# Session 1: Introduction aux données spatiales

Tutoriel de afrimapr à userR! 2021

07 July 2021

## Contents



Ceci est un tutoriel développé pour userR! 2021 sur la cartographie en R avec des données sur le continent africain. La cible de ce tutoriel est une audience avec une expérience de R et de GIS limitée.

Le tutoriel est basé sur le package afrilearnr qui contient des tutoriels ayant pour objectif d'enseigner des compétences de manipulation de données spatiales avec des données africaines. Il fait partie du projet afrimapr qui est financé par *Wellcome Trust's Open Research Fund and Data for Science and Health.*

Ce tutoriel a été adapté en fonction des besoins de useR! 2021. Pour plus d'informations et de leçons, veuillez consulter les tutoriels en ligne.

Des fichiers PDF des tutoriels ont été inclus dans le dossier du projet.

## A. Les grandes lignes de la session

Ceci est une introduction aux données spatiales qui utilise des exemples sur le continent africain.

#### Résultats d'apprentissage

À la fin de cette session, vous aurez appris comment stocker et manipuler les données spatiales ainsi que comment faire des cartes statiques et interactives. De façon plus spécifique, vous serez à mesure de:

- vous rappeler des fonctions utiles à la cartographie
- comprendre la classification de plusieurs types de données spatiales
- utiliser des packages R pour travailler avec des données vectorielles et maillées (raster)
- créer des cartes statiques et interactives en utilisant le package tmap avec des données disponible dans le package `afrilearndata`
- superposer plusieurs types de données (des strates) sur une carte
- changer la palette des couleurs afin de représenter les données sur une carte

N'hésitez pas à contacter les formateurs et les autres participants durant et/ou après le tutoriel pour leur faire savoir vos difficultés éventuelles.

## B. Charger les packages et les données

Les packages contiennent des fonctions et des données qui élargissent les fonctionalités de la version R de base. Il est utile de voir R comme un téléphone mobile qui vient avec des fonctionalités de base. Ensuite, chaque utilisateur peut y installer des applications (packages) afin d'effectuer des tâches plus spécifiques avec le téléphone.

Nous chargerons un package du nom de `afrilearndata` qui contient des exemples de données.

Nous utiliserons aussi des packages qui nous permettrons de manipuler les données spatiales. Les villes, les autoroutes et les démarcations sont, respectivement, des exemples de données ponctuelles, linéaires et vectorielles. Tandis que les données telles que les densités de population sont appelées les données maillées (raster). Nous en parlerons avec plus de détails dans la section C ci-dessous.

Les packages `sf` et `raster` nous permettent de manipuler les données vectorielles et maillées.

Deux étapes sont requises afin d'utiliser un package en R:

1. `install.packages("[nom_du_package]")` installe le package à partir d'internet et ne doit se faire qu'une seule fois (remplacez `[nom_du_package]` par le nom de votre package)
2. `library([nom_du_package])` est nécessaire toutes les fois que vous ouvrez une nouvelle session R.

Ces deux étapes sont comme installer une application sur votre téléphone mobile (à faire une seule fois à moins que vous ayiez désinstallé l'application) et ouvrir l'application (à faire toute fois que vous voulez utiliser l'application).

Afin de vérifier si les applications ont bel et bien été installées, essayez d'exécuter les commandes de la forme `library([nom_du_package])` ci-dessous. Si elles ont été installées, rien ne devrait se passer. R ne produira un message d'erreur que s'il y a un problème.

Si vous recevez un message vous indiquant qu'un de ces packages n'a pas été installé, alors vous pourrez utiliser `install.packages("[nom_du_package]")` pour l'installer. Pour les besoins de ce tutoriel useR! 2021, les packages ont déjà été installés dans le projet RStudio Cloud. Vous n'aurez donc qu'à exécuter les codes ci-dessous afin de les charger (ouvrir) dans l'environnement de la session courante (voir étape 2 ci-dessus). Cependant, si vous travaillez sur ce tutoriel à partir d'une installation locale de R/RStudio sur votre machine, alors vous aurez tout d'abord à installer les packages si vous ne l'avez pas déjà fait. Un script qui installera tous les packages requis est disponible dans le dossier principal du projet sous le nom de: `packages_and_data.R`.

```
#### SECTION B: CHARGER LES PACKAGES ET LES DONNÉES ----

# pour manipuler les données vectorielles
library(sf)

# pour manipuler les données maillées (raster)
library(raster)

# exemples de données spatiales sur le continent africain
library(afrilearndata)

# pour des cartes statiques et intéractives
library(tmap)

# pour créer des objets de type RasterLayer
library(rgdal)
```

## C. Les données spatiales: qu'est-ce c'est?

- Les villes, les autoroutes et les démarcations sont, respectivement, des exemples de données ponctuelles, linéaires et polygonales toutes connues comme des **données vectorielles**. Les données vectorielles représentent typiquement des objects distincts tels que des pompes à incendie, des routes, des barrages hydroélectriques, des aéroports, ou encore des pays. Les données vectorielles sont généralement sous forme tabulaire avec une colonne contenant les coordonnées et d'autres colonnes contenant des attributs.
- **Les données maillées (raster)** sont des données continues qui n'ont aucune démarcation définie mais qui contiennent plusieurs informations telles que la densité d'une population, la tempèrature, les précipitations ou encore les niveaux d'élevation d'un terrain. C'est une maille contenant des pixels de même taille.

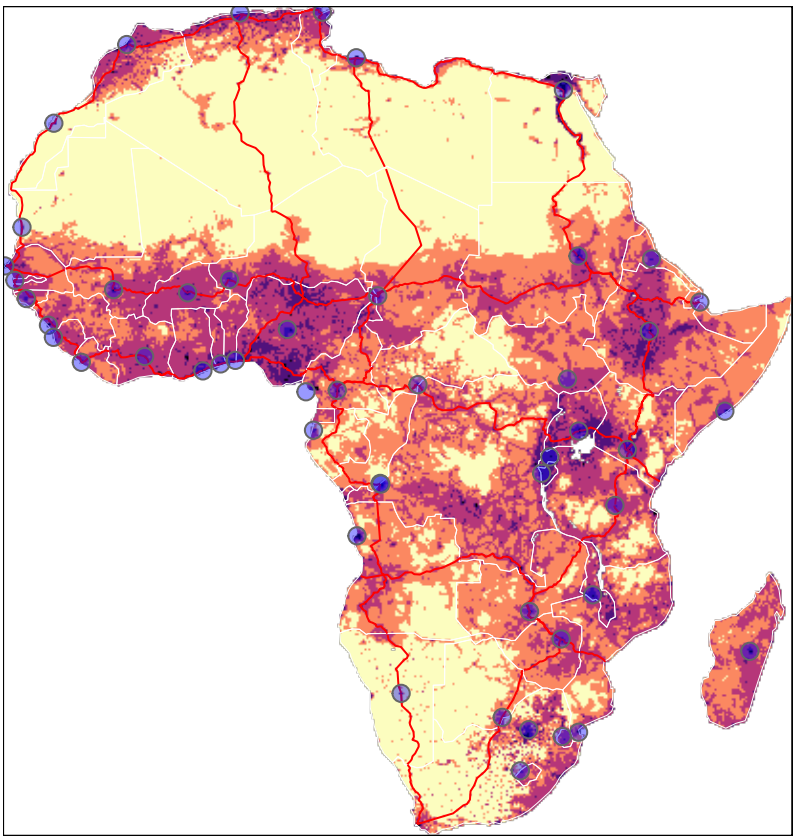
Nous commencerons par regarder ces différents types de données spatiales dans le contexte du continent africain en utilisant le package `afrilearndata` qui fait partie du projet `afrimapr`. Ces données incluent:

