Handout – Working with Geo-Data

NaWi-Workshop: Obtaining, linking and plotting geographic data

Markus Konrad [markus.konrad@wzb.eu](mailto:markus.konrad@wzb.eu)

May 6, 2019

Table of Contents

# Plotting with *ggplot2*

TODO

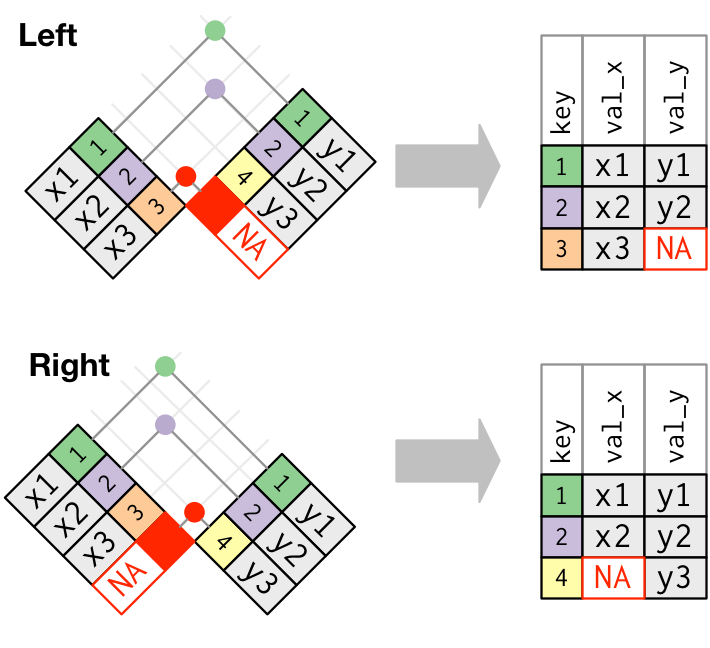
<https://r4ds.had.co.nz/data-visualisation.html>

# Data linkage with *dplyr*

## Left and right (outer) joins

*Left and right outer joins* keep all observations on the left-hand or right-hand side data sets respectively. Unmatched rows are filled up with *NA*s:

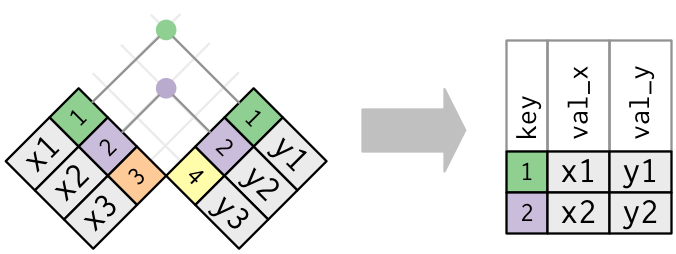
Syntax: inner\_join(a, b, by = <criterion>)



Left and right join. Source: Grolemund, Wickham 2017: R for Data Science

## Inner joins

An *inner join* matches keys that appear in both data sets and returns the combined observations:



Inner join. Source: Grolemund, Wickham 2017: R for Data Science

Syntax: inner\_join(a, b, by = <criterion>)

## Specifying matching criteria

Parameter by can be:

1. a character string specifying the key for both sides, e.g.: inner\_join(pm, city\_coords, by = 'city') will match city column in pm with city column in city\_coords;
2. a vector of character strings specifying several keys to match both sides, e.g.: inner\_join(pm, city\_coords, by = c('city', 'country') will match those rows, where city *and* country columns match;
3. a *named* character string vector like inner\_join(pm, city\_coords, by = c('cityname' = 'id'), which will match the column cityname in pm with the column id in city\_coords

# Specific hints / further information for excercises

## Exercise 2

### Finding out geo-coordinates

We will later learn how to use the Google Maps API to geocode (i.e. get the geo-coordinates) places programmatically. For the purpose of this excerise, it’s enough to do it manually.

There are several websites that offer free manual geocoding, e.g.:

* <https://google-developers.appspot.com/maps/documentation/utils/geocoder/>
* <https://www.mapdevelopers.com/geocode_tool.php>

Both work the same way: You enter a request (i.e. an address, city name, restaurant name, etc.) and it spits out the result, including the longitude and latitude. **Please be aware that the first service returns the geo-coordinate with latitude first, followed by longitude (“Location: …”).**

### Constructing a dataset quickly from within R

You can construct the small dataset directly within R, by passing place labels, longitude and latitudes as separate column vectors:

places <- data.frame(  
 label = c('born', 'living', 'neven been there'),  
 lng = c( 12.590, 13.402, 8.0456),  
 lat = c( 51.279, 52.520, 52.276)  
)

