Session 1: Introduction to spatial data

userR! 2021 afrimapr tutorial

07 July 2021

Contents

This is a 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

This is an entry level introduction to spatial data in R using examples from Africa.

Learning outcomes

By the end of this session, you will have learnt how to store and handle spatial data, and how to make static and interactive maps. Specifically, you will be able to:

- recall R functions that are used in mapping
- understand the classification of different spatial data types
- use packages in R to work with vector and raster data (sf and raster)
- create static and interactive maps using the tmap package and example data available from the afrilearndata package
- overlay several data types (layers) on a map
- learn how to change the colour palette to represent data on a map

Please do 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

Packages in R contain extra methods and data to add to base R. Think of R like a mobile phone that comes with basic functionality, but each person can then choose apps (packages) to install to be able to do more specific tasks.

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

We will also use packages that allow us to deal with spatial data. Cities, highways and boundaries are examples of point, line and polygon data termed **vector data**, while data such as gridded population density are termed **raster data**. We explain this in more detail in Section C below.

The packages `sf` and `raster` allow us to deal with vector and raster data.

Using an R package requires a 2 step process:

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

These two steps are like installing an app on your mobile phone (typically you only have to do that once unless you delete the app) and opening the app (which you have to do every time you use the specific app).

To check that the packages have been installed, try running the `library([package_name])` commands below. If they have been installed, nothing should happen. R will only give a message if there is a problem but not when this command is run successfully.

If you happen to get messages indicating any of the packages are not installed, you can use `install.packages("[package_name]")` to install them. 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 (open) the packages into this working environment (see step 2 above). 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`.

```r
#### SECTION B: LOADING PACKAGES AND DATA ----

# for vector data handling
library(sf)

# for raster data handling
library(raster)

# example spatial data for Africa
library(afrilearndata)

# for static and interactive mapping
library(tmap)

# to create RasterLayer object
library(rgdal)
```

C. Spatial data overview

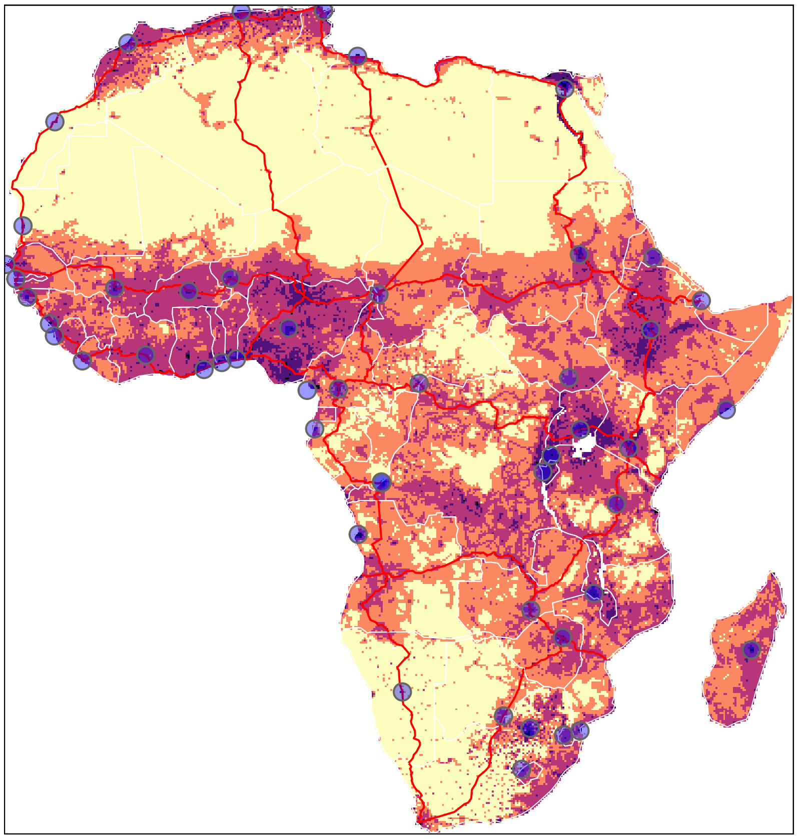
- Cities, highways and boundaries are examples of point, line and polygon data termed **vector data**. Vector data typically represents discrete objects e.g. fire hydrants, roads, dams, airports, countries. Vector data can typically be thought of as tabular data with a column for coordinates and other columns with attributes.

- **Raster data** is continuous data, which does not have clear and definable boundaries, but rather shows varying information such as gridded population density, temperature, precipitation, elevation. It's a grid of regularly sized pixels.

We will start by looking at these spatial data for Africa, shown in the map below, using data from the `afrilearndata` package, which is part of the `afrimapr` project. These include:

1. Capital city locations (points) `africapitals`
2. A highway network (lines) `afrihighway`
3. Country boundaries (polygons) `africountries`
4. Population density (gridded or raster data) `afripop2020`

Now, to view these data on a map:



In R there is often more than one package that does the same thing. Which one is 'best' for you can depend on preference and context 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.

D. Spatial data objects

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

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.

In R there are various functions that can help us explore what an object contains. There is some overlap between them, but we find these particularly useful:

1. `str()` shows the 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`:

```
# sf-points-str

str(africapitals)
```

```
## Classes 'sf' and 'data.frame':   50 obs. of  5 variables:
##  $ capitalname: chr  "Abuja" "Accra" "Addis Abeba" "Algiers" ...
##  $ countryname: chr  "Nigeria" "Ghana" "Ethiopia" "Algeria" ...
##  $ pop        : int  178462 2029143 2823167 2029936 1463754 578860 1342519 547668 34388 404119 ...
##  $ iso3c      : chr  "NGA" "GHA" "ETH" "DZA" ...
##  $ geometry   :sfc_POINT of length 50; first list element:  'XY' num  7.17 9.18
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA
##   ..- attr(*, "names")= chr [1:4] "capitalname" "countryname" "pop" "iso3c"
```

```
# sf-points-head

head(africapitals)
```

```
## Simple feature collection with 6 features and 4 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -0.2 ymin: -18.89 xmax: 47.51 ymax: 36.77
## Geodetic CRS:  WGS 84
##        capitalname countryname     pop iso3c             geometry
## 280          Abuja     Nigeria  178462   NGA    POINT (7.17 9.18)
## 308          Accra       Ghana 2029143   GHA    POINT (-0.2 5.56)
## 382    Addis Abeba    Ethiopia 2823167   ETH   POINT (38.74 9.03)
## 996        Algiers     Algeria 2029936   DZA   POINT (3.04 36.77)
## 1584 Antananarivo  Madagascar 1463754   MDG POINT (47.51 -18.89)
## 2193        Asmara     Eritrea  578860   ERI  POINT (38.94 15.33)
```

```
# sf-points-names

names(africapitals)
```

```
## [1] "capitalname" "countryname" "pop"         "iso3c"       "geometry"
```

```
# sf-points-class

class(africapitals)
```

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

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.

