

Join to spatial data (how to create a map from a spreadsheet that only has region names)

Outline of this tutorial

Use-case:

- You want to plot data on a map but your data only have names of regions or places and no coordinates.
- You also can get some spatial data that does have the coordinates of the regions or places.

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.

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.

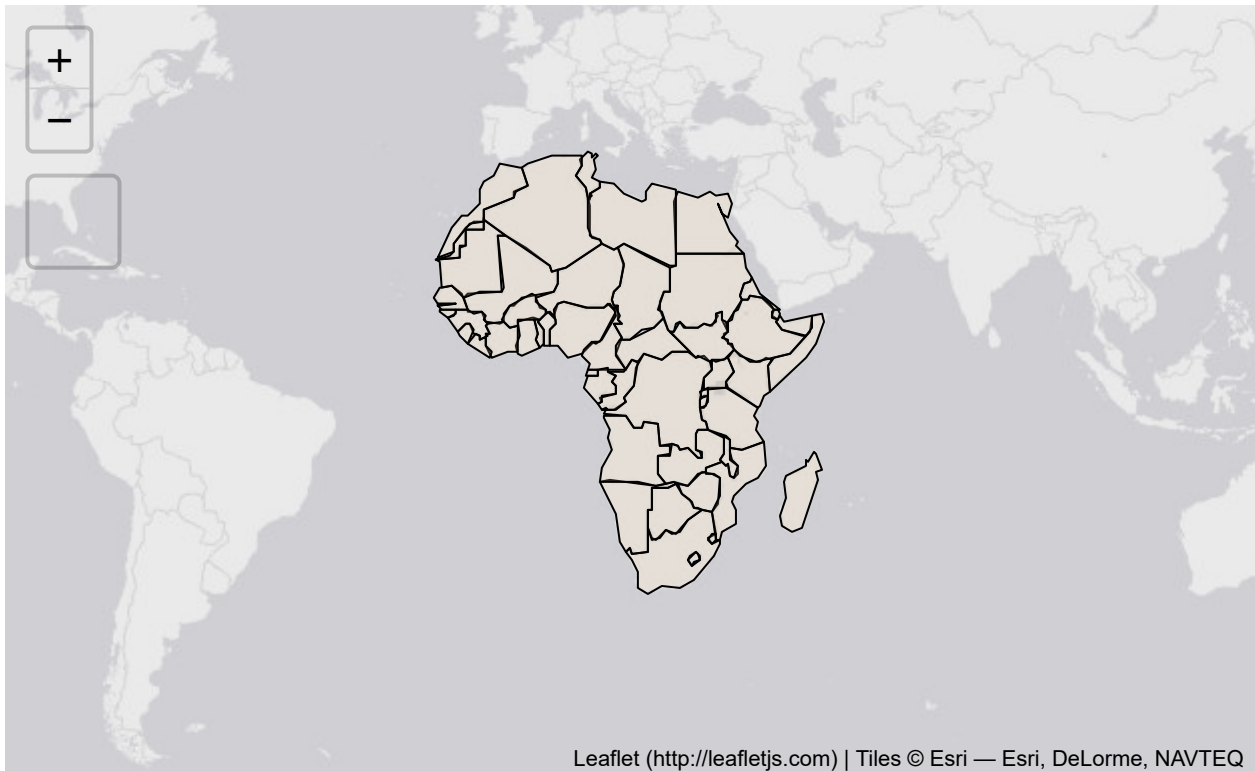
How this tutorial relates to others in afrilearnr

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
join-admin	dealing with data referenced by names rather than coordinates	3 this one
afrilearnr-crash-course	gallery of plots & code with minimal explanation	4

Joining is the technical term for the process of combining a table of data containing names with another containing names and spatial coordinates.

There are often difficulties in joining data to due to differences in the spelling of names, capitalisation, accents etc. afrimapr wants to make joining easier, particularly for users relatively new to these issues.

Here we start to develop a checklist identifying code steps that users can work through to aid the joining process.