### Loading the worldmap dataset

The following loads the world map dataset from the maps package as *Simple Features* spatial dataset:

library(maps)  
library(sf)  
  
worldmap\_data <- st\_as\_sf(map('world', plot = FALSE, fill = TRUE))

### Filtering the worldmap dataset

You can filter the worldmap data from the maps package by using the “ID” column:

sweden <- worldmap\_data[worldmap\_data$ID == 'Sweden',]

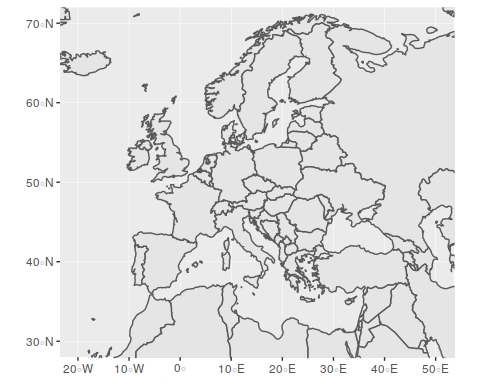
Using the %in% operator when selecting several countries:

scandinavia <- worldmap\_data[worldmap\_data$ID %in% c('Sweden', 'Denmark', 'Finland', 'Norway', 'Iceland'),]

### Restricting the display window

You can specify a “display window” (i.e. “zooming in” to a certain region) by setting a limit on the displayed longitude range (xlim) and latitude range (ylim) in the coord\_sf() function:

ggplot() + geom\_sf(data = worldmap\_data) + coord\_sf(xlim = c(-20, 50), ylim = c(30, 70))



We will learn more options on how to specify display windows in the second part of the workshop.

## Exercise 3

## Exercise 4

When loading the bln\_plr\_sozind\_data.csv dataset, make sure that the variable SCHLUESSEL is loaded as character string, **not** as integer (use colClasses = c('SCHLUESSEL' = 'character') in read.csv()).

After loading the spatial dataset bln\_plr.geojson make sure to set the CRS: st\_crs(<DATASET>) <- 25833.

More information on the bln\_plr\_sozind\_data.csv dataset:

* source: Berlin Senate Dept. for Urban Dev. and Housing, *Monitoring Soziale Stadtentwicklung 2017* via [FIS-Broker](https://fbinter.stadt-berlin.de/fb/index.jsp)
* variables:
* STATUS1: Unemployment rate 2016 in percent
* STATUS2: Long term unemployment rate 2016 in percent
* STATUS3: Pct. of households that obtain social support (“Hartz IV”) 2016
* STATUS4: Portion of children under 15 living in household that obtains social support (“Hartz IV”) 2016
* DYNAMO1 to 4: Change in the above indicators from the previous year

## Exercise 5

After loading the spatial dataset nutsrg\_2\_2016\_epsg3857\_20M.json make sure to set the CRS: st\_crs(<DATASET>) <- 3857.

More information on the tgs00010\_unempl\_nuts2.csv dataset:

* source: [Eurostats / Regions & cities](https://ec.europa.eu/eurostat/web/regions/data/main-tables)
* variables:
* sex: F means unemployment rate for women, M for men, T for both
* nuts: NUTS level-2 region code
* year: year when the data was collected
* unempl\_pct: unemployment rate in percent

In case you want to use a different Eurostats dataset or a different NUTS map, you can download these resources here:

* for the datasets: <https://ec.europa.eu/eurostat/data/browse-statistics-by-theme>
* for the NUTS maps: <https://github.com/eurostat/Nuts2json>

# Sources for geo-data

## R packages

The following packages come directly with geo-data or provide means to download them programmatically:

* [maps](https://cran.r-project.org/web/packages/maps/index.html): World, USA, US states, US counties and more
* [mapdata](https://cran.r-project.org/web/packages/mapdata/index.html): World in higher resolution, China, Japan and more
* [rnaturalearth](https://cran.r-project.org/web/packages/rnaturalearth/index.html): *R package to hold and facilitate interaction with natural earth vector map data.* → see next slides
* [OpenStreetMap](https://cran.r-project.org/web/packages/OpenStreetMap/): Access to the OpenStreetMap API → see next slides

## Natural Earth Data

[naturalearthdata.com](http://www.naturalearthdata.com/): *Natural Earth is a* ***public domain map dataset*** *available at 1:10m, 1:50m, and 1:110 million scales. Featuring tightly integrated vector and raster data, with Natural Earth you can make a variety of visually pleasing, well-crafted maps with cartography or GIS software.*

Provides vector data for:

* countries and provinces, departments, states, etc.
* populated places (capitals, major cities and towns)
* physical features such as lakes, rivers, etc.

