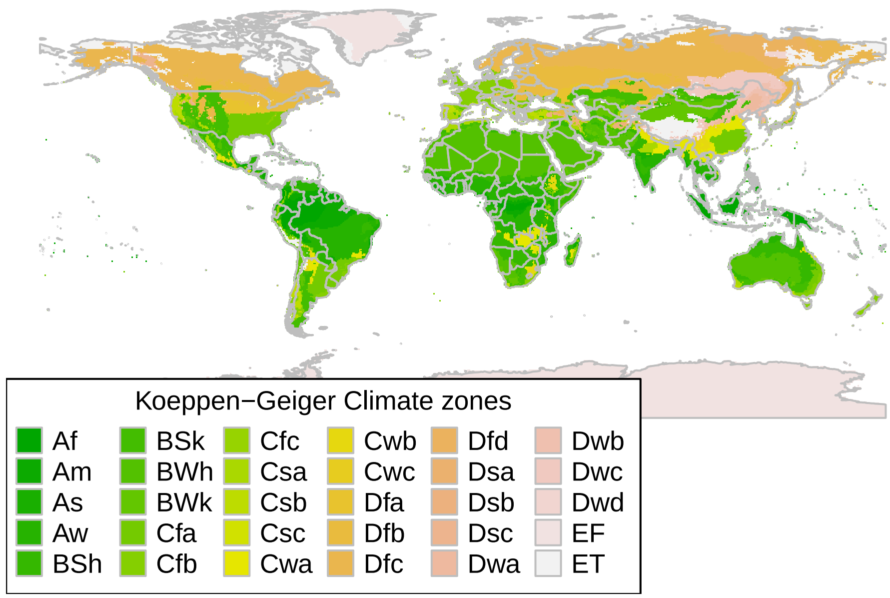
```
> mapCountryData(sPDF, nameColumnToPlot='BIODIVERSITY')
```



For identifying countries, the interactive function `identifyCountries()` allows users to click close to a country centroid in order to add the country name to the map.

Gridded data from the web can also be read in and plot. Please go to `http://koeppen-geiger.vu-wien.ac.at/present.htm` and download the ascii file with the Koeppen Geiger gridded climatic regions. Have a look at the file structure. Then, follow these instructions:

```
> file1 <- 'Koeppen-Geiger-ASCII.txt'
> kdata <- read.table(file1,header=TRUE, as.is=TRUE)
> # convert table to SpatialPointsDataFrame
> coordinates(kdata) <- c("Lon","Lat")
> # convert spodf to SpatialPixelsDataFrame
> gridded(kdata) <- TRUE
> # promote spidf to SpatialGridDataFrame
> kgrid <- as(kdata, "SpatialGridDataFrame")
> # plotting map
> kmap <- mapGriddedData(kgrid,catMethod='categorical',addLegend=FALSE,
+ colourPalette=terrain.colors(30))
> # adding formatted legend
> do.call(addMapLegendBoxes, c(kmap, cex=0.8, ncol=6, x='bottomleft',
+ title='Koeppen-Geiger Climate zones'))
```



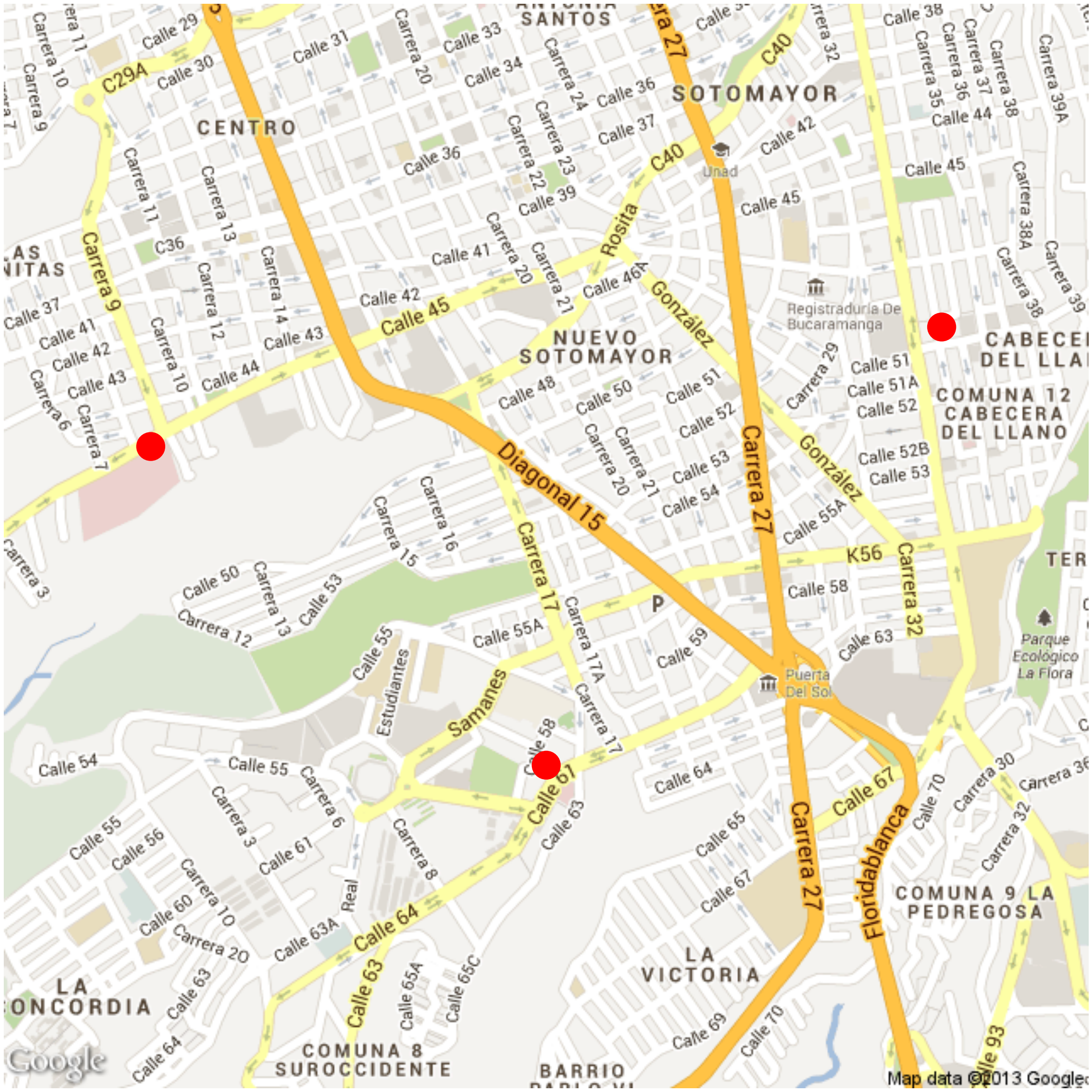
For detailed information of *rworldmap* functionalities have a look at `http://journal.r-project.org/archive/2011-1/Rjournal_2011-1_S`.

