Creating NUTS-2 Choropletes from Eurostat Data

Albrecht Gradmann

March 5th, 2015

Introduction

This document shows how to create a choroplete map from Eurostat Data using R. For this example I aimed to automate the entire process from downloading the data to the creation of the figure. The idea is to present an entirely reproducible process. The text shows the overall process from a birds-eye perspective. In particular it is not intended to serve as a general introdution to R[1] or a discussion of the Pros and cons of using choropletes to display data.

[1] One of many good introductory sites for R. `http://www.statmethods.net/`

This tutorial shows the process of creating plot in R that visualizes the EU 2013 unemployment rates in the EU as percentage of total population. The final graphic will be a so-called choroplete map. This is a map where geographical regions are coloured by some kind of metric. For this kind of figure to work well the individual regions should be roughly about the same size. For a more in-depth discussion you may refer to Leland Wilkinsons Grammar of Graphics [2] which we have available at Ecologic. The theoretical foundations presented in this book lay the foundations for the `ggplot2`package by Hadley Wickham [3], the approach used in this document

[2] Wilkinson: The Grammar of Graphics. Springer-Verlag New York, 2005.

[3] Hadley Wickhams `ggplot2` is a very versatile and popular graphics package for R. `http://ggplot2.org/`

I plan to create other handouts in the future. Please let me know if you would like to request a specific topic.

Setting the scene

This tutorial gives a brief walk-throug of the steps needed to create a choroplete map of EU 2013 unemployment rates by NUTS-2 regions as a percentage of total population. Unemployment data as well as the geospatial data will be taken from Eurostat. Apart from unemployment data Eurostat provides several datasets on NUTS2 or even NUTS3 level. Adapting the present example to other datasets requires only minor modifications.

The example relies on a series of libraries which have to be installed on your system. Package installation in R is a simple and standard task and it can be done in several ways. For more information, you can either type `help("install.packages")` in the R Console or look for one of the numerous tutorials on the internet[4]. Below is the series of commands you need to run first to attach the packages to your R session.

[4] How to install packages in R `http://tinyurl.com/nh7whtp`

If you want to reproduce the results present in this document, you can copy-paste the code snippets into an R session. If you prefer the original source files, please send me an email and I will provide them happily.

```
library(maptools) # Dealing with spatial data
library(ggplot2)  # Hadley Wickham plotting package
library(ggmap)    # Mapping with ggplot
library(RJSDMX)   # Query Eurostat REST-interface
library(grid)     # Needed for unit() function
library(rgdal)    # Provides driver to read .shp files
```

Getting the unemployment data

Eurostat provides several interfaces for access to data ranging from the online data-browser over the 'bulk-download-facility' to programmatic interfaces using `REST` or `SOAP`[5]. While each of these interfaces has its own merits, a programmatic access to the `REST` interface is shown below. A programmatic access ensures that the creation of the figure is indeed reproducible. Provided access to the script used the process of creating the figure is unambiguously defined. In addition this approach allows the re-use and modularization of individual steps at a later stage.

[5] `REST` and `SOAP` the names of internet protocols. You don't have to bother with the details.

We use the `getTimeSeries()` function to download the dataset from Eurostat. R provides some facilities to browse the Eurostat database. The details of these facilities and an explanation how to define the arguments will be topic of another handout. The code below basically tells R to got to the Eurostat website, look for a dataset called `tgs00010` and from this to select annual data (that's the `A`), measured as percentage (the `PC` part) for both sexes (`T` stands for total) and for all available geographic entities (∗ means 'take all'). The `start` and `stop` arguments define the timespan for which we want to get the data.

```
# Get the data formatted as list from Eurostat
tsList <- getTimeSeries('EUROSTAT',
                        'tgs00010.A.PC.T.*',
                        start = "2013",
                        end="2013")
```

The next step is to convert the downloaded data into a format suitable for further processing. Without going into too much detail, a `data-frame` is basically the way most R methods read data. It is basically a collection of variables of the same length.

```
# Convert the list into a dataframe
tsDf <- sdmxdf(tsList,meta = T)
```

The table below shows the resulting data which is now stored in the `data-frame`.

