afrimapr intro to spatial data in R

A. Outline of this tutorial

This is an entry level introduction to spatial data in R using examples from Africa. It is aimed at those with a little knowledge of R.

Outline of afrilearnr

This tutorial is part of the afrilearnr (https://github.com/afrimapr/afrilearnr) package containing tutorials to teach spatial data skills in R with African data. It is part of the afrimapr (https://afrimapr.github.io/afrimapr.website/) project.

How this tutorial relates to others in afrilearnr

tutorial name	outline	recommended order
`intro-to-spatial-r`	an introduction to spatial data in R	1 this one
`get-my-data-in`	getting your own spatial data into R	2
`join-admin`	dealing with data referenced by names rather than coordinates	3
`afrilearnr-crash-course`	gallery of plots & code with minimal explanation	4

How to use this tutorial

Click 'Next Topic' to move between sections, or select section headings on the left. If you want to return the tutorial to its original state you can press 'Start Over' on the upper left.

Through the magic of learnr (https://rstudio.github.io/learnr/) you can modify the R code in the boxes below and press run to see results.

If you are accessing this from shinyapps, another option is to install afrilearnr from github (https://github.com/afrimapr/afrilearnr) and run these tutorials locally.

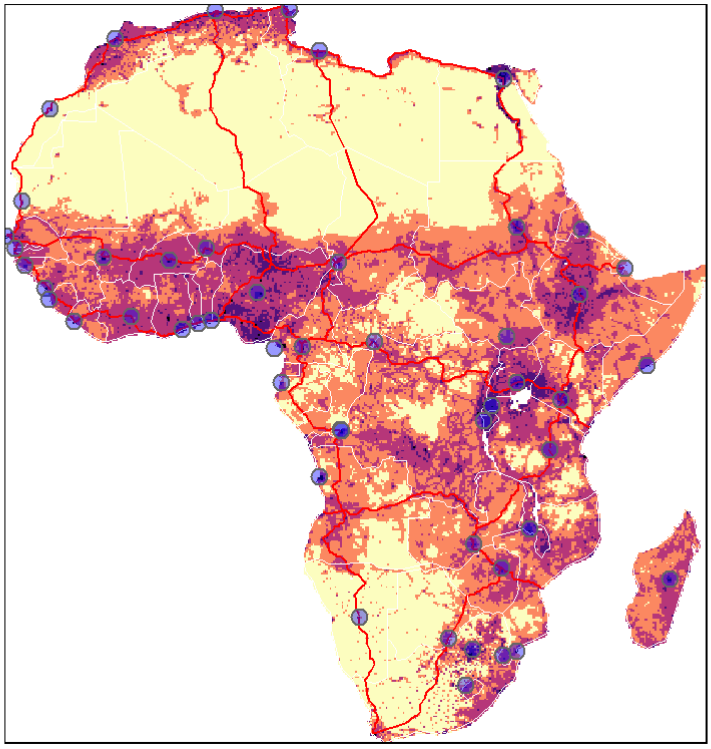
A third option, for lower bandwidths, is to download a pdf of the tutorial (https://github.com/afrimapr/afrilearnr/tree/main/inst/pdfs) and run through by copying the code into an R console.

B. Spatial data outline

We will start by looking at these spatial data for Africa, shown in the map below.

1. Capital city locations (points)
2. A highway network (lines)

3. Country boundaries (polygons)
4. Population density (gridded or raster data)



Cities, highways and boundaries are examples of point, line and polygon data termed **vector data**.

The gridded population density data are termed **raster data**.

In R there is often more than one package that does the same thing. Which one is 'best' for you may depend on preference and can change over time. This is true for R spatial operations.

In R the `sf` package deals with vector data (points, lines and polygons), and the `raster` package deals with raster data.

There are other packages too but we don't need those for now.

C. Loading packages and data

Packages in R contain extra methods and data to add to base R.

We will be loading a package (`afrilearndata`) containing example data for us to look at.

We will also use the packages `sf` and `raster` , which allow us to deal with vector and raster data.