4.2 Mapping data using google maps

A three step process is necessary: (i) load the *RgoogleMaps* package; (ii) get maps of desired locations from google maps, and save them; and, (iii) provide points of interest:

```
> #options(width=40)
```

```
> library(RgoogleMaps)
> # first, provide map center
> map1 ← GetMap(center=c(−73.1, 7.1227), zoom=12,
+ destfile="map1.png", maptype="satellite")
> # now, define bounding box
> map2 ← GetMap.bbox(lonR=c(−73.0, −73.4), latR=c(7.0,7.3),
+ destfile="map2.png", maptype="terrain")
> # try another map type
> map3 ← GetMap.bbox(lonR=c(−73.0, −73.4), latR=c(7.0,7.3),
+ destfile="map3.png", maptype="satellite")
> # now plot data into these maps
> PlotOnStaticMap(lat=c(7.104, 7.115, 7.112), lon=c(−73.12,
+ −73.11, −73.13), zoom=12, cex=2, pch=19,col="red",
+ FUN=points, add=F)
```



5 Accessing geoservices

Using free web services such as the GeoNames, available via the package **geonames**, it is possible to obtain, for a given location, its elevation, name of the closest city and/or actual weather. Please install first the **geonames**, and then use this code:

```
> library(geonames)
> # search by bounding box
```

```
> mycities ←GNcities(north=6,south=3,east=−73,west=−76,lang="de")
> mycities
```

```
                                           fcodeName    toponymName  countrycode fcl
1                    capital of a political entity         Bogot
CO    P
2  seat of a first-order administrative division         Ibagu
CO    P
3  seat of a first-order administrative division       Pereira
CO    P
4  seat of a first-order administrative division       Armenia
CO    P
5  seat of a first-order administrative division     Manizales
CO    P
6  seat of a first-order administrative division Villavicencio
CO    P
7  seat of a first-order administrative division         Tunja
CO    P
8                                 populated place       Cartago
CO    P
9                                 populated place      Girardot
CO    P
10                                populated place    Facatativ
CO    P
             fclName           name wikipedia       lng fcode geonameId
lat
1  city, village,...         Bogot             -74.08175  PPLC   3688689 4.609706
2  city, village,...         Ibagu             -75.23222  PPLA   3680656 4.438889
3  city, village,...       Pereira            -75.69611  PPLA   3672486 4.813333
4  city, village,...       Armenia            -75.68111  PPLA   3689560 4.533889
5  city, village,...     Manizales            -75.51738  PPLA   3675443 5.068890
6  city, village,... Villavicencio            -73.62664  PPLA   3665900 4.142002
7  city, village,...         Tunja            -73.36778  PPLA   3666608 5.535278
8  city, village,...       Cartago            -75.91167   PPL   3687230 4.746389
9  city, village,...      Girardot            -74.80468   PPL   3682028 4.298659
10 city, village,...     Facatativ             -74.35453   PPL   3682516 4.813668
   population
1     7102602
2      421685
3      440118
4      315328
5      357814
6      321717
7      117479
8      134827
9      130289
10      94611
```

```
> # search using a precise location
> bog_alt ← GNsrtm3(lat=4.6 , lng=−74.1)
> bog_alt
```

```
  srtm3   lng lat
1  2566 -74.1 4.6
```

```
> # weather query
```

```
> bog_weather <- GNweather(north=4.8, east=-73.8, south=4.4, west=-74.2)
> bog_weather
```

```
            clouds weatherCondition
1 scattered clouds              n/a
                                                                       observation
1 SKBO 092200Z 28004KT 7000 VCSH SCT017 BKN080 14/12 A3022 RERA RMK/VCSH/NW/SE
  windDirection ICAO cloudsCode       lng temperature dewPoint windSpeed
1           280 SKBO        SCT -74.11667          14       12
04
  humidity       stationName            datetime lat
1       87 Bogota / Eldorado 2013-08-09 22:00:00 4.7
```

```
> # search by name
> bog <- GNsearch(q="bogota",maxRows=10)
> bog
```

```
   countryId adminCode1   countryName                    fclName countryCode
1    3686110         34      Colombia          city, village,...
CO
2    3686110         34      Colombia country, state, region,...
CO
3    2017370         11        Russia          city, village,...
RU
4    6252001         NJ United States          city, village,...
US
5    3686110         34      Colombia country, state, region,...
CO
6    6252001         TN United States          city, village,...
US
7    6252001         IL United States          city, village,...
US
8    3057568       <NA>      Slovakia    mountain,hill,rock,...
SK
9    3686110         25      Colombia          city, village,...
CO
10   2139685         00 New Caledonia    mountain,hill,rock,...
NC
         lng                             fcodeName                toponymName
1  -74.08175        capital of a political entity
Bogot
2  -74.18333  first-order administrative division Distrito Capital de Bogot
3  110.40000                       populated place
Bogota
4  -74.02986                       populated place
Bogota
5  -74.08333 second-order administrative division
Chipaque
6  -89.43841                       populated place
Bogota
7  -88.24004                       populated place
Bogota
8   21.51645                                  peak
Bogota
9  -81.35000                       populated place
Bogot
10 166.00000                             peninsula          Presqule Bogota
```

```
   fcl              name fcode geonameId         lat
1    P             Bogot   PPLC    3688689   4.609706
2    A       Bogota D.C.  ADM1    3688685   4.250000
3    P            Bogota   PPL    2026556  51.650000
4    P            Bogota   PPL    5095808  40.876211
5    A          Chipaque  ADM2    3686533   4.500000
6    P            Bogota   PPL    4050494  36.163958
7    P            Bogota   PPL    4828343  38.918378
8    T            Bogota    PK    7732440  48.701030
9    P             Bogot    PPL    3688687  13.333333
10   T  P r e s q u le  Bogota    PEN    2141841  -21.466667
                                                   adminName1  population
1                                                 Bogota D.C.     7102602
2                                                 Bogota D.C.     6840116
3                                                   Buryatiya
0
4                                                  New Jersey        8187
5                                                 Bogota D.C.
0
6                                                   Tennessee
0
7                                                    Illinois
0
8
0
9  Archipi lago de San Andr s , Providencia y Santa Catalina
0
10
0
```