1. La localisation des capitales (points) `africapitals`
2. Un réseau d'autoroutes (lignes) `afrihighway`
3. Les frontières/démarcations des pays (polygones) `africountries`
4. La densité des populations (données maillées/raster) `afripop2020`

Tout d'abord, jettons un coup d'oeil à ces données:

```
# Voir les données

#View(afripop2020)
#View(africountries)
#View(afrihighway)
#View(africapitals)
```

À présent, visualisons ces données sur une carte:



Dans R, il y a très souvent le choix! Plusieurs packages peuvent accomplir les même tâches. Lequel de ces packages est le "meilleur" dépend de vos préférences et de votre contexte qui peuvent changer avec le temps. Ceci est vrai pour les opérations spatiales.

Le package `sf` travaille avec les données vectorielles (le points, les lignes et les polygones) et le package `raster` travaille avec les données maillées (raster).

Il y a d'autres packages qui font pareil mais nous n'en aurons pas besoin pour le moment.

## D. Les objects spatiaux

Nous allons à présent regarder de plus près l'objet spatial utilisé pour créer la carte ci-dessus.

Nous les appelons des "objets" parce que les données sont déjà stockées dans R. Celà nous permet aussi de faire la différence d'avec un "fichier" qui est sur votre ordinateur. Lorsqu'un fichier est importé dans R, il devient un "objet". Nous reviendrons là-dessus par la suite.

Dans R il y a plusieurs fonctions qui peuvent nous permettre d'explorer le contenu d'un objet. Plusieurs de ces fonctions ont des fonctionalités communes, cependant les fonctions suivantes sont d'une utilité particulière:

1. `str()` montre la structure d'un objet, affiche les noms et les valeurs
2. `head()` affiche les premières lignes des données avec les noms des colonnes
3. `names()` affiche uniquement que les noms des colonnes
4. `class()` montre la classe de l'objet, c'est-à-dire le type d'objet que c'est