Using an R package requires a 2 step process :

1. `install.packages` needed only once to install a package from the internet
2. `library([package_name])` needed each time you start a new R session

A package may have been installed on your system already because it is a 'dependency' needed for another package.

In this case the packages `afrilearndata` , `sf` and `raster` should already have been installed when you installed this package.

To check that the packages have been installed, try running the `library` commands below. If they have been installed, nothing should happen. Not very interesting but a good check and a good reminder that this is what you will need to do each time you start a new R session.

R Code | Start Over | Run Code

```
# for vector data handling
library(sf)

# for raster data handling
library(raster)

# example spatial data for Africa
library(afrilearndata)

# for mapping
library(tmap)
```

If you happen to get messages indicating any of the packages are not installed, you can use `install.packages` to install them.

D. Spatial data objects

We are going to take a look at the spatial data objects used to create the map shown at the start of the tutorial.

We call them 'objects' because the data are already stored in R. This is also to make clear the difference from a 'file' that is stored elsewhere on your computer. A 'file' can be read into an R 'object' and we will come to that later.

First we will look at capital cities which are stored in an object called `africapitals`.

Using the plot method should display a number of maps of the point locations of African capitals.

R Code | Start Over | Run Code

```
plot(africapitals)
#plot(sf::st_geometry(africapitals))

```

This uses the plot function defined in the `sf` package. It creates a series of maps and in each one the points are coloured according to the values stored in one column. There is a function in `sf` called `st_geometry()` that allows you to get just the spatial parts without the attributes and you can see the result of that by removing the # in front of the 2nd line of code in the window above and pressing 'Run code'. But we are getting ahead of ourselves; let us have a look at the structure of the object itself.

In R there are various functions that can help us explore what an object contains. We find these particularly useful; there is some overlap between them.

1. str() structure of the object, displays both names and values
2. head() displays the first few rows of data with the column names
3. names() gives just column names
4. class() gives the class of the object, that is broadly what sort of object it is

Have a look at the outputs for `africapitals` (uncomment the later lines to see their outputs) :

R Code | Start Over | Run Code

```
str(africapitals)
#head(africapitals)
#names(africapitals)
#class(africapitals)
```

These show us that `africapitals` is of class `sf` and `data.frame` and contains a series of columns including ones named : ‘capitalname’, ‘countryname’ and ‘geometry’.

`data.frame` , often referred to as just dataframe, is the most common object type in R certainly for new users. Dataframes store data in rows and named columns like a spreadsheet.

`sf` objects are a special type of dataframe with a column called ‘geometry’ that contains the spatial information, and one row per feature. In this case the features are points.

If you look at the output from the `str()` command above you should see that the first value in the geometry column has the coordinates 7.17 9.18. Because the capitals data are points, they just have a single coordinate pair representing the longitude and latitude of each capital.

The highway and countries objects are also of class `sf` and contain geometry columns. You can uncomment the lines below and run to see what is contained in the first cell of the geometry column for the other vector objects.

R Code | Start Over | Run Code

```
#The `paste()` command converts the object to text to ensure it appears in the learnr window

paste(africapitals$geometry[1])
#paste(afrihighway$geometry[1])
#paste(africountries$geometry[1])
```

You should have seen that the geometry columns for the other objects contain multiple coordinates representing lines and polygons.

E. First maps with `tmap`

There are a number of packages for making maps that extend what is available from `sf` .

Package `tmap` is a good place to start; it offers both static and interactive mapping.

We can start with static plots of the capitals (points).