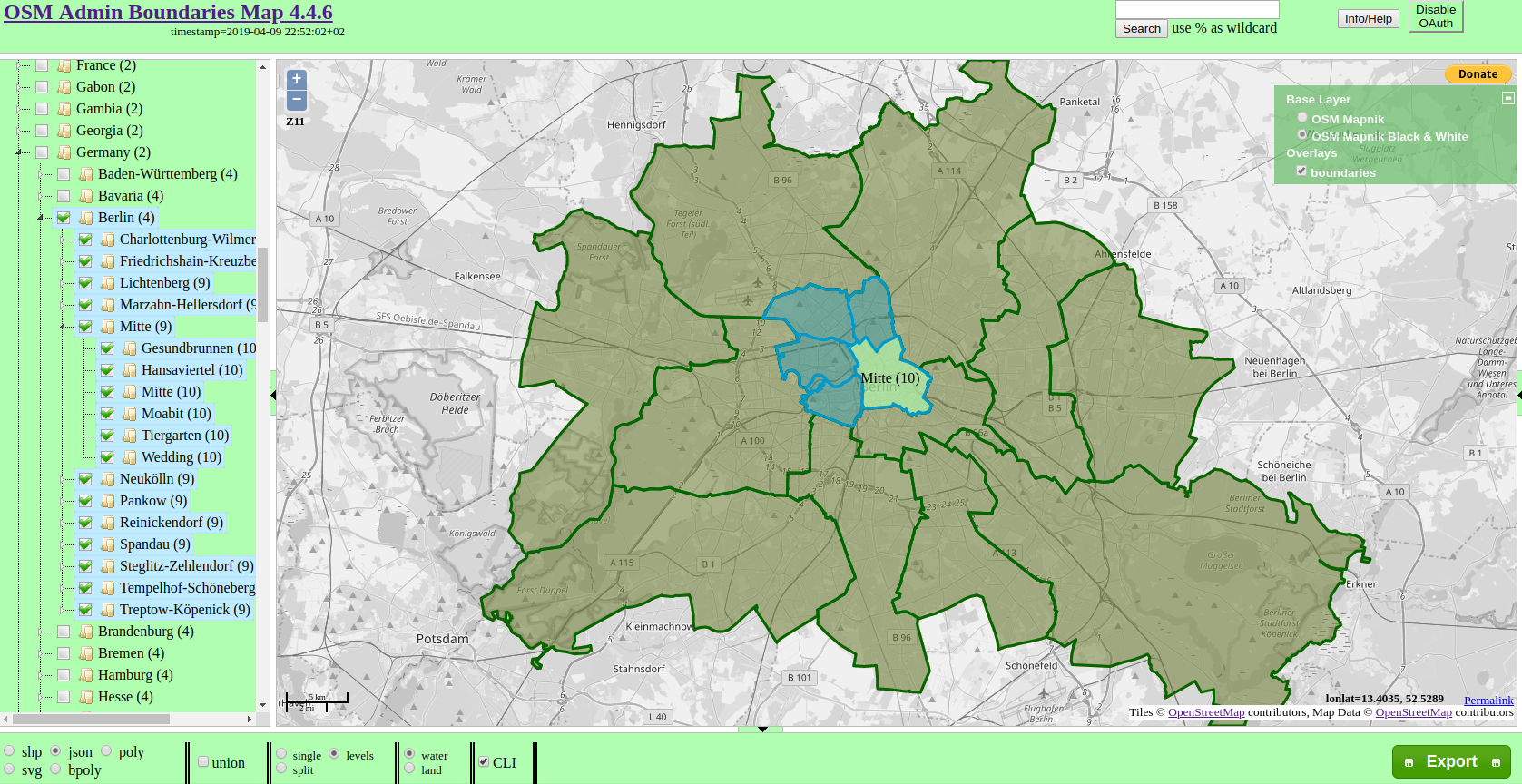
You can either download the data directly from the website or use the package [rnaturalearth](https://cran.r-project.org/web/packages/rnaturalearth/index.html).