Regardons les résultats de ces fonctions lorsqu'elles sont appliquées à `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"
```

Nous voyons que `africapitals` est de classe `sf` et `data.frame` et contient une série de colonnes sont certaines sont nommées: `capitalname`, `countryname` et `geometry`.

La classe `data.frame` (ou encore `dataframe`) est le type d'objet que les nouveaux utilisateurs de R rencontrent le plus souvent. Les dataframes stockent les données de façon rectangulaire avec des lignes et des colonnes nommées comme des feuilles de calcul.

Les objets `sf` sont un type spécial de dataframe avec une colonne appelée `geometry` qui contient des informations spatiales et dans lesquels chaque ligne représente une entité spatiale. Dans ce cas, les entités spatiales sont des points.

Si vous regardez le résultat de la fonction `str()` ci-dessus, vous remarquerez que la première valeur de la colonne `geometry` a les coordonnées 7,17 9,18. Parce que les données sur les capitales sont des points, elles ont chacune une paire de coordonnées qui représentent leur longitudes et la latitudes respectives.

## E. Nos premières cartes avec le package `tmap`

Il existe plusieurs packages qui ajoutent aux fonctionnalités de `sf` pour faire des cartes.

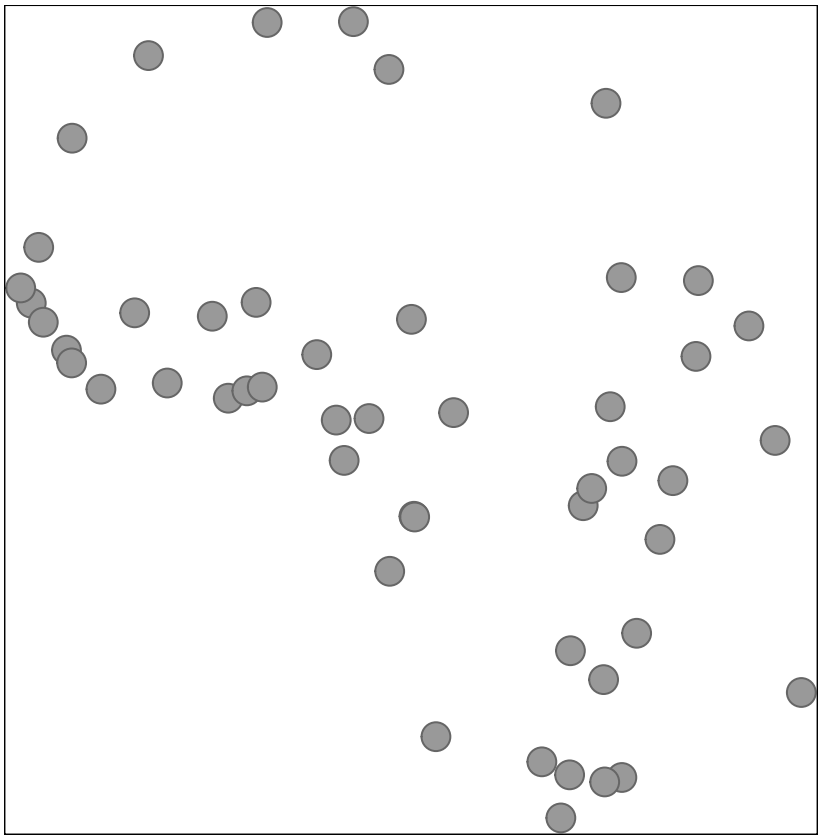
Le package `tmap` est un bon départ car il offre la possibilité de créer des cartes statiques et intéractives.

**Données vectorielles: les points**

Commençons avec des cartes statiques montrant les capitales (en tant que points)

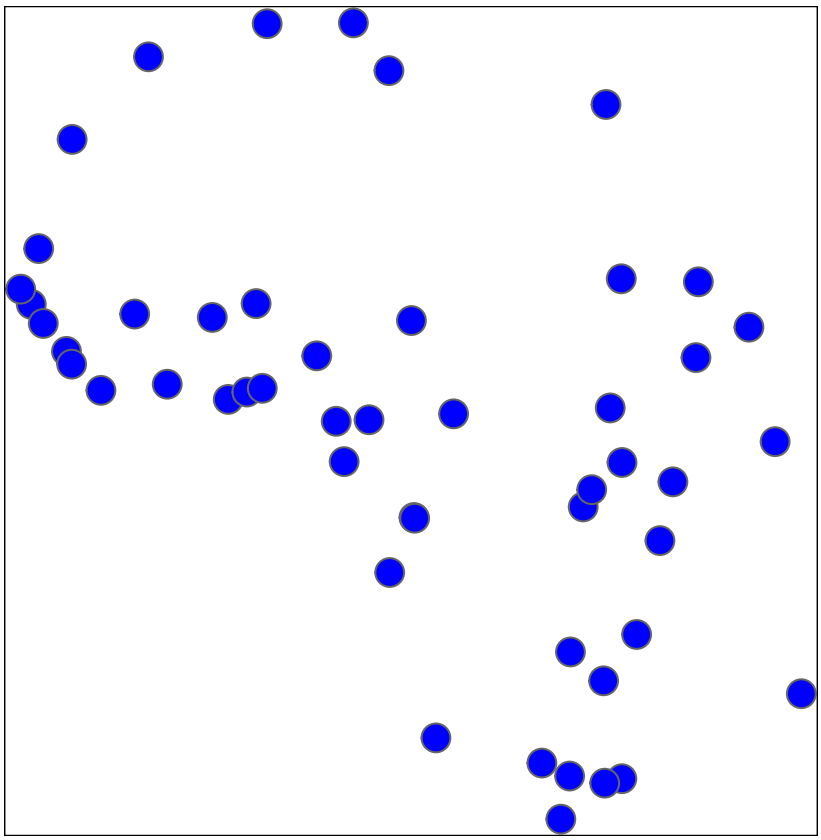
Dans le package `tmap`, la fonction `tm_shape([nom_objet])` définit l'objet à utiliser. Ensuite, le symbole `+` est utilisé pour ajouter du code pour définir comment les données sont présentées. Par exemple, la fonction `tm_symbols()` présente les données en tant que points. Des arguments additionnels peuvent être spécifiés afin de modifier la présentation des données.

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



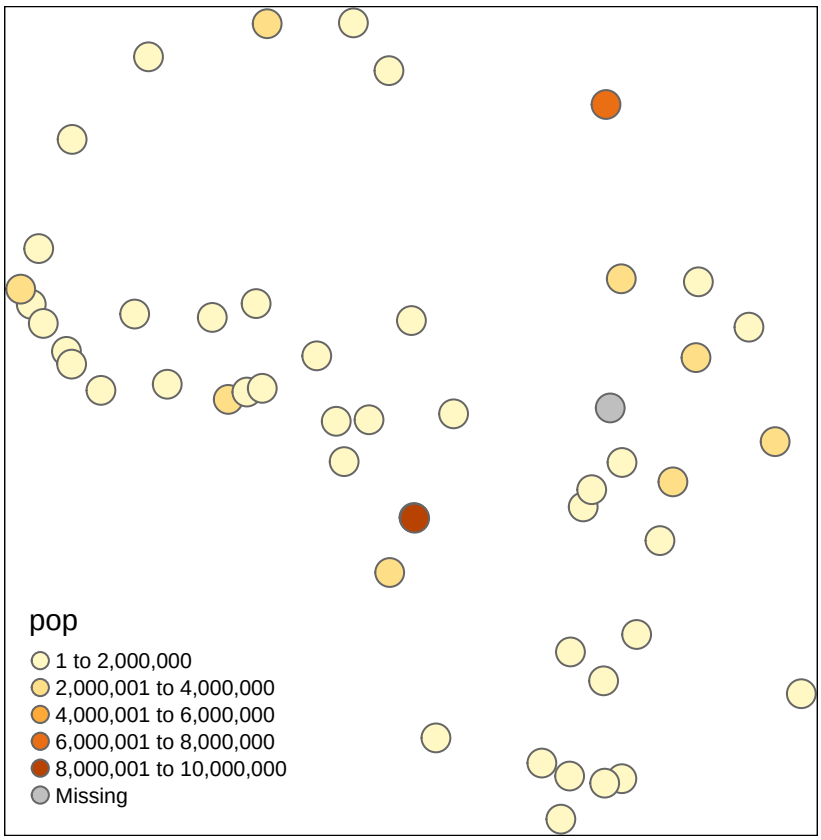
Afin de pouvoir la couleur des points en bleu, on peut ajouter l'argument `col` comme suit: `tm_symbols(col = "blue")`. Essayez d'autres noms de couleurs (notez que les noms des couleurs doivent être en anglais). Vous trouverez que la majorité des noms de couleurs que vous utiliserez seront reconnus par R (si vous exécutez la fonction `colours()`, vous obtiendrez une liste de plus de 600 noms de couleurs que R reconnait).