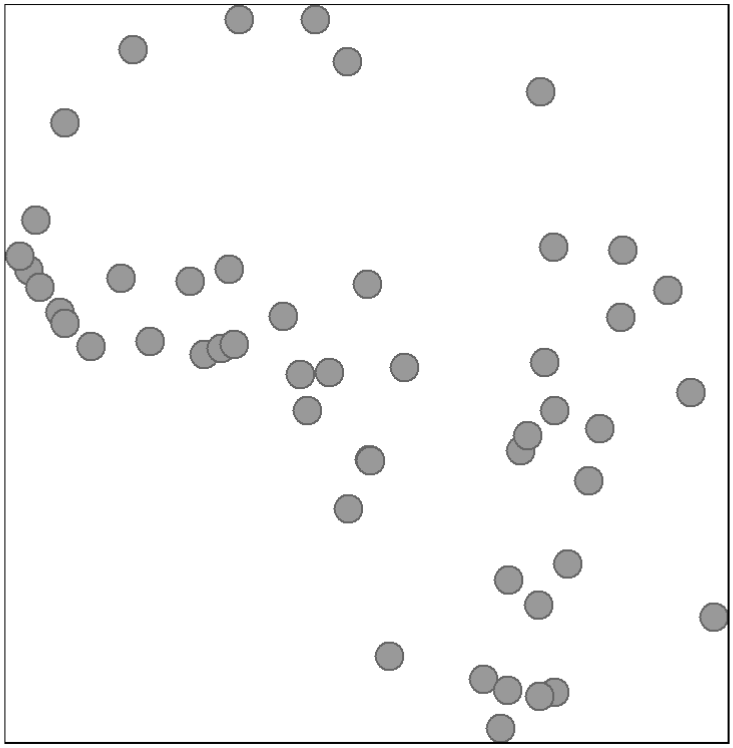
In `tmap` `tm_shape([object_name])` defines the data to be used. Then `+` to add code that defines how the data are displayed, e.g. `tm_symbols()` for points. Extra arguments can be specified to modify the data display.

See the hint button for how to set colour with `tm_symbols(col = "blue")` and `col=[column_name]` to set the colour of each point according to the data value stored in a column.

R Code | Start Over | Hint | Run Code

```
tm_shape(africapitals) +
  tm_symbols()              #default grey points

```

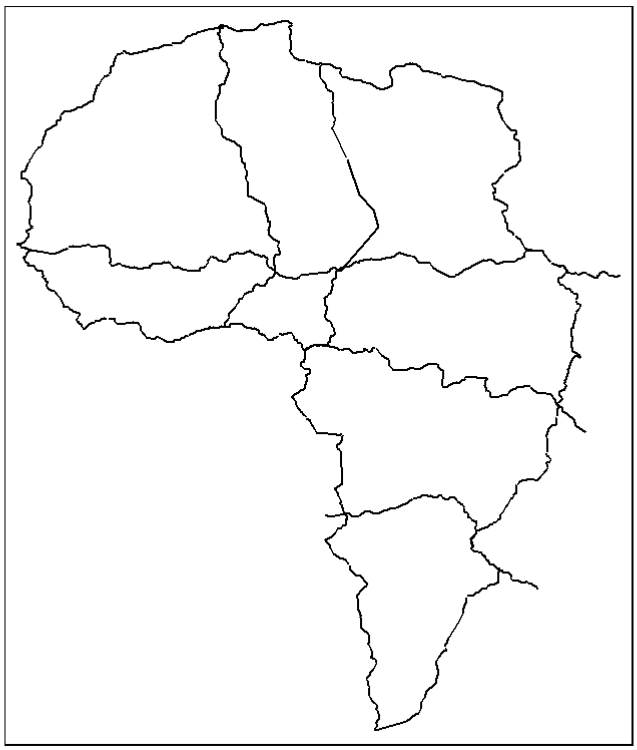


The highway network (lines) can be plotted using the same `tm_shape([object_name])` to start, then adding `tm_lines()` to display the lines. The hints button below shows options for colouring lines.

R Code | Start Over | Hint | Run Code

```
tm_shape(afrihighway) +
    tm_lines()

