How to get your own spatial data into R

A. Outline of this tutorial

This entry level tutorial will demonstrate getting spatial data of different types into R. The aim is to support you getting your own data into R before making maps or other plots.

Outline of afrilearnr

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

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 you can also install afrilearnr from github (https://github.com/afrimapr/afrilearnr) and run these tutorials locally.

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
`get-my-data-in`	getting your own spatial data into R	2 this one
`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 selection section headings on the left. If you want to return the tutorial to its original state you can press 'Start Over' on the lower left.

To repeat these steps locally you would need the following packages.

```
library(readr)            # reading text files
library(sf)               # working with vector data
library(mapview)          # interactive mapping
library(raster)           # raster manipulation
library(afrilearndata)    # example data, not required if using only your own data
```

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

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` space delimited or various spreadsheet formats including `.xls`.

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

1. read the data file into an R dataframe
2. convert the dataframe into an R spatial (package `sf`) object
3. plot the `sf` object

Here we will demonstrate the 3 steps using some airport data from the excellent ourairports (https://ourairports.com/continents/AF/airports.csv) that we have extracted and saved in the afrilearndata (https://github.com/afrimapr/afrilearndata) package.

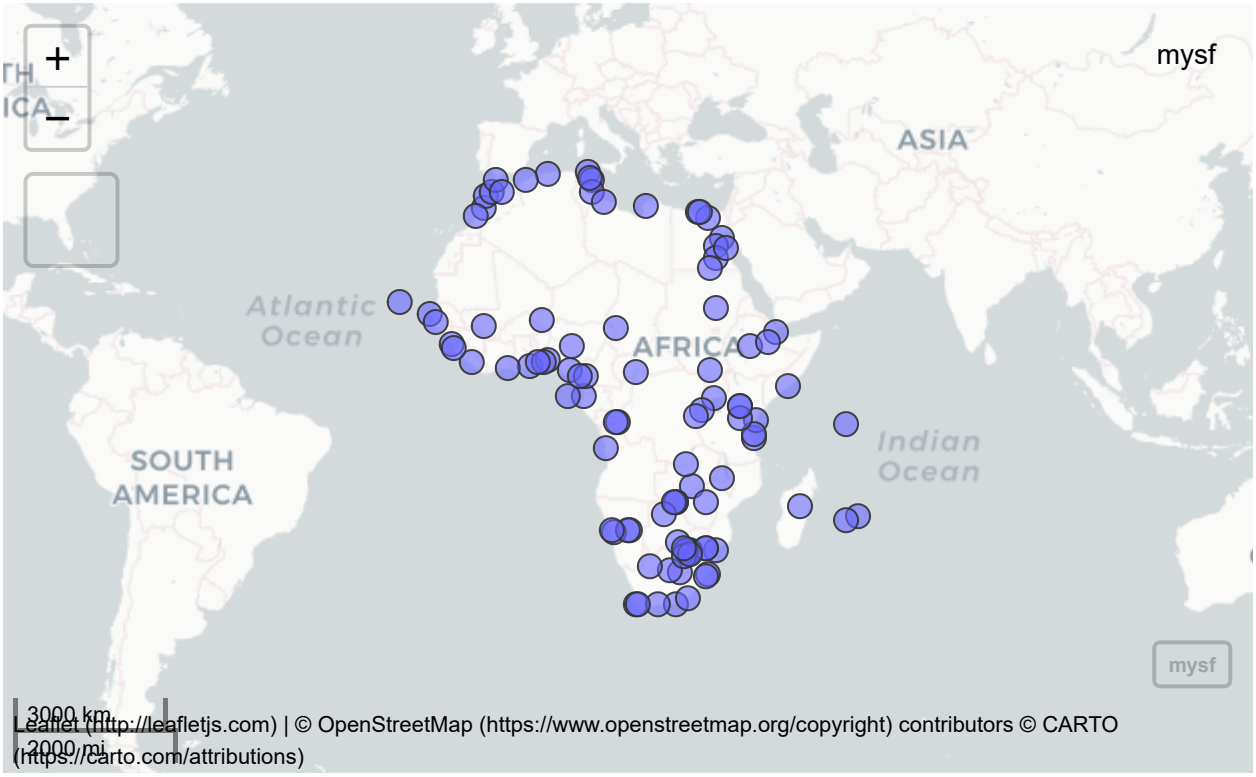
R Code | Start Over | Run Code

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

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

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

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



To apply the code chunk above to your own data :