E. First maps with `tmap`

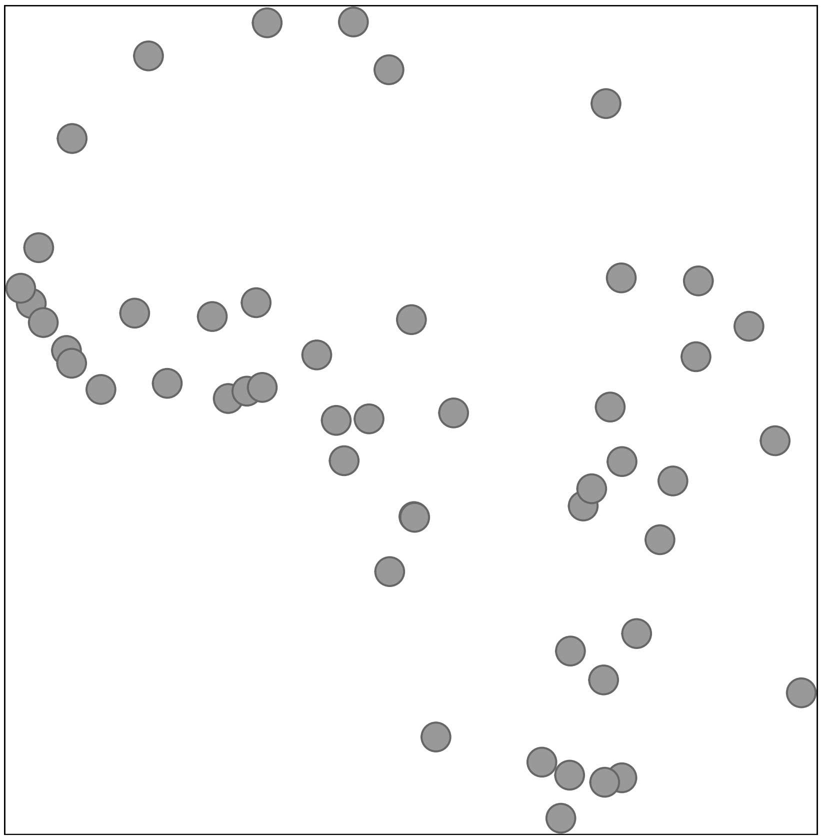
There are a number of packages for making maps that extend what is available from `sf`.

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

Vector data: points 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.

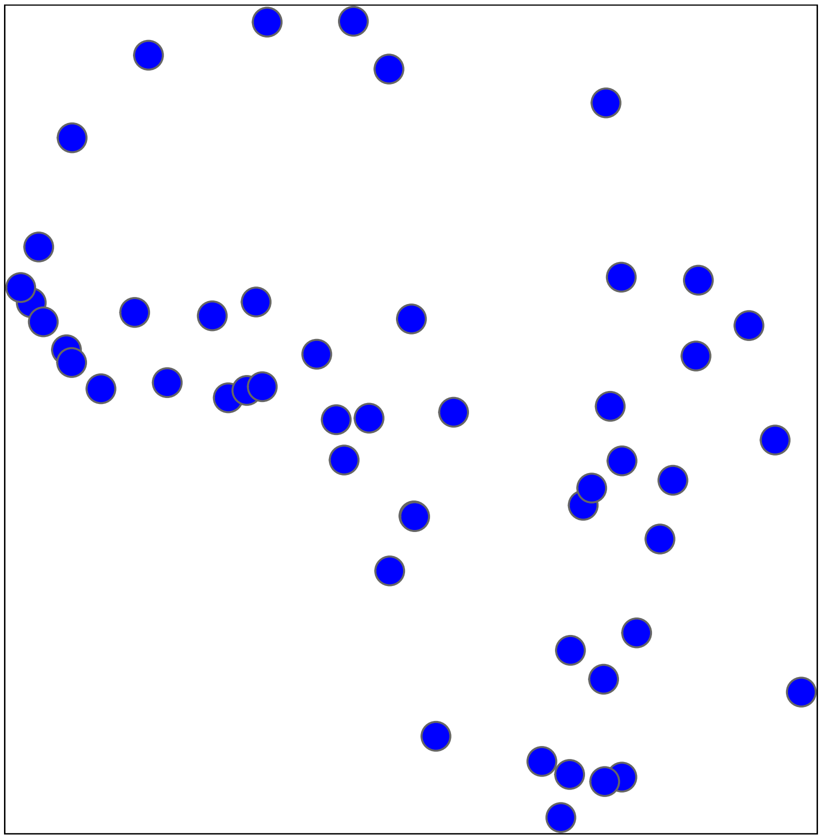
```
# tmap-points1a
tmap_mode('plot')
tm_shape(africapitals) +
   tm_symbols()
```



See how to set colour with `tm_symbols(col = "blue")`. Try changing this to other colour names. You may find that not all colours that you can think of are recognised by R. (Typing `colours()` will give you a list of > 600 colours that R does recognise).

