Importing data and performing spatial analysis with R

Introduction

This tutorial is designed to provide participants with an simple introduction to techniques for handling, analysing and visualising spatial data in R.

Learning outcomes

By the end of this course, participants will be able to:

- Import data into R
- Perform several types of spatial analyses in R
- Plot and export figure/maps

R and RStudio

If you plan to follow along with the R coding during the workshop, please ensure that you have the latest versions of R and RStudio installed on your computer.

First, you will need to download and install from https://cran.r-project.org.

Next you will need to download and install RStudio from https://rstudio.com/products/rstudio/download/#download.

Setting the working directory

We map our working directory to the `ruminant-feed-balance` folder we created earlier. We assign the folder the variable name `root`.

For Linux/Unix systems

```
# linux systems
root <- "/home/AU_IBAR/ruminant-feed-balance"
```

For Windows system

```
# for windows systems
root <- "c:/Documents/AU_IBAR/ruminant-feed-balance"
```

Install R packages

```
install.packages("sf")
install.packages("ggplot2")
install.packages("terra")
```

Reading vector data

We will import the administrative boundaries data we downloaded in the previous section stored the data in `AdminBound` folder. We use `st_read()`, which simply takes the path of the directory with the shapefile as argument.

```
library(sf)

indir <- paste0(root, "/src/1Data-download/SpatialData/inputs/AdminBound")

aoi1 <- read_sf(paste0(indir, "/gadm40_BFA_1.shp"))
```

Take a look at what we've got

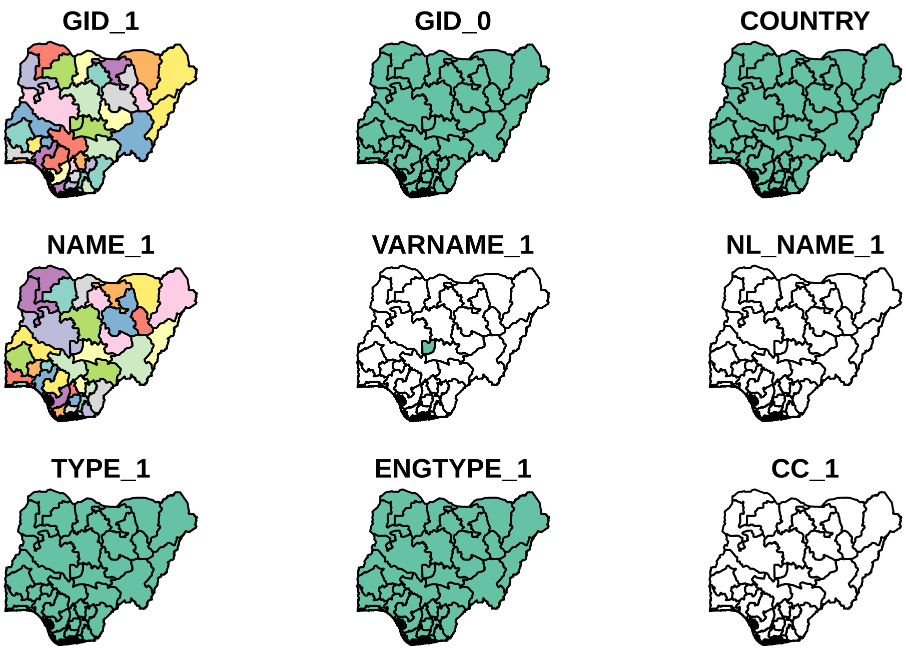
```
str(aoi1)  # note again the geometry column
```

```
sf [37 x 12] (S3: sf/tbl_df/tbl/data.frame)
 $ GID_1    : chr [1:37] "NGA.1_1" "NGA.2_1" "NGA.3_1" "NGA.4_1" ...
 $ GID_0    : chr [1:37] "NGA" "NGA" "NGA" "NGA" ...
 $ COUNTRY  : chr [1:37] "Nigeria" "Nigeria" "Nigeria" "Nigeria" ...
 $ NAME_1   : chr [1:37] "Abia" "Adamawa" "Akwa Ibom" "Anambra" ...
 $ VARNAME_1: chr [1:37] NA NA NA NA ...
 $ NL_NAME_1: chr [1:37] NA NA NA NA ...
 $ TYPE_1   : chr [1:37] "State" "State" "State" "State" ...
 $ ENGTYPE_1: chr [1:37] "State" "State" "State" "State" ...
 $ CC_1     : chr [1:37] NA NA NA NA ...
 $ HASC_1   : chr [1:37] "NG.AB" "NG.AD" "NG.AK" "NG.AN" ...
 $ ISO_1    : chr [1:37] "NG-AB" "NG-AD" "NG-AK" "NG-AN" ...
 $ geometry :sfc_MULTIPOLYGON of length 37; first list element: List of 1
  ..$ :List of 1
```

```
  .. ..$ : num [1:302, 1:2] 7.46 7.46 7.47 7.48 7.48 ...
  ..- attr(*, "class")= chr [1:3] "XY" "MULTIPOLYGON" "sfg"
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA N
  ..- attr(*, "names")= chr [1:11] "GID_1" "GID_0" "COUNTRY" "NAME_1" ...
```