```
#tmap-points1b
tmap_mode('plot')
tm_shape(africapitals) +
   tm_symbols(col = "blue")
```



Dans la carte ci-dessus, tous les points ont été générés avec la même couleur. Il est aussi possible de faire de sorte que la couleur de chaque point (chaque ligne) dépende de la valeur correspondante d'une colonne du dataframe. Pour celà, on peut ajouter l'argument `col = [nom_colonne]`.

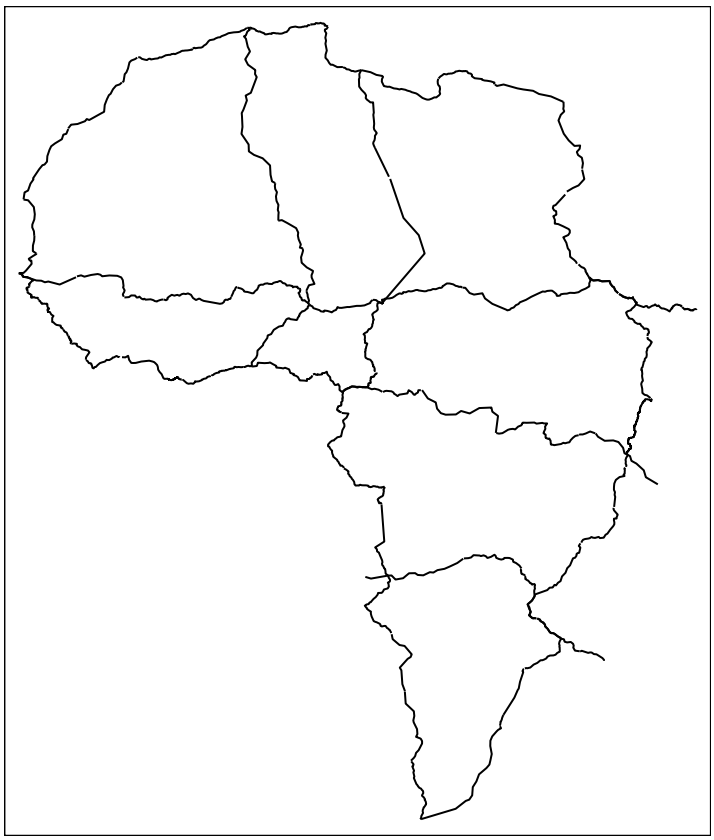
```
# tmap-points1c
tmap_mode('plot')
tm_shape(africapitals) +
  tm_symbols(col = "pop")
```




**Données vectorielles: les lignes**

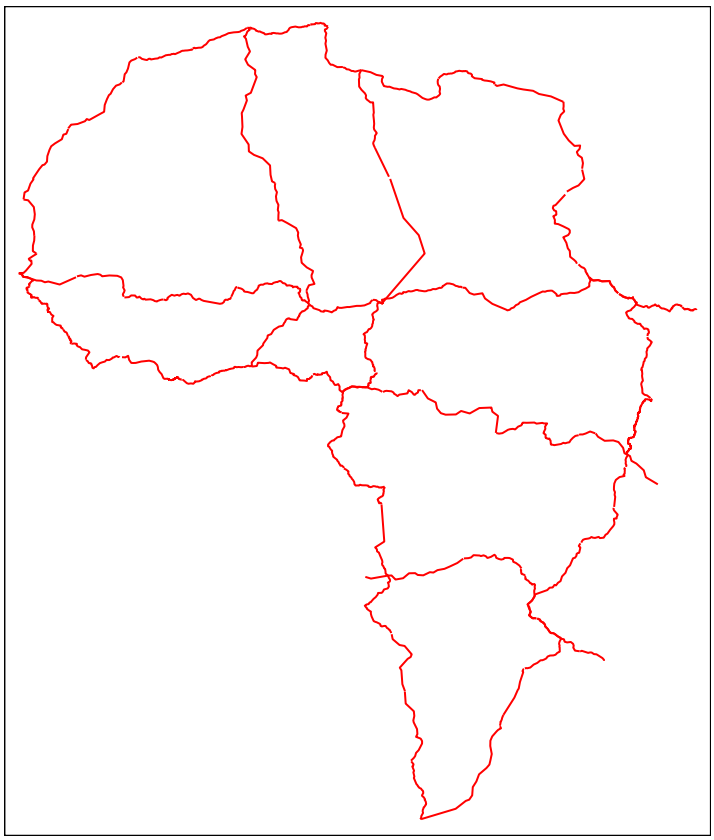
Le réseau d'autoroutes (les lignes) peuvent être visualisées en utilisant la même commande `tm_shape([nom_objet])` au début et en ajoutant, ensuite, la commande `tm_lines()` afin d'afficher les lignes. Veuillez regarder les exemples ci-dessous pour voir comment appliquer des couleurs aux lignes.

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



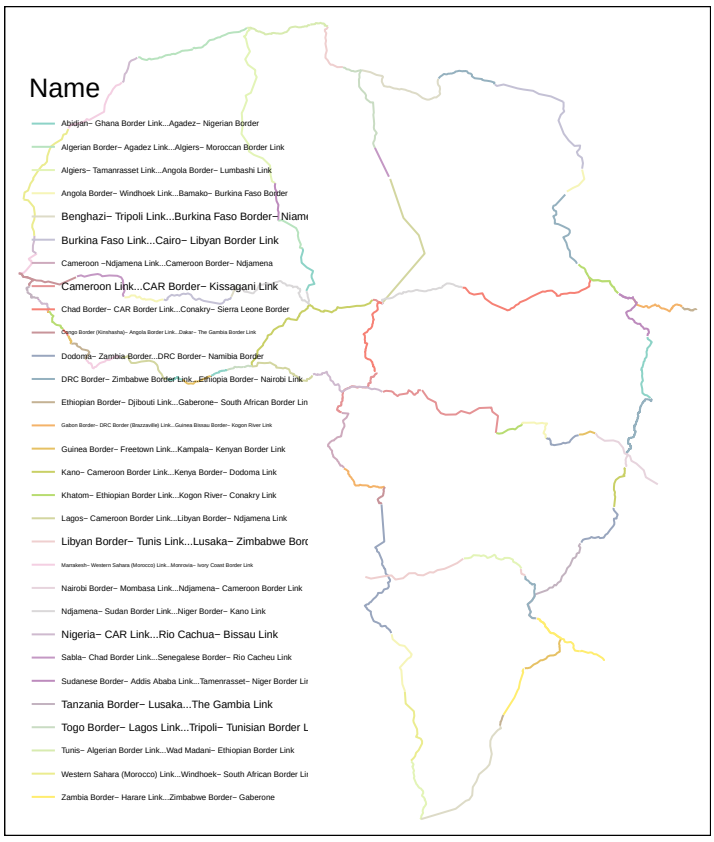
Une couleur pour toutes les lignes:

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



Les couleurs des lignes dépendent des valeurs spécifiques d'une colonne dans l'objet dataframe:

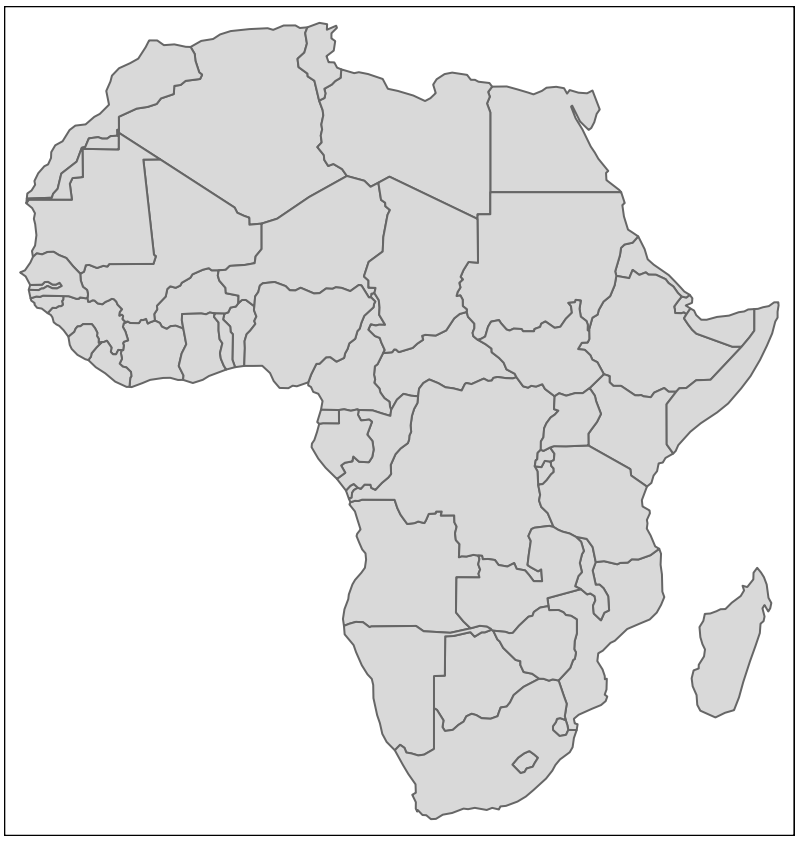
```
# tmap-lines1c
tmap_mode('plot')
tm_shape(afrihighway) +
   tm_lines(col = "Name")  # use a column name from the object
```



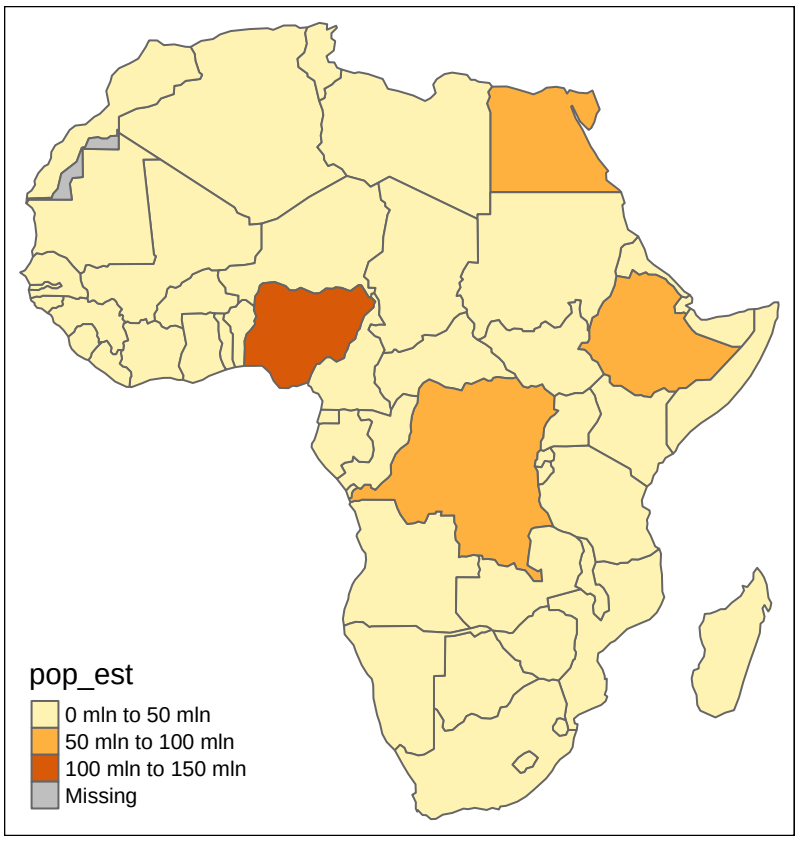

### Données vectorielles: les polygones

Les démarcations des pays (les polygones) peuvent aussi être cartographiés en utilisant les fonctions `tm_shape()` et `tm_polygons()`. Encore une fois, veuillez consulter les exemples ci-dessous afin de voir comment appliquer des couleurs aux différents pays:

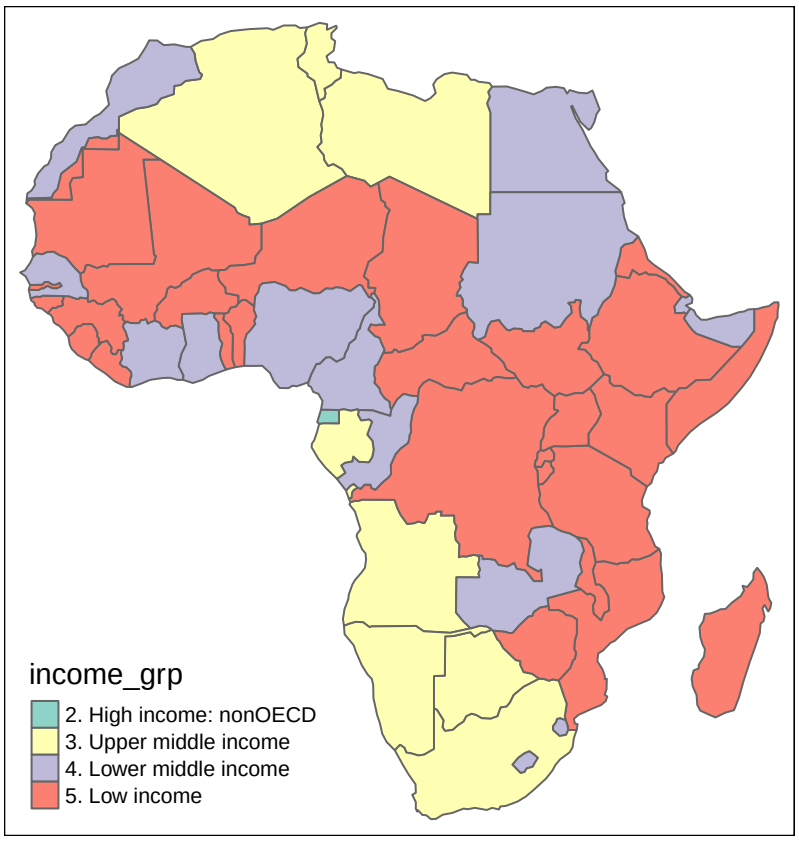
```
# tmap-polygons-1a
tmap_mode('plot')
tm_shape(africountries) +
     tm_polygons()
```



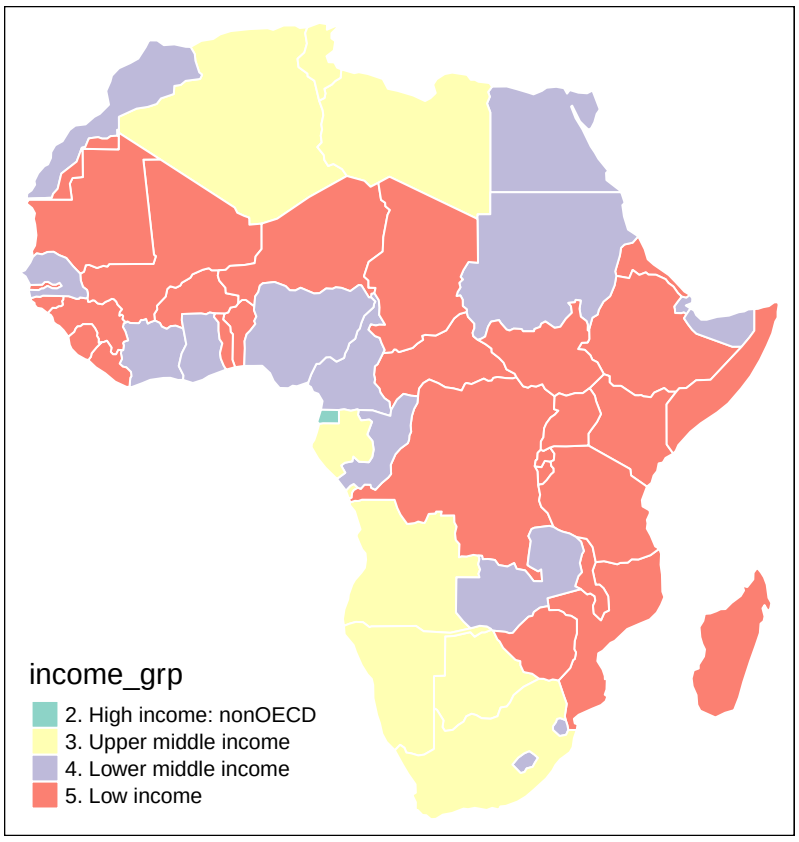
```
# tmap-polygons-1b
tmap_mode('plot')
tm_shape(africountries) +
      tm_polygons(col="pop_est")
```




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




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



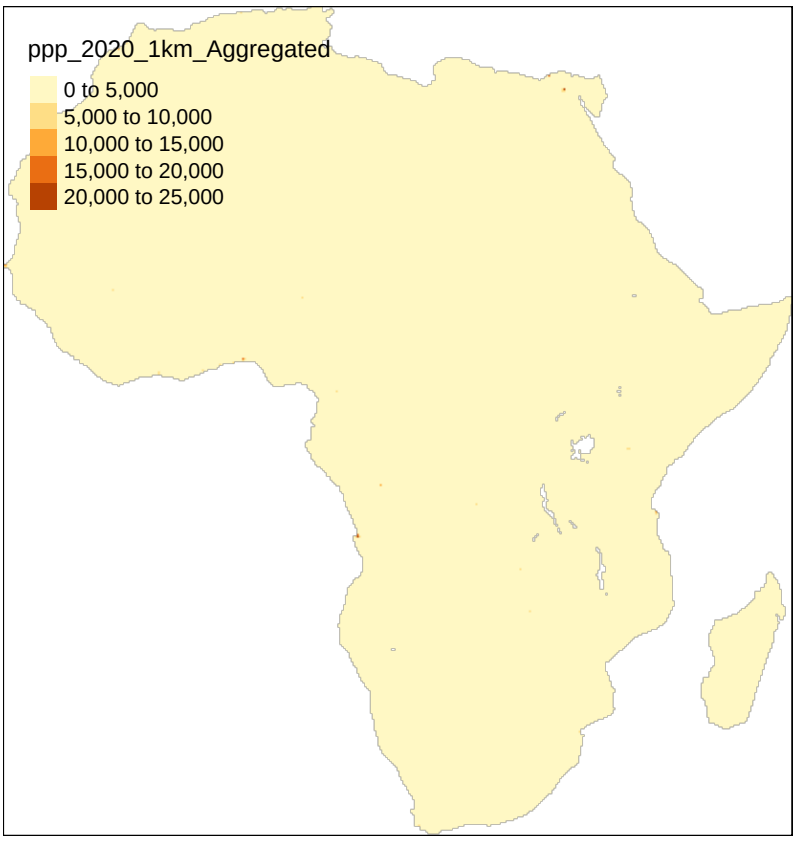

**Données maillées (raster)**

Les données maillées (raster) peuvent représenter des données collectées par télédetection ou des données modélisées.