```

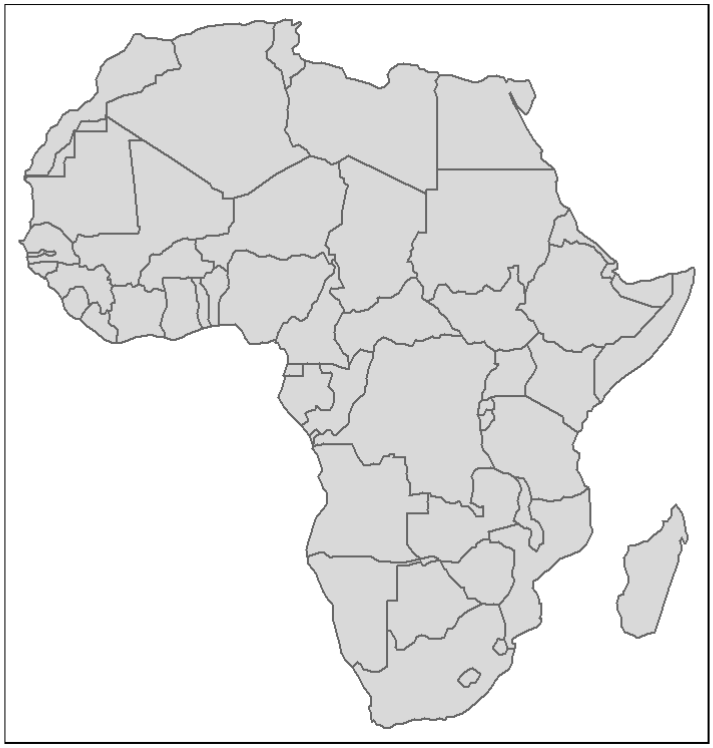


Countries (polygons) can similarly be mapped using `tm_shape` and `tm_polygons`. See the hint button for options for colouring countries.

R Code | Start Over | Hint | Run Code

```
tm_shape(africountries) +
    tm_polygons()

```



Gridded (raster) data can represent e.g. remotely sensed or modelled data.

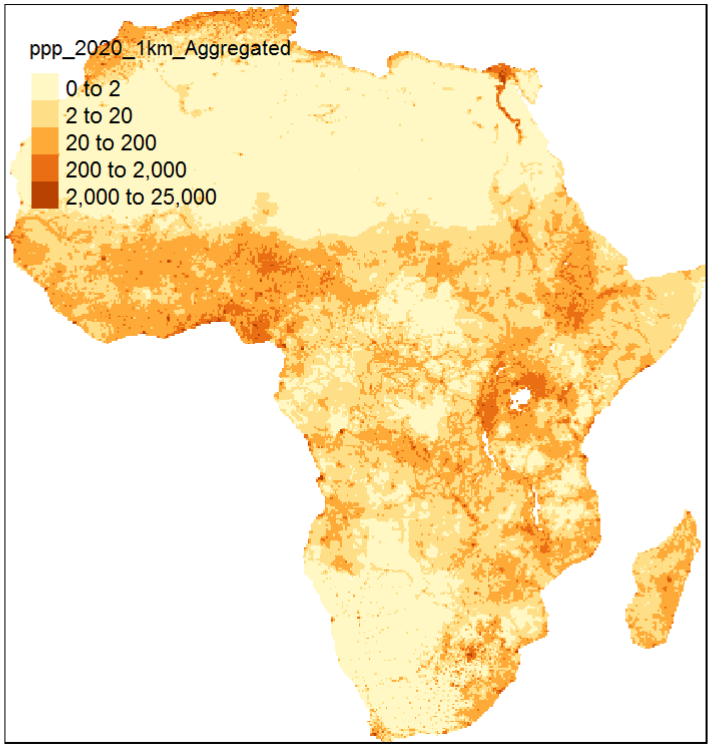
It can be displayed with `tm_shape([object_name])` & `tm_raster`. Here we specify the `breaks` or cutoffs to the different colour bands.

In this example, if you use the default breaks by not specifying any arguments with `tm_raster()` (see the hint) the map looks entirely one colour. This is because there are 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 map falls in the lowest category. If you look very closely you can see a few very high value cells e.g. in Lagos & Cairo.

R Code | Start Over | Hint | Run Code

```
tm_shape(afripop2020) +
    tm_raster(breaks=c(0,2,20,200,2000,25000))

```



F. Mapping multiple ‘layers’

In the previous section we showed how to make maps of individual data objects. Those sections of code can be combined to create multiple ‘layer’ maps as shown in the example below.

