Introduction to the analysis of spatial data using R

Ana I. Moreno-Monroy

April 11 2019

Chapter 5: The power of rasters

- Raster data: the basics
- Crop, mask and plot
- Extracting raster data by polygons: an example for Poland

Raster data: the basics

- Raster data is a type of spatial data organised in grid (matrix) format of equally-spaced grid-cells (e.g. 1-square km)
- Each grid-cell of a raster contains information on a continous variable (elevation, tempetarure, population density) for each grid-cell
- Raster can also contain categorical data (resulting from processed continous data, e.g. land use)
- The extent of rasters can be manipulated to match a spatial object in order to extract information from grid-cells

- Raster data can be transformed into spatial data (points or spatial polygons), and spatial data can be converted into a raster (i.e. can be "rasterized")

Raster data features

- Raster data can be quite heavy if it has global extent and high-level of spatial resolution (e.g. 100-square meters)
- Shapefile folders is often available in GeoTIFF format (with .tif or .tiff extension)
- Examples of raster data available for free online are the friction surface (372MB) and accessibility to cities (520MB)
- The main package to work with rasters in R is the **raster** package

Importing a raster into R

- We can use the **raster** function of the **raster** package to import a raster

```r
travel_time <- raster("Q:/accessibility_to_cities_2015_v1.tif")
```

- We can verify that the properties of the imported "Rasterlayer" object share some similarities with other spatial objects (CRS definition) but do not share the same structure

```r
str(travel_time)
```

```
## Formal class 'RasterLayer' [package "raster"] with 12 slots
##   ..@ file    :Formal class '.RasterFile' [package "raster"] with 13 slots
##   .. .. ..@ name        : chr "Q:\\accessibility_to_cities_2015_v1.tif"
##   .. .. ..@ datanotation: chr "INT4S"
##   .. .. ..@ byteorder   : chr "little"
##   .. .. ..@ nodatavalue : num -Inf
##   .. .. ..@ NAchanged   : logi FALSE
```

```
##   .. .. ..@ nbands      : int 1
##   .. .. ..@ bandorder   : chr "BIL"
##   .. .. ..@ offset      : int 0
##   .. .. ..@ toptobottom : logi TRUE
##   .. .. ..@ blockrows   : int 8
##   .. .. ..@ blockcols   : int 43200
##   .. .. ..@ driver      : chr "gdal"
##   .. .. ..@ open        : logi FALSE
##   ..@ data    :Formal class '.SingleLayerData' [package "raster"] with 13 slots
##   .. .. ..@ values    : logi(0)
##   .. .. ..@ offset    : num 0
##   .. .. ..@ gain      : num 1
##   .. .. ..@ inmemory  : logi FALSE
##   .. .. ..@ fromdisk  : logi TRUE
##   .. .. ..@ isfactor  : logi FALSE
##   .. .. ..@ attributes: list()
##   .. .. ..@ haveminmax: logi TRUE
##   .. .. ..@ min       : num -2.15e+09
##   .. .. ..@ max       : num 2.15e+09
##   .. .. ..@ band      : int 1
##   .. .. ..@ unit      : chr ""
##   .. .. ..@ names     : chr "accessibility_to_cities_2015_v1"
##   ..@ legend  :Formal class '.RasterLegend' [package "raster"] with 5 slots
##   .. .. ..@ type      : chr(0)
##   .. .. ..@ values    : logi(0)
##   .. .. ..@ color     : logi(0)
##   .. .. ..@ names     : logi(0)
##   .. .. ..@ colortable: logi(0)
##   ..@ title   : chr(0)
##   ..@ extent  :Formal class 'Extent' [package "raster"] with 4 slots
##   .. .. ..@ xmin: num -180
##   .. .. ..@ xmax: num 180
##   .. .. ..@ ymin: num -60
##   .. .. ..@ ymax: num 85
##   ..@ rotated : logi FALSE
##   ..@ rotation:Formal class '.Rotation' [package "raster"] with 2 slots
##   .. .. ..@ geotrans: num(0)
##   .. .. ..@ transfun:function ()
##   ..@ ncols   : int 43200
##   ..@ nrows   : int 17400
##   ..@ crs     :Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
##   ..@ history : list()
##   ..@ z       : list()
```

