**Background**

Geospatial data is becoming increasingly used to solve numerous ‘real-life’ problems (check out some examples [here](http://gisgeography.com/gis-applications-uses/)). In turn, R is becoming more equipped than ever to handle this type of data. Thus, providing an exceptional open-source solution to solve many problems in the Geographic Information Sciences and Remote Sensing domains.

In general, two types of geospatial data models are used to represent, visualize, and model spatial phenomena. These are:

* **Vector Data**: This represents the world in three simple geometries: **points**, **lines**, and **polygons**. As such, it allows you to represent spatial phenomena or variables that are typically discrete and with well-defined boundaries (e.g., touristic points-of-interest, gas stations, rivers, roads, drainage basins, country GDP).
* **Raster Data**: This provides support for representing spatial phenomena by diving the surface into a grid (or matrix) composed of cells of regular size. Each raster dataset has a certain number of columns and rows and each cell contains a value with information for the variable of interest. Stored data can be either: (i) thematic – representing a *discrete* variable (e.g., land cover classification map) or *continuous* (e.g., elevation).

Choosing the appropriate data model to use depends on the domain of application and the specific problem at hand. Typically, people from social sciences tend to use the vector data model more. R packages such as **sp** or **sf** (a relatively new package, starting in 2016), provide support for this type of data. In contrast, in the environmental sciences, the raster data model is more often used because of satellite data, or the need to represent spatially continuous phenomena, such as pollution levels, temperature or precipitation values, the abundance or habitat suitability for a species, etc. The **raster** package, introduced in March 2010 by Robert Hijmans & Jacob van Etten, currently provides many useful functions for using this type of data. Despite these differences, GIS specialists and researchers often use both data models to tackle their problems.

Throughout these posts, we will cover the basics, intermediate, and some advanced stuff in **raster data** handling, manipulation, and modeling in R. Examples will be given along with the tutorials. Some exercises, with different difficulty levels, will be provided so you can practice.

The raster package currently provides an extensive set of functions to create, read, export, manipulate, and process raster datasets. It also provides low-level functionalities for creating more advanced processing chains, as well as the ability to manage large datasets. For more information see: vignette("functions", package = "raster").

This first post on raster data is divided into two sub-sections:  
(i) Accessing raster attributes, and,

(ii) Viewing raster values and calculating simple statistics.

First, we need to install the raster package (as well as sp and rgdal):

if(!("rgdal" %in% installed.packages()[,1]))

install.packages(c("rgdal"), dependencies = TRUE)

if(!("sp" %in% installed.packages()[,1]))

install.packages(c("sp"), dependencies = TRUE)

if(!("raster" %in% installed.packages()[,1]))

install.packages(c("raster"), dependencies = TRUE)

library(rgdal)

library(sp)

library(raster)

Next, download and unzip the sample data. We will use SRTM – version 4.1 elevation data (in meters a.s.l.) for the Peneda-Geres National Park – Portugal ([+info](https://en.wikipedia.org/wiki/Peneda-Ger%C3%AAs_National_Park)) in the examples.

## Create a folder named data-raw inside the working directory to place downloaded data

if(!dir.exists("./data-raw")) dir.create("./data-raw")

## If you run into download problems try changing: method = "wget"

download.file("https://raw.githubusercontent.com/joaofgoncalves/R\_exercises\_raster\_tutorial/master/data/srtm\_pnpg.zip", "./data-raw/srtm\_pnpg.zip", method = "auto")

## Uncompress the zip file

unzip("./data-raw/srtm\_pnpg.zip", exdir = "./data-raw")

**Raster Attributes**

In the first part (of two) of this tutorial, we will focus on reading raster data and accessing its core attributes.

After finishing the download, load the data into R using the raster function (see ?raster for more details). Then use print to inspect the “essential” attributes of the dataset.

# In this example the function uses a string with the data location

rst <- raster("./data-raw/srtm\_pnpg.tif")

# Print raster attributes

print(rst)

## class : RasterLayer

## dimensions : 579, 555, 321345 (nrow, ncol, ncell)

## resolution : 80, 80 (x, y)

## extent : 549619.7, 594019.7, 4613377, 4659697 (xmin, xmax, ymin, ymax)

## coord. ref. : +proj=utm +zone=29 +datum=WGS84 +units=m +no\_defs +ellps=WGS84 +towgs84=0,0,0

## data source : D:\MyDocs\R-dev\R\_exercises\_raster\_tutorial\data-raw\srtm\_pnpg.tif

## names : srtm\_pnpg

## values : 9, 1520 (min, max)

From the above, we can see some important information about our raster dataset. Given that we used the raster function for data loading, we have now created a RasterLayer, ex., a raster object with a single layer. We can also see its dimension: 579 rows, 555 columns and the pixel size in x and y dimensions, a.k.a. the **spatial resolution**, equal to 80m (we are using a projected coordinate system with units in meters; more on this below).