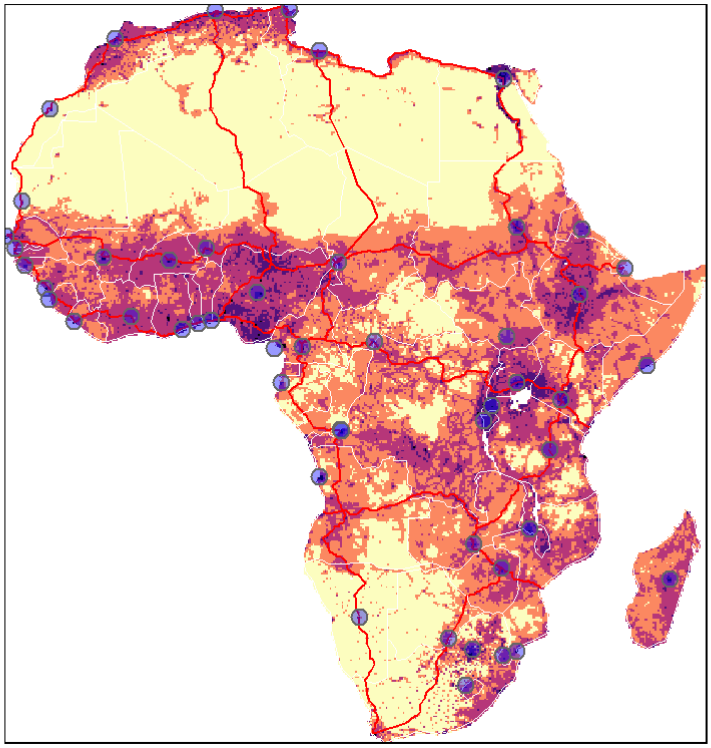
`tmap` (and other map packages) use the `+` symbol to combine layers.

Experiment with commenting out & in lines in the code below by adding and removing # at the start of lines and pressing Run Code.

Try to make maps : 1. without the highway network 2. without the raster population layer & with country boundaries that are visible 3. with text labels for ISO country codes

R Code | Start Over | Run Code

```
tmap::tm_shape(afripop2020) +
    tm_raster(palette = rev(viridisLite::magma(5)), breaks=c(0,2,20,200,2000,25000)) +
tm_shape(africountries) +
    tm_borders("white", lwd = .5) +
    #tm_text("iso_a3", size = "AREA") +
tm_shape(afrihighway) +
    tm_lines(col = "red") +
tm_shape(africapitals) +
    tm_symbols(col = "blue", alpha=0.4, scale = .6 ) +
tm_legend(show = FALSE)
```



G. Interactive maps

The maps created so far have been static. There are also great options for creating interactive maps, which are useful for web pages or reports where readers can zoom, pan and enable/disable layers.