Basic raster operations

- The function **getValues** can be used to extract values stored in grid-cells. The function **cellStats** will compute a specified summary statistic (e.g. min, max, mean) across the raster cells
- The value of raster cells can be modified using algebraic operators (+, -, *, /), logical operators (e.g., >, <=), or functions (e.g. abs, min)
- The **raster** package provides its own plot functionality. It can be implemented by simply calling the **plot** function on the raster object

- Large raster files will take time to process. As an example, running **cellStats**(travel_time, 'mean') takes an i7, 2.60GHz laptop 153.3 seconds
- Before plotting or performing other operations on rasters, it is often handy to crop (global/large) rasters to smaller extends to increase speed

Adapting rasters to a given extent

- The function **crop** provides a fast way to obtain a rectangle where a spatial object fits entirely
- As an example, we can crop the accessibility to cities raster to the extent of the NUTS3 regions in Poland
- First we import the polygons, verify the two objects share the same CRS and reproject if necessary:

```
poland_borders <- readOGR(dsn = "Q:/TL2", layer = "Poland_NUTS2")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "Q:\TL2", layer: "Poland_NUTS2"
## with 16 features
## It has 4 fields
```

```
proj4string(poland_borders) ==  proj4string(travel_time)
```

```
## [1] FALSE
```

```
poland_borders <- spTransform(poland_borders, proj4string(travel_time))
```

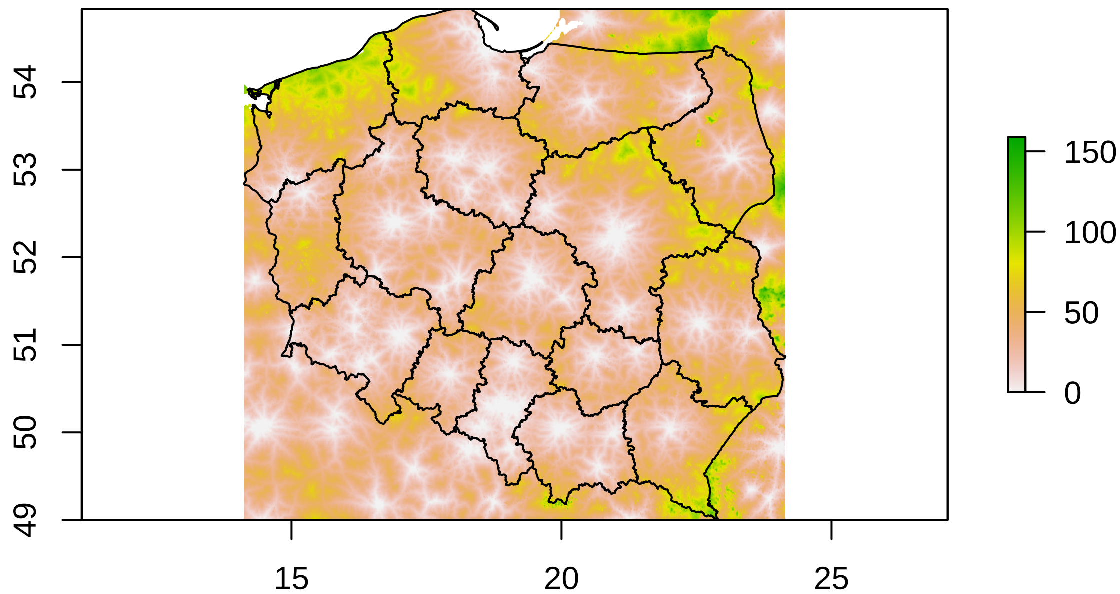
Cropping and plotting

- We then **crop** the raster to the extent of the polygons and **plot** the resulting raster and overlay the polygons to verify the result
- The scale indicates that travel times to the most proximate city within the extent of Poland vary from 0 to 150 minutes. We can get the mean value by using the **cellStats** function

```
travel_time_poland <- crop(travel_time, poland_borders)
cellStats(travel_time_poland, 'mean')
```

```
## [1] 35.7546
```