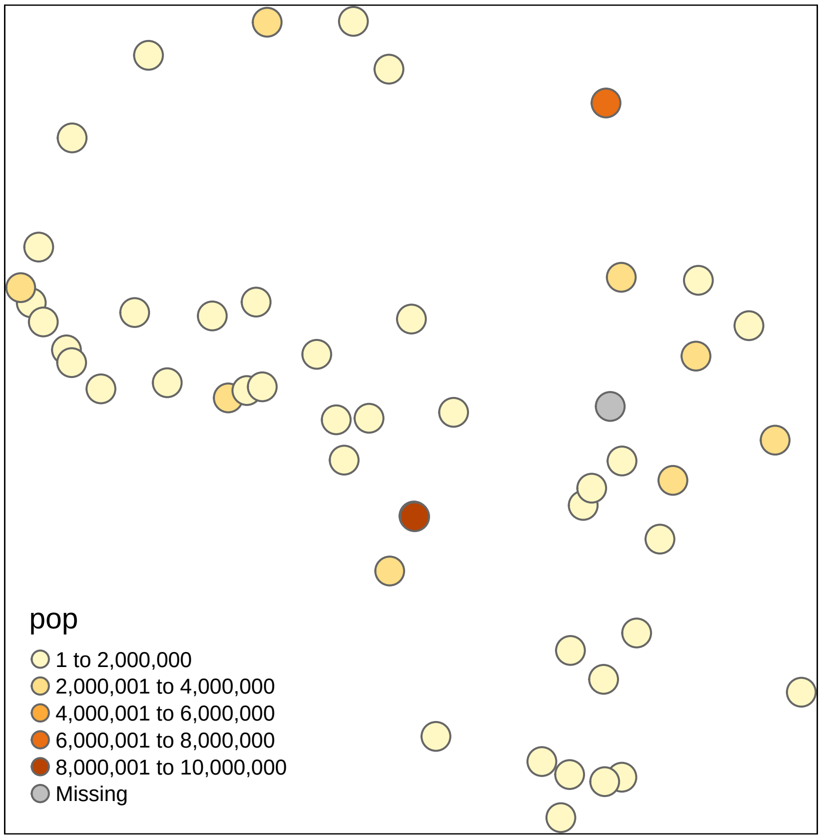
```
# tmap-points1b
tmap_mode('plot')
```

```r
tm_shape(africapitals) +
   tm_symbols(col = "blue")
```



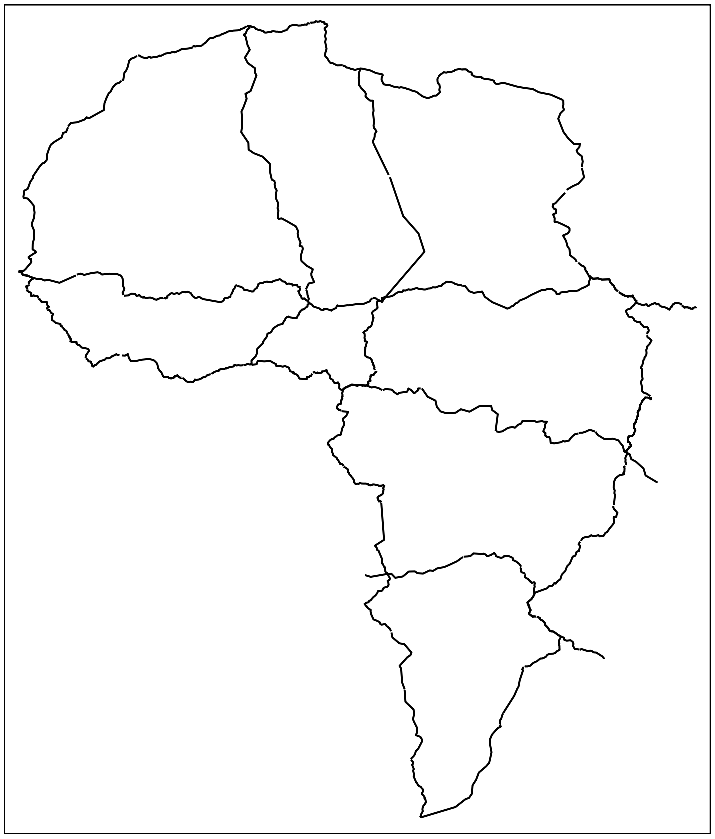
Above we set the colour of all points to be the same. It is also possible to set the colour of each feature (row) to be dependent on the value stored in one of the columns of the dataframe using `col=[column_name]`.

```r
# tmap-points1c
tmap_mode('plot')
tm_shape(africapitals) +
  tm_symbols(col = "pop")
```



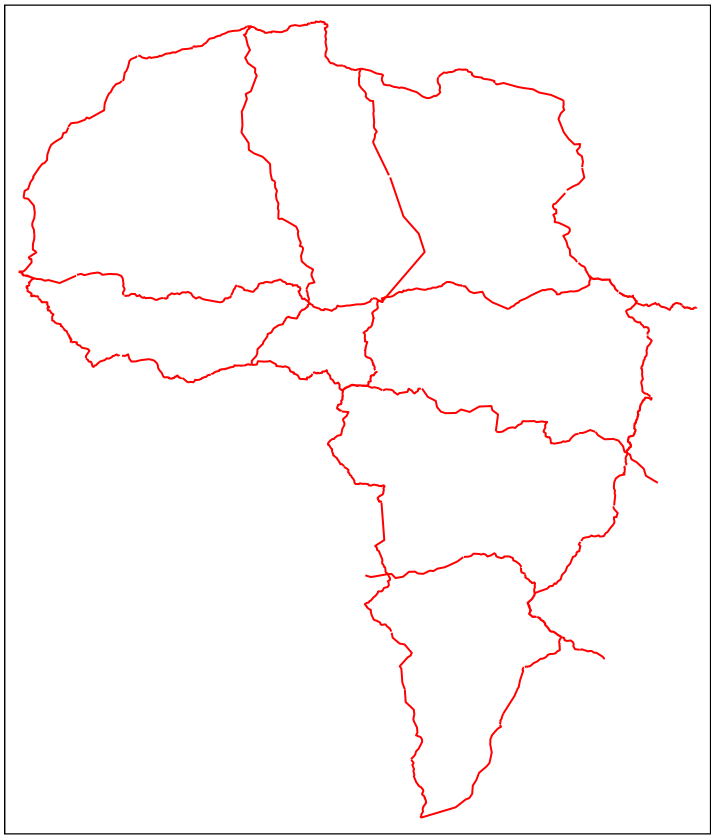
Vector data: lines The highway network (lines) can be plotted using the same `tm_shape([object_name])` to start, then adding `tm_lines()` to display the lines. See other options below for colouring lines.

```
# tmap-lines1a
tmap_mode('plot')
tm_shape(afrihighway) +
  tm_lines()
```