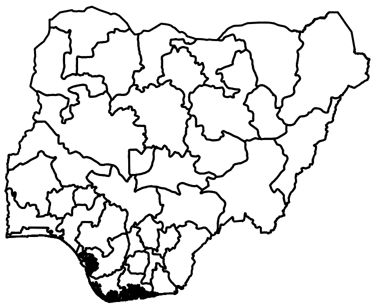
The default `plot` method for an sf object generates a multi-plot displaying the first few attributes. If there are more attributes than can be plotted, a warning is issued.

```
plot(aoi1)
```



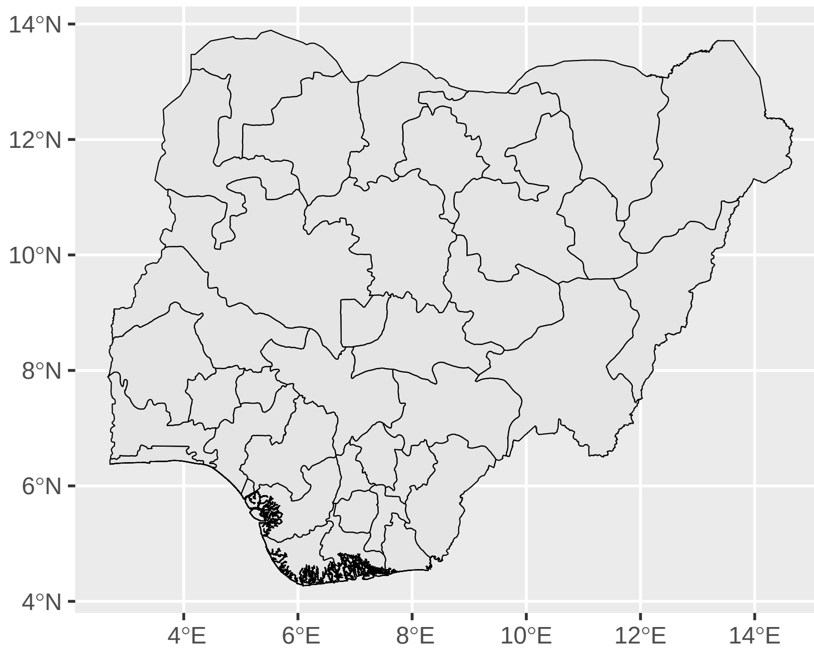
In order to only plot the polygon boundaries we need to directly use the geometry column. We use the `st_geometry()` function. It extracts the `sfc` object (the simple feature geometry list column):

```
plot(st_geometry(aoi1))
```



However, we can use the ggplot2 R package to make a better plot

```
library(ggplot2)
ggplot() + geom_sf(data = aoi1, colour = "black", show.legend = F) +
    coord_sf(xlim = c(2.1, 15.1), ylim = c(3.8, 14.3), expand = FALSE)
```



Reading Raster data

Raster files have a more compact structure than vectors. Since their grid is regular, coordinates for each pixel or cell don't need to be recorded individually. A raster is defined by:

- A Coordinate Reference System (CRS)
- Coordinates of its origin
- Cell size or distance in each direction

- Dimensions (number of cells) in each direction
- An array of cell values

In this tutorial, we will use the `terra` R package that has functions for creating, reading, manipulating, and writing raster data.

We use the `rast()` function to create these objects. For example, to create a raster object from scratch we would do the following:

```
library(terra)
r <- rast(nrows=20, ncols=20, # number of cells in x and y dimension
          xmin=0, xmax=360) # min and max x coordinates (left-right borders)
r
```

```
class       : SpatRaster
dimensions  : 20, 20, 1  (nrow, ncol, nlyr)
resolution  : 18, 9  (x, y)
extent      : 0, 360, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
```

Some useful functions to look at individual properties of the raster object. For examle for the number of cells:

```
ncell(r)
```

```
[1] 400
```

To retrieve number of bands, we use `nlyr()` function.

```
nlyr(r)
```

```
[1] 1
```

To find out about the Coordinate Reference System (CRS), use the `crs` function

```
crs(r, proj = TRUE)
```

```
[1] "+proj=longlat +datum=WGS84 +no_defs"
```

To plot

```r
plot(r)
```

However, it's empty! because the cells do not have any values. To add some random values to the cells we can take advantage of the `ncells()` function and do this:

```r
values(r) <- runif(ncell(r))
r
```

```
class       : SpatRaster
dimensions  : 20, 20, 1  (nrow, ncol, nlyr)
resolution  : 18, 9  (x, y)
extent      : 0, 360, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
source(s)   : memory
name        :       lyr.1
min value   : 0.001094768
max value   : 0.998038380
```

And now we have a plot.

```r
plot(r)
```