Elles peuvent être visualisées avec les fonctions `tm_shape([nom_objet])` and `tm_raster()`.

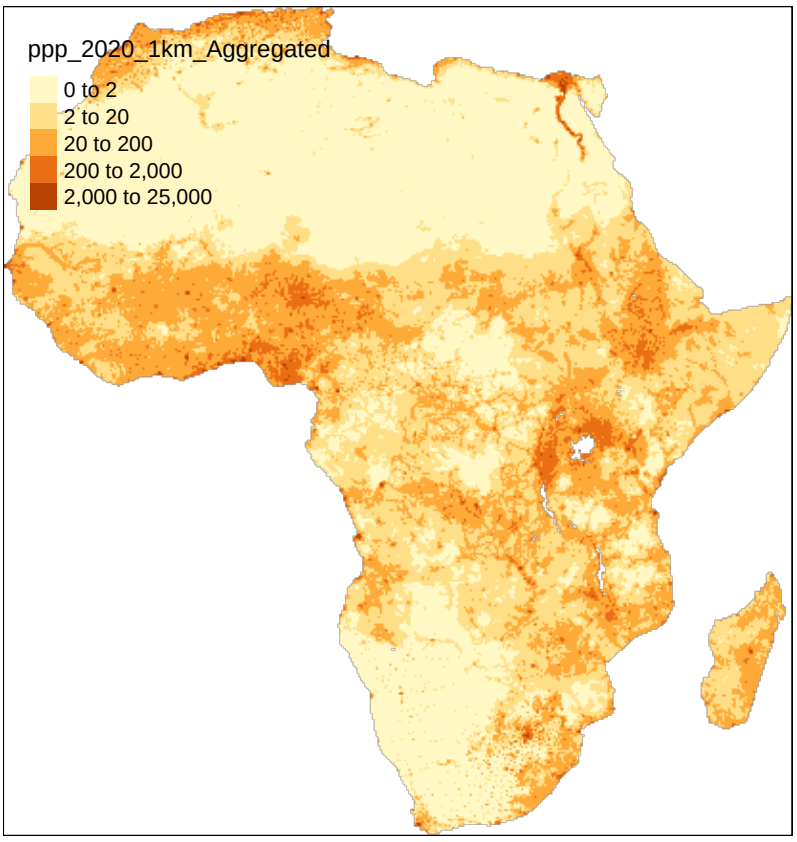
Dans l'exemple ci-dessous, si vous utilisez les intervalles par défaut en ne spécifiant aucun argument dans `tm_raster()`, alors la carte semblera ne contenir qu'une seule couleur. Ceci est dû au fait qu'il y ait très peu de cellules à très forte densité et que la très grande majorité des cellules sont de faible densité. Ceci est un problème recurrent avec les données sur les populations. Les classifications par défaut (qui sont à intervalles égaux) ne sont pas appropriées puisqu'une très grande partie de la carte appartient à la catégorie des valeurs faibles. Avec des efforts, il est possible de voir des cellules de haute densité (Lagos et le Caire).

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



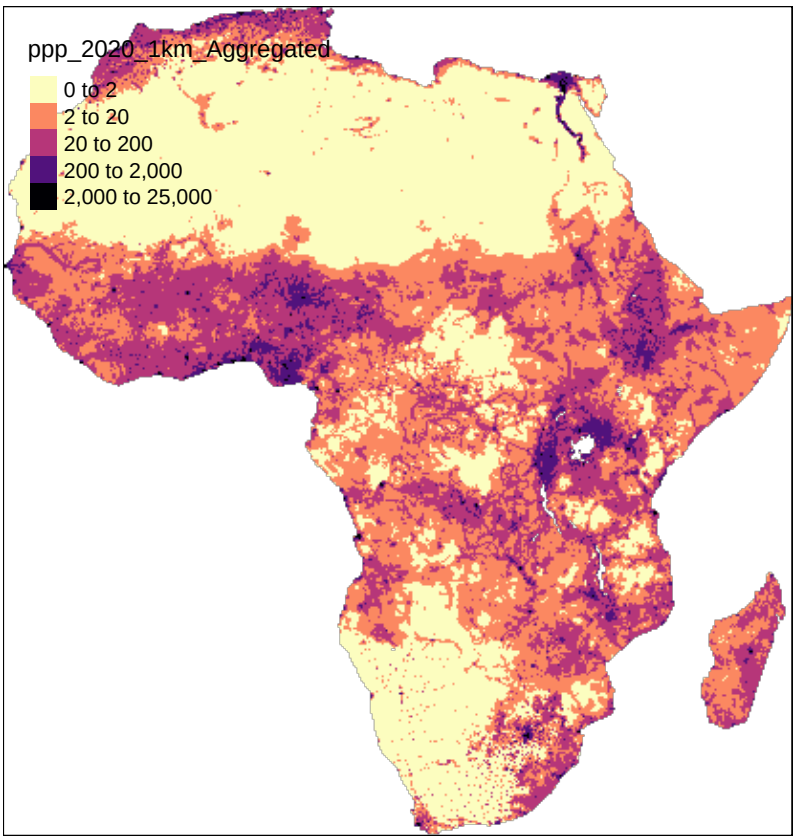

Il est possible de spécifier les intervalles pour plusieurs couleurs afin de montrer de façon adéquate les différences entre les valeurs faibles. Vous pouvez vous essayer à le faire en changeant les valeurs de l'argument `breaks =`.

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




En plus de changer les intervalles, il est aussi possible de changer la palette des couleurs utilisées avec l'argument `palette =`. Dans l'exemple ci-dessous, nous utilisons la fonction `rev()` afin de renverser l'ordre des couleurs. L'objectif de cette manipulation est d'utiliser les couleurs les plus sombres pour représenter les populations les plus fortes. Essayez de retirer `rev()` du code afin de voir le résultat.

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



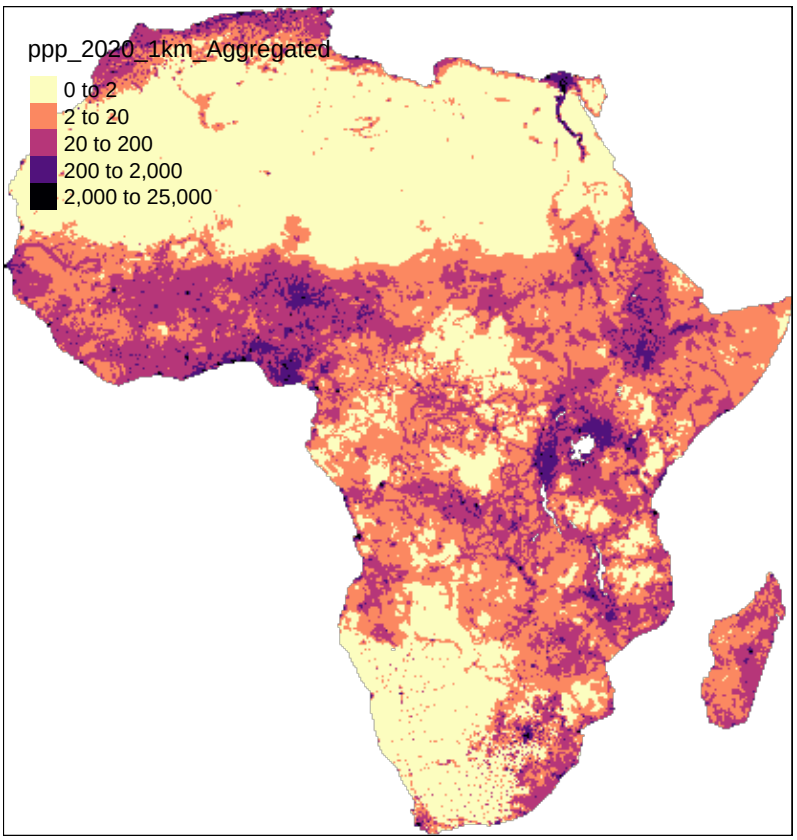

## F. Cartographier plusieurs strates (layers)

Dans la section précédente, nous vous avons appris comment faire des cartes avec des objets individuels contenant des données. Les codes que nous avons utilisé peuvent être combinés afin de créer des cartes avec plusieurs strates (layers) comme vous pourrez le voir dans les exemples qui suivent.

Le package `tmap` (ainsi que plusieurs autres packages de cartographie) utilisent le symbole `+` afin de combiner les strates. Dans les cartes ci-dessous, une nouvelle strate est ajoutée.

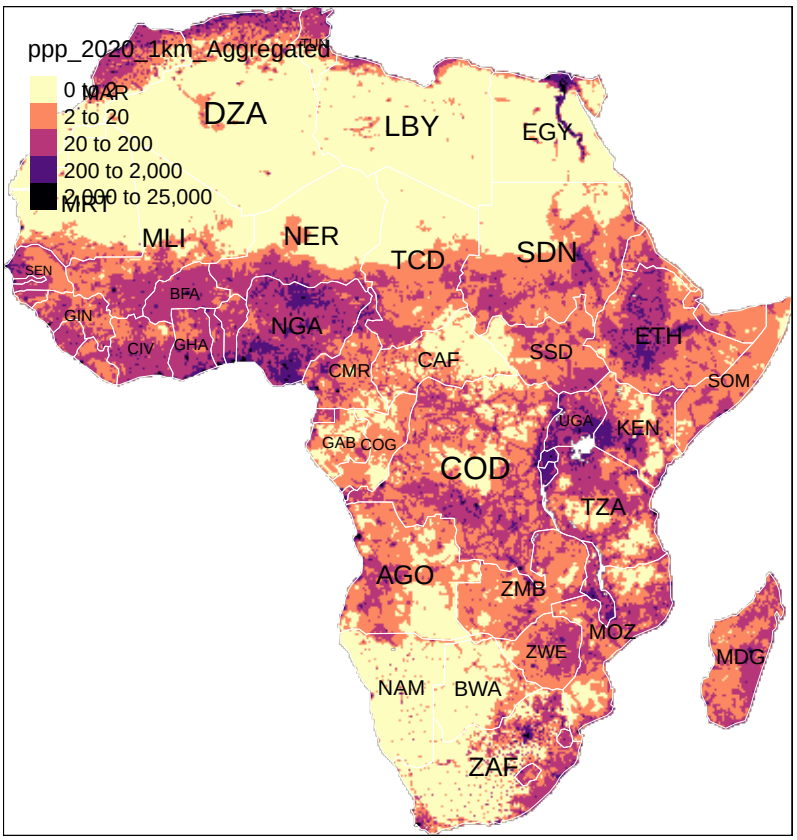
### Données maillée (raster) uniquement