We can use the function inMemory to check if the raster dataset is currently stored on RAM:

inMemory(rst)

## [1] FALSE

As we can see, the raster data is currently stored on the disk. So, at this point, our RasterLayer object is actually “made of” metadata and a link to the actual raster data on disk. This allows preserving RAM space.

The package also provides several functions to access each raster attribute individually.

## Raster layer name(s) / more useful for multi-layer rasters

## By default coincides with the file name without extension

names(rst)

## [1] "srtm\_pnpg"

## Number of rows, columns and layers

dim(rst)

## [1] 579 555 1

## Nr of rows

nrow(rst)

## [1] 579

# Nr of columns

ncol(rst)

## [1] 555

## Total number of grid cells

ncell(rst)

## [1] 321345

## Spatial resolution in x and y dimensions

res(rst)

## [1] 80 80

## Data type - see ?dataType for more details

dataType(rst)

## [1] "INT2S"

## Extent (returns a S4 object of class "Extent")

extent(rst)

## class : Extent

## xmin : 549619.7

## xmax : 594019.7

## ymin : 4613377

## ymax : 4659697

Info on extent coordinates can be retrieved individually:

xmin(rst)

## [1] 549619.7

xmax(rst)

## [1] 594019.7

ymin(rst)

## [1] 4613377

ymax(rst)

## [1] 4659697

Finally, we can also see info about the Coordinate Reference System (CRS) used to represent the data. Many different CRS is used to describe geographic data, depending on the location, extent, time, domain (among other features) of the collected data.

crs(rst)

## CRS arguments:

## +proj=utm +zone=29 +datum=WGS84 +units=m +no\_defs +ellps=WGS84

## +towgs84=0,0,0

For the raster package, a **proj4string** is used to set and define the CRS of the data. This string contains some important details of the CRS, such as the *Projection*, the *Datum*, the *Ellipsoid* and the *units* (e.g., meters, degree). Use the site [spatialreference.org](http://spatialreference.org/) to find the appropriate *proj4string* (or other information) for the CRS of your choice.

**Raster Values**

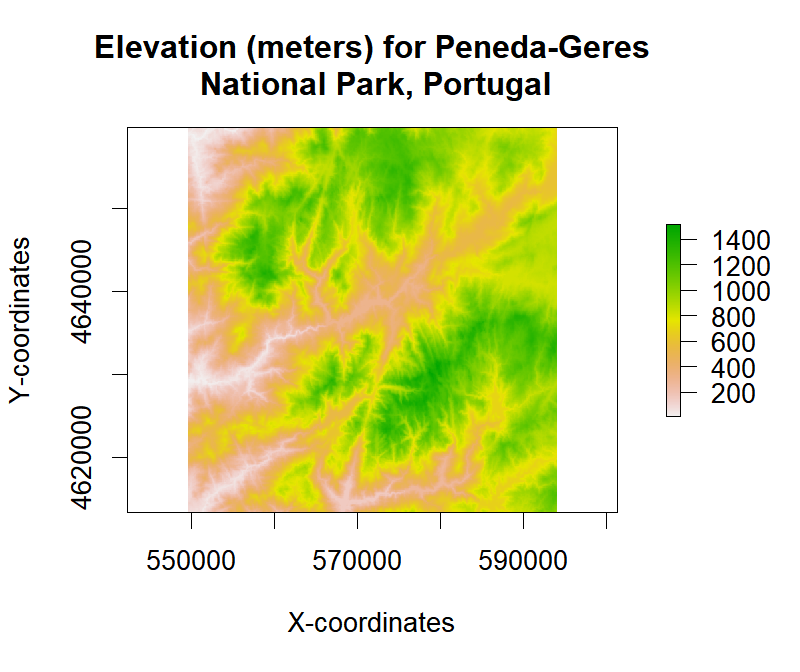
**Summary Statistics**

For the second, and last part, of this tutorial, we are going to explore raster functions for visualizing, summarizing and accessing/querying values at specific locations.

To visualize the data we can simply use the function plot.

plot(rst, main="Elevation (meters) for Peneda-Geres\n National Park, Portugal",

xlab="X-coordinates", ylab="Y-coordinates")



We can also use a histogram to visualize the distribution of elevation values in the sample data.