You may think GeoNames output is quite messy for a real world application. However, making your effort you could extract useful data from such output. Anyway, it seems much more interesting to access WPS WorldGrids from R. You have to install the *GISF* package using instructions available at `http://worldgrids.org`.

The first step is to connecting to the WPS server and to get a list of available services. Then, to defining a raster layer as an object of class WPS. In the following code, such a layer is a bioclimatic layer –Annual Precipitation–

```
> library(GSIF)
> #
> URI = "http://wps.worldgrids.org/pywps.cgi"
> server <- list(URI=URI, request="execute", version="version=1.0.0",
+ service.name="service=wps", identifier="identifier=sampler_local1pt_nogml")
> biocl12.wps <- new("WPS", server=server, inRastername="biocl12")
> str(biocl12.wps)
```

```
Formal class 'WPS' [package "GSIF"] with 2 slots
  ..@ server      :List of 5
  .. ..$ URI          : chr "http://wps.worldgrids.org/pywps.cgi"
  .. ..$ request      : chr "execute"
  .. ..$ version      : chr "version=1.0.0"
  .. ..$ service.name: chr "service=wps"
  .. ..$ identifier  : chr "identifier=sampler_local1pt_nogml"
  ..@ inRastername: chr "biocl12"
```

Once a WorldGrids raster layer has been defined as WPS-class object, it can be manipulated as any other spatial grid-type object, available for example via the sp package. To find out what is available via this WPS, you can fetch the processes and required arguments by using:

```
> prl <- getProcess(biocl12.wps)
> prl[7]
```

```
overlay TIFF and report statistics
                         "overlay"
```

To fetch values of a WPS raster layer at some point locations we can use the standard *over* method, previously "overlay", available via the sp package that is in the GSIF package extended to WPS-type objects. We first need to define the points of interest (must be projected in geographical coordinates with the WGS84 ellipsoid/datum). Then, these points can be overlaid over the WPS object created previously. Surprisingly, the *over* method does not return a vector with raster layer values at every location, it outputs a single value (the first one). This means that,in order to get the intended output, users need to looping through the spatial points and overlaying one point at a time.

```
> library(sp)
> p1 <- data.frame(lon=c(-73,-74.5,-72.3,-76.6),lat=c(7,4.5,11.3,5.7),
+ city=c("BUCARAMANGA","BOGOTA","MAICAO","QUIBDO"))
> coordinates(p1) <- ~lon+lat
> proj4string(p1) <- CRS("+proj=longlat +datum=WGS84")
> # looping through the points
> names=c("BUCARAMANGA","BOGOTA","MAICAO","QUIBDO")
> for(i in names)
+ {
+ p2 <- subset(p1, city==i)
+ precip <- over(biocl12.wps, p2)
+ print(precip)
+ }
```

```
[1] "1442"
[1] "1371"
[1] "765"
[1] "7480"
```

An advantage of using WPS services is that there is no need to download complete grids. Instead, users can subset grids using a bounding box. The following code allows users to extract (and save a TIFF) with a specified extend from one file from the worldgrids repository:

```
> library(raster)
> library(rasterVis)
> #bounding box should be in format LonMin, LatMin, LonMax, LatMax:
> biocl12 <- subset(biocl12.wps, bbox=matrix(c(-76,4,-72,8),nrow=2))
```

```
biocl12_-76_4_-72_8.tif has GDAL driver GTiff
and has 80 rows and 80 columns
```