```
# 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))
```



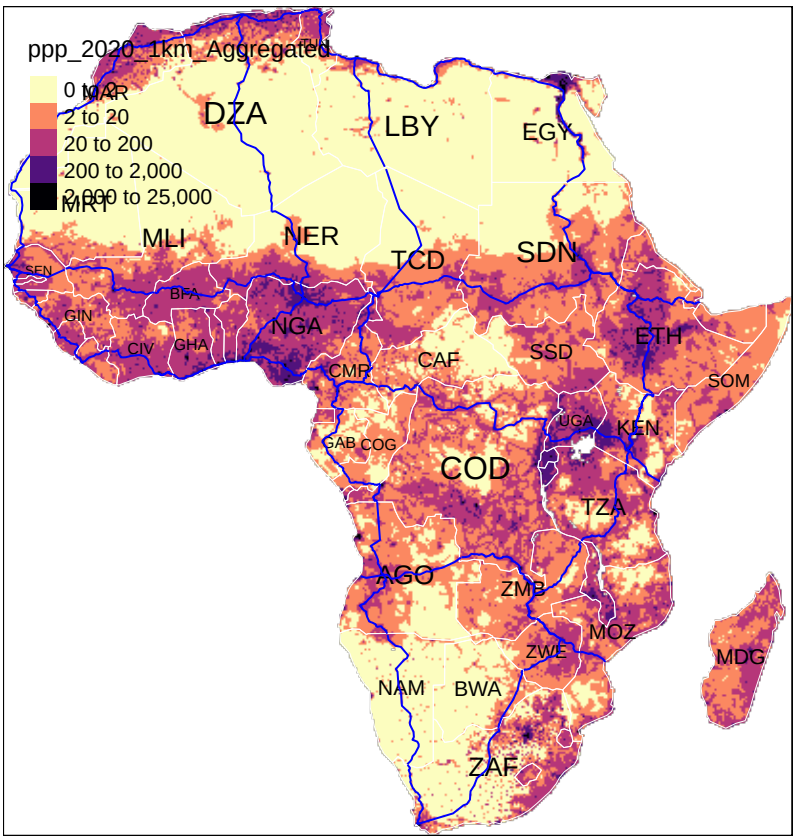

**Ajout des données polygonales des pays et des noms abbrégés (données textuelles)**