# Generate histogram from a sample of pixels (by default 100K are randomly used)

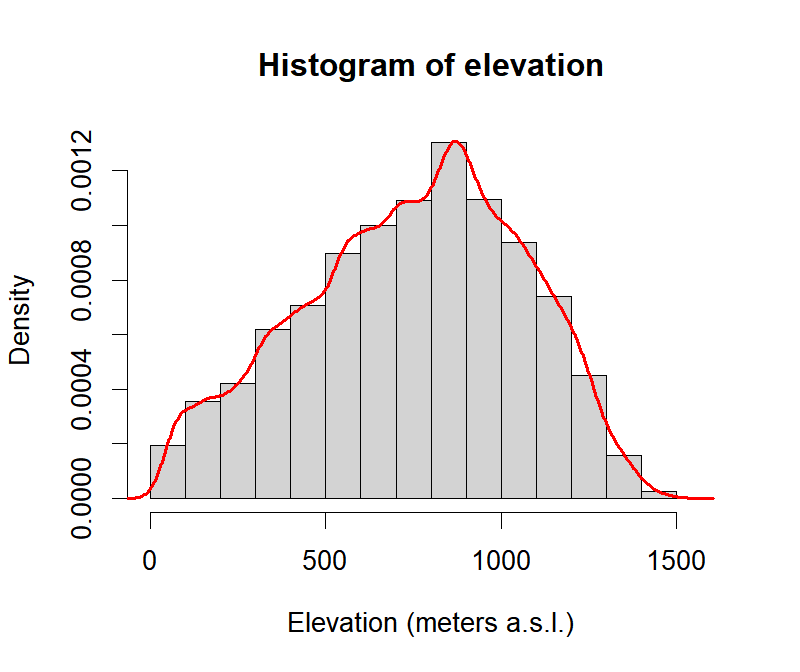
hist(rst, col="light grey", main = "Histogram of elevation", prob = TRUE,

xlab = "Elevation (meters a.s.l.)")

# Generate the density plot object and then overlap it

ds <- density(rst, plot = FALSE)

lines(ds, col = "red", lwd = 2)



Calculating summary statistics is fairly easy using the raster package. The generic method summary is available for this type of object (note: this function will use a sample of pixels to calculate statistics).

summary(rst)

## Warning in .local(object, ...): summary is an estimate based on a sample of 1e+05 cells (31.12% of all cells)

## srtm\_pnpg

## Min. 11

## 1st Qu. 529

## Median 776

## 3rd Qu. 984

## Max. 1520

## NA's 0

Minimum and maximum values can be calculated with the following functions (no sample employed):

## Min

minValue(rst)

## [1] 9

## Max

maxValue(rst)

## [1] 1520

The package also provides a more general interface to calculate cell statistics using the cellStats function (no sample employed).

## Mean

cellStats(rst, mean)

## [1] 747.2759

## Standard-deviation

cellStats(rst, sd)

## [1] 311.8615

## Median

cellStats(rst, median)

## [1] 774

## Median-absolute deviation (MAD)

cellStats(rst, mad)

## [1] 336.5502

## Quantiles

## 5%, 25%, 50%, 75% and 95%

cellStats(rst, function(x,...) quantile(x, probs=c(0.05, 0.25, 0.5, 0.75, 0.95),...))

## 5% 25% 50% 75% 95%

## 186 527 774 983 1224

cellStats does not use a random sample of the pixels to calculate. Hence, it will fail (gracefully) for very large Raster\* objects, except for certain predefined functions: sum, mean, min, max, sd, 'skew', and 'rms'.

**Extracting Values**

The raster package allows several possibilities to extract data at specific points, lines or polygons. The extract function used for this purpose allows a two-column matrix or data.frame (with x, y coordinates) or spatial objects from the sp package such as: SpatialPoints\*, SpatialPolygons\*, SpatialLines or Extent as input.

For the first example, we will start by extracting raster values using points as input:

## One specific point location (with coordinates in the same CRS as the input raster)

xy <- data.frame(x = 570738, y = 4627306)

xy <- SpatialPoints(xy, proj4string = crs(rst))

extract(rst, xy)

##

## 611

## Extract raster values for 20 randomly located points

xy <- data.frame(x = runif(20, xmin(rst), xmax(rst)), y = runif(20, ymin(rst), ymax(rst)))

xy <- SpatialPoints(xy, proj4string = crs(rst))

extract(rst, xy)

## [1] 137 761 1034 828 834 837 691 342 597 272 1263 935 270 1240

## [15] 339 1136 1245 991 1073 1153

Typically, we are also interested in extracting raster values for specific regions-of-interest (ROI). In this example, we will use a polygon (a broad-leaf forest area) to assess the distribution of elevation values within it.

