Session 2: Reading my spatial data into R

userR! 2021 afrimapr tutorial

07 July 2021

Contents

This is Session 2 of the tutorial developed for useR! 2021 on mapping spatial data in R using African data. It is aimed at participants with limited R and GIS experience.

The tutorial is based on the afrilearnr package containing tutorials to teach spatial data skills in R with African data. It is part of the afrimapr project, which is funded through the Wellcome Trust's Open Research Fund and Data for Science and Health.

The tutorial has been adapted for useR! 2021 - please see the online tutorials for more detailed information and lessons.

PDFs of the tutorials have been included in the project documents folder.

A. Outline of this tutorial session

An entry level outline to demonstrate getting spatial data of different types into R. The aim is to support you reading your own data into R before making maps or other plots. It is aimed at participants with limited R and GIS experience.

Learning outcomes

By the end of this session, you will have learnt how to read different spatial data types into R, and how to use these data to create static and interactive maps. Specifically, you will be able to:

- read spatial data in R using `sf` and `raster`
- make a map from tabular data with coordinates included
- understand the importance of CRS (Coordinate Reference System) to place data on a world map
- read in other spatial data files for vector and raster data
- create static and interactive maps using the `mapview` package and example data available from the `afrilearndata` package
- understand options for making `mapview` maps more useful

Please reach out during and/or after the course to the trainers and other participants to address any difficulties you come across.

B. Loading packages and data

For the purposes of this useR! 2021 tutorial, the packages have been installed into the RStudio Cloud project. Because we have already installed the packages for you, you only have to run the code below to load the packages into this working environment. However, if you run this tutorial locally on your own computer, you will need to install these packages if you haven't already done so. You can find a script that will install all the necessary packages in the project main directory `packages_and_data.R`.

```
# for working with vector data
library(sf)

# for raster data handling
library(raster)

# example spatial data for Africa
library(afrilearndata)

# for static and interactive mapping
library(tmap)

# for interactive mapping
library(mapview)

# for reading text files
library(readr)
```

C. Read spatial data from files and preview with `mapview`

So far we have been using data that already exists within R as an R object.

The same things can be done using data coming from a file.

Spatial data can be read into R using `sf::st_read()` for vector data (points, lines and polygons) and the `raster` package for gridded data.

We show examples below using files that are stored in the package. To use with your own data:

- Skip step 1 where the filename is created (You should know what your file is called and where it is saved on your computer)
- Replace filename1 or filename2 with the path to your vector or raster file on your computer

```
library(sf)
filename1 <- system.file("extdata","africountries.shp", package="afrilearndata", mustWork=TRUE)
myobject1 <- sf::st_read(filename1)
```

```
## Reading layer `africountries' from data source
##   `/home/rstudio-user/R/x86_64-pc-linux-gnu-library/4.0/afrilearndata/extdata/africountries.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 51 features and 11 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -17.62504 ymin: -34.81917 xmax: 51.13387 ymax: 37.34999
## Geodetic CRS:  WGS 84
```

```
library(raster)
filename2 <- system.file("extdata","afripop2020.tif", package="afrilearndata", mustWork=TRUE)
myobject2 <- raster::raster(filename2)
```

Remember, for these commands to work, your data already has to be in either vector format (e.g shapefile, geopackage or kml) or in raster format (e.g. .tif or .grd). Find more information on which file formats will work here at https://en.wikipedia.org/wiki/GIS_file_formats.

Earlier we demonstrated using the package `tmap`. Here we are going to introduce the package `mapview` for quickly making interactive maps. In many ways, the interactive maps made by `mapview` are very similar to those from `tmap`. We like `mapview` because it is super-easy to make a first preview map to look at your data. All you need is `mapview([name_of_spatial_object])` and it should make a map. Unlike with `tmap`, you don't need to know whether the object contains points, lines, polygons or raster data, and you don't need to remember any additional options to get that first preview.

Note that interactive maps will not render to pdf using `knit`.

```
library(mapview)
mapview(myobject1)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

```
mapview(myobject2)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
```

```
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Later we will see that `mapview` arguments can be used to give you more control over map appearance.

D. .csv, .txt or .xls file with coordinates

Until now, we have been using files and objects that are already stored as spatial objects - they came from an R spatial package or GIS. Now we are going to move onto looking at data that are not yet in a spatial format. This could be just be a spreadsheet or text file with a spatial component that you have created yourself.

Text files containing point data are one of the commonest file types that we see in small-scale operational mapping of data. Usually these consist of one row per record (e.g. the location of a health facility or disease case or dwelling) with two columns containing the coordinates of the location (e.g. longitude & latitude or x & y), and other columns containing attributes of that location (e.g. facility or disease type).

attribute1	longitude	latitude
location1	-10	20
location2	10	0
...		