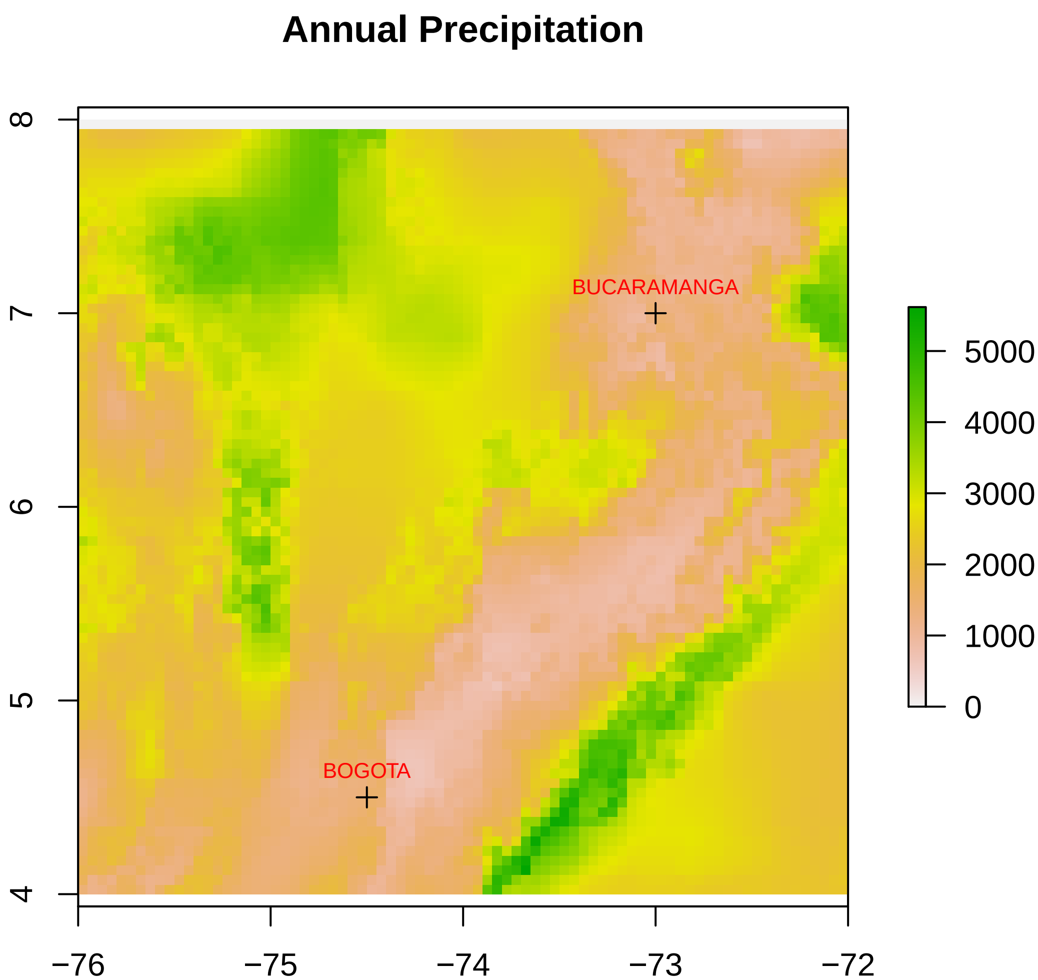
```
> str(biocl12)
```

```
Formal class 'SpatialGridDataFrame' [package "sp"] with 4 slots
  ..@ data       :'data.frame': 6400 obs. of  1 variable:
  .. ..$ biocl12: int [1:6400] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] -75.97 4.03
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : num [1:2] 0.05 0.05
  .. .. ..@ cells.dim        : int [1:2] 80 80
  ..@ bbox       : num [1:2, 1:2] -76 4 -72 8
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

```
> bio12 <- raster(biocl12)
> bio12
```

```
class       : RasterLayer
dimensions  : 80, 80, 6400  (nrow, ncol, ncell)
resolution  : 0.05, 0.05  (x, y)
extent      : -76, -72, 4, 8  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : biocl12
values      : 0, 5618  (min, max)
```

```
> plot(bio12, main="Annual Precipitation")
> x <- coordinates(p1)[,1]
> y <- coordinates(p1)[,2]
> plot(p1, add=TRUE)
> text(x, y, labels = names, cex= 0.7, pos=3, col = "red")
```



Let's try downloading (and plotting) MERIS-based land cover data:

```
> #
> landcover.wps <- new("WPS", server=server, inRastername="glcesa3a")
> str(landcover.wps)
```

```
Formal class 'WPS' [package "GSIF"] with 2 slots
  ..@ server      :List of 5
  .. ..$ URI         : chr "http://wps.worldgrids.org/pywps.cgi"
  .. ..$ request     : chr "execute"
  .. ..$ version     : chr "version=1.0.0"
  .. ..$ service.name: chr "service=wps"
```

```
  .. ..$ identifier  : chr "identifier=sampler_local1pt_nogml"
  ..@ inRastername: chr "glcesa3a"
```