	TIME	OBS	FREQ	UNIT	SEX	GEO
1	2013	4.00	A	PC	T	AT11
2	2013	4.50	A	PC	T	AT12
3	2013	8.40	A	PC	T	AT13
4	2013	5.30	A	PC	T	AT21
5	2013	4.00	A	PC	T	AT22
6	2013	4.00	A	PC	T	AT31

Table 1: First rows of tsDf

If you ever bowsed for data on the Eurostat website, you will probably recognize some of the variables:

- TIME designates the year of the observation.
- OBS is the actual numeric observaion value.
- FREQ tells us the observation frequency. Our data is annual (A) based on the data retrieval call.
- UNIT hast been set to percentage (PC) in the data retrieval call.
- SEX is the total (T) for both sexes.
- GEO lists the geographic entities at NUTS-2 level.

Possible values for SEX are Male, Female and Total (not transgender as you might think).

Further inspection of the dataset reveals that figures for Turkey are included in the dataset. We use the following command to filter out all rows that start with TR. While in this case a relatively simple search string can be used, you should know that R is fully capable of interpreting regular expressions (REGEX) allowing the definition of more sophisticated filter criteria. [6] the A last step we need to take for the

[6] While intimidating at first regular expressions are a very powerful tool for any data work. `http://www.regular-expressions.info/rlanguage.html`

```
# Subset to exclude Turkey
tsDfsub <-  subset(tsDf,
                   !grepl(c("TR"),GEO))
```

Now as we have gathered and prepared the unemployment data from Eurostat, our next step will be the acquisition of spatial data.

Getting the empty maps

For the creation of choroplete maps we need information about the geographical extent of the NUTS-2 regions. This information is provided from Eurostat in the form of ESRI-shapefiles [7]. The appropriate shapefiles will be downloaded programmatically. As they are provided as zip-files we need to unpack them for further processing.

[7] Administrative boundary shape files at Eurostat `http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units`

```
# Get the shapefile from Eurostat
download.file(paste0("http://ec.europa.eu/eurostat/cache/",
                     "GISCO/geodatafiles/NUTS_2010_60M_SH.zip"),
              destfile = "/NUTS_2010_60M_SH.zip")

# Unzip the data in a temporary location
unzip("/NUTS_2010_60M_SH.zip",
      exdir="NUTS_2010_60M_SH")
```

Once the archive has been extracted, we can create the information into a SpatialPolygonsDataframe which is basically a processed shapefile ready for further treatment in R. We also set the map projection to WGS84. While a discussion of projections [8] is fay beyond the scope of this paper it is important to understand that this is a crucial step in any mapwork, especially if you aim to merge spatial data from different sources. Not knowing the projection of the data will then most likely lead to desaster.

[8] An overview of map projections. http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj_f.html

```
# Read administrative boundaries
# and set appropriate Transformation
eurMap <- readOGR(paste0("NUTS_2010_60M_SH/NUTS_2010_60M_SH/",
                         "NUTS_2010_60M_SH/data"),
                  "NUTS_RG_60M_2010")

# Set appropriate projection
eurMap <- spTransform(eurMap,
                      CRS("+proj=longlat +datum=WGS84"))
```

As a last step we need to convert the SpatialPolygonsDataframe into a regular dataframe. This requirement comes from our usage of the ggplot package for doing the acutal plotting routine, which expects dataframes as inputs.

```
# Convert into a dataframe suitable for plotting
eurMapDf <- fortify(eurMap, region="NUTS_ID")
```

Table 2: First rows of tsDf

	long	lat	order	hole	piece	group	id
1	15.95	48.79	1	FALSE	1	AT.1	AT
2	16.04	48.77	2	FALSE	1	AT.1	AT
3	16.05	48.76	3	FALSE	1	AT.1	AT
4	16.07	48.76	4	FALSE	1	AT.1	AT
5	16.10	48.75	5	FALSE	1	AT.1	AT
6	16.10	48.75	6	FALSE	1	AT.1	AT

The resulting dataframe contains 7 Variables defining and naming the polygons for plotting:

- long longitudinal coordinate of polygon edge.
- lat latitudinal coordinate of polygon edge.
- order order in which polygon edges are connected.
- hole defines the type of polygon.
- piece defines part of polygon.
- group groups edges into polygon
- id lists the geographic entities at NUTS-2 level.