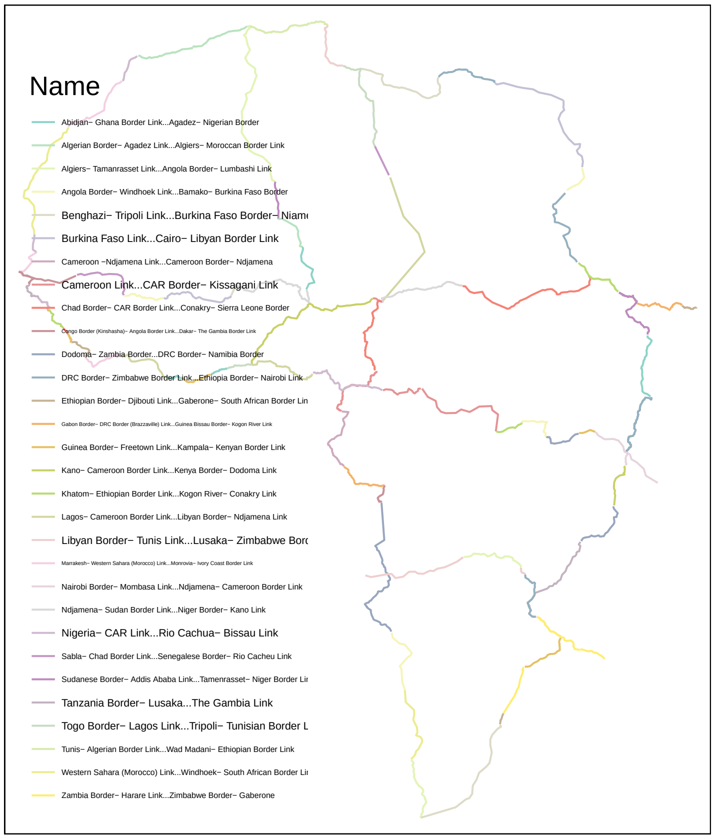
One colour for all lines:

```
# tmap-lines1b
tmap_mode('plot')
tm_shape(afrihighway) +
    tm_lines(col = "red")
```



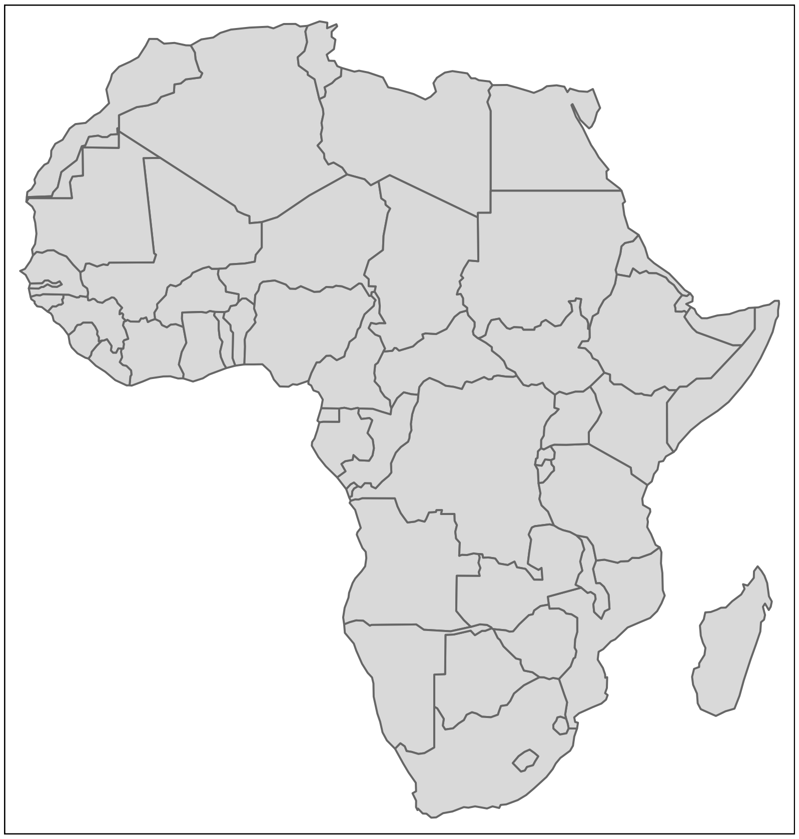
Colour dependent on the specific value in the provided dataframe column for each feature:

```
# tmap-lines1c
tmap_mode('plot')
tm_shape(afrihighway) +
   tm_lines(col = "Name")  # use a column name from the object
```