```
> prl <- getProcess(landcover.wps)
> prl[7]
```

```
overlay TIFF and report statistics
                         "overlay"
```

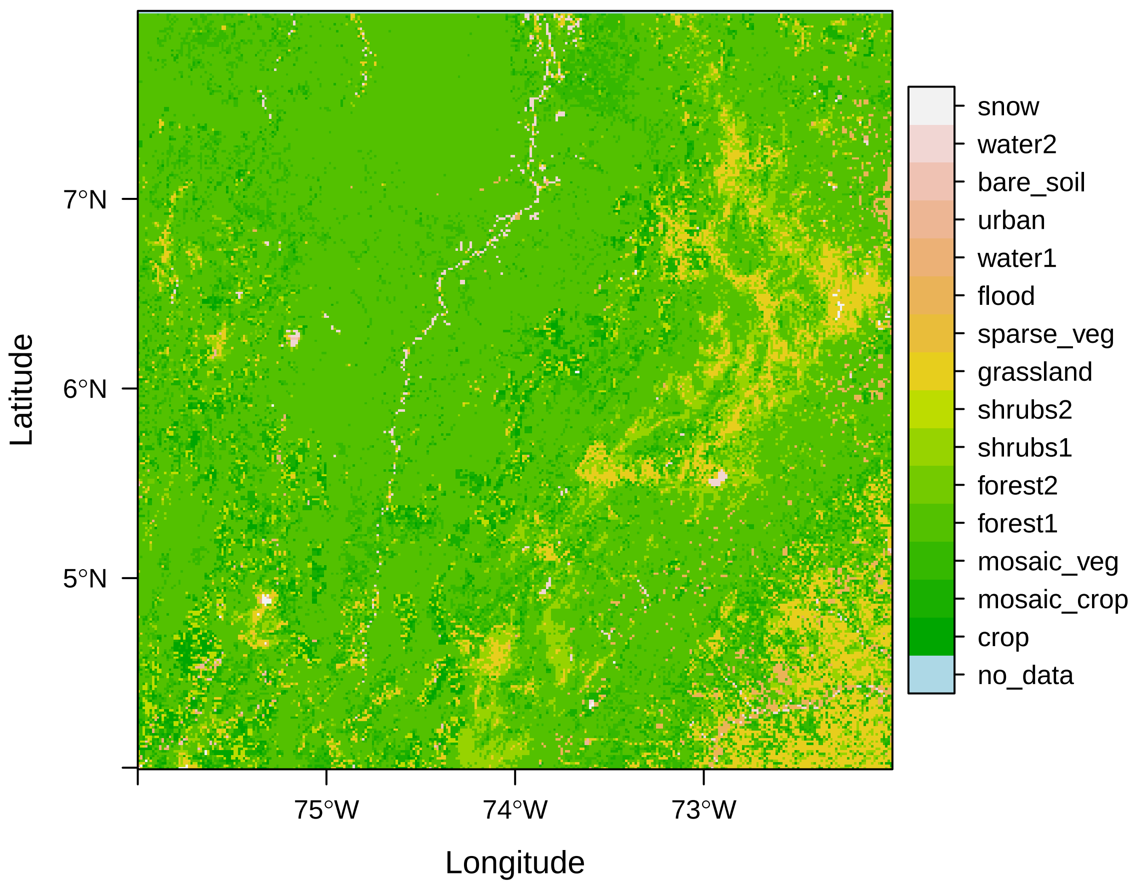
```
> library(raster)
> library(rasterVis)
> #bounding box should be in format LonMin, LatMin, LonMax, LatMax:
> lc <- subset(landcover.wps, bbox=matrix(c(-76,4,-72,8),nrow=2))
```

```
glcesa3a_-76_4_-72_8.tif has GDAL driver GTiff
and has 480 rows and 480 columns
```

```
> str(lc)
```

```
Formal class 'SpatialGridDataFrame' [package "sp"] with 4 slots
  ..@ data       :'data.frame': 230400 obs. of  1 variable:
  .. ..$ glcesa3a: int [1:230400] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] -76 4
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : num [1:2] 0.00833 0.00833
  .. .. ..@ cells.dim        : int [1:2] 480 480
  ..@ bbox       : num [1:2, 1:2] -76 3.99 -72 7.99
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

```
> #
> # plotting using rasterVis
> rlc <- raster(lc)
> rlc <- ratify(rlc)
> rat <- levels(rlc)[[1]]
> # be careful, this text needs validation
> rat["txt"] <- c("no_data", "crop", "mosaic_crop", "mosaic_veg",
+ "forest1", "forest2", "shrubs1","shrubs2","grassland", "sparse_veg",
+ "flood", "water1", "urban", "bare_soil", "water2","snow")
> levels(rlc) <- rat
> myPal <- c('lightblue', terrain.colors(16))
> levelplot(rlc, col.regions=myPal)
> #
```



Detailed information on *GSIF* functionalities can be found at: `http://gsif.r-forge.r-project.org/00Index.html`

6 Basic analysis of spatial data sets

Let's say that you think GSIF data (or another website providing global spatial data sets) are too coarse to be good enough for your own purposes. In such a case, it may be that you have several shapefiles covering your country. You can download, for example, a shapefile of Colombia's municipalities from this link:

http://db.tt/AXhS1Lz2.

Assuming that you have downloaded and extracted that shapefile on subdirectory *MunWGS84*, follow these instructions to read data, make attribute and location based selection, and plot the result:

```
> library(rgdal)
> dsn <- "./MunWGS84"
> mun <- readOGR(dsn, layer="MpiosWGS84")
```

```
OGR data source with driver: ESRI Shapefile
Source: "./MunWGS84", layer: "MpiosWGS84"
with 1126 features and 5 fields
Feature type: wkbPolygon with 2 dimensions
```

```
> #
> class(mun)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
> # bounding box
> box1 <- bbox(mun)
> # to view attribute table --first 5 records--
> mun@data[1:5,]
```

```
       NMG NOMBREDEPT  DANE     sq_km pop_dens
0   URIBIA La Guajira 44847 7857.8819 14.97528
1  MANAURE La Guajira 44560 1618.6131 41.75427
2   MAICAO La Guajira 44430 1731.1086 71.49003
3 RIOHACHA La Guajira 44001 3009.3334 55.78146
4  ALBANIA La Guajira 44035  590.1936 35.26809
```

```
> # to select municipalities inside a department
> guajira <- subset(mun,NOMBREDEPT=="La Guajira")
> # to view selected data
> guajira@data
```

```
                       NMG NOMBREDEPT  DANE     sq_km pop_dens
0                   URIBIA La Guajira 44847 7857.8819 14.97528
1                  MANAURE La Guajira 44560 1618.6131 41.75427
2                   MAICAO La Guajira 44430 1731.1086 71.49003
3                 RIOHACHA La Guajira 44001 3009.3334 55.78146
4                  ALBANIA La Guajira 44035  590.1936 35.26809
6                  DIBULLA La Guajira 44090 1799.5368 12.11312
7              HATO\\NUEVO La Guajira 44378  215.8112 75.91357
8                BARRANCAS La Guajira 44078  941.5915 27.96223
13                 FONSECA La Guajira 44279  658.4152 40.75088
14            DISTRACCI N La Guajira 44098  218.7932 54.67263
15 SAN JUAN\\DEL CESAR La Guajira 44650 1444.4310 23.29914
40              EL\\MOLINO La Guajira 44110  230.8492 31.68735
42              VILLANUEVA La Guajira 44874  284.6481 82.69157
47                 URUMITA La Guajira 44855  301.1741 44.32321
51 LA JAGUA\\DEL PILAR La Guajira 44420  222.4435 12.23232
```