The first six variables contain the geometric definitions of the polygons. You will only probably never have to deal with them in any serious manner. The only exception is the variable `order` of which you have to make sure that it is sorted from smallest to largest before plotting. This step will be carried out in the upcoming section.

The last variable `id` however is crucial. It contains the names of all polygons contained in the shape file by their NUTS name. As you can see from the first lines displayed in the table they are not limited to NUTS-2 but include also all other levels of NUTS. We will use this variable to link the unemployment data to the geometry definition. This process is called merging and is explained in the nest section.

Merge Unemployment Data with Map Data

Merging the unemployment and geographical data is achieved by telling R the respective datasets and name the columns to merge on. Note that this process of join two data frames by one or more common key variables is what is called an 'inner join' in `SQL`. R also provides functions to carry out more complex joins.

```
# merge map and data
tsMapDf <- merge(eurMapDf, tsDfsub,
                 by.x="id", by.y="GEO")
```

As noted before, the records of the merged dataframe have to be sorted in the order in which they will be drawn to the graphics device.

```
# put data in correct order for plotting
tsMapDf <- tsMapDf[order(tsMapDf$order),]
```

With the merge of the two datasets we have concluded all steps in the process of gathering data and preparing it for plotting.

Plotting the data

The remainder of this document will deal with the process of a final plot from an initial draft. The steps shown provide examples of steps commonly carried out when producing a figure. It goes without saying that they are to a great extent matter of subjective taste. Maps are a special kind of display in which the general visualization is less strictly defined by the underlying data than e.g. a bar graph. While all of the following steps are in some way influenced by an underlying theory of perception, creating maps remains to a large extent an art.

Creating the first draft

Creating a first draft of the plot does not take much code. The syntax used is based on Leland Wilkinsons Grammar of Graphics as mentioned in the introduction. This may be a puzzling concept at first. In the end however it proves to be a tool that enables the user to concisely describe the components of a graphic in a logical manner.

```
# inverse order (to have visible borders)
map <- ggplot(data = tsMapDf, aes(x = long, y = lat,
    group = group))
map <- map + geom_polygon(aes(fill = OBS))
map
```

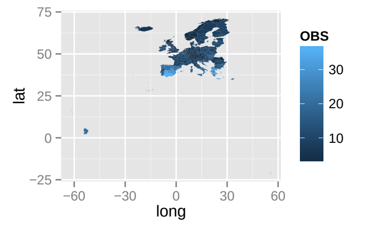


Figure 1: An initial version of our map

Although this first draft is far from perfect it is already suitable to check if the graphic makes sense and actually reveals any interesting information. An initial inspeciont reveals that unemployment rates are highest in southern Spain and Greece. If this is a point we would like to make, we can further tune the graphic. While this is an initial version still, it can make sense to include it as a placeholder in a report draft. R can output the graphics in many different file formats [9] that play well with different text editing and layout software. In Word you can link the figure data file so that it updates once a new version has been created in R.[10]