## Download the vector data with the woodland patch ROI

## If you run into download problems try changing: method = "wget"

download.file("https://raw.githubusercontent.com/joaofgoncalves/R\_exercises\_raster\_tutorial/master/data/WOODS\_PNPG.zip", "./data-raw/WOODS\_PNPG.zip", method = "auto")

## Uncompress the data

unzip("./data-raw/WOODS\_PNPG.zip", exdir = "./data-raw")

## Convert the data into SpatialPolygons (discards the attached attribute but keeps geometry)

woods <- as(readOGR(dsn = "./data-raw", layer = "woods\_pnpg"), "SpatialPolygons")

## OGR data source with driver: ESRI Shapefile

## Source: "./data-raw", layer: "woods\_pnpg"

## with 1 features

## It has 6 fields

Let’s check out the polygon data with a simple plot:

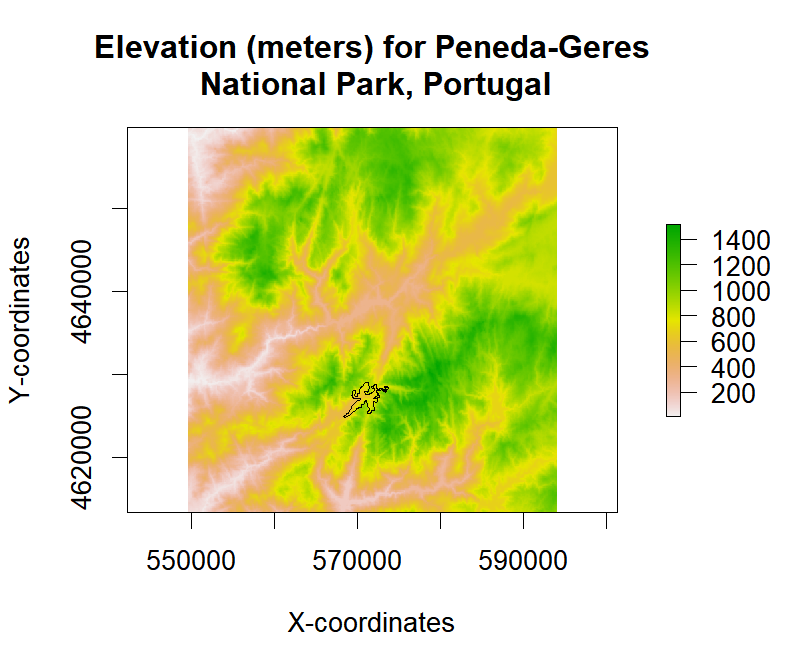
## Plot elevation raster

plot(rst, main="Elevation (meters) for Peneda-Geres\n National Park, Portugal",

xlab="X-coordinates", ylab="Y-coordinates")

## Add the ROI

plot(woods, add = TRUE)



Now, let’s extract the raster values from the polygon ROI and calculate some statistics:

elev <- extract(rst, woods)[[1]] ## Subset the first (and only) geometry element

# Tukey's five number summary: minimum, lower-hinge, median, upper-hinge, and, maximum

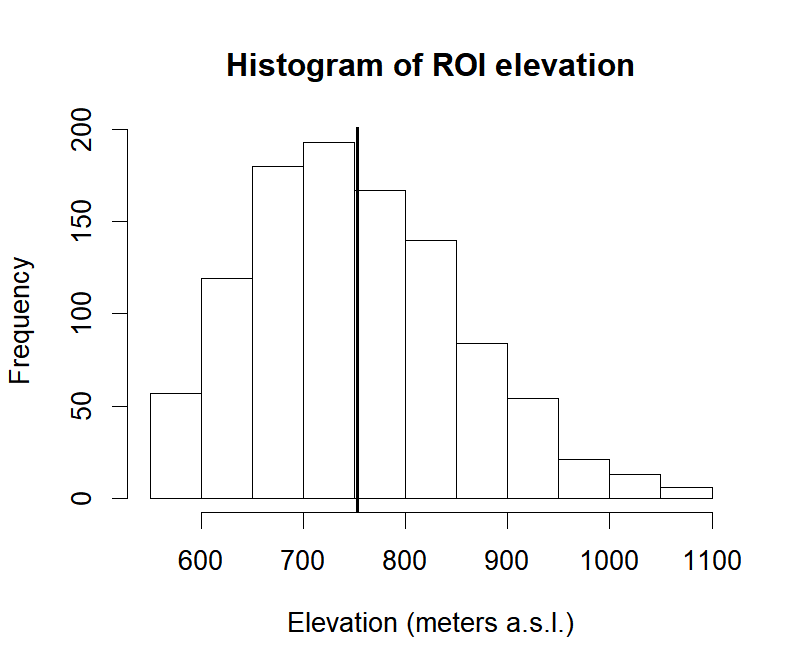
fivenum(elev)

## [1] 556 677 745 822 1086

When using extract with a SpatialPolygons\* object, by default, we get a list containing a set of raster values for each individual polygon.  
Now, using the extracted values, we can investigate the distribution of elevation values for the target patch.

hist(elev, main = "Histogram of ROI elevation", xlab = "Elevation (meters a.s.l.)")

abline(v = mean(elev), lwd = 2) ## Mean line



This concludes our first exploration of the raster package – an awesome resource for handling geospatial data in R! Hope you find this post useful.