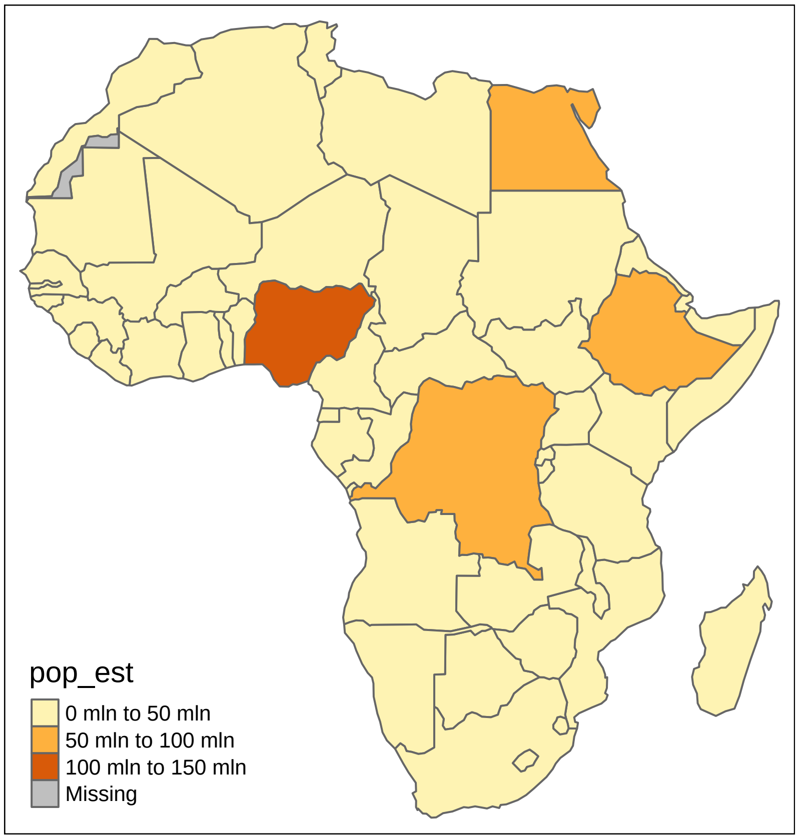

Name
Abidjan− Ghana Border Link...Agadez− Nigerian Border
Algerian Border− Agadez Link...Algiers− Moroccan Border Link
Algiers− Tamanrasset Link...Angola Border− Lumbashi Link
Angola Border− Windhoek Link...Bamako− Burkina Faso Border
Benghazi− Tripoli Link...Burkina Faso Border− Niame
Burkina Faso Link...Cairo− Libyan Border Link
Cameroon −Ndjamena Link...Cameroon Border− Ndjamena
Cameroon Link...CAR Border− Kissagani Link
Chad Border− CAR Border Link...Conakry− Sierra Leone Border
Congo Border (Kinshasha)− Angola Border Link...Dakar− The Gambia Border Link
Dodoma− Zambia Border...DRC Border− Namibia Border
DRC Border− Zimbabwe Border Link...Ethiopia Border− Nairobi Link
Ethiopian Border− Djibouti Link...Gaberone− South African Border Lin
Gabon Border− DRC Border (Brazzaville) Link...Guinea Bissau Border− Kogon River Link
Guinea Border− Freetown Link...Kampala− Kenyan Border Link
Kano− Cameroon Border Link...Kenya Border− Dodoma Link
Khatom− Ethiopian Border Link...Kogon River− Conakry Link
Lagos− Cameroon Border Link...Libyan Border− Ndjamena Link
Libyan Border− Tunis Link...Lusaka− Zimbabwe Borc
Marrakesh− Western Sahara (Morocco) Link...Monrovia− Ivory Coast Border Link
Nairobi Border− Mombasa Link...Ndjamena− Cameroon Border Link
Ndjamena− Sudan Border Link...Niger Border− Kano Link
Nigeria− CAR Link...Rio Cachua− Bissau Link
Sabla− Chad Border Link...Senegalese Border− Rio Cacheu Link
Sudanese Border− Addis Ababa Link...Tamenrasset− Niger Border Lir
Tanzania Border− Lusaka...The Gambia Link
Togo Border− Lagos Link...Tripoli− Tunisian Border L
Tunis− Algerian Border Link...Wad Madani− Ethiopian Border Link
Western Sahara (Morocco) Link...Windhoek− South African Border Lir
Zambia Border− Harare Link...Zimbabwe Border− Gaberone

Vector data: polygons Countries (polygons) can similarly be mapped using `tm_shape` and `tm_polygons`. Similar to above, see other options for colouring countries.

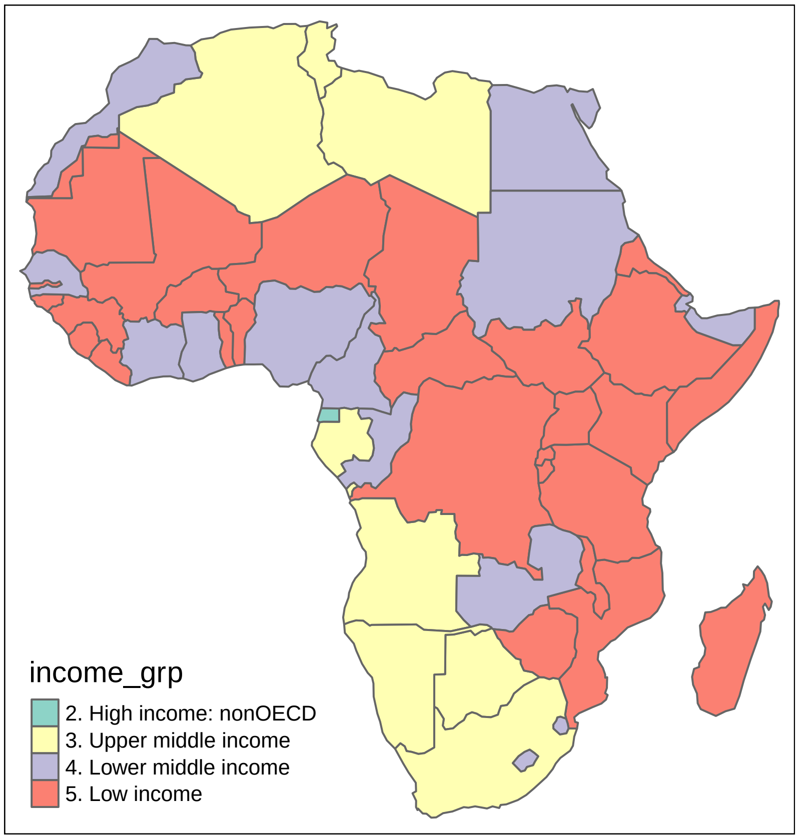
```
# tmap-polygons-1a
tmap_mode('plot')
tm_shape(africountries) +
     tm_polygons()
```



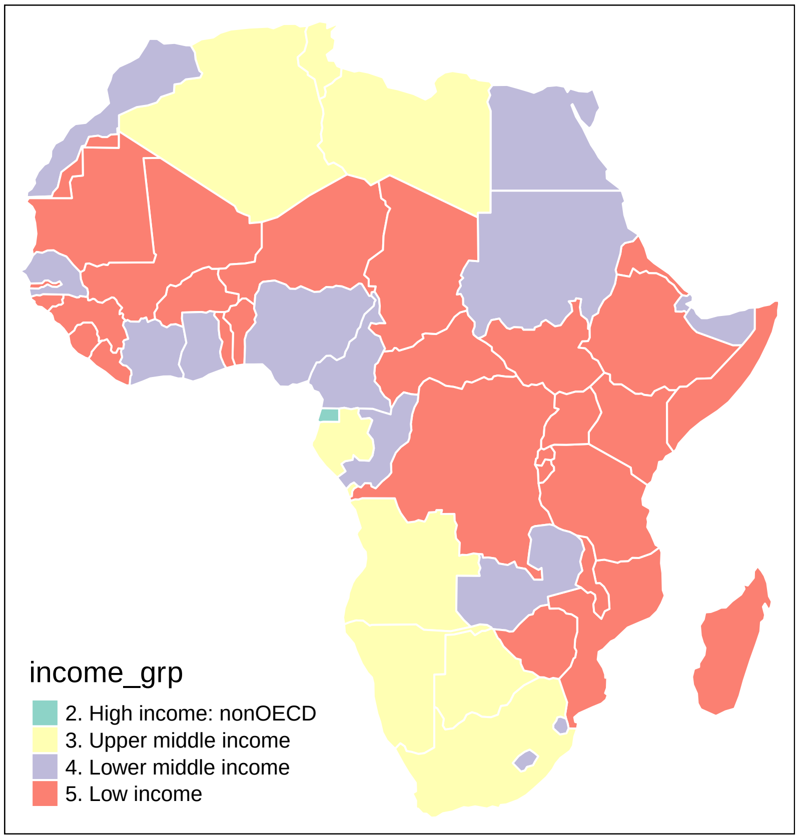
```
# tmap-polygons-1b
tmap_mode('plot')
tm_shape(africountries) +
      tm_polygons(col="pop_est")
```



```
# tmap-polygons-1c
tmap_mode('plot')
tm_shape(africountries) +
      tm_polygons(col="income_grp")
```



```r
# tmap-polygons-1d
tmap_mode('plot')
tm_shape(africountries) +
  tm_polygons(col="income_grp", border.col = "white")
```