- The data you have are in a spreadsheet type format and the spatial data are in some kind of GIS format (e.g. .shp).
- To create the map, you need to join the two datasets in order to create your map. This requires identifying the columns with the admin names, checking for misspellings, and joining them together.

- **Joining your data.frame:**

country	gdpPercap
Algeria	6223.3675
Angola	4797.2313
Benin	1441.2849
Botswana	12569.8518
Burkina Faso	1217.0330

- **To the spatial information:**

name_long	geometry
Algeria	MULTIPOLYGON (((-8.6844 27....
Angola	MULTIPOLYGON (((12.99552 -4...
Benin	MULTIPOLYGON (((2.691702 6....
Botswana	MULTIPOLYGON (((29.43219 -2...

- To create a **spatial data.frame** that combines a `data.frame` with a spatial object

name	name_long	pop_est	gdp_md_est	lastcensus	income_grp	iso_a3	na
Tanzania	Tanzania	53950935	150600.0	2002	5. Low income	TZA	Ta
W. Sahara	Western Sahara	603253	906.5	NA	5. Low income	ESH	Se oc
Dem.	Democratic	83301151	66010.0	1984	5. Low	COD	Cc

STEPS

1. Install and load the relevant libraries

2. Import the data.frame

- Read the data into R as a `data.frame` (`dfdata`).
- Check that `dfdata` is a `data.frame` with `class(dfdata)`.
- Identify the column that contains the admin unit information.

3. Import the spatial data

- Read in the spatial data to R as a `sf` object (`sfshapes`).
- Check that `sfshapes` is an `sf` object with `class(sfshapes)`.
- Plot `sfshapes` to check the locations.
- Identify the column that contains the admin unit information.

4. Joins: use `anti_join` to detect mismatches in admin units

- Use an `anti_join` to detect mismatches in the admin units.
- Mismatches may occur due to different spellings or missing information.

5. Joins: Renaming admin units

- Correct any mismatches due to differences in spellings.

6. Joins: using `left_join` to join the datasets.

- Use a `left_join` to join the `data.frame` to the spatial `data.frame`.

7. Plot the data on a map

1. Set up

Installing and Loading the Relevant Libraries

To repeat this tutorial locally you will need the following packages

```
# working with geographic data
library(sf)

# data wrangling
library(dplyr)

# plotting
library(ggplot2)

# plotting and exploring geographic data
library(mapview)

## non-spatial data
library(gapminder) #gapminder data

# spatial data
library(afrilearndata)

# view the data interactively
library(tmap)
```

2. Importing the data.frame

We will be using data from the `gapminder` package (<https://github.com/jennybc/gapminder>) by Jenny Bryan. The data comes from the gapminder (<http://www.gapminder.org/data/>) project and includes statistics for countries around the world including life expectancy, population, and GDP per capita.

Preparing the data.frame

The `gapminder` data is automatically read in when the `gapminder` package is loaded.

We can start by inspecting the data using `head()`, this will return the first six rows of the `data.frame`.

R Code Start Over Run Code

```
1 head(gapminder)
2
3
```

country <fct>	continent <fct>	year <int>	lifeExp <dbl>	pop <int>	gdpPercap <dbl>
Afghanistan	Asia	1952	28.801	8425333	779.4453
Afghanistan	Asia	1957	30.332	9240934	820.8530
Afghanistan	Asia	1962	31.997	10267083	853.1007
Afghanistan	Asia	1967	34.020	11537966	836.1971
Afghanistan	Asia	1972	36.088	13079460	739.9811
Afghanistan	Asia	1977	38.438	14880372	786.1134

6 rows

We can prepare the data by filtering out just the data for the continent “Africa” using `filter(continent == "Africa")` . Can you add to the filter to get data just for the year 2007 ?

R Code Start Over Hint Run Code

```
1 africa_gap <- gapminder %>%
2   filter(continent == "Africa")
3
```

Can you remember how to inspect the first 6 rows of the new data.frame ? Experiment in the window below.

R Code Start Over Hint Run Code

```
1
2
3
```

We can look at the range of values and the type of data using `summary()` .

R Code Start Over Run Code

```
1 summary(africa_gap)
2
3
```

- ? What is the mean `gdpPercap` ?