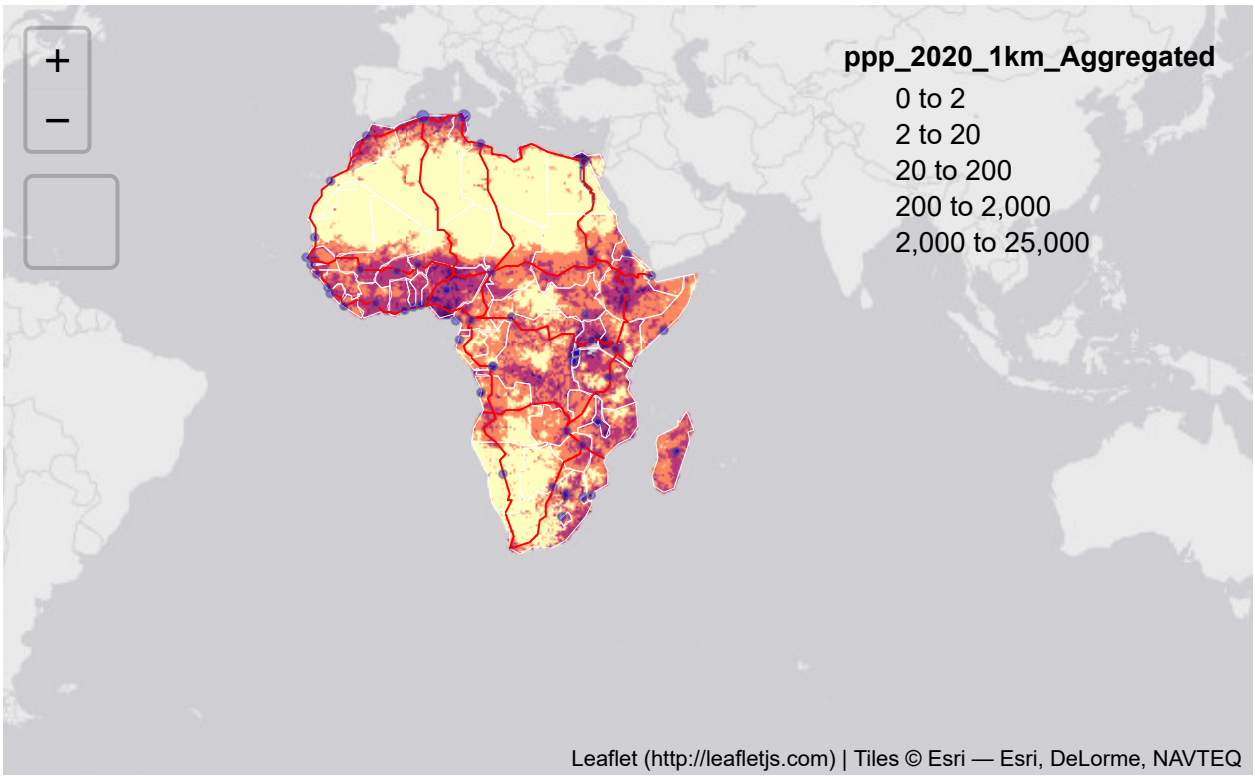
In `tmap` you can keep the identical code that we've looked at so far and just add a single line before : `tmap_mode('view')` to change to interactive 'view' mode. View mode will remain active for your R session and you can switch back to static `plot` mode using `tmap_mode('plot')`.

This is the identical code from the previous section but shown in view mode.

R Code | Start Over | Run Code

```
tmap_mode('view')

tmap::tm_shape(afripop2020) +
    tm_raster(palette = rev(viridisLite::magma(5)), breaks=c(0,2,20,200,2000,25000)) +
tm_shape(africountries) +
    tm_borders("white", lwd = .5) +
    #tm_text("iso_a3", size = "AREA") +
tm_shape(afrihighway) +
    tm_lines(col = "red") +
tm_shape(africapitals) +
    tm_symbols(col = "blue", alpha=0.4, scale = .6 )+
tm_legend(show = FALSE)
```



You may want to go back to the earlier plots and see how they are modified by adding `tmap_mode('view')` before the code.

H. Read spatial data from files

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

The same things can be done on 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 replace filename1 & 2. (Note that these specified file names can also be a URL for data provided on the web.)

R Code | Start Over | Run Code

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

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

Summary

Good persistence for getting this far !

We hope you've enjoyed this brief intro to mapping with R.

We've shown you :

i. storing and handling spatial data with the package `sf`
ii. making static & interactive maps with package `tmap`
iii. reading in data from files using `sf` and `raster`

This is a start; there are plenty of other options (e.g. maps can also be made with the packages `mapview` & `ggplot2`). We will cover other options in later tutorials. Have a look at the afrilearnr-crash-course (https://andysouth.shinyapps.io/afrilearnr-crash-course/) for some potential next steps. We welcome feedback - different ways of getting in touch are on our website (https://afrimapr.github.io/afrimapr.website/get-involved/).

There is a short quiz if you'd like to test your learning.

Quiz

Quiz

What is the term for point, line and polygon data ?

- ◯ raster
- ◯ sf
- ◯ vector
- ◯ squiggles

Submit Answer

In an `sf` object, which column contains spatial information ?

- ◯ spatial
- ◯ longitude
- ◯ geometry
- ◯ name

Submit Answer

How can I read a shapefile into R ?

- ◯ read.csv()
- ◯ <-
- ◯ myobject <- sf::st_read()
- ◯ tmap

Submit Answer

Which R packages can be used to make maps ?

- ☐ tmap
- ☐ raster
- ☐ ggplot
- ☐ sf
- ☐ mapview

Submit Answer