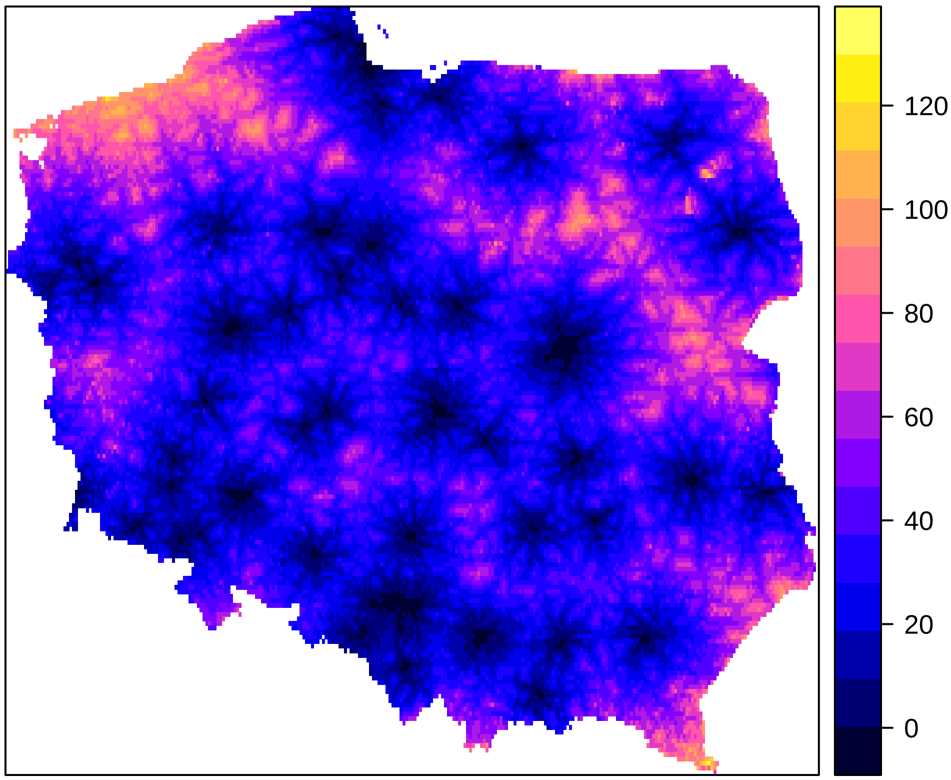
```
plot(travel_time_poland)
plot(poland_borders, add=TRUE)
```



Masking

- If we want the crop the raster object more closely to a spatial object, we can use the function **mask** to set all values in the extent outside polygon borders to NA
- The **mask** function runs slower than **crop**. Running it on rasters of smaller extent (like those that had been previously cropped) can help
- As an alternative to **plot**, we can use the **spplot** function of the raster package to visualise the result

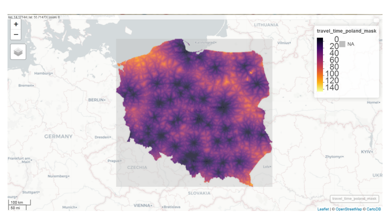
```
travel_time_poland_mask <- mask(travel_time_poland, poland_borders)
spplot(travel_time_poland_mask)
```



Rasters in interactive maps

- To get an interactive map with several options for background tiles we can use **mapview** on the masked raster (though mapview adjusts to a smaller resolution)

```
mapview::mapview(travel_time_poland_mask)
```



The html can be accessed here

Extracting raster data by polygons

- The function **extract** of the **raster** package allows perfoming operations on grid-cells that fall on a given extent, such as sum, mean, and median or any user-speficied function with the option "fun"
- The same operation is performed across each observation of the spatial object when combined with the option "byid=TRUE"
- For instance, for the case of Poland, we can extract the median travel time by NUTS3 region (leaving out NAs) by simply specifying:

```r
median_travel_time <- extract(travel_time_poland, poland_borders, fun=median, na.rm=TRUE)
```

- Which returns a [1:R, 1] matrix, where R is the number of polygons (NUTS3 regions) in our spatial polygon data frame

```r
str(median_travel_time)
```

```
##  num [1:16, 1] 28 33 21 40 38 24 37 13 38 27 ...
```

Merging and plotting extracted data

- As the order of observations is preserved, we can add this variable directly to the SpatialPolygonDataFrame and visualise the result using the **spplot** function from the **sp** package

```r
poland_borders$median_travel_time <- median_travel_time[,1]
summary(poland_borders$median_travel_time)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   13.00   26.25   30.50   32.19   38.50   60.00
```

```r
my.palette <- RColorBrewer::brewer.pal(n = 7, name = "OrRd")
spplot(poland_borders, "median_travel_time", col.regions = my.palette, main = "Minutes to the closest c
```