[9] Output options from R `http://www.cookbook-r.com/Graphs/Output_to_a_file/`

[10] How to link figures in Word. `http://tinyurl.com/ccumxd`

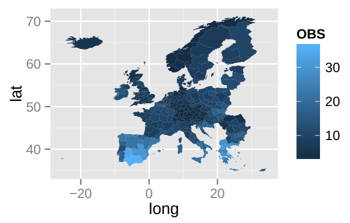
Putting Europe into focus

Our initial map contains regions which are far away from mainland Europe. These are mostly french DOM regions. To make the plot more readable we limit the graphic to mainland Europe.

```
#limit data to main Europe
europe.limits <- geocode(c("Cape Fligely, Rudolf Island,
                           Franz Josef Land, Russia",
                                   "Gavdos, Greece", "Faja Grande,Azores",
                                   "Severny Island, Novaya Zemlya, Russia"))

# apply the limits to our dataset
tsMapDf <- subset(tsMapDf,
                       long > min(europe.limits$lon) &
                       long < max(europe.limits$lon) &
                       lat > min(europe.limits$lat) &
                       lat < max(europe.limits$lat))

# and re-read the plot with the new data
map <- map %+% tsMapDf
map
```

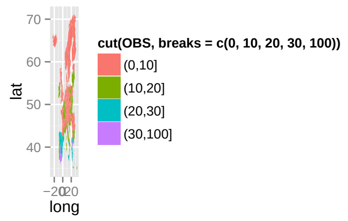


If DOM or similar regions are of specific interest, they could be added as inset graphics at a later stage.

Bin data into classes

R treats the unemployment figures as a continuous variable. To get a better feeling of the actual level of unemployment in the respective regions we can group the regions into different classes. This process is calles *binning* and is essentially a transformation into a discrete variable.

```
map <- ggplot(data=tsMapDf,
              aes(x=long,
                  y=lat,
                  group=group))
map <- map + geom_polygon(aes(fill=cut(OBS,
                                       breaks=c(0,10,20,30,100))))
map
```



If we re-plot the map with binned data we get an awkward looking result; especially when printed in the margin. By default the software assigns automatic values for the legend labels resulting in a crushed plot.

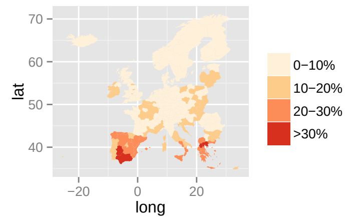
Fix color and legend

The default color palette is optimized for qualitative variables which have no intrinsic ordering. For our case this is not appropriate as the bins are ordered. For the selection of the appropriate color scale we turn to Cynthia Brewers website [11] which provides advice for choosing appropriate colors for mapping. The service suggests the

[11] Color advice for maps. `http://colorbrewer2.org/`

palette 'OrRd' as being photocopy safe and appropriate for four classes of sequential data.

```
map <- map + scale_fill_brewer(palette = "OrRd",
                               name="",
                               labels=c("0-10%",
                                        "10-20%",
                                        "20-30%",
                                        ">30%"),
                               guide="legend")
map
```



Re-coloring the classified data we can identify areas of high unemployment as red areas which actually makes sense as these are the areas that need attention.

The labelling text of the legend has also been adapted in this step.

Add country borders

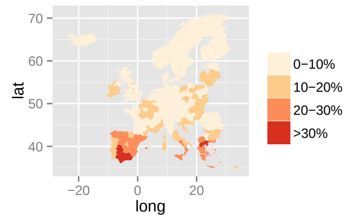
Here we add the borders of regions an countries in different colors.

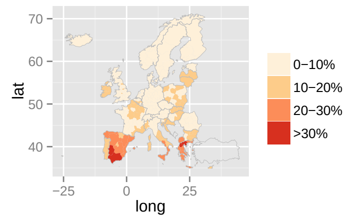
```
# Borders on NUTS2 level in white
#map <- map + geom_path(color="white",size=0.01)
map


# Borders on NUTS1 level in grey
rindex <- grep(eurMapDf$id,pattern=c("^[[:alpha:]]{2}$"))
eurMapDf_NUTS1 <- eurMapDf[rindex,]

eurMapDf_NUTS1 <- subset(eurMapDf_NUTS1,
                       long > min(europe.limits$lon) &
                         long < max(europe.limits$lon) &
                         lat > min(europe.limits$lat) &
                         lat < max(europe.limits$lat))

map <- map + geom_path(data=eurMapDf_NUTS1, color='grey', size=0.0
map
```