```
# 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")
```



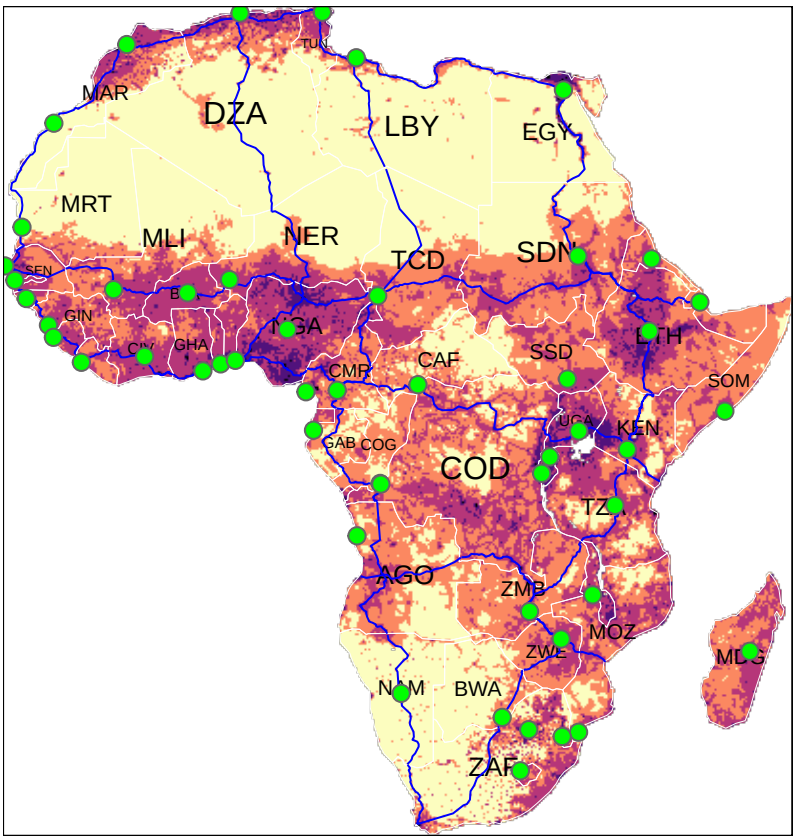

**Ajout des données linéaires des routes**