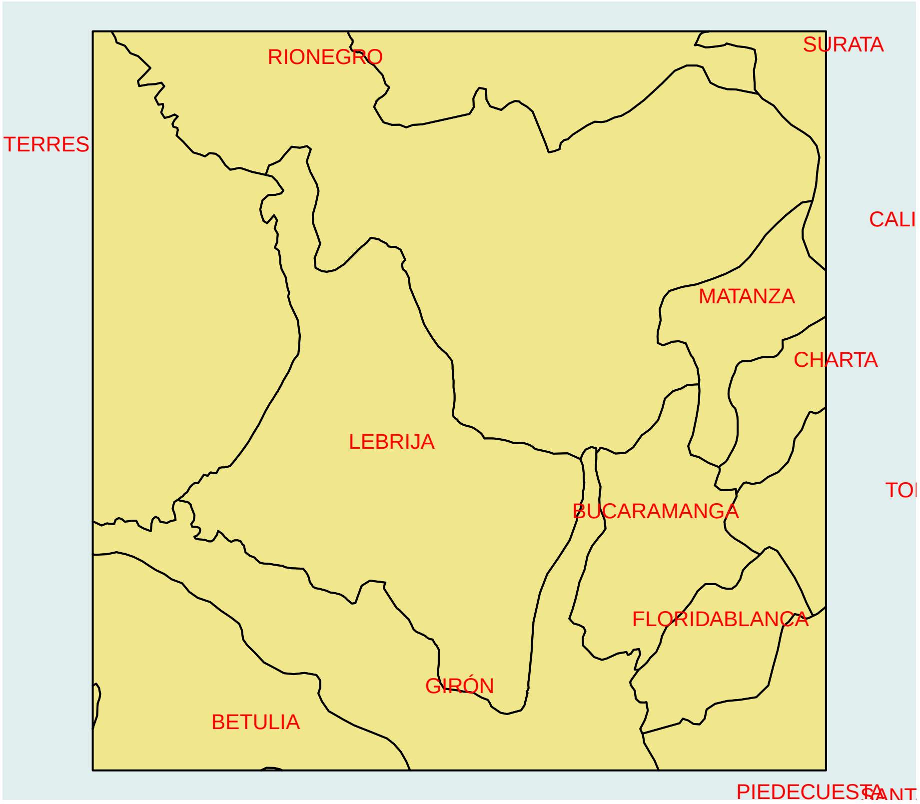
```
> # to make a selection based on attributes
> h_pop <- subset(guajira, pop_dens>70)
> h_pop@data
```

```
           NMG NOMBREDEPT  DANE     sq_km pop_dens
2       MAICAO La Guajira 44430 1731.1086 71.49003
7   HATO\\NUEVO La Guajira 44378  215.8112 75.91357
42  VILLANUEVA La Guajira 44874  284.6481 82.69157
```

```
> # to make a selection based on location
> # first, let's create a clipping polygon
> library(raster)
> cpoly <- as(extent(-73.5,-73,7,7.5),"SpatialPolygons")
> proj4string(cpoly) <- CRS(proj4string(mun))
> library(rgeos)
> # rgeos is a wrapper for the GEOS library
> selected <- gIntersection(mun,cpoly,byid=TRUE)
> bbox(selected)
```

```
    min   max
x -73.5 -73.0
y   7.0   7.5
```

```
> plot(selected,col="khaki",bg="azure2")
> x <- coordinates(mun)[,1]
> y <- coordinates(mun)[,2]
> #plot(mun, add=TRUE)
> names <-mun[["NMG"]]
> text(x, y, labels = names, cex= 0.7, pos=3, col = "red")
```



In case users do not like to use locally stored data, administrative areas of any country can be downloaded from internet. Let's download spatial data corresponding to departments of Colombia. Then, let's find which departments neighbor La Guajira and plot them:

```
> # get administrative units at level 1 -
> con <- url("http://www.r-gis.org/rgis/data/adm/COL_adm1.RData")
> print(load(con))
```

```
[1] "gadm"
```

```
> class(gadm)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

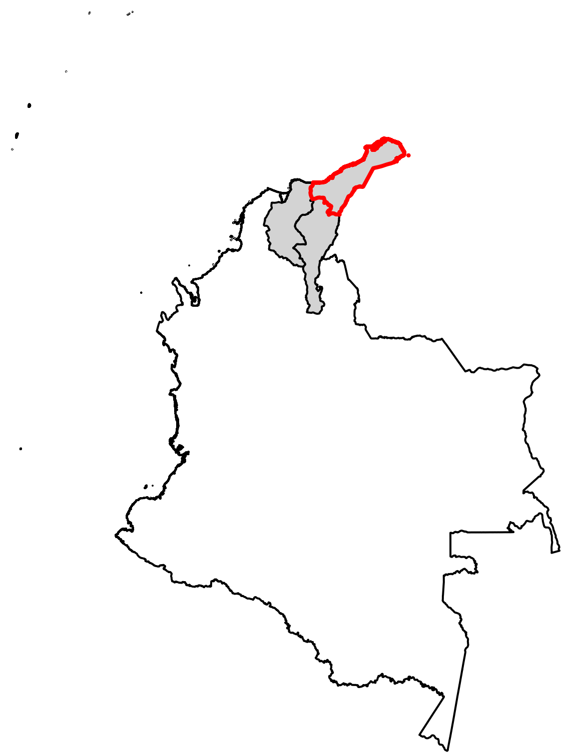
```
> # note gadm is of class SpatialPolygonsDataFrame
> names(gadm)
```