These files can be `.csv` comma delimited, or `.txt` tab delimited or various spreadsheet formats including `.xls`.

To map these data in R usually requires a 3 step process:

Here we will demonstrate the 3 steps using some airport data from the excellent ourairports dataset that we have extracted and saved in the afrilearndata package.

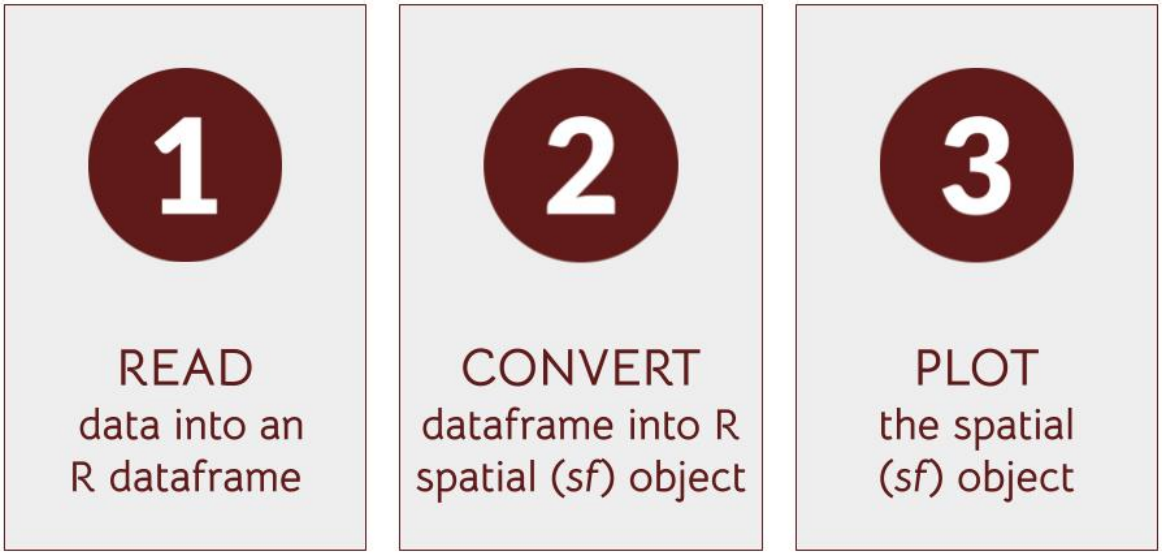

Figure 1: **Mapping steps**

```
# Step 1. Read into dataframe

filename <- system.file("extdata", "afriairports.csv", package="afrilearndata", mustWork=TRUE)
mydf <- readr::read_csv(filename)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   id = col_double(),
##   latitude_deg = col_double(),
##   longitude_deg = col_double(),
##   elevation_ft = col_double(),
##   scheduled_service = col_double(),
##   score = col_double(),
##   last_updated = col_datetime(format = "")
## )
## i Use `spec()` for the full column specifications.
```

```
mydf <- mydf[(1:100), ] # select first 100 rows just to make analysis quicker here
```

```
# Step 2. Convert to sf object and set CRS (Coordinate Reference System)

mysf <- sf::st_as_sf(mydf,
                     coords=c("longitude_deg", "latitude_deg"),
                     crs=4326)
```

```
# Step 3. Quick interactive plot
# Here we are using `mapview` - another package for creating maps that is an alternative way of making
```

```
mapview(mysf)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

To apply the code chunk above to your own data: The below steps apply if you have point data saved in tabular format with two columns for coordinates, containing the latitude and longitude coordinates for each point (row).

- set filename to the path to your file (this might just be something like `"mydata/myfile.csv"`)
- replace `"longitude_deg", "latitude_deg"` with the names of the columns containing the longitude and latitude coordinates in your data
- you may need to change `crs=4326` as explained below

(On RStudio Cloud, you would need to first upload your data, we will come to that in session 3).

CRS

`crs` stands for Coordinate Reference System. It determines how coordinates are converted to a location on the Earth. In this case it tells `sf` what system to expect. In the majority of cases, coordinates (e.g. collected from a GPS) are stored in a system represented by the code `4326`. 4326 is the EPSG (European Petroleum Survey Group) code for longitude, latitude using the WGS84 datum, but you don't really need to know that. 4326 is a good number to remember! If you would like to read more, visit https://epsg.io/about.

See what happens when the `crs=4326` argument is not included in Step 2 in the code below:

```r
# Step 1. Read into dataframe

filename <- system.file("extdata", "afriairports.csv", package="afrilearndata", mustWork=TRUE)
mydf <- readr::read_csv(filename)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   id = col_double(),
##   latitude_deg = col_double(),
##   longitude_deg = col_double(),
##   elevation_ft = col_double(),
##   scheduled_service = col_double(),
##   score = col_double(),
##   last_updated = col_datetime(format = "")
## )
## i Use `spec()` for the full column specifications.
```

```r
mydf <- mydf[(1:100), ] # select first 100 rows just to make quicker online


# Step 2. Convert to sf object - NOTE crs missing

mysf <- sf::st_as_sf(mydf,
                     coords=c("longitude_deg", "latitude_deg"))


# Step 3. Quick interactive plot

mapview(mysf)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

When there is no `crs` argument, the `sf` object is still created but `mapview` is unable to position it in the world. The points still appear but there is no map background.

.xls files

For Microsoft Excel files, you just need to change the code for step 1 in the code we provided above. The steps still stay the same: read, convert, map.

You can read an excel file into a dataframe using the package readxl with something like `readxl::read_excel("[filename]")`. Another option is to save the sheet that you want as a .csv file from MS Excel itself and run through the steps as explained above for .csv files.

E. Directly create an R object

An alternative to loading a file is to directly create a dataframe containing coordinates within R. When you create a dataframe in R, you will not need to read in a file because the data is created within R itself.

In the example below, try changing the coordinates within the dataframe at step 1, and run to see the points change.

```
# 1. create dataframe
mydf <- data.frame(x=c(-10,10,30),
                   y=c(20,0,-20),
                   attribute=c("a","b","c"))

# 2. convert to sf object
mysf <- sf::st_as_sf(mydf,
                     coords=c("x", "y"),
```

```
                    crs=4326)

# 3. quick interactive plot
mapview(mysf)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Note that in this example the coordinates are stored in columns named x & y, which is passed to `sf::st_as_sf` as `coords=c("x", "y")`. To find out more about the arguments for any function you can type `?` and the function name e.g `?st_as_sf`

F. Shapefiles (.shp)

Shapefiles continue to be a common format for spatial data despite the fact that they are rather old now and some things about them are not ideal. One thing that can confuse users is that a shapefile consists of a collection of files with the same name and different suffixes. If some of the files are not present, then it may no longer be possible to work with the data.

e.g. myfile.shp, myfile.shx, myfile.dbf, myfile.prj

If you only have a single file named `*.shp`, you will not be able to map it in R. You need all of the files.

Shapefiles can store points, lines or polygons. The example below uses a shapefile containing polygons.

```
# read file into a spatial object
filename <- system.file("extdata","africountries.shp", package="afrilearndata", mustWork=TRUE)
```

```r
africountries <- sf::read_sf(filename)

# quick interactive plot
mapview(africountries)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

Because shapefiles are spatial files, they can be read directly into a spatial (`sf`) object in R with `sf::read_sf(filename)`. This combines steps 1 and 2 from the csv example. In addition, you don't need to specify which columns contain the coordinates or what the Coordinate Reference System (crs) is. This is effectively because these two steps will have been done when the file was created. In other words, these spatial files already tell R what the values are for those two settings without us having to specify it.

G. .kml, .gpkg and .json

For other spatial vector formats (e.g. kml, geopackage & geojson) the same approach as for a shapefile usually works i.e. `sf::read_sf(filename)`.

Here we show an example with a .kml file of the simplified African highway network from the `afrilearndata` package demonstrated earlier.

```r
filename <- system.file("extdata","trans-african-highway.kml", package="afrilearndata", mustWork=TRUE)

afrihighway <- sf::read_sf(filename)
```

```
# quick interactive plot
#mapviewOptions(fgb = FALSE) # to fix error with mapview rendering the plot on mac
mapview(afrihighway)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

H. raster tiff

To read in raster data, we can use the package `raster` instead of `sf`.

(Note that newer packages `stars` and `terra` may replace `raster` in the coming months and years, but for now we decided to stick with `raster` that is more well known and tested. Feel free to experiment with other packages as you become more confident with mapping in R!)

The reading function in `raster` is also called `raster`. To read in a file use `myrast <- raster::raster(filename)` or just `myrast <- raster(filename)`. Similar to vector formats, you can also use `mapview` to give a quick view of raster objects by simply passing the object name e.g. `mapview(myrast)`.

`raster(filename)` will also work with other raster formats such as ascii grids or .jpg.

```
filename <- system.file("extdata","afripop2020.tif", package="afrilearndata", mustWork=TRUE)

myrast <- raster::raster(filename)

# quick interactive plot
mapview(myrast)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

```
#mapview(myrast, at=c(0,1,10,100,1000,10000,100000))
```

Note that the map above appears mostly dark. This is the same issue we came across in the first session. This is because there are only a few very high density cells and a majority of cells with very low values. This is a common issue with population data. The default, equal-interval classification, doesn't work well, most of the cells on the map fall in the lowest category. If you look very closely you can see a few very high value cells e.g. in Lagos & Cairo.