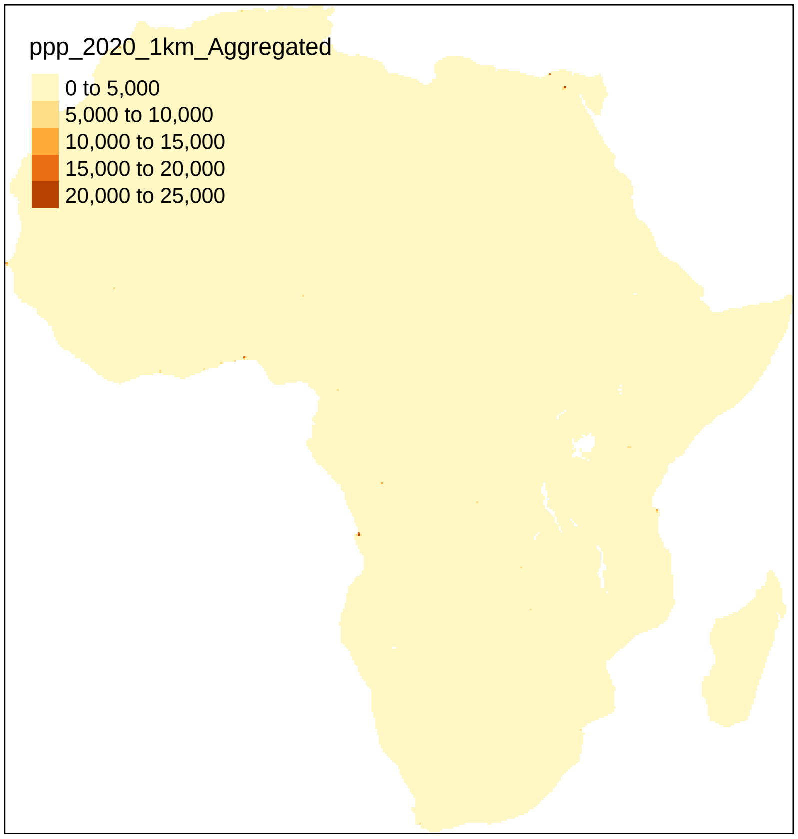


Raster data Gridded (raster) data can represent e.g. remotely sensed or modelled data.

It can be displayed with `tm_shape([object_name])` and `tm_raster`.

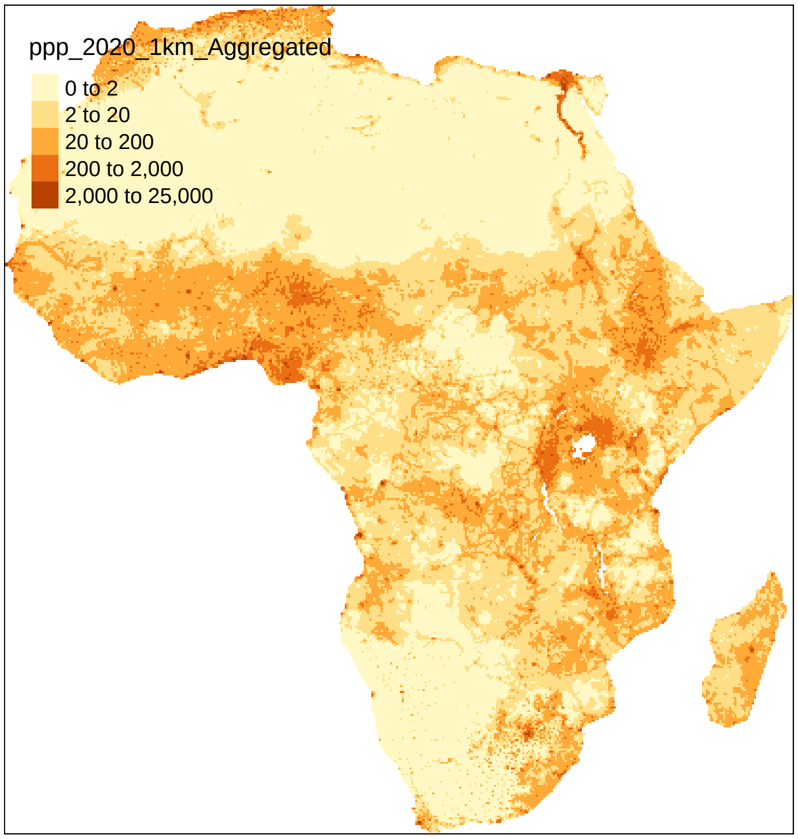
In this example, if you use the default breaks by not specifying any arguments with `tm_raster()`, the map looks as if it is entirely made up of one colour. This is because there are a few very high density cells and a majority of cells with very low values. This is a common issue with population data. The default (equal-interval) classification doesn't work well; most of the map falls in the lowest category. If you look very closely you can see a few very high value cells e.g. in Lagos and Cairo.

```
# tmap-raster1a
tmap_mode('plot')
tm_shape(afripop2020) +
  tm_raster()
```