- 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 coordinates in your data
- you may need to change `crs=4326` as explained below

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 code for longitude, latitude using the WGS84 datum, but you don't really need to know that. 4326 is a good number to remember !

Question : What happens when the `crs=4326` argument is not included in the code below ? Try adding it back in and see what the difference is.

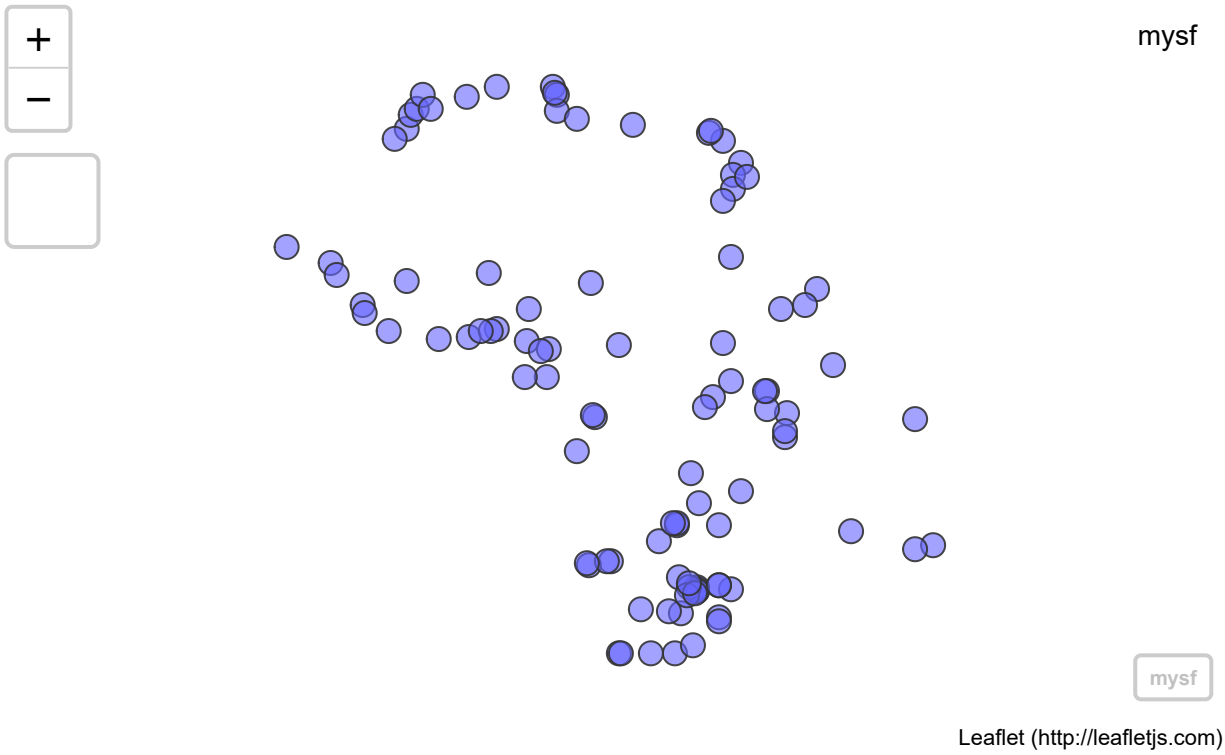
R Code | Start Over | Run Code

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

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

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

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



Answer : You should see that 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.