## Open Street Map

* provides even more detail than *Natural Earth Data*: streets, pathways, bus stops, metro lines, etc.
* [GeoFabrik](https://download.geofabrik.de/) provides downloads of the raw data
* is much harder to work with b/c of the complexity of the data

[OSM Admin Boundaries Map](https://wambachers-osm.website/boundaries/): web-service to download administrative boundaries worldwide for different levels in different formats (shapefile, GeoJSON, etc.); contains meta-data (depending on country) such as AGS in Germany

[This wiki article](https://wiki.openstreetmap.org/wiki/DE:Grenze#Gemeinden_.E2.80.93_admin_level.3D7.E2.80.938) explains which OpenStreetMap administrative boundary levels correspond to which regional level in Germany (e.g. level 6 corresponds to “Kreise”).



OSM Admin Boundaries screenshot

## Administrative authorities in the EU

Administrative authorities often provide geo-data. In the EU, the main source is [Eurostat](https://ec.europa.eu/eurostat) which provides data referenced by NUTS code.

* [main NUTS datasets as SHP, GeoJSON, TopoJSON, SVG](https://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units/nuts)
* [Nuts2json](https://github.com/eurostat/Nuts2json) provides another overview for GeoJSON and TopoJSON datasets
* [correspondence tables](https://ec.europa.eu/eurostat/web/nuts/correspondence-tables/postcodes-and-nuts) map national structures and postcodes to NUTS regions

## Administrative authorities in Germany

*Statistisches Bundesamt* provides [geo-referenced data](https://www.destatis.de/DE/Themen/Laender-Regionen/Regionales/_inhalt.html), such as:

* [Gemeindeverzeichnis](https://www.destatis.de/DE/Themen/Laender-Regionen/Regionales/Gemeindeverzeichnis/_inhalt.html): AGS, area, population, etc.
* [Regionaldatenbank](https://www.regionalstatistik.de/genesis/online): GDP, building land value, etc.
* [govdata.de](https://www.govdata.de/): Open data portal for Germany – lots of data, but not very well curated and documented

Berlin:

* [Senate Department for Urban Development and Housing](http://www.stadtentwicklung.berlin.de/planen/basisdaten_stadtentwicklung/index.shtml) for example provides datasets based on [LOR units](http://www.stadtentwicklung.berlin.de/planen/basisdaten_stadtentwicklung/lor/index.shtml)
* [FIS Broker](https://fbinter.stadt-berlin.de/fb/index.jsp) is a web-service providing all publicly available geo-referenced data – [this post](https://lab.technologiestiftung-berlin.de/projects/fisbroker-to-qgis/index.html) shows how to use it

## What about historical data?

Geographic areas such as administrative borders change. Identifiers may change, too. Make sure to use the version that matches your dataset!

* *Eurostat* provides historical NUTS areas back to 2003
* *Statistisches Bundesamt* also provides an archive

# Glossary

**AGS:** *Amtlicher Gemeindeschlüssel* – municipality identificator in Germany.

**CRS:** Coordinate reference system – defines the coordinate system (spherical, ellipsoid, cartesian, etc.), unit of measurment (degrees, meters, etc.) and map projection of points in a spatial dataset in order to locate geographical entities

**CRAN:** *Comprehensive R Archive Network* – repository of packages that extend the statistical software suite R.

**EPSG:** *European Petroleum Survey Group* – a scientific organization tied to European petroleum industry. Created the *EPSG Geodetic Parameter Set*, which among other things contains a database of →CRS identified by EPSG →SRID code

**ETRS89:** *European Terrestrial Reference System 1989* – EU-recommended frame of reference for geodata for Europe; defines a →CRS.

**GIS:** *Geographic information system* – a system such as a software like →QGIS designed to work with geographic data.

**Lat / Latitude:** Geographic coordinate that defines the north-south position of a point on Earth as an angle between -90° (south pole) and 90° (north pole). The equator is located at 0° latitude.

**Lon / Long / Lng / Longitude:** Geographic coordinate that defines the east-west position of a point on Earth as an angle between -180° (westward) and 180° (eastward). The Prime Meridian is located at 0° longitude.

**LOR:** *Lebensweltlich orientierte Räume* – structures the city area of Berlin into sub-regions at three different levels; each area is identified by a LOR code.

**NUTS:** *Nomenclature of Territorial Units for Statistics* – divides the EU territory into regions at 3 different levels for socio-economic analyses of the regions; each area is identified by a NUTS code.

**QGIS:** free and open-source →GIS application.

**SRID:** *Spatial Reference System Identifier* – identifies a →CRS by a unique code number which is listed in the →EPSG database. Because of this, it is often also called EPSG code or number. Examples: EPSG:4326 refers to →WGS84; EPSG:4258 refers to →ETRS89.

**SRS:** *Spatial Reference System* – see →CRS.

**WGS84:** *World Geodetic System 1989* – defines a →CRS at global scale. Coordinates are defined in degrees as →longitude and →latitude.