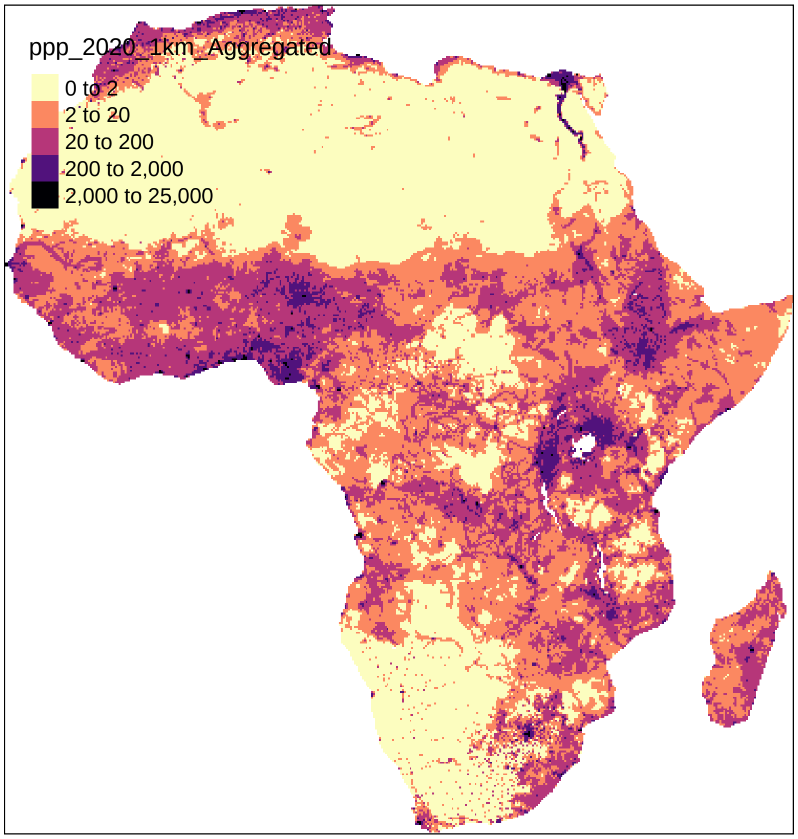
We can specify the `breaks` or cutoffs for different colours to show differences between the lower values more clearly. Experiment with changing the `breaks=` values.

```
# tmap-raster1b
tmap_mode('plot')
tm_shape(afripop2020) +
  tm_raster(breaks=c(0,2,20,200,2000,25000))
```



As well as changing the breaks, we can also change the palette of colours used with `palette =`. In this example we use `rev` to reverse the order of the colours to make higher population values darker. Try removing `rev` and look at the result.

```r
tmap_mode('plot')
#changing the colour palette
tm_shape(afripop2020) +
  tm_raster(palette = rev(viridisLite::magma(5)), breaks=c(0,2,20,200,2000,25000))
```



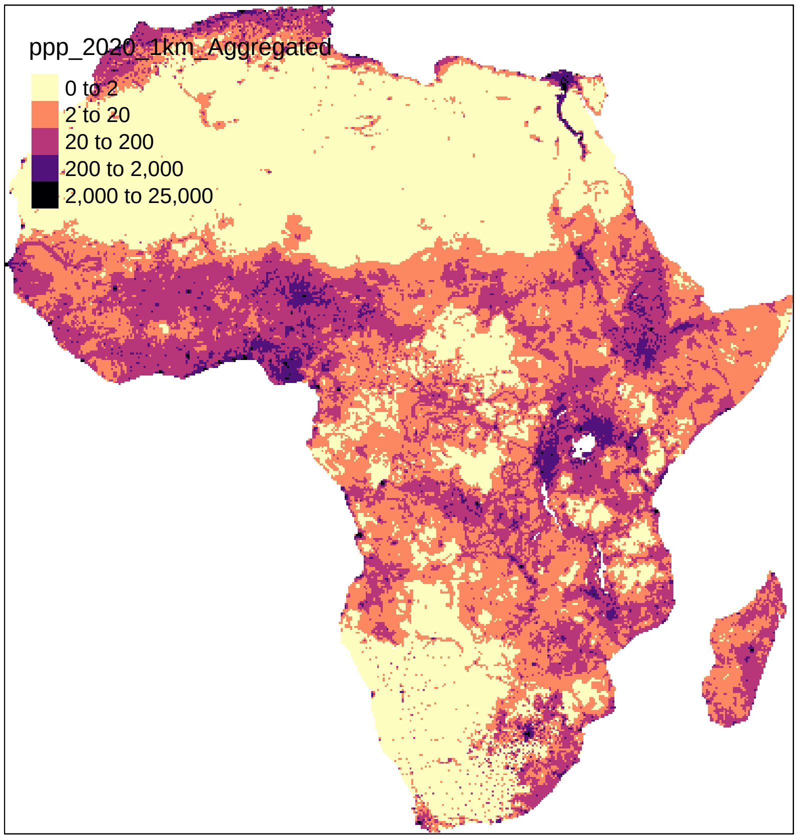
F. Mapping multiple 'layers'

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 some other map packages) use the `+` symbol to combine layers. See the maps below where another layer is added for each map.

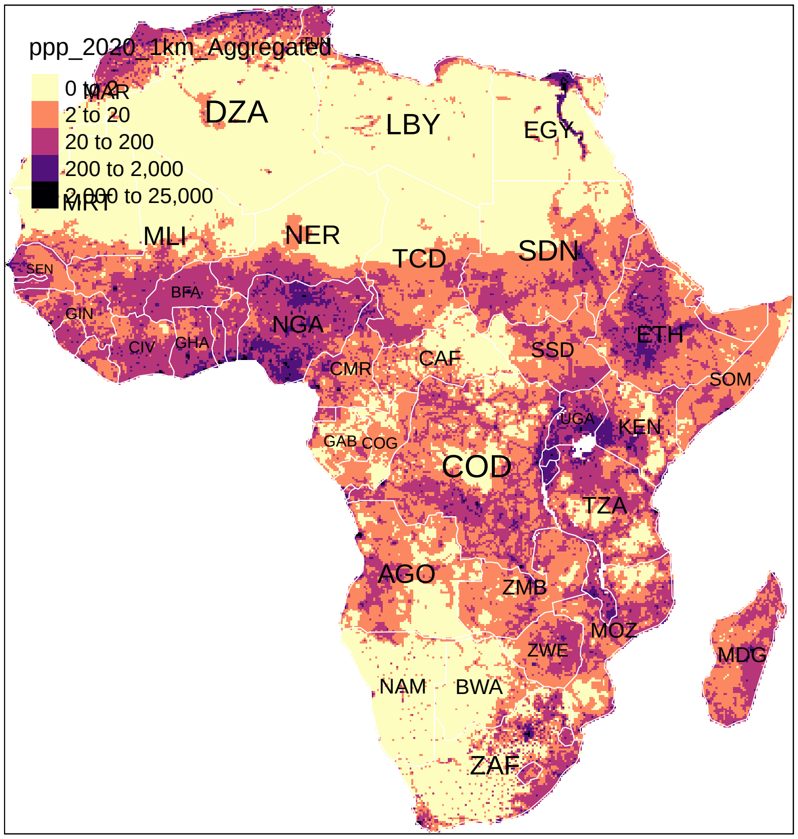
Raster only:

```
# tmap-vector-raster1a

tmap_mode('plot')
tmap::tm_shape(afripop2020) +
  tm_raster(palette = rev(viridisLite::magma(5)), breaks=c(0,2,20,200,2000,25000))
```