data structure

We can look at the structure of the original dataframe and the sf object using `names()` to show the column names and `head()` which returns the first six rows. What is the difference between them ?

```
# original dataframe
names(mydf)
```

```
##  [1] "id"              "ident"             "type"
##  [4] "name"            "latitude_deg"      "longitude_deg"
##  [7] "elevation_ft"    "continent"         "country_name"
## [10] "iso_country"     "region_name"       "iso_region"
## [13] "local_region"    "municipality"      "scheduled_service"
## [16] "gps_code"        "iata_code"         "local_code"
## [19] "home_link"       "wikipedia_link"    "keywords"
## [22] "score"           "last_updated"
```

```
# sf object
names(mysf)
```

```
##  [1] "id"                "ident"             "type"
##  [4] "name"              "elevation_ft"      "continent"
##  [7] "country_name"      "iso_country"       "region_name"
## [10] "iso_region"        "local_region"      "municipality"
## [13] "scheduled_service" "gps_code"          "iata_code"
## [16] "local_code"        "home_link"         "wikipedia_link"
## [19] "keywords"          "score"             "last_updated"
## [22] "geometry"
```

```
# original dataframe
head(mydf)
```

id <dbl>	ident <chr>	type <chr>	name <chr>	latitude_deg <dbl>
31055	FAOR	large_airport	OR Tambo International Airport	-26.13920
3183	HECA	large_airport	Cairo International Airport	30.12190
2775	FACT	large_airport	Cape Town International Airport	-33.96480
3206	HKJK	large_airport	Jomo Kenyatta International Airport	-1.31924
3115	GMMX	medium_airport	Menara Airport	31.60690
3113	GMMN	large_airport	Mohammed V International Airport	33.36750

6 rows | 1-6 of 23 columns

```
# sf object
head(mysf)
```

id <dbl>	ident <chr>	type <chr>	name <chr>	elevation_ft <dbl>
31055	FAOR	large_airport	OR Tambo International Airport	5558
3183	HECA	large_airport	Cairo International Airport	382
2775	FACT	large_airport	Cape Town International Airport	151
3206	HKJK	large_airport	Jomo Kenyatta International Airport	5330
3115	GMMX	medium_airport	Menara Airport	1545
3113	GMMN	large_airport	Mohammed V International Airport	656

6 rows | 1-6 of 22 columns

You should see that the columns containing coordinates in the original dataframe are no longer there. In the new `sf` object there is a new column called `geometry` at the end that stores the spatial information. This demonstrates that an `sf` object behaves like a dataframe in many ways.

.xls files

For Microsoft Excel files you just need to change step 1 of the three step approach. You can read an excel file into a dataframe using the package readxl (https://readxl.tidyverse.org/) 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.

C. Directly create an R object

An alternative is directly to create a dataframe within R containing coordinates. This is similar to the approach from the previous section except that dataframe creation replaces file reading at step 1.