We can check the names of the columns (our variables) using `names()` . This is particularly helpful to check the spelling!

R Code Start Over Run Code


```
1 names(africa_gap)
2
3
```

Quiz

Using the output from the commands `head()`, `summary()`, and `names()` above. Which column contains the country name?

- ☐ continent
- ☐ year
- ☐ country
- ☐ gdpPercap

Submit Answer

-  Not all countries are included in the gapminder dataset. Can you find your country there by replacing your country name below ?

R Code [Start Over](#) ▶ Run Code

```
1 africa_gap %>%  
2   filter(country == "Rwanda")  
3
```

Having trouble finding it? Maybe it is spelled differently in the dataset. We can search for it by looking at all the **distinct** countries in the dataset using the following:

R Code [Start Over](#) ▶ Run Code

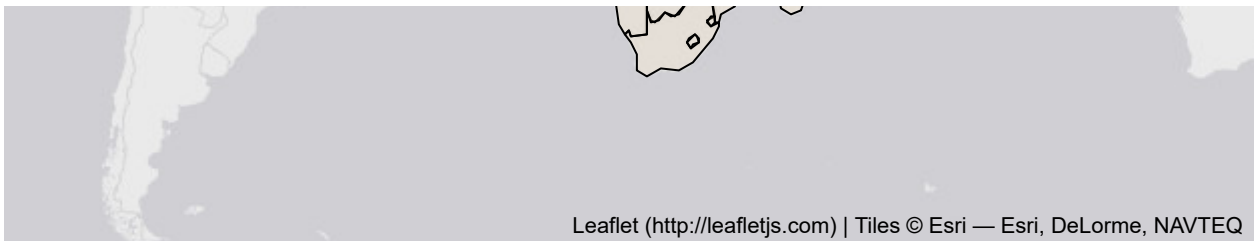
```
1 africa_gap %>%  
2   distinct(country)  
3
```



3. Importing the spatial data

Preparing the spatial data



The spatial data we will use is contained in the `afrilearndata` package. In this tutorial we'll be working with the country-level spatial data `africountries`.





- Click the layers symbol  on the map to view the other datasets contained in `afrilearnrdata`.
-  Can you guess which layers represent point, polygon, line, and raster data?



Read in the Africa country data `africountries.shp` from the `afrilearnrdata` package. This data is subset from the `rnaturalearth` package.

R Code  Start Over  Run Code

```
1 filename <- system.file("extdata","africountries.shp", package="afrilearnrdata", mustWork=TRUE)
2 africountries <- sf::read_sf(filename)
3
```



Visualising the spatial data

You can run a quick check to see where the data is using `plot`.

R Code  Start Over  Run Code

```
1 plot(africountries)
2
3
```

We can also explore a single variable on its own. Let's take a look at `pop_est`.



R Code  Start Over  Run Code

```
1 plot(africountries["pop_est"])
2
3
```


-  Which country has the highest population? What colour is it?

Simple features (sf) spatial data.frame

Try exploring the data, change the code to check the **head** of the data instead of the tail.

R Code  Start Over  Run Code

```
1 tail(africountries)
2
3
```

-  What is the name of the first country in the dataset?

Doublecheck that the class is `sf` using the function `class`.

R Code Start Over Run Code

```
1 class(africountries)
2
3
```

```
[1] "sf"          "tbl_df"      "tbl"         "data.frame"
```

Check the names of the columns using `names()`

R Code Start Over Run Code

```
1 names(africountries)
2
3
```

```
[1] "name"      "name_long"  "pop_est"    "gdp_md_est" "lastcensus"
[6] "income_grp" "iso_a3"     "name_fr"    "name_pt"     "name_af"
[11] "name_sw"    "geometry"
```

Exercises

Quiz

In which column do you find the spatial information?

- ☐ country
- ☐ iso_a3
- ☐ geometry
- ☐ continent

Submit Answer

What type of geometry is the spatial file?

- ☐ point
- ☐ line
- ☐ polygon
- ☐ multipolygon

Submit Answer

4. Joins: using anti-join to detect mismatches

Joins: a toy example

Let's first learn about joins using two toy datasets containing information about `national_animals` and `football_mascots`.