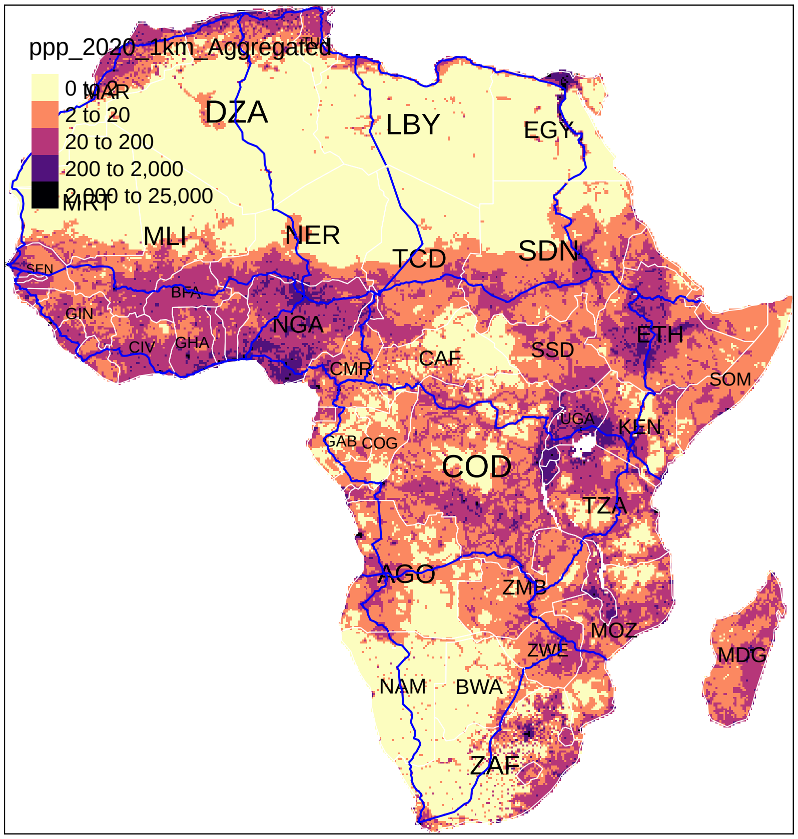
Adding country polygons and text labels:

```
# tmap-vector-raster1b

tmap_mode('plot')
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")
```



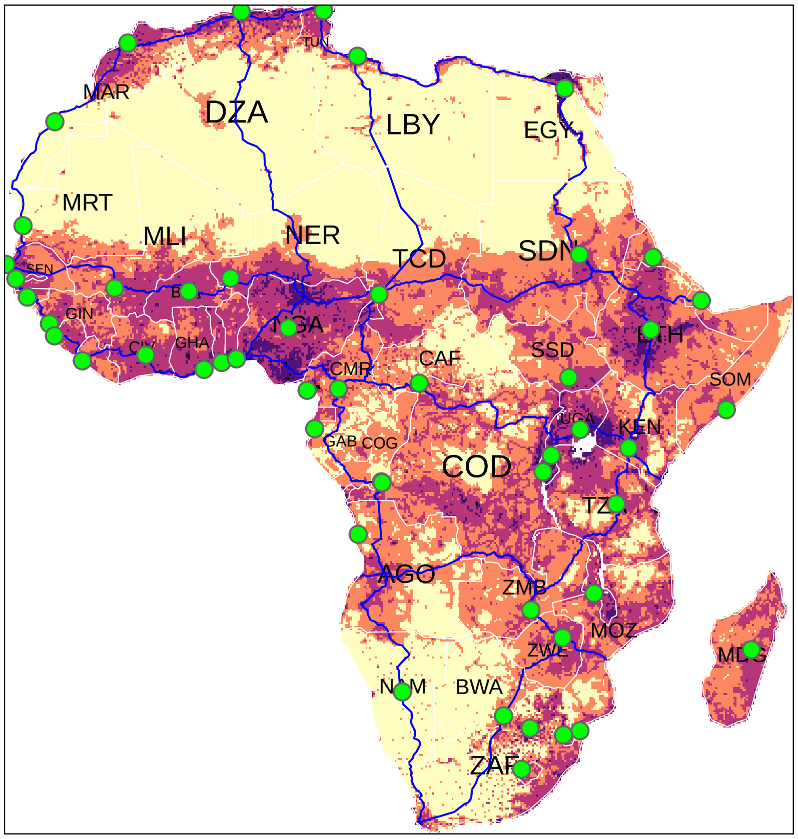
Adding road lines:

```
# tmap-vector-raster1c

tmap_mode('plot')
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 = "blue")
```



Adding capitals points:

```
# tmap-vector-raster1d

tmap_mode('plot')
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 = "blue") +
  tm_shape(africapitals) +
  tm_symbols(col = "green",  scale = .6 ) +
  tm_legend(show = FALSE)
```



Note that the order of the layers in the code matters. It is a bit like a painting. Later layers are painted on top of the earlier layers. This is why we put the raster layer (population) first, otherwise it would cover up the vector layers (country polygons, road lines and capital points).

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 online 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 at the beginning, `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.

Experiment by adding and removing the # sign at the start of lines in the code below. Try to make maps:

1. without the highway network
2. without the raster population layer and with country boundaries that are visible
3. with text labels for ISO country codes

```
# tmap-interactive

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 = "blue") +
  tm_shape(africapitals) +
  tm_symbols(col = "green",  scale = .6 ) +
  tm_legend(show = FALSE)
```

```
## 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.
```

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

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

Summary

Good persistence for getting this far!

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

We've shown you:

1. storing and handling spatial data with the packages `sf` and `raster`
2. making static and interactive maps with package `tmap`

This is a start; there are plenty of other options (e.g. maps can also be made with the packages `mapview` and `ggplot2`).

useR! 2021: Next session

In this session we relied on data that was already saved as R objects rather than reading it in from data files. Our next session in this useR! 2021 tutorial will be an entry level outline to demonstrate getting spatial data of different types into R as a first step to mapping it. The aim is to support you getting your own data into R before making maps or other plots.

BREAK TIME OF 15 MINS