```
 [1] "ID_0"        "ISO"         "NAME_0"      "ID_1"        "NAME_1"
 [6] "VARNAME_1"   "NL_NAME_1"   "HASC_1"      "CC_1"        "TYPE_1"
[11] "ENGTYPE_1"   "VALIDFR_1"   "VALIDTO_1"   "REMARKS_1"   "Shape_Leng"
[16] "Shape_Area"
```

```
> #
> row.names(gadm) = as.character(gadm[["NAME_1"]])
> #
> col <- gUnionCascaded(gadm)
> #
> guaj_i <- which(gadm[["NAME_1"]] =="La Guajira")
> # guaj_i <- which(gadm\$NAME_1" =="La Guajira"
> guaj_neighbors <- gIntersects(gadm[guaj_i,],gadm,byid=TRUE)
> which(guaj_neighbors)
```

```
[1] 11 18 19
```

```
> neighbors <- gIntersects(gadm, byid=TRUE)
> #
> plot(col)
> plot(gadm[which(guaj_neighbors),],add=T,col="lightgrey")
> plot(gBoundary(gadm[guaj_i,]),add=T,col='red',lwd=2)
```



Now, let's extract a raster of precipitation for the whole country:

```
> library(rgdal)
> # get administrative units at level 2
> con <- url("http://www.r-gis.org/rgis/data/adm/COL_adm2.RData")
> print(load(con))
```

```
[1] "gadm"
```

```
> polys <- SpatialPolygons(gadm@polygons)
> proj4string(polys) <- proj4string(gadm)
> #
```

```
> library(raster)
> library(rasterVis)
> # bounding box should be in format LonMin, LatMin, LonMax, LatMax:
> colbox=matrix(c(-79.03,-4.22,-66.85,12.46),nrow=2)
> biocl12 <- subset(biocl12.wps, bbox=colbox)
```

```
biocl12_-79.03_-4.22_-66.85_12.tif has GDAL driver GTiff
and has 334 rows and 244 columns
```

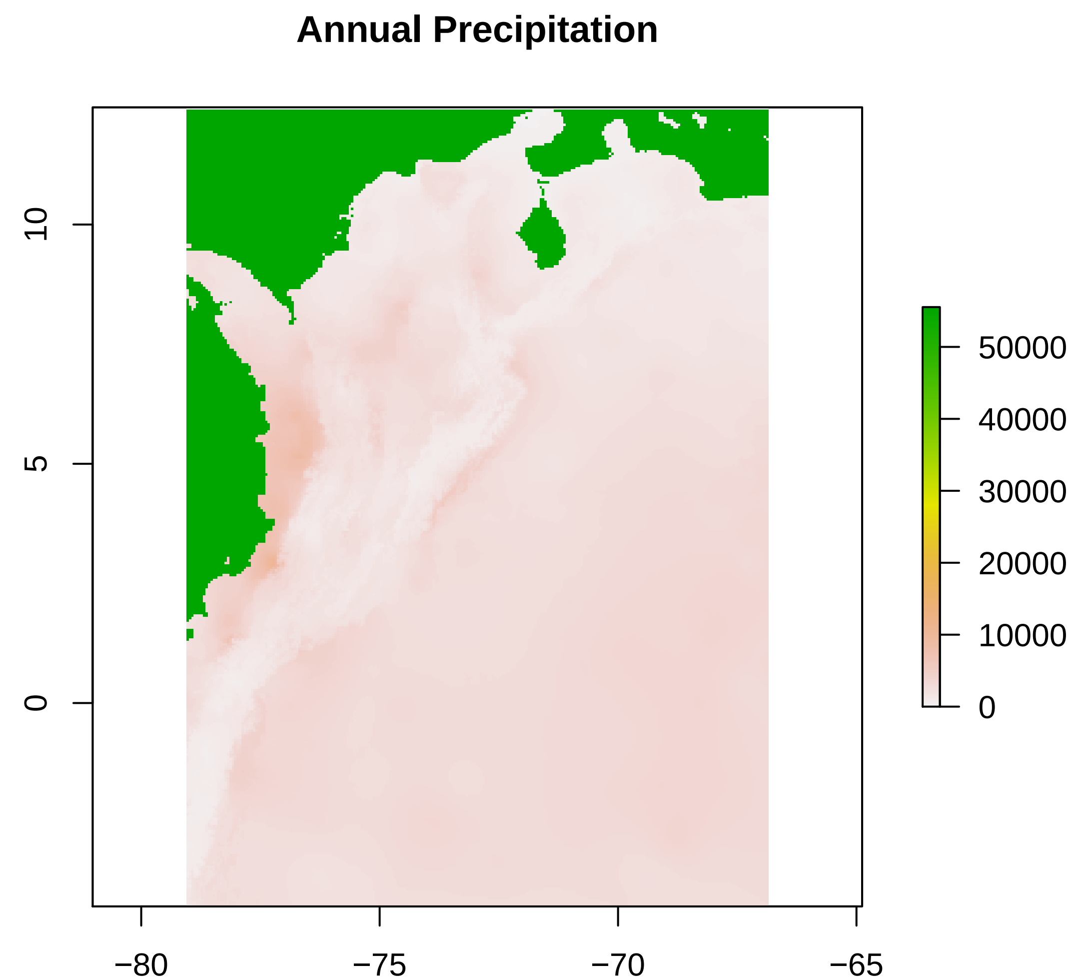
```
> str(biocl12)
```