- 🌍 `national_animals` contains a range of countries from around the world and their national animals.

country	animal
Democratic Republic of the Congo	Okapi
Gabon	Black panther
Indonesia	Komodo dragon
Scotland	Unicorn
Uganda	Grey-crowned crane

- Can you spot which countries' national animals are not real animals 🐉?
- `football_mascots` contains a subset of national football team mascots 🐾 from African countries.

country	mascot
Democratic Republic of the Congo	The Leopards
Gabon	The Panthers
Uganda	The Cranes
Sudan	The Nile Crocodiles
Botswana	The Zebras

Aim: We wish to join the `football_mascots` dataset with our `national_animals` dataset to make a single dataset to compare the national animal to the mascot.

- This type of join is called a **mutating join** which allows us to combine variables from two tables R for Data Science (<https://r4ds.had.co.nz/relational-data.html>).
- First observations are matched based on common information (a **key**), then variables are copied from one table to the other.
- The first step is to identify the **key** (common information) that can be used to **link** the data together.

Quiz

Look at the `national_animals` and `football_mascots` datasets above.

Quiz

What variable occurs in each dataset?

- ☐ mascot
- ☐ animal
- ☐ country

Submit Answer

Notes on Joins

- Joining columns do not have to have the same name
- Rows do not have to be in the same order
- Data columns can contain repeated values
- Join columns (country) must not contain repeated values (only 1 mascot allowed)!
- Values (i.e.) that are identical will be joined (watch out for misspelling and capitalisation!).

To join the two datasets we can use a `left_join`, this will join the `national_animals` dataset to the `football_mascots` dataset.

```
R Code Start Over Run Code  
1 football_mascot_vs_animal <- left_join(x = football_mascots, y = national_animals, by = "country")  
2  
3 football_mascot_vs_animal
```

All values that are identical will be joined, i.e. countries that matched.

We can check which rows did not match using an `anti_join`

```
R Code Start Over Run Code  
1 non_matches_mascot_vs_animal <- anti_join(x = football_mascots, y = national_animals, by = "country")  
2  
3 non_matches_mascot_vs_animal
```

We can see all the data using a `full_join`

```
R Code Start Over Run Code  
1 non_matches_mascot_vs_animal <- full_join(x = football_mascots, y = national_animals, by = "country")  
2  
3 non_matches_mascot_vs_animal
```

Types of Joins

- **left_join():** All rows from the left side of the join even if there are no matching rows on the right side. You only get rows from the right side where there's a join match to a row on the left.
- **right_join():** All rows from the left side of the join only where there's a match on the right. You get all rows from the right side of the join even if there are no matching rows on the left.
- **full_join():** All rows from the left and right hand side, joined where the criteria matches.
- **inner_join():** All rows where there's a match on the join criteria are returned. Unmatched rows are excluded.
- **anti_join():** All rows from the right side of the join where there's no match on the left.

Using an anti_join to detect mismatches

You will now have identified that the country names are in the `country` column in `africa_gap` and the `name_long` column in `africountries`.

We can check how many countries are in each dataset using `distinct` and `count`.

R Code Start Over Run Code

```
1 africa_gap %>%
2   select(country) %>%
3   distinct(country) %>%
4   count()
```

- ? How many unique countries are there in the dataset?

R Code Start Over Run Code

```
1 africountries %>%
2   select(name_long) %>%
3   distinct(name_long) %>%
4   count()
```

- ? How many unique countries are there in the spatial `data.frame` ?

We can see that the `data.frame` and the spatial `data.frame` have a different number of countries!

Before we join the datasets, we can use an `anti_join` to find out what countries are missing, and which ones do not match!

R Code Start Over Run Code

```

1 mismatch_df_spatial <- dplyr::anti_join(x = africa_gap,
2                                         y = africountries,
3                                         by = c("country" = "name_long")
4                                         )
5
6 mismatch_df_spatial %>%
7   select(country)

```

R Code Start Over Run Code

```

1 mismatch_spatial_df <- dplyr::anti_join(x = africountries,
2                                         y = africa_gap,
3                                         by = c("name_long" = "country")
4                                         )
5
6 mismatch_spatial_df %>%
7   select(name_long)

```

From this we can see that there are a couple of things happening.