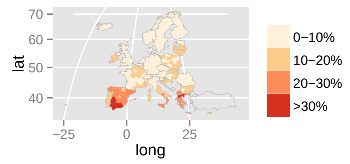


Change projection

Maps can be displayed in different projections and they reveal a lot about the creators personality.[12]

[12] xkcd explains what map projections tell about you `http://xkcd.com/977/`

```
map <- map + coord_map(projection = "mollweide")
map
```



Some window dressing

These final steps address the overall appearance of the figure. While there is a lot of fine tuning going on, it is possible to save this information to a *theme* which can be applied to all figures in a given text. This allows to streamline graphics within a publication or to adhere to standards of corporate design.

```
map <- map + theme_bw()
map <- map + theme(
    plot.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    axis.ticks = element_blank(),
    axis.text = element_blank(),
    axis.title = element_blank(),
    legend.key.size = unit(10,"pt"),
    text=element_text(size=8),
    legend.position=c(0.1, 0.2))
```

The Result

Below you will find the so-far final version of the plot. There are still several imperfections like a missing scale and a text box with source references. Especially troublesome is the eye-catching hole in the adriatic coast. While this could be due to rapid sink of the Kars Mountains followed by horrendous floods it is more likely an issue with the shape file addressing EU countries only.[13] However all these issues are solvable and given that this is a tutorial and not the preparation of publication-ready graphics I will refrain from dealing with them here.

[13] Analysis of the availability and the quality of data on western balkans and Turkey - 2011 http://tinyurl.com/ngrhbpd

Concluding remarks and Credits

One of the great advantages of R over other similar software packages is a very active and helpful user community. This document has been prepared by drawing on the following sources:

A cheatsheet for working with maps and ggplot [14]. This is a great concise resource that discusses how to include maps from google, open street maps or stamen.

[14] https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/ggmap/ggmapCheatsheet.pdf

Stackoverflow[15] is a question and answer site for professional and enthusiast programmers. It's built and run by you as part of the Stack Exchange network of Q&A sites. Given that you post a well-defined question you will most likely get help here within hours.

[15] Stack-Overflow - a community based help site. http://stackoverflow.com/

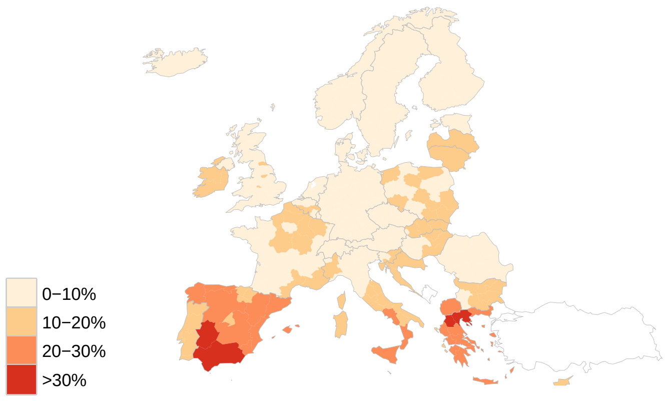


Figure 2: EU 2013 Unemployment rates by NUTS2 regions, percentage of total population, both sexes Source: Eurostat (tgs00010)

A post on Max Marchis Blog[16] was the source of the initial draft of this document. However this is just one of many. Searcing for maps, R, choropletes and Eurostat will certainly reveal numerous other sources.

[16] Max Marchis Blog `http://www.milanor.net/blog/?p=594`

The `ggplot2`[17] package has a great online documentation with many practical examples.

[17] Online Documentation of Hadley Wickhams `ggplot2` package. `http://docs.ggplot2.org/current/`

Last but not least this document has been created to get used to the Tufte-LaTeX [18] document classes which define a style similar to the style Edward Tufte uses in his books and handouts. Tufte's style is known for its extensive use of sidenotes, tight integration of graphics with text, and well-set typography.

[18] `https://code.google.com/p/tufte-latex/`