```
# 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")
```




**Ajout des données ponctuelles des capitales**

```
# 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)
```




L'ordre d'ajout des strates est important. C'est un peu comme faire de la peinture. Les strates ajoutées à la fin sont au-dessus des strates ajoutées au début. C'est pourquoi nous avons placé les données maillées (raster) d'abord, sinon elles couvriraient toutes les autres strates (les polygones des pays, les lignes des routes et les points des capitales).

## G. Les cartes interactives

Toutes les cartes que nous avons créées jusqu'ici ont été statiques. Nous avons aussi à notre disposition de très puissantes options pour créer des cartes interactives qui sont utiles pour des pages web ou des rapports en ligne. Ces cartes donneront, par exemple, la possibilité aux lecteurs de zoomer, panoramiquer, activer et désactiver les strates.

Dans `tmap`, nous pouvons garder tous les codes que nous avons utilisé jusqu'ici et juste ajouter le code suivant au début: `tmap_mode('view')`. Ceci permet de passer en mode interactif "view". Le mode interactif "view" restera actif pendant le reste de la session R et vous pourrez basculer au mode statique "plot" en utilisant le code `tmap_mode('plot')`.

Le code qui suit est identique au code utilisé dans la section précédente sauf qu'il utilise le mode "view".

Exécutes le code en ajoutant et retirant le symbole `#` au début de chaque ligne. Essaie de créer des cartes différentes avec les spécifications suivantes:

1. sans le réseau routier
2. sans la maille des populations mais avec les démarcations des pays
3. avec les codes ISO des noms des pays

```
# 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.
```

Vous pourrez retourner aux cartes que nous avons créées plus tôt afin de les rendre intéractives en ajoutant `tmap_mode('view')` au début des codes.

**Il est à noter que les cartes intéractives n'apparaitront pas dans des fichiers PDF créés avec `knit`.**

## En résumué

Félicitations d'être arrivé aussi loin!

Nous espérons que vous avez aimé cette brève introduction à la cartographie dans R.

Nous vous avons appris à:

1. stocker et manipuler les données spatiales avec les packages `sf` et `raster`
2. créer des cartes statiques et interactives avec le package `tmap`

Ceci n'est que le début, il existe beaucoup plus d'alternatives. Par exemple, nous pouvons aussi utiliser les packages `mapview` et `ggplot2` pour créer des cartes.

### useR! 2021: Session suivante

Dans cette session, nous n'avons pas eu à importer des données spatiales qui se trouvaient dans des fichiers car les données que nous avons utilisées étaient déjà sauvegardées dans des objets R. Dans la session suivante de ce tutoriel, nous démontrerons comment importer des données spatiales de plusieurs types dans R avant de pouvoir les utiliser pour créer des cartes. L'objectif est de vous aider à utiliser vos propres données pour faire de la cartographie et autres visualisations dans R.

## PAUSE DE 15 MINS