1. **Spelling** Some countries are spelled differently in the two datasets:

- *Congo, Dem. Rep.* in `africa_gap` is *Democratic Republic of the Congo* in `africountries`
- *Cote d'Ivoire* in `africa_gap` is *Côte d'Ivoire* in `africountries`
- *Gambia* in `africa_gap` is *The Gambia* in `africountries`
- *Congo, Rep.* in `africa_gap` is *Republic of Congo* in `africountries`

2. **Countries missing from spatial data.frame** There are several countries in `africa_gap` that are not in `africountries` !

- Islands: Comoros, Mauritius, Reunion, Sao Tome and Principe.

3. **Countries missing from data.frame** There are several countries in `africountries` that are not present in `africa_gap` .

- Western Sahara
- Somaliland
- South Sudan

5. Joins: Renaming placenames

At this stage we have a couple of decisions to make based on our discoveries from the `anti_join` .

1. **Some countries are spelled differently in the two datasets:**

africa_gap	africountries	final join
Congo, Dem. Rep.	Democratic Republic of the Congo	Democratic Republic of the Congo

africa_gap	africountries	final join
Cote d'Ivoire	Côte d'Ivoire	Côte d'Ivoire
Gambia	The Gambia	The Gambia
Congo, Rep.	Republic of Congo	Congo

- Before changing the names so that they match, we want to decide what we want the name to be in the final join. This is not always an easy task, names vary in different languages and can also be political.
 - For this example, we will use the short name of countries from the Inter-institutional Country Style Guide (https://publications.europa.eu/code/en/en-5000500.htm#fn-gm*) from the European Union.
 - In the table above, we've noted the changes we need to make in order to match the country names in the `africa_gap` and `africountries` datasets and to bring them in line with the Inter-institutional Country Style Guide.

Which names should be changed?

- The names need to match between the two datasets in order for the join to work.
- As to what you choose, that will depend on what names you ultimately want to include on the map.
- You may wish to follow a style guide, for instance, in this example we use the Interinstitutional Style Guide (https://publications.europa.eu/code/en/en-5000500.htm#fn-gm*) put together by the European Union.
- Depending on the names you wish to have, you can change either dataset or both!

Changing the names

- We can change the country names so that they match using `recode` with `mutate`.
 - `recode` takes the form "old name" = "new name".

R Code
Start Over
Run Code

```

1 africa_gap <- africa_gap %>%
2     mutate(country = recode(country,
3                             "Congo, Dem. Rep." = "Democratic Republic of the C
4                             "Congo, Rep." = "Congo",
5                             "Cote d'Ivoire" = "Côte d'Ivoire"))

```

- Note that we are then saving the changes we make back into the original data.frame using the assignment arrow `africa_gap`.

Fill in the blank to recode the country *Republic of Congo* in `africountries` to *Congo*.

R Code [Start Over](#) [Run Code](#)

```
1 africountries <- africountries %>%
2   mutate(name_long = recode(name_long,
3     "Republic of Congo" = "_____"))
```

R Code [Start Over](#) [Run Code](#)

```
1 africountries <- africountries %>%
2   mutate(name_long = recode(name_long,
3     "Republic of Congo" = "Congo"))
```

Let's run our `anti_join` again to see what we still need to change.

R Code [Start Over](#) [Run Code](#)

```
1 mismatch_df_spatial <- dplyr::anti_join(x = africa_gap,
2   y = africountries,
3   by = c("country" = "name_long")
4 )
5
6 mismatch_df_spatial %>%
7   select(country)
```

R Code [Start Over](#) [Run Code](#)

```
1 mismatch_spatial_df <- dplyr::anti_join(x = africountries,
2   y = africa_gap,
3   by = c("name_long" = "country")
4 )
5
6 mismatch_spatial_df %>%
7   select(name_long)
```

6. Joins: using left-join to join the datasets

Joining the datasets

Now that we've done the prep work so that the countries match, we can now join the two datasets together.