```
Formal class 'SpatialGridDataFrame' [package "sp"] with 4 slots
  ..@ data       :'data.frame': 81496 obs. of  1 variable:
  .. ..$ biocl12: int [1:81496] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ grid       :Formal class 'GridTopology' [package "sp"] with 3 slots
  .. .. ..@ cellcentre.offset: Named num [1:2] -79.02 -4.22
  .. .. .. ..- attr(*, "names")= chr [1:2] "x" "y"
  .. .. ..@ cellsize         : num [1:2] 0.05 0.05
  .. .. ..@ cells.dim        : int [1:2] 244 334
  ..@ bbox       : num [1:2, 1:2] -79.05 -4.25 -66.85 12.45
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slots
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

```
> bio12 <- raster(biocl12)
> bio12
```

```
class       : RasterLayer
dimensions  : 334, 244, 81496  (nrow, ncol, ncell)
resolution  : 0.05, 0.05  (x, y)
extent      : -79.05, -66.85, -4.25, 12.45  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : biocl12
values      : 0, 55537  (min, max)
```

```
> plot(bio12, main="Annual Precipitation")
> plot(polys, ADD=TRUE)
```



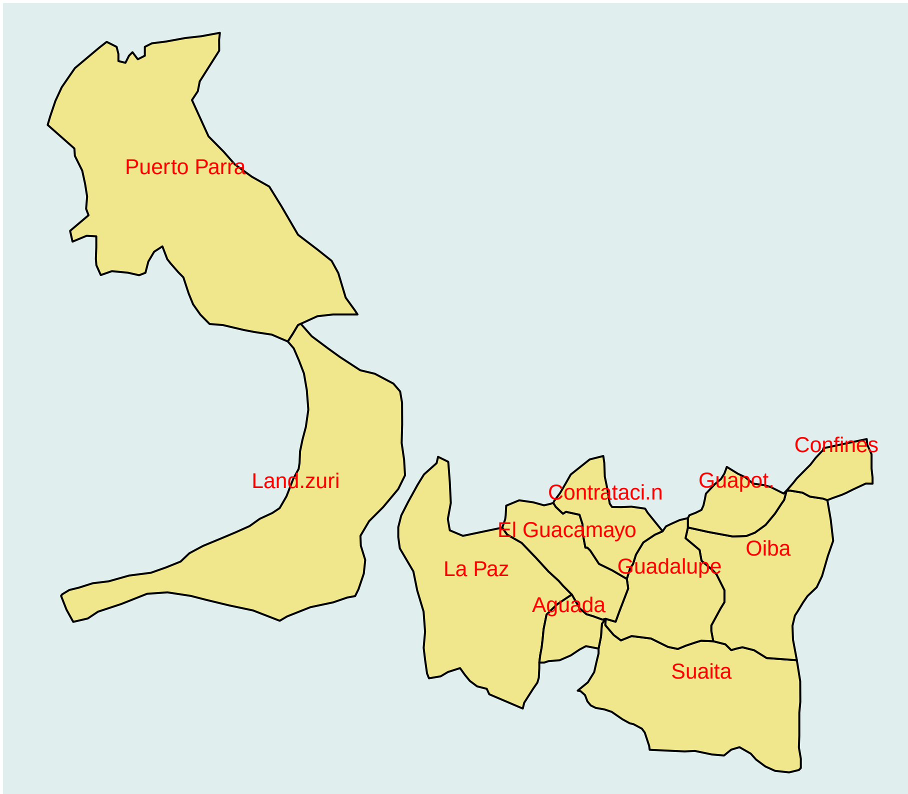
Now, let's do some kind of spatial aggregation:

```
> # extraction of precipitation values for each municipality cell
> # it may take some time for the whole country!
> #mprec <- extract(bio12,polys,small=TRUE)
> # mean value of precipitation at each municipality
> #mean_precip <- sapply(mprec, function(x) apply(x,2,mean,na.rm=T)
> # it is better to try it using a spatial subset
> stder <- subset(gadm, NAME_1=="Santander")
> polys2 <- SpatialPolygons(stder@polygons)
> proj4string(polys2) <- proj4string(stder)
> #
```

```
> mprec1 <- extract(bio12,polys2,weights=TRUE,fun=mean)
> mprec1
```

```
 [1] 1359.292 2717.818 2604.169 2125.338 2835.426 2509.534 1246.449 1210.046
 [9] 2467.613 2699.215 1111.674 1424.840 2843.772 1944.982 1463.486 2503.375
[17] 1367.232 2797.830 1827.749 2565.221 1297.631 1522.481 1112.302 1543.673
[25] 1479.249 1086.187 1438.646 2612.751 1264.398 2505.767 2400.187 2703.758
[33] 1561.584 2801.219 2813.413 1885.413 1863.777 2449.027 2869.499 2725.565
[41] 1855.832 1564.180 1633.820 2628.862 1181.442 1816.785 2080.593 1721.984
[49] 1341.109 2992.755 2942.670 2506.878 2710.956 2105.805 2075.749 1222.455
[57] 2559.605 2879.797 2946.234 1780.032 1198.466 1402.423 1366.892 1359.750
[65] 2213.272 1533.079 2671.941 3028.781 1554.869 1870.643 2521.502 2507.322
[73] 1211.662 1945.372 1832.917 2877.854 2569.749 2155.783 2641.361 1480.647
[81] 2794.740 1885.452 1860.380 1413.031 1435.649 2180.756 1556.892
```

```
> stder[["prec"]] <- mprec1
> rainy <- subset(stder, prec>2800)
> plot(rainy,col="khaki",bg="azure2")
> x <- coordinates(rainy)[,1]
> y <- coordinates(rainy)[,2]
> #plot(mun, add=TRUE)
> names <-rainy[[7]]
> text(x, y, labels = names, cex= 0.7, pos=3, col = "red")
```



I am happy to have completed these exercises using `Rgis` functionalities. I hope you have replicated all of them without tears. I wish you are now eager to conduct your own GIS work with `R`. See you on the road!