In session 1 of this tutorial, we fixed the problem in `tmap` using the `breaks=` argument to set the breakpoints between colours. In `mapview` we can achieve the same using `at=`.

To try replace the final line above with this : `mapview(myrast, at=c(0,1,10,100,1000,10000,100000))`. Experiment with different breakpoints.

I. `mapview` options

In these examples we have used `mapview` to give us a quick view by passing it only the spatial object and not specifying any other options. `mapview` is very flexible and can be made much more informative by passing just a few arguments the map. Try copy and pasting this line to replace the final line in the code window below and running it. It uses the columns named `type` and `name` from the datafile to colour and label the points. `cex` sets the size of the points, in this case making them smaller. `mapview(mysf, zcol='type', label='name', cex=2)`

```r
# 1. read into dataframe
filename <- system.file("extdata", "afriairports.csv", package="afrilearndata", mustWork=TRUE)
mydf <- readr::read_csv(filename)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   id = col_double(),
##   latitude_deg = col_double(),
##   longitude_deg = col_double(),
##   elevation_ft = col_double(),
##   scheduled_service = col_double(),
##   score = col_double(),
##   last_updated = col_datetime(format = "")
## )
## i Use `spec()` for the full column specifications.
```

```r
mydf <- mydf[(1:100), ] #select first 100 rows just to make quicker online

# or can select a single country:
# mydf <- mydf[which(mydf$country_name == "Burkina Faso"), ]

# 2. convert to sf object
mysf <- sf::st_as_sf(mydf,
                     coords=c("longitude_deg", "latitude_deg"),
                     crs=4326)

# 3. quick interactive plot
#mapview(mysf)
mapview(mysf, zcol='type', label='name', cex=2)
```

```
## QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-rstudio-user'
## TypeError: Attempting to change the setter of an unconfigurable property.
## TypeError: Attempting to change the setter of an unconfigurable property.
```

To find out more about `mapview` options, type `?mapview` into the console and press enter. This should display the manual page for the `mapview` function in the Help tab of the bottom right pane in RStudio. Scroll down to where you see **Arguments** in bold where it gives more information about settings. Note that not all of these are available for `sf` vector files.

These `mapview` arguments are the most useful :

argument	value	what does it do ?
zcol	a column name	determines how features are coloured and the legend
label	a column name or some text	gives a label that appears when mouse is hovered over
cex	number e.g. 2 or a column name	sets point size to a constant number or the value held in a column
col.regions	`'blue'`	a colour palette or individual colour for circle interiors
color	`'red'`	a colour palette or individual colour for circle borders
alpha.regions	a number between 0 & 1	opacity of the circle interior, 0=invisible
alpha	a number between 0 & 1	opacity of the circle outline, 0=invisible which removes circle border and can be effective

argument	value	what does it do ?
legend	`TRUE` or `FALSE`	whether to plot a legend, TRUE by default
map.types	`c('CartoDB.Positron', 'Esri.WorldImagery', 'OpenStreetMap.HOT')`	background map layer options
at	a series of numeric values e.g. `c(0,1,10)`	breakpoints between colours

Next steps

This is a start; there are plenty of other options for making maps in R. For more detailed information and lessons, please see the online tutorials and the afrilearnr package containing these tutorials on github.

More information on reading different spatial file formats into R can be found in this section in the excellent Geocomputation in R.

Summary

We hope you've enjoyed this brief intro to getting your own spatial data into R.

We've shown you : 1. how to read in data from files using `sf` and `raster` 2. how to make a map from a coordinates text file 3. the importance of CRS (Coordinate Reference System) to place data on a world map 4. how to read in other spatial data files for vector and raster data 5. options for making `mapview` maps more useful

useR! 2021: Next session

Our next session in this useR! 2021 tutorial will be a practical session where you can practice using your own data for mapping. For this, you will need coordinate data to use the skills you have learnt today. It is possible that your data will be in a different format, for example your data may only have names of regions or places and no coordinates. For this, you will need to learn how to join these names to spatial data that include the coordinates. If this is the case, please work through the afrimapr online tutorial on joining; also available via afrilearnr.

If you do not have your own data, or data in a different format, we have provided an example dataset in the data folder in the RStudio Cloud project called `health_demo.csv` (a subset of the dataset: World Health Organization (WHO) (2019) A spatial database of health facilities managed by the public health sector in sub-Saharan Africa. Reference Source).

BREAK TIME OF 15 MINS