- Because we are interested in plotting the data, we will join the `africa_gap` data to the spatial data.frame `sfcountries` using a `left_join()`.
- Note that the join takes the same form as the `anti_join`. Because we are joining the `africa_gap` data to the `africountries` dataset.

R Code [Start Over](#) [Run Code](#)

```
1 africa_df <- dplyr::left_join(x = africountries,
2   y = africa_gap,
3   by = c("name_long" = "country")
4 )
```

Let's check the results of our `left_join`.

Does it have the correct number of rows?

R Code [Start Over](#) [Run Code](#)

```
1 africa_df %>%
2   head()
3
```

- How many countries are there?

R Code [Start Over](#) [Run Code](#)

```
1 africa_df %>%
2   distinct(name_long) %>%
3   count()
```

Let's make a couple of changes to the dataset before plotting.

- Let's rename the column `name_long` to `country` using `rename`.

R Code [Start Over](#) [Run Code](#)

```
1 africa_df <- africa_df %>%
2   rename(country = name_long)
3
```

7. Plot the data on a map

- If you are familiar with `ggplot`, `geom_sf` is just another `geom` you can specify where the geometry is a map.
- In order to use the `geom_sf` layer, the data will have to be of class `sf`. We can check this using `class(africa_df)`.

R Code [Start Over](#) [Run Code](#)

```
1 class(africa_df)
2
3
```

- When you run the call above, you'll see that the dataset we are using contains many classes, one of which is `sf`.
- You can specify which variable to use in the `aes` thetics for the fill. In this case we can use the estimated population `pop_est`.

R Code [Start Over](#) [Run Code](#)

```
1 ggplot(data = africa_df) +
2   geom_sf(aes(fill = pop_est))
3
```

- Take a look at the other variables in the dataset. Create a new plot by choosing a different variable to fill the polygons.

```
R Code Start Over Run Code
1 head(africa_df)
2
3
```

name <chr>	country <chr>	pop_est <chr>	gdp_md_... <dbl>	lastce
Tanzania	Tanzania	53950935	150600.0	
W. Sahara	Western Sahara	603253	906.5	
Dem. Rep. Congo	Democratic Republic of the Congo	83301151	66010.0	
Somalia	Somalia	7531386	4719.0	
Kenya	Kenya	47615739	152700.0	
Sudan	Sudan	37345935	176300.0	

6 rows | 1-5 of 17 columns

```
R Code Start Over Run Code
1 ggplot(data = africa_df) +
2   geom_sf(aes(fill = pop_est))
3
```

Let's now refine our plot with labels, a new colour scheme, and axis labels.

Changing the colour palette

- We can change the fill using several options from the `scale_fill_` layer try typing `scale_fill_` + TAB to see the different options

```
R Code Start Over Run Code
1 scale_fill_
2
3
```

Let's try out a new fill scheme using `viridis`. The `viridis` scales provide colour maps that are perceptually uniform in both colour and black-and-white. The scale `viridis_c` is optimised for color vision deficiencies.


```
R Code Start Over Run Code
1 ggplot(data = africa_df) +
2   geom_sf(aes(fill = pop_est)) +
3   scale_fill_viridis_c()
```

Adding country labels

We can add labels for the countries using `geom_sf_label`.

R Code


 Start Over


 Run Code

```
1 ggplot(data = africa_df) +  
2   geom_sf(aes(fill = pop_est)) +  
3   scale_fill_viridis_c() +  
4   geom_text_repel(aes(label = name_long, geometry = geometry), stat = "sf_coordinates")
```

Axis labels

R Code

 Start Over

 Run Code

```
1 ggplot(data = africa_df) +  
2   geom_sf(aes(fill = pop_est)) +  
3   scale_fill_viridis_c() +  
4   geom_text_repel(aes(label = name_long, geometry = geometry), stat = "sf_coordinates") +  
5   labs(x = "Longitude", y = "Latitude", title = "Population Estimate 2000", fill = "Popula")
```