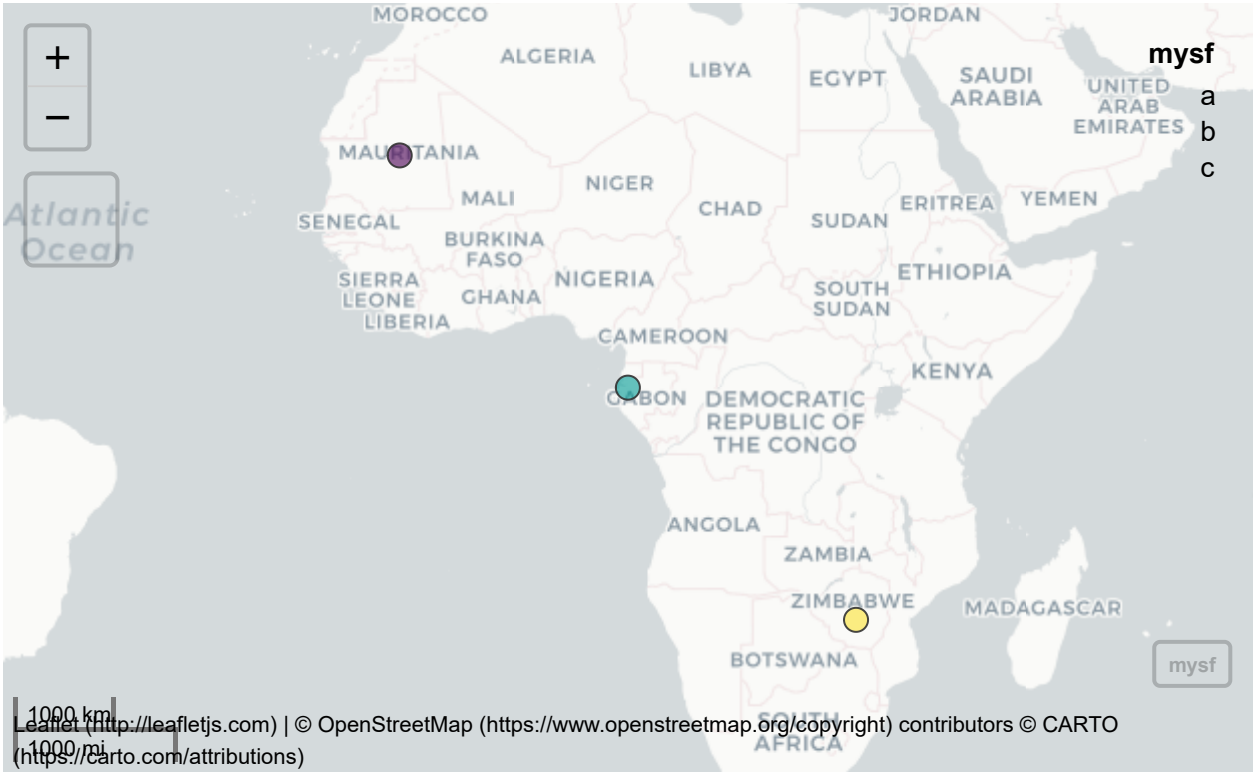
In the example below try changing the coordinates within the dataframe at step 1, and run to see the points change.

R Code | Start Over | Run Code

```
# 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)
```



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`

D. 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 get at the data.

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

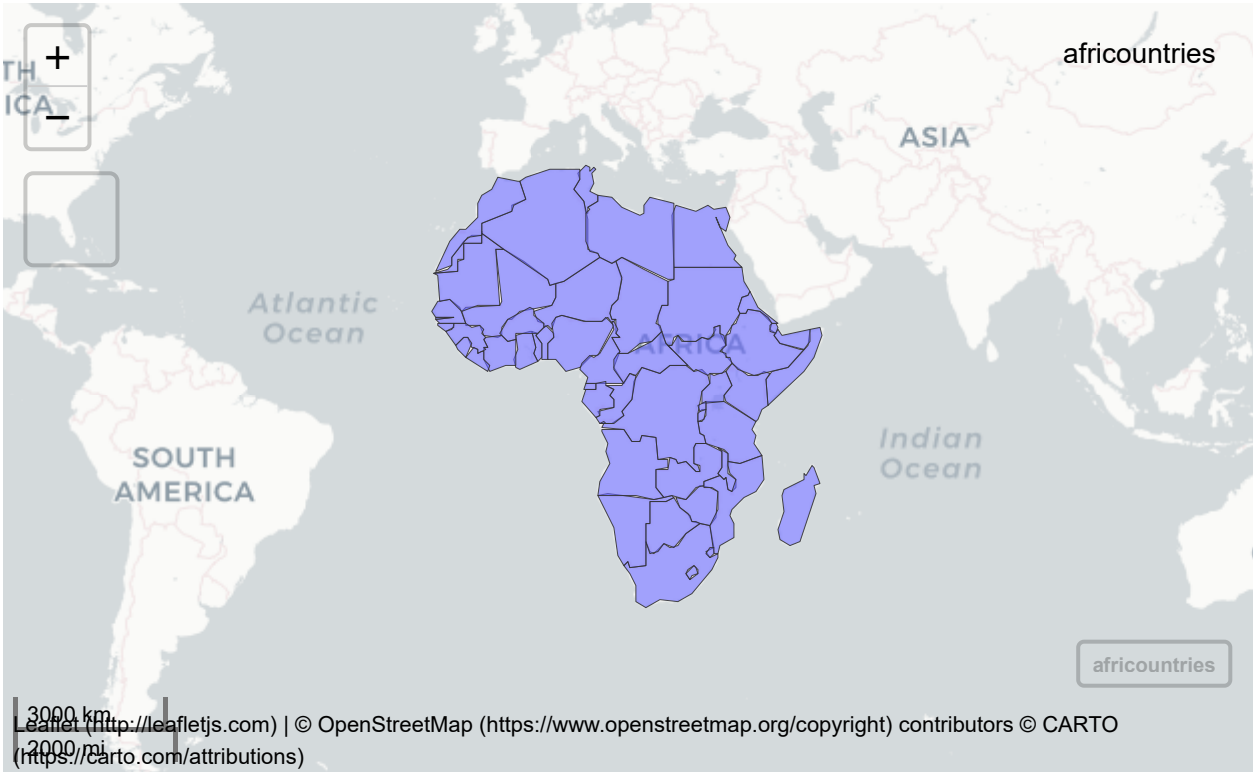
If a colleague emails use just a single file named `*.shp` then you will not be able to map it in R. You would need to ask them to email you all of the files.

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

R Code | Start Over | Run Code

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

# quick interactive plot
mapview(africountries)
```



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 & 2 from the csv example. In addition you don't need to specify in R 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.

E. .kml, .gpkg & .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.

R Code | Start Over | Run Code

```
filename <- system.file("extdata","trans-african-highway.kml", package="afrilearndata", must

afrihighway <- sf::read_sf(filename)

# quick interactive plot
mapview(afrihighway)
```

F. raster tiff

To read in raster data we need to use the package `raster` instead of `sf` . 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.

R Code | Start Over | Run Code

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

myrast <- raster::raster(filename)

# quick interactive plot
mapview(myrast)
```

Note that the map above appears mostly dark. This is the same issue we came across in the intro-to-spatial-r tutorial. 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.

In intro-to-spatial-r we fixed this 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.

G. `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 by passing just a few arguments the map can be made much more informative. Try copy & 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 Code | Start Over | Run Code

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

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

# 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)
```

To find out more about `mapview` options, type just `?mapview` into the window above and press run. This should display the manual page for the `mapview` function, scroll down to where you see **Arguments** in bold and 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

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

Try modifying the airports maps above using some of these options.

Next steps

More information on reading different spatial file formats into R can be found in this section in the excellent Geocomputation in R (https://geocompr.robinlovelace.net/read-write.html#data-input)

Summary

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

We've shown you :

i. how to make a map from a coordinates text file
ii. the importance of CRS (Coordinate Reference System) to place data on a world map
iii. how to read in other spatial data files for vector and raster data
iv. options for making mapview maps more useful

This is a start, there are plenty of other options for making maps in R. 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 order of steps when reading in a text file of coordinates ?

- ◯ 1.read into dataframe 2.convert to sf object & set crs 3.quick interactive plot
- ◯ 1.convert to sf object & set crs 1.read into dataframe 3.quick interactive plot
- ◯ 1.read into dataframe 2.quick interactive plot 3.convert to sf object & set crs
- ◯ 1.quick interactive plot 2.read into dataframe 3.convert to sf object & set crs

Submit Answer

What is the function to convert a dataframe to spatial sf format ?

- ◯ sf::
- ◯ st_as_sf()
- ◯ st_read()
- ◯ read_csv()

Submit Answer

What is the most common crs (coordinate reference system) for longitude, latitude data ?

- ◯ c('longitude', 'latitude')
- ◯ 4326
- ◯ '4326'
- ◯ EPSG

Submit Answer

In a coordinates file what names can be used for the columns containing coordinates ?

- ☐ longitude,latitude
- ☐ x,y
- ☐ long,lat
- ☐ horizontal,vertical
- ☐ salt